GRADA

PRŮVODCE

Jiří Kosek

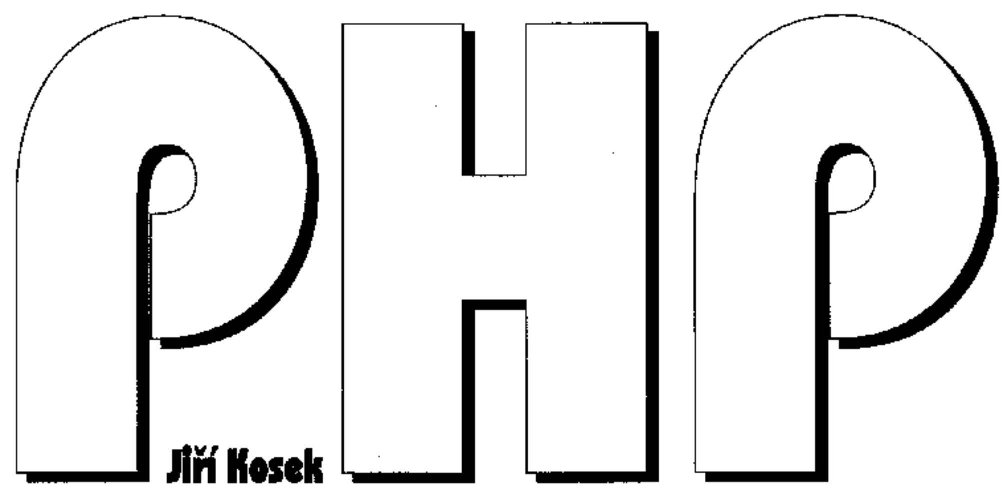
PHP

TVORBA INTERAKTIVNÍCH INTERNETOVÝCH APLIKACÍ

Podrobný průvodce

- srozumitelný popis skriptovacího jazyka PHP3
- tvorba a obsluha webových formulářů
- snadné propojení stránek s databázemi, úvod do jazyka SQL
- podrobný popis více než 600 funkcí

GRADA



TVORBA INTERAKTIVNÍCH INTERNETOVÝCH APLIKACÍ

Podrobný průvodce

GRADA Publishing, 1999

Jiří Kosek

PHP – tvorba interaktivních internetových aplikací

© Grada Publishing, spol. s r.o., 1999

Sazba knihy byla připravena v typografickém systému TeX z písma Computer Modern
ve variantě CS-font.

V knize použité názvy programových produktů, firem apod. mohou být ochrannými známkami
nebo registrovanými ochrannými známkami příslušných vlastníků.

ISBN 80-7169-373-1

Stručný obsah

Předmluva ... 11

Typografické konvence ... 14

1. Úvod ... 15

2. Rychlé seznámení ... 25

3. Jazyk PHP ... 37

4. Ladění skriptů a ošetření chyb ... 87

5. Formuláře ... 93

6. Spolupráce s databázemi ... 121

7. Praktické ukázky ... 179

8. Bezpečné aplikace ... 243

9. Referenční přehled funkcí ... 253

10. Protokol HTTP ... 433

11. Instalace a konfigurace ... 459

12. Zdroje informací na Internetu ... 473

Literatura ... 476

Rejstřík ... 478

Tematický přehled funkcí ... 489

Podrobný obsah

Předmluva ... **11**

Typografické konvence ... **14**

1. Úvod ... **15**

1.1 Celosvětová epidemie WWW ... 15

1.2 Jak přišlo PHP na svět ... 20

1.3 Kde můžeme získat PHP ... 21

1.4 Co budeme potřebovat pro tvorbu aplikací v PHP ... 22

2. Rychlé seznámení ... **25**

2.1 Náš první skript ... 25

2.2 Snadná práce s formuláři ... 32

2.3 Spolupráce s databázemi ... 33

3. Jazyk PHP ... **37**

3.1 Vkládání PHP do HTML ... 37

3.2 Základy syntaxe ... 38

3.3 Proměnné ... 40
- Typ integer ... 41
- Typ double ... 41
- Typ string ... 41

3.4 Pole ... 43
- Inicializace pole ... 43
- Funkce pro práci s polem ... 44
- Vícerozměrná pole ... 45

3.5 Přetypování proměnných ... 45
- Zjištění typu proměnné ... 46
- Změna typu proměnné ... 46
- Přetypování ... 46
- Konverze řetězců na čísla ... 46

3.6 Výrazy ... 47
- Matematické operátory ... 48
- Operátory pro manipulaci s bity čísla ... 49
- Logické výrazy a operátory ... 51

Operátor pro spojování řetězců ... 52
Podmíněný operátor ... 53
Priorita operátorů ... 53

3.7 Příkazy pro větvení programu ... 54
Příkaz if ... 54
Příkaz if–else ... 56
Příkaz if–elseif–else ... 57
Příkaz switch ... 58

3.8 Příkazy cyklu ... 60
Příkaz while ... 60
Cyklus do–while ... 61
Cyklus for ... 61
Příkaz break ... 63
Příkaz continue ... 64

3.9 Příkazy pro načítání skriptů ... 64
Příkaz require ... 64
Příkaz include ... 66

3.10 Definice vlastních funkcí ... 66
Rozsah platnosti proměnných ... 68
Předávání parametrů ... 69
Standardní hodnoty parametrů ... 70
Statické proměnné ... 71
Rekurze ... 72

3.11 Vlastnosti jazyka, které se jinam nevešly ... 72
Ukončení běhu skriptu ... 72
Eval – magická hůlka ... 73
Konstanty ... 73
Proměnná chameleon ... 74

3.12 Objekty a PHP ... 75
Členské proměnné ... 76
Členské funkce ... 77
Konstruktory ... 79
Dědičnost ... 80

3.13 Regulární výrazy ... 82

4. Ladění skriptů a ošetření chyb ... 87

4.1 Syntaktické chyby ... 87

4.2 Logické chyby ... 89

4.3 Ošetření chyb 91

4.4 Interní debugger 92

5. Formuláře 93

5.1 Byrokratické základy 93

5.2 Základní prvky formulářů 94
Vstupní pole pro krátký text 94
Tlačítko pro odeslání formuláře 95
Heslo 96
Zaškrtávací pole 97
Přepínací tlačítka 98
Tlačítko pro vynulování formuláře 99
Tlačítko s obrázkem 99
Odeslání souboru 101
Skrytá pole 102
Seznamy 102
Víceřádkový text 103

5.3 Profesionální formuláře 104

5.4 Rozšíření formulářů z dílny HTML 4.0 114
Nové druhy tlačítek 115
Popisky vstupních polí, horké klávesy a tabulátor 115
Sdružování vstupních polí do bloků 117
Sdružování položek seznamu do skupin 119
Zakázané ovoce 120

5.5 Pár závěrečných poznámek k formulářům 120

6. Spolupráce s databázemi 121

6.1 Co je to databáze 121

6.2 Jak komunikují SŘBD se zbytkem světa 127

6.3 Výběr a instalace SQL-serveru 130
MySQL 131
PostgreSQL 133
MS SQL Server 134

6.4 Lehký úvod do jazyka SQL 135
Vytvoření tabulky 137
Přidání nového záznamu do tabulky 139
Výběr a prohlížení záznamů 140
Mazání záznamů 143
Modifikace záznamu 144

Nastavení přístupových práv ... 145

6.5 Podpora databází v PHP ... 147
Podpora MySQL ... 147
Podpora ODBC ... 150
Podpora PostgreSQL ... 152

6.6 Manipulace s daty pomocí prohlížeče ... 154

6.7 Transakční zpracování ... 172

6.8 Práce s binárními daty ... 174
Vkládání dat ... 174
Čtení dat ... 175

6.9 Persistentní spojení s databázovým serverem ... 177

7. Praktické ukázky ... 179

7.1 Počítadla přístupů ... 179
Textové počítadlo vložené přímo do stránky ... 180
Počítadlo jako obrázek ... 184
Objektové a databázové – zkrátka superpočítadlo ... 188

7.2 Kniha hostů ... 197

7.3 Uživatelé Internetu mají smysl pro humor ... 203
Atraktivní domovská stránka ... 206
Automatické zasílání vtipů mailem ... 207

7.4 PHP jako brána k dalším službám ... 217
Vyhledávání e-mailových adres ... 217
Jednoduchý POP klient ... 221
Odesílání MIME-dopisů ... 227

7.5 Udělejme si vlastní Seznam ... 229

7.6 On-line demokracie aneb hlasování ... 237

8. Bezpečné aplikace ... 243

8.1 Ochrana skriptů před nepovolanými zraky ... 243
Útok zvenku pomocí prohlížeče ... 244
Pod svícnem je tma ... 244

8.2 Autentifikace uživatelů ... 246
Nechme za sebe pracovat jiné ... 246
Podpora autentifikace v PHP ... 246

8.3 Šifrování přenášených dat ... 248

8.4 Bezpečnější než sex ... 249

PHP jako modul ... 249
PHP jako CGI-skript ... 249

8.5 Bezpečný režim ... 251

9. Referenční přehled funkcí ... 253

10. Protokol HTTP ... 433

10.1 Hypertext Transfer Protocol ... 433
Jak to funguje ... 434
Formát požadavku ... 435
Formát odpovědi ... 436
Hlavičky ... 438
Práce s hlavičkami v PHP ... 442

10.2 Cookies ... 443
Uložení cookies ... 445
Čtení cookies ... 447
Jednoduchá ukázka použití cookies ... 447

10.3 Proč potřebujeme knihovnu PHPLIB? ... 448

11. Instalace a konfigurace ... 459

11.1 Instalace ve Windows ... 459
PWS a IIS3 ... 460
IIS4 ... 460
Apache 1.3.x ... 461

11.2 Instalace na Unixu ... 461

11.3 Konfigurace ... 463
Obecné konfigurační direktivy ... 464
Konfigurace elektronické pošty ... 468
Konfigurace bezpečného režimu ... 468
Konfigurace debuggeru ... 468
Konfigurace dynamicky zaváděných modulů ... 469
Konfigurace zvýrazňování syntaxe ... 469
Konfigurace ODBC ... 470
Konfigurace MySQL ... 471
Konfigurace PostgreSQL ... 471

11.4 Parametry příkazové řádky ... 472

12. Zdroje informací na Internetu ... 473

12.1 Kde získat PHP? ... 473

12.2 Podpora uživatelů ... 473

12.3 Knihovny hotových skriptů ... 474

12.4 Webové servery ... 474

12.5 Databázové servery ... 475

Literatura ... 476

Rejstřík ... 478

Tematický přehled funkcí ... 489

Funkce pracující pouze na serveru Apache ... 489
Funkce pro práci s adresáři ... 489
Funkce pro práci s COM-objekty ... 489
Funkce pro práci s databází dbm ... 489
Funkce pro práci s databází MySQL ... 490
Funkce pro práci s databází PostgreSQL ... 490
Funkce pro práci s datem a časem ... 491
Funkce pro práci s datovými zdroji ODBC ... 491
Funkce pro práci s daty různých kalendářů ... 492
Funkce pro práci s elektronickou poštou ... 492
Funkce pro práci s obrázky ... 492
Funkce pro práci s poli ... 493
Funkce pro práci s protokolem HTTP ... 493
Funkce pro práci s regulárními výrazy ... 493
Funkce pro práci s textovými řetězci ... 494
Funkce pro práci s URL adresami ... 494
Funkce pro práci se soubory ... 495
Funkce pro práci se soubory dBase ... 496
Funkce pro práci s protokolem SNMP ... 496
Funkce pro přesné aritmetické operace ... 496
Funkce pro přístup k adresářovým službám ... 496
Funkce pro spouštění externích programů ... 497
Konfigurace a informace o PHP ... 497
Konstanta ... 497
Matematická funkce ... 498
Ostatní funkce ... 498
Podpora protokolu IMAP ... 499
Proměnná ... 500
Síťové funkce ... 500

Předmluva

Vážení čtenáři,

kniha, kterou jste právě otevřeli, mne stála půl roku života. Původně měla mít pouze třista stran, ale záhy se ukázalo, že je to málo. Možnosti PHP jsou opravdu nečekané a byla by velká škoda vás o některé z nich připravit. Originální dokumentace není tak podrobná, jak byste si možná přáli, a navíc je v angličtině. Rozhodl jsem se proto zpřístupnit možnosti PHP všem. Výsledkem je, že se na necelých pětistech stranách dozvíte vše, co potřebujete znát pro vývoj webových aplikací v systému PHP.

Začněme však popořádku. Cílem předmluvy je stručně seznámit laskavého čtenáře s obsahem knihy.

V úvodu se seznámíte s vývojem technologií, které se používají pro tvorbu webových aplikací — a to jak na straně klienta (v prohlížeči), tak na straně serveru.

Druhá kapitola vám na několika málo stránkách stručně, ale jasně ukáže, k čemu se dá PHP využít. Pokud tedy nevíte, jestli vám může být PHP k něčemu dobré, přečtěte si ji — stihnete to ještě předtím, než začnete být v knihkupectví nápadní.

Třetí kapitola vychází z toho, co se čtenář dozvěděl v druhé kapitole. Jsou zde podrobně popsány všechny rysy a vlastnosti jazyka PHP, bez jejichž dobré znalosti nelze psát efektivní aplikace.

Následující kapitola se věnuje hledání a odstraňování chyb ve vašich aplikacích. Léty prověřená programátorská poučka říká, že v každém programu je alespoň jedna chyba. Ze začátku budou chyby i ve vašich aplikacích, a proto je dobré vědět, jak jim předcházet a jak je ošetřit.

Komunikace webových aplikací s uživatelem je obvykle založena na formulářích vložených do HTML stránky. V páté kapitole se proto podrobně seznámíme s možnostmi, které HTML nabízí pro tvorbu formulářů. Výklad je úzce provázán se způsoby, jak získané údaje zpracovávat pomocí PHP. Stranou nezůstane ani problematika správného rozdělení obsluhy formulářů mezi server (PHP) a klienta (JavaScript).

Převážná většina dnes provozovaných aplikací jsou různé (především podnikové) informační systémy. Jejich základem je vždy databáze, do které se ukládají všechny potřebné (a někdy i nepotřebné) informace. Problematice spolupráce PHP s databázemi se proto věnujeme velice důkladně v šesté kapitole. Součástí kapitoly je i stručný úvod do jazyka SQL, který je dnes standardem pro komunikaci s databázovými servery.

Po prostudování šesté kapitoly budete mít již tolik znalostí, že se můžete pustit do vývoje komplexních webových aplikací. Jako zdroj inspirace vám poslouží sedmá kapitola, která obsahuje ukázky řešení různých problémů pomocí PHP.

Jestliže mluvíme o komerčním využití Internetu, dostává se automaticky do popředí otázka bezpečnosti. Ta je mnohdy velmi podceňována, a proto jsme ji věnovali celou osmou kapitolu. Podrobně se zde seznámíte se způsoby, jak psát skripty, které neohrozí bezpečnost vašeho systému.

PHP by nebylo tím, čím je (v současnosti asi nejlepším skriptovacím prostředím pro tvorbu dynamicky generovaných stránek), kdyby neobsahovalo rozsáhlou knihovnu funkcí. V deváté kapitole naleznete podrobný popis všech funkcí, včetně jejich parametrů, popisu jejich použití a samozřejmě praktických ukázek využití.

Desátá kapitola se věnuje protokolu HTTP a jeho rozšíření o cookies. Profesionál by měl znát vše, a tedy i protokol, který ke své komunikaci používají prohlížeče a webové servery — zvláště, pokud pomocí protokolu HTTP můžete aplikacím přidat zajímavé schopnosti.

Každý software je potřeba před jeho použitím nainstalovat a nakonfigurovat. A o tom je právě jedenáctá kapitola. Je zařazena až na konci, protože se často stane, že PHP je už na vašem systému nainstalováno a vy se o instalaci nemusíte starat.

PHP je na Internetu jako doma a není divu, že zde nalezneme mnoho zajímavých informačních zdrojů, které s PHP souvisí. Jejich přehled naleznete ve dvanácté kapitole. Kniha je uzavřena přehledem použité literatury a velice podrobným rejstříkem.

Co byste měli znát, než se pustíte do studia knihy? Nezbytným předpokladem je dobrá znalost jazyka HTML. Pro jeho nastudování vám mohu doporučit knihu *HTML – tvorba dokonalých WWW stránek* [13]. Pokud máte nějaké zkušenosti s programováním, určitě se vám budou hodit. Pokud ne, nevadí — s trochou přirozené inteligence vše pochopíte.

V celé knize naleznete mnoho ukázkových aplikací a skriptů. Pokud si je chcete vyzkoušet, nemusíte je opisovat. Na adrese `http://www.kosek.cz/php/` naleznete zdrojové texty všech ukázek. Kromě toho se zde pokusím zveřejňovat různé další ukázky využití PHP. Určitě zde naleznete materiál, který se bude podrobně věnovat práci s XML-dokumenty — na tuto zajímavou problematiku nám v omezeném rozsahu knihy bohužel nezbyl prostor.

Doufám, že vám kniha dobře poslouží. Snažil jsem se ji napsat, jak nejlépe to šlo. V době psaní knihy nebyla originální dokumentace zdaleka kompletní, a tak jsem činnost mnoha funkcí musel luštit přímo ze zdrojových kódů. Tato práce je pomalá a zdlouhavá. Potvrdilo se však, že zdrojové kódy jsou nejlepší dokumentací — nakonec v nich naleznete vše. Doufám, že v této knize s mnohem menším úsilím naleznete to, co potřebujete znát. Pokud se vše povede, bude

kniha na pultech českých knihkupectví v době, kdy se na trhu objeví první knihy o PHP v angličtině.

I přes veškerou snahu se však člověk nevyvaruje chyb. Pokud v knize nějaké chyby naleznete nebo budete mít ke knize nějaké výhrady či připomínky, dejte mi vědět na adresu `jirka@kosek.cz`. Pomůžete tak zkvalitnit další vydání knihy.

Rád bych na závěr poděkoval především všem tvůrcům Open Source Software. Ukazuje se, že software vytvořený na nekomerční bázi, ke kterému jsou k dispozici zdrojové texty, je mnohdy mnohem kvalitnější než drahé komerční produkty. Děkuji na tomto místě všem lidem, kteří se podílejí na vývoji PHP. O tom, že jejich snaha není marná, svědčí více jak 200 000 serverů pracujících právě s PHP.

Můj dík patří i profesoru Knuthovi, který již před dvaceti lety vyvinul dosud v mnoha ohledech nepřekonaný sázecí systém TEX, který mnoho dalších dobrovolníků doplnilo o další podpůrné a užitečné programy.

Dík si zaslouží i nakladatelství Grada, zvláště pak šéfredaktor počítačové redakce Ruda Pecinovský, kteří podpořili vydání této knihy vybočující z řady ostatních, více komerčně a spotřebně laděných titulů.

Pokud se vám kniha bude líbit, může zato i Lenka M. A. Třísková, která knihu pečlivě přečetla a zahrnula mě (nejen;–) řadou cenných připomínek.

Přeji vám příjemné a spadnutím počítače nerušené čtení knihy.

Jirka Kosek

Praha – Podolí, 26. ledna 1999

Typografické konvence

Aby byl text knihy pro čtenáře srozumitelnější, používám několik typografických konvencí, na které jste už zvyklí z mých předchozích knih a z knih od nakladatelství Grada.

- Pro zápis příkazů, funkcí, výpisů stránek a programů používám `neproporcionální písmo`.
- *Kurzívu* používám pro zvýraznění nových pojmů v textu.
- V případě potřeby používám uzavření obecného pojmu do francouzských uvozovek. Tento pojem se pak v praxi vždy nahradí nějakou konkrétní hodnotou. (Např. *«soubor»* se nahradí konkrétním jménem souboru.)

Některé úseky textu jsou označeny piktogramy. Jejich význam je následující:

Takto označený text obsahuje důležitou informaci, jejíž neznalost vám může zkomplikovat život.

Např. *Pokud sáhnete na horká kamna, spálíte si ruku.*

Text obsahuje informaci, jejíž znalost vám může život usnadnit. Většinou zde naleznete různé tipy a triky, jak vylepšit vaše skripty a zefektivnit jejich tvorbu.

Např. *Pokud chcete sahat na horká kamna, pořiďte si azbestovou rukavici.*

Informace uvedené v takto označeném textu jsou zajímavé, ale jejich neznalost negativně neovlivní vaše základní životní funkce.

Např. Oblíbená hudební skupina autora knihy jsou Jethro Tull.

1. Úvod

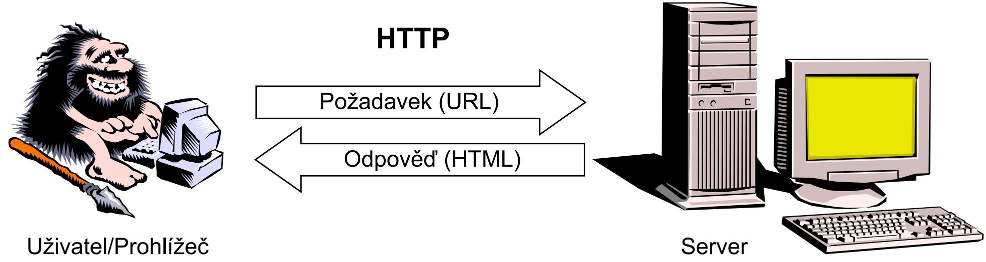
Než se pustíme do výkladu samotného systému PHP, nebude na škodu zopakovat si vývoj, kterým Web prošel od svého zrodu před necelými deseti lety. Uvědomíme si, kolik relativně nových technologií je s Webem spojeno a jak spolu tyto technologie souvisejí.

1.1 Celosvětová epidemie WWW

V roce 1990, kdy byla služba World-Wide Web poprvé spuštěna na půdě výzkumného centra CERN, jsme si vystačili s pouhými třemi technologiemi. První z nich byl jazyk *HTML (HyperText Markup Language)*, který sloužil k zápisu webových stránek. HTML je dodnes ústřední technologií Webu, okolo které se vše točí. Dnes již sice existuje jazyk HTML ve verzi 4.0, nicméně je stále zpětně kompatibilní s původní jednoduchou verzí HTML.

Druhou nezbytnou technologií je přenosový protokol *HTTP (HyperText Transfer Protocol)*, který zajišťuje přenos HTML-stránek z WWW-serveru do prohlížeče. Původní verze HTTP 0.9 byla velmi jednoduchá. V důsledku zvýšených požadavků na možnosti kontroly přenosu dokumentů a samotné zrychlení přenosu postupně vznikly nové verze HTTP 1.0 a 1.1. HTTP 1.1 se dnes stává standardem, který podporují všechny nejvýznamnější WWW-servery a prohlížeče.

Třetí technologií nezbytnou pro implementování služby WWW jsou *URL (Uniform Resource Locator)*. Každý objekt přístupný na Webu má svoji jedinečnou URL-adresu, která slouží k vytváření odkazů na daný objekt.



Obr. 1-1: Průběh komunikace mezi prohlížečem a WWW-serverem

Z dnešního pohledu spojení těchto tří technologií nenabízí mnoho — umožňuje pouze prohlížení elektronických dokumentů, které jsou provázány systémem odkazů. Jak se tedy ubíral vývoj dál k dnešní podobě Webu, který je interaktivní a reaguje na požadavky uživatele?

První inovací byla možnost automatického generování stránek, které obsahují informace proměnlivé v čase. HTML-stránka je soubor uložený na disku WWW-serveru, který má své URL. Nic však nebrání tomu, aby URL ukazovalo na nějaký spustitelný soubor (program), který vygeneruje HTML-stránku. Tato stránka pak může obsahovat aktuální informace. Spustitelný soubor je vyvoláván WWW-serverem a bylo proto zapotřebí rozhraní, které by definovalo způsob spuštění programu a předávání dat mezi WWW-serverem a programem. Rozhraní se jmenuje *CGI (Common Gateway Interface)*. Programům, které generují HTML-stránky, se proto často říká CGI-skripty.

Další vývoj přirozeně směřoval k tomu, aby uživatel mohl ovlivnit chování CGI-skriptu. V HTML 2.0 se tedy objevily elementy, které umožňovaly na stránce definovat formulář. Údaje vyplněné uživatelem do formuláře odeslal prohlížeč serveru a ten je pomocí rozhraní CGI předal CGI-skriptu k dalšímu zpracování. Tímto způsobem funguje na Internetu mnoho služeb dodnes. Různé vyhledávací servery jsou typickým příkladem. Uživatel zadá do vstupního pole formuláře klíčová slova. Ta se odešlou vyhledávacímu serveru, kde CGI-skript prohledá indexy. Výsledkem běhu CGI-skriptu je pak stránka v HTML, která obsahuje odkazy na stránky vyhovující dotazu.

Vše vypadá jednoduše, ale skutečnost bývá složitější. Psaní CGI-skriptů nebylo úplně snadné. Pro jejich psaní se používaly nejčastěji různé interpretované jazyky, jako Perl nebo příkazové shelly Unixu. Nebyl však problém použít v podstatě libovolný programovací jazyk a tak existuje mnoho CGI-skriptů napsaných v jazycích C a C++. Nový módní jazyk Java lze pro psaní CGI-skriptů rovněž použít — takovým programům se pak říká servlety. Velké databázové systémy, jako např. Oracle, umožňují psaní CGI-skriptu přímo ve svém vlastním jazyce (např. PL/SQL).

Pro vytvoření CGI-skriptu tedy byla nutná znalost nějakého programovacího jazyka. Kromě toho musel člověk ovládat rozhraní CGI, které nepředávalo parametry zrovna v šikovném formátu. Většina CGI-skriptů se proto z větší části skládala z kódu, který převáděl získané parametry do použitelné podoby.

V interpretovaných CGI-skriptech navíc spoustu práce stálo dostatečné zabezpečení skriptu. Šikovný hacker totiž mohl odesláním speciálního textu v polích formuláře získat přístup k systému, na kterém běžel WWW-server.

CGI-skripty generují svůj výstup v jazyce HTML. Jazyk HTML však nejde přímo kombinovat s jinými jazyky, a proto bylo generování HTML-kódu v CGI-skriptu otravná záležitost, kde se každá řádka HTML-kódu zadávala jako parametr příkazu `print` či `echo` podle použitého jazyka. Správa větších aplikací je

rovněž náročná, protože aplikace je roztroušena v mnoha samostatných souborech s HTML-stránkami a CGI-skripty.

Šikovný programátor však pomocí CGI-skriptů dokázal vytvořit přímo kouzla. I technologie CGI má však své meze. Vidíme, že CGI-skripty se provádějí na WWW-serveru. Uživatelská odezva je tedy velmi pomalá — uživatel si stáhne stránku, poté vyplní a odešle formulář zpět na server, server spustí CGI-skript a od něj získaný výstup zašle zpět do uživatelova prohlížeče.

Zhruba ve stejné době jako CGI-skripty se poměrně rozšířila i další technologie *SSI (Server Side Includes)*. SSI byly jednoduché příkazy, které se zadávaly do HTML-stránky jako komentář. Stránky však byly uloženy v souborech se speciální příponou `.shtml` a tak WWW-server věděl, že před odesláním stránky v ní má provést všechny SSI. SSI umožnily provádění jednoduchých úkonů, jako vložení jiného souboru do stránky, nebo vypsání data poslední modifikace dokumentu. Své uplatnění nalezly především na rozsáhlých serverech, které chtěly mít na všech stránkách standardizované záhlaví a patičku — optimální úkol pro nasazení SSI.

Řešení pomalé odezvy CGI-skriptů spočívalo v přesunutí provádění programů na stranu klienta — do prohlížeče. Zhruba ve stejné době — během roku 1996 — byly představeny dvě různé technologie, který daný problém řeší.

První technologií byl nový jazyk *Java* představený firmou Sun Microsystems. Tento jazyk umožňoval psaní Java-apletů, což byly krátké programy, které byly začleněny přímo do HTML-stránky. Ve stránce měly vyhrazen prostor, který byl zcela pod jejich kontrolou. Možnosti Javy jsou opravdu široké — od jednoduchých animací oživujících stránku až po zábavné hry či aplikace, které vám spočítají hladinu alkoholu v krvi.[1]

Velkou výhodou Javy byla její nezávislost na platformě — programy se po síti přenášely ve formě tzv. *byte-code* (bajtového kódu), který je spustitelný v libovolném operačním systému, pokud pro něj existuje interpret Javy (JVM). Dnes můžeme s odstupem času říci, že Java je výborná a perspektivní technologie. Na běžných stránkách se s ní však zatím moc nepotkáme, protože je kvůli své univerzálnosti poměrně náročná na systémové zdroje počítače.

Druhou novou technologií roku 1996 byl *JavaScript*. JavaScript je jednoduchý jazyk se syntaxí vycházející z jazyku Java. S JavaScriptem přišla firma Netscape a zabudovala jej do svého legendárního prohlížeče Netscape Navigator. JavaScript se zapisoval přímo do HTML-kódu stránky a uměl posloužit v mnoha situacích. Jeho nejčastější použití bylo ve spojení s formuláři. Krátké skripty v JavaScriptu mohly kontrolovat správnost údajů v polích formuláře ještě před odesláním na server. Uživatel tak získal nesrovnatelně rychlejší odezvu v porovnání s klasickým způsobem využívajícím pouze CGI-skripty. Druhou

[1] Musíte samozřejmě zadat všechny pangalaktické megacloumáky, které jste předtím prolili některým ze svých hrdel.

oblastí použití JavaScriptu byla drobná vylepšení interaktivnosti stránek — celkem snadno šlo např. zařídit, aby odkaz změnil barvu po přejetí myší. Uživatel tak hned věděl, že za odkazem se (možná) skrývá něco zajímavého.

Dnes Javu i JavaScript podporují oba nejrozšířenější prohlížeče Netscape Navigator (NN) i Microsoft Internet Explorer (MSIE). Smutnou pravdou však je, že implementace JavaScriptu v MSIE není zcela kompatibilní s NN. To nutí tvůrce stránek k vytváření složitějších skriptů, které se umějí přizpůsobit vlastnostem jednotlivých prohlížečů.

Pojďme se však přesunout zpět na stranu serveru. Úspěch JavaScriptu byl tak obrovský, že se firma Netscape rozhodla pro využití JavaScriptu na straně serveru. Na serverech Netscape šlo do HTML-stránek psát skripty, které se provedou přímo na serveru. Uvnitř stránky byly skripty uzavřeny mezi tagy `<SERVER>` a `</SERVER>` a server tak snadno rozpoznal, které části stránky má interpretovat. Výsledkem skriptů musel být HTML-kód, který se doplnil do zbytku stránky a prohlížeči se již zasílala obyčejná HTML-stránka. Řešení se dříve šířilo pod názvem LiveWire, dnes je jméno výstižnější — *SSJS (Server Side JavaScript)*.

V SSJS je k dispozici mnoho objektů, které umožňují snadno pracovat s daty z formulářů, s databázemi atd. Výsledný efekt aplikací napsaných v SSJS je tedy stejný jako u CGI-skriptů, s tím rozdílem, že psaní SSJS je mnohem jednodušší.

Aby Microsoft nezůstal pozadu, uvedl na trh *ASP (Active Server Pages)*. ASP jsou obdobou SSJS. Jako programovací jazyk je možno využít VBScript nebo JScript, což je microsoftí implementace JavaScriptu. Systémy samozřejmě nejsou kompatibilní — ASP používá jiné značky k oddělení skriptu od stránky a hierarchie objektů, které zpřístupňují všechny důležité údaje, je rovněž rozdílná.

Kromě VBScriptu a JScriptu je možno v ASP používat další jazyky, které dodávají třetí firmy — Perl, REXX, Python.

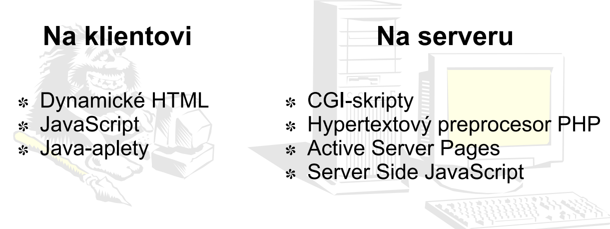
Už se to blíží — na scénu webových technologií brzy vstoupí systém PHP. SSJS i ASP mají jednu velkou společnou nevýhodu — jsou to komerční produkty, které nejsou nikterak levné a jejich použití je navíc svázáno s použitím WWW-serveru dané firmy. ASP navíc běží pouze na platformě Windows — pokud se tedy v budoucnu rozhodneme ASP-aplikaci z Windows NT přesunout na výkonnější Unix, máme prostě smůlu.

Všechny tyto nedostatky a mnohé další odstraňuje systém PHP. Princip použití PHP je obdobný jako u SSJS a ASP. Narozdíl od nich je však šířen celý produkt jako freeware — to znamená bezplatně. Pokud se vám na tomto způsobu šíření softwaru něco nezdá, zeptám se jinak — „*Jaký je nejpoužívanější webový server?*“. Správná odpověď zní Apache. Apache je používán na téměř 50 % všech WWW-serverů a vznikl na stejné bázi jako PHP. Mezi další osvědčené a volně šířené systémy patří například operační systém Linux a typografický systém TEX — opět špičky ve svém oboru.

Další výhodou PHP je jeho nezávislost na platformě. Dnes jsou k dispozici verze PHP pro Unix a Windows. PHP není svázáno s žádným konkrétním serverem, může běžet na libovolném. Nejlépe si však dnes rozumí se serverem Apache.

Kromě těchto objektivních příčin mi PHP i subjektivně připadá mnohem použitelnější. PHP-skript, který má stejnou funkčnost jako ASP, je obvykle mnohem kratší a srozumitelnější. Během čtení knihy sami poznáte, že PHP bylo vytvořeno přesně pro psaní skriptů začleňovaných do HTML stránek, jeho použití je jasné, intuitivní a přímočaré. To dokazuje i počet serverů, na kterých je PHP používáno. V druhé polovině roku 1998 tento počet přesahoval 150 000.

V další sekci se podrobněji zaměříme na historii vývoje systému PHP. Nyní ještě v krátkosti dokončíme stručný přehled technologií, které umožňují vytváření interaktivních stránek.



Obr. 1-2: Kde, která technologie běží…

Možnosti JavaScriptu na straně klienta byly v původní implementaci od firmy Netscape nevyužity, a proto přišel v roce 1997 Microsoft ve svém prohlížeči MSIE 4.0 s novou technologií *dynamického HTML (DHTML)*. DHTML vychází ze stejné myšlenky jako JavaScript, ale mnohem více ji dotahuje. V DHTML je přístupný každý objekt webovské stránky a s každým objektem můžeme kdykoliv manipulovat — měnit jeho obsah, způsob zobrazení či polohu. Jako objekty jsou přístupné i styly připojené ke stránce a jejich změnou lze velice jednoduše dosáhnout překvapivých vizuálních efektů. Teprve stránky zapsané pomocí DHTML jsou skutečně interaktivní. Velkou výhodou DHTML je zachování obousměrné kompatibility s běžnými HTML-stránkami.

Všechny výše zmíněné technologie můžeme rozdělit do dvou skupin podle toho, zda jsou prováděny na serveru nebo na klientovi. Nic však nebrání jejich vzájemné kombinaci, pokud je účelná. V naší knize se budeme věnovat programování na straně serveru, ale ukážeme si, jak je vhodně kombinovat např. s klientským JavaScriptem při kontrole údajů v odesílaném formuláři.

1.2 Jak přišlo PHP na svět

Seznamme si nyní s krátkou historií PHP. Na počátku zrodu systému stál Rasmus Lerdorf. Psal se rok 1994 a Rasmus si ve volném čase vytvořil v Perlu jednoduchý systém pro evidování přístupu k jeho stránkám. Jelikož neustálé spouštění interpretu Perlu velmi zatěžovalo WWW-server, přepsal autor systém do jazyka C.

Ačkoliv byl celý systém původně určen pro osobní Rasmusovo použití, zalíbil se i ostatním uživatelům serveru a začali ho používat. Systém se stal oblíbeným a používalo jej stále více uživatelů. Ti přicházeli s požadavky na vylepšení celého systému. Autor proto systém rozšířil, doplnil o dokumentaci a uvolnil jej pod názvem Personal Home Page Tools, který se později změnil na Personal Home Page Construction Kit. V téže době autor zprovoznil elektronickou konferenci, která sloužila jako prostor pro výměnu zkušeností mezi uživateli systému.

Kromě již zmíněného systému pro evidování přístupů ke stránkám vytvořil pan Lerdorf i nástroj, který umožňoval začleňování SQL-dotazů do stránek, vytváření formulářů a zobrazování výsledků dotazů. Program, který umožnil zpřístupnění databází na Webu, se jmenoval Form Interpreter (FI).

Celosvětovou proslulost si získal systém PHP/FI 2.0. Tento systém vznikl spojením dvou předchozích programů autora. V této podobě se jednalo o jednoduchý programovací jazyk, který se zapisoval přímo do HTML-stránek. PHP/FI 2.0 se rozšířilo opravdu po celém světě a pracovalo i na mnoha českých serverech.

Poslední verzí je PHP 3.0. Oproti předchozí verzi byl celý systém zrychlen a rozšířen o mnoho dalších funkcí. Tato verze PHP je v ostré verzi k dispozici od poloviny roku 1998. Narozdíl od verze 2.0 běží nová verze i pod Windows. Projekt se rozšířil a na jeho vývoji pracuje Rasmus Lerdorf s několika dalšími vývojáři.

Obsah zkratky PHP nyní zcela ztrácí na svém původním významu a doporučené označení celého systému je *hypertextový preprocesor PHP* (PHP: Hypertext Preprocessor). My budeme používat zažitou zkratku PHP.

Vývoj PHP však není zdaleka ukončen. Do jazyka jsou přidávány stále nové možnosti, které umožňují efektivnější vytváření internetových aplikací. V době dokončování knihy byla zveřejněna zpráva o vývoji nového jádra PHP pod názvem Zend. Podle testů se ukazuje, že nový engine je schopný skripty provádět až desetkrát rychleji než verze 3.0. Nemusíme se tedy bát, že by PHP bylo nějakou mrtvou a dále nevyvíjenou technologií.

1.3 Kde můžeme získat PHP

Jak již jsme řekli, PHP je volně šiřitelný software. Z toho mimo jiné vyplývá, že ho nekoupíme v běžném obchodě se softwarem. Zdarma si jej můžeme stáhnout na Internetu z adresy:

`http://www.php.net/`

Na této adrese nalezneme i spoustu dalších informací, které se týkají PHP — jedná se totiž o oficiální server celého projektu. Na serveru nalezneme i seznam jeho zrcadel. Pro samotné stahování si můžeme vybrat zrcadlo, ke kterému máme nejrychlejší přístup.

Další možností, jak s ještě menší námahou získat systém PHP, jsou různé distribuce operačních systémů. Zejména několik posledních distribucí Linuxu v sobě obsahovalo systém PHP.

Nejčastější případ asi bude, že se o stahování a instalaci PHP už nebudete vůbec zajímat. Systém bude nainstalován na vašem serveru — v práci, ve škole nebo u poskytovatele připojení. Vaším zájmem tedy bude samotný jazyk PHP.

Těm z vás, kteří stojí před úkolem instalace PHP, je určena jedenáctá kapitola, která se podrobně věnuje instalaci a konfiguraci systému. Na začátek jsme ji neumístili záměrně, protože ji nevyužije zdaleka každý.

Podívejme se nyní ještě, na jakých platformách můžeme dnes PHP provozovat a jaké verze jsou k dispozici. V současné době podporuje PHP operační systémy Unix, 32bitová Windows (95, 98 a NT) a OS počítačů Macintosh.

Samotné PHP pak může pracovat s libovolným WWW-serverem, který umožňuje spouštění CGI-skriptů. V tomto případě PHP pracuje jako CGI-skript, který zpracovává jednotlivé stránky. Výhodou tohoto řešení je kompatibilita s téměř všemi WWW-servery. Konkurenční technologie Microsoftu (ASP) a Netscapu (SSJS) jsou pevně svázány s firemním WWW-serverem.

Spouštění PHP jako CGI-skript má i své nevýhody. Pro každou stránku zapsanou v PHP se musí znovu spustit interpret PHP, který obslouží požadavek. Opakované spouštění může na hodně zatížených serverech zdržovat. Proto existuje i druhá možnost, jak může systém PHP pracovat.

PHP může být přímo zakompilováno jako modul do serveru Apache. Interpret jazyka PHP je pak stále zaveden v paměti společně s WWW-serverem a odpadají tak veškeré režijní náklady spojené s opakovaným spouštěním interpretu PHP. Pokud jako WWW-server používáte Apache pod Unixem, je použití modulu PHP tím nejlepším způsobem. Navíc pokud PHP pracuje jako modul serveru Apache, může PHP využívat některé vnitřní mechanismy Apache jako je např. kontrola přístupu k dokumentům apod.

Běží-li váš server na Unixu a jmenuje se Apache, neuvažujte o jiné verzi PHP, než o modulu přímo určeném pro práci s Apache. Jedná se o vůbec nejvýkonnější variantu systému PHP.

Tým vývojářů PHP v současné době usilovně pracuje i na dokončení verze PHP, která bude pracovat jako modul ISAPI. ISAPI je rozhraní používané ve Windows pro komunikaci WWW-serveru s dalšími aplikacemi. ISAPI verze PHP je DLL-knihovna, která je také neustále zavedena v paměti a nabízí stejné možnosti jako modul pro Apache. ISAPI je standard, který podporuje více serverů pro Windows, a tak nebude využití ISAPI verze PHP svázáno pouze s jedním konkrétním serverem.

V době, kdy čtete knihu, bude ISAPI verze PHP již pravděpodobně hotova. Pokud váš server pracuje pod Windows, měli byste používat výhradně tuto verzi systému PHP, protože nabízí mnohem větší výkon a méně zatěžuje systémové zdroje celého serveru.

1.4 Co budeme potřebovat pro tvorbu aplikací v PHP

Již jsme se zmínili, že snadnost práce s PHP spočívá v tom, že se příkazy PHP přirozeně kombinují s HTML-kódem. Základním nástrojem, který budeme potřebovat pro tvorbu skriptů, je editor. Většinou si vystačíme s editorem, který jsme až dosud používali pro tvorbu stránek. Jediným požadavkem je, aby editor uměl pracovat přímo se zdrojovým textem stránky v HTML. Různé WYSIWYG editory jako např. Microsoft FrontPage nebo Netscape Composer jsou pro tyto účely zcela nevhodné.

S výhodou však můžeme použít editory pracující přímo se zdrojovým zápisem stránky — např. ve Windows oblíbený HomeSite. Samozřejmě můžeme použít jakýkoliv textový editor. Počínaje Poznámkovým blokem z Windows a konče nepřekonatelným editorem MultiEdit, který je opravdu špičkou. Pro méně majetné jsou k dispozici i výborné freewarové editory — pro Windows například Programmer's File Editor.

Pokud hodláte vytvářet skripty v prostředí Unixu, je asi zbytečné rozebírat, jaký editor můžete použít. Používaní textových editorů patří v Unixu k dennímu chlebu a každý uživatel má svůj editor, na který nedá dopustit — ať je to `vi`, `emacs`, `joe` či `pico`. (Jestli jsem na váš oblíbený zapomněl, nekamenujte mě prosím. ;–)

Není důležité, jaký editor používáte. Důležité je, aby se vám s ním dobře pracovalo a uměli jste jej ovládat. Vychytanější editory jako MultiEdit a `emacs` vám však mohou nabídnout mnohé užitečné funkce navíc — např. zvýrazňovaní syntaxe, které zpřehlední orientaci ve vytvářených skriptech.

Když v editoru napíšeme nějaký skript, musíme jej uložit na WWW-server, který podporuje skripty v PHP. K nějakému takovému serveru tedy musíme mít přístup. Pro naše první pokusy nemusí být dotyčný server vůbec připojen k Internetu. Na svém počítači si můžeme spustit nějaký jednoduchý WWW-server doplněný o podporu PHP. Všechny skripty si vyzkoušíme, pořádně prostudujeme jazyk PHP a až nakonec umístíme hotovou sadu skriptů tvořících aplikaci na nějaký veřejně dostupný server v Internetu nebo Intranetu. Informace o instalaci PHP na váš počítač naleznete v jedenácté kapitole.

Pro prohlížení výsledku běhu našich skriptů si vystačíme s libovolným prohlížečem. Ten má jistě každý z nás na svém počítači nainstalován.

Pokud chceme vytvářet opravdu profesionální aplikace, měli bychom si naše dílo prohlédnout ve všech nejpoužívanějších prohlížečích. Dnes mezi ně patří Netscape Navigator, MS Internet Explorer a textový Lynx. Prohlížeče občas některé části HTML-stránky zobrazují odlišně. Není to však problém skriptů, ale samotného jazyka HTML, resp. jeho podpory jednotlivými prohlížeči.

Pokud máme ambice vytvářet na Webu databázové aplikace, musíme si k tomu pořídit vhodný databázový server. V tuto chvíli nám bude stačit informace, že PHP si rozumí se všemi běžně používanými databázemi. Podrobnější informace naleznete v šesté kapitole věnované tvorbě databázových aplikací.

2. Rychlé seznámení

Cílem této kapitoly je stručně vás seznámit s tím, co PHP dovede. Na několika jednoduchých příkladech si ukážeme, jak se v PHP vytvářejí aplikace. Je poměrně pravděpodobné, že všem příkazům a konstrukcím v ukázkách nebudete napoprvé rozumět. To však vůbec nevadí — vše postupně a hlavně srozumitelně probereme v následujících kapitolách. Kapitola slouží pouze jako prostředník, který vás seznámí se způsobem práce celého systému PHP. Tuto představu budeme potřebovat pro správné pochopení dalšího výkladu.

2.1 Náš první skript

Skripty v PHP jsou obyčejné HTML-stránky doplněné o výkonné příkazy. Aby server od sebe odlišil obyčejné HTML-stránky a ty zapsané PHP, ukládají se PHP-skripty do souborů se zvláštní příponou. Tato přípona bývá nejčastěji `.phtml`, `.php3` nebo `.php` — záleží na konfiguraci serveru.

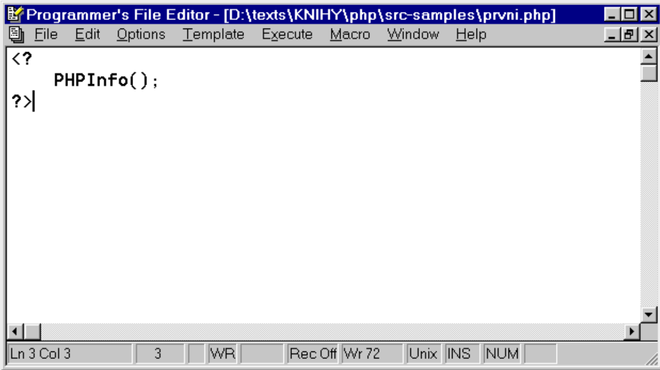
Pokud WWW-server obdrží požadavek na soubor, který končí na některou z uvedených přípon, předá požadavek systému PHP. Ten se postará o provedení všech příkazů uložených v souboru a výsledek předá zpět serveru. Server výslednou stránku odešle jako odpověď klientovi.

V naší knize budeme pro skripty používat příponu `.php`, která je nejkratší a přitom zcela výstižná.

Pojďme si tedy ukázat první jednoduchý skript. V textovém editoru vytvoříme soubor `prvni.php` s následujícím obsahem:

```
<?
    PHPInfo();
?>
```

Soubor musíme uložit do adresáře, který je součástí stromu dokumentů WWW-serveru. Tento adresář je na každém systému jiný. Na Unixu se nejčastěji jedná o adresář `public_html`, který je ve vašem domovském adresáři. Ve Windows jméno adresáře záleží na použitém serveru. Vždy se však jedná o stejný adresář, do kterého jste dosud ukládali vaše webové stránky.

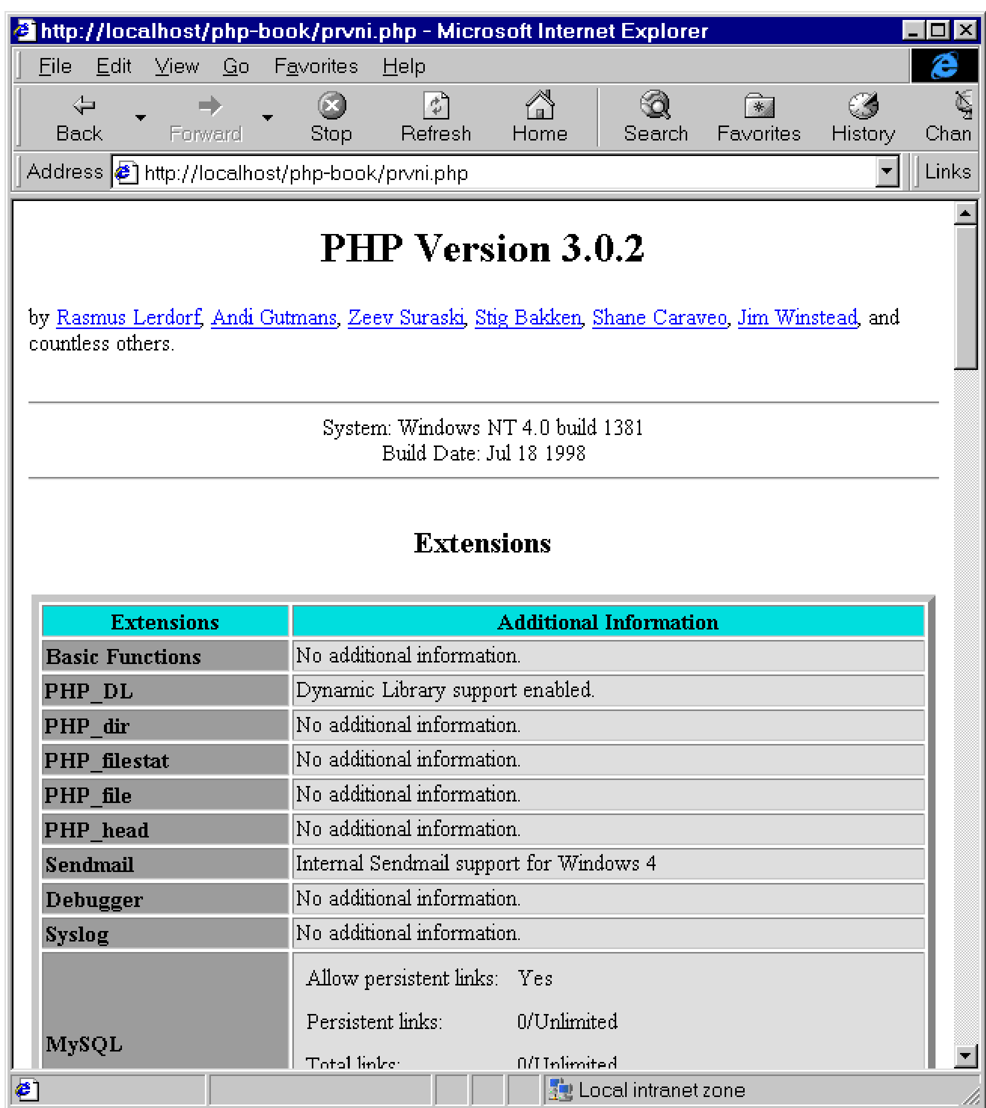


Obr. 2-1: Vytváření skriptu v textovém editoru

Po uložení skriptu do souboru nás vždy čeká mrazivá chvilka. Provede skript to, co má, a my uvidíme v prohlížeči požadovaný výsledek? Nebudeme si tyto napjaté chvíle prodlužovat a spustíme prohlížeč. V prohlížeči zadáme URL našeho skriptu, abychom si mohli prohlédnout jeho výsledek.

> Skript musíme vždy vyvolat pomocí jeho URL adresy. V prohlížeči tedy zadáme jeho adresu, která může vypadat třeba `http://www.nekde.cz/~jirka/prvni.php` nebo třeba `http://localhost/prvni.php`. Skript však nesmíme v prohlížeči otevírat jako soubor přímo z disku. Pokud v prohlížeči zadáme jako URL dokumentu pouze cestu k souboru — např. `d:\webpub\php\prvni.php`, nezobrazí se výsledek skriptu. Skript musí být před zasláním interpretován systémem PHP. Interpret je vyvolán pouze v případě, kdy k dokumentu (skriptu) přistupujeme přes WWW-server, tj. pomocí URL, které začíná na `http`.

Nyní si prohlédneme výsledek našeho prvního skriptu v prohlížeči (obr. 2-2 na následující straně). Pokud je vše v pořádku, měla by se zobrazit stránka, která obsahuje informace o používané verzi systému PHP.



Obr. 2-2: Zobrazení našeho prvního skriptu v prohlížeči

Pokud se vám nezobrazil požadovaný výsledek, zkontrolujte, zda jste při opisování skriptu neudělali překlep. Pokud to nepomůže, ujistěte se, že jste soubor uložili do správného adresáře a v prohlížeči máte správně zadané URL dokumentu. Pokud i zde je vše v pořádku, bude patrně problém v instalaci a konfiguraci PHP. Problém konzultujte se správcem vašeho systému nebo s jedenáctou kapitolou věnovanou instalaci a konfiguraci.

Proč se v prohlížeči zobrazilo právě to, co vidíme? Náš skript obsahoval příkaz `PHPInfo`, který vygeneruje informace o celém systému. Je to poněkud netypický příkaz, protože vygeneruje celou HTML-stránku. Náš první skript proto nemusel obsahovat žádné značky jazyka HTML.

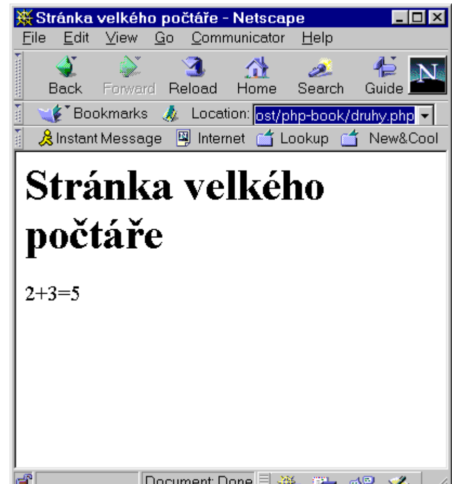
Dvojice znaků ‘`<?`’ a ‘`?>`’, uvnitř kterých je příkaz uzavřen, slouží k oddělení příkazů PHP od HTML-kódu. V našem skriptu žádný HTML-kód sice není, ale přesto musíme znaky použít. PHP interpretuje pouze příkazy uvedené mezi těmito speciálními značkami.

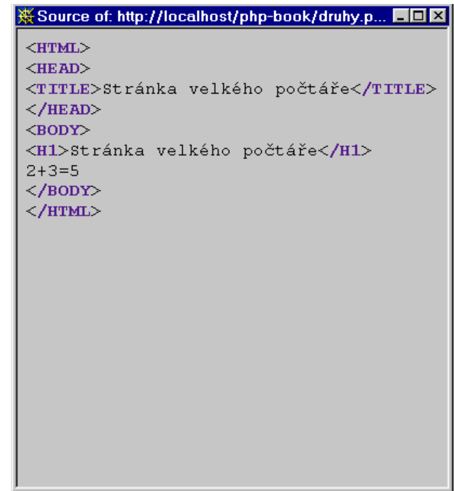
Ukažme si nyní, jak se dají příkazy PHP kombinovat s HTML-kódem. Vytvoříme stránku, která nám sama spočítá, kolik je $2 + 3$.

Příklad: `druhy.php`

```
<HTML>
<HEAD>
<TITLE>Stránka velkého počtáře</TITLE>
</HEAD>
<BODY>
<H1>Stránka velkého počtáře</H1>
<?
  $a = 2;
  $b = 3;
  $c = $a+$b;
  echo "$a+$b=$c\n";
?>
</BODY>
</HTML>
```

V této ukázce je již patrné, jak se kombinuje HTML-kód s příkazy PHP. Ve skriptu jsme použili tři proměnné `$a`, `$b` a `$c`. Do proměnných můžeme přiřazovat různé hodnoty a vzájemně je kombinovat pomocí matematických operací.





Obr. 2-3: Výsledkem skriptu je vždy čisté HTML

Příkaz `echo` slouží k vypsání textu za ním uvedeného. V tomto textu můžeme samozřejmě používat dříve vytvořené proměnné.

Výsledné zobrazení v prohlížeči přináší obrázek 2-3. Na stejném obrázku je vidět i zdrojový text stránky, která dorazila do prohlížeče. Zobrazení zdrojového textu vyvoláme v prohlížeči příkazem Zobrazit ▷ Zdroj nebo View ▷ Page Source. Vidíme, že do prohlížeče dorazí již pouhá webová stránka. Všechny příkazy PHP jsou na serveru provedeny a výsledkem je čisté HTML, které umí zobrazit každý prohlížeč.

V PHP není problém doplnit stránku o zobrazení aktuálního času. Ukážeme si to na následujícím příkladě.

Příklad: `treti.php`

```
<HTML>
<HEAD>
<TITLE>Superserver s hodinami</TITLE>
</HEAD>
<BODY>
<DIV ALIGN=CENTER>
<H1>Vítejte na serveru</H1>
Právě je <?echo Date("H:i");?>
</DIV>
</BODY>
</HTML>
```

Obr. 2-4: Zobrazení aktuálního času není v PHP problém

Vše obstará funkce `Date()`, jejímž parametrem je požadovaný formát zobrazení času. V našem případě zápis `"H:i"` způsobí vypsání hodin a minut oddělených dvojtečkou.

Pokud vytváříme stránku, kde se nějaké úseky HTML opakují, můžeme k usnadnění práce použít cykly. Naše další ukázka nemá valný praktický význam, slouží pouze jako vhodná ukázka.

Příklad: `ctvrty.php`

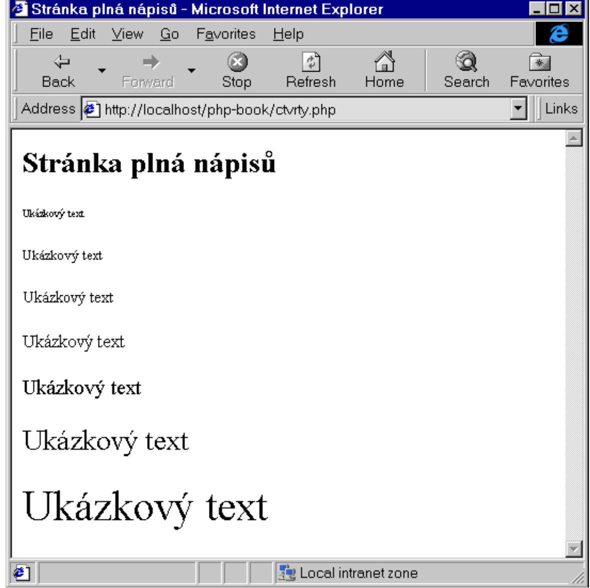
```
<HTML>
<HEAD>
<TITLE>Stránka plná nápisů</TITLE>
</HEAD>
<BODY>
<H1>Stránka plná nápisů</H1>
<?for($i=1; $i<=7; $i++):?>
  <P><FONT SIZE="<?echo $i?>">Ukázkový text</FONT>
<?endfor?>
</BODY>
</HTML>
```

Tento skript dorazí po svém provedení do prohlížeče jako mnohem delší stránka v HTML:

```
<HTML>
<HEAD>
<TITLE>Stránka plná nápisů</TITLE>
```

```
</HEAD>
<BODY>
<H1>Stránka plná nápisů</H1>
  <P><FONT SIZE="1">Ukázkový text</FONT>
  <P><FONT SIZE="2">Ukázkový text</FONT>
  <P><FONT SIZE="3">Ukázkový text</FONT>
  <P><FONT SIZE="4">Ukázkový text</FONT>
  <P><FONT SIZE="5">Ukázkový text</FONT>
  <P><FONT SIZE="6">Ukázkový text</FONT>
  <P><FONT SIZE="7">Ukázkový text</FONT>
</BODY>
</HTML>
```

Pokud vás zajímá, jak to dopadne v prohlížeči, prohlédněte si obrázek 2-5. Poznamenejme ještě, že o vše se postaral příkaz `for`. Vše co je mezi příkazy `for` a `endfor` se provedlo celkem sedmkrát, protože proměnná `$i` nabývala postupně hodnot od `1` do `7`.



Obr. 2-5: Cykly nám v PHP usnadní mnoho práce

2.2 Snadná práce s formuláři

Téměř všechny dosavadní ukázky možností PHP byly sice pěkné a efektní, ale popravdě řečeno užitečné zrovna dvakrát nebyly. To se však již nedá říci o možnostech, které PHP nabízí při zpracování dat z formulářů.

Formuláře jsou vstupní branou do interaktivního světa Webu. Pomocí formulářů mohou uživatelé na stránkách zadávat objednávky, klást dotazy na různé databáze apod. Aby však nebyl formulář jen černou dírou, musí k němu existovat skript, který uživatelem zadaná data zpracuje a na jejich základě poskytne uživateli adekvátní odezvu.

My si snadnost práce s parametry z formuláře ukážeme na jednoduché aplikaci. Uživatel do formuláře zadá své jméno a svůj věk. Skript pak vysloví nemilosrdný soud nad jeho věkem. Do stránky umístíme následující formulář:

```
<FORM ACTION="obsluha.php" METHOD=POST>
Jméno: <INPUT NAME=Jmeno><BR>
Věk  : <INPUT NAME=Vek><BR>
<INPUT TYPE=Submit VALUE="Odeslání formuláře">
</FORM>
```

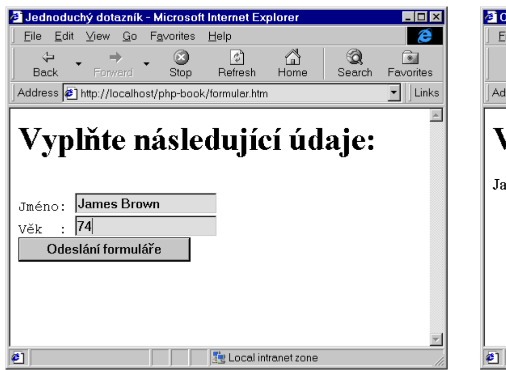
Obsah atributu `ACTION` určuje skript, který bude vyvolán pro obsluhu dat formuláře. Skript `obsluha.php` se vyvolá poté, co uživatel vyplní formulář a odešle jej stiskem tlačítka Odeslání formuláře.

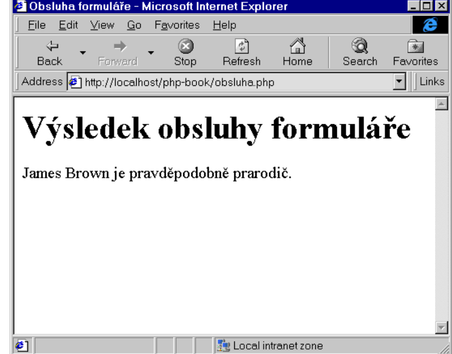
Příklad: `obsluha.php`

```
<HTML>
<HEAD>
<TITLE>Obsluha formuláře</TITLE>
</HEAD>
<BODY>
<H1>Výsledek obsluhy formuláře</H1>
<?  echo $Jmeno ?> je
<?  if ($Vek < 10):
        echo "pěknej mlíčňák";
    elseif ($Vek < 20):
        echo "teenager";
    elseif ($Vek < 60):
        echo "v nejlepších letech";
    elseif ($Vek < 100):
        echo "pravděpodobně prarodič";
    else:
        echo "někde mezi stovkou a smrtí";
    endif
```

```
?>.
</BODY>
</HTML>
```

Ve skriptu máme obsah polí přístupný pomocí proměnných. V našem případě máme k dispozici jméno a věk uživatele v proměnných `$Jmeno` a `$Vek`. Pomocí podmínek pak skript v závislosti na zadaném věku vybere správný text, který vypíše.





Obr. 2-6: Formulář a výsledek jeho zpracování

2.3 Spolupráce s databázemi

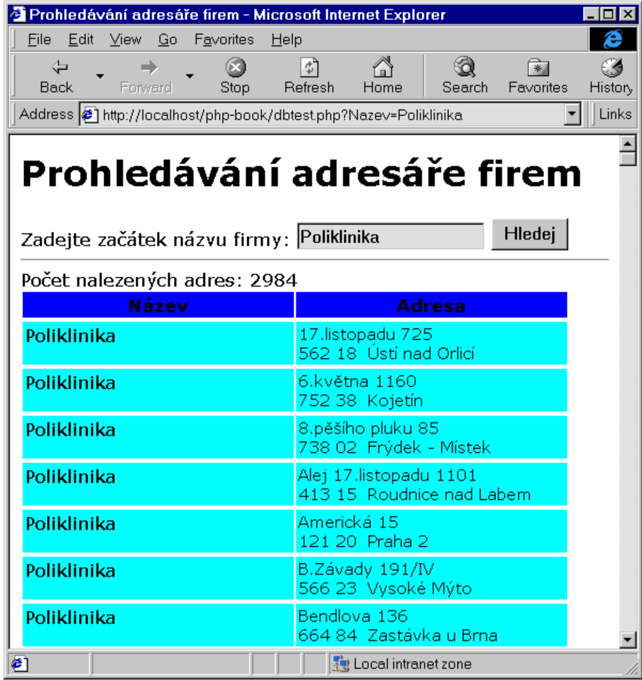
Opravdovou lahůdkou, kterou nám PHP nabízí, je snadná spolupráce s databázemi. Skripty mohou pomocí jazyka SQL provádět s údaji v databázích libovolné operace.

Možnosti praktického využití databází jsou opravdu velmi široké. My si ukážeme jednoduchý skript, který uživatelům umožňuje pohodlné vyhledávání v adresáři firem.

Příklad: `dbtest.php`

```
<HTML>
<HEAD>
<TITLE>Prohledávání adresáře firem</TITLE>
</HEAD>
<BODY>
<H1>Prohledávání adresáře firem</H1>
<FORM ACTION="dbtest.php">
Zadejte začátek názvu firmy: <INPUT NAME=Nazev VALUE="<?echo $Nazev?>">
<INPUT TYPE=Submit VALUE="Hledej">
```

```
</FORM>
<HR>
<?
if ($Nazev!=""):
    $spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):
        echo "Nepodařilo se připojit k databázi.";
    else:
        $vysledek = ODBC_Exec($spojeni,
                    "SELECT * FROM adresy WHERE Nazev LIKE '$Nazev%'");
        if (!$vysledek):
            echo "Chyba při provádění dotazu v databázi";
        else:
            echo "Počet nalezených adres: ".ODBC_Num_Rows($vysledek);
            echo "<TABLE CELLPADDING=2>\n";
            echo "<TR><TH BGCOLOR=BLUE WIDTH=250>Název</TH>
                  <TH BGCOLOR=BLUE WIDTH=250>Adresa</TH></TR>\n";
            echo "</TABLE>\n";
            while (ODBC_Fetch_Row($vysledek)):
                echo "<TABLE CELLPADDING=2>\n";
                echo "<TR VALIGN=TOP><TD BGCOLOR=AQUA WIDTH=250>".
                     ODBC_Result($vysledek, "Nazev")."</TD>";
                echo "<TD BGCOLOR=AQUA WIDTH=250><SMALL>".
                     ODBC_Result($vysledek, "Ulice")."<BR>".
                     ODBC_Result($vysledek, "PSC")."  ".
                     ODBC_Result($vysledek, "Mesto")."</SMALL><TD></TR>";
                echo "</TABLE>\n";
            endwhile;
        endif;
        ODBC_Close($spojeni);
    endif;
endif;
?>
</BODY>
</HTML>
```



Obr. 2-7: PHP si s velkými databázemi snadno poradí

Jeden z možných výsledků použití našeho skriptu vidíme na obrázku 2-7.

Dovolte alespoň stručný komentář k celému skriptu. Náš příklad pro přístup k databázi používá standardní rozhraní ODBC. Je to jedna z možností, kterou PHP při přístupu k databázím nabízí. Voláním funkce `ODBC_Connect()` se připojíme k databázovému serveru. Funkce `ODBC_Exec()` slouží k zadání dotazu v jazyce SQL.

Výsledek dotazu postupně zpracováváme pomocí funkcí `ODBC_Fetch_Row()` a `ODBC_Result()`. Jak jste jistě vytušili, funkce `ODBC_Num_Rows()` vrací počet záznamů, které vyhovují dotazu.

Výsledek přehledně formátujeme pomocí tabulek. Pro každou adresu vytváříme vlastní tabulku, aby bylo vykreslování celé stránky rychlejší.

Problematika databází je velice zajímavá a široká. Náš příklad byl pouze nepatrnou ukázkou toho, co PHP v této oblasti dovede. Velice podrobně se budeme databázemi zabývat v šesté kapitole. Kromě jiného se v ní seznámíme s jazykem SQL a s tím, jak vytvářet efektivní databázové aplikace.

3. Jazyk PHP

Již máme představu o tom, co za malá kouzla lze provádět v PHP. Je proto pravý čas na podrobný průzkum jazyka PHP. V této kapitole si ukážeme všechny způsoby, kterými lze PHP začleňovat do HTML-stránek. Následovat bude popis práce s proměnnými různých typů. S tím úzce souvisí i problematika výrazů. Dozvíme se, co je to výraz a jaké operátory můžeme používat při zápisu výrazů. V poslední části kapitoly podrobně probereme jednotlivé příkazy, které nám PHP nabízí — včetně příkazů pro definování vlastních funkcí a objektových tříd.

3.1 Vkládání PHP do HTML

Zatím jsme si ukázali pouze jeden způsob, kterým se dají oddělovat příkazy PHP od kódu HTML. Všechny příkazy se zapisovaly mezi znaky ‘<?’ a ‘?>’. Volba právě těchto znaků není náhodná. Elementy, které můžeme používat v jazyce HTML, jsou definovány pomocí jazyka SGML. Jazyk SGML zároveň definuje, že příkazy pro různé preprocesory, které dokument zpracovávají, mají být uzavřeny právě v této dvojici znaků.[1]

```
<? echo "Stručně, jasně, věcně."; ?>
```

Druhá možnost je již trošku upovídanější. Pokud chceme PHP použít pro generování XML-dokumentů, musíme použít zahajovací značku ‘<?php’. XML je zjednodušená verze SGML, která v některých rysech reaguje na nově vzniklé požadavky. Jedním z nich je možnost zařazení příkazů pro více různých preprocesorů najednou. Proto je vždy nutné určit, pro jaký systém jsou vkládané příkazy určeny. K označení systému PHP slouží právě identifikátor php:

```
<?php echo "Generování XML-dokumentů. Žádný problém"; ?>
```

Poslední možnost, jak zařadit skript do stránky, je opravdu výřečná:

```
<SCRIPT LANGUAGE="php">
    echo "Některé HTML-editory si se skripty jinak neporadí.";
</SCRIPT>
```

[1] Přesněji řečeno, v SGML se pro oddělení různých příkazů od textu dokumentu používá speciální sekvence znaků označovaná jako PI (Processing instructions). Standardně se zahajovací sekvence skládá ze dvou znaků ‘<?’ a ukončovací pouze ze znaku ‘>’. Otazník z ukončovací značky bloku příkazů PHP je v podstatě ještě součástí skriptu. Skriptovací engine PHP jej však ignoruje. Důvod je prostý — použití ukončovacího znaku ‘?>’ je mnohem přehlednější než použití samotného ‘>’. I přesto je však zachována plná kompatibilita se standardem SGML. Jazyk XML už rovnou počítá s ukončovacím znakem ‘?>’.

Tento způsob byl do PHP přidán kvůli některým editorům, které si neporadí s kratší formou zápisu. Jedná se většinou o WYSIWYG editory jako je třeba FrontPage.

Vidíme, že první, nejjednodušší, varianta, je pro zařazování skriptů do stránek ve většině případů nejlepší.

Kvůli kompatibilitě s některými HTML-editory byla do PHP přidána i podpora oddělovacích značek, které normálně používá ASP. Pokud je v konfiguračním souboru povolíme direktivou `asp_tags`, *můžeme nyní používat značky '<%...%>' a '<%=...%>'. První z nich je ekvivalentní zápisu '<?...?>'. Ve druhém způsobu zastupuje znak '=' příkaz* `echo`. *K vypsání aktuálního data pak můžeme použít*

```
<%= Date("d.m.Y")%>
```

ASP-značky bychom měli používat pouze tehdy, pokud nám již nic jiného nezbývá a musíme zajistit funkčnost PHP s některým ignorantským softwarem. Použití klasických značek PHP je mnohem elegantnější a plně vyhovuje standardům HTML, SGML a XML.

3.2 Základy syntaxe

Jako v každém programovacím jazyce, i v PHP je každý program (skript) složen ze sekvence příkazů. Jednotlivé příkazy probereme podrobně v dalších částech této kapitoly. Uvidíme, že jejich syntaxe se poměrně dost podobá jazyku C. Již nyní si však musíme říci, jak se od sebe jednotlivé příkazy oddělují. Existují dvě možnosti. Jednak můžeme každý příkaz uvést samostatně mezi znaky `<?` a `?>`:

```
<? $promenna = 20 ?>
<? echo $promenna ?>
```

Druhou možností je uvést více příkazů najednou. V tomto případě však musí být odděleny pomocí středníku:

```
<? $promenna = 20; echo $promenna ?>
```

Mezi příkazy můžeme vkládat libovolný počet mezer, tabelátorů a konců řádek. Toho s výhodou využijeme pro přehlednější zformátování zdrojového textu:

```
<?
    $promenna = 20;
    echo $promenna;
?>
```

Další podstatnou vlastností každého jazyka je, jak reaguje na malá a velká písmena. V PHP můžeme použít pro zápis příkazů a funkcí libovolná písmena — malá, velká či jejich kombinaci. Je tedy jedno, zda použijeme `echo`, `Echo` nebo `ECHO`. Na velikosti písmen záleží pouze v názvech proměnných. `$Promenna`, `$promenna`, `$PROMENNA` a `$proMENNA` jsou čtyři různé proměnné.

V názvech proměnných můžeme kromě libovolných písmen používat i podtržítko a číslice. První znak názvu proměnné však musí být pouze písmeno nebo podtržítko. Přípustnými názvy proměnných jsou tedy `$_ahoj`, `$Baba_Jaga`, `$citac0`. Nepřípustná je však například proměnná `$1x`, protože začíná číslicí.

Zajímavostí je, že v názvech proměnných lze používat i české znaky s diakritickými znaménky. Kromě znaků anglické abecedy, podtržítka a číslic můžeme použít libovolný znak, jehož kód je 127–255. Ve všech běžně používaných kódováních (ISO 8859-2, Windows CP 1250) do tohoto rozsahu spadnou i české znaky.

U delších skriptů, které používají ne úplně zřejmé programátorské obraty, je vhodné některá místa okomentovat pomocí komentářů. Komentář nijak neovlivňuje program, je pouze pomůckou pro ty, kteří po nás budou program číst. Komentáře bychom neměli podceňovat, protože se i nám může stát, že za půl roku budeme potřebovat oprášit nějaký skript a vůbec se v něm nevyznáme.

V PHP můžeme používat dva druhy komentářů. Pro kratší poznámky je vhodné použít komentář '`//`'. Vše od této dvojice znaků až do konce řádky, je ve skriptu ignorováno.

```
<?
    $promenna = 20;      // Přiřazení do proměnné
    echo $promenna;      // Vypsání obsahu proměnné
?>
```

Druhý druh komentáře nalezne uplatnění zejména pro delší poznámky. Vše mezi dvojicemi znaků '`/*`' a '`*/`' se považuje za komentář. Tento druh komentáře může zabírat i více řádek.

```
<?
    /* Následující skript je dílem Dr. Chytrolína.
       Nejprve cosi přiřadíme do proměnné, abychom
       to následně čímsi vytiskli. */

    $promenna = 20;
    echo $promenna;
?>
```

3.3 Proměnné

Proměnné slouží pro uchovávání hodnot, které potřebujeme v našich skriptech. Proměnné začínají znakem '`$`', po kterém následuje jméno proměnné. V PHP není potřeba proměnné deklarovat předem. Proměnná je deklarována okamžikem, kdy ji poprvé použijeme. Pokud do proměnné nic neuložíme přiřazovacím příkazem, má proměnná obsah prázdného řetězce.

> Před použitím proměnné bychom do ní vždy měli uložit nějakou hodnotu. Neměli bychom se spoléhat na to, že proměnná bude prázdná. Obsah proměnné lze totiž nastavit šikovným parametrem zadaným za URL. Např. URL končící na `?x=10` přiřadí do proměnné `$x` hodnotu 10. Náš skript pak nemusí pracovat správně, pokud předpokládáme, že v `$x` je prázdný řetězec.

V okamžiku přiřazení do proměnné se automaticky určí její typ. V PHP máme k dispozici pět typů:

`integer` celé číslo

`double` desetinné číslo

`string` znakový řetězec

`array` pole

`object` objekt

Jak vypadá automatické rozpoznání typů, si nyní ukážeme na jednoduchém příkladě:

```
$a = 5;     // $a je celé číslo
$b = 5.0;   // $b je desetinné číslo
$c = "5";   // $c je znakový řetězec
```

O typy proměnných se však většinou nemusíme starat, protože se podle potřeby automaticky konvertují na vhodný typ.

Typ integer

Celá čísla jsou v PHP reprezentována čtyřmi bajty a mají tedy rozsah od −2 147 483 648 do +2 147 483 647. Celočíselné konstanty zapisujeme jako čísla v desítkové soustavě případně doplněná o znaménko.

Číselné konstanty můžeme zapisovat i v šestnáctkové soustavě. Musíme před ně však přidat dvojici znaků `0x` nebo `0X`. Při zápisu šestnáctkových číslic můžeme používat malá i velká písmena 'A'–'F'.

```
$p = 0xff;              // v $p bude hodnota 255
$q = 0x12345678;        // v $q bude hodnota 305419896
```

V šestnáctkové soustavě akceptuje PHP pouze čísla v rozsahu od `0x0` do `0x7fffffff`. Ostatní šestnáctková čísla považuje za nulu.

PHP se nebrání ani soustavě osmičkové. Pokud na začátku číselné konstanty uvedeme 0, považuje se za číslo v osmičkové soustavě. Zápis `0333` je tedy osmičková reprezentace čísla 219.

Typ double

Desetinná čísla jsou v PHP reprezentována jako typ `double`. Ten je na většině platforem uložen v 8 bajtech. Jeho rozsah je přibližně od $-1{,}7 \cdot 10^{308}$ do $1{,}7 \cdot 10^{308}$. Nejmenší kladné číslo, které dokáže typ `double` pojmout je $3{,}4 \cdot 10^{-324}$. Při použití tohoto typu musíme počítat s tím, že k dispozici máme 15 platných číslic. Pokud pro naše výpočty potřebujeme větší přesnost, můžeme využít matematickou knihovnu BC, jejíž podpora je do PHP zahrnuta.

Číselné konstanty typu double se zapisují jako čísla s volitelným znaménkem a s desetinnou tečkou. Pokud chceme číslo zapsat v exponenciálním tvaru, následuje písmeno 'e' nebo 'E' a za ním je celé číslo vyjadřující exponent (např. `4.2e3` $= 4,2 \cdot 10^3$).

Typ string

Řetězce můžeme v PHP zapisovat do uvozovek nebo do apostrofů. Pokud potřebujeme uvozovky použít uvnitř řetězce, použijeme přepis pomocí tzv. escape sekvence '`\"`'. Pro zápis samotného zpětného lomítka použijeme '`\\`'. Pomocí

Sekvence	Význam
`\\`	'\' — zpětné lomítko
`\n`	nová řádka
`\r`	návrat vozíku
`\t`	tabulátor
`\"`	'"' — uvozovky
`\$`	'$' — znak dolaru

Tab. 3-1: Escape sekvence použitelné v řetězcových konstantách

zpětného lomítka můžeme do řetězce vložit i další užitečné znaky. Jejich přehled přináší tabulka 3-1.

Kromě escape sekvencí uvedených v tabulce můžeme znak zapsat přímo jeho ASCII-kódem. Escape sekvence '\«nnn»' zastupuje znak, jehož kód je v osmičkové soustavě *«nnn»*. Podobně zápis '\x«NN»' zastupuje znak, jehož kód v šestnáctkové soustavě je *«NN»*. Tak např. řetězec `"A\301\x41"` obsahuje tři písmena 'A'.

Escape sekvence můžeme používat pouze v řetězcích uzavřených do uvozovek. Stejně tak se můžeme na proměnné odkazovat pouze v řetězcích uzavřených v uvozovkách. Řetězce uzavřené do apostrofů interpretují celý svůj obsah pouze jako obyčejný text bez speciálního významu. Malá ukázka:

```
$var = 20;
echo "V proměnné \$var je uložena hodnota $var";
echo 'V proměnné \$var je uložena hodnota $var';
```

První příkaz `echo` zobrazí následující text:

```
V proměnné $var je uložena hodnota 20
```

Ve výsledku druhého příkazu nebude zobrazen obsah proměnné, protože řetězce uvnitř apostrofů nejsou kontrolovány na obsah proměnných a escape sekvencí:

```
V proměnné \$var je uložena hodnota $var
```

Pokud uložíme řetězec do proměnné, můžeme s ním pracovat dvěma základními způsoby. Jednak lze s řetězcem pracovat jako s celkem — použijeme obyčejnou proměnnou. Pokud chceme přistupovat k jednotlivým znakům řetězce, napíšeme pozici požadovaného znaku v řetězci do hranatých závorek za jméno řetězcové proměnné. Znaky jsou přitom číslovány od nuly.

```
$text = "Ukázkový řetězec";
echo $text;                     // Vytiskne Ukázkový řetězec
echo $text[2];                  // Vytiskne á
```

3.4 Pole

Pole je speciální datová struktura, ve které jedna proměnná může obsahovat několik hodnot. Tyto hodnoty jsou přístupné pomocí indexu. Index prvku pole se zapisuje do hranatých závorek:

```
$a[0] = "Jablka";
$a[1] = "Hrušky";
$a[2] = "Švestky";
```

Nyní máme k dispozici pole `$a`, které obsahuje tři prvky. Jednotlivé prvky pole mohou být libovolného typu, takže následující skript není žádným problémem:

```
$a[0] = "Jablka";
$a[1] = 12;
$a[2] = 3.1415926535;
```

V PHP můžeme jako index pole používat i znakové řetězce. Získáme pak tzv. asociativní pole:

```
$adresy["Jirka"] = "xkosj@vse.cz";
$adresy["Karel"] = "carlos@mbox.freemail.cz";
```

Inicializace pole

Pole můžeme inicializovat tak, že přiřadíme hodnotu jeho jednotlivým prvkům. Existují však i pohodlnější způsoby. Pokud u pole použijeme prázdný index, hodnota se automaticky uloží na místo prvního volného indexu. Indexy přitom začínají od nuly. Následující skript je tedy ekvivalentní naší výše uvedené ukázce:

```
$a[] = "Jablka";
$a[] = "Hrušky";
$a[] = "Švestky";
```

Pole však lze inicializovat ještě úspornějším a pohodlnějším zápisem. Využijeme k tomu speciální příkaz `Array`:

```
$a = Array("Jablka", "Hrušky", "Švestky");
```

Pokud chceme, aby bylo pole inicializováno od jiného indexu, použijeme modifikovaný příkaz `array`:

```
$dny = Array(1=>"Po", "Út", "St", "Čt", "Pá", "So", "Ne");
```

Nyní bude mít první prvek pole (`Po`) index 1. Dopadne to tedy stejně, jako kdybychom napsali:

```
$dny[1] = "Po";
$dny[2] = "Út";
$dny[3] = "St";
$dny[4] = "Čt";
$dny[5] = "Pá";
$dny[6] = "So";
$dny[7] = "Ne";
```

Kterou variantu inicializace použijete, závisí jen na vás. I zde však platí, že co není v hlavě, musí být v rychlých prstech.

Pokud inicializujeme pole pomocí `Array`, můžeme index přiřadit zvlášť každé hodnotě. Operátor `=>` můžeme použít u libovolného prvku pole. Nejčastěji se však použije pouze u prvního prvku nebo u všech prvků. Pokud chceme vytvořit asociativní pole pro převod anglických zkratek dne v týdnu na české, použijeme následující inicializaci:

```
$day2Den = Array("Mon" => "Po", "Tue" => "Út", "Wed" => "St",
                 "Thu" => "Čt", "Fri" => "Pá", "Sat" => "So",
                 "Sun" => "Ne");

echo $day2Den["Fri"];       // Vypíšeme českou zkratku pro pátek
```

Funkce pro práci s polem

Pro práci s poli je v PHP mnoho funkcí. Jejich úplný výčet nalezneme v referenčním přehledu funkcí. Nyní se seznámíme alespoň s těmi nejdůležitějšími.

Pro zjištění počtu prvků pole slouží funkce `Count(`*«pole»*`)`. Pokud máme pole vytvořeno standardní cestou, a je tudíž indexováno od nuly, můžeme pro zpracování všech prvků pole `$x` použít následující kód:

```
for($i=0; $i<Count($x); $i++):
    echo $x[$i];
endfor;
```

Pokud používáme asociativní pole, nebo pole s indexem, který nezačíná od nuly, můžeme pro zpracování celého pole použít funkce `Reset()`, `Current()`, `Next()` a `Key()`. Pro každé pole totiž v PHP existuje ukazatel, který ukazuje na některý prvek pole. Pomocí `Reset(`*«pole»*`)` přesuneme tento ukazatel na první prvek pole. Funkce `Current(`*«pole»*`)` vrací hodnotu prvku pole, na který ukazuje ukazatel. Podobně funguje i funkce `Next(`*«pole»*`)`. Rozdíl je v tom, že se nejprve ukazatel přesune na další prvek pole a teprve poté se vrátí hodnota prvku pole.

Funkce `Key(«pole»)` vrací index prvku pole, na který ukazuje ukazatel. Vypsání celého obsahu pole lze realizovat např. takto:

```
Reset($x);
while(Current($x)):
    echo "Index: ".Key($x)."\n";
    echo "Hodnota: ".Current($x)."\n";
    Next($x);
endwhile;
```

Velice užitečnou funkcí při práci s poli je funkce `list()`. Funkce může mít libovolný počet parametrů a slouží k načtení prvků pole do proměnných. Pokud máme např. pole `$x`, které má čtyři prvky, můžeme tyto prvky uložit do proměnných `$p1`, `$p2`, `$p3` a `$p4` pomocí následujícího příkazu:

```
list($p1, $p2, $p3, $p4) = $x;
```

Pokud nás zajímá jen první a čtvrtý prvek, můžeme některé parametry vynechat:

```
list($p1,,,$p4) = $x;
```

Vícerozměrná pole

V PHP můžeme používat i vícerozměrná pole. Pro každý rozměr stačí do hranatých závorek přidat další index. Např. dvojrozměrná pole se často používají pro uchování různých matic. Pro přístup k prvku ve třetím řádku a pátém sloupci dvojrozměrného pole `$mat` můžeme použít zápis `$mat[3][5]`.

Dvojrozměrné pole se přitom chová jako jednorozměrné pole, jehož prvky jsou pole. Tato pole mohou mít dokonce různý počet prvků. Zápis `$mat[3]` tedy vrací pole, které obsahuje všechny prvky třetího řádku naší pomyslné matice.

Vícerozměrná pole můžeme inicializovat pomocí vnořených příkazů `array`. K vytvoření jednotkové matice 3×3 použijeme příkaz:

```
$I = array(array(1, 0, 0),
           array(0, 1, 0),
           array(0, 0, 1));
```

3.5 Přetypování proměnných

Ve většině případů není potřeba měnit typ proměnných — o vše se postará PHP samo. Pokud například sčítáme čísla, z nichž je jedno `double`, všechna ostatní se převedou na typ `double` a výsledek bude také typu `double`. Pokud jsou všechna čísla `integer`, bude také výsledek `integer`.

Zjištění typu proměnné

Ke zjištění typu proměnné můžeme použít funkci `GetType(`*«proměnná»*`)`. Ta vrací jméno typu *«proměnné»*. Pokud chceme zjistit, zda má proměnná nějaký konkrétní typ, můžeme použít funkce `Is_Integer()`, `Is_Double()`, `Is_String()`, `Is_Array()` a `Is_Object()`.

Změna typu proměnné

Pro změnu typu proměnné použijeme funkci `SetType(`*«proměnná»*`, `*«typ»*`)`. Pokud tedy chceme z číselné proměnné `$N` vytvořit proměnnou `$N` typu `string`, použijeme volání `SetType($N, "string")`.

Přetypování

Funkce `SetType()` změní typ proměnné. Pokud však potřebujeme typ nějaké proměnné změnit pouze jednorázově při jejím použití ve výrazu, použijeme přetypování. Přetypování provedeme tak, že před proměnnou napíšeme do kulatých závorek typ, na který ji chceme převést. Malý příklad:

```
$i = 10;                // $i je typu integer
$x = (double) $i;       // $x je typu double
```

Při přetypování můžeme použít `(int)` a `(integer)` pro celá čísla; `(double)`, `(real)` a `(float)` pro desetinná; `(string)` pro textové řetězce; `(array)` pro pole a `(object)` pro objekty.

Uvnitř závorek s typem můžeme zapisovat libovolný počet mezer, následující zápisy jsou tedy identické:

```
$x = (double) $i;
$x = ( double ) $i;
```

Konverze řetězců na čísla

Pokud použijeme řetězec v místech, kde je očekáván číselný výraz, provede se automatická konverze na číselný typ. Otázkou však je, zda výsledkem konverze bude typ `integer` nebo `double`.

Výsledek bude číslo typu `double`, pokud řetězec obsahuje některý ze znaků ‘`.`’, ‘`E`’ a ‘`e`’. Jinak bude výsledek typu `integer`.

Při zjišťování číselné hodnoty řetězce se v úvahu bere jeho počáteční část. Pokud obsahuje číslo, je výsledkem konverze typů toto číslo. Pokud řetězec nezačíná číslem, výsledkem konverze bude 0.

Jak vypadá číslo? Začíná nepovinným znaménkem, za kterým následuje libovolný počet cifer, které mohou obsahovat jednu desetinnou tečku (‘`.`’). Poté

může následovat znak ‘e’ nebo ‘E’ doplněný o celé číslo vyjadřující exponent. Exponent může být i záporný. Několik praktických ukázek:

```
                                 // Typ výsledku         Hodnota výsledku
$k = 1 + "13.5";                 // double               14.5
$k = 1 + "-1.3e3";               // double               -1299
$k = 1 + "bob-1.3e3";            // double               1
$k = 1 + "bobík3";               // integer              1
$k = 1 + "20 malých prasat";     // integer              21
$k = 1 + "20 malých prasátek";   // double               21
```

Poslední ukázka dává výsledný typ `double`, protože řetězec „20 malých prasátek“ obsahuje znak ‘e’. Výsledek konverze tedy bude typu `double`.

3.6 Výrazy

Výrazy jsou základním kamenem většiny moderních programovacích jazyků. Výraz je v podstatě cokoliv, co má nějakou hodnotu. Nejjednodušším výrazem je tedy konstanta či proměnná, jako ‘12’ nebo ‘`$delta`’.

Výrazem je však i volání funkce. Například funkce `Rand()` vrací náhodné číslo. Její volání je tedy také výrazem, protože funkce vrací hodnotu — nějaké číslo. Nutno ovšem podotknout, že v PHP nemusí být hodnotou výrazu jen čísla. Výraz může mít libovolný typ, který podporuje PHP (`integer`, `double`, `string`, `array` a `object`).

Podívejme se nyní na složitější případ. Dejme tomu, že máme následující přiřazení `$x = 10`. Do proměnné `$x` jsme přiřadili hodnotu 10. Výrazem je v tomto případě číslo 10. Výraz je však i `$x` — jeho hodnota je po přiřazení rovněž 10. V levé části přiřazení může být jen proměnná. Přiřazení má tedy obvykle následující obecný tvar:

«proměnná» = *«výraz»*

Zajímavostí je, že v PHP je celé přiřazení chápáno jako výraz. Jeho hodnota je shodná s hodnotou výrazu v pravé části přiřazení. Toho lze využít v mnoha případech. Pokud chceme vynulovat proměnné `$a` a `$b`, můžeme použít následující kód:

```
$a = 0; $b = 0;
```

Protože máme po prvním přiřazení v `$a` hodnotu nula, je možný i zápis:

```
$a = 0; $b = $a;
```

Protože však zápis `$a = 0` je kromě přiřazení i výraz, můžeme použít následující:

```
$b = ($a = 0);
```

Protože jsou přiřazení vyhodnocována zprava doleva, můžeme závorky vynechat a psát:

```
$b = $a = 0;
```

Výraz získáme i tak, že zde uvedené elementární výrazy poskládáme dohromady pomocí různých operátorů. Pojďme se podívat, jaké operátory máme k dispozici v jazyce PHP.

Matematické operátory

PHP nám nabízí standardní operátory pro sčítání ('+'), odčítání ('-'), násobení ('*') a dělení ('/'). Přitom operátory pro násobení a dělení mají větší prioritu než sčítání a odečítání. Pokud nám tyto priority nevyhovují, můžeme je změnit uzávorkováním:

```
echo 2+3*5;              // Vytiskne 17
echo (2+3)*5;            // Vytiskne 25
```

Dalším z matematických operátorů je '%'. Ten slouží k určení zbytku po dělení:

```
$zbytek = 10%3;          // Zbytek by měl být 1
echo "Zbytek po dělení 10:3 = $zbytek";
```

Velice častou operací v mnoha skriptech je přičtení nějakého výrazu k proměnné. To můžeme provést jednoduše příkazem `$x = $x +` *«výraz»*. Tento zápis je však poněkud zdlouhavý, a proto PHP nabízí kratší cestu: `$x +=` *«výraz»*. Oba dva příkazy přitom mají stejný efekt — do proměnné `$x` přičtou *«výraz»*. Kromě operátoru '+=' máme k dispozici i další: '-=', '*=', '/=' a '%='.

```
$x += 16;                // $x = $x + 16
$x -= $a+$b;             // $x = $x - ($a+$b)
$x /= 2;                 // $x = $x / 2;
$x *= $y;                // $x = $x * $y;
$a %= $b;                // $a = $a % $b;
```

Vidíme, že PHP myslí na programátora a snaží se mu ušetřit zbytečné psaní. Dalším příkladem efektivnosti syntaxe jazyka jsou operátory inkrementace ('++') a dekrementace ('--'). Tyto operátory slouží ke zvýšení, resp. snížení obsahu proměnné o jedna. Zápis `++$x` je ekvivalentní zápisu `$x = $x + 1`. Obdobně `--$i` je totéž jako `$i = $i - 1`.

Operátory inkrementace a dekrementace však můžeme psát i za proměnnou. Jejich chování je však poněkud odlišné — vše souvisí s tím, že i zápisy `++$i` a `$i++` jsou výrazy. První výraz pracuje tak, že zvýší obsah proměnné `$i` o jedničku a takto získanou hodnotu vrátí jako svůj výsledek. Oproti tomu hodnota výrazu `$i++` je shodná s hodnotou proměnné `$i`. Po vrácení této hodnoty se proměnná `$i` zvýší o jedna.

Výrazu `++$i` se říká preinkrementace, protože je před vrácením hodnoty výrazu provedena inkrementace proměnné. Postinkrementace je pak jméno pro výraz typu `$i++`, kdy je nejprve vrácena hodnota a teprve poté je zvýšen obsah proměnné. Obdobně vše funguje i pro dekrementaci. Pro lepší pochopení malá ukázka:

```
$i = 10;                // $i = 10
$a = $i++;              // $a = 10, $i = 11
$b = ++$i;              // $b = 12, $i = 12
$c = $a + $i--;         // $c = 22, $i = 11
```

Operátory pro manipulaci s bity čísla

Dnešní osobní počítače jsou digitální zařízení, a proto jsou i čísla vnitřně reprezentována posloupností nul a jedniček — binárním (dvojkovým) číslem. PHP nabízí několik operátorů, které umožňují pracovat s touto binární reprezentací. Pro správné pochopení činnosti některých operátorů je dobré si uvědomit, že PHP pracuje s čísly o velikosti 32 bitů.

Prvním z operátorů je logický součin zapisovaný pomocí operátoru '`&`'. Jeho výsledkem je číslo, ve kterém jsou jedničky na těch binárních cifrách, kde byly jedničky v obou dvou operandech. Na ostatních místech jsou nuly.

```
$x      = 00110011001100110101010101010101
$y      = 01010101010101010011001100110011
------------------------------------------
$x & $y = 00010001000100010001000100010001
```

Dalším operátorem je logický součet '`|`'. Ve výsledku jsou jedničky na místech, kde byla jednička alespoň v jednom z operandů.

```
$x      = 00110011001100110101010101010101
$y      = 01010101010101010011001100110011
------------------------------------------
$x | $y = 01110111011101110111011101110111
```

Dalším binárním operátorem je nonekvivalence '`^`'. Jeho výsledkem je první číslo, ve kterém jsou změněny ty bity, které jsou v druhém operandu jedničkové. Programátoři tento operátor znají obvykle pod názvem `xor`.

```
$x      = 00110011001100110101010101010101
$y      = 01010101010101010011001100110011
------------------------------------------
$x ^ $y = 01100110011001100110011001100110
```

Další operátor, na který se podíváme, je unární. To znamená, že má pouze jeden operand. Jeden operand má logická negace ‘~’. Vrací výsledek, ve kterém jsou prohozené všechny hodnoty bitů.

```
$x      = 00110011001100110101010101010101
-----------------------------------------
~$x     = 11001100110011001010101010101010
```

Pokud si chcete s těmito operátory hrát, budou se vám hodit funkce `BinDec()` a `DecBin()`. Ty převádějí čísla mezi dvojkovou a desítkovou soustavou. K otestování našeho prvního příkladu můžeme použít následující příkaz:

```
echo DecBin(BinDec("00110011001100110101010101010101") &
            BinDec("01010101010101010011001100110011"));
```

Další operátory, které manipulují přímo s jednotlivými bity čísla, jsou operace pro bitový posuv. Tyto operátory umožňují posunout bity v čísle o zadaný počet míst vlevo nebo vpravo. Pro posun doleva slouží operátor ‘<<’ a pro posun doprava ‘>>’. Malý příklad:

```
$x      = 00110011001100110101010101010101
-----------------------------------------
$x >> 3 = 00000110011001100110101010101010

$x      = 00110011001100110101010101010101
-----------------------------------------
$x << 1 = 01100110011001101010101010101010
```

Vidíme, že při posunu se automaticky na okraj čísla doplňují nuly, podle toho na jakou stranu s číslem posouváme. Počet míst k posunutí může být libovolný výraz, který lze převést na celé číslo. Vzhledem k tomu, že celá čísla jsou v PHP 32bitová, nemá cenu provádět posun o více než 31 míst.

PHP pracuje s celými čísly v rozsahu od −2 147 483 648 *do* +2 147 483 647. *Nejvyšší bit čísla nese informaci o znaménku — je nastaven na jedničku, pokud je číslo záporné. Pokud při našich manipulacích nastavíme nejvyšší bit a necháme číslo vypsat, dostaneme záporné desítkové číslo.*

Pro všechny bitové operátory s výjimkou ‘~’ existují i jejich varianty spojené s přiřazovacím příkazem: ‘<<=’, ‘>>=’, ‘&=’, ‘|=’ a ‘^=’.

Logické výrazy a operátory

Velice často potřebujeme zkonstruovat výraz, jehož výsledek je pravda nebo nepravda. Takovéto výrazy se používají jako podmínka v cyklech či podmíněných příkazech. PHP nemá speciální datový typ určený pro logické proměnné. Místo toho je použit celočíselný typ. Pokud je hodnota nějakého výrazu nenulová, považuje se tento výraz za pravdivý (hodnota `true`). Pokud je hodnota výrazu nulová, považuje se výraz za nepravdivý (hodnota `false`).

Narozdíl od jiných jazyků jsou záporná čísla v PHP považovaná za pravdivé výrazy.

Pokud v roli logického výrazu použijeme řetězec, považuje se za nepravdivý, pokud je prázdný nebo jde o řetězec `"0"`. Ostatní řetězce jsou považovány za pravdivé výrazy. Pokud použijeme nějaký složitější typ — pole nebo objekt, bude chápán jako `true` v případě, že není prázdný.

Pro snazší práci máme v PHP definovány dvě konstanty `true` a `false`. `true` odpovídá pravdivé hodnotě a je reprezentováno jako celé číslo 1. Oproti tomu `false` odpovídá nepravdě a jde o prázdný řetězec.

Typickými operátory, které se používají pro získání logického výsledku, jsou relační operátory. Relační operátory slouží k porovnávání hodnot dvou výrazů.

Pokud chceme zjistit, zda mají dva výrazy stejnou hodnotu, použijeme operátor rovnosti ‘`==`’:

```
if ($a==$b) echo "Proměnné 'a' a 'b' obsahují shodnou hodnotu.";
```

V operátoru rovnosti jsou použity dva znaky rovnítka ihned za sebou. Častou chybou je používaní pouze jednoho rovnítka. Jedno rovnítko je přiřazovací příkaz, ale zároveň funguje i jako výraz — interpret PHP tedy nezjistí žádnou syntaktickou chybu a náš skript nepracuje tak, jak by měl. Toto varování by si měli vzít k srdci zejména programátoři zvyklí na jazyk Pascal. Vím, co říkám.

Pokud chceme otestovat nerovnost dvou výrazů, s výhodou použijeme jeden z operátorů ‘`!=`’ nebo ‘`<>`’. Oba dva se chovají zcela shodně, mají pouze odlišnou syntaxi.

```
if ($a!=$b) echo "Proměnné 'a' a 'b' neobsahují shodnou hodnotu.";
```

K dispozici máme samozřejmě i operátory ostré a neostré nerovnosti:

```
if ($a>$b) echo "Proměnná 'a' je větší než proměnná 'b'.";
if ($a>=$b) echo "Proměnná 'a' je větší nebo rovna proměnné 'b'.";
if ($a<$b) echo "Proměnná 'a' je menší než proměnná 'b'.";
if ($a<=$b) echo "Proměnná 'a' je menší nebo rovna proměnné 'b'.";
```

Jednotlivé logické výrazy můžeme skládat dohromady pomocí logických spojek. K dispozici máme zejména operátor logického součinu '&&' a logického součtu '||'.

Logický součin pracuje jako spojka „a zároveň“. Výsledkem logického součinu je pravdivá hodnota, pokud jsou oba spojované výrazy pravdivé.

```
if ($a>=10 && $a<=20) echo "Proměnná 'a' je někdy mezi 10 a 20.";
```

Výsledkem logického součtu je pravdivá hodnota, pokud je alespoň jeden z operandů pravdivý:

```
if ($a==10 || $a==20) echo "Proměnná 'a' má hodnotu 10 nebo 20.";
```

Jako synonymum k operátoru '&&' můžeme použít AND. Podobně pro '||' existuje OR. Tyto operátory se chovají stejně, mají však nižší prioritu. Navíc můžeme použít operátor XOR, který vrací pravdivou hodnotu, pokud je právě jeden z operandů pravdivý výraz.

Pokud chceme nějaký logický výraz znegovat, umístíme před něj operátor '!'.

```
if (!$spojeni) echo "Bez spojení není velení.";
```

Operátor pro spojování řetězců

Pro spojování řetězců slouží speciální operátor '.'. Funguje tak, že své operandy převede nejprve na znakové řetězce a pak je spojí v jeden řetězec.

```
echo "Zbytek po dělení 10:3 je ".(10 % 3);
```

Okolo operátoru můžeme napsat mezer, kolik chceme, a zpřehlednit tak program:

```
echo "Zbytek po dělení 10:3 je " . (10 % 3);
```

Pokud potřebujeme k nějaké proměnné připojit řetězec, můžeme samozřejmě použít následující postup:

```
$s = $s . "a něco málo na konec řetězce";
```

Pro tyto případy však s výhodou využijeme operátor '.=':

```
$s .= "a něco málo na konec řetězce";
```

Používání operátorů jako '.=' nebo '+=' nejen zkracuje výsledný kód, ale i při provádění je podstatně rychlejší než klasický zápis, ve kterém se proměnná opakuje.

Podmíněný operátor

Podmíněný operátor je jediným ternárním operátorem[2] v jazyce PHP. Jeho syntaxe je následující:

«výraz1» ? *«výraz2»* : *«výraz3»*

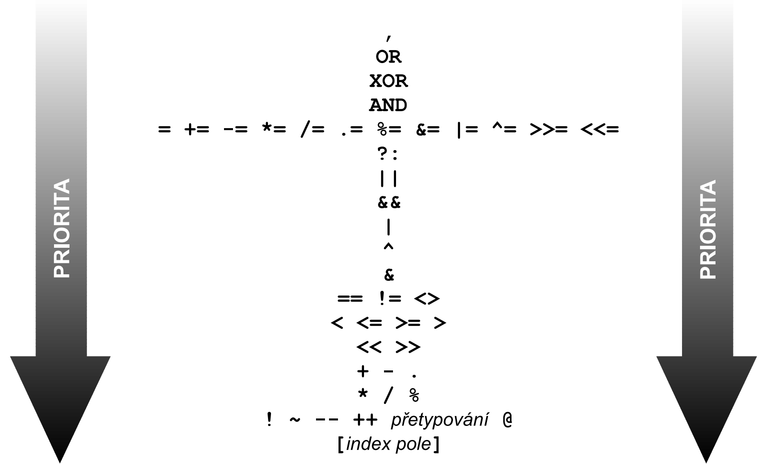
Pokud je *«výraz1»* pravdivý, je výsledkem *«výraz2»*. V opačném případě je výsledkem *«výraz3»*. Použití si ukážeme na několika jednoduchých příkladech:

```
$min = $x<$y ? $x : $y;    // Do proměnné $min uložíme minimum z $x a $y
$abs = $x<0 ? -$x : $x;    // Do proměnné $abs uložíme absolutní hodnotu $x
```

Priorita operátorů

Z hodin matematiky každý víme, že např. násobení má přednost před sčítáním. Platí tedy, že $3 + 5 \cdot 4 = 23$. V řeči programovacích jazyků bychom řekli, že operátor sčítání '+' má nižší prioritu než násobení '·'. V PHP má každý operátor definovanou svoji prioritu, takže je vždy jednoznačné, která operace se provede dříve. Přehled priorit jednotlivých operátorů přináší obrázek 3-1. Operátory uvedené níž mají vyšší prioritu.

Pokud se sejdou operátory se stejnou prioritou, provádějí se postupně zleva doprava. Výjimkou jsou operátory na posledních dvou řádcích obrázku. U nich probíhá vyhodnocování zprava doleva. To je způsobeno tím, že se jedná o unární operátory, u nichž je pouze jeden operand, který se píše až za operátor.



Obr. 3-1: Priorita operátorů stoupá směrem dolů

[2] Ternární operátor je operátor, který má tři operandy.

Vidíme, že např. index pole má nejvyšší prioritu. Pokud tedy napíšeme `++$a[0]`, PHP ví, že chceme zvýšit první prvek pole `$a`, a ne zvýšit proměnnou `$a` a poté se koukat do nějakého pole.

Pokud nám z nějakého důvodu priority nevyhovují, můžeme je změnit pomocí závorek. Závorky můžeme použít i v případech, kdy si nejsme jisti, který operátor má vyšší prioritu. Víme-li, že operátor rovnosti '`==`' má vyšší prioritu než logický součet '`||`', můžeme psát:

```
if ($a==10 || $a==20) echo "Proměnná 'a' má hodnotu 10 nebo 20.";
```

Pokud si však nejsme jisti, nic nezkazíme použitím závorek, které jasně řeknou, jak jsme to vlastně mysleli:

```
if (($a==10) || ($a==20)) echo "Proměnná 'a' má hodnotu 10 nebo 20.";
```

3.7 Příkazy pro větvení programu

Snad každý algoritmus, kromě těch zcela triviálních, musí reagovat na okolní podmínky. Podle nich se pak různě větví, aby ošetřil všechny možnosti, které mohou nastat.

PHP nám nabízí dva příkazy, které umožňují provádění části skriptu podmíněně — na základě splnění určité podmínky. Tyto příkazy jsou `if` a `switch`. Podívejme se teď podrobně na jejich syntaxi.

Příkaz if

Příkaz `if` slouží k podmíněnému provedení příkazu, pokud je splněna určitá podmínka. Podmínku musíme zadat jako výraz, který vrací logickou hodnotu:

```
if («výraz») «příkaz»;
```

Následující jednoduchá ukázka uloží do proměnné `$podil` podíl proměnných `$a` a `$b`. Podíl se však uloží pouze v případě, že `$b` není nulové. Dělení nulou by jinak vyvolalo chybu:

```
if ($b!=0) $podil = $a/$b;
```

V PHP jsou logické hodnoty reprezentovány jako obyčejné číslo. Nula přitom odpovídá hodnotě `false`. Toho můžeme využít a podmínku zapsat zkráceně:

```
if ($b) $podil = $a/$b;
```

Pokud chceme podmíněně provést více příkazů, uzavřeme je do složených závorek.

```
if ($a<$b)
{
    $p = $a;
    $a = $b;
    $b = $p;
}
```

Podmíněně můžeme pomocí PHP samozřejmě vkládat i HTML-kód:

```
<? if ($pocetPristupu==0)
   {
       echo "<H1>Vítejte na našem serveru. Jste tu poprvé.</H1>";
       echo "Můžete se <A HREF=\"register.php\">zaregistrovat</A>.";
   }
?>
```

Pokud potřebujeme podmíněně vložit delší HTML-kód, je používaní příkazu `echo` těžkopádné. Využijeme tedy toho, že PHP se s HTML může krásně prolínat.

```
<? if ($pocetPristupu==0)
   { ?>
       <H1>Vítejte na našem serveru. Jste tu poprvé.</H1>
       Můžete se <A HREF="register.php">zaregistrovat</A>
<? } ?>
```

Jak vidíme, zápis není moc přehledný a ve složitějších konstrukcích snadno ztratíme přehled o tom, k čemu která složená závorka patří. V těchto případech se hodí druhá možná syntaxe příkazu `if`:

```
if («výraz»):
    «příkaz1»;
    «příkaz2»;
    ...
endif;
```

Naši poslední ukázku tedy můžeme přepsat do přehlednější formy:

```
<? if ($pocetPristupu==0): ?>
       <H1>Vítejte na našem serveru. Jste tu poprvé.</H1>
       Můžete se <A HREF="register.php">zaregistrovat</A>
<? endif ?>
```

Při používaní `if`...`endif` nesmíme zapomenout na dvojtečku, která se uvádí za podmínkou.

Příkaz if–else

Častým případem je, že v závislosti na nějaké podmínce potřebujeme provést dvě různé činnosti. Jednu, pokud je podmínka splněna, a tu druhou v případě nesplnění podmínky. V těchto případech nalezne uplatnění příkaz `if` doplněný o část `else`. Příkaz uvedený za `else` se provede v případě nesplnění podmínky.

```
if («výraz»)
    «příkaz1»;
else
    «příkaz2»;
```

Pokud je *«výraz»* pravdivý, provede se *«příkaz1»*. V opačném případě se provede *«příkaz2»*. Pokud chceme podmíněně provést více příkazů, opět je můžeme sloučit pomocí složených závorek:

```
if ($b!=0)
{
    $podil = $a/$b;
    echo "$a/$b = $podil";
}
else
{
   echo "Nemohu dělit nulou.";
   echo "To bys mohl vědět.";
}
```

Opět máme k dispozici alternativní syntaxi:

```
if («výraz»):
    «příkaz1-1»;
    «příkaz1-2»;
    ...
else:
    «příkaz2-1»;
    «příkaz2-2»;
    ...
endif;
```

Dvojtečku v tomto případě musíme uvést i za klíčovým slovem `else`.

Pokud do sebe vnořujeme více příkazů `if–else`, váže se `else` vždy k nejbližšímu příkazu `if`:

```
if («výraz1»)
if («výraz2») «příkaz1»; else «příkaz2»;
```

je ekvivalentní se zápisem:

```
if («výraz1»):
    if («výraz2»):
        «příkaz1»;
    else:
        «příkaz2»;
    endif;
endif;
```

Příkaz if–elseif–else

Klíčové slovo `elseif` dále rozšiřuje možnosti příkazu `if`. Příkaz uvedený za `elseif` se provede v případě, že není splněna podmínka pro `if` a zároveň je splněna podmínka za `elseif`. Částí `elseif` můžeme použít i několik za sebou. Nakonec můžeme použít i část `else`, která se provede pouze v případě, že není splněna ani jedna z předchozích podmínek:

```
if («výraz1»)
    «příkaz1»;
elseif («výraz2»)
    «příkaz2»;
elseif («výraz3»)
    «příkaz3»;
...
else
    «příkazN»;
```

Malá ukázka:

```
if ($a<$b)
    echo "a je menší než b";
elseif ($a==$b)
    echo "a je rovno b";
else
    echo "a je větší než b";
```

Pokud v jedné větvi potřebujeme provést více příkazů, opět je můžeme sdružovat pomocí složených závorek. K dispozici máme i alternativní syntaxi:

```
if («výraz1»):
    «příkaz1-1»;
    «příkaz1-2»;
    ...
elseif («výraz2»):
    «příkaz2-1»;
```

```
        «příkaz2-2»;
        ...
    elseif («výraz3»):
        «příkaz3-1»;
        «příkaz3-2»;
        ...
    ...
    else:
        «příkazN-1»;
        «příkazN-2»;
        ...
    endif;
```

Příkaz switch

Velice často potřebujeme na základě hodnoty jednoho výrazu provést jednu z několika větví skriptu. Pro tyto případy se optimálně hodí příkaz `switch`. Jeho syntaxe je následující:

```
switch («výraz»)
{
    case «hodnota1»:
        «příkaz1-1»;
        «příkaz1-2»;
        ...
        break;
    case «hodnota2»:
        «příkaz2-1»;
        «příkaz2-2»;
        ...
        break;
    ...
    default:
        «příkazX-1»;
        «příkazX-2»;
        ...
}
```

Jak přesně pracuje příkaz `switch`? Nejprve je vyhodnocen *«výraz»*. Pak jsou postupně procházeny hodnoty uvedené za klíčovým slovem `case`, dokud se nenalezne hodnota shodná s hodnotou *«výrazu»*. Následně jsou vykonávány příkazy, dokud se nenarazí na příkaz `break` nebo na konec příkazu `switch`.

Jednotlivé varianty příkazu `switch` nesmíme zapomenout ukončit příkazem `break`. Pokud na `break` zapomeneme, provedou se i příkazy, které jsme provést nechtěli a které patří k jiným větvím podmíněného příkazu.

Uvedenou vlastnost však můžeme šikovně využít. Pokud chceme, aby se jedna větev příkazů provedla pro více různých hodnot výrazu, můžeme za sebou uvést několik částí `case`:

```
switch ($pismeno)
{
    case "A":
    case "a":
        echo "Malé velké, stejně to je 'A'.";
        break;
    case "B":
    case "b":
        echo "Malé velké, stejně to je 'B'.";
        break;
    default:
        echo "Jiná písmena nepoznám.";
}
```

Jak jste již vytušili, příkazy uvedené za `default` se provedou v případě, že *«výrazu»* nevyhoví ani jedna z hodnot uvedených za slovem `case`. Použití `default` není povinné, ale dobrý program by měl reagovat na všechny možné situace, a proto jeho použití nezbývá než vřele doporučit.

K příkazu `switch` samozřejmě existuje alternativní syntaxe:

```
switch («výraz»):
    case «hodnota1»:
        «příkaz1-1»;
        «příkaz1-2»;
        ...
        break;
    case «hodnota2»:
        «příkaz2-1»;
        «příkaz2-2»;
        ...
        break;
    ...
    default:
```

```
        «příkazX-1»;
        «příkazX-2»;
        ...
endswitch;
```

3.8 Příkazy cyklu

Často potřebujeme některou část skriptu provést opakovaně. Pro tyto případy nám PHP nabízí poměrně širokou škálu příkazů cyklu. K dispozici jsou cykly `while`, `do–while` a `for`. Všechny příkazy mají syntaxi převzatou z jazyka C. K dispozici je ovšem i alternativní syntaxe, která je vhodnější pro kombinování skriptů s HTML. Podívejme se však na jednotlivé příkazy cyklu podrobněji.

Příkaz while

Příkaz `while` je nejjednodušším příkazem cyklu. Zadaný příkaz provádí, dokud platí určitá podmínka. Často se mu říká cyklus s podmínkou na začátku.

```
while («výraz») «příkaz»;
```

Větší množství příkazů, které chceme provádět v cyklu, můžeme opět sdružit pomocí složených závorek:

```
$i = 0;
while ($i<100)
{
    echo "$i^2=".($i*$i)."<BR>";
    $i++;
}
```

K dispozici je i alternativní syntaxe:

```
while («výraz»):
    «příkaz1»;
    «příkaz2»;
    ...
endwhile;
```

Náš skript, který vypisuje tabulku druhých mocnin, můžeme tedy zapsat také takto:

```
$i = 0;
while ($i<100):
```

```
    echo "$i^2=".($i*$i)."<BR>";
    $i++;
endwhile;
```

Cyklus do–while

Tento cyklus pracuje podobně jako cyklus `while`, podmínka je však umístěna až na konci cyklu. Rozdíl je v tom, že příkazy v těle cyklu `do–while` jsou vykonány alespoň jednou. Oproti tomu tělo cyklu `while` nemusí být provedeno ani jednou, pokud hned na začátku není splněna podmínka pro provádění cyklu.

```
do
    «příkaz»;
while («výraz»);
```

Naši ukázku cyklu `while` můžeme zapsat i pomocí `do–while`:

```
$i = 0;
do
{
    echo "$i^2=".($i*$i)."<BR>";
    $i++;
} while ($i<100);
```

K příkazu `do–while` neexistuje jeho ekvivalent s alternativní syntaxí.

Cyklus for

Cyklus `for` je ze všech příkazů cyklu asi nejvýkonnější a také nejsložitější. Jeho syntaxe je:

```
for («výraz1»;«výraz2»;«výraz3») «příkaz»;
```

«výraz1» je vyhodnocen před začátkem provádění cyklu. Poté je vyhodnocen *«výraz2»*. Pokud je jeho hodnota `true`, provede se tělo cyklu. Na konci každého průchodu tělem cyklu se vyhodnotí *«výraz3»*. Tělo cyklu se provádí tak dlouho, dokud má *«výraz2»* hodnotu `true`.

Nejobvyklejší použití cyklu `for` je při provádění cyklů, u kterých předem známe počet opakování. Ve *«výrazu1»* inicializujeme řídící proměnnou, *«výraz3»* se stará o pravidelnou aktualizaci této proměnné a *«výraz2»* je podmínka pro provádění cyklu. Náš slavný příklad můžeme pomocí `for` zapsat takto:

```
for ($i=0;$i<100;$i++) echo "$i^2=".($i*$i)."<BR>";
```

V tomto případě se na začátku cyklu do proměnné `$i` uloží nula. Tělo cyklu (příkaz `echo`) se provádí, dokud je proměnná `$i` menší než 100. Na konci každého průchodu cyklem je proměnná `$i` zvýšena o jedničku (pomocí `$i++`).

Pokud se má v těle cyklu provádět více příkazů, opět je uzavřeme do složených závorek:

```
for ($i=0;$i<100;$i++)
{
    echo "$i^2=".($i*$i)."\t";
    echo "$i^3=".($i*$i*$i)."<BR>";
}
```

K dispozici máme i příkaz `for` s alternativní syntaxí:

```
for («výraz1»;«výraz2»;«výraz3»):
    «příkaz1»;
    «příkaz2»;
    ...
endfor;
```

Naši poslední ukázku tedy můžeme zapsat i takto:

```
for ($i=0;$i<100;$i++):
    echo "$i^2=".($i*$i)."\t";
    echo "$i^3=".($i*$i*$i)."<BR>";
endfor;
```

Specialitou příkazu `for` je možnost vynechání libovolného z výrazů, které řídí jeho činnost. Vynecháme-li přitom druhý výraz — podmínku pro provádění cyklu, považuje se výraz za pravdivý a získáme nekonečnou smyčku. Na první pohled tato vlastnost vypadá nesmyslně — k čemu nám bude nikdy nekončící program? Za chvíli se však dozvíme, že provádění cyklu lze přerušit příkazem `break`. Na závěr je tu ukázka cyklu, který bychom v našich skriptech neměli používat:

```
for (;;) echo "Za chvíli už budu v nekonečnu.\n";
```

Výrazy v příkazu `for` mají ještě jednu nestandardní vlastnost. Ve výrazech můžeme používat čárku. Ta slouží k oddělení výrazů, kterých můžeme použít i více. Pokud tedy chceme na začátku provádění cyklu zincializovat dvě proměnné, můžeme psát:

```
for ($i=0, $k=70; $i<10; $i++) «tělo cyklu»;
```

Pokud použijeme čárku ve druhém výrazu pro ukončení těla cyklu, jako podmínka pro ukončení cyklu se použije poslední výraz.

Příkaz break

Příkaz `break` umožňuje okamžité ukončení provádění cyklu. Následující ukázka kopíruje znaky z řetězce `$vstup` do řetězce `$vystup`. Ze vstupního řetězce však znaky kopírujeme pouze do té doby, než narazíme na znak pro konec řádky (`\n`).

```
for ($i=0;$i<StrLen($vstup);$i++)
{
    $vystup .= $vstup[$i];
    if ($vstup[$i]=="\n") break;
}
```

Ve spojení s cyklem `do–while` můžeme `break` výhodně používat pro psaní skriptů, které určitou činnost mohou provést pouze při splnění většího počtu vstupních podmínek. Vše si ukážeme na malé ukázce. Dejme tomu, že chceme vypsat údaje z nějaké tabulky uložené v databázi. Nejprve se musíme připojit k databázi, pak vyvolat dotaz. Při všech těchto činnostech může dojít k chybě, kterou bychom měli ošetřit. Poměrně elegantně to můžeme vyřešit následujícím skriptem:

```
do {
    $spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni) break;
    $vysledek = ODBC_Exec($spojeni, "SELECT * FROM Pokus");
    if (!$vysledek) break;
    «zpracování výsledku dotazu»
} while (false);
```

Klasické řešení využívající vnořené příkazy `if` není tak elegantní, zvláště pokud je možných chybových stavů ještě více než v naší ukázce:

```
$spojeni = ODBC_Connect("test", "", "");
if ($spojeni):
    $vysledek = ODBC_Exec($spojeni, "SELECT * FROM Pokus");
    if ($vysledek):
        «zpracování výsledku dotazu»
    endif;
endif;
```

Za příkazem `break` můžeme uvést číslo, které udává počet cyklů, které budou ukončeny. To využijeme v případech, kdy máme do sebe vnořených více cyklů. Následující skript hledá dvě čísla, jejichž vynásobením získáme číslo 3 476.

```
for ($i=0;$i<100;$i++)
    for ($j=0;$j<100;$j++)
```

```
        if ($i*$j == 3476) break 2;
echo "$i, $j";
```

Pomocí příkazu `break 2` jsme ukončili oba dva cykly najednou ihned po nalezení prvních dvou čísel, která vyhovují naší podmínce.

Příkaz continue

Příkaz `continue` okamžitě přeskočí příkazy ve zbývající části těla cyklu a začne provádět další iteraci. Před prováděním iterace jsou samozřejmě testovány podmínky pro provádění cyklu.

Použití si ukážeme na jednoduchém příkladě. Dejme tomu, že v poli `$adresy` máme uložen adresář lidí. Každý prvek pole obsahuje asociativní pole s údaji, jako je jméno, adresa a datum narozenin. Naším úkolem bude vypsat jména těch osob, které jsou z Prahy a narodili se na silvestra.

```
for ($i=0; $i<Count($adresy); $i++):
    if ($adresy[$i]["Mesto"]!="Praha") continue;
    if ($adresy[$i]["Narozen"]!="31.12.") continue;
    echo $adresy[$i]["Jmeno"];
endfor;
```

Podobně jako u příkazu `break` i u `continue` můžeme zadat počet cyklů, jejichž těla se mají přeskočit.

3.9 Příkazy pro načítání skriptů

Při rutinní tvorbě stránek pomocí skriptů se velice často setkáme s potřebou načítání jiných skriptů či částí stránek. Načítané soubory mohou obsahovat předdefinované funkce, které používáme na více stránkách, nebo třeba standardizované hlavičky a patičky stránek. Pro načítání skriptů máme v PHP k dispozici příkazy `require` a `include`.

Příkaz require

Příkaz `require` slouží k načtení skriptu ze souboru. Načítaný skript se vloží v místě použití příkazu. Provedou se všechny jeho příkazy a poté se v provádění pokračuje příkazem uvedeným za `require`. Syntaxe příkazu:

```
require «jméno souboru»;
```

«jméno souboru» je přitom libovolný výraz, který se vyhodnotí jako řetězec. Specialitou příkazu `require` je, že se provede pouze jednou. Pokud bychom ho volali v cyklu a chtěli načíst soubor vícekrát, načte se soubor pouze při prvním průchodu cyklem. Pro opakované čtení souborů je potřeba využít příkaz `include`.

Parametrem příkazu je jméno souboru. Jméno souboru samozřejmě může obsahovat i cestu. Cestu lze zadat i relativní; jako základní adresář se bere adresář, ze kterého je spouštěn skript obsahující příkaz `require`.

Nejčastěji se `require` používá pro načtení uživatelem definovaných funkcí:

```
require "mojefunkce.php";
```

Mnoho tvůrců knihoven (souborů s různými užitečnými funkcemi) pro PHP používá pro tyto knihovny příponu `.inc`. Pokud však někdo zjistí URL vaší knihovny, a ta má příponu `.inc`, může si ji stáhnout a prohlížet si zdrojový kód vašeho skriptu. Pokud však knihovnu uložíte do souboru s příponou `.php`, bude vždy interpretována systémem PHP a neoprávněnému uživateli dorazí zcela prázdná stránka.

Pokud u jména souboru, který chceme vložit, neuvedeme cestu, hledá se v adresářích určených pomocí direktivy `include_path` v konfiguračním souboru `php3.ini`.

`include_path` standardně neobsahuje nic, a tedy ani tečku označující aktuální adresář. Při načítání souborů z aktuálního adresáře bychom na to měli pamatovat a explicitně adresář určit:

```
require "./mojefunkce.php";
```

Uživatelům Windows možná přijde divné použití lomítka jako znaku oddělujícího jednotlivé adresáře. Tento znak se běžně používá k oddělování adresářů v operačním systému Unix. Aby byly skripty v PHP přenositelné mezi platformami, převádí automaticky PHP pro Windows lomítka na zpětná lomítka. Ve skriptech je proto lepší rovnou používat normální lomítka — neuzavřeme si tak cestu pro přenositelnost našich skriptů.

Příkaz include

Příkaz `include` slouží k načtení skriptu ze souboru. Jeho použití je stejné jako u příkazu `require`. Rozdíl je v tom, že jeden příkaz `include` můžeme volat několikrát — např. uvnitř cyklu. Budeme-li chtít do stránky vložit desetkrát obsah souboru `nejakablbost.inc`, můžeme použít skript:

```
for ($i=0;$i<10;$i++):
    include "./nejakablbost.inc";
endfor;
```

Možná vás napadlo, proč nepoužijeme kratší syntaxi příkazu `for`, když je za ním stejně jen jeden příkaz — příkaz `include`:

```
for ($i=0;$i<10;$i++)
    include "./nejakablbost.inc";
```

Musíme si však uvědomit, že PHP je interpret. Příkaz `include` je v našem případě ekvivalentní mnoha příkazům, případně HTML-kódu, který je obsažen v souboru `nejakablbost.inc`. Nemůžeme jej tedy použít v místě, kde je očekáván jen jeden příkaz. Výše zmíněná ukázka způsobí vypsání deseti chybových hlášení, po kterých bude následovat donekonečna načítání souboru `nejakablbost.inc`.

3.10 Definice vlastních funkcí

Pokud v našich skriptech na různých místech provádíme stejnou sekvenci příkazů, je praktické a užitečné tuto sekvenci příkazů nadefinovat jako funkci. V místech, kde jsme příkazy používali, pak stačí uvést volání naší funkce. Funkce se hodí i v mnoha dalších případech — pokud náš program zahrnuje složitější manipulace s daty, je vhodné je oddělit do samostatných funkcí. Výsledný skript je přehlednější a snadněji se v něm hledají případné chyby.

Definice funkce má následující tvar:

```
function «jméno» («parametr1»,«parametr2»,...)
{
    «příkazy»;
    return «hodnota»;
}
```

Pomocí parametrů funkci předáváme hodnoty, které má zpracovat. Příkazem `return` funkce vrací svůj výsledek. U jednoduchých funkcí můžeme parametry

zcela vynechat, stejně tak jako vracení hodnoty. Definujme si tedy jednoduchou funkci `Pozdrav`, která vypíše pozdrav:

```
function Pozdrav()
{
    echo "Zdravím vás. Doufám, že se máte dobře.";
}
```

Za definicí funkce ve skriptu ji můžeme kdykoliv vyvolat pomocí `Pozdrav()`. Závorky za názvem funkce musíme používat, i když funkce nepracuje s žádnými parametry.

Velká síla funkcí však spočívá právě v možnosti předávat jim parametry. Můžeme si tak definovat různé užitečné funkce, které nám PHP standardně nenabízí k dispozici. Následující program vypíše tabulku matematické funkce faktoriál pro parametry od 1 do 100. Výpočet faktoriálu s výhodou realizujeme jako funkci:

```
function Faktorial($n)
{
    for($i=1; $n>0; $n--) $i *= $n;
    return $i;
}

for($k=1; $k<=100; $k++) echo "$k! = ".Faktorial($k)."<BR>";
```

V definici funkce říkáme, že funkce `Faktorial` má jeden parametr. Jeho obsah je v těle funkce přístupný pomocí proměnné `$n`. Příkaz `for` v těle funkce spočítá hodnotu faktoriálu z `$n` a uloží ji do proměnné `$i`. Tuto hodnotu poté pomocí příkazu `return` vrátíme jako výsledek funkce.

Funkce může mít parametrů samozřejmě více:

```
function Max($a,$b)
{
    return $a>$b ? $a : $b;
}
```

Pomocí funkce `Max()` můžeme zjišťovat, které číslo je větší. Jako parametry můžeme předávat proměnné `Max($a,$q)`, konstanty `Max(10,12)` nebo libovolné výrazy `Max(10*7+$b/$y,Pi())`.

Jako parametr můžeme funkci předávat všechny typy dat, které PHP podporuje — čísla, řetězce ale i pole a objekty. Stejné typy dat může funkce i vracet.

Rozsah platnosti proměnných

V běžných skriptech je proměnná se stejným názvem pořád tatáž proměnná a reprezentuje jednu hodnotu. Pokud však používáme funkce, situace se mění. Proměnné, které použijeme uvnitř funkce nemají s ostatními proměnnými nic společného — dokonce i když se stejně jmenují. Zkuste, co udělá následující skript:

```
function ZmenA()
{
    $a = 10;
}

$a = 20;
ZmenA();
echo $a;
```

Tento skript vypíše číslo 20. Hodnota 10 je do proměnné `$a` přiřazována uvnitř funkce. Změna proměnné `$a` uvnitř funkce nijak neovlivní hodnotu proměnné `$a` použité na nejvyšší úrovni programu.

Důvod pro toto chování PHP je celkem zřejmý — programátor nemůže omylem ve funkci změnit obsah nějaké globální proměnné.

V PHP nejsou narozdíl od jiných jazyků, jako C a Pascal, automaticky k dispozici globální proměnné uvnitř funkcí.

Občas však potřebujeme ve funkci pracovat s hodnotou globální proměnné. V těchto případech můžeme na začátku funkce určit, které globální proměnné může funkce používat. Globální proměnné se funkci zpřístupní pomocí příkazu `global`, za který napíšeme čárkami oddělený seznam proměnných:

```
function ZmenA()
{
    global $a;
    $a = 10;
}

$a = 20;
ZmenA();
echo $a;
```

Upravená ukázka již vytiskne číslo 10. V tomto případě je proměnná `$a` uvnitř funkce totožná s globální proměnnou `$a`, a může tedy měnit její hodnotu.

Kromě použití příkazu `global` máme ještě jednu možnost. Můžeme použít asociativní pole `$GLOBALS`, ve kterém jsou přístupné všechny globální proměnné. Jako index pole použijeme název proměnné (bez znaku dolar):

```
function ZmenA()
{
     $GLOBALS["a"] = 10;
}

$a = 20;
ZmenA();
echo $a;
```

Uvnitř funkcí můžeme definovat i další funkce. Neexistuje však způsob, jak z těchto vnořených funkcí přistupovat k proměnným funkce, která je o úroveň výš. Přístup si můžeme zajistit pouze ke globálním proměnným.

Předávání parametrů

Ve funkcích, které jsme si zatím ukázali, jsme používali způsob předávání parametrů známý jako předávání hodnotou. Při volání funkce jsme na místě parametru uvedli libovolný výraz. Jeho hodnota se předala funkci, a ta na jeho základě něco spočítala.

V PHP však můžeme psát funkce, které mění obsah proměnné, jež je použita jako parametr. Při tomto způsobu předávání parametrů se funkci nepředává hodnota proměnné, ale přímo odkaz na ní — funkce pak může přímo měnit její hodnotu. Tato metoda předávání parametrů se proto jmenuje předávání odkazem.

Funkce, které se mají parametry předávat odkazem, má stejnou syntaxi jako ostatní funkce. Pouze v definici parametrů použijeme znak ‘`&`’ před těmi parametry, které se mají předávat odkazem. Malá ukázka:

```
function Test(&$a,$b)
{
    $a = 10;
    $b = 20;
}

$x = 1;
$y = 2;
Test($x,$y);
echo $x, $y;
```

Po provedení funkce `Test()` bude v proměnné `$x` uložena hodnota 10, protože první parametr funkce je předáván odkazem. Proměnné `$y` zůstane její původní obsah 2, protože je do funkce předána jako parametr předávaný hodnotou.

> Předávání parametrů odkazem má mnoho užitečných uplatnění, ale při jeho použití nesmíme zapomenout, že nám funkce mění obsah proměnné. Při nepozornosti můžeme do skriptu zavléci chyby, jejichž odhalení chvíli potrvá.

Parametry můžeme předávat odkazem i do funkcí, které očekávají pouze parametry předané hodnotou. Pokud při volání funkce použijeme jako parametr proměnnou a před její jméno umístíme znak ‘`&`’, bude proměnné předána odkazem. Pokud bychom v naší poslední ukázce použili volání funkce `Test($x,&$y)`, bude obsah proměnné `$y` po provedení funkce 20, protože v tomto případě bylo `$y` předáno odkazem.

Standardní hodnoty parametrů

V praxi nalezneme mnoho případů funkcí, které se většinou volají se stejnými parametry. Je potom zbytečné chtít po programátorovi, aby do volání funkce pořád opisoval stejné parametry jen kvůli tomu, že občas je potřeba funkci zavolat s jinými parametry.

PHP tento problém elegantně řeší. V definici funkce můžeme určit pro každý parametr standardní hodnotu, která se použije, pokud při volání funkce tento parametr nepoužijeme.

```
function Umocni($zaklad, $exponent = 2)
{
    $pom = $zaklad;
    for($i=1; $i<$exponent; $i++) $pom *= $zaklad;
    return $pom;
}

echo Umocni(10);        // totéž jako Umocni(10,2)
echo Umocni(10,3);
```

Funkce `Umocni()` slouží k počítání mocnin. Protože nejčastěji potřebujeme spočítat druhou mocninu, je definice funkce napsána tak, aby volání funkce s jedním parametrem automaticky způsobilo výpočet druhé mocniny.

Parametry, které mají standardní hodnotu, musí být v seznamu parametrů funkce uvedeny vždy poslední. V opačném případě by PHP nemohlo dosadit

standardní hodnoty parametrů. Pokud máme definovanou funkci s p parametry a při volání ji předáme pouze r parametrů, musí být pro posledních $p - r$ parametrů funkce definována standardní hodnota.

Jako standardní hodnotu parametru funkce můžeme použít pouze skalární typy: `integer`, `double` *a* `string`. *Nelze používat pole a objekty.*

Statické proměnné

Představme si následující situaci: chceme vytvořit funkci pracující jako čítač. Při každém zavolání vrátí hodnotu o jedna vyšší než při minulém volání. Počítání začne od jedničky. První idea funkce může vypadat takto:

```
function Citac()
{
    $n = 0;
    $n++;
    return $n;
}
```

Tato funkce však při každém svém zavolání vrátí 1, protože při každém volání funkce je proměnná `$n`, která slouží jako čítač, vynulována.

V tomto případě s výhodou využijeme možnost definování statických proměnných. Statické proměnné se hodnota přiřadí při definici funkce a při dalších voláních funkce již není znovu inicializována. Statické proměnné se deklarují pomocí příkazu `static`. Nyní již tedy funkční verze našeho čítače:

```
function Citac()
{
    static $n = 0;

    $n++;
    return $n;
}
```

Rekurze

PHP samozřejmě umožňuje rekurzivní volání funkcí. O co jde? Při rekurzivním volání se uvnitř těla funkce použije volání jí samé. Pomocí rekurze lze poměrně stručně vyjádřit některé programy. Rekurze však přináší i mnohá rizika. Při přílišné hloubce volání funkce může dojít k přetečení zásobníku.

Použití rekurze si ukážeme na funkci pro výpočet faktoriálu. Předem poznamenejme, že mnohem lepší je použití nerekurzivní funkce, kterou jsme si ukázali na straně 67.

```
function Faktorial($n)
{
    return $n==0 ? 1 : $n * Faktorial($n-1);
}
```

Pokud vás baví psaní úsporného kódu, zkuste přijít na to, proč můžeme za `return` použít výraz `$n ? $n * Faktorial($n-1) : 1` a celá funkce bude pořád pracovat.

3.11 Vlastnosti jazyka, které se jinam nevešly

Celý systém PHP byl navržen s ohledem na programátory. Obsahuje proto mnoho různých možností, jak si zpříjemnit a zjednodušit psaní skriptů. Ten kdo psal někdy nějaké skripty např. v ASP od Microsoftu, musí uznat, že syntaxe PHP je mnohem úspornější. V této sekci se seznámíme s několika dalšími vlastnostmi jazyka PHP, na které nám dosud nezbyl čas.

Ukončení běhu skriptu

V kterékoliv fázi běhu skriptu jej můžeme ukončit použitím příkazu `exit` nebo funkce `Die()`. Při používání této funkce bychom měli být opatrní. Pokud v našem skriptu kombinujeme HTML s příkazy PHP a použijeme někde uprostřed skriptu `exit`, do prohlížeče se nic dalšího neposílá — tedy ani případné ukončovací tagy jako `</BODY></HTML>`. Náš skript v tomto případě vygeneruje nekorektní HTML-stránku. Tento nedostatek můžeme v některých případech odstranit pomocí funkce, která se volá při ukončení skriptu (viz funkce `Register_ShutDown_Function()`).

Eval – magická hůlka

Příkaz `eval` je v mnoha ohledech bez nadsázky magický. Jako parametr mu můžeme předat libovolný řetězec. Obsah toto řetězce je proveden, jakoby se jednalo o příkazy PHP. Malá ukázka:

```
$fnName = 'Soucet($x, $y)';
$fnValue = '$x + $y';
eval ("function $fnName { return $fnValue; }");   // definujeme funkci Soucet
echo Soucet(10, 20);
```

`eval` může nalézt uplatnění v případech, kdy potřebujeme do databáze uložit kód, který se má provést později. Neustále musíme mít na paměti, že jako parametr příkazu `eval` je potřeba zadat syntakticky správný kód — příkazy musí být oddělené středníky a musíme si dát pozor na to, kdy použít escape sekvence.

```
eval ("echo \$text;");          // Vytiskne obsah proměnné $text
eval ("\$text = \"Ahoj\";");    // Do proměnné $text uloží text Ahoj
eval ("$text = \"Nazdar\";");   // Ohlásí chybu, protože příkaz
                                // Ahoj = "Nazdar"; není syntakticky správný
```

Konstanty

Konstanty podobně jako proměnné slouží k uchovávání hodnot, které můžeme používat v našich skriptech. Jak již z názvu vyplývá, nejde však konstanty, narozdíl od proměnných, během chodu skriptu měnit. Konstantu jednou nadefinujeme a od té doby ji nelze měnit.

Pro jména konstant platí totéž, co pro jména proměnných. Rozdíl je pouze v tom, že před jménem konstanty se nepoužívá znak '`$`'. Pro definování konstanty se používá funkce `define()`. Její syntaxe je:

```
define(«jméno konstanty», «hodnota»)
```

Konstanty můžeme definovat pouze skalárního typu — `integer`, `double` a `string`. Pro definici konstanty `MaxN`, která bude mít hodnotu 100, můžeme použít:

```
define("MaxN", 100);
```

Ve skriptu nyní můžeme konstantu `MaxN` používat ve všech výrazech místo čísla 100:

```
for($i=0; $i<MaxN; $i++) «dělej něco»;
```

Narozdíl od proměnných mají konstanty globální platnost. Jsou přístupné i ve všech funkcích a objektech.

Pokud chceme definovat konstantu tak, aby v jejím jménu nezáleželo na velikosti písmen, můžeme u `define` použít třetí parametr a jako jeho hodnotu uvést `true`:

```
define("MaxN", 100, true);
```

Nyní můžeme používat konstanty `MaxN`, `maxn`, `MAXN`, `MAXn` a další. Všechny zastupují hodnotu 100.

Ke zjištění, zda je nějaká konstanta definována, slouží funkce `defined()`. Jako parametr se uvádí jméno konstanty.

```
if (defined("MaxN"))
    echo "Konstanta MaxN je již definována";
else
    define("MaxN", 100);
```

Proměnná chameleon

PHP nabízí zajímavou možnost pro přístup k proměnným. Dejme tomu, že v proměnné `$a` máme uložen text ‘`b`’. Pokud ve skriptu použijeme zápis `$$a`, je to totéž, jako kdybychom použili `$b`. Proč to tak funguje? PHP se nejprve podívalo do proměnné `$a` a její obsah použilo jako název proměnné, protože jsme měli ještě jeden neobsloužený `$`.

Nemusíme však být žádní troškaři a dolarů můžeme použít, kolik chceme. Při vyhodnocování jména proměnné se bude postupovat obdobně jako v předchozím případě. Malá ukázka:

```
$a = "b";
$b = "c";
$c = "d";
$d = "Chameleon se nezdá.";
echo $$$$a;                     // Vypíše Chameleon se nezdá
```

Se jmény proměnných si můžeme hrát ještě mnohem více. K proměnné totiž můžeme přistupovat i pomocí zápisu `${`*«výraz»*`}`. Hodnota *«výrazu»* je přitom považována za jméno proměnné. Do proměnné `$mereni13` tedy můžeme uložit hodnotu 27 i pomocí následujícího skriptu:

```
$index = 13;
${"mereni".$index} = 27;
```

Do podobné kategorie jako výše zmíněné vlastnosti PHP spadá i možnost vytváření ukazatelů na funkce. Vše si ukážeme na malé ukázce:

```
$fptr = "Abs";
echo $fptr(-10);
```

Výsledný efekt je stejný jako při napsání `echo Abs(-10)`.

Pokud vás nenapadá k čemu výše zmíněné machrování se jmény proměnných použít, klidně na ně zapomeňte.

3.12 Objekty a PHP

Objektově orientované programování (OOP) se stalo hitem v osmdesátých letech. Od té doby se přišlo na to, že OOP není samospasitelné. I přesto však OOP do programování přineslo mnoho zajímavých a užitečných myšlenek. Přestože je PHP velice jednoduchý skriptovací jazyk, nabízí i základní podporu pro OOP.

Výhodou OOP je možnost definice tzv. tříd. Třída je speciální typ, který může obsahovat data a zároveň funkce, které s těmito daty umožňují pracovat. Tento přístup nám umožňuje umístit do jedné kompaktní jednotky — třídy — vše, co řeší nějakou elementární úlohu. Naše programy tak můžeme zpřehlednit, místo velkého množství funkcí a globálních proměnných si můžeme nadefinovat několik tříd.

Pokud již máme třídu nadefinovanou, můžeme v programu vytvářet její instance — tzv. objekty. Objekt není nic jiného než proměnná, jejíž typ je určitá třída. Pomocí této proměnné pak můžeme přistupovat k datům objektu a k jeho funkcím.

Pro podrobné vysvětlení principů OOP máme příliš úzký prostor. Ukážeme si proto pouze, jak se s objekty v PHP pracuje. Zájemce s hlubším zájmem o OOP mi nezbývá než odkázat na další literaturu, které v českém jazyce zase tak moc není.

Definice třídy se řídí následujícím schématem:

```
class «jméno třídy»
{
    «definice členských proměnných»
    «definice členských funkcí»
}
```

Členské proměnné

Nejjednodušší třídy nemusí mít vůbec definovány žádné členské funkce. Ač se to nezdá, třída, která obsahuje pouze data, má mnoho praktických oblastí využití. Můžeme ji využít pro vytváření složitějších datových struktur, které jsou obdobou struktur z jazyka C či záznamů z jazyka Pascal.

Definice členských proměnných se uvozuje klíčovým slovem `var`. Za ním následuje seznam proměnných oddělených čárkami. Částí `var` můžeme uvést i více za sebou. Velice jednoduchá třída pro uchovávání adresy může vypadat třeba takto:

```
class CAdresa
{
    var $Jmeno, $Prijmeni;
    var $Ulice, $Obec, $PSC;
}
```

> Pro pořádek je dobré třídy pojmenovávat jednotným způsobem — třeba jejich jméno začínat písmenem ‘C’.

Pokud máme vytvořenou třídu, můžeme vytvářet její instance:

```
$adresa = new CAdresa;
```

Proměnná `$adresa` je nyní objekt — instance třídy `CAdresa`. Pomocí zápisu `$proměnná->`*«členská proměnná»*, můžeme přistupovat k jednotlivým členským proměnným objektu:

```
$adresa->Jmeno = "Jan";
$adresa->Prijmeni = "Novák";
$adresa->Ulice = "Dlouhá 13";
$adresa->Obec = "Praha 1";
$adresa->PSC = "110 00";
```

Členské proměnné můžeme při jejich definici i inicializovat.

```
class CAdresa
{
    var $Jmeno, $Prijmeni;
    var $Ulice, $Obec = "Praha 1", $PSC = "110 00";
}

$adr = new CAdresa;
echo $adr->Obec;        // Vytiskne Praha 1
```

```
echo $adr->PSC;         // Vytiskne 110 00
echo $adr->Jmeno;       // Nevytiskne nic, protože zatím není zinicializováno
```

Členské funkce

Jako členské funkce se obvykle definují funkce, které manipulují s daty objektu. V našem případě může být užitečné vytvoření členské funkce `Tisk()`, která bude sloužit k tisku adresy:

```
class CAdresa
{
    var $Jmeno, $Prijmeni;
    var $Ulice, $Obec, $PSC;

    function Tisk()
    {
        echo $this->Jmeno." ".$this->Prijmeni.": ";
        echo $this->Ulice.", ".$this->Obec.", ".$this->PSC;
    }
}
```

Pokud se uvnitř definice členské funkce odvoláváme na členské proměnné, musíme používat speciální zápis `$this->`*«členská proměnná»*. Obsah proměnné `$this` je při skutečném volání funkce nahrazen objektem, ze kterého je volán. Nyní si vytvoříme instanci třídy a vyzkoušíme funkci `Tisk()`:

```
$adresa = new CAdresa;
$adresa->Jmeno = "Jan";
$adresa->Prijmeni = "Novák";
$adresa->Ulice = "Dlouhá 13";
$adresa->Obec = "Praha 1";
$adresa->PSC = "110 00";
$adresa->Tisk();              // Volání členské funkce
```

Jako výsledek bychom měli získat:

```
Jan Novák: Dlouhá 13, Praha 1, 110 00
```

Členské funkce mohou mít samozřejmě i parametry. Funkci `Tisk()` můžeme modifikovat tak, že bude schopna tisknout adresu do jedné řádky nebo pod sebe po jednotlivých položkách. Vzhled tisku budeme řídit právě pomocí parametru funkce. Parametru zároveň přiřadíme standardní hodnotu tak, aby vyvolání funkce bez parametru způsobilo tisk do jedné řádky:

```
class CAdresa
{
    var $Jmeno, $Prijmeni;
    var $Ulice, $Obec, $PSC;

    function Tisk($podsebe = false)
    {
        if ($podsebe):
            echo $this->Jmeno." ".$this->Prijmeni."<BR>";
            echo $this->Ulice."<BR>";
            echo $this->Obec."<BR>";
            echo $this->PSC."<BR>";
        else:
            echo $this->Jmeno." ".$this->Prijmeni.": ";
            echo $this->Ulice.", ".$this->Obec.", ".$this->PSC;
        endif;
    }
}

$adresa = new CAdresa;
$adresa->Jmeno = "Jan";
$adresa->Prijmeni = "Novák";
$adresa->Ulice = "Dlouhá 13";
$adresa->Obec = "Praha 1";
$adresa->PSC = "110 00";
$adresa->Tisk(true);
```

Výsledek našeho snažení si můžeme prohlédnout na obrázku 3-2.

Jan Novák
Dlouhá 13
Praha 1
110 00

Obr. 3-2: Výsledek našeho prvního objektového skriptu

Zatím vše kolem objektů vypadá jako nějaká pěkná hračka, která není moc užitečná. Objekt však může být prvkem pole. S využitím této vlastnosti lze již dosáhnout zajímavých efektů. Do pole můžeme ukládat objekty s jednotlivými údaji a pak je hromadně zpracovávat tím, že postupně na celé pole aplikujeme nějakou členskou funkci. Možností je opravdu mnoho a záleží pouze na naší fantazii.

Konstruktory

Speciální členskou funkcí je konstruktor. Konstruktor je členská funkce, jejíž název se shoduje s názvem třídy. Konstruktor je volán vždy, když je nějaký objekt vytvářen pomocí operátoru `new`. Konstruktor se nejčastěji používá pro usnadnění inicializace objektu a případně pro naalokování zdrojů, se kterými objekt pracuje. My si použití ukážeme na našem příkladě. Vytvoříme konstruktor, který usnadní inicializaci nových instancí adresy:

```
class CAdresa
{
    var $Jmeno, $Prijmeni;
    var $Ulice, $Obec, $PSC;

    function CAdresa($Jmeno, $Prijmeni, $Ulice, $Obec, $PSC)
    {
        $this->Jmeno = $Jmeno;
        $this->Prijmeni = $Prijmeni;
        $this->Ulice = $Ulice;
        $this->Obec = $Obec;
        $this->PSC = $PSC;
    }

    function Tisk($podsebe = false)
    {
        if ($podsebe):
            echo $this->Jmeno." ".$this->Prijmeni."<BR>";
            echo $this->Ulice."<BR>";
            echo $this->Obec."<BR>";
            echo $this->PSC."<BR>";
        else:
            echo $this->Jmeno." ".$this->Prijmeni.": ";
            echo $this->Ulice.", ".$this->Obec.", ".$this->PSC;
        endif;
    }
}

$adresa = new CAdresa("Jan", "Novák", "Dlouhá 13", "Praha 1", "110 00");
$adresa->Tisk();
```

Konstruktor můžeme ještě vylepšit tím, že nadefinujeme standardní hodnoty parametrů. Objekt pak půjde vytvořit i před tím, než budeme znát hodnoty, které do něj chceme uložit:

```
function CAdresa($Jmeno="", $Prijmeni="", $Ulice="", $Obec="", $PSC="")
```

Pak máme mnohem víc možností inicializace objektu:

```
$adresa1 = new CAdresa;
$adresa2 = new CAdresa();
$adresa3 = new CAdresa("Jan", "Novák", "Dlouhá 13", "Praha 1", "110 00");
$adresa4 = new CAdresa("Karel", "Procházka");
```

Dědičnost

Poslední možností, o které jsme se v souvislosti s objekty nezmínili, je dědičnost. V PHP máme možnost vytvořit třídu, která zdědí všechny členské proměnné a funkce z nějaké jiné třídy. V nově vzniklé třídě můžeme přidat další členské proměnné a funkce. V případě potřeby můžeme změnit zděděné členské funkce.

Vše si postupně ukážeme na malém příkladě. Dejme tomu, že chceme vytvořit třídu pro ukládání adresy firem. Můžeme využít toho, že již máme k dispozici třídu pro ukládání obyčejných adres. U firem potřebujeme mít navíc k dispozici název firmy. Definujeme si tedy novou třídu `CFiremniAdresa`, která bude potomkem třídy `CAdresa`:

```
class CFiremniAdresa extends CAdresa
{
    var $Nazev;
}
```

Vytvoříme-li nyní instanci této třídy

```
$firma = new CFiremniAdresa;
```

můžeme nastavit její jednotlivé členské proměnné, včetně těch, které jsme zdědili:

```
$firma->Nazev = "Megamix, s.r.o";
$firma->Ulice = "Krátká 31";
$firma->Obec = "Praha 10";
$firma->PSC = "101 00";
```

Můžeme zavolat i členskou funkci `$firma->Tisk()`. Bohužel se v tomto případě nevytiskne název firmy. Nevytiskne se, protože funkce `Tisk()` je zděděná ze třídy `CAdresa`, která o žádném názvu firmy nemá ani ponětí.

Rovněž by bylo vhodné v nové třídě upravit konstruktor tak, aby pomocí něj šlo snadno inicializovat údaje o firmě. Upravíme tedy definici naší nové třídy:

```
class CFiremniAdresa extends CAdresa
{
```

```
    var $Nazev;

    function CFiremniAdresa($Nazev, $Ulice, $Obec, $PSC)
    {
        $this->Nazev = $Nazev;
        $this->Ulice = $Ulice;
        $this->Obec = $Obec;
        $this->PSC = $PSC;
    }

    function Tisk($podsebe = false)
    {
        if ($podsebe):
            echo $this->Nazev."<BR>";
            echo $this->Ulice."<BR>";
            echo $this->Obec."<BR>";
            echo $this->PSC."<BR>";
        else:
            echo $this->Nazev.": ";
            echo $this->Ulice.", ".$this->Obec.", ".$this->PSC;
        endif;
    }
}
```

Nyní již můžeme použít následující elegantní zápis:

```
$firma = new CFiremniAdresa("Megamix, s.r.o", "Krátká 31",
                            "Praha 10", "101 00");
$firma->Tisk();
```

Při definování konstruktoru jsme si mohli ušetřit práci. K dispozici máme totiž zděděný konstruktor — funkci `CAdresa()`. Stačilo by, aby konstruktor třídy `CFiremniAdresa` zavolal původní konstruktor z třídy `CAdresa` a sám nainicializoval jen členskou proměnnou pro název firmy. Po úpravě tedy dostaneme kratší:

```
class CFiremniAdresa extends CAdresa
{
    var $Nazev;

    function CFiremniAdresa($Nazev, $Ulice, $Obec, $PSC)
    {
        $this->CAdresa("", "", $Ulice, $Obec, $PSC);
        $this->Nazev = $Nazev;
```

```
    }

    function Tisk($podsebe = false)
    {
        if ($podsebe):
            echo $this->Nazev."<BR>";
            echo $this->Ulice."<BR>";
            echo $this->Obec."<BR>";
            echo $this->PSC."<BR>";
        else:
            echo $this->Nazev.": ";
            echo $this->Ulice.", ".$this->Obec.", ".$this->PSC;
        endif;
    }
}
```

Podobný mechanismus bohužel nelze použít u ostatních členských funkcí. V PHP neexistuje způsob, jak se uvnitř členské funkce odkázat na jejího předka.

Tímto ukončíme naši krátkou exkurzi do světa objektů, která nás seznámila s jejich principy. Několik dalších ukázek využití objektů naleznete v kapitole obsahující praktické ukázky použití PHP.

3.13 Regulární výrazy

Ačkoliv má tato sekce ve svém názvu slovo „výrazy", se sekcí této kapitoly věnované výrazům takřka nesouvisí. Regulární výrazy jsou výbornou pomůckou při zpracování textových řetězců. Pomocí regulárního výrazu můžeme velice snadno definovat masku, která se pak používá ke zjišťování obsahu textových řetězců. V PHP jsou k dispozici funkce, které zjišťují, zda daný řetězec vyhovuje masce. Nalezneme zde i mocnější funkce, které umožňují na základě regulárního výrazu z řetězce separovat jeho jednotlivé části.

Regulární výraz je textový řetězec, který obsahuje text. Řetězec pak regulárnímu výrazu vyhovuje, pokud v sobě obsahuje tento text. Tak například regulárnímu výrazu `abc` vyhovují řetězce `abc`, `abcdef` i `xxabcq` — všechny v sobě obsahují řetězec `abc`.

Vše vypadá velice jednoduše — zatím. Situaci však komplikuje to, že v regulárních výrazech mají některé znaky speciální význam. Těmto znakům říkáme metaznaky. Tak například metaznak '`^`' zastupuje začátek řetězce. Regulárnímu výrazu `^abc` tedy vyhoví pouze ty řetězce, které začínají na znaky `abc` — např. `abc` a `abcdef`.

Podobně funguje i metaznak '`$`', který naopak označuje konec řetězce. Chceme-li tedy vyrobit regulární výraz, kterému vyhoví řetězce končící na určitou sekvenci znaků, použijeme na jeho konci znak '`$`'.

Možná vás teď napadá, jak vyrobit regulární výraz, kterému vyhovuje např. řetězec `2^3`. Pokud chceme použít metaznak v jeho původním významu, přidáme před něj lomítko. Výše zmíněný řetězec tedy vyhoví regulárnímu výrazu `2\^3`. Obdobně můžeme zapisovat i ostatní metaznaky, pokud je chceme použít jako normální znak. Pro zápis zpětného lomítka použijeme dvě zpětná lomítka '`\\`'.

Opravdová síla regulárních výrazů však spočívá v dalších metaznacích, které nám PHP nabízí. Velice užitečný metaznak je '`.`'. Tečka zastupuje jeden libovolný znak. Regulárnímu výrazu `a.c` vyhoví např. řetězce `abc`, `axc` a `a0c`.

Další metaznaky '`*`', '`+`' a '`?`' slouží k určení, kolikrát se má výraz uvedený před nimi opakovat. Hvězdička přitom říká, že výraz se může opakovat kolikrát chce, dokonce nemusí být přítomný vůbec. Plus naopak říká, že se výraz musí vyskytovat alespoň jednou. Pokud použijeme otazník, vyhoví ty řetězce, kde se výraz vyskytuje nejvýše jednou.

Několik příkladů. Regulárnímu výrazu `.*` vyhoví libovolný řetězec, dokonce i prázdný. Tečka zastupuje libovolný znak a hvězdička za ní nám říká, že se tento znak může opakovat, kolikrát bude potřeba.

Regulárnímu výrazu `ab+a` vyhoví všechny řetězce, které obsahují dva znaky `a`, mezi kterými je libovolný nenulový počet znaků `b`. Např. `aba`, `abba`, `abbbbbbbba`, `qwerabbaiop` a další.

Naopak regulárnímu výrazu `ab?a` vyhoví pouze řetězce obsahující `aba` nebo `aa`.

Vidíme, že metaznaky pro opakování se aplikují pouze na poslední znak. Pokud chceme zajistit opakování větší části regulárního výrazu, stačí ji uzávorkovat a vytvořit tak regulární podvýraz. Chceme-li vytvořit regulární výraz, kterému vyhoví pouze řetězce obsahující dva znaky `ab`, které se pořád opakují, použijeme regulární výraz `^(ab)+$`.

Pokud chceme mít nad počtem opakování ještě lepší kontrolu, můžeme použít metaznak '`{`'. Zápisem `{`*«min»*`,`*«max»*`}` říkáme, že předcházející podvýraz se má opakovat minimálně *«min»*krát a maximálně *«max»*krát. Výrazu `x{2,10}` vyhoví řetězce, které obsahují 2–10 znaků `x` zapsaných těsně za sebou.

Poslední metaznak má i dvě další varianty. Pokud chceme určit pouze minimální počet opakování a maximální počet nechceme omezit, použijeme metaznak `{`*«min»*`,}`. Pokud chceme určit, že podvýraz se má opakovat přesně *«n»*krát, použijeme `{`*«n»*`}`.

Dej si pauzu, dej si ________________ (Vyplňte dle osobních preferencí).

Osvěženi, můžeme pokračovat. Již známe metaznak '.', který zastupuje libovolný znak. Často však potřebujeme použít pouze jeden z nějaké množiny znaků. Pro tyto účely slouží metaznak '['. Pokud do hranatých závorek napíšeme několik znaků, zastupuje tento regulární výraz jeden libovolný znak uvedený v hranatých závorkách. Regulárnímu výrazu [ab]x vyhoví řetězce obsahující ax nebo bx. Znaků můžeme uvnitř závorek použít, kolik chceme. Není tedy problém napsat regulární výraz, kterému vyhoví pouze zápisy celých čísel:

```
^[\+\-]?[0123456789]+$
```

Malý komentář pro ty, kteří se právě probudili. Regulární výraz začíná metaznakem '^' pro začátek řetězce a končí metaznakem '$' pro konec řetězce. To znamená, že zbytku regulárního výrazu musí vyhovovat celý řetězec, nestačí pouze jeho část. Celé číslo může začínat znaménkem + nebo -. Jde však o metaznaky, a proto jsme je museli zapsat pomocí přepisu \+ a \-. Znaménko však v čísle být vůbec nemusí, a proto je za výrazem pro znaménka uveden otazník. Následující část výrazu nám říká, že se mohou opakovat cifry 0 až 9. Cifra však musí být alespoň jedna, protože jsme použili metaznak '+'.

Zápisy uvnitř metaznaku '[' můžeme i efektivně zkracovat. Náš poslední regulární výraz by šel zapsat i zkráceně jako ^[\+\-]?[0-9]+$. Zápis *«x»*-*«y»* zastupuje množinu všech znaků, které lexikograficky leží mezi znaky *«x»* a *«y»* včetně. Regulární výraz zastupující libovolné písmeno anglické abecedy tedy je [a-zA-Z].

Pokud jako první znak množiny znaků uvedeme '^', regulární výraz vyhoví libovolnému znaku, kromě těch uvedených. Regulárnímu výrazu mikro[^ns] tedy vyhoví např. slovo mikroklima nebo mikrob, nikoli však slova mikron a mikros.

Při konstrukci složitějších regulárních výrazů se nám může hodit metaznak '|'. Ten slouží k rozdělení regulárního výrazu na několik podvýrazů. Řetězec regulárnímu výrazu pak vyhovuje, pokud vyhovuje alespoň jedné z jeho částí. Regulární výraz, který zjistí, zda řetězec obsahuje zkratku dne v týdnu, může vypadat takto:

```
Po|Út|St|Čt|Pá|So|Ne
```

Podle potřeby můžeme metaznak '|' používat i s podvýrazy. Pokud chceme vyrobit regulární výraz, kterému vyhoví zkratka dne v týdnu, a to buď česká, nebo anglická, můžeme použít následující výraz:

```
(Po|Út|St|Čt|Pá|So|Ne)|(Mon|Tue|Wed|Thu|Fri|Sat|Sun)
```

Pokud bychom chtěli vytvořit regulární výraz, kterému nezáleží na velikosti prvního písmene ve zkratce dne, dostaneme už poměrně dlouhou obludu:

```
((P|p)o|(Ú|ú)t|(S|s)t|(Č|č)t|(P|p)á|(S|s)o|(N|n)e)|
((M|m)on|(T|t)ue|(W|w)ed|(T|t)hu|(F|f)ri|(S|s)at|(S|s)un)
```

Zápis lze zkrátit, pokud si uvědomíme, jak pracuje metaznak '`[...]`':

```
([Pp]o|[Ůú]t|[Ss]t|[Čč]t|[Pp]á|[Ss]o|[Nn]e)|
([Mm]on|[Tt]ue|[Ww]ed|[Tt]hu|[Ff]ri|[Ss]at|[Ss]un)
```

Zajímavé jsou poněkud obskurní regulární výrazy `[[:<:]]` a `[[:>:]]`, které zastupují začátek a konec slova. Pokud chceme vytvořit regulární výraz, kterému vyhoví řetězec obsahující slovo `Novák`, ale nikoliv již řetězec se jménem jeho manželky, můžeme použít regulární výraz:

```
[[:<:]]Novák[[:>:]]
```

Poslední možností regulárních výrazů, se kterou se seznámíme, jsou třídy znaků. Výraz `[[:`*«třída»*`:]]` zastupuje jeden libovolný znak, který patří do *«třídy»*. K dispozici máme několik tříd — např. `alpha` pro písmena, `digit` pro číslice, `xdigit` pro šestnáctkové číslice. Kompletní přehled je uveden v tabulce 3-2.

Při použití tříd znaků si musíme uvědomit, že pracují s anglickou abecedou, a tak třeba třída pro písmena vůbec nezahrnuje znaky s diakritickými znaménky.

Třída	**Popis**
`alnum`	Alfanumerické znaky (písmena anglické abecedy a číslice)
`alpha`	Písmena anglické abecedy
`blank`	Mezera a tabulátor
`cntrl`	Řídící znaky
`digit`	Číslice
`graph`	Všechny znaky, které mají grafické znázornění
`lower`	Malá písmena anglické abecedy
`print`	Tisknutelné znaky (jako `print`, navíc je přidán znak mezera)
`punct`	Interpunkční a další znaky (závorky, zavináče a podobná zvěrstva)
`space`	Jakákoliv mezera (mezera, tabulátor, nová řádka, nová stránka, vertikální tabulátor)
`upper`	Velká písmena anglické abecedy
`xdigit`	Šestnáctkové číslice (číslice a písmena 'a'–'f', 'A'–'F')

Tab. 3-2: Přehled tříd znaků použitelných v regulárních výrazech

Regulární výraz, kterému vyhoví řetězce obsahující čtyřmístné hexadecimální číslo, může vypadat třeba takto:

```
[[:<:]][[:xdigit:]]{4}[[:>:]]
```

Z posledního příkladu je jasně vidět, že je nejvyšší čas s regulárními výrazy skončit. Probrali jsme vše až na několik téměř nepoužívaných či spíše nepoužitelných konstrukcí. Zájemci si mohou podrobně prostudovat dokumentaci k regulárním výrazům po zadání příkazu `man regex`. Pokud vám příkaz nefunguje, s největší pravděpodobností pracujete pod Windows. A to byste již měli být zvyklí na to, že v nápovědě se dozvíte pouze o tom, jak s myší klikat po různých oknech a tlačítkách, která občas škodolibě uhýbají.

4. Ladění skriptů a ošetření chyb

Stará známá programátorská poučka říká, že v každém programu je nejméně jedna chyba.[1] O pravdivosti toto tvrzení bychom mohli dlouze diskutovat, pravdou však je, že alespoň ze začátku budou vaše skripty obsahovat nějaké syntaktické chyby — tu zapomenete na středník, tu na uzavření řetězce do úvozovek. Dlouhá praxe v psaní skriptů vás však vycvičí a syntaktické chyby budete dělat jen zřídkakdy. Občas se však žádný programátor nevyhne tomu, aby v programu udělal nějakou logickou chybičku, která způsobí nesprávné chování programu. V této kapitole si proto povíme, jak zjistit a odstranit syntaktické chyby v našich skriptech a jak psát programy tak, aby nebyly náchylné na logické chyby. K chybě může samozřejmě dojít i při volání nějaké interní funkce systému PHP. Ukážeme si proto, jak tyto chyby zjistit a jak je ošetřit.

4.1 Syntaktické chyby

PHP je interpretovaný jazyk. To s sebou přináší mnoho výhod i nevýhod. Asi největší výhodou je velká rychlost vývoje aplikací. Stačí napsat skript a prohlédnout si jeho výsledek v prohlížeči. Narozdíl od jazyků, které se musí překládat do strojového kódu, je to nesmírná úspora času — nemusí se spouštět kompilátor a linker.

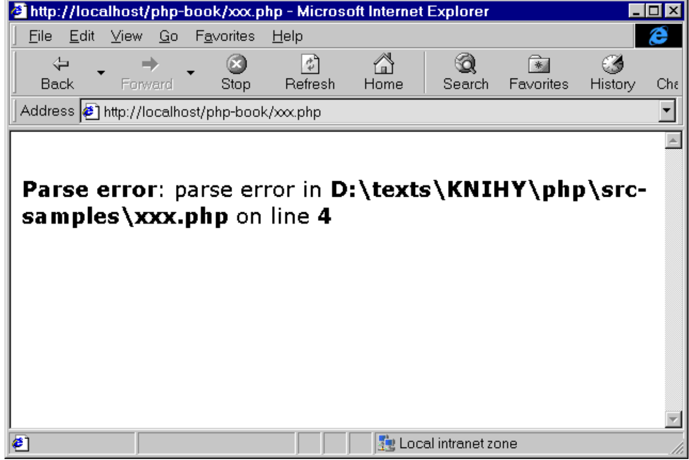
Na druhou stranu kompilátor odhalí všechny syntaktické chyby, jako jsou zkomolené názvy proměnných a funkcí, opomenuté středníky mezi příkazy či řetězce neuzavřené do uvozovek. PHP podobné chyby odhalí až při samotném vykonávání skriptu. Pokud PHP na nějakou syntaktickou chybu narazí, vypíše chybové hlášení a přeruší běh skriptu. Jak takové hlášení může vypadat, si můžeme prohlédnout na obrázku 4-1 na následující straně.

Z hlášení je patrné, že jde o syntaktickou chybu (Parse error), v jakém je skriptu a na jaké řádce. Podívejme se tedy na skript, který způsobil chybu:

```
1   <?
2       $x = 10;
3       $y = 20
4       echo $x+$y;
5   ?>
```

Na první pohled je chyba jasná — za příkazem `$y = 20` chybí středník, který by příkaz ukončil. Trošku nás může mást, že pro nás je chyba na řádku 3, kdežto PHP hlásí chybu na řádku 4. Vysvětlení je prosté. Na třetím řádku není nic

[1] Další poučka říká, že každý program lze zkrátit o jednu instrukci. Spojením předchozích dvou pouček zjistíme, že každý program lze zkrátit na jednu chybnou instrukci.



Obr. 4-1: PHP se na nás zlobí — ve skriptu je syntaktická chyba

špatně. Na dalším řádku totiž může pokračovat výraz, který přiřazujeme do proměnné `$y`. Na čtvrtém řádku PHP však zjistí, že výraz nepokračuje ani není ukončen středníkem, ale z ničeho nic se objevil příkaz `echo`, který tu nemá co dělat. PHP proto ohlásí chybu.

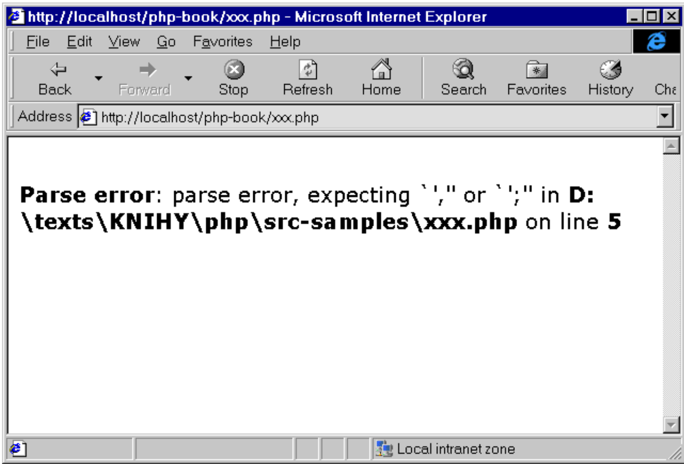
Pokud nám PHP ohlásí, že je na nějakém řádku chyba, chyba může být klidně na některém z předchozích řádků.

První syntaktickou chybu jsme odhalili. Pojďme však dál. Co může být příčinou výřečnějšího chybového hlášení, které vidíme na obrázku 4-2 na následující straně?

Chybové hlášení nám nyní říká, že PHP na řádce 5 očekává čárku nebo středník. Hlášení zní podivně a ani skript, který způsobil chybu, nám na první pohled moc nepomůže.

```
1   <?
2   if ($Jmeno=="Novák")
3       echo "Ahoj Nováku!\n;
4   else
5       echo "Tebe neznám příšero!\n";
6   ?>
```

Nevím, jak vy, ale já na řádce 5 žádnou chybu nevidím. Nezbývá nám tedy nic jiného, než zapátrat na předchozích řádkách. Vidíme, že chyba je na třetí řádce — řetězec za příkazem `echo` není na svém konci uzavřen uvozovkou. Proč však PHP



Obr. 4-2: PHP od nás očekává čárku nebo středník a vyšší inteligenci

hlásí chybu až na řádce pět? Odpověď je jednoduchá. Na třetí řádce začíná uvozovkami textový řetězec, nejbližší další uvozovka, která řetězec ukončí, je až na pátém řádku za slovem `echo`. Příkaz `echo` na třetí řádce má díky naší nechtěné chybě jako argument textový řetězec „`Ahoj Nováku!\n;\nelse\n    echo`“. Za tímto argumentem příkaz `echo` očekává buď další argument oddělený čárkou, nebo středník ukončující příkaz. Proto PHP vypsalo chybovou hlášku, kterou jsme viděli.

4.2 Logické chyby

Ne, že by syntaktické chyby byly nějak příjemné, ale jejich odhalení je většinou snadné. Interpret PHP se na první syntaktické chybě ve skriptu zastaví a vypíše hlášení, které nám většinou pomůže velice rychle lokalizovat příčinu chyby. Mnohem horší jsou logické chyby v našich programech. Ty PHP těžko zjistí, protože nemůže vědět, co chceme naprogramovat.

Pokud zjistíme, že náš skript nepracuje tak, jak očekáváme, musíme se pokusit zjistit, kde je chyba. Bohužel nejsem kouzelný dědeček a nemohu vám dát univerzální radu, jak odhalit chyby ve skriptech. Existuje však několik postupů, které nám mohou pomoci odhalit chyby.

I v programování platí klasické přísloví dvakrát měř a jednou řež. Pokud potřebujeme napsat nějaký skript nebo aplikaci, je dobré si celý problém nejprve pořádně promyslet. Musíme se rozhodnout, jaké použijeme datové struktury, jaké algoritmy a jak spolu budou jednotlivé části aplikace spolupracovat. Pokud se do psaní aplikace pustíme zcela bez rozmyšlení, je celkem pravděpodobné, že se do ní zamotáme a vytvoříme téměř nepoužitelný a nepřehledný bastl.

Při psaní skriptu si musíme dávat pozor na obvyklé chybičky, které se nenápadně vloudí do každého skriptu. Z vlastní zkušenosti vím, že v PHP se chyby nejčastěji zapříčiní použitím neinicializovaných proměnných a záměnou operátoru přiřazení '=' a porovnání '=='.

V PHP nemusíme proměnné před jejich použitím deklarovat, což šetří psaní kódu. Snadno se však stane, že proměnnou před jejím použitím nezinicializujeme (třeba proto, že inicializace se provádí podmíněně). Takto vzniklá chyba se v dlouhých skriptech odhaluje obtížně, ale v této situaci nám dokáže účinně pomoci samo PHP. Pomocí konfigurační direktivy `error_reporting`[2] nebo pomocí funkce `Error_Reporting()` můžeme aktivovat vypisování varovných hlášení. Pokud hodnotu nastavíme na 15, jsou kromě chybových hlášení vypisována i varování mezi něž patří použití nezinicializovaných proměnných. Při vývoji aplikací vřele doporučuji tuto volbu používat a po odladění aplikace a jejím ostrém spuštění ji vypnout.[3]

Další častou chybou je omylem použité přiřazení místo porovnání. Například chceme část skriptu provést podmíněně a použijeme:

```
if ($a=10)
    echo "\$a je 10\n";
```

s hrůzou potom zjistíme, že tělo podmínky se vykoná vždy. Výraz `$a=10` totiž přiřadí do proměnné `$a` hodnotu 10 a tuto hodnotu vrátí i jako svůj výsledek. Číslo 10 je však interpretováno jako hodnota `true` a podmínka je splněna vždy. Na tuto chybu musíme přijít sami.

Co dělat, když skript nedělá to, co by měl, a my v něm žádnou chybu nevidíme. V tom případě můžeme na různých místech skriptu použít funkci `echo` a vypisovat si obsahy proměnných, které jsou důležité pro řízení chodu skriptu. Tak můžeme zjistit, že některá proměnná v danou chvíli obsahuje jinou hodnotu, než by měla, a z toho usoudit na místo výskytu chyby. Toto je zatím jediná možná metoda, protože v současné době zatím neexistuje pro PHP debugger, který by umožňoval krokování skriptu po jednotlivých příkazech, kontrolování obsahu proměnných a podobné funkce, na které jsme zvyklí z ostatních vývojových prostředí.

---

[2] O tom kde a jak nastavit konfigurační direktivy se dočtete v jedenácté kapitole.

[3] Znamená to tedy, že během ladění aplikace budeme mít v konfiguračním souboru `php3.ini` příkaz `error_reporting = 15`.

4.3 Ošetření chyb

Při volání funkcí v PHP může dojít k chybě — například neexistuje soubor, který chceme otevřít, nelze se spojit s databázovým serverem, ze kterého chceme číst data apod. V tomto případě PHP normálně do generované stránky vypíše chybové hlášení. Profesionálně napsané aplikace by měly uživatele od těchto hlášení odstínit. Proto lze v PHP vypnout generování chybových hlášení.

Hlášení můžeme vypnout buď pro celý skript pomocí konfigurační direktivy `error_reporting` nebo funkce `Error_Reporting()`. Druhou možností je vypnout generování chybových hlášení pouze pro volání nějaké konkrétní funkce nebo pro nějaký příkaz.

Pokud chceme, aby nebylo nějakou funkcí generováno chybové hlášení, zapíšeme před její volání zavináč '`@`'. Zda došlo k chybě, musíme ošetřit sami. Většinou využijeme toho, že neúspěšně provedená funkce vrací hodnotu `false`. Pokud například v našich skriptech pracujeme se soubory, měli bychom je otevírat následujícím bezpečným způsobem:

```
$fp = @FOpen("soubor.txt", "r");
if (!$fp):
    echo "Nepodařilo se otevřít datový soubor. Kontaktujte
          správce aplikace!!!\n";
    exit;
endif;
```

Pokud jeden příkaz volá více funkcí, můžeme potlačit vypisování chybových zpráv uvedením znaku '`@`' na začátku příkazu:

```
@$line = FGetS($fp1, 512) . FGetS($fp2, 512) . FGetS($fp3, 512);
```

Pokud chceme mít přehled o tom, kde v našich aplikacích dochází k chybám, můžeme si chyby zapisovat do protokolu chyb. Pokud navíc aktivujeme konfigurační direktivu `track_errors`, budeme mít v proměnné `$php_errormsg` k dispozici text posledního chybového hlášení. Do protokolu chyb můžeme kromě chyby zapsat i číslo řádku a jméno skriptu, ve kterém došlo k chybě:

```
OpenLog("PHP", LOG_PID|LOG_CONS, 0);
@$spojeni = ODBC_Connect("testdb", "", "");
if (!$spojeni):
    SysLog(LOG_ERR,
           $php_errormsg." on line ".(__LINE__-2)." in file ".__FILE__);
endif;
```

Pokud nemáme čas důkladně testovat skripty, je jednou z možností automatický záznam všech chyb do protokolu chyb. Stačí zapnout direktivu `log_errors` a všechna chybová hlášení se budou ukládat do systémového protokolu. Zároveň můžeme pomocí `error_reporting` vypnout generování chybových hlášení

pro koncové uživatele. Řešení to však není zdaleka nejlepší, protože takový napůl funkční skript, který nevypisuje žádná chybová hlášení, může pěkně zmást uživatele.

4.4 Interní debugger

Ačkoliv v době psaní knihy nejsou k dispozici žádné uživatelsky přítulné debuggery, již nyní PHP obsahuje interní debugger. Pokud je debugger zapnut, na určitý TCP port jsou vypisována všechna chybová hlášení včetně souboru a čísla řádky. Z tohoto portu může data číst nějaký program, který je uživateli zpřístupní v použitelné podobě.

V dnešní podobě je interní debugger téměř nepoužitelný, takže zbytek kapitoly radši ani nečtěte.

Pokud si chcete prohlédnout, co za informace je na port vypisováno, můžete debugger zapnout pomocí direktivy `debugger.enabled` a pomocí příkazu

```
socket -l -s 1400
```

sledovat všechny údaje zapisované interpretem PHP na port debuggeru. Na serveru projektu PHP je k dispozici verze programu `socket` určená i pro Windows.

5. Formuláře

Pro vytvoření opravdové aplikace v prostředí Internetu potřebujeme do hry aktivně zapojit uživatele. Aplikace musí od uživatele získat vstupní informace, aby mu mohla poskytnout požadované údaje. Jazyk HTML proto obsahuje podporu formulářů. Součástí webové stránky se tak stanou různá vstupní pole a tlačítka — vznikne nám něco velice podobné dialogovým oknům. Uživatel do formuláře zadá data, která jsou odeslána skriptu na serveru ke zpracování. Skript získaná data nějakým způsobem zpracuje a poskytne uživateli odpověď. Vidíme, že bez formulářů by uživatel neměl příliš šanci aktivně vstoupit do dění. V této kapitole se proto podrobně seznámíme s tvorbou formulářů v jazyce HTML a se zpracováním dat z formulářů právě pomocí skriptů v PHP. Stranou nenecháme ani možnosti kontroly dat vkládaných uživatelem do formuláře.

5.1 Byrokratické základy

Pro vložení formuláře do stránky slouží HTML element `FORM`. Pro správnou funkci celého formuláře jsou nezbytné dva atributy — `ACTION` a `METHOD`.

Atribut `ACTION` určuje URL skriptu, který se použije pro zpracování dat z formuláře. V PHP je obvykle stránka s formulářem i skript v jednom adresáři a s výhodou použijeme samotné jméno skriptu, které v tomto případě odpovídá relativnímu URL.

Pomocí atributu `METHOD` určujeme způsob, jakým budou data z formuláře předána zpět serveru. K dispozici jsou dvě metody — GET a POST. Metoda GET je vhodná pro přenášení kratších dat z malých formulářů, protože data formuláře se připojí na konec URL ukazujícího na obslužný skript.

Metoda POST se hodí pro odesílání větších formulářů, které obsahují mnoho dat. Údaje z formuláře jsou v tomto případě přenášeny v těle HTTP požadavku.

Z hlediska psaní skriptů v PHP není mezi oběma metodami žádný rozdíl — data jsou vždy přístupná stejným způsobem.

Výše zmíněné poznatky shrneme a ukážeme si, jak má vypadat základ každého formuláře:

```
<FORM ACTION="«URL obslužného skriptu»" METHOD="«metoda»">
«definice formuláře»
</FORM>
```

V definici samotného formuláře můžeme použít téměř libovolné elementy HTML — styly písma, tabulky, obrázky a mnoho dalšího. Kromě toho můžeme použít i speciální elementy, které slouží pro vytvoření různých vstupních polí a tlačítek. Formuláře jsou součástí HTML již dlouho (od verze 2.0) a dnes je

podporují snad všechny prohlížeče. Ve formulářích můžeme používat následující tři elementy:

- element `INPUT` slouží pro definici většiny prvků — vstupních polí, polí pro zadání hesla, zaškrtávacích polí (checkboxes), přepínacích tlačítek (radio buttons), tlačítek pro odeslání a smazání formuláře, skrytých polí, odesílání souboru a tlačítek s obrázkem;
- `SELECT` umožňuje vytvořit seznamy, ze kterých je možno vybírat jednu i více položek;
- `TEXTAREA` slouží k vytvoření vstupního pole pro víceřádkový text.

Vývoj se však nezastavil a jak jistě víte, dnes aktuální verze HTML nese číslo 4.0. Právě HTML 4.0 přidalo několik novinek pro tvorbu formulářů — budeme se jim věnovat v samostatné sekci, protože je dnes ještě většina prohlížečů nepodporuje. Již nyní si můžeme říci, že stránky vyhovující standardu HTML 4.0 mohou ve formuláři navíc obsahovat elementy `OPTGROUP`, `LABEL`, `FIELDSET` a `BUTTON`.

Nyní se podíváme na to, jak můžeme pomocí tří výše zmíněných elementů `INPUT`, `SELECT` a `TEXTAREA` vytvářet všechny možné formulářové prvky.

5.2 Základní prvky formulářů

Nejčastěji používaným elementem ve formulářích je `INPUT`. Je to mimo jiné dáno tím, že může vystupovat v mnoha různých podobách v závislosti na hodnotě atributu `TYPE`. Tento atribut určuje typ vstupního prvku, který se ve formuláři objeví.

Všechny prvky formuláře však mají jednu shodnou vlastnost — mají své jméno. Prvku jméno přiřadíme pomocí atributu `NAME`. Pro nás je důležité, že pod tímto jménem je obsah prvku dostupný v obslužném skriptu, jak si za chvíli ukážeme.

Podívejme se však na jednotlivá vstupní pole.

Vstupní pole pro krátký text <INPUT TYPE=TEXT...>

Vstupní pole slouží k zadání krátkého textu, např. klíčových slov, jména či adresy elektronické pošty. Pokud u elementu `INPUT` neuvedeme atribut `TYPE`, předpokládá se, že jde právě o tento druh vstupního prvku.

Maximální délka zadávaného textu není omezena, můžeme ji však omezit použitím atributu `MAXLENGTH`. Velikost vstupního pole lze určit atributem `SIZE` — jako hodnota se udává počet znaků, které má pole pojmout. Pokud je `MAXLENGTH` větší než `SIZE`, nic se neděje — zadávaný text bude ve vstupním poli rolovat. Pokud chceme, aby v poli byla nějaká počáteční hodnota, můžeme ji specifikovat

pomocí atributu VALUE. Jednoduchý formulář pro zadání jména tedy vytvoříme velice snadno:

```
<FORM ACTION="obsluha.php" METHOD=GET>
Jméno: <INPUT TYPE=TEXT NAME=Jmeno VALUE="Sem napište své jméno">
</FORM>
```

Vidíme, že popisek vstupního pole formuláře je potřeba zapsat zcela klasicky. Vstupní pole vyhradí pouze prostor na samotné zapsání hodnoty. Možnost míchat dohromady prvky formuláře s ostatními elementy lze šikovně využít. Pokud chceme mít jednotlivá políčka formuláře pěkně zarovnaná, můžeme použít tabulku. V jednom sloupci budou uvedeny popisky polí a ve druhém samotná vstupní pole:

```
<FORM ACTION="obsluha.php" METHOD=POST>
<TABLE FRAME=BOX RULES=NONE BORDER=1 CELLPADDING=4>
<TR><TD>Jméno:    <TD><INPUT TYPE=TEXT NAME=Jmeno SIZE=30>
<TR><TD>Příjmení: <TD><INPUT TYPE=TEXT NAME=Prijmeni SIZE=30>
<TR><TD>E-mail:   <TD><INPUT TYPE=TEXT NAME=Email SIZE=20>
</TABLE>
</FORM>
```

Tlačítko pro odeslání formuláře <INPUT TYPE=SUBMIT...>

Asi druhým nejpoužívanějším prvkem formuláře je tlačítko sloužící k odeslání formuláře. K vytvoření tlačítka slouží opět element INPUT, musíme však použít atribut TYPE=SUBMIT. Na tlačítku se objeví nápis, který nastavujeme pomocí atributu VALUE:

```
<INPUT TYPE=SUBMIT VALUE="Odeslání dotazníku">
```

Nyní již můžeme sestrojit první opravdu funkční formulář. Ukážeme si na něm, jak jsou v PHP přístupné obsahy vstupních polí. Vytvoříme jednoduchý formulář:

```
<FORM ACTION="obsluha.php" METHOD=GET>
Jméno: <INPUT TYPE=TEXT NAME=Jmeno><BR>
<INPUT TYPE=SUBMIT VALUE="Odešli">
</FORM>
```

Po vyplnění jména může formulář uživatel odeslat. V tom okamžiku je však vyvolán skript `obsluha.php`, který se má postarat o zpracování údajů z formuláře.

Práce s daty z formuláře je velice jednoduchá, protože PHP nám automaticky vytvoří pro každé pole formuláře proměnnou, ve které je přístupná hodnota zadaná uživatelem. V našem případě máme ve skriptu `obsluha.php` k dispozici proměnnou `$Jmeno`. Část skriptu `obsluha.php` proto může vypadat třeba takto:

```
Mám tě! Odeslal jsi formulář a jmenuješ se <?echo $Jmeno?>.
```

Vraťme se však k tlačítku pro odesílání formuláře — nabízí nám ještě jednu zajímavou možnost. Pokud u odesílacího tlačítka definujeme i atribut `NAME`, můžeme v jednom formuláři použít několik odesílacích tlačítek najednou. S daty formuláře se totiž odešle i informace o tom, které tlačítko bylo stisknuté. Ve skriptu pak máme k dispozici proměnnou se jménem stejným jako obsah atributu `NAME`. Hodnota proměnné odpovídá textu uvedenému v atributu `VALUE` tlačítka, kterým byl formulář odeslán. Malá ukázka části formuláře:

```
Souhlasím se zněním vyplněné smlouvy:
<INPUT TYPE=SUBMIT NAME="Potvrzeni" VALUE="Ano">
<INPUT TYPE=SUBMIT NAME="Potvrzeni" VALUE="Ne">
```

Souhlasím se zněním vyplněné smlouvy: Ano Ne

V obslužném skriptu pak snadno zjistíme, kterým tlačítkem byl formulář odeslán:

```
if ($Potvrzeni=="Ano")
    echo "Uděláme něco...";
else
    echo "Nebudeme dělat nic...";
```

Heslo <INPUT TYPE=PASSWORD...>

Použití a chování tohoto prvku je stejné jako u textového pole (`TYPE=TEXT`). Pouze při psaní do pole jsou místo znaků zobrazovány hvězdičky. Nikdo nám tak nemůže přes rameno přečíst heslo. Ve skriptu je obsah pole přístupný v proměnné, která odpovídá názvu pole pro zadání hesla.

Heslo: ********

Zaškrtávací pole <INPUT TYPE=CHECKBOX...>

Tento prvek slouží pro vstup logických hodnot. Prohlížečem bývá zobrazován jako malé okénko, které je buď prázdné, nebo zaškrtnuté. Kromě jména (`NAME`) musíme vždy nastavit i atribut `VALUE`. Pokud je pole zaškrtnuté, pošle se jeho jméno společně s hodnotou atributu `VALUE` serveru. Pokud pole není zaškrtnuto, neposílá se z tohoto prvku formuláře nic.

Malá ukázka zaškrtávacích polí:

```
Do kávy si dáte:<BR>
<INPUT TYPE=CHECKBOX NAME=Cukr VALUE=Ano> cukr<BR>
<INPUT TYPE=CHECKBOX NAME=Mleko VALUE=Ano> mléko<BR>
```

Do kávy si dáte:
☑ cukr
☐ mléko

V obslužném skriptu budeme mít k dispozici proměnné `$Cukr` a `$Mleko`, které budou obsahovat řetězec `Ano` v případě, že uživatel před odesláním formuláře zaškrtl příslušné pole:

```
if ($Cukr=="Ano") OsladKavu();
if ($Mleko=="Ano") PridejDoKavyMleko();
```

V jednom formuláři můžeme použít více zaškrtávacích polí se stejným jménem. V tomto případě může uživatel zaškrtnout libovolné z těchto polí nezávisle na sobě. Skriptu jsou posílány dvojice jméno a hodnota všech zaškrtlých polí (se stejným jménem tedy může dorazit více položek). Aby se jednotlivé položky nepřebily, je potřeba seznam zaškrtlých polí předávat jako pole. Ve jméně zaškrtávacího pole proto na konci použijeme prázdné hranaté závorky — PHP pak automaticky z příchozích dat vytvoří pole. Malá ukázka:

```
<FORM ACTION=obsluha.php METHOD=GET>
<TABLE FRAME=BORDER RULES=NONE CELLPADDING=4>
<CAPTION>Vyberte si doplňující konfiguraci počítače:</CAPTION>
<TR><TD><INPUT TYPE=CHECKBOX NAME="Konf[]" VALUE=CD-ROM>Mechanika CD-ROM
<TR><TD><INPUT TYPE=CHECKBOX NAME="Konf[]" VALUE=ZIP>Mechanika ZIP 100MB
<TR><TD><INPUT TYPE=CHECKBOX NAME="Konf[]" VALUE=SOUND>Zvuková karta
<TR><TD><INPUT TYPE=CHECKBOX NAME="Konf[]" VALUE=MODEM CHECKED>Modem
<TR><TH><INPUT TYPE=SUBMIT VALUE="Potvrzení konfigurace">
</TABLE>
</FORM>
```

Vyberte si doplňující konfiguraci počítače:

- Mechanika CD-ROM
- Mechanika ZIP 100MB
- Zvuková karta
- Modem

Potvrzení konfigurace

Pokud chceme, aby bylo některé pole zaškrtnuto ihned po načtení stránky, použijeme atribut CHECKED — stejně jako u pole pro doplnění konfigurace o modem.

Po odeslání formuláře budeme mít ve skriptu obsluha.php k dispozici pole $Konf[], jehož prvky budou obsahovat jména vybraných komponent (CD-ROM, ZIP, SOUND nebo MODEM). Velice snadno tak můžeme zjistit, která pole uživatel zaškrtl:

```
for ($i=0; $i<Count($Konf); $i++):
    if ($Konf[$i]=="CD-ROM"):
        echo "Namontuj CD-ROM!<BR>";
    elseif ($Konf[$i]=="ZIP"):
        echo "Namontuj mechaniku ZIP!<BR>";
    elseif ($Konf[$i]=="SOUND"):
        echo "Namontuj zvukovou kartu!<BR>";
    elseif ($Konf[$i]=="MODEM"):
        echo "Namontuj modem!<BR>";
    endif;
endfor;
```

Přepínací tlačítka <INPUT TYPE=RADIO...>

Tento typ vstupního prvku použijeme v případě, kdy chceme uživateli nabídnout možnost výběru právě jedné z několika variant. Všechny varianty musí mít stejné jméno (NAME) a rozdílnou hodnotu (VALUE). Jedna z variant by měla být označena pomocí atributu CHECKED — ta bude automaticky vybrána při zobrazení stránky.

```
Vyberte si velikost pevného disku:
<BLOCKQUOTE>
<INPUT TYPE=RADIO NAME=Disk VALUE=850 CHECKED>850 MB<BR>
<INPUT TYPE=RADIO NAME=Disk VALUE=1200>1,2 GB<BR>
<INPUT TYPE=RADIO NAME=Disk VALUE=1600>1,6 GB<BR>
<INPUT TYPE=RADIO NAME=Disk VALUE=2100>2,1 GB
</BLOCKQUOTE>
```

Vyberte si velikost pevného disku:

- ◉ 850 MB
- ○ 1,2 GB
- ○ 1,6 GB
- ○ 2,1 GB

V obslužném skriptu pak máme k dispozici proměnnou `$Disk`, která obsahuje jednu z hodnot `850`, `1200`, `1600` nebo `2100`.

Tlačítko pro vynulování formuláře <INPUT TYPE=RESET...>

Po stisku tohoto tlačítka se všechna pole formuláře nastaví na původní hodnoty. Popis tlačítka lze určit pomocí atributu `VALUE`. Toto tlačítko nikdy není posíláno skriptu jako součást dat formuláře. Slouží pouze k ulehčení života uživatele. (Anebo také ke ztížení, pokud si tlačítkem omylem vymaže pracně vyplněný formulář. ;–)

Tlačítko s obrázkem <INPUT TYPE=IMAGE...>

Toto tlačítko slouží k odeslání formuláře podobně jako tlačítko `SUBMIT`. Tlačítko má podobu obrázku, jehož URL zadáme pomocí atributu `SRC`. Zarovnání obrázku s okolím můžeme ovlivnit atributem `ALIGN`, který má stejný význam jako u elementu `IMG` pro vkládání obrázku. Můžeme použít i další atributy běžné u obrázků — např. `ALT` pro popis obrázku.

Tlačítko s obrázkem funguje podobně jako klikací mapa — s ostatními daty formuláře jsou odeslány i informace o místě, kde došlo ke kliknutí. Místo kliknutí je odesláno ve dvou položkách udávajících souřadnici x a y. Tyto složky jsou ve skriptu přístupné v proměnných, jejichž jméno odpovídá obsahu atributu `NAME` doplněnému o `_x`, resp. `_y`.

Opravdoví znalci HTML jsou nyní možná překvapeni, že souřadnice jsou dostupné v proměnných «jméno»`_x` *a «jméno»*`_y`*. Podle specifikace HTML však prohlížeče předávají souřadnice kliknutí pod jmény «jméno»*`.x` *a «jméno»*`.y`*. PHP však nemůže v názvu proměnné obsahovat tečku, a proto ji automaticky převede na podtržítko.*

Vše si ukážeme na příkladě formuláře odesílaného pomocí tlačítka s obrázkem:

```
<FORM ACTION="obsluha.php" METHOD=GET>
<TABLE BORDER=0>
<TR VALIGN=TOP>
<TD>
    Vyberte si ukazatel:
    <BLOCKQUOTE>
    <INPUT TYPE=RADIO NAME=Ukazatel VALUE=Natalita CHECKED>Natalita<BR>
    <INPUT TYPE=RADIO NAME=Ukazatel VALUE=Mortalita>Mortalita<BR>
    <INPUT TYPE=RADIO NAME=Ukazatel VALUE=Rozvodovost>Rozvodovost<BR>
    <INPUT TYPE=RADIO NAME=Ukazatel VALUE=Obyvatelstvo>Počet obyvatel<BR>
    <INPUT TYPE=RADIO NAME=Ukazatel VALUE=Vek>Průměrný věk obyvatel<BR>
    </BLOCKQUOTE>
</TD>
<TD>
    Vyberte si stát:<BR>
    <INPUT TYPE=IMAGE NAME=Mapa SRC="au-mapa.gif" ALT="Mapa Austrálie">
</TD>
</TR>
</TABLE>
</FORM>
```

Vyberte si ukazatel:

- Natalita
- Mortalita
- Rozvodovost
- Počet obyvatel
- Průměrný věk obyvatel

Vyberte si stát:

Ve skriptu pro obsluhu formuláře budou souřadnice kliknutí myší přístupné v proměnných `$Mapa_x` a `$Mapa_y`. Z nich můžeme zjistit, na který stát uživatel kliknul.

Není-li to nezbytně nutné, používání těchto tlačítek se ve formulářích raději vyhneme. V textových prohlížečích je totiž takový formulář úplně k ničemu.

Odeslání souboru <INPUT TYPE=FILE...>

Tento ovládací prvek použijeme v případech, kdy chceme uživateli umožnit odeslání souboru společně se zbytkem formuláře. Obvykle se prvek zobrazí jako vstupní textové pole a tlačítko. Do pole můžeme přímo vepsat jméno souboru. Můžeme proto použít atributy `SIZE` a `MAXLENGTH`, které mají stejný význam jako u polí s typem `TEXT`. Tlačítko slouží k vyvolání dialogu, který nám umožní pohodlné vybrání souboru.

Pomocí atributu `ACCEPT` můžeme určit MIME typy souborů, které je přípustné vybrat. Následující malý formulář může sloužit třeba pro přihlášení k nějakému chatu — zadáme svoje jméno a spolu s ním pošleme naši podobenku. Ta přitom může být buď obrázkem (`image/*`), nebo pseudoobrázkem poskládaným z ASCII-znaků (`text/plain`):

```
<FORM ACTION="login.php" METHOD=POST ENCTYPE="multipart/form-data">
Zadej svoje jméno: <INPUT NAME=Jmeno SIZE=20><BR>
A teď připoj obrázek s tvojí originální podobou:
<INPUT TYPE=FILE NAME=Foto ACCEPT="image/*,text/plain"><BR>
Stiskni <INPUT TYPE=SUBMIT VALUE="OK"> a vítej v našem virtuálním světě.
</FORM>
```

Zadej svoje jméno: jkj
A teď připoj obrázek s tvojí originální podobou: D:\pic\jkj.gif Browse...
Stiskni OK a vítej v našem virtuálním světě.

Vidíme, že u tagu `<FORM>` nám přibyl nový atribut — `ENCTYPE`. Ten určuje způsob kódování dat formuláře při jejich přenosu. Jeho standardní hodnota je `application/x-www-form-urlencoded`. Tento způsob kódování je pro přenos souborů nepoužitelný, a proto musíme použít novější způsob kódování dat `multipart/form-data`. Každé pole formuláře je pak přenášeno jako jedna část MIME zprávy. Pokud používáme kódování `multipart/form-data`, musíme pro odesílání formuláře použít metodu POST.

Otázkou však stále zůstává, jak se k odeslanému souboru dostaneme v PHP. PHP každý soubor odeslaný pomocí formuláře uloží do dočasného souboru v adresáři, který můžeme určit pomocí direktivy `upload_tmp_dir`. Našemu skriptu pak PHP předá čtyři proměnné, které obsahují informace o přeneseném souboru. Jména všech proměnných začínají stejně jako jméno formulářového pole pro zadání jména souboru. V našem případě budou proměnné začínat na `$Foto`.

V proměnné `$Foto` bude dostupné jméno souboru, do kterého nám PHP zaslaný soubor dočasně uložilo. Tento soubor je po dokončení běhu skriptu automaticky smazán, a pokud jej chceme zachovat, musíme jej přejmenovat nebo překopírovat do jiného adresáře.

V proměnné `$Foto_size` je uložena velikost přeneseného souboru v bajtech. MIME-typ souboru získáme v proměnné `$Foto_type`. Konečně proměnná `$Foto_name` obsahuje původní jméno souboru na počítači uživatele.

Použití si ukážeme na jednoduché obsluze našeho formuláře. Získaný soubor s obrázkem přesuneme do adresáře `/data/chat/` a pojmenujeme jej podle jména uživatele:

```
if ($Foto_type=="text/plain")
    Copy($Foto, "/data/chat/$Jmeno.txt");
elseif ($Foto_type=="image/gif)
    Copy($Foto, "/data/chat/$Jmeno.gif");
elseif ($Foto_type=="image/jpeg")
    Copy($Foto, "/data/chat/$Jmeno.jpeg);
else
    echo "Nepodporovaný formát obrázku: $Foto_type.";
```

Skrytá pole <INPUT TYPE=HIDDEN...>

Tato pole se ve formuláři vůbec neobjeví. Slouží k uchování stavové informace, která je odeslána s vyplněným formulářem zpět skriptu. U pole použijeme pouze atributy `NAME` a `VALUE`, které slouží k definovaní stavové proměnné a její hodnoty. Ještě o krůček lepší funkčnost v uchovávání stavové informace nabízejí cookies (viz strana 443). Praktické příklady použití skrytých polí a cookies naleznete v kapitole věnované ukázkám.

Seznamy <SELECT>...</SELECT>

Tento element lze použít v případech, kdy chceme uživateli nabídnout výběr jedné z několika položek. Položky se uvádějí mezi tagy `<SELECT>` a `</SELECT>`. Pokud použijeme atribut `MULTIPLE`, může být ze seznamu najednou vybráno i více položek. Jako u většiny ostatních prvků formuláře, i zde musíme uvést jméno pomocí atributu `NAME`. Počet najednou zobrazených řádek seznamu můžeme nastavit pomocí atributu `SIZE`.

Jednotlivé položky seznamu se uvádějí jako obsah elementu `OPTION`. Ukončovací tag však můžeme vždy vynechat. Položky, u nichž použijeme atribut `SELECTED`, budou již předem vybrány. Jako hodnota vybrané položky se přenáší obsah položky. Pokud je příliš dlouhý, můžeme pomocí atributu `VALUE` nastavit kratší hodnotu, která bude identifikovat položku seznamu. Použití si ukážeme na malé ukázce:

```
Vyberte si čísla, o která máte zájem:
<BLOCKQUOTE>
```

```
<SELECT NAME="Cisla[]" SIZE=4 MULTIPLE>
<OPTION VALUE="1">1/96
<OPTION VALUE="2">2/96
<OPTION VALUE="3">3/96
<OPTION VALUE="4">4/96
</SELECT>
</BLOCKQUOTE>

Vyberte si formát, ve kterém chcete zvolená čísla zaslat:
<BLOCKQUOTE>
<SELECT NAME="Format" SIZE=1>
<OPTION VALUE="HTML">HTML
<OPTION VALUE="PDF" SELECTED>Portable Document Format (PDF)
<OPTION VALUE="PS">PostScript
<OPTION VALUE="ASCII">obyčejný text
</SELECT>
</BLOCKQUOTE>
```

Vyberte si čísla, o která máte zájem:

1/96
2/96
3/96
4/96

Vyberte si formát, ve kterém chcete zvolená čísla zaslat:

Portable Document Format (PDF)

Jelikož si čísel může uživatel objednat více najednou, musíme jako jméno vstupního pole použít `Cisla[]` — v PHP pak budou vybraná čísla přístupná pomocí pole podobně jako u zaškrtávacích tlačítek. U druhého seznamu jsme pole už použít nemuseli, protože uživatel si může vybrat pouze jednu možnost (nepoužili jsme atribut `MULTIPLE`).

Víceřádkový text <TEXTAREA>...</TEXTAREA>

Element `TEXTAREA` slouží pro zadávání delších textů ve formuláři. Jako všechny ostatní elementy má atribut `NAME`. Pomocí atributu `ROWS` můžeme určit počet řádek vstupního pole a pomocí `COLS` počet sloupců. Obsahem elementu je text, který se ve vstupním poli objeví na začátku editace (ukončovací tag tedy nesmí být vynechán):

```
<TEXTAREA NAME="Komentar" ROWS=5 COLS=40>
Zde nám prosím napište připomínky k našemu programu.
</TEXTAREA>
```

```
Zde nám prosím napište připomínky k
našemu programu.
```

V našem skriptu bude zadaný text již tradičně přístupný v proměnné `$Komentar`. Jediný problém může nastat při jeho zobrazování. Pokud uživatel do vstupního pole zadá více řádek ukončených stiskem klávesy ENTER a my chceme obsah pole zobrazit na stránce, musíme převést konce řádků na tagy `<BR>`, které v HTML vyvolají ukončení řádky. Není to však složitý úkol, protože k dispozici máme funkci `NL2BR()`, která potřebnou konverzi obstará za nás:

```
Děkujeme za komentář ve znění:
<BLOCKQUOTE>
<?echo NL2BR($Komentar)?>
</BLOCKQUOTE>
```

5.3 Profesionální formuláře

Při vytváření webovských formulářů máme poměrně velkou volnost. Není tak problém vytvořit formulář, který je pro uživatele nepřehledný, nebo naopak formulář, který je radost vyplňovat. Vytvořit přehledný formulář, zvláště pokud má obsahovat mnoho vstupních polí, je někdy umění. I přesto existuje několik zásad, jejichž dodržování nám pomůže vytvářet pro uživatele přehledné formuláře.

Přehledný formulář má jednotlivá vstupní pole jasně označena a přehledně zarovnána pod sebou. K dosažení tohoto efektu lze využít tabulku — první sloupec bude obsahovat popis vstupního pole a druhý samotné vstupní pole. Podobného efektu můžeme dosáhnout i použitím kaskádových stylů.

Celkem příjemné je, když uživateli napovíme, v jaké formě má data do jednotlivých vstupní polí zadávat. Do tabulky můžeme například přidat třetí sloupec, který bude znázorňovat ukázkově vyplněná data. Druhou možností je zobrazení ukázkových dat přímo ve vstupním poli s využitím atributu `VALUE`. Uživatel pak pouze přepíše předpřipravené údaje.

Pro uživatele je příjemné, pokud vidí celý vyplňovaný formulář najednou na obrazovce. Pokud potřebujeme od uživatele tolik údajů, že se formulář nevejde najednou na obrazovku, je lepší vytvořit několik stránek s menšími formuláři, které postupně od uživatele vyzvědí požadované údaje.

Mnoho formulářů na Webu (např. různé registrace) vyžadují vyplnění pouze některých povinných částí formuláře. V těchto případech je vhodné povinná pole nějak odlišit — např. použitím výraznějšího písma nebo barevného písma u popisky pole.

Pokud obsluhující skript zjistí, že uživatel nevyplnil všechny údaje, požaduje doplnění chybějících. V tomto případě by se před uživatelem měl objevit formulář s již dříve zadanými daty. Bohužel na mnoha stránkách se v tomto případě objeví nový prázdný formulář a uživatel musí zadávat vše znovu. Takový formulář i v poměrně klidném uživateli probudí tolik agrese vůči tvůrci stránky, že jediným autorovým štěstím je jeho úkryt za několika firewally a kilometry optických kabelů.

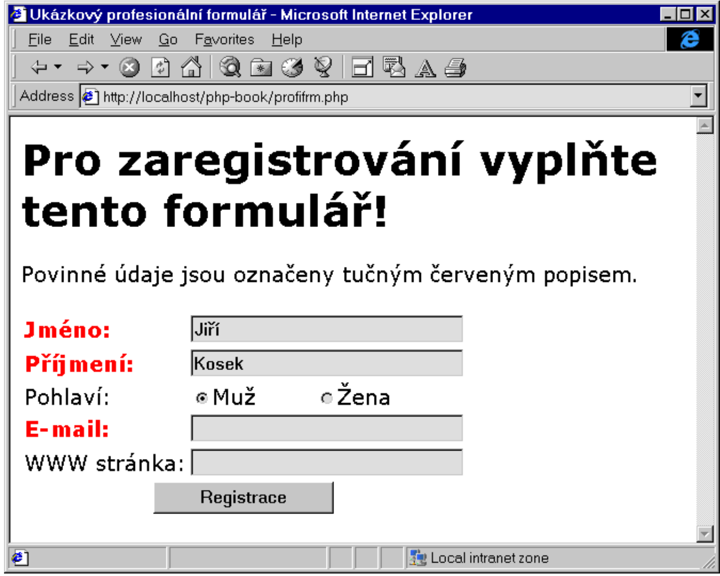
My si proto ukážeme, jak vytvořit formulář, který nefrustruje uživatele. Na formuláři bude na první pohled zřejmé, která pole se musí vyplnit. Pokud uživatel na nějaké pole zapomene, formulář si vyžádá doplnění pouze těch chybějících — nebude nutit uživatele vyplňovat vše znovu.

Příklad: `profifrm.php`

```
<HTML>
<HEAD>
<TITLE>Ukázkový profesionální formulář</TITLE>
<STYLE TYPE="text/css">
<!--
TD B { color: red }     /* Tučné písmo v buňce tabulky bude navíc červené */
-->
</STYLE>
</HEAD>
<BODY>
<?
$zobrazitFormular = true;   // příznak zobrazování formuláře

if ($Odeslano):             // byl již formulář odeslán?

  if ($Jmeno=="" ||         // kontrola vyplnění povinných údajů
      $Prijmeni=="" ||
      $Email==""):

    echo "<H1>Musíte vyplnit všechny povinné údaje!</H1>";
    echo "Jsou označeny tučným červeným popisem.";

  else:                     // máme povinná data, zpracujeme je

    $zobrazitFormular = false;
    echo "<H1>Děkujeme za registraci!</H1>";
    // Zde se mohou získané informace libovolně zpracovat

  endif;
```

```
else:                      // formulář je zobrazován poprvé

  echo "<H1>Pro zaregistrování vyplňte tento formulář!</H1>";
  echo "Povinné údaje jsou označeny tučným červeným popisem.";

endif;
?>

<?if ($zobrazitFormular):     // má se zobrazovat formulář? ?>
    <FORM METHOD=POST>
    <TABLE BORDER=0>
    <TR>
        <TD><B>Jméno:</B>
        <TD><INPUT NAME=Jmeno VALUE="<?echo $Jmeno?>" SIZE=30>
    <TR>
        <TD><B>Příjmení:</B>
        <TD><INPUT NAME=Prijmeni VALUE="<?echo $Prijmeni?>" SIZE=30>
    <TR>
        <TD>Pohlaví:
        <TD><INPUT TYPE=RADIO NAME=Pohlavi VALUE="Muz"
                   <?echo $Pohlavi=="Muz" ? " CHECKED" : ""?>>Muž
                  
            <INPUT TYPE=RADIO NAME=Pohlavi VALUE="Zena"
                   <?echo $Pohlavi=="Zena" ? " CHECKED" : ""?>>Žena
    <TR>
        <TD><B>E-mail:</B>
        <TD><INPUT NAME=Email VALUE="<?echo $Email?>" SIZE=30>
    <TR>
        <TD>WWW stránka:
        <TD><INPUT NAME=Web VALUE="<?echo $Web?>" SIZE=30>
    <TR>
        <TH COLSPAN=2><INPUT TYPE=SUBMIT VALUE="    Registrace    ">
    </TABLE>
    <INPUT TYPE=HIDDEN NAME=Odeslano VALUE=true>
    </FORM>
<?endif?>
</BODY>
</HTML>
```

Celý skript si zajistí jistě malý komentář. Výsledkem skriptu mohou být tři odlišné stránky — viz obrázky 5-1, 5-2 na následující straně a 5-3 na následující straně. Pokud je skript vyvolán uživatelem poprvé, zobrazí se prázdný formulář. Pokud uživatel formulář odešle, mohou nastat dva případy. Pokud jsou všechny povinné údaje vyplněny, skript zobrazí pouze potvrzení registrace. Pokud nějaký povinný údaj chybí, zobrazí se formulář, který obsahuje již dříve vyplněná pole a požaduje doplnění chybějících polí.



Obr. 5-1: Do formuláře jsem nevyplnil všechny povinné údaje

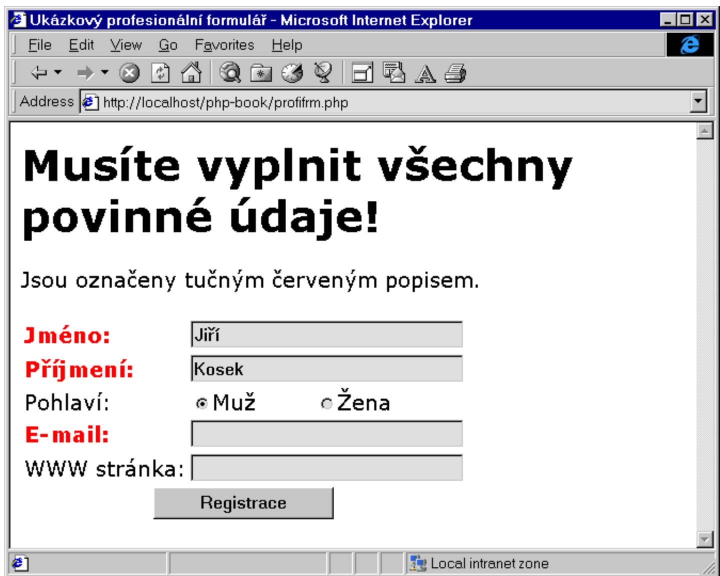
Skript na obsluhu dat z formuláře volá sám sebe, a proto nemusíme u elementu `FORM` uvádět pomocí atributu `ACTION` jméno skriptu určeného pro zpracování formuláře.

Jak skript pozná, zda je uživatelem vyvolán poprvé nebo zda jen uživatel nevyplnil žádné pole? Do formuláře jsme přidali skryté pole (`TYPE=HIDDEN`), kterým nastavíme proměnnou `$Odeslano` na hodnotu `true`. Toto pole se nezobrazuje, ale jeho hodnota se přenáší s ostatními údaji z formuláře. Podmínkou

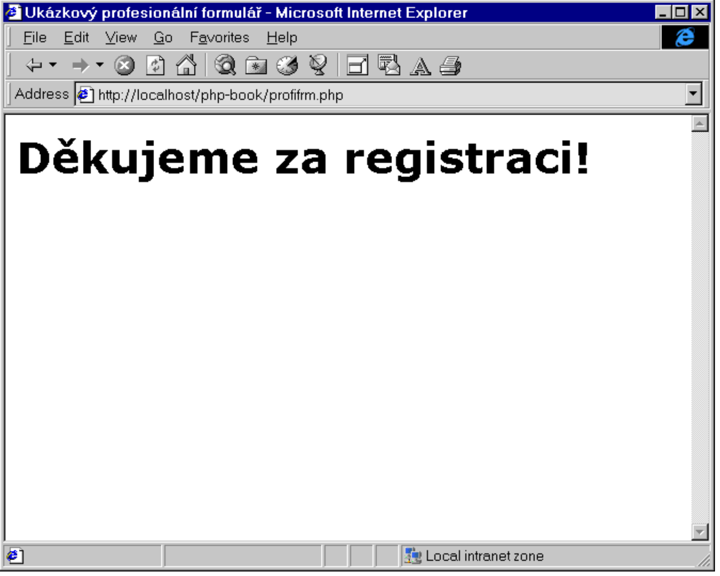
```
if ($Odeslano):
```

tak snadno otestujeme, zda do skriptu již putují údaje z formuláře nebo zda je spouštěn poprvé.

V další části skriptu testujeme, zda všechna povinná pole obsahují nějakou hodnotu. Pokud ne, vypíšeme záhlaví stránky, které požaduje doplnění chybějících údajů. Pokud jsou všechna povinná pole vyplněna, poděkuje skript za vyplnění formuláře.



Obr. 5-2: Jsem požádán o doplnění chybějících údajů — doplním e-mailovou adresu



Obr. 5-3: Zdá se, že se vše podařilo

Další větev skriptu patří k první podmínce. Provede se pouze tehdy, pokud je skript volán uživatelem poprvé a vyzve ho k vyplnění formuláře.

Následuje samotný HTML kód pro zobrazení formuláře. Tento kód se do prohlížeče odešle pouze tehdy, pokud má proměnná `$zobrazitFormular` hodnotu `true`. Tato proměnná je nastavena na hodnotu `false` tehdy, když skript obdržel všechna povinná pole a my již po uživateli nechceme doplnit žádné údaje.

Jako hodnotu atributu `VALUE` u jednotlivých vstupních polí používáme krátký PHP skript, který doplní obsah pole podle posledního odesílaného formuláře. Tím máme zaručeno, že pokud uživatel vyplní pouze některé údaje a je požádán o doplnění chybějících povinných údajů, nemusí znovu vyplňovat celý formulář. Při prvním spuštění skriptu neobsahují použité proměnné žádnou hodnotu a vstupní pole budou prázdná (viz obr. 5-1 na straně 107).[1]

Zatímco doplnění hodnoty textového vstupního pole je jednoduché, u přepínacích tlačítek je situace trošku složitější. Pokud má být tlačítko zaškrtnuto, je potřeba u příslušného tagu `<INPUT>` uvést atribut `CHECKED`. Použili jsme proto konstrukci:

```
<?echo $Pohlavi=="Muz" ? " CHECKED" : ""?>
```

Pokud jsme tedy odeslali formulář se zaškrtnutým tlačítkem muž, vrátí skript přepínací tlačítko:

```
<INPUT TYPE=RADIO NAME=Pohlavi VALUE="Muz" CHECKED>
```

Pokud nebylo pohlaví muž vybráno, dorazí naopak:

```
<INPUT TYPE=RADIO NAME=Pohlavi VALUE="Muz">
```

a tlačítko tedy zůstane nezaškrtnuté. Obdobně ošetříme i tlačítko pro něžné pohlaví.

Náš formulář se nyní chová již poměrně uživatelsky přítulně a korektně. Pro pohodlí uživatele však můžeme udělat ještě mnohem více. Vyplnění povinných polí formuláře můžeme kontrolovat ještě před odesláním formuláře pomocí klientského skriptu. Výhodou je především rychlost — data jsou zkontrolována okamžitě, nemusí probíhat žádná komunikace se vzdáleným serverem.

Ukážeme si nyní, jak lze jednoduše kombinovat skript v PHP s klientským skriptem v JavaScriptu. JavaScript jsme vybrali, protože se jedná o nejrozšířenější skriptovací jazyk — podporují jej Netscape Navigator i Internet Explorer. Principiálně však můžeme použít libovolný skriptovací jazyk — např. VBScript, Tcl nebo Python. Skript však bude funkční pouze v omezeném spektru prohlížečů.

---

[1] Pokud ve vstupních polích vidíte text `<BR>`, máte patrně nastaven `error_reporting` na hodnotu 15. V místě vypsání neinicializované proměnné (atribut `VALUE` vstupního pole) se pak vypíše hlášení o přístupu k neinicializované proměnné.

Z předchozího vyplývá, že na kontrolu dat klientským skriptem se nemůžeme zcela spolehnout. Data musíme vždy finálně zkontrolovat skriptem v PHP. Některé prohlížeče JavaScript nepodporují vůbec nebo mají jeho podporu vypnutou a uživatel tak může odeslat formulář, který neobsahuje vyplněná všechna povinná pole. To však neznamená, že klientské skripty nemá cenu používat — velké většině uživatelů usnadní a urychlí vyplňování formulářů na našich stránkách.

V naší knize bohužel nemáme prostor pro podrobné vysvětlování všech tajů a fines JavaScriptu. Ostatně na toto téma bylo vydáno již několik obsáhlých knih. V příkladě použijeme tedy pouze nezbytné obraty JavaScriptu a ty vysvětlíme.

Kontrolování správnosti dat ve formuláři pomocí JavaScriptu spočívá ve vytvoření funkce, která kontroluje správnost jednotlivých polí formuláře, a ve vytvoření události, která funkci aktivuje v okamžiku odesílání formuláře pomocí tlačítka Submit.

Nejprve tedy vytvoříme funkci `validate()`, která vrací `true`, pokud jsou vyplněny všechny povinné údaje. Pokud nějaký údaj bude chybět nebo bude ve špatném formátu, vypíše funkce hlášení, nastaví kurzor do chybného pole a vrátí `false`.

Funkce `validate()` bude mít jeden parametr `formular`, do kterého budeme předávat jméno formuláře, který chceme zkontrolovat. Pomocí objektového modelu JavaScriptu pak v proměnných `formular.Jmeno.value`, `formular.Prijmeni.value` a `formular.Email.value` získáme obsah jednotlivých polí formuláře.

Pokud bude některé pole prázdné, zobrazíme varovné okénko pomocí funkce `alert()`. Následně vyvoláme metodu `focus`, která způsobí přesun kurzoru do pole, které nesplnilo naše požadavky.

Funkce `validate()` může vypadat zhruba takto:

```
function validate(formular)
{
    if (formular.Jmeno.value=="")
    {
        alert("Jméno musíte vyplnit!");
        formular.Jmeno.focus();
        return false;
    }
    else if (formular.Prijmeni.value=="")
    {
        alert("Příjmení musíte vyplnit!");
        formular.Prijmeni.focus();
        return false;
```

```
        }
        else if (formular.Email.value=="")
        {
            alert("Adresu elektronické pošty musíte vyplnit!");
            formular.Email.focus();
            return false;
        }
        else
            return true;
    }
```

Nyní musíme s naší funkcí spojit událost, která se vyvolá při pokusu o odeslání formuláře. Při odesílání formuláře je vyvolána událost `onSubmit`. Pokud je jejím výsledkem hodnota `true`, je formulář odeslán. Pokud však obsluha události vrací `false`, formulář zůstane neodeslán a uživatel může pokračovat v práci s formulářem. Obsluhu události k formuláři přidáme následovně:

```
<FORM ACTION=«skript» METHOD=«metoda» onSubmit="return validate(this)">
```

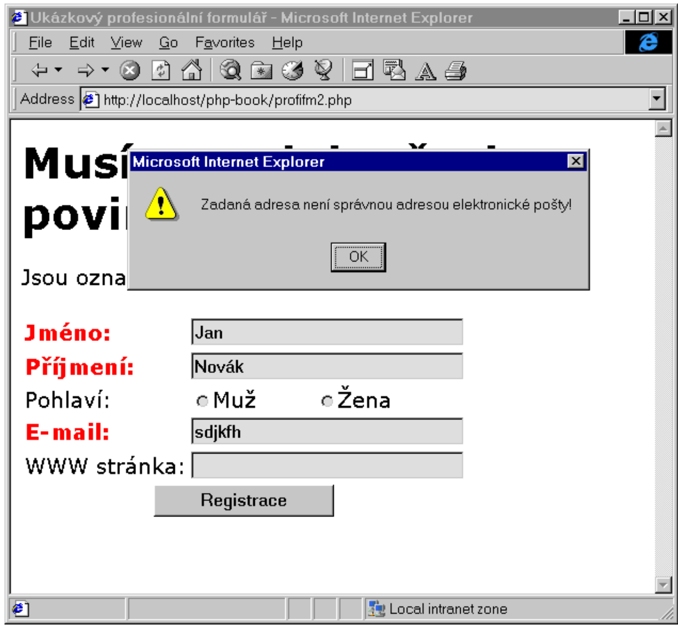
Výraz `this` v tomto případě zastupuje aktuální formulář, který předáváme funkci `validate()` k ověření. Náš skript tedy získá novou podobu:

```
<HTML>
<HEAD>
<TITLE>Ukázkový profesionální formulář</TITLE>
<STYLE TYPE="text/css">
<!--
TD B { color: red }     /* Tučné písmo v buňce tabulky bude navíc červené */
-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript"><!--
function validate(formular)
{
    if (formular.Jmeno.value=="")
    {
        alert("Jméno musíte vyplnit!");
        formular.Jmeno.focus();
        return false;
    }
    else if (formular.Prijmeni.value=="")
    {
        alert("Příjmení musíte vyplnit!");
        formular.Prijmeni.focus();
        return false;
```

```
    }
    else if (formular.Email.value=="")
    {
        alert("Adresu elektronické pošty musíte vyplnit!");
        formular.Email.focus();
        return false;
    }
    else if (window.RegExp)
    {
        re = new RegExp("^[^@]+@[^.]+\..+$");
        if (!re.test(formular.Email.value))
        {
            alert("Zadaná adresa není správnou adresou elektronické pošty!");
            formular.Email.focus();
            return false;
        }
    }
    else
        return true;
}
// -->
</SCRIPT>
</HEAD>
<BODY>
<?
$zobrazitFormular = true;   // příznak zobrazování formuláře

if ($Odeslano):             // byl již formulář odeslán?

  if ($Jmeno=="" ||         // kontrola vyplnění povinných údajů
      $Prijmeni=="" ||
      $Email==""):

    echo "<H1>Musíte vyplnit všechny povinné údaje!</H1>";
    echo "Jsou označeny tučným červeným popisem.";

  else:                     // máme povinná data, zpracujeme je

    $zobrazitFormular = false;
    echo "<H1>Děkujeme za registraci!</H1>";
    // Zde se mohou získané informace libovolně zpracovat
```

```
  endif;

else:                      // formulář je zobrazován poprvé

  echo "<H1>Pro zaregistrování vyplňte tento formulář!</H1>";
  echo "Povinné údaje jsou označeny tučným červeným popisem.";

endif;
?>

<?if ($zobrazitFormular):     // má se zobrazovat formulář? ?>
    <FORM METHOD=POST onSubmit="return validate(this)">
    <TABLE BORDER=0>
    <TR>
        <TD><B>Jméno:</B>
        <TD><INPUT NAME=Jmeno VALUE="<?echo $Jmeno?>" SIZE=30>
    <TR>
        <TD><B>Příjmení:</B>
        <TD><INPUT NAME=Prijmeni VALUE="<?echo $Prijmeni?>" SIZE=30>
    <TR>
        <TD>Pohlaví:
        <TD><INPUT TYPE=RADIO NAME=Pohlavi VALUE="Muz"
                   <?echo $Pohlavi=="Muz" ? " CHECKED" : ""?>>Muž
                  
            <INPUT TYPE=RADIO NAME=Pohlavi VALUE="Zena"
                   <?echo $Pohlavi=="Zena" ? " CHECKED" : ""?>>Žena
    <TR>
        <TD><B>E-mail:</B>
        <TD><INPUT NAME=Email VALUE="<?echo $Email?>" SIZE=30>
    <TR>
        <TD>WWW stránka:
        <TD><INPUT NAME=Web VALUE="<?echo $Web?>" SIZE=30>
    <TR>
        <TH COLSPAN=2><INPUT TYPE=SUBMIT VALUE="    Registrace    ">
    </TABLE>
    <INPUT TYPE=HIDDEN NAME=Odeslano VALUE=true>
    </FORM>
<?endif?>
</BODY>
</HTML>
```

Funkci `validate()` jsme v něm ještě mírně rozšířili, takže kontroluje, zda uživatelem zadaná e-mailová adresa má tvar *«něco»*`@`*«něco»*`.`*«něco»*. To alespoň některé uživatele odradí od zadání e-mailové adresy typu `sdkhghjks`, jak vidíme na obrázku 5-4.



Obr. 5-4: Formulář a jeho kontrola pomocí JavaScriptu

Připadá vám, že v posledním příkladě jsme za hodně peněz měli málo muziky? Vězte však, že právě podle detailů, kterých si na první pohled nikdo nevšimne a které usnadňují uživateli práci, se poznají opravdu profesionální stránky a internetové aplikace.

5.4 Rozšíření formulářů z dílny HTML 4.0

HTML 4.0 přineslo do oblasti formulářů několik novinek. Jejich použití nikterak neovlivňuje zpracování dat z formulářů ve skriptech. Všechny novinky směřují k možnosti vytváření formulářů, se kterými se v prohlížeči lépe pracuje. My se o nich zmíníme, protože ve většině knih věnovaných jazyku HTML ještě nejsou popsány.

> Jestliže se vám stane, že některá z dále popsaných věcí vám nebude fungovat, váš prohlížeč ještě nepodporuje HTML 4.0. Ve „čtyřkových" verzích prohlížečů byla podpora HTML 4.0 velice slabá. Snad se tedy pořádné podpory dočkáme v „pětkových" verzích Netscape Navigatoru a Internet Exploreru.

Nové druhy tlačítek

HTML 4.0 přidalo nový druh vstupního prvku `INPUT`. Pokud nyní použijeme atribut `TYPE=BUTTON` získáme tlačítko, jehož popis můžeme určit atributem `VALUE`. Tlačítko samo o sobě nemá žádnou funkčnost — nelze jím formulář odeslat ani smazat. Tlačítko však můžeme pomocí události spojit s klientským skriptem — typicky napsaným v JavaScriptu. Skript pak může na stisk tlačítka podle potřeby zareagovat. Pokud však chceme vytvářet skutečně přenositelné aplikace, neměli bychom je stavět na této vlastnosti — ne všechny prohlížeče podporují JavaScript. Některé prohlížeče, jako textový Lynx, podporu vůbec neobsahují a v těch ostatních lze podporu JavaScriptu z bezpečnostních důvodů vypnout. Praktické uplatnění nalezne tlačítko snad jen v intranetových řešeních, kde můžeme zajistit podporu JavaScriptu na všech klientských stanicích.

O mnoho užitečnější je nový element `<BUTTON>`. Jeho obsah se zobrazí jako popis tlačítka — na tlačítku tedy můžeme kombinovat text i obrázky. Samotný typ tlačítka určíme pomocí atributu `TYPE` podobně jako u elementu `INPUT`. Přípustné hodnoty jsou `SUBMIT`, `RESET` a `BUTTON`:

```
<BUTTON TYPE=SUBMIT>
<IMG SRC=check.gif WIDTH=32 HEIGHT=31> <B>OK</B>
</BUTTON>
```

Popisky vstupních polí, horké klávesy a tabulátor

Nově nyní můžeme popisku vstupního pole označit elementem `LABEL` a pomocí atributu `FOR` ji s polem spojit. Jako hodnota atributu `FOR` se uvádí `ID` vstupního pole. Toto spojení popisky se vstupním polem se využije při převodu do mluvené řeči:

```
<LABEL FOR=Jmeno>Jméno:</LABEL>
<INPUT TYPE=TEXT ID=Jmeno NAME=Jmeno>
```

Ještě větší uplatnění nalezne element LABEL ve spojení s atributem ACCESSKEY. Jako hodnota tohoto atributu se uvádí písmeno, které bude použito pro klávesovou zkratku. Použijeme-li např. u LABEL ACCESSKEY=N, pomocí stisku Alt + N se kurzor okamžitě přesune do odpovídajícího vstupního pole. Malá ukázka:

```
<FORM ACTION="obsluha.php" METHOD=POST>
<TABLE>
<TR><TD><LABEL FOR=Jmeno ACCESSKEY=J><U>J</U>méno:</LABEL>
    <TD><INPUT TYPE=TEXT ID=Jmeno NAME=Jmeno>
<TR><TD><LABEL FOR=Prijmeni ACCESSKEY=P><U>P</U>říjmení:
    </LABEL>
    <TD><INPUT TYPE=TEXT ID=Prijmeni NAME=Prijmeni>
<TR><TD><LABEL FOR=EMail ACCESSKEY=E><U>E</U>-mail:</LABEL>
    <TD><INPUT TYPE=TEXT ID=EMail NAME=EMail>
<TR><TH COLSPAN=2><LABEL FOR=Send ACCESSKEY=O></LABEL>
    <BUTTON TYPE=SUBMIT ID=Send><U>O</U>deslání formuláře
    </BUTTON>
</TABLE>
</FORM>
```

Jméno:
Příjmení:
E-mail:
Odeslání formuláře

V tomto formuláři se mezi jednotlivými vstupními poli můžeme pohybovat pomocí Alt + J, Alt + P a Alt + E. Stiskem Alt + O formulář odešleme. U popisů vstupních polí je dobré vyznačit horkou klávesu například podtržením. Abychom mohli podtrhnout i písmeno klávesové zkratky pro odeslání formuláře, použili jsme pro vytvoření tlačítka element BUTTON místo klasického <INPUT TYPE=Submit...>.

Ve většině operačních systémů se pro přechod mezi jednotlivými vstupními poli formuláře používá klávesa Tab. Přesouvání po vstupních polích probíhá v tom pořadí, v jakém byly do stránky vloženy. Někdy by však bylo logické toto pořadí změnit — například když je složitý formulář formátován do tabulky. Pro tyto účely můžeme nyní u elementů A, AREA, OBJECT, INPUT, SELECT, TEXTAREA a BUTTON použít nový atribut TABINDEX. Jako jeho hodnotu můžeme použít libovolné celé číslo. Klávesou Tab se pak po jednotlivých prvcích pohybujeme tak, aby hodnota TABINDEX postupně rostla.

Sdružování vstupních polí do bloků

Přehlednost větších formulářů můžeme zvýšit tím, že celý formulář rozdělíme do několika logických bloků, kde každý blok sdružuje příbuzná vstupní pole.

Ke sdružení vstupních prvků slouží element `FIELDSET`. Ihned za tagem `<FIELDSET>` můžeme použít element `LEGEND`, který slouží k zadání popisu bloku vstupních prvků. Vše si nejlépe ukážeme na formuláři, který oba elementy využívá:

```
<HTML>
<HEAD>
<TITLE>Formulář2000</TITLE>
<STYLE TYPE="text/css"><!--
FIELDSET  { padding: 8px; }
LEGEND    { color: blue;
            padding-bottom: 6px; }
.text     { position: absolute;
            left: 120px; }
.button   { text-align: center;
            margin: 8px; }
--></STYLE>
</HEAD>
<BODY>

<FORM ACTION=obsluha.php METHOD=POST>
    <FIELDSET>
        <LEGEND ACCESSKEY=O><U>O</U>sobní údaje</LEGEND>
        <LABEL ACCESSKEY=J FOR=Jmeno><U>J</U>méno:</LABEL>
        <INPUT ID=Jmeno NAME=Jmeno TABINDEX=11 CLASS=TEXT><BR>
        <LABEL ACCESSKEY=P FOR=Prijmeni><U>P</U>říjmení:</LABEL>
        <INPUT ID=Prijmeni NAME=Prijmeni TABINDEX=12 CLASS=TEXT><BR>
        <LABEL ACCESSKEY=V FOR=Vek><U>V</U>ěk:</LABEL>
        <INPUT ID=Vek NAME=Vek TABINDEX=13 CLASS=TEXT><BR>
    </FIELDSET>

    <FIELDSET>
        <LEGEND ACCESSKEY="I">V<U>i</U>rtuální adresa</LEGEND>
        <LABEL ACCESSKEY=E FOR=EMail><U>E</U>-mail:</LABEL>
        <INPUT ID=EMail NAME=EMail TABINDEX=21 CLASS=TEXT><BR>
        <LABEL ACCESSKEY=W FOR=Web><U>W</U>eb:</LABEL>
        <INPUT ID=Web NAME=Web TABINDEX=22 CLASS=TEXT><BR>
    </FIELDSET>
```

```
    <FIELDSET>
        <LEGEND ACCESSKEY="Z"><U>Z</U>ájmy</LEGEND>
        <TABLE BORDER=0 WIDTH="100%">
        <TR><TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="Sport">Sport
            <TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="Kultura">Kultura
        <TR><TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="Počítače">Počítače
            <TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="Internet">Internet
        <TR><TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="Erotika">Erotika
            <TD><INPUT TYPE=CHECKBOX NAME="Zajmy[]" VALUE="TV">TV
        </TABLE>
    </FIELDSET>

    <DIV CLASS=button>
    <BUTTON TYPE=SUBMIT>Odešli dotazní<U>k</U></BUTTON>
    </DIV>
</FORM>

</BODY>
</HTML>
```

Osobní údaje
Jméno:
Příjmení:
Věk:

Virtuální adresa
E-mail:
Web:

Zájmy
Sport Kultura
Počítače Internet
Erotika TV

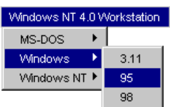
Odešli dotazník

Sdružování položek seznamu do skupin

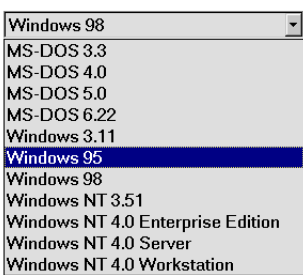
Pokud chceme zpřehlednit seznam (element `SELECT`), můžeme jej nyní hierarchicky rozčlenit pomocí elementu `OPTGROUP`. Celý seznam se pak zobrazí jako hierarchický systém menu. Text, který se objeví jako jednotlivé položky menu můžeme určit pomocí atributu `LABEL`:

```
<SELECT NAME=OS>
<OPTGROUP LABEL=MS-DOS>
  <OPTION LABEL=3.3 VALUE=DOS3.3>MS-DOS 3.3
  <OPTION LABEL=4.0 VALUE=DOS4.0>MS-DOS 4.0
  <OPTION LABEL=5.0 VALUE=DOS5.0>MS-DOS 5.0
  <OPTION LABEL=6.22 VALUE=DOS6.22>MS-DOS 6.22
</OPTGROUP>
<OPTGROUP LABEL=Windows>
  <OPTION LABEL=3.11 VALUE=Win3.11>Windows 3.11
  <OPTION LABEL=95 VALUE=Win95>Windows 95
  <OPTION LABEL=98 VALUE=Win98>Windows 98
</OPTGROUP>
<OPTGROUP LABEL="Windows NT">
  <OPTION LABEL=3.51 VALUE=NT3.51>Windows NT 3.51
  <OPTION LABEL="4.0 Enterprise Edition" VALUE=NT4.0EE>
                                       Windows NT 4.0 Enterprise Edition
  <OPTION LABEL="4.0 Server" VALUE=NT4.0SRV>Windows NT 4.0 Server
  <OPTION LABEL="4.0 Workstation" VALUE=NT4.0WKS>Windows NT 4.0 Workstation
</OPTGROUP>
</SELECT>
```

V prohlížeči, který podporuje HTML 4.0, se seznam zobrazí jako systém menu a submenu:



Ve starších prohlížečích se všechny možnosti zobrazí v jednom lineárním seznamu:



Zakázané ovoce

V praxi poměrně často potřebujeme některé vstupní prvky dočasně vyřadit z provozu. Například tlačítko pro odeslání formuláře by mělo být neaktivní do té doby, než ve formuláři vyplníme všechny povinné údaje.

Tento požadavek uspokojuje atribut `DISABLED`, který můžeme použít u všech formulářových elementů. Pokud u některého z nich atribut použijeme, je prvek vyřazen z provozu — uživatel ho nemůže měnit ani jej aktivovat. Změnu stavu prvku lze vyvolat pouze klientskými skripty.

Podobné chování má atribut `READONLY`, s tím rozdílem, že prvek pouze nelze měnit, ale lze jej aktivovat. Změnu stavu prvku lze opět provést pouze klientským skriptem. Atribut `READONLY` můžeme použít pouze u elementů `TEXTAREA` a `INPUT` (s typy `TEXT` a `PASSWORD`).

5.5 Pár závěrečných poznámek k formulářům

PHP bylo navrženo speciálně jako jazyk pro tvorbu dynamicky generovaných webových stránek a v mnoha ohledech se tomuto požadavku přizpůsobuje. Říkali jsme si, že data mohou být skriptu odeslána metodami GET a POST. PHP pak automaticky nastaví odpovídající proměnné. Kromě toho máme k dispozici dvě asociativní pole `$HTTP_GET_VARS` a `$HTTP_POST_VARS`, která obsahují všechny proměnné odeslané metodami GET a POST. Tímto způsobem můžeme zpracovávat data i z formulářů, u kterých neznáme jména vstupních polí.

V příští kapitole uvidíme, že data získaná z formulářů se velmi často používají přímo jako vstup do příkazů v jazyce SQL pro databázové servery. PHP automaticky (díky zapnuté direktivě `magic_quotes_gpc`) v proměnných získaných z formulářů nahrazuje některé znaky jako ‘'’ a ‘"’ escape sekvencemi vhodnými pro většinu databázových serverů. To nám často ušetří spoustu práce. Pokud však takovou proměnnou vypíšeme na stránku, může obsahovat nechtěná zpětná lomítka — ty můžeme odstranit voláním `StripSlashes()`.

Standardně PHP převádí obsah proměnných tak, aby jim rozuměl server MySQL. Standard SQL však definuje, že znak ‘'’ se nahrazuje dvojicí ‘''’ a nikoliv ‘\'’. V tomto případě ještě musíme aktivovat direktivu `magic_quotes_sybase`. Zapotřebí je to např. u serverů Sybase a Microsoft.

6. Spolupráce s databázemi

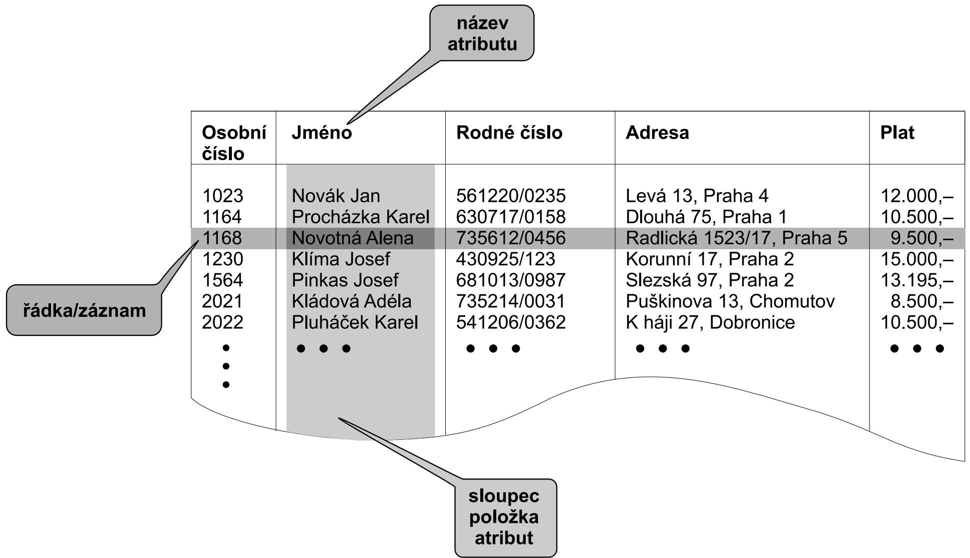
V této kapitole se dostaneme k tématu, které vás asi bude velmi zajímat. Je to celkem pochopitelné. Pryč jsou doby, kdy pro kvalitní prezentaci firmy na Webu stačilo pár statických stránek provázaných odkazy. Díky silné konkurenci je dnes nezbytné podbízet se zákazníkům dalšími užitečnými službami. Na firemním Webu by tedy neměl chybět ceník produktů s možností vyhledávání, stránka sloužící k objednání výrobků apod. Mnohé firmy jdou ještě dál a na webových technologiích mají postaveny celé své informační systémy. Všechny zmíněné aplikace pracují s velkým množstvím dat, které jsou poměrně dobře strukturovány a ukládají se proto do databází. V následující kapitole se proto budeme podrobně zabývat tím, jak dohromady skloubit databáze a Web. Nejprve si stručně vysvětlíme alespoň ty nejdůležitější pojmy, které jsou nutné pro správné pochopení principu práce s databázemi. Následně si ukážeme, jak připravit databázový server pro spolupráci s PHP. Poté bude následovat seznámení s jazykem SQL, který se využívá při práci s databázemi. Ve zbytku kapitoly si ukážeme, jak z prostředí PHP využívat služby mnoha databázových programů.

6.1 Co je to databáze

Téma věnované databázím by samo o sobě vydalo na nejednu tlustou knihu. Tolik prostoru nám však naše kniha nenabízí a seznámíme se tedy pouze s tím nejnutnějším. Zájemce o detailnější a ucelenější informace odkážu na odbornou literaturu.

Databázi si můžeme představit jako místo, kam se ukládají všechny potřebné údaje. Přístup k údajům uloženým v databázi obstarává program, kterému se říká *SŘBD — Systém Řízení Báze Dat*. Tento poněkud krkolomný název vznikl přeložením původního anglického termínu DBMS — DataBase Management System. Mezi SŘBD patří takové programy jako Oracle, MS SQL Server, Sybase, Informix, Progress či InterBase. Nejedná se o programy nikterak levné. Jejich cena se pohybuje v desítkách a většinou spíše i ve stovkách tisíc korun. Naštěstí i na poli SŘBD existují programy šířené zdarma jako freeware — např. mSQL, MySQL a PostgreSQL.

Převážná většina dnes používaných SŘBD při uspořádání údajů v databázi vychází z *relačního modelu dat*. Název tohoto modelu vychází z relační algebry, což je matematický aparát, na kterém relační model dat staví. V tomto modelu jsou údaje uspořádány do *tabulek*. Tabulka zpravidla shromažďuje údaje o jednom druhu objektů. Můžeme tak mít například tabulku s osobními údaji zaměstnanců. Jednotlivé řádky odpovídají jednotlivým zaměstnancům. Sloupce pak obsahují informace o pracovnících — v našem případě by to mohly být následující údaje: osobní číslo, jméno, rodné číslo, adresa a výše platu. Sloupcům



Obr. 6-1: Tabulka v relačním modelu dat

tabulky obvykle říkáme v databázové terminologii *položky* nebo *atributy*. Jednotlivé řádky se pak nazývají *záznamy*. Vše se nám pokusí přiblížit obrázek 6-1.

Aby šlo s tabulkami a v nich uloženými údaji pracovat, musí být nějak jednoznačně identifikovány. Každý sloupec je proto pojmenován — má svůj název. Tento název pak používáme, když se odvoláváme na obsah určitého atributu, a ne na celý záznam. Jménem atributu jsou v naší tabulce např. `Jméno` a `Plat`.

Častou operací prováděnou v tabulkách je změna obsahu jednoho atributu u určitého záznamu. Pro provedení této operace však musíme mít k dispozici způsob, jak jednoznačně určit požadovaný záznam. Pro tyto účely by každá tabulka měla obsahovat tzv. *primární klíč*. Primární klíč je atribut, jehož hodnota je pro každý záznam jedinečná. V našem případě tedy jako primární klíč může posloužit atribut `Osobní číslo`, protože každý zaměstnanec má své vlastní osobní číslo. Našim požadavkům na primární klíč vyhoví i rodné číslo — to má každý občan České republiky jedinečné.

Chybou by však bylo za primární klíč zvolit např. atribut `Jméno`. Na první pohled se jména zaměstnanců liší. Není však problém, aby se ve větší firmě vyskytli dva Janové Nováci. Pak bychom je podle jména rozlišili velice těžko.

Pro každý atribut tabulky musíme určit, jaký typ dat může obsahovat. Mezi nejběžněji používané typy patří celá čísla, znakové řetězce a logické hodnoty (ano/ne). Další velmi často používané typy jsou reálná čísla, měnové údaje, datum a čas. Mnoho SŘBD podporuje i složitější typy, jako je obrázek, video či audio klip.

Databáze může samozřejmě obsahovat větší množství tabulek — záleží na tom, co vše za údaje chceme do databáze zaznamenat. Každá tabulka má proto své jméno, které ji v rámci databáze jednoznačně identifikuje. V našem případě by tabulka měla nejspíše název `Zaměstnanci`. Někdy se používá pouze jednotné číslo, potom by naše tabulka měla jméno `Zaměstnanec`. Jméno tabulky by vždy mělo odpovídat jejímu obsahu, usnadníme si tak pozdější orientaci ve větším množství tabulek.

Vidíme, že tabulka obvykle obsahuje ucelené informace. To však neznamená, že nijak nesouvisí s ostatními tabulkami. Představme si, že v naší fiktivní firmě chceme evidovat informace o odběratelích. U každého odběratele budeme chtít evidovat jeho IČO, název, sídlo a našeho zaměstnance, který má styk s odběratelem na starosti. Je jasné, že tabulka `Odběratelé` musí obsahovat atributy IČO, název a sídlo. Má však u každého odběratele obsahovat kompletní informace o zaměstnanci (tj. osobní číslo, jméno, rodné číslo, adresu a výši platu). To by jistě nebylo nejefektivnější — pokud by někdo měl na starosti více odběratelů, informace by se v tabulce zbytečně opakovaly.

Výše zmíněné problémy se řeší pomocí tzv. *vztahů* mezi tabulkami. V našem případě může mít jeden zaměstnanec na starosti několik odběratelů a hovoříme tedy o vztahu $1 : N$. V praxi se tento vztah řeší tak, že tabulka `Odběratelé` má atribut, který obsahuje primární klíč určující zaměstnance pověřeného stykem s firmou. Primární klíč v tabulce zaměstnanci je osobní číslo, u každého odběratele tedy uvedeme osobní číslo zaměstnance pověřeného jednat za naši firmu (viz obr. 6-2 na následující straně).

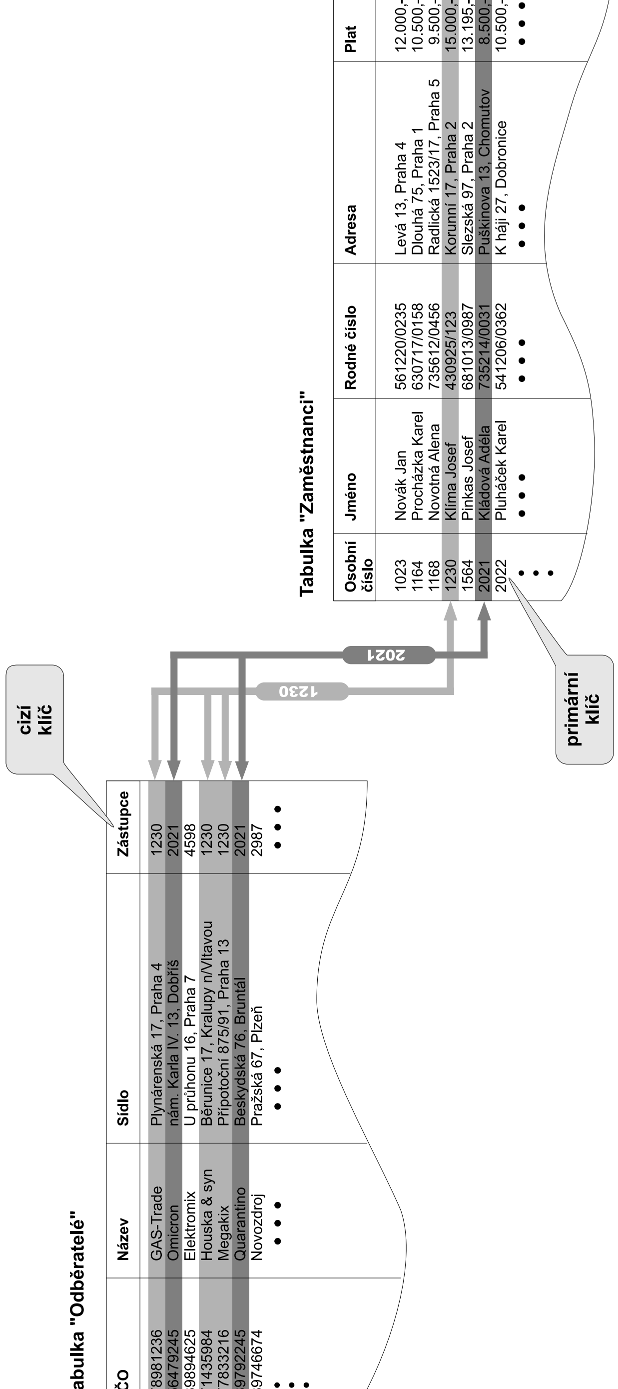
Vidíme, že použité uspořádání údajů do tabulek nám dává k dispozici všechny informace. Pokud chceme zjistit, kdo má na starosti firmu Omicron, podíváme se do tabulky `Odběratelé`. V ní zjistíme, že se jedná o zaměstnance s osobním číslem `2021`. Podíváme se do tabulky zaměstnanci a zjistíme, že osobní číslo `2021` má Adéla Kládová.

Můžeme postupovat i opačně. Pokud nás zajímá, kteří odběratelé spadají do péče Josefa Klímy, zjistíme jeho osobní číslo — `1230`. Příslušní odběratelé v tabulce `Odběratelé` mají toto číslo uloženo v atributu `Zástupce`.

Atributu, který slouží jako odkaz na jinou tabulku a obsahuje tedy primární klíče jiné tabulky, říkáme *cizí klíč*. Pokud se tabulka účastní více vztahů s ostatními tabulkami, může obsahovat více cizích klíčů. Primární klíč je však v každé tabulce vždy jen jeden.

Mezi tabulkami mohou existovat i jiné vztahy než $1 : N$, i když nejsou tak časté. Vztah $1 : 1$ vyjadřuje případy, kdy záznam jedné tabulky odpovídá jednomu záznamu jiné tabulky. Tyto vztahy se obvykle řeší jako speciální případ vztahu $1 : N$ s využitím cizího klíče.

Jisté komplikace přináší vztahy typu $M : N$. Jak takový vztah vypadá? Představme si, že naše fiktivní firma zpracovává najednou několik projektů.



Obr. 6-2: Ukázka vztahu 1:N

Na každém projektu může pracovat několik zaměstnanců, ale zároveň může jeden zaměstnanec pracovat na více projektech. Vztah mezi tabulkami `Projekty` a `Zaměstnanci` je právě vztahem $M : N$.

Na první pohled se vztah $M : N$ nedá do relačního modelu dat napasovat. Naštěstí lze každý vztah $M : N$ rozložit na dva vztahy $1 : N$ s využitím pomocné tabulky. Ukažme si vše na našem příkladě.

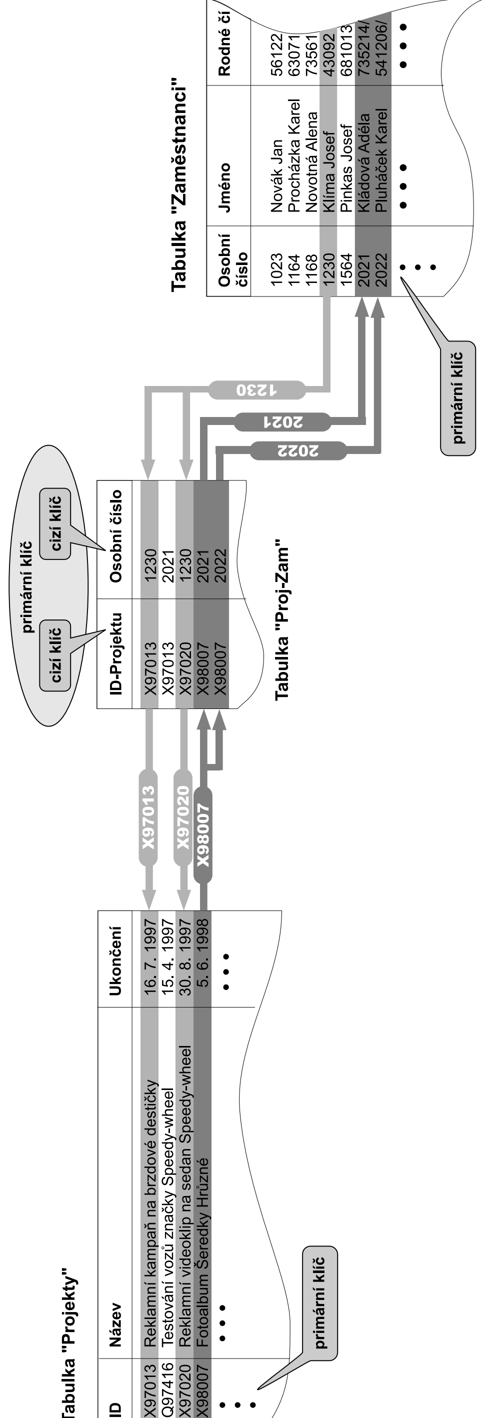
Informace o projektech budeme ukládat do tabulky `Projekty`. Předpokládejme, že každý projekt je identifikován svým ID-číslem, které zvolíme jako primární klíč tabulky. Vztahy mezi projekty a zaměstnanci zachytíme v tabulce `Proj-Zam`. Každý záznam této tabulky obsahuje ID projektu a číslo zaměstnance, který na něm pracuje. Pokud na nějakém projektu pracuje více zaměstnanců, bude v tabulce několik záznamů se stejným ID projektu a rozdílným číslem zaměstnance. Zcela obdobně to bude platit pro zaměstnance. Pokud jeden zaměstnanec pracuje na více projektech, bude v tabulce několik řádků se stejným osobním číslem zaměstnance a různým ID projektu (viz obr. 6-3 na následující straně).

Pomocí tabulky `Proj-Zam` snadno zjistíme, že Josef Klíma pracuje na projektech X97013 a X97020. Stejně snadné je i zjištění, že na projektu X98007 pracují Kládová a Pluháček.

Mezi tabulkami `Projekty` a `Proj-Zam` je vztah $1 : N$. Stejný vztah je i mezi tabulkami `Zaměstnanci` a `Proj-Zam`. Vidíme tedy, že původního vztahu $M : N$ jsme se celkem jednoduše zbavili. Nově vzniklá tabulka `Proj-Zam` obsahuje pouze cizí klíče z tabulek `Projekty` a `Zaměstnanci`. Primární klíč tabulky je tvořen dohromady oběma cizími klíči.

Z této ukázky je vidět, že primární klíč nemusí být tvořen pouze jedním atributem, ale i několika dohromady. Pro každý záznam však stále platí, že jeho primární klíč je jedinečný. Každý záznam tedy musí obsahovat jedinečnou kombinaci hodnot atributů, které jsou součástí primárního klíče. Přitom se vždy snažíme, aby velikost primárního klíče bylo co nejmenší.

Z předchozího textu je doufám patrné, že nejdůležitější je při tvorbě aplikace vymyslet, jak budou data uspořádána do tabulek. Pro jednoduché aplikace většinou vystačíme se selským rozumem a s trochou zkušeností. Pro větší aplikace se však vyplatí použít některou z metodik, které usnadňují vytvoření správného datového modelu pro danou problematiku. Někdy nám práci může usnadnit nakreslení datového modelu např. pomocí ER-diagramu. Na škodu není ani kontrola, že jsou výsledné tabulky v normální formě a neobsahují žádné nežádoucí závislosti. Pokud chcete vyvíjet větší aplikace, nezbyde vám nic jiného než si kromě této knihy nastudovat i další materiály věnované návrhu databázových tabulek a projektování informačních systémů.



Obr. 6-3: Rozložení vztahu M:N

6.2 Jak komunikují SŘBD se zbytkem světa

Přístup k údajům uloženým v databázích obstarává SŘBD. Aby mohly být údaje z databáze přístupné ostatním aplikacím, musí SŘBD nabízet rozhraní, pomocí kterého s ním mohou spolupracovat ostatní programy.

Způsob komunikace se SŘBD je velice obdobný komunikaci s WWW-serverem. Dnes je SŘBD nejčastěji nepřetržitě spuštěn jako démon (na Unixu) nebo jako služba (ve Windows NT) a na určitém socketu[1] očekává požadavky klientů (ostatních aplikací). Na tyto požadavky pak odpovídá. Vidíme tedy, že i zde funguje osvědčený model klient/server. V roli serveru je nyní SŘBD a někdy se mu proto také říká databázový server.

Pro zadávání požadavků na databázový server aplikace nejčastěji používají jazyk *SQL (Structured Query Language)*. Tento jazyk prošel dlouhým vývojem a v různé míře jej dnes podporují téměř všechny běžně používané databázové servery. Někdy se proto databázovým serverům říká zjednodušeně *SQL-servery*. Jazyk SQL nabízí vše potřebné pro vytváření, modifikování a rušení tabulek a pro práci s údaji v tabulce — vyhledávání, přidávání, modifikování a mazání údajů.

V roli klienta pro SQL-server může vystupovat i skript zapsaný v PHP. To znamená, že naše skripty mohou obsahovat příkazy zapsané v jazyce SQL a zpracovávat jejich výsledky po provedení na SQL-serveru. Nic tedy nebrání tomu, aby byl přes Web zpřístupněn obsah nějaké databáze.

Bohužel, v praxi je vše samozřejmě o něco složitější. Každý SQL-server má svůj vlastní protokol, kterým s ním může klient komunikovat. Pokud má klient umět komunikovat s více různými servery, musí podporovat více protokolů. To není zrovna nejšťastnější řešení, a proto na platformě Windows vzniklo rozhraní ODBC. To slouží jako prostředník mezi klientskou aplikací a databázovým serverem. Rozhraní ODBC se volá jednotně a ODBC-ovladač pak požadavek předá databázovému serveru pomocí správného protokolu.

Velkou výhodou ODBC tedy je, že stejným způsobem můžeme přistupovat k libovolné databázi. Pokud se tedy z nějakého důvodu změní SQL-server, na kterém běží naše aplikace, nemusíme měnit v PHP skriptech žádný kód.

Počáteční nevýhodou, která mluvila proti použití ODBC, byl nižší výkon oproti nativním ovladačům. Staré ODBC ovladače sloužily pouze jako mezistupeň mezi aplikací a nativním protokolem databáze. Novější ODBC ovladače jsou však optimalizovány a k databázovému serveru přistupují přímo — jejich výkon je srovnatelný s použitím nativních ovladačů. Některé novější SŘBD obsahují jako svůj jediný nativní protokol právě ODBC.

[1] Socket si můžeme představit jako vstupní bod k určité službě přístupné přes síť. Socket je identifikován adresou počítače, číslem portu a protokolem (TCP nebo UDP). Například WWW-servery nejčastěji naslouchají na socketu, jehož port je 80 a protokol TCP.

Poslední otázka, která se nabízí, se týká využitelnosti ODBC na jiných platformách, než jsou Windows. ODBC bylo původně vyvinuto pro platformu Windows, ale dnes jeho implementace existuje i pro všechny významnější verze operačního systému Unix.

PHP samozřejmě obsahuje podporu pro práci s databázemi pomocí rozhraní ODBC. Kromě toho máme k dispozici funkce, které umožňují pomocí nativního protokolu pracovat s následujícími SŘBD:

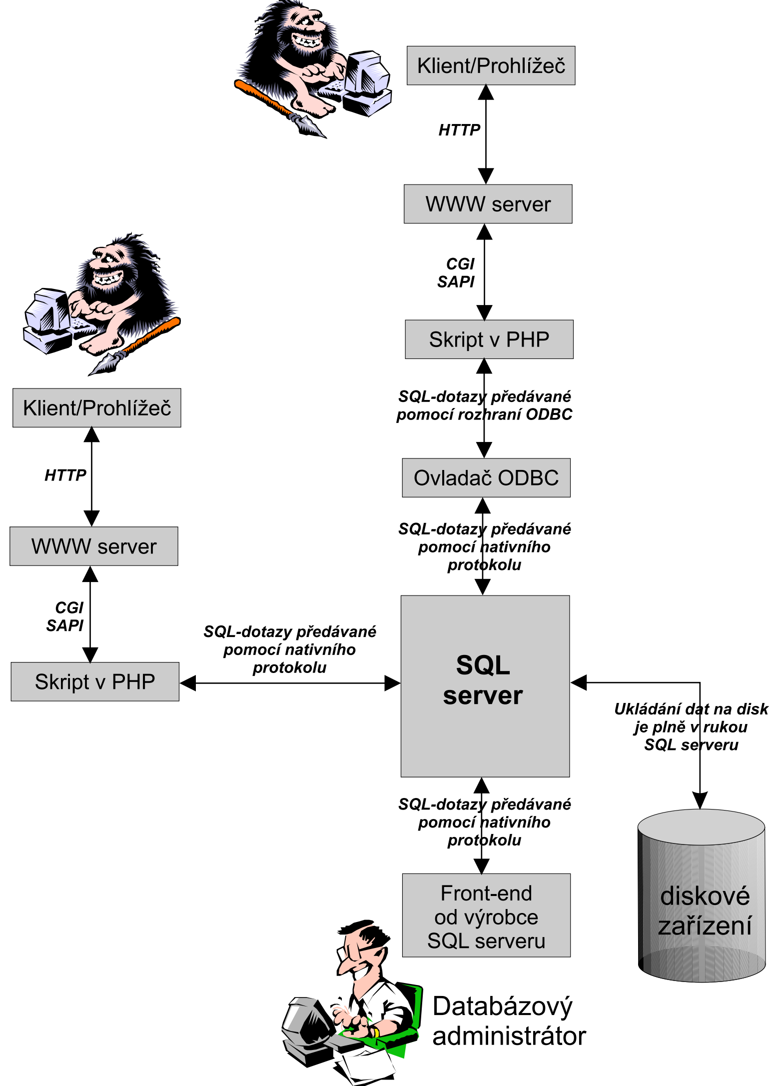
Adabas D	MySQL	Sybase
Informix	Oracle	Velocis
MS SQL Server	PostgreSQL	
mSQL	Solid	

Kromě podpory moderních SQL-serverů, obsahuje PHP i funkce pro práci se staršími databázovými technologiemi. V minulosti se často pro práci s daty používal přímý přístup k souboru, který obsahoval data. Asi nejznámějšími zástupci této skupiny jsou systémy dBase (v DOSu a ve Windows) a dbm (v Unixu). PHP obsahuje funkce pro práci se soubory dBase i dbm. Navíc obsahuje podporu pro u nás málo známý systém FilePro. Těmito systémy se v naší publikaci zabývat nebudeme, protože dnes se databáze vyvíjejí zcela jiným směrem. Podpora pro tyto starší systémy se však může hodit, pokud aplikaci v PHP potřebujeme napojit do stávajícího informačního systému, který využívá dBase nebo dbm. Referenční přehled funkcí proto obsahuje popis funkcí pro práci se soubory dBase a dbm.

Častým dotazem mnoha uživatelů je, zda mohou přistupovat k datům z programů MS Access a MS Excel. Tento přístup je možný — nejjednodušší je asi přes rozhraní ODBC, protože ovladače pro Access a Excel bývají standradní součástí distribuce obou programů.

Obrázek 6-4 na následující straně nám přibližuje tři způsoby komunikace s SQL-serverem, které budeme v naší knize používat. Naše aplikace napsané v PHP budou se serverem komunikovat buď přes rozhraní ODBC, nebo přímo pomocí nativního protokolu. Je zajímavé si uvědomit, jak zdlouhavá je v tomto případě cesta mezi uživatelem naší aplikace a samotným SQL-serverem. Uživatel ve svém prohlížeči provede akci, jejímž výsledkem je požadavek na určité URL. Prohlížeč se tedy spojí s daným WWW-serverem a předá mu požadavek. WWW-server spustí interpret PHP na požadovaný skript a předá mu parametry zaslané uživatelem. Skript se pak přímo nebo přes rozhraní ODBC připojí k SQL-serveru (ten může běžet dokonce na jiném počítači než WWW-server). Odpověď klientovi směřuje stejnou cestou zpět.

Na obrázku je ještě zachycena front-end aplikace, která slouží ke správě SQL-serveru. Každý server potřebuje údržbu — musíme v něm vytvářet databáze, spravovat uživatelské účty, provádět pravidelné zálohy dat, monitorovat zatížení serveru apod. K těmto účelům se ke každému serveru dodává speciální aplikace.



Obr. 6-4: S jedním SQL-serverem může najednou spolupracovat mnoho aplikací

Většina freewarových serverů obsahuje pouze jednoduchý řádkový monitor, pomocí kterého můžeme zadávat příkazy ovlivňující chod serveru. Komerčně šířené servery většinou obsahují speciálně vytvořený grafický nástroj pro správu serveru.

Většina front-end aplikací nám rovněž umožňuje zadávat serveru přímo příkazy v jazyce SQL. Můžeme je s výhodou použít na vyzkoušení toho, zda jsou

naše SQL-příkazy správné. Proto si před samotným výkladem jazyka SQL ukážeme, jak v různých SŘBD zadávat přímo SQL dotazy a jak je připravit na další práci s PHP.

6.3 Výběr a instalace SQL-serveru

Výběr databáze pro každou větší aplikaci je jistě klíčový. Musíme zvážit mnoho faktorů, jako je cena, výkon a kompatibilita s ostatními programy. V našich podmínkách se vyplatí zvažovat i další kritéria — například, zda server podporuje správné řazení záznamů podle české abecedy.

Pokud je limitujícím faktorem cena, je vhodným kandidátem PostgreSQL. Jedná se o databázový server původně vyvíjený na Kalifornské univerzitě v Berkeley, který obsahuje některé pokročilé technologie, jako objekty, dědičnost, možnost definice vlastních datových typů a funkcí. Jeho nevýhodou je poměrně malý výkon, který může vadit zejména u hodně exponovaných aplikací nebo aplikací, které pracují nad rozsáhlými datovými tabulkami. PostgreSQL je však zcela zdarma pro všechny verze Unixu. To je jeden z důvodů, proč je na mnoha serverech používán i přes svůj menší výkon.

Pokud je vaším primárním zájmem rychlost, může být zajímavé MySQL. MySQL je velice rychlý databázový server, který běží na Unixu i na Windows. Jeho rychlost vychází zejména z toho, že neobsahuje podporu pro transakční zpracování dat — bez něj se však často obejdeme. Novější verze MySQL obsahují velice dobrou podporu češtiny. Pro nekomerční použití je MySQL zcela zdarma, komerční uživatelé musí za použití zaplatit několik tisíc — to je však směšná cena v porovnání s ostatními komerčními servery.

Pokud používáte Linux, asi víte, že během roku 1998 byly uvolněny k volnému použití na této platformě komerční SŘBD InterBase a Sybase. Otázkou je, jak dlouho budou tyto servery pro Linux zdarma. Jedná se však o špičkové produkty.

V Čechách jsou produkty firmy Microsoft rozšířeny poněkud více než ve zbytku světa. To platí i o databázích. Na serverech pracujících pod Windows NT často jako databáze běží MS SQL Server. Server to není nikterak levný. Navíc, pokud chceme v něm uložená data zpřístupnit na Webu, měli bychom si zakoupit zvláštní licenci, která stojí zhruba 100 000 Kč.

Komerční servery můžeme získat zcela běžnou cestou jako ostatní komerční software. Freewarové servery a omezené zkušební verze serverů jsou k dispozici ke stažení z Internetu nebo je můžeme získat na různých discích CD-ROM.

Tak například u nás tolik oblíbená distribuce Linuxu RedHat obsahuje server PostgreSQL. K dispozici jsou i disky s pohodlnou instalací Apache, PHP a demoverze databázového serveru Solid.

My si ukážeme, co je potřeba udělat pro přístup k databázím MySQL, PostgreSQL a MS SQL Server ze skriptů PHP.

MySQL

MySQL si můžeme stáhnout na adrese `http://www.tcx.se` nebo na českém zrcadle `http://mirror.opf.slu.cz/mysql/`. Po úspěšné instalaci MySQL by měl být spuštěn démon `mysqld` (na Windows NT služba), který zajišťuje činnost serveru. Standardně máme k dispozici databázi `test`, která se hodí pro testovací účely — právo přístupu k ní má každý uživatel.

Pro první seznámení s MySQL můžeme použít jednoduchého klienta `mysql`, který je součástí distribuce. Tento klient umožňuje zadávání příkazů SQL a několika dalších příkazů souvisejících se správou databází.

```
MySQL
exit    (\a)    Exit mysql. Same as quit
go      (\g)    Send command to mysql server
print   (\p)    print current command
quit    (\q)    Quit mysql
rehash  (\#)    Rebuild completion hash
status  (\s)    Get status information from the server
use     (\u)    Use another database. Takes database name as argument

Connection id: 3  (Can be used with mysqladmin kill)

mysql> SELECT Stranka, Pridano, Pocet FROM Pristupy WHERE Pocet > 10;
+------------------------------------------+------------+-------+
| Stranka                                  | Pridano    | Pocet |
+------------------------------------------+------------+-------+
| /php-book/xxx.php                        | 1998-09-29 |    77 |
| /~jkj/indexa.html                        | 1998-09-29 |    11 |
| /~jkj/www/index.html                     | 1998-09-29 |    11 |
| /~jkj/articles/dhtml/uvod.html           | 1998-09-29 |    11 |
| /~jkj/articles/dhtml/java.html           | 1998-09-29 |    15 |
| /~jkj/articles/dhtml/css-properties.html | 1998-09-29 |    12 |
| /~jkj/html/ukazky.html                   | 1998-09-29 |    31 |
+------------------------------------------+------------+-------+
7 rows in set (0.01 sec)

mysql>
```

Obr. 6-5: Prostředí řádkového klienta MySQL

Pro práci s klientem jsou důležité dvě věci. První z nich je, jak klienta ukončit — `mysql` ukončíme příkazem `quit`. Další věcí, která často dělá problémy, je nutnost ukončit každý SQL-příkaz středníkem '`;`'. Dokud středník nezadáme, klient neodešle příkaz ke zpracování a my neuvidíme žádné výsledky.

Pro první pokusy si vystačíme s databází `test`, kam mají přístup všichni. Jméno databáze se zadává jako parametr řádkového klienta `mysql`. Pokud chceme pracovat s databází `test`, můžeme použít příkaz

```
mysql test
```

Pokud chceme vytvořit novou databázi, poslouží nám k tomu příkaz

`mysqladmin create` *«jméno databáze»*

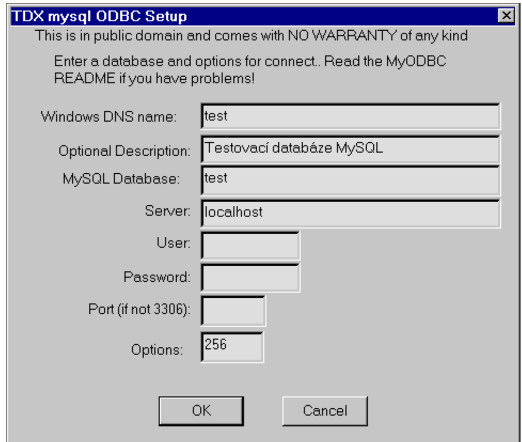
Novou databázi bychom měli vytvářet pro každou novou aplikaci, aby byla všechna data přehledně uspořádána. U nově vytvořené databáze musíme správně nastavit přístupová práva, jinak se nám stane, že s databází nebudeme moci pracovat. Popis nastavení přístupových práv je uveden v dokumentaci a my se o něm za chvíli ještě zmíníme.

Užitečným příkazem je `mysqlshow`, který nám zobrazí všechny dostupné databáze. Pokud jako parametr příkaz použijeme jméno nějaké databáze, vypíší se všechny tabulky obsažené v databázi.

Další podrobnosti o MySQL naleznete v originální dokumentaci. My si ještě ukážeme, jak správně nastavit ODBC ovladače ve Windows tak, abychom mohli pro přístup k MySQL používat ODBC místo nativních funkcí. Pokud chcete k MySQL přistupovat pomocí nativních funkcí, nemusíte se vůbec instalací ODBC ovladačů zabývat.

Ovladače ODBC pro Windows získáme na stejné adrese jako samotné MySQL. Samotnou instalaci spustíme příkazem `setup`. Po instalaci již zbývá jen správně nakonfigurovat datové zdroje.

Každý datový zdroj ODBC odpovídá jedné databázi a pomocí něj mohou k databázi přistupovat všechny aplikace, které podporují ODBC. Pro vytvoření zdroje spustíme v Ovládacím panelu ikonu ODBC a vybereme záložku System DSN. Nyní pomocí tlačítka Add… přidáme nový datový zdroj. V následujícím dialogovém okně jako server samozřejmě vybereme MySQL. Objeví se okno (viz obr. 6-6) pro nastavení parametrů zdroje dat.



Obr. 6-6: Nastavení parametrů datového zdroje

Do pole Windows DSN Name[2] vyplníme jméno datového zdroje. Doporučuji pro datový zdroj použít jméno stejné, jako má databáze — alespoň se v tom později vyznáme. Mezi další důležité parametry patří jméno databáze (`test`) a adresa serveru. Ta bude nejčastěji `localhost`, protože SQL-server běží na stejném počítači jako WWW-server. Parametry doporučuji nastavit na hodnotu 256, která zamezí případným problémům s některými staršími verzemi ODBC.

Do polí User a Password můžeme vyplnit jméno a heslo, pod kterým se hlásíme k databázi. Tato pole ve většině případů necháme prázdná, a uživatele a heslo nastavíme až přímo při vytváření spojení v PHP skriptu. Pokud bychom jméno a heslo uvedli v konfiguraci ODBC zdroje, získal by přístup k databázi každý, kdo má přístup k počítači.

PostgreSQL

Domovská stránka serveru PostgreSQL je `http://www.postgresql.org`. Po úspěšné instalaci na počítači běží démon `postmaster`. Ten pracuje jako SQL-server a umí odpovídat na požadavky klientů. Společně s PostgreSQL je dodáván i jednoduchý řádkový klient `psql`.

Nejčastěji použijeme `psql` pouze s parametrem, který určuje jméno databáze. V tomto případě se spustí řádkový klient, který nám umožní zadávání SQL-příkazů nad danou databází, podobně jako program `mysql` pro server MySQL.

Důležité je opět vědět, jak se `psql` ukončuje. Slouží k tomu příkaz (poněkud atypický) `\q`. SQL-příkazy musíme ukončovat středníkem, jinak se neodešlou na server ke zpracování.

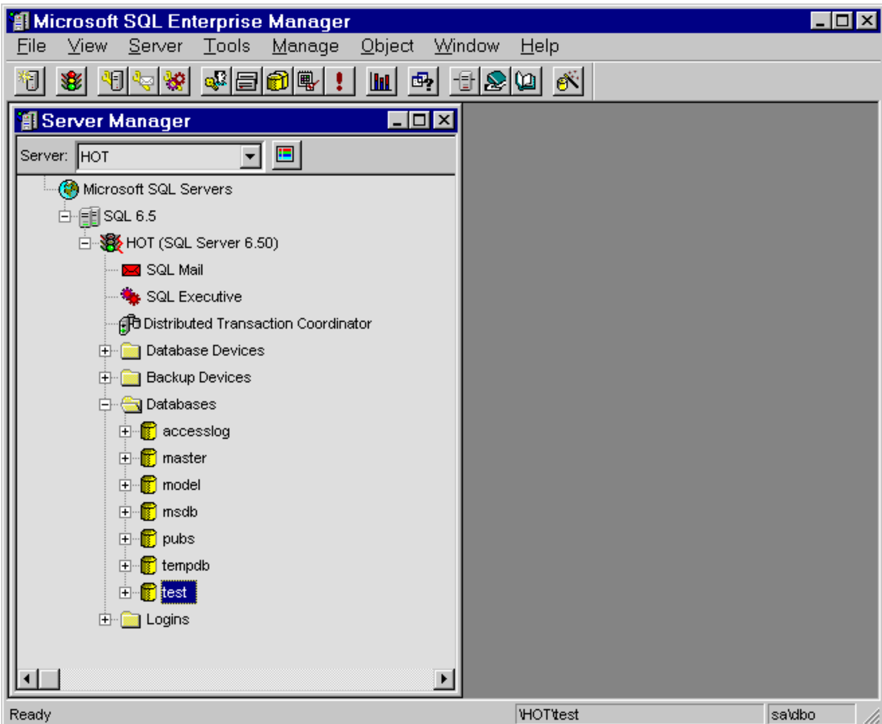
Užitečným příkazem je `psql -l`, který nám zobrazí všechny dostupné databáze.

Pokud použijeme příkaz `psql` *«databáze»* `-c \\d`, vypíší se všechny tabulky obsažené v databázi.

Abychom mohli s PostgreSQL pracovat, musí být naše uživatelské jméno zaregistrováno pro přístup k databázi. Toho dosáhneme pomocí příkazu `createuser`. Příkaz `createuser` je často dostupný pouze správci systému PostgreSQL a není dostupný běžnému uživateli. V tomto případě o přidělení přístupu musíme požádat správce. Databázi si vytvoříme pomocí příkazu `createdb` *«jméno databáze»*.

---

[2] Na obrázku sice vidíte pole Windows DNS Name, ale autoři se nejspíše jen přeťukli na klávesnici a místo DSN použili zkratku DNS.



Obr. 6-7: Správa serveru myší — to je SQL Enterprise Manager

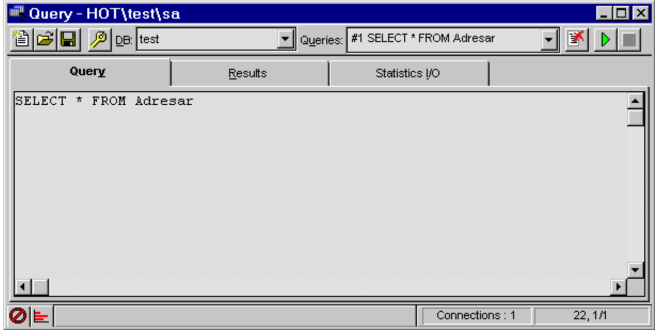
MS SQL Server

MS SQL Server se instaluje, jako snad všechny programy Microsoftu, pomocí instalačního programu `setup`. Pro správu databáze je asi nejvýkonnější nástroj SQL Enterprise Manager. Pomocí tohoto nástroje můžeme spravovat celý SQL-server.

Pokud chceme pracovat s nějakou databází, musíme ji samozřejmě nejprve vytvořit. Pro každou databázi musíme vytvořit databázové zařízení (database device) pomocí příkazu New Device... vybraném z pop-up menu na složce Database Devices.

Zcela obdobně vytvoříme novou databázi pomocí příkazu New Database..., který můžeme vybrat z pop-up menu na složce Databases.

Pro každou databázi můžeme vytvořit uživatele, kteří k ní mají přístup. Každý takový uživatel však musí být spojen ještě s účtem pro přihlášení k SQL-serveru (složka Logins). Nejlepší je proto vytvořit pro přístup k databázi uživatele, jehož jméno je totožné s nějakým jménem pro přihlášení. Pro testovací účely můžeme použít uživatele `guest`, který již má vytvořen i účet pro přihlášení (účet není chráněn heslem).



Obr. 6-8: SQL Query Tool

Pokud si v MS SQL Serveru chceme zkoušet SQL-dotazy nanečisto, můžeme k tomu použít příkaz Tools ▷ SQL Query Tool...

SQL-server budeme z PHP nejspíše obsluhovat pomocí rozhraní ODBC. Musíme si proto vytvořit pro každou databázi nový datový zdroj. My si ukážeme vytvoření datového zdroje pro databázi `test`.

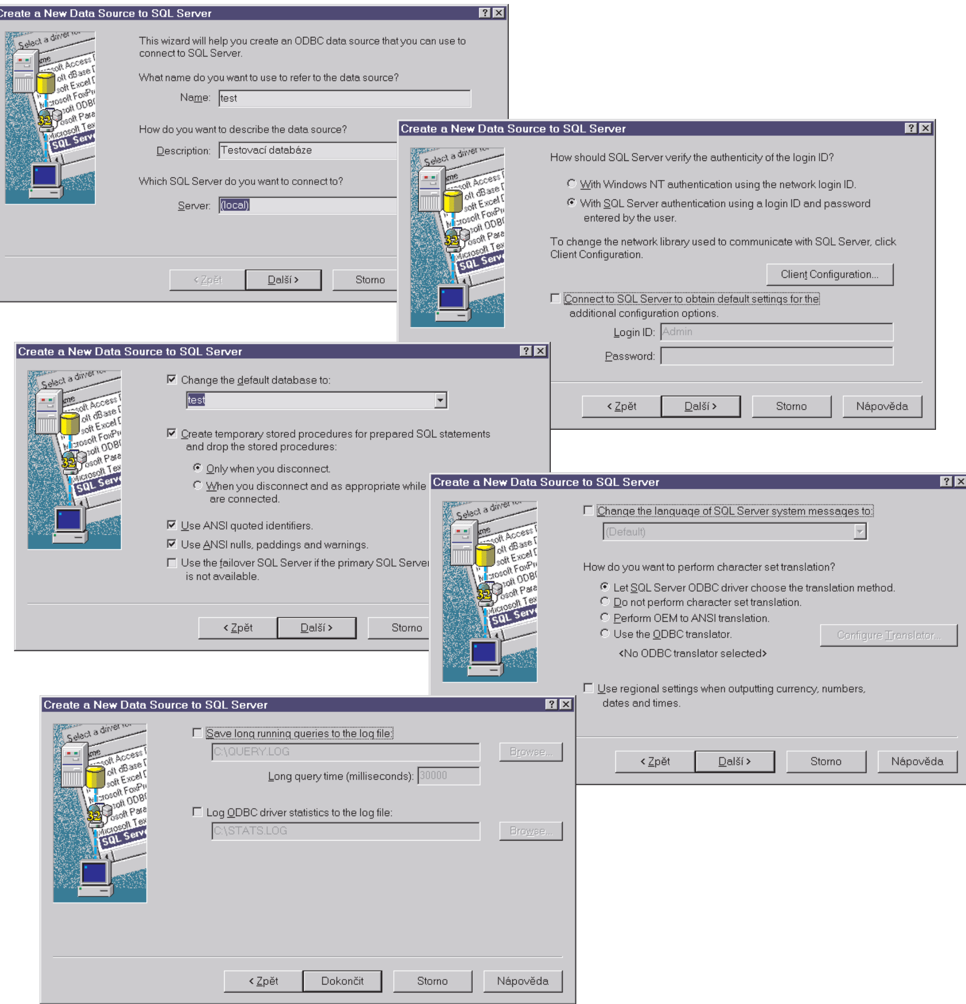
Pro vytvoření zdroje spustíme v Ovládacím panelu ikonu ODBC a vybereme záložku System DSN. Nyní pomocí tlačítka Add... přidáme nový datový zdroj. V následujícím dialogovém okně jako server samozřejmě vybereme SQL Server. Postupně se objeví několik oken (viz obr. 6-9 na následující straně) pro nastavení parametrů zdroje dat.

Většinou jsou standardní hodnoty vyhovující, i přesto však musíme provést několik malých změn. V druhém dialogovém okně musíme změnit způsob autentifikace uživatelů z Windows NT na SQL Server a vypnout zjišťování dodatečných informací. Na další kartě bychom měli vybrat databázi, ke které bude datový zdroj připojen — v našem případě jsme vybrali databázi `test`. Všechny ostatní volby už můžeme nechat tak, jak jsou.

Na konci nám je nabídnuta možnost otestování datového zdroje. Klidně si ji vyberte, ale buďte si jisti, že test proběhne neúspěšně. Je to způsobeno tím, že jsme použili autentifikaci SQL-serveru místo Windows NT. Testovací procedura se nás však neptá na jméno a heslo, které se má použít při připojování k databázi. To však není problém, protože je můžeme specifikovat v našich PHP skriptech.

6.4 Lehký úvod do jazyka SQL

Už víme, jak v prostředí několika databázových serverů zadávat SQL-příkazy. V ostatních to bude velice podobné a vy je máte přece pořízeny legálně i s dokumentací. Můžeme se tedy vrhnout přímo do hlubin jazyka SQL. Nemusíme se však bát dekompresní nemoci, protože se neponoříme příliš hluboko. Jazyk SQL prošel dlouhým vývojem, existuje několik jeho verzí, každý server má svá



Obr. 6-9: Konfigurace datového zdroje pro MS SQL Server

vlastní rozšíření nad rámec standardu. My se proto seznámíme pouze se základními příkazy SQL, které dnes podporují snad všechny SQL-servery. Kromě popisu příkazů si ukážeme jejich praktické použití na tabulkách popsaných na začátku kapitoly. Budeme přitom předpokládat, že máme vytvořenou novou databázi `test`, ke které jsme připojeni. Podrobnější informace o SQL naleznete v dokumentaci k vašemu serveru nebo ve výborné Šnekově publikaci [18].

Vytvoření tabulky

Pro vytvoření tabulky v databázi slouží SQL-příkaz `CREATE TABLE`. Jeho syntaxe je následující:

```
CREATE TABLE «jméno tabulky» (
    «jméno 1. položky» «typ»,
    «jméno 2. položky» «typ»,
    ...
    «jméno n. položky» «typ»)
```

Typy použitelné v SQL se příliš neliší od typů, které známe z jiných jazyků. K dispozici máme typy pro celá i reálná čísla, pro textové řetězce, pro datum a čas, pro binární data, jako jsou obrázky apod. Přehled nejpoužívanějších typů si můžeme prohlédnout v tabulce 6-1.

Typ	Popis
`int`	celé číslo v rozsahu od −2 147 483 648 do 2 147 483 647
`smallint`	celé číslo v rozsahu od −32 768 do 32 767
`tinyint`	celé číslo v rozsahu od 0 do 255
`float`	číslo s pohyblivou řádovou čárkou
`char(`n`)`	textový řetězec o délce n (maximálně však 255 znaků)
`varchar(`n`)`	textový řetězec o maximální délce n (maximálně však 255 znaků)
`decimal(`p`)`	desetinné číslo s p platnými číslicemi
`decimal(`p`,`d`)`	desetinné číslo s p platnými číslicemi a s d desetinnými místy
`money`	peněžní částka (tento typ nepodporují zdaleka všechny servery, můžeme ho však snadno nahradit například pomocí `decimal(12,2)`)
`datetime`	údaj o čase a datu ve formátu `RRRR-MM-DD HH:MM:SS`
`time`	údaj o čase ve formátu `HH:MM:SS`
`date`	údaj o datu ve formátu `RRRR-MM-DD`
`blob, image`	speciální typy pro uložení dlouhých binárních dat (každý server používá vlastní typ)

Tab. 6-1: Přehled datových typů SQL

Pro vytvoření naší tabulky `Zamestnanci` proto můžeme použít následující příkaz SQL:

```
CREATE TABLE Zamestnanci (
    OsobniCislo int,
    Jmeno       varchar(40),
    RC          char(11),
    Adresa      varchar(60),
    Plat        decimal(10,2))
```

Při vytváření tabulek však můžeme u každé položky nastavit i několik dalších vlastností. Jednou z nejdůležitějších je vlastnost `NOT NULL`. Takto definovaná položka nemůže obsahovat prázdnou hodnotu. Primární klíč tabulky musíme definovat s tímto modifikátorem, abychom zajistili, že bude vždy obsahovat nějakou hodnotu. Zároveň pomocí direktivy `PRIMARY KEY` musíme říci, která položka bude primárním klíčem. Nakonec tedy dostaneme:

```
CREATE TABLE Zamestnanci (
    OsobniCislo int NOT NULL PRIMARY KEY,
    Jmeno       varchar(40),
    RC          char(11),
    Adresa      varchar(60),
    Plat        decimal(10,2))
```

Pokud je primární klíč složen z více atributů (položek), musíme použít odlišný způsob pro zadání primárního klíče:

```
CREATE TABLE Proj_Zam (
    ID_Projektu char(6) NOT NULL,
    OsobniCislo int NOT NULL,
    PRIMARY KEY (ID_Projektu, OsobniCislo))
```

V názvech tabulek a atributů je lepší se obejít bez českých diakritických znamének, protože mohou některým serverům vadit. Některým serverům vadí v názvech i jiné znaky — např. pomlčky. Pokud však budeme používat pouze písmena a podtržítka, budeme si bezpečně rozumět se všemi servery.

Zkuste si nyní sami vytvořit tabulky `Projekty` a `Odberatele`, ať se pocvičíte v ovládání SQL-klienta vašeho serveru.

Přidání nového záznamu do tabulky

Máme vytvořeny tabulky, ale bohužel jsou zatím prázdné. Pro přidávání nových záznamů slouží v SQL příkaz `INSERT INTO`. Jeho základní syntaxe je:

```
INSERT INTO «jméno tabulky» VALUES (
    «hodnota 1. položky»,
    «hodnota 2. položky»,
    ...
    «hodnota n. položky»)
```

Pro vložení údajů do tabulky `Zamestnanci` můžeme použít příkaz typu:

```
INSERT INTO Zamestnanci VALUES (
    1023, 'Novák Jan', '561220/0235', 'Levá 13, Praha 4', 12000)
```

Všimněme si, že hodnoty položek textového typu musíme uzavírat do apostrofů. Totéž platí i pro položky obsahující datum a čas. Jediné položky, které můžeme zadávat přímo bez apostrofů, jsou číselné.

Většina klientů nám po zadání výše uvedeného příkazu oznámí něco jako:

```
Query OK, 1 row(s) affected
```

Tím se nám snaží říci, že příkazem byla ovlivněna jedna řádka (tj. byl přidán jeden záznam do tabulky).

```
MySQL
mysql> CREATE TABLE Zamestnanci (
    ->     OsobniCislo int NOT NULL PRIMARY KEY,
    ->     Jmeno       varchar(40),
    ->     RC          char(11),
    ->     Adresa      varchar(60),
    ->     Plat        decimal(10,2));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO Zamestnanci VALUES (
    ->     1023, 'Novák Jan', '561220/0235', 'Levá 13, Praha 4', 12000);
Query OK, 1 row affected (0.20 sec)

mysql> SELECT * FROM Zamestnanci;
+-------------+-----------+-------------+------------------+----------+
| OsobniCislo | Jmeno     | RC          | Adresa           | Plat     |
+-------------+-----------+-------------+------------------+----------+
|        1023 | Novák Jan | 561220/0235 | Levá 13, Praha 4 | 12000.00 |
+-------------+-----------+-------------+------------------+----------+
1 row in set (0.04 sec)

mysql> _
```

Obr. 6-10: Vytvoření tabulky a přidání záznamu v prostředí klienta MySQL

Sami vidíme, že přidávání záznamů do tabulky tímto způsobem není nikterak uživatelsky pohodlné. V další sekci si proto ukážeme, jak snadno lze kombinací PHP a SQL vytvořit formulář sloužící pro pohodlné přidávání záznamů do tabulky.

Výběr a prohlížení záznamů

Asi nejpoužívanějším SQL-příkazem je SELECT. Ten umožňuje výběr a prohlížení záznamů uložených v tabulkách. Pro vypsání všech záznamů dané tabulky můžeme použít příkaz

```
SELECT * FROM «tabulka»
```

Po zadání příkazu

```
SELECT * FROM Zamestnanci
```

tedy dostaneme

```
OsobniCislo Jmeno           RC          Adresa                    Plat
----------- --------------- ----------- ------------------------- --------
1023        Novák Jan       561220/0235 Levá 13, Praha 4          12000.00
1164        Procházka Karel 630717/0158 Dlouhá 75, Praha 1        10500.00
1168        Novotná Alena   735612/0456 Radlická 1523/17, Praha 5  9500.00
1230        Klíma Josef     430925/123  Korunní 17, Praha 2       15000.00
1564        Pinkas Josef    681013/0987 Slezská 97, Praha 2       13195.00
2021        Kládová Adéla   735214/0031 Puškinova 13, Chomutov     8500.00
2022        Pluháček Karel  541206/0362 K háji 27, Dobronice      10500.00
```

Příkaz SELECT toho nabízí samozřejmě mnohem více. Jeho syntaxe může být poměrně bohatá, jak se ostatně můžeme přesvědčit z následujícího schématu

```
SELECT «seznam výstupních položek»
    FROM «seznam tabulek»
    WHERE «podmínka»
    GROUP BY «seznam položek»
    HAVING «skupinová podmínka»
    ORDER BY «kritéria třídění»
```

Pokud vám teď případ SELECT připadá složitý, buďte si jisti, že jsme jej hodně zjednodušili. Pojďme se však podívat na jeho jednotlivé části.

Přímo za příkazem SELECT můžeme uvést *«seznam výstupních položek»*. Pokud tedy chceme z tabulky vytáhnout pouze některé údaje, nemusíme používat hvězdičku pro vypsání všech, ale požadované údaje určit:

```
SELECT Jmeno, Plat FROM Zamestnanci

Jmeno                                    Plat
---------------------------------------- -------------
Novák Jan                                12000.00
Procházka Karel                          10500.00
```

```
Novotná Alena                          9500.00
Klíma Josef                           15000.00
Pinkas Josef                          13195.00
Kládová Adéla                          8500.00
Pluháček Karel                        10500.00
```

Kromě názvů položek můžeme používat i různé funkce a matematické operátory. Pokud chceme zjistit počet zaměstnanců, použijeme příkaz

```
SELECT Count(*) FROM Zamestnanci
```

Místo funkce `Count()` můžeme použít `Avg()` a rázem zjistíme průměrný plat ve firmě:

```
SELECT Avg(Plat) FROM Zamestnanci
```

Pokud nás nezajímají všechny záznamy v tabulce, můžeme je omezit pomocí podmínky zadané za klauzulí `WHERE`. Následující příkaz například zjistí jména a osobní čísla těch zaměstnanců, jejichž plat je vyšší než 12 000 Kč:

```
SELECT Jmeno, OsobniCislo FROM Zamestnanci WHERE Plat > 12000
```

Podobně můžeme zjistit názvy odběratelů, o které se stará zaměstnanec s číslem 1230:

```
SELECT Nazev FROM Odberatele WHERE Zastupce = 1230
```

Při porovnávání položek, které obsahují textový řetězec, nemůžeme používat '='. Místo toho máme k dispozici operátor `LIKE`. Pokud chceme zjistit vše o Janu Novákovi, použijeme příkaz

```
SELECT * FROM Zamestnanci WHERE Jmeno LIKE 'Novák Jan'
```

V řetězci za operátorem `LIKE` můžeme použít dva znaky se speciálním významem. Znak procento '%' nahrazuje libovolnou skupinu písmen. Znak podtržítko '_' nahrazuje jeden libovolný znak. Pokud chceme vybrat všechny zaměstnance, jejichž jméno začíná na `Nov`, můžeme použít příkaz

```
SELECT * FROM Zamestnanci WHERE Jmeno LIKE 'Nov%'
```

Jednotlivé podmínky můžeme navzájem kombinovat pomocí logických spojek `AND`, `OR` a `NOT`. Můžeme tak vybrat všechny zaměstnance, kteří mají plat menší než 7 000,– a nejmenují se Novák:

```
SELECT * FROM Zamestnanci
WHERE (Plat < 7000) AND NOT (Jmeno LIKE 'Novák %')
```

Podmínky za `WHERE` mají ještě jednu důležitou úlohu. Umožňují propojení více tabulek v jednom dotazu. Na začátku kapitoly jsme si vysvětlili, co jsou to vztahy mezi tabulkami a jak nám umožňují zachytit různé situace. Tabulky,

které spolu byly ve vztahu, byly propojeny pomocí primárního a cizího klíče. Primární a cizí klíč přitom není nic jiného než položky, které mají pro související záznamy stejnou hodnotu. Pokud tedy chceme zjistit, kdo má kterého odběratele na starosti, použijeme příkaz:

```
SELECT Nazev, Jmeno FROM Odberatele, Zamestnanci
WHERE Zamestnanci.OsobniCislo = Odberatele.Zastupce
```

Pro ilustraci si ukážeme výsledek dotazu

```
Nazev              Jmeno
------------------ -----------------
Omicron            Kládová Adéla
Houska & syn       Klíma Josef
Megakix            Klíma Josef
GAS-Trade          Klíma Josef
Quarantino         Kládová Adéla
```

Pokud chceme zjistit, kdo na jakém projektu pracuje, bude podmínka již složitější, protože musíme spojit tři tabulky dohromady:

```
SELECT Jmeno, Nazev
FROM Zamestnanci, Proj_Zam, Projekty
WHERE (Zamestnanci.OsobniCislo = Proj_Zam.OsobniCislo)
      AND (Proj_Zam.ID_Projektu = Projekty.ID)
```

Pokud by nás zajímalo, na kolika projektech najednou pracovníci pracují, museli bychom použít klauzuli `GROUP BY`. Výsledky posledního dotazu sdružíme do skupin určených osobním číslem a pro každou skupinu určíme počet projektů.

```
SELECT Zamestnanci.OsobniCislo, Jmeno, Count(ID)
FROM Zamestnanci, Proj_Zam, Projekty
WHERE (Zamestnanci.OsobniCislo = Proj_Zam.OsobniCislo)
      AND (Proj_Zam.ID_Projektu = Projekty.ID)
GROUP BY Zamestnanci.OsobniCislo
```

Výsledek dopadne tak, jak jsme jej očekávali

```
OsobniCislo Jmeno          Count(ID)
----------- -------------- ---------
1230        Klíma Josef    2
2021        Kládová Adéla  2
2022        Pluháček Karel 1
```

Všimněme si ještě, že v dotazech s více tabulkami musíme pro položky, které se pod stejným jménem vyskytují ve více tabulkách, explicitně určit jméno tabulky pomocí notace *«tabulka»*`.`*«položka»*.

Pokud se nám u výsledné tabulky nelíbí její nadpis Count(ID), můžeme použít příkaz ve tvaru

```
SELECT Zamestnanci.OsobniCislo, Jmeno, Count(ID) AS "Počet projektů"
...
```

a počet projektů bude zobrazen pod jménem, které požadujeme.

Na začátku popisu příkazu `SELECT` jsme si říkali, že můžeme použít klauzuli `HAVING`. Tu použijeme v případech, kdy chceme dotaz omezit podmínkou, která obsahuje výraz vzniklý až po sloučení výsledku do skupin pomocí `GROUP BY`. Pokud chceme vypsat ty zaměstnance, kteří pracují alespoň na dvou projektech, použijeme příkaz

```
SELECT Zamestnanci.OsobniCislo, Jmeno, Count(ID)
FROM Zamestnanci, Proj_Zam, Projekty
WHERE (Zamestnanci.OsobniCislo = Proj_Zam.OsobniCislo)
      AND (Proj_Zam.ID_Projektu = Projekty.ID)
GROUP BY Zamestnanci.OsobniCislo
HAVING Count(ID) > 1
```

Dosud jsme si nepopsali význam klauzule `ORDER BY`. Pomocí ní určujeme způsob řazení výsledku. Výsledek je řazen podle obsahu položky uvedené za `ORDER BY`. Pokud tedy chceme zaměstnance vypsat podle abecedy, použijeme

```
SELECT * FROM Zamestnanci
ORDER BY Jmeno
```

Pokud chceme zaměstnance setřídit sestupně (od Z do A), použijeme modifikátor `DESC`

```
SELECT * FROM Zamestnanci
ORDER BY Jmeno DESC
```

Dosť bolo selektu. Podívejme se nyní na další příkazy SQL.

Mazání záznamů

Pro mazání záznamů z tabulky slouží příkaz `DELETE FROM` s následující syntaxí:

```
DELETE FROM «jméno tabulky» WHERE «podmínka»
```

Takový příkaz smaže z *«tabulky»* všechny záznamy, které vyhovují *«podmínce»*. Většinou potřebujeme smazat pouze jeden přesně určený záznam. Z tohoto důvodu se jako *«podmínka»* nejčastěji používá rovnost primárního klíče tabulky s nějakou hodnotou:

```
DELETE FROM Zamestnanci WHERE OsobniCislo = 1023
```

Dejte si na tuto funkci pozor. SQL-servery nebývají vybaveny funkcí Undo. Pokud si záznamy jednou smažete a nemáte databázi zazálohovanou, máte prostě smůlu. Také si dejte pozor na příkaz

```
DELETE FROM Zamestnanci
```

který smaže všechny záznamy v tabulce `Zamestnanci`.

Modifikace záznamu

Další často prováděnou operací se záznamy je jejich modifikace. V SQL k tomu slouží příkaz `UPDATE`:

```
UPDATE «jméno tabulky»
SET «jméno položky» = «hodnota položky»,
    «jméno položky» = «hodnota položky»,
    ...
    «jméno položky» = «hodnota položky»
WHERE «podmínka»
```

Pomocí *«podmínky»* určujeme, které záznamy se mají změnit. Pokud se například Alenka Novotná vdá a vezme si Procházku, musíme upravit její jméno v tabulce `Zamestnanci`. Víme, že Alenčino osobní číslo je 1168, takže změna není žádný problém

```
UPDATE Zamestnanci
SET Jmeno = 'Procházková Alena'
WHERE OsobniCislo = 1168
```

Příkaz `UPDATE` se hodí i pro provádění větších úprav v tabulkách. Například můžeme jednoduše zvýšit plat o pětistovku všem, kteří berou méně než 10 000 Kč:

```
UPDATE Zamestnanci
SET Plat = Plat + 500
WHERE Plat < 10000
```

Nastavení přístupových práv

Databáze často obsahují choulostivá data, která je potřeba chránit před nepovolanými zraky. Proto dnes již většina serverů vyžaduje, aby se uživatel na začátku práce se serverem přihlásil pomocí svého jména a hesla.

Pro každou tabulku v databázi pak můžeme definovat, který uživatel k ní má přístup a jaké operace s ní může provádět. K nastavení přístupových práv slouží příkaz `GRANT`.

```
GRANT «práva»
ON «jméno tabulky»
TO «jméno uživatele»
```

«Práva» odpovídají jednotlivým příkazům pro manipulaci se záznamy. V této části příkazu `GRANT` můžeme použít seznam některých ze slov `SELECT`, `INSERT`, `UPDATE` a `DELETE`. Každé slovo přitom odpovídá právu pro provádění daného SQL-příkazu. Pokud chceme přidělit všechna práva, můžeme použít slovo `ALL`.

Za direktivou `ON` musíme zadat jméno tabulky, pro kterou daná oprávnění platí. Jako poslední se uvádí jméno uživatele, kterému mají být přidělena nastavovaná práva.

Pokud se nějaký uživatel proviní a chceme mu nějaká práva odebrat, použijeme příkaz `REVOKE`, jehož parametry jsou obdobné jako u příkazu `GRANT`.

```
REVOKE «práva»
ON «jméno tabulky»
FROM «jméno uživatele»
```

Ve většině databázových systémů má uživatel, který databázi vytvořil, práva k provádění všech operací. Z našich skriptů bychom se však měli hlásit pod jiným uživatelským jménem. Tomuto uživateli pak přiřadíme pouze potřebná práva. Pokud bude skript sloužit pouze k prohlížení nějaké tabulky, je zbytečné k ní přistupovat jako uživatel, který má i právo pro mazání a vkládání záznamů.

Pokud chceme například uživateli `guest` poskytnout plný přístup k tabulce `Zamestnanci`, použijeme příkaz

```
GRANT ALL ON Zamestnanci TO guest
```

Pokud chceme uživateli `browser` poskytnout pouze přístup k prohlížení tabulky, použijeme

```
GRANT SELECT ON Zamestnanci TO browser
```

Tento způsob nastavování přístupových práv funguje na většině serverů. Existují však i výjimky, na které upozorníme v následujícím textu.

Systém MySQL rovněž podporuje příkaz `GRANT`. Podpora je však čistě iluzorní. MySQL se tváří tak, jako by příkaz znalo, aby zachovalo kompatibilitu se standardem SQL. Ve skutečnosti se však žádná práva nezmění. MySQL používá

vlastní mechanismus řízení přístupů, který využívá tří tabulek `user`, `db` a `host` v databázi `mysql`.

Podrobný popis nastavení přístupových práv nalezneme v dokumentaci. My si ukážeme, jak přidat nového uživatele a jak mu povolit přístup k určitým operacím v dané databázi.

Nejprve se však musíme přihlásit jako správce databáze `mysql`. K tomu můžeme použít např. příkaz:

```
mysql -u root -p mysql
```

Nyní přidáme uživatele `johny`, který bude mít heslo `WaLk@R3` a bude se moci hlásit pouze z lokálního počítače. To znamená, že WWW-server, který bude obsluhovat skript přistupující pod jménem `johny` k MySQL, musí běžet na stejném počítači jako MySQL. Pokud běží na jiném, musíme odpovídajícím způsobem změnit text `localhost` v následujícím příkazu:[3]

```
INSERT INTO user (host, user, password)
VALUES ('localhost', 'johny', password('WaLk@R3'));
```

Nyní uživateli `johny` přidáme přístup pro čtení do databáze `napoje`:

```
INSERT INTO db (host, db, user, select_priv)
VALUES ('%', 'napoje', 'johny', 'Y');
```

Pokud bychom chtěli přidat i další práva, můžeme pomocí příkazu `INSERT` nastavit na '`Y`' i další položky `insert_priv`, `delete_priv` a `update_priv`. Každá položka odpovídá jednomu SQL-příkazu. Kromě toho existují ještě privilegia `create_priv` a `drop_priv`, která umožňují vytvoření, resp. smazání tabulky v databázi. Tyto příkazy asi nebudeme v běžných skriptech používat, a proto nemá cenu tato práva přiřazovat uživateli používanému pro přihlašování k MySQL ze skriptů.

Po změně přístupových práv, musíme MySQL server přinutit k znovunačtení přístupových práv. To provedeme příkazem

```
mysql -u root -p reload
```

Nesmíme zapomenout, že pomocí příkazu `GRANT` se nastavují přístupová práva pro jednotlivé tabulky. Oproti tomu MySQL umožňuje nastavení přístupových práv pouze pro celou databázi najednou.

[3] Pozorný čtenář si jistě všiml, že příkaz `INSERT` má trošku jinou syntaxi, než jakou jsme popsali dříve. Zde uvedená syntaxe umožňuje vložení záznamu, i když nezadáme hodnotu pro všechny položky. Nezadané položky jsou v tomto případě vyplněny standardní hodnotou, kterou lze určit v příkazu `CREATE TABLE`. Standardní hodnota se zadává za typem položky pomoci direktivy `DAFAULT`, za kterou následuje standardní hodnota.

6.5 Podpora databází v PHP

PHP podporuje několik databázových systémů. I když k práci s každým z nich slouží jiné funkce, jejich použití i jména jsou velice podobná. Obecně můžeme říci, že funkce pro práci s určitým SŘBD začínají jeho názvem. PHP tak obsahuje funkce začínající na `MySQL_`, které slouží pro práci s MySQL, funkce začínající na `Pg_`, které slouží pro práci s PostgreSQL, funkce začínající na `Ifx_` slouží k práci s Informixem a mohli bychom pokračovat dál a dál. Pokud to snad ještě někdo nepochopil, připomínám, že funkce pro práci s ODBC začínají na `ODBC_`.

Práce s databázemi je v PHP velice jednoduchá. Nejprve se musíme připojit k SQL-serveru. K tomu obvykle slouží funkce pojmenovaná `xxx_Connect()`. Po připojení k serveru můžeme zadávat SQL-příkazy. Nejčastěji k tomu slouží funkce `xxx_Exec()` nebo `xxx_Query()`. Získaný výsledek můžeme číst funkcemi `xxx_Fetch_Row()` a `xxx_Result()`. Těchto pár funkcí nám při práci s databází obvykle stačí.

V referenčním přehledu funkcí naleznete podrobně popsané všechny funkce, které můžete použít pro práci s ODBC, MySQL a PostgreSQL. Pokud potřebujete pracovat s jinou databází, naleznete popis funkcí v originální dokumentaci k PHP. Funkce se pouze nepatrně liší svou syntaxí, jejich význam a použití je stejné jako u funkcí, které zde popíšeme.

V této sekci stručně vyložíme, jak v PHP používat funkce pro práci s ODBC, MySQL a PostgreSQL. V další podrobnější sekci si ukážeme, jak v PHP vytvořit skripty, které uživatelům zpřístupní všechny operace prováděné s databází, jako je prohlížení, prohledávání, vkládání, rušení a modifikování záznamů.

Podpora MySQL

Práci s MySQL musíme zahájit připojením k serveru pomocí `MySQL_Connect()`. Parametrem funkce je jméno počítače, na kterém MySQL běží. Pokud chceme přistupovat k databázi, která je chráněná jménem a heslem, zadáme i jméno a heslo. Výsledkem funkce je číslo, které identifikuje připojení k serveru a můžeme jej použít v dalších funkcích.

```
$spojeni = MySQL_Connect("localhost");   // připojení k anonymní databázi
```

Nyní musíme vybrat databázi, se kterou budeme pracovat, pomocí funkce `MySQL_Select_DB()`. Parametrem této funkce je jméno databáze a číslo spojení. Pokud ve skriptu pracuje pouze s jedním spojením a jednou databází, můžeme klidně číslo spojení vynechat. Použije se to jediné již vytvořené:

```
MySQL_Select_DB("test");        // vybereme naši pokusnou databázi test
```

Nyní již můžeme do databáze klást dotazy zapsané v jazyce SQL. Slouží k tomu funkce `MySQL_Query()`. Jako první parametr zadáme SQL-příkaz a jako druhý číslo spojení. Jelikož pracujeme pouze s jedním spojením k databázi, můžeme ho

s klidným srdcem vynechat. Výsledkem funkce `MySQL_Query()` je číslo identifikující výsledek. Pomocí tohoto čísla se pak na výsledek SQL-příkazu odvoláváme v dalších funkcích. Pro výběr všech údajů z tabulky `Zamestnanci` můžeme použít:

```
$vysledek = MySQL_Query("SELECT * FROM Zamestnanci ORDER BY Jmeno");
```

Proměnná `$vysledek` nyní obsahuje identifikátor výsledku. Pomocí něj můžeme číst jednotlivé záznamy výsledku a zjišťovat spoustu dalších zajímavých věcí. Například pomocí funkce `MySQL_Num_Rows()` zjistíme počet záznamů výsledku. Ten může například vypsat:

```
echo "V tabulce Zamestnanci je ".MySQL_Num_Rows($vysledek)." záznamů";
```

Pro čtení jednotlivých záznamů výsledku můžeme použít několik funkcí. Asi nejpraktičtější z nich je `MySQL_Fetch_Array()`, která přečte jeden záznam výsledku a obsah jeho položek uloží do asociativního pole.

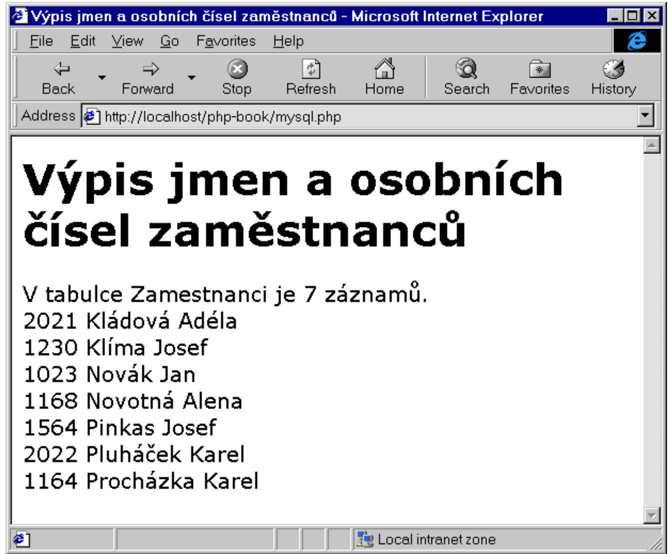
Funkce vždy vrátí obsah záznamu a přesune se na další. Pokud již funkce vrátila všechny záznamy, vrací hodnotu `false`. Této vlastnosti můžeme šikovně využít pro zobrazení výsledku. Aby byl kód jednoduchý, budeme zobrazovat pouze osobní číslo a jméno každého zaměstnance:

```
while ($zaznam = MySQL_Fetch_Array($vysledek))
    echo $zaznam["OsobniCislo"]." ".$zaznam["Jmeno"]."<BR>\n";
```

A tím jsme v podstatě hotovi. Skript se připojil k databázi, provedl dotaz a zobrazil jeho výsledek. Ve funkci `MySQL_Query()` můžeme samozřejmě používat všechny SQL-příkazy, nejen příkaz `SELECT`. Použití ostatních příkazů je do jisté míry jednodušší, protože jimi vrácený výsledek neobsahuje žádné záznamy.

Poslední, co je dobré při práci s databází udělat, je na konci skriptu ukončit spojení s MySQL pomocí volání `MySQL_Close($spojeni)`. Pokud na volání zapomeneme, nic se neděje. PHP na konci každého skriptu automaticky ukončí spojení se všemi SQL-servery.

Zatím jsme nediskutovali problémy ošetření chyb. V krajních případech může dojít k tomu, že se PHP nemůže připojit k MySQL nebo že provádění SQL-příkazu způsobí chybu. Pokud k tomuto stavu dojde, PHP normálně vypíše chybové hlášení a ukončí běh skriptu. Mnohem elegantnější a uživatelsky přijatelnější je vlastní obsloužení chyb. Pomocí znaku '`@`' potlačíme zobrazování standardních chybových zpráv a chybové stavy ošetříme sami. Ošetření je velice snadné. Pokud se nepodaří připojit k MySQL, vrací funkce `MySQL_Connect()` hodnotu `false`. Zcela obdobně i `MySQL_Query()` vrací `false`, pokud při provádění SQL-příkazu dojde k chybě. Konečná verze naše skriptu `mysql.php` počítá i s ošetřením chyb.



Obr. 6-11: Výsledek našeho skriptu pro spolupráci s MySQL

Příklad: `mysql.php`

```
<HTML>
<HEAD>
<TITLE>Výpis jmen a osobních čísel zaměstnanců</TITLE>
</HEAD>
<BODY>
<H1>Výpis jmen a osobních čísel zaměstnanců</H1>
<?
do {
    @$spojeni = MySQL_Connect("localhost");
    if (!$spojeni):
        echo "Nepodařilo se připojit k MySQL.<BR>\n";
        break;
    endif;
    MySQL_Select_DB("test");
    @$vysledek = MySQL_Query("SELECT * FROM Zamestnanci ORDER BY Jmeno");
    if (!$vysledek):
        echo "Došlo k chybě při zpracování dotazu v databázi.<BR>\n";
        break;
    endif;
    echo "V tabulce Zamestnanci je ".MySQL_Num_Rows($vysledek)." záznamů.<BR>\n";
    while ($zaznam = MySQL_Fetch_Array($vysledek))
```

```
        echo $zaznam["OsobniCislo"]." ".$zaznam["Jmeno"]."<BR>\n";
    MySQL_Close($spojeni);
} while (false);
?>
</BODY>
</HTML>
```

Podpora ODBC

Jak jsme již řekli, před zahájením práce s databázovým serverem se k němu musíme připojit. V ODBC k tomu slouží funkce `ODBC_Connect()`. Funkce má tři parametry. Prvním je jméno datového zdroje. Toto jméno je shodné se jménem zadávaným v manažeru ODBC při vytváření datového zdroje. Dalšími parametry jsou uživatelské jméno a heslo, kterými je přístup ke zdroji chráněn.

Funkce `ODBC_Connect()` vrací číslo, které identifikuje vytvořené spojení. Toto číslo pak používáme v dalších funkcích pro identifikaci spojení. Výsledek funkce si proto musíme uložit do nějaké proměnné, pro další použití:

```
$spojeni = ODBC_Connect("test", "guest", "");
```

Vytvořili jsme spojení k datovému zdroji `test`. Pro přihlášení k serveru jsme použili jméno `guest`, které je bez hesla. Nyní již můžeme serveru zadávat SQL--příkazy. Slouží k tomu funkce `ODBC_Exec()`. Jejími parametry jsou číslo spojení a dotaz zapsaný v jazyce SQL. Funkce vrací identifikátor výsledku, kterým se na získaná data odvoláváme v dalších funkcích. Pro výběr všech údajů z tabulky `Zamestnanci` můžeme použít:

```
$vysledek = ODBC_Exec($spojeni, "SELECT * FROM Zamestnanci ORDER BY Jmeno");
```

Pomocí proměnné `$vysledek` nyní můžeme přistupovat k výsledku. Nejčastěji k tomu použijeme funkci `Fetch_Row()`. Tato funkce zpřístupní vždy další záznam výsledku. Pokud již nejsou k dispozici žádné záznamy, vrací funkce `false`. Díky tomu můžeme pomocí jednoduchého cyklu zpracovat všechny záznamy výsledku. Jednotlivé položky aktuálního záznamu můžeme číst pomocí funkce `ODBC_Result()`. Jejími parametry jsou identifikátor výsledku a jméno položky. K zobrazení osobních čísel a jmen našich zaměstnanců můžeme použít následujících pár[4] řádek:

```
while (ODBC_Fetch_Row($vysledek))
    echo ODBC_Result($vysledek, "OsobniCislo")." ".
         ODBC_Result($vysledek, "Jmeno")."<BR>\n";
```

[4] Jak vidíte, v době počítačů nemusí být pár vždy dvě.

No a tím jsme v podstatě hotovi. Slušný skript by měl na konci ukončit spojení se zdrojem dat pomocí funkce `ODBC_Close()`, ale není to nezbytně nutné, protože PHP všechna otevřená uzavře automaticky samo.

Zatím jsme nediskutovali problémy ošetření chyb. V krajních případech může dojít k tomu, že se PHP nemůže připojit ke zdroji dat nebo že provádění SQL-příkazu způsobí chybu. Pokud k tomuto stavu dojde, PHP normálně vypíše chybové hlášení a ukončí běh skriptu. Mnohem elegantnější a uživatelsky přijatelnější je vlastní obsloužení chyb. Pomocí znaku '`@`' potlačíme zobrazování standardních chybových zpráv a chybové stavy ošetříme sami. Ošetření je velice snadné. Pokud se nepodaří připojit k datovému zdroji, vrací funkce `ODBC_Connect()` hodnotu `false`. Zcela obdobně i `ODBC_Exec()` vrací `false`, pokud při provádění SQL-příkazu dojde k chybě. Konečná verze našeho skriptu `odbc.php` počítá i s ošetřením chyb.

Příklad: `odbc.php`

```
<HTML>
<HEAD>
<TITLE>Výpis jmen a osobních čísel zaměstnanců</TITLE>
</HEAD>
<BODY>
<H1>Výpis jmen a osobních čísel zaměstnanců</H1>
<?
do {
    @$spojeni = ODBC_Connect("test", "guest", "");
    if (!$spojeni):
        echo "Nepodařilo se připojit ke zdroji dat.<BR>\n";
        break;
    endif;
    @$vysledek = ODBC_Exec($spojeni, "SELECT * FROM Zamestnanci ORDER BY Jmeno");
    if (!$vysledek):
        echo "Došlo k chybě při zpracování dotazu v databázi.<BR>\n";
        break;
    endif;
    echo "V tabulce Zamestnanci je ".ODBC_Num_Rows($vysledek)." záznamů.<BR>\n";
    while (ODBC_Fetch_Row($vysledek))
        echo ODBC_Result($vysledek, "OsobniCislo")." ".
             ODBC_Result($vysledek, "Jmeno")."<BR>\n";
    ODBC_Close($spojeni);
} while (false);
?>
</BODY>
</HTML>
```

Ve skriptu jsme navíc použili funkci ODBC_Num_Rows(), která vrací počet záznamů ve výsledku. Některé ODBC ovladače ji však nepodporují a vrací vždy −1. Nejedná se tedy o chybu vašeho skriptu, ale neschopnost ostatního softwaru.

Podpora PostgreSQL

Práci s PostgreSQL musíme zahájit připojením k serveru pomocí funkce Pg_Connect(). Parametrem funkce je jméno počítače a databáze, ke které se připojujeme. Výsledkem funkce je číslo, které identifikuje připojení k serveru a můžeme jej použít v dalších funkcích.

```
$spojeni = Pg_Connect("host=localhost dbname=test");
```

Nyní již můžeme serveru zadávat SQL-příkazy. Slouží k tomu funkce Pg_Exec(). Jejími parametry jsou číslo spojení a dotaz zapsaný v jazyce SQL. Funkce vrací identifikátor výsledku, kterým se na získaná data odvoláváme v dalších funkcích. Pro výběr všech údajů z tabulky Zamestnanci můžeme použít:

```
$vysledek = Pg_Exec($spojeni, "SELECT * FROM Zamestnanci ORDER BY Jmeno");
```

Proměnná $vysledek nyní obsahuje identifikátor výsledku. Pomocí něj můžeme číst jednotlivé záznamy výsledku a zjišťovat spoustu dalších zajímavých věcí. Například pomocí funkce Pg_NumRows() zjistíme počet záznamů výsledku. Ten můžeme například vypsat:

```
echo "V tabulce Zamestnanci je ".Pg_NumRows($vysledek)." záznamů";
```

Pro čtení jednotlivých záznamů výsledku můžeme použít několik funkcí. Asi nejpraktičtější z nich je Pg_Fetch_Array(), která přečte jeden záznam výsledku a obsah jeho položek uloží do asociativního pole. Funkci jako parametr musíme předat číslo záznamu, který chceme přečíst. Záznamy jsou číslovány od nuly. K zobrazení osobních čísel a jmen všech zaměstnanců můžeme použít následující kód:

```
for ($i=0; $i < Pg_NumRows($vysledek); $i++):
    $zaznam = Pg_Fetch_Array($vysledek, $i);
    echo $zaznam["OsobniCislo"]." ".$zaznam["Jmeno"]."<BR>\n";
endfor;
```

A tím jsme v podstatě hotovi. Skript se připojil k databázi, provedl dotaz a zobrazil jeho výsledek. Ve funkci Pg_Exec() můžeme samozřejmě používat všechny SQL-příkazy, nejen příkaz SELECT. Použití ostatních příkazů, jako INSERT, UPDATE nebo DELETE, je do jisté míry jednodušší, protože jimi vrácený výsledek neobsahuje žádné záznamy.

Poslední, co je dobré při práci s databází udělat, je na konci skriptu ukončit spojení s PostgreSQL pomocí volání Pg_Close($spojeni). Pokud na volání

zapomeneme, nic se neděje. PHP na konci každého skriptu automaticky ukončí spojení se všemi SQL-servery.

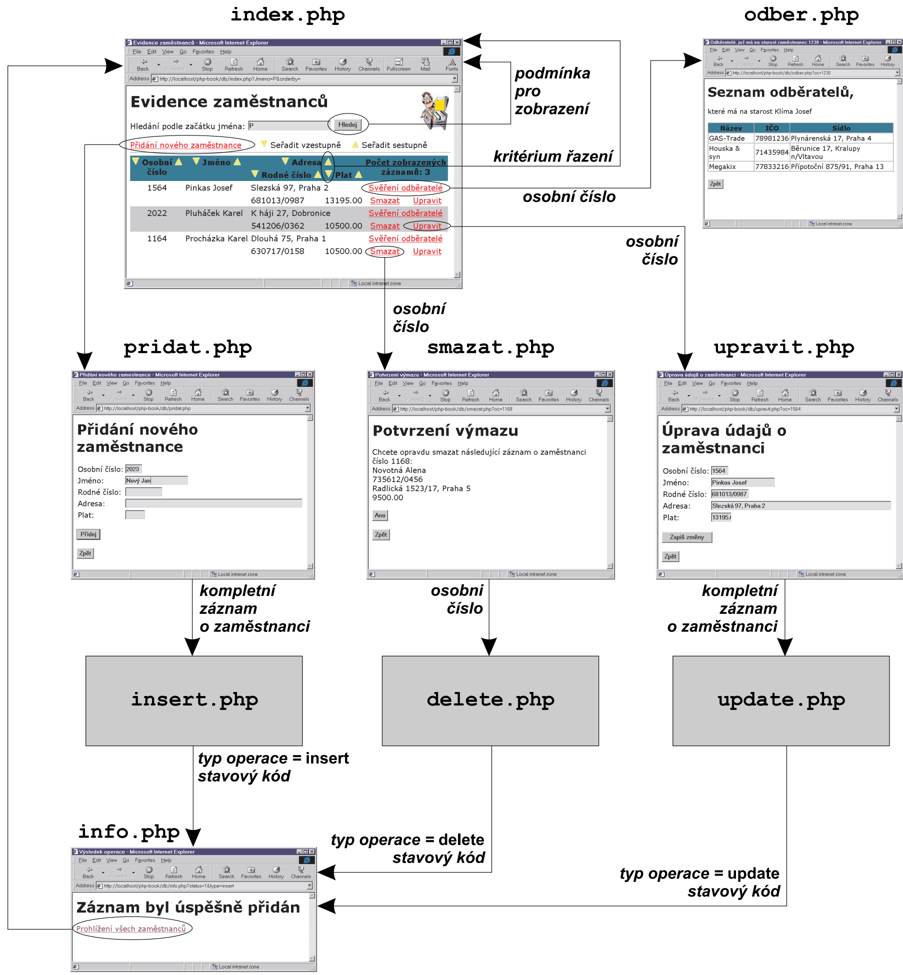
Zatím jsme nediskutovali problémy ošetření chyb. V krajních případech může dojít k tomu, že se PHP nemůže připojit k PostgreSQL nebo že provádění SQL-příkazu způsobí chybu. Pokud k tomuto stavu dojde, PHP normálně vypíše chybové hlášení a ukončí běh skriptu. Mnohem elegantnější a uživatelsky přijatelnější je vlastní obsloužení chyb. Pomocí znaku '`@`' potlačíme zobrazování standardních chybových zpráv a chybové stavy ošetříme sami. Ošetření je velice snadné. Pokud se nepodaří připojit k PostgreSQL, vrací funkce `Pg_Connect()` hodnotu `false`. Zcela obdobně i `Pg_Exec()` vrací `false`, pokud při provádění SQL-příkazu dojde k chybě. Konečná verze naše skriptu `psql.php` počítá i s ošetřením chyb.

Příklad: `psql.php`

```
<HTML>
<HEAD>
<TITLE>Výpis jmen a osobních čísel zaměstnanců</TITLE>
</HEAD>
<BODY>
<H1>Výpis jmen a osobních čísel zaměstnanců</H1>
<?
do {
    @$spojeni = Pg_Connect("host=localhost dbname=test");
    if (!$spojeni):
        echo "Nepodařilo se připojit k PostgreSQL.<BR>\n";
        break;
    endif;
    @$vysledek = Pg_Exec($spojeni, "SELECT * FROM Zamestnanci ORDER BY Jmeno");
    if (!$vysledek):
        echo "Došlo k chybě při zpracování dotazu v databázi.<BR>\n";
        break;
    endif;
    echo "V tabulce Zamestnanci je ".Pg_NumRows($vysledek)." záznamů.<BR>\n";
    for ($i=0; $i < Pg_NumRows($vysledek); $i++):
        $zaznam = Pg_Fetch_Array($vysledek, $i);
        echo $zaznam["OsobniCislo"]." ".$zaznam["Jmeno"]."<BR>\n";
    endfor;
    Pg_Close($spojeni);
} while (false);
?>
</BODY>
</HTML>
```

6.6 Manipulace s daty pomocí prohlížeče

Víme, co je to databáze, známe základy jazyka SQL a umíme ho zařadit do našich skriptů. Nyní tedy zbývá pouhá maličkost — zkombinovat vše dohromady a vytvořit sadu skriptů, která umožní uživateli v prostředí jeho prohlížeče kompletní práci s daty v databázi. To kromě prohlížení záznamů zahrnuje i jejich prohledávání, přidávání, mazání a modifikaci. Celé se to ještě může zkomplikovat existencí vztahů mezi tabulkami. V této sekci si proto na příkladě naší databáze ukážeme, jak její část zcela zpřístupnit pomocí několika poměrně jednoduchých skriptů v PHP.



Obr. 6-12: Schéma ukázkové aplikace

Naším úkolem bude pomocí webového rozhraní zpřístupnit údaje uložené v tabulce `Zamestnanci`. Základem aplikace bude stránka, která umožní prohlížení tabulky. Kromě toho nám stránka umožní nastavit filtr, kterým se budou určovat jména zaměstnanců, jejichž záznamy chceme vidět. Tím zaručíme, že pomocí skriptu půjde pracovat i s velkými tabulkami, které mají mnoho záznamů a jejichž ruční prohledávání by bylo zdlouhavé. Navíc umožníme uživateli seřadit výstup podle obsahu libovolné z položek. Tato úvodní stránka, uložená ve skriptu `index.php`, bude samozřejmě obsahovat odkaz, který vyvolá stránku pro přidání nového zaměstnance. Kromě toho bude u každého záznamu odkaz, který umožní záznam upravit nebo smazat. K dispozici bude i odkaz, který vypíše všechny odběratele, jež má daný zaměstnanec na starost.

Podívejme se tedy na klíčové části skriptu `index.php`, který je vstupní branou k tabulce `Zamestnanci`. Skript začíná, stejně jako všechny ostatní, příkazem, který s odpovědí zašle i čas vypršení její platnosti. Čas nastavíme na aktuální čas. Tím máme zaručeno, že stránka bude vždy čtena ze serveru a bude obsahovat aktuální informace. V opačném případě by mohla být stránka načítána pouze z vyrovnávací paměti prohlížeče nebo proxy-serverů a mohlo by dojít k zobrazení neaktuálních dat.

```
Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT");
```

Další dva řádky skriptu

```
if (!IsSet($Jmeno)) $Jmeno = "";
if (!IsSet($orderby)) $orderby = "";
```

slouží k inicializaci proměnných `$Jmeno` a `$orderby`. Proměnná `$Jmeno` obsahuje řetězec, kterým mají začínat jména všech zobrazovaných zaměstnanců. Proměnná `$orderby` obsahuje jméno položky, podle které se má výstup řadit.

Pro zadání začátku jména, které se má v tabulce `Zamestnanci` vyhledat, slouží následující formulář:

```
<FORM ACTION=index.php>
Hledání podle začátku jména:
<INPUT NAME=Jmeno SIZE=25 VALUE="<?echo $Jmeno?>">
<INPUT TYPE=SUBMIT VALUE="Hledej">
<INPUT TYPE=HIDDEN NAME=orderby VALUE="<?echo $orderby?>">
</FORM>
```

Formulář do vstupního pole pomocí `<?echo $Jmeno?>` vypíše vždy poslední hodnotu prohledávání, takže uživatel nemusí prohledávací řetězec zadávat znovu po každém znovunačtení stránky. Aby byl zachován způsob řazení pro různá hledaná jména, je způsob řazení předáván pomocí skrytého pole formuláře.

V další části skriptu si definujeme vlastní funkci `TlacitkoProRazeni()`. Předávají se jí dva parametry — jméno položky v tabulce a jméno, pod kterým

má být položka zobrazena. Funkce vrací jméno položky a navíc před ní i za ní umístí obrázek malého trojúhelníčku, který slouží jako odkaz. Odkaz vede zpět na stránku `index.php`, navíc je však pomocí odkazu nastavena proměnná `$orderby`, která definuje položku, podle které se budou řadit záznamy. Jako parametr odkazu předáváme i podmínku pro vyhledávání, aby zůstala zachována i po změně kritéria řazení.

```
function TlacitkoProRazeni($polozka, $popis)
{
    global $Jmeno;

    return
        "<A HREF='index.php?orderby=$polozka&Jmeno=".
        URLEncode($Jmeno)."'>".
        "<IMG SRC=asc.gif BORDER=0 WIDTH=20 HEIGHT=20></A> ".
        $popis." ".
        "<A HREF='index.php?orderby=$polozka+DESC&Jmeno=".
        URLEncode($Jmeno)."'>".
        "<IMG SRC=desc.gif BORDER=0 WIDTH=20 HEIGHT=20></A>";
}
```

Pokud tedy funkci vyvoláme např. s parametry `RC` a `Rodné číslo` a aktuální podmínka pro zobrazené záznamy je `Nov`, je vrácen následující HTML-kód:

```
<A HREF='index.php?orderby=RC&Jmeno=Nov'>
    <IMG SRC=asc.gif BORDER=0 WIDTH=20 HEIGHT=20></A>
 Rodné číslo 
<A HREF='index.php?orderby=RC+DESC&Jmeno=Nov'>
    <IMG SRC=desc.gif BORDER=0 WIDTH=20 HEIGHT=20></A>
```

Po zvolení jednoho z odkazů bude znovu vyvolán skript `index.php`. Proměnná `$orderby` bude nastavena na hodnotu `RC` nebo `RC DESC` (pro sestupné třídění) a proměnná `$Jmeno` na hodnotu `Nov`.

Následně se ve skriptu musíme připojit k databázi

```
@$spojeni = ODBC_Connect("test", "", "");
if (!$spojeni):
    echo "Nepodařilo se připojit k databázi.\n";
    break;
endif;
```

Poté si připravíme SQL-dotaz. Pokud je proměnná `$Jmeno` prázdná, budeme zobrazovat všechny záznamy. V opačném případě omezíme záznamy pomocí klauzule `WHERE`:

```
if ($Jmeno!="")
    $Podminka = "WHERE Jmeno LIKE '".AddSlashes($Jmeno)."%'";
else
    $Podminka = "";
```

Funkce `AddSlashes()` převede nebezpečné znaky, které by mohly narušit syntaxi dotazu, na escape sekvence. Za obsah formulářového pole ještě doplníme '%' a tím máme zaručeno, že výsledkem dotazu budou všechny záznamy, jejichž jméno začíná na text uložený v proměnné `$Jmeno`.

Za normálních okolností řadíme záznamy podle jména. Proměnná `$orderby` to však může změnit:

```
if ($orderby!="")
    $OrderBy = "ORDER BY $orderby";
else
    $OrderBy = "ORDER BY Jmeno";
```

Nyní odešleme databázovému serveru dotaz, který vybere všechny záznamy vyhovující dané podmínce a seřadí je podle našeho přání (proměnné `$Podminka` a `$OrderBy` obsahují části příkazu `SELECT`):

```
@$vysledek = ODBC_Exec($spojeni,
    "SELECT * FROM Zamestnanci $Podminka $OrderBy");
if (!$vysledek):
    echo "Zadanému kritériu nevyhovuje žádný zaměstnanec.\n";
    break;
endif;
```

Nyní máme výsledek dotazu. Stačí jej přehledně vypsat. My zvolíme výpis do tabulky. Náš skript musí vygenerovat záhlaví tabulky. V záhlaví tabulky použijeme funkci `TlacitkoProRazeni()`, která vytvoří tlačítka pro změnu řazení. Poté pomocí cyklu vypíšeme kompletní výsledek dotazu. Kromě jednotlivých položek tabulky vypisujeme pro každý záznam odkazy, které vedou k dalším skriptům pro výmaz a modifikaci záznamu a pro vypsání svěřených odběratelů. Součástí odkazu je osobní číslo zaměstnance, které ho jednoznačně identifikuje.

Příklad: `index.php`

```
<?
    Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT");// neukládej do cache
    if (!IsSet($Jmeno)) $Jmeno = "";            // inicializace proměnných
    if (!IsSet($orderby)) $orderby = "";
?>
<HTML>
<HEAD>
<TITLE>Evidence zaměstnanců</TITLE>
```

```
<STYLE TYPE="text/css"><!--
:link, :visited, :active { color: red } /* všechny odkazy budou červené */
--></STYLE>
</HEAD>
<BODY>
<IMG SRC=evidence.jpg WIDTH=74 HEIGHT=100 ALIGN=RIGHT>
<H1>Evidence zaměstnanců</H1>

<FORM ACTION=index.php>
Hledání podle začátku jména:
<INPUT NAME=Jmeno SIZE=25 VALUE="<?echo $Jmeno?>">
<INPUT TYPE=SUBMIT VALUE="Hledej">
<INPUT TYPE=HIDDEN NAME=orderby VALUE="<?echo $orderby?>">
</FORM>

<P><A HREF="pridat.php">Přidání nového zaměstnance</A>
     
<IMG SRC="asc.gif" WIDTH=20 HEIGHT=20> Seřadit vzestupně
  
<IMG SRC="desc.gif" WIDTH=20 HEIGHT=20> Seřadit sestupně
<HR>

<?
function TlacitkoProRazeni($polozka, $popis)
// vygeneruje tlačítka pro změnu řazení
// $polozka - název položky tabulky
// $popis - text, který se použije jako popis
{
    global $Jmeno;

    return
        "<A HREF='index.php?orderby=$polozka&Jmeno=".
         URLEncode($Jmeno)."'>".
        "<IMG SRC=asc.gif BORDER=0 WIDTH=20 HEIGHT=20></A> ".
        $popis." ".
        "<A HREF='index.php?orderby=$polozka+DESC&Jmeno=".
         URLEncode($Jmeno)."'>".
        "<IMG SRC=desc.gif BORDER=0 WIDTH=20 HEIGHT=20></A>";
}

do {
    @$spojeni = ODBC_Connect("test", "", "");       // připojení k databázi
```

```
if (!$spojeni):
    echo "Nepodařilo se připojit k databázi.\n";
    break;
endif;

if ($Jmeno!="")             // vytvoření podmínky pro zobrazované záznamy
    $Podminka = "WHERE Jmeno LIKE '".AddSlashes($Jmeno)."%'";
else
    $Podminka = "";

if ($orderby!="")                   // vytvoření kritéria pro řazení
    $OrderBy = "ORDER BY $orderby";
else
    $OrderBy = "ORDER BY Jmeno";    // standardně se řadí podle jména

@$vysledek = ODBC_Exec($spojeni,    // výběr údajů z tabulky
    "SELECT * FROM Zamestnanci $Podminka $OrderBy");
if (!$vysledek):
    echo "Zadanému kritériu nevyhovuje žádný zaměstnanec.\n";
    break;
endif;

// záhlaví tabulky pro výsledky
echo "<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=4>\n";
echo "<TR BGCOLOR=TEAL VALIGN=TOP>\n";
echo "<TH ROWSPAN=2>".TlacitkoProRazeni("OsobniCislo", "Osobní").
     "<BR>číslo</TH>\n";
echo "<TH ROWSPAN=2>".TlacitkoProRazeni("Jmeno", "Jméno")."</TH>\n";
echo "<TH COLSPAN=2>".TlacitkoProRazeni("Adresa", "Adresa")."</TH>\n";
echo "<TH COLSPAN=2 ROWSPAN=2>";
if (ODBC_Num_Rows($vysledek)!=-1)
    echo "Počet zobrazených<BR>záznamů: ".ODBC_Num_Rows($vysledek);
echo "</TH></TR>\n";
echo "<TR BGCOLOR=TEAL>\n";
echo "<TH>".TlacitkoProRazeni("RC", "Rodné číslo")."</TH>\n";
echo "<TH>".TlacitkoProRazeni("Plat", "Plat")."</TH>\n";
echo "</TR>\n";

$i = 0;                         // čítač řádků tabulky
while(ODBC_Fetch_Row($vysledek)):
    if (($i%2)==1)              // sudé a liché řádky mají jinou barvu
        echo "<TR VALIGN=TOP BGCOLOR=SILVER>";
```

```
        else
            echo "<TR VALIGN=TOP>";

        // vypsání jednoho záznamu tabulky
        $OC = ODBC_Result($vysledek, "OsobniCislo");
        echo "<TD ROWSPAN=2 ALIGN=CENTER>$OC</TD>";
        echo "<TD ROWSPAN=2>".ODBC_Result($vysledek, "Jmeno")."</TD>";
        echo "<TD COLSPAN=2>".ODBC_Result($vysledek, "Adresa")."</TD>";
        echo "<TD COLSPAN=2 ALIGN=CENTER>". // odkaz pro prohlížení odběratelů
            "<A HREF='odber.php?oc=$OC'>Svěření odběratelé</A></TD>";
        echo "</TR>";
        if (($i%2)==1)
            echo "<TR VALIGN=TOP BGCOLOR=SILVER>";
        else
            echo "<TR VALIGN=TOP>";
        echo "<TD>".ODBC_Result($vysledek, "RC")."</TD>";
        echo "<TD ALIGN=RIGHT>".ODBC_Result($vysledek, "Plat")."</TD>";
        echo "<TD ALIGN=CENTER>".          // odkaz pro smazání záznamu
            "<A HREF='smazat.php?oc=$OC'>Smazat</A></TD>";
        echo "<TD ALIGN=CENTER>".          // odkaz pro úpravu záznamu
            "<A HREF='upravit.php?oc=$OC'>Upravit</A></TD>";
        $i++;                   // aktualizace čítače
    endwhile;
    echo "</TABLE>\n";          // konec tabulky
} while(false);
?>

</BODY>
</HTML>
```

Část aplikace, která slouží k prohlížení, je hotova. Nyní nás čeká další úkol — naprogramovat přidávání nových zaměstnanců do tabulky. Víme, že pro přidání nového zaměstnance je volán skript `pridat.php`. Úkolem tohoto skriptu je vygenerovat formulář sloužící pro zapsání údajů o zaměstnanci. Skript navrhneme tak protřele, aby automaticky doplnil nějaké volné osobní číslo. Dosud neobsazené osobní číslo můžeme získat například jako největší dosavadní osobní číslo zvýšené o jedna.

Příklad: `pridat.php`

```
<?
    Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT") // neukládat do cache
?>
<HTML>
<HEAD>
<TITLE>Přidání nového zamestnance</TITLE>
</HEAD>
<BODY>
<?
do {
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):                     // podařilo se připojit k databázi
        echo "Nepodařilo se připojit k databázi.\n";
        break;
    endif;
    @$vysledek = ODBC_Exec($spojeni,    // zjištění nového čísla zaměstnance
        "SELECT Max(OsobniCislo)+1 FROM Zamestnanci");
    if (!$vysledek):                    // nalezli jsme nové číslo?
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;
    if (ODBC_Fetch_Row($vysledek))      // přečtení nového osobního čísla
        $OC = ODBC_Result($vysledek, 1);
    else
        $OC = "";
} while (false);
?>

<H1>Přidání nového zaměstnance</H1>

<FORM ACTION="insert.php" METHOD=POST>
<TABLE>
<TR><TD>Osobní číslo:<TD><INPUT NAME=OsobniCislo VALUE="<?echo $OC?>" SIZE=4>
<TR><TD>Jméno:<TD><INPUT NAME=Jmeno>
<TR><TD>Rodné číslo:<TD><INPUT NAME=RC SIZE=11>
<TR><TD>Adresa:<TD><INPUT NAME=Adresa SIZE=60>
<TR><TD>Plat:<TD><INPUT NAME=Plat SIZE=5>
</TABLE>
<P><INPUT TYPE=SUBMIT VALUE="Přidej">
</FORM>
```

```
<FORM ACTION="index.php">
<INPUT TYPE=SUBMIT VALUE="Zpět">
</FORM>

</BODY>
</HTML>
```

Na konci skriptu jsme použili osvědčenou fintu pro vytváření tlačítek. Tlačítko Zpět jsme vytvořili jako jednoduchý formulář obsahující pouze tlačítko pro odeslání formuláře.

Po vyplnění údajů může uživatel formulář odeslat. Vyplněné údaje jsou předány skriptu `insert.php`, který se postará o vložení údajů do tabulky.

Příklad: `insert.php`

```
<?
$status = true;            // indikátor úspěšnosti přidání záznamu
do {
    if (!IsSet($OsobniCislo)):  // známe osobní číslo?
        $status = false;
        break;
    endif;
    $Plat += 0;                 // pokud nebyl plat zadán, bude z něj nula
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):             // podařilo se připojit k databázi
        $status = false;
        break;
    endif;
    @$vysledek = ODBC_Exec($spojeni,    // vložení nového záznamu
        "INSERT INTO Zamestnanci
         VALUES ($OsobniCislo, '$Jmeno', '$RC', '$Adresa', $Plat)");
    if (!$vysledek):            // podařilo se záznam vložit?
        $status = false;
        break;
    endif;
} while (false);
$type = "insert";               // typ operace
                                // přesměrování na "bezpečnou" stránku
$path = SubStr($SCRIPT_NAME, 0, StrRPos($SCRIPT_NAME, "/")).
    "/info.php?status=$status&type=$type";
Header("Location: http://$SERVER_NAME:$SERVER_PORT$path");
?>
```

Proměnná `$status` obsahuje `true`, pokud se záznam podařilo vložit, v opačném případě obsahuje `false`. Pokud nám z formuláře nedorazilo osobní číslo, ukončíme celou operaci, protože pro vkládání nového záznamu musíme znát alespoň primární klíč.

Příkaz `$Plat += 0` má pouze jeden účel. Ať už uživatel do formuláře jako plat napsal cokoliv, je to převedeno na číslo. Navíc, pokud uživatel zadal opravdu číslo, jeho hodnota se nezmění, protože přičítáme nulu.

Na dalších řádcích skriptu se připojíme k databázi a zapíšeme do ní nový záznam pomocí SQL-příkazu `INSERT`.

Zajímavé jsou poslední dva řádky. Ty způsobí to, že jako výsledek skriptu `insert.php` se do prohlížeče odešle pouze instrukce k načtení jiné stránky. K přesměrování prohlížeče slouží právě HTTP hlavička `Location`. Přesměrování provádíme na skript `info.php` a nastavujeme přitom parametry `type` a `status`, které určují typ prováděné operace (v našem případě `insert` pro vložení) a její úspěšnost.

Jako obsah hlavičky `Location` musí být uvedeno absolutní URL, proto si zjišťujeme adresu aktuálního skriptu, pomocí funkce `SubStr()` z ní odstraníme jméno skriptu a připojíme jméno skriptu `info.php` společně s parametry.

Skript `info.php` pouze vypíše hlášení o úspěšnosti či neúspěšnosti prováděné operace. Oprávněně vás teď napadá, proč to děláme tak složitě. Důvod je prostý — je jím tlačítko Back, které nalezneme v každém prohlížeči. Kdyby skript `insert.php` rovnou generoval HTML-stránku popisující úspěšnost operace, mohli bychom se na ní v budoucnosti právě pomocí tlačítka Back nebo historie vrátit. Jelikož skripty nejsou ukládány do cache, vyvolal by tento návrat nové odeslání parametrů formuláře serveru. Skript by se pokusil znovu přidat záznam a pravděpodobně by neuspěl — vypsal by tedy chybové hlášení. Výsledkem by byl totálně zmatený uživatel. Některé prohlížeče se sice před znovu odesláním dat ptají uživatele, zda si přeje Repost form data?, ale který z uživatelů tomuto hlášení rozumí.

Tím, že skript `insert.php` nevrací HTML-stránku, ale pouze příkaz k přesměrování, nebude přidán do historie navštívených stránek. Tím zabrání uživateli omylem provést operaci vložení znovu. V historii navštívených dokumentů tedy bude stránka `pridej.php` a za ní hned `info.php`. Stránku `insert.php` budeme hledat marně.

Výše uvedený přístup k operacím, které mění data v tabulkách, budeme používat i pro další operace. Je to jedna z mála cest, jak zabránit zmatení uživatele, který se rád pohybuje v historii dokumentů.

Další operací, kterou naše aplikace nabízí, je možnost mazání záznamů. Pokud chceme nějaký záznam smazat, je volán skript `smazat.php` a pomocí parametru `oc` je mu předáno osobní číslo zaměstnance, kterého chceme smazat.

Příklad: `smazat.php`

```
<?
  Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT") // neukládat do cache
?>
<HTML>
<HEAD>
<TITLE>Potvrzení výmazu</TITLE>
</HEAD>
<BODY>
<H1>Potvrzení výmazu</H1>

<?
do {
  if (!IsSet($oc)):              // známe osobní číslo?
    echo "Nebyl určen zaměstnanec k výmazu.\n";
    break;
  endif;
  @$spojeni = ODBC_Connect("test", "", "");
  if (!$spojeni):                // podařilo se připojit k databázi
    echo "Nepodařilo se připojit k databázi.\n";
    break;
  endif;

  // ověření, zda se na zaměstnance neodkazujeme z tabulky "Odberatele"
  @$vysledek = ODBC_Exec($spojeni,
    "SELECT ICO FROM Odberatele WHERE Zastupce = $oc");
  if (!$vysledek):
    echo "Chyba při kontrole závislosti dat.\n";
    break;
  endif;
  if (ODBC_Fetch_Row($vysledek)): // má zaměstnanec na starosti odběratele?
    echo "Zaměstnanec nemůže být smazán,
      protože zodpovídá za některé odběratele.\n";
    break;
  endif;

  // vypsání celého záznamu, který chceme smazat
  @$vysledek = ODBC_Exec($spojeni,   // přečtení údajů z tabulky
```

```
        "SELECT * FROM Zamestnanci WHERE OsobniCislo = $oc");
    if (!$vysledek):
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;
    if (!ODBC_Fetch_Row($vysledek)):    // existuje daný zaměstnanec
        echo "Požadovaný záznam neexistuje.\n";
        break;
    endif;
    echo "Chcete opravdu smazat následující záznam o zaměstnanci číslo ".
        $oc.":<BR>\n";
    echo ODBC_Result($vysledek, "Jmeno")."<BR>\n";
    echo ODBC_Result($vysledek, "RC")."<BR>\n";
    echo ODBC_Result($vysledek, "Adresa")."<BR>\n";
    echo ODBC_Result($vysledek, "Plat")."<BR>\n";
?>

<FORM ACTION=delete.php>
<INPUT TYPE=SUBMIT VALUE="Ano">
<INPUT TYPE=HIDDEN NAME=oc VALUE=<?echo $oc?>>
</FORM>

<?
} while (false);
?>

<FORM ACTION=index.php>
<INPUT TYPE=SUBMIT VALUE="Zpět">
</FORM>

</BODY>
</HTML>
```

Skript nejprve kontroluje, zda zaměstnanec, kterého chceme smazat, nemá na starost některého z odběratelů. Pokud bychom totiž záznam o zaměstnanci bez skrupulí smazali, mohlo by se stát, že budeme mít u některých odběratelů odkaz na neexistující zaměstnance.

Pokud naše skripty manipulují s tabulkami, mezi kterými existují vztahy, musíme vždy velice pečlivě dbát na zachování integrity dat. Kontrolu integrity dat je u lepších SQL-serverů možno definovat přímo při vytváření tabulek pomocí klauzulí `FOREIGN KEY` a `REFERENCES`. Podobného výsledku lze dosáhnout pomocí vhodně nastavených triggerů. Podrobnosti naleznete v dokumentaci k vašemu serveru.

Pokud je zaměstnance možno smazat, zobrazí skript jeho osobní údaje a zeptá se, zda má záznam opravdu smazat. Pokud uživatel stiskne tlačítko Ano, vyvolá skript `delete.php`, který zajišťuje vlastní smazání. Číslo zaměstnance ke smazání je určeno pomocí parametru `oc`.

Příklad: `delete.php`

```
<?
$status = true;             // indikátor úspěšnosti smazání záznamu
do {
    if (!IsSet($oc)):       // známe osobní číslo?
        $status = false;
        break;
    endif;
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):             // podařilo se připojit k databázi
        $status = false;
        break;
    endif;
    @$vysledek = ODBC_Exec($spojeni,            // smazání zázmamu
        "DELETE FROM Zamestnanci WHERE OsobniCislo = $oc");
    if (!$vysledek):            // podařilo se záznam smazat?
        $status = false;
        break;
    endif;
} while (false);
$type = "delete";           // typ operace
                            // přesměrování na "bezpečnou" stránku
$path = SubStr($SCRIPT_NAME, 0, StrRPos($SCRIPT_NAME, "/")).
    "/info.php?status=$status&type=$type";
Header("Location: http://$SERVER_NAME:$SERVER_PORT$path");
?>
```

Celý skript je velice podobný skriptu `insert.php`. Skript zkontroluje, zda mu dorazilo číslo zaměstnance, a pomocí SQL-příkazu `DELETE` záznam smaže. Nakonec přesměruje prohlížeč na skript `info.php` a jako parametry mu předá jméno prováděné operace a její úspěšnost.

Mezi další běžně používané operace patří změna některých položek záznamu. Tuto funkci zajišťuje skript `upravit.php`. Jako parametr očekává osobní číslo zaměstnance, jehož záznam chceme měnit. Do formuláře zobrazí všechny údaje o zaměstnanci a umožní jejich editaci.

Příklad: `upravit.php`

```
<?
    Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT") // neukládat do cache
?>
<HTML>
<HEAD>
<TITLE>Úprava údajů o zaměstnanci</TITLE>
</HEAD>
<BODY>
<?
do {
    if (!IsSet($oc)):               // známe osobní číslo zaměstnance?
        echo "Nebyl zadán zaměstnanec, který se má modifikovat.\n";
        break;
    endif;
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):                 // podařilo se připojit k databázi
        echo "Nepodařilo se připojit k databázi.\n";
        break;
    endif;

    // načtení záznamu pro úpravy
    @$vysledek = ODBC_Exec($spojeni, "SELECT * FROM Zamestnanci
                                      WHERE OsobniCislo = '$oc'");
    if (!$vysledek):
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;
    if (ODBC_Fetch_Row($vysledek)):            // čtení položek záznamu
        $OC = ODBC_Result($vysledek, "OsobniCislo");
        $Jmeno = ODBC_Result($vysledek, "Jmeno");
        $RC = ODBC_Result($vysledek, "RC");
        $Adresa = ODBC_Result($vysledek, "Adresa");
```

```
        $Plat = ODBC_Result($vysledek, "Plat");
    else:
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;
?>

<H1>Úprava údajů o zaměstnanci</H1>

<!-- vypsání položek záznamu do formuláře pro úpravy -->
<FORM ACTION="update.php" METHOD=POST>
<TABLE>
<TR><TD>Osobní číslo:<TD><INPUT NAME=OsobniCislo VALUE="<?echo $OC?>" SIZE=4>
<TR><TD>Jméno:<TD><INPUT NAME=Jmeno VALUE="<?echo $Jmeno?>">
<TR><TD>Rodné číslo:<TD><INPUT NAME=RC VALUE="<?echo $RC?>"SIZE=11>
<TR><TD>Adresa:<TD><INPUT NAME=Adresa VALUE="<?echo $Adresa?>"SIZE=60>
<TR><TD>Plat:<TD><INPUT NAME=Plat VALUE="<?echo $Plat?>"SIZE=5>
</TABLE>
<P><INPUT TYPE=SUBMIT VALUE="Zapiš změny">
</FORM>

<?
} while (false);
?>

<FORM ACTION="index.php">
<INPUT TYPE=SUBMIT VALUE="Zpět">
</FORM>

</BODY>
</HTML>
```

Po provedení změn je obsah formuláře odeslán skriptu `update.php`, který provede samotnou aktualizaci údajů v tabulce. O úspěšnost či neúspěšnost prováděných změn informuje opět pomocí skriptu `info.php`, na který je prováděno přesměrování.

Příklad: `update.php`

```
<?
$status = true;            // indikátor úspěšnosti změny záznamu
do {
    if (!IsSet($OsobniCislo)):  // známe osobní číslo?
        $status = false;
        break;
    endif;
    $Plat += 0;                 // pokud nebyl plat zadán, bude z něj nula
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):             // podařilo se připojit k databázi
        $status = false;
        break;
    endif;
    @$vysledek = ODBC_Exec($spojeni,            // změna záznamu
        "UPDATE Zamestnanci SET
            OsobniCislo = $OsobniCislo,
            Jmeno = '$Jmeno',
            RC = '$RC',
            Adresa = '$Adresa',
            Plat = $Plat WHERE OsobniCislo = $OsobniCislo");
    if (!$vysledek):            // podařilo se záznam změnit
        $status = false;
        break;
    endif;
} while (false);
$type = "update";               // typ operace
                                // přesměrování na "bezpečnou" stránku
$path = SubStr($SCRIPT_NAME, 0, StrRPos($SCRIPT_NAME, "/")).
    "/info.php?status=$status&type=$type";
Header("Location: http://$SERVER_NAME:$SERVER_PORT$path");
?>
```

Poznamenejme, že skript `update.php` by měl mnohem lépe ošetřovat změnu osobního čísla. Například by měl osobní číslo změnit i u všech odpovídajících záznamů v ostatních svázaných tabulkách, jako je např. tabulka `Odberatele`.

Nyní se konečně dostáváme ke skriptu `info.php`, který informuje uživatele o úspěšnosti výše prováděných operací. Skript v závislosti na parametrech `type` a `status` vypíše jedno z šesti hlášení. Parametr `type` určuje typ prováděné operace — vkládání, mazání a modifikace. Parametr `status` určuje, zda byla prováděná operace úspěšná.

Příklad: `info.php`

```
<HTML>
<HEAD>
<TITLE>Výsledek operace</TITLE>
</HEAD>
<BODY>
<?$msg = Array("insert"=>Array("přidat", "přidán"),     // tabulka zpráv
               "delete"=>Array("smazat", "smazán"),
               "update"=>Array("modifikovat", "změněn")
        )?>
<?if ($status):?>
    <H1>Záznam byl úspěšně <?echo $msg[$type][1]?></H1>
<?else:?>
    <H1>Záznam se nepodařilo <?echo $msg[$type][0]?></H1>
<?endif?>
<A HREF="index.php">Prohlížení všech zaměstnanců</A>
</BODY>
</HTML>
```

Poslední skript `odber.php`, který je určen pro výpis všech odběratelů, kteří spadají pod určitého zaměstnance, je opravdu jednoduchý. Jako parametr mu předáme osobní číslo zaměstnance a skript vypíše pouze odpovídající záznamy z tabulky `Odberatele`.

Příklad: `odber.php`

```
<?
    Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT") // neukládej do cache
?>
<HTML>
<HEAD>
<TITLE>Odběratelé, jež má na starost zaměstnanec <?echo $oc?></TITLE>
</HEAD>
<BODY>
<?
do {
    if (!IsSet($oc)):        // potřebujeme znát číslo zaměstnance
        echo "Nebyl určen zaměstnanec, o jehož odběratele máte zájem.\n";
        break;
    endif;
    @$spojeni = ODBC_Connect("test", "", "");
    if (!$spojeni):          // podařilo se připojit k databázi
        echo "Nepodařilo se připojit k databázi.\n";
```

```
        break;
    endif;

    // zjištění jména zaměstnance podle osobního čísla
    @$vysledek = ODBC_Exec($spojeni,
        "SELECT Jmeno FROM Zamestnanci WHERE OsobniCislo = $oc");
    if (!$vysledek):
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;
    if (ODBC_Fetch_Row($vysledek)):
        $Jmeno = ODBC_Result($vysledek, "Jmeno");
    else:
        echo "Chyba při prohledávání seznamu zaměstnanců.\n";
        break;
    endif;

    // vypsání seznamu odběratelů
    echo "<H1>Seznam odběratelů,</H1>\nkteré má na starost $Jmeno\n<P>";
    @$vysledek = ODBC_Exec($spojeni,        // dotaz do tabulky
        "SELECT ICO, Nazev, Sidlo FROM Odberatele
         WHERE Zastupce = $oc ORDER BY Nazev");
    if (!$vysledek):
        echo "Chyba při prohledávání seznamu odběratelů.\n";
        break;
    endif;

    // hlavička tabulky
    echo "<TABLE CELLSPACING=0 BORDER=1 CELLPADDING=2>\n";
    echo "<TR BGCOLOR=TEAL><TH>Název</TH><TH>IČO</TH><TH>Sídlo</TH></TR>\n";
    // vypsání všech odběratelů pro daného zaměstnance
    while (ODBC_Fetch_Row($vysledek)):
        echo "<TR>";
        echo "<TD>".ODBC_Result($vysledek, "Nazev")."</TD>";
        echo "<TD>".ODBC_Result($vysledek, "ICO")."</TD>";
        echo "<TD>".ODBC_Result($vysledek, "Sidlo")."</TD>";
        echo "</TR>\n";
    endwhile;
    echo "</TABLE>\n";          // konec tabulky
} while (false);
?>
```

```
<FORM ACTION="index.php">
<INPUT TYPE=SUBMIT VALUE="Zpět">
</FORM>

</BODY>
</HTML>
```

Vidíme, že práce s databázemi je v PHP velice snadná. Na druhou stranu může zpřístupnění velké databáze, která obsahuje více provázaných tabulek, obnášet napsání několika desítek velice podobných skriptů. V těchto případech stojí za úvahu vytvoření jakési speciální metadatabáze, která bude obsahovat popis tabulek a způsobu jejich zpřístupnění. Pomocí skriptu pak z této metadatabáze budeme generovat jednotlivé formuláře pro práci s tabulkami. Tento přístup se z dlouhodobějšího hlediska určitě vyplatí, zvláště pro větší aplikace. Pokud se nechcete do tvorby takového systému pouštět, můžete využít již hotové systémy. V době psaní knihy však byly všechny takové systémy, o kterých vím, dostupné pouze komerčně.

6.7 Transakční zpracování

V praxi se často setkáme s případy, kdy je určitý proces reálného světa zachycen zároveň několika elementárními operacemi s daty v tabulkách. Například převod peněz v bance z jednoho účtu na druhý jsou dvě operace — u jednoho účtu o příslušnou částku snížíme zůstatek a u druhého naopak zvýšíme. Je zřejmé, že tyto dvě operace spolu úzce souvisí a jejich samostatné provedení nemá smysl.

Dobře navržený a implementovaný informační systém musí být odolný vůči různým nepříznivým vlivům. Mezi tyto vlivy patří například výpadek proudu, zaplnění diskové kapacity apod. Všechny tyto nechtěné stavy by mohly způsobit, že se peníze v našem příkladě z účtu pouze odečtou, ale druhá operace, zvýšení zůstatku, již neproběhne, protože počítač se vypne nebo nebude místo pro uložení dat na disku. Počítače si s tímto problémem musí nějak poradit, není přece možné, aby peníze nenávratně mizely někde v útrobách banky.

Poměrně elegantní řešení zmíněného problému přinášejí *transakce*. Transakce je posloupnost operací, které se buď provedou všechny, nebo se neprovede ani jedna z nich. Tím je zaručeno, že data před i po ukončení transakce budou konzistentní. Valná většina dnešních databázových serverů transakce samozřejmě podporuje. Pokud v průběhu provádění transakce dojde k chybě, nejsou data vůbec ovlivněna. Změny v datech se provedou až po úspěšném skončení transakce.

Jelikož je práce s transakcemi v každé databázi trošku jiná, ukážeme si jejich použití na rozhraní ODBC, které je široce podporováno.

Při použití ODBC jsou transakce normálně vypnuty — to znamená, že výsledky všech operací jsou ihned promítány do tabulek. Transakce je v tomto případě tvořena vždy jediným příkazem. Pokud chceme tento stav změnit, musíme použít funkci `ODBC_AutoCommit()`. Jejím prvním parametrem je číslo spojení s datovým zdrojem a druhým je příznak automatického potvrzování transakcí. Práci s transakcemi proto zahájíme příkazem:

```
if (!ODBC_AutoCommit($spojeni, false))
    echo "S transakcemi není něco v pořádku!"
```

Od této doby jsou všechny prováděné příkazy (volání `ODBC_Exec()`) považovány za součást transakce. Konec transakce oznámíme voláním funkce `ODBC_Commit()`:

```
if (!ODBC_Commit($spojeni))
    echo "Transakci se nepodařilo úspěšně ukončit!"
```

Pokud v průběhu transakce dojde k chybě a my chceme všechny kroky transakce zrušit, použijeme funkci `ODBC_RollBack()`, která databázi uvede do stavu před začátkem transakce.

Náš převod peněz z jednoho účtu na druhý můžeme provést následujícím skriptem:

```
$castka = 10000;        // částka pro převod mezi účty
$dluznik = 16689454;    // číslo účtu dlužníka
$veritel = 56455479;    // číslo účtu věřitele
$rollBack = false;      // indikátor chyby v průběhu zpracování transakce
$commitError = false;   // indikátor chyby při potvrzování transakce
$spojeni = ODBC_Connect("bu", "clearing", "EkoKP&#1y");
                        // připojení k databázi běžných účtů (bu)
ODBC_AutoCommit($spojeni, false);   // zahájení transakce

if (!ODBC_Exec($spojeni, "UPDATE BezneUcty SET Zustatek = Zustatek - $castka
                          WHERE Ucet = $dluznik"))
    $rollBack = true;

if (!$roolBack && !ODBC_Exec($spojeni, "UPDATE BezneUcty
                                        SET Zustatek = Zustatek + $castka
                                        WHERE Ucet = $veritel"))
    $rollBack = true;

if ($rollBack)
    ODBC_RollBack($spojeni);    // došlo k chybě, rušíme transakci
else
```

```
    $commitError = !ODBC_Commit($spojeni);
                         // vše proběhlo v pořádku, potvrdíme transakci

if ($rollBack)
    echo "Nastala chyba při provádění některého z kroků transakce!";
elseif ($commitError)
    echo "Transakci se nepodařilo potvrdit!";
else
    echo "Převod peněz byl úspěšně ukončen.";
```

6.8 Práce s binárními daty

Původně byly relační databáze vyvinuty zejména pro uchovávání dobře strukturovaných a poměrně krátkých informací. Dnešní multimediální doba si však klade mnohem větší nároky, a tak můžeme mít v tabulkách položky, které slouží pro uložení obrázků, zvukových nahrávek či celých videoklipů.

Pro ukládání výše zmíněných velkých bloků binárních dat slouží datový typ BLOB (Binary Large OBject). Nutno podotknout, že předávání tohoto typu dat mezi klientem a serverem není vždy bez problémů — data jsou příliš dlouhá a navíc binární. Rozchození práce s BLOBy proto někdy zabere více času stráveného experimenty. Nakonec se však většinou vše úspěšně podaří. Podívejme se na to, jak se dá řešit vkládání binárních dat do tabulek a jejich čtení.

Vkládání dat

Pro vkládání nových záznamů slouží v SQL příkaz `INSERT`. Pro vkládání BLOBů mají některé servery speciální příkazy, ale většinou můžeme použít i příkaz `INSERT`. Binární data musíme stejně jako řetězce předávat uzavřené do apostrofů. Jelikož však binární data často obsahují znaky se speciálním významem — např. znak s kódem 0, který ukončuje řetězec — musíme tyto nebezpečné znaky převést na escape sekvence pomocí funkce `AddSlashes()`. Server by z této podoby měl sám data zkonvertovat zpět do binární podoby.

> Některé starší servery akceptují binární data, pouze pokud jsou zapsána jako sekvence dvoumístných šestnáctkových čísel, kde vždy dva znaky určují hodnotu jednoho bajtu. V PHP si můžeme napsat funkci, která převod provede, ale bude poměrně pomalá. Lepším řešením je tedy starý SQL-server vyhodit z okna, a pokud to nejde, tak si převodní funkci napsat v jazyce C a doplnit ji přímo do PHP.

Vše si ukážeme prakticky. Předpokládejme, že máme k dispozici tabulku `Obrazky`, která má dvě položky. První položka (`Jmeno`) obsahuje jméno obrázku a druhá (`Obrazek`) samotný obrázek. Následující skript přidá do tabulky nový obrázek se jménem `Mt Blanc`, který je uložen v souboru `mont-blanc.gif`.

```
$soubor = "./mont-blanc.gif";       // jméno souboru
$fp = FOpen($soubor, "r");          // otevření souboru s obrázkem
$data = AddSlashes(FRead($fp, FileSize($soubor)));
                                    // načtení obrázku do proměnné $data
FClose($fp);                        // zavření souboru
$spojeni = ODBC_Connect("test", "", "");    // připojení k databázi
$vysledek = ODBC_Exec($spojeni, "INSERT INTO Obrazky
                                 VALUES ('Mt Blanc', '$data')");
if (!$vysledek)
    echo "Tak se nám to nepovedlo!";
```

Nepříjemným omezením mnoha SQL-serverů je pevně daná velikost jejich vstupně/výstupních bufferů. Takové servery nejsou často schopné pracovat s bloky dat většími než 64 KB. Tím je omezena i velikost přenášených obrázků na necelých 64 KB. Na druhou stranu bychom na našich stránkách neměli vůbec tak velké obrázky používat, protože jejich stahování je poměrně zdlouhavé. Nedokonalost některých softwarových produktů nám ve výsledném efektu může pomoci.

Čtení dat

Naše aplikace by rozhodně neměly fungovat jako černé díry, do kterých můžeme do nekonečna ukládat data, ale žádná nezískáme zpět. Pro čtení binárních dat uložených v tabulce používáme stejné funkce jako pro čtení běžných dat. Obsah BLOB položky tedy získáme pomocí funkce `ODBC_Result()`. Chování funkce však můžeme ovlivnit pomocí funkcí `ODBC_BinMode()` a `ODBC_LongReadLen()`.

Standardně jsou nastaveny hodnoty `BinMode` na 1 a `LongReadLen` na 4 096. To znamená, že funkce `ODBC_Result()` vrátí binární data, ale maximálně o velikosti 4 096 bajtů. Většinou však máme v tabulce uložena binární data o mnohem větší velikosti. Musíme proto voláním `ODBC_LongReadLen()` zvětšit velikost čtených dat. Pokud budeme např. předpokládat, že v tabulce `Obrazky` není obrázek větší než 32 KB, použijeme volání `ODBC_LongReadLen(32768)`. Skript, který jako svůj obsah vrátí obrázek, může vypadat zhruba následovně:

```
<?
    Header("Content-type: image/gif");
    $spojeni = ODBC_Connect("test", "", "");
    $vysledek = ODBC_Exec($spojeni, "SELECT Obrazek FROM Obrazky
```

```
                                 WHERE Jmeno LIKE 'Mt Blanc'");
    if (ODBC_Fetch_Row($vysledek)):
        ODBC_LongReadLen($vysledek, 32768);
        echo ODBC_Result($vysledek, "Obrazek");
    endif;
?>
```

V reálných aplikacích by skript samozřejmě vybíral obrázek na základě nějakého parametru předaného jako parametr pomocí URL nebo odeslaného jako položka formuláře.

Pokud předem neznáme limit pro velikost čtených binárních dat, můžeme nastavit `BinMode` a `LongReadLen` na nulu. V tomto případě budou data získaná pomocí `ODBC_Result()` zapsána přímo na výstup skriptu. Režim se tedy hodí pouze pro aplikace, kdy lze obsah BLOB položky odeslat přímo jako odpověď prohlížeči. Pokud před odesláním potřebujeme získaná data ještě zpracovat, nemůžeme tuto metodu použít. Všimněte si, že upravený skript již neobsahuje příkaz `echo`:

```
<?
    Header("Content-type: image/gif");
    $spojeni = ODBC_Connect("test", "", "");
    $vysledek = ODBC_Exec($spojeni, "SELECT Obrazek FROM Obrazky
                                 WHERE Jmeno LIKE 'Mt Blanc'");
    if (ODBC_Fetch_Row($vysledek)):
        ODBC_BinMode($vysledek, 0);
        ODBC_LongReadLen($vysledek, 0);
        ODBC_Result($vysledek, "Obrazek");
    endif;
?>
```

V kombinaci s některými ODBC-ovladači nejsou při použití poslední metody čtení BLOBů vždy získána všechna data uložená v položce tabulky.

6.9 Persistentní spojení s databázovým serverem

Před začátkem práce s tabulkami se musí náš skript připojit k databázovému serveru pomocí funkce `Connect` (`ODBC_Connect()`, `MySQL_Connect()`, `Pg_Connect()`, ...). Pravdou ovšem je, že samotné vytvoření spojení s databází je často poměrně časově náročná operace. Spojení se musí znovu a znovu vytvářet pro každý spouštěný skript, který pracuje s databází. Na hodně zatížených serverech to ve výsledku může vést až k pozorovatelnému zpomalení odezvy.

Řešení výše zmíněného problému přinášejí tzv. *persistentní spojení*. Princip persistentních spojení spočívá v tom, že PHP trvale udržuje otevřené spojení s databází i po ukončení běhu skriptu. Pokud pak nějaký další skript pracuje se stejnou databází, použije se již existující spojení a nemusí se tedy zdlouhavě vytvářet nové spojení.

Používání persistentních spojení je velice jednoduché. V našich skriptech stačí nahradit slovo `Connect` slovem `PConnect` — dostaneme tedy `ODBC_PConnect()`, `MySQL_PConnect()` nebo `Pg_PConnect()` podle toho, jakou databázi používáme.

Persistentní spojení jsou funkční pouze v případě, kdy PHP běží jako modul serveru Apache, případně jako ISAPI, NSAPI nebo WSAPI modul na serverech ve Windows. Je to celkem pochopitelné, protože PHP musí v paměti udržovat informace o připojení k jednotlivým databázím. Pokud interpret PHP spouštíme jako CGI skript, je PHP pro každý skript spuštěno znovu a ihned po ukončení skriptu je ukončen běh samotného PHP. PHP v tomto případě nemá k dispozici žádnou stálou paměť, kde by uchovávalo informace o vytvořených spojeních.

Funkce pro vytváření persistentních spojení však můžeme používat, i když běží PHP jako CGI-skript. Místo persistentního spojení je vytvořeno obyčejné spojení. Výhodou tohoto řešení je, že po přechodu na verzi PHP pracující jako modul, nemusíme měnit naše skripty a zcela bez práce urychlíme jejich běh.

7. Praktické ukázky

Na následujících stránkách naleznete příklady jednoduchých aplikací, které slouží především jako ukázky tvorby aplikací v PHP. Většinu z nich můžete s minimálními úpravami používat na vašich stránkách (např. počítadla přístupů). Další složitejší příklady vás seznámí s použitím různých knihoven a funkcí, které PHP nabízí. Všechny aplikace jsou podrobně komentovány a všechny neobvyklé programátorské obraty jsou vysvětleny. Doufám, že kapitola bude inspirací pro vaše vlastní aplikace. Zjistíte, že pomocí PHP můžete realizovat i ty nejnáročnější nápady.

7.1 Počítadla přístupů

Asi nejpoužívanější aplikací skriptů, se kterou se setkáte na webových stránkách, jsou počítadla přístupů. Na stránce je zobrazeno číslo, které udává, kolikrát již byla stránka webovým serverem poskytnuta uživatelům. Toto číslo je zajímavé jak pro návštěvníky stránky — stránka má hodně přístupů, prohlížím si tedy dobrou stránku — tak pro autory stránky — mé stránky jsou opravdu dobré, když si je prohlíží tolik lidí. Jak taková počítadla fungují? Existují dva základní přístupy:

- *Počítadlo jako součást stránky* — v tomto případě je celá stránka, která obsahuje počítadlo, skriptem. V místě, kde chceme mít na stránce počítadlo, vložíme pár příkazů PHP, které se postarají o vygenerování správného počtu přístupů.

- *Počítadlo jako vložený obrázek* — tento přístup slouží ke vložení počítadla do libovolné stránky. Na místo počítadla vložíme do stránky obrázek. URL obrázku však směřuje ke skriptu, který automaticky vygeneruje aktuální počet přístupů v podobě obrázku.

Otázkou zůstává, který z přístupů je lepší. Odpověď však není zcela jednoznačná. Pokud zařadíme počítadlo přímo do stránky, bude viditelné ve všech prohlížečích — i v textových nebo v těch, které mají vypnuto stahování obrázků. Nevýhodou těchto stránek je, že se nemohou uchovávat ve vyrovnávací paměti prohlížeče ani proxy-serverů. Aby bylo počítadlo přístupů stále aktuální, musí být každý požadavek na stránku vyřízen znovu serverem. Pokud je tedy počítadlo jediný dynamicky generovaný prvek stránky, je lepší nezbavovat se výhody ukládání stránky do vyrovnávacích pamětí a použít druhou metodu — vložení počítadla jako obrázku.

Pokud vkládáme počítadlo jako obrázek, zůstává naše stránka zcela statická — bude se uchovávat ve vyrovnávacích pamětech a její opakované čtení

nebude zbytečně zatěžovat síť. Opakovaně se bude přenášet pouze malý obrázek, zobrazující počet přístupů. Na první pohled se v tomto případě pro každý požadavek na stránku přenáší více dat — obrázek jsou v podstatě data navíc. Díky tomu, že se v mnoha případech uplatní proxy nebo cache prohlížeče, kde je uložena stránka, je výsledné zatížení přenosových kapacit Internetu menší. Toto řešení má však i své zápory — počet přístupů je většinou podhodnocen, protože nejsou započítány přístupy prohlížečů, které nezobrazují obrázky. Jak za chvíli uvidíme, dalším problémem je, že u špatně navrženého obrázkového počítadla, nám na našich stránkách může kdokoliv neoprávněně zvedat počet přístupů.

Základní princip, na kterém počítadla pracují, zůstává stejný. Pro každou stránku musíme mít někde uloženo, kolikrát na ni již bylo přistoupeno. Tento údaj můžeme uložit například do textového souboru nebo do databázové tabulky. Pokaždé, když bude potřeba počítadlo aktualizovat, zjistíme poslední počet přístupů, zvětšíme ho o jedna a přepíšeme jím starou hodnotu. Podívejme se nyní na několik možných implementací počítadel přístupu.

Textové počítadlo vložené přímo do stránky

Pro praktické použití bude nejlepší, když bude veškerý kód, který generuje počítadlo, uložen v jednom souboru, který podle potřeby načteme do stránky pomocí příkazu `require`.

Počet přístupů budeme pro každou stránku uchovávat ve zvláštním souboru. Všechny datové soubory soustředíme do jednoho adresáře, abychom si ve všem zachovali pořádek. Jméno adresáře si pro další zpracování uložíme do proměnné `$datadir`:

```
$datadir = "d:\\work\\counter\\";
```

V našem případě se budou soubory ukládat do adresáře `d:\work\counter`. Pokud pracujeme na Unixu, zvolíme např. adresář `/var/counter`. Do tohoto adresáře bude PHP zapisovat datové soubory. Proto je důležité, aby do tohoto adresáře mělo PHP spouštěné WWW-serverem práva zápisu. Na unixových serverech je WWW-server spouštěn nejčastěji jako uživatel `nobody` nebo `httpd` — tomuto uživateli musíme do adresáře povolit zápis. Pokud používáme Windows NT a IIS, jmenuje se tento uživatel `IUSR_XXX`, kde `XXX` je jméno počítače, na kterém spouštíme IIS.

Jméno aktuálně zpracovávané stránky máme v PHP přístupné v proměnné `$SCRIPT_NAME`. Z obsahu této proměnné musíme nějakým způsobem vygenerovat jméno datového souboru, aby měla každá stránka s počítadlem svůj datový soubor. Neupravený obsah proměnné `$SCRIPT_NAME` však použít nemůžeme, protože obsahuje jméno stránky včetně cesty. Ta obsahuje lomítka, která by nám ve jméně souboru působila problémy. Není však žádný problém nahradit lomítka

nějakým neutrálním znakem — např. podtržítkem '_'. Kromě lomítek nahradíme neutrálním jménem ještě tečku, aby nám nevznikaly soubory obsahující ve jméně více teček. K získání „normalizovaného“ jména stránky s výhodou použijeme funkci `StrTr()`, která umí provádět transformaci znaků. Jméno datového souboru tedy získáme příkazem:

```
$datafile = StrTR($SCRIPT_NAME, "./\\", "___");
```

Kompletní jméno datového souboru nyní složíme ze jména adresáře pro datové soubory, ze jména souboru a z přípony `dat`:

```
$filename = $datadir.$datafile.".dat";
```

Nyní, když máme jméno datového souboru, není nic snazšího, než z něj přečíst počet přístupů, zvětšit jej o jedna a nový stav zapsat zpět. Tato část skriptu však musí speciálně ošetřit případ, kdy je ze stránky počítadlo voláno poprvé a pro stránku dosud neexistuje datový soubor. V tomto případě nemůžeme z datového souboru číst, ale nejprve ho musíme vytvořit. V PHP máme naštěstí k dispozici funkci `File_Exists($filename)`, která vrací `true`, pokud soubor `$filename` existuje.

K otevření datového souboru použijeme funkci `FOpen($filename, `*«mód»*`)`. Jako *«mód»* použijeme `w` pro vytvoření souboru nebo `r+` pro současné čtení i zápis do souboru.

Pokud datový soubor existuje, přečteme z něj počet přístupů pomocí funkce `FGetS()`, zvětšíme ho o jedna a uložíme do proměnné `$hits`. Pomocí funkce `Rewind()` se nastavíme zpět na začátek datového souboru, kam budeme zapisovat aktualizovaný počet přístupů.

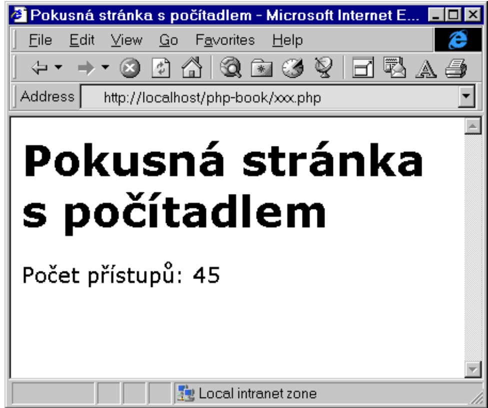
Pokud soubor vytváříme, znamená to, že ke stránce přistupujeme poprvé a do proměnné `$hits` uložíme jedničku. Nakonec pomocí `FPutS()` zapíšeme do souboru novou hodnotu počtu přístupů, soubor uzavřeme a vypíšeme proměnnou `$hits`, která obsahuje právě počet přístupů:

```
if (File_Exists($filename)):        // existuje datový soubor?
    $fp = FOpen($filename, "r+");   // otevření souboru
    $hits = FGetS($fp,10) + 1;      // přečtení počtu přístupů
    Rewind($fp);                    // příprava souboru na zápis
else:
    $fp = FOpen($filename, "w");    // vytvoření dat. souboru
    $hits = 1;                      // inicializace počtu přístupů
endif;

FPutS($fp, $hits);                  // zapsání nového počtu přístupů
FClose($fp);                        // zavření datového souboru
echo $hits;                         // vypsání počtu přístupů
```

Skript pro počítání přístupů je hotov. Pro jeho praktické používaní ho umístíme do zvláštního souboru `counter1.php`. Tento soubor umístíme do libovolného adresáře, který nemusí být (a z bezpečnostních důvodů by ani neměl být) součástí stromu dokumentů webového serveru. Abychom si ušetřili prsty při psaní cesty ke skriptu, můžeme jméno adresáře, ve kterém je skript umístěn, přidat do direktivy `include_path` konfiguračního souboru. Na stránkách pak bude využívání počítadla velice jednoduché. Jen nesmíme zapomenout, že stránkám musíme změnit příponu z `html` na `php`, aby byly „proháněny" přes interpret jazyka PHP. Druhou možností je nakonfigurovat WWW-server tak, aby skripty hledal i v souborech s příponou `html`. Jednoduchá stránka s počítadlem může vypadat zhruba takto:

```
<HTML>
<HEAD>
<TITLE>Pokusná stránka s počítadlem</TITLE>
</HEAD>
<BODY>
<H1>Pokusná stránka s počítadlem</H1>
Počet přístupů: <?require "counter1.php"?>
</BODY>
</HTML>
```

Pro profesionální použití celý skript `counter1.php` ještě trošku upravíme. Předně z celého počítadla uděláme funkci — všechny používané proměnné se tím pádem stanou lokální a nebudou kolidovat s případnými dalšími proměnnými v ostatních skriptech. Do funkce navíc přidáme kontrolu, která ověří, zda se datový soubor podařilo otevřít či vytvořit.



Obr. 7-1: Stránka s naším prvním počítadlem

Příklad: `counter1.php`

```
<?
function GetHits()
{
    $datadir = "d:\\work\\counter\\";       // adresář pro datové soubory
    $datafile = StrTR($GLOBALS["SCRIPT_NAME"], "./\\", "___");
        // nahrazení nebezpečných znaků ze jména datového souboru
    $filename = $datadir.$datafile.".dat";
        // kompletní cesta k datovému souboru

    if (File_Exists($filename)):            // existuje datový soubor?
        $fp = FOpen($filename, "r+");       // otevření souboru
        if (!$fp) return;
        $hits = FGetS($fp,10) + 1;          // přečtení počtu přístupů
        Rewind($fp);                        // příprava souboru na zápis
    else:
        $fp = FOpen($filename, "w");        // vytvoření dat. souboru
        if (!$fp) return;
        $hits = 1;                          // inicializace počtu přístupů
    endif;

    FPutS($fp, $hits);                      // zapsání nového počtu přístupů
    FClose($fp);                            // zavření datového souboru
    echo $hits;                             // vypsání počtu přístupů
}
GetHits();
?>
```

Vidíme, že na první pohled počítadlo pracuje správně. Není to však tak docela pravda. Zkuste si v prohlížeči otevřít stránku s počítadlem, poté se pomocí tlačítka Back vraťte na předchozí stránku a pomocí Forward na stránku s počítadlem. Počítadlo přístupů se nezměnilo. To je ovšem chyba — vždyť jsme provedli další přístup ke stránce. Problém je způsoben tím, že stránku si prohlížeč podržel ve své vyrovnávací paměti. To se však dá poměrně snadno odstranit — stačí spolu se stránkou poslat HTTP hlavičku `Expires`, která udává časovou platnost stránky (viz strana 441).

Pokud nastavíme časovou platnost na aktuální okamžik, bude se stránka vždy znovu načítat ze serveru. Ke každé stránce s počítadlem stačí na začátek přidat jednu řádku, která odešle příslušnou hlavičku:

```
<?Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT")?>
<HTML>
```

```
<HEAD>
<TITLE>Pokusná stránka s počítadlem</TITLE>
</HEAD>
<BODY>
<H1>Pokusná stránka s počítadlem</H1>
Počet přístupů: <?require "counter1.php"?>
</BODY>
</HTML>
```

Počítadlo jako obrázek

Pokud budeme konstruovat počítadlo přístupů jako vkládaný obrázek, můžeme použít již hotový mechanismus zápisu a čtení počtu přístupů do datových souborů. Náš nový skript se bude lišit pouze ve dvou věcech — nemůže získat jméno stránky, pro které počítáme přístup, v proměnné `$SCRIPT_NAME` a jeho výstupem musí být obrázek ve formátu GIF.

První problém vyřešíme velice snadno — skriptu budeme předávat jméno stránky jako parametr, který bude součástí URL. Bude-li tedy naše počítadlo uloženo v souboru `counter2.php`, můžeme počítadlo vložit do stránky následovně:

```
<HTML>
<HEAD>
<TITLE>Pokusná stránka s grafickým počítadlem</TITLE>
</HEAD>
<BODY>
<H1>Pokusná stránka s grafickým počítadlem</H1>
Počet přístupů: <IMG SRC="counter2.php?href=stranka.html">
</BODY>
</HTML>
```

Vidíme, že se jedná o zcela běžnou HTML-stránku, která neobsahuje nic speciálního. Počítadlo je vloženo jako obrázek. Obrázek je však generován dynamicky skriptem `counter2.php`. V tomto skriptu budeme mít k dispozici proměnnou `$href`, která bude obsahovat jméno stránky, pro kterou počítáme přístupy. Tím máme vyřešen problém se získáním jména stránky, pro kterou vytváříme počítadlo.

Nyní zbývá vytvořit takový skript, jehož výstupem bude obrázek ve formátu GIF. V tomto okamžiku nám poslouží knihovna GD, která slouží k tvorbě obrázků právě ve formátu GIF. Knihovna může být zakompilována přímo jako součást PHP, nebo může být uložena v externí knihovně. Pokud je uložena v externí knihovně, musíme ji načíst pomocí funkce `Dl()` — ve Windows například pomocí:

Obr. 7-2: Grafické počítadlo přístupů

```
Dl("php3_gd.dll");
```

Nyní můžeme využívat mnoho grafických funkcí knihovny GD. Aby prohlížeč poznal, že náš skript vygeneroval obrázek, musí skript generovat HTTP hlavičku `Content-type` s typem `image/gif`, který odpovídá právě obrázku ve formátu GIF. Na začátku skriptu proto musíme použít:

```
Header("Content-type: image/gif");
```

Generování obrázků pomocí knihovny GD je velmi jednoduché. Nejprve musíme pomocí funkce `ImageCreate()` obrázek vytvořit. Funkce vrací číslo, pod kterým pak můžeme s obrázkem dále pracovat. Další funkce nám umožní do obrázku kreslit základní grafické objekty, jako čáry a kružnice, psát nápisy apod. Konečně pomocí funkce `ImageGif()` hotový obrázek zapíšeme na standardní výstup.

Podívejme se tedy postupně na jednotlivé kroky. Obrázek vytvoříme pomocí následujících příkazů:

```
$fontsize = 5;
$digits = Ceil(Log10($hits+1));
$img = ImageCreate(ImageFontWidth($fontsize)*$digits,
                   ImageFontHeight($fontsize));
```

Do proměnné `$fontsize` jsme si uložili velikost písma, které budeme používat pro počítadla. Následuje příkaz, který do proměnné `$digits` uloží počet cifer čísla vyjadřujícího počet přístupů (v našem případě je tento počet uložen v proměnné `$hits`).

Poslední příkaz vytvoří obrázek a uloží jeho identifikátor do proměnné `$img`. Parametry funkce `ImageCreate()` udávají šířku a výšku obrázku. Tu vypočítáme ze znalosti počtu cifer zobrazovaného čísla a velikosti písmen zjištěné pomocí funkcí `ImageFontWidth()` a `ImageFontHeigth()`.

Nyní, když máme obrázek vytvořen, nastavíme jeho barvu pozadí na bílou:

```
$backgroundcolor = ImageColorAllocate($img, 255, 255, 255);
```

Tuto barvu však ihned nastavíme jako transparentní, aby šlo počítadlo použít na stránce s libovolným pozadím:

```
ImageColorTransparent($img, $backgroundcolor);
```

Nyní si musíme alokovat barvu, kterou bude zobrazen text počítadla. My jsme zvolili červenou barvu (RGB složky jsou 255, 0, 0):

```
$textcolor = ImageColorAllocate($img, 255, 0, 0);
```

K vypsání řetězce do obrázku slouží funkce `ImageString()`. Jejími parametry jsou číslo obrázku, velikost písma, souřadnice textu, text a barva textu:

```
ImageString($img, $fontsize, 0, 0, $hits, $textcolor);
```

Nakonec zapíšeme obrázek na standardní výstup skriptu, aby byl odeslán prohlížeči:

```
ImageGif($img);
```

Za chvíli si ukážeme kompletní výpis skriptu. Skript jsme rozšířili ještě tak, aby mohl být volán s parametry `r`, `g` a `b`, které udávají barevné složky barvy, kterou bude zobrazen počet přístupů.

Pokud na naši stránku, která má URL `http://www.server.cz/cenik.html`, chceme umístit počítadlo v modré barvě, zařadíme do ní následující tag:

```
<IMG SRC="/counter2.php?href=http_//www.server.cz/cenik.html&r=0&g=0&b=255">
```

za předpokladu, že skript `counter2.php` je uložen v kořenovém adresáři našeho WWW-serveru. Ve jméně stránky, které předáváme skriptu jako parametr `href`, musíme nahradit pro URL nepřípustné znaky nějakým vhodným znakem — např. podtržítkem. Podívejme se na náš skript pro počítání přístupů v celé kráse:

Příklad: `counter2.php`

```
<?
// Vygeneruje počítadlo přístupů jako obrázek GIF
//
// Parametry předávané v URL:
//
//      href    adresa stránky, pro kterou se počítají přístupy
```

```
//      r       intenzita červené složky barvy počítadla
//      g       intenzita zelené složky barvy počítadla
//      b       intenzita modré složky barvy počítadla
//

dl("php3_gd.dll");          // načtení knihovny GD, pokud není přikompilována

Header("Content-type: image/gif");      // výstupem skriptu je obrázek
Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT");
                // výstup skriptu se nesmí uchovávat ve vyrovnávací paměti

$datadir = "d:\\work\\counter\\";       // adresář pro datové soubory
$datafile = StrTR($href, "./", "__");   // vygenerování jména datového souboru
$filename = $datadir.$datafile.".dat";  // celé jméno datového souboru

if (File_Exists($filename)):            // test existence datového souboru
    $fp = FOpen($filename, "r+");       // otevření datového souboru
    if (!$fp) return;                   // povedlo se soubor otevřít?
    $hits = FGetS($fp,20) + 1;          // přečtení počtu přístupů
    Rewind($fp);                        // příprava na zapsání nového počtu
else:
    $fp = FOpen($filename, "w");        // vytvoření datového souboru
    if (!$fp) return;                   // povedlo se
    $hits = 1;                          // inicializace počtu přístupů
endif;

FPutS($fp, $hits);                      // zapsání nového počtu přístupů
FClose($fp);                            // uzavření datového souboru

$fontsize = 5;                          // velikost písma
$digits = Ceil(Log10($hits+1));         // počet cifer v počtu přístupů
$img = ImageCreate(ImageFontWidth($fontsize)*$digits,
                   ImageFontHeight($fontsize));
                             // vytvoření obrázku o potřebné velikosti
$backgroundcolor = ImageColorAllocate($img, 255, 255, 255);
                             // alokace barvy pozadí
ImageColorTransparent($img, $backgroundcolor);
                             // barva pozadí je průhledná

if (IsSet($r) && IsSet($g) && IsSet($b))            // uživatelské barvy?
    $textcolor = ImageColorAllocate($img, $r, $g, $b);
else
```

```
    $textcolor = ImageColorAllocate($img, 255, 0, 0);// standardní červená barva

ImageString($img, $fontsize, 0, 0, $hits, $textcolor);// vypsání počtu přístupů
ImageGif($img);                              // zapsání obrázku na výstup

?>
```

Počítadla vkládaná jako obrázky mají jednu nepříjemnou vlastnost — kdokoliv z celého Internetu může do své stránky umístit odkaz na obrázek, který zaktivuje počítadlo přístupů pro naši stránku. Na naší stránce pak přístupy přibývají častěji, než by měly. Proti tomuto „čítačovému pirátství“ se může náš skript trochu bránit. Skript má totiž k dispozici obsah HTTP hlavičky `Referer`, která obsahuje jméno stránky, v níž byl obrázek umístěn. V našem skriptu tedy můžeme testovat, zda je právě prováděné volání skriptu vyvoláno z naší stránky. Stačí porovnat obsah proměnné `$href` s proměnnou `$HTTP_REFERER`. Nesmíme však zapomenout na to, že do proměnné `$href` dostáváme URL stránky lehce upravené, aby neobsahovalo některé znaky se speciálním významem v URL. Pokud se obsah proměnných neshoduje, je počítadlo voláno z jiné stránky, a tento přístup bychom neměli započítávat.

Objektové a databázové – zkrátka superpočítadlo

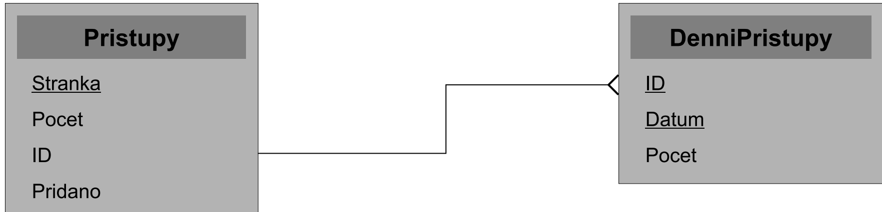
Předchozí dva způsoby implementace ukládaly informace o přístupu ke stránkám do textových souborů. Tyto informace však můžeme s výhodou ukládat do databázových tabulek. To nám přinese mnohé výhody — jednou z největších je patrně snadná analýza takto uložených dat. Pomocí vhodně formulovaných dotazů v jazyce SQL a grafického znázornění jejich výsledků, můžeme velice přehledně zobrazit počty přístupů k jednotlivým stránkám.

V následující ukázce proto vytvoříme počítadlo přístupů, které bude brát ohled na pozdější statistické zpracování dat o přístupech ke stránkám. U každé stránky budeme kromě samotných počtů přístupů evidovat i datum, od kdy jsou pro stránku počítány přístupy. Počty přístupů budeme navíc evidovat pro každý den zvlášť, abychom mohli provádět statistické analýzy.

Pro naše potřeby si proto vytvoříme novou databázi a pojmenujeme ji například `statistika`. Informace o přístupech budeme evidovat ve dvou tabulkách — `Pristupy` a `DenniPristupy` (viz obr. 7-3 na následující straně).

Tabulka `Pristupy` bude sloužit k uložení informací o přístupech pro jednu stránku. Nalezneme v ní proto atribut `Stranka` pro uložení jména souboru se stránkou, atribut `Pocet` obsahující počet přístupů a atribut `Pridano`, který obsahuje datum přidání záznamu o stránce do tabulky. Atribut `Stranka` bude sloužit jako primární klíč, protože cesta ke stránce je jedinečná.

Poslední atribut `ID` je generován uměle a slouží pro spojení s druhou tabulkou. Použijeme ho, protože námi vygenerované ID bude kratší než atribut



Obr. 7-3: ER-diagram vystihující vztahy mezi tabulkami počítadla

`Stranka`, což se projeví ve větší rychlosti práce s databází a v úspoře místa v tabulce `DenniPristupy`.

K čemu slouží tabulka `DenniPristupy`? V této tabulce jsou obsaženy počty přístupů pro jednotlivé stránky a dny. Stránka je identifikována právě pomocí `ID`. Den pak jako datum v atributu `Datum`. Tyto dva atributy společně splňují požadavky na primární klíč. Třetí atribut `Pocet` vyjadřuje počet přístupů k dané stránce v daný den.

Obě dvě tabulky můžeme vytvořit pomocí následujících příkazů SQL:

```
CREATE TABLE Pristupy (
  Stranka char(80) NOT NULL,
  Pocet int,
  ID char(14) NOT NULL,
  Pridano date,
  PRIMARY KEY (Stranka),
)

CREATE TABLE DenniPristupy (
  ID char(14) NOT NULL,
  Datum date NOT NULL,
  Pocet int,
  PRIMARY KEY (ID,Datum)
)
```

Příkazy zadáme tím způsobem, který vyhovuje našemu SQL-serveru. Většina serverů umožňuje spustit jednoduchý front-end program, který k zadávání podobných příkazů slouží. Pokud používáme MySQL, spustíme příkaz `mysql`, pokud používáme PostgreSQL, spustíme `psql` a v MS SQL-Serveru spustíme Enterprise Manager a v něm z menu vybereme příkaz Tools ▷ SQL Query Tool. Podrobnější informace naleznete v kapitole věnované databázím a v dokumentaci k vašemu serveru. Nezapomeňte před vytvářením tabulek vytvořit novou databázi `statistika`.

Pro práci s tabulkami si vytvoříme uživatele `stat`, který bude mít práva pro výběr, přidávání a změnu údajů v tabulkách. V našich příkladech budeme

předpokládat, že heslo tohoto uživatele je `x`. V příkladech budeme pro přístup k databázi používat rozhraní ODBC, musíme proto pro databázi `statistika` vytvořit odpovídající zdroj dat `statistika`.

Samotné počítadlo implementujeme jako třídu — proto slovo „objektové“ v názvu sekce. Konstruktor třídy se hned při vytváření instance počítadla postará o aktualizaci údajů v tabulkách a o zjištění potřebných údajů. Zjištěné informace o počtu přístupů, denním počtu přístupů a začátku počítání přístupů uloží konstruktor do členských proměnných. Pomocí dalších metod třídy zpřístupníme skriptu údaje z těchto proměnných.

Příklad: `counter3.php`

```
<?
class Pocitadlo
{
    var $_pocet = 0;
    var $_denniPocet = 0;
    var $_od = "1970-01-01";

    function Pocitadlo()    // konstruktor, který aktualizuje počty přístupů
    {
        if ($GLOBALS["COUNTING"]=="OFF") return;    // někdy přístup nepočítáme
        $dnes = Date("Y-m-d");
        $jmenoSkriptu = $GLOBALS["SCRIPT_NAME"];

        @$spojeni = ODBC_PConnect("statistika", "stat", "x");
        if (!$spojeni) return;

        @$vysledek = ODBC_Exec($spojeni, // zjistíme, zda je stránka v databázi
                        "SELECT Pocet, ID, Pridano FROM Pristupy
                         WHERE Stranka = '$jmenoSkriptu'");
        if (!ODBC_Fetch_Row($vysledek)):
            $this->_pocet = 1;         // stránku přidáme do databáze
            $ID = UniqID("");
            $this->_od = Date("Y-m-d");
            @ODBC_Exec($spojeni, "INSERT INTO Pristupy VALUES (
                                  '$jmenoSkriptu', 0,
                                  '$ID', '$dnes')");
        else:
            $this->_pocet = ODBC_Result($vysledek, "Pocet") + 1;
            $ID = ODBC_Result($vysledek, "ID");
            $this->_od = ODBC_Result($vysledek, "Pridano");
        endif;
```

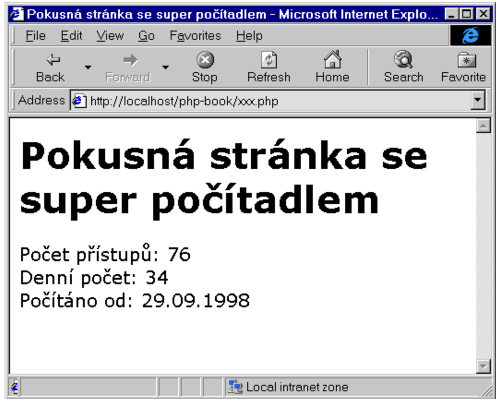
```
        @$vysledek = ODBC_Exec($spojeni, // zjistíme, zda je stránka v databázi
                             "SELECT Pocet FROM DenniPristupy
                              WHERE ID = '$ID' AND Datum = '$dnes'");
        if (!ODBC_Fetch_Row($vysledek)):
            $this->_denniPocet = 1;     // přidání stránky do denních přístupů
            @ODBC_Exec($spojeni, "INSERT INTO DenniPristupy VALUES (
                                   '$ID', '$dnes', 0)");
        else:
            $this->_denniPocet = ODBC_Result($vysledek, "Pocet") + 1;
        endif;

        // aktualizace počtu přístupů
        @ODBC_Exec($spojeni, "UPDATE Pristupy SET Pocet = Pocet + 1
                             WHERE Stranka = '$jmenoSkriptu'");
        @ODBC_Exec($spojeni, "UPDATE DenniPristupy SET Pocet = Pocet + 1
                             WHERE ID = '$ID' AND Datum = '$dnes'");
    }

    function PocetPristupu()        // metoda vracející celkový počet přístupů
    {
        return $this->_pocet;
    }

    function DenniPocet()           // metoda vracející denní počet přístupů
    {
        return $this->_denniPocet;
    }

    function PocitanoOd()       // metoda vracející datum zaregistrování stránky
    {
        return EReg_Replace("([0-9]{4})-([0-9]{2})-([0-9]{2})",
                           "\\3.\\2.\\1", $this->_od);
    }
}
?>
```

Podívejme se nyní podrobněji na funkci konstruktoru Pocitadlo(). Hned první příkaz nám může připadat podivný. Nedělá však nic jiného, než že počítadlo deaktivuje v případě, že je skriptu předán parametr COUNTING s hodnotou OFF. Tuto vlastnost oceníme dále při vyhodnocování počtu přístupů ke stránkám.

Následně zjišťujeme dosavadní počet přístupů, ID a datum počátku počítání přístupů pro aktuální stránku. Pokud se tento údaj nepodaří zjistit, znamená to, že stránka dosud není zaregistrována v našich tabulkách. Vygenerujeme proto pro stránku jedinečné ID a přidáme ji do tabulky Pristupy s nulovým počtem přístupů a aktuálním datem přidání. Následně do členských proměnných _pocet a _od uložíme celkový počet přístupů a datum, odkdy jsou pro stránku přístupy počítány.

Protože již nyní známe ID stránky, můžeme zjistit její denní počet přístupů. Pokud se ho nepodaří zjistit, přidáme pro stránku nový záznam do tabulky DenniPristupy. V tomto okamžiku také můžeme nastavit členskou proměnnou _denniPocet, která obsahuje denní počet přístupů ke stránce.

Nakonec pomocí SQL-příkazu UPDATE zvýšíme o jedna celkový i denní počet přístupů ke stránce.



Obr. 7-4: Použití superpočítadla na stránce

Následující skript demonstruje, jak naše nové počítadlo zařadit na stránku:

```
<HTML>
<HEAD>
<TITLE>Pokusná stránka se super počítadlem</TITLE>
</HEAD>
<BODY>
<H1>Pokusná stránka se super počítadlem</H1>
<?
require "./counter3.php";
```

```
$pocitadlo = new Pocitadlo();
echo "Počet přístupů: ".$pocitadlo->PocetPristupu()."<BR>";
echo "Denní počet: ".$pocitadlo->DenniPocet()."<BR>";
echo "Počítáno od: ".$pocitadlo->PocitanoOd()."<BR>";
?>
</BODY>
</HTML>
```

Nyní máme hotové počítadlo přístupů. Kromě toho však máme v tabulkách `Pristupy` a `DenniPristupy` uloženy informace o přístupu ke stránkám. Nic nebrání vytvoření dalších skriptů, které budou obsahy tabulek analyzovat a ukážou nám, které stránky jsou nejnavštěvovanější a jak se návštěvnost stránek mění v čase.

Možností, jak analýzu dat pojmout, je mnoho, a záleží jen na nás a naší fantazii, kterou si zvolíme. My si ukážeme vytvoření jednoduchého skriptu, který zobrazí počty přístupů k jednotlivým stránkám v daném dni. Pro zjištění potřebných údajů můžeme použít SQL-příkaz:

```
SELECT c.Stranka, d.Pocet
FROM Pristupy c, DenniPristupy d
WHERE (c.ID = d.ID) AND (d.Datum = '«datum»')
ORDER BY d.Pocet DESC
```

kde *«datum»* je datum dne, pro který nás údaje zajímají. Získané údaje pak zobrazíme ve formě tabulky kombinované se sloupcovým grafem (viz obr. 7-5 na následující straně).

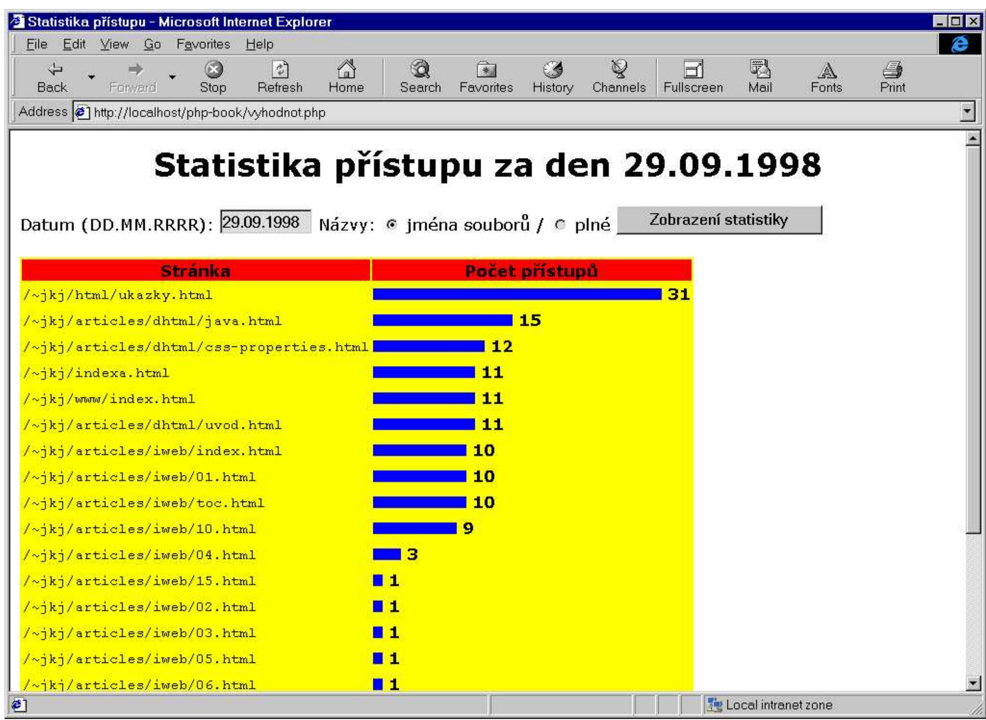
Pro běžného uživatele by bylo mnohem přehlednější, kdyby byl v grafu zobrazen místo jména souboru název stránky. Toho můžeme dosáhnout v zásadě dvěma způsoby — změnou skriptu pro počítání přístupů nebo dodatečným zjištěním názvu stránky.

První metoda předpokládá, že do tabulky `Pristupy` přidáme nový atribut pro název stránky. Skript pro počítadlo pak upravíme tak, aby do tabulky název stránky zapisoval.

Pokud však nechceme nic měnit, můžeme název stránky zjistit dodatečně. Jelikož známe jméno souboru se stránkou, stačí se podívat na začátek stránky a zjistit název uvedený mezi tagy `<TITLE>` a `</TITLE>`. Konečně se tedy dostáváme k samotnému skriptu, který generuje statistiku přístupů.

Příklad: `vyhodnot.php`

```
<HTML>
<HEAD>
<TITLE>Statistika přístupu</TITLE>
</HEAD>
<BODY>
```



Obr. 7-5: Statistika přístupu ke stránkám

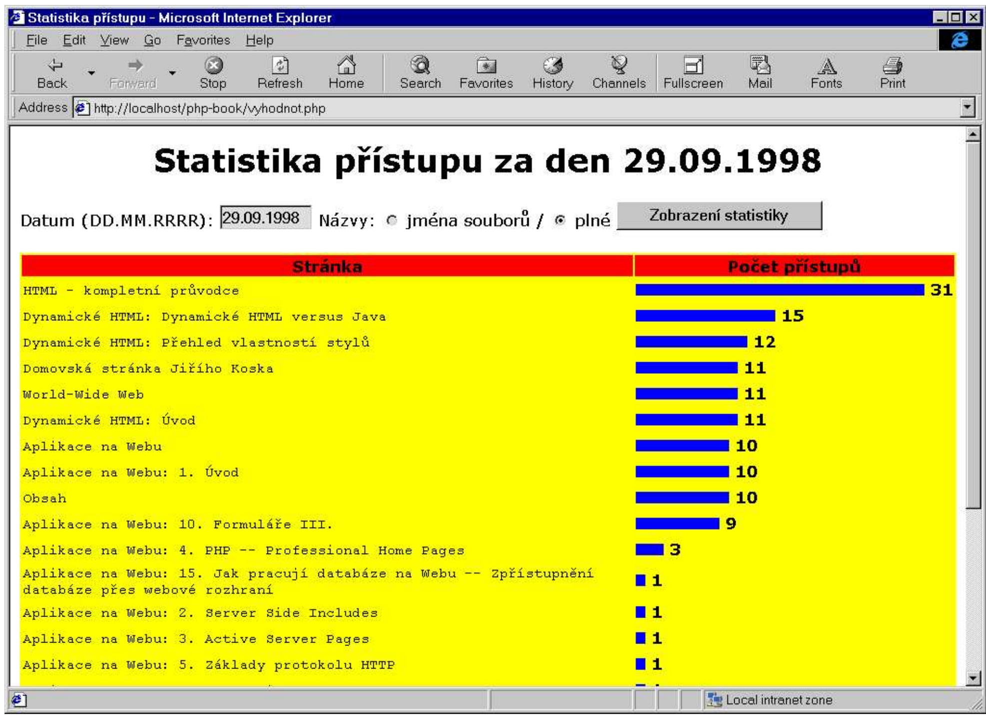
```
<?
// kontrola korektního formátu parametrů
if(!IsSet($Datum) || !EReg("[0-9]{2}.[0-9]{2}.[0-9]{4}", $Datum))
    $Datum = Date("d.m.Y");
if(!IsSet($NazvyStranek)) $NazvyStranek="Ne";
?>
<H1 ALIGN=CENTER>Statistika přístupu za den <?echo $Datum?></H1>

<FORM METHOD=POST>
Datum (DD.MM.RRRR): <INPUT NAME=Datum VALUE="<?echo $Datum?>" SIZE=10>
Názvy:
<INPUT TYPE=RADIO NAME=NazvyStranek
       VALUE="Ne"<?echo ($NazvyStranek=="Ne")?" CHECKED":""?>> jména souborů /
<INPUT TYPE=RADIO NAME=NazvyStranek
       VALUE="Ano"<?echo ($NazvyStranek=="Ano")?" CHECKED":""?>> plné
<INPUT TYPE=SUBMIT VALUE="Zobrazení statistiky">
</FORM>
```

```
<?
// funkce zjišťující název stránky pomocí HTTP požadavku
function GetTitle($path)
{
    if ($GLOBALS["NazvyStranek"]=="Ne") return $path;
    $fp = FOpen("http://localhost".$path."?COUNTING=OFF", "r");
    if (!$fp) return $path;
    ERegI("<TITLE>(.*)</TITLE>", FRead($fp, 256), $title);
    FClose($fp);
    return ($title[1]=="") ? $path : $title[1];
}

// zobrazení přístupů za den $Datum
$Date = EReg_Replace("([0-9]{2}).([0-9]{2}).([0-9]{4})",
                     "\\3-\\2-\\1", $Datum);// převod data na anglický formát
do {
    // připojení k datovému zdroji
    @$spojeni = ODBC_PConnect("statistika", "stat", "x");
    if (!$spojeni) break;

    // zjištění denního maxima pro určení měřítka
    @$vysledek = ODBC_Exec($spojeni, "SELECT Max(Pocet) FROM DenniPristupy
                                  WHERE Datum = '$Date'");
    if (ODBC_Fetch_Row($vysledek))
        $maxPocet = ODBC_Result($vysledek, 1);
    else
        $maxPocet = 1;

    // vykreslení sloupcového grafu
    @$vysledek = ODBC_Exec($spojeni, "
        SELECT c.Stranka, d.Pocet FROM Pristupy c, DenniPristupy d
        WHERE (c.ID = d.ID) AND (d.Datum = '$Date')
        ORDER BY d.Pocet DESC");
    if (!$vysledek) break;
    echo "<TABLE BGCOLOR=YELLOW>\n",
         "<TR><TH BGCOLOR=RED>Stránka<TH BGCOLOR=RED>Počet přístupů\n";
    while (ODBC_Fetch_Row($vysledek)):
        echo "<TR><TD>",
             "<CODE>".GetTitle(ODBC_Result($vysledek, "Stranka"))."</CODE>",
             "<TD><IMG SRC=\"bluedot.gif\" ALIGN=MIDDLE HEIGHT=12 WIDTH=".
                Round(ODBC_Result($vysledek, "Pocet") * 300 / $maxPocet).
                "> ", "<B>".ODBC_Result($vysledek, "Pocet")."</B>",
```

```
            "\n";
    endwhile;
    echo "</TABLE>\n";
} while(false);
?>
</BODY>
</HTML>
```

Ve funkci `GetTitle()`, která zjišťuje název dané stránky, otevíráme stránku pomocí protokolu HTTP. To by za normálních okolností způsobilo další započítání přístupu. Proto stránku vyvoláme s parametrem `COUNTING=OFF`. V tomto případě naše počítadlo přístup nezapočítává.

Řešení, kdy názvy stránek zjišťujeme dodatečně, je plně funkční (obr. 7-6). Jeho nevýhodou je, že se pro zjištění názvu každé stránky musí obsloužit jeden HTTP požadavek. Skript je proto velice časově náročný. Když takový skript zveřejníme, může několik uživatelů, kteří k němu přistupují zároveň, pěkně zatopit našemu serveru.



Obr. 7-6: Statistika přístupu ke stránkám s celými názvy stránek

7.2 Kniha hostů

Co by to bylo za knihu o tvorbě webových aplikací, kdyby v ní nebyl uveden návod na vytvoření knihy hostů. Jak kniha hostů vypadá? Jedná se o jednoduchou aplikaci, která umožní každému návštěvníkovi serveru zanechat zprávu. Všichni si pak takto zanechané zprávy mohou číst.

Skript, který bude realizovat knihu hostů, musí zvládnout tři činnosti. Jednak musí být schopen vypsat dosavadní zápisy uložené v knize hostů. Kromě toho však musí umožnit přidání nové zprávy. To sestává ze dvou operací — ze zobrazení formuláře pro zadání zprávy a z následného zpracování formuláře a uložení získaných hodnot do knihy hostů.

Začněme tedy s výpisem knihy hostů. Asi nejjednodušší řešení je ukládat nové zprávy do souboru již se všemi potřebnými HTML značkami, které se postarají o správné formátování. My budeme předpokládat, že zprávy jsou uloženy v souboru `kniha.body`. Kromě toho vytvoříme soubory `kniha.head` a `kniha.tail`, které budou obsahovat začátek a konec HTML dokumentu a po složení se souborem `kniha.body` dají dohromady validní stránku. Soubor `kniha.head` může obsahovat zhruba následující text:

```
<HTML>
<HEAD>
<TITLE>Kniha hostů</TITLE>
</HEAD>
<BODY>
<H1>Kniha hostů</H1>

<P>Vítejte v naší knize hostů. Můžete si zde přečíst názory na náš
server. Budeme rádi, když nám <A HREF="kniha.php">napíšete i váš
názor</A>.

<P><HR>
```

Odkaz `kniha.php` přitom nevede nikam jinam než ke skriptu, který slouží k přidání nové zprávy. Soubor `kniha.tail` obsahuje pouze ukončovací tagy, abychom dostali validní HTML dokument:

```
</BODY>
</HTML>
```

Postupné vypsání obsahu tří souborů na výstup skriptu je v PHP velice jednoduché. Stačí jednotlivé soubory otevřít pomocí funkce `FOpen()` pro čtení a pomocí funkce `FPassThru()` je celé poslat na výstup skriptu:

```
// vypsání standardní hlavičky
$fp = FOpen("./kniha.head", "r");
```

```
FPassThru($fp);

// vypsání zpráv
$fp = FOpen("./kniha.body", "r");
FPassThru($fp);

// vypsání záhlaví stránky
$fp = FOpen("./kniha.tail", "r");
FPassThru($fp);
```

Pro někoho podivná dvojice znaků './' před názvy souborů neříká nic jiného, než že se soubory budou hledat v aktuálním adresáři, tj. v adresáři, kde je uložen i samotný skript. Jednotlivé soubory nemusíme uzavírat ručně pomocí `FClose()`, protože je automaticky uzavře funkce `FPassThru()`.[1]

Výpis z knihy hostů máme hotov, podívejme se tedy na přidávání nových zpráv. Musíme samozřejmě vytvořit formulář, který bude sloužit pro zadání zprávy. Od uživatele vyzískáme jeho jméno (ve formulářovém poli `Jmeno`), e-mailovou adresu (v poli `Email`), adresu webové stránky (v poli `Web`) a zprávu (v poli `Zprava`). Nyní musíme navrhnout skript, který takto získané údaje vhodně zformátuje a přidá do souboru `kniha.body` k ostatním zprávám.

Nejprve tedy musíme otevřít soubor `kniha.body` pro přidávání údajů za jeho konec. Toho dosáhneme použitím `a` v druhém parametru funkce `FOpen()`:

```
$fp = FOpen("./kniha.body", "a");
```

Nyní s výhodou použijeme funkcí `FPutS()` a na konec souboru zapíšeme data získané z formuláře, doplněné o tagy, které zajistí úhledné zformátování knihy hostů v prohlížeči.

Nejprve tedy přidáme tučně zobrazené jméno doplněné o datum a čas vložení zprávy do knihy hostů:

```
FPutS($fp,"<B>$Jmeno</B> nám nechal(a) vzkaz ".Date("d.m.Y v H:i").":<BR>\n");
```

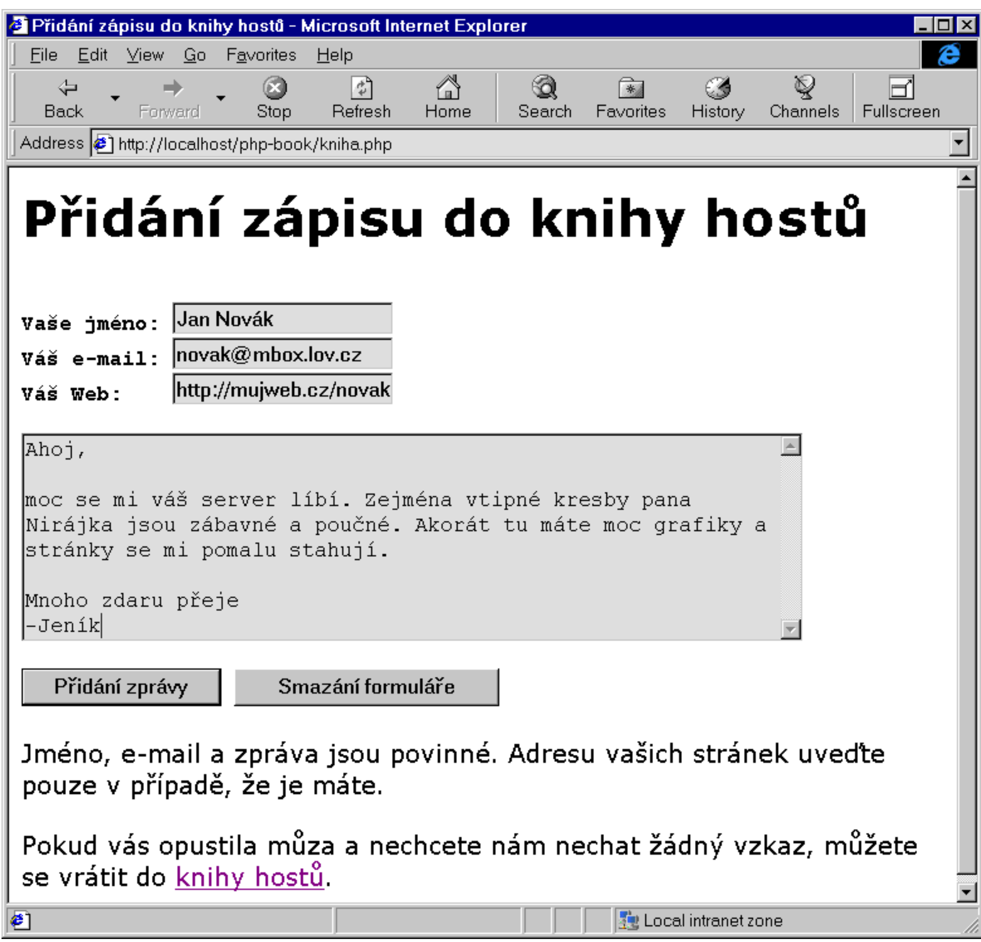
Následně vložíme e-mailovou adresu jako odkaz typu `mailto`, aby šlo pouhým kliknutím na odkaz aktivovat poštovního klienta:

```
FPutS($fp, "E-mail: <A HREF='mailto:$Email'>$Email</A><BR>\n");
```

Pokud uživatel zadal adresu svých stránek, vygenerujeme na ní odkaz:

```
if ($Web!="")
    FPutS($fp, "Web: <A HREF='$Web'>$Web</A><BR>\n");
```

[1] Celé to lze udělat ještě mnohem jednodušeji pomocí funkce `File()`. Kdo si ale má všechny ty funkce pamatovat?!



Obr. 7-7: Přidání nového záznamu do knihy hostů

Nakonec vložíme samotnou zprávu do elementu BLOCKQUOTE. Ve zprávě nahradíme všechny nebezpečné znaky, které by mohly porušit naše formátování pomocí funkce HTMLSpecialChars(). Kromě toho nahradíme v textu zprávy všechny konce řádků tagem <BR>, aby bylo zachováno formátování zprávy do řádek. Použijeme k tomu funkci NL2BR():

```
FPutS($fp, "<BLOCKQUOTE><I>\n");
FPutS($fp, NL2BR(HTMLSpecialChars($Zprava)));
FPutS($fp, "</I></BLOCKQUOTE>\n<P><HR>\n\n");
```

Za konec zprávy jsme navíc vložili tag <HR>, který používáme pro optické oddělování jednotlivých zpráv. Nakonec ještě musíme zavřít soubor a jsme hotovi:

```
FClose($fp);
```

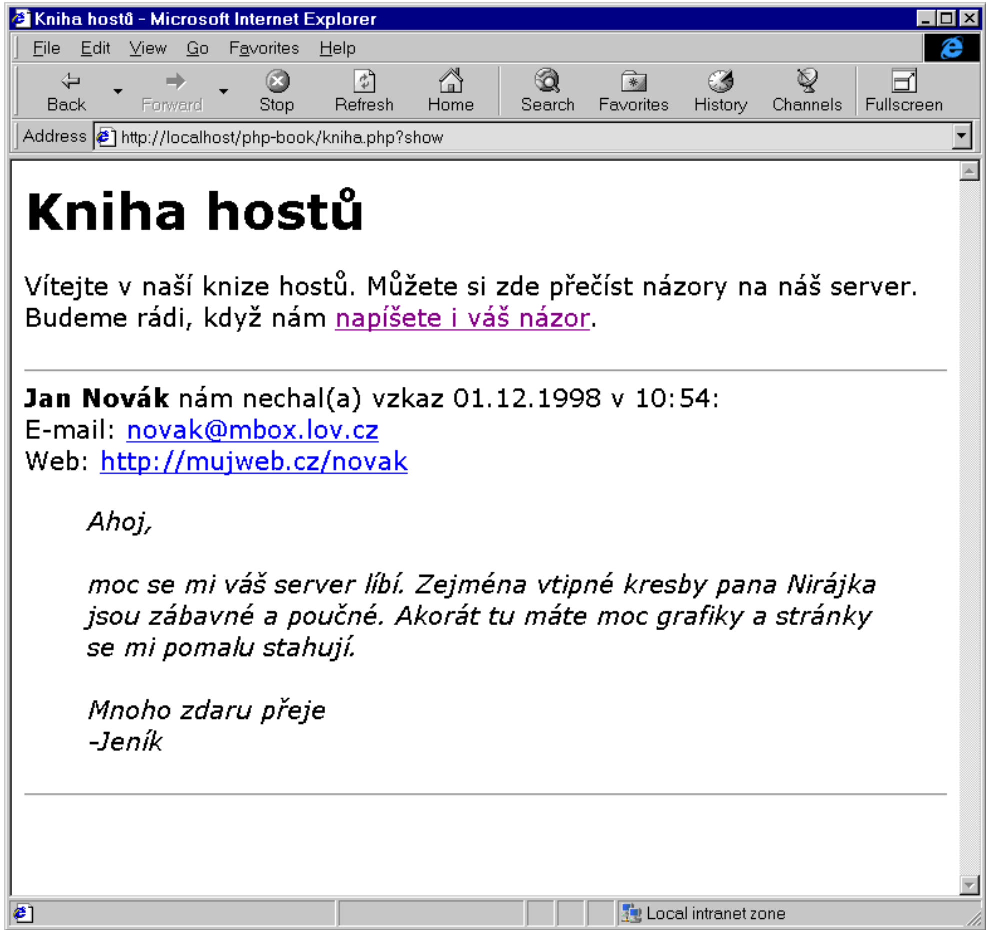
Nyní můžeme jednotlivé části skriptu poskládat dohromady v jeden plně funkční celek.

Příklad: `kniha.php`

```
<?
if ($QUERY_STRING=="show"):            // zobrazení knihy hostů

    // vypsání standardní hlavičky
    $fp = FOpen("./kniha.head", "r");
    FPassThru($fp);

    // vypsání zpráv
    $fp = FOpen("./kniha.body", "r");
    FPassThru($fp);

    // vypsání záhlaví stránky
    $fp = FOpen("./kniha.tail", "r");
    FPassThru($fp);

    exit;                              // ukončení skriptu
endif;

if (($Email!="") && ($Jmeno!="") && ($Zprava!="")): // přidání do knihy hostů

    $fp = FOpen("./kniha.body", "a");  // otevření souboru s knihou

    // přidání nové zprávy
    FPutS($fp, "<B>$Jmeno</B> nám nechal(a) vzkaz ".
               Date("d.m.Y v H:i").":<BR>\n");
    FPutS($fp, "E-mail: <A HREF='mailto:$Email'>$Email</A><BR>\n");
    if ($Web!="")
        FPutS($fp, "Web: <A HREF='$Web'>$Web</A><BR>\n");
    FPutS($fp, "<BLOCKQUOTE><I>\n");
    FPutS($fp, NL2BR(HTMLSpecialChars($Zprava)));
    FPutS($fp, "</I></BLOCKQUOTE>\n<P><HR>\n\n");

    FClose($fp);                       // zavření souboru s knihou

    // přesměrování na skript s parametrem ?show, který knihu zobrazí
    Header("Location: http://$SERVER_NAME$SCRIPT_NAME?show");
```

```
    exit;                               // ukončení skriptu
endif;

// Kniha se nezobrazuje ani nepřidáváme nově zaslanou zprávu a zobrazíme
// proto formulář pro přidání nové zprávy. Formulář se zobrazí i v případě,
// kdy uživatel nezaslal všechny povinné údaje zprávy.
?>
<HTML>
<HEAD>
<TITLE>Přidání zápisu do knihy hostů</TITLE>
<META NAME="Author" CONTENT="Jiří Kosek">
</HEAD>
<BODY>
<H1>Přidání zápisu do knihy hostů</H1>

<PRE>
<FORM METHOD=POST>
<B>Vaše jméno:</B> <INPUT NAME=Jmeno VALUE="<?echo $Jmeno?>">
<B>Váš e-mail:</B> <INPUT NAME=Email VALUE="<?echo $Email?>">
<B>Váš Web:</B>    <INPUT NAME=Web VALUE="<?echo $Web?>">

<TEXTAREA NAME=Zprava COLS=60 ROWS=8>
Sem napište váš vzkaz
</TEXTAREA>

<INPUT TYPE=Submit VALUE="Přidání zprávy">
<INPUT TYPE=Reset VALUE="Smazání formuláře">
</FORM>
</PRE>

Jméno, e-mail a zpráva jsou povinné. Adresu vašich stránek uveďte pouze
v případě, že je máte.

<P>Pokud vás opustila můza a nechcete nám nechat žádný vzkaz, můžete se
vrátit do <A HREF="kniha.php?show">knihy hostů</A>.

</BODY>
</HTML>
```

Tento jeden skript se chová třemi způsoby v závislosti na parametrech, které mu jsou předány. Pokud je mu předán parametr `show`, zobrazí pouze obsah knihy hostů. Zda byl skript volán s parametrem `show`, zjistíme pomocí obsahu proměnné `QUERY_STRING`, která obsahuje parametry předané pomocí URL za znakem `?`.



Obr. 7-8: První záznam do naší knihy byl úspěšně přidán

Pokud je však skript volán a jsou mu předána data z formuláře, přidá tato data jako nový záznam do knihy hostů. Data jsou přidána pouze v případě, že uživatel vyplnil všechny povinné údaje. Po přidání dat je prohlížeči pouze zaslán požadavek na přesměrování na náš skript s parametrem `show`, což způsobí zobrazení aktualizované knihy hostů v prohlížeči uživatele.

Pokud nenastane ani jedna z předchozích dvou situací, je zobrazen formulář, pomocí něhož může uživatel přidat nový zápis do knihy hostů.

Z našich stránek asi budeme chtít mít odkaz na knihu hostů, a ne rovnou na formulář pro přidání nové zprávy. Odkaz na knihu hostů proto musí obsahovat parametr `show`, aby skript zobrazil knihu:

```
Na našem serveru provozujeme zajímavou aplikaci
<A HREF="/sluzby/kniha.php?show">kniha hostů</A>.
```

7.3 Uživatelé Internetu mají smysl pro humor

Nevím jestli je tvrzení uvedené v nadpisu této sekce podloženo nějakým výzkumem, ale pravdou je, že dobrému vtipu se rád zasměje snad každý. Není proto lepší recept na oživení firemních stránek, než při každém přístupu na stránku zobrazit jiný vtip.

Vtipy mají i významnou úlohu při motivaci podřízených. Jeden můj kamarád měl problémy s tím, že si jeho zaměstnanci nečetli pravidelně elektronickou poštu. Začal jim proto posílat každé ráno jeden vtip právě pomocí mailu. Od té doby začali poštu pravidelně číst všichni zaměstnanci. Toto řešení má jeden problém — šéf musí chodit do práce včas, disponovat zásobou vtipů a každé ráno poslat mail všem zaměstnancům. To lze snadno odstranit skriptem, který bude každý den automaticky zasílat vtipy na zvolené adresy.

V této sekci si ukážeme, jakým způsobem na firemní stránky zařadit náhodný vtip nebo jak vytvořit službu pro automatické posílání vtipů pomocí mailu. Než se pustíme do jednotlivých aplikací, musíme si ujasnit jednu podstatnou věc — kde získat dostatečně obsáhlou sbírku vtipů.

Naštěstí lze na Internetu získat mnoho kolekcí vtipů, jejichž užívání není omezeno autorským zákonem. Velmi často seženeme kolekce vtipů jako připravené soubory pro program `fortune`. Tento program se často používá na mnoha unixových serverech k vypsání náhodně zvoleného vtipu po přihlášení k systému. My si ukážeme, jak v PHP pracovat právě s vtipy uloženými ve formátu používaném programem `fortune`.

Samotné vtipy jsou uloženy v textovém souboru (např. `vtipy.txt`) a jednotlivé vtipy jsou odděleny řádkou, která obsahuje pouze znak procento '`%`'. Část tohoto souboru může vypadat například takto:

```
Zeleznicar: Co mi tu behate po trati?
Policajti: Kdo je u Vas 'potrat'? My bezime k rychliku...
Zeleznicar: Kdo je u Vas 'krychlik'?
%
Zastavi policajt na krizovatce auto a povida:
"Poctvrte jsem vas dnes zastavil a poctvrte vam rikam, ze vam tece chladic!"
```

```
Ridic se vykloni z okna a odpovi:
"A ja vam poctvrte rikam, ze jsem kropici vuz!"
%
"Kde bydlite?"
"Nikde," odpovi tulak.
"A vy?" obrati se policista na druheho.
"Taky nikde, my jsme sousedi."
%
Ucitel hudby varuje zaka:
"Jestli me budes dale takhle zlobit, tak namluvim tve matce, ze mas velky
talent."
%
```

Aby šly náhodné vtipy rychle vybírat, je pro každý soubor s vtipy vygenerován indexový soubor, který se může jmenovat např. `vtipy.dat`. Tento soubor je binární a obsahuje postupně pozice začátku jednotlivých vtipů v textovém souboru. Pozice je vždy zapsána ve čtyřech bajtech. Začátek indexového souboru zapsaný v šestnáctkové soustavě může vypadat třeba takto:

```
0000: 00 00 00 01 00 00 08 4E 00 00 0F 30 00 00 00 20
0010: 00 00 00 00 25 00 00 00 00 00 00 00 00 00 00 5E
0020: 00 00 00 94 00 00 00 D3 00 00 02 D1 00 00 03 CB
0030: 00 00 04 CE 00 00 05 6D 00 00 07 C7 00 00 08 34
```

Pokud chceme nějaký vtip náhodně vybrat, uděláme to nejlépe tak, že si náhodně vybereme pozici nějakého vtipu v indexovém souboru a pak z datového souboru přečteme daný vtip. Podívejme se na to, jak náhodně vybrat pozici vtipu v indexovém souboru. Nejprve musíme soubor samozřejmě otevřít:

```
$fp = FOpen("./vtipy.dat", "r");
```

Vtipy chceme vybírat náhodně, musíme proto inicializovat generátor náhodných čísel pomocí funkce `SRand()`. Aby bylo číslo, kterým bude inicializovat generátor hodně proměnlivé, využijeme aktuální počet mikrosekund, který můžeme získat pomocí funkce `MicroTime()`:

```
SRand((double)MicroTime()*1e6);
```

Nyní stojíme před úkolem načíst náhodně jednu položku z indexového souboru. K nastavení pozice v souboru můžeme s výhodou využít funkci `FSeek()`. Pozici vygenerujeme následujícím způsobem. Nejprve vygenerujeme náhodné číslo, které určuje pořadí položky v indexovém souboru — počet položek v indexovém souboru zjistíme snadno — je to velikost tohoto souboru vydělená čtyřmi (`FileSize("./vtipy.dat")/4`). Skutečnou pozici v souboru pak získáme vynásobením získané náhodné hodnoty čtyřmi:

```
FSeek($fp, Rand(0, FileSize("./vtipy.dat")/4) * 4);
```

Nyní stačí přečíst ze souboru čtyři bajty dat:

```
$pos = FRead($fp, 4);
```

Nyní musíme čtyři bajty uložené v řetězci `$pos` převést na číslo:

```
$pozice = Ord($pos[3]) + 256 * (Ord($pos[2]) + 256 * (Ord($pos[1]) +
              256 * Ord($pos[0])));
```

Nyní již můžeme indexový soubor zavřít a otevřeme naopak datový soubor:

```
FClose($fp);
$fp = FOpen("./vtipy.txt", "r");
```

V souboru se nastavíme na získanou pozici — začíná na ní nějaký vtip. Pozici však musíme zmenšit o jedna, protože pozice v indexovém souboru začínají od 1, ale funkce `FSeek()` určuje pozici v souboru od 0:

```
FSeek($fp, $pozice-1);
```

Nyní můžeme vtip načíst např. do proměnné `$vtip` tím, že budeme soubor číst, dokud nenarazíme na řádku začínající znakem '%'.

```
while ($l = FGetS($fp, 128)):
    if ($l[0] == "%") break;
    $vtip .= $l;
endwhile;
```

Nakonec ještě uzavřeme soubor a jsme hotovi. Celý postup můžeme definovat jako jednu funkci, která bude vracet náhodný vtip. Funkci pak využijeme v dalších aplikacích.

Příklad: `vtip.php`

```
<?
function GetVtip()
{
    $indexFile = "./vtipy.dat";             // indexový soubor
    $dataFile = "./vtipy.txt";              // datový soubor
    $vtip = "";                             // text vtipu

    $fp = FOpen($indexFile, "r");           // otevření ind. souboru
    SRand((double)MicroTime()*1e6);         // inicializace gen náh. čísel
    FSeek($fp, Rand(0, FileSize("./vtipy.dat")/4) * 4);
                                            // nastavíme se na náhodný ukazatel
    $pos = FRead($fp, 4);                   // přečteme jej
    $pozice = Ord($pos[3]) + 256 * (Ord($pos[2]) + 256 *
```

```
            (Ord($pos[1]) + 256 * Ord($pos[0])));
                                            // spočítáme pozici v datovém souboru
    FClose($fp);                            // zavřeme indexový soubor

    $fp = FOpen("./vtipy.txt", "r");        // otevřeme datový soubor
    FSeek($fp, $pozice-1);                  // nastavíme se na vtip
    while ($l = FGetS($fp, 128)):           // čteme vtip
        if ($l[0] == "%") break;            // dokud nenarazíme na jeho konec
        $vtip .= $l;
    endwhile;
    FClose($fp);                            // zavřeme datový soubor

    return $vtip;                           // vrátíme vtip
    }
?>
```

Atraktivní domovská stránka

Internet a zejména pak Web se v poslední době stal opravdu oblíbeným místem pro zveřejňování informací a reklamy. Velkým uměním je však uživatele zaujmout tak, aby naše stránky často navštěvoval. Jedním z nejúčinnějších způsobů je nabízet na stránkách zajímavé nebo zábavné informace, které se však často aktualizují. Přesně do tohoto kontextu zapadá zařazení náhodně generovaného vtipu na domovskou stránku. Pokud vtipy budou kvalitní, máme zaručeno, že naše stránky budou uživatelé navštěvovat častěji než stránky konkurenční firmy, která má na Internetu vystaven pouze svůj ceník aktualizovaný jednou za tři měsíce.

Vzhledem k tomu, že již máme hotovou funkci pro získání náhodně vybraného vtipu, je zařazení náhodně generovaného vtipu na stránku hračkou:

Příklad: `home.php`

```
<HTML>
<HEAD>
<TITLE>Vítejte ve firmě DíkyVtipůmÚspěšná a.s.</TITLE>
</HEAD>
<BODY>
<H1>Vítejte ve firmě DíkyVtipůmÚspěšná a.s.</H1>

<P>Na našich stránkách získáte všechny podrobné informace o našich
výrobcích včetně cen. Od června je v provozu i náš virtuální obchodní
dům.
```

```
<P>Abyste se nenudili, máme pro vás pokaždé nachystán čerstvý vtip:

<BLOCKQUOTE>
<?
    require("./vtip.php");
    echo NL2BR(GetVtip());
?>
</BLOCKQUOTE>

</BODY>
</HTML>
```

Pomocí příkazu `require` načteme soubor s definicí funkce `GetVtip()`. V textu vtipu pomocí funkce `NL2BR()` nahradíme všechny konce řádků tagem `<BR>`, aby bylo zachováno formátování.

Automatické zasílání vtipů mailem

Když už máme k dispozici funkci `GetVtip()`, která nám generuje náhodné vtipy, byla by škoda ji nevyužít i nějakým dalším způsobem. Ukážeme si proto, jak pomocí PHP vytvořit službu, která zájemcům každý den odešle požadovaný počet vtipů elektronickou poštou. V tomto příkladu si ukážeme spoustu zajímavých věcí — mimo jiné i to, jak pomocí PHP skriptů provádět nějaké pravidelné činnosti, které nezávisí na vyvolání požadavku uživatelem.

Jak by měla vypadat taková služba zasílající vtipy e-mailem? Je jasné, že se bude skládat nejméně ze dvou částí. První částí bude skript, který umožní uživateli přihlásit si odběr vtipů. Tento skript musí někam uložit seznam všech adres, na které se mají vtipy posílat. Druhou částí aplikace bude skript, který se bude volat pravidelně každý den. Úkolem tohoto skriptu bude rozeslání vtipů na adresu všech přihlášených uživatelů.

Oba dva skripty musí společně sdílet seznam e-mailových adres, na které se mají posílat vtipy. Seznam adres můžeme uložit několika způsoby — např. do databáze nebo do obyčejného souboru. My v naší ukázce budeme používat jednoduchou souborovou databázi dbm, která je standardní součástí většiny unixových distribucí. Využít ji můžeme i ve Windows, protože PHP pro Windows obsahuje knihovnu pro práci se soubory dbm.

Dbm je velice jednoduchá databáze, která však nalezne uplatnění v mnoha aplikacích díky své poměrně velké rychlosti a nízkým nárokům na systémové zdroje. Databáze je v dbm chápána jako množina uspořádaných dvojic — klíč, hodnota. Klíč i hodnota jsou přitom libovolné textové řetězce. Funkce dbm umožňují velice snadno a efektivně pod daný klíč uložit nějakou hodnotu a tuto hodnotu podle klíče zase získat.

Před prací s databází ji nejprve musíme vytvořit. Databázi vytvoříme tak, že při jejím otevírání pomocí funkce `dbmOpen()` specifikujeme režim `c` (create — vytvoření). Chceme-li seznam adres pro posílání vtipů uložit do databáze `uzivatele`, můžeme použít následující krátký skript pro její vytvoření:

```
$uzivatele = "./uzivatele";
$db = dbmOpen($uzivatele, "c");
if (!$db)
   echo "Nepodařilo se vytvořit databázi.";
dbmClose($db);
```

Pokud již máme databázi vytvořenou, můžeme ji otevírat dvěma způsoby: buď pouze pro čtení (režim `r`), nebo pro čtení/zápis (režim `w`).

Podívejme se nyní na to, co budeme do databáze ukládat. Určitě zde budeme mít uloženou e-mailovou adresu, kam se mají vtipy posílat. Navíc si ke každé adrese uložíme počet vtipů, které se mají zasílat. Aby nemohl někdo neoprávněně měnit nastavení zasílání vtipů ostatních uživatelů, uložíme společně s počtem vtipů i heslo, kterým budeme chránit přístup k nastavení.

Vidíme, že v naší aplikaci se jako klíč pro záznamy v databázi hodí e-mailová adresa, protože je pro každého uživatele jedinečná. Pro každý klíč můžeme do databáze uložit pouze jednu hodnotu. Musíme tedy uložit najednou požadovaný počet vtipů i heslo — uložíme je tedy jako jeden řetězec, kde bude počet vtipů od hesla oddělen dvojtečkou.

Ukažme si jak vypadá přidání uživatele s adresou `jkj@ucw.cz` do databáze, pokud by chtěl 5 vtipů denně a jeho heslo by bylo `kojot`. Pro vložení záznamu použijeme funkci `dbmInsert()` a předpokládáme, že databáze je již otevřena pro čtení a její identifikátor máme uložen v proměnné `$db`:

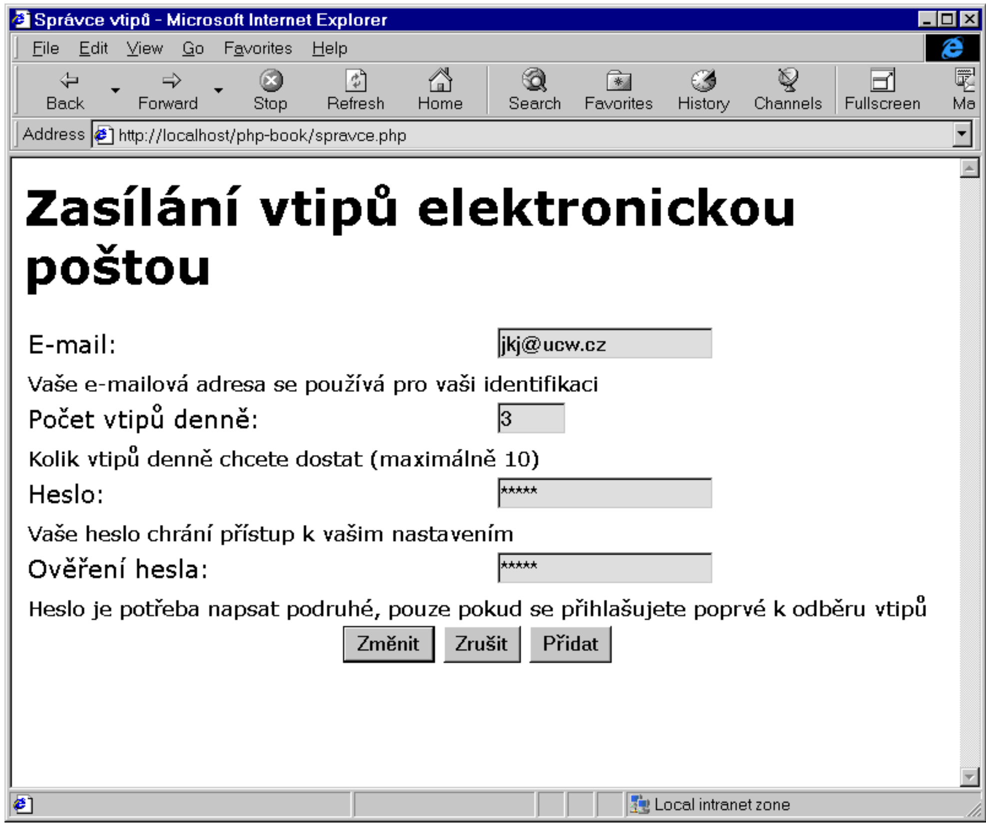
```
dbmInsert($db, "jkj@ucw.cz", "5:kojot");
```

Jelikož nejsou soubory dbm ukládány nijak šifrované, můžeme pro zvýšení úrovně zabezpečení ukládaná hesla šifrovat. V PHP se pro tento účel nabízejí hned dvě funkce — `Crypt()` a `MD5()`. My použijeme třeba druhou z nich:

```
$klic = "jkj@ucw.cz";
$hodnota = "5:".MD5("kojot");
dbmInsert($db, $klic, $hodnota);
```

V tomto případě bude například ukládaná hodnota následující

```
5:76aabf1501c675d9eed275f6beb25c02
```



Obr. 7-9: Rozhraní pro přihlášení k odběru vtipů

Na první pohled je heslo nečitelné a z principu algoritmu MD5 je zřejmé, že heslo nelze ze zobrazené hodnoty odvodit jinak než pomocí útoku hrubou silou.

Pokud se uživatel rozhodne změnit počet zasílaných vtipů, musíme pod jeho e-mailovou adresu uložit do databáze novou hodnotu. K tomu slouží funkce `dbmReplace()`. Její parametry jsou shodné jako u funkce `dbmInsert()` — rozdíl je v tom, že hodnota není přidána, ale změněna.

Pokud se uživatel rozhodne ukončit odběr vtipů, musíme jej z databáze vyřadit. K tomu slouží funkce `dbmDelete()`. Záznam, který chceme smazat, se určuje opět pomocí svého klíče:

```
dbmDelete($db, "jkj@ucw.cz");
```

Poslední funkcí pro práci s dbm, kterou náš skript používá, je `dbmFetch()`. Tato funkce vrací hodnotu uloženou v databázi pro určitý klíč. Počet vtipů a zašifrované heslo pro našeho uživatele můžeme získat voláním

```
$hodnota = dbmFetch($db, "jkj@ucw.cz");
```

Nyní nám již nic nebrání ve vytvoření skriptu, který umožní uživatelům přihlásit se k odběru vtipů, změnit počet odebíraných vtipů či se od odběru zase odhlásit.

Příklad: spravce.php

```
<HTML>
<HEAD>
<TITLE>Správce vtipů</TITLE>
</HEAD>
<BODY>
<H1>Zasílání vtipů elektronickou poštou</H1>
<?

Dl("php3_dbm.dll");                  // knihovna pro práci s dbm (ve Windows)

$chyba = false;                      // indikátor chyby při běhu skriptu
$uzivatele = "./uzivatele";          // soubor s databází

function Chyba($zprava)              // nastavení chyby a chybového hlášení
{
    global $chyba, $chybaText;

    $chyba = true;
    $chybaText = $zprava;
}


// akčí část skriptu - práce s databází
switch ($Akce):                      // větvíme skript podle druhu akce

    case "Přidat":                   // přidání nové adresy pro odběr vtipů

        $db = dbmOpen($uzivatele, "w");     // otevření databáze
        if (!$db):
            Chyba("Nepodařilo se přidat nový záznam do databáze.");
            break;
        endif;
        if (dbmExists($db, $Email)):// je daný e-mail v databázi?
            Chyba("Záznam pro tuto adresu již existuje.");
            break;
        endif;
```

```
    if ($Heslo!=$Heslo2):       // shodují se obě zadaná hesla?
        Chyba("Zadaná hesla se neshodují, zadejte hesla znovu.");
        break;
    endif;
    if (($Pocet<0) || ($Pocet>10)):     // správný počet vtipů?
        Chyba("Počet zasílaných vtipů musí být od 0 do 10.");
        break;
    endif;
    $hodnota = $Pocet.":".MD5($Heslo);// hodnota pro přidání do databáze
    if (dbmInsert($db, $Email, $hodnota)!=0):   // přidání záznamu
        Chyba("Záznam se bohužel nepodařilo do databáze přidat.");
        break;
    endif;
    dbmClose($db);              // zavření databáze

    break;

case "Změnit":                  // změna počtu odebíraných vtipů

    $db = dbmOpen($uzivatele, "w");     // otevření databáze
    if (!$db):
        Chyba("Nepodařilo se změnit počet odebíraných vtipů.");
        break;
    endif;
    $hodnota = dbmFetch($db, $Email);   // zjištění stávající hodnoty
    if (!$hodnota):
        Chyba("Požadovaný záznam není v databázi.
               Asi jste zadali špatnou mailovou adresu.");
        break;
    endif;
    list($staryPocet, $stareHeslo) = Explode(":", $hodnota);
            // do proměnné $staryPocet načteme původní počet vtipů
            // do proměnné $stareHeslo načteme heslo
    if (MD5($Heslo)!=$stareHeslo):  // shoduje se zadané a uložené heslo?
        Chyba("Špatné heslo.");
        break;
    endif;
    if (($Pocet<0) || ($Pocet>10)): // správný počet vtipů
        Chyba("Počet zasílaných vtipů musí být od 0 do 10.");
        break;
    endif;
    $hodnota = $Pocet.":".MD5($Heslo);  // nová hodnota do databáze
```

```
        if (dbmReplace($db, $Email, $hodnota)!=0):// změna stávající hodnoty
            Chyba("Záznam se bohužel nepodařilo změnit.");
            break;
        endif;
        dbmClose($db);          // zavření databáze

        break;

    case "Zrušit":              // odhlášení odběru vtipů

        $db = dbmOpen($uzivatele, "w");     // otevření databáze
        if (!$db):
            Chyba("Nepodařilo se zrušit odběr vtipů.");
            break;
        endif;
        $hodnota = dbmFetch($db, $Email);   // přečtení hodnoty záznamu
        if (!$hodnota):
            Chyba("Požadovaný záznam není v databázi.
                   Asi jste zadali špatnou mailovou adresu.");
            break;
        endif;
        list($Pocet, $stareHeslo) = Explode(":", $hodnota);
            // nyní nás zajímá zejména heslo, které uložíme do proměnné
            // $stareHeslo, proměnná $Pocet obsahuje počet odebíraných vtipů
        if (MD5($Heslo)!=$stareHeslo):  // shoduje se zadané heslo s uloženým?
            Chyba("Špatné heslo.");
            break;
        endif;
        if (dbmDelete($db, $Email)!=0): // smazání záznamu
            Chyba("Nepodařilo se zrušit odběr vtipů.");
            break;
        endif;
        dbmClose($db);          // zavření databáze

        break;

endswitch;

// vypsání chybového hlášení v případě chyby
if ($chyba):
    echo "Došlo k následující chybě:<BR>\n";
    echo "<FONT COLOR=RED>$chybaText</FONT>\n";
```

```
else:
    switch ($Akce):

        case "Přidat":
            echo "Následující záznam byl úspěšně přidán.\n";
            break;

        case "Změnit":
            echo "Záznam byl úspěšně změněn. Aktuální nastavení:\n";
            break;

        case "Zrušit":
            echo "Automatické zasílání vtipů bylo zrušeno.
                  Vaše poslední nastavení:\n";
            break;

    endswitch;
endif;

?>

<FORM ACTION="spravce.php" METHOD=POST>
<TABLE>
<TR><TD>E-mail:<TD><INPUT NAME=Email VALUE="<?echo $Email?>">
<TR><TD COLSPAN=2><SMALL>Vaše e-mailová adresa se používá pro vaši
                  identifikaci</SMALL>
<TR><TD>Počet vtipů denně:<TD><INPUT NAME=Pocet SIZE=5 VALUE="<?echo $Pocet?>">
<TR><TD COLSPAN=2><SMALL>Kolik vtipů denně chcete dostat (maximálně 10)</SMALL>
<TR><TD>Heslo:<TD><INPUT TYPE=Password NAME=Heslo>
<TR><TD COLSPAN=2><SMALL>Vaše heslo chrání přístup k vašim nastavením</SMALL>
<TR><TD>Ověření hesla:<TD><INPUT TYPE=Password NAME=Heslo2>
<TR><TD COLSPAN=2><SMALL>Heslo je potřeba napsat podruhé,
                  pouze pokud se přihlašujete poprvé k odběru vtipů</SMALL>
<TR><TH COLSPAN=2>
    <INPUT TYPE=Submit NAME=Akce VALUE="Změnit">
    <INPUT TYPE=Submit NAME=Akce VALUE="Zrušit">
    <INPUT TYPE=Submit NAME=Akce VALUE="Přidat">
</TABLE>
</FORM>

</BODY>
</HTML>
```

Poněkud podivný příkaz `Dl("php3_dbm.dll")` slouží pro načtení knihovny pro práci s dbm. Tento příkaz používáme, pokud běží PHP ve Windows. V unixové verzi musíme mít obvykle podporu dbm přímo zakompilovanou do systému PHP.

Stojíme nyní před úkolem vytvořenit skript, který každý den odešle vtipy elektronickou poštou na požadované adresy. Samotný skript bude velice jednoduchý. Jeho úkolem bude na každou adresu v databázi odeslat požadovaný počet vtipů. Průchod celou databází nám usnadní funkce `dbmFirstKey()` a `dbmNextKey()`. První funkce vrací hodnotu prvního klíče uloženého v databázi. Funkce `dbmNextKey()` vrací klíč, který následuje ihned po zadaném klíči. K odeslání pošty využijeme funkci `Mail()`.

Příklad: `mailvtip.php`

```
<?
    Set_Time_Limit(0);              // skript může běžet neomezeně dlouho
    require("./vtip.php");          // funkce pro generování vtipů
    Dl("php3_dbm.dll");             // knihovna pro práci s dbm

    $uzivatele = "./uzivatele";     // soubor s databází
    $db = dbmOpen($uzivatele, "r"); // otevření databáze
    if (!$db) exit;                 // při chybě skončíme

    $email = dbmFirstKey($db);      // první e-mail v databázi

    while ($email):                 // postupně zpracujeme celou databázi

        list($pocet,) = Explode(":", dbmFetch($db, $email));
                                    // do proměnné $pocet uložíme počet vtipů
        $mail = "";                 // text dopisu

        // generování požadovaného počtu vtipů
        for ($i=0; $i<$pocet; $i++):
            $mail .= GetVtip();     // připojení vtipu do dopisu
            $mail .= "\n====================\n";    // oddělovač vtipů
        endfor;

        $mail .= "Vtipy vám automaticky rozesílá služba
                  http://mujserver.cz/vtipy/.\nPokud nechcete vtipy dále
                  odebírat, můžete si je odhlásit na adrese služby.\n";

        if ($pocet>0)               // pokud jsou nějaké vtipy, odešleme je
            mail($email, "Vtipy na ".Date("d.m.Y"),
                 $mail, "X-Mailer: VtipDeamon/1.0");
```

```
        $email = dbmNextKey($db, $email);   // další e-mail v databázi

    endwhile;

    dbmClose($db);                    // uzavření databáze
?>
```

Na začátku skriptu pomocí funkce `Set_Time_Limit()` vypneme maximální dobu, po kterou může skript běžet. Tento limit je standardně 30 sekund a mohlo by se stát, že v tak krátkém čase by se nestihly odeslat všechny maily. Novinkou pro nás může být funkce `Mail()`, která slouží k odeslání dopisu. Jejím prvním parametrem je e-mailová adresa, na kterou se mají posílat vtipy, druhým pak předmět (subject) dopisu. V našem případě do předmětu dopisu doplníme aktuální datum. Třetí parametr je samotné tělo dopisu. Poslední parametr je nepovinný a umožňuje přidání dalších dodatečných hlaviček, které se odešlou jako součást dopisu.

Kdykoliv nyní spustíme skript `mailvtip.php`, odešlou se na všechny adresy v databázi vtipy. My však potřebujeme, aby se skript spouštěl automaticky každý den. Pro splnění tohoto požadavku již nevystačíme pouze s webovým serverem, který podporuje PHP. Musíme sáhnout k možnostem našeho operačního systému. Naštěstí má valná většina operačních systémů službu, která umožňuje v daný časový okamžik spustit libovolný program. Ve Windows NT je to služba Schedule, v Unixu pak nejčastěji démon `crond`.

Podívejme se nejprve na to, jak zajistit automatické spouštění skriptu `mailvtip.php` v prostředí Windows NT. Prvním předpokladem je spuštěná služba Schedule.

Služba Schedule není standardně ve Windows zapnuta. Musíme ji proto ručně spustit buď pomocí příkazu

```
net start schedule
```

nebo pomocí příkazu Services v Control Panelu. Rovnou bychom měli změnit způsob spouštění této služby z Manual na Automatic, aby se služba automaticky spustila po každém restartování počítače. Vzhledem k tomu, jak často je obvykle potřeba NT restartovat, ušetříme si tím spoustu práce.

Pokud nám běží služba Schedule, můžeme používat příkaz `at`, který slouží ke spouštění příkazu v daný čas. Příkaz, který budeme chtít spouštět, bude mít tvar

```
c:\php3\php.exe -f c:\skripty\mailvtip.php
```

Tento příkaz vyvolá spuštění interpretu PHP a předá mu ke zpracování skript uložený v souboru `c:\skripty\mailvtip.php`.

Pokud budeme chtít automatické rozesílání vtipů spouštět každý den v týdnu v jednu hodinu ráno, použijeme následující příkaz

```
at 01:00 /every=M,T,W,Th,F,S,Su c:\php3\php.exe -f c:\skripty\mailvtip.php
```

Pokud se chceme přesvědčit, zda je opravdu skript zaregistrován pro automatické spouštění, stačí zadat příkaz `at` bez parametrů. Vypíše se pak seznam všech automaticky spouštěných programů.

Ve Windows jsme již náš problém vyřešili, pojďme se tedy podívat, jak na něj v Unixu. V Unixu situaci poněkud komplikuje to, že PHP je obvykle používáno jako modul zakompilovaný přímo do serveru Apache. Nemůžeme jej pak volat z příkazové řádky pro interpretování nějakého skriptu. Jedinou možností vyvolání skriptu je zaslání požadavku webovému serveru. Asi nejlepší cestou je zkompilovat PHP ještě jednou, tentokráte však jako CGI-skript. Vznikne nám tak spustitelný soubor `php`, který můžeme použít k interpretování skriptů. Interpret pak umístíme do nějakého adresáře — např. `/sbin/php`.

Pro pravidelné spouštění programů se na Unixu používá démon `crond`. K jeho ovládání slouží příkaz `crontab`. Po zadání příkazu `crontab -e` se spustí editor, ve kterém přidáme následující řádku:[2]

```
0 1 * * * /sbin/php -f /skripty/mailvtip.php
```

Po přidání této řádky soubor v editoru uložíme a z editoru vyskočíme. Příkaz `crontab` následně provede úpravu datových souborů pro `crond`. Od této chvíle se každý den v jednu hodinu ráno (1 hodina a 0 minut) spustí PHP a předá se mu ke zpracování skript uložený v souboru `/skripty/mailvtip.php`.

Skripty, které se pravidelně spouštějí pro splnění nějakých administrativních úkolů, by neměly být umístěny v adresářovém stromu WWW-serveru. Pokud by skript byl dostupný z venku pomocí prohlížeče, mohl by kdokoliv zvenčí vyvolat spuštění skriptu, což může vést k narušení funkčnosti aplikace.

[2] Často se jako implicitní editor spouští `vi`. Pokud nevíte, jak z něj vyskočit, napište sekvenci znaků `:q`. Editor, který program `crontab` spouští, můžete nastavit pomocí proměnné `VISUAL`. Pokud je vaším oblíbeným editorem `joe`, můžete pro jeho nastavení použít příkaz `export VISUAL=joe`, resp. `setenv VISUAL joe` podle používaného shellu.

7.4 PHP jako brána k dalším službám

PHP v sobě obsahuje mnoho funkcí, které podporují několik běžně používaných internetových protokolů. Nalezneme zde funkce podporující protokoly HTTP, FTP, SMTP, SNMP, IMAP, POP, NNTP, LDAP a další. Kromě toho můžeme do PHP přidat podporu libovolného dalšího protokolu. Rozšíření můžeme provést přes API pro programovací jazyk C nebo můžeme funkce realizovat přímo v PHP s využitím funkcí pro práci se sockety. V této sekci si ukážeme využití podpory různých protokolů.

Vyhledávání e-mailových adres

Služeb pro vyhledávání e-mailových adres je dnes na Internetu mnoho. V poslední době patří mezi nejpopulárnější metody ukládání adres do adresářů přístupných pomocí protokolu LDAP (Lightweight Directory Access Protocol). V Česku mezi servery podporující tento protokol patří například `http://lide.seznam.cz` a `http://lide.atlas.cz`. Na uvedených adresách je k dispozici webové rozhraní, které umožňuje vyhledávání adres a jejich přidávání. Kromě toho lze k informacím o mailových adresách přistupovat pomocí protokolu LDAP. Toho využívají mnohé novější poštovní programy a umožňují přímo ze svého prostředí provést dotaz na e-mailovou adresu.

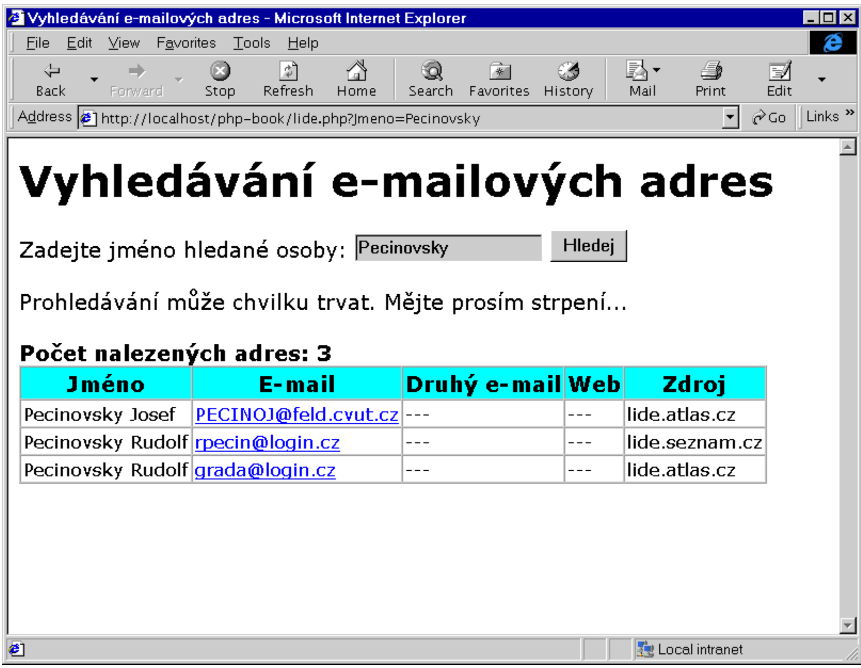
Podpora LDAP je k dispozici i v PHP, a tak nestojí nic v cestě pro vytvoření služby, která bude hledat e-mailovou adresu hned v několika adresářích najednou — tím ušetříme uživateli spoustu času, protože by jinak musel dotaz zadávat několikrát pro různé adresářové servery.

Nejprve se podívejme na výsledek celého našeho snažení. Na obrázku 7-10 na následující straně můžeme vidět, jak dopadlo hledání e-mailové adresy pana Pecinovského pomocí našeho vyhledávacího skriptu.

Celá služba je přitom implementována jedním, poměrně jednoduchým skriptem.

Příklad: `lide.php`

```
<HTML>
<HEAD>
<TITLE>Vyhledávání e-mailových adres</TITLE>
</HEAD>
<BODY>
<H1>Vyhledávání e-mailových adres</H1>
<FORM>
Zadejte jméno hledané osoby:
<INPUT NAME=Jmeno VALUE="<?echo $Jmeno?>">
<INPUT TYPE=Submit VALUE="Hledej">
```



Obr. 7-10: Výsledek hledání adresy v několika adresářových službách

```
</FORM>

<?
// funkce slouží pro porovnání prvků pole osoby podle uloženého jména
function OsobyCmp($a, $b)
{
    return StrCaseCmp($a["Jmeno"], $b["Jmeno"]);
}

if ($Jmeno!=""):        // hledáme pouze, pokud je zadán nějaký dotaz

    echo "<P>Prohledávání může chvilku trvat. Mějte prosím strpení...\n";
    Flush();            // okamžité odeslání předchozího textu do prohlížeče
    Set_Time_Limit(0);  // skript je časově náročný -- vypneme timeout

    Dl("php3_ldap.dll");    // DLL knihovna pro práci s LDAP (pouze Windows)

    $servery = Array("lide.atlas.cz", "lide.seznam.cz");
                        // seznam prohledávaných serverů
    UnSet($osoby);      // nulování pole pro uložení výsledků
```

```
for ($i=0; $i<Count($servery); $i++):   // pro všechny servery
    $ds = LDAP_Connect($servery[$i]);   // se připojíme k serveru
    if (!$ds) continue;
                    // nepodařilo se připojit => další server
    if (!LDAP_Bind($ds)) continue;
                    // nepodařilo se autentifikovat => další server
    $vysledek = LDAP_Search($ds, "", "cn=$Jmeno*",
        Array("sn", "givenname", "rfc822mailbox", "othermailbox", "url"));
                    // vlastní provedení dotazu v adresáři
    if (!$vysledek) { LDAP_Close($ds); continue; };
                    // dotaz zhavaroval => další server
    $polozka = LDAP_First_Entry($ds, $vysledek);
                    // načteme první položku výsledku prohledávání
    while ($polozka):                   // pro všechny položky
        $attr = LDAP_Get_Attributes($ds, $polozka); // přečteme atributy
        $osoby[] = Array("Jmeno" => $attr["sn"][0]." ".
                            $attr["givenname"][0],
                     "Mail" => $attr["rfc822mailbox"][0],
                     "Mail2" => $attr["othermailbox"][0],
                     "URL" => $attr["url"][0],
                     "Server" => $i);
                     // přidáme obsah atributů do pole $osoby s výsledky
        $polozka = LDAP_Next_Entry($ds, $polozka);// čteme další položku
    endwhile;
    //LDAP_Free_Result($vysledek);
                    // uvolnění paměti výsledku (na Windows způsobí pád
                    // PHP, proto jsme příkaz odkomentovali)
    LDAP_Close($ds);// ukončení spojení se serverem
endfor;

if (Count($osoby)==0):                  // nalezli jsme vůbec něco?
    echo "<P><STRONG>Vašemu dotazu nevyhovuje žádný záznam.</STRONG>\n";
else:                                   // vypsání výsledku
    echo "<P><STRONG>Počet nalezených adres: ",
          Count($osoby), "</STRONG>\n"; // vypsání počtu nalezených adres

    USort($osoby, "OsobyCmp");          // seřazení záznamů podle abecedy

    echo "<TABLE BORDER=1 CELLSPACING=0 CELLPADDING=2>\n";  // tabulka
    echo "<TR BGCOLOR=AQUA><TH>Jméno<TH>E-mail<TH>Druhý e-mail
                        <TH>Web<TH>Zdroj</TR>\n";
```

```
        for ($i=0; $i<Count($osoby); $i++): // zpracování všech adres
            echo "<TR>";
            echo "<TD><SMALL>", $osoby[$i]["Jmeno"], "</SMALL>",
                 "<TD><SMALL>", $osoby[$i]["Mail"]=="" ?
                           "---" :         // když není mail
                           "<A HREF=\"mailto:".$osoby[$i]["Mail"]."\">".
                               $osoby[$i]["Mail"]."</A>",
                     "</SMALL>",
                 "<TD><SMALL>", $osoby[$i]["Mail2"]=="" ?
                           "---" :         // když není 2. mail
                           "<A HREF=\"mailto:".$osoby[$i]["Mail2"]."\">".
                               $osoby[$i]["Mail2"]."</A>",
                     "</SMALL>",
                 "<TD><SMALL>", $osoby[$i]["URL"]=="" ?
                           "---" :         // když není domovská stránka
                           "<A HREF=\"".$osoby[$i]["URL"]."\">".
                               $osoby[$i]["URL"]."</A>",
                     "</SMALL>",
                 "<TD><SMALL>", $servery[$osoby[$i]["Server"]], "</SMALL>";
                                           // server, který poskytl adresu
            echo "</TR>\n";
        endfor;
        echo "</TABLE>\n";
    endif;

endif;
?>

</BODY>
</HTML>
```

My si alespoň stručně popíšeme princip funkce celého skriptu. Začátek skriptu obsahuje jednoduchý formulář, který slouží k zadání jména osoby, pro níž hledáme e-mailovou adresu. Jméno je uloženo do proměnné `$Jmeno`.

Nalezené adresy ukládáme do pole `$osoby`, které na začátku vynulujeme pomocí funkce `UnSet()`. Hledání probíhá na všech serverech, které jsou uloženy v poli `$servery`. Pokud bychom chtěli adresy hledat i na jiných serverech podporujících protokol LDAP, stačí je přidat jako další prvky pole.

Pro každý server pak opakujeme jednoduchý postup. Nejprve pomocí funkce `LDAP_Connect()` vytvoříme spojení se serverem. Poté se k serveru anonymně přihlásíme voláním funkce `LDAP_Bind()`.

O samotné prohledání adresáře se postará funkce `LDAP_Search()`. Jako podmínku hledání jsme specifikovali, že hledáme všechny položky adresáře, které mají v atributu `cn` uloženo jméno hledané osoby. Atribut `cn` se často používá jako jakýsi identifikátor položek v adresáři a obsahuje jméno a e-mailovou adresu. Poslední parametr předaný funkci `LDAP_Search()` je pole obsahující seznam atributů, které chceme pro každou položku získat. Každá položka v adresáři má obvykle několik desítek atributů. Nás však zajímají pouze některé, a proto požádáme pouze o přenesení požadovaných atributů. Atribut `sn` obsahuje příjmení (surname), `givenname` obsahuje křestní jméno, `rfc822mailbox` adresu elektronické pošty, `othermailbox` alternativní adresu elektronické pošty a atribut `url` je vyhrazen pro uložení adresy domovské stránky.

Na tomto místě musíme poznamenat, že pojmenování atributů v LDAP adresářích je zcela ponecháno na vůli jejich tvůrců. Výše uvedené atributy jsou však jakýmsi nepsaným standardem, který používá valná většina veřejně dostupných serverů.

Po získání výsledku prohledávání pak zpracováváme jeho jednotlivé položky. Pomocí funkce `LDAP_First_Entry()` načteme první položku výsledku. Z ní pomocí funkce `LDAP_Get_Attributes()` získáme obsah jejích atributů. Hodnoty atributů ukládáme do pole `$osoby` — každou položku výsledku ukládáme jako malé asociativní pole. Pomocí funkce `LDAP_Next_Entry()` se posouváme stále na další a další položky výsledku, dokud nějaké existují. Nakonec uzavřeme spojení se serverem pomocí `LDAP_Close()` a pokračujeme vyhledáváním na dalším serveru ze seznamu `$servery`.

Po skončení vyhledávání setřídíme pole s výsledky podle jména a výsledek vypíšeme v přehledné formě do tabulky.

Jednoduchý POP klient

Pro přístup ke schránkám elektronické pošty se nejčastěji využívá protokol POP (Post Office Protocol). Některé poštovní servery podporují i novější protokol IMAP. V PHP máme k dispozici knihovnu pro práci s protokolem IMAP. Tato knihovna však umí pracovat i s protokolem POP. My si teď ukážeme, jak vytvořit jednoduchého klienta pro čtení pošty uložené v nějaké schránce, ke které máme přístup pomocí protokolu POP.

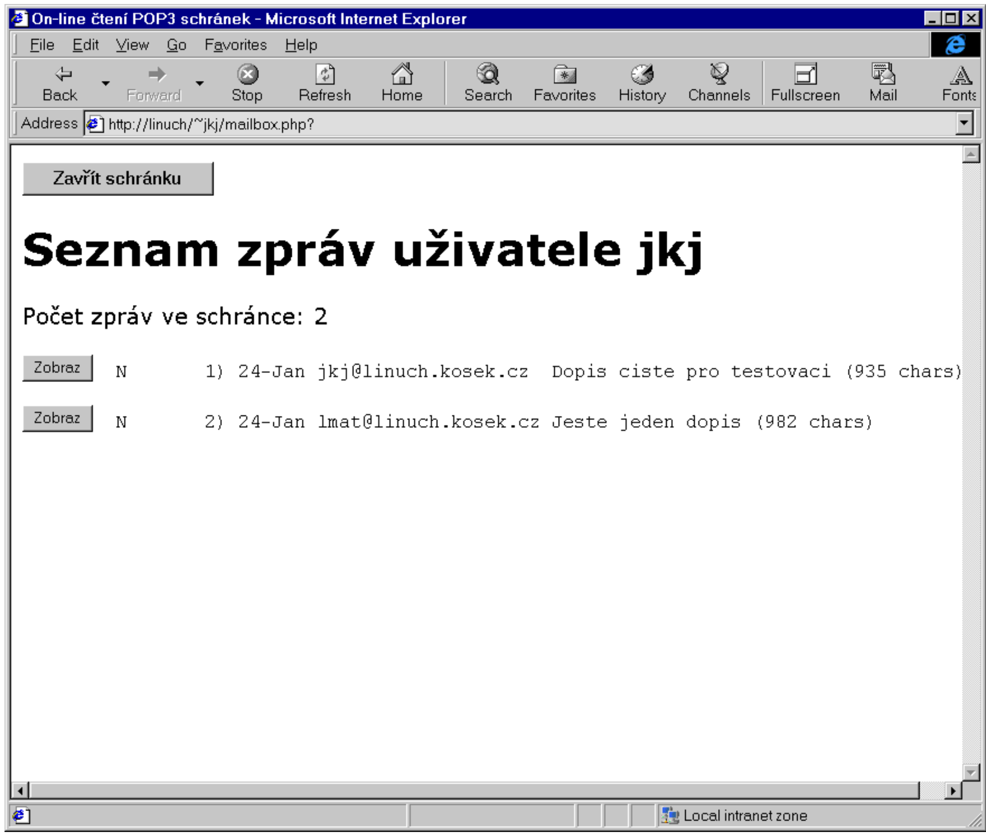
Klient nám umožní zadat adresu serveru, jméno a heslo. Poté vypíše seznam zpráv, které schránka obsahuje. U každé zprávy bude umístěno tlačítko, které nám umožní prohlédnout si kompletní obsah dopisu.

Nejprve si popišme speciální funkce, které budeme pro naši aplikaci potřebovat. Pro připojení k POP serveru využijeme funkci `IMAP_Open()`. Jejími parametry jsou adresa serveru, použitý protokol, jméno schránky, jméno uživatele a heslo. Pokud tedy máme poštovní schránku na počítači `mbox.nekde.cz`, naše jméno je `hogo` a heslo máme `fogo`, připojíme se k serveru pomocí příkazu:

```
$spojeni = IMAP_Open("{mbox.nekde.cz/pop3}INBOX", "hogo", "fogo");
```

Funkce vrací identifikátor spojení, který využíváme v dalších funkcích. Pokud se nepodaří k serveru připojit, vrací funkce `false`.

Nyní můžeme využívat nepřeberné množství funkcí pro manipulaci se schránkou. Jejich kompletní výčet a popis naleznete v referenční části knihy — my se seznámíme pouze s některými z nich.



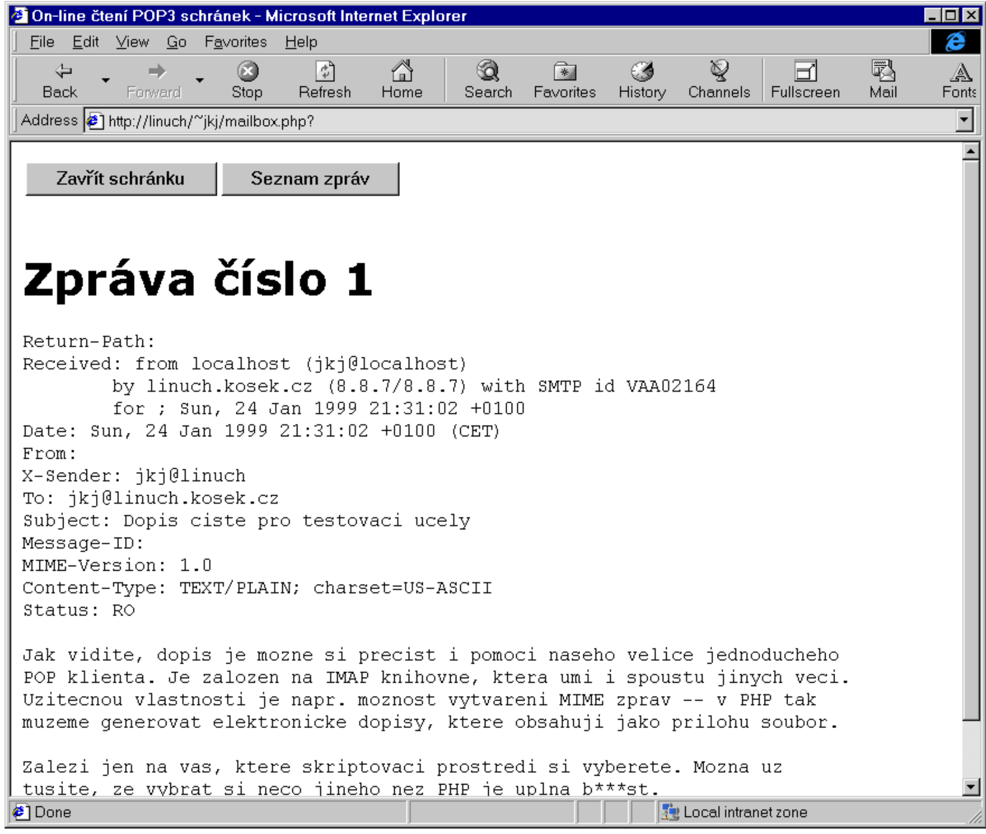
Obr. 7-11: Přehled zpráv v poštovní schránce

Pomocí funkce `IMAP_Num_Msg()` zjistíme počet zpráv ve schránce. Tato funkce je důležitá, protože v ostatních funkcích se odvoláváme na číslo zprávy — musíme tedy vědět, kolik zpráv je ve schránce k dispozici. Zprávy jsou přitom číslovány od jedné.

Pokud chceme získat hlavičku dopisu, použijeme `IMAP_FetchHeader()`. Jako parametr musíme předat číslo spojení se serverem a číslo zprávy. Pokud chceme přečíst a zobrazit hlavičky třetího dopisu, můžeme použít následující kód:

```
echo IMAP_FetchHeader($spojeni, 3);
```

Zcela obdobně můžeme získat i tělo dopisu pomocí funkce `IMAP_Body()`.



Obr. 7-12: Kompletní dopis

Mezi další zajímavé funkce patří například `IMAP_Headers()`. Funkce vrací pole, kde každý prvek obsahuje řetězec se stručnou charakteristikou jedné zprávy. Charakteristika zahrnuje datum odeslání zprávy, jejího odesílatele, předmět zprávy a její velikost.

Slušností bývá po ukončení práce se serverem se od něj odpojit pomocí `IMAP_Close()`. Jako parametr musíme uvést číslo spojení vytvořeného pomocí `IMAP_Open()`.

Pomocí výše uvedených funkcí již můžeme dát dohromady jednoduchou aplikaci, která umožňuje přístup k poštovním schránkám.

Příklad: `mailbox.php`

```
<HTML>
<HEAD>
<TITLE>On-line čtení POP3 schránek</TITLE>
</HEAD>
<BODY>
<?

if ($jmeno=="" || $server==""):
    $akce = "login";                        // uživatel se musí přihlásit
else:
    $spojeni = @IMAP_Open("{$server/pop3}INBOX", $jmeno, $heslo);
    if (!$spojeni) $akce = "badlogin";      // špatné jméno, heslo
                                            // nebo adresa serveru
endif;

if ($akce=="login" || $akce=="badlogin"):// potřebujeme přihlašovací formulář

    if ($akce=="login")
        echo "<H1>Zadejte údaje potřebné pro připojení
                    k vaší poštovní schránce</H1>";
    else
        echo "<H1>Některý z údajů je zadán špatně
                    nebo server nefunguje</H1>";
?>

<FORM METHOD=POST>          <!-- Formulář pro přihlášení k POP serveru -->
<TABLE>
<TR><TD>Adresa POP3 serveru:
    <TD><INPUT NAME=server VALUE="<?echo $server?>">
<TR><TD>Uživatelské jméno:
    <TD><INPUT NAME=jmeno VALUE="<?echo $jmeno?>">
```

```
<TR><TD>Heslo:
    <TD><INPUT NAME=heslo TYPE=Password VALUE="<?echo $heslo?>">
<TR><TH COLSPAN=2><INPUT TYPE=Submit VALUE="Otevřít schránku">
</TABLE>
<INPUT TYPE=Hidden NAME=akce VALUE=list>
</FORM>

<?
else:                                   // už jsme přihlášení

  switch ($akce):

    case "list":                    // vypsání zpráv ve schránce
        echo "<FORM><INPUT TYPE=Submit VALUE=\"Zavřít schránku\"></FORM>\n";
        echo "<H1>Seznam zpráv uživatele $jmeno</H1>";
        echo "<P>Počet zpráv ve schránce: " . IMAP_Num_Msg($spojeni) . "\n";
        echo "<PRE>\n";
        $zpravy = IMAP_Headers($spojeni);
        for ($i=0; $i<Count($zpravy); $i++):
            echo "<FORM METHOD=POST STYLE=\"margin: 0px\">";
            echo "<INPUT TYPE=Hidden NAME=server VALUE=\"" . $server . "\">";
            echo "<INPUT TYPE=Hidden NAME=jmeno VALUE=\"" . $jmeno . "\">";
            echo "<INPUT TYPE=Hidden NAME=heslo VALUE=\"" . $heslo . "\">";
            echo "<INPUT TYPE=Hidden NAME=akce VALUE=\"show\">";
            echo "<INPUT TYPE=Hidden NAME=cislo VALUE=\"" . ($i+1) . "\">";
            echo "<INPUT TYPE=Submit VALUE=\"Zobraz\"
                         STYLE=\"font-size: 6px; padding: 0px\">  ";
            echo $zpravy[$i] . "</FORM>\n";
        endfor;
        echo "</PRE>\n";
        break;

    case "show":                    // zobrazení zprávy
        echo "<TABLE><TR>";
        echo "<TD><FORM><INPUT TYPE=Submit VALUE=\"Zavřít schránku\"></FORM>";
        echo "<TD><FORM METHOD=POST>";
        echo "<INPUT TYPE=Hidden NAME=server VALUE=\"" . $server . "\">";
        echo "<INPUT TYPE=Hidden NAME=jmeno VALUE=\"" . $jmeno . "\">";
        echo "<INPUT TYPE=Hidden NAME=heslo VALUE=\"" . $heslo. "\">";
        echo "<INPUT TYPE=Hidden NAME=akce VALUE=\"list\">";
        echo "<INPUT TYPE=Submit VALUE=\"Seznam zpráv\">";
        echo "</FORM></TABLE>\n";
```

```
        echo "<H1>Zpráva číslo " . $cislo . "</H1>\n";
        echo "<PRE>\n";
        echo IMAP_FetchHeader($spojeni, $cislo);
        echo IMAP_Body($spojeni, $cislo);
        echo "</PRE>\n";
        break;

  endswitch;

endif;

if ($spojeni) IMAP_Close($spojeni);    // uzavření spojení s POP serverem

?>
</BODY>
</HTML>
```

Podívejme se nyní podrobněji na činnost celého skriptu. Aby skript mohl správně pracovat, musí mít k neustále k dispozici adresu serveru, uživatelské jméno a heslo. Tyto informace (zejména heslo) by neměly být přenášeny tak, aby byly viditelné např. v URL adrese stránky. Pro jejich přenos proto využíváme formulář odesílaný metodou POST. Všechny přechody mezi stránkami (např. přechod ze seznamu zpráv na zobrazení zprávy) proto musíme realizovat pomocí tlačítek pro odeslání formuláře — formulář pak ve skrytých polích obsahuje všechny potřebné informace. Celý problém můžeme mnohem elegantněji vyřešit například použitím knihovny PHPLIB (viz strana 448).

Celá aplikace je uložena v jednom skriptu, jehož chování ovládáme pomocí parametru `akce`. Pokud nemáme k dispozici údaje o uživateli a jeho poštovním serveru, vyžádáme si je. Po jejich zadání je `akce` automaticky nastavena na hodnotu `list`, která vyvolá vypsání obsahu všech zpráv ve schránce. U každé zprávy je přitom tlačítko, které slouží k jejímu zobrazení (viz obr. 7-11 na straně 222).

Zobrazení zprávy probíhá v režimu `show`, kdy zároveň pomocí parametru `cislo` předáváme číslo zprávy, kterou chceme zobrazit.

Našeho klienta můžeme v mnoha směrech vylepšovat. Funkce `IMAP_Body()` vrací celé tělo zprávy. Pokud však zpráva obsahuje například soubory připojené jako přílohy, uvidíme je v prohlížeči pouze jako změť podivných znaků a čísel. K dispozici je však funkce `IMAP_FetchStructure()`, která nám vrátí kompletní informace o struktuře zprávy — jaké má části, jaký typ dat obsahují apod. Jednotlivé části zprávy pak můžeme číst pomocí funkce `IMAP_FetchBody()`. Narozdíl od funkce `IMAP_Body()` můžeme použít třetí parametr, který určuje požadovanou část zprávy.

Pokud má tedy třetí dopis ve schránce dvě části — text dopisu a soubor jako přílohu, můžeme jednotlivé části získat pomocí

```
$text = IMAP_FetchBody($spojeni, 3, "1");
$soubor = IMAP_FetchBody($spojeni, 3, "2");
```

Většinou se ještě musíme postarat o rozkódování dat — k dispozici máme opět funkce, které zjistí typ dat a provedou dekódování.

Není tedy problém vyvinout webového klienta, který kromě textových částí dopisu zobrazí i obrázky, zprávy ve formátu HTML a u připojených souborů umožní jejich uložení do souboru.

Odesílání MIME-dopisů

Odeslání jednoduchého e-mailu není díky funkci `Mail()` žádný problém. Problém však nastane, pokud chceme v dopise posílat různé přílohy jako soubory, obrázky apod. Přenos několika různých různých entit v jednom mailu umožňuje MIME. Ruční vytvoření zprávy, která odpovídá specifikaci MIME je však poměrně náročná činnost. V PHP máme k dispozici nedokumentovanou funkci[3] `IMAP_Mail_Compose()`, která umožňuje vytvoření zprávy ve formátu MIME.

Funkce má dva parametry. První obsahuje pole s definicí hlaviček společných pro celou zprávu. Druhá část obsahuje pole, které podrobně specifikuje jednotlivé části dopisu. Činnost funkce je zřejmá z následující ukázky:

```
// Hlavička dopisu
$envelope["from"] = "php@nekde.cz";         // odesílatel dopisu
$envelope["to"] = "jkj@ucw.cz";             // příjemce dopisu
$envelope["cc"] = "smirak@hotmail.com";     // kopie dopisu

// První část dopisu -- zapouzdření ostatních
$part1["type"] = TYPEMULTIPART;             // dopis má více částí
$part1["subtype"] = "mixed";                // s různým typem dat

// Nyní do dopisu přidáme soubor /tmp/obrazek.gif jako přílohu
$filename = "/tmp/obrazek.gif";             // jméno souboru
$fp = FOpen($filename, "r");                // otevření souboru
$contents = FRead($fp, FileSize($filename));// načtení souboru do proměnné
FClose($fp);                                // uzavření souboru

// Druhá část dopisu -- soubor jako příloha
```

[3] Funkce zatím nemusí správně pracovat pro všechny možné vstupní kombinace dat, a proto není nijak veřejně ohlašována.

```
$part2["type"] = TYPEAPPLICATION;           // typ dat je obecně neznámý
$part2["encoding"] = ENCBINARY;             // jde o binární data
$part2["subtype"] = "octet-stream";         // typ application/octet-stream
$part2["description"] = BaseName($filename);// jméno souboru jako popis
$part2["contents.data"] = $contents;        // obsah souboru

// Třetí část je obyčejný text
$part3["type"] = TYPETEXT;                  // typ dat je obyčejný text
$part3["subtype"] = "plain";                // obyčejný text
$part3["description"] = "Textova zprava";   // popis části zprávy
$part3["contents.data"] = "Ahoj Jirko,\n\nzkousim novy skript v PHP\n";
                                            // samotný text části

// Celý dopis si přerovnáme do jednoho pole
$body[1]=$part1;                            // první část
$body[2]=$part2;                            // druhá část
$body[3]=$part3;                            // třetí část

// V proměnné $message vytvoříme zprávu
$message = IMAP_Mail_Compose($envelope, $body);
```

V proměnné `$message` máme nyní uložen dopis přesně podle specifikace MIME. K jeho odeslání můžeme použít funkci `Mail()`. Jediný problém je v tom, že funkci musíme zvlášť předat hlavičky a tělo dopisu. V proměnné `$message` máme však obojí uloženo dohromady. Musíme proto získat zvlášť hlavičky a tělo dopisu. Využijeme toho, že hlavičky jsou od těla dopisu odděleny prázdnou řádkou.

```
// Pozici konce hlaviček uložíme do proměnné $pos
$pos = StrPos($message, "\r\n\r\n");

// Získáme hlavičku včetně posledního konce řádky
$header = SubStr($message, 0, $pos+2);

// Obdobně získáme i tělo dopisu
$body = SubStr($message, $pos, StrLen($message) - $pos);
```

Při volání funkce `Mail()` již nemusíme nastavovat adresáta dopisu, protože je obsažen ve vygenerovaných hlavičkách. Musíme však nastavit předmět dopisu — s ním si zatím β-verze `IMAP_Mail_Compose()` neporadí.

```
// Zaslání MIME-zprávy
Mail("", "Předmět dopisu je zcela výstižný", $body, $header);
```

7.5 Udělejme si vlastní Seznam

Mezi uživateli jsou velice populární seznamy odkazů na zajímavé zdroje Internetu. Mezi nejznámější servery nabízející tyto služby patří patrně asi celosvětové Yahoo nebo český Seznam.

My si ukážeme, jak navrhnout vhodnou strukturu databáze pro uložení odkazů a jak vytvořit skript, který bude obsah databáze prezentovat uživatelům. Uvidíme, že to není až tak těžké — největším problémem tedy zůstane naplnit databázi odkazy na zajímavé stránky.

Než se pustíme do samotné tvorby aplikace, musíme si přesně ujasnit požadavky. U každého odkazu budeme chtít ukládat jeho název, URL-adresu a popis. Odkazy půjde členit do kategorií. Počet úrovní kategorií přitom nebude nijak omezen. Navíc budeme chtít, aby mohl jeden odkaz patřit do několika kategorií zároveň.

Odkazy včetně kategorií a vztahů mezi nimi budeme ukládat do databáze. Možností, jak data uložit je mnoho, my si vybereme následující způsob. Samotné odkazy a kategorie budeme ukládat do tabulky `Links`. Každý záznam zde bude mít své jednoznačné identifikační číslo `ID`. Odkazy od kategorií odlišíme pomocí atributu `Type`, který bude pro odkazy obsahovat písmeno `L` (link — odkaz) a pro kategorie písmeno `F` (folder — složka).

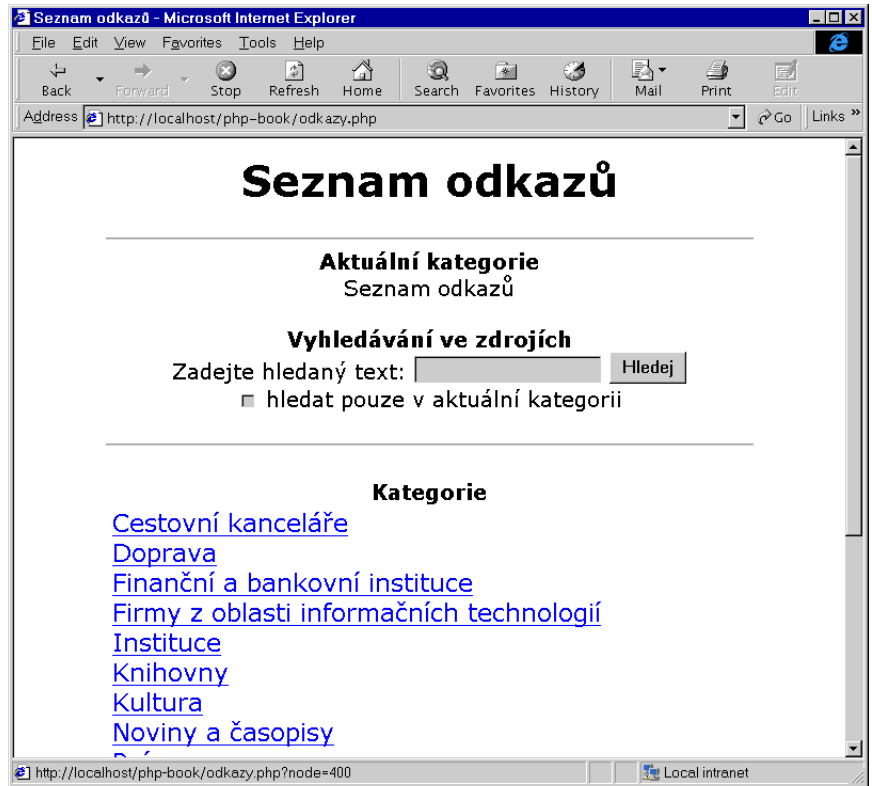
Příslušenství odkazů do jednotlivých kategorií uložíme do tabulky `Tree`. Tato tabulka bude obsahovat pro každý odkaz (nebo kategorii) patřící do nějaké kategorie jeden záznam. Tento záznam bude obsahovat `ID` odkazu (nebo kategorie) a `ID` kategorie, do které daný odkaz (nebo kategorie) patří. V tabulce links ještě vytvoříme fiktivní kategorii nejvyšší úrovně s `ID` 0. Tato kategorie bude rodičem všech odkazů a kategorií na nejvyšší úrovni.

Vytvoříme si tedy například databázi `Links` a v ní naše dvě tabulky, pomocí následujících SQL-příkazů:

```
CREATE TABLE Links (
    ID int NOT NULL,
    Type char (1) NULL,
    Name varchar (255) NULL,
    URL varchar (255) NULL,
    Description text NULL,
    PRIMARY KEY (ID)
)

CREATE TABLE Tree (
    ID int NOT NULL,
    Parent int NOT NULL,
    PRIMARY KEY (ID, Parent),
```

```
    FOREIGN KEY (ID) REFERENCES Links,
    FOREIGN KEY (Parent) REFERENCES Links
)
```

Typ `text` u atributu `Description` (popis odkazu) je závislý na použité databázi. Některé databáze používají typ `blob`. V obou případech jde o typ, který umožní do položky tabulky uložit text delší než 255 znaků, což je omezení typů `char` a `varchar`.



Obr. 7-13: Seznam odkazů — přehled kategorií

Abychom správně pochopili způsob ukládání informací do tabulek, předpokládejme, že náš systém obsahuje pouze dvě kategorie — „Vyhledávače“ a „Seznamy odkazů“. Každá kategorie obsahuje dva odkazy. Kategorie „Vyhledávače“ obsahuje odkazy na Sherlocka a Atlas. Kategorie „Seznamy odkazů“ pak obsahuje odkazy na Seznam a Atlas. Celou situaci můžeme do našich tabulek zachytit takto:

```
Tabulka: Links

ID   Type  Name            URL                       Description
---  ----  --------------  ----------------------    -------------------------
0    F     Náš Seznam
100  F     Vyhledávače
200  F     Seznamy odkazů
101  L     Sherlock        http://www.sherlock.cz   Vyhledávač pro východ...
102  L     Atlas           http://www.atlas.cz      Vyhledávač kombinovaný...
201  L     Seznam          http://www.seznam.cz     Největší seznam odkazů...


Tabulka: Tree

ID   Parent
---  ------
100  0
200  0
101  100
102  100
201  200
102  200
```

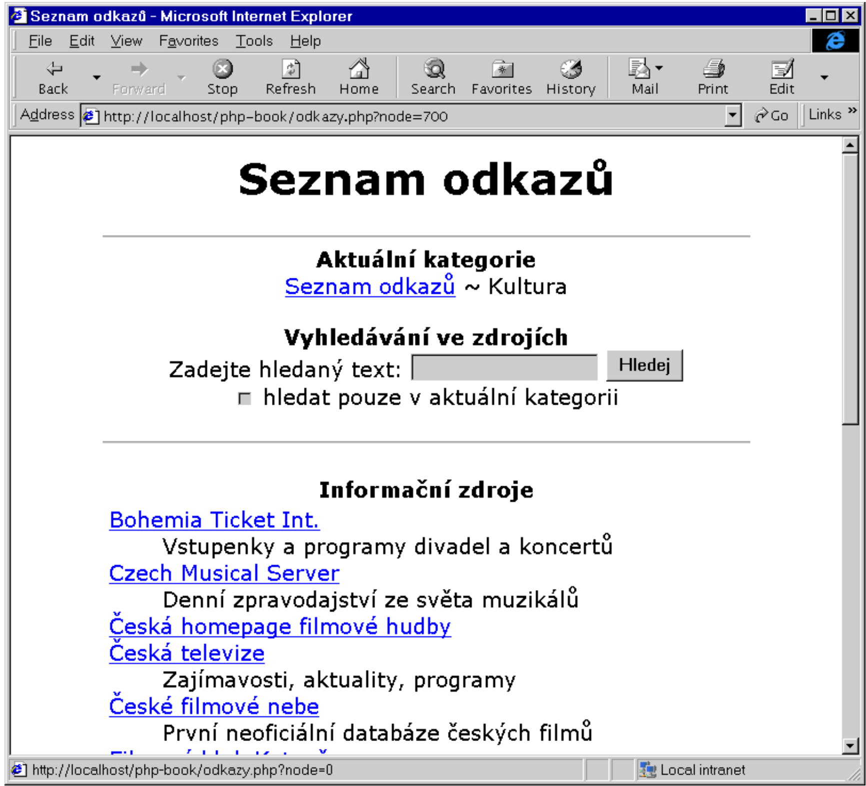
Uvedený způsob uložení informací nám umožní velice snadno provést výběr všech odkazů náležejících do určité kategorie. Pokud například chceme vybrat všechny odkazy, které patří do kategorie s číslem 200, použijeme následující SQL-příkaz:

```
SELECT Links.ID, Type, Name, URL, Description
FROM Links INNER JOIN Tree ON Links.ID = Tree.ID
WHERE Tree.Parent = 200
ORDER BY Type, Name
```

Pro některé z nás neznámá klauzule `INNER JOIN` slouží k propojení dvou tabulek. Jedná se o přehlednější způsob klasického zápisu

```
SELECT Links.ID, Type, Name, URL, Description
FROM Links, Tree
WHERE (Tree.Parent = 200) AND (Links.ID = Tree.ID)
ORDER BY Type, Name
```

Pomocí `ORDER BY` uspořádáme výsledek tak, že nejprve obsahuje kategorie (`Type = 'F'`) a teprve za nimi odkazy (`Type = 'L'`) — odpovídá to pořadí, ve



Obr. 7-14: Seznam odkazů — odkazy v jedné kategorii

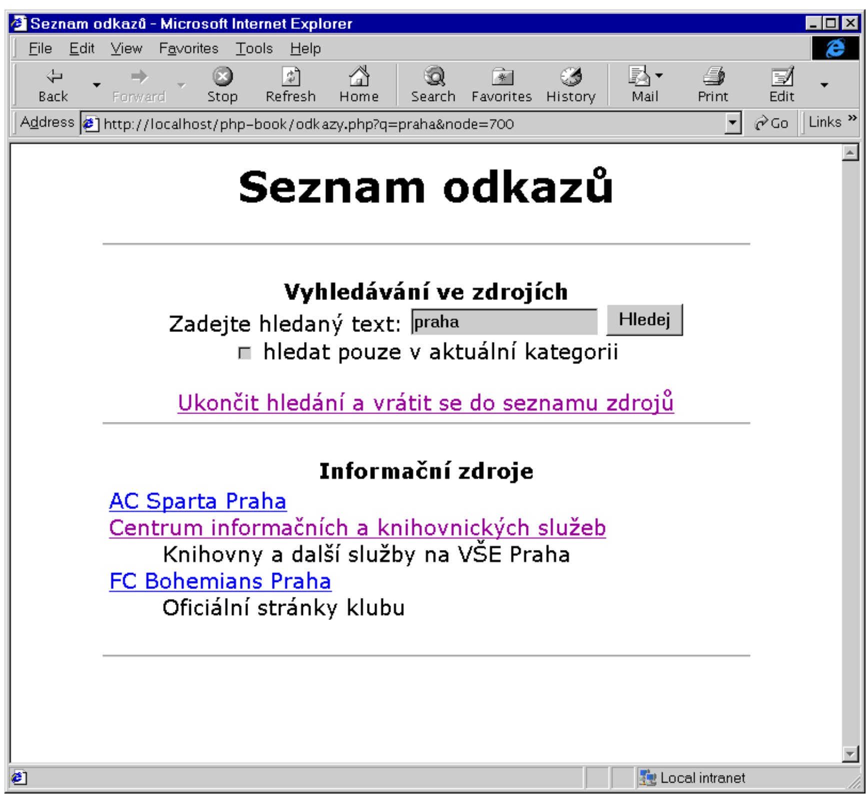
kterém chceme obsah dané kategorie zobrazit. Kategorie i odkazy jsou samozřejmě seřazeny podle svého názvu (položka `Name`). Nyní nic nebrání tomu je vypsat.

Celý skript `odkazy.php`, který zajišťuje uživatelský přístup k seznamu odkazů, si neustále předává číslo aktuální kategorie v proměnné `$node`.

Kromě výše zmíněných vlastností umožňuje skript i celou databázi odkazů prohledávat. Navíc vždy zobrazí všechny nadřazené kategorie, aby po nich umožnil uživateli snadný pohyb.

Příklad: `odkazy.php`

```
<HTML>
<HEAD>
<TITLE>Seznam odkazů</TITLE>
</HEAD>
<BODY BGCOLOR=WHITE>
<DIV ALIGN=CENTER>
```



Obr. 7-15: Seznam odkazů — výsledek prohledávání

```
<H1>Seznam odkazů</H1>
<HR WIDTH="80%">

<?
// zjištění node
$node = $node + 0;           // konverze na číslo
if ($node=="") $node = 0;

// připojení k databázi
$conn = ODBC_Connect("Links", "guest", "");
if (!$conn)
    echo "Nepodařilo se připojit k databázi.";

$q = StripSlashes($q);
if ($q==""):                 // nehledáme v odkazech
```

```
    // zkontrolujeme, zda zadané node vůbec existuje
    $result = ODBC_Exec($conn,
                "SELECT Count(*) AS 'Count' FROM Tree WHERE Parent = $node");
    ODBC_Fetch_Row($result);
    if (ODBC_Result($result, "Count")==0) $node = 0;

    // vytvoření heirarchie
    $path = "";
    $parent = $node;

    do {

        $result = ODBC_Exec($conn,
                    "SELECT ID, Name FROM Links WHERE ID = $parent");
        ODBC_Fetch_Row($result);
        if ($parent==$node)
            $path = ODBC_Result($result, "Name");
        else
            $path = "<A HREF=\"$SCRIPT_NAME?node=$parent\">".
                     ODBC_Result($result, "Name"). "</A> ~ ". $path;
        $result = ODBC_Exec($conn,
                    "SELECT Parent FROM Tree WHERE ID = $parent");
        if (ODBC_Fetch_Row($result))
            $parent = ODBC_Result($result, "Parent");
        else
            break;

    } while (true);

    echo "<STRONG>Aktuální kategorie</STRONG><BR>\n";
    echo $path."\n";

endif;

?>

<FORM ACTION="odkazy.php">
<STRONG>Vyhledávání ve zdrojích</STRONG><BR>
Zadejte hledaný text:
<INPUT NAME=q VALUE="<?echo $q?>">
<INPUT TYPE=Submit VALUE="Hledej"><BR>
<INPUT TYPE=CHECKBOX NAME=qtarget
```

```
       VALUE=local<?echo ($qtarget=="local")?" CHECKED":""?>>
hledat pouze v aktuální kategorii
<INPUT TYPE=HIDDEN NAME=node VALUE="<?echo $node?>">
</FORM>

<?

if ($q!=""):    // hledání

    echo "<P><A HREF=\"$SCRIPT_NAME?node=$node\">Ukončit hledání
                a vrátit se do seznamu zdrojů</A>\n";
    $qq = EReg_Replace("\'", "''", $q);

    if ($qtarget=="local")
        $query = "SELECT Links.ID, Type, Name, URL, Description
                  FROM Links INNER JOIN Tree ON Links.ID = Tree.ID
                  WHERE ((Name LIKE '%$qq%') OR (Description LIKE '%$qq%'))
                        AND Tree.Parent = $node
                  ORDER BY Type, Name";
    else
        $query = "SELECT ID, Type, Name, URL, Description
                  FROM Links
                  WHERE (Name LIKE '%$qq%') OR (Description LIKE '%$qq%')
                  ORDER BY Type, Name";

    $result = ODBC_Exec($conn, $query );

else:           // všechny odkazy

    $result = ODBC_Exec($conn,
                "SELECT Links.ID, Type, Name, URL, Description
                 FROM Links INNER JOIN Tree ON Links.ID = Tree.ID
                 WHERE Tree.Parent = $node
                 ORDER BY Type, Name");

endif;

$currentType = "";

while(ODBC_Fetch_Row($result)):

    $Type = ODBC_Result($result, "Type");
```

```
    if ($currentType != $Type):
        if ($Type=="F"):            // začátek skupin
            echo "<HR WIDTH=\"80%\">\n";
            echo "<P><STRONG>Kategorie</STRONG>\n";
            echo "<TABLE CLASS=tiny WIDTH=600><TR><TD>\n";
            echo "<DL>\n";
        elseif ($Type=="L"):        // začátek odkazů
            if ($currentType=="F") echo "</DL></TD></TR></TABLE>\n";
            echo "<HR WIDTH=\"80%\">\n";
            echo "<P><STRONG>Informační zdroje</STRONG>\n";
            echo "<TABLE CLASS=tiny WIDTH=600><TR><TD>\n";
            echo "<DL>\n";
        endif;
        $currentType = $Type;
    endif;

    $ID = ODBC_Result($result, "ID");
    $Name = ODBC_Result($result, "Name");
    $URL = ODBC_Result($result, "URL");
    $Description = ODBC_Result($result, "Description");

    switch ($Type):

        case "F":       // zobrazení odkazů do dalších podkategorií
            echo "<DT><BIG><A HREF=\"$SCRIPT_NAME?node=$ID\">$Name</A></BIG>\n";
            echo "<DD>$Description\n";
            break;

        case "L":       // zobrazení odkazů
            echo "<DT><A HREF=\"$URL\">$Name</A>\n";
            echo "<DD>$Description\n";
            break;

    endswitch;

endwhile;

if ($currentType!="") echo "</DL></TD></TR></TABLE>\n";
echo "<HR WIDTH=\"80%\">\n";

if ($q==""):            // nehledáme-li v odkazech, zobrazíme aktuální kategorii
    echo "<STRONG>Aktuální kategorie</STRONG><BR>\n";
```

```
    echo $path."\n";
    echo "<HR WIDTH=\"80%\">\n";
endif;

?>

</DIV>
</BODY>
</HTML>
```

Vidíme, že skript, který nám pohodlně zpřístupní i velmi rozsáhlý seznam odkazů, je poměrně jednoduchý. Můžeme jej doplnit o další skripty, které umožní pohodlnou správu odkazů uložených v databázi. Rovněž můžeme vytvořit skripty, které budou automaticky ověřovat existenci odkazů, a pokud nějaký odkaz přestane fungovat, upozorní na to správce systému.

Pokud bude námi vytvořený seznam odkazů navštěvovat mnoho uživatelů zároveň, začneme mít problémy s výkonem serveru — pro každou stránku provádí skript několik dotazů do SQL-databáze. V těchto případech se obvykle z obsahu databáze vygenerují statické stránky, které obsahují celou hierarchii odkazů. Skripty a přístup k databázi se použije pouze v případech, kdy v seznamu odkazů hledáme nějaký text. Statické stránky s odkazy generujeme podle potřeby například jednou denně nebo týdně.

7.6 On-line demokracie aneb hlasování

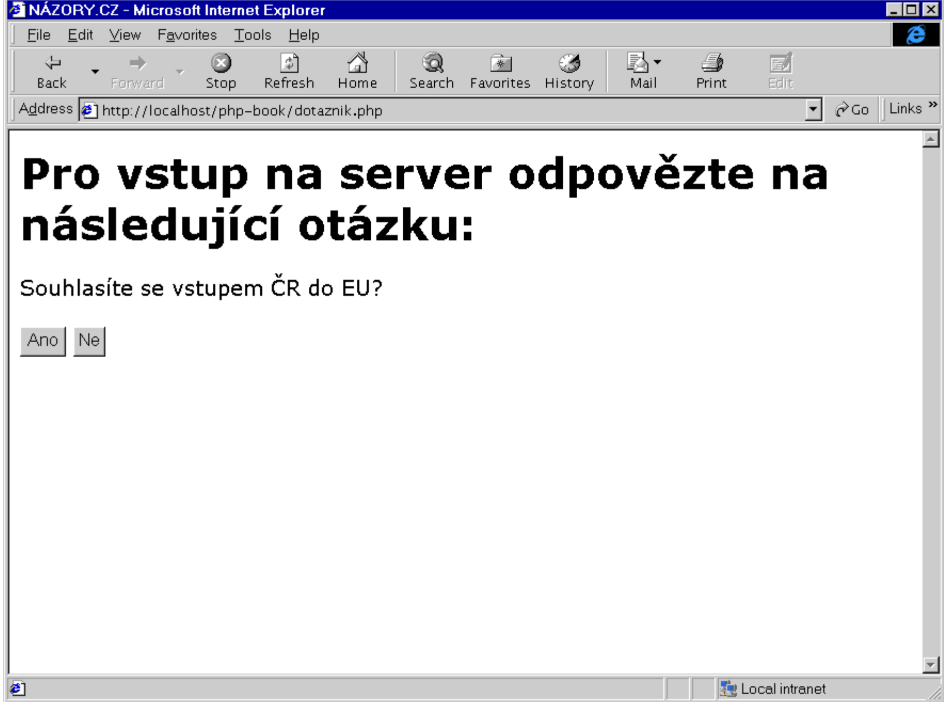
Častým prvkem mnoha serverů, zvláště zpravodajských, bývají různé ankety, které zjišťují názor návštěvníků na danou problematiku. Hlasování je většinou čistě dobrovolné. My nyní vyřešíme trošku odlišný problém. Vytvoříme skript, který přístup k nějaké stránce umožní až po zodpovězení otázky. Skript přitom bude nabízet několik otázek a bude se snažit o to, aby jednomu člověku nepokládal tutéž otázku vícekrát. Pro ukládání již zodpovězených otázek s výhodou použijeme cookies (více se o nich dočtete v desáté kapitole).

Jednotlivé otázky si uložíme do databáze `Dotaznik` do tabulky `Dotaznik`. U každé otázky máme uloženo její identifikační číslo, znění a počet hlasů pro a proti. K vytvoření tabulky proto můžeme použít SQL-příkaz:

```
CREATE TABLE Dotaznik (
    ID int NOT NULL,
    Otazka varchar(255),
    Ano int,
    Ne int,
    PRIMARY KEY (ID)
```

```
)
```

V řešení našeho úkolu nám výborně pomohou cookies, protože právě pomocí nich si budeme u každého uživatele evidovat, na které otázky již odpověděl. Z databáze při přihlášení k serveru vybereme vždy dosud nepokládanou otázku — to zamezí zbytečnému zkreslení výsledků tím, že někdo vícekrát odpoví na jednu otázku. (Zamezení však nebude 100% — pokud má někdo cookies ve svém prohlížeči vypnuté, dostane jednu otázku vícekrát.)



Obr. 7-16: Položení dotazu uživateli

Protože pro každého uživatele potřebujeme evidovat větší počet zodpovězených otázek, využijeme toho, že cookie může mít libovolné jméno — můžeme vytvářet cookies, z kterých po načtení do PHP vznikne pole. My si cookie pojmenujeme `zodpovezeneDotazy`. Po zodpovězení otázky nastavíme příslušným způsobem cookie. Pokud například uživatel odpoví na druhou otázku, nastavíme cookie `zodpovezeneDotazy[2]` na hodnotu identifikátoru otázky, tj. opět 2.

Následující skript z tabulky vybere první dosud nepoložený dotaz a zobrazí jej:

Příklad: `dotaznik.php`

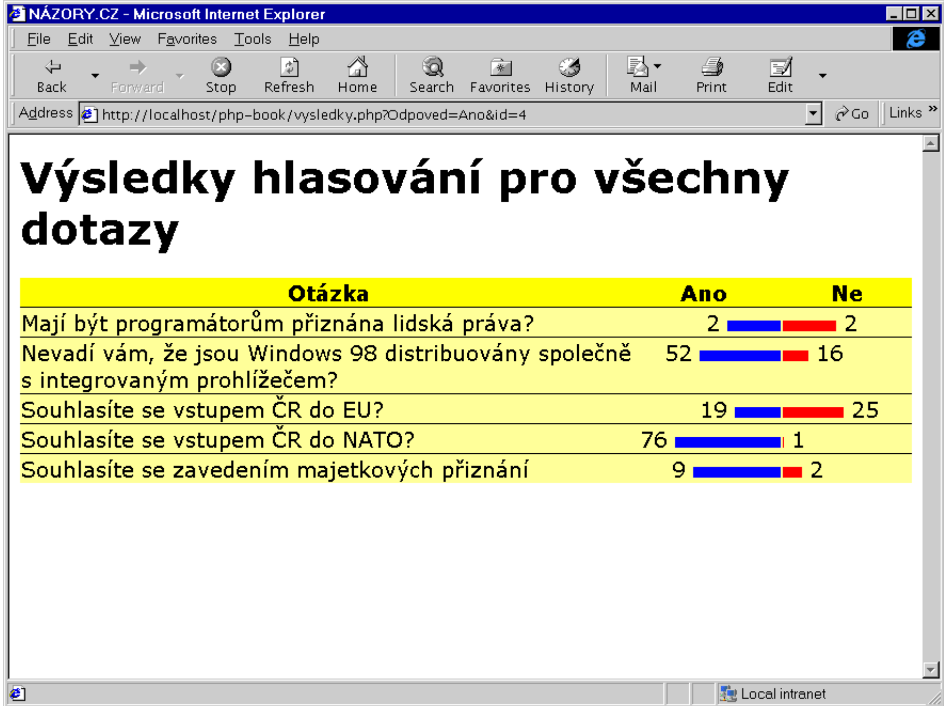
```
<HTML>
<HEAD>
<TITLE>NÁZORY.CZ</TITLE>
</HEAD>
<BODY>
<?
    $sql = "SELECT * FROM Dotaznik";        // vytvoření SQL-dotazu
    if (IsSet($zodpovezeneDotazy))
        $sql .= " WHERE ID NOT IN (".
                Implode($zodpovezeneDotazy, ",").
                ")";

    $spojeni = ODBC_Connect("Dotaznik", "guest", "");
    $vysledek = ODBC_Exec($spojeni, $sql);
    if (ODBC_Fetch_Row($vysledek)):         // přečtení první otázky z výsledku
        $id = ODBC_Result($vysledek, "ID");
        $dotaz = ODBC_Result($vysledek, "Otazka");
    endif;

  if (IsSet($id)): ?>                       // pokud máme otázku

  <H1>Pro vstup na server odpovězte na následující otázku:</H1>
  <FORM ACTION="vysledky.php">
  <?echo $dotaz?><BR><BR>
  <INPUT TYPE=Submit NAME=Odpoved VALUE="Ano">
  <INPUT TYPE=Submit NAME=Odpoved VALUE="Ne">
  <INPUT TYPE=Hidden NAME=id VALUE="<?echo $id?>">
  </FORM>

<?  else:   ?>                              // pokud nemáme otázku

  <H1>Dnes to bude bez otázky</H1>
  <A HREF="vysledky.php">Vstupte na náš server</A>

<?  endif   ?>
</BODY>
</HTML>
```

Zajímavé je použití funkce `Implode()`, která vezme jednotlivé prvky pole a navzájem je spojí do jednoho řetězce — k oddělení prvků pole v řetězci se používá znak předaný jako druhý parametr. Voláním

```
Implode($zodpovezeneDotazy, ",")
```

tedy získáme seznam identifikačních čísel již zodpovězených dotazů, který s výhodou použijeme při zadávání SQL-dotazu, který z tabulky `Dotazy` vybírá pouze dosud nepoložené otázky.

Nyní zbývá vytvořit skript `vysledky.php`, který zpracuje odpověď uživatele a případně zobrazí nějaké další informace — my se omezíme pouze na zobrazení výsledků hlasování.



Obr. 7-17: Výsledky ankety

Skript `vysledky.php` má na starosti mnoho věcí — předně musí klientovi odeslat cookie, která obsahuje číslo zodpovězené otázky. Poté musí v databázi aktualizovat počet odpovědí pro/proti u dané otázky. Nakonec skript vypíše přehled odpovědí na všechny otázky, abychom měli přehled.

Příklad: `vysledky.php`

```
<?
    if(IsSet($id))
        SetCookie("zodpovezeneDotazy[$id]", $id, Time()+2592000);
?>
<HTML>
<HEAD>
<TITLE>NÁZORY.CZ</TITLE>
</HEAD>
<BODY>
<H1>Výsledky hlasování pro všechny dotazy</H1>
<TABLE CELLSPACING=0 BGCOLOR=BLACK>
<TR BGCOLOR=YELLOW><TH>Otázka<TH WIDTH=120>Ano<TH WIDTH=120>Ne</TR>
<?
    $spojeni = ODBC_Connect("Dotaznik", "guest", "");
    $sql = "UPDATE Dotaznik";       // vytvoření SQL-dotazu
    if ($Odpoved=="Ano")
        $sql .= " SET Ano = Ano + 1";
    else
        $sql .= " SET Ne = Ne + 1";
    $sql .= " WHERE ID = $id";
    $vysledek = ODBC_Exec($spojeni, $sql);  // přidání hlasu
    if (!$vysledek)
        echo "Nepodařilo se zapsat vaši odpověď.";

    // výběr a zobrazení výsledků
    $sql = "SELECT * FROM Dotaznik ORDER BY Otazka";
    $vysledek = ODBC_Exec($spojeni, $sql);
    define("SirkaGrafu", 100);

    while (ODBC_Fetch_Row($vysledek)):      // pro všechny otázky
        $ano = ODBC_Result($vysledek, "Ano");
        $ne = ODBC_Result($vysledek, "Ne");
        echo "<TR><TD COLSPAN=3></TR><TR VALIGN=TOP BGCOLOR='#FFFF80'><TD>" .
             ODBC_Result($vysledek, "Otazka") . "<TD ALIGN=RIGHT>$ano " .
             ($ano ?
                "<IMG SRC=bluedot.gif WIDTH=" .
                      Round(SirkaGrafu/($ano+$ne)*$ano) .
                      " HEIGHT=10>" : "") .
             "<TD ALIGN=LEFT>" .
            ($ne ?
```

```
                "<IMG SRC=reddot.gif WIDTH=" .
                      Round(SirkaGrafu/($ano+$ne)*$ne) .
                      " HEIGHT=10>" : "")." $ne" .
            "</TR>";
    endwhile;
?>
</TABLE>
</BODY>
</HTML>
```

U odesílané cookie jsme nastavili platnost na 30 dní — nepředpokládáme, že jeden dotaz bude na serveru delší dobu a je zbytečné, aby byl prohlížeč uživatele zahlcen nepotřebnými cookies. V hranatých závorkách za názvem cookie uvádíme jedinečný index — PHP nám následně umožní s cookie pracovat jako s polem (to využíváme v prvním skriptu).

8. Bezpečné aplikace

Možnost vzájemné komunikace počítačů v Internetu přináší mnoho nových a užitečných způsobů rychlého vyměňování a sdílení informací. Zároveň toto prostředí přináší jistá bezpečnostní rizika. Počítačoví zločinci se mohou snažit získat zdrojové kódy vašich pracně napsaných aplikací, získat tajná data nebo neoprávněně s vašimi daty manipulovat. V této kapitole se zaměříme na čtyři oblasti, které úzce souvisejí s bezpečností. První část kapitoly bude věnovaná způsobu ochrany našich zdrojových kódů před nepovolanými zraky. Další část se bude věnovat způsobům, jak přístup ke skriptu povolit pouze některým uživatelům. Ve třetí části se budeme zabývat ochranou dat před odposlechem při jejich cestě komunikačními linkami Internetu. Poslední část rozebere možnosti PHP při omezování práv, které mají PHP skripty k dispozici.

Téma bezpečnosti na Internetu je velice obsáhlé. Pokud to tedy s bezpečností vašich aplikací a dat myslíte opravdu vážně, měli byste prostudovat speciální literaturu věnovanou obecně bezpečnosti informačních systémů a dále knihy popisující bezpečnost operačního systému, na kterém běží vaše aplikace a WWW-server.

8.1 Ochrana skriptů před nepovolanými zraky

Proč je vůbec potřeba chránit zdrojové texty skriptů před ostatními uživateli? Hlavní důvody jsou dva. Prvním z nich je přítomnost některých důležitých informací přímo ve zdrojových textech. V kapitole věnované databázím jsme například viděli, že do skriptu se přímo zapisuje uživatelské jméno a heslo, pod kterým se přihlašujeme k databázi. Kdokoliv, kdo by získal přístup ke zdrojovému textu tohoto skriptu, by mohl získat neoprávněný přístup k databázi.

Druhým důvodem je ochrana vlastních produktů. Pokud budete firmou, která se živí vývojem informačních systémů, které pak instaluje u různých zákazníků, nebudete si asi přát, aby vaši zákazníci nelegálně prodávali zdrojové texty aplikace vašim konkurentům.

Útok na naše zdrojové texty může přijít v podstatě ze dvou stran — z okolí počítače pomocí standardních služeb Internetu nebo přímo od nějakého uživatele počítače, na němž je aplikace instalována. Pro naše účely je útočník zvenku, který neoprávněně získá běžný přístup k našemu počítači, chápán jako druhý případ — interní útok.

Útok zvenku pomocí prohlížeče

Skripty zapsané v PHP jsou vždy uloženy v adresáři, který je WWW-serverem zpřístupněn všem uživatelům Internetu. Kdokoliv se tedy může dotazem na správné URL pokusit náš skript přečíst. Fígl však spočívá v tom, že skripty v PHP mají příponu `.php` nebo `.php3` a všechny soubory s touto příponou předá WWW-server ke zpracování interpretu PHP. Uživateli je tedy zpět zaslán pouze výsledek skriptu, který v žádném případě neobsahuje zdrojový kód skriptu.

Slabinou celého systému však může být nějaký nevhodný CGI-skript, který je umístěn někde na serveru. Tento CGI-skript může pracovat (ať chtěně či nechtěně) tak, že po zadání nějakého souboru jako parametru, odešle specifikovaný soubor bez jakýchkoliv úprav zpět jako odpověď. Měli bychom se proto vždy ujistit, že náš server neobsahuje žádné podezřelé skripty. Na začátku léta 1998 například světem proběhla informace o tom, že pomocí jednoho ze skriptů dodávaného s IIS lze přečíst zdrojový kód libovolného ASP-skriptu. První reakcí správců mnoha systémů bylo vyřazení serverů na nějakou dobu z provozu a kontrola všech skriptů, které byly na systému nainstalovány.

Poslední věcí, kterou nesmíme zanedbat, je správné umístění a pojmenování knihoven napsaných v PHP. Při psaní skriptů často využíváme společný kód uložený ve skriptech (knihovnách), které obsahují pouze definice užitečných funkcí a tříd. Častým zvykem je ukládání těchto knihoven do souborů s příponou `.inc`. Soubory s touto příponou však nejsou WWW-serverem interpretovány jako skript v PHP a jsou bez jakýchkoliv změn odeslány zpět uživateli. Pokud tedy někdo náhodou uhodne URL, které ukazuje přímo k nějaké knihovně, má vyhráno. Tím spíše, že právě knihovny často obsahují funkce pro připojení k databázi, kde často nechybějí přístupová jména a hesla. Ochrana před tímto typem útoku je jednoduchá. Buď musíme knihovny umístit do adresáře, který není součástí stromu dokumentů WWW-serveru, nebo knihovny umístit do souborů s příponou `.php` či `.php3` tak, aby tyto soubory byly vždy před odesláním interpretovány systémem PHP.

Pod svícnem je tma

Zdaleka však není pravda, že útok na skripty je veden jen z vnějšku. Mnohem větší prostředky pro neoprávněný přístup k našim skriptům nabízí operační systém, na kterém běží WWW-server a PHP. Pokud je tento systém dostupný více uživatelům, je potřeba skripty ochránit před nechtěnými zraky.

Tento problém je asi mnohem palčivější na Unixu než na Windows NT. To vyplývá ze skutečnosti, že narozdíl od Windows NT je Unix víceuživatelský. Pomocí terminálů, telnetu nebo ssh může být k jednomu unixovému stroji připojeno více uživatelů. Typicky tato situace nastává v případech, kdy s PHP pracujeme na univerzitních počítačích nebo na počítačích poskytovatelů připojení, kde jeden počítač obsluhuje několik virtuálních domén.

Kořen problému vězí v principu přístupových práv, která používá Unix. Většina WWW-serverů na Unixu běží jako uživatel `nobody` nebo `httpd`. Aby tito uživatelé mohli číst HTML-stránky a PHP-skripty v adresářích uživatelů, musí být u těchto souborů povolen přístup pro čtení všem uživatelům. Implicitně tedy mohou skripty číst všichni uživatelé systému.

Pomoc je naštěstí poměrně jednoduchá. Na většině unixových instalací jsou všichni uživatelé členy skupiny `users`. Přístupová práva pro skupiny mají vyšší prioritu než přístupová práva pro ostatní. Pokud tedy u našich skriptů odepřeme přístup skupině `users`, skripty budou čitelné pouze pro nás jako pro vlastníka, pro WWW-server a pro administrátora `root`. Změnu přístupových práv můžeme provést pomocí příkazu

```
chmod 604 «jméno skriptu»
```

Pokud vytváříme více skriptů, můžeme si pomocí příkazu `umask` nastavit odpovídající přístupová práva pro všechny nově vytvářené soubory.

Ve Windows NT je situace o něco lepší. Ke skriptům máme obvykle přístup pouze my a speciální uživatel, pod kterým je spouštěn WWW-server. Tento uživatel má obvykle uživatelské jméno `IUSR_XXX`, kde `XXX` je jméno počítače. Není však od věci přístupová práva kontrolovat a přístup znemožnit všem nepovolaným uživatelům a skupinám. Ve Windows NT dokonce můžeme přístup zakázat i administrátorovi systému. Ten pak přístup může získat jedině tak, že převezme vlastnictví. Tím sice získá přístup k souboru, ale my to můžeme zjistit a na nenechavého správce si došlápnout. Druhá věc je, že chytrý správce může pouze dočasně změnit konfiguraci WWW-serveru tak, aby skripty neinterpretoval, přečíst si je, a my nemáme šanci nic zjistit. To je však obecný nešvar většiny dnes používaných operačních systémů — systémový administrátor může se systémem dělat téměř cokoliv. Pryč jsou doby, kdy šlo silná administrátorská práva rozdělit více uživatelům jako např. v operačním systému VMS.

Konec předchozího odstavce nás nepotěší, pokud se živíme psaním aplikací v PHP a dodáváme je našim klientům. Jejich správce systému tak snadno získá zdrojové kódy našich pracně vytvářených aplikací. S tím bohužel nejde dnes nic dělat. Vývojový tým PHP však pracuje na rozšíření PHP, které umožní distribuci zašifrovaných skriptů, které nepůjde převést zpět do zdrojového tvaru. Toto rozšíření bude narozdíl od samotného PHP šířeno úplatně jako komerční systém.

8.2 Autentifikace uživatelů

Ne všechny aplikace, které budeme v PHP psát, musí být veřejně přístupné. Například podnikový informační systém musí být přístupný pouze zaměstnancům daného podniku. Proto bývá každému oprávněnému uživateli přiděleno uživatelské jméno a heslo, kterým se aplikaci autentifikuje — prokazuje, že má oprávnění používat aplikaci.

PHP podporuje autentifikaci obsaženou v protokolu HTTP. Než si popíšeme její použití, zmíníme se o jednodušší možnosti autentifikace.

Nechme za sebe pracovat jiné

Většina WWW-serverů má v sobě rovněž zabudovanou podporu autentifikace. Mnohem jednodušší proto bývá implementovat autentifikaci přímo prostředky WWW-serveru. Například server Apache umožňuje použití modulů mod_auth, mod_auth_db nebo mod_auth_dbm, které zajišťují autentifikaci uživatelů. Uživatelská jména a hesla jsou přitom uložena v textovém souboru nebo v databázi. Tyto moduly navíc podporují definování skupin uživatelů, což velice usnadňuje přidělování přístupových práv k aplikaci a jejím částem. Podobně i u IIS lze nastavit autentifikaci uživatelů. Slouží k tomu záložka Directory Security, kterou nalezneme v dialogovém okně pro nastavení vlastností adresáře v nástroji Microsoft Management Console. IIS autentifikuje uživatele proti již existujícím účtům na systému. To může být poněkud nevýhodné, protože někdy chceme provádět autentifikaci i jiných uživatelů než těch, kteří mají na serveru účet. Na druhou stranu umožňuje IIS ve spolupráci s Internet Explorerem zasílání hesel v zašifrované podobě i bez použití SSL (viz další sekce).

Podpora autentifikace v PHP

Podpora autentifikace pracuje v PHP pouze tehdy, pokud PHP běží jako modul WWW-serveru. V tomto případě můžeme pomocí funkce `Header()` zaslat prohlížeči HTTP-hlavičku, která si vyžádá zadání jména a hesla. Na to prohlížeč zareaguje tak, že si od uživatele vyžádá jméno a heslo. Toto jméno a heslo je vráceno zpět skriptu, který má v proměnných `$PHP_AUTH_USER`, `$PHP_AUTH_PW` a `$PHP_AUTH_TYPE` dostupné uživatelské jméno, heslo a typ autentifikace (zatím je podporována pouze autentifikace typu `basic`). Obsah proměnných může náš skript porovnat například s nějakou tabulkou, která obsahuje jména a hesla uživatelů, kteří mají k aplikaci přístup.

Pokud chceme přístup na naše stránky podmínit autentifikací, musíme na začátek každé stránky umístit kód, který vypadá zhruba následovně. Informace o uživatelích a heslech máme přitom uloženy v tabulce `Users` přístupné pomocí ODBC.

```
<?
    if (!IsSet($PHP_AUTH_USER)):
        Header("HTTP/1.0 401 Unauthorized");
        Header("WWW-Authenticate: Basic realm=\"«název aplikace»\"");
        echo "Přístup k těmto stránkám je vázán na zadání jména a hesla.";
        exit;
    else:
        @$spojeni = ODBC_Connect("authdb", "authenticator", "pxQy3U");
        if (!$spojeni):
            echo "Vaše uživatelské jméno nelze ověřit.";
            exit;
        endif;
        @$vysledek = ODBC_Exec("SELECT Passwd FROM Users
                                WHERE User = '$PHP_AUTH_USER'");
        if (!$vysledek):
            echo "Vaše uživatelské jméno nelze ověřit.";
            exit;
        endif;
        if (!ODBC_Fetch_Row($vysledek)):
            echo "Neexistující uživatelské jméno.";
            exit;
        else:
            if (ODBC_Result($vysledek, "Passwd") != $PHP_AUTH_PW):
                echo "Špatně zadané heslo.";
                exit;
            endif;
        endif;
    endif;
?>
<HTML>
    «stránka přístupná pouze pro autentifikované uživatele»
</HTML>
```

Starší verze Internet Exploreru neporozuměly správně hlavičkám vyžadujícím autentifikaci. V těchto případech musíme prohodit pořadí volání funkcí `Header()`. Výsledkem je sice hlavička, která neodpovídá standardu HTTP, ale funguje i ve starších verzích IE.

Prohlížeče si během svého spuštění pamatují jméno a heslo pro danou aplikaci (určenou pomocí parametru `realm`). Uživateli tedy stačí zadat jméno a heslo jednou, dále je již automaticky posílá prohlížeč. Na to je třeba uživatele upozornit. Pokud by totiž uživatel po skončení práce s chráněnou aplikací neukončil prohlížeč, kdokoliv jiný by se k aplikaci dostal, protože by prohlížeč automaticky

odeslal zapamatované jméno a heslo. Tento problém můžeme vyřešit několika způsoby. Jedním ze způsobů je implementování odhlašovací funkce v aplikaci. Odhlašovací funkce by měla po stisku tlačítka formuláře odeslat prohlížeči zpět informaci o neautorizovaném přístupu, i když je jméno a heslo zadáno správně. To by mělo prohlížeč donutit k zapomenutí autentifikačních informací.

Další z možností je při každém přihlášení uživatele do `realm` uložit jednoznačný identifikátor, který bude platný pouze do doby, než se uživatel pomocí nějaké funkce aplikace neodhlásí.

Další z možností je `realm` v určitých časových periodách měnit. Tento přístup však nelze použít osamoceně, protože do jednoho časového okamžiku se může směstnat odchod autentifikovaného uživatele a přístup neznámého útočníka. Tato metoda však částečně řeší problém, kdy se uživatel zapomene od aplikace odhlásit. Po určité době je pak jaksi odhlášen automaticky, protože je změněn `realm` a server si přes prohlížeč vyžádá nové zadání jména a hesla. Nevýhodou tohoto řešení je, že otravuje uživatele častými požadavky na znovuzadání jména a hesla během práce s aplikací.

8.3 Šifrování přenášených dat

Data přenášená mezi serverem a prohlížečem mohou často obsahovat důvěrné informace, které by se neměly dostat do rukou někoho nepovolaného. Informace mezi serverem a prohlížečem jsou normálně posílány nezašifrované, a tak je může kdokoliv, kdo má přístup k některému počítači nebo routeru, přes který data putují, odposlouchávat. Typickým příkladem takto citlivých dat jsou uživatelská jména a hesla zasílaná při autentifikaci uživatele. Pokud posíláme tyto nebo podobné citlivé údaje, měli bychom se postarat o to, aby byl přenos dat zašifrován a nebyl tedy odposlouchávatelný.

K šifrování dat putujících mezi serverem a prohlížečem se dnes nejčastěji používá protokol SSL (Secure Sockets Layer). Protokol dnes existuje v několika verzích z nichž poslední je 3.0. Ta je ze všech nejbezpečnější. Pokud chceme, aby komunikace mezi serverem a prohlížečem probíhala pomocí SSL, musíme nejprve v konfiguraci WWW-serveru pro adresář, ve kterém jsou skripty uloženy, zapnout podporu SSL. To se dělá v každém serveru jinak. Komerční servery většinou mají k dispozici nějaký grafický konfigurační program, který nás celým procesem provede. Součástí konfigurace SSL je i získání digitálního certifikátu. Ten může vydat pouze certifikační server. Ve světě jich existuje mnoho, ale ne všechny jsou stejně důvěryhodné. Navíc se za poskytnutí certifikátu musí obvykle platit. Pokud tedy naši aplikaci budou využívat například jen naši zaměstnanci, nemusíme si certifikát pořizovat u specializované a dobře zavedené firmy, ale můžeme si sami zřídit vlastní certifikační server.

Podpora SSL existuje i pro server Apache. Jednak existuje speciální verze Apache, která v sobě SSL přímo zahrnuje — tzv. Stronghold. Bohužel Stronghold v sobě obsahuje PHP/FI 2.0 a jeho rekompilace s podporou PHP3 není úplně triviální záležitostí. Naštěstí pro Apache existuje modul mod_ssl, který využívá knihovnu SSLeay a podporu SSL doplní do běžného Apache.

Přesné instrukce pro aktivování podpory ve vašem WWW-serveru naleznete v jeho dokumentaci.

Pokud máme podporu SSL zprovozněnou na serveru, stačí prohlížeč nasměrovat na stránky, jejichž přenos je chráněn pomocí SSL. V URL, které ke stránkám směřuje, však musíme použít schéma `https` místo `http`, aby prohlížeč věděl, že má použít SSL.

8.4 Bezpečnější než sex

Jelikož PHP bylo navrženo speciálně pro potřeby generování dynamických stránek, již jeho návrh počítal s metodami, jak zvýšit bezpečnost celého systému. Na skripty jsou tedy aplikována všechna bezpečnostní omezení jako na ostatní dokumenty.

PHP jako modul

Pokud PHP běží jako modul serveru, jsou všechny spouštěné skripty nejprve kontrolovány samotným WWW-serverem. Teprve pokud server zjistí, že skript může být spuštěn, předá jej interpretu PHP.

PHP jako CGI-skript

Pokud PHP běží jako CGI-skript, je v činnosti několik mechanismů, které chrání systém před napadením.

Prvním z ochranných mechanismů zajistí, že PHP ignoruje všechny parametry předané z příkazové řádky. Tím je zamezeno útokům typu

```
http://www.server.cz/cgi-bin/php?/etc/passwd
```

protože text za otazníkem je předáván jako parametr příkazové řádky. Normálně PHP jako parametr očekává jméno souboru, který má zpracovat. Zabudovaná ochrana však zabrání předání parametrů a neoprávněnému získání obsahu souboru.

Další problémy přináší automatické přesměrování skriptů. Webový server můžeme většinou nakonfigurovat tak, že požadavek na `http://server/tajne/skript.php` se převede na volání `http://server/cgi-bin/php/tajne/skript.php`. Při zadání tohoto požadavku server kontroluje oprávněnost přístupu k souboru `/tajne/skript.php`. Pokud však zadáme rovnou cestu k PHP a jako parametr uvedeme jméno skriptu, kontrola přístupu je prováděna pouze

pro soubor /cgi-bin/php. Tímto způsobem může uživatel neoprávněně přistupovat k souborům, pro které jinak server vyžaduje autentifikaci.

Pokud chceme zamezit volání interpretu pomocí URL jako http://server/cgi-bin/php/tajne/skript.php, musíme při kompilaci PHP zapnout volbu --enable-force-cgi-redirect. V tomto případě PHP odmítne všechna svá volání, kdy URL obsahuje /cgi-bin/php, zpracována budou pouze přesměrování z adresy http://server/tajne/skript.php obsloužená WWW-serverem. Tato vlastnost nemusí fungovat na jiných serverech než Apache.

Další možnost, jak zvýšit bezpečnost serveru, je oddělit adresáře používané pro běžné dokumenty, které vyžadují autentifikaci, a pro skripty. Slouží k tomu konfigurační direktivy doc_root a user_dir.

Pokud nastavíme pouze doc_root, nebude PHP spouštět skripty z jiných adresářů než podadresářů doc_root. Tím znemožníme přístup k dokumentům uloženým v normálním stromě dokumentů WWW-serveru. Požadavek na stránku http://server/tajne/skript.php tedy ve skutečnosti vyvolá skript tajne/skript.php uložený v adresáři doc_root.

Pokud není nastavena direktiva user_dir, nejsou přesměrovávány požadavky na adresáře jednotlivých uživatelů pro ta URL, která obsahují vlnku '~'. Místo toho je hledán odpovídající podadresář (včetně vlnky na začátku) v adresáři doc_root.

Pokud nastavíme direktivu user_dir například na hodnotu public_php, je soubor pro obsloužení URL typu http://server/~jkj/skript.php hledán v podadresáři public_php domovského adresáře uživatele jkj.

Další možností, jak zvýšit bezpečnost CGI-verze PHP, je její umístění mimo strom dokumentů WWW-serveru. Nevýhodou je, že na začátek každého skriptu v PHP musíme přidat následující řádku

```
#!/usr/bin/php
```

Zároveň musíme skriptu nastavit atribut spustitelnosti, například pomocí chmod +x skript.php. Vidíme, že skripty v PHP se v tomto případě chovají jako CGI-skripty napsané v jakémkoliv jazyce např. v Perlu nebo příkazovém shellu. Aby tento režim správně pracoval, měli bychom při kompilaci PHP použít volbu --enable-discard-path.

8.5 Bezpečný režim

Pokud vás trápí paranoia, asi se vám bude hodit bezpečný režim, který nabízí PHP. Pokud se teď smějete a říkáte si, že vy a paranoia nemáte nic společného, je to chyba. Dobrý správce serveru nesmí postrádat správnou dávku paranoi, která mu pomáhá dobře zabezpečit server.

Pomocí konfigurační direktivy `safe_mode` můžeme PHP spouštět v bezpečném režimu. V tomto režimu se uplatňují následující omezení:

- *Práce se soubory* — skript může pomocí funkcí pracovat jen s těmi soubory, jejichž vlastník je shodný s vlastníkem spuštěného skriptu. Omezení se týká i příkazů pro vkládání skriptů, jako jsou `include` a `require`.
- *Spouštění externích programů* — ze skriptu mohou být spuštěny pouze programy, které se nacházejí v adresáři určeném pomocí direktivy `safe_mode_exec_dir`.
- *HTTP autentifikace* — pokud skript používá autentifikaci a odesílá HTTP-hlavičku `WWW-Authenticate`, je před `realm` doplněno číslo uživatele (UID), jenž je vlastníkem skriptu. Tím znemožníme ostatním uživatelům serveru napsání skriptu, který by zjistil přístupová jména a hesla pro ostatní aplikace běžící na stejném serveru.

Bezpečný režim je výborným pomocníkem a nezbytnou pomůckou na všech serverech, kde skripty vytváří více uživatelů.

9. Referenční přehled funkcí

Právě jste se dočetli až k největší kapitole, která má více jak 170 stran. Vzhledem k tomu, že kapitola obsahuje refenční přehled funkcí, konstant a proměnných, nikdo po vás nechce, abyste kapitolu četli celou od začátku do konce. Naleznete zde abecedně setříděný seznam funkcí i s popisem jejich parametrů a činnosti. Pokud tedy potřebujete rychle najít popis použití nějaké funkce, hledejte zde.

> Ve volném časem si zběžně přečtětě alespoň stručný popis všech funkcí. Budete mít mnohem lepší přehled o tom, co PHP umí, a zjistíte, že mnoho věcí lze dělat mnohem jednodušeji s využitím funkcí, které PHP standardně nabízí.

Podívejme se nyní na způsob, jakým je referenční přehled uspořádán. Popis každé funkce začíná jejím jménem a skupinou, do které funkce patří. Pod názvem funkce naleznete její stručný popis.

Další řádka je již informačně bohatá, protože obsahuje informace o parametrech funkce a typu dat, které vrací. Obecný formát je následující

«typ_výsledku» «jméno_funkce»(*«parametry»*)

«typ_výsledku» nám říká, jakého typu je výsledek vrácený funkcí. Funkce vrací výsledek v některém z datových typů PHP — `integer` (celé číslo), `double` (reálné číslo), `string` (textový řetězec), `array` (pole) a `object` (objekt). Pokud funkce vrací různé datové typy v závislosti na předaných parametrech, vyjádříme to pomocí slova `mixed`.

> Většina funkcí vrací hodnotu `false` v případě, že při jejich provádění došlo k chybě. Tím pádem by všechny funkce, které vracejí jiný typ než `integer` (ten se používá pro uložení logických hodnot), měly mít návratovou hodnotu typu `mixed`. Aby byl popis syntaxe užitečnější, uvádíme u funkcí typ, který vracejí v případě úspěšného průběhu.

Pokud funkce nevrací žádnou hodnotu, vyjadřujeme to pomocí speciálního slova `void`, které velice dobře znají všichni, kdo přišli do styku s programovacím jazykem C.

Posledním typem, který v popisu funkcí používáme, je typ `fptr`. Tento typ slouží k označení ukazatelů na funkci a ve skutečnosti je shodný s typem `string`. Jako hodnotu parametru však musíme uvést jméno nějaké existující funkce.

Popis *«parametrů»*, které funkce akceptuje, je velice jednoduchý. Jedná se o seznam čárkami oddělených definic parametrů. Každá definice parametru přitom sestává ze dvou částí — z určení typu parametru a ze jména parametru. Jméno parametru již na první pohled říká, co funkci máme předat a navíc se na ně odvoláváme v popisu funkce. A na závěr malá ukázka.

§ `integer Rand(integer` *«min»*`, integer` *«max»*`)`

Deklarace funkce nám říká, že funkce `Rand()` vrací hodnotu typu `integer`. Funkce má dva parametry *«min»* a *«max»*, které jsou typu `integer`. Přesný význam parametrů *«min»* a *«max»* nalezneme v popisu funkce.

Popis funkce je další velice důležitou částí celé reference. Nalezneme v něm přesný popis chování funkce, parametrů a vracených hodnot.

U některých funkcí je uvedena krátká ukázka použití, která by měla objasnit všechny případné pochybnosti o jejich činnosti a používání.

V takto označeném odstavci naleznete odkazy na funkce, které mají s právě popisovanou funkcí něco společného.

Pokud jsme funkci použili v jiné části knihy, naleznete zde odkazy, které vás nasměrují na další ukázky použití funkce.

Ačkoliv je referenční přehled poměrně rozsáhlý, neobsahuje všechny funkce, které PHP podporuje. Záměrně jsem vynechal popis funkcí pro práci s méně rozšířenými databázemi Adabas D, FilePro, mSQL, Sybase, Oracle, Solid a MS SQL Server. Tyto funkce využije málokdo, jejich zařazení by knihu zvětšilo o 100 stran a tím nepříznivě zvýšilo i cenu. Popis těchto funkcí naleznete v originální dokumentaci k PHP, navíc je použití funkcí zcela obdobné jako u ostatních funkcí pro práci s databázemi. V referenci se nedostalo ani na popis funkcí pro spolupráci se systémem pro správu dokumentů HyperWave — o žádném uživateli tohoto komerčního systému v Čechách nevím. Poměrně podrobnou dokumentaci k HyperWave naleznete opět v originální dokumentaci. Poslední skupinou funkcí, kterou jsme vynechali, jsou funkce pro práci se soubory komprimovanými metodou gzip. S těmito funkcemi se pracuje stejně jako s funkcemi pro práci se soubory — jediný rozdíl je v tom, že soubory jsou nyní transparentně komprimovány/dekomprimovány podle potřeby.

Kromě výše zmíněných funkcí obsahuje referenční přehled všechny funkce, které podporuje PHP ve verzi 3.0.5.

$CONTENT_LENGTH — Proměnná

Délka dat zasílaných metodou POST

$CONTENT_TYPE — Proměnná

MIME typ dat zasílaných metodou POST

$GATEWAY_INTERFACE — Proměnná

Použitá verze rozhaní CGI, pokud PHP běží jako CGI-skript (nejčastěji `CGI/1.1`)

$HTTP_COOKIE_VARS — Proměnná

Asociativní pole obsahující všechny cookies

$HTTP_GET_VARS — Proměnná

Asociativní pole obsahující všechny parametry předané metodou GET

Ukázky použití naleznete na straně 120.

$HTTP_POST_VARS — Proměnná

Asociativní pole obsahující všechny parametry předané metodou POST

Ukázky použití naleznete na straně 120.

$PATH_INFO — Proměnná

Cesta ke skriptu, který má být zpracován

$PATH_TRANSLATED — Proměnná

Skutečná cesta ke skriptu, který má být zpracován

$PHP_AUTH_PW — Proměnná

Heslo získané pomocí HTTP autentifikace

Ukázky použití naleznete na straně 246.

$PHP_AUTH_TYPE Proměnná

Typ HTTP autentifikace — nejčastěji `basic`

Ukázky použití naleznete na straně 246.

$PHP_AUTH_USER Proměnná

Uživatelské jméno získané při HTTP autentifikaci

Ukázky použití naleznete na straně 246.

$PHP_SELF Proměnná

Jméno právě prováděného skriptu

$QUERY_STRING Proměnná

Nerozkódovaná data předaná metodou GET

$REMOTE_ADDR Proměnná

IP-adresa, ze kterého přišel požadavek

$REMOTE_HOST Proměnná

Doménová adresa počítače, ze kterého přišel požadavek

$REQUEST_METHOD Proměnná

Způsob předání parametrů (`GET` nebo `POST`)

$SCRIPT_FILENAME Proměnná

Jméno souboru, ve kterém je uložen právě prováděný skript

$SCRIPT_NAME Proměnná

Jméno právě prováděného skriptu

Ukázky použití naleznete na straně 180.

$SERVER_NAME Proměnná

Adresa serveru (IP-adresa, doménová adresa nebo alias)

$SERVER_PORT Proměnná

Číslo portu, na kterém běží WWW-server

$SERVER_PROTOCOL Proměnná

Jméno a verze protokolu, kterým přišel požadavek (nejčastěji `HTTP/1.0` nebo `HTTP/1.1`)

$SERVER_SOFTWARE Proměnná

Název a verze WWW-serveru

__FILE__ Konstanta

Tato konstanta obsahuje jméno souboru, ve kterém je uložen právě zpracovávaný skript

---

__LINE__ Konstanta

Tato konstanta obsahuje číslo řádku, na kterém je konstanta použita

```
echo "Právě jsem v souboru ".__FILE__." na řádce ".__LINE__;
```

Abs Matematická funkce

Absolutní hodnota

§ `mixed Abs(mixed` *«výraz»*`)`

Vrací absolutní hodnotu *«výrazu»*. Pokud je argument typu `double`, výsledek je rovněž `double`. Pokud je typu `integer`, výsledek je `integer`.

ACos

Matematická funkce

Arkus kosinus

§ `double ACos(double «výraz»)`

Vrací arkus kosinus zadané hodnoty.

`ASin()` na straně 260 a `ATan()` na straně 260.

AddSlashes

Funkce pro práci s textovými řetězci

Doplnění zpětných lomítek před citlivé znaky

§ `string AddSlashes(string «řetězec»)`

V *«řetězci»* nahradí všechny výskyty apostrofu ‘'’, úvozovek ‘"’ a zpětného lomítka ‘\’ znaky ‘\'’, ‘\"’ a ‘\\’. Funkce je užitečná pro úpravu řetězců před jejich začleněním do dotazů v jazyce SQL.

`StripSlashes()` na straně 417 a `QuoteMeta()` na straně 402.

Ukázky použití naleznete na stranách 157, 174.

Apache_Note

Funkce pracující pouze na serveru Apache

Nastavení/přečtení poznámky požadavku

§ `string Apache_Note(string «jméno», string «hodnota»)`

Pokud je funkce volána pouze s prvním parametrem, vrací hodnotu poznámky *«jméno»*. Pokud je volána s oběma parametry, nastaví poznámku *«jméno»* na *«hodnotu»* a vrátí předchozí hodnotu poznámky. Funkce pracuje pouze tehdy, pokud používáme PHP jako modul serveru Apache. Funkce se používá pro předávání informací mezi moduly serveru, které se podílejí na zpracování jednoho požadavku.

Array

Funkce pro práci s poli

Vytvoření pole

§ `array Array(...)`

Tento příkaz slouží k vytvoření a inicializaci pole. Podrobný popis nalezneme na straně 43.

`List()` na straně 355.

Ukázky použití naleznete na stranách 43, 44, 273.

Array_Walk

Funkce pro práci s poli

Na všechny prvky pole aplikuje zadanou funkci

§ `void Array_Walk(array` *«pole»*`, fptr` *«funkce»*`)`

Funkce pro každý prvek *«pole»* provede zadanou *«funkci»*. Funkce musí být definována s jedním parametrem, do kterého je vždy předán právě zpracovávaný prvek pole.

Následující skript nám umožní snadno vypsat obsah pole:

```
function ElementPrint($x)
{
    echo $x."<BR>\n";
}

$p = Array(10, 12, 7, 9, 16, "Něco", "dáme", "do", "pole");
Array_Walk($p, "ElementPrint");
```

ARSort

Funkce pro práci s poli

Sestupně setřídí pole a zachová indexy prvků

§ `void ARSort(array` *«pole»*`)`

Funkce sestupně setřídí prvky *«pole»*. Po setřídění jsou zachovány indexy jednotlivých prvků.

Dejme tomu, že máme asociativní pole, kde rodnému číslu odpovídá jméno osoby. My chceme toto pole setřídit podle jména, abychom celé pole později mohli vytisknout v abecedním pořadí. Pole je vytvořeno následujícím způsobem:

```
$adresar = Array("565712/0457" => "Hromádková Kateřina",
                 "690823/0987" => "Novák Jan",
                 ...);
```

Pokud toto pole setřídíme pomocí `ARSort($adresar)`, zůstanou zachovány asociace mezi rodným číslem a jménem, ale prvky budou v poli uspořádány sestupně podle jména.

`ASort()` na straně 260, `RSort()` na straně 406, `Sort()` na straně 411 a `KSort()` na straně 344.

ASort

Funkce pro práci s poli

Setřídí pole a zachová indexy prvků

§ `void ASort(array` *«pole»*`)`

Funkce setřídí prvky *«pole»*. Po setřídění jsou zachovány indexy jednotlivých prvků.

`ARSort()` na straně 259, `RSort()` na straně 406, `Sort()` na straně 411 a `KSort()` na straně 344.

ASin

Matematická funkce

Arkus sinus

§ `double ASin(double` *«výraz»*`)`

Vrací arkus sinus zadané hodnoty.

`ACos()` na straně 258 a `ATan()` na straně 260.

ATan

Matematická funkce

Arkus tangens

§ `double ATan(double «výraz»)`

Vrací arkus tangens zadané hodnoty.

`ACos()` na straně 258 a `ASin()` na straně 260.

Base64_Encode

Funkce pro práci s URL adresami

Zakóduje řetězec pomocí MIME-kódování Base64

§ `string Base64_Encode(string «řetězec»)`

Funkce zakóduje *«řetězec»* kódováním Base64, které je definováno ve standardu MIME (RFC 2045). Výsledkem kódování je řetězec, který lze bez problému přenášet i 7bitovými komunikačními kanály. Hodí se především pro začleňování binárních dat do e-mailů. Zakódováním se řetězec prodlouží asi o 33 %.

`Base64_Decode()` na straně 261.

Base64_Decode

Funkce pro práci s URL adresami

Rozkóduje řetězec zakódovaný pomocí MIME-kódování Base64

§ `string Base64_Decode(string «řetězec»)`

Funkce rozkóduje *«řetězec»* zakódovaný metodou Base64.

`Base64_Encode()` na straně 261.

BaseName

Funkce pro práci se soubory

Zjistí jméno souboru z úplně zadané cesty k souboru

§ `string BaseName(string «cesta»)`

Zjistí samotné jméno souboru, který je určen celou svojí *«cestou»*.

Uložení jména souboru do proměnné `$jmeno`:

```
$jmeno = BaseName("/home/jkj/var/htmldoc.txt");
echo $jmeno;          // Vypíše htmldoc.txt
```

DirName() na straně 282.

BCAdd Funkce pro přesné aritmetické operace

Sčítání

§ `string BCAdd(string «a», string «b», integer «přesnost»)`

Funkce přesně sečte čísla *«a»* a *«b»* a vrátí jejich součet. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst.

BCSub() na straně 264.

BCComp Funkce pro přesné aritmetické operace

Porovnání dvou čísel

§ `integer BCComp(string «a», string «b», integer «přesnost»)`

Funkce porovná dvě čísla *«a»* a *«b»*. Pokud jsou obě čísla shodná, vrací funkce nulu. Pokud je *«a»* větší než *«b»*, vrací funkce hodnotu 1. Pokud je naopak *«b»* větší než *«a»*, vrací funkce −1.

Pomocí nepovinného parametru *«přesnost»* můžeme určit kolik desetinných míst se bere v úvahu při porovnávání.

BCDiv Funkce pro přesné aritmetické operace

Podíl

§ `string BCDiv(string «a», string «b», integer «přesnost»)`

Funkce vrací podíl čísel *«a»* a *«b»*. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst výsledku.

BCMul() na straně 263 a BCMod() na straně 262.

BCMod Funkce pro přesné aritmetické operace

Zbytek po dělení

§ `string BCMod(string «a», string «b»)`

Funkce vrací zbytek po dělení čísla *«a»* číslem *«b»*.

BCDiv() na straně 262 a BCMul() na straně 263.

BCMul

Funkce pro přesné aritmetické operace

Součin

§ `string BCMul(string` *«a»*, `string` *«b»*, `integer` *«přesnost»*`)`

Funkce vrací součin čísel *«a»* a *«b»*. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst výsledku.

`BCDiv()` na straně 262 a `BCMod()` na straně 262.

BCPow

Funkce pro přesné aritmetické operace

Umocnění

§ `string BCPow(string` *«a»*, `string` *«exponent»*, `integer` *«přesnost»*`)`

Funkce umocní číslo *«a»* na *«exponent»*. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst výsledku.

`BCSqrt()` na straně 263.

BCScale

Funkce pro přesné aritmetické operace

Nastavení přesnosti

§ `integer BCScale(integer` *«přesnost»*`)`

Funkce nastaví *«přesnost»*, která se bude používat v ostatních funkcích knihovny BC, které nemají nastavenu přesnost. Funkce vždy vrací `true`.

BCSqrt

Funkce pro přesné aritmetické operace

Druhá odmocnina

§ `string BCSqrt(string` *«x»*, `integer` *«přesnost»*`)`

Funkce vrací druhou odmocninu z čísla *«x»*. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst výsledku.

BCSub

Funkce pro přesné aritmetické operace

Rozdíl

§ `string BCSub(string` *«a»*`, string` *«b»*`, integer` *«přesnost»*`)`

Funkce přesně odečte čísla *«a»* a *«b»* a vrátí jejich rozdíl. Pomocí nepovinného parametru *«přesnost»* můžeme nastavit počet desetinných míst výsledku.

`BCAdd()` na straně 262.

BinDec

Matematická funkce

Převod dvojkového čísla na desítkové

§ `integer BinDec(string` *«dvojkové číslo»*`)`

Funkce převádí *«dvojkové číslo»* zapsané jako řetězec složený z nul a jedniček na jeho desítkové vyjádření. Funkce může převádět čísla od nuly do binárního čísla vyjádřeného 31 jedničkami — tj. 2 147 483 647 desítkově.

`DecBin()` na straně 280.

Ceil

Matematická funkce

Zaokrouhlení desetinného čísla nahoru

§ `integer Ceil(double` *«výraz»*`)`

Vrací nejbližší vyšší celočíselnou hodnotu z *«výrazu»*. Použití funkce `Ceil()` na celá čísla je zbytečné plýtvání časem.

`Floor()` na straně 294 a `Round()` na straně 406.

ChDir

Funkce pro práci s adresáři

Nastavení aktuálního adresáře

§ `integer ChDir(string` *«adresář»*`)`

Funkce nastaví aktuální adresář na *«adresář»*. Pokud se změna povede, vrací funkce `true`. V opačném případě `false`.

CheckDate

Funkce pro práci s datem a časem

Kontrola správnosti data

§ `integer CheckDate(integer` *«den»*`, integer` *«měsíc»*`, integer` *«rok»*`)`

Funkce kontroluje, zda kombinace *«den»*, *«měsíc»* a *«rok»* je platným datem. Pokud ano, vrací `true`. V opačném případě vrací `false`. *«Rok»* můžeme zadat od 1900 do 32 767, *«měsíc»* mezi 1 až 12 a *«den»* od 1 do 28, 29, 30 nebo 31 — podle toho jaký je měsíc. Funkce samozřejmě bere v úvahu přestupné roky.

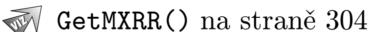
CheckDNSRR

Síťové funkce

Zjištění existence záznamu určitého typu v DNS

§ `integer CheckDNSRR(string` *«počítač»*`, string` *«typ»*`)`

Funkce vrací `true`, pokud pro *«počítač»* existuje v DNS záznam typu *«typ»*. Jako *«počítač»* můžeme použít doménovou nebo IP-adresu. Pokud záznam typu *«typ»* pro počítač v DNS není nebo došlo k chybě, vrací funkce `false`. Pokud *«typ»* neuvedeme, hledá se typ `MX`. Další typy, které můžeme použít, jsou: `A`, `MX`, `SOA`, `PTR`, `CNAME` a `ANY`. Poslední z typů vyhledává ve všech typech záznamů DNS.


`GetMXRR()` na straně 304.

ChGrp

Funkce pro práci se soubory

Změní skupinu uživatelů, které soubor náleží

§ `integer ChGrp(string` *«soubor»*`, mixed` *«skupina»*`)`

U *«souboru»* změní skupinu uživatelů, které soubor náleží, na *«skupinu»*. *«skupinu»* můžeme zadat jménem nebo jejím číslem (GID).

Funkce vrací `true`, pokud se podařilo změnu skupiny úspěšně provést. V opačném případě vrací funkce `false`.

Funkce správně pracuje pouze na systémech, které tuto funkci podporují — to jsou především všemožné i nemožné odnože Unixu.

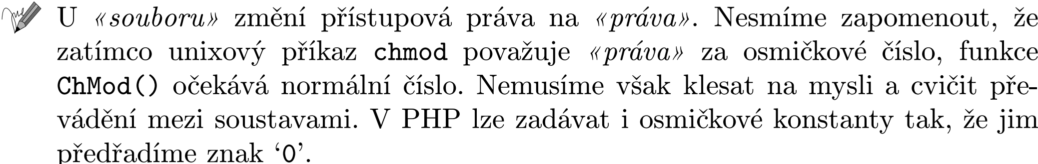
`ChOwn()` na straně 266 a `ChMod()` na straně 266.

ChMod

Funkce pro práci se soubory

Změní přístupová práva k souboru

§ `integer ChMod(string «soubor», integer «práva»)`



U *«souboru»* změní přístupová práva na *«práva»*. Nesmíme zapomenout, že zatímco unixový příkaz `chmod` považuje *«práva»* za osmičkové číslo, funkce `ChMod()` očekává normální číslo. Nemusíme však klesat na mysli a cvičit převádění mezi soustavami. V PHP lze zadávat i osmičkové konstanty tak, že jim předřadíme znak '`0`'.

Funkce vrací `true`, pokud se podařilo úspěšně změnit přístupová práva. V opačném případě vrací funkce `false`.

Funkce správně pracuje pouze na systémech, které tuto funkci podporují — to jsou především všechny možné i nemožné odnože Unixu.

U souboru `refreshdb` nastavíme přístupová práva na 755 — tj. vlastník souboru má plný přístup, skupina uživatelů a všichni ostatní mají pouze přístup pro čtení a spouštění:

```
ChMod("refreshdb", 0755);   // Nula před 755 je důležitá
```

`ChGrp()` na straně 265 a `ChOwn()` na straně 266.

Chop

Funkce pro práci s textovými řetězci

Odstranění mezer a tabulátorů z konce řetězce

§ `string Chop(string «řetězec»)`

Z konce řetězce odstraní všechny netisknutelné znaky, jako mezery, tabulátory a konce řádků.

`Trim()` na straně 425, `LTrim()` na straně 356 a `RTrim()` na straně 407.

ChOwn

Funkce pro práci se soubory

Změní vlastníka souboru

§ `integer ChOwn(string «soubor», mixed «vlastník»)`

U *«souboru»* změní vlastníka na *«vlastníka»*. *«vlastníka»* můžeme zadat uživatelským jménem nebo jeho číslem (UID).

Funkce vrací `true`, pokud se podařilo změnu úspěšně provést. V opačném případě vrací funkce `false`.

Funkce správně pracuje pouze na systémech, které tuto funkci podporují — to jsou především všemožné i nemožné odnože Unixu.

`ChGrp()` na straně 265 a `ChMod()` na straně 266.

Chr

Funkce pro práci s textovými řetězci

Vrací znak s určitým ASCII-kódem

§ `string Chr(integer` *«kód»*`)`

Vrací znak, jehož kód je *«kód»*.

Připojení znaku velké ‘A’ (kód 65) na konec řetězce `$s`:

```
$s .= Chr(65);
```

`Ord()` na straně 386.

ClearStatCache

Funkce pro práci se soubory

Vymazání vyrovnávací paměti volání `stat()`

§ `void ClearStatCache(void)`

Na mnoha systémech je volání `stat()`, které zjišťuje některé stavové informace o souborech, poměrně časově náročné. Proto si volání pamatuje všechny stavové údaje pro posledně ověřovaný soubor. Může se však stát, že mezi voláními `stat()` je soubor změněn nebo vymazán a funkce by vracela nesprávné údaje. Funkce `ClearStatCache()` vymaže vyrovnávací paměť příkazu `stat()`.

Zde jsou funkce závislé na systémovém volání `stat()`: `Stat()`, `File_Exists()`, `FileCTime()`, `FileATime()`, `FileINode()`, `FileGroup()`, `FileOwner()`, `FileSize()`, `FileType()` a `FilePerms()`.

CloseDir

Funkce pro práci s adresáři

Uzavření adresáře otevřeného pro čtení

§ `void CloseDir(integer «dp»)`

Funkce uzavře adresář *«dp»* otevřený pomocí funkce `OpenDir()`.

`OpenDir()` na straně 384, `ReadDir()` na straně 404 a `RewindDir()` na straně 406.

CloseLog

Konfigurace a informace o PHP

Zavření systémového protokolu

§ `void CloseLog(void)`

Volání funkce ukončí přístup skriptu k systémovému protokolu.

`OpenLog()` na straně 385 a `SysLog()` na straně 423.

COM_Get

Funkce pro práci s COM-objekty

Zjištění hodnoty vlastnosti COM-objektu

§ `mixed COM_Get(integer «idispatch», string «vlastnost»)`

Funkce vrací hodnotu *«vlastnosti»* objektu *«idispatch»*, který byl předtím vytvořen pomocí `COM_Load()`.

S využitím XML-parseru od Microsoftu zjitíme použitou verzi XML u určitého dokumentu:

```
... inicializace objektu a načtení dokumentu ...
echo "Použitá verze XML: ".COM_Get($xml, "version");
```

`COM_Set()` na straně 269, `COM_Load()` na straně 269 a `COM_Invoke()` na straně 269.

COM_Invoke

Funkce pro práci s COM-objekty

Vyvolání metody COM-objektu

§ `mixed COM_Invoke(ineteger` *«idispatch»*`, string` *«metoda»*`,` *«parametry»*`)`

Funkce vyvolá *«metodu»* objektu *«idispatch»* a předá jí *«parametry»*. Pokud má metoda více parametrů, při volání funkce `COM_Invoke()` se rovněž použije více parametrů. Funkce vrací tu hodnotu, kterou obdrží od volané metody.

Vytvoření spojení s datovým zdrojem `test` pomocí objektů ADO:

```
$spojeni = COM_Load("ADODB.Connection");
COM_Invoke($spojeni, "Open", "DSN=test");
```

`COM_Get()` na straně 268, `COM_Set()` na straně 269 a `COM_Load()` na straně 269.

COM_Load

Funkce pro práci s COM-objekty

Vytvoření instance COM-objektu

§ `integer COM_Load(string` *«ProgID»*`)`

Funkce vytvoří instanci COM-objektu se jménem *«ProgID»* a vrátí jeho identifikátor *«idispatch»*. Pokud se objekt nepodaří vytvořit, vrací funkce `false`.

Vytvoření objektu pro čtení XML-dokumentů pomocí parseru `msxml` od Microsoftu:

```
$xml = COM_Load("msxml");
```

`COM_Get()` na straně 268, `COM_Set()` na straně 269 a `COM_Invoke()` na straně 269.

COM_Set

Funkce pro práci s COM-objekty

Nastavení vlastnosti COM-objektu

§ `integer COM_Set(integer` *«idispatch»*`, string` *«vlastnost»*`, mixed` *«hodnota»*`)`

Objektu určenému pomocí *«idispatch»* nastaví *«vlastnost»* na *«hodnotu»*. Funkce vrací `true`, pokud se vlastnost podařilo nastavit. V opačném případě vrací `false`.

Nastavením vlastnoti `URL` objektu `msxml` určujeme adresu, ze které bude nahrán XML dokument určený ke zpracování:

```
$xml = COM_Load("msxml");
if (!COM_Set($xml, "URL", "http://nekde/docs/soubor.xml"))
    echo "Tak se nám někde něco...";
```

`COM_Get()` na straně 268, `COM_Invoke()` na straně 269 a `COM_Load()` na straně 269.

Copy

Funkce pro práci se soubory

Zkopírování souboru

§ `integer Copy(string` *«zdroj»*`, string` *«cíl»*`)`

Vrací `true` pokud se povedlo soubor *«zdroj»* zkopírovat do souboru *«cíl»*.

Malá ukázka vytvoření záložní kopie souboru:

```
if (!Copy($soubor, "$soubor.bak"))
    echo "Nepodařilo se vytvořit záložní kopii.";
```

`Rename()` na straně 405.

Cos

Matematická funkce

Kosinus

§ `double Cos(double` *«výraz»*`)`

Vrací kosinus *«výrazu»*. *«výraz»* je považován za velikost úhlu zadanou v radiánech.

`Sin()` na straně 409 a `Tan()` na straně 424.

Count

Funkce pro práci s poli

Zjištění počtu prvků proměnné

§ `integer Count(mixed «proměnná»)`

Funkce vrátí počet prvků uložených v *«proměnné»*. Obvykle je *«proměnná»* typu pole. Pokud použijeme proměnnou skalárního typu, vrací funkce hodnotu 1. Pokud *«proměnná»* nemá hodnotu, vrací funkce nulu.

Průchod polem indexovaným od nuly:

```
for ($i=0; $i<Count($pole); $i++)
    echo $pole[i];
```

`SizeOf()` na straně 410, `IsSet()` na straně 341 a `Is_Array()` na straně 337.

Ukázky použití naleznete na stranách 44, 141.

Crypt

Funkce pro práci s textovými řetězci

Zašifruje text pomocí algoritmu DES

§ `string Crypt(string «řetězec», string «salt»)`

Zašifruje *«řetězec»* metodou DES. Parametr *«salt»* je nepovinný a slouží k předání dvouznakového řetězce, na kterém se založí šifrování. Funkce pracuje jen na některých systémech. Velkou nevýhodou je, že neexistuje funkce, která by takto zašifrovaný řetězec uměla dešifrovat.

V praxi funkci využijeme pouze v případech, kdy skriptem potřebujeme do nějakého souboru, jako `/etc/passwd` nebo `.htpasswd`, zapsat hesla.

Ukázky použití naleznete na straně 208.

Current

Funkce pro práci s poli

Vrací hodnotu aktuálního prvku pole

§ `mixed Current(array «pole»)`

Funkce vrací prvek *«pole»*, na který ukazuje interní ukazatel *«pole»*.

`Each()` na straně 282, `Prev()` na straně 399, `Next()` na straně 372, `Reset()` na straně 405, `End()` na straně 284 a `Array_Walk()` na straně 259.

Ukázky použití naleznete na straně 44.

Date

Funkce pro práci s datem a časem

Formátování časových údajů

§ `string Date(string «formát», integer «čas»)`

Funkce slouží k formátování údajů o čase a datu. Výsledný formát je určen formátovacím řetězcem. V něm můžeme použít znaky se speciálním významem uvedené v tabulce 9-1. Pokud použijeme znak, který nemá speciální význam, stane se součástí zformátovaného řetězce. Druhý parametr je nepovinný a udává čas jako počet sekund od 1. ledna 1970. Pokud parametr nepoužijeme, automaticky se použije údaj o aktuálním čase, který vrací funkce `Time()`.

Znak	Popis
`Y`	Rok jako čtyřčíslí (např. `1998`)
`y`	Rok jako dvojčíslí (např. `98`)
`M`	Anglická zkratka jména měsíce (např. `Jan`)
`m`	Číslo měsíce (01–12)
`F`	Anglické jméno měsíce (např. `January`)
`D`	Anglická zkratka dne v týdnu (např. `Fri`)
`l`	Anglické jméno dne v týdnu (např. `Friday`)
`w`	Číslo dne v týdnu (1–7)
`d`	Číslo dne v měsíci (01–31)
`j`	Číslo dne v měsíci (1–31)
`z`	Číslo dne v roce (001–365)
`H`	Hodina (00–23)
`h`	Hodina (01–12)
`i`	Minuta (00–59)
`s`	Sekunda (00–59)
`U`	Počet sekund od začátku času (od 1. ledna 1970)
`S`	Anglická koncovka pořadového čísla dne v měsíci
`A`	Indikátor dopoledne/odpoledne (`AM/PM`)
`a`	Indikátor dopoledne/odpoledne (`am/pm`)

Tab. 9-1: Parametry formátovacího řetězce funkce Date()

`GMDate()` na straně 306, `Time()` na straně 424, `MkTime()` na straně 359 a `StrFTime()` na straně 415.

Ukázky použití naleznete na straně 30.

dBase_Add_Record

Funkce pro práci se soubory dBase

Přidání záznamu do databáze

§ `integer dBase_Add_Record(integer` *«databáze»*`, array` *«záznam»*`)`

Funkce do *«databáze»* přidá nový *«záznam»*. Záznam se předává jako pole, kde každý prvek pole odpovídá jedné položce. Počet prvků pole musí odpovídat počtu položek databáze, jinak funkce neproběhne úspěšně a vrátí `false`. Pokud se záznam podaří do databáze přidat, vrací funkce `true`.

Při konstrukci záznamu pro vložení můžeme s výhodou použít příkaz `Array`:

```
$dbid = dBase_Open("./Zam.dbf", 2);
if (!dBase_Add_Record($dbid, Array("Jan Novák", "731116/0578", "12700")))
    echo "Nový záznam se nepodařilo přidat.";
```

dBase_Close

Funkce pro práci se soubory dBase

Zavření databáze

§ `integer dBase_Close(integer` *«databáze»*`)`

Funkce zavře databázi určenou jejím identifikátorem *«databáze»*. Pokud se databázi podaří uzavřít, vrací funkce `true`, v opačném případě `false`.

dBase_Create

Funkce pro práci se soubory dBase

Vytvoření dBase databáze

§ `integer dBase_Create(string` *«jméno_souboru»*`, array` *«položky»*`)`

Funkce vytvoří novou databázi v daném souboru. Popis položek vytvářené databáze je předán pomocí pole *«položky»*. Toto pole obsahuje pro každou položku pole, které postupně obsahuje prvky se jménem položky, typem položky, délkou položky a přesností položky. Typ položky je jednoznakový řetězec s následujícím významem:

- `L` logická hodnota (ano/ne);
- `M` memo — textová položka neomezené délky, která není v PHP podporována; typ nemá ani délku ani přesnost;
- `D` datum uložené ve formátu `RRRRMMDD`; typ nemá ani délku ani přesnost;
- `N` číslo; číslo má svoji délku a přesnost (počet cifer za desetinnou čárkou);
- `C` textový řetězec; tento typ má délku, která odpovídá maximální délce řetězce.

Pokud se podaří databázi vytvořit, vrací funkce identifikátor databáze. V případě chyby je vrácena hodnota `false`.

Vytvoření databáze `Zam.dbf`, která obsahuje položky pro jméno, rodné číslo a plat zaměstnanců:

```
$dbid = dbase_Create("./Zam.dbf", Array(Array("Jmeno", "C", 30, 0),
                                        Array("RC", "C", 11, 0),
                                        Array("Plat", "N", 10, 2)));
if (!$dbid)
    echo "Databázi se nepodařilo vytvořit!";
```

dBase_Delete_Record

Funkce pro práci se soubory dBase

Smazání záznamu

§ `integer dBase_Delete_Record(integer` *«databáze»*`,`
`integer` *«číslo_záznamu»*`)`

Funkce smaže záznam z databáze. Záznam je určen pomocí svého čísla; záznamy jsou číslovány od nuly. Záznam ve skutečnosti není smazán, ale pouze označen jako smazaný. K fyzickému odstranění ze souboru databáze dojde až po zavolání funkce `dBase_Pack()`. Pokud se záznam podaří smazat, vrací funkce `true`, v opačném případě `false`.

dBase_Get_Record

Funkce pro práci se soubory dBase

Přečtení záznamu z databáze

§ `array dBase_Get_Record(integer` *«databáze»*`, integer` *«číslo_záznamu»*`)`

Funkce vrací záznam databáze určený pomocí pořadového čísla. Záznamy jsou přitom číslovány od jedné. Všechny datové typy dBase jsou převedeny na odpovídající datový typ PHP s výjimkou data, které je převedeno na řetězec. Záznam je vrácen jako pole, které je indexováno od nuly. Pokud pole obsahuje prvek s indexem `deleted` a tento prvek obsahuje hodnotu 1, jedná se o záznam označený jako smazaný.

Pro získání obsahu jednotlivých položek záznamu s výhodou použijeme příkaz `list`:

```
list($Jmeno, $RC, $Plat) = dBase_Get_Record($dbid, $i);
echo $Jmeno;
echo $RC;
```

```
echo $Plat;
```

---

dBase_Get_Record_With_Names Funkce pro práci se soubory dBase

Přečtení záznamu z databáze

§ `array dBase_Get_Record_With_Names(integer` *«databáze»*`,`
`integer` *«číslo_záznamu»*`)`

Funkce vrací záznam databáze určený pomocí pořadového čísla. Záznamy jsou přitom číslovány od jedné. Všechny datové typy dBase jsou převedeny na odpovídající datový typ PHP s výjimkou data, které je převedeno na řetězec. Záznam je vrácen jako asociativní pole, kde indexy obsahují názvy jednotlivých položek. Pokud pole obsahuje prvek s indexem `deleted` a tento prvek obsahuje hodnotu 1, jedná se o záznam označený jako smazaný.

Se jmény položek se pracuje mnohem lépe než s jejich čísly:

```
$zaznam = dBase_Get_Record_With_Names($dbid, $i);
echo $zaznam["Jmeno"];
echo $zaznam["RC"];
echo $zaznam["Plat"];
```

---

dBase_NumFields Funkce pro práci se soubory dBase

Funkce vrací počet položek, které obsahuje databáze

§ `integer dBase_NumFields(integer` *«databáze»*`)`

Funkce vrací počet položek, které obsahuje databáze.

---

dBase_NumRecords Funkce pro práci se soubory dBase

Zjistí počet záznamů v databázi

§ `integer dBase_NumRecords(integer` *«databáze»*`)`

Funkce vrací počet záznamů, které obsahuje *«databáze»*.

dBase_Open

Funkce pro práci se soubory dBase

Otevření databáze

§ `integer dBase_Open(string «jméno_souboru», integer «režim»)`

Funkce otevře databázi uloženou v zadaném souboru. Pomocí parametru *«režim»* ovlivňujeme způsob otevření: `0` — přístup pouze pro čtení, `1` — přístup pouze pro zápis a `2` — přístup pro čtení i zápis.

Funkce vrací identifikátor databáze v případě, že se databázi podařilo otevřít. V opačném případě vrací `false`.

dBase_Pack

Funkce pro práci se soubory dBase

Odstranění smazaných záznamů ze souboru s databází

§ `integer dBase_Pack(integer «databáze»)`

Funkce z *«databáze»* odstraní všechny smazané záznamy a tím pádem i zmenší soubor o již nepotřebné místo.

dbList

Funkce pro práci s databází dbm

Informace o použitém druhu knihovny dbm

§ `string dbList(void)`

Funkce vrací řetězec, který obsahuje popis použité knihovny dbm.

dbmClose

Funkce pro práci s databází dbm

Zavření databáze

§ `integer dbmClose(integer «databáze»)`

Funkce zavře *«databázi»* určenou jejím identifikátorem. Pokud se databázi podaří zavřít, vrací funkce hodnotu `true`. V opačném případě `false`.

dbmDelete Funkce pro práci s databází dbm

Smazání hodnoty uložené pod daným klíčem

§ `integer dbmDelete(integer «databáze», string «klíč»)`

Funkce smaže z databáze *«klíč»* a jemu příslušející hodnotu. V případě úspěšného smazání vrací funkce `true`. Pokud dojde k chybě nebo pokud klíč v databázi není, vrací funkce hodnotu `false`.

Ukázky použití naleznete na straně 209.

dbmExists Funkce pro práci s databází dbm

Zjištění, zda pro daný klíč existuje v databázi hodnota

§ `integer dbmExists(integer «databáze», string «klíč»)`

Funkce vrací `true`, pokud v *«databázi»* existuje pro *«klíč»* nějaká hodnota. Pokud neexistuje, vrací funkce `false`.

dbmFetch Funkce pro práci s databází dbm

Získání hodnoty uložené v databázi pod nějakým klíčem

§ `string dbmFetch(integer «databáze», string «klíč»)`

Funkce vrátí hodnotu, která je v *«databázi»* uložena pod určitým *«klíčem»*. Pokud takový záznam v databázi není, vrací funkce `false`.

Ukázky použití naleznete na straně 209.

dbmFirstKey Funkce pro práci s databází dbm

Funkce vrací hodnotu prvního klíče v databázi

§ `string dbmFirstKey(integer «databáze»)`

Funkce vrací hodnotu prvního klíče uloženého v *«databázi»*. Pokud při provádění funkce dojde k chybě (např. v databázi nejsou žádné klíče), je vrácena hodnota `false`.

Ukázky použití naleznete na straně 214.

dbmInsert

Funkce pro práci s databází dbm

Vložení hodnoty do databáze

§ `integer dbmInsert(integer «databáze», string «klíč», string «hodnota»)`

Funkce vloží do *«databáze»* *«hodnotu»* s daným *«klíčem»*. Pokud bylo vložení úspěšné, vrací funkce hodnotu 0. Pokud byla databáze otevřena pro čtení, vrací funkce hodnotu −1. Pokud již klíč v databázi existoval, vrací funkce hodnotu 1. V tomto případě můžeme ke změně hodnoty použít funkci `dbmReplace()`.

`dbmReplace()` na straně 279.

Ukázky použití naleznete na stranách 208, 209.

dbmNextKey

Funkce pro práci s databází dbm

Zjištění hodnoty následujícího klíče

§ `string dbmNextKey(integer «databáze», string «klíč»)`

Funkce vrací klíč, který v *«databázi»* následuje po *«klíči»*. Pokud už v databázi žádný další klíč není, vrací funkce `false`.

Funkci můžeme snadno využít na zpracování obsahu celé databáze:

```
$db = dbmOpen("./databaze", "r");
$key = dbmFirstKey($db);
while ($key):
    echo dbmFetch($db, $key)."<BR>\n";
    $key = dbmNextKey($db, $key);
endwhile;
```

`dbmFirstKey()` na straně 277.

Ukázky použití naleznete na straně 214.

dbmOpen
Funkce pro práci s databází dbm

Otevření databáze

§ `integer dbmOpen(string` *«soubor»*`, string` *«režim»*`)`

Funkce otevře databázi uloženou v *«souboru»* a vrátí identifikátor, který používáme v dalších funkcích pro práci s databází. Pomocí parametru *«režim»* určujeme způsob, jakým se má soubor otevřít. Parametr podporuje hodnotu `r` pro otevření pro čtení, `n` pro vytvoření nové databáze a `w` pro otevření pro zápis. Pokud se databázi nepodaří otevřít, vrací funkce hodnotu `false`.

Ukázky použití naleznete na straně 208.

dbmReplace
Funkce pro práci s databází dbm

Nahrazení hodnoty pro daný klíč v databázi

§ `integer dbmReplace(integer` *«databáze»*`, string` *«klíč»*`,`
`string` *«hodnota»*`)`

Funkce nahradí hodnotu uloženou v *«databázi»* pod *«klíčem»* zadanou *«hodnotou»*. Pokud klíč zatím v databázi neexistuje, jsou klíč i hodnota do databáze přidány. V případě úspěšnosti operace vrací funkce `true`. V případě chyby vrací `false`.

Ukázky použití naleznete na straně 209.

Debugger_Off
Konfigurace a informace o PHP

Vypne interní debugger

§ `void Debugger_Off(void)`

Funkce vypne interní debugger.

Debugger_On
Konfigurace a informace o PHP

Zapne interní debugger

§ `void Debugger_On(string` *«počítač»*`)`

Zapne interní debugger. Debugger se připojí na *«počítač»*. Port debuggeru lze nastavit v konfiguračním souboru, standardně se jedná o port 7869.

DecBin

Matematická funkce

Převádí desítkové číslo na jeho binární reprezentaci

§ `string DecBin(integer «výraz»)`

Převede celočíselný *«výraz»* na jeho reprezentaci v dvojkové soustavě. *«Výraz»* přitom může být celé číslo od 0 do 2 147 483 647.

`BinDec()` na straně 264.

Ukázky použití naleznete na straně 50.

DecHex

Matematická funkce

Převod desítkového čísla na šestnáctkové

§ `string DecHex(integer «výraz»)`

Převede celočíselný *«výraz»* na jeho reprezentaci v šestnáctkové soustavě. *«Výraz»* přitom může být celé číslo od 0 do 2 147 483 647.

`HexDec()` na straně 307, `DecBin()` na straně 280 a `BinDec()` na straně 264.

DecOct

Matematická funkce

Převod desítkového čísla na osmičkové

§ `string DecOct(integer «výraz»)`

Převede celočíselný *«výraz»* na jeho reprezentaci v osmičkové soustavě. *«Výraz»* přitom může být celé číslo od 0 do 2 147 483 647.

`OctDec()` na straně 374, `DecHex()` na straně 280, `HexDec()` na straně 307, `DecBin()` na straně 280 a `BinDec()` na straně 264.

Deg2Rad

Matematická funkce

Převod stupňů na radiány

§ `double Deg2Rad(double «úhel»)`

Funkce převede *«úhel»* ve stupních na radiány.

`Rad2Deg()` na straně 402.

Die

Konfigurace a informace o PHP

Funkce ukončí běh skriptu a vypíše hlášení

§ `void Die(string «hlášení»)`

Funkce vypíše *«hlášení»* a ukončí běh skriptu, podobně jako `exit`.

Bezpečné otevření souboru může vypadat třeba takto:

```
$filename = "/cesta/k/souboru.txt";
$fp = FOpen($filename, "r") or Die ("Nemohu otevřít soubor '$filename'");
```

Ukázky použití naleznete na straně 72.

Dir

Funkce pro práci s adresáři

Pseudotřída pro práci s adresáři

§
```
object Dir(string «adresář»)
integer Dir->handle
integer Dir->path
string Dir->Read()
void Dir->Rewind()
void Dir->Close()
```

`Dir` je pseudotřída pro práci s adresáři. Má dvě členské proměnné `handle` a `path`. První z nich obsahuje ukazatel na adresář (*«dp»*) a druhá jméno čteného adresáře. Členské funkce jsou obdobou funkcí `ReadDir()`, `RewindDir()` a `CloseDir()`.

Následující ukázka vypíše obsah zadaného adresáře:

```
$adresar = Dir("c:\\winnt\\system32");
while ($polozka=$adresar->Read())
    echo $polozka."<BR>";
$adresar->Close();
```

DirName

Funkce pro práci se soubory

Zjistí adresářovou část z úplně zadané cesty k souboru

§ `string DirName(string` *«cesta»*`)`

Zjistí jméno adresáře, který je určen úplnou *«cestou»* k nějakému souboru.

```
$cesta = DirName("/home/jkj/var/htmldoc.txt");
echo $cesta;                                    // /home/jkj/var
```

`BaseName()` na straně 261.

Dl

Ostatní funkce

Načtení dynamické knihovny

§ `integer Dl(string` *«knihovna»*`)`

Načte dynamickou knihovnu PHP. Knihovny se načítají z adresáře určeného v konfiguračním souboru pomocí direktivy `extension_dir`.

Ukázky použití naleznete na straně 184.

DoubleVal

Proměnná

Hodnota výrazu jako typ `double`

§ `double DoubleVal(mixed` *«výraz»*`)`

Převede *«výraz»* na hodnotu typu `double`. Typ *«výrazu»* musí být skalární.

`IntVal()` na straně 336 a `StrVal()` na straně 422.

Each

Funkce pro práci s poli

Vrátí hodnotu indexu a obsah prvku pole a přesune ukazatel na další prvek pole

§ `mixed Each(array` *«pole»*`)`

Funkce vrátí hodnotu indexu a obsah prvku pole, na který ukazuje ukazatel. Ukazatel je poté přesunut na další prvek pole.

Výsledek je vracen ve speciálním tvaru. Normálně je výsledkem funkce asociativní pole se čtyřmi prvky. Prvky s indexy `0` a `key` obsahují hodnotu indexu aktuálního prvku pole. Prvky s indexy `1` a `value` obsahují hodnotu aktuálního prvku pole.

Pokud ukazatel ukazuje za poslední prvek pole, vrací funkce hodnotu `false`. Toho můžeme využít pro elegantní zpracování celého pole.

Pokud naše aplikace pracuje s cookies, může se nám hodit následující kód, který vypíše všechny cookies, které nám zaslal prohlížeč:

```
while (List($index, $hodnota) = Each($HTTP_COOKIE_VARS))
    echo "$index: $hodnota<BR>";
```

`Current()` na straně 271, `Prev()` na straně 399, `Next()` na straně 372, `Reset()` na straně 405, `End()` na straně 284 a `Array_Walk()` na straně 259.

Echo

Vypsání jednoho nebo více výrazů

§ `Echo string` *«výraz1»*, `string` *«výraz2»*, ..., `string` *«výrazN»*

Vypíše hodnotu všech výrazů. Nejedná se o klasickou funkci, ale o příkaz. Všechny výrazy jsou automaticky převedeny na typ `string`.

`Printf()` na straně 400 a `Print` na straně 399.

Empty

Proměnná

Zjistí, zda je proměnná prázdná

§ `integer Empty(mixed` *«proměnná»*`)`

Funkce vrací `false`, pokud *«proměnná»* existuje a má nějakou hodnotu. V ostatních případech vrací `true`.

`IsSet()` na straně 341.

End

Funkce pro práci s poli

Nastaví ukazatel na konec pole

§ `void End(array «pole»)`

Nastaví ukazatel *«pole»* na jeho poslední prvek.

`Current()` na straně 271, `Prev()` na straně 399, `Next()` na straně 372, `Reset()` na straně 405, `Each()` na straně 282 a `Array_Walk()` na straně 259.

EReg

Funkce pro práci s regulárními výrazy

Zjistí, zda řetězec vyhovuje regulárnímu výrazu

§ `integer EReg(string «regexp», string «řetězec», array «shody»)`

Funkce vrací `true`, pokud *«řetězec»* vyhovuje regulárnímu výrazu *«regexp»*, v opačném případě vrací `false`. Pokud použijeme třetí parametr *«shody»*, uloží funkce do pole řetězce, které vyhovují jednotlivým podvýrazům regulárního výrazu (podvýraz regulárního výrazu je ohraničen závorkami). Jako prvek pole *«shody»* s indexem nula se uloží část *«řetězce»*, která vyhovuje celému regulárnímu výrazu.

Funkce je citlivá na velikost písmen.

Následující skript převádí datum z formátu `YYYY-MM-DD` na formát `DD.MM.YYYY`:

```
$Datum = "1998-08-30";

if (EReg("^([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})$", $Datum, $Cast))
    echo "Datum dle našich zvyklostí: $Cast[3].$Cast[2].$Cast[1]";
else
    echo "Špatný formát data: $Datum";
```

`ERegI()` na straně 285, `EReg_Replace()` na straně 284 a `ERegI_Replace()` na straně 286.

EReg_Replace

Funkce pro práci s regulárními výrazy

Nahrazení řetězce podle regulárního výrazu

§ `string EReg_Replace(string «regexp», string «náhrada», string «řetězec»)`

Funkce nahradí část *«řetězce»*, která vyhovuje regulárnímu výrazu *«regexp»* řetězcem *«náhrada»*. V řetězci náhrada můžeme používat speciální metaznak ‘*«číslice»*’, který zastupuje odpovídající podvýraz regulárního výrazu. Metaznak ‘\\0’ přitom zastupuje celý řetězec, který vyhovuje regulárnímu výrazu *«regexp»*.

Funkce se hodí pro dělání všelijakých kejklí s řetězci:

```
$Datum = "1998-08-30";
echo EReg_Replace("[0-9]{4}-([0-9]{1,2})-([0-9]{1,2})", "\\2.\\1.", $Datum);
// Vypíše 30.08.
```

Volání

```
EReg_Replace(«výraz», "\\0", «řetězec»)
```

vrátí zcela nezměněný řetězec. Ukážeme si i složitější příklad. Dejme tomu, že chceme v celé HTML-stránce nahradit odkazy typu

```
<A HREF="«URL»">«Název odkazu»</A>
```

novým tvarem odkazů

```
<A HREF="«URL»" TITLE="«Název odkazu»">«URL»</A>
```

K tomu nám poslouží následující kód:

```
$stranka = EReg_Replace('<A HREF="([^"]*)">([^<]*)</A>',
                        '<A HREF="\\1" TITLE="\\2">\\1</A>',
                        $stranka);
```

ERegI_Replace() na straně 286, EReg() na straně 284 a ERegI() na straně 285.

ERegI

Funkce pro práci s regulárními výrazy

Zjistí, zda řetězec vyhovuje regulárnímu výrazu. Při porovnávání se v úvahu nebere velikost písmen

§ `integer ERegI(string «regexp», string «řetězec», array «shody»)`

Funkce je identická s funkcí EReg() s tím rozdílem, že porovnávání není citlivé na velikost písmen.

EReg() na straně 284, EReg_Replace() na straně 284 a ERegI_Replace() na straně 286.

EReg I_Replace

Funkce pro práci s regulárními výrazy

Nahrazení řetězce podle regulárního výrazu, ve kterém se ignoruje velikost písmen

§ `string EReg I_Replace(string` *«regexp»*`, string` *«náhrada»*`, string` *«řetězec»*`)`

Funkce nahradí část *«řetězce»*, která vyhovuje regulárnímu výrazu *«regexp»* řetězcem *«náhrada»*. V řetězci náhrada můžeme používat speciální metaznak ‘`\\`*«číslice»*’, který zastupuje odpovídající podvýraz regulárního výrazu. Metaznak ‘`\\0`’ přitom zastupuje celý řetězec, který vyhovuje regulárnímu výrazu *«regexp»*. Při porovnávání řetězce s regulárním výrazem není brán ohled na velikost písmen — v tom se tato funkce liší od `EReg_Replace()`.

Dejme tomu, že chceme v celé HTML-stránce nahradit odkazy typu

<A HREF="*«URL»*">*«Název odkazu»*</A>

novým tvarem odkazů

<A HREF="*«URL»*" TITLE="*«Název odkazu»*">*«URL»*</A>

K tomu nám poslouží následující kód:

```
$stranka = EReg I_Replace('<A HREF="([^"]*)">([^<]*)</A>',
                         '<A HREF="\\1" TITLE="\\2">\\1</A>',
                         $stranka);
```

Použití `EReg I_Replace()` je v tomto případě vhodnější než `EReg_Replace()`, protože v jazyce HTML nezáleží na velikosti písmen v názvech tagů a atributů.

`EReg_Replace()` na straně 284, `EReg()` na straně 284 a `EReg I()` na straně 285.

Error_Log

Konfigurace a informace o PHP

Zaslání chybového hlášení

§ `integer Error_Log(string` *«zpráva»*`, integer` *«typ»*`, string` *«cíl»*`, string` *«hlavičky»*`)`

Funkce zašle chybovou *«zprávu»* adresátovi určenému zbývajícími parametry. Funkce vrací `true`, pokud se zprávu podařilo odeslat. V opačném případě vrací `false`.

Pokud je *«typ»* 0, zpráva se zapíše do systémového protokolu.

Pokud je *«typ»* 1, zpráva se odešle elektronickou poštou na adresu *«cíl»*. V tomto případě můžeme použít poslední parametr *«hlavičky»* a přidat do e-mailu některé hlavičky.

Pokud je *«typ»* 2, zpráva je odeslána po síťovém spojení určeném pro ladění PHP. *«cíl»* v tomto případě obsahuje jméno počítače (či IP-adresu) případně doplněné o port a určuje tak socket, na kterém jsou přijímány ladící informace.

Pokud je *«typ»* 3, zpráva se zapíše na konec souboru *«cíl»*.

`OpenLog()` na straně 385, `SysLog()` na straně 423 a `CloseLog()` na straně 268.

Error_Reporting

Konfigurace a informace o PHP

Určení chybových zpráv k hlášení

§ `integer Error_Reporting(integer` *«úroveň»*`)`

Funkce určí druhy chybových zpráv, jež budou vypisovány. Druhy zpráv se určují parametrem *«úroveň»*, kterému předáme logický součet konstant odpovídajících jednotlivým chybovým zprávám. Přehled konstant nalezneme v tabulce 9-2. Funkce vrací poslední hodnotu úrovně hlášených chybových zpráv.

Konstanta	Hodnota	Popis
E_ERROR	1	chyba
E_WARNING	2	varování
E_PARSE	4	syntaktická chyba
E_NOTICE	8	upozornění (např. použití neinicializované proměnné)
E_CORE_ERROR	16	interní chyba jádra
E_CORE_WARNING	32	interní varování

Tab. 9-2: Konstanty určující druhy hlášených zpráv

Při ladění aplikací si můžeme nechat vypisovat i upozornění — můžeme tak odhalit některé skryté chyby:

```
Error_Reporting(E_ERROR|E_WARNING|E_PARSE|E_NOTICE)
```

Ukázky použití naleznete na stranách 90, 91.

EscapeShellCmd

Funkce pro spouštění externích programů

Nahrazení všech nebezpečných znaků escape sekvencí

§ `string EscapeShellCmd(string «příkaz»)`

Funkce v *«příkazu»* nahradí výskyty všech znaků, které by mohly zmást příkazový interpret, příslušnou escape sekvencí. Funkci použijeme zejména tehdy, když spouštíme externí program s parametry, které zadal uživatel. Znemožníme tak uživateli — hackerovi — provádění neoprávněných operací.

`Exec()` na straně 288, `System()` na straně 423 a `PassThru()` na straně 387.

Exec

Funkce pro spouštění externích programů

Vyvolání externího programu

§ `string Exec(string «příkaz», array «výsledek», integer «status»)`

Funkce spustí *«příkaz»* a vrací poslední řádku jeho výstupu. Pokud chceme mít k dispozici celý výstup programu, použijeme druhý parametr. Druhým parametrem je pole předané odkazem — na jeho konec se připojí pro každou řádku výstupu programu jeden prvek. Poslední parametr *«status»* je rovněž nepovinný a slouží ke zjištění návratového kódu programu. Parametr by měl být opět předáván odkazem.

Pokud *«příkaz»* obsahuje parametry zadávané uživatelem, měli bychom na příkaz aplikovat funkci `EscapeShellCmd()`. Uživateli tak zabrání v „ošálení“ příkazu.

`System()` na straně 423, `PassThru()` na straně 387, `POpen()` na straně 398 a `EscapeShellCmd()` na straně 288.

Exp

Matematická funkce

Umocní číslo na Eulerovu konstantu

§ `double Exp(double «výraz»)`

Spočítá hodnotu výrazu $e^{«výraz»}$, kde $e \doteq 2{,}718\,281\,828$ je Eulerova konstanta.

`Pow()` na straně 399.

Explode

Funkce pro práci s textovými řetězci

Rozdělí řetězec na části

§ `array Explode(string «separátor», string «řetězec»)`

Výsledkem funkce je pole řetězců, které jsou obsaženy v *«řetězci»* a odděleny *«separátorem»*.

Funkce `Explode()` je výtečný pomocník při zpracování textových souborů, kde jsou jednotlivé údaje odděleny nějakým separátorem. Příkladem může být pohodlné čtení údajů o jednotlivých uživatelích ze souboru `/etc/passwd`:

```
$passwdline = "jirka:Bh1Qx13:101:5:Jirka Kosek:/home/jirka:/bin/bash";
List($user,$passwd,$uid,$gid,$name,$home,$shell) = Explode(":", $passwdline);
```

`Split()` na straně 412 a `Implode()` na straně 336.

FClose

Funkce pro práci se soubory

Zavření souboru

§ `integer FClose(integer «fp»)`

Funkce zavře soubor otevřený funkcí `FOpen()` nebo `FSockOpen()`. *«fp»* musí být platný ukazatel na soubor. Pokud se soubor podaří zavřít, vrací funkce hodnotu `true`, v opačném případě `false`.

`FOpen()` na straně 294, `POpen()` na straně 398 a `FSockOpen()` na straně 297.

Ukázky použití naleznete na straně 198.

FEof

Funkce pro práci se soubory

Test konce souboru

§ `integer FEof(integer «fp»)`

Funkce vrací `true`, pokud jsme již na konci souboru *«fp»*. V opačném případě vrací `false`.

FGetC

Funkce pro práci se soubory

Přečtení jednoho znaku ze souboru

§ `string FGetC(integer «fp»)`

Funkce vrací jednoznakový řetězec přečtený ze souboru *«fp»*. Pokud při čtení došlo k chybě nebo byl dosažen konec souboru, vrací funkce hodnotu `false`.

FGetS() na straně 290.

FGetS

Funkce pro práci se soubory

Přečtení jedné řádky textu ze souboru

§ `string FGetS(integer «fp», integer «délka»)`

Funkce ze souboru *«fp»* přečte jednu řádku o maximální délce (*«délka»* − 1) znaků. Pokud při čtení dojde k chybě, vrací funkce hodnotu `false`.

FGetS() na straně 290.

Ukázky použití naleznete na straně 181.

FGetSS

Funkce pro práci se soubory

Přečtení řádky ze souboru a odstranění všech HTML a PHP tagů

§ `string FGetSS(integer «fp», integer «délka»)`

Funkce přečte jednu řádku ze souboru podobně jako `FGetS()`. Rozdíl je v tom, že tato funkce se pokusí z textu odstranit všechny HTML a PHP tagy.

FGetS() na straně 290.

File

Funkce pro práci se soubory

Načtení celého souboru do pole

§ `array File(string «soubor», integer «použít_include_path»)`

Funkce načte celý *«soubor»* do pole. Každý prvek pole obsahuje jednu řádku *«souboru»*.
Pokud jako hodnotu posledního nepovinného parametru *«použít_include_path»* uvedeme `true`, bude se soubor pro otevření hledat i v adresářích uvedených v direktivě `include_path` v konfiguračním souboru `php3.ini`.

`ReadFile()` na straně 404.

Ukázky použití naleznete na straně 198.

File_Exists

Funkce pro práci se soubory

Zjištění, zda daný soubor existuje

§ `integer File_Exists(string` *«soubor»*`)`

Funkce vrací `true`, pokud *«soubor»* existuje. Pokud *«soubor»* neexistuje, vrací `false`.

`ClearStatCache()` na straně 267.

Ukázky použití naleznete na straně 181.

FileATime

Funkce pro práci se soubory

Zjištění času posledního přístupu k souboru

§ `integer FileATime(string` *«soubor»*`)`

Funkce vrací čas posledního přístupu k *«souboru»*. Čas je udán jako počet sekund od 1. ledna 1970. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

`Date()` na straně 272.

FileCTime

Funkce pro práci se soubory

Zjištění času vytvoření souboru

§ `integer FileCTime(string` *«soubor»*`)`

Funkce vrací čas vytvoření *«souboru»*. Čas je udán jako počet sekund od 1. ledna 1970. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

`Date()` na straně 272.

FileGroup Funkce pro práci se soubory

Zjištění skupiny uživatelů, které soubor patří

§ `integer FileGroup(string` *«soubor»*`)`

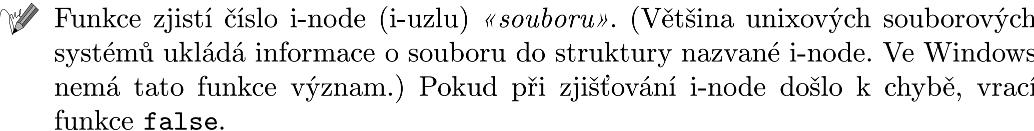
Funkce vrací číslo skupiny uživatelů (GID), které *«soubor»* patří.

`ChGrp()` na straně 265.

FileINode Funkce pro práci se soubory

Vrací číslo i-node souboru

§ `integer FileINode(string` *«soubor»*`)`



Funkce zjistí číslo i-node (i-uzlu) *«souboru»*. (Většina unixových souborových systémů ukládá informace o souboru do struktury nazvané i-node. Ve Windows nemá tato funkce význam.) Pokud při zjišťování i-node došlo k chybě, vrací funkce `false`.

FileMTime Funkce pro práci se soubory

Zjištění času poslední modifikace souboru

§ `integer FileMTime(string` *«soubor»*`)`

Funkce vrací čas poslední modifikace *«souboru»*. Čas je udán jako počet sekund od 1. ledna 1970. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

`Date()` na straně 272.

FileOwner Funkce pro práci se soubory

Zjištění uživatele, kterému soubor patří

§ `integer FileOwner(string` *«soubor»*`)`

Funkce vrací číslo uživatele (UID), kterému *«soubor»* patří.

`ChOwn()` na straně 266.

FilePerms

Funkce pro práci se soubory

Zjištění přístupových práv k souboru

§ `integer FilePerms(string` *«soubor»*`)`

Funkce vrací přístupová práva k *«souboru»*.

`ChMod()` na straně 266.

FileSize

Funkce pro práci se soubory

Zjištění velikosti souboru

§ `integer FileSize(string` *«soubor»*`)`

Funkce zjistí velikost *«souboru»* v bajtech. Pokud při zjišťování velikosti souboru dojde k chybě, vrací funkce `false`.

FileType

Funkce pro práci se soubory

Zjištění typu souboru

§ `string FileType(string` *«soubor»*`)`

Funkce zjišťuje typ souboru. Jako výsledek můžeme získat jeden z následujících řetězců:

`fifo`	zásobník;
`char`	znakové zařízení;
`dir`	adresář;
`block`	blokové zařízení;
`link`	symbolický odkaz;
`file`	soubor;
`unknown`	neznámý typ.

Pokud při zjišťování typu souboru dojde k chybě, vrací funkce hodnotu `false`.

Floor

Matematická funkce

Zaokrouhlení desetinného čísla dolů

§ `integer Floor(double «výraz»)`

Zaokrouhlí *«výraz»* na nejbližší nižší celé číslo. Pokud jako *«výraz»* použijeme celé číslo, nemá funkce žádný efekt.

`Ceil()` na straně 264 a `Round()` na straně 406.

Flush

Funkce pro práci s textovými řetězci

Vyprázdnění výstupního bufferu

§ `void Flush(void)`

Funkce vyprázdní všechny výstupní buffery PHP. Výsledkem je okamžité odeslání dosavadního výstupu skriptu do prohlížeče uživatele.

FOpen

Funkce pro práci se soubory

Funkce otevře soubor nebo URL-adresu

§ `integer FOpen(string «soubor», string «mód»,`
`integer «použít_include_path»)`

Funkce otevře *«soubor»* a vrací ukazatel (*«fp»*), pomocí kterého můžeme se souborem pracovat v dalších funkcích.

Pokud *«soubor»* začíná `http://`, odešle se požadavek na dané URL pomocí protokolu HTTP/1.0. Funkce pak vrací *«fp»* na začátek odpovědi serveru.

Obdobně, pokud *«soubor»* začíná na `ftp://`, otevře se spojení s FTP-serverem a je vrácen *«fp»* na požadovaný soubor.

Pomocí parametru *«mód»* určujeme, v jakém režimu chceme soubor otevřít. *«mód»* se zadává jako řetězec složený z několika písmen, kde má každé písmeno speciální význam:

- `r` otevře soubor pro čtení;
- `w` otevře soubor pro zápis; pokud soubor již existuje, je smazán;
- `a` otevře soubor pro doplňování — zápis za stávající konec souboru;
- `r+` otevře soubor pro čtení a zápis (soubor již musí existovat);
- `w+` otevře soubor pro čtení a zápis; pokud soubor již existuje, je smazán;
- `a+` otevře soubor pro doplňování a čtení.

Pokud jako hodnotu posledního nepovinného parametru *«použít_include_path»* uvedeme `true`, bude se soubor pro otevření hledat i v adresářích uvedených v direktivě `include_path` v konfiguračním souboru `php3.ini`.

Jestliže se soubor nepodaří otevřít, vrací funkce `false`.

`FClose()` na straně 289, `POpen()` na straně 398 a `FSockOpen()` na straně 297.

Ukázky použití naleznete na stranách 181, 197, 198.

FPassThru

Funkce pro práci se soubory

Zapíše zbývající obsah soubor na standardní výstup

§ `integer FPassThru(integer` *«fp»*`)`

Funkce zapíše zbývající obsah souboru na standardní výstup a soubor *«fp»* uzavře. Pokud při provádění operace dojde k chybě, vrací funkce hodnotu `false`.

`FClose()` na straně 289.

Ukázky použití naleznete na stranách 197, 198.

FPutS

Funkce pro práci se soubory

Zapíše řetězec do souboru

§ `integer FPutS(integer` *«fp»*`, string` *«řetězec»*`, integer` *«délka»*`)`

Funkce zapíše *«řetězec»* do souboru *«fp»*. Pokud použijeme nepovinný parametr *«délka»*, je z řetězce zapsáno pouze *«délka»* znaků.

Pokud použijeme parametr *«délka»*, je ignorována konfigurační direktiva `magic_quote_runtime` a z *«řetězce»* nejsou před zapsáním odstraňována zpětná lomítka.

Funkce je synonymem k funkci `FWrite()`.

`FWrite()` na straně 298, `FOpen()` na straně 294, `POpen()` na straně 398 a `FSockOpen()` na straně 297.

Ukázky použití naleznete na stranách 181, 198.

FRead

Funkce pro práci se soubory

Binární čtení ze souboru

§ `string FRead(integer «fp», integer «délka»)`

Funkce ze souboru *«fp»* binárně přečte *«délka»* bajtů. Čtení skončí po přečtení *«délka»* bajtů nebo po dosažení konce souboru.

Následující ukázka přečte celý soubor a uloží jej do jedné řetězcové proměnné:

```
$soubor = "prihlasky.dat";
$fp = FOpen($soubor, "r");
$obsah = FRead($fp, FileSize($soubor));
FClose($fp);
```

`FOpen()` na straně 294, `FClose()` na straně 289, `FWrite()` na straně 298, `FGetS()` na straně 290 a `FGetSS()` na straně 290.

FrenchToJD

Funkce pro práci s daty různých kalendářů

Převod data francouzského republikového kalendáře na juliánské datum

§ `integer FrenchToJD(integer «měsíc», integer «den», integer «rok»)`

Funkce převede datum francouzského republikového kalendáře na juliánské datum. Funkce akceptuje data zadaná od roku 1 do roku 14 (v gregoriánském kalendáři od 22. září 1792 do 22. září 1806).

`JDToFrench()` na straně 342.

FSeek

Funkce pro práci se soubory

Nastavení aktuální pozice v souboru

§ `integer FSeek(integer «fp», integer «pozice»)`

Funkce nastaví aktuální pozici pro čtení/zápis souboru na *«pozici»*. Pozice se přitom udává jako počet bajtů od začátku souboru.

Funkce vrací 0, pokud se podařilo nastavit požadovanou pozici v souboru. V případě, že pozici nastavit nešlo, vrací funkce −1. Poznamenejme, že nastavení aktuální pozice za konec souboru není považováno za chybu.

Funkci nelze použít na soubory otevřené pomocí HTTP nebo FTP spojení.

`FTell()` na straně 298 a `Rewind()` na straně 405.

Ukázky použití naleznete na stranách 204, 205.

FSockOpen Síťové funkce

Otevření socketu

§ `integer FSockOpen(string` *«počítač»*`, integer` *«port»*`,`
`integer` *«chybový_kód»*`, string` *«chybové_hlášení»*`)`

Funkce vrací ukazatel *«fp»* na otevřený socket. S takto vytvořeným síťovým spojením se pracuje stejně jako se souborem. Povinnými parametry funkce jsou jméno *«počítače»* a *«portu»*, ke kterému se připojujeme. Jako *«počítač»* můžeme uvést IP-adresu nebo doménovou adresu.

U funkce můžeme použít dva nepovinné parametry, které slouží k předání chybového kódu a textu chybové zprávy v případě, že se nepodaří vytvořit požadované spojení. Tyto parametry by měly být funkci předány odkazem. Pokud se spojení nepodaří vytvořit, vrací funkce navíc hodnotu `false`.

Jednoduchý kód, který zjistí typ WWW-serveru používaný na určitém serveru:

```
$host = "www.seznam.cz";                              // adresa serveru
$port = 80;                                           // port serveru
$fp = FSockOpen($host, $port);                        // otevření spojení
if ($fp):
    FPutS($fp, "HEAD / HTTP/1.0\n\n");                // HTTP požadavek
    while ($line=FGetS($fp,128)):                     // čtení odpovědi
        if (EReg("^Server:.*$", $line)):              // hledání HTTP hlavičky
            echo $line;                               // vytištění verze serveru
            break;                                    // ukončení hledání
        endif;
    endwhile;
endif;
FClose($fp);
```

`FOpen()` na straně 294, `FGetS()` na straně 290, `FGetSS()` na straně 290, `FWrite()` na straně 298, `File()` na straně 290, `ReadFile()` na straně 404 a `FPassThru()` na straně 295.

FTell

Funkce pro práci se soubory

Zjištění aktuální pozice v souboru

§ `integer FTell(integer «fp»)`

Funkce vrací aktuální pozici v souboru *«fp»*. Pokud při provádění funkce došlo k chybě, vrací funkce `false`.

`FSeek()` na straně 296 a `Rewind()` na straně 405.

FWrite

Funkce pro práci se soubory

Zapíše řetězec do souboru

§ `integer FWrite(integer «fp», string «řetězec», integer «délka»)`

Funkce zapíše *«řetězec»* do souboru *«fp»*. Pokud použijeme nepovinný parametr *«délka»*, je z řetězce zapsáno pouze *«délka»* znaků.

Pokud použijeme parametr *«délka»*, je ignorována konfigurační direktiva `magic_quote_runtime` a z *«řetězce»* nejsou před zapsáním odstraňována zpětná lomítka.

`FPutS()` na straně 295, `FOpen()` na straně 294, `POpen()` na straně 398 a `FSock-Open()` na straně 297.

Get_Browser

Konfigurace a informace o PHP

Zjistí důležité informace o prohlížeči uživatele

§ `object Get_Browser(string «prohlížeč»)`

Funkce zjistí důležité parametry prohlížeče, kterým je přistupováno k právě zpracovávanému skriptu. Informace jsou vráceny jako členské proměnné objektu. Jaké členské proměnné budou definovány, záleží na obsahu souboru `browscap.ini`, kde jsou definovány parametry jednotlivých prohlížečů. Cestu k tomuto souboru musíme nastavit pomocí konfigurační direktivy `browscap`.

Pokud funkci voláme bez parametrů, získáme informace o prohlížeči, který poslal požadavek (tj. z HTTP hlavičky `User-Agent`). Funkci však můžeme volat i s parametrem, který obsahuje identifikační řetězec prohlížeče. Pokud při volání funkce dojde k chybě, vrací hodnotu `false`.

Mezi nejčastější členské proměnné, které jsou definovány na základě `browscap.ini`, patří:

`parent`	označení prohlížeče, ze kterého je daný prohlížeč odvozen;
`browser`	jméno prohlížeče;
`version`	verze prohlížeče;
`majorver`	hlavní číslo verze;
`minorver`	číslo podverze;
`platform`	platforma, na které je prohlížeč spuštěn;
`tables`	obsahuje `true`, pokud prohlížeč podporuje tabulky;
`frames`	obsahuje `true`, pokud prohlížeč podporuje rámy;
`javaapplets`	obsahuje `true`, pokud prohlížeč podporuje Java-aplety;
`javascript`	obsahuje `true`, pokud prohlížeč podporuje JavaScript;
`vbscript`	obsahuje `true`, pokud prohlížeč podporuje VB Script;
`cookies`	obsahuje `true`, pokud prohlížeč podporuje cookies;
`activexcontrols`	obsahuje `true`, pokud prohlížeč podporuje prvky ActiveX.

Použití této funkce umožňuje snadou tvorbu skriptů, které svůj výstup přizpůsobují schopnostem prohlížeče. Tento postup je dnes bohužel mnohdy nutný, protože každá z firem produkujících prohlížeče implementuje standardy jako HTML, kaskádové styly či JavaScript trošku jinak. Pokud se vám tento stav zcela oprávněně nezamlouvá, podívejte se na zajímavou iniciativu za jednotné standardy na Webu — `http://www.webstandards.org`.

Následuje jednoduchá ukázka skriptu, který pro méně schopné prohlížeče formátuje údaje pomocí prostředí `<PRE>`. V ostatních prohlížečích je výstup zobrazen pomocí tabulek.

```
$browser = Get_Browser();
if ($browser->tables):
    echo "<TABLE>\n";
    «generujeme tabulku»
    echo "</TABLE>\n";
else:
    echo "<PRE>\n";
    «generujeme textový výstup»
    echo "</PRE>\n";
endif;
```

Ukázky použití naleznete na straně 464.

Get_Cfg_Var Konfigurace a informace o PHP

Zjištění hodnoty konfigurační proměnné

§ `string Get_Cfg_Var(string` *«jméno_proměnné»*`)`

Funkce vrací hodnotu zadané konfigurační proměnné.

Například cestu, ve které se hledají soubory pro vložení, můžeme zjistit pomocí příkazu:

```
echo Get_Cfg_Var("include_path")
```

Get_Current_User Konfigurace a informace o PHP

Vrací jméno uživatele, pod kterým je spuštěn aktuální skript

§ `string Get_Current_User(void)`

Funkce vrací jméno uživatele, pod kterým je spuštěn aktuální skript. Obvykle se jedná o uživatele, pod kterým je spuštěn WWW-server.

Get_Meta_Tags Funkce pro práci s textovými řetězci

Zjištění obsahu META-tagů v HTML souboru

§ `array Get_Meta_Tags(string` *«soubor»*`, integer` *«použít_include_path»*`)`

Funkce v daném souboru nalezne všechny tagy `<META>` a vrátí asociativní pole, které jako index obsahuje hodnoty atributu `NAME` a jako prvky obsahuje hodnoty atributu `CONTENT`.

Indexy pole jsou převedeny na malá písmena a všechny speciální znaky jsou nahrazeny podtržítkem.

Pokud jako hodnotu druhého nepovinného parametru použijeme `true`, bude se soubor hledat i v adresářích určených pomocí direktivy `include_path`.

GetAllHeaders

Funkce pracující pouze na serveru Apache

Přečtení všech HTTP-hlaviček požadavku

§ `array GetAllHeaders(void)`

Funkce vrací asociativní pole obsahující všechny hlavičky právě zpracovávaného požadavku.

Pro zobrazení všech hlaviček můžeme použít skript:

```
$headers = GetAllHeaders();
while (list($hlavicka, $hodnota) = each($headers))
    echo "$hlavicka: $hodnota<BR>\n";
```

Ukázky použití naleznete na straně 442.

GetDate

Funkce pro práci s datem a časem

Zjištění časových informací

§ `array GetDate(integer` *«čas»*`)`

Funkce vrací asociativní pole, které obsahuje informace o zadaném *«čase»*. *«Čas»* se zadává v sekundách od 1. ledna 1970. Parametr *«čas»* je nepovinný a pokud jej nepoužijeme, použije se aktuální čas. Přehled indexů asociativního pole, ve kterém je vrácen výsledek, přináší tabulka 9-3.

Index	Popis
`seconds`	Sekundy
`minutes`	Minuty
`hours`	Hodiny
`mday`	Den v měsíci (1–31)
`wday`	Den v týdnu (1–7)
`mon`	Měsíc (1–12)
`year`	Rok
`yday`	Den v roce (1–365)
`weekday`	Jméno dne v týdnu
`month`	Jméno měsíce

Tab. 9-3: Indexy asociativního pole vraceného funkcí GetDate()

`Date()` na straně 272 a `MkTime()` na straně 359.

GetEnv

Konfigurace a informace o PHP

Zjištění hodnoty proměnné prostředí

§ `string GetEnv(string` *«jméno_proměnné»*`)`

Funkce vrací hodnotu proměnné prostředí. Funkci použijeme v těch případech, kdy je proměnná prostředí překryta proměnnou s obsahem pole formuláře.

Funkce má přístup jen k těm proměnným, které má ve svém prostředí WWW-server — nemusí to být tedy všechny proměnné, které má systém normálně k dispozici.

Pokud ve skriptech potřebujeme proměnnou, která není normálně zpřístupněna, musíme její zpřístupnění povolit v konfiguraci serveru. Například u serveru Apache k tomu slouží direktiva `PassEnv`.

GetHostByAddr

Síťové funkce

Převod IP-adresy na adresu doménovou

§ `string GetHostByAddr(string` *«IP-adresa»*`)`

Funkce vrací doménovou adresu odpovídající zadané *«IP-adrese»*. Pokud při převodu došlo k chybě, je vrácena zadaná IP-adresa.

`GetHostByName()` na straně 302.

GetHostByName

Síťové funkce

Převod doménové adresy na IP

§ `string GetHostByName(string` *«doménová-adresa»*`)`

Funkce vrací IP-adresu odpovídající zadané *«doménové-adrese»*. Pokud při převodu došlo k chybě, je vrácena doménová-adresa.

`GetHostByAddr()` na straně 302 a `GetHostByNameL()` na straně 303.

GetHostByNameL

Síťové funkce

Zjištění všech IP-adres, které odpovídají jedné doménové adrese

§ `array GetHostByNameL(string` *«doménová-adresa»*`)`

Funkce vrací pole obsahující IP-adresy odpovídající zadané *«doménové-adrese»*. Pokud při převodu došlo k chybě, je vrácena doménová-adresa.

Následující skript vypíše všechny IP-adresy přiřazené stroji `www.vse.cz`:

```
$domena = "www.vse.cz";
$seznamip = GetHostByNameL($domena);
while(list(,$ip)=each($seznamip)) echo $ip."<BR>";
```

`GetHostByAddr()` na straně 302 a `GetHostByName()` na straně 302.

GetImageSize

Funkce pro práci s obrázky

Zjištění velikosti obrázku GIF, JPEG nebo PNG

§ `array GetImageSize(string` *«soubor»*`)`

Funkce zjistí velikost a typ obrázku uloženého v *«souboru»*. Výsledkem funkce je pole se čtyřmi prvky. Prvek s indexem 0 obsahuje šířku obrázku v pixelech, prvek s indexem 1 pak výšku. Prvek s indexem 2 obsahuje informaci o typu souboru: 1 pro GIF, 2 pro JPEG a 3 pro PNG. Poslední prvek, s indexem 3, obsahuje řetězec ve tvaru `height=`*«y»* `width=`*«x»*, který můžeme rovnou použít jako součást atributů tagu `<IMG>`, pokud chceme u obrázku určit jeho velikost.

Následující kód vloží do stránky obrázek i s určením jeho velikosti:

```
<?
    $soubor = "images/ferda.jpg";
    $size = GetImageSize($soubor);
?>
<IMG SRC="<?echo $soubor?>" <?echo $size[3]?>>
```

GetLastMod

Konfigurace a informace o PHP

Zjištění data poslední modifikace skriptu

§ `integer GetLastMod(void)`

Funkce vrací datum poslední modifikace skriptu jako počet sekund od 1. ledna 1970.

Na mnoha stránkách nalezneme informaci o poslední modifikaci stránky. V PHP můžeme vše zařídit zcela automaticky — na konec každé stránky stačí přidat:

```
Poslední modifikace: <?echo Date("d.m.Y", GetLastMod())?>
```

`Date()` na straně 272.

GetMXRR

Síťové funkce

Přečtení MX záznamu z DNS

§ `integer GetMXRR(string` *«doména»*`, array` *«počítače»*`, array` *«váhy»*`)`

Funkce prohledá DNS na přítomnost záznamů typu MX pro *«doménu»*. Pokud je záznam nalezen, vrací funkce `true`. V opačném případě `false`.

Seznam vyhovujících záznamů je uložen v poli *«počítače»*. Pokud je při volání funkce použit parametr *«váhy»*, jsou do něj uloženy váhy (priority) jednotlivých počítačů uvedených v DNS.

Druhý a třetí parametr by měl být funkci vždy předáván odkazem.

MX záznamy v DNS nesou informaci o tom, na které počítače se bude ukládat elektronická pošta pro danou doménu. Pro jednu doménu může být těchto počítačů několik a můžeme u nich určit jejich prioritu (0 je nejvyšší priorita).

`CheckDNSRR()` na straně 265.

GetMyINode

Konfigurace a informace o PHP

Vrací číslo i-node právě prováděného skriptu

§ `integer GetMyINode(void)`

Funkce vrací číslo i-node právě prováděného skriptu. Pokud při zjišťování dojde k chybě, vrací funkce 0.

GetMyPID

Konfigurace a informace o PHP

Zjištění čísla procesu PHP

§ `integer GetMyPID(void)`

Funkce vrací číslo procesu (PID), pod kterým právě běží PHP. Při chybě vrací `false`. Pokud PHP běží jako modul serveru, může mít více volání stejného skriptu stejné PID.

GetMyUID

Konfigurace a informace o PHP

Zjištění čísla vlastníka skriptu

§ `integer GetMyUID(void)`

Funkce vrací číslo vlastníka (UID) právě prováděného skriptu. Při chybě je vracena hodnota `false`.

GetRandMax

Matematická funkce

Nejvyšší hodnota, kterou může vrátit `Rand()`

§ `integer GetRandMax(void)`

Vrátí nejvyšší možnou hodnotu, kterou může vrátit funkce `Rand()` pro generování pseudonáhodných čísel.

`Rand()` na straně 402 a `SRand()` na straně 413.

GetType

Proměnná

Zjištění typu proměnné

§ `string GetType(mixed «proměnná»)`

Zjistí typ *«proměnné»*. Vrácená hodnota je jeden z následujících řetězců: `integer`, `double`, `string`, `array`, `class`, `object` a `unknown type` (neznámý typ).

`SetType()` na straně 409.

Ukázky použití naleznete na straně 46.

GMDate

Funkce pro práci s datem a časem

Vrací zformátovaný údaj o Greenwichském čase

§ `string GMDate(string «formát», integer «čas»)`

Funkce se chová stejně jako funkce `Date()`. Rozdíl je pouze v tom, že časový údaj je automaticky převeden na centrální Greenwichský čas (GMT). Přehled znaků použitelných ve formátovacím řetězci nalezneme v tabulce 9-1 na straně 272.

`Date()` na straně 272, `Time()` na straně 424, `GMMkTime()` na straně 306 a `MkTime()` na straně 359.

GMMkTime

Funkce pro práci s datem a časem

Získání časového údaje z Greenwichského času

§
```
integer GMMkTime(integer «hodina», integer «minuta»,
          integer «sec», integer «měsíc»,
          integer «den», integer «rok»)
```

Pracuje stejně jako `MkTime()`, s tím rozdílem, že parametry jsou považovány za Greenwichský čas (GMT).

`MkTime()` na straně 359, `Date()` na straně 272 a `GMDate()` na straně 306.

GregorianToJD Funkce pro práci s daty různých kalendářů

Převod data gregoriánského kalendáře na juliánské datum

§ `integer GregorianToJD(integer` *«měsíc»*`, integer` *«den»*`, integer` *«rok»*`)`

Funkce převede datum gregoriánského kalendáře na juliánské datum. Funkce akceptuje data zadaná od roku 4 714 př. n. l. do roku 9 999 n. l. Prakticky se však gregoriánský kalendář používá od 15. října 1582.

`JDToGregorian()` na straně 342.

Header Funkce pro práci s protokolem HTTP

Zaslání HTTP hlavičky

§ `integer Header(string` *«hlavička»*`)`

Funkce slouží k zaslání určité HTTP hlavičky klientovi v odpovědi. Popis HTTP hlaviček naleznete v kapitole věnované protokolu HTTP. Funkci `Header()` musíme volat ještě předtím, než se na výstupu skriptu objeví HTML nebo výstupy PHP.

Velmi často potřebujeme uživatele automaticky přesměrovat na jinou stránku:

```
Header("Location: http://www.nekde.jinde.cz");
```

Ukázky použití naleznete na stranách 246, 442, 446.

HexDec Matematická funkce

Převod šestnáctkového čísla na desítkové

§ `integer HexDec(string` *«šestnáctkové číslo»*`)`

Převede *«šestnáctkové číslo»* uložené v řetězci na desítkové číslo. Šestnáctkové číslo může být v rozsahu od 0 do `7fffffff`. Pokus o převod jiných čísel vrátí 0.

`DecHex()` na straně 280, `BinDec()` na straně 264, `DecBin()` na straně 280, `OctDec()` na straně 374 a `DecOct()` na straně 280.

HighLight_File

Ostatní funkce

Zobrazí soubor se skriptem s použitím zvýrazněné syntaxe

§ `void HighLight_File(string` *«jméno souboru»*`)`

Výsledkem této funkce je vypsání HTML-kódu, který reprezentuje obsah zadaného souboru s použitím zvýrazněné syntaxe. Funkce je synonymem staršího příkazu `Show_Source`.

`HighLight_String()` na straně 308.

HighLight_String

Ostatní funkce

Zobrazí řetězec jako skript s použitím zvýrazněné syntaxe

§ `void HighLight_String(string` *«jméno souboru»*`)`

Výsledkem této funkce je vypsání HTML-kódu, který reprezentuje obsah řetězce s použitím zvýrazněné syntaxe na příkazy PHP.

`HighLight_File()` na straně 308.

HTMLEntities

Funkce pro práci s textovými řetězci

Převod všech možných znaků na znakové entity HTML

§ `string HTMLEntities(string` *«řetězec»*`)`

Převede všechny znaky ze znakové sady ISO 8859-1 na příslušné znakové entity HTML. Kromě znakových entit, které převádí funkce `HTMLSpecialChars()`, převádí i znakové entity uvedené v tabulce 9-4 na následující straně.

`HTMLSpecialChars()` na straně 308 a `NL2BR()` na straně 373.

HTMLSpecialChars

Funkce pro práci s textovými řetězci

Převod speciálních znaků na znakové entity HTML

§ `string HTMLSpecialChars(string` *«řetězec»*`)`

Kód	Entita	Znak
160		
161	¡	¡
162	¢	¢
163	£	£
164	¤	¤
165	¥	¥
166	¦	¦
167	§	§
168	¨	¨
169	©	©
170	ª	ª
171	«	«
172	¬	¬
173		-
174	®	®
175	¯	¯
176	°	°
177	±	±
178	²	²
179	³	³
180	´	´
181	µ	µ
182	¶	¶
183	·	·
184	¸	¸
185	¹	¹
186	º	º
187	»	»
188	¼	¼
189	½	½
190	¾	¾
191	¿	¿
192	À	À
193	Á	Á
194	Â	Â
195	Ã	Ã
196	Ä	Ä
197	Å	Å
198	Æ	Æ
199	Ç	Ç
200	È	È
201	É	É
202	Ê	Ê
203	Ë	Ë
204	Ì	Ì
205	Í	Í
206	Î	Î
207	Ï	Ï

Kód	Entita	Znak
208	Ð	Ð
209	Ñ	Ñ
210	Ò	Ò
211	Ó	Ó
212	Ô	Ô
213	Õ	Õ
214	Ö	Ö
215	×	×
216	Ø	Ø
217	Ù	Ù
218	Ú	Ú
219	Û	Û
220	Ü	Ü
221	Ý	Ý
222	Þ	Þ
223	ß	ß
224	à	à
225	á	á
226	â	â
227	ã	ã
228	ä	ä
229	å	å
230	æ	æ
231	ç	ç
232	è	è
233	é	é
234	ê	ê
235	ë	ë
236	ì	ì
237	í	í
238	î	î
239	ï	ï
240	ð	ð
241	ñ	ñ
242	ò	ò
243	ó	ó
244	ô	ô
245	õ	õ
246	ö	ö
247	÷	÷
248	ø	ø
249	ù	ù
250	ú	ú
251	û	û
252	ü	ü
253	ý	ý
254	þ	þ
255	ÿ	ÿ

Tab. 9-4: Znakové entity pro kódování ISO Latin 1

Převede všechny speciální znaky HTML ('<', '>', '"' a '&') na znakové entity ('<', '>', '"' a '&'). Funkce nepřevádí žádné jiné znaky, jako např. písmena s diakritikou. Pro tyto účely slouží funkce `HTMLEntities()`.

Funkci bychom měli použít např. tehdy, když do stránky zahrnujeme text odeslaný uživatelem z formuláře. Při převedení „nebezpečných" znaků na znakové entity nemá uživatel možnost narušit náš HTML kód a způsobit tak špatné formátování.

`HTMLEntities()` na straně 308 a `NL2BR()` na straně 373.

Ukázky použití naleznete na straně 199.

ImageArc

Funkce pro práci s obrázky

Nakreslení části elipsy

§
```
integer ImageArc(integer «obrázek», integer «x», integer «y»,
            integer «a», integer «b», integer «α», integer «β»,
            integer «barva»)
```

Funkce do *«obrázku»* nakreslí *«barvou»* část elipsy. Elipsa je přitom určena souřadnicemi svého středu (*«x»* a *«y»*) a průměrem hlavní a vedlejší poloosy (*«a»* a *«b»*). Elipsa je vykreslena jen mezi počátečním a koncovým bodem, které jsou určeny pomocí úhlů *«α»* a *«β»* zadaných ve stupních. Pokud chceme nakreslit celou elipsu, zadáme úhel od 0 do 359.

ImageChar

Funkce pro práci s obrázky

Nakreslení znaku

§
```
integer ImageChar(integer «obrázek», integer «font»,
            integer «x», integer «y»,
            string «znak», integer «barva»)
```

Funkce na daných souřadnicích zobrazí první znak řetězce *«znak»*. K zobrazení je použita *«barva»* a *«font»*. Pokud je *«font»* 1–5, použije se vestavěný font, další fonty mohou být zpřístupněny pomocí funkce `ImageLoadFont()`.

`ImageLoadFont()` na straně 317.

ImageCharUp

Funkce pro práci s obrázky

Nakreslení znaku ve vertikálním směru

§ `integer ImageChar(integer` *«obrázek»*`, integer` *«font»*`,`
`integer` *«x»*`, integer` *«y»*`,`
`string` *«znak»*`, integer` *«barva»*`)`

Funkce na daných souřadnicích zobrazí první znak řetězce *«znak»* otočený o 90° proti směru hodinových ručiček. K zobrazení je použita *«barva»* a *«font»*. Pokud je *«font»* 1–5, použije se vestavěný font, další fonty mohou být zpřístupněny pomocí funkce `ImageLoadFont()`.

`ImageLoadFont()` na straně 317.

ImageColorAllocate

Funkce pro práci s obrázky

Alokování a vytvoření barvy pro obrázek

§ `integer ImageColorAllocate(integer` *«obrázek»*`,`
`integer` *«R»*`, integer` *«G»*`, integer` *«B»*`)`

Funkce v *«obrázku»* alokuje novou barvu, jejíž jednotlivé barevné složky v modelu RGB jsou *«R»*, *«G»*, *«B»* — rozsah složek barev je od 0 do 255. Každou barvu, kterou chceme v obrázku použít, musíme nejprve vytvořit pomocí této funkce. Na barvu se pak odvoláváme pomocí indexu do palety obrázku, který vrací tato funkce.

ImageColorAt

Funkce pro práci s obrázky

Zjištění indexu barvy daného bodu

§ `integer ImageColorAt(integer` *«obrázek»*`, integer` *«x»*`, integer` *«y»*`)`

Funkce vrací index barvy bodu, který má v *«obrázku»* souřadnice *«x»* a *«y»*.

ImageColorClosest

Funkce pro práci s obrázky

Získání indexu barvy, která je nejbližší zadané barvě

§
```
integer ImageColorClosest(integer «obrázek»,
          integer «R», integer «G», integer «B»)
```

Funkce vrací index barvy *«obrázku»*, která je nejbližší barvě zadané pomocí barevných složek *«R»* (červená), *«G»* (zelená) a *«B»* (modrá). Při výběru nejbližší barvy je vybírána barva, která je zadané barvě nejblíže v třírozměrném prostoru tvořeném jednotlivými barevnými složkami.

ImageColorExact

Funkce pro práci s obrázky

Zjištění indexu zadané barvy

§
```
integer ImageColorExact(integer «obrázek»,
          integer «R», integer «G», integer «B»)
```

Funkce vrací index barvy, jejíž jednotlivé složky jsou *«R»*, *«G»* a *«B»*. Pokud taková barva v *«obrázku»* neexistuje, vrací funkce hodnotu −1.

ImageColorsForIndex

Funkce pro práci s obrázky

Zjištění barevných složek dané barvy

§
```
array ImageColorsForIndex(integer «obrázek», integer «barva»)
```

Funkce pro danou *«barvu»* obrázku vrátí její barevné složky v asociativním poli s indexy `red`, `green` a `blue`.

ImageColorResolve

Funkce pro práci s obrázky

Zjištění indexu zadané barvy nebo nejbližší barvy

§
```
integer ImageColorResolve(integer «obrázek»,
          integer «R», integer «G», integer «B»)
```

Funkce vrací index barvy, jejíž jednotlivé složky jsou *«R»*, *«G»* a *«B»*. Pokud taková barva v *«obrázku»* neexistuje, vrací funkce index barvy, která je zadané barvě nejblíže v třírozměrném prostoru určeném jednotlivými barevnými složkami.

ImageColorSet

Funkce pro práci s obrázky

Nastavení položky palety na určitou barvu

§ `integer ImageColorSet(integer` *«obrázek»*`, integer` *«barva»*`,`
`integer` *«R»*`, integer` *«G»*`, integer` *«B»*`)`

Funkce změní barvu určité položky palety (určené číslem *«barvy»*). Tímto způsobem můžeme jednou rychlou operací změnit barvu i velké části obrázku.

ImageColorsTotal

Funkce pro práci s obrázky

Zjištění počtu barev v obrázku

§ `integer ImageColorsTotal(integer` *«obrázek»*`)`

Funkce vrací počet barev (položek palety), které obsahuje daný *«obrázek»*.

ImageColorTransparent

Funkce pro práci s obrázky

Nastavení transparentní (průhledné) barvy obrázku

§ `integer ImageColorTransparent(integer` *«obrázek»*`, integer` *«barva»*`)`

Funkce v *«obrázku»* nastaví *«barvu»* jako transparentní. *«Barva»* je přitom index palety vrácený pomocí funkce `ImageColorAllocate()`.

`ImageColorAllocate()` na straně 311.

Ukázky použití naleznete na straně 316.

ImageCopyResized

Funkce pro práci s obrázky

Kopírování části obrázku se změnou velikosti

§ `integer ImageCopyResized(integer` *«cobrázek»*`, integer` *«zobrázek»*`,`
`integer` *«cx»*`, integer` *«cy»*`,`
`integer` *«zx»*`, integer` *«zy»*`,`
`integer` *«cšířka»*`, integer` *«cvýška»*`,`
`integer` *«zšířka»*`, integer` *«zvýška»*`)`

Funkce zkopíruje pravoúhlou oblast z obrázku *«zobrázek»* do obrázku *«cobrázek»*. Pokud se liší šířky a výšky zdrojové a cílové oblasti, je přenášená část obrázku příslušným způsobem zvětšena/zmenšena. Funkci můžeme používat i pro kopírování oblastí v rámci jednoho obrázku (*«cobrázek»* = *«zobrázek»*). Pokud se však zdrojová a cílová oblast překrývají, nemusí funkce pracovat správně.

ImageCreate Funkce pro práci s obrázky

Vytvoření obrázku

§ `integer ImageCreate(integer «šířka», integer «výška»)`

Funkce vytvoří prázdný obrázek o dané *«šířce»* a *«výšce»*. Vrácen je identifikátor obrázku, který se používá v ostatních funkcích pro práci s obrázky.

Ukázky použití naleznete na stranách 185, 186.

ImageCreateFromGIF Funkce pro práci s obrázky

Vytvoření obrázku podle obrázku ze souboru nebo z určitého URL

§ `integer ImageCreateFromGIF(string «soubor»)`

Funkce vrací identifikátor obrázku uloženého v *«souboru»* nebo na URL adrese obsažené v parametru *«soubor»*. Uložený obrázek musí být ve formátu GIF.

ImageDashedLine Funkce pro práci s obrázky

Nakreslení čárkované čáry

§
```
integer ImageDashedLine(integer «obrázek»,
            integer «x1», integer «y1»,
            integer «x2», integer «y2», integer «barva»)
```

Funkce nakreslí čárkovanou čáru mezi body určenými souřadnicemi *«x1»*, *«y1»* a *«x2»*, *«y2»*. Úsečka je nakreslena *«barvou»*.

`ImageLine()` na straně 317.

ImageDestroy

Funkce pro práci s obrázky

Uvolnění obrázku z paměti

§ `integer ImageDestroy(integer` *«obrázek»*`)`

Funkce uvolní veškerou paměť spojenou s *«obrázkem»*.

ImageFill

Funkce pro práci s obrázky

Vyplnění oblasti

§ `integer ImageFill(integer` *«obrázek»*`, integer` *«x»*`, integer` *«y»*`,`
`integer` *«barvou»*`)`

Funkce vyplní *«barvou»* ohraničenou oblast, uvnitř které leží bod o souřadnicích *«x»* a *«y»*.

ImageFilledPolygon

Funkce pro práci s obrázky

Nakreslení polygonu vyplněného barvou

§ `integer ImageFilledPolygon(integer` *«obrázek»*`, array` *«vrcholy»*`,`
`integer` *«počet_vrcholů»*`, integer` *«barva»*`)`

Funkce nakreslí vybarvený polygon, jehož souřadnice vrcholů jsou uloženy v poli *«vrcholy»*. V poli jsou postupně uloženy souřadnice x a y jednotlivých vrcholů (`vrcholy[0]=`x_0, `vrcholy[1]`$=y_0$, `vrcholy[2]=`x_1, `vrcholy[3]`$=y_1$, …).

ImageFilledRectangle

Funkce pro práci s obrázky

Nakreslení obdélníku vyplněného barvou

§ `integer ImageFilledRectangle(integer` *«obrázek»*`,`
`integer` *«x1»*`, integer` *«y1»*`,`
`integer` *«x2»*`, integer` *«y2»*`, integer` *«barva»*`)`

Funkce nakreslí obdélník určený souřadnicemi protilehlých vrcholů *«x1»*, *«y1»* a *«x2»*, *«y2»*. Obdélník je vyplněn zadanou *«barvou»*.

ImageFillToBorder Funkce pro práci s obrázky

Vyplnění oblasti jejíž hranice je dána barvou

§ `integer ImageFillToBorder(integer «obrázek», integer «x», integer «y», integer «barva_hranice», integer «barva»)`

Funkce vyplní *«barvou»* oblast, uvnitř které leží bod o souřadnicích *«x»* a *«y»*. Hranice vyplněné oblasti jsou dány barvou *«barva_hranice»*.

ImageFontHeight Funkce pro práci s obrázky

Zjištění velikosti písma v bodech

§ `integer ImageFontHeight(integer «font»)`

Funkce vrací velikost znaků daného *«fontu»* v bodech (pixelech).

`ImageFontWidth()` na straně 316.

ImageFontWidth Funkce pro práci s obrázky

Zjištění šířky písma v bodech

§ `integer ImageFontWidth(integer «font»)`

Funkce vrací šířku znaků daného *«fontu»* v bodech (pixelech).

`ImageFontHeight()` na straně 316.

Ukázky použití naleznete na straně 186.

ImageGIF Funkce pro práci s obrázky

Zapsání obrázku na výstup nebo do souboru

§ `integer ImageGIF(integer «obrázek», string «soubor»)`

Funkce zapíše *«obrázek»* do *«souboru»*. Pokud parametr *«soubor»* vynecháme, je obrázek zapsán na standardní výstup skriptu. Tímto způsobem může náš skript přímo generovat obrázky; stačí, když na začátku náš skript odešle hlavičku `Content-type: image/gif`. Vygenerovaný obrázek je formátu GIF87a. Pokud jsme v obrázku definovali transparentní barvu pomocí `ImageColorTransparent()`, bude obrázek ve formátu GIF89a.

ImageInterlace
Funkce pro práci s obrázky

Zapnutí/vypnutí prokládání obrázku

§ `integer ImageInterlace(integer` *«obrázek»*`, integer` *«prokládání»*`)`

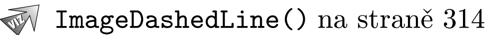
Funkce určuje, zda bude daný *«obrázek»* prokládaný. Parametr *«prokládání»* může mít hodnotu `true` (prokládaný obrázek) nebo `false` (neprokládaný obrázek). Prokládané obrázky jsou pro uživatele příjemnější, protože již po přenesení první části obrázku je vidět přibližný obsah obrázku.

ImageLine
Funkce pro práci s obrázky

Nakreslení čáry

§ `integer ImageLine(integer` *«obrázek»*`, integer` *«x1»*`, integer` *«y1»*`,`
`integer` *«x2»*`, integer` *«y2»*`, integer` *«barva»*`)`

Funkce nakreslí čáru mezi body určenými souřadnicemi *«x1»*, *«y1»* a *«x2»*, *«y2»*. Čára je nakreslena *«barvou»*.


`ImageDashedLine()` na straně 314.

ImageLoadFont
Funkce pro práci s obrázky

Nahrání nového fontu ze souboru

§ `integer ImageLoadFont(string` *«soubor»*`)`

Funkce nahraje font ze *«souboru»* a vrátí jeho identifikátor. Takto zavedený font můžeme používat ve všech funkcích, které umožňují pracovat se zabudovanými fonty. Popis formátu fontů naleznete v originální dokumentaci. Kromě těchto bitmapových fontů můžeme využívat i fonty TrueType pomocí funkce `ImageTTFText()`.

ImagePolygon
Funkce pro práci s obrázky

Nakreslení polygonu

§ `integer ImagePolygon(integer` *«obrázek»*`, array` *«vrcholy»*`,`
`integer` *«počet_vrcholů»*`, integer` *«barva»*`)`

Funkce nakreslí polygon, jehož souřadnice vrcholů jsou uloženy v poli *«vrcholy»*. V poli jsou postupně uloženy souřadnice x a y jednotlivých vrcholů (`vrcholy[0]`=x_0, `vrcholy[1]`=y_0, `vrcholy[2]`=x_1, `vrcholy[3]`=y_1, ...).

ImageRectangle

Funkce pro práci s obrázky

Nakreslení obdélníku

```
integer ImageRectangle(integer «obrázek»,
          integer «x1», integer «y1»,
          integer «x2», integer «y2», integer «barva»)
```

Funkce nakreslí obdélník určený souřadnicemi protilehlých vrcholů *«x1»*, *«y1»* a *«x2»*, *«y2»*.

ImageSetPixel

Funkce pro práci s obrázky

Nakreslení jednoho bodu

```
integer ImageSetPixel(integer «obrázek»,
          integer «x», integer «y», integer «barva»)
```

Funkce na souřadnicích *«x»* a *«y»* zobrazí bod dané *«barvy»*.

ImageString

Funkce pro práci s obrázky

Vypsání textového řetězce

```
integer ImageString(integer «obrázek», integer «font»,
          integer «x», integer «y»,
          string «text», integer «barva»)
```

Funkce na daných souřadnicích zobrazí *«text»*. K zobrazení je použita *«barva»* a *«font»*. Pokud je *«font»* 1–5, použije se vestavěný font; další fonty mohou být zpřístupněny pomocí funkce `ImageLoadFont()`.

`ImageLoadFont()` na straně 317.

Ukázky použití naleznete na straně 186.

ImageStringUp

Funkce pro práci s obrázky

Vypsání textového řetězce ve vertikálním směru

§ `integer ImageString(integer` *«obrázek»*`, integer` *«font»*`,`
`integer` *«x»*`, integer` *«y»*`,`
`string` *«text»*`, integer` *«barva»*`)`

Funkce na daných souřadnicích zobrazí *«text»* otočený o 90° proti směru hodinových ručiček. K zobrazení je použita *«barva»* a *«font»*. Pokud je *«font»* 1–5, použije se vestavěný font, další fonty mohou být zpřístupněny pomocí funkce `ImageLoadFont()`.

`ImageLoadFont()` na straně 317.

ImageSX

Funkce pro práci s obrázky

Zjištění šířky obrázku

§ `integer ImageSX(integer` *«obrázek»*`)`

Funkce vrací šířku *«obrázku»* v pixelech.

`ImageSY()` na straně 319.

ImageSY

Funkce pro práci s obrázky

Zjištění výšky obrázku

§ `integer ImageSY(integer` *«obrázek»*`)`

Funkce vrací výšku *«obrázku»* v pixelech.

`ImageSX()` na straně 319.

ImageTTFBBox

Funkce pro práci s obrázky

Zjištění plochy, kterou zabere text zobrazený TrueType fontem

§ `array ImageTTFBBox(integer` *«velikost»*`, integer` *«úhel»*`,`
`string` *«font»*`, string` *«text»*`)`

Funkce vrací velikost pravoúhlé plochy, která je potřebná pro zobrazení *«textu»* pomocí TrueType písma. Písmo je určeno parametrem *«font»*, který by měl obsahovat jméno souboru, ve kterém je uložen požadovaný TrueType font. Rovněž musíme určit *«velikost»* písma v pixelech a *«úhel»*, pod kterým bude text vypsán.

Funkce vrací ohraničující oblast v poli s 8 prvky. Prvky s níže uvedeným indexem obsahují následující údaje:

0 x-souřadnice levého dolního rohu;
1 y-souřadnice levého dolního rohu;
2 x-souřadnice pravého dolního rohu;
3 y-souřadnice pravého dolního rohu;
4 x-souřadnice levého horního rohu;
5 y-souřadnice levého horního rohu;
6 x-souřadnice pravého horního rohu;
7 y-souřadnice pravého horního rohu.

`ImageTTFText()` na straně 320.

ImageTTFText

Funkce pro práci s obrázky

Zobrazení textu pomocí TrueType písma

§
```
array ImageTTFText(integer «obrázek», integer «velikost»,
          integer «úhel», integer «x», integer «y»,
          integer «barva», string «font», string «text»)
```

Funkce zobrazí daný *«text»* pomocí TrueType písma uloženého v souboru *«font»*. Text je zobrazen na souřadnicích *«x»* a *«y»* ve směru určeném *«úhlem»* zadaném ve stupních. Druhý parametr *«velikost»* určuje velikost písma v pixelech.

Souřadnice *«x»* a *«y»* se vztahují k levému dolnímu rohu prvního písmene, na rozdíl od funkce `ImageString()`, kde se vztahují k levému hornímu rohu.

«text» může obsahovat znakové sekvence UTF-8 ve formě `&#nnnn`, které slouží pro přístup ke znakům s kódem větším než 255.

Funkce vrací pole obsahující souřadnice pravoúhlé oblasti, do které se zobrazený text vejde (podobně jako u funkce `ImageTTFBBox()`).

Funkce pro svou správnou činnost vyžaduje jak knihovnu GD, tak knihovnu FreeType.

`ImageTTFBBox()` na straně 319.

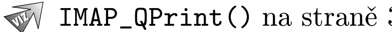
IMAP_8Bit

Podpora protokolu IMAP

Zakóduje text metodou quoted-printable

§ `string IMAP_8Bit(string` *«text»*`)`

Funkce zakóduje *«text»* metodou quoted-printable. Metoda quoted-printable převádí netisknutelné znaky (tj. znaky s kódem menším než 32 a větším než 126) na sekvenci '`=`*«kód»*', kde *«kód»* je kód příslušného znaku zapsaný v šestnáctkové soustavě. Výjimku tvoří znak '`=`', který je kvůli svému speciálnímu významu převeden na sekvenci '`=3D`'.


`IMAP_QPrint()` na straně 332.

IMAP_Append

Podpora protokolu IMAP

Přidání textové zprávy do poštovní schránky

§ `integer IMAP_Append(integer` *«spojení»*`, string` *«schránka»*`,`
`string` *«zpráva»*`, string` *«příznaky»*`)`

Funkce připojí *«zprávu»* do *«schránky»*. Schránka již musí existovat a zpráva by měla být e-mailovou zprávou ve formátu RFC 822. Pomocí nepovinného parametru *«příznaky»* můžeme nastavit příznaky ukládané zprávy.

> Funkce pouze uloží zprávu do poštovní schránky. V žádném případě neslouží k odesílání elektronické pošty — na to máme v PHP funkci `Mail()`.

IMAP_Base64

Podpora protokolu IMAP

Dekódování textu zakódovaného metodou Base64

§ `string IMAP_Base64(string` *«text»*`)`

Funkce dekóduje *«text»* zakódovaný metodou Base64.

`IMAP_Binary()` na straně 321 a `Base64_Decode()` na straně 261.

IMAP_Binary

Podpora protokolu IMAP

Zakódování textu metodou Base64

§ `string IMAP_Binary(string` *«text»*`)`

Funkce zakóduje *«text»* metodou Base64. Na rozdíl od funkce `Base64_Encode()` je výsledný text zarovnán do více řádek tak, aby jejich délka nepřekročila 60 znaků.

`IMAP_Base64()` na straně 321 a `Base64_Encode()` na straně 261.

IMAP_Body

Podpora protokolu IMAP

Přečtení těla zprávy

§ `string IMAP_Body(integer` *«spojení»*`, integer` *«číslo_zprávy»*`, integer` *«možnosti»*`)`

Funkce vrací tělo zprávy uložené v aktuální poštovní schránce pod číslem *«číslo_zprávy»*. Činnost funkce můžeme ovlivnit třetím nepovinným parametrem *«možnosti»*. Jako hodnotu příznaků můžeme vzájemně kombinovat následující konstanty:

`FT_UID`	*«číslo_zprávy»* není pořadové číslo, ale jedinečný identifikátor;
`FT_PEEK`	přečtení těla zprávy nevyvolá nastavení příznaku `\Seen` zprávy;
`FT_INTERNAL`	vrácený text bude v interním formátu, nedojde k převodu konců řádek.

Ukázky použití naleznete na stranách 223, 226.

IMAP_Check

Podpora protokolu IMAP

Kontrola aktuální poštovní schránky

§ `object IMAP_Check(integer` *«spojení»*`)`

Funkce vrací informace o aktuální poštovní schránce. V případě chyby vrací funkce hodnotu `false`. Informace jsou vráceny jako objekt, který má následující členské proměnné:

`Date`	datum zprávy;
`Driver`	ovladač;
`Mailbox`	jméno poštovní schránky;
`Nmsgs`	počet zpráv v poštovní schránce;

`Recent` počet nových zpráv od poslední kontroly schránky.

IMAP_ClearFlag_Full Podpora protokolu IMAP

Smazání příznaků u zpráv

§ `integer IMAP_ClearFlag_Full(integer` *«spojení»*`, string` *«zprávy»*`,`
`string` *«příznaky»*`, integer` *«možnosti»*`)`

Funkce u všech *«zpráv»* smaže zadané *«příznaky»*. Pokud použijeme parametr *«možnosti»* a jako jeho hodnotu použijeme konstantu `ST_UID`, neobsahuje parametr *«zprávy»* pořadová čísla zpráv, ale identifikační čísla.

IMAP_Close Podpora protokolu IMAP

Uzavření spojení s IMAP-serverem

§ `integer IMAP_Close(integer` *«spojení»*`, integer` *«možnosti»*`)`

Funkce uzavře spojení s IMAP-serverem. Pokud jako hodnotu nepovinného parametru *«možnosti»* použijeme konstantu `CL_EXPUNGE`, budou před uzavřením spojení ze schránky odstraněny všechny zprávy označené pro smazání.

Ukázky použití naleznete na straně 224.

IMAP_CreateMailBox Podpora protokolu IMAP

Vytvoření nové poštovní schránky

§ `integer IMAP_CreateMailBox(integer` *«spojení»*`, string` *«jméno»*`)`

Funkce vytvoří novou poštovní schránku daného *«jména»*. Pokud se stránku podaří vytvořit, vrací funkce `true`. V opačném případě vrací `false`.

IMAP_Delete Podpora protokolu IMAP

Označení zprávy pro smazání

§ `integer IMAP_Delete(integer` *«spojení»*`, integer` *«číslo_zprávy»*`)`

Funkce označí zprávu s daným *«číslem_zprávy»* ke smazání. Faktické smazání je provedeno až příkazem `IMAP_Expunge()`.

IMAP_DeleteMailBox

Podpora protokolu IMAP

Smazání poštovní schránky

§ `integer IMAP_DeleteMailBox(integer` *«spojení»*`, string` *«jméno_schránky»*`)`

Funkce smaže schránku. Pokud se schránku podaří smazat, vrací funkce `true`, v opačném případě `false`.

IMAP_Expunge

Podpora protokolu IMAP

Smazání všech zpráv označených pro smazání

§ `integer IMAP_Expunge(integer` *«spojení»*`)`

Funkce smaže všechny zprávy označené pomocí funkce `IMAP_Delete()` ke smazání.

IMAP_FetchBody

Podpora protokolu IMAP

Přečtení jedné části těla dopisu

§ `string IMAP_FetchBody(integer` *«spojení»*`, integer` *«číslo_zprávy»*`, string` *«část»*`, integer` *«možnosti»*`)`

Funkce přečte danou *«část»* zprávy určené pomocí jejího čísla z aktuální poštovní schránky. *«Část»* je řetězec obsahující čísla a podčísla části oddělené tečkami. Podrobnější popis a přesný význam tohoto parametru naleznete ve specifikaci protokolu IMAP [6].

Činnost funkce můžeme ovlivnit posledním nepovinným parametrem *«možnosti»*. Jako hodnotu příznaků můžeme vzájemně kombinovat následující konstanty:

- `FT_UID` *«číslo_zprávy»* není pořadové číslo, ale jedinečný identifikátor;
- `FT_PEEK` přečtení těla zprávy nevyvolá nastavení příznaku `\Seen` zprávy;
- `FT_INTERNAL` vrácený text bude v interním formátu, nedojde k převodu konců řádek.

Ukázky použití naleznete na straně 226.

IMAP_FetchHeader

Podpora protokolu IMAP

Přečtení hlavičky zprávy

§ `string IMAP_FetchHeader(integer` *«spojení»*`, integer` *«číslo_zprávy»*`,`
`integer` *«možnosti»*`)`

Funkce přečte celou hlavičku dané zprávy a vrátí ji jako zcela neupravený řetězec.

Činnost funkce můžeme ovlivnit posledním nepovinným parametrem *«možnosti»*. Jako hodnotu příznaků můžeme vzájemně kombinovat následující konstanty:

- `FT_UID` — *«číslo_zprávy»* není pořadové číslo, ale jedinečný identifikátor;
- `FT_INTERNAL` — vrácený text bude v interním formátu, nedojde k převodu konců řádek;
- `FT_PREFETCHTEXT` — spolu s hlavičkou bude přednačteno i tělo zprávy — jeho následné čtení pak bude mnohem rychlejší.

Ukázky použití naleznete na straně 223.

IMAP_FetchStructure

Podpora protokolu IMAP

Zjištění struktury zprávy

§ `object IMAP_FetchStructure(integer` *«spojení»*`, integer` *«číslo_zprávy»*`,`
`integer` *«možnosti»*`)`

Funkce zjistí informace o struktuře dané zprávy. Poslední parametr *«možnosti»* je nepovinný. Pokud jako jeho hodnotu použijeme konstantu `FT_UID`, bude funkce předpokládat, že parametr *«číslo_zprávy»* obsahuje identifikační číslo zprávy. Funkce vrací zjištěné informace jako objekt s následujícími členskými proměnnými:

- `type` — číslo udávající typ zprávy; pro snazší práci máme v PHP definovány následující konstanty:
  - `TYPETEXT` — text (MIME typ `text/*`);
  - `TYPEMULTIPART` — zpráva s více částmi (MIME typ `multipart/*`);
  - `TYPEMESSAGE` — vložená zpráva (MIME typ `message/*`);
  - `TYPEAPPLICATION` — data nějaké aplikace (MIME typ `application/*`);
  - `TYPEAUDIO` — zvuková data (MIME typ `audio/*`);

- `TYPEIMAGE` obrázek — nejčastěji např. GIF nebo JPEG (MIME typ `image/*`);
- `TYPEVIDEO` video (MIME typ `video/*`);
- `TYPEOTHER` neznámý typ.

`encoding` číslo určující typ kódování; opět máme k dispozici několik užitečných konstant:

- `ENC7BIT` 7bitová data rozdělená do řádek;
- `ENC8BIT` 8bitová data rozdělená do řádek;
- `ENCBINARY` 8bitová binární data;
- `ENCBASE64` data zakódovaná metodou Base64;
- `ENCQUOTEDPRINTABLE` data zakódovaná metodou quoted-printable;
- `ENCOTHER` neznámý typ kódování.

`ifsubtype` obsahuje `true`, pokud má zpráva i podtyp;

`subtype` řetězec obsahující podtyp zprávy;

`ifdescription` obsahuje `true`, pokud má zpráva popis;

`description` popis zprávy;

`ifid` obsahuje `true`, pokud má zpráva ID-číslo;

`id` řetězec obsahující ID-číslo zprávy;

`lines` číslo obsahující počet řádek zprávy;

`bytes` velikost zprávy v bajtech;

`ifdisposition` obsahuje `true`, pokud zpráva obsahuje hlavičku `Content-disposition`;

`disposition` obsah hlavičky `Content-disposition`, který určuje, zda má být daná část zprávy zobrazena přímo ve zprávě nebo jako příloha;

`ifdparameters` obsahuje `true`, pokud hlavička `Content-disposition` obsahuje nějaké parametry;

`dparameters` pole objektů obsahujících dvě členské proměnné `attribute` (atribut) a `value` (jeho hodnota);

`ifparameters` obsahuje `true`, pokud má zpráva nějaké parametry;

`parameters` pole objektů obsahujících dvě členské proměnné `attribute` (atribut) a `value` (jeho hodnota);

`parts` pokud zpráva obsahuje více částí, obsahuje toto pole pro každou část objekt, který má stejné členské vlastnosti jako právě popisovaný objekt (objekty jsou do sebe rekurentně zanořeny).

Ukázky použití naleznete na straně 226.

IMAP_Header Podpora protokolu IMAP

Přečtení hlavičky zprávy

§ `object IMAP_Header(integer` *«spojení»*`, integer` *«číslo_zprávy»*`, integer` *«délka_From»*`, integer` *«délka_předmětu»*`, string` *«standardní_doména»*`)`

Funkce vrací hlavičku dané zprávy. Poslední tři parametry jsou nepovinné. Funkce vrací objekt s následujícími členskými proměnnými:

`remail` (Napište mi, až zjistíte, co `remail` obsahuje.)

`date`, `Date` datum odeslání zprávy;

`subject`, `Subject`
předmět zprávy;

`in_reply_to` identifikace zprávy, na kterou aktuální zpráva odpovídá;

`message_id` identifikátor zprávy;

`newsgroups` jméno diskusní skupiny;

`followup` obsah hlavičky `Followup`;

`references` obsah hlavičky `References`;

`toaddress` plná adresa příjemce dopisu;

`to` pole obsahující pro každou adresu v poli příjemce dopisu objekt; každý objekt obsahuje členské proměnné `personal` (osobní jméno), `adl` (at-domain-list), `mailbox` (jméno poštovního účtu) a `host` (adresa poštovního serveru);

`fromaddress` úplná adresa odesílatele dopisu;

`from` pole obsahující pro každou adresu v položce odesílatele (`From`) jeden objekt;

`ccaddress` obsah pole specifikujícího adresy, na které se posílají kopie zprávy;

`cc` pole obsahující pro každou adresu v položce kopie (`Cc`) jeden objekt;

`bccaddress` obsah pole specifikujícího adresy, na které se posílají neviditelné kopie zprávy;

`bcc` pole obsahující pro každou adresu v položce neviditelné kopie (`Bcc`) jeden objekt;

`reply_toaddress`
adresa, na kterou se má zaslat odpověď na zprávu;

`reply_to` pole obsahující pro každou adresu v položce adresa pro odpověď (`Reply-To`) jeden objekt;

`senderaddress`
: adresa odesílatele;

`sender`
: pole obsahující pro každou adresu v položce adresa odesílatele (`Sender`) jeden objekt;

`return_pathaddress`
: adresy pro návrat odpovědi na zprávu;

`return_path`
: pole obsahující pro každou adresu v položce `return_path` jeden objekt;

`Recent`
: obsahuje `R`, pokud zpráva nebyla ve schránce při poslední kontrole, a `N`, pokud je zpráva nová;

`Unseen`
: obsahuje `U`, pokud zpráva ještě nebyla přečtena;

`Flagged`
: obsahuje `F`, pokud je zpráva označena;

`Answered`
: obsahuje `A`, pokud již byla zpráva zodpovězena;

`Deleted`
: obsahuje `D`, pokud je zpráva určena k výmazu;

`Msgno`
: obsahuje číslo zprávy;

`MailDate`
: datum odeslání zprávy;

`Size`
: velikost zprávy;

`udate`
: čas odeslání zprávy jako počet sekund od 1. ledna 1970;

`fetchfrom`
: obsahuje pole odesílatele zprávy o délce maximálně *«délka_From»* znaků;

`fetchsubject`
: obsahuje předmět zprávy o délce maximálně *«délka_předmětu»* znaků.

IMAP_Headers

Podpora protokolu IMAP

Získání hlaviček všech zpráv v poštovní schránce

§ `array IMAP_Headers(integer` *«spojení»*`)`

Funkce z aktuální schránky načte hlavičky všech zpráv a vrátí je jako pole. Hlavičky jedné zprávy jsou zformátovány do jednoho řetězce a uloženy jako jeden prvek pole.

Ukázky použití naleznete na straně 224.

IMAP_ListMailBox

Podpora protokolu IMAP

Zjištění všech dostupných poštovních schránek

§ `array IMAP_ListMailBox(integer` *«spojení»*`)`

Funkce vrací pole, které obsahuje názvy všech dostupných poštovních schránek.

IMAP_ListSubscribed

Podpora protokolu IMAP

Zjištění všech poštovních schránek zapsaných k odběru

§ `array IMAP_ListSubscribed(integer «spojení»)`

Funkce vrací pole, které obsahuje názvy všech poštovních schránek určených k odběru.

IMAP_Mail_Copy

Podpora protokolu IMAP

Zkopírování zpráv do poštovní schránky

§ `integer IMAP_Mail_Copy(integer «spojení», string «schránka», string «zprávy», integer «možnosti»)`

Funkce zkopíruje z aktuální schránky zprávy do *«schránky»*. *«Zprávy»*, které se budou kopírovat, můžeme určit intervalem a zkopírovat jich tak více najednou. Interval se zadává jako čísla počáteční a koncové zprávy oddělená dvojtečkou. Pokud jednotlivé intervaly oddělíme čárkami, můžeme jich použít více najednou. Kromě intervalu můžeme samozřejmě použít i samostatné číslo jedné zprávy.

Jako hodnotu nepovinného parametru *«možnosti»* můžeme použít logický součet některých z konstant `CP_UID` a `CP_MOVE`. Pokud použijeme `CP_MOVE`, zpráva se nebude kopírovat, ale přesune se. Příznak `CP_UID` zase říká, že zprávy nejsou určeny pořadovým číslem, ale svým ID-číslem.

Pokud se zprávu podaří zkopírovat, vrací funkce hodnotu `true`. V opačném případě `false`.

IMAP_Mail_Move

Podpora protokolu IMAP

Přesunutí zpráv do jiné poštovní schránky

§ `integer IMAP_Mail_Move(integer «spojení», string «schránka», string «zprávy»)`

Funkce přesune z aktuální schránky zprávy do *«schránky»*. *«Zprávy»*, které se budou přesouvat, můžeme určit intervalem a zkopírovat jich tak více najednou. Pokud se zprávu podaří zkopírovat, vrací funkce hodnotu `true`. V opačném případě `false`.

IMAP_MailBoxMsgInfo

Podpora protokolu IMAP

Zjištění informací o aktuální poštovní schránce

§ `object IMAP_MailBoxMsgInfo(integer` *«spojení»*`)`

Funkce zkontroluje aktuální stav schránky na serveru a vrací informace v podobě objektu s následujícími členskými proměnnými:

`Date`	datum poslední zprávy;
`Driver`	ovladač;
`Mailbox`	jméno poštovní schránky;
`Nmsgs`	počet zpráv ve schránce;
`Recent`	počet nových zpráv;
`Unread`	počet nepřečtených zpráv;
`Size`	velikost schránky.

V případě chyby vrací funkce `false`.

IMAP_Num_Msg

Podpora protokolu IMAP

Zjištění počtu zpráv v aktuální schránce

§ `integer IMAP_Num_Msg(integer` *«spojení»*`)`

Funkce vrací počet zpráv v aktuální poštovní schránce.

Ukázky použití naleznete na straně 223.

IMAP_Num_Recent

Podpora protokolu IMAP

Zjistí počet nových zpráv ve schránce

§ `integer IMAP_Num_Recent(integer` *«spojení»*`)`

Funkce vrací počet nových zpráv v aktuální poštovní schránce.

IMAP_Open

Podpora protokolu IMAP

Otevření spojení s IMAP serverem

§ `integer IMAP_Open(string` *«schránka»*`, string` *«uživatel»*`,`
`string` *«heslo»*`, integer` *«možnosti»*`)`

Funkce vrací číslo *«spojení»* s IMAP serverem, pokud se *«schránku»* podaří otevřít pod daným *«uživatelem»* a *«heslem»*. V případě neúspěchu vrací funkce `false`. Poslední parametr *«možnosti»* je nepovinný a může obsahovat kombinaci následujících konstant:

- `OP_READONLY` otevře schránku jen pro čtení;
- `OP_ANONYMOUS` server nebude upravovat soubor `.newsrc` při práci s news;
- `OP_HALFOPEN` pro IMAP a NNTP spojení vytvoří spojení, ale neotevře žádnou schránku;
- `CL_EXPUNGE` při uzavírání schránky automaticky smaže všechny zprávy označené ke smazání.

Formát parametru *«schránka»* může být velice rozmanitý. Nejčastěji má tvar `{`*«počítač»*`}`*«schránka»*. Pro specifikování schránky došlé pošty na serveru `mail.nekde.cz` můžeme použít zápis `{mail.nekde.cz}INBOX`. Za adresu počítače můžeme za dvojtečku připojit i číslo portu: `{mail.nekde.cz:143}INBOX`. Pokud je schránka uložena na lokálním počítači, můžeme celou část určující adresu počítače vynechat a zadat pouze jméno schránky.

Pomocí funkcí pro práci s IMAP můžeme pracovat i s jinými protokoly. Například se můžeme připojit ke starším poštovním serverům, které podporují pouze protokol POP: `{mail.nekde.cz/pop3}INBOX`. Kromě protokolů IMAP a POP můžeme použít i protokol NNTP (`nntp`), který se používá pro čtení síťových news.

Ukázky použití naleznete na stranách 222, 224.

IMAP_Ping

Podpora protokolu IMAP

Kontrola aktivity spojení

§ `integer IMAP_Ping(integer` *«spojení»*`)`

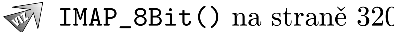
Funkce vrací `true`, pokud je *«spojení»* stále aktivní. Pokud je spojení přerušeno, vrací funkce `false`. Tuto funkci můžeme s výhodou použít pro vyvolání kontroly nové pošty nebo pro udržování spojení se servery, které mají nastaven time-out.

IMAP_QPrint Podpora protokolu IMAP

Dekódování textu zakódovaného metodou quoted-printable

§ `string IMAP_QPrint(string` *«text»*`)`

Funkce rozkóduje *«text»* zakódovaný metodou quoted-printable. Metoda quoted-printable převádí netisknutelné znaky (tj. znaky s kódem menším než 32 a větším než 126) na sekvenci ‘`=`*«kód»*’, kde *«kód»* je kód příslušného znaku zapsaný v šestnáctkové soustavě. Výjimku tvoří znak ‘`=`’, který je kvůli svému speciálnímu významu převeden na sekvenci ‘`=3D`’.



`IMAP_8Bit()` na straně 320.

IMAP_RenameMailBox Podpora protokolu IMAP

Přejmenování schránky

§ `integer IMAP_RenameMailBox(integer` *«spojení»*`, string` *«staré_jméno»*`,`
`string` *«nové_jméno»*`)`

Funkce přejmenuje schránku *«staré_jméno»* na *«nové_jméno»*. Pokud se název schránky podaří změnit, vrací funkce `true`, v opačném případě `false`.

IMAP_ReOpen Podpora protokolu IMAP

Nastavení aktuální schránky pro spojení

§ `integer IMAP_ReOpen(integer` *«spojení»*`, string` *«schránka»*`, integer`
«možnosti»`)`

Funkce pro dané *«spojení»* otevře *«schránku»* a nastaví ji jako aktuální schránku, se kterou se pracuje ve většině ostatních funkcí. Způsob otevření schránky můžeme ovlivnit pomocí *«možností»*:

- `OP_READONLY` otevře schránku jen pro čtení;
- `OP_ANONYMOUS`
  server nebude upravovat soubor `.newsrc` při práci s news;
- `OP_HALFOPEN` pro IMAP a NNTP spojení zachová spojení, ale neotevře žádnou schránku;
- `CL_EXPUNGE` při uzavření schránky se automaticky smažou všechny zprávy označené ke smazání.

Pokud se funkci podaří úspěšně provést, vrací hodnotu `true`. V případě neúspěchu dostaneme `false`.

IMAP_RFC822_Parse_AdrList

Podpora protokolu IMAP

Zjištění údajů z řetězce obsahujícího e-mailové adresy

§ `array IMAP_RFC822_Parse_AdrList(string` *«adresy»*`, string` *«počítač»*`)`

Funkce rozlouská e-mailové *«adresy»* a pro každou vrátí jeden objekt (objekty jsou uložené v poli). Každý objekt obsahuje následující členské proměnné:

`mailbox`	jméno poštovního účtu;
`host`	jméno počítače (pokud jej adresa neobsahuje, použije se hodnota předaná pomocí parametru *«počítač»*);
`personal`	osobní jméno;
`adl`	at-domain-list.

IMAP_RFC822_Write_Address

Podpora protokolu IMAP

Vytvoření korektní e-mailové adresy

§ `string IMAP_RFC822_Write_Address(string` *«jméno_účtu»*`,`
`string` *«počítač»*`, string` *«jméno»*`)`

Funkce vytvoří platnou e-mailovou adresu. Například po volání

```
$adresa = IMAP_RFC822_Write_Address("jkj", "ucw.cz", "Jirka Kosek");
```

dostaneme v proměnné `$adresa` adresu `Jirka Kosek <jkj@ucw.cz>`.

IMAP_ScanMailBox

Podpora protokolu IMAP

Nalezení schránek, které ve svém názvu obsahují daný text

§ `array IMAP_ScanMailBox(integer` *«spojení»*`, string` *«text»*`)`

Funkce vrací pole obsahující názvy všech schránek, které ve svém názvu mají *«text»*. V případě chyby je vráceno `false`.

IMAP_SetFlag_Full

Podpora protokolu IMAP

Nastavení příznaků u zpráv

§ `integer IMAP_SetFlag_Full(integer` *«spojení»*, `string` *«zprávy»*,
`string` *«příznaky»*, `integer` *«možnosti»*)

Funkce u všech *«zpráv»* nastaví zadané *«příznaky»*. Zprávy můžeme zadat pomocí intervalu, jehož syntaxe je popsána u funkce `IMAP_Mail_Copy()` na straně 329. Pokud použijeme parametr *«možnosti»* a jako jeho hodnotu použijeme konstantu `ST_UID`, neobsahuje parametr *«zprávy»* pořadová čísla zpráv, ale identifikační čísla.

Jako obsah parametru *«příznaky»* můžeme použít následující řetězce: `\Seen`, `\Deleted`, `\Draft`, `\Flagged` a `\Answered`.

IMAP_Sort

Podpora protokolu IMAP

Vrátí seznam zpráv setříděných podle určitého kritéria

§ `array IMAP_Sort(integer` *«spojení»*, `integer` *«kritéria»*,
`integer` *«sestupně»*, `integer` *«možnosti»*)

Funkce setřídí zprávy v aktuální schránce podle *«kritérií»* a vrátí pole obsahující čísla zpráv v požadovaném pořadí. Jako *«kritérium»* můžeme použít součet následujících konstant:

`SORTDATE`	setřiď podle data odeslání zprávy;
`SORTARRIVAL`	setřiď podle data doručení;
`SORTFROM`	setřiď podle první adresy v poli odesílatel;
`SORTSUBJECT`	setřiď podle předmětu zprávy;
`SORTTO`	setřiď podle adresy příjemce;
`SORTCC`	setřiď podle první adresy uvedené v poli kopie;
`SORTSIZE`	setřiď podle velikosti.

Pokud jako hodnotu parametru *«sestupně»* uvedeme `true`, setřídí se zprávy sestupně, a ne vzestupně. Nepovinný parametr *«možnosti»* může obsahovat součet následujících konstant:

`SE_UID`	vrácené pole bude obsahovat identifikační čísla zpráv místo čísel pořadových;
`SE_NOPREFETCH`	prohledané zprávy nebudou přednačítány.

IMAP_Subscribe

Podpora protokolu IMAP

Přihlášení schránky k odběru

§ `integer IMAP_Subscribe(integer` *«spojení»*`, string` *«schránka»*`)`

Funkce přihlásí *«schránku»* k odběru. Pokud se podaří schránku přihlásit, vrací funkce `true`. V opačném případě `false`.

IMAP_UID

Podpora protokolu IMAP

Vrátí identifikační číslo zprávy

§ `integer IMAP_UID(integer` *«spojení»*`, integer` *«číslo_zprávy»*`)`

Funkce vrací identifikační číslo zprávy, kterou určíme pomocí jejího pořadového *«čísla_zprávy»*.

IMAP_Undelete

Podpora protokolu IMAP

Zruší označení zprávy pro smazání

§ `integer IMAP_UnDelete(integer` *«spojení»*`, integer` *«číslo_zprávy»*`)`

Funkce zruší příznak, který zprávu označuje ke smazání. Pokud se podaří příznak změnit, vrací funkce `true`, v opačném případě `false`.

`IMAP_Delete()` na straně 323.

IMAP_Unsubscribe

Podpora protokolu IMAP

Odhlášení schránky z odběru

§ `integer IMAP_Unsubscribe(integer` *«spojení»*`, string` *«schránka»*`)`

Funkce odhlásí *«schránku»* z odběru. Pokud se podaří schránku odhlásit, vrací funkce `true`, v opačném případě `false`.

Implode

Funkce pro práci s textovými řetězci

Spojí prvky pole zadaným textem do jednoho řetězce

§ `string Implode(array` *«pole»*`, string` *«spojovací text»*`)`

Spojí jednotlivé prvky *«pole»* *«spojovacím textem»* a vytvoří tak jeden řetězec.

Výstupem následujícího skriptu

```
$adresa = ("Jan Novák", "Dlouhá 13", "Praha 1", "110 00");
echo Implode($adresa, ", ");
```

bude adresa s jednotlivými částmi přehledně oddělenými čárkou:

```
Jan Novák, Dlouhá 13, Praha 1, 110 00
```

`Explode()` na straně 289, `Join()` na straně 343 a `Split()` na straně 412.

Ukázky použití naleznete na straně 240.

IntVal

Proměnná

Celočíselná hodnota proměnné

§ `integer IntVal(mixed` *«výraz»*`, integer` *«základ»*`)`

Vrátí celočíselnou část *«výrazu»*. *«Výraz»* může být pouze skalárního typu `integer`, `double` a `string`. Nejčastěji se funkce používá na řetězce. Volitelný parametr *«základ»* pak určuje v jaké číselné soustavě je číslo zapsáno. Pokud *«základ»* nepoužijeme, použije se jeho standardní hodnota 10.

Převod čísla `21212` z trojkové soustavy:

```
echo IntVal("21212", 3);
```

`DoubleVal()` na straně 282 a `StrVal()` na straně 422.

Is_Array

Proměnná

Zjištění zda výraz je pole

§ `integer Is_Array(mixed «výraz»)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `array` (tj. jedná se o pole). Jinak vrací `false`.

`Is_Double()` na straně 337, `Is_Float()` na straně 338, `Is_Int()` na straně 338, `Is_Integer()` na straně 339, `Is_Real()` na straně 340, `Is_String()` na straně 340, `Is_Long()` na straně 339 a `Is_Object()` na straně 339.

Ukázky použití naleznete na straně 46.

Is_Dir

Funkce pro práci se soubory

Zjištění, zda dané jméno souboru je adresář

§ `integer Is_Dir(string «jméno_souboru»)`

Funkce vrací `true`, pokud *«jméno_souboru»* existuje a je to adresář. V opačném případě vrací `false`.

`Is_File()` na straně 338, `Is_Link()` na straně 339 a `Is_Executable()` na straně 338.

Is_Double

Proměnná

Zjištění zda výraz je typu `double`

§ `integer Is_Double(mixed «výraz»)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `double`. Jinak vrací `false`.

`Is_Array()` na straně 337, `Is_Float()` na straně 338, `Is_Int()` na straně 338, `Is_Integer()` na straně 339, `Is_Real()` na straně 340, `Is_String()` na straně 340, `Is_Long()` na straně 339 a `Is_Object()` na straně 339.

Ukázky použití naleznete na straně 46.

Is_Executable

Funkce pro práci se soubory

Zjistí, zda je zadaný soubor spustitelný

§ `integer Is_Executable(string` *«jméno_souboru»*`)`

Funkce vrací `true`, pokud soubor *«jméno_souboru»* existuje a je to spustitelný soubor. V opačném případě vrací `false`.

`Is_File()` na straně 338 a `Is_Link()` na straně 339.

Is_File

Funkce pro práci se soubory

Zjistí, zda zadaný soubor je normální soubor

§ `integer Is_File(string` *«jméno_souboru»*`)`

Funkce vrací `true`, pokud soubor *«jméno_souboru»* existuje a je to normální soubor. V opačném případě vrací `false`.

`Is_Dir()` na straně 337 a `Is_Link()` na straně 339.

Is_Float

Proměnná

Zjištění, zda je výraz typu `double`

§ `integer Is_Float(mixed` *«výraz»*`)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `double`. Jinak vrací `false`. Jde o synonymum k funkci `Is_Double()`.

`Is_Array()` na straně 337, `Is_Double()` na straně 337, `Is_Int()` na straně 338, `Is_Integer()` na straně 339, `Is_Real()` na straně 340, `Is_String()` na straně 340, `Is_Long()` na straně 339 a `Is_Object()` na straně 339.

Is_Int

Proměnná

Zjištění, zda je výraz typu `integer`

§ `integer Is_Int(mixed` *«výraz»*`)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `integer`. Jinak vrací `false`. Jde o synonymum k funkci `Is_Integer()`.

Is_Array() na straně 337, Is_Double() na straně 337, Is_Float() na straně 338, Is_Integer() na straně 339, Is_Real() na straně 340, Is_String() na straně 340, Is_Long() na straně 339 a Is_Object() na straně 339.

Is_Integer

Proměnná

Zjištění, zda je výraz typu integer

§ `integer Is_Integer(mixed «výraz»)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `integer`. Jinak vrací `false`.

Is_Array() na straně 337, Is_Double() na straně 337, Is_Float() na straně 338, Is_Int() na straně 338, Is_Real() na straně 340, Is_String() na straně 340, Is_Long() na straně 339 a Is_Object() na straně 339.

Ukázky použití naleznete na straně 46.

Is_Link

Funkce pro práci se soubory

Zjistí, zda je zadaný soubor symbolický odkaz

§ `integer Is_Link(string «jméno_souboru»)`

Funkce vrací `true`, pokud soubor *«jméno_souboru»* existuje a je to symbolický odkaz. V opačném případě vrací funkce `false`.

Is_Dir() na straně 337 a Is_File() na straně 338.

Is_Long

Proměnná

Zjištění, zda je výraz typu integer

§ `integer Is_Long(mixed «výraz»)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `integer`. Jinak vrací `false`. Jedná se o synonymum k funkci `Is_Integer()`.

Is_Array() na straně 337, Is_Double() na straně 337, Is_Float() na straně 338, Is_Int() na straně 338, Is_Real() na straně 340, Is_String() na straně 340, Is_Integer() na straně 339 a Is_Object() na straně 339.

Is_Object

Proměnná

Zjištění, zda je výraz typu `object`

§ `integer Is_Object(mixed` *«výraz»*`)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `object`. Jinak vrací `false`.

`Is_Array()` na straně 337, `Is_Double()` na straně 337, `Is_Float()` na straně 338, `Is_Int()` na straně 338, `Is_Real()` na straně 340, `Is_String()` na straně 340, `Is_Integer()` na straně 339 a `Is_Long()` na straně 339.

Ukázky použití naleznete na straně 46.

Is_Readable

Funkce pro práci se soubory

Zjistí, zda lze zadaný soubor číst

§ `integer Is_Readable(string` *«jméno_souboru»*`)`

Funkce vrací `true`, pokud soubor *«jméno_souboru»* existuje a lze z něj číst. V opačném případě vrací `false`. Při ověřování přístupových práv k souboru si musíme uvědomit, že PHP k souboru přistupuje pod UID WWW-serveru, kterým je nejčastěji uživatel `nobody`.

Tato funkce nebere v úvahu omezení způsobená používáním bezpečného módu.

`Is_Writeable()` na straně 341.

Is_Real

Proměnná

Zjištění, zda je výraz typu `double`

§ `integer Is_Real(mixed` *«výraz»*`)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `double`. Jinak vrací `false`. Jde o synonymum k funkci `Is_Real()`.

`Is_Array()` na straně 337, `Is_Double()` na straně 337, `Is_Int()` na straně 338, `Is_Integer()` na straně 339, `Is_Float()` na straně 338, `Is_String()` na straně 340, `Is_Long()` na straně 339 a `Is_Object()` na straně 339.

Is_String

Proměnná

Zjištění, zda je výraz typu `string`

§ `integer Is_String(mixed` *«výraz»*`)`

Vrací hodnotu `true`, pokud je *«výraz»* typu `string`. Jinak vrací `false`.

`Is_Array()` na straně 337, `Is_Double()` na straně 337, `Is_Int()` na straně 338, `Is_Integer()` na straně 339, `Is_Float()` na straně 338, `Is_Real()` na straně 340, `Is_Long()` na straně 339 a `Is_Object()` na straně 339.

Ukázky použití naleznete na straně 46.

Is_Writeable

Funkce pro práci se soubory

Zjistí, zda lze do zadaného souboru zapisovat

§ `integer Is_Writeable(string` *«jméno_souboru»*`)`

Funkce vrací `true`, pokud soubor *«jméno_souboru»* existuje a lze do něj zapisovat. V opačném případě vrací `false`. Při ověřování přístupových práv k souboru si musíme uvědomit, že PHP k souboru přistupuje pod UID WWW-serveru, kterým je nejčastěji uživatel `nobody`.

Tato funkce nebere v úvahu omezení způsobená používáním bezpečného módu.

`Is_Readable()` na straně 340.

IsSet

Proměnná

Zjištění, zda je proměnná zinicializována

§ `integer IsSet(mixed` *«proměnná»*`)`

Vrací `true`, pokud *«proměnná»* existuje a obsahuje nějakou hodnotu. V opačném případě vrací `false`.

`Empty()` na straně 283.

JDDayOfWeek

Funkce pro práci s daty různých kalendářů

Získání jména dne v týdnu

§ `string JDDayOfWeek(integer` *«juliánské_datum»*`, integer` *«mód»*`)`

Funkce vrací jméno dne v týdnu odpovídající *«juliánskému_datu»*. Parametr *«mód»* určuje výsledný formát jména dne v týdnu:

- 0 číslo dne v týdnu (0ť=ťneděle, 1ť=ťpondělí, ...);
- 1 jméno dne v týdnu (anglicky);
- 2 anglická zkratka dne v týdnu.

JDMonthName

Funkce pro práci s daty různých kalendářů

Získání jména měsíce

§ `string JDMonthName(integer` *«juliánské_datum»*`, integer` *«mód»*`)`

Funkce vrací jméno měsíce odpovídající *«juliánskému_datu»*. Parametr *«mód»* určuje druh kalendáře a použitý formát jména měsíce:

- 0 gregoriánský kalendář — zkratka jména měsíce;
- 1 gregoriánský kalendář — jméno měsíce;
- 2 juliánský kalendář — zkratka jména měsíce;
- 3 juliánský kalendář — jméno měsíce;
- 4 židovský kalendář — jméno měsíce;
- 5 francouzský kalendář — jméno měsíce.

JDToFrench

Funkce pro práci s daty různých kalendářů

Převod juliánského data na den ve francouzském republikovém kalendáři

§ `string JDToFrench(integer` *«juliánské_datum»*`)`

Funkce převede *«juliánské_datum»* na den ve francouzském republikovém kalendáři. Výsledné datum má tvar *«měsíc»*`/`*«den»*`/`*«rok»*.

`FrenchToJD()` na straně 296.

JDToGregorian

Funkce pro práci s daty různých kalendářů

Převod juliánského data na den v gregoriánském kalendáři

§ `string JDToGregorian(integer` *«juliánské_datum»*`)`

Funkce převede *«juliánské_datum»* na den v gregoriánském kalendáři. Výsledné datum má tvar *«měsíc»*/*«den»*/*«rok»*.

`GregorianToJD()` na straně 307.

JDToJewish

Funkce pro práci s daty různých kalendářů

Převod juliánského data na den v židovském kalendáři

§ `string JDToJewish(integer` *«juliánské_datum»*`)`

Funkce převede *«juliánské_datum»* na den v židovském kalendáři. Výsledné datum má tvar *«měsíc»*/*«den»*/*«rok»*.

`JewishToJD()` na straně 343.

JDToJulian

Funkce pro práci s daty různých kalendářů

Převod juliánského data na den v juliánském kalendáři

§ `string JDToJulian(integer` *«juliánské_datum»*`)`

Funkce převede *«juliánské_datum»* na den v juliánském kalendáři. Výsledné datum má tvar *«měsíc»*/*«den»*/*«rok»*.

`JulianToJD()` na straně 344.

JewishToJD

Funkce pro práci s daty různých kalendářů

Převod data židovského kalendáře na juliánské datum

§ `integer JewishToJD(integer` *«měsíc»*`, integer` *«den»*`, integer` *«rok»*`)`

Funkce převede datum židovského kalendáře na juliánské datum. Funkce akceptuje data zadaná od roku 3 761 př. n. l.

`JDToJewish()` na straně 343.

Join

Funkce pro práci s textovými řetězci

Spojí prvky pole zadaným textem do jednoho řetězce

§ `string Join(array` *«pole»*`, string` *«spojovací text»*`)`

Spojí jednotlivé prvky *«pole»* *«spojovacím textem»* a vytvoří tak jeden řetězec. Jde o synonymum k funkci `Implode()`.

`Explode()` na straně 289, `Implode()` na straně 336 a `Split()` na straně 412.

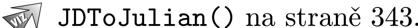
JulianToJD

Funkce pro práci s daty různých kalendářů

Převod data juliánského kalendáře na juliánské datum

§ `integer JulianToJD(integer` *«měsíc»*`, integer` *«den»*`, integer` *«rok»*`)`

Funkce převede datum juliánského kalendáře na juliánské datum. Funkce akceptuje data zadaná od roku 4 713 př. n. l. do roku 9 999 n. l. Prakticky se však juliánský kalendář používá až od roku 46 př. n. l.



`JDToJulian()` na straně 343.

Key

Funkce pro práci s poli

Zjistí index prvku pole, na který je nastaven ukazatel

§ `mixed Key(array` *«pole»*`)`

Funkce vrátí index prvku *«pole»*, na který ukazuje ukazatel.

`Current()` na straně 271, `Prev()` na straně 399, `Next()` na straně 372, `Reset()` na straně 405, `Each()` na straně 282, `End()` na straně 284 a `Array_Walk()` na straně 259.

Ukázky použití naleznete na straně 44.

KSort

Funkce pro práci s poli

Setřídí pole podle obsahu indexů a zachová indexy prvků

§ `void KSort(array «pole»)`

Funkce setřídí prvky *«pole»* podle hodnoty jejich indexů. Po setřídění jsou zachovány indexy jednotlivých prvků.

`ASort()` na straně 260, `ARSort()` na straně 259, `RSort()` na straně 406 a `Sort()` na straně 411.

LDAP_Add

Funkce pro přístup k adresářovým službám

Přidání položky do adresáře LDAP

§ `integer LDAP_Add(integer «spojení», string «dn», array «položka»)`

Funkce slouží k přidání nové položky do LDAP adresáře. Parametr *«dn»* určuje DN jméno nově přidávané položky. Novou *«položku»* pak předáváme jako asociativní pole, kde index prvku obsahuje jméno atributu a samotný prvek obsahuje hodnotu atributu. Pokud má jeden atribut obsahovat více hodnot, použijeme vícerozměrné pole, kde kromě indexu se jménem atributu použijeme druhý index udávající pořadí hodnoty atributu (počítáno od nuly).

Funkce vrací `true`, pokud se novou položku podaří přidat. V opačném případě vrací `false`.

Vložení nové položky do adresáře:

```
<?
$ds = LDAP_Connect("localhost");
    // předpokládáme, že LDAP server běží na stejném počítači jako skript

if ($ds):
    // autentifikované připojení k adresáři
    if (LDAP_Bind($ds, "cn=root, o=Grada, c=CZ", "heslo")):
        // připravíme data
        $info["cn"]="Jan Novák";
        $info["sn"]="Novák";
        $info["mail"]="jan@grada.cz";
        $info["objectclass"]="person";

        // a přidáme je do adresáře
        if (!LDAP_Add($ds, "cn=Jan Novák, o=Grada, c=CZ", $info))
```

```
            echo "Nepodařilo se přidat novou položku.";
    else:
        echo "Nepodařilo se připojit k adresáři.";
    endif;
    LDAP_Close($ds);
else:
    echo "Nepodařilo se připojit k serveru.";
endif;
?>
```

LDAP_Bind

Funkce pro přístup k adresářovým službám

Přihlášení k adresáři LDAP

§ `integer LDAP_Bind(integer` *«spojení»*`, string` *«rdn»*`, string` *«heslo»*`)`

Funkce se přihlásí k adresáři určenému pomocí *«spojení»*. Pokud použijeme parametry *«rdn»* a *«heslo»*, jsou využity při autentifikaci. Pokud je vynecháme, je provedeno anonymní přihlášení. Parametr *«rdn»* slouží pro zadání tzv. RDN (Relative Distinguished Name), které identifikuje položku v adresáři. Nejčastěji má tvar `cn=`*«identifikátor»*.

Pokud se k adresáři podaří přihlásit, vrací funkce `true`. V opačném případě `false`.

Ukázky použití naleznete na straně 220.

LDAP_Close

Funkce pro přístup k adresářovým službám

Uzavření spojení s LDAP serverem

§ `integer LDAP_Close(integer` *«spojení»*`)`

Funkce uzavře *«spojení»* se serverem. Pokud se spojení podaří uzavřít, vrací funkce `true`. V opačném případě `false`.

Ukázky použití naleznete na straně 221.

LDAP_Connect Funkce pro přístup k adresářovým službám

Připojení k LDAP serveru

§ `integer LDAP_Connect(string` *«počítač»*`, integer` *«port»*`)`

Funkce vrací identifikátor spojení s LDAP serverem určeným pomocí jeho adresy a portu. Pokud *«port»* neuvedeme, použije se standardní port 389. Pokud neuvedeme žádný parametr, vrací funkce identifikátor již existujícího spojení. V případě, že se spojení se serverem nepodaří vytvořit, vrací funkce `false`.

Ukázky použití naleznete na straně 220.

LDAP_Count_Entries Funkce pro přístup k adresářovým službám

Zjištění počtu položek výsledku hledání

§ `integer LDAP_Count_Entries(integer` *«spojení»*`, integer` *«výsledek»*`)`

Funkce vrací počet položek, které obsahuje daný *«výsledek»* hledání v adresáři. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

LDAP_Delete Funkce pro přístup k adresářovým službám

Vymazání položky z adresáře

§ `integer LDAP_Delete(integer` *«spojení»*`, string` *«dn»*`)`

Funkce smaže z adresáře položku určenou pomocí DN (Distinguished Name) uloženého v parametru *«dn»*. Pokud se položku podaří smazat, vrací funkce `true`. V opačném případě `false`.

LDAP_DN2UFN Funkce pro přístup k adresářovým službám

Funkce převede jméno DN do lidsky čitelné podoby

§ `string LDAP_DN2UFN(string` *«dn»*`)`

Funkce převede DN jméno do lidsky čitelnější podoby tím, že z něj odstraní všechny názvy typů.

LDAP_Explode_DN

Funkce pro přístup k adresářovým službám

Rozložení DN jména na jednotlivé části

§ `array LDAP_Explode_DN(string «dn», integer «atributy»)`

Funkce rozloží DN jméno předané pomocí parametru *«dn»* na jeho jednotlivé části, tzv. RDN. Výsledek je vrácen jako pole, kde každý prvek obsahuje jednu část původního DN. Pokud chceme u RDN kromě jejich hodnoty získat i jejich jméno, uvedeme jako hodnotu parametru *«atributy»* `true`. Pokud nás jména nezajímají, použijeme `false`. Pokud při provádění funkce dojde k chybě, je vrácena hodnota `false`.

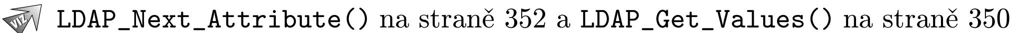
LDAP_First_Attribute

Funkce pro přístup k adresářovým službám

Zjištění jména prvního atributu položky výsledku

§ `string LDAP_First_Attribute(integer «spojení», integer «položka», integer «id»)`

Funkce zjistí jméno prvního atributu *«položky»* výsledku. Konkrétní obsah atributu můžeme zjistit pomocí funkce `LDAP_Get_Values()`. Parametr *«id»* musí být předáván odkazem a používá se i funkce `LDAP_Next_Attribute()` — v parametru je interně uložena aktuální pozice v seznamu atributů. Pokud funkce neproběhne úspěšně, je vrácena hodnota `false`.


`LDAP_Next_Attribute()` na straně 352 a `LDAP_Get_Values()` na straně 350.

LDAP_First_Entry

Funkce pro přístup k adresářovým službám

Získání identifikátoru první položky výsledku

§ `integer LDAP_First_Entry(integer «spojení», integer «výsledek»)`

Funkce vrací identifikátor první položky *«výsledku»*. Tento identifikátor můžeme použít v dalších funkcích pro získání jednotlivých atributů položky — např. ve funkci `LDAP_Get_Attributes()`. Identifikátor použijeme i ve funkci `LDAP_Next_Entry()`, která vrací identifikátor další položky výsledku. Pokud při provádění funkce dojde k chybě, je vrácena hodnota `false`.

Ukázky použití naleznete na straně 221.

LDAP_Free_Result Funkce pro přístup k adresářovým službám

Uvolnění výsledku prohledávání adresáře z paměti

§ `integer LDAP_Free_Result(integer` *«výsledek»*`)`

Funkce uvolní *«výsledek»* prohledávání adresáře z paměti. Pokud se paměť podaří uvolnit, vrací funkce `true`. V opačném případě vrací funkce `false`.

LDAP_Get_Attributes Funkce pro přístup k adresářovým službám

Zjištění všech atributů pro danou položku výsledku

§ `array LDAP_Get_Attributes(integer` *«spojení»*`, integer` *«položka»*`)`

Funkce přečte všechny atributy pro *«položku»* výsledku prohledávání adresáře. Atributy jsou vráceny jako několikarozměrné pole, jehož struktura je popsána níže. Pokud při zjišťování atributů dojde k chybě, je vrácena hodnota `false`.

Pro snadné pochopení struktury uložení výsledku budeme předpokládat, že výsledek volání funkce jsme uložili do proměnné `$x`. Potom prvek `$x["count"]` obsahuje počet vrácených atributů. Jména jednotlivých atributů jsou v poli uložena v prvcích `$x[0]`, `$x[1]`, `$x[2]` atd.

Podrobnější informace o každém atributu jsou uloženy v dalším rozměru pole. Pro každý atribut máme k dispozici počet hodnot, které obsahuje, v prvku `$x["`*«jméno_atributu»*`"]["count"]`. Jednotlivé hodnoty jsou pak uloženy v prvcích: `$x["`*«jméno_atributu»*`"][0]`, `$x["`*«jméno_atributu»*`"][1]`, `$x["`*«jméno_atributu»*`"][2]` atd.

`LDAP_First_Attribute()` na straně 348 a `LDAP_Next_Attribute()` na straně 352.

Ukázky použití naleznete na straně 221.

LDAP_Get_DN Funkce pro přístup k adresářovým službám

Zjištění DN jména položky výsledku

§ `string LDAP_Get_DN(integer` *«spojení»*`, integer` *«položka»*`)`

Funkce vrací DN jméno *«položky»* výsledku prohledávání adresáře. DN jméno jednoznačně identifikuje každou položku uloženou v adresáři.

`LDAP_Explode_DN()` na straně 347 a `LDAP_DN2UFN()` na straně 347.

LDAP_Get_Entries

Funkce pro přístup k adresářovým službám

Přečtení všech položek výsledku prohledávání adresáře

§ `array LDAP_Get_Entries(integer «spojení», integer «výsledek»)`

Funkce vrací informace o všech položkách *«výsledku»* v mnoharozměrném poli. Pokud se informace nepodaří zjistit, vrací funkce `false`.

Pro snadné pochopení struktury uložení výsledku budeme předpokládat, že výsledek volání funkce jsme uložili do proměnné `$x`. Potom prvek `$x["count"]` obsahuje počet vrácených položek. Prvky `$x[0]`, `$x[1]`, `$x[2]`, ... obsahují informace o jednotlivých položkách výsledku.

K atributům jednotlivých položek výsledku se dostaneme přes další rozměry vráceného pole. Jednak máme pro každou položku, jejíž pořadí je *«i»*, k dispozici její DN jméno v prvku `$x[«i»]["dn"]`. Počet atributů, které položka obsahuje, je uložen v prvku `$x[«i»]["count"]`, jeho jméno pak v prvku `$x[«i»][«j»]`, kde *«j»* je pořadí atributu. Položky výsledku (*«i»*) i atributy (*«j»*) jsou číslovány od nuly.

Hodnoty jednotlivých atributů jsou přístupné v třetím rozměru pole. Počet hodnot pro nějaký atribut můžeme zjistit v prvku `$x[«i»]["«jméno_atributu»"]["count"]`. Musíme si dát pozor na to, že *«jméno_atributu»* je převedeno na malá písmena. Jednotlivé hodnoty atributu pak nalezneme v prvcích `$x[«i»]["«jméno_atributu»"][«j»]`, kde *«j»* je pořadí hodnoty daného atributu opět počítané od nuly.

LDAP_Get_Values

Funkce pro přístup k adresářovým službám

Přečtení všech hodnot atributu položky výsledku

§ `array LDAP_Get_Values(integer «spojení», integer «položka», string «atribut»)`

Funkce vrací pole obsahující hodnoty *«atributu»* jedné *«položky»* výsledku prohledávání adresáře. V případě chyby vrací funkce hodnotu `false`.

Počet hodnot, které daný atribut obsahuje, zjistíme v prvku s indexem `count` (např. `$x["count"]`). Jednotlivé hodnoty jsou přístupné v prvcích s indexy 0, 1, 2, ...

LDAP_List

Funkce pro přístup k adresářovým službám

Prohledání jedné úrovně adresářového stromu

§ `integer LDAP_List(integer «spojení», string «dn», string «filtr», array «atributy»)`

Funkce prohledá jednu úroveň LDAP adresáře určenou pomocí jejího DN jména předaného v parametru *«dn»*. Pokud něco hledáme ve veřejně přístupných LDAP serverech s e-mailovými adresami, můžeme parametr *«dn»* klidně vynechat — údaje v těchto serverech stejně nejsou nijak hierarchicky uspořádány. Parametr *«filtr»* obsahuje kritéria prohledávání. Možnosti filtrů jsou poměrně bohaté a jsou popsány ve specifikaci LDAP. Nám bude stačit, že hvězdička '`*`' nahrazuje libovolnou skupinu znaků.

Poslední parametr *«atributy»* je nepovinný. Předává se v něm pole řetězců, které obsahují jména atributů, jež mají být přístupné funkcím pro čtení atributů jednotlivých položek výsledku.

Funkce vrací identifikátor výsledku, který můžeme použít v dalších funkcích. Pokud dojde k chybě, vrací funkce `false`.

Funkce pracuje podobně jako `LDAP_Search()`, prohledává však pouze jednu úroveň adresářového stromu.

Nalezení všech, jejichž jména začínají na `Nov`:

```
$vysledek = LDAP_List($spojeni, "", "sn=Nov*");
```

LDAP_Modify

Funkce pro přístup k adresářovým službám

Změna položky v adresáři LDAP

§ `integer LDAP_Modify(integer «spojení», string «dn», array «položka»)`

Funkce slouží k modifikaci položky v LDAP adresáři. Parametr *«dn»* určuje DN jméno modifikované položky. Nový obsah *«položky»* pak předáváme jako asociativní pole, kde index prvku obsahuje jméno atributu a samotný prvek obsahuje hodnotu atributu. Pokud má jeden atribut obsahovat více hodnot, použijeme vícerozměrné pole, kde kromě indexu se jménem atributu použijeme druhý index udávající pořadí hodnoty atributu (počítáno od nuly).

Funkce vrací `true`, pokud se položku podaří změnit. V opačném případě vrací `false`.

LDAP_Next_Attribute Funkce pro přístup k adresářovým službám

Zjištění jména dalšího atributu položky výsledku

§ `string LDAP_Next_Attribute(integer` *«spojení»*`, integer` *«položka»*`,`
`integer` *«id»*`)`

Funkce zjistí jméno dalšího atributu *«položky»* výsledku. Konkrétní obsah atributu můžeme zjistit pomocí funkce `LDAP_Get_Values()`. Parametr *«id»* musí být předáván odkazem — v parametru je interně uložena aktuální pozice v seznamu atributů. Pokud funkce neproběhne úspěšně, je vrácena hodnota `false`.

`LDAP_First_Attribute()` na straně 348 a `LDAP_Get_Values()` na straně 350.

LDAP_Next_Entry Funkce pro přístup k adresářovým službám

Přečtení další položky výsledku

§ `integer LDAP_Next_Entry(integer` *«spojení»*`, integer` *«položka»*`)`

Funkce vrací identifikátor další položky výsledku prohledávání adresáře. Jako parametr *«položka»* musíme uvádět identifikátor položky získaný posledním voláním funkce `LDAP_Next_Entry()` nebo `LDAP_First_Entry()`. Pokud již výsledek neobsahuje žádnou další položku, vrací funkce `false`.

Tuto funkci můžeme s výhodou použít pro zpracování celého výsledku prohledávání adresáře. Nejprve pomocí `LDAP_First_Entry()` získáme první položku výsledku a pak postupně voláme funkci `LDAP_Next_Entry()`, dokud nám nevrátí hodnotu `false`.

`LDAP_Get_Entries()` na straně 349 a `LDAP_First_Entry()` na straně 348.

Ukázky použití naleznete na straně 221.

LDAP_Read Funkce pro přístup k adresářovým službám

Nalezení položky v adresáři

§ `integer LDAP_Read(integer` *«spojení»*`, string` *«dn»*`, string` *«filtr»*`,`
`array` *«atributy»*`)`

Funkce nalezne položku LDAP adresáře určenou pomocí jejího DN jména předaného v parametru *«dn»*. Parametr *«filtr»* obsahuje kritéria prohledávání. Možnosti filtrů jsou poměrně bohaté a jsou popsány ve specifikaci LDAP. Nám bude stačit, že hvězdička '`*`' nahrazuje libovolnou skupinu znaků.

Poslední parametr *«atributy»* je nepovinný. Předává se v něm pole řetězců, které obsahují jména atributů, jež mají být přístupné funkcím pro čtení atributů jednotlivých položek výsledku.

Funkce vrací identifikátor výsledku, který můžeme použít v dalších funkcích. Pokud dojde k chybě, vrací funkce `false`.

Funkce pracuje podobně jako `LDAP_Search()`, prohledává však pouze jednu položku.

LDAP_Search

Funkce pro přístup k adresářovým službám

Prohledání adresářového stromu

§ `integer LDAP_Search(integer` *«spojení»*`, string` *«dn»*`, string` *«filtr»*`,`
`array` *«atributy»*`)`

Funkce prohledá LDAP adresář od úrovně určené pomocí DN jména předaného v parametru *«dn»*. Pokud něco hledáme ve veřejně přístupných LDAP serverech s e-mailovými adresami, můžeme parametr *«dn»* klidně vynechat — údaje v těchto serverech stejně nejsou nijak hierarchicky uspořádány. Parametr *«filtr»* obsahuje kritéria prohledávání. Možnosti filtrů jsou poměrně bohaté a jsou popsány ve specifikaci LDAP. Nám bude stačit, že hvězdička '`*`' nahrazuje libovolnou skupinu znaků.

Poslední parametr *«atributy»* je nepovinný. Předává se v něm pole řetězců, které obsahují jména atributů, jež mají být přístupné funkcím pro čtení atributů jednotlivých položek výsledku.

Funkce vrací identifikátor výsledku, který můžeme použít v dalších funkcích. Pokud dojde k chybě, vrací funkce `false`.

Nalezení všech, jejichž jména začínají na `Nov`:

```
$vysledek = LDAP_Search($spojeni, "", "sn=Nov*");
```

Ukázky použití naleznete na straně 221.

LDAP_UnBind

Funkce pro přístup k adresářovým službám

Odhlášení se od LDAP adresáře

§ `integer LDAP_UnBind(integer` *«spojení»*`)`

Funkce se odhlásí od LDAP adresáře, ke kterému jsme připojeni pomocí *«spojení»*. Pokud se odhlášení úspěšně podaří, vrací funkce `true`, v opačném případě `false`.

Leak

Ostatní funkce

Nenávratná alokace paměti

§ `void Leak(integer` *«velikost»*`)`

Funkce nenávratně alokuje část paměti. Využije se při ladění správce paměti.

Link

Funkce pro práci se soubory

Vytvoření pevného odkazu na soubor

§ `integer Link(string` *«cílový soubor»*`, string` *«odkaz»*`)`

Funkce vytvoří pevný *«odkaz»* na *«cílový soubor»*. Pokud se odkaz povedlo vytvořit, vrací funkce `true`, v opačném případě `false`.

`SymLink()` na straně 423, `ReadLink()` na straně 404 a `LinkInfo()` na straně 354.

LinkInfo

Funkce pro práci se soubory

Zjištění informací o odkazu

§ `integer LinkInfo(string` *«cesta»*`)`

Funkce vrací `true`, pokud odkaz *«cesta»* existuje. V opačném případě vrací `false`.

`Link()` na straně 354, `SymLink()` na straně 423 a `ReadLink()` na straně 404.

List

Funkce pro práci s poli

Přiřadí do proměnných prvky pole

§ `List(...)`

Příkaz `List()` slouží k nastavení proměnných podle hodnot obsažených v poli. Podrobný výklad nalezneme na straně 45.

`Array()` na straně 259.

Log

Matematická funkce

Přirozený logaritmus

§ `double Log(double` *«výraz»*`)`

Vrací přirozený logaritmus *«výrazu»*.

`Exp()` na straně 288.

Log10

Matematická funkce

Desítkový logaritmus

§ `double Log10(double` *«výraz»*`)`

Vrátí hodnotu logaritmu o základu 10 z *«výrazu»*.

`Log()` na straně 355, `Exp()` na straně 288 a `Pow()` na straně 399.

LStat

Funkce pro práci se soubory

Zjištění informací o symbolickém odkazu

§ `array LStat(string` *«soubor»*`)`

Funkce zjistí informace o symbolickém odkazu. Informace jsou vráceny v poli, kde jsou postupně uloženy následující údaje: zařízení, číslo i-node, počet odkazů, UID vlastníka, GID vlastníka, typ zařízení, velikost v bajtech, čas posledního přístupu, čas poslední změny, čas vytvoření, velikost bloku a počet alokovaných bloků. Ve Windows většina těchto údajů obsahuje hodnotu -1, protože funkce vrací některé parametry, které úzce souvisí se souborovými systémy používanými v Unixu.

`FileATime()` na straně 291, `FileCTime()` na straně 291, `FileMTime()` na straně 292, `FileOwner()` na straně 292, `FileGroup()` na straně 292, `FilePerms()` na straně 293, `FileSize()` na straně 293 a `FileType()` na straně 293.

LTrim

Funkce pro práci s textovými řetězci

Odstraní mezery ze začátku řetězce

§ `string LTrim(string` *«řetězec»*`)`

Funkce odstraní ze začátku řetězce všechny netisknutelné znaky — mezery, tabulátory a konce řádků.

`Trim()` na straně 425, `RTrim()` na straně 407 a `Chop()` na straně 266.

M_PI

Konstanta

Konstanta obsahuje přibližnou hodnotu čísla π.

Mail

Funkce pro práci s elektronickou poštou

Odeslání e-mailu

§ `integer Mail(string` *«komu»*`, string` *«předmět»*`, string` *«zpráva»*`,`
`string` *«hlavičky»*`)`

Funkce odešle *«zprávu»* pomocí e-mailu na adresu určenou parametrem *«komu»*. Pokud chceme dopis poslat na více adres, můžeme v parametru *«komu»* uvést více adres oddělených mezerou. *«předmět»* slouží k zadání stručného popisu *«zprávy»*.

Poslední parametr *«hlavičky»* je nepovinný a můžeme pomocí něj nastavit přídavné hlavičky, které se stanou součástí hlavičky dopisu.

Funkce vrací `true`, pokud se dopis podařilo odeslat, v opačném případě vrací `false`.

Následující ukázka odešle jednoduchý dopis. Zároveň nastaví hlavičky, aby měl dopis správného odesílatele a aby byl identifikován program použitý pro odesílání elektronické pošty:

```
if (Mail("nekdo@mail.cz", "Pokusny dopis",
    "Ahoj\nPosilam veledulezitou informaci!\n\nJirka",
    "From: php-engine@server.cz\nX-Mailer: PHP3"))
```

```
echo "Dopis byl úspěšně odeslán.";
else
echo "Dopis se nepodařilo odeslat.";
```

Ukázky použití naleznete na stranách 214, 215, 227, 228, 468.

Max Matematická funkce

Nalezení maxima z daných hodnot

§ `mixed Max(mixed` *«výraz1»*`, mixed` *«výraz2»*`, ..., mixed` *«výrazN»*`)`

Vrací maximum ze zadaných hodnot. Pokud je *«výraz1»* pole, vrací funkce maximální hodnotu uloženou v tomto poli. Pokud je *«výraz1»* skalárního typu, musí být funkce volána alespoň se dvěma parametry. Funkce vrací nejvyšší hodnotu ze všech parametrů *«výraz1»* až *«výrazN»*. Jestliže je alespoň jeden z parametrů `double` a ostatní jsou `integer`, je výsledek funkce `double`. Počet parametrů funkce není omezen.

`Min()` na straně 358.

MD5 Funkce pro práci s textovými řetězci

Spočítá hodnotu hashovací funkce MD5 pro zadaný text

§ `string MD5(string` *«řetězec»*`)`

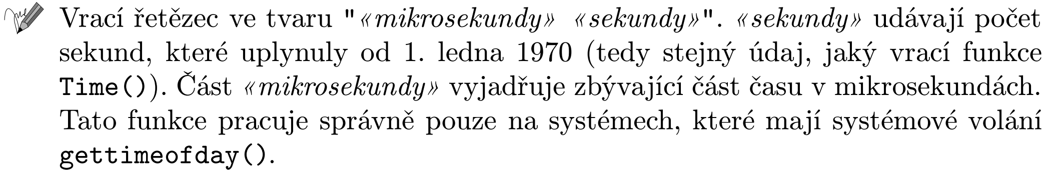
Algoritmus MD5 vygeneruje pro libovolný *«řetězec»* 128bitové číslo, které je s velmi vysokou pravděpodobností pro každý *«řetězec»* jedinečné. Algoritmus se používá zejména pro tvorbu digitálních podpisů apod. Podrobný popis algoritmu MD5 nalezneme v RFC1321.

Ukázky použití naleznete na stranách 208, 450.

MicroTime Funkce pro práci s datem a časem

Zjištění aktuálního časového údaje s přesností na mikrosekundy

§ `string MicroTime(void)`



Vrací řetězec ve tvaru "*«mikrosekundy» «sekundy»*". *«sekundy»* udávají počet sekund, které uplynuly od 1. ledna 1970 (tedy stejný údaj, jaký vrací funkce `Time()`). Část *«mikrosekundy»* vyjadřuje zbývající část času v mikrosekundách. Tato funkce pracuje správně pouze na systémech, které mají systémové volání `gettimeofday()`.

`Time()` na straně 424.

Ukázky použití naleznete na stranách 204, 413.

Min — Matematická funkce

Nalezení minima z daných hodnot

§ `mixed Min(mixed` *«výraz1»*`, mixed` *«výraz2»*`, ..., mixed` *«výrazN»*`)`

Vrací minimum ze zadaných hodnot. Pokud je *«výraz1»* pole, vrací funkce nejnižší hodnotu uloženou v tomto poli. Pokud je *«výraz1»* skalárního typu, musí být funkce volána alespoň se dvěma parametry. Funkce vrací nejmenší hodnotu ze všech parametrů *«výraz1»* až *«výrazN»*. Jestliže je alespoň jeden z parametrů `double` a ostatní jsou `integer`, je výsledek funkce `double`. Počet parametrů funkce není omezen.

`Max()` na straně 357.

MkDir — Funkce pro práci se soubory

Vytvoření adresáře

§ `integer MkDir(string` *«cesta»*`, integer` *«práva»*`)`

Vytvoří adresář *«cesta»* s danými přístupovými *«právy»*. Nesmíme zapomenout na to, že v Unixu se práva zadávají jako osmičkové číslo, funkce `MkDir()` však očekává normální číslo. V PHP lze naštěstí zadávat i osmičkové konstanty tak, že jim předřadíme znak ‘`0`’.

Funkce vrací `true`, pokud se podařilo adresář úspěšně vytvořit. V opačném případě vrací funkce `false`.

`RmDir()` na straně 406.

MkTime

Funkce pro práci s datem a časem

Získání časového údaje

§ `integer MkTime(integer` *«hodina»*`, integer` *«minuta»*`, integer` *«sec»*`,`
`integer` *«měsíc»*`, integer` *«den»*`, integer` *«rok»*`)`

Získá časový údaj na základě parametrů. Časový údaj je vyjádřen jako počet sekund od 1. ledna 1970 do zadaného časového okamžiku. Parametry můžeme postupně od konce vynechávat — dosadí se podle aktuálního systémového času.

Funkce si poradí i s nekorektně zadanými údaji — sama je správně přepočte. Například následující tři volání vrátí stejné datum — 1.ť1.ť1998:

```
echo Date("d.m.Y", MkTime(0,0,0,1,1,1998));
echo Date("d.m.Y", MkTime(0,0,0,12,32,1997));
echo Date("d.m.Y", MkTime(0,0,0,13,1,1997));
```

`Date()` na straně 272, `Time()` na straně 424 a `GMMkTime()` na straně 306.

MySQL_Affected_Rows

Funkce pro práci s databází MySQL

Počet záznamů ovlivněných posledním příkazem

§ `integer MySQL_Affected_Rows(integer` *«spojení»*`)`

Funkce vrací počet záznamů ovlivněných posledním příkazem `INSERT`, `UPDATE` a `DELETE` provedeným na daném *«spojení»* s databází. Pokud parametr *«spojení»* vynecháme, automaticky se použije poslední vytvořené spojení.

Pokud chceme zjistit počet záznamů vrácených příkazem `SELECT`, měli bychom použít funkci `MySQL_Num_Rows()`.

`MySQL_Num_Rows()` na straně 369.

MySQL_Close

Funkce pro práci s databází MySQL

Uzavření spojení s databází MySQL

§ `integer MySQL_Close(integer` *«spojení»*`)`

Funkce uzavře *«spojení»* s databází. Pokud se spojení podaří uzavřít, vrací funkce `true`. V opačném případě vrací funkce `false`. Pokud parametr *«spojení»* vynecháme, je uzavřeno poslední vytvořené spojení.

Funkci nemusíme používat, protože na konci každého skriptu jsou všechna spojení automaticky uzavřena. Funkce neumí zavřít persistentní spojení vytvořená pomocí funkce `MySQL_PConnect()`.

`MySQL_Connect()` na straně 360 a `MySQL_PConnect()` na straně 370.

Ukázky použití naleznete na straně 148.

MySQL_Connect
Funkce pro práci s databází MySQL

Vytvoření spojení s databázovým serverem

§ `integer MySQL_Connect(string` *«počítač»*`, string` *«uživatel»*`,`
`string` *«heslo»*`)`

Funkce vrací číslo spojení s databází MySQL, které je využíváno v ostatních funkcích pro práci s databází. Jako *«počítač»* uvádíme adresu počítače, na kterém běží MySQL server. Pokud server běží na nestandardním portu, můžeme jej přidat do identifikátoru počítače za dvojtečku (`db-server.firma.cz:4478`). V případě, že je přístup ke zdroji vázán na jméno a heslo, musíme je zadat. Pokud se spojení nepodaří vytvořit, vrací funkce `false`.

Libovolný z parametrů můžeme vynechat. V tom případě se použijí standardní hodnoty: počítač `localhost`, uživatel bude odpovídat uživateli, pod kterým běží právě spuštěný skript, a heslo bude prázdné.

Pokud funkci `MySQL_Connect()` zavoláme podruhé se stejnými parametry, není vytvořeno nové spojení a je vráceno již existující číslo spojení.

Spojení se serverem je automaticky ukončeno po ukončení běhu skriptu. Předčasně můžeme spojení ukončit pomocí funkce `MySQL_Close()`.

`MySQL_Close()` na straně 359 a `MySQL_PConnect()` na straně 370.

MySQL_Create_DB
Funkce pro práci s databází MySQL

Vytvoření nové databáze

§ `integer MySQL_Create_DB(string` *«jméno_databáze»*`, integer` *«spojení»*`)`

Funkce vytvoří databázi, jejíž jméno je určeno prvním parametrem. Pokud vynecháme parametr *«spojení»*, použije se poslední vytvořené spojení s MySQL serverem. Pokud se databázi podařilo vytvořit, vrací funkce `true`. V opačném případě `false`.

Abychom mohli databázi vůbec vytvořit, musíme být k serveru připojeni jako uživatel, který má příslušná práva pro vytváření databází.

`MySQL_Drop_DB()` na straně 362.

MySQL_Data_Seek

Funkce pro práci s databází MySQL

Přesun ukazatele na aktuální záznam

§ `integer MySQL_Data_Seek(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`)`

Funkce přesune interní ukazatel výsledku na záznam určený *«číslem_záznamu»*. Následné volání funkce `MySQL_Fetch_Row()` vrátí takto nastavený záznam. Záznamy jsou číslovány od nuly. Pokud funkce úspěšně proběhne, vrátí `true`. V opačném případě `false`.

`MySQL_Fetch_Row()` na straně 366.

MySQL_DBName

Funkce pro práci s databází MySQL

Přečtení jména databáze

§ `string MySQL_DBName(integer` *«výsledek»*`, integer` *«pořadí»*`)`

Funkce vrací jméno databáze. *«Výsledek»* obsahuje seznam všech databází získaný pomocí funkce `MySQL_List_DBs()`. *«Pořadí»* určuje pořadí databáze ve výsledku (začíná se od 0).

Vypsání všech databází na daném serveru:

```
MySQL_Connect(«počítač», «uživatel», «heslo»);
$vysledek = MySQL_List_DBs();
for ($i=0; $i < MySQL_Num_Rows($vysledek); $i++)
    echo MySQL_DBName($vysledek, $i)."<BR>\n";
```

`MySQL_List_DBs()` na straně 368.

MySQL_DB_Query

Funkce pro práci s databází MySQL

Vykonání SQL-příkazu

§ `integer MySQL_DB_Query(string` *«databáze»*`, string` *«SQL-příkaz»*`,`
`integer` *«spojení»*`)`

Funkce provede na daném *«spojení»* *«SQL-příkaz»* nad danou *«databází»* a vrátí identifikátor výsledku. Pokud při provádění příkazu došlo k chybě, vrací funkce `false`. Pokud parametr *«spojení»* neuvedeme, použije se poslední vytvořené spojení. Pokud žádné spojení neexistuje, pokusí se funkce vytvořit spojení s pomocí standardních parametrů jako u funkce `MySQL_Connect()`.

Ve starších verzích PHP se místo funkce `MySQL_DB_Query()` *používala funkce* `MySQL()`*. Ta je stále podporována pro zachování kompatibility, v nových skriptech bychom však měli přejít na* `MySQL_DB_Query()`*.*

`MySQL_Connect()` na straně 360.

MySQL_Drop_DB

Funkce pro práci s databází MySQL

Smazání databáze

§ `integer MySQL_Drop_DB(string` *«jméno_databáze»*`, integer` *«spojení»*`)`

Funkce se pokusí smazat databázi daného jména. Pro přístup k databázi je použito *«spojení»*. Pokud parametr spojení vynecháme, použije se poslední vytvořené spojení. Pokud se databázi podaří smazat, vrací funkce `true`. V opačném případě vrací `false`. Aby mohla být databáze smazána, musíme pomocí `MySQL_Connect()` vytvořit spojení pro uživatele, který má dostatečná přístupová práva.

Smazáním databáze nenávratně přijdeme o všechna data v ní uložená. Funkci `MySQL_Drop_DB()` proto musíme používat s rozvahou a pokud ji používáme ve veřejně přístupném skriptu, musíme si dát o to větší pozor, aby uživatel nemohl příkaz neoprávněně spustit podstrčením nějakých nestandardních a neočekávaných parametrů.

`MySQL_Create_DB()` na straně 360 a `MySQL_Connect()` na straně 360.

MySQL_ErrNo

Funkce pro práci s databází MySQL

Chybový kód posledního volání MySQL

§ `integer MySQL_ErrNo(void)`

Funkce vrací číslo chyby vyvolané poslední funkcí pro práci s MySQL. Text chybové hlášky je dostupný pomocí funkce `MySQL_Error()`.

`MySQL_Error()` na straně 363.

MySQL_Error

Funkce pro práci s databází MySQL

Text chybového hlášení posledního volání MySQL

§ `string MySQL_Error(void)`

Funkce vrací text chybového hlášení vyvolaného poslední funkcí pro práci s MySQL.

`MySQL_ErrNo()` na straně 363.

MySQL_Fetch_Array

Funkce pro práci s databází MySQL

Načte záznam výsledku do asociativního pole

§ `array MySQL_Fetch_Array(integer` *«výsledek»*`)`

Funkce načte jeden záznam *«výsledku»* do asociativního pole. Obsah každé položky je uložen do prvku, jehož index odpovídá názvu položky. To nám velmi usnadní další práci se záznamem. Pokud při čtení záznamu dojde k chybě, vrací funkce `false`.

☺ Funkce `MySQL_Fetch_Array()` není o mnoho pomalejší než `MySQL_Fetch_Row()` a není tedy důvod proč ji nepoužívat, když je s ní mnohem snazší práce.

Vypsání obsahu některých položek tabulky pomocí `MySQL_Fetch_Array()`:

```
$vysledek = MySQL_DB_Query("test", "select * from adresar");
while ($zaznam = MySQL_Fetch_Array($vysledek)):
    echo $zaznam["jmeno"]." ";
    echo $zaznam["email"]."<BR>\n";
endwhile;
```

`MySQL_Fetch_Row()` na straně 366.

Ukázky použití naleznete na straně 148.

MySQL_Fetch_Field

Funkce pro práci s databází MySQL

Získání informací o položce výsledku

§ `object MySQL_Fetch_Field(integer` *«výsledek»*`, integer` *«položka»*`)`

Výsledkem volání funkce je objekt, který obsahuje informace o dané *«položce»* *«výsledku»*. Položky jsou číslovány od nuly. Pokud *«položku»* vynecháme, je vrácena další dosud nepřečtená položka. V tomto případě vrací funkce `false`, pokud už není k dispozici další nepřečtená položka. Získaný objekt obsahuje následující členské proměnné:

`name`	jméno položky;
`table`	jméno tabulky;
`max_length`	maximální délka položky;
`not_null`	obsahuje 1, pokud položka nemůže obsahovat hodnotu NULL;
`primary_key`	obsahuje 1, pokud je položka součástí primárního klíče;
`unique_key`	obsahuje 1, pokud je položka součástí jedinečného indexu;
`multiple_key`	obsahuje 1, pokud je položka součástí indexu;
`numeric`	obsahuje 1, pokud je položka číselná;
`blob`	obsahuje 1, pokud je položka BLOB (Binary Large OBject);
`type`	typ položky;
`unsigned`	obsahuje 1, pokud je typ číslo bez znaménka;
`zerofill`	obsahuje 1, pokud je položka doplněna nulami na svoji délku.

`MySQL_Field_Seek()` na straně 366.

MySQL_Fetch_Lengths
Funkce pro práci s databází MySQL

Zjištění délek položek aktuálního záznamu výsledku

§ `array MySQL_Fetch_Lengths(integer` *«výsledek»*`)`

Funkce vrací pole, které pro každou položku posledně získaného záznamu *«výsledku»* obsahuje její délku ve znacích. Položky jsou v tomto případě číslovány od nuly a pod stejným indexem i uloženy do pole. Pokud při volání funkce dojde k chybě, vrací `false`.

> Před použitím funkce musí být z výsledku načtena nějaká data funkcí `MySQL_Fetch_Row()` nebo `MySQL_Fetch_Array()`.

`MySQL_Fetch_Row()` na straně 366 a `MySQL_Fetch_Array()` na straně 363.

MySQL_Fetch_Object
Funkce pro práci s databází MySQL

Načte záznam výsledku do objektu

§ `object MySQL_Fetch_Object(integer` *«výsledek»*`)`

Funkce načte jeden záznam *«výsledku»* do objektu. Obsah každé položky je uložen jako členská proměnná, jejíž název odpovídá názvu položky. Pokud při čtení záznamu dojde k chybě, vrací funkce `false`. Funkce není o mnoho pomalejší než `MySQL_Fetch_Row()`.

Vypsání obsahu některých položek tabulky pomocí `MySQL_Fetch_Object()`:

```
$vysledek = MySQL_DB_Query("test", "select * from adresar");
while ($zaznam = MySQL_Fetch_Object($vysledek)):
    echo $zaznam->jmeno." ";
    echo $zaznam->email."<BR>\n";
endwhile;
```

`MySQL_Fetch_Array()` na straně 363 a `MySQL_Fetch_Row()` na straně 366.

MySQL_Fetch_Row

Funkce pro práci s databází MySQL

Načte záznam výsledku do pole

§ `array MySQL_Fetch_Row(integer` *«výsledek»*`)`

Funkce načte jeden záznam *«výsledku»* do pole. Obsah každé položky je uložen do jednoho prvku pole. Pokud při čtení záznamu dojde k chybě (např. již nejsou žádné další záznamy k přečtení), vrací funkce `false`.

Vypsání obsahu některých položek tabulky pomocí `MySQL_Fetch_Row()`:

```
$vysledek = MySQL_DB_Query("test", "select * from adresar");
while ($zaznam = MySQL_Fetch_Row($vysledek)):
    echo $zaznam[0]." ";
    echo $zaznam[1]."<BR>\n";
endwhile;
```

`MySQL_Fetch_Array()` na straně 363 a `MySQL_Fetch_Object()` na straně 365.

MySQL_Field_Name

Funkce pro práci s databází MySQL

Zjištění názvu položky

§ `string MySQL_Field_Name(integer` *«výsledek»*`, integer` *«položka»*`)`

Funkce vrací jméno položky *«výsledku»*. Položka je určena číslem; položky jsou číslovány od nuly. Pokud při běhu funkce dojde k chybě, vrací funkce `false`.

`MySQL_Field_Name()` na straně 366.

MySQL_Field_Seek

Funkce pro práci s databází MySQL

Nastavení aktuálního indexu položky

§ `integer MySQL_Field_Seek(integer` *«výsledek»*`, integer` *«položka»*`)`

Funkce nastaví jako aktuální *«položku»*. Položky jsou přitom číslovány od nuly. Volání funkce `MySQL_Fetch_Field()` bez druhého parametru pak přečte položku určenou právě pomocí funkce `MySQL_Field_Seek()`. Pokud se podaří aktuální položku změnit, vrací funkce `true`. V opačném případě `false`.

`MySQL_Fetch_Field()` na straně 364.

MySQL_Field_Table

Funkce pro práci s databází MySQL

Zjištění tabulky, ze které pochází položka

§ `string MySQL_Field_Table(integer` *«výsledek»*, `integer` *«položka»*)

Funkce vrací jméno tabulky, ze které pochází *«položka»* daného *«výsledku»*. Položky jsou číslovány od nuly. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

MySQL_Field_Type

Funkce pro práci s databází MySQL

Zjištění typu položky

§ `string MySQL_Field_Type(integer` *«výsledek»*, `integer` *«položka»*)

Funkce vrací typ *«položky»* daného *«výsledku»*. Položky jsou číslovány od nuly. Pokud při provádění funkce dojde k chybě, vrací funkce `false`. Vrácený typ může být jedním z `int`, `real`, `string` a `blob`.

MySQL_Field_Flags

Funkce pro práci s databází MySQL

Zjištění doplňkových informací o položce

§ `string MySQL_Field_Flags(integer` *«výsledek»*, `integer` *«položka»*)

Funkce vrací doplňkové informace o *«položce»* daného *«výsledku»*. Položky jsou číslovány od nuly. Pokud při provádění funkce dojde k chybě, vrací funkce `false`. Vrácený řetězec může obsahovat texty jako ‘`not null`’ a ‘`primary key`’.

MySQL_Field_Len

Funkce pro práci s databází MySQL

Zjištění délky položky

§ `string MySQL_Field_Len(integer` *«výsledek»*, `integer` *«položka»*)

Funkce vrací délku *«položky»* daného *«výsledku»*. Položky jsou číslovány od nuly. Pokud při provádění funkce dojde k chybě, vrací funkce `false`.

MySQL_Free_Result

Funkce pro práci s databází MySQL

Uvolnění výsledku z paměti

§ `integer MySQL_Free_Result(integer` *«výsledek»*`)`

Funkce uvolní daný *«výsledek»* z paměti. Funkce vrací hodnotu `true`, pokud *«výsledek»* existuje. Pokud *«výsledek»* neexistuje, není co uvolňovat a funkce vrací `false`. Funkci ve většině skriptů není třeba volat, protože se paměť automaticky uvolní po skončení běhu skriptu. Volání má smysl pouze v dlouhých skriptech, kdy chceme minimalizovat paměťové nároky.

MySQL_Insert_Id

Funkce pro práci s databází MySQL

Zjištění hodnoty ID posledního příkazu `INSERT`

§ `integer MySQL_Insert_Id(void)`

Funkce vrací hodnotu poslední položky ID automaticky generované příkazem `INSERT`. Automaticky je ID generováno pro ty položky, které ve své deklaraci obsahují direktivu `AUTO_INCREMENT`.

MySQL_List_Fields

Funkce pro práci s databází MySQL

Získání výsledku s obsahem položek zadané tabulky

§ `integer MySQL_List_Fields(string` *«databáze»*`, string` *«tabulka»*`,`
`integer` *«spojení»*`)`

Funkce načte informace o položkách *«tabulky»* v *«databázi»*. Výsledkem je identifikátor výsledku, který můžeme použít ve funkcích `MySQL_Field_Flags()`, `MySQL_Field_Len()`, `MySQL_Field_Name()` a `MySQL_Field_Type()`. Pokud při běhu funkce dojde k chybě, vrací `false`. Pro komunikaci s databází je využito *«spojení»*. Pokud spojení vynecháme, použije se poslední vytvořené spojení.

`MySQL_Field_Flags()` na straně 367, `MySQL_Field_Len()` na straně 367, `MySQL_Field_Name()` na straně 366 a `MySQL_Field_Type()` na straně 367.

MySQL_List_DBs

Funkce pro práci s databází MySQL

Zjištění všech databází dostupných na serveru

§ `integer MySQL_List_DBs(integer` *«spojení»*`)`

Funkce zjistí všechny databáze dostupné na serveru, ke kterému je vytvořeno *«spojení»*. Pokud parametr *«spojení»* nepoužijeme, vezme se poslední vytvořené spojení. Funkce vrací identifikátor výsledku, který můžeme použít ve funkci `MySQL_DBName()`. Pokud dojde k chybě, vrací funkce `false`.

`MySQL_DBName()` na straně 361.

MySQL_List_Tables

Funkce pro práci s databází MySQL

Zjištění všech tabulek uložených v databázi

§ `integer MySQL_List_Tables(string` *«databáze»*`, integer` *«spojení»*`)`

Funkce zjistí jména všech tabulek uložených v *«databázi»* dostupné pomocí *«spojení»*. Pokud parametr *«spojení»* nepoužijeme, vezme se poslední vytvořené spojení. Funkce vrací identifikátor výsledku, který můžeme použít ve funkci `MySQL_TableName()`. Pokud dojde k chybě, vrací funkce `false`.

`MySQL_TableName()` na straně 372.

MySQL_Num_Fields

Funkce pro práci s databází MySQL

Zjistí počet položek výsledku

§ `integer MySQL_Num_Fields(integer` *«výsledek»*`)`

Funkce vrátí počet položek obsažených ve *«výsledku»*. Při chybě vrací funkce `false`.

`MySQL_Fetch_Field()` na straně 364 a `MySQL_Num_Rows()` na straně 369.

MySQL_Num_Rows

Funkce pro práci s databází MySQL

Zjistí počet záznamů výsledku

§ `integer MySQL_Num_Rows(integer` *«výsledek»*`)`

Funkce vrátí počet záznamů, které obsahuje *«výsledek»*. Při chybě vrací funkce `false`.

`MySQL_Fetch_Row()` na straně 366 a `MySQL_Num_Fields()` na straně 369.

Ukázky použití naleznete na straně 148.

MySQL_PConnect

Funkce pro práci s databází MySQL

Vytvoří persistentní spojení s databázovým serverem

§ `integer MySQL_PConnect(string` *«počítač»*`, string` *«uživatel»*`,`
`string` *«heslo»*`)`

Funkce vrací číslo spojení s databází MySQL, které je využíváno v ostatních funkcích pro práci s databází MySQL. Jako *«počítač»* uvádíme adresu počítače, na kterém běží MySQL server. Pokud server běží na nestandardním portu, můžeme jej přidat do identifikátoru počítače za dvojtečku (`db-server.firma.cz:4478`). V případě, že je přístup ke zdroji vázán na jméno a heslo, musíme je zadat. Pokud se spojení nepodaří vytvořit, vrací funkce `false`.

Libovolný z parametrů můžeme vynechat. V tom případě se použijí standardní hodnoty: počítač `localhost`, uživatel bude odpovídat uživateli, pod kterým běží právě spuštěný skript, a heslo bude prázdné.

Pokud funkci `MySQL_PConnect()` zavoláme podruhé se stejnými parametry, není vytvořeno nové spojení a je vráceno již existující číslo spojení.

Spojení se serverem není automaticky ukončeno po ukončení běhu skriptu. Další požadavky na připojení k MySQL využívají již otevřené spojení. Použití této funkce může urychlit provádění skriptů, které se často připojují k databázi.

Persistentní spojení pracují pouze, pokud PHP běží jako modul serveru. Pokud funkci využijeme v PHP, které je spuštěno jako CGI--skript, bude se chovat stejně jako `MySQL_Connect()`*.*

`MySQL_Connect()` na straně 360.

MySQL_Query

Funkce pro práci s databází MySQL

Vykonání SQL-příkazu

§ `integer MySQL_Query(string` *«SQL-příkaz»*`, integer` *«spojení»*`)`

Funkce provede na daném *«spojení» «SQL-příkaz»* nad právě aktivní databází a vrátí identifikátor výsledku. Pokud při provádění příkazu došlo k chybě, vrací funkce `false`. Pokud parametr *«spojení»* neuvedeme, použije se poslední vytvořené spojení. Pokud žádné spojení neexistuje, pokusí se funkce vytvořit spojení s pomocí standardních parametrů jako u funkce `MySQL_Connect()`.

MySQL_Connect() na straně 360, MySQL_Select_DB() na straně 371 a MySQL_DB_Query() na straně 362.

Ukázky použití naleznete na stranách 147, 148.

MySQL_Result

Funkce pro práci s databází MySQL

Získání hodnoty jedné položky výsledku dotazu

§ `string MySQL_Result(integer` *«výsledek»*`, integer` *«záznam»*`,`
`mixed` *«položka»*`)`

Funkce slouží k získání položky *«výsledku»*. Položku musíme určit číslem *«záznamu»* a číslem nebo jménem *«položky»*. Položky i záznamy jsou číslovány od nuly. Pokud *«položku»* vynecháme, vrátí funkce první položku daného záznamu. Pokud se funkci položku nepodaří přečíst, vrací `false`.

> Používání této funkce zejména na větší výsledky je poměrně pomalé. Mnohem rychlejší je využití funkcí `MySQL_Fetch_Row()`, `MySQL_Fetch_Array()` a `MySQL_Fetch_Object()`, které načítají celý záznam najednou.

MySQL_Fetch_Row() na straně 366, MySQL_Fetch_Array() na straně 363 a MySQL_Fetch_Object() na straně 365.

MySQL_Select_DB

Funkce pro práci s databází MySQL

Výběr aktivní databáze

§ `integer MySQL_Select_DB(string` *«databáze»*`, integer` *«spojení»*`)`

Funkce nastaví aktivní *«databázi»* pro *«spojení»*. Pokud druhý parametr nepoužijeme, automaticky se dosadí poslední vytvořené spojení. Pokud žádné spojení neexistuje, pokusí se funkce vytvořit spojení s pomocí standardních parametrů jako u funkce `MySQL_Connect()`.

Aktivní databáze má význam ve funkci `MySQL_Query()`, která provádí SQL-příkaz právě nad aktivní databází.

MySQL_Query() na straně 370, MySQL_DB_Query() na straně 362 a MySQL_Connect() na straně 360.

MySQL_TableName

Funkce pro práci s databází MySQL

Přečtení jména tabulky

§ `string MySQL_DBName(integer` *«výsledek»*`, integer` *«pořadí»*`)`

Funkce vrací jméno tabulky. *«Výsledek»* obsahuje seznam všech tabulek získaný pomocí funkce `MySQL_List_Tables()`. *«Pořadí»* určuje pořadí tabulky ve výsledku (začíná se od 0).

Vypsání všech tabulek v *«databázi»*:

```
MySQL_Connect(«počítač», «uživatel», «heslo»);
$vysledek = MySQL_List_Tables(«databáze»);
for ($i=0; $i < MySQL_Num_Rows($vysledek); $i++)
    echo MySQL_TableName($vysledek, $i)."<BR>\n";
```

`MySQL_List_Tables()` na straně 369.

Next

Funkce pro práci s poli

Vrací hodnotu následujícího prvku pole

§ `mixed Next(array` *«pole»*`)`

Funkce posune ukazatel *«pole»* na další prvek a vrátí jeho obsah. Funkce vrací `false`, pokud dosáhne za konec pole.

> Funkce vrací `false`, pokud i prvek pole obsahuje `false`. To může znemožnit průchod celého pole příkazem typu `while (Next(...)) ....` Pro bezpečný průchod celým polem bychom měli používat funkci `Each()`.

`Each()` na straně 282, `Prev()` na straně 399, `Current()` na straně 271, `Reset()` na straně 405, `End()` na straně 284 a `Array_Walk()` na straně 259.

Ukázky použití naleznete na straně 44.

NL2BR

Funkce pro práci s textovými řetězci

Převádí konce řádků v řetězci na tag

§ `string NL2BR(string «řetězec»)`

Funkce v *«řetězci»* nahradí všechny znaky konce řádku '`\n`' tagem <BR>.

Následující skript vypíše obsah souboru:

```
$fp = fopen("soubor.txt", "r");
while (!feof($fp))
    echo NL2BR(HTMLSpecialChars(fgets($fp, 1024)));
fclose($fp);
```

`HTMLSpecialChars()` na straně 308 a `HTMLEntities()` na straně 308.

Ukázky použití naleznete na stranách 104, 199, 207.

Number_Format

Matematická funkce

Formátování čísla pro ekonomické výstupy

§ `string Number_Format(double «výraz», integer «des. místa»,`
`string «des. čárka», string «odd. tisíců»)`

Funkce vrací číslo zformátované pro potřeby ekonomických výstupů. Funkce může být volána s jedním, dvěma nebo čtyřmi parametry. *«Výraz»* obsahuje číslo, které chceme zformátovat. Počet desetinných míst použitých při formátování bude *«des. místa»*.

Pokud nepoužijeme třetí a čtvrtý parametr, jako desetinná čárka se použije znak '`.`' a jako oddělovač tisíců se použije čárka '`,`'. Pokud máme jinou představu, můžeme znak použitý pro desetinnou čárku určit pomocí *«des. čárka»* a oddělovač tisíců pomocí *«odd. tisíců»*.

V Čechách je zvykem používat desetinnou čárku a tisíce oddělovat tečkou. Desetinná místa se pro potřeby účetnictví používají většinou dvě. Můžeme si tedy definovat funkci `Částka()`, která nám vrátí číslo zformátované přesně podle našich zvyklostí:

```
function Částka($n)
{
    return Number_Format($n, 2, ",", ".");
}
```

OctDec

Matematická funkce

Převod osmičkového čísla na desítkové

§ `integer OctDec(string «osmičkové číslo»)`

Převede *«osmičkové číslo»* uložené v řetězci na desítkové číslo. Osmičkové číslo může být v rozsahu od 0 do 17777777777. Pokus o převod jiných čísel vrátí 0.

`DecOct()` na straně 280, `BinDec()` na straně 264, `DecBin()` na straně 280, `HexDec()` na straně 307 a `DecHex()` na straně 280.

ODBC_AutoCommit

Funkce pro práci s datovými zdroji ODBC

Nastavení automatického potvrzování transakcí

§ `integer ODBC_AutoCommit(integer «spojení», integer «ano/ne»)`

Pro dané *«spojení»* s ODBC zdrojem nastaví způsob potvrzování transakcí. Normálně se všechny operace potvrzují po určité době automaticky. Pokud chceme používat transakce, musíme tuto vlastnost vypnout tím, že jako hodnotu parametru *«ano/ne»* zadáme `false`.

Pokud se podaří změnit způsob potvrzování transakcí, vrací funkce `true`. V opačném případě `false`.

`ODBC_Commit()` na straně 376 a `ODBC_RollBack()` na straně 384.

Ukázky použití naleznete na straně 173.

ODBC_BinMode

Funkce pro práci s datovými zdroji ODBC

Nastavení režimu práce s binárními položkami

§ `integer ODBC_BinMode(integer «výsledek», integer «režim»)`

Nastaví režim práce s binárními položkami `BINARY`, `VARBINARY` a `LONGVARBINARY`. K dispozici jsou následující tři *«režimy»*:

0 binární data jsou zapsána přímo na výstup;
1 binární data jsou vrácena v neupravené podobě;
2 binární data jsou převedena na znakový tvar.

Pokud jsou data převáděna na znakový tvar, je každý bajt dat převeden na dva znaky, které obsahují hodnotu bajtu zapsanou v šestnáctkové soustavě.

Pokud jako *«výsledek»* použijeme 0, nastavený režim se použije pro všechny nově získávané výsledky.

BinMode	LongReadLen	Popis
0	0	zápis dat na standardní výstup
1	0	zápis dat na standardní výstup
2	0	zápis dat na standardní výstup
0	> 0	zápis dat na standardní výstup
1	> 0	vrácení dat v neupravené podobě
2	> 0	vrácení dat ve znakovém formátu

Tab. 9-5: Nastavení práce s binárními daty

Chování binárních položek kromě `ODBC_BinMode()` ovlivňuje i nastavení `ODBC_LongReadLen()`. Jejich vzájemné kombinace a výsledné chování přináší tabulka 9-5. Standardní nastavení je 1 pro `BinMode` a 4 096 pro `LongReadLen`.

ODBC_Close

Funkce pro práci s datovými zdroji ODBC

Uzavření spojení s ODBC datovým zdrojem

§ `void ODBC_Close(integer` *«spojení»*`)`

Funkce uzavře *«spojení»* s datovým zdrojem. Pokud spojení obsahuje neuzavřené transakce (nepotvrzené jednou z funkcí `ODBC_Commit()` nebo `ODBC_RollBack()`), spojení s datovým zdrojem se neukončí.

`ODBC_Commit()` na straně 376 a `ODBC_RollBack()` na straně 384.

Ukázky použití naleznete na straně 151.

ODBC_Close_All

Funkce pro práci s datovými zdroji ODBC

Uzavření všech ODBC spojení

§ `void ODBC_Close_All(void)`

Funkce uzavře všechna spojení s datovými zdroji ODBC. Pokud některé spojení obsahuje neuzavřené transakce (nepotvrzené jednou z funkcí `ODBC_Commit()` nebo `ODBC_RollBack()`), spojení s datovým zdrojem se neukončí.

`ODBC_Close()` na straně 375, `ODBC_Commit()` na straně 376 a `ODBC_RollBack()` na straně 384.

ODBC_Commit

Funkce pro práci s datovými zdroji ODBC

Potvrzení transakce

§ `integer ODBC_Commit(integer «spojení»)`

Funkce potvrdí a uzavře všechny transakce na daném *«spojení»*. Pokud jsou všechny transakce úspěšně potvrzeny a ukončeny, vrací funkce `true`, v opačném případě `false`.

`ODBC_AutoCommit()` na straně 374 a `ODBC_RollBack()` na straně 384.

Ukázky použití naleznete na stranách 173, 375.

ODBC_Connect

Funkce pro práci s datovými zdroji ODBC

Vytvoří spojení s datovým zdrojem

§ `integer ODBC_Connect(string «DSN», string «uživatel», string «heslo»)`

Funkce vrací číslo spojení s datovým zdrojem, které je využíváno v ostatních funkcích pro práci s ODBC. Datový zdroj, ke kterému se chceme připojit, určujeme pomocí jeho jména *«DSN»*. V případě, že je přístup ke zdroji vázán na jméno a heslo, musíme je zadat. Pokud se spojení nepodaří vytvořit, vrací funkce `false`.

Ukázky použití naleznete na stranách 35, 150, 151, 177, 470.

ODBC_Cursor

Funkce pro práci s datovými zdroji ODBC

Zjištění jména kurzoru pro výsledek

§ `string ODBC_Cursor(integer «výsledek»)`

Zjistí jméno kurzoru pro daný *«výsledek»*.

ODBC_Do

Funkce pro práci s datovými zdroji ODBC

Vykonání SQL-příkazu

§ `integer ODBC_Do(integer` *«spojení»*`, string` *«SQL-příkaz»*`)`

Funkce provede *«SQL-příkaz»* na daném *«spojení»*. Jde o synonymum k funkci `ODBC_Exec()`.

`ODBC_Exec()` na straně 377.

ODBC_Exec

Funkce pro práci s datovými zdroji ODBC

Vykonání SQL-příkazu

§ `integer ODBC_Exec(integer` *«spojení»*`, string` *«SQL-příkaz»*`)`

Funkce provede na daném *«spojení»* *«SQL-příkaz»* a vrátí identifikátor výsledku. Pokud při provádění příkazu došlo k chybě, vrací funkce `false`.

`ODBC_Prepare()` na straně 382 a `ODBC_Execute()` na straně 377.

Ukázky použití naleznete na stranách 35, 150, 151, 173.

ODBC_Execute

Funkce pro práci s datovými zdroji ODBC

Vykonání předzpracovaného SQL-příkazu

§ `integer ODBC_Execute(integer` *«výsledek»*`, array` *«parametry»*`)`

Funkce provede SQL-příkaz, který byl již dříve připraven funkcí `ODBC_Prepare()`. Jako parametr *«výsledek»* se používá právě hodnota již dřívě vrácená funkcí `ODBC_Prepare()`. Pole *«parametry»* obsahuje jednotlivé hodnoty, kterými se v předpřipraveném dotazu nahradí znaky '`?`'. Výsledkem dotazu je `true`, pokud se dotaz provedl úspěšně. Pokud došlo k chybě, je vráceno `false`.

Tím, že funkce neumí vracet data z tabulky, hodí se pouze na provádění takových příkazů jako `INSERT`, `UPDATE` a `DELETE`.

`ODBC_Prepare()` na straně 382.

Ukázky použití naleznete na straně 382.

ODBC_Fetch_Into

Funkce pro práci s datovými zdroji ODBC

Načtení záznamu do pole

§ `integer ODBC_Fetch_Into(integer «výsledek», integer «záznam», array «pole»)`

Funkce do *«pole»* uloží jednotlivé položky *«záznamu»*. Funkce vrací počet položek uložených do pole. Pokud při běhu funkce došlo k chybě, vrací funkce `false`. *«Výsledek»* určuje výsledek dotazu, ze kterého bude záznam přečten. Pole pro uložení záznamu musí být předáno odkazem.

Většina databázových serverů neumí vrátit přímo požadovaný záznam, a tak použití parametru *«záznam»* ignorují a vrací vždy další záznam.[1] Parametr *«záznam»* můžeme vynechat — je nepovinný. Záznamy se číslují od jedné.

Způsobů, jak vypsat výsledek dotazu, je mnoho. Jedna z možností staví právě na funkci `ODBC_Fetch_Into()`:

```
$spojeni = ODBC_Connect("«zdroj»", "", "");
$vysledek = ODBC_Exec($spojeni, "select * from «tabulka»");
while ($n = ODBC_Fetch_Into($vysledek, &$x)):
    for ($i=0; $i<$n; $i++)
        echo $x[$i]." ";
    echo "<BR>\n";
endwhile;
ODBC_Close($spojeni);
```

`ODBC_Fetch_Row()` na straně 378.

ODBC_Fetch_Row

Funkce pro práci s datovými zdroji ODBC

Načtení záznamu výsledku

§ `integer ODBC_Fetch_Row(integer «výsledek», integer «záznam»)`

Pokud je funkce volána pouze s prvním parametrem, načte další záznam *«výsledku»*. Pokud použijeme parametr *«záznam»*, načte se určitý záznam výsledku. Záznamy jsou přitom číslovány od jedničky. Načítání konkrétního záznamu však nepodporují všechny ODBC ovladače — v tomto případě je parametr *«záznam»* ignorován a je načtena další řádka.

[1] S parametrem si správně poradí pouze ty ODBC ovladače, které podporují funkci `SQLExtendedFetch`.

Funkce vrací `true`, pokud se záznam podařilo načíst. V opačném případě vrací `false`. Jednotlivé položky načteného záznamu získáme pomocí funkce `ODBC_Result()`.

Způsobů, jak vypsat výsledek dotazu, je mnoho. Jedna z možností staví právě na funkci `ODBC_Fetch_Row()`:

```
$spojeni = ODBC_Connect("«zdroj»", "", "");
$vysledek = ODBC_Exec($spojeni, "select * from «tabulka»");
while (ODBC_Fetch_Row($vysledek)):
    for ($i=1; $i<=ODBC_Num_Fields($vysledek); $i++)
        echo ODBC_Result($vysledek, $i)." ";
    echo "<BR>\n";
endwhile;
ODBC_Close($spojeni);
```

`ODBC_Fetch_Into()` na straně 378.

Ukázky použití naleznete na straně 35.

ODBC_Field_Name

Funkce pro práci s datovými zdroji ODBC

Zjištění jména položky

§ `string ODBC_Field_Name(integer «výsledek», integer «číslo_položky»)`

Funkce vrací jméno položky, která je součástí *«výsledku»* a jejíž číslo je *«číslo_položky»*. Položky jsou číslovány od jedné. Pokud dojde k chybě, vrací funkce `false`.

S využitím této funkce není problém napsat skript, který zobrazí libovolnou tabulku i s názvy jednotlivých položek:

```
$spojeni = ODBC_Connect("«zdroj»", "", "");
$vysledek = ODBC_Exec($spojeni, "select * from «tabulka»");
echo "<TABLE>\n<TR>";
for ($i=1; $i<=ODBC_Num_Fields($vysledek); $i++)
    echo "<TH>".UCFirst(ODBC_Field_Name($vysledek, $i));
echo "\n";
while (ODBC_Fetch_Row($vysledek)):
    echo "<TR>";
    for ($i=1; $i<=ODBC_Num_Fields($vysledek); $i++)
        echo "<TD>".ODBC_Result($vysledek, $i);
    echo "\n";
```

```
endwhile;
echo "</TABLE>\n";
ODBC_Close($spojeni);
```

`ODBC_Field_Num()` na straně 380.

ODBC_Field_Num Funkce pro práci s datovými zdroji ODBC

Zjištění čísla položky

§ `integer ODBC_Field_Num(integer` *«výsledek»*`, string` *«jméno_položky»*`)`

Funkce vrací číslo položky, která je součástí *«výsledku»* a jejíž jméno je *«jméno_položky»*. Položky jsou číslovány od jedné. Pokud dojde k chybě, vrací funkce `false`.

`ODBC_Field_Name()` na straně 379.

ODBC_Field_Type Funkce pro práci s datovými zdroji ODBC

Zjištění typu položky

§ `string ODBC_Field_Type(integer` *«výsledek»*`, integer` *«číslo_položky»*`)`

Funkce vrací typ položky, která je součástí *«výsledku»* a jejíž číslo je *«číslo_položky»*. Položky jsou číslovány od jedné. Pokud dojde k chybě, vrací funkce `false`.

ODBC_Free_Result Funkce pro práci s datovými zdroji ODBC

Uvolnění výsledku z paměti

§ `integer ODBC_Free_Result(integer` *«výsledek»*`)`

Funkce uvolní daný *«výsledek»* z paměti. Pokud je zakázáno automatické potvrzování transakcí, musíme před voláním `ODBC_Free_Result()` potvrdit transakce voláním `ODBC_Commit()`. V opačném případě budou všechny operace s datovým zdrojem zrušeny. Funkce vždy vrací hodnotu `true`.

Funkci ve většině skriptů není třeba volat, protože se paměť automaticky uvolní po skončení běhu skriptu. Volání má smysl pouze v dlouhých skriptech, kdy chceme minimalizovat paměťové nároky.

ODBC_LongReadLen

Funkce pro práci s datovými zdroji ODBC

Nastaví maximální počet bajtů čtených z dlouhých položek

§ `integer ODBC_LongReadLen(integer` *«výsledek»*`, integer` *«počet»*`)`

Funkce nastaví maximální *«počet»* bajtů/znaků, které budou vraceny jako obsah položek typu `LONGVARCHAR` a `LONGVARBINARY`. Pokud nastavíme *«počet»* na nulu, nebude délka položky omezena a čtená data budou zapsána přímo na výstup skriptu. Funkce vrací `true`, pokud se hodnotu podaří změnit, v opačném případě vrací `false`.

`ODBC_BinMode()` na straně 374.

Ukázky použití naleznete na stranách 175, 470.

ODBC_Num_Fields

Funkce pro práci s datovými zdroji ODBC

Zjistí počet položek ve výsledku

§ `integer ODBC_Num_Fields(integer` *«výsledek»*`)`

Funkce vrací počet položek ve *«výsledku»*. Pokud dojde k chybě, vrací funkce −1.

ODBC_PConnect

Funkce pro práci s datovými zdroji ODBC

Vytvoří persistentní spojení s datovým zdrojem

§ `integer ODBC_PConnect(string` *«DSN»*`, string` *«uživatel»*`,`
`string` *«heslo»*`)`

Funkce vrací číslo spojení s datovým zdrojem, které je využíváno v ostatních funkcích pro práci s ODBC. Datový zdroj, ke kterému se chceme připojit, určujeme pomocí jeho jména *«DSN»*. V případě, že je přístup ke zdroji vázán na jméno a heslo, musíme je zadat. Pokud se spojení nepodaří vytvořit, vrací funkce `false`.

Funkce pracuje podobně jako `ODBC_Connect()`. Rozdíl je v tom, že spojení s databází není po skončení skriptu uzavřeno. Další požadavky na připojení k datovému zdroji se stejným *«DSN»*, *«uživatelem»* a *«heslem»* využívají již otevřené spojení. Použití této funkce může urychlit provádění skriptů, které se často připojují k databázi.

Persistentní spojení pracují pouze tehdy, pokud PHP běží jako modul serveru. Pokud funkci využijeme v PHP, které je spuštěno jako CGI--skript, bude se chovat stejně jako `ODBC_Connect()`.

`ODBC_Connect()` na straně 376.

Ukázky použití naleznete na stranách 177, 470.

ODBC_Prepare

Funkce pro práci s datovými zdroji ODBC

Funkce připraví SQL-příkaz pro opakované provedení

§ `integer ODBC_Prepare(integer «spojení», string «příkaz»)`

Funkce připraví *«příkaz»* pro pozdější provedení. Pokud funkce proběhla úspěšně, vrací identifikátor výsledku. Pokud při provádění došlo k chybě, vrací funkce `false`. Parametr *«spojení»* obsahuje číslo spojení s datovým zdrojem.

V příkazu můžeme použít otazník '`?`' na místech, kde chceme dosazovat různé hodnoty při provádění příkazu pomocí `ODBC_Execute()`.

Malá ukázka použití použití `ODBC_Prepare()` a `ODBC_Execute()`:

```
$spojeni = ODBC_Connect("«zdroj»", "", "");
$vysledek = ODBC_Prepare($spojeni, "INSERT INTO Adresar VALUES (?, ?)");
ODBC_Execute($vysledek, array("Jan", "Novák"));
```

`ODBC_Execute()` na straně 377.

ODBC_Num_Rows

Funkce pro práci s datovými zdroji ODBC

Vrací počet záznamů výsledku

§ `integer ODBC_Num_Rows(integer «výsledek»)`

Funkce vrací počet záznamů, které obsahuje *«výsledek»*. Pokud při volání funkce dojde k chybě, vrací funkce −1.

Po volání SQL-příkazů `INSERT`, `UPDATE` a `DELETE` funkce vrací počet záznamů, které byly příkazem ovlivněny (vloženy, změněny či smazány).

Mnoho ODBC ovladačů bude vždy při volání vracet hodnotu −1, protože tuto funkci nepodporují. Naše skripty by proto neměly na volání `ODBC_Num_Rows()` spoléhat. Pro průchod celým výsledkem použijeme postupné volání funkcí `ODBC_Fetch_Row()` nebo `ODBC_Fetch_Into()`.

Pokud potřebujeme znát počet záznamů, které vyhovují určitému dotazu, můžeme to obejít použitím funkce `Count()` jazyka SQL. Místo

```
$vysledek = ODBC_Exec($spojeni, "SELECT Jmeno, Plat FROM Zamestnanci
                                 WHERE Plat > 10000");
$pocetDobrePlacenych = ODBC_Num_Rows($vysledek);
```

můžeme použít

```
$vysledek = ODBC_Exec($spojeni, "SELECT Count(*) FROM Zamestnanci
                                 WHERE Plat > 10000");
ODBC_Fetch_Row($vysledek);
$pocetDobrePlacenych = ODBC_Result($vysledek, 1);
ODBC_Free_Result($vysledek);
$vysledek = ODBC_Exec($spojeni, "SELECT Jmeno, Plat FROM Zamestnanci
                                 WHERE Plat > 10000");
```

Ukázky použití naleznete na stranách 35, 152.

ODBC_Result

Funkce pro práci s datovými zdroji ODBC

Získání jedné položky výsledku

§ `string ODBC_Result(integer` *«výsledek»*`, mixed` *«položka»*`)`

Funkce vrací obsah jedné položky právě aktuálního záznamu *«výsledku»*. Položku můžeme určit buď jejím pořadovým číslem, nebo jménem. Pořadová čísla se počítají od jedné a položky jsou vždy uspořádány v tom pořadí, jak jsou uvedeny za příkazem `SELECT`.

`ODBC_BinMode()` na straně 374, `ODBC_LongReadLen()` na straně 381 a `ODBC_Result_All()` na straně 384.

Ukázky použití naleznete na stranách 35, 150, 175, 176.

ODBC_Result_All

Funkce pro práci s datovými zdroji ODBC

Vypsání celého výsledku dotazu ve formě HTML-tabulky

§ `integer ODBC_Result_All(integer «výsledek», string «formát»)`

Funkce zobrazí *«výsledek»* ve formě tabulky. Pomocí nepovinného parametru *«formát»* můžeme určit nastavení atributů, které se použije u tagu `<TABLE>`. Funkce vrací počet záznamů, které byly zobrazeny.

Pokud chceme výsledek zobrazit ve žluté tabulce s rámečkem, použijeme:

```
$spojeni = ODBC_Connect(«DSN», «uživatel», «heslo»);
$vysledek = ODBC_Exec($spojeni, "SELECT * FROM «tabulka»");
$pocetZaznamu = ODBC_Result_All($vysledek,
                              "FRAME=BOX RULES=NONE BGCOLOR=YELLOW");
echo "Tabulka «tabulka» obsahuje $pocetZaznamu záznamů.";
```

`ODBC_Result()` na straně 383.

ODBC_RollBack

Funkce pro práci s datovými zdroji ODBC

Zrušení rozpracované transakce

§ `integer ODBC_RollBack(integer «spojení»)`

Funkce zruší všechny příkazy právě prováděné transakce na *«spojení»*. Pokud se příkazy podaří zrušit, vrací funkce `true`. V opačném případě vrací `false`.

`ODBC_AutoCommit()` na straně 374 a `ODBC_Commit()` na straně 376.

Ukázky použití naleznete na stranách 173, 375.

OpenDir

Funkce pro práci s adresáři

Otevření adresáře

§ `integer OpenDir(string «adresář»)`

Funkce vrací ukazatel *«dp»*, který můžeme používat v dalších funkcích, které umožňují pracovat s obsahem adresářů. Ukazatel ukazuje na první položku *«adresáře»*. Funkce vrací `false`, pokud se adresář nepodařilo otevřít.

`CloseDir()` na straně 268, `ReadDir()` na straně 404 a `RewindDir()` na straně 406.

OpenLog

Konfigurace a informace o PHP

Vytvoření přístupu k protokolu systémových událostí

§ `void OpenLog(string` *«identifikátor»*`, integer` *«volby»*`, integer` *«možnosti»*`)`

Funkce umožní skriptu přístup do protokolu systémových událostí. *«indentifikátor»* je řetězec, který slouží k identifikaci námi generovaných událostí — jeho obsahem může být např. PHP nebo jméno našeho skriptu.

Další dva parametry umožňují poměrně bohatě konfigurovat možnosti zapisování událostí do protokolu. Parametr *«volby»* můžeme snadno nastavit s využitím následujících konstant:

`LOG_PID`	Do protokolu se bude zapisovat číslo procesu.
`LOG_CONS`	V případě problémů se zápisem do souboru s protokolem vypíše zprávu rovnou na konzoli.
`LOG_ODELAY`	Počkej s otevřením protokolu až do prvního volání funkce `SysLog()` — standardní nastavení.
`LOG_NDELAY`	Okamžité otevření protokolu.
`LOG_NOWAIT`	Nečekej na vytvoření procesu pro konzoli.
`LOG_PERROR`	Hlášení navíc zapisuj na standardní chybový výstup.

Jednotlivé volby můžeme navzájem kombinovat, nejlépe tak, že při volání funkce `OpenLog()` použijeme logický součet konstant:

```
OpenLog("PHP", LOG_PID|LOG_CONS, 0)
```

Pro nastavení *«možností»* máme k dispozici také několik předdefinovaných konstant, které udávají, od koho pochází zpráva zapsaná do protokolu.

`LOG_KERN`	jádro systému;
`LOG_USER`	normální uživatelský proces;
`LOG_MAIL`	poštovní systém;
`LOG_DAEMON`	systémový démon;
`LOG_AUTH`	autorizační systém (login, su apod.);
`LOG_SYSLOG`	protokolovací systém;
`LOG_LPR`	tiskový systém;
`LOG_NEWS`	subsystém síťových news;
`LOG_UUCP`	subsystém UUCP;
`LOG_CRON`	systém opakovaně spouštěných úloh;
`LOG_AUTHPRIV`	
`LOG_LOCAL0–7`	uživatelem definované.

Volání funkce `OpenLog()` není nezbytné. Funkce je vyvolána automaticky, pokud poprvé použijeme `SysLog()` — nemáme pak ovšem možnost ovlivnit parametry ovlivňující zápis zpráv do protokolu.

`SysLog()` na straně 423 a `CloseLog()` na straně 268.

Ord Funkce pro práci s textovými řetězci

Vrací ASCII-kód prvního znaku v řetězci

§ `integer Ord(string` *«znak»*`)`

Vrací ASCII-kód prvního znaku v řetězci *«znak»*. Funkce je opakem funkce `Chr()`.

`Chr()` na straně 267.

Parse_Str Funkce pro práci s textovými řetězci

Analyzuje obsah řetězce a uloží jej do proměnných

§ `void Parse_Str(string` *«řetězec»*`)`

Převede obsah *«řetězce»* do proměnných tak, jako kdyby *«řetězec»* byl dotaz uvedený na konci URL.

Malý příklad vše ozřejmí:

```
$str = "jmeno=Novák+Jan&jazyky[]=NJ&jazyky[]=AJ";
Parse_Str($str);
echo $jmeno;                // vypíše Jan Novák
echo $jazyky[0]; // vypíše NJ
echo $jazyky[1]; // vypíše AJ
```

Parse_URL Funkce pro práci s URL adresami

Zjištění jednotlivých částí URL

§ `array Parse_URL(string` *«URL»*`)`

Funkce rozloží *«URL»* na jednotlivé části a ty uloží do asociativního pole. Jaké indexy asociativního pole máme k dispozici si ukážeme na příkladě URL:

```
http://guest:anonymous@www.kdesi.cz:9001/pub/lib/search?Jan+Nov%E1k#f13
```

Po zavolání funkce získáme asociativní pole s následujícími indexy:

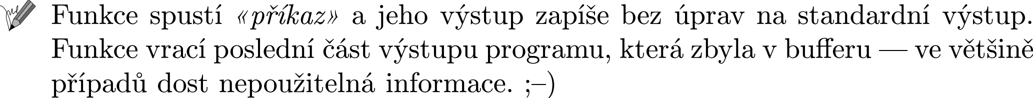
`scheme`	použité schéma (`http`);
`host`	adresa počítače (`www.kdesi.cz`);
`port`	port služby (`9001`);
`user`	uživatelské jméno pro přístup ke službě (`guest`);
`pass`	heslo (`anonymous`);
`path`	cesta k dokumentu (`pub/lib/search`);
`query`	dotaz (`Jan+Nov%E1k`);
`fragment`	fragment (`f13`).

PassThru

Funkce pro spouštění externích programů

Spuštění externího programu a zobrazení neupraveného výstupu

§ `string PassThru(string` *«příkaz»*`, integer` *«status»*`)`



Funkce spustí *«příkaz»* a jeho výstup zapíše bez úprav na standardní výstup. Funkce vrací poslední část výstupu programu, která zbyla v bufferu — ve většině případů dost nepoužitelná informace. ;–)

Druhý parametr *«status»* je nepovinný a slouží ke zjištění návratového kódu programu. Parametr by měl být opět předáván odkazem.

Funkci využijeme zejména v těch případech, kdy pomocí PHP generuje binární data — např. obrázky. V PHP odešleme HTTP hlavičku určující typ dat a poté spustíme program, který vygeneruje např. obrázek.

Pokud *«příkaz»* obsahuje parametry zadávané uživatelem, měli bychom na příkaz aplikovat funkci `EscapeShellCmd()`. Uživateli tak zabráníme v „ošálení“ příkazu.

`Exec()` na straně 288, `System()` na straně 423, `EscapeShellCmd()` na straně 288 a `POpen()` na straně 398.

PClose

Funkce pro práci se soubory

Funkce uzavře ukazatel na rouru

§ `integer PClose(integer` *«fp»*`)`

Funkce uzavře rouru vytvořenou pomocí funkce `POpen()`. Výsledkem funkce je návratový kód ukončeného programu.

`POpen()` na straně 398.

Pg_Close

Funkce pro práci s databází PostgreSQL

Uzavření spojení se serverem

§ `integer Pg_Close(integer «spojení»)`

Funkce uzavře *«spojení»* se serverem. Pokud se spojení podaří uzavřít, vrací funkce `true`. V opačném případě `false`. Pokud funkci zavoláme bez parametru, uzavře se poslední otevřené spojení.

`Pg_Connect()` na straně 388.

Ukázky použití naleznete na straně 152.

Pg_CmdTuples

Funkce pro práci s databází PostgreSQL

Vrací počet záznamů ovlivněných posledním příkazem

§ `integer Pg_CmdTuples(integer «výsledek»)`

Funkce vrací počet záznamů ovlivněných příkazem `INSERT`, `UPDATE` a `DELETE`. Pokud nedošlo k žádné změně, vrací funkce 0.

`Pg_CmdTuples()` na straně 388.

Pg_Connect

Funkce pro práci s databází PostgreSQL

Připojení k databázi PostgreSQL

§ `integer Pg_Connect(string «počítač», string «port», string «volby»,`
`string «tty», string «databáze»)`

Funkce vrací číslo spojení, které se používá v dalších funkcích pro práci s databází PostgreSQL. Pokud se spojení nepodaří vytvořit, vrací funkce `false`. Parametry *«volby»* a *«tty»* můžeme vynechat — obvykle se tedy spojení vytváří pomocí příkazu:

```
$spojeni = Pg_Connect("localhost", "5432", "dbtest")
```

Další možností je použít pouze jeden parametr. V tomto případě je parametr chápán jako řetězec, který může obsahovat nastavení všech možných parametrů:

```
$spojeni = Pg_Connect("host=localhost port=5432  dbname=dbtest
                       user=john password=WalKERx3")
```

Kromě parametrů `host`, `port`, `dbname`, `user` a `password` můžeme použít `options` a `tty`.

Pg_PConnect() na straně 395.

Ukázky použití naleznete na stranách 152, 153, 177.

Pg_DBName Funkce pro práci s databází PostgreSQL

Zjištění jména databáze, ke které jsme připojeni

§ `string Pg_DBName(integer` *«spojení»*`)`

Funkce vrací název databáze, ke které jsme připojeni. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

Pg_ErrorMessage Funkce pro práci s databází PostgreSQL

Zjištění chybového hlášení

§ `string Pg_ErrorMessage(integer` *«spojení»*`)`

Funkce vrací text posledního chybového hlášení vráceného serverem pro *«spojení»*. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

Pg_Exec Funkce pro práci s databází PostgreSQL

Provedení SQL-příkazu

§ `integer Pg_Exec(integer` *«spojení»*`, string` *«SQL-příkaz»*`)`

Funkce na daném *«spojení»* provede *«SQL-příkaz»*. Funkce vrací identifikátor výsledku. Pokud dojde k chybě, vrací funkce `false`. Pokud funkci zavoláme bez parametru *«spojení»*, použije se poslední otevřené spojení.

Ukázky použití naleznete na stranách 152, 153.

Pg_Fetch_Array Funkce pro práci s databází PostgreSQL

Načtení záznamu do asociativního pole

§ `array Pg_Fetch_Array(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`)`

Funkce načte jeden záznam výsledku do pole. Jednotlivé položky jsou uloženy do prvků jejichž index odpovídá názvu položky. Kromě toho jsou položky uloženy do prvků, jejichž index odpovídá pořadí položky v záznamu. Pokud při čtení záznamu dojde k chybě, vrací funkce `false`. Záznamy jsou číslovány od nuly.

`Pg_Fetch_Row()` na straně 390 a `Pg_Fetch_Array()` na straně 389.

Ukázky použití naleznete na straně 152.

Pg_Fetch_Object Funkce pro práci s databází PostgreSQL

Načte záznam výsledku do objektu

§ `object Pg_Fetch_Object(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`)`

Funkce načte jeden záznam *«výsledku»* do objektu. Obsah každé položky je uložen jako členská proměnná, jejíž název odpovídá názvu položky. Pokud při čtení záznamu dojde k chybě, vrací funkce `false`. Záznamy jsou číslovány od nuly.

`Pg_Fetch_Row()` na straně 390 a `Pg_Fetch_Array()` na straně 389.

Pg_Fetch_Row Funkce pro práci s databází PostgreSQL

Načte záznam výsledku do pole

§ `array Pg_Fetch_Row(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`)`

Funkce načte jeden záznam *«výsledku»* do pole. Obsah každé položky je uložen do jednoho prvku pole. Položky jsou ukládány postupně počínaje indexem 0. Pokud při čtení záznamu dojde k chybě (např. již nejsou žádné další záznamy k přečtení), vrací funkce `false`. Záznamy jsou číslovány od nuly.

`Pg_Fetch_Array()` na straně 389 a `Pg_Fetch_Object()` na straně 390.

Pg_FieldIsNull Funkce pro práci s databází PostgreSQL

Test, zda je položka NULL

§ `integer Pg_FieldIsNull(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`,`
`mixed` *«položka»*`)`

Funkce vrací `true`, pokud je položka prázdná (obsahuje hodnotu NULL). V opačném případě vrací `false`. Položka je čtena ze záznamu, který je určen *«výsledkem»* a *«číslem_záznamu»* (číslováno od nuly). Samotnou položku můžeme určit buď jejím pořadím ve výsledku (číslováno od nuly), nebo jménem.

Pg_FieldName Funkce pro práci s databází PostgreSQL

Zjištění jména položky

§ `string Pg_FieldName(integer` *«výsledek»*`, integer` *«číslo_položky»*`)`

Funkce vrací jméno položky, která je součástí *«výsledku»*. Položky jsou číslovány od nuly.

Pg_FieldNum Funkce pro práci s databází PostgreSQL

Zjištění čísla položky

§ `integer Pg_FieldNum(integer` *«výsledek»*`, string` *«jméno_položky»*`)`

Funkce vrací číslo položky, jejíž jméno jsme zadali pomocí parametru *«jméno_položky»*.

Pg_FieldPrtLen Funkce pro práci s databází PostgreSQL

Zjištění délky položky ve znacích

§ `integer Pg_FieldPrtLen(integer` *«výsledek»*`, integer` *«číslo_záznamu»*`,`
`mixed` *«položka»*`)`

Funkce vrací počet znaků potřebných pro vytištění obsahu dané položky.

Pg_FieldSize Funkce pro práci s databází PostgreSQL

Zjištění velikosti místa potřebného pro uložení položky

§ `integer Pg_FieldSize(integer` *«výsledek»*`, integer` *«číslo_položky»*`)`

Funkce zjistí počet bajtů, který je využíván pro uložení dané položky výsledku.

Pg_FieldType Funkce pro práci s databází PostgreSQL

Zjištění typu položky

§ `string Pg_FieldType(integer` *«výsledek»*`, integer` *«číslo_položky»*`)`

Funkce vrací typ položky. Položku určíme pomocí jejího čísla (začínají od nuly) a výsledku, ve kterém je obsažena.

Pg_FreeResult

Funkce pro práci s databází PostgreSQL

Uvolnění výsledku z paměti

§ `integer Pg_FreeResult(integer` *«výsledek»*`)`

Funkce uvolní daný *«výsledek»* z paměti. Funkce vrací hodnotu `true`, pokud *«výsledek»* existuje. Pokud *«výsledek»* neexistuje, není co uvolňovat a funkce vrací `false`. Funkci ve většině skriptů není třeba volat, protože se paměť automaticky uvolní po skončení běhu skriptu. Volání má smysl pouze v dlouhých skriptech, kdy chceme minimalizovat paměťové nároky.

Pg_GetLastOID

Funkce pro práci s databází PostgreSQL

Zjištění OID posledně vloženého záznamu

§ `integer Pg_GetLastOID(integer` *«výsledek»*`)`

Funkce slouží k získání OID posledního záznamu vloženého do tabulku pomocí SQL-příkazu `INSERT`. *«Výsledek»* proto musí obsahovat právě výsledek příkazu `INSERT`. OID je jedinečné číslo, které je automaticky přiřazeno každému novému záznamu tabulky v PostgreSQL.

Pg_Host

Funkce pro práci s databází PostgreSQL

Zjistí počítač, na kterém běží PostgreSQL

§ `string Pg_Host(integer` *«spojení»*`)`

Funkce vrací jméno počítače, ke kterému je vytvořeno *«spojení»*. Pokud se funkci nepodaří provést, je vráceno `false`. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

Pg_LOClose

Funkce pro práci s databází PostgreSQL

Zavření velkého objektu

§ `integer Pg_LOClose(integer` *«fd»*`)`

Funkce zavře přístup k velkému objektu, který je identifikován pomocí *«fd»*.

`Pg_LOCreate()` na straně 393 a `Pg_LOOpen()` na straně 393.

Pg_LOCreate

Funkce pro práci s databází PostgreSQL

Vytvoření velkého objektu

§ `integer Pg_LOCreate(integer «spojení»)`

Funkce vytvoří nový velký objekt a vrací jeho OID. Objekt je vytvořen v databázi, ke které je vytvořeno *«spojení»*. Objekt je vytvořen s právy pro čtení i zápis. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

Všechny operace prováděné s velkými objekty by měly být součástí jedné transakce. Měly by tedy začínat příkazem `BEGIN` a končit příkazy `COMMIT` a `END`.

```
$spojeni = pg_Connect ("localhost", "", "", "", "imagedb");
Pg_Exec($spojeni, "BEGIN");
    $oid = Pg_LOCreate($spojeni);
    $loh = Pg_LOOpen($spojeni, $oid, "w");
    Pg_LOWrite($loh, "«nějaká skutečně velká data»");
    Pg_LOClose($loh);
Pg_Exec($spojeni, "COMMIT");
Pg_Exec($spojeni, "END");
```

`Pg_LOUnLink()` na straně 394.

Pg_LOOpen

Funkce pro práci s databází PostgreSQL

Otevření velkého objektu

§ `integer Pg_LOOpen(integer «spojení», int «OID», string «režim»)`

Funkce otevře velký objekt a vrací identifikátor *«fd»* pro další práci s tímto objektem. Objekt, který chceme otevřít, musíme určit pomocí jeho jedinečného identifikátoru *«OID»*. *«Režim»* určuje způsob otevření objektu: `r` (pouze pro čtení), `w` (pouze pro zápis) nebo `rw` (čtení/zápis). Pokud funkci zavoláme bez parametru *«spojení»*, použije se poslední otevřené spojení.

`Pg_LOClose()` na straně 392.

Pg_LORead Funkce pro práci s databází PostgreSQL

Čtení dat z velkého objektu

§ `string Pg_LORead(integer` *«fd»*, `integer` *«délka»*)

Funkce přečte a vrátí maximálně *«délka»* bajtů z velkého objektu *«fd»*. V případě chyby vrací funkce `false`.

`Pg_LOReadAll()` na straně 394 a `Pg_LOWrite()` na straně 394.

Pg_LOReadAll Funkce pro práci s databází PostgreSQL

Přečtení všech dat z velkého objektu

§ `integer Pg_LOReadAll(integer` *«fd»*)

Funkce vypíše celý obsah velkého objektu *«fd»* na výstup skriptu. Funkce vrací počet bajtů takto zapsaných na výstup. V případě chyby vrací funkce `false`. Funkce se hodí zejména pro generování binárních dat z databáze — např. obrázků.

`Pg_LORead()` na straně 394.

Pg_LOUnLink Funkce pro práci s databází PostgreSQL

Smazání velkého objektu

§ `integer Pg_LOUnLink(integer` *«spojení»*, `integer` *«OID»*)

Funkce smaže objekt identifikovaný svým číslem *«OID»*. Parametr *«spojení»* můžeme vynechat, použije se poslední vytvořené spojení. Funkce vrací `true`, pokud se objekt podaří smazat. V opačném případě vrací `false`.

`Pg_LOCreate()` na straně 393.

Pg_LOWrite Funkce pro práci s databází PostgreSQL

Zápis dat do velkého objektu

§ `integer Pg_LOWrite(integer` *«fd»*, `string` *«data»*)

Funkce do objektu *«fd»* zapíše *«data»*. Funkce vrací počet skutečně zapsaných bajtů. V případě chyby vrací funkce `false`.

`Pg_LORead()` na straně 394 a `Pg_LOReadAll()` na straně 394.

Pg_NumFields

Funkce pro práci s databází PostgreSQL

Zjistí počet položek výsledku

§ `integer Pg_NumFields(integer` *«výsledek»*`)`

Funkce vrací položek obsažených ve *«výsledku»*. V případě chyby vrací funkce `false`.

`Pg_NumRows()` na straně 395.

Pg_NumRows

Funkce pro práci s databází PostgreSQL

Zjistí počet záznamů výsledku

§ `integer Pg_NumRows(integer` *«výsledek»*`)`

Funkce vrací počet záznamů ve *«výsledku»*. V případě chyby vrací funkce `false`.

`Pg_CmdTuples()` na straně 388.

Ukázky použití naleznete na straně 152.

Pg_Options

Funkce pro práci s databází PostgreSQL

Funkce vrací volby nastavené pro spojení

§ `string Pg_Options(integer` *«spojení»*`)`

Funkce vrací volby nastavené pro *«spojení»*. Pokud parametr *«spojení»* nepoužijeme, použije se poslední vytvořené spojení.

Pg_PConnect

Funkce pro práci s databází PostgreSQL

Vytvoření persistentního spojení s databází

§ `integer Pg_Connect(string` *«počítač»*`, string` *«port»*`, string` *«volby»*`,`
`string` *«tty»*`, string` *«databáze»*`)`

Funkce vytvoří persistentní spojení s databází. Funkce se chová stejně jako `Pg_Connect()` s jedinou výjimkou — pokud v dalším skriptu zavoláme funkci `Pg_PConnect()` se stejnými parametry, je použito již existující spojení s databází a nemusí se znovu zdlouhavě navazovat.

Persistentní spojení pracují pouze tehdy, pokud PHP běží jako modul serveru. Pokud funkci využijeme v PHP, které je spuštěno jako CGI--skript, bude se chovat stejně jako `Pg_Connect()`.

`Pg_Connect()` na straně 388.

Ukázky použití naleznete na straně 177.

Pg_Port — Funkce pro práci s databází PostgreSQL

Zjistí port, na kterém běží PostgreSQL

§ `integer Pg_Port(integer «spojení»)`

Funkce vrací číslo portu, ke kterému je vytvořeno *«spojení»*. Pokud se funkci nepodaří provést, je vráceno `false`. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

Pg_Result — Funkce pro práci s databází PostgreSQL

Přečtení jedné položky výsledku

§ `mixed Pg_Result(integer «výsledek», integer «číslo_záznamu»,`
`mixed «položka»)`

Funkce vrací obsah položky *«výsledku»*. Položka je určena číslem záznamu (začínají od nuly) a číslem (opět číslováno od nuly) nebo jménem položky. V případě chyby vrací funkce `false`.

Používání této funkce, zejména na větší výsledky, je poměrně pomalé. Mnohem rychlejší je využití funkcí `Pg_Fetch_Row()`, `Pg_Fetch_Array()` a `Pg_Fetch_Object()`, které načítají celý záznam najednou.

`Pg_Fetch_Row()` na straně 390, `Pg_Fetch_Array()` na straně 389 a `Pg_Fetch_Object()` na straně 390.

Pg_tty Funkce pro práci s databází PostgreSQL

Zjistí jméno zařízení tty

§ `string Pg_tty(integer «spojení»)`

Funkce vrací jméno zařízení tty, na které jsou vypisovány ladící informace. Pokud se funkci nepodaří provést, je vráceno `false`. Pokud funkci zavoláme bez parametru, použije se poslední otevřené spojení.

PHP_OS Konstanta

Konstanta obsahuje jméno operačního systému, na kterém PHP právě běží

PHP_VERSION Konstanta

Konstanta obsahuje číslo verze právě používaného systému PHP

PHPInfo Konfigurace a informace o PHP

Zobrazení komplexních informací o PHP

§ `integer PHPInfo(void)`

Funkce vypíše ve formátu HTML mnoho zajímavých informací o právě spuštěné verzi PHP. Funkce vždy vrací hodnotu `true`.

Ukázky použití naleznete na stranách 28, 472.

PHPVersion Konfigurace a informace o PHP

Zjištění verze PHP

§ `string PHPVersion(void)`

Funkce vrací označení aktuální verze systému PHP. V závislosti na tom, kterou verzi používáme, dostaneme např. řetězec `3.0.3` nebo `3.1 alpha 0`.

Pi

Matematická funkce

Vrací hodnotu Ludolfova čísla π

§ `double Pi(void)`

Vrací přibližnou hodnotu Ludolfova čísla π ($\pi \doteq 3{,}141\,592\,653\,589\,8$).

POpen

Funkce pro práci se soubory

Otevře rouru k nově spuštěnému procesu

§ `integer POpen(string «příkaz», string «mód»)`

Funkce spustí *«příkaz»* a vrací ukazatel *«fp»* na jeho vstup/výstup. Pokud jako *«mód»* použijeme `r`, ukazuje *«fp»* na výstup skriptu. Pokud použijeme *«mód»* `w`, ukazuje *«fp»* na vstup skriptu. Se získaným ukazatelem *«fp»* můžeme pracovat obdobným způsobem jako s ukazatelem získaným pomocí funkce `FOpen()`. Ukazatel musíme zavřít pomocí funkce `PClose()`.

Pokud při volání funkce došlo k chybě, je vrácena hodnota `false`.

Pokud chceme zpracovávat výstup příkazu `ls`, můžeme si jej zpřístupnit následujícím způsobem:

```
$fp = POpen("/bin/ls", "r");
```

`PClose()` na straně 387 a `FOpen()` na straně 294.

Pos

Funkce pro práci s poli

Vrací hodnotu aktuálního prvku pole

§ `mixed Pos(array «pole»)`

Funkce vrací prvek *«pole»*, na který ukazuje interní ukazatel *«pole»*. Jedná se o synonymum k funkci `Current()`.

`Each()` na straně 282, `Prev()` na straně 399, `Next()` na straně 372, `Reset()` na straně 405, `End()` na straně 284 a `Array_Walk()` na straně 259.

Pow

Matematická funkce

Výpočet mocnin

§ `double Pow(double «základ», double «exponent»)`

Umocní *«základ»* na *«exponent»*, tj. vypočte hodnotu *«základ»*$^{«exponent»}$.

`Exp()` na straně 288, `Log()` na straně 355 a `Log10()` na straně 355.

Prev

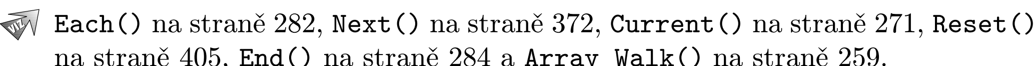
Funkce pro práci s poli

Vrací hodnotu předchozího prvku pole

§ `mixed Prev(array «pole»)`

Funkce posune ukazatel *«pole»* na předcházející prvek a vrátí jeho obsah. Funkce vrací `false`, pokud dosáhne před začátek pole.

> Funkce vrací `false`, pokud i prvek pole obsahuje `false`. To může znemožnit průchod celého pole příkazem typu `while (Prev(...)) ....` Pro bezpečný průchod celým polem bychom měli používat funkci `Each()`.



`Each()` na straně 282, `Next()` na straně 372, `Current()` na straně 271, `Reset()` na straně 405, `End()` na straně 284 a `Array_Walk()` na straně 259.

Print

Vytiskne obsah řetězce

§ `Print «řetězec»`

Vytiskne obsah *«řetězce»*. Podobně jako `Echo` i `Print` je příkaz PHP. Na rozdíl od `Echo` může mít jen jeden parametr a vrací hodnotu `true`.

`Echo()` na straně 283, `Printf()` na straně 400 a `Flush()` na straně 294.

Printf

Funkce pro práci s textovými řetězci

Vytiskne zformátovaný řetězec

§ `integer Printf(string` *«formát»*`, mixed` *«výraz1»*`, mixed` *«výraz2»*`, ...)`

Vytiskne *«výraz1»* až *«výrazN»* zformátované podle *«formátu»*. Podívejme se nyní na to, jak vypadá popis formátu pro funkci `Printf()`.

Formátovací řetězec může obsahovat libovolný znak a ten se stane součástí zformátovaného výstupu. Výjimku tvoří znak '%', který slouží právě k určení způsobu formátování jednotlivých výrazů. Pro každý výraz tedy musí *«formát»* obsahovat jeden znak '%' následovaný specifikací zobrazení.

Nejjednodušší tvar určení formátu je řetězec ve tvaru `"%`*«typ»*`"`. *«typ»* je písmeno, které určuje, jakého typu je odpovídající argument a jak má být zobrazen. Přehled všech typů nalezneme v tabulce 9-6.

Znak	Popis
`%`	Vytiskne obyčejný znak '%'.
`b`	Parametr je typu `integer` a bude zobrazen ve dvojkové soustavě.
`c`	Parametr je typu `integer` a bude zobrazen jako písmeno s odpovídajícím ASCII-kódem.
`d`	Parametr je typu `integer` a bude zobrazen jako celé číslo v desítkové soustavě.
`f`	Parametr je typu `double` a bude také tak zobrazen.
`o`	Parametr je typu `integer` a bude zobrazen v osmičkové soustavě.
`s`	Parametr je typu `string` a bude zobrazen jako textový řetězec.
`x`	Parametr je typu `integer` a bude zobrazen v šestnáctkové soustavě.
`X`	Parametr je typu `integer` a bude zobrazen v šestnáctkové soustavě (šestnáctkové číslice budou zobrazeny jako velká písmena).

Tab. 9-6: Typy parametrů použitelné ve formátovacím řetězci funkce Printf()

Formátování jednotlivých parametrů však můžeme ovládat mnohem lépe. Obecně může mít formátovací řetězec pro jeden parametr následující tvar:

%*«výplň»«zarovnání»«délka»«desetinná místa»«typ»*

Všechny specifikace kromě *«typu»* jsou přitom nepovinné.

«výplň» určuje znak, který bude použit na vyplnění prázdného místa tak, aby byl parametr zobrazen na požadovaném počtu znaků. Standardně je výplňový znak mezera. Pokud chceme jako výplňový znak použít nulu, použijeme jako *«výplň»* právě znak `0`. Pokud chceme vyplnit prázdné místo jiným znakem, uvedeme jej za apostrof '`'`'.

Standardně je obsah parametru zarovnán v pravo. Pokud jej chceme zarovnat vlevo, použijeme jako *«zarovnání»* znak minus '`-`'.

«délka» je číslo, které určuje minimální počet znaků, které odpovídající parametr zabere po zformátování.

Pokud je parametrem výraz typu `double` (formátovací typ `f`), můžeme určit počet desetinných míst, která se budou tisknout. *«desetinná místa»* se zapisují jako tečka následovaná počtem požadovaných desetinných míst (např. `.2` pro dvě desetinná místa). Počet desetinných míst odpovídajícím způsobem zvětší *«délku»* zformátovaného řetězce.

Několik ukázek použití funkce `Printf()`. Jako komentář je uveden výsledek.

```
Printf("%10f\n", Pi());                 //         3.141593
Printf("%-10f\n", Pi());                // 3.141593
Printf("%'$10f\n", Pi());               // $$$$$$$$$3.141593
Printf("%010f\n", Pi());                // 0000000003.141593
Printf("%10.2f\n", Pi());               //         3.14
Printf("%-10.20f\n", Pi());             // 3.14159265358979311600
Printf("%f\n", 65505);                  // 65505.000000
Printf("%b\n", 65505);                  // 1111111111100001
Printf("%d\n", 65505);                  // 65505
Printf("%o\n", 65505);                  // 177741
Printf("%x\n", 65505);                  // ffe1
Printf("%X\n", 65505);                  // FFE1
Printf("Znak s kódem 65: %c\n", 65);    // Znak s kódem 65: A
Printf("0x%X=%d\n", 250975, 250975);    // 0x3D45F=250975
Printf("%'--20s\n", "Printf() je boží");// Printf() je boží----
```

`SPrintf()` na straně 412, `Echo()` na straně 283 a `Print()` na straně 399.

PutEnv

Konfigurace a informace o PHP

Nastavení proměnné prostředí

§ `void PutEnv(string «nastavení»)`

Funkce slouží k nastavení proměnné prostředí. Samotné nastavení má tvar *«proměnná»*=*«hodnota»* např. `TEMP=/tmp`. Proměnná je nastavena pouze v prostředí PHP. I to se však může hodit — např. v případech, kdy z PHP voláme externí program, který ke své správné činnosti potřebuje nastavit některé proměnné prostředí.

`GetEnv()` na straně 302, `Exec()` na straně 288, `System()` na straně 423 a `PassThru()` na straně 387.

QuoteMeta

Funkce pro práci s textovými řetězci

Nahradí metaznaky escape sekvencí

§ `string QuoteMeta(string «řetězec»)`

V *«řetězci»* nahradí všechny metaznaky '`.\+*?[]()^$`' příslušnou escape sekvencí (tj. předřadí před metaznak zpětné lomítko). Funkce se hodí zejména pokud chceme nějaký řetězec použít uvnitř regulárního výrazu a nechceme nechtěně interpretovat metaznaky.

Rad2Deg

Matematická funkce

Převod radiánů na stupně

§ `double Rad2Deg(double «úhel»)`

Funkce převede *«úhel»* v radiánech na stupně.

`Deg2Rad()` na straně 280.

Rand

Matematická funkce

Generování náhodné hodnoty

§ `integer Rand(integer «min», integer «max»)`

Bez parametrů vrací funkce pseudonáhodnou hodnotu od 0 do `RAND_MAX`. Pokud chceme získat hodnotu z intervalu $\langle a, b\rangle$, použijeme přepočet `Rand()%($b-$a+1)+$a`. Před použitím funkce `Rand()` bychom měli generátor náhodných čísel zinicializovat pomocí funkce `SRand()`.

Od verze 3.0.4 můžeme u funkce použít dva parametry *«min»* a *«max»*, které určují rozsah generovaných čísel.

`SRand()` na straně 413 a `GetRandMax()` na straně 305.

RawURLDecode

Funkce pro práci s textovými řetězci

Rozkóduje řetězec zakódovaný jako URL

§ `string RawURLDecode(string` *«řetězec»*`)`

Rozkóduje *«řetězec»* tak, jako by se jednalo o text zakódovaný podle pravidel pro URL. To znamená, že všechny výskyty znaku '`%`' následovaného dvěma šestnáctkovými číslicemi budou nahrazeny znakem s odpovídajícím ASCII-kódem.

Tak například příkaz `echo RawURLDecode("Jan%20Nov%E1k%40mail.cz")` vypíše text `Jan Novák@mail.cz`.

`RawURLEncode()` na straně 403.

RawURLEncode

Funkce pro práci s textovými řetězci

Zakóduje řetězec tak, aby byl použitelný v URL

§ `string RawURLEncode(string` *«řetězec»*`)`

Vrací řetězec, ve kterém jsou všechny nealfanumerické znaky s výjimkou znaků '`.`', '`-`' a '`_`' nahrazeny znakem procenta '`%`' následovaným dvěma šestnáctkovými číslicemi. Šestnáctkové číslo přitom vyjadřuje kód zakódovaného znaku. Způsob kódování je definován v RFC 1738.

Funkce nám umožňuje snadno předávat parametry pomocí URL:

```
echo '<A HREF="search.php?query=', RawURLEncode("Jan Novák"), '">Hledej</A>';
```

`RawURLDecode()` na straně 403.

ReadDir

Funkce pro práci s adresáři

Funkce přečte název jednoho souboru z adresáře

§ `string ReadDir(integer «dp»)`

Funkce vrací název dalšího souboru v adresáři *«dp»*. Pokud již z adresáře byly názvy všech souborů přečteny, vrací funkce `false`.

`OpenDir()` na straně 384, `CloseDir()` na straně 268 a `RewindDir()` na straně 406.

ReadFile

Funkce pro práci se soubory

Výpis souboru na standardní výstup

§ `integer ReadFile(string «soubor», integer «použít_include_path»)`

Funkce obsah *«souboru»* vypíše na standardní výstup. Pokud *«soubor»* začíná na `http://`, vypíše se výsledek požadavku na dané URL vznesený pomocí protokolu HTTP/1.0. Obdobně, pokud *«soubor»* začíná na `ftp://`, otevře se spojení s FTP-serverem a požadovaný soubor je přenesen a vypsán na standardní výstup.
Pokud jako hodnotu posledního nepovinného parametru *«použít_include_path»* uvedeme `true`, bude se soubor pro otevření hledat i v adresářích uvedených v direktivě `include_path` v konfiguračním souboru `php3.ini`.

`FPassThru()` na straně 295, `File()` na straně 290 a `FOpen()` na straně 294.

ReadLink

Funkce pro práci se soubory

Zjištění, kam ukazuje symbolický odkaz

§ `string ReadLink(string «odkaz»)`

Funkce vrací reálné jméno souboru (včetně cesty), na který ukazuje symbolický *«odkaz»*. Pokud při provádění funkce došlo k chybě, vrací funkce hodnotu `false`.

`SymLink()` na straně 423, `Link()` na straně 354 a `LinkInfo()` na straně 354.

Register_ShutDown_Function

Konfigurace a informace o PHP

Zaregistrování funkce, která se zavolá při skončení skriptu

§ `void Register_ShutDown_Function(fptr «funkce»)`

«Funkce» bude zavolána po ukončení skriptu. Je jedno, zda je skript ukončen regulárně nebo předčasně pomocí příkazu `exit` nebo `return`.

Rename

Funkce pro práci se soubory

Přejmenování souboru

§ `integer Rename(string «staré_jméno», string «nové_jméno»)`

Funkce přejmenuje soubor *«staré_jméno»* na *«nové_jméno»*. Po úspěšném přejmenování vrací funkce hodnotu `true`, v opačném případě `false`.

Reset

Funkce pro práci s poli

Nastaví ukazatel na začátek pole

§ `void Reset(array «pole»)`

Nastaví ukazatel *«pole»* na jeho první prvek.

`Current()` na straně 271, `Prev()` na straně 399, `Next()` na straně 372, `End()` na straně 284, `Each()` na straně 282 a `Array_Walk()` na straně 259.

Ukázky použití naleznete na straně 44.

Rewind

Funkce pro práci se soubory

Nastavení aktuální pozice souboru na jeho začátek

§ `integer Rewind(integer «fp»)`

Aktuální pozici souboru *«fp»* nastaví na jeho začátek. Pokud vše proběhlo úspěšně, vrací funkce `true`. V opačném případě vrací funkce `false`.

Volání `Rewind($fp)` je shodné s voláním `!FSeek($fp,0)`.

`FSeek()` na straně 296 a `FTell()` na straně 298.

Ukázky použití naleznete na straně 181.

RewindDir

Funkce pro práci s adresáři

Přesun na první položku otevřeného adresáře

§ `void RewindDir(integer «dp»)`

Po zavolání funkce `RewindDir()` budou položky adresáře *«dp»* čteny znovu od začátku.

`OpenDir()` na straně 384, `CloseDir()` na straně 268 `ReadDir()` na straně 404.

RmDir

Funkce pro práci se soubory

Odstranění adresáře

§ `integer RmDir(string «adresář»)`

Odstraní *«adresář»*. Funkce vrací `true`, pokud se podařilo adresář odstranit. V opačném případě vrací funkce `false`.

`MkDir()` na straně 358.

Round

Matematická funkce

Zaokrouhlení desetinného čísla

§ `integer Round(double «výraz»)`

Zaokrouhlí *«výraz»*.

`Ceil()` na straně 264 a `Floor()` na straně 294.

RSort

Funkce pro práci s poli

Sestupně setřídí pole

§ `void RSort(array «pole»)`

Funkce sestupně setřídí prvky *«pole»*. Při třídění nejsou zachovány indexy jednotlivých prvků. Funkce se proto ve většině případů nehodí pro třídění asociativních polí — pro ně použijeme funkci `ARSort()`.

`ARSort()` na straně 259, `ASort()` na straně 260, `Sort()` na straně 411 a `KSort()` na straně 344.

RTrim

Funkce pro práci s textovými řetězci

Odstraní mezery z konce řetězce

§ `string RTrim(string «řetězec»)`

Funkce odstraní z konce řetězce všechny netisknutelné znaky — mezery, tabulátory a konce řádků. Je synonymem funkce `Chop()`.

`Trim()` na straně 425, `LTrim()` na straně 356 a `Chop()` na straně 266.

SetCookie

Funkce pro práci s protokolem HTTP

Zaslání cookie klientovi

§
```
integer SetCookie(string «jméno», string «hodnota»,
          integer «platnost», string «cesta»,
          string «doména», integer «zabezpečení»)
```

Funkce zašle cookie klientovi. Všechny parametry kromě *«jména»* jsou nepovinné. Pokud zašleme samotné *«jméno»*, cookie se na klientovi smaže.

«platnost» cookie se zadává jako čas určený počtem sekund od 1. ledna 1970, s výhodou proto můžeme pro její nastavení použít funkce `Time()` a `MkTime()`.

«cesta» a *«doména»* umožňují přesněji omezit, pro které servery v doméně a pro které dokumenty na serveru bude cookie platná. Pokud *«zabezpečení»* nastavíme na `true`, bude se cookie přenášet pouze v případě, že spojení je zajištěno vrstvou SSL (Secure Socket Layer).

Funkci `SetCookie()` musíme podobně jako `Header()` používat na začátku skriptu, před výstupy pro prohlížeč.

Data jsou před odesláním automaticky převedena do požadovaného formátu (URL kódování textů, standardizovaný formát času...).

Pro odeslání cookie, která bude mít platnost jednu hodinu, poslouží následující kód:

```
SetCookie("SessionID", UniqID("sid"), Time()+3600);
```

Při obsluze dalších požadavků máme obsah cookie automaticky přístupný v proměnné `$SessionID`.

Ukázky použití naleznete na straně 446.

SetLocale

Funkce pro práci s textovými řetězci

Nastavení podpory národních prostředí

§ `string SetLocale(string «kategorie», string «jazyk»)`

Funkce slouží k nastavení národního prostředí. Nastavovat můžeme různé *«kategorie»* národní podpory (viz tabulka 9-7). *«Jazyk»* se udává svým názvem — např. `Czech` pro češtinu, `English` pro angličtinu a `Slovak` pro slovenštinu. Pokud jako jazyk uvedeme hodnotu 0, funkce vrátí aktuální nastavení.

Kategorie	Popis
LC_ALL	Nastaví všechny kategorie najednou.
LC_COLLATE	Nastaví způsob porovnávání řetězců — zatím není v PHP implementováno.
LC_CTYPE	Pro konverzi a klasifikaci znaků (např. převody mezi malými a velkými písmeny).
LC_MONETARY	Měnové otázky — dosud neimplementováno.
LC_NUMERIC	Oddělovač celé a desetinné části čísla.
LC_TIME	Formát data — dosud neimplementováno.

Tab. 9-7: Kategorie, pro které můžeme nastavit národní zvyklosti

Pokud jako *«jazyk»* použijeme prázdný řetězec, převezme se nastavení z odpovídajících proměnných prostředí, případně z proměnné `LANG`.

Pokud se úspěšně podaří kategorii změnit, vrací funkce novou kategorii. V opačném případě vrací hodnotu `false`.

Set_Socket_Blocking

Síťové funkce

Nastavení blokujícího/neblokujícího režimu pro socket

§ `integer Set_Socket_Blocking (integer «fp», integer «mód»)`

Pokud jako hodnotu parametru *«mód»* použijeme `false`, bude socket *«fp»* přepnut do neblokujícího režimu. V tomto režimu jsou všechny funkce pro čtení ze socketu (např. `FGetS()`) ihned provedeny a nečeká se až přijdou nějaká data. Standardně se pracuje se sockety v blokujícím režimu, kdy funkce čeká do té doby, než po síti přijdou očekávaná data.

Funkce vrací `true`, pokud se podařilo provést změnu režimu čtení ze socketu. V opačném případě vrací `false`.

`FSockOpen()` na straně 297.

Set_Time_Limit

Konfigurace a informace o PHP

Nastaví maximální dobu provádění skriptu

§ `void Set_Time_Limit(integer` *«sekundy»*`)`

Funkce nastaví limit, do kterého musí být provádění skriptu ukončeno. Pokud skript do této doby sám neskončí, je násilně přerušen. Časový limit se počítá od provedení příkazu. Standardní hodnota je 30 sekund a lze ji nastavit v konfiguračním souboru. Pokud jako limit zadáme 0 sekund, skript může běžet libovolně dlouhou dobu.

Ukázky použití naleznete na straně 215.

SetType

Proměnná

Nastavení typu proměnné

§ `integer SetType(mixed` *«proměnná»*`, string` *«typ»*`)`

Nastaví typ *«proměnné»* na *«typ»*. Použitelné typy jsou `integer`, `double`, `string`, `array` a `object`. Funkce vrací `true`, pokud se změnu typu podařilo provést, jinak vrací `false`.

`GetType()` na straně 306.

Ukázky použití naleznete na straně 46.

Sin

Matematická funkce

Sinus

§ `double Sin(double` *«výraz»*`)`

Vrátí hodnotu funkce sinus ze zadaného *«výrazu»*, který je v radiánech.

`Cos()` na straně 270 a `Tan()` na straně 424.

SizeOf
Funkce pro práci s poli

Zjištění počtu prvků proměnné

§ `integer SizeOf(mixed «proměnná»)`

Funkce vrátí počet prvků uložených v *«proměnné»*. Obvykle je *«proměnná»* typu pole. Pokud použijeme proměnnou skalárního typu, vrací funkce hodnotu 1. Pokud *«proměnná»* nemá hodnotu, vrací funkce nulu. Funkce je synonymem k funkci `Count()`.

`Count()` na straně 271, `IsSet()` na straně 341 a `Is_Array()` na straně 337.

Sleep
Ostatní funkce

Pozastavení skriptu

§ `void Sleep(integer «sekundy»)`

Funkce způsobí zastavení provádění skriptu na daný počet *«sekund»*.

`uSleep()` na straně 428.

SNMPGet
Funkce pro práci s protokolem SNMP

Získání hodnoty jednoho SNMP objektu

§ `string SNMPGet(string «adresa», string «komunita», string «id»)`

Funkce vrací hodnotu daného objektu, v případě chyby vrací `false`. Funkce čte hodnotu objektu určeného pomocí *«id»* z SNMP agenta, který je na dané *«adrese»*. Při čtení objektu je agentovi předán parametr *«komunita»*, který slouží k určení oprávnění pro přístup.

Následující kód vypíše kontakt na správce zařízení, které má IP-adresu 146.17.13.2:

```
echo SNMPGet("146.17.13.2:, "public", "system.sysContact.0");
```

SNMPWalk

Funkce pro práci s protokolem SNMP

Přečtení všech SNMP objektů od dané úrovně

§ `array SNMPWalk(string «adresa», string «komunita», string «id»)`

Funkce vrací pole, které obsahuje názvy všech objektů, jež agent běžící na *«adrese»* obsahuje pod objektem *«id»*. Pokud jako *«id»* zadáme prázdný řetězec, čtou se všechny objekty. K agentovi je přistupováno pomocí zadaného jména *«komunita»*.

Následující příklad vypíše všechny objekty dostupné všem z lokálně běžícího SNMP agenta:

```
$x = SNMPWalk("127.0.0.1", "public", "");
for ($i=0; $i<Count($x); $i++)
    echo $x[$i]."<BR>\n";
```

Sort

Funkce pro práci s poli

Setřídí pole

§ `void Sort(array «pole»)`

Funkce setřídí prvky *«pole»*. Při třídění nejsou zachovány indexy jednotlivých prvků. Funkce se proto ve většině případů nehodí pro třídění asociativních polí — pro ně použijeme funkci `ASort()`.

`ARSort()` na straně 259, `ASort()` na straně 260, `RSort()` na straně 406 a `KSort()` na straně 344.

Soundex

Funkce pro práci s textovými řetězci

Vrátí hodnotu klíče Soundex pro zadaný text

§ `string Soundex(string «řetězec»)`

Soundex je funkce, která podobně znějícím slovům přiřazuje stejný klíč. Často se využívá při prohledávání databází, kdy neznáme přesný pravopis daného slova. Klíč je třímístné číslo, kterému předchází jedno písmeno. Tato funkce bohužel správně funguje pouze pro angličtinu. Pro češtinu nejsou výsledky nijak závratné.

V jistém ohledu nám následující program potvrdí, že Gauss toho má mnoho společného s husou:[2]

```
echo Soundex("Gauss"); // Vypíše G200
echo Soundex("goose"); // Vypíše G200
```

Split

Funkce pro práci s regulárními výrazy

Rozdělí řetězec na části a uloží je do pole

§ `array Split(string` *«regexp»*`, string` *«řetězec»*`, integer` *«limit»*`)`

Funkce rozloží *«řetězec»* do pole po částech, které jsou odděleny textem, který vyhovuje regulárnímu výrazu *«regexp»*. Nepovinný parametr *«limit»* určuje maximální počet částí, na které bude řetězec rozložen. Pokud je *«limit»* menší než počet částí v řetězci, obsahuje poslední prvek pole celou zbývající část řetězce.

Text věty můžeme snadno rozdělit na jednotlivá slova:

```
$slova = Split("[^[:alnum:]]+", $veta);
```

Slova jsou v našem případě oddělena vším, co není písmeno nebo číslo (alfanumerický znak).

Pro většinu aplikací vystačíme s jednodušším oddělovačem, a tedy i s funkcí `Explode()` popsanou na straně 289, která je mnohem rychlejší.

SPrintf

Funkce pro práci s textovými řetězci

Uloží zformátovaný řetězec do proměnné

§ `string SPrintf(string` *«formát»*`, mixed` *«výraz1»*`, mixed` *«výraz2»*`,`
`...)`

Funkce se chová stejně jako funkce `Printf()`. Rozdíl je v tom, že zformátovaný řetězec není vytisknut, ale vrácen jako hodnota funkce.

`Printf()` na straně 400.

[2] Za předpokladu, že „mnoho“ definujeme jako hodnotu funkce `Soundex()`.

SQL_RegCase

Funkce pro práci s regulárními výrazy

Vytvoří regulární výraz pro hledání řetězce bez závislosti na velikosti písmen

§ `string SQL_RegCase(string «řetězec»)`

Funkce vrátí regulární výraz, který odpovídá *«řetězci»*. Vrácený regulární výraz však ignoruje velikost písmen. Výsledkem volání `SQL_RegCase("Karel Čapek")` bude řetězec `[Kk][Aa][Rr][Ee][Ll][ ][Čč][Aa][Pp][Ee][Kk]`. Funkci použijeme zejména při zadávání podmínek v jazyce SQL pro ty servery, které nepodporují hledání nezávislé na velikosti písmen.

Sqrt

Matematická funkce

Druhá odmocnina

§ `double Sqrt(double «výraz»)`

Vrátí druhou odmocninu z *«výrazu»*.

SRand

Matematická funkce

Inicializace náhodného generátoru

§ `void SRand(integer «seed»)`

Inicializuje generátor náhodných čísel hodnotou *«seed»*. Jako tuto hodnotu je vhodné zvolit pokaždé jiné číslo.

Hodnotu *«seed»* můžeme odvodit např. od aktuálního času. Můžeme využít toho, že funkce `MicroTime()` vrací řetězec, na jehož začátku je počet mikrosekund, které uběhly od začátku aktuální sekundy. Inicializace generátoru náhodných čísel může vypadat třeba takto:

```
SRand((double)MicroTime()*1e6)
```

`Rand()` na straně 402 a `GetRandMax()` na straně 305.

Ukázky použití naleznete na straně 204.

Stat

Funkce pro práci se soubory

Zjištění informací o souboru

§ `array Stat(string «soubor»)`

Funkce zjistí informace o souboru. Informace jsou vráceny v poli, kde jsou postupně uloženy následující údaje: zařízení, číslo i-node, počet odkazů, UID vlastníka, GID vlastníka, typ zařízení, velikost v bajtech, čas posledního přístupu, čas poslední změny, čas vytvoření, velikost bloku a počet alokovaných bloků. Ve Windows většina těchto údajů obsahuje hodnotu −1, protože funkce vrací některé parametry, které úzce souvisí se souborovými systémy používanými v Unixu.

`FileATime()` na straně 291, `FileCTime()` na straně 291, `FileMTime()` na straně 292, `FileOwner()` na straně 292, `FileGroup()` na straně 292, `FilePerms()` na straně 293, `FileSize()` na straně 293 a `FileType()` na straně 293.

StrCaseCmp

Funkce pro práci s textovými řetězci

Porovnání řetězců bez ohledu na malá a velká písmena

§ `integer StrCaseCmp(string «řetězec1», string «řetězec2»)`

Funkce vrací zápornou hodnotu, pokud je *«řetězec1»* menší než *«řetězec2»*. Funkce vrací kladnou hodnotu, pokud je *«řetězec1»* větší než *«řetězec2»*. Pokud se řetězce shodují, vrací funkce 0. Porovnání není citlivé na velikost písmen.

StrChr

Funkce pro práci s textovými řetězci

Nalezení prvního výskytu znaku v řetězci

§ `string StrChr(string «řetězec», string «znak»)`

Funkce nalezne v *«řetězci»* první výskyt *«znaku»* a vrátí obsah *«řetězce»* až od tohoto *«znaku»* včetně.

Malý příklad:

```
echo StrChr("abeceda", "e") // Vytiskne eceda
```

Funkce je synonymem k funkci `StrStr()` na straně 419.

StrCmp

Funkce pro práci s textovými řetězci

Porovnání řetězců

§ `integer StrCmp(string` *«řetězec1»*`, string` *«řetězec2»*`)`

Funkce vrací zápornou hodnotu, pokud je *«řetězec1»* menší než *«řetězec2»*. Funkce vrací kladnou hodnotu, pokud je *«řetězec1»* větší než *«řetězec2»*. Pokud se řetězce shodují, vrací funkce 0. Porovnání je citlivé na velikost písmen.

StrCSpn

Funkce pro práci s textovými řetězci

Vrací index prvního znaku řetězce, který je prvkem množiny znaků

§ `integer StrCSpn(string` *«řetězec»*`, string` *«množina znaků»*`)`

Funkce zjistí index prvního znaku *«řetězce»*, který je prvkem *«množiny znaků»*.

Malý příklad:

```
echo StrSpn("abeceda", "efgh");
```

Vypíše 2, protože `a` a `b` nejsou v množině znaků, ale třetí znak `e` (s indexem 2) je v množině znaků `efgh`.

`StrSpn()` na straně 419.

StrFTime

Funkce pro práci s datem a časem

Formátování časových údajů

§ `string StrFTime(string` *«formát»*`, integer` *«čas»*`)`

Funkce slouží k formátování údajů o čase a datu. Výsledný formát je určen formátovacím řetězcem. V něm můžeme použít znaky se speciálním významem uvedené v tabulce 9-8 na následující straně. Pokud použijeme znak, který nemá speciální význam, stane se součástí zformátovaného řetězce. Druhý parametr je nepovinný a udává čas jako počet sekund od 1. ledna 1970. Pokud parametr nepoužijeme, automaticky se použije údaj o aktuálním čase, který vrací funkce `Time()`. Funkce nemusí pracovat na všech operačních systémech, ale pouze na těch, které podporují volání `strftime()`.

Funkce `StrFTime()` na rozdíl od `Date()` již nyní vrací časové údaje na základě nastavených národních zvyklostí.

Znak	Popis
%a	Zkratka dne v týdnu
%A	Jméno dne v týdnu
%b	Zkratka jména měsíce
%B	Jméno měsíce
%c	Reprezentace data a času podle národních zvyklostí
%d	Číslo dne v měsíci (01–31)
%H	Hodina (00–23)
%I	Hodina (01–12)
%j	Číslo dne v roce (001–366)
%m	Číslo měsíce (01–12)
%M	Minuta (00–59)
%p	Indikátor dopoledne/odpoledne
%S	Sekunda (00–59)
%U	Číslo týdne v roce (01–51) — týden začíná nedělí
%w	Číslo dne v týdnu (0–6; 0 odpovídá neděli)
%W	Číslo týdne v roce (01–51) — týden začáná pondělím
%x	Reprezentace data podle národních zvyklostí
%X	Reprezentace času podle národních zvyklostí
%y	Rok jako dvojčíslí (např. 98)
%Y	Rok jako čtyřčíslí (např. 1998)
%z, %Z	Časová zóna nebo její zkratka
%%	Znak '%'

Tab. 9-8: Parametry formátovacího řetězce funkce StrFTime()

Výpis data v češtině:

```
SetLocale("LC_ALL", "Czech");
echo StrFTime("%A %d. %B %Y"); // Vypíše čtvrtek 03. září 1998
```

`Date()` na straně 272, `GMDate()` na straně 306, `Time()` na straně 424, `MkTime()` na straně 359.

StripSlashes Funkce pro práci s textovými řetězci

Odstranění lomítek umístěných před citlivé znaky funkcí `AddSlashes()`

§ `string StripSlashes(string` *«řetězec»*`)`

Vrátí *«řetězec»* s odstraněnými zpětnými lomítky (např. z ‘\"’ zůstane pouze ‘"’).

`AddSlashes()` na straně 258.

Ukázky použití naleznete na straně 120.

StrLen Funkce pro práci s textovými řetězci

Zjistí délku řetězce

§ `integer StrLen(string` *«řetězce»*`)`

Vrátí délku *«řetězce»* ve znacích.

StrPos Funkce pro práci s textovými řetězci

Nalezení podřetězce v řetězci

§ `integer StrPos(string` *«řetězec»*`, string` *«hledaný text»*`)`

Funkce vrací pozici *«hledaného textu»* v *«řetězci»*. Pokud *«řetězec» «hledaný text»* neobsahuje, vrací funkce hodnotu `false`.

Pokud jako *«hledaný text»* nepoužijeme výraz typu `string`, převede se parametr na celé číslo a hledá se pak výskyt znaku s tímto ASCII-kódem.

Pokud je hledaný znak na prvním místě řetězce, vrací funkce pozici 0. Nula však odpovídá hodnotě `false`. Na to si musíme dát pozor — snadno můžeme vytvořit podmínku, která nerozpozná, zda je hledaný znak na prvním místě v řetězci nebo zda znak v řetězci vůbec není. Řešením je použití následující kódu, který v řetězci `$kupkaSena` hledá znak uložený v proměnné `$jehla`:

```
if (($pos=StrPos($kupkaSena, $jehla)) || $kupkaSena[0]==$jehla)
    echo "Znak $jehla je na pozici $pos";
else
    echo "Řetězec neobsahuje znak $jehla";
```

`StrRPos()` na straně 418, `StrStr()` na straně 419, `StrChr()` na straně 414 a `SubStr()` na straně 422.

StrRev Funkce pro práci s textovými řetězci

Obrátí text v řetězci

§ `string StrRev(string` *«řetězec»*`)`

Vrátí obsah *«řetězce»* napsaný pozpátku.

StrRChr Funkce pro práci s textovými řetězci

Nalezení posledního výskytu znaku v řetězci

§ `string StrRChr(string` *«řetězec»*`, string` *«znak»*`)`

Funkce nalezne v *«řetězci»* poslední výskyt *«znaku»* a vrátí obsah *«řetězce»* až od tohoto *«znaku»* včetně. Pokud *«znak»* v řetězci není, funkce vrací hodnotu `false`. Pokud jako *«znak»* použijeme jiný typ než `string`, je převeden na číslo a interpretován jako znak s odpovídajícím ASCII-kódem.

Malý příklad:

```
echo StrRChr("abeceda", "e")        // Vytiskne eda
```

StrRPos Funkce pro práci s textovými řetězci

Nalezení posledního výskytu znaku v řetězci

§ `integer StrRPos(string` *«řetězec»*`, string` *«znak»*`)`

Funkce vrací pozici posledního výskytu *«znaku»* v *«řetězci»*. Pokud *«znak»* v řetězci není, vrací funkce hodnotu `false`.

Jako *«znak»* můžeme u této funkce použít pouze jeden znak. Pokud použijeme delší řetězec, bude se stejně hledat jen jeho první znak.

Pokud jako *«znak»* nepoužijeme výraz typu `string`, převede se parametr na celé číslo a hledá se pak výskyt znaku s tímto ASCII-kódem.

Pokud je hledaný znak na prvním místě řetězce, vrací funkce pozici 0. Nula však odpovídá hodnotě `false`. Na to si musíme dát pozor — snadno můžeme vytvořit podmínku, která nerozpozná, zda je hledaný znak na prvním místě v řetězci nebo zda znak v řetězci vůbec není. Řešením je použití následující kódu, který v řetězci `$kupkaSena` hledá znak uložený v proměnné `$jehla`:

```
if (($pos=StrRPos($kupkaSena, $jehla)) || $kupkaSena[0]==$jehla)
    echo "Znak $jehla je na pozici $pos";
else
    echo "Řetězec neobsahuje znak $jehla";
```

`StrPos()` na straně 417, `StrStr()` na straně 419, `StrRChr()` na straně 418 a `SubStr()` na straně 422.

StrSpn Funkce pro práci s textovými řetězci

Vrací počet znaků ze začátku řetězce, které vyhovují množině znaků

§ `integer StrSpn(string` *«řetězec»*`, string` *«množina znaků»*`)`

Funkce zjistí kolik znaků od začátku *«řetězce»* je prvkem *«množiny znaků»*.

Malý příklad:

```
echo StrSpn("abeceda", "abcd");
```

Vypíše 2, protože třetí písmeno — `e` — není mezi znaky `abcd`.

`StrCSpn()` na straně 415.

StrStr Funkce pro práci s textovými řetězci

Nalezení prvního výskytu textu v řetězci

§ `string StrStr(string` *«řetězec»*`, string` *«hledaný text»*`)`

Funkce nalezne v *«řetězci»* první výskyt *«hledaného textu»* a vrátí obsah *«řetězce»* až od tohoto *«hledaného textu»* včetně. Pokud *«řetězec» «hledaný text»* neobsahuje, vrací funkce hodnotu `false`. Pokud jako *«hledaný text»* použijeme jiný typ než `string`, je převeden na `integer` a hledá se písmeno s odpovídajícím ASCII-kódem.

Malý příklad:

```
echo StrStr("kokon", "ko")          // Vytiskne kokon
```

StrTok

Funkce pro práci s textovými řetězci

Rozložení řetězce na části.

§ `string StrTok(string «řetězec», string «oddělovače»)`

Funkce slouží k postupnému získávání částí řetězce, které jsou navzájem oddělené oddělovačem. Oddělovač je jeden znak a může jich být použito více najednou. Při prvním volání se funkce volá s oběma parametry. Při dalších voláních se jako parametr udávají již pouze *«oddělovače»*. Funkce vrací příslušnou část *«řetězce»*. Pokud je již řetězec zpracován celý, funkce vrací hodnotu `false`.

Pokud chceme zahájit rozkládání dalších řetězců, musíme znovu funkci `StrTok()` inicializovat tím, že ji zavoláme s oběma parametry.

Pokud chceme vypsat jednotlivá slova věty „Ahoj Babi, jak se máš?“, můžeme použít následující skriptíček:

```
$text = "Ahoj Babi, jak se máš?";
$token = StrTok($text," ");
while($token):
    echo "$token<BR>";
    $token = StrTok(" ");
endwhile;
```

Funkce `StrTok()` si však sama o sobě neporadí s texty, které bezprostředně za sebou obsahují dva oddělovače. V tomto případě vrátí `StrTok()` prázdný řetězec, který je interpretován jako `false` a dojde k ukončení provádění cyklu. Problém může vyřešit následující funkce `NStrTok()`, která si interně uchovává potřebné informace pro správnou tokenizaci i těchto problematických řetězců. Použití funkce je zřejmé z ukázky:

```
function NStrTok(&$token, $string, $separators)
{
    static $lastString = "";
    static $stringLen = 0;

    if ($lastString!=$string):
        $lastString = $string;
        $stringLen = StrLen($string);
        $aStrTok = StrTok($string, $separators);
```

```
        $stringLen -= StrLen($aStrTok) + 1;
    else:
        $aStrTok = StrTok($separators);
        $stringLen -= StrLen($aStrTok) + 1;
    endif;

    $token = $aStrTok;
    if (!$aStrTok)
        return $stringLen>=0;
    else
        return true;
}

$text = "Ahoj       Babi, jak  se máš?";
while(NStrTok($token, $text, " ")):
    echo $token."<BR>";
endwhile;
```

`Split()` na straně 412 a `Explode()` na straně 289.

StrToLower Funkce pro práci s textovými řetězci

Převede řetězec na malá písmena

§ `string StrToLower(string` *«řetězec»*`)`

V *«řetězci»* převede všechna velká písmena na malá. Tato funkce nemusí správně zpracovat znaky s českou diakritikou, pokud není v hostitelském systému správně nastavena podpora národního prostředí.

`StrToUpper()` na straně 421, `SetLocale()` na straně 408 a `StrTr()` na straně 422.

StrToUpper Funkce pro práci s textovými řetězci

Převede řetězec na velká písmena

§ `string StrToUpper(string` *«řetězec»*`)`

V *«řetězci»* převede všechna malá písmena na velká. Tato funkce nemusí správně zpracovat znaky s českou diakritikou, pokud není v hostitelském systému správně nastavena podpora národního prostředí.

`StrToLower()` na straně 421, `SetLocale()` na straně 408 a `StrTr()` na straně 422.

StrTr — Funkce pro práci s textovými řetězci

Převod znaků v řetězci podle konverzní tabulky

§ `string StrTr(string` *«řetězec»*`, string` *«původní»*`, string` *«nové»*`)`

Funkce v *«řetězci»* převede všechny znaky obsažené v řetězci *«původní»* na odpovídající znaky z řetězce *«nové»*

Pokud chceme z řetězce odstranit znaky s přehláskou, použijeme:

```
$str = StrTr($str, "äöüÄÖÜ", "aouAOU");
```

`StrToUpper()` na straně 421 a `StrToLower()` na straně 421.

Ukázky použití naleznete na straně 181.

StrVal — Proměnná

Převod hodnoty na řetězec

§ `string StrVal(mixed` *«výraz»*`)`

Převede *«výraz»* na řetězec. Typ *«výrazu»* přitom může být `integer`, `double` nebo `string`.

`DoubleVal()` na straně 282 a `IntVal()` na straně 336.

SubStr — Funkce pro práci s textovými řetězci

Vrátí část řetězce

§ `string SubStr(string` *«řetězec»*`, integer` *«začátek»*`, integer` *«délka»*`)`

Funkce slouží k získání části řetězce. Část je určena pozicí prvního znaku (parametr *«začátek»*) a počtem znaků (parametr *«délka»*). Pokud je *«začátek»* záporné číslo, pozice prvního znaku bude počítána od konce *«řetězce»*. Pokud jako *«délku»* uvedeme záporné číslo, vracený řetězec bude končit *«délka»* znaků před koncem *«řetězce»*.

Malý příklad:

```
echo SubStr("0123456789", -3, 3);   // Vypíše poslední tři znaky 789
echo SubStr("0123456789", 5, 2);    // Vypíše 56
echo SubStr("0123456789", 6, -2);   // Vypíše 67
```

Ukázky použití naleznete na straně 163.

SymLink — Funkce pro práci se soubory

Vytvoření symbolického odkazu na soubor

§ `integer SymLink(string` *«cílový soubor»*`, string` *«odkaz»*`)`

Funkce vytvoří symbolický *«odkaz»* na *«cílový soubor»*. Pokud se odkaz povedlo vytvořit, vrací funkce `true`, v opačném případě `false`.

`Link()` na straně 354, `ReadLink()` na straně 404 a `LinkInfo()` na straně 354.

SysLog — Konfigurace a informace o PHP

Zápis zprávy do systémového protokolu

§ `void SysLog(integer` *«priorita»*`, string` *«zpráva»*`)`

Funkce zapíše do systémového protokolu *«zprávu»*. Parametr *«priorita»* určuje závažnost zapisované zprávy. K dispozici máme následující konstanty:

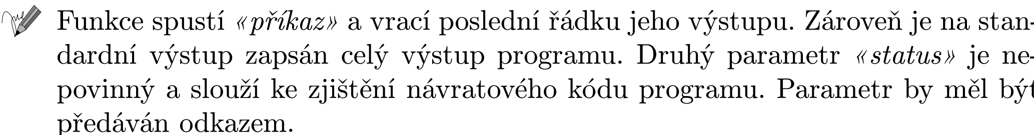
`LOG_EMERG`	havarijní stav;
`LOG_ALERT`	varování;
`LOG_CRIT`	kritická situace — např. hardwarové porucha;
`LOG_ERR`	běžná chyba;
`LOG_WARNING`	upozornění;
`LOG_NOTICE`	stav, který není chybou, ale měl by být zaznamenán;
`LOG_INFO`	informativní zpráva;
`LOG_DEBUG`	ladicí zpráva.

System

Funkce pro spouštění externích programů

Spuštění externího programu a zobrazení výstupu

§ `string System(string «příkaz», integer «status»)`



Funkce spustí *«příkaz»* a vrací poslední řádku jeho výstupu. Zároveň je na standardní výstup zapsán celý výstup programu. Druhý parametr *«status»* je nepovinný a slouží ke zjištění návratového kódu programu. Parametr by měl být předáván odkazem.

Funkce se snaží po každém řádku výstupu programu vyprázdnit výstupní buffery WWW-serveru tak, aby k uživateli výstup dorazil okamžitě.

Pokud *«příkaz»* obsahuje parametry zadávané uživatelem, měli bychom na příkaz aplikovat funkci `EscapeShellCmd()`. Uživateli tak zabráníme v „ošálení" příkazu.

`Exec()` na straně 288, `PassThru()` na straně 387, `EscapeShellCmd()` na straně 288 a `POpen()` na straně 398.

Tan

Matematická funkce

Tangens

§ `double Tan(double «výraz»)`

Spočítá hodnotu funkce tangens pro *«výraz»* zadaný v radiánech.

`Cos()` na straně 270 a `Sin()` na straně 409.

TempNam

Funkce pro práci se soubory

Vytvoření jedinečného jména souboru

§ `string TempNam(string «adresář», string «prefix»)`

Funkce vygeneruje jedinečné jméno souboru v *«adresáři»* začínající na *«prefix»*. Funkce vrací celé jméno souboru včetně cesty. Funkci využijeme zejména pro generování jmen dočasných souborů.

Pro vytvoření dočasného souboru můžeme použít příkaz:

```
$jmenoa = TempNam("/tmp", "php");          // v Unixu
$jmenob = TempNam("c:\\temp", "php");      // ve Windows
```

Time

Funkce pro práci s datem a časem

Zjištění aktuálního časového údaje

§ `integer Time(void)`

Vrací aktuální časový údaj vyjádřený počtem sekund, které uplynuly od 1. ledna 1970.

`Date()` na straně 272 a `GMDate()` na straně 306.

Touch

Funkce pro práci se soubory

Nastavení času poslední modifikace souboru

§ `integer Touch(string` *«soubor»*`, integer` *«čas»*`)`

Funkce nastaví čas poslední modifikace *«souboru»* na *«čas»*, který se udává jako počet sekund od začátku 1. ledna 1970. Parametr *«čas»* je nepovinný, a pokud jej vynecháme, použije se aktuální čas. Pokud *«soubor»* neexistuje, volání funkce jej vytvoří.

Úspěšné volání funkce vrátí hodnotu `true`, neúspěšné `false`.

`FileMTime()` na straně 292.

Trim

Funkce pro práci s textovými řetězci

Odstranění mezer a tabulátorů ze začátku a konce řetězce

§ `string Trim(string` *«řetězec»*`)`

Z konce a ze začátku řetězce odstraní všechny netisknutelné znaky, jako mezery, tabulátory a konce řádků.

`Chop()` na straně 266, `LTrim()` na straně 356 a `RTrim()` na straně 407.

UASort

Funkce pro práci s poli

Setřídí pole na základě uživatelem definované funkce pro porovnání a zachová indexy prvků

§ `void UASort(array` *«pole»*`, fptr` *«funkce»*`)`

Funkce setřídí prvky *«pole»* podle uživatelem definované *«funkce»*. Podrobné informace o tom, jak si nadefinovat *«funkci»*, nalezneme u popisu funkce `USort()`. Po setřídění jsou zachovány indexy jednotlivých prvků.

`USort()` na straně 429, `KSort()` na straně 344, `ASort()` na straně 260, `ARSort()` na straně 259, `RSort()` na straně 406 a `Sort()` na straně 411.

UCFirst

Funkce pro práci s textovými řetězci

Převede první znak řetězce na velké písmeno

§ `string UCFirst(string` *«řetězec»*`)`

V *«řetězci»* převede první písmeno z malého na velké. Tato funkce nemusí správně zpracovat znaky s českou diakritikou, pokud není v hostitelském systému správně nastavena podpora národního prostředí.

`UCWords()` na straně 426, `SetLocale()` na straně 408, `StrToLower()` na straně 421 a `StrToUpper()` na straně 421.

UCWords

Funkce pro práci s textovými řetězci

Převede první znak každého slova v řetězci na velké písmeno

§ `string UCWords(string` *«řetězec»*`)`

V *«řetězci»* převede první písmeno každého slova z malého na velké. Za začátek slova se považuje první znak řetězce a písmeno za mezerou. Tato funkce nemusí správně zpracovat znaky s českou diakritikou, pokud není v hostitelském systému správně nastavena podpora národního prostředí.

`UCFirst()` na straně 425, `SetLocale()` na straně 408, `StrToLower()` na straně 421 a `StrToUpper()` na straně 421.

UKSort

Funkce pro práci s poli

Setřídí pole podle obsahu indexů na základě uživatelem definované funkce pro porovnání a zachová indexy prvků

§ `void UKSort(array` *«pole»*`, fptr` *«funkce»*`)`

Funkce setřídí prvky *«pole»* podle hodnoty jejich indexů. Indexy se porovnávají pomocí *«funkce»*. Podrobné informace o tom, jak si nadefinovat *«funkci»*, nalezneme u popisu funkce `USort()`. Jediný rozdíl je v tom, že *«funkci»* jsou k porovnání předány indexy prvků, a nikoliv obsah prvků pole. Po setřídění jsou zachovány indexy jednotlivých prvků.

`USort()` na straně 429, `KSort()` na straně 344, `ASort()` na straně 260, `ARSort()` na straně 259, `RSort()` na straně 406 a `Sort()` na straně 411.

UMask — Funkce pro práci se soubory

Nastavení masky přístupových práv pro nově vytvářené soubory

§ `integer UMask(integer` *«maska»*`)`

Volání funkce `UMask()` nastaví masku přístupových práv na *«maska»* `& 0777`. Maska přístupových práv určuje, která přístupová práva nebudou u nově vytvářených souborů nastavena. Funkce vrací starou hodnotu masky přístupových práv. Pokud funkci zavoláme bez parametrů, vrací aktuální hodnotu masky přístupových práv.

Pokud používáme PHP jako serverový modul, je po ukončení obsluhy každého požadavku nastavena maska přístupových práv na svoji původní hodnotu.

UnLink — Funkce pro práci se soubory

Smazání souboru

§ `integer UnLink(string` *«soubor»*`)`

Funkce smaže *«soubor»*. Funkce vrací `true`, pokud se povede soubor smazat. Pokud ne, vrací `false`.

`RmDir()` na straně 406.

UnSet — Proměnná

Zrušení proměnné

§ `void UnSet(mixed` *«proměnná»*`)`

Funkce uvolní paměť vyhrazenou pro *«proměnnou»*. Funkci můžeme použít, pokud chceme uvolnit paměť používanou již nepotřebnými proměnnými. Ještě častější využití je v případech, kdy chceme mít ve skriptu obsah některé proměnné skutečně prázdný a nepřejeme si nežádoucí ovlivnění počáteční hodnoty proměnné pomocí parametru předaného v URL.

`IsSet()` na straně 341.

Ukázky použití naleznete na straně 220.

URLDecode

Funkce pro práci s URL adresami

Rozkóduje řetězec zakódovaný jako URL

§ `string URLDecode(string` *«řetězec»*`)`

Rozkóduje *«řetězec»* tak, jako by se jednalo o text zakódovaný podle pravidel pro URL. To znamená, že všechny výskyty znaku ‘`%`’ následovaného dvěma šestnáctkovými číslicemi budou nahrazeny znakem s odpovídajícím ASCII-kódem; znak ‘`+`’ je nahrazen mezerou. Právě nahrazováním plusu mezerou se tato funkce liší od `RawURLDecode()`. Pro dekódování dotazu zaslaného společně s URL nebo dat zaslaných metodou POST bychom měli používat funkci `URLDecode()`, protože starší prohlížeče kódují mezeru jako ‘`+`’.

Tak například příkaz

```
echo URLDecode("Jan+Nov%E1k%40mail.cz")
```

vypíše text

```
Jan Novák@mail.cz
```

`RawURLDecode()` na straně 403 a `URLEncode()` na straně 428.

URLEncode

Funkce pro práci s URL adresami

Zakóduje řetězec tak, aby byl použitelný v URL

§ `string URLEncode(string` *«řetězec»*`)`

Vrací řetězec, ve kterém jsou všechny nealfanumerické znaky s výjimkou znaků ‘`.`’, ‘`-`’ a ‘`_`’ nahrazeny znakem procenta ‘`%`’ následovaným dvěma šestnáctkovými číslicemi. Šestnáctkové číslo přitom vyjadřuje kód zakódovaného znaku. Mezery jsou nahrazeny znakem ‘`+`’.

Funkce se hodí pro předávní parametrů pomocí URL:

```
echo '<A HREF="search.php?query=', URLEncode("Jan Novák"), '">Hledej</A>';
```

`URLDecode()` na straně 427 a `RawURLEncode()` na straně 403.

---

uSleep

Ostatní funkce

Pozastavení provádění skriptu

§ `void uSleep(integer` *«mikrosekundy»*`)`

Pozastaví běh skriptu na zadaný počet *«mikrosekund»*. (Pokud bádáte nad názvem funkce, zkuste se zamyslet nad tím, jak moc podobné je písmeno `u` řeckému μ.)

`Sleep()` na straně 410.

USort

Funkce pro práci s poli

Setřídí pole na základě uživatelem zadané funkce pro porovnávání prvků pole

§ `void USort(array` *«pole»*`, fptr` *«funkce»*`)`

Funkce setřídí *«pole»* na základě uživatelem definované *«funkce»* pro porovnání prvků pole. Funkci využijeme v případech, kdy pole jako prvky obsahuje další pole nebo objekty, pro které potřebujeme speciálně definovat relaci uspořádání.

«funkce» musí vracet 0, pokud jsou dva prvky pole stejné. Pokud je první prvek předaný funkci k porovnání větší, vrací funkce 1, pokud je menší, −1. Vše si ukážeme na příkladě.

Máme vytvořenou třídu pro uchovávání jmen a příjmení lidí. Pole `$seznam` obsahuje jména několika lidí. Protože pole obsahuje jako prvky objekty, nemůžeme jej setřídit běžnou funkcí `Sort()`. Použijeme proto `USort()` a nadefinujeme si vlastní funkci pro porovnání:

```
class CClovek                          // Třída pro uchování jména a příjmení
{
    var $Jmeno, $Prijmeni;             // členské proměnné

    function CClovek($Jmeno, $Prijmeni) // konstruktor
    {
        $this->Jmeno = $Jmeno;
        $this->Prijmeni = $Prijmeni;
```

```
    }
}

function CClovekCompare($a, $b)         // Funkce porovnávající dva lidi
{
    if ($a->Prijmeni == $b->Prijmeni):  // Shoda příjmení?
        if ($a->Jmeno == $b->Jmeno):    // ==> musíme porovnat i jména
            return 0;                   // Shodné jméno i příjmení
        else:
            return ($a->Jmeno > $b->Jmeno) ? 1 : -1;  // Porovnání jmen
        endif;
    else:
        return ($a->Prijmeni > $b->Prijmeni) ? 1 : -1; // Porovnání příjmení
    endif;
}

function CClovekPrint($a)               // Vytištění jména člověka
{
    echo $a->Prijmeni." ".$a->Jmeno."\n";
}

// Inicializace pole
$seznam[] = new CClovek("Jan", "Novák");
$seznam[] = new CClovek("Josef", "Procházka");
$seznam[] = new CClovek("Alena", "Kládová");
$seznam[] = new CClovek("Aleš", "Novák");

Array_Walk($seznam, "CClovekPrint");    // Vytištění nesetříděného pole
USort($seznam, "CClovekCompare");       // Setřídění pole
Array_Walk($seznam, "CClovekPrint");    // Vytištění setříděného pole
```

`ARSort()` na straně 259, `ASort()` na straně 260, `KSort()` na straně 344, `RSort()` na straně 406 a `Sort()` na straně 411.

UniqID

Ostatní funkce

Jedinečný identifikátor

§ `string UniqID(string` *«prefix»*`)`

Funkce vygeneruje jedinečný identifikátor, který začíná řetězcem *«prefix»*. Funkci můžeme využít např. pro generování jedinečných identifikátorů jednotlivých připojení k serveru zasílaných jako cookies. Prefix může být dlouhý až 114 znaků.

Ukázky použití naleznete na straně 450.

Virtual

Funkce pracující pouze na serveru Apache

Provedení požadavku obslouženého Apachem

§ `integer Virtual(string` *«URL»*`)`

Funkce slouží k okamžitému vyvolání dalšího požadavku, jehož výstup se stane součástí aktuálně prováděného skriptu. *«URL»* požadavku však musí směrovat na stejný server — musíme použít relativní URL. Příkaz se hodí pro začleňování výstupu CGI-skriptů či stránek používajících SSI. Volání funkce je ekvivalentní s SSI-příkazem:

```
<!--#include virtual="«URL»"-->
```

Pokud se vše podaří, vrací funkce `true`, v opačném případě `false`. Funkci nemůžeme používat pro začleňování výstupu dalších PHP skriptů — k tomu slouží příkazy `include` a `require`.

10. Protokol HTTP

Protokol HTTP (Hypertext Transfer Protokol) je jedním ze stavebních kamenů celé služby World-Wide Web. Používá se pro komunikaci mezi prohlížečem a serverem. Pro vývoj profesionálních aplikací musíme mít představu o tom, jak protokol pracuje, jaké má schopnosti a jaká omezení. V první části této kapitoly se proto s protokolem HTTP podrobně seznámíme. Navíc si ukážeme, jak můžeme chování protokolu ovlivnit přímo z našich skriptů.

Druhá část kapitoly se věnuje rozšíření protokolu HTTP, které je známé pod názvem cookies (sušenky, koláčky). Protokol HTTP je sám o sobě bezstavový — každý požadavek prohlížeče na server je samostatná operace a server nepozná, jestli souvisí s nějakým jiným požadavkem. Pro mnoho aplikací je však potřeba při přechodu mezi jednotlivými stránkami zachovávat různé stavové informace. Cookies jsou jedním z řešení, jak se vypořádat z bezstavovostí protokolu HTTP.

Poslední část kapitoly se věnuje popisu práce s knihovnou PHPLIB. Tato knihovna mimo jiné umožňuje některé proměnné definovat jako globální pro celou aplikaci a určitého uživatele. Tyto proměnné jsou pak přístupné ve všech skriptech, které slouží ke generování stránek, po kterých se pohybuje jeden uživatel. Bez této funkce si lze jen těžko představit nějakou větší fungující aplikaci. PHPLIB je zařazena do této kapitoly, protože je založena na současném využití cookies a databází.

10.1 Hypertext Transfer Protocol

Protokol HTTP vychází z architektury klient/server. Klient — v našem případě prohlížeč — se spojí se serverem a pošle mu požadavek. Server jako reakci na klientův požadavek zasílá odpověď. Aby si spolu klient se serverem rozuměly, musí používat jistá pravidla komunikace, kterým se říká protokol. Přesný formát požadavku a odpovědi je v tomto případě definován ve specifikaci protokolu HTTP. Celou situaci mírně komplikuje to, že dnes existují tři verze protokolu — 0.9, 1.0 a 1.1. Formát požadavku a odpovědi se v jednotlivých verzích odlišuje.

Protokol HTTP je aplikačním protokolem, který pro vlastní přenos dat sítí používá protokol nižší úrovně, jež zajišťuje spolehlivé datové spojení. Tímto protokolem je nejčastěji protokol TCP. Klient se nejčastěji připojuje k WWW-serveru na port 80, i když je možno v konfiguraci serveru vybrat libovolný jiný port.

Jak to funguje

Podívejme se nejprve na nejstarší verzi protokolu HTTP. V HTTP/0.9 je formát požadavku velice jednoduchý. Jde o jeden řádek, kde je za klíčovým slovem `GET` uvedena cesta k požadovanému dokumentu. Pokud chceme pomocí HTTP/ /0.9 získat například dokument `http://www.server.cz/~nekdo/linky.html`, musíme se připojit k počítači `www.server.cz` na port 80 (standardní port služby WWW) a zaslat požadavek:

```
GET /~nekdo/linky.html
```

Jako odpověď nám přijde obsah souboru `linky.html`. Vše si můžeme sami vyzkoušet a zahrát si na prohlížeč. Stačí spustit telnet a přihlásit se k nějakému serveru na port 80. Pak napíšeme HTTP požadavek a necháme se překvapit, co nám server odešle zpět.

Na tomto místě musíme připomenout, že cesta za slovem GET nemůže být prázdná a musí vždy obsahovat alespoň lomítko:

```
GET /
```

Samotné lomítko se v tomto případě odkazuje na kořenový adresář WWW-serveru.

Možná vás napadlo, jaký dokument odešle server jako odpověď, pokud v požadavku není určen přímo soubor, ale pouze adresář (nebo dokonce jen kořenový adresář jako v našem posledním příkladě). V tomto případě WWW-server hledá v adresáři soubor, který se jmenuje `index.html`, `default.html` či `welcome.html` — záleží na konfiguraci. Pokud jej nalezne, odešle prohlížeči zpět informaci o přesném umístění souboru a prohlížeč již požádá o správný soubor.

Pokud soubor s vyhrazeným jménem v adresáři neexistuje, odešle server výpis obsahu adresáře (pokud to není v konfiguraci serveru zakázáno).

Jména souborů, které se hledají v případě požadavku, který obsahuje pouze jméno adresáře, lze pro server Apache nastavit v konfiguračním souboru pomocí direktivy `DirectoryIndex`. V IIS máme obdobnou možnost. V okně pro nastavení vlastností serveru vybereme kartu Documents a zaškrtneme na ní volbu Enable Default Document. Do seznamu pak můžeme přidávat jména souborů, které se prohlížeči vracejí v případě nezadání jména souboru.

Pokud na svých serverech hodláte hodně používat PHP skripty, vyplatí se vám nastavit si v konfiguraci jako „standardně servírovaný“ dokument i soubor typu `index.php`, aby jste nebyli omezeni pouze na statické HTML stránky.

Formát požadavku

V novějších verzích protokolu HTTP (1.0 a 1.1) je syntaxe požadavku již o něco složitější:

«metoda» «URL dokumentu» «verze HTTP»
«hlavičky»
«prázdná řádka»

Nejpoužívanější metodou je GET — ta slouží k získání daného dokumentu ze serveru. Další dvě metody, které můžeme použít v HTTP/1.0, jsou HEAD a POST. První z nich způsobí pouze zaslání hlaviček, které obsahují různé metainformace o dokumentu — např. datum poslední modifikace apod. Metoda POST slouží k odeslání dat z formuláře na server. V tomto případě za prázdnou řádkou obsahuje požadavek ještě hodnoty jednotlivých polí formuláře.

HTTP/1.1 pak definuje ještě další metody. Pro potřeby pohodlného publikování na Webu jsou to především metody PUT a DELETE. První z nich slouží k uložení zaslaného objektu (nejčastěji HTML stránky) na dané URL. HTML-editor tak může přímo na WWW-server uložit nově vytvořenou nebo modifikovanou stránku, aniž by se o to musel snažit uživatel sám například pomocí FTP. Metoda DELETE slouží k odstranění stránky ze serveru. Další metody TRACE, CONNECT a OPTIONS slouží ke zjišťování, analyzování a nastavení způsobu spojení.

Součástí HTTP požadavku od verze 1.0 je i identifikace použitého protokolu, která se uvádí za jméno požadovaného dokumentu.

Hlavičky slouží pro přenos různých doplňujících informací. Každá hlavička je na samostatné řádce a má tvar

«jméno hlavičky»: *«hodnota»*

Význam jednotlivých hlaviček popíšeme později. Jejich použití je nepovinné, a proto nejjednodušší požadavek v HTTP/1.0 vypadá takto:

```
GET /~nekdo/linky.html HTTP/1.0
«prázdná řádka»
```

Jeden z rozdílů mezi HTTP/1.0 a HTTP/1.1 spočívá v povinném používání hlavičky `Host` v novější verzi protokolu. Jako hodnota hlavičky se uvádí doménové jméno serveru, ze kterého požadujeme stránku. Nejjednodušší požadavek pak vypadá takto

```
GET /~nekdo/linky.html HTTP/1.1
Host: www.server.cz
«prázdná řádka»
```

Použití této hlavičky je povinné kvůli virtuálním serverům. Dnes je na Internetu běžné, že poskytovatelé připojení umožňují na svých serverech

vystavovat stránky svým zákazníkům. Jako službu nabízejí i umístění stránek na adrese typu `http://www.firma.cz` místo dříve obvyklejšího `http://www.poskytovatel.cz/firma/`. Samozřejmě, že poskytovatel na jednom serveru (počítači) vystavuje stránky několika firem. Problém je však v tom, že server má obvykle jen jednu IP-adresu, ke které se připojují klienti. Prohlížeč se tedy připojí k serveru na port 80 a v požadavku HTTP/1.0 pošle pouze cestu k dokumentu. Server nemá šanci zjistit, z kterého virtuálního serveru je dokument požadován. Tato situace se řešila přiřazením několika IP-adres jednomu počítači. Pro každý virtuální server musela existovat jedinečná IP-adresa. V tomto případě již server podle IP-rozhraní, ke kterému se prohlížeč připojil, mohl určit virtuální server, na který požadavek směřuje. Toto ne příliš elegantní řešení zbytečně plýtvalo IP-adresami a kladlo zvýšené požadavky na konfiguraci serveru při přidání nového virtuálního serveru. Tím, že požadavky HTTP/1.1 obsahují jméno serveru, odpadá potřeba zřizování nové IP-adresy pro každý virtuální server. Na jedné IP-adrese nyní může být přístupný neomezený počet virtuálních serverů. Jediným háčkem jistě elegantního řešení problému je fakt, že starší prohlížeče nepodporují HTTP/1.1.

Formát odpovědi

Narozdíl od HTTP/0.9 je v novějších verzích protokolu HTTP odpověď doplněna o hlavičky a další užitečné informace. Její obecný formát je zhruba následující:

«protokol» «stavový kód» «stavové hlášení»
«hlavičky»
«prázdná řádka»
«obsah odpovědi»

Protokol udává použitou verzi protokolu (`HTTP/1.0` nebo `HTTP/1.1`). Stavový kód je třímístné číslo, které indikuje, jak se povedlo uspokojit požadavek. Přehled stavových kódů je uveden v tabulce 10-1 na následující straně.

Stavové hlášení je slovní popis stavového kódu, který je pro člověka přeci jen srozumitelnější než nic neříkající číslo. Pokud vše proběhlo v pořádku, měla by první řádka odpovědi vypadat následovně (pokud server pro odpověď použil HTTP/1.1):

```
HTTP/1.1 200 OK
```

Protokol HTTP ve verzi 1.0 a nižší vytvářel pro každý objekt nové spojení, po kterém se objekt přenesl. Pokud tedy např. webová stránka obsahovala čtyři obrázky, pro její stažení bylo potřeba navázat 5 spojení (1 stránka + 4 obrázky). Navázání spojení je však náročné jak na čas, tak na přenosovou kapacitu sítě. HTTP/1.1 proto spojení mezi klientem a serverem po vyřízení požadavku neuzavírá, ale umožňuje jej použít pro přenos více objektů. Tím dochází k ušetření

Kód	Popis
1xx – Informační kódy	
`100 Continue`*	Klient může pokračovat v zasílání požadavku
`101 Switching Protocols`*	Server mění protokol používaný při komunikaci
2xx – Úspěšné vyřízení požadavku	
`200 OK`	Požadavek byl úspěšně zpracován
`201 Created`	Výsledkem požadavku je nově vytvořený objekt
`202 Accepted`	Požadavek byl přijat, ale dosud není zpracován
`204 No content`	Požadavek byl úspěšně zpracován, ale jeho výsledkem nejsou žádná data pro klienta
3xx – Přesměrování	
`300 Multiple Choices`*	Požadovaný objekt je dostupný ve více formátech
`301 Moved Permanently`	Požadovaný objekt byl trvale přemístěn na jinou adresu
`302 Moved Temporarily`	Požadovaný objekt byl dočasně přemístěn na jinou adresu
`304 Not Modified`	Objekt nebyl změněn (odpověď při podmíněném požadavku pomocí hlavičky `If-Modified-Since`)
4xx – Chyba klienta	
`400 Bad Request`	Špatná syntaxe dotazu
`401 Unauthorized`	Objekt je dostupný pouze po autentifikaci
`403 Forbidden`	Požadavek je v pořádku, ale server nemá povoleno jej vykonat
`404 Not Found`	Požadovaný objekt nebyl na serveru nalezen
`405 Method Not Allowed`*	Požadovaný objekt není dostupný použitou metodou
`406 Not Acceptable`*	Požadovaný objekt není k dispozici ve formátu podporovaném klientem
`408 Request Timeout`*	Klient nedokončil požadavek ve stanoveném čase
`410 Gone`*	Požadovaný objekt je trvale odstraněn
`415 Unsupported Media Type`*	Požadavek obsahuje data ve formátu, který server nepodporuje
5xx – Chyba na straně serveru	
`500 Internal Server Error`	Serveru se něco stalo a nemůže splnit požadavek
`501 Not Implemented`	Server nepodporuje metodu uvedenou v požadavku
`502 Bad Gateway`	Server, pracující jako gateway, dostal špatnou odpověď od dalšího serveru
`503 Service Unavailable`	Služba je nedostupná (přetížení, údržba serveru)
`505 HTTP Version Not Supported`*	Server nepodporuje verzi HTTP použitou v požadavku
* Stavové kódy označené hvězdičkou jsou součástí HTTP/1.1	

Tab. 10-1: Nejčastější stavové kódy a hlášení HTTP/1.0 a HTTP/1.1

času potřebného k přenosu webových stránek. Spojení může ukončit klient nebo server tím, že do požadavku/odpovědi zařadí hlavičku `Connection: close`.

Hlavičky

Verze protokolu HTTP 1.0 definovala 17 hlaviček. HTTP/1.1 tento počet ještě zvětšilo. My si stručně objasníme význam nejdůležitějších hlaviček. Každý požadavek/odpověď může obsahovat některou z následujících obecných hlaviček.

- `Date`
Hlavička `Date` obsahuje informaci o čase vytvoření zprávy. Používaný formát času by měl odpovídat dokumentu RFC 1123 — např.:

```
Date: Sun, 06 Nov 1994 08:49:37 GMT
```

Jak vidíte, čas je udáván v Greenwichském čase.

- `Cache-Control` — HTTP/1.1
Pomocí této hlavičky můžeme ovlivnit chování vyrovnávacích pamětí a proxy--serverů. Pokud jako její hodnotu uvedeme `no-cache`, odpověď serveru se nebude ukládat do vyrovnávací paměti.

- `Connection` — HTTP/1.1
Pokud chce klient nebo server uzavřít spojení, pošle jako obsah této hlavičky hodnotu `close`.

- `Pragma`
Pomocí této hlavičky můžeme odesílat různé nestandardní hlavičky. Mnoho HTTP/1.0 klientů i serverů např. rozumí hodnotě `no-cache`, která má stejný význam jako uvedení této hodnoty v nové hlavičce `Cache-Control`.

Speciálně pro potřeby požadavků jsou vyhrazeny následující hlavičky.

- `Accept` — HTTP/1.1
Tato hlavička obsahuje informace o typech dat podporovaných klientem. U každého typu můžeme ještě pomocí parametru `q` určit akceptovatelnost daného typu dat od nuly do jedné. Následující hlavička např. říká, že náš prohlížeč akceptuje HTML stránky a má radši obrázky ve formátu PNG než GIF:

```
Accept: text/html, image/png; q=0.8, image/gif; q=0.4
```

- `Accept-Charset` — HTTP/1.1
Pomocí této hlavičky můžeme určit, které znakové sady podporuje náš prohlížeč. Tato hlavička je v Čechách obzvláště užitečná, protože umožní serveru automaticky vyřešit dilema s výběrem vhodného kódování češtiny. Pokud nemáme rádi

kódování Windows, ale na druhou stranu máme radši Windows než stránky bez diakritiky, můžeme to vyjádřit následovně:

```
Accept-Charset: iso-8859-2, windows-1250; q=0.8, us-ascii; q=0.2
```

- `Accept-Language` — HTTP/1.1
V této hlavičce klient uvádí své jazykové preference. Na jejich základě pak může server odeslat patřičnou jazykovou mutaci stránky:

```
Accept-Language: cs, en; q=0.5
```

- `Authorization`
Informace potřebné pro autentifikaci uživatele. Hlavička obsahuje jméno použitého autorizačního mechanismu (nejčastěji `Basic`). Za ním následuje jméno a heslo uživatele oddělené dvojtečkou a zakódované metodou Base64. Kódování Base64 je pouze transportní kódování nikoliv kryptografické a tak nemůžeme heslo zakódované Base64 považovat za bezpečně přenášené. Pokud chceme zajistit bezpečný přenos hesla, musíme pro komunikaci použit SSL nebo jiný mechanismus pro vytvoření šifrovaného spojení mezi serverem a klientem.

- `From`
Tato hlavička obsahuje elektronickou adresu uživatele, který pracuje s klientem. Valná většina prohlížečů však tuto informaci s požadavky neposílá, aby ochránila soukromí uživatele.

- `Host` — HTTP/1.1
Tato hlavička obsahuje doménovou adresu počítače, kterému klademe dotaz. Hlavička nalézá uplatnění při jednoznačné identifikaci virtuálního serveru.

- `If-Modified-Since`
Pokud v požadavku použijeme tuto hlavičku společně s nějakým datem, server nám požadovaný objekt vrátí pouze tehdy, pokud byl od zadaného data změněn. Příklad: Chceme získat dokument pouze v případě, pokud se od 30. března 1998 změnil:

```
If-Modified-Since: Mon, 30 Mar 1998 12:00:00 GMT
```

- `Referer`
Tato hlavička obsahuje adresu stránky, na které bylo získáno URL právě kladeného požadavku. Server tak může zjistit, na kterých stránkách jsou na něj odkazy.

- `User-Agent`
V hlavičce `User-Agent` posílá prohlížeč svoji identifikaci — obvykle své jméno, číslo verze a platformu, na které je spuštěn.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)
```

Stejně tak máme k dispozici i několik hlaviček speciálně určených pro použití v odpovědi.

- `Location`

Tato hlavička obsahuje adresu dokumentu, který byl přesunut. Tuto hlavičku server posílá v případech, kdy stavový kód požadované operace začíná na 3. Prohlížeč většinou automaticky nahraje stránku, na kterou Location ukazuje. Jako adresu je potřeba uvést úplné absolutní URL. Např.

```
Location: http://manes.vse.cz/~xkosj06/index.html
```

URL použité jako hodnota hlavičky `Location` musí být absolutní. Mnoho serverů a skriptů v Internetu posílá špatnou hlavičku, která obsahuje pouze relativní URL — je to však v rozporu se standardem HTTP.

- `Server`

V hlavičce `Server` posílá server svoji identifikaci — obvykle své jméno a číslo verze. Například:

```
Server: Apache/1.3b3
```

- `WWW-Authenticate`

Tuto hlavičku server odesílá klientovi v případě, že požadovaný zdroj je přístupný pouze autentifikovaným uživatelům. Obsahem hlavičky je typ autentifikace (nejčastěji `Basic`) a jméno aplikace určené pomocí atributu `realm`. Například

```
WWW-Authenticate: Basic realm="vshop"
```

Následující hlavičky může obsahovat jak požadavek, tak odpověď — vážou se totiž k tělu zprávy.

- `Allow`

Pomocí této hlavičky určujeme, jakými metodami je předávaný objekt dostupný. Pokud v požadavku použijeme nepodporovanou metodu, vrátí nám server v této hlavičce seznam metod, které můžeme použít pro přístup k objektu:

```
Allow: GET, HEAD
```

- `Content-Type`

Tato hlavička udává typ přenášených dat. Typ dat se zapisuje pomocí MIME konvence. Pro HTML stránky máme typ `text/html`, pro obyčejný text se používá typ `text/plain`, obrázky mají podle použitého formátu jeden z typů `image/gif`, `image/jpeg` nebo `image/png`. Podle typu dat prohlížeč pozná, jak příchozí data interpretovat. HTML stránka zasílaná prohlížeči jako odpověď, proto mezi hlavičkami obsahuje následující řádku

```
Content-Type: text/html
```

Za typem dat můžeme uvést ještě doplňující atributy. Společně s typem `text/html` můžeme zadat atribut `charset`, který určuje znakovou sadu použitou v HTML dokumentu:

```
Content-Type: text/html; charset=iso-8859-2
```

- `Expires`

Pomocí této hlavičky určujeme dobu, dokdy je odpověď platná. Po uplynutí daného časového okamžiku nesmí být odpověď čtena z vyrovnávací paměti, ale musí být znovu přečtena ze serveru. Formát data používaný v hlavičce `Expires` je stejný jako pro hlavičku `Date`.

Pokud nechceme, aby byla odpověď uchovávána ve vyrovnávací paměti, musíme čas v hlavičce `Expires` nastavit na stejnou hodnotu jako hlavičku `Date` — tj. na čas vygenerování požadavku.

Z mnoha zdrojů se dozvíme, že jako obsah hlavičky `Expires` můžeme používat hodnotu 0. V tomto případě se odpověď nebude ukládat do vyrovnávacích pamětí. Použití 0 a ostatních hodnot, které neodpovídají podporovanému formátu data, je však v rozporu se standardem HTTP. Musíme mít vždy na paměti, že bez dodržovaní standardů by Internet nebyl dnes tím, čím je. V následující sekci si ukážeme, jak snadno vygenerovat správnou hlavičku `Expires` pomocí PHP.

- `Last-Modified`

Tato hlavička obsahuje datum poslední modifikace dokumentu zaslaného jako odpověď.

Práce s hlavičkami v PHP

Při psaní skriptů můžeme s hlavičkami pracovat dvěma způsoby. Jednak můžeme číst hlavičky, které s požadavkem zaslal prohlížeč. Druhé využití spočívá v odeslání vlastních hlaviček společně s odpovědí vygenerovanou skriptem.

Podívejme se nejprve na čtení hlaviček, které zaslal prohlížeč s požadavkem. Hlavičky jsou přístupné hned dvěma způsoby. Jednak jsou hlavičky předávány pomocí proměnných. Tyto proměnné předá systému PHP buď rozhraní CGI, nebo přímo příslušný WWW-server. Např. hlavička `User-Agent` bude dostupná v proměnné `$HTTP_USER_AGENT`. Jak vidíme, před název hlavičky se doplní `HTTP_`, název hlavičky se převede na velká písmena a všechny znaky '`-`' se ve jméně hlavičky nahradí podtržítkem '`_`'.

Čtení obsahu hlaviček pomocí proměnných se hodí v případech, kdy potřebujeme znát konkrétní obsah některé hlavičky. Pokud však chceme znát obsahy všech hlaviček, použijeme s výhodou funkci `GetAllHeaders()`, která v asociativním poli vrací všechny hlavičky. Zpracování všech hlaviček je pak velice jednoduché. Můžeme k tomu použít obdobu následujícího skriptu:

```
$headers = GetAllHeaders();
while (list($hlavicka, $hodnota) = each($headers))
    echo "$hlavicka: $hodnota<BR>\n";
```

Jedinou nevýhodou funkce `GetAllHeaders()` je to, že ji PHP podporuje pouze tehdy, pokud je zkompilováno jako modul serveru Apache.

Pokud má naopak náš skript nějakou hlavičku vygenerovat, musíme použít funkci `Header()`. Parametrem funkce je již přímo HTTP-hlavička, která se má odeslat prohlížeči. Protože se hlavičky odesílají ještě před samotným tělem odpovědi, musíme funkci `Header()` používat ještě před tím, než PHP vygeneruje nějaký výstup.

Pokud například vytváříme skript, který generuje obrázek ve formátu GIF, měli bychom na jeho začátku použít následující příkaz

```
Header("Content-Type: image/gif");
```

Tím prohlížeči sdělíme, že následující data představují obrázek ve formátu GIF.

Funkci pro generování hlaviček můžeme s výhodou kombinovat s dalšími funkcemi, které má PHP k dispozici. Pokud chceme mít jistotu, že se výsledek skriptu nebude ukládat do vyrovnávací paměti, můžeme na začátku dokumentu použít:

```
Header("Pragma: no-cache");
Header("Cache-Control: no-cache");
Header("Expires: ".GMDate("D, d M Y H:i:s")." GMT");
```

Poslední řádka přitom nedělá nic jiného, než že do hlavičky `Expires` uloží aktuální čas ve správném formátu.

V předchozím textu jsme si říkali, že před hlavičkami odpovědi je zasílán stavový kód požadavku. Pokud chceme napsat skript, který umí vracet nestandardní stavové kódy, nic nám v PHP nestojí v cestě. Naše skripty se však budou lišit pro CGI-verzi PHP a pro PHP jako modul serveru.

Pokud používáme PHP přes CGI, můžeme šikovně využít vlastnosti rozhraní CGI. To použije obsah vrácené hlavičky `Status` k vygenerování stavového kódu:

```
Header("Status: 402 Payment Required");
```

Pokud používáme PHP jako serverový modul, musíme stavový kód odeslat jako vůbec první hlavičku a musíme jej vygenerovat včetně použité verze protokolu:

```
Header("HTTP/1.1 402 Payment Required");
```

Pokud chceme automaticky dosadit verzi HTTP, kterou použil klient, můžeme využít následující úpravu:

```
Header("$SERVER_PROTOCOL 402 Payment Required");
```

10.2 Cookies

Pomocí systému PHP lze vytvářet opravdu zajímavé aplikace. Ovšem i aplikace napsané v PHP mají jistá nepříjemná omezení vyplývající z principu protokolu HTTP. Protokol HTTP je nestavový. Znamená to, že pro přenos každé stránky se otevírá nové zvláštní HTTP spojení, které se ihned po přenosu uzavře. Server a potažmo i náš skript nemá tedy moc šancí zjistit, zda jej nějaký uživatel spouští poprvé nebo podesáté.

Částečnou možností řešení tohoto problému je ukládání pomocných stavových informací do skrytých polí formuláře anebo tyto informace přidávat do cesty v URL za jméno skriptu. Tyto techniky jsou využívány poměrně často a mnoho problémů uspokojivě vyřeší. Ovšem ani tyto dva způsoby neřeší problém trvalého uložení nějakých informací. Když totiž prohlížeč příští den spustíme znovu, server už o tom, že jsme po něm včera brouzdali, vůbec neví. Tento problém řeší *cookies*, což je rozšíření protokolu HTTP pocházející z dílny firmy Netscape. Dnes je toto rozšíření podporováno snad všemi prohlížeči. My se nejprve podíváme na to, co nám cookies nabídnou z uživatelského hlediska. Pak si ukážeme, jak cookies využít v našich skriptech.

Pokud si chceme prohlédnout stránku uloženou na některém serveru, může server v odpovědi na náš požadavek zaslat i informaci, kterou má klient (prohlížeč) uložit pro další použití. Pokud je pak někdy v budoucnosti navazováno spojení se stejným serverem, jsou mu tyto informace zaslány zpět. Informace jsou ukládány do souboru, který bývá uložen ve stejném adresáři jako prohlížeč nebo v uživatelově adresáři. Ve jméně souboru či adresáře, kam se cookies

ukládají, poměrně často narazíme na slovo cookies. Většinou jsou informace o cookies uloženy v textovém souboru, a proto si je můžeme prohlédnout téměř libovolným textovým editorem.

Cookies nejsou vázány na naše jméno ani e-mailovou adresu. Jsou společné pro jednu instalaci prohlížeče (i ta však může být víceuživatelská). Server se naše jméno ani další soukromé údaje z cookies nemůže dozvědět. Jedinou možností by bylo vyplnění těchto údajů do nějakého formuláře na serveru a zaslání těchto údajů jako cookies zpět prohlížeči pomocí skriptu, který obsluhuje formulář. Cookies tedy neumožňují přenos uživatelského jména a dalších osobních údajů bez vědomí uživatele, jak se mnoho neinformovaných uživatelů Internetu domnívá.

Server si může například uchovávat informace o tom, jaké stránky jsme navštívili a kolik času jsme strávili jejich prohlížením. Tyto informace mohou být využity při statistickém vyhodnocování návštěvnosti jednotlivých stránek serveru. Mohly by být využity i nějakou marketingovou společností, pokud by stránky obsahovaly např. nabídky zboží a služeb či inzerci. V tomto případě bychom již mohli diskutovat o tom, zda je využívaní těchto informací legitimní. Vzhledem k tomu, že bez našeho vědomí nelze získat naši adresu, nemůže se nám stát, že by z ničeho nic naši poštovní schránku zavalily nabídky sekaček na trávu.

Mnohem užitečnější se jeví využití cookies při ukládání konfiguračních informací. Server si tak může zjistit naše posledně použité nastavení těch WWW-stránek, které byly generovány dynamicky. Dnes se této vlastnosti vznešeně říká personalizace.

Jako reakce na tlak uživatelů je ve většině dnešních prohlížečů možno podporu cookies vypnout, a to jak napořád, tak i jen pro jednu relaci. Tato funkce je však z dnešního pohledu mor pro tvůrce webových aplikací. V mnoha uživatelích přetrvává zcela neoprávněný pocit, že cookies ohrožují jejich soukromí, a proto si cookies vypínají. Aplikace se pak s prohlížeči, které cookies nepodporují, musí nějak vypořádat.

Poněkud problematičtější je zajištění informací uložených pomocí cookies před servery, které k nim nemají mít přístup (tj. nevytvořily je). I když prohlížeč vrátí cookies pouze tomu serveru, který je uložil, teoreticky existují metody, jejichž použitím se server může prokazovat jako nějaký jiný server.

Cookies mohou být vázány i na konkrétní podadresáře serveru, nemusí tedy být společné pro jeden server (doménu). Cookies se přenáší pomocí protokolu HTTP. Specifikace vyžaduje, aby byl klient schopen uložit alespoň 300 cookies po 4 KB a alespoň 20 cookies pro jeden server, případně doménu.

Z předchozího textu vidíme, při používání cookies je potřeba provádět dvě operace: ukládat si cookies na klientovi a číst cookies zaslané klientem spolu s požadavkem na nějaké URL.

Uložení cookies

Cookies se klientovi posílají v HTTP hlavičce. Obecný formát je

```
Set-Cookie: «jméno»=«hodnota»; expires=«datum»; path=«cesta»;
            domain=«doménová adresa»; secure
```

Jedinou povinnou částí je přitom *«jméno»*=*«hodnota»*. Ta slouží k nastavení cookie se jménem *«jméno»* na hodnotu *«hodnota»*.

- *«jméno»*=*«hodnota»*

Slouží k nastavení cookie na určitou hodnotu. Pokud později přijde od serveru cookie se stejným jménem, ale jinou hodnotou, má cookie tuto novou hodnotu. Celý řetězec nesmí obsahovat středník a mezery. V případě, že je potřebujeme, musíme je překódovat pomocí procentové notace podobně jako v URL. Nastavení cookie `Jmeno` na hodnotu `Jan Novák` tedy provedeme následovně:

```
Set-Cookie: Jmeno=Jan%20Novák
```

protože hexadecimálně vyjádřený kód znaku mezera je 20.

- `expires=`*«datum»*

Atribut `expires` určuje, dokdy má cookie platnost. Po určeném datu se cookie neukládá ani se neposílá serveru. Formát data si ukážeme na čase oběda v pondělí 20. ledna 1997:

```
Mon, 20-Jan-97 12:00:00 GMT
```

Formát data je velice podobný formátu data používanému přímo v protokolu HTTP. Jednotlivé části data však musí být odděleny spojovníkem a rok se udává pouze dvojciferně. Mnohé novější prohlížeče jsou však již připraveny na Y2K a podporují i v cookies čtyřmístný rok.

Atribut `expires` lze použít i k vymazání nepotřebné cookie. Pokud zašleme cookie s dobou platnosti v minulosti, prohlížeč na cookie zapomene.

- `domain=`*«doménová adresa»*

Pokaždé, když se prohlížeč chystá odeslat cookies, porovnává doménovou adresu z URL, které má být vyvoláno, s atributem `domain`. Při porovnávání stačí, aby `domain` bylo částí doménové adresy. Pokud by mohla být cookie zaslána díky shodě domén, je provedeno ještě porovnání cesty (atribut `path`). Pokud bude tedy `domain=acme.com`, vyhoví adresy `anvil.acme.com` i `shipping.create.acme.com` a může se přistoupit k porovnání cesty.

Pokud atribut `domain` nenastavíme, použije se doménová adresa serveru, který cookie prohlížeči poslal. Tento atribut využijeme, pokud potřebujeme, aby se cookies předávala více serverům z jedné domény.

• `path=`*«cesta»*
Pomocí tohoto atributu určujeme tu část URL v doméně, pro kterou je cookie platná. Jestliže cookie vyhověla porovnání domén, provede se porovnání cest. Obsah atributu `/kolo` vyhoví např. těmto cestám v URL `/kolo/author.html`, `/kolowrat`. Pokud atribut nastavíme na `/`, vyhoví to všem URL, kde se shoduje doména.

Pokud atribut `path` nespecifikujeme, použije se cesta z URL dokumentu, s nímž byla cookie zaslána.

• `secure`
Pokud byla cookie zaslána prohlížeči s tímto atributem, bude na server poslána zpět pouze v případě, že spojení je bezpečné. To prakticky znamená, že spojení probíhá pomocí SSL (Secure Socket Layer), což je specifikace zabezpečené komunikace mezi klientem a serverem.

Pro zaslání cookies ve skriptu tedy stačí přidat do HTTP-hlavičky jednu nebo více položek `Set-Cookie`.

V PHP máme dvě možnosti. Jednak můžeme použít funkci `Header()` k přidání cookie do hlavičky odpovědi:

```
Header("Set-Cookie: Jmeno=Jan%20Novák");
```

Druhou, používanější metodou, je využití funkce `SetCookie()`. Použití je velmi jednoduché:

```
SetCookie("Jmeno", "Jan Novák");
```

Při použití funkce `SetCookie()` se PHP samo postará o vygenerování správné hlavičky a konverzi všech nepřípustných znaků v hodnotě cookies.

> Cookies se odesílají společně s HTTP hlavičkami. Funkci `SetCookie()` musíme stejně jako funkci `Header()` použít ještě před tím, než náš skript produkuje nějaký výstup.

Funkce `SetCookie()` nabízí i několik nepovinných parametrů, pomocí kterých můžeme nastavit všechny výše popsané atributy. Například hned třetím parametrem je doba platnosti cookie. Udává se stejně jako v ostatních funkcích PHP jako počet sekund od 1. ledna 1970. Pokud tedy chceme odeslat cookie s dobou platnosti jedna hodina ($60 \times 60 = 3600$ sekund), můžeme použít následující kód:

```
SetCookie("Jmeno", "Jan Novák", Time()+3600);
```

Využíváme přitom skutečnosti, že funkce `Time()` vrací aktuální čas.

Čtení cookies

Klient s každým požadavkem na dokument o určitém URL posílá i všechny vyhovující cookies. Ty jsou posílány jako součást HTTP hlavičky v následujícím tvaru:

Cookie: *«jméno1»*=*«hodnota1»*; *«jméno2»*=*«hodnota2»*; ...

Pro nás je však důležitější, že takto zaslanou hlavičku předá server našemu skriptu v proměnné `$HTTP_COOKIE`. Odtud si můžeme cookies přečíst a naložit s nimi dle vlastního uvážení.

Nebylo by to však PHP, aby nám nenabízelo i mnohem pohodlnější způsob pro práci s cookies. Ze všech cookies, které z prohlížeče dorazí, nám PHP vytvoří proměnné. Například obsah naší dříve nastavené cookie bychom mohli zjistit v proměnné `$Jmeno`.

Třetí variantou pro zjištění cookies je použití pole `$HTTP_COOKIE_VARS`. Toto asociativní pole obsahuje všechny cookies, které skriptu prohlížeč zaslal.

Jednoduchá ukázka použití cookies

Použití cookies si samozřejmě ukážeme i prakticky. Napíšeme skript, který bude vytvářet speciální stránku. Tato stránka bude ukazovat, kolikrát jsme ji již navštívili. Přístupy se však budou narozdíl od klasických počítadel přístupů počítat zvlášť pro každého uživatele. Navíc nás stránka při prvním navštívení uvítá trošku jiným textíkem.

```
<?
    $pocet++;
    SetCookie("pocet", $pocet);
?>
<HTML>
<HEAD>
<TITLE>Inteligentní společník</TITLE>
</HEAD>
<BODY>

<? if ($pocet==1): ?>
    <H1>Vítáme vás poprvé na našem serveru</H1>
<? else: ?>
    <H1>Vítáme vás na našem serveru</H1>
    Jste tu již po <?echo $pocet?>.
<? endif ?>

<P>V případě problémů nebo dotazů kontaktujte
```

```
<A HREF="mailto:webmaster@server.cz">Webmastera</A>.
</BODY>
</HTML>
```

Celý skript pracuje na velice jednoduchém principu. Využívá cookie s názvem `pocet`, do které ukládá počet návštěv serveru. Při požadavku na skript je obsah cookie předán v proměnné `$pocet`. Proměnná je nejprve o jedničku zvětšena a aktualizovaná hodnota je poté odeslána zpět prohlížeči. Stránka vygenerovaná skriptem se liší podle počtu dosavadních přístupů (viz obr. 10-1 na následující straně).

S počáteční inicializací proměnné `$pocet` si v tomto případě nemusíme lámat hlavu. Pokud nám klient obsah cookie nezašle (první přístup k serveru nebo zakázané cookies), bude proměnná `$pocet` obsahovat prázdnou hodnotu, kterou PHP chápe jako nulu. Následnou inkrementací dostaneme v proměnné `$pocet` počáteční počet návštěv roven jedné.

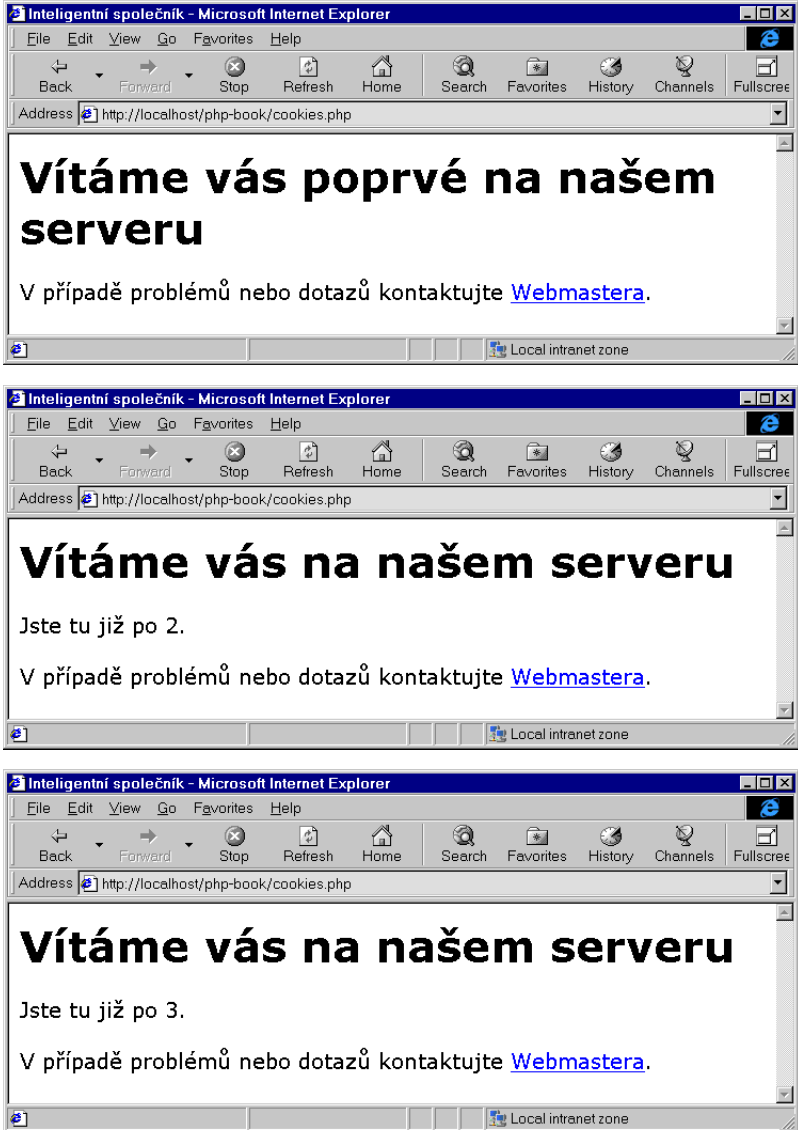
Naše skripty bychom měli navrhnout tak, aby fungovaly (klidně s omezeným komfortem), i když klient cookies nepodporuje. Uživatelé, kteří mají cookies vypnuty nebo mají starší prohlížeč, by neměli být diskriminováni.

Naše aplikace můžeme plně založit na cookies pouze v případě, kdy zaručíme, že všichni uživatelé mají podporu cookies. Této ideální situace můžeme většinou dosáhnout pouze při vývoji Intranetových aplikací.

10.3 Proč potřebujeme knihovnu PHPLIB?

Cookies řeší některé nedostatky, které má protokol HTTP. Samy o sobě však nejsou samospasitelné. Pro skutečně komplexní webovou aplikaci potřebujeme více. Potřebujeme uchovávat stavovou informaci pro každého uživatele, který s aplikací pracuje. Stavová informace popisuje, co právě aplikace dělá a odkud se do tohoto místa dostala.

Odmysleme si nyní na chvíli Web a představme si obyčejnou aplikaci — např. textový editor. Textový editor obsahuje kód, který umožňuje zpracování textu. V editoru máme obvykle otevřeno několik dokumentů v několika oknech. Každé okno s dokumentem má vlastní kurzor na nějaké pozici, vlastní nastavení aktuálního stylu písma a samozřejmě vlastní text dokumentu. Máme tedy jeden společný kód a proměnné „pozice kurzoru“, „styl písma“ a „text“ pro každé okno.



Obr. 10-1: Ukázka jednoduchého využití cookies

Distribuovaná webová aplikace má také jeden sdílený kód a několik oken s dokumenty — každý prohlížeč je jedním takovým oknem. Při tvorbě webové aplikace musíme mít pro každý prohlížeč připojený k aplikaci jednu instanci všech stavových proměnných.

Protokol HTTP bohužel neumožňuje od sebe rozeznat jednotlivé prohlížeče. Pro naše aplikace bychom potřebovali označit každý prohlížeč nějakým jednoznačným identifikátorem a přinutit prohlížeč k tomu, aby tento identifikátor přenášel s každým požadavkem.

Pokud budeme mít k dispozici jedinečný identifikátor pro každý prohlížeč, můžeme si na serveru pro každý identifikátor ukládat obsah všech potřebných stavových proměnných. Pro složitější aplikace tedy potřebujeme:

1. Mechanismus pro označení prohlížeče jednoznačným identifikátorem.

2. Stálý úložný prostor, kde pro každý identifikátor můžeme uložit proměnné používané našimi skripty. Na začátku každého skriptu načteme z tohoto úložného prostoru hodnotu stavových proměnných a na jeho konci zase obsah proměnných uložíme. Ne vždy potřebujeme ukládat všechny proměnné, a proto potřebujeme jednoduchý způsob k označení proměnných, které se mají pro aplikaci uchovávat.

3. Všechny zmíněné mechanismy musí být dostatečně bezpečné.

Pro přenos jednoznačného identifikátoru se výborně hodí cookies. Pokud tedy prohlížeč přistoupí k našim stránkám poprvé, nemá žádný identifikátor nastaven. To nám říká, že prohlížeč zatím neznáme a že se uživatel ještě k naší aplikaci nepřihlásil. Vygenerujeme tedy identifikátor a odešleme jej jako cookies. Zároveň odešleme prohlížeči úvodní stránku aplikace. V dalších přístupech k naší aplikaci se již prohlížeč ohlásí pomocí svého identifikátoru.

Kromě toho, že by identifikátor měl být jedinečný, je dobré, když je ho těžké uhodnout. Zabráníme tím potencionálnímu narušení bezpečnosti dat v případech, kdy by někdo uhodl identifikátor někoho jiného. V PHP můžeme pro generování jednoznačných a těžko odhadnutelných identifikátorů použít funkci `UniqID()`, jejíž výstup ještě upravíme pomocí funkce `MD5()`. Dostaneme tak 32místné hexadecimální číslo, které je pro někoho jiného těžko vygenerovatelné.

Jediný problém výše popsaného řešení jsou uživatelé, kteří vedeni neoprávněným strachem vypínají podporu cookies ve svém prohlížeči. V tomto případě musíme zajistit přenos identifikátoru jiným způsobem. Jedinou možností je předávat identifikátor pomocí metody GET jako parametr ve všech odkazech a adresách skriptů pro obsluhu formuláře. Tato metoda je poměrně pracná, protože musíme upravit všechny odkazy. Na druhou stranu pak bude naše aplikace správně fungovat i s prohlížeči, které nepodporují cookies.

Pokud tedy cookie s identifikátorem nastavujeme například pomocí příkazu

```
SetCookie("session", $sessid)
```

musíme upravit všechny odkazy a formuláře ve stránkách na následující tvar

```
<A HREF="jinastranka.php?session=<?echo $sessid?>">odkaz</A>
```

respektive

```
<FORM ACTION="jinyskript.php?session=<?echo $sessid?>">
...
</FORM>
```

Ve formulářích můžeme využít i skrytá pole:

```
<FORM ACTION="jinyskript.php">
<INPUT TYPE=Hidden NAME=session VALUE=<?echo $sessid?>>
...
</FORM>
```

Vidíme, že ten, kdo si vypíná cookies, tvůrci webových aplikací opravdu zatopí.

Když máme vyřešen problém s identifikací prohlížeče, musíme se postarat o ukládání stavových proměnných pro každý identifikátor. Jelikož většina webových aplikací využívá nějaký SQL-server pro ukládání dat, jeví se jako přirozené využít jej i pro uložení stavových proměnných.

Zůstává však otázka, jak do tabulky s přesně stanovenou strukturou umístit aplikační proměnné různého typu. Řešením je převést obsah všech proměnných na jeden dlouhý řetězec a ten ukládat do položky typu BLOB. Pokud ze stavových proměnných vytvoříme řetězec, který obsahuje příkazy pro jejich inicializaci, můžeme k obnovení obsahu proměnných využít funkci `eval()`. Vytvoření řetězce z aplikačních proměnných je možné jen díky tomu, že PHP obsahuje funkce pro zjištění typu a hodnoty proměnné. Od PHP verze 3.0.6 budou k dispozici funkce `serialize()` a `unserialize()`, které umožní převod proměnné libovolného typu na řetězec a zpět.

Do databázové tabulky pak pro každý identifikátor budeme ukládat příslušnou BLOB-položku. Navíc můžeme uložit například čas poslední změny záznamu. Tuto informaci můžeme využít pro zvýšení bezpečnosti našich skriptů. Pokud například uživatel přistoupí k aplikaci po větší než třicetiminutové pauze, může si aplikace automaticky vyžádat nové zadání jména a hesla uživatele.

S využitím výše popsaných postupů můžeme naši aplikaci doplnit o mnoho užitečných a potřebných funkcí. Pokud však chceme šetřit čas a peníze, můžeme využít knihovnu PHPLIB. Knihovna staví na popsaných principech. Kromě možnosti identifikace prohlížeče a automatického udržování proměnných nabízí i další funkce, jako autentifikaci a autorizaci uživatelů a standardní třídu pro přístup k mnoha databázím. Navíc je PHPLIB neustále rozšiřována o další užitečné funkce (např. tvorba formulářů a grafů, elektronický obchod apod.).

My si ukážeme použití knihovny PHPLIB při vytvoření jednoduchého virtuálního obchodu. Vytvoříme několik stránek s nabídkou zboží, které půjde přidávat do nákupního košíku. Obsah košíku přitom bude zachován při přechodech

mezi jednotlivými stránkami s nabídkou zboží. Pro zachování obsahu košíku využijeme právě PHPLIB a její schopnost udržovat obsah zvolených proměnných mezi stránkami.

Funkčnost PHPLIB na našich stránkách aktivujeme načtením souboru `prepend.php3`. Pokud používáme PHPLIB na všech stránkách, vyplatí se tento soubor načítat automaticky pomocí konfigurační direktivy `auto_prepend_file`.

Na začátku stránky pak musíme použít funkci `page_open()`, která aktivuje potřebné funkce PHPLIB — například nastaví správně obsah stavových proměnných z předchozí stránky. Chování PHPLIB ovládáme pomocí parametru předaného funkci `page_open()`. Pokud chceme využít pouze možnosti knihovny, které umožňují udržování obsahu proměnných mezi stránkami, použijeme následující volání funkce

```
page_open(array("sess" => "IShop_Session"));
```

`IShop_Session` je přitom třída definovaná v souboru `local.inc`, který je součástí distribuce PHPLIB. V třídě musíme nastavit parametry pro přístup k databázi, do které se ukládají obsahy všech persistentních proměnných.

Pokud chceme nyní obsah některé proměnné uchovávat mezi stránkami, musíme ji zaregistrovat. Pro proměnnou `$x` můžeme použít příkaz

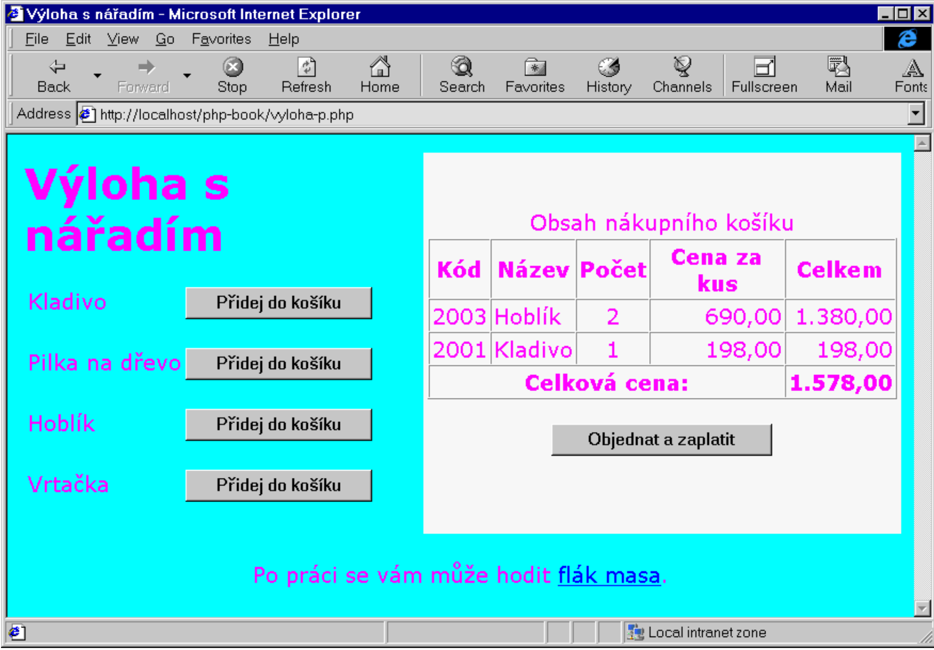
```
$sess->register("x");
```

Aby PHPLIB mohla na konci běhu každého skriptu uložit obsah zaregistrovaných proměnných, musíme použít funkci `page_close()`.

Nyní se již podívejme na naši ukázkovou aplikaci. Pro potřeby virtuálního obchodu musíme znát u každého zboží jeho název a cenu. Ve skutečné aplikaci bychom měli tyto údaje nejspíše uloženy v databázi, která by byla propojena s informačním systémem celé firmy. Abychom náš příklad zbytečně nekomplikovali prací s databází, uložíme si informace o zboží do asociativního pole `$seznamZbozi`. Jednotlivé položky pole jsou indexovány číslem zboží a obsahují informaci o názvu a ceně zboží uloženou strukturovaně pomocí třídy `Zbozi`.

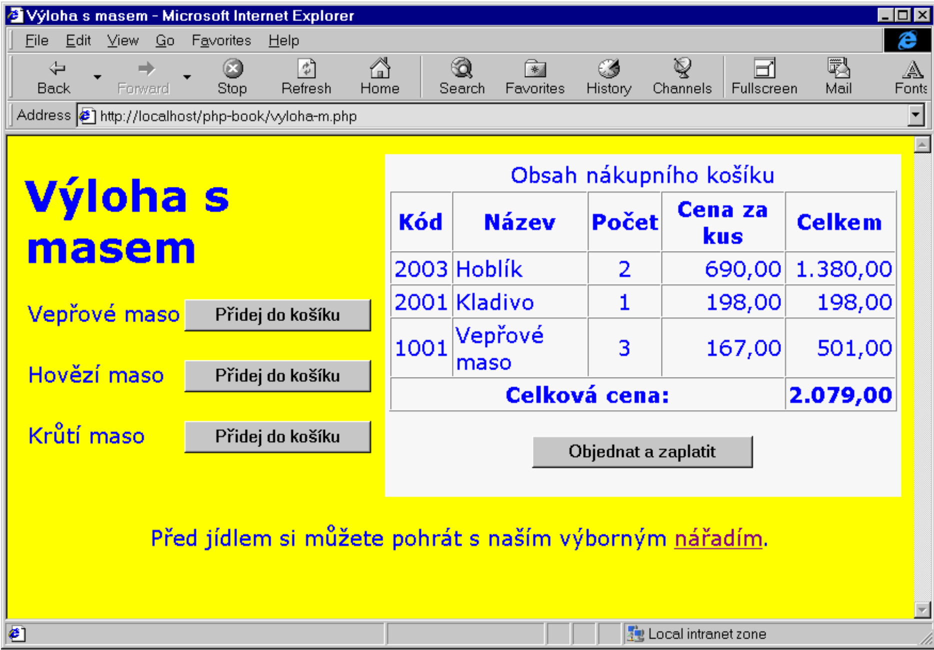
Příklad: `zbozi.php`

```
<?
// Definice zboží pro náš IShop

class Zbozi                   // třída pro uložení informací o zboží
{
    var $nazev, $cena;

    function Zbozi($aNazev, $aCena)
    {
        $this->nazev = $aNazev;
```



Obr. 10-2: Nejprve si objednáme nějaké nářadí

```
        $this->cena = $aCena;
    }
}

UnSet($seznamZbozi);        // vymažeme pole

// maso
$seznamZbozi[1001] = new Zbozi("Vepřové maso", 167);
$seznamZbozi[1002] = new Zbozi("Hovězí maso", 172);
$seznamZbozi[1003] = new Zbozi("Krůtí maso", 90);

// průmysl
$seznamZbozi[2001] = new Zbozi("Kladivo", 198);
$seznamZbozi[2002] = new Zbozi("Pilka na dřevo", 478);
$seznamZbozi[2003] = new Zbozi("Hoblík", 690);
$seznamZbozi[2004] = new Zbozi("Vrtačka", 3500);
?>
```



Obr. 10-3: K objednávce přidáme ještě něco na zub

Stránek, kde si může zákazník vybrat zboží, bude mnoho a my na nich chceme mít zobrazen nákupní košík. Bude proto vhodné do nějakého skriptu oddělit funkce pro přidání zboží do košíku a pro zobrazení obsahu košíku.

Vytvoříme proto skript `kosik.php`, který budeme načítat na začátku všech stránek s nabídkou zboží. Pokud bude proměnná `$pridej` obsahovat kód zboží, bude toto zboží přidáno do nákupního košíku. Ve skriptu definujeme funkci `UkazKosik()`, která zobrazí obsah nákupního košíku ve formě přehledné tabulky.

Obsah nákupního košíku budeme uchovávat v poli `$kosik`. Toto pole proto musíme zaregistrovat, aby jeho obsah PHPLIB uchovával mezi stránkami.

Příklad: `kosik.php`

```
<?
// Jednoduchý nákupní košík
// Nakoupené zboží je uloženo v poli $kosik
// Pokud proměnná $pridej obsahuje číslo zboží, je přidáno do košíku

$sess->register("kosik");

// přidání zboží do košíku
if ($pridej!="")                // máme něco na přidání
```

```
  if (!IsSet($kosik[$pridej]))
    $kosik[$pridej] = 1;        // první kus daného druhu zboží
  else
    $kosik[$pridej]++;          // další kus daného druhu zboží

// definice funkce pro vypsání obsahu košíku
function UkazKosik()
{
  global $kosik, $seznamZbozi;

  $celkem = 0;                  // celková cena nákupu
  echo "<TABLE CELLPADDING=2 CELLSPACING=0 BORDER=1>\n";
  echo "<CAPTION>Obsah nákupního košíku</CAPTION>\n";
  echo "<TR><TH>Kód<TH>Název<TH>Počet<TH>Cena za kus<TH>Celkem</TR>\n";
  if (Is_Array($kosik))         // košík zobrazíme pouze, když v něm něco je
    while (list($kod, $pocet)=each($kosik)):
      echo "<TR>";
      echo "<TD>$kod<TD>" . $seznamZbozi[$kod]->nazev .
           "<TD ALIGN=CENTER>$pocet<TD ALIGN=RIGHT>" .
           Number_Format($seznamZbozi[$kod]->cena, 2, ",", ".") .
           "<TD ALIGN=RIGHT>" .
           Number_Format($seznamZbozi[$kod]->cena * $pocet, 2, ",", ".") .
           "</TR>\n";
      $celkem += $seznamZbozi[$kod]->cena * $pocet;
    endwhile;
  echo "<TR><TH COLSPAN=4>Celková cena:<TH ALIGN=RIGHT>" .
       Number_Format($celkem, 2, ",", ".") . "</TR>\n";
  echo "</TABLE>\n";
  if ($celkem > 0)              // je něco k zaplacení
    echo "<P ALIGN=CENTER><FORM ACTION=koupit.php>
          <INPUT TYPE=SUBMIT VALUE=\"Objednat a zaplatit\"></FORM>\n";
}
?>
```

Samotné stránky virtuálního obchodu jsou již velice jednoduché. Na svém začátku načtou skripty s definicí pomocných funkcí. Pro každé zboží definujeme malý formulář, který ve skrytém poli odešle kód výrobku pro zařazení do košíku. Kód je přitom předán tomu samému skriptu, proto je po přidání zboží do košíku uživatel stále na téže stránce. Na libovolné místo stránky můžeme vložit kód pro zobrazení aktuálního obsahu nákupního košíku pomocí funkce `UkazKosik()`.

Příklad: `vyloha-p.php`

```
<?
    require "prepend.php3";
    page_open(array("sess" => "IShop_Session"));
    require "./kosik.php";
    require "./zbozi.php";
?>
<HTML>
<HEAD>
<TITLE>Výloha s nářadím</TITLE>
</HEAD>
<BODY BGCOLOR=CYAN TEXT=MAGENTA>
<TABLE CELLPADDING=4>
<TR>
<TD>
    <H1>Výloha s nářadím</H1>
    <TABLE>
    <TR VALIGN=TOP><TD><?echo $seznamZbozi[2001]->nazev?>
        <TD><FORM METHOD=POST>
            <INPUT TYPE=HIDDEN NAME=pridej VALUE=2001>
            <INPUT TYPE=SUBMIT VALUE="Přidej do košíku">
            </FORM></TR>
    <TR VALIGN=TOP><TD><?echo $seznamZbozi[2002]->nazev?>
        <TD><FORM METHOD=POST>
            <INPUT TYPE=HIDDEN NAME=pridej VALUE=2002>
            <INPUT TYPE=SUBMIT VALUE="Přidej do košíku">
            </FORM></TR>
    <TR VALIGN=TOP><TD><?echo $seznamZbozi[2003]->nazev?>
        <TD><FORM METHOD=POST>
            <INPUT TYPE=HIDDEN NAME=pridej VALUE=2003>
            <INPUT TYPE=SUBMIT VALUE="Přidej do košíku">
            </FORM></TR>
    <TR VALIGN=TOP><TD><?echo $seznamZbozi[2004]->nazev?>
        <TD><FORM METHOD=POST>
            <INPUT TYPE=HIDDEN NAME=pridej VALUE=2004>
            <INPUT TYPE=SUBMIT VALUE="Přidej do košíku">
            </FORM></TR>
    </TABLE>
</TD>
<TD BGCOLOR="#F0F0F0">
    <P><? UkazKosik(); ?>
```

```
</TD>
</TR>
</TABLE>

<P ALIGN=CENTER>Po práci se vám může hodit
<A HREF="vyloha-m.php">flák masa</A>.

</BODY>
</HTML>
<?  page_close() ?>
```

Tím je náš virtuální obchod hotov. Stačí doplnit skript `koupit.php`, který od uživatele získá jeho adresu a zprostředkuje zaslání požadovaného zboží na dobírku.

11. Instalace a konfigurace

Většinu času při vývoji aplikací strávíme jejich psaním a laděním. Předtím však musíme mít k dispozici nainstalované a správně nakonfigurované PHP. Následující kapitola popisuje nejběžnější způsoby instalace PHP a podrobně rozebírá všechny způsoby, jak ovlivnit chování PHP pomocí konfiguračních direktiv. Poslední část popisuje parametry, které můžeme použít při volání PHP z příkazové řádky.

11.1 Instalace ve Windows

Pro úspěšnou instalaci musíme mít k dispozici distribuci PHP pro Windows. Tu si můžeme stáhnout ze serveru PHP nebo z některého z jeho zrcadel. Získáme soubor `php3-3.0.x-win32.zip`. Samotná instalace probíhá ve dvou fázích. Nejprve nainstalujeme samotné PHP a poté přidáme jeho podporu do WWW-serveru, který máme k dispozici.

Instalace PHP je velice jednoduchá:

1. Nejprve rozbalíme distribuční soubor do nějakého adresáře. My budeme předpokládat adresář `c:\php3`.
2. Po rozbalení nalezneme v adresáři `c:\php3` soubor `php3-dist.ini`. Tento soubor zkopírujeme do adresáře s Windows a přejmenujeme na `php3.ini`. Windows jsou nejčastěji nainstalovány v adresáři `c:\windows` (Windows 95/ /98) nebo `c:\winnt` (Windows NT).
3. V souboru `php3.ini` provedeme potřebné úpravy. Jediná skutečně nutná úprava je nastavení cesty k DLL-knihovnám. Direktivu `extension_dir` proto nastavíme na hodnotu `c:\php3`. Popis všech konfiguračních direktiv naleznete na konci této kapitoly.

PHP je nyní na našem systému nainstalováno. Zbývá přidat jeho podporu do našeho oblíbeného serveru.

PWS a IIS3

Instalace pro Personal Web Server a Internet Information Server 3.0 je zcela shodná. Součástí distribuce je soubor `php_iis_reg.inf`. Tento soubor obsahuje příkazy pro změnu všech potřebných položek registrů. Pokud jsme PHP nainstalovali do jiného adresáře než `c:\php3`, musíme v souboru příslušným způsobem upravit cesty. Poté stačí vyvolat v Průzkumníkovi lokální nabídku pro soubor `php_iis_reg.inf` a z ní vybrat příkaz Install.

Adresáře, ve kterých chceme spouštět skripty, musejí mít nastaveno právo na čtení (Read) a spouštění (Execute). Správné nastavení zajistíme pomocí Personal Web Manageru. V něm aktivujeme okno Advanced, vybereme požadovaný adresář a stiskneme tlačítko Edit Properties. Zkontrolujeme, zda jsou volby Read a Execute aktivovány.

IIS4

Pokud chceme nainstalovat PHP do Internet Information Serveru 4.0, musíme se připravit na krátké klikání myší:

1. Nejprve musíme spustit program Internet Service Manager, který se však hlásí jako Microsoft Management Console. Vybereme si buď celý server, nebo jen adresář, ve kterém je uložena aplikace využívající PHP.

2. Vyvoláme okno s vlastnostmi — například tak, že z lokální nabídky vybereme příkaz Properties. V okně vybereme záložku Home Directory, Virtual Directory nebo Directory.

3. Stiskneme tlačítko Configuration a v nově zobrazeném okně vybereme záložku App Mapings.

4. Stiskneme tlačítko Add, protože chceme přidat mapování nové přípony souboru. Objeví se další dialogové okno. Do vstupního pole Executable vepíšeme `c:\php3\php.exe %s %s`. Poněkud kuriozní kombinaci znaků `%s %s` nesmíme vynechat — bez ní by PHP neběželo.

5. Do vstupního pole Extension zadáme příponu pro PHP skripty — např. `.php3`. Dialogové okno zavřeme stiskem tlačítka OK.

6. Kroky 4 a 5 můžeme opakovat pro všechny přípony, které budeme používat pro PHP skripty.

7. Nyní musíme nastavit správná přístupová práva. Všechny adresáře obsahující skripty musí mít nastaveno právo Script. Toto nastavíme ve vlastnostech adresáře na záložce Home Directory, Virtual Directory nebo Directory přepnutím volby Permission na hodnotu Script.

8. Pokud používáte NTFS, ujistěte se ještě, zda má uživatel `IUSR_XXX` přístup k souboru `php.exe`.

Apache 1.3.x

Aby Apache správně spolupracoval s PHP, musíme přidat několik řádek do konfiguračního souboru `srm.conf`.

```
ScriptAlias /php3/ "c:/php3/"
AddType application/x-httpd-php3 .php3 .php .phtml
Action application/x-httpd-php3 "/php3/php.exe"
```

Po úpravě konfigurace musíme Apache zrestartovat, aby si načetl nové nastavení konfiguračních souborů. Naše nastavení bude PHP skripty hledat v souborech s příponami `.php3`, `.php` a `.phtml`.

11.2 Instalace na Unixu

Na Unixu se PHP nejčastěji používá v kombinaci se serverem Apache. V této sekci se proto jinými servery nebudeme zabývat. PHP s nimi však může bez problémů spolupracovat — stačí jej zkompilovat jako CGI verzi.

Nejčastěji se PHP používá jako modul serveru Apache. Obvyklá instalace se provádí kompilací ze zdrojových textů Apache a PHP. Toto řešení je nejlepší a přináší největší flexibilitu díky bohatým možnostem konfigurace. Pokud nejste příznivci tohoto způsobu instalace, naleznete na serveru PHP i několik předkompilovaných distribucí. K dispozici jsou verze pro Solaris, HP-UX, Irix, FreeBSD a Linux. Distribuce jsou nejčastěji ve tvaru souborů `.tar.gz` nebo balíků RPM.

Pokud se rozhodneme pro instalaci přímo ze zdrojových textů, musíme si pořídit zdrojové texty PHP a serveru Apache. Pokud chceme v PHP využívat některé další moduly např. pro práci s databázemi či knihovnou GD, musíme mít na systému nainstalovány příslušné hlavičkové soubory a knihovny.

My budeme dále předpokládat, že máme v adresáři `/usr/local/src` distribuci Apache (`apache_1.3.x.tar.gz`) a PHP (`php-3.0.x.tar.gz`).[1]

PHP lze bezproblémově zkompilovat i se staršími verzemi Apache 1.2.x. Pokud máte nějaký závažný důvod pro použití starší verze Apache, nic vám nebrání. Postup instalace se trošku liší od verze 1.3.x a je popsán v originální dokumentaci. Doporučuji však použít verzi 1.3.x, která přináší mnohá vylepšení.

---

[1] Písmeno `x` v našich příkladech nahrazuje verzi programu — např. `apache_1.3.3.tar.gz` a `php-3.0.5.tar.gz`.

Nejprve musíme rozbalit distribuce Apache a PHP — použijeme k tomu následující příkazy:

```
gunzip apache_1.3.x.tar.gz
tar xvf apache_1.3.x.tar
gunzip php-3.0.x.tar.gz
tar xvf php-3.0.x.tar
```

Před samotnou kompilací PHP musíme alespoň jednou spustit konfigurační skript Apache:

```
cd apache_1.3.x
./configure --prefix=/usr/local/apache
```

`/usr/local/apache` v našem případě určuje adresář, kam se později Apache nainstaluje.

Nyní přistoupíme k samotné kompilaci PHP. Přepneme se do adresáře se zdrojovými texty

```
cd ../php-3.0.x
```

Nyní musíme pro PHP správně nastavit konfiguraci překladu. K tomu lze využít příkaz `./configure` doplněný o potřebné parametry. Jejich aktuální popis nalezneme v distribuci PHP. Pokud nás nezajímá žádná databázová podpora, stačí spustit skript `./configure` s parametrem určujícím cestu k distribuci Apache:

```
./configure --with-apache=../apache_1.3.x
```

Alternativou k příkazu `./configure` je spuštění skriptu `./setup`, který se zeptá na několik otázek a na základě odpovědí pak spustí příkaz `./configure` s požadovanými parametry. Samotné volání skriptu `./configure` je uloženo v souboru `./do-conf`, kde jej můžeme dále upravovat.

Po úspěšné konfiguraci přichází samotná fáze kompilace PHP a úpravy Apache pro spolupráci s PHP. Zadáme příkazy

```
make
make install
```

Pokud vše proběhlo bez problémů, můžeme nyní přistoupit ke kompilaci Apache. Přepneme se do adresáře s Apachem a nakonfigurujeme podporu pro modul PHP. Poté již můžeme spustit kompilaci a instalaci Apache.

```
cd ../apache_1.3.x
./configure --prefix=/usr/local/apache \
            --activate-module=src/modules/php3/libphp3.a
make
make install
```

V adresáři `/usr/local/apache/sbin` by nyní měla být nová binární podoba démonu `httpd`. Než s ní nahradíme předchozí server, musíme ho nejprve zastavit. Přesný způsob zastavení serveru se liší v závislosti na konkrétní předchozí instalaci. Například v distribuci Linuxu RedHat slouží k ukončení činnosti WWW-serveru příkaz

```
/etc/rc.d/init.d/httpd stop
```

Nyní musíme nové `httpd` překopírovat na místo starého. Aby Apache správně obsluhoval stránky v PHP, musíme upravit konfigurační soubor `srm.conf`. Ten by měl být uložen v adresáři `/usr/local/apache/etc`. Do souboru přidáme řádku

```
AddType application/x-httpd-php3 .php3 .php .phtml
```

pokud chceme, aby byly příkazy PHP interpretovány v souborech s příponami `.php3`, `.php` a `.phtml`.

Nyní zbývá překopírovat konfigurační soubor `php3.ini-dist` z distribuce PHP do adresáře `/usr/local/lib` pod jménem `php3.ini`. Možnosti konfigurace PHP pomocí konfiguračního souboru jsou popsány v následující sekci.

Nyní můžeme Apache nastartovat a zkusit, zda si správně poradí s PHP skripty.

Modul PHP lze při kompilaci Apache kombinovat s dalšími moduly — např. s modulem mod_czech (dynamická změna kódování češtiny) a s modulem poskytujícím rozšíření pro FrontPage.

Během konfigurace modulu PHP můžeme rovněž určit, které knihovny se mají k jádru PHP přikompilovat. Tímto způsobem se přidává například podpora pro jednotlivé databáze.

11.3 Konfigurace

PHP můžeme konfigurovat pomocí mnoha konfiguračních direktiv. Tyto direktivy ukládáme do souboru `php3.ini`. Všechny konfigurační direktivy mohou být používány v konfiguračních souborech serveru Apache, pokud používáme PHP jako modul. Před jméno direktivy je v tomto případě ještě doplněno `php3_`.

Konfigurace je čtena vždy při inicializaci parseru. Pro serverový modul PHP je parser inicializován pouze při startu WWW-serveru. CGI verze inicializuje parser pro každý požadavek znovu.

Hodnota direktivy je většinou patrná z jejího popisu. U direktiv, jejichž typ je *«boolean»*, můžeme použít hodnoty `On` (odpovídá ano) a `Off` (odpovídá ne).

Obecné konfigurační direktivy

• `asp_tags` *«boolean»*
Pomocí této direktivy můžeme povolit tagy ve stylu ASP pro oddělování PHP kódu od HTML <%. . . %>. Používání těchto oddělovačů se může hodit v kombinaci s některými HTML editory, které si neporadí s běžnými oddělovači <?. . . ?>.

• `auto_append_file` *«řetězec»*
Pomocí této direktivy můžeme určit soubor, který se automaticky zpracuje po každém skriptu. Soubor je zpracován stejným způsobem, jako kdyby byl načten příkazem `include`. Speciální hodnota *«řetězce»* `none` vypíná automatické doplňování souboru za skript.

Pokud skript ukončíme pomocí `exit`*, soubor se automaticky nepřipojí.*

• `auto_prepend_file` *«řetězec»*
Direktiva má podobný účel jako předchozí. Specifikovaný soubor je však načten ještě před samotným skriptem.

• `bcmath.scale` *«přesnost»*
Počet desetinných míst používaných pro operace s knihovnou BC.

• `browscap` *«soubor»*
Tato direktiva určuje soubor obsahující definici schopností jednotlivých prohlížečů. Obsah tohoto souboru řídí hodnoty vracené funkcí `Get_Browser()`.

• `display_errors` *«boolean»*
Direktiva určuje, zda budou vypisována chybová hlášení.

• `doc_root` *«adresář»*
Pokud je PHP spuštěno v bezpečném režimu, nejsou čteny skripty z jiného adresáře. Direktiva se používá pouze v případě, že obsahuje nějakou hodnotu.

• `engine` *«boolean»*
Direktiva umožňuje zakázat spouštění PHP. Její význam je pouze pro verzi PHP, která běží jako modul. Pomocí této direktivy můžeme PHP zakázat pro některé virtuální servery nebo adresáře.

• `error_append_string` *«řetězec»*
Tato direktiva určuje text, který se objeví po vypsání chybové hlášky. Obvykle obsahuje kód, který ukončí změnu řezu nebo barvy použitého písma v HTML.

• `error_log` *«řetězec»*
«řetězec» obsahuje jméno souboru, do kterého jsou zaznamenávány chyby. Pokud použijeme speciální hodnotu `syslog`, jsou chyby zapisovány do systémového logu. Ve Windows NT do event logu a na Unixu do syslogu. Systémový log není podporován ve Windows 95.

• `error_prepend_string` *«řetězec»*
Tato direktiva určuje text, který se objeví před vypsáním chybové hlášky. Obvykle obsahuje kód, který změní řez nebo barvu použitého písma v HTML.

• `error_reporting` *«číslo»*
Pomocí této direktivy můžeme určit, která chybová hlášení budou vypisována. Jako hodnotu uvádíme součet konstant z tabulky 11-1.

Hodnota	Popis
1	chyba
2	varování
4	syntaktická chyba
8	upozornění (např. použití neinicializované proměnné)

Tab. 11-1: Úrovně hlášení chyb

• `open_basedir` *«adresář»*
Pomocí této direktivy můžeme nastavit adresář, ve kterém může PHP pracovat se soubory. Pokud tuto direktivu použijeme, nebude PHP schopné otevřít soubor v jiném než zde uvedeném adresářovém stromě. Funkce nijak speciálně neošetřuje symbolické linky. Standardní nastavení povoluje přístup ke všem souborům.

Pokud jako hodnotu uvedeme tečku '`.`', bude povolena práce jen s těmi soubory, které jsou ve stejném adresáři jako skript.

• `gpc_order` *«řetězec»*
Pomocí této direktivy určujeme pořadí, ve kterém jsou nastavovány proměnné podle hodnot získaných metodami GET, POST a pomocí cookies. Později zpracovávaný zdroj bude mít vyšší prioritu, pokud bude mít stejné jméno.

«řetězec» obsahuje maximálně tři písmena, z nichž každé odpovídá jednomu zdroji dat pro proměnné. Metodě GET odpovídá `G`, metodě POST `P` a cookies mají `C`. Standardní nastavení je `GPC`, což znamená, že proměnné získané pomocí POST mají vyšší prioritu než ty získané pomocí GET. Cookies mají nejvyšší prioritu.

Při nastavení `GP` bude PHP zcela ignorovat cookies a nebude pro ně vytvářet proměnné. Data odeslaná metodou post budou mít i nyní vyšší prioritu.

- `include_path` *«řetězec»*
Pomocí této direktivy můžeme nastavit adresáře, ve kterých budou hledány soubory načítané pomocí `include`, `require` a několika dalších funkcí. Formát *«řetězce»* je stejný jako pro proměnnou `PATH` v daném systému — v Unixu se pro oddělování jednotlivých adresářů používá dvojtečka a ve Windows středník. Standardní nastavení je '`.`', které umožňuje načítání souborů z aktuálního adresáře.

- `log_erros` *«boolean»*
Tato direktiva určuje, zda budou chybová hlášení zapisována do chybového protokolu serveru.

- `magic_quotes_gpc` *«boolean»*
Pokud je tato direktiva aktivní, jsou všechny výskyty znaků apostrof '`'`', uvozovky '`"`', zpětné lomítko '\' a NUL (kód 0) nahrazeny odpovídající escape sekvencí ve všech proměnných získaných z dat odeslaných metodou GET nebo POST nebo z cookies.

Pokud je zároveň ještě zapnuta volba `magic_quotes_sybase`, je apostrof nahrazován dvěma apostrofy tak, jak to mají rády databáze od Sybase a Microsoftu.

- `magic_quotes_runtime` *«boolean»*
Pokud je aktivní tato volba, jsou nebezpečné znaky nahrazovány escape sekvencemi i ve výsledcích získaných za běhu skriptu, např. z funkcí pro čtení výsledků SQL-dotazů nebo funkcí pro spouštění externích programů.

- `magic_quotes_sybase` *«boolean»*
Pokud je tato direktiva aktivní a současně je aktivní i `magic_quotes_gpc`, je ve vstupních datech nahrazován apostrof dvěma apostrofy.

- `max_execution_time` *«sekundy»*
Pomocí této direktivy můžeme nastavit maximální čas běhu skriptu, než bude automaticky ukončen. Čas se zadává jako počet sekund. Tato direktiva slouží především jako ochrana před přetížením serveru chybně napsanými skripty. Pokud jako hodnotu direktivy uvedeme 0, nebude běh skriptu omezen žádným časovým limitem.

- `memory_limit` *«bajty»*
Tato direktiva určuje maximální velikost paměti, kterou může alokovat jeden skript při svém běhu. Velikost paměti je udávána v bajtech. Direktiva opět slouží jako ochrana serveru před špatně napsanými skripty s velkou paměťovou náročností.

- `precision` *«číslo»*
Počet významných číslic zobrazovaných při tisku čísel s plovoucí řadovou čárkou.

- `short_open_tag` *«boolean»*
Pomocí této direktivy můžeme povolit zkrácené tagy pro oddělování PHP kódu od HTML `<?...?>`. Pokud chceme PHP používat dohromady s XML, měli bychom tuto volbu vypnout a používat oddělovače `<?php...?>`.

- `track_errors` *«boolean»*
Pokud tuto direktivu zapneme, bude vždy poslední chybové hlášení uloženo v globální proměnné `$php_errormsg`.

- `track_vars` *«boolean»*
Pokud tuto direktivu zapneme, budou data získaná metodami GET a POST a z cookies přístupná v globálních asociativních polích `$HTTP_GET_VARS`, `$HTTP_POST_VARS` a `$HTTP_COOKIE_VARS`.

- `upload_max_filesize` *«bajty»*
Tato direktiva určuje maximální velikost souboru získaného pomocí uploadu souborů. Větší soubory budou pro upload odmítnuty.

- `upload_tmp_dir` *«adresář»*
Direktiva určuje adresář, kam se budou dočasně ukládat soubory získané uploadem souborů. PHP musí mít v adresáři právo zápisu. Optimálně se pro tyto účely hodí adresáře `/tmp`, resp. `c:\temp` apod.

- `user_dir` *«řetězec»*
Jméno adresáře, kde mají uživatelé uloženy své PHP skripty. Nastavení této direktivy má smysl pouze v případě, kdy chceme, aby i jednotliví uživatelé měli oddělený strom HTML dokumentů a skriptů.

- `warn_plus_overloading` *«boolean»*
Pokud zapneme tuto direktivu, ohlásí PHP varování vždy, když použijeme operátor ‘`+`’ na řetězce. Direktiva je užitečná v případech, kdy chceme odhalit skripty, které potřebují přepsat tak, aby pro spojování řetězců používaly operátor ‘`.`’.

- `y2k_compliance` *«boolean»*
Pomocí této volby můžeme nastavit, zda všechna odeslaná data v HTTP-hlavičkách budou obsahovat čtyřmístný rok. Zapnutí této volby může způsobit problémy některým starším prohlížečům, které podporují pouze dvoumístný rok.

Konfigurace elektronické pošty

Následující direktivy je potřeba správně nastavit, pokud chceme používat funkci `Mail()` pro odesílání elektronické pošty.

- `SMTP` *«počítač»*
Direktiva určuje adresu (doménovou nebo IP) počítače, přes který bude odesílána pošta (SMTP server). Direktiva má význam pouze ve Windows.

- `sendmail_from` *«e-mailová adresa»*
Pomocí této direktivy nastavujeme hlavičku dopisu `From`. Tato hlavička slouží k zadání adresy odesílatele. Adresu bychom měli nastavit na nějakou skutečně existující, aby byla možnost upozornit na případně nechtěně šířený mail generovaný našimi skripty.

- `sendmail_path` *«cesta»*
Cesta k programu `sendmail`. Program `sendmail` se používá pro odesílání pošty na Unixu. Pokud používáte jiný program než `sendmail` (např. `qmail`), nastavte direktivu tak, aby ukazovala na příslušný wrapper, který emuluje funkce sendmailu.

Konfigurace bezpečného režimu

- `safe_mode` *«boolean»*
Pomocí této direktivy můžeme zapnout bezpečný režim.

- `safe_mode_exec_dir` *«adresář»*
Pokud je PHP v bezpečném režimu, neumožní spouštět programy z jiného než zadaného adresáře.

Konfigurace debuggeru

- `debugger.host` *«počítač»*
Doménová nebo IP-adresa počítače používaného debuggerem.

- `debugger.port` *«port»*
Číslo portu, na kterém se můžeme připojit k debuggeru.

- `debugger.enabled` *«boolean»*
Pomocí této direktivy můžeme zapnout debugger.

Konfigurace dynamicky zaváděných modulů

- `enable_dl` *«boolean»*

Pomocí této volby můžeme povolit/zakázat používání dynamicky nahrávaných modulů. Hlavní využití této vlastnosti je pro modulovou verzi PHP, kde můžeme pro jednotlivé virtuální servery nebo adresáře zakazovat a povolovat načítání dynamických knihoven. Dynamické knihovny mohou obsahovat funkce, které obcházejí bezpečnostní mechanismy PHP, a proto je jejich načítání v bezpečném režimu zakázáno.

- `extension_dir` *«adresář»*

Jméno adresáře, ve kterém jsou umístěny dynamicky zaváděné moduly.

- `extension` *«modul»*

Pomocí této direktivy můžeme určit moduly, které se nahrají automaticky po startu PHP. Nahrávat bychom měli jen ty moduly, které používáme na většině stránek. Automatické nahrávání speciálních modulů by jen zbytečně zdržovalo ostatní aplikace a zvětšovalo paměťové nároky.

Konfigurace zvýrazňování syntaxe

Jako parametr následujících direktiv můžeme uvést cokoliv, co je platným obsahem atributu `COLOR` u elementu `FONT`. Daná barva se pak použije při zvýrazňování syntaxe zdrojových textů skriptů.

- `highlight.bg` *«barva»*

Pozadí stránky se zdrojovým kódem.

- `highlight.comment` *«barva»*

Barva komentářů.

- `highlight.default` *«barva»*

Standardní barva textu.

- `highlight.html` *«barva»*

Barva HTML kódu.

- `highlight.keyword` *«barva»*

Barva klíčových slov.

- `highlight.string` *«barva»*

Barva pro textové řetězce.

Konfigurace ODBC

- `uodbc.default_db` *«datový zdroj»*
Pomocí této direktivy můžeme určit standardní datový zdroj, který se použije v případě, kdy neuvedeme datový zdroj ve funkci `ODBC_Connect()` nebo `ODBC_PConnect()`.

- `uodbc.default_user` *«uživatel»*
Pomocí této direktivy můžeme určit uživatelské jméno, pod kterým se budeme připojovat k datovému zdroji v případě, kdy uživatelské jméno neuvedeme ve funkci `ODBC_Connect()` nebo `ODBC_PConnect()`.

- `uodbc.default_pw` *«heslo»*
Pomocí této direktivy můžeme určit heslo, pod kterým se budeme připojovat k datovému zdroji v případě, kdy heslo ve funkci `ODBC_Connect()` nebo `ODBC_PConnect()` neuvedeme.

- `uodbc.allow_persistent` *«boolean»*
Pomocí této direktivy můžeme povolit/zakázat používání persistentních spojení při připojování k ODBC datovým zdrojům.

- `uodbc.max_persistent` *«počet»*
Direktiva určuje maximální počet najednou otevřených persistentních spojení. Speciální hodnota −1 označuje neomezený počet.

- `uodbc.max_links` *«počet»*
Direktiva určuje maximální počet najednou otevřených spojení (normálních i persistentních). Speciální hodnota −1 označuje neomezený počet.

- `uodbc.defaultlrl` *«číslo»*
Pomocí této direktivy můžeme nastavit standardní počet bajtů přenášených pro BLOBy. Změnu tohoto nastavení můžeme provést pomocí volání funkce `ODBC_LongReadLen()`.

- `uodbc.defaultbinmode` *«režim»*
Pomocí této direktivy můžeme určit standardní způsob zacházení s BLOBy. Hodnoty jsou stejné jako u funkce `ODBC_BinMode()`.

Konfigurace MySQL

- `mysql.allow_persistent` *«boolean»*

Pomocí této direktivy můžeme povolit/zakázat používání persistentních spojení při připojování k databázi MySQL.

- `mysql.max_persistent` *«počet»*

Direktiva určuje maximální počet najednou otevřených persistentních spojení. Speciální hodnota −1 označuje neomezený počet.

- `mysql.max_links` *«počet»*

Direktiva určuje maximální počet najednou otevřených spojení (normálních i persistentních). Speciální hodnota −1 označuje neomezený počet.

Konfigurace PostgreSQL

- `pgsql.allow_persistent` *«boolean»*

Pomocí této direktivy můžeme povolit/zakázat používání persistentních spojení při připojování k databázi PostgreSQL.

- `pgsql.max_persistent` *«počet»*

Direktiva určuje maximální počet najednou otevřených persistentních spojení. Speciální hodnota −1 označuje neomezený počet.

- `pgsql.max_links` *«počet»*

Direktiva určuje maximální počet najednou otevřených spojení (normálních i persistentních). Speciální hodnota −1 označuje neomezený počet.

Konfigurační soubor může obsahovat ještě další konfigurační direktivy, které slouží k nastavení možností práce s databázemi mSQL, Sybase, Sybase-CT a Informix. Informace o konfiguraci těchto méně používaných databází naleznete v originální dokumentaci k PHP. Neocenitelným zdrojem informací o konfiguraci je rovněž sám standardní konfigurační soubor, který je velmi bohatě komentován.

11.4 Parametry příkazové řádky

PHP pro Windows je běžný spustitelný soubor, kterým můžeme interpretovat skripty přímo z příkazové řádky. Na Unixu můžeme PHP zkompilovat tak, že dostaneme rovněž spustitelný soubor použitelný pro interpretaci skriptů.

PHP můžeme z příkazové řádky spouštět s následujícími parametry (jak se dozvíme po zadání příkazu `php -h`):

```
Usage: php [-q] [-h] [-s] [-v] [-i] [-f <file>] | {<file> [args...]}
  -q       Quiet-mode.  Suppress HTTP Header output.
  -s       Display colour syntax highlighted source.
  -f<file> Parse <file>.  Implies '-q'
  -v       Version number
  -p       Pretokenize a script (creates a .php3p file)
  -e       Execute a pretokenized (.php3p) script
  -c<path> Look for php3.ini file in this directory
  -i       PHP information
  -h       This help
```

Podívejme se na význam jednotlivých parametrů. Parametr `-q` způsobí, že PHP nebude vypisovat žádné HTTP-hlavičky. To se hodí v případech, kdy není výstup skriptu odesílán do prohlížeče. PHP totiž vždy generuje alespoň hlavičku `Content-Type`.

Pokud vyvoláme PHP s volbou `-s`, nebude se zadaný skript provádět, ale vygeneruje se HTML kód, který obsahuje zdrojový text skriptu doplněný o zvýraznění syntaxe.

Za volbou `-f` se uvádí jméno souboru, který obsahuje skript pro zpracování. Tato volba automaticky potlačí generování HTTP-hlaviček (volba `-q`). Druhou možností, jak zadat jméno souboru se skriptem, je uvést ho jako samostatný parametr.

V obou případech za jménem skriptu můžeme uvést další parametry. Ty jsou ve skriptu přístupné pomocí pole `$argv[]`, které je indexováno od 0.

Volba `-v` vypíše číslo verze PHP. Poněkud výřečnější je volba `-i`. Ta vypíše podrobné informace o PHP ve formátu HTML. Je obdobou výstupu funkce `PHPInfo()`.

Pomocí parametru `-c` můžeme určit cestu k adresáři, který obsahuje soubor `php3.ini`.

Pokud PHP vyvoláme s parametrem `-p`, je vygenerována předtokenizovaná podoba skriptu a je uložena do souboru s příponou `.php3p`. Pomocí volby `-e` pak takový skript můžeme zjistit. V současné verzi předtokenizované skripty nepřinášejí téměř žádné zvýšení výkonu.

12. Zdroje informací na Internetu

Nebýt Internetu, PHP by těžko spatřilo světlo světa. Nikoho tedy dnes nepřekvapí, že v podstatě všechny informace o PHP jsou dostupné právě na Internetu. V následující krátké kapitole naleznete odkazy na zajímavé zdroje, které souvisejí se systémem PHP. Pokud je to možné, uvadíme zde i adresy zrcadel na českých serverech (v jednom případě tu máme i slovenský server).

12.1 Kde získat PHP?

Začít musíme samozřejmě adresou serveru projektu PHP. Na tomto serveru naleznete zdrojový kód PHP, přeložené verze PHP pro Windows a některé verze Linuxu a kompletní dokumentaci v několika formátech. Kromě toho stránky obsahují odkazy na mnoho dalších zajímavých zdrojů. Na adrese je i odkaz na vývojový strom (CVS), ze kterého si můžete stáhnout úplně poslední verzi systému.

`http://www.php.net`
`http://www.php.cz`

12.2 Podpora uživatelů

Podpora uživatelů volně šířeného softwaru je realizována především pomocí různých diskusních skupin a mailing listů. Hlavním diskusním listem systému PHP je `php3@lists.php.net`. Přihlásit se do něj můžete zasláním prázdné zprávy na adresu `php3-subscribe@lists.php.net`.

Provoz na diskusním listu je velký a pohybuje se přibližně kolem 100 zpráv denně. Pokud vám tento objem zpráv připadá neúnosný, můžete si přihlásit odběr pouze výtahů z `php3`, který je zasílán dvakrát denně. K přihlášení stačí zaslat prázdnou zprávu na adresu `php3-digest-subscribe@lists.php.net`.

Pokud máte nějaký problém, je velice pravděpodobné, že se s ním setkal už někdo před vámi. Než se na něco budete ptát v diskusním listu, je slušné nejprve prohledat archiv dřívějších zpráv. Prohledávatelný archiv zpráv naleznete na adrese:

`http://www.progressive-comp.com/Lists/?l=php3-general#php3-general`

Pokud neholdujete anglickému jazyku, můžete vyzkoušet diskusní fórum s webovým rozhraním na adrese:

`http://www.pruvodce.cz/kluby/php3`

12.3 Knihovny hotových skriptů

Člověk je povahou většinou líná bytost. Pokud tedy něco dělat nemusí, tak to nedělá. Práci si můžete ušetřit tím, že použijete některé již hotové knihovny, které do PHP přidávají další funkčnost.

Asi nejkomplexnější knihovnou pro PHP je PHP Base Library (PHPLIB). S touto knihovnou jsme se stručně seznámili v desáté kapitole. Knihovna nabízí ucelenou sadu funkcí a tříd pro jednotný přístup k databázím, pro autentifikaci uživatelů, pro práci s proměnnými, jež mají platnost po dobu relace jednoho uživatele. Kromě toho knihovna obsahuje i další funkce — např. pro snadnou tvorbu formulářů a jejich zpracování. Knihovna je k dispozici na adrese:

`http://phplib.shonline.de`

Následující dva servery obsahují databáze menších a většinou jednoúčelových skriptů. Mohou se však někdy hodit — především jako inspirace pro tvorbu vlastních skriptů.

`http://php.netvision.net.il/examples/`
`http://px.sklar.com`
`http://php.codebase.org`
`http://www.phpbuilder.com`

Pro informace chtivé vývojáře je na Webu několik serverů, které přinášejí více či méně zajímavé zprávy ze světa PHP. Nejlépe, když je vyzkoušíte sami.

`http://io.incluso.com`
`http://chopshop.infonaut.net`
`http://www.xs4all.be/~mosaic/phpnewsledger/`

12.4 Webové servery

Bez nějakého pořádného webového serveru nám bude PHP téměř k ničemu. Nejpopulárnějším serverem je dnes Apache. Pro něj hovoří i to, že PHP je pro něj speciálně přizpůsobeno. Server Apache můžete získat na serveru projektu Apache.

`http://www.apache.org`
`http://sunsite.mff.cuni.cz/web/apache/`

12.5 Databázové servery

Bez databázového serveru si jen velmi těžko můžeme představit nějakou větší webovou aplikaci. Populární server MySQL můžete získat na adrese

```
http://www.tcx.se
http://mirror.opf.slu.cz/mysql/
```

Pokud dáváte přednost raději PostgreSQL, zkuste následující adresy

```
http://www.postgresql.org
http://www.sk.postgresql.org
```

Literatura

[1] Bakken, S. S. a kol: *PHP3 Manual*. PHP Documentation Group 1998
`http://www.php.net`

[2] Berners-Lee, T. – Fielding, R. – Frystyk, H.: *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945, 1996
`ftp://ftp.vse.cz/pub/docs/rfc/rfc1945.txt`

[3] Berners-Lee, T. – Masinter, L. – McCahill, M.: *Uniform Resource Locators (URL)*. RFC 1738, 1994
`ftp://ftp.vse.cz/pub/docs/rfc/rfc1738.txt`

[4] Bradley, N.: *The concise SGML companion*. Addison Wesley Longman 1997. ISBN 0-201-41999-8

[5] Bray, T. – Paoli, J. – Sperberg-McQueen, C.M.: *Extensible Markup Language (XML)*. W3C Recommendation, 1998
`http://www.w3.org/TR/1998/REC-xml-19980210`

[6] Crispin, M.: *Internet Message Access Protocol – version 4*. RFC 1730, 1994
`ftp://ftp.vse.cz/pub/docs/rfc/rfc1730.txt`

[7] *ECMAScript: A general purpose, cross-platform programming language*. Standard ECMA-262, 1997
`http://www.ecma.ch`

[8] Fielding, R.: *Relative Uniform Resource Locators*. RFC 1808, 1995
`ftp://ftp.vse.cz/pub/docs/rfc/rfc1808.txt`

[9] Fielding, R. – Gettys, J. – Mogul, J. C. – Frystyk, H. – Masinter, L. – Leach, P. – Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. HTTP Working Group, 1998

[10] Garfinkel, S. – Spafford, G.: *Bezpečnost Unixu a Internetu v praxi*. Computer Press 1998. ISBN 80-7226-082-0.

[11] Gundavaram, S.: *CGI programování*. Computer Press 1998. ISBN 80-7226-088-X

[12] Håkon, W. L. – Bos, B.: *Cascading Style Sheets, level 1*. W3C recommendation, 1996
`http://www.w3.org/TR/REC-CSS1`

[13] Kosek, J.: *HTML – tvorba dokonalých WWW stránek – podrobný průvodce*. Grada Publishing 1998. ISBN 80-7169-608-0
`http://www.kosek.cz/html/`

[14] Netscape Communications Corp.: *Persistent Client State – HTTP Cookies*. 1996
`http://home.netscape.com/newsref/std/cookie_spec.html`

[15] Ragget, D. – Hors, A. L. – Jacobs, I.: *HTML 4.0 Specification*. W3C Recommendation, 1997
`http://www.w3.org/TR/REC-html40`

[16] Satrapa, P. – Randus, J. A.: *LINUX – Internet server*. Neokortext 1998. ISBN 80-902230-3-6

[17] Schramm, K. – Burns, G. J.: *Jethro Tull – complete lyrics*. Palmyra 1993. ISBN 0-86359-977-X

[18] Šimůnek, M.: *SQL – kompletní kapesní průvodce*. Grada Publishing 1999. ISBN 80-7169-692-7

[19] Welsh, M. – Kaufman, L.: *Používáme Linux*. Computer Press 1997. ISBN 80-7226-001-4

Rejstřík

Rychlé hledání v knize vám usnadní rejstřík. Pokud je u nějakého hesla číslo stránky normálním písmem i tučně, obsahuje tučná stránka přesnou definici pojmu — například odkaz do referenčního přehledu funkcí nebo do popisu konfiguračních direktiv. Ostatní odkazy pak směřují na stránky s příklady použití.

@

$, 42
$CONTENT_LENGTH, **257**
$CONTENT_TYPE, **257**
$GATEWAY_INTERFACE, **257**
$HTTP_COOKIE_VARS, **257**
$HTTP_GET_VARS, 122, **257**
$HTTP_POST_VARS, 122, **257**
$PATH_INFO, **257**
$PATH_TRANSLATED, **257**
$PHP_AUTH_PW, 248, **257**
$PHP_AUTH_TYPE, 248, **258**
$PHP_AUTH_USER, 248, **258**
$php_errormsg, 93
$PHP_SELF, **258**
$QUERY_STRING, **258**
$REMOTE_ADDR, **258**
$REMOTE_HOST, **258**
$REQUEST_METHOD, **258**
$SCRIPT_FILENAME, **258**
$SCRIPT_NAME, 182, **258**
$SERVER_NAME, **259**
$SERVER_PORT, **259**
$SERVER_PROTOCOL, **259**
$SERVER_SOFTWARE, **259**
., 54
<<, 52
<?, 30, 39
<?php, 39
==, 53
>>, 52
?>, 30, 39
__FILE__, **259**
__LINE__, **259**

A

Abs, **259**
Accept-Language, 441
ACos, **260**
Active Server Pages, *viz* ASP
AddSlashes, 159, 176, **260**
adresa
  převod, 304
  vyhledání, 219
adresář
  čtení, 283
  vytvoření, 360
  změna, 266
  zrušení, 408
AND, 54, 143
Apache_Note, **260**
aplikace
  webová, 450
array, 45
Array, 45, 46, **261**, 275
Array_Walk, **261**
ARSort, **261**
ASin, **262**
ASort, **262**
ASP, 20
  oddělovače, 40
asp_tags, 466
ATan, **263**
autentifikace, 248, 257, 441, 442
  v PHP, 248
  ve WWW-serveru, 248
auto_append_file, 466
AUTO_INCREMENT, 370
auto_prepend_file, 454, 466
Avg, 143

B

Base64_Decode, **263**
Base64_Encode, **263**
BaseName, **263**
BCAdd, **264**
BCComp, **264**
BCDiv, **264**
bcmath.scale, 466
BCMod, **264**
BCMul, **265**
BCPow, **265**
BCScale, **265**
BCSqrt, **265**
BCSub, **266**
BEGIN, 395
bezpečnost, 18, 245–253
  cookies, 448
  generování MD5 součtu, 359
  přístupová práva, 147
  spouštění příkazů, 290
  šifrování dat, 250
  zadání hesla, 98

bezpečný režim, 253, 470
BINARY, 376
BinDec, 52, **266**
bitový
  posun, 52
break, 60, 61, 64–66
browscap, 466

C

case, 60, 61
Ceil, **266**
CERN, 17
CGI, 18
  bezpečnost, 18
  rozhraní, 18
  skript, 18
ClearStatCache, **269**
CloseDir, **270**
CloseLog, **270**
COM_Get, **270**
COM_Invoke, **271**
COM_Load, **271**
COM_Set, **271**
COMMIT, 395
Common Gateway Interface, *viz* CGI
Content-type, 187
continue, 66
cookies, 239, 445–450, 452
  bezpečnost, 448
  čtení, 257, 449
  nastavení, 409, 447
  použití, 449
  trvanlivost, 447
Copy, **272**
Cos, **272**
Count, 46, 143, **273**
CREATE TABLE, 139, 148
Crypt, 210, **273**
Current, 46, **273**

Č

čas
  formátování, 274, 308, 417
  přesný, 359
  zjištění, 308, 361, 427
číslo
  formátování, 375
  náhodné, 404
  zaokrouhlení, 408

D

DAFAULT, 148
data
  binární, 176
  šifrování, 250
databáze, 123–179
  dBase, 275
  dbm, 209, 278
  definice, 123
  komunikace, 129
  podpora v PHP, 149
  relační vztahy, 125
  spolupráce s PHP, 35
  tabulka, 123
  vytvoření, 133, 135, 136
  zpřísutpnění v prohlížeči, 156
Date, 32, **274**
datum
  kontrola, 267
dBase, 130, 275–278
dBase_Add_Record, **275**
dBase_Close, **275**
dBase_Create, **275**
dBase_Delete_Record, **276**
dBase_Get_Record, **276**
dBase_Get_Record_With_Names, **277**
dBase_NumFields, **277**
dBase_NumRecords, **277**
dBase_Open, **278**
dBase_Pack, **278**
dbList, **278**
dbm, 130, 209, 278–281
dbmClose, **278**
dbmDelete, 211, **279**
dbmExists, **279**
dbmFetch, 211, **279**
dbmFirstKey, 216, **279**
dbmInsert, 210, 211, **280**
dbmNextKey, 216, **280**
dbmOpen, 210, **281**
dbmReplace, 211, **281**
debugger, 94, 470
debugger.enabled, 470
debugger.host, 470
debugger.port, 470
Debugger_Off, **281**
Debugger_On, **281**
DecBin, 52, **282**
DecHex, **282**
DecOct, **282**
dědičnost, 82
default, 61
define, 75, 76
defined, 76
Deg2Rad, **282**
dekódování
  Base64, 263
dekrementace, 50
DELETE, 154, 169, 379, 384
DELETE FROM, 145
démon
  crond, 218
DESC, 145
DHTML, 21
diagram

  ER, 127
Die, 74, **283**
Dir, **283**
DirName, **284**
display_errors, 466
Dl, 186, **284**
do, 62, 63, 65
doc_root, 252, 466
dopis
  MIME, 229
  odeslání, 216, 229, 358, 470
  přečtení, 324, 326
  smazání, 325
  získání hlavičky, 225
  získání těla, 225
  zkopírování, 331
DoubleVal, **284**

E

Each, **284**
echo, 31, 44
Echo, **285**
else, 58, 59
elseif, 59
Empty, **285**
enable_dl, 471
End, **286**
END, 395
endfor, 33
endif, 57
engine, 466
EReg, **286**
EReg_Replace, **286**
ERegI, **287**
ERegI_Replace, **288**
error_append_string, 466
error_log, 467
Error_Log, **288**
error_prepend_string, 467
error_reporting, 467
Error_Reporting, 92, 93, **289**
escape sekvence, 43
EscapeShellCmd, **290**
eval, 75, 453
Exec, **290**
exit, 74, 283, 466
Exp, **290**
Expires, 185
Explode, **291**
extension, 471
extension_dir, 471

F

false, 53
FClose, 200, **291**
FEof, **291**
Fetch_Row, 152
FGetC, **292**
FGetS, 183, **292**
FGetSS, **292**
File, 200, **292**
File_Exists, 183, **293**
FileATime, **293**
FileCTime, **293**
FileGroup, **294**
FileINode, **294**
FileMTime, **294**
FileOwner, **294**
FilePerms, **295**
FilePro, 130
FileSize, **295**
FileType, **295**
Floor, **296**
Flush, **296**
FOpen, 183, 199, 200, **296**
for, 33, 62–64, 68, 69
FOREIGN KEY, 168
formulář, 95–122
  HTML 4.0, 116
  kódování dat, 103
  kontrola vstupu, 111
  odeslání, 97
  omezení přístupu, 122
  parametry, 34
  pole pro zadání hesla, 98
  popisky polí, 117
  použití, 34
  profesionální, 106
  přepínací tlačítko, 100
  rozdělení na části, 119
  seznam, 104, 121
  skryté pole, 104
  textové pole, 96
  tlačítko s obrázkem, 101
  víceřádkový text, 105
  vynulování, 101
  základní prvky, 96–106
  zaškrtávací pole, 99
  zpracování, 95
  zpracování dat, 467
  způsob odeslání dat, 95
FPassThru, 199, 200, **297**
FPutS, 183, 200, **297**
FRead, **298**
FrenchToJD, **298**
FSeek, 206, 207, **298**
FSockOpen, **299**
FTell, **300**
funkce, 68–74
  členská, 79
  definice, 68
  návrat, 68
  parametry, 68
  předávání parametrů, 71

FWrite, **300**

G

generátor náhodných čísel, 206
  inicializace, 206, 415
Get_Browser, **300**, 466
Get_Cfg_Var, **302**
Get_Current_User, **302**
Get_Meta_Tags, **302**
GetAllHeaders, **303**, 444
GetDate, **303**
GetEnv, **304**
GetHostByAddr, **304**
GetHostByName, **304**
GetHostByNameL, **305**
GetImageSize, **305**
GetLastMod, **306**
GetMXRR, **306**
GetMyINode, **307**
GetMyPID, **307**
GetMyUID, **307**
GetRandMax, **307**
GetType, 48, **308**
global, 70, 71
GLOBALS, 71
GMDate, **308**
GMMkTime, **308**
gpc_order, 467
Grada, 15
GRANT, 147, 148
GregorianToJD, **309**
GROUP BY, 144, 145

H

HAVING, 145
Header, 248, **309**, 444, 448
HexDec, **309**
highlight.bg, 471
highlight.comment, 471
highlight.default, 471
highlight.html, 471
highlight.keyword, 471
highlight.string, 471
HighLight_File, **310**
HighLight_String, **310**
hlasování, 239
hlášení
  chybové, 89, 93
hlavička, 440–445
  Accept, 440
  Accept-Charset, 440
  Allow, 442
  Authorization, 441
  Cache-Control, 440
  Connection, 440
  Content-Type, 187, 443
  čtení, 444
  Date, 440
  Expires, 185, 443
  From, 441
  Host, 437, 441
  If-Modified-Since, 441
  Last-Modified, 443
  Location, 165, 442
  práce v PHP, 444–445
  Pragma, 440
  Referer, 441
  Server, 442
  User-Agent, 441
  WWW-Authenticate, 442
  zaslání, 309, 444
HTML, 17
  dynamické, *viz* DHTML
  editor, 24
HTMLEntities, **310**
HTMLSpecialChars, 201, **310**
HTTP, 17, 435–445
  hlavička, *viz* hlavička
  odpověď, 436, 438–440
  požadavek, 436–438
  stavové hlášení, 438
  stavový kód, 438
  verze, 17, 435, 438
Hypertext Markup Language, *viz* HTML
Hypertext Transfer Protocol, *viz* HTTP

ch

ChDir, **266**
CheckDate, **267**
CheckDNSRR, **267**
ChGrp, **267**
ChMod, **268**
Chop, **268**
ChOwn, **268**
Chr, **269**

Ch

chyba
  hlášení, 89
  logická, 91
  ošetření, 93, 150, 153, 155
  sledování, 93
  syntaktická, 89

I

identifikátory, 41
if, 56–59, 65
ImageArc, **312**
ImageColorAllocate, **313**
ImageColorAt, **313**
ImageColorClosest, **314**
ImageColorExact, **314**
ImageColorResolve, **314**
ImageColorSet, **315**
ImageColorsForIndex, **314**
ImageColorsTotal, **315**
ImageColorTransparent, **315**, 318

ImageCopyResized, **315**
ImageCreate, 187, 188, **316**
ImageCreateFromGIF, **316**
ImageDashedLine, **316**
ImageDestroy, **317**
ImageFill, **317**
ImageFilledPolygon, **317**
ImageFilledRectangle, **317**
ImageFillToBorder, **318**
ImageFontHeight, **318**
ImageFontHeigth, 188
ImageFontWidth, 188, **318**
ImageGif, 187
ImageGIF, **318**
ImageChar, **312**
ImageCharUp, **313**
ImageInterlace, **319**
ImageLine, **319**
ImageLoadFont, **319**
ImagePolygon, **319**
ImageRectangle, **320**
ImageSetPixel, **320**
ImageString, 188, **320**
ImageStringUp, **321**
ImageSX, **321**
ImageSY, **321**
ImageTTFBBox, **321**
ImageTTFText, **322**
IMAP, 223, 323–338
IMAP_8Bit, **323**
IMAP_Append, **323**
IMAP_Base64, **323**
IMAP_Binary, **324**
IMAP_Body, 225, 228, **324**
IMAP_ClearFlag_Full, **325**
IMAP_Close, 226, **325**
IMAP_CreateMailBox, **325**
IMAP_Delete, **325**
IMAP_DeleteMailBox, **326**
IMAP_Expunge, **326**
IMAP_FetchBody, 228, **326**
IMAP_FetchHeader, 225, **327**
IMAP_FetchStructure, 228, **327**
IMAP_Header, **329**
IMAP_Headers, 226, **330**
IMAP_Check, **324**
IMAP_ListMailBox, **330**
IMAP_ListSubscribed, **331**
IMAP_Mail_Compose, 229, 230
IMAP_Mail_Copy, **331**
IMAP_Mail_Move, **331**
IMAP_MailBoxMsgInfo, **332**
IMAP_Num_Msg, 225, **332**
IMAP_Num_Recent, **332**
IMAP_Open, 224, 226, **333**
IMAP_Ping, **333**
IMAP_QPrint, **334**
IMAP_RenameMailBox, **334**
IMAP_ReOpen, **334**
IMAP_RFC822_Parse_AdrList, **335**
IMAP_RFC822_Write_Address, **335**
IMAP_ScanMailBox, **335**
IMAP_SetFlag_Full, **336**
IMAP_Sort, **336**
IMAP_Subscribe, **337**
IMAP_UID, **337**
IMAP_Undelete, **337**
IMAP_Unsubscribe, **337**
Implode, 242, **338**
include, 66, 68, 253, 433, 466, 468
include_path, 67, 184, 468
inkrementace, 50
INSERT, 148, 154, 165, 176, 379, 384
INSERT INTO, 141
instalace, 461–465
  Apache, 463
  IIS4, 462
  PWS a IIS3, 462
  Unix, 463–465
  Windows, 461–463
InterBase, 132
IntVal, **338**
Is_Array, 48, **339**
Is_Dir, **339**
Is_Double, 48, **339**
Is_Executable, **340**
Is_File, **340**
Is_Float, **340**
Is_Int, **340**
Is_Integer, 48, **341**
Is_Link, **341**
Is_Long, **341**
Is_Object, 48, **342**
Is_Readable, **342**
Is_Real, **342**
Is_String, 48, **343**
Is_Writeable, **343**
ISAPI, 24
IsSet, **343**

J

Java, 19
Java-aplet, 19
JavaScript, 19
jazyk
  Java, 19
  JavaScript, 19, 20, 111
  JScript, 20
  Perl, 20
  PHP, 39–88
  PHP/FI, 22
  Python, 20
  REXX, 20

VBScript, 20
JDDayOfWeek, **344**
JDMonthName, **344**
JDToFrench, **344**
JDToGregorian, **345**
JDToJewish, **345**
JDToJulian, **345**
Jethro Tull, 16
JewishToJD, **345**
Join, **346**
JulianToJD, **346**

K

Key, 46, **346**
klávesa
  horká, 117
klíč
  cizí, 125
  primární, 124, 140
kniha hostů, 199
knihovna
  načtení, 186, 216, 284
Knuth, D. E., 15
kód
  ASCII, 44
kódování
  Base64, 263
komentáře, 41
konfigurace, 465–473
konstanta, 75
  v osmičkové soustavě, 43
  v šestnáctkové soustavě, 43
konstruktor, 81
konvence
  typografické, 16
KSort, **347**

L

LDAP, 219, 347–356
  hledání, 222
  připojení, 222
LDAP_Add, **347**
LDAP_Bind, 222, **348**
LDAP_Close, 223, **348**
LDAP_Connect, 222, **349**
LDAP_Count_Entries, **349**
LDAP_Delete, **349**
LDAP_DN2UFN, **349**
LDAP_Explode_DN, **350**
LDAP_First_Attribute, **350**
LDAP_First_Entry, 223, **350**
LDAP_Free_Result, **351**
LDAP_Get_Attributes, 223, **351**
LDAP_Get_DN, **351**
LDAP_Get_Entries, **352**
LDAP_Get_Values, **352**
LDAP_List, **353**
LDAP_Modify, **353**
LDAP_Next_Attribute, **354**
LDAP_Next_Entry, 223, **354**
LDAP_Read, **354**
LDAP_Search, 223, **355**
LDAP_UnBind, **356**
Leak, **356**
Lerdorf, R., 22
Lightweight Directory Access Protocol, *viz* LDAP
LIKE, 143
Link, **356**
LinkInfo, **356**
Linux, 20
list, 47, 276
List, **357**
LiveWire, 20
Log, **357**
Log10, **357**
log_erros, 468
lokalizace
  nastavení, 410
LONGVARBINARY, 376, 383
LONGVARCHAR, 383
LStat, **357**
LTrim, **358**

M

M_PI, **358**
magic_quotes_gpc, 122, 468
magic_quotes_runtime, 468
magic_quotes_sybase, 122, 468
Mail, 216, 217, 229, 230, **358**, 470
Max, **359**
max_execution_time, 468
MD5, 210, **359**, 452
memory_limit, 468
metoda
  CONNECT, 437
  DELETE, 437
  GET, 95, 437
  OPTIONS, 437
  POST, 95
  PUT, 437
  TRACE, 437
MicroTime, 206, **359**, 415
Min, **360**
MkDir, **360**
MkTime, **361**
mod_ssl, 250
model
  datový, 127
model dat
  relační, 123
MS SQL Server, 132, 136–137
  vytvoření databáze, 136
MySQL, 132–135, 361–374
  klient, 133

nastavení přístupových práv, 147
ODBC ovladače, 134
spolupráce s PHP, 149
vytvoření databáze, 133
mysql.allow_persistent, 473
mysql.max_links, 473
mysql.max_persistent, 473
MySQL_Affected_Rows, **361**
MySQL_Close, 150, **361**
MySQL_Connect, 149, 150, 179, **362**
MySQL_Create_DB, **362**
MySQL_Data_Seek, **363**
MySQL_DB_Query, **364**
MySQL_DBName, **363**
MySQL_Drop_DB, **364**
MySQL_ErrNo, **365**
MySQL_Error, **365**
MySQL_Fetch_Array, 150, **365**
MySQL_Fetch_Field, **366**
MySQL_Fetch_Lengths, **367**
MySQL_Fetch_Object, **367**
MySQL_Fetch_Row, **368**
MySQL_Field_Flags, **369**
MySQL_Field_Len, **369**
MySQL_Field_Name, **368**
MySQL_Field_Seek, **368**
MySQL_Field_Table, **369**
MySQL_Field_Type, **369**
MySQL_Free_Result, **370**
MySQL_Insert_Id, **370**
MySQL_List_DBs, **370**
MySQL_List_Fields, **370**
MySQL_List_Tables, **371**
MySQL_Num_Fields, **371**
MySQL_Num_Rows, 150, **371**
MySQL_PConnect, 179, **372**
MySQL_Query, 149, 150, **372**
MySQL_Result, **373**
MySQL_Select_DB, 149, **373**
MySQL_TableName, **374**

N

negace, 54
nerovnost
neostrá, 53
ostrá, 53
new, 81
Next, 46, **374**
NL2BR, 106, 201, 209, **375**
nonekvivalence, 51
NOT, 143
NOT NULL, 140
Number_Format, **375**

O

objekt, 77–84
COM, 270
členská proměnná, 78
konstruktor, 81
objektově orientované programování, *viz* OOP
obrázek
generování, 312–323
jako počítadlo přístupů, 186
kopírování, 315
na tlačítku, 101
velikost, 305
vytvoření, 187, 316
zapsání, 318
OctDec, **376**
ODBC, 129, 134, 137, 376–386
spolupráce s PHP, 152
ODBC_AutoCommit, 175, **376**
ODBC_BinMode, 177, **376**, 472
ODBC_Close, 153, **377**
ODBC_Close_All, **377**
ODBC_Commit, 175, 377, **378**
ODBC_Connect, 37, 152, 153, 179, **378**, 472
ODBC_Cursor, **378**
ODBC_Do, **379**
ODBC_Exec, 37, 152, 153, 175, **379**
ODBC_Execute, **379**, 384
ODBC_Fetch_Into, **380**
ODBC_Fetch_Row, 37, **380**
ODBC_Field_Name, **381**
ODBC_Field_Num, **382**
ODBC_Field_Type, **382**
ODBC_Free_Result, **382**
ODBC_LongReadLen, 177, **383**, 472
ODBC_Num_Fields, **383**
ODBC_Num_Rows, 37, 154, **384**
ODBC_PConnect, 179, **383**, 472
ODBC_Prepare, **384**
ODBC_Result, 37, 152, 177, 178, **385**
ODBC_Result_All, **386**
ODBC_RollBack, 175, 377, **386**
OOP, 77
Open Source Software, 15
open_basedir, 467
OpenDir, **386**
OpenLog, **387**
operátor
bitový, 51
logický, 53
matematický, 50
nerovnosti, 53
podmíněný, 55
priorita, 55
přiřazení, 91
relační, 53
rovnosti, 53, 91
ternární, 55
unární, 52
OR, 54, 143

Ord, **388**
ORDER BY, 145

P

pamě
  vyrovnávací
    vypnutí, 157
parametr
  předávání, 68, 71
  předávání odkazem, 71
  z příkazové řádky, 474
Parse_Str, **388**
Parse_URL, **388**
PassThru, **389**
PClose, **389**
Pecinovský, R., 15
Pg_Close, 154, **390**
Pg_CmdTuples, **390**
Pg_Connect, 154, 155, 179, **390**
Pg_DBName, **391**
Pg_ErrorMessage, **391**
Pg_Exec, 154, 155, **391**
Pg_Fetch_Array, 154, **391**
Pg_Fetch_Object, **392**
Pg_Fetch_Row, **392**
Pg_FieldIsNull, **392**
Pg_FieldName, **393**
Pg_FieldNum, **393**
Pg_FieldPrtLen, **393**
Pg_FieldSize, **393**
Pg_FieldType, **393**
Pg_FreeResult, **394**
Pg_GetLastOID, **394**
Pg_Host, **394**
Pg_LOClose, **394**
Pg_LOCreate, **395**
Pg_LOOpen, **395**
Pg_LORead, **396**
Pg_LOReadAll, **396**
Pg_LOUnLink, **396**
Pg_LOWrite, **396**
Pg_NumFields, **397**
Pg_NumRows, 154, **397**
Pg_Options, **397**
Pg_PConnect, 179, **397**
Pg_Port, **398**
Pg_Result, **398**
Pg_tty, **399**
pgsql.allow_persistent, 473
pgsql.max_links, 473
pgsql.max_persistent, 473
PHP, 13–479
  CGI-skript, 251
  historie, 22
  modul, 251
  verze 3.0, 22
  zobrazení verze, 28
PHP_OS, **399**
PHP_VERSION, **399**
PHPInfo, 30, **399**, 474
PHPLIB, 450–459
PHPVersion, **399**
Pi, **400**
PI, 39
písmena
  velikost, 41
písmo
  velikost, 187
platforma
  podporovaná, 23
počítadlo přístupů, 181–198
podmínka, 56
pole, 45–47
  asociativní, 45
  dvojrozměrné, 47
  funkce, 46
  index, 45
  inicializace, 45
  načtení do proměnných, 357
  počet prvků, 46
  prvek, 45
  setřídění, 261, 262, 347, 408, 413, 427, 428, 431
  vícerozměrné, 47
  vytvoření, 261
POpen, **400**
port, 435
Pos, **400**
Post Office Protocol, *viz* protokol, POP
PostgreSQL, 132, 135, 390–399
  klient, 135
  spolupráce s PHP, 154
  vytvoření databáze, 135
postinkriminace, 50
pošta
  čtení, 223
Pow, **401**
precision, 468
preinkrementace, 50
Prev, **401**
PRIMARY KEY, 140
Print, **401**
Printf, **402**
Processing instructions, *viz* PI
program
  spuštění, 290, 426
prohlížeč
  detekce, 300, 466
  identifikace, 452
  přesměrování, 165
proměnná, 42–49
  členská, 78
  deklarace, 42, 92

globální, 70
inicializace, 42, 78, 92
název, 41
počet prvků, 273
prostředí, 304, 404
přetypování, 47–49
statická, 73
stavová, 454
typ, 48
zjištění typu, 308
změna typu, 48
protokol
HTTP, *viz* HTTP
IMAP, 223
připojení k serveru, 333
LDAP, 347
podpora v PHP, 219
POP, 223
SNMP, 412
verze, 438
příkaz
cyklu, 62–64
nekonečný, 64
oddělování, 40
skládání, 56
větvení, 56–62
přiřazení, 49
`PutEnv`, **404**

Q

`QUERY_STRING`, 204
`QuoteMeta`, **404**

R

`Rad2Deg`, **404**
`Rand`, **404**
`Rand()`, 49
`RawURLDecode`, **405**
`RawURLEncode`, **405**
`ReadDir`, **406**
`ReadFile`, **406**
`ReadLink`, **406**
`REFERENCES`, 168
`Register_ShutDown_Function`, **407**
rekurze, 74
`Rename`, **407**
`require`, 66–68, 182, 209, 253, 433, 468
`Reset`, 46, **407**
`return`, 68, 69, 74
`REVOKE`, 147
`Rewind`, 183, **407**
`RewindDir`, **408**
režim
bezpečný, 253
`RmDir`, **408**
`Round`, **408**
rozhraní
CGI, 18
ODBC, 129
`RSort`, **408**
`RTrim`, **409**

Ř

řetězec
část, 424
délka, 419
porovnání, 416, 417
spojování, 54
tisk, 401, 402
výpis, 188

S

`safe_mode`, 470
`safe_mode_exec_dir`, 253, 470
Secure Sockets Layer, *viz* SSL
`SELECT`, 142, 145, 150, 154, 159, 385
`sendmail_from`, 470
`sendmail_path`, 470
`serialize`, 453
server
databázový, 25, 129
výběr, 132
získání, 477
LDAP, 219
virtuální, 437
WWW
získání, 476
Server Side Includes, *viz* SSI
Server Side JavaScript, *viz* SSJS
servlet, 18
`Set_Socket_Blocking`, **410**
`Set_Time_Limit`, 217, **411**
`SetCookie`, **409**, 448
`SetLocale`, **410**
`SetType`, 48, **411**
seznam odkazů, 231
SGML, 39
`short_open_tag`, 469
schéma
https, 251
`Sin`, **411**
`SizeOf`, **412**
skript
automatické spouštění, 217
knihovny funkcí, 476
komentáře, 41
ladění, 94
maximální doba běhu, 217, 411, 468
načtení, 66
oddělení od HTML, 30
oddělení od XML, 39
ochrana zdrojového kódu, 245
pozastavení, 412, 431
první, 27
ukončení, 74
umístění, 246

zvýraznění syntaxe, 310
Sleep, **412**
služba
Schedule, 217
SMTP, 470
SNMPGet, **412**
SNMPWalk, **413**
socket
otevření, 299
Sort, **413**
soubor
binární, 206
čtení, 183, 292, 406
binární, 298
jméno, 263
nastavení pozice, 206, 298
odeslání z formuláře, 103
otevření, 296
přejmenování, 407
přípona, 27
smazání, 429
textový, 205
upload, 103, 469
velikost, 295
vložení, 199
vypsání, 199
zápis, 183, 200, 297, 300
zavření, 291
zkopírování, 272
součet
logický, 51, 54
součin
logický, 51, 54
Soundex, **413**
Split, **414**
spojení
persistentní, 179
spojky
logické, 54
SPrintf, **414**
SQL, 129, 137–148
PL/SQL, 18
SQL Enterprise Manager, 136
SQL_RegCase, **415**
Sqrt, **415**
SRand, 206, **415**
SŘBD, 123
SSI, 19
SSJS, 20
SSL, 250, 448
pro Apache, 250
SSLeay, 250
Stat, **416**
stavový kód
generování, 445
stránka
platnost, 443
přesměrování, 165
StrCaseCmp, **416**
StrCmp, **417**
StrCSpn, **417**
StrFTime, **417**
StrChr, **416**
StripSlashes, 122, **419**
StrLen, **419**
Stronghold, 250
StrPos, **419**
StrRev, **420**
StrRChr, **420**
StrRPos, **420**
StrSpn, **421**
StrStr, **421**
StrTok, **422**
StrToLower, **423**
StrToUpper, **423**
StrTr, 183, **424**
Structured Query Language, *viz* SQL
StrVal, **424**
SubStr, 165, **424**
switch, 56, 60, 61
Sybase, 132
SymLink, **425**
SysLog, **425**
System, **426**
systém řízení báze dat, *viz* SŘBD

Š
šifrování
DES, 273

T
tabulka
atribut, 123
databázová, 123
položka, 123
propojení, 143
přístupová práva, 147
vytvoření, 139, 191, 231, 239
vztah, 125
záznam, 123
Tan, **426**
TempNam, **426**
TEX, 20
TEXTAREA, 105
Time, **427**, 448
Touch, **427**
track_errors, 469
track_vars, 469
transakce, 174
Trim, **427**
true, 53
třída
dědičnost, 82
definice, 77

Třísková, L. M. A., 15
typ
  array, 42
  atributu, 124, 139
  automatické rozpoznání, 42
  double, 42, 43
  integer, 42–43
  konverze, 48
  nastavení, 411
  object, 42
  pole, 45–47
  string, 42–44
  zjištění, 48
  změna, 48

U

UASort, **427**
UCFirst, **428**
UCWords, **428**
UKSort, **428**
UMask, **429**
Uniform Resource Locator, *viz* URL
UniqID, **432**, 452
UnLink, **429**
unserialize, 453
UnSet, 222, **429**
uodbc.allow_persistent, 472
uodbc.default_db, 472
uodbc.default_pw, 472
uodbc.default_user, 472
uodbc.defaultbinmode, 472
uodbc.defaultlrl, 472
uodbc.max_links, 472
uodbc.max_persistent, 472
UPDATE, 146, 154, 194, 379, 384
upload_max_filesize, 469
upload_tmp_dir, 469
URL, 17
URLDecode, **430**
URLEncode, **430**
user_dir, 252, 469
uSleep, **431**
USort, **431**
uživatel
  autentifikace, 248
  podpora, 475

V

var, 78
VARBINARY, 376
Virtual, **433**
výraz, 49–56
  logický, 53
  regulární, 84–88, 286–288
  výpis, 285
vztah
  1:1, 125
  1:N, 125
  M:N, 125
  rozložení, 127

W

warn_plus_overloading, 469
WHERE, 143, 158
while, 62, 63, 65
World-Wide Web, 17–21

X

XOR, 54

Y

y2k_compliance, 469

Z

záznam, 123
  identifikace, 124
  modifikace, 146, 169
  počet, 150
  přidání, 141
  řazení, 233
  smazání, 145, 166
  výběr, 142
zdroj
  datový, 134, 137
    připojení, 152
zpracování
  transakční, 174

Tematický přehled funkcí

Echo — Vypsání jednoho nebo více výrazů ... 285
Print — Vytiskne obsah řetězce ... 401

Funkce pracující pouze na serveru Apache

Apache_Note — Nastavení/přečtení poznámky požadavku ... 260
GetAllHeaders — Přečtení všech HTTP-hlaviček požadavku ... 303
Virtual — Provedení požadavku obslouženého Apachem ... 433

Funkce pro práci s adresáři

CloseDir — Uzavření adresáře otevřeného pro čtení ... 270
Dir — Pseudotřída pro práci s adresáři ... 283
ChDir — Nastavení aktuálního adresáře ... 266
OpenDir — Otevření adresáře ... 386
ReadDir — Funkce přečte název jednoho souboru z adresáře ... 406
RewindDir — Přesun na první položku otevřeného adresáře ... 408

Funkce pro práci s COM-objekty

COM_Get — Zjištění hodnoty vlastnosti COM-objektu ... 270
COM_Invoke — Vyvolání metody COM-objektu ... 271
COM_Load — Vytvoření instance COM-objektu ... 271
COM_Set — Nastavení vlastnosti COM-objektu ... 271

Funkce pro práci s databází dbm

dbList — Informace o použitém druhu knihovny dbm ... 278
dbmClose — Zavření databáze ... 278
dbmDelete — Smazání hodnoty uložené pod daným klíčem ... 279
dbmExists — Zjištění, zda pro daný klíč existuje v databázi hodnota ... 279
dbmFetch — Získání hodnoty uložené v databázi pod nějakým klíčem ... 279
dbmFirstKey — Funkce vrací hodnotu prvního klíče v databázi ... 279
dbmInsert — Vložení hodnoty do databáze ... 280
dbmNextKey — Zjištění hodnoty následujícího klíče ... 280
dbmOpen — Otevření databáze ... 281
dbmReplace — Nahrazení hodnoty pro daný klíč v databázi ... 281

Funkce pro práci s databází MySQL

MySQL_Affected_Rows — Počet záznamů ovlivněných posledním příkazem 361
MySQL_Close — Uzavření spojení s databází MySQL 361
MySQL_Connect — Vytvoření spojení s databázovým serverem 362
MySQL_Create_DB — Vytvoření nové databáze 362
MySQL_Data_Seek — Přesun ukazatele na aktuální záznam 363
MySQL_DB_Query — Vykonání SQL-příkazu 364
MySQL_DBName — Přečtení jména databáze 363
MySQL_Drop_DB — Smazání databáze 364
MySQL_ErrNo — Chybový kód posledního volání MySQL 365
MySQL_Error — Text chybového hlášení posledního volání MySQL 365
MySQL_Fetch_Array — Načte záznam výsledku do asociativního pole 365
MySQL_Fetch_Field — Získání informací o položce výsledku 366
MySQL_Fetch_Lengths — Zjištění délek položek aktuálního záznamu výsledku 367
MySQL_Fetch_Object — Načte záznam výsledku do objektu 367
MySQL_Fetch_Row — Načte záznam výsledku do pole 368
MySQL_Field_Flags — Zjištění doplňkových informací o položce 369
MySQL_Field_Len — Zjištění délky položky 369
MySQL_Field_Name — Zjištění názvu položky 368
MySQL_Field_Seek — Nastavení aktuálního indexu položky 368
MySQL_Field_Table — Zjištění tabulky, ze které pochází položka 369
MySQL_Field_Type — Zjištění typu položky 369
MySQL_Free_Result — Uvolnění výsledku z paměti 370
MySQL_Insert_Id — Zjištění hodnoty ID posledního příkazu INSERT 370
MySQL_List_DBs — Zjištění všech databází dostupných na serveru 370
MySQL_List_Fields — Získání výsledku s obsahem položek zadané tabulky 370
MySQL_List_Tables — Zjištění všech tabulek uložených v databázi 371
MySQL_Num_Fields — Zjistí počet položek výsledku 371
MySQL_Num_Rows — Zjistí počet záznamů výsledku 371
MySQL_PConnect — Vytvoří persistentní spojení s databázovým serverem 372
MySQL_Query — Vykonání SQL-příkazu 372
MySQL_Result — Získání hodnoty jedné položky výsledku dotazu 373
MySQL_Select_DB — Výběr aktivní databáze 373
MySQL_TableName — Přečtení jména tabulky 374

Funkce pro práci s databází PostgreSQL

Pg_Close — Uzavření spojení se serverem 390
Pg_CmdTuples — Vrací počet záznamů ovlivněných posledním příkazem 390
Pg_Connect — Připojení k databázi PostgreSQL 390
Pg_DBName — Zjištění jména databáze, ke které jsme připojeni 391
Pg_ErrorMessage — Zjištění chybového hlášení 391
Pg_Exec — Provedení SQL-příkazu 391
Pg_Fetch_Array — Načtení záznamu do asociativního pole 391
Pg_Fetch_Object — Načte záznam výsledku do objektu 392
Pg_Fetch_Row — Načte záznam výsledku do pole 392
Pg_FieldIsNull — Test, zda je položka NULL 392
Pg_FieldName — Zjištění jména položky 393
Pg_FieldNum — Zjištění čísla položky 393
Pg_FieldPrtLen — Zjištění délky položky ve znacích 393
Pg_FieldSize — Zjištění velikosti místa potřebného pro uložení položky 393
Pg_FieldType — Zjištění typu položky 393
Pg_FreeResult — Uvolnění výsledku z paměti 394
Pg_GetLastOID — Zjištění OID posledně vloženého záznamu 394
Pg_Host — Zjistí počítač, na kterém běží PostgreSQL 394

`Pg_LOClose` — Zavření velkého objektu ... 394
`Pg_LOCreate` — Vytvoření velkého objektu ... 395
`Pg_LOOpen` — Otevření velkého objektu ... 395
`Pg_LORead` — Čtení dat z velkého objektu ... 396
`Pg_LOReadAll` — Přečtení všech dat z velkého objektu ... 396
`Pg_LOUnLink` — Smazání velkého objektu ... 396
`Pg_LOWrite` — Zápis dat do velkého objektu ... 396
`Pg_NumFields` — Zjistí počet položek výsledku ... 397
`Pg_NumRows` — Zjistí počet záznamů výsledku ... 397
`Pg_Options` — Funkce vrací volby nastavené pro spojení ... 397
`Pg_PConnect` — Vytvoření persistentního spojení s databází ... 397
`Pg_Port` — Zjistí port, na kterém běží PostgreSQL ... 398
`Pg_Result` — Přečtení jedné položky výsledku ... 398
`Pg_tty` — Zjistí jméno zařízení tty ... 399

Funkce pro práci s datem a časem

`Date` — Formátování časových údajů ... 274
`GetDate` — Zjištění časových informací ... 303
`GMDate` — Vrací zformátovaný údaj o Greenwichském čase ... 308
`GMMkTime` — Získání časového údaje z Greenwichského času ... 308
`CheckDate` — Kontrola správnosti data ... 267
`MicroTime` — Zjištění aktuálního časového údaje s přesností na mikrosekundy ... 359
`MkTime` — Získání časového údaje ... 361
`StrFTime` — Formátování časových údajů ... 417
`Time` — Zjištění aktuálního časového údaje ... 427

Funkce pro práci s datovými zdroji ODBC

`ODBC_AutoCommit` — Nastavení automatického potvrzování transakcí ... 376
`ODBC_BinMode` — Nastavení režimu práce s binárními položkami ... 376
`ODBC_Close` — Uzavření spojení s ODBC datovým zdrojem ... 377
`ODBC_Close_All` — Uzavření všech ODBC spojení ... 377
`ODBC_Commit` — Potvrzení transakce ... 378
`ODBC_Connect` — Vytvoří spojení s datovým zdrojem ... 378
`ODBC_Cursor` — Zjištění jména kurzoru pro výsledek ... 378
`ODBC_Do` — Vykonání SQL-příkazu ... 379
`ODBC_Exec` — Vykonání SQL-příkazu ... 379
`ODBC_Execute` — Vykonání předzpracovaného SQL-příkazu ... 379
`ODBC_Fetch_Into` — Načtení záznamu do pole ... 380
`ODBC_Fetch_Row` — Načtení záznamu výsledku ... 380
`ODBC_Field_Name` — Zjištění jména položky ... 381
`ODBC_Field_Num` — Zjištění čísla položky ... 382
`ODBC_Field_Type` — Zjištění typu položky ... 382
`ODBC_Free_Result` — Uvolnění výsledku z paměti ... 382
`ODBC_LongReadLen` — Nastaví maximální počet bajtů čtených z dlouhých položek ... 383
`ODBC_Num_Fields` — Zjistí počet položek ve výsledku ... 383
`ODBC_Num_Rows` — Vrací počet záznamů výsledku ... 384
`ODBC_PConnect` — Vytvoří persistentní spojení s datovým zdrojem ... 383
`ODBC_Prepare` — Funkce připraví SQL-příkaz pro opakované provedení ... 384
`ODBC_Result` — Získání jedné položky výsledku ... 385
`ODBC_Result_All` — Vypsání celého výsledku dotazu ve formě HTML-tabulky ... 386
`ODBC_RollBack` — Zrušení rozpracované transakce ... 386

Funkce pro práci s daty různých kalendářů

FrenchToJD — Převod data francouzského republikového kalendáře na juliánské datum . . 298
GregorianToJD — Převod data gregoriánského kalendáře na juliánské datum . . . 309
JDDayOfWeek — Získání jména dne v týdnu . . . 344
JDMonthName — Získání jména měsíce . . . 344
JDToFrench — Převod juliánského data na den ve francouzském republikovém kalendáři . 344
JDToGregorian — Převod juliánského data na den v gregoriánském kalendáři . . . 345
JDToJewish — Převod juliánského data na den v židovském kalendáři . . . 345
JDToJulian — Převod juliánského data na den v juliánském kalendáři . . . 345
JewishToJD — Převod data židovského kalendáře na juliánské datum . . . 345
JulianToJD — Převod data juliánského kalendáře na juliánské datum . . . 346

Funkce pro práci s elektronickou poštou

Mail — Odeslání e-mailu . . . 358

Funkce pro práci s obrázky

GetImageSize — Zjištění velikosti obrázku GIF, JPEG nebo PNG . . . 305
ImageArc — Nakreslení části elipsy . . . 312
ImageColorAllocate — Alokování a vytvoření barvy pro obrázek . . . 313
ImageColorAt — Zjištění indexu barvy daného bodu . . . 313
ImageColorClosest — Získání indexu barvy, která je nejbližší zadané barvě . . . 314
ImageColorExact — Zjištění indexu zadané barvy . . . 314
ImageColorResolve — Zjištění indexu zadané barvy nebo nejbližší barvy . . . 314
ImageColorSet — Nastavení položky palety na určitou barvu . . . 315
ImageColorsForIndex — Zjištění barevných složek dané barvy . . . 314
ImageColorsTotal — Zjištění počtu barev v obrázku . . . 315
ImageColorTransparent — Nastavení transparentní (průhledné) barvy obrázku . . . 315
ImageCopyResized — Kopírování části obrázku se změnou velikosti . . . 315
ImageCreate — Vytvoření obrázku . . . 316
ImageCreateFromGIF — Vytvoření obrázku podle obrázku ze souboru nebo z určitého URL 316
ImageDashedLine — Nakreslení čárkované čáry . . . 316
ImageDestroy — Uvolnění obrázku z paměti . . . 317
ImageFill — Vyplnění oblasti . . . 317
ImageFilledPolygon — Nakreslení polygonu vyplněného barvou . . . 317
ImageFilledRectangle — Nakreslení obdélníku vyplněného barvou . . . 317
ImageFillToBorder — Vyplnění oblasti jejíž hranice je dána barvou . . . 318
ImageFontHeight — Zjištění velikosti písma v bodech . . . 318
ImageFontWidth — Zjištění šířky písma v bodech . . . 318
ImageGIF — Zapsání obrázku na výstup nebo do souboru . . . 318
ImageChar — Nakreslení znaku . . . 312
ImageCharUp — Nakreslení znaku ve vertikálním směru . . . 313
ImageInterlace — Zapnutí/vypnutí prokládání obrázku . . . 319
ImageLine — Nakreslení čáry . . . 319
ImageLoadFont — Nahrání nového fontu ze souboru . . . 319
ImagePolygon — Nakreslení polygonu . . . 319
ImageRectangle — Nakreslení obdélníku . . . 320
ImageSetPixel — Nakreslení jednoho bodu . . . 320
ImageString — Vypsání textového řetězce . . . 320
ImageStringUp — Vypsání textového řetězce ve vertikálním směru . . . 321
ImageSX — Zjištění šířky obrázku . . . 321
ImageSY — Zjištění výšky obrázku . . . 321
ImageTTFBBox — Zjištění plochy, kterou zabere text zobrazený TrueType fontem . . . 321
ImageTTFText — Zobrazení textu pomocí TrueType písma . . . 322

Funkce pro práci s poli

`Array` — Vytvoření pole ... 261
`Array_Walk` — Na všechny prvky pole aplikuje zadanou funkci ... 261
`ARSort` — Sestupně setřídí pole a zachová indexy prvků ... 261
`ASort` — Setřídí pole a zachová indexy prvků ... 262
`Count` — Zjištění počtu prvků proměnné ... 273
`Current` — Vrací hodnotu aktuálního prvku pole ... 273
`Each` — Vrátí hodnotu indexu a obsah prvku pole a přesune ukazatel na další prvek pole 284
`End` — Nastaví ukazatel na konec pole ... 286
`Key` — Zjistí index prvku pole, na který je nastaven ukazatel ... 346
`KSort` — Setřídí pole podle obsahu indexů a zachová indexy prvků ... 347
`List` — Přiřadí do proměnných prvky pole ... 357
`Next` — Vrací hodnotu následujícího prvku pole ... 374
`Pos` — Vrací hodnotu aktuálního prvku pole ... 400
`Prev` — Vrací hodnotu předchozího prvku pole ... 401
`Reset` — Nastaví ukazatel na začátek pole ... 407
`RSort` — Sestupně setřídí pole ... 408
`SizeOf` — Zjištění počtu prvků proměnné ... 412
`Sort` — Setřídí pole ... 413
`UASort` — Setřídí pole na základě uživatelem definované funkce pro porovnání a zachová indexy prvků ... 427
`UKSort` — Setřídí pole podle obsahu indexů na základě uživatelem definované funkce pro porovnání a zachová indexy prvků ... 428
`USort` — Setřídí pole na základě uživatelem zadané funkce pro porovnávání prvků pole .. 431

Funkce pro práci s protokolem HTTP

`Header` — Zaslání HTTP hlavičky ... 309
`SetCookie` — Zaslání cookie klientovi ... 409

Funkce pro práci s regulárními výrazy

`EReg` — Zjistí, zda řetězec vyhovuje regulárnímu výrazu ... 286
`EReg_Replace` — Nahrazení řetězce podle regulárního výrazu ... 286
`ERegI` — Zjistí, zda řetězec vyhovuje regulárnímu výrazu. Při porovnávání se v úvahu nebere velikost písmen ... 287
`ERegI_Replace` — Nahrazení řetězce podle regulárního výrazu, ve kterém se ignoruje velikost písmen ... 288
`Split` — Rozdělí řetězec na části a uloží je do pole ... 414
`SQL_RegCase` — Vytvoří regulární výraz pro hledání řetězce bez závislosti na velikosti písmen ... 415

Funkce pro práci s textovými řetězci

AddSlashes — Doplnění zpětných lomítek před citlivé znaky ... 260
Crypt — Zašifruje text pomocí algoritmu DES ... 273
Explode — Rozdělí řetězec na části ... 291
Flush — Vyprázdnění výstupního bufferu ... 296
Get_Meta_Tags — Zjištění obsahu META-tagů v HTML souboru ... 302
HTMLEntities — Převod všech možných znaků na znakové entity HTML ... 310
HTMLSpecialChars — Převod speciálních znaků na znakové entity HTML ... 310
Chop — Odstranění mezer a tabulátorů z konce řetězce ... 268
Chr — Vrací znak s určitým ASCII-kódem ... 269
Implode — Spojí prvky pole zadaným textem do jednoho řetězce ... 338
Join — Spojí prvky pole zadaným textem do jednoho řetězce ... 346
LTrim — Odstraní mezery ze začátku řetězce ... 358
MD5 — Spočítá hodnotu hashovací funkce MD5 pro zadaný text ... 359
NL2BR — Převádí konce řádků v řetězci na tag
 ... 375
Ord — Vrací ASCII-kód prvního znaku v řetězci ... 388
Parse_Str — Analyzuje obsah řetězce a uloží jej do proměnných ... 388
Printf — Vytiskne zformátovaný řetězec ... 402
QuoteMeta — Nahradí metaznaky escape sekvencí ... 404
RawURLDecode — Rozkóduje řetězec zakódovaný jako URL ... 405
RawURLEncode — Zakóduje řetězec tak, aby byl použitelný v URL ... 405
RTrim — Odstraní mezery z konce řetězce ... 409
SetLocale — Nastavení podpory národních prostředí ... 410
Soundex — Vrátí hodnotu klíče Soundex pro zadaný text ... 413
SPrintf — Uloží zformátovaný řetězec do proměnné ... 414
StrCaseCmp — Porovnání řetězců bez ohledu na malá a velká písmena ... 416
StrCmp — Porovnání řetězců ... 417
StrCSpn — Vrací index prvního znaku řetězce, který je prvkem množiny znaků ... 417
StrChr — Nalezení prvního výskytu znaku v řetězci ... 416
StripSlashes — Odstranění lomítek umístěných před citlivé znaky funkcí AddSlashes() 419
StrLen — Zjistí délku řetězce ... 419
StrPos — Nalezení podřetězce v řetězci ... 419
StrRev — Obrátí text v řetězci ... 420
StrRChr — Nalezení posledního výskytu znaku v řetězci ... 420
StrRPos — Nalezení posledního výskytu znaku v řetězci ... 420
StrSpn — Vrací počet znaků ze začátku řetězce, které vyhovují množině znaků ... 421
StrStr — Nalezení prvního výskytu textu v řetězci ... 421
StrTok — Rozložení řetězce na části. ... 422
StrToLower — Převede řetězec na malá písmena ... 423
StrToUpper — Převede řetězec na velká písmena ... 423
StrTr — Převod znaků v řetězci podle konverzní tabulky ... 424
SubStr — Vrátí část řetězce ... 424
Trim — Odstranění mezer a tabulátorů ze začátku a konce řetězce ... 427
UCFirst — Převede první znak řetězce na velké písmeno ... 428
UCWords — Převede první znak každého slova v řetězci na velké písmeno ... 428

Funkce pro práci s URL adresami

Base64_Decode — Rozkóduje řetězec zakódovaný pomocí MIME-kódování Base64 ... 263
Base64_Encode — Zakóduje řetězec pomocí MIME-kódování Base64 ... 263
Parse_URL — Zjištění jednotlivých částí URL ... 388
URLDecode — Rozkóduje řetězec zakódovaný jako URL ... 430
URLEncode — Zakóduje řetězec tak, aby byl použitelný v URL ... 430

Funkce pro práci se soubory

`BaseName` — Zjistí jméno souboru z úplně zadané cesty k souboru 263
`ClearStatCache` — Vymazání vyrovnávací paměti volání `stat()` 269
`Copy` — Zkopírování souboru 272
`DirName` — Zjistí adresářovou část z úplně zadané cesty k souboru 284
`FClose` — Zavření souboru 291
`FEof` — Test konce souboru 291
`FGetC` — Přečtení jednoho znaku ze souboru 292
`FGetS` — Přečtení jedné řádky textu ze souboru 292
`FGetSS` — Přečtení řádky ze souboru a odstranění všech HTML a PHP tagů 292
`File` — Načtení celého souboru do pole 292
`File_Exists` — Zjištění, zda daný soubor existuje 293
`FileATime` — Zjištění času posledního přístupu k souboru 293
`FileCTime` — Zjištění času vytvoření souboru 293
`FileGroup` — Zjištění skupiny uživatelů, které soubor patří 294
`FileINode` — Vrací číslo i-node souboru 294
`FileMTime` — Zjištění času poslední modifikace souboru 294
`FileOwner` — Zjištění uživatele, kterému soubor patří 294
`FilePerms` — Zjištění přístupových práv k souboru 295
`FileSize` — Zjištění velikosti souboru 295
`FileType` — Zjištění typu souboru 295
`FOpen` — Funkce otevře soubor nebo URL-adresu 296
`FPassThru` — Zapíše zbývající obsah soubor na standardní výstup 297
`FPutS` — Zapíše řetězec do souboru 297
`FRead` — Binární čtení ze souboru 298
`FSeek` — Nastavení aktuální pozice v souboru 298
`FTell` — Zjištění aktuální pozice v souboru 300
`FWrite` — Zapíše řetězec do souboru 300
`ChGrp` — Změní skupinu uživatelů, které soubor náleží 267
`ChMod` — Změní přístupová práva k souboru 268
`ChOwn` — Změní vlastníka souboru 268
`Is_Dir` — Zjištění, zda dané jméno souboru je adresář 339
`Is_Executable` — Zjistí, zda je zadaný soubor spustitelný 340
`Is_File` — Zjistí, zda zadaný soubor je normální soubor 340
`Is_Link` — Zjistí, zda je zadaný soubor symbolický odkaz 341
`Is_Readable` — Zjistí, zda lze zadaný soubor číst 342
`Is_Writeable` — Zjistí, zda lze do zadaného souboru zapisovat 343
`Link` — Vytvoření pevného odkazu na soubor 356
`LinkInfo` — Zjištění informací o odkazu 356
`LStat` — Zjištění informací o symbolickém odkazu 357
`MkDir` — Vytvoření adresáře 360
`PClose` — Funkce uzavře ukazatel na rouru 389
`POpen` — Otevře rouru k nově spuštěnému procesu 400
`ReadFile` — Výpis souboru na standardní výstup 406
`ReadLink` — Zjištění, kam ukazuje symbolický odkaz 406
`Rename` — Přejmenování souboru 407
`Rewind` — Nastavení aktuální pozice souboru na jeho začátek 407
`RmDir` — Odstranění adresáře 408
`Stat` — Zjištění informací o souboru 416
`SymLink` — Vytvoření symbolického odkazu na soubor 425
`TempNam` — Vytvoření jedinečného jména souboru 426
`Touch` — Nastavení času poslední modifikace souboru 427
`UMask` — Nastavení masky přístupových práv pro nově vytvářené soubory 429
`UnLink` — Smazání souboru 429

Funkce pro práci se soubory dBase

dBase_Add_Record — Přidání záznamu do databáze 275
dBase_Close — Zavření databáze 275
dBase_Create — Vytvoření dBase databáze 275
dBase_Delete_Record — Smazání záznamu 276
dBase_Get_Record — Přečtení záznamu z databáze 276
dBase_NumFields — Funkce vrací počet položek, které obsahuje databáze 277
dBase_NumRecords — Zjistí počet záznamů v databázi 277
dBase_Open — Otevření databáze 278
dBase_Pack — Odstranění smazaných záznamů ze souboru s databází 278
dBase_Get_Record_With_Names — Přečtení záznamu z databáze 277

Funkce pro práci s protokolem SNMP

SNMPGet — Získání hodnoty jednoho SNMP objektu 412
SNMPWalk — Přečtení všech SNMP objektů od dané úrovně 413

Funkce pro přesné aritmetické operace

BCAdd — Sčítání 264
BCComp — Porovnání dvou čísel 264
BCDiv — Podíl 264
BCMod — Zbytek po dělení 264
BCMul — Součin 265
BCPow — Umocnění 265
BCScale — Nastavení přesnosti 265
BCSqrt — Druhá odmocnina 265
BCSub — Rozdíl 266

Funkce pro přístup k adresářovým službám

LDAP_Add — Přidání položky do adresáře LDAP 347
LDAP_Bind — Přihlášení k adresáři LDAP 348
LDAP_Close — Uzavření spojení s LDAP serverem 348
LDAP_Connect — Připojení k LDAP serveru 349
LDAP_Count_Entries — Zjištění počtu položek výsledku hledání 349
LDAP_Delete — Vymazání položky z adresáře 349
LDAP_DN2UFN — Funkce převede jméno DN do lidsky čitelné podoby 349
LDAP_Explode_DN — Rozložení DN jména na jednotlivé části 350
LDAP_First_Entry — Získání identifikátoru první položky výsledku 350
LDAP_Free_Result — Uvolnění výsledku prohledávání adresáře z paměti 351
LDAP_Get_Attributes — Zjištění všech atributů pro danou položku výsledku 351
LDAP_Get_DN — Zjištění DN jména položky výsledku 351
LDAP_Get_Entries — Přečtení všech položek výsledku prohledávání adresáře 352
LDAP_Get_Values — Přečtení všech hodnot atributu položky výsledku 352
LDAP_List — Prohledání jedné úrovně adresářového stromu 353
LDAP_Modify — Změna položky v adresáři LDAP 353
LDAP_Next_Attribute — Zjištění jména dalšího atributu položky výsledku 354
LDAP_Next_Entry — Přečtení další položky výsledku 354
LDAP_Read — Nalezení položky v adresáři 354
LDAP_Search — Prohledání adresářového stromu 355
LDAP_UnBind — Odhlášení se od LDAP adresáře 356
LDAP_First_Attribute — Zjištění jména prvního atributu položky výsledku 350

Funkce pro spouštění externích programů

`EscapeShellCmd` — Nahrazení všech nebezpečných znaků escape sekvencí 290
`Exec` — Vyvolání externího programu 290
`PassThru` — Spuštění externího programu a zobrazení neupraveného výstupu 389
`System` — Spuštění externího programu a zobrazení výstupu 426

Konfigurace a informace o PHP

`CloseLog` — Zavření systémového protokolu 270
`Debugger_Off` — Vypne interní debugger 281
`Debugger_On` — Zapne interní debugger 281
`Die` — Funkce ukončí běh skriptu a vypíše hlášení 283
`Error_Log` — Zaslání chybového hlášení 288
`Error_Reporting` — Určení chybových zpráv k hlášení 289
`Get_Browser` — Zjistí důležité informace o prohlížeči uživatele 300
`Get_Cfg_Var` — Zjištění hodnoty konfigurační proměnné 302
`Get_Current_User` — Vrací jméno uživatele, pod kterým je spuštěn aktuální skript 302
`GetEnv` — Zjištění hodnoty proměnné prostředí 304
`GetLastMod` — Zjištění data poslední modifikace skriptu 306
`GetMyINode` — Vrací číslo i-node právě prováděného skriptu 307
`GetMyPID` — Zjištění čísla procesu PHP 307
`GetMyUID` — Zjištění čísla vlastníka skriptu 307
`OpenLog` — Vytvoření přístupu k protokolu systémových událostí 387
`PHPInfo` — Zobrazení komplexních informací o PHP 399
`PHPVersion` — Zjištění verze PHP 399
`PutEnv` — Nastavení proměnné prostředí 404
`Register_ShutDown_Function` — Zaregistrování funkce, která se zavolá při skončení skriptu 407
`Set_Time_Limit` — Nastaví maximální dobu provádění skriptu 411
`SysLog` — Zápis zprávy do systémového protokolu 425

Konstanta

`__FILE__` — Tato konstanta obsahuje jméno souboru, ve kterém je uložen právě zpracovávaný skript 259
`__LINE__` — Tato konstanta obsahuje číslo řádku, na kterém je konstanta použita 259
`M_PI` — Konstanta obsahuje přibližnou hodnotu čísla π 358
`PHP_OS` — Konstanta obsahuje jméno operačního systému, na kterém PHP právě běží 399
`PHP_VERSION` — Konstanta obsahuje číslo verze právě používaného systému PHP 399

Matematická funkce

`Abs` — Absolutní hodnota ... 259
`ACos` — Arkus kosinus ... 260
`ASin` — Arkus sinus ... 262
`ATan` — Arkus tangens ... 263
`BinDec` — Převod dvojkového čísla na desítkové ... 266
`Ceil` — Zaokrouhlení desetinného čísla nahoru ... 266
`Cos` — Kosinus ... 272
`DecBin` — Převádí desítkové číslo na jeho binární reprezentaci ... 282
`DecOct` — Převod desítkového čísla na osmičkové ... 282
`Deg2Rad` — Převod stupňů na radiány ... 282
`DecHex` — Převod desítkového čísla na šestnáctkové ... 282
`Exp` — Umocní číslo na Eulerovu konstantu ... 290
`Floor` — Zaokrouhlení desetinného čísla dolů ... 296
`GetRandMax` — Nejvyšší hodnota, kterou může vrátit `Rand()` ... 307
`HexDec` — Převod šestnáctkového čísla na desítkové ... 309
`Log` — Přirozený logaritmus ... 357
`Log10` — Desítkový logaritmus ... 357
`Max` — Nalezení maxima z daných hodnot ... 359
`Min` — Nalezení minima z daných hodnot ... 360
`Number_Format` — Formátování čísla pro ekonomické výstupy ... 375
`OctDec` — Převod osmičkového čísla na desítkové ... 376
`Pi` — Vrací hodnotu Ludolfova čísla π ... 400
`Pow` — Výpočet mocnin ... 401
`Rad2Deg` — Převod radiánů na stupně ... 404
`Rand` — Generování náhodné hodnoty ... 404
`Round` — Zaokrouhlení desetinného čísla ... 408
`Sin` — Sinus ... 411
`Sqrt` — Druhá odmocnina ... 415
`SRand` — Inicializace náhodného generátoru ... 415
`Tan` — Tangens ... 426

Ostatní funkce

`Dl` — Načtení dynamické knihovny ... 284
`HighLight_File` — Zobrazí soubor se skriptem s použitím zvýrazněné syntaxe ... 310
`HighLight_String` — Zobrazí řetězec jako skript s použitím zvýrazněné syntaxe ... 310
`Leak` — Nenávratná alokace paměti ... 356
`Sleep` — Pozastavení skriptu ... 412
`UniqID` — Jedinečný identifikátor ... 432
`uSleep` — Pozastavení provádění skriptu ... 431

Podpora protokolu IMAP

`IMAP_8Bit` — Zakóduje text metodou quoted-printable ... 323
`IMAP_Append` — Přidání textové zprávy do poštovní schránky ... 323
`IMAP_Base64` — Dekódování textu zakódovaného metodou Base64 ... 323
`IMAP_Binary` — Zakódování textu metodou Base64 ... 324
`IMAP_Body` — Přečtení těla zprávy ... 324
`IMAP_ClearFlag_Full` — Smazání příznaků u zpráv ... 325
`IMAP_Close` — Uzavření spojení s IMAP-serverem ... 325
`IMAP_CreateMailBox` — Vytvoření nové poštovní schránky ... 325
`IMAP_Delete` — Označení zprávy pro smazání ... 325
`IMAP_DeleteMailBox` — Smazání poštovní schránky ... 326
`IMAP_Expunge` — Smazání všech zpráv označených pro smazání ... 326
`IMAP_FetchBody` — Přečtení jedné části těla dopisu ... 326
`IMAP_FetchHeader` — Přečtení hlavičky zprávy ... 327
`IMAP_FetchStructure` — Zjištění struktury zprávy ... 327
`IMAP_Header` — Přečtení hlavičky zprávy ... 329
`IMAP_Headers` — Získání hlaviček všech zpráv v poštovní schránce ... 330
`IMAP_Check` — Kontrola aktuální poštovní schránky ... 324
`IMAP_ListMailBox` — Zjištění všech dostupných poštovních schránek ... 330
`IMAP_ListSubscribed` — Zjištění všech poštovních schránek zapsaných k odběru ... 331
`IMAP_Mail_Copy` — Zkopírování zpráv do poštovní schránky ... 331
`IMAP_Mail_Move` — Přesunutí zpráv do jiné poštovní schránky ... 331
`IMAP_MailBoxMsgInfo` — Zjištění informací o aktuální poštovní schránce ... 332
`IMAP_Num_Msg` — Zjištění počtu zpráv v aktuální schránce ... 332
`IMAP_Num_Recent` — Zjistí počet nových zpráv ve schránce ... 332
`IMAP_Open` — Otevření spojení s IMAP serverem ... 333
`IMAP_Ping` — Kontrola aktivity spojení ... 333
`IMAP_QPrint` — Dekódování textu zakódovaného metodou quoted-printable ... 334
`IMAP_RenameMailBox` — Přejmenování schránky ... 334
`IMAP_ReOpen` — Nastavení aktuální schránky pro spojení ... 334
`IMAP_RFC822_Parse_AdrList` — Zjištění údajů z řetězce obsahujícího e-mailové adresy ... 335
`IMAP_RFC822_Write_Address` — Vytvoření korektní e-mailové adresy ... 335
`IMAP_ScanMailBox` — Nalezení schránek, které ve svém názvu obsahují daný text ... 335
`IMAP_SetFlag_Full` — Nastavení příznaků u zpráv ... 336
`IMAP_Sort` — Vrátí seznam zpráv setříděných podle určitého kritéria ... 336
`IMAP_Subscribe` — Přihlášení schránky k odběru ... 337
`IMAP_UID` — Vrátí identifikační číslo zprávy ... 337
`IMAP_Undelete` — Zruší označení zprávy pro smazání ... 337
`IMAP_Unsubscribe` — Odhlášení schránky z odběru ... 337

Proměnná

$CONTENT_LENGTH — Délka dat zasílaných metodou POST ... 257
$CONTENT_TYPE — MIME typ dat zasílaných metodou POST ... 257
$GATEWAY_INTERFACE — Použitá verze rozhaní CGI, pokud PHP běží jako CGI-skript (nejčastěji CGI/1.1) ... 257
$HTTP_COOKIE_VARS — Asociativní pole obsahující všechny cookies ... 257
$HTTP_GET_VARS — Asociativní pole obsahující všechny parametry předané metodou GET 257
$HTTP_POST_VARS — Asociativní pole obsahující všechny parametry předané metodou POST ... 257
$PATH_INFO — Cesta ke skriptu, který má být zpracován ... 257
$PATH_TRANSLATED — Skutečná cesta ke skriptu, který má být zpracován ... 257
$PHP_AUTH_PW — Heslo získané pomocí HTTP autentifikace ... 257
$PHP_AUTH_TYPE — Typ HTTP autentifikace — nejčastěji basic ... 258
$PHP_AUTH_USER — Uživatelské jméno získané při HTTP autentifikaci ... 258
$PHP_SELF — Jméno právě prováděného skriptu ... 258
$QUERY_STRING — Nerozkódovaná data předaná metodou GET ... 258
$REMOTE_ADDR — IP-adresa, ze kterého přišel požadavek ... 258
$REMOTE_HOST — Doménová adresa počítače, ze kterého přišel požadavek ... 258
$REQUEST_METHOD — Způsob předání parametrů (GET nebo POST) ... 258
$SCRIPT_FILENAME — Jméno souboru, ve kterém je uložen právě prováděný skript ... 258
$SCRIPT_NAME — Jméno právě prováděného skriptu ... 258
$SERVER_NAME — Adresa serveru (IP-adresa, doménová adresa nebo alias) ... 259
$SERVER_PORT — Číslo portu, na kterém běží WWW-server ... 259
$SERVER_PROTOCOL — Jméno a verze protokolu, kterým přišel požadavek (nejčastěji HTTP/1.0 nebo HTTP/1.1) ... 259
$SERVER_SOFTWARE — Název a verze WWW-serveru ... 259
DoubleVal — Hodnota výrazu jako typ double ... 284
Empty — Zjistí, zda je proměnná prázdná ... 285
GetType — Zjištění typu proměnné ... 308
IntVal — Celočíselná hodnota proměnné ... 338
Is_Array — Zjištění zda výraz je pole ... 339
Is_Double — Zjištění zda výraz je typu double ... 339
Is_Float — Zjištění, zda je výraz typu double ... 340
Is_Int — Zjištění, zda je výraz typu integer ... 340
Is_Integer — Zjištění, zda je výraz typu integer ... 341
Is_Long — Zjištění, zda je výraz typu integer ... 341
Is_Object — Zjištění, zda je výraz typu object ... 342
Is_Real — Zjištění, zda je výraz typu double ... 342
Is_String — Zjištění, zda je výraz typu string ... 343
IsSet — Zjištění, zda je proměnná zinicializována ... 343
SetType — Nastavení typu proměnné ... 411
StrVal — Převod hodnoty na řetězec ... 424
UnSet — Zrušení proměnné ... 429

Síťové funkce

FSockOpen — Otevření socketu ... 299
GetHostByAddr — Převod IP-adresy na adresu doménovou ... 304
GetHostByName — Převod doménové adresy na IP ... 304
GetHostByNameL — Zjištění všech IP-adres, které odpovídají jedné doménové adrese ... 305
GetMXRR — Přečtení MX záznamu z DNS ... 306
CheckDNSRR — Zjištění existence záznamu určitého typu v DNS ... 267
Set_Socket_Blocking — Nastavení blokujícího/neblokujícího režimu pro socket ... 410

Jiří Kosek

PHP – tvorba interaktivních internetových aplikací

Odpovědný redaktor Václav Urban
Návrh a grafická úprava obálky Adéla Bělovská

Počet stran 492

Vydala Grada Publishing, spol. s r. o.
U Průhonu 22, Praha 7

1998
Vydání 1.

Vytiskly Tiskárny Havlíčkův Brod, a.s.
Husova ulice 1881, Havlíčkův Brod

V poslední době můžeme na Internetu sledovat obrovskou změnu ve způsobu poskytování informací. Klasické statické webové stránky jsou postupně nahrazovány stránkami umožňujícími interaktivní přístup k předkládaným informacím. Pro tvorbu takových stránek je přímo optimální prostředí skriptovacího jazyka PHP, jenž umožňuje vývoj různorodých aplikací počínaje jednoduchým počítadlem přístupů a konče tvorbou podnikového informačního systému v prostředí intranetu či extranetu. Možnosti PHP umocňuje důsledná integrace s mnoha databázovými systémy. Velká výhoda PHP spočívá v jeho nezávislosti na používaných platformách. Výsledkem běhu PHP-skriptu je obyčejný HTML dokument, který umí zobrazit každý prohlížeč – odpadají tedy problémy s kompatibilitou, které přináší některé jiné vývojové prostředky.

Jiří Kosek, který je autorem řady bestselerů seznamujících s nejrůznějšími aspekty Internetu, v knize podrobně popisuje všechny vlastnosti jazyka PHP. Jediné, co od čtenáře očekává, je schopnost logicky a analyticky uvažovat a základní znalost jazyka HTML. Možnosti PHP ilustruje na mnoha praktických příkladech – např. na adresáři, objednávkové službě či virtuální jednací síni. Kromě samotného PHP se podrobně zabývá i jeho možnostmi napojení na databáze, zejména na volně šiřitelný databázový systém MySQL.

Kniha je určena pro všechny správce a také autory webových stránek.

Informace o dalších knihách a bezplatné zasílání katalogů zajišťuje zákaznický servis:
tel. 02/20 386 511-512,
fax: 02/20 386 400
e-mail: obchod@gradapublishing.cz
prodej na Internetu: www.gradapublishing.cz

GRADA Publishing, spol. s r. o., U Průhonu 22, Praha 7

Z dalších titulů nakladatelství GRADA Publishing:

ISBN 80-7169-373-1
9 788071 693734

Foto na přední straně obálky David Jelen, Milan Sklenička.