Programovanie v PHP

Prvá časť: Úvod

História

Vznik PHP sa datuje do roku 1994, pričom šlo o program v Perle na evidenciu prístupov k WWW stránkam. Jeho stvoriteľom je Rasmus Lerdorf, ktorý následne prepísal svoj softvér do Céčka. Časom ho začalo používať čoraz viac programátorov, a preto bol softvér rozšírený o požiadavky používateľov a o dokumentáciu a vydaný pod názvom Personal Home Page Tools. Neskôr bol do systému zabudovaný SQL modul, vďaka čomu mohol podporovať databázy. Tak vznikol spojením s pôvodným PHP systém Personal Home Page Form Interpreter PHP/FI, ktorý umožňoval spracúvať dáta z formulárov. V roku 1998 bola vydaná posledná verzia PHP 3.0. Dnes je už k dispozícii verzia PHP 4.0, vyvíjaná pod kódovým názvom ZEND. Verzia PHP 4.0 je niekoľkonásobne rýchlejšia ako predchádzajúca a bola rozšírená o niekoľko vlastností jazyka C/C++. Nejde zatiaľ o príliš rozšírenú verziu, pretože vždy nejaký čas trvá, kým správcovia serverov upgradujú sofvér na novšie verzie. To však nebráni, aby sme ju v našich budúcich programoch použili.

Prečo používať PHP?

Odpoveď je veľmi jednoduchá. V dnešnej dobe je k dispozícii niekoľko programovacích jazykov, ktoré slúžia na vytváranie dynamických stránok. Do popredia sa dostávajú hlavne Perl, ASP a v neposlednom rade PHP. Čím ďalej, tým viac serverov poskytuje podporu PHP skriptov, pričom určite zaváži aj fakt, že je zadarmo (aj keď je pravda, že to neplatí len o PHP, ale napr. aj o Perle). V každom prípade je PHP asi najjednoduchší z uvedených jazykov, takže je vhodný aj pre začiatočníkov v oblasti dynamicky generovaných stránok.

Inštalácia PHP a servera Apache

Pred inštaláciou pod Windows (95, 98, NT) si musíte stiahnuť PHP zo stránky http://www.php.net (verziu pre Win32, sami porozmýšľajte, či sa spoľahnete na starú verziu 3.x alebo využijete možnosti najnovšej verzie 4.0.1pl2). Ďalej si musíte nainštalovať WWW server, ktorý je schopný spúšťať CGI skripty. Druhou možnosťou (a myslím, že rozumnejšou) je stiahnuť si server Apache a spúšťať skripty ako moduly tohto servera. Ten nájdete na adrese http://www.apache.org (samozrejme, stiahnite si verziu pre Windows), kde nájdete aj prodrobné informácie o inštalácii. Po nainštalovaní servera Apache je potrebné ho nakonfigurovať. Pre konfiguráciu servera otvorte súbor httpd.conf (prípadne srm.conf pre staršie verzie). Tento súbor nájdete v podadresári CONF hlavného adresára Apache (tam, kde ste Apache inštalovali). Nájdite riadok obsahujúci na začiatku slovo ServerName (ak je pred týmto slovom znak #, zmažte ho, lebo riadky začínajúce sa týmto znakom sa pokladajú za komentáre a sú ignorované). Za toto slovo musíte napísať meno servera (nezáleží na tom, aké je, pomenujte si ho podľa seba), teda riadok môže vyzerať napr. takto:

```
ServerName andy
```

Teraz je potrebné nakonfigurovať server na prácu s PHP. Najprv rozbaľte inštalačný súbor, ktorý ste stiahli, do želaného adresára (napr. d:\php). Ak nemáte súbor php.ini, premenujte súbor php.ini-dist na php.ini. Otvorte tento súbor a nájdite riadok obsahujúci doc_root = , ktorý následne zmeňte na niečo ako:

```
doc_root = „d:\programs“
```

(v úvodzovkách uveďte adresár, kde budete mať svoje skripty)

Súbor php.ini uložte a zatvorte. Potom vytvorte adresár, ktorý ste uviedli v úvodzovkách. Súbor php.ini spolu so všetkými DLL súbormi v adresári PHP prekopírujte do adresára SYSTEM (SYSTEM32 pre WIN NT) v adresári Windows. PHP už máme nakonfigurované, teraz stačí dokonfigurovať Apache. Pokračujte v editovaní súboru httpd.conf. Ďalej pridajte postupne tieto riadky:

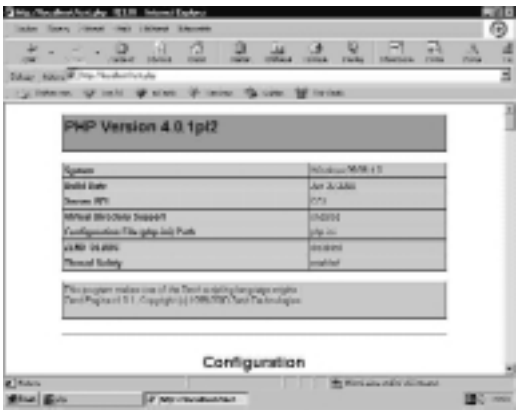
```
ScriptAlias /php/ „d:/php/“
AddType application/x-httpd-php .php .phtml
Action application/x-httpd-php „/php/php.exe“
DocumentRoot „d:/programs/“
<Directory „d:/programs“>
```

Samozrejme, názvy adresárov môžete nastaviť tak, ako to budete považovať za vhodné. Pri konfigurovaní Apache nezabudnite na opačné lomky. Teraz môžeme otestovať funkčnosť Apache aj PHP napísaním prvého skriptu.

Vytvorte súbor test.php v dokumentovom adresári (adresár, ktorý ste zadali v riadku DocumentRoot=, teda v našom prípade d:\programs). Otvorte súbor test.php a napíšte tam nasledujúci kód:

```
<?php
 PhpInfo();
?>
```

Súbor uložte a spustite WWW server (Apache). Spustite browser (Netscape alebo Internet Explorer) a do adresového riadka zadajte http://localhost. Ak sa vám zobrazil výpis dokumentového adresára, máte Apache nakonfigurovaný správne. Mali by ste teda vidieť aj súbor test.php. Kliknite naň a mal by sa vám zobraziť výpis informácií o PHP, tak ako vidíte na obrázku. V prípade, že sa tak stalo, máte správne nakonfigurované PHP. Teraz už môžete začať písať svoje prvé PHP skripty.



Obr. 1

Vloženie kódu do dokumentu

Veľkou výhodou PHP oproti iným programovacím jazykom využívaným v dnešných internetových aplikáciách je to, že kód PHP sa môže veľmi jednoducho a efektívne prelínať s HTML kódom. Preto nemusíte časti dokumentu, ktoré sú statické, a nie dynamické, stále vypisovať pomocou procedúr programovacieho jazyka. Súbor PHP skriptu môže teda obsahovať iba PHP kód, iba HTML kód alebo oba naraz. Prvým prípadom je program z predchádzajúceho príkladu test.php – obsahuje iba čistý PHP kód. Ako som už napísal, PHP a HTML kódy sa však môžu plynule prelínať:

```
HTML tagy
<?php
 PHP kód
>
HTML tagy
<?php
 PHP kód
>
HTML tagy
```

Ako vidíte, PHP kód sa do dokumentu vkladá pomocou špeciálnych tagov <?php a ?>. Podľa konfigurácie je možné uzatvárať kód aj do tagov <? a ?>. Ďalšou možnosťou, ako vložiť PHP kód do dokumentu, je využiť párový tag <SCRIPT>:

```
<SCRIPT LANGUAGE=“php“>
PHP kód
</SCRIPT>
```

Prvý program

Takže to nepríjemnejšie (inštaláciu) máme za sebou, teraz sa pustíme do programovania. Na začiatok si neodpustím programátorskú klasiku „Hello, world!“:

```
<?php
 echo „Hello, world!“;   // vytlaci Hello, world do vys-
tupu
?>
```

Po spustení skriptu by sa vám mal v prehliadači zobraziť reťazec „Hello, world!“. Takže si poďme program trochu objasniť. Vyskytuje sa tu príkaz echo, ktorý slúži na tlač reťazca do výstupu. Reťazec musí byť ohraničený úvodzovkami („) alebo apostrofmi (‘). Viac si o reťazcoch povieme neskôr. Ako ste si mohli všimnúť, do kódu je možné aj vpisovať komentáre podobným spôsobom, ako v Céčku. Buď použijete dve lomky (//) ako v našom príklade, alebo uzatvoríte komentár do znakov /* a */. Rozdiel je v tom, že ak použijete sekvenciu ‘//’, tak bude ignorované všetko od tejto sekvencie po koniec riadka. Ak použijete sekvencie ‘/*’ a ‘*/’, bude ignorované všetko medzi týmito sekvenciami.

Syntax PHP

PHP však nemá s Céčkom spoločný iba spôsob zápisu komentárov. Syntax PHP je značne podobná Céčku, aj keď nie totožná. Napriek tomu sa dá povedať, že kto ovláda C/C++, má spolovice vyhrané. Odlišností je naozaj minimum. Najväčším rozdielom je pravdepodobne práca s premennými. V PHP nie je potrebné premenné dopredu definovať a deklarovať. Premenná vzniká pri jej prvom použití a jej typ závisí od priradenej hodnoty. Rovnako PHP umožňuje jednoduchú konverziu medzi rôznymi typmi premenných. Ak napríklad budete mať v premennej uložené číslo a niekde s ňou začnete pracovať ako s reťazcom, interpreter PHP zabezpečí automatickú konverziu. Názvy premenných sa musia začínať znakom $ (čo je ďalšia odchýlka od jazyka C) a musia byť zložené iba z veľkých a malých znakov abecedy, číslic a znaku _. Prvým znakom (za znakom $) musí byť písmeno alebo podčiarkovač, nie číslica. Ukážka správneho a nesprávneho pomenovania premenných:

Správne pomenovania premenných: $a, $a1, $a_1, $_a1, $_1
Nesprávne pomenovania premenných: $1a, $a 1
Za každým príkazom musí nasledovať bodkočiarka (;).

Typy premenných

Ako som už napísal, v PHP nemusíte premenné deklarovať. PHP premennú zadeklaruje podľa toho, akú hodnotu jej priradíte. Tu je zoznam typov premenných v PHP:

a) integer – základná číselná premenná, slúži na uchovávanie celých čísel v desiatkovej sústave, prípadne aj v osmičkovej (prefix 0, napr. 0777) alebo šestnástkovej (prefix 0x, napr. 0xff):

```
$premenna = 3;     // premennej $premenna priradí číslo 3
$premenna = -8;    // záporné číslo
$premenna = 0777;  // osmičková sústava
$premenna = 0xff;  // šestnástková sústava
```

b) floating-point numbers (čísla s plávajúcou čiarkou, takisto reálne čísla) – zvané aj „double“, slúžia na uchovávanie reálnych čísel:

```
$premenna = 5.83;
$premenna = 3.2e0;  // zápis s exponentom
```

c) string – reťazce, ako už názov hovorí, uchovávajú reťazce. Reťazce zapisujeme do úvodzoviek alebo apostrofov:

```
$retazec = „Retazec v premennej“;
$retazec = ‘Aj pomocou apostrofov to ide’;
```

Môžeme sa dostať do situácie, že do reťazca chceme zakomponovať nejaký špeciálny znak ($,“,’,\ a pod.). Vtedy treba použiť escape sekvenciu (Céčkari iste poznajú). Escape sekvencie:

sekvencia	znamená v reťazci
\n	nový riadok
\r	carriage return (návrat vozíka)
\t	tabulátor
\\	lomka (\)
\$	dolár ($)
\“	úvodzovky
\’	

V prípade, že zapisujeme reťazce v apostrofoch (‘), je situácia trošku odlišná. Jedinými akceptovanými escape sekvenciami sú \\ a \’. Ostatné znaky je možné zapisovať priamo. Teraz sa možno pýtate, prečo teda zapisovať reťazce pomocou úvodzoviek, keď to ide pomocou apostrofov, pričom stačí dávať pozor iba na výskyt dvoch znakov. Odpoveď je veľmi jednoduchá – pri použití apostrofov nie možné tzv. variables expansion, teda rozvoj premenných. O čo ide? Často sa stane, že do reťazca chceme zakomponovať nejakú premennú, resp. jej hodnotu. Máme napríklad premennú $vek, v ktorej máme uložený vek osoby. Dajme tomu, že jej hodnota je 8. Ak chceme vypísať reťazec „Janko má 8 rokov“ tak, že číslo 8 vložíme pomocou premennej $vek, urobíme to takto:

```
$vek = 8;
echo „Janko má $vek rokov“;  // vypíše „Janko má 8 rokov“
```

Takto sme jednoducho vložili hodnotu premennej do reťazca. Ak by sme použili apostrofy, program by vypísal reťazec „Janko má $vek rokov“, pretože v apostrofoch je znak $ ignorovaný, a teda premenná nie je rozvinutá. V tomto prípade by sme museli použiť operátor spájania reťazcov (o operátoroch sa zmienime neskôr).

Ešte si niečo povieme o konverzii reťazcov. Ide vcelku o zložitú záležitosť. Pri sčítaní dvoch premenných, z ktorých je jedna číselná a druhá reťazec, platia tieto pravidlá: Výsledný typ premennej závisí od typu druhej premennej v súčte:

```
$foo = „10.0“ + 1;      // premenná $foo bude obsahovať hodnotu 11 a bude typu integer, pretože druhá hodnota je celočíselná
$foo = „10.0“ + 1.0;    // premenná $foo bude obsahovať hodnotu 11 a bude typu double, pretože druhá premenná je typu double
```

V prípade, že druhá hodnota je reťazec, bude vyhodnotená ako double vtedy, ak tento reťazec obsahuje niektorý zo znakov „.“, „e“ alebo „E“. Inak bude interpretovaná ako integer. Ak sa reťazec začína nejakým číslom, bude toto číslo interpretovanou hodnotou reťazca. Ak sa reťazec začína hocijakým iným znakom, bude interpretovanou hodnotou 0:

```
$foo = 1 + „10.5“;          // premenná $foo obsahuje hodnotu 11.5 a je typu double
$foo = 1 + „-1.3e3“;        // premenná $foo obsahuje hodnotu -1299 a je typu double
$foo = 1 + „a-1.3e3“;       // premenná $foo obsahuje hodnotu 1 a je typu integer
$foo = 1 + „b3“;            // premenná $foo obsahuje hodnotu 1 a je typu integer
$foo = 1 + „10 abcdef“;     // premenná $foo obsahuje hodnotu 11 a je typu integer
```

d) arrays – polia. Premenné typu array prezentujú základný štruktúrovaný typ. Pole je prezentované názvom ako jedna premenná a umožní vám ukladať viacero prvkov rovnakého typu (tzv. homogénne pole). Prvky v poli sú indexované. Rozlišujeme skalárne a asociované polia, pričom PHP podporuje oba typy. Rozdiel medzi asociovaným a skalárnym poľom spočíva v type jeho indexov. Skalárne pole je indexované celočíselnými hodnotami, zatiaľ čo indexy asociovaných polí sú reťazcového typu. Inak medzi nimi nie je nijaký rozdiel. Pole (asociatívne i skalárne) môžete vytvoriť funkciami array() a list() alebo zadaním hodnoty každému prvku:

```
$pole[0] = 1;
$pole[1] = 2;
$a[„ret“] = 3;
```

Toto všetko sa dá urobiť funkciou array() tak, že ako argumenty zadávate jednotlivé kľúče, ktorým priraďujete (pomocou =>) hodnoty:

```
$pole = array (0   => 1,
               1   => 2,
               „ret“ => 3);
```

Indexy však zadávať nemusíte, v takom prípade budú prvky automaticky pridávané na koniec poľa:

```
$a[] = 1;
$a[] = 2;
```

Polia budeme sem-tam potrebovať aj trochu pretriediť, k čomu nám PHP implementuje niekoľko funkcií, pričom naužitočnejšia je asi funkcia sort(), zvyšné nájdete v manuáli. Funkcia sort() roztriedi prvky poľa od najnižšej hodnoty po najvyššiu (alebo v prípade, že pole obsahuje reťazce, zoradí prvky abecedne). Ďalšou užitočnou funkciou je funkcia count(), ktorá zistí počet prvkov v poli (jeho meno musíte zadať ako parameter).

PHP taktiež umožňuje prácu s viacrozmernými poľami:

```
$pole[1] = 1;          // jednorozmerné pole
$pole[1][1] = 2;       // dvojrozmerné pole, matica
$pole[1][„a“][2] = 3;  // trojrozmerné pole, môžete mixovať reťazcové a číselné indexy, funguje pri poliach všetkých rozmerov
```

e) Classes – triedy. Objektovo orientované programovanie v C patrí už do výbavy kvalitného programátora. O výhodách oproti štruktúrovanému programovaniu sa tu nebudem vyjadrovať. Začiatočníci (ktorým je tento seriál určený) sa OOP ešte iste nebudú zaoberať, preto uvediem iba malý príklad pre pokročilých programátorov, z ktorého pochopia, že triedy fungujú v PHP rovnako ako v C:

Definícia triedy s jedným atribútom a jednou metódou:

```
class Trieda {
  int $atribut;
  function metoda() {
    echo „Telo funkcie“;
  }
}
```

Deklaráciu premennej typu Trieda:

```
$premenna = new Trieda;
```

Nabudúce sa budeme venovať operátorom a kontrolným štruktúram.

Druhá časť: Operátory a kontrolné štruktúry

Vítam vás pri druhej časti seriálu o programovaní v PHP. Na začiatku sme si povedali čoto o PHP, jeho histórii, vkladaní kódu a typoch premenných. Viem, že táto teoretická časť je značne nezáživná, ale na zvládnutie programovania v PHP potrebná. Teraz budeme v teórii pokračovať, pretože sa dozviete niečo o operátoroch a kontrolných štruktúrach.

Operátory
Operátory sú špeciálne znaky (alebo sekvencie znakov), ktoré slúžia na zápis operácií s hodnotami (premennými). Operátory PHP sú takmer totožné s operátormi C, takže opäť podotýkam, že Céčkari majú vyhrané.
1. Aritmetické operátory – základné, používajú sa na zápis vyhodnocovania aritmetických výrazov:

Znak	Zápis	Názov	Opis
+	$a+$b	Sčítanie	Súčet argumentov (v tomto prípade premenných $a a $b)
-	$a-$b	Odčítanie	Rozdiel argumentov
*	$a*$b	Násobenie	Súčin argumentov
/	$a/$b	Delenie	Podiel argumentov (od typu premennej, ktorej hodnotu priraďujeme, závisí od toho, či bude celočíselný alebo s desatinnou čiarkou)
%	$a%$b	Modulus	Zvyšok po celočíselnom delení argumentov

2. Priraďovacie – slúžia na priradenie hodnoty (ktorá môže byť výsledkom aritmetickej operácie alebo návratovou hodnotou funkcie) premennej:

```
$a = 1;      // premennej $a priradíme hodnotu 1
$a += 1;     // premennej $a priètame hodnotu 1,
                ekvivalentné k $a = $a+1;
             //  dá sa pou i  aj s ostatnými aritmetickými
                operátormi (-,*,/,%)
$a = „a“;    // premennej $a priradí re azec „a“
$a .= „bc“; // premennej $a pripojí re azec „bc“, tak e
                bude obsahova  re azec „abc“
```

3. Porovnávacie operátory – v kontrolných štruktúrach slúžia na vyhodnocovanie podmienok:

Znak	Zápis	Názov	Opis
==	$a==$b	Rovná sa	Vráti TRUE, ak sa argumenty rovnajú (v tomto prípade $a a $b)
===	$a===$b	Identické	Vráti TRUE, ak sú argumenty identické (rovnaký typ i hodnota) – až v PHP4
!=	$a!=$b	Nerovná sa	Vráti TRUE, ak sa argumenty nerovnajú
<	$a<$b	Menší ako	Vráti TRUE, ak je prvý argument menší ako druhý
>	$a>$b	Väčší ako	Vráti TRUE, ak je prvý argument väčší ako druhý
<=	$a<=$b	Menší alebo rovný	Vráti TRUE, ak je prvý argument menší alebo rovný druhému
>=	$a>=$b	Väčší alebo rovný	Vráti TRUE, ak je prvý argument väčší alebo rovný druhému

4. Inkrementačné a dekrementačné – slúžia na zvýšenie hodnoty premennej o 1 (rovnaké ako v C), môžu byť prefixové alebo postfixové, pri prvom type je najprv modifikovaná hodnota premennej a až potom je vrátená jej hodnota, druhý typ je opakom prvého:

Znak	Zápis	Názov	Opis
++	++$a	Inkrementuj o 1	Zvýši hodnotu premennej $a o 1, potom vráti hodnotu
++	$a++	Inkrementuj o 1	Najprv vráti hodnotu $a, potom ju zvýši o 1
—	—$a	Dekrementuj o 1	Zníži hodnotu premennej $a o 1, potom vráti hodnotu
—	$a—	Dekrementuj o 1	Najprv vráti hodnotu $a, potom ju zníži o 1

Na pochopenie nám poslúži jednoduchý príklad:

```
 $a=2;
echo $a++;   //vypíše 2 a zvýši hodnotu premennej $a o 1
```

```
   $a=2;
   echo ++$a;   //najprv inkrementuje premennú $a, potom
vypíše jej hodnotu (3)
```

5. Logické operátory – slúžia takisto na vyhodnocovanie podmienok, no pri zložených podmienkach, kombinujú viacero podmienok (tvorených pomocou porovnávacích alebo aj logických operátorov)

Názov	Zápis	Opis
and	$a and $b	Vráti TRUE, ak oba argumenty sú TRUE (v tomto prípade $a a $b) – možno nahradiť Céčkarským zápisom &&
or	$a or $b	Vráti TRUE, ak aspoň jeden argument je TRUE – možno nahradiť Céčkarským zápisom ||
xor	$a xor $b	Vráti TRUE, ak argumenty majú rôznu pravdivostnú hodnotu (jeden TRUE, druhý FALSE)
not	! $a	Vráti TRUE, ak argument je FALSE

6. Bitové operátory – slúžia na nastavovanie jednotlivých bitov v číslach:

Znak	Zápis	Názov	Opis
&	$a & $b	AND	Bitový súčin. Nastaví na 1 bity, ktoré sú nastavené na 1 v oboch argumentoch (v tomto prípade $a a $b)
|	$a | $b	OR	Bitový súčet. Nastaví na 1 bity, ktoré sú nastavené na 1 v hociktorom argumente
^	$a ^ $b	XOR	Bitová nonekvivalencia. Nastaví na 1 bity, ktoré sú nastavené na 1 buď v jednom, alebo v druhom argumente, nie však v oboch
~	~ $a	NOT	Bitová negácia. Nastaví na 1 bity, ktoré sú nastavené na 0 a naopak
<<	$a << $b		Bitový posun doľava. Posunie všetky bity prvého argumentu doľava o počet miest špecifikovaných v druhom argumente
>>	$a >> $b		Bitový posun doprava. Posunie všetky bity prvého argumentu doprava o počet miest špecifikovaných v druhom argumente

Kontrolné štruktúry a vetvenie programu
Kontrolné štruktúry slúžia na to, aby sa program mohol na základe rôznych podmienok a vstupov vetviť (prípadne cykliť) a dosahovať tak príslušné rôzne výstupy. Každá kontrolná štruktúra má nejakú riadiacu premennú alebo podmienku, podľa ktorej sa určuje, ktorá vetva kontrolnej štruktúry sa vykoná, prípadne koľkokrát sa vykoná blok daný ako iterácia nejakého cyklu. Vo všeobecnosti platí, že kontrolná premenná cyklu alebo podmienka musí byť vždy uzavretá v zátvorkách. V prípade, že vytvárame zloženú podmienku, tá musí byť celá v zátvorkách, no jej podpodmienky, z ktorých je zložená, v zátvorkách byť nemusia. Ak by sme sa však chceli striktne držať Céčkarskej syntaxe, mali by sme aj tieto podpodmienky uzatvárať do zátvoriek, aby sme tak mohli určovať postupnosť vyhodnocovania jednotlivých podpodmienok. V opačnom prípade sa môže stať, že zložitejšia zložená podmienka nemusí fungovať tak, ako by sme si predstavovali.

IF
Najzákladnejšou a najjednoduchšou kontrolnou štruktúrou je podmienka (if). Jej syntax je zhodná so syntaxou z C:

```
if (podmienka) prikaz;
```

Najprv sa vyhodnotí podmienka a v prípade, že je vyhodnotená ako pravdivá (TRUE), vykoná sa príkaz. Ak potrebujeme vykonať viac príkazov, musíme ich uzavrieť zloženými zátvorkami ako blok:

```
 if (podmienka) {
     prikaz1;
     prikaz2;
     prikaz3;
 }
```

Niekedy (skoro vždy) však potrebujeme vykonať nejaké príkazy v prípade, že podmienka nebola splnená. Na to slúži výraz else, ktorý sa pridáva za podmienku:

```
 if (podmienka) {
     prikaz1;
     prikaz2;
     prikaz3;
 }
   else {         // bude vykonané iba vtedy, ak bude
podmienka vyhodnotená ako FALSE
     prikaz1;
     prikaz2;
     prikaz3;
 }
```

Špeciálnym prípadom je elseif, ktorý má podobnú funkciu ako else, ibaže pridáva ešte jednu podmienku, teda rovná sa konštrukcii else if (podmienka):

```
 if (podmienka) {
     prikaz1;
```

```
     prikaz2;
     prikaz3;
  }
    elseif (podmienka2) {       // bude vykonané iba vtedy,
ak bude podmienka vyhodnotená ako FALSE
     prikaz1;                      // a zároveò podmienka2
bude TRUE
     prikaz2;
     prikaz3;
  }
    else {        // bude vykonané iba vtedy, ak bude pod-
mienka aj podmienka2 vyhodnotená ako FALSE
     prikaz1;
     prikaz2;
     prikaz3;
  }
```

Podmienka if ponúka aj alternatívnu syntax – za podmienkou nasleduje dvojbodka (:) a na konci výraz endif (osobne však radšej používam tú Céčkarskú):

```
if ($a > $b):
     echo „a je vacsie ako b“;
     echo „<BR>“;
  elseif ($a == $b):
     echo „a sa rovna b“;
     echo „<BR>“;
  else:
     echo „a je mensie ako b“;
     echo „<BR>“;
  endif;
```

WHILE

Ide o najjednoduchší typ cyklu. Má podobnú syntax ako v C:

```
while (výraz) príkaz;
```

príkaz je vykonávaný dovtedy, kým je hodnota výrazu TRUE. Nazýva sa aj cyklom s podmienkou na začiatku. Vždy na začiatku cyklu sa otestuje výraz a v prípade, že hodnota je TRUE, vykoná sa príkaz a opäť od začiatku. V prípade, že potrebujeme vykonať viac príkazov (čo je takmer vždy), musíme použiť blok, teda príkazy uzavrieť do zložených zátvoriek:

```
while (výraz) {
  príkaz1;
  príkaz2;
  príkaz3;
  atd...
}
```

Cyklus while, samozrejme, tiež umožňuje alternatívnu syntax. Tá je podobná, ako bola pre IF, tu nám poslúži konštrukcia while(výraz): a endwhile, pričom syntax je nasledujúca:

```
while (výraz):
   príkaz1;
   príkaz2;
   atd...
endwhile;
```

DO..WHILE

Cyklus do..while je veľmi podobný cyklu while. Rozdiel tkvie v tom, že zatiaľ čo v predchádzajúcom prípade bola podmienka testovaná na začiatku, tu je to zase na konci. Z toho vyplýva, že cyklus do..while musí prebehnúť minimálne raz, zatiaľ čo pre cyklus while toto neplatí. Aj cyklus do..while sa vykonáva dovtedy, kým je hodnota výrazu TRUE. Syntax cyklu do..while je takáto:

```
do {
  príkaz1;
  príkaz2;
  atd...
} while (podmienka);
```

Môže sa stať, že budete potrebovať cyklus, ktorý bude nekonečný a bude sa dať ukončiť iba zvnútra cyklu. Na to nám poslúži príkaz break (ktorý je použiteľný pri všetkých typoch cyklov a slúži na ukončenie práve prebiehajúceho bloku či cyklu):

```
do {
  if (podmienka) break;
} while (1);
```

Posledný riadok nám zabezpečí, že cyklus sa nikdy neukončí (1 je vždy vyhodnocovaná ako TRUE) a dá sa ukončiť iba v prípade, že je splnená podmienka príkazom break.

FOR

Cyklus for je jedným za najkomplexnejších v PHP. Jeho syntax je opäť podobná Céčku:

```
for (výraz1; výraz2; výraz3) príkaz1;
```

výraz1 je vyhodnotený (vykonaný) iba raz, na začiatku cyklu; výraz2 slúži ako kontrolná podmienka pre cyklus, pretože je vyhodnocovaná na začiatku každej iterácie. V prípade, že je vyhodnotená ako TRUE, je iterácia vykonaná, ak FALSE, celý cyklus sa končí; výraz3 slúži hlavne na modifikáciu riadiacej premennej a je vykonaný po skončení každej iterácie. Každý z týchto výrazov je nepovinný, takže konštrukcia cyklu for môže vyzerať aj takto:

```
for (;;)
{
 príkaz1;
 príkaz2;
 atd...
}
```

Samozrejme, tu nesmiete zabudnúť na modifikáciu a kontrolu riadiacej premennej vnútri a pred začiatkom cyklu, inak sa ľahko môže stať, že sa zacyklí.

Aj cyklus for má svoju alternatívnu syntax:

```
for (výraz1; výraz2; výraz3):
 príkaz1;
 príkaz2;
 atd...
endfor;
```

Programátorskou špecialitkou je vkladanie jednoduchého kódu namiesto výrazu, kde sa modifikuje riadiaca premenná. Ak máme napríklad pole čísel a chceme zistiť ich súčet bez použitia funkcie zabudovanej v PHP, môžeme to urobiť takto (v premennej $pole máme uložené prvky postupnosti, $suc bude po skončení cyklu súčet):

```
for ($i=0;$i<count($pole);$suc+=$pole[$i++]);
```

Ako vidíte, namiesto modifikovania premennej $i je celý výraz, ktorý pričíta k premennej $suc hodnotu prvku poľa $pole s indexom $i, ktorý hneď na to inkrementuje, čím zabezpečí modifikáciu riadiacej premennej. Funkcia count(), ktorú sme použili, vracia počet prvkov poľa, ktoré zadávame ako argument. Preto bude mať tento cyklus presne toľko iterácií, koľko má pole $pole prvkov (nezabúdajme, že prvky poľa sú implicitne indexované od 0, preto je premenná $i inicializovaná nulou). Keby sme celý tento cyklus chceli zapísať klasicky, teda bez príkazu namiesto modifikácie premennej, vyzeral by asi takto:

```
for ($i=0;$i<count($pole);$i++) $suc+=$pole[$i];
No povedzte, nevyzerá ten predchádzajúci príklad lepšie?
```

CONTINUE

Ide o špeciálny príkaz, ktorý slúži na skok na ďalšiu iteráciu. Jednoducho povedané, ukončí momentálnu iteráciu skokom na koniec bloku cyklu a tým sa zabezpečí prechod na ďalšiu iteráciu. Tento príkaz môžete použiť na všetky typy cyklov, teda while, do..while aj for.

```
for (výraz1; výraz2; výraz3) {
  if (podmienka) continue;  // ak bola podmienka splnená,
skoè na koniec bloku
  príkaz1;                    // tu sa pokraèuje iba
vtedy, ak podmienka nebola splnená
  príkaz2;
  atï...
}
```

SWITCH

Poslednou kontrolnou štruktúrou, o ktorej si povieme, je switch. Céčkari si opäť prídu na svoje, pretože syntax je zasa podobná C. Switch slúži na nahradenie po sebe idúcich podmienok IF s rovnakou premennou v podmienke. Do zátvoriek uvedieme premennú, ktorú chceme testovať, a potom nasledujú bloky case s hodnotami, pričom program vždy skočí na tú vetvu, pri ktorej je uvedená hodnota zhodná s hodnotou uloženou v premennej. Treba si však dať pozor na to, že po vykonaní niektorej z vetiev program pokračuje aj po ostatných, uvedených za ňou. Preto je potrebné na konci každej vetvy použiť príkaz BREAK. Ak by ste tak neurobili, program by skočil na vyhovujúcu vetvu a vykonal by aj ostatné za ňou.

```
switch ($premenna) {
    case 0:
        print „premenna sa rovna 0“;
        break;
    case 1:
```

```
        print „premenna sa rovna 1“;
        break;
    case 2:
        print „premenna sa rovna 2“;
        break;
}
```

Ako vidíte, jazyk PHP nie je až taký zložitý. Nemáme síce za sebou priveľa teórie, ale myslím, že aj ten, kto nepracoval s C/C++ a nepozná ich, by nemal mať problémy naučiť sa PHP. Chce to však, ako pri všetkých programovacích jazykoch, sadnúť si za počítač a skúšať, skúšať a ešte raz skúšať. Keď niečo nefunguje, zistiť prečo, pretože na chybách sa človek najlepšie učí. Dúfam, že tých chýb veľa neurobíte. Preto tu je aj náš seriál. Nabudúce si povieme niečo o súborovom systéme PHP. Zatiaľ dovidenia.

Tretia časť: PHP a Filesystem

Vítam vás už pri tretej časti nášho seriálu, ktorý má za úlohu naučiť vás programovať v PHP, teda v jazyku, ktorý má vo svete dynamicky generovaných stránok čoraz väčšiu podporu a obľubu. Po tom, čo sme si osvojili základy syntaxe jazyka, značne podobného jazyku C, ako sú typy premenných, kontrolné štruktúry, vetvenia a cykly, všetko z oblasti zväčša všeobecných čŕt jazyka, prejdeme teraz na niečo trošku konkrétnejšie. Čaká na nás filesystem čiže po slovensky súborový systém. Ide o problematiku značne citlivú hlavne na bezpečnosť vzhľadom na to, že nie každý programátor musí byť skúseným správcom servera a pri práci so súbormi môže urobiť niečo, čo môže mať neblahý dosah na beh celého servera. Preto hneď na začiatku upozorňujem, že si treba dávať pozor na to, kde a s akými súbormi pracujete. Ak totiž v PHP pracujete a máte svoje skripty na freehostingových serveroch, ako napríklad www.miesto.sk, musíte nastavovať práva súborov cez FTP klienta, pričom ich väčšinou budete nastavovať na rwe-rwe-rwe (teda všetky práva pre všetkých používateľov – read, write, execute). To isté platí aj pre adresáre. Keby niekto zistil, že u seba takýto súbor alebo adresár máte, mohol by vám vyviesť najrôznejšie zákernosti. Najlepším spôsobom, ako to obísť, je vytvoriť taký adresár, ktorého majiteľom bude iba server a iba ten bude mať právo zápisu, čítania a vykonávania (exekúcie) v tomto adresári. V ňom by ste si potom mohli robiť, čo chcete, zapisovať i čítať. Problémom je, že nejde o jednoduchú vec a na freehostingových serveroch sa takýchto možností nedočkáte. Neostáva vám teda nič iné ako rozumne navrhnúť program a rozhodiť dátové súbory do iných adresárov, ako sú spustiteľné skripty, a tým aspoň čiastočne program zabezpečiť. Vďaka týmto problémom je filesystem v internetových diskusných fórach často vďačnou témou. Poďme sa však už konkrétne pozrieť na ten v PHP.

Základné funkcie

Základnými funkciami filesystemu môžeme nazvať funkcie na otváranie a zatváranie súborov. Ide o funkcie `fopen()` a `fclose()`. Asi ste už usúdili, že prvá funkcia bude slúžiť na otvorenie súboru. Vracia premennú typu file pointer (teda niečo ako ukazovateľ súboru), ktorá má význam pri pohybe po súbore. Má tri parametre, pričom ten tretí je nepovinný a skoro sa nepoužíva. Preto sa budeme venovať tým prvým dvom. Prvým parametrom je reťazec obsahujúci meno súboru, ktorý chcete otvoriť. Ak sa tento reťazec začína „http://“, otvorí sa HTTP pripojenie na špecifikovaný server. Takisto môžete zadávať mená súborov na FTP, takže pripojenie na FTP je realizované v prípade, že meno súboru sa začína na „ftp://“. Ak sa reťazec začína čímkoľvek iným, súbor je otvorený z lokálneho filesystemu. Druhým parametrom je takisto reťazec, v ktorom je identifikátor módu, akým budete pristupovať k súboru. Ten môže byť jeden z nasledujúcich:

r – otvorí súbor iba na čítanie, ukazovateľ je nastavený na začiatok súboru

r+ – otvorí súbor na čítanie a na zápis, ukazovateľ je nastavený na začiatok súboru

w – otvorí súbor iba na zápis, ukazovateľ je nastavený na začiatok súboru a dĺžka súboru je skrátená na nulu (teda všetko zo súboru je zmazané). Ak súbor neexistuje, pokúsi sa ho vytvoriť

w+ – otvorí súbor na zápis i čítanie, ukazovateľ je nastavený na začiatok súboru a dĺžka súboru je skrátená na nulu. Ak súbor neexistuje, pokúsi sa ho vytvoriť

a – otvorí súbor iba na zápis, ukazovateľ je nastavený na koniec súboru. Ak súbor neexistuje, pokúsi sa ho vytvoriť

a+ – otvorí súbor na čítanie i zápis, ukazovateľ je nastavený na koniec súboru. Ak súbor neexistuje, pokúsi sa ho vytvoriť

Ako som už písal, funkcia vracia ukazovateľ daného súboru. Aby sme ho potom mohli používať, musíme túto návratovú hodnotu priradiť nejakej premennej. Táto premenná nám potom poslúži pri práci so súbormi pri ostatných funkciách. Teraz malá ukážka otvárania súborov:

```
$subor = fopen(„data/meno.dat“,“r“);    //otvorí súbor
meno.dat v adresári data pre èítanie
$subor = fopen(„info.dat“,“w+“);        //otvorí súbor
info.dat pre zápis i èítanie, zma e všetko, èo v òom je,
ak neexistuje, skúsi ho vytvori
$subor = fopen(„ftp://login:heslo@server.sk“,“w“);
//otvorí súbor na FTP
```

Funkcia `fclose()` má jediný parameter, a to už spomínaný ukazovateľ súboru, teda file pointer. Táto funkcia nerobí nič iné, iba zatvorí súbor, ktorému je priradený príslušný ukazovateľ, zadaný ako parameter tejto funkcie. Funkcia vracia hodnoty true alebo false v závislosti od toho, či prebehlo zatváranie súboru úspešne alebo nie. Ako príklad si uvedieme zatvorenie súboru otvoreného v predchádzajúcom príklade:

```
fclose($subor);
```

Ďalšie funkcie na prácu s filesystemom

Takže tie dve základné funkcie máme za sebou, ale iste uznáte, že to nám zatiaľ príliš nepomôže. Preto si teraz ukážeme ďalšie, ktoré nám umožnia plnohodnotne pracovať so súbormi.

Tou prvou, o ktorej si povieme, je funkcia `fgets()`. Táto funkcia slúži na prečítanie riadka zo súboru. To, čo zo súboru prečíta, je interpretované ako návratová hodnota typu reťazec, takže výsledok tejto funkcie by bolo vhodné priradiť nejakej premennej. Funkcia má dva parametre, pričom oba sú povinné. Prvým je file pointer súboru, z ktorého chceme čítať (z čoho vyplýva, že pred použitím funkcie musíme súbor otvoriť). Druhý parameter špecifikuje počet bajtov (znakov), ktorý má byť prečítaný zo súboru (pričom je prečítaný vždy počet znakov o 1 nižší, než je zadané týmto parametrom, teda ak zadáme 100, bude prečítaných iba 99 znakov). Znaky však budú čítané iba na jednom riadku, pretože čítanie sa môže končiť nielen dosiahnutím dĺžky zadanej druhým parametrom, ale aj dosiahnutím príznaku konca riadka alebo konca súboru. To znamená, že ak prikážeme prečítať väčší počet znakov, ako sa nachádza v riadku, bude prečítaný nižší počet znakov, resp. budú prečítané iba znaky nachádzajúce sa v riadku, po prechode na nový riadok sa čítanie končí. Tu je malá ukážka použitia funkcie `fgets()`:

```
$subor = fopen(„text.txt“,“r“);  //otvorenie súboru na
èítanie
$text = fgets($subor,1000);   //preèíta 999 znakov, prí-
padne celý riadok zo súboru text.txt do premennej $text
fclose($subor);  //zatvorí súbor text.txt
```

V prípade, že budete chcieť zo súboru načítať iba jeden znak, môžete použiť funkciu `fgetc()`. Tá má iba jeden parameter, ktorým je file pointer súboru, z ktorého chcete čítať. Návratovou hodnotou je jeden znak načítaný zo súboru, ktorý je identifikovaný týmto file pointerom. V prípade, že ukazovateľ je už na konci súboru (príznak EOF), návratová hodnota je FALSE.

Ďalšou funkciou, o ktorej si povieme, je `fputs()`. Podľa mena ste už možno usúdili, že pôjde o funkciu, ktorá zapíše reťazec do súboru. Táto funkcia má tri parametre, pričom prvý je už známy file pointer. Druhým parametrom je reťazec alebo premenná typu reťazec, ktorej obsah chceme zapísať do súboru. Tretí parameter je nepovinný a špecifikuje počet znakov, ktorý má byť do súboru zapísaný. Ak tento parameter neuvedieme, zapíše sa celý reťazec. Samozrejme, platí, že ak bude špecifikovaný počet znakov väčší ako dĺžka samotného reťazca, bude zapísaných iba toľko znakov, koľko ich reťazec skutočne obsahuje. Ukážka použitia funkcie:

```
$text = „Ahoj, ako sa máte?\n“;
$subor = fopen(„text.txt“,“w“);  //otvorenie súboru na
zápis, skrátenie na nulovú då ku
fputs($subor,$text);   //zapíše re azec v premennej $text
do súboru text.txt
fclose($subor);  //zatvorí súbor text.txt
```

Možno sa niekto môže začudovať tomu, že reťazec $text je ukončený sekvenciou „\n“. Kto však pozorne čítal predchádzajúce časti, iste vie, že ide o escape sekvenciu, ktorá znamená ukončenie riadka a prechod na nový (newline).

Teraz sa budeme venovať ďalšej funkcii z filesystemu PHP, konkrétne je to funkcia `file_exists()`. Táto funkcia nachádza pomerne časté využitie, pretože slúži na overenie skutočnosti, že daný súbor existuje. Preto ak chcete z nejakého súboru čítať, mali by ste použiť najprv túto funkciu, aby ste mali istotu, že vám program nespadne pre pokus čítať z neexistujúceho súboru. Každý dobrý programátor by sa mal snažiť ošetriť svoje programy pred pádom pre chybu pri vstupe. Poďme však naspäť k funkcii `file_exists()`. Tá má jediný parameter, ktorým je reťazec obsahujúci meno súboru. Z logických dôvodov sa tu už nepoužíva náš starý známy file pointer, pretože testovať, či existuje súbor, ktorý máme otvorený, je nezmyselné. Takže najprv test pomocou `file_exists()` a až potom otvorenie pomocou `fopen()`. Funkcia vracia true vtedy, ak súbor existuje, v opačnom prípade false. Preto otváranie súboru text.txt v adresári docs by malo prebiehať asi takto:

```
if (file_exists(„docs/text.txt“)) {
        $subor = fopen(„docs/text.txt“,“r“);
        .
        .
```

```
        práca so súborom
        .
        fclose($subor);
}
else {
        echo „Bohu ia¾, súbor docs/text.txt neexistuje.“;
}
```

Takto ošetrený súbor nebude mať vo výstupe nijaké chyby či varovania od interpretera.

Treba však pripomenúť, že výsledky tejto funkcie sú ukladané do tzv. cache, čo je pamäť, z ktorej sa „vyberajú“ výsledky pri ďalšom volaní tejto funkcie. Preto vrátená hodnota funkcie bude rovnaká ako prvýkrát, aj keď ju zavoláte znovu niekoľkokrát po sebe. Možno sa pýtate, prečo je to takto zbytočne komplikované. Všetko má však svoju príčinu – exekúcia niektorých funkcií filesystemu dosť zaťažuje systém. Medzi takéto funkcie patrí aj zmienená `file_exists()`. Preto sú hodnoty, ktoré tieto funkcie vracajú, ukladané do pamäte cache, aby boli tieto návratové hodnoty pri ďalšom volaní rýchlejšie dostupné. Niekedy však potrebujeme, aby bola funkcia skutočne vykonaná, a nie aby boli iba interpretované výsledky z predošlých volaní. Preto existuje ďalšia funkcia `clearstatcache()`. Táto funkcia nemá nijaké parametre a nevracia nijakú hodnotu, jej úloha spočíva vo vymazaní spomínanej pamäte cache, čím sa zabezpečí získanie skutočných a aktuálnych výsledkov funkcií, ktoré by inak boli vyhodnocované z pamäte cache.

Ďalšia funkcia v poradí je `feof()`. Ide o funkciu testujúcu koniec súboru, ktorého file pointer musíte odovzdať ako parameter. Tento parameter je zároveň jediným parametrom tejto funkcie. Návratovou hodnotou môže byť true alebo false v závislosti od toho, či je ukazovateľ súboru na jeho konci (príznak EOF). Ak ukazovateľ koniec súboru dosiahol, vracia funkcia true. Ukážka použitia je v príklade, kde vypisujeme riadky súboru do výstupu dovtedy, kým nedosiahneme jeho koniec:

```
$subor = fopen(„text.txt“,“r“);
while (!feof($subor)) {
        echo fgets($subor,$text);
}
fclose($subor);
```

Môže sa stať, že budete čítať zo súboru a potom niekde v jeho prostriedku znovu chcete začať čítať zo začiatku. Jedným zo spôsobov, ako to urobiť, je súbor zatvoriť a znovu otvoriť, čo je programátorsky dosť „neslušné“ riešenie. Máme tu však funkciu `rewind()`. Tá nám previnie ukazovateľ súboru, ktorý jej zadáme ako parameter, na jeho začiatok. Tento file pointer je zároveň jediným parametrom funkcie. Návratovou hodnotou je 0 v prípade, že nastala nejaká chyba.

Na posun ukazovateľa po súbore slúži aj funkcia `fseek()`, ktorá nastaví file pointer na pozíciu definovanú parametrom. Parametre sú dva, prvým je file pointer súboru, ktorým chceme manipulovať, druhým je offset, teda pozícia, na ktorú chceme ukazovateľ umiestniť. Offset vyjadruje počet bajtov od začiatku súboru k žiadanej pozícii. Návratová hodnota je 0 v prípade úspešného presunutia, -1 v prípade, že sa vyskytla nejaká chyba. Zadanie offsetu väčšieho, ako je veľkosť súboru, teda pokus umiestniť ukazovateľ za koniec súboru (EOF), sa nepokladá za chybu. Na zistenie aktuálnej pozície ukazovateľa súboru využijeme funkciu `ftell()`. Jediným parametrom je file pointer súboru, návratovou hodnotou je pozícia ukazovateľa súboru (offset), v prípade chyby vracia false.

Teraz sa budeme venovať niekoľkým systémovým funkciám, na ktorých použitie nie je potrebné súbor otvárať. Prvou je funkcia `copy()`, ktorá slúži na kopírovanie súboru a zároveň jeho premenovanie. Má dva parametre, source a destination, teda zdroj a cieľ. Prvý parameter (zdroj) špecifikuje meno súboru (prípadne aj s cestou k nemu), ktorý chceme kopírovať, druhý (cieľ) znamená umiestnenie cieľového (skopírovaného) súboru a aj jeho meno. To značí, že pri kopírovaní môžeme zároveň súbor premenovať, takže je možné kopírovať do rovnakého adresára, v akom sa nachádza zdroj. Funkcia vracia true, ak prebehlo kopírovanie v poriadku, inak vráti false:

```
if (!copy(„zdroj.dat“, „zdroj.bak“)) {
    echo „Chyba pri kopírovaní...“;
}
else {
    echo „Kopírovanie prebehlo v poriadku...“;
}
```

Ak chceme súbor iba premenovať, a nie kopírovať, môžeme použiť funkciu `rename()`. Má dva parametre, prvým je meno súboru, ktorý chceme premenovať, druhým je nové meno, ktorým chceme súbor nazvať. Návratové hodnoty sú rovnaké ako pri `copy()`, teda true pri úspešnom premenovaní, inak false.

Ďalšou užitočnou funkciou je `filesize()`. Táto funkcia umožňuje zistiť veľkosť súboru. Jej jediným parametrom je meno súboru, ktorého veľkosť potrebujeme poznať. Návratovou hodnotou je veľkosť súboru alebo v prípade chyby false. Treba podotknúť, že výsledky tejto funkcie sú ukladané do pamäte cache, a preto pri jej opätovnom použití je vhodné podľa okolností najprv zavolať funkciu `clearstatcache()`. Viac o funkciách, ktorých výsledky sú ukladané do cache, ste si mohli prečítať pri opise funkcie `file_exists()`. Podobne je to s funkciou `fileatime()`. Táto funkcia vracia čas posledného prístupu k súboru, ktorý jej zadáte ako jediný parameter. Pri použití funkcie sa musíte teda opäť vyrovnať s pamäťou cache. No a ešte si ukážeme, ako súbor zmazať. Na to nám poslúži funkcia unlink(), ktorá má opäť jediný parameter – meno súboru, ktorý chceme zmazať. Návratová hodnota je 0 alebo false v prípade výskytu chyby.

Teraz si povieme o ďalších dvoch možnostiach čítania súborov. Ide o využitie funkcií `file()` a `readfile()`. Obe funkcie majú niekoľko spoločných vlastností – každá má jeden povinný parameter, ktorým je meno súboru, ktorý chceme čítať. Každá má aj jeden nepovinný parameter, ktorý nemá nejaké významné využitie, a preto sa mu nebudeme venovať. No a obe funkcie slúžia na načítanie celého obsahu súboru. V čom je teda rozdiel? Odpoveď je jednoduchá – rozdiel spočíva v spôsobe interpretácie obsahu súboru. Funkcia `readfile()` obsah po načítaní celý vypíše do výstupu (teda zväčša do výsledného generovaného dokumentu). Na druhej strane funkcia `file()` načíta súbor do poľa, pričom jednému elementu poľa zodpovedá jeden riadok. Jednoducho povedané, súbor s desiatimi riadkami je načítaný do poľa s desiatimi prvkami. Z toho vyplýva, že výsledok funkcie `file()` musíme priradiť premennej typu pole, zatiaľ čo výsledok `readfile()` je počet prečítaných bajtov, takže ak túto informáciu nepotrebujeme, nemusíme ju priraďovať žiadnej premennej. Ešte podotýkam, že na použitie týchto funkcií netreba súbor otvárať, pretože parametrom je meno súboru, nie jeho file pointer.

Problémy pri paralelnej práci so súbormi

Najväčším problémom pri práci so súbormi na serveri, kde má prístup relatívne neobmedzený počet používateľov, je takmer istá pravdepodobnosť, že sa naraz pokúsi pracovať so súborom viac skriptov. To má väčšinou za následok poškodenie súboru, stratu dát a aj krach programu. Preto je potrebné takéto prípady nejakým spôsobom ošetriť. Jedným z riešení je vždy si vytvárať nejaký pracovný súbor, ktorého existencia signalizuje, že s potrebným súborom sa pracuje, a preto treba počkať. Implementácia riešenia by bola približne takáto:

```
while (file_exists(„work“)) {
        sleep(1);
        clearstatcache();
}
$subor = fopen(„work“,“w“);
fclose($subor);
.
.
práca s potrebným súborom
.
.
unlink(„work“);
```

Poďme si to trošku objasniť. Takže na začiatku je cyklus while, ktorý kontroluje, či existuje súbor `work`. Ak áno, na jednu sekundu sa exekúcia skriptu zastaví, na čo nám poslúži funkcia `sleep()`. Potom sa vymaže pamäť cache a opäť otestujeme existenciu súboru `work`. Toto sa opakuje, kým súbor `work` existuje. Hneď ako je tento súbor zmazaný, pokračuje skript tým, že vytvorí znovu súbor `work`, čím si rezervuje právo používať potrebné súbory, takže všetky ostatné skripty budú musieť počkať. Vytvorenie súboru `work` pozostáva z otvorenia na zápis a okamžitého zatvorenia. Potom sa môžeme venovať práci so samotnými súbormi, pre ktoré je skript určený. Po ukončení práce skript zmaže súbor `work` a tým umožní ďalšiemu skriptu (alebo ďalšej inštancii toho istého skriptu) prácu so súbormi a sám končí. Ide o značne „sedliacke“ riešenie, no funkčné a použiteľné.

Druhá možnosť je použiť funkciu (nebojte sa, už je to posledná, o ktorej si dnes povieme ;-)) na „uzamknutie“ súborov. Ide o funkciu `flock()`. Prvým parametrom je file pointer, takže pred uzamknutím súboru je potrebné najprv ho otvoriť. Druhým parametrom je číselný kód operácie, ktorú chcete na súbore vykonať:

1 – zdieľaný prístup, sprístupní súbor na čítanie

2 – exkluzívny prístup, sprístupní súbor na zápis

3 – uvoľní (odomkne) súbor uzamknutý predchádzajúcim volaním funkcie flock()s parametrom 1 alebo 2

Takže ak chceme čítať zo súboru, otvoríme ho a následne takto uzamkneme a odomkneme:

```
flock($subor,1);
.
.
práca so súborom (èítanie)
.
.
flock($subor,3);
```

Nezabudnite, že ak chcete, aby bol systém používajúci `flock()` funkčný, musia túto funkciu používať všetky skripty, inak nemá význam.

Tak to by bolo nadnes všetko, nabudúce si povieme niečo o formulároch a možno sa mi podarí vtesnať aj nejaký praktický príklad, v ktorom by sme využili doposiaľ získané vedomosti. Zatiaľ si v pohode užite sviatky.

Štvrtá časť: PHP a formuláre

V novom roku 2001 sa stretávame pri štvrtej časti seriálu o programovaní v PHP. Tentoraz bude reč o formulároch a spracovaní dát z nich PHP skriptami.

Skripty na strane servera slúžia na zlepšenie možností interakcie medzi používateľom a serverom. Skripty môžu generovať stránky z dát v databázach podľa kritérií zadaných používateľom, ale aj ukladať dáta získané od používateľov do databázy, kde sú potom k dispozícii tomu, pre koho sú určené (aspoň by to teda malo tak byť, aj keď sa to možno hackerom nepáči ;-).

Pred tým, než začneme používať PHP skripty na spracovanie dát z formulárov, musíme si osvojiť pravidlá písania HTML kódu. Podstatné je pomenúvať všetky položky formulárov. V prípade, že ste zvykli nepomenúvať položky, rýchlo sa tohto zlozvyku zbavte. Mená položiek sú dôležité na ďalšie spracovanie v PHP skripte. Nemali by ste zabudnúť na kontrolu formulára ešte pred odoslaním. Zabránite tak odoslaniu chybne vyplneného formulára a ušetríte tým aj používateľovi dosť času. Táto kontrola sa vykonáva väčšinou pomocou JavaScriptu, pričom však nie je vždy možné zabezpečiť 100-percentné ošetrenie vstupu (dokonca používateľ môže JavaScript vypnúť), takže je potrebné vykonať kontrolu správnosti aj v samotnom PHP skripte.

Princíp spracovania dát z formulára v PHP je značne jednoduchý a efektívny. Ešte treba pripomenúť, že dáta z formulárov možno posielať dvoma metódami: GET a POST. Metóda GET funguje tak, že pripojí na koniec URL adresy, na ktorú sa dáta posielané, reťazec, ktorý obsahuje zadané dáta. Za URL adresou nasleduje znak ? (otáznik) a za ním vyplnené dáta z formulára, pričom v reťazci sa nachádza meno položky a jej hodnota vo forme meno=hodnota. Takéto jednotlivé položky sú oddelené znakom & (ampresand), takže výsledná URL má približne takúto podobu:

http://www.server.sk/skript.php?meno=Peter&vek=17&skola=gymnazium

Takto nejako bude vyzerať URL, keď vyplníme vo formulári položky meno, vek a škola a tento formulár pošleme na spracovanie skriptu skript.php. Pravdepodobne viete, akým spôsobom sa definuje skript, na ktorý majú smerovať dáta z vyplneného formulára po odoslaní. Pre úplnosť to však pripomeniem, opakovanie je predsa matka múdrosti ;-) – v tagu FORM definujeme parameter ACTION, ktorého hodnotou je reťazec obsahujúci umiestnenie (URL) skriptu, ktorý má dáta spracovať. Teraz sa vrátime k druhej metóde, teda metóde POST. Tá sa väčšinou viac odporúča na odosielanie dát vo formulári, pretože nepripája nijaké vyplnené dáta za adresu URL, čo znamená vyššiu bezpečnosť a aj „úhľadnejšie“ riešenie. Z pohľadu bezpečnosti je dôležité to, že požiadavky vášho browsera na server (teda URL adresy) sú viditeľné pre administrátora servera a vlastne pre hocikoho, kto má k tomuto serveru fyzický prístup. To znamená, že pri prenose informácií metódou GET môže vidieť aj dáta obsiahnuté v URL adrese, teda dáta vyplnené používateľom. To je, ako mi iste dáte za pravdu, nežiaduce, hlavne v prípadoch, že sa prenášajú osobné a citlivé údaje, ako môže byť napríklad prístupové heslo. Pri použití metódy POST takéto nebezpečenstvo nehrozí, pretože dáta v URL adrese nie sú, a teda nemôžu byť ani viditeľné v požiadavke na serveri.

Príklad

Ako príklad nám poslúži jednoduchý formulár so 4 položkami, ktorý odošleme na spracovanie skriptu. Na začiatok nám bude stačiť, ak skript vypíše na obrazovku odoslané dáta. Takže tu je formulár v HTML:

```
<SCRIPT LANGUAGE="JavaScript">

function check() {
 if (document.formular.meno.value=="") {
        alert('Vyplòte prosím vaše meno');
        document.formular.meno.focus();
        return false;
 }
 if (document.formular.priezvisko.value=="") {
        alert('Vyplòte prosím vaše priezvisko');
        document.formular.priezvisko.focus();
        return false;
 }
 if (document.formular.vek.value=="") {
        alert('Vyplòte prosím váš vek');
        document.formular.vek.focus();
        return false;
 }
 if (document.formular.adresa.value=="") {
        alert('Vyplòte prosím vašu adresu');
        document.formular.adresa.focus();
        return false;
 }
}

</SCRIPT>

<FORM NAME="formular" ACTION="spracuj.php" METHOD=POST
onSubmit="return check()">
Meno: <INPUT TYPE="text" NAME="meno" SIZE="15"
MAXLENGTH="20"></INPUT><BR>
Priezvisko: <INPUT TYPE="text" NAME="priezvisko" SIZE="15"
MAXLENGTH="20"></INPUT><BR>
Vek: <INPUT TYPE="text" NAME="vek" SIZE="2"
MAXLENGTH="2"></INPUT><BR>
Adresa: <INPUT TYPE="text" NAME="adresa" SIZE="30"
MAXLENGTH="50"></INPUT><BR>
<INPUT TYPE="submit" VALUE="Pošli"></INPUT>
</FORM>
```

Tak poďme k rozboru. Na začiatku je v tagu SCRIPT napísaná funkcia `check()` v JavaScripte, ktorá slúži na kontrolu nevyplnených polí vo formulári. Funkcia sa volá vždy pri pokuse o odoslanie formulára a nedovolí ho odoslať, ak nie sú vyplnené všetky položky. Vzhľadom na to, že toto je seriál o programovaní v PHP, necháme už JavaScript na pokoji. Ako ste si mohli všimnúť, formulár má štyri položky, pomenované meno, priezvisko, vek a adresa. Na odoslanie sa použije metóda POST a spracúvať dáta bude formulár spracuj.php. Ako teda týmto skriptom spracovať dáta? PHP poskytuje jednoduchý spôsob: po odoslaní dát má skript k dispozícii premenné, ktorých meno je zhodné s menom položky vo formulári, s tým rozdielom, že na začiatku mena premennej, samozrejme, píšeme znak $ (dolár). Hodnoty položiek meno, priezvisko, vek a adresa budeme mať teda v skripte k dispozícii v premenných `$meno`, `$priezvisko`, `$vek` a `$adresa`. Súbor s formulárom si uložíme napr. pod názvom form.html a teraz príde ukážka, ako môže byť implementované spracovanie dát v spracuj.php:

```
<?php
 if
((!IsSet($meno))||(!IsSet($priezvisko))||(!IsSet($vek))||(
!IsSet($adresa))) {
        echo „Vyplòte prosím všetky polo ky formulára.“;
        exit;
 }
 echo „
 Zadali ste nasledujúce hodnoty:<BR>
 Meno: $meno<BR>
 Priezvisko: $priezvisko<BR>
 Vek: $vek<BR>
 Adresa: $adresa“;
?>
```

Na začiatku sa otestuje, či boli odoslané všetky položky. V prípade, že sa niekto snaží o nekorektný prístup (napr. priamym spustením skriptu), vypíše sa chybové hlásenie a beh skriptu je ukončený príkazom `exit`. Ešte musím spomenúť použitie funkcie `IsSet()`, ktorá zistí, či premenná odovzdaná ako parameter existuje. Ak áno, vráti true, inak false. Ak premenná neexistuje, znamená to, že skript nie je volaný korektne cez formulár, ale pravdepodobne bol spustený priamo a bez parametrov. V prípade, že skript má k dispozícii všetky premenné, prejde podmienkou a vykoná ďalšie príkazy, ktoré za ňou nasledujú. V tomto prípade ide o výpis odoslaných premenných, čo však slúži len ako demonštračný príklad, pretože v praxi by takýto skript asi nemal veľké uplatnenie. My však už vieme, ako používať súbory, takže prečo nezapísať dáta napríklad do nejakého súboru? Skúste túto alternatívu implementovať sami.

Zatiaľ sme si ukázali, ako odoslať textové položky formulára. Poznáme však viacero typov položiek, preto sa teraz budeme venovať tým ostatným. Jedným z typov vstupu môže byť checkbox, teda zaškrtávacie políčko. V tagu INPUT je definované ako TYPE="checkbox". Takéto políčko môže nadobúdať iba dva stavy – buď je zaškrtnuté, alebo nie je. V skripte, ktorý spracúva dáta, sa táto skutočnosť prejaví tak, že premenná charakterizujúca dané políčko buď je „setnutá“, alebo nie, teda buď existuje, alebo nie. Na overenie tejto skutočnosti nám poslúži už spomínaná funkcia `IsSet()`, ktorá zisťuje, či premenná daná ako parameter existuje, alebo nie. Ak teda checkbox zaškrtnutý bol, bude v skripte k dispozícii aj premenná s menom checkboxu a funkcia `IsSet()` vracia true a naopak, ak políčko zaškrtnuté nebolo, nie je k dispozícii ani premenná, a teda `IsSet()` vracia false. Ešte pripomínam, že meno premennej získate rovnako ako pri textových položkách, teda ak meno položky formulára je napríklad student, meno premennej v skripte bude `$student`. Aj keď na overenie zaškrtnutia nám stačí overiť existenciu premennej, môžeme v HTML formulári definovať aj hodnotu položky (checkboxu). Hodnota sa priraďuje parametrom VALUE v tagu INPUT. V prípade, že políčko bolo zaškrtnuté, máte v skripte k dispozícii nielen premennú, ale v nej aj hodnotu checkboxového poľa z formulára.

Ďalším typom, ktorým sa budeme zaoberať, je radio-button. Ide vlastne o viac zaškrtávacích políčok, pričom zaškrtnuté môže byť iba jedno. Ako príklad nám môže poslúžiť zistenie veku používateľa z formulára, kde je napríklad 5 radio položiek: menej ako 18, 18 – 25, 26 – 40, 41 – 55, viac ako 55. Definovaných je 5 políčok, ale len jedno z nich môže byť zaškrtnuté. To zabezpečíme tak, že každé políčko bude mať rovnaké meno, teda rovnakú hodnotu atribútu NAME. Políčka by sa však logicky mali líšiť svojou hodnotou čiže parametrom VALUE. Spomínaný príklad by sme mohli zapísať takto:

```
<INPUT TYPE="radio" NAME="vek" VALUE="1">menej ako
18</INPUT><BR>
<INPUT TYPE="radio" NAME="vek" VALUE="2" CHECKED>18 -
25</INPUT><BR>
<INPUT TYPE="radio" NAME="vek" VALUE="3">26 -
40</INPUT><BR>
<INPUT TYPE="radio" NAME="vek" VALUE="4">41 -
55</INPUT><BR>
<INPUT TYPE="radio" NAME="vek" VALUE="5">viac ako
55</INPUT>
```

V druhom poli si môžete všimnúť atribút CHECKED, ktorý slúži na implicitné zaškrtnutie políčka. Ak ho použijete, zabezpečíte tým, že používateľ odošle formulár, v ktorom bude určite niektorý z radio buttonov zaškrtnutý, či už ten implicitný, alebo explicitne zvolený používateľom. Ak atribút CHECKED nepoužijete, dovolíte používateľovi, aby položku nevyplnil, resp. aby nezvolil ani jeden z radio buttonov. V skripte sa potom správa odoslaná položka podobne ako checkbox. Vytvorená premenná má meno korešpondujúce s menom položky, takže v našom prípade by to bola premenná vek. Ak nebol zaškrtnutý ani jeden z radio buttonov, premenná neexistuje a funkcia IsSet() vracia false. Ak radio button zaškrtnutý bol, k dispozícii je premenná príslušného mena a jej hodnota zodpovedá hodnote parametra VALUE v príslušnom tagu INPUT. Ak by teda niekto v našom príklade zvolil vek 18 – 25, premenná $vek by obsahovala hodnotu 2.

Teraz sa budeme venovať rozbaľovacím menu. V HTML ide o tag SELECT, pričom slangovo sa tento typ položky nazýva aj combo-box. Táto položka sa líši svojím zápisom v HTML kóde od doteraz prebraných vstupov tým, že sa nedefinuje ako typ v tagu INPUT, ale má svoj vlastný špecifický tag, ako som už spomínal, SELECT. Tieto položky typu SELECT môžu byť dvojakého typu – buď je možné vybrať iba jednu položku z menu, alebo pomocou klávesu CTRL aj viacero položiek. Najprv sa budeme venovať prvému prípadu. Vtedy tag SELECT má iba jeden parameter, ktorým je NAME, kde špecifikujeme meno položky, a teda aj premennej, ktorá bude obsahovať zvolenú hodnotu v skripte spracujúcom dáta. Tag SELECT je párový, teda musí mať aj ukončovací tag. Medzi nimi sa nachádzajú jednotlivé položky menu, definované párovými tagmi OPTION. Tento tag má takisto iba jeden parameter, ktorým je VALUE, a obsahuje hodnotu pre danú položku menu. Medzi tagmi OPTION definujeme reťazec, ktorý bude prezentovaný navonok ako položka menu, takže v menu nemusia byť zákonite zobrazené hodnoty, ale vlastne ich môžete nahradiť nejakými aliasmi, ktoré používateľ uvidí ako položky menu. Na objasnenie uvádzam predchádzajúci príklad pre radio buttony, prepísaný do formy SELECT menu:

```
<SELECT NAME="vek">
<OPTION VALUE="1">menej ako 18</OPTION>
<OPTION VALUE="2" SELECTED>18 - 25</OPTION>
<OPTION VALUE="3">26 - 40</OPTION>
<OPTION VALUE="4">41 - 55</OPTION>
<OPTION VALUE="5">viac ako 55</OPTION>
</SELECT>
```

Ako vidíte, máme definovaných 5 položiek s hodnotami od 1 po 5, pričom v menu používateľ vidí to, čo je definované medzi tagmi OPTION. Atribút SELECTED slúži podobne ako CHECKED pri checkboxoch na implicitné označenie niektorej položky. Po odoslaní formulára obsahujúceho takto definované menu má skript k dispozícii premennú $vek a v nej hodnotu od 1 do 5 v závislosti od toho, ktorú položku používateľ vybral.

Trošku zložitejšia situácia nastáva vtedy, keď potrebujeme definovať SELECT menu, kde je možné zvoliť viacero položiek. Samozrejme, zložitejšie je to implementovať tak, aby dáta mohol skript spracovať. HTML zápis problémom nie je, stačí definovať atribút MULTIPLE a parameter SIZE, ktorému priradíme hodnotu v závislosti od toho, koľko chceme mať viditeľných položiek, teda aké veľké má byť menu, pokiaľ ide o počet riadkov. Tento parameter je možné definovať aj pri menu s jednou položkou, ale nemá to význam. Z toho vyplýva, že obyčajné a viacnásobné menu sa od seba líšia iba atribútom MULTIPLE. Pravda, je tu ešte jeden rozdiel, a to v mene položky. Vzhľadom na to, že označených môže byť viacero položiek, nebolo by asi najšťastnejším riešením dávať každej z nich osobitné meno. Pri vyšších počtoch položiek by asi nejednému programátorovi došla fantázia pri vymýšľaní mien premenných. Preto sa tu priam ponúka použiť nie skalárnu premennú, ale pole. Preto musíme definovať meno v parametri NAME tagu SELECT ako názov poľa, teda pridať nakoniec sekvenciu [], teda napríklad NAME="kategorie[]". Najlepšie nám to zasa vysvetlí príklad:

```
<SELECT NAME="kategorie[]" SIZE=3 MULTIPLE>
<OPTION VALUE="1">Kanc. softvér</OPTION>
<OPTION VALUE="2">Hry</OPTION>
<OPTION VALUE="3">Oper. systémy</OPTION>
<OPTION VALUE="4">Databázy</OPTION>
<OPTION VALUE="5">Server. aplikácie</OPTION>
<OPTION VALUE="6">Archivaèný softvér</OPTION>
<OPTION VALUE="7">Multimédiá</OPTION>
</SELECT>
```

V takomto menu môžeme vybrať ľubovoľný počet položiek, pričom počet naraz zobrazených položiek je 3 (dané parametrom SIZE). Ak odošleme formulár obsahujúci takéto menu, v skripte máme k dispozícii pole `$kategorie`, ktorého počet prvkov je rovný počtu vybraných položiek v menu. Prvky sú indexované od nuly a obsahujú hodnoty vybraných položiek. Ak teda vyberieme z menu položky Hry, Databázy a Multimédiá, bude mať pole $kategorie tri prvky, pričom prvok `$kategorie[0]` bude obsahovať hodnotu 2, `$kategorie[1]` hodnotu 4 a `$kategorie[2]` hodnotu 7. Na zistenie počtu prvkov v poli môžeme použiť funkciu count(), ktorá vracia počet prvkov poľa, ktorého meno jej treba zadať ako jediný parameter, teda v tomto prípade count($kategorie).

Z klasických formulárových položiek sme ešte nespomenuli položku TEXTAREA, čo je vlastne dvojrozmerný textový box. Od položky typu TEXT sa teda líši iba tým, že má viac ako jeden riadok a nezapisuje sa ako typ v parametri tagu INPUT, ale existuje pre ňu špeciálny tag TEXTAREA. V ňom definujete parametrom ROWS počet riadkov a COLS počet stĺpcov (počet znakov v riadku). Ďalej je potrebné parametrom NAME priradiť položke nejaké meno, napr. message. Tag je párový, teda musí obsahovať uzatvárací tag. Medzi ne je možné vložiť nejakú implicitnú hodnotu (reťazec). Po odoslaní sa položka správa podobne ako ostatné doteraz spomenuté. To znamená, že skript má k dispozícii premennú rovnakého mena (`$message`) a v nej uložený reťazec, zadaný používateľom.

Upload súborov

No a teraz prejdeme na lahôdku pri odosielaní formulárových dát. Položka definovaná ako tag INPUT typu FILE (TYPE="FILE") je síce iba text obsahujúci meno súboru, no v skutočnosti znamená, že daný súbor chceme uploadovať. Upload súborov je trošku zložitejšia záležitosť než doposiaľ spomínané položky, aj keď to nie je nič náročné. Po odoslaní formulára s položkou definovanou ako súbor má skript k dispozícii viacero premenných, samozrejme, v prípade, že položka bola vyplnená. Prvou je premenná s rovnakým názvom, ako bolo meno položky vo formulári, so znakom $ na začiatku (tak ako to bolo v doterajších prípadoch). Táto premenná obsahuje meno súboru, pod ktorým je uploadovaný súbor uložený na serveri. Vďaka tomu ho môžeme čítať, kopírovať, presúvať, či mazať – môžete využiť vedomosti nadobudnuté v predošlej časti o PHP a jeho filesysteme. Pokiaľ súbor nezmažete, bude zmazaný automaticky po tom, čo naň zaniknú akékoľvek požiadavky na prístup, spravidla po skončení behu skriptu. Ďalšie premenné sú od neho odvodené. Prvú získame pridaním „_name" za meno spomínanej premennej a v tejto je uložené meno, pod ktorým je súbor reprezentovaný na systéme odosielateľa. Ďalšou premennou je tá, ktorú získame pridaním „_size" za meno pôvodnej premennej a v nej je uložená veľkosť uploadovaného súboru v bajtoch. A poslednú premennú získame pridaním „_type" a v tej máme zase uložený MIME typ uploadovaného súboru (napr. „image/gif"). Vďaka týmto premenným môžete ľubovoľne špecifikovať, aké súbory akej veľkosti môže používateľ uploadovať. Verím, že to bolo vysvetlené dosť zložito, preto uvádzam príklad na objasnenie. Definujme si položku:

```
<INPUT TYPE="file" NAME="userfile"></INPUT>
```

Potom budeme mať v skripte spracúvajúcom dáta tieto premenné:

```
$userfile – meno, pod ktorým je súbor doèasne (do zániku
po iadavky) na serveri
$userfile_name – meno, pod ktorým bol súbor umiestnený
v systéme pou ívate¾a
$userfile_size – ve¾kos  súboru v bajtoch
$userfile_type – MIME typ súboru
```

Ešte pripomínam, že vo formulári v tagu FORM treba definovať parameter ENCTYPE, ktorému je potrebné priradiť hodnotu „multipart/form-data" (ENCTYPE="multipart/form-data"). V prípade, že tak neurobíte, nemusí vám upload fungovať.

To je zatiaľ všetko, nabudúce sa konečne dostaneme k väčšiemu príkladu, na ktorom budete môcť uplatniť doteraz nadobudnuté vedomosti.

Piata časť

Vitajte pri ďalšom pokračovaní seriálu o programovaní v PHP, tentoraz už piatom. V predchádzajúcich častiach sme sa naučili syntax jazyka, oboznámili sme sa s niekoľkými funkciami, osobitne s filesystemom, a naučili sa pracovať s dátami z formulárov. Teraz by som sa rád zameral na praktické využitie týchto vedomostí a zadal úlohu, ktorú budeme postupne riešiť.

Zadanie

Na to, aby sme mohli nejakú úlohu vyriešiť, potrebujeme, samozrejme, zadanie. Rozhodol som sa pre jednoduché diskusné fórum (alebo ak chcete, auditórium). Pre tých, ktorí nevedia, o čo ide, prinášam malé vysvetlenie – ide o program, ktorý umožní používateľom vymieňať si názory a diskutovať prostredníctvom internetu. Takéto programy obyčajne majú nejaký prihlasovací a registračný formulár. Používateľ sa cez registračný formulár zaregistruje a prostredníctvom prihlasovacieho sa prihlási. Potom môže ľubovoľne prispievať svojimi príspevkami do diskusie. My si registrovanie a prihlasovanie používateľov zatiaľ ešte odpustíme a budeme sa venovať diskusii bez nevyhnutnosti registrácie používateľov. Postačí, ak budeme mať stránku s formulárom, ktorý bude obsahovať položky Nick, E-mail a Správa. Použitie je zrejmé – Nick je prezývka používateľa, E-mail je jeho e-mailová adresa a Správa je príspevok do diskusného fóra. Tento formulár bude na vrchu stránky a pod ním sa budú nachádzať vypísané správy. Samozrejme, správ môže byť veľa, preto obmedzíme ich počet na 400. Vypisovať toľko správ na jednu stránku však nie je príliš vhodné, takže obmedzíme počet na 20 správ na stránku. Medzi stránkami bude používateľ ľubovoľne listovať, pričom bude mať k dispozícii šípky na posun hore, dole, úplne hore a úplne dole v zozname správ. Na ukladanie správ použijeme filesystem, takže je najlepšie navrhnúť nejaký spôsob (formát), ako budú údaje ukladané do súborov. Takisto nesmiete zabúdať na problém zdieľanej práce so súbormi. Toľko k zadaniu, poďme sa pozrieť na riešenie.

Analýza

Najprv je potrebné navrhnúť nejakú štruktúru súborov, do ktorých budeme ukladať dáta, a takisto formát, v akom ich budeme ukladať. Vzhľadom na to, že nepotrebujeme uchovávať nijaké špecifické informácie o používateľovi, ktorý zaslal správu, ani o čase jej zaslania, bude asi najjednoduchším spôsobom ukladať celú správu aj s používateľom, časom, prípadne inými položkami do jedného riadka, najlepšie priamo v HTML kóde. Riadky nebudú nijako indexované, preto si určíme, že pozícia daného riadka v súbore bude aj indexom daného odkazu. Zostáva sa iba rozhodnúť, či budú riadky indexované zhora alebo zdola. Aj keby sa mohlo zdať, že logické je indexovať zhora (od prvého riadka k poslednému), bude to presne naopak. Dôvod je jednoduchý – pri dopisovaní riadka do súboru nie je možné zapísať ho na začiatok tak, aby sa ostatné posunuli dole, museli by sme celý súbor vždy nanovo prepisovať. Preto je lepšie dopisovať riadok na koniec a potom pole tvorené riadkami otočiť pre potreby výpisu. Prepis súboru však bude nevyhnutný v prípade, že budeme chcieť vymazávať najstaršie správy, teda riadky s najvyšším indexom, t. j. zo začiatku súboru.

Takisto predpokladáme, že budeme mať nejakú premennú `$jump`, ktorá znamená skok v zozname správ, resp. index prvej vypisovanej správy (indexujeme od 0), čo nám umožní vypisovať aj staršie správy pri konštantnom počte vypisovaných správ. Na inicializáciu je potrebné nastaviť hodnotu premennej `$jump` na 0, teda iba v prípade, že premenná nie je inicializovaná, čo uskutočníme použitím funkcie `IsSet()`. Potom nesmieme zabudnúť zabudovať skrytý parameter jump do HTML kódu. Takže poďme kódovať.

Skok po formulári

Ošetrenie premennej `$jump` je jednoduchá záležitosť, ale treba to urobiť na začiatku, pretože ju budeme potrebovať hneď v HTML formulári:

```
<?
 if (!IsSet($jump)) $jump=0;
?>
```

HTML návrh

Základom je vytvoriť si vhodnú HTML štruktúru, ktorú neskôr sfunkčníme PHP skriptami. Preto si vytvoríme HTML návrh formulára a vzhľadu stránky. Pre jednoduchosť nebudeme používať nijaké grafické objekty, iba čistý formátovaný text, napríklad môžeme uvažovať s modrou farbou liniek, čiernou farbou textu a bielou farbou pozadia. Celkový kód diskusného fóra je rozsekaný na časti, preto neskôr budem v ostatných kódoch pre úsporu miesta odkazovať na ostatné časti kódu len ich názvom podľa ich kapitol. Tu je HTML návrh formulára, časť kódu s rovnakým názvom:

```
<body bgcolor="white" text="black" link="#8888ff"
alink="#8888ff" vlink="#8888ff">
<center>
<font color=#aaaaff face="Verdana, Helvetica"
size=4>Diskusné fórum</font><br><br>
<table border=0 cellpadding=0 cellspacing=5>
<form name="data" METHOD=POST action="index.php">
<input type="hidden" name="jump" value="0"></input>
<tr>
<td align=center valign=middle rowspan=4><textarea
name="message" cols=30 rows=5></textarea></td>
</tr>
<tr>
<td align=left valign=middle><font face="Arial"
color=#ffcc00 size=3>Nick</font></td>
<td align=left valign=middle><input type=text name="nick"
size=10 maxlength=40></input></td>
</tr>
<tr>
<td align=left valign=middle><font face="Arial"
color=#ffcc00 size=3>E-mail</font></td>
<td align=left valign=middle><input type=text name="mail"
size=10 maxlength=30></input></td>
</tr>
<tr>
<td valign=middle align=center colspan=2><input type="sub-
mit" value="Pošli"></input></td>
</tr>
</form>
</table>
```

Hlavičku HTML štruktúry iste dokážete doplniť sami.

Ako vidíte, už máme zadefinovaný skrytý parameter jump, ktorého funkcia je opísaná v analýze. Zápis, ktorý sme ho dostali do HTML kódu, by vám tiež nemal byť neznámy, mali by ste si ho pamätať zo začiatku tohto seriálu. Tento zápis sa často používa, a ak sa raz budete venovať profesionálne programovaniu v PHP, určite sa s ním neraz stretnete.

Takýto HTML kód je síce nekorektný z hľadiska HTML špecifikácie (neukončené tagy HTML a BODY, chýbajúci tag HEAD), ale inak je úplne funkčný. Ak si ho uložíte pod názvom index.php, môžeme začať robiť prvé pokusy. Ako vidíte, formulár odkazuje na súbor index.php, teda sám na seba. To znamená, že samotný skript, ktorý generuje HTML stránku, by sa mal postarať o údaje, ktoré mu používateľ zaslal (samozrejme, iba v prípade, že naozaj nejaké zaslal, mohlo sa stať, že nezaslal nijaké). Toto nám osvetlí ďalšia časť kódu.

Zápis dát

V našom príklade nám bohato postačí jediný súbor, ktorý pomenujeme napríklad data.msg (netreba ho vytvárať, skript si ho vytvorí v prípade, že neexistuje). Do tohto súboru zapíšeme odoslané dáta v prípade, že používateľ odoslal nejaké dáta prostredníctvom formulára. To zabezpečíme kontrolou niektorého zo vstupných premenných, napríklad `$message`. Na to nám poslúži funkcia `IsSet()` a takisto je vhodná kontrola, či nie je v premennej `$message` (ak existuje) prázdna hodnota. V tele podmienky stačí potom uskutočniť zápis do súboru. Pred zápisom by bolo vhodné urobiť úpravu odkazu, ako je napríklad odstránenie HTML tagov, čím zabezpečíme to, aby používatelia nemohli ľubovoľne pomocou tagov manipulovať so stránkou. To zaistíme nahradením znakov < a > sekvenciami < a >, ktoré predstavujú entity týchto znakov. Nahrádzanie v reťazci zabezpečí funkcia `str_replace()`, ktorej prvý parameter predstavuje znak(y), ktorý chceme nahradiť, druhý znak(y), ktorým chceme nahradzovať, a posledným je reťazec, na ktorom bude operácia vykonaná. Výsledok funkcie je nový reťazec, ktorý treba uložiť do premennej, môže byť aj do tej istej ako ten, ktorý je tretím parametrom funkcie. Keďže celý odkaz zahŕňa aj meno používateľa, e-mail i dátum, je potrebné zakomponovať tieto údaje do správy. Všetko je okomentované priamo v kóde. Takže nasleduje kód zápisu dát:

```
<?
if ((IsSet($message))&&($message!="„))
{
 $datum=getdate();                          //naèítanie
aktuálneho dátumu
 if ($datum[„hours“]<10)
$datum[„hours“]="0".$datum[„hours“];   //formátovanie
dátumu
 if ($datum[„minutes“]<10)
$datum[„minutes“]="0".$datum[„minutes“];
 $fdatum=$datum[„hours“].":".$datum[„minutes“]." -
„.$datum['mday'].".".$datum['mon'].".".$datum['year'];
//formátovaný dátum pre správu
 $message=str_replace(„<“,“&lt;“,$message);     //preè s
HTML tagmi
 $message=str_replace(„>“,“&gt;“,$message);
```

```
 $sprava="<a href=\"mailto:".$mail."\"
class=\"amsg\">".$nick."</a> <span class=\"casboard\"> -
$fdatum</span><br><span
class=\"msg\">".$message."</span><br><br>";  //skomponu-
jeme správu
 $subor=fopen(„data.msg","a");           //otvoríme súbor
na zápis a nastavíme sa na jeho koniec
 flock($subor,2);                        //uzamkneme súbor
(exclusive lock - zápis)
 fputs($subor,$sprava."\n");             //zapíšeme riadok
do súboru a vlo íme znak koniec riadka
 flock($subor,3);                        //odomkneme súbor
 fclose($subor);                  //zatvoríme súbor
}
?>
```

Poďme si ozrejmiť operácie, ktoré by sa vám mohli zdať nejasné. Na získanie aktuálneho dátumu sme použili funkciu getdate(), ktorá bez parametrov vracia asociatívne pole s aktuálnym dátumom, pričom premenná, ktorej priradíte výsledok funkcie, nadobudne indexy (resp. kľúče) s nasledujúcimi hodnotami:

Kľúč	Hodnota
year	aktuálny rok
mon	aktuálny mesiac (číselne)
mday	aktuálny deň v mesiaci (číselne)
hours	aktuálny čas – hodina
minutes	aktuálny čas – minúta

Takže ak chceme získať z poľa rok, získame ho z položky **$datum[„year"]**. Funkcia getdate() vracia viac hodnôt, ktoré zahŕňajú okrem iného sekundy, poradie dňa v týždni a roku či názov mesiaca a dňa v týždni. Tieto hodnoty však zatiaľ nepotrebujeme a vystačíme si s tými v tabuľke. Hodnoty obsahujúce minúty a hodiny sme ešte sformátovali preto, aby sme dosiahli dvojmiestne hodnoty tých premenných. Celý dátum potom už v podobe, v akej bude v správe, teda formátovaný, vložíme do premennej `$fdatum`. Nasleduje eliminovanie HTML tagov pomocou už spomínanej funkcie `str_replace()`. No a nakoniec skomponujeme správu ako reťazec, ktorý uložíme do premennej $sprava. Dosiahneme tým formátovanú podobu správy, kde bude meno používateľa aktívnym linkom na jeho e-mail a pod ním sa bude nachádzať samotná správa. Ako vidíte, sú použité kaskádové štýly, preto je potrebné ich dodefinovať (tie, ktoré sú uvedené parametrom class). Či už to urobíte priamo vložením do hlavičky, alebo prilinkovaním externého CSS súboru, to nechám na vás. No a na konci časti kódu prebehne zápis do samotného súboru. Ten by vám mal byť už dostatočne známy z predchádzajúcich častí seriálu a takisto z komentára, takže len v skratke: otvoríme súbor `data.msg` metódou „a", teda na zápis a nastavenie ukazovateľa na koniec súboru. Súbor uzamkneme pomocou funkcie `flock()` a vyhradíme si naň právo zápisu, teda nebudeme ho zdieľať. Potom doň zapíšeme skomponovanú správu spolu so znakom konca riadka (\n). Následne súbor odmokneme a zatvoríme.

Kontrola počtu správ

Po zapísaní správy by bolo vhodné skontrolovať celkový počet správ, resp. riadkov v súbore. Ako sme na začiatku stanovili, maximálnou hranicou bude 400 správ. Kontrolu počtu správ uvádzam ako samostatný blok kódu, nič by sa však nestalo, keby ste tento blok vsunuli do bloku, kde sa zapisuje správa, priamo do podmienky. Dôvod je jednoduchý: je jasné, že počet správ stačí kontrolovať iba po zapísaní novej správy, pretože len vtedy môže byť prekročený limit. Ak sa nič nezapisuje, je kontrola zbytočná. To však už nechám na doladenie vami, čitateľmi. Poďme k priebehu kontroly. Tá bude mať nasledujúci sled: najprv načítame všetky riadky do jednej premennej typu pole pomocou funkcie `file()`. Nazvime toto pole napríklad `$riadky`. Počet prvkov tohto poľa po prečítaní súboru predstavuje počet riadkov, ktoré sú v súbore zapísané. Potom stačí použiť funkciu `count()`, ktorá vracia počet prvkov poľa zadaného ako parameter, a tak zistiť počet riadkov, ktoré obsahuje súbor. V prípade, že počet riadkov presiahol 400 (to znamená, že výraz `count($riadky)>400` zadáme ako podmienku), treba vykonať zmazanie starých správ. To môžeme uskutočniť napríklad tak, že zapíšeme do súboru riadky, ktoré neprekračujú svojím indexom limit. Ak teda bude počet riadkov 402, zapíšeme všetky riadky od indexu 2 vrátane (nezabúdajte, že sa indexuje od nuly). Najjednoduchší spôsob, ako toto všetko urobiť, je súbor otvoriť na zápis tak, aby sa vytvoril nový a zmazalo sa z neho všetko, čo tam je, a následne sa zapísali dané riadky. Nasleduje kód na kontrolu počtu správ:

```
<?
 $riadky=file(„data.msg");       //naèítame riadky do po¾a
 if (count($riadky)>400) {       //a ak je poèet vyšší ako
400, zma eme staré správy
        $subor=fopen(„data.msg","w");
        flock($subor,2); //uzamkneme súbor
        for ($i=count($riadky)-
400;$i<=count($riadky);$i++) {
                fputs($subor,$riadky[$i]);
//zapíšeme riadky do súboru
        }
        flock($subor,3);         //odomkneme a zatvoríme
súbor
        fclose($subor);
 }
?>
```

Výpis správ

Takže teraz je na rade asi to najdôležitejšie z hľadiska používateľa, ale asi aj najjednoduchšie z hľadiska programátora. Pretože ak sme ukladali do súboru správne formátované riadky, ktoré už netreba inak upravovať, stačí ich iba vypísať do HTML dokumentu. Najjednoduchšie by to asi bolo pomocou funkcie `readfile()`. No tu je problém s tým, že táto funkcia vypíše do výstupu celý súbor. Preto musíme napísať nejaký kód, ktorý zabezpečí vypísanie iba určitých riadkov, pričom nesmieme zabúdať na to, že riadky sú indexované od posledného (najspodnejšieho) riadka k prvému (najvrchnejšiemu) riadku. Rovnako nesmieme zabudnúť na použitie premennej `$jump`, ktorá nám udáva index prvého riadka, od ktorého budeme vypisovať. Riadky sú indexované od nuly, čomu zodpovedá aj počiatočná hodnota premennej `$jump`. Pri výpise by však nultý riadok mal byť prezentovaný ako prvý, a teda aj všetky ostatné by mali mať index zvýšený o 1, pretože nie je príliš vhodné používateľovi prezentovať niektorý riadok ako nultý či vypisovať „Vypisujem riadky 0 – 9 zo 129"... Pred samotným výpisom je ešte potrebné vykonať kontrolu premennej `$jump`, či náhodou neobsahuje záporné číslo alebo či jej hodnota nepresahuje počet riadkov v súbore. Nasleduje kód výpisu správ spolu s formátovacími tagmi:

```
<br><hr color=#ddaa00 size=2>
<p align=justify>
<?
 $vypis=file(„data.msg");
 if ($jump>=count($vypis)) $jump=$count($vypis)-20;
 if ($jump<0) $jump=0;
 $koniec=$jump+20;
 if ($koniec>count($vypis)) $koniec=count($vypis);
 for ($i=count($vypis)-$jump-1;$i>=count($vypis)-
$koniec;$i—) {
        echo $vypis[$i];
 }
?>
</p>
```

Načítanie súboru je asi všetkým jasné. Nasleduje ošetrenie premennej `$jump`, ktorá, ak je menšia ako nula, nadobudne hodnotu nula. Toto sa môže stať v prípade, že používateľ bol o jednu stránku ďalej a posunul sa o jednu k novším správam, ktorých mohlo byť menej ako 20 (k tomu sa o chvíľu dostaneme). V prípade, že je premenná `$jump` väčšia alebo rovná počtu správ, je potrebné ju zmenšiť, ideálne tak, aby posledná správa na stránke bola zároveň poslednou v súbore (myslené z časového hľadiska, z hľadiska polohy ide o prvú správu v súbore). To dosiahneme priradením hodnoty o 20 menšej, ako je počet správ v súbore. Takisto je vhodné určiť si poslednú správu, ktorú budeme vypisovať. Tou bude dvadsiata správa od prvej vypisovanej. Preto bude hodnota premennej $koniec väčšia o 20 ako `$jump`. Jej použitie by sa mohlo zdať zbytočné, veď načo nám je premenná `$koniec`, keď je to iba `$jump` zväčšená o 20. Lenže sa môže stať, že počet správ za aktuálnym skokom v premennej `$jump` je menší ako 20. Preto nasleduje ošetrenie premennej `$koniec`, aby nepresahovala počet riadkov daný počtom prvkov poľa `$vypis`. Nasleduje cyklus vypisujúci správy, ktorý sa začína na pozícii `$jump`, ale keď rátame od konca, pretože najnovšie riadky sú na konci. Preto postupujeme od konca poľa k začiatku, pričom sa vypíšu správy s indexom menším alebo rovným rozdielu počtu prvkov poľa a premennej `$koniec`. Znižovanie riadiacej premennej `$i` je zabezpečené priamo v príkaze výpisu, pričom najprv je vrátená hodnota `$i` na výpis a následne je hodnota v `$i` dekrementovaná po každej iterácii.

Pohyb v zozname správ

Vzhľadom na to, že správ bude viac, ako ich bude vypísaných na jednej stránke, bude potrebné nejako zabezpečiť pohyb po stránke. To sa bude diať pomocou premennej `$jump`, ako sme si už vysvetlili. Pre našu úlohu nám postačí, ak na stránke bude zobrazený rozsah vypisovaných správ z celkového počtu a ďalej šípky s možnosťou pohybu vpred, vzad, na začiatok a na koniec. Všetky údaje už máme k dispozícii po predchádzajúcej časti kódu vďaka premenným `$jump`, `$koniec` a poľu `$vypis`. Nasleduje posledná časť kódu, kód pohybu v zozname správ:

```
<?
 if ($jump>0) echo „<a href='index.php?jump=0'><font
size=2 face='Arial'
color='#8888ff'>&lt;&lt;</font></a>  
                    <a href='index.php?jump=“.($jump-
20).“'><font size=2 face='Arial'
color='#8888ff'>&lt;</font></a>“;
 echo „ Vypísané správy „.($jump+1).“ - $koniec z
„.count($vypis).“ “;
 if ($koniec<count($vypis)) echo „<a
href='index.php?jump=“.($jump+20).“'><font size=2
face='Arial' color='#8888ff'>&gt;</font></a>  
                                    <a
href='index.php?jump=“.(count($vypis)-20).“'><font size=2
face='Arial' color='#8888ff'>&gt;&gt;</font></a>“;
?>
```

Tento kód nám vypíše správu o tom, ktoré správy z akého počtu sú vypísané. Takisto zobrazí šípku a dvojitú šípku doľava v prípade, že prvá vypisovaná správa nie je prvou v súbore, resp. ak existujú ešte novšie správy než tie, čo sú vypísané. Rovnako ak existujú staršie správy než tie, čo sú zobrazené, vypíšu sa šípka a dvojitá šípka doprava. Jednoduchá šípka znamená vždy posun o jednu stránku, dvojitá vždy na začiatok, resp. na koniec zoznamu správ. Všetko spočíva v správnom definovaní hodnoty `$jump`.

Teraz stačí celý kód pospájať v takom poradí, ako ho máme napísaný, teda aký je sled v tomto článku: najprv ošetrenie premennej `$jump`, potom HTML návrh formulára, zápis dát, kontrola počtu správ, výpis správ a nakoniec pohyb v zozname správ. Takýto kód je už úplne funkčný, takže po pomenovaní súboru index.php môžete vykonávať prvé testy.

Keď si kód odskúšate, odporúčam vám doplniť chýbajúce HTML tagy, CSS štýly. Ďalej skúste vymyslieť, ako by bolo možné zobraziť informácie o počte správ a šípky na pohyb v zozname správ aj medzi formulárom a začiatkom výpisu správ (teda na začiatku stránky). Takisto môžete implementovať možnosť pohybu po zozname správ pomocou zobrazenia čísel, z ktorých každé bude znamenať 20 správ, teda napr. kliknutím na číslo 3 sa zobrazí 20 správ na tretej stránke od začiatku. Úloha má veľa alternatív, takže vždy bude čo dorábať.

Nateraz je to už všetko, želám vám úspešné kódovanie a málo preklepov ;-).

Šiesta časť: PHP a SQL databázy

Vítam vás už pri záverečnej časti nášho seriálu o PHP. Tentoraz si povieme o databázach SQL, ktoré robia z PHP mocný nástroj na tvorbu dynamických WWW. Nevyhnutnosťou skriptov na tvorbu dynamických WWW stránok je možnosť ukladať dáta na vzdialenom serveri a takisto ich poskytovať používateľom. Jednou možnosťou je uložiť dáta do súborov pomocou filesystemu, ako sme si ukázali v jednej z predchádzajúcich častí nášho seriálu. Ak ste poctivo skúšali programovať s použitím filesystemu, iste ste prišli na to, že je to dosť zložitá a pritom nie príliš bezpečná ani rýchla záležitosť. A práve tento problém riešia databázy SQL, ktorých podpora je v PHP implementovaná. Databázy SQL slúžia na uchovávanie dát, pričom na prácu s nimi existujú funkcie podporované priamo v PHP. V podstate máte dve možnosti prístupu k databáze: buď pomocou driverov ODBC, alebo priamo cez natívne drivery danej databázy SQL. Výhodou používania databáz SQL v PHP je predovšetkým to, že sa nemusíte starať o zápis súborov na disk a čítanie, ako to bolo pri filesysteme. Databáza SQL urobí všetko za vás, pričom je aj zabezpečená ochrana pred cudzím zápisom na disk, čo v prípade filesystemu je asi najväčším nebezpečenstvom. Takisto použitie databázy SQL značne ovplyvňuje rýchlosť exekúcie skriptu v pozitívnom smere a zároveň sa nemusíte starať o zdieľanie súborov ako pri filesysteme, pretože aj o to sa postará databáza SQL. Najpoužívanejšou databázou medzi bežnými používateľmi a na nekomerčných serveroch je MySQL. Táto databáza je k dispozícii zadarmo, výnimkou je verzia pre Windows, ktorá je zadarmo iba 30 dní, teda je k dispozícii ako shareware. MySQL si môžete stiahnuť z domovskej stránky www.mysql.com, prípadne niektorého mirroru. Dajte si však pri tom pozor, aby ste stiahli verziu pre svoj OS. Inštaláciou ani syntaxou MySQL sa zaoberať nebudeme vzhľadom na to, že na stránkach PC REVUE nájdete takisto seriál o MySQL, kde sa nachádzajú všetky potrebné informácie. Ešte podotknem, že existujú aj ďalšie databázy SQL, ako napríklad Microsoft SQL či PostgreSQL, ktorých podpora je v PHP takisto implementovaná. Funkcie na použitie týchto databáz sú podobné ako pre MySQL, líšia sa iba v predpone, ktorá vždy charakterizuje danú databázu (napr. MySQL_Connect(), ktorej obdoba pre Microsoft SQL bude MsSQL_Connect(), pre PostgreSQL zasa pg_connect()). Samozrejme, podobnosť nie je absolútna, každá databáza SQL má svoje špecifiká. PHP podporuje aj ODBC, ale ako som uviedol, budeme sa venovať iba natívnym driverom a databáze MySQL.

Konfigurácia

Po nainštalovaní a spustení MySQL by ste mali mať možnosť hneď ju v PHP používať. Pokiaľ máte niektorú zo starších verzií PHP, kde podpora MySQL nie je súčasťou štandardného balíčka, musíte prekonfigurovať súbor php.ini. V tomto súbore nájdite riadok
;extension=php_mysql.dll
a zmažte znak „;“ z jeho začiatku. Súbor uložte a skúste následne reštartovať server daemon. Teraz by vám MySQL malo fungovať korektne. Ak stále nefunguje, musíte si skúsiť stiahnuť zo stránky PHP súbor php_mysql.dll, ale radšej by som vám odporúčal stiahnutie najnovšieho balíka, ktorý obsahuje aj priamu podporu MySQL a reinštaláciu PHP. Najlepšie otestujete, či vám MySQL funguje korektne pod PHP, ak napíšete jednoduchý skript (stačí iba pripojenie k DB) a skúsite ho spustiť. Ak sa vypíšu chybové hlásenia, nastala niekde chyba.

Základné funkcie

Základnými funkciami MySQL v PHP sú funkcie, ktoré slúžia na pripojenie a odpojenie databázy. Databázu je možné používať až po tom, čo sa k nej pripojíme, na čo nám slúži funkcia `MySQL_Connect()`. Táto funkcia má tri parametre, pričom všetky sú nepovinné. Prvým parametrom je hostname servera, na ktorom beží databáza. Druhý parameter je používateľské meno a tretí heslo na pripojenie k databáze. V prípade, že nepoužijeme parametre, sú implicitnými hodnotami pre hostname „localhost“, pre používateľské meno meno používateľa vlastniaceho proces a prázdne heslo. Funkcia vracia tzv. „link identifier“, čo je vlastne identifikačné číslo otvoreného pripojenia k databáze. V prípade, že používate naraz iba jedno pripojenie, netreba túto hodnotu uchovávať v nejakej premennej. Uloženie hodnoty identifikátora pripojenia využijete iba vtedy, keď sa pripájate k viacerým databázovým serverom naraz, teda otvárate viac pripojení. Keď potom používate ostatné funkcie MySQL, každá má nepovinný parameter predstavujúci identifikátor pripojenia. V prípade, že máte otvorených viac pripojení, treba špecifikovať, ktorému pripojeniu, resp. ktorej databáze je funkcia určená. Ak skript volá viackrát funkciu `MySQL_Connect()` s rovnakými parametrami počas jedného procesu, neotvára sa nové pripojenie, ale funkcia vráti identifikátor už existujúceho pripojenia. Použitie funkcie `MySQL_Connect()`:

```
$link1=MySQL_Connect(„localhost“,“root“,“heslo“);
```

Ďalšou funkciu, ktorou sa môžete pripojiť k databáze, je `MySQL_pConnect()`. V parametroch je zhodná s funkciou `MySQL_Connect()`, líši sa však návratovou hodnotou, ktorou je perzistentný identifikátor pripojenia. Funkcia najprv zistí, či takýto identifikátor náhodou neexistuje. Ak áno, vráti jeho hodnotu. V prípade, že nenájde nijaký perzistentný identifikátor, vytvorí ho a vráti ako svoju návratovú hodnotu. Takýto link je charakteristický tým, že nie je zatvorený po skončení exekúcie skriptu (linky otvorené funckiou `MySQL_Connect()` sa zatvárajú po skončení behu skriptu, ak neboli zatvorené skôr).

Poslednou funkciou z kategórie základných, o ktorej si povieme, je `MySQL_Close()`. Táto funkcia slúži na zatvorenie pripojenia otvoreného funkciou `MySQL_Connect()`, ale nezatvára pripojenia otvorené funkciou `MySQL_pConnect()`. Funkcia má jeden nepovinný parameter, ktorým je identifikátor pripojenia. V prípade, že ho použijete, je zatvorené práve dané pripojenie. Ak nepoužijete tento nepovinný parameter, je zatvorené pripojenie, ktoré bolo otvorené ako posledné. Z toho vyplýva, že používanie tohto nepovinného parametra je zbytočné v prípade, že používate naraz iba jedno pripojenie. Ukážka práce s identifikátorom pripojenia pri pripájaní sa a odpájaní z databázy:

```
$link1=MySQL_Connect(„localhost“,“root“,“heslo“);
// ......
// práca s databázou
// ......
MySQL_Close($link1);
```

Funkcie pre operácie s databázami

Teraz si povieme niečo o funkciách, ktoré slúžia na manipuláciu s databázami. Prvou z nich bude funkcia `MySQL_Create_DB()`. Tá slúži na vytvorenie databázy s menom, ktoré jej zadáte ako parameter. Môžete definovať aj druhý parameter, ktorým je identifikátor pripojenia, o ktorom sme už uviedli, že je zbytočné ho používať, ak máme otvorené iba jedno pripojenie. Ak teda chceme vytvoriť databázu s názvom „test“, urobíme to takto:

```
MySQL_Create_DB(„test“);
```

Na zmazanie databázy slúži funkcia `MySQL_Drop_DB()`. Povinným parametrom je opäť meno databázy, tentoraz tej, ktorú chceme zmazať, a nepovinným identifikátor pripojenia. Funkcia vracia hodnotu true, ak sa databázu podarilo zmazať, v opačnom prípade false. Databázu vytvorenú v predchádzajúcom príklade by sme teda zmazali takto:

```
MySQL_Drop_DB(„test“);
```

Pozor, po použití tejto funkcie zmažete celú databázu aj s jej štruktúrou a dátami, pričom sa ani štruktúra, ani dáta už nedajú obnoviť.

Ak chceme získať zoznam všetkých databáz, ktoré sú k dispozícii z momentálneho MySQL daemona, použijeme funkciu `MySQL_List_DBs()`. Jediným a nepovinným parametrom je identifikátor pripojenia. Funkcia vracia hodnotu, ktorá je ukazovateľom na databázy na serveri. Na získanie názvov týchto databáz musíme danú návratovú hodnotu použiť ako parameter funkcie `MySQL_Tablename()`. Prvým parametrom funkcie je návratová hodnota predchádzajúcej funkcie a druhým je číslo riadka (návratová hodnota obsahuje riadky, z ktorých každý obsahuje meno databázy). Preto je potrebné na výpis všetkých databáz použiť nejaký cyklus. Ukážka jednej z možností použitia:

```
$result = MySQL_List_DBs();
$i = 0;
```

```
while ($i < MySQL_Num_Rows ($result)) {
    echo MySQL_Tablename($result, $i).“<BR>“;
    $i++;
}
```

Týmto kódom zabezpečíme výpis všetkých databáz na serveri (na ktorý sme sa predtým pripojili funkciou MySQL_Connect() alebo MySQL_pConnect()). V kóde sa vyskytuje aj funkcia MySQL_Num_Rows(), o ktorej si povieme až neskôr.

Podobne môžeme postupovať, ak chceme získať výpis všetkých tabuliek v databáze. V tomto prípade však použijeme funkciu `MySQL_List_Tables()`. Povinným parametrom funkcie je meno databázy, z ktorej chceme získať mená tabuliek. Nepovinným parametrom je opäť identifikátor pripojenia. Funkcia opäť vracia ukazovateľ, ktorého obsah získame obdobným použitím funkcie `MySQL_Tablename()`, ako to bolo v predchádzajúcom prípade. Ako príklad uvádzam kód na výpis všetkých tabuliek v databáze test:

```
$result = MySQL_List_Tables(„test“);
$i = 0;
while ($i < MySQL_Num_Rows ($result)) {
    echo MySQL_Tablename($result, $i).“<BR>“;
    $i++;
}
```

Poslednou zo skupiny funkcií pre operáciu s databázami, o ktorej si povieme, je `MySQL_Select_DB()`. Táto funkcia slúži na nastavenie databázy, na ktorú sa budú vzťahovať požiadavky, ktoré nie sú adresované priamo nijakej databáze. Na vysvetlenie: existujú funkcie, ktoré posielajú požiadavku, resp. príkaz na nejakú databázu, ktorej meno špecifikujú ako jeden z parametrov. Existujú však aj také funkcie, ktoré meno databázy nešpecifikujú, teda posielajú iba požiadavku. Preto, aby bolo jasné, kam budú smerovať požiadavky takýchto funckií, treba použiť funkciu `MySQL_Select_DB()`. Prvým a povinným parametrom je meno databázy, druhým a nepovinným je identifikátor pripojenia. Funkcia nastaví aktívnu databázu pre daný identifikátor, pričom v prípade, že nie je špecifikovaný, použije sa identifikátor posledne otvoreného pripojenia. V prípade, že nie je otvorené nijaké pripojenie, skúsi ho otvoriť ako funkcia `MySQL_Connect()` a následne použiť ako svoj druhý parameter. Na nastavenie aktívnej databázy „test“ píšeme:

```
MySQL_Select_DB(„test“);
```

Posielanie požiadaviek na databázu

Základom práce s databázou sú operácie umožňujúce vyberať, vkladať, mazať či modifikovať riadky tabuliek, prípadne samotné operácie s tabuľkami, ako vytváranie, mazanie alebo modifikovanie. Aby ste mohli všetky tieto operácie vykonávať, odporúčam vám preštudovať si seriál o MySQL. Tu sa budeme venovať tomu, ako jednotlivé príkazy odoslať databáze z PHP skriptu.

Prvou možnosťou je použiť funkciu `MySQL_DB_Query()`. Táto funkcia má dva povinné parametre. Prvým je meno databázy, na ktorú sa bude požiadavka vzťahovať. Druhým parametrom je samotná požiadavka, teda reťazec obsahujúci príkaz pre databázu MySQL, ktorý jej chcete odoslať. Tretím a nepovinným parametrom je opäť identifikátor pripojenia. V prípade, že ho nezadáte, použije sa posledne otvorené pripojenie; ak nijaké neexistuje, pokúsi sa funkcia vytvoriť ho spôsobom ako `MySQL_Connect()` volaná bez parametrov. V prípade, že nastala chyba, vráti funkcia hodnotu false, v opačnom prípade vracia identifikátor výsledku, pre ktorého spracovanie treba túto návratovú hodnotu uložiť do premennej. Ukladať hodnotu nie je potrebné vtedy, ak nás výsledok operácie nezaujíma, napr. pri použití príkazov DELETE, INSERT, UPDATE, atď. Pri použití príkazu SELECT je nevyhnutné ukladať návratovú hodnotu vždy, inak by nebolo možné spracovať výsledky z databázy. Takže na vybranie celého obsahu tabuľky info z databázy test bude kód nasledujúci:

```
$result = MySQL_DB_Query(„test“,“select * from info“);
```

Druhou možnosťou, ako odoslať požiadavku na databázu, je použiť funkciu `MySQL_Query()`. Táto funkcia sa líši od predchádzajúcej tým, že má iba jeden povinný parameter, a to požiadavku na databázu. Nemá teda prvý parameter požadujúci meno databázy, na ktorú sa požiadavka vzťahuje. Požiadavka je automaticky smerovaná na aktívnu databázu daného pripojenia, ktorého identifikátor môžeme špecifikovať ako druhý a nepovinný parameter. V prípade, že tak neurobíme, funkcia sa správa podobne ako `MySQL_DB_Query()`. Výsledok funkcie je rovnaký, ako to bolo pri `MySQL_DB_Query()`, teda identifikátor výsledku operácie, pričom ho môžeme priradiť premennej, čo má však význam iba pri použití príkazu SELECT. Na nastavenie aktívnej databázy je potrebné použiť funkciu MySQL_Select_DB(). Ukážka predchádzajúceho príkladu, ale s použitím `MySQL_Query()`:

```
MySQL_Select_DB(„test“);$result = MySQL_Query(„select *
from info“);
```

Spracovanie dát z databázy

Preto, aby malo využitie databáz nejaký význam, potrebujeme hodnoty vrátené funkciami posielajúcimi požiadavky na databázu nejakým spôsobom spracovať. Na to nám slúži niekoľko funkcií implementovaných v MySQL. Najjednoduchším spôsobom, ako výsledok funkcie `MySQL_Query()`, príp. `MySQL_DB_Query()` spracovať, je použiť funkciu `MySQL_Result()`. V drvivej väčšine prípadov potrebujeme spracúvať výsledky po použití príkazu SELECT, ktorý môže vrátiť rozličný počet riadkov. Preto je rozumné v požiadavkách na databázu používať klauzulu ORDER BY, ktorou získame riadky už usporiadané. Využijeme to vtedy, keď chceme získať nejaký konkrétny riadok z výsledku. Riadky sú číslované od nuly. Teraz k spomínanej funkcii `MySQL_Result()`. Má dva povinné parametre a jeden nepovinný parameter. Prvým parametrom je premenná, ktorej obsah chceme spracovať z požiadavky na databázu. Druhým parametrom je index riadka, ktorý chceme spracovať. Ako sme už uviedli, nezabudnite, že sú indexované od nuly. Posledným parametrom je offset stĺpca, ktorého hodnotu chcete získať. Takisto je možné použiť meno stĺpca, namiesto jeho číselného offsetu. V prípade, že tento parameter nezadáte, je implicitne nastavená na prvý stĺpec. Možno to teoreticky vyzeralo trochu zložito, ale v skutočnosti ide o veľmi jednoduchú záležitosť. Azda to osvetlí nasledujúci príklad:

```
MySQL_Select_DB(„test“);
$result = MySQL_Query(„select * from info“);
echo MySQL_Result($result,0,’id’);
```

Tento kód vypíše na obrazovku hodnotu stĺpca `id`, z prvého riadka z tých, ktoré boli špecifikované ako návratová hodnota premennej $result, teda výber (netriedený) z tabuľky info z databázy test. Druhou možnosťou, ako spracovať dáta z návratovej hodnoty, je použiť funkciu `MySQL_Fetch_Row()`. Táto funkcia slúži na uloženie jedného riadka z návratovej hodnoty funkcie `MySQL_Query()` alebo `MySQL_DB_Query()` do poľa so zodpovedajúcimi offsetovými indexmi, pričom sa indexuje od nuly. Hodnoty daného riadka tak máte k dispozícii v poli obsiahnutom v premennej, ktorej bol výsledok funkcie `MySQL_Fetch_Row()` priradený. Preto jediným a povinným parametrom tejto funkcie je identifikátor výsledku. Použitím tejto funkcie bude predchádzajúci príklad vyzerať takto:

```
MySQL_Select_DB(„test“);
$result = MySQL_Query(„select * from info“);
$pole=MySQL_Fetch_Row($result);
echo $pole[0];
```

Opakované volanie funkcie `MySQL_Fetch_Row()` má za následok to, že funkcia vracia vždy ďalší riadok v poradí, pričom v prípade, že už nijaký riadok nenasleduje, vracia funkcia false.

Rozšíreným variantom funkcie `MySQL_Fetch_Row()` je `MySQL_Fetch_Array()`. Táto funkcia má obdobné vlastnosti ako predchádzajúca s tým rozdielom, že nevracia skalárne, ale asociatívne pole. Kľúčmi tohto poľa sú názvy jednotlivých stĺpcov, hodnotami sú príslušné hodnoty stĺpcov daného riadka. Funkcia však uchováva aj skalárne indexy, takže k jednotlivým prvkom poľa môžete pristupovať aj spôsobom, ako to bolo v prípade `MySQL_Fetch_Rows()`. Ukážka predchádzajúceho príkladu použitím MySQL_Fetch_Array():

```
$pole=MySQL_Fetch_Array($result);
echo $pole[„id“];
```

Pri opakovanom volaní tejto funkcie sa správa rovnako ako predchádzajúca, teda vždy vráti nasledujúci riadok v poradí. V prípade, že nijaký nenasleduje, vracia false.

Veľmi dôležitou a často využívanou funkciou je `MySQL_Num_Rows()`. Tá slúži na zistenie počtu riadkov, ktoré boli vybrané predchádzajúcim použitím niektorej funkcie posielajúcej požiadavku na databázu a ktoré sú identifikované identifikátorom výsledku. Tento identifikátor je povinným a jediným parametrom funkcie `MySQL_Num_Rows()`. Funkcia teda vracia počet riadkov vo výsledku špecifikovanom identifikátorom výsledku a túto hodnotu môžete využiť na kontrolu riadiacej premennej v cykle. V záujme rýchlosti je, samozrejme, rozumnejšie uložiť si hodnotu do nejakej premennej než volať túto funkciu pri každej iterácii.

Dodatkové funkcie

Stane sa (azda nie pričasto), že kód odoslaný na databázu je nekorektný, a preto funkcia `MySQL_Query()`, resp. `MySQL_DB_Query()` nevracia nijakú hodnotu a pri použití takéhoto výsledku v ďalších funkciách nastane chyba. Keďže chyba v kóde pre MySQL nie je chybou PHP, nedá sa hneď odhadnúť, kde táto chyba v kóde nastala. Preto je vhodné použiť funkciu `MySQL_Error()`, ktorá vracia chybovú správu (ak nejaká je) poslednej operácie na databáze. Preto je možným spôsobom, ako túto chybu odhaliť, vypísať si návratovú hodnotu tejto funkcie na obrazovku.

Poslednou funkciou, ktorú spomeniem, je `MySQL_Free_Result()`. Jej parametrom je identifikátor výsledku, a ako názov napovedá, funkcia uvoľní všetku pamäť asociovanú s daným výsledkom (čím u,ž samozrejme, daný výsledok nebude k dispozícii). V prípade, že ju nepoužijete, sú všetky výsledky uvoľnené z pamäte po skončení behu skriptu.

Záver

Čo dodať na záver? Ak ste zvládli všetky časti PHP, neznamená to ešte, že ste zvládli aj samotné PHP. Dôležité sú aj skúsenosti, ktoré nadobudnete, a schopnosť naučiť sa niečo nové aj bez cudzej pomoci. V tých šiestich častiach iste nie je zďaleka prebrané všetko, a ako sa hovorí, všetko nevie nikto. Takže veľa šťastia pri kódovaní.

Andrej Chu