

PHP - 1. díl - úvod

Tímto dílem zahajujeme seriál zaměřený na PHP. Seriál si klade za cíl naučit pozorného čtenáře používat PHP.

Pro mnoho lidí je PHP tajemná zkratka. Pokud se pohybujete častěji na internetu, určitě na ní poměrně často narazíte. Ale pro mnoho lidí zůstává její skutečný význam trochu zahalen v nádechu tajemna. Občas se najde i nějaká ta úsměvná historka, za co všechno lidé PHP považují. Jednu takovou najdete třeba [zde](#). Pokud sami netušíte, co to PHP je, nezoufejte, tento seriál vám to prozradí.

Protože PHP úzce souvisí s internetem a webovými stránkami, zaměříme pozornost právě tam. Pokud si budete chtít vytvořit vlastní webové stránky, máte celou řadu možností. Asi prvním krokem takového běžného laického zájemce bývá vytvořit webové stránky jako statické textové stránky. To znamená, že každou stránku si napíšete podobně jako textový dokument. Je celkem jedno, zda použijete nějaký vizuální editor typu Front Page, a nebo se speciálně naučíte jazyk HTML k tomu určený a budete psát stránky přímo v HTML. Podstatné je, že vaše stránky budou statické.

Statické stránky vám zobrazí jen to, co jste napsali a víc ani čárku. Pokud chcete třeba na každou stránku dopsat na konec Vaše jméno a adresu, nezbyde vám nic jiného, než ručně nakopírovat tyto údaje na každou stránku. Pokud těch stránek máte hodně, jedná se o velké množství ruční práce. Další váš požadavek třeba může být zobrazit ve stránce dnešní datum. To už je pomocí statických stránek prakticky nemožné, nechcete-li každý den přepisovat datum ručně. A právě proto chytré hlavy vymyslely dynamické stránky.

Dynamické stránky jsou stránky, které přidávají do stránek něco navíc. Mohou vám třeba připsat automaticky adresu na konec stránky, nebo třeba přidat ten dnešní datum. Možnosti dynamických stránek jsou obrovské a jejich hranice jsou v zásadě dány jen fantazií autorů. Často se dynamické stránky propojují zejména s databázemi.

Asi již tušíte, kam vlastně mířím. PHP je jedním z prostředků, které vám umožňují vytvořit dynamické webové stránky. PHP je v podstatě jednoduchý programovací jazyk. Použitím PHP můžete s webovými stránkami dělat hotové divy. Je toho opravdu hodně, co vám PHP umožňuje, tak jen pro příklad: práce s databázemi, odesílání mailů ze stránek, vytváření a editaci obrázků, automatizovanou práci s textem, zpracování údajů od uživatele, a mnohem mnohem více. Pro vstup do světa PHP by měl sloužit právě tento seriál.

Pokud chcete tvořit dynamické stránky, máte v zásadě dvě možnosti. Můžete dynamiku do webových stránek přidat buď na serveru (pak se používají tzv. serverové skriptovací jazyky). A nebo můžete přidat dynamiku do stránek u uživatele (pak se používají tzv. klientské skriptovací jazyky). Každá z těchto možností má své výhody a nevýhody. Serverové skripty máte více pod kontrolou a jejich možnosti jsou obecně větší. Klientské skripty zase postrádají mnoho z možností serverových skriptů, ale na druhé straně umožňují vytvořit interaktivní prostředí ve stylu změna obrázku při přejetí myši, apod.

Samotné PHP patří do skupiny serverových skriptovacích jazyků. Umožňuje tedy tvořit dynamické webové stránky na serveru. Jako ve všem na světě má PHP celkem zdatnou konkurenci. Existují tedy i další prostředky, jak dosáhnout toho samého výsledku, jako s PHP. Obrovskou výhodou PHP jsou jeho tři základní vlastnosti. První je fakt, že PHP je naprosto

zdarma. Za používání PHP neplatíte autorům žádné poplatky, můžete si PHP libovolně stahovat, kopírovat, množit, prodávat, zkrátka, jak je libo. Druhou výhodou je, že PHP funguje všude. PHP může pracovat stejně dobře pod Windows, jako pod Linuxem i na dalších systémech. A poslední třetí vlastností je, že PHP je jeho kvalita. Toto všechno pomohlo PHP dostat se na výsluní.

Abych alespoň pro úplnost zmínil nějaké konkurenty PHP, tak uvedu, že nejvýznamnějším protihráčem je technologie ASP firmy Microsoft. Technologii ASP používá pro dynamické stránky i server Živě.cz. Nevýhodou ASP oproti PHP je fakt, že není k dispozici zdarma a slušné využití technologie ASP je vázáno pouze na Windows. Dalšími konkurenty PHP jsou také třeba JSP, případně další technologie. Detaily se již vymykají tématu tohoto seriálu.

Tento úvodní díl je psán spíše teoreticky a je určen pro hrubou představu, k čemu PHP vlastně slouží. Další díly se naproti tomu budou již věnovat praktickému používání PHP a praktickým příkladům. Budu se snažit, aby každý další díl byl proložený praktickým příkladem.

PHP - 2. díl – na čem zkoušet

Další díl seriálu o PHP vám umožní získat prostředí pro zkoušení a vývoj v tomto jazyce.

Protože tento seriál chci založit spíše praktickým způsobem, nehodlám vysvětlovat pouze suchou teorii, kterou si nemůžete okamžitě vyzkoušet. S nadsázkou se dá říci, že seriál se budu snažit psát "materialisticky", tedy pouze na hmatatelných věcech. Je tedy logické, že nejdříve musím čtenářům Živě dát k dispozici nástroj, na kterém by si mohli PHP skripty vyzkoušet. Budou tak moci okamžitě otestovat a zkoumat příklady z dalších dílů v chodu.

Nejjednodušší, i když nepříliš pohodlnou možností, jak vyzkoušet jakýkoli PHP skript, je nahrát tento skript na webový server s podporou PHP někde na internetu. Výsledek PHP skriptu se pak prohlédne jednoduše v browseru, například v Internet Exploreru. Dnes podporuje placenou, nebo neplacenou formu PHP skriptování naprostá většina webových serverů. Získat webhosting s PHP skriptováním zdarma je možné například na www.webzdarma.cz, nebo na www.miesto.sk. Znovu ale opakuji, že pro testování skriptů jde o velmi nepohodlný způsob, kdy musíte neustále žhavit FTP přenos, a kdy musíte být připojeni k internetu.

Věřím, že spíše dáte přednost tomu, abyste mohli vaše PHP skripty zkoušet přímo na vašem počítači. Proto další kroky tohoto článku budou směřovat tímto směrem.

Odborníci, kutilové a další lidé mohou rozchodit PHP skriptování na svém počítači s pomocí informací, které naleznou na mateřském webu PHP, na www.php.net. Zde je k dispozici možnost stažení potřebných souborů, včetně další dokumentace, jak rozjet PHP skriptování pod vším možným. Nutno ovšem říci, že je to cesta vedoucí čtením anglického textu a nastavováním konfiguračních souborů. Ovšem určitému typu lidí spíše kutilského typu bude tato cesta vyhovovat.

Většine čtenářůale spíše bude vyhovovat nějaké hotové řešení. **A právě pro ně jsem připravil program „Intranetový server“, který nainstaluje na počítač webový server s podporou**

PHP a podporou databáze MySQL (databázi využijeme v pozdějších dílech seriálu). Instalační program provede sám všechna nutná nastavení pro bezproblémový chod. Předem uvádím, že vlastní instalační program je určen pouze pro Windows. U uživatelů Linuxu předpokládám, že hotové řešení bude součástí běžně používaných distribucí.

Instalační program si můžete stáhnout [na tomto místě](#). Jedná se cca o 9 MB veliký program, který bude pracovat na všech typech Windows, tj. na Windows 95, 98, NT, ME, 2000, XP, 2003. Webový server je extrémně nenáročný a bude pracovat i na těch nejpomalejších počítačích. Zbytek článku budu věnovat popisu instalace a rozchození tohoto programu.

Po stáhnutí „Intranetového serveru“ instalační program spustíte a odklepejte všechny otázky. Tím vykonáte instalaci. Pokud jste program nainstalovali na Windows 95/98, a nebo na počítač, kde nejsou k dispozici Microsoft Office, je vhodné si přecíst soubor C:\inet_srv\reseni_problemu.txt.

Po instalaci lze webový server ovládat přes startovací menu ve složce Start menu -> Programy -> Intranetový server. Vyzkoušíme proto ještě jednoduchý test, zda intranetový server pracuje správně, a zda je připraven pro testy PHP skriptů z dalších dílů tohoto seriálu.

Nejdříve spustíte webový server. To se provede pomocí startovacího menu přes Start menu -> Programy -> Intranetový server -> Apache WWW server. Po spuštění by vám mělo zůstat dole na liště okno s titulkem „Apache WWW server“. To znamená, že webový server běží. Nechte tedy okno tam kde je, spustíte Internet Explorer a do řádky s adresou napište **http://localhost/** a potvrďte klávesou Enter. Webový server by vám měl odpovědět a zobrazit prázdný seznam souborů na modrém pozadí s fialově podkreslenými dvěma řádky. Pokud se tak stalo, vše je v pořádku, i samotný seznam je totiž vykreslen pomocí PHP skriptů. Pak můžete v klidu webový server zase ukončit přes startovací menu. Stačí spustit Start menu -> Programy -> Intranetový server -> Ukončit Apache WWW server. Vše je tedy připraveno pro pokusy s PHP skripty v dalších dílech.

PHP - 3. díl – tvoříme první skripty

V tomto díle seriálu o PHP vytvoříme první sice jednoduché, ale skutečné, skripty.

V nadpisu článku slibuji první PHP skripty. Prosím, čtěte povídání kolem příkladů velmi pozorně, protože jsem se do povídání snažil zahrnout i mnoho důležitých informací, které budete potřebovat při vytváření každého PHP skriptu.

Půjdu přímo k věci a uvedu hned příklad velmi jednoduchého skriptu. Představme si, že třeba vytvoříme soubor s názvem první.php, který bude obsahovat následující:

```
<html>
<head>
<title>První PHP skript</title>
</head>
<body>
<h1><?php echo date("d.m.Y H:i:s"); ?></h1>
```

```
</body>
</html>
```

Pokud tento PHP skriptu spustíte, zobrazí se Vám v prohlížeči aktuální datum a čas.

Jak spustit tento PHP skript?

Pokud máte nainstalovaný intranetový server z minulého dílu, tak spustíte PHP skript takto:

- Nejdříve nainstalujte intranetový server pomocí Start menu -> Programy -> Intranetový server -> Apache WWW server.
- Vytvořte soubor první.php a nakopírujte ho do složky C:\inet_srv\http\doc_root\.
- Všechny další pokusné skripty se budou kopírovat do stejné adresáře.
- Spustíte Internet Explorer (a nebo jiný browser, který používáte). Do řádku s adresou napište **http://localhost/** a odklepněte klávesou Enter.
- V seznamu souborů se Vám objeví nakopírovaný skript první.php. Kliknutím na nápis první.php skript pustíte a uvidíte výsledek přímo v Internet Exploreru.
- Pokud budete chtít spustit další skripty, tak stačí pouze nakopírovat příslušné soubory do složky C:\inet_srv\http\doc_root\ a poslat Internet Explorer na adresu http://localhost/, kde můžete klepnutím spustit libovolný skript.

Pokud nemáte nainstalovaný intranetový server a chcete zkoušet PHP skripty přímo na webu, pak prostě soubor první.php nahrajte pomocí FTP na web jako běžnou HTML stránku. Pak Internet Explorer nasměrujte na adresu vašeho webu a na konec přidejte /první.php. Stejně tak to udělejte i s dalšími pokusnými skripty v tomto i dalším díle.

Tím jsem popsal, jak skript vyzkoušet, a vrátím se k vysvětlení, jak vlastně náš první PHP skript pracuje.

Jak pracuje tento PHP skript?

Když dojde ke spuštění skriptu, PHP si všímá jen toho místa v našem souboru, které začíná **<?php** a končí **?>**. Jen tento malý kousek je totiž skutečné PHP. Všechno ostatní je jen čistý HTML jazyk, který je při spuštění skriptu pouze beze změny poslán browseru.

Pokud to trochu zjednoduším, každý PHP soubor je tedy střídavě z kousků HTML a z kousků pravých PHP skriptů. Tedy určité kousky PHP souboru jsou vykonávány v HTML módu, tedy naprosto doslova a beze změny jsou přímo posílány browseru. A jiné kousky jsou vykonávány v PHP módu a jsou prováděny jako příkazy programovací jazyka. Takové kousky, které chceme vykonávat v PHP módu musíme speciálně označit. V našem příkladě je označený kousek v PHP módu tak, že začíná značkou **<?php** a končí značkou **?>**. Možností, jak označit kousek v PHP módu je ovšem více:

- První možností je označit začátek značkou **<? a konec značkou ?>**. Tento způsob je často používaný, ale pokud můžete, nepoužívejte jej, a rozhodně dejte přednost následující druhé možnosti. Tento první způsob je navíc možné v konfiguraci PHP zakázat, takže nemáte ani jistotu, že bude fungovat vždy a všude.
- Druhou možností je označit začátek značkou **<?php** a konec značkou **?>**, tedy stejně, jako v našem příkladu. Pokud můžete, používejte tento způsob. Jednak pracuje

zaručeně naprosto vždy a všude a navíc umožňuje nejlepší spolupráci s XML a XHTML technologiemi.

- Další možností je označit začátek `<script language="php">` a konec značkou `</script>`. Tato možnost se skoro nepoužívá, a je k dispozici hlavně pro spolupráci se staršími verzemi Front Page. Rovněž tato možnost funguje vždy, ale pokud můžete používejte druhou možnost.
- Poslední možností je používání tzv. ASP stylu, tedy začátek je označen jako `<%` a konec jako `%>`. Pokud můžete, tak tuto čtvrtou možnost nepoužívejte vůbec, mimo jiné i proto, že funguje jenom někde.

Pokud to tedy shrnu, jistotu, že to bude fungovat máte jenom u 2. a 3. možnosti. Je všeobecně doporučováno používat striktně 2. možnost, kterou jsem použil i já, a i do budoucna nebudu používat nic jiného.

Dostal jsem se postupně tedy k tomu, že skutečný kousek PHP v našem příkladu je pouze

```
echo date("d.m.Y H:i:s");
```

Co tedy tento kousek provádí? Používám v něm příkaz `echo`, který slouží jako univerzální příkaz k výpisům. Za slovem `echo` je uvedeno, co se požaduje vypsát. V našem příkladě je to tedy funkce s názvem `date`, která se používá pro zobrazení data, času, nebo obojího dohromady. Funkce `date` je velmi univerzální a dokáže vypsát datum a čas prakticky v jakémkoli formátu, na který si jenom vzpomenete. V řetězci uvnitř závorek je pak uvedeno, jaký formát data a času si vlastně přejeme. Každé písmeno značí určitý druh údaje a jeho způsob výpisu. Co které písmeno znamená se můžete podívat do tabulky, která je uvedena níže.

Pokud se tedy podívám do našeho příkladu, je zde uvedeno, že si přejeme formát datumu a času zapsaný takto:

```
d.m.Y H:i:s
```

Rozkódovat tento formát můžeme podle níže uvedených tabulek. Písmeno `d` znamená, že chci vypsát den v měsíci. Protože pokračuje tečka, za dnem v měsíci bude tedy tečka. Dále je uvedeno písmeno `m`, což znamená vypiš číslo měsíce, pak za něj tečku. Pokračuje se písmenem `Y`, což značí vypiš rok. Tím je vypsáno celé datum. Za datum vypiše mezeru a za ní bude uveden čas ve formátu hodiny:minuty:sekundy, jednotlivé části času budou odděleny dvojtečkami.

Písmeno	Význam písmena uvnitř funkce date
a	Vypíše am pro dopoledne, nebo pm pro odpoledne.
A	Vypíše AM pro dopoledne, nebo PM pro odpoledne.
B	Vypíše tzv. internetový čas Swatch (je v rozsahu 000 až 999)
d	Den měsíce, dvě číslice s úvodními nulami (01 až 31)
D	Anglická třípísmenná zkratka dnes v týdnu (Mon až Sun)
F	Anglický název měsíce (January až December)
g	12-hodinový formát hodiny bez úvodních nul (1 až 12)

G	24-hodinový formát hodiny bez úvodních nul (0 až 23)
h	12-hodinový formát hodiny s úvodními nulami (01 až 12)
H	24-hodinový formát hodiny s úvodními nulami (00 až 23)
i	Minuty s úvodními nulami (00 až 59)
I	Vypíše 1, pokud je letní čas 1, nebo 0, pokud není.
j	Den v měsíci bez úvodních nul (1 až 31)
l	Anglický název dne v týdnu (Monday až Sunday)
L	Vypíše 1, pokud je přestupný rok, nebo 0, pokud není
m	Číslo měsíce s úvodními nulami (01 až 12)
M	Anglická třípísmenná zkratka měsíce (Jan až Dec)
n	Číslo měsíce bez úvodních nul (1 až 12)
O	Odchylka od Greenwichského času (GMT) v hodinách. Příklad: +0200
r	Datum formátované podle normy RFC 822 Příklad: Thu, 21 Dec 2000 16:01:07 +0200
s	Sekundy s úvodními nulami (00 až 59)
S	Anglická pořadová přípona dne v měsíci, 2 znaky st, nd, rd nebo th.
t	Počet dní v daném měsíci (28 až 31)
T	Nastavení časového pásma na tomto počítači Příklad: EST, MDT ...
U	Počet sekund od 1. ledna 1970
w	Číselná reprezentace dne v týdnu 0 (pro neděli) až 6 (pro sobotu)
W	Číslo týdne (0 až 51)
Y	Rok jako čtyřmístné číslo. Příklad: 1999 nebo 2003
y	Poslední dvě číslice roku. Příklad: 99 nebo 03
z	Pořadové číslo dne v roce (0 až 366)
Z	Posun časového pásma v sekundách. (-43200 až 43200)

Druhý příklad: výpis čistého textu

Zkusme teď druhý příklad. Ten ukáže, jak pomocí příkazu `echo` vypsát čistý text. Řekněme, že pomocí PHP budeme chtít vypsát třeba větu „Buď vítán!“. Pomocí příkazu `echo` to uděláme velice snadno:

```
echo "Buď vítán!";
```

Já myslím, že toto je jasné, ale příkaz `echo` umožňuje vypsát najednou i více řetězců. Jenom je potřeba je oddělit čárkami:

```
echo "Buď", " ", "vítán!";
```

Jak vidíte, právě jsem stejnou větu vypsál pomocí tří řetězců. Jednotlivé řetězce je potřeba oddělit čárkami. Vše je potřebné ukončit středníkem, tak jako v předcházejících případech.

Středníkem se ukončuje každý příkaz v PHP a podle středníku pozná PHP při spuštění, kde končí jeden příkaz a začíná další.

Pokud byste si chtěli i tyto příklady s příkladem echo vyzkoušet, jednoduše vytvořte soubor s názvem třeba druhý.php, do kterého uložte třeba:

```
<html>
<head>
<title>První PHP skript</title>
</head>
<body>
<h1><?php echo "Bud' vítán!"; ?></h1>
</body>
</html>
```

PHP - 4. díl – proměnné

V dnešním dílu seriálu budete provedeni proměnnými a při tom si vyzkoušíte další užitečné skripty.

Tento díl se bude věnovat proměnným. Při povídání o proměnných v PHP budou uvedeny i různé skripty na vyzkoušení, které vás dovedou zase o kousek dále. Prosím přečtěte si tyto kapitoly pozorně, i když místy jsou trochu teoretické, ale jsou to nejzákladnější stavební kameny PHP.

Skoro všechny údaje, se kterými se pracuje uvnitř PHP jsou uloženy do tzv. proměnných. Proměnné jsou celkem klíčovou součástí každého programovacího jazyka, PHP nevyjímaje.

Názvy proměnných

Každý, kdo programoval, nebo se alespoň setkal s nějakým programovacím jazykem ví, co jsou to proměnné. Je to pojmenované místo, do kterého můžete uložit hodnotu, ať už číslo, řetězec, objekt či něco jiného. Každá proměnná je v PHP pojmenována a při použití se dává v PHP před název znak dolaru \$. Viz náš první ukázkový skript, který ukazuje velmi jednoduché použití proměnné:

```
<html>
<head>
<title>Příklad 1. ze 4. dílu</title>
</head>
<body>
<h1><?php
    $počet = 3;
    echo "Proměnná s názvem počet má hodnotu ", $počet, "<br>";
    $počet = "tři";
    echo "Proměnná s názvem počet má teď hodnotu ", $počet, "<br>";
?></h1>
</body>
</html>
```

Uložte si výše uvedený skript do nějakého souboru, a můžete jej vyzkoušet. Pokud nevíte jak, stáhněte si [Intranetový server](#) a přečtěte se ve 3. díle tohoto seriálu, jak zkoušet PHP skripty na „Intranetovém serveru“.

Jak pracuje výše uvedený skript? Je v něm použita jediná proměnná s názvem počet. Nejdříve příkazem \$počet = 3 vložíme do proměnné s názvem počet číslo 3. Od té chvíle všude, kde napíšeme \$počet bude se v PHP toto místo nahrazovat číslem 3, a to až do příštího vložení jiné hodnoty do proměnné počet. Takže následující řádek vypíše tento text: „Proměnná počet má hodnotu 3
“. Na 3. řádku vložíme do PHP jinou hodnotu, a to řetězec „tři“. Od tohoto vložení už bude v PHP nahrazován výraz \$počet tímto řetězcem „tři“. Proto poslední 4. řádek vypíše „Proměnná má teď hodnotu tři
“.

Stálo by za to zmínit se, jaké názvy proměnných jsou dovolené a jaké ne. Jednoduše by se dalo napsat, že název proměnné smí obsahovat písmena a číslice, ale nesmí začínat číslicí. Takže proměnná s názvem tygr23 je správná, ale název 1bota je nesprávný. V mnoha programovacích jazycích je možné používat jen písmena anglické abecedy, ale PHP je jiný. V PHP klidně můžete používat háčky, čárky, přehlásky, atd., jak je Vám libo. Takže název Délka, nebo název Množství jsou platné názvy proměnných v PHP skriptech. Čeština není žádným problémem pro PHP. Protože v názvu proměnných není možné používat mezeru, je dovoleno namísto toho používat znak podtržítka . Takže celkem běžné jsou názvy jako Počet_Čtenářů, nebo Uživatelské_Heslo.

Co je to pole proměnných?

V PHP se hodně často používají tzv. pole proměnných. O co jde? Pole je vlastně zkratkou za mnoho proměnných. Zkusme třeba takový příklad, potřebujete si zapamatovat, kolik má který měsíc dnů (řekněme, že únor nebude v přestupném roce). Pak to můžeme uložit do proměnných třeba takto:

```
$dni_leden = 31;
$dni_unor = 28;
$dni_březen = 31;
$dni_duben = 30;
$dni_květen = 31;
$dni_červen = 30;
$dni_červenec = 31;
$dni_srpen = 31;
$dni_září = 30;
$dni říjen = 31;
$dni_listopad = 30;
$dni_prosinec = 31;
```

To je hrozně nepraktické. Zkusme třeba jiný příklad. Řekněme, že programujeme sklad, ve kterém je několik tisíc druhů zboží. A bude nás zajímat kolik kusů od každého zboží máme na skladě. Založíme tedy několik tisíc proměnných:

```
$kusů_zboží1 = 5;
$kusů_zboží2 = 10;
$kusů_zboží3 = 3;
```

```
/* a tak dále */
```

Tak takto to tedy nejde. Museli bychom být blázni, abychom psali tisíce proměnných. Namísto toho tady existují pole. Jak by se tedy vyřešil pomocí polí příklad s měsíci:

```
$dni["leden"] = 31;
$dni["unor"] = 28;
$dni["březen"] = 31;
$dni["duben"] = 30;
$dni["květen"] = 31;
$dni["červen"] = 30;
$dni["červenec"] = 31;
$dni["srpen"] = 31;
$dni["září"] = 30;
$dni["říjen"] = 31;
$dni["listopad"] = 30;
$dni["prosinec"] = 31;
```

O co tedy u polí jde? Přeci o to, že máme jenom jednu proměnnou, v našem případě se jmenuje \$dni, ale tahle proměnná je vlastně velkým skladištěm hodnot odlišených tzv. indexem. Index je to, co je uvnitř hranatých závorek [a]. Pole mají jakožto skladiště mnoha hodnot velké výhody. Možná si řeknete, když se podíváte na uložení počtu dní v měsíci, řeknete si, kde jsou výhody pole? Vždyť je to skoro stejné jako předtím!

Pole se v PHP hodně často používá, protože na pole je PHP velmi schopné. Výhody pole jsou především s tím, že se s polem zachází jako s celkem. Pro pole existuje řada akcí na automatické zpracování jako celku, se kterými se dají dělat hotové divy.

Snad jenom poznámku, že jako indexy polí se v PHP dají používat čísla a řetězce.

Předdefinované proměnné

V předchozím příkladu byla vložena hodnota do proměnná naším přičiněním. Napsáním třeba \$počet = 3 bylo dosazena do proměnné s názvem \$počet číslo 3. V PHP ovšem existuje spousta proměnných, které dostaly hodnotu odněkud shůry. Jinak řečeno, spousta proměnných má hodnotu, aniž bychom se o to vůbec nějak přičinili. To slouží k tomu, abychom se mohli dozvědět spoustu důležitých údajů. Můžete se například dozvědět spoustu věcí o čtenáři, který si právě prohlíží Vaší stránku.

Pokud chcete vidět všechny předdefinované proměnné, a kromě toho spoustu dalších informací zkuste následující, velmi jednoduchý skript. Funkce phpinfo() slouží ke zjištění současného nastavení PHP serveru.

```
<?php echo phpinfo(); ?>
```

Pokud výše uvedený jednořádkový skript spustíte, objeví se detailní popis současného nastavení PHP serveru přehledně uspořádaný do tabulek, bohužel v angličtině. Dole v úplně poslední tabulce nadepsané PHP Variables pak můžete najít výpis všech předdefinovaných proměnných. Všimněte si, že se jedná o pole.

Následující skript ukáže základní informace o tom, kterým prohlížeč používá čtenář stránky s PHP skriptem:

```
<html>
<head>
<title>Příklad 3. ze 4. dílu</title>
</head>
<body>
<h1><?php echo $_SERVER["HTTP_USER_AGENT"]; ?></h1>
</body>
</html>
```

Jak jste si možná všimli v příkladu uvedeném výše, pro zjištění informací o prohlížeči používám pole s názvem \$_SERVER (na začátku je znak podtržítka). Do tohoto pole ukládá PHP důležité informace související s webem a webovým serverem. Například v proměnné \$_SERVER s indexem HTTP_USER_AGENT jsou obsaženy informace o prohlížeči. Pokud naproti tomu zobrazím hodnotu proměnné \$_SERVER s indexem HTTP_HOST, dozvím se, na jaké doméně jsou PHP stránky uloženy:

```
<html>
<head>
<title>Příklad 4. ze 4. dílu</title>
</head>
<body>
<h1><?php
    echo „Stránky jsou uloženy na doméně: “, $_SERVER["HTTP_HOST"];
?></h1>
</body>
</html>
```

Neexistence proměnné

S proměnnou můžu udělat v podstatě dvě věci. Můžu do proměnné uložit hodnotu a můžu hodnotu použít, třeba jí vypsát. PHP založí každou proměnnou, která byla naplněna nějakou hodnotou. Co se ale stane, pokud se pokusím vypsát hodnotu proměnné, která neexistuje?

Můžeme si to hned vyzkoušet na skriptu, který se pokouší vypsát neexistující proměnnou \$x. Proměnnou \$x jsme nikdy žádnou hodnotou nenaplnili, proto neexistuje:

```
<html>
<head>
<title>Příklad 5. ze 4. dílu</title>
</head>
<body>
<h1><?php
    echo „Výpis neexistující proměnné: “, $x;
?></h1>
</body>
</html>
```

Pokud si výše uvedený skript vyzkoušíte, může nastat jedna ze dvou věcí. Vždycky je výsledkem text „Výpis neexistující proměnné:“. Tedy PHP prostě namísto vypsání \$x nevypíše nic, protože neexistuje. Pokud má PHP v konfiguraci nastaveno, aby upozorňoval na všechny, i ty nejmenší chybičky, pak k výpisu ještě přidá něco jako „*Notice: Undefined variable: x in c:\inet_srv\http\doc_root\null.php on line 7*“. Jinými slovy, PHP nám sděluje že je použita neexistující proměnná s názvem x na 7. řádce.

Použití neexistující proměnné považuje PHP za malou chybičku (tzv. notice). Naprostá většina všech webových serverů má PHP nastaveno tak, že malé chybičky nevypisuje. Snadno se tak ovšem udělá chyba při překlepu, jako například v následujícím skriptu:

```
<html>
<head>
<title>Příklad 6. ze 4. dílu</title>
</head>
<body>
<h1><?php
    $překlep = "toto je překlep";
    echo "Hodnota = ", $přelep;
?></h1>
</body>
</html>
```

Je jasné, že jsme chtěli vypsát hodnotu proměnné \$překlep, ale došlo k překlepu. Pokud PHP je nastaveno tak, že malé chybičky potlačuje, nebudeme na náš překlep v názvu proměnné nijak upozorněni. Proto i „Intranetový server“ vypisuje každou, i nejmenší chybičku.

Pokud přesto někdy budete potřeboval potlačit výpis chybu, je tu jednoduchá rada. Stačí před výraz, který může chybičku vyvolat napsat znak zavináč. Viz další příklad, který zaručuje, že nikdy žádná chyba vypsaná nebude:

```
<html>
<head>
<title>Příklad 7. ze 4. dílu</title>
</head>
<body>
<h1><?php
    echo „Výpis neexistující proměnné: ", @$x;
?></h1>
</body>
</html>
```

PHP - 5. díl – konstrukce if, podmínky a příkaz die

Obsahem 5. dílu seriálu o PHP bude povídání o konstrukci logických podmínkách a vším, co s tím souvisí.

Tento díl je zaměřený na konstrukci if. Samozřejmě v praktickém provedení s dalšími PHP skripty. V tomto díle si možná trochu procvičíte i logické myšlení. Nejdříve však bude malé dokončení k proměnným. Jak jsem byl v diskusi k předchozímu dílu správně upozorněn, opomněl jsem pasáž o velikosti písmen v názvech proměnných.

Velikosti písmen v názvech proměnných

PHP rozlišuje v názvech proměnných velikosti písmen. To znamená, že třeba proměnné s názvy \$počet, \$Počet a \$POČET jsou tři naprosto různé proměnné. Této vlastnosti se odborně

říká "case sensitive", tedy citlivost na velikost písmen. Pozor na to, PHP vás tedy nutí dodržovat stejné velikosti písmen v proměnných, jinak mohou vznikat ošklivé a těžko hledatelné chyby.

Konstrukce if

Téměř každý běžný programovací jazyk (to zní skoro jako běžný prací prášek) má konstrukci if, PHP nevyjímaje. Konstrukce if je v podstatě něco, co umožní provést kus PHP kódu jen za určité podmínky. Následující PHP skript vypíše buď "Právě je dopoledne", nebo "Právě je odpoledne" podle aktuálního času.

```
<html>
<head>
<title>Příklad 1. z 5. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<h1><?php
    if (date("A") == "AM")
        echo "Právě je dopoledne";
    else
        echo "Právě je odpoledne";
?></h1>
</body>
</html>
```

Jak to funguje? Abych to vysvětlil, začnu trochu od podlahy. V příkladu je použita konstrukce if. Celá konstrukce if má zhruba takovýto tvar:

```
if (podmínka)
    příkaz, který se vykoná, pokud je podmínka splněna;
else
    příkaz, který se vykoná, pokud není podmínka splněna;
```

Konstrukce if potom pracuje tak, že se podívá na podmínku. Zjistí, zda je zapsaná podmínka za slovem if splněna, nebo ne, a podle toho vykoná příslušný příkaz. Buď ten, který se má vykonat pokud podmínka splněna je, a nebo příkaz, který se má vykonat, když podmínka splněna není.

V našem příkladě má podmínka příkazu if tuto podobu:

```
date("A") == "AM"
```

Co tato podmínka znamená? Základem je výraz

```
date("A")
```

který jsem vysvětlil ve 3. dílu tohoto seriálu. Tento výraz vrací řetězec AM pro dopoledne, a nebo řetězec PM pro odpoledne.

K tomu, abychom mohli podmínku rozluštit musím vysvětlit, že dvojice znaků == znamená podmínku na rovnost. Tedy v našem případě testujeme, jestli výraz date("A") je stejný (tedy rovná se) jako řetězec "AM". Jinými slovy řečeno, podmínka za slovem if je splněna tehdy,

pokud výraz `date("A")` se rovná řetězci `"AM"`. A to je splněno pouze dopoledne, protože výraz `date("A")` dává `"AM"` pro dopoledne, a `"PM"` pro odpoledne.

Jednoduše shrnuto, podmínka v našem příkladu je splněna pouze pro dopoledne.

Náš příklad tedy pro dopoledne vypíše příkaz, který splňuje podmínku (tedy vypíše `"Právě je dopoledne"`), a jindy vypíše příkaz, který nesplňuje podmínku (tedy vypíše `"Právě je odpoledne"`).

Nedá se ani dostatečně zdůraznit, jak moc užitečná je konstrukce `if`, a jak moc často se v PHP skriptech používá. Upřímně řečeno, bez konstrukce `if` se dokonce spousta věcí ani udělat v PHP vůbec nedá.

Porovnávací operátory

V našem prvním příkladě byla v podmínce použit test na rovnost. Kromě toho je možné zadat v podmínce třeba to, že chceme, aby něco bylo menší, než to druhé a další. Pro vysvětlení budu předpokládat, že mám dvě proměnné s názvem `$a` a `$b`:

Příklad	Vysvětlení
<code>\$a == \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> byla rovná hodnotě proměnné <code>\$b</code>
<code>\$a != \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> nebyla rovná hodnotě proměnné <code>\$b</code>
<code>\$a <> \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> nebyla rovná hodnotě proměnné <code>\$b</code>
<code>\$a < \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> byla menší, než hodnota proměnné <code>\$b</code>
<code>\$a <= \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> byla menší, nebo rovná hodnotě proměnné <code>\$b</code>
<code>\$a > \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> byla větší, než hodnota proměnné <code>\$b</code>
<code>\$a >= \$b</code>	Požadujeme, aby hodnota proměnné <code>\$a</code> byla větší, nebo rovná hodnotě proměnné <code>\$b</code>

Pro lepší pochopení zkusme další příklad:

```
<html>
<head>
<title>Příklad 2. z 5. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<h1><?php
    if (3 > 2)
        echo "Podmínka je splněná";
    else
        echo "Podmínka není splněná";
?></h1>
</body>
</html>
```

Tento příklad je velmi jednoduchý. Podmínka v konstrukci `if` je tato:

3 > 2

Tato podmínka požaduje, aby číslo 3 bylo větší, než číslo 2. Protože trojka je větší, než dvojka, podmínka je tedy automaticky splněná. Podle předpokladů by se tedy měl provést příkaz, který vypíše `"Podmínka je splněná"`.

Logické operátory AND a OR

Někdy potřebujeme složitější podmínky. Řekněme třeba, že máme pracovní dobu od 6:00 do 14:00. Chceme, aby nám PHP skript vypsál, zda je právě pracovní doba, nebo ne. Tady už s jednoduchou podmínkou nevystačíme. Potřebovali bychom testovat nějakou podmínku ve stylu `"hodina je mezi šestkou a čtrnáctkou"`. Ale PHP nám nic takového v podmínkách nenabízí. Když se podíváme trochu výše na tabulku, dá se odvodit, že bychom to zmákli, kdyby šlo použít podmínky dvě. První by testovala, jestli je hodina větší, nebo rovna šestce. A druhá by zase testovala, jestli je hodina menší, nebo rovná čtrnáctce. A pokud by platily obě podmínky najednou, je právě pracovní doba.

No a právě takové "zmnožování" podmínek nabízejí logické operátory. Základní a nejpoužívanější logické operátory jsou v zásadě dva. V našem příkladě s pracovní dobou bychom použili operátor zvaný AND (je vhodné jej číst jako "a zároveň"), kdy požadujeme, aby obě podmínky platily nejednou. Ještě existuje operátor OR (je vhodné jej číst jako "nebo"), kdy žádáme, aby platila alespoň jedna z těch dvou podmínek. Kromě toho existuje ještě logická operace XOR, která se ale používá velmi velmi zřídka, takže se jím zatím detailněji zabývat nebudu.

Operátor	Zápis v PHP	Jiný zápis v PHP	Vysvětlení
AND	<code>A && B</code>	<code>A and B</code>	Podmínka A i podmínka B musí obě platit najednou.
OR	<code>A B</code>	<code>A or B</code>	Musí platit aspoň jedna z podmínek A, nebo B.
XOR	<code>A xor B</code>	<code>A xor B</code>	Musí platit buď A, nebo B, ale ne obě najednou.

Možná vám operátory AND a OR nejsou ještě jasné, ale pomocí příkladů to napravíme. Zde je příklad pro námi uvedenou pracovní dobu:

```
<html>
<head>
<title>Příklad 3. z 5. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<h1><?php
    $hodina = date("G");
    if ($hodina >= 6 && $hodina <= 14)
        echo "Je pracovní doba";
    else
        echo "Není pracovní doba";
?></h1>
```

```
</body>
</html>
```

Příklad pracuje následovně. Nejdříve zjistíme hodinu pomocí výrazu

```
date ("G")
```

a uložíme jí do proměnné s názvem \$hodina. Hned za tím následuje konstrukce if, kde v podmínce je obsažen zápis &&, který označuje operátor AND (= a zároveň). Jinak řečeno, naše podmínka je vlastně složena z podmínek dvou, které musí platit zároveň. První podmínka požaduje, aby hodina byla větší, nebo rovná šesti. Druhá podmínka žádá, aby hodina zároveň byla menší, nebo rovná 14. Obě podmínky najednou platí jen tehdy, pokud je hodina někde mezi šestou a čtrnáctou.

Příkaz die

Pokud jste dočetli až sem, možná jste už trochu unavení. Proto jsem zařadil pasáž o příkazu die, abych vystřídal téma na chvíli něčím jednodušším.

Příkaz s názvem die je trochu morbidní, koneckonců i v angličtině znamená toto slovo "zemřít!". Což také naprosto přesně vystihuje funkci. Pokud použijete příkaz die, PHP skript okamžitě "umírá" a zašeptá vám ještě v posmrtné agónii poslední text, který napíšete jako parametr tohoto příkazu. Cokoli dalšího, co je za příkazem die už se neprovádí, nedojde na to řada, protože skript už "umřel", sotva narazil na příkaz die:

```
<html>
<head>
<title>Příklad 4. z 5. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<h1><?php
    echo "Ještě žiju!<br>";
    die("Právě umírám! Tohle je moje poslední věta!<br>");
    echo "Tohle už nikdy nevypíšu<br>";
?></h1>
</body>
</html>
```

Ve výše uvedeném příkladu se vypíše text "Ještě žiju!", potom volám příkaz die, a v závorkách uvádím, jaká mají být poslední slova, tedy co má skript ještě vypsát, než "umře". Následují větu "Tohle už nikdy nevypíšu" už skript nestihne vypsát.

Příkaz die se velmi hodí v případech, kdy nastane ve skriptu chyba, a pokračovat dál ve vykonávání PHP skriptu už nemá smysl. Například můžete chtít vypsát obsah souboru, který už neexistuje. Nebo nastane jiná chyba. Něco se někde ztratí, není k dispozici, prostě cokoliv vám zabrání úspěšně dokončit to, co jste v PHP naprogramovali.

Dodatek

Chtěl jsem v tomto díle seriálu ještě napsat o způsobu vyhodnocování podmínek, stejně tak jako o operátoru NOT, ale nakonec jsem se rozhodl, že bych tento díl přelácal. I tak je tento

díl trochu obtížnější. V nejlepším se má přestat, a proto tyto věci vysvětlím až o několik dílů dále.

PHP - 6. díl – cyklus for a bloky kódu

Současný díl bude zaměřený na cyklus for. A opět v praktickém provedení s dalšími ukázkovými skripty. Dotkneme se i dalších témat, jako jsou bloky kódu a alternativního způsobu syntaxe pro tento cyklus for.

Cyklus for

Pro slušné programování je potřeba cyklus. Cyklus je kus programu, který se může vykonávat vícekrát dokola, tedy "cyklí". Samotné PHP podporuje více druhů cyklů, v tomto současném díle se budu věnovat pouze jednomu z nich, a to cyklu for.

Zjednodušeně napsáno má cyklus for tento tvar:

```
for (počáteční příkaz; podmínka; příkaz po každém cyklu)
    příkaz uvnitř cyklu;
```

Takto to možná vypadá složitě. Za slovem for jsou v závorce 3 výrazy oddělené vzájemně středníky. Podstatou cyklu for je to, že se neustále dokolečka vykonává opakovaně znovu a znovu "příkaz uvnitř cyklu". Toto opakované vykonávání příkazu uvnitř cyklu má samozřejmě svoje zákonitosti, a je vlastně celé řízeno tím, co je zapsáno v závorce za slovem for.

Celý cyklus for probíhá přesně takto:

- Proveďte se ještě před zahájením cyklu "počáteční příkaz", tedy první výraz v závorce za slovem for. Tento příkaz vlastně ani příliš s průběhem cyklu nesouvisí, je jen takovou přípravou před startem cyklu.
- Teď začne vlastní cyklus.
- Otestuje se "podmínka", tedy druhý výraz v závorce za slovem for. Pokud podmínka platí, je pravdivá, bude následovat další kolo cyklu, tedy pokračuje se bodem 4. Pokud podmínka neplatí, je nepravdivá, cyklus je okamžitě ukončen a dál už se nepokračuje, jde se tedy na bod 7.
- Proveďte se "příkaz uvnitř cyklu".
- Proveďte se "příkaz po každém cyklu", tedy třetí výraz v závorce za slovem for.
- Pokračuje se bodem 3.
- Cyklus je ukončen.

Vypadá to složitě? Celé je to složité, pokud neuvidíme, k čemu se vlastně tento cyklus vymýšlel. Cyklus for je jako stvořený pro cykly s předem daným počtem opakování. Následující příklad vám vypíše text "Ahoj lidi!" tak, že postupně použije všech šest úrovní nadpisu v HTML, tedy značky od <h1> až do <h6>:


```

<html>
<head>
<title>Příklad 1. z 6. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    for ($i = 1; $i <= 6; ++$i)
        echo "<h", $i, ">Ahoj lidi!</h", $i, ">";
?>
</body>
</html>

```

V příkladu tedy vidíte typické použití cyklu for. Zavedl jsem proměnnou \$i, která počítá průchody cyklem. Ve vysvětlení použiji stejné očíslování kroků jako ve vysvětlení o pár odstavců výše:

1. Provede se ještě před zahájením cyklu "počáteční příkaz": \$i = 1. Tedy vytvořil jsem na začátku proměnnou \$i a nastavil její hodnotu na jedničku. Tím je cyklus for připraven.
2. Teď začne vlastní cyklus.
3. Otestuje se podmínka: \$i <= 6. Podmínka se ptá, jestli proměnná \$i obsahuje hodnotu menší, nebo rovnou šestce. To je splněno, cyklus tedy pokračuje.
4. Hned poté se provede "příkaz uvnitř cyklu", který vypadá takto:

```
echo "<h", $i, ">Ahoj lidi!</h", $i, ">";
```

Protože při prvním průchodu obsahuje proměnná \$i jedničku, příkaz echo vypíše toto:
<h1>Ahoj lidi!</h1>

5. Hned poté se provede "příkaz po každém cyklu": ++\$i. Tento zápis neznamena nic jiného, než že se proměnná \$i zvětší o jedničku. Tedy po provedení tohoto příkazu bude obsahovat proměnná \$i dvojku.

Tím máme za sebou první průchod cyklu. V tomto prvním průchodu cyklu se vypsál nadpis první úrovně uvozený mezi <h1> a </h1>. Jsme připraveni na druhý průchod, proměnná \$i je naplněná dvojkou. Jak bude probíhat druhý cyklus?

3. Otestuje se podmínka: \$i <= 6. Podmínka se ptá, jestli proměnná \$i obsahuje hodnotu menší, nebo rovnou šestce. To je splněno, cyklus tedy pokračuje.
4. Hned poté se provede "příkaz uvnitř cyklu", který vypadá takto:

```
echo "<h", $i, ">Ahoj lidi!</h", $i, ">";
```

 Protože ale při druhém průchodu obsahuje proměnná \$i už dvojku, příkaz echo vypíše něco trochu jiného:
<h2>Ahoj lidi!</h2>
5. Hned poté se provede "příkaz po každém cyklu": ++\$i, tedy proměnná \$i zvětší zase o jedničku a bude obsahovat trojku..

Tím máme za sebou druhý cyklus. Vypsali jsme už nadpis druhé úrovně a jsme připraveni na další průchod cyklem. Dál už nebudu pokračovat, je zřejmé, že se to bude opakovat stále dokola.

Protože "podmínka" v cyklu for je nastavena na \$i <= 6, provede se šest průchodů cyklem. Během těchto šesti průchodů se vypíší nadpisy první až šesté úrovně a poté cyklus skončí.

Další příklady na cyklus for

Pro větší názornost zkusme ještě další příklad cyklu for. Bude to příklad, který vypíše všechna čísla od jedné do deseti:

```

<html>
<head>
<title>Příklad 2. z 6. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    for ($i = 1; $i <= 10; ++$i)
        echo $i, "<br>";
?>
</body>
</html>

```

Zkuste se podívat a projít si v myšlenkách, nebo na papíře, všechny kroky, kterými cyklus prochází. Věřím, že to pro vás bude jednoduché. Namísto opakování již napsaného dám k dispozici ještě jeden příklad. Následující PHP skript vypíše součet čísel 1+2+3+ až +100:

```

<html>
<head>
<title>Příklad 3. z 6. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $součet = 0;
    for ($i = 1; $i <= 100; ++$i)
        $součet = $součet + $i;
    echo "Výsledek = ", $součet;
?>
</body>
</html>

```

Bloky kódu

Pokud si všimnete, tak "příkaz uvnitř cyklu" může být jenom jeden. Dost často se vyskytuje potřeba namísto jednoho příkazu tam nacpat příkazů několik. Takhle potřeba může vzniknout nejenom u cyklu for, ale v podstatě všude, kde je předepsán jeden příkaz, ale my potřebujeme tam dostat příkazů více.

A právě tuhle potřebu řeší blok kódu. Blok kódu je vlastně velmi jednoduchý, prostě stačí více příkazů uzavřít do složených závorek { a } a dát to kam potřebujeme. Zde je příklad s cyklem for, který namísto jednoho "příkazu uvnitř cyklu" používá více příkladů v bloku kódu:

```

<html>

```

```

<head>
<title>Příklad 4. z 6. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    for ($i = 1; $i <= 100; ++$i)
    {
        echo $i, " ";
        echo "Ahoj!";
        echo "<br>";
    }
?>
</body>
</html>

```

Ve výše uvedeném příkladu PHP vypíše 100x text "Ahoj!", přičemž to i pěkně očísluje. "Příkaz uvnitř cyklu" je tam použit jako blok o 3 příkazech echo.

Jiná syntaxe cyklu for

Pro cyklus for (stejně tak jako pro některé další konstrukce a cykly) existuje alternativní syntaxe. Ta se hodí zejména v případě, kdy "příkaz uvnitř cyklu" je velmi dlouhý a obsahuje blok mnoha příkazů. Nicméně použit lze i pro jeden příkaz. Alternativní syntaxe příkazu for vypadá takto:

```

for (počáteční příkaz; podmínka; příkaz po každém cyklu):
    první příkaz uvnitř cyklu;
    druhý příkaz uvnitř cyklu;
    ...
endfor;

```

Jak vidíte, alternativní syntaxe se liší tím, že za závorkami za slovem for je dvojtečka. Poté může následovat libovolný počet "příkazů uvnitř cyklu", které se provedou jako jeden blok příkazů. Konec pak PHP pozná podle slova endfor následovaným středníkem.

Zde je předchozí příklad, který vypíše 100x text "Ahoj!" přepsaný do alternativní syntaxe příkazu for:

```

<html>
<head>
<title>Příklad 5. z 6. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    for ($i = 1; $i <= 100; ++$i):
        echo $i, " ";
        echo "Ahoj!";
        echo "<br>";
    endfor;
?>
</body>
</html>

```

Vynechání některých výrazů v cyklu for

Cyklus for obsahuje za závorkami 3 výrazy, jak bylo uvedeno v základním tvaru cyklu for:

```

for (počáteční příkaz; podmínka; příkaz po každém cyklu)
    příkaz uvnitř cyklu;

```

Může se stát, že některé z těchto výrazů nepotřebujeme. Proto kterýkoli ze tří výrazů za závorkami je možné vynechat, stejně tak jako je možné vynechat i "příkaz uvnitř cyklu". Pokud se vynechá "podmínka", stává se cyklus for nekonečným cyklem, který opakuje cyklus stále dokola bez ukončení a musí být ukončen jiným způsobem.

Pokud se vynechá jakýkoli jiný příkaz, než je "podmínka", tak se prostě na místě tohoto příkazu jednoduše nevykoná nic.

PHP - 7. díl – include a spol.

V 7. dílu seriálu o PHP navážeme povídáním o vkládání jiných souborů do PHP skriptů pomocí include a dalších.

Příkaz include - první příklad

Příkaz include slouží k vložení souboru dovnitř PHP skriptu. Je to velmi užitečná věc sama o sobě, a znám lidi, kteří neumějí z PHP skriptů nic jiného, než příkaz include. A ten čile využívají a chválí si ho a jen kvůli němu jim stojí za to PHP využívat.

O co tedy jde? Představte si třeba, že chcete na konec každé své HTML stránky přidat stejnou věc. Řekněme třeba svojí adresu s kontakty. Nebo určitý obrázek, logo, apod.. Jedna z možností samozřejmě je poctivě svou adresu opsat na každou HTML stránku, kterou napíšete. PHP ovšem nabízí i jiné možnosti. Například i tu možnost, že adresu prostě napíšete do zvláštního souboru a pomocí příkazu include jí "vlepíte" do každé vaší stránky.

Zkusme si to tedy. Mějme soubor s adresou, který bude nazván adresa.html a bude to kus HTML kódu představující adresu:

```

<em>Kontakt:</em><br>
<strong>František Vocásek</strong><br>
Divadelní 18<br>
999 99 Lukavice<br>

```

A do každé stránky vložíme tento soubor pomocí příkladu include. Takže by to mohlo vypadat třeba takto:

```

<html>
<head>
<title>Příklad 1. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>

```

```
<body>
<p>Toto je obsah stránky.</p>
<?php
    include "adresa.html";
?>
</body>
</html>
```

Funkce je jasná. Jakmile PHP narazí na příkaz include, vloží obsah souboru, v našem případě tu adresu. Můžeme mít takto třeba stovku stránek se stejnou adresou na konci. Obrovská výhoda je, že pokud nastane změna v adrese, stačí opravit vložený soubor adresa.html a změna se promítne všude, kde jsme adresu vložili.

Stejně tak můžeme mít jiné prvky, které vkládáme do každé stránky. Například každá stránka může mít stejné menu. Tak je potom možné menu vložit pomocí příkazu include. Nebo stejný začátek stránky, atd.. Fantazii se meze nekladou.

Příkaz include - vložení PHP kódu

Příkaz include slouží k tomu, aby bylo možné rozdělit jednu stránku na více souborů. V takovém projektu se pak je možné lépe orientovat, případně každou část může vytvářet i jiná osoba. Jenomže příkaz include kromě vkládání textu považuje vložený soubor za plnohodný PHP skript. To tedy znamená, že PHP číhá, zda ve vloženém souboru nejsou také kusy PHP kódu, a pokud jsou, tak je provede.

Ten samý soubor lze vkládat i vícekrát, pokud to má smysl. Ve většině případů to smysl nemá. Přesto zde uvádím příklad, který vloží jeden soubor vícekrát.

Vložený soubor se bude jmenovat pocitej.php a bude prostě zobrazovat, kolikrát byl vložen:

```
<p><?php
    ++$počet;
    echo "Tento skript byl vložen celkem ", $počet, " krát.";
?></p>
```

Samotný hlavní skript bude vypadat takto:

```
<html>
<head>
<title>Příklad 2. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $počet = 0;
    include "pocitej.php";
    include "pocitej.php";
    include "pocitej.php";
?>
</body>
</html>
```

Celý příklad funguje tak, že vkládaný soubor pocitej.php používá proměnnou s názvem \$počet. Řádka ++\$počet znamená, že se k hodnotě proměnné \$počet přičte jednička. Následující příkaz echo pak vypíše kolikrát byl soubor vložen.

Na tomto příkladu jsem chtěl mimo jiné demonstrovat, že i proměnné mezi hlavním skriptem a vloženým souborem jsou společné.

V hlavním skriptu je nastavena hodnota proměnné \$počet na nulu. Poté se třikrát vloží soubor pocitej.php, a ve výsledku každý z nich vypíše, kolikrát se do té doby vložil.

Co když soubor chybí?

Protože chybička se vždycky vloudí, může nás zajímat, co se stane, když soubor chybí? Můžeme si to zkusit na příkladu:

```
<html>
<head>
<title>Příklad 3. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    include "neexistujici_soubor";
    echo "<p>Nějaký text po použití příkazu include.</p>";
?>
</body>
</html>
```

V příkladu vkládám pomocí příkazu include soubor, který zaručeně neexistuje. A za tímto příkazem vypíšu pomocí příkazu echo nějaký text.

Pokud si příklad spustíte, zjistíte, že PHP vypíše anglické varování (Warning), ve kterém vám sděluje, že nemůže vložit soubor. Taková varování uvidíte na webu poměrně často, neboť i známým webům se stává, že nějaký soubor někde smažou při změnách, nebo zapomenou, a hlášení je na světě. Nicméně přesto všechno to PHP nepovažuje za tak závažnou chybu, aby se PHP skript zhroutil a předčasně skončil. PHP skript pokračuje tedy dál jakoby příkaz include ani neexistoval a následující text vypsáný příkazem echo se objeví.

Je třeba ale říci, že to právě hodně často nechceme. Hodně často se stává, že pokud se nevloží pomocí příkazu include důležitý soubor, ani nemá smysl, aby skript pokračoval. I to ovšem jde PHP říci, a to tak, že namísto příkazu **include** se použije příkaz **require**. Příkaz require dělá naprosto to samé, co include, tedy vkládá soubor. Jediný rozdíl je v tom, jak se zachová, když soubor chybí. Pokud soubor chybí, tak příkaz require předčasně skončí PHP skript a dál se nepokračuje. Zkusme stejný příklad, jako předchozí, ale s příkazem require:

```
<html>
<head>
<title>Příklad 4. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    require "neexistujici_soubor";
    echo "<p>Nějaký text po použití příkazu include.</p>";
```

```
?>
</body>
</html>
```

Pokud si spustíte tuto variantu příkladu, zjistíte, že se zachová odlišně. Anglické hlášení o tom, že nemůže vložit soubor vypíše také, ale rozdíl je, že už to není varování (anglicky Warning), ale "osudná chyba" (anglicky Fatal error). Kromě toho se skript ukončí, takže text u příkazu echo už se nikdy nevypíše.

Nutno říci, že příkazy require a include mohou zkrachovat nejenom z důvodu toho, že neexistují, ale také z jiných příčin. Velmi častou příčinou jsou špatná přístupová práva, tedy skript nemá práva na čtení souboru. To se dost často stává zejména na linuxových serverech. Kromě toho může být soubor poničený, ale to už spadá do kategorie hardwarových chyb.

Zamezení vícenásobného vložení souboru

V jednom předchozím příkladu jsem vložit jeden soubor třikrát. V mnohých případech tomu naopak chceme zamezit. Tedy chceme, aby v případě, kdy je tentýž soubor vkládám podruhé, potřetí, atd., aby se vložení ignorovalo. I na to se v PHP myslí a existují příkazy **include_once** a **require_once**, které slouží pro vkládání pouze jednou. Zkusme třeba použít náš předchozí příklad, kde došlo k trojnásobnému vložení, ale namísto include použijme include_once. Tedy soubor pocitej.php zůstane beze změny a příklad změníme takto:

```
<html>
<head>
<title>Příklad 5. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $počet = 0;
    include_once "pocitej.php";
    include_once "pocitej.php";
    include_once "pocitej.php";
?>
</body>
</html>
```

Pokud si příklad zkusíte spustit, zjistíte, že k vložení dojde pouze poprvé. Podruhé a potřetí se ignoruje.

Příkaz return - předčasné vypadnutí z vloženého souboru

Ještě zde uvedu příkaz **return**. Anglické slovo return znamená v češtině návrat a to zcela vystihuje jeho funkci. Příkaz return v kontextu vloženého souboru slouží k tomu, aby v tomto místě se vkládání zastavilo a zbytek souboru se už nevložil. To je užitečné hlavně ve spojení s podmínkami. Například následující příklad použije vložený soubor k tomu, aby vypsal dopoledne větu "Těšíš na dnešní oběd?" Je to příklad trochu vyumělkovaný, ale demonstruje příkaz return dobře.

Vložený soubor se bude jmenovat obed.php:

```
<?php
    if (date("A") == "PM")
        return;
    echo "Těšíš na dnešní oběd?";
?>
```

Vlastní hlavní skript bude pak vypadat takto:

```
<html>
<head>
<title>Příklad 6. z 7. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    include "obed.php";
?>
</body>
</html>
```

Jak je vidět v příkladu, soubor pocitej.php nejdříve použije výraz date("A"), který vrátí "AM" pro dopoledne a "PM" pro odpoledne. Pokud je odpoledne, tedy výraz date("A") je roven "PM", pak konstrukce if vykoná příkaz return. A ten ukončí vkládání, takže následující příkaz echo už se nevloží. Pokud je dopoledne, vloží se i příkaz echo a věta se vypíše.

PHP - 8. díl – funkce

V 8. dílu seriálu o PHP se bude mluvit o funkcích a se vším, co s tím souvisí.

Funkce - úvod

Funkce jsou v podstatě malé kusy skriptů, které je možné opakovaně používat. To je asi nejjednodušší vysvětlení, co jsou to funkce. Ale musí se k tomu postupně přidat spoustu dalšího výkladu, aby to bylo košér. Pro nejrychlejší pochopení to chce asi první praktický příklad:

```
<html>
<head>
<title>Příklad 1. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    function pozdrav()
    {
        echo "Ahoj!<br>";
    }

    pozdrav();
    pozdrav();
?>
</body>
</html>
```

Jak je v příkladu vidět, každá funkce začíná klíčovým slovem **function**. Za slovem function je název funkce. Každá funkce je pojmenovaná, funkce v našem příkladu se jmenuje pozdrav. Za jménem funkce jsou kulaté závorky. V našem případě jsou prázdné, ale uvnitř mohou být parametry, jak uvidíme dále.

Protože funkce jsou jenom v podstatě kusy skriptu, je potřeba někde ten kus skriptu zapsat. Ten se zapisuje mezi levou složenou závorku { a pravou složenou závorku }. Může tam být libovolná řada příkazů, v našem případě je tam pouze příkaz echo, který vypíše "Ahoj!".

Pod funkcí v našem příkladu je potom dvakrát použita naše funkce s názvem ahoj. Název funkce v podstatě znamená, že jsme získali nový příkaz s názvem pozdrav. Použijeme jej tak, že napíšeme jméno funkce, a za něj kulaté závorky. Jako každý příkaz je pak potřeba ukončit vše středníkem.

Když to shrnu, funkce se sestává ze dvou částí. První část, to je to, co je za klíčovým slovem function až do konce pravé složené závorky }, a to se nazývá definice funkce. Tím určujeme, co funkce vůbec bude provádět. Druhá část je volání funkce. To je ten zápis pozdrav(), který způsobuje, že dojde k vykonání toho malého kousku skriptu.

V našem případě každé volání funkce pozdrav vypíše ahoj. Protože voláme funkci pozdrav dvakrát, vykoná se dvakrát i ten malý kousek skriptu v definici funkce pozdrav. Protože v definici funkce pozdrav je vypsání textu "Ahoj!", vypíše se tedy dvakrát i tento text.

Funkce jsou nesmírně užitečnými pomocníky zejména pro zpřehledňování kódu. Kromě toho umožňují napsat určitou činnost jenom jednou, ale nechat vykonat mnohokrát.

Funkce - parametry

Uvedl jsem funkci pozdrav, ale funkci je možné předat i nějaké hodnoty, tzv. parametry. V následujícím příkladě předám funkci dvě čísla a funkce s názvem sečti vypíše jejich součet.

```
<html>
<head>
<title>Příklad 2. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    function sečti($a, $b)
    {
        echo $a, " + ", $b, " = ", $a+$b, "<br>";
    }

    sečti(1,2);
    sečti(3,4);
?>
</body>
</html>
```

V tomto příkladě je tedy funkce sečti, která má dva parametry pojmenované jako \$a, \$b. Jak je vidět, parametry se zapisují do kulatých závorek za název funkce a oddělují se čárkami. Parametry jsou v podstatě proměnné, proto pro jejich pojmenování a další používání platí všechna pravidla jako pro proměnné.

Funkce sečti tedy dostane při svém volání dvě čísla. První čísla se jakoby uloží do proměnné s názvem \$a, druhé číslo se uloží jakoby do proměnné \$b. Uvnitř funkce se s proměnnými \$a, \$b může zacházet jako s proměnnými.

V definici funkce sečti se vypíše text, který ukazuje, která dvě čísla se sčítají a výsledek. Na konci je volání funkce sečti. Nejříve voláme funkci sečti, abychom se dozvěděli, kolik je 1+2, ve druhém volání potom 3+4.

Celé to funguje tak, že při volání funkce sečti(1,2) se vytvoří dočasně dvě proměnné. Proměnná \$a dostane hodnotu čísla 1 a proměnná \$b dostane hodnotu čísla 2. Pak proběhne ten malý kousek skriptu uvnitř definice funkce sečti. Po vykonání jsou obě proměnné zrušeny.

Pochopení toho, jak pracují funkce s parametry je velmi důležité. Takové funkce se vyskytují v PHP velmi často. Skoro není možné programovat v PHP a nenapsat žádnou funkci s parametry. A ten hlavní důvod je, že i samotné PHP obsahuje spoustu zabudovaných funkcí už v sobě. K mnoha věcem v PHP se není možné dostat jinak, než přes funkce s parametry. Dokonce i v našem seriálu jsme funkce PHP už použili, například funkci die v 5. díle seriálu, nebo funkci date ve 3. díle seriálu.

Příkaz return - předčasné ukončení funkce

Příkaz **return** je možné použít uvnitř definice funkce, a nebo mimo ni. Pokud se použije uvnitř definice funkce, pak dojde k ukončení funkce, a další příkazy v definici funkce už se nevykonávají. Pokud se použije mimo definice funkce, dojde k ukončení vykonávání souboru, o tom viz 7. díl seriálu a příkaz include.

Příklad trochu složitější funkce vyděl je v následujícím příkladu. Jak jistě víte, nulou se dělit nedá, a tak následující příklad se postará, aby k tomu nedošlo:

```
<html>
<head>
<title>Příklad 3. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    function vydel($a, $b)
    {
        if ($b == 0)
            return;
        echo $a, " / ", $b, ` = `, $a/$b, "<br>";
    }

    vydel(10,5);
    vydel(2,0);
    vydel(100,4);
?>
</body>
</html>
```

Funkce vyděl pracuje podobně jako předchozí funkce sečti. V definici funkce vyděl jsou vidět dva parametry s názvy \$a, \$b, což jsou dvě čísla, která se potom vydělí. Na rozdíl od funkce sečti je funkce vyděl maličko složitější. Nejříve se totiž ptáme, jestli platí podmínka, že \$b, tedy druhé číslo, je rovno nule. Pokud tato podmínka platí, dělíme nulou, a to se nesmí. Pak

tedy raději vykonáme příkaz return, tedy ukončení funkce. Tak jsme se postarali o to, že v případě dělení nulou se funkce vyděl předčasně ukončila.

Dále zavoláme funkci vyděl třikrát, ale pokud si skript spustíme, uvidíte jen dva výsledky. Druhé použití funkce vyděl totiž vede k dělení nulou, a tak dojde k předčasnému ukončení funkce vyděl příkazem return.

Funkce strlen - počet znaků řetězce

V této části chci demonstrovat, že funkce umí taky vracet hodnoty. Použiji k tomu tentokrát funkci **strlen**, kterou nemusím definovat, protože patří do sady funkcí zabudovaných přímo v PHP. Funkce strlen očekává jeden parametr, a to řetězec, a celá její činnost spočívá v tom, že nám řekne, kolik má řetězec znaků. Zde je příklad použití:

```
<html>
<head>
<title>Příklad 4. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $kolik = strlen("Ahoj");
    echo $kolik, "<br>";
    echo strlen("Nazdar");
?>
</body>
</html>
```

Zajímavý je tento řádek:

```
$kolik = strlen("Ahoj");
```

Ten vezme řetězec "Ahoj", který je předán jako parametr do funkce strlen. Funkce strlen spočítá, kolik má znaků a zjistí, že celkem čtyři znaky. Výsledek ovšem nikam nevypisuje, ale namísto toho ho vrátí jako hodnotu funkce. Takže v proměnné \$kolik zůstane číslo 4.

Další zajímavý řádek je:

```
echo strlen("Nazdar");
```

Tady máme řetězec "Nazdar", který je předán funkci strlen jako parametr. Ten zjistí, že se jedná o šestiznakový řetězec. Výslednou hodnotu ale rovnou vypisujeme příkazem echo, který vypíše šestku.

Z toho vidíme, že existuje možnost, aby funkce vrátila hodnotu. Jak se to dělá, se dozvíme dále.

Příkaz return - vrácení hodnoty funkce

Příkaz return slouží k předčasnému ukončení funkce. Ale to je jen jedna část jeho funkce. Ve skutečnosti příkaz return slouží i k vrácení hodnoty. Každá funkce totiž může vracet hodnotu, ale blíže viz další příklad, který počítá násobení trochu jinak, než v předchozích příkladech:

```
<html>
<head>
<title>Příklad 5. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    function vynasob($a, $b)
    {
        return $a * $b;
    }

    $výsledek = vynasob(2,3);
    echo $výsledek, "<br>";
    echo vynasob(4,5);
?>
</body>
</html>
```

Začnu s vysvětlením od konce, tedy od volání funkce vynasob. Jedna z řádků v příkladu je tato:

```
$výsledek = vynasob(2,3);
```

Zavolám prostě funkci vynasob pro vynásobení 2x3 a výsledek uložím proměnné \$výsledek. Vypadá to celkem přirozeně. Další taková řádka je:

```
echo vynasob(4,5);
```

Vynásobím prostě 4x5 a výsledek rovnou vypíšu příkazem echo.

Myslím si, že takto to vypadá celkem přirozeně. Teď jde o to ještě funkci naučit, jakou hodnotu funkce vrátí. A to se dělá právě příkazem return. Pokud za příkaz return zapíšu jakoukoli hodnotu, nebo výraz, určím tím, jakou hodnotu vrátí funkce.

Zkusme teď jiný, mnohem jednodušší příklad, který nám vrátí Ludolfovo číslo π , které má hodnotu asi 3.14:

```
<html>
<head>
<title>Příklad 6. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    function p()
    {
        return 3.141592653589;
    }

    echo "Pi = ", p(), "<br>";
?>
</body>
</html>
```

Zde je krásně vidět příkaz return, který vrací hodnotu přímo v akci. Funkce s jednopísmenným názvem p při každém zavolání vrací hodnotu Ludolfova čísla π .

Musím se přiznat k tomu, proč jsem funkci pojmenoval jenom p. Je to proto, že přímo PHP už má v sobě funkci s názvem **pi**, která funguje naprosto stejně jako naše funkce p, tedy vrátí hodnotu Ludolfova čísla π . Můžete tedy funkci pi přímo používat ve svých skriptech.

Funkce mt_rand - náhodné číslo

V podstatě to nejzákladnější o funkcích jsem už vysvětlil. Pokud umíte dobře pracovat s funkcemi, máte velkou část PHP světa k dispozici, protože to, co je zabudováno v PHP s námi komunikuje hlavně pomocí funkcí. Pokud si pořídíte oficiální manuál PHP zjistíte, že je tam popsána velká spousta funkcí, kterou můžete využívat ve svých skriptech. Dokážou opravdu kde co, od odeslání mailu, práce se soubory, různé matematické, grafické, textové a další funkce.

Jako malou demonstraci jsem připravil příklad, který vypíše náhodné oslovení. Tedy přesněji nevypíše náhodné oslovení, ale náhodně vybere jednu ze dvou možností oslovení. K tomu se používá funkce **mt_rand**, která už je zabudována přímo v PHP. Funkce mt_rand vrátí náhodné číslo, má 2 parametry, a to minimální a maximální číslo z rozsahu, který má vrátit. A teď ten příklad:

```
<html>
<head>
<title>Příklad 7. z 8. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $náhodné_číslo = mt_rand(1,2);
    if ($náhodné_číslo == 1)
        echo "Ahoj frajere!";
    else
        echo "Ahoj kamaráde!";
?>
</body>
</html>
```

Základem celého příkladu je řádek:

```
$náhodné_číslo = mt_rand(1,2);
```

Ten použije funkci mt_rand k tomu, aby vrátil náhodné číslo v rozsahu od jedné do dvou (parametry jsou čísla 1 a 2). Protože funkce mt_rand vrátí jen celá čísla, tak náhodně vrátí buď číslo jedničku, nebo dvojku.

Hned za tím je konstrukce if, která otestuje, jestli náhodné číslo je rovno jedničce. Pokud podmínka platí, vypíše se "Ahoj frajere!", jestli podmínka neplatí, vypíše se "Ahoj kamaráde!".

Pokud si tento příklad budete zkoušet, spusťte ho alespoň pětkrát po sobě. Zjistíte, že někdy vám vypíše jednu možnost, někdy druhou, podle náhody.

Poznámka na závěr

Rád bych dodal, že ačkoli tento díl seriálu postihuje snad to nejdůležitější o funkcích, není zde ještě vše, co by bylo dobré o funkcích napsat. Ale vzhledem k rozsahu jsem se rozhodl navázat v některém z dalších dílů.

PHP - 9. díl – práce se soubory a počítadlo stránek

9. dílu seriálu o PHP bude pojednávat o práci se soubory. Jako bonus si uděláme jednoduché textové počítadlo stránek.

Protože už umíme slušné základy PHP z minulých dílů seriálu, máme znalosti k tomu, abychom uváděli příklady, které budou zajímavé a které budou užitečné i v praxi. Pro dnešní díl to bude počítadlo, které nám ukáže, kolikrát si návštěvníci prohlíželi naší stránku. Počítadlo bude využívat znalosti dnešního dílu, a to práce se soubory.

Soubory - úvod

Se soubory se setkal asi každý, kdo si kdy sedl k počítači. Práce se soubory patří k základním věcem, které se dají dělat. V zásadě tento díl bude seznamovat se zabudovanými funkcemi v PHP, které mohou být užitečné pro práci se soubory. Tyto zabudované funkce v PHP je možné rozdělit do dvou částí: Na první část, které pracuje se soubory pomocí tzv. handlů a na druhou část, která handly nepotřebuje. V tomto díle se budu zabývat pouze přístupem pomocí handlů.

Dnešní díl bude trochu rozsáhlejší i náročnější, ale pomůže vám orientovat se v souborech v rámci PHP skriptů. A se soubory se dá dělat velmi mnoho užitečných věcí.

Přístup pomocí handlů - otevření a zavření souboru

Základní přístup při práci se soubory je přístup pomocí handlů. Pokud potřebujeme pracovat s nějakým souborem, můžeme si to představit jako práci se šanonem plným papírů. Nejdříve vezmu šanon, otevřu ho, listuji v něm, čtu si, případně přidám další dokument a nakonec šanon zavřu a uložíím do poličky.

Představa práce se šanonem je analogická k tomu, jak se pracuje se souborem pomocí handlů. Nejdříve se soubor otevře, pracuje se s ním, čtou se, případně zapisují data do souboru a nakonec se zavře. Protože PHP dovoluje mít otevřeno naráz více souborů, má každý otevřený soubor tzv. **handle**, aby se vědělo, se kterým souborem se právě bude pracovat.

A teď konkrétně, pro otevření souboru existuje funkce **fopen**, které můžeme předat 2, nebo 3 parametry. Pro zavření souboru, pak mám funkci **fclose**, která má 1 parametr. Nejjednodušší příklad, který mohu udělat je soubor otevřít, a hned zavřít. I takto jednoduché použití může mít svůj význam, jak je ukázáno v následujícím příkladu:

```
<html>
<head>
<title>Příklad 1. z 9. dílu</title>
</head>
<body>
```

```
<?php
    $handle = fopen("a.txt", "w+");
    fclose($handle);
?>
</body>
</html>
```

Pokud si příklad spustíte, zdánlivě to vůbec nic nedělá. Nic nevypíše, ale jednu věc přesto udělá. Založí nový prázdný soubor s názvem a.txt, který se Vám objeví ve stejném adresáři, jako máte skript. Tedy výsledkem je založení nového souboru. Funkce fopen totiž při otevírání může volitelně založit i nový soubor, pokud ještě neexistuje.

Zkusme si vzít na mušku nejdříve funkci **fopen**, která má tento tvar při použití dvou parametrů:

```
fopen (jméno_souboru, způsob_otevření);
```

Jako první parametr se zadává jméno souboru se kterým chceme pracovat. V našem příkladě to bylo a.txt, ale použít můžeme samozřejmě cokoliv jiného. Pokud to způsob otevření podporuje, soubor s takovým jménem nemusí existovat, a funkce fopen ho pak založí.

Jako druhý parametr se pak zadává způsob otevření souboru. V zásadě jsou tři základní způsoby otevření:

r	ze souboru budeme chtít číst (soubor musí existovat)
w	do souboru budeme chtít zapisovat (pokud existuje soubor, smaže ho, vždy vytvoří nový soubor)
a	budeme chtít připsovat na konec souboru nová data (pokud soubor neexistuje, vytvoří ho)

Kromě toho můžeme za písmenko přidat ještě znak +, aby se umožnilo to druhé, tedy:

r+	budeme chtít číst i zapisovat (soubor musí existovat)
w+	budeme chtít zapisovat i číst (pokud existuje soubor, smaže ho, vždy vytvoří nový soubor)
a+	budeme chtít připsovat na konec souboru nová data i číst (pokud soubor neexistuje, vytvoří ho)

To jsou tedy parametry funkce fopen. Funkce fopen vrátí tzv. handle, což je hodnota, kterou uložíme do proměnné a používáme jí při dalších akcích se souborem. Stejně jako v příkladu. Výsledný handle jde přímo použít i jako podmínka, abychom otestovali, zda všechno dobře dopadlo:

```
<html>
<head>
<title>Příklad 2. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "w+");
    if ($handle)
```

```
        echo "Otevření souboru se povedlo.";
    else
        echo "Otevření souboru selhalo.";
?>
</body>
</html>
```

Výše uvedený příklad obsahuje otevření souboru i s kontrolou, zda všechno proběhlo v pořádku. Protože otevření souboru se nemusí podařit, měli byste vždy udělat tuto kontrolu. Soubor, který požadujete nemusí existovat, nebo k němu nemáte práva a spousta jiných dalších příčin může způsobit selhání.

Pokud někdy soubor otevřete, mělo by někdy dojít k zavření souboru. Pokud to neuděláte v PHP skriptu, tak na konci skriptu PHP samo všechny soubory zavře. A nebo to můžete udělat vy sami funkcí **fclose**, která má takový tvar:

```
fclose (handle);
```

Jako parametr předáte handle získaný pomocí funkce fopen. Po zavolání funkce fclose je handle dále už neplatný a nedá se použít pro práci se souborem. Tady je tedy celý příklad s otevřením i zavřením souboru a kontrolu selhání:

```
<html>
<head>
<title>Příklad 3. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "w+");
    if ($handle)
        echo "Otevření souboru se povedlo.";
    else
        die("Otevření souboru selhalo.");

    fclose($handle);
?>
</body>
</html>
```

Zápis do souboru

Pokud jste to vydrželi až sem, vážení čtenáři, ukážu vám, jak zapisovat do souboru. Nejlépe přímo na příkladu. Následující příklad přepíše do souboru slovo "Haló":

```
<html>
<head>
<title>Příklad 4. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "a");
    if ($handle)
        echo "Otevření souboru se povedlo.";
    else
```



```

        die("Otevření souboru selhalo.");

fwrite($handle, "Haló");

fclose($handle);
?>
</body>
</html>

```

Příklad obsahuje už známé části, a kromě toho obsahuje funkci **fwrite**, která slouží pro zápis do souboru. Má dva parametry, jako první se jí musí předat handle a jako druhý se jí předá, co se má zapsat do souboru.

Protože soubor je otevřen tak, že druhý parametr funkce fopen je "a", tak se veškeré zápisy do souboru připsují na konec. Pokud tedy spustíte tento příklad vícekrát, slovo "Haló" se napíše na konec několikrát, což si můžete vyzkoušet.

Pro vyzkoušení příkladu se pak musíte podívat do souboru s názvem a.txt, jestli obsahuje slovo "Haló".

Funkce **fwrite** má následující tvar:

```
fwrite (handle, co_se_má_zapsat, maximální_délka_zápisu);
```

Funkce fwrite může mít dva, nebo tři parametry. Pokud použijeme poslední třetí parametr, je to číslo, které říká, kolik znaků se má maximálně zapsat. Funkce fwrite si pak dá pozor na to, aby nezapsala více. Následující příklad je stejný jako předchozí, ale říká, že se mají zapsat maximálně dva znaky z "Haló", zapíše se tedy jenom "Ha":

```

<html>
<head>
<title>Příklad 5. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "a");
    if ($handle)
        echo "Otevření souboru se povedlo.";
    else
        die("Otevření souboru selhalo.");

    fwrite($handle, "Haló", 2);

    fclose($handle);
?>
</body>
</html>

```

Čtení ze souboru

Čtení ze souboru se provádí pomocí funkce **fread**. Zkusme hned přejít na příklad:

```

<html>
<head>
<title>Příklad 6. z 9. dílu</title>

```

```

<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "r");
    if ($handle)
        echo "Otevření souboru se povedlo.<br>";
    else
        die("Otevření souboru selhalo.");

    $načteno = fread($handle, 1000);

    echo "Přečetl jsem : ", $načteno;

    fclose($handle);
?>
</body>
</html>

```

Příklad přečte, co jsme až dosud do souboru a.txt v předchozích příkladech zapsali. Samotné čtení se provádí funkcí fread, která má dva parametry. Prvním parametrem je už obligátní handle a druhým parametrem je kolik znaků máme maximálně načíst. V našem příkladě načteme maximálně 1000 znaků.

Pozice v souboru

Každý otevřený soubor má přiřazeno myšlené ukazovátko, které ukazuje z jakého místa v souboru se provede příští zápis, nebo čtení. Po otevření souboru se bude číst ze začátku souboru, nebo zapisovat na začátek souboru (pokud není použit způsob otevření "a"), a ukazovátko pozice je tedy na začátku souboru.

Pozice ukazovátko se vyjadřuje ve znacích, které jakoby musíme přeskočit, abychom se k současné pozici dostali. Je-li tedy ukazovátko pozice na začátku souboru, pozice je nula. Pokud přečtu 4 znaky, posune se ukazovátko na pozici 4, a další čtení, nebo zápis bude pokračovat z této pozice.

Jednodušší je asi pochopení na příkladu. Použiji funkci **ftell**, která mi vrátí pozici ukazovátko:

```

<html>
<head>
<title>Příklad 7. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $handle = fopen("a.txt", "r");
    if ($handle)
        echo "Otevření souboru se povedlo.<br>";
    else
        die("Otevření souboru selhalo.");

    $pozice = ftell($handle);
    echo "Pozice ukazovátko je ", $pozice, "<br>";

    $načteno = fread($handle, 4);
    echo "Přečetl jsem : ", $načteno, "<br>";

```

```
$pozice = ftell($handle);
echo "Pozice ukazovátka je ", $pozice, "<br>";

fclose($handle);
?>
</body>
</html>
```

V příkladu je po otevření souboru použita funkce `ftell` ke zjištění, na jaké pozici je ukazovátko v souboru. Mělo by se vyspat, že je na nulté pozici, protože je na začátku souboru. Poté přečtu ze souboru 4 znaky a znovu se zeptám na pozici. Měl bych dostat číslo 4.

Samo o sobě je pozice v souboru zajímavá i proto, že je možné pozici ovlivňovat. Například funkce **rewind** vrátí pozici znovu na začátek souboru, takže je možné číst, nebo zapisovat soubor znovu od začátku. Tuto funkci použiji ve svém počítadle stránek.

Počítadlo stránek

Počítadlo stránek bude využívat soubor s názvem `kolik.bin`, ve kterém si bude pamatovat, kolikrát byla naše stránka už návštěvníky prohlédnuta. Po každém prohlédnutí stránky se zvětší číslo v souboru `kolik.bin` o jedničku:

```
<html>
<head>
<title>Příklad 8. z 9. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$handle = fopen("kolik.bin", "a");
fclose($handle);

$handle = fopen("kolik.bin", "r+");
$kolik = fread($handle, 10);
++$kolik;
rewind($handle);
fwrite($handle, $kolik);
fclose($handle);

echo "Počet zobrazení stránky: ", $kolik;
?>
</body>
</html>
```

Pokud si toto počítadlo vyzkoušíte, zjistíte, že pokaždé, když znovu spustíte počítadlo, přiskakují vám tam počty zobrazení stránek. A jak to funguje?

První 2 řádky v PHP mají za úkol jenom vytvořit soubor `kolik.php`, pokud by čistě náhodou neexistoval:

```
$handle = fopen("kolik.bin", "a");
fclose($handle);
```

Způsob otevření "a" je vhodný pro takovou akci. Pokud soubor s názvem `kolik.php` ještě neexistuje, pak jej způsob otevření "a" nechá beze změny. Pokud ještě neexistuje, pak jej

způsob otevření "a" vytvoří. Protože ale použitý způsob otevření nám nevyhovuje pro další práci, soubor hned poté zavřeme a otevřeme ho jinak. Důležité pro nás je, že po provedení prvních dvou řádků PHP kódu zaručeně existuje soubor s názvem `kolik.bin`.

Na dalším řádku otevíráme soubor pro čtení i zápis. Použijeme proto způsob otevření "r+":

```
$handle = fopen("kolik.bin", "r+");
```

O řádek dál přečtu prvních 10 znaků ze souboru. To pro mě bude znamenat uložené číslo, kolikrát se moje stránka už zobrazila:

```
$kolik = fread($handle, 10);
```

Toto číslo musím zvětšit o jedničku, protože teď se zobrazuje právě tato stránka znovu:

```
++$kolik;
```

Další řádky pak slouží k tomu, aby tohle nové číslo bylo zapsáno zpět do souboru `kolik.bin`. Funkcí `rewind` si posunu ukazovátko pozice zpátky na začátek, abych číslo zapsal zpět na začátek souboru:

```
rewind($handle);
fwrite($handle, $kolik);
```

Pak následuje už jen zavření souboru a pak vypsaní počtu zobrazení.

PHP - 10. díl – formuláře

Dnešní díl PHP bude věnován oblasti formulářů, tedy zadávání dat od čtenářů a návštěvníků stránek.

Formuláře - úvod

Formuláře slouží k zadávání dat uživatelem webových stránek. Samo o sobě formuláře nejsou součástí PHP, ale už HTML jazyka. V samotném HTML jazyce existují prvky, které slouží pro zadávání dat a posílání těchto dat jiným stránkám, nebo skriptům. Nutno říci, že mnoho lidí ve formulářích tápe, i když jinak třeba umějí zbylé části HTML excelentně. Proto dnes učiním výjimku, a částečně formuláře vysvětlím, ač se to spíše týká HTML. Upozorňuji, že se budu soustředit spíše na použití formulářů v PHP skriptech, a proto některé věci mírně zjednoduším.

Značka <form>

Jednouše vzato, formulář je vše, co je v HTML mezi tagy `<form>` a `</form>`. Jedna HTML stránka může obsahovat jeden, žádný, nebo libovolně mnoho formulářů na jedné stránce. Protože formulář slouží k posílání dat, má značka `<form>` několik atributů, které slouží právě k určení, kam se data z formuláře mají poslat, a jakým způsobem.

Značka `<form>` v obecném pojetí může mít atributy **action**, který určuje kam se budou data posílat. Dalším atributem je **method**, který určuje jakým způsobem se budou data posílat. Existují ještě některé další atributy, ale ty už pro dnešní vysvětlování nebudou podstatné. Typické použití formuláře je například takového:

```
<form action="mujskript.php" method="post">
<!-- datové položky formuláře -->
</form>
```

Jak je vidět, za atribut action se dosazuje adresa PHP skriptu, který přijme zasláná data. Jako hodnotu atributu method se dosazuje nejčastěji **get**, nebo **post**. Rozdíl mezi get a post je ve způsobu zaslání dat. Metoda get zašle data jako součást adresy webové stránky, zatímco metoda post zašle data odděleně, takže nejsou vidět v adrese webové stránky. Metoda post je schopná zpracovat v principu rozsáhlejší data, a pokud nevíte, doporučuji používat spíše post.

Akce submit - odeslání dat formuláře

Data z formuláře se odesílají speciální akcí, která se hodně spojuje s anglickým slovem **submit**. Asi nejjednodušším způsobem je umístit tlačítko typu "submit" do formuláře. Vyzkoušejte si třeba následující kus HTML kódu:

```
<form action="mujskript.php" method="post">
<!-- datové položky formuláře -->
<input type="submit" value="Odeslat data formuláře">
</form>
```

Pokud si zkusíte zobrazit tento kód zjistíte, že se vám objeví tlačítko s nápisem "Odeslat data formuláře" a po kliknutí na tlačítko dostanete nejspíše chybovou zprávu prohlížeče. Po kliknutí na tlačítko se totiž formulář pokusil odeslat data skriptu, který je zapsán v atributu action u značky `<form>`. To je u nás skript s názvem "mujskript.php", který zatím nemáme, takže pokus o odeslání dat formuláře skriptu skončil chybou.

Přidáváme datové údaje

Zatím jsem použil jenom prázdný formulář, který má prázdná, tedy žádná data. Data se přidávají za značku `<form>` speciálními HTML značkami, jako jsou třeba značky `<input>`, `<select>`, `<textarea>` a další. Zkusme třeba přidat jednoduché políčko, do kterého napíšeme text (uložte si následující příklad do souboru form1.html):

```
<form action="mujskript.php" method="post">
Zadej jméno: <input type="text" name="jmeno"><br>
<input type="submit" value="Odeslat data formuláře">
</form>
```

Po spuštění výše uvedeného HTML kódu by se měla objevit i kolonka na zadání jména. Pod tím je stále tlačítko na odeslání dat formuláře, ale protože nemáme náš skript, odeslání skončí chybou, pokud klikneme na toto tlačítko.

Přijímáme data formuláře

Jak je vidět, formuláře tedy potřebují dva skripty, jeden, který obsahuje data formuláře, a druhý, který odeslaná data přijme.

Zkusme si nechat výše uvedený poslední příklad HTML kódu s formulářem a jednou kolonkou na jméno a zkusme k němu napsat PHP skript, který by data přijmul a zpracoval. Takový PHP skript může být velmi jednoduchý (je potřeba ho uložit jako soubor mujskript.php):

```
<html>
<head>
<title>Příklad 4. z 10. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $jmeno = $_POST["jmeno"];
    echo "Vaše jméno je: ", $jmeno
?>
</body>
</html>
```

Pro vyzkoušení příkladu spusťte form1.html, tedy předchozí příklad na formulář. Zadejte do kolonky nějaké jméno a klikněte na tlačítko "Odeslat data formuláře". Dojde k tomu, že se sám nahraje náš skript s názvem mujskript.php a předají se mu data. Jméno, které jste zadali do kolonky pak skript vypíše.

Celý příklad obsahuje pouhé dvě řádky PHP kódu. Na první řádce přijme jako data hodnotu kolonky s názvem jmeno (zadali jsme jí ve formuláři jako name="jmeno" ve značce `<input>`) a uloží jméno do proměnné \$jmeno.

```
$jmeno = $_POST["jmeno"];
```

Používám zde pole s názvem \$_POST, protože jsem data předával metodou post (viz atribut method ve značce `<form>`), pokud bych předával metodou get, použil bych namísto toho pole s názvem \$_GET.

Na druhé řádce prostě jméno jenom vypisuji pomocí příkazu echo.

Další příklad - zadejte správné heslo

Pomocí formulářů jde udělat velmi mnoho věcí, protože dovoluji reagovat na to, co zadá uživatel. Jedna z možností je kontrola, zda zadal správné heslo:

Vlastní formulář bude vypadat takto (uložte ho do souboru vložheslo.html):

```
<form action="prijmiheslo.php" method="post">
Zadej heslo pro vstup: <input type="password" name="heslo"><br>
<input type="submit" value="Odeslat heslo">
</form>
```

PHP skript, který přijímá data bude vypadat takto (uložte ho do souboru s názvem prijmiheslo.php):

```
<html>
<head>
<title>Příklad 6. z 10. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
```

```

</head>
<body>
<?php
    $heslo = $_POST["heslo"];
    if ($heslo == "lokomotiva")
        echo "<h1>Heslo je správně!</h1>";
    else
        echo "<h1>Heslo není správně</h1>";
?>
<a href="vlozheslo.html">Klikni sem a zkus to znovu</a>
</body>
</html>

```

Pro zkoušení je třeba nastartovat stránku vlozheslo.html a zadat heslo. Správné heslo je lokomotiva, pokud zadáme jiné bude to brát jako špatné heslo.

Ve formuláři máme jedinou kolonku, a to s názvem heslo (name="heslo" ve značce <input>). Tato kolonka se při kliknutí na tlačítko "Odeslat heslo" pošle našemu skriptu s názvem prijmiheslo.php. Ten nejdříve zjistí v poli \$_POST jaké že jsme to zadali heslo. Potom heslo porovná s řetězcem "lokomotiva" a pokud jsou shodné, vypíše, že heslo je správné. V opačném případě vypíše, že heslo je špatné. Pod tím vším je odkaz, kde je to možné zkusit znovu.

Jednoduchá kalkulačka na sčítání

Jako další příklad uvedu jednoduchou kalkulačku, která umí sečíst dvě čísla, která napíšete do kolonky. V tomto příkladu budeme pracovat s formulářem už se dvěma kolonkami.

Takže vzhůru na to, následující příklad si uložte do souboru s názvem kalkulačka.html:

```

<form action="pocitej.php" method="post">
Zadej první číslo: <input type="text" name="prvni"><br>
Zadej druhé číslo: <input type="text" name="druhe"><br>
<input type="submit" value="Spočítej součet">
</form>

```

A přijímací skript, který uložte do souboru pocitej.php:

```

<html>
<head>
<title>Příklad 8. z 10. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $prvni = $_POST["prvni"];
    $druhe = $_POST["druhe"];
    $vysledek = $prvni + $druhe;
    echo $prvni, " + ", $druhe, " = ", $vysledek;
?>
<br>
<a href="kalkulacka.html">Zadej další příklad</a>
</body>
</html>

```

Pro vyzkoušení příkladu je potřeba spustit nejdříve soubor kalkulacka.html a druhý skript s názvem pocitej.php pak dostane automaticky data po kliknutí na tlačítko "Spočítej součet".

PHP - 11. díl – psaní řetězců pod lupou

Dnešní díl PHP bude věnován detailnějšímu pojednání o textových řetězcích.

Do této chvíle jsem všechny textové řetězce uzavíral do dvojitých uvozovek, například "Ahoj". Ovšem to není v PHP jediná možnost. Proto v následujících kapitolách proberu další možnosti. Možnosti zápisu budou rozebrány i z hlediska jejich dalších možností.

V zásadě jsou tři možné způsoby, jak psát textové řetězce. První možnost je uzavřít řetězec do apostrofů, jako například: `Jak se máš?`. Druhá možnost je uzavřít řetězec do uvozovek, jak jsme to zatím psali všude, například "Jak se máš?". A třetí možnost nazývaná v dokumentaci heredoc syntaxe je trochu komplikovanější, povíme si o ní na konci.

Řetězce uzavřené v apostrofech

Řetězce uzavřené v apostrofech jsou nejjednodušší možné řetězce. Jak poznáte porovnáním dále, jsou řetězce uzavřené v apostrofech ty nejjednodušší, které toho tolik neumějí. Jejich výhodou je, že jsou to prostě řetězce ve své nejčistější podobě. A pokud nebudete chtít různé ty serepetičky, které, jak poznáte později, jsou navěšeny na řetězcích uzavřených uvozovkách, budete dávat přednost řetězcům v apostrofech.

Protože řetězce v apostrofech jsou ty nejjednodušší, a PHP k nim nic dalšího nepřidává, měly by teoreticky být i mírně rychlejší při zpracování, než další způsoby zápisů řetězců. Používejte proto uvozovky uzavřené v apostrofech přednostně před dalšími způsoby.

Pokud byste chtěli napsat jako součást řetězce uzavřeného v apostrofech přímo znak apostrof, je to také možné. Prostě namísto jednoho znaku apostrof napíšete dovnitř dva znaky ``, tedy zpětné lomítko a pak samotný apostrof. Viz následující jednoduchý příklad:

```

<html>
<head>
<title>Příklad 1. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Takhle se vypisuje apostrof: \`.`;
?>
</body>
</html>

```

Jak je vidět, i apostrof se dá vypsát. Ještě speciální způsob má samotný znak zpětné lomítko \. Pokud chcete napsat zpětné lomítko, je dobré ho zdvojit. Tedy napíšete-li dvakrát za sebou zpětné lomítko takto: \\, ve skutečnosti jste vložili jen jedno:

```

<html>
<head>
<title>Příklad 2. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Tohle je jedno zpětné lomítko: \\.`;
    echo `Tohle jsou dvě zpětná lomítka: \\\\.`;
?>
</body>
</html>

```

Těmto dvojicím znaků, které začínají zpětným lomítkem \ se říká escape sekvence.

Nic jiného se s řetězcem uzavřeným v apostrofech neděje. Pokud tedy nemáte s textovým řetězcem další záměry, dejte mu přednost.

Řetězce uzavřené v uvozovkách - expanze proměnných

Řetězce, které jsou uzavřené v uvozovkách jsou daleko prošípanější než různé přidání hodnotou. To na jedné straně může být velmi příjemné, na druhé straně v některých případech bude těžké ohlídat všechny možnosti.

Základním rozšířením jsou tzv. expanze proměnných. Pokud dovnitř řetězce napíšeme název proměnné začínající dolarem, pak se dovnitř řetězce vloží hodnota proměnné. Viz třeba tento malý příklad:

```

<html>
<head>
<title>Příklad 3. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $jmeno = "Pavel";
    $napoj = "pivo";
    echo "$jmeno je dobrý kamarád!<br>";
    echo "$jmeno pije $napoj.<br>";
?>
</body>
</html>

```

Pokud si příklad vyzkoušíte, zjistíte, že namísto řádky

```
echo "$jmeno je dobrý kamarád!<br>";
```

bude vypsáno "Pavel je dobrý kamarád!", protože PHP si za \$jmeno dosadil proměnnou \$jmeno, která má hodnotu "Pavel". Obdobně:

```
echo "$jmeno pije $napoj.<br>";
```

se vypíše jako "Pavel pije pivo.", protože zde dojde k expanzi dvou proměnných, a to proměnné \$jmeno, která má hodnotu "Pavel" a proměnné \$napoj, která má hodnotu "pivo".

Expanze proměnných je celkem užitečná věc v mnoha případech, je třeba počítat s tím, že PHP se snaží vytvořit co nejdělsí jméno proměnné, kterou nahrazuje. Takže třeba následující příklad selže:

```

<html>
<head>
<title>Příklad 4. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $napoj = "šarátice";
    echo "Pavel pije $napoje.<br>";
?>
</body>
</html>

```

V tomto příkladu mám proměnnou \$napoj, ale v řetězci se hledá proměnná \$napoje. Výsledkem je, že proměnnou \$napoje nenajde. Což v některých případech může skončit i anglickým hlášením o chybějící proměnné. Každopádně jsme Pavla od šarátice zachránili. A jak to má vypadat správně, aby se doplnila proměnná \$napoj? Takto (máte hned dvě možnosti, jak to napravit):

```

<html>
<head>
<title>Příklad 5. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $napoj = "šarátice";
    echo "Pavel pije {$napoj}e.<br>";
    echo "Pavel pije ${napoj}e.<br>";
?>
</body>
</html>

```

Jak je vidět ve výše uvedeném příkladu, je v takovém případě potřeba do složených závorek { a } oddělit tu část jména proměnné, která je platná. Dolar, který předchází jménu proměnné můžeme napsat dovnitř složené závorky, a nebo před ní, jak chcete. Vyjde vám jen neobvyklý řetězec "Pavel pije šaráticee", což je v češtině špatně, ale berte to jako příklad na funkčnost.

Složené závorky { a } se dají použít i tehdy, pokud chci vypsát proměnnou, která je součástí pole jako třeba v následujícím příkladu, který využije předdefinovanou proměnnou. Příklad vypíše informace o Vašem prohlížeči:

```

<html>
<head>
<title>Příklad 6. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo "Informace o prohlížeči: {$_SERVER['HTTP_USER_AGENT']}.";
?>
</body>
</html>

```

Řetězce uzavřené v uvozovkách - escape sekvence

Protože se tvůrci PHP snažili, aby pomocí řetězců uzavřených v uvozovkách šly napsat opravdu všechny znaky, vytvořily se tzv. escape sekvence. Což jsou sekvence znaků, které začínají znakem zpětné lomítka \, a které nám umožňují zapsat opravdu všechny znaky. Pro zapsání zpětného lomítka napíšeme dvě zpětná lomítka za sebou \\. Pro zapsání samotných uvozovek napíšeme \", pro zapsání dolaru napíšeme \\$. Řetězce v dvojitéch uvozovkách umí také \n pro znak nového řádku (LF), \r pro návrat vozíku (CR) a \t pro tabulátor:

```
<html>
<head>
<title>Příklad 7. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo "Toto jsou uvozovky \".<br>";
    echo "Toto je zpětné lomítko \\.<br>";
    echo "A toto je znak dolar \$.<br>";
?>
</body>
</html>
```

Užitečné také je zapsání znaku v hexadecimální notaci. Zapiše se tak, že napíšu dvojici znaků \x a pokračuji jedním, nebo dvěma hexadecimálními číslicemi. Pokud nechápete, co je to hexadecimální notace, tak tento odstavec přeskočte. Například mezeru můžu napsat jako \x20 v hexadecimální notaci.

Heredoc syntaxe

Heredoc syntaxe je užitečná u skutečně dlouhých řetězců. Rovnou uvedu jeden příklad:

```
<html>
<head>
<title>Příklad 8. z 11. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
echo <<<QQQ
Nějaký dlouhý text, který potřebujete napsat jako jeden řetězec
je možné napsat v here doc syntaxi. Uvnitř tohoto řetězce můžete
používat vše, na co jste zvyklí z řetězců uzavřených v uvozovkách,
tedy i expanzi proměnných a escape sekvence.
QQQ;
?>
</body>
</html>
```

Heredoc syntaxe začíná <<< a za ním nějakým textem. V našem případě jsem zvolil QQQ, důležité je, že stejným textem QQQ řetězec i skončí. Můžete samozřejmě použít cokoli jiného namísto QQQ, důležité je, aby se ten text nevyskytoval uvnitř řetězce. Mezitím je dlouhý text, který je rozdělený na několik řádek.

Samotná heredoc syntaxe má všechny možnosti, které má řetězec uzavřený v uvozovkách, tedy i expanzi proměnných a také escape sekvence.

Závěr: Kdy používat který zápis?

Pokud bych měl shrnout, kdy doporučuji který zápis, napsal bych to asi takto. Pokud nečekám expanzi proměnných a nepotřebuji některé speciální znaky, uzavírám řetězce do apostrofů. Z hlediska HTML a XHTML značkách jsou řetězce zapsané v apostrofech i jednodušší a čitelnější.

Pokud potřebuji expanzi proměnných, uzavírám znaky do uvozovek.

Co se týká heredoc syntaxe, tu používám u skutečně dlouhých řetězců.

PHP - 12. díl – existence a typy proměnných

Dnešní díl PHP bude věnován ověření existence proměnné a jednotlivým typům proměnných.

Zjištění existence proměnné a poslání proměnné "k čertu"

Často používanou věcí je možnost zjištění, zda proměnná s určitým názvem vůbec existuje. Můžeme se tak vyhnout někdy chybovým hlášením, někdy chybám v programu samotném. Pro zjištění, zda proměnná vůbec existuje je používána funkce **isset**. Samotná funkce **isset** se dá používat v podmínkách:

```
<html>
<head>
<title>Příklad 1. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $a = 3;

    if (isset($a))
        echo `Proměnná $a existuje.<br>`;
    else
        echo `Proměnná $a neexistuje.<br>`;

    if (isset($b))
        echo `Proměnná $b existuje.<br>`;
    else
        echo `Proměnná $b neexistuje.<br>`;
?>
</body>
</html>
```

Ve výše uvedeném příkladu byla založena proměnná s názvem \$a s hodnotou 3. Pak jsem se ptal pomocí **isset(\$a)**, zda proměnná a existuje. Pak jsem se pomocí **isset(\$b)** zeptal, zda existuje proměnná \$b. Pokud si příklad vyzkoušíte, zjistíte, že skript vypíše, že proměnná \$a existuje, zatímco proměnná \$b neexistuje.

Funkce `isset` je také používána k nastavení proměnné na nějakou základní hodnotu, pokud daná proměnná neexistuje. Dost často se používá třeba ve spojení s formuláři, apod..

Pokud v PHP vytvoříte proměnnou, za normálních okolností se jí už nezabýváte. Proměnná, kterou jste jednou vytvořili už existuje dál až do konce skriptu. Po celou dobu skriptu zabírá kus paměti počítače, a to i tehdy, když už jí na nic nepotřebujete. Někdy se proto hodí mít možnost takovou proměnnou zrušit, poslat jí "k čertu". Tím uvolníte prostředky počítače, což se v náročnějších PHP skriptech může hodit. Pro takovou příležitost oceníte funkci **`unset`**:

```
<html>
<head>
<title>Příklad 2. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $a = 3;

    if (isset($a))
        echo `Proměnná $a existuje.<br>`;
    else
        echo `Proměnná $a neexistuje.<br>`;

    unset($a);

    if (isset($a))
        echo `Proměnná $a existuje.<br>`;
    else
        echo `Proměnná $a neexistuje.<br>`;
?>
</body>
</html>
```

V tomto příkladě je založená proměnná `$a` s hodnotou 3. Potom se vypíše, zda proměnná existuje. Pak je proměnná `$a` smazána pomocí funkce `unset` a znovu se vypíše, zda proměnná `$a` existuje. Pokud si příklad spustíte, zjistíte, že poprvé bude tvrdit, že proměnná `$a` existuje, a po druhé bude tvrdit, že naopak neexistuje.

Typy proměnných

Každá proměnná má nějakou hodnotu, a to hodnotu určitého typu. Proměnná může obsahovat nejčastěji číslo, nebo textový řetězec. To, jaký druh hodnoty proměnná obsahuje se nazývá typem proměnné. Pokud obsahuje proměnná celé číslo, nazývá se tento typ **`integer`**, pokud číslo s desetinnou čárkou, nazývá se typ **`double`**. Textový řetězec je typ **`string`**, a ještě jsme v minulých dílech občas použili typ pole nazvaný vnitřně v PHP jako **`array`**. Existují ještě další typy proměnných, se kterými se později seznámíme.

Pro zjištění, jaký typ obsahuje proměnná slouží funkce s názvem **`gettype`**, která přímo vrátí název typu:

```
<html>
<head>
<title>Příklad 3. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
```

```
<body>
<?php
    $a = 3;
    echo `Proměnná obsahuje typ `, gettype($a), "<br>";
    $a = 3.14;
    echo `Proměnná obsahuje typ `, gettype($a), "<br>";
    $a = "ahoj";
    echo `Proměnná obsahuje typ `, gettype($a), "<br>";
?>
</body>
</html>
```

V příkladu je použita 3x funkce `gettype`, aby nám postupně vypsala typ hodnoty. Pokaždé předtím dáme proměnné `$a` jinou hodnotu jiného typu. Tento příklad by měl, pokud si jej zkusíte spustit, vypsát toto:

```
Proměnná obsahuje typ integer
Proměnná obsahuje typ double
Proměnná obsahuje typ string
```

A proč se vůbec o typech proměnných bavím? Prostě proto, že PHP s každým typem zachází trochu jinak. Pro každý typ jsou legální trochu jiné operace s proměnnými.

PHP je ohledně typů hodně benevolentní programovací jazyk. Pokud někde očekává jiný typ, než mu dáte, snaží se typ převést na ten, který potřebuje, pokud to jde. Pak jde napsat i takové kuriózní skripty, jako je následující příklad:

```
<html>
<head>
<title>Příklad 4. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $vysledek = "5 jablek" + "3 jablka";
    echo $vysledek;
?>
</body>
</html>
```

Pokud si příklad zkusíte spustit, měl by vypsát číslo 8. Jak k němu PHP dospěl? Na řádku:

```
$vysledek = "5 jablek" + "3 jablka";
```

je použito sčítání, ale PHP umí sčítat jen čísla, neumí sčítat řetězce, jako je to v našem příkladě. Přesto si PHP poradí. Zjistí, že řetězec "5 jablek" začíná číslem 5, a proto si ho vnitřně převede na číslo 5. Druhý řetězec si stejným postupem převede na číslo 3. A ve skutečnosti tedy výše uvedený řádek počítá jako:

```
$vysledek = 5 + 3;
```

Jak je vidět, PHP je hodně přizpůsobivý jazyk a snaží se odhadnout, jaký typ proměnné chcete dostat. Proto je důležité používat ty správné operace. Pokud použijeme pro sčítání znak `+`, skončí to nakonec jako sčítání čísel. Pokud bychom chtěli něco jiného, a to spojit dohromady řetězce, museli bychom použít znak tečku:

```
<html>
<head>
<title>Příklad 5. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $vysledek = 5 + "3 jablka";
    echo $vysledek, "<br>";
    $vysledek = 5 . "3 jablka";
    echo $vysledek, "<br>";
?>
</body>
</html>
```

Výše uvedený příklad právě demonstruje rozdíl mezi sčítáním čísel pomocí znaku + a spojováním řetězců pomocí znaku tečka. Pokud se sčítá pomocí znaku +, PHP všechno převádí na čísla a i výsledek vyjde jako číslo. Pokud se používá znak tečka, pak PHP všechno převádí na řetězce a i výsledek vyjde jako řetězec.

Typ boolean

Zvláštním typem proměnné je typ boolean, kterému se často říká logická proměnná. V běžném životě si můžete představit proměnnou typu číslo, nebo textový řetězec, protože s nimi běžně pracujeme mimo počítač. Proměnná typu boolean je naproti tomu ryze počítačová, a proto někdy činí její pochopení trochu problém. V zásadě typ boolean má jenom dvě hodnoty: **TRUE** (= pravda) a **FALSE** (= lež).

Logická proměnná typu boolean umí vyhodnocovat podmínky a umí ukládat výsledek. Umí vyhodnotit všechno, co se dá napsat jako podmínka do konstruktu if, a podobně. Třeba v následujícím příkladě použijeme proměnnou typu boolean ke zjištění, zda je číslo 2 větší, než číslo 1:

```
<html>
<head>
<title>Příklad 6. z 12. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $bool = 2 > 1;
    if ($bool == TRUE)
        echo "TRUE";
    else
        echo "FALSE";
?>
</body>
</html>
```

Po vyzkoušení příkladu bychom měli dostat na výpisu hodnotu TRUE, tedy, že je pravda, že dvojka je větší, než jednička. V takto jednoduchém pojetí nejsou proměnné typu boolean příliš užitečné, ale používá se to ke zjištění mnoha jiných věcí. Například i funkce **isset**, která zjišťuje, zda proměnná existuje, a o které jsem se zmínil na začátku tohoto dílu vlastně vrací hodnotu typu boolean. Vrací TRUE, když je pravda, že proměnná existuje a FALSE, když proměnná neexistuje. Vlastně PHP obsahuje spoustu funkcí, které vrací typ boolean.

Další typy proměnných

Kromě zde uvedených existují ještě další typy proměnných, které jsem tady nezmínil. Je to třeba typ **resource**, se kterým jsme se setkali v kapitole o souborech. Typ resource PHP používá k ukládání různých prostředků, jako je handle souboru. Tedy prostředků, které souvisí s externími věcmi.

Protože PHP je objektový jazyk, má i typ **object** reprezentující objekty. Podrobněji o objektech budu psát později.

PHP - 13. díl – kouzla s funkcí header

V dnešním dílu seriálu o PHP bude pojednávat detailně o funkci header a možnostech s tím spojených.

Funkce header - úvod

K pochopení funkce header je potřeba trochu osvětlit, jak pracují hlavičky a HTTP protokol. Jednoduše se dá napsat, že při odesílání webové stránky směrem k uživateli vše probíhá ve dvou fázích. Nejdříve se odešlou hlavičky HTTP protokolu, kterým se vlastně stránky přenáší. A pak se odešle samotný obsah stránky, tedy nejčastěji HTML kód stránky.

V hlavičkách HTTP protokolu se přenášejí nejrůznější údaje, a pomocí PHP funkce header můžete přidávat svoje vlastní údaje a ovlivňovat tak spoustu věcí. Důležité ovšem je, abychom funkci leader použili dříve, než vypíšeme jakýkoli výstup týkající se vlastní webové stránky. Jakmile totiž cokoli vypíšeme třeba pomocí echo, nelze už žádnou hlavičku HTTP protokolu odeslat a použití funkce header skončí chybou.

Samotné hlavičky HTTP protokolu nejsou nic jiného, než textové řetězce. Proto vhodně zvolenými textovými řetězci můžeme ovlivňovat průběh posílání webové stránky.

Přesměrování stránky

Jednu z věcí, kterou lze pomocí hlaviček HTTP protokolu zařídit, je přesměrování stránky. Jedná se o to, že prostě prohlížeči řekneme, aby natáhnul jinou stránku, než tu, kterou právě načítá. Třeba následující příklad, pokud si jej zkusíte spustit skončí natáhnutím úvodní stránky Computer pressu:

```
<?php
    header(`Location: http://www.cpress.cz/`);
?>
```

Důležité je, aby začátek skriptu, tj. <?php byl skutečně na prvním řádku, a nebyla předtím žádná mezera. Jinak by to nefungovalo.

Můžete svojí stránku přesměrovat kamkoli, záleží jenom na tom, jakou adresu uvedete za dvojtečkou po slově Location. Můžete samozřejmě použít celý svůj arzenál PHP znalostí a

kombinovat funkci header s jakýmkoli dalším PHP kódem. Například následující kód vás dopoledne přesměruje na Vltava.cz a odpoledne na stránky Cpress.cz:

```
<?php
if (date('A') == 'AM')
    header('Location: http://www.vltava.cz/');
else
    header('Location: http://www.cpress.cz/');
?>
```

Jak vidíte, síla přesměrování pomocí funkce header je v tom, že můžete přesměřovat kamkoli, a dokonce se v průběhu zpracování PHP skriptu rozhodnout, kam bude přesměrováno. Po přesměrování už nemá smysl cokoli vypisovat ve vašem vlastním skriptu, protože by to stejně nemělo být zobrazeno. Proto se často používá příkaz **exit** jako pojistka proti dalšímu vykonávání skriptu. Příkaz exit okamžitě ukončuje provádění PHP skriptu, takže se dále nepokračuje. Proto první příklad i s příkazem exit by vypadal takto:

```
<?php
header('Location: http://www.cpress.cz/');
exit;
?>
```

Funkce headers_sent

Protože hlavičky HTTP protokolu musí být poslány dříve, než jakýkoli výstup ve skriptu, může PHP odesílat hlavičky jen do té doby, dokud neprovedeme nějaký výstup. Někdy ve složitějších PHP skriptech je těžké poznat, zda již nějaký výstup byl, nebo ne. A v tomto přichází na pomoc funkce **headers_sent**, která jednoduše informuje, jestli už byly hlavičky HTTP protokolu poslány. Pokud byly hlavičky poslány, znamená to, že už nějaký výstup, třeba pomocí příkazu echo, nebo jinak, byl proveden. A pak už není možné žádné hlavičky poslat.

Jak to dopadne, pokud nějaký výstup už byl poslán si snadno můžeme vyzkoušet na příkladu:

```
<?php
echo `ahoj`;
header('Location: http://www.cpress.cz/');
exit;
?>
```

Pokud zkusíme tento příklad, PHP skript vypíše text "ahoj", a pak chybové hlášení v anglickém jazyce. Tím nám sděluje, že hlavičky už byly poslány, protože jsme už cosi vypsalí na výstup. A že nemůže naši hlavičku přidat k seznamu hlaviček, prostě funkce header totálně selže.

Pokud bychom se chtěli chybové zprávy vyvarovat, můžeme použít právě funkce headers_sent ke zjištění, jestli poslat hlavičku můžeme, a nebo už to nejde:

```
<?php
echo `ahoj<br>`;
if (headers_sent())
    echo `nejde odeslat hlavičku`;
else
    header('Location: http://www.cpress.cz/');
```

```
exit;
?>
```

Pokud si výše uvedený příklad spustíme, žádné chybové hlášení už nevznikne, ale pouze se nám vypíše "nejde odeslat hlavičku". Všimněte si, mimochodem, prázdných kulatých závorek za headers_sent v příkladu. Žádná funkce, i když nemá žádné parametry, nesmí být ošizena o kulaté závorky.

Pokud si předchozí příklad zmodifikujeme tak, aby nevypisoval "ahoj", není žádný problém hlavičku odeslat a přesměrovat tak na www.zive.cz:

```
<?php
if (headers_sent())
    echo `nejde odeslat hlavičku`;
else
    header('Location: http://www.cpress.cz/');
exit;
?>
```

Vypnutí cacheování stránky

Zkusíme teď další figl. Každý prohlížeč, nebo proxy cache, se snaží pamatovat si poslední stránky. To je na jednu stranu příjemná věc, protože šetříme tak linku k internetu. Na druhé straně nám tak může prohlížeč, nebo proxy cache tvrdošíjně nabízet staré stránky, které už mezitím byly aktualizovány, ale prohlížeč si to třeba nemyslí.

Proto existují i hlavičky, které určují, jak dlouho bude stránka ještě platná, případně i hlavičky, které pamatování si stránek přímo zakazují. Pokud vám tedy někdy prohlížeč nechce tvrdošíjně načíst nový obsah stránek, i když jste svoje stránky změnily, zkuste použít tvrdý kalibr na začátku své webové stránky:

```
<?php
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');
header('Last-Modified: `gmdate('D, d M Y H:i:s`)` GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');

// Tady pokračuje váš PHP skript.
?>
```

Výše zmíněný příklad obsahuje čtyři hlavičky, které donutí prakticky každý prohlížeč, nebo proxy cache přestat si stránku pamatovat, a vždy znovu a znovu natahovat stránku čerstvou. Někdy vám to může velmi pomoci. Dobré bude, když si zmíněnou kombinaci hlaviček dáte na každou stránku, která se často mění. Může to být třeba stránka novinek, nebo podobně.

Dnes jsem popsal některé možnosti odesílání hlaviček HTTP protokolu a příště téma dokončím dílem o zasílání a přijímání cookies, které k hlavičkám také patří.

PHP - 14. díl – cookies

Ve 14. díle seriálu o PHP bude pojednávat detailně o cookies. Na závěr si uděláme počítadlo, které bude měřit počty návštěv každému uživateli zvlášť.

Co jsou to cookies?

Anglický termín "cookies" má relativně hodně významů, ale pro naše účely se nejčastěji překládá jako sušenka. V podstatě je to pár bajtů dat, který si webový server uloží v prohlížeči přímo u uživatele. Pokud uživatel znovu přistoupí na stejnou stránku, webový server si "sušenku" znovu přečte. K čemu je to dobré?

Pomocí cookies můžete zjišťovat informaci o každém uživateli zvlášť, a tento komfort vám nic jiného, než cookies neposkytne. Můžete je využít ke zpříjemnění prostředí pro uživatele samotného. Můžete si třeba uložit, které diskusní příspěvky už uživatel viděl, a nabídnout mu ty nepřečtené. Můžete si v cookies pamatovat jeho nastavení. U internetového obchodu si můžete pamatovat, které položky má návštěvník ve svém košíku. Prostě můžete uložit cokoli, co se týká konkrétního uživatele.

Každý z vás má pravděpodobně ve svém prohlížeči uloženy desítky, možná stovky cookies. Většina lidí to ani neví, nebo nevnímá. Mnohé webové servery jsou dokonce na cookies tak závislé, že bez nich nefungují, například mnohé internetové obchody.

Cookies - technický úvod

Jak už jsem psal výše, cookies je pár bajtů dat, v podstatě krátký text, který zasíláte prohlížeči. Prohlížeč u sebe cookies uloží, a při příští návštěvě od stejného uživatele je znovu předá. Obsahem cookies může být cokoli, pouze celková velikost cookies je omezena na 4 KB.

Pokud vysíláte cookies na uživatelův prohlížeč, má každá z nich tyto části:

- ***Jméno*** - každá cookies se musí nějak jmenovat. Můžete vyslat více cookies, ale každá musí mít jiné jméno.
- ***Data*** - data, která jsou uložena do cookies, a kvůli kterým vlastně cookies vůbec posíláme.
- ***Doba expirace*** - každá cookies se po určité době sama smaže. Doba expirace představuje datum a čas, kdy ji prohlížeč automaticky odstraní. Po tomto datumu a času jsou data již stará a nepotřebná. Pokud se nezadá žádné datum expirace, cookies se automaticky smaže už při zavření stránky.
- ***Doména*** - cookies platí jen pro určitou doménu, například pro zive.cz. Jiná doména nemůže cookies přijmout. Části doména a cesta jsou uvedeny z bezpečnostních důvodů a určují, kdo smí vyslané cookie přijmout.
- ***Cesta*** - cookies smí přijmout jen stránky s určitou virtuální cestou. Pokud se neuvede žádná cesta, jsou cookies přístupné jen webovým stránkám ze stejného adresáře, jako je stránka, která cookies vyslala.
- ***Secure*** - určuje, zda cookies se smí přijímat pouze pomocí zabezpečeného SSL kanálu, nebo zda může přijít i normálním nezabezpečeným kanálem.

Tyto části jsou důležité hlavně proto, že při vysílání cookies se tyto jednotlivé části nastavují.

Vysílání a příjem cookies v PHP

Pro vysílání cookies pomocí PHP slouží funkce **setcookie**. Předem je potřeba říci, že tato funkce zapisuje do hlaviček HTTP protokolu (viz minulý díl seriálu o PHP). Důsledkem je hlavně to, že funkce setcookie se musí použít dříve, nežli cokoli vypíšeme na výstup, jinak funkce setcookie selže a nic nevyšle.

Funkce setcookie má volitelně jeden až šest parametrů, kterými se předávají jednotlivé části cookie. Mohou se zprava vynechávat:

```
setcookie(jméno, data, doba expirace, cesta, doména, secure);
```

Pro příjem cookies je určeno pole proměnných s názvem \$HTTP_COOKIE_VARS, které obsahuje všechna cookies, které webová stránka přijala.

Pro demonstraci vysílání a příjmu cookies v PHP si můžeme uvést velmi jednoduchý příklad:

```
<?php
    setcookie('test', 'pokusná data', time()+3600);
?>
<html>
<head>
<title>Příklad 1. z 14. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    if (isset($HTTP_COOKIE_VARS['test']))
        echo $HTTP_COOKIE_VARS['test'];
    else
        echo 'cookie nepřijmuta';
?>
</body>
</html>
```

Příklad začíná PHP částí s funkcí setcookie. Protože funkce setcookie musí být uvedena dříve, než přijde jakýkoli výstup, je tato funkce úplně na začátku příkladu. V našem příkladě funkce setcookie má 3 parametry. Prvním parametrem je název cookie, v našem příkladě se jmenuje test. Druhý parametr jsou samotná data, zvolil jsem řetězec "pokusný text". Třetí, poslední parametr obsahuje dobu expirace. Tu jsem odvodil od funkce time(), která vrací aktuální datum a čas a přičetl jsem k ní 3600 sekund, tedy jednu hodinu. Doba expirace se tedy dá přeložit jako od tohoto okamžiku za hodinu vyprší platnost cookie.

Dále v příkladu pokračuje začátek HTML kódu. Pak nastává další sekce PHP kódu, kde testuji proměnnou \$HTTP_COOKIE_VARS['test']. Tedy testuji, zda jsem přijal cookie s názvem test.

Pokud si příklad vyzkoušíte poprvé, bude vyslána cookie s názvem test a uložena ve vašem prohlížeči. Protože cookie může být přijmuta až při dalším načtení stránky, při prvním spuštění dostanete výpis textu "cookie nepřijmuta". Při dalším spuštění ovšem začíná vypisovat data, tedy náš text "pokusný text". Pokud byste počkali hodinu, což je doba, za kterou naše cookie vyprší, opět zahlásí "cookie nepřijmuta".

Počítadlo návštěv uživatele

Pomocí cookies se dá napsat velmi jednoduché počítadlo, které každému uživateli sdělí, kolikrát navštívil naší webovou stránku:

```
<?php
    if (isset($HTTP_COOKIE_VARS['navsteva']))
        $navsteva = $HTTP_COOKIE_VARS['navsteva'];
    else
        $navsteva = 0;

    ++$navsteva;

    setcookie('navsteva', $navsteva, time()+1000000);
?>
<html>
<head>
<title>Příklad 2. z 14. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    echo `Tuto stránku jsi navštívil celkem `, $navsteva, ` krát.`;
?>
</body>
</html>
```

Pokud si příklad spustíte, a nejlépe opakovaně, zjistíte, že při každém dalším spuštění vám zvýší počet návštěv na stránce. Počty návštěv poctivě ukládá každému uživateli zvlášť do cookies do jeho prohlížeče.

K ukládání počtu návštěv je použito cookie s názvem navsteva. Zbytek PHP kódu byste už měli bezpečně přechíst, protože se v něm nepoužívá nic nového. Doba expirace cookie je nastavena na milión sekund, což je asi 11 dní. Později bude uvedena metoda, jak vymyslet rozumněji zapsanou dobu expirace. Jako data cookie se ukládá právě proměnná \$navsteva s počtem návštěv.

Jak uvádět dobu expirace?

Pro uvedení doby expirace můžeme použít dvě funkce, a to buď funkci **time**, nebo **mktime**. Funkce time nám dobře poslouží k tomu, abychom stanovili za kolik sekund od této chvíle má cookie vypršet. A funkce mktime je zase vhodná, když chceme přímo zadat datum a čas, kdy má přesně dojít k vypršení cookie.

Funkci time jsme už používali, prostě uvedením výrazu

```
time() + počet_sekund
```

řekneme, za kolik sekund má cookie vypršet.

Funkce mktime má v ideálním případě 6 parametrů, které znamenají jednotlivé fragmenty data a času. Podivné pořadí v parametrech berme jako chyták PHP:

```
mktime(hodina, minuta, sekunda, měsíc, den, rok)
```

Takže teď můžu poopravit počítadlo tak, že uvedu dobu expirace na 31. prosince 2030, což je dostatečně dlouhá doba:

```
<?php
    if (isset($HTTP_COOKIE_VARS['navsteva']))
        $navsteva = $HTTP_COOKIE_VARS['navsteva'];
    else
        $navsteva = 0;

    ++$navsteva;

    setcookie('navsteva', $navsteva, mktime(0,0,0,12,31,2030));
?>
<html>
<head>
<title>Příklad 3. z 14. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    echo `Tuto stránku jsi navštívil celkem `, $navsteva, ` krát.`;
?>
</body>
</html>
```

PHP - 15. díl – pole proměnných

O polích proměnných jsem psal už v minulých dílech, ale tento díl se polím bude věnovat více do hloubky.

Pole proměnných - úvod

Pole proměnných jsou velmi efektivní a mocná zbraň PHP. Pole v PHP jsou optimalizovaná v několika směrech, takže jsou velmi univerzální.

Pole proměnných je v podstatě proměnná, která má mnoho hodnot, které se liší indexem. Hned na úvod je tu jednoduchý příklad, jak pomocí pole přeložit anglické názvy dní v týdnu do češtiny:

```
<html>
<head>
<title>Příklad 1. z 15. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    $den['Mon`] = `pondělí`;
    $den['Tue`] = `úterý`;
    $den['Wed`] = `středa`;
    $den['Thu`] = `čtvrtek`;
    $den['Fri`] = `pátek`;
    $den['Sat`] = `sobota`;
    $den['Sun`] = `neděle`;
```

```
echo `Dnes je `, $den[date("D")];

?>
</body>
</html>
```

Pokud si příklad spustíte, zjistíte, že vypíše, jaký je právě den v týdne. Vypíše to česky, a k překladu do češtiny používá pole.

Zjistit, jaký je právě den v týdnu je možné pomocí výrazu `date("D")`. Funkce `date` v tomhle případě vrátí trojpísmennou zkratku dne v týdnu, ale bohužel v angličtině. Pokud se podíváte do příkladu, tak zjistíte, že v něm je pole `$den`, které používá jako indexy právě anglické zkratky dní v týdnu. A hodnoty jsou české názvy dní v týdnu. Pak mi pro překlad stačí použít výraz

```
$den[date("D")]
```

Výraz `date("D")` dodá anglickou třípísmennou zkratku dne v měsíci. Ta se použije jako index do pole `$den` a vyjde z toho český název dne v týdnu.

Další příklad

V dalším příkladu zkusíme přeložit celé datum do českého jazyka:

```
<html>
<head>
<title>Příklad 2. z 15. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    $mesic[1] = `leden`;
    $mesic[2] = `únor`;
    $mesic[3] = `březen`;
    $mesic[4] = `duben`;
    $mesic[5] = `květen`;
    $mesic[6] = `červen`;
    $mesic[7] = `červenec`;
    $mesic[8] = `srpen`;
    $mesic[9] = `září`;
    $mesic[10] = `říjen`;
    $mesic[11] = `listopad`;
    $mesic[12] = `prosinec`;

    echo `Dnes je `,date(`j`),`. `, $mesic[date(`n`)],` `,date(`Y`);
?>
</body>
</html>
```

Pro zobrazení celého data použijeme tři výrazy: `date('j')` vrátí číslo dne v měsíci, `date('n')` vrátí číslo měsíce (to poté přeložíme na název měsíce) a `date('Y')`, který poslouží ke zjištění, jaký rok právě je.

Pro české datum potřebujeme název měsíce v češtině. K tomu si připravíme pole `$mesic`, které bude mít indexy 1 až 12, hodnotou budou české názvy měsíců.

Jazykový konstrukt array

PHP má speciální jazykový konstrukt **array**, který slouží pro definování polí. Slouží jako zkratka pro definování polí, je přehlednější a úspornější, než způsob, který je použit třeba v předchozích příkladech. Nejnázornější bude příklad, jak může vypadat první příklad, pokud použijeme konstrukt array:

```
<html>
<head>
<title>Příklad 3. z 15. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    $den = array
    (
        `Mon` => `pondělí`,
        `Tue` => `úterý`,
        `Wed` => `středa`,
        `Thu` => `čtvrtek`,
        `Fri` => `pátek`,
        `Sat` => `sobota`,
        `Sun` => `neděle`
    );

    echo `Dnes je `, $den[date("D")];

?>
</body>
</html>
```

Příklad je úplně stejný jako první, pouze je pro definici pole použit konstrukt array. Ten umožňuje definovat celé pole naráz pomocí syntaxe `index => hodnota`, kde všechny prvky pole oddělujeme čárkou. Jak už jsme se přesvědčili dříve, jako indexy je možné použít buď čísla, nebo textové řetězce.

Druhý příklad je zase možné pomocí konstruktů array přepsat takto:

```
<html>
<head>
<title>Příklad 4. z 15. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    $mesic = array
    (
        1 => `leden`,
        2 => `únor`,
        3 => `březen`,
        4 => `duben`,
        5 => `květen`,
        6 => `červen`,
        7 => `červenec`,
        8 => `srpen`,
        9 => `září`,
        10 => `říjen`,
        11 => `listopad`,
    );
?>
</body>
</html>
```

```

    12 => `prosinec`
);

echo `Dnes je `,date(`j`),`. `, $mesic[date(`n`)],` `,date(`Y`);
?>
</body>
</html>

```

Pokud používáme jako indexy vzestupnou řadu čísel, je možné u konstruktů array vynechat indexy i značku => úplně. Pokud totiž v konstruktě array index chybí, automaticky se dosadí o jedničku více, než největší dosud použité číslo. Proto je poslední možný příklad možné zapsat ještě jednodušeji:

```

<html>
<head>
<title>Příklad 5. z 15. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?
    $mesic = array
    (
        1 => `leden`,
        `únor`,
        `březen`,
        `duben`,
        `květen`,
        `červen`,
        `červenec`,
        `srpen`,
        `září`,
        `říjen`,
        `listopad`,
        `prosinec`
    );

    echo `Dnes je `,date(`j`),`. `, $mesic[date(`n`)],` `,date(`Y`);
?>
</body>
</html>

```

Předdefinovaná pole

PHP používá pole velmi často i k tomu, aby vám předával různé hodnoty, informoval vás o různých stavech, apod.. Velmi důležité jsou různá předdefinovaná pole. S některými z nich jsem vás seznámil už v předchozích dílech seriálu. Jsou to například pole `$_SERVER`, které slouží pro předávání informací z webového serveru. Pro práci s formuláři slouží pole `$_GET` a `$_POST` a mnohé další.

O předdefinovaných polích se zmiňuji hlavně proto, abyste si byli vědomi, že pole je velmi důležitá součást PHP a objevuje se v PHP na každém rohu.

PHP - 16. díl – začínáme s regulárními výrazy

Regulární výrazy představují velmi silný nástroj pro práci s textem, který je velmi známý z unixových systémů.

Znalosti regulárních výrazů můžete využít mnohem širěji, než pouze v PHP. Regulární výrazy pronikly snad do všech oblastí zpracování textů.

Regulární výrazy jsou tedy velmi silným a univerzálním nástrojem pro zpracování textů, pomocí kterého můžete:

- vytahovat z textů údaje, které vás zajímají
- velmi efektivně vyhledávat v textech
- nahrazovat v textech vzájemně různé údaje

Tester regulárních výrazů

Protože bude potřeba regulární výrazy testovat, vytvoříme si alespoň malý tester pro regulární výrazy. Jeho funkci teď nebudu vysvětlovat, potřebné znalosti k pochopení funkce testeru získáte v průběhu dalšího čtení.

```

<html>
<head>
<title>Příklad 1. z 16. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<h1>Tester regulárních výrazů</h1>
<?php
if (!isset($reg_vyraz))
    $reg_vyraz = ``;
if (!isset($text))
    $text = ``;
?>
<form action="" method="post">
<b>Regulární výraz:</b>
<input type="text" name="reg_vyraz" value="<?php echo $reg_vyraz; ?>">
<br>
<b>Text:</b>
<input type="text" name="text" value="<?php echo $text; ?>">
<br>
<input type="submit" value="Porovnat text vůči regulárnímu výrazu">
<br>
</form>
<?php
if (@ereg($reg_vyraz, $text, $pole_vysledku))
    echo `Regulárnímu výrazu vyhovuje: `, $pole_vysledku[0];
else
    echo `Text nevyhovuje regulárnímu výrazu`;
?>
</body>
</html>

```

Tento tester můžete použít k ověřování regulárních výrazů v tomto díle seriálu.

První regulární výrazy

Nejjednodušším regulárním výrazem je jedno písmeno. Když například budete jako mít regulární výraz písmeno h, pak při hledání v textu to najde právě písmeno h. Zkuste si třeba spustit výše uvedený tester regulárních výrazů, do kolonky regulární výraz zadejte písmeno h a do kolonky text zadejte nějaký text. Pokud text bude obsahovat alespoň jedno písmeno h, třeba slovo ahoj, pak se po kliknutí na tlačítko objeví věta: "Regulárnímu výrazu vyhovuje: h". Pokud text nebude obsahovat písmeno h, pak se po kliknutí objeví věta: "Text nevyhovuje regulárnímu výrazu".

Najít písmeno h, nebo jiné písmeno v textu je ale tak jednoduchá úloha, že zkusíme postoupit kousek dál. Můžete také hledat celé slovo. Pokud například jako regulární výraz zadáte více písmen, třeba slovo ahoj, bude se hledat v textu celé slovo ahoj. Vyhledávání slov je vlastně velmi jednoduchou, ale zároveň často používanou aplikací regulárních výrazů.

Funkce ereg - testování regulárních výrazů

Pro testování regulárních výrazů slouží v PHP funkce **ereg**. Funkce má 2, nebo 3 parametry, teď ukážeme verzi se dvěma parametry:

```
ereg (regulární výraz, text)
```

Jako první parametr se dává regulární výraz, jako druhý parametr testovaný text. Funkce vrací hodnotu, která říká, jestli text vyhovuje regulárnímu výrazu. Příklad použití funkce ereg ukáží na příkladě, který otestuje, zda je správně zadána e-mailová adresa. Kontrolu e-mailové adresy uděláme jako formulář (viz 10. díl seriálu), tedy roztažený do dvou souborů. První soubor uložíte s názvem zadani.php. Nebudete to vlastně ani PHP skript, jen čistý HTML soubor sloužící pro zadání e-mailové adresy:

```
<html>
<head>
<title>Příklad 2. z 16. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<form action="kontrola.php" method="post">
<b>Zadej e-mailovou adresu:</b>
<input type="text" name="mail" value="">
<br>
<input type="submit" value="Otestuj e-mailovou adresu">
<br>
</form>
</body>
</html>
```

Druhý soubor uložíte s názvem kontrola.php:

```
<html>
<head>
<title>Příklad 3. z 16. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$mail = $_POST['mail'];
if (ereg('~@', $mail))
    echo `E-mailová adresa je správně`;
```

```
else
    echo `E-mailová adresa není správně`;
?>
<br><br>
<a href="zadani.php">Zkontrolovat další mail</a>
</body>
</html>
```

Příklad si spustíte tak, že nainstalujete zadani.php a zadáte nějaký mail, ať už správný, nebo chybný. Po kliknutí na tlačítko "Otestuj e-mailovou adresu" zjistíte, jestli je e-mailová adresa správně.

A jak to pracuje? Velmi jednoduše. V souboru zadani.php je pouze zadání e-mailové adresy, která se po kliknutí na tlačítko přenese do PHP skriptu kontrola.php. Tam se vyzvedne a zkontroluje funkcí ereg. Jako regulární výraz je použit zavináč '@', takže se vlastně kontroluje, zda je v textu znak zavináč. Pokud tam je, e-mailová adresa se považuje za správnou. Pokud tam není, považuje se e-mail za nesprávný.

Takže zde vidíte využití i velmi jednoduchého regulární výrazu. Je jasné, že kontrola správnosti e-mailové adresy jen podle přítomnosti znaku zavináč není ještě to pravé ořechové. Ale postupně kontrolu e-mailové adresy zpřesníme.

Složitější regulární výrazy - trocha teorie

Pro hledání v textu často potřebujeme složitější věci, než jenom vyhledání písmena, nebo slova. Nažhavte proto znovu tester regulárních výrazů z prvního příkladu a půjdeme dále.

V regulárních výrazech slouží tečka k hledání libovolného znaku. Je to speciální znak, například tento regulární výraz:

```
.ost
```

najde slovo kost, post, most, dost a další slova, je prostě jedno, co je na prvním místě za písmeno, nebo za znak. Vyzkoušejte si to v testeru regulárních výrazů.

Někdy ale chceme, aby to nebylo až tak jedno. Pak můžeme použít hranaté závorky [] a vyjmenovat do nich znaky, které chceme dovolit. Například tento regulární výraz najde pouze slova most a kost:

```
[mk]ost
```

Výše uvedený regulární výraz je potřeba číst tak, že najde čtyřpísmenné slovo, kde na prvním místě je buď písmeno m, nebo k a zbylá tři písmena jsou ost.

V hranatých závorkách [] se ale můžeme rozšoupnout trochu více. Pokud třeba chceme najít dvoumístnou číslici, můžeme to pomocí regulárního výrazu napsat takto:

```
[123456789][01234567890]
```

Tím říkám, že hledám dva znaky, přičemž první je číslice z rozsahu 1 až 9, druhý znak je číslice z rozsahu 0 až 9. Tak najdu dvoumístnou číslici. Tento regulární výraz je možné napsat i kratším způsobem, a to použitím pomlčky:

[1-9] [0-9]

Pomlčka slouží k uvedení rozsahu znaků, v mém případě na prvním místě znak z rozsahu 1 až 9, na druhém místě hledám znak z rozsahu 0 až 9.

Pokud za levou hranatou závorkou [použiji znak ^, pak se význam obrací. Hledám znaky, které nejsou uvedeny v hranatých závorkách. Například následujícím regulárním výrazem hledám znak, který není číslicí 0 až 9:

[^0-9]

Jako složitější příklad si v testeru regulárních výrazů můžete zkusit vyhledat datum třeba ve formátu 31/12/2003, nebo 01-04-2003. Vyhledání takového data lze podle jednoduchého regulárního výrazu (předpokládám, že dni i měsíce mají pokaždé dvě číslice):

[0-3] [0-9] . [01] [0-9] . [12] [0-9] [0-9] [0-9]

Tento díl je úvodem do regulárních výrazů, příští díl seriálu bude v regulárních výrazech pokračovat.

PHP - 17. díl – jedeme dále s regulárními výrazy

Dnešní část seriálu o PHP budeme opět věnovat regulárním výrazům. Článek velmi těsně navazuje na předchozí díl.

Opakování v regulárních výrazech - trocha teorie

Pro následující povídání si připravte tester regulárních výrazů z [minulého dílu](#). Bude dobré, když si na něm vyzkoušíte následující řádky.

Silným prvkem v regulárních výrazech je opakování. Základním výrazem pro opakování je hvězdička *. Hvězdička říká, že bezprostředně předcházející část regulárního výrazu se může libovolně opakovat. Libovolně znamená, že se může opakovat tolikrát, kolikrát to jde. Jako příklad lze třeba uvést regulární výraz:

[0-9]*

Tento výše uvedený výraz znamená, že číslice 0 až 9 se může opakovat, vyhoví jí tedy jakkoli dlouhá číslice. Jiným příkladem je třeba

A*hoj

Tento výraz vyhoví slovu Ahoj, ale také AAhoj, nebo AAAhoj, prostě libovolnému opakování písmenu A následované hoj. Protože hvězdička * znamená jakýkoli počet opakování, znamená to i opakování v počtu nula. Proto vyhoví i samotné slovo hoj.

Regulární výrazy jsou hladové, snaží se tedy, aby opakování požralo co nejvíce znaků.

Následující výraz znamená prostě libovolný řetězec znaků a v tomto významu se často využívá:

.*

Jak jistě víte z minulého dílu, tečka znamená libovolný znak. A tečka s hvězdičkou znamená libovolný znak, který se může opakovat, výsledkem je tedy obecný řetězec znaků.

Někdy se vám může hodit vyhledávání pomocí opakovacího znaku plus +. Má naprosto stejný význam jako hvězdička *, ale na rozdíl od ní plus + říká, že předchozí část se musí opakovat alespoň jednou. Tedy následujícímu regulárnímu výrazu vyhoví slovo Ahoj, stejně tak jako AAhoj, ale už nevyhoví hoj:

A+hoj

Posledním opakovacím znakem, který tu vysvětlím, je znak otazník ?. Znamená prostě jen to, že předchozí část je volitelná. Buď tam je jednou, a nebo tam není vůbec. Například následující regulární výraz vyhoví dvěma slovům, a to mód a móda (poslední a na konci je volitelné díky otazníku):

móda?

Základní pozicování v regulárních výrazech - opět trocha teorie

Občas potřebujeme uvnitř regulárních výrazů pevný bod, protože do této chvíle se prostě hledal nějaký kus textu kdekoli. Základní pozicování je stříška ^, která označuje začátek řádku a dolar \$, který označuje konec řádku. Pokud máme jednořádkový text, pak stříška ^ znamená začátek textu a dolar \$ znamená konec textu.

Například následující regulární výraz najde řádky, které začínají na a:

^a

Nebo pro nalezení řádků končících na slovo test použijeme:

test\$

Závorky v regulárních výrazech - poslední ždibek teorie

Uvnitř regulárních výrazů je možné použít i závorky. Závorky můžete použít k tomu samému, co v matematice, a to k seskupování. Například když chci povolit opakování většího celku. Například následující regulární výraz znamená libovolné opakování slova ahoj:

(ahoj)*

Test správnosti e-mailové adresy - příklad

V minulém díle jsem uvedl test správnosti e-mailové adresy, který byl ale velmi jednoduchý. Testoval pouze, zda adresa obsahuje znak zavináč @. Takový test je ale příliš jednoduchý a propustí i mnoho nesprávných e-mailových adres. Proto teď zkusíme formulovat složitější regulární výraz, který by otestoval e-mailovou adresu s větší spolehlivostí.

Zkusíme pro kontrolu e-mailové adresy použít regulární výraz:

```
@[^@]+[.][a-zA-Z]+$
```

V tomto regulárním výrazu kontrolujeme, zda e-mailová adresa obsahuje znak zavináč `@`, který je následován alespoň jedním znakem, který není zavináč a končí tečkou a jedním, nebo více písmeny. Znak dolar na konci označuje konec řádky, tedy v našem případě konec e-mailové adresy.

Nyní zkusíme poopravit příklad z minulého dílu na tento regulární výraz. Soubor `zadani.php`, kterým zadáváme e-mailovou adresu zůstane beze změny:

```
<html>
<head>
<title>Příklad 1. z 17. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<form action="kontrola.php" method="post">
<b>Zadej e-mailovou adresu:</b>
<input type="text" name="mail" value="">
<br>
<input type="submit" value="Otestuj e-mailovou adresu">
<br>
</form>
</body>
</html>
```

Druhý soubor s názvem `kontrola.php` dostane tento tvar:

```
<html>
<head>
<title>Příklad 2. z 17. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$mail = $_POST['mail'];
if (ereg('~@[^@]+[.][a-zA-Z]+$`', $mail))
    echo `E-mailová adresa je správně`;
else
    echo `E-mailová adresa není správně`;
?>
<br><br>
<a href="zadani.php">Zkontrolovat další mail</a>
</body>
</html>
```

Chcete-li si příklad vyzkoušet, spusťte první soubor `zadani.php`, kde zadáte pokusnou e-mailovou adresu a po kliknutí na tlačítko zjistíte, zda je e-mailová adresa zadána správně. Můžete si vyzkoušet, že nyní je adresa testována daleko dokonaleji.

Funkce `ereg` a `eregi`

Funkci **`ereg`** jsem představil už v minulém dílu. Funkce má dva, nebo tři parametry, a určí, jestli text vyhovuje danému regulárnímu výrazu. Funkci jsem použil například v testu na správnost e-mailové adresy. Pokud se použijí tři parametry jsou významy následující:

`ereg (regulární výraz, text, pole)`

kde do "pole" se uvede název proměnné. Funkce `ereg` pak vytvoří pole proměnných, kde do indexu nula uvede celý kus textu vyhovujícím regulárnímu výrazu. Zkuste se podívat v minulém dílu do testu regulárních výrazů, kde toto používám.

Pokud regulární výraz obsahuje i závorky, pak pole proměnných obsahuje v indexu jedna kus textu vyhovující obsahu první závorky, do pole dva kus textu vyhovující druhé závorce, atd..

Funkce **`eregi`** je naprosto shodná s funkcí `ereg`, pouze `eregi` nerozlišuje malá a velká písmena, zatímco `ereg` dělá mezi malými a velkými písmeny rozdíl.

Funkce `ereg_replace` a `eregi_replace`

Funkce **`ereg_replace`** slouží k náhradě textu podle regulárního výrazu. Má tři parametry

`ereg_replace (regulární_výraz, čím_nahradit, text)`

Funkce `ereg_replace` najde v textu všechna místa, kde se hodí regulární výraz, a nahradí jej "čím_nahradit". Viz třeba následující příklad, který v textu nahradí slovo `test` slovem `zkouška`:

```
<html>
<head>
<title>Příklad 3. z 17. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `Toto je jenom test a test`;
    echo ereg_replace(`test`, `zkouška`, $text);
?>
</body>
</html>
```

Příklad se dá modifikovat, aby nahradil třeba jen `test` na konci textu:

```
<html>
<head>
<title>Příklad 4. z 17. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `Toto je jenom test a test`;
    echo ereg_replace(`test$`, `zkouška`, $text);
?>
</body>
</html>
```

Funkce **`eregi_replace`** je naprosto shodná s `ereg_replace`, pouze v regulárním výrazu nerozlišuje malá a velká písmena.

V 16. a 17. díle jsem napsal o regulárních výrazech jen to nejdůležitější. Obvykle se o regulárních výrazech píše na větším prostoru. O některých možnostech regulárních výrazů jsem se vůbec nezminil, ale myslím, že můj přehled na základní seznámení stačí.

PHP - 18. díl – základní funkce pro práci s řetězci

V dnešním dílu se podrobněji podíváme na základní funkce, které využijete při zpracování textových řetězců.

Funkce strlen - délka řetězce

Základní funkce **strlen** vrací délku řetězce. Její použití je ve tvaru

```
strlen(řetězec)
```

a vrací číslo, které říká, kolik znaků obsahuje řetězec. Použití funkce strlen je možné ukázat na jednoduchém příkladu:

```
<html>
<head>
<title>Příklad 1. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $delka = strlen(`pokusný řetězec`);
    echo `Délka řetězce je `, $delka, ` znaků.`;
?>
</body>
</html>
```

Funkce strtolower, strtoupper

Funkce **strtolower** slouží k převodu řetězce na malá písmena. Naproti tomu funkce **strtoupper** slouží k převodu řetězce na velká písmena. Používají se ve tvaru

```
strtolower(řetězec)
strtoupper(řetězec)
```

Použití těchto dvou funkcí si můžete demonstrovat na jednoduchém příkladě:

```
<html>
<head>
<title>Příklad 2. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `PHP je fajn`;
    echo `Text malými písmeny: `, strtolower($text), ` <br>`;
    echo `Text velkými písmeny: `, strtoupper($text), ` <br>`;
?>
```

```
?>
</body>
</html>
```

Funkce substr

Funkce **substr** slouží k vrácení jenom části řetězce. Pokud potřebujeme z řetězce vybrat jenom část, a zbytek odseknout, potom je funkce substr to pravé. Funkce má tvar:

```
substr(řetězec, počáteční_pozice, délka)
```

Parametr počáteční pozice označuje první znak, který se objeví jako výsek řetězce. První znak má počáteční pozici nula, druhý znak má počáteční pozici jedna, atd. Parametr délka určuje, kolik znaků bude ve vyseknutém řetězci. Parametr délka může i chybět, potom se ve vyseknutém řetězci objeví znaky až do konce řetězce.

Zde je jednoduchý příklad:

```
<html>
<head>
<title>Příklad 3. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `abcdefgh`;
    echo substr($text,2,4), ` <br>`;
    echo substr($text,3), ` <br>`;
?>
</body>
</html>
```

V příkladu mám jednoduchý řetězec `abcdefgh`. Nejdříve použiji substr(\$text,2,4), což znamená, že chci výsek řetězce, která začíná třetím znakem (počáteční pozice je 2) a je dlouhý 4 znaky. Vypíše se tedy `cdef`. Potom volám substr(\$text,3), tedy chci výsek řetězce začínající čtvrtým znakem (počáteční pozice je 3), a protože mi chybí parametr délka, chci výsek až do konce řetězce. Vypíše se tedy `defgh`.

Funkce substr je daleko mazanější. Jako počáteční pozici vám umožňuje zadat i záporné číslo. Pak se počáteční pozice počítá od konce řetězce. Zadáte-li jako počáteční pozici číslo -1, pak výsek řetězce bude začínat posledním znakem od konce. Zadáte-li jako počáteční pozici -2, výsek řetězce začne druhým, tedy předposledním znakem od konce. Atd.. I to si můžeme zkusit na příkladu:

```
<html>
<head>
<title>Příklad 4. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `abcdefgh`;
    echo substr($text,-3,2), ` <br>`;
    echo substr($text,-1,1), ` <br>`;
?>
```

```
</body>
</html>
```

Ve výše uvedeném příkladu používám substr(\$text,-3,2), kde chci výsek řetězce začínající třetím znakem od konce (počáteční pozice je -3) a dlouhý dva znaky. Vypíše se tedy `fg`. A potom používám substr(\$text,-1,1), což znamená, že chci výsek začínající posledním znakem v řetězci (počáteční pozice je -1) a výsek bude dlouhý jeden znak. Vypíše se tedy `h`.

Funkce substr dokonce umožňuje zadat i zápornou délku. Pokud je zadána záporná délka, znamená to, kolik znaků před koncem má výsek řetězce skončit. Protože záporná délka se používá velmi velmi zřídka, nebudu uvádět příklad na zápornou délku.

Funkce strpos

Funkce **strpos** slouží pro hledání textu v nějakém řetězce. Funkce strpos potom vrátí počáteční pozici, na které se text v řetězci vyskytuje. Funkce má tvar:

```
strpos (řetězec, hledaný_text)
```

nebo

```
strpos (řetězec, hledaný_text, odkud_hledat)
```

Pro ilustraci jednoduchý příklad:

```
<html>
<head>
<title>Příklad 5. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $text = `Právě jsem na internetu.`;
    echo strpos($text, `internet`);
?>
</body>
</html>
```

Ve výše uvedeném příkladu mám řetězec `Právě jsem na internetu` a hledám v něm text `internet`. Pokud si příklad spustíte, vypíše se vám číslo 14. To znamená, že text `internet` byl nalezen na pozici číslo 14. Protože se pozice znaků číslují od nuly, znamená pozice 14, že byl nalezen na patnáctém znaku.

Pokud funkce text v řetězci nenajde, vrátí false. To je snadno zaměnitelné s případem, kdy text je nalezen hned na prvním znaku (tedy na pozici nula). Jak se má správně postupovat, aby se rozlišil případ, kdy text není nalezen, je uvedeno v následujícím příkladě:

```
<html>
<head>
<title>Příklad 6. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
```

```
    $text = `Právě jsem na internetu.`;
    if (strpos($text, `houba`) === false)
        echo `Text nenalezen`;
    else
        echo `Text nalezen`;
?>
</body>
</html>
```

Funkce strpos najde vždy první výskyt textu v řetězci. Pokud se daný text vyskytuje v řetězci vícekrát, funkce strpos najde vždy první výskyt.

Funkce strstr

Funkce **strstr** slouží k nalezení textu v řetězci a vrácení výseku začínající tímto textem. Samotná funkce má tvar:

```
strstr (řetězec, hledaný_text)
```

O tom, co funkce strstr přesně dělá si nejlépe ukážeme na příkladech:

```
<html>
<head>
<title>Příklad 7. z 18. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $email = `adam@hotmail.com`;
    echo strstr($email, `@`);
?>
</body>
</html>
```

Pokud si výše uvedený příklad spustíte, funkce strstr bude hledat v daném e-mailu znak zavináč. Protože ho najde, tak vrátí výsek e-mailu, který začíná zavináčem. Vypíše se tedy `@hotmail.com`.

PHP - 19. díl – odesílání mailů v PHP

V dnešním pokračování se více zaměříme na využití PHP k odesílání e-mailů z webových stránek.

Jak připravit PHP k odesílání mailů

Odesílání mailů pomocí PHP není nijak složité, je tu nicméně spousta ale. Pokud máte hosting, který povoluje mailování přes PHP, nemusíte dělat vůbec žádnou přípravu, a příklady z tohoto dílu vám půjdou okamžitě bez problémů. Tedy v případě, že příklady budete zkoušet na tomto hostingu.

Pokud si ovšem chcete zkoušet mailování doma třeba pomocí intranetového serveru ze 2. dílu tohoto seriálu, musíte si nejdříve PHP nastavit. V opačném případě pokusy o mailování skončí neúspěchem.

Nastavení PHP je velice jednoduché, v zásadě jde o to upravit jednu jedinou řádku v konfiguračním souboru php.ini. Proto nejdříve nejděte tento soubor, pokud máte nainstalovaný pokusný intranetový server ze 2. dílu, najdete jej v souboru C:\inet_srv\apache\bin\php.ini. Otevřete tento soubor v nějakém editoru, třeba v Poznámkovém bloku a najděte řádek, který vypadá takto:

```
SMTP = localhost
```

Namísto slova localhost musíte napsat adresu poštovního serveru, přes který odesíláte poštu (SMTP). Většinou adresa bývá ve tvaru smtp.něco.cz. Tuto adresu máte nastavenou i v Outlooku, nebo v jiném programu, přes který odesíláte svou poštu. Je potřeba napsat platnou adresu poštovního serveru, jinak vám odesílání mailů nepoběží.

Po této úpravě uložte změněné php.ini, a restartujte, tedy ukončíte a znovu spustíte webový server, aby se mohly změny souboru php.ini projevit. Nyní je vše připraveno k pokusům s odesíláním pošty.

Je samozřejmé, že při pokusech s odesíláním pošty pomocí PHP musíte být připojeni k internetu. PHP odesílání mailů okamžitě, a ve chvíli odesílání mailu musí být připojení k internetu.

Funkce mail

Pro odesílání mailů je v PHP připravena funkce s celkem didaktickým názvem **mail**. Může mít tři, nebo čtyři parametry, podle toho, jak moc si chceme s odesíláním mailů vyhrát. Funkce mail má tento tvar:

```
mail (adresa_příjemce, předmět_zprávy, text_zprávy,
dodatečné_hlavičky_mailu)
```

Poslední čtvrtý parametr může chybět. Vrhněme se tedy do prvního, jednoduchého odesílače mailů:

```
<html>
<head>
<title>Příklad 1. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $email = `vase.adresa@neco.cz`;
    $vysledek = mail($email, `Předmět mailu`, `Text mailu`);
    if ($vysledek)
        echo `Mail úspěšně odeslán`;
    else
        echo `Mail nebyl odeslán, nastala chyba`;
?>
</body>
</html>
```

Než budete zkoušet, přepište si prosím adresu vase.adresa@neco.cz na svou vlastní adresu, abyste mohli zkontrolovat, zda vám mail skutečně přijde. Mail je velmi jednoduchý, ale to není podstatné, podstatné je si vyzkoušet, že mail umíme odeslat.

Odeslání víceřádkového mailu

Pokud budeme chtít odeslat v textu zprávy, tedy ve třetím parametru funkce mail, víceřádkový text, potom ukončujeme jednotlivé řádky pomocí znaku konce řádku \n. Takový řetězec potom musí být uzavřen do dvojítych uvozovek, jinak znak konce řádku nebude rozpoznán:

```
<html>
<head>
<title>Příklad 2. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $email = `vase.adresa@neco.cz`;
    $vysledek = mail($email, `Předmět mailu`, "Řádek 1\nŘádek 2\nŘádek 3");
    if ($vysledek)
        echo `Mail úspěšně odeslán`;
    else
        echo `Mail nebyl odeslán, nastala chyba`;
?>
</body>
</html>
```

Odeslání více příjemcům

Pokud budeme chtít mail najednou odeslat více příjemcům, stačí je napsat naráz a oddělit čárkami (před zkoušením nezapomeňte nahradit e-maily [první.adresa@neco.cz](mailto:prvni.adresa@neco.cz) a druha.adresa@neco.cz nějakými vašimi mailovými existujícími adresami):

```
<html>
<head>
<title>Příklad 3. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $email = `prvni.adresa@neco.cz,druha.adresa@neco.cz`;
    $vysledek = mail($email, `Předmět mailu`, "Řádek 1\nŘádek 2\nŘádek 3");
    if ($vysledek)
        echo `Mail úspěšně odeslán`;
    else
        echo `Mail nebyl odeslán, nastala chyba`;
?>
</body>
</html>
```

Funkce mail ještě zvládá mailovou adresu ve tvaru se jménem. Například pokud by měl Petr adresu petr.zavadil@neco.cz, potom stačí uvést:

```
$email = `Petr<petr.zavadil@neco.cz>`;
```

A co další možnosti?

Funkce mail umožňuje vkládat i další informace do mailu, ale to lze už jen přes čtvrtý parametr funkce mail, který znamená dodatečné hlavičky.

Hlavička znamená určitou informaci, kterou dáváme poštovnímu programu. Každá hlavička končí znakem konce řádku \n.

Zde je příklad, jak pomocí dodatečné hlavičky s názvem From můžeme určit, od koho je mail odeslán. V následujícím příkladě dostaneme mail od Jánošíka:

```
<html>
<head>
<title>Příklad 4. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $email = `vase.adresa@neco.cz`;
    $hlavicky = "From:janosik@neco.cz\n";
    $vysledek = mail($email, `Předmět mailu`, "Text mailu", $hlavicky);
    if ($vysledek)
        echo `Mail úspěšně odeslán`;
    else
        echo `Mail nebyl odeslán, nastala chyba`;
?>
</body>
</html>
```

Jednoduchý formulář pro odeslání mailu

Na závěr si ukážeme, jak udělat jednoduchý formulář pro odeslání zprávy z webové stránky na váš mail. Potřebuji k tomu celkem dva skripty, jeden bude dělat formulář a druhý odešle mail.

Zde je první skript, uložte ho do souboru s názvem formulář.php:

```
<html>
<head>
<title>Příklad 5. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<form action="odmailuj.php" method="post">
<b>Zpráva:</b>
<br>
<textarea name="zprava" rows="5">
</textarea>
<br>
<input type="submit" value="Odešli zprávu">
</form>
</body>
</html>
```

Tento skript vytvoří jednoduchý formulář pro zadání zprávy.

A zde je druhý skript, který uložte do souboru s názvem odmailuj.php (nezapomeňte přepsat adresu vase.adresa@neco.cz vaší skutečnou mailovou adresou):

```
<html>
<head>
<title>Příklad 6. z 19. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $zprava = $_POST[`zprava`];
    $email = `vase.adresa@neco.cz`;
    $vysledek = mail($email, `Mail z WWW`, $zprava);
    if ($vysledek)
        echo `Mail úspěšně odeslán`;
    else
        echo `Mail nebyl odeslán, nastala chyba`;
?>
</body>
</html>
```

Pro vyzkoušení spusťte první skript, tedy formular.php, druhý se zavolá automaticky pro odeslání zprávy.

PHP - 20. díl – úvod do konfigurace PHP

Mnoho vlastností PHP závisí na tom, jak je nastavena jeho konfigurace. Vlastnosti lze měnit v poměrně širokých mezích, dají se přidávat, či odebírat různé PHP moduly a mnohé další. Je načase říci si o konfigurování pár slov nebo alespoň základní informace.

Základní konfigurační soubor pro PHP je soubor s názvem php.ini, který může být podle okolností uložen v nejrůznějších adresářích. To, v jakém je adresáři, závisí především na operačním systému, pod kterým PHP běží. Pokud máte nainstalovaný pokusný intranetový server ze 2. dílu, najdete jej uložený na tomto místě: C:\inet_srv\apache\bin\php.ini.

Mnoho PHP skriptů je napsáno tak, že funguje jenom s určitým nastavením, a pokud nastavíte PHP jinak, pak takový skript nebude vůbec fungovat, nebo bude fungovat špatně. Měli byste se v rozumné míře snažit, aby vaše PHP skripty fungovaly v nejširším rozmezí nastavení PHP konfigurace.

Jaká je struktura souboru php.ini?

Pokud se podíváte do souboru php.ini, najdete tam řadu řádků, které jsou pořád ve stejném tvaru:

```
parametr = hodnota
```

Pokud řádek začíná středníkem, jedná se o komentářový řádek, který může obsahovat cokoli, a který je celý ignorován.

Právě dvojice parametr = hodnota mají nejvyšší důležitost při konfiguraci PHP. Každý parametr má určitý význam a nastavuje se jeho hodnota.

Například najdete-li řádku s parametrem **max_execution_time**, nastavujete, kolik sekund maximálně smí PHP skript běžet. Pokud PHP skript neproběhne ve stanovené lhůtě, je násilně ukončen. Taková řádka pak vypadá obvykle takto:

```
max_execution_time = 30
```

Výše uvedená řádka říká, že PHP skript má maximálně 30 sekund na svoji činnost.

Najdete-li řádku s parametrem **memory_limit**, pak vězte, že tento parametr určuje, kolik paměti smí PHP skript maximálně zabrat. Standardně bývá nastaveno 8 MB:

```
memory_limit = 8M
```

Jedna ze zajímavých voleb je **short_open_tag**. Tato volba povoluje (hodnota On), nebo zakazuje (hodnota Off) uzavírání PHP kódů do <? a ?>. Pokud potřebujete rozchodit nějaký PHP skript z internetu, jehož autor neuzavíral kódy do <?php a ?>, ale pouze do <? a ?>, musíte povolit tuto volbu:

```
short_open_tag = On
```

Pokud budete mít volbu short_open_tag zakázanou, nebudou vám takové PHP skripty, které jsou uzavřeny do <? a ?> vůbec chodit.

Velmi dalekosáhlá volba je ukrytá v parametru **safe_mode**. Ta zapíná tzv. bezpečný režim, který zavádí řadu omezení do PHP z bezpečnostních důvodů. Pokud je volba povolena (hodnota On), je zvolen bezpečný režim, a některé funkce PHP jsou omezeny, nebo úplně znemožněny. Pokud je volba zakázána (hodnota Off), PHP je v normálním režimu. Bezpečný režim má význam hlavně pro hostingy, a všude, kde by se v PHP mohly potenciálně dít nebezpečné operace.

Samotných možných parametrů je daleko více, výše uvedené jsou jenom pro ilustraci. Všechny volby jsou popsány v manuálu k PHP, a budete-li chtít se vrtat v konfiguraci PHP více, určitě si budete muset vzít k ruce manuál.

Stejně tak najdete v souboru php.ini řadu dalších parametrů s hodnotami. Každý parametr má svůj význam a představuje tak volbu, kterou lze nějak nastavit.

Jak zjistit současné nastavení PHP?

Pro první přehled, a při hledání problémů s PHP se velmi hodí možnost zjistit současnou konfiguraci PHP. Taková možnost přímo v PHP je zabudována, a to pomocí funkce **phpinfo**, která vypíše velké množství informací o současném nastavení PHP. Samotný výstup je velmi pěkně formátován jako HTML výstup. Samotný skript, který zařídí vypsání informací pomocí funkce phpinfo už nemůže být jednodušší:

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

Samotná funkce phpinfo vypisuje velmi podrobnou zprávu o konfiguraci PHP, a tak se stává neocenitelným pomocníkem při řešení problémů s PHP. Jediné, co vám může dělat problém je to, že phpinfo vypisuje informace v anglickém jazyce. Každopádně informace jsou zformátovány do přehledných tabulek, ve kterých se velmi dobře orientuje.

Pokud máte hosting, který sami nespravujete, je použití funkce phpinfo často jediná rozumná volba, jak vůbec nastavení PHP zjistit. V takovém případě se totiž nemůžete podívat do konfiguračního souboru php.ini, ani přímo netušíte, s jakou verzí PHP máte co do činění.

Někdy se může hodit použití méně komplexní funkce **phpversion**, která vrací jenom řetězec s číslem verze PHP, pod kterou váš skript zrovna běží:

```
<html>
<head>
<title>Příklad 2. z 20. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Běžím pod verzí PHP `,phpversion();
?>
</body>
</html>
```

Co jsou to moduly?

PHP je stavěno trochu jako skládačka. Má svojí základní část a mnohá funkcionalita PHP je součástí externích modulů. Proto některé funkce mohou někde chodit a někde naopak nefungovat. Záleží totiž na tom, jaké moduly jsou zrovna nataženy. Všechny natažené externí moduly mají v php.ini řádek:

```
extension = jméno_modulu
```

Často jsou v php.ini vyjmenovány externí moduly všechny, ale před nepoužívanými je středník, takže je řádka braná jako komentář, a tudíž ignorována.

Zjistit, které všechny moduly právě v PHP běží je také možné pomocí funkce phpinfo. I když funkce phpinfo vypisuje nejenom externí moduly, ale také moduly zkompilované přímo do PHP.

Nabízených standardních modulů dodávaných s PHP je asi několik desítek, a pokud budete šikovný programátor, můžete si napsat klidně i modul vlastní, pokud to pro vás bude mít smysl. Moduly umožňují rozšiřování PHP a přidávání dalších funkcí, a dělají tak z PHP vlastně takovou malou stavebnici. Vlastně se dá s trochou nadsázky říci, že PHP je celé složeno z modulů.

Počítejte s tím, že občas bude nutné natáhnout některý modul, aby se vám rozběhly některé skripty třeba z internetu. Například pro práci s obrázky je nutné natáhnout GD modul, pro práci s PDF soubory je potřeba natáhnout PDF modul, apod.. Pokud nebude příslušný modul natažen, PHP se bude tvářit, jako kdyby vůbec dané funkce neznal a neuměl používat.

Ovlivňování konfigurace za běhu

Do této chvíle jsem psal, že všechno nastavení je v php.ini. A je to v zásadě pravda, protože konfigurační soubor php.ini je základ, od kterého se odvíjí veškeré další nastavení. Je dále možné mírně pozměňovat nastavení i za běhu PHP skriptů, ale to slouží především k mírnému doladování. Navíc nastavovat konfigurační parametry za běhu není možné vždycky, je-li PHP v tzv. bezpečném režimu, jsou tyto funkce značně omezeny.

Existuje například funkce **set_time_limit**, která umožňuje nastavit maximální dobu provádění PHP skriptu v sekundách. Můžeme tak pro nějaký dlouho trvající PHP skript nastavit dostatečně dlouhou dobu, a zabránit tak tomu, aby se předčasně neukončil. Standardně má PHP skript půl minuty na to, aby vykonal vše co je potřeba, ale pokud potřebujeme prodloužit tento interval třeba na pět minut, stačí začít PHP skript takto:

```
set_time_limit(300);
```

Kromě toho existuje i obecná funkce **ini_set**, která umožňuje nastavovat parametry i za běhu. Tato funkce má tvar:

```
ini_set (parametr, hodnota)
```

Jako parametr se zadává parametr konfigurační volby jako v php.ini. Jako hodnotu zadáme novou hodnotu. Funkce ini_set vrací předchozí, starou hodnotu, pokud je úspěšná. Zde je malý příklad, jak pomocí funkce ini_set dokázat to samé, co o pár odstavců výše. Tedy prodloužit maximální dobu provádění PHP skriptu na 5 minut:

```
<html>
<head>
<title>Příklad 3. z 20. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $predchozi_limit_casu = ini_set('max_execution_time',300);
    echo $predchozi_limit_casu;
?>
</body>
</html>
```

Příklad by měl nastavit maximální dobu provádění skriptu na 5 minut a vypsát předchozí hodnotu limitu v sekundách (většinou vypíše 30).

PHP - 21. díl – začínáme s obrázky

PHP dokáže vytvářet, měnit a provádět nejrůznější akce s obrázky, což se může v mnoha případech hodit. Ať už pro vytváření grafů, změnu velikostí obrázků či další použití. Podíváme se, jak na to.

Možnost práce s obrázky není v PHP automatická, ale opírá se o takzvanou GD knihovnu, kterou je většinou nutné připojit jako externí modul. Pokud nemáte podporu GD knihovny, pak ani PHP s obrázky pracovat nedokáže. Pokud zkoušíte na intranetovém serveru ze 2. dílu, pak podpora GD knihovny je už automaticky zahrnuta.

Nejdříve bude tedy vhodné vůbec zjistit, zda PHP, na kterém zkoušíte, vůbec umí pracovat s obrázky. Nejjednodušeji to lze zjištěním konfigurace pomocí už nám známého jednoduchého skriptu:

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

Pokud se ve výpisu konfigurace objeví tabulka nadepsaná gd, potom PHP má podporu obrázků. V tabulce pod nápisem gd pak můžete zjistit detaily ohledně podpory obrázků.

S jakými formáty obrázků dokáže pracovat PHP? Pokud máte obvyklou GD knihovnu verze 2, pak bezproblémové jsou zejména formáty obrázků JPEG a PNG. Celkem běžný formát obrázku GIF je bohužel problematický, protože od jeho podpory se kdysi upustilo z důvodů licenčních potíží.

Zjištění informací o souboru s obrázkem

Pro zjištění informací o souboru s obrázkem existuje funkce **getimagesize**, která zjistí formát obrázku a jeho velikost. Připravte si proto nějaký obrázek, třeba ve formátu JPEG, přejmenujte ho na soubor obr1.jpg, nahrajte ho do stejného adresáře, jako následující PHP skript, a zkusme si zjistit, jakou má obrázek velikost:

```
<html>
<head>
<title>Příklad 1. z 21. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<body>
<?php
    $informace = getimagesize('obr1.jpg');
    echo `Šířka obrázku je `, $informace[0], ` pixelů.<br>`;
    echo `Výška obrázku je `, $informace[1], ` pixelů.<br>`;
    echo `Formát obrázku je číslo `, $informace[2], `.`;
?>
</body>
</html>
```

Pokud si výše uvedený příklad spustíte, zjistíte šířku a výšku obrázku a také číslo formátu (mělo by být 2). Číslo formátu určuje, o jaký formát se jedná, a může vám posloužit, pokud potřebujete nutně znát, o jaký formát jde. Co které číslo formátu znamená se dozvíte v následující tabulce:

Číslo	Formát obrázku
1	GIF
2	JPEG
3	PNG
4	SWF
5	PSD
6	BMP
7	TIFF(intel byte order)
8	TIFF(motorola byte order)

9	JPC
10	JP2
11	JPX
12	JB2
13	SWC
14	IFF
15	WBMP
16	XBM

Funkce `getimagesize` vrací pole, kde index s číslem nula obsahuje šířku obrázku v pixelech, index s číslem jedna výšku obrázku v pixelech a index s číslem dva vrací číslo formátu. Tak je to ostatně vidět i v příkladu.

Samotná funkce `getimagesize`, kterou jsme použili pro určení vlastností obrázků, má jedno výjimečné postavení. Nepotřebuje GD knihovnu, a proto výše uvedený příklad bude pracovat naprosto všude i tam, kde GD knihovna není k dispozici. Protože funkce `getimagesize` nemá ani omezení GD knihovny, dokáže pracovat se spoustou formátů.

Jak se pracuje s funkcemi obrázkové knihovny?

Skoro všechny funkce, které se starají o vytváření nebo úpravu obrázků, začínají v názvu slovem `image`. Teď popíšu, jaká je filozofie práce při editaci obrázků.

Každý obrázek, se kterým se pracuje, je reprezentován speciálním obrázkovým identifikátorem. Každý obrázek jednoduše existuje jako jedna proměnná, která je v manuálu nazývána obrázkový identifikátor. A veškeré akce se dějí právě s touto proměnnou.

Každý obrázek, s nímž chceme pracovat, zpracováváme tak, že nejdříve z něj vytvoříme obrázkový identifikátor. To znamená, že obrázek nahrajeme buď z souboru, nebo vytvoříme prázdný.

Zde je jednoduchý příklad práce s obrázkem v PHP. Přiznám se, že teď trochu předbíhám pro ilustraci, jak pracovat s obrázky, a proto je příklad trochu komplexnější:

```
<html>
<head>
<title>Příklad 2. z 21. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Vytvářím červený banner`;
    $obrazek = imagecreatetruecolor(468,60);
    $cervena_barva = imagecolorallocate($obrazek,255,0,0);
    imagefill($obrazek,0,0,$cervena_barva);
    imagepng($obrazek,`obr2.png`);
    imagedestroy($obrazek);
?>
</body>
</html>
```

Pokud si příklad spustíte, zjistíte, že v adresáři, kde je umístěný skript, vám vznikl obrázek v souboru s názvem `obr2.png`. Obrázek obsahuje banner o velikosti 468 x 80 pixelů a je vybarven červenou barvou. Na příkladu chci hlavně ukázat, jaký je základní princip práce s obrázkem v PHP:

- Vytvořím obrázek, ať už prázdný, nebo ze souboru, a obdržím proměnnou typu obrázek (obrázkový identifikátor). V našem příkladu je to proměnná s názvem `$obrazek`.
- Pracuji s obrázkem pomocí proměnné typu obrázek (obrázkového identifikátoru) a postupně na obrázek použít veškeré editační akce, které chci s obrázkem provést.
- Použiji výsledný obrázek.
- Zruším obrázkový identifikátor pomocí funkce **`imagedestroy`**. Pokud jej zapomenu zrušit, udělá to PHP skript sám automaticky na konci skriptu.

Výše uvedené 4 body v zásadě platí pro každou práci s obrázkem, ať provádíme cokoli.

Jak vytvořit proměnnou typu obrázek?

Před jakoukoli prací je potřeba vytvořit proměnnou typu obrázek. Toho je možné docílit několika způsoby, které lze rozdělit v zásadě na vytvoření nového prázdného obrázku nebo nahrátí obrázku ze souboru.

Pro tvorbu prázdného obrázku máme k dispozici dvě funkce: funkci **`imagecreatetruecolor`** a **`imagecreate`**. Rozdíl mezi nimi je malý, osobně doporučuji používat tu první z nich. Při pokusech s funkcí `imagecreate` se mi totiž pravidelně stávalo, že nezvládla pracovat s více než 256 barvami. Pokud obrázek obsahoval více barev, funkce `imagecreate` tyto barvy poničila. Proto doporučuji používat `imagecreatetruecolor`. Tato funkce vytvoří prázdný obrázek o zadané šířce a výšce:

```
imagecreatetruecolor (šířka, výška)
```

Funkce vrátí proměnnou typu obrázek (obrázkový identifikátor), kterou je možné potom použít pro další akce s obrázkem.

Kromě toho existují i jiné funkce, jak vytvořit proměnnou typu obrázek, a to nahrátím ze souboru. Zde se musí použít ta správná funkce podle typu obrázku. Pokud máte obrázek typu JPEG, pak se použije funkce **`imagecreatefromjpeg`**. Pokud máte obrázek typu PNG, pak se musí použít funkce **`imagecreatefrompng`**. Obě funkce očekávají jako parametr jméno souboru a vracejí proměnnou typu obrázek:

```
imagecreatefromjpeg (jméno_souboru)
```

```
imagecreatefrompng (jméno_souboru)
```

Jak použít proměnnou typu obrázek?

Pokud už máme obrázek v proměnné typu obrázek, můžeme provádět libovolné akce s obrázkem. Výsledkem by ale mělo být, že by se obrázek měl k něčemu použít. A to něco je buď uložení do souboru, nebo poslání někam.

Uložení obrázku do souboru je velice jednoduché. Pokud chci uložit obrázek jako typ PNG, potom použiji funkci **imagepng**. Funkce může mít buď jeden, nebo dva parametry:

```
imagepng (proměnná_typu_obrázek, jméno_souboru)
```

Pokud chybí druhý parametr u funkce imagepng, tedy chybí jméno souboru, potom se výsledný obrázek vypíše na standardní výstup, přímo jako součást výsledného HTML. Jak toho využít si řekneme později, zatím budeme vždy uvádět jméno souboru.

Pokud chci uložit obrázek ve formátu JPEG, musím použít funkci **imagejpeg**. Funkce může mít jeden, dva nebo tři parametry.

```
imagejpeg (proměnná_typu_obrázek, jméno_souboru, kvalita)
```

Třetí parametr kvalita je číslo od nuly do sta, přičemž nula znamená nejhorší kvalita, ale také nejmenší velikost souboru. Stovka znamená největší kvalita, ale také největší soubor. Pokud se třetí parametr kvalita nepoužije, potom se automaticky dosadí číslo 75.

Druhý parametr jméno_souboru je jméno souboru, pod kterým bude obrázek uložen. Pokud druhý parametr chybí, výsledný obrázek se vypíše na standardní výstup, tedy jako součást výsledného HTML.

Jednoduchý příklad na převod formátů

Ted' už známe dost na to, abychom mohli uvést jednoduchý příklad na převod formátů obrázků. Připravte si nějaký pokusný JPEG obrázek v souboru obr1.jpg a dejte ho do stejného adresáře jako následující PHP skript:

```
<html>
<head>
<title>Příklad 3. z 21. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Převod z formátu JPEG do PNG.`;
    $obrazek = imagecreatefromjpeg(`obr1.jpg`);
    imagepng($obrazek, `obr1.png`);
    imagedestroy($obrazek);
?>
</body>
</html>
```

Po spuštění příkladu se obrázek převede do formátu PNG. Ve stejném adresáři jako je PHP skript se vám objeví i soubor s obrázkem obr1.png.

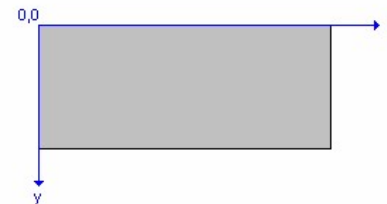
Jak to celé funguje? Jednoduše. Nejdříve se pomocí funkce imagecreatefromjpeg natáhne obrázek ve formátu JPEG. Pokud je obrázek již natáhnutý do proměnné typu obrázek, potom nic nebrání uložit jej v jakémkoli formátu požadujeme. Proto na dalším řádku využiji funkce imagepng k uložení obrázku ve formátu PNG. A nakonec uvolním proměnnou typu obrázek pomocí imagedestroy, protože ji už nepotřebuji.

PHP - 22. díl – vytváření obrázků

Tento díl těsně navazuje na předchozí díl o základech práce s obrázky. Dnešní část bude věnována hlavně základním editačním akcím.

Souřadnicový systém obrázků

Prakticky při každé editační akci se zadávají souřadnice uvnitř obrázku. Souřadnice jsou natočeny tak, že bod 0,0 je vlevo nahoře. Souřadnice x se počítá směrem doprava a souřadnice y se počítá směrem dolů. Ostatně podívejte se na obrázek.



Jak vyplnit obdélník barvou?

Jedním ze základních editačních příkazů je vyplnění obdélníku barvou. Začnu hned příkladem, kdy vyrobím banner o velikosti 468 x 60 pixelů, který vyplním dvěma barvami, červenou a modrou. Zkuste si spustit tento příklad, který po spuštění zanechá ve stejném adresáři, jako je PHP skript soubor banner1.png s obrázkem:

```
<html>
<head>
<title>Příklad 1. z 22. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    echo `Dvoubarevný banner`;
    $obrazek = imagecreatetruecolor(468,60);
    $modra = imagecolorallocate($obrazek,0,0,255);
    $cervena = imagecolorallocate($obrazek,255,0,0);
    imagefilledrectangle($obrazek,0,0,467,59,$cervena);
    imagefilledrectangle($obrazek,117,15,351,45,$modra);
    imagepng($obrazek, `banner1.png`);
    imagedestroy($obrazek);
?>
</body>
</html>
```

V příkladu je krásně vidět práce s barvami. Každou barvu, kterou chcete na obrázek použít, musíte nejdříve naalokovat, tedy vytvořit. Tedy použít funkci **imagecolorallocate**, která

jakoby vyrobí novou barvu a zapamatovat si její návratovou hodnotu. Návratovou hodnotu pak použijeme všude, kde se požaduje zadaná barva.

Funkce `imagecolorallocate` má následující parametry:

```
imagecolorallocate (obrázek, červená_složka, zelená_složka, modrá_složka)
```

První parametr obrázek je proměnná typu obrázek (obrázkový identifikátor), další parametry udávají hodnotu jednotlivých barevných složek. Každá složka může být číslo od nuly do 255.

Funkce `imagecolorallocate` nám tedy namíchá novou barvu, a když si zapamatujeme její návratovou hodnotu, můžeme do obrázku dosazovat namíchanou barvu všude, kde je nám libo.

Další funkce **`imagefilledrectange`** slouží k vyplnění obdélníku barvou. Samotná funkce má následující parametry:

```
imagefilledrectangle (obrázek, x1, y1, x2, y2, barva)
```

První parametr je proměnná typu obrázek (obrázkový identifikátor). Druhý a třetí parametr jsou `x` a `y` souřadnice levého horního rohu, který se má vyplnit. Čtvrtý a pátý parametr jsou `x` a `y` souřadnice pravého dolního rohu, který se má vyplnit. A konečně poslední parametr je barva, kterou se bude vyplňovat.

V příkladu jsou použity dvě volání funkce `imagefilledrectangle`, vyplňují se tedy dva obdélníky. Nejříve se celý banner vyplní červenou barvou a pak uvnitř menší obdélník modrou barvou.

Jak vykreslit obrázek přímo?

Veškeré příklady do této chvíle vykreslovaly obrázek do souboru. Pro účely psaní webů ale potřebujeme dané obrázky zobrazit, a to je při ukládání do souborů trochu přes ruku. Ovšem existuje i taková možnost, jak obrázky zobrazit v prohlížeči přímo. Stačí k tomu poslat hlavičku s tzv. MIME typem, tedy typem obsahu. Stačí si zapamatovat, že JPEG soubor má MIME typ `image/jpeg` a PNG soubor má MIME typu `image/png`. O hlavičkách jsem mluvil ve 13. dílu seriálu, tady stačí zopakovat, že hlavička se musí poslat dříve, než jakýkoli výstup. Proto v následujícím příkladu zmizí i HTML okolí:

```
<?php
$obrazek = imagecreatetruecolor(468,60);
$modra = imagecolorallocate($obrazek,0,0,255);
$cervena = imagecolorallocate($obrazek,255,0,0);
imagefilledrectangle($obrazek,0,0,467,59,$cervena);
imagefilledrectangle($obrazek,117,15,351,45,$modra);
header('Content-Type: image/png');
imagepng($obrazek);
imagedestroy($obrazek);
?>
```

Pokud si vyzkoušíte tento příklad, zjistíte, že výsledkem je stejný banner, jako v předchozím příkladě, ale na rozdíl od něj vidíte banner přímo v prohlížeči. Mohou za to řádky:

```
header('Content-Type: image/png');
imagepng($obrazek);
```

Funkce `header` přidá hlavičky typu "Content-Type", za kterou následuje MIME typ obsahu, v našem příkladě je to `image/png`, tedy PNG obrázek. V dalším řádku je funkce `imagepng`, která ukládá obrázek v PNG formátu, ale na rozdíl od předchozího příkladu nám tu chybí jméno souboru. A pokud chybí jméno souboru, pošle se obrázek na standardní výstup, tedy přímo do prohlížeče.

Jak vykreslovat jednotlivé body?

Jednotlivé body lze vykreslit pomocí funkce **`imagesetpixel`**. Tato funkce nakreslí jeden jediný bod zadanou barvou:

```
imagesetpixel (obrázek, x, y, barva)
```

Prvním parametrem je proměnná typu obrázek, následuje `x` a `y` souřadnice bodu, který chceme nakreslit a posledním parametrem je barva bodu.

Jednoduchý příklad, který nakreslí červený banner, a do něj malinkou čárečku ze tří modrých bodů:

```
<?php
$obrazek = imagecreatetruecolor(468,60);
$modra = imagecolorallocate($obrazek,0,0,255);
$cervena = imagecolorallocate($obrazek,255,0,0);
imagefilledrectangle($obrazek,0,0,467,59,$cervena);
imagesetpixel($obrazek,117,15,$modra);
imagesetpixel($obrazek,117,16,$modra);
imagesetpixel($obrazek,117,17,$modra);
header('Content-Type: image/png');
imagepng($obrazek);
imagedestroy($obrazek);
?>
```

Jak kreslit čáry?

Pro kreslení čar existuje funkce **`imageline`**:

```
imageline (obrázek, x1, y1, x2, y2, barva)
```

Prvním parametrem je proměnná typu obrázek, následující 2 parametry jsou `x` a `y` začátku čáry, další 2 parametry jsou `x` a `y` konce čáry a posledním parametrem je barva.

Zde je jednoduchý příklad na kreslení čáry:

```
<?php
$obrazek = imagecreatetruecolor(468,60);
$zelena = imagecolorallocate($obrazek,0,255,0);
$cervena = imagecolorallocate($obrazek,255,0,0);
imagefilledrectangle($obrazek,0,0,467,59,$zelena);
imageline($obrazek,0,30,467,30,$cervena);
header('Content-Type: image/png');
imagepng($obrazek);
```

```
    imagedestroy($obrazek);
?>
```

Pro práci s čárami existuje ještě funkce **imagegetthickness**, která nastavuje šířku čáry:

```
imagegetthickness (obrázek, šířka_čáry_v_pixelech)
```

Standardně je nastavena šířka čáry 1 pixel, ale můžeme snadno zvětšit tloušťku čáry třeba na 10 pixelů:

```
<?php
$obrazek = imagecreatetruecolor(468,60);
$zelena = imagecolorallocate($obrazek,0,255,0);
$cervena = imagecolorallocate($obrazek,255,0,0);
imagefilledrectangle($obrazek,0,0,467,59,$zelena);
imagegetthickness($obrazek,10);
imageline($obrazek,0,30,467,30,$cervena);
header('Content-Type: image/png');
imagepng($obrazek);
imagedestroy($obrazek);
?>
```

Jak psát do obrázků?

Do obrázků lze přímo psát i texty. A to jednoduše pomocí funkce **imagestring**:

```
imagestring (obrázek, číslo_fontu, x, y, text, barva)
```

Prvním parametrem je proměnná typu obrázek. Druhým parametrem je číslo fontu, následují dva parametry udávající souřadnice levého horního rohu, kam se začne psát. Předposledním parametrem je text, který se má vypsát a posledním parametrem je barva textu.

Pokud se jako číslo fontu použije číslo od jedné do pěti, znamená to vestavěné fonty. Číslo 1 představuje nejmenší font, číslo 5 je největší font.

Tady je jednoduchý příklad, který do banneru vypíše text "Ahoj!":

```
<?php
$obrazek = imagecreatetruecolor(468,60);
$zelena = imagecolorallocate($obrazek,0,255,0);
$cervena = imagecolorallocate($obrazek,255,0,0);
imagefilledrectangle($obrazek,0,0,467,59,$zelena);
imagestring($obrazek, 5, 0, 0, "Ahoj!", $cervena);
header('Content-Type: image/png');
imagepng($obrazek);
imagedestroy($obrazek);
?>
```

Tento díl se bude zabývat často opomíjeným tématem v PHP, a to chybami. Povíme si, jak vznikají, jak se dají ošetřit, a vůbec, co všechno nám PHP v souvislosti s chybami dovoluje.

Kategorie chyb v PHP

Samotný systém PHP rozděluje chyby do několika kategorií, zjednodušeně se dá říci, že podle závažnosti chyby a podle zdroje chyby:

Kategorie chyby	Popis	Číslo chyby
E_ERROR	Fatální chyba, skript je po výskytu takové chyby předčasně ukončen.	1
E_WARNING	Varovná chyba.	2
E_PARSE	Chyba při kompilaci skriptu.	4
E_NOTICE	Poznámka, která může znamenat nějaké opomenutí, nebo chybu.	8
E_CORE_ERROR	Fatální chyba generovaná jádrem PHP.	16
E_CORE_WARNING	Varovná chyba generovaná jádrem PHP.	32
E_COMPILE_ERROR	Fatální chyba vzniklá při kompilaci skriptu.	64
E_COMPILE_WARNING	Varovná chyba vzniklá při kompilaci skriptu.	128
E_USER_ERROR	Uživatelé generovaná chyba.	256
E_USER_WARNING	Uživatelé generované varování.	512
E_USER_NOTICE	Uživatelé generovaná poznámka.	1024

Některé kategorie chyb je možné do jisté míry kontrolovat. Je možné ošetřovat chyby, které vzniknout za běhu skriptu. Naopak s chybami, které vzniknou například jako důsledek syntaktické chyby ve skriptu moc nenaděláme.

Konfigurace v php.ini

V konfiguračním souboru php.ini jsou dva důležité parametry, které ovlivňují výpisy chyb. Pokud totiž nastane chyba, obvykle nastane výpis chyby na obrazovku. Tento výpis chyby můžeme řídit pomocí parametrů `error_reporting` a `display_errors`. Parametr `error_reporting` nastavuje, které kategorie chyb se budou hlásit na obrazovku. Obvyklé je hlášení všech kategorií chyb s výjimkou `E_NOTICE`. Parametr `display_errors` nastavuje, zda se vůbec budou chyby vypisovat jako součást výstupu, nebo ne. Obvykle je nastaveno, že se chyby vypisují.

Příklady chyb

Aby to celé bylo jasnější, uvedu zde několik příkladů generování chyby. Zde je syntaktická chyba ve skriptu (chybí středník na konci prvního příkazu):

```
<?php
$a = 1 + 1
$b = $a + 1;
?>
```

Pokud si zkusíte výše uvedený PHP skript spustit, skončí to výpisem chyby kategorie `E_PARSE`. Tato chyba označuje chybu v zápisu skriptu.

Příklad chyby za běhu skriptu lze také vytvořit snadno, například takto (v našem příkladě dělení nulou):

```
<html>
<head>
<title>Příklad 2. z 23. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $a = 0;
    echo `Před dělením nulou<br>`;
    $b = 3 / $a;
    echo `Po dělení nulou<br>`;
?>
</body>
</html>
```

Pokud si příklad vyzkoušíte, zjistíte, že PHP vám do něj vypíše anglickou zprávu o tom, že jste dělili nulou. Protože však jde o pouhé varování, tedy kategorii E_WARNING, nepovažujte to za důvod pro ukončení skriptu, který běží dále až do samého konce.

Obyčejnou, ale často přehlíženou chybou je i použití neexistující proměnné. Většina PHP systémů je totiž nastavena tak, že tuto chybu nijak neoznámí:

```
<?php
    $b = $a + 1;
    echo $b;
?>
```

Ve výše uvedeném příkladu používám proměnnou \$a, která ale není předtím nikde vytvořena. PHP skript potom vygeneruje chybu kategorie E_NOTICE, tedy nejméně závažnou chybu. Většina PHP systémů takovou chybu nevypíše, ale pokud používáte intranetový server z 2. dílu na testování, chybovou zprávu dostanete.

Při použití neexistující proměnné v našem příkladu se PHP skript zachová tak, jako kdyby neexistující proměnná měla nulovou hodnotu. Následující příkaz echo proto vypíše jedničku.

Operátor řízení chyb - @

V PHP existuje operátor řízení chyb, který blokuje výpis chybových zpráv. Je to znak zavináč @, který můžete předřadit před jakýkoli výraz, kde získáváte hodnotu. V tomto výrazu pak jsou blokovány jakékoli, i kritické chybové zprávy.

Například takto zablokujeme chybovou zprávu o dělení nulou:

```
<html>
<head>
<title>Příklad 4. z 23. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    $a = 0;
    echo `Před dělením nulou<br>`;
```

```
    $b = @(3 / $a);
    echo `Po dělení nulou<br>`;
?>
</body>
</html>
```

Jak je vidět, samotný výraz, kde se dělilo nulou jsem uzavřel do závorek, kterým jsem předřadil znak zavináč @. Ten zajistí, že žádná chybová zpráva o dělení nulou, ani žádná jiná, která by na tomto místě vznikla, nebude vypsána.

Stejně jednoduše je možné zablokovat chybové zprávy v jakémkoli dalším příkladě, například i zprávu o případné neexistující proměnné:

```
<?php
    $b = @$a + 1;
    echo $b;
?>
```

Blokování chyb pomocí znaku zavináče @ pracuje jen v případě chyb za běhu. Nelze se takto zbavovat chybových hlášení tam, kde je špatná syntaxe, apod..

Každopádně používejte operátor řízení chyb, tedy znak zavináč @ jenom v co nejmenší možné míře. Mohli byste tak přijít o zobrazení závažné chyby, kterou byste potom dlouho hledali.

Generování vlastních chyb

Někdy se hodí vygenerovat vlastní chybu. Hodí se to v situaci, kdy pro náš případ znamená určitá situace chybu, i když pro PHP je tato situace naprosto v pořádku. Generovat se dají tři kategorie chyb, a to chyba E_USER_ERROR, E_USER_WARNING a E_USER_NOTICE. První z nich je nejzávažnější a předčasně ukončuje běh skriptu, druhá je méně závažná a třetí je nejméně závažná. Druhá a třetí kategorie dovoluje PHP skriptu pokračovat v běhu.

Pro generování vlastních chyb slouží funkce trigger_error:

```
<html>
<head>
<title>Příklad 6. z 23. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
    trigger_error(`Moje vlastní chyba`, E_USER_ERROR);
?>
</body>
</html>
```

Výše uvedený příklad vypíše vlastní chybu "Moje vlastní chyba".

Funkce trigger_error má dva parametry:

```
trigger_error (chybová_zpráva, kategorie_chyby)
```

Vlastní ošetřování chyb

Pro určité případy se nám může hodit vlastní ošetření chyb namísto standardního vypsaní na obrazovku. PHP umožňuje zařídit právě to, aby se chyby automaticky posílaly námi napsané funkci namísto standardního zpracování chyb.

Zde je jednoduchý příklad, jak výpisy chyb počítit pomocí vlastního ošetření chyb:

```
<html>
<head>
<title>Příklad 7. z 23. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php

function MojeZpracovaniChyby ($cislo, $zprava, $soubor, $radka)
{
    echo "Chyba číslo $cislo: $zprava na řádku $radka v souboru
    $soubor.<br>";
}

set_error_handler(`MojeZpracovaniChyby`);

trigger_error(`Moje vlastní chyba`, E_USER_ERROR);

?>
</body>
</html>
```

V příkladu je vidět, jak se dělá vlastní obsluha chyb. Připravil jsem si vlastní funkci, která bude ošetřovat chyby, a kterou jsem nazval MojeZpracovaniChyby. Důležité je, že funkce má nejobvyklejší čtyři parametry, a to postupně číslo chyby, chybovou zprávu, název souboru a číslo řádku, kde nastala chyba.

Pak název funkce, která bude ošetřovat chyby předám do set_error_handler a od této chvíle se bude PHP při chybě obracet na mojí funkci, kdykoli nějaká chyba nastane.

PHP - 24. díl – konstrukce switch, cyklus while

Dnešní díl seriálu o PHP bude věnován dalším, zatím nevysvětleným řídicím strukturám switch, do, while, break a continue.

Konstrukce switch

Konstrukce switch slouží k porovnávání stejného výrazu s mnoha hodnotami. Přesně řečeno, je to něco, co byste zvládli napsat pomocí konstrukce if, ale switch může být daleko šikovnější. Například vypsaní školní známky slovně můžeme napsat pomocí konstrukcí if takto:

```
<html>
<head>
<title>Příklad 1. z 24. dílu</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$znamka = 3;

if ($znamka == 1)
    echo `jednička`;
if ($znamka == 2)
    echo `dvojka`;
if ($znamka == 3)
    echo `trojka`;
if ($znamka == 4)
    echo `čtyřka`;
if ($znamka == 5)
    echo `pětka`;
?>
</body>
</html>
```

V příkladu je nastavena proměnná \$znamka na trojku, můžete si vyzkoušet nastavení na jinou hodnotu. Ale hlavní je, že právě porovnávání s mnoha hodnotami je doména konstrukce switch. Podívejme se, jak by stejný příklad vypadal pomocí konstrukce switch:

```
<html>
<head>
<title>Příklad 2. z 24. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$znamka = 3;

switch ($znamka)
{
    case 1:
        echo `jednička`;
        break;
    case 2:
        echo `dvojka`;
        break;
    case 3:
        echo `trojka`;
        break;
    case 4:
        echo `čtyřka`;
        break;
    case 5:
        echo `pětka`;
        break;
}

?>
</body>
</html>
```

Tento výše uvedený příklad funguje naprosto stejně jako předchozí jenom je zapsán pomocí konstrukce switch. Konstrukce switch pracuje tak, že vyhodnotí výraz za slovem switch a pak

hledá příslušný řádek s case s odpovídající hodnotou. Protože v našem případě je výraz \$znamka za slovem switch roven hodnotě tři, začne vykonávání u řádku s case 3.

Je důležité pochopit, jak se příkaz switch provádí, aby se zabránilo chybám. Příkaz switch provádí řádek po řádku. Pouze tehdy, když se najde case s hodnotou odpovídající hodnotě výrazu u switch, začne PHP provádět následující příkazy. Vykonávání pokračuje, dokud se nedosáhne konce bloku switch nebo prvního příkazu break. Pokud nenapišete na konec bloku po case příkaz break, bude PHP pokračovat v provádění dalších příkazů (po dalším case). Zkuste si příklad, jak by to vypadalo, když chybí příkazy break:

```
<html>
<head>
<title>Příklad 3. z 24. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$znamka = 3;

switch ($znamka)
{
    case 1:
        echo `jednička`;
    case 2:
        echo `dvojka`;
    case 3:
        echo `trojka`;
    case 4:
        echo `čtyřka`;
    case 5:
        echo `pětka`;
}

?>
</body>
</html>
```

Pokud si tento příklad bez příkazů break zkusíte, zjistíte, že se provedou všechny příkazy za case 3, a to až do konce příkazu switch. To je proto, že žádný příkaz break nezabránil pokračování. Výsledkem bude vypsání nejen slova trojka, ale i slova čtyřka a pětka.

U příkazu switch existuje alternativní syntaxe. Náš příklad s vypsáním známky v alternativní syntaxi by vypadal takto:

```
<html>
<head>
<title>Příklad 4. z 24. dílu</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
</head>
<body>
<?php
$znamka = 3;

switch ($znamka):
    case 1:
        echo `jednička`;
        break;
```

```
    case 2:
        echo `dvojka`;
        break;
    case 3:
        echo `trojka`;
        break;
    case 4:
        echo `čtyřka`;
        break;
    case 5:
        echo `pětka`;
        break;
endswitch;
```

```
?>
</body>
</html>
```

Cyklus while

Cyklus while je nejjednodušším cyklem v PHP. Jeho tvar je:

```
while (podmínka)
    příkaz;
```

Cyklus while provádí příkaz stále dokola tak dlouho, dokud platí podmínka. Zde je příklad, který pomocí cyklu while vypíše čísla od jedné do deseti:

```
<html>
<head>
<title>Příklad 5. z 24. dílu</title>
</head>
<body>
<?php
    $i=1;
    while ($i <= 10)
    {
        echo $i,`<br>`;
        ++$i;
    }
?>
</body>
</html>
```

U cyklu while existuje alternativní syntaxe, která vypadá takto:

```
while (podmínka):
    seznam příkazů;
endwhile;
```

Předchozí příklad zapsaný pomocí alternativní syntaxe by vypadal takto:

```
<html>
<head>
<title>Příklad 6. z 24. dílu</title>
</head>
<body>
<?php
```

```

    $i=1;
    while ($i <= 10):
        echo $i,`<br>`;
        ++$i;
    endwhile;
?>
</body>
</html>

```

Cyklus do..while

Cyklus do..while je velmi podobný cyklu while s tím rozdílem, že podmínka se testuje až na konci každého provedeného cyklu, a nikoli na začátku, jako u cyklu while. Cyklus do..while má tento tvar:

```

do
    příkaz;
while (podmínka);

```

Protože se podmínka cyklu do..while testuje až na konci každého cyklu, musí se každý cyklus do..while provést alespoň jednou.

Zde je příklad, který zařídí vypsání čísel od jedné do deseti pomocí cyklu do..while:

```

<html>
<head>
<title>Příklad 7. z 24. dílu</title>
</head>
<body>
<?php
    $i=1;
    do
    {
        echo $i,`<br>`;
        ++$i;
    }
    while ($i <= 10);
?>
</body>
</html>

```

Příkaz break

S příkazem break jsme se už setkali u konstrukce switch. Příkaz break předčasně ukončuje provádění konstrukce switch, nebo jakéhokoliv cyklu, ať už cyklu while, do..while, for, nebo jiného. Například následující cyklus while vypíše díky příkazu break čísla jen od jedné do pěti:

```

<html>
<head>
<title>Příklad 8. z 24. dílu</title>
</head>
<body>
<?php
    $i=1;
    while (i <= 10)

```

```

    {
        echo $i, `<br>`;
        if (i >= 5)
            break;
        ++$i;
    }
?>
</body>
</html>

```

Ve výše uvedeném příkladu uvnitř cyklu while provedeme příkaz break, pokud se proměnná i stane větší, nebo rovnou pěti. To předčasně ukončí cyklus while už po vypsání čísla pět.

Příkaz break při použití uvnitř cyklu tedy ukončuje provádění dalších cyklů.

Příkaz continue

Příkaz continue se používá uvnitř cyklů. Dělá v podstatě velmi jednoduchou věc, a to tu, že se přeskočí zbytek aktuální cyklu a začne se cyklus nový. Vysvětlíme si to na příkladu, který vypíše pouze sudá čísla do deseti:

```

<html>
<head>
<title>Příklad 9. z 24. dílu</title>
</head>
<body>
<?php
    $i=1;
    while ($i <= 10)
    {
        ++$i;
        if (($i % 2) != 0)
            continue;
        echo $i, `<br>`;
    }
?>
</body>
</html>

```

Nejdříve musím vysvětlit, že výraz (\$i % 2) znamená zbytek po dělení dvěma. Pokud tedy zbytek po dělení dvěma není roven nule, jedná se o liché číslo a tehdy se dostane ke slovu příkaz continue. Protože continue přeskočí zbytek cyklu, přeskočí tedy příkaz echo, který se pro lichá čísla neprovede. Tím se zobrazí pouze sudá čísla. Příkaz continue způsobí, že se znovu pokračuje následující řádkou za while a přeskočí se zbytek cyklu.

PHP - 25. díl – úvod do databází

Pokud se řekne PHP, často se druhým dechem řekne databáze, nejčastěji MySQL. Protože použití databází spolu s PHP je velmi časté a účelné, budeme i v tomto seriálu věnovat databázím.

Samotné PHP dokáže spolupracovat s velkou spoustou různých databází, i když se asi nejčastěji používá s MySQL. V seriálu se zaměříme na základy používání pouze s MySQL databází, ale to vůbec neznamená, že PHP není použitelné s celou spoustou jiných databází.

Dnešní část bude spíše teoretický a krátký úvod do databází.

Co je to MySQL?

Jednoduše se dá říci, že MySQL je databázový systém. Takových systémů existuje celá řada, například Oracle, MS SQL, nebo třeba Sybase. MySQL je systém, který se etabloval především ve webových aplikacích, a který je dost preferovaný při spolupráci s PHP.

Samo o sobě patří MySQL spíše k těm jednoduchým databázovým systémům. Ve své podstatě je MySQL ořezaný o některé možnosti, kterou mají jiné databázové systémy. Důsledkem ořezání je nenáročnost MySQL na zdroje počítače a zvýšení rychlosti u některých operací. Jednoduše shrnuto, MySQL je malý, rychlý, jednoduchý a nenáročný databázový systém.

MySQL patří mezi databázové systémy, které vyvolávají mohutné diskuse pod články. Někteří lidé MySQL zatracují, jiní ji zase mohutně chválí. Pravdou je, že MySQL nemá takové schopnosti a možnosti jako mnohé velké databázové systémy, ale MySQL do této oblasti ani nemíří. MySQL je projektován pro jednoduché databáze, byť třeba s obrovskou spoustou údajů. Každopádně MySQL se prosadila především do oblasti malých webových databází, kde jí budeme používat i my.

Další díly seriálu pak zaměřím pouze na MySQL databázový systém.

Co je to databáze?

Databázi si můžeme představit jako prostor, do kterého se ukládají všechny potřebné údaje. Zpracováním a přístupem údajů v databázi bývá pověřen program, kterému se říká DBMS (DataBase Management System), nebo česky SRBD (Systém Řízení Báze Dat). Celé je to tedy zařízeno tak, že veškeré zpracování, ukládání a získávání dat je prováděno DBMS programem. Jakákoli aplikace, která databázi používá vždy přistupuje přes tento DBMS program.

Mezi DBMS patří řada programů v nejrůznějších cenových kategoriích. Většina profesionálních a komerčních DBMS nejsou programy nikterak levné a jejich cena se pohybuje ve velmi vysokých částkách. Naštěstí existují v kategorii DBMS i programy, které jsou v nízké cenové relaci, nebo zcela zdarma. Představitelem těchto DBMS systémů jsou například MySQL, PostgreSQL, nebo Firebird.

Naprostá většina dnešních DBMS staví na tzv. relačním modelu dat. V tomto modelu jsou veškerá data uspořádána do databázových tabulek. Každá tabulka zpravidla uspořádává údaje o určitém objektu. Databázová tabulka je uspořádána do řádků a sloupců. Například budeme-li mít tabulku se seznamem osob, bude každý řádek odpovídat jedné osobě. Sloupce pak obsahují konkrétní údaje o osobách, v našem případě by to třeba mohlo být: Jméno, Rodné číslo, Adresa, Telefon.

Co je to SQL?

Jak už jsem zmínil, přístup k údajům uloženým v databázi obstarává DBMS program. Aby mohly jiné aplikace používat také údaje z databáze, musí DBMS program nabízet rozhraní, pomocí kterého budou moci ostatní aplikace s DBMS spolupracovat.

Při práci s DBMS programem se používá osvědčený model klient/server. V roli serveru vystupuje DBMS program nabízející své služby. DBMS program je tak nepřetržitě spuštěn a čeká na požadavky od klientů - jiných aplikací. Na tyto požadavky pak odpovídá a reaguje na ně. Protože DBMS program vystupuje v roli serveru, říká se mu také databázový server.

Pro zápis požadavků na databázový server se dnes nejčastěji používá SQL jazyk. SQL je anglická zkratka znamenající Structured Query Language. Tento jazyk dnes používá naprostá většina databázových serverů. Proto se takovým serverům říká zjednodušeně SQL servery. S SQL jazykem se postupně seznámíme, protože i v PHP zapisujeme požadavky pomocí tohoto jazyka. Samotný SQL obsahuje vše potřebné pro práci s databázemi - ať už příkazy pro práci na vytvoření, rušení, modifikování tabulky, tak i pro práci s údaji samotnými - přidávání, změnu, rušení a vyhledávání údajů.

Jazyk SQL má za sebou poměrně dlouhý vývoj. Jeho prototyp by poprvé implementován v roce 1974 společností IBM. Od té doby má za sebou SQL bouřlivý vývoj, a dá se říci, že jazyk SQL se dodnes stále vyvíjí, hlavně co se týká přidávání nových technologií. Skutečným faktickým standardem se stal SQL až v roce 1986, kdy se dočkal standardizace díky organizaci ANSI. Od té doby se tedy SQL stal univerzálním nástrojem pro programování databází. Dnes se každý, kdo se dostává do styku s vývojem, či laděním databází, musí seznámit s jazykem SQL, který je prakticky u databází všudypřítomný.

Pomocí jazyka SQL tedy říkáme, co chceme v databázi provést. Jazyk SQL se skládá ze dvou základních částí, jednak z části DDL, a jednak z části DML. Příkazy DDL jsou zkratkou za Data Definition Language, a slouží k definici a modifikaci tabulek a dalších databázových objektů. Příkazy DML jsou zase zkratkou za Data Manipulation Language a slouží již opravdu pro manipulaci s databázovými údaji.

Jak to vše souvisí s PHP?

V roli klienta pro databázový server může jako aplikace vystupovat i PHP. Tedy PHP se může spojit s databázovým serverem a využívat jeho služby, požadovat zápis, nebo získání určitých údajů z databáze. To znamená, že naše PHP skripty mohou obsahovat příkazy zapsané v jazyce SQL a PHP může zpracovávat výsledky SQL po provedení na databázovém serveru.

V praxi má každý SQL server svůj vlastní protokol, a proto PHP musí umět s každým SQL serverem komunikovat trochu jinak. Ale jak už jsem napsal, PHP umí komunikovat s mnoha SQL servery, mimo jiné třeba s Oracle, Sybase, Informix, MySQL, PostgreSQL. Pro účely našeho seriálu se však omezíme na MySQL.

Dnešní díl byl velice krátkým úvodem do databází. V příštích dílech se postupně začneme zabývat SQL jazykem samotným.

PHP - 26. díl – spouštíme MySQL

Dnes se budeme věnovat problému spuštění MySQL a navazujícím oblastem. Samo o sobě je spuštění MySQL velmi jednoduché. Já se budu věnovat spuštění MySQL pouze na intranetovém serveru, který jsem uvedl ve druhém díle seriálu. Ve druhém díle je odkaz, kde si intranetový server můžete stáhnout.

Jak spouštět MySQL na intranetovém serveru

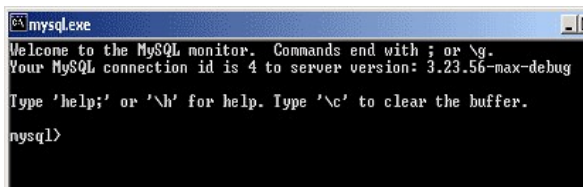
Pokud máte nainstalovaný intranetový server, pak je spuštění vlastní MySQL velmi jednoduché. Stačí jít do startovacího menu ve Windows, dále do složky Programy -> Intranetový server -> MySQL databáze. Pokud spustíte MySQL, pak krátce problikne DOSové okno, které hned zmizí. Tím je spuštěna MySQL v pozadí, možná vás bude mást, že nikde nic nevidíte, ani na liště, ani žádné okno. Pouze ve správci úloh můžete najít proces s názvem mysqld, který odpovídá spuštěné MySQL. Každopádně je ale MySQL spuštěna, a jako DBMS program čeká v pozadí na příkazy a požadavky na databázi.

Pokud budete chtít MySQL ukončit, pak stačí jít do startovací menu, dále do Programy -> Intranetový server -> Ukonči MySQL databázi. Pak se MySQL korektně ukončí. Měli byste takto ukončovat MySQL před vypnutím vždy.

Pokud nám běží spuštěná MySQL, běží nám v podstatě serverová část databáze. Nyní se může pokusit připojit jednoduchým řádkovým klientem, který se jmenuje mysql.

Jednoduchý řádkový klient mysql

Jednoduchý řádkový klient mysql je součástí balíku MySQL. V intranetovém serveru na něj není žádná ikonka ve startovacím menu, ani na ploše, je potřeba přímo spustit z jeho cesty zde: C:\inet_srv\mysql\bin\mysql.exe. Pokud jej přímo spustíte, objeví se vám okno:



```
mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 3.23.56-max-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Toto okno se vám objeví pouze tehdy, pokud je spuštěná i serverová část. Pak to znamená, že se podařilo spojit se serverovou částí a čeká se na zadání příkazů. Pokud není spuštěna serverová část, pak klient pouze pípne a ukončí se.

Základním příkazem je exit, který ukončuje práci s řádkovým klientem. Všechny příkazy můžete vidět po zadání příkazu help. Kromě toho je možné zadávat i příkazy pro databázi v jazyce SQL.

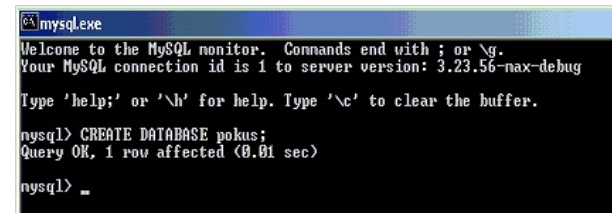
První příkaz v jazyce SQL - vytvoření nové databáze

Na začátku ještě nemáme vytvořenou žádnou databázi. Samotná MySQL dokáže pracovat s mnoha databázemi, tedy úložišti dat, naráz. Každá databáze má své jméno, je dobré vybírat jména bez háčků a čárek, protože ne každý MySQL server si s nimi poradí.

Vytvoříme tedy pokusnou databázi s názvem pokus. K tomu slouží SQL příkaz:

```
CREATE DATABASE pokus;
```

Tento SQL příkaz zadáme do řádkového klienta a potvrdíme klávesou Enter:



```
mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 3.23.56-max-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE pokus;
Query OK, 1 row affected (0.01 sec)

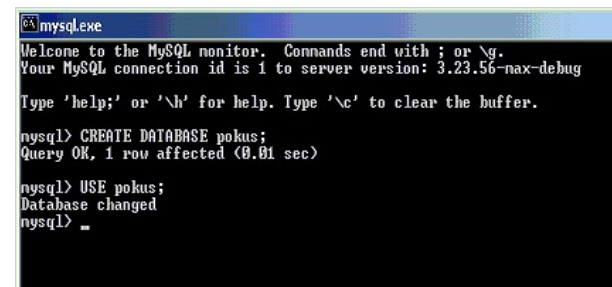
mysql> _
```

Databáze pokus je tedy vytvořena.

Po spuštění řádkového klienta mysql je potřeba nastavit, s jakou databází má pracovat. Používá se na to SQL příkaz:

```
USE pokus;
```

Tímto příkazem říkáme řádkovému klientovi, že od teď má používat databázi s názvem pokus.



```
mysql.exe
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 3.23.56-max-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE pokus;
Query OK, 1 row affected (0.01 sec)

mysql> USE pokus;
Database changed
mysql> _
```

Pokračujeme v SQL - vytvoření databázové tabulky

Nyní již máme vytvořenou prázdnou databázi s názvem pokus, ale abychom mohli ukládat data, potřebujeme mít alespoň jednu databázovou tabulku. Pro vytvoření tabulky se používá příkaz:

```
CREATE TABLE název_tabulky (jméno_sloupce typ_sloupce, ...);
```

Nejpoužívanější typy sloupce jsou **int** pro celé číslo a **varchar(n)** pro textový řetězec o maximální délce n znaků.

Vytvoříme například tabulku se seznamem osob, která bude obsahovat následující sloupce: Jméno, Rodné číslo, Adresa, Telefon.


```
CREATE TABLE Osoby(
  Jmeno varchar(30),
  RodneCislo varchar(11) NOT NULL,
  Adresa varchar(50),
  Telefon varchar(12),
  PRIMARY KEY (RodneCislo)
);
```

Názvy tabulek a sloupců je dobré zadávat bez háčeků a čárek, protože ne každý MySQL server si s diakritickými znaménky poradí.

Výše uvedeným SQL příkazem vytvářím tabulku s názvem Osoby, která obsahuje 4 sloupce, všechny obsahují různé dlouhé textové řetězce. Sloupec s rodným číslem má atribut NOT NULL, který říká, že rodné číslo nemůže obsahovat prázdnou hodnotu. To je nutné, protože rodné číslo jsem použil jako tzv. primární klíč tabulky. Primární klíč znamená sloupec, který má pro každý řádek jinou hodnotu. Primární klíč se definuje pomocí slov PRIMARY KEY.

Přidání řádku do tabulky

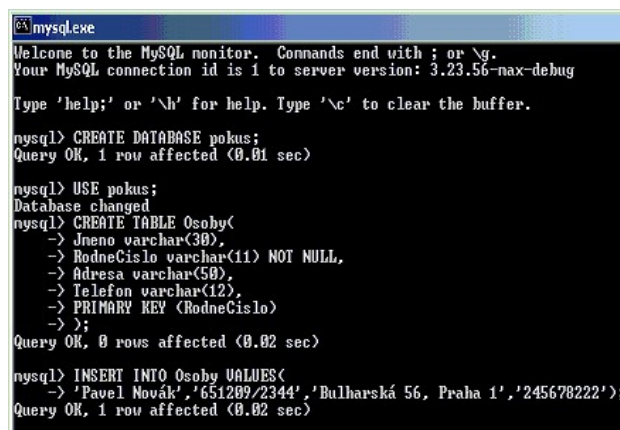
Nyní již máme databázovou tabulku s názvem Osoby, a potřebujeme přidávat nové řádky do tabulky. K tomu slouží SQL příkaz INSERT:

```
INSERT INTO název_tabulky VALUES(hodnota1, hodnota2, ... hodnotaN);
```

Pro přidání pana Nováka do tabulky Osoby můžeme použít například SQL příkaz:

```
INSERT INTO Osoby VALUES(
  'Pavel Novák', '651209/2344', 'Bulharská 56, Praha 1', '245678222');
```

Jak to dopadne při zadání příkazu do řádkového klienta se můžeme podívat zde:



```
mysql> Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 3.23.56-max-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> CREATE DATABASE pokus;
Query OK, 1 row affected (0.01 sec)

mysql> USE pokus;
Database changed
mysql> CREATE TABLE Osoby(
  -> Jmeno varchar(30),
  -> RodneCislo varchar(11) NOT NULL,
  -> Adresa varchar(50),
  -> Telefon varchar(12),
  -> PRIMARY KEY (RodneCislo)
  -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO Osoby VALUES(
  -> 'Pavel Novák', '651209/2344', 'Bulharská 56, Praha 1', '245678222');
Query OK, 1 row affected (0.02 sec)
```

Vidíme, že řádkový klient nám píše v angličtině, že byl přidán jeden řádek, a že to trvalo dvě setiny sekundy.

Podobným způsobem je možné přidat i další osoby. Je samozřejmé, že tento způsob zadávání dat není zrovna příjemný, ale nakonec jakékoli zadávání dat skončí uvnitř programu na podobných SQL příkazech.

Prohlížení zadaných řádků

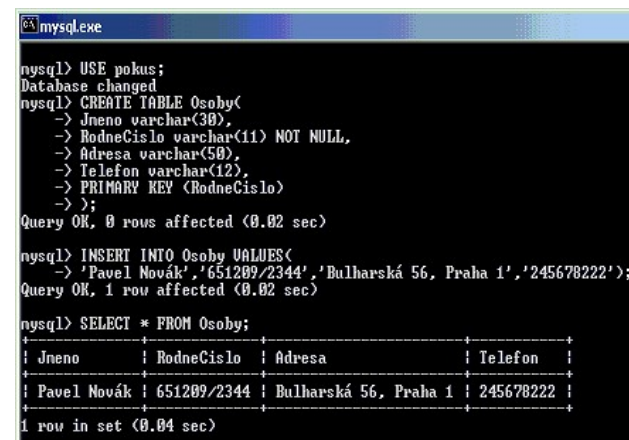
Pro získávání už zadaných dat z databázových tabulek slouží SQL příkaz SELECT. Má mnoho podob a tváří, ale pro ten nejjednodušší případ, kdy chceme prostě vidět všechno v tabulce má velmi jednoduchý tvar:

```
SELECT * FROM název_tabulky;
```

Pro prohlížení řádků zadaných v tabulce Osoby prostě napíšeme:

```
SELECT * FROM Osoby;
```

A takto to dopadne, pokud zadáme SQL příkaz do řádkového klienta:



```
mysql> USE pokus;
Database changed
mysql> CREATE TABLE Osoby(
  -> Jmeno varchar(30),
  -> RodneCislo varchar(11) NOT NULL,
  -> Adresa varchar(50),
  -> Telefon varchar(12),
  -> PRIMARY KEY (RodneCislo)
  -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO Osoby VALUES(
  -> 'Pavel Novák', '651209/2344', 'Bulharská 56, Praha 1', '245678222');
Query OK, 1 row affected (0.02 sec)

mysql> SELECT * FROM Osoby;
+-----+-----+-----+-----+
| Jmeno   | RodneCislo | Adresa                | Telefon |
+-----+-----+-----+-----+
| Pavel Novák | 651209/2344 | Bulharská 56, Praha 1 | 245678222 |
+-----+-----+-----+-----+
1 row in set (0.04 sec)
```

PHP - 27. díl – začlenění SQL příkazů do PHP

V dnešním dílu našeho seriálu si povíme více o reálném nasazení SQL.

Jak probíhá komunikace PHP s MySQL?

Jak už jsme napsali dříve, chceme-li se připojit k MySQL databázi pomocí PHP skriptů, stává se PHP klientem databáze. Každý databázový klient provádí stejné posloupnosti příkazů, pokud se chce pracovat s databázovým serverem.

Základní věc, kterou takové PHP musí udělat, pokud chce pracovat s databází, je nejprve se k databázi připojit. Po připojení může komunikovat s databázovým serverem, posílat mu SQL příkazy, přebírat získaná data, apod.. Až PHP provede veškerou práci s databází, provede se odpojení od databáze.

Takže komunikace PHP s MySQL probíhá v tom duchu, že nejdříve se k databázi připojí, pak pošle sérii SQL dotazů, přebere si případná výsledná data a nakonec se odpojí.

Připojení k MySQL v PHP (a odpojení se)

Pro připojení k MySQL slouží v PHP funkce **mysql_connect**. Tato funkce se obvykle používá se třemi parametry:

```
mysql_connect ( adresa_serveru, uživatelské_jméno,
uživatelské_heslo )
```

Parametr adresa serveru je adresa databázového serveru MySQL. Pokud zkoušíte databázi na lokálním počítači, jako je to v případě, kdy používáte na zkoušení intranetový server ze 2. dílu seriálu, pak adresa serveru je řetězec "localhost".

Druhý a třetí parametr je uživatelské jméno a heslo k databázovému serveru. Pokud používáte na zkoušení intranetový server, pak uživatelské jméno je "root" a uživatelské heslo je prázdné, tedy "".

Samotná funkce mysql_connect vám vrací k dispozici tzv. identifikátor spojení, který se bude hodit k posílání příkazů do MySQL. Pokud se spojení nezdařilo, funkce vrátí hodnotu false, takže můžete otestovat v podmínce, jestli spojení s databází proběhlo v pořádku.

Zde je příklad, který se pokusí připojit k databázi a hned se zase odpojí:

```
<html>
<head>
<title>Příklad 1. z 27. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<?php
$id_spojeni = mysql_connect(`localhost`,`root`,``);
if ($id_spojeni)
echo `Spojení s MySQL databází se podařilo.`;
else
echo `Spojení s MySQL databází se nezdařilo.`;
if ($id_spojeni)
mysql_close($id_spojeni);
?>
</body>
</html>
```

Pokud zkoušíte tento, nebo další příklady na intranetovém serveru ze 2. dílu, pak musíte mít kromě webového serveru spuštěnu i MySQL databázi.

Pokud zkoušíte tento příklad na něčem jiném, musíte příslušně upravit parametry za voláním funkce mysql_connect.

Pokud se spojení s MySQL databází podaří, PHP skript toto vypíše. Pokud se spojení s MySQL nepodaří, tak PHP skript vypíše příslušné hlášení, ale kromě toho automaticky naskočí i další anglická varování, která PHP skript píše, kdykoli se mu něco nelíbí.

K odpojení se od databáze slouží funkce **mysql_close**, která byla už použita v příkladu. Má jediný parametr, a to identifikátor spojení:

```
mysql_close ( identifikátor_spojění )
```

Pokud nepoužijeme funkci mysql_close, nebo jí zapomeneme použít, provede se odpojení od databáze automaticky při ukončení PHP skriptu.

Vybrání databáze

Protože v předchozím díle jsme vytvořili databázi s názvem pokus, musíme si jí vybrat, abychom právě s touto databází mohli pracovat. Jak už jsem v předchozím díle psal, každý databázový server může spravovat mnoho databází. Proto je dobré hned po připojení se k databázi vybrat, se kterou databází si přejeme pracovat. K tomu slouží funkce **mysql_select_db**.

Funkci mysql_select_db je dobré používat se dvěma parametry:

```
mysql_select_db ( název_databáze, identifikátor_spojění )
```

Pokud bychom měli náš předchozí příklad rozšířit o vybrání databázi pokus, mohlo by to dopadnout třeba takto:

```
<html>
<head>
<title>Příklad 2. z 27. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<?php
$id_spojeni = mysql_connect(`localhost`,`root`,``);
if (!$id_spojeni)
die(`Spojení s MySQL databází se nezdařilo.`);
$vysedek_vybrani = mysql_select_db(`pokus`,`$id_spojeni`);
if ($vysedek_vybrani)
echo `Úspěšně jsme vybrali databázi pokus.`;
else
echo `Databázi pokus se nám nepodařilo vybrat.`;
mysql_close($id_spojeni);
```

```
?>
</body>
</html>
```

Pro úspěšné proběhnutí příkladu je potřeba, abyste předtím provedli kroky z minulého, 26. dílu seriálu. Pokud se totiž nepodaří databázi pokus vybrat, znamená to, že databáze pokus neexistuje. A přitom databázi pokus jsme založili právě v předchozím díle.

Poslání SQL příkazu

Protože v [minulém díle](#) jsme se naučili pár jednoduchých SQL příkazů, a mimo jiné také založili databázovou tabulku Osoby, zkusíme toho využít a poslat SQL příkaz přímo do databáze. Použijeme proto příkaz pro zobrazení dat z tabulky Osoby:

```
SELECT * FROM Osoby;
```

K zaslání SQL příkazu můžeme použít funkci **mysql_query**. Tuto funkci je dobré používat se dvěma parametry:

```
mysql_query ( sql_příkaz, identifikátor_spojení )
```

Funkce mysql_query vrací buď identifikátor výsledku, pokud SQL příkaz vrací nějaká data, nebo pouze hodnotu true, pokud SQL dotaz nevrací žádná data. Při chybě vrací funkce mysql_query vždy hodnotu false.

Rozšířit tedy předchozí příklad o volání SQL dotazu není nijak těžké:

```
<html>
<head>
<title>Příklad 3. z 27. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<?php
$id_spojeni = mysql_connect(`localhost`,`root`,``);
if (!$id_spojeni)
    die(`Spojení s MySQL databází se nezdařilo.`);

$vysledek_vybrani = mysql_select_db(`pokus`,`$id_spojeni`);
if (!$vysledek_vybrani)
    die(`Databázi pokus se nám nepodařilo vybrat.`);

$id_vysledku = mysql_query(`SELECT * FROM Osoby`,`$id_spojeni`);
if (!$id_vysledku)
    die(`Nepodařilo se nám poslat SQL dotaz do databáze.`);

echo `Dotaz byl poslán do databáze.`;

mysql_close($id_spojeni);
```

```
?>
</body>
</html>
```

Celý tento příklad má jeden háček. Poslali jsme SQL příkaz, který nám má vrátit všechna data z databázové tabulky Osoby. V předchozím příkladu jsme úspěšně poslali SQL příkaz do databáze, ale nezískali jsme data z tabulky Osoby. Zkusíme tedy pokračovat.

Čtení výsledků SQL dotazu

Pro čtení výsledků SQL dotazu existují dvě šikovné funkce, a to **mysql_fetch_row** a **mysql_fetch_array**. Obě se nejčastěji používají s jedním parametrem:

```
mysql_fetch_row ( identifikátor_výsledku )
```

```
mysql_fetch_array ( identifikátor_výsledku )
```

Obě funkce načtou další řádek z výsledků SQL dotazu a vrací buď pole s hodnotami, a nebo false, pokud bylo už všechno přečteno. Liší se od sebe jen tím, že mysql_fetch_row načte hodnoty do pole s číselnými indexy, zatímco funkce mysql_fetch_array přidá i indexy s názvy sloupců.

Aby vše bylo jasnější, zkusme si přečíst a zobrazit výsledek SQL příkazu z minulého příkladu:

```
<html>
<head>
<title>Příklad 4. z 27. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<?php
$id_spojeni = mysql_connect(`localhost`,`root`,``);
if (!$id_spojeni)
    die(`Spojení s MySQL databází se nezdařilo.`);

$vysledek_vybrani = mysql_select_db(`pokus`,`$id_spojeni`);
if (!$vysledek_vybrani)
    die(`Databázi pokus se nám nepodařilo vybrat.`);

$id_vysledku = mysql_query(`SELECT * FROM Osoby`,`$id_spojeni`);
if (!$id_vysledku)
    die(`Nepodařilo se nám poslat SQL dotaz do databáze.`);
?>
<table border="1">
<tr>
<th>Jméno</th>
<th>Rodné číslo</th>
<th>Adresa</th>
```

```

<th>Telefon</th>
</tr>
<?php
while($radek = mysql_fetch_row($id_vysledku))
{
    for ($i=0; $i<4; ++$i)
        echo `<td>`, $radek[$i], `</td>`;
}
?>
</table>
<?php
mysql_close($id_spojeni);
?>
</body>
</html>

```

To je z dnešního článku vše.

PHP - 28. díl – vložení nového řádku pomocí PHP

28. díl seriálu o PHP bude věnován tomu, jak vkládat nové řádky do databáze pomocí PHP

Databázové funkce v PHP jsou přímo ideální pro použití formulářů. Zejména zadávání nových hodnot, které se ukládají do databáze, po využití formulářů přímo volají. Jeden skript slouží k zadání nových hodnot a druhý skript slouží k uložení zadaných dat. Pro další text budu předpokládat, že máme databázi pokus a databázovou tabulku Osoby, kterou jsme vytvořili ve 26. díle tohoto seriálu.

Formulář k zadání nové osoby

Nejdříve vytvoříme formulář, který bude sloužit k zadání nové osoby. Tento formulář pak později napojíme na skript, který provede uložení nové osoby do databáze.

```

<html>
<head>
<title>Příklad 1. z 28. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<form action="uloz_radek.php" method="post">
<table>
<tr>
<td>Jméno:</td>
<td><input type="text" name="Jmeno"></td>
</tr>
<tr>

```

```

<td>Rodné číslo:</td>
<td><input type="text" name="RodneCislo"></td>
</tr>
<tr>
<td>Adresa:</td>
<td><input type="text" name="Adresa"></td>
</tr>
<tr>
<td>Telefon:</td>
<td><input type="text" name="Telefon"></td>
</tr>
<tr>
<th colspan="2"><input type="submit" value="Přidání
osoby"></th>
</tr>
<tr>
</form>
</body>
</html>

```

Tento skript je v podstatě čistý HTML kód, který obsahuje pole pro zadání nové osoby a tlačítko pro odeslání údajů skriptu s názvem uloz_radek.php. PHP skript s názvem uloz_radek.php napíšeme později.

PHP skript k uložení řádku

Vlastní PHP skript, který přijímá data od předchozího formuláře a uloží řádek, je zde (uložte jej do čistého txt souboru s názvem uloz_radek.php):

```

<html>
<head>
<title>Příklad 2. z 28. dílu</title>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1250">
</head>
<body>
<?php
$id_spojeni = mysql_connect(`localhost`,`root`,``);
if (!$id_spojeni)
    die(`Spojení s MySQL databází se nezdařilo.`);

$vysledek_vybrani = mysql_select_db(`pokus`,`$id_spojeni`);
if (!$vysledek_vybrani)
    die(`Databázi pokus se nám nepodařilo vybrat.`);

$Jmeno = $_POST[`Jmeno`];
$RodneCislo = $_POST[`RodneCislo`];
$Adresa = $_POST[`Adresa`];
$Telefon = $_POST[`Telefon`];

```

```

$sql = "INSERT INTO Osoby
VALUES(`$Jmeno`,`$RodneCislo`,`$Adresa`,`$Telefon`)";
$vysledek = mysql_query($sql,$id_spojeni);
if (!$vysledek)
    die(`Nepodařilo se vložit nový řádek.`);

$sql = "SELECT * FROM Osoby";
$id_vysledku = mysql_query($sql,$id_spojeni);
if (!$id_vysledku)
    die(`Nepodařilo se nám načíst řádky z databáze.`);
?>
<table border="1">
<tr>
<th>Jméno</th>
<th>Rodné číslo</th>
<th>Adresa</th>
<th>Telefon</th>
</tr>
<?php
while($radek = mysql_fetch_row($id_vysledku))
{
    echo `<tr>`;
    for ($i=0; $i<4; ++$i)
        echo `<td>`, $radek[$i], `</td>`;
    echo `</tr>`;
}
?>
</table>
<?php
mysql_close($id_spojeni);
?>
</body>
</html>

```

Samotný skript na uložení řádku už je docela složitý, ale je to tím, že kromě ukládání řádků ještě vypisuje, co už je všechno uloženo. Pokud budete přidávání řádku zkoušet, buďte si vědomi toho, že databázová tabulka "Osoby" požaduje jedinečné rodné číslo. Budete-li chtít uložit osobu s rodným číslem, které už je v databázi, vložení nového řádku se nepodaří.

Jak vlastně skript na přidání nového řádku funguje? V podstatě docela jednoduše. Na začátku jsou nejdříve povinné tanečky, tedy připojení se k databázovému serveru MySQL a poté vybrání databáze s názvem "Pokus". Potom přečteme všechny údaje předané z formuláře pomocí pole \$ POST. V dalším kroku vytvoříme z těchto údajů SQL příkaz INSERT. A pomocí mysql_query jej pošleme do databáze. Protože SQL příkaz INSERT nevrací žádná data, použijeme návratovou hodnotu funkce mysql_query pouze k otestování, zda příkaz proběhl správně. Následuje výpis všech řádků z tabulky Osoby.

Ačkoli uvnitř skriptů používáme SQL příkazy, navenek jsme vytvořili poměrně příjemnou aplikaci k zadávání nových řádků. Takto pracuje naprostá většina databázových aplikací. Ačkoli uvnitř se pracuje s SQL příkazy, které mohou být pro uživatele složité, navenek jsou skryty a uživatel se s nimi nesetkává.

PHP - 29. díl – použití SQL příkazu SELECT

Dnešní díl našeho pravidelného týdenního seriálu bude věnován jednomu z nejdůležitějších příkazů při práci s databázemi.

Úvod

SQL jazyk je základní jazyk pro práci s databázemi. Bez jeho základní znalosti prakticky nemůžeme v PHP s databázemi vůbec pracovat. Proto budu věnovat několik dalších dílů základům jazyka SQL ve verzi pro MySQL. Určité nejjednodušší SQL příkazy jsem uvedl ve 26. díle seriálu, a tyto příkazy právě rozšíříme. Protože je tak důležité umět pracovat s SQL příkazy, budu po několik dílů rozebírat SQL příkazy ve verzi pro MySQL podrobněji.

Co je phpMyAdmin?

Ovládat databázi MySQL a zadávat SQL příkazy lze několika způsoby. Můžete využít například řádkového klienta jako jsem to udělal ve 26. díle seriálu. Jinou možností je použít projekt phpMyAdmin, který je vlastně souborem PHP skriptů. Aplikaci phpMyAdmin nabízí i mnoho poskytovatelů webového prostoru. Protože se v diskusi objevila poznámka k aplikaci phpMyAdmin, rozhodl jsem se, že dám čtenářům možnost vyzkoušet si i tuto aplikaci. Můžete si proto stáhnout na [této adrese](#) novou verzi intranetového serveru s přidanou aplikací phpMyAdmin. Vše funguje naprosto stejně jako intranetový server z 2. dílu seriálu, pouze je přidána aplikace phpMyAdmin. Kdo chce, může si nový intranetový server nainstalovat a vyzkoušet si ovládání MySQL pomocí phpMyAdmin.

SQL příkaz SELECT

Po předchozím díle je databázová tabulka Osoby alespoň částečně zaplněna údaji o několika osobách. Nyní bychom potřebovali v datech vybírat, listovat jimi, zkrátka se nějak lépe v datech pohybovat. K vybírání dat podle mnoha kritérií slouží právě SQL příkaz SELECT. Ve 26. díle jsem uvedl jeho nejjednodušší možnou podobu, ale ta v mnoha případech nestačí. Proto zde uvedu téměř úplnou podobu SQL příkazu SELECT:

```

SELECT výraz FROM seznam_tabulek WHERE podmínka GROUP BY
seskupení_sloupců HAVING podmínka ORDER BY třídící_kritérium
LIMIT limit_počtu_řádků

```

Vypadá to dost složitě, ale to je jenom zdání. Postupně rozklíčujeme jednotlivé části SQL příkazu SELECT. Ve valné většině případů se v příkazu SELECT nebudou vyskytovat všechny části. Následující příkazy typu SELECT můžete zkoušet například v řádkovém klientu pro MySQL jako jsme to dělali ve 26. díle seriálu, a nebo třeba zkusit použít phpMyAdmin.

Nejjednodušší SELECT příkaz je v tomto tvaru:

```
SELECT * FROM Osoby
```

Tento SQL příkaz vybere všechny řádky z databázové tabulky s názvem Osoby. Hvězdička za slovem SELECT znamená, že vybíráme i všechny sloupce z této tabulky. Namísto hvězdičky můžeme napsat seznam sloupců, které nás zajímají, například:

```
SELECT Jmeno, Telefon FROM Osoby
```

V tomto případě nás zajímá jméno a telefon z tabulky Osoby. Ostatní sloupce nás nezajímají a ve výsledných datech ani nebudou přítomny.

Setřídění výsledných dat - klauzule ORDER BY

Výsledná data mohou být setříděna podle nejrůznějších kritérií. Například takto dostaneme seznam osob se jménem, rodným číslem a telefonem setříděné podle telefonního čísla:

```
SELECT Jmeno, RodneCislo, Telefon FROM Osoby ORDER BY Telefon
```

Nyní je tedy jasné, že za klauzulí ORDER BY uvádíme podle čeho mají být data setříděna. Mohou být setříděna vzestupně i sestupně. Pokud chceme třídit sestupně, jednoduše uvedeme za sloupec DESC:

```
SELECT Jmeno, RodneCislo, Telefon FROM Osoby ORDER BY Telefon  
DESC
```

Jde třídit i podle více sloupců. Například takto vyberu všechny sloupce z tabulky Osoby a setřídím je podle jména. Pokud se vyskytnou osoby se stejným jménem, budou dále setříděny podle rodného čísla:

```
SELECT * FROM Osoby ORDER BY Jmeno, RodneCislo
```

Limit počtu řádků - klauzule LIMIT

Představte si, že byste v tabulce Osoby měli obrovský počet osob. Například několik miliónů lidí. Pak pokud byste zkoušeli výše uvedené příkazy SELECT, určitě je hodně nepraktické zpracovávat všechny osoby naráz a určitě je nevhodné je třeba všechny zobrazovat na jedné HTML stránce. Proto existuje klauzule LIMIT, která omezí počet vrácených řádků z databáze. Například následující příkaz vrátí prvních maximálně 10 osob z tabulky Osoby:

```
SELECT * FROM Osoby LIMIT 10
```

Klauzule LIMIT nám také umožňuje vrátit řádky ne od začátku. Například následující příkaz vrátí řádky od desáté osoby a omezí počet osob na maximálně deset:

```
SELECT Jmeno, RodneCislo FROM Osoby LIMIT 9, 10
```

Pokud chceme vrátit řádky od desáté osoby, musíme v klauzuli LIMIT použít číslo 9, protože řádky se číslují v MySQL od nuly. Tedy první řádek má číslo 0, druhý řádek má číslo 1, třetí řádek má číslo 2, atd.

Pomocí klauzule LIMIT snadno zařídíme v PHP skriptech listování, kdy třeba na první stránce bude prvních 20 osob, na druhé stránce bude dalších 20 osob, a tak dále. Prostě pro první stránku použijeme SQL příkaz:

```
SELECT * FROM Osoby LIMIT 0, 20
```

Pro druhou stránku pak použijeme příkaz:

```
SELECT * FROM Osoby LIMIT 20, 20
```

A tak dále.

Výběr řádků - klauzule WHERE

Do této chvíle jsme vraceli celý obsah tabulky Osoby maximálně omezený klauzulí LIMIT. Ale často se stává, že chceme jenom určité řádky. Například v našem případě můžeme chtít osoby, jejichž telefonní číslo začíná dvojkou. Nebo třeba hledáme osobu s určitým rodným číslem. Pro tyto případy slouží právě klauzule WHERE.

Například následující SQL příkaz najde osobu s daným rodným číslem:

```
SELECT * FROM Osoby WHERE RodneCislo = `651209/2344`
```

A tento příkaz najde osobu s daným telefonem:

```
SELECT * FROM Osoby WHERE Telefon = `245678222`
```

Následující příkaz vybere všechna jména, která začínají na Pavel:

```
SELECT * FROM Osoby WHERE Jmeno LIKE `Pavel%`
```

Operátor LIKE slouží ke hledání podle vzoru. Znak % má ve vzoru speciální význam, nahrazuje libovolnou sekvenci znaků. Tedy vzor `Pavel%` znamená, že hledáme vše, co začíná slovem Pavel.

Operátor LIKE můžeme stejně dobře využít ke hledání třeba všech telefonních čísel začínajících na dvojku, tedy pražských telefonních čísel:

```
SELECT * FROM Osoby WHERE Telefon LIKE `2%`
```

V podmínkách pro klauzuli WHERE můžeme využít i logické spojky AND (= a zároveň) a OR (= nebo). Například následující SQL příkaz najde všechny Pavly s pražským telefonním číslem:

```
SELECT * FROM Osoby WHERE Telefon LIKE `2%` AND Jmeno LIKE  
`Pavel%`
```

Je samozřejmé, že klauzuli WHERE můžeme kombinovat i s dalšími klauzulemi, které jsme si už ukázali. Například následující příkaz ukáže všechny Nováky seřazené podle jména:

```
SELECT * FROM Osoby WHERE Jmeno LIKE `%Novák%` ORDER BY Jmeno
```

PHP - 30. díl – SQL příkazy INSERT a DELETE

30. díl seriálu o PHP bude věnován představení dvou důležitých SQL příkazů.

Úvod

Dnešní díl bude věnován dvou protikladným SQL příkazům, a to příkazu INSERT na přidávání celých nových řádků a příkazu DELETE na mazání celých řádků.

SQL příkaz INSERT

SQL příkaz INSERT slouží k přidávání nových řádků do databáze. Samotný příkaz INSERT není příliš složitý a patří k těm jednodušším příkazům. Téměř úplná podoba nejběžnější formy příkazu INSERT je následující:

```
INSERT INTO tabulka (seznam_sloupců) VALUES (seznam_hodnot)
```

Nejjednodušší možná forma příkazu INSERT je naplnění všech sloupců v tabulce hodnotami přesně v tom pořadí, v jakém jsou sloupce definovány při vytvoření tabulky. Například v tabulce Osoby, kterou jsme používali v předchozích dílech jsou sloupce definovány v pořadí: Jmeno, RodneCislo, Adresa, Telefon. Pak by INSERT příkaz mohl vypadat třeba takto:

```
INSERT INTO Osoby
VALUES (`Jan Novotný`, `701212/1234`, `Novohradská
5`, `303456789`)
```

Tento SQL příkaz přidá jeden řádek do tabulky Osoby. V mnoha případech je ale nepraktické předpokládat, že sloupce budou přesně v tom pořadí, v jakém jsou definovány v tabulce. Proto je mnohem lepší, budeme-li do každého příkazu INSERT přidávat seznam sloupců. Například stejný příkaz, jako je výše lze se seznamem sloupců napsat takto:

```
INSERT INTO Osoby (Jmeno, RodneCislo, Adresa, Telefon)
VALUES (`Jan Novotný`, `701212/1234`, `Novohradská
5`, `303456789`)
```

Je lepší přidávat seznam sloupců ke každému příkazu INSERT a navíc tím získáme tu výhodu, že nemusíme zadávat všechny sloupce v tabulce, pokud je nechceme všechny vyplnit. Například pokud chceme vyplnit pouze sloupce Jmeno a RodneCislo, můžeme napsat takovýto příkaz:

```
INSERT INTO Osoby (Jmeno, RodneCislo) VALUES (`Pavel
Novák`, `600512/7894`)
```

Přitom je nutné podotknout, že ne každý sloupec lze vynechat. Některé sloupce musíme povinně vyplnit, například v našem případě musíme vždy vyplnit sloupec s rodným číslem, jinak nám velmi brzy začnou příkazy INSERT selhávat.

Jedním příkazem INSERT lze najednou přidat více řádků. Například pro přidání dvou řádků můžeme použít následující příkaz:

```
INSERT INTO Osoby (Jmeno, RodneCislo)
VALUES (`Alena Válová`, `555612/7845`), (`Aleš
Blamák`, `881102/5621`)
```

Tímto příkazem je možné najednou přidat libovolné množství řádků, stačí prostě hodnoty v závorkách oddělovat čárkou.

SQL příkaz DELETE

Příkaz DELETE slouží k mazání celých řádků z databázových tabulek. Jedná se o poměrně nebezpečný příkaz, kterým můžeme přijít velmi rychle o data.

Nejjednodušší tvar příkazu DELETE slouží k vymazání všech údajů v databázové tabulce. Pokud byste spustili následující příkaz, budete mít tabulku Osoby naprosto čistou bez jakýchkoli dat:

```
DELETE FROM Osoby
```

Tento nejjednodušší příkaz DELETE je příliš hrubý pro valnou většinu použití. Málokdy potřebujeme takto rozsáhle ničit data v celých tabulkách. Většinou potřebujeme jemnější způsob, jak mazat řádky v tabulkách. Proto úplná podoba příkazu DELETE je trochu složitější:

```
DELETE FROM tabulka WHERE podmínka ORDER BY třídící_kritérium
LIMIT limit_počtu_řádků
```

Klauzule WHERE v příkazu DELETE

Pro selektivní mazání řádků je k dispozici klauzule WHERE. Platí o ní to, že pokud je použita, vymažou se jenom ty řádky, které splňují podmínku danou za klauzulí WHERE. O tom, jaké mohou být podmínky za klauzulí WHERE jsem psal ve 29. díle u příkazu SELECT. U příkazu DELETE jsou možnosti podmínek naprosto stejné.

Například tento příkaz vymaže všechny řádky se jménem začínajícím na Pavel:

```
DELETE FROM Osoby WHERE Jmeno LIKE `Pavel%`
```

A tento příkaz lze využít ke smazání osob s telefonními čísly začínající na dvojku, tedy majitele pražských telefonních čísel:

```
DELETE FROM Osoby WHERE Telefon LIKE `2%`
```

A nakonec ještě třeba jiný typický příklad, jak třeba vymazat osobu s konkrétním rodným číslem:

```
DELETE FROM Osoby WHERE RodneCislo = `651209/2344`
```

Pokud se nám podaří v klauzuli WHERE zadat takové podmínky, že se nevymaže žádný řádek, nebere se to za chybu, ale vše proběhne bezchybně.

Klauzule LIMIT v příkazu DELETE

Klauzule LIMIT je jiným způsobem, jak omezit počet mazaných řádků. Například následující příkaz smaže prvních 10 řádků z tabulky Osoby:

```
DELETE FROM Osoby LIMIT 10
```

Sama o sobě je klauzule LIMIT málo užitečná, pokud není spojena se seřazením řádků pomocí klauzule ORDER BY. Není totiž v našem případě přesně jasné, kterých 10 řádků se vlastně smaže.

Klauzule ORDER BY v příkazu DELETE

Klauzule ORDER BY má svůj užitek především při spojení s klauzulí LIMIT. Klauzule ORDER BY slouží k seřazení řádků podle určitých sloupců. Použití klauzule ORDER BY je naprosto stejné jako u příkazu SELECT ve 29. díle seriálu, podívejte se proto tam, jak se detailně používá.

Příkladem může být třeba příkaz, který seřadí řádky podle telefonního čísla a pak vymaže 3 osoby s nejmenšími telefonními čísly:

```
DELETE FROM Osoby ORDER BY Telefon LIMIT 3
```