

Kapitola 3

Základy PHP

Máme za sebou pouhé dvě kapitoly, a už jsme probrali poměrně dost základních informací o jazyku PHP. Seznámili jste se s pozadím a historií vzniku jazyka a hluboko jste se zavrtali do pojmů a postupů souvisejících s instalací a konfigurací. Tím jste si vytvořili dobrou výchozí pozici k tomu, co je jádrem zbývajících částí knihy: vytváření vyspělých aplikací PHP. Jejich výklad teď začíná, uvádí se v něm mnoho základních schopností jazyka. Konkrétně se budou probírat následující témata:

- Jak se odděluje kód PHP, což umožňuje enginu pro analýzu určit, které oblasti skriptu má analyzovat, a které má ignorovat.
- Jak se do kódu vkládají komentáře pomocí různých metodologií, které jsou vypůjčené ze skriptování shellu Unixu a jazyků C a C++.
- Jak dostanete data na výstup pomocí příkazů `echo()`, `print()`, `printf()` a `sprintf()`.
- Výklad typů dat PHP, proměnných, operátorů a příkazů.
- Podrobné pojednání o klíčových řídicích strukturách a příkazech PHP: `if-else-elseif`, `while`, `foreach`, `include/require`, `break`, `continue` a `declare`.

V kapitole si osvojíte nejen vědomosti nezbytné k tomu, abyste mohli vytvářet sice jen základní, ale přesto prospěšné aplikace PHP. Pochopíte také to, co vám umožní vytěžit co nejvíce z látky probírané v následujících kapitolách.

Únik k PHP

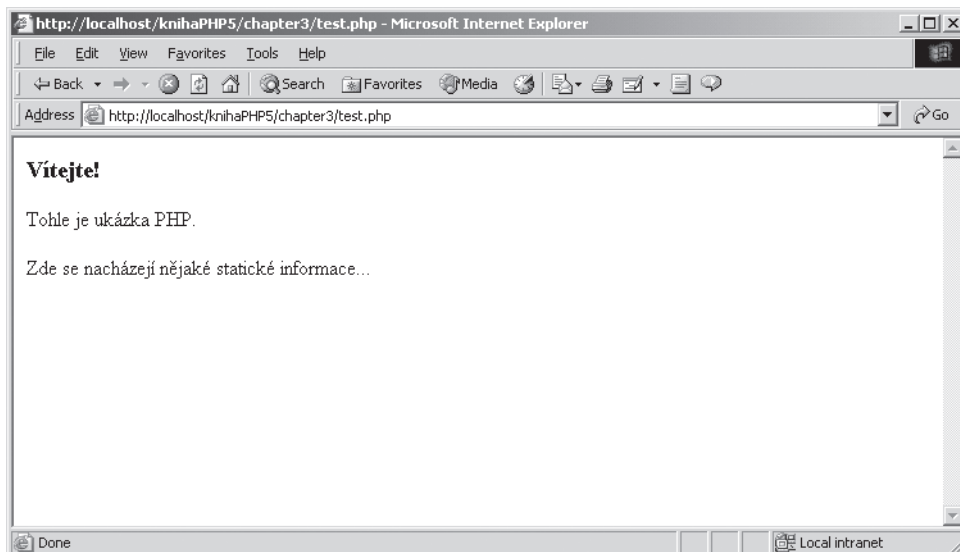
Jednou z předností PHP je, že jeho kód můžete vkládat přímo do statických stránek HTML. Aby ale kód mohl něco dělat, musí se stránka předat enginu PHP, který ji bude interpretovat. Při interpretaci kódu by ale bylo hodně neefektivní, kdyby se měl každý řádek brát jako potenciální příkaz PHP. Proto parser potřebuje nějaké prostředky, aby mohl okamžitě určit, které oblasti na stránce představují kód PHP. To se logicky docílí tím, že se kód PHP oddělí. Existují čtyři varianty, jak lze kód oddělit. Jejich popis následuje.

Výchozí syntax

Výchozí syntax oddělení kódu začíná znaky `<?php` a končí znaky `?`, jako zde:

```
<h3>Vítejte!</h3>
<?php
    print "<p>Tohle je ukázka PHP.</p>";
?>
<p>Zde se nacházejí nějaké statické informace...</p>
```

Uložíte-li kód uvedený výše jako `test.php` a zavoláte ho z nějakého webového serveru, který má zapnutou podporu PHP, uvidíte výstup jako na obrázku 3-1:



Obrázek 3-1 Ukázka výstupu PHP

Krátké značky

Pro lenochy je k dispozici ještě kratší syntax oddělovačů. Říká se jí krátký styl značek (short-tags). Obejde se bez odkazu `php`, který vyžaduje výchozí syntax. Chcete-li ale používat krátké značky, musíte zapnout direktivu `short_open_tag` PHP. Příklad:

```
<?
    print "Tohle je další ukázka PHP.";
?>
```

Upozornění

Přestože jsou krátké značky pohodlné, mějte na paměti, že kolidují s XML, a tedy i se syntaxí XHTML. Proto se kvůli souladu doporučuje, abyste používali výchozí syntax..

Informace se obvykle zobrazují příkazy `print` nebo `echo`. Když jsou zapnuté krátké značky, můžete se bez těchto příkazů obejít, a využít variantu výstupu známou jako zkrácená výstupní syntax:

```
<?="Tohle je další ukázka PHP.";?>
```

Což je funkčně ekvivalentní oběma následujícím variantám:

```
<? print "Tohle je další ukázka PHP."; ?>
```

```
<? php print "Tohle je další ukázka PHP";?>
```

Styl <script>

Z historických příčin měly některé editory, konkrétně editor FrontPage společnosti Microsoft, potíže s únikovou syntaxí, kterou se obracíme na PHP. Proto byla do PHP začleněna ještě jedna varianta oddělovací syntaxe, `<script>`:

```
<script language="php">
    print "Tohle je další ukázka PHP.";
</script>
```

Tip

Editor FrontPage společnosti Microsoft rozpoznává oddělovací syntax ve stylu ASP, která následuje.

Styl ASP

Stránky ASP společnosti Microsoft používají obdobnou strategii. Také oddělují statickou syntax od dynamické pomocí předem definovaného vzorku znaků. Dynamická syntax se uvozuje znaky `<%` a ukončuje znaky `%>`. Přicházíte-li z krajiny ASP a rádi byste i nadále používali tuto syntax, PHP ji podporuje. Podívejte se na ukázkou:

```
<%
    print "Tohle je další ukázka PHP.";
%>
```

Vkládání několika bloků kódu

Na PHP se můžete obracet na dané stránce kolikrát, kolikrát jenom chcete. Například, následující ukázka je formálně zcela v pořádku:

```
<html>
<head>
    <title><?php echo "Vítejte na mém webu!";?></title>
</head>
<body>
    <?php
        $date = "18. září 2004";
    ?>
```

```
<h3>Dnes je <?=$date;?></h3>
</body>
</html>
```

Připomínám, že jakékoli proměnné, které se deklarují před blokem kódu, se „zapamatují“ pro potřeby následných bloků, což je v našem příkladu případ proměnné `$date`.

Komentáře

Zdůraznit, jak je důležité prokládat kód pečlivými komentáři, není nikdy zbytečné. PHP pro komentáře nabízí několik syntaktických variant, jejichž popis následuje.

Syntax jediného řádku z C++

Pro komentář často stačí jediný řádek. Protože je komentář krátký, není třeba ho ukončovat speciálním ukončovacím oddělovačem, protože tuto roli uspokojivě zvládne znak pro nový řádek (`\n`). PHP podporuje jednořádkové komentáře ve stylu C++, které jsou uvozené dvěma lomítky (`//`), jako zde:

```
<?php
// Titulek: Můj program PHP
// Autor: Jason
print "Tohle je program PHP";
?>
```

Syntax shellu

PHP také podporuje alternativu k syntaxi ve stylu C++, které se říká syntax shellu. Komentář je uvozen znakem hash (`#`). Předchozí ukázka pak vypadá takto:

```
<?php
# Titulek: Můj program PHP
# Autor: Jason
print "Tohle je program PHP";
?>
```

Komentář na několika řádcích ve stylu C

Často je žádoucí vložit obsírnější popis, jak daná část kódu funguje, nebo jiné poznámky ke kódu. Takové vysvětlení se obvykle na jediný řádek nevejde. Přestože byste mohli postupovat tak, že byste každý řádek zahájili oddělovači ve stylu C++ nebo shellu, nabízí PHP pohodlnější variantu s uvozovacím a ukončovacím oddělovačem komentáře. Podívejte se na několikařádkový komentář:

```
<?php
/*
Titulek: Můj program PHP
Autor: Jason
```

Datum: 10. října 2005

*/

?>

Syntax komentáře na několika řádcích se hodí zejména tehdy, když z kódu generujete dokumentaci, protože nabízí možnost, jak odlišit skutečné komentáře od okomentovaného kódu, což není tak snadné, jako když se používá jednořádková syntax komentářů.

Výstup

Většina webových aplikací je značně interaktivních. Dobře napsané skripty trvale komunikují s uživateli, a to přes nástroje rozhraní, i prostřednictvím odpovědí na požadavky. PHP nabízí pro zobrazování informací řadu prostředků, jejichž popis následuje.

print()

boolean print(*argument*)

Účelem příkazu `print` je poskytnout uživateli zpětnou vazbu. Umí zobrazit prosté řetězce i obsah proměnných. Všechny následující příkazy jsou akceptovatelné příkazy **print**:

```
<?php
    print("<p>Zbožňuji léto.</p>");
?>
```

```
<?php
    $obdobi = "léto";
    print "<p> Zbožňuji $obdobi.</p>";
?>
```

```
<?php
    print "<p> Zbožňuji
    léto.</p>";
?>
```

```
<?php
    $obdobi = "léto";
    print "<p> Zbožňuji ".$obdobi."</p>";
?>
```

Všechny příkazy vypíší:

Zbožňuji léto.

Zatímco tři první varianty jsou patrně velmi snadno pochopitelné, poslední z nich tak zřejmě být nemusí. V poslední variantě jsem totiž zřetězil tři řetězce do jediného pomocí tečky, která slouží v tomto kontextu jako operátor řetězení. Je to technika, která se běžně používá při řetězení proměnných, konstant a statických řetězců. S uvedenou strategií se budete průběžně setkávat na různých místech knihy.

Poznámka

Přestože nás oficiální syntax vybízí, abychom argument dávali do závorek, máte možnost je vynechat. Mnozí programátoři je neuvádějí prostě proto, že je cílový argument zřejmý i bez nich. .

echo()

```
void echo(string argument1 [, ...string argumentN])
```

Příkaz `echo` funguje obdobně jako `print`, ale se dvěma odlišnostmi. Zaprvé nemůže být částí složitějšího výrazu, protože vrací `void`, zatímco `print` vrací hodnotu typu `boolean`. Zadruhé `echo` umí vypsat několik řetězců. Prospěšnost té druhé vymoženosti je problematická, vypadá to, že je to více než cokoli jiného, jen otázka osobních preferencí. Nicméně je k dispozici, pokud pocítíte neodolatelou potřebu ji použít. Tady máte ukázkou:

```
<?php
    $heavyweight = "Lennox Lewis";
    $lightweight = "Floyd Mayweather";
    echo $heavyweight, " a ", $lightweight, " jsou skvělí bojovníci.";
?>
```

Kód vyprodukuje následující výstup:

Lennox Lewis a Floyd Mayweather jsou skvělí bojovníci.

Tip

Co je rychlejší, `echo()` nebo `print()`? To, že jsou funkčně zaměnitelné, způsobuje, že si mnozí kladou tuto otázku. Odpověď zní, že funkce `echo()` je nepatrně rychlejší, protože nic nevrací, kdežto `print()` vrací booleovskou hodnotu, kterou informuje volajícího, zda příkaz zvládl výstup úspěšně, nebo ne. Je zcela nepravděpodobné, že byste si všimli nějakého rozdílu v rychlosti, takže se můžete rozhodnout podle toho, co vám více vyhovuje stylisticky.

printf()

```
boolean printf (string format [, mixed args])
```

Funkce `printf()` je funkčně identická s `print()`. Jde o výstup argumentů specifikovaných v `args`, ale výstup se naformátuje podle `format`. To umožňuje získat značnou kontrolu nad výstupem dat, pokud jde o takové věci, jako jsou zarovnání, přesnost, typ či pozice. Argument se může skládat až z pěti komponent, které se musejí ve `format` objevit ve stanoveném pořadí:

- **Specifikátor doplnění:** volitelná komponenta, která určuje, jakým znakem se výstup doplní na správnou délku řetězce. Výchozí znak je mezera. Alternativní znak se specifikuje tak, že se před něj napíše apostrof.
- **Specifikátor zarovnání:** volitelná komponenta, která určuje, zda má být výstup zarovnaný doleva nebo doprava. Výchozí je doprava. Zarovnání vlevo vynutíte tím, že uvedete znaménko minus.

- **Specifikátor délky:** volitelná komponenta, která určuje minimální počet znaků, který má výstup z funkce obsahovat.
- **Specifikátor přesnosti:** volitelná komponenta, která určuje, kolik se má zobrazit desetinných míst. Má vliv pouze na data typu `float`.
- **Specifikátor typu:** určuje, jak se argument přetypuje. Podporované specifikátory typu jsou uvedené v tabulce 3-1.

Tabulka 3-1. Podporované specifikátory typu.

Typ	Popis
%b	Argument se považuje za celé číslo, zobrazí se jako binární číslo.
%c	Argument se považuje za celé číslo, zobrazí se znak odpovídající dané hodnotě ASCII.
%d	Argument se považuje za celé číslo, zobrazí se jako dekadické číslo se znaménkem.
%f	Argument se považuje za číslo v pohyblivé řádové čárce a tak se také zobrazí.
%o	Argument se považuje za celé číslo, zobrazí se jako oktalové číslo.
%s	Argument se považuje za řetězec, zobrazí se jako řetězec.
%u	Argument se považuje za celé číslo, zobrazí se jako dekadické číslo bez znaménka.
%x	Argument se považuje za celé číslo, zobrazí se jako hexadecimální číslo s malými písmeny.
%X	Argument se považuje za celé číslo, zobrazí se jako hexadecimální číslo s velkými písmeny

Podívejte se na několik ukázek:

```
printf("%$01.2f", 43.2); // $43.20
printf("%s je %d lahví piva", "Ve sklepě", 100); // Ve sklepě je 100 lahví piva
printf("%15s", "Nějaký text"); //   Nějaký text
```

Někdy se hodí změnit výstupní pořadí argumentů, nebo zopakovat výstup některého konkrétního argumentu, aniž byste ho museli opakovaně uvádět v seznamu argumentů. To se dělá tak, že se odkážete na argument, a zároveň na jeho pozici. Například, `%2$` vyjadřuje argument umístěný v seznamu argumentů na druhé pozici, zatímco `%3$` vyjadřuje třetí. Když to však umístíte do řetězce `format`, musíte před znak dolar uvést obrácené lomítko, například `%2\`. Dvě ukázky:

```
printf("Náš %2\$s rád %1\$s", "štěká", "pes");
// Náš pes rád štěká
printf("Náš %1\$s říká: %2\$s, %2\$s.", "pes", "haf");
// Náš pes říká: haf, haf.
```

sprintf()

```
string sprintf (string format [, mixed arguments])
```

Funkce `sprintf()` se funkčně shoduje s `printf()` až na to, že se výstup přiřadí do řetězce, nejde přímo na standardní výstup. Ukázka:

```
$cena = sprintf("%$01.2f", 43.2); // $cena = $43.20
```

Typy dat

Datový typ je obecný název přiřazený nějaké množině dat, která sdílí nějakou společnou množinu charakteristik. Mezi běžné datové typy patří řetězce, celá čísla, čísla v pohyblivé řádové čárce a booleovské hodnoty. PHP už dlouho nabízí bohatou sadu datových typů. K nim přibýly ve verzi 5 další. V tomto oddílu projdu jednotlivé datové typy, které lze všechny rozdělit do tří kategorií: *skalární*, *složené* a *speciální*.

Skalární datové typy

Skalární datové typy mohou obsahovat jen jediný prvek informace. Sem spadá několik datových typů: `boolean`, `integer`, `float` a `string`.

Boolean

Datový typ `boolean` se jmenuje po matematikovi George Booleovi (1815–1864), který je pokládám za jednoho ze zakladatelů teorie informace. Booleovská proměnná reprezentuje pravdivost, a podporuje tedy pouze dvě hodnoty: `TRUE` (pravda) nebo `FALSE` (nepravda); na velikosti písmen nezáleží. Alternativou je reprezentovat `FALSE` hodnotu nula, a `TRUE` reprezentovat jakoukoli nenulovou hodnotou.

```
$jenazivu = false;    # $jenazivu je false.
$jenazivu = 1;        # $jenazivu je true.
$jenazivu = -1;       # $jenazivu je true.
$jenazivu = 5;        # $jenazivu je true.
$jenazivu = 0;        # $jenazivu je false.
```

Integer

Typ `integer` vyjadřuje prostě celé číslo, neboli takové číslo, které nemá desetinnou část. Sem spadají čísla dekadická (o základu 10), oktalová (o základu 8) a hexadecimální (o základu 16). Několik ukázek:

```
42      # dekadické
-678900 # dekadické
0755    # oktalové
0xC4E   # hexadecimální
```

Jaké je největší podporované celé číslo, závisí na platformě, i když obvykle je to plus nebo minus 231. Pokusíte-li se tento limit v nějakém skriptu PHP překročit, převede se takové číslo automaticky do pohyblivé řádové čárky (na `float`). Ukázka:

```
<?php
$val = 45678945939390393678976;
echo $val + 5;
?>
```

Výsledek bude:

```
4.567894593939E+022
```