Jakub Vrána

1001 TIPŮ A TRIKŮ PRO PHP

CD obsahuje
veškeré zdrojové kódy z knihy, instalaci Apache, MySQL a další užitečné nástroje

Nejlepší postupy a řešení pro vaše webové aplikace

- Využití AJAXu, návrhových vzorů, export a import dat
- Zabezpečení dat, ladění výkonu aplikace
- Práce se soubory, XML dokumenty, knihovnami a frameworky
- Regulární výrazy, zpracování formulářů a e-mailů

CPRESS

Jakub Vrána

1001 tipů a triků pro PHP

Computer Press
Brno
2012

1001 tipů a triků pro PHP

Jakub Vrána

Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-2940-1

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 097.

Dotisk 1. vydání.

ALBATROS MEDIA a.s.

Stručný obsah

Obsah

Webové aplikace 167

Úvod

Kromě toho, že se živím programováním, tak se také podílím na tvorbě oficiální dokumentace PHP a učím základy vytváření webových stránek na vysoké škole. Vedle toho píšu *blog php.vrana.cz* a vedu několik školení o tvorbě webových aplikací se zaměřením na PHP, MySQL a JavaScript. Ze všech těchto vstupů čerpám inspiraci, a proto doufám, že všechny tipy a triky v této knize budou pro čtenáře nějakým způsobem zajímavé. S vatovými triky následujícího typu se tedy v této knize rozhodně nesetkáte:

1. Věděli jste, že oblouk nakreslíte funkcí `imageArc`?
2. Věděli jste, že obdélník nakreslíte funkcí `imageRectangle`?
3. Věděli jste, že čáru nakreslíte funkcí `imageLine`?

Místo toho najdete třeba i tipy týkající se výkonnosti webových aplikací, verzovacích systémů nebo dokonce vývoje PHP. I to je totiž pro programátora v PHP důležité.

Kniha u čtenáře předpokládá znalost syntaxe PHP a schopnost dohledávat si informace v dokumentaci. Pokud tedy nějaký tip např. pojednává o zajímavém využití nějaké funkce, tak se nevysvětluje význam jednotlivých parametrů, protože ten je popsán v dokumentaci.

Kniha byla napsaná v době PHP 5.3, v době zrušení vývojové verze PHP 6, která měla zavést podporu pro Unicode. Zabývá se tedy verzemi PHP 5.3 a staršími.

Kompletní text knihy lze prohledávat pomocí hledání na webu *php.vrana.cz*.

Komu je kniha určena

Tipy jsou rozděleny do tří kategorií pro začátečníky, pokročilé a znalce. Rozdělení ale není ani tak podle náročnosti, jako spíš podle jejich potřebnosti.

Tipy pro začátečníky by měl znát i začínající programátor (i když někdy můžou být složitější).

Tipy pro pokročilé by měla znát většina programátorů PHP.

Bez znalosti tipů pro znalce se dá často obejít, i když samozřejmě můžou poskytnout nějakou zkratku nebo usnadnění práce.

Odlišení písma

V knize se používá tučné písmo pro názvy produktů, kurziva pro odborné termíny, neproporcionální písmo pro ukázky kódu. V ukázkách kódu jsou tučně klíčová slova a vestavěné funkce, kurzivou proměnné.

Poděkování

Chtěl bych poděkovat Davidu Grudlovi, který je autorem několika knihoven použitých v této knize a také spoluautorem kódu u několika tipů. Také mu děkuji za řadu cenných připomínek.

Doprovodné CD

Doprovodný disk obsahuje kromě zdrojových kódů také řadu odkazů na užitečné stránky a také několik užitečných nástrojů, jež vám programování v jazyce PHP výrazně usnadní nebo alespoň zpříjemní.

CD stačí vložit do počítače a rozhraní se spustí automaticky. Pokud nemáte automatické spouštění disků povoleno, vyhledejte na CD kořenový adresář a otevřete soubor spustit_CD.html.

Jestliže rozhraní CD otevřete v prohlížeči Internet Explorer, Opera nebo Google Chrome, budete z CD moci instalovat doprovodný software okamžitě. V případě jiných prohlížečů se zobrazí výzva k uložení instalačního souboru na pevný disk. V tomto případě doporučujeme spustit instalaci přímo z CD. Obsah CD najdete ve složce obsah.

Přehledný kód

1 Jak je to s velikostí písmen

V důsledku toho, že PHP převzalo řadu funkcí a obratů z jiných programovacích jazyků, nemá jednotné pravidlo pro používání velkých a malých písmen. Alespoň některá pravidla jsou ale společná:

- Proměnné a funkce je zvykem začínat malým písmenem.
- Názvy tříd obvykle začínají velkým písmenem.
- Konstanty se píšou celé velkými písmeny a slova se oddělují podtržítkem.

Nejednotnost tkví hlavně v oddělování slov. Většina vestavěných funkcí PHP používá pro oddělení slov podtržítko (např. `array_keys`), pokud to ovšem nejsou zkratky (jako např. `strPos`). Některé funkce se toho ale nedrží a slova nijak neoddělují (např. `imageCreate`).

Objektový kód obvykle slova odděluje zvětšením písmene (např. `PDO::errorCode`), i když ani to neplatí vždy (`mysqli::select_db`).

U funkcí a tříd PHP velikost písmen nerozlišuje, u proměnných ale ano. V této knize budeme pro oddělování slov používat zvětšení písmene, z důvodu lepší čitelnosti i u vestavěných funkcí PHP.

2 Jak je to s velikostí písmen u zkratek

Pokud je součástí identifikátoru (např. proměnné nebo funkce) nějaká zkratka, je vhodné ji zapisovat malými písmeny stejně jako ostatní slova. Jinak bude identifikátor hůře čitelný, obzvlášť v situaci, kdy by zkratek bylo více za sebou.

```
<?php
// dobře čitelné
createXmlHttpRequest($xmlHttp);

// špatně čitelné, proměnné by navíc měly začínat malým písmenem
createXMLHTTPRequest($XMLHTTP);
?>
```

3 Jak je to s velikostí písmen v URL

Adresu URL je zvykem zapisovat malými písmeny a slova oddělovat spojovníkem, stejný způsob pojmenování tedy volíme u skriptů přímo zobrazovaných na webu.

U souborů vkládaných do jiných skriptů odvíjíme název souboru od toho, co v souboru definujeme. Pokud to je např. třída, tak soubor můžeme pojmenovat stejně jako třídu, včetně velikosti písmen.

4 Diakritika v identifikátorech

PHP dovoluje v názvech proměnných a v dalších identifikátorech používat i znaky z horní poloviny ASCII tabulky. Pokud se tedy rozhodneme zadávat identifikátory česky, můžeme je psát i s diakritikou. V praxi se toho ale téměř nevyužívá, protože tento způsob zápisu nemusí podporovat všechny editory.

5 Diakritika v komentářích

Pokud se rozhodneme komentáře k programu psát česky, měli bychom zásadně používat diakritiku (háčky a čárky). Souvislejší texty bez diakritiky se totiž velmi špatně čtou. Zdrojové kódy se v poslední době obvykle píšou v kódování UTF-8.

6 Rozmístění tříd do souborů

Každá třída se obvykle umisťuje do samostatného souboru pojmenovaného podle jejího názvu. Toto pravidlo se občas porušuje u definic výjimek, které se buď definují hromadně v jednom souboru, nebo spolu s třídou, která je vyvolává.

7 Pojmenování rozhraní

Název rozhraní (`interface`) v některých projektech začíná znakem I, zdaleka to ale není pravidlem. Stejně jako třídy se rozhraní obvykle umisťují do souborů pojmenovaných podle jejich názvu.

8 Co to je značka Byte-Order-Mark

Soubory ve znakové sadě Unicode mohou začínat značkou *Byte-Order-Mark* (BOM). To je posloupnost bajtů na začátku souboru, která obsahuje informaci o použitém kódování. Platí to i pro kódování UTF-8, které lze používat pro vytváření PHP skriptů. Tuto značku ale PHP nijak neinterpretuje, takže ji není vhodné používat. Jinak může dojít k tomu, že např. funkce `header` skončí chybou, protože už skript odeslal nějaký výstup (neviditelnou značku BOM) nebo že skript odešle těchto značek více (pokud do sebe vkládáme více skriptů, které by měly značku uvedenou).

9 Čeština nebo angličtina?

Při programování bychom si měli vybrat nejen programovací jazyk (což je nevyhnutelné), ale i ten lidský, který budeme používat pro identifikátory a texty v programu. U identifikátorů, jako je název proměnných nebo funkcí, je obvykle přirozené používat angličtinu, protože i návazný kód používá angličtinu.

```
<?php
$xmlReader = new XmlReader; // působí přirozeně
$cteckaXml = new XmlReader; // působí krkolomně
?>
```

V některých případech se navíc používají ustálené obraty jako např. název metody `getConnection`. České varianty, ať už v podobě `getPripojeni` nebo třeba `dejPripojeni`, působí nepřirozeně.

V knize tedy budeme používat anglické identifikátory.

Poznámka: Stejné pravidlo lze použít i pro pojmenování souborů.

10 Jazyk textů a komentářů

U vypisovaných textů je situace daná tím, zda vytváříme jednojazyčnou nebo vícejazyčnou aplikaci. U jednojazyčné je přirozené psát texty rovnou ve výsledném jazyce, u vícejazyčné potom v angličtině (pokud je jedním z výsledných jazyků).

U komentářů záleží na tom, kdo s nimi bude pracovat. Pokud všichni autoři i čtenáři komentářů ovládají dostatečně dobře angličtinu, tak je vhodné psát komentáře právě v ní, aby byl jazyk v souladu s identifikátory. Pokud taková situace nenastane, tak je vhodné psát komentáře jazykem, který všichni zúčastnění ovládají nejlépe.

Vypisované texty a komentáře v knize budou tedy česky.

11 Pojmenování funkcí

Název funkcí a metod je vhodné začínat slovesem určujícím, co přesně vlastně funkce nebo metoda dělá. Např. `getConnection` je lepší název než samotné `connection`, protože je z názvu jasné, že jde o získání připojení, a ne např. o jeho nastavení. Slovesem je vhodné název funkce vždy začínat, `getConnection` je lepší než `connectionGet`. U neobjektového kódu je možné názvu funkce předřadit modul, kterého se týká (např. `imageCreate`), u metod objektů to není potřeba.

12 Název aplikace v identifikátorech

Budoucí verze PHP mohou používat nová klíčová slova nebo jiné identifikátory, které by mohly kolidovat s identifikátory použitými naší aplikací. V zájmu dopředné kompatibility bychom tedy v hlavním jmenném prostoru aplikace neměli používat příliš obecné identifikátory. Např. PHP 5.1 zavedlo třídu `Date` (ta byla po sporech nakonec přejmenována na `DateTime`). Stejně tak mohou nové identifikátory zavádět používané knihovny.

Předřazovat všem identifikátorům název aplikace ale také není zrovna nejšťastnější, protože to identifikátory prodlužuje a znepřehledňuje.

```
<?php
// v budoucnu může kolidovat
class Database {
}

// dlouhý a nepřehledný identifikátor
class MyAppDatabase {
}
```

```
?>
```

Čisté řešení tohoto problému nabízí jmenné prostory, které jsou k dispozici od PHP 5.3.

```
<?php
namespace MyApp;
class Database {
}
new Database; // použití zevnitř jmenného prostoru
new \MyApp\Database; // použití kdekoliv
?>
```

13 Velikost písmen v SQL dotazech

V SQL dotazech je zvykem klíčová slova uvádět velkými písmeny a vlastní identifikátory (především názvy tabulek a sloupců) malými písmeny. Slova se obvykle oddělují podtržítkem. Někdo se kloní k oddělování slov zvětšením písmene stejně jako v PHP kódu, s tím ale mohou být za určitých okolností problémy (např. MySQL v závislosti na konfiguraci může názvy tabulek převádět na malá písmena).

Pokud se u identifikátorů rozhodneme používat češtinu, je zvykem ji psát bez diakritiky, i když by si většina databázových serverů poradila i s ní. V komentářích tabulek a sloupců je naopak diakritiku vhodné používat.

14 Jak vypadá Spaghetti code

O špagetovém kódu mluvíme tehdy, když jsou jednotlivé části aplikace promíchané tak, že se v nich dá jen těžko vyznat. U webových aplikací je obzvlášť jednoduché takový kód vytvořit, protože se v nich používá řada jazyků zapisovaných v prostém textu:

- **JavaScript** je vhodné vyčlenit do samostatného souboru, který z HTML odkážeme značkou `<script src="">`. Přímo v HTML kódu je vhodné uvádět pouze inicializaci proměnných JavaScriptu, které jsou specifické pro daného uživatele.
- **CSS** je také vhodné dát do samostatného souboru a odkázat ho značkou `<link rel="stylesheet" href="">`. Provázání s HTML kódem zajišťují názvy značek, tříd a identifikátorů.
- **HTML** je vhodné nemíchat s řídicím PHP kódem, ale vyčlenit ho do samostatné šablony. Ta může využívat nějaký šablonovací systém nebo může být zapsaná také v PHP, vždy ale platí, že už slouží jen k vypsání dat.
- **SQL** je užitečné centralizovat do tzv. *modelu* (což je část aplikace zajišťující především získávání a ukládání dat), případně se bez něj díky ORM můžeme dokonce obejít.

Výhodou rozdělení aplikace do jednotlivých částí je jejich menší složitost a snazší udržovatelnost. Umožňuje to také větší specializaci vývojářů, kdy nehrozí, že HTML kodér poškodí práci PHP programátora.

Poznámka: Některé ukázky kódu v knize se této zásady nebudou kvůli jednoduchosti držet.

15 Jak na odsazování kódu

Bílé znaky jako mezery a konce řádků se sice v PHP až na výjimky ignorují, i tak je ale vhodné kvůli čitelnosti dodržovat pevná pravidla jejich zápisu. Každý příkaz se obvykle píše na jeden řádek – dlouhé příkazy lze rozdělit, ale nikdy bychom neměli na jeden řádek dávat příkazů více. Bloky kódu je zvykem odsazovat, otevírací závorka se umisťuje obvykle na konec předchozího řádku, u funkcí a tříd ji lze dát i na samostatný řádek.

Poznámka: V této knize se budeme zásad správného formátování kódu držet, otevírací složená závorka bude vždy na konci předchozího řádku.

16 Znak pro odsazování kódu

O znacích používaných pro odsazování kódu se vedlo mnoho bezvýsledných debat. Někdo používá dvě mezery, někdo čtyři, někdo trvá na osmi mezerách. Osobně používám tabulátor z těchto důvodů:

- Každý programátorský editor dovoluje nastavit jeho zobrazovanou šířku. Ostatní editory ho obvykle zobrazují jako osm mezer.
- Odsazení je otázkou jednoho stisknutí klávesy v libovolných podmínkách.
- V uloženém souboru zabírá tabulátor jen jeden bajt.

17 Odsazování konstrukce switch

Formátování konstrukce `switch` je ošemetné, protože vytváří dvě úrovně. Někdo doporučuje zarovnávat `case` na stejnou úroveň jako `switch`, někdo o úroveň dál. Pokud je ve větvích `case` jen jeden příkaz, tak se mi zdá nejpřehlednější ho uvést rovnou na řádek `case`:

```
<?php
function numbers($number) {
    switch ($number) {
        case 0: return "nula";
        case 1: return "jedna";
        case 2: return "dva";
        default: return null;
    }
}
?>
```

18 Odsazování kódu v kombinaci s HTML

Pokud kombinujeme PHP kód s HTML, tak může být správné odsazování oříškem. Můžeme odsazovat buď pouze PHP kód, nebo pouze HTML, nejpřehlednější je ale asi odsazování obojího.

```
<!-- Odsazování pouze PHP -->
<?php foreach ($data as $row) { ?>
    <tr>
    <td><?php echo htmlSpecialChars($row["name"]); ?></td>
```

```
    </tr>
<?php } ?>

<!-- Odsazování pouze HTML -->
<?php foreach ($data as $row) { ?>
<tr>
    <td><?php echo htmlSpecialChars($row["name"]); ?></td>
</tr>
<?php } ?>

<!-- Kombinace odsazování -->
<?php foreach ($data as $row) { ?>
    <tr>
        <td><?php echo htmlSpecialChars($row["name"]); ?></td>
    </tr>
<?php } ?>
```

Situace se ovšem zkomplikuje, když chceme v bloku PHP kódu vypsat neuzavřenou značku, kterou uzavřeme později. Tam kombinace odsazování příliš dobře nefunguje.

19 Zarovnávání operátorů

S formátováním kódu není vhodné to přehánět. Někteří programátoři se vyžívají třeba v tom, že při více přiřazeních pod sebou zarovnávají operátor přiřazení:

```
<?php
$int   = 0;
$s     = "";
$array = array();
?>
```

S takovýmto odsazováním je spousta práce, která čitelnost žádným zásadním způsobem nezlepší. Problém nastane hlavně v případě, kdy chceme přiřadit další proměnnou s delším názvem – pak musíme zarovnání všude opravit.

Zcela tragicky takovéto zarovnání dopadne v případě, kdy se na zdrojový kód podíváme s proporcionálním písmem, které někteří programátoři používají (protože je čitelnější a vejde se ho na obrazovku víc).

```
<?php
$int   = 0;
$s     = "";
$array = array();
?>
```

Obrázek 1. Zobrazení kódu proporcionálním písmem v editoru SciTE

20 Odřádkování před operátorem

Pokud by jeden příkaz vytvořil příliš dlouhý řádek, lze ho rozdělit na více řádků. Odřádkování je vhodné udělat před operátorem a pokračování příkazu odsadit:

```
<?php
$mail
```

```
    ->setFrom("alice@example.com")
    ->addTo($email)
    ->setSubject("Pokusná zpráva")
;
?>
```

21 Jak na konce řádků

Na Windows se řádky ukončují znaky CR+LF (v PHP řetězci se zapisuje jako `\r\n`), na Linuxu pouze znakem LF (`\n`). PHP koncům řádek nepřikládá zvláštní význam, takže je mu jedno, co použijeme. Programátorské editory si poradí s oběma druhy konců řádků, primitivní textové editory (např. **Notepad** na Windows) vyžadují nativní konce řádků.

Kvůli menší velikosti se obvykle používá znak LF, každopádně je vhodné v celém souboru používat stejný znak.

22 Konec řádku na konci souboru

Poslední řádek souboru je vhodné ukončit koncem řádku (klávesa `Enter`). Usnadňuje to přidávání dalších příkazů a zpřehledňuje to rozdíly mezi verzemi, které něco doplňují na konec souboru.

23 Ukončení posledního prvku pole

Za posledním prvkem v definici pole je možné v PHP uvést čárku stejně jako za ostatními prvky. Velmi užitečné je to hlavně u definice složitějších polí, kde jednotlivé prvky napíšeme na samostatné řádky. Zjednodušuje se tím budoucí přidávání nebo prohazování prvků.

```
<?php
$array = array(
    "první",
    "druhý",
    "třetí",
);
?>
```

24 Pořadí parametrů funkcí

Při vytváření funkce s více než jedním parametrem bychom se měli zamyslet nad tím, v jakém pořadí budeme parametry předávat. U funkcí, které s něčím manipulují, je zvykem v prvním parametru předávat právě tuto hodnotu. Pokud funkce manipulují s objektem, tak je obvykle lepší funkci změnit na metodu tohoto objektu. Další parametry se obvykle uvádí od těch přesnějších po ty vágnější (nejdříve např. předáme ID a až potom název). Volitelné parametry uvádíme pochopitelně až na konci, jinak to v PHP ani nejde.

Funkce by každopádně neměly mít parametrů příliš mnoho, ukázkou z tohoto pohledu nevhodně navržené funkce je `setCookie`, která má 7 parametrů.

25 Pojmenování parametrů funkcí

PHP na rozdíl od některých jiných programovacích jazyků nedovoluje při volání funkce pojmenovat předávané parametry. Všechny parametry je potřeba předat v pořadí, které funkce definuje. Pokud má funkce více podobných parametrů za sebou, je vhodné si hodnotu parametru zapsat do proměnné s popisným názvem.

```
<?php
// nepřehledné
$sqlite->singleQuery($query, true, false);

// přehlednější
$firstRowOnly = true;
$decodeBinary = false;
$sqlite->singleQuery($query, $firstRowOnly, $decodeBinary);

// třetí možnost
$sqlite->singleQuery($query,
    $firstRowOnly = true,
    $decodeBinary = false
);

// využití komentáře
$sqlite->singleQuery($query,
    true, // firstRowOnly
    false // decodeBinary
);

// dá se využít i toho, že se řetězec převede na pravdivostní hodnotu
$sqlite->singleQuery($query, "firstRowOnly", false);
?>
```

Dobrý editor zdrojového kódu nám samozřejmě význam parametru odhalí, ale při prohlížení kódu nemusí být editor k dispozici (ukázky kódu v článcích, prohlížení kódu z repozitáře, atd.).

26 Závorky kolem ternárního operátoru

Kolem ternárního operátoru (neboli podmíněného výrazu) je vhodné dělat závorky, což zlepší jeho čitelnost:

```
<?php
$a = ($m ? f() : ($n ? g() : $o));
?>
```

Ještě přehlednější je v takovýchto případech použít řídicí konstrukci `if`.

27 Použít apostrofy nebo uvozovky?

PHP dovoluje pro zápis řetězců používat uvozovky nebo apostrofy. Uvnitř uvozovek lze používat proměnné a speciální znaky, uvnitř apostrofů ne. Při výběru zápisu bychom

se měli řídit hlavně přehledností. Pokud vypisovaný kód obsahuje uvozovky, tak bývá přehlednější ho uzavřít do apostrofů, jinak budeme v této knize používat uvozovky.

```
<?php
echo '<option value="">(vyberte)</option>';
echo "<div>$html</div>\n";
?>
```

28 Řídicí konstrukce s bloky kódu

Pokud za řídicími konstrukcemi jako `if`, `while` nebo `foreach` uvedeme jenom jeden příkaz, nemusíme ho uzavírat do bloku kódu (složených závorek). I tak je vhodné to ale udělat, protože se tím zlepší přehlednost kódu a usnadní se jeho další modifikace (přidání příkazu).

```
<?php
// přehledný a snadno rozšiřitelný kód
if ($valid) {
    f();
}

// funkce se zavolá vždy, protože se vykoná pouze prázdný příkaz (;)
if ($valid); // chyba
    f();
?>
```

29 Volání funkce před její definicí

PHP dovoluje použít funkci v kódu dříve, než je definovaná. Definice funkce totiž vzniká ve fázi kompilace skriptu, ale zavolá se až ve fázi spuštění. Lepší je ale obvykle funkci nejprve definovat a pak teprve zavolat. Pokud bychom se totiž v budoucnu rozhodli funkci přesunout do vloženého souboru, tak přestane fungovat (vkládané soubory se totiž kompilují až při spuštění hlavního skriptu).

```
<?php
// bude fungovat
function defineFirst() {
}
defineFirst();

// bude fungovat
runFirst();
function runFirst() {
}

// bude fungovat
include "defineFirst.inc.php";
defineFirst();

// nebude fungovat
runFirst();
include "runFirst.inc.php";
?>
```

Do souboru s definicí funkcí je lepší nedávat žádný globální kód. Když toto pravidlo, např. u jednoduchého skriptu spouštěného z příkazové řádky, porušíme a v jednom souboru nadefinujeme funkce i globální kód, tak je zvykem nejprve uvádět definice konstant a funkcí a pak teprve spouštěný kód.

30 V jakém pořadí definovat metody

Na pořadí definic metod stejně jako u funkcí nezáleží, kvůli čitelnosti je ale vhodné dodržovat určitá pravidla. Pořadí definic ve třídě může být takovéto:

- Nejprve vlastnosti v pořadí `public`, `protected`, `private`.
- Konstruktor, destruktor a další magické metody.
- Další metody seřazené opět podle viditelnosti a podle logických celků.

31 Funkce vypisující data

Ve většině programovacích jazyků je lepší se vyhnout vytváření funkcí, které přímo vypisují nějaká data. Co kdybychom pak výstup chtěli použít někde jinde, třeba ho umístit do e-mailu? Místo toho proto funkce obvykle vrací řetězec, který vypíše až volající.

V PHP se velmi pohodlně kombinuje textový výstup s programovým kódem a jakýkoliv výstup lze snadno zachytit. Proto lze ospravedlnit i vytváření funkcí, které něco přímo vypisují. Pokud bychom náhodou s výstupem chtěli udělat něco jiného, můžeme použít *output buffering*:

```
<?php
ob_start();
phpInfo();
$phpInfo = ob_get_clean();
?>
```

32 Postradatelné proměnné a funkce

Pokud výsledek nějakého výpočtu používáme na více místech, je vhodné si ho přiřadit do proměnné. Pokud stejnou posloupnost příkazů používáme víckrát, je vhodné je uzavřít do funkce a volat tuto funkci. Když více funkcí pracuje se stejným objektem, je vhodné je uzavřít do třídy. Tyto zásady vedou k vytváření přehledného, výkonného a snadno udržitelného kódu.

Co když ale výsledek výpočtu používáme jen na jednom místě, co když funkci voláme jen jednou? Vyplatí se proměnnou nebo funkci vytvářet v takovém případě? Z pohledu běhu programu to je zbytečné, výkonnost to zcela nepatrně zhorší, ale program to obvykle zpřehlední – kus kódu tak vlastně dostane své jméno. Porovnejme následující tři ukázky kódu:

```
<?php
// nejefektivnější, ale nepřehledný kód
echo simplexml_load_file(mysql_result(mysql_query("
```

```
    SELECT filename
    FROM xml
    WHERE id = $id
"), 0))->title;

// pojmenování jednotlivých výpočtů
$result = mysql_query("SELECT filename FROM xml WHERE id = $id");
$filename = mysql_result($result, 0);
$xml = simplexml_load_file($filename);
echo $xml->title;

// pojmenování celé operace
function getTitleFromXml($id) {
    $result = mysql_query("SELECT filename FROM xml WHERE id = $id");
    $filename = mysql_result($result, 0);
    $xml = simplexml_load_file($filename);
    return $xml->title;
}
echo getTitleFromXml($id);
?>
```

Postradatelné proměnné a funkce jsou vlastně způsob dokumentování kódu. Pokud si tedy nějaký blok kódu zaslouží komentář, může ho zpřehlednit také vyčlenění do funkce nebo pojmenování vypočtených hodnot.