

Kapitola 7

Pokročilé schopnosti OOP

V kapitole 6 jste absolvovali základy objektově orientovaného programování v PHP. V této kapitole budeme na těchto základech stavět. Seznámíte se s několika vyspělejšími schopnostmi OOP, které byste si měli zařadit do svého repertoáru hned poté, co budete mít v malíku základy. Konkrétně se v kapitole hovoří o pěti schopnostech:

- **Klonování objektů.** Jedním z hlavních zdokonalení modelu OOP v PHP verze 5 je to, že se všechny objekty považují za odkazy, ne za hodnoty. Jak se pak ovšem vypořádáme s úlohou vytvořit kopii objektu, když se všechny objekty považují za odkazy? Klonováním objektu, což je nová schopnost v PHP 5.
- **Dědění.** Jak už bylo zmíněno v kapitole 6, je schopnost budovat hierarchie tříd prostřednictvím dědění klíčovým pojmem OOP. Dozvíte se, co je to dědění, jakou má v PHP 5 syntax, a uvidíte několik příkladů, které předvádějí tuto klíčovou schopnost OOP.
- **Rozhraní.** Rozhraní je nějaká kolekce neimplementovaných definic metod a konstant. Slouží jako jistý náčrt třídy. Rozhraní přesně definují, co se může ve třídě dělat, ale nezatěžují se konkrétními podrobnostmi implementace. V této kapitole se dozvíte, jakou podporu poskytuje PHP 5 pro rozhraní, a uvidíte také několik příkladů, které předvádějí tuto klíčovou schopnost OOP.
- **Abstraktní třídy.** Abstraktní třída je v podstatě taková třída, ze které se nedají vytvářet instance. Účelem abstraktních tříd je to, aby se z nich odvozovaly třídy, jejichž instance se už dají vytvářet (říká se jim někdy konkrétní třídy). Abstraktní třídy lze implementovat plně, nebo implementovat jen částečně, nebo neimplementovat vůbec. V této kapitole se uvádějí všeobecné pojmy, které se točí okolo abstraktních tříd spolu s úvodem, jak se abstraktní třídy implementují v PHP 5.
- **Reflexe.** Jak jste se dozvěděli v kapitole 6, skrývání „odpudivých“ podrobností aplikace přivítavými rozhraními (zapouzdření) je jedním z hlavních pilířů OOP. Nicméně programátoři ovšem potřebují nějaké pohodlné prostředky, jimiž se dá prozkoumávat chování třídy. Tuto schopnost poskytuje pojem reflexe, který se zde vysvětluje.

Schopnosti OOP nepodporované v PHP

Máte-li zkušenosti s jinými objektově orientovanými jazyky, možná po přečtení výše uvedeného seznamu nespokojeně potřásáte hlavou, a ptáte se, proč v něm chybí jedna nebo několik konkrétních schopností OOP, které důvěrně znáte z jiných jazyků. Je docela dobře možné, že je to kvůli tomu, že PHP tuto schopnost nepodporuje. Takže abyste si hlavu úplně nevyviklali z pantů, jsou v následujícím výčtu uvedena ta vyspělá témata OOP, která PHP nepodporuje, a proto jejich výklad v kapitole nenajdete.

- **Jmenné prostory** (namespace). Přestože byly původně do PHP 5 naplánované, od začlenění jmenných prostorů bylo brzo upuštěno. Není jasné, zda bude podpora jmenných prostorů integrována do nějaké budoucí verze.
- **Přetěžování metod** (overloading). Možnost docílit polymorfického účinku prostřednictvím přetěžování funkcí PHP nepodporuje a podle diskuse na webových stránkách Zend Technologies, patrně ani nikdy podporovat nebude. Více se o tom můžete dozvědět na adrese http://www.zend.com/php/ask_experts.php.
- **Přetěžování operátorů**. Možnost přiřadit dodatečné významy stávajícím operátorům na základě typu dat, která se pokoušíte modifikovat, zatím není na pořadu dne. Podle výše zmíněné diskuse na webu Zend Technologies, není pravděpodobné, že by se tato schopnost někdy v budoucnu implementovala.
- **Vícenásobné dědění**. PHP nepodporuje vícenásobné dědění. Implementace více rozhraní se však podporuje.

Jen čas ukáže, budou-li se v budoucnu některé z těchto schopností v PHP podporovat.

Klonování objektů

Jednou z největších stinných stránek objektově orientovaných schopností PHP 4 byla skutečnost, že zacházel s objekty jako s jinými datovými typy. Tato praxe byla překážkou využití mnohých běžných metodologií OOP, jako jsou návrhové vzory. Takové metodologie závisejí na možnosti předávat objekty do jiných metod třídy odkazem, ne hodnotou, což byla výchozí praktika v PHP. Naštěstí byla tato záležitost v PHP 5 vyřešena, a nyní se standardně se všemi objekty zachází jako s odkazy. Protože se ale nyní se všemi objekty zachází jako s odkazy, ne jako s hodnotami, je zase obtížnější udělat kopii objektu. Pokušíte-li se zkopírovat odkazovaný objekt, bude prostě ukazovat zpět na adresované umístění původního objektu. PHP se s tím vypořádal tak, že nabízí explicitní prostředek pro klonování objektu.

Příklad klonování

Klon objektu vytvoříte tak, že před ním uvedete klíčové slovo `clone`, jako zde:

```
clonový_objekt = clone zdrojový_objekt;
```

Vyčerpávající příklad klonování objektu nabízí výpis 7-1. V příkladu jsem vytvořil ukázkovou třídu s názvem `parazitFirma`, která obsahuje dva členy (`idZamestnance` a `barvaKravaty`), a jejich odpovídající gettery a settery. V příkladu se vytvoří instance `parazitFirma` a použije se jako základna pro předvedení, jaké účinky má operace klonování.

Výpis 7-1. Klonování objektu pomocí klíčového slova clone

```
<?php
class parazitFirmy {
    private $idZamestnance ;
    private $barvaKravaty;

    // Definuje setter a getter pro $idZamestnance
    function nastavitIdZamestnance ($idZamestnance ) {
        $this->idZamestnance = $idZamestnance ;
    }
    function ziskatIdZamestnance() {
        return $this->idZamestnance ;
    }

    // Definuje setter a getter pro $barvaKravaty
    function nastavitBarvuKravaty ($barvaKravaty) {
        $this->barvaKravaty = $barvaKravaty;
    }

    function ziskatBarvuKravaty() {
        return $this->barvaKravaty;
    }
}

// Vytvoří nový objekt parazita firmy
$parazit1 = new parazitFirmy();

// Nastaví člen idZamestnance parazita 1
$parazit1->nastavitIdZamestnance ("12345");

// Nastaví člen barvaKravaty parazita 1
$parazit1->nastavitBarvuKravaty ("červená");

// Vytvoří klon parazita 1
$parazit2 = clone $parazit1;

// Nastaví člen idZamestnance parazita 2
$parazit2->nastavitIdZamestnance ("67890");

// Výstup členů prvního a druhého parazita
echo "parazit1 id zaměstnance: ".$parazit1->ziskatIdZamestnance ()."<br />";
echo "parazit1 barva kravaty: ".$parazit1->ziskatBarvuKravaty()."<br />";
echo "parazit2 id zaměstnance: ".$parazit2->ziskatIdZamestnance ()."<br />";
echo "parazit2 barva kravaty: ".$parazit2->ziskatBarvuKravaty()."<br />";
?>
```

Když kód spustíte, vrátí:

```
parazit1 id zaměstnance: 12345
parazit1 barva kravaty: červená
parazit2 id zaměstnance: 67890
parazit2 barva kravaty: červená
```

Jak vidíte, `$parazit2` se stal objektem typu `parazitFirmy` a zdědil hodnoty členů z `$parazit1`. Aby se zdůraznilo, že je `$parazit2` opravdu typu `parazitFirmy`, nastavil se znovu i jeho člen `idZamestnan-`
`ce`.

Metoda `__clone()`

Při klonování objektu si dokonce můžete přizpůsobit chování tohoto procesu, když si ve třídě objektu definujete metodu `__clone()`. Kromě toho, že se do cílového objektu zkopírují všechny existující členy objektu, vykoná se také vše, co je umístěné v této metodě. Upravte třídu `parazitFirmy` tak, že do ní přidáte metodu:

```
function __clone() {
    $this->barvaKravaty = "modrá";
}
```

Až s tím budete hotovi, vytvořte nový objekt `parazitFirmy`, přidejte hodnotu členu `idZamestnance`, udělejte klon objektu, a nakonec vypište potřebná data, abyste si ověřili, že se `barvaKravaty` klonovaného objektu skutečně nastavila prostřednictvím metody `__clone()`. Příklad vidíte ve výpisu 7-2.

Výpis 7-2. Rozšíření možností klíčového slova `clone` metodou `__clone()`

```
// Vytvoří objekt nového parazita firmy
$parazit1 = new parazitFirmy();
// Nastaví člen idZamestnance parazita 1
$parazit1->nastavitIdZamestnance ("12345");
// Vytvoří klon objektu $parazit1
$parazit2 = clone $parazit1;
// Nastaví člen idZamestnance parazita 2
$parazit2->nastavitIdZamestnance ("67890");
// Výstup členů obou parazitů
echo "parazit1 id zaměstnance: ".$parazit1->ziskatIdZamestnance ()."<br />";
echo "parazit2 id zaměstnance: ".$parazit2->ziskatIdZamestnance ()."<br />";
echo "parazit2 barva kravaty: ".$parazit2->ziskatBarvuKravaty ()."<br />";
```

Když kód vykonáte, vypíše:

```
parazit1 id zaměstnance: 12345
parazit2 id zaměstnance: 67890
parazit2 barva kravaty: modrá
```

Dědění

Lidé mají docela dobrou výbavu pro uvažování v takových termínech, jakým je třeba uspořádaná hierarchie, proto není nijak překvapující, že je takové nahlížení na správu mnoha aspektů našeho každodenního života široce rozšířené. Struktury managementu firmy, daňový systém Spojených Států, i náš pohled na říše rostlin a živočichů, to je jen několik příkladů systémů, které se převážně spoléhají na hierarchie. Protože je objektově orientované programování založeno na premise, že my lidé dokážeme velmi dobře modelovat vlastnosti a chování prostředí skutečného světa, které se snažíme implementovat pomocí kódu, dá se z toho rozumně usoudit, že jsme také schopni znázorňovat tyto hierarchické vztahy.

Předpokládejme například, že vaše aplikace potřebuje třídu nazvanou `Zamestnanec`, jejíž účelem má být reprezentace charakteristik a chování, jaká člověk může očekávat od nějakého zaměstnance. Mezi členy takové třídy by mohly patřit

- `jmeno`: jméno zaměstnance
- `vek`: věk zaměstnance
- `plat`: plat zaměstnance
- `zamestnan_let`: počet let, které daný zaměstnanec u firmy pracuje.

Mezi metody třídy `Zamestnanec` by mohly patřit:

- `vykonatPraci`: vykoná nějakou práci související se zadaným úkolem.
- `snistObed`: vezme si přestávku na oběd.
- `vzitDovolenou`: týká se nejcennějších týdnů v roce.

Tyto charakteristiky a chování budou relevantní pro všechny druhy zaměstnanců, bez ohledu na účel nebo pozici zaměstnance v organizaci. Je zřejmé, že budou mezi zaměstnanci také jisté rozdíly; například, pracovník exekutivy může mít v držení jisté akcie firmy, a bude sto společnost tunelovat, zatímco jiní zaměstnanci takové výdobytky nemají. Sekretářka musí umět napsat dopis a vedoucí prodejny musí umět udělat inventuru. Navzdory těmto rozdílům by ale bylo velmi neefektivní, kdybyste museli vytvářet a udržovat redundantní struktury tříd pro ty atributy, které sdílejí všichni zaměstnanci. Vývoj schématu OOP to bere v úvahu. Umožňuje dědit z existujících tříd a dále na nich budovat.

Dědění třídy

V PHP se dědění třídy docílí klíčovým slovem `extends`. Předvádí je výpis 7-3, kde se nejprve definuje třída `Zamestnanec`, pak se vytvoří třída `Vedouci`, která bude dědit ze třídy `Zamestnanec`.

Poznámka

Třídě, která dědí z jiné třídy, se říká dceřiná (child) třída, podtřída nebo odvozená třída. Dceřiná třída dědí ze své rodičovské (parent), neboli základní (base) třídy.

Výpis 7-3. Dědění ze základní třídy

```
<?php
# Definuje třídu Zamestnanec
class Zamestnanec {
    private $jmeno;

    # Definuje funkci set vlastnosti pro soukromý člen $jmeno.
    function nastavitJmeno($jmeno) {
        if ($jmeno == "") echo "Jméno nesmí zůstat prázdné!";
        else $this->jmeno = $jmeno;
    }

    # Definuje funkci get vlastnosti pro soukromý člen $jmeno
    function ziskatJmeno() {
        return "Jmenuji se ".$this->jmeno."<br />";
    }

} #konec třídy Zamestnanec

# Definuje třídu Vedoucí, která bude dědit z třídy Zamestnanec
class Vedoucí extends Zamestnanec {

    # Definuje metodu jedinečnou pro Zamestnanec
    function tunelovatFirmu() {
        echo "Prodám aktiva firmy, abych mohl financovat svou jachtu!";
    }

} #konec třídy Vedoucí

# vytvoří nový objekt Vedoucí
$exec = new Vedoucí();

# zavolá metodu nastavitJmeno(), která je definovaná ve třídě Zamestnanec
$exec->nastavitJmeno("Richard");

# Zavolá metodu ziskatJmeno()
echo $exec->ziskatJmeno();

# zavolá metodu tunelovatFirmu()
$exec->tunelovatFirmu();
?>
```

Kód vrátí

Jmenuji se Richard.

Prodám aktiva firmy, abych mohl financovat svou jachtu!