Marian Böhmer

Zend Framework

Programujeme webové aplikace v PHP

Výuka na konkrétních příkladech

Využití všech důležitých komponent

Naučte se také modernímu stylu programování

CD obsahuje

Zdrojové kódy příkladů
Instalaci Zend Framework
Referenční příručku

CPRESS

Marian Böhmer

Zend Framework
Programujeme webové aplikace v PHP

Computer Press, a. s.
Brno
2010

Zend Framework
Programujeme webové aplikace v PHP

Marian Böhmer

Computer Press, a. s., 2010. Vydání první.

Jazyková korektura: Alena Láníčková
Vnitřní úprava: Jiří Matoušek
Sazba: Ctibor Foltýn
Rejstřík: Daniel Štreit
Obálka: Martin Sodomka
Komentář na zadní straně obálky: Martin Domes

Technická spolupráce: Jiří Matoušek, Zuzana Šindlerová, Dagmar Hajdajová
Odpovědný redaktor: Martin Domes
Technický redaktor: Jiří Matoušek
Produkce: Petr Baláš

Computer Press, a. s.,
Holandská 8, 639 00 Brno

Objednávky knih:
http://knihy.cpress.cz
distribuce@cpress.cz
tel.: 800 555 513

ISBN 978-80-251-2965-4
Prodejní kód: K1788
Vydalo nakladatelství Computer Press, a. s., jako svou 3739. publikaci.

Stručný obsah

Část III
Praxe

Část IV
Dodatek

Obsah

Kapitola 9

Kapitola 10

Část III
Praxe

Manželke Kataríne a dcére Dominike

Marian Böhmer, 2010

Předmluva

Během poslední dekády si PHP získávalo stále více popularity. Díky nástupu PHP 5 se mohlo více prosadit i objektově orientované programování a vznikly nové skupiny tříd.

V dnešní době se stávají internetové aplikace stále komplexnějšími. Běží na stovkách až tisících serverů, zpracovávají transakce v objemech miliard eur. Na jedné straně musí být vysoce výkonné a na straně druhé nízkonákladové, co se vývoje týče. Standardní úlohy jako indexování souborů, MVC, šablony nebo tvorba PDF souborů musí být rychle realizovatelné. Použití AJAXu se už vyžaduje od každé internetové aplikace.

Programátor často stojí před rozhodnutím, zda sestrojit požadované komponenty z velkého množství různých dílčích částí (aplikací), nebo na velkou část svých potřeb použít standardizovaný framework. Daný framework samozřejmě nepokryje všechny požadavky, je ale důležité, že neomezí nasazení jiných nástrojů potřebných k dosažení požadovaného cíle.

Jedním z takovýchto Open Source frameworků je i Zend Framework, inicializovaný a dále podporovaný firmou Zend. Za poslední roky se stal Zend Framework frameworkem, který poskytuje řadu předností: silnou průmyslovou podporu firmy Zend, partnery jako IBM, Microsoft, Google, Dojo, Adobe, celosvětovou komunitu vývojářů, kteří neustále přidávají nové komponenty a opravují chyby a také vynikající dokumentaci ve více světových jazycích. K tomu nabízí i prostředky na „ajaxování" internetových aplikací, jednoduchost a z toho plynoucí vysokou produktivitu pro vývojáře při používání jeho komponent a jednoduchou a pro firmy přijatelnou licenci.

Jsem přesvědčen, že Zend Framework stojí na špičce hodnocení Open Source PHP frameworků a dlouho bude. Z tohoto důvodu je smysluplné, aby se s ním každý PHP vývojář seznámil a pokud možno jej i využil pro svoje projekty.

Struktura knihy

Tato kniha se dělí na čtyři části. V první části (kapitoly 1–3) se naučíte základy Zend Frameworku. Poznáte jeho výhody a jeho komunitu a také se dozvíze, jak můžete sestavit Zend Framework pro váš další projekt.

V druhé části (kapitoly 4–16) vám budou představeny všechny komponenty Zend Frameworku. Komponenty jsou tematicky seřazeny a jednotlivé kapitoly můžete číst nezávisle na sobě, avšak jako dobrý základ bych vám doporučil minimálně kapitolu 4. Jestliže se zajímáte o architekturu MVC (Model View Controller), měli byste se nejprve zaměřit na kapitoly 5 až 7.

Třetí část (kapitoly 17–21) vám nabízí více návodů. Naučíte se sestavit a použít modely, rozdělit aplikaci do více modulů a také se dozvíte, jak můžete pomocí Zend Frameworku sestrojit kompletní aplikaci. Další návod vám blíže objasní téma testování Zend Framework aplikací a dá tipy k „testy řízenému" vývoji. Naučíte se také, jak můžete pomocí Zend Frameworku refaktorovat starou PHP 4 aplikaci.

Na konci knihy najdete dodatek s informacemi k návrhovým vzorům a plánované novinky v další verzi Zend Frameworku.

Komu je tato kniha určena

Tato kniha se primárně zaměřuje na začátečníky v Zend Frameworku. Nevyžaduje žádné dosavadní znalosti programování v Zend Frameworku. Měli byste však mít zkušenosti s programováním v PHP a také

by vám nemělo být cizí objektově orientované programování v PHP 5. Jestliže disponujete těmito předpoklady, potom byste měli být, s pomocí této knihy, rychle úspěšní.

Kromě začátečníků se tato kniha obrací v části III s množstvím užitečných tipů také na pokročilejší programátory. Pokud chcete vědět, jak zrealizujete pomocí Zend Framework komponent kompletní aplikaci, nebo chcete-li vědět, jak testovat aplikace nebo jak přepracovat krok po kroku vaše staré fórum z dob PHP 4, potom vám bude tato kniha velmi nápomocná.

Co najdete na přiloženém CD

Na přiloženém CD najdete aktuální verzi Zend Frameworku (v době psaní knihy to byla verze 1.10.3), referenční příručku a API dokumentaci v anglickém jazyce.

Kromě toho na něm najdete všechny výpisy kódu z knihy, které jsou doplněny o další in-line dokumentaci. V některých kapitolách najdete i hotové aplikace, které jsou také na tomto CD a které můžete hned vyzkoušet a především prozkoumat.

ČÁST I

Úvod

KAPITOLA 1

Úvod do Zend Frameworku

Chcete vytvořit novou internetovou aplikaci? Máte už po krk znovu vyvíjet jedny a tytéž komponenty pro každý nový projekt nebo trávit víc času vývojem vlastního frameworku než se věnovat placeným zákaznickým projektům? Potom byste se měli rozhodnout pro nějaký PHP framework.

Přehled PHP frameworků

PHP frameworků je v dnešní době jako máku. Existují velké a malé, rychlé a pomalé, placené a volně šiřitelné, červené a zelené. Jedny jsou odnoží Ruby on Rails, jiné kopiemi známých redakčních systémů (CMS) a jiné znovu zkrachují na vytrvalosti vývojového týmu ještě před vydáním první stabilní verze. Existují frameworky s pevnou strukturou, ale i volné, distribuované jako sbírka komponent. A existuje Zend Framework.

Když v říjnu 2005 zveřejnila firma Zend Technologies Inc., že se v rámci PHP Collaboration projektu vyvíjí nový PHP framework, byly názory mezinárodní PHP komunity velmi různé. Jedni si stěžovali: „Už žádný další PHP framework", jiní se naopak těšili: „Konečně, na toto jsem čekal". Během toho, jak zaměstnanci firmy Zend pracovali na první alfa verzi, neproniklo na veřejnost mnoho informací. Hodně se spekulovalo a předpokládal se zánik většiny PHP frameworků.

V březnu 2006 byla veřejnosti prezentována první verze s označením Pre Alpha Version 0.1.1. Tato velmi raná verze nebyla doporučena pro nasazení do praxe. Na druhé straně ale nabízela některé komponenty, které ještě dnes tvoří jádro Zend Frameworku. Pomocí velkého množství dobrovolných přispěvatelů byl Zend Framework od té doby razantně vylepšován. Další mílové kroky ve vývoji udělala první produktivní verze 1.0 v červenci 2007 a pak verze 1.5 v březnu 2008.

Přispěl Zend Framework k masovému vymírání PHP Frameworků a postaral se o monokulturu? Určitě ne. I v roce 2010 stále existuje množství více či méně vážně braných PHP Frameworků.

Obohatil Zend Framework svět PHP frameworků? V každém případě ano! Tato kniha vám chce pomoct porozumět Zend Frameworku a jejím cílem je, abyste jím byli nadšeni.

Výhody Zend Frameworku

Existuje mnoho dobrých důvodů, proč použít PHP Framework. Ve všeobecnosti jsou to tyto:

- Komponenty pro opakující se úlohy jako databázové dotazy, zpracování šablon, tvorba formulářů, ověřování údajů nebo posílání e-mailů nemusí být pro každý nový projekt znovu napsány a mohou být použity opakovaně.
- Dlouhá doba zaučení se při prvním projektu sice neukazuje jako efektivní, ale s každým dalším projektem poznáváte Zend Framework lépe a lépe a můžete se od té chvíle soustředit na podstatné věci, jako jsou požadavky vašeho klienta nebo implementace přání vašeho týmu.
- Jestliže se raději rozhodnete pro Open Source framework, přesouváte vývoj frameworku směrem ven a máte tak čas na vlastní aplikaci. Komunita se stará o další vývoj frameworku a vy můžete pomocí žádoucích změn a hlášení o chybách ovlivnit jeho evoluci.

Tyto výhody platí pro většinu PHP frameworků, které jsou v současnosti na trhu a jsou dále vyvíjeny. Jaké výhody však nabízí Zend Framework, aby se vyplatilo do něho investovat čas?

- Jedním z důležitých principů Zend Frameworku je „use at will“ (použít podle potřeby). Nikdo není nucen používat jeho komponenty jen určitým způsobem. Potřebujete jen zpracování formulářů, generování PDF souborů a vyhledávání na stránce? Žádný problém. Plánujete zkombinovat `Zend_Controller` se Smarty (*www.smarty.net*), PEAR::MDB2 a PDFlib? Také žádný problém. Nemusíte se hodiny a hodiny prohrabávat ve složitých konfiguračních souborech, abyste vůbec mohli začít. Použijte Zend Framework přesně tak, jak to potřebujete.
- *Velká rozšířenost* – podle statistik je evidovaných více než 10 milionů stažení z oficiálních serverů. To jsou čísla, o kterých mohou ostatní frameworky jen snít.
- *Velké množství přispěvatelů* – více než 500 dobrovolníků z celého světa už přispělo k vývoji, testování, dokumentaci, překladu, konceptu, navrhli nové komponenty nebo ohlašovali či opravovali chyby.
- *Dobrá dokumentace* – už od první verze je Zend Framework dodávaný s referenční příručkou. Žádná nová komponenta nebude zahrnuta do distribuce, dokud nebude obsahovat svou kapitolu v referenční příručce. Ta obsahuje kromě anglického originálu i překlady, např. německý, ruský atd. Kromě toho existuje ke všem komponentám kompletní popis rozhraní (API) vytvořený pomocí `phpdoc`.
- *Solidní testovací základna* – je jedním z elementárních cílů Zend Frameworku už od začátku. Všechny komponenty jsou testovány pomocí jednotkových testů (anglicky Unit Tests) a tyto testy jsou dodávané spolu s frameworkem. Tyto vysoce kladené požadavky na testovatelnost komponent se znovu odráží v tom, že i aplikace založené na Zend Frameworku jsou jednoduše testovatelné.
- *Profesionální podpora* – Zend Technologies není jen eponym pro Zend Framework, ale i hnací síla v pozadí. Kromě dobrovolných vývojářů zaměstnává firma Zend i několik programátorů, kteří mají za úkol výhradně další vývoj Zend Frameworku. Zend Technologies má velký zájem na tom, aby byl Zend Framework neustále zlepšován a rozšiřován.
- *Velké množství uživatelů* – díky velké rozšířenosti Zend Frameworku stoupá pravděpodobnost, že v případě potřeby nebo v tísni najdete nového zaměstnance. V případě, že se tento už setkal s Zend Frameworkem, případně v něm už vytvořil nějaký projekt, rychle se přizpůsobí novému prostředí a může být hned produktivní. Zkuste někdy pro „Georgův PHP výtvor 0.0.5.18“ najít narychlo někoho kompetentního, když je George náhodou nemocen!

- *Velká komunita* – Zend Framework komunita je velká a dobře propojená. Existují tucty blogů a diskuzních skupin, kde diskutují vývojáři z celého světa a kde málokterá otázka zůstane nezodpovězená. Kromě toho píšou někteří kmenoví vývojáři a profesionální uživatelé na svých blozích o nejnovějším vývoji Zend Frameworku.
- *PHP 5, OOP, MVC a rozšiřitelnost* – Protože byl Zend Framework už od začátku vyvíjen pro PHP 5, nemusí být z důvodů zpětné kompatibility vlečena zbytečná břemena. Díky striktnímu nasazení objektově orientovaného programování (OOP) a vybraných návrhových vzorů (např. MVC) je rozšiřitelnost existujících komponent velmi jednoduchá.
- *AJAX a Web 2.*0 – Zend Framework umožňuje už „od výroby" nasazení nejmodernějších internetových technologií. Velmi úzce integrovaný je Dojo Toolkit a mnoho webových služeb nabízí přímé napojení známých Web 2.0 aplikací, například YouTube, Google, Delicious, Flickr nebo Yahoo. Také zpracování a tvorba RSS nebo správa uživatelů přes OpenID je velmi jednoduchá.
- *Licence* – Téma licencování má pro komerční uživatele velký význam. Firmy si už v dnešní době nemohou dovolit nasadit software, u kterého by mohlo dojít k problémům s autorskými právy. Zend Framework je distribuovaný pod novou BSD licencí. Každý, kdo chce přispívat, podepíše příslušnou licenční smlouvu (anglicky Contributor License Agreement, CLA) *http://framework.zend.com/cla* a tím potvrdí, že jeho příspěvek neporušuje práva třetích osob.

Přehled komponent

Abyste si vytvořili přehled o dostupných komponentách, shrnul jsem je pro vás do několika tematických celků. Tyto komponenty budou na základě této kapitoly představeny v části II. Jestliže nepochopíte ihned některé pojmy, nedělejte si s tím žádné starosti. V příslušných kapitolách budou všechny pojmy přesně vysvětleny.

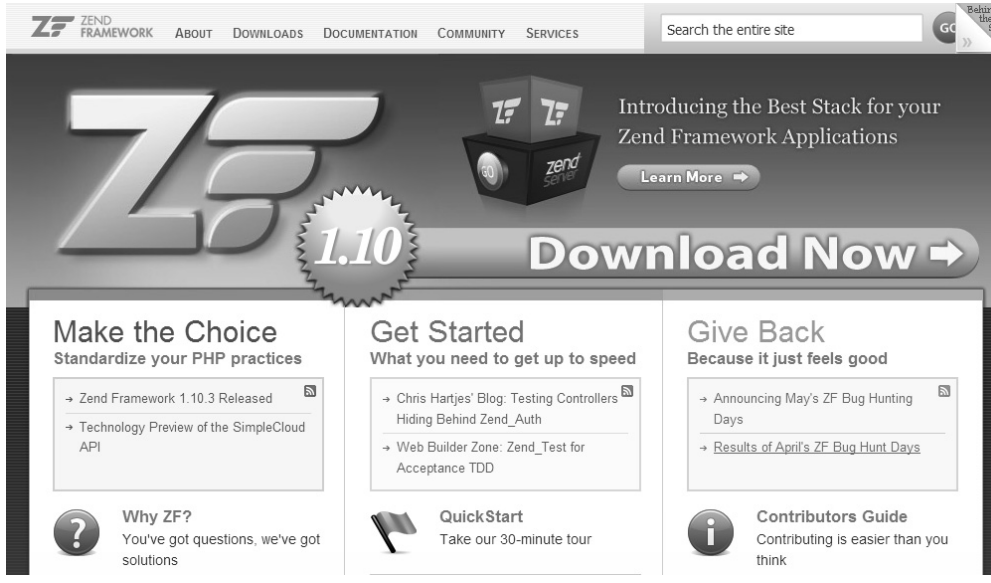
- *Základní komponenty (kapitola 4, Základní komponenty)* – K základním komponentám patří ty komponenty, které najdou uplatnění v každém projektu. Chcete standardizovat zavádění vaší aplikace, načítat třídy, kešovat údaje, používat konfigurace, ukládat objekty s globální platností, posílat e-maily, logovat události, filtrovat a ověřovat údaje? Základní komponenty jsou určené přesně pro tyto účely.
- *Komponenty řadiče (kapitola 5, Řadič)* – Jedněmi z nejdůležitějších komponent jsou komponenty řadiče. Tady najdete hlavně front controller, request a response objekty, směrovač, dispečer a action controller. Dále tu najdete různé pomocné třídy a zásuvné moduly, o které můžete rozšířit část řadiče.
- *Komponenty pohledu (kapitola 6, Pohled*) – Za výstup vaší aplikace jsou zodpovědné komponenty pohledu (View komponenty). Pomocí nich můžete používat view skripty, psát vlastní action helpery nebo integrovat různé systémy šablon. Vývojáři také mysleli na použití vícestupňového layoutu.
- *Databázové komponenty (kapitola 7, Databáze)* – V dnešní době se už sotva nějaká webová aplikace obejde bez napojení na databázi, a proto nabízí Zend Framework množství databázových adaptérů, dotazovací nástroje a implementaci obou dvou návrhových vzorů, Table Data Gateway a Row Data Gateway.
- *Komponenty na správu uživatelů (kapitola 8, Správa uživatelů)* – Bez uživatelů nemá ani ta nejlepší aplikace žádný význam, a proto poskytuje Zend Framework komponenty pro autorizaci, autentizaci a správu uživatelských relací (anglicky Sessions). Ani externí napojení přes OpenID, InfoCard, nebo LDAP nepředstavují žádnou překážku.
- *Zpracování formulářů (kapitola 9, Formuláře)* – Kromě uživatelů jsou ve Web 2.0 aplikacích důležité i jejich vstupy. Proto poskytuje Zend Framework komponenty pro tvorbu, konfiguraci a zpracování formulářů. Jednoduše realizovatelný je i upload souborů a zabezpečení pomocí CAPTCHA.

- *Internacionalizace a lokalizace (kapitola 10, Internacionalizace a lokalizace)* – Komponenty pro internacionalizaci a lokalizaci vás podpoří při nasazení tzv. Locales, při překladu textů a při manipulaci s datem, měnou a měřicími jednotkami.
- *Vyhledávání (kapitola 11, Vyhledávání pomocí Lucene)* – Zend Framework vám nabízí Lucene, implementaci Apache vyhledávače založenou na PHP. Tím si můžete na vašich stránkách vytvořit fulltextové vyhledávání.
- *Tvorba PDF dokumentů (kapitola 12, Tvorba PDF dokumentů)* – Chcete nabídnout uživateli vaší stránky příspěvek v tištěné podobě nebo generovat faktury? Ani toto není s Zend Frameworkem žádný problém.
- *AJAX a Zend Framework (kapitola 13, Zend Framework a JavaScript*) – Zend Framework vám pomůže při tvorbě frontendu vaší aplikace. Pomocí úzké integrace DOJO Toolkitu je tvorba AJAX aplikací, které interagují s vaším serverem, bezproblémová. JQuery je taktéž podporováno.
- *Komponenty pro webové služby (kapitola 14, RSS a webové služby)* – Zend Framework poskytuje množství komponent pro webové služby. `Zend_Gdata` komponenty pro YouTube, Google Base nebo Google Calendar představují Googlem oficiálně podporovaná napojení na Google webové služby. Kromě toho nabízí Zend Framework podporu i pro Yahoo, Delicious, Amazon, Akismet, Technorati, Flickr anebo LiveDocx.
- *Komponenty pro komunikaci (kapitola 15, Komunikace)* – Pro komunikaci s jinými webovými servery poskytuje Zend Framework různé komponenty. Chcete přes HTTP protokol přistupovat k jinému webovému serveru, vytvořit SOAP rozhraní nebo posílat požadavky pomocí XML-RPC? Se Zend Frameworkem to není žádný problém.
- *Ostatní komponenty (kapitola 16, Ostatní komponenty)* – Kromě těch právě jmenovaných nabízí Zend Framework i méně často používané komponenty, například pro účely ladění, zpracování URI atd.

Užitečné odkazy

Součástí prostředí Zend Framewoku jsou i mnohé komunity, internetové stránky a aplikace, které byste měli při práci s ním podpořit. V následujícím seznamu najdete užitečné odkazy, které byste měli navštívit a hned si je i přidat do záložek.

- *http://framework.zend.com/*
  Oficiální stránka Zend Frameworku.
- *http://framework.zend.com/download/*
  Tady najdete aktuální stabilní verzi i download archiv.
- *http://framework.zend.com/manual/en/*
  Oficiální referenční příručka pro všechny komponenty rozdělená podle verzí frameworku a dostupná v několika jazykových mutacích.
- *http://framework.zend.com/apidoc/core/*
  API dokumentace; výběr komponent najdete v pravém horním rohu.
- *http://framework.zend.com/about/faq/*
  Často kladené otázky (FAQ).
- *http://framework.zend.com/wiki/*
  Wiki pro vývojáře a přispěvatele, ale i pro ty, kteří se jimi chtějí stát.
- *http://framework.zend.com/wiki/display/ZFPROP/Home/*
  Wiki pro všechny návrhy; obsahuje taktéž diagram, který znázorňuje životní cyklus návrhu.



Obrázek 1.1: Oficiální stránka Zend Frameworku

- *http://framework.zend.com/issues/browse/ZF/*
  Issue Tracker – zde najdete všechny otevřené chyby, problémy nebo přání rozšíření a také si můžete prohlédnout stav jednotlivých problémů.
- *http://www.zend.com/en/resources/webinars/framework/*
  Zend Technologies pořádá v nepravidelných intervalech tzv. webináře k tématu Zend Framework. Jedná se o semináře, které si můžete na počítači prohlédnout naživo nebo po skončení ze záznamu.
- *http://www.thomasweidner.com/*
  Thomas je šéfem týmu zodpovědného za komponenty pro internacionalizaci a lokalizaci.
- *http://weierophinney.net/matthew/*
  Blog jednoho z kmenových tvůrců a v současnosti (jaro 2010) i šéfa celého vývojového týmu Zend Frameworku.
- *http://www.zendframework.cz/*
  Český portál patřící Zend Frameworku vedený Martinem Hujerem a Tomášem Fejfarem.
- *http://forum.zendframework.cz/*
  České fórum k Zend Framemwroku.
- *http://www.zendframework.sk/*
  Slovenský portál věnovaný Zend Frameworku. V době psaní knihy ve výstavbě.
- *http://akrabat.com/*
  Rob Allen píše pravidelně o Zend Frameworku.

Pravidla programování v Zend Frameworku

Všechny výpisy kódů v této knize stejně jako na přiloženém CD jsou v souladu s pravidly programování v Zend Frameworku. Ty nejdůležitější jsou shrnuty v této kapitole.

- V souborech, které obsahují jenom PHP kód, nesmí být použita ukončovací značka (`?>`). Tato není překladačem PHP vůbec vyžadována. Jejím vynecháním zabráníte tomu, že nedopatřením budou odeslané prázdné znaky do prohlížeče uživatele ještě před samotným obsahem.
- Odsazení je nutné realizovat vždy pomocí čtyř mezer, nikdy ne pomocí tabulátoru.
- Jednotlivé řádky by měly obsahovat maximálně 80 znaků. V nepočetných výjimečných případech je též povoleno maximálně 120 znaků.
- Označení tříd by mělo odrážet jejich umístění v adresářové struktuře a smí obsahovat pouze alfanumerické znaky. Třídu `Zend_Controller_Action` můžete tedy najít v souboru `Zend/Controller/Action.php`.
- V názvech souborů jsou přípustné jen alfanumerické znaky, znak podtržítka (_) a znak pomlčky (-). Všechny ostatní znaky, například mezery, jsou zakázané.
- Názvy funkcí a metod musí obsahovat jen alfanumerické znaky a jsou udávané v tzv. „camelCase" formátu. Například `getBookById()`, `myLittleSecretMethod()` nebo `setOrders()`.
- Názvy proměnných musí obsahovat jen alfanumerické znaky a také jsou udávané v tzv. „camelCase" formátu.
- Názvy konstant musí obsahovat jen alfanumerické znaky a znak podtržítka (_). Je třeba užívat velká písmena.
- Znak apostrofu (') je upřednostňovaný před znakem uvozovek (").
- Při víceřádkovém zřetězení by měly být znaky zřetězení (.) umístěny pod znakem rovnosti (=)

```
$sql = "SELECT `id`, `name` FROM `items` "
     . "WHERE `type` = 'book' "
     . "ORDER BY `name` ASC ";
```

- Třídy mají mít vždy blok dokumentace. Otevírací množinová závorka ({) se nachází pod názvem třídy na samostatném řádku na prvním místě. Kód programu v rámci třídy musí být odsazen o čtyři mezery.
- Stejně jako u tříd se i u metod nachází otevírací množinová závorka ({) na samostatném řádku.
- U argumentů (proměnných) a metod (funkcí) třídy se vždy udává jejich viditelnost `private`, `protected` nebo `public`.
- U výrazů jako například `if elseif else` jsou otevírací množinové závorky ({) vždy na konci řádku podmínky. Ukončovací množinová závorka (}) se nachází vždy pod spouštěcí `if` podmínkou.
- Také u výrazu `switch` je otevírací množinová závorka ({) umístěna na konci řádku podmínky. `Case` bloky jsou vycházeje od výrazu `switch` odsazeny o čtyři mezery a vždy musí být udaný `default` blok. Programové kódy v rámci `case` bloku včetně výrazu `break` jsou vycházeje od výrazu `case` odsazeny o čtyři mezery.

Kompletní sestavu pravidel programování (Coding Standards) pro Zend Framework najdete na adrese *http://framework.zend.com/manual/en/coding-standard.html*. Chtěl bych vás upozornit na alternativní možnost definice polí. V případě polí jsou povoleny dvě možnosti zápisu.

První možnost je následující:

```
$sampleArray = array('firstKey'  => 'firstValue',
                     'secondKey' => 'secondValue');
```

V této knize, stejně jako na přiloženém CD, najdete alternativní možnost zápisu, která používá trochu odlišné formátování.

```
$sampleArray = array(
    'firstKey'  => 'firstValue',
    'secondKey' => 'secondValue',
);
```

Struktura Zend Frameworku

Zend Framework se skládá z velkého počtu komponent, které se opět skládají z dalších dílčích komponent. Tyto komponenty a dílčí komponenty sestávají z více tříd, přičemž každá je uložena ve vlastním souboru. Tyto soubory jsou rozděleny do víceúrovňové stromové struktury. Všechny soubory se nacházejí v adresáři `Zend/`, respektive v nějakém podadresáři adresáře `Zend/`. Názvy tříd odrážejí pozici souboru v rámci adresářové struktury. Tím můžete za základě názvu třídy okamžitě rozpoznat, kde se daná třída nachází v rámci adresářové struktury Zend Frameworku. Například:

`Zend_Form_Element_Text` je uložený v `Zend/Form/Element/Text.php`

`Zend_Acl` je uložený v `Zend/Acl.php`

`Zend_Db_Adapter_Pdo_Mysql` je uložený v `Zend/Db/Adapter/Pdo/Mysql.php`

Čím víc se Zend Frameworkem budete zabývat, tím větší bude pravděpodobnost, že narazíte na komponenty začínající `ZendX` nebo `ZendL`.

Komponenty začínající `ZendX` nejsou oficiálně podporované týmem Zend Frameworku. (X znamená „extra".) Jsou zpravidla provozované komunitou podle konvencí Zend Frameworku. Příkladem jsou například komponenty `ZendX_JQuery` nebo `ZendX_Whois`. Pokud se nějaká komponenta nachází ve stadiu vývoje a zatím nebyla oficiálně zahrnuta do Zend Frameworku, začíná na `ZendL`, kde L znamená laboratoř (anglicky Laboratory). Tato komponenta zatím není úplná nebo k ní může chybět dokumentace či testy. Není proto doporučeno takovéto komponenty používat na produkčních serverech. Když je daná komponenta kompletní a oficiálně přijata do Zend Frameworku, bude její jméno změněno, například. `ZendL_Service_Book` na `Zend_Service_Book`.

Postupně se v knize dozvíte, jak můžete rozšířit Zend Framework pro své vlastní potřeby. Pro své rozšíření byste si měli zvolit vlastní prefix. MaBo e-shop může například svoje rozšiřující třídy začínat pomocí prefixu `Mabo`. Rozšíření třídy `Zend_Form` se bude potom jmenovat `Mabo_Form`.

Kromě toho používá třída `Zend_Loader` tuto jmennou konvenci na to, aby mohla dodatečně zavést nedefinované třídy na základě jejich názvu. Víc se o komponentě `Zend_Loader` dozvíte v kapitole 4, Základní komponenty.

MaBo e-shop

Možná se ptáte, co dělá nějaký e-shop v úvodní kapitole knihy o Zend Frameworku. Důvod je velmi jednoduchý. Protože každá kniha žije ze svých příkladů, rozhodl jsem se jako pomůcku při vysvětlování zvolit právě e-shop zabývající se prodejem knih. Nebude to sice kompletní aplikace, ale mnoho příkladů v knize se bude vztahovat právě na něj. Toto do velké míry ulehčí vysvětlování.

Referenční příručka a dokumentace k API

Tato kniha nechce nahradit referenční příručku. To znamená, že tu nenajdete žádné kompletní reference metod, parametrů a konstant pro všechny komponenty. Takováto reference by byla už při vydání knihy zastaralá. Jestliže budete mít někdy otázky ke komponentám, které v této knize nebudou zodpovězeny, nahlédněte do referenční příručky.

Tým Zend Frameworku klade vysoké nároky na kvalitu dodávané dokumentace, proto tam často najdete odpovědi na své otázky. K dispozici máte online verzi referenční příručky na adrese *http://framework.zend.com/manual/en/* nebo použijte její offline verzi, kterou si můžete stáhnout do počítače, případně použijte tu z přibaleného CD.

Pokud chcete přesně vědět, jaké parametry potřebuje určitá metoda, jaké údaje bude vracet nebo jestli může vyvolat výjimku, potom nahlédněte do API dokumentace. Tu najdete také online na *http://framework.zend.com/apidoc/core/*; v případě potřeby si ji můžete stáhnout nebo použit tu z přiloženého CD.

Jestliže máte i po tom všem stále nezodpovězené otázky, můžete nakonec nahlédnout do zdrojových kódů jednotlivých komponent. Tam najdete vedle dokumentačních bloků – podle komplexnosti metod – vždy i objasňující inline dokumentaci.

Shrnutí

V této první kapitole jste se dozvěděli něco málo o historii Zend Frameworku. Poznali jste jeho přednosti, které hovoří pro jeho použití. Dále jste získali množství užitečných odkazů, s kterými se můžete pohybovat v komunitě Zend Frameworku. Kromě toho jste se dozvěděli, zda je tato kniha pro vás vůbec určená a jakým stylem je tvořena.

V následujících kapitolách se dozvíte více o MVC (Model View Controller) architektuře, která je základem Zend Frameworku, a také o tom, co udělat, abyste mohli Zend Framework začít používat.

KAPITOLA 2

Instalace Zend Frameworku

Instalace Zend Frameworku není vůbec složitá. Čím víc projektů postavených na Zend Frameworku vytvoříte, tím se stává jeho instalace jednodušší. V této kapitole se naučíte, odkud a jak můžete Zend Framework stáhnout a jak ho nainstalujete. Dále poznáte možné varianty adresářové struktury a dozvíte se, jak můžete použít moduly a integrovat vlastní rozšíření. Jako malý bonus zde najdete pár tipů, jak nakonfigurovat webový server Apache.

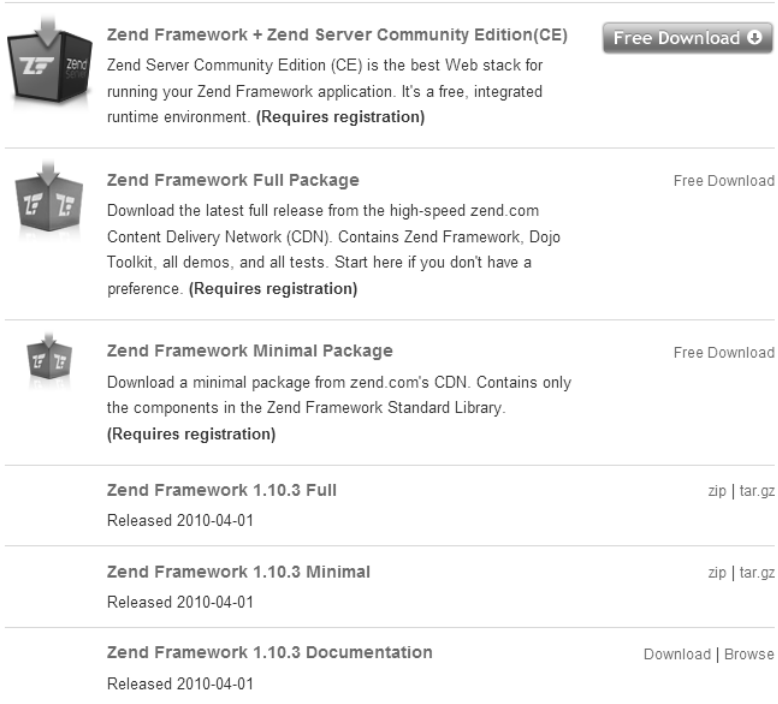
Zdroje Zend Frameworku

Aktuální verzi Zend Frameworku můžete stáhnout z internetové stránky Zend Frameworku *http://framework.zend.com/download/latest*. Můžete si vybrat ze dvou možností. Kompletní verze obsahuje Zend Framework, Dojo Toolkit, názorné ukázky (Demos) a všechny testy. Minimální verze obsahuje jen Zend Framework. Jestliže stahujete Zend Framework poprvé, měli byste si na to vyhradit víc času a stáhnout kompletní verzi.

Jestliže použijete některý z odkazů, u kterých jsou obrázky, budete přesměrováni na stránku Zend Technologies a musíte se tam v případě potřeby zaregistrovat. Má to tu výhodu, že budete mít aktuální informace z dílny Zend a samozřejmě o Zend Frameworku. Pokud chcete stáhnout data bez registrace, použijte odkazy na konci stránky.

Zend Framework poskytuje oficiální implementaci webových služeb (anglicky Web Services) pro některá nejdůležitější Google API. Z tohoto důvodu je k dispozici ke stažení (*http://framework.zend.com/download/webservices*) samostatný balík pro `Zend_Gdata` komponenty. `Zend_Gdata` komponenty jsou samozřejmě obsaženy i v normálním balíku Zend Frameworku. Totéž platí i pro komponenty `Zend_InfoCard`, pro které také existuje zvláštní balíček.

Pro všechny, kteří se zajímají o aktuální stav vývoje, je k dispozici ke stažení (*http://framework.zend.com/download/snapshot*) denně aktualizovaná verze (Snapshot). Kromě toho máte k dispozici na adrese *http://framework.zend.com/download/archives* archiv jednotlivých verzí. U každé verze najdete i odpovídající odkaz ke stažení dokumentace. Jedinou výjimkou jsou denně aktualizované verze, kde je dokumentace součástí balíčku.



Obrázek 2.1: Download Zend Frameworku

Požadavky na webové technologie

Tato kniha předpokládá, že už máte na svém počítači nainstalovaný webový server (například Apache) kromě toho máte zřízené PHP 5 a nějakou databázi (například SQLite nebo MySQL). Po zavolání následujícího skriptu byste měli dostat informace, že instalace byla úspěšná, a můžete začat pracovat.

```
phpinfo();
```

Zaměřte se přitom hlavně na odstavce popisující konfiguraci Apache serveru a MySQL. Pro podrobnější návod na instalaci a používání PHP a MySQL bych vám doporučil tuto knihu:

> *PHP 6, MySQL, Apache - Vytváříme webové aplikace*
> *http://knihy.cpress.cz/k1698*

Další předpoklady, které musí být splněny pro použití Zend Frameworku, jsou přehlédnutelné. Potřebná je verze PHP 5.2.4 nebo vyšší.

Jestliže používáte server Apache a chcete přepsat „klasické" URL na URL optimalizované pro vyhledávače, potom musíte aktivovat modul `mod_rewrite`. Další informace o tom, jak to udělat, najdete v kapitole 3, Konfigurace Apache serveru.

Totéž platí i pro ostatní typy jako například Microsoft IIS nebo Lighttpd, u kterých také potřebujete moduly na přepisování URL, například ISAPI_Rewrite nebo IIRF pro IIS.

S PHP rozšířeními je to trochu komplikovanější. PHP rozšíření `ctype`, `pcre`, `Reflection`, `session` a `SPL` musí být v každém případě aktivovaná, protože jsou používána všemi důležitými komponentami. Tyto jsou ve většině PHP instalací už aktivované. Kromě toho vyžadují některé komponenty další PHP rozší-

ření. Například použití komponenty `Zend_Db_Adapter_Pdo_Mysql` (PDO adaptér pro MySQL) vyžaduje mimo jiné i rozšíření `pdo` a `pdo_mysql`.

V případě, že vaše aplikace skončí neočekávaným chybovým hlášením, které kritizuje nedostatek rozšíření, nahlédněte do referenční příručky (*http://framework.zend.com/manual/en/requirements.introduction.html*). Vývojáři Zend Frameworku tam pro vás připravili seznam, ve kterém se dozvíte, které komponenty potřebují která rozšíření. Jestliže u vás nastane problém s chybějícími rozšířeními, PHP manuál (*http://www.php.net/manual/en/install.php*) vám poskytne východisko, jak chybějící rozšíření nainstalovat, případně aktivovat.

Adresářová struktura

Než začnete s instalací, musíte se ještě zabývat tématem adresářové struktury.

Jednou z velkých výhod Zend Frameworku je jeho flexibilita. Teoreticky můžete své soubory ukládat libovolně podle abecedy nebo úplně bez řazení. Přece jen byste je ale měli řadit a udržovat mezi nimi přehled.

Tvůrci Zend Frameworku doporučují adresářovou strukturu, které byste se podle možností měli držet.

Vytvoření adresáře pro projekt

Pro každý nový projekt musíte vytvořit nový adresář. Do něho ukládáte všechny soubory a adresáře pro váš projekt tak, že tento je dostupný prostřednictvím webového serveru. Následující výklad se vztahuje na webový server Apache. V případě, že používáte nějaký jiný typ, musíte potřebné kroky přizpůsobit.

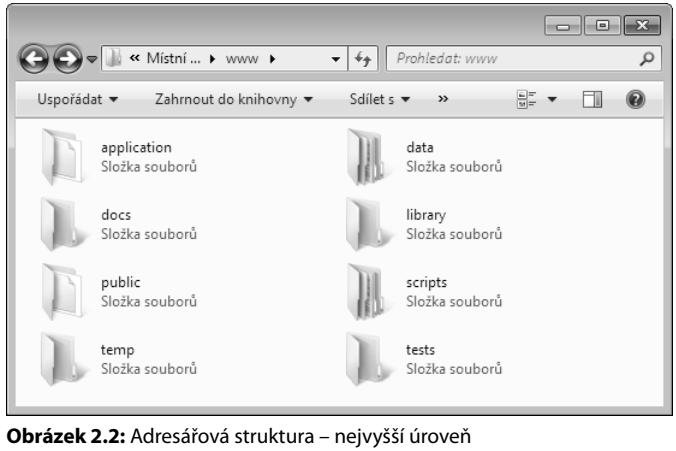
Zpravidla nevyvíjíte přímo na serveru, kde bude aplikace v praxi nasazená, ale na vývojovém serveru. Tam také většinou nevyvíjíte jen jeden projekt, ale vyvíjíte jich více. Zjistěte si proto, který adresář máte nastavený jako *DocumentRoot*. Tento se může v závislosti na použitém operačním systému lišit, například `c:\wamp\apache\htdocs` pro Windows nebo `/var/www/localhost/htdocs` pro Gentoo Linux.

Jestliže jste identifikovali váš *DocumentRoot* adresář, vytvořte v něm adresář pro váš projekt.

Adresářová struktura – nejvyšší úroveň

V adresáři projektu, který jste právě vytvořili, vytvořte následující adresáře z obrázku 2.2, které tvoří nejvyšší úroveň.

- `application/` – Tento adresář je jádrem vaší aplikace a obsahuje konfiguraci, řadiče, formuláře, modely, moduly, layouty, pohledy i soubor `Bootstrap.php`.



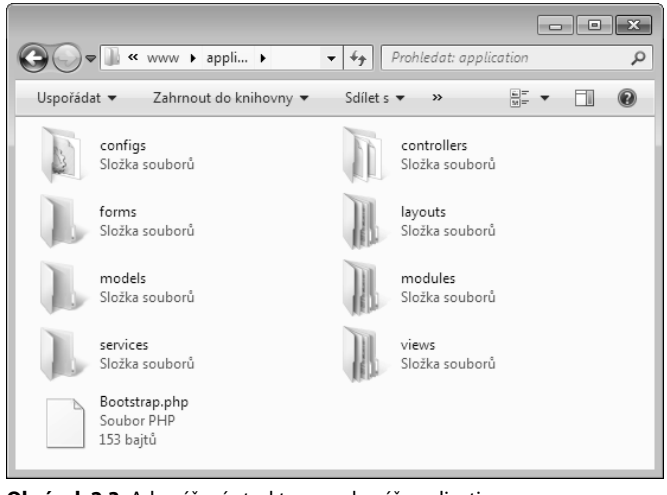
Obrázek 2.2: Adresářová struktura – nejvyšší úroveň

- `data/` – Tento adresář obsahuje všechny údaje z aplikace, které vznikly během jejího běhu. Patří sem indexy, logy, sessions atd.
- `docs/` – Sem můžete uložit dokumentaci k vaší aplikaci.
- `library/` – V tomto adresáři je umístěn Zend Framework a je také určen pro vlastní rozšíření.
- `public/` – Tento adresář obsahuje soubory, které mají být veřejně dostupné přes webový server, tedy obrázky, CSS a JavaScript soubory a také soubor `index.php`.
- `scripts/` – Tento adresář je určen například pro úlohy cronu nebo build skripty.
- `temp/` – Jak už název napovídá, je tento adresář určen pro dočasné soubory.
- `tests/` – Do tohoto adresáře byste měli ukládat všechny své testy (Unit Tests).

Možná se vám význam některých adresářů nezdá na první pohled důležitý, ale navzdory tomu byste měli všechny adresáře vytvořit pro zajištění jednotné struktury pro budoucí rozšíření.

Adresářová struktura – adresář application

Na obrázku 2.3 vidíte další dělení adresáře `application`. Ten obsahuje jádro vaší aplikace. Zde se nachází i soubor `Bootstrap.php`.



Obrázek 2.3: Adresářová struktura – adresář application

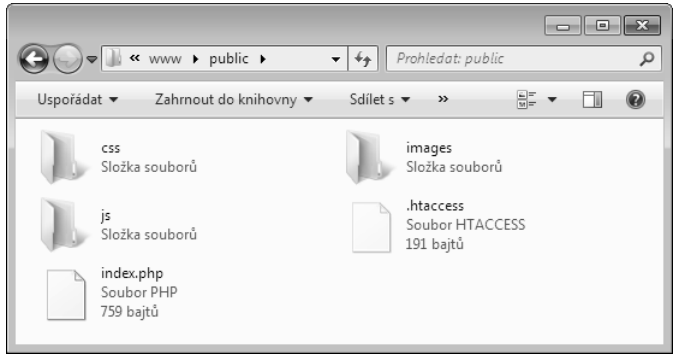
- `configs/` – Do tohoto adresáře ukládejte konfigurační soubory vaší aplikace.
- `controllers/` – Všechny řadiče mají být uloženy v tomto adresáři. Kromě toho se v něm nachází i podadresář `helpers/` pro action helpery specifické pro vaši aplikaci.
- `forms/` – Obsahuje všechny soubory s formuláři.
- `layouts/` – Jestliže používáte layouty, uložte jejich soubory do tohoto adresáře. Tento adresář obsahuje ještě další tři podadresáře, `filters/`, `helpers/` a `scripts/`, pro filtry, helpery a skripty.
- `models/` – Vaše modely ukládejte do tohoto adresáře.
- `modules/` – Tento adresář je používaný pro moduly. Více se o adresářové struktuře při používání více modulů dozvíte později v této kapitole, v části Použití více modulů.
- `services/` – Tento adresář je určen pro webové služby, které jsou specifické pro vaši aplikaci a které budou jí nabízeny. Také je určený pro implementaci servisní vrstvy (anglicky Service Layer) pro modely.

- `views/` – Všechny skripty pohledů (anglicky View Scripts) a šablony (anglicky Templates) jsou uloženy v tomto adresáři. Obsahuje i další tři podadresáře, `filters/`, `helpers/` a `scripts/`, pro filtry, helpery a skripty.
- `Bootstrap.php` – Do nástupu komponenty `Zend_Application` byly v tomto souboru inicializované různé komponenty, například `Zend_Config`, `Zend_Db` nebo `Zend_View`. S nástupem komponenty `Zend_Application` se vaše aplikace obejde i bez tohoto souboru. Více se o souboru `Bootstrap.php` dozvíte v kapitole 3.

I zde platí: Vytvořte všechny adresáře, i když na začátku vašeho projektu zůstanou prázdné.

Adresářová struktura – adresář public

Na obrázku 2.4 je znázorněna další struktura adresáře `public/`. Obsahuje všechny soubory, které mají být veřejně dostupné přes webový server.



Obrázek 2.4: Adresářová struktura – adresář public

- `css/` – Všechny css soubory leží v tomto adresáři.
- `images/` – Obrázky pro vaši aplikaci patří do tohoto adresáře.
- `js/` – Sem patří JavaScript soubory. Jestliže používáte Dojo Toolkit, je tento adresář to správné místo pro uložení jeho souborů.
- `.htaccess` – Soubor obsahuje konfigurační nastavení pro server Apache. Více o tomto souboru najdete v kapitole 3, Konfigurace serveru Apache.
- `index.php` – Úlohou tohoto souboru je nastavit PHP prostředí a odstartovat front controller. Více informací o tomto souboru najdete v kapitole 3, Soubor index.php.

Na další dělení adresářů `css/`, `images/` a `js/` neexistují žádná další doporučení ani pravidla.

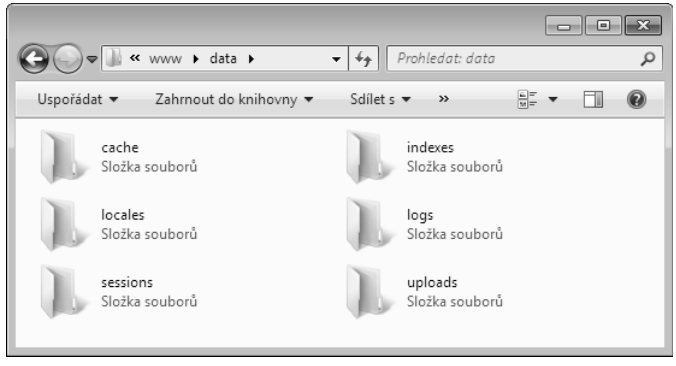
Adresářová struktura – adresář data

Na obrázku 2.5 je znázorněna další struktura adresáře `data/`. Obsahuje všechny údaje z aplikace, které budou vytvořeny a potřebné během jejího běhu.

- `cache/` – Obsahuje všechny cache soubory, které byly vytvořeny například komponentou `Zend_Cache`.
- `indexes/` – Obsahuje indexy, které vytvoří komponenta `Zend_Search_Lucene`.
- `locales/` – Obsahuje soubory s překlady pro `Zend_Translate`.
- `logs/` – Jestliže používáte komponentu `Zend_Log`, můžete do tohoto adresáře ukládat soubory s logy.

- `sessions/` – Do tohoto adresáře můžete ukládat všechny údaje ohledně uživatelských relací (anglicky Sessions).
- `uploads/` – Slouží pro dočasné uložení uploadovaných souborů pomocí `Zend_File_Transfer`.

Tyto adresáře jsou jen doporučením. Do adresáře `data` můžete ukládat i jiné údaje, například SQLite databáze.



Obrázek 2.5: Adresářová struktura – adresář data

Vzorová adresářová struktura

Doufám, že po poslední kapitole o adresářové struktuře nejste úplně zmatení a že budete ve čtení pokračovat dál. Adresářová struktura vypadá na první pohled dost komplexně a právě pro začátečníky v Zend Frameworku nemusí být smysl jednotlivých adresářů hned zřejmý.

S postupem času si však danou adresářovou strukturu osvojíte, a pokud se jí budete držet, velmi tím ulehčíte novým členům týmu nástup na váš projekt. Jestliže byste i vy měli přistoupit k nějakému už běžícímu projektu, budete velmi šťastní, pokud se kolegové drželi této struktury při vytváření projektu.

Tip

Jako pomůcku pro začátečníky najdete na přiloženém CD malý soubor, který obsahuje probranou adresářovou strukturu i zmíněné soubory a také některé základní soubory, například `IndexController` a `ErrorController`. Jestliže začínáte nový projekt, stačí daný soubor s předlohou jen rozbalit a můžete hned začít pracovat. Soubor se nachází na CD v adresáři pro druhou kapitolu a je pojmenovaný `adresarovastruktura.zip`.

Instalace

Po úspěšném sestavení adresářové struktury z předcházející části se instaluje Zend Framework velmi jednoduše. Po stažení rozbalte framework do libovolného adresáře kdekoliv na disku vašeho počítače. Vytvoří se nový adresář `ZendFramework-x.y.z/`, kde x, y a z udávají verzi Zend Frameworku. Přesuňte se do tohoto nově vytvořeného adresáře. V něm najdete adresář `library/`, do kterého se též přesuňte.

Označte adresář `Zend/` a překopírujte ho včetně všech souborů a podadresářů v rámci adresáře `library/` do vašeho projektového adresáře. To je vše.

Jestliže chcete realizovat více než jeden projekt, můžete si Zend Framework uložit na nějaké centrální místo mimo váš projektový adresář. Potom vám stačí jenom jedna kopie Zend Frameworku. I přesto vám však doporučuji uložit soubory Zend Frameworku pro každý projekt do adresáře `library/Zend/`. Jako

další alternativu můžete všechny vámi používané verze Zend Frameworku spravovat na jednom místě a propojit je s vaším projektovým adresářem pomocí symbolického odkazu. Toto ale také není podporované všemi operačními systémy. Starší verze Windows tuto možnost nenabízí (*http://cs.wikipedia.org/wiki/Symbolický_odkaz*).

Určitě přijde den, kdy budete chtít vy nebo jeden z vašich zákazníků přejít s některým z projektů na novou verzi. Všechny ostatní projekty zůstanou na původní verzi. Může se to stát z různých důvodů a vy nemusíte mít čas otestovat tucty projektů na novou verzi a v případě potřeby je měnit. Dále to mohou být například i technické důvody. V případě změny verze z 1.x na 2.x se může změnit chování jednotlivých komponent. Když nastane tento den, vzpomenete si na moje slova. Budete šťastní, že jste pro každý projekt použili samostatnou kopii Zend Frameworku, nebo mě budete do smrti nenávidět za to, že jsem vás dostatečně neodradil od nápadu šetření místa na disku.

Upozornění

A propos: šetření místa na disku. Možná časem přijdete i na nápad očesat Zend Framework o pár komponent. Budete si myslet, že ty komponenty které nepotřebujete, můžete smazat a tím ušetřit pár megabajtů místa na disku. Důrazně vás upozorňuji: Nedělejte to! Dejte od toho ruce pryč a okamžitě na tu myšlenku zapomeňte!

Většina komponent na sobě nezávisí (s výjimkou `Zend_Exception`, která souvisí se všemi komponentami). Existují však výjimky a nikdy nevíte, jaké nepředvídatelné chyby se mohou objevit, protože jedna komponenta potřebuje jinou. V referenční příručce najdete dobrý přehled o závislostech mezi jednotlivými komponentami (*http://framework.zend.com/manual/en/requirements.dependencies.html*), kde se dozvíte, která komponenta s kterou souvisí.

Tip

Kromě manuálního způsobu instalace poskytuje Zend Framework od verze 1.8 i komponentu `Zend_Tool`. Tato komponenta instalaci značně ulehčí. Více se o komponentě `Zend_Tool` dozvíte v kapitole 4, Základní komponenty.

Vlastní rozšíření Zend Frameworku

Dříve nebo později nastane situace, kdy budete chtít implementovat vlastní rozšíření Zend Frameworku. Může to být například potřeba použití vlastního zásuvného modulu, psaní vlastních filtrů a validátorů nebo změna či rozšíření některé z komponent Zend Frameworku. Také můžete napsat vlastní webovou službu nebo úplně novou komponentu zpracování nákupního košíku.

Než tato rozšíření neplánovaně uložíte někam do svého projektového adresáře, měli byste vytvořit speciální adresář, který rozšiřuje funkce Zend Frameworku. Ten byste měli vytvořit paralelně k adresáři `Zend/` v adresáři `library/`. Pojmenovat ho můžete principiálně libovolně, doporučuji vám však pojmenovat ho vaším jménem, jménem vaší firmy, případně jménem projektu nebo zákazníka. Zvolený název musí být jednoznačný, sestávající jen z písmen a začínající velkým písmenem. Pro příklady v této knize bude použit název `Mabo`.

Odtéto chvíle budou všechna rozšíření uložena v adresáři `library/Mabo/`. Vlastní rozšíření třídy `Zend_Form` tedy bude `Mabo_Form` uložené v souboru `Form.php` a nacházející se v adresáři `library/Mabo/`. Novým filtrem na odstranění gramatických chyb může být například třída `Mabo_Filter_Spelling`. Bude uložená v adresáři `library/Mabo/Filter` v souboru `Spelling.php`.

Protože adresář `library/` musí být obsažen v `include_path`, můžete používat své vlastní rozšíření pomocí komponenty `Zend_Loader` bez jakékoliv další konfigurace. Kromě toho vždy víte, kam musíte své rozšiřující třídy uložit.

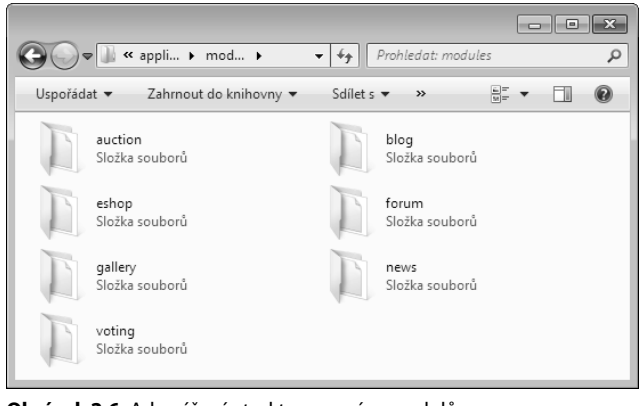
> **Upozornění**
>
> Tento výše zmíněný adresář je určený jen pro třídy rozšiřující Zend Framework, které můžete v případě potřeby znovu použít i v jiných projektech. Žádné komponenty (řadiče, modely, formuláře atd.), které se týkají konkrétně vaší aplikace, nepatří do tohoto adresáře. Na tyto účely je určen adresář `application/`.

Použití více modulů

Představte si například stránku, která obsahuje blog, e-shop, fórum, fotogalerii, aukci, anketu nebo novinky. Takovéto stránky ve skutečnosti existují. Vznikají a zanikají každý den. Pro každý z těchto oborů potřebujete vlastní řadiče, modely, pohledy i konfigurační soubory. Jestliže byste na toto všechno chtěli použít zatím známou adresářovou strukturu, pravděpodobně byste rychle ztratili přehled.

Zend Framework vám proto poskytuje podporu v podobě modulů. Modul je v principu shrnutí nebo skupina řadičů, modelů, pohledů a podobně, které spolu logicky souvisí. Modul pro blog bude například obsahovat všechny řadiče, modely, pohledy a podobně, které s ním souvisí. Jestliže jste si vytvořili vlastní blogovací systém postavený na Zend Frameworku a byl sestaven modulárně, můžete ho znovu použít i pro jiné projekty.

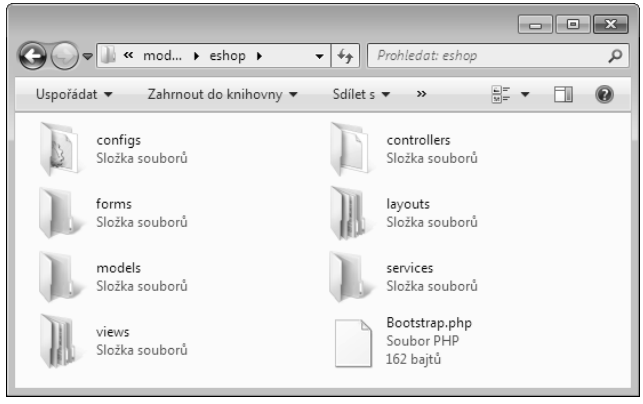
Své moduly můžete ukládat do adresáře `application/modules/`. Pro každý modul vytvořte vlastní adresář. Pro příklad uvedený výše bude daný adresář vypadat tak, jak je zobrazeno na obrázku 2.6.



Obrázek 2.6: Adresářová struktura po více modulů

Pro každý z těchto modulů můžete zopakovat adresářovou strukturu z nadřazeného adresáře `application/` – až na jednu výjimku. Tento adresář neobsahuje za normálních okolností adresář `modules/`. Obrázek 2.7 ukazuje, jak bude vypadat další dělení tohoto adresáře.

Adresář `views/` znovu obsahuje podadresáře `filters/`, `helpers/` a `scripts/` pro filtry, helpery a skripty pohledů a adresář `controllers/` a podadresář `helpers/` pro action helpery.



Obrázek 2.7: Další dělení adresáře modules/

Tip

Na přiloženém CD najdete též soubor, který obsahuje adresářovou strukturu demonstrovanou na posledním příkladu. Daná rozšířená adresářová struktura najde uplatnění při používání modulů. Soubor se nachází v adresáři pro kapitolu 2 pod názvem modulovastruktura.zip.

Adresáře pro řadiče, modely a pohledy, které se nenacházejí v žádném z modulových adresářů, ale přímo v adresáři `application/`, vytvářejí `default` modul. To znamená, že tyto adresáře nemusíte explicitně přesouvat do adresáře `application/modules/default`. Front controller používá interně tento `default` modul i v případě, že jste nevytvořili žádné další moduly.

Zend Framework a poskytovatelé hostingu

Ne pro každý projekt postavený na bázi Zend Framework budete provozovat zvlášť server. Poskytovatelé hostingu často výhodně nabízejí balíky, kde je jeden server poskytovaný více zákazníkům. V tomto případě jsou z bezpečnostních i provozních důvodů vaše možnosti přístupu značně omezené. Velmi častý problém je, že nemůžete ukládat soubory do jiného adresáře než do toho, který je přístupný přes webový server, tedy váš *DocumentRoot*. Avšak i v tomto případě se doporučuje, aby pouze soubor `index.php`, a žádné jiné soubory aplikace, byl přes webový server dostupný. V následujících odstavcích bych vám rád objasnil pár možných řešení tohoto problému.

Vycházet se bude z toho, že jste se drželi představené adresářové struktury. V tomto případe byste měli do všech adresářů s výjimkou adresáře `public/` uložit soubor s názvem `.htaccess` s následujícím obsahem:

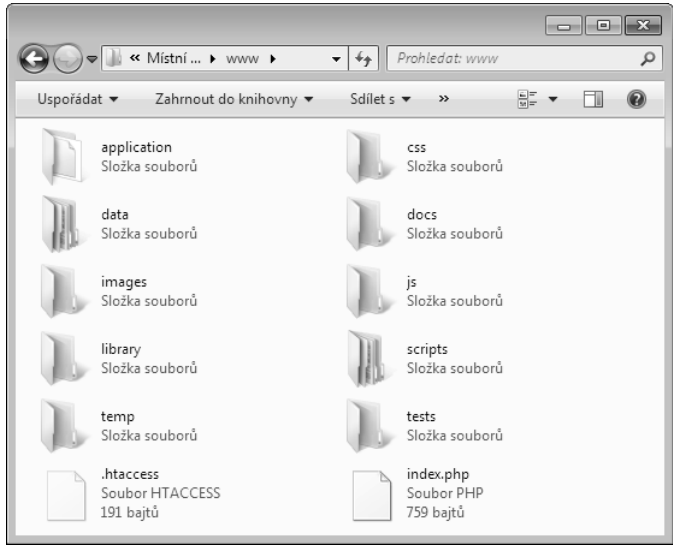
```
deny from all
```

Tím zamezíte přístup přes webový server k těm adresářům ve vašem projektu, které jsou určené jen pro vás. Samozřejmě můžete k těmto adresářům přistupovat pomocí FTP nebo SSH.

Také chcete dosáhnout toho, aby váš projekt nebyl dostupný jen přes adresu *http://www.example.com/public/*. Jestliže vám váš poskytovatel hostingu umožní v rámci vašeho adresářového prostoru pro každou doménu jiný adresář, potom je to jednoduché. Přesměrujete danou doménu na adresář `public/` a volání *http://www.example.com/* bude stejně úspěšné.

V případě, že vám poskytovatel nikdy nenabídl více adresářů, měli byste ho okamžitě vyměnit. Skutečně. V tomto případě to bude trochu složitější. Za takovýchto okolností musíte kromě zabránění přístupu

k ostatním adresářům pomocí souboru `.htaccess` provést i pár změn v adresářové struktuře. Adresář `public/` se v tomto případě „rozpustí“ a všechny soubory a adresáře z něho budou přesunuty o jednu úroveň výše. Adresářová struktura v takovém případě může vypadat tak, jak je to znázorněno na obrázku 2.8.



Obrázek 2.8: Adresářová struktura – nejvyšší úroveň, sdílené hostování

Upozornění

U této adresářové struktury musíte bezpodmínečně zabránit přístupu k adresářům `application/`, `data/`, `docs/`, `library/`, `scripts/`, `temp/` a `tests/` pomocí už zmíněného souboru `.htaccess`. Obsah souboru:

```
deny from all
```

Kromě toho byste neměli používat žádné řadiče s názvem některých adresářů, protože by mohlo dojít k neočekávaným problémům. Ještě musíte v souboru `index.php` upravit definici konstanty `APPLICATION_PATH`.

```
<?php
define('APPLICATION_PATH',
    realpath(dirname(__FILE__) . '/application')
);
[...]
```

Ještě jednou vás chci upozornit, že případná změna poskytovatele hostingu je v takovémto případě možná lepší řešení. V dnešní době už většina masových poskytovatelů, ať už na domácím nebo zahraničním trhu, nabízí vlastní adresáře pro každou z domén ve všech svých balíčcích služeb.

Tvorba virtuálních hostitelů

Musím se přiznat, že jsem velkým fanouškem virtuálních hostitelů při vývoji internetových aplikací. Představte si, že se staráte například o dva projekty:

- *http://www.mujprivatniblog.cz/*
- *http://www.mabo-eshop.cz/*

Jestliže upravujete tyto projekty na vašem vývojovém serveru, potom je budete chtít otestovat i v prohlížeči. V případě, že nepracujete s více Apache servery, budou vypadat tyto adresy víceméně následovně:

- *http://localhost/mujblog/public/*
- *http://localhost/eshop/public/*

Projekty jsou uloženy v rámci adresáře *DocumentRoot*, který používá váš lokální webový server. Protože chcete na základě této knihy vytvořit své projekty také na bázi Zend Frameworku, potřebujete k tomu ještě adresář `public/` při volání projektu z prohlížeče. Trochu komplikované, nemyslíte? Nechtěli byste namísto toho používat následující adresy?

- *http://dev.mujprivatniblog.cz/*
- *http://dev.mabo-eshop.cz/*

Jak toho můžete dosáhnout? Odpověď zní, virtuální hostitel (anglicky Virtual Hosts), přesněji řečeno Name-based Virtual Hosts. Postup bude vysvětlen na příkladu použití webového serveru Apache.

Tvorba virtuálních hostitelů pro Windows

Na příkladu webového serveru Apache 2.2 si vysvětlíme postup při tvorbě virtuálních hostitelů pod Windows. Otevřete soubor `httpd.conf` a najděte řádek:

```
NameVirtualHost *:80
```

Jestliže tento řádek začíná znakem mřížka (#), odstraňte jej. Potom vložte na konec souboru následující řádky:

```
<VirtualHost *:80>
ServerName dev.mujprivatniblog.cz
DocumentRoot c:/wamp/www/mujblog/public
</VirtualHost>

<VirtualHost *:80>
ServerName dev.mabo-eshop.cz
DocumentRoot c:/wamp/www/eshop/public
</VirtualHost>
```

V případě, že vaše projekty leží v nějakém jiném adresáři, upravte k němu cestu. V závislosti na verzi Apache můžete `:80` vynechat. Znak hvězdičky (`*`) však musí zůstat.

Nakonec musíte ještě upravit soubor `hosts`. Ve Windows 7 se nachází v adresáři `c:\windows\system32\drivers\etc\`. U jiných verzí Windows může tento adresář vypadat jinak. Otevřete daný soubor a vložte do něho nasledující řádky:

```
127.0.0.1 dev.mujprivatniblog.cz
127.0.0.1 dev.mabo-eshop.cz
```

Restartujte Apache a zkuste zadat do prohlížeče změněnou adresu, mělo by to fungovat. U jiné verze Apache se může tvorba virtuálních hostitelů lišit v detailech. V tomto případě se obraťte na referenční příručku příslušné verze nebo požádejte o pomoc na některém z fór.

Tvorba virtuálních hostitelů pro Linux

Na příkladu webového serveru Apache 2.2 vám vysvětlím postup tvorby virtuálních hostitelů pro Gentoo Linux. Přihlaste se jako uživatel `root` nebo uživatel s `root` právy. Vytvořte soubor v adresáři `/etc/apache2/vhosts.d/`, pojmenujte ho `mujblog` a vložte do něho následující obsah:

```
<VirtualHost *:80>
    ServerName dev.mujprivatniblog.cz
    DocumentRoot /var/www/localhost/mujblog/public
</VirtualHost>
```

V případě, že vaše projekty leží v jiném adresáři, upravte k němu cestu. Pomocí následujícího příkazu restartujete webový server:

```
sudo /etc/init.d/apache2 restart
```

Nakonec otevřete soubor `/etc/hosts` a vložte do něho tento řádek:

```
127.0.0.1 dev.mujprivatniblog.cz
```

V této chvíli byste měli být schopni přistupovat k lokální vývojové verzi přes nově vytvořeného virtuálního hostitele. Stejné kroky zopakujte i pro MaBo-eshop.

Použití Zend Frameworku jinak než jako framework

Zend Framework můžete používat i bez toho, abyste ho brali jako framework. Většina komponent je mezi sebou jen velmi volně propojena, to znamená, že mohou být použity nezávisle na druhých. Pokud se vám nelíbí koncept komponenty `Zend_Controller` nebo chcete místo toho použít vlastní MVC implementaci, můžete se použití komponenty `Zend_Controller` úplně vyhnout.

U tvorby adresářové struktury se nemusíte držet žádných konvencí, pravidla na přepisování URL adres mohou též vypadat úplně jinak. Zend Framework můžete použít jen jako sbírku komponent podobně jako například u PEAR.

Vhodnými kandidáty na nezávislé použití jsou například komponenty `Zend_Acl`, `Zend_Auth`, `Zend_Cache`, `Zend_Config`, `Zend_Form`, `Zend_Pdf`, `Zend_Registry`, `Zend_Search_Lucene`, `Zend_Translate`, `Zend_Db` nebo také komponenty pro webové služby.

Tato nezávislost komponent (Use at will) je jedna z největších výhod Zend Frameworku. Dělejte s ním jednoduše to, co potřebujete.

Shrnutí

Pokud si člověk uvědomí, že instalace Zend Frameworku je z principu velmi jednoduchá, dozvěděli jste se v této kapitole množství informací, jak daný framework sestavit. Nyní už víte, odkud můžete Zend Framework získat, jaké předpoklady musí být splněny pro jeho použití a jak sestrojíte doporučenou adresářovou strukturu. Po instalaci jste se dozvěděli, jak použijete více modulů, jaké máte možnosti sestavení Zend Frameworku u poskytovatelů hostingu a jak můžete vytvořit virtuální hostitele na vašem vývojovém serveru.

KAPITOLA 3

Rychlý start se Zend Frameworkem

Po úvodních informacích k instalaci Zend Frameworku nabízí táto kapitola možnost rovnou se s ním seznámit v praxi. Příklad, který tu bude probrán, bude postaven na architektuře MVC. V případě, že vám daný pojem zatím není dost známý, nahlédněte do přílohy B, do části MVC architektura.

Jestliže neporozumíte všemu, co bude v této kapitole vysvětleno a ukázáno, nic si z toho nedělejte. Je však dobré, abyste na začátek aplikaci vytvořenou pomocí Zend Frameworku aspoň viděli. Při nejasnostech se stále můžete obrátit na odkazy, které budou uváděny v textu. Jako doplněk k této kapitole se vyplatí navštívit stránku s oficiální ukázkovou aplikací (*http://framework.zend.com/docs/quickstart*).

Tip

Na přiloženém CD najdete v adresáři `kapitola03/rychlystart/` kompletní aplikaci k této kapitole. Kromě toho najdete v adresáři `kapitola03/` všechny výpisy, které tu budou probírány. To vám ušetří množství času při psaní v případě, že postupujete krok za krokem s výkladem.

Tvorba projektu

Vytvořte nový adresář projektu (kapitola 2, Vytvoření adresáře pro projekt) a pojmenujte ho `rychlystart`. Z CD nakopírujte doporučenou adresářovou strukturu `adresarovastruktura.zip` z adresáře `kapitola02` do nově vytvořeného adresáře `rychlystart/`.

Rozbalte zkopírovaný soubor `adresarovastruktura.zip` a smažte ho. Zkontrolujte, zda váš rozbalovací program nevytvořil v adresáři `rychlystart/` adresář `adresarovastruktura/`. V případě že ne, pokračujte dalším odstavcem. Jestliže daný adresář přece jen existuje, přesuňte obsah z `rychlystart/adresarovastruktura/` o jednu úroveň výše do adresáře `rychlystart/` a smaž-

te adresář `rychlystart/adresarovastruktura/`. Přesvědčte se, že existuje soubor `rychlystart/public/index.php`.

Jako poslední krok zkopírujte celý obsah adresáře `ZendFramework-1.10.3/library/` do `rychlystart/library/`. Tím je příprava projektu ukončena.

Tip

Od verze 1.8 máte možnost sestavit projekt pomocí komponenty `Zend_Tool`, která celý proces ulehčuje. Více se o komponentě `Zend_Tool` dozvíte v kapitole 4, Základní komponenty.

Ještě předtím, než budete pokračovat ve vytváření dalších souborů, budou v krátkosti představeny už existující soubory. Tyto se mohou odlišovat v detailech podle toho, zda je převezmete z předlohy na CD nebo použijete k jejich vytvoření komponentu `Zend_Tool`.

Konfigurace serveru Apache

V této kapitole bude vysvětlena konfigurace webového serveru Apache pro váš projekt. V případě, že používáte jiný webový server, například Lighttpd nebo Microsoft IIS, najdete návod na jejich konfiguraci v referenční příručce na adrese *http://framework.zend.com/manual/en/zend.controller.router.html*.

V adresáři `rychlystart/public/` se vedle souboru `index.php` nachází i soubor `.htaccess`. Výpis 3.1 znázorňuje možný obsah tohoto souboru.

Výpis 3.1: Definování přepisovacích pravidel v souboru .htaccess

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^.*$ - [NC,L]
RewriteRule ^.*$ index.php [NC,L]
```

Smyslem a účelem těchto řádků je, že všechny požadavky na webový server budou přesměrovány na soubor `index.php` v adresáři `public/`. Pomocí tzv. RewriteEngine (Apache modul, který přepisuje URL adresy) můžete realizovat URL optimalizované pro vyhledávače. V překladu to znamená, že namísto adresy *http://www.example.com/?controller=book&action=buy&id=1788* můžete použít adresu *http://www.example.com/book/buy/id/1788*.

Na prvním řádku výpisu 3.1 bude aktivován RewriteEngine. Za ním následují podmínky. Přeloženo do lidské řeči znamenají další řádky následující: v případě, že daný požadavek `{REQUEST_FILENAME}` je platný symbolický odkaz (`-s`) nebo neprázdný soubor (`-l`) nebo adresář (`-d`), vykoná se tento beze změny (1. RewriteRule). V opačném případě bude daný požadavek přesměrován na soubor `index.php` (2. RewriteRule). Velká a malá písmena přitom nebudou rozlišována (`[NC]`) a zpracování se ukončí, jakmile bude splněna některá z podmínek (`[L]`).

Myslete přitom na to, že odkazy na neexistující soubory budou také přesměrovány na soubor `index.php`. To znamená, že pokud vložíte na stránku soubor, který neexistuje fyzicky na serveru, bude tento požadavek přesměrován na soubor `index.php`. Toto se může negativně projevit na rychlosti vaší Zend Framework aplikace hlavně u malých neexistujících souborů s grafikou, které se používají na vytváření layoutu stránky. Měli byste proto vždy dbát na to, aby soubory, na které se odkazujete, opravdu existovaly.

Jestliže hledáte více informací ohledně modulu `mod_rewrite`, doporučím vám stránku *http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html*, kde najdete další užitečné informace ohledně Apache modulu.

Tip

Jestliže přepisování URL adres nefunguje, může být problém v konfiguraci Apache serveru. Zkontrolujte proto, zda máte pro svůj Apache server aktivovaný modul `mod_rewrite`. Aktivace tohoto modulu záleží na vámi používané verzi Apache a může se od verze k verzi lišit. Pro přesný postup, jak to udělat, se obraťte na referenční příručku konkrétní verze.

Soubor index.php

Soubor `index.php` se nachází v adresáři `rychlystart/public/` a měl by být jediným spustitelným PHP souborem ve vašem projektu. Tento soubor je zodpovědný za konfiguraci PHP prostředí a za start front controlleru. Front controller dále zabezpečí, že všechny požadavky na webový server budou zpracovány pomocí tohoto souboru. Ve výpisu 3.2 je znázorněn vzorový soubor `index.php`.

Výpis 3.2: Příklad vzorového souboru index.php

```
<?php
define('APPLICATION_PATH',
    realpath(dirname(__FILE__) . '/../application')
);
if($_SERVER['HTTP_HOST'] == 'localhost') {
    define('APPLICATION_ENVIRONMENT', 'development');
} else {
    define('APPLICATION_ENVIRONMENT', 'production');
}

set_include_path(implode(PATH_SEPARATOR, array(
    realpath(APPLICATION_PATH . '/../library'),
    get_include_path(),
)));

require_once 'Zend/Application.php';

$application = new Zend_Application(
    APPLICATION_ENVIRONMENT,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()
            ->run();
```

První řádky tohoto souboru mají za úlohu nakonfigurovat PHP prostředí: nejprve nastavit konstantu `APPLICATION_PATH` a za ní v závislosti na názvu serveru konstantu `APPLICATION_ENVIRONMENT`. Na dalších řádcích je přidán do `include_path` adresář `library/`, který obsahuje Zend Framework a váš adresář rozšiřující Zend Framework. Za ním bude načtena třída komponenty `Zend_Application` a následně daná komponenta inicializována. Komponenta `Zend_Application` se postará o start front controlleru. Víc o možnostech použití komponenty `Zend_Application` se dočtete v kapitole 4, Základní komponenty.

Soubor Bootstrap.php

Zatímco se soubor `index.php` stará o konfiguraci PHP prostředí, soubor `Bootstrap.php` je zodpovědný za konfiguraci prostředí Zend Frameworku. Do nástupu verze 1.8 byl tento soubor potřeba pro konfiguraci Zend Frameworku, kde bylo možné použít buď procedurální kód, nebo v něm definovat třídu, která provede konfiguraci objektově orientovanou. S nástupem verze 1.8 a příchodem komponenty `Zend_Application` se způsob konfigurace Zend Frameworku díky dané komponentě standardizoval a hlavně zjednodušil. Ve výpisech 3.3 až 3.5 najdete pro porovnání způsob konfigurace Zend Frameworku sta-

rým způsobem a způsobem s použitím komponenty `Zend_Application`. Ve výpisu 3.3 je demonstrována základní konfigurace front controlleru bez použití komponenty `Zend_Application`. Stejné nastavení bude též vykonáno ve výpisech 3.4 a 3.5.

Výpis 3.3: Vzorový Bootstrap.php soubor bez použití komponenty Zend_Application

```
<?php
$frontController = Zend_Controller_Front::getInstance();

$frontController->setControllerDirectory(
    APPLICATION_PATH . '/controllers'
);

$frontController->addModuleDirectory(
    APPLICATION_PATH . '/modules'
);
$frontController->setParam('root', APPLICATION_PATH);

unset($frontController);
```

Ve výpisu 3.4 je znázorněn soubor Bootstrap.php při použití komponenty `Zend_Application`.

Výpis 3.4: Soubor Bootstrap.php při použití komponenty Zend_Application

```
<?php
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{

}
```

Jak můžete vidět na výpisu 3.4, celá konfigurace se přesunula do konfiguračního souboru `application.ini` v adresáři `rychlystart/application/configs`. Jeho obsah můžete vidět ve výpisu 3.5.

Výpis 3.5: Soubor application.ini

```
[production]
phpSettings.display_startup_errors = 0
phpSettings.display_errors = 0
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory =
    APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 0

[staging : production]

[testing : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1

[development : production]
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
resources.frontController.params.displayExceptions = 1
```

Záznamy v souboru `application.ini` oznamují komponentě `Zend_Application`, kde má hledat Bootstrap soubor a jak se volá daná třída. Další dva řádky jsou určeny pro nastavení front controlleru – určení adresáře s řadiči a moduly. Jak sami vidíte, je způsob konfigurace prostředí Zend Frameworku pomocí komponenty `Zend_Application` mnohem jednodušší a přehlednější. Co všechno můžete nastavit v konfiguračním souboru, jakým způsobem a jaké další možnosti poskytuje třída v souboru `Bootstrap.php`, se dočtete v kapitole 4, Základní komponenty.

Řadič

Nastal čas věnovat se řadičům, a tedy komponentě `Zend_Controller`. Informace o funkci řadiče v MVC architektuře najdete na konci knihy v příloze, v části MVC architektura – Funkce řadiče v MVC. Informace o komponentě `Zend_Controller` jsou v kapitole 5, Řadič.

Vymezení pojmů

Na úvod musíte vědět, že všechny akce v Zend Framework aplikaci můžete shrnout do action controlleru. V rámci jednoho action controlleru můžete vytvořit libovolný počet akcí. Teoreticky můžete všechny akce na své internetové stránce uložit do jednoho action controlleru, což pro zlepšení přehlednosti nedoporučuji.

Následuje malý příklad k objasnění pojmů action controller a akce (anglicky Action Method). Představte si nákupní košík pro MaBo eshop. Musíte přitom:

- vložit novou knihu do košíku,
- vložené knihy upravit nebo odstranit,
- zobrazit nákupní košík,
- odeslat nákupní košík jako objednávku.

Potřebujete jeden action controller pro nákupní košík, který obsahuje akce – metody pro vložení, úpravu, odstranění, zobrazení a odeslání. Pro správu produktů (knih) bych doporučil mít vlastní action controller. Také se vyplatí mít vlastní action controller pro správu objednávek. Alternativně můžete též pracovat s moduly (kapitola 2, Použití více modulů) a dané action controllery rozdělit do dvou modulů – veřejnosti přístupný a `admin` modul.

IndexController

V souboru `rychlystart/application/controllers/IndexController.php` najdete třídu s názvem `IndexController`. Daná třída obsahuje akci `indexAction()`.

Tato akce bude zavolána při volání úvodní stránky vaší aplikace. V každém projektu byste měli vytvořit minimálně tento action controller. Metoda `indexAction()` je zatím prázdná, což se teď změní. Na úvodní stránce tohoto vzorového projektu má být zobrazen seznam knih. Změňte metodu `indexAction()` tak, jak je to uvedeno ve výpisu 3.6. Nejdřív se definují údaje, které potom budou předány pohledu. Kromě toho vidíte ve výpisu 3.6 i metodu `showAction()`, která slouží k zobrazení konkrétní knihy. Také zde budou údaje pevně stanoveny.

Výpis 3.6: Action controller pro úvodní stránku

```
<?php
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $bookList = array(
            array(
                'book_id'          => 1,
                'book_name'        => 'Zend Framework',
                'book_description' =>
                    'Programujeme webové aplikace v PHP'
            ),
            array(
                'book_id'          => 2,
                'book_name'        => 'PHP 6, MySQL, Apache',
                'book_description' =>
```

```
                    'Vytváříme webové aplikace',
            ),
        );
        $this->view->bookList = $bookList;
    }

    public function showAction()
    {
        $bookData = array(
            'book_id'          => 1,
            'book_name'        => 'Zend Framework',
            'book_description' =>
                'Programujeme webové aplikace v PHP',
        );
        $this->view->bookData = $bookData;
    }
}
```

ErrorController

Kromě `IndexController`u byste měli vytvořit i `ErrorController`. Ten má za úkol zobrazit všechny chyby v aplikaci tak, že uživatel nedostane žádné přímé chybové hlášení. Zend Framework přesměruje při všech výjimkách zpracování požadavku na metodu `errorAction()` v `ErrorController`u. Tady můžete pro uživatele vytvořit nějaké všeobecné hlášení a skutečný záznam o chybě zapsat do log souboru (viz kapitolu 4, komponentu `Zend_Log`).

V souboru `rychlystart/application/controllers/ErrorController.php` najdete třídu s názvem `ErrorController` a prázdnou metodu `errorAction()`. Upravte tuto metodu podle výpisu 3.7. Jako první bude objekt popisující chybu získaný z request objektu. Na základě typu chyby budou webovému serveru odeslány stavové kódy. Nakonec bude pohledu předán text popisující chybu, typ prostředí (`production`, `testing`) a také odchycená výjimka.

Výpis 3.7: Action controller pro zpracování chyb

```
<?php
class ErrorController extends Zend_Controller_Action
{
    public function errorAction()
    {
        $errors = $this->_getParam('error_handler');

        switch ($errors->type) {
            case Zend_Controller_Plugin_ErrorHandler::         EXCEPTION_NO_ROUTE:
            case Zend_Controller_Plugin_ErrorHandler::         EXCEPTION_NO_CONTROLLER:
            case Zend_Controller_Plugin_ErrorHandler::         EXCEPTION_NO_ACTION:
                $this->getResponse()->setHttpResponseCode(404);
                $this->view->message = 'Stránka nebyla nalezena';
            break;
            default:
                $this->getResponse()->setHttpResponseCode(500);
                $this->view->message = 'Vyskytla se chyba';
            break;
        }

        $this->view->exception   = $errors->exception;
        $this->view->environment = APPLICATION_ENVIRONMENT;
    }
}
```

S těmito dvěma Controllery můžeme tedy začít. Počet action controllerů ani metod v nich není, jak už jsem uvedl, nijak omezen.

Struktura stránky a pohledy

Jako další jsou na řadě pohledy. Informace ohledně funkce pohledu v MVC architektuře najdete na konci knihy v příloze, v části MVC architektura – Funkce pohledu v MVC. V této kapitole budou použity komponenty `Zend_View` a `Zend_Layout` k vytvoření centrálního layoutu. Další informace ohledně obou komponent najdete v kapitole 6, Pohled.

Vytvoření centrálního layoutu

Centrální layout stránky vytvoříte pomocí komponenty `Zend_Layout`. Tím dosáhnete toho, že HTML kód, který je stejný ve všech stránkách, můžete uložit a spravovat na jednom centrálním místě a vaše skripty pohledů se mohou koncentrovat jen na HTML výstup jednotlivých akcí.

Pro vytvoření centrálního layoutu stránky vytvořte nejprve soubor `rychlystart/application/layouts/scripts/layout.phtml`, jak je to zobrazeno ve výpisu 3.8. Rozhodující je přitom řádek `$this->layout()->content`, který vypíše pomocí view helperu komponenty `Zend_Layout` samotný obsah volané akce.

Výpis 3.8: Soubor s centrálním layoutem pro vaši stránku

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>MaBo eshop</title>
</head>
<body>
<div id="page" style="width: 800px;">
<div id="title" style="border-bottom: 1px solid;">
<h1>MaBo eshop</h1>
</div>
<div id="main">
<div id="menu" style="float:left; width: 120px; margin-right: 10px;">
<ul style="list-style: none; margin: 0; padding: 0;">
<?php $url = $this->url(
    array('controller' => 'index', 'action' => 'index'), null, true); ?>
<li><a href="<?php print $url; ?>">Seznam knih</a></li>
</ul>
</div>
<div id="content" style="border-left: solid 150px #fff; background: #eee;">
<?php print $this->layout()->content; ?>
</div>
<br class="clear" />
</div>
<div id="footer" style="text-align: center; border-top: 1px solid;">
&copy; 2010 MaBo eshop
</div>
</div>
</body>
</html>
```

Aby Zend Framework věděl, že chcete použít komponentu `Zend_Layout`, musíte přidat do vašeho konfiguračního souboru `rychlystart/application/configs/application.ini` (nejlépe do sekce `production`) následující řádek:

```
resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts"
```

Další informace ohledně komponenty `Zend_Layout` najdete v kapitole 6, Pohled.

Skript pohledu pro úvodní stránku

Pro vytvoření skriptu pohledu pro úvodní stránku se seznamem knih otevřete soubor `index.phtml` v adresáři `rychlystart/application/views/scripts/index/`. Smažte jeho obsah a vložte ten z výpisu 3.9. Tím budou na úvodní stránce zobrazeny knihy, které byly vytvořeny ve výpisu 3.6. Tento skript sestává z HTML kódu i z PHP fragmentů. Detaily najdete v kapitole 6, Pohled, v části `Zend_View` skripty.

Výpis 3.9: Skript pohledu, který vypíše seznam knih

```
<h2>Seznam knih</h2>
<p>Vyberte si jednu z knih!</p>
<table border="1">
<tr><th>Kniha</th><th>Popis</th></tr>
<?php foreach ($this->bookList as $bookData) : ?>
<?php $url = $this->url(array(
    'controller' => 'index',
    'action'     => 'show',
    'id'         => $bookData['book_id']
)); ?>
<tr>
<td><a href="<?php print $url ?>">
    <?php print $bookData['book_name'] ?></a></td>
<td><a href="<?php print $url ?>">
    <?php print $bookData['book_description'] ?></a></td>
</tr>
<?php endforeach; ?>
</table>
```

K proměnným s údaji o knihách, které byly předány pohledu, přistupujete pomocí `this->bookList`. U `$this` se jedná o instanci objektu `Zend_View`, což umožňuje přistupovat ke všem předaným proměnným jako k atributům tohoto objektu.

Za zmínku stojí i `$this->url(...)` – tím se volá určitý view helper, v tomto případě view helper `url`. View helpery zapouzdřují opakující se části HTML kódu v skriptech pohledů. view helper `url` slouží k vytvoření URL adresy na základě předaných parametrů pro řadič, model, akci atd. Více informací o view helperech najdete v kapitole 6, Pohled v části View helper.

Výsledek výpisu 3.9 si můžete prohlédnout na obrázku 3.1.

Skript pohledu pro zobrazení knihy

V adresáři `rychlystart/application/views/scripts/index/` vytvořte soubor `show.phtml` a vložte do něho kód z výpisu 3.10. K údajům se přistupuje pomocí `$this->bookData`.

Výpis 3.10: Skript pohledu pro zobrazení konkrétní knihy

```
<h2><?php print $this->bookData['book_name'] ?></h2>
<p><?php print $this->bookData['book_description'] ?></p>
<hr />
<?php $url = $this->url(
    array('controller' => 'index', 'action' => 'index'),
    null, true
); ?>
<p><a href="<?php print $url ?>">zpět k seznamu knih</a></p>
```

Obrázek. 3.1: Skript pohledu, který vypíše seznam knih

Protože jsou údaje předané pohledu v řadiči pevně definovány, budou při každém volání tohoto skriptu zobrazeny stejné údaje.

Výsledek výpisu 3.10 si můžete prohlédnout na obrázku 3.2.

Obrázek. 3.2: Skript pohledu, který zobrazí konkrétní knihu

Skript pohledu zobrazující chybové hlášení

Jako poslední je potřeba skript pohledu pro vypisování chybových hlášení. Otevřete soubor `error.phtml` v adresáři `rychlystart/application/views/scripts/error/` a upravte jeho obsah podle výpisu 3.11.

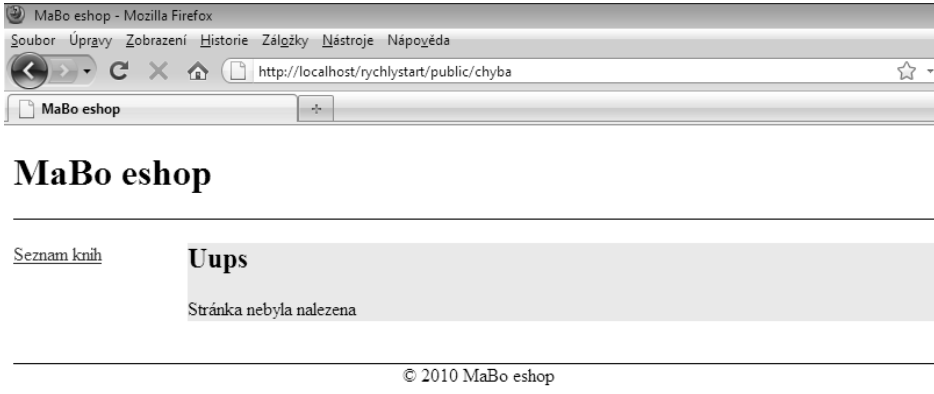
Výpis 3.11: Skript pohledu pro chybové hlášení

```
<h2>Uups</h2>
<p><?php print $this->message; ?></p>
<?php if ('development' == $this->environment): ?>
<h3>Výjimka:</h3>
```

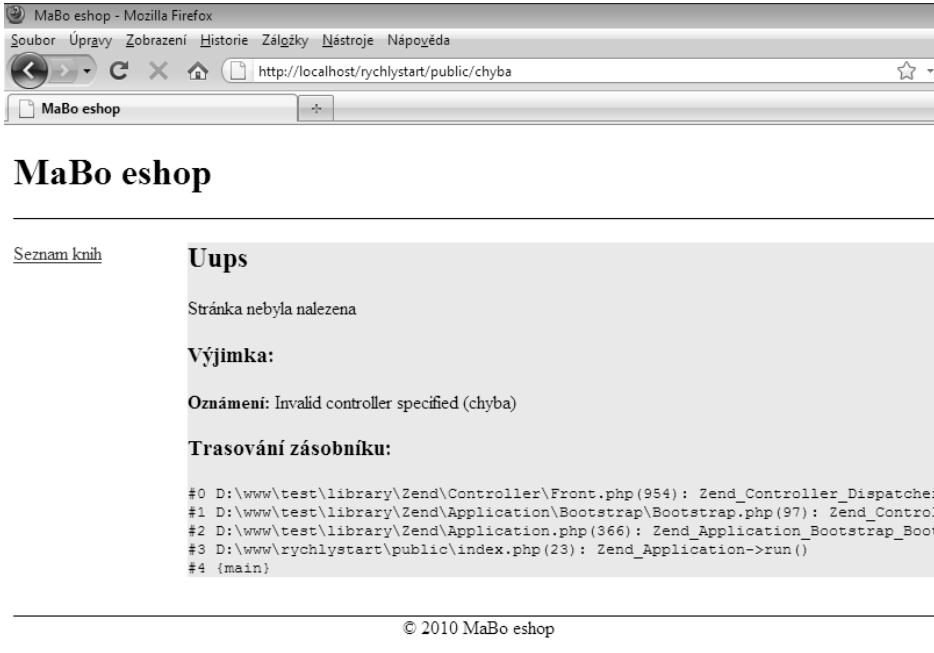
```
<p><b>Oznámení:</b>
    <?php print $this->exception->getMessage(); ?></p>
<h3>Trasování zásobníku:</h3>
<pre><?php print $this->exception->getTraceAsString(); ?></pre>
<?php endif; ?>
```

V závislosti na nastavení PHP prostředí budou zobrazeny i podrobnosti k vyvolané výjimce. V případě produkčního serveru uvidí uživatel pouze chybové hlášení.

Na následujících obrázcích 3.3 a 3.4 vidíte výpisy chybových hlášení v závislosti na nastavení prostředí. Na obrázku 3.3 je chybové hlášení v případě produkčního serveru a na obrázku 3.4 v případě neprodukčního serveru (vývoj, testovaní a podobně).



Obrázek. 3.3: Skript pohledu s chybovým hlášením pro produkční server



Obrázek. 3.4: Skript pohledu s chybovým hlášením pro neprodukční server

Tip

Po vytvoření a uložení všech potřebných skriptů nastal čas, abyste zavolali svoji aplikaci z internetového prohlížeče. URL adresa daného projektu se může lišit podle toho, kde jste ho uložili. V případě, že se adresář `rychlystart/` nachází přímo v *DocumentRoot* v adresáři webového serveru, může vypadat volání následovně:

http://localhost/rychlystart/public/

Po načtení stránky byste měli vidět seznam knih. Po klepnutí na jednu z dvou knih byste měli vidět konkrétní knihu. Při zadání chybné URL adresy by se mělo objevit chybové hlášení:

http://localhost/rychlystart/public/chyba

Konfigurace

Každá moderní internetová aplikace potřebuje konfigurační soubor. Ten slouží například ke konfiguraci databázového připojení, k uložení centrálních e-mailových adres nebo cest k log souborům a podobně. Zend Framework poskytuje pro tyto potřeby komponentu `Zend_Config`, přičemž můžete své konfigurační soubory ukládat jako INI soubory, soubory ve formátu XML nebo jako PHP pole.

Vytvoření konfiguračního souboru

Pro příklad vysvětlovaný v této knize bude použit konfigurační soubor v INI formátu. Otevřete soubor `application.ini` v adresáři `rychlystart/application/configs` a doplňte do něj na příslušná místa kód z výpisu 3.12.

Výpis 3.12: Konfigurační soubor v INI formátu

```
[production]
resources.db.adapter = Pdo_Sqlite
resources.db.params.dbname = "/local/database/server/books.sqlite"
resources.db.params.sqlite2 = true
resources.db.isDefaultTableAdapter = true

[development : production]
resources.db.params.dbname = "../data/books.sqlite"
```

Konfigurační údaje jsou definovány odděleně pro každé prostředí. V tomto případě se rozlišuje mezi prostředími `production` a `development`. Údaje můžete přiřazovat zadáním názvu prostředí v hranatých závorkách. Také je tu možnost dědění pomocí použití znaku dvojtečky (`:`). Podle výpisu 3.12 jsou tedy údaje pro konfiguraci databáze stejné pro obě dvě prostředí až na výjimku `resources.db.params.dbname`, která určuje cestu k souboru s databází.

Načítání a zpřístupnění konfiguračního souboru

Do nástupu komponenty `Zend_Application` byl tento krok velmi důležitý. S jejím nástupem je však jeho manuální načítání ve většině malých aplikací nepotřebné, protože důležité údaje z něj jsou automaticky načítány komponentou `Zend_Application` a jejími subkomponentami.

Jako malý bonus tu uvedu, jak je možné realizovat jeho načítání a zpřístupnění. Používá se na to komponenta `Zend_Config`. Otevřete soubor `Bootstrap.php` v adresáři `rychlystart/application/` a přidejte do něj následující metodu:

```
protected function _initConfig()
    {
        $config = $this->getOptions();
        Zend_Registry::set('config', $config);
```

```
        return $config;
    }
```

Metoda `_initConfig()` je volána při zavádění aplikace. Nejprve budou načítány konfigurační údaje ze souboru a bude vytvořena instance objektu `Zend_Config`. Aby bylo možné použít tuto instanci v celé aplikaci, bude použita komponenta `Zend_Registry`. Tím je umožněno, že můžete k daným údajům přistupovat například v action controlleru nebo ve skriptu pohledu.

```
$config = Zend_Registry::get('config');
```

Proměnná `$config` obsahuje `Zend_Config` objekt. Další informace o komponentách `Zend_Config` a `Zend_Registry` najdete v kapitole 4, Základní komponenty.

Modely a databáze

Práce pouze se statickými daty je už v dnešní době velmi kontraproduktivní, proto jako další krok vytvoříme pro tento vzorový příklad modely. Informace o funkci modelu v MVC architektuře najdete na konci knihy v příloze, v části MVC architektura – Funkce modelu v MVC. K vytvoření modelů budou použity komponenty `Zend_Db_Adapter` a `Zend_Db_Table`. Informace k těmto komponentám najdete v kapitole 7, Databáze.

Příprava

Nejprve potřebujeme pár údajů. Zkopírujte proto z přiloženého CD soubor `data/database/books.sqlite` do `rychlystart/data/books.sqlite`. Tento soubor obsahuje databázi, která je postačující pro účely testování. Jako alternativu můžete použít výpis z SQL souboru `rychlystart/data/books.sql`.

O přípravu a inicializaci databázového adaptéru `Zend_Db_Adapter` se postará komponenta `Zend_Application` na základě údajů, které byly zadány v konfiguračním souboru.

Vytvoření modelů

Pro zjednodušení věcí bude jako základ pro modely použita komponenta `Zend_Db_Table`. To, že modely mohou být i něco víc než jen databázové tabulky, ke kterým můžete přistupovat, najdete blíže vysvětleno v kapitole 17, Použití modelů. Tuto kapitolu byste si měli projít, až budete mít víc zkušeností se Zend Frameworkem.

Vytvořte soubor `rychlystart/application/models/Books.php` a vložte do něj obsah z výpisu 3.13. Pomocí třídy `Books` můžete od této chvíle přistupovat k údajům v SQLite databázi v tabulce `books`.

Výpis 3.13: Model přístupu k tabulce books

```
class Application_Model_Books extends Zend_Db_Table_Abstract
{
    protected $_name    = 'books';
    protected $_primary = 'book_id';

    public function fetchRowWithTags($id)
    {
        $id = Zend_Filter::filterStatic($id, 'Int');
        $data = $this->find($id)->current();
        if (empty($data)) {
            throw new Zend_Db_Exception(
                'ID knihy "' . $id . '" je neplatné'
            );
        }

        $data = $data->toArray();
```

```
        $select = $this->getAdapter()->select();
        $select->from('tags');
        $select->join(
            'book_tags', 'bo2ta_tag_id = tag_id', array()
        );
        $select->where('bo2ta_book_id = ?', $id);

        $data['tags'] = $this->getAdapter()->fetchAll($select);
        return $data;
    }
}
```

Jako rozšíření byla vytvořena metoda `fetchRowWithTags()`:

- Předaný parametr bude filtrován pomocí komponenty `Zend_Filter`. Více se o této komponentě dozvíte v kapitole 4, Základní komponenty.
- Načítají se údaje pro aktuální knihu.
- Jestliže žádné údaje nemohly být načteny, bude vyvolána výjimka.
- Načítané údaje budou překonvertovány do pole.
- Za účelem načítání značek (anglicky Tags) bude vytvořen databázový dotaz pomocí komponenty `Zend_Db_Select`. Více se o této komponentě dozvíte v kapitole 7, Databáze.
- Údaje o značkách budou načítány a zkombinovány s údaji o knize.

Jak vidíte ve výpisu 3.13, můžete vašeho potomka třídy `Zend_Db_Table` rozšířit o vlastní metody. Jako alternativu můžete také použít podporu relací mezi tabulkami – kapitola 7, Databáze, část Relace mezi tabulkami.

Použití modelů

Nastal čas nově vytvořený model použít. K tomu je potřeba upravit akci v action controlleru `IndexController`, jak je to zobrazeno ve výpisu 3.14. V obou dvou akcích bude inicializována instance třídy `Application_Model_Books`. Potom budou z databáze načítány požadované údaje a budou předány pohledy.

Výpis 3.14: Action controller s použitím modelu

```
class IndexController extends Zend_Controller_Action
{
    public function indexAction()
    {
        $books = new Application_Model_Books();
        $bookList = $books->fetchAll()->toArray();
        $this->view->bookList = $bookList;
    }

    public function showAction()
    {
        $id = $this->getRequest()->getParam('id');
        $books = new Application_Model_Books();
        $bookData = $books->fetchRowWithTags($id);
        $this->view->bookData = $bookData;
    }
}
```

Tím budou zobrazeny skutečné údaje. Když zavoláte svoji aplikaci z internetového prohlížeče *http://localhost/rychlystart/public/*, měli byste dostat seznam pěti knih, jak je znázorněno na obrázku 3.5. Pokud klepnete na jednu z nich, dostanete údaje o této knize. V případě, že zadáte chybnou adresu, měli byste dostat chybové hlášení. Vyzkoušejte *http://localhost/rychlystart/public/index/show/id/xyz*.

Obrázek 3.5: Výpis seznamu knih z databáze

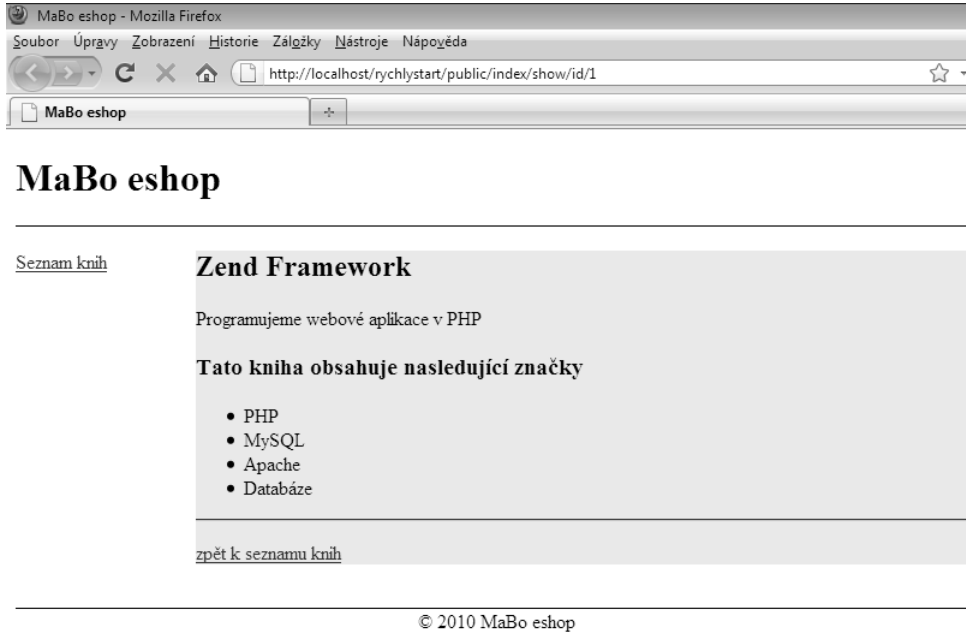
Úprava pohledu pro zobrazení knihy

Jako poslední krok musíte ještě upravit skript pohledu pro detailní zobrazení knihy, abyste mohli zobrazit načítané údaje. Upravte soubor `show.phtml` v adresáři `rychlystart/application/views/scripts/index/` podle výpisu 3.15.

Výpis 3.15: Upravený skript pohledu pro zobrazení detailu knihy

```
<h2><?php print $this->bookData['book_name'] ?></h2>
<p><?php print $this->bookData['book_description'] ?></p>
<h3>Tato kniha obsahuje následující značky</h3>
<ul>
<?php foreach ($this->bookData['tags'] as $tag) : ?>
<li><?php print $tag['tag_name'] ?></li>
<?php endforeach; ?>
</ul>
<hr />
<?php $url = $this->url(
    array('controller' => 'index', 'action' => 'index'),
    null, true
); ?>
<p><a href="<?php print $url; ?>">zpět k seznamu knih</a></p>
```

Zavolejte znovu svoji aplikaci (*http://localhost/rychlystart/public/*) a klepněte na některou z knih. Nyní byste už měli vidět i jednotlivé kategorie, do kterých daná kniha patří, tak, jak je to zobrazeno na obrázku 3.6.



Obrázek 3.6: Zobrazení knihy s kategoriemi

Shrnutí

Gratuluji! Právě jste vytvořili svoji první malou Zend Framework aplikaci. Přitom jste se po vytvoření projektu naučili, jaké úlohy mají soubory `index.php` a `Bootstrap.php`. Poznali jste nejdůležitější prvky MVC architektury, řadič, model a pohled, a také víte, jak se používá konfigurační soubor a komponenta `Zend_Registry`.

Na přiloženém CD v adresáři `kapitola03/rychlystart/` najdete též kompletní aplikaci pro porovnání s tou vaší. Pro lepší zapamatování naučeného můžete nyní přejít k druhé části této knihy a poznat jednotlivé komponenty Zend Frameworku. Nebo přejděte rovnou k třetí části a podle potřeby se zpětně vracejte.

ČÁST II

Komponenty

KAPITOLA 4

Základní komponenty

Možná se divíte, proč tato kniha nezačíná základními komponentami pro MVC aplikaci. Má to svoje důvody. V Zend Frameworku existují komponenty, které mohou být použity samostatně nebo i ve spolupráci s MVC komponentami. Tyto základní komponenty řeší mnoho každodenních problémů v životě programátora a poskytují solidní základy při zaškolení se do Zend Frameworku.

Zend_Application

Komponenta `Zend_Application` standardizuje proces zavádění aplikace pro znovu použitelné zdroje, jako jsou například databázové adaptéry nebo normálně a modulově závislé Bootstrap třídy, a kontroluje jejich závislosti. Také se stará o nastavení PHP prostředí a automatické nahrávání tříd.

Do verze 1.7.4 neexistovala na sestavení Bootstrap souboru žádná standardizovaná doporučení. Od verze 1.8 nabízí `Zend_Application` flexibilní způsob pro proces zavádění. V souboru `application/Bootstrap.php` můžete vytvořit třídu, která bude obsahovat inicializační metody. Také Můžete také použít jen konfigurační soubor a úplně se vzdát Bootstrap třídy. V tomto případě inicializuje vaši aplikaci přímo soubor `index.php`. Můžete též použít kombinaci obou způsobů, což je v praxi nejčastější případ.

Konfigurace Zend_Application pomocí konfiguračního souboru

Ve výpisu 4.1 vidíte příklad inicializace `Zend_Application` v souboru `public/index.php` a konfiguraci pomocí konfiguračního souboru.

Výpis 4.1: Konfigurace Zend_Application pomocí konfiguračního souboru

```
define('APPLICATION_PATH',
    realpath(dirname(__FILE__) . '/../application')
);
define('APPLICATION_ENVIRONMENT', 'development');

set_include_path(implode(PATH_SEPARATOR, array(
    realpath(APPLICATION_PATH . '/../library'),
```

```
    get_include_path(),
)));

require_once 'Zend/Application.php';
$application = new Zend_Application(
    APPLICATION_ENVIRONMENT,
    APPLICATION_PATH . '/configs/application.ini'
);
$application->bootstrap()->run();
```

Jako první jsou definovány konstanty pro PHP prostředí a adresář aplikace. Následuje nastavení `include_path`. Nakonec je načten soubor s komponentou `Zend_Application`, je vytvořena instance třídy a jsou zavolány metody `bootstrap()` a `run()`, které se postarají o zavedení aplikace a odstartování front controlleru. Konstruktoru třídy jsou předány dva parametry: definované PHP prostředí a cesta ke konfiguračnímu souboru. Ukázku konfiguračního souboru najdete ve výpisu 4.2.

Výpis 4.2: Konfigurační soubor pro Zend_Application

```
resources.db.adapter = Pdo_Mysql
resources.db.params.host = 'localhost'
resources.db.params.dbname = 'maboeshop'
resources.db.params.username = 'mabo'
resources.db.params.password = 'eshop'
resources.db.isDefaultTableAdapter = true
```

Konfigurační soubor definuje databázový adaptér, parametry potřebné pro připojení k databázi a nastaví jej jako předvolený, viz kapitolu 7, Databáze.

Přímá konfigurace Zend_Application

Namísto cesty ke konfiguračnímu souboru můžete jako druhý parametr předat komponentě `Zend_Application` pole s hodnotami a nakonfigurovat ji přímo v souboru `index.php`. Výpis 4.3 to demonstruje.

Výpis 4.3: Přímá konfigurace Zend_Application

```
$config = array(
    'resources' => array(
        'db' => array(
            'adapter'               => 'Pdo_Mysql',
            'isDefaultTableAdapter' => true,
            'params' => array(
                'host'     => 'localhost',
                'dbname'   => 'maboeshop',
                'username' => 'mabo',
                'password' => 'eshop',
            ),
        ),
    ),
);

$application = new Zend_Application(
    APPLICATION_ENVIRONMENT,
    $config,
);
$application->bootstrap()->run();
```

Konfigurace Zend_Application pomocí Bootstrap třídy

Alternativně můžete pro zavedení své aplikace použít i vlastní Bootstrap třídu. Tato třída musí být potomkem třídy `Zend_Application_Bootstrap_Bootstrap`. Uložit ji můžete na libovolném místě, standardně se na to používá soubor `application/Bootstrap.php`. Výpis 4.4 demonstruje jednoduchý příklad. Myslete na to, že vaše inicializační metody musí začínat `_init` a musí být deklarovány jako `protected`.

Výpis 4.4: Bootstrap třída

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
    protected function _initDb()
    {
        //inicializace DB adaptéru
    }
}
```

Tvorba vlastních zdrojů

Pod pojmem zdroj zavádění (anglicky Bootstrap Resource) se rozumí třída, která je potomkem třídy `Zend_Application_Resource_ResourceAbstract`. Zend Framework už přináší vlastní zdroje. Jednotlivé zdroje budou vždy vysvětleny u komponenty, která se dá pomocí nich konfigurovat. Každý zdroj se stará o jednu oblast konfigurace vaší aplikace. Tím může jeden zdroj konfigurovat databázové připojení, další pohled a znovu další se stará o konfiguraci komponenty `Zend_Translate` a `Zend_Locale`. Výpis 4.5 demonstruje vytvoření vlastního zdroje, který vytvoří instanci třídy `Mabo_AppInfo` a uloží ji do registru. Ve výpisu 4.6 dále vidíte konfigurační údaje, které daný zdroj použije.

Výpis 4.5: Vytvoření vlastního zdroje

```
<?php
class Mabo_Application_Resource_Appinfo extends
    Zend_Application_Resource_ResourceAbstract
{
    const REGISTRY_KEY = 'AppInfo';

    protected $_appInfo;

    public function init()
    {
        return $this->getAppInfo();
    }

    public function getAppInfo()
    {
        if (null === $this->_appInfo){
            $options = $this->getOptions();
            $name    = $options['name'];
            $author  = $options['author'];
            $version = $options['version'];
            $email   = $options['email'];

            $this->_appInfo = new Mabo_AppInfo(
                $name, $author, $version, $email
            );

            Zend_Registry::set(
                self::REGISTRY_KEY, $this->_appInfo
            );
        }
        return $this->_appInfo;
    }
}
```

Výpis 4.6: Konfigurační údaje

```
resources.appinfo.name = "MaBo eshop"
resources.appinfo.version = "1.0"
resources.appinfo.author = "Marian Böhmer"
resources.appinfo.email = "maboeshop@mabo-eshop.cz"
```

Zend_Tool

`Zend_Tool` jako takový není žádná komponenta, ale jmenný prostor pro komponenty, který poskytuje Zend Frameworku různé nástroje. Vzhledem k tomu neexistuje ani žádná třída `Zend_Tool`.

Instalace Zend_Tool

Po stažení Zend Frameworku z Internetu (viz kapitolu 2) nebo zkopírování z přiloženého CD a jeho rozbalení najdete v adresáři `ZendFramework-x.y.z/bin` (x, y a z určují číslo verze) soubory, pomocí kterých můžete používat komponentu `Zend_Tool` jako konzolovou aplikaci. Pro Unix/Linux systémy je určen soubor `zf.sh` a pro Windows `zf.bat`.

Abyste mohli tuto aplikaci používat pod systémem Unix/Linux, stačí zpravidla jeden příkaz, který musíte v konzole zadat. Tento příkaz vytvoří v systémovém adresáři symbolický odkaz na soubor `zf.sh`. Pro Gentoo Linux vypadá tento příkaz následovně. Zend Framework se nachází v adresáři `/cesta/k/zf/`.

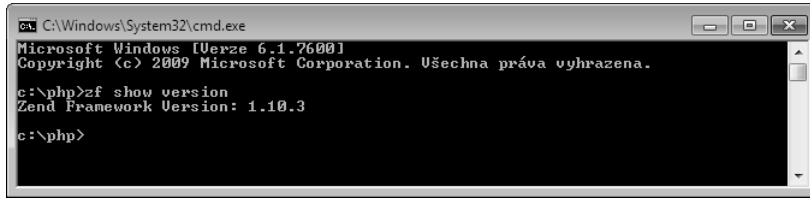
```
sudo ln -s /cesta/k/zf/bin/zf.sh /usr/bin/zf
```

Windows systémy nepodporují úplné symbolické odkazy, a proto musíte postupovat odlišně. Zkopírujte soubory `zf.bat` a `zf.php` z adresáře `bin/` do adresáře, kde se nachází váš `php.exe` soubor, nebo přidejte cestu k němu do systémových proměnných – proměnná `PATH`. Jako druhý krok musíte přidat do souboru `php.ini` do `include_path` cestu k `library/` adresáři Zend Frameworku.

Po instalaci otestujete `Zend_Tool` pomocí následujícího příkazu:

```
zf show version
```

V závislosti na používané verzi Zend Frameworku byste měli uvidět výpis podobný tomu na obrázku 4.1.



Obrázek 4.1: Výpis verze Zend Frameworku pomocí Zend_Tool

Výpis možností, které můžete použít, získáte zavoláním příkazu:

```
zf
```

Konfiguraci PHP získáte po zavolání příkazu:

```
zf show phpinfo
```

Zend_Tool Provider

`Zend_Tool` poskytuje tzv. poskytovatele (anglicky Provider) pro různé účely použití.

Některé z nich jste už poznali v předchozí podkapitole. Pro zobrazení verze jste použili provider `version` a pro zobrazení PHP konfigurace provider `phpinfo`. Každý provider sestává z jedné nebo více akcí, které mohou být vykonány. U provideru `version` stejně jako u `phpinfo` jste použili akci `show`. Jak vidíte, volání příkazu `zf` má vždy stejnou syntaxi:

```
zf <název akce> <název providera>
```

V následující podkapitole vám budou představeny providery `project` a `profile`.

Vytvoření projektu pomocí Zend_Tool

Vytvoření nového projektu pomocí `Zend_Tool` a provideru `project` je velmi jednoduché. V konzole zadáte:

```
zf create project cesta/k/projektu/názevprojektu
```

`Zend_Tool` vykoná následující úlohy:

- Vytvoří nový adresář `cesta/k/projektu/názevprojektu`.
- V adresáři projektu vytvoří adresář `application/` s podadresáři pro konfigurační soubory, řadiče, modely a pohledy. Kromě toho budou vytvořeny další soubory: `Bootstrap.php`, `IndexController` a `ErrorController` stejně jako k nim příslušející skripty pohledů `index/index.phtml` a `error/error.phtml`.
- Vytvoří adresář `library/`.
- Vytvoří adresář `public/`, v něm soubor `.htacces` s pravidly na přepisování URL adres pro Apache server a soubor `index.php`.
- Jako poslední vytvoří soubor `.zfproject.xml` s údaji ohledně vytvořeného Zend Framework projektu.

Toto vám ulehčí trochu práce při vytváření nových projektů.

Pomocí poskytovatele profile zobrazíte obsah souboru `.zfproject.xml` v textové podobě.

```
zf show profile
```

V případě, že se v daném adresáři nenachází soubor `.zfproject.xml`, dostanete příslušné chybové hlášení.

Podrobný popis dalších možností použití a rozšíření této komponenty by byl nad rámec této knihy a hlavně by byl už v době jejího vydání zastaralý. Více ohledně `Zend_Tool` najdete v referenční příručce Zend Frameworku na adrese *http://framework.zend.com/manual/en/zend.tool.html*.

Zend_Exception

`Zend_Exception` není komponenta v pravém slova smyslu. Ve skutečnosti je to potomek třídy `Exception` v PHP 5. I když komponenta `Zend_Exception` nepřispívá žádnou další funkcionalitou, můžete ji použít stejně jako třídu `Exception`. Další informace k PHP třídě `Exception` najdete v PHP manuálu na adrese *http://php.net/manual/en/class.exception.php*.

Skutečný význam komponenty `Zend_Exception` je ten, že každá komponenta obsahuje svoji vlastní `Exception` třídu, která je potomkem třídy `Zend_Exception` a která může obsahovat vlastní programový kód. Tímto je dosaženo toho, že při ošetřování výjimek (anglicky Exception Handling) můžete explicitně uvést, jak chcete reagovat na určitý typ chyby.

Výpis 4.7: Příklad ošetření výjimek pomocí Zend_Exception

```
try {
    $user->authenticate();
} catch (Zend_Db_Exception $e) {
    print 'Chyba během připojování k databázi: ' . $e->getMessage();
} catch (Zend_Auth_Exception $e) {
    print 'Chyba během přihlašování: ' . $e->getMessage();
} catch (Zend_Exception $e) {
    print 'Chyba Zend Frameworku: ' . $e->getMessage();
} catch (Exception $e) {
    print 'PHP chyba: ' . $e->getMessage();
}
```

Ve výpisu 4.7 jsou ošetřeny očekávané chyby během připojování k databázi i během přihlašování. Kromě toho mohou být vyvolány i neočekávané výjimky. Uvedený příklad rozlišuje mezi výjimkami, které vyvolala nějaká komponenta Zend Frameworku, a mezi takovými, které byly vyvolány samotným PHP.

Jak můžete realizovat ošetření výjimek v rámci MVC aplikace, se dočtete v kapitole 5, Řadič.

Zend_Loader

`Zend_Loader` je komponenta, která slouží k načítání souborů. Kromě toho poskytuje tato komponenta i možnost automatického nahrávání souborů.

Nahrávání souborů a tříd

Výpis 4.8 demonstruje použití metod `Zend_Loader::loadFile()` a `Zend_Loader::loadClass()`. U metody `Zend_Loader::loadFile()` se jedná v podstatě jen o obálku pro PHP funkci `include()`.

Výpis 4.8: Příklady použití komponenty Zend_Loader

```
Zend_Loader::loadFile('ZendFramework.php', '/project/Kniha');
Zend_Loader::loadFile('ZendFramework.php', '/project/Kniha', true);

Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Kniha_ZendFramework', '/project');
```

Na prvním řádku bude načten soubor `ZendFramework.php` z adresáře `/project/Kniha`. Na druhém řádku bude načten soubor jen v případě, že předtím ještě načten nebyl.

Příklad na třetím řádku načítá třídu `Zend_Controller_Front` z adresáře, který se musí nacházet v `include_path`. Znaky podtržítka (_) budou změněny na lomítka (/) a na konec názvu bude přidána přípona `.php` tak, aby mohl být soubor `Front.php` načítán z adresáře `Zend/Controller/`. Jako druhý parametr můžete metodě `Zend_Loader::loadClass()` zadat adresář nebo pole s adresáři, v kterých má danou třídu hledat. Adresáře budou prohledávány v pořadí, v jakém byly zadány. Při prvním výskytu požadované třídy se prohledávání ukončí. V případě, že daná třída nebyla nalezena, budou prohledané cesty v `include_path`. Jestliže ani potom nebude daná třída existovat, bude vyvolána výjimka. V posledním příkladu bude třída `Kniha_ZendFramework` načtena z adresáře `/project/Kniha`.

Tip

Komponentu `Zend_Loader` je výhodné použít tehdy, jestliže se název nahrávané třídy nebo souboru dynamicky mění. Při použití statických názvů neposkytuje komponenta `Zend_Loader` v porovnání s PHP funkcí `require_once()` žádné výhody.

Automatické nahrávání

Manuální nahrávání tříd je namáhavé a náchylné na chyby. Jestliže chcete být připraveni na všechny možnosti, musíte nahrát víc tříd, než budete pro zpracování jednoho požadavku potřebovat. Z tohoto důvodu poskytuje komponenta `Zend_Loader_Autoloader` (*http://framework.zend.com/manual/en/zend.loader.autoloader.html*) možnost tzv. automatického nahrávání. Třída `Zend_Loader_Autoloader` implementuje návrhový vzor jedináček (anglicky Singelton). Víc o návrhovém vzoru jedináček se dozvíte v dodatku B této knihy.

Při vytvoření instance této třídy se zaregistrují prefixy `Zend_` a `ZendX_` pro komponenty Zend Frameworku. Jestliže chcete použít vlastní jmenný prostor, zaregistrujete ho pomocí metody `registerNamespace()`.

```
$autoloader = Zend_Loader_Autoloader::getInstance();
$autoloader->registerNamespace('Mabo_');
```

Upozornění

Při registrovaní vlastního jmenného prostoru pomocí funkce `registerNamespace()` musí jeho název obsahovat znak podtržítka (_).

Někdy se stane, že chcete použít knihovny jiných frameworků, které nesplňují konvence Zend Frameworku, mají však definovaný vlastní autoload. V tomto případě využijete metodu `pushAutoloader()`.

```
$autoloader = Zend_Loader_Autoloader::getInstance();
$autoloader->pushAutoloader(
    array('ezcBase', 'autoload'), 'ezc'
);
```

Jako první parametr bude metodě předána metoda autoloaderu a jako druhý parametr prefix, pro který se autoloader použije. Alternativa k metodě `pushAutoloader()` je metoda `unshiftAutoloader()`, která přidá předaný autoloader na prvé místo.

Mezi další zajímavé metody patří metoda `suppressNotFoundWarnings()`, jejímž nastavením bude v případě potřeby potlačeno zobrazování výjimek v případě, že se danou třídu nepodařilo nahrát.

Pokud chcete použít automatické nahrávání pro jakýkoliv jmenný prostor, například u tříd z knihovny PEAR, využijete metodu `setFallbackAutoloader()`.

Resource Autoloaders

Resource Autoloader (*http://framework.zend.com/manual/en/zend.loader.autoloader-resource.html*) je určen pro správu jmenných prostorů knihoven Zend Frameworku. Tyto knihovny splňují konvence Zend Frameworku, avšak mapování názvů tříd a adresářové struktury není 1:1.

Jako dobrý příklad poslouží třídy formulářů a modulů specifické pro aplikaci. Tento příklad předpokládá následující uložení tříd:

- `application/forms/Login.php`
- `application/models/User.php`

A také předpokládá následující: všechny třídy v tomto adresáři mají prefix `Application_`. V rámci adresáře `forms` je použit další prefix `Form_`, výsledná třída se tedy bude nazývat `Application_Form_Login` a bude uložena v souboru `Login.php`. Třídy v rámci adresáře `models` obsahují prefix `Model_`, výsledná třída se tedy bude jmenovat `Application_Model_User` a bude uložena v souboru `User.php`.

V takovémto případě bude vytvoření Resource Autoloaderu vypadat tak, jak je to ukázáno ve výpisu 4.9. Nejprve bude vytvořena nová instance třídy `Zend_Loader_Autoloader_Resource`, které bude jako parametr předáno pole s určením cesty k adresáři a název jmenného prostoru. Za ním následuje registrace jednotlivých typů.

Výpis 4.9: Vytvoření Resource Autoloaderu

```
$resourceLoader = new Zend_Loader_Autoloader_Resource(array(
    'basePath'  => APPLICATION_PATH,
    'namespace' => 'Application'
));

$resourceLoader->addResourceType('form', 'forms/', 'Form')
               ->addResourceType('model', 'models/', 'Model');
```

Komponenta `Zend_Application` obsahuje třídu `Zend_Application_Module_Autoloader`, která přednastavuje různé typy podle doporučené adresářové struktury (*http://framework.zend.com/manual/en/zend.loader.autoloader-resource.html#zend.loader.autoloader-resource.module).*

Nahrávání zásuvných modulů

Pomocí `Zend_Loader_PluginLoader` (*http://framework.zend.com/manual/en/zend.loader.pluginloader.html*) můžete nahrávat zásuvné moduly z různých adresářů. Představte si následujících šest zásuvných modulů:

- `application/modules/blog/controllers/plugins/PluginA.php`
- `application/controllers/plugins/PluginA.php`

- application/controllers/plugins/PluginB.php
- library/Mabo/Controller/Plugin/PluginA.php
- library/Mabo/Controller/Plugin/PluginB.php
- library/Mabo/Controller/Plugin/PluginC.php

Zásuvné moduly v adresáři `library/Mabo/Controller/Plugin/` můžete sice nahrávat pomocí funkce automatického nahrávání, s ostatními to ale nepůjde. V tomto případě vám pomůže `PluginLoader`, který umožňuje nahrávat zásuvné moduly z různých adresářů.

Pomocí metody `addPrefixPath()` můžete přidat adresář s určitým prefixem. `PluginLoader` postupuje při prohledávání přidaných adresářů způsobem LIFO (Last In First Out), to znamená, že posledně přidaný adresář bude prohledán jako první. Výpis 4.10 demonstruje použití `PluginLoader` vztahující se na uvedených šest zásuvných modulů.

Upozornění

Metoda `load()` vrací název třídy, ne její inicializovaný objekt.

Výpis 4.10: Nahrávání zásuvných modulů pomocí PluginLoader

```
$loader = new Zend_Loader_PluginLoader();
$loader->addPrefixPath(
    'Mabo_Controller_Plugin', 'Mabo/Controller/Plugin'
);
$loader->addPrefixPath(
    'Default_Plugin', 'application/controllers/plugins'
);
$loader->addPrefixPath(
    'Blog_Plugin', 'application/modules/blog/controllers/plugins'
);

$pluginAClass = $loader->load('PluginA');
$pluginBClass = $loader->load('PluginB');
$pluginCClass = $loader->load('PluginC');

$pluginA = new $pluginAClass();
$pluginB = new $pluginBClass();
$pluginC = new $pluginCClass();
```

Zásuvné moduly v tomto příkladu jsou jeden za druhým nahrávány a inicializovány. Jak vidíte, je přitom zohledněno pořadí přidaných adresářů.

- application/modules/blog/controllers/plugins/PluginA.php
- application/controllers/plugins/PluginB.php
- library/Mabo/Controller/Plugin/PluginC.php

Zend_Registry

`Zend_Registry` (*http://framework.zend.com/manual/en/zend.registry.html*) implementuje návrhový vzor Registr (viz přílohu B, Návrhový vzor Registr) a poskytuje vám registraci objektů (anglicky Object Registry). K objektům, které jste uložili v registru, můžete přistupovat z libovolné části své aplikace po celou dobu jejího běhu. Ve výpisu 4.11 najdete jednoduchý příklad, který vysvětluje použití této komponenty.

Výpis 4.11: Příklad použití Zend_Registry

```
$translate = new Zend_Translate();
Zend_Registry::set('translate', $translate);

class nejakaTrida
{
    public function nejakaMetoda()
```

```
    {
        $tr = Zend_Registry::get('translate');
        print $tr->translate('Ahoj světe!');
    }
}
```

Nejprve je inicializován objekt `Zend_Translate` a uložen v registru pod indexem `translate`. Následuje definice třídy. V rámci metody definované v této třídě bude načten objekt z registru. Na posledním řádku je vlastní použití objektu.

Tímto jsou za námi časy, kdy se na ukládání objektů musela zneužívat globální PHP pole. Přepisovat objekty definované v registru můžete pomocí metody `Zend_Registry::set()` tak, že použijete při jejím volání už definovaný index.

Třídu `Zend_Registry` můžete rozšířit o přidání vlastní funkcionality. Výpis 4.12 znázorňuje příklad, ve kterém bude třída `Zend_Registry` změněna na základě volání metody `Zend_Registry::setClassName()`. Tato nová třída musí být potomkem třídy `Zend_Registry` a musí být nastavena ještě před prvním přístupem k objektu.

Výpis 4.12: Rozšíření třídy Zend_Registry

```
Zend_Registry::setClassName('Mabo_Registry');

$translate = Zend_Translate();
Zend_Registry::set('translate', $translate);
```

V případě potřeby můžete kompletní registr vrátit do počátečního stavu. Při odstraňovaní objektů může dojít ke ztrátě údajů. Používejte proto tuto metodu opatrně.

```
Zend_Registry::_unsetInstance();
```

Zend_Config

Komponenta `Zend_Config` (*http://framework.zend.com/manual/en/zend.config.html*) je „švýcarský nůž", co se týče konfigurace vaší aplikace, a tím i vhodný kandidát na uložení v `Zend_Registry`, protože právě přístup ke konfiguračním údajům je velmi často požadován na všech úrovních vaší aplikace.

`Zend_Config` dokáže pracovat s klasickými PHP poli i s INI a XML soubory. Nehledě na odlišnou definici konfiguračních souborů zůstává použití komponenty stejné.

Použití PHP polí

Ve výpisu 4.13 je příklad konfiguračního souboru `config.php`, který sestává jen z jednoho PHP pole. Výpis by neměl potřebovat vysvětlení.

Výpis 4.13: Konfigurační soubor jako PHP pole

```
return array (
    'eshop' => 'Mabo eshop',
    'boss'  => array(
        'name'  => 'Mabo',
        'mail'  => 'mabo@mabo-eshop.cz',
        'phone' => '0123/456789'
    )
);
```

Následovně sestrojíte pomocí daného PHP souboru objekt `Zend_Config`:

```
$config = new Zend_Config(require 'config.php');
```

V případě, že potřebujete pouze část pole, postupujte následovně:

```
$data = require 'config.php';
$config = new Zend_Config($data['boss']);
```

Použití INI souborů

Ve výpisu 4.14 vidíte v principu stejné konfigurační údaje jako v předcházejícím souboru. Tentokrát se jedná o INI soubor.

Výpis 4.14: Konfigurační soubor v INI formátu

```
[production]
eshop      = Mabo eshop
boss.name  = Mabo
boss.mail  = mabo@mabo-eshop.cz
boss.phone = 0123/456789

[development : production]
boss.mail = mabo@localhost
```

Konfigurační údaje mohou být dále děleny pomocí znaku tečky (`.`), jako například definice `boss.name`. Na rozdíl od použití PHP polí nabízí `Zend_Config` pro INI soubory podporu sekcí.

Každá sekce je uvedena pomocí označení v hranatých závorkách, například `[production]`. Jestliže má nějaká sekce rozšířit nebo přepsat už definovanou sekci, bude za jejím označením následovat znak dvojtečky (`:`) a název sekce, kterou má rozšířit/přepsat, například `[development : production]`.

`Zend_Config_Ini` objekty pro sekce `production` a `development` vytvoříte následujícím způsobem:

```
$config = new Zend_Config_Ini('config.ini', 'production');
$config = new Zend_Config_Ini('config.ini', 'development');
```

Použití XML souborů

Podobně jako pro INI soubory poskytuje `Zend_Config` podporu sekcí i pro XML soubory. Výpis 4.15 ukazuje stejné konfigurační údaje, tentokrát uložené v XML souboru.

Výpis 4.15: Konfigurační soubor v XML formátu

```
<?xml version="1.0"?>
<configdata>
    <production>
        <eshop>MaBo eshop</eshop>
        <boss>
            <name>MaBo</name>
            <mail>mabo@mabo-eshop.cz</mail>
            <phone>0123/456789</phone>
        </boss>
    </production>
    <development extends="production">
        <boss>
            <mail>mabo@localhost</mail>
        </boss>
    </development>
</configdata>
```

Jednotlivé sekce musí být uzavřeny další značkou. V daném příkladu je použita značka `<configdata>`. V opačném případě není komponenta `Zend_Config_Xml` schopna zadaný soubor správně načíst. Pro rozšíření sekce se používá atribut `extends`.

Vytvoření `Zend_Config_Xml` objektů je velmi podobné:

```
$config = new Zend_Config_Xml('config.ini', 'production');
$config = new Zend_Config_Xml('config.ini', 'development');
```

Přístup ke konfiguračním údajům

Když jste poznali jednotlivé možnosti definice konfiguračních údajů, budete je určitě chtít i použít. Výpis 4.16 demonstruje různé možnosti přístupu ke konfiguračním údajům.

Výpis 4.16: Možnosti přístupu ke konfiguračním údajům

```
$config = new Zend_Config_Ini('config.ini', 'production');

print $config->eshop . "\n";
print $config->boss->mail . "\n";

foreach ($config->boss as $key => $value) {
    print $key . ' -> ' . $value . "\n";
}

$all = $config->toArray();
$boss = $config->boss->toArray();
```

Na začátku je vytvořen objekt `Zend_Config_Ini`. Po vytvoření objektu budou vypsány hodnoty atributů `eshop` a `mail`. Následuje vypsání údajů pomocí `foreach()`. Pomocí metody `toArray()` můžete všechny konfigurační údaje překonvertovat do pole. Toto funguje i pro dílčí části v hierarchii, například `$config->boss->toArray()`.

Zend_Config_Writer

Pomocí této dílčí komponenty můžete konfigurační údaje upravit, respektive vytvořit úplně nové. Z formátů máte k dispozici INI nebo XML soubory i PHP soubor s definovaným polem. Ve výpisu 4.17 najdete malý příklad, který vytvoří nový konfigurační soubor v INI formátu.

Výpis 4.17: Zápis konfiguračních údajů do INI souboru

```
$config = new Zend_Config(array(), true);

$config->kniha = array('Zend Framework', 'Kniha 1', 'Kniha 2');
$config->settings = array();
$config->settings->foo = 'bar';
$config->settings->bar = 'foo';

$writer = new Zend_Config_Writer_Ini();
$writer->write('../tmp/config.ini', $config);
```

Jestliže chcete upravit konfigurační soubor, načtěte ho do `Zend_Config` objektu. Změňte parametr `allowModifications` na `true`, čímž se stanou hodnoty modifikovatelné. Při změně hodnoty nebude vyvolána výjimka.

```
$config = new Zend_Config(require 'config.php', true);
$config = new Zend_Config_Ini('config.ini', 'production', true);
$config = new Zend_Config_Xml('config.ini', 'production', true);
```

Uložení proběhne jako v předcházejícím příkladu. `Zend_Config_Writer` též podporuje použití sekcí. Příklad k této problematice najdete v referenční příručce Zend Frameworku na adrese *http://framework.zend.com/manual/en/zend.config.writer.html*.

Zend_Cache

Komponenta `Zend_Cache` (*http://framework.zend.com/manual/en/zend.cache.html*) je jednou z nejstarších komponent Zend Frameworku. Jak už sám název napovídá, je určena pro kešování údajů. Kromě jednotlivých stránek nebo jen jejich částí můžete pomocí této komponenty kešovat i funkce a metody, což se značně odrazí na výkonu vaší aplikace.

Při používání komponenty `Zend_Cache` musíte rozlišovat frontendy a backendy. Při výběru frontendu určujete, které údaje mají být uloženy ve vyrovnávací paměti. Backend určuje, kde, respektive jak, mají být údaje uloženy.

Ve výpisu 4.18 najdete jednoduchý příklad, jak vytvořit `Zend_Cache` objekt. Při volání metody `Zend_Cache::factory()` musí být uvedeny typy frontendu a backendu jako i další konfigurační údaje. Vytvořený objekt bude nakonec uložen v registru, abyste k němu mohli kdykoliv přistupovat.

Výpis 4.18: Příklad vytvoření Zend_Cache objektu

```
$optionsFront = array(
    'lifetime'                => 3600,
    'automatic_serialization' => true
);

$optionsBack = array(
    'cache_dir' => '../data/cache/'
);

$cache = Zend_Cache::factory(
    'Core', 'File', $optionsFront, $optionsBack
);
Zend_Registry::set('cache', $cache);
```

`Zend_Cache` se dobře kombinuje s komponentami `Zend_Translate` (kapitola 10, Internacionalizace a lokalizace) nebo s komponentou `Zend_Db_Table` (kapitola 7, Databáze).

Backendy pro Zend_Cache

Při určování backendu v metodě `Zend_Cache::factory()` stačí uvést pouze poslední část názvu třídy, prefix `Zend_Cache_Backend_` můžete vynechat. Úplný přehled konfiguračních možností najdete v referenční příručce na adrese *http://framework.zend.com/manual/en/zend.cache.backends.html*.

`Zend_Cache_Backend_File` ukládá kešované údaje do souborů. Důležité parametry při konfiguraci jsou: `cache_dir`, který určuje adresář, do kterého jsou ukládány kešované údaje. `cache_file_umask` určuje oprávnění pro vytvořené soubory s údaji. Parametr `hashed_directory_level` určuje, jestli se mají v adresáři `cache_dir` použít i podadresáře. Toto je výhodné při práci s velkým množstvím kešovacích souborů. `hashed_directory_level` udává hloubku podadresářů. Doporučené hodnoty jsou `1` nebo `2`, standardně je hodnota nastavena na `0`. To znamená, že v rámci `cache_dir` adresáře nemohou být vytvořeny žádné další podadresáře.

`Zend_Cache_Backend_Sqlite` ukládá kešované údaje do SQLite databáze. Mezi konfigurační parametry patří `cache_db_complete_path` (povinný údaj), který určuje cestu k databázi, a `automatic_vacuum_factor`, který slouží k řízení defragmentace databáze. Hodnota `0` znamená žádná defragmentace. Při hodnotě `1` bude defragmentace vykonána při každém odstranění záznamu. Při vyšších hodnotách bude defragmentace vykonávána náhodně. Hodnota `10` určuje pravděpodobnost 1 : 10.

`Zend_Cache_Backend_Memcached` používá Memcache server. K tomu potřebujete memcached démona a `memcache` PECL rozšíření pro PHP. Při konfiguraci udáváte parametrem `servers` pole se servery, které mají být použity.

`Zend_Cache_Backend_Apc` ukládá soubory do sdílené paměti (anglicky Shared Memory) a vyžaduje APC rozšíření pro PHP.

`Zend_Cache_Backend_Xcache` ukládá údaje také do sdílené paměti a vyžaduje Xcache rozšíření. Parametry jsou `user` a `password`.

`Zend_Cache_Backend_ZendPlatform` používá Zend platformu. Při zadávání tohoto typu backendu metodě `Zend_Cache::factory()` musíte namísto `ZendPlatform` uvést `Zend Platform`. V tomto případě nejsou nutné žádné další konfigurační parametry. Samozřejmostí však je, že máte nainstalovanou Zend Platformu (*http://www.zend.com/products/platform/*).

`Zend_Cache` je sestrojena tak, že můžete implementovat i další typy backendů.

Kešování databázových dotazů

Jedním z nejčastějších případů kešování je kešování databázových dotazů. V tomto případe se jako frontend používá `Zend_Cache_Core`. Tento frontend je základem pro všechny ostatní frontendy, které jsou od něj i odvozeny. Všechny konfigurační parametry platné pro `Zend_Cache_Core` můžete proto použít i u ostatních typů.

Důležité parametry jsou `caching` pro zapnutí a vypnutí kešování, `lifetime` pro určení doby platnosti v sekundách a `automatic_serialization` pro zapnutí a vypnutí automatické serializace údajů. Úplný přehled všech parametrů najdete v referenční příručce na adrese *http://framework.zend.com/manual/en/zend.cache.frontends.html#zend.cache.frontends.core*.

Výpis 4.19 zobrazuje jednoduchý příklad kešování databázových dotazů pomocí `Zend_Cache_Core`. Na začátku příkladu bude inicializován `Zend_Cache` objekt, kde bude jako backend použit `Zend_Cache_Backend_File`.

Výpis 4.19: Uložení databázového dotazu pomocí Zend_Cache_Core

```
$optionsFront = array(
    'lifetime' => 3600
);
$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
    'Core', 'File', $optionsFront, $optionsBack
);

$cacheId = 'seznamknih';

if(!$bookList = $cache->load($cacheId)) {
    $bootstrap = $this->getInvokeArg('bootstrap');
    $db = $bootstrap->getResource('db');
    $bookList = $db->fetchAll(
        'SELECT * FROM books ORDER BY name'
    );
    $cache->save($bookList, $cacheId);
}
```

Na začátku se stanoví tzv. Cache ID, aby bylo možné identifikovat údaje ve vyrovnávací paměti. V případě, že načítání dat z vyrovnávací paměti nebylo úspěšné, vykoná se jejich načítání z databáze a uloží se do vyrovnávací paměti pod Cache ID. U `$this` se v tomto případě jedná o objekt action controlleru – instanci třídy `Zend_Controller_Action`. Jestliže bylo načítání úspěšné, obsahuje proměnná `$bookList` seznam knih.

`Zend_Cache` můžete také použít na kešování databázových dotazů v rámci modelu nebo v rámci třídy `Zend_Db_Table` (viz kapitolu 7, Databáze). Také nesmíte zapomenout v případě potřeby na vymazání vyrovnávací paměti, když to aplikace vyžaduje.

Kešování funkcí

Na kešování funkcí slouží `Zend_Cache_Frontend_Function`. Tento frontend obsahuje kromě standardních konfiguračních parametrů i parametry `cached_functions` a `non_cached_functions`. První parametr udává pole funkcí, které budou vždy kešované. Druhý parametr je opakem prvního a určuje funkce, které nebudou nikdy kešované. Kromě toho `cache_by_default` udává, zda mají být funkce standardně uloženy ve vyrovnávací paměti. Kešované mohou být všechny zabudované PHP funkce i funkce vlastní. Výjimku tvoří jen funkce `array()`, `echo()`, `empty()`, `eval()`, `exit()`, `isset()`, `list()`, `print()` a `unset()`.

Příklad kešování funkcí pomocí `Zend_Cache_Frontend_Function` demonstruje výpis 4.20. Funkce `random()` má být kešovaná jen v případě, že parametr `cache_by_default` je nastavený na hodnotu `true`, funkce `sub()` vždy a funkce `add()` nemá být kešovaná nikdy. Všechny funkce mají být ve vyrovnávací paměti uloženy maximálně dvě sekundy.

Výpis 4.20: Uložení volání funkcí pomocí Zend_Cache_Frontend_Function

```
function random()
{
    return mt_rand(200, 899);
}
function sub($number)
{
    return $number - mt_rand(10, 99);
}
function add($number)
{
    return $number + mt_rand(10, 99);
}

$optionsFront = array(
    'lifetime'             => 2,
    'cache_by_default'     => true,
    'cached_functions'     => array('sub'),
    'non_cached_functions' => array('add'),
);

$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
    'Function', 'File', $optionsFront, $optionsBack
);

for ($i = 0; $i < 5; $i++) {
    $number = $cache->call('random');

    print random() . ' - ';
    print $number . ' | ';
    print sub($number) . ' - ';
    print $cache->call('sub', array($number)) . ' | ';
    print add($number) . ' - ';
    print $cache->call('add', array($number)) . '<br />';

    sleep(1);
}
```

Na začátku příkladu jsou definovány funkce a vytvořen `Zend_Cache` objekt. V těle cyklu, který se vykoná pětkrát, je pomocí funkce `random()` vygenerováno náhodné číslo a v případě potřeby uloženo do vyrovnávací paměti. Následuje střídavé volání funkcí – jedna přímo a jedna přes `Zend_Cache` objekt. Nakonec je vypsán výsledek. Mezi jednotlivými cykly je jednosekundová pauza. Obrázek 4.2 ukazuje výstup z výpisu 4.20.

883 - 626 | 605 - 613 | 700 - 709
721 - 626 | 541 - 613 | 665 - 708
303 - 626 | 560 - 613 | 722 - 722
410 - 569 | 508 - 532 | 634 - 579
535 - 569 | 543 - 532 | 666 - 652

Obrázek 4.2: Příklad z výpisu 4.20

Kešování metod třídy

Podobně jako pro funkce funguje i kešování metod pomocí `Zend_Cache_Frontend_Class`. Kromě standardních parametrů potřebuje tento frontend i parametr `cached_entity` udávající název třídy a parametr `cache_by_default`, který funguje stejně jako u `Zend_Cache_Frontend_Function`. Parametry udávající názvy metod, které mají být vždy, respektive nemají být nikdy, kešované, jsou `cached_methods` a `non_cached_methods`.

Výpis 4.21 demonstruje stejný příklad jako výpis 4.20, pouze namísto funkcí budou kešované metody třídy. Definována je třída se třemi metodami. Pro danou třídu bude vytvořen objekt a jeho instance bude předána parametru `cached_entity`. Zbytek příkladu je velmi podobný tomu předcházejícímu, jenom metody jako takové jsou volány jinak. Namísto volání pomocí metody `call()` objektu `Zend_Config` mohou být volány přímo jako metody `Zend_Config` objektu.

Výpis 4.21: Uložení volání metod třídy pomocí Zend_Cache_Frontend_Class

```
class myClass
{
    public function random()
    {
        return mt_rand(200, 899);
    }
    public function sub($number)
    {
        return $number - mt_rand(10, 99);
    }
    public function add($number)
    {
        return $number + mt_rand(10, 99);
    }
}
$myObject = new myClass();

$optionsFront = array(
    'lifetime'           => 2,
    'cached_entity'      => $myObject,
    'cache_by_default'   => true,
    'cached_methods'     => array('sub'),
    'non_cached_methods' => array('add'),
);
$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
    'Class', 'File', $optionsFront, $optionsBack
);

for ($i = 0; $i < 10; $i++) {
    $number = $cache->random();

    print $myObject->random() . ' - ';
    print $number . ' | ';
    print $myObject->sub($number) . ' - ';
    print $cache->sub($number) . ' | ';
    print $myObject->add($number) . ' - ';
    print $cache->add($number) . '<br />';

    sleep(1);
}
```

`Zend_Cache_Frontend_Class` můžete používat i pro statické metody. V tomto případě nepředáte parametru `cached_entity` instanci objektu, ale název dané třídy, která obsahuje statické metody. Ostatní se nemění.

Kešování souborů

Představte si komplexní XML soubor nebo nekonečně dlouhý CSV soubor, které musíte načíst a zpracovat ještě předtím, než použijete údaje v nich obsažené.

Přesně pro tyto případy, kdy musí být soubor načten a zpracován, je určen frontend `Zend_Cache_Frontend_File`. Kromě standardních konfiguračních parametrů je ještě potřeba parametr `master_files`, který udává pole kešovaných souborů včetně cest k nim. Parametr `automatic_serialization` musí být nastaven na hodnotu `true` pro korektní ukládání objektů a polí.

Na příkladu ve výpisu 4.22 je ukázáno kešování souborů, kde bude načten XML soubor a pomocí funkce `simplexml_load_file()` překonvertován na objekt.

Výpis 4.22 Ukládání souborů pomocí Zend_Cache_Frontend_File

```
$xmlFile = 'data.xml';
$optionsFront = array(
    'lifetime'                => 2,
    'automatic_serialization' => true,
    'master_files'            => array($xmlFile),
);
$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
    'File', 'File', $optionsFront, $optionsBack
);

$cacheId = md5($xmlFile);

if(!$xmlObject = $cache->load($cacheId)) {
    $xmlObject = simplexml_load_file($xmlFile);

    $cache->save($xmlObject, $cacheId);
}
```

Na začátku příkladu jsou nastaveny parametry kešování, název kešovaného souboru a vytvořený `Zend_Cache` objekt. Cache ID je generované z názvu souboru pomocí `md5()` funkce. V případě, že volání dat z vyrovnávací paměti nebylo úspěšné, jsou data zpracována funkcí `simplexml_load_file()` a vložena do vyrovnávací paměti.

Kešování výstupu pro prohlížeče

Na kešování výstupu, který má být odeslán do prohlížeče uživatele, máte k dispozici `Zend_Cache_Frontend_Output`. Tento frontend neobsahuje žádné další konfigurační parametry kromě těch poskytovaných třídou `Zend_Cache_Core`. Ve výpisu 4.23 najdete jednoduchý příklad, který demonstruje použití tohoto typu frontendu.

Výpis 4.23: Kešování výstupu pomocí Zend_Cache_Frontend_Output

```
$optionsFront = array(
    'lifetime' => 5,
);
$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
```

```
    'Output', 'File', $optionsFront, $optionsBack
);

$cacheId = 'seznamknih';

print '<h1>MaBo eshop - seznam knih</h1>';

if(!$cache->start($cacheId)) {
    print 'Kniha Zend Framework byla objednána v '
    . date('H:i:s', time()) . ' <br />';

    $cache->end();
}

print '<br />';
print '<em>Aktuální čas ' . date('H:i:s', time()) . ' </em><br />';
```

Kešování celých stránek

Jestliže nechcete kešovat jen dílčí výstupy, ale rovnou celé stránky, máte k dispozici frontend `Zend_Cache_Frontend_Page`.

Na rozdíl od všech ostatních zatím probíraných typů frondendů si tento frontend stanoví cache ID sám. Použije na to hodnotu proměnné `$_SERVER['REQUEST_URI']`. V závislosti na nastavených parametrech budou k sestavení cache ID použity i hodnoty z proměnných `$_GET`, `$_POST`, `$_SESSION`, `$_COOKIE` a `$_FILES`. Například mohou být rozdílně stanovena cache ID pro stránky */book/show/id/1* a */book/show/id/2*, ale také i pro stránku */book/list* v případě, že se hodnoty v proměnné `$_GET` odlišují a toto pole se podílí na tvorbě cache ID.

Konfigurační parametry pro `Zend_Config_Cache_Page` jsou trochu komplexnější než pro ostatní frontendy. Parametr `debug_header` udává, jestli má být před každou kešovanou stránku přidán text používaný při ladění (anglicky debugging). Parametr `default_options` obsahuje pole se standardními hodnotami. Parametr `regexps` definuje pole obsahující údaje pro stránky, které se odlišují od těch předvolených. Indexy tohoto pole jsou regulární výrazy, jeho hodnoty jsou asociativní pole s parametry. Pomocí parametru `memorize_headers` můžete určit, které hlavičky mají být vypsány, jestliže byla stránka úspěšně načtena z vyrovnávací paměti. Kromě toho platí už známé parametry ze `Zend_Cache_Core`.

Parametr `default_options` obsahuje pole s předvolenými hodnotami. Pomocí `cache` určíte, jestli má být stránka kešovaná (předvolené `true`). Pokud je `cache_with_get_variables` nastaveno na `true`, bude daná stránka kešovaná i v případě, že proměnná `$_GET` obsahuje nějaké hodnoty. Předvolená hodnota je `false`. V případě, že je `make_id_with_cache_variables` nastaveno na hodnotu `true`, bude obsah proměnné `$_GET` též zahrnut od tvorby cache ID. Předvolená hodnota pro `make_id_with_cache_variables` je `true`. Kromě parametrů pro `$_GET` existují stejné parametry i pro `$_POST`, `$_SESSION`, `$_COOKIE` a `$_FILES`, jejichž předvolené hodnoty vypadají stejně.

Jestliže nezměníte žádnou z předvolených hodnot, bude ve vyrovnávací paměti vytvořen záznam pro každou jedinečnou `REQUEST_URI`. V případě, že některé z PHP polí `$_GET`, `$_POST`, `$_SESSION`, `$_COOKIE` a `$_FILES` obsahuje údaje, nebude daná stránka kešovaná.

Výpis 4.24 vysvětluje daný typ frontendu na jednoduchém příkladu. Daný kód je nejlepší vložit do souboru `Bootstrap.php`, aby v případě úspěšného nalezení stránky byla tato načítána z vyrovnávací paměti a proces zavádění aplikace byl přerušen.

Výpis 4.24: Ukládání stránek pomocí Zend_Cache_Frontend_Page

```
$optionsFront = array(
    'lifetime'        => 7200,
```

```
    'default_options' => array(
        'cache_with_get_variables'  => true,
    ),
    'regexps' => array(
        '^/$'       => array('cache' => false),
        '^/admin/$' => array('cache' => false),
        '^/book/$'  => array('cache' => true),
        '^/book/create$' => array('cache' => false),
        '^/book/search$' => array(
            'cache'                       => true,
            'cache_with_post_variables'   => true,
            'make_id_with_post_variables' => true,
        ),
    ),
);

$optionsBack = array(
    'cache_dir' => '../data/cache/'
);
$cache = Zend_Cache::factory(
    'Page', 'File', $optionsFront, $optionsBack
);

$cache->start();

print '<h1>MaBo eshop</h1>';
```

Na začátku příkladu jsou definovány parametry pro použitý frontend. Na základě parametrů v `default_options` jsou stránky kešované i v případě, že pole `$_GET` obsahuje údaje. Údaje v `regexps` určují způsob kešování pro určité stránky, kde má být toto odlišné od standardního nastavení.

- Nekešovat hlavní stránku.
- Nekešovat stránky určené pro administrátory.
- Vždy kešovat stránky začínající */book*.
- Nekešovat stránky, kde se přidávají nové knihy.
- Vždy kešovat stránky, na kterých se vyhledávají knihy, i v tom případě, že proměnná `$_POST` obsahuje nějaké údaje, tyto údaje použít na vytvoření cache ID.

Poslední bod má smysl tehdy, když je vyhledávání knih realizováno přes formulář a hodnoty z něj odesílány metodou `POST`. V tom případě se nacházejí odeslané údaje v proměnné `$_POST` a měly by se zohledňovat při vytváření cache ID.

Dále v příkladu následuje vytvoření `Zend_Cache` objektu a volání metody `start()`. V případě, že ve vyrovnávací paměti neexistuje pro volanou stránku žádný platný záznam, pokračuje se dále v programu. V opačném případě bude daná stránka načítána z vyrovnávací paměti a další vykonávání programu bude přerušeno.

Použití značek

Pomocí značek můžete označovat uložené záznamy ve vyrovnávací paměti. Značky jsou předané metodě zodpovědné za ukládání údajů do vyrovnávací paměti ve formě pole (například `array('kniha', 'zendframework')`). V závislosti na zvoleném frontendu jsou tyto metody rozdílné.

Pro `Zend_Cache_Core` a `Zend_Cache_Frontend_File` je to třetí parametr metody `save()`. Pro `Zend_Cache_Frontend_Function` je to třetí parametr metody `call()`. Pro `Zend_Cache_Frontend_Output` je to první parametr metody `end()`. Pro `Zend_Cache_Frontend_Class` musí být značky nastaveny pomocí metody `setTagsArray()` objektu `Zend_Cache` ještě před voláním kešovací metody. `Zend_Cache_Frontend_Page` zatím podporu značek neposkytuje.

Pár příkladů na demonstraci:

```
//nastavení značek při použití frontendu Core
$cache->save($data, $cacheId, array('tag1', 'tag2'));
// nastavení značek při použití frontendu Function
$cache->call('function', array($param), array('tag1', 'tag2'));
// nastavení značek při použití frontendu Output
$cache->end(array('tag1', 'tag2'));
// nastavení značek při použití frontendu Class
$cache->setTagsArray(array('tag1', 'tag2'));
$cache->method($param);
```

Kromě omezení ve frontendech byste měli mít též na paměti, že backendy `Zend_Cache_Backend_Memcached`, `Zend_Cache_Backend_Apc` a `Zend_Cache_Backend_Xcache` nepodporují značky. `Zend_Cache_Backend_ZendPlatform` sice značky ve všeobecnosti podporuje, ale nepodporuje způsob mazání `CLEANING_MODE_NOT_MATCHING_TAG`.

Vyprázdnění vyrovnávací paměti

Pro odstranění záznamu z vyrovnávací paměti použijte v případě, že znáte jeho cache ID, metodu `remove()`:

```
$cache->remove($cacheId);
```

Jestliže chcete smazat víc než jeden záznam, použijte metodu `clean()`.

```
//smaže všechny záznamy
$cache->clean(Zend_Cache::CLEANING_MODE_ALL);

//smaže všechny záznamy s vypršenou dobou životnosti
$cache->clean(Zend_Cache::CLEANING_MODE_OLD);

//smaže všechny záznamy obsahující značky
//'kniha' a 'zendframework'
$cache->clean(Zend_Cache::CLEANING_MODE_MATCHING_TAG,
    array('kniha', 'zendframework')
);
//smaže všechny záznamy neobsahující značky
//'kniha' a 'zendframework'
$cache->clean(Zend_Cache::CLEANING_MODE_NOT_MATCHING_TAG,
    array('kniha', 'zendframework')
);
```

Poslední dva příklady platí jen pro `Zend_Cache` objekty, jejichž frontendy a backendy podporují značky.

Cache Manager

V praxi se člověk často střetne se situací, kdy potřebuje použít různé typy vyrovnávací paměti v závislosti na použitém řadiči, knihovně nebo modelu, ke kterým se přistupuje. Cache Manager (*http://framework.zend.com/manual/en/zend.cache.cache.manager.html)* byl vytvořen za účelem zjednodušení konfigurace cache objektů a také za účelem minimalizace konfigurace ve zdrojovém kódu aplikace. Používán je především třídou `Zend_Application_Resource_Cachemanager`, která zabezpečí načítání konfigurace při zavádění aplikace, a také je používán třídou `Zend_Controller_Action_Helper_Cache`.

Cache Manager poskytuje konfiguraci jednotlivých typů vyrovnávací paměti v podobě tzv. šablon (anglicky Templates). Jednotlivé typy vyrovnávacích pamětí nebudou inicializovány okamžitě, ale až v případě, kdy k nim uživatel přistoupí pomocí metody `getCache()`. Ve výpisu 4.25 je na jednoduchém příkladu vysvětleno používání Cache Manageru.

Výpis 4.25: Konfigurace vyrovnávací paměti pomocí Cache Manageru

```
$manager = new Zend_Cache_Manager();

$dbCache = array(
    'frontend' => array(
        'name' => 'Core',
        'options' => array(
            'lifetime' => 3600,
            'automatic_serialization' => true
        )
    ),
    'backend' => array(
        'name' => 'File',
        'options' => array(
            'cache_dir' => '../data/cache'
        )
    )
);

$manager->setCacheTemplate('database', $dbCache);
```

Na začátku příkladu je inicializován Cache Manager, následuje konfigurace frontend a backend parametrů a nakonec je daná šablona uložena pomocí metody `setCacheTemplate()`. Načítání uložené šablony se vykoná pomocí metody `getCacheTemplate()` a načítání cache instance pomocí metody `getCache()`. Oběma metodám se předá jako parametr název požadované vyrovnávací paměti. V případě, že chcete zjistit, zda je vámi požadovaná vyrovnávací paměť spravována Cache Managerem, máte k dispozici metodu `hasCahe()` pro instance a `hasCacheTemplate()` pro šablony.

Konfigurace vyrovnávací paměti pomocí konfiguračního souboru

Pro tento účel poskytuje komponenta `Zend_Application` zásuvný modul `Zend_Application_Resource_Cachemanager`, pomocí kterého je možné nakonfigurovat různé typy vyrovnávací paměti z konfiguračního souboru. Výpis 4.26 demonstruje konfiguraci z předcházejícího výpisu pomocí konfiguračního souboru.

Výpis 4.26: Konfigurace vyrovnávací paměti pomocí konfiguračního souboru

```
resources.cachemanager.database.frontend.name = Core
resources.cachemanager.database.frontend.options.lifetime = 3600
resources.cachemanager.database.frontend.options.automatic_serialization = true
resources.cachemanager.database.backend.name = File
resources.cachemanager.database.backend.options.cache_dir =
    APPLICATION_PATH "/../data/cache"
```

Zend_Log

Komponenta `Zend_Log` (*http://framework.zend.com/manual/en/zend.log.html*) slouží k zaznamenávání událostí. Tato komponenta neposkytuje jen logování do souborů, ale i do databázové tabulky nebo Firebug konzoly.

Každá událost, kterou chcete zaznamenat, dostane přidělený určitý stupeň priority. `Zend_Log` nabízí standardně osm typů priorit (`EMERG`, `ALERT`, `CRIT`, `ERR`, `WARN`, `NOTICE`, `INFO`, `DEBUG`).

Logování do souboru

Nejpoužívanější možnost je používání jednoduchých logovacích souborů. Ve výpisu 4.27 je ukázáno vytvoření logovacího souboru. Na začátku je inicializován `Zend_Log` objekt. Po inicializaci následuje přidání tzv.

zapisovače (anglicky Writer), který má v tomto příkladu zapisovat do souboru. `Zend_Log` můžete vybavit i víc než jedním zapisovačem, jestliže chcete například současně zapisovat do souboru a do databáze.

Výpis 4.27: Logování do souboru pomocí Zend_Log

```
$logger = new Zend_Log();
$logger->addWriter(
    new Zend_Log_Writer_Stream('../data/logs/log.txt')
);

$logger->info('Objednávka odeslaná');
$logger->log('Zásoby docházejí', Zend_Log::ALERT);
$logger->warn('Kniha není na skladě');
```

Na konci příkladu jsou zaznamenané tři události. Pro každou prioritu můžete použít vlastní metodu, přičemž ta metoda odpovídá zkratce dané priority napsané malými písmeny. Kromě toho můžete použít i metodu `log()` a jako druhý parametr uvést prioritu jako konstantu třídy `Zend_Log`.

Zapisovač `Zend_Log_Writer_Stream` nabízí kromě zapisování do souboru i možnost zapisování do PHP streamu. Co přesně streamy znamenají a jak je můžete použít, se dočtete v PHP manuálu na adrese *http://www.php.net/stream*.

Logování do databáze

Logování do databáze funguje podobně jako logovaní do souboru. Potřebujete k tomu jenom použít jiný typ zapisovače `Zend_Log_Writer_Db`. Tento zapisovač však potřebuje jinou konfiguraci.

Výpis 4.28 na příkladu demonstruje použití `Zend_Log_Writer_Db`. Podmínkami použití jsou přístupové údaje k databázi a existence tabulky, do které se má zapisovat.

Výpis 4.28: Logování do databáze pomocí Zend:Log

```
$db = Zend_Db::factory($dbType, $dbParams);
$tableName = 'system_log';
$columnMapping = array(
    'date'  => 'timestamp',
    'level' => 'priority',
    'text'  => 'message',
);

$logger = new Zend_Log();
$logger->addWriter(
    new Zend_Log_Writer_Db($db, $tableName, $columnMapping)
);

$logger->info('Objednávka odeslaná');
$logger->log('Zásoby docházejí', Zend_Log::ALERT);
$logger->warn('Kniha není na skladě');
```

V úvodu je inicializován databázový adaptér a určen název tabulky a ke sloupcům v tabulce jsou přiřazeny hodnoty. Dále je inicializován objekt `Zend_Log` a přiřazen k němu zapisovač `Zend_Log_Writer_Db`, kterému jsou předány předem nakonfigurované parametry.

Logování do Firebug konzoly

V případě, že používáte Mozilla Firefox a máte nainstalováno rozšíření Firebug, bude pro vás následující příklad jistě zajímavý. Jestliže chcete pomocí `Zend_Log` logovat do Firebug konzoly, musíte mít nainstalováno následující:

- Prohlížeč Firefox, ideálně verze 3, verze 2 je též podporována
- Rozšíření Firebug (*https://addons.mozilla.org/en-US/firefox/addon/1843*)

- Rozšíření FirePHP (*https://addons.mozilla.org/en-US/firefox/addon/6149*)

Výpis 4.29 demonstruje použití `Zend_Log_Write_Firebug`. Než budete moci logovat, musíte vytvořit kanál pro `Zend_Wildfire` a údaje po skončení logování poslat do prohlížeče.

Výpis 4.29: Logování do Firebug konzoly

```
$logger = new Zend_Log();
$logger->addWriter(new Zend_Log_Writer_Firebug());

$request = new Zend_Controller_Request_Http();
$response = new Zend_Controller_Response_Http();
$channel = Zend_Wildfire_Channel_HttpHeaders::getInstance();
$channel->setRequest($request);
$channel->setResponse($response);

ob_start();

$logger->info('Objednávka odeslaná');
$logger->log('Zásoby docházejí', Zend_Log::ALERT);
$logger->warn('Kniha není na skladě');

$channel->flush();
$response->sendHeaders();
```

Tento příklad vám má objasnit jen všeobecný způsob činnosti. Funguje jen tehdy, když nepoužíváte `Zend_Controller`. Pro další otázky a také příklad použití s komponentou `Zend_Controller` nahlédněte do referenční příručky ke komponentám `Zend_Log` (*http://framework.zend.com/manual/en/zend.log.writers.html#zend.log.writers.firebug)* a `Zend_Wildfire` (*http://framework.zend.com/manual/en/zend.wildfire.html)*.

Další možnosti logování

Jestliže vám existujících osm priorit nestačí, můžete používat i vlastní. Existující priority však nesmí být přepsány.

```
$logger->addPriority('KNIHA', 8);

$logger->log('Kniha je na skladě', 8);
$logger->kniha('Kniha je na skladě');
```

Změnit standardní formátování v log souborech též není žádný problém. Zapisování údajů do XML souboru namísto klasických záznamů vykonáte následovně:

```
$writer = new Zend_Log_Writer_Stream('../data/logs/log.txt');
$writer->setFormatter(new Zend_Log_Formatter_Xml());

$logger->addWriter($writer);
```

Při klasickém řádkovém zápisu můžete též změnit pořadí jednotlivých údajů:

```
$format = '%timestamp% - %priority% '
        . '(%priorityName%) -> %message%' . PHP_EOL;
$formatter = new Zend_Log_Formatter_Simple($format);

$writer = new Zend_Log_Writer_Stream(
    '../data/logs/log.txt'
);
$writer->setFormatter($formatter);
```

Jestliže si nevystačíte s údaji `timestamp`, `priority`, `priorityName` a `message`, můžete si stejným způsobem definovat vlastní. Po definici musí být jejich existence též oznámená aktuálnímu formátovači.

```
$logger->setEventItem('pid', getmypid());
```