HANDLING DATABASE OBJECTS IN PHP

Ondřej Navrátil
Master Degree Programme (2), FIT BUT
E-mail: xnavra23@stud.fit.vutbr

Supervised by: Alexander Meduna
E-mail: meduna@fit.vutbr.cz

Abstract: In the development of dynamic web applications, it is a common need to provide an interface for manipulation with objects based on database entries. Performance and straight-forward use are essential requirements of such tools in order to eliminate verbosity of SQL language and reduce the number of queries. This paper presents PHP class DbObject, which is aimed to manipulate standard data models in an effective and easy-to-use way. Basic ideas are introduced along with examples and compared with its analogies in popular Zend framework.

Keywords: PHP, MySQL, Zend, Database abstraction, Late static bindings

1 ÚVOD

Jazyk PHP a DBMS MySQL jsou technologie hojně využívané při vývoji webových aplikací. Některé vlastnosti SQL jazyka však mohou zpomalovat vývoj či snižovat čitelnost kódu. Vývojáři proto zpravidla používají abstraktní programové vrstvy pro odstínění těchto nežádoucích jevů. Různé frameworky poskytují pro tyto účely rozličné třídy, obecně však pracují se schématem databáze, tabulka, záznam. Práce s kompozitními objekty rozprostřenými mezi více tabulek zpravidla není nijak implicitně poskytována.

Následující text se věnuje autorem navržené a implementované třídě `DbObject`, jejímž účelem je překlenout tyto nedostatky s pomocí dynamických vlastností jazyka PHP. Práce popisuje též další utility třídy a příklady použití. Skript je dostupný na autorově githubu[1].

2 ZÁKLADNÍ PRÁCE S TŘÍDOU DBOBJECT

Pro použití stačí podědit z `DbObject` a implementovat metody `dbConnection` a `tables`. První metodou určíme připojení k databázi (často jen vrací superglobální proměnnou), posledně jmenovaná definuje pole tabulek, se kterými bude daná třída objektů pracovat. `DbObject` si pomocí SQL metody `DESCRIBE` zjistí strukturu těchto tabulek, kterou později použije pro vkládání, mazání a úpravu řádků.

Obě zmíněné metody jsou statické, neb jsou potřebné i v několika dalších statických utilitách. Pro správné určení jmenného prostoru je použito pozdní statické vazby[3] představené ve verzi PHP 5.3.0, která umožňuje pomocí jmenného prostoru `static` dynamicky přetěžovat statické metody podobně jako je tomu u polymorfních objektů.

Je-li objekt uložen ve více tabulkách, `DbObject` předpokládá uvedení tabulek v pořadí od nejobecnějšího nejspecifičtější, tedy že každá tabulka kromě první obsahuje pole omezené cizím klíčem odkazujícím na předchozí tabulku (např. `array("objednavky", "objednavky_textil")` pro hypotetický eshop). Toto je podstatné pro vkládání a mazání objektů, kdy je třeba zachovávat integritu databáze.

[1] `https://github.com/LordNavro/DbObject`

S třídou můžeme ihned pracovat. Po vytvoření objektu konstruktorem lze získávat/nastavovat hodnoty jednotlivých atributů metodami `getAttr/setAttr`. Vytvořený objekt vložíme do databáze metodou `insert`. `DbObject` v závislosti na strukturách tabulek sestaví a provede korektní SQL dotazy. Podstatná je zde implementace podpory `AUTO_INCREMENT` sloupců, kdy je do odkazujících tabulek doplněna správná hodnota v závislosti na `lastInsertId` v odkazované tabulce.

Pro úpravu již existujících objektů je nejdříve třeba naplnit atributy pomocí metod `fromPrimary` (naplnění na základě hodnot primárních klíčů) nebo `fromArray` (přímé naplnění na základě poskytnutých řádků, vhodné pro úpravu množin objektů získaných komplikovanějšími selecty). Provedené změny lze poté přímo zapsat metodou `update`. `DbObject` poskytuje také možnost automatického zapsání změn při destrukci objektu v závislosti na „dirty bit“ proměnné. Tímto lze snížit množství generovaných dotazů, jsou-li atributy upravovány na více místech v kódu, neboť do databáze jsou promítnuty až konečné hodnoty u všech sloupců najednou a to pouze v případě potřeby.

Použití DbObjectu a jeho alternativu v Zend frameworku[1] ilustruje příklad 2.1. Úspora kódu i lepší čitelnost je zřejmá.

Příklad 2.1

```
/**** ZEND FRAMEWORK ****/
class T1 extends Zend_Db_Table_Abstract{
  protected $_name = "t1"; //cols: id (A_I), val1
  protected $_primary = "id";
}
class T2 extends Zend_Db_Table_Abstract{
  protected $_name = "t2"; //cols: id (FK to t1), val2
  protected $_primary = "id";
}
$table1 = new T1();
$id = $table1->insert(array("val1" => "1"));
$table2 = new T2();
$table2->insert(array("val2" => "2", "id" => $id));
                /**** DbObject ****/
class MyObject extends DbObject{
  static function tables(){return("t1", "t2");}
}
$object = new MyObject();
$object->setAttr("val1", "1")
  ->setAttr("val2", "2")
  ->insert();
```

3 ROZŠIŘUJÍCÍ MOŽNOSTI

Následující text rozebírá další utility, které postupně rozšířily základní koncept třídy ve snaze usnadnit často prováděné operace.

3.1 SOUVISEJÍCÍ OBJEKTY A SOUBORY

Přetížením metod `relatedFiles` a `relatedObjects` lze docílit automatického mazání souvisejících objektů a souborů ze serveru při volání metody `delete`. Vhodným použitím metod zajistíme korektní uvolnění alokovaných zdrojů a lze jimi simulovat též `ON DELETE CASCADE` klauzule, pakliže ji zvolený database engine nepodporuje[2].

[2]Např. při použití MyISAM pro fulltextové vyhledávání

3.2 PŘEKLADY

Pro vývoj multilinguálních aplikací jsou užitečné metody `getTrans`, `setTrans` a metoda `translations`, která určuje sloupce s obsahem závislým na jazyce. Pro fungování tohoto rozšíření musí databáze obsahovat tabulky `translations` a `translation_items` pro uložení hodnot.

3.3 FETCHOBJECTARRAY

Při modelování vztahů s kardinalitou 1:m často potřebujeme pracovat s odkazujícími objekty. Díky schopnosti PHP dynamicky vytvořit objekt pomocí názvu třídy a metodě `fetchObjectArray` lze tento úkol jednoduše vyřešit, jak demonstruje příklad 3.1 pro získání obsahu objednávky v hypotetickém eshopu.

Příklad 3.1
```
function orderItems(){
  $stmt = $this->dbConnection()->query("SELECT * FROM `order_items`
    WHERE `orderId` = ".$this->getOrderId());
  return $this->fetchObjectArray($stmt, "OrderItem");
}
```

3.4 ŠABLONA PRO ECLIPSE

Pro větší komfort při vývoji je výhodné vytvořit si metody pro nastavení/získání jednotlivých atributů objektu. Problém lze vyřešit pomocí magic metody `__call`[2], tento přístup by však omezil nápovědu poskytovanou code assistentem v IDE Eclipse. Z tohoto důvodu je pohodlnější vytvořit si šablonu pro páry těchto metod a v každé třídě dědící z `DbObject` je explicitně uvést. Vzor šablony ilustruje příklad 3.2. Dokumentovaný setter vrací ukazatel na původní objekt z důvodu řetězení.

Příklad 3.2
```
function get_${name}(){return $$this->getAttr("${name}");}
/**
* @return ${class_container}
*/
function set_${name}($$v){return $$this->setAttr("${name}", $$v);}
```

4 ZÁVĚR

Třída `DbObject` poskytuje efektivní a přehlednou sadu nástrojů pro manipulaci s databázovými objekty. Její jednoduchost, vyjadřovací účinnost a rychlost nasazení v součinnosti s využitím dynamických a reflektivních vlastností použitých technologií ji odlišují od většiny komerčně využívaných frameworků. Její použití bylo velkým přínosem v autorem vyvíjených aplikacích. Zatím bohužel nenašla širší uplatnění na trhu, autor však pevně věří že navržené principy pomohou zefektivnit propojení mezi databází a server-side skripty.

REFERENCE

[1] Zend Technologies Ltd.: *Zend_Db_Table*. (2013).
`http://framework.zend.com/manual/1.12/en/zend.db.table.html`

[2] The PHP Group: *PHP: Overloading*. (2013).
`http://www.php.net/manual/en/language.oop5.overloading.php`

[3] The PHP Group: *PHP: Late Static Bindings*. (2013).
`http://php.net/manual/en/language.oop5.late-static-bindings.php`