Май '04 № 3

PHP Inside

электронный журнал для веб-разработчиков

специальный выпуск специальный выпуск специальный выпуск

phpCONF 2004

phpclub

5 лет с вами!

Содержание

Обратная связь

Обзор конференции

Елена Тесля

Этот номер мы решили целиком посвятить второй международной конференции «Современные технологии эффективной разработки веб-приложений с использованием PHP», прошедшей 14 мая в Москве. Благодаря Клубу разработчиков PHP (http://www.phpclub.ru) и агентству “Третья Планета” (http://www.3planeta.ru) удалось организовать конференцию и собрать людей, занимающихся веб-технологиями. Материалы конференции и составляют содержание этого номера журнала.

Те, кто побывал на конференции, наверняка получили не только новую информацию с профессиональной точки зрения. Общение с давно известными по форуму http://phpclub.ru/talk личностями, приехавшими из разных стран и городов, также добавило положительных моментов. Как и общий дружеский настрой мероприятия.

Поначалу народ вел себя достаточно стеснительно, однако, все очень быстро познакомились, а общность интересов и актуальные темы создали благоприятную почву для общения.

Кирилл Шепитко [Yamert], г. Рига, был первым докладчиком на конференции, если не считать вступительного слова Александра Смирнова [PHPClub], собственно, организовавшего всю эту конференцию вместе с Дмитрием Коробицыным (за что им большое спасибо не только от меня).

Алексей Борзов [Sad Spirit], выступил даже не с одним, а с двумя докладами: о PEAR и PostgreSQL. После своего живого и яркого выступления он также блестяще отвечал на вопросы, с завидным постоянством отвечая: «Смотрите примеры» и «Принимаю патчи,» - в ответ на вопросы о багах или новых особенностях.

В противовес этому бодрому докладчику выступил Михаил Курмаев [Demiurg], сонный после проведенной в подготовке раздаточных материалов ночи. Поэтому его доклад о шаблонах Smarty был похож на колыбельную – весь зал, в том числе и докладчик, засыпали. Отсюда вынесли очень важный вывод: докладчики не должны участвовать в подготовке конференции.

Виктория Резниченко [Апельсин], г. Киев, начав свой доклад с того, что о недостатках MySQL все уже слышали, все же поведала о достоинствах и нововведениях в MySQL.

Алексей Рыбак [fisher] затронул очень интересную для всех тему тему об объектно-реляционных отображениях, и вот тут-то все очень пожалели, что на доклад выделено определенное количество времени – времени явно не хватило.

Денис Жолудов [DAN], хоть и опоздал на конференцию, тем не менее, сразу влился в обсуждение текущей темы и хорошо раскрыл свою тему об XML-технологиях.

Эта конференция собрала самых активных и заинтересованных людей.

Перерывы на кофе и чай с печеньем и бутербродами для многих оказались приятным сюрпризом, как и полноценный обед в кафе, раздаточные материалы и бейджики на разноцветных веревочках, которые раздавались каждому участнику конференции.

Когда настала пора подписывать сертификаты, оказалось, что программисты в основе своей не обладают красивым почерком, ибо пользуются в основном принтером. Поэтому вопрос «У кого красивый почерк?» вызвал только взрыв хохота.

Теплота атмосферы не нарушалась даже в спорах, которые возникали в процессе обсуждения интересующих тем – все мнения выслушивались, а острые моменты сглаживались шутками.

Эта конференция собрала самых активных и заинтересованных людей. К сожалению, не все желающие смогли попасть на нее. Но ведь в Москве конференция проводилась впервые, так что это только начало – теперь мы знаем, на что способны, знаем, куда стремиться и что стоит исправлять.

Еще раз хотелось бы поблагодарить всех участников конференции за присутствие, а особенно докладчиков – за выступления. И, конечно же, благодарности устроителям – Александру Смирнову (http://phpclub.ru) и спонсору – студии «Третья Планета» (http://www.3planeta.ru).

В фокусе

Smarty - компилируемые шаблоны

Автор доклада:

Михаил Курмаев

Как известно, PHP изначально был языком, встраиваемым в HTML. Таким образом, скрипты, написанные на PHP, содержали в себе смесь PHP-кода и HTML. Это было удобно, когда речь шла о небольших проектах, в которых интерпретатор занимался тем, что подставлял в страницу общие для всего сайта куски html и подставлял некоторые переменные (например, текущую дату или значение, пришедшее из формы). Со временем проекты стали расти. Практически во всех проектах стали использоваться базы данных, иногда с довольно сложной структурой. Используемые конструкции все усложнялись и усложнялись. Со временем скрипты превращались в месиво из управляющих конструкций PHP-скриптов и HTML. Смена дизайна в таких проектах была равносильна переписыванию всего проекта заново.

Как известно, над проектом обычно работают программист и дизайнер. На дизайнере лежит ответственность за HTML (в данном случае подразумевается технический дизайнер), а на программисте — за все остальное. В случае совмещения PHP-скриптов и кода HTML дизайнер должен, по крайней мере, уметь читать скрипты, чтобы не навредить им, а программист должен уметь легко отсеивать из получившихся файлов HTML.

Свет в конце туннеля

Решение вышеозначенной задачи нашлось в области Объектно Ориентированного Проектирования, а именно в схеме Model/View/Controller (MVC). MVC предлагает разделять логику модели (Model), логику представления (View) и бизнес-логику (Controller). В нашем случае, то есть в веб-проектах, в большинстве случаев мы уже имеем отделенную логику модели — базу данных. Как известно, СУБД имеют собственные методы поддержки целостности баз данных, что позволяет вынести из скриптов логику модели. Теперь остается решить, как отделить бизнес логику от логики представления. Выражение «логика представления» очень хорошо сочетается с выражением «markup language» (язык разметки). Остался третий компонент — controller, то есть то, что всем управляет. На эту роль очень хорошо подходит PHP.

Реализация MVC в WEB

Итак, мы имеем три компонента, давайте посмотрим, как все это должно работать. При поступлении запроса начинает работать PHP-скрипт (Controller). Исходя из полученных параметров, он запрашивает данные у базы данных (Model), затем выбирает шаблон (View), если он не известен заранее, который и отображает эти данные. Одна из разновидностей этой схемы — это когда шаблон сам запрашивает у скрипта нужные ему данные, но об этом попозже.

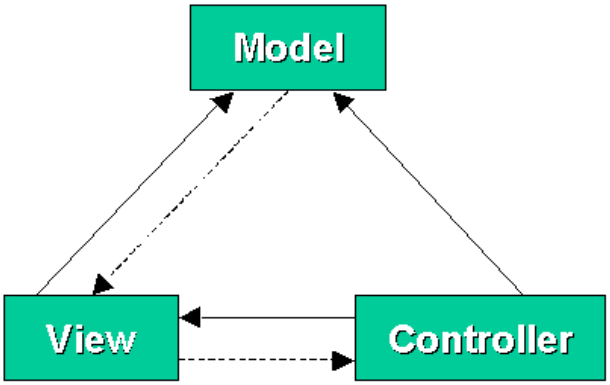


Рис. 1. Классическая схема MVC

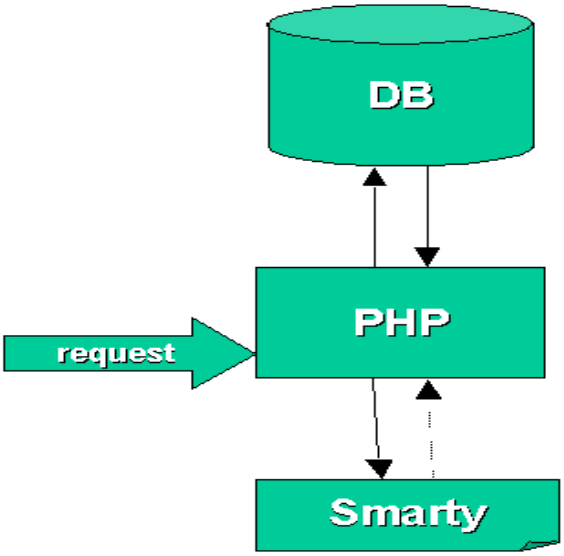


Рис. 2. Схема веб-приложения

Подобные схемы применяются довольно давно, и существует множество разновидностей таких решений (шаблонных движков). Все они различаются скоростью, возможностями и способами применения. Несмотря на это, многие разработчики стараются написать свои шаблонные движки вместо того, чтобы использовать и развивать уже готовые продукты.

Smarty VS Others или разделяй и властвуй

Далее будет рассмотрен один из самых мощных шаблонных движков — Smarty. В отличие от других движков, Smarty действительно позволяет разделить бизнес-логику и логику представления, а не просто вынести HTML за пределы скриптов. Для сравнения рассмотрим характерные примеры использования Smarty и Sigma из библиотеки PEAR.

Пример из документации по Sigma.

Скрипт:

```
<?php
// подключаем библиотеку

$data = array (
    array("Stig", "Bakken"),
    array("Martin", "Jansen"),
    array("Alexander", "Merz")
);

$tpl =& new HTML_Template_Sigma('.');
$tpl->loadTemplateFile('table.html');
foreach ($data as $name) {
    // assign data
    $tpl->setVariable(array(
        'first_name'  => $name[0],
        'last_name' => $name[1]
    ));
    // process the block
    $tpl->parse('table_row');
}
// print out the output
$tpl->show();
?>
```

Шаблон:

```
<html>
<body>
<table cellpadding="2" cellspacing="0" border="1">
<!-- BEGIN table_row -->
    <tr>
        <td>{first_name}</td>
        <td>{last_name}</td>
    </tr>
<!-- END table_row -->
</table>
</body>
</html>
```

Пример из документации по Smarty.

Скрипт:

```
<?php
// подключаем библиотеку
$smarty = new Smarty;
$smarty->assign("contacts",
array(
        array("phone" => "1", "fax" => "2", "cell" => "3"),
        array("phone" => "555-4444", "fax" => "555-3333", "cell" => "760-
1234")
));

$smarty->display("template.tpl");

?>
```

Шаблон:

```
{foreach name=outer item=contact from=$contacts}
  {foreach key=key item=item from=$contact}
    {$key}: {$item}<br>
  {/foreach}
{/foreach}
```

Несмотря на то, что пример со Smarty получился компактнее, такая цель не ставилась. Как видно, в случае использования Smarty скрипт передает данные Smarty в виде массива и указывает, какой шаблон использовать. В случае же с Sigma скрипт явно указывает, как именно выводить тот или иной блок, а само HTML-представление находится в шаблоне. Таким образом, при изменении логики представления для Smarty требуется только изменение шаблона, а в случае Sigma ─ также и скриптов.

Для большей ясности приведем пример. Предположим, что производится вывод ленты новостей. Новости берутся из базы из одной таблицы с сортировкой по дате. Первоначально все новости выводятся одинаково. Затем, в процессе эксплуатации, становится ясно, что для того, чтобы ленту было удобно читать, не теряя при этом информативности, можно первые три новости оставить в том же виде, а остальные урезать и выводить только заголовки. Налицо смена логики представления. Smarty с этим справляется довольно легко, необходимо просто немного поменять шаблон, и нет необходимости задумываться о том, что нужно менять в скриптах.

Однако, следует хорошо разделять логику представления и бизнес-логику. Например, выделение «сегодняшних» новостей ─ это, скорее, бизнес-логика, и следует помечать в скриптах такие новости как «сегодняшние». Smarty, конечно, имеет возможность сравнивать даты. Но, как показывает практика, термин «сегодняшняя новость» со временем может применяться, например, к новостям, добавленным сегодня, или к новостям за последние сутки. Очевидно, что логика представления при этом никак не меняется, и нет смысла при таких изменениях постоянно изменять шаблоны.

Синтаксис, функции и модификаторы Smarty

Рассмотрим возможности Smarty. Поскольку у Smarty есть хорошо написанная документация, то нет смысла рассказывать здесь о синтаксисе и перечислять все параметры функций.

Smarty не зря называется компилирующим шаблонным движком (compiling PHP template engine). Он не просто разбирает и собирает шаблоны, а, в первую очередь, компилирует их в PHP-скрипт.

Когда шаблонизатору нужно отобразить какой-либо шаблон, он сравнивает время изменения шаблона и его откомпилированного представления. И, в зависимости от результата, либо перекомпилируется шаблон, либо просто исполняется готовый скрипт. Ясно, что это увеличивает общую производительность системы.

Компиляция шаблонов происходит крайне редко в рабочем режиме, и её время не влияет на общую производительность. Конечно, то, что шаблоны компилируются в PHP-скрипты, не значит, что по откомпилированным шаблонам можно произвести отладку или поменять их, делать это категорически не рекомендуется.

Функции цикла

Итак, как было сказано выше, Smarty получает от скрипта данные, которые он должен отобразить, и шаблон, который он должен использовать при этом, поэтому сам шаблонизатор должен иметь довольно мощные средства управления. Как ни странно, он их имеет. Например, в Smarty существует целых две функции цикла, естественно, каждая со своими особенностями. Основной функцией цикла является `section`. Вот пример её использования:

```
{section name=customer loop=$custid}
    id: {$custid[customer]}<br>
{/section}
```

Пример результата работы такого шаблона:

```
id: 1000<br>
id: 1001<br>
id: 1002<br>
```

Сразу скажу, что в Smarty все функции имеют вид:

```
{название_функции параметр1=значение1 параметр2=значение2 ...}
   тело_функции
{/название_функции }
```

или просто

```
{название_функции параметр1=значение1 параметр2=значение2 ...}
```

Другая функция цикла `foreach` проще, она используется обычно для прохода по ассоциативному массиву. Обе функции имеют небольшое удобное дополнение: sectionelse и foreachelse, — которые работают, когда не было совершено ни одной итерации цикла.

Например:

```
{section name=customer loop=$custid}
    id: {$custid[customer]}<br>
{sectionelse}
    there are no values in $custid.
{/section}
```

Кроме этого, имеются другие возможности, такие, как шаг итерации, указания начала и конца цикла и т.д.

Условные функции

Также имеется условный оператор (`if`). Ни синтаксисом, ни поведением он практически не отличается от аналогичного оператора в php.

Просто рассмотрим красноречивый пример:

```
{if $name=="Fred" || $name=="Wilma"}
    ...
{* 0=even, 1=even, 2=even, 3=odd, 4=odd, 5=odd, etc. *}
{elseif $var is even by 3}
    ...
{else}
    ...
{/if}
```

Функция присваивания

Функция присваивания выглядит несколько сложнее, чем обычно:

```
{assign var="name" value="Bob"}
```

С одной стороны это несколько неудобно, с другой — в Smarty эта функция используется нечасто, в отличие от языков программирования.

Пользовательские функции

Но на встроенных в Smarty операторах его функциональность не заканчивается. Smarty позволяет добавлять свои операторы, причем сделать это совсем не сложно. Достаточно добавить в нужную директорию файл с именем function.название_функции.php и определением php-функции smarty_function_название_функции ($params, &$smarty), и теперь можно использовать вашу функцию в шаблонах. Параметры, переданные из шаблона, придут в массиве $params. Вот пример из документации:

```
function smarty_function_eightball($params, &$smarty)
{
    $answers = array('Yes',
                     'No',
                     'No way',
                     'Outlook not so good',
                     'Ask again soon',
                     'Maybe in your reality');

    $result = array_rand($answers);
    return $answers[$result];
}
```

Модификаторы встроенные и пользовательские

Модификаторы в Smarty являются удобным средством, позволяющим привести данные в нужный вид. Например, для того, чтобы вывести строку, в которой необходимо экранировать весь код html, достаточно написать в шаблоне:

```
{$str|escape:"html"}
```

Модификаторы могут быть использованы последовательно:

```
{$str|escape:"html"|upper}
```

Также, как и операторы, модификаторы могут быть добавлены пользователем. Делается это тоже просто. Опять нужно создать файл modifier.название_модификатора.php с определением PHP-функции smarty_modifier_ название_модификатора ($string). Пример:

```
function smarty_modifier_capitalize($string)
{
    return ucwords($string);
}
```

В случае, если указанный модификатор не найден в стандартных, и его нет среди пользовательских модификаторов, то вызывается PHP-функция с именем модификатора. Например, можно использовать стандартную функцию `str_repeat`:

```
{$str|str_repeat:3}
```

Конфигурационные файлы

Smarty предоставляет удобное средство в виде конфигурационных файлов для вынесения различных параметров в отдельные файлы. Такими данными могут быть, например, цвета или зависимые от языка текстовые сообщения. Конфигурационные файлы находятся в отдельной папке и имеют вид:

```
# global variables
pageTitle = "Main Menu"
bodyBgColor = #000000
tableBgColor = #000000
rowBgColor = #00ff00

[Customer]
pageTitle = "Customer Info"

[Login]
pageTitle = "Login"
focus = "username"
Intro = """This is a value that spans more
           than one line. you must enclose
           it in triple quotes."""
```

Конфигурационные файлы могут загружаться в шаблон функцией config_load. Как видно, в файлах конфигурации существуют разделы, которые дают больше возможностей настройки. Например, можно разделить переменные по языковому признаку для облегчения переключения между языками.

Для получения переменной из конфигурационного файла достаточно в шаблоне написать {#имя_переменной#} или {$smarty.config. имя_переменной}.

Отладка

К сожалению, трудно сказать, что Smarty предоставляет мощные средства для отладки шаблонов. Например, при наличии синтаксической ошибки в шаблоне, шаблонизатор молча откомпилирует шаблон, и ошибку выдаст уже сам PHP, причем со ссылкой на откомпилированное представление, из которого, как было сказано выше, ничего хорошего узнать не удастся. Спасает в этой ситуации то, что обычно шаблоны имеют довольно простой синтаксис, и такие ошибки находятся визуально.

Ошибки, вроде пропущенных параметров в функциях, Smarty локализует сам и выдает сообщение (например: Fatal error: Smarty: [in index.tpl line 28]: syntax error: missing section name in / path/to/smarty/Smarty.class.php on line 1041).

Тем не менее, Smarty отлично использует схему MVC для отображения отладочной информации. Как известно, MVC не только разделяет приложение на три компонента, отвечающих отдельным задачам, но и позволяет иметь несколько View-компонентов, причем, каждый такой компонент ничего не знает о других.

Для отображения данных, пришедших в шаблон, Smarty добавляет еще один View-компонент (при включенном режиме отладки). На самом деле, это просто еще один шаблон, который выводит в popup-окне нужную информацию. Этот шаблон можно изменять и настраивать под свои нужды.

Кеширование

Кеширование в Smarty является, вероятно, одной из самых сильных сторон. Кеширование происходит на уровне шаблонов, то есть результат работы шаблона Smarty сохраняется в виде файла, и если в следующий раз будет принято решение отобразить шаблон из кеша, то происходит простое чтение файла.

Возникает вопрос: как долго должен хранится кеш? Smarty позволяет задавать время хранения кеша как общее для всех шаблонов, так и своё для каждого шаблона. Кроме того, кеш может быть удален вручную.

Но само название «шаблон» говорит о том, что он только описывает представление данных и может использоваться для отображения различных данных. В таком случае одного экземпляра кеша для одного шаблона оказывается мало.

Например, мы отображаем статьи и хотим, чтобы они кешировались, а не брались постоянно из базы данных.

Smarty позволяет при кешировании и отображении использовать так называемый cache_id, который является идентификатором для каждого экземпляра кэша:

```
$smarty->caching = true;

$my_cache_id = $_GET['article_id'];
if(!$smarty->is_cached('index.tpl',$my_cache_id)) {
    // No cache available, do variable assignments here.
    $contents = get_database_contents();
    $smarty->assign($contents);
}
$smarty->display('index.tpl',$my_cache_id);
```

Cache_id используется для именования файлов кеша, поэтому не все символы можно использовать в его значении. Но есть один символ, который имеет специальное значение. Кеш можно разбивать по группам, для этого cache_id должен иметь вид «`group|subgroup|id`».

Оказывается, это очень удобно, когда надо удалить весь кеш, входящий в некое множество. Например, в предыдущем примере статьи можно было разбить по разделам и при необходимости удалять кеш всего раздела.

Итак, кеширование в Smarty осуществляется на уровне шаблонов. Но что делать, если необходимо закешировать некую страницу, представленную одним шаблоном, но на этой странице есть некие динамические элементы (например, баннеры)? Решение есть.

Оказывается, Smarty не кеширует результат работы функции insert. Функция insert вставляет в шаблон результат работы заданной PHP-функции. Таким образом, осуществляется обратная связь от шаблона к управляющему скрипту. Это именно тот способ, когда компонента View запрашивает данные у компоненты Controller.

Заключение

Как стало ясно из вышесказанного, Smarty действительно разделяет бизнес-логику и логику приложения в WEB-проектах. Он не только отделяет HTML от PHP-скриптов но и имеет мощные синтаксические средства, сравнимые с языками программирования. Кроме того, кеширование позволяет сильно оптимизировать проект в узких местах. Но у Smarty есть один большой минус — сложно рассказать обо всех его возможностях на нескольких страницах текста, еще сложнее понять, почему он называется «Smarty» не начав его использовать. Больше информации можно получить на сайте http://smarty.php.net/

Эффективное использование MySQL

Нововведения в версии 4.1

Автор доклада:

Виктория Резниченко

На данный момент ветка 4.1 находится на стадии активной разработки. В этой ветке появились такие функциональности как:

- **Кодировки**. В версии 4.1. появилась возможность задания кодировок на уровне базы данных, таблиц, столбцов, а также поддержка Unicode (utf8 и ucs2). Все это позволит более гибко проектировать многоязычные приложения;
- **GIS**. Для таблиц типа MyISAM была реализована поддержка геометрических типов данных. Они базируются на модели OpenGIS. По этой модели каждый геометрический объект обладает следующими свойствами:
- Он ассоциируется с SRS (Spatial Reference System), описывающей координатное пространство, в котором определен объект;
- Он принадлежит к какомунибудь геометрическому классу;

```
CREATE TABLE t1(
l1 LINESTRING
);

INSERT INTO t1 VALUES
(LineStringFromText('LINESTRING(0 0, 1 1, 2 2)'));
```

- **Подготовленные выражения (prepared statements)**. Использование подготовленных выражений MySQL может привести к повышению производительности следующими способами:
- При изменении параметров нет необходимости делать повторный парсинг выражения;
- Меньше данных передается между сервером и клиентом;
- В версии 4.1 был изменен **механизм хеширования паролей**. Новый механизм хеширования повышает безопасность, однако у вас могут возникнуть проблемы с совместимостью. Приложения, использующие старые библиотеки (версии ниже чем 4.1), при попытке соединения с сервером MySQL могут получить ошибку:

```
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

Чтобы обойти это проблему, вы можете:

1. Использовать новую клиентскую библиотеку;
2. Не обновлять таблицы привилегий и запустить MySQL с опцией -old-passwords.

- Среди новых функциональностей также хотелось бы выделить **агрегатную функцию GROUP_CONCAT()**, которая для каждой группы возвращает строку из сконкатенированных (сцепленных) значений;

- Стоит обратить внимание на модификатор `WITH ROLLUP`, который используется вместе с оператором `GROUP BY`. `ROLLUP` позволяет осуществлять многоуровневый анализ в одном запросе и предоставляет суммарную информацию. Например:

```
mysql> SELECT country, year, SUM(profit) FROM t1 GROUP BY country, year
WITH ROLLUP;
+----------+--------+----------------+
|  country |  year  |   SUM(profit)  |
+----------+--------+----------------+
|   Albania |   2001 |           300 |
|   Albania |   2002 |           100 |
|   Albania |   2003 |           600 |
|   Albania |   NULL |          1000 |
| Denmark   |   2001 |           800 |
| Denmark   |   2002 |          1800 |
| Denmark   |   2003 |          1600 |
| Denmark   |   NULL |          4200 |
|   Greece  |   2001 |          1500 |
|   Greece  |   2002 |          1100 |
|   Greece  |   2003 |           700 |
|   Greece  |   NULL |          3300 |
|  NULL     |   NULL |          8500 |
+----------+--------+----------------+
13 rows in set (0.00 sec)
```

Еще одним интересным нововведением стало появление `ON DUPLICATE KEY UPDATE` в выражении `INSERT`.

`INSERT .. ON DUPLICATE KEY UPDATE` при дублировании уникального или первичного ключа позволяет обновлять значения остальных полей в строке.

Для типа `TIMESTAMP` добавлены атрибуты `DEFAULT CURRENT_TIMESTAMP` и `ON UPDATE CURRENT_TIMESTAMP`.

При помощи атрибута `DEFAULT CURRENT_TIMESTAMP` для первого столбца `TIMESTAMP` указывается, что при `INSERT` будет устанавливаться текущая дата и время, если при этом задать атрибут `ON UPDATE CURRENT_TIMESTAMP`, то текущая дата и время будут устанавливаться также при выполнении `UPDATE`.

Выражение `CREATE TABLE` при использовании этих атрибутов будут выглядеть следующим образом:

```
CREATE TABLE t1(
dt timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
id int
);

CREATE TABLE t2(
dt timestamp DEFAULT CURRENT_TIMESTAMP,
id int
);
```

Нововведения в версии 5.0

В ветке 5.0 основным нововведением стало появление хранимых процедур и функций. Хранимые процедуры — это такой набор SQL-выражений, который хранится на сервере. В MySQL синтаксис хранимых процедур реализован согласно стандарту SQL:2003. Этот же синтаксис используется DB2.

Ниже в качестве примера приведены хранимая процедура и функция:

```
CREATE PROCEDURE proc1 (
IN param1 INT,
OUT param2 INT)
BEGIN
SELECT SQRT(param1) INTO param2;
END


CREATE FUNCTION func1 (name CHAR(20)) RETURNS CHAR(50)
RETURN CONCAT(name,' - kanistra!');
```

Внутри хранимых процедур и функций реализованы курсоры. На данный момент курсоры могут быть только `ASENSITIVE`, `READ ONLY` и не-`SCROLL`.

В ветке 5.0 будет продолжена реализация представлений (VIEW). Неименованные представления (derived tables), они же вложенные запросы в части FROM, появились еще в ветке 4.1.

В ветке 5.0 будет продолжена реализация VIEW, но в полном объеме они будут реализованы только в районе версии 5.1.

Сертификация

Существует две программы сертификации: Базовая сертификация MySQL (MySQL Core Certification) и Профессиональная сертификация MySQL (MySQL Professional Certification).

Базовая сертификация представляет собой проверку знаний для опытных пользователей. Здесь будут заданы вопросы, касающиеся диалекта SQL, используемого в MySQL, об утилитах используемых для администрирования, общие вопросы касающиеся компании MySQL AB, политика лицензирования и т.д.

Тест состоит из 70 вопросов, и у вас будет 1,5 часа для того, чтобы на них ответить.

Пример теста вы можете посмотреть по этой ссылке:

http://www.mysql.com/training/certification/selftest/core/index.php

Вопросы, на которые вам предстоит отвечать, проходя профессиональную сертификацию, касаются больше администрирования, установки, настройки, оптимизации сервера MySQL. Также включают вопросы, касающиеся использования и обслуживания таблиц InnoDB, вопросы безопасности и репликации.

Этот тест также состоит из 70 вопросов, на которые вам предстоит ответить в течении 1,5 часа.

На данный момент стоимость каждого из экзаменов составляет 135 евро.

Политика лицензирования

MySQL использует модель двойного лицензирования. По этой модели пользователь может использовать MySQL под лицензией GNU GPL или под коммерческой лицензией.

Под лицензией GPL MySQL доступен для свободного использования. Пользователи могут изменять, интегрировать и распространять его. Однако, пользователи должны придерживаться правил, которые оговаривают, что если приложение базируется на MySQL, то весь исходный код должен быть открыт и доступен.

Для тех, кто не хочет придерживаться лицензии GPL, MySQL предлагает коммерческую лицензию.

Таким образом вы можете использовать MySQL свободно под лицензией GPL либо под коммерческой лицензией, когда использование GPL не допустимо.

Вы можете свободно распространять производную работу, которая целиком оформлена из работ, распространяемых под одной или более лицензиями, перечисленными ниже, без учета срока лицензии так долго, пока вы:

1. Пока вы подчиняетесь GNU GPL во всех программах, относящихся к производной работе, за исключением опознаваемой части работы, которая не является производной и которая может считаться независимой и отдельной частью работы;
2. Вы распространяете все части производной работы, не происходящие от Программы и считающиеся независимыми и отдельными от остальных работ, которые распространяются под одной из перечисленных ниже лицензий;
3. Производная работа не включает в себя и не агрегирует ни одну из частей сервера MySQL или любую модификацию, перевод или любые другие видоизменения;
4. Если ни одно из вышеперечисленных условий удовлетворяется, то Программа может только копироваться, распространятся или использоваться только согласно срокам и условиям GPL или другим лицензиям от MySQL AB.

При распространении под следующими лицензиями в продукты могут включаться клиентские библиотеки MySQL:

Название лицензии	Версия/Дата
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0
Artistic license	Perl 5.8.0
BSD license	«July 22 1999»

Название лицензии	Версия/Дата
Common Public License	1.0
GNU General Public License (GPL)	2.0
GNU Library или "Lesser" General Public License (LGPL)	2.1
Jabber Open Source License	1.0
MIT license	-
Mozilla Public License (MPL)	1.0/1.1
PHP License	3.0
Python license (CNRI Python License)	-
Python Software Foundation License	2.1.1
Sleepycat License	«1999»
W3C License	«2001»
Zlib/libpng License	-
Zope Public License	2.0

Если упростить, то существует несколько правил лицензирования:

- Если ваше программное обеспечение распространяется под одной из совместимых с GPL лицензией, которая определена Free Software Foundation или утверждена OSI, то используйте продукт под версией GPL;
- Если вы распространяете собственное приложение любым способом и при этом вы не лицензировали и не распространяете исходный под лицензией GPL, то вам необходимо приобрести коммерческую лицензию MySQL;
- Если вы не уверены, необходимо ли вам приобрести лицензию, обращайтесь licensing@mysql.com.

Те пользователи, которые приобрели коммерческие лицензии, также получают коммерческую тех. поддержку.

В любом случае вы можете взять для тестирования некоммерческую версию MySQL до того, как вы переведете свой продукт под коммерческую лицензию.

ООП в PHP: применение, эволюция

Автор доклада:

Кирилл Шепитко

Почему и когда следует использовать ОО-подход в программировании на PHP?

На PHP, как языке веб-программирования, можно написать достаточно разнообразные приложения — от персональной странички до системы электронных платежей. Несомненно, каждая отдельно взятая система требует особенного подхода в разработке и реализации; однако можно сказать с уверенностью, что те элементарные средства, которые вполне удовлетворительны для реализации простых систем, плохо подходят для реализации сложных. Именно поэтому я считаю, что в разработке и реализации крупных веб-систем с использованием PHP нужно использовать объектно-ориентированный подход (ООП), а не функциональный.

Для тех, кто не использует ООП

ООП, хотя бы с практической точки зрения, имеет следующие преимущества:

1. Инкапсуляция данных. При функциональном стиле программирования, когда система имеет значительные размеры, приходится к функциям, имеющим одинаковый смысл (и как следствие — название), а также переменным прибавлять какой-то уникальный префикс — иными словами, всячески избегать нежелательных коллизий имён. Когда систему разрабатывает не один человек, а несколько, это сделать не так уж легко. Когда переменные (атрибуты) и функции (методы) находятся внутри класса, этого не происходит к тому же класс позволяет сгруппировать логику, относящуюся к одной и той же сфере (например, работа с LDAP), собрать её в одном месте и изолировать от прочего;
2. Отсутствие необходимости в глобальных переменных. К примеру, если есть функции для работы с базой даных, им нужно передавать переменную соединения с базой, или делать её глобальной. Это неудобно. В классе, который делает то же самое, эту переменную можно сделать атрибутом класса, к которой будут обращаться методы;
3. Наследование классов. Потомок наследует логику предка, и дополняет своей, или переопределяет её. К примеру, базовый класс Logger реализует общую логику по ведению логов (формат и т.п.). Далее два класса: FileLogger и DBLogger — наследуют от этого класса, и всё, что им надо — это реализовать свою конкретную функциональность для записи данных в соотв. хранилище данных;

Наиболее, на мой взгляд, существенные минусы ООП в PHP 4:

1. Все методы и переменные — public static. Модификаторы private и protected (есть в PHP 5) особенно важны, когда ваш класс использует разработчик, незнакомый с логикой, которую вы заложили в этот класс;

2. Нет перегрузки функций. Аргументы по умолчанию иногда использовать неудобно (нужна лишняя логика для их анализа);
3. Неудобно, что в методах класса обязательно должно быть обращение `$this->`... перед инициализацией.

Пример использования ООП и паттерна MVC: создание электронного магазина

При проектировании нашей системы мы будем использовать паттерн программирования MVC — (Model-View-Controller, Модель-Вид-Контроллер). Этот паттерн предусматривает разбиение программы на 3 части, что позволяет нам максимально просто вносить изменения в неё:

1. **Модель**. В модели содержатся данные конкретного модуля, а также методы для работы с этими данными;
2. **Вид** создаёт интерфейс, основываясь на текущем состоянии модели;
3. **Контроллер** обрабатывает запросы пользователя к системе, и интерпретирует их в действия, которые должна совершить модель. В зависимости от результатов исполнения вызванных методов модели контроллер отвечает пользователю, выбирая для него соответствующий вид.

От распределения системы на эти 3 части мы имеем следующие выгоды:

1. Повторное использование компонент модели;
2. Более простая поддержка разнородных клиентов (для браузеров - HTML, для моб.телефонов — WML, и т.д.) — нужно просто выбрать соотв. вид; бизнес-логика работает та же самая;
3. Легко вносить изменения, модифицировать структуру.

Итак, теперь перейдём к описанию нашей будущей системы.

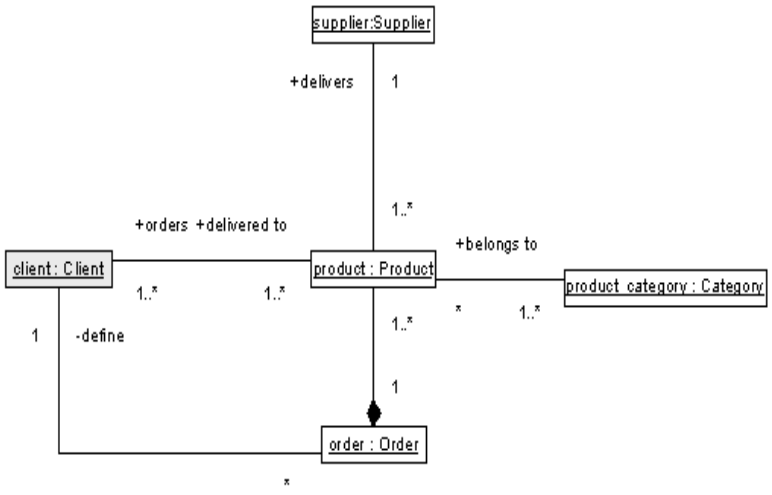


Рис. 1. Сущности системы.

На рисунке 1 представлена диаграмма классов, на которой показаны сущности системы, и их взаимосвязи:

1. **client** — пользователь магазина (Василий Пупкин). Он заказывает товары product, помещая их в свою корзину покупателя (таким образом определяя заказ order), и затем получает их на указанный адрес;
2. **product** — товар (копыто венерианского скалозуба), предлагаемый магазином. Товар относится к одной или более категориям товаров (венерианские сувениры, рога&копыта). Товар поставляет поставщик Supplier;
3. **product_category** — категория товара;
4. **order** — заказ, который содержит информацию о сделке: список купленных товаров, адрес покупателя и др.;
5. **supplier** — поставщик товара, ответственный и за доставку товара клиенту.

Теперь перейдем к архитектуре системы (паттерну — MVC).

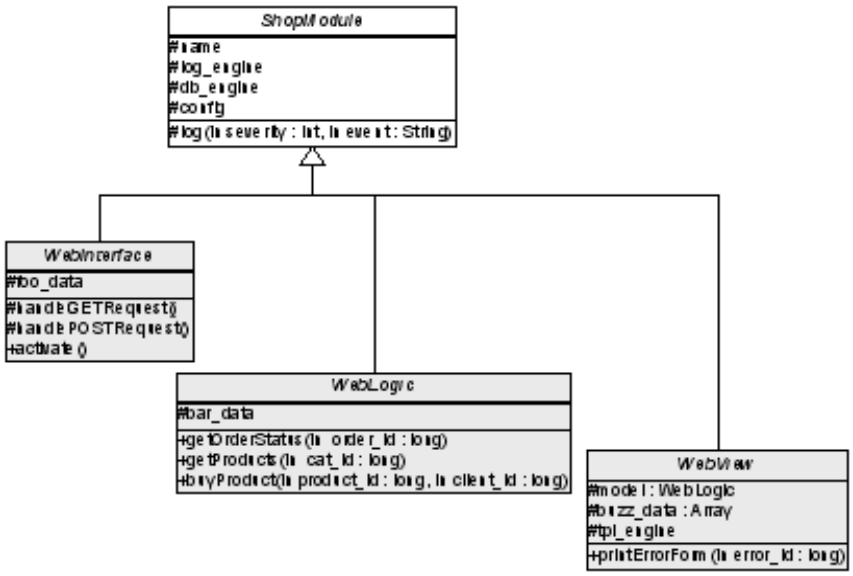


Рис.2 Основные классы системы.

На рисунке 2 представлена диаграмма классов, на которой показаны основные классы системы, реализующие MVC-архитектуру. Все представленные классы — абстрактные (т.е. функциональность доступна только в потомках).

Класс ShopModule — базовый для всех прочих (кроме класса Registry, о нём вскоре пойдёт речь) классов. У него есть protected-метод log(), с помощью которого в лог системы заносятся записи.

Конструктор класса присваивает значения атрибутам name (уникальный идентификатор класса), log_engine (движок для логгирования (к примеру, PEAR::Log)), db_engine (движок для работы с базой данных), config (ассоц.массив с конфигурацией системы).

Класс WebInterface — «контроллер». protected-методы handleGETRequest(), handlePOSTrequest() предназначены для обработки соотв.запросов (в этом классе — просто «заглушка», вроде редиректа на начальную страницу, или выдачи сообщения об ошибке - это удобство на случай, если данный HTTP-метод не будет обрабатываться в дочернем классе).

Единственный метод, который можно будет вызывать у объекта потомка этого класса (скажем, CartInterface — интерфейс модуля корзины покупателя) извне — activate(). В нём-то и анализируется, что должен выполнить контроллер, какой вид выбрать, и какую модель задействовать. Итак, в данном случае всё, что нужно сделать для начала работы модуля корзины покупателя — это инициализировать объект класса CartInterface, и вызвать его метод activate().

Класс WebLogic — «модель». В этом базовом классе разумно разместить логику, которая по умолчанию будет общей для всех моделей — к примеру, пусть в данном случае это будет получение списка продуктов категории(getProducts()), покупка продукта (buyProduct()) и получение статуса заказа (getOrderStatus()).

Класс WebView — «вид». Так как «вид» должен реагировать на изменения в «модели», ссылка на объект нужной модели находится в атрибуте model. Другой атрибут, tpl_engine — ссылка на объект шаблонного движка (например, PEAR::Sigma), который используется для конструирования вида. В этот же класс можно поместить общие методы для всех «видов» — например, здесь это метод printErrorForm(), который выдаёт на экран стандартную форму сообщения об ошибке. Можно также сделать подклассы «вида» WebView, каждый из которых создаёт интерфейс для отдельного вида браузера (браузер PC, моб.телефон, PDA). В данном случае «вид» заточен под работу именно со стандартным браузером PC.

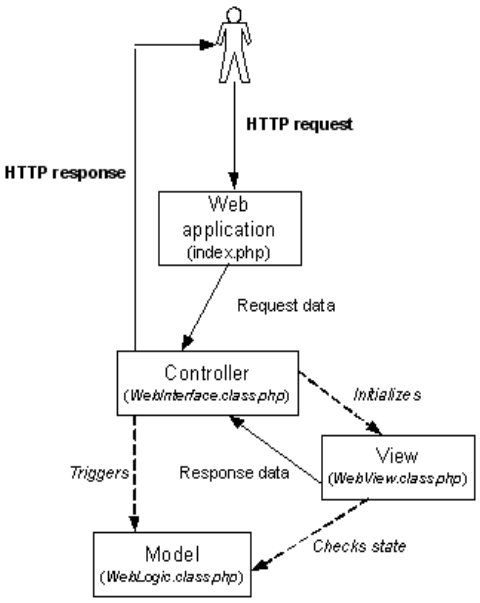


Рисунок 3. Использование MVC паттерна в нашей системе.

Отступая от шаблона MVC, мы создадим некоторое подобие регистра ресурсов — класс Registry(см.рис.4), который создаёт соединение с базой, инициализирует шаблонный движок и т.п.

Их он сохраняет в статических массивах, и затем возвращает лишь ссылки на элементы массива. Это централизует хранение базовых ресурсов системы. Все прочие классы (потомки ShopModule) лишь вызывают соотв.методы для получения ссылок на ресурсы.

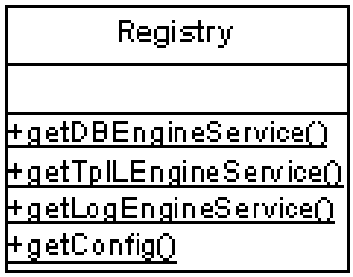


Рис. 4. Регистр ресурсов.

Методы этого класса могут быть статичны (модификатор static), и не должны переопределяться (модификатор final).

У нас есть костяк. Теперь разработаем остальные классы.

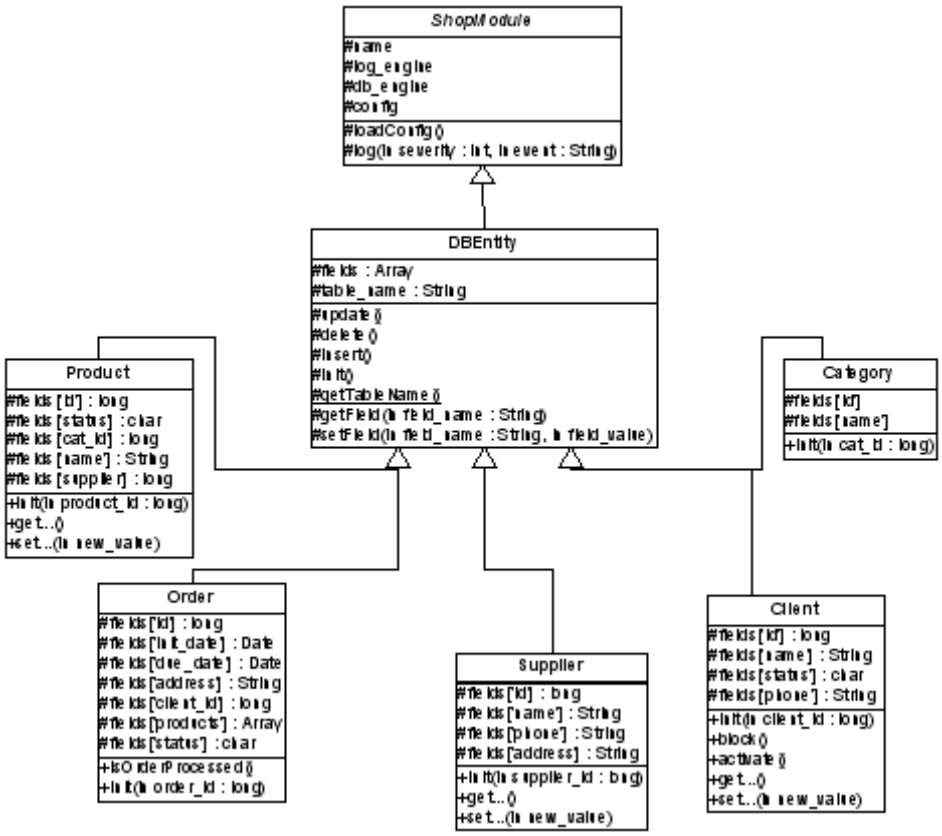


Рис.5. Классы для реализации бизнес-логики.

Для каждой сущности, представленной на рис.1, создадим отдельный класс.

Очевидно, что все они имеют связь с базой данных системы (её мы здесь рассматривать не будем). Поэтому создадим базовый класс для работы с таблицей, DBEntity — от этого класса будут происходить все прочие:

- метод **init()** делает поиск по указанному уникальному ключу, и в случае, если запись найдена, разбрасывает результаты по ассоциативному массиву для хранения ряда результатов;
- методы **update()**, **delete()**, **insert()** делают соотв. SQL-запросы;
- статический метод **getTableName()** возвращает имя таблицы;
- метод **getField()** возвращает значение соотв. поля ряда (из массива fields);
- метод **setField()** присваивает значение соотв. полю ряда.

Для дочерних классов остаётся только определить имя таблицы, имена доступных полей, переопределить метод init(), и создать get- и set-методы.

После этого можно определить методы, свойственные конкретному классу (к примеру, методы класса Client block() и activate ()).

Теперь рассмотрим нашу модель в действии.

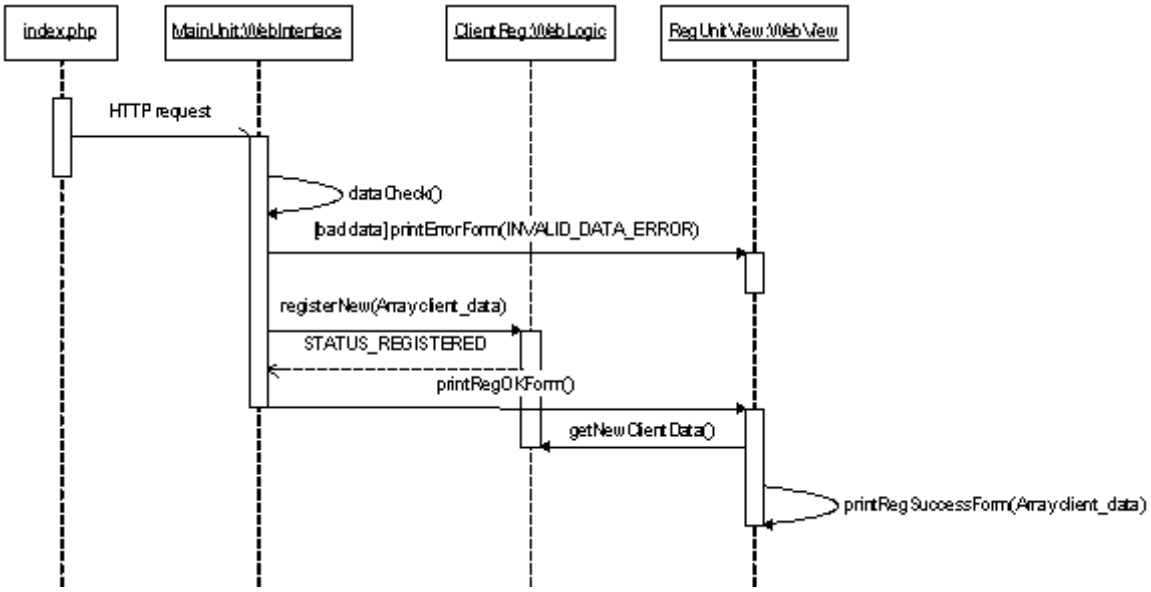


Рис.6. Регистрация клиента.

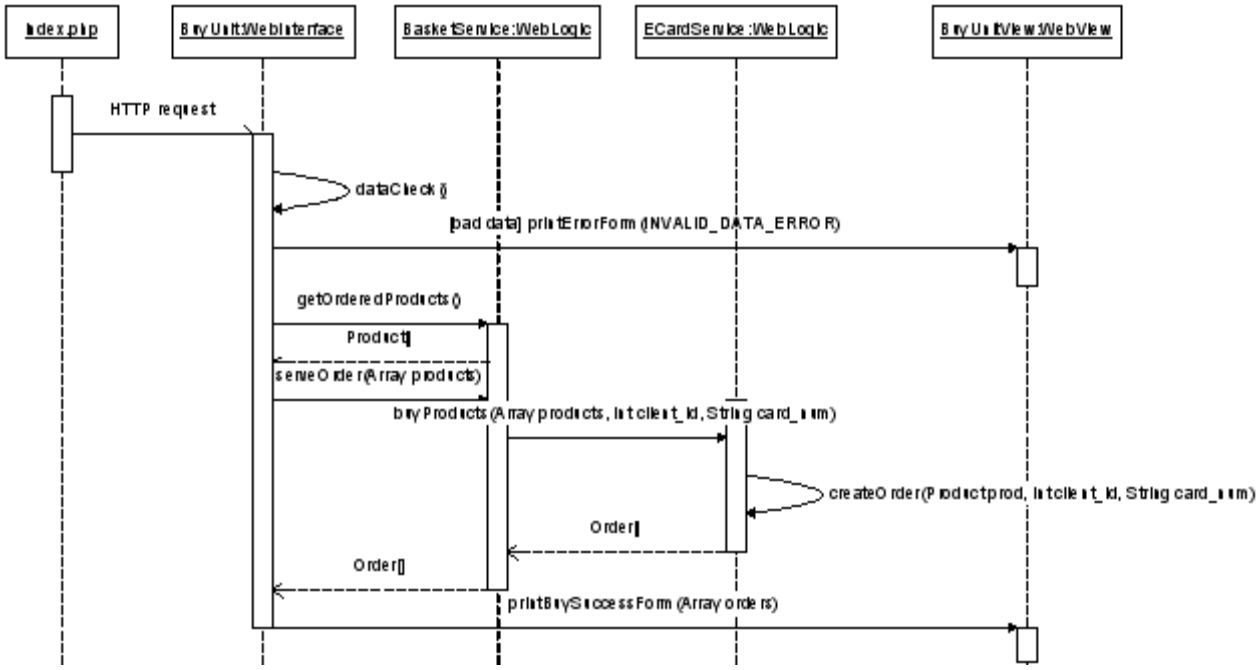


Рис.7. Покупка (2 «модели»).

Что дают нововведения в PHP5 для ОО-подхода в программировании на PHP?

На мой взгляд, в PHP5 в области ОО доведено до ума очень многое. Вот то, что бы выделил я из новинок (в порядке убывания важности):

1. Исключения (try/catch механизм);
2. модификаторы доступа private, protected, public;
3. abstract, final;
4. интерфейсы;
5. константы и static-свойства в классах (частная конфигурация);
6. возможность указать класс аргумента метода.

А вот и то, что не относится к ООП: возможность присваивания значений по умолчанию параметрам функций, переданных по ссылке.

Большинство изменений обеспечивают больше жёсткости и контроля в ООП PHP. При этом хорошо, что можно использовать старый подход, однако, если вы желаете больше контроля и стабильности в использовании ваших классов, разумно использовать средства, предоставляемые ZE2.

Минусы же таковы:

1. Несмотря на то, что появилась возможность указать класс аргумента метода, нет перегрузки (overloading) методов;
2. Конструкторы и деструкторы родительских классов не вызываются автоматически;
3. Несмотря на то, что и у методов появился модификатор static, то, что в методе используется $this, обнаруживается лишь при запуске самого метода, а не при чтении объявления класса. Таким образом, роль этого модификатора чисто декларативная;
4. Не все переменные передаются автоматом по ссылке, только объекты.

Однако, следует помнить и о некоторых нюансах, которые связаны с миграцией с PHP4 на PHP5:

1. Объект без атрибутов не является empty;
2. Если в классе нет конструктора, но есть метод `__construct()`, вызовется он;
3. Класс должен быть объявлен прежде, чем создан его объект.

Хотя, в общем-то, на мой взгляд, при правильном использовании ООП в PHP4 проблем с миграцией быть не должно.

Реализация примера в PHP4 и PHP5

При реализации результатов проектирования в PHP4 возникают следующие проблемы:

- Нельзя создать абстрактные классы:
- Невозможно определить уровень доступа к членам классов;
- Без `try/catch` код для обработки исключений принимает чудовищные объёмы.

В PHP5 вся описанная функциональность реализуется без проблем.

Эффективное использование XML-технологий в веб-проектах

Автор доклада:
Денис Жолудов

В последнее время разработчики веб-приложений проявляют все больший интерес к технологиям, основанным на языке разметки XML (eXtensible Markup Language). Этот интерес вызван прежде всего тем, что данные технологии существенно облегчают разработку и внедрение программ и приложений. Имея статус рекомендаций консорциума W3C, xml-технологии подпадают под определенные стандарты и тем самым представляют собой гибкий, но сильный инструмент для разработчика. Кроме того, стандарты обеспечивают кроссплатформенность при использовании xml-технологий. Эти и многие другие факторы побуждают разработчиков обращать все более пристальное внимание на xml-технологии.

Поддержка XML в PHP 4

Поддержка XML в PHP осуществляется начиная с 3-й версии языка PHP. Наиболее богатые возможности использования xml-технологий в php-скриптах на сегодняшний день представлены в стабильной, четвертой, версии PHP. Работа с xml-документами основывается на двух моделях представления xml-данных:

1. **SAX** (Simple API for XML) ─ событийная модель обработки документа;
2. **DOM** (Document Object Model) ─ объектная модель документа.

Поддержка XML–технологий в PHP 4

Библиотека	php-модуль	модель обработки данных	ресурсы	скорость
· expat	XML	SAX	+	–
· sablotron	XSLT	SAX	+	–
· libxml	DOMXML	DOM	–	+
· libxslt	DOMXML	DOM	–	+
· libexslt	DOMXML	DOM	–	–

Рис. 1. Поддержка XML в PHP 4.

SAX-модель реализована во встроенном php-модуле xml на основе библиотеки expat. Документ обрабатывается на основе событий: открытия и закрытия тегов.

DOM-модель реализована в подключаемом php-модуле domxml и использует интерфейсы библиотеки libxml. Документ представляется в виде древовидной структуры в памяти ЭВМ.

Приемущество обработки документа на основе SAX-модели состоит в том, что документ в процессе разбора (parsing) подгружается в память частями. Таким образом, оперативная память расходуется экономно. Напротив, DOM-модель требует загрузки в память всего документа, что порой бывает ресурсоемко. С другой стороны, разбор документа на основе SAX-модели происходит медленнее, чем при использовании объектной модели документа.

Таким образом, разработчик имеет выбор инструментов для работы с xml-документами в php-скриптах. Если документ по объему данных небольшой, представляется удобным использовать функции модуля domxml. Если же есть необходимость разбирать документы большого объема, рационально использовать встроенный php-модуль xml. Методы и возможности этих модулей неоднократно описывались в отечественных и зарубежных изданиях и интернет-ресурсах, поэтому останавливаться на них мы не будем.

Поддержка XSLT в PHP 4

XSLT (XSL Transformations) ─ преобразования xml-документа на основе расширяемых таблиц стилей (eXtensible Stylesheet Language).

В PHP 4 данные преобразования основаны на использовании API двух библиотек: sablotron (в виде функций модуля xslt) и libxslt (в виде функций модуля domxml). Упомянутые API отличаются друг от друга и используют различные методы для преобразования xml-документа.

К сожалению, так как API модулей domxml и xslt периодически меняются, и сами модули являются экспериментальными, их установка вместе с PHP «по-умолчанию» отсутствует. Более того, функции модуля domxml не соответствуют стандарту DOM консорциума W3C.

Разработчики PHP стараются по мере возможности исправлять эти недостатки, но, все же, данные модули никогда не будут переведены в разряд стабильных, и веб-разработчики будут использовать их «на свой страх и риск». Все эти негативные моменты привели энтузиастов, занимающихся поддержкой XML в PHP, к выработке подходов и написанию модулей в PHP 5 «с нуля».

Поддержка XML, XSLT и других XML-технологий в PHP 5

В PHP 5 поддержка XML основана на мощной по функциональности библиотеке libxml2. На основе интерфейсов этой библиотеки в пятой версии PHP появились следующие модули:

- XML;
- DOM;
- SimpleXML;
- XmlReader;

- XSL;
- SOAP.

Модуль xml представляет поддержку SAX модели. Интерфейс модуля остался таким же, как и в PHP 4. Единственное отличие заключается в том, что модуль использует API библиотеки libxml2, а не expat.

Модуль dom полностью поддерживает стандарт DOM консорциума W3C, включая имена методов и свойств объектов. Совместимость с модулем domxml в составе PHP 4 не гарантирована в силу различия их интерфейсов. Примечателен тот факт, что работа с модулем dom существенно упрощается после прочтения спецификации DOM.

Модуль simplexml является нововведением в обработке xml-документов с помощью PHP. Цель simplexml ─ предоставить разработчику простой интерфейс для доступа к элементам xml-документа.

Фактически, модуль воспринимает документ, как объект древовидной структуры с данными, как свойствами элементов этой структуры. Используя итераторы (такие как foreach()), имеется возможность получить значения всех элементов документа.

Для выборки определенных элементов документа в simplexml имеется возможность выполнять XPath-запросы. Также имеются возможности записи значений в атрибуты объекта-документа и преобразования элементов документа в объекты DOM. Данный модуль очень эффективен при разборе четко структурированных документов (каталоги, анкеты и т.д.). Этот модуль существенно облегчит работу по разбору документа, а программный код будет выглядеть элегантнее.

Xmlreader ─ модуль PHP из PECL, предоставляющий последовательный доступ документу. В первом приближении можно сказать, что это упрощенная SAX модель обработки документа. Модуль является экспериментальным.

Модуль xsl использует API библиотеки libxslt для преобразования xml-документа. Эта библиотека была выбрана в качестве основной и единственной потому, что она базируется на libxml2 и очень удачно вписывается в концепцию поддержки XML в PHP 5. Еще одним преимуществом библиотеки libxslt является возможность вызова расширенных xslt-функций, используя API библиотеки libexslt.

Поддержка других библиотек xsl-преобразования в пятой версии PHP отсутствует. PHP-разработчикам будет интересен тот факт, что теперь из xsl-шаблона можно вызывать php-функции как внешние xslt-функции. Хотя данной возможностью стоит пользоваться очень аккуратно.

Помимо основных возможностей для работы с xml, реализованы дополнительные возможности, такие как:

- Полная совместимость со стандартами консорциума W3C;
- Поддержка пространства имен во всех модулях;

- Поддержка XPath в модулях dom, simplexml, xslt;
- Поддержка XPointer, XIncludes в модуле dom;
- Валидация документов на основе схем (Schema), RelaxNG, DTD;
- Обмен данными между различными xml-модулями;
- Наследование от классов DOM;
- Враппер PHP Streams.

Рассмотрим некоторые из перечисленных возможностей подробнее.

Реализация xincludes отдаленно напоминает функцию `include ()` в PHP. Данная функция предоставляет разработчику дополнительные возможности для включения в основной документ сторонних данных. Как будет сказано дальше, вместе с переопределением потоков ввода-вывода в php, xincludes дают очень гибкие возможности для создания и обработки документа.

В PHP 5 появилась возможность переопределения потока ввода-вывода, путем написания собственного обработчика. Вот пример такого обработчика:

```
class VariableStream {
    var $position;
    var $varname;
    function stream_open($path, $mode, $options, &$opened_path) {
        $url = parse_url($path);
        $this->varname = $url["host"];
        $this->position = 0;
        return true;
    }
    function stream_read($count) {
        $ret = substr($GLOBALS[$this->varname], $this->position, $count);
        $this->position += strlen($ret);
        return $ret;
    }
    function stream_eof() {
        return $this->position >= strlen($GLOBALS[$this->varname]);
    }
    function url_stat() {
        return array();
    }
}
```

Теперь, переопределив поток ввода-вывода и направив его обработку приведенному выше классу, появляется возможность обрабатывать специальным образом инструкции xincludes по включению в основной документ сторонних данных (см. листинг 1 ниже).

Таким образом, регистрируя тот или иной враппер, достигается гибкость в получении нужных данных.

```
stream_wrapper_register("var", "VariableStream");
$GLOBALS["bar"] = "<foo><bar>hello world</bar></foo>";
$dom = new domdocument;
$dom->load('xinclude.xml');
$dom->xinclude();
print $dom->saveXML();

 --- xinclude.xml ---
<root xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="var://bar#xpointer(/foo/bar)"/>
</root>

 --- output --
<root xmlns:xi="http://www.w3.org/2001/XInclude">
    <bar xml:base="var://bar">hello world</bar>
</root>
```

Листинг 1.

Не менее интересное решение появилось с внедрением в PHP 5 поддержки libxml2 «по-умолчанию». Теперь появилась возможность наследовать от базовых классов, представленных в объектной модели документа DOM. Проиллюстрируем сказанное примером:

```
Class books extends domDocument {
    function __construct() {
        parent::__construct();
    }
    function addBook($title, $author) {
        $titleElement = $this->createElement("title");
        $titleElement->appendChild($this->createTextNode($title));
        $authorElement = $this->createElement("author");
        $authorElement->appendChild($this->createTextNode($author));
        $bookElement = $this->createElement("book");
        $bookElement->appendChild($titleElement);
        $bookElement->appendChild($authorElement);
        $this->documentElement->appendChild($bookElement);
    }
}
$dom = new books();
$dom->load("books.xml");
$dom->addBook("PHP de Luxe", "Richard Samar, Christian Stocker");
print $dom->saveXML();




 --- output ---
  <books>
    <book>
      <title>PHP de Luxe</title>
      <author>Richard Samar, Christian Stocker</author>
    </book>
  </books>
```

Приведенный пример показывает, насколько более гибко, по сравнению с модулем domxml в PHP 4, можно работать с xml-документами.

Несомненно, нововведения и возможности работы с xml-технологиями, реализуемые в различных модулях PHP 5, дают намного больше пространства для маневров и пищи для ума. Естественно ожидать некоторые ошибки и неполную функциональность отдельных модулей, но стоит также заметить, что официального выпуска стабильной версии PHP 5 еще не было, а работа по внедрению поддержки xml-технологий проведена колоссальная. Остается лишь ждать официального релиза PHP 5 и дальнейшего расширения возможностей работы с XML и сопряженными технологиями.

Эффективность применения XML-технологий в веб-приложениях

Возвращаясь к началу, вспомним, что XML - это прежде всего язык разметки документа, отвечающий определенному стандарту. В соответствии с тем, какую информацию несет в себе документ, применение XML в веб-приложениях может быть рассмотрено с нескольких позиций.

- Представление информации о состоянии приложения. Любая информационная система (ИС) так или иначе хранит информацию о своем начальном и текущем состоянии. В этом смысле показательным является описание начального состояния ИС в формате XML. Преимущества такого описания состоят в том, что такая форма записи является более понятной для человека и легко поддается изменениям. Также в описание ИС можно вынести часть логики работы системы. Ярким примером такого подхода является проект Jakarta Struts, в котором данные о начальном состоянии системы и ее поведении содержатся в конфигурационном файле в xml-формате. PHP-портом этого приложения является проект phpMVC;
- Представление результирующих данных приложения. Другим, не менее интересным, применением XML в веб-приложениях является представление результирующих данных в XML-формате. В качестве примера возьмем паттерн MVC. Здесь роль XML ярко проявляется в процессе обмена данными между моделью (model) и видом (view). Более того, используя XSL-преобразования, мы достигаем большей гибкости в представлении данных в зависимости от запроса пользователя. Также XML может выступать как формат представления данных в хранилище данных (как то СУБД, файловая система, другие виды хранилища).
  Возможна и другая архитектура системы, когда «главный» контроллер собирает данные от всех компонент, вызываемых в соответствии с запросом пользователя, и только потом определяет внешнее представление полученных данных, отдаваемых пользователю. При таком подходе XML также может сыграть существенную роль, ведь будет намного удобнее структурировать данные, поступающие от компонент в общий, результирующий документ, а затем представить этот документ в виде, удобном для пользователя.

Немаловажными в этом случае являются возможности обработки xml-документов, такие как XPath-запросы, XSL-преобразования, методы DOM-объектов и другие. Процесс разработки, программирования и внедрения ИС, спроектированной с четким, но, тем не менее, гибким форматом описания данных на основе XML, в этом случае становится более предсказуемым и менее затратным;

- Передача информации внутри приложения и между приложениями. В данном контексте ведущая роль XML стала уже очевидным фактом. В современных протоколах, таких как SOAP, данные описываются в xml-формате. Многие веб-сервисы передают данные, описываемые XML Schema.

  Передача специализированных данных по сети Internet также производится в формате XML. На сегодняшний день существует множество схем представления данных в формате XML от математических формул (MathML) до электронной цифровой подписи (XML Signature). Различные компании объединяют свои усилия по выработке стандартов представления коммерческой информации (CommerceML, EbXML):

XML и веб-формы

Динамичность развития веб-приложений напрямую зависит от их интерактивности с конечным пользователем. Известно, что основной возможностью для интерактивной работы приложения являются веб-формы. Создание веб-форм (далее просто форм) и обработка данных, поступающих от них, вводят в замешательство не только новичков, но порой и опытных программистов.

Сегодня существует множество решений для упрощения процесса создания и обработки форм, например, PEAR::HTML_QuickForm. Но все же HTML-формы нарушают многие принципы языка разметки, часто перемешивая представление и данные. В этом случае представляется перспективным использовать технологию XForms, получившую статус рекомендации консорциума W3C.

XForms — это усовершенствованная версия HTML-форм, которая предоставляет расширяемые средства, позволяющие включать в HTML-документы более богатые и более динамичные формы. С помощью XForms можно ускорить и упростить создание форм. XForms допускают поддержку многочисленных устройств и структурированных данных форм, как, например, XML-документы.

Они избавляют разработчиков от необходимости писать скрипты, которые используются при генерировании динамических форм, необходимых как для объединения многочисленных форм на одной и той же странице, так и управления данными. Наконец, хотя каждая часть XForms, а именно: модель данных (data model), вид (view) и контроллер (controller) — полностью отделима от других и может использоваться с другими технологиями, важно осознать, что мощь этих частей — в их объединении в приложении.

Модель данных XForms позволяет объявлять единицы данных и отделять структуру от любого набора элементов управления, используемых для отображения значений этих данных. XForms описывают средства связывания этих единиц данных с управляющими элементами отдельно от самого объявления режима данных. Помимо этого, для каждой отдельной единицы данных определяются декларативные средства поведения при изменении значений.

Благодаря XForms разработчики могут сосредоточить свои усилия в первую очередь на данных, которые необходимо собирать. Использование стандартных XML-схем позволяет явно определить структуру и тип данных. XForms расширяют эту модель, включив в спецификацию дополнительные ограничения и зависимости. XForms-процессор оценивает и реализует эти ограничения, не испытывая необходимости в дополнительном коде. Процессор проверяет типы данных и ограничения до того, как эти данные представлены для обработки.

Спецификация XForms также предусматривает создание динамических форм посредством задания условий, зависящих от данных. Больше не нужно писать специальный код, который будет генерировать пользовательские формы, основываясь на отклике пользователей. XForms могут настраивать формы «на лету» при сборе данных и изменении условий.

Навигация по XForms обрабатывается моделью событий XForms, которая не зависит от того, как она изображается на клиентской машине. Так, можно представить на одной клиентской машине XForms как отдельную страницу, и на другой — как множество страниц, не заботясь при этом о сохранении состояния и представлении соответствующих контролов для навигации.

Благодаря краткой спецификации и мощным функциональным возможностям, формы, созданные с XForms, легче поддерживать по сравнению с обычными веб-формами. Также отсутствует необходимость в дополнительном коде, предназначенном для проверки типа данных. Структура данных оказывается выведенной из разметки представления формы.

Независимо от того, являются ли XForms в действительности окончательным решением для разделения предназначения и представления, XForms на самом деле являются следующим поколением Web-форм.

Выводы

Язык XML на сегодняшний день является одним из привлекательных языков, способных упростить разработку веб-приложений и продвинуть их на более высокий уровень.

Использование XML-технологий в передаче данных между приложениями и внутри приложения позволяет создавать более гибкие (настраиваемые) интерфейсы как для сторонних приложений, так и для конечного пользователя.

Новые стандарты в области взаимодействия между клиентом и веб-приложением, такие как XForms, позволят расширить возможности интерактивных пользовательских интерфейсов.

Почему PostgreSQL?

Автор доклада:
Алексей Борзов

Для того, чтобы лучше понять текущее положение PostgreSQL в ряду продуктов с открытым кодом, а также для объяснения ряда его уникальных особенностей, стоит обратиться в первую очередь к истории его создания.

Краткая история PostgreSQL

Корни PostgreSQL уходят в 1977 год, в проект Ingres, разрабатывавшийся в 1977 ─ 1985 годах в Университете штата Калифорния. Этот же код развивался компанией Relational Technologies / Ingres (впоследствии купленной компанией Computer Associates), на его основе была создана одна из первых коммерчески успешных РСУБД.

Собственно проект POSTGRES (от Post Ingres) был начат в 1986 в Университете штата Калифорния в Беркли под руководством профессора Майкла Стоунбрейкера. Основной идеей было сделать расширяемую систему: с возможностью добавлять свои типы данных, операторы, методы доступа, а также создавать триггеры и уведомления. Первая версия, доступная для некоторых внешних пользователей, вышла в 1989 году. В ответ на их критические замечания в код вносились изменения, вскоре вышли версии 2 и 3. На основе POSTGRES были реализованы исследовательские и коммерческие проекты, сам POSTGRES использовался для обучения в нескольких университетах. Впоследствии код POSTGRES был коммерциализован компанией Illustra, купленной затем Informix, в свою очередь купленной IBM.

Количество внешних пользователей в 1993 году удвоилось, и стало очевидно, что на поддержку существующего кода и работу с пользователями тратится слишком много времени, которое могло бы быть использовано для исследований в области баз данных. Поэтому проект POSTGRES был закрыт в 1994 году на версии 4.2.

В 1994 году два выпускника Калифорнийского университета, Джолли Чен и Эндрю Ю, добавили в POSTGRES поддержку языка SQL (до этого в нём использовался свой язык запросов PostQUEL). Результат этой работы получил имя Postgres95 и был выпущен в свет под свободной лицензией.

В 1996 году стало очевидно, что выбор имени Postgres95 был, мягко говоря, не слишком удачной идеей, и проект был переименован в PostgreSQL. Новое имя указывало как на его корни — на это же указывало и начало нумерации версий с 6.0, — так и на поддержку языка SQL. Тогда же была организована и Всемирная[1] группа разработчиков PostgreSQL, осуществляющая с тех пор развитие проекта.

1 В группу входят программисты из Северной Америки, Европы, Японии. Огромный вклад в PostgreSQL внёс программист из России Вадим Михеев, в настоящее время отошедший от активной разработки.

Особенности разработки и распространения

В отличие от прочих известных СУБД с открытым кодом, в истории PostgreSQL нет периода коммерческой разработки. Также нет компании, которая могла бы контролировать проект, и нет единственного «диктатора» (как, например, Линус Торвальдс для Линукса), который определял бы направление разработки и единолично принимал или отвергал предлагаемые изменения. Руководство проектом осуществляет Управляющий комитет (Steering Commitee), при этом соблюдается правило, что большинство его членов не могут быть сотрудниками одной компании.

PostgreSQL распространяется под лицензией BSD, позволяющей, в частности, распространять изменённые версии PostgreSQL, не открывая их исходный код. Так, например, существуют несколько коммерческих версий PostgreSQL для платформы Windows, в то время как порт с открытым кодом лишь ожидается в версии 7.5.

Огромные возможности расширения сервера, заложенные ещё в рамках проекта POSTGRES, позволяют реализовывать в рамках независимых проектов функции, которые в прочих СУБД должны быть встроены в «ядро» сервера. В PostgreSQL, например, нет встроенной репликации, хотя существует несколько внешних решений для её реализации; геометрические типы данных для PostgreSQL разрабатываются в рамках независимого проекта PostGIS.

Все эти особенности позволяют многим компаниям (а не одной) основывать свой бизнес на PostgreSQL. Такие компании как Red Hat, Afilias (управляющая доменами .org и .info), Fujitsu дают работу ведущим разработчикам PostgreSQL и/или оплачивают разработку новых возможностей. Существует несколько независимых компаний, осуществляющих техническую поддержку PostgreSQL, при этом если одна из них вдруг обанкротится[2], это не будет концом технической поддержки и тем более концом проекта.

Обзор возможностей PostgreSQL

По имеющейся функциональности PostgreSQL приближается к ведущим коммерческим РСУБД и считается самым продвинутым из баз с открытым кодом. При этом большая часть этой функциональности была реализована несколько лет назад и с тех пор была хорошо протестирована в «боевой» обстановке.

Транзакции. Видимо, поддерживаются с самого начала.

Версионный движок. С версии 6.5 (вышла 1999.06.09) используется версионная модель обеспечения целостности данных при параллельном доступе. Это означает, что транзакция видит версию данных, актуальную на момент своего начала, при этом другая транзакция параллельно может записать новую версию этих же самых данных.

2 А такой случай в истории PostgreSQL уже был: компания Great Bridge

Это позволяет запросам на чтения не ждать окончания запросов на запись. С версии 7.1 (вышла 2001.04.13) реализован лог транзакций (Write Ahead Log), позволивший отказаться от принудительного сброса данных на диск при успешном окончании транзакции.

Подзапросы. Реализованы начиная с версии 6.3 (вышла 1998.03.01), подзапросы в конструкции `FROM` (известные как derived tables) поддерживаются с версии 7.1.

Внешние ключи. Синтаксис поддерживается с версии 7.0 (вышла 2000.05.08), до этого внешние ключи можно было реализовать вручную через использование триггеров.

Представления. Текущая реализация появилась в версии 6.4 (1998.10.30), в какой-то форме существовали с самого начала. Следует отметить, что возможна реализация обновляемых представлений, для этого используется собственное расширение стандарта SQL — правила (RULE).

Хранимые процедуры. В версии 6.4 появилась возможность писать хранимые процедуры на языке PL/PgSQL[3]. С тех пор была добавлена поддержка ещё нескольких языков (TCL, Perl, Python), в настоящее время в разработке находятся PL/Java и PL/PHP, речь о котором пойдёт ниже.

Триггеры. Появились в версии 6.2 (вышла 1997.10.02).

Unicode. Многобайтовые кодировки (в частности Unicode) поддерживаются с версии 6.4. Именно в этом причина популярности PostgreSQL в Японии и практической неизвестности там более распространённых в других частях света СУБД с открытым кодом.

Внешние проекты

Как уже было сказано выше, часть дополнительных возможностей реализуется в рамках отдельных проектов. Некоторые из них[4] распространяются вместе с сервером в пакете postgresqlontrib (каталог contrib/ в случае полного набора исходников), некоторые отдельно. Существует также сайт http://www.pgfoundry.org/, предоставляющий хостинг[5] для проектов с открытым кодом, использующих PostgreSQL, и предыдущая версия этого сайта по адресу http://gborg.postgresql.org/.

Перечислим несколько полезных проектов:

- Репликация;
- contrib/rserv: простейшее средство асинхронной репликации с одним мастером;
- ERServer: коммерческое решение для репликации использующееся, в частности, для поддержки доменов .org и .info, выпущенное недавно под лицензией BSD;

3 Сильно напоминает PL/SQL из Oracle.
4 Требующие исходного кода сервера для сборки или могущие впоследствии быть включены в сервер.
5 Сам сайт использует проект GForge (http://www.gforge.org/) — fork кода SourceForge, работающий на PostgreSQL.

- Slony I[6]: новое решение для асинхронной репликации, в настоящее время приближается к этапу бетатестирования;
- Полнотекстовый поиск: наиболее продвинутым решением является contrib/tsearch2;
- Геометрические типы: реализованы в проекте PostGIS (http://www.postgis.org/).

Зачем нужны все эти возможности?

Реализация логики на стороне сервера позволяет:

- Разгрузить клиентский код. Кроме того, при написании нескольких клиентов для одной и той же базы не понадобится дублировать одну и ту же функциональность в каждом клиенте;
- Уменьшить трафик между клиентом и сервером (пересылка запросов на сервер, получение промежуточных результатов);
- Провести денормализацию базы для увеличения производительности без потери преимуществ нормализации (а основное её преимущество — поддержка целостности и непротиворечивости данных);

Не говоря уже о том, что часть функциональности (например, триггеры) на стороне клиента полноценно реализовать невозможно.

Пример: реализуем функциональность MySQL 4.1.x

В ветке 4.1.x СУБД MySQL появилась новая агрегатная функция group_concat(), позволяющая получить конкатенацию группы строк. В PostgreSQL такой функции нет, зато есть возможность определить свою агрегатную функцию, которой мы и воспользуемся для создания самодельного варианта group_concat():

```
--Функция, возвращающая объединение двух строк
CREATE FUNCTION sql_concat(text, text) RETURNS text AS '
SELECT $1 || $2; ' LANGUAGE 'sql' WITH (ISSTRICT);
--агрегатная функция, использующая sql_concat
CREATE AGGREGATE group_concat (
BASETYPE = text,
SFUNC  = sql_concat,
STYPE  = text
);
```

6 Имеется в виду русское слово «слоны».

Протестируем результат наших трудов:

```
test=# select field from foobar;
field
--------
foo
bar
(2 rows)
test=# select group_concat(field) from foobar;
group_concat
--------
foobar
(1 row)
```

К сожалению, у нас нет возможности задать дополнительный параметр для агрегатной функции и использовать его в качестве разделителя, но эту проблему можно обойти. Оставим это в качестве упражнения для читателей.

Также в ветке 4.1.x MySQL появился расширенный синтаксис для полей типа timestamp, позволяющий указать, что при вставке и обновлении записи в это поле должно вставиться текущее время. В PostgreSQL никакого расширенного синтаксиса не нужно: есть возможность указать функцию в качестве значения по умолчанию для поля (для вставки) и можно создать триггер, изменяющий значение поля (для обновления). Продемонстрируем:

```
CREATE TABLE foo (
...
modified timestamp(0) with time zone DEFAULT now() NOT NULL );
--функция для изменения поля
CREATE FUNCTION foo_modified()
RETURNS TRIGGER
AS '
BEGIN
        NEW.modified := now();
   RETURN NEW;
END;

' LANGUAGE 'plpgsql';
--Триггер, изменяет поле до обновления записи
CREATE TRIGGER foo_modified
BEFORE UPDATE ON foo FOR EACH ROW
EXECUTE PROCEDURE foo_modified();
```

Пример: встраиваем phpBB2 в сайт

Разберём решение достаточно часто встречающейся в реальной жизни задачи: требуется добавить в существующий сайт форум phpBB2[7]. Проблема в том, что сайт использует для хранения информации о пользователях свою таблицу users (с небольшим количеством полей), а форум — свою таблицу phpbb_users. С ходу возникают следующие идеи:

1. Заполнять обе таблицы. Но: ненужное дублирование данных, требуются изменения в код сайта и в код phpBB2;

7 Спасибо его разработчикам за поддержку PostgreSQL.

2. Заставить сайт использовать таблицу phpbb_users. Но: возникает абсолютно ненужная «завязанность» на конкретный форум;
3. Заставить phpBB2 использовать таблицу users. Но, изменения в коде phpBB2 вызовут проблемы при попытке его обновить.

Реализуем решение, лишённое вышеперечисленных недостатков:

```
   DROP TABLE phpbb_users;
--таблица с полями, присутствовавшими в phpbb_users,
--но отсутствующими в users
CREATE TABLE phpbb_users_info (
        ...
);
--представление с полями, полностью соответствующими
--бывшей таблице phpbb_users
CREATE OR REPLACE VIEW phpbb_users AS
SELECT ...
FROM users u, phpbb_users_info ui
WHERE u.user_id = ui.user_id;
```

Таким образом у нас есть представление, с точки зрения форума неотличимое от обычно используемой им таблицы. Остаётся добавить возможность добавлять в «таблицу» записи и обновлять и удалять существующие:

```
CREATE OR REPLACE RULE phpbb_users_insert AS
ON INSERT TO phpbb_users DO INSTEAD (
INSERT INTO users ...;
INSERT INTO phpbb_users_info ...;
);

CREATE OR REPLACE RULE phpbb_users_update AS
ON UPDATE TO phpbb_users DO INSTEAD (
UPDATE users SET ...;
UPDATE phpbb_users_info SET ...;
);
CREATE OR REPLACE RULE phpbb_users_delete AS
ON DELETE TO phpbb_users DO INSTEAD (

DELETE FROM users WHERE user_id = OLD.user_id;
--А из таблицы phpbb_users_info удалит внешний ключ
);
```

Основная функциональность реализована, при этом каких-либо изменений в код phpBB2 (да и сайта) не понадобилось. Обновить phpBB2 при необходимости будет достаточно легко.

Разрабатываемая функциональность

Процедурный язык PL/PHP

Язык PL/PHP позволяет вам писать функции (также известные как хранимые процедуры) для PostgreSQL на языке PHP. Официальный сайт проекта: http://plphp.commandprompt.com/, в настоящий момент доступна версия RC1.

В язык PHP добавляется новая функция spi_exec_query(), позволяющая выполнять запросы в контексте той транзакции, в которой была вызвана функция на PL/PHP. Функции могут возвращать наборы записей, использоваться для создания триггеров.

Как и некоторые другие процедурные языки, PL/PHP существует в двух вариантах: trusted и untrusted. Процедуры на trusted языке могут создавать все пользователи, но функциональность языка ограничена (используется PHP safe mode), на untrusted — только суперпользователи, но доступна вся функциональность PHP.

Приведём пример функции на PL/PHP:

```
CREATE TYPE __testphptype AS (field text);
CREATE OR REPLACE FUNCTION phpsplit(text)

RETURNS SETOF __testphptype AS '
$words = preg_split("/\\\\s+/", $args[0]);
$ret = array();
for ($i = 0, $cnt = count($words); $i < $cnt; $i++) {
$ret[$i][0] = $words[$i];
}
return $ret;
' LANGUAGE 'plphp' WITH (ISSTRICT);
```

и пример ее работы:

```
testphp=# select * from phpsplit('This is PL/PHP at work.');
field
----------
This
is
PL/PHP
at
work.
(5 rows)
```

Версия 7.5: PostgreSQL для платформы Win32

До настоящего времени у желающих использовать PostgreSQL на платформе Windows было две возможности:

- Запускать его с использованием окружения Cygwin, переводящего системные вызовы Linux в вызовы Windows;
- Купить коммерческую версию у компании, продающей такую версию.

Работа над портом под Windows с открытым кодом была начата в процессе разработки версии 7.4, но к выходу этой версии её, к сожалению, закончить не успели. С высокой вероятностью этот порт выйдет в версии 7.5, уже сейчас он практически готов: ознакомиться с текущим статусом можно на странице http://candle.pha.pa.us/main/writings/pgsql/project/win32.html.

На данной момент нет (и до начала бетатестирования версии 7.5 не будет) официальной сборки PostgreSQL для Windows, но на вышеуказанной странице есть ссылка на неофициальную ежедневную сборку.

Также достаточно несложно собрать этот порт самому:

1. Сборка осуществляется компилятором gcc в среде MinGW (для запуска собранной версии MinGW не требуется). Скачать MinGW и MSYS можно на сайте http://mingw.org/;
2. Также нужны некоторые дополнительные пакеты (zlib, bison, flex), проще всего скачать их бинарные версии по адресу http://www63.tok2.com/home/bitwalk/download.html
3. Требуется язык Perl. Подойдёт любой дистрибутив для Windows, например ActivePerl (http://activestate.com/);
4. Вам, естественно, понадобится CVS-версия исходного кода PostgreSQL, её можно скачать на любом FTP зеркале (http://www.postgresql.org/mirrors-ftp.html) в каталоге dev/;
5. Дальше всё стандартно:

```
$ ./configure && make && make install
```

Собранную (или скачанную из интернета) бинарную версию поместите, например, в C:\pgsql, добавьте в переменную окружения PATH пути к подкаталогам bin/ и lib/, создайте новую переменную окружения PGDATA, которая будет указывать на путь к каталогу с базами данных. После этого проинициализируйте этот каталог командой:

```
> initdb L «C:/pgsql/share»
```

и можете запускать сервер (postmaster.exe).

Существует (опять же неофициальный) PostgreSQL service manager, позволяющий запускать PostgreSQL не вручную, а как сервис. Скачать его можно по адресу http://www.unm.edu/~efesar/pgsvc/index.html.

Ответы на вопросы

В этом разделе я постараюсь дать ответы на вопросы, заданные мне на конференции. Очевидно, что ответы (как впрочем и вся эта статья) отражают мою собственную точку зрения, и вы имеете полное право с ними не согласиться.

Недостатки

Основных недостатков я вижу два:

- Отсутствие версии для Windows. Современные версии Windows уже подходят под роль серверной платформы, да и огромное количество программистов пишут свои программы именно в среде Windows. Поэтому необходимость отдельного Unix-сервера или использования «обёртки» Cygwin (не способствующей повышению устойчивости и производительности) мешает более широкому использованию PostgreSQL;

- То, что по-английски называется steep learning curve. Для того, чтобы полноценно использовать PostgreSQL, надо быть достаточно хорошо знакомым с языком SQL и реляционной моделью. Для того, чтобы добиться хорошей производительности, надо посвятить некоторое время настройке сервера и т.д.

Целесообразность использования

Для использования на большей части сайтов подойдёт вообще любая СУБД (в качестве примера позвольте привести «СУБД» SQLite), следовательно, подойдёт и PostgreSQL. Есть сайты (например, поисковые системы), где не подойдёт вообще никакая, кроме специально написанной.

Если серьёзно, то сильные стороны PostgreSQL — средства поддержания целостности и хорошая масштабируемость при смешанной (чтение и запись) нагрузке. Целесообразно использовать его на сайтах с часто изменяющимися данными (особенно если эти данные для вас представляют ценность) и большим количеством посещений.

Производительность

Производительность бывает двух видов: удовлетворительная и неудовлетворительная. Если оцениваемые продукты имеют удовлетворительную производительность, то можно задуматься и о прочих критериях отбора (например, функциональности). Кроме того, имеет смысл сравнивать производительность приложений, а не серверов БД, т.к. высокая скорость сервера может с лихвой компенсироваться низкой скоростью клиентского кода, реализующего отсутствующую в сервере функциональность.

Лично я не видел опубликованных результатов независимого тестирования производительности СУБД, включающих более менее новую версию PostgreSQL. Желающим же провести такие тесты желаю всяческих успехов и предлагаю ознакомиться со своей статьёй, посвящённой повышению производительности PostgreSQL: http://detail.phpclub.net/store/html/postgresql/.

Использование PEAR для ускорения разработки веб-приложений

Автор доклада:

Алексей Борзов

Достаточно часто возникает вопрос: стоит ли использовать чужой код? Дать однозначный ответ на этот вопрос невозможно, ибо надо принимать во внимание как вашу собственную квалификацию, так и качество этого самого кода и, следовательно, квалификацию его автора.

Интересный взгляд на эту проблему предлагает Джоэл Спольски в своём эссе In Defense of «Not Invented Here» Syndrome (http://joelonsoftware.com/articles/fog0000000007.html). Основная мысль такова: если в вашей команде собраны лучшие программисты, то, конечно же, стоит писать всё самим, ибо код, написанный другими — хлам. Если же это не вполне так, то, по крайней мере, надо реализовывать своими силами наиболее важные и уникальные для вашего приложения части.

Итак, для стандартных задач (которых в веб-программировании много) проще использовать готовые стандартные решения. Достоинством их, в частности, является большое число пользователей и, следовательно, малый шанс на наличие серьёзных неисправленных ошибок. А сэкономленное время стоит посвятить решению нестандартных задач, тех, что выделяют ваш сайт из общей массы.

Таким образом, мы переходим к следующему вопросу: где искать хороший чужой код? Один из возможных ответов: в PEAR.

Обзор PEAR

PEAR расшифровывается как PHP Extension and Application Repository (Репозиторий расширений и приложений для PHP). Проект предлагает следующее:

- Структурированную библиотеку пакетов с открытым кодом;
- Стандартный стиль кодирования и встроенной документации API;
- Систему распространения и обновления пакетов.

PEAR, впрочем, не является библиотекой классов в том смысле, в котором это понимается, например, в Java. Не является он также и каркасом (Framework) для веб-приложений.

Особенности PEAR

Возникает следующий закономерный вопрос: не очередная ли это помойка (или кладбище) скриптов на PHP, которых в интернете и так изрядное количество?

Ответ таков: основатели PEAR думали о возможности превращения в «помойку» и старались его не допустить.

Для этого предусмотрены следующие барьеры:

- Для помещения своего пакета в PEAR разработчику надо выполнить определённые условия: пакет должен удовлетворять стандартам кодирования и должен иметь документацию (как минимум в виде комментариев в стандарте PHPDoc);
- Предложение нового пакета должно пройти через процедуру голосования, где разработчики PEAR могут высказать свои замечания и потребовать их исправления до включения пакета в репозиторий;
- Дублирование функциональности, как правило, не приветствуется. Это не означает, что такой пакет не может быть принят, но его автору придётся объяснить, чем он лучше уже существующего. Это, хотя и мешает, к сожалению, естественному отбору, но зато и не позволяет появиться, например, десяткам пакетов «абстрактного доступа к БД».

Превращению же PEAR в «кладбище» мешает наличие сообщества разработчиков. Большая часть разработчиков с благодарностью принимает патчи и предложения по улучшению пакетов, а часто приветствует и новых разработчиков для своих пакетов. Если же изначальный разработчик пакета вдруг теряет к нему интерес (или попросту не имеет времени на поддержку), то есть хороший шанс найти заинтересованных людей, готовых продолжить разработку. Так, например, из шести пакетов, которые поддерживает автор этих строк, с нуля им были написаны лишь два.

Инфраструктура PEAR

PEAR предлагает разработчикам и пользователям большой набор возможностей. Основу инфраструктуры составляет, безусловно, сайт http://pear.php.net, который имеет два интерфейса:

- Веб-интерфейс для людей;
- XML-RPC интерфейс. Именно к нему вы обращаетесь, когда используете команду pear для установки пакетов;

Собственно сайт предоставляет пользователям следующие сервисы:

- Просмотр списка пакетов, поиск, скачивание;
- Статистика скачиваний;
- Багтрекер;
- Документация. Сюда входит как руководство по PEAR, сгенерированное по файлам в формате DocBook, которые авторы (или переводчики) пишут вручную, так и документация API для каждой версии каждого пакета, автоматически генерируемая программой phpDocumentor по специальным комментариям в коде.

Для разработчиков пакетов на сайте также доступны возможности по их управлению: закачивание новых версий, редактирование информации и т.п. Также в эту группу входит весьма важный сервис PEPr (PEAR Proposals), посредством которого проходят голосования по новым пакетам и всяческим предложениям (RFC) по дальнейшему улучшению PEAR. Существует несколько списков рассылки, посвящённых PEAR, основные:

- pear-general список для пользователей PEAR, здесь разработчики пакетов отвечают на вопросы и консультируют по поводу оптимального использования своих творений;
- Pear-dev список для общения разработчиков PEAR;
- Pear-doc список, посвящённый вопросам документирования PEAR и перевода документации.

Наиболее популярные пакеты

Первый же взгляд на статистику скачиваний подтверждает, что наибольшей популярностью пользуются пакеты, предназначенные для решения как раз вполне стандартных и весьма часто встречающихся задач веб-программирования.

Так, в первую двадцатку попадают пакеты Mail, Mail_Mime и Net_SMTP, облегчающие задачу отправки писем, в том числе с вложениями, пакет HTTP_Request для отправки запросов HTTP, а также полезные вспомогательные пакеты Auth, Date, Log, XML_Parser. Чуть не дотягивают до первой двадцатки пакеты HTML_TreeMenu (создание древовидных меню) и Pager (разбиение на страницы).

В первую двадцатку попадает и пакет для облегчения работы с HTML формами HTML_QuickForm, речь о котором пойдёт далее.

Пакет HTML_QuickForm

Перечислим основные задачи, возникающие при работе с HTML формами:

- Вывод формы: пустой либо, в случае формы для редактирования чего-либо, со значениями по умолчанию. Уже эта задача не вполне тривиальна для полей типа select, radio, checkbox;
- Фильтрация введённых значений. Если мы знаем, что поле ID всегда целочисленное, то имеет смысл обработать его функцией intval();
- Проверка введённых значений. Если поле в форме называется «email», то надо проверить, что туда был введён именно электронный адрес, а не кличка любимой собаки пользователя;
- Для минимизации затрат на обработку неверно заполненных форм на сервере имеет смысл реализовать также проверку на стороне клиента;
- В случае ошибки ввода форма должна быть выведена вновь с соответствующими сообщениями. При этом хорошим тоном считается подстановка введённых значений, дабы пользователю было легче исправить ошибки.

Затраты времени и сил на написание всего необходимого для решения этих задач кода «с нуля» будут весьма велики. Не говоря уже о том, что работа по написанию такого кода очень однообразна и утомительна. Поэтому различные библиотеки для работы с формами пользуются большим успехом, пакет HTML_QuickForm, например, скачивался уже более 55000 раз.

Возможности HTML_QuickForm

- Более 20 типов элементов. Сюда входят как стандартные элементы HTML, так и собственные: дата, иерархический выбор. Есть также возможность добавить свой тип (для этого надо создать подкласс HTML_QuickForm_element и реализовать в нём ряд методов);
- Возможность объединять элементы в группу. Это позволяет обращаться с несколькими элементами как с одним, что полезно при выводе формы и при проверке введённых значений;
- Проверка на стороне сервера. Более 10 встроенных правил, есть возможность добавить свои: либо как подклассы HTML_QuickForm_Rule (в этом случае можно реализовать также и проверку на стороне клиента), либо как callback функции;
- Проверка на стороне клиента. Может включаться в дополнение к проверке на стороне сервера при добавлении правила, реализуется в тех же подклассах HTML_QuickForm_Rule;
- Полностью настраиваемый вывод, осуществляющийся через подклассы HTML_QuickForm_Renderer, реализующие паттерн «Посетитель». Поддерживается вывод напрямую в HTML, в массив, через шаблонный движок. Есть поддержка Smarty, IT[X]/Sigma, Flexy;
- При работе загружается только реально используемый код.

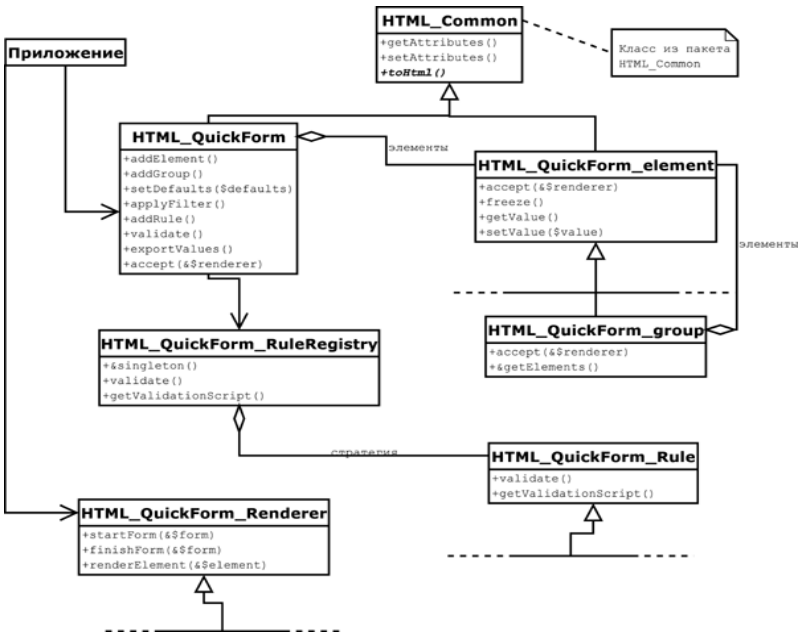


Рис. 1. Основные классы пакета QuickForm.

На рисунке 1 (выше) приведена диаграмма базовых классов пакета HTML_QuickForm с основными методами (полная диаграмма потребовала бы слишком много места).

Пример работы

Приведём простейший пример работы с QuickForm

```
<?php
// загружаем основной класс, создаём объект
 require_once 'HTML/QuickForm.php';
$form = new HTML_QuickForm('firstForm');
// задаём значения по умолчанию
$form->setDefaults(array('name' => 'Вася Пупкин'));
// добавляем специальный элемент: заголовок в форме
$form->addElement('header', null, 'Пример использования QuickForm');
// добавляем обычные элементы: поле для ввода и кнопку
$form->addElement('text', 'name', 'Введите имя:', array('size' => 50,
'maxlength' => 255));
$form->addElement('submit', null, 'Отправить');
// убираем лишние пробелы в поле name
$form->applyFilter('name', 'trim'); // добавляем правило для проверки.
// Обратите внимание: включаем также проверку на стороне клиента
$form->addRule('name', 'Пожалуйста введите имя', 'required', null, 'client');
// Обрабатываем введённые значения в случае успешной проверки
if ($form->validate()) {
       echo '<h1>Здравствуй, ' . htmlspecialchars($form->exportValue('name')) .
'!</h1>';
        // Выводим форму (используя вывод по умолчанию) в противном случае
} else {
        $form->display();
}
?>
```

Листинг 1

Здесь показаны все основные моменты использования пакета: добавление элементов в форму методом `addElement()`, указание правил проверки методом `addRule()`, проверка формы методом validate(), получение введённых значений методом `exportValues()` и вывод формы методом `display()`.

К пакету прилагаются более интересные примеры, покрывающие большую часть его функциональности, в том числе настройку внешнего вида формы. В руководстве по PEAR также имеется достаточно подробная документация.

Пакет HTML_QuickForm_Controller

HTML_QuickForm_Controller — это надстройка над пакетом HTML_QuickForm, в первую очередь предназначенная для создания многостраничных форм.

Пакет реализует паттерн PageController, в применении к формам это означает следующее: есть один скрипт, который выводит и проверяет различные формы в зависимости от параметров запроса.

Принцип его работы: кнопкам отправки формы присваиваются специальные имена, содержащие имя формы и действие, экземпляр класса Controller проверяет пришедшие в запросе переменные и вызывает соответствующий обработчик. Также Controller отвечает за хранение в сессии данных формы.

В пакет включены обработчики по умолчанию для стандартных действий: «Вперёд» и «Назад» в форме типа wizard, перехода на конкретную страницу многостраничной формы и т.п.

Диаграмма основных классов пакета приведена на рисунке 2.

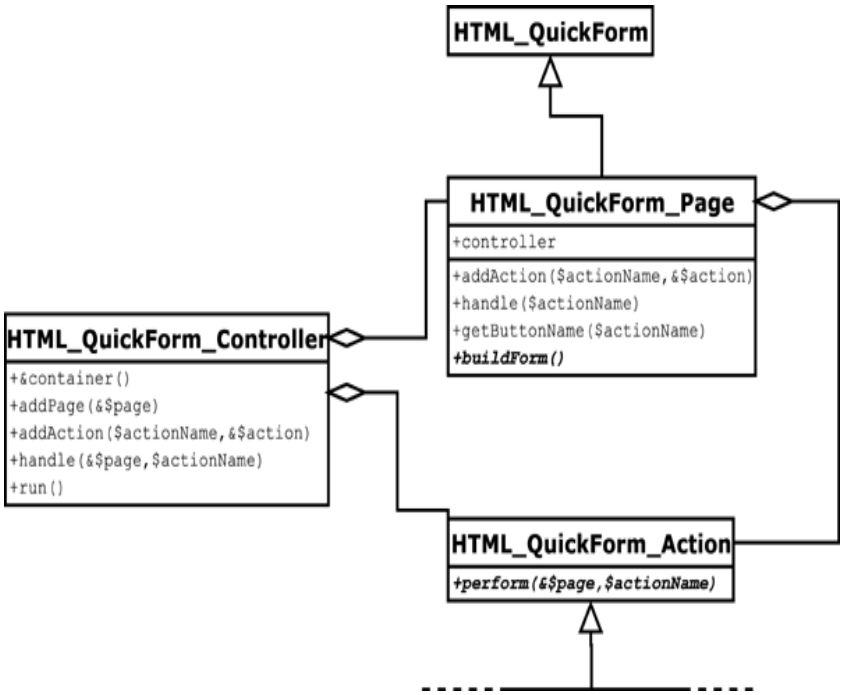


Рис. 2: Основные классы пакета QuickForm_Controller

Пример работы

Приводимый ниже пример по функциональности абсолютно аналогичен примеру для пакета HTML_QuickForm (см. листинг 2 ниже).

Как можно заметить, пакет QuickForm_Controller рассчитан на больших любителей ООП: простейший пример требует объявления двух новых классов. Его достоинства проявляются на более сложных многостраничных формах, примеры которых прилагаются к пакету.

```
<?php
require_once 'HTML/QuickForm/Controller.php'; // Загружаем базовые классы.
require_once 'HTML/QuickForm/Action.php';
class FirstPage extends HTML_QuickForm_Page // Класс, представляющий форму
{
function buildForm()
{
$this->_formBuilt = true;
$this->addElement('header', null, 'Пример для QF_Controller');
$this->addElement('text', 'name', 'Введите своё имя:',
array('size' => 50, 'maxlength' => 255));
// Устанавливаем имя для кнопки: будет вызван обработчик
$this->addElement('submit', $this->getButtonName('submit'),
'Отправить'); $this->applyFilter('name', 'trim'); $this->addRule('name',
'Please enter your name', 'required', null, 'client');
// Действие по умолчанию (можно нажать Enter, а не кнопку)
$this->setDefaultAction('submit');
}
}
// Действие для обработки формы в случае успешной отправки
class ActionProcess extends HTML_QuickForm_Action
{

function perform(&$page, $actionName) { echo '<h1>Привет, ' . htmlspecialchars
($page->exportValue('name')) . '!</h1>';
} }
// Создаём объект для страницы и добавляем в него обработчик
// Для остальных действий обработчики по умолчанию
$page =& new FirstPage('firstForm'); $page->addAction('process', new
ActionProcess());
// Создаём объект класса Controller, задаём значения по умолчанию
$controller =& new HTML_QuickForm_Controller('tutorial');
$controller->setDefaults(array( 'name' => 'Вася Пупкин'
));
$controller->addPage($page);
$controller->run(); // Обрабатываем запрос
?>
```

Листинг 2

Ответы на вопросы

В этом разделе я постараюсь дать чуть более развёрнутые ответы на вопросы, которые задавались мне на конференции, в том числе в кулуарах.

Наиболее часто задавался вопрос о средствах генерирования кода для QuickForm, например, по описанию формы в XML. Такие идеи были, существуют даже тестовые реализации. Несколько месяцев назад Бертран (ведущий разработчик QuickForm) начал прорабатывать вопрос о создании пакета под условным названием QuickForm Builder.

Я решил не принимать в проекте участие за недостатком времени (да и интереса), и потому не был в курсе его текущего статуса. После конференции я проконсультировался у Бертрана, и узнал следующее: все наработки по проекту есть в Wiki по адресу http://www.mamasam.com/qfbuilder , работа по написанию не велась опять же за недостатком времени. Если есть желание поучаствовать, изучайте Wiki, задавайте вопросы в списке рассылки pear-dev.

Есть ещё достаточно интересный пакет DB_DataObject_FormBuilder, надстройка над ORM пакетом DB_DataObject. Он позволяет сгенерировать форму по подклассу DB_DataObject, который в свою очередь генерируется по таблице в базе.

О стандарте XForms: вопрос разработчиками QuickForm изучался, пришли к выводу, что идеология XForms слишком отличается от того, что реализовано на данный момент в QuickForm.

На вопросы о всяческом хитром выводе формы могу сказать следующее: наиболее гибкий вариант — вывести форму в массив и далее делать с этим массивом всё, что угодно.

О расширении функциональности пакета: как уже было сказано выше, предусмотрено подключение ваших собственных типов элементов, правил проверки и даже классов вывода (Renderer). Вопросы создания своих элементов и классов вывода освещены в документации. Раздел про правила в документации, к сожалению, пока не появился, но написать свой класс для проверки не слишком сложно. В архивах списка рассылки pear-dev можно найти, например, предлагавшийся элемент для работы с wysiwyg редактором htmlarea.

Приложение I. Слайды

Использование технологии объектно-реляционных отображений в разработке веб-проектов

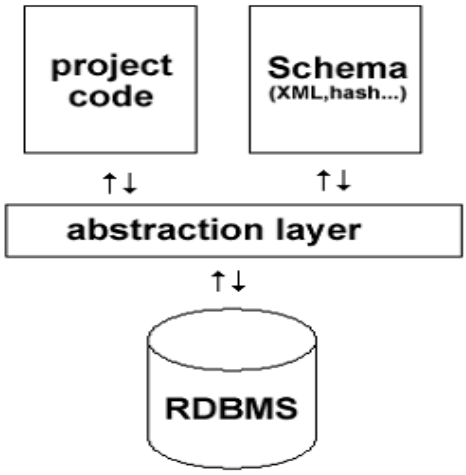
В этом докладе:

- Основы ORM-подхода, базовые модели и приципы отображения, популярные решения для JAVA, Perl, PHP, Python;
- Пример реализации с использованием PHP;
- Типичные практические вопросы;
- Резюме, достоинства и недостатки метода;

Автор доклада:

Алексей Рыбак

ORM: схема приложения, пример (Propel)



ORM: наиболее известные проекты

JAVA:

- TopLink (Oracle),
- WebObjects (Apple),
- JDO,Hibernate,Torque,
- OJB,Castor,Cocobase, VBSF ...

Python

- DbObj, SQLObject,db_row,
- Forget SQL, Middle Kit,
- Modelling OR Bridge,
- OR Membrane, PyDo...

Perl

- Class::DBI, Alzabo,
- Tangram, SPOPS,
- DBIx::DBO2 ...

PHP

- PEAR::DB_DataObject,
- Metastorage (MDB),
- Propel ...

Базовые «модели» и операции

CRUD: `create`, `read`, `update`, `delete`

Взгляд со стороны модели сущность-связь:

- (M1) одна таблица, простой первичный ключ

  A {id, a1, ... aN }

- (M2) 1:m, 2 таблицы:

  A{ id, a1, ... aN}, B{id,id_A,b1,...bN}

  id_A - внешний ключ

- (M3) m:n, 3 таблицы: 2 (сущности) + 1 (связь)

  A{id, a1, ... aN}, B {id,b1,...,bN}, A_B{id_A,id_B,...},

  id_A, id_B - внешние ключи

M1: одна таблица, никаких внешних ключей:

READ/SELECT

- атрибуты
- операции над ними в SQL-выражении
- частичная загрузка (lasy loading) - например, LOB
- where: формализация условий
- разбивка на страницы (для ряда СУБД - подзапросы)
- order by, сортировка «по умолчанию»
- bind параметров (настройка производительности)
- group by (having)

CREATE/INSERT, UPDATE

- атрибуты: defaults, triggered
- значения:
- экранировка (value quoting, разные реализации)
- автоинкремент-поля (MySQL autoincrement, последовательности, триггеры)
- «внутренние» параметры (текущее время, пользователь, etc)
- LOB-поля (для ряда СУБД работа с такими полями нетривиальна)
- where(для UPDATE): обычно просто по значению ключа (но в общем случае - конечно, нет)

DELETE

where: также обычно по значению ключа

M2: две таблицы, 1:m:

A{ id, a1, ... aN}, B{id,id_A,b1,...bN}, id_A - внешний ключ в дополнение к (M1) ─ составные объекты и поддержка целостности

A: содержит «коллекцию» B

Операции над коллекцией должны отображатся в операции над B:

- Чтение A ─ опциональная выборка соотвествующих B (lasy);
- Удаление A или элемента из массива 2 варианта: SET NULL или удаление в B

выборка: становится необходимым подсчет числа ссылок

B: ассоциирован с A

чтение B - чтение A:

- Lasy: игнорировать;
- Отдельный запрос по первичному ключу;
- Соединение при выборке из B (left outer join).

В большинстве случаев обновление B не отображается в операции над A.

M3: три таблицы, m:n:

A{id, a1, ... aN}, B {id,b1,...,bN}, A_B{id_A,id_B,...}

Каждый из объектов может содержать коллекцию ссылок на «связанные» объекты

Операции (из-за симметрии - и A, и B):

- Выборка
- Каскадная загрузка, полная или «ленивая» (например, только pkeys, или pkeys + поля-заголовки)
- Подсчёт числа связанных объектов
- Обновление
- Операции над коллекциями (вставка, удаление)
- Операции над атрибутами связи
- Удаление (каскадирование)

В ряде случаев необходимо отображение «связи» в отдельный объект.

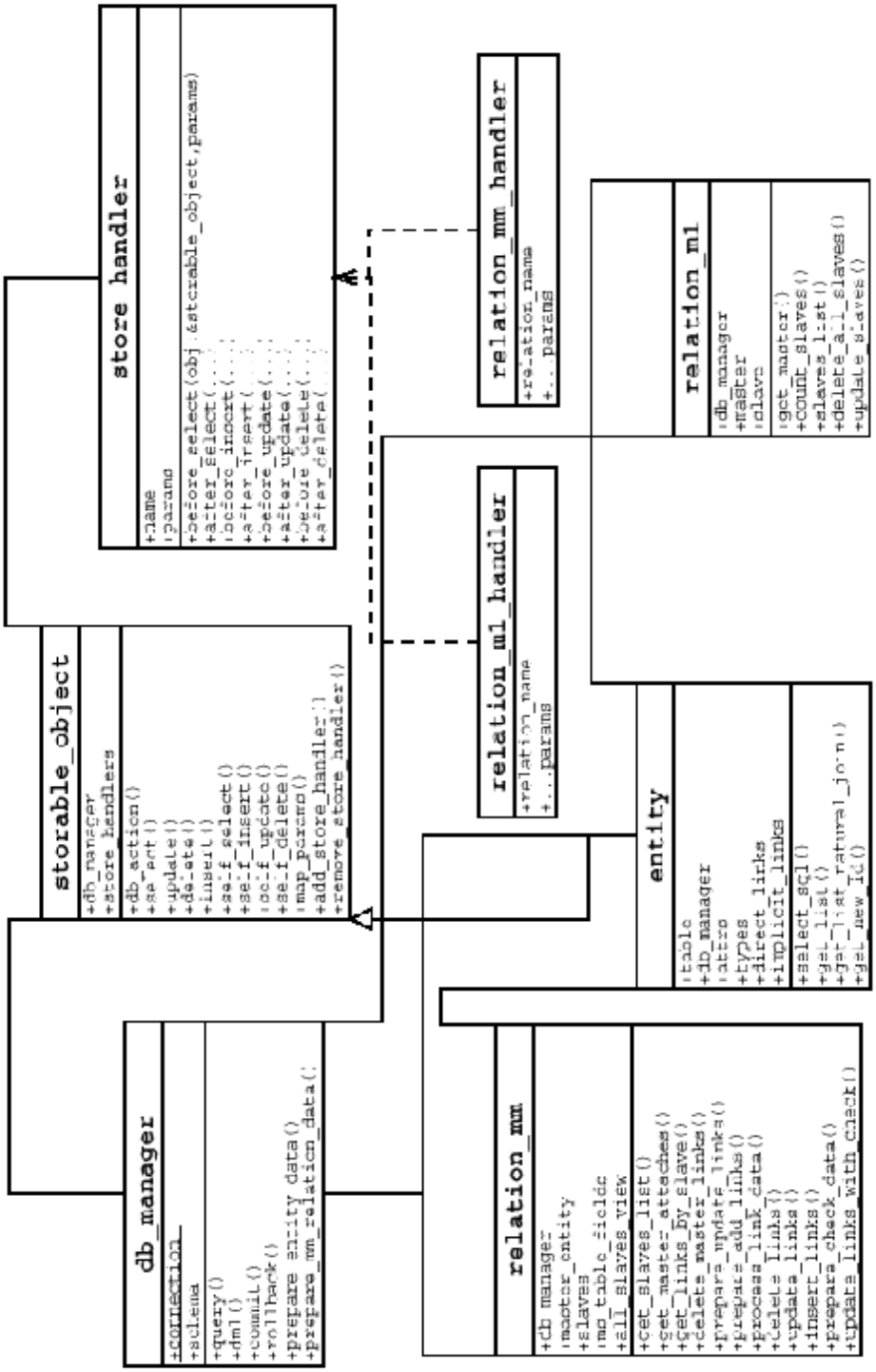
Пример реализации и использования ORM (PHP)

Два-в-одном:

- Реализация: учебный (упрощенный) пример ядра. Схема данных, менеджер базы данных, сущности, связи 1:m/m:n, обработчики CRUD-операций (логические триггеры уровня приложения);

- Использование: хранилище документов

Ядро: диаграмма классов



Учебная задача: хранилище документов

1. Редактирование и поиск документов по атрибутам;
2. Потребовалось добавить разделение материалов по степени подготовки (статус, 1:m);
3. Появилась необходимость более гибкого тематического объединения документов (корзина, m:n);
4. Организация полнотектового поиска по документам с учетом морфологии (интеграция со сторонним решением).

1: редактирование/поиск

```
$schema = array (
    ##############################
    'DOC' => array (
    ##############################
        TITLE_NAME => 'TITLE',
        FIELDS => array(
            'ID'=>NUMBER,
            'TITLE'=>VARCHAR,
            'ANONS'=>VARCHAR,
            'BODY'=>CLOB,
            'AUTHOR'=>VARCHAR,
            'SRC'=>VARCHAR,
            'DATE_REC'=>DATE,
            'PRIORITY'=>NUMBER,
            'ID_STATUS'=>NUMBER,
            'DATE_EDIT'=>DATE,
        ),
        PRIMARY_KEY => 'ID',
        AUTO_INCREMENT => 'S_DOC',
        DATE_FORMATS=>array(
            'DATE_REC'=>'dd/mm/yyyy hh24:mi',
            'DATE_EDIT'=>'dd/mm/yyyy hh24:mi',
        ),
        DONT_UPDATE => array('DATE_EDIT'=>1),
        NO_QUOTES => array('DATE_REC' => 1),
        DEFAULTS=>array(
            'STATUS'=>1,
            'PRIORITY'=>10,
        ),
        PREFERENCES => array (
            'STATUS' => 'ID_STATUS'
        )
    )
);
```

```
# конструктор внутри объявления класса
################################################
function article(&$db_manager) {
################################################
     $this0>entity('DOC', $db_manager);
};

$db_manager = & new db_manager($conn,$owner,$schema);
$art = & new article($db_manager);

# загузка по значению идентификатора
if (1 != $art->select(array($art->pkeyname => $id)) {
    # document not found
    ...
}



# поиск данных
$art_data = araay();
# из HTTP параметров
$sql_where = $art_controller->build_where($art->atts);
$n_rows = $art->get_list($art_data,
    array(WHERE_CLAUSE=>$sql_where,ORDER_CLAUSE=> 'datetime desc')
); # возможно решение с $arty->get_collction (массив объектов)

# обработка пользовательских операций, фиксация/откат
# action - DML: DELETE, UPDATE, INSERT ($id='new')
if (1 == $art->db_action($params,$action) && is_null($db_manager->error)) {
    if ($db_manager->connit()) { # OK
    ..
    } else {
        # например, не прошла проверка целостности
        # по "отложенным" до фиксации ограничениям
        ...
        $db_manager->rollback();
    }
} else { # ошибка отработки операции
    ...
    $db_manager->rollback();
}
```

2: редактируемый набор статусов

```
# добавляем в описание данных:
#####################################
'STATUS' => array (
#####################################
    TITLE_NAME => 'NAME',
    FIELDS => array(
        'ID'=>NUMBER,
        'NAME'=>VARCHAR,
    ),
    PRIMARY_KEY=>'ID',
    AUTO_INCREMENT=> 'S_STATUS'
)
#аналогично документам
$status = & new status($db_manager);
```

```
# если нам хочется автозагрузку статуса (title)
# решение через 2 последовательных запроса по ключам - переопределением
##############################################################################
function self_select (&$params) {
##############################################################################
    if(1 != parent::selfselect($params)) return NULL;
    if($this->id_status>0
      && 1 != $this->status->select(array(
      $this->status->pkeyname => $this->id_status)
      ))
    { # например, ошибка целостности
        ...
    }
}



# ...но если на странице редактирования нам нужен полный список,
# то автозагрузка необязательна и мы выбираем просто все статусы
$status->get_list();

# редактирование статуса: возможна ситуация, когда при удалении объекта
# необходимо "обнулить" поле статуса у соответствующих документов (NULL -
неизвестен)
# несколько вариантов:
# - переопределение метода self_delete
# - добавление стандартного обработчика relation_m1_handler
# - добавление внешнего обработчика - наследника store_handler
# relation_m1_handler
$this->docs_relation = & new relation_m1($this,$this->docs_entity,
$this->db_manager);
$this->add_store_handler (
    new relation_m1_handler('docs_relation',array(ON_DELETE=>SET_NULL))
); # теперь поддержкой целостности занимается обработчик
```

3: корзина - объединение документов

```
# добавление в схему описаний
#############################################################
'BASKET' => array (
#############################################################
    TITLE_NAME => 'NAME',
    FIELDS => array(
        'ID'=>NUMBER,
        'NAME'=>VARCHAR,
    ),
    PRIMARY_KEY => 'ID',
    AUTO_INCREMENT => 'S_BASKET'
),
```

```
#################################################################
'BASKET_DOC' => array (
#################################################################
    TYPE => RELATION,
    TITLE_NAME => 'NAME',
    FIELDS => array(
        'ID_BASKET'=>NUMBER,
        'ID_DOC'=>NUMBER,
    ),
    REFERENCES => array(
        'DOC' => 'ID_DOC',
        'BASKET' => 'ID_BASKET'
    )
)
# в конструктор класса basket:
# теперь поддержкой операций над связями корзины с документами
# занимается обработчик
$this->docs_relation = & new relation_mm
('DOC','BASKET_DOC',$db_manager);
$this->add_store_handler(new relation_mm_handler('docs_relation'));
```

4: интеграция с системой индексирования

```
# в конструктор класса документа:
$this->add_store_handler($new serarch_handler(
    array('TITLE'=>10,'BODY'=>5)
));
# новый обработчик
include_once('search_engine.php');
#########################################################
class search_handler extends store_handler {
#########################################################
var $fields_weight;

    ########################################################
    function search_handler ($fields_weight) {
    ########################################################
    parent::store_handler();
    $this->fields_weight = $fields_weight;
    }

    ########################################################
    function before_delete(&$obj,&$params) {
    ########################################################
    if(1 != delete_object(...) )
    {
        ... # дополнительно обработать ошибку
        return NULL;
    }
    return 1;
    }
```

```
# before_insert - аналогично before_update
##################################################
function before(&$obj,&$params) {
...
if (0== $status) {
    if (1 != add_object(...)) {
    ..
    return NULL;
    }
    return 1;
} else {
    if (1 != delete_object(...)) {
    ...
    return NULL;
    }
    return 1;
}
```

Выводы по примеру

Очевидные плюсы (поверхностный взгляд, не забудем, что пример - простой)

1. Минимум рутинного SQL (меньше ошибок, RAD)
2. Легко менять атрибуты (схема + шаблоны презентации)
3. Типовые связи реализуются по единой программе, легко добавить/убрать
4. Дополнительная гибкость при помощи единого механизма обработчиков. Код обработчика отделен от кода сохраняемого объекта, можно иметь отдельные библиотеки под ряд задач. Путем регистрации/дерегистрации обработчиков можно динамически менять поведение объектов.

Далее в этом докладе

- Пример классификации ORM (advanced feature list)
- Lasy loading и прокси-объекты, кэширование
- Плюсы и минусы метода, специфика веб-проектов

Пример сравнительного анализа различных ORM

www.c2.com/cgi/wiki?ObjectRelationalToolComparison (выборочно)

1. Возможность обойтись без «ручного» SQL-кодирования;
2. Поддержка различных РСУБД;
3. Поддержка отношений между объектами;
4. Поддержка `GROUP BY`, функций агрегации;
5. Степень поддержки LASY LOADING;
6. «Равенство» уникальных объектов (по ключу, по ссылочному циклу);

7. Поддержка составных первичных ключей, составных актрибутов объекта;
8. Поддержка 1:1, 1:m, m:n отношений;
9. Поддержка тернарных отношений;
10. Стратегии отображения наследования table-per-{hierarchy, concrete-class, subclass};
11. Возможность выборки связанных объектов с использованием `OUTER JOIN`;
12. Блокировка/версионирование (vesion numbers, timestamps,..);
13. Транзакции на уровне объекта;
14. Необходимость в дополнительных таблицах БД (мета-данные, блокировки, последовательности);
15. Поддержка кластеризации и обновременного доступа из различных приложений без потери целостности;
16. Требует генерации кода/обработки байт-кода;
17. Встроенная поддержка кэширования запросов;
18. Поддержка различных стратегий генерации значений уникальных ключей;
19. Отобранеие нескольк таблиц в один класс, нескольких классов в одну таблицу;
20. Журналирование всех SQL-запросов.

Прокси-объекты

Динамические прокси

m:n(коллекция):без использования прокси грузятся все подчиненные объекты, даже если мы в них не заинтересованы.вместо загруженной коллекции можно получить коллекцию прокси-объектов, которые, например, содержат только значения первичных ключей и пустую ссылку на реальный объект. Эти объекты «материализуются» только при дальнейшем обращении (механизм динамического делегирования запросов от прокси к материализованному субъекту).

Настройка динамических прокси — в схеме данных. Стратегии: материализация коллекций (m:n, CollectionProxy), материализация ссылок (1:m,ReferenceProxy), ...

Прокси разработчика

переопределение классов ядра, реализующих стандартные стратегии ленивой загрузки (MyCollectionProxy, MyReferenceProxy, ...)

Runtime SQL-building

- Проблемы: богатство SQL, нетривиальные модели, различные реализации поставщиков СУБД;
- 4 основных типа запросов: Criteria, OQL, named SQL, hand-coded. В большом проекте с большой вероятностью будет смесь из нескольких типов;

- Реализация объектного интерфейса к любым SQL-операциям в общем виде не приводит к упрощению, а порой наоборот;
- Путь OQL(Hibernate - HQL): from a where a.b.b_attr = ...

Основные плюсы подхода

- концентрация кода, относящегося к СУБД и описанию данных в отдельном(одном) месте;
- автогенерация рутинных запросов, унификация базовых стратегий отображения. единая концепция, меньше ошибок, возможность быстрого развертывания проекта в полу-рабочее состояние (большой плюс, т.к. разработка всегда итеративна);
- следствие (1): дополнительные возможности автоматизации разработки,отладки, тестирования (автогенерация кода, тестов по схеме данных — в ряде случаев удобно);
- относительный плюс: удобство объектной модели (радикальная версия, крайне неверная - «программисту проще с классами»).

Основные минусы подхода

- Богатство реляционных операций несозмеримо больше возможностей большинства ORM. PM имеет серьезную теоретическую базу, OM — нестрогие («инженерные», ненаучные), удобные человеку представления об объектах реального мира. неизбежны: доводка, смесь подходов (риск получить неуправляемый код);
- Решения на базе ORM сложно назвать «лёгкими» (дополнительная проблема скриптовых языков — компиляция);
- Члены команды должны «говорить на одном языке», а принципы ORM стратегии нетривиальны для полного понимания и эффективного использования — необходимы дополнительные затраты на обучение;
- Сложность унификации отображения для разных СУБД. из-за этого многие ORM имеют ряд неочевидных ограничений, которые могут быть замечены уже после выбора стратегии и начала работ;
- Для большинства «продвинутых» СУБД эффективнее бизнес-логику переносить в хранимые процедуры и триггеры (производительность может отличаться на порядок и больше). ORM как правило «навязывает» другую стратегию;
- Из-за специфического разделения кода «модули» в большом проекте должны удовлетворять особенным требованиям, особенно если необходима настраиваемая «сборка» из различных компонент.

Специфика веб-проектов

- Почти всегда bottleneck-база;.
- Для разных классов задач распространены свои механизмы кэширования (кэширование отрендеренных шаблонов(СМИ) вместо кэширования объектов(сообщества, персонифицированные веб-сервисы));

- Отсутствие толстых клиентов;
- Редко встречаются «долгоживущие» серверные приложения;
- Для многих проектов ER-модель фактически определет ТЗ на разработку, это хорошая база для успешного использования ORM (CMS frameworks: от разработки с нуля к настройке и внедрению);
- Часто в проектах есть существенное различие между «модераторской» и «пользовательской» частями (требования к нагрузке, «структурированность» интерфейсов). Можно осуществлять аккуратную смесь подходов.

Качественная оценка эффективности использования

Разделение проектов весьма условное.

(A) - ORM, (B) ─ «традиционные» методы

1. **Простой**: (B) ─ эффективно, (A) ─ нет (в ближайший ларёк на такси);
2. **Средний**: (A) может оказаться эффективным как минимум за счёт RAD-преимуществ;
3. **Сложный:** (A) может оказаться менее эффективным из-за сложной логики, которую ORM не поддерживает или делает это неэффективно с точки зрения производительности. существенную роль начинают играть организационные причины (единый ORM-слой накладывает ограничения на свободу взаимодействия внутри команды).

PHP - решения, перспективы

Propel, http://propel.phpdb.org (2004)

Meta Storage (MDB), http://www.meta-language.net (1999)

PEAR::DB_DataObjects, http://pear.php.net/package/DB_DataObject (2002)

PEAR::DB_QueryTool, http://pear.php.net/package/DB_QueryTool (2003)

Команда этого выпуска

Авторы докладов

- Михаил Курмаев [Demiurg];
- Виктория Резниченко [Апельсин];
- Кирилл Шепитко [Yamert];
- Денис Жолудов [DAN];
- Алексей Борзов [SadSpirit], avb@php.net;
- Алексей Рыбак [fisher];

Редакционная коллегия, корректировка

- Александр Смирнов [PHPclub];
- Елена Тесля [Lenka];
- Александр Войцеховский [young];
- Антон Чаплыгин;
- Андрей Олищук [nw], координатор проекта PHP Inside, г. Коломна;
- Александр Ширяев [Dallas];
- Дмитрий Попов;
- http://phpclub.ru

Подготовка обложки, верстка

- Денис Зенькович;
- Алексей Володин [kosmonavt], http://kosmo.wmaster.ru, г. Мурманск
- Андрей Олищук [nw].

Спасибо всем участникам проекта!

Координаты журнала:

http://mag.phpclub.ru

project@yugovostok.ru

Обсуждаем номер на

http://phpclub.ru/talk

Для заметок

(Сюда можно заносить свои примечания)