

PHPInside

электронный журнал для веб-разработчиков



PHPInside Art Contest

Результаты конкурса (часть 2)



В фокусе	
Теория разработки framework-систем.....	5
Динамический генератор функций.....	18
Транзакции – головная боль или крылья?.....	24
Безопасно-ориентированное программирование.....	30
Изменим жизнь к лучшему с помощью буферизации вывода.....	35
Идеи	
Шаблонизация на XSLT. Приемы и примеры.....	42
Форум	
Работа над ошибками. Подход №1.....	49
Работа над ошибками. Подход №2 (PHP5).....	50
Постраничный вывод данных из MySQL.....	52
Вывод данных из файла «через строку».....	52
План врезок.....	53

Конференция в Киеве

PHPClub (<http://phpclub.ru>) совместно с компанией Миротел (<http://mirotel.net/>) и редакцией журнала «PHP Inside» (<http://www.phpinside.ru/>) проводят **4-ю международную конференцию «Современные технологии эффективной разработки веб-приложений с использованием PHP»**.

Киев, Украина 12-13 мая 2005 г.

<http://www.phpconf.ru/2005/>

Наши цели

Конференция, проводимая под эгидой PHP-Клуба в последние два года, является уникальным мероприятием, собирающими со всего постсоветского пространства ведущих веб-программистов, PHP-энтузиастов и других талантливых программистов, чья профессиональная деятельность в той или иной степени связана с веб-технологиями.

В рамках конференции у вас будет отличная возможность познакомиться с современными тенденциями разработки веб-приложений, обсудить со специалистами интересующие вас вопросы, найти оптимальные решения для вашего бизнеса, да и просто пообщаться с единомышленниками в теплой и неформальной обстановке. Предыдущие PHP конференции показали, что материалы, информация и опыт, полученные на конференции, участники с успехом применяют на практике, значительно повышая эффективность своей работы.

Команда номера

Авторы и переводчики

Денис Баженов
Дмитрий Заболотский
Александр Косарев
Борис Вольфсон
Михаил Мотыженков
Александр Календарев

Редакционная коллегия

Александр Смирнов
Александр Войцеховский
Андрей Олищук [nw]
Антон Чаплыгин
Дмитрий Попов
Елена Тесля

Выпуск номера

Андрей Олищук [nw]
Антон Чаплыгин
Денис Зенькович

Контактные данные

<http://phpinside.ru>
nw@phpinside.net

Место проведения

Киев - величественная красота его золотых куполов, раскинутых на правом берегу Днепра среди цветущих каштанов, которые каждую весну превращают город в сказочное королевство. Приглашаем Вас своими глазами увидеть достойный удивления и восхищения неповторимый облик Киева.

Тематика конференции включает следующие направления

- PHP 5
- PHP & базы данных
- PHP & системы масштаба предприятия
- PHP & XML
- Ядро PHP и разработка расширений
- Производительность веб-приложений
-

Труды конференции будут опубликованы в виде полных текстов принятых статей в специальном номере журнала PHPInside.

Время и место проведения конференции

12-13 мая 2005 года. г. Киев.

КиевЭкспоПлаза, павильон №2, зал №1

Представление докладов

Рабочий язык конференции - русский и английский, представление докладов возможно на любом из них. Тезисы докладов представляются в программный комитет на русском или английском языке электронной почтой по адресу 2005@phpconf.ru в формате RTF.

Мы надеемся, что тезисы будут давать достаточно полное представление о содержании предлагаемого доклада. Мы не ограничиваем вас перечисленными выше темами и готовы рассмотреть любые интересные доклады, так или иначе связанные с разработкой веб-приложений.

Важные даты:

- 7 февраля - Окончание приема тезисов докладов
- 11 февраля - Утверждение программы конференции, извещение о статусе доклада (принят/не принят)
- 25 апреля - Окончательный срок сдачи материалов для включения в информационный справочник конференции

Стоимость

Стоимость регистрационного взноса: 560 грн. (3000 руб.)

В стоимость регистрационного взноса входит:

- Возможность участия во всех мероприятиях конференции (2 дня);
- Получение раздаточных материалов;
- Кофе-брейки и обеды;
- Участие в лотерее участников конференции;
- Участие докладчиков в конференции - бесплатно (+ небольшой бонус).

Внимание! Действует система скидок "early-bird" - для тех, кто оплачивает до 1 апреля 2005 года, регистрационный взнос составляет 500 грн. - 2600 руб.

Увидимся в Киеве!

Орг.комитет конференции

WWW: <http://www.phpconf.ru/2005/>

E-mail: 2005@phpconf.ru

Тел.: +380 44 494 03 50

Факс: +380 44 494 03 51

Председатель оргкомитета: Андрей Зинченко

Информационный партнер в Москве: Александр Смирнов 7-095-783-2659

Теория разработки framework-систем

Существует вполне оправданное мнение, что все хорошие программисты должны быть чуточку ленивы, дабы воплощать в жизнь идеи, которые смогут снизить количество труда, необходимое для реализации какой-либо конечной цели. Так было всегда. Человечество и теперь не хочет сходить с протоптанной дорожки (не только программисты). Вопросы, рассматриваемые в данной статье, предоставят начинающим программистам вектор изучения методов проектирования веб-приложений, а опытным – пищу для ума и возможность сделать свою жизнь и жизнь своих коллег проще.

Задумываясь о вопросах снижения сроков разработки веб-приложения и увеличения отказоустойчивости кода, на ум приходит только **PEAR**. Немного подумав, вспоминаются такие понятия, как **ADODB**, **Smarty**, **XML**. Затем более фундаментальные и теоретические вещи: шаблон проектирования **MVC** и framework-системы.

Наверное, не стоит объяснять вам смысл последнего термина, но следуя хорошим традициям, изучение любой предметной области необходимо начинать с определения используемых понятий. Итак, под framework-системой понимается набор средств языка, логически объединенных в один пакет и выступающих в качестве основы (каркаса) для создания приложений любой сложности.

Но не все так просто. То обилие framework-систем, которое сейчас существует, и их сложность не дает сделать однозначных выводов по поводу их полезности. К сожалению, большинство из них не обладает нормальной документацией (хотя бы английской), а остальные представляют собой что-то дикое, написанной явно не для «ВСЕХ», а для «СЕБЯ» или «СВОИХ».

В итоге я пришел к пониманию того, что если на свете и существует относительно простая, документированная и достаточно абстрактная framework-система, то от меня ее прячут по непонятным мне причинам. Да, есть такие системы, как **CVPFramework** и **php-MVC**. Да, возможно они кем то удачно используются, но, как оказалось, эти системы являются скорее попыткой переноса методологии **Java** в **PHP**. Я считаю, что такое скрещивание неприемлемо, так как эти два языка хоть и работают на одном поприще, но работают они по-разному и обладают разными средствами (например, **Java** имеет гораздо более «продвинутой» объектную модель по сравнению с **PHP**).

Переходя более конкретно к описанию системы, надо все таки определиться с задачами этой системы. Совершенно понятно, что основная задача состоит в ускорении процесса создания веб-приложений, но это это не единственная цель.

Автор:

Денис Баженов [.sid]

denis.bazhenov@vvsu.ru



Тем временем...

Если при работе с PostgreSQL возникла необходимость оставить доступ к таблицам только посредством хранимых процедур, то следует создать функцию под пользователем с правами администратора с опцией SECURITY DEFINER («Определитель безопасности»).

При этом, для простых пользователей необходимо закрыть доступ к нужным таблицам, но оставить права для выполнения функций.

В PHP существует функция `array_flip($arr)`, основная задача которой – менять местами ключи и значения массива (т.е. Ключи становятся значениями, а значения – ключами). Однако особенностью этой функции является отбрасывание повторяющихся значений. Это свойство может пригодиться при обработке поисковых запросов. Например:

```
$search = «we are you are»;
$words = explode(" ", trim($search));
$words = array_flip($words);
```

Результат:
Array ([we] => 0 [are] => 3 [you] => 2)

Применим еще раз:
\$words = array_flip(\$words);

Получим:
Array ([0] => we [3] => are [2] => you)

При удачном инжиниринге системы она может и должна выполнять следующие задачи:

1. Обеспечение единого стандарта разработки веб-приложений, а это приводит к более легкому сопровождению и отказоустойчивости приложения;
2. Предотвращение утечек технической информации;
3. Обеспечение адекватного реагирования приложения на непредвиденные ситуации;
4. Централизация управления приложениями (прозрачный для приложения перевод текста из одной кодировки в другую, сжатие данных перед отсылкой их пользователю и т.д.)

За счет чего это достигается? Все веб-приложения, какова бы ни была их спецификация, требуют определенной базовой функциональности для выполнения своих задач. К такой функциональности можно отнести получение данных от клиентов (методом **GET** или **POST**), получение результатов запросов к базе данных и вывод данных пользователю.

Два из трех этих этапов отлично вписываются в паттерн **MVC**. Если приложение должно обеспечивать хоть какой-то уровень безопасности, то к функциональности добавляется обязательная проверка данных на соответствие определенному типу или экранирование определенных символов.

В реальных веб-приложениях таких условий не одно и не два, а писать код заново – это очень непродуктивно, хотя иногда полезно. Повторному использованию кода мы все уже научились, поэтому я пропущу этот вопрос. Скажу только, что framework-системы способствуют этому процессу.

При проектировании таких систем нужно понять одну простую истину, в которой заключается специфика веб-приложений

Framework-система не является пакетом модулей для выполнения любых прихотей программиста. Наоборот, она создана для написания этих модулей программистами. Framework-система является ключевым звеном ваших приложений, поэтому она не должна быть большой и прожорливой, она должна быть лаконичной, быстрой, абстрактной!!! На самом деле последнее требование противоречит первым двум. Программист должен найти оптимальный баланс между этими двумя требованиями.

Устройство framework-систем

Что же из себя представляет framework-система «в разрезе»? С технической точки зрения, это набор скриптов, которые контролируют выполнение приложения и обеспечивают некий общий стандарт кодирования. Логически система разделена на ядро, загрузчик ядра, библиотеку общих функций и хранилище конфигурации.

Тем временем...

Директива Options Multiviews в `httpd.conf` заставляет веб-сервер (apache) абстрагироваться от расширения и искать любой файл с именем 123.

Другими словами, даже при отключенном `mod_rewrite`, по запросу: <http://localhost/123> может быть выдан файл <http://localhost/123.php> если конечно он существует в этой директории.

Для проверки данных на соответствие какому либо критерию может пригодиться функция `array_filter($arr,$func)` Она применяет к каждому элементу массива `$arr` функцию `$func` (передавая значение элемента в качестве аргумента функции).

К примеру, если необходимо из массива `$_POST` выкинуть все пустые элементы и элементы состоящие только из пробелов, то можно применить следующий код:

```
function clean($arg)
{ return strlen(trim($arg)); }
$post = array_filter($_POST, 'clean');
```

В большинстве случаев это обыкновенный php-скрипт с конфигурационными переменными. Любители XML могут представить конфигурационное хранилище в виде XML-файла. Такая схема используется в некоторых системах, но я считаю, что это не самый лучший подход.

Эта статья будет посвящена в основном проектированию ядра подобных систем. Процесс создания остальных компонентов не вызывает особого труда, но вкратце мы его рассмотрим. Зародыш загрузчика ядра нашей системы в простейшем виде можно представить вот таким листингом:

```
<?php
// filename: index.php (загрузчик)
// Этот файл находится в папке /www

// Подключаем ядро системы
include('./framework.lib.php');
$file = $_GET['file'];
$app = $_GET['app'];
$file_name = './'.basename($file).'/'.$.basename($app);
if ( file_exists($file_name) )
{
    include($file_name);
    // Здесь можно выполнять постпроцессорную обработку
}else{
    // Данный файл не существует на сервере i.e. HTTP/404
    echo "Запрашиваемый вами файл не найден.";
}
// Завершение работы системы
exit;
?>
```

Если теперь произвести запрос следующего вида:

```
http://www.server.com/?app=guestbook&file=index.php
```

то ваш скрипт загрузит на исполнение файл /www/guestbook/index.php, если таковой существует. Если потратить пару минут на конфигурацию Apache, то, используя модуль mod_rewrite, можно будет выполнить данный запрос в более лаконичном виде, который гораздо понятней пользователю:

```
http://www.server.com/guestbook/index.php
```

В таких рамках выполняются все приложения. Плюсы такой централизации очевидны. У всех приложений существует один хозяин, с помощью которого вы можете контролировать работу приложений. Например, можно централизованно хранить имя сервера баз данных, либо лимитировать доступ клиентов к приложению на основе единой схемы прав доступа.

В вышеприведенном листинге показана система, которая принимает решение о загрузке того или иного приложения, то есть, приложение работает в рамках системы. Существует еще один подход к построению таких систем.

Он заключается в том, что вы включаете код системы в ваши приложения, то есть система выполняется в рамках приложения. По понятным причинам этот подход использовать не стоит. Он ограничивает возможности и централизацию системы.

Тем временем...

Требования к framework-системам

Так какие задачи должно выполнять ядро системы? Давайте попытаемся сформулировать перечень требований к нашей системе:

1. Наличие инфраструктуры предназначенной для использования различных модулей с возможностью их замены. В одном этом требовании формулируется вся наивная простота и большая сложность framework-систем;
2. При использовании каркасной системы приложения должны сохранять высокий уровень индивидуальности, несмотря на централизацию, которая характерна для системы;
3. Приложения, написанные без использования Framework-системы, должны легко интегрироваться с ней.

Вот уже и заметны противоречия, завязанные на централизованность. В этом заключается еще одна проблема проектирования таких систем – необходимо определить оптимальный уровень интеграции системы в код подвластных ей приложений.

Мы уже оговорились, что эта система предназначена для обеспечения базиса для написания любого приложения (будь то гостевая книга или интернет-магазин), то есть в первую очередь она предназначена для программистов.

Выше приведенный перечень требований предъявляется к универсальной framework-системе, то есть системе, под управлением которой одновременно работает более одного приложения. Индивидуальные каркасные системы работают только с одним приложением одновременно, следовательно, вторым требованием из нашего списка можно пренебречь. Однако такие системы показывают более низкий уровень адаптации для приложений «внешних» по отношению к системе (то есть не выполняется третье требование).

Проектирование framework-системы и ее составных частей

При проектировании framework-системы необходимо учитывать несколько факторов:

1. Вы ничего не знаете об операционной системе, на которой будет выполняться система;
2. Вы ничего не знаете о приложении, которое будет выполняться на вашей системе;
3. Вы ничего не знаете о компонентах, которые будет использовать конечный пользователь вместе с вашей системой.

Писать скрипты в стиле `register_globals = On` (когда переменные GET и POST не было необходимости извлекать из соответствующих массивов) можно и при установке этой опции в Off. Для этого в начале скрипта достаточно написать (к примеру): `extract($_POST);`

Теперь переменные переданные методом POST будут доступны не только как `$_POST[«var»]`, но и просто как `$var`.

Это приводит в некоторое замешательство, но не стоит отчаиваться. Этот перечень не принесет вам больших неприятностей, если вы будете во всем придерживаться абстрактности. Абстрактность подразумевает увеличение масштабируемости системы, и, как следствие, некоторое ее усложнение.

Система как таковая не должна вмешиваться в работу приложений. Ее работа состоит в инициализации окружения сервера. Затем управление отдается приложению, а после того, как приложение выполнится, система производит постпроцессорную обработку, отправляет результат клиенту и завершает свое выполнение. Эта последовательность хорошо отражена на листинге №1.

Двигаясь дальше, необходимо конкретизировать API системы (ее ядро). Основные функции делятся на:

1. Функции работы с модулями;
2. Функции инициализации системы;
3. Функции обеспечения безопасности выполнения;
4. Вспомогательные функции (функции верификации клиента, функции работы с внешними данными и др.).

Я не зря на первое место поставил функции работы с модулями. В этих функциях скрыт весь смысл работы системы. Эти функции должны загружать модуль и делать его код доступным для использования. Однако вы должны помнить, что приложения, написанные без использования вашей системы, должны легко с ней работать, поэтому в большинстве случаев работа данных функций ограничивается оператором `include`. Те модули, которые написаны специально для вашей системы, могут содержать определенную информацию, которую сможет понять ядро. Простейший пример модуля приведен на листинге ниже:

```
<?php
// filename: /www/modules/gb.module.php
$_MCFG['NAME'] = 'GB module for FrameworkSystem';
$_MCFG['VERSION'] = 0.5;
$_MCFG['AUTHOR'] = 'Author <author@server.com>';

class GuestBook
{
    // Код класса
}
?>
```

Наличие в модуле информационной структуры (ассоциативного массива `$_MCFG`) не обязательно, но при ее присутствии она может быть использована приложениями. Ниже приведен примерный листинг приложения, которое использует этот модуль.

```
<?php
//filename: /www/guestbook/index.php
if ( !load_module('gb') ) {
    echo "Unable to load module";
    return 0;
}
```

Продолжение на следующей странице.

Абстрактность подразумевает увеличение масштабируемости системы, и как следствие – некоторое ее усложнение

```
$info &= get_module_info('gb');
if ( isset($info['VERSION']) || $info['VERSION'] < 0.5 )
{
    echo "У вас слишком старая версия GB";
    return 0;
}
$gb &= new GuestBook;
// Код приложения
?>
```

Окончание листинга. Начало на предыдущей странице.

Любители объектного подхода могут создать класс примерно с таким прототипом:

```
<?php
class FrameworkModule
{
    var $moduleName;
    var $moduleVersion;
    var $moduleAuthor;

    function FrameworkModule()
    {
        // Здесь может содержаться код регистрации модуля
        // или другой служебный код.

        // Не забывайте вызывать эту функцию, если вы
        // используете этот класс в качестве базового.
    }
}
?>
```

Класс подобного типа может использоваться в качестве хранилища информации о модуле.

Допустим, с модулями понятно, а как же быть с пресловутым паттерном MVC? Как реализовать абстрактность от базы данных? Как получить в свое распоряжение хороший шаблонный движок? Ответы на все эти вопросы выходят далеко за рамки данной статьи. Скорее всего, позже, в курсе статей, я уделю внимание данным темам, которые очень актуальны на сегодняшний день (особенно последняя). В рамках же нашей статьи необходимо рассмотреть совсем другой вопрос.

Не то, какой механизм абстракции необходимо использовать, а то, как стать независимым от данных механизмов абстракции. Как бы это не было смешно, но такой подход к проблеме имеет право на жизнь, потому что иногда приходится менять слои абстракции по каким-то причинам. Вообще, как механизм абстракции, так и шаблонный движок в состав ядра не входят и считаются обыкновенными модулями.

Наше решение позволит менять, например, ADOdb на PEAR::DB без изменения кода.

Для этого необходимо создать абстрактный класс подобного рода:

```
<?php
class Framework_DB_Common
{
    function connect($host, $user, $pass);
    function select_db($db_name);
    function &exec_query($sql);
    function start_transaction();
    function rollback_transaction();
    function commit_transaction();
    function get_current_db();
    function get_affected_rows();
    function create_database($db_name);
    function drop_database($db_name);
    function get_last_error_code();
    function get_last_error_description();
    function exec_sql_file($file_name);
    function server_version();
    function dbal_version();
    function disconnect();
};
?>
```

Этих методов вполне достаточно, чтобы организовать работу с любой реляционной (и не только) БД, так как, независимо от типа СУБД, они позволяют выполнить практически все вышеперечисленные действия.

Конечно, не все базы данных поддерживают транзакции, но соответствующие методы будут перегружены в дочерних драйверах, которые ответственны за конкретный механизм абстракции от БД, либо в классах прямого доступа (рассмотрим далее). А это значит, что в нашу задачу входит только унификация методов доступа. То есть анализ возвращаемого значения из механизма абстракции и приведение его к одному стандартному набору значений.

Например, в случае ошибки некоторые механизмы абстракции могут возвращать значение -1. В драйвере вам необходимо подменить это значение на что-то свое, заранее установленное. Например, заведите для этих целей константу DB_ERROR_FAILED. Данная схема не нова, так что за основу вы можете взять соответствующие значения констант из PEAR::DB. В своей практике я часто встречал людей, которые очень критиковали такой подход к работе с БД, мотивируя это тем, что существенно снижается производительность.

Да, производительность снижается из-за наличия еще одного промежуточного класса, но слово «существенно» имеет очень плавающие рамки, и от проекта к проекту эти рамки меняются очень сильно. Объектная модель в PHP довольно слабая. И с точки зрения производительности, функциональный подход, конечно, быстрее. В принципе, с этой точки зрения ситуация не особо изменилась с выходом PHP/5, даже несмотря на ссылки, используемые при присвоении объектов. Именно поэтому в ядре я специально не использовал классы. Но в данном случае, если это и зло, то зло необходимое. Задача объектно-ориентированной модели (естественно, не единственная) как раз и состоит в развитии повторного использования кода путем наследования классов.

Совсем другая ситуация с классами прямого доступа. Они наследуются прямо от класса «Framework_DB_Common». Производительность при работе с ними выше. Правда, их для начала придется написать. Для одной-двух самых часто используемых БД я советую так и сделать. Благо, это задача не из сложных.

Аналогичная операция применяется для классов «Framework_DB_Result_Common» и «Framework_DB_Row_Common». «Framework_DB_Result_Common» является представлением ресурса результатов запроса. «Framework_DB_Row_Common» представляет собой строку результирующего запроса. Чем хороша такая схема представления данных? Рассмотрим такой пример:

```
<?php
/*
 * $db class Framework_DB_MySQL
 * $result class Framework_DB_Result_MySQL
 * $row class Framework_DB_Row_MySQL
 */
$db = DB_connect('mysql://user:pass@localhost/test');
$result = $db->exec_query("SELECT `id`, `car_name`, `cost` FROM
`cars`");
while ( $row = $result->next() )
{
    if ( $row->data['cost'] > 10000 )
    {
        $row->data['cost'] -= $row->data['cost']*0.05;
        $row->update_entry();
    }
}
$db->disconnect();
?>
```

Этот скрипт извлечет из таблицы с автомобилями все записи, а затем в цикле для тех, чья цена выше \$10 000 сделает новогоднюю скидку 5% и занесет результаты обратно в таблицу. Естественно, эту задачу можно было бы решить вообще одним запросом к базе данных. На этом примере я просто показал, как можно легко обрабатывать данные, в том случае, когда результат обработки данных невозможно вычислить на этапе самой обработки, либо этот результат не обладает прямой зависимостью от имеющихся данных. Попросту говоря, когда нам все равно, приходится извлекать данные из таблицы и проверять их вручную по каким-то причинам. Это своеобразный менеджер автоматизации управления данными.

Метод `update_entry` генерирует соответствующий SQL-код и выполняет его. Здесь нет никаких препятствий, так как имена полей ему известны, а имя таблицы мы можем получить, наследуя класс «Framework_DB_Row» от другого класса (назовем его «Framework_DB_Automation_Layer»), в котором будет указываться имя таблицы и ссылка на класс, который представляет собой соединение с базой данных (через него мы будем выполнять запрос), в этом классе как раз и находится метод `update_entry`. Генерация SQL-кода – вопрос, который не вписывается в нашу статью, однако скажу, что методы генерации ограничены вашей фантазией и требованием поддержания целостности данных и индексов.

Можно в наш класс добавить метод `get_entry`, который бы занимался извлечением данных на основе переданных ему индексов или других критериев поиска. Или, скажем, метод `delete_entry`, задача которого – удалять данные из таблицы, подходящие под определенные критерии. Общее дерево классов можно представить в следующем виде:

```
Framework_DB_Common
|
+-Framework_DB_Driver_PEAR
+-Framework_DB_Driver_ADO
+-Framework_DB_MySQL
+-Framework_DB_MsSQL
+-Framework_DB_Oracle

Framework_DB_Result_Common
|
+-Framework_DB_Result_Driver_PEAR
+-Framework_DB_Result_Driver_ADO
+-Framework_DB_Result_MySQL
+-Framework_DB_Result_MsSQL
+-Framework_DB_Result_Oracle

Framework_DB_Automation_Layer
|
+-Framework_DB_Row
```

Листинг 1. Дерево классов типичного механизма абстракции

Дочерние классы, которые имеют в своем имени слово «Driver», являются драйверами к какому-либо механизму абстракции. Все остальные являются классами, представляющими какую-либо конкретную БД, которые написаны вами (классы прямого доступа). Осталось написать функцию, которая будет создавать экземпляр соответствующего класса, и наш механизм абстракции от механизмов абстракции БД готов (извините за каламбур).

Аналогичная схема используется для внедрения абстракции от шаблонизаторов (шаблонных движков).

Следующий вопрос, который необходимо затронуть, это инициализация системы перед запуском приложений. Это одна из самых ответственных этапов выполнения системы, потому как инициализацией вы должны добиться определенного уровня единообразия окружения. Это позволяет более легко строить приложения, привыкая к определенному окружению, а также более быстро отлавливать ошибки в приложениях. Под окружением следует понимать набор переменных и конфигурационных опций интерпретатора. Следует настроить их так, чтобы они обеспечивали максимальную эффективность и безопасность выполнения. К этим опциям можно отнести:

1. Уровень сообщения об ошибках по умолчанию;
2. Установки «магических кавычек» (magic quotes);
3. Установки регистрации глобальных переменных;
4. Технические настройки компилятора («timelimit», «memorylimit»).

Этот список можно продолжить, но лучше загляните в файл «php.ini» и продумайте, какие опции вам необходимо стандартизировать. Необходимо помнить, что если интерпретатор выполняется в safe mode, то он скорее всего не позволит вам сменить некоторые опции, например, «timelimit».

Также на стадии инициализации желательно определить некоторые общедоступные переменные:

1. Реальный IP-адрес клиента (если он использует прокси сервер);
2. Режим выполнения системы (normal, debug – рассматривается ниже);
3. Имя используемого интерпретатором интерфейса («cgi», «sapi», «apache2handler», «apache2filter», «nsapi» и т.д.);
4. Тип используемой операционной системы

Этот список тоже можно продолжить и включить в него, скажем, версию интерпретатора. Но следует помнить, что эти переменные будут вычисляться при каждом клиентском запросе. Поэтому этот список должен быть кратким и состоять из самых необходимых вещей. Кроме того, на этой стадии определяются необходимые системные константы ядра.

Также за счет стадий инициализации и деинициализации можно реализовывать различные служебные функции. С использованием функции microtime можно написать элементарный профайлер системы, который будет подсчитывать примерное время выполнения приложения и ключевых этапов работы системы, снабжая вас информацией о производительности вашего приложения и каркасной системы в целом.

Стадия деинициализации находится сразу за стадией выполнения постпроцессорной деятельности и предназначена для подготовки системы к завершению.

Функции безопасности выполнения предназначены для обработки ошибок и исключительных ситуаций, возникающих при работе приложения. От всего защититься невозможно, но утечку информации с сервера мы должны предотвратить. Делается это установкой своего обработчика ошибок (PHP-функция set_error_handler), логика которого будет зависеть от режима работы системы (normal/debug). Этот режим можно устанавливать, основываясь на IP-адресе клиента. Если адрес входит в доверенный диапазон, то система запускается в режиме debug, и в случае возникновения ошибки, предоставляет всю информацию о произошедшей ситуации. Если нет, то система ограничивается лаконичным «HTTP 500», или приятным сообщением с просьбой обратиться к администратору. Хендлер ошибок должен также при необходимости вести лог ошибок, или, возможно, даже дампы ключевых системных переменных и переменных, находящихся в области видимости на момент возникновения ошибки. Это может быть необходимо для дальнейшей отладки приложения.

Подходя абстрактно к этой задаче, я бы предложил следующую схему:

```
<?php
function __error_handler($err_no, $err_str, $err_file, $err_line,
    $err_scope = NULL){}
function __eh_mail($error_info){}
function __eh_log($error_info){}
function __eh_dump($error_info){}
?>
```

Первая функция является как раз хендлером, а все остальные являются фильтрами вывода (передача информации по почте, запись в лог, запись дампа соответственно) и вызываются хендлером при необходимости выполнить соответствующую операцию. Можно создать отдельную функцию генерации исключительных ситуаций, вместо PHP-функции `trigger_error`.

Также нелишним будет в хендлере анализировать состояние стека интерпретатора, используя функцию `debug_backtrace`, дабы определить точное место возникновения ошибки.

Это необходимо, например, в ситуации, когда ошибка произошла в методе класса абстракции от БД из-за передачи ему неверных параметров. В данном случае физически ошибка произошла не в самом методе, а логически в месте, где произошел вызов этого метода с неверными параметрами. Естественно интерпретатор выдаст информацию о «физической» ошибке, которая, как таковая, для вас абсолютно бесполезна.

При отсутствии проверки возвращаемых значений функциями, такие ошибки очень трудно отловить в больших приложениях!

Следующая и последняя часть ядра – это вспомогательные функции. Они призваны обеспечивать базовую функциональность работы с данными. Сюда относятся функции получения данных и обработки данных. Разберем прототип следующей функции:

```
function rvar($name, $def=NULL, $scope=NULL, $type=NULL, $func=NULL){}
```

Эта функция получает данные из ассоциативных массивов `$_GET`, `$_POST`, `$_COOKIE`, `$_SESSION` и приводит полученные данные к определенному типу.

```
<?php
$file = rvar('file', 'index.php', SC_GET|SC_POST, TP_STRING,
    "basename", "strtolower");
$id = rvar('id', 0, SC_SESSION, TP_UINT);
?>
```

В этом листинге мы получаем две переменные. Первую переменную «file» мы получаем методом GET либо POST, если ее не существует, используем значение «index.php», а если существует, то обрабатываем значение функциями «basename» и «strtolower». Во втором случае мы получаем переменную «id», приводим ее к типу «целое беззнаковое» или используем по умолчанию значение 0. В виде такой функции мы имеем фильтр вводных данных, который в какой-то степени делает наше приложение более безопасным.

Следующая пара функций предназначена для работы с «магическими кавычками»:

```
<?php
function mq_encode(&$data) {}
function mq_decode(&$data) {}
?>
```

Эти функции обрабатывают данные с помощью PHP-функций `addslashes/stripslashes` соответственно. Однако у этих функций (`mq_encode/mq_decode`) есть одна особенность. Они предотвращают повторную обработку данных в том случае, когда они уже обработаны при помощи «магических кавычек» PHP.

Как и все предыдущие, этот список нуждается в расширении. Не следует только перенасыщать его, во избежание падения производительности из-за абсолютно ненужной функциональности.

Подводные камни

Какие проблемы возникают при написании подобных систем? На самом деле написание такой системы, так чтобы она являлась настолько абстрактной, чтобы на ней выполнялись любые приложения (не только те, которые вы будете писать, но и уже написанные другими людьми), а также настолько удобной в программировании, чтобы сократить количество кода ваших приложений на порядок – дело не легкое, потому что эти два требования исключают друг друга. Поэтому приходится искать оптимальное соотношение конечных качеств продукта. Ваша задача как программиста – как раз найти это оптимальное соотношение и воплотить его в жизнь. В зависимости от ваших условий и требований, можно в какой-то степени пожертвовать тем или иным качеством системы. Однако существуют подходы, которые лучше не использовать при создании framework-систем. Все же применение таких подходов иногда встречается.

Например, система `CVPFramework` имеет такую особенность. В ней сделали переопределение всех стандартных типов данных в виде классов (`String`, `Float`, `Integer` и так далее). Метод увеличения высокоуровневости языка, который применяется в `.NET Framework`, используется во многих языках. По отношению к веб-приложениям этот подход имеет ряд недостатков. Во-первых, при использовании объектной модели страдает производительность. Во-вторых, так как в PHP отсутствует перегрузка операций, то производить математические операции напрямую над таким классом вы не сможете. В-третьих, передача таких аргументов стандартным функциям тоже затруднена, потому как операторы приведения к типу в PHP отсутствуют.

В ZendEngine2 присутствует один оператор приведения к типу. Это функция “_toString”, которая, по сути, приводит класс к строке в том случае, если в контексте использования объекта необходима строка.

Перед тем как использовать объектную модель в ключевых компонентах системы, критичных ко времени выполнения, задумайтесь над тем, насколько она вам там нужна

Перед тем, как использовать объектную модель в ключевых компонентах системы, критичных ко времени выполнения, задумайтесь над тем, насколько она вам там нужна. Объектная модель хорошо подходит для реализации прикладных задач, где я и советую ее применять. Для создания ядра framework-системы лучше использовать функциональный подход.

Какой подход использовать, решать вам. Тем более некорректно говорить, что объектный подход слишком медленный, так как точного числового определения слова «слишком» не существует. Но имейте в виду, что объектный подход медленней функционального. А мы должны максимально использовать все возможности языка для достижения конечной цели, в том числе и производительности.

Но на самом деле самая большая сложность, которая встает перед разработчиком, – это абстрактность системы от окружения. То множество неизвестных условий говорит о том, что мы не можем прогнозировать ситуацию, в которой будет выполняться наша система. Поэтому необходимо тщательно продумывать стадию инициализации системы, которая сглаживает индивидуальные особенности различных платформ. И, конечно же, необходимо тестировать продукт на различных операционных системах. Вопросы платформозависимости необходимо учитывать еще при проектировании системы. Например, в качестве разделителя пути всегда используйте символ прямого слеша, который работает как на операционных системах *nix, так и на операционных системах семейства WindowsNT. Таких нюансов не один и не два, так что рассматривать их все смысла не имеет.

Заключение

Надеюсь, что статья хоть немного пролила свет на эту довольно темную и пока только развивающуюся нишу интернет-программирования. Темную потому, что конкретной документации по этой теме нет, как и качественных систем. Я знаю, что многие люди пытаются писать свои каркасные системы, и это правильно. Но написание такой системы – дело нелегкое, так как данная тема по своей сути очень условна и иногда «веет» философией.

Многие вопросы остались без рассмотрения, ввиду отведенных на статью объемов, но я надеюсь, что смогу рассмотреть оставшиеся вопросы в цикле статей о «Проектировании Framework-систем с применением паттерна MVC и основах теории шаблонизации».

Динамический генератор функций

В данной статье в качестве источника идей использована одна из глав книги о PHP («Разработка веб-приложений с помощью PHP 4.0»). Она мне показалась очень интересной с точки зрения возможностей использования PHP. Не полнитесь и запустите приведенный код – это жутко интересно.

Автор:
Дмитрий Заболотский

Пока авторы писали эту книгу, кто-то в немецком листе по PHP спросил о том, как можно обрабатывать математические функции, введенные пользователем. Он хотел знать, как можно отобразить функцию, если пользователь ввел формулу.



Дискуссия быстро пошла традиционным путем; каждый хотел сделать это сложным, очень сложным, вместо того, чтобы посмотреть на все проще. Идея в следующем.

Вот шаги, происходящие при вводе функции:

1. Анализ введенных данных;
2. Создание разобранного представления (нечто связанное с технологиями компиляторов);
3. Позволить процессору работать с разобранном представлением и шаг за шагом генерировать выходные данные.

Например, разбор $f(x) = m * x + b$ будет выглядеть примерно так:

1. Мы видим, что функция зависит от x , a и b – константы.
2. Создадим структуру для более легкого разбора. Например, создадим дерево связей переменных со знаками умножения и сложения.
3. Сделаем x переменной и будем сохранять результат в u .

Фактически это метод, преподаваемый в университетах и включающий в себя очень сложные программы. Кажется, что никто не способен освободить себя от заранее существующих решений и создать более прогрессивные решения. вы когда-нибудь думали о том, как PHP интерпретирует скрипты?

1. Анализ входных данных;
2. Генерация разобранного представления;
3. Запуск этого представления на исполнение.

Итак, это очень похоже на то, что нам нужно. Так почему бы не трансформировать входные функции в PHP-код и позволить PHP работать за нас? PHP поддерживает динамическое программирование, и эта задача может стать тривиальной.

На самом деле, регулярные выражения для трансформации математических функций в PHP-код очень просты.

Предполагая, что все переменные состоят из нескольких символов латинского алфавита и соединяются математическими PHP-операциями (+, -, * и т.д.), следующая строчка может сделать за нас всю работу:

```
$php_code = ereg_replace("[a-zA-Z]+", "$\\0", $input_function);
```

Выражение “ $m * x + b$ ” будет трансформировано в “ $\$m * \$x + \$b$ ”. Создав немного кода вокруг этого регулярного выражения и сделав упрощающие предположения, мы сможем быстро сделать динамический построитель графиков.

Честно говоря, прием, предложенный авторами книги, очень интересен, но меня лично не устроило то, что я не могу, например, написать: $\sin(x)$. Что же это за построитель графиков без возможности использования стандартных математических функций?

Также необходимо осветить некоторые проблемы безопасности при использовании такого подхода. Техника непосредственного исполнения PHP-кода с помощью `eval()`, введенного пользователем, приведенная здесь никогда не должна быть использована в реальных скриптах. Исполнение пользовательского кода - это огромная дыра в вашей системе, т.к. любой может выполнить нечто, вроде `system("rm -r /*");` и удалить всю информацию на вашем веб-сервере, к которой вы имеете доступ. Этот пример был приведен только как демонстрация динамической генерации кода и его выполнения.

Но, как ни странно, существуют некоторые пути выхода из этой ситуации, например, замена всех ”плохих” функций еще при вводе, или отказ в допуске опасного кода к исполнению, если таковой вообще имеется. Например, потенциально нехорошей будет функция `system()`. Я думаю, не стоит надеяться на `disable_functions` (если вдруг забыли, то это такая директива в `php.ini`).

А вот с параметрами, которые необходимы для математических расчетов (константы), можно поступить проще – договориться, что параметры указываются в виде:

```
параметр1=значение1, параметр2=значение2, ...
```

В итоге получаем следующий код:

```
//  
// Определяем глобальные константы  
//  
define('PLOT_MIN', 0.1);  
define('PLOT_MAX', 100);  
define('PLOT_STEP', 0.5);  
define('DIAGRAM_HEIGHT', 300);  
define('DIAGRAM_HORIZON', 150);
```

Продолжение на следующей странице.

```

function parse_function($in_string,$params)
{
    //делаем проверку на нехорошие функции, а заодно и на кавычки
    $f_bad=array
('system','eval','popen','fopen','exec','passthru','shell_exec','','');
    for ($i=0;$i<=sizeof($f_bad)-1;$i++)
        if (strpos($in_string,$f_bad[$i])!==false) die("bad function
used");

    $header = '';
    $header = $header."function calculate(\$req_code, \$x)\n";
    $header = $header."{\n";

    $header .= p_trans($params);
    $footer = "\n}\n";

// конвертируем все символы в PHP-переменные
// заменяем все, что похоже на несколько идущих подряд символов,
// на переменную с таким именем. Мы знаем, что функции на конце имеют //
открывающую скобку
$out_string = ereg_replace( '[a-zA-Z]+' , "$\\0" , $in_string);
// и потому убираем в начале символ "$", если это похоже на функцию
$out_string = preg_replace( "/\\$([a-zA-Z]+)([ ]/" , "\\1(" , $out_string);

    $out_string = $header."return(.$out_string.);\n".$footer;

    return($out_string);
}
//эта функция добавлена лично мной
function p_trans($req_code)
{
    $tmp=explode(',',$req_code);
    for ($i=0,$out='';$i<=sizeof($tmp)-1;$i++)
        if (ereg(('[a-z]+)([\\.-9+\\-]+)', $tmp[$i], $regs)) $out.="\\n
\\$regs[1]=$regs[2];";
    return $out;
}

```

Окончание листинга. Начало смотрите на предыдущей странице.

Давайте рассмотрим то, что я тут переделал и добавил. В переменной `$f_bad` объявляются все нехорошие, с точки зрения безопасности, функции, позволяющие пользователю сотворить больше, нежели просто нарисовать график. Возможно, список и не полон, но я постарался учесть все. А если и забыта какая-нибудь функция, то я проверяю на наличие кавычек (одинарных и двойных), т.к. при вызове любой функции, более или менее опасной, придется пользоваться кавычками, а в математических функциях они даром не нужны.

В цикле с помощью функции `strpos()` я ищу попадание плохих функций в строку для преобразования, и если таковая найдена, то скрипт умирает.

Перед тем, как делать `eval()` переданных параметров, нам надо преобразовать их в PHP-код, раз уж мы не хотим получить от пользователя что-нибудь неприличное и здесь. Этим и занимается функция `p_trans()`. Сначала [в функции] с помощью функции `explode()` разбиваем входную строку на массив, где разделителем служит запятая.

Затем каждый элемент массива проверяем функцией `ereg()` на соответствие регулярному выражению, может, и жутко страшному на первый взгляд, но означающему лишь одно: сначала стоит одна или несколько букв, затем знак равенства, а затем число, как положительное, так и отрицательное, как целое, так и дробное. И если это так, то “создаем” PHP-код вида `$переменная=значение`. Каждое такое вхождение склеиваем в одну строку и возвращаем как результат.

Также, еще одной проверкой правильности введенного математического выражения может служить равенство числа открывающих и закрывающих скобок и отсутствие символа “точка с запятой”. Но это представляется тривиальным, и если вы захотите, вы легко можете сделать это сами.

Продолжим сам код:

```
function create_image()
{
    global $color_plot;

    $width = PLOT_MAX / PLOT_STEP;
    $height = DIAGRAM_HEIGHT;
    $image = imagecreate($width, $height);
    // создадим цвета
    $color_backgr = imagecolorallocate($image, 255, 255, 255);
    $color_grid = imagecolorallocate($image, 0, 0, 0);
    $color_plot = imagecolorallocate($image, 255, 0, 0);
    // очистим изображение
    imagefilledrectangle($image, 0, 0, $width - 1, $height - 1,
    $color_backgr);
    // рисуем оси
    imageline($image, 0, 0, 0, $height - 1, $color_grid);
    imageline($image, 0, DIAGRAM_HORIZON, $width - 1,
    DIAGRAM_HORIZON, $color_grid);
    // печатаем текст
    imagestring($image, 3, 10, DIAGRAM_HORIZON + 10, PLOT_MIN,
    $color_grid);
    imagestring($image, 3, $width - 30, DIAGRAM_HORIZON + 10,
    PLOT_MAX, $color_grid);
    // возвращаем изображение
    return($image);
}
function plot($image, $x, $y)
{
    global $color_plot;
    // сделаем их статичными для запоминания координат
    static $old_x = PLOT_MIN;
    static $old_y = 0;

    if($old_x != PLOT_MIN)
    imageline($image, $old_x / PLOT_STEP, DIAGRAM_HEIGHT -
    ($old_y + DIAGRAM_HORIZON), $x / PLOT_STEP, DIAGRAM_HEIGHT -
    ($y + DIAGRAM_HORIZON), $color_plot);
    $old_x = $x;
    $old_y = $y;
}
```

Продолжение на следующей странице.

```

if(!isset($function_string))
{
    // функция еще не создана
    print("<html><body>");
    print("<form action=\"\".basename($PHP_SELF).\"\" method=\"post\">");
    print("Function definition: <input type=\"text\"
        name=\"function_string\" value=\"(m*x+b)/(x/3)\"><br>");
    print("Required constants: <input type=\"text\" name=\"req_code\"
        value=\"m=10,b=20\"><br>");
    print("<input type=\"submit\" value=\"Parse\">");
    print("</form>");
    print("</body></html>");
}
else
{
    // создадим PHP-код
    $parsed_function = parse_function($function_string,$req_code);

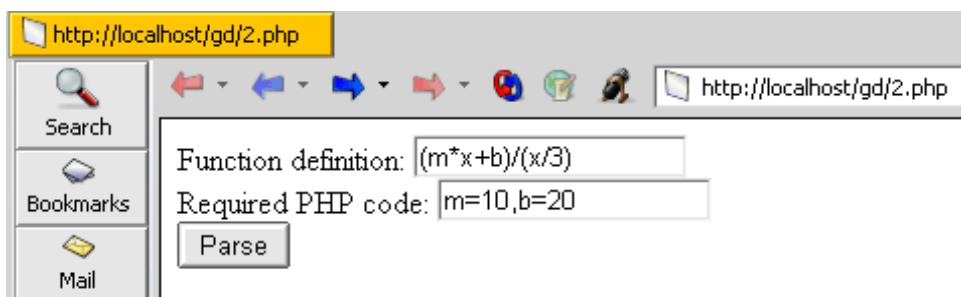
    eval($parsed_function);
    // создадим изображение
    $image = create_image();
    // рисуем график
    for($x = PLOT_MIN; $x < PLOT_MAX; $x += PLOT_STEP)
    {
        $y = calculate($req_code, $x);
        plot($image, $x, $y);
    }
    // выставляем тип
    header("Content-type: image/png");
    // посылаем изображение
    imagepng($image);
}

```

Окончание листинга. Начало на предыдущей странице.

Тут все довольно прозрачно и не требует больших комментариев. Стоит оговориться, что вам хотя бы в общих принципах необходимо знать то, как работать с библиотекой GD.

Этот скрипт исполняем, вы можете сразу же посмотреть его работу в браузере. После первого вызова вы сможете указать функцию для исполнения в небольшой форме, как здесь:



Первое поле предназначено для ввода функции рисования. В нашем примере мы делаем предположение, что функция всегда зависит только от x . Во втором поле вы можете определить свои переменные.

Теперь вы можете нажать кнопку "Parse".

На рисунке справа показано то, что может получиться. Так как же скрипт преобразует данные формы в график?

Давайте обсудим это шаг за шагом. После того, как вы отправили форму, скрипт начинает выполнять веточку else в главном if-условии. Первая функция вызывается следующим образом:

```
$parsed_function = parse_function ($function_string, $req_code);
```

parse_function() создает PHP-код, введенный пользователем, применяя к нему регулярное выражение. Для комфортного использования математических функций этот код встроен внутрь маленькой функции, которая просто создает переменные из констант введенных пользователем, а затем выполняя математическое выражение и возвращая результирующее значение.

Функция, генерируемая при помощи parse_function() для выражения "m * x + b", может быть следующей:

```
function calculate($req_code, $x)
{
    $m=1;
    $b=2;
    return ( ($m*$x+$b) / ($x/3) );
}
```



Транзакции – головная боль или крылья?

Автор начал изучать PHP, хорошенько освоив Делфи в связке с InterBase. В те давние времена проект «FireBird» (старшая сестра ИнтерБейса) только-только зарождался, однако и InterBase уже был транзакционным. Автор привык к транзакциям. Последствием такого стечения обстоятельств, судя по всему, явилась лень. Автор считает людей, блокирующих таблицы и строки «вручную», трудолюбивыми и бесстрашными программистами.

Автор:
Александр Косарев
Пловдив, Болгария

Сам автор там, где надо и когда надо просто использует транзакции. Термин «транзакция» был объяснён ему на реальном примере, что дало понимание того, как работают транзакции без знания их научного определения.



В этой статье автор ставит своей целью изложить один из многих путей познания столь страшного термина «транзакция».

В примере будет использована столь близкая сердцу многих веб-разработчиков СУБД MySQL. Для начала экспериментов в этой СУБД необходимо включить поддержку транзакций.

В MySQL 3.23 для этого нужно пересобрать пакет, так как по умолчанию поддержка необходимых нам типов таблиц выключена.

Скрипт конфигурирования в *nix нужно запустить так:

```
./configure --with-innodb
```

Для пользователей Windows лучше взять пакет MySQL-Max-3.23 и далее в my.cnf для минимальной поддержки добавить:

```
innodb_data_file_path=ibdata:30M
```

Версии 4.x пересобирать не надо (необходимость существует только в том случае, если поддержка транзакционных типов таблиц выключена), в них нужно настроить поддержку транзакционных таблиц. Для этого в my.cfg добавить:

```
innodb_data_file_path=ibdata:30M
```

MySQL поддерживает два типа транзакционных таблиц: InnoDB и DBD. В данной статье автор приводит пример с использованием InnoDB. Перейдем к php-коду.

```
<?php
  $host = 'localhost';
  $user = 'test';
  $pass = 'test';
  $db_name = 'test_transaction';
?>
```

Листинг 1. Файл config.php

В данном скрипте читателю необходимо поменять настройки на свои. Задача этого файла – обеспечить доступ к конфигурационным данным в одном источнике.

Теперь создадим таблицу, с которой и проведём все эксперименты.

```
<?php
include('config.php');
mysql_connect($host, $user, $pass) or die
(__FILE__.':'.__LINE__.':'.mysql_error());

mysql_select_db($db_name) or die(__FILE__.':'.__LINE__.':'.mysql_error());

$sql = 'create table if not exists table_trans(id int, name varchar(50),
primary key(id))
type=InnoDB';//заявка для создания таблицы типа InnoDB

mysql_query($sql) or die(__FILE__.':'.__LINE__.':'.mysql_error());

$sql = 'insert into table_trans values(1, \'notest\')';
mysql_query($sql);
if ('1062' == mysql_errno())
{
echo 'Тестовая запись уже существует.<br />';
} elseif (mysql_errno())
{
die(__FILE__.':'.__LINE__.':'.mysql_errno().':'.mysql_error());
}

echo 'Подготовка проведена.';
?>
```

Листинг 2. Файл prepare.php

Повторный запуск prepare.php после успешного предыдущего приведёт к ошибке дублирования ключей, которая значится под номером 1062. Её благополучно отлавливаем. Для многих программистов, возможно, будет проще взглянуть на код для работы с транзакциями в целом. Внимательно посмотрите скрипт test1.php, приведенный в приложении к журналу.

В скрипте автор акцентирует внимание читателя словом "ВНИМАНИЕ" на различных уровнях изоляции транзакций. Убирая символы комментариев перед парами строк и запуская скрипт, читатель ясно и чётко сможет увидеть результаты влияния типов транзакций на их работу.

Будет это выглядеть примерно так:

```
шаг первый
произведён insert в первой транзакции.
результат первой транзакции после выполнения в ней
insert и последующего select
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

```
результат второй транзакции после выполнения в первой
insert и последующего select
+---+-----+
|1 | notest |
+---+-----+
```

шаг второй
произведён commit в первой транзакции.

результат видимости для первой транзакции после завершения первой транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после завершения первой транзакции (во второй ещё не произведено commit)

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

шаг третий
произведён commit во второй транзакции.

результат видимости для первой транзакции после завершения второй транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после её завершения

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

Эксперимент произведён с изолирующим уровнем READ COMMITTED. Что означает: изменения, произведённые в данной транзакции, не будут видны остальным, пока она не будет завершена.

Следующий эксперимент с уровнем изоляции READ UNCOMMITTED даёт следующий результат:

шаг первый
произведён insert в первой транзакции.

результат первой транзакции после выполнения в ней insert и последующего select

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат второй транзакции после выполнения в первой insert и последующего select

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

шаг второй
произведён commit в первой транзакции.

результат видимости для первой транзакции после завершения первой транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после завершения первой транзакции (во второй ещё не произведено commit)

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

шаг третий
произведён commit во второй транзакции.

результат видимости для первой транзакции после завершения второй транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после её завершения

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

Отсюда видно, что **READ UNCOMMITTED** даёт возможность любой транзакции "видеть" результат изменений в других транзакциях, даже не завершённых.

Следующий уровень **REPEATABLE READ**

шаг первый произведён insert в первой транзакции.
результат первой транзакции после выполнения в ней insert и последующего select

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат второй транзакции после выполнения в первой insert и последующего select

```
+---+-----+
|1 | notest |
+---+-----+
```

шаг второй
произведён commit в первой транзакции.

результат видимости для первой транзакции после завершения первой транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после

завершения первой транзакции (во второй ещё не произведено commit)

```
+---+-----+
|1 | notest |
+---+-----+
```

шаг третий

произведён commit во второй транзакции.

результат видимости для первой транзакции после завершения второй транзакции

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

результат видимости для второй транзакции после её завершения

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

Итак REPEATABLE READ скрывает все изменения, произведённые другими транзакциями, до тех пор, пока не будет произведён commit.

И последний тип изолирующего уровня SERIALIZABLE выдаёт краткий и жесткий результат:

шаг первый произведён insert в первой транзакции.

результат первой транзакции после выполнения в ней insert и последующего select

```
+---+-----+
|1 | notest |
|2 | test1.1|
+---+-----+
```

Warning: mysql_query(): Unable to save result set in /usr/ports/hosts/test/data/www/test1.php on line 106

/usr/ports/hosts/test/data/www/test1.php:108:Lock wait timeout exceeded; Try restarting transaction

Появление Warning произошло оттого что все заявки SELECT превращаются в SELECT ... LOCK IN SHARE MODE, то есть после первого выполнения select в первой транзакции, select, произведённый во второй, будет ждать некоторое время и завершится с 108-ой ошибкой.

При работе с транзакциями их обязательно нужно завершать, иначе при запуске скрипта можно получить:

/usr/ports/hosts/test/data/www/test1.php:1205:Lock wait timeout exceeded; Try restarting transaction

.

Автор старательно избегал излагать какие либо научные определения так как в понимании работы транзакций они не всегда помогают. Об этом говорит опыт автора.

Когда автор слышит от коллег по работе слово «понял» - это, для него означает лишь то, что изложенная информация явилась лишь информацией, но для знания нужен опыт, а опыт человек может приобрести только своим трудом.

Все листинги, приведенные в данной статье читатель может найти в приложении к журналу.

Безопасно-ориентированное программирование

Проблема создания безопасных скриптов всегда стояла довольно остро. Даже если вы создаете просто домашнюю страницу для себя, вам вряд ли будет приятно, если какой-нибудь умелец ее взломает. Что говорить о крупных корпоративных и коммерческих сайтах, взлом которых может нанести довольно значительный финансовый ущерб.

Чаще всего взломщики пользуются различными недоработками в скриптах, прежде всего недостаточной проверкой поступающей извне (чаще всего от посетителя сайта) информации. В данной статье я покажу, как, используя современные технологии программирования, можно организовать гибкую и надежную систему проверки данных, полученных от пользователя. Говоря «современные технологии программирования», я прежде всего имею в виду шаблоны проектирования. Для понимания статьи желательно владеть PHP5 и ООП. Тем, кто заинтересуется шаблонами проектирования, очень рекомендую книгу «Design Patterns: Elements of Reusable Object-Oriented Software» «банды четырех F».

Автор:

Борис Вольфсон

borisvolfson@mail.ru

<http://www.borisvolfson.h11.ru>



Уязвимости

Какой код может быть уязвимым к атакам взломщиков? Как взломщики пользуются дырами в скриптах? Прежде всего отмечу, что достаточно грамотно написанный скрипт достаточно трудно взломать, более того, большинство взломов происходит из-за недостаточной бдительности и доверчивости программистов. «Проверять, проверять и еще раз проверять входные данные» - это лозунг для написания надежных и безопасных скриптов.

Имена файлов

Чаще всего входные данные поступают из заполненных форм, либо из адресной строки. Данные из форм передаются с помощью метода POST, а чтобы передать данные, используя гипертекстовую ссылку, применяют метод GET. Например, у вас на сайте размещены различные статьи, и для показа статьи вы используете скрипт. Зачастую информация о статьях хранится в базе данных, а специальный скрипт выдает примерно такой список:

```
<a href="show.php?filename=article1.html">Статья 1</a>
<a href="show.php?filename=article2.html">Статья 2</a>
<a href="show.php?filename=article3.html">Статья 3</a>
```

Каждая ссылка указывает на скрипт show.php, который показывает статьи. А в качестве параметра ему передается имя файла.

Сам скрипт `show.php` обычно содержит следующий код:

```
// вывод верхней части страницы
...
echo file_get_contents($filename);
...
// вывод нижней части страницы
```

Конечно, это самый запущенный случай: в данном коде две ошибки с точки зрения написания безопасного кода. Во-первых, для получения внешних данных (имени файла) используется глобальная переменная `$filename`, вместо этого лучше использовать ассоциативные массивы `$_REQUEST`, `$_POST` и `$_GET`. Во-вторых, `$filename` никак не проверяется. Например, взломщик может вместо имени файла в строке ввода адреса указать `index.php` и получить текст скрипта, а такой ценной информацией он без труда воспользуется для дальнейшего поиска недочетов в вашей системе безопасности.

Выхода в принципе два: можно вместо имени файла передавать идентификатор статьи, а затем преобразовывать его в имя файла, либо тщательно проверять имя файла на допустимые имена, на слеш и обратные слеш, чтобы взломщик не смог путешествовать по вашим каталогам.

Функция `system()`

Бывают более патологические случаи: некоторые «программисты» используют внешние данные в функции `system()`, которая выполняет произвольную команду операционной системы. В таком случае до дефейса сайта один шаг. Функцию `system`, вообще не следует применять без экстренной необходимости. Но если пришлось использовать ее вместе с внешними данными, то тут надо будет осуществить очень строгую проверку на спец. символы: от всех специальных символов *nix-систем до символа с кодом 0.

Базы данных

Все реже и реже для хранения данных напрямую используются файлы, и все чаще и чаще используются базы данных. В связке с PHP обычно применяют базу данных MySQL. Доступ к данным в таком случае осуществляется при помощи языка запросов SQL. В такие запросы приходится вставлять внешние данные, например, для гостевой книги в базе данных надо сохранить, как минимум, ник и сообщение пользователя. Язык SQL очень гибок, поэтому надо быть внимательным к входным данным. Кроме «вредоносных» данных, следует обрабатывать неправильный ввод пользователя, например, слишком длинную строку.

Чаты, гостевые, форумы

В данном случае нужно волноваться также и о взломе другого типа: взломщик может попытаться воспользоваться тегами HTML, то есть нельзя допускать ввода символов "<" и ">".

Их надо либо экранировать, либо выводить сообщение о неправильном вводе. Если вы не учитываете это правило, в лучшем случае можете получить у себя в гостевой что-нибудь вроде:

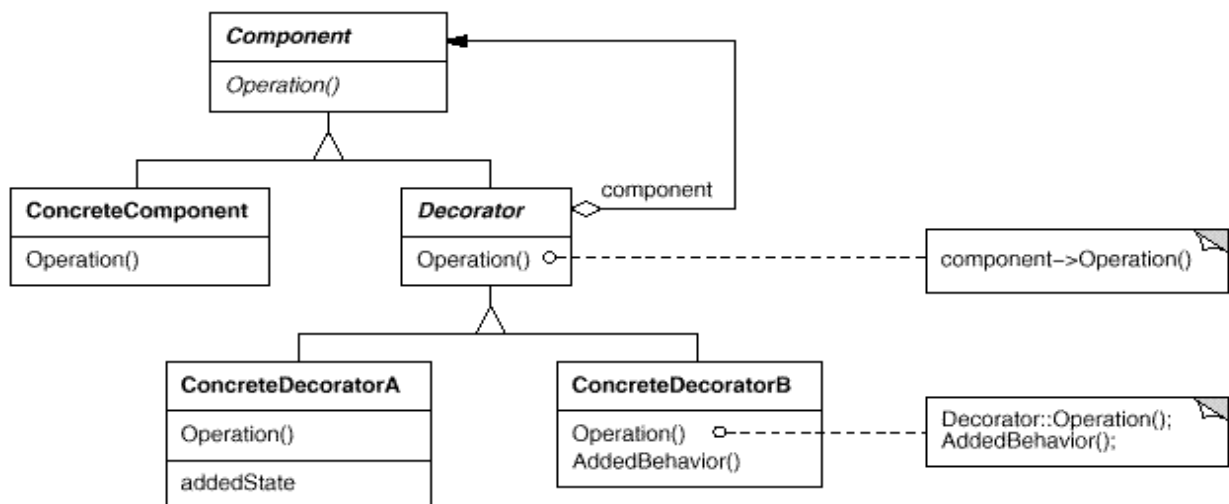
This site was hacked by *****

Конечно, сейчас на сайтах используют готовые гостевые, чаты, форумы и тому подобное, но стоит их проверять перед тем, как использовать, ведь недочет в гостевой может стать причиной взлома вашего сайта.

Решение проблемы

Чтобы решить данную проблему, можно написать функции для различных проверок, однако PHP5 довольно неплохо поддерживает объектно-ориентированное программирование - более мощный и гибкий подход. Итак, пусть мы решили, что, прежде всего, будем проверять размер строки. Можно написать соответствующий класс. Далее нам приходит в голову, что неплохо бы сделать проверку на отсутствие слешей и обратных слешей. Мысль опытного программиста на этом не остановится, он придумает другие проверки. Как же нам их все учитывать? Нам будет нужно применять эти проверки в произвольном порядке. Давайте четко выясним, что для этого нужно сделать. Нам необходимо динамически добавлять новое поведение объекту – то есть, различного рода проверки.

Можно, конечно, придумать решение самому, но у программистов есть поговорка: «Не надо изобретать колесо (велосипед)». Иногда лучше воспользоваться уже готовым решением - шаблоном проектирования, который используют профессиональные программисты с огромным опытом. В выше упомянутой книге «Банды четырех» (Gang of Four, GoF) есть подходящий нам шаблон проектирования - Decorator. «Decorator» переводится как «украшатель», то есть мы декорируем нашу проверку длины строки дополнительными проверками. Приведу схему, по которой мы будем дальше действовать:



Component (Checker) - это простой абстрактный класс проверки, ConcreteComponent (StringChecker) - это класс для проверки длины строки, который мы будем декорировать дополнительным поведением. Decorator (Decorator) - это абстрактный класс «декораторы», который нужен для того, чтобы организовать цепочку декораторов. ConcreteDecorator (SlashChecker, BackSlashChecker, ... - это классы с дополнительным поведением (в нашем случае это проверки).

PHP-код, реализующий данную диаграмму, можно найти в приложении к журналу. Этот код должен выдавать следующий результат:

```
Проверка строки: 'Строка /для/ проверки'
Проверка строки на обратные слэши: Проверка пройдена
Проверка строки на слэши: Проверка не пройдена
Проверка размера строки: Проверка пройдена
Проверка строки: '1365m434\'
Проверка строки на символы, отличные от цифр: Проверка не пройдена
Проверка строки на обратные слэши: Проверка не пройдена
Проверка размера строки: Проверка пройдена
```

Класс Checker представляет собой абстрактный класс для проверки, то есть в нем объявлены функции для проверки, но использовать мы его не сможем, так как функции не реализованы. Например, объявлена функция Check для проверки строки. В классе StringChecker функции Check уже реализована для проверки размера строки. В абстрактном классе Decorator добавлена переменная MyChecker для организации цепочки декораторов.

Функция __construct(Checker \$MyChecker) - это конструктор, который вызывается при создании объекта класса. Конструктору передается объект Checker, который будет сохранен для вызова функции Check. Функция Check класса Decorator просто вызывает \$MyChecker->Check(\$StringToCheck). В классах для конкретных проверок нужно просто переписать функцию Check и не забыть в конце ее вызвать parent::Check(\$StringToCheck), то есть Check класса Decorator.

Посмотрим, как эта система работает. Необходимо создать объект, который будет выполнять проверку. Пусть нам надо проверить строку на то, состоит ли она из одних цифр, есть ли в ней слэши и уточнить ее размер:

```
$Checker = new DigitsChecker(new BackSlashChecker(new StringChecker()));
```

Теперь просто вызовем метод Check():

```
$Checker->Check($S); // $S - строка для проверки
```

Проследим цепочку вызовов. Объект \$Checker является объектом класса DigitsChecker, поэтому сначала будет произведена проверка на наличие символов, отличных от цифр. Но в конце этого метода стоит строка, которая вызывает родительский метод Check.

```
parent::Check($StringToCheck);
```

Метод `Check` класса `Decorator` вызывает `BackSlashChecker::Check`, а он в свою очередь вызовет `StringChecker::Check`, на чем проверка и закончится.

Заключение

Применение шаблона проектирования `Decorator` позволило создать достаточно гибкую и надежную систему проверки. Для создания нового типа проверки достаточно просто создать новый класс от класса декоратор и переопределить метод `Check`. Проверки можно применять в любой последовательности, которая определяется при создании объекта проверки. Если необходимо, чтобы последовательность проверок была обратной (то есть первой выполнялась проверка `StringChecker::Check`), надо просто поместить вызов метода `Decorator::Check($StringToCheck)` в начале методов проверки.

Изменим жизнь к лучшему с помощью буферизации вывода

По моему предвзятому мнению, буферизация вывода является одной из самых изящных возможностей в PHP 4. Несмотря на всю свою простоту, этот инструмент позволяет разработчикам создавать современные и эффективные программы. В этой статье я расскажу о заголовке HTTP, о том как с помощью буферизации вывода справиться с трудностями этого заголовка и затрону остальные случаи удобного применения этой возможности.

Автор:
Zeev Suraski
Перевод:
Михаил Мотыженков
Сайт оригинала:
<http://zend.com>

Заголовок HTTP

Что такое заголовок и зачем нам нужно о нём заботиться? Для каждого запроса через HTTP-протокол есть, по существу, две части, которые составляют ответ веб-сервера: заголовок и тело. Например, у вас есть текстовый файл «example.txt» в корневом каталоге сервера, который содержит фразу «Hello world!». Ответ с сервера о нём будет выглядеть примерно так:



```
HTTP/1.1 200 OK
Date: Sat, 02 Sep 2000 21:40:08 GMT
Server: Apache/1.3.11 (Unix) mod_macro/1.1.1 PHP/4.0.2-dev
Last-Modified: Sat, 02 Sep 2000 21:39:49 GMT
ETag: "12600b-e-39b173a5"
Accept-Ranges: bytes
Content-Length: 14
Connection: close
Content-Type: text/plain

Hello, world!
```

HTTP-заголовком в данном случае будет первый абзац. Заголовок невидим для конечного пользователя, но содержит достаточно полезную информацию для браузера: тип содержимого, версия используемого протокола, время последнего изменения файла и т.д. Для заголовков существует не так много правил, ценных с практической точки зрения. Вообще говоря, он состоит из строк следующего формата:

Поле: значение

От тела документа заголовок отделяется пустой строкой. Как вы уже, наверное, догадались, эту информацию можно добавить или изменить через PHP-скрипт. Первый вариант - сделать это напрямую, используя функцию `header()`:

```
header("Location: http://www.php.net/"); //перенаправление
```

Или же осуществить это неявным образом через функцию `SetCookie()`:

```
SetCookie("foo", "bar");
```

HTTP cookies являются средством реализации HTTP заголовков. Пусть имеется такой PHP-файл:

```
SetCookie("foo", "bar");  
print "Set cookie.";
```

Ответ на запрос HTTP для него выглядел бы примерно так:

```
HTTP/1.1 200 OK  
Date: Sat, 02 Sep 2000 21:43:02 GMT  
Server: Apache/1.3.11 (Unix) mod_macro/1.1.1 PHP/4.0.2-dev  
X-Powered-By: PHP/4.0.2-dev  
Set-Cookie: foo=bar  
Connection: close  
Content-Type: text/html  
  
Set cookie.
```

Браузер прочтёт пришедший с сервера заголовок HTTP и поймёт, что cookie с именем «foo» и значением «bar» было установлено.

Необходимость буферизации выхода

Истоки буферизации выхода относятся к PHP/FI 2.0, когда необходимость в ней стала очевидна. Если вы имели дело с этой древней версией PHP, то, вероятно, помните легендарное сообщение об ошибке "Oops, SetCookie called after header has been sent".

В PHP 3.0 и 4.0 вы могли встретиться с этим сообщением немного в другой форме: "Oops, php_set_cookie called after header has been sent". Так же вы могли столкнуться с сообщением «Cannot add header information - headers already sent» при попытке вызвать функцию `header()`.

Изначально буферизация выхода была разработана, чтобы положить конец этим раздражающим сообщениям, позволив, в то же время, разработчикам использовать её для решения задач повышенной сложности.

Давайте разберёмся, когда возникали эти ошибки. Обычно они появляются, когда вы пытаетесь добавить или изменить информацию заголовка после того, как PHP-скрипт уже отправил HTTP-заголовок, дойдя до пустой линии - разделителя тела и заголовка документа. Чтобы понять, почему такое происходит, рассмотрим механизм обработки заголовка и тела PHP-скриптом.

В момент, когда скрипт запускается на исполнение, он может изменять информацию и в заголовке и в теле. Заголовок отправляется не сразу, а лишь сохраняется в списке. Это позволяет нам модифицировать заголовки в случае необходимости, включая стандартные заголовки (такие, как "тип содержимого").

Тем не менее, как только скрипт доходит до информации, не относящейся к заголовку (например, HTML-блок или вызов `print()`), PHP-скрипт должен сначала послать заголовок и пустую строку, заканчивающую данный раздел. Только после этого скрипт может работать с телом файла. Отсюда и далее, любая попытка добавить или изменить содержимое заголовка повлечёт за собой одну из выше рассмотренных ошибок с соответствующим сообщением.

Как работает буферизация выхода?

Если буферизация выхода активирована, PHP не посылает HTTP-заголовок, когда скрипт доходит до тела. Вместо этого он перенаправляет информацию в динамический буфер (это стало возможным, начиная с версии PHP 4.0). Благодаря этому, вы всё ещё можете добавлять строки в заголовок, изменять их и устанавливать cookies, потому что заголовок фактически не отправлен. В простейшем случае, когда скрипт заканчивает выполнение, PHP автоматически посылает HTTP-заголовок и затем всё остальное содержание буфера браузеру.

Элементарное использование

Существует четыре функции, которые помогают контролировать буферизацию выхода:

1. `ob_start()` Включает буферизацию выхода. Она поддерживает множество уровней - то есть, вы можете вызвать `ob_start()` несколько раз;
2. `ob_end_flush()` Посылает содержимое буфера и отключает буферизацию выхода;
3. `ob_end_clean()` Очищает буфер, не посылая его, и отключает буферизацию выхода;
4. `ob_get_contents()` Возвращает текущий буфер в виде строки. Это позволяет Вам следить, что скрипт уже послал.

Так же вы можете активировать директиву `output_buffering` в `php.ini`. В этом случае каждый PHP-скрипт будет вести себя так, как если бы он встретил вызов `ob_start()`.

Пример 1

```
<?php ob_start(); ?>

<h1>Пример 1</h1>

<?php

print "Привет, $user\n";

SetCookie("Wow", "Эта cookie была установлена даже не смотря на то, что мы уже начали тело документа!")

?>
```

Как видите, несмотря на то, что вы уже отправили информацию на вывод (дважды: в блоке HTML и функцией `print()`), `SetCookie()` была выполнена успешно, благодаря механизму буферизации выхода. Прошу заметить, что использование буферизации выхода отнимает несколько процентов производительности - поэтому будет правильнее не включать её по умолчанию. Однако для сложных сценариев, в которых буферизация выхода может существенно упростить задачу, не давая отправлять заголовок вначале - это лучший вариант использования.

Пример 2

```
<?php
ob_start();
print "Вот очень нерациональный способ вычисления длины строки.";
$length = strlen(ob_get_contents());
ob_end_clean();
?>
```

Данный пример демонстрирует крайне неэффективный способ определения длины строки. Вместо того чтобы просто скормить её функции `strlen()`, мы включаем буферизацию выхода, печатаем строку и вычисляем длину буфера. После этого нужно будет очистить буфер (не посылая содержимого) и отключить его.

Продвинутое использование

Хотя буферизация выхода изначально была предназначена для решения проблем с HTTP-заголовками, она показала себя как гораздо более мощное средство. Первые признаки вы можете увидеть уже во втором примере, который был рассмотрен чуть выше. Я намекаю на очень полезную функцию `ob_get_contents()`.

Вместо отправки буфера как есть, вы можете просмотреть его, управлять им и даже отменить, если он чем-то Вам не понравился. Всё это можно сделать без каких-либо изменений в Вашем старом коде, разве что только придётся включить буферизацию выхода.

Очень важное замечание для такого использования - буферизация выхода является реентерабельной. Это значит, что одна буферизация выхода может быть расположена внутри другой. Такая возможность позволяет Вам писать функции, которые используют буферизацию выхода, не зависимо от того, включена она сейчас в скрипте или нет.

Буферизация выхода также может быть очень полезна при разработке веб-приложений. Я приведу два примера: альтернативный вариант функции `eval()` и очень простой способ сократить Ваш HTTP-трафик, используя сжатие данных.

my_eval() - слегка изменённая версия eval()

Для тех, кто не в курсе, `eval()` - это функция, позволяющая исполнять код, находящийся внутри строки. Многие люди удивились, когда узнали, что у `eval()` возвращаемым значением является не результат, выполнения кода, а просто получившийся код. Конечно же, это правильное поведение - `eval()` исполняет код так, как если бы код был в файле, вызванном через `include()`. Однако иногда это неправильное поведение может сильно пригодиться. Когда мы используем буферизацию выхода, заставить `eval()` работать "неправильно" очень легко:

```
function my_eval($code)
{
    ob_start();
    eval($code);
    $output = ob_get_contents();
    ob_end_clean();
    return $output;
}
```

Эта функция будет возвращать значение, которое получилось в ходе выполнения кода внутри `eval()`. В этом случае настоящее значение `eval()` будет утеряно, но оно довольно редко используется (в PHP 4.0, то, что вы пишете после «return» в коде `eval()`, будет возвращаемым значением). Обратите внимание ещё раз на то, что буферизацию выхода мы вызывали внутри функции; её тело будет работать правильно вне зависимости от того, была включена буферизация выхода до вызова функции или нет.

Уменьшаем свой HTTP-трафик - сжимаем данные

Многие браузеры поддерживают сжатие данных по алгоритму `gzip`. Если сервер обнаружит, что браузер поддерживает кодирование `gzip`, то он может сжать данные и послать их в таком виде (в заголовке HTTP нужно добавить строчку «Content-Encoding: gzip»). Как только браузер получит заархивированную информацию, он расшифрует её на лету. Трафик от сервера к браузеру сократится, время на загрузку страницы уменьшится.

Реализация этой идеи чуть сложнее, чем в предыдущем примере и требует добавления нескольких строк в каждый файл, который вы хотите сжать. Рациональнее будет включить эти строки в автоматически присоединяемый файл.

```
function compress_output($output)
{
    // Здесь можно осуществить дополнительные действия с $output, например,
    // удалить лишние пробелы
    return gzencode($output);
}

// Проверяем, поддерживает ли браузер сжатие gzip - HTTP_ACCEPT_ENCODING
if (strstr($_SERVER_VARS['HTTP_ACCEPT_ENCODING'], 'gzip')) {
    // Активируем буферизацию выхода и регистрируем compress_output() (об этом
    // - ниже)
    ob_start("compress_output");
    // Сообщаем браузеру, что содержимое сжато по алгоритму gzip
    header("Content-Encoding: gzip");
}
```

С такими комментариями разобраться в коде не составит труда. Заметьте, что мы использовали имя функции в качестве аргумента для `ob_start()` (до этого мы её вызывали без параметров).

Что же делает эта функция? Когда вы вызываете `ob_start()` с аргументом в виде имени функции, то после отключения буферизации выхода (в данном случае - когда скрипт завершит исполнение) PHP не отправит содержимое буфера сразу. Вместо этого, вызовется зарегистрированная функция с буфером в качестве параметра, и после этого будет отправлено возвращаемое значение этой функции.

Ещё один вариант определения такой функции - директива `output_handler` в `php.ini`. В этом случае все ваши PHP-скрипты будут её использовать.

С этим кодом PHP-скрипт не пошлёт то, что будет сгенерировано потом, а передаст всё функции `compress_output()`, которая, в свою очередь, заархивирует данные; только после этого информация будет отослана - уже в сжатом виде. В нашем примере мы могли бы зарегистрировать сразу `gzencode()`, так как она имеет один параметр и возвращает результат. Для большей ясности я использовал функцию-обёртку (`compress_output()` - прим. пер.). Замечу, что данный пример требует PHP 4.0.4 (который не был выпущен на момент написания статьи) или более поздней версии, скомпилированного с включённой поддержкой `zlib`. Ранние версии PHP могут не поддерживать необязательный параметр для `ob_start()` и не иметь функции `gzencode()`.

Сделаем ещё проще

В PHP 4.0.4 была добавлена новая функция - `ob_gzhandler()`. Она создана для проверки строки `Content-Encoding` заголовка HTTP. Если она находит там `deflate` или `gzip`, то автоматически сжимает данные через поддерживаемый алгоритм кодирования. Использовать её так же просто, как и включить буферизацию выхода - достаточно передать её в качестве параметра для `ob_start()`:

```
ob_start("ob_gzhandler");
```

Или добавить в `php.ini` такую строку (включаем автоматическое сжатие данных):

```
output_handler = ob_gzhandler;
```

Все просто!

А моя жизнь не изменилась!

Хотя буферизация выхода могла и не изменить Вашу жизнь (тогда вы, наверное, счастливый человек), я всё же надеюсь, что вы найдёте ей достойное применение.

Уверен, что мы ещё не раз столкнёмся с кодом, использующим данный инструмент.

Существует достаточно много ситуаций, в которых буферизация выхода может стать незаменимым помощником. Например, очень популярным может стать прозрачное отображение XML-страниц через таблицы стилей XSL. Вместо того чтобы вручную отображать XML через XSL на каждой странице, вы могли бы сгенерировать простые XML-страницы и прогонять их через таблицы стилей XSL, используя функцию буферизации выхода `ob_start()` - и всё это без «мусора» кода отображения в XML-страницах.

И, конечно, самое интересное нас ещё ждёт впереди!

Оригинал статьи находится по адресу: <http://www.zend.com/zend/art/buffering.php>

Шаблонизация на XSLT. Приемы и примеры

Проходясь по страницам нашего форума (<http://www.phpclub.ru/talk>) и отвечая на вопросы, я пришел к выводу, что большинство спрашивающих посетителей в основных чертах знают основы применения XSLT-преобразования, но на практике сталкиваются с рядом вопросов - а как же это использовать? Про XSLT-преобразование написано много статей, и мне не хотелось бы повторяться и пересказывать теорию XSLT. В данной статье речь пойдет о некоторых приемах и хитростях XSLT-разработчика. На такое громкое название, как CookBook (книга рецептов), статья не претендует, но парой хитростей я поделюсь. Статья рассчитана на начинающих пользователей XSLT-шаблонизации, знающих хотя бы ее основы.

Автор:

Александр Календарев
[Alexandre]
akalend@mail.ru

Начнем с азов

Получение выходного HTML-кода получается путем преобразования XSLT-процессором входных XML-данных по XSL-шаблону. Соответственно, умение организовать правильный выходной поток, т.е. наш HTML-код, состоит из умения правильно организовать XML-данные и умения правильно писать шаблоны. (Хотелось бы отметить, что использование XSLT-преобразования не ограничивается генерацией только выходного HTML-кода).

Отсюда напрашивается вывод, что, прежде чем искать ошибку в шаблонах, надо проверить сами XML-данные, т.е. их структуру и состав. Проверка организуется двумя способами: либо выводом в лог-файл, либо выводом XML-данных вместо результата XSLT-процессора.

Если мы используем второй способ (вывод в браузер), то, чтобы эти данные опознал браузер как XML поток и представил в удобопонимаемом виде, необходимо выдать заголовок Content-type: text/xml:

```
<?
. . .

Header ( "Content-type", "text/xml" );
print( $xml );
?>
```

Необходимо отметить, что, когда пишется XSL-шаблон, то это уже не HTML, а XML, и надо руководствоваться правилами валидности XML:

- Каждому открывающему тегу должен соответствовать закрывающий тег. Это, в основном, касается парности таких тегов, как <table>, <td>, <tr>.

- Если тег представлен без пары (одинарный), то он должен иметь закрывающий слеш. Это в основном касается таких одинарных тегов как `
` ``

Практически это означает, что нельзя перемешивать теги, должна быть четкая иерархия вложенности. Например, такие конструкции как, ` bla-bla-bla <i> bla-bla-bla bla-bla-bla</i>` - валидны в HTML, но недопустимы в XML.

Текст спецификации XML (версии 1.0) можно найти по адресу: http://citforum.ru/internet/xml/xml1_1/ или <http://pyramidin.narod.ru/xml/xml1/index.htm>

Использование включений

Вот уже готово ваше преобразование, осталось только придать внешний вид сгенерированной странице. Нет проблем, когда страница одна или две. Проблемы появляются, когда их больше десятка, и весь сайт надо сделать в едином стиле.

Действительно, не будешь же вносить повторяющиеся части HTML-кода в каждый шаблон. В PHP-шаблонизаторах это решалось путем включений:

```
%include header.tpl
....
// текст шаблона
...
%include footer.tpl
```

В XSLT-преобразованиях есть аналогичный механизм `<xsl:include />`.

Имеется файл `main.xml`, который содержит «генеральный» шаблон, единый для всех страниц:

```
<xsl:template match="root">
  <HTML>
  <BODY>
    <div align="center"><b>TABLE OF PRICE </b><br/>
    <table border="0" bgcolor="#000080" cellpadding="1"
cellspacing="1">
....
      <xsl:apply-templates select="item" />

    </table></div>
  </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

В данном шаблоне есть правило: `<xsl:apply-templates select="item" />`, - которое применяется ко всем элементам xml-документа. Данная инструкция, может быть заменена на: `<xsl:call-template name="item" />`.

В нашем преобразовании, должна быть инструкция `<xsl:include href="main.xml" />` и, соответственно, шаблонное правило, определенное в «генеральном» шаблоне:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="Windows-1251"/>

<xsl:include href="main1.xml" />

<xsl:template match="/">
  <xsl:apply-templates select="root" />
  <!--Вызывает правило, определенное в генеральном шаблоне -->
</xsl:template >

<!--Правила, локальных шаблонов -->
<xsl:template match="item">
  <tr bgcolor="#ffffff" align="center">
    <td align="left">
      <xsl:value-of select="description"/>
    </td>
    <td align="center">
      <xsl:value-of select="price"/>
    </td>
  </tr>
</xsl:template >
</xsl:stylesheet>
```

Если мы используем в «генеральном» шаблоне инструкцию `<xsl:call-template name="item" />`, то вместо `<xsl:template match="item">` используем именной шаблон: `<xsl:template name="item">`. В этом случае и технология разработки шаблонов иная.

В любом случае при использовании «генеральных» шаблонов надо придерживаться строго определенного формата выходных xml-данных.

Например, у меня есть следующая структура:

```
<root>
<menu>
  <item name="" />
  <!-- пункты динамически сформированного меню -->
  ...
</menu>
<<action>>
<</action>>
</root>
```

Под `<<action>>` понимается имя тега, соответствующему экшену в модуле, например, для экшена edit будет тег `<edit>`. Но разработка структуры - дело сугубо индивидуальное.

В заключение хотелось бы заметить, что при использовании XSLT-парсера slabotron (версия php 4) необходимо установить базовую директорию, где лежат файлы включений функцией: `xslt_set_base($xh, $filebase);` Где, переменная `$filebase` должна содержать полный путь к директории.

Если что-то не получается.

К сожалению, существующие XSLT-процессоры не имеют средств отладки. Приходится применять «дедовские способы» и «идти по шагам к цели», т.е. идти от разработки общих шаблонов к разработке частных шаблонов. Иначе это называют проектированием сверху вниз.

Как правило, разработка шаблонов соответствует порядку разбора шаблонов, т.е. начинается с корневого элемента xml данных. Может, само обрамление дизайна – это уже дело рук самого дизайнера, но заготовку в виде пустого шаблона `<xsl:template match="root">` надо предусмотреть. И пошли так далее, углубляясь по ходу обработки структуры xml данных.

Если шаблон ничего не выводит, то надо проверить:

- вызывается ли вообще данный шаблон (применяется ли к нему шаблонное правило)
- наличие данных, обрабатываемых данным шаблоном
- результат преобразования.

Первое (вызывается ли вообще данный шаблон) проверяется довольно просто, заменой нашего шаблона конструкцией типа:

```
<xsl:template match="item">
*****
</xsl:template >
```

Если, шаблон вызывался (к нему применялось шаблонное правило), то в результирующем преобразовании увидим наши звездочки.

Несколько труднее определить, какие входные узлы обрабатывал наш шаблон. Это определяется либо копированием обрабатываемых узлов в результирующее преобразование, либо определением имени обрабатываемого тега.

Копирование узла осуществляется заменой нашего шаблона на:

```
<xsl:template match="item">
    <node>
    <xsl:copy-of select="."/>
    </node>
</xsl:template >
```

Результатом применения будет текст, содержащий значение текущего узла, обрамленного тегами `<node>`.

Определение имени обрабатываемого узла осуществляется заменой нашего шаблона на:

```
<xsl:template match="item">
    <name>
    <xsl:value-of select="name(.)"/>
    </name>
</xsl:template >
```

Результатом применения будет текст, содержащий имя текущего узла, обрамленного тегами `<name>`.

Иногда, полезно знать положение узла в контексте, тогда применим функцию `position(.)` к текущему узлу. Использование `position(.)` показывает нам динамику прохода, т.е. применение шаблонных правил к текущей ветке xml-документа. Как правило, функцию `position(.)` применяют в циклах `<xsl:for-each>`, но иногда применяют и к вызовам шаблонных правил.

Проверка результата преобразования - это моделирование части работы шаблона (или всего полностью). Как и всякое моделирование, оно заключается в подготовке входных данных, которые должны представлять часть нашего xml-документа, самого шаблонного правила, т.е. набора части задействованных шаблонов, и, соответственно, простого скрипта преобразования, суть действий которого – взять файл xml-документа с диска, файл xsl-шаблона и вывести результат преобразования.

Ближе к практике. Сортировка.

Одна из часто применяемых практических задач – это вывод отсортированной таблицы, причем критерий сортировки определен данными, т.е. при разработке шаблона мы заранее знаем только перечень критериев, по которым будет производиться сортировка. Для примера возьмем прайс-лист, который можно отсортировать по:

- цене;
- наименованию товара;
- наименованию группы товаров.

Если в шаблоне не запланировано вывода нескольких таблиц, то имя критерия сортировки можно содержать в корневом элементе. В случае, если таблиц несколько, то имя критерия сортировки, можно хранить в атрибуте корневого элемента фрагмента данных таблицы.

Лично я использую для этого корневой элемент. Допустим, имеется следующая структура данных:

```
<root sort="type">
. . .
  <table>
    <item price="12" type=" Сувениры" name="Сувенир Мозайка">
    <item price="13" type=" Сувениры" name="Сувенир Шкатулка">
. . .
    <item price="85" type="Фотопленки" name="Пленка 12/200 Kodak">
  </table>
</root>
```

Значение атрибута `sort` корневого элемента `root` определяет, какой тип сортировки использовать. В данном примере используется сортировка по категориям товаров (`type`).

Шаблон, обрабатывающий тег <table> и «строющий» таблицу, будет выглядеть следующим образом:

```
<xsl:template match="/">
<html>
  <body>
    <xsl:apply-templates select="root/table" />
  </body>
</html>
</xsl:template >

<xsl:template match="table">
  <xsl:variable name="sort" select="//root/@sort"/>
  <table>
    <xsl:for-each select="item">
      <xsl:sort select="@name[$sort='name']"/>
      <xsl:sort select="@price[$sort='price']"/>
      <xsl:sort select="@type[$sort='type']"/>
      <tr>
        <td><xsl:value-of select="@name"/></td>
        <td><xsl:value-of select="@type"/></td>
        <td><xsl:value-of select="@price"/></td>
      </tr>
    </xsl:for-each >
  </table>
</xsl:template >
```

Для реализации данного шаблона использован XSLT-элемент <xsl:variable>, который определяет XSLT переменную \$sort. Данная переменная содержит значение атрибута sort корневого элемента root.

Для определения критерия сортировки использовались предикаты выборки. При вычислении предиката его результат приводится к булевому типу. Синтаксически предикат заключен в квадратные скобки, т.е. конструкция типа: <xsl:sort select="@type [\$sort='type']"/> - показывает, что сортировка будет осуществляться для всех узлов по ключу, соответствующему атрибуту @type, при условии, что значение переменной \$sort будет равно стоковому выражению «type». Соответственно, по каждому критерию, по которому будет осуществлена сортировка, необходима своя команда <xsl:sort>.

Заключение

Конечно, кол-во всяких методов и приемов при разработке XSLT шаблонов множество и «нельзя объять необъятное». В настоящее время даже появились разные стили написания шаблонов. Я попытался охватить лишь первые этапы «Путешествия в страну Шаблонизации».

В данной статье были отражены в основном главные проблемы, с которыми сталкиваются юные XSLT-шаблонизаторы. Планируется продолжение.

Уже после написания данной статьи автор узнал о существовании инструментария XSLT-разработчика, включающего средства XSLT-отладки. Это такие программные продукты, как: xselerate, Stylus Studio, Altova XML StyleVision. Данные программные продукты являются платными.

Автор будет благодарен за любую критику данного материала, а также будет рад услышать ваши пожелания и узнать, с какими же вы сталкивались трудностями при разработке XSLT-шаблонов. Все замечания будут учтены при подготовке следующей статьи из цикла: «Шаблонизация на XSLT».

В прошлом номере журнала (PHP Inside №8) мы открыли новый раздел журнала под названием «Форум». Содержанием этого раздела стали заметки и советы, собранные в форумах <http://phpclub.ru> и <http://php.com.ua> самими участниками этих популярных сообществ. Новый раздел не остался незамеченным – мы получили отклики, и в этот раз решили развить прошлый опыт и снова обратились за помощью к двум сообществам. Результат публикуем ниже.

По материалам:

<http://phpclub.ru>

<http://php.com.ua>

Работа над ошибками. Подход №1

На форуме <http://php.com.ua> встретила ссылка на оригинальный подход к обработке ошибок (Warnings and Notices). Он замечателен тем, что позволяет избежать прямого вывода в браузер стандартных сообщений об ошибках и, приукрасив их, помогает вывести с некоторым «графическим интерфейсом», сделав разработку чуточку приятнее. Однако, как говорится в одном анекдоте, «мы его не за это любим». Вам, думаю, будет интересно посмотреть на принцип его работы.

Итак, листинг файла `buger.php`:

```
<?
$_error['path']='buger/';

//$_error[2]='red_sign_exc.gif';
//E_WARNING      Report nonfatal errors at runtime
//$_error[8]='blue_bolt.gif';
//E_NOTICE
//Report notices, notifications that something you have done might be an error

$_error[2]='baloon_x.gif';
//E_WARNING      Report nonfatal errors at runtime

$_error[8]='baloon_exc.gif';
//E_NOTICE
// Report notices, notifications that something you have done might be an error

//$_error[2]='red_bug.gif';
//E_WARNING      Report nonfatal errors at runtime
//$_error[8]='yellow_black_bug.gif';
//E_NOTICE
//Report notices, notifications that something you have done might be an error

error_reporting(E_ALL);
set_error_handler("buger");
```

Листинг 1. Продолжение листинга на следующей странице.

```
function buger($errno, $errstr, $errfile, $errline){
global $_error;
$type="";
if ($errno==8) $type='NOTICE: ';
if ($errno==2) $type='WARNING: ';
$p=$_error['path'].$_error[$errno];
?><IMG SRC='<?=$p?>' BORDER='0'
TITLE="<?=$type?><?=$errstr?>
File:<?=$errfile?>
at Line:<?=$errline?>"
HSPACE="0" VSPACE="0"><? }
?>
```

Листинг 1. Окончание .

Файл `buger.php` и упомянутые в листинге графические файлы вы также найдете в приложении к журналу или по оригинальной ссылке <http://apps.webtrox.com/buger.zip>

Этот файл достаточно включить с помощью `include` в любой PHP-файл, и он поможет отследить ваши ошибки.

Работа над ошибками. Подход №2

Все актуальнее становится разработка веб-приложений с использованием PHP 5. Как известно, в новой версии PHP есть возможность создавать собственные обработчики ошибок с помощью, например, `try { }... catch { }` и предопределенного класса `Exception`.

Вывод ошибок на экран можно разобрать с помощью класса `myException`. Вот его листинг (вы найдете его также и в приложении к журналу):

```
<?php
/*
  SyBe engine for Zend PHP5 Contest
  Andrey Korolev      begemot@php.com.ua   ICQ 70901279
  Dmytro Sychevsky   sych@php.com.ua      ICQ 75339803

  http://www.php.com.ua
*/
class myException extends Exception {
    private $except_dom;
    private $debug_level;
function __construct($exception,$debug_level = 0) {
    $this->message = $exception;
    $this->debug_level = $debug_level;
    $this->create_except_dom();
    $this->display();
}
function display(){
    $xslt = new DOMDocument;
    $xslt->load($_SERVER['DOCUMENT_ROOT'].'/xslt/exception.xslt');
    $xslt_proc = new xsltProcessor;
    $xslt_proc->importStyleSheet($xslt);
    echo $xslt_proc->transformToXML($this->except_dom);
}
}
```

Листинг 2. Окончание листинга на следующей странице.

```

function create_except_dom() {
    $this->except_dom = new DOMDocument("1.0");
    $root = $this->except_dom->createElement("exception");
    $this->except_dom->appendChild($root);
    if ($this->debug_level >= 0) {
        $mess = $this->except_dom->createElement("message", $this->message);
        $root->appendChild($mess);
    }
    if ($this->debug_level >= 1) {
        $file = $this->except_dom->createElement("file", $this->file);
        $root->appendChild($file);
        $line = $this->except_dom->createElement("line", $this->line);
        $root->appendChild($line);
    }
}

function Error2Exception($errno, $errstr, $errfile, $errline) {
    throw new MyException($errstr);
}
?>

```

Листинг 2. Окончание.

Из кода веб-приложения он может использоваться следующим образом:

```

try {
    //какой то php код
    $sql = "select * from tbl";
    // сюда же можно прилепить сообщение об ошибке
    if (!mysql_query($sql)){
        //второй параметр debug level (0,1)
        throw new myException(mysql_error(), '1');
    }
} catch (myException $e){ }

```

XSL-файл:

```

<?xml version="1.0" encoding="windows-1251"
<xsl:stylesheet version = '1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
<html><head><title>Fatal Error</title></head>
<body bgcolor="#eaeaea" text="#000000" id="all">
    <table border="0" cellspacing="1" cellpadding="10" bgcolor="#333333">
        <tr><td bgcolor="#BEBEBE" align="center"><p>
            <xsl:value-of select="exception/message" disable-output-escaping="yes"/>
            <xsl:if test="exception/file">
                <br/> File : <xsl:value-of select="exception/file"/><br/>
            </xsl:if>
            <xsl:if test="exception/line">
                Line : <xsl:value-of select="exception/line"/>
            </xsl:if><br/><br/></font>
            <a href="javascript:history.back()">BACK</a>
        </p></td></tr></table></body></html>
</xsl:template>
</xsl:stylesheet>

```

Постраничный вывод данных из MySQL

Существует множество вариантов постраничного вывода данных из MySQL. Приводим код одного из них.

```
/*постраничный вывод: */
/* $page = номер страницы , которую пытаются выбрать */
/* $rows_in_select = количество результатов на странице */
/* $sql = Ваш запрос к базе , который пытаетесь вывести постранично*/
/* $db_link = линк на открытую БД */
function query_page ($page, $rows_in_select, $sql, $db_link) {
    /* получили количество рядов */
    if (--$page < 0) $page=0;
    $pre_sql = preg_replace("/SELECT (.*) FROM/i", "SELECT count(*) AS
total_rows FROM", $sql);
    $result = mysql_query($pre_sql, $db_link);
    if (empty($result)) return NULL;
    $total_rows = mysql_fetch_assoc ($result);
    $total_rows = $total_rows['total_rows'];

    if ($page*$rows_in_select > $total_rows ) $page = 0;
    $post_sql = $sql.' LIMIT ' .($page*$rows_in_select).',
'. $rows_in_select;
    return mysql_query($post_sql, $db_link);
}
```

Вывод данных из файла «через строку»

Задача: вывести данные из текстового файла на экран через строку. Т.е. из исходного файла (voda.txt) с содержанием:

```
Я
не
люблю
эту
воду
```

необходимо получить строку:

```
Я люблю воду
```

Задача решается очень просто посредством следующих строк кода:

```
$lines = file('voda.txt');
for ($i=0,$lines_cnt=count($lines);$i<$lines_cnt;$i+=2) echo $lines[$i];
```

План врезок

Как запретить доступ к таблицам PostgreSQL, оставив возможность работы с хранимыми процедурами.....	3
Удаление повторов в поисковых запросах с помощью array_flip(\$arr).....	3
Options Multiviews в apache.....	4
Фильтрация элементов массива с помощью array_filter().....	4
Работа с переменными в стиле register_globals = On при register_globals = Off.....	6