

Январь - Февраль '06 № 17

php {**INSIDE**}

электронный журнал для веб-разработчиков



В ОЖИДАНИИ
PHP 6

Содержание

Анонс.....	2
Биржа Труда.....	5
Тема с обложки	
Готовьтесь к PHP 6.....	7
Идеи	
Аспектно-ориентированная веб-разработка и PHP.....	11
Создание дружественных URL.....	21
Лучшая практика.....	26
Persistent Objects в ezComponents.....	35
Кеширование средствами HTTP.....	41
Виртуальная файловая система. Внедряем в веб-сайт.....	46
HowTo	
Конвертация из разных систем счисления.....	49

Анонс: Четвертая PHP-конференция пройдет в Москве

За последние несколько лет программирования PHP прошел значительный эволюционный путь и всё еще продолжает развиваться. Развивается сам язык, растет и спектр применения PHP — как на предприятиях малого и среднего бизнеса, так и в крупнейших транснациональных корпорациях.

В 2003 году международный клуб веб-разработчиков PHPClub <http://phpclub.ru> принял решение проводить ежегодные конференции, посвященные всем аспектам применения PHP-технологий, на которых ведущие разработчики, аналитики и менеджеры проектов пост-советского пространства могли бы обмениваться новыми идеями и методиками производства программного обеспечения.

За два года наши конференции стали заметным событием в среде профессионалов веб-программирования, предоставляя уникальный шанс продемонстрировать свой потенциал и познакомиться с новыми технологиями и последними тенденциями в области веб-разработки.

Мы рады сообщить вам, что следующая PHP-конференция пройдет в мае 2006 г. в Москве.

Тематика конференции

- Применение PHP в веб-разработке: приёмы, методы и парадигмы программирования в PHP, полезные модули и библиотеки.
- Эффективное создание приложений: фреймворки, гибкие методологии, экстремальное программирование, TDD-методики.
- PHP и корпоративные информационные системы: разработка систем масштаба предприятия, интеграция корпоративных ИС.
- Будущее PHP: основные направления развития, PHP6, мультязычные приложения, Unicode.

Совместно с конференцией пройдет серия мастер-классов (практических занятий)

- Test driven development (TDD) — 2.5 дня: приемы, паттерны, тестирование операций с БД, рефакторинг, функциональное тестирование, использование SimpleTest и Selenium.
- Разработка расширений PHP — 0.5 дня:
- Объектно-ориентированные модули изнутри, портирование PHP классов на C, Zend API.

Если вы планируете участвовать в мастер-классах, просим вас учесть, что число мест на каждый мастер-класс ограничено (~24 чел.), поскольку мастер-классы ориентированы на большое количество практики.

Участие в мастер-классах оплачивается отдельно от конференции и требует наличия ноутбука с определенным ПО (Apache+MySQL+PHP5+клиент cvs).

Основные даты

1 марта — рассылка третьего информационного сообщения с программой работы конференции.

1 апреля — прием докладов и оплата участия по программе предварительной регистрации (действуют скидки корпоративным участникам, от 3-х и более человек).

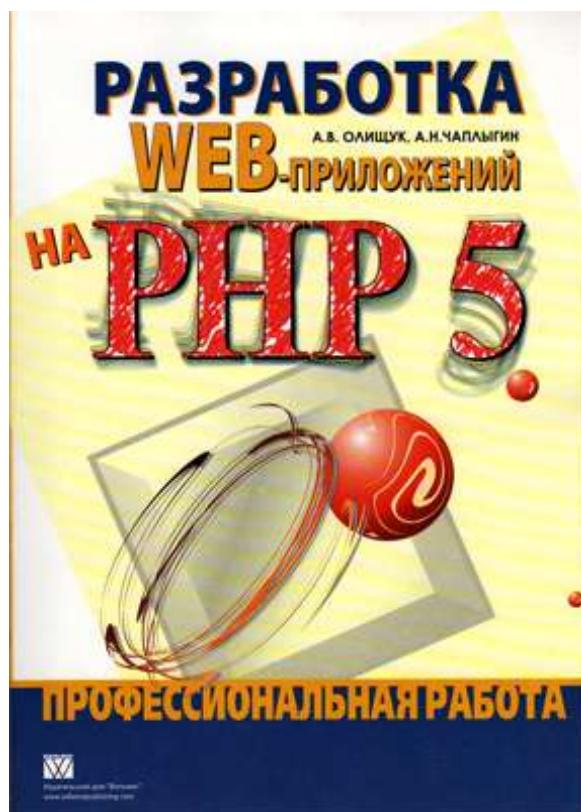
Середина мая - мастер-классы (3дня), рабочие дни конференции (2дня)

Наши координаты

Адрес электронной почты оргкомитета: 2006@phpconf.ru.

По вопросам спонсорского участия обращайтесь по адресу: sp2006@phpconf.ru.

Официальный сайт PHP-конференции <http://www.phpconf.ru/>



Представляем новую книгу о PHP 5

Книга писалась сразу о PHP 5 (в т.ч. PHP 5.1), а не перерабатывалась из изданий о PHP 4, что несомненно является ее большим плюсом.

Отличительной чертой так же является большое количество лаконичных примеров кода. Здесь вы практически не найдете длинных "портянок" листингов, в которых сложно разобраться, но с другой стороны, каждая функция или понятие рассмотрено на небольших примерах.

Книга будет интересна как новичкам PHP-программирования, так и тем, кто уже сделал свои первые шаги и приблизился к среднему уровню знаний.

Новички найдут здесь как теоретические основы (здесь поясняется что та

кое массивы, функции, объекты, циклы, рекурсия и т.д), так и практические знания. Примеры кода дают возможность сразу "поиграться" с функционалом PHP. Помимо этого, довольно подробно расписан процесс установки MySQL, Apache и PHP на Windows и Linux. Это поможет новичкам сразу разобраться - где здесь веб-сервер, а где база данных.

Разработчикам поопытней, наверняка будут интересны темы объектного программирования (с пояснений таких понятий как инкапсуляция, полиморфизм, интерфейсы...), работа с СУБД через PDO, стандартная библиотека SPL, системы исключений, работа с расширением MySQLi и создание веб-сервисов (как клиентов, так и серверов). Много внимания уделено DOM и SimpleXML (к примеру, как расширять функционал SimpleXML-объектов за счет наследования и проч).

Купить книгу онлайн можно в магазине Bolero (самый экономный вариант): <http://www.bolero.ru/basket/basket.php?pid=39737674&partner=phpinside>

Посмотреть подробное оглавление, можно на сайте издательства «Вильямс»: <http://www.williamspublishing.com/cgi-bin/heading.cgi?isbn=5-8459-0944-9>

Вакансии с <http://phpclub.ru>

Веб-программист, Москва \$1000

В компанию "Компания "Камео"" (Russia,Москва) требуется веб-программист

Зарплата от \$1000;

Занятость Полная (основное место работы);

Возраст:от 25 до 40;

Пол:не имеет значения;

Образование:не имеет значения;

Опыт работы:1 год

Языки и технологии: PHP, XML, XSL, XSLT,

СУБД: MySQL,

Прочие требования: Плюсом является знание PostgreSQL и умение писать хранимые процедуры.

Пользователь Unix/Linux

Обязанности: Разработка и модернизация сайтов

Условия работы: ст.м. Нахимовский Проспект. Рабочий день с 9 до 19. Бесплатные обеды.

Контактная информация: Жердев Сергей, 7462488, 7462430, .

WEB-сайт компании : <http://www.quickmarket.ru>, www.kameo.ru/

Вакансия предоставлена прямым работодателем 22-02-2006 и будет актуальна до 24-03-2006

Веб-разработчик, США, Бостон (\$5000)

В компанию "Интернет-Ходинг" (United States,Boston) требуется WEB-разработчик (США, Бостон)

Зарплата от \$5000;

Занятость Полная (основное место работы);

Возраст:от 25 до 34;

Пол:мужчина;

Образование:Высшее;

Опыт работы:5 лет

Языки и технологии: PHP,

СУБД: MySQL,

Операционные системы: Linux,

Прочие требования: Международной компании требуется разработчик программного обеспечения с опытом работы и способный к работе в многозадачном режиме. Это великолепная возможность войти в реальный мир Интернет-коммерции и работать в команде профессионалов. Требования к кандидату: - Технологии: PHP, MySQL, XML/XSLT, OO Concepts, Apache, Linux/Windows; - Опыт проектирования и разработки систем e-Commerce и веб-приложений: "Catalog management, Order processing, Payment systems, CRM, CMS, Affiliates" и др.

Обязанности: Обязанности: - проектирование систем электронной коммерции и "Community" систем; - бизнес анализ; - разработка, внедрение и поддержка.

Условия работы: Условия и возможности: Компания спонсирует визу Н1В для вас и членов вашей семьи, прибытие, размещение и проживание. Вознаграждение включает в себя основную зарплату + бонусы. Для инициативных кандидатов мы обещаем быстрый профессиональный рост. Наша компания расположена в г. Бостоне, США и состоит из дружного и творческого коллектива. Ждем вас в нашей команде профессионалов!

Контактная информация: Мария, 8903-710-13-31, .

WEB-сайт компании : <http://www.unipointhome.com/>

Вакансия предоставлена прямым работодателем 22-02-2006 и будет актуальна до 24-03-2006

Веб-программист (BITRIX), Удаленно, Москва (\$1200)

В компанию "ООО Целевое управление" (Russia, Москва) требуется **Web-программист (BITRIX)** - удаленный проект

Зарплата от \$1200;

Занятость Удаленная работа (freelance);

Возраст:от 25 до 40;

Пол:не имеет значения;

Образование: не имеет значения;

Опыт работы:1 год

Языки и технологии: HTML, DHTML, CSS, Java, Perl, CGI, PHP, XML, XSL, XSLT,

СУБД: MS SQL Server, MySQL, Oracle, PostgreSQL, SQLBase,

Обязанности: График работы: свободный/удалённая работа Опыт работы: желательно - минимум один завершённый проект с использованием BITRIX Описание работы 1.Создание сайта, на основе готового технического задания. 2.Немного программирования, в основном сборка и реализация стандартных функций самого BITRIX. 3.Интернет магазин, каталог товаров, новостные блоки, статьи, сопутствующие товары/статьи. Уже реализовано 1.Форум 2.HTML - вёрстка вся 3.Авторизация 4.Новостные блоки 5.Техническое задание

Условия работы: Бюджет \$1200 Для тех кто знает о чём речь, работа на 2 недели максимум. В случае досрочной сдачи работы возможны бонусы.

Контактная информация: Марина, 797-47-17, .

WEB-сайт компании : <http://www.Lityagin.ru/>

Вакансия предоставлена прямым работодателем 21-02-2006 и будет актуальна до 23-03-2006

Веб-программист, Минск (\$400)

В компанию "Artics - Минск" (Belarus, Минск) требуется **PHP- программист**

Зарплата от \$400;

Занятость Полная (основное место работы);

Возраст:от 21 до 35;

Пол: не имеет значения;

Образование: Высшее;

Опыт работы:1 год

Языки и технологии: HTML, DHTML, CSS, PHP, UML,

СУБД: MySQL, PostgreSQL,

Операционные системы: Linux,

Прочие требования: * Опыт разработки WEB приложений на PHP не менее года * Хорошее знание SQL * Знание и умение использовать ООП * Опыт работы в команде

Обязанности: Проектирование классов, частично архитектуры, написание кода. Работка сайтов, систем.

Условия работы: Испытательный срок 1-2 мес. Зарплата по результатам собеседования. Отпуск, больничные -- все как полагается.

Контактная информация: Юрий Шиляев, 625-01-28, .

WEB-сайт компании : <http://minsk.artics.ru/>

Вакансия предоставлена прямым работодателем 22-02-2006 и будет актуальна до 24-03-2006

Готовьтесь к PHP 6

Автор: Richard Davey

Перевод: Андрей Олищук

Окончательные релизы PHP 6 появятся не скоро, но профессиональные разработчики должны знать к чему готовиться...

Вы уже, наверное, в курсе, что группа разработчиков ядра PHP 6 встречалась в ноябре 2005 года в Париже. Это была захватывающая встреча, на которой разработчики обсудили развитие PHP со своей точки зрения. Прежде чем я начну свой рассказ о принятых решениях, вынужден предупредить – все что здесь перечислено, это не «100% решенные вопросы» и в окончательном релизе PHP 6 дела могут обстоять иначе. Конечно же, все эти моменты будут обсуждаться подробнее, но уже сейчас можно говорить о перспективах.

Юникод

Поддержка юникода в настоящий момент может быть установлена только на уровне «per request», т.е. для каждого запрашиваемого файла. Это означает, что PHP придется сохранять варианты классов, имен методов и функций одновременно в таблице символов Unicode и в non-Unicode, что, конечно же, увеличивает количество потребляемых ресурсов. Разработчики решили сделать настройку юникода на уровне всего сервера, а не запроса. Отключение поддержки юникода, если таковая не требуется, может увеличить производительность строковых функций до 300% и приложений в целом до 25%. Вынос настроек в `php.ini` позволит снять заботу о юникоде с разработчика и переложить ее на администраторов хоста.

Если вы самостоятельно собираете PHP и несете всю ответственность за ваши серверы, то вам будет полезно знать, что для PHP 6 потребуются библиотеки ICU (в зависимости от того, будет ли нужен Unicode или нет).

Register Globals уйдут в прошлое

Попрощайтесь с этой настройкой, она будет окончательно убрана. Такой настройки больше не будет в `php.ini` и если вы ее внесете, то получите ошибку уровня `E_CORE_ERROR`. Это означает, что PHP 6 наконец-то завершит эру скриптов PHP 3. Это серьезный, но очень нужный шаг.

Magic Quotes будут исключены

Опция `magic quotes` будет так же исключена из PHP и при попытке включения такой настройки, «выскочит» ошибка `E_CORE_ERROR`. Это повлияет на `magic_quotes`, `magic_quotes_sybase` и `magic_quotes_gpc`.

Больше не будет Safe Mode

Это понравится тем, кто хостится на серверах с обязательным включением Safe Mode. Теперь, включение опции будет вызывать ошибку E_CORE_ERROR. Причиной для этого становится механизм посылки «сигнала опасности», который делает PHP более безопасным. Сохранится лишь open_basedir.

'var' будет полным алиасом 'public'

Var используется в классах PHP 4. В объектном подходе PHP 5, употребление var вызывает ошибку уровня E_STRICT. В PHP 6 предупреждение об ошибке будет устранено и var станет полным синонимом public. Это вполне нормальное решение, однако, те, кто «подогнал» свои скрипты под PHP 5, сделали в этом плане лишнюю работу.

Возврат по ссылке вызовет ошибку

И '\$foo =& new StdClass()' и 'function &foo' теперь вызовут ошибку уровня E_STRICT.

Будет убрана совместимость с zend.ze1

Ze1 всегда пытался поддерживать старое поведение PHP 4, но не всегда «работал на 100%», поэтому в шестой версии PHP он будет полностью исключен.

Поддержка Freetype 1 и GD 1 будет убрана

Поддержка обоих (очень-очень старых) библиотек будет убрана.

dl() останется только в SAPI

Каждую функцию SAPI нужно будет регистрировать отдельно и только CLI и embed SAPI будут работать с этого момента. В других вариантах dl() работать не будет.

FastCGI всегда включена

FastCGI код будет «вылизан» и станет включенным по-умолчанию для CGI SAPI. При этом, поддержку FastCGI будет невозможно отключить.

Старые названия глобальных массивов будут удалены

Помните старые массивы HTTP_*_VARS? Если вы еще не начали использовать \$_GET и \$POST, то начните делать это прямо сейчас, потому что в PHP 6 эти массивы будут вызывать E_CORE_ERROR.

Перемещения расширений

Расширения XMLReader и XMLWriter войдут в дистрибутив и станут доступны по умолчанию. Расширение ereg для работы с регулярными выражениями переедет в PECL (т.е. будет удалено из PHP). Это означает, что PCRE будет по умолчанию недоступно и выключено. Такой шаг делается для включения нового расширения для работы с регулярными выражениями, основанными на ICU. Чрезвычайно полезное расширение Fileinfo будет включено в дистрибутив и доступно по умолчанию.

Дополнения к движку PHP

64 битный целочисленный тип данных

В движок будет добавлен новый тип данных - int64. Он будет использоваться по умолчанию для integer.

Goto

Никаких команд goto не будет добавлено. Однако, команда break расширится статической меткой, поэтому будет возможно написать break foo и это перекинет на метку foo: в вашем коде.

ifsetor()

Похоже, что мы не увидим эту функцию, что обидно. В операторе ?: можно будет опускать один параметр, что позволит писать так: "\$foo = \$_GET['foo'] ? : 42;" (т.е. если foo является истиной, то \$foo будет равно 42). Это экономит немного кода, однако будет не так читаемо, как при использовании ifsetor().

foreach для многоуровневых массивов

Это отличное новшество – вы сможете проходить с помощью foreach по нескольким уровням массива, к примеру "foreach(\$a as \$k => list(\$a, \$b))".

{} против []

Сейчас вы можете использовать и {} и [] для обращения к строковым индексам. Однако, запись {} уже сейчас вызовет E_STRICT в PHP 5.1 и будет полностью устранена в PHP 6. К тому же, [] частично заменят substr и array_slice и вы сможете использовать "[2,]" для получения символов от второго и до конца строки. Очень удобно.

Изменения в объектном стиле

Связка static

Будет добавлено новое ключевое слово для доступа к последующей связке – static::static2(), что позволит управлять static во время выполнения.

Пространства имен

Этот момент остается нерешенным и по сей день. Мой совет – не задерживайте дыхание.

Типизированные значения при возврате из функций

Разработчики высказались против типизации, потому как это «не в стиле PHP». Однако такая возможность будет добавлена, но не решен вопрос ее синтаксиса. В любом случае, это будет полезная возможность.

Вызов динамических функций как статических приведет к E_FATAL

Сейчас вы можете вызывать статические и динамические методы не обращая внимание на то, статические они или динамические. Вызов динамической функции как статической, вызовет E_FATAL.

Добавления в PHP

APC войдет в ядро

Работа APC с байткодом будет включена в основную поставку PHP в качестве стандарта, но, вероятно, не будет активизирована по-умолчанию, но результаты ее работы будут стимулировать хостеров включать эту опцию.

Hardened PHP патч

Этот патч выполняет большое количество дополнительных проверок на безопасность. Разработчики тщательно изучают этот патч и некоторые элементы найдут свое место в PHP: защита от разделения HTTP-запроса, allow_url_fopen будет разделена на две: allow_url_fopen и allow_url_include. Первая опция будет по-умолчанию включена, а вторая – отключена.

E_STRICT войдет в E_ALL

Вау, это серьезная штука! Сообщения об ошибках наконец-то войдут в E_ALL по умолчанию. Это демонстрирует старания разработчиков научить «лучшей практике программирования» посредством сообщений «Эй, ты делаешь неправильно!».

Прощайте asp-тэги <%

Будет удалена поддержка ASP-тэгов, но останутся короткие PHP-тэги <? ?>.

Заключение

PHP 6 движется в интересном направлении – разработчики PHP начали направлять программистов на правильный путь, вместо того чтобы упреждать «так делать не надо, потому что это устарело». Категоричное искоренение таких моментов как register globals, magic quotes, long arrays, {} string indexes и call-time-pass-by-references наконец-то заставит программистов «вычистить» свой код. Плохо ли это? Я так не думаю, но все это сделает переход на PHP 6 еще более медленным, чем мы наблюдаем с PHP 5 (который итак происходит ужасно!).

Потратьте немного времени на чтение официальной информации: <http://www.php.net/~derick/meeting-notes.html>

Оригинал статьи: <http://www.corephp.co.uk/archives/19-Prepare-for-PHP-6.html>

Аспектно-ориентированная веб-разработка и PHP

Автор: Дмитрий Шейко

Как можно использовать AOSD в PHP проектах уже сегодня

Уже много лет объектно-ориентированный подход к программированию пользуется широкой популярностью. В небольших краткосрочных проектах едва ли будут заметны его преимущества, но без него любой крупный проект фактически обречен. Именно объектно-ориентированные языки программирования имеют все необходимое, для того, что бы представить бизнес-логику проекта в наглядном виде. Даже при проектировании самой логики системы ныне напрашивается диаграмма классов UML. Наглядная бизнес логика позволяет легко включаться в проект новым участникам, бережет время авторам кода, вернувшись в проект после длительного перерыва. Наглядная бизнес логика ощутимо сокращает число ошибок в проекте. Но достаточно ли использования объектно-ориентированного подхода к программированию для того, что бы достичь столь желанной наглядной бизнес логики? Очевидно - нет. Добиться изящной объектно-ориентированной программной архитектуры достаточно сложно. Но если вы использовали приемы из книги "Refactoring: Improving the Design of Existing Code", Martin Fowler, возможно вам это удалось.

Однако даже теперь мы можем найти в коде сквозную функциональность (crosscutting concerns), участвующую в самых различных классах (протоколирование, кеширование, синхронизация, трассировка, контроль безопасности, контроль транзакций). Организовать подобную программную логику поможет AOSD (Аспектно-ориентированная разработка программного обеспечения, http://en.wikipedia.org/wiki/Aspect-oriented_programming).

Дмитрий Шейко занимается разработкой программного обеспечения с 1987 года. Начиная с 1998 года опубликовал более 50 технических статей в специализированных изданиях. С 2001 года разрабатывает архитектурные решения и инструментальные средства для управления содержанием (Content Management, CMF, ECM). За прошедшее время спроектировал ряд успешных коммерческих продуктов среду разработки веб-приложений Site Sapiens (www.sitesapiens.com). В 2004 году разработал и опубликовал спецификацию универсального языка для разработчиков CMS XML Sapiens (www.xmlsapiens.org).



Что такое AOSD?

Аспектно-ориентированная разработка программного обеспечения (AOSD) относительно новая парадигма разработки бизнес приложений. Основа данного подхода – Аспект. Это точка зрения, с которой может быть рассмотрено какое-либо понятие, процесс, перспектива.

Что бы быстрее вникнуть в суть подхода давайте, рассмотрим веб-сайт в различных аспектах. Информационная архитектура описывает сайт в аспекте организации его структуры. Юзабилити описывает сайт в аспекте удобства его использования. Графический дизайн представляет сайт в аспекте его визуального восприятия. Функциональная модель описывает сайт в аспекте его бизнес-логики.

Все это различные составляющие процесса разработки веб-сайта, каждая из которых требует специфичных ресурсов, подходов и реализаций. Успешный проект подразумевает качественное решение со стороны каждого из этих аспектов. Если кому-либо данный пример покажется сложным, можно обратиться к более простой и универсальной схеме. Когда проектируется жилой дом, архитекторы проектируют каркасный чертеж, далее подготавливается схема электропроводки, схема водоснабжения и прочее.

Очевидно, что каждый этап является самостоятельным, но необходимым для успеха всего проекта. Каждый этап – это аспект, в котором можно рассматривать проект. Как это ни банально, но в разработке программного обеспечения может быть использован тот же принцип выделения аспектов бизнес логики приложений. Более 20 лет назад Bjarne Stroustrup (http://en.wikipedia.org/wiki/Bjarne_Stroustrup) воплотил в C++ компоновку программного кода в наборы логических объектов, поведение которых и соотношение друг с другом может быть определено различным образом (наследование, икапсуляция, абстракция, полиморфизм). Это надежная, проверенная временем парадигма разработки программного обеспечения – объектно-ориентированное программирование.

Однако этот подход имеет свои ограничения в декомпозиции аспектов бизнес логики приложения. За прошедшее время было разработано множество новых подходов для преодоления данных ограничений. Среди них можно назвать адаптивное программирование, композиционные фильтры, гиперпространства, аспектно-ориентированное программирование, моделирование ролей, предмет-ориентированное программирование и т.д. Последнее время подобные изыскания стали подаваться под эгидой аспектно-ориентированной разработкой программного обеспечения.

Как уже говорилось выше, код сквозной функциональности, так или иначе, будет распределен по различным модулям, что наихудшим образом скажется на качестве программного обеспечения с точки зрения наглядности бизнес логики, ее адаптации и способности к развитию. Задача AOSD в том, что бы выделить сквозную функциональность и вынести ее за пределы бизнес логики приложений.

Давайте представим какое-нибудь комплексное веб-приложение, например CMS, и рассмотрим на каком этапе и каким образом мы можем столкнуться с указанными проблемами. Допустим некоторое число функций системы, требует преобразования входных данных из XML в массивы. Так как в данном случае аспект XML-парсинга затрагивает лишь небольшое число функций и не предполагает существенного развития.

Как мы видим, здесь не требуется расширенной декомпозиции, нет очевидной необходимости в использовании AOSD. Логичнее всего определить эту процедуру в метод корневого класса (или же внешнего класса, смотря по обстоятельствам) и вызывать его по мере необходимости.

Однако если же речь идет о мониторинге производительности и наша задача – по требованию снять показания со всех выполняемых функций на их входа и выходах едва ли помогут какие-либо манипуляции с объектами. Впрочем, дотошный читатель предложит воспользоваться предыдущим примером в масштабе всего приложения и в дальнейшем использовать триггер для включения и выключения сбора статистических данных. Остается его предупредить, что когда вдруг возникнет необходимость задействовать протоколирование системных транзакций для заданных случаев, при данном подходе, придется вспоминать все детали программной архитектуры и реорганизовывать ее.

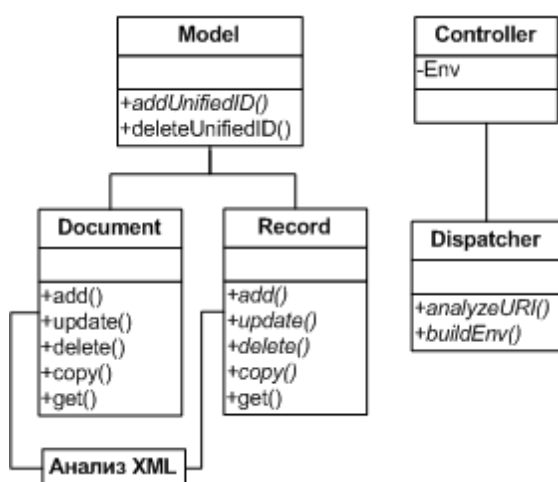


Рис. 1. Приемлемая декомпозиция.

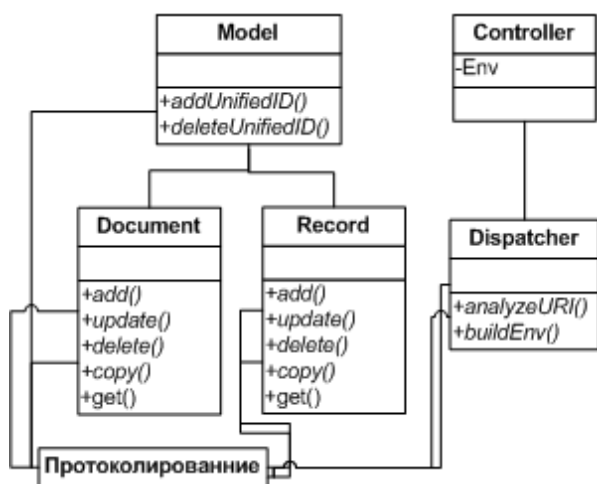


Рис. 2. Неприемлемая декомпозиция.

Насколько было бы удобнее попросить систему в рамках некоторого аспекта должным образом обслужить заданные события в определенных объектах. Скажем, проходит время для нашего большого и сложного проекта приходят новые требования по безопасности. Мы пишем процедуру дополнительных проверок безопасности и требуем от системы в рамках аспекта безопасности при вызове заданных методов выполнять данную процедуру.

Парадигма аспектно-ориентированной разработки программного обеспечения подразумевает это. Мы выделяем новый уровень абстракции вне существующей программной архитектуры и декларируем в нем функциональность в том или ином отношении применимую к системе в целом. Как видно из рисунка 3 мы можем выделить программные процедуры, обслуживающие систему, например, в рамках аспекта безопасности и вынести их из основных классов.

Далее мы можем проделать тоже в отношении процедур аспекта контроля входных данных. Таким образом, при каждом проходе мы освобождаем бизнес-логику системы, делаем ее более наглядной. Фактически мы получаем в результате наглядную бизнес-логику в основных классах системы и аккуратно расфасованную сквозную функциональность вне базовой модели.

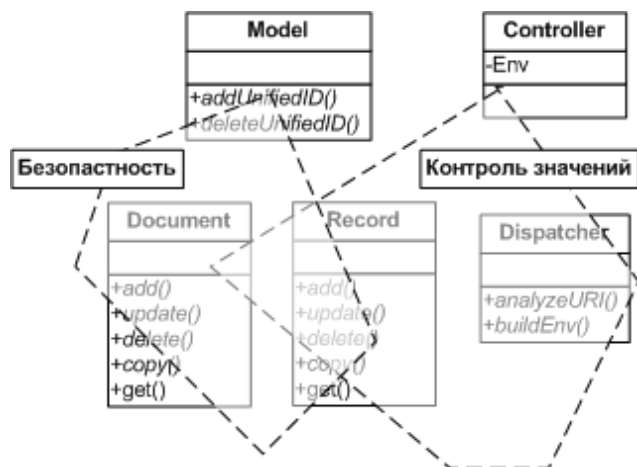


Рис. 3. Аспектная декомпозиция.

Итак, резюмируем:

- Основа аспектно-ориентированного подхода в идентификации общности программного кода в рамках каких-либо аспектов и вынесение выделенных процедур за пределы основной бизнес-логики;
- Процесс аспектной ориентации и разработки программного обеспечения может включать моделирование, дизайн, программирование, обратный инжиниринг, реинжиниринг.
- Зона покрытия аспектно-ориентированной разработки ПО включает приложения, компоненты, базы данных;
- Взаимодействие и интеграция с прочими парадигмами осуществляется посредством фреймворков, генераторов, языков программирования и языков описания архитектуры (ADL).

Основы аспектно-ориентированного подхода

Аспектно-ориентированное программирование позволяет выделить сквозную функциональность в отдельные декларации – аспекты. Можно определить функциональность для строго заданных точек выполнения программы JoinPoints (вызов метода, инициация класса, доступ к полю класса и т.д.). В языках, поддерживающих АОП, чаще используются назначения для множества точек - Pointcut. Функциональность в точках определяет программный код, который принято вежливо называть Advice (AspectJ). Таким образом, в пространстве аспектов описывается сквозная функциональность для определенного множества компонентов системы. В самих компонентах представлена лишь бизнес-логика, которую они призваны реализовать. В ходе компиляции программы компоненты связываются с аспектами (Weave).

Что бы лучше понять базисы АОП давайте, вернемся к примеру с выделением аспекта мониторинга производительности (См. Picture 2). Нам требуется снять показания таймера на входе и на выходе всех методов классов Model, Document, Record, Dispatcher. Итак, мы имеем аспект Logging. Нам потребуется завести к нему Pointcut с перечислением всех требуемых функций. В большинстве языков программирования, поддерживающих АОП для охвата сразу всех методов класса можно использовать специальную маску. Теперь можно описать для этого Pointcut. Создаем Advice на входе в методы из списка Pointcut (Before) и на выходе из них (After). Advice для событий Before, After, Around – наиболее популярны в языках, поддерживающих АОП, но порой доступны и другие события.

Итак, по результатам данного примера можно отметить для себя по поводу базисных деклараций АОО следующие:

Aspect – определение, некоторого набора сквозной функциональности, нацеленной на конкретную задачу;

Pointcut – код применимости аспекта. Отвечает на вопросы где и когда может быть применена функциональность данного аспекта (См. Рис. 3)

Advice – код функциональности объекта. Непосредственно код функциональности для заданных событий. Другими словами, это то, что будет выполнено для объектов, указанных в Pointcut.

Все еще сложно вникнуть в этот подход? Полагаю, все встанет на свои места, когда мы привнесем немного предметности. Давайте начнем с самого простого примера <http://www.phpclasses.org/browse/package/2633.html>.

Я некогда написал эту маленькую библиотеку специально, что бы проиллюстрировать как преимущества подхода АОО, так и его доступность. Кроме того, для того, чтобы воспользоваться данной библиотекой вам не потребуются глубоких знаний PHP и специального программного обеспечения. Вам будет достаточно включить библиотеку `aop.lib.php` в ваших скриптах под управлением PHP 4 (или выше) в его стандартной комплектации.

Мы можем определить некоторый аспект для сквозной функциональности (скажем, ведение журналов транзакций), посредством инициирования класса Aspect.

```
$aspect1 = new Aspect();
```

Далее мы можем создать Pointcut и сообщаем, какие методы он затрагивает.

```
$pc1 = $aspect1->pointcut("call Sample::Sample or call Sample::Sample2");
```

Осталось лишь сообщить программный код для входной и выходной точек методов текущего среза.

```
$pc1->_before("print 'Aspect1 preprocessor<br />';");  
$pc1->_after("print 'Aspect1 postprocessor<br />';");
```

Аналогичным образом мы можем описать дополнительный аспект, например:

```
$aspect2 = new Aspect();  
$pc2 = $aspect2->pointcut("call Sample::Sample2");  
$pc2->_before("print 'Aspect2 preprocessor<br />';");  
$pc2->_after("print 'Aspect2 postprocessor<br />';");
```

Для того, что бы задействовать один или несколько аспектов достаточно воспользоваться функцией `Aspect::apply()`

```
Aspect::apply($aspect1);  
Aspect::apply($aspect2);
```

Как вам возможно известно, в PHP до 5й версии достаточно проблематично обслужить события методов и классов. Если для событий глобального характера, таких как ошибки PHP, можно написать собственный обработчик, то для обработки, скажем, событий на входе и выходе методов придется «вручную» расставить «оповещатели». В нашем случае потребуется расставлять специальные функции `Advice::_before()`; и `Advice::_after()`;

```
class Sample {
```

```
function Sample() {
    Advice::_before();
    print 'Class initialization<br />';
    Advice::_after();
    return $this;
}
function Sample2() {
    Advice::_before();
    print 'Business logic of Sample2<br />';
    Advice::_after();
    return true;
}
}
```

Как видно из примера, до кода бизнес логики метода и после него установлены «оповещатели» этих событий.

Когда процессор PHP минует такой «оповещатель» он проверяет, нет ли активных аспектов. В случае наличия такового, PHP проверяет указана ли текущая функция в диапазоне Pointcut. Если и это условие верно, вызывается назначенная нами функция для данного события (например для Advice::_before()). Видите - как я и обещал, все достаточно просто. Но дает ли этот подход реальную пользу?

Давайте представим, что мы расставили во всех методах классов наших скриптов «оповещатели» и подключили библиотеку aop.lib.php. И вот однажды нам потребовалось получить подробный отчет о распределении нагрузки по выполняемым функциям нашего проекта. Мы создаем аспект и назначаем ему Pointcut, охватывающий все функции проекта.

```
$Monitoring = new Aspect();
$pc3 = $Monitoring->pointcut("call *::*");
```

Как показано в примере, мы можем воспользоваться маской *::*. Далее в Advice мы можем воспользоваться традиционной функцией вычисления точного времени в миллисекундах

```
function getMicrotime() {
    list($usec, $sec) = explode(" ", microtime());
    return ((float)$usec + (float)$sec);
}
```

И с помощью нее замерять время на входе в каждую функцию проекта и сверять его с показаниями на выходе из этой функции, а время выполнения операций бизнес логики в теле функции помещать в глобальную переменную отчета. Осталось лишь вывести на экран отчет в конце программы. Пример использования аспекта мониторинга производительности представлен в скрипте sample_trace.php в архиве дистрибутивного пакета.

Что бы у вас не сложилось мнение, что АОП помогает решать лишь какие-то частные задачи, давайте рассмотрим еще пример.

Как известно, PHP лояльно относится к типам переменных. С одной стороны, это не может не радовать нас, так нет необходимости постоянно заботиться о соответствии заданным типам и тратить время на их декларацию. С другой стороны – это причина множества возможных ошибок. При большом количестве функций в проекте, едва ли возможно удержать в памяти, заведенный нами же синтаксис для них. Но достаточно лишь поменять местами аргументы функции и ее поведение может стать непредсказуемым. Может ли здесь нам помочь АОП? А почему бы и нет?!

Давайте вспомним диаграмму, приведенную на Рис. 2. Как мы видим, классы Document и Record содержат однотипные методы add, update, delete, copy, get. Хорошо организованная программная архитектура подразумевает однотипный синтаксис для этих методов: add(\$id, \$data), update(\$id, \$data), delete(\$id), copy(\$id1,\$id2), get(\$id). АОП нам может помочь организовать и как программную архитектуру, так и самих себя. Мы можем завести аспект валидации входных параметров и определить диапазон Pointcut для методов классов Document и Record. Функция события входа для методов add, update, delete, copy, get может проверять тип первого аргумента.

Если он не является целочисленным (integer), то можно смело сообщать об ошибке. Можно также завести второй Pointcut для методов add и update. В данном случае будет проверяться тип второго аргумента. Он, очевидно, должен соответствовать типу массив (array).

Таким же образом, мы можем выносить за пределы бизнес логики проекта протоколирование транзакций, мы можем в любой момент определить диапазон функций требующих дополнительных проверок на предмет безопасности и т.д.

Что особенно интересно, посредством АОП мы можем назначить специфичный вывод сообщения о системной ошибке для определенного набора функций. Допустим, ряд наших функций участвует в формировании кода разметки WML (WAP), WSDL (SOAP), RSS/ATOM или SVG. Разумеется, в данном случае недопустимо выводить на экран HTML-разметку с сообщением об ошибке. «Оповещатель» в обработчике ошибок PHP заставит систему отобразить сообщение либо в требуемой разметке XML, либо оповестить нас без отображения сообщения на экране (например, по Email).

Любой, кому приходилось участвовать в разработке тиражируемых программных продуктов, знает насколько это не просто решить проблему обновления версий продукта. Конечно, все мы знаем о наличие специального программного обеспечения для контроля версий, например о CVS (Concurrent Versions System, http://en.wikipedia.org/wiki/Concurrent_Versions_System). Однако проблем в том, что каждый новый продукт на базе нашего тиражируемого продукта требует некоторой кастомизации и часто достаточно сложно выяснить, не затронет ли обновление области адаптированные под конкретный проект.

Наверняка кто-нибудь из вас встречался с проблемой, когда после очередного обновления версии базового продукта, приходилось восстанавливать весь проект из резервных копий. А представьте себе ситуацию, когда «пропажа» кастомизации в отдельных интерфейсах проекта выясняется спустя продолжительное время после обновления версии базового продукта! Вы скажите «А причем здесь АОП?!». Дело в том, что АОП как раз может помочь в решении данной проблемы. Мы ведь можем перенести весь код кастомизации проекта как сквозную функциональность за пределы основной бизнес логики. Достаточно определить аспект наших интересов, указать область его применимости (Pointcut) и разработать соответствующий код функциональности. Имеет смысл взглянуть на то, как это может работать.

Вернемся к моему излюбленному примеру с ПО для управления содержанием сайта. Подобный продукт наверняка будет содержать функцию для отображения списков записей (recordsets) Record::getList() и подготовки кода для отображения данного списка View::applyList(). Record::getList() получает в аргументах идентификатор recordset и параметры выборки в нем. Возвращает эта функция массив с данными по результатам выборки. Функция View::applyList() принимает на входе этот массив и формирует код оформления для него, например HTML код таблицы. Допустим, в нашем продукте каталог товаров представлен в виде подобных списков записей. Это универсальное решение для тиражируемого продукта, но для конкретного проекта, базированного на нем требуется отображать дополнительную колонку в списках.

Например, в базовом продукте принято отображать таблицу с полями артикул, наименование товара. А требуется к ним добавить поле «Рейтинг покупателей». Для этого мы всего лишь пишем Advice для функции Record::getList(), по которому на ее выходе в возвращаемый массив будет введена дополнительная колонка. Если наша функция View::applyList() не способна автоматически адаптироваться под изменения во входящем массиве, то придется написать Advice и для нее. Допустим, спустя время заказчик потребовал от нас выделить все строки в списках, обозначающие товары, которые отсутствуют на складе. Мы дописываем Advice для View::applyList(), где сверяем значение атрибута записей «Наличие на складе» и соответствующим образом оформляем их. Обратите внимание, что мы можем завести отдельную папку Plugins для деклараций аспектов и скриптов их функциональности.

Таким образом, вся кастомизация для любого из проектов будет сосредоточена в одно заданной папке. В дальнейшем у нас уже не будет проблем с обновлением версий. Мы сможем смело обновлять любые системные скрипты, за исключением тех, что представлены в папке Plugins.

Аспектно-Ориентированная Разработка ПО на PHP

В настоящий момент имеется ряд инициативных проектов, авторы которых, представили различные способы реализации АОП и PHP. В проекте аоPHP (<http://www.aophp.net>) представлен препроцессор PHP, написанный на Java 1.5. Мы можем писать привычный PHP-код, но должны будем сообщать препроцессору о нашем желании приобщения к АОП. Для этого вместо конструкции `<?PHP .. ?>` мы будем использовать `<?АОП ?>`. Сквозную функциональность мы сможем разместить в отдельных скриптах.

```
before(): execr(add($x,$y)) | execr(sub($x,$y)){
    echo "<font color=red>Im About To Add/Sub $x & $y</font><br>";
}
```

Эти скрипты при необходимости могут быть задействованы путем указания при декларации кода АОП

```
<?aophp filename="aotest.aophp,aotest2.aophp" debug="off"
// PHP code
?>
```

В проекте Seasar.PHP (<http://www.seasar.org/en/php5/index.html>) применен иной путь. Здесь для структурирования деклараций аспектов используется XML, а компоновку производит сам PHP, после чего выполняет результирующий код посредством функции eval().

В проекте MFAOP (www.mfaop.com) используется принцип немного похожий, на тот, что я демонстрировал выше в примерах. Автор проекта рекомендует первоначально назначить некоторый Pointcut и в дальнейшем уже его применять в различных аспектах.

```
$pointCut = new PointCut();
$pointCut->addJoinPoint('Example', 'Foo');
$pointCut->addJoinPoint('Example', 'Bar');
$test1 = new Aspect($pointCut, before, 'echo "Before $MethodName";');
$test2 = new Aspect($pointCut, after, 'echo "After $MethodName";');
```

В отличие от библиотеки `aop.lib.php` в данном решении вам нет необходимости расставлять «оповещатели» «вручную» для каждой функции. Но придется установить на сервере дополнительное расширение PHP PECL Classkit.

На мой взгляд, наиболее элегантное решение получилось у авторов проекта `PHRASpect` (www.phpaspect.org). Это стало возможным благодаря эффективному использованию новых возможностей PHP5, в частности возможности создания абстрактных классов. `PHRASpect` вводит специальную конструкцию в язык PHP, которая наглядно представляет декларируемый аспект.

```
aspect TraceOrder{
    pointcut logAddItem:exec(public Order::addItem(2));
    pointcut logTotalAmount:call(Order->addItem(2));
    after logAddItem{
        printf("%d %s added to the cart\n", $quantity, $reference);
    }
    after logTotalAmount{
        printf("Total amount of the cart : %.2f €\n", $thisJoinPoint->getObject
())->getAmount());
    }
}
```

Как видно в примере область заданного аспекта четко определена. Задание `Pointcut` и `Advise` столь лаконично, но емко, что складывается впечатление, будто это «родной» синтаксис PHP. Данный проект предлагает обслуживание событий `Join point` 7 (!) типов: вызов метода (`call`), выполнение метода (`exec`), инициализация класса (`new`), запись в атрибут (`set`), чтение атрибута (`get`), деструкция класса (`unset`) и захват блока (`catch`). Возможно задание `Advise` трех типов `before`, `after`, `around`. Проект позволяет использовать неожиданно гибкие маски для задания областей наблюдения в `Pointcut`. Так, к примеру, есть возможность задания области для всех классов с заданным префиксом в имени.

```
new(* (*));
exec(* Order::addItem(2));
call(DataObject+>update(0));
```

Для установки `PHRASpect` вам потребуется PHP версии не ниже 5.0.0 и установленные библиотеки `PEAR Console_Getopt`, `Console_Progressbar`, `PHP_Beautifier`.

Данный проект был с успехом представлен в прошлом году на PHP конференции (<http://afup.org/pages/forumphp/>) во Франции (на родине авторов). И судя по всему, активно развивается и ныне. Вполне возможно, что `Zend Inc.` обратит на него внимание и учтет этот опыт в следующих версиях PHP.

Заключение

Безусловно, AOSD вовсе не панацея. АОП не избавит нас от ошибок в программном обеспечении, да и программы не станут писаться сами собой. Едва ли каждый программист получит по нобелевской премии, только за то, что использует АОП. Более того, нельзя сказать, что этот подход в программировании будет доминирующим в будущем. Вспомните, за 2 десятка лет объектно-ориентированное программирование обрело популярность, но по-прежнему большое число программистов ограничиваются процедурным кодом.

С другой стороны, преимущества АОП перед ООП очевидны, так же как преимущества ООП перед процедурным программированием. Можно сказать, что программисты, использующие АОП на шаг впереди.

Аспектно-ориентированная веб-разработка и PHP

Их код лучше в плане удобочитаемости, соответственно в нем меньше ошибок, он лучше подходит для развития. Таким образом, инженеры AOSD способны реализовать более масштабные, более надежные, более универсальные проекты, нежели программисты не использующие АОП. И заметьте, АОП не требует кардинальной переквалификации программистов. АОП не меняет логику программирования, до неузнаваемости как это было при переходе от процедурного программирования к ООП. Можно сказать, что АОП лишь расширяет ее. Проектируя новую программную архитектуру, вы лишь выносите из объектов все то, что «мозолит вам глаза» и определяете все это в подходящее место.

Представьте, что вы упорно терпите в своем доме статуэтку, которая абсолютно не вписывается в дизайн интерьера, но дорога вам как память. Но однажды вы делаете скрытую от праздного взгляда нишу в стене, где все неудобные предметы находят свое истинное место. После чего остается лишь только восклицать: «Ну, правильно! Именно здесь оно и должно быть!».

Парадигма	Начало эры программирования	Машинный/мнемонический код	Модульное/структурированное пр.	Объект-ориентированное программирование	Аспект-ориентированное программирование
Основ-полагающий язык	Ada	Autocoder	Fortran	Simula 67	AspectJ
	с 1945	с 1952	с 1954	с 1967	с 2001

Рис. 4. Эволюция программных парадигм.

АОП также не требует длительного периода привыкания, как, например TDD (http://en.wikipedia.org/wiki/Test_driven_development). По началу можно выносить в аспекты простую сквозную функциональность, такую как протоколирование. А со временем все более и более расширять декомпозицию программной архитектуры, собирая в областях аспектов различные сферы применимости приложения.

Возможно, кого-то смущает тот факт, что официально PHP не поддерживает АОП в настоящее время. Однако, как раз в этой статье я показал вам на примерах, что без особого труда вы сами можете обеспечить поддержку основных принципов АОП. Не так важна реализация идеи AOSD, как ее суть. Какой бы вы образ ни выбрали, но если вам удалось более качественная декомпозиция в программной архитектуре, это в любом случае улучшит качество ваших программ.

Сегодня АОП действительно поддерживается по большей части в расширениях к популярным языкам программирования, а не в них самих. Однако ведущие игроки на рынке не остаются в стороне, и АОП постепенно пробивает себе дорогу в популярные программные платформы (<http://www.internetnews.com/dev-news/print.php/3106021>). Не стоит ожидать технологической революции под знаменами AOSD. Но эволюция неизбежна. Догонять ее или следовать во главе ее – выбор за нами.

Создание дружественных URL

Автор: Quentin Zervaas

Перевод: Андрей Олищук

В наше время предъявляются высокие требования к URL, которые должны быть легко читаемы и человеку и поисковику

Одной из главных причин использования серверных языков программирования, таких как PHP, является возможность динамического управления контентом. Часто случается, что лишь один скрипт управляет выводом всего контента в зависимости от переданных в URL параметров. Эта статья посвящена технике и методам отображения таких параметров в понятной и «дружелюбной» форме и их обработке в PHP-скриптах.

Вступление

Чтобы вы поняли, о чем идет речь, рассмотрим пример. Сайт хранит свои статьи в таблице СУБД articles. Чтобы показать ту или иную статью, мы ссылаемся на нее по ID:

http://phpriot.com/article.php?article_id=1234

Это самый простой, но не самый лучший путь. Во-первых, если посетитель сайта просмотрел много статей, то адресная строка его браузера будет переполнена различными ID и он не сможет вернуться к нужной статье не прибегая к помощи закладок или перехода на индексную страницу, так как не запомнит соответствия ID и статей. Во-вторых, что еще важнее, при таких адресах теряется важная для поисковиков информация, и они хуже индексируют сайт.

К примеру, сайт phpriot.com использует такой «человеко-понятный» URL для статей:

<http://phpriot.com/d/articles/php/application-design/multi-step-wizards/index.html>

В этой статье будет рассказано, как программно прочитать такой URL и спроецировать полученные параметры на данные в вашей базе. В PHP существует несколько методов для решения подобных задач. Мы рассмотрим их все и обсудим «за» и «против» каждого метода.

Apache и mod_rewrite

Первым методом, который мы рассмотрим, будет модуль mod_rewrite для веб-сервера Apache. Этот модуль работает по принципу разбора адреса URL и выделения параметров, которые и передаются в скрипт, по подготовленному вами шаблону.

Допустим, в корневой директории веб-сервера есть скрипт news.php (т.е. доступ к нему осуществляется как к <http://example.com/news.php>). Этот скрипт отвечает за вывод одной новости, ID которой передается в URL. Другими словами, если вам нужно получить новость с ID 63, то в обычной ситуации вы обратились к скрипту так: http://example.com/news.php?news_id=63.

Создание дружественных URL

Но вместо этого, мы сделаем URL лучше читаемым и нам понадобится обратиться к тому же скрипту как: `http://example.com/news/63.html` (в качестве примера).

В любом случае, для того, чтобы сделать такое обращение работоспособным, необходимо внести одно небольшое «правило» в файл конфигурации сервера `httpd.conf` или в файл `.htaccess` в директории сайта. Текст, который необходимо добавить в файл, может быть следующим:

```
RewriteEngine on
RewriteRule ^/news/([0-9]+)\.html /news.php?news_id=$1
```

Используя регулярное выражение, мы сопоставляем все запросы к веб-серверу, которые начинаются со строки «news/», а далее содержат любое количество цифр и строку «.html». Символы, которые заключены в квадратные скобки будут присвоены переменной `$1` (символы в следующих квадратных скобках, были бы присвоены `$2`, `$3` и т.д, но в нашем примере только одни квадратные скобки). Значение переменной `$1` будет подставлено в конечный URL. Таким образом, произойдет «незаметная» замена URL и скрипт получит обычную `$_GET`-переменную `$news_id`. Вот пример:

```
<?php
    $news_id = $_GET['news_id'];
?>
```

Дополнительные параметры URL

Иногда могут возникать ситуации, когда в скрипт нужно передать дополнительные параметры. Применительно к нашему примеру с `news.php`, это может быть дополнительный параметр “print”, который указывает на то, что страницу нужно отобразить в версии для печати (технически это делается с помощью CSS, но CSS – это отдельная тема).

В обычном случае, вы бы обратились к `news.php` следующим образом:

```
http://example.com/news.php?news_id=63&print=1
```

Используя наш вариант с `mod_rewrite`, мы могли бы обратиться к скрипту как: `http://example.com/news/63.html?print=1`

Однако, наше регулярное выражение никак не выделит параметр `print`, поэтому в него нужно внести некоторые коррективы, используя внутреннюю переменную Apache `{QUERY_STRING}`. Мы просто добавим ее к `news_id` с амперсандом.

```
RewriteEngine on
RewriteRule ^/news/([0-9]+)\.html /news.php?news_id=$1&{%QUERY_STRING}
```

Теперь оба параметра доступны через `$_GET`.

```
<?php
    $news_id = $_GET['news_id'];
    $printVersion = isset($_GET['print']);
?>
```

Все это работает с помощью модуля `mod_rewrite`. Это очень мощный и многосторонний модуль, поэтому при его изучении вы легко можете столкнуться с проблемами. Поначалу, к примеру, бывает сложно заставить его работать с вашими шаблонами. Существует несколько отладочных опций, с помощью которых вы можете попробовать решить свою проблему.

Использование директивы Apache – ForceType

Альтернативой `mod_rewrite` является директива того же веб-сервера Apache – `ForceType`. Что же она делает? Она позволяет выполняться PHP-скриптам, у которых нет расширения `.php`. Обычно, веб-сервер настраивается так, чтобы обрабатывать файлы с расширением `php`, как PHP-приложения, а файлы с другими расширениями (к примеру `html`) не обрабатывать интерпретатором.

Возвращаясь к нашему примеру с `mod_rewrite`, вместо файла `news.php` в корневой директории будет лежать файл `news` (без расширения). Он будет доступен как `http://example.com/news`

Для этого необходимо внести изменения в `httpd.conf` или `.htaccess` и записать в них следующее:

```
<Files news>
    ForceType application/x-httpd-php
</Files>
```

Теперь, для доступа к нашей статье мы сможем вызывать:

`http://www.example.com/news/63.html`

В этом примере, файл будет доступен напрямую и нам останется обработать только строку `/63.html`. Она сохранена в серверной переменной `PATH_INFO`.

```
<?php
echo $_SERVER['PATH_INFO'];
// выведет '/63.html'
?>
```

Теперь нам нужно использовать регулярные выражения для извлечения числа 63 из строки. Перед тем как перейти к регулярным выражениям, скажу, что параметров после имени файла `news` может быть несколько (к примеру, `news/sport/63.htm`), в таких случаях, строку предварительно необходимо обработать функцией `explode()` с разделителем `/`, что разобьет ее на массив значений. Однако, в нашем примере параметр один, и мы коснемся регулярных выражений.

Приведу пример регулярного выражения (совместимого с `preg_match()`), которое ищет строку, предваряемую слешем и завершаемую сочетанием `.html`. Затем все найденные совпадения помещаются в массив, из которого мы и будем брать `article_id`.

```
<?php
$path = $_SERVER['PATH_INFO'];
preg_match('!^/(\d+)\.html$!', $path, $matches);

// $matches[0] сохранит всю строку, в то время как $matches[1]
// сохранит совпадение по первым скобкам. Нужно проследить,
// чтобы эта строка была числом, поэтому приведем тип к int:
$news_id = (int) $matches[1];
?>
```

Обычно мы используем слеш как ограничитель выражения, но здесь для этого лучше использовать другие символы (в этом случае – `«!»`). Так же добавим, что здесь мы ищем одну 1 (+) или несколько цифр (`\d`). Здесь мы должны экранировать точку, так как точка в синтаксисе регулярных выражений обычно означает «любой символ», а нам точка нужна именно как символ точки.

Создание дружественных URL

В целом, это все что касается данной темы. Теперь вы можете использовать `$news_id` так как и положено в скрипте. Если путь не будет соответствовать регулярному выражению, то `$news_id` примет значение 0, после того как мы приведем адресную строку к типу `integer`. Такой статьи может не быть, поэтому ошибку будет нужно обработать.

Использование своего обработчика 404 ошибки

Вероятно, это самый запутанный способ достижения нужного нам результата, не смотря на его простоту и мощность. Используя преимущества обработки 404 ошибки в Apache, вам будет достаточно создать один контролирующий скрипт, который будет решать, как обрабатывать тот или иной запрос. Конечно же, этот метод будет работать только в том случае, когда запрашиваемого файла действительно не существует в файловой системе. К примеру, если на вашем сайте хранятся картинки, вы сможете получить к ним доступ напрямую и тогда обработчик ошибки 404 не будет задействован.

Стоит добавить, что вы можете использовать преимущества функции `header()` и послать вместо «404 File not found», заголовок «200 ОК». В таком случае, конечный пользователь даже и не догадается, что запрашиваемый им файл реально не существует.

Пример использования

Основываясь на этой идее, вам не придется беспокоиться за то, разрабатываете вы систему для работы с новостями, или что-то большое с разными типами контента.

К примеру, посмотрите на URL: `http://www.phpriot.com/articles/php/index.html`. Вместо того, чтобы нам создавать физический путь на сервере, мы используем обработчик 404 ошибки для работы с путем `/articles/php/index.html`.

Реализация обработчика 404 ошибки

Здесь мы рассмотрим пример реализации все той же системы новостей, но заодно мы коснемся и других сфер, которые можно обрабатывать таким образом, включая странички сообщений об ошибках.

Первое, что нужно сделать, это установить свой обработчик 404-ой ошибки. Это нужно прописать в `.htaccess` или конфигурационном файле `httpd.conf`:

```
ErrorDocument 404 /handler.php
```

Данная директива означает, что все запросы к несуществующим на сервере файлам будут перенаправлены в файл `handler.php` в корне `document_root`. Соответственно, в этом скрипте можно будет обрабатывать все поступающие запросы. Для получения исходного адреса нужно использовать серверную переменную `REDIRECT_URL`.

```
<?php
$request = $_SERVER['REDIRECT_URL'];

// разделим запрос на несколько частей
$parts = explode('/', $request);

// установим флаг, нашли ли мы контент content
$found = false;

array_shift($parts);

// теперь определим тип контента
switch ($parts[0]) {
    case 'news':
        // используем простой регексп
        preg_match('!^(\\d+)\\.html$', $parts[1], $matches);
```

Создание дружественных URL

```
$news_id = (int) $matches[1];

$output = getNewsArticle($news_id);
// эта функция на самом деле не существует
// но мы предположим, что она возвращает контент статьи
// или NULL если статьи не существует

if ($output !== null)
    $found = true;

break;

case 'articles':
    // здесь мы реализуем обработчик для вывода на экран

    break;

default:
}

if ($found) {
    // вывести заголовок, что контент найден, или отправить 404 ошибку
    header('HTTP/1.1: 200 OK');
    echo $output;
}
else {
    header('HTTP/1.0 404 Not Found');
    echo 'File not found';
}
?>
```

Конечно, этот скрипт еще достаточно «сырой», однако он наглядно демонстрирует принцип работы метода.

Выводы

В этой статье мы рассмотрели несколько способов манипуляции вашими адресами URL, с целью приведения их к более «человекочитаемой» форме. Это будет полезно как для лучшей индексации сайта поисковиками, так и для более прозрачной работы пользователей. В реальности, я полагаю, такие поисковики как Google справятся и с GET параметрами, но «человекочитаемые» URL будут более прозрачными и для них.

Полезные ссылки

- Apache 2.0 URL rewriting guide: <http://httpd.apache.org/docs/2.0/misc/rewriteguide.html>
- Apache ForceType directive : <http://httpd.apache.org/docs/2.0/mod/core.html#forcetype>
- Apache ErrorDocument directive <http://httpd.apache.org/docs/2.0/mod/core.html#errordocument>

Оригинал статьи: <http://www.phpriot.com/d/articles/php/application-design/search-engine-urls/index.html>

Лучшая практика

Автор: Ligaya Turmelle

Перевод: Крушняков Кирилл

Мало знать синтаксис языка, нужно уметь программировать

"Лучшая практика" – что это значит? Одно из определений, найденных Google, гласит: "Процессы и действия, которые на практике доказали свою наибольшую эффективность". Что это даёт лично вам? Эти "приемчики" работают - используйте их! И хотя, я буду рассуждать исключительно в аспекте использования PHP, эти методы могут быть применимы для любого другого языка. Эти простые указания помогут сделать ваши скрипты более читаемыми и более понятными (для вас и для вашего коллеги), сделать более безошибочными и сохранить ваше время. Полагаю, что лучший вопрос для вас: "Почему вы еще НЕ используете их?" Примечание: Это не всеохватывающий набор приемов, но это то, что поможет вам начать.

Думайте, прежде чем писать код

У вас есть голова - используйте ее прежде, чем прикасаться к клавиатуре. Потратив некоторое время, для принятия небольшого количества решений и обдумывания общего дизайна кода, вы можете сохранить для себя часы, дни и даже недели в случае, если ваша "отличная" задумка окажется не столь отличной, на самом-то деле. То, что должен иметь под рукой каждый программист - это обычная ручка и листок бумаги. Писать код - это просто. Если у вас есть качественный план - всё что вы делаете, это просто его запись в нужном синтаксисе.

Процедурно или объектно ориентированный код

Одно из решений, которые вы должны принять перед тем, как начать кодировать - это методология кода, которой вы будете следовать. Будете ли вы создавать процедуры с библиотекой функций, объектно ориентированный код с классами, или объединение и того и другого подхода? PHP - язык, который не принуждает исключительно к одному из этих методов, что, по моему убеждению, является одной из его сильнейших сторон. Каждый из методов имеет свои преимущества и недостатки, так что выбирайте тот метод, который наилучшим образом подойдет вашему скрипту.

Вы слышали о блок-схемах или псевдо-коде? Это простые неязыковые специфические методы проработки логики скриптов. Вашей ручкой вы записываете/рисуете логику вашего кода на бумаге и "на пальцах" исполняете его, проверяя его теоретическую работоспособность. Вот, на что вы должны потратить основное ваше время. Вы можете расписать вашу логику от пошагового уровня переходов по if / else до уровня вызова конкретных функций или классов. Не пожалейте на это вашего времени. Чем больше проблем вы обнаружите на этой стадии, тем меньше ошибок и непредвиденных ситуаций придется разрешать позже. Скомкать, выкинуть бумажку да перерисовать заново значительно проще, чем переписывать тысячу строк кода.

Разбивайте

Теперь, когда основная логика Вашего кода обозначена, начинайте искать элементы, которые выполняются многократно или просто по логике своей кажутся объединяемыми. Возможно такие участки кода стоит выделять в небольшие самостоятельные функции или классы. Разбивание вашего кода на многократно применимые части сохраняет время на печати кода и поиске ошибок. Печати, ибо не придется набивать одно и то же многократно, поиске ошибок, ибо функция будучи безошибочной при первом вызове останется таковой и при следующем (Разумеется, если она будет безошибочна на всем диапазоне значений аргументов - примечание переводчика)

Комментируйте

Зачем комментировать? Ну, на самом деле это нужно не вам, а тому парню, что придет после вас (которым можете быть вы сами, через пол-годика) Не стоит комментировать каждую строчку, но используйте достаточное количество комментариев для того, чтобы "парень за вами" мог без труда проследить за вашим ходом мысли, воплощенной бизнес-логикой и "приколами" вашего кода.

Блочные комментарии

Блочные комментарии (Использующие комментарии в стиле языка C /* */) используются в основном с функциями, классами или более сложной логикой. Таким образом, вы можете снабдить комментарий значительным количеством информации, или, если вы будете соблюдать формат комментариев PHPDoc, вы так же сможете сгенерировать HTML документацию для ваших классов или функций. Вот пример:

```
<?php
/**
 * Заводит временный файл или директорию
 *
 * При исполнении деструктора все
 * заведенные временные файлы или директории уничтожаются
 *
 * @param string $file имя файла или директории
 *
 * @return void
 *
 * @access public
 */
?>
```

Строчные комментарии

Строчные комментарии (//) позволяют вам прояснить ход вашей логики более "точечно". Эти комментарии "спасут вам жизнь", когда пол-года спустя, вы решите добавить новую функциональность, или отследить баг, который упорно продолжает проявлять себя. Я предпочитаю помечать такие вещи, как валидация, обработка ошибок, взаимодействия с СУБД, или эпизодические логические исключения.

```
<?php
class mySession
{
public function __construct()
{
// Ку-ку! Я конструктор, который слишком ленив, чтобы что-то делать!
}
}
?>
```

Стиль кода

К счастью или нет, но дни кода-спагетти (сленговое обозначение плохо спроектированного или структурированного кода, сложного для чтения и понимания, словно клубок спагетти) канули в лету, так что то, КАК вы пишете практически так же важно, как то, ЧТО вы пишете.

Я не хочу разводить флейм по поводу того, чьи "стандарты" кодинга лучше, а чьи хуже, например, использование пробелов или табуляции. Неважно, какой именно стиль кодирования вы используете, до тех пор, пока вы продолжаете следовать ему на протяжении создания всего вашего скрипта. Примером этого может быть стандарт кодирования PEAR.

Отступы

Один из моментов, которые я бы хотела обязательно упомянуть это отступы. Отступы используются для визуальной группировки логических блоков, для облегчения чтения и понимания, а так же помогает созрывать соответствие скобок. (Речь идет о скобках "{" и "}" функций и операторов if,while... - прим. перев.) Используйте, двух, трех, четырех-пробельные отступы, табы, или что еще вы предпочитаете, главное придерживайтесь этого постоянно.

```
<?php
if (isset($_SERVER["REMOTE_ADDR"])) {
    $octet = substr($_SERVER["REMOTE_ADDR"], 0, strpos($_SERVER["REMOTE_ADDR"],
    "."));
    switch ($octet) {
    case 127:
        $network_class = "Зарезервированное значение 'петли'";
        break;
    case ($octet <= 126):
        $network_class = "Класс А";
        break;
    case ($octet <= 191):
        $network_class = "Класс В";
        break;
    case ($octet <= 223):
        $network_class = "Класс С";
        break;
    case ($octet <= 239):
        $network_class = "Класс D";
        break;
    case ($octet <= 255):
        $network_class = "Класс Е";
        break;
    default:
        $network_class = "Неизвестный класс";
        break;
    }
    } else {
        $network_class = "Неизвестный класс";
    }
    echo "Ваш IP адрес принадлежит: $network_class";
    ?>
```

Именованние переменных

В PHP переменные должны начинаться с буквы или подчеркивания, сопровождающихся любым количеством букв, цифр или подчеркиваний.

```
<?php
$x // Совершенно не понятно, что это такое
```

Лучшая практика

```
$num // Уже лучше, понятно, что это номер,  
// но номер чего?  
$cc_num // Еще лучше, понятно, что эта переменная хранит номер  
// кредитной карты. (А может товарный номер сыра чеддар (cheddar cheese)?)  
$cred_card_num // Оптимальное имя переменной  
$chedChezStockNum // Оптимальное имя переменной  
?>
```

Следует, так же, отметить, что имена переменных PHP чувствительны к регистру, так что \$Var и \$var - это две различные переменные. Будьте, так же, уверены, что, задавая имя переменной, вы обеспечиваете им достаточное количество информации о том, для чего предназначена эта переменная, или откуда она образуется. Нет конкретных пределов для длины переменной, единственный предел - ваша готовность их набирать.

Сообщения об ошибках

Вы не можете исправить ошибку, если Вы не знаете, в чем она заключается. Хотите верить, хотите нет, но быть оповещенным об ошибках это замечательно, иначе у вас будет проблема, без сведений об этой проблеме. PHP обеспечивает несколько способов формирования об ошибках, которые могут появиться в ваших скриптах. Есть несколько директив, с которыми вы можете поиграть в файле `php.ini`, которые отвечают за оповещение об ошибках.

Установки в `php.ini` влияют на ВСЕ PHP скрипты, запущенные на сервере, не только ваши. Все директивы документированы, дабы помочь вам в их использовании (смотрите документацию PHP). Там есть, разумеется и директива `error_reporting`, а есть так же и `display_error`, `log_error` и `error_log`. `display_error` просто указывает, выводить или нет сообщения об ошибках на экран и обычно установлен в `false` на "боевых" серверах. Вместо этого, сообщения об ошибках могут быть вынесены в специальный файл лога, установкой директив `log_errors` и `error_log`.

Скрипт

Если у вас нет доступа к файлу `php.ini` вашего сервера - не спешите паниковать. У вас всё еще остается возможность некоторого контроля надо сообщениями об ошибках. Простейший способ убедиться, что вы можете видеть все сообщения об ошибках в вашем конкретном скрипте - не подавлять их с помощью "@".

Если вам ну очень нужно использовать "@" - добавляйте их ПОСЛЕ отлаживания и тестирования вашего скрипта. После этого, установите `error_reporting()` вашего скрипта в `E_ALL` (или `E_ALL | E_STRICT` для PHP5). Мы можем использовать следующий пример в нашем коде для перенаправления всех ошибок в `logfile` вне директории доступной для веб-сервера, когда мы начинаем производственную эксплуатацию:

```
<?php  
error_reporting(E_ALL | E_STRICT);  
ini_set("display_errors", False);  
ini_set("log_errors", True);  
ini_set("error_log", "../outside/of/web/root/error_log.txt");  
?>
```

Обработка ошибок

Закон Мёрфи гласит: "Если что-то может пойти не так, так оно и случится". Сервера отключаются от электричества, время соединений превышает допущенный предел, пути ведут в никуда, а жесткие диски сыпятся.

Лучшая практика

Это то, что наш скрипт не в состоянии контролировать. Работа программиста заключается в том, чтобы снабжать скрипт способами обработки этих или других исключительных ситуаций. И не будем забывать, что обработка ошибок может помочь нам в отладке проблемы, позволяя узнать о её существовании и её сути.

Мои примеры написаны в процедурном стиле PHP4 - но те же самые концепции применимы и для объектно ориентированного подхода. К примеру, существует класс PEAR_Error, а PHP5 обладает полной поддержкой механизма "исключений" (exceptions) и структуры "try/catch". Используйте их, если считаете нужным.

Пример 1

Обработка ошибок, при взаимодействии с СУБД - момент, часто упускаемый программистами новичками. Большинство новичков используют запросы соединения и произведения выборки (select), но когда получение выборки генерирует ошибку они не представляют, что произошло не так. Обработка ошибок во всем коде позволит вам ТОЧНО определить, где изначально возникла проблема. Простой пример, с использованием СУБД MySQL:

```
<?php
$link = mysql_connect($host, $usr, $pass);
// обрабатываем ошибку соединения
if(!$link)
{
// обрабатываем ошибку - выводим на экран, в лог, делаем перенаправление, что
угодно
// если мы в состоянии отладки - это оповестит меня о возникшей проблеме
// в противном случае спрячет диагностику от злоумышленника
if($debug)
{
echo 'Ошибка: соединение с СУБД'.mysql_error();
}
// аккуратно выходим
exit;
}
?>
```

Пример 2

Что насчет тех, кто хочет работать с файлами - читать, писать, за/скачивать?

```
<?php
$basepath = '/home/usr/www/downloads/';
switch($_POST['type'])
{
case 'audio':
$type = 'audio/';
break;
case 'video':
$type = 'vid/';
break;
// catch anything
default:
$type = 'lyric/';
}
$path = $basepath.$type.$user.'/';
// обработка ошибки пути
if(!is_dir($path))
{
// обрабатываем ошибку - выводим на экран, в лог, делаем перенаправление, что
угодно
// возвращаемся назад в форму, или аккуратно выходим
}
```

```
}  
?>
```

Безопасность

Проблемы безопасности в PHP происходят из-за некачественного PHP-кода, а не из-за того, что PHP плох, как язык. Он просто даёт вам широкие возможности, пригодные и для того, чтобы вы могли навредить себе тоже. ("It just gives you enough rope to hang yourself.") Есть множество прекрасных пособий (как online, так и книг) которые излагают методику написания безопасного PHP кода. Мы лишь кратко затронем несколько аспектов безопасности.

Для более глубокого изучения данной темы, пожалуйста скачайте и прочтите "Открытое руководство по безопасности PHP от Консорциума по безопасности PHP" ("PHP Security Consortium's free PHP Security Guide")

Аутентификационные параметры доступа к СУБД НИКОГДА, повторяю, НИКОГДА не должны быть помещаемы в пределах каталога веб сервера. Хотите знать почему? Сходите сюда: <http://www.google.com/search?q=inurl%3Adb.inc> (впервые опубликовано в блоге Chris Shiflett). Если Ваши данные доступа помещены в корне вебсервера, потенциально кто угодно в состоянии как следует повеселиться с вашей базой данных. Если это произойдет, то любые персональные данные и пароли могут быть украдены.

Лично я предпочитаю хранить всю информацию доступа (БД, файлы паролей, конфигурационные файлы) в каталоге доступа, или произвольном, вне корневого каталога web. И тогда я просто включаю их в выполняемый скрипт.

Примечание переводчика: На самом деле, приведенный пример дыры имеет место быть не потому, что файл с логином и паролем лежит внутри каталога веб-сервера, а потому, что он имеет не ассоциированное с PHP расширение ".inc", в результате чего Apache (или другой веб-сервер) просто отдает его содержимое по запросу веб-браузера, как обычный, неисполняемый файл. Выход в том, чтобы отказаться от хранения скриптов и их данных в любых файлах, отличных от расширения .php, или отделять их в каталоги, для которых веб-серверу запрещен доступ, например, для Apache директивами

```
<Directory /var/www/mysite/mydata>  
  Deny from All  
</Directory>
```

В том случае, когда вы храните пароли в виде части PHP кода, в файле, например, db.inc.php - пользователь (тот же Google) не сможет получить его содержимое, он получит лишь результат его выполнения. В свою очередь, напротив, доступ интерпретатора PHP к файлам вне каталога вебсервера лучше ограничить директивами файла

```
<Directory>  
  /var/www/mysite/>php_admin_value open_basedir /var/www/mysite/  
</Directory>
```

дабы если уж злоумышленник и взломает ваш скрипт, он не смог получить доступа к остальным файлам системы.

Проверяйте вводимые данные

НИКОГДА, НИКОГДА, НИКОГДА не доверяйте данным, введенным пользователем. Мне нужно повторить? Никогда не доверяйте данным, введенным пользователем. Не буду перечислять все причины почему нет, но скажу, что принятые "плохие" данные от пользователя могут доставить огромные неприятности вашему скрипту.

Лучшая практика

Так что же нам делать, если мы не доверяем вводимым данным? Проверять всё, что нам подадут на вход. PHP располагает рядом функций, способном вам в этом помочь. К сожалению, у меня нет времени и достаточного места, чтобы рассмотреть их все, так что я только укажу несколько опций в примере. Обязательно рассмотрите функции из примера и все с ними связанные.

```
<?php
if ((isset($_POST["username"]) &&
(trim($_POST["username"]) != "") && (ctype_alnum(trim($_POST["username"]))) &&
(strlen(trim($_POST["username"])) >= 4) && (strlen(trim($_POST["username"])) <=
10))
?)
```

Ммм.. Всё еще не упомянуты функции регулярных выражений, функция `checkdate()` и возможно еще несколько, но я думаю вы уловили общую мысль.

Необходимо обезопасить вывод

Зачем обрабатывать вывод, когда вы так старательно обрабатываете ввод? Ну, неважно насколько мы умны (или думаем, что умны), кто-то всегда умнее. Так что, мы должны убедиться, что ничего вредоносного, даже если оно "проберется" сквозь нашу фильтрацию ввода, не сможет ничего натворить при выводе страницы (что-то, навряде "дефейса" вашей страницы или вредоносного же JavaScript). Итак, как обезопасить вывод? Ну, `addslashes()`, `urlencode()` и `htmlspecialchars()` - хорошее начало.

```
<?php
$query = sprintf("INSERT INTO myTable (comments) VALUES ('%s')",
mysql_real_escape_string($comments));
echo "Следующий комментарий будет добавлен в базу данных:<br />";
echo htmlspecialchars($content);
?>
```

Инициализация переменных

Просто производите инициализацию. Если вы не сделаете этого, кто-то другой сделает это, что приведет к печальным последствиям и огромным дырам в безопасности. Это включает переменные, которые могут прийти в ваш скрипт извне, скажем POST или GET переменные, к примеру. Если вы используете в вашем скрипте временную переменную - убедитесь в том, что вы присваиваете ей начальное значение. Используйте триарный оператор для POST, GET и REQUEST переменных:

```
<?php
$user = (isset($_POST['user']))?$_POST['user']:'';
?>
```

Изобретая колесо

В PHP есть несколько мест, куда вам стоит обратиться, чтобы проверить, не существует ли уже того, в чем вы нуждаетесь. Они могут спасти ваше время и ваши усилия, пускай вам возможно придется приспособить то, что вы получите оттуда под свои нужды.

Классы абстракций баз данных, расширения PHP, шаблоны, системы управления контентом, движки-фреймворки - лишь некоторые из этих уже изобретенных колес. Зачем изобретать колесо, если в этом нет необходимости?

PEAR (PHP Extension and Application Repository) и **PECL** (PHP Extension Community Library) - официальные репозитории PHP. PEAR для ваших классов for PHP, которые вы используете в ваших скриптах, для добавления функциональности. Пара часто используемых классов PEAR - PEAR::DB и PEAR::MailMime. PECL служит для PHP расширений, написаны на C, которые вы встраиваете в ваш интерпретатор PHP для увеличения набора функций используемых вашим кодом. Расширения PECL иногда попадают в набор библиотек ядра PHP. Пара популярных расширений PECL - APC (Alternative PHP Cache) и pdflib.

Sourceforge.net и **freshmeat.net** - два репозитория для проектов с открытым исходным кодом. Каждый включает тысячи проектов, всевозможных категорий и различных степеней готовности. То, что вы хотите или в чем нуждаетесь возможно уже доступно.

Google ваш друг

Когда сомневаетесь - используйте поисковик, например Google. Многие проекты располагают собственными сайтами или доступны на многочисленных сайтах разработчиков PHP и могут быть найдены, только если вы используете поисковики. Руководства, куски кода, классы кустарного изготовления, фреймворки, вот лишь несколько из вещей, которые можно успешно искать в вашем любимом поисковике.

Дополнительно

Ок, после того, как основы были пройдены, я быстро пробежусь по нескольким "продвинутым" приемам кодирования. Каждая из этих тем требует более внимательного изучения, в отдельных статьях, так что, пожалуйста просмотрите их, когда у вас будет свободное время.

CVS

Система контроля версий CVS, или его альтернатива, Subversion, это системы контроля версий и распределения труда с открытым кодом. Они используются с целью разделения, сохранения и восстановления информации о версиях для людей пишущих и редактирующих код. Даже если вы единственный программист - это хорошая идея, использовать их. Мы все делали изменения в нашем коде (иногда весьма значительные) только для того, чтобы убедиться, что он перестает работать после них, или мешает работать чему-то другому, или является лишь старой, неудачной идеей. Системы контроля версия позволяют вам отменить эти изменения, прежде, чем они станут необратимыми.

Отладка ошибок

Отладка ошибок это процесс использующийся для записи и отслеживания ошибок кода, после их обнаружения. Процесс включает запись рапорта об ошибке, обзора и записи о необходимых исправлениях и патчи, требуемые и/или примененные. Одной из лучших систем отслеживания ошибок является Bugzilla.

Модульное тестирование

Модульное тестирование - это метод тестирования отдельного модуля исходного кода. Входные данные общих и экстремальных значений подаются на обработку модуля и сверяются результаты. Некоторые считают, что модульные тесты должны писаться перед собственно кодом. Модульное тестирование часто ассоциируется с экстремальным программированием, хотя и существовало задолго до него.

Заключение

Итак, вот они, несколько простых рекомендаций, могущих помочь вам писать более правильный код. Используйте их и почувствуйте разницу.

Об авторе

Lig - богиня на полной ставке, PHP разработчик по совместительству и заядлый путешественник. Она счастливая жена, мать прекрасной дочери, работает в небольшой компании, в качестве местного "компьютерного лунатика". Её блог доступен по адресу <http://blog.khankennels.com>

О переводчике

Переводчик - жалкий неудачник, дилетант широкого профиля. Холост, сын несчастных родителей. До собственного блога он еще не дорос ;-)

Комментарий «О переводчике» предоставлен самим переводчиком – прим. Редакции.

Оригинал статьи: <http://codewalkers.com/tutorials/94/1.html>

Persistent Objects в ezComponents

Авторы: компания eZ Systems

Перевод: Роман Толкачев

Реализация постоянных объектов с помощью ezComponents

Здесь и далее под постоянными объектами будут подразумеваться объекты, которые сохраняют свои состояния между запусками скриптов.

Persistent Object предоставляет возможности для работы с постоянными объектами на основе базы данных. Persistent Object использует компонент Database для абстракции от базы данных, но не использует какие-либо генераторы PHP кода или свои структуры для хранения информации о типах объектов (как это делает Propel).

Компонент Persistent Object ориентирован, прежде всего, на быструю разработку постоянных объектов по принципу, который используется для создания обычных объектов.

EzComponents (http://ez.no/products/ez_components) – продукт компании eZ Systems (<http://ez.no>). Он представляет собой готовую платформу для разработки PHP-приложений и состоит из отдельных компонент-блоков. Важным преимуществом компонентов является их распространение по BSD-совместимой лицензии. Полный список доступен по адресу: http://ez.no/products/ez_components/ez_components_list

Обзор класса

ezcPersistentSession

ezcPersistentSession - основной класс, реализующий API для управления "постоянством" объектов. Загрузка, сохранение, удаление - всё совершается в этом классе.

Основы использования

В этой части рассматривается типичное использование компонента Persistent Object. Будет задействован один тип объекта и база данных MySQL в качестве хранилища.

Класс постоянного объекта

Итак, начнём с создания класса, представляющего человека. Это простой класс, реализующий малое количество функций и свойств.

```
<?php
class Person
{
private id = null;
public name = null;
public age = null;
```

Persistent Objects в ezComponents

```
public function getState()
{
    $result = array();
    $result['id'] = $this->id;
    $result['name'] = $this->name;
    $result['age'] = $this->age;
    return $result;
}

public function setState( array $properties )
{
    foreach( $state as $key => $value )
    {
        $this->{$key} = $value;
    }
}
?>
```

Здесь, свойство `id` является идентификатором объекта (отвечает за уникальность). По-умолчанию должно быть `null`, это не обязательно для других свойств. Свойство `id` используется так же для генерации уникальных записей.

Для упрощения, свойства `name` (имя) и `age` (возраст) общедоступны, но это не обязательно. В своих классах вы можете по разному реализовывать доступ к ним.

Все постоянные объекты должны иметь методы `getState()` и `setState()`. Они нужны для получения информации, которую следует о них сохранить и для установки свойств при загрузке. Метод `getState()` обязан всегда возвращать полную информацию о состоянии объекта, тогда как `setState()` должен устанавливать информацию только об одном объекте за раз.

Привязывание

Теперь нам нужно привязать объект к таблице

```
CREATE TABLE persons
(
    id integer unsigned not null auto_increment,
    full_name varchar(255),
    age integer,
    PRIMARY KEY (id)
) TYPE=InnoDB;
```

В данном случае мы реализуем связь "поле таблицы - свойство объекта". Использование

InnoDB не обязательно, но мы рекомендуем использовать именно их, поскольку таблицы такого типа поддерживают транзакции. Поле `id` установлено как `auto_increment`. Это требуется для нашего генератора уникальных ID, при использовании других генераторов, это можно упустить.

Для того, чтобы Persistent Object мог сохранять объекты класса `Person` в таблицу БД, нам нужно указать ему как связаны столбцы таблицы со свойствами класса. Мы используем `ezcPersistentCodeManager` для подгрузки таких данных тогда, когда они нам понадобятся.

```
<?php
$def = new ezcPersistentObjectDefinition();
$def->table = "persons";
$def->class = "Person";
```

Persistent Objects в ezComponents

```
$def->idProperty = new ezcPersistentObjectIdProperty;
$def->idProperty->columnName = 'id';
$def->idProperty->propertyName = 'id';
$def->idProperty->generator = new ezcPersistentGeneratorDefinition(
    'ezcPersistentSequenceGenerator' );

$def->properties['name'] = new ezcPersistentObjectProperty;
$def->properties['name']->columnName = 'full_name';
$def->properties['name']->propertyName = 'name';
$def->properties['name']->propertyType =
    ezcPersistentObjectProperty::PHP_TYPE_STRING;

$def->properties['age'] = new ezcPersistentObjectProperty;
$def->properties['age']->columnName = 'age';
$def->properties['age']->propertyName = 'age';
$def->properties['age']->propertyType =
    ezcPersistentObjectProperty::PHP_TYPE_INT;

return $def;
?>
```

В первом блоке мы создаём связывающий объект, задаём имя класса и таблицы. Во втором блоке определяется связь между id полем таблицы и свойством id класса, а так же указывается метод, которым будет генерироваться идентификатор для нового объекта. В данном случае мы используем `ezcPersistentSequenceGenerator`, который просто возвращает новый id по `auto_increment`.

В двух следующих блоках реализуется привязка имени и возраста. Их имена одинаковы как в объекте, так и в таблице.

Каждый хранимое свойство класса должно быть внесено в свойство `properties` объекта `ezcPersistentGeneratorDefinition`, которое является ассоциативным массивом, ключём в котором выступает имя свойства, а содержимым, объект `ezcPersistentObjectProperty`.

Если вы посмотрите интерфейс `ezcPersistentObjectDefinition`, то найдёте свойство `columns`, которое напоминает `properties`, но является её обратным вариантом, т.е. ключём массива выступает имя поля в БД, а не свойство класса.

В конце мы возвращаем объект. Без этого связи не будут работать.

Для работы связей с `ezcPersistentCodeManager`, они должны быть помещены в одну директорию и названы именем класса в нижнем регистре. В нашем случае это `person.php`.

Объект сессии

Объект сессии предназначен для сохранения и загрузки постоянных объектов. Он может быть создан так:

```
<?php
$session = new ezcPersistentSession(
    ezcDbInstance::get(),
    new ezcPersistentCodeManager( "path/to/definitions" )
);
?>
```

Конструктор объекта принимает два параметра: экземпляр объекта БД и `CodeManager`, служащий для получения информации о связях. Мы используем `ezcPersistentCodeManager`, который загружает связи из файлов, находящихся в одной директории, при его создании вы должны указать директорию, в которой находится `person.php`.

Persistent Objects в ezComponents

Возможно, вам не захочется создавать новый объект сессии несколько раз, решить эту проблему можно обёртывание класса сессии в Singleton.

Создание и обновление объекта

Создадим объект Person и сохраним его:

```
<?php
$object = new Person();
$object->name = "Guybrush Threepwood";
$object->age = 31;

$session->save( $object );
?>
```

Теперь объект сессии создал сохранил для нас этот объект в БД. Идентификатор объекта сохранился в свойстве id объекта. Потому как Guybrush - наш первый объект, то его id будет равен 1.

Конечно же, Guybrush Threepwood, оказался моложе, чем мы предположили. Для исправления его свойств, просто измените свойства объекта и передайте сессии таким образом:

```
<?php
$object->age = 25;
$session->update( $object );
?>
```

Заметим, что здесь мы используем метод update, для того чтобы явно указать, что мы хотим именно обновить существующий, а не создать новый объект.

Поиск объекта

У нас есть несколько способов получить объект. Простейший из них, получить его по идентификатору:

```
<?php
$object = $session->load( 'Person', 1 );
?>
```

Данный код возвратит объект Guybrush Threepwood, созданный выше.

Если вы сохранили множество объектов и хотите извлечь их списком, то вы можете воспользоваться методом find. Этот метод требует объект запроса, как один из параметров. Сам объект запроса можно получить из сессии.

```
<?php
$q = $session->createFindQuery( 'Person' );
$q->where( $q->expr->gt( 'age', 15 ) )
->orderBy( 'full_name' )
->limit( 10 );
$objects = $session->find( $q, 'Person' );
?>
```

Данный код возвратит не более 10 человек старше 15, с сортировкой по их имени

Метод find() сразу извлечёт все объекты из result set и передаст вам уже готовый массив. Это не очень удачное решение, если вы извлекаете большое количество объектов и вам требуется производительность. Для этого существует fetchIterator()

```
<?php
$q = $session->createFindQuery( 'Person' );
```

Persistent Objects в ezComponents

```
$q->where( $q->expr->gt( 'age', 15 ) )
->orderBy( 'name' )
->limit( 10 );
$objects = $session->findIterator( $q, 'Person' );

    foreach( $objects as $object )
{
// ...
}
?>
```

Этот код возвращает тот же результат что и `find()`. Только данные извлекаются из БД только тогда, когда они востребованы.

Удаление объектов

Простейший способ удалить объект - воспользоваться методом `delete()` сессии:

```
<?php
$object = $session->load( 'Person', 1 );
$session->delete( $object );
?>
```

Таким образом вы можете удалить уже загруженный объект. Если вам нужно удалить множество объектов по условию, то воспользуйтесь методом `deleteFromQuery()`:

```
<?php
$q = $session->createDeleteQuery( 'Person' );
$q->where( $q->expr->gt( 'age', 15 ) );
$session->deleteFromQuery( $q );
?>
```

Данный код удалит всех людей, старше 15 лет.

Генерация идентификаторов

Все постоянные объекты имеют идентификатор. Алгоритм генерации идентификаторов показывает каким должен быть новый ID. В этой части мы будет рассказано о уже существующем генераторе.

ezcPersistentSequenceGenerator

Данный генератор использует `PDO::lastInsertId()` для получения нового ID,

Рекомендуется использовать `auto_increment` id для поддерживаемых БД (MySQL и SQLite). Для других БД нужно использовать счётчики. Например, для PostgreSQL:

```
CREATE TABLE persons
(
id integer unsigned not null,
full_name varchar(255),
age integer,
PRIMARY KEY (id)
);
CREATE SEQUENCE person_seq START 1;
```

Если в вашей БД используются счётчики, то вам это нужно указать:

```
<?php
$def->idProperty->generator = new ezcPersistentGeneratorDefinition(
'ezcPersistentSequenceGenerator',
```

```
array( 'sequence' => 'person_sequence' )
);
?>
```

Определение типов объектов

Объект сессии должен получать информацию о связывании БД и объекта.

ezcPersistentCodeManager

Это единственный существующий менеджер определений. Он просто загружает требуемые определения из файлов, не проводя при этом никаких проверок.

Модификация менеджера определений

Самое простое - создать свой загрузчик определений. Для этого достаточно создать потомка `ezcPersistentDefinitionManager` и переопределить метод `fetchDefinition`:

```
<?php
class ezcPersistentCodeManager extends ezcPersistentDefinitionManager
{
public function fetchDefinition( $class )
{
}
}
?>
```

Метод `fetchDefinition()` должен создать определение структуры или выкинуть исключение.

Оригинал статьи: [http://ez.no/doc/components/view/\(file\)/1.0/introduction_PersistentObject.html](http://ez.no/doc/components/view/(file)/1.0/introduction_PersistentObject.html)

Кеширование средствами HTTP

Автор: Joe Gregorio

Перевод: Андрей Олищук

Используем HTTP-протокол для повышения производительности веб-приложений

Я не раз говорил, что при разработке сервисов в веб-окружении, необходимо основательно подходить к заголовкам HTTP-протокола, которые позволяют улучшить производительность приложений с помощью кеширования средствами HTTP-протокола. Чтобы получить преимущества, вам необходимо познакомиться поближе с кешированием на базе HTTP. В этой статье вы не найдете ни объяснений по настройке кеширования для вашего личного веб-сервера, ни описаний других видов кеширования. Если вам нужна такая информация, то рекомендуем заглянуть сюда: http://www.mnot.net/cache_docs/

Цели

Для начала, попробуем разобраться с моделью кеширования на основе HTTP. Главная цель такого кеширования – дать понять клиенту и серверу, что в данный момент необходимо работать с закешированной ранее копией. При этом, главной сложностью является контроль за тем, чтобы не подsunуть пользователю явно устаревший вариант документа.

Модель кеширования HTTP основывается на валидаторах, которые являются «метками». На базе них клиент может принимать решение – устарел документ или нет. Фундаментальное значение имеет тот факт, что подобные валидаторы позволяют запрашивать статус ресурса без пересылки дополнительных запросов: сервер отправляет тело ответа только в том случае, если последний раз запрашивался устаревший документ.

Валидаторы

Одним из валидаторов в HTTP является Etag. ETag похож на отпечаток пальцев для байтов – если хотя бы один байт в представлении меняется, то меняется весь ETag.

Использование валидаторов предполагает, что вы как минимум один раз сделали GET-запрос к ресурсу. Кеш сохраняет значение заголовка ETag (если таковой был представлен) и затем использует это значение в последующих запросах к тому же самому URI.

К примеру, если я отправляю запрос к `example.com`, то могу получить такой ответ:

```
HTTP/1.1 200 OK
Date: Fri, 30 Dec 2005 17:30:56 GMT
Server: Apache
ETag: "11c415a-8206-243aea40"
Accept-Ranges: bytes
Content-Length: 33286
Vary: Accept-Encoding, User-Agent
Cache-Control: max-age=7200
Expires: Fri, 30 Dec 2005 19:30:56 GMT
```

Кеширование средствами HTTP

```
Content-Type: image/png
```

```
-- binary data --
```

Когда я буду делать следующий запрос, я могу вставить валидатор. Заметьте, что значение ETag вставляется в заголовок If-None-Match.

```
GET / HTTP/1.1
Host: example.org
If-None-Match: "11c415a-8206-243aea40"
```

Если никаких изменений не было, то сервер вернет заголовок 304 Not Modified.

```
HTTP/1.1 304 Not Modified
Date: Fri, 30 Dec 2005 17:32:47 GMT
```

Если изменения были, то сервер вернет заголовок 200, тело документа и новый ETag.

```
HTTP/1.1 200 OK
Date: Fri, 30 Dec 2005 17:32:47 GMT
Server: Apache
ETag: "0192384-9023-1a929893"
Accept-Ranges: bytes
Content-Length: 33286
Vary: Accept-Encoding, User-Agent
Cache-Control: max-age=7200
Expires: Fri, 30 Dec 2005 19:30:56 GMT
Content-Type: image/png

-- binary data --
```

Контроль кеширования

Если валидаторы используются для проверки валидности документа, то заголовок Cache-Control используется для указаний – как долго документ должен сохраняться в кеше. Одной из главных директив Cache-Control является max-age. Данная директива указывает, сколько секунд должно пройти, перед тем, как возникнет необходимость проверить документ на валидность. Заметьте, что директива max-age может употребляться как в заголовках запроса, так и в заголовках ответа. Если кеш ответа является свежим, то мы можем сразу же отдать его пользователю, но если кеш устарел, то мы должны проверить его валидность, перед тем как вернуть клиенту.

Давайте по другому взглянем на приведенный выше пример. Стоит учесть, что значение Cache-Control установленное в 7200, означает, что документ будет закеширован на два часа.

```
HTTP/1.1 200 OK
Date: Fri, 30 Dec 2005 17:32:47 GMT
Server: Apache
ETag: "0192384-9023-1a929893"
Accept-Ranges: bytes
Content-Length: 33286
Vary: Accept-Encoding, User-Agent
Cache-Control: max-age=7200
Expires: Fri, 30 Dec 2005 19:30:56 GMT
Content-Type: text/xml
```

Существует много директив, которые могут быть помещены в заголовок Cache-Control.

Кеширование средствами HTTP

Директива	Описание
no-cache	Игнорирование даже действительного кеша
no-store	Не сохранять ответ в кеше
max-age=delta-seconds	Клиент принимает кешированную копию без валидации на протяжении n секунд
max-stale=delta-seconds	Клиент принимает кешированную копию, которая не старше n секунд
min-fresh=delta-seconds	Принимает только кешированный ответ, который не старше n секунд от настоящего времени
no-transform	Тело не должно быть трансформировано
only-if-cached	Принимает ответ, только в том случае, если существует кеш данного документа

Таблица 1. Директивы запроса

Директива	Описание
public	Документ может быть закеширован любым клиентом
private	Возможность кеширования ограничена
no-cache	Кеш должен в любом случае проходить валидацию
no-store	Не кешировать данный ответ
no-transform	Нельзя трансформировать тело
must-revalidate	Если кеш устарел, он должен быть перепроверен перед выдачей любому клиенту
max-age=delta-seconds	Клиент должен принимать кеш документа на протяжении n секунд
s-maxage=delta-seconds	То же самое, что и max-age, но верно только для разделяемого кеша
proxy-revalidate	Если используется прокси, то обязательно должна проходить ревалидация

Таблица 2. Директивы ответа

Давайте посмотрим некоторые примеры Cache-Control:

```
Cache-Control: private, max-age=3600
```

Если такой заголовок отправляется сервером, то он дает указание на ограниченное кеширование на один час.

```
Cache-Control: public, must-revalidate, max-age=7200
```

Это означает, что ответ может быть закеширован любым клиентом на два часа. После этого, кеш должен пройти ревалидацию.

```
Cache-Control: must-revalidate, max-age=0
```

Эта директива вынуждает клиента провести ревалидацию. Так как max-age=0, кеш будет помечен как заведомо устаревший. Хорошие примеры использования этих директив вы можете найти здесь: <http://www.mnot.net/blog/2005/11/26/caching>

```
Cache-Control: no-cache
```

Кеширование средствами HTTP

Эта директива очень близка к «must-revalidate, max-age=0», за исключением того, что клиент может использовать свои ограничения по максимальному сроку валидности и получать уже устаревшие документы. Некоторую эффективность может давать одновременное использование директив со стороны клиента и сервера.

Сейчас мы рассмотрели использование заголовков на стороне сервера, но некоторые директивы можно использовать и на стороне клиента в запросе.

```
Cache-Control: no-cache
```

Данная директива вынуждает клиента загружать новую версию документа и обновлять кеш с сервера.

```
Cache-Control: min-fresh=200
```

Клиент сообщает серверу, что ему нужен ответ, который не является старше, чем 200 секунд.

Vary

Сервер может иметь систему управления контентом и выдавать различные данные при обращении к одному URI. Для таких ситуаций используется заголовок Vary. Данный заголовок информирует, какие другие заголовки могут повлечь изменение контента.

К примеру, если для одного и того же URI меняется Content-Type, то на один и тот же URI может выдаваться различный тип контента. В таких случаях, на стороне сервера можно добавить Vary для этого URI.

```
Date: Mon, 23 Jan 2006 15:37:34 GMT
Server: Apache
Accept-Ranges: bytes
Vary: Accept-Encoding, User-Agent
Content-Encoding: gzip
Cache-Control: max-age=7200
Expires: Mon, 23 Jan 2006 17:37:34 GMT
Content-Length: 5073
Content-Type: text/html; charset=utf-8
```

В этом примере сервер определяет, что кеширование включается на два часа, но эти ответы могут меняться в части заголовков: Accept-Encoding и User-Agent

Connection

Когда сервер подтверждает, что кеш еще актуален (к примеру, используя If-None-Match), то сервер возвращает код статуса 304 Not Modified. Так что же, по выдаче кода 304 ничего не происходит? Не совсем. На самом деле сервер посылает обновленные заголовки, которые могут быть обновлены в кеше. Сервер может так же послать заголовок Connection, который укажет, какие из заголовков обновлять не следует.

Некоторые заголовки по умолчанию исключены из списка заголовков, предназначенных для кеширования. В этот список входят следующие заголовки: Connection, Keep-Alive, Proxy-Authenticate, Proxy-Authorization, TE, Trailers, Transfer-Encoding и Upgrade.

```
HTTP/1.1 304 Not Modified
Content-Length: 647
Server: Apache
Connection: close
Date: Mon, 23 Jan 2006 16:10:52 GMT
```

Кеширование средствами HTTP

```
Content-Type: text/html; charset=iso-8859-1
```

```
...
```

В приведенном выше примере нет некешируемых заголовков, а так же, нет указаний на отмену обновления кеша заголовков в директиве Connection. Поэтому, заголовок Date будет обновлен.

Если бы все было так легко

Одной из проблем в той сфере, является ситуация, когда нам приходится работать с сервером, поддерживающим протокол HTTP 1.0 и набором заголовков в кеше, созданном по стандартам HTTP 1.1.

Старая модель контроля кеширования, пришедшая из HTTP 1.0 основывается в основном на факторе времени. Валидатор Last-Modified представляет собой ничто иное, как дату последней модификации документа. Чтобы указать на изменения, кеш использует заголовки: Date:, Expires:, Last-Modified: и If-Modified-Since.

Если вы разрабатываете клиент, то должны использовать все валидаторы, так как заранее неизвестно, какого стандарта будет придерживаться сервер – HTTP 1.0 или HTTP 1.1. С другой стороны, спецификация HTTP 1.1 была опубликована семь лет назад и с тех пор многое изменилось, поэтому поддержка двух стандартов напоминает одновременное использование ремня и подтяжек.

Прим. переводчика: напомним, что в PHP использование HTTP-заголовков возможно, к примеру, при работе с сокетами.

Виртуальная файловая система. Внедряем в веб-сайт

Автор: Роман Толкачев

Перенос идей из обычных программ в веб

Много интересного можно подчеркнуть из «больших» программ, причём, многое можно применить и к веб-сайту. В данной статье я постараюсь рассказать о том, как можно применить на сайтах (скорее это применимо к CMS) виртуальную файловую систему, а главное какие преимущества это нам даст.

Хотелось бы сразу огорчить – вы не найдёте здесь ни работающего PHP кода, ни даже нормального SQL, потому как я не ставлю перед собой цели показать реализацию, а лишь наталкиваю на новые идеи.

Начну с того, откуда я для себя придумал название «Виртуальная Файловая Система» (далее ВФС). Задача данной системы ставилась, как структурирование объектов сайта по типу файловой системы, потому изначально, она была названа «Виртуальная Объектная Система». После долгих раздумий, и усовершенствований в системе, слово «Объектная» было заменено на «Файловая». Почему? Ну, файл это хранилище информации, а чем объект не хранилище? К тому же было введено многое из того, что чисто к объектной системе сайта отношение не имело.

Файловая система ОС, будь то Windows или *nix, представляет собой совокупность нескольких типов объектов, «распианных» по папкам (к слову, «папка» - отдельный тип). Что реально даёт нам такое устройство файловой системы, так это относительные пути. Предположим, существует приложение 'doit.exe', обрабатывающее файлы i*.part, и таких наборов файлов несколько. В файловой системы мы раскидываем такие наборы по разным директориям, а затем из этих директорий выполняем doit.exe, не заморачиваясь на перенастройку, а уж тем более, боже упаси, на исправление исходного кода. Теперь представим, что файловой системы нет... Во-первых, нужно будет каждый набор хранить с разными именами (т.е. один i*.part, а другой k*.part), а во-вторых переконфигурировать doit.exe. На лицо увеличение производительности.

Чем же ограничивает нас файловая система, и почему нельзя использовать её для структурирования информации об объектах сайта? Проблема в том, что мы не в силах создать новый тип объекта в том же NTFS, а решения, связанные, например, с сериализацией объектов в файл, попросту не эффективны в плане производительности. Так же стоит отметить то, что мы не сможем в этом случае быстро получить все объекты определённого типа, так как они разбросаны повсюду, а в веб-строительстве это самая частая операция.

Теперь попробуем представить как это должно выглядеть в реализации PHP + MySQL. Директории, по сути, это узлы в дереве, поэтому их можно представить через NetedSets. Можно, конечно, и объекты впихнуть в это дерево, но стоит учитывать уменьшение скорости при увеличении объёмов дерева, поэтому все объекты выносим в отдельные таблицы (по таблице на тип). Для примера:

```
CREATE TABLE `dirs` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `title` VARCHAR( 30 ) NOT NULL ,
```

Виртуальная файловая система. Внедряем в веб-сайт

```
`full_path` VARCHAR( 255 ) NOT NULL ,  
-- Тут ещё NestedSets данные  
PRIMARY KEY ( `id` ) ,  
INDEX ( `title` ) , INDEX ( `full_path` )  
);
```

Не скупитесь на индексы, они тут очень нужны! Чтобы было с чем работать – введём в систему объект article:

```
CREATE TABLE `obj_article` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `dir_id` INT NOT NULL ,  
  `name` VARCHAR( 30 ) NOT NULL ,  
  `title` VARCHAR( 255 ) NOT NULL ,  
  `text` TEXT NOT NULL ,  
  PRIMARY KEY ( `id` ) ,  
  INDEX ( `dir_id` ) ,  
  INDEX ( `name` )  
);
```

Важным тут является `dir_id`, привязывающее объект к Виртуальной Директории. `name` – имя объекта (файла). Востальном, ничего интересного. Теперь можем создать любое количество объектов `article` в разных директориях. Остаётся реализовать API для работы со всем этим. Думаю, это не составит труда. Оно должно уметь:

- Выбирать объекты из директории
- Задавать директорию как текущую (все запросы после установки будут относительно текущей директории)
- Соответственно, к первому пункту, легко определять ID директории по пути.

Теперь, алгоритм таков: пишем приложение, которому каким-либо образом передаётся «текущая директория», и она устанавливается таковой через API. Далее алгоритм всегда один и тот же – просто указываем не абсолютный, а относительный путь к объекту (или вовсе извлекаем все объекты из директории).

Существенным недостатком данной системы является то, что объект может находиться только в одной папке, тогда как очень часто требуется одной категории сопоставить несколько новостей. Как вариант – использование системного объекта «символьная ссылка» по типу Unix систем. В нашем случае, будет проще, представить всё как ссылки. Т.е. не держать отдельно в каждом типе объектов поле `'dir_id'`, а собирать это всё в одну таблицу. К примеру, она может выглядеть так:

```
CREATE TABLE `obj2dir` (  
  `id` INT NOT NULL AUTO_INCREMENT ,  
  `dir_id` INT NOT NULL ,  
  `obj_id` INT NOT NULL ,  
  `type` VARCHAR( 10 ) NOT NULL ,  
  PRIMARY KEY ( `id` ) ,  
  INDEX ( `dir_id` , `obj_id` , `type` )  
);
```

В данном случае «тип объекта» представим текстом, что наносит существенный удар по производительности, поэтому описания типов рекомендуется хранить в отдельной таблице, а типе в `obj2dir` иметь как INT.

Виртуальная файловая система. Внедряем в веб-сайт

Помимо того, что мы пытались достичь (один объект в нескольких директориях), мы ещё одну немаловажную вещь сделали. Теперь возможно получить все директории, в которых находится объект одним запросом:

```
SELECT d.full_path FROM dirs d, obj2dir o
WHERE o.obj_id=ид_объекта
      AND o.type=тип_объекта
      AND o.dir_id = d.id
```

Фактически, в случае со статьями, мы можем одинаково легко получить все статьи в категории и все категории, в которых находится статья.

Теперь всё готово. Осталось лишь добавить в API функцию, которая бы извлекала все объекты из текущей и всех вложенных в неё директорий, что с NestedSets не составит труда. На этом базовая часть готова. Далее я опишу лишь идеи, без какого-либо описания способов реализации, потому как засорять тему не эффективными алгоритмами не хотелось бы, а эффективные каждый находит для себя сам.

Директория? Нет, узел.

Главное понять, что термина «директории» как такового нет. Есть понятие «узел дерева». И всё что хотите – можете с этим узлом делать. Например, может быть «папка», содержимое которой формируется динамически по SQL запросу. Можно так же сделать ссылку на другой узел, или, даже на директорию сервера. Всё упирается в реализацию обработки этих узлов. Я уже писал, что эта статья лишь подтолкнёт вас на новые идеи – задачу реализовывать их, я оставляю вам.

Всегда готов к диалогу по данному вопросу, как на форуме phpinside.ru, так и по e-mail: romeo@rikt.ru.

Howto: Конвертация из разных систем счисления

Автор: Филипп Лебедев

Простой пример применения функций для конвертаций из одной системы счисления в другую

Во многих учебных заведениях на занятиях, связанных с информатикой, приходится изучать системы счислений. Суть таких задач заключается в том, чтобы преобразовать число из одной системы счисления в другую. Порой это становится долгим и мучительным процессом. При необходимости, процесс можно автоматизировать, написав небольшой скрипт. Пример работы готового скрипта в онлайн можно посмотреть здесь: <http://omeo.h15.ru/php/ss/> и скачать здесь: <http://omeo.h15.ru/php/ss/ss.rar> (обратите внимание, что это только демонстрационные скрипты, в которых не уделяется внимания безопасности!)

В языке PHP есть функции для преобразований систем счислений:

- `base_convert()`;
- `bindec()`;
- `decbin()`;
- `hexdec()`;
- `dechex()`;
- `octdec()`;
- `decoct()`;

Функции `base_convert()` передаётся три параметра: число для преобразования; основание его системы счисления; основание СС, в которую нужно преобразовать число. Именно эта функция используется в скрипте, ссылка на который дана выше, т.к данная функция работает со множеством систем счислений.

Я немного расскажу и об остальных функциях. **`bindec()`** преобразует двоичное число в десятичное (десятичная СС – это те цифры и числа, которые мы используем каждый день), а **`decbin()`** – это обратный процесс. **`hexdec()`** переводит число из шестнадцатеричной СС в десятичную. **`dechex()`** - обратная `hexdec()`. **`octdec()`** переводит из восьмеричной СС в десятичную, а **`decoct()`** – наоборот.

Рассмотрим простой пример формы для ввода изначальных данных:

```
<form name="form1" method="get" action="convert.php">
<div align="left">
<p>Число, которое переводим:
<input name="f" type="text" id="f">
<br>
Основание её системы счисления:
<input name="d" type="text" id="d">
<br>
В какую СС перевести:
<input name="t" type="text" id="t">
```

```
</p>
<p>
<input type="submit" name="Submit" value="Преобразовать">
</p>
</div>
</form>
```

Форма передаёт странице `convert.php` три переменные: `f` – число, которое должен преобразовать сценарий, `d` – основание его системы счисления и, наконец, `t` – Основание той системы счисления, в которую мы переводим число. После щелчка на кнопку «преобразовать» мы попадаем на страницу `convert.php`. Рассмотрим её код:

```
<?
$result=base_convert($_GET["f"], $_GET["d"], $_GET["t"]);
echo "Мы преобразовали число:" . $_GET["f"] . "из" . $_GET["d"] . " СС в" .
$_GET["t"] . " СС и получили результат: <h1><strong>$result</h1> <span
class=стиль1>(" . $_GET["t"] . ")</span></strong></p>";
?>
```

Переменная `$result` – это результат нашего перевода, то есть число `$f` в `$d`-ичной системе счисления, преобразованное в `$t`-ичную СС.