Июнь '04 № 4

PHP Inside

электронный журнал для веб-разработчиков

Переход на PHP 5

- Практическое применение ORM
- Написание пользовательских функций для управления сессиями
- Интервью с Мануэлем Лемосом, основателем phpclasses.org

phpclub

5 лет с вами!

Содержание

В фокусе
Что нового в PHP 5?.....2
Переход к PHP 5.....14
Идеи
Практическое применение ORM (Object-Relational mapping) в PHP.....33
Создание пользовательских функций управления сессиями.....43
Люди
Интервью с основателем phpclasses.org – Мануелем Лемосом.....53
Кто-то должен быть первым.....57
Команда этого выпуска.....63
Для заметок.....64

Обратная связь

Здравствуйте!

Рады приветствовать вас в четвертом выпуске журнала PHP Inside! Журнал продолжает развиваться, находить все новых и новых читателей, получать интересные отзывы.

Вот и мы сменили свои контактные адреса. Теперь для связи с редакцией пишите на nw@phpinside.ru или nw@phpinside.net. Как и прежде, присылайте нам свои отзывы и пожелания, предложения и материалы.

Сейчас в самом разгаре лето, пора отпусков, но мы постараемся не сходить с боевой вахты и выпускать в свет все новые и новые номера нашего журнала. Для этого нам нужна ваша помощь в подготовке статей и переводов англоязычных материалов. По вопросам участия в проекте пишите по указанным выше адресам электронной почты (там живу я, координатор проекта PHP Inside - Андрей Олищук [nw]).

Если вы являетесь администратором или владельцем посещаемого интернет-ресурса, посвященного PHP или смежным технологиям, и готовы публиковать информацию о наших новых выпусках, сообщите нам, и вы самыми первыми будете получать информацию о выходе новых номеров журнала, а также увидите ссылку на свой ресурс на страницах журнала или на нашем сайте.

Не забудьте новые адреса:

nw@phpinside.net

nw@phpinside.ru

В фокусе

Что нового в PHP 5?

Только время покажет, будет ли PHP 5 столь же успешен, как его предшественники (PHP 3 и PHP 4). Новые возможности и изменения призваны избавить PHP от его слабых мест и гарантировать, что он по-прежнему будет занимать лидирующее положение в вебе.

Автор:
Энди Гутманс
Перевод:
Вадим Крючков

Новая объектно-ориентированная модель

Когда Зив Сураски (Zeev Suraski) добавил объектно-ориентированный (OO) синтаксис в PHP 3, это можно было рассматривать как «синтаксический подсластитель для поддержки классов» («syntactic sugar for accessing collections»). Объектно-ориентированная модель получила поддержку наследования и позволяла классу (и объекту) объединять методы и свойства, но не более того. Когда Зив и Анди переписали движок для PHP 4, это был полностью новый движок, работающий много быстрее, намного стабильнее и с еще многими другими возможностями. Однако, изменения практически не затронули OO модель, первоначально введенную еще в PHP 3.

Хотя объектная модель имела серьезные ограничения, она широко использовалась, часто в очень больших приложениях, написанных на PHP. Это победное чествование парадигмы ООП, даже такой ограниченной в PHP 4, привело к тому, что изменения объектной модели стали центральными в новом релизе PHP.

Какие были ограничения в PHP 3 и 4? Самым большим ограничением (которое и приводило ко всем остальным ограничениям) был тот факт, что семантика экземпляра объекта была такой же, что и для родных типов. Как это фактически отражалось на разработчиках? Когда вы присваивали переменную (которая указывает на объект) другой переменной, то создавалась копия объекта. Мало того, что это влияло на производительность, но и это обычно приводило к ошибкам в приложении, потому что многие разработчики думали, что обе переменные будут указывать на тот же самый объект. А они указывали на разные копии того же самого объекта, поэтому, изменяя один объект, мы не меняли другой. Смотрите пример в листинге 1.

В PHP 4, код, приведенный в листинге 1, выведет «Andi». Причина кроется в том, что мы передаем объект `$person` в функцию `changeName()` по значению, а не по ссылке, таким образом, объект `$person` будет скопирован, и `changeName()` будет работать уже с копией объекта `$person`.

Такое поведение не является интуитивно понятным. Действительно, многие разработчики ожидали Java-подобного поведения. В Java, переменные фактически являются указателями на объект, и поэтому при дублировании будет скопирован указатель, а не сам объект.

Было два вида разработчиков: те, кто знал об этой проблеме, и те, кто не знал. Последние, обычно, не сталкивались с этой проблемой, потому что их код был написан так, что было безразлично, существует ли такая проблема или нет. Конечно, некоторые из этих разработчиков проводили бессонные ночи в «увлекательных» поисках «сверхъестественных» ошибок.

Первая группа также имела проблему, поскольку приходилось вручную определять передачу объекта по ссылке, запрещая движку копировать объекты, и код был испещрен многочисленными знаками '&'.

```
<?php
class Person {
        var $name;
        function getName() {
          return $this->name;
        }
        function setName($name) {
          $this->name = $name;
        }
        function Person($name) {
          $this->setName($name);
        }
}
function changeName($person, $name) {
        $person->setName($name);
}
$person = new Person("Andi");
changeName($person, "Stig");
print $person->getName();
?>
```

Листинг 1.

Старая объектная модель приводит не только к вышеупомянутым проблемам, но также вскрывает более фундаментальные проблемы, которые на существующей объектной модели не позволяли осуществлять другие возможности.

В PHP 5 объектная модель была полностью переписана для того, чтобы сразу работать с указателями на объект. Если вы явно не клонируете объект, используя ключевое слово `clone`, вы никогда не будете работать с копией объекта, думая, что работаете с самим объектом. В PHP 5 уже не нужно явно передавать объекты или присваивать их по ссылке, это делается автоматически.

Обратите внимание: явная передача и присваивание по ссылке также поддерживается, на тот случай, если вы хотите изменить содержимое переменной или объекта.

Новое в объектно-ориентированном подходе

Новые возможности объектной модели являются слишком многочисленными, чтобы дать детальное описание в этой главе. В главе, посвященной объектной модели, будут рассмотрены в деталях все новые возможности.

Ниже представлен обзор главных изменений:

- `public/private/protected` – модификаторы доступа для методов и свойств. Позволяют управлять доступом к методам и свойствам.

```
<?php
class MyClass {
        private $id = 18;
        public function getId() {
          return $this->id;
        }
}
?>
```

- Унифицированный конструктор `__construct()`.

  Конструктор, ранее совпадавший с названием класса, теперь необходимо объявлять как `__construct()`, что позволит легче перемещать классы в иерархиях.

```
<?php
class MyClass {
        function __construct() {
          print "Inside constructor";
        }
}
?>
```

- Поддержка деструктора для класса, определяемого как метод `__destructor()`.

  Позволяет определить функцию деструктора, которая будет выполнена при уничтожении объекта.

```
<?php
class MyClass {
        function __destruct() {
          print "Destroying object";
        }
}
?>
```

- Интерфейсы.

  Класс может наследовать только один класс, но при этом может иметь столько интерфейсов, сколько потребуется.

```
<?php
interface Display {
        function display();
}

class Circle implements Display {
        function display() {
          print "Displaying circle ";
        }
}
?>
```

- Оператор `instanceof`.

  Поддержка проверки зависимости от других объектов. Функцией is_a(), известной из PHP 4, пользоваться теперь не рекомендуется.

```
<?php
if ($obj instance of Circle) {
        print '$obj is a Circle';
}
?>
```

- Метод `final`.

  Ключевое слово `final` позволяет вам помечать методы, чтобы наследующий класс не мог перегрузить их.

```
<?php
class MyClass {
        final function getBaseClassName() {
          return __CLASS__;
        }
}
?>
```

- Классы, помеченные как `final`.

  После объявления класса `final` он не может быть унаследован. Следующий пример вызовет ошибку:

```
<?php
final class FinalClass {
}

class BogusClass extends FinalClass {
}
?>
```

- Явное клонирование объекта

  Чтобы явно клонировать объект, вы должны использовать ключевое слово `clone`. Вы можете объявить метод `__clone()`, который будет вызван при клонировании объекта (после того, как все свойства будут скопированы из исходного объекта).

```
<?php
class MyClass {
        function __clone() {
          print "Object is being cloned";
        }
}
$obj = new MyClass();
clone $obj;
?>
```

- Константы класса.

  В определения классов теперь можно включить константы, и ссылаться на них, используя объект.

```
<?php
class MyClass {
        const SUCCESS = "Success";
        const FAILURE = "Failure";
}
print MyClass::SUCCESS;
?>
```

- Статические члены класса.

  Определения классов могут теперь включить статических членов класса (свойства и методы), доступ к которым осуществляется через класс. Общее использование статических членов показано на примере:

```
<?php
class Singleton {
        static private $instance = NULL;
        private function __construct() {
        }
        static public function getInstance() {
          if (self::$instance == NULL) {
          self::$instance = new Singleton();
          }
          return self::$instance;
        }
}
?>
```

- Статические методы.

  Теперь вы можете определить методы как статические, разрешая им быть вызванными вне контекста объекта. Статические методы не определяются через переменную `$this`, поскольку они не должны быть ограничены определенным объектом.

```
<?php
class MyClass {
        static function helloWorld() {
          print "Hello, world";
        }
}
MyClass::helloWorld();
?>
```

- Абстрактные классы.

  Класс может быть объявлен как абстрактный при помощи использования ключевого слова `abstract`, для исключения из обработки движком описания класса. Однако, вы можете наследовать абстрактные классы.

```
<?php
abstract class MyBaseClass {
        function display() {
          print "Default display routine being called";
        }
}
?>
```

- Абстрактные методы.

  Метод может быть объявлен как `abstract`, таким образом отложив его определение наследуемым классом. Класс, который включает абстрактные методы, должен быть объявлен как `abstract`.

```
<?php
abstract class MyBaseClass {
        abstract function display();
}
?>
```

- Указание класса как типа.

  Определения функции могут включить указание типа класса, передаваемого в качестве параметра. Если функция будет вызвана с неправильным типом, произойдет ошибка.

```
<?php
function expectsMyClass(MyClass $obj) {
}
?>
```

- Поддержка разыменования объектов, которые возвращаются методами.

  В PHP 4 вы не могли непосредственно разыменовывать объекты, которые возвращаются из методов. Вы должны были бы сначала присвоить такой объект некой фиктивной переменной. Поясним на примере.

  В PHP 4:

```
<?php
$dummy = $obj->method();
$dummy->method2();
?>
```

  В PHP 5:

```
<?php
$obj->method()->method2();
?>
```

- Итераторы.

  PHP 5 позволяет последовательно получать доступ к элементам класса через конструкцию `foreach()`.

```
<?php
$obj = new MyIteratorImplementation();
foreach ($obj as $value) {
        print "$value";
}
?>
```

- `__autoload()`.

  Многие разработчики, пишущие объектно-ориентированные приложения, создают один файл, в котором содержится определение класса.

Очень неудобно писать в начале каждого скрипта длинный список включаемых файлов по одному на каждый класс.

В PHP 5 в этом больше нет необходимости. Вы можете определить функцию `__autoload()` , которая автоматически будет вызываться в случае использования класса, который не был определен выше. Вызывая такую функцию, Zend Engine дает возможность загрузить файл с определением класса прежде, чем будет сформировано сообщение об ошибке и выполнение скрипта прекратится.

```
<?php
function __autoload($class_name) {
        include_once($class_name . "php");
}

$obj  = new MyClass1();
$obj2 = new MyClass2();
?>
```

Обработка исключений

- PHP 5 добавляет парадигму обработки исключений, вводя структуру `try/throw/catch`. Вам остается только создать объекты, которые наследуют класс исключений `Exception`.

```
<?php
class SQLException extends Exception {
        public $problem;
        function __construct($problem) {
        $this->problem = $problem;
        }
}

try {
        ...
        throw new SQLException("Couldn't connect to database");
        ...
} catch (SQLException $e) {
        print "Caught an SQLException with problem $obj->problem";
} catch (Exception $e) {
        print "Caught unrecognized exception";
}
?>
```

В настоящее время в целях обратной совместимости большинство внутренних функций не использует исключения. Однако, все новые расширения будут иметь такую возможность, и вы можете использовать такую конструкцию в своем исходном тексте. Кроме того, подобно уже существующей функции `set_error_handler()`, вы можете использовать `set_exception_handler ()`, чтобы отловить необработанное исключение прежде, чем выполнение скрипта будет закончено.

- `foreach` с сылкой.

  В PHP 4 вы не могли пройтись с помощью `foreach()` по массиву, изменяя его значения.

В PHP 5 разрешено выполнять `foreach ()`, используя признак ссылки ('&'), таким образом, меняя переменную, вы меняете элементы массива, по которому проходит итерация.

```
<?php
foreach ($array as &$value) {
        if ($value === "NULL") {
        $value = NULL;
        }
}
?>
```

- Значения по умолчанию для параметров, передаваемых по ссылке.

  В PHP 4 задать значение по умолчанию можно было только для параметров, которые передаются по значению. Теперь поддерживается установка значений по умолчанию для параметров, передаваемых по ссылке.

```
<?php
function my_func(&$arg = null) {
        if ($arg === NULL) {
          print '$arg is empty';
        }
}
my_func();

?>
```

XML и Web Services

После изменений, касающихся самого языка, изменения в работе с XML в PHP 5 являются, вероятно, самыми существенными и захватывающими. Расширение функциональных возможностей XML в PHP 5 делают язык полностью равноправным другим, используемыми в сети.

Основы

Поддержка XML в PHP 4 базировалась на разнообразных библиотеках XML. Поддержка SAX осуществлялась старой библиотекой Expat, для использования XSLT необходима была библиотека Sablotron, для DOM использовалась более мощная libxml2 - библиотека проекта GNOME.

Использование разнообразных сторонних библиотек не делало PHP 4 лучше других языков, когда дело касалось поддержки XML. Сопровождение библиотек расширений было слабое, новые стандарты XML не всегда поддерживались, производительность была не столь хороша, как была возможна, взаимодействия между различными расширениями XML не существовало.

В PHP 5 все расширения XML были переписаны, чтобы использовать отличный набор инструментов, предоставляемый libxml2 (http://www.xmlsoft.org/). Это библиотека богатая возможностями, отлично сопровождаемая и эффективно реализовавшая стандарты XML, предоставляющая передовые возможности технологии XML в PHP.

Все вышеупомянутые расширения (SAX, DOM и XSLT) теперь используют libxml2, включая новые дополнительные расширения – SimpleXML и SOAP.

SAX

Как говорилось выше, новая реализация SAX переведена от использования Expat к libxml2. Хотя новое расширение должно быть совместимо, могут быть некоторые тонкие различия. Разработчики, которые хотят продолжать работать с библиотекой Expat, могут это сделать, конфигурируя и собирая PHP соответствующим образом (не рекомендовано).

DOM

Хотя поддержка DOM в PHP 4 была также основана на библиотеке libxml2, она изобиловала ошибками, грешила утечками памяти, и API во многих случаях не соответствовал рекомендациям W3C. Расширение DOM было полностью переписано для PHP 5. Мало того, что расширение было коренным образом переписано, теперь оно соответствует рекомендациям W3C. Например, названия функций теперь используют нотацию `studlyCaps` (присваивание имен со смешанным употреблением заглавных и строчных букв) как предписано по стандарту W3C, облегчая для вас применение прочитанного в документации W3C в PHP. Кроме того, расширение DOM теперь поддерживает три вида схем для валидации XML документов – DTD, XML Schema и RelaxNG.

В результате этих изменений код, написанный для PHP 4 с использованием DOM, не всегда будет выполняться в PHP 5. Однако, простая корректировка названий функций к новому стандарту в большинстве случаев решает проблему.

XSLT

В PHP 4, было два расширения, которые поддерживали XSL-преобразования. Первое использовало Sablotron, а второе – поддержку XSLT в расширении DOM. В PHP 5 новое расширение XSL было основано на использовании libxml2. Так, в PHP 5 XSL-преобразование не принимает таблицу стилей XSLT в качестве параметра, но зависит от расширения DOM, чтобы загрузить ее. Таблица стилей может кэшироваться в памяти, и может быть применена ко многим документам для существенной экономии времени выполнения.

SimpleXML

Вероятно, через год или два, оглянувшись назад, мы сможем сказать, что SimpleXML коренным образом изменил работу с XML-документами для PHP разработчиков.

SimpleXML можно было бы действительно называть «XML для чайников». Вместо того, чтобы иметь дело с DOM или, что еще хуже, SAX, SimpleXML позволяет представить ваш XML-документ как родной объект PHP. Вы можете читать, писать или пробегаться по вашему XML-документу, с легкой непринужденностью получая доступ к элементам и атрибутам.

Рассмотрите следующий XML-документ:

```
<clients>
<client>
    <name>John Doe</name>
    <account_number>87234838</account_number>
</client>
<client>
    <name>Janet Smith</name>
    <account_number>72384329</account_number>
</client>
</clients>
```

Следующий фрагмент кода печатает имя каждого клиента и номер его аккаунта:

```
<?php
$clients = simplexml_load_file('clients.xml');
foreach ($clients->client as $client) {
        print "$client->name has account number $client->account_number ";
}
?>
```

Как видим, SimpleXML действительно прост.

А в случае, если есть необходимость сделать что-то, что невозможно выполнить в рамках SimpleXML, вы можете преобразовать свой объект SimpleXML в дерево DOM, вызвав функцию dom_import_simplexml(), выполнить необходимое, и вернуться назад к SimpleXML, используя simplexml_import_dom(). Благодаря тому, что оба расширения реализованы в одной библиотеке XML, переключения между ними теперь реальны.

SOAP

Официально поддержка SOAP в PHP 4 отсутствовала. Обычно, при необходимости использовать SOAP, приходилось использовать PEAR, но поскольку код был написан полностью на PHP, приложение не могло выполняться так же, как встроенное C-расширение. Другие доступные C-расширения так и не смогли достичь стабильной версии и широкого применения, и поэтому не были включены в PHP 5.

Поддержка SOAP в PHP 5 была полностью переписана как C-расширение и, хотя на текущий момент находится последней стадии в бета-тестирования, было решено включить в его стандартный дистрибутив, поскольку он практически полностью реализует стандарт SOAP.

Следующий код демонстрирует вызов функции `SomeFunction ()`, определенной в WSDL-файле:

```
<?php
$client = new SoapClient("some.wsdl");
$client->SomeFunction($a, $b, $c);
?>
```

Новое расширение MySQLi (усовершенствованная MySQL)

В PHP 5 для MySQL AB (http://www.mysql.com/) было написано новое расширение MySQL, которое позволяет вам полностью использовать преимущества новых функциональных возможностей в MySQL 4.1 и более новых версий. В противовес старому расширению MySQL новое дает вам возможность использовать оба интерфейса: функциональный и объектно-ориентированный. Так что теперь у вас есть выбор что предпочесть. Новые возможности, поддерживаемые этим расширением, включают контроль транзакций, поддержку репликаций, SSL и многое другое...

Расширение SQLite

Поддержка SQLite (http://www.sqlite.org/) изначально была введена в PHP 4.3.x. Это встроенная библиотека SQL, которая не требует SQL сервера и очень подходит для приложений, которые не требуют масштабируемых SQL-серверов, или если вы разворачиваете свое приложение у провайдера, который не предоставляет вам доступ к SQL-серверу. Вопреки названию, SQLite очень богата возможностями и поддерживает транзакции, вложенные выборки, представления (view) и большие DB-файлы. Здесь это упомянуто, как возможность PHP 5, потому что поддержка SQLite была введена довольно в поздних релизах PHP 4, и поскольку PHP 5 предоставляет новый объектно-ориентированный интерфейс и поддерживает итераторы.

Tidy расширение

PHP 5 включает поддержку очень полезной библиотеки Tidy (http://tidy.sf.net/). Она позволяет разработчикам PHP разбирать, диагностировать, чистить и восстанавливать документы HTML. Tidy расширение поддерживает как функциональный, так и объектно-ориентированный интерфейс, и ее API использует механизм исключений PHP 5.

Perl extension

Хотя и не включенное по умолчанию в PHP 5, расширение Perl позволяет вам вызывать Perl-скрипты, использовать объекты Perl и использовать другие функциональные возможности Perl прямо из кода PHP. Это новое расширение можно найти в репозитарии PECL http://pecl.php.net/package/perl.

Новый менеджер памяти

Zend Engine использует новый менеджер памяти. Два главных его преимущества: улучшенная поддержка многопоточных сред (распределенные блоки не нуждаются во взаимных исключительных блокировках) и то, что после каждого запроса намного эффективнее освобождаются распределенные блоки памяти. Поскольку это касается в основном изменений инфраструктуры, вы, как конечный пользователь, не заметите этого.

Прекращена поддержка Windows 95

Запуск PHP на платформе Windows 95 более не поддерживается, потому что невозможна поддержка функциональных возможностей, которые использует PHP. Поскольку Microsoft официально прекратила поддерживать эту платформу более года назад, разработчики PHP решили, что это будет мудрое решение.

Выводы

Вы, наверное, впечатлены количеством усовершенствований PHP 5. Как говорилось выше, эта глава не сможет раскрыть все усовершенствования, тут нашли отражения только главные. Следующие главы книги дадут вам всестороннее освещение упомянутых новых особенностей и других, которые были опущены.

Статья предоставлена сайтом http://detail.phpclub.net

Переход к PHP 5

Автор:
Александр Неткачев

PHP, фактически, является самым распространенным языком для веб-программирования. В достижении этого статуса он прошел множество этапов, от простого языка для веб-программирования который уступал многим (php3), до сегодняшнего быстрого мощного и расширяемого (PHP4). Тем более приятно, что PHP не стоит на месте и продолжает развиваться оставаясь таким-же простым для начинающих и предоставляя все больше возможностей более квалифицированным разработчикам.

Введение

В преддверьи выхода PHP5 появляется множество информации о улучшениях в PHP 5, включая информацию от самих разработчиков. Например, замечательная статья от одного из авторов Zend Engine – Zeev Suraski (http://detail.phpclub.net/article/2004-01-07 или выше в данном номере журнала) и её перевод на phpclub.ru очень помогут в понимании тонкостей изменения работы с объектами. Существуют, конечно, и общие обзоры возможностей PHP5, сделанные как отечественными так и зарубежными авторами. В своем обзоре я попытался дать наиболее полное представление о изменениях в PHP5, на сколько это возможно. Эта статья также является в некотором роде «отчетом о проделанной работе», поскольку обзор каждого изменения сопровождался детальным его изучением и попыткой использовать на практике.

Хочу также выразить благодарность людям, внесшим вклад в улучшение этой статьи и обсуждавшим её на форуме phpclub.ru.

На этом заканчиваем введение и приступаем к основной части.

Изменения в PHP5

Новые уровни доступа private и public

В PHP5 добавлены новые модификаторы уровня доступа для переменных классов. Как и во многих других языках программирования, они носят названия `private`, `protected` и `public`.

`Private` – самый ограничивающий модификатор. Private переменная класса может быть использована только в классе, в котором она объявлена. К ней невозможно обратиться из другого программного кода.

`Protected` – расширение области `private`, добавляющее к ней возможность обращаться к переменной из классов-потомков.

`Public` – расширяющий `protected` модификатор, задающий наиболее широкую область доступа. К возможности использовать переменную в классах-потомках, добавляется возможность обращаться к переменной непосредственно из другого кода. Строго говоря, `public` не является новой областью доступа. Ранее в PHP все переменные классов являлись public-переменными.

`Private` переменные применяются для алгоритмов, которые используются только в текущем классе и не могут быть переопределены в классах-потомках. `Protected` может быть использован, когда организовывается семейство объектов, обладающих сходными алгоритмами и организованных в иерархию. Использование `public` переменных, обычно, не является хорошей практикой но иногда оправданно. Их можно использовать, если у класса много свойств, которые должны быть доступны всем алгоритмам, использующим этот класс.

Аналогично, `private/protected/public` модификаторы применяются к методам класса. Методы, объявленные без модификатора, являются `public` методами.

Если метод или переменная переопределяются в классе-наследнике, то уровень доступа должен быть таким-же или выше. Например, `protected` метод в классе-потомке можно сделать `public`, но нельзя `private`.

Для примера рассмотрим классы NewClass и NewClass1.

```
<?php
class NewClass {
    // new PHP5 modifiers
    private $myPrivateVar = 'myPrivateVar';
    protected $myProtectedVar = 'myProtectedVar';
    public $myPublicVar = 'myPublicVar';
    // old PHP declaration
    var $myVar = 'myVar';
}

class NewClass1 extends NewClass {
    function getProtectedVar() {
        return $this->myProtectedVar;
    }
}
?>
```

NewClass содержит несколько переменных с разными областями доступа. NewClass1 используется для тестирования областей видимости, связанных с наследованием.

Создаем объекты классов:

```
$c = new NewClass();
$c1 = new NewClass1();
```

Обращаемся к переменным:

print $c->myPrivateVar; Непосредственное обращение к `private` переменной приводит к ошибке.

print $c->myProtectedVar; Непосредственное обращение к `protected` переменной приводит к ошибке.

print $c->myPublicVar; Обращение к `public` переменной возвращает её значение.

print $c->myVar; Обращение к переменной, объявленной в старом стиле, равносильно обращению к `public` переменной.

print $c1->myPrivateVar; `Private` переменная не была унаследована классом NewClass1. Обращение к ней равнозначно обращению к необъявленной переменной.

print $c1->myProtectedVar; `Protected` переменная была унаследована и непосредственное обращение к ней приводит к ошибке. Для проверки, что она была унаследована вместе с начальным значением, можно вызвать "print $c1->getProtectedVar();".

print $c1->myPublicVar; `Public` переменная была унаследована и обращение к ней возвращает её значение.

Абстрактные классы и методы (abstract)

Абстрактные классы используются для создания семейства объектов, обладающих единым интерфейсом. Также они используются, когда нужно запретить создание объекта некоторого класса.

Пример создания и использования абстрактного класса:

```
<?php

abstract class NewClass {
    abstract function myMethod();
}

class NewClass1 extends NewClass {
    function myMethod() {
        return 'myMethod';
    }
}

$c = new NewClass1();
print $c->myMethod();

?>
```

Если метод определяется как `abstract`, он должен быть переопределен в классе-потомке. При этом параметры переопределённого метода должны совпадать с параметрами абстрактного метода. Модификатор уровня доступа для абстрактных методов не учитывается. Уровень доступа определяется методом, переопределяющим абстрактный.

Интерфейсы (interface)

Интерфейсы похожи на абстрактные классы, за исключением того, что использование интерфейсов позволяет использовать множественное наследование. Таким образом, класс может реализовывать несколько интерфейсов одновременно, а не расширять только один абстрактный класс.

Пример использования интерфейса:

```
<?php

interface Printable {
    public function dump();
}
interface Editable {
    public function edit();
}

class NewClass implements Printable, Editable {
    function dump() { }
    function edit() { }
}

$c = new NewClass();
print (($c instanceof Printable) ? 'true' : 'false');

?>
```

Типизация пареметров функций на уровне классов

Для параметров функций можно задавать класс, объект которого может быть передан по этому параметру. Во время работы скрипта, конструкция

```
function myFunction(MyClass $obj) {
}
```

равнозначна конструкции

```
<?php

function myFunction($obj) {
    if (!($obj instanceof MyClass || $obj == null)) {
        die('Argument 1 must be an instance of ClassName');
    }
}

?>
```

При этом `instanceof` распространяется не только на имя класса, но и на всех его предков и реализуемые интерфейсы.

Например, следующий код выполнится без ошибок:

```
<?php

interface Editable {
    function edit();
}

abstract class View {
    abstract function createView();
}

class NewClass extends View implements Editable {
    function createView() { }
    function edit() { }
    function createMyView(View $obj) { }
    function doEdit(Editable $obj) { }
}

$c = new NewClass();
$c->createMyView($c);
$c->doEdit($c);

?>
```

Финальные классы и методы (final)

Финальный метод невозможно переопределить в классе-наследнике. Финальный класс невозможно использовать для создания классов-наследников. Это может пригодиться, когда необходимо сохранить алгоритм, инкапсулированный в классе, неизменным. Например, что бы ограничить программиста, использующего библиотеку, от переопределения поведения. Использование финальных классов вместе с типизацией параметров функций создает практически 100% препятствие на пути расширения или подмены функциональности. Естественно, при открытом исходном коде убрать `final` у класса или метода не является трудной задачей, но, например, `final` часто используется у классов, определенных в самом PHP или его расширениях (Exception class, DOM extention).

Пример финального класса и финального метода:

```
<?php

final class Security {
    function createUser() {
        ...
    }
}

class View {
    final static function createView(Security $user) {
        ...
    }
}

?>
```

Поскольку класс Security является финальным, а параметром функции View::createView может быть только объект финального класса или null, это дает 100% гарантию, что в если в функцию createView будет передан объект, то это будет только объект класса Security, а не подмененный.

Клонирование объектов

В PHP4 для клонирования объекта достаточно было простой операции $clonedObject = $object. Все свойства объекта $object просто копировались в объект $clonedObject. Изменить алгоритм клонирования можно было написав собственный метод для этого. В PHP5 для этого метода ввели специальное имя `__clone` и упростили доступ к созданному объекту. Для обращения к новому объекту используется `$this`, для обращения к уже существующему (чей клон делается), соответственно, `$that`.

Если метода `__clone` нет, то вызовется стандартный метод, копирующий все свойства объекта.

На примере это выглядит так:

```
<?php

class Node {
    private $next;
    private $name;

    function __clone() {
        $this->name = $that->name;
        $this->next = null;
    }

    function setName($name) { $this->name = $name; }
    function getName() { return $this->name; }
    function setNext(Node $next) { $this->next = $next; }
}

$n1 = new Node();
$n1->setName('Node1');

$n2 = new Node();
$n2->setName('Node2');
$n1->setNext($n2);

$n = $n2->__clone();
print_r($n);

?>
```

В примере рассматривается класс для создания списка, т.е. цепочки объектов, в которой каждый объект содержит указатель на следующий. При этом можно получить клон любого объекта в цепочке, и новый объект будет «вынутым» из цепочки (не содержать ссылки на следующий объект).

Пример также демонстрирует, что к можно внутри метода `__clone` можно получить доступ к `private` переменным объектов `$this` и `$that`.

Конструкторы

Основным недостатком структуры конструкторов в PHP4 является необходимость синхронизации имени конструктора и имени класса. Поскольку имя конструктора должно совпадать с именем класса, то, при изменении имени класса, приходится переименовывать и конструкторы. В случае, если класс имеет несколько наследников, приходится аккуратно изменять в классах наследниках наследуемый класс (extends) и вызов конструктора класса-предка (parent).

Введение в PHP5 конструктора для класса с общим именем `__construct` упрощает переименовывание классов во время их разработки. Если в классе есть и `__construct` и функция, имя которой совпадает с именем класса, то в качестве конструктора будет вызванно `__construct`. При перегрузке метода-конструктора вызов конструктора класса-предка осуществляется через `parent::__construct()`.

Пример использования конструкторов:

```
<?php

class NewClass1 {
    function __construct() {
        print 'NewClass1::__construct called';
    }
}

class NewClass2 extends NewClass1 {
}

class NewClass3 extends newClass2 {
    function __construct() {
        print 'NewClass3::__construct called';
        parent::__construct();
    }
}

$n1 = new NewClass1();
// выводится NewClass1::__construct called

$n2 = new NewClass2();
// выводится NewClass1::__construct called - конструктор унаследован и
//вызван

$n3 = new NewClass3();
// выводится NewClass3::__construct called и NewClass1::__construct called


?>
```

При этом, если конструктор объявлен с модификатором `private`, то класс с таким конструктором создать невозможно. Однако обращение `parent::__construct` возможно.

Это дает еще один способ избежать создания класса, помимо объявления его `abstract`.

Деструкторы

Деструкторы являются нововведением для PHP. Они очень полезны для совершения работы по освобождению ресурсов, таких как закрытие открытых файлов или соединения с базой данных. Для деструкторов определенно имя `__destruct`. Как и для конструкторов, если деструктор унаследован и не перегружен он вызовется. Если он перегружен, то вызовется только перегруженный конструктор. Для вызова деструктора объекта-предка надо использовать `parent::__destruct()`. Деструктор вызывается без параметров.

Пример использования деструктора:

```
<?php

class Computer {
    function compute() {
        // большие ресурсоемкие вычисления.
    }

    function __destruct() {
        // отправить письмо, что все выполнилось
    }
}

$c = new Computer();
$c->compute();

?>
```

Константы

В классах могут быть объявлены константы. Это является еще одним методом (вместе с `final` классами и методами) для повышения структурности и удобочитаемости кода.

Пример определения и использования констант:

```
<?php
final class ControlTypes {
    const Textbox = 1;
    const Label = 2;
    const Listbox = 3;
    const Textarea = 4;
    const Link = 7;
    const Button = 6;
}
class Control {
    private $type;
    function __construct($type) {
        $this->type = $type;
    }
}
$c = new Control(ControlTypes::Textbox);
?>
```

К константам невозможно применять модификаторы `public, protected, private`. Константы всегда `public`. Обращаться к константам можно только через имя класса, например `ControlType::Textbox`. Обращения через `$this` или другой указатель на объект класса не поддерживаются. В константе может быть только значение примитивного типа, т.е. строка или число. Константы наследуются и могут быть переопределены в классах-потомках.

Интересной особенностью является то, что интерфейсы могут содержать константы. Например:

```
interface myInterface {
    const test = 2;
}
```

Система перехвата исключений (exceptions)

Exceptions (исключения) – это неотъемлемая часть любого современного языка. Система перехвата исключений объединяет в себе оператор `throw`, стрктуру языка "`try` { .. } `catch` ()[`catch` () ...]" и основной объект `Exception`. В отличии от Java exceptions, в PHP отсутствует завершающий блок `finally`.

Основное применение системы исключений состоит в использовании структуры `try/catch` для отделения основного кода программы и блоков обработки ошибок. Механизм exceptions позволяет также корректно обрабатывать исключения, возникшие не непосредственно в выполняемом коде, а в используемых функциях.

Следующий пример демонстрирует отделение кода от обработчиков нестандартных ситуаций:

```
<?php

/**
* Замечания:
*   Конструктор DatabaseConnection может бросить DatabaseException
*   Метод getUser() может бросить UserNotFoundException
*   Метод sendMail() может бросить MailServiceException
*/
try {
  $cn = new DatabaseConnection();
  $admin = cn->getUser('Admin');
  $admin->sendMail('Database check is complete');
} catch (DatabaseException $e) {
   print "Невозможно создать соединение с базой данных. Причина: " . $e-
>getMessage();
} catch (UserNotFoundException $e) {
  print "Пользователя не существует";
} catch (MailServiceException $e) {
  print "Ошибка отправки письма: " . $e->getMessage();
} catch (Exception $e) {
  print "Общая ошибка: " . $e->getMessage();
}

?>
```

В общем случае, использование системы исключений можно заменить на использование структур `if` и `goto` или только `if`, но код программы в результате становиться значительно более громоздким.

Система исключений в PHP работает только с исключениями, «бросаемыми» оператором `throw`. Ошибки синтаксиса языка не обрабатываются блоками `try/catch` по очевидным причинам.

В PHP на данный момент определен только один класс исключений: `Exception`. Для более гибкой работы с системой сообщений можно добавлять свои классы исключений но наследовать их от базового класса `Exception`, чтобы всегда можно было поймать исключение (catch exception).

Основными методами класса `Exception` являются: `getMessage()`, `getCode()`, `getTrace()`, `getFile()`, `getTraceAsString()`, `_toString()`. Все методы являются финальными, кроме конструктора и `_toString()`. Таким образом, дополнительная функциональность классов-потомков `Exception` (отправка почты с информацией о ошибке, запись в log) может быть реализована в конструкторе.

Класс Exception объявляется непосредственно в PHP Engine, но его примерная модель может быть представленна таким образом (по материалам www.zend.com):

```
<?php

class Exception {
          function __construct(string $message=NULL, int $code=0) {
            if (func_num_args()) {
              $this->message = $message;
            }
            $this->code = $code;
            $this->file = __FILE__; // of throw clause
            $this->line = __LINE__; // of throw clause
            $this->trace = debug_backtrace();
            $this->string = StringFormat($this);
          }

          protected $message = 'Unknown exception';  // exception message
          protected $code = 0; // user defined exception code
          protected $file;     // source filename of exception
          protected $line;     // source line of exception

          private $trace;      // backtrace of exception
          private $string;     // internal only!!

          final function getMessage() {
            return $this->message;
          }
          final function getCode() {
            return $this->code;
          }
          final function getFile() {
            return $this->file;
          }
```

(продолжение кода на следующей странице)

```
          final function getTrace() {
            return $this->trace;
          }
          final function getTraceAsString() {
            return self::TraceFormat($this);
          }
          function _toString() {
            return $this->string;
          }
          static private function StringFormat(Exception $exception) {
            // ... a function not available in PHP scripts
            // that returns all relevant information as a string
          }
          static private function TraceFormat(Exception $exception) {
            // ... a function not available in PHP scripts
            // that returns the backtrace as a string
          }
        }

?>
```

Использование объектов без ссылок на них

Очень серьезным неудобством в PHP4 было вызывание цепочки методов. В PHP4 невозможно создать объект без ссылки на него, поскольку объекты фактически являлись только синтаксической конструкцией и на уровне ядра были эквивалентны массивам. Это порождало, например, такие конструкции:

```
$page = &$this->getPage();
$page->registerControl($this);
```

Конечно, это не очень удобно. Созданная на уровне ядра PHP5, таблица ссылок на объекты делает необязательным существование ссылок на объект. Благодаря этому становится возможной следующая конструкция:

```
$this->getPage()->registerControl($this);
```

Но нужно заметить, что хотя такой подход и более краток по написанию, неразумное его использование чревато очень нерациональным кодом. Например, крайне не рекомендуется делать таким образом:

```
for($i = 0; $i < 100; $i++)
    $myObject->getProperty('relatedObject')->getAncestor($i)->update();
```

Во время работы этого кода осуществляется создание двухсот объектов и трехсот вызов методов.

Очень простым образом можно сократить до создания ста объектов и двухсот одного вызова методов:

```
$relatedObject = $myObject->getProperty('relatedObject');
for($i = 0; $i < 100; $i++)
    $relatedObject->getAncestor($i)->update();
```

Несмотря на очевидность такого подхода, довольно часто написанный код может быть улучшен с его помощью.

В следующих версиях PHP, скорее всего, можно ожидать расширения подобного подхода и к обычным массивам. Если они, конечно, еще останутся – объекты имеют тенденцию захватывать все больше и больше функциональности :-). Тогда, возможно, будет доступна следующая конструкция: print ((new ServerEnvironment()). getServerVariables())['REQUEST_URI'].

Инициализация переменных класса вне конструктора

Начальное значение переменной класса теперь можно указать непосредственно при её объявлении. Однако, её значение может быть только примитивного типа, т.е. строкой или числом. Тем не менее, этот метод является единственно возможным для задания значения статической переменной класса. Например:

```
class MathUtils {
    static private pi = 3.1415926;
    ...
}
```

По другому pi определить невозможно, поскольку для статических переменных не существует "статического" конструктора.

Статические методы класса

Статические методы класса могут быть вызваны непосредственно у класса, а не через его один из его объектов. Соответственно, указатель `$this` в статических методах недоступен.

Фактически, объявление класса со статическими методами является, в большей мере, методом группировки функций и общих для них констант и переменных.

Применение такого подхода гарантирует, что все классы доступа к базе данных будут реализовывать один интерфейс (заменяемость), уменьшает вероятность конфликтности имен, упрощает существование нескольких версий класса доступа к базе и т.д.

instanceof оператор

Новый оператор «проверяемый объект instanceof проверяемый класс» позволяет проверить, попадает ли проверяемый класс в список дерева наследования класса, экземпляром которого является проверяемый объект.

На примере это выглядит так:

```
<?php

interface Editable {
 function startEdit();
 function endEdit();
}

class Control {
 function getValue() {
 //...
 }
}

class EditableControl extends Control implements Editable {
 function startEdit() {
 //...
 }
 function endEdit() {
 //...
 }
}

$c = new Control();
$ec = new EditableControl();
print '$c instanceof Editable = ' . ($c instanceof Editable ? 'true' :
'false') . '<br>';

print '$c instanceof Control = ' . ($c instanceof Control ? 'true' :
'false') . '<br>';

print '$c instanceof EditableControl = ' . ($c instanceof EditableControl
? 'true' : 'false') . '<br>';

print '$ec instanceof Editable = ' . ($ec instanceof Editable ? 'true' :
'false') . '<br>';

print '$ec instanceof Control = ' . ($ec instanceof Control ? 'true' :
'false') . '<br>';

print '$ec instanceof EditableControl = ' . ($ec instanceof
EditableControl ? 'true' : 'false');

?>
```

Результатом работы этого кода будет:

```
$c instanceof Editable = false
$c instanceof Control = true
$c instanceof EditableControl = false
$ec instanceof Editable = true
$ec instanceof Control = true
$ec instanceof EditableControl = true
```

Таким образом, для $c instanceof возвращает true только для класса Control, для $ec instanceof вернет true только для Editable, Control, EditableControl. Для null всегда возвращается false.

Статичекие переменные функций

Переменные внутри функции могут быть объявлены как `static`. Static переменная функции – это общая переменная для всех вызовов этой функции. Static переменная по смыслу примерно равна глобальной переменной, используемой только внутри функции.

Необязательные передающиеся по ссылке параметры функций

Передающиеся по ссылке параметры в PHP4 не могут иметь default значение. Это приводит к невозможности сделать функцию с необязательным объектным параметром. Но общественность требовала и в PHP5 появилась возможность задать для объектного параметра значение по умолчанию. Надо заметить, что возможно единственное значение по умолчанию для таких параметров – null.

Пример использования:

```
class Unrequired {
    ...
}
function myFunction(Unrequired $param = null) {
    ...
}
myFunction();
myFunction(new Unrequired());
```

Функция-событие при создании объекта неизвестного класса (__autoload())

PHP не держит все приложение в памяти. Более того, для каждой страницы он заново подгружает все файлы с кодом и преобразует в удобную для выполнения форму. Хорошо помогают различные акселераторы PHP кода, которые сохраняют в памяти непосредственно преобразованный в исполняемый код php-страницу. Но даже в случае использования такого оптимизатора нежелательно подключать к скрипту все файлы с классами и функциями, которые могут понадобится, но реально не используются. Настройка подключение только необходимых классов к каждой конкретной странице – занятие, требующее большой аккуратности и вызывающее большое желание это каким-то образом автоматизировать.

Возможно, именно поэтому и была введена функция-событие с названием `__autoload()`, которая срабатывает при попытке обращения к неизвестному классу или интерфейсу. Под обращением понимается попытка создания объекта класса, создание класса-потомка на основе класса, создание класса, реализующего интерфейс.

Еще одна проблема, которую снимает `__autoload` – это размещение включений файлов в порядке иерархии наследования. Например, если MyClass1 находится в файле MyClass1.php, а MyClass2 – в файле MyClass2.php и MyClass2 extends MyClass1, то с помощью include их надо подключать только в порядке inlude ('MyClass1.php'); include('MyClass2.php'); Когда 2 файла – не страшно. Но когда их несколько десятков – это уже сложнее.

И, наконец, пример использования __autoload:

```
test.php ============================
function __autoload($name) {
    include_once('classes/' . $name . '.php');
}
$t = new Textbox();

Control.php =========================
class Control {
    // ...
}
Textbox.php =========================
class Textbox extends Control {
    // ...
}
```

При попытке создания Textbox будет загружен файл Textbox.php. Поскольку Textbox extends Control, тут же будет загружен Control.php.

Функции-события при обращении к свойству класса (__get(), __set())

Функции __get и __set могут рассматриваться как возможность реализации свойств, аналогичным свойствам в .NET, VBScript (ASP) или VB. Но в отличие от перечисленных языков (технологий), в PHP __get и __set выполняются для всех (!) свойств.

Вызов методов __get() и __set() при обращении к свойству происходит только если переменной класса с таким именем не существует. Если она существует, то в результате обращения из основной программы можно получить либо ошибку (если переменная private или protected), либо, собственно, переменную (если она public).

Цепочки свойств ($myObj->parent->value) работают корректно. Пример:

```
class Node {
    private $mValue = 1;
    private $mParent;
    function __get($name) {
        switch($name) {
            case 'Value':
                return $this->mValue;
            case 'Parent':
                return $this->mParent;
        }
    }
    function __set($name, $value) {
        switch($name) {
            case 'Value':
                $this->mValue = $value;
                break;
            case 'Parent':
```

(продолжение кода на следующей странице)

```
            $this->mParent = $value;
 break;
 }
 }
}
$n1 = new Node();
$n2 = new Node();
$n2->Parent = $n1;
$n1->Value = 2;
print $n2->Parent->Value; // Выводит 2.
```

Функция-событие при обращении к методу класса (__call())

Функция-событие `__call()`, возможно, введена вместе с `__get()` и `__set()`. Скорее всего эта функция найдет свое применение в дальнейшем. Например, она может применяться для эмуляции перегрузки методов.

Итерация по свойствам класса

Все переменные класса, доступные в текущем контексте, могут быть перебраны циклом `foreach`. Такая итерация по свойствам класса может очень пригодится при клонировании объектов. Например, если необходимо создать клон объекта с большим количеством переменных класса, то можно сделать примерно так:

```
class Node {
    private $value;
    private $parent;
    ...
    private $type;

    function __clone() {
        foreach ($that as $propertyName => $propertyValue) {
            $this->$propertyName = $propertyValue;
        }
    }

    function setParent($value) { $this->parent = $value; }
    function getParent() { return $this->parent; }
    function setValue($value) { $this->value = $value; }
    function getValue() { return $this->value; }
    ...
    function setType($value) { $this->type = $value; }
    function getType() { return $this->type; }
}
$myNode = new Node();
$myNode->setValue(10);
$myNextNode = $myNode->__clone();
print $myNextNode->getValue();
```

Простой цикл очень хорошо заменяет большое количество присваиваний и избавляет от необходимости синхронизировать присваивания при клонировании со списком всех переменных класса. Очевидным образом, эта итерация не может быть применена к свойствам класса, реализованных через `__get()`/`__set()` функции.

Изменение стандартной итерации по свойствам

Стандартными элементами языка, позволяющими итерацию, являются массивы. Но их недостатки очевидны – можно по ошибке поместить в массив элемент другого типа. К массиву не добавишь методы проверки содержимого и т.п.

Для введения дополнительных сущностей (классов), позволяющих итерацию по своим элементам, предусмотрено 2 интерфейса: `IteratorAggregate` и `Iterator`.

```
interface IteratorAggregate {
    function getIterator(); // возвращает массив или объект
}
```

`IteratorAggregate` может использоваться, когда данные для итерации можно предоставить в одной из стандартных конструкций PHP, позволяющих итерацию: массива или объекта, реализующего `Iterator`.

```
interface Iterator {
    function rewind(); // переводит итерацию к первому элементу
    function next(); // подготавливает к выводу следующий элемент
    function key(); // возвращает ключ текущего элемента
    function current(); // возвращает текущий элемент
     function hasMore(); // возвращает true, если есть еще элементы, иначе
false
}
```

Метод __toString()

Когда переменная-объект преобразуется к строке, в результате возвращается строка "Object id #n", где n – номер объекта в глобальной таблице объектов. Если понадобится (пусть и крайне редко), этот механизм можно изменить, создав у класса метод `__toString ()`, возвращающий некоторое строковое представление текущего объекта.

Хотя PHP5 beta 3 этот алгоритм проработан не полностью (`__toString()` срабатывает только во время использования указателя на объект в операторе `print`), это открывает интересные перспективы. Например, следующий код представляет собой вариацию на тему типизации PHP:

```
class Integer {
    private $value;
    function __construct($val) {
        $this->value = $val;
    }
    function __toString() {
        return (string)($this->value);
    }
}
```

(продолжение кода на следующей странице)

```php
$i = new Integer(10);
/**
    Теоретически, $i при преобразовании к строке должно дать "10",
    и, поскольку число 10 сравнивается со строкой, оно тоже должно
    быть приведено к строке. Получится "10" == "10". На практике, в
    этом случае преобразование $i к строке осуществляется по варианту
    PHP4 (т.е. в результате получаем строку "Object").
*/

if (10 == $i)
    echo '10!!!! :-)';
```

Reflection API

`Reflection` не является новым понятием для PHP, но только в PHP5 предпринята попытка привести работу со структурными объектами языка к общему виду. Под структурными объектами понимаются функции, классы, интерфейсы, параметры и расширения.

Классы Reflection позволяют получать информацию о объектах языка непосредственно во время выполнения скрипта. Например, можно получить информацию о некотором объекте, включая его методы, их параметры, в каком файле находится описание объекта и даже какой документационный комментарий находится перед ним.

Reflection классы разработаны для каждого структурного объекта языка:

- `Reflection_Function`
- `Reflection_Parameter`
- `Reflection_Method`
- `Reflection_Class`
- `Reflection_Property`
- `Reflection_Extension`

В отличии от большинства других изменений в PHP5, Reflections уже неплохо документированы. Подробное описание доступно по адресу http://sitten-polizei.de/php/reflection_api/docs/language.reflection.html

```php
/**
    MyClass просто пример класса для демонстрации Reflection.
        Этот код будет выведен как документация к классу MyClass при
Reflection.
*/
class MyClass {
    /**
        А это комментарий к конструктору
    */
    function __construct() {
    }
}
Reflection::export(new Reflection_Class('MyClass'));
```

Результатом работы этого кода будет следующее описание класса:

```
/**
    MyClass просто пример класса для демонстрации Reflection.
       Этот код будет выведен как документация к классу MyClass при
Reflection.
*/
Class [ <user>  class myclass ] {
  @@ /home/alex/public_html/devlink_draft/articles/docs/test.php 7-14

  - Constants [0] {
  }

  - Static properties [0] {
  }

  - Static methods [0] {
  }

  - Properties [0] {
  }

  - Methods [1] {
    /**
        А это комментарий к конструктору
    */
    Method [ <user> <ctor> public method __construct ] {
       @@ /home/alex/public_html/devlink_draft/articles/docs/test.php 12 -
13
    }
  }
}
```

Статья предоставлена сайтом: http://detail.phpclub.ru
Сайт автора статьи: http://devlink.crimea.ua

Идеи

Практическое применение ORM (Object-Relational mapping) в PHP

Автор:
Киран Мэтиесон
Перевод:
nw

PHP-приложения используют множество технологий, таких как HTML, реляционные базы данных, JavaScript, объектно-ориентированное программирование (ООП) и прочие. С большинством из них PHP сочетается вполне удачно. Например, PHP может легко генерировать код JavaScript, а JavaScript в свою очередь будет обращаться к объектной модели браузера (DOM) и менять HTML.

Однако остаются такие вещи, как ООП и реляционные базы данных, во взаимодействии между которыми не все так гладко. Когда программисты пишут код в виде PHP-классов, взаимодействующий с базой данных, они должны убедиться, что классы не находятся в зависимости друг от друга. В противном случае изменения в одном классе, потребуют изменения других классов, вызывая повышение стоимости разработки и сопровождения. Если программное обеспечение становится слишком дорогим или требует больших временных затрат, то оно начинает скорее тормозить инновационные процессы бизнеса, чем подталкивать их вперед.

Эта статья рассматривает практическую реализацию ORM в PHP. Практический означает соответствующий условиям, встречающимся в реальной разработке программного обеспечения. Вот некоторые из таких условий:

- Ограниченные финансовые ресурсы;
- Ограниченные временные рамки;
- Ограниченный опыт разработки;
- Серверы, на которых будет размещено приложение, разработчиками полностью не контролируются;
- Используются только основные инструменты, такие как PHP и MySQL;
- Предусмотрен обмен конечного приложения с другими системами, такими как Access и Excel;
- Конечное приложение должно работать с аутентификацией, правами доступа, записью результатов транзакций (ведения логов), системой обработки ошибок, одновременными обновлениями и прочим.

Мы будем использовать термин «вспомогательные задачи приложения», подразумевая под ним задачи, перечисленные в последнем элементе вышеприведенного списка. Зачастую именно эти задачи поглощают основную массу ресурсов. Аккуратное управление вспомогательными задачами очень важно для стабильности и производительности приложения. Лучше всего начать разработку таких задач в самом начале проекта, нежели потом пытаться добавить их в готовое приложение.

Изложенный в данной статье материал рассчитан на то, что вы уже знакомы с PHP, объектно-ориентированным программированием и реляционными базами данных.

Приступим

Допустим, ваша компания выращивает гиппопотамов для армии. Одни из них используются для переноса грузов, а другие экипированы оружием, таким как тортометы и плевалки. Ваша компания не устанавливает вооружение, а только готовит гнезда для его пристыковки. Экипированный гиппопотам может иметь более одной точки монтирования (гнезда для установки вооружения).

Первым делом давайте создадим таблицу Hippos для хранения информации о гиппопотамах, где будет два поля: id и name.

Поле	Тип	Примечание
id	integer	Primary Key auto_increment
name	text	

Теперь немного PHP. Для простоты некоторые методы мы пропустили.

```
<?php
class Hippo
{
    var $id = null;
    var $name = null;
    //Load record from DB
    function load($id)
    {
        ...
    }
    //Save record to DB
    function save()
    {
        ...
    }
}
?>
```

Теперь давайте напишем `Hippo::Load()` и `Hippo:Save ()`. Здесь мы используем запись `Hippo::Load()`, чтобы сослаться на `Load()` метод `Hippo`. В данном контексте такая запись не имеет отношения к статическим методам. Код получения, установки параметров и проверки ошибок опущены для простоты.

Гиппопотам в нашем случае относится к «бизнес классам», т.е. таким классам, которые описывают некий объект, имеющий определенные ценные свойства для нетехнического персонала компании.

Некоторые классы не относятся к товарам. Например, классы для объектов баз данных, таких как таблица, запись или поле, ничего не значат для торгового представителя. Однако торговый представитель поймет, что есть гиппопотам, и что у него есть свой порядковый номер и имя.

```
<?php
//Hippo methods
function load($id)
{
    $this->id = $id;
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    $result = mysql_query("select name from hippos where id=$id");
    $this->name = mysql_result($result, 0, 'name');
    application_utility_tasks();
}
function save()
{
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    if ( is_null($this->id) )
    {
        //New record
        $query = "insert into hippos (name) values ('".$this->name."')";
        mysql_query($query);
        $this->id = mysql_insert_id();
    }
    else
    {
        //Update existing record
        $query = "update hippos set name='".$this->name . "' where id=" .
$this->id;
        mysql_query($query);
    }
    application_utility_tasks();
}
?>
```

В `Hippo::Load()` мы копируем параметр метода в атрибут id. Далее открываем соединение с базой данных и получаем имя гиппопотама. `Hippo::Save()` использует атрибут id для того, чтобы определить, существует ли запись о данном гиппопотаме в базе, или это новый экземпляр. Если id равно `null`, тогда метод создает новую запись в базе. В противоположном случае – обновляет уже существующую. Вызов функции application_utility_tasks() условен и служит лишь для напоминания нам о том, что не стоит забывать и о вспомогательных задачах.

Теперь создадим наследуемый от `Hippo` класс и назовем его `ArmedHippo` (вооруженный или экипированный гиппопотам – прим. переводчика). `Hippo` имеет атрибуты id и name. Класс `ArmedHippo` унаследует эти атрибуты и добавит еще один – количество гнезд для установки вооружения.

```php
<?php
class ArmedHippo extends Hippo
{
    var $num_mount_points = null;
    //Load record from DB
    function load($id)
    {
        ...
    }
    //Save record to DB
    function save()
    {
         ...
    }
}
?>
```

В качестве следующего шага, необходимо написать `ArmedHippo::load()` и `ArmedHippo::save()`. И вот теперь могут возникнуть определенные трудности. Если мы не будем осмотрительными, то случайно можем написать код, который будет трудно расширять. Чем меньше мы уделим внимания подготовке базиса, тем сложнее будет работать с приложением в дальнейшем.

Давайте взглянем, что может произойти в дальнейшем:

- Выполнение `ArmedHippo::load()` и `ArmedHippo::save ();`
- Запись веса каждого гиппопотама. В PHP4 самый простой способ сделать это – изменить класс `Hippo`, добавив в него новый атрибут. Но сможем ли мы сделать это, никак не повлияв на класс `ArmedHippo`?
- Запись информации о скрытности гиппопотамов. Чтобы повысить скрытность гиппопотама, его можно покрасить в черный цвет или снабдить системой обнаружения и подавления запахов (СОПЗ – это специальное военное обозначение дезодоранта). Класс `StealthHippo` расширит `ArmedHippo`. Сможем ли мы добавить его, не повлияв на другие классы?

Взглянем на некоторые варианты.

Вариант 1: повторение кода

Самый простой путь для выполнения методов по работе с БД из `ArmedHippo` – это добавление нового столбца в нашу таблицу `Hippos`. Она будет выглядеть вот так:

Поле	Тип	Примечание
id	integer	Primary Key auto_increment
name	text	
num_mount_points	integer	

Теперь можно скопировать код из `Hippo::load()` и `Hippo::save()` и вставить его в `ArmedHippo`, добавив num_mount_points.

```
<?php
//ArmedHippo methods
function load($id) {
    $this->id = $id;
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    $result = mysql_query("select name,
    num_mount_points from hippos where id=$id");
    $this->name = mysql_result($result, 0, 'name');
    $this->num_mount_points =  mysql_result($result, 0,
'num_mount_points');
    application_utility_tasks();
}
function save()
{
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    if ( is_null($this->id) )
    {
        //New record
        $query = "insert into hippos (name, num_mount_points)
        values ('".$this->name."', ".
        $this->num_mount_points.")";
        mysql_query($query);
        $this->id = mysql_insert_id();
    }
    else
    {
        //Update existing record
        $query = "update hippos set name='".$this->name."',
        num_mount_points=".$this->num_mount_points."
        where id=".$this->id;
        mysql_query($query);
    }
    application_utility_tasks();
}
?>
```

Метод `save()` получает значение атрибута id, и если он равен null, то добавляется новая запись, при этом значение id обновляется на id только что внесенной записи. Если id не равно `null`, то оно передается как ключ в SQL-выражение `UPDATE`.

Этот вариант работает, но используя его, мы теряем некоторые преимущества ООП. Если мы добавим атрибут weight (вес – прим. переводчика) в класс `Hippo`, то будем вынуждены менять код и в `ArmedHippo::load` и в `ArmedHippo::save`. Если мы добавим еще и `StealthHippo`, то мы получим еще больше кода для правки при изменении `Hippo`. К тому же каждый метод `load()` и `save()` содержит в себе отдельный вызов `application_utility_tasks()`. Если нам будет необходимо изменить вызов, то придется менять его во всех местах.

Вариант 2: дополнительные запросы для новых полей

Другой вариант – добавить num_mount_points в таблицу Hippos (как мы это делали в предыдущем примере), но изменить `ArmedHippo` так, что бы он вызывал `Hippo::load()` для получения id и name и работал с num_mount_points самостоятельно.

Таким образом мы можем записать `ArmedHippo::load()` так:

```
<?
function load($id) {
    parent::load($id);
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    $result = mysql_query("select num_mount_points from hippos where
id=$id");
    $this->num_mount_points = mysql_result($result, 0,
'num_mount_points');
    application_utility_tasks();
}
?>
```

Во второй строке фактически вызывается `Hippo::load()` для получения id и name из базы данных. Далее уже сам метод подключается к базе, выполняет SQL-запрос и достает значение num_mount_points самостоятельно.

Как это будет работать, когда мы добавим атрибут `weight`? Вполне сносно, ведь все изменения в таблице hippos повлияют только на методы класса `Hippo`. Но это повлияет на производительность. Представте себе, что каждый `ArmedHippo::load()` будет вызывать `Hippo::load()`, который подключается к базе данных и выполняет один или несколько SQL-запросов.

Затем уже сам `ArmedHippo::load()` снова подключается к базе данных и выполняет еще некоторые запросы. Если мы добавим `StealthHippo`, то все станет еще хуже. Кеширование и пул соединений помогут сгладить проблему, но полностью ее не искоренят. И опять же `application_utility_tasks()` вызывается из каждого метода отдельно.

Все это иллюстрирует главную проблему ООП. Классы прячут детали выполнения, включая использование ресурсов. Иногда это отрицательно влияет на производительность системы. И пока вы нормально не взглянете на весь код в существующих классах перед их расширением, вы можете не узнать, почему производительность системы внезапно уменьшилась.

Вариант 3: новые таблицы для субклассов

К счастью, здесь есть выход. Допустим, мы вставляем в систему новый класс, называемый `BusinessBase`, и делаем его основным классом для всех бизнес-классов. `BusinessBase` выполняет вспомогательные задачи (права доступа, логи транзакций).

Он также отвечает за каждый SQL-запрос. Ни один из бизнес классов не подключается к базе данных напрямую. Вместо этого они используют структуры данных, чтобы обмениваться информацией с `BusinessBase`.

Вот новый `Hippo`:

```
<?php
class Hippo extends BusinessBase
{
    var $id = null;
    var $name = null;
    //Constructor
    function Hippo()
    {
        $this->table_name = 'hippos';
        $this->addField('id', new DataField(DataField::TYPE_NUMERIC(),
true) );
        $this->addField('name', new DataField(DataField::TYPE_STRING(),
false) );
    }
    //Load record from DB
    function load($id)
    {
        parent::load($id);
        $this->id = $this->getDBValue('id');
        $this->name = $this->getDBValue('name');
    }
    //Save record to DB
    function save() {
    $this->setDBValue('id', $this->id );
    $this->setDBValue('name', $this->name );
    parent::save();
    //Get id number supplied by INSERT
    $this->id = $this->getDBValue('id');
    }
}
?>
```

Конструктор `Hippo` создает структуру данных, которая обменивается информацией с `BusinessBase`. Первым делом устанавливается имя таблицы, в которой `Hippo` будет хранить свои данные.

Затем указанием имени поля и вызовом объекта `DataField` определяется каждое поле в таблице. Объект `DataField` содержит в себе тип данных для поля и флаг, указывающий, определять ли поле как Primary Key.

`Hippo::Load()` отправляет id записи в `BusinessBase::load()`, затем получает значения, которые ему необходимы (в данном случае это id и name). `Hippo::save()` сохраняет значения, которые ему необходимо сохранить, и вызывает `BusinessBase::save()`.

Если `Hippo::save()` создает новую запись, `BusinessBase::save()` сообщит ее id, который `Hippo::save ()` и установит в качестве текущего.

Код для `ArmedHippo` выглядит еще проще:

```
<?php
class ArmedHippo extends Hippo
{
    var $num_mount_points;
    //Constructor
    function ArmedHippo()
    {
        parent::Hippo();
        $this->addField('num_mount_points',
        new DataField(DataField::TYPE_NUMERIC(), true) );
    }
    //Load record from DB
    function load($id)
    {
        parent::load($id);
        $this->num_mount_points = $this->getDBValue('num_mount_points');
    }
    //Save record to DB
    function save()
    {
        $this->setDBValue('num_mount_points', $this->num_mount_points );
        parent::save();
    }
}
?>
```

Конструктор вызовет конструктор `Hippo` и добавит новое поле num_mount_points. `ArmedHippo::load()` вызовет `Hippo::load()` и затем получит значения нужных ему полей. `ArmedHippo::save()` сохранит значение num_mount_points и вызовет `Hippo::save()`.

Теперь мы получили то, что хотели. `ArmedHippo` может быть добавлен без правки `Hippo`. Атрибут `weight` может быть добавлен в `Hippo` без осложнений для `ArmedHippo`. `StealthHippo` может быть легко унаследован от `ArmedHippo` без внесения изменений в существующий код. Плюс к этому, все вызовы `application_utility_tasks()` централизованы в `BusinessBase` и ни разу не появляются во всех других классах.

`BusinessBase` может начинаться примерно так:

```
<?php
class BusinessBase
{
    var $table_name = null;
    var $record = null;
    var $_id_field = null;
}
?>
```

`$table_name` – это конечное имя таблицы для класса. `$record` – это ассоциативный массив, который описывает все записи в таблице. В индексе этого массива находятся названия полей (например, id). Значением каждого элемента этого поля является объект `DataField`.

```
<?php
class DataField
{
    function TYPE_STRING()
    {
        return 1;
    }
    function TYPE_NUMERIC()
    {
        return 2;
    }
    var $db_type = null;
    var $is_primary_key = null;
    var $value = null;
    function DataField($col_type, $is_pk)
    {
        $this->db_type = $col_type;
        $this->is_primary_key = $is_pk;
    }
}
?>
```

Сначала мы фактически определяем две константы, хотя на самом деле это два статических метода, и вызываются они соответственно (см. код нового `Hippo` выше). Декларируя их таким образом, мы определяем для них ту же область видимости (scope) и правила наследования, как и для других методов.

`BusinessBase::addField()` добавляет поле в `$records`.

```
<?php
function addField( $db_col_name, $field )
{
    $this->record[$db_col_name] = $field;
}
?>
```

BusinessBase::load() выглядит примерно так:

```
function load($id)
{
    $this->find_id_field();
    $conn = mysql_connect('localhost', 'hippo_user', 'hippo_user');
    mysql_select_db('hippos');
    $query = 'select * from '.$this->table_name.' where '.$this->
_id_field." = $id";
    $fetched_record = mysql_fetch_assoc( mysql_query($query) );
    foreach ($this->record as $col_name=>$field)
            $this->record[$col_name]->value = $fetched_record[$col_name];
    application_utility_tasks();
}
```

`BusinessBase::load()` вызывает `find_id_field()`, чтобы обнаружить поле с первичным ключем и сохранить его имя в `$this->_id_field` для последующего использования. Затем `BusinessBase::load()` подключается к базе данных, формирует SQL-запрос и выполняет его. Следующие строки заносят данные в массив `$record`, откуда затем могут быть извлечены, например, таким выражением из `Hippo`:

```
<?
$this->id = $this->getDBValue('id');
?>
```

Вывод

Эта статья показывает один из способов работы с ORM. Данный подход имеет следующие положительные стороны:

- Бизнес-классы могут быть изменены без нарушения работы других классов;
- Новые классы могут добавляться без правки уже существующих;
- Методы для работы с БД (например, `Hippo::load()`) несложно писать;
- Вспомогательные задачи приложения (аутентификация, обработка ошибок и т.п.) собраны в одном месте, что упрощает управление ими;
- База данных сохраняет свою естественную структуру и легко интегрируется с генераторами запросов, дата-майнерами и прочим;
- `BusinessBase` класс лежит в основе всего и может быть использован разработчиком в других приложениях.

Вышеприведенный код, конечно же, упрощен для фокусировки на проблематике объектно-реляционного подхода. Здесь не происходит проверки на ошибки, которые могут случиться при соединении с базой данных или при повторении имен полей (например, `ArmedHippo` может попытаться определить имя поля, которое уже было определено из `Hippo`). Не были рассмотрены коллекции классов, например, `HippoHerd` (стадо гиппопотамов – прим. переводчика). Не рассматривались и сложные индексы и возможность использования абстрактных слоев (таких как ADODB, например). Не были зашифрованы параметры обращения к базе данных (имя пользователя, пароль) и многое другое.

Однако, все эти проблемы могут быть решены внутри обрисованной структуры. `BusinessBase::save()` может осуществлять проверку ошибок, записывать логи изменения базы данных, использовать абстрактные слои и прочее. Добавление таких вещей повысит эффективность, масштабируемость и даст возможность повторно использовать написанный код, что упрощает разработку программного обеспечения.

Оригинал статьи находится по адресу:
http://www.phpbuilder.com/columns/mathieson20040309.php3?print_mode=1

Создание пользовательских функций управления сессиями

Автор:
Шон Коутс
Перевод:
Александр Меженков

Одной из наиболее привлекательных особенностей PHP является возможность замещения многих его внутренних подсистем. Для вас, как для разработчика, это означает возможность заставить PHP работать наиболее оптимальным образом с точки зрения задач вашего приложения.

Эта особенность PHP часто позволяет сэкономить время, а значит и деньги, неважно идет ли речь о создании пользовательских функций обработки ошибок, программирования условий завершения работы или написания собственных процедур управления сессиями. Эта статья посвящена последней из перечисленных подсистем, а именно – программированию пользовательских функций управления сессиями.

Что представляют собой пользовательские функции управления сессиями?

Для того, чтобы понять когда и почему разработчику потребуется заместить стандартную подсистему управления сессиями, встроенную в PHP, необходимо прежде всего разобраться, а как эта встроенная подсистема, собственно, работает. Однако прежде, чем переходить к описанию встроенной системы управления сессиями, надо вначале понять и твердо усвоить концепцию сессий.

Веб-программирование является довольно юной отраслью инженерных знаний. Это, однако, не означает, что создание приложений, ориентированных на использование в Интернете, находится в зачаточном состоянии. Многие из общих проблем уже успешно решены, так что мы скорее можем говорить об «отроческом» периоде жизни искусства веб-программирования.

Одной из наиболее болезненных проблем юношеского возраста была невозможность сохранения текущего состояния. (Эта проблема была обусловлена самой природой HTTP протокола, не поддерживающего механизм сохранения состояния между двумя последовательными запросами – прим. перев.)

Устойчивыми приложениями в традиционном смысле называют такие, которые сохраняют информацию о пользователе в промежутке между его действиями. Например, ваша операционная система не требует от вас заново вводить логин и пароль каждый раз, когда вы щелкаете кнопкой или запускаете новое приложение. Веб-протоколы не являются в этом смысле устойчивыми и «забывают» всю информацию о пользователе немедленно после того, как очередной пользовательский запрос обработан.

Для решения этой проблемы отцы-основатели веб программирования создали концепцию веб сессий – механизма, в чем-то напоминающего технологию cookies (ключиков) или GET/POST переменных.

Оба этих механизма позволяют идентифицировать пользователя всякий раз, как он посещает сайт. В результате веб серверы и языки программирования уже давно научились сохранять информацию о любом пользователе в течение сеанса его работы (или сессии).

Для поддержки механизма сессий PHP присваивает уникальный идентификатор сессии каждому новому посетителю сайта/веб-сервера/PHP-приложения. Этот идентификатор однозначно связан с хранящимся на сервере набором данных, специфическим для текущего посетителя. По умолчанию, PHP хранит информацию о сессии в виде специальным образом сериализованных массивов в файлах (по одному на сессию) на сервере в директории для хранения временных файлов (см. пример ниже). Если PHP у вас установлен на диске C:, то вы можете увидеть эти файлы на своем компьютере в директории C:\PHP\sessiondata Имена фалов имеют вид sess_4daa75c46854a4a112e031ab81d7d832, где все, что находится после «sess_» и есть уникальный идентификатор сессии.

```
[root@iconoclast tmp]# ls sess_*
sess_317d2281d389f81acf98b3fbf5500b67
sess_b78888da60ac711522c3aa7c918ab700
sess_408c12fda146e4da82cc0134266a0fbd
sess_bc08a8e3a1a04c3167d1a06ee8ecbf86
sess_522ffc61e642470224296ee9ad6f8fea
sess_c89f1ba055be48e01dcb7002f6b6c08f
sess_aa8b0c24b0e3cd7f358230291b228410

[root@iconoclast tmp]# cat sess_c89f1ba055be48e01dcb7002f6b6c08f
colours|a:4:
{s:11:"scheme_name";s:5:"blues";s:12:"colour_light";s:6:"99CCFF";s:10:"col
our_med";s:6:"6699FF";s:11:"colour_dark";s:6:"0033FF
";}messaging|a:4:
{s:6:"on_off";s:2:"on";s:6:"cookie";b:1;s:5:"popup";b:1;s:5:"prefs";b:0;}
messaging_nick|s:0:"";
```

PHP скрипт может получить доступ к переменным сессии двумя способами: во-первых, используя функцию `session_register()` (и дополнительно глобальные переменные) и во-вторых – используя внутреннюю переменную `$_SESSION`, имеющую superglobal область видимости (т.е. она доступна во всех скриптах и функциях). Мы рассмотрим только второй метод, поскольку именно он официально рекомендован разработчиками, а кроме того, он гораздо мощнее и проще в использовании.

Для чего использовать пользовательские методы управления сессиями

Существуют много причин, почему разработчик может захотеть заместить принятые по умолчанию методы управления сессиями. Рассмотрим наиболее часто встречающиеся.

- Для размещения своего сайта вы используете разделенный сервер провайдера и обеспокоены проблемой безопасности

Почти каждый PHP разработчик рано или поздно сталкивается с необходимостью работать в условиях разделенной с другими пользователями среды разработки.

Мы никогда не задумываемся о подобных вещах, пока не вынудит нужда. Кроме невозможности диктовать провайдеру требуемую конфигурацию, разработчику, как правило, приходится мириться с рядом других неудобств, не последним из которых является проблема безопасности, особенно когда вам приходится делить среду разработки с потенциально опасными соседями.

По умолчанию, виртуальные сайты (при виртуальном хостинге сайты разных клиентов провайдер держит на одном физическом сервере) хранят переменные сессии в одном общем для всех сайтов месте. Мало того, что ваши данные хранятся там, где их кто угодно может обнаружить случайно или по злому умыслу. Данные сессий также доступны вашим общим пользователям (например, к ним имеет доступ userid, с которым в данный момент работает демон веб сервера). Разработчик-злоумышленник, имеющий доступ к одному из виртуальных сайтов, совсем не обязательно к вашему, может легко получить доступ к чужим переменным сессий. Существуют различные способы избежать подобных осложнений. Например, использовать «safe mode», однако это ведет к другим проблемам, описание которых, впрочем, выходит за рамки данной статьи.

Наиболее надежным способом решения описанной проблемы безопасного хранения сессионных данных является использование собственных процедур управления сессиями. При этом мы, в частности, получаем возможность перехватывать управление записью переменных сессии и можем кодировать их до того, как они будут записаны на диск. Немного позже мы коснемся деталей этого метода.

- Серверы с симметричной нагрузкой (Кластеризация)

  Противоположной ситуацией по сравнению с разделенной средой (много сайтов на одном сервере) является кластерная среда (много серверов для одного сайта). Кластеризация позволяет системным администраторам распределить нагрузку сайтов с высокой нагрузкой между различными физическими серверами (узлами). Часто эти физически различные серверы используют общую базу данных. Это снижает нагрузку на каждый отдельно взятый узел и дает дополнительную избыточность, что позволяет в правильно сконфигурированных (в смысле распределения нагрузки) системах отключать от кластеров отдельные сервера в случае их сбоев или некорректной работы.

  В кластерных системах сессии обычно хранятся на локальных дисках каждого узла. При этом совместное использование сессионных данных различными узлами становится затруднительным. Эта проблема легко решается если данные сессии (как и в предыдущем случае) перехватываются и направляются на общий сервер баз данных, вместо того, чтобы записывать их на локальный диск каждого сервера.

Надо заметить, что многие системы распределения нагрузки имеют «липкий» режим, который позволяет «приклеивать» пользователей к одному узлу, (в этом случае данные сессии текущего пользователя гарантированно будут сохранены на локальном диске сервера) однако этот режим противоречит самой цели выравнивания нагрузки.

- Вам необходимо хранить данные сессий на другом носителе.

  Кластеризация - не единственная ситуация, при которой хранение данных сессий на альтернативном носителе может оказаться полезным.

  Возможно, например, что у вас имеется неограниченный резерв оперативной памяти (при нынешних ценах на память это не такая уж и фантастика) и вы размышляете, на какое бы полезное дело ее употребить. Если вы решите хранить ваши данные в памяти, это существенно снизит нагрузку на диски и значительно ускорит процесс чтения/записи.

  А вот другой случай, когда вы можете захотеть изменить стандартное место хранения сессионных данных. Например, некоторому серверному сервису или процессу, никак не связанному с вашим приложением, требуется обеспечить доступ к вашим сессионным данным. В этом случае вы можете отключить стандартные процедуры записи данных сессии и перенаправить переменные сессии этому внешнему сервису, используя один из веб-протоколов, вроде XML-RPC или SOAP, или напрямую через механизм сокетов.

Как PHP работает с пользовательскими обработчиками сессий

Пользовательский механизм упраления сессиями строится вокруг PHP функции `session_set_save_handler()`. Эта функция позволяет разработчикам заместить механизмы, используемые по умолчанию. Механизмы, которые могут быть замещены на пользовательские, это доступ к хранилищу сессионных данных (open), чтения данных сессии (read), записи (write), завершения сессии (close), уничтожения сессии (destroy) и, наконец, сборщик мусора (garbage collector), этот механизм определяет условия уничтожения данных сессии с диска в случае, если сессия не была явным образом разрушена с помощью механизма destroy.

Руководство PHP содержит подробное описание замещающих функций (см. http://www.php.net/manual/en/function.session-set-save-handler.php). Если вы решили написать свой механизм управления сессиями, то вы должны заместить все функции. Тоесть нельзя, например, применить собственную функцию для механизма разрушения сессии без того, чтобы переписать и все остальные методы: open, read, write и т.д.

PHP вызывает метод open всякий раз, когда возникает событие «session start». Это событие возникает, либо в момент вызова в вашем сценарии функции `session_start()`, либо автоматически при каждом запросе. Автоматическое чтение переменных сессии при каждом запросе обеспечивается с помощью директивы `session.auto_start` файла php.ini (см. http://www.php.net/manual/en/ref.session.php#ini.session.auto-start). По умолчанию эта директива отключена.

Назначение метода open, это подготовка сессии к чтению. Например, если вы используете базу данных как хранилище для переменных сессии, то метод open является идеальным местом для подключения к серверу баз данных и выбору нужной базы. Вызывая метод open, PHP передает ему два параметра: `$save_path` и `$session_name`. Значения этим параметрам PHP назначает самостоятельно, впрочем, вы сами можете задать их явным образом с помощью функций `session_save_path()` и `session_name()` соответственно.

Метод read также вызывается всякий раз, когда возникает событие «session start». Назначение этого метода – загрузить все переменные сессии в массив `$_SESSION`, имеющий superglobal область видимости. Причина, по которой метод read вызывается только один раз в момент поступления запроса не так очевидна, как может показаться на первый взгляд.

Разумно было бы предположить, что метод read должен должен вызываться всякий раз, как происходит обращение к массиву `$_SESSION`, однако подобный подход был бы слишком ресурсоемким. Вместо этого, функция чтения вызывается один раз за время работы сценария. При этом PHP предполагает, что она читает сразу все переменные сессии (именно так поступает встроенный механизм чтения переменных сессии).

Функция чтения имеет только один параметр `$id`, через который PHP передает уникальный идентификатор той сессии, данные которой следует прочесть. Этот параметр может быть изменен с помощью функций `session_id()` и `session_regenerate_id()`.

Параметр `$id` следует использовать для однозначной привязки переменных сессии к конкретной сессии конкретного браузера (например, он может быть использован как ключ базы данных). Вам также может потребоваться вернуть из функции прочитанные данные. Данные возвращаются в специальным образом «сериализованном» формате, который мы рассмотрим подробно чуть позже.

Последним действием в процессе обработки события «session start» является возможный вызов функции сборщика мусора или gc метода. Мы говорим «возможный вызов», так как вероятность вызова gc определяется в конфигурационном файле php.ini директивой `session.gc_probability`, которая также может быть изменена с помощью функции `ini_set()`.

Сборка мусора осуществляется для очистки «мертвых» сессий, которые пережили максимальный срок жизни, отпущенный сессиям, например, в результате того, что пользователь покинул сайт. Назначение сборщика мусора в том, чтобы освободить не используемые более системные ресурсы. В результате общий уровень беспорядка в системе уменьшается и не влияет на производительность. Например, как известно PHP хранит данные сессий в файлах и когда сохраненные сессии более не актуальны, сборщик мусора уничтожает ассоциированные с ними файлы.

Если сборщик мусора вызывается, то его вызов происходит сразу после завершения метода read. Метод gc также принимает только один параметр `$maxlife-time`, значение которого определяется директивой `session.gc_maxlifetime` в файле php.ini.

Значение этой директивы используется для определения, какие сессии следует рассматривать как «мусор», основываясь на их возрасте, измеренном в секундах. Важное уточнение: термин «"возраст» означает не общее время существования сессии, а время, прошедшее с момента последнего к ней обращения.

Метод destroy вызывается только явным образом из вашего скрипта. Его назначение – обеспечить условие перезапуска сессии или уничтожение всех переменных сессии. После вызова метода destroy бессмысленно пытаться записывать какие бы то ни было данные в массив `$_SESSION`, так как методы write и close уже не будут вызваны в конце работы скрипта. Метод destroy вызывается с параметром `$id`, который имеет такое же значение, как и в методе read.

Оставшиеся методы «write» и «close» вызываются автоматически при выключении или при завершении работы скрипта. Метод write сохраняет данные сессии на носителе, определенном вами, как месте хранения сессионной информации. Этот метод в паре с методом read позволяет сохранять данные на различных носителях, либо вызывать срабатывания другого события, которое в свою очередь сохраняет данные или вызывает очередное событие и так далее.

Метод write вызывается интерпретатором PHP с двумя параметрами: `$id` и `$sess_data`. Параметр `$sess_data` передается в специальным образом сериализованном формате и может быть записан на выбранный вами носитель непосредственно в этом формате.

Последний метод «close» вызывается после того, как все остальные методы, имеющие отношение к механизму сессий, завершили работу. Его назначение – освободить любые ресурсы, которые могли быть использованы методами «read» (например, соединение с базой данных или область памяти). Метод close не имеет никаких параметров.

Код из Листинга 1 (в приложенном к журналу файле listing1.phps) показывает пример того, как и когда вызываются методы управления сессиями. Вы можете абсолютно безопасно запускать его, поскольку указанные там методы, замещающие стандартные функции управления сессией, всего лишь заглушки, приведенные здесь исключительно с иллюстративной целью.

Тем не менее, обратите, внимание на то, что сессии не будут корректно работать, без замены этих заглушек на реально работающие read/write функции. Также обратите внимание на то, что если вы хотите увидеть в работе метод destroy, вам следует убрать комментарии с соответствующих строк.

Следующий ниже код показывает результат работы кода Листинга 1 (из приложенного к журналу файла listing1.phps).

```
'open' method called
'read' method called
'gc' method called
Called session_start()
Registering 'apples'
Registering 'oranges'
Dumping session:
Array
(
[apples] => 3
[oranges] => 8
)
'write' method called
'close' method called
```

Вернемся теперь к «специальному сериализированному формату», о котором мы говорили выше. Этот формат похож на результат работы функции `serialize()`, хотя и есть некоторые отличия, которые, к сожалению, в своем большинстве остаются недокументированными. Для работы с этим форматом представления данных как правило используются PHP функции `session_encode ()` и `session_decode()`, хотя вы можете подставить и свои собственные процедуры, изменив директиву `session.serialize_handler` в файле php.ini так, чтобы она указывала на ваши функции.

Теперь, когда вы увидели каким образом можно изменять стандартные процедуры управления сессиями, вы, возможно, захотите узнать, возможно ли упаковать собственные функции в компактный объектно-ориентированный пакет. Вы будете удивлены, насколько это просто. Основная идея заключается в том, чтобы упаковать свои функции в класс как статические методы и обеспечить замещение стандартных методов, передав свои собственные методы в параметрах функции `session_set_save_handler()` как массивы с двумя элементами, первый их которых есть имя класса, а второй – имя соответствующего метода в следующем порядке: open, close, read, write, destroy, gc.

Листинг 2 (приложенный к журналу файл listing2.phps) показывает как это сделать. По сути, Листинг 2 есть объектно-ориентированная версия Листинга 1.

Подобный способ упаковки методов в класс уменьшает риск возможного конфликта имен функций с именами в других скриптах того же приложения, а также позволяет легко переносить созданный класс в другие приложения или передавать его другим разработчикам.

Практическое использование примеров.

Как уже неоднократно упоминалось в процессе изложения, одной из наиболее веских причин замещения стандартных функций управления сессиями является изменение места хранения данных сессии. В качестве примера рассмотрим использование базы данных как хранилища сессионных данных.

Чтобы не усложнять изложения, будем считать, что у нас в качестве системы управления базами данных используется MySQL. Мы также предполагаем, что у вас есть некоторый практический опыт работы с базами данных и вам знакомы функции PHP для доступа к ним.

Итак, во-первых, нам потребуется место для хранения наших сессий. Создадим для этого таблицу в базе данных. Поле session_data таблицы sessions как раз и будет тем самым местом, где собственно и хранятся данные.

```
CREATE TABLE `sessions` (
  `session_id`   INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `create_ts`    DATETIME DEFAULT 'now()' NOT NULL,
  `valid`        ENUM('yes','no') DEFAULT 'yes' NOT NULL,
  `session_key`  VARCHAR(255) NOT NULL ,
  `session_data` TEXT NOT NULL
) COMMENT = 'Session Data Storage';
```

Проверим:

```
mysql> describe sessions;
+------------+----------------+-----+-----+-------------------+--------------+
| Field      | Type           | Null| Key | Default           | Extra        |
+------------+----------------+-----+-----+-------------------+--------------+
|session_id  |int(11)         |     | PRI |NULL               |auto_increment|
|create_ts   |datetime        |     |     |0000-00-00         |              |
|            |                |     |     |00:00:00           |              |
|valid       |enum('yes','no')|     |     |yes                |              |
|session_key |varchar(255)    |     |     |                   |              |
|session_data|text            |     |     |                   |              |
+------------+----------------+-----+-----+-------------------+--------------+
5 rows in set (0.00 sec)
```

Во-вторых, нам необходимо создать наши собственные функции управления сессиями. Листинг 3 (приложенный к журналу файл listing3.phps) показывает как это сделать. Вам необходимо включить этот код ДО вызова функции `session_start()`. Запустив приведенный ниже короткий скрипт, вы увидете, как данные ваших сессий «волшебным» образом будут сохраняться в MySQL базе данных.

Попробуйте проделать это на вашем веб сервере (убедитесь при этом, что база данных модифицирована, как было описано выше, то-есть в ней была создана таблица sessions, кроме того, вам, возможно, нужно будет изменить константы, определенные в начале Листинга 3, так, чтобы они соответствовали вашей реальной конфигурации).

```
<?php
        require_once('listing3.phps');
        // Или имя файла, в котором вы сохранили Листинг 3
        session_start()
        $_SESSION['cow'] = 'moo';
?>
```

Откройте в вашем браузере страничку, созданную таким образом, а затем проверьте базу данных. Вы должны увидеть новую запись для сессии, которую вы только что открыли, обратившись к созданной страничке. Если вы теперь поэкспериментируете в коде странички с различными процедурами, выполняющими чтение переменных сессии или уничтожение всех переменных с помощью `session_destroy()` и время от времени будете проверять базу данных, вы увидите как меняются данные по мере того, как вы изменяете код скрипта.

Обратите внимание, что вам необходимо явно вызывать функцию `session_start()` с тем, чтобы ваши функции управления сессиями работали, в противном случае (без явного вызова `session_start()`) PHP будет автоматически использовать встроенные методы управления сессиями.

Не правда ли, все оказалось не так уж и сложно. Особая прелесть заключается в том, что теперь вы можете без труда модифицировать все ваши старые программы, работающие с сессиями. Вам нужно всего лишь инициализировать свой собственный механизм управления сессиями, не трогая основной код программ.

Дополнительные преимущества хранения переменных сессий в базе данных

Теперь, когда данные ваших сессий хранятся в базе данных, перед вами открываются дополнительные возможности. Например, хотелось бы вам узнать сколько посетителей в данный момент активны на вашем сайте? Запрос, приведенный ниже, даст вам число активных сессий:

```
SELECT
  count(*)
FROM
  sessions
WHERE
  valid = 'yes'
```

Если вы уменьшите максимальное время жизни сессии в состоянии ожидания, результат будет более точным, впрочем, за это придется расплачиваться слишком быстрым окончанием сессий для задумчивых посетителей. Поэкспериментируйте с директивой `session.gc_maxlifetime` в php.ini для определения оптимального значения для ваших задач.

Другим преимуществом использования собственных функций управления сессиями является возможность применения неразрушительных методов сборки мусора. Если вы внимательно посмотрите на Листинг 3, то заметите, что gc метод не удаляет данные из таблицы sessions. Вместо этого, он меняет значение поля valid на "no". Подобный подход сохраняет простейшую историю всех сессий, хотя рано или поздно, вам все равно захочется удалить неактивные сессии, которые имеют время, прошедшее с момента последнего к ним доступа (поле create_ts) больше некоторого значения. Это нормально, хотя бы для того, чтобы держать размеры таблицы в разумных пределах.

Если вы решите использовать функции управления сессиями из Листинга 3, вам следует позаботиться о безопасности данных. Поскольку в переменных сессии может храниться самая различная информация (например номера кредитных карточек), всегда надо уделять особое внимание проблеме их безопасности.

Возможно, вы захотите несколько модифицировать процедуру сборки мусора gc так, чтобы удалять часть информации из колонки session_data в тот момент, когда значение поля valid меняется на «no» или, возможно, вместо удаления, шифровать всю сессионную информацию, чтобы скрыть ее от любопытных глаз. Используя библиотеку mcrypt (http://www.php.net/mcrypt) или любой другой метод шифрования, вы можете модифицировать методы read и write для осуществления любого уровня шифрования переменных сессий перед их записью. Естественно, что после чтения зашифрованных данных, вам надо позаботиться об их декодировании.

Заключение

Теперь вы обладаете достаточными знаниями для разработки своих собственных методов управления сессиями, будь то с целью решения безопасного их хранения или для совместного использования этих данных с внешними приложениями. Код, приведенный в этой статье, никоим образом не претендует на удовлетворение всех ваших требований по хранению сессионных данных, но безусловно может служить хорошей платформой для претворения в жизнь ваших собственных идей для решения возникающих проблем.

Об авторе

Шон Коутс является членом PHP группы и профессиональным разработчиком приложений на PHP, работающим на компанию, оказывающую финансовые услуги. Он использует PHP еще со времен 3.x версий и при каждом удобном случае пропагандирует его использование. Шон живет в Монреале и с ним можно связаться через его веб-сайт http://sean.caedmon.net или по электронной почте sean@php.net

Люди

Интервью с основателем phpclasses.org – Мануелем Лемосом

Интервью брал:
nw

Phpclasses.org — это репозиторий php-классов. Сайт представляет экспериментальный сервис, суть которого заключается в свободном распространении классов, написанных на PHP. Целью данного сервиса является создание базы программных компонентов, готовых для использования при разработке PHP-приложений. Любой разработчик может добавить в базу свой класс или поискать готовое решение от других разработчиков. На сегодняшний день, в коллекции phpclasses.org уже более 1350 классов, и коллекция постоянно растет. Любой желающий может также подписаться на рассылку и получать уведомления о выходе новых классов по электронной почте. Помимо коллекции классов, на сайте можно также найти информацию о конкурсе «PHP Programming Innovation Award» и группах пользователей по всему миру. Все это – последние инициативы phpclasses.org. В середине апреля сего года Мануель Лемос, основатель и главный разработчик сайта, любезно согласился дать интервью для нашего журнала PHP Inside.

nw: В чем заключается главная задача существования phpclasses.org?

ML: Изначально я планировал размещать на сайте только свои классы, чтобы другие разработчики могли их скачать, попробовать в действии и, возможно, сообщить мне о допущенных ошибках или предложить новые идеи.

Позднее я решил предоставить место на сайте и для других авторов, чтобы любой желающий мог показать свою работу PHP-сообществу и получить полезные отзывы.

nw: Когда сайт был основан и кто его создавал?

ML: Проект стартовал в середине 1999 года. Тогда я помогал пользователям некоторых листов рассылок о PHP, и моя помощь часто заключалась в том, что я делился своими наработками. Я предлагал написать мне письмо с запросом моих классов всем тем, кто был в этом заинтересован.

Однажды один австралийский разработчик остался настолько довольным моим классом для генерации и проверки форм (а это был один из моих самых популярных классов), что в списке рассылки публично поблагодарил меня и всем порекомендовал использовать именно мое решение. После этого мне стали приходить десятки писем с просьбой прислать мой класс.

Сначала я записывал адреса всех желающих и по выходу нового релиза рассылал им новые версии. Но со временем запросов становилось все больше и больше, и ручная рассылка превратилась в тяжелый труд. Тогда я начал подыскивать сайты, которые могли бы предоставить мне возможность организовать подписку и рассылку сообщений о выходе обновлений.

Мне нравился сайт PX, Девида Склара (David Sklar), который позволял пользователям обмениваться PHP-кодом. Однако он не предоставлял сервисов подписки и рассылки сообщений, которые были мне столь необходимы.

Я поделился с Девидом своими идеями и даже предложил собственноручно воплотить их в жизнь, но Девида мое предложение не заинтересовало, и тогда я решил создать собственный сайт. Этот сайт должен был объединить в себе хорошие черты от PX и мои собственные идеи. Основа для него была разработана буквально через неделю.

Многие говорили, что перешли на объектно-ориентированный подход благодаря моему сайту, так как ранее не имели серьезных навыков программирования.

nw: Сколько человек в настоящее время ведут сайт? Откуда они? Сколько им лет? Расскажите пожалуйста о себе.

ML: В основном, я делаю все сам. Мне 35 лет, и с 2001 года я все свое время посвящаю работе над сайтом.

nw: Как много классов в вашей коллекции? Можете ли вы выделить «жемчужины»?

ML: В данный момент около 1350 утвержденных классов. За месяц на сайте появляется примерно 50 новых. Мне сложно выделить из них самые лучшие, но некоторые я нахожу наиболее интересными. Это:

Класс: Generalized CachedTemplate class.
Автор: Jesus Castagnetto.
Описание: абстрактный слой для использования нескольких классов шаблонов (template classes) с одним API.
Ссылка: http://www.phpclasses.org/cachedtemplate/

Класс: anyDB.
Автор: Lennart Groetzbach
Описание: абстрактный слой, имеющий единый API для использования нескольких абстрактных слоев по работе с БД.
Ссылка: http://www.phpclasses.org/anydb/

nw: Откуда географически основная часть ваших пользователей? Кто они?

ML: Наши пользователи — разработчики, которым нужны наши пакеты для решения своих проблем. Некоторые из них (но не все), утверждают, что PHP предоставляет достаточные для них объектно-ориентированные средства. Многие говорили, что перешли на объектно-ориентированный подход благодаря моему сайту, так как ранее не имели серьезных навыков программирования.

Примерно 60% пользователей сайта находятся в Европе, 20% в Северной Америке, около 12% из Азии и Океании, 8% из Южной Америки и менее 1% из остальных уголков планеты.

nw: В PHP5 объектная модель была переписана. Повлияет ли этот факт на вашу коллекцию?

ML: Не думаю. Большинство компилируемого кода из PHP4 будет работать и под PHP5. На данный момент у нас есть только один класс, который использует новое встроенное в PHP5 расширение SOAP. Этот класс предоставляет интерфейс для поиска в Google: http://www.phpclasses.org/googleclient — Google client.

nw: Что вы думаете о новой объектной модели PHP5?

ML: Я считаю PHP уже достаточно сформировавшимся языком. Объектная модель PHP5 стала ответом на пожелания тех, кто хотел видеть в PHP некоторые черты от Java. Эти люди теперь должны быть счастливее.

Большинство новых возможностей не являются ключевыми, кроме, пожалуй, обработки исключений. Это конечно не относится напрямую к объектной модели, но действительно поможет отделить код обработки ошибок от остальной логики.

Не могу понять реальной причины, почему пространство имен было исключено из последней стабильной реализации (PHP 4). Это помогло бы интегрировать классы из разных источников в одном приложении и при этом избежать коллизий. Это также позволило бы связанным классам получать доступ к переменным и функциям друг друга, в то время, как для других классов их методы оставались бы private.

Пространство имен — это возможность, которая позволяет более правильно организовывать классы. До PHP5 мы были вынуждены искать обходные пути, и, я думаю, никто не будет скучать по ним.

nw: И наконец, что вы можете пожелать нашему журналу и сообществу?

ML: Как и любому другому сообществу на земле, я желаю вам собираться вместе и помогать друг другу, так как взаимодействие — это одна из самых сильных сторон такого сообщества, как у PHP. Тем более, за плечами этого языка не стоит большой коммерческой организации, которая продвигала бы PHP на рынок, как надежное средство для разработки веб-приложений. По этой причине PHP всегда испытывает трудности при проникновении на корпоративные рынки.

Хорошо организованные сообщества помогают преодолеть эти трудности, организуя регулярные (ежемесячные) встречи, где люди могут лично получить помощь в вопросах разработки, организовать локальные конференции, сформировать партнерские отношения и т.д.

Сайт phpclasses.org проявил инициативу в этом вопросе и реализовал сервис, который позволяет локальному сообществу рассказать о своем существовании другим пользователям сайта из того же региона.

Если в той местности, где вы живете еще нет группы пользователей, вы можете сформировать новую. Проще всего пойти на Yahoo groups, начать там новую дискуссию и пригласить для участия разработчиков из своего региона.

Если вы уже состоите в локальной группе пользователей PHP, то сообщите об этом на phpclasses.org и доверьте нашему сайту продвигать ваше сообщество. Эта инициатива ни в коем случае не позиционируется, как конкурент для других сайтов со списками групп пользователей. Если вы знаете другие подобные сайты, сообщите о своем сообществе и на них.

Чтобы найти группу пользователей в своем регионе или сообщить о таковой, посетите

http://www.phpclasses.org/browse/group/

Если вам небезразлична судьба PHP, то все в ваших руках! Удачи!

Оригинал интервью можно найти по адресу: http://phpinside.net/docs/mlemos.htm

Кто-то должен быть первым...

Интервью брала:
Елена Тесля [Lenka]

Не так давно, в мае 2004 года, сообществу PHPClub исполнилось 5 лет. И мы решили немного покопаться в его истории, пообщавшись с людьми, которые сыграли значительную роль в развитии Клуба. Кто-то ушел, кто-то остался, кто-то доволен тем, что происходит сейчас в Клубе, кто-то – нет. В любом случае каждый почерпнул из этого сообщества что-то полезное для себя. В этом выпуске мы берем интервью у Дмитрия Лебедева [Detail], который создал и долгое время вел раздел «Детали» (http://detail.phpclub.net), и у Антона Калмыкова [Antonio], старожила PHPClub.

Интервью с Дмитрием Лебедевым [Detail]

Lenka: Дима, когда ты впервые узнал о Клубе, и что он тогда из себя представлял?

Detail: Это было в мае 2000 года. Клуб был в нынешней форме — форум и архив с файлами и статьями: несколько наших, например, от Scarab, и переводные. Разумеется, не было «деталей».

L: А как появились сами «детали»?

D: Начать надо с 1999 года, когда я понял, что руками и на JavaScript нельзя поддерживать сайт с кучей записей. Начал изучать. Не было ничего вообще. Английский я учил с 89 года, но в документации была специфическая незнакомая терминология, осваивал не сразу. Из русского была документация на webclub.ru. Книг не было никаких, в компьютерной конторе, где был книжный отдел, я спросил у менеджеров по продажам - они не знали ни про такие книги, ни про PHP.

К тому времени я умел программировать на бейсике, чуть-чуть на паскале и совсем мало на C++. Не понимал ничего в принципах синтаксиса, контрольных структурах, операторах. Методом научного тыка и чтением документации освоил базовые функции, потом поставил дома Apache и MySQL, освоился с ними.

Больше всего помогло чтение английской документации. Года три безуспешно пытался понять из умных книжек по программированию, что такое ООП и как оно делается. Так ничего не понял, пока не открыл такой раздел в английском мануале по PHP.

В итоге к осени 2000 года я решил, что материалов в сети очень мало, и я могу начать писать материалы для начинающих по PHP-программированию. Со Смирновым Сашей (Саша Смирнов [admin, PHPClub], собственно, и был родоначальником PHPClub — прим. автора) мы общались по ICQ, и была прямая дорога к нему. Я сделал несколько пилотных материалов, показал ему сайт, который работал на моей домашней машине (через dial-up). Он поддержал идею и помог.

L: Т.е. по сути в этом плане ты самоучка? Кстати, ты ведь экономист?

D: Основы программирования узнал в школьные годы. Остальное – сам. Да, экономист.

L: Ты только учишься или еще и работаешь?

D: Только учусь.

L: Когда закончишь, будешь экономистом или все же программистом?

D: Думаю, экономистом или журналистом.

L: А почему в программирование не тянет?

D: Ну, я вот ещё увлекаюсь волейболом, много в нём знаю, автогонки смотрел с 1992 года. Это же не значит, что я должен трудоустраиваться в спорте. Хотя я искренне хочу проехать к 30 годам Париж-Дакар на мотоцикле.

Нужно делать закрытую, элитарную, если хочешь, компанию, клуб в смысле круга людей, а не танцпола с попсой драками.

L: Когда и кого из Клуба ты увидел вживую впервые? Расскажи про пати.

D: Эти мероприятия начались 22.12.2000 в Миллениум-пати. Мы даже отправили тост московским друзьям, которые только начинали, когда мы расходились.

Люди были разные, из постоянных местных – Grizly. Потом появились Scarab, AnToXa и Nail. Самую главную и весёлую – 21 мая 2001 – я пропустил, потому что был на московской. Закончились тусовки 26 января 2002 как-то странно. Потом, кроме нашего ядра (я, Grizly, AnToXa и Nail), никого больше собрать не удавалось (Scarab уехал в Москву, а потом и AnToXa).

Последняя встреча, уже скорее наш междусобойчик, был в январе прошлого года. А потом мне совсем надоело пить пиво и ходить в боулинг/бильярд/бар как таковые.

L: Сколько народу из Клуба собиралось в НСК?

D: Максимум было до 10, по-моему. Разных людей всего было человек 20-25.

L: Как нашел PHPClub? Интересно, кто тогда уже был в Клубе из Team или просто из старожилов?

D: Ой, не помню. 4 года прошло всё-таки. RomikChef, потом позже появился tony2001.

L: Тебе не очень нравится, что происходит сейчас в форуме?

D: Я давно не общался в форуме помногу, поэтому вообще-то мне всё равно. Могу ошибаться, но представляю обстановку. Она мне, конечно, не нравилась бы, будь я там чаще, хотя лучше или хуже в таком варианте сделать не получится. Чтобы не шёл поток дураков и халявщиков, нужно делать закрытую, элитарную, если хочешь, компанию, клуб в смысле круга людей, а не танцпола с попсой драками. :)

L: Вернемся к деталям. Когда ты решил отказаться от их ведения, как я поняла, ты уже не верил в то, что они нужны? Или почему пришло решение оставить этот проект?

D: Решил, что не нужны, появилась куча эпигонов, статьи перепечатывали, и, наконец, такой формат рассчитан на халявщиков. И интерес как-то пропал, я ведь всё время оставался любителем, а не профессиональным программистом.

L: А почему ты сомневаешься в написанном тобою?

D: Часто появлялись люди, которые знали вопрос чуть лучше.

L: Но они ничего не писали при этом.. По-моему, уже то, что ты начал весь этот проект - было нужным делом. Особенно, учитывая, что многим людям это помогло...

D: Может быть, просто я был излишне честным и требовательным к себе.

Не так давно в форуме Дмитрий написал:

«Последнюю свою статью в «деталях» я выложил под виртуалом «Jackie Treehorn» . Поругался под его именем с tony2001 и понял, что проект и я сам деградирую, и что если убрать имя и имеющиеся в виду некоторые былые заслуги, то материал очень спорный получается.

Удивительно, как Young вдохнул в проект новую жизнь, и рядом заработал журнал. Я думал, что подобное уже никому не нужно. Наверное, просто кто-то должен был быть первым. Следующие сделают больше. Они - профессионалы, а первопроходцы-любители уходят в сторону».

L: А что ты думаешь о том, что будет с PHPClub? Он разовьется во что-то большее, останется тем, что есть сейчас, или вообще, может быть, уйдет в историю?

D: По-моему, новосибирская тусовка развалилась из-за малочисленности и разных интересов — кто-то спортсмен, кто-то алкоголик. У вас в московско-украинской компании собрались люди, близкие по интересам, так что всё будет в порядке.

Интервью с Антоном Калмыковым [Antonio]

Lenka: Antonio, когда ты попал в Клуб? И как ты его обнаружил?

Antonio: Первый раз я попал на сайт клуба (тогда еще phpclub.unet.ru) в конце 1998 года, когда искал удобоваримый скриптовый язык для разработки веб-проектов. В то время я использовал ASP, но этот язык не удовлетворял меня своей функциональностью и гибкостью. URL мне дал мой сисадмин, который сначала пытался по моему заказу написать что-то подобное PHP, но поленился и начал «рыть» интернет на предмет существующих инструментов.

L: А что тогда представлял из себя Клуб?

A: Тогда клуб представлялся в единственном лице Саши Смирнова. (Может, я и ошибаюсь, но много воды утекло с тех пор). Форумы клуба только начинали наполняться посетителями. Вопросы на форумах были совсем тривиальными (с нынешней точки зрения). Да, клуб сильно изменился с тех пор.

Вместе с ним выросло много профессиональных веб-разработчиков, образовалось Russian PHP community. Нынешний уровень новичков выше, чем был в те времена.

L: Тогда вряд ли можно было сказать, что все это выльется в такую громадную структуру (ну вот уже конференции проводятся, пати многочисленные), как сейчас? Или можно было догадываться, что рано или поздно это произойдет?

A: Я тогда об этом даже не задумывался. PHP представлялся идеальным средством веб-разработки и был интересен сам по себе как новый инструмент, тем более, что PHP в то время только начинал свое развитие, поэтому было желание освоить PHP и помочь в этом другим, поскольку «не хочется проходить мимо хорошей вещи, не поучаствовав в ее улучшении».

Было желание освоить PHP и помочь в этом другим, поскольку «не хочется проходить мимо хорошей вещи, не поучаствовав в ее улучшении»

L: Как ты присоединился к Клубу?

A: PHP в то время содержал большое число ошибок, в устранении которых я решил поучаствовать. Для этого поставил себе цель — собрать PHP под Windows и начать отладку. К тому времени в стандартном дистрибутиве PHP для Windows набор модулей был не полон, т.е. не все модули, которые должны были работать под Windows, работали и вообще присутствовали в собранном виде. Эксперимент удался, сборка была произведена, после этого я написал короткую инструкцию по сборке PHP под NT и отправил ее Саше для размещения на клубе. С этого и началось наше знакомство.

L: В дальнейшем ты писал что-то для Клуба?

A: Нет, я не помогал разрабатывать движок клуба. Я выступил в качестве организатора 1-й пати и нескольких последующих, активно участвовал в форумной жизни клуба.

L: А с кем тогда познакомился вживую? И кто на тот момент был активным участником Клуба? Может, уже кого-то сейчас и нет в Клубе?

A: Удаленно активно общались с Димой Бородиным, после первой пати появился более широкий круг общения. Обычно собирались с PHPClub, Haba-Haba, Voodoo, KDK в неформальной обстановке. Vodoo сейчас в Германии.

L: Получается, насовсем из Клуба никто за это время не ушел?

A: Да. Клуб рос и растет.

L: Антонио, а еще такой вопрос: где ты работаешь? Если, конечно, это не секрет.

A: Сейчас работаю начальником отдела тестирования приложений в компании 3WGraphics.

L: А работа с PHP связана?

A: Да, PHP используется для создания тестов. Конкретное применение - тестирование логики протоколов (имеются в виду внутренние протоколы для обмена данными между приложениями, которые были разработаны программистами компании), функциональное тестирование серверных приложений, и.т.д.

L: Чем занимаешься сейчас в Клубе?

A: Я заявил о своем выходе из Team 26 сентября 2003 года. Причины описаны в посте на форуме:

Привет всем, дорогие коллеги.

Поводом для написания этого письма/сообщения/очерка послужила мысль о том, что уходить, не попрощавшись, некрасиво, да и не по-людски. Время, как говорится, разбрасывать камни и время их собирать.

Хочу сказать ОГРОМНОЕ СПАСИБО всем вам за то, что процесс изучения веб-технологий в общении с вами проходил легко и непринужденно. Спасибо за человеческое понимание, профессиональную и дружескую отзывчивость.

Не могу не объяснить причину своего выхода из Team.

Я думаю, каждый программист на этапе своего становления мечтает свернуть горы и осчастливить мир своими идеями и гениальными программами. Время идет, растет опыт, и удовлетворения, получаемого от работы уже не хватает, чтобы ублажить чувство собственной значимости в этом мире быстро развивающихся технологий. Но во время изучения чего-то нового такие сайты, как PHPClub, дают замечательную возможность почувствовать себя нужным профессиональному сообществу, получить то профессиональное удовлетворение от знаний, которыми обладаешь и которые лежат на чердаке в дальнем ящике.

Я тоже шел таким путем. И я не скажу что это плохо. Это хорошо, это замечательно. Но наступает время, когда выполняемая работа становится рядовой, а клуб представляется как хорошая компания замечательных людей с которыми шагал по технической изнанке веба.

И печально сознавать, что ХОТЕЛ бы сделать что-то для клуба, а банально нет времени. Хотел бы пойти пообщаться, ан нет – ворох других дел (и личных в том числе).

Но не все так плохо как нарисовалось! Просто надо в нужный момент отойти в сторону и начать делать свое дело, реализовывать свои идеи (благо, они есть) и освобождать дорогу творческой молодежи, а ее у нас на клубе предостаточно. И очень хочется верить, что для новых талантов клуб станет той школой технических дебрей, которая приведет их к новым замечательным идеям и программам.

Мы встретимся, мы обязательно встретимся!

A: Сейчас являюсь скорее просто посетителем форумов, но всегда рад встретиться с коллегами и выпить по кружке пива :)

L: А какое пиво, кстати, предпочитаешь?

A: Ярпиво «Янтарное» исключительно в баночном или разливном исполнении.

L: Еще вопрос об истории Клуба. Когда появилась необходимость создать Team? И кто сначала в него входил?

A: Необходимость появилась, когда понадобилось финансирование. 1 февраля 2003 года была VIP-встреча, на которой было принято официальное решение о создании Team. Естественно, неформально Team уже существовал. Перечислять не буду, поскольку могу кого-нибудь забыть, людям будет обидно...

L: И как решился вопрос финансирования?

A: Положительно :)

L: Это были какие-то денежные вливания от активных участников или?..

A: Да. Денежное и трудовое участие.

L: Сейчас на форуме то и дело возникают мысли и вопросы, касающиеся командной работы - создать из активных участников команду, которая будет зарабатывать деньги. Эти идеи появились только сейчас?

A: Нет, они появились давно. Проблема в том, что такой команде будет нужен грамотный Project Manager. Есть ли таковой среди нас? Думаю, что нет, хотя могу и ошибаться.

L: Т.е. ты все-таки скорее не веришь, чем веришь, в такую перспективу - создание реально работающей команды профессионалов?

A: Это реально, если профи по программированию начнут интересоваться процессом управления разработкой софта. Данная тематика никак не затрагивается в данный момент на форумах PHPClub. Может быть, пора создать новый раздел — Project Management? Тема крайне актуальна.

L: Ты имел в виду, что найдутся те люди, которые смогут держать команду в руках, так? Или что программисты начнут, так сказать, расти в других направлениях?

A: Почти так. Управление процессом разработки - это наука. В команде должен быть управленец, владеющий технологией и инструментами управления, иначе большинство проектов будут обречены на провал.

L: Ну и напоследок... Наверняка в Клубе ты познакомился с приятными людьми, наверняка было множество приятных моментов. А были какие-то важные события, в которых сыграл роль Клуб?

A: Клуб оправдал свое создание уже хотя бы тем, что выступает в качестве места, где новички могут совершенно бесплатно повышать свой уровень квалификации с помощью профессионалов, а профессионалы – обмениваться опытом. Бесплатное образование в области передовых IT-технологий – мечта любого начинающего программиста. Мечта стала реальностью.

Команда этого выпуска

Авторы и переводчики

- Вадим Крючков;
- Александр Неткачев. http://devlink.crimea.ua/ ;
- Андрей Олищук [nw];
- Александр Меженков;
- Елена Тесля [Lenka];

Редакционная коллегия, корректировка

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

Подготовка обложки, верстка

- Денис Зенькович;
- Антон Чаплыгин;
- Андрей Олищук [nw].

Спасибо всем участникам проекта!

Координаты журнала:

http://mag.phpclub.ru

http://phpinside.ru

nw@phpinside.net

Обсуждаем номер на

http://phpclub.ru/talk

Для заметок

(Сюда можно заносить свои примечания)