PHP: Правильный путь

Pavel Savinov

PHP: Правильный путь

Pavel Savinov

This book is for sale at http://leanpub.com/ruphptherightway

This version was published on 2014-02-03

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Pavel Savinov by spreading the word about this book on Twitter!

The suggested hashtag for this book is #ruphptherightway.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#ruphptherightway

Оглавление

Начало

Использование стабильной версии (5.5)

Если вы только начинаете работу с PHP, убедитесь в том, что вы используете текущую стабильную версию PHP 5.5[1]. За последние несколько лет PHP добился больших успехов, добавив новые возможности. Не дайте скромной разнице между числами 5.2 и 5.5 ввести вас в заблуждение, эта разница представляет *важные* изменения. Если вам нужна функция или пример её использования, вы всегда можете найти документацию на php.net[2].

Встроенный веб-сервер

Вы можете начать изучение PHP без необходимости в установке и конфигурировании полноценного веб-сервера (необходим PHP 5.4). Для запуска сервера вам необходимо выполнить следующую команду из терминала в корневой папке веб-проекта:

```
1  > php -S localhost:8000
```

- Подробнее о встроенном консольном веб-сервере[3]

Установка на Mac

OSX поставляется с предзапакованным PHP, но, в лучшем случае, он немного отстает от стабильной версии. Lion поставляется с PHP 5.3.6 и Mountain Lion имеет 5.3.10.

Для обновления PHP в OSX вы можете установить его с помощью нескольких пакетных менеджеров[4], наиболее рекомендуемый из которых php-osx by Liip[5].

Другой вариант, скомпилировать самостоятельно[6], в этом случае убедитесь, что у вас установлен либо Xcode, либо его аналог от Apple "CLI для Xcode"[7], который можно загрузить с Apple Mac Developer Center.

В качестве полного набора «всё-в-одном», который включает PHP, веб-сервер Apache и СУБД MySQL, и всё это с хорошим управлением через GUI, попробуйте MAMP[8].

---

[1] http://www.php.net/downloads.php
[2] http://www.php.net/manual/ru/
[3] http://www.php.net/manual/ru/features.commandline.webserver.php
[4] http://www.php.net/manual/ru/install.macosx.packages.php
[5] http://php-osx.liip.ch/
[6] http://www.php.net/manual/ru/install.macosx.compile.php
[7] https://developer.apple.com/downloads
[8] http://www.mamp.info/en/downloads/index.html

Установка в Windows

PHP для Windows можно получить несколькими путями. Вы можете загрузить установочные файлы[9] и, до недавнего времени, вы могли использовать '.msi' установщик. Начиная с PHP версии 5.3.0 установщик не поддерживается.

Для изучения и локальной разработки вы можете использовать встроенный в PHP 5.4 веб-сервер, о конфигурации которого можно не беспокоиться. Если вы предпочитаете сервера «всё-в-одном», которые включают в себя полноценный веб-сервер и MySQL, тогда можете воспользоваться такими инструментами, как Web Platform Installer[10], Zend Server CE[11], XAMPP[12] или WAMP[13], которые помогут быстро развернуть окружение для разработки в Windows. Но, стоит сказать, что эти инструменты будут отличаться от продакшна, так что будьте осторожны и учитывайте эти различия, если вы работаете на Windows и деплоите на Linux.

Если вам нужно запустить конечную систему на Windows, то IIS7 даст вам лучшую стабильность и производительность. Вы можете использовать phpmanager[14] (плагин для IIS7) для легкого конфигурования и управления PHP. IIS7 поставляется с встроенным FastCGI, вам нужно просто настроить PHP в качестве обработчика. Для получения помощи и дополнительной информации посетите iis.net[15].

Vagrant

Запуск вашего приложения в разных окружениях на этапе разработки и продакшна может привести к различным багам, которые дадут о себе знать уже непосредственно при работе приложения. Также сложно поддерживать в разных окружениях стабильные версии для всех библиотек, которые используются при работе в команде разработчиков.

Если вы разрабатываете на Windows и деплоите на Linux (или что-либо отличающееся от Windows) или разрабатываете в команде, вы должны рассмотреть возможность использования виртуальной машины. Это звучит сложно, но, используя Vagrant[16], вы можете установить простую виртуальную машину всего лишь в несколько шагов. Они могут быть как выполнены вручную, так и с помощью специализированного софта, например, Puppet[17] или Chef[18], который автоматизирует эту задачу. Использование этого софта гарантирует использование одинаковой конфигурации для нескольких машин, что избавляет вас от

[9] http://windows.php.net
[10] http://www.microsoft.com/web/downloads/platform.aspx
[11] http://www.zend.com/en/products/server-ce/
[12] http://www.apachefriends.org/en/xampp.html
[13] http://www.wampserver.com/
[14] http://phpmanager.codeplex.com/
[15] http://php.iis.net/
[16] http://vagrantup.com/
[17] http://www.puppetlabs.com/
[18] http://www.opscode.com/

необходимости поддержки сложных списков установки. Вы также можете удалить вашу машину, и пересоздать её без большого количества ручных шагов, что делает создание «свежей» виртуалки очень простым.

Vagrant создает общие папки, которые используются для совместного использования кода между вашим хостом и виртуальной машиной, а это означает, что вы можете создавать и редактировать ƒайлы на хосте и позже запускать код в вашей виртуальной машине.

Стандарты написания кода

Сообщество PHP является очень большим и разнообразным, сочетая в себе бесчисленное количество библиотек, фреймворков, и различных компонентов. Для PHP разработчика это обычная практика — выбрать несколько из них и соединить в одном проекте. Очень важно придерживаться общих стандартов написания кода (так точно, насколько это возможно) в своём PHP коде, чтобы позволить разработчикам сочетать и использовать различные библиотеки для своих проектов.

Группа Совместимости Фреймворков[19] предложила и одобрила ряд стилевых рекомендаций, известных как PSR-0[20], PSR-1[21] и PSR-2[22]. Не дайте веселым именам смутить вас, эти рекомендации представляют собой набор правил, которых начинают придерживаться такие проекты, как Drupal, Zend, Symfony, CakePHP, phpBB, AWS SDK, FuelPHP, Lithium и другие. Вы можете использовать их при работе над собственным проектом, или в дальнейшем использовать ваш собственный стиль.

В идеале, вы должны писать PHP код, придерживаясь известных стандартов. Это может быть любая комбинация PSR-ов, или один из стандартов кода, сделанных PEAR или Zend. Это позволит другим разработчикам легко читать и работать с вашим кодом, и приложения, которые используют компоненты, смогут сохранить структуру приложения, даже работая с огромным количеством стороннего кода.

- Подробнее о PSR-0[23]
- Подробнее о PSR-1[24]
- Подробнее о PSR-2[25]
- Подробнее о Стандартах PEAR[26]
- Подробнее о Стандартах Zend[27]

Вы можете использовать PHP_CodeSniffer[28] чтобы проверить код на соответствие одной из этих рекомендаций, а также плагин для текстовых редакторов, таких как, к примеру, Sublime Text 2[29] чтобы получить отчёт в реальном времени.

---

[19] http://www.php-fig.org/
[20] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-0.md
[21] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md
[22] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md
[23] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-0.md
[24] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md
[25] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md
[26] http://pear.php.net/manual/ru/standards.php
[27] http://framework.zend.com/wiki/display/ZFDEV2/Coding+Standards
[28] http://pear.php.net/package/PHP_CodeSniffer/
[29] https://github.com/benmatselby/sublime-phpcs

Используйте PHP Coding Standards Fixer[30], созданный Фабиеном Потенсьером, для автоматического исправления синтаксиса вашего кода так, чтобы он соответствовал этим стандартам, что спасет вас от исправления каждой проблемы вручную.

Английский язык является наиболее предпочтительным для всех символических имен и инфраструктуры кода. Комментарии могут быть написаны на любом языке, который будет легко читаем текущими и будущими разработчиками, которым предстоит работать на кодом.

[30] http://cs.sensiolabs.org/

Основные моменты языка

Парадигмы программирования

PHP представляет собой гибкий, динамичный язык, который поддерживает несколько техник программирования. Он значительно развился в течение последних нескольких лет: добавлена мощная объектно-ориентированная модель в PHP 5.0 (2004), анонимные функции (замыкания) и пространства имен в PHP 5.3 (2009), а также трейты в PHP 5.4 (2012).

Объектно-ориентированное программирование

PHP реализует очень большой набор особенностей объектно-ориентированного программирования, включая поддержку классов, абстрактных классов, интерфейсов, наследования, конструкторов, клонирования, исключений и т.д.

- Подробнее об объектно-ориентированном PHP[31]
- Подробнее о трейтах[32]

Функциональное программирование

PHP поддерживает первоклассные функции, т.е. функция может быть применена к переменной. И определенные пользователем, и встроенные функции могут быть применены к переменной и вызываться динамически. Функции могут быть переданы, как аргумент к другой функции (эта особенность называется функцией высшего порядка), а также функция может возвращать другую функцию.

Рекурсия — это особенность, которая позволяет функции вызывать саму себя, это поддерживается языком, но большая часть кода PHP фокусируется на итерации.

Анонимные функции (замыкания) поддерживаются PHP начиная с версии 5.3 (2009).

В PHP 5.4 добавлена возможность связывать замыкание с областью видимости объекта, а также улучшена поддержка callables (всё, что может быть вызвано), так что они могут быть использованы наравне с анонимными функциями практически во всех случаях.

- Продолжить чтение про Функциональное программирование PHP

[31] http://www.php.net/manual/ru/language.oop5.php
[32] http://www.php.net/manual/ru/language.oop5.traits.php

- Подробнее об Анонимных Функциях[33]
- Подробнее о классе Closure[34]
- Больше информации в Closures RFC[35]
- Подробнее о Callables[36]
- Узнать о динамически вызываемых функциях с `call_user_func_array`[37]

Meta Programming

PHP поддерживает несколько форм метапрограммирования, что реализуется с помощью таких механизмов, как Reflection API и Магические Методы. Доступно много Магических Методов, например: `__get()`, `__set()`, `__clone()`, `__toString()`, `__invoke()`, и т.д., которые позволяют отслеживать поведение внутри класса. Разработчики Ruby часто говорят, что PHP не хватает `method_missing`, но он доступен, как `__call()` и `__callStatic()`.

- Подробнее о Магических Методах[38]
- Подробнее о Reflection[39]

Пространства имен

Как было сказано выше, сообщество PHP состоит из множества разработчиков, создающих очень много кода. Это значит, что одна библиотека PHP может иметь такое же название класса, как и другая. Когда обе библиотеки используются в одном пространстве имен, они конфликтуют и возникают проблемы.

Пространства имен решают эту проблему. Как описано в руководстве PHP, пространства имен можно сравнить с папками операционной системы, которые являются *пространствами имен* файлов; два файла с одинаковым именем могут сосуществовать в разных директориях. Подобно этому, два PHP класса с одинаковым названием могут существовать в разных пространствах имен PHP.

Использование пространств имен необходимо для того, чтобы избежать конфликтов при использовании вашего кода с библиотеками других разработчиков.

Один из рекомендуемых способов использования пространств имен описан в PSR-0[40], который призван обеспечить стандарты для описания файлов, классов и пространств имен, что позволяет создавать PnP код.

[33] http://www.php.net/manual/ru/functions.anonymous.php
[34] http://php.net/manual/ru/class.closure.php
[35] https://wiki.php.net/rfc/closures
[36] http://php.net/manual/ru/language.types.callable.php
[37] http://php.net/manual/ru/function.call-user-func-array.php
[38] http://php.net/manual/ru/language.oop5.magic.php
[39] http://www.php.net/manual/ru/intro.reflection.php
[40] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-0.md

- Подробнее о пространствах имен[41]
- Подробнее о PSR-0[42]

Стандартная Библиотека PHP (SPL)

Стандартная библиотека PHP (SPL) поставляется вместе с PHP и предоставляет набор классов и интерфейсов. Она состоит в основном из часто используемых классов структур данных (стек, очередь, куча, и т.д.), а также итераторов, которые предназначены для прохождения через эти структуры данных или ваши собственные классы, которые реализуют интерфейсы SPL.

- Подробнее о SPL[43]

Интерфейс командной строки

Главная цель, с которой был создан PHP — это разработка веб-приложений, но он также полезен при написания кода для интерфейса командной строки (CLI). PHP программы командной строки могут помочь вам автоматизировать такие общие задачи, как тестирование, развертывание и администрирование приложения.

CLI PHP программы очень мощные, потому что вы можете использовать код вашего приложения напрямую, без нужды в создании и обеспечении безопасности веб-интерфейса (GUI) для него. Только убедитесь, что вы не используете для ваших скриптов (CLI) корень вашего веб-сервера.

Попробуйте запустить PHP из консоли:

```
1 > php -i
```

Опция `-i` выдаст вам конфигурацию вашего PHP, подобно функции [`phpinfo`][phpifno].

Опция `-a` предоставляет доступ к интерактивной оболочке, подобно ruby IRB или интерактивной оболочки python. Также существует целый ряд других полезных опций командной строки[44].

Давайте напишем простую «Привет, $name» программу CLI. Чтобы это сделать, создайте файл с именем `hello.php`, как показано ниже.

---

[41] http://php.net/manual/en/language.namespaces.php
[42] https://github.com/getjump/fig-standards/blob/master/accepted/PSR-0.md
[43] http://php.net/manual/ru/book.spl.php
[44] http://www.php.net/manual/ru/features.commandline.options.php

```php
<?php
if ($argc != 2) {
    echo "Использование: php hello.php [name].\n";
    exit(1);
}
$name = $argv[1];
echo "Привет, $name\n";
```

PHP устанавливает две специальные переменные, основанных на аргументах, с которыми запущен ваш скрипт. `$argc`[45] — это переменная с числовым значением, которая содержит количество переданных аргументов, `$argv`[46] — это массив, содержащий значение каждого аргумента. Первый аргумент — всегда название вашего PHP скрипта, в этом случае `hello.php`.

Выражение `exit()` используется с ненулевым числом, чтобы дать оболочке понять, что команда не удалась. Часто используемые коды завершения можно найти здесь[47]

Для запуска сценария, указанного выше, наберите в командной строке:

```
> php hello.php
Использование: php hello.php [name]
> php hello.php Мир
Привет, Мир
```

- Подробнее о запуске PHP из командной строки[48]
- Подробнее о настройке Windows для запуска PHP из командной строки[49]

XDebug

Один из самых полезных инструментов в разработке программного обеспечения — хороший отладчик. Он позволяет вам отследить исполнение вашего кода и контролировать содержимое вашего стека. XDebug — это PHP отладчик, который может использоваться различными IDE, чтобы дать вам возможность устанавливать Брейкпоинты (точки отладки кода) и контролировать стек. Он также позволяет использовать такие инструменты, как PHPUnit и KCacheGrind, для покрытия кода тестами и его профилирования.

Если вы оказываетесь в безвыходном положении при использовании var_dump/print_r, и у вас не получается найти решение, то возможно вам поможет использование отладчика.

[45] http://php.net/manual/ru/reserved.variables.argc.php
[46] http://php.net/manual/ru/reserved.variables.argv.php
[47] http://www.gsp.com/cgi-bin/man.cgi?section=3&topic=sysexits
[48] http://php.net/manual/ru/features.commandline.php
[49] http://www.php.net/manual/ru/install.windows.commandline.php

Установка XDebug[50] может оказаться сложной, но одна из самых полезных его функций это «Удаленная отладка» — если вы разрабатываете код локально и затем тестируете его в локальной машине или на другом сервере, Удаленная Отладка — это возможность, которую вы захотите сразу же включить.

Стандартно, вы отредактируете ваш Apache VHost или .htaccess файл со следующими значениями:

```
1 php_value xdebug.remote_host=192.168.?.?
2 php_value xdebug.remote_port=9000
```

“remote_host” и “remote_port” будут указывать на ваш локальный компьютер и порт, который вы указали в вашей IDE для прослушивания. Дальше достаточно включить режим «ожидания соединений» в вашей IDE, и загрузить URL:

```
1 http://your-website.example.com/index.php?XDEBUG_SESSION_START=1
```

Ваша IDE теперь будет перехватывать текущее состояние, позволяя вам устанавливать брейкпоинты и исследовать значения в памяти по мере выполнения скрипта.

- Подробнее о XDebug[51]

[50] http://xdebug.org/docs/install
[51] http://xdebug.org/docs/

Менеджер зависимостей

Существует много библиотек, фреймворков и компонентов PHP на выбор. Ваш проект, скорее всего, будет использовать некоторые из них — это и есть зависимости проекта. До недавнего времени в PHP не существовало удобного способа для управления зависимостями проекта. Даже если вы управляете ими вручную, вам приходилось беспокоиться об автозагрузчиках. Больше это не требуется.

В настоящее время существует две основные системы управления пакетами для PHP — Composer и PEAR. Какая из них подходит именно вам? Ответ — обе.

- Используйте **Composer** для управления зависимостями одного проекта.
- Используйте **PEAR** для управления зависимостями всех проектов во всей вашей системе.

В общем, пакеты Composer будут доступны только в проектах, для которых вы явно укажете его использование, тогда как пакеты PEAR будут доступны во всех ваших PHP проектах. PEAR, на первый взгляд, может показаться более простым подходом, но есть преимущества в использовании подхода «проект-к-проекту» для зависимостей.

Composer и Packagist

Composer является **блестящим** менеджером зависимостей для PHP. Укажите список зависимостей вашего проекта в файле `composer.json` и, с помощью нескольких простых команд, Composer автоматически скачает зависимости вашего проекта и установит для вас автозагрузку.

На данный момент существует много PHP библиотек, которые совместимы с Composer, готовых для использования в вашем проекте. Список этих «пакетов» есть на Packagist[52], официальном репозитории для Composer-совместимых PHP библиотек.

Как установить Composer

Вы можете установить Composer локально (в вашей текущей рабочей директории; хотя это не рекомендуется) или глобально (например /usr/local/bin). Предположим, вы хотите установить Composer локально. Из корневой директории вашего проекта выполните:

[52] http://pear.php.net/

```
curl -s https://getcomposer.org/installer | php
```

Это позволит загрузить файл `composer.phar` (бинарный PHP-архив). Вы можете запустить его, используя `php` для управления зависимостями вашего проекта. Если вы скачаете код напрямую в ваш интерпретатор, пожалуйста, сперва прочитайте код онлайн, для подтверждения его безопасности.

Как установить Composer (вручную)

Ручная установка Composer — это продвинутая техника; однако, существуют причины, по которым разработчик может предпочесть именно этот метод использованию интерактивной установки. Интерактивная установка проверяет настройки PHP, чтобы подтвердить, что:

- Используется необходимая версия PHP
- Файлы `.phar` могут быть верно выполнены
- Определенные права на каталог достаточны
- Не установлены конфликтные расширения
- Установлены необходимые настройки `php.ini`

В случае, если ни одно из этих условий не соблюдено, вы должны принять решение стоит ли идти на такой компромисс. Ниже описано, как установить Composer вручную:

```
curl -s http://getcomposer.org/composer.phar -o $HOME/local/bin/composer
chmod +x $HOME/local/bin/composer
```

Путь `$HOME/local/bin` (или другой каталог, выбранный вами) должен находиться в вашей переменной окружения `$PATH`. Это позволит быть доступной команде `composer`.

Если вы прочтете документацию Composer, которая гласит, что нужно запускать Composer с помощью команды `php composer.phar install`, вы можете заменить эту команду на:

```
composer install
```

Как объявить и установить зависимости

Composer продолжает следить за зависимостями вашего проекта в файле `composer.json`. Вы можете управлять им вручную, если вам нравится, или же использовать сам Composer. Команда `php composer.phar require` добавляет зависимость в проект и, если в каталоге нет файла `composer.json`, он будет создан. Далее мы рассмотрим пример, который добавляет Twig[53], как зависимость вашего проекта. Запустите это в корневой директории вашего проекта, куда вы загружали `composer.phar`:

[53] http://pear.php.net/manual/ru/installation.getting.php

```
php composer.phar require twig/twig:~1.8
```

Аналогично команда `php composer.phar init` проведет вас через создание полного файла `composer.json` для вашего проекта. Есть и другой путь, когда вы создадите файл `composer.json` вы можете сказать Composer, чтобы он скачал все ваши зависимости в папку `vendors/`. Это также применимо для проектов, которые вы загрузили и которые предоставляют файл `composer.json`:

```
php composer.phar install
```

Затем добавьте этот код в основной PHP-файл вашего приложения; это укажет PHP использовать автозагрузчик Composer для зависимостей вашего проекта.

```
<?php
require 'vendor/autoload.php';
```

Теперь вы можете использовать зависимости вашего проекта и они будут автоматически загружаться (по требованию).

Обновление зависимостей

Composer создает файл `composer.lock` который хранит точную версию каждого пакета, который он загрузил во время первого запуска `php composer.phar install`. Если вы поделились проектом с другими разработчиками и файл `composer.lock` является частью него, то при запуске `php composer.phar install` они получат ту же версию, что и вы. Чтобы обновить ваши зависимости запустите `php composer.phar update`.

Очень удобно гибко указывать требуемые версии. Если вы нуждаетесь в версии ~1.8, что значит "всё что новее 1.8.0, но меньше 2.0.x-dev". Вы также можете использовать шаблон *, например `1.8.*`. Теперь команда Composer `php composer.phar update` обновит все ваши зависимости до новейших версий, которые соответствуют указанным ограничениям.

Проверка ваших зависимостей на безопасность

Security Advisories Checker[54] является веб-сервисом и инструментом командной строки, оба из которых изучают ваш файл `composer.lock` и скажут вам если вам нужно обновить любое из ваших зависимостей.

- Подробнее о Composer[55]

[54] http://pear.php.net/packages.php

[55] http://pear.php.net/manual/ru/guide.users.commandline.channels.php

PEAR

Другим ветераном среди пакетных менеджеров, которым наслаждаются многие PHP-разработчики, является PEAR[56]. Он работает практически так же, как и Composer, но имеет несколько важных отличий.

PEAR требует, чтобы каждый пакет имел определенную структуру, это означает, что автор пакета должен подготовить его для использования с PEAR. Использование проекта, который не был подготовлен для работы с PEAR невозможно.

PEAR устанавливает пакеты глобально, что означает то, что после установки, они доступны всем проектам на этом сервере. Это может быть полезно, если много проектов строятся на тех же пакетах с той же версией, но может привести к проблемам, если проекты разрабатывались для разных версий.

Как установить PEAR

Вы можете установить PEAR, загрузив установщик phar и выполнив его. Документация PEAR содержит подробную инструкцию по установке[57] для каждой операционной системы.

Если вы используете Linux, вы также можете посмотреть наличие PEAR в пакетном менеджере вашего дистрибутива. Debian и Ubuntu, к примеру, содержат информацию о пакете `php-pear` в пакетном менеджере apt.

Как установить пакет

Если пакет существует в списке пакетов PEAR[58], вы можете установить его, указав официальное название:

```
1 pear install foo
```

Если пакет выложен на другом канале, вам нужно сначало сделать `discover` этого канала и затем указать его во время установки. Подробнее об этом в использование каналов[59].

- Подробнее о PEAR[60]

[56] http://pear.php.net/
[57] http://pear.php.net/manual/ru/installation.getting.php
[58] http://pear.php.net/packages.php
[59] http://pear.php.net/manual/ru/guide.users.commandline.channels.php
[60] http://pear.php.net/

Обработка зависимостей PEAR с Composer

Если вы уже используете Composer[61] и желаете установить какой-то код из PEAR, вы можете использовать Composer для обработки зависимостей PEAR. Этот пример установит код из `pear2.php.net`:

```
{
    "repositories": [
        {
            "type": "pear",
            "url": "http://pear2.php.net"
        }
    ],
    "require": {
        "pear-pear2/PEAR2_Text_Markdown": "*",
        "pear-pear2/PEAR2_HTTP_Request": "*"
    }
}
```

Первый раздел `"repositories"` даст понять Composer, что он должен сделать “initialise” (или “discover” в терминологии PEAR) репозиторий pear. Затем секция require укажет именам пакетов префикс, как ниже:

> pear-channel/Package

Префикс “pear” жестко ограничен, чтобы избежать любых конфликтов, так как каналы Pear могут быть схожи с другими поставщиками пакетов например, вместо короткого имени (или полного URL) может быть использовано для объявления в каком канале находится пакет.

Когда код будет установлен он будет доступен в вашей папке vendor и автоматически доступен через автозагрузчик (файл Autoload) Composer.

> vendor/pear-pear2.php.net/PEAR2_HTTP_Request/pear2/HTTP/Request.php

Чтобы использовать этот пакет PEAR просто объявите как ниже:

```
$request = new pear2\HTTP\Request();
```

- Подробнее о использовании PEAR с Composer[62]

[61] /#composer_и_packagist

[62] http://getcomposer.org/doc/05-repositories.md#pear

Практики написания кода

Основы

PHP — это обширный язык, который позволяет разработчикам всех уровней писать код не только быстро, но и эффективно. В любом случае, изучая язык, мы нередко забываем основы, которые мы изучали изначально (или бегло просмотрели), в пользу коротких путей и/или вредных привычек. Чтобы помочь в борьбе с этой общей проблемой, эта секция предназначена для напоминания разработчикам основ практик написания кода PHP.

- Продолжить чтение Основы

Дата и время

PHP содержит встроенный класс DateTime, предназначенный для чтения, записи, сравнения и вычисления даты или времени. Также в PHP много функций, связанных с датой и временем, помимо класса DateTime, но класс предоставляет хороший объектно-ориентированный интерфейс для решения большинства задач. Он способен даже обрабатывать временные зоны, но это уже не рассматривается в данном коротком введении.

Для начала работы с DateTime, сконвертируйте «сырую» строку даты и времени в объект с помощью фабричного метода `createFromFormat()` или выполните `new \DateTime`, чтобы получить текущую дату и время. Используйте метод `format()` для конвертирования DateTime обратно в строку для вывода.

```php
<?php
$raw = '22. 11. 1968';
$start = \DateTime::createFromFormat('d. m. Y', $raw);

echo 'Start date: ' . $start->format('m/d/Y') . "\n";
```

Вычисления с DateTime возможны с использованием класса DateInterval. У класса DateTime есть методы `add()` и `sub()`, которые принимают DateInterval, как аргумент. Не пишите код, который ожидает одинаковое число секунд каждый день, перевод часов и смена часовых поясов разрушат это предположение. Вместо этого используйте интервалы дат. Для расчета разницы между датами используйте метод `diff()`. Он вернет новый объект DateInterval, который очень легко отобразить.

```php
<?php
// создает копию $start и добавляет 1 месяц и 6 дней
$end = clone $start;
$end->add(new \DateInterval('P1M6D'));

$diff = $end->diff($start);
echo 'Difference: ' . $diff->format('%m месяц, %d дней (total: %a дней)') . "\n";
// Разница : 1 месяц, 6 дней (всего : 37 дней)
```

С объектами DateTime, вы можете использовать стандартные методы сравнения:

```php
<?php
if ($start < $end) {
    echo "Начальная дата раньше конечной!\n";
}
```

И последний пример для демонстрации класса DatePeriod. Он используется для перебора повторяющихся событий. Класс может принимать два объекта DateTime, начало и конец, и интервал, для которого он вернет все события между ними.

```php
<?php
// выводит все четверги между началом и концом
$periodInterval = \DateInterval::createFromDateString('first thursday');
$periodIterator = new \DatePeriod($start, $periodInterval, $end, \DatePeriod::EXC\
LUDE_START_DATE);
foreach ($periodIterator as $date) {
    // вывести каждую дату в периоде
    echo $date->format('m/d/Y') . ' ';
}
```

- Подробнее о DateTime[63]
- Подробнее о форматировании даты[64] (разрешенные опции строки формата даты)

Design Patterns

При построении приложения полезно использовать в коде шаблоны, а также придерживаться некоторых стандартов для всей структуры проекта. Использование шаблонов полезно,

[63] http://www.php.net/manual/ru/book.datetime.php
[64] http://www.php.net/manual/ru/function.date.php

потому что оно упрощает управление кодом, а также позволяет другим разработчикам быстро понять, как всё взаимодействует друг с другом.

Если вы используете фреймворк, то большинство высокоуровневого кода и структура проекта будет основываться на архитектуре фреймворка, поэтому большинство решений относительно шаблона сделано за вас. Но всё же наиболее верным решением будет выбрать наиболее подходящие шаблоны и следовать им в коде, который вы пишете на базе фреймворка. С другой стороны, если вы не используете фреймворк для построения приложения, тогда вы должны найти шаблоны, которые наилучшим образом соответствуют типу и размеру приложения, которое вы создаете.

- Продолжить чтение Шаблоны проектирования

Исключения

Исключения — это неотъемлимая часть большинства популярных языков программирования, но зачастую PHP разработчики не уделяют им должного внимания. Языки, подобные Ruby, очень подробно обрабатывают исключения, поэтому, если что-то идёт не верно, например: не удался HTTP запрос, запрос к базе данных происходит неправильно или если запрошенное изображение не было найдено, Ruby (или используемые гемы) выбросит исключение на экран, помогающее понять где вы допустили ошибку.

PHP сам по себе довольно слаб в плане этого и вызов `file_get_contents()`, как правило, даст вам только `FALSE` и предупреждение. Многие устаревшие PHP-фреймворки, как CodeIgniter, просто вернут false, добавят сообщение в свой собственный журнал и, может быть, дадут вам использовать метод, как `$this->upload->get_error()`, чтобы посмотреть, что пошло не так. Проблема в том, что вы должны искать ошибку и проверять документацию, чтобы понять, какой ошибочный метод существует в этом классе, вместо того, чтобы сделать это всё более очевидным.

Еще одна проблема в том, что классы автоматически выдают ошибку на экран и закрывают процесс. Когда вы делаете это, вы не даете другому разработчику динамически обработать эту ошибку. Исключения должны быть выброшены, чтобы дать разработчику знать об ошибке и выбрать, как ее обработать. Например:

```php
<?php
$email = new Fuel\Email;
$email->subject('My Subject');
$email->body('How the heck are you?');
$email->to('guy@example.com', 'Some Guy');

try
{
    $email->send();
}
catch(Fuel\Email\ValidationFailedException $e)
{
    // Валидация не удалась
}
catch(Fuel\Email\SendingFailedException $e)
{
    // Драйвер не может отправить сообщение
}
```

Исключения SPL

Универсальный класс `Exception` предоставляет очень мало отладочного контекста для разработчика; как бы то ни было, для того чтобы исправить это, можно создать специализированный класс, который будет расширять возможности универсального класса `Exception`:

```php
<?php
class ValidationException extends Exception {}
```

Это означает, что вы можете добавить несколько блоков отлова и обрабатывать разные исключения по-разному. Это может привести к созданию измененных Исключений, некоторые из которых можно было бы избежать, используя Исключения SPL, предоставляемые расширением SPL[65].

Например, если вы используете магический метод `__call()` и вами был вызван неизвестный метод, то вместо выбрасывания стандартного исключения, которое очень расплывчато, или вместо создания своего исключения, вы можете просто использовать `throw new BadFunctionCallException;`.

- Подробнее об Исключениях[66]

[65] /#standard_php_library
[66] http://php.net/manual/ru/language.exceptions.php

- Подробнее о SPL Исключениях[67]
- Вложенные исключения в PHP[68]
- Лучшие практики использования исключений в PHP 5.3[69]

Автоматически очищено с помощью PDO $stmt->execute(); ~~~~~~~~

Это правильный код. Он использует связанный параметр в выражении PDO. Это позволяет избежать ввода некоректного ID перед тем, как передать запрос в базу данных, тем самым предотвращая потенциальные SQL-инъекции.

- Подробнее о PDO[70]

Вы также должны понимать, если подключение не закрыто должным образом, то оно использует много ресурсов, которые тратятся впустую, впрочем это больше относится к другим языкам. Используя PDO, вы можете неявно закрывать подключение уничтожив объект — все ссылки на него будут удалены, т.е. установлены в NULL. Если не сделать этого явно, PHP закроет подключение за вас, когда выполнение скрипт завершится, если только вы не используете постоянные подключения.

- Подробнее о подключениях PDO[71]

Уровни абстракции

Многие фреймворки предоставляют собственный уровень абстракции, который может строиться на основе PDO. Такая фактическая абстракция баз данных позволяет оборачивать запросы на PHP в методы, которые отсутсвуют в одной системе баз данных, но работают в другой. Это, конечно, добавит небольшие накладные расходы, но если вы строите портативные приложения, которым необходима работа с MySQL, PostgreSQL и SQLite, тогда, для чистоты кода, минимальными накладными расходами можно пренебречь.

Некоторые уровни абстракции построены с использованием PSR-0 стандарта, поэтому могут быть установлены в любое приложение:

- Aura SQL[72]
- Doctrine2 DBAL[73]
- ZF2 Db[74]
- ZF1 Db[75]

[67] http://php.net/manual/ru/spl.exceptions.php
[68] http://www.brandonsavage.net/exceptional-php-nesting-exceptions-in-php/
[69] http://ralphschindler.com/2010/09/15/exception-best-practices-in-php-5-3
[70] http://www.php.net/manual/ru/book.pdo.php
[71] http://php.net/manual/ru/pdo.connections.php
[72] https://github.com/auraphp/Aura.Sql
[73] http://www.doctrine-project.org/projects/dbal.html
[74] http://packages.zendframework.com/docs/latest/manual/en/index.html#zend-db
[75] http://framework.zend.com/manual/ru/zend.db.html

Безопасность

Безопасность веб-приложений

Есть плохие люди, которые могут и хотят взломать ваши веб-приложения. Важно принять необходимые меры предосторожности, чтобы укрепить безопасность вашего приложения. К счастью, прекрасные люди в The Open Web Application Security Project[76](OWASP) составили полный список известных проблем безопасности и методов защиты от них. Это должно быть прочитано любым разработчиком, заботящемся о безопасности.

- Прочитать руководство по безопасности OWASP[77]

Хэширование паролей

Наверное, каждый PHP-разработчик занимается разработкой приложений, которые нуждается в пользовательской авторизации. Имя пользователя и пароль хранятся в базе данных и позже используются для авторизации пользователя.

Очень важно правильно *хэшировать*[78] пароль перед его сохранением. Хэширование пароля является необратитым, односторонняя функция применяется на пользовательских паролях. Она возвращает строку определенной длины, которую невозможно расшифровать. Это значит, что вы можете сравнить один хэш с другим, чтобы понять, что они пришли из одной и той же исходной строки, но вы не можете определить оригинальную строку. Если ваши пароли не захэшированы, и доступ к базе данных получен третьей стороной, то ваши пользовательские аккаунты теперь скомпрометированы. Некоторые пользователи (к сожалению) могут использовать один и тот же пароль для разных сервисов. Как бы то ни было, очень важно серьезно относиться к безопасности.

Хэширование паролей с функцией `password_hash`

В PHP 5.5 была представлена функция `password_hash`. Сейчас она использует BCrypt, сильнейший алгоритм, поддерживаемый PHP. Она будет обновлена в будущем, для поддержки бОльшего числа алгоритмов, по мере необходимости. Библиотека `password_compat` была создана для обратной совместимости с PHP >= 5.3.7.

Ниже мы хэшируем строку и далее сверяем его с новой строкой. Поскольку наши две исходных строки отличны (‘secret-password’ и ‘bad-password’) эта авторизация будет неудачной.

[76] http://www.php.net/manual/ru/book.filter.php
[77] http://www.php.net/manual/ru/filter.filters.sanitize.php
[78] http://www.php.net/manual/ru/filter.filters.validate.php

```
<?php                                                                    \
                                                                         \

require 'password.php';

$passwordHash = password_hash('secret-password', PASSWORD_DEFAULT);

if (password_verify('bad-password', $passwordHash)) {
    //Правильный пароль
} else {
    //Неправильный пароль
}
```

- Подробнее о `password_hash`[79]
- `password_compat` для PHP >= 5.3.7 && < 5.5[80]
- Подробнее о хэшировании в отношении криптографии[81]
- PHP `password_hash` RFC[82]

Фильтрация данных

Никогда не доверяйте пользовательскому вводу, который передается вашему PHP коду. Всегда проверяйте и очищайте пользовательский ввод перед его использованием в коде. Функции `filter_var` и `filter_input` помогут очистить переменные, а также проверить соответствие введенных данных некоторому формату (например адрес электронной почты).

Пользовательский ввод может быть различным: `$_GET` и `$_POST`, данные введенные в форму, некоторые значения в суперглобальной переменной `$_SERVER` и тело HTTP запроса открытое с помощью `fopen('php://input', 'r')`. Запомните, что пользовательский ввод не ограничивается данными формы, отправленной пользователем. Отправляемые и загружаемые файлы, значения сессий, данные cookie и данные сторонних веб-сервисов также приравниваются к пользовательскому вводу.

Хотя пользовательские данные могут быть без проблем сохранены, скомбинированы и к ним может быть получен доступ позже, они всё ёще является пользовательским вводом. Каждый раз, когда вы что-либо обрабатываете, объединяете или подключаете данные в ваш код, спросите себя, отфильтрованы ли эти данные и можно ли им доверять.

Данные могут быть *отфильтрованы* по-разному, в зависимости от их назначения. Например, когда нефильтрованные данные, введенные пользоватем, передаются в HTML код

[79] http://www.php.net/manual/ru/book.filter.php
[80] http://www.php.net/manual/ru/filter.filters.sanitize.php
[81] http://www.php.net/manual/ru/filter.filters.validate.php
[82] http://php.net/manual/ru/function.filter-var.php

страницы, он может выполнить HTML и JavaScript на вашем сайте! Этот тип атаки известен, как Cross-Site-Scripting (XSS) и может иметь очень серьзные последствия. Один из способов избежать XSS заключается в очистке ввода от всех HTML тэгов (их удалением, или заменой на HTML символы) с помощью функции `strip_tags` или экранирование символов в равносильные им HTML сущности с функцией `htmlentities` или `htmlspecialchars`.

Другой пример, передача данных для выполнения командной строкой. Это может быть крайне опасно (и, как правило — это плохая идея), но вы можете использовать встроенную функцию `escapeshellarg` для очистки аргументов командной строки.

Последний пример, принимает пользовательский ввод, чтобы определить, какой файл загружать из файловой системы. Это может быть использовано, для изменения имени файла, на путь файла. Вам нужно убрать "/", "../", нулевые байты[83] или другие символы из пути файла, так чтобы скрипт не мог загружать скрытые, непубличные или конфиденциальные файлы.

- Подробнее о фильтрации данных[84]
- Подробнее о функции `filter_var`[85]
- Подробнее о функции `filter_input`[86]
- Подробнее о обработке нулевых байтов[87]

Санитизация

Санитизация удаляет (или экранирует) неправильные или небезопасные символы из пользовательского ввода.

Например, вам необходимо нормализовать пользовательский ввод перед подключением ввода в HTML или его вставкой в сырой SQL запрос. Когда вы используете связанные параметры с PDO, они будут очищать ввод за вас.

Иногда требуется разрешить некоторые безопасные HTML тэги в вводе, когда он подключается в HTML страницу. Это очень трудно сделать и многие избегают этого, используя ограниченное форматирование, как например Markdown или BBCode, либо библиотеки с белым списком, как HTML Purifier[88] существующие по этой причине.

Санитизационные фильтры[89]

[83] http://php.net/manual/ru/security.filesystem.nullbytes.php
[84] http://www.php.net/manual/ru/book.filter.php
[85] http://php.net/manual/ru/function.filter-var.php
[86] http://www.php.net/manual/ru/function.filter-input.php
[87] http://php.net/manual/ru/security.filesystem.nullbytes.php
[88] http://htmlpurifier.org/
[89] http://www.php.net/manual/ru/filter.filters.sanitize.php

Валидация

Валидация гарантирует, что пользовательский ввод, является тем, что вы ожидаете. Например, вы можете валидировать : адрес электронной почты, номер телефона или возраст при обработке запроса регистрации.

Валидационные фильтры[90]

Конфигурационные файлы

Когда вы создаете файлы конфигурации для ваших приложений, рекомендуется использование одного из следующих способов:

- Рекомендуется хранить вашу конфигурационную информацию там, где к ней не может быть получен доступ напрямую, а доступ к ней осуществлялся через файловую систему.
- Если вы вынуждены хранить конфигурационные файлы в корневом каталоге, именуйте файл с расширением `.php`. Это гарантирует, что, если к скрипту обратятся напрямую, он не будет выведен, как обычный текст.
- Информация в файлах конфигурации, должна быть защищена соотвественно, либо с помощью шифрования или системных прав группы/пользователя файла.

Использование глобальных переменных

Примечание: С появлением PHP 5.4 директива `register_globals` была удалена и больше не может быть использована. Это касается тех, кому нужно обновить старое приложение.

Включенный параметр конфигурации `register_globals` делает несколько типов переменных(в том числе из `$_POST`, `$_GET` и `$_REQUEST`) глобальными, доступными в глобальной области видимости вашего приложение. Это может легко привести к проблемам с безопасностью, поскольку ваше приложение не сможет эффективно определить откуда пришли данные.

Например : `$_GET['foo']` будет доступна через `$foo`, которая может заместить переменную, которая не была объявлена. Если вы используете PHP < 5.4.0 **убедитесь** что `register_globals` **off** (выключена).

- Register_globals в руководстве PHP[91]

[90] http://www.php.net/manual/ru/filter.filters.validate.php
[91] http://www.php.net/manual/ru/security.globals.php

Сообщения об ошибках

Логирование ошибок полезно при поиске проблемных мест вашего приложения, также логирование может выдать информацию о структуре вашего приложения. Для эффективной защиты вашего приложения от проблем, которые могут быть вызваны выводом этих сообщений, вам необходимы различные настройки сервера для разработки и продакшна.

Разработка

Для того, чтобы видеть все возможные ошибки во время , настройте следующие параметры в вашем `php.ini`:

```
1 display_errors = On
2 display_startup_errors = On
3 error_reporting = -1
4 log_errors = On
```

> Установка значения в `-1` покажет каждую возможную ошибку, даже если новые уровни и константы будут добавлены в новых версиях PHP. Константа `E_ALL` ведет себя так-же в PHP 5.4. — php.net[92]

Константа уровня ошибок `E_STRICT` была введена в 5.3.0 и не является частью `E_ALL`, как бы то ни было, она стала частью `E_ALL` в 5.4.0 Что это значит? Для вывода всех возможных ошибок в версии 5.3 вам нужно использовать либо `-1` либо `E_ALL | E_STRICT`.

Вывод всех ошибок разнымми версиями PHP

- < 5.3 `-1` or `E_ALL`
- 5.3 `-1` or `E_ALL | E_STRICT`
- > 5.3 `-1` or `E_ALL`

Продакшн

Чтобы спрятать все ошибки вашей среды во время , настройте ваш `php.ini` следующим образом:

[92] http://php.net/manual/function.error-reporting.php

```
display_errors = Off
display_startup_errors = Off
error_reporting = E_ALL
log_errors = On
```

С этими настройками в продакшне, ошибки всё также будут записываться в лог ошибок веб сервера, но не будут показаны пользователю. Для подробной информации о этих настройках, смотрите руководство PHP:

- error_reporting[93]
- display_errors[94]
- display_startup_errors[95]
- log_errors[96]

[93] http://php.net/manual/errorfunc.configuration.php#ini.error-reporting
[94] http://php.net/manual/errorfunc.configuration.php#ini.display-errors
[95] http://php.net/manual/errorfunc.configuration.php#ini.display-startup-errors
[96] http://php.net/manual/errorfunc.configuration.php#ini.log-errors

Тестирование

Написание автоматизированных тестов для вашего кода PHP, считается наилучшей практикой. Автоматизированные тесты являются отличным инструментов для подтверждения того, что ваше приложение не сломается, когда вы внесете изменения или добавите новую функциональность.

Существует несколько различных типов инструментов тестирования (или фреймворков) доступных для PHP, которые используют различные подходы — все, пытаются избежать ручного тестирования и нуждаются в больших командах проверки качества, чтобы убедиться, что изменения не сломают существующую функциональность.

Тесто-ориентированная разработка

Из Википедии[97]:

> Разработка через тестирование (TDD) представляет собой процесс разработки программного обеспечения, который опирается на повторении очень короткого цикла разработки: сперва, разработчик пишет автоматизированные тесты, которые определяют желаемое улучшение или новую функцию, далее производит код, который успешной пройдет этот тест и наконец производит рефактор кода для соответствия со стандартами. Kent Beck, человек которому приписывают статус разработчика или “переоткрывателя” техники, TDD предлагает простую конструкцию, а также вселяет уверенность.

Существует несколько различных типов тестирования, которые вы можете сделать для вашего приложения

Модульное тестирование (Unit Testing)

Модульное тестирование — это подход к программированию, который позволяет удостовериться, что функции, классы и методы работают, как ожидается с момента начала и до конца разработки. Проверяя значения, которые приходят и выходят из различных функций и методом, вы можете быть уверены, что внутренняя логика работает правильно. Используя Внедрение Зависимостей и внедрение классов «макетов» и заглушек, вы можете убедиться, что зависимости используются правильно для большего покрытия тестами.

[97] http://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D1%87%D0%B5%D1%80%D0%B5%D0%B7_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

При создании класса или функции, вы должны создать модульный тест для каждого возможного поведения. На базовом уровне, вы должны убедиться, что ваш код выдает ошибку, если вы отсылаете неправильные аргументы и работает, если вы отсылаете правильные аргументы, соответсвенно. Это поможет убедиться в том, что изменения, которые вы сделаете относительно этого класса или функции позднее не помешают старым работать как ожидалось. Единственная альтернатива этому var_dump() в test.php, который не является способом создания приложений — больших или маленьких.

Еще одно использование модульных тестов — вклад в open source. Если вы можете писать тесты, которые показывают сломанную функциональность, тогда почините её, и покажите, что тест пройден, патчи имеют больше шансов быть принятыми. Если вы запускаете проект, который допускает Pull Request, тогда вы должны указать это в качестве требования.

PHPUnit[98] является фреймворком тестирования стандарта де-факто для написания модульных тестов в PHP приложениях, но также существует несколько альтернатив.

- SimpleTest[99]
- Enhance PHP[100]
- PUnit[101]
- atoum[102]

Интеграционное тестирование

Из Википедии[103]:

> Интеграционное тестирование (иногда называется Интеграция и Тестирование, с аббревиатурой “I&T”) это фаза в тестирование програмнного обеспечения, в котором отдельные модули, комбинируются и тестируются, как группа. Это происходит после модульного тестирования и перед валидационным тестированием. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Многие инструменты, которые могуть быть использованы для модульного тестирования, так-же могут быть использованы для интеграционного тестирования, поскольку используются схожие принципы.

---

[98] http://phpunit.de

[99] http://simpletest.org

[100] http://www.enhance-php.com/

[101] http://punit.smf.me.uk/

[102] https://github.com/atoum/atoum

[103] http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

Функциональное тестирование

Также известное, как подтверждающее тестирование, функциональное тестирование состоит из использования инструментов для создания автоматизированных тестов, которые по-настоящему используют ваше приложение, а не просто проверяют, что отдельные куски (модули) кода ведут себя и могут взаимодействовать правильно. Эти инструменты обычно работают, используя реальные данные и симулируя реальных пользователей приложения.

Инструменты функционального тестирование

- Selenium[104]
- Mink[105]
- Codeception[106] это фреймворк для тестирования (всё-в-одном), включающий инструменты подтверждающего тестирования

Поведенческо-ориентированная разработка

Существует две разновидности Поведенческо-ориентированной разработки (BDD): SpecBDD и StoryBDD. SpecBDD концентрируется на техническом поведении или коде, в то время как StoryBDD концентрируется на деле, будущем поведении или взаимодействии. PHP имеет фреймворки для обоих типов BDD.

Используя StoryBDD, вы пишите читаемые людьми истории, которые объясняют поведение вашего приложения. Эти истории могут быть запущены, как актуальные тесты для вашего приложения. Фреймворк используемый в PHP приложения для StoryBDD - Behat, который вдохновлен проектом для Ruby Cucumber[107] и реализует Gherkin DSL для объяснения особенностей поведения.

Вместе со SpecBDD, вы пишите спецификацию, которая объясняет как ваш код должен себя вести. Вместо тестирования функции или метода, вы объясняете, как эта функция или метод должен себя вести. PHP предлагает фреймворк PHPSpec для данных целей. Этот фреймворк вдохновлен проектом RSpec[108] для Ruby.

Инструменты

- Behat[109], StoryBDD фреймворк для PHP, вдохновленный проектом для Ruby Cucumber[110];
- PHPSpec[111], SpecBDD фреймворк для PHP, вдохновленный проектом для Ruby RSpec[112];

[104] http://seleniumhq.com
[105] http://mink.behat.org
[106] http://codeception.com
[107] http://cukes.info/
[108] http://rspec.info/
[109] http://behat.org/
[110] http://cukes.info/
[111] http://www.phpspec.net/
[112] http://rspec.info/

- Codeception[113] это фреймворк для тестирования (всё-в-одном), использующий принципы BDD;

Дополнительные инструменты тестирования

Помимо индивидуального тестирования и поведенческо-ориентированных фрэймворков, существует ряд общих фреймворков и вспомогательных библиотек, полезных для любого предпочтительного подхода.

Инструменты

- Selenium[114] автоматизационный инструмент для браузера интегрируемый с PHPUnit[115]
- Mockery[116] Mock Object Framework интегрируемый с PHPUnit[117] или PHPSpec[118]

[113] http://www.codeception.com
[114] http://seleniumhq.org/
[115] http://www.phpunit.de/manual/3.1/en/selenium.html
[116] https://github.com/padraic/mockery
[117] http://phpunit.de/
[118] http://www.phpspec.net/

Сервера и развертывание

PHP приложения могут быть развернуты и запущены на продакшн веб-сервере рядом способов.

Платформа, как сервис (PaaS)

PaaS предоставляет системную и сетевую архитектуры, необходимые для запуска PHP приложений в веб. Это означает, как минимум, отсутствие настройки для запуска PHP приложений или PHP фреймворков.

В недавнее время PaaS стал очень популярным методом для развертывания, хостирования и расширения PHP приложений всех размеров. Вы можете найти список провайдеров PHP PaaS в нашей ресурсной секции.

Виртуальный или выделенный сервер

Если вы знакомы с администрированием системы, или заинтересованы в изучении его, виртуальный или выделенный сервер даст вам полный контроль над средой продакшна вашего приложения.

nginx и PHP-FPM

PHP, через встроенный в него менеджер процессов FastCGI (FPM), очень хорошо сочетается с nginx[119], который является легковесным и высокопроизводительным веб-сервером. Он использует меньше памяти, чем Apache и может лучше обрабатывать конкурентные запросы. Это особенно важно на виртуальном сервере, для которого может быть критичен объем используемой памяти.

- Подробнее о nginx[120]
- Подробнее о PHP-FPM[121]
- Подробнее об безопасной установке nginx и PHP-FPM[122]

[119] http://nginx.org
[120] http://nginx.org
[121] http://php.net/manual/ru/install.fpm.php
[122] https://nealpoole.com/blog/2011/04/setting-up-php-fastcgi-and-nginx-dont-trust-the-tutorials-check-your-configuration/

Apache и PHP

PHP и Apache имеют длинную совместную историю. Apache широконастраиваемый и имеет большое количество доступных модулей[123] для расширения функциональности. Это очень популярный выбор для виртуальных хостингов и лёгкой установки PHP фреймворков и приложений с открытым исходным кодом, как WordPress. К сожалению, Apache использует больше ресурсов, чем nginx, и не может выдержать столько же посетителей одновременно.

Apache имеет несколько возможных конфигураций для запуска PHP. Самая популярная и лёгкая для установки prefork MPM[124] вместе с mod_php5. Хотя это не самое эффективное в отношении памяти решение, оно очень просто для установки и использования. Наверное, это лучшее решение, если вы не хотите углубляться в серверное администратирование. Если вы хотите использовать mod_php5, вы обязаны использовать prefork MPM.

Если вы хотите получить больше производительности и стабильности с Apache, тогда вы можете взглянуть на ту-же FPM систему, как в nginx и запустить worker MPM[125] или event MPM[126] , используя mod_fastcgi или mod_fcgid. Эта конфигурация позволит получить существенную экономию в памяти и будет намного быстрее, но потребует больше работы для установки.

- Подробнее на Apache[127]
- Подробнее о Multi-Processing Modules[128]
- Подробнее о mod_fastcgi[129]
- Подробнее о mod_fcgid[130]

Виртуальный хостинг

PHP, благодаря своей популярности, установлен на большом количестве виртуальных хостингов. Очень трудно найти хостинг без установленного PHP, но очень важно убедиться в том, что установлена последняя версия. Виртуальные хостинги позволяют вам и другим разработчикам развертывать веб-сайты на одной машине. Достоинством виртуального хостинга является его низкая цена. Недостатком — то, что вы не знаете, чем будут заниматься ваши соседи; сильно загружая сервер или открывая бреши в безопасности. Если бюджет вашего проекта позволяет избежать использования Виртуальных хостингов, то рассмотрите другие варианты. Martin Fowler*

[123] http://httpd.apache.org/docs/2.4/mod/
[124] http://httpd.apache.org/docs/2.4/mod/prefork.html
[125] http://httpd.apache.org/docs/2.4/mod/worker.html
[126] http://httpd.apache.org/docs/2.4/mod/event.html
[127] http://httpd.apache.org/
[128] http://httpd.apache.org/docs/2.4/mod/mpm_common.html
[129] http://www.fastcgi.com/mod_fastcgi/docs/mod_fastcgi.html
[130] http://httpd.apache.org/mod_fcgid/

Существует разные пути, для осуществления непрерывной интеграции для PHP. Недавно Travis CI[131] закончил великолепную работу по созданию непрерывной интеграции, реально даже для маленьких проектов. Travis CI — сервис непрерывной интеграции для сообщества с открытым исходным кодом. Оно интегрируется с GitHub и предлагает первоклассную поддержку для многих языков, включая PHP.

Для дальнейшего изучения:

- Непрерывная интеграция с Jenkins[132]
- Непрерывная интеграция с Teamcity[133]

[131] https://travis-ci.org/
[132] http://jenkins-ci.org/
[133] http://www.jetbrains.com/teamcity/

Кэширование

PHP очень быстр сам по себе, но такие ресурсоемкие операции, как установка удаленного подключения и, к примеру, загрузка файлов, могут привести к существенному падению скорости работы приложения. К счастью, есть различные инструменты, позволяющие поднять скорость определенных частей вашего приложения, или уменьшить количество запусков этих «тяжелых» задач.

Кэширование байткода

Во время исполнения PHP файла, на низком уровне он сперва компилируется в байткод (или опкод) и только потом исполняется байткод. Если PHP файл не изменен, то байткод будет всегда одинаков. Это значит, что шаг компиляции — пустая трата процессорных ресурсов.

И тут настает время Кэширования байткода. Оно предотвращает избыточные компиляции, сохраняя в памяти байткод и переиспользуя его в последовательных вызовах. Установка подобного кэширования — вопрос пары минут, а скорость вашего приложения поднимется существенно. Нет реальной причины не использовать кэширование.

Популярные системы подобного кэширования:

- APC[134]
- XCache[135]
- Zend Optimizer+[136] (часть Zend Server)
- WinCache[137] (расширение для MS Windows Server)

Кэширование объектов

Бывают моменты, когда вам необходимо кэшировать определенные объекты в вашем коде, такие как данные, которые неразумно получать еще раз из базы данных, когда результат вряд ли изменится. Вы можете использовать ПО для кэширования чтобы сохранить эти кусочки данных в памяти, что позволит чрезвычайно быстро обратиться к ним позже. Если вы сохраните эти объекты в хранилище данных и после получите их и выдадите

[134] http://php.net/manual/ru/book.apc.php
[135] http://xcache.lighttpd.net/
[136] http://www.zend.com/products/server/
[137] http://www.iis.net/download/wincacheforphp

напрямую из кэша для некоторых запросов, вы можете получить существенное улучшение производительности и уменьшение нагрузки на сервер базы данных.

Множество популярных решений для кэширования байткода также дают вам кэшировать данные, поэтому нет причин, чтобы не воспользоваться ими. APC, XCache, и WinCache предоставляют API для сохранения данных из вашего PHP кода в свой кэш в памяти.

Самыми популярными системами кэширования объектов являются APC и memcached. APC — идеальный выбор для кэширования объектов, он включает простой API для добавления данных в кэш память и при этом очень просто устанавливается и используется. Единственное существующее ограничение APC состоит в том, что он привязан к серверу на котором установлен. Memcached, напротив, устанавливается как отдельный сервис, и к нему можно получить доступ по сети, что позволяет хранить объекты в очень быстром централизованном хранилище данных и множество других систем могут получать эти данные из него.

Учтите, если PHP запущен как (Fast-)CGI приложение внутри вашего веб-сервера, то каждый PHP процесс будет иметь собственный кэш, например APC данные не будут расшарены между вашими процессами. В этом случае имеет смысл подумать об использовании вместо него memcached, так как он не ограничен процессом PHP.

В сетевой конфигурации APC зачастую выигрывает у memcached в плане скорости доступа, но memcached обладает возможностью масштабироваться быстрее и более широко. Если иметь несколько серверов для обслуживания приложения не входит в ваши планы или вам не нужны дополнительные возможности memcached, тогда APC будет лучшим выбором для кэширования объектов.

Пример использования APC:

```
1  <?php
2  // проверяем есть ли данные сохраненный как 'expensive_data' в кэше
3  $data = apc_fetch('expensive_data');
4  if ($data === false) {
5      // данных нет в кэше; сохраняем результат вызова функции для дальнейшего испо\
6  льзования
7      apc_add('expensive_data', $data = get_expensive_data());
8  }
9
10 print_r($data);
```

Подробнее о популярных системах кэширования объектов:

- Функции APC[138]

[138] http://php.net/manual/ru/ref.apc.php

- Memcached[139]
- Redis[140]
- XCache API[141]
- Функции WinCache[142]

[139] http://memcached.org/
[140] http://redis.io/
[141] http://xcache.lighttpd.net/wiki/XcacheApi
[142] http://www.php.net/manual/ru/ref.wincache.php

Ресурсы

Из источника

- Официальный сайт PHP[143]
- Документация PHP[144]

Их следует читать в твиттере

- Rasmus Lerdorf[145]
- Fabien Potencier[146]
- Derick Rethans[147]
- Chris Shiflett[148]
- Sebastian Bergmann[149]
- Matthew Weier O'Phinney[150]
- Pádraic Brady[151]
- Anthony Ferrara[152]
- Nikita Popov[153]

Наставничество

- phpmentoring.org[154] - Формальное, контакт к контакту сообщество наставников PHP.

[143] http://php.net/
[144] http://php.net/docs.php
[145] http://twitter.com/rasmus
[146] http://twitter.com/fabpot
[147] http://twitter.com/derickr
[148] http://twitter.com/shiflett
[149] http://twitter.com/s_bergmann
[150] http://twitter.com/weierophinney
[151] http://twitter.com/padraicb
[152] http://twitter.com/ircmaxell
[153] http://twitter.com/nikita_ppv
[154] http://phpmentoring.org/

PHP PaaS поставщики

- PagodaBox[155]
- AppFog[156]
- Heroku[157] (PHP поддержка не документирована но основана на стабильном сотрудничестве с Facebook link[158])
- fortrabbit[159]
- Engine Yard Orchestra PHP Platform[160]
- Red Hat OpenShift Platform[161]
- dotCloud[162]
- AWS Elastic Beanstalk[163]
- cloudControl[164]
- Windows Azure[165]
- Zend Developer Cloud[166]

Фреймворки

Вместо того, чтобы заново изобретать колесо, многие PHP разработчики используют для построения веб-приложений фреймворки. Фреймворки позволяют абстрагироваться от низкоуровневой разработки и предоставляют удобный интерфейс для выполнения общих задач.

Не обязательно использовать фреймворк в каждом своем проекте. Иногда чистый PHP является оптимальным путём, но, если вам нужен фреймворк, то выберите наиболее подходящий вам тип:

- Микрофреймворки
- Фреймворки «всё-в-одном»
- Компонентные фреймворки

[155] https://pagodabox.com/
[156] https://appfog.com/
[157] https://heroku.com
[158] http://net.tutsplus.com/tutorials/php/quick-tip-deploy-php-to-heroku-in-seconds/
[159] http://fortrabbit.com/
[160] http://www.engineyard.com/products/orchestra/
[161] http://www.redhat.com/products/cloud-computing/openshift/
[162] http://docs.dotcloud.com/services/php/
[163] http://aws.amazon.com/elasticbeanstalk/
[164] https://www.cloudcontrol.com/
[165] http://www.windowsazure.com/
[166] http://www.phpcloud.com/develop

Микрофреймворки, в большинстве, предоставляют оболочку для маршрутизации HTTP запросов к контроллеру, методу и т.д., так быстро, как это возможно, и иногда поставляются с несколькими библиотеками для помощи разработчикам, как например простая оболочка базы данных и подобного. Они часто используются для построения удаленных HTTP сервисов.

Многие фреймворки добавляют значительное количество возможностей поверх того, что доступно в микрофреймворках, такие известны, как Фреймворки «всё-в-одном». Они зачастую поставляются с ORM, пакетами Аутентификации и т.д..

Компонентно-ориентированные фреймворки являются коллекциями специализированных и узко-специализированных библиотек. Разрозненные компонентно-ориентированный фреймворки, могут быть использованы для создания микро- или «всё-в-одном» фреймворка.

- Популярные PHP фреймворки[167]

Компоненты

Как упоминалось выше, «Компоненты» являются еще одним подходом к общей цели создания, распространения и внедрения кода. Существуют различные репозитории для компонентов, основные два:

- Packagist[168]
- PEAR[169]

Оба содержат инструменты командной строки для облегчения процедур установки и обновления, что более подробно объясняется в разделе Управление зависимостями[170].

Также существуют компонент-ориентированные фреймворки, которые позволяют вам использовать их компоненты с минимальными (или нет) требованиями. Например, вы можете использовать Валидационный пакет FuelPHP[171], без нужды в использовании фреймворка FuelPHP. Эти проекты по существу являются еще одним репозиторием для повторно используемых компонентов:

- Aura[172]
- FuelPHP (2.0 only)[173]
- Laravel's "Illuminate Components"[174]
- Symfony Components[175]

[167] https://github.com/codeguy/php-the-right-way/wiki/Frameworks
[168] /#composer_and_packagist
[169] /#pear
[170] /#dependency_management
[171] https://github.com/fuelphp/validation
[172] http://auraphp.github.com/
[173] https://github.com/fuelphp
[174] https://github.com/illuminate
[175] http://symfony.com/doc/current/components/index.html

Сообщество

Сообщество PHP так же разнообразно, как и велик сам язык. Члены сообщества готовы помочь начинающим PHP программистам. Подумайте о вступлении в вашу местную группу PHP пользователей (PUG — PHP User Group) или об участии в больших PHP конференциях для изучения лучших практик. Вы можете пообщаться в IRC в канале #phprc на irc.freenode.com[176] и зафолловить твиттер аккаунт @phpc[177] . Знакомьтесь с новыми разработчиками, изучайте новые темы и, помимо всего этого, заводите новых друзей! Также полезны сообщества Google+ PHP Сообщество разработчиков[178] и StackOverflow[179].

Официальный календарь событий PHP[180]

Пользовательские группы PHP

Если вы живёте в большом городе, есть шанс, что в нем существует группа PHP пользователей. Если же таковой группы нет в официальном списке PUG, вы можете легко найти её в поиске Google[181] или Meetup.com[182]. Если вы живёте в маленьком городе, в котором нет своей PUG, то создайте свою!

Подробнее о группах пользователей PHP на PHP Wiki[183]

Конференции PHP

Сообщество PHP также поддерживает региональные и национальные конференции в разных странах мира. Широко известные члены сообщества PHP часто выступают на этих крупных событиях, так что это хорошая возможность непосредственного обучения от лидеров индустрии.

Найти конференцию PHP[184]

---

[176] http://webchat.freenode.net/
[177] https://twitter.com/phpc
[178] https://plus.google.com/u/0/communities/104245651975268426012
[179] http://stackoverflow.com/questions/tagged/php
[180] http://www.php.net/cal.php
[181] https://www.google.com/search?q=php+user+group+near+me
[182] http://www.meetup.com/find/
[183] https://wiki.php.net/usergroups
[184] http://php.net/conferences/index.php

Советы по повышению эффективности PHP

Author: Eric Higgins, Google Webmaster

Recommended experience: Beginner to intermediate PHP knowledge

PHP - очень популярный язык программирования, используемый на многих веб сайтах. В этой статье, мы надеемся помочь вам улучшить производительность ваших PHP скриптов с изменениями, которые вы можете внести очень быстро и безболезненно. Учтите, что производительность может сильно меняться в зависимости от версии PHP на вашем веб сервере и в целом от вашего кода.

Профилируйте ваш код для обнаружения узких мест

Изречение Хора гласит, что преждевременная оптимизация - это корень всех зол, важно это учитывать, делая ваши веб сайты быстрее. Перед изменением вашего кода, вам нужно определить, что приводит к замедлению. Вы можете прочитать это руководство, а так-же многие другие по теме оптимизации PHP, в то время, как ваши проблемы могут зависить от базы данных или сети. [Профилируя ваш PHP код][http://www.google.com/webhp?#q=profiling+php], вы можете попробывать определить узкие места.

Обновите ваш PHP

Команда разработчиков, которая поддерживает ядро PHP, внесло большое число важных исправлений производительности за последние несколько лет. Если на ваш веб сервер всё ещё работает со старыми версиями, как PHP 3 или PHP 4, то прежде всего вам нужно подумать об обновлении PHP, прежде чем вы попытаетесь оптимизировать ваш код.

- Миграция от PHP 4 до PHP 5.0.x[185]
- Миграция от PHP 5.0.x до PHP 5.1.x[186]
- Миграция от PHP 5.1.x до PHP 5.2.x[187]

[185] http://www.php.net/manual/migration5.php
[186] http://www.php.net/manual/migration51.php
[187] http://www.php.net/manual/migration52.php

Кэширование

Использование модули кэширования, таких как Memcache, или шаблонизаторов, которые поддерживают кэширование, таких как Smarty, может улучшить производительность ваших веб сайтов с помощью кэширования результатов запросов к базам данных или отрендеренных страниц.

Использование буферизации вывода

PHP использует буффер памяти, чтобы сохранить все данные, которые ваш скрипт пытается вывести. Этот буффер может сделать ваши страницы визуально медленными, поскольку вашим пользователям придётся ждать пока буффер заполнится перед тем, как он отправит какие-либо данные. К счастью, вы можете сделать некоторые изменения, которые заставят PHP вернуть выводной буффер раньше, а так-же чаще, что сделает ваш сайт быстрее для ваших пользователей.

- Контроль выводного буффера[188]

Избегайте написания наивных геттеров и сеттеров

Когда вы пишите классы в PHP, вы можете сохранить время и повысить скорость ваших скриптов, работая со свойствами объекта напрямую, вместо написания наивных сеттеров и геттеров. В следующем примере, класс dog использует методы `setName()` и `getName()` для доступа к свойству `name`.

```
1  <?php
2
3  class dog {
4    public $name = '';
5
6    public function setName($name) {
7      $this->name = $name;
8    }
9
10   public function getName() {
11     return $this->name;
12   }
13 }
```

[188] http://www.php.net/manual/book.outcontrol.php

Имейте в виду, что `setName()` и `getName()` не делают большего, чем сохраняют и возвращают свойство `name`, соответственно.

```
<?php

$rover = new dog();
$rover->setName('rover');
echo $rover->getName();
```

Установка и вызов свойства `name` напрямую может позволить скрипту работать до 100% быстрее, так-же, как и сократить время разработки.

```
<?php

$rover = new dog();
$rover->name = 'rover';
echo $rover->name;
```

Не копируйте переменные без причины

Иногда новички в PHP пытаются сделать их код “чище”, копируя предустановленные переменные в переменные с более короткими именами, перед работой с ними. Это приводит к увеличенному потреблению памяти(когда переменные переназначаются), а так-же к замедлению скриптов. В следующем примере, что происходит если пользователь вставил 512KB символов в поле textarea. Это исполнение приведет к потреблению примерно 1MB памяти.

```
<?php

$description = strip_tags($_POST['description']);
echo $description;
```

Нет причины копировать переменную в примере выше. Вы можете просто сделать эту операцию одной линией и избежать сверх-потребление памяти:

```
<?php

echo strip_tags($_POST['description']);
```

Избегайте SQL запросов в цикле

Частая ошибка - размещение SQL запросов в цикле. Это приводит к нескольким циклам запросов к базе данных, и существенно замедляет скрипты. В примере ниже, вы можете изменить цикл, чтобы он строил одиночный SQL запрос и вставлял всех пользователей за один раз.

```
<?php

foreach ($userList as $user) {
  $query = 'INSERT INTO users (first_name,last_name) VALUES("' . $user['first_nam\
e'] . '", "' . $user['last_name'] . '")';
  mysql_query($query);
}
```

Выполняет запрос:

```
INSERT INTO users (first_name,last_name) VALUES("John", "Doe")
```

Вместо использования цикла, вы можете скомбинировать данные в одиночный запрос к базе.

```
<?php

$userData = array();
foreach ($userList as $user) {
    $userData[] = '("' . $user['first_name'] . '", "' . $user['last_name'] . '")';
}
$query = 'INSERT INTO users (first_name,last_name) VALUES' . implode(',', $userDa\
ta);
mysql_query($query);
```

Выполняет запрос:

```
INSERT INTO users (first_name,last_name) VALUES("John", "Doe"),("Jane", "Doe")...
```

Синтаксис MySQL INSERT[189]

Дополнительные руководства

PHP Memcache модуль[190] Шаблонизатор Smarty[191]

[189] http://dev.mysql.com/doc/mysql/ru/insert.html
[190] http://www.php.net/memcache
[191] http://www.smarty.net/

Основы

Операторы сравнения

Операторы сравнения часто упускается из виду аспект PHP, который может привести ко многим неожиданным результатам. Одна из таких проблем возникает из-за строгого сравнения (сравнение логических значений в виде целых чисел).

```
<?php
$a = 5;   // 5 как целое число (integer)

var_dump($a == 5);       // Сравниваются значения; Вернёт true
var_dump($a == '5');     // Сравниваются значения (игнорируя типы); Вернёт true
var_dump($a === 5);      // Сравниваются типы и значения (integer vs. integer); B\
ернёт true
var_dump($a === '5');    // Сравниваются типы и значения (integer vs. string); Be\
рнёт false

/**
 * Строгое сравнение
 */
if (strpos('testing', 'test')) {    // 'test' находится в 0 позиции, результатом \
будет 'false'
    // Ваш код...
}

vs.

if (strpos('testing', 'test') !== false) {    // Результатом будет 'true', т.к. т\
ут строгое сравнение (0 !== false)
    // Ваш код...
}
```

- Операторы сравнения[192]
- Таблица сравнения типов[193]

[192] http://php.net/manual/ru/language.operators.comparison.php
[193] http://php.net/manual/ru/types.comparisons.php

Условные операторы

Оператор “If”

При использовании операторов ‘if/else’ внутри функции или класса, существует распространенное заблуждение, что ‘else’ должен быть использован при возврате результатов выполнения. Если условие не выполняется и при этом возвращается значение (return $value), то использование ‘else’ может быть спорным.

```
<?php
function test($a)
{
    if ($a) {
        return true;
    } else {
        return false;
    }
}

vs.

function test($a)
{
    if ($a) {
        return true;
    }
    return false;    // else использовать не обязательно
}
```

- Оператор “If”[194]

Оператор “Switch”

Оператор “Switch” является отличным способом, чтобы не использовать много операторов “if” с использованием “elseif”, но необходимо знать некоторые вещи:

- Оператор “Switch” сравнивает только значения, но не типы данных (равнозначно логической операции ‘==’)
- Этот оператор сравнивает выражение с каждым значением, пока не найдёт нужное. Если не нашёл, использует “default” (если определён)

[194] http://php.net/manual/ru/control-structures.if.php

- Без использования ‘break’, выражение будет сравниваться со всеми значениями по-порядку, пока не встретит “break” или “return”
- Если вы используете для возврата результата ‘return’ то ‘break’ можно опустить.

```
<?php
$answer = test(2);    // Этот код выберет 'case 2' и 'case 3'.

function test($a)
{
    switch ($a) {
        case 1:
            // Код...
            break;             // Прекратит выполнение switch тут если $a == 1, т\
.к. используется 'break'
        case 2:
            // Код...         // Код без 'break', поэтому будет выполнено сравнен\
ие с 'case 3'
        case 3:
            // Код...
            return $result;    // within a function, 'return' will end the functi\
on
        default:
            // Код...
            return $error;
    }
}
```

- Оператор Switch[195]
- PHP switch[196]

Глобальное пространство имён

Когда вы используете пространство имён (namespaces), вы можете обнаружить, что некоторые функции вам скрыты, недоступны. Что исправить это, указываете что это глобальная функция, использовав обратную косую черту ‘/’ перед именем функции.

[195] http://php.net/manual/ru/control-structures.switch.php
[196] http://phpswitch.com/

```php
<?php
namespace phptherightway;

function fopen()
{
    $file = \fopen();    // Функция имеет имя такое же как и глобальная функция '\
fopen'.
                         // Чтобы их различать используйте в глобальных '\'.
}

function array()
{
    $iterator = new \ArrayIterator();   // ArrayIterator внешний класс. Если испо\
льзовать его без '/'
                                        // то интерпретатор PHP будет пытаться на\
йти его в пространстве 'phptherightway'.
}
```

- Глобальное пространство имён[197]
- Правила именования[198]

Строки

Конкатенация (сцепление)

- Если ваша строка на одной строке занимает не рекомендуемую длину (120 символов), используйте конкатенацию.
- Для удобства чтения лучше использовать конкатенацию, чем операторы присваивания.
- Если используете новую строку при сцеплении строк, делайте относительно оригинальной строки отступы.

[197] http://php.net/manual/ru/language.namespaces.global.php
[198] http://php.net/manual/ru/userlandnaming.rules.php

```php
<?php
$a  = 'Multi-line example';    // Оператор сцепления строк (.=)
$a .= "\n";
$a .= 'of what not to do';

vs.

$a = 'Multi-line example'      // Оператор объединения (.)
    . "\n"                     // используя новые строки.
    . 'of what to do';
```

- Операторы для строк[199]

Тип строки.

Строки постоянный спутник для сообщества PHP, надеемся, эта статья объяснит различия между строковыми типами и их преимущества/использование.

Одиночные кавычки

Часто самый быстрый способ отделить строку - эта использовать одинарные кавычки. Скорость заключается в том, что PHP не анализирует строку (не ищет в ней переменные). Одинарные кавычки лучше всего подходят для:

- строк, которые не нужно анализировать,
- имён переменных, которые нужно написать как текст.

```php
<?php
echo 'Посмотрите как прекрасна моя симпатичная строка.';    // анализ этой строке\
 не нужен.

/**
 * Выведет:
 *
 * Посмотрите как прекрасна моя симпотичная строка.
 */
```

- Одиночные ковычки[200]

[199] http://php.net/manual/ru/language.operators.string.php
[200] http://www.php.net/manual/ru/language.types.string.php#language.types.string.syntax.single

Двойные кавычки

Двойные кавычки, как швейцарский нож. Они лучше всего используются для:

- Escaped strings
- строк с использованием переменных,
- использования Condensing multi-line concatenation, для удобство просмотра кода.

```
<?php
echo 'phptherightway is ' . $adjective . '.'     // эта строка использует multipl\
e concatenating для
    . "\n"                                       // отделения переменных от строк\
и.
    . 'I love learning' . $code . '!';

vs.

echo "phptherightway is $adjective.\n I love learning $code!"  // А тут использую\
тся двойные кавычки.
```

При использовании двойных кавычки часто бывает, что переменную нужно использовать в чуть изменнёном виде. Но при анализе строки PHP не сможет определить что это переменная. Для решения этой проблемы, оберните переменную в фигурные скобки.

```
<?php
$juice = 'plum';
echo "I drank some juice made of $juices";    // $juices не определена

vs.

$juice = 'plum';
echo "I drank some juice made of {$juice}s";    // $juice будет анализирована.

/**
 * Комплексные переменные также оборачивайте в фигурные скобки.
 */

$juice = array('apple', 'orange', 'plum');
echo "I drank some juice made of {$juice[1]}s";   // $juice[1] будет анализирован\
а.
```

- Двойные кавычки[201]

[201] http://www.php.net/manual/ru/language.types.string.php#language.types.string.syntax.double

Nowdoc синтаксис

Nowdoc синтаксис был введён в php 5.3 и используется также как и одинарные кавычки, только для использования нескольких строк не нужно использовать объединение.

```
<?php
$str = <<<'EOD'            // Инициализируется <<<
Example of string
spanning multiple lines
using nowdoc syntax.
$a does not parse.
EOD;                       // закрывается 'EOD' (должен быть на новой строке и б\
ез отступов).

/**
 * Вывод:
 *
 * Example of string
 * spanning multiple lines
 * using nowdoc syntax.
 * $a does not parse.
 */
```

- Nowdoc[202]

Heredoc синтаксис

Heredoc синтаксис работает также как и двойные кавычки, но также для использования нескольких строк не нужно использовать объединение.

```
<?php
$a = 'Variables';

$str = <<<EOD              // Инициализируется <<<
Example of string
spanning multiple lines
using heredoc syntax.
$a are parsed.
EOD;                       // закрывается  'EOD' (должен быть на новой строке и \
без отступов).

```

[202] http://www.php.net/manual/ru/language.types.string.php#language.types.string.syntax.nowdoc

```
/**
 * Вывод:
 *
 * Example of string
 * spanning multiple lines
 * using heredoc syntax.
 * Variables are parsed.
 */
```

- Heredoc[203]

Тернарный оператор

Тернарный оператор (‘(expr1) ? (expr2) : (expr3)’) используется для удобства объединения кода в одну строку, но часто избыточен. Хоть он может быть вложенным, рекомендуется его использовать один на строку.

```
<?php
$a = 5;
echo ($a == 5) ? 'yay' : 'nay';

vs.

//Вложения
$b = 10;
echo ($a) ? ($a == 5) ? 'yay' : 'nay' : ($b == 10) ? 'excessive' : ':(';    // Вл\
ожения трудно читаемы.
```

Используя ‘return’ будьте внимательны:.

```
<?php
$a = 5;
echo ($a == 5) ? return true : return false;    // этот пример будет выдавать соо\
бщение об ошибке

vs.

$a = 5;
return ($a == 5) ? 'yay' : 'nope';    // этот пример вернёт 'yay'
```

- Тернарный оператор[204]

[203] http://www.php.net/manual/ru/language.types.string.php#language.types.string.syntax.heredoc
[204] http://php.net/manual/ru/language.operators.comparison.php

Объявление переменных

Время от времени разработчики пытаются сделать свой код “чище” использую предопределённые переменные. Обычно это только ведёт к увеличению используемой памяти. Для примера сообщите какой-нибудь переменной строку размером 1мб, в результате вы скопируете это дважды.

```
<?php
$about = 'Очень большой текст';     // будет использоваться 2MB памяти
echo $about;

vs.

echo 'Очень большой текст';         // а тут всего лишь 1MB
```

- Советы по оптимизации (en)[205]

[205] https://developers.google.com/speed/articles/optimizing-php

Функциональное программирование в PHP

PHP поддерживает перво-классные функции, это значит, что функция может быть применена к переменной. И определенные пользователем, и встроенные функции могут быть применены к переменной и вызываться динамически. Функции могут быть переданы, как аргумент к другой функции (эта особенность называется функцией высшего порядка), а также функция может возвращать другую функцию.

Рекурсия, особенность, которая позволяет функции вызывать саму себя, это поддерживается языком, но большая часть кода PHP фокусируется на итерации.

Новые анонимные функции(с поддержкой для замыканий) присутствую с PHP 5.3 (2009).

В PHP 5.4 добавлена возможность связывать замыкание с областью видимости объекта, а так-же улучшена поддержка callables(всё что может быть вызвано), так что они могут быть использованы наравне с анонимными функциями практически во всех случаях.

Наиболее распространным использованием функций высшего порядка, является реализация паттерна стратегия. Встроенная функция `array_filter` спрашивает одинаково, как входной массив(данные), так и функцию (стратегия или callback), используемая, как фильтр для каждого элемента массива.

```
<?php
$input = array(1, 2, 3, 4, 5, 6);

// Создает новую анонимную функцию и присваевает её к переменной
$filter_even = function($item) {
    return ($item % 2) == 0;
};

// Встроенная функция принимает, как массив, так и функцию
$output = array_filter($input, $filter_even);

// Функции не обязательно нужно быть присвоенной к переменной. Это так-же работае\
т:
$output = array_filter($input, function($item) {
    return ($item % 2) == 0;
});

print_r($output);
```

Замыкания - это анонимные функции, которые могут получить доступ к переменные, импортированным извне области видимости, без использования любых глобальных переменных. Теоретически, замыкание - функция с закрытыми некоторыми аргументами (например фиксированными) окружением, когда они объявлены. Замыкания могут обойти ограничения области видимости переменных, чистым способом.

В следующем примере, мы используем замыкания для объявления функции, возвращающей одну функцию фильтр для `array_filter` из семьи фильтрирующих функций.

```
1  <?php
2  /**
3   * Создает анонимную функцию фильтр позволяющую значение > $min
4   *
5   * Возвращает один фильтр типа "больше чем n".
6   */
7  function criteria_greater_than($min)
8  {
9      return function($item) use ($min) {
10         return $item > $min;
11     };
12 }
13
14 $input = array(1, 2, 3, 4, 5, 6);
15
16 // Используем array_filter на вводе, с указанной функцией фильтром
17 $output = array_filter($input, criteria_greater_than(3));
18
19 print_r($output); // значения > 3
```

Каждая функция фильтр в семье, принимает только те элементы, значение которых больше, определенного минимального значения. Одиночный фильтр возвращается с помощью замыкания `criteria_greater_than` с аргументом `$min` с закрытым значением в области видимости (даётся, как аргумент, когда `criteria_greater_than` вызывается).

Ранее связывание, используется по умолчания, для импортирования переменной `$min` для передачи переменной в созданную функцию. Для настоящих замыканий с поздним связыванием обязательно следует использовать ссылку при импортировании. Представьте себе библиотеки шаблонизации или валидации ввода, где замыкания объявлены для захвата переменных в области видимости и доступа к ним позже, когда анонимные функции исполняются.

- Подробнее об Анонимных функциях[206]

[206] http://www.php.net/manual/ru/functions.anonymous.php

- Больше деталей в Closures RFC[207]
- Узнать о динамически вызываемых функция с `call_user_func_array`[208]

[207] https://wiki.php.net/rfc/closures
[208] http://php.net/manual/ru/function.call-user-func-array.php

Шаблоны проектирования

Есть множество способов структурировать и проектировать код веб приложения и вы можете приложить максимум усилий или немного подумать, чтобы понять какая вам нравиться архитектура. В любом случае это хорошая идея использовать общие шаблоны проектирования, потому что это делает код для других более понятным и легко используемым.

- Архитектура программного обеспечения на Википедии[209]
- Шаблон проектирования на Википедии[210]

Фабрика (англ. Factory)

Этот шаблон является одним из часто используемых. В нём класс просто создаёт объект, который вам необходим. Рассмотрим следующий пример шаблон фабрики:

```
1  <?php
2  class Automobile
3  {
4      private $vehicle_make;
5      private $vehicle_model;
6
7      public function __construct($make, $model)
8      {
9          $this->vehicle_make = $make;
10         $this->vehicle_model = $model;
11     }
12
13     public function get_make_and_model()
14     {
15         return $this->vehicle_make . ' ' . $this->vehicle_model;
16     }
17 }
18
```

[209] https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F

[210] https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F

```php
class AutomobileFactory
{
    public static function create($make, $model)
    {
        return new Automobile($make, $model);
    }
}

// фабрика создаёт автомобильные объекты
$veyron = AutomobileFactory::create('Bugatti', 'Veyron');

print_r($veyron->get_make_and_model()); // выведет "Bugatti Veyron"
```

Этот код создаст объект “Автомобиль”, используя фабрику. Сделав так вы получите два преимущества: во-первых, если вам в дальнейшем нужно изменить, переименовать или заменить класс Automobile, то вы легко это сделаете, просто изменив код в фабрике AutomobileFactory, вместо того, чтобы менять его во всех местах проекта, которые используют класс Automobile; во-вторых, если вам нужно при создании объекта выполнять какие-то операции с этим объектом, то вы можете описать эти операции в фабрике, вместо того чтобы каждый раз их описывать при создании нового объекта.

Использовать шаблон Фабрика не всегда необходимо (или слишком мудро). В примере, что выше код настолько прост, что использование фабрики добавляет не нужную сложность. Однако если вы делаете довольно большой и сложный проект, то использование шаблона Фабрика позволит вам избежать многих хлопот.

- [Фабрика на Википедии](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B1%D1%80%D0%B8%D1%8
  %D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D

Одиночка (англ. Singleton)

При создании веб приложения, часто требуется концептуально и архитектурно предоставить доступ только к одному экземпляру определённого класса. Шаблон Одиночка позволяет это сделать.

```php
<?php
class Singleton
{
    /**
     * Returns the *Singleton* instance of this class.
     *
     * @staticvar Singleton $instance The *Singleton* instances of this class.
     *
     * @return Singleton The *Singleton* instance.
     */
    public static function getInstance()
    {
        static $instance = null;
        if (null === $instance) {
            $instance = new static;
        }

        return $instance;
    }

    /**
     * Protected constructor to prevent creating a new instance of the
     * *Singleton* via the `new` operator from outside of this class.
     */
    protected function __construct()
    {
    }

    /**
     * Private clone method to prevent cloning of the instance of the
     * *Singleton* instance.
     *
     * @return void
     */
    private function __clone()
    {
    }

    /**
     * Private unserialize method to prevent unserializing of the *Singleton*
     * instance.
     *
```

```
43          * @return void
44          */
45         private function __wakeup()
46         {
47         }
48 }
49
50 class SingletonChild extends Singleton
51 {
52 }
53
54 $obj = Singleton::getInstance();
55 \var_dump($obj === Singleton::getInstance());             // bool(true)
56
57 $anotherObj = SingletonChild::getInstance();
58 \var_dump($anotherObj === Singleton::getInstance());      // bool(false)
59
60 \var_dump($anotherObj === SingletonChild::getInstance()); // bool(true)
```

Этот код реализует данный шаблон, используя *статические* переменные[211] и статический метод `getInstance()`. Обратите внимание на следующее:

- Конструктор `__construct`[212] сделан защищённым (protected), чтобы запретить создание нового объекта с помощью оператора `new`.
- Магический метод `__clone`[213] определён как частный (private), чтобы предотвратить клонирование экземпляра класса с помощью `clone`[214].
- Магический метод `__wakeup`[215] определён как частный (private), чтобы предотвратить десериализации экземпляра класса через глобальную функцию `\unserialize()`[216].
- Новый экземпляр создаётся с помощью позднего статического связывания[217] в статическом методе `getInstance()` с ключевым словом `static`. This allows the subclassing of the class `Singleton` in the example.

Шаблон Одиночка полезен тогда, когда нужно быть уверенным, что экземпляр класса только один во жизненном цикле запроса для веб приложения. Обычно это происходит, когда имеется глобальный объект (например Configuration класс) или общий ресурс (например очередь событий).

[211] http://php.net/language.variables.scope#language.variables.scope.static
[212] http://php.net/language.oop5.decon#object.construct
[213] http://php.net/language.oop5.cloning#object.clone
[214] http://php.net/language.oop5.cloning
[215] http://php.net/language.oop5.magic#object.wakeup
[216] http://php.net/function.unserialize
[217] http://php.net/language.oop5.late-static-bindings

Вы должны быть осторожными, используя этот шаблон, поскольку по своей природе он вводит глобальное утверждение экземпляра в приложении, понижая тем самым тестируемость. В большинстве случаев внедрение зависимостей могут (должны) использоваться вместо Singleton класса. Используя внедрение зависимости, означает, что мы не вводим ненужных соединений в дизайн наших приложения, а объект, используя общий или глобальный ресурс, не требует знания конкретного класса.

- Шаблон Одиночка на Википедии[218]

Фронт-контролер (англ. Front Controller)

Шаблон Фронт-контроллер использует единую точку входа для приложения (например, index.php), которая обрабатывает все запросы. Код этого шаблона отвечает за загрузку всех зависимостей, обработку и отправку запроса в браузере. Фронт-контроллер может быть полезным, поскольку способствует модульному коду и предоставляет центральное место, в которое можно внедрить код для каждого запроса (например, санитарная обработка входных данных).

- Фронт-контролер на Википедии[219]

Модель-представление-контроллер (англ. Model-View-Controller)

Модель-представление-контроллер (далее MVC) шаблон из то же серии, что и HMVC, MVVM. MVC позволяет разбить код приложения на логические объекты, которые предназначены для под конкретные задачи. Модель служит как слой к доступу данных и возвращает их в том формате, который необходим приложению. Контроллеры обрабатывают запросы, обрабатывают данные, полученные из модели, и загружают представления, посылая в него ответ. Представления содержат шаблоны (markup, xml и другие), которые отправляются в браузер.

MVC является наиболее распространенным архитектурным шаблоном, который используется в популярных PHP фреймворков[220].

Больше информации по подобным шаблонам вы можете подчеркнуть в следующих ссылках:

- MVC[221]

[218] https://ru.wikipedia.org/wiki/Singleton
[219] https://en.wikipedia.org/wiki/Front_Controller_pattern
[220] https://github.com/codeguy/php-the-right-way/wiki/Frameworks
[221] https://ru.wikipedia.org/wiki/Model-View-Controller

- HMVC[222]
- MVVM[223]

[222] https://ru.wikipedia.org/wiki/HMVC
[223] https://ru.wikipedia.org/wiki/Model-View-ViewModel