PHP之道

极客学院出版

前言

目前网络上充斥着大量的过时资讯，让 PHP 新手误入歧途，并且传播着错误的实践以及不安全的代码。PHP 之道收集了现有的 PHP 最佳实践、编码规范和权威学习指南，方便 PHP 开发者阅读和查找。

使用 PHP 没有规范化的方式。本网站主要是向 PHP 新手介绍一些他们没有发现或者是太晚发现的主题， 或是经验丰富的专业人士已经实践已久的做法提供一些新想法。本网站也不会告诉您应该使用什么样的工具，而是提供多种选择的建议，并尽可能地说明方法及用法上的差异。

当有更多有用的资讯以及范例时，此文件会随着相关技术的发展而持续更新。

版本信息

语言/框架	版本信息
PHP	5.6

翻译

PHP 之道 已经翻译成多种语言：

- English
- Bulgarian
- Chinese (Simplified)
- Chinese (Traditional)
- French
- German
- Indonesian
- Italian
- Japanese
- Korean
- Persian

- Polish
- Portuguese
- Romanian
- Russian
- Serbian
- Slovenian
- Spanish
- Thai
- Turkish
- Ukrainian

目录

1

入门指南

使用当前稳定版本（5.6）

如果你刚开始学习 PHP，请使用最新的稳定版本 PHP 5.6。PHP 近年来有了巨大的改进，增加了许多强大的 新特性。虽然 5.2 和 5.6 之间增加的版本号似乎很小， 但它代表了 *重大的* 改进。如果你想查找一个函数及其用法，可以去官方手册 php.net 中查找。

内置的 web 服务器

PHP 5.4 之后，你可以不用安装和配置功能齐全的 Web 服务器，就可以开始学习 PHP。 要启动内置的 Web 服务器，需要从你的命令行终端进入项目的 Web 根目录，执行下面的命令：

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

- 了解更多内置的命令行服务器

Mac 安装

OSX 系统会预装 PHP， 只是一般情况下版本会比最新稳定版低一些。目前 Lion 是 5.3.10， Mavericks 是 5.4.17， Yosemite 则是 5.5.9， 但在 PHP 5.6 出来之后， 这些往往是不够的。

这里有许多方式在 OS X 上安装 PHP 。

通过 Homebrew 安装 PHP

Homebrew 是一个强大的 OS X 专用包管理器， 它可以帮助你轻松的安装 PHP 和各种扩展。 Homebrew PHP 是一个包含与 PHP 相关的 Formulae，能让你通过 homebrew 安装 PHP 的仓库。

也就是说，你可以通过 `brew install` 命令安装 `php53`， `php54`， `php55` 或者 `php56` ，并且通过修改 `PATH` 变量来切换各个版本。或者你也可以使用 brew-php-switcher 来自动切换。

通过 Macports 安装 PHP

MacPorts 是一个开源的，社区发起的项目，它的目的在于设计一个易于使用的系统，方便编译，安装以及升级 OS X 系统上的 command-line, X11 或者基于 Aqua 的开源软件。

MacPorts 支持预编译的二进制文件，因此你不必每次都重新从源码压缩包编译，如果你的系统没有安装这些包，它会节省你很多时间。

此时，你可以通过 `port install` 命名来安装 `php53`， `php54`， `php55` 或者 `php56`，比如：

```
sudo port install php54
sudo port install php55
```

你也可以执行 `select` 命令来切换当前的 php 版本：

```
sudo port select --set php php55
```

通过 phpbrew 安装 PHP

phpbrew 是一个安装与管理多个 PHP 版本的工具。它在应用程序或者项目需要不同版本的 PHP 时非常有用，让你不再需要使用虚拟机来处理这些情况。

通过 Liip's binary installer 安装 PHP

php-osx.liip.ch 是另一种流行的选择，它提供了从5.3到5.6版本的单行安装功能。 它并不会覆盖Apple集成的PHP文件，而是将其安装在了一个独立的目录中(/usr/local/php5)。

源码编译

另一个让你控制安装 PHP 版本的选择就是 自行编译。 如果使用这种方法， 你必须先确认是否已经通过 「Apple's Mac Developer Center」 下载、安装 Xcode 或者 "Command Line Tools for XCode"。

集成包 (All-in-One Installers)

上面列出的解决方案主要是针对 PHP 本身， 并不包含：比如 Apache，Nginx 或者 SQL 服务器。 集成包比如 MAMP 和 XAMPP 会安装这些软件并且将他们绑在一起，不过易于安装的背后也牺牲了一定的弹性。

Windows 安装

你可以从 windows.php.net/download 下载二进制包。 解压后，最好为你的 PHP 所在的根目录（php.exe 所在的文件夹）设置 PATH，这样就可以从命令行中直接执行 PHP。

Windows 下有多种安装 PHP 的方式，你可以 下载二进制安装包 并使用 `.msi` 安装程序。从 PHP 5.3.0 之后，这个安装程序将不再提供下载支持。

如果只是学习或者本地开发，可以直接使用 PHP 5.4+ 内置的 Web 服务器， 还能省去配置服务器的麻烦。如果你想要包含有网页服务器以及 MySql 的集成包，那么像是Web Platform Installer, XAMPP, EasyPHP 和 WAMP 这类工具将会帮助你快速建立 Windows 开发环境。不过这些工具将会与线上环境有些许差别，如果你是在 Windows 下开发，而生产环境则部署至 Linux ，请小心。

如果你需要将生产环境部署在 Windows 上，那 IIS7 将会提供最稳定和最佳的性能。你可以使用 phpmanager (IIS7 的图形化插件) 让你简单的设置并管理 PHP。IIS7 也有内置的 FastCGI ，你只需要将 PHP 配置为它的处理器即可。更多详情请见dedicated area on iis.net。

2

代码风格指南

PHP 社区百花齐放，拥有大量的函数库、框架和组件。PHP 开发者通常会在自己的项目中使用若干个外部库，因此 PHP 代码遵循（尽可能接近）同一个代码风格就非常重要，这让开发者可以轻松地将多个代码库整合到自己的项目中。

PHP标准组 提出并发布了一系列的风格建议。其中有部分是关于代码风格的，即 PSR-0, PSR-1, PSR-2 和 PSR-4。这些推荐只是一些被其他项目所遵循的规则，如 Drupal, Zend, Symfony, CakePHP, phpBB, AWS SDK, FuelPHP, Lithium 等。你可以把这些规则用在自己的项目中，或者继续使用自己的风格。

通常情况下，你应该遵循一个已知的标准来编写 PHP 代码。可能是 PSR 的组合或者是 PEAR 或 Zend 编码准则中的一个。这代表其他开发者能够方便的阅读和使用你的代码，并且使用这些组件的应用程序可以和其他第三方的组件保持一致。

- 阅读 PSR-0
- 阅读 PSR-1
- 阅读 PSR-2
- 阅读 PSR-4
- 阅读 PEAR 编码准则
- 阅读 Symfony 编码准则

你可以使用 PHP_CodeSniffer 来检查代码是否符合这些准则，文本编辑器 Sublime Text 的插件也可以提供实时检查。

你可以通过以下两个工具来自动修正你的程序语法，让它符合标准。 一个是 PHP Coding Standards Fixer，它具有良好的测试。 另外一个工具是 php.tools， 它是 sublime text 的一个非常流行的插件sublime-phpfmt，虽然比较新，但是在性能上有了很大的提高，这意味着实时的修复语法会更加的流畅。

你也可以手动运行 phpcs 命令：

```
phpcs -sw --standard=PSR2 file.php
```

它会显示出相应的错误以及如何修正的方法。同样地，这条命令也可以用在 git hook 中，如果你的分支代码不符合选择的代码标准则无法提交。

所有的变量名称以及代码结构建议用英文编写。注释可以使用任何语言，只要让现在以及未来的小伙伴能够容易阅读理解即可。

3

语言亮点

编程范式

PHP 是一个灵活的动态语言，支持多种编程技巧。这几年一直不断的发展，重要的里程碑包含 PHP 5.0 (2004) 增加了完善的面向对象模型，PHP 5.3 (2009) 增加了匿名函数与命名空间以及 PHP 5.4 (2012) 增加的 traits。

面向对象编程

PHP 拥有完整的面向对象编程的特性，包括类，抽象类，接口，继承，构造函数，克隆和异常等。

- 阅读 PHP 面向对象编程
- 阅读 Traits

函数式编程 Functional Programming

PHP 支持函数是"第一等公民"，即函数可以被赋值给一个变量，包括用户自定义的或者是内置函数，然后动态调用它。函数可以作为参数传递给其他函数（称为高阶函数），也可以作为函数返回值返回。

PHP 支持递归，也就是函数自己调用自己，但多数 PHP 代码使用迭代。

自从 PHP 5.3 (2009) 之后开始引入对闭包以及匿名函数的支持。

PHP 5.4 增加了将闭包绑定到对象作用域中的特性，并改善其可调用性，如此即可在大部分情况下使用匿名函数取代一般的函数。

- 学习更多 PHP 函数式编程
- 阅读匿名函数
- 阅读闭包类
- 更多关于 Closures RFC
- 阅读 Callables
- 阅读动态调用函数 `call_user_func_array()`

元编程

PHP 通过反射 API 和魔术方法，可以实现多种方式的元编程。开发者通过魔术方法，如 `__get()`，`__set()`，`__clone()`，`__toString()`，`__invoke()`，等等，可以改变类的行为。Ruby 开发者常说 PHP 没有 `method_missing` 方法，实际上通过 `__call()` 和 `__callStatic()` 就可以完成相同的功能。

- 阅读魔术方法
- 阅读反射
- 阅读重载

命名空间

如前所述，PHP 社区已经有许多开发者开发了大量的代码。这意味着一个类库的 PHP 代码可能使用了另外一个类库中相同的类名。如果他们使用同一个命名空间，那将会产生冲突导致异常。

命名空间 解决了这个问题。如 PHP 手册里所描述，命名空间好比操作系统中的目录，两个同名的文件可以共存在不同的目录下。同理两个同名的 PHP 类可以在不同的 PHP 命名空间下共存，就这么简单。

因此把你的代码放在你的命名空间下就非常重要，避免其他开发者担心与第三方类库冲突。

PSR-4 提供了一种命名空间的推荐使用方式，它提供一个标准的文件、类和命名空间的使用惯例，进而让代码做到随插即用。

2014 年 10 月，PHP-FIG 废弃了上一个自动加载标准： PSR-0，而采用新的自动加载标准 PSR-4。但 PSR-4 要求 PHP 5.3 以上的版本，而许多项目都还是使用 PHP 5.2，所以目前两者都能使用。如果你在新应用或扩展包中使用自动加载标准，应优先考虑使用 PSR-4。

- 阅读命名空间
- 阅读 PSR-0
- 阅读 PSR-4

PHP 标准库

PHP 标准库（SPL）随着 PHP 一起发布，提供了一组类和接口。包含了常用的数据结构类（堆栈，队列，堆等等），以及遍历这些数据结构的迭代器，或者你可以自己实现 SPL 接口。

- 阅读 SPL
- Lynda.com 上的 SPL 视频教程(付费)

命令行接口

PHP 是为开发 Web 应用而创建，不过它的命令行脚本接口(CLI)也非常有用。PHP 命令行编程可以帮你完成自动化的任务，如测试，部署和应用管理。

CLI PHP 编程非常强大，可以直接调用你自己的程序代码而无需创建 Web 图形界面，需要注意的是**不要**把 CLI PHP 脚本放在公开的 web 目录下！

在命令行下运行 PHP ：

```
{% highlight console %}
> php -i
{% endhighlight %}
```

选项 `-i` 将会打印 PHP 配置，类似于 `phpinfo()` 函数。

选项 `-a` 提供交互式 shell，和 Ruby 的 IRB 或 python 的交互式 shell 相似，此外还有很多其他有用的命令行选项。

接下来写一个简单的 "Hello, $name" CLI 程序，先创建名为 `hello.php` 的脚本：

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

PHP 会在脚本运行时根据参数设置两个特殊的变量，`$argc` 是一个整数，表示参数个数，`$argv` 是一个数组变量，包含每个参数的值， 它的第一个元素一直是 PHP 脚本的名称，如本例中为 `hello.php` 。

命令运行失败时，可以通过 `exit()` 表达式返回一个非 0 整数来通知 shell，常用的 exit 返回码可以查看列表.

运行上面的脚本，在命令行输入：

```
{% highlight console %}
> php hello.php
Usage: php hello.php [name]
```

```
> php hello.php world
Hello, world
{% endhighlight %}
```

- 学习如何在命令行运行 PHP
- 学习如何在 Windows 环境下运行 PHP 命令行程序

Xdebug

合适的调试器是软件开发中最有用的工具之一，它使你可以跟踪程序执行结果并监视程序堆栈中的信息。 Xdebug 是一个 php 的调试器，它可以被用来在很多 IDE(集成开发环境) 中做断点调试以及堆栈检查。它还可以像 PHPUnit 和 KCacheGrind 一样，做代码覆盖检查或者程序性能跟踪。

如果你仍在使用 `var_dump()` / `print_r()` 调错，经常会发现自己处于困境，并且仍然找不到解决办法。这时，你该使用调试器了。

安装 Xdebug 可能很费事，但其中一个最重要的「远程调试」特性 —— 如果你在本地开发，并在虚拟机或者其他服务器上测试，远程调试可能是你想要的一种方式。

通常，你需要修改你的 Apache VHost 或者 .htaccess 文件的这些值:

```
{% highlight ini %}
php_value xdebug.remote_host=192.168.?.?
php_value xdebug.remote_port=9000
{% endhighlight %}
```

「remote host」 和 「remote port」 这两项对应和你本地开发机监听的地址和端口。然后将你的 IDE 设置成「listen for connections」模式，并访问网址:

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

你的 IDE 将会拦截当前执行的脚本状态，运行你设置的断点并查看内存中的值。

图形化的调试器可以让你非常容易的逐步的查看代码、变量，以及运行时的 evel 代码。许多 IDE 已经内置或提供了插件支持 XDebug 图形化调试器。比如 MacGDBp 是 Mac 上的一个免费，开源的单机调试器。

- 学习更多 Xdebug
- 学习更多 MacGDBp

4

依赖管理

PHP 有很多可供使用的库、框架和组件。通常你的项目都会使用到其中的若干项 - 这些就是项目的依赖。直到最近，PHP 也没有一个很好的方式来管理这些项目依赖。即使你通过手动的方式去管理，你依然会为自动加载器而担心。但现在这已经不再是问题了。

目前 PHP 有两个使用较多的包管理系统 — Composer 和 PEAR。Composer 是 PHP 所使用的主要的包管理器，然而在很长的一段时间里，PEAR 曾经扮演着这个角色。你应该了解 PEAR 是什么，因为即使你从来没有使用过它，你依然有可能会碰到对它的引用。

Composer 与 Packagist

Composer 是一个**杰出** 的依赖管理器。在 `composer.json` 文件中列出你项目所需的依赖包，加上一点简单的命令，Composer 将会自动帮你下载并设置你的项目依赖。

现在已经有许多 PHP 第三方包已兼容 Composer，随时可以在你的项目中使用。这些「packages(包)」都已列在 Packagist，这是一个官方的 Composer 兼容包仓库。

如何安装 Composer

你可以安装 Composer 到局部 (在你当前工作目录;这里不是很推荐)或是全局(e.g. /usr/local/bin)。我们假设你想安装 Composer 到局部。在你的项目根目录输入:

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

这条命令将会下载 `composer.phar` (一个 PHP 二进制档)。你可以使用 `php` 执行这个文件用来管理你的项目依赖。 **请注意：** 假如你是直接下载代码来编译，请先在线阅读代码确保它是安全的。

Windows环境下安装

对于Windows 的用户而言最简单的获取及执行方法就是使用 ComposerSetup 安装程序，它会执行一个全局安装并设置你的 `$PATH`，所以你在任意目录下在命令行中使用 `composer`。

如何手动安装 Composer

手动安装 Compose r是一个高端的技术;仅管如此还是有许多开发者有各种原因喜欢使用这种交互式的应用程序安装 Composer。在安装前请先确认你的PHP安装项目如下：

- 正在使用一个满足条件的 PHP 版本
- `.phar` 文件可以正确的被执行
- 相关的目录有足够的权限
- 相关有问题的扩展没有被载入
- 相关的 `php.ini` 设置已完成

由于手动安装没有执行这些检查，你必须自己衡量决定是否值得做这些事，以下是如何手动安装 Composer ：

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

路径 `$HOME/local/bin` (或是你选择的路径) 应该在你的 `$PATH` 环境变量中。这将会影响 `composer` 这个命令是否可用.

当你遇到文档指出执行 Composer 的命令是 `php composer.phar install`时，你可以使用下面命令替代:

```
composer install
```

本章节会假设你已经安装了全局的 Composer。

如何设置及安装依赖

Composer 会通过一个 `composer.json` 文件持续的追踪你的项目依赖。 如果你喜欢，你可以手动管理这个文件，或是使用 Composer 自己管理。`composer require` 这个指令会增加一个项目依赖，如果你还没有 `composer.json` 文件，将会创建一个。这里有个例子为你的项目加入 Twig 依赖。

```
composer require twig/twig:~1.8
```

另外 `composer init` 命令将会引导你创建一个完整的 `composer.json` 文件到你的项目之中。无论你使用哪种方式，一旦你创建了 `composer.json` 文件，你可以告诉 Composer 去下载及安装你的依赖到 `vendors/` 目录中。这命令也适用于你已经下载并已经提供了一个 `composer.json` 的项目:

```
composer install
```

接下来，添加这一行到你应用的主要 PHP 文件中，这将会告诉 PHP 为你的项目依赖使用 Composer 的自动加载器。

```
{% highlight php %}
<?php
require 'vendor/autoload.php';
{% endhighlight %}
```

现在你可以使用你项目中的依赖，且它们会在需要时自动完成加载。

更新你的依赖

Composer 会建立一个 `composer.lock` 文件，在你第一次执行 `php composer.phar install` 时，存放下载的每个依赖包精确的版本编号。假如你要分享你的项目给其他开发者，并且 `composer.lock` 文件也在你分享的文件

之中的话。 当他们执行 `php composer.phar install` 这个命令时，他们将会得到与你一样的依赖版本。 当你要更新你的依赖时请执行 `php composer.phar update`。

当你需要灵活的定义你所需要的依赖版本时，这是最有用。 举例来说需要一个版本为 `~1.8` 时，意味着 “任何大于 1.8.0 ，但小于 2.0.x-dev 的版本”。你也可以使用通配符 `*` 在 `1.8.*` 之中。现在Composer在`composer update` 时将升级你的所有依赖到你限制的最新版本。

更新通知

要接收关于新版本的更新通知。你可以注册 VersionEye，这个 web 服务可以监控你的 Github 及 BitBucket 帐号中的 `composer.json` 文件，并且当包有新更新时会发送邮件给你。

检查你的依赖安全问题

Security Advisories Checker 是一个 web 服务和一个命令行工具，二者都会仔细检查你的 `composer.lock` 文件，并且告诉你任何你需要更新的依赖。

处理 Composer 全局依赖

Composer 也可以处理全局依赖和他们的二进制文件。用法很直接，你所要做的就是在命令前加上`global`前缀。如果你想安装 PHPUnit 并使它全局可用，你可以运行下面的命令：

```
{% highlight console %}
composer global require phpunit/phpunit
{% endhighlight %}
```

这将会创建一个 `~/.composer` 目录存放全局依赖，要让已安装依赖的二进制命令随处可用，你需要添加 `~/.composer/vendor/bin` 目录到你的 `$PATH` 变量。

- 其他学习 Composer 相关资源

PEAR 介绍

PEAR 是另一个常用的依赖包管理器，它跟 Composer 很类似，但是也有一些显著的区别。

PEAR 需要扩展包有专属的结构，开发者在开发扩展包的时候要提前考虑为 PEAR 定制，否则后面将无法使用 PEAR.

PEAR 安装扩展包的时候，是全局安装的，意味着一旦安装了某个扩展包，同一台服务器上的所有项目都能用上，当然，好处是当多个项目共同使用同一个扩展包的同一个版本，坏处是如果你需要使用不同版本的话，就会产生冲突.

如何安装 PEAR

你可以通过下载 `.phar` 文件来安装 PEAR. 官方文档安装部分 里面有不同系统中安装 PEAR 的详细信息.

如果你是使用 Linux，你可以尝试找下系统应用管理器，举个栗子，Debian 和 Ubuntu 有个 `php-pear` 的 apt 安装包.

如何安装扩展包

如果扩展包是在 PEAR packages list 这个列表里面的，你可以使用以下命令安装:

```
{% highlight console %}
pear install foo
{% endhighlight %}
```

如果扩展包是托管到别的渠道上，你需要 `发现 (discover)` 渠道先，请见文档 使用渠道.

- Learn about PEAR

使用 Composer 来安装 PEAR 扩展包

如果你正在使用 Composer，并且你想使用一些 PEAR 的代码，你可以通过 Composer 来安装 PEAR 扩展包.

下面是从 `pear2.php.net` 安装代码依赖的示例:

```
{% highlight json %}
{
```

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

第一部分 `"repositories"` 是让 Composer 从哪个渠道去获取扩展包，然后，`"repositories"` 部分使用下面的命名规范：

> pear-channel/Package

前缀 "pear" 是为了避免冲突写死的.

成功安装扩展包以后，代码会放到项目的 `vendor` 文件夹中，并且可以通过加载 Composer 的自动加载器进行加载：

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

在代码里面可以这样使用：

```
{% highlight php %}
<?php
$request = new pear2\HTTP\Request();
{% endhighlight %}
```

- 学习更多 PEAR 和 Composer 的使用

5

开发实践

基础知识

PHP 是一门庞大的语言，各个水平层次的开发者都可以利用它进行迅捷高效的开发。然而在对语言逐渐深入的学习过程中，我们往往会因为走捷径和/或不良习惯而忘记（或忽视掉）基础的知识。为了帮助彻底解决这个问题，这一章的目的就是提醒开发人员注意有关 PHP 的基础编程实践。

- 学习更多基础知识

日期和时间

PHP 中 DateTime 类的作用是在你读、写、比较或者计算日期和时间时提供帮助。除了 DateTime 类之外，PHP 还有很多与日期和时间相关的函数，但 DateTime 类为大多数常规使用提供了优秀的面向对象接口。它还可以处理时区，不过这并不在这篇简短的介绍之内。

在使用 DateTime 之前，通过 `createFromFormat()` 工厂方法将原始的日期与时间字符串转换为对象或使用 `new DateTime` 来取得当前的日期和时间。使用 `format()` 将 DateTime 转换回字符串用于输出。

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

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

对 DateTime 进行计算时可以使用 DateInterval 类。DateTime 类具有例如 `add()` 和 `sub()` 等将 DateInterval 当作参数的方法。编写代码时注意不要认为每一天都是由相同的秒数构成的，不论是夏令时（DST）还是时区转换，使用时间戳计算都会遇到问题，应当选择日期间隔。使用 `diff()` 方法来计算日期之间的间隔，它会返回新的 DateInterval，非常容易进行展示。

```
{% highlight php %}
<?php
// create a copy of $start and add one month and 6 days
$end = clone $start;
$end->add(new DateInterval('P1M6D'));

$diff = $end->diff($start);
echo 'Difference: ' . $diff->format('%m month, %d days (total: %a days)') . "\n";
// Difference: 1 month, 6 days (total: 37 days)
{% endhighlight %}
```

DateTime 对象之间可以直接进行比较：

```
{% highlight php %}
<?php
if ($start < $end) {
    echo "Start is before end!\n";
}
{% endhighlight %}
```

最后一个例子来演示 DatePeriod 类。它用来对循环的事件进行迭代。向它传入开始时间、结束时间和间隔区间，会得到这其中所有的事件。

```
{% highlight php %}
<?php
// output all thursdays between $start and $end
$periodInterval = DateInterval::createFromDateString('first thursday');
$periodIterator = new DatePeriod($start, $periodInterval, $end, DatePeriod::EXCLUDE_START_DATE);
foreach ($periodIterator as $date) {
    // output each date in the period
    echo $date->format('Y-m-d') . ' ';
}
{% endhighlight %}
```

- 阅读 DateTime
- 阅读日期格式（支持的日期字符串格式）

设计模式

当你在编写自己的应用程序时，最好在项目的代码和整体架构中使用通用的设计模式，这将帮助你更轻松地对程序进行维护，也能够让其他的开发者更快地理解你的代码。

当你使用框架进行开发时，绝大部分的上层代码以及项目结构都会基于所使用的框架，因此很多关于设计模式的决定已经由框架帮你做好了。当然，你还是可以挑选你最喜欢的模式并在你的代码中进行应用。但如果你并没有使用框架的话，你就需要自己去寻找适合你的应用的最佳模式了。

- 学习更多设计模式

使用 UTF-8 编码

本章是由 Alex Cabal 最初撰写在 PHP Best Practices 中的，我们使用它作为进行建议的基础。

> **这不是在开玩笑。请小心、仔细并且前后一致地处理它。**

目前，PHP 仍未在底层实现对 Unicode 的支持。虽然有很多途径可以确保 UTF-8 字符串能够被正确地处理，但这并不是很简单的事情，通常需要对 Web 应用进行全方面的检查，从 HTML 到 SQL 再到 PHP。我们将争取进行一个简洁实用的总结。

PHP 层面的 UTF-8

最基本的字符串操作，像是连结两个字符串或将字符串赋值给变量，并不需要对 UTF-8 做特别的处理。然而大多数字符串的函数，像 `strpos()` 和 `strlen()`，确实需要特别的处理。这些函数名中通常包含 `mb_*`：比如，`mb_strpos()` 和 `mb_strlen()`。这些 `mb_*` 字符串是由 Multibyte String Extension 提供支持的，它专门为操作 Unicode 字符串而特别进行了设计。

在操作 Unicode 字符串时，请你务必使用 `mb_*` 函数。例如，如果你对一个 UTF-8 字符串使用 `substr()`，那返回的结果中有很大可能会包含一些乱码。正确的方式是使用 `mb_substr()`。

最难的地方在于每次都要记得使用 `mb_*` 函数。如果你哪怕只有一次忘记了使用，你的 Unicode 字符串就有在接下来的过程中变成乱码的风险。

不是所有的字符串函数都有一个对应的 `mb_*` 函数。如果你想要的功能没有对应的 `mb_*` 函数的话，那只能说你运气不佳了。

你应该在你所有的 PHP 脚本（或全局包含的脚本）的开头使用 `mb_internal_encoding()` 函数，然后紧接着在会对浏览器进行输出的脚本中使用 `mb_http_output()`。在每一个脚本当中明确声明字符串的编码可以免去很多日后的烦恼。

另外，许多对字符串进行操作的函数都有一个可选的参数用来指定字符串编码。当可以设定这类参数时，你应该始终明确指定使用 UTF-8。例如，`htmlentities()` 有一个字符编码的选项，你应该始终将其设为 UTF-8。从 PHP 5.4.0 开始，`htmlentities()` 和 `htmlspecialchars()` 的编码都已经被默认设为了 UTF-8。

最后，如果你所编写的是分布式的应用程序并且不能确定 `mbstring` 扩展一定开启的话，可以考虑使用 patchwork/utf8 Composer 包。它会在 `mbstring` 可用时自动使用，否则自动切换回非 UTF-8 函数。

数据库层面的 UTF-8

如果你使用 PHP 来操作到 MySQL，有些时候即使你做到了上面的每一点，你的字符串仍可能面临在数据库中以非 UTF-8 的格式进行存储的问题。

为了确保你的字符串从 PHP 到 MySQL都使用 UTF-8，请检查确认你的数据库和数据表都设定为 `utf8mb4` 字符集和整理，并且确保你的 PDO 连接请求也使用了 `utf8mb4` 字符集。请看下方的示例代码，这是 *非常重要* 的。

请注意为了完整的 UTF-8 支持，你必须使用 `utf8mb4` 而不是 `utf8`！你会在进一步阅读中找到原因。

浏览器层面的 UTF-8

使用 `mb_http_output()` 函数来确保 PHP 向浏览器输出 UTF-8 格式的字符串。

随后浏览器需要接收 HTTP 应答来指定页面是由 UTF-8 进行编码的。以前这一步是通过在页面 `<head>` 标签下包含字符集 `<meta>` 标签实现的，这是一种可行的方式。但更好的做法是在 `Content-Type` 响应头中进行设置，因为这样做的速度会更快。

```
{% highlight php %}
<?php
// Tell PHP that we're using UTF-8 strings until the end of the script
mb_internal_encoding('UTF-8');

// Tell PHP that we'll be outputting UTF-8 to the browser
mb_http_output('UTF-8');

// Our UTF-8 test string
$string = 'Êl síla erin lû e-govaned vîn.';

// Transform the string in some way with a multibyte function
// Note how we cut the string at a non-Ascii character for demonstration purposes
$string = mb_substr($string, 0, 15);

// Connect to a database to store the transformed string
// See the PDO example in this document for more information
// Note the `charset=utf8mb4` in the Data Source Name (DSN)
$link = new PDO(
    'mysql:host=your-hostname;dbname=your-db;charset=utf8mb4',
    'your-username',
    'your-password',
    array(
```

```
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_PERSISTENT => false
    )
);

// Store our transformed string as UTF-8 in our database
// Your DB and tables are in the utf8mb4 character set and collation, right?
$handle = $link->prepare('insert into ElvishSentences (Id, Body) values (?, ?)');
$handle->bindValue(1, 1, PDO::PARAM_INT);
$handle->bindValue(2, $string);
$handle->execute();

// Retrieve the string we just stored to prove it was stored correctly
$handle = $link->prepare('select * from ElvishSentences where Id = ?');
$handle->bindValue(1, 1, PDO::PARAM_INT);
$handle->execute();

// Store the result into an object that we'll output later in our HTML
$result = $handle->fetchAll(\PDO::FETCH_OBJ);

header('Content-Type: text/html; charset=UTF-8');
?><!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>UTF-8 test page</title>
    </head>
    <body>
        <?php
        foreach($result as $row){
            print($row->Body);  // This should correctly output our transformed UTF-8 string to the browser
        }
        ?>
    </body>
</html>
{% endhighlight %}
```

延伸阅读

- PHP 手册：字符串运算符
- PHP 手册：字符串函数
    - `strpos()`

    - `strlen()`
    - `substr()`
- PHP 手册：多字节字符串函数
    - `mb_strpos()`
    - `mb_strlen()`
    - `mb_substr()`
    - `mb_internal_encoding()`
    - `mb_http_output()`
    - `htmlentities()`
    - `htmlspecialchars()`
- PHP UTF-8 Cheatsheet
- Handling UTF-8 with PHP
- Stack Overflow: What factors make PHP Unicode-incompatible?
- Stack Overflow: Best practices in PHP and MySQL with international strings
- How to support full Unicode in MySQL databases
- Bringing Unicode to PHP with Portable UTF-8
- Stack Overflow: DOMDocument loadHTML does not encode UTF-8 correctly

6

依赖注入

出自维基百科 Wikipedia:

> 依赖注入是一种允许我们从硬编码的依赖中解耦出来，从而在运行时或者编译时能够修改的软件设计模式。

这句解释让依赖注入的概念听起来比它实际要复杂很多。依赖注入通过构造注入，函数调用或者属性的设置来提供组件的依赖关系。就是这么简单。

基本概念

我们可以用一个简单的例子来说明依赖注入的概念

下面的代码中有一个 `Database` 的类，它需要一个适配器来与数据库交互。我们在构造函数里实例化了适配器，从而产生了耦合。这会使测试变得很困难，而且 `Database` 类和适配器耦合的很紧密。

```
{% highlight php %}
<?php
namespace Database;

class Database
{
    protected $adapter;

    public function __construct()
    {
        $this->adapter = new MySqlAdapter;
    }
}

class MysqlAdapter {}
{% endhighlight %}
```

这段代码可以用依赖注入重构，从而解耦

```
{% highlight php %}
<?php
namespace Database;

class Database
{
    protected $adapter;

    public function __construct(MySqlAdapter $adapter)
    {
        $this->adapter = $adapter;
    }
}

class MysqlAdapter {}
{% endhighlight %}
```

现在我们通过外界给予 `Database` 类的依赖，而不是让它自己产生依赖的对象。我们甚至能用可以接受依赖对象参数的成员函数来设置，或者如果 `$adapter` 属性本身是 `public` 的，我们可以直接给它赋值。

复杂的问题

如果你曾经了解过依赖注入，那么你可能见过 *"控制反转"(Inversion of Control)* 或者 *"依赖反转准则"(Dependency Inversion Principle)*这种说法。这些是依赖注入能解决的更复杂的问题。

控制反转

顾名思义，一个系统通过组织控制和对象的完全分离来实现"控制反转"。对于依赖注入，这就意味着通过在系统的其他地方控制和实例化依赖对象，从而实现了解耦。

一些 PHP 框架很早以前就已经实现控制反转了，但是问题是，应该反转哪部分以及到什么程度？比如， MVC 框架通常会提供超类或者基本的控制器类以便其他控制器可以通过继承来获得相应的依赖。这就是控制反转的例子，但是这种方法是直接移除了依赖而不是减轻了依赖。

依赖注入允许我们通过按需注入的方式更加优雅地解决这个问题，完全不需要任何耦合。

依赖反转准则

依赖反转准则是面向对象设计准则 S.O.L.I.D 中的 "D" ,倡导 *"依赖于抽象而不是具体"*。简单来说就是依赖应该是接口/约定或者抽象类，而不是具体的实现。我们能很容易重构前面的例子，使之遵循这个准则

```
{% highlight php %}
<?php
namespace Database;

class Database
{
    protected $adapter;

    public function __construct(AdapterInterface $adapter)
    {
        $this->adapter = $adapter;
    }
}

interface AdapterInterface {}

class MysqlAdapter implements AdapterInterface {}
{% endhighlight %}
```

现在 `Database` 类依赖于接口，相比依赖于具体实现有更多的优势。

假设你工作的团队中，一位同事负责设计适配器。在第一个例子中，我们需要等待适配器设计完之后才能单元测试。现在由于依赖是一个接口/约定，我们能轻松地模拟接口测试，因为我们知道同事会基于约定实现那个适配器

这种方法的一个更大的好处是代码扩展性变得更高。如果一年之后我们决定要迁移到一种不同的数据库，我们只需要写一个实现相应接口的适配器并且注入进去，由于适配器遵循接口的约定，我们不需要额外的重构。

容器

你应该明白的第一件事是依赖注入容器和依赖注入不是相同的概念。容器是帮助我们更方便地实现依赖注入的工具，但是他们通常被误用来实现反模式设计 Service Location 。把一个依赖注入容器作为 Service Locator 注入进类中隐式地建立了对于容器的依赖，而不是真正需要替换的依赖，而且还会让你的代码更不透明，最终变得更难测试。

大多数现代的框架都有自己的依赖注入容器，允许你通过配置将依赖绑定在一起。这实际上意味着你能写出和框架层同样干净、解耦的应用层代码。

延伸阅读

- Learning about Dependency Injection and PHP
- What is Dependency Injection?
- Dependency Injection: An analogy
- Dependency Injection: Huh?
- Dependency Injection as a tool for testing

7

数据库

绝大多数时候你的 PHP 程序都需要使用数据库来长久地保存数据。这时你有一些不同的选择可以来连接并与数据库进行交互。在 **PHP 5.1.0 之前**，我们推荐的方式是使用例如 mysqli，pgsql，mssql 等原生驱动。

原生驱动是在只使用 一个 数据库的情况下的不错的方式，但如果，举个例子来说，你同时使用了 MySQL 和一点点 MSSQL，或者你需要使用 Oracle 的数据库，那你就不能够只使用一个数据库驱动了。你需要为每一个数据库去学习各自不同的 API — 这样做显然不科学。

MySQL 扩展

PHP 中的 mysql 扩展已经不再进行新的开发了，并且已经在 PHP 5.5.0 版本中正式被废弃，这意味着它将会在接下来的更新中被移除。如果你使用 `mysql_*` 开头的函数，例如 `mysql_connect()` 和 `mysql_query()` 的话，它们将会在后续的 PHP 版本无法使用。因此在以后的某个时间你需要重写你的代码。最好的办法是在你的开发计划中使用 mysqli 或 PDO 来取代 mysql 扩展，这样你才不会在后面手忙脚乱。

如果你正从零开始，请一定避免使用 mysql 扩展：请选择 MySQLi 扩展,或者使用 PDO。

- PHP: MySQL增强版扩展
- MySQL 开发者 PDO 使用教程

PDO 扩展

PDO 是一个数据库连接抽象类库 — 自 5.1.0 版本起内置于 PHP 当中 — 它提供了一个通用的接口来与不同的数据库进行交互。比如你可以使用相同的简单代码来连接 MySQL 或是 SQLite：

```
{% highlight php %}
<?php
// PDO + MySQL
$pdo = new PDO('mysql:host=example.com;dbname=database', 'user', 'password');
$statement = $pdo->query("SELECT some_field FROM some_table");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['some_field']);

// PDO + SQLite
$pdo = new PDO('sqlite:/path/db/foo.sqlite');
$statement = $pdo->query("SELECT some_field FROM some_table");
$row = $statement->fetch(PDO::FETCH_ASSOC);
echo htmlentities($row['some_field']);
{% endhighlight %}
```

PDO 并不会对 SQL 请求进行转换或者模拟实现并不存在的功能特性；它只是单纯地使用相同的 API 连接不同种类的数据库。

更重要的是，`PDO` 使你能够安全的插入外部输入(例如 ID)到你的 SQL 请求中而不必担心 SQL 注入的问题。这可以通过使用 PDO 语句和限定参数来实现。

我们来假设一个 PHP 脚本接收一个数字 ID 作为一个请求参数。这个 ID 应该被用来从数据库中取出一条用户记录。下面是一个`错误`的做法：

```
{% highlight php %}
<?php
$pdo = new PDO('sqlite:/path/db/users.db');
$pdo->query("SELECT name FROM users WHERE id = " . $_GET['id']); // <-- NO!
{% endhighlight %}
```

这是一段糟糕的代码。你正在插入一个原始的请求参数到 SQL 请求中。这将让被黑客轻松地利用[SQL 注入]方式进行攻击。想一下如果黑客将一个构造的 `id` 参数通过像 `http://domain.com/?id=1%3BDELETE+FROM+users` 这样的 URL 传入。这将会使 `$_GET['id']` 变量的值被设为 `1;DELETE FROM users` 然后被执行从而删除所有的 user 记录！因此，你应该使用 PDO 限制参数来过滤 ID 输入。

```
{% highlight php %}
<?php
```

```
$pdo = new PDO('sqlite:/path/db/users.db');
$stmt = $pdo->prepare('SELECT name FROM users WHERE id = :id');
$id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT); // <-- filter your data first (see [Data Filtering](ht
$stmt->bindParam(':id', $id, PDO::PARAM_INT); // <-- Automatically sanitized for SQL by PDO
$stmt->execute();
{% endhighlight %}
```

这是正确的代码。它在一条 PDO 语句中使用了一个限制参数。这将对外部 ID 输入在发送给数据库之前进行转义来防止潜在的 SQL 注入攻击。

对于写入操作，例如 INSERT 或者 UPDATE，进行数据过滤并对其他内容进行清理（去除 HTML 标签，Javascript 等等）是尤其重要的。PDO 只会为 SQL 进行清理，并不会为你的应用做任何处理。

- 了解 PDO

你也应该知道数据库连接有时会耗尽全部资源，如果连接没有被隐式地关闭的话，有可能会造成可用资源枯竭的情况。不过这通常在其他语言中更为常见一些。使用 PDO 你可以通过销毁（destroy）对象，也就是将值设为 NULL，来隐式地关闭这些连接，确保所有剩余的引用对象的连接都被删除。如果你没有亲自做这件事情，PHP 会在你的脚本结束的时候自动为你完成 —— 除非你使用的是持久链接。

- 了解 PDO 连接

数据库交互

当开发者第一次接触 PHP 时，通常会使用类似下面的代码来将数据库的交互与表示层逻辑混在一起：

```
{% highlight php %}
<ul>
<?php
foreach ($db->query('SELECT * FROM table') as $row) {
    echo "<li>".$row['field1']." - ".$row['field1']."</li>";
}
?>
</ul>
{% endhighlight %}
```

这从很多方面来看都是错误的做法，主要是由于它不易阅读又难以测试和调试。而且如果你不加以限制的话，它会输出非常多的字段。

其实还有许多不同的解决方案来完成这项工作 — 取决于你倾向于 面向对象编程（OOP）还是函数式编程 — 但必须有一些分离的元素。

来看一下最基本的做法：

```
{% highlight php %}
<?php
function getAllFoos($db) {
    return $db->query('SELECT * FROM table');
}

foreach (getAllFoos($db) as $row) {
    echo "<li>".$row['field1']." - ".$row['field1']."</li>"; // BAD!!
}
{% endhighlight %}
```

这是一个不错的开头。将这两个元素放入了两个不同的文件于是你得到了一些干净的分离。

创建一个类来放置上面的函数，你就得到了一个「Model」。创建一个简单的 `.php` 文件来存放表示逻辑，你就得到了一个「View」。这已经很接近 MVC — 一个大多数框架常用的面向对象的架构。

foo.php

```
{% highlight php %}
<?php
$db = new PDO('mysql:host=localhost;dbname=testdb;charset=utf8', 'username', 'password');
```

```
// Make your model available
include 'models/FooModel.php';

// Create an instance
$fooModel = new FooModel($db);
// Get the list of Foos
$fooList = $fooModel->getAllFoos();

// Show the view
include 'views/foo-list.php';
{% endhighlight %}
```

models/FooModel.php

```
{% highlight php %}
<?php
class FooModel
{
    protected $db;

    public function __construct(PDO $db)
    {
        $this->db = $db;
    }

    public function getAllFoos() {
        return $this->db->query('SELECT * FROM table');
    }
}
{% endhighlight %}
```

views/foo-list.php

```
{% highlight php %}
<?php foreach ($fooList as $row): ?>
    <?= $row['field1'] ?> - <?= $row['field1'] ?>
<?php endforeach ?>
{% endhighlight %}
```

向大多数现代框架的做法学习是很有必要的，尽管多了一些手动的工作。你可以并不需要每一次都完全这么做，但将太多的表示逻辑层代码和数据库交互掺杂在一些将会为你在想要对程序进行单元测试时带来真正的麻烦。

PHPBridge 具有一项非常棒的资源叫做创建一个数据类。它包含了非常相似的逻辑而且非常适合刚刚习惯数据库交互概念的开发者使用。

数据库抽象层

许多框架都提供了自己的数据库抽象层，其中一些是设计在 PDO 的上层的。这些抽象层通常将你的请求在 PHP 方法中包装起来，通过模拟的方式来使你的数据库拥有一些之前不支持的功能。这种抽象是真正的数据库抽象，而不单单只是 PDO 提供的数据库连接抽象。这类抽象的确会增加一定程度的性能开销，但如果你正在设计的应用程序需要同时使用 MySQL，PostgreSQL 和 SQLite 时，一点点的额外性能开销对于代码整洁度的提高来说还是很值得的。

有一些抽象层使用的是PSR-0 或 PSR-4 命名空间标准，所以他们可以安装在任何你需要的应用程序中。

- Aura SQL
- Doctrine2 DBAL
- Propel
- ZF2 Db

8

使用模板

模板提供了一种简便的方式，将展现逻辑从控制器和业务逻辑中分离出来。

一般来说，模板包含应用程序的 HTML 代码，但也可以使用其他的格式，例如 XML 。

模板通常也被称为「视图」，而它是 模型-视图-控制器 (MVC) 软件架构模式第二个元素的 一部份 。

好处

使用模板的主要好处是可以将呈现逻辑与应用程序的其他部分进行分离。模板的单一职责就是呈现格式化后的内容。它不负责数据的查询，保存或是其他复杂的任务。进一步促成了更干净、更具可读性的代码，在团队协作开发中尤其有用，开发者可以专注服务端的代码（控制器、模型），而设计师负责客户端代码（网页）。

模板同时也改善了前端代码的组织架构。一般来说，模板放置在「视图」文件夹中，每一个模板都放在独立的一个文件中。这种方式鼓励代码重用，它将大块的代码拆成较小的、可重用的片段，通常称为局部模板。举例来说，网站的头、尾区块可以各自定义为一个模板，之后将它们放在每一个页面模板的上、下位置。

最后，根据你选择的类库，模板可以通过自动转义用户的内容，从而带来更多的安全性。有些类库甚至提供沙箱机制，模板设计者只能使用在白名单中的变量和函数。

原生 PHP 模板

原生 PHP 模板就是指直接用 PHP 来写模板，这是很自然的选择，因为 PHP 本身其实是个模板语言。这代表你可以在其他的语言中结合 PHP 使用，比如 HTML 。这对 PHP 开发者相当有利，因为不需要额外学习新的语法，他们熟知可以使用的函数，并且使用的编辑器也已经内置了语法高亮和自动补全。此外，原生的 PHP 模板没有了编译阶段，速度会更快。

现今的 PHP 框架都会使用一些模板系统，这当中多数是使用原生的 PHP 语法。在框架之外，一些类库比如 Plates 或 Aura.View，提供了现代化模板的常见功能，比如继承、布局、扩展，让原生的 PHP 模板更容易使用。

原生 PHP 模板的简单示例

使用 Plates 类库。

```
{% highlight php %}
<?php // user_profile.php ?>

<?php $this->insert('header', ['title' => 'User Profile']) ?>

<h1>User Profile</h1>
<p>Hello, <?=$this->escape($name)?></p>

<?php $this->insert('footer') ?>
{% endhighlight %}
```

原生 PHP 模板使用继承的示例

使用 Plates 类库。

```
{% highlight php %}
<?php // template.php ?>

<html>
<head>
    <title><?=$title?></title>
</head>
<body>

<main>
```

```
    <?=$this->section('content')?>
</main>

</body>
</html>
{% endhighlight %}

{% highlight php %}
<?php // user_profile.php ?>

<?php $this->layout('template', ['title' => 'User Profile']) ?>

<h1>User Profile</h1>
<p>Hello, <?=$this->escape($name)?></p>
{% endhighlight %}
```

编译模板

尽管 PHP 不断升级为成熟的、面向对象的语言，但它作为模板语言 没有改善多少。编译模板，比如 Twig 或 Smarty* ，提供了模板专用的新语法，填补了这片空白。从自动转义到继承以及简化控制结构，编译模板设计地更容易编写，可读性更高，同时使用上也更加的安全。编译模板甚至可以在不同的语言中使用，Mustache 就是一个很好的例子。由于这些模板需要编译，在性能上会带来一些轻微的影响，不过如果适当的使用缓存，影响就变得非常小了。

**虽然 Smarty 提供了自动转义的功能，不过这个功能默认是关闭的*

编译模板简单示例

使用 Twig 类库。

```
{% highlight html+jinja %}
{% raw %}
{% include 'header.html' with {'title': 'User Profile'} %}

<h1>User Profile</h1>
<p>Hello, {{ name }}</p>

{% include 'footer.html' %}
{% endraw %}
{% endhighlight %}
```

编译模板使用继承示例

使用 Twig 类库。

```
{% highlight html+jinja %}
{% raw %}
// template.html

<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
```

```
<main>
    {% block content %}{% endblock %}
</main>

</body>
</html>
{% endraw %}
{% endhighlight %}

{% highlight html+jinja %}
{% raw %}
// user_profile.html

{% extends "template.html" %}

{% block title %}User Profile{% endblock %}
{% block content %}
    <h1>User Profile</h1>
    <p>Hello, {{ name }}</p>
{% endblock %}
{% endraw %}
{% endhighlight %}
```

延伸阅读

文章与教程

- Templating Engines in PHP
- An Introduction to Views & Templating in CodeIgniter
- Getting Started With PHP Templating
- Roll Your Own Templating System in PHP
- Master Pages
- Working With Templates in Symfony 2
- Writing Safer Templates

类库

- Aura.View *(native)*
- Blade *(compiled, framework specific)*
- Brainy *(compiled)*
- Dwoo *(compiled)*
- Latte *(compiled)*
- Mustache *(compiled)*
- PHPTAL *(compiled)*
- Plates *(native)*
- Smarty *(compiled)*
- Twig *(compiled)*
- Zend\View *(native, framework specific)*

9

错误与异常

错误

在许多「重异常」(exception-heavy) 的编程语言中，一旦发生错误，就会抛出异常。这确实是一个可行的方式。不过 PHP 却是一个 「轻异常」(exception-light) 的语言。当然它确实有异常机制，在处理对象时，核心也开始采用这个机制来处理，只是 PHP 会尽可能的执行而无视发生的事情，除非是一个严重错误。

举例来说:

```
{% highlight console %}
$ php -a
php > echo $foo;
Notice: Undefined variable: foo in php shell code on line 1
{% endhighlight %}
```

这里只是一个 notice 级别的错误，PHP 仍然会愉快的继续执行。这对有「重异常」编程经验的人来说会带来困惑，例如在 Python 中，引用一个不存在的变量会抛出异常：

```
{% highlight console %}
$ python
>>> print foo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'foo' is not defined
{% endhighlight %}
```

本质上的差异在于 Python 会对任何小错误进行抛错，因此开发人员可以确信任何潜在的问题或者边缘的案例都可以被捕捉到，与此同时 PHP 仍然会保持执行，除非极端的问题发生才会抛出异常。

错误严重性

PHP 有几个错误严重性等级。三个最常见的的信息类型是错误（error）、通知（notice）和警告（warning）。它们有不同的严重性: `E_ERROR` 、`E_NOTICE` 和 `E_WARNING` 。错误是运行期间的严重问题，通常是因为代码出错而造成，必须要修正它，否则会使 PHP 停止执行。通知是建议性质的信息，是因为程序代码在执行期有可能造成问题，但程序不会停止。 警告是非致命错误，程序执行也不会因此而中止。

另一个在编译期间会报错的信息类型是「E_STRICT」。这个信息用來建议修改程序代码以维持最佳的互通性并能与今后的 PHP 版本兼容。

更改 PHP 错误报告行为

错误报告可以由 PHP 配置及函数调用改变。使用 PHP 内置的函数 `error_reporting()`，可以设定程序执行期间的错误等级，方法是传入预定义的错误等级常量，意味着如果你只想看到警告和错误（而非通知），你可以这样设定：

```
{% highlight php %}
<?php
error_reporting(E_ERROR | E_WARNING);
{% endhighlight %}
```

你也可以控制错误是否在屏幕上显示 （开发时比较有用）或隐藏后记录日志 （适用于正式环境）。如果想知道更多细节，可以查看 错误报告 章节。

行内错误抑制

你可以让 PHP 利用错误控制操作符 `@` 来抑制特定的错误。将这个操作符放置在表达式之前，其后的任何错误都不会出现。

```
{% highlight php %}
<?php
echo @$foo['bar'];
{% endhighlight %}
```

如果 `$foo['bar']` 存在，程序会将结果输出，如果变量 `$foo` 或是 `'bar'` 键值不存在，则会返回 null 并且不输出任何东西。如果不使用错误控制操作符，这个表达式会产生一个错误信息 `PHP Notice: Undefined variable: foo` 或 `PHP Notice: Undefined index: bar` 。

这看起来像是个好主意，不过也有一些讨厌的代价。PHP 处理使用 `@` 的表达式比起不用时效率会低一些。过早的性能优化在所有程序语言中也许都是争论点，不过如果性能在你的应用程序 / 类库中占有重要地位，那么了解错误控制操作符的性能影响就比较重要。

其次，错误控制操作符会 **完全** 吃掉错误。不但没有显示，而且也不会记录在错误日志中。此外，在正式环境中 PHP 也没有办法关闭错误控制操作符。也许你认为那些错误时无害的，不过那些较具伤害性的错误同时也会被隐藏。

如果有方法可以避免错误抑制符，你应该考虑使用，举例来说，上面的程序代码可以这样重写：

```
{% highlight php %}
<?php
```

```
echo isset($foo['bar']) ? $foo['bar'] : '';
{% endhighlight %}
```

当 `fopen()` 载入文件失败时，也许是一个使用错误抑制符的合理例子。你可以在尝试载入文件前检查是否存在，但是如果这个文件在检查后才被删除，而此时 `fopen()` 还未执行 （听起来有点不太可能，但是确实会发生），这时 `fopen()` 会返回 `false` *并且* 抛出操作。这也许应该由 PHP 本身来解决，但这时一个错误抑制符才能有效解决的例子。

前面我们提到在正式的 PHP 环境中没有办法关闭错误控制操作符。但是 Xdebug 有一个 `xdebug.scream` 的 ini 配置项，可以关闭错误控制操作符。你可以按照下面的方式修改 `php.ini`。

```
{% highlight ini %}
xdebug.scream = On
{% endhighlight %}
```

你也可以在执行期间通过 `ini_set` 函数来设置这个值：

```
{% highlight php %}
<?php
ini_set('xdebug.scream', '1')
{% endhighlight %}
```

「Scream」这个 PHP 扩展提供了和 xDebug 类似的功能，只是 Scream 的 ini 设置项叫做 `scream.enabled`。

当你在调试代码而错误信息被隐藏时，这是最有用的方法。请务必小心使用 `scream`，而是把它当时暂时性的调试工具。有许多的 PHP 函数类库代码也许无法在错误抑制操作符停用时正常使用。

- Error Control Operators
- SitePoint
- Xdebug
- Scream

错误异常类

PHP 可以完美化身为「重异常」的程序语言，只需要几行代码就能切换过去。基本上你可以利用 `ErrorException` 类抛出「错误」来当做「异常」，这个类是继承自 `Exception` 类。

这在大量的现代框架中是一个常见的做法，比如 Symfony 和 Laravel。Laravel 默认使用 Whoops! 扩展包来处理错误，如果 `app.debug` 启动的话，会将错误当成异常显示出来，而关闭则会隐藏。

在开发过程中将错误当作异常抛出可以更好的处理它，如果在开发时发生异常，你可以将它包在一个 catch 语句中具体说明这种情况如何处理。每捕捉一个异常，都会使你的应用程序越来越健壮。

更多关于如何使用 `ErrorException` 来处理错误的细节，可以参考 ErrorException Class。

- Error Control Operators
- Predefined Constants for Error Handling
- `error_reporting()`
- Reporting

异常

异常是许多流行编程语言的标配，但它们往往被 PHP 开发人员所忽视。像 Ruby 就是一个极度重视异常的语言，无论有什么错误发生，像是 HTTP 请求失败，或者数据库查询有问题，甚至找不到一个图片资源，Ruby （或是所使用的 gems），将会抛出异常，你可以通过屏幕立刻知道所发生的问题。

PHP 处理这个问题则比较随意，调用 `file_get_contents()` 函数通常只会给出 `FALSE` 值和警告。许多较早的 PHP 框架比如 CodeIgniter 只是返回 false，将信息写入专有的日志，或者让你使用类似 `$this->upload->get_error()` 的方法来查看错误原因。这里的问题在于你必须找出错误所在，并且通过翻阅文档来查看这个类使用了什么样的错误的方法，而不是明确的暴露错误。

另一个问题发生在当类自动抛出错误到屏幕时会结束程序。这样做会阻挡其他开发者动态处理错误的机会。应该抛出异常让开发人员意识到错误的存在，让他们可以选择处理的方式，例如：

```
{% highlight 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)
{
    // 验证失败
}
catch(Fuel\Email\SendingFailedException $e)
{
    // 这个驱动无法发送 email
}
finally
{
    // 无论抛出什么样的异常都会执行，并且在正常程序继续之前执行
}
{% endhighlight %}
```

SPL 异常

原生的 `Exception` 类并没有提供太多的调试情境给开发人员，不过可以通过建立一个特殊的 `Exception` 来弥补它，方式就是建立一个继承自原生 `Exception` 类的一个子类：

```
{% highlight php %}
<?php
class ValidationException extends Exception {}
{% endhighlight %}
```

如此一来，可以加入多个 catch 区块，并且根据不同的异常分别处理。通过这样可以建立 *许多*自定义异常，其中有些已经在 SPL 扩展 提供的 SPL 异常中定义了。

举例来说，如果你使用了 `__call()` 魔术方法去调用一个无效的方法，而不是抛出一个模糊的标准 Exception 或是建立自定义的异常处理，你可以直接抛出 `throw new BadMethodCallException;` 。

- Read about Exceptions
- Read about SPL Exceptions
- Nesting Exceptions In PHP
- Exception Best Practices in PHP 5.3

10

安全

Web 应用程序安全

攻击者无时无刻不在准备对你的 Web 应用程序进行攻击，因此提高你的 Web 应用程序的安全性是非常有必要的。幸运的是，来自开放式 Web 应用程序安全项目 (OWASP) 的有心人已经整理了一份包含了已知安全问题和防御方式的全面的清单。这份清单对于具有安全意识的开发者来说是必读的。

- 阅读 OWASP 安全指南

密码哈希

每个人在建构 PHP 应用时终究都会加入用户登录的模块。用户的帐号及密码会被储存在数据库中，在登录时用来验证用户。

在存储密码前正确的哈希密码是非常重要的。哈希密码是单向不可逆的，该哈希值是一段固定长度的字符串且无法逆向推算出原始密码。这就代表你可以哈希另一串密码，来比较两者是否是同一个密码，但又无需知道原始的密码。如果你不将密码哈希，那么当未授权的第三者进入你的数据库时，所有用户的帐号资料将会一览无遗。有些用户可能（很不幸的）在别的网站也使用相同的密码。所以务必要重视数据安全的问题。

使用 `password_hash` 来哈希密码

`password_hash` 函数在 PHP 5.5 时被引入。 此函数现在使用的是目前 PHP 所支持的最强大的加密算法 BCrypt 。 当然，此函数未来会支持更多的加密算法。 `password_compat` 库的出现是为了提供对 PHP >= 5.3.7 版本的支持。

在下面例子中，我们哈希一个字符串，然后和新的哈希值对比。因为我们使用的两个字符串是不同的（'secret-password' 与 'bad-password'），所以登录失败。

```
{% highlight php %}
<?php
require 'password.php';

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

if (password_verify('bad-password', $passwordHash)) {
    // Correct Password
} else {
    // Wrong password
}
{% endhighlight %}
```

- 了解 `password_hash()`
- PHP >= 5.3.7 && < 5.5 的 `password_compat`
- 了解密码学中的哈希
- PHP `password_hash()` RFC

数据过滤

永远不要信任外部输入。请在使用外部输入前进行过滤和验证。`filter_var()` 和 `filter_input()` 函数可以过滤文本并对格式进行校验（例如 email 地址）。

外部输入可以是任何东西：`$_GET` 和 `$_POST` 等表单输入数据，`$_SERVER` 超全局变量中的某些值，还有通过 `fopen('php://input', 'r')` 得到的 HTTP 请求体。记住，外部输入的定义并不局限于用户通过表单提交的数据。上传和下载的文档，session 值，cookie 数据，还有来自第三方 web 服务的数据，这些都是外服输入。

虽然外部输入可以被存储、组合并在以后继续使用，但它依旧是外部输入。每次你处理、输出、连结或在代码中包含时，请提醒自己检查数据是否已经安全地完成了过滤。

数据可以根据不同的目的进行不同的 *过滤* 。比如，当原始的外部输入被传入到了 HTML 页面的输出当中，它可以在你的站点上执行 HTML 和 JavaScript 脚本！这属于跨站脚本攻击（XSS），是一种很有杀伤力的攻击方式。一种避免 XSS 攻击的方法是在输出到页面前对所有用户生成的数据进行清理，使用 `strip_tags()` 函数来去除 HTML 标签或者使用 `htmlentities()` 或是 `htmlspecialchars()` 函数来对特殊字符分别进行转义从而得到各自的 HTML 实体。

另一个例子是传入能够在命令行中执行的选项。这是非常危险的（同时也是一个不好的做法），但是你可以使用自带的 `escapeshellarg()` 函数来过滤执行命令的参数。

最后的一个例子是接受外部输入来从文件系统中加载文件。这可以通过将文件名修改为文件路径来进行利用。你需要过滤掉 `"/"`，`"../"`，null 字符或者其他文件路径的字符来确保不会去加载隐藏、私有或者敏感的文件。

- 学习更多数据过滤
- 学习更多 `filter_var`
- 学习更多 `filter_input`
- 学习更多 null 字符问题

数据清理

数据清理是指删除（或转义）外部输入中的非法和不安全的字符。

例如，你需要在将外部输入包含在 HTML 中或者插入到原始的 SQL 请求之前对它进行过滤。当你使用 PDO 中的限制参数功能时，它会自动为你完成过滤的工作。

有些时候你可能需要允许一些安全的 HTML 标签输入进来并被包含在输出的 HTML 页面中，但这实现起来并不容易。尽管有一些像 HTML Purifier 的白名单类库为了这个原因而出现，实际上更多的人通过使用其他更加严格的格式限制方式例如使用 Markdown 或 BBCode 来避免出现问题。

查看 Sanitization Filters

有效性验证

验证是来确保外部输入的是你所想要的内容。比如，你也许需要在处理注册申请时验证 email 地址、手机号码或者年龄等信息的有效性。

查看 Validation Filters

配置文件

当你在为你的应用程序创建配置文件时，最好的选择时参照以下的做法：

- 推荐你将你的配置信息存储在无法被直接读取和上传的位置上。
- 如果你一定要存储配置文件在根目录下，那么请使用 `.php` 的扩展名来进行命名。这将可以确保即使脚本被直接访问到，它也不会被以明文的形式输出出来。
- 配置文件中的信息需要被针对性的保护起来，对其进行加密或者设置访问权限。

注册全局变量

注意： 自 PHP 5.4.0 开始，`register_globals` 选项已经被移除并不再使用。这是在提醒你如果你正在升级旧的应用程序的话，你需要注意这一点。

当 `register_globals` 选项被开启时，它会使许多类型的变量（包括 `$_POST`，`$_GET` 和 `$_REQUEST`）被注册为全局变量。这将很容易使你的程序无法有效地判断数据的来源并导致安全问题。

例如：`$_GET['foo']` 可以通过 `$foo` 被访问到，也就是可以对未声明的变量进行覆盖。如果你使用低于 5.4.0 版本的 PHP 的话，请 **确保** `register_globals` 是被设为 `off` 的。

- 在 PHP 手册中了解 Register_globals

错误报告

错误日志对于发现程序中的错误是非常有帮助的，但是有些时候它也会将应用程序的结构暴露给外部。为了有效的保护你的应用程序不受到由此而引发的问题。你需要将在你的服务器上使用开发和生产（线上）两套不同的配置。

开发环境

为了在**开发**环境中显示所有可能的错误，将你的 `php.ini` 进行如下配置：

```ini
{% highlight ini %}
display_errors = On
display_startup_errors = On
error_reporting = -1
log_errors = On
{% endhighlight %}
```

> 将值设为 `-1` 将会显示出所有的错误，甚至包括在未来的 PHP 版本中新增加的类型和参数。 和 PHP 5.4 起开始使用的 `E_ALL` 是相同的。- php.net

`E_STRICT` 类型的错误是在 5.3.0 中被引入的，并没有被包含在 `E_ALL` 中。然而从 5.4.0 开始，它被包含在了 `E_ALL` 中。这意味着什么？这表示如果你想要在 5.3 中显示所有的错误信息，你需要使用 `-1` 或者 `E_ALL | E_STRICT`。

不同 PHP 版本下开启全部错误显示

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

生产环境

为了在**生产**环境中隐藏错误显示，将你的 `php.ini` 进行如下配置：

```ini
{% highlight ini %}
display_errors = Off
display_startup_errors = Off
error_reporting = E_ALL
```

```
log_errors = On
{% endhighlight %}
```

当在生产环境中使用这个配置时，错误信息依旧会被照常存储在 web 服务器的错误日志中，唯一不同的是将不再显示给用户。更多关于设置的信息，请参考 PHP 手册：

- 错误报告
- 显示错误
- 显示启动错误
- 记录错误

11

测试

为你的 PHP 程序编写自动化测试被认为是最佳实践，可以帮助你建立良好的应用程序。 自动化测试是非常棒的工具，它能确保你的应用程序在改变或增加新的功能时不会影响现有的功能，不应该忽视。

PHP 有一些不同种类的测试工具（或框架）可以使用，它们使用不同的方法 - 但他們都试图避免手动测试和大型 QA 团队的需求，确保最近的变更不会破坏既有功能。

测试驱动开发

Wikipedia 上的定义:

> 测试驱动开发（TDD）是一种以非常短的开发周期不断迭代的软件开发过程:首先开发者对将要实现的功能或者新的方法写一个失败的自动化测试用例，然后就去写代码来通过这个测试用例，最终通过重构代码让一其达到可接受的水准。Kent Beck， 这个技术创造者或者说重新发现者，在2003年声明TDD 鼓励简单的设计和激励信心。

目前你可以应用的几种不同类型的测试:

单元测试

单元测试是一种编程方法来确认函数，类和方法以我们预期的方式来工作，单元测试会贯穿整个项目的开发周期。通过检查各个函数和方法的输入输出，你就可以保证内部的逻辑已经正确执行。通过使用依赖注入和编写"mock" 类以及 stubs 来确认依赖被正确的使用，提高测试覆盖率。

当你创建一个类或者一个函数，你应该为它们的每一个行为创建一个单元测试。至少你应该确认当你输入一个错误参数会触发一个错误，你输入一个有效的参数会得到正确的结果。这会帮助你在开发周期后段对类或者函数做出修改后，确认已有的功能任然可以正常的工作。可替代的方法是在源码中使用 `var_dump()` ，但这种方法却不能去构建一个或大或小的应用。

单元测试的其他用处是在给开源项目贡献代码时。如果你写了一个测试证明代码有bug，然后修复它，并且展示测试的过程，这样补丁将会更容易被接受。如果你在维护一个项目，在处理 pull request 的时候可以将单元测试作为一个要求。

PHPUnit 是业界PHP应用开发单元测试框架的标准，但也有其他可选的框架:

- atoum
- Enhance PHP
- PUnit
- SimpleTest

集成测试

Wikipedia 上的定义:

> 集成测试（有时候称为集成和测试，缩写为 I&T）是把各个模块组合在一起进行整体测试的软件测试阶段。它处于单元测试之后，验收测试之前。集成测试将已经经过了单元测试的模块做为输入模块，组合成一个整体，然后运行集成测试用例，然后输出一个可以进行系统测试的系统。

许多相同的测试工具既可以运用到单元测试，也可以运用到集成测试。

功能性测试

有时候也被称之为验收测试，功能测试是通过使用工具来生成自动化的测试用例，然后在真实的系统上运行。而不是单元测试中简单的验证单个模块的正确性和集成测试中验证各个模块间交互的正确性。这些工具会使用代表性的真实数据来模拟真实用户的行为来验证系统的正确性。

功能测试的工具

- Selenium
- Mink
- Codeception 是一个全栈的测试框架包括验收性测试工具。
- Storyplayer 是一个全栈的测试框架并且支持随时创建和销毁测试环境。

行为驱动开发

有两种不同的行为驱动开发 (BDD): SpecBDD 和 StoryBDD。 SpecBDD 专注于代码的技术行为，而 StoryBDD 专注于业务逻辑或功能的行为和互动。这两种 BDD 都有对应的 PHP 框架。

采用 StoryBDD 时，你编写可读的故事来描述应用程序的行为。接着这些故事可以作为应用程序的实际测试案例执行。Behat 是使用在 PHP 应用程序中的 StoryBDD框架，它受到 Ruby 的 Cucumber 项目的启发并且实现了 Gherkin DSL 来描述功能的行为。

采用 SpecBDD 时，你编写规格来描述实际的代码应该有什么行为。你应该描述函数或者方法应该有什么行为，而不是测试函数或者方法。PHP 提供了 PHPSpec 框架来达到这个目的，这个框架受到了 Ruby 的 RSpec project 项目的启发。

BDD 链接

- Behat, PHP 的 StoryBDD 框架， 受到了 Ruby's Cucumber 项目的启发。
- PHPSpec, PHP 的 SpecBDD 框架， 受到了 Ruby's RSpec 项目的启发。
- Codeception 是一个使用 BDD 准则的全栈测试框架。

其他测试工具

除了个别的测试驱动和行为驱动框架之外，还有一些通用的框架和辅助函数类库，对任何的测试方法都很有用。

工具地址

- Selenium 是一个浏览器自动化工具 integrated with PHPUnit
- Mockery 是一个可以跟 PHPUnit 或者 PHPSpec 整合的 Mock 对象框架
- Prophecy 是个有自己的想法，且非常强大灵活的 PHP 对象 mocking 框架。它整合了 PHPSpec 并且可以和 PHPUnit 一起使用

12

服务器与部署

部署 PHP 应用程序到生产环境中有多种方式。

Platform as a Service (PaaS)

PaaS 提供了运行 PHP 应用程序所必须的系统环境和网络架构。这就意味着只需做少量配置就可以运行 PHP 应用程序或者 PHP 框架。

现在，PaaS 已经成为一种部署、托管和扩展各种规模的 PHP 应用程序的流行方式。你可以在 资源部分 查看 PHP PaaS "Platform as a Service" 提供商 列表。

虚拟或专用服务器

如果你喜欢系统管理员的工作，或者对这方面感兴趣，虚拟或者专用服务器可以让你完全控制自己的生产环境。

nginx 和 PHP-FPM

PHP 通过内置的 FastCGI 进程管理器（FPM），可以很好的与轻量级的高性能 web 服务器 nginx 协作使用。nginx 比 Apache 占用更少内存而且可以更好的处理并发请求，这对于并没有太多内存的虚拟服务器尤其重要。

- 阅读更多 nginx 的内容
- 阅读更多 PHP-FPM 的内容
- 学习如何配置安全的 nginx 和 PHP-FPM

Apache 和 PHP

PHP 和 Apache 有很长的合作历史。Apache 有很强的可配置性和大量的 扩展模块 。是共享主机中常见的Web服务器，完美支持各种 PHP 框架和开源应用(如 WordPress)。可惜的是，默认情况下，Apache 会比 nginx 消耗更多的资源，而且并发处理能力不强。

Apache 有多种方式运行 PHP，最常见的方式就是使用 mode_php5 的 prefork MPM 方式。但是它对内存的利用效率并不高，如果你不想深入服务器管理方面学习，那么这种简单的方式可能是你最好的选择。需要注意的事如果你使用 mod_php5，就必须使用 prefork MPM。

如果你追求高性能和高稳定性，可以为 Apache 选择与 nginx 类似的的 FPM 系统 worker MPM 或者 event MPM，它们分别使用 mod_fastcgi 和 mod_fcgid。这种方式可以更高效的利用内存，运行速度也更快，但是配置也相对复杂一些。

- 阅读更多 Apache
- 阅读更多多进程模块
- 阅读更多 mod_fastcgi
- 阅读更多 mod_fcgid

共享主机

PHP 非常流行，很少有服务器没有安装 PHP 的，因而有很多共享主机，不过需要注意服务器上的 PHP 是否是最新稳定 版本。共享主机允许多个开发者把自己的网站部署在上面，这样的好处是费用非常便宜，坏处是你不知道将和哪些 网站共享主机，因此需要仔细考虑机器负载和安全问题。如果项目预算允许的话，避免使用共享主机是上策。

构建及部署应用

如果你在手动的进行数据库结构的修改或者在更新文件前手动运行测试，请三思而后行！因为随着每一个额外的手动任务的添加都需要去部署一个新的版本到应用程序，这些更改会增加程序潜在的致命错误。即使你是在处理一个简单的更新，全面的构建处理或者持续集成策略，构建自动化绝对是你的朋友。

你可能想要自动化的任务有：

- 依赖管理
- 静态资源编译、压缩
- 执行测试
- 文档生成
- 打包
- 部署

构建自动化工具

构建工具可以认为是一系列的脚本来完成应用部署的通用任务。构建工具并不属于应用的一部分，它独立于应用层 '之外'。

现在已有很多开源的工具来帮助你完成构建自动化，一些是用 PHP 编写，有一些不是。应该根据你的实际项目来选择最适合的工具，不要让语言阻碍了你使用这些工具，如下有一些例子：

Phing 是一种在 PHP 领域中最简单的开始自动化部署的方式。通过 Phing 你可以控制打包，部署或者测试，只需要一个简单的 XML 构建文件。Phing（基于Apache Ant）提供了在安装或者升级 web 应用时的一套丰富的任务脚本，并且可以通过 PHP 编写额外的任务脚本来扩展。

Capistrano 是一个为 *中高级程序员* 准备的系统，以一种结构化、可复用的方式在一台或多台远程机器上执行命令。对于部署 Ruby on Rails 的应用，它提供了预定义的配置，不过也可以用它来 **部署 PHP 应用** 。如果要成功的使用 Capistrano ，需要一定的 Ruby 和 Rake 的知识。

对 Capistrano 感兴趣的 PHP 开发者可以阅读 Dave Gardner 的博文 PHP Deployment with Capistrano ，来作为一个很好的开始。

Chef 不仅仅只是一个部署框架， 它是一个基于 Ruby 的强大的系统集成框架，除了部署你的应用之外，还可以构建整个服务环境或者虚拟机。

Deployer 是一个用 PHP 编写的部署工具，它很简单且实用。并行执行任务，原子化部署，在多台服务器之间保持一致性。为 Symfony、Laravel、Zend Framework 和 Yii 提供了通用的任务脚本。

适用于 PHP 开发者的 Chef 资源：

- Three part blog series about deploying a LAMP application with Chef, Vagrant, and EC2
- Chef Cookbook which installs and configures PHP 5.3 and the PEAR package management system
- Chef video tutorial series by Opscode, the makers of chef

延伸阅读：

- Automate your project with Apache Ant

持续集成

> 持续集成是一种软件开发实践，团队的成员经常用来集成他们的工作， 通常每一个成员至少每天都会进行集成 — 因此每天都会有许多的集成。许多团队发现这种方式会显著地降低集成问题， 并允许一个团队更快的开发软件。
>
> -- *Martin Fowler*

对于 PHP 来说，有许多的方式来实现持续集成。近来 Travis CI 在持续集成上做的很棒，对于小项目来说也可以很好的使用。Travis CI 是一个托管的持续集成服务用于开源社区。它可以和 Github 很好的集成，并且提供了很多语言的支持包括 PHP 。

延伸阅读：

- 使用 Jenkins 进行持续集成
- 使用 PHPCI 进行持续集成
- 使用 Teamcity 进行持续集成

13

虚拟化技术

在开发和线上阶段使用不同的系统运行环境的话，经常会遇到各种各样的 BUG，并且在团队开发的时候，让所有成员都保持使用最新版本的软件和类库，也是一件很让人头痛的事情.

如果你是在 Windows 下开发，线上环境是 Linux（或者别的非 Windows 系统）的话，或者团队协同开发的时候，建议使用虚拟机.

除了大家熟知的 VMware 和 VirtualBox 外，还有很多工具可以让你快速，轻松的用上虚拟环境.

Vagrant 简介

Vagrant 可以让你使用单一的配置信息来部署一套虚拟环境，最后打包为一个所谓的 box（就是已经部署好环境的虚拟机器）. 你可以手动来安装和配置 box，也可以使用自动部署工具，如 Puppet 或者 Chef .

自动部署工具可以让你快速部署一套一模一样的环境，避免了一大堆的手动的命令输入，并且允许你随时删除和重建一个全新的 box，虚拟机的管理变得更加简单.

Vagrant 还可以在虚拟机和主机上分享文件夹，意味着你可以在主机里面编辑代码，然后在虚拟机里面运行.

需要更多的帮助?

下面是一些其他的软件，可以帮助你更好的使用 Vagrant:

- Rove: 使用 Chef 自动化安装一些常用的软件，PHP 包含在内.
- Puphpet: 简单的 Web 图形界面用来生成部署 PHP 环境的 Puppet 脚本，此项目不仅可以用在开发上，也可以在生产环境中使用.
- Protobox: 是一个基于 vagrant 的一个层，还有 Web 图形界面，允许你使用一个 YAML 文件来安装和配置虚拟机里面的软件.
- Phansible: 提供了一个简单的 Web 图形界面，用来创建 Ansible 自动化部署脚本，专门为 PHP 项目定制.

Docker 简介

除了 Vagrant, Docker 是另一个实现生产和开发环境统一的非常棒的方案.

Docker 为各种应用程序提供了 Linux 容器.

你可以安装 Docker 镜像, 如 MySQL 和 PostgreSQL 等, 并且不会污染到你的本地机器, 可以看下 Docker Hub Registry, 在这里你可以找到你想要的, 提前配置好的, 允许你简单几部就能运行起来的 Linux 容器.

例子: 在 Docker 里面运行 PHP 应用

在你成功 安装 Docker 后, 你只需要一步就可以安装 Apache + PHP.

下面的命令, 会下载一个功能齐全的 Apache 和 最新版本的 PHP, 并会设置 WEB 目录 `/path/to/your/php/files` 运行在 `http://localhost:8080`:

```
{% highlight console %}
docker run -d --name my-php-webserver -p 8080:80 -v /path/to/your/php/files:/var/www/html/ php:apache
{% endhighlight %}
```

在使用 `docker run` 命令以后, 如果你想停止, 或者再次开启容器的话, 只需要执行以下命令:

```
{% highlight console %}
docker stop my-php-webserver
{% endhighlight %}

{% highlight console %}
docker start my-php-webserver
{% endhighlight %}
```

了解更多关于 Docker 的信息

The commands mentioned above only show a quick way to run an Apache web server with PHP support but there are a lot more things that you can do with Docker.

上面的命令能让你轻松使用 Apache + PHP 环境, 然而, Docker 还提供了好多别的命令, 例如, 作为 PHP 程序员, 一个最重要的事情, 是让你的 Web Server 和数据库链接起来, 怎么实现可以仔细看下 Docker User Guide.

- Docker Website

- Docker Installation
- Docker Images at the Docker Hub Registry
- Docker User Guide

14

缓存

PHP 本身来说是非常快的，但是但你当发起远程连接、加载文件等操作时也会遇到瓶颈。 幸运的是，有各种各样的工具可以用来加速你应用程序某些耗时的部分，或者说减少某些耗时任务所需要运行的次数。

Opcode 缓存

当一个 PHP 文件被解释执行的时候，首先是被编译成名为 opcode 的中间代码，然后才被底层的虚拟机执行。如果PHP文件没有被修改过，opcode 始终是一样的。这就意味着编译步骤白白浪费了 CPU 的资源。

此时 opcode 缓存就派上用场了。通过将 opcode 缓存在内存中，它能防止冗余的编译步骤，并且在下次调用执行时得到重用。设置 opcode 缓存只需要几分钟的时间，你的应用程序便会因此大大加速，实在没有理由不用它。

PHP 5.5 中自带了 opcode 缓存工具，叫做OPcache，早期的版本也能通过一定的配置使用它。 更多关于 opcode 缓存的资料： * OPcache (built-in since PHP 5.5) * APC (PHP 5.4 and earlier) * XCache * Zend Optimizer+ (part of Zend Server package) * WinCache (extension for MS Windows Server) * list of PHP accelerators on Wikipedia

对象缓存

有时缓存代码中的单个对象会很有用，比如有些需要很大开销获取的数据或者一些结果集不怎么变化的数据库查询。你可以使用一些缓存软件将这些数据存放在内存中以便下次高速获取。如果你获得数据后把他们存起来，下次请求直接从缓存里面获取数据，在减少数据库负载的同时能极大提高性能。

许多流行的字节码缓存方案也能缓存定制化的数据，所以更有理由好好使用它们了。APCu、XCache 以及 WinCache 都提供了 API，以便你将数据缓存到内存中

最常用的内存对象缓存系统是 APCu 和 Memcached 。APCu 对于对象缓存来说是个很好的选择，它提供了简单的 API 让你能将数据缓存到内存，并且很容易设置和使用。APCu 的局限性表现在它依赖于所在的服务器。另一方面，Memcached 以独立的服务的形式安装，可以通过网络交互，这意味着你能将数据集中存在一个高速存取的地方，而且许多不同的系统能从中获取数据。

值得注意的是当你以 CGI(FastCGI) 的形式使用 PHP 时，每个进程将会有各自的缓存，比如说，APCu 缓存数据无法在多个工作进程中共享。在这种情况下，你可能得考虑 Memcached 了，由于它独立于 PHP 进程。

通常 APCu 在存取速度上比 Memcached 更快，但是 Memcached 在扩展上更有优势。如果你不希望应用程序涉及多个服务器，或者不需要 Memcached 提供的其他特性，那么 APCu 可能是最好的选择。

使用 APCu 的例子：

```
{% highlight php %}
<?php
// check if there is data saved as 'expensive_data' in cache
$data = apc_fetch('expensive_data');
if ($data === false) {
    // data is not in cache; save result of expensive call for later use
    apc_add('expensive_data', $data = get_expensive_data());
}
print_r($data);
{% endhighlight %}
```

注意在 PHP 5.5 之前，APC 同时提供了对象缓存与字节码缓存。APCu 是为了将 APC 的对象缓存移植到 PHP 5.5+ 的一个项目，因为现在 PHP 有了内建的字节码缓存方案 (OPcache)。

更多关于缓存系统的项目：

- APCu
- APC Functions

- Memcached
- Redis
- XCache APIs
- WinCache Functions

15

文档撰写

PHPDoc

PHPDoc 是注释 PHP 代码的非正式标准。它有许多不同的标记可以使用。完整的标记列表和范例可以查看 PHPDoc 指南。

如下是撰写类方法时的一种写法：

```
{% highlight php %}
<?php
/**
 * @author A Name <a.name@example.com>
 * @link http://www.phpdoc.org/docs/latest/index.html
 */
class DateTimeHelper
{
    /**
     * @param mixed $anything Anything that we can convert to a \DateTime object
     *
     * @throws \InvalidArgumentException
     *
     * @return \DateTime
     */
    public function dateTimeFromAnything($anything)
    {
        $type = gettype($anything);

        switch ($type) {
            // Some code that tries to return a \DateTime object
        }

        throw new \InvalidArgumentException(
            "Failed Converting param of type '{$type}' to DateTime object"
        );
    }

    /**
     * @param mixed $date Anything that we can convert to a \DateTime object
     *
     * @return void
     */
    public function printISO8601Date($date)
    {
        echo $this->dateTimeFromAnything($date)->format('c');
```

```
    }

    /**
     * @param mixed $date Anything that we can convert to a \DateTime object
     */
    public function printRFC2822Date($date)
    {
        echo $this->dateTimeFromAnything($date)->format('r');
    }
}
{% endhighlight %}
```

这个类的说明使用了 @author 和 @link标记， @author 标记是用來说明代码的作者，在多位开发者的情况下，可以同时列出好几位。其次 @link 标记用来提供网站链接，进一步说明代码和网站之间的关系。

在这个类中，第一个方法的 @param 标记，说明类型、名字和传入方法的参数。此外，@return 和 @throws 标记说明返回类型以及可能抛出的异常。

第二、第三个方法非常类似，和第一个方法一样使用一个 @param 标记。第二、和第三个方法之间关键差别在注释区块使用 / 排除 @return 标记。`@return void` 标记明确告诉我们没有返回值，而过去省略 `@return void` 声明也具有相同效果（没有返回任何值）。

16

资源

PHP 官方

- PHP 官方网站
- PHP 官方文档

值得关注的大牛

- Rasmus Lerdorf
- Fabien Potencier
- Derick Rethans
- Chris Shiflett
- Sebastian Bergmann
- Matthew Weier O'Phinney
- Pádraic Brady
- Anthony Ferrara
- Nikita Popov

指导

- phpmentoring.org - PHP 社区中的一对一指导。

PHP 的 Paas 提供商

- PagodaBox
- AppFog
- Heroku
- fortrabbit
- Engine Yard Cloud
- Red Hat OpenShift Platform
- dotCloud
- AWS Elastic Beanstalk
- cloudControl
- Windows Azure
- Google App Engine
- Jelastic

框架

许多的 PHP 开发者都使用框架，而不是重新造轮子来构建 Web 应用。框架抽象了许多底层常用的逻辑，并提供了有益又简便的方法來完成常见的任务。

你并不一定要在每个项目中都使用框架。有时候原生的 PHP 才是正确的选择，但如果你需要一个框架，那么有如下三种主要类型：

- 微型框架
- 全栈框架
- 组件框架

微型框架基本上是一个封装的路由，用来转发 HTTP 请求至一个闭包，控制器，或方法等等，尽可能地加快开发的速度，有时还会使用一些类库来帮助开发，例如一个基本的数据库封装等等。他們用来构建 HTTP 的服务卓有成效。

许多的框架会在微型框架上加入相当多的功能，我们则称之为全栈框架。这些框架通常会提供 ORMs ，身份认证扩展包等等。

组件框架是多个独立的类库所结合起来的。不同的组件框架可以一起使用在微型或是全栈框架上。

- 热门的 PHP 框架

组件

正如标题提到的，「组件」是另一种建立，发布及推动开源的方式。现在存在的各种的组件库，其中最主要的两个为：

- Packagist
- PEAR

这两个组件库都有用來安装及升级的命令行工具，这部分已经在這部分已經在[依赖管理]中解释过。

此外，还有基于组件的构成的框架的提供商提供不包含框架的组件。这些项目通常和其他的组件或者特定的框架没有依赖关系。

例如，你可以使用 [FuelPHP 验证类库]，而不使用 FuelPHP 整个框架。

- Aura
- FuelPHP
- Hoa Project
- Orno
- Symfony Components
- The League of Extraordinary Packages
- Laravel's Illuminate Components
    - Eloquent ORM
    - Queue

Laravel 的 [Illuminate 组件] 和 Laravel 框架将变得更加解耦。 现在我们只列出和 Laravel 框架最没有依赖关系的组件。

其他有用的资源

Cheatsheets

- PHP Cheatsheets - for variable comparisons, arithmetics and variable testing in various PHP versions
- PHP Security Cheatsheet

更多最佳实践

- PHP Best Practices
- Best practices for Modern PHP Development

PHP 世界

- PHP Developer blog

Video Tutorials

Youtube 视频

- PHP Academy
- The New Boston
- Sherif Ramadan
- Level Up Tuts

付费视频

- Standards and Best practices
- PHP Training on Pluralsight
- PHP Training on Lynda.com
- PHP Training on Tutsplus
- Laracasts

书籍

市面上有很多关于 PHP 的书，但遗憾的是很多都已经非常陈旧而且不正确的资料。甚至还有出版商发布「 PHP 6 」，这是不存在的书，而且永远不会出现。因为那些书，所以 PHP 的下一个版本为「 PHP 7 」。

这个章节的目录主要是针对 PHP 开发，并且会随着最新的技术趋势而更新。如果你想在这里加入你的书，请发送一个 PR ，我们将会审查你提供的内容是否有相关性。

免费书籍

- PHP The Right Way - This website is available as a book completely for free.

付费书籍

- Build APIs You Won't Hate - Everyone and their dog wants an API, so you should probably learn how to build them.
- Building Secure PHP Apps - Learn the security basics that a senior developer usually acquires over years of experience, all condensed down into one quick and easy handbook
- Modernizing Legacy Applications In PHP - Get your code under control in a series of small, specific steps
- Securing PHP: Core Concepts - A guide to some of the most common security terms and provides some examples of them in every day PHP
- Scaling PHP - Stop playing sysadmin and get back to coding
- Signaling PHP - PCNLT signals are a great help when writing PHP scripts that run from the command line.
- The Grumpy Programmer's Guide To Building Testable PHP Applications - Learning to write testable code doesn't have to suck

17

社区

PHP 社区多元化并且规模庞大，成员们也乐意并随时准备好帮助新人。你可以考虑加入当地的 PHP 使用者社区（PUG） 或者参加教大型的 PHP 会议，从中学习更多最佳实践。你也可以使用 IRC 逛逛 irc.freenode.com 上的 #phpc 频道，也可以关注 @phpc 的Twitter 账号。试着去多结交一些新的开发者，学习新的东西，总之，交一些新朋友！其他的社区资源包含 Google+ 的 PHP Programmer community 以及 StackOverflow。

阅读 PHP 官方事件日历

PHP 用户群

如果你住在较大的城市，附近应该就有 PHP 用户群。你可以通过基于 PHP.ug 的 usergroup-list at php.net 这个地址找到当地的 PUG。也可以通过 Meetup.com 或者使用搜索引擎（i.e. Google）搜索 `php user group near me` 。如果你住在比较小的城镇，当地也许还没有 PUG ，如果是这种情形，不妨就开始组建一个。

这里要特别提到两个全球的用户组：NomadPHP 和 PHPWomen。NomadPHP 提供每月两次的在线用户组会议，由 PHP 社区里顶尖的高手进行演讲。PHPWomen 原本是针对女性 PHP 开发者的非排他性的用户组。会员资格发放给那些支持多元化社区的人。PHPWomen 提供了一技术支持，指导和教育的个平台，并且促进了女性的创造力以及专业的氛围。

了解关于 PHP Wiki 上的用户群

PHP 会议

世界各地的 PHP 社区也会举办一些较大型的区域性或国际性的会议，一些知名的社区成员通常会在这些大型活动中现身演讲，这是一个直接和业内领袖学习的好机会。

查找 PHP 会议

ElePHPants

ElePHPant is that beautiful mascot of the PHP project with elephant in their design. It was originally designed for the PHP project in 1998 by Vincent Pontier - spiritual father of thousands of elePHPants around the world and 10 years later adorable plush elephant toy came to birth as well. Now elePHPants are present at many PHP conferences and with many PHP developers at their computers for fun and inspiration.

Interview with Vincent Pontier