

Notes Collection of PHP

PHP

THEQIONG.COM

theqiong.com

PHP Notes

theqiong.com

2018 年 3 月 6 日

目录

Cover	1
I Introduction	13
1 Overview	15
1.1 History	17
1.2 Dynamic Parser	20
1.3 Template Engine	21
1.3.1 Plates	21
1.3.2 Smarty	22
1.4 Dynamic Tracing	23
1.5 Garbage Collection	31
1.5.1 Reference Counting	36
1.5.2 Collecting Cycles	39
1.6 Development Server	42
1.6.1 Router	43
1.6.2 Template	44
1.7 Closure/Anonymous	44
1.8 Meta Programming	45
1.9 Functional Program	45
1.10 Dependency Injection	48
1.10.1 Inversion of Control	49
1.10.2 Dependency Inversion	50
1.11 Framework Container	50
1.12 Commandline Interface	51

1.12.1	\$argv	53
1.12.2	\$argv	54
1.12.3	shibang	54
2	Syntax	57
2.1	Constructs	57
2.2	Delimiters	58
2.3	Variables	58
2.4	Structures	58
3	Encoding	61
3.1	String	61
3.1.1	strpos()	62
3.1.2	strlen()	62
3.1.3	substr()	62
3.1.4	mb_strpos()	62
3.1.5	mb_strlen()	62
3.1.6	mb_substr()	62
3.2	Browser	62
3.2.1	htmlentities()	62
3.2.2	htmlspecialchars()	62
3.2.3	mb_http_output()	62
3.2.4	mb_internal_encoding()	64
3.3	Database	64
4	Data types	65
4.1	Object	66
4.2	String	67
4.2.1	curl	67
4.2.2	pcre	67
4.2.3	pspell	67
4.3	Array	67
4.3.1	wddx	67
4.4	Resource	68

5	Functions	69
6	Objects	73
7	Filesystem	77
7.1	Directory	77
7.2	Journal	77
7.3	File	77
8	Protocol	79
9	Development	81
9.1	Interface	81
9.2	Kernel	81
9.3	PEAR	81
9.4	PECL	82
9.4.1	pecl	82
9.4.2	phpize	83
9.4.3	php-config	83
9.5	CGI	85
10	Installation	87
10.1	UNIX	87
10.2	Mac OSX	91
10.2.1	Homebrew	91
10.2.2	MacPorts	91
10.2.3	PHPBrew	91
10.3	Windows	93
10.3.1	Compile	94
10.3.2	Install	95
10.3.3	Manual	95
10.3.4	Update	96
10.3.5	Extend	97
10.3.6	Config	99
10.3.7	Regedit	100

10.4 Docker	101
11 Accelerator	103
11.1 Overview	103
11.1.1 eAccelerator	103
11.2 Encoder	105
11.3 Engine	105
11.4 Compiler	106
11.5 WSGI	106
12 Runtime	109
12.1 php.ini	109
12.2 php-SAPI.ini	110
12.3 .user.ini	110
12.4 directive	112
13 Security	115
13.1 Remote files	115
13.2 Authentication	116
13.3 Password Hash	120
13.4 Variable filter	121
13.5 Variable clean	121
13.6 Validity Verify	122
13.7 Configure file	122
13.8 Global Variable	122
13.9 Error Reporting	122
13.10 Upload Handling	123
13.10.1 memory_limit	124
13.10.2 post_max_size	124
13.10.3 upload_tmp_dir	124
13.10.4 max_input_time	125
13.10.5 upload_max_filesize	126
13.10.6 max_execution_time	126
13.11 Batch Uploading	126

13.12 Request Method	127
13.12.1 GET	127
13.12.2 POST	127
13.12.3 PUT	127
13.13 Connection handling	129
13.14 Persistent connections	129
13.15 Thread Safe	131
13.16 Safe Mode	132
13.16.1 safe_mode	132
13.16.2 safe_mode_gid	132
13.16.3 safe_mode_exec_dir	132
13.16.4 safe_mode_include_dir	133
13.16.5 safe_mode_allowed_env_vars	133
13.16.6 safe_mode_protected_env_vars	133
13.16.7 disable_functions	134
13.16.8 open_basedir	134

插图

1.1	array zval	33
1.2	修改 array zval	34
1.3	把数组作为一个元素添加到自己	35
1.4	删除数组元素	36
1.5	内存回收算法	37
1.6	GC 内存使用情况	40

表格

1.1	PHP DTrace 静态探针	24
1.2	CLI SAPI cli_server.color 配置选项	43
1.3	CLI SAPI 强制覆盖的设置选项	52
9.1	PEAR 计划	82
9.2	php-config 脚本命令行选项	84
10.1	PHP5 和 PHP4 的文件对应关系	94
10.2	PHP 扩展库	97
12.1	PHP_INI_* 模式的定义	111
13.1	保安措施和安全模式配置指令	132

Part I

Introduction

Chapter 1

Overview

PHP¹是一种开源的通用计算机脚本语言，尤其适用于网络开发（并可嵌入 HTML 中使用）。

PHP 支持 CGI²标准，因此 PHP 可以作为 CGI 处理器来使用。具体来说，PHP 可以实现让一个客户端从网页浏览器等向执行在网络服务器上的程序请求数据的功能，CGI 则是一种负责描述服务器和请求处理程序之间传输数据的标准。

PHP 针对大多数 HTTP 服务器都提供了相应的模块，而且 PHP 程序的运行性能和自身代码的模块性使其成为了一种高效率的选择。例如，PHP 内置的标准库（SPL）提供了一组类和接口来支持常用的数据结构类（例如堆栈、队列和堆等）以及遍历这些数据结构的迭代器，用户也可以自己实现 SPL 接口。

PHP Group 和开放源代码社区负责维护 PHP 标准，并以 PHP License 作为许可协议来发布 PHP（仅限制 PHP 名称的使用），这一点和 GPL（开放源代码许可协议）不兼容。

- 服务器端程序
 1. PHP 解析器（CGI 或服务器模块）
 2. Web 服务器（Nginx 和 Apache 等）
 3. Web 浏览器
- 命令行脚本
 1. PHP 命令行模式
 2. 定时任务（Cron 或 Task Scheduler）

“.php”是 PHP 的默认扩展名，以.php 结尾的文件都将由 PHP 来处理，只需要把.php 文件部署到支持 PHP 的服务器上的 Web 目录中，服务器将自动解析这些文件。

¹PHP: Hypertext Preprocessor（超文本预处理器）就是一种递归命名。

²CGI 是服务器和请求处理程序之间传输数据的一种标准。

PHP 项目文件无需编译也不用安装任何其它的工具，仅仅只需把 PHP 文件想象成简单的 HTML 文件即可，只不过多了一种新的标识符 `<?php ... ?>`，最终可以被解析成代码流并输出执行结果。

PHP 的语法借鉴了 C、Java 和 Perl 等语言，而且针对不同层次的用户提供了相关的特性（包括高级特性），因此 PHP 可以被应用到服务端应用和其他领域中。

PHP 的跨平台特性使其可以在多数的服务器和操作系统上运行，而且 PHP 针对大多数 HTTP 服务器都提供了相应的模块来增强互操作性，因此云计算平台（例如 Google App Engine、Amazon EC2 和 Microsoft Azure 等）也都提供了对 PHP 的支持。

PHP 的扩展机制使其除了可以向最终用户提供数据库、协议和 API 的基础结构之外，还允许开发者自己加入并提交新的模块。

PHP 可以用来收集表单数据、生成动态网页或者发送/接收 Cookies 等，而且 PHP 的功能并不局限于输出 HTML，还能被用来动态输出图像、PDF 文件、Flash 动画（使用 libswf 和 Ming）。其中，PHP 在处理 HTML 表单时，表单的任何元素都在 PHP 脚本中自动生效。

```
<form action="action.php" method="post">
<p>姓名: <input type="text" name="name" /></p>
<p>年龄: <input type="text" name="age" /></p>
<p><input type="submit" /></p>
</form>
```

当用户填写了该表单并点击了提交按钮后，页面 `action.php` 将被调用。

```
你好, <?php echo htmlspecialchars($_POST['name']); ?>。
你 <?php echo (int)$_POST['age']; ?> 岁了。
```

`htmlspecialchars()` 使得 HTML 之中的特殊字符被正确的编码，从而不会被使用者在页面注入 HTML 标签或者 Javascript 代码（例如 `age` 字段），也可以使用 PHP 的 `filter` 扩展来自动完成该工作，PHP 会自动设置 `$_POST['name']` 和 `$_POST['age']` 变量。

用户也可以在 PHP 中处理 XForms 的输入，尽管用户可能更喜欢使用长久以来支持良好的 HTML 表单。

PHP 可以非常简便的输出文本（例如 XHTML 以及任何其它形式的 XML 文件），PHP 还可以通过在服务端开辟出一块动态内容的缓存从而直接把它们打印出来，或者将它们存储到文件系统中。

PHP 支持使用 Perl 兼容正则表达式 (PCRE) 以及其他扩展和工具来解析和访问 XML 文档，同时 PHP 还将所有的 XML 功能标准化在 `libxml2` 库中，并且还增加了 `SimpleXML`，`XMLReader` 以及 `XMLWriter` 来支持并扩充功能。

PHP 可以在开发时选择使用过程和面向对象, 或者两者混合的方式来开发。例如, 从 PHP 5 开始引入了完全的对象模型, 弥补了 PHP 4 对 OOP 支持的不足。

PHP 对数据库的支持很全面, 用户可以使用任何针对某数据库的扩展 (例如 `mysql`) 编写数据库支持的网页, 也可以使用抽象层 (例如 `PDO`) 或者通过 `ODBC` 扩展连接到任何支持 `ODBC` 标准的数据库。其它一些数据库也可能会用 `cURL` 或者 `sockets` (例如 `CouchDB`)。

PHP 还支持利用 `LDAP`、`IMAP`、`SNMP`、`NNTP`、`POP3`、`HTTP`、`COM` (Windows 环境) 等协议的服务, 还可以通过开放原始网络端口来使得任何其它的协议能够协同工作。

PHP 支持和所有 Web 开发语言之间的 `WDDX` 复杂数据交换, 而且 PHP 支持 Java 对象的即时连接, 可以透明地将其用作 PHP 对象。

PHP 框架可以使项目得到更快更简单的部署和更加敏捷的开发效率, 因此在实际生产中框架的使用已非常普遍, 例如 PHP 官方的框架为 `Zend framework`。

在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提前实现。

1.1 History

PHP 最初是 `Rasmus Lerdorf` 为了要维护个人网页, 而用 C 语言开发的一些用以取代原先使用 `Perl` 封装的 `CGI` 工具程序集。这些工具程序用来显示 `Rasmus Lerdorf` 的个人履历以及统计网页流量。他将这些程序和一些窗体解释器集成起来, 称为 `PHP/FI`, 其中 `FI` 可以和数据库连接, 从而让 `PHP` 可以产生简单的动态网页程序。

`Rasmus Lerdorf` 在 1995 年 6 月 8 日将 `PHP/FI` 公开发布, 希望可以通过社区来加速程序开发与查找错误。这个发布的版本命名为 `PHP 2`, 已经有今日 `PHP` 的一些雏型, 像是类似 `Perl` 的变量命名方式、窗体处理功能、以及嵌入到 `HTML` 中运行的能力。程序语法上也类似 `Perl`, 有较多的限制, 不过更简单、更有弹性。

1997 年, `Zeev Suraski` 和 `Andi Gutmans` 不满足于 `PHP` 在大型项目中的表现, 从而重写了 `PHP` 的语法分析器, 这成为了 `PHP 3` 的基础, 而 `PHP` 也在这个时候改称为 `PHP: Hypertext Preprocessor`。

1997 年 11 月, 开发团队发布了 `PHP/FI 2`, 随后就开始 `PHP 3` 的开放测试, 并于 1998 年 6 月正式发布 `PHP 3`。

`Zeev Suraski` 和 `Andi Gutmans` 在 `PHP 3` 发布后开始改写 `PHP` 的核心, 这个在 1999 年发布的语法分析器称为 `Zend Engine`, 他们也在以色列的 `Ramat Gan` 成立了 `Zend Technologies` 来管理 `PHP` 的开发。

2000 年 5 月 22 日，以 Zend Engine 1.0 为基础的 PHP 4 正式发布，增加了数组操作函数、完整的会话机制、对输出缓存的支持等。

2004 年 7 月 13 日发布的 PHP 5 使用了第二代的 Zend Engine，而且 PHP5 包含了许多新特色，比如强化的面向对象功能、引入 PDO 以及许多性能上的增强。

从 2008 年开始，PHP 5 成为了 PHP 唯一维护中的稳定版本，PHP 4 已经不会继续更新，以鼓励用户迁移到 PHP 5。

PHP 7 的开发正在进行中，主要的改进有 PHPNG、JIT 引擎、抽象语法树编译、异步编程。

要查看当前 PHP 信息，可以通过网页或者命令行，其中通过网页可以直观的显示。

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

或者在命令行中输入如下的命令：

```
$ php -i
//or
$ php -r 'phpinfo();'
```

phpinfo() 函数可以得到很多有关当前系统的有用信息，例如预定义变量、已经加载的 PHP 模块和配置信息。

为了移除敏感信息，比如 AUTH_USER 和 AUTH_PASSWORD，可以修改代码如下：

```
<?php
// start output buffering
ob_start();

// send phpinfo content
phpinfo();

// get phpinfo content
$html = ob_get_contents();

// flush the output buffer
ob_end_clean();

// remove auth data
if(isset($_SERVER['AUTH_USER']))
    $html = str_replace($_SERVER['AUTH_USER'], '<i>no value</i>', $html);
if(isset($_SERVER['AUTH_PASSWORD']))
    $html = str_replace($_SERVER['AUTH_PASSWORD'], '<i>no value</i>', $html);
```

```
echo $html;  
?>
```

为了比较上述两种方法的异同，可以在一个页面中同时运行这两个测试页面：

```
<html>  
  <frameset cols="50%,50%">  
    <frame src="phptest.php">  
    <frame src="phpinfo.php">  
  </frameset>  
</html>
```

在兼容性方面，PHP 有可能影响到老版本的代码的最重要的两点改动分别是：

- 取消了旧的 `$HTTP_*_VARS` 数组（在函数或者方法中原本是全局变量）。

PHP 4.1.0 版本引入了如下超全局数组变量：

- `$_GET`

PHP 保留的超全局变量变量 `$_GET` 包含所有的 GET 数据。

- `$_POST`

PHP 保留的超全局变量 `$_POST` 包含所有的 POST 数据。

- `$_COOKIE`

PHP 保留的超全局变量 `$_COOKIE` 包含所有的 COOKIE 数据。

- `$_SERVER`

PHP 保留的超全局变量 `$_SERVER` 包含 Web 服务器提供的所有信息。

- `$_FILES`

PHP 保留的超全局变量 `$_FILES` 包含所有的 FILES 数据。

- `$_ENV`

PHP 保留的超全局变量 `$_ENV` 包含所有的 ENV 数据。

- `$_REQUEST`

PHP 保留的超全局变量 `$_REQUEST` 包含所有的 GET、POST 和 COOKIE 的数据。

- `$_SESSION`

PHP 保留的超全局变量 `$_SESSION` 包含所有的 SESSION 数据。

有些 `$HTTP_*_VARS` 数组（诸如 `$HTTP_POST_VARS` 等）从 PHP 3 就已经开始使用，从 PHP 5.4.0 开始将不再有效。

自 PHP 5.0.0 起，用 `register_long_arrays` 设置选项可禁用长类型的 PHP 预定义变量数组。

- 外部变量不再被默认注册为全局变量。也就是说，从 PHP 4.2.0 版开始，`php.ini` 中的设置选项 `register_globals` 默认值变成了 `off`。

建议用以上提到的超全局数组变量来访问这些值。

如果该选项被设置为 `on`, 则可以在 <http://www.example.com/foo.php?id=42> 中直接使用变量 `$id`。不过, 不管被设置为 `on` 还是 `off`, `$_GET['id']` 一直有效。

1.2 Dynamic Parser

无论以模块还是 CGI 的形式安装 PHP, PHP 的解释器都可以在服务器上执行访问文件、运行命令以及创建网络连接等操作, 而且这些操作可能产生安全隐患, 因此需要正确地安装和配置 PHP 以及编写安全的代码。

PHP 提供的 CLI SAPI 不会将当前目录改为已运行的脚本所在的目录。例如, 下面示例可以反映 CLI SAPI 模块与 CGI SAPI 模块之间的不同:

```
<?php
// 名为 test.php 的简单测试程序
echo getcwd(), "\n";
```

CGI SAPI 将当前目录修改为运行 PHP 脚本所在的目录, 因此输出结果如下:

```
$ pwd
/tmp
$ php-cgi -f another_directory/test.php
/tmp/another_directory
```

CLI SAPI 将使用当前的目录, 因此输出结果如下:

```
$ pwd
/tmp
$ php -q another_directory/test.php
/tmp
```

用户在命令行运行 PHP CGI 应用程序时可以给 CGI SAPI 加上 `-C` 参数来使其支持 CLI SAPI 的功能。

CLI SAPI 模块有以下三种不同的方法来获取要运行的 PHP 代码:

1. 使用 PHP 运行指定文件

```
$ php my_script.php
$ php -f my_script.php
```

CLI SAPI 指定的 PHP 脚本并非必须要以 `.php` 为扩展名, 可以有任意的文件名和扩展名来运行任何文件。

2. 使用 PHP 在命令行直接运行 PHP 代码

```
$ php -r 'print_r(get_defined_constants());'
```

3. 使用 PHP 运行通过标准输入 (stdin) 提供的 PHP 代码

```
$ some_application | some_filter | php | sort -u >final_output.txt
```

CLI SAPI 可以动态地生成 PHP 代码并通过命令行运行这些代码。

以上三种运行代码的方法不能同时使用。

1.3 Template Engine

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（比如 HTML）中结合 PHP 使用。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。

原生的 PHP 模板没有了 Blade 等模板引擎的编译阶段，速度会更快。

现今的 PHP 框架都会使用一些模板系统，其中大多数是使用原生的 PHP 语法。在框架之外，一些类库（比如 Plates 或 Aura.View）都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。

- Aura.View(<https://github.com/auraphp/Aura.View>)
- Blade(<https://github.com/laravel/laravel>)
- Brainy(<https://github.com/box/brainy>)
- Dwoo(<https://github.com/dwoo-project/dwoo>)
- Latte(<https://github.com/nette/latte>)
- Mustache(<https://github.com/bobthecow/mustache.php>)
- PHPTAL(<https://github.com/phptal/PHPTAL>)
- Plates(<https://github.com/thephpleague/plates>)
- Smarty(<https://github.com/smarty-php/smarty/>)
- Twig(<https://github.com/twigphp/Twig>)
- Zend\View(<https://github.com/zendframework/zend-view>)

1.3.1 Plates

```
<?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') ?>
```

Example 1 原生 PHP 模板使用 Plates 类库实现继承

```
<?php // template.php ?>

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

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

</body>
</html>
```

```
<?php // user_profile.php ?>

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

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

1.3.2 Smarty

尽管 PHP 不断升级为成熟的、面向对象的语言，但是作为模板语言的特性并没有改善多少。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³到继承以及简化控制结构，编译模板被设计地更容易编写，可读性更高，同时使用上也更加的安全。

编译模板也可以在不同的语言中使用（例如 Mustache），虽然这些编译模板时需要在性能上产生一些轻微的影响，不过如果适当的使用缓存，影响就变得非常小了。

Example 2 使用 Twig 类库编译模板

³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。


```
{% include 'header.html' with {'title': 'User Profile'} %}  
  
<h1>User Profile</h1>  
<p>Hello, {{ name }}</p>  
  
{% include 'footer.html' %}
```

Example 3 使用 Twig 类库实现继承

```
// template.html  
  
<html>  
<head>  
    <title>{% block title %}{% endblock %}</title>  
</head>  
<body>  
  
    <main>  
        {% block content %}{% endblock %}  
    </main>  
  
</body>  
</html>
```

```
// user_profile.html  
  
{% extends "template.html" %}  
  
{% block title %}User Profile{% endblock %}  
{% block content %}  
    <h1>User Profile</h1>  
    <p>Hello, {{ name }}</p>  
{% endblock %}
```

1.4 Dynamic Tracing

DTrace 跟踪调试框架永远可用，并且额外消耗极低。

DTrace 可以跟踪操作系统行为和用户程序的执行情况，可以显示参数值，也可以用来分析性能问题。例如，用户可以使用 DTrace D 脚本语言创建脚本文件来监控探针来高效地分析数据指针。

除非用户使用 DTrace D 脚本激活并监控 PHP 探针，否则它并不会运行，所以不会给正常的应用执行带来任何性能损耗。即使 PHP 探针被激活，它的消耗也是非常低的，甚至可以直接在生产系统中使用。

在运行时可以激活 PHP “用户级静态定义跟踪” (USDT) 探针。举例来说，如果 D 脚本正在监控 PHP `function-entry` 探针，那么，每当 PHP 脚本函数被调用的时候，这个探针将被触发，同时 D 脚本中所关联的动作代码将被执行。在 D 脚本的动作代码中，可以打印出诸如函数所在源文件等信息的探针参数，也可以记录诸如函数执行次数这样的聚合数据。

虽然不同平台提供的 DTrace 功能并不完全相同，不过 DTrace 一般都可以跟踪函数调用以及操作系统行为等。

`--enable-dtrace` 配置参数可以用来开启 PHP 核心的静态探针，PHP 扩展可以拥有额外的静态探针，不过提供了自有探针的 PHP 扩展需要分别构建。

```
$ ./configure --enable-dtrace
$ make
$ make install
```

表 1.1: PHP DTrace 静态探针

探 针 名 称	探针描述	探针参数
request-startup	请求开始时触发。	char *file, char *request_uri, char *request_method
request-shutdown	请求关闭时触发。	char *file, char *request_uri, char *request_method
compile-file-entry	脚本开始编译时触发。	char *compile_file, char *compile_file_translated
compile-file-return	脚本完成编译时触发。	char *compile_file, char *compile_file_translated

探针名称	探针描述	探针参数
execute-entry	操作数数组开始执行时触发，例如函数调用、文件包含以及生成器恢复时会被触发。	char *request_file, int lineno
execute-return	操作数数组执行完毕之后触发。	char *request_file, int lineno
function-entry PHP	引擎进入 PHP 函数或者方法调用时触发。	char *function_name, char *request_file, int lineno, char *classname, char *scope
function-return	PHP 引擎从 PHP 函数或者方法调用返回后触发。	char *function_name, char *request_file, int lineno, char *classname, char *scope
exception-thrown	有异常抛出时触发。	char *classname
exception-caught	有异常被捕获时触发。	char *classname
error	无论 error_reporting 的设置如何，在发生错误时都会触发。	char *errormsg, char *request_file, int lineno

要列出 PHP 中可用的静态探针，开启一个 PHP 进程，然后执行 `dtrace -l`，输出结果如下：

```
$ dtrace -l
ID PROVIDER      MODULE      FUNCTION NAME
[ . . . ]
4  php15271      php         dtrace_compile_file compile-file-entry
5  php15271      php         dtrace_compile_file compile-file-return
6  php15271      php         zend_error error
7  php15271      php  ZEND_CATCH_SPEC_CONST_CV_HANDLER exception-caught
8  php15271      php  zend_throw_exception_internal exception-thrown
9  php15271      php         dtrace_execute_ex execute-entry
10 php15271      php         dtrace_execute_internal execute-entry
11 php15271      php         dtrace_execute_ex execute-return
12 php15271      php         dtrace_execute_internal execute-return
13 php15271      php         dtrace_execute_ex function-entry
14 php15271      php         dtrace_execute_ex function-return
15 php15271      php         php_request_shutdown request-shutdown
16 php15271      php         php_request_startup request-startup
```

如果运行的是 Apache web 服务器，那么模块名称可能是 libphp5.so，并且可能会出现多块信息，每个运行中的 Apache 进程对应一个输出块。

- Provider 列由 PHP 和当前进程 id 的组成。
- Function Name 列表示 Provider 对应的 PHP 内部 C 实现函数名称。

如果没有运行任何 PHP 进程，那么就不会显示任何 PHP 探针。

下面的示例展示了基本的 DTrace D 脚本。

Example 4 PHP 静态探针

```
#!/usr/sbin/dtrace -Zs

#pragma D option quiet

php*:::compile-file-entry
{
    printf("PHP compile-file-entry\n");
    printf(" compile_file      %s\n", copyinstr(arg0));
    printf(" compile_file_translated %s\n", copyinstr(arg1));
}

php*:::compile-file-return
{
    printf("PHP compile-file-return\n");
    printf(" compile_file      %s\n", copyinstr(arg0));
}
```

```
    printf(" compile_file_translated %s\n", copyinstr(arg1));
}

php*:::error
{
    printf("PHP error\n");
    printf(" errormsg           %s\n", copyinstr(arg0));
    printf(" request_file       %s\n", copyinstr(arg1));
    printf(" lineno             %d\n", (int)arg2);
}

php*:::exception-caught
{
    printf("PHP exception-caught\n");
    printf(" classname           %s\n", copyinstr(arg0));
}

php*:::exception-thrown
{
    printf("PHP exception-thrown\n");
    printf(" classname           %s\n", copyinstr(arg0));
}

php*:::execute-entry
{
    printf("PHP execute-entry\n");
    printf(" request_file         %s\n", copyinstr(arg0));
    printf(" lineno              %d\n", (int)arg1);
}

php*:::execute-return
{
    printf("PHP execute-return\n");
    printf(" request_file         %s\n", copyinstr(arg0));
    printf(" lineno              %d\n", (int)arg1);
}

php*:::function-entry
{
    printf("PHP function-entry\n");
    printf(" function_name        %s\n", copyinstr(arg0));
}
```

```

    printf(" request_file      %s\n", copyinstr(arg1));
    printf(" lineno          %d\n", (int)arg2);
    printf(" classname       %s\n", copyinstr(arg3));
    printf(" scope            %s\n", copyinstr(arg4));
}

php*:::function-return
{
    printf("PHP function-return\n");
    printf(" function_name      %s\n", copyinstr(arg0));
    printf(" request_file        %s\n", copyinstr(arg1));
    printf(" lineno            %d\n", (int)arg2);
    printf(" classname         %s\n", copyinstr(arg3));
    printf(" scope             %s\n", copyinstr(arg4));
}

php*:::request-shutdown
{
    printf("PHP request-shutdown\n");
    printf(" file                %s\n", copyinstr(arg0));
    printf(" request_uri         %s\n", copyinstr(arg1));
    printf(" request_method      %s\n", copyinstr(arg2));
}

php*:::request-startup
{
    printf("PHP request-startup\n");
    printf(" file                %s\n", copyinstr(arg0));
    printf(" request_uri         %s\n", copyinstr(arg1));
    printf(" request_method      %s\n", copyinstr(arg2));
}

```

此脚本在 `dtrace` 命令中使用了 `-Z` 选项，此选项保证即使在没有任何 PHP 进程运行的时候脚本也能够正确执行。如果省略了此选项，当没有任何探针可监控的时候，脚本会立即终止执行。

在运行此脚本的过程中，它将监控全部 PHP 核心静态探针。

在下面的示例，运行 D 脚本后再运行一个 PHP 脚本或者 PHP 应用，用来进行监控的 D 脚本就会输出每个探针被触发时所携带的参数。

```
# ./all_probes.d
```

在多 CPU 的主机上，探针的显示顺序可能不是连续的，主要取决于哪颗 CPU 执行探针以及多个 CPU 之间的线程迁移情况，可以通过显示探针时间戳来减少混淆。

```
php*:::function-entry
{
    printf("%lld: PHP function-entry ", walltimestamp);
    [ . . . ]
}
```

在某些 Linux 发行版中，可以使用 SystemTap 工具来监控 PHP 内置的 DTrace 静态探针，或者可以配置 PHP 打开 DTrace 静态探针。例如，在 Oracle Linux 系统上启动 UEK3 内核，并进行如下操作可以打开 DTrace 静态探针。

```
# yum install systemtap-sdt-devel
# ./configure --enable-dtrace ...
# make
# make install
# modprobe fasttrap
# chmod 666 /dev/dtrace/helper
```

除了 `chmod` 命令，也可以使用 ACL 包规则来限制特定用户对于设备的访问权限。
`stap` 命令可以列出 PHP 静态探针，例如：

```
# stap -l 'process.provider("php").mark("*)' -c 'sapi/cli/php -i'
```

输出如下：

```
process("sapi/cli/php").provider("php").mark("compile__file__entry")
process("sapi/cli/php").provider("php").mark("compile__file__return")
process("sapi/cli/php").provider("php").mark("error")
process("sapi/cli/php").provider("php").mark("exception__caught")
process("sapi/cli/php").provider("php").mark("exception__thrown")
process("sapi/cli/php").provider("php").mark("execute__entry")
process("sapi/cli/php").provider("php").mark("execute__return")
process("sapi/cli/php").provider("php").mark("function__entry")
process("sapi/cli/php").provider("php").mark("function__return")
process("sapi/cli/php").provider("php").mark("request__shutdown")
process("sapi/cli/php").provider("php").mark("request__startup")
```

Example 5 使用 SystemTap 追踪 PHP 静态探针

```
probe process("sapi/cli/php").provider("php").mark("compile__file__entry") {
    printf("Probe compile__file__entry\n");
    printf(" compile_file %s\n", user_string($arg1));
}
```

```

    printf(" compile_file_translated %s\n", user_string($arg2));
}
probe process("sapi/cli/php").provider("php").mark("compile__file__return")
{
    printf("Probe compile__file__return\n");
    printf(" compile_file %s\n", user_string($arg1));
    printf(" compile_file_translated %s\n", user_string($arg2));
}
probe process("sapi/cli/php").provider("php").mark("error") {
    printf("Probe error\n");
    printf(" errormsg %s\n", user_string($arg1));
    printf(" request_file %s\n", user_string($arg2));
    printf(" lineno %d\n", $arg3);
}
probe process("sapi/cli/php").provider("php").mark("exception__caught") {
    printf("Probe exception__caught\n");
    printf(" classname %s\n", user_string($arg1));
}
probe process("sapi/cli/php").provider("php").mark("exception__thrown") {
    printf("Probe exception__thrown\n");
    printf(" classname %s\n", user_string($arg1));
}
probe process("sapi/cli/php").provider("php").mark("execute__entry") {
    printf("Probe execute__entry\n");
    printf(" request_file %s\n", user_string($arg1));
    printf(" lineno %d\n", $arg2);
}
probe process("sapi/cli/php").provider("php").mark("execute__return") {
    printf("Probe execute__return\n");
    printf(" request_file %s\n", user_string($arg1));
    printf(" lineno %d\n", $arg2);
}
probe process("sapi/cli/php").provider("php").mark("function__entry") {
    printf("Probe function__entry\n");
    printf(" function_name %s\n", user_string($arg1));
    printf(" request_file %s\n", user_string($arg2));
    printf(" lineno %d\n", $arg3);
    printf(" classname %s\n", user_string($arg4));
    printf(" scope %s\n", user_string($arg5));
}
probe process("sapi/cli/php").provider("php").mark("function__return") {

```



```

printf("Probe function__return: %s\n", user_string($arg1));
printf(" function_name %s\n", user_string($arg1));
printf(" request_file %s\n", user_string($arg2));
printf(" lineno %d\n", $arg3);
printf(" classname %s\n", user_string($arg4));
printf(" scope %s\n", user_string($arg5));
}
probe process("sapi/cli/php").provider("php").mark("request__shutdown") {
    printf("Probe request__shutdown\n");
    printf(" file %s\n", user_string($arg1));
    printf(" request_uri %s\n", user_string($arg2));
    printf(" request_method %s\n", user_string($arg3));
}
probe process("sapi/cli/php").provider("php").mark("request__startup") {
    printf("Probe request__startup\n");
    printf(" file %s\n", user_string($arg1));
    printf(" request_uri %s\n", user_string($arg2));
    printf(" request_method %s\n", user_string($arg3));
}

```

在 PHP 脚本的执行过程中，上述脚本会跟踪所有的 PHP 核心静态探针：

```
# stap -c 'sapi/cli/php test.php' all_probes.stp
```

1.5 Garbage Collection

每个 php 变量存在一个叫"zval" 的变量容器中。

一个 zval 变量容器，除了包含变量的类型和值，还包括两个字节的额外信息。

第一个是"is_ref"，本身是一个用来标识这个变量是否是属于引用集合 (reference set) 的 bool 值。通过这个字节，php 引擎才能把普通变量和引用变量区分开来。

PHP 允许用户通过使用 & 来使用自定义引用，因此 zval 变量容器中还有一个内部引用计数机制来优化内存使用。

第二个额外字节是"refcount"，用以表示指向这个 zval 变量容器的变量 (也称符号即 symbol) 个数，所有的符号存在一个符号表中，其中每个符号都有作用域 (scope)，那些主脚本（例如通过浏览器请求的 PHP 脚本）和每个函数或者方法也都有作用域。

当一个变量被赋常量值时，就会生成一个 zval 变量容器。

Example 6 生成一个新的 zval 容器

```
<?php
$a = "new string";
```

上述 PHP 脚本中定义了新的变量 `a`，是在当前作用域中生成的，同时还生成了类型为 `string` 和值为 `new string` 的变量容器，其额外的两个字节的的信息如下：

- `"is_ref"` 被默认设置为 `FALSE`，因为没有任何自定义的引用生成。
- `"refcount"` 被设定为 `1`，因为这里只有一个变量使用这个变量容器。

当 `"refcount"` 的值是 `1` 时，`"is_ref"` 的值总是 `FALSE`，使用 Xdebug 提供的函数 `xdebug_debug_zval()` 可以显示 `"refcount"` 和 `"is_ref"` 的值。

Example 7 显示 `zval` 信息

```
<?php
xdebug_debug_zval('a');
```

上述示例的结果如下：

```
a: (refcount=1, is_ref=0)='new string'
```

把一个变量赋值给另一变量将增加引用次数 (`refcount`)，变量容器在 `refcount` “变成 `0` 时就被销毁，而且当任何关联到某个变量容器的变量离开它的作用域（例如函数执行结束）或者对变量调用了函数 `unset()` 时，” `refcount` “就会减 `1`。

Example 8 增加一个 `zval` 的引用计数

```
<?php
$a = "new string";
$b = $a;
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=2, is_ref=0)='new string'
```

同一个变量容器被变量 `a` 和变量 `b` 关联时，如果没有必要，PHP 就不会去复制已生成的变量容器。

Example 9 减少一个 `zval` 的引用计数

```
<?php
$a = "new string";
$c = $b = $a;
```

```
xdebug_debug_zval( 'a' );
unset( $b, $c );
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=3, is_ref=0)=new string'
a: (refcount=1, is_ref=0)=new string'
```

如果现在执行 `unset($a);`，那么包含类型和值的这个变量容器就会从内存中删除。

PHP 的复合数据类型（例如 `array` 和 `object`）的值与标量 (`scalar`) 类型的值不同，`array` 和 `object` 类型的变量把它们的成员或属性存在自己的符号表中。例如，下面的例子将生成三个 `zval` 变量容器：

Example 10 生成一个 `array zval`

```
<?php
$a = array( 'meaning' => 'life', 'number' => 42 );
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=1, is_ref=0)=array (
  'meaning' => (refcount=1, is_ref=0)=life',
  'number' => (refcount=1, is_ref=0)=42
)
```

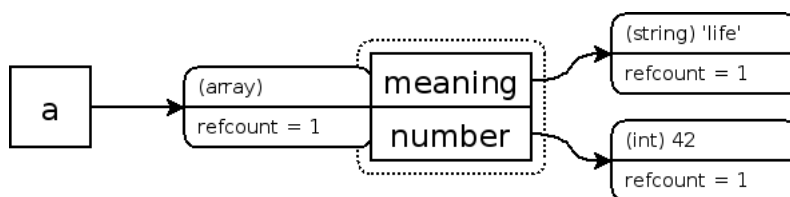


图 1.1: array zval

这里的 `array` 生成的三个 `zval` 变量容器分别是 `a`、`meaning` 和 `number`，增加和减少“`refcount`”的规则和上面提到的相同。

下面的示例在数组中再添加一个元素，并且把它的值设为数组中已存在元素的值：

Example 11 添加一个已经存在的元素到数组中

```
<?php
$a = array( 'meaning' => 'life', 'number' => 42 );
$a['life'] = $a['meaning'];
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=1, is_ref=0)=array (
  'meaning' => (refcount=2, is_ref=0)='life',
  'number' => (refcount=1, is_ref=0)=42,
  'life' => (refcount=2, is_ref=0)='life'
)
```

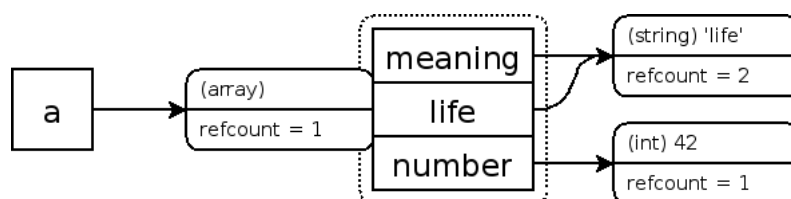


图 1.2: 修改 array zval

从上面的 `xdebug` 输出信息可以看到原有的数组元素和新添加的数组元素关联到同一个“`refcount=2`”的 `zval` 变量容器。

尽管 `Xdebug` 的输出显示两个值为‘`life`’的 `zval` 变量容器, 其实是同一个, `xdebug_debug_zval()` 不显示这个信息, 不过还是能通过显示内存指针信息来看到。

删除数组中的一个元素类似于从作用域中删除一个变量, 删除后的数组中的这个元素所在的容器的“`refcount`”值减少。同样地, 当“`refcount`”为 0 时, 这个变量容器就从内存中被删除。

Example 12 从数组中删除一个元素

```
<?php
$a = array( 'meaning' => 'life', 'number' => 42 );
$a['life'] = $a['meaning'];
unset( $a['meaning'], $a['number'] );
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=1, is_ref=0)=array (
  'life' => (refcount=1, is_ref=0)='life'
)
```

在添加一个数组本身作为这个数组的元素时，需要加入引用操作符，否则 PHP 将生成一个复制。

Example 13 把数组作为一个元素添加到自己

```
<?php
$a = array( 'one' );
$a[] =& $a;
xdebug_debug_zval( 'a' );
```

上述示例的结果如下：

```
a: (refcount=2, is_ref=1)=array (
  0 => (refcount=1, is_ref=0)='one',
  1 => (refcount=2, is_ref=1)=...
)
```

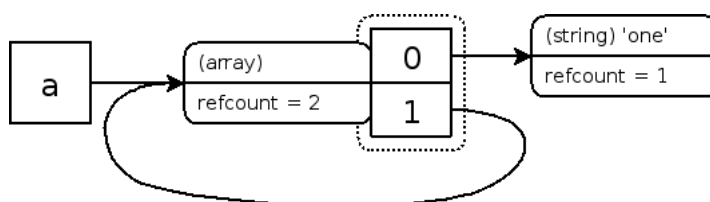


图 1.3: 把数组作为一个元素添加到自己

数组变量 (a) 同时也是这个数组的第二个元素 (1) 指向的变量容器中的“refcount”为 2，上面的输出结果中的“...”说明发生了递归操作，显然在这种情况下意味着“...”指向原始数组。

对一个变量调用 `unset` 将删除这个符号，而且它指向的变量容器中的引用次数也减 1，所以如果在执行完上面的代码后对变量 `$a` 调用 `unset`，那么变量 `$a` 和数组元素“1”所指向的变量容器的引用次数减 1，即从“2”变成“1”。

Example 14 删除数组元素

```
(refcount=1, is_ref=1)=array (
  0 => (refcount=1, is_ref=0)='one',
  1 => (refcount=1, is_ref=1)=...
)
```

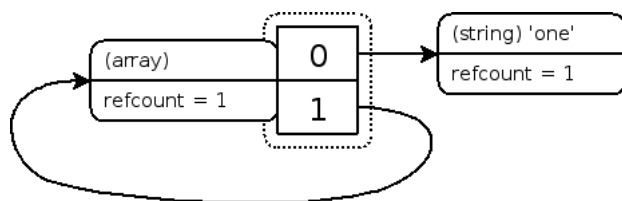


图 1.4: 删除数组元素

另外，尽管不再有某个作用域中的任何符号指向这个结构 (就是变量容器)，由于数组元素“1”仍然指向数组本身，所以这个容器不能被清除。事实上，如果没有其他的符号指向它，用户没有办法清除这个结构，结果就会导致内存泄漏。

PHP 通过在脚本执行结束时清除这个数据结构来避免内存泄露，但是在 PHP 清除之前需要耗费不少内存。如果要实现分析算法，或者要做其他像一个子元素指向它的父元素这样的操作，这种情况就会经常发生。

上述同样的情况也会发生在对象上，实际上对象更有可能出现这种情况，因为对象总是隐式的被引用，结果就是上面这样的情况发生仅仅一两次可能没什么问题，但是如果出现多次的内存泄漏，结果就显然是个大问题。

这样的问题往往发生在长时间运行的脚本中，比如请求基本上不会结束的守护进程 (daemons) 或者单元测试中的大的套件 (sets) 中。其中，后者的例子就是在给巨大的 eZ 组件库的模板组件执行单元测试就可能会出现内存问题，有时测试可能需要耗用 2GB 的内存，而测试服务器很可能没有这么大的内存。

总而言之，引用计数内存机制无法处理循环的引用内存泄漏，需要考虑使用引用计数系统中的同步周期回收中的同步算法来处理这个内存泄漏问题。

1.5.1 Reference Counting

首先，我们先要建立一些基本规则，如果一个引用计数增加，它将继续被使用，当然就不再在垃圾中。如果引用计数减少到零，所在变量容器将被清除 (free)。就是说，仅仅在引用计数减少到非零值时，才会产生垃圾周期 (garbage cycle)。

其次，在一个垃圾周期中，通过检查引用计数是否减 1，并且检查哪些变量容器的引用次数是零，来发现哪部分是垃圾。

在步骤 A 中，为避免不得不检查所有引用计数可能减少的垃圾周期，这个算法把所有可能根 (possible roots 都是 zval 变量容器) 都放在根缓冲区 (root buffer) 中 (用紫色来标记，称为疑似垃圾)，这样可以同时确保每个可能的垃圾根 (possible garbage root) 在缓冲区中只出现一次。

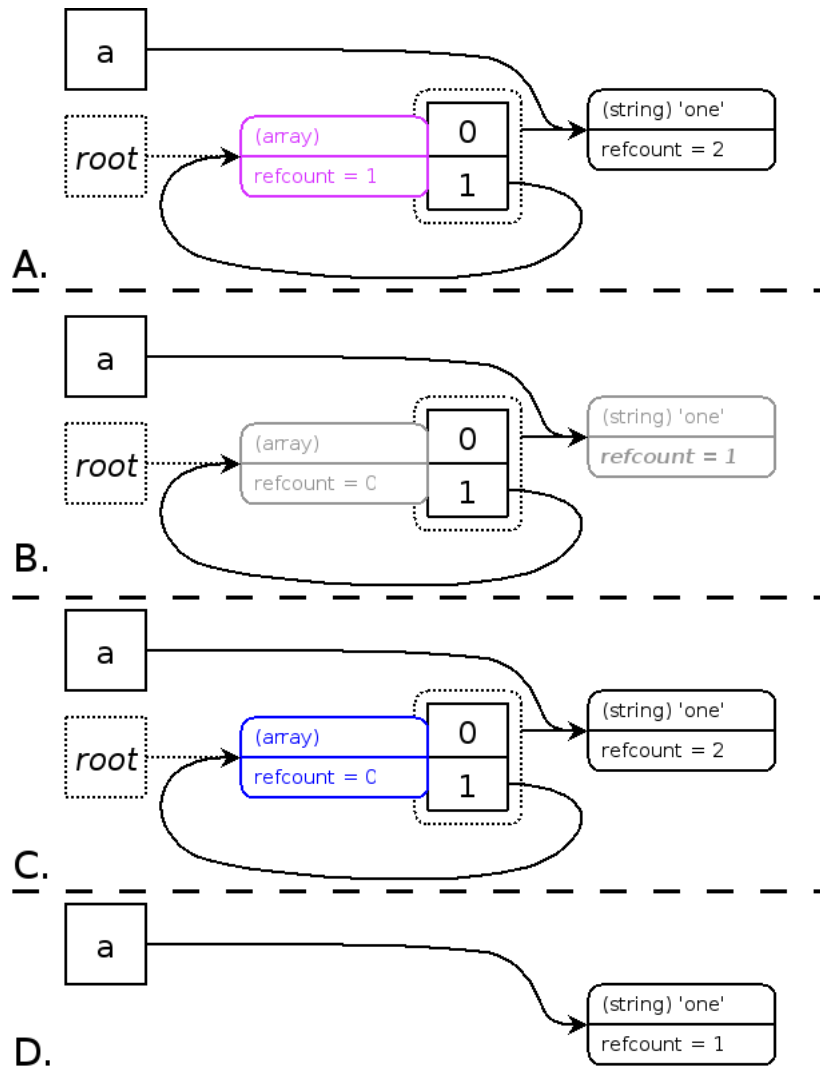


图 1.5: 内存回收算法

仅仅在根缓冲区满了时，才对缓冲区内部所有不同的变量容器执行垃圾回收操作。

在步骤 B 中，模拟删除每个紫色变量。模拟删除时可能将不是紫色的普通变量引用数减 1，如果某个普通变量引用计数变成 0 了，就对这个普通变量再做一次模拟删除。每个变量只能被模拟删除一次，模拟删除后标记为灰。

在步骤 C 中，模拟恢复每个紫色变量。恢复是有条件的，当变量的引用计数大于 0 时才对其做模拟恢复。同样每个变量只能恢复一次，恢复后标记为黑，基本就是步骤 B 的逆运算。这样剩下的一堆没能恢复的就是该删除的蓝色节点了，在步骤 D 中遍历出来真的删除掉。

算法中都是模拟删除、模拟恢复、物理删除，都使用简单的遍历即可（最典型的深度搜索遍历），复杂度为执行模拟操作的节点数正相关，不只是紫色的那些疑似垃圾变量。

默认情况下，PHP 的垃圾回收机制是打开的，而且允许在 `php.ini` 设置和修改 `zend.enable_gc` 配置项。

当垃圾回收机制打开时，每当根缓存区存满时，就会执行上面描述的循环查找算法。

根缓存区有固定的大小，可以存储 10,000 个可能根，也可以通过修改 PHP 源码文件 `Zend/zend_gc.c` 中的常量 `GC_ROOT_BUFFER_MAX_ENTRIES` 并重新编译 PHP 来修改这个 10,000 值。

当垃圾回收机制关闭时，循环查找算法永不执行，实际上可能根就将一直存在根缓冲区中，不管在配置中垃圾回收机制是否激活。

当垃圾回收机制关闭时，如果根缓冲区存满了可能根，更多的可能根显然不会被记录。那些没被记录的可能根，将不会被这个算法来分析处理。如果它们是循环引用周期的一部分，将永不能被清除进而导致内存泄漏。

即使在垃圾回收机制不可用时，可能根也被记录的原因在于相对于每次找到可能根后检查垃圾回收机制是否打开而言，记录可能根的操作更快。不过垃圾回收和分析机制本身要耗不少时间。

除了修改配置 `zend.enable_gc`，也能通过分别调用 `gc_enable()` 和 `gc_disable()` 函数来打开和关闭垃圾回收机制。

调用 `gc_enable()` 和 `gc_disable()` 函数与修改配置项来打开或关闭垃圾回收机制的效果是一样的。即使在可能根缓冲区还没满时，也能强制执行周期回收，通过调用 `gc_collect_cycles()` 函数可以达到这个目的，这个函数将返回使用这个算法回收的周期数。

允许打开和关闭垃圾回收机制并且允许自主的初始化的原因，是由于应用程序的某部分可能是高时效性的。在这种情况下，用户就可能不想使用垃圾回收机制。

对应用程序的某部分关闭垃圾回收机制是在冒着可能内存泄漏的风险，因为一些可能根也许存不进有限的根缓冲区，因此就在调用 `gc_disable()` 函数释放内存之前，先调用

`gc_collect_cycles()` 函数可能比较明智。因为这将清除已存放在根缓冲区中的所有可能根，然后在垃圾回收机制被关闭时，可留下空缓冲区以有更多空间来存储可能根。

1.5.2 Collecting Cycles

虽然回收可能根有细微的性能上影响，例如记录可能根相对于完全不记录可能根要慢些，不过后来对 **PHP run-time** 的其他修改减少了这个性能损失。

在进行垃圾回收时产生的性能影响中，第一个是内存占用空间的节省，另一个是垃圾回收机制执行内存清理时的执行时间增加 (**run-time delay**)。

首先，实现垃圾回收机制的整个原因是为了，一旦先决条件满足，通过清理循环引用的变量来节省内存占用。

在 PHP 执行中，一旦根缓冲区满了或者调用 `gc_collect_cycles()` 函数时，就会执行垃圾回收。例如，下面的示例显示了在 PHP 5.2 和 PHP 5.3 环境下的内存占用情况，其中排除了脚本启动时 PHP 本身占用的基本内存。

Example 15 GC 内存使用情况

```
<?php
class Foo
{
    public $var = '3.1415962654';
}

$baseMemory = memory_get_usage();

for ( $i = 0; $i <= 100000; $i++ )
{
    $a = new Foo;
    $a->self = $a;
    if ( $i % 500 === 0 )
    {
        echo sprintf( '%8d: ', $i ), memory_get_usage() - $baseMemory, "\n";
    }
}
```

在这个示例中，首先创建了一个对象，这个对象中的一个属性被设置为指回对象本身。在循环的下一个重复 (**iteration**) 中，当脚本中的变量被重新复制时，就会发生典型性的内存泄漏。

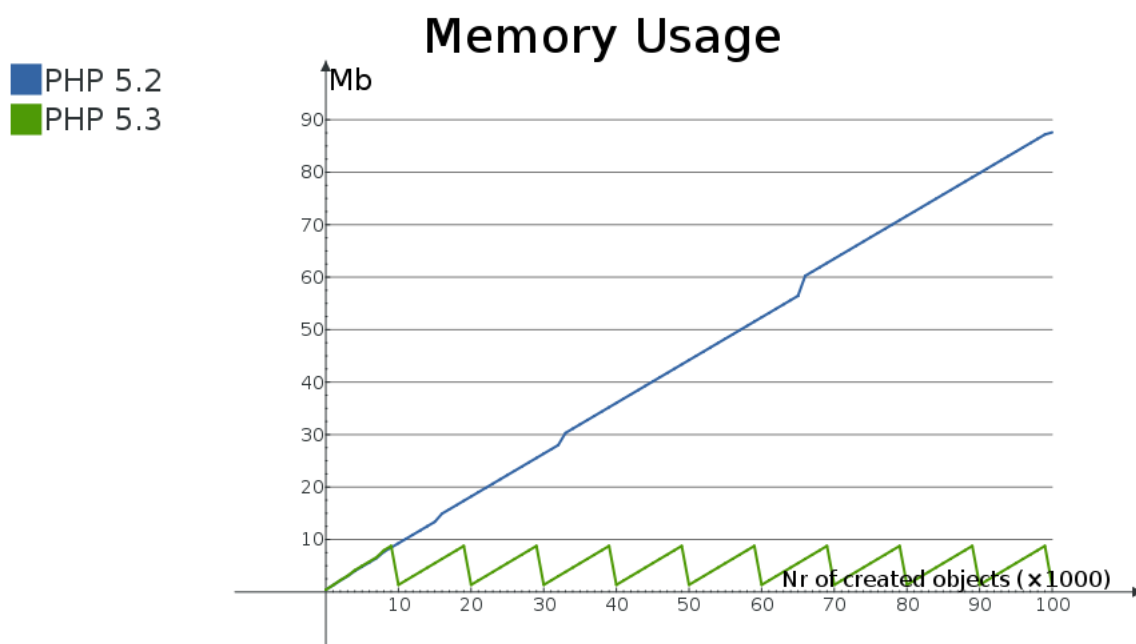


图 1.6: GC 内存使用情况

在这个例子中，两个变量容器是泄漏的 (对象容器和属性容器)，但是仅仅能找到一个可能根——就是被 `unset` 的那个变量。在 10,000 次重复后 (也就产生总共 10,000 个可能根)，当根缓冲区满时，就执行垃圾回收机制，并且释放那些关联的可能根的内存。

从 PHP 5.3 的锯齿型内存占用图中很容易就能看到，每次执行完 10,000 次重复后，执行垃圾回收，并释放相关的重复使用的引用变量。这里由于泄漏的数据结构非常简单，所以垃圾回收机制本身不必做太多工作。从这个图表中，可以看到 PHP 5.3 的最大内存占用大概是 9 Mb，而 PHP 5.2 的内存占用一直增加。

垃圾回收影响性能的第二个领域是它释放已泄漏的内存耗费的时间，可以修改上述的示例来查看这个耗时的多少，有更多次数的重复并且删除了循环中的内存占用计算。

Example 16 GC 性能影响

```
<?php
class Foo
{
    public $var = '3.1415962654';
}

for ( $i = 0; $i <= 1000000; $i++ )
{
```

```

    $a = new Foo;
    $a->self = $a;
}
echo memory_get_peak_usage(), "\n";

```

下面运行这个脚本两次，一次通过配置 `zend.enable_gc` 打开垃圾回收机制时，另一次是它关闭时。

```

$ time php -dzend.enable_gc=0 -dmemory_limit=-1 -n example2.php
10.7s
$ time php -dzend.enable_gc=1 -dmemory_limit=-1 -n example2.php
11.4s

```

后面一次执行比第一次执行的结果在时间上增加了 7%，不过执行这个脚本时内存占用的峰值降低了 98%（从 931Mb 降到 10Mb），从而可以显示出垃圾回收机制在内存占用方面的好处就是在执行中出现更多的循环引用变量时，内存节省的更多的情况下，每次时间增加的百分比都是 7%。

为了在 PHP 内部可以显示更多的关于垃圾回收机制如何运行的信息，需要先重新编译 PHP 使 `benchmark` 和 `data-collecting code` 可用，在执行 `./configure` 之前先把环境变量 `CFLAGS` 设置成 `-DGC_BENCH=1`。

Example 17 启用 GC benchmarking

```

$ export CFLAGS=-DGC_BENCH=1
$ ./config.nice
$ make clean
$ make

```

在打开 PHP 的 `benchmark` 和 `data-collecting code` 来重新执行上面的例子代码时，PHP 执行结束后可以看到下面的 GC 统计信息：

```

GC Statistics
-----
Runs:          110
Collected:    2072204
Root buffer length: 0
Root buffer peak: 10000

Possible      Remove from Marked
Root  Buffered  buffer  grey
-----

```

```
ZVAL 7175487 1491291 1241690 3611871
ZOBJ 28506264 1527980 677581 1025731
```

其中，主要的信息统计在第一个块，可以看到垃圾回收机制运行了 110 次，而且在这 110 次运行中，总共有超过两百万的内存分配被释放。只要垃圾回收机制运行了至少一次，根缓冲区峰值 (Root buffer peak) 总是 10000。

通常情况下，PHP 中的垃圾回收机制仅仅在循环回收算法确实运行时会有时间消耗上的增加，但是在平常的 (更小的) 脚本中没有性能影响。

在普通的 PHP 脚本中有循环回收机制运行的情况下，内存的节省将允许更多的 PHP 脚本同时运行在服务器上，而且总共使用的内存并不会达到上限。

这种好处在长时间运行脚本（例如长时间的测试套件或者 `daemon` 脚本等）中尤其明显，同时对通常比 Web 脚本运行时间长的 PHP-GTK 应用程序，新的垃圾回收机制也可以大大改变一直以来认为内存泄漏问题难以解决的看法。

1.6 Development Server

PHP CLI SAPI 提供了一个内置的 Web 服务器，从命令行终端进入项目的 Web 根目录并执行下面的命令就可以启动内置的 Web 服务器来进行本地开发调试。

```
$ cd ~/public_html
$ php -S localhost:8000
PHP Development Server started at Tue Dec 6 10:03:32 2016
Listening on http://localhost:8000
Document root is /home/me/public_html
Press Ctrl-C to quit.
```

URI 请求会被发送到 PHP 所在的工作目录 (Working Directory) 进行处理，除非使用了 `-t` 参数来自定义不同的目录。

```
$ cd ~/public_html
$ php -S localhost:8000 -t foo/
PHP Development Server started at Tue Dec 6 10:03:32 2016
Listening on http://localhost:8000
Document root is /home/me/public_html/foo
Press Ctrl-C to quit.
```

如果请求未指定执行哪个 PHP 文件，则默认执行目录内的 `index.php` 或者 `index.html`。如果这两个文件都不存在，服务器会返回 404 错误。例如，访问 `http://localhost:8000/` 和 `http://localhost:8000/myscript.html` 的输出窗口如下显示：

```
$ cd ~/public_html
$ php -S localhost:8000
PHP Development Server started at Tue Dec 6 10:03:32 2016
Listening on http://localhost:8000
Document root is /home/me/public_html
Press Ctrl-C to quit.
::1:39144 GET /favicon.ico - Request read
::1:39146 GET / - Request read
::1:39147 GET /favicon.ico - Request read
::1:39148 GET /myscript.html - Request read
::1:39149 GET /favicon.ico - Request read
```

修改 `php.ini` 中的 `cli_server.color` 项可以控制内置 Web Server 的终端输出有无颜色。

表 1.2: CLI SAPI `cli_server.color` 配置选项

名字	默认	可修改范围
<code>cli_server.color</code>	"0"	PHP_INI_ALL

1.6.1 Router

如果在启动 PHP 内置的 Web 服务器时指定了一个 PHP 文件，则这个文件会作为一个“路由”脚本，意味着每次请求都会先执行这个脚本。

```
<?php
// router.php
if (preg_match('/\.(?:png|jpg|jpeg|gif)$/i', $_SERVER["REQUEST_URI"]))
    return false; // 直接返回请求的文件
else {
    echo "<p>Welcome to PHP</p>";
}
?>
```

如果指定了路由脚本，那么可以设置请求图片直接显示图片，请求 HTML 则显示“Welcome to PHP”。例如，执行之后终端显示：

```
$ cd ~/public_html
$ php -S localhost:8000 router.php
PHP Development Server started at Tue Dec 6 10:03:32 2016
Listening on http://localhost:8000
Document root is /home/me/public_html
Press Ctrl-C to quit.
```

```
::1:55801 GET /mylogo.jpg - Request read
::1:55803 GET /abc.html - Request read
::1:55804 GET /favicon.ico - Request read
```

如果路由脚本返回 `FALSE`，那么直接返回请求的文件（例如请求静态文件不作任何处理），否则会把输出返回到浏览器。

1.6.2 Template

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，但是也可以使用其他的格式（例如 `XML`）。

模板通常也被称为「视图」，模板其实只是 `MVC`（模型-视图-控制器）软件架构模式第二个元素的一部份。

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，模板的单一职责只是呈现格式化后的内容。

模板不负责数据的查询、保存或是其他复杂的任务，从而进一步促成了更干净、更具可读性的代码，开发者在团队协作开发中可以专注服务端的代码（控制器、模型），而设计师则专注客户端代码（网页）。

模板还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。

模板的组织方式鼓励代码重用，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，根据用户选择的类库，模板可以通过自动转义用户的内容来提供更多的安全性。有些类库可以提供沙箱机制，模板设计者只能使用在白名单中的变量和函数。

1.7 Closure/Anonymous

`PHP` 从 5.3（2009 年）之后开始引入对闭包以及匿名函数的支持，并在接下来的 `PHP` 5.4 中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，从此在大部分情况下可以使用匿名函数取代一般的函数。

1.8 Meta Programming

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（例如 `__get()`、`__set()`、`__clone()`、`__toString()`、`__invoke()` 等）可以改变类的行为。

PHP 通过 `__call()` 和 `__callStatic()` 可以实现和 Ruby 的 `method_missing` 方法完成相同的功能。

1.9 Functional Program

函数式编程是一种不同于对象式编程的思想，PHP 并不是天生就属于函数式编程的语言。

PHP 本身的语法、语义不能完全很好地支持函数式编程，不过可以通过封装类库来完成 PHP 对函数式编程的过滤、映射和归约的支持。

```
<?php
/**
 * 函数式编程
 *
 * @author dogstar 20161113
 */

class Functional_Lite {

    protected $data;

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

    /** ----- middle operation ----- */

    public function filter($callback) {
        $newData = array();

        if (!is_callable($callback)) {
            $callback = create_function('$it', 'return ' . $callback . ';');
        }
    }
}
```

```

foreach ($this->data as $item) {
    if (call_user_func($callback, $item)) {
        $newData[] = $item;
    }
}

$this->data = $newData;

return $this;
}

public function map($callback) {
    if (!is_callable($callback)) {
        $callback = create_function('$it', 'return ' . $callback . ';');
    }

    foreach ($this->data as &$itemRef) {
        $itemRef = call_user_func($callback, $itemRef);
    }

    return $this;
}

/** ----- end operation ----- */

public function reduce($callback, $accumulator) {
    $rs = $accumulator;

    if (!is_callable($callback)) {
        $callback = create_function('$it_1, $it_2', 'return ' . $callback . ';');
    }

    foreach ($this->data as $item) {
        $rs = call_user_func_array($callback, array($rs, $item));
    }

    return $rs;
}

public function foldLeft($callback, $accumulator = NULL) {

```



```

    // default
    if ($accumulator === NULL) {
        $accumulator = array_shift($this->data);
    }

    return $this->reduce($callback, $accumulator);
}

public function foldRight($callback, $accumulator = NULL) {
    $this->data = array_reverse($this->data);
    return $this->foldLeft($callback, $accumulator);
}

/** ----- extra operation ----- */
}

```

假设我们需要计算 1 到 5 之间的偶数之和的两倍，那么：

```

$fun = new Functional_Lite(range(1, 5));

$rs = $fun->filter(function ($x) { return $x % 2 == 0;})
    ->map(function ($x) {return $x * 2;})
    ->reduce(function ($x, $y) { return $x + $y;}, 0);

var_dump($rs); // 2 * 2 + 4 * 2 = 12

```

使用字符串表达式进一步缩短代码，可以这样：

```

$fun = new Functional_Lite(range(1, 5));

$rs = $fun->filter('$it % 2 == 0')
    ->map('$it * 2')
    ->reduce('$it_1 + $it_2', 0);

var_dump($rs); // 2 * 2 + 4 * 2 = 12

```

这里使用了 `$it` 来表示每次迭代的元素。例如，对于 `reduce` 操作，使用了 `$it_1`、`$it_2` 分别表示第一个元素、第二个元素。

除了对数值进行操作，也可以对字符串或其他类型的集合进行操作。假设需要过滤一个字符的单词，然后把剩下的单词转大写，再用空格拼接起来。

```
$fun = new Functional_Lite(array('dogstar', 'x', 'love', 'y', 'yoyo', '!!!'));

$rs = $fun->filter(function ($x) { return strlen($x) > 1;})
    ->map(function ($x) { return strtoupper($x);})
    ->reduce(function ($x, $y) { return $x . ' ' . $y;}, 'RS:');

var_dump($rs); // RS: DOGSTAR LOVE YOYO !!
```

对于最后的结果，我们在累加量初始了“RS:”，所以最后的结果里有“RS:”的前缀。同样地，使用字符串表达式可以简化上面的处理以达到同样的效果：

```
$rs = $fun->filter('strlen($it) > 1')
    ->map('strtoupper($it)')
    ->reduce('$it_1 . " " . $it_2', 'RS:');
```

PHP 实际上距离函数式编程的道路还很遥远，比如柯里化、部分施用、偏函数、Either 类、Option 类、值不可变，而且有很多在 PHP 中是明显很难实现的（例如上下文的处理）。

1.10 Dependency Injection

依赖注入是一种允许用户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。

具体来说，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。

下面我们可以用一个简单的例子来说明依赖注入的概念，代码中有一个 Database 的类需要一个适配器来与数据库交互，在构造函数里实例化了适配器就会产生耦合，使得测试变得很困难，而且 Database 类和适配器耦合的很紧密。

```
<?php
namespace Database;

class Database
{
    protected $adapter;

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

```
class MysqlAdapter {  
}
```

上述这段代码可以用依赖注入进行重构来实现解耦。

```
<?php  
namespace Database;  
  
class Database  
{  
    protected $adapter;  
  
    public function __construct(MySqlAdapter $adapter)  
    {  
        $this->adapter = $adapter;  
    }  
}  
  
class MySqlAdapter {  
}
```

依赖注入使得代码通过外界给予 `Database` 类的依赖，而不是让它自己产生依赖的对象，还可以使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` 的就可以直接给它赋值。

“控制反转”(Inversion of Control) 或者“依赖反转准则”(Dependency Inversion Principle) 是依赖注入可以解决的更复杂的问题。

1.10.1 Inversion of Control

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来实现解耦。

虽然 PHP 框架可以实现控制反转，不过现实问题是应该反转哪些部分以及到反转到什么程度。比如，MVC 框架通常会提供超类 (Superclass) 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。

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

1.10.2 Dependency Inversion

依赖反转准则（Dependency Inversion Principle）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，依赖反转准则就是指依赖应该是接口/约定或者抽象类，而不是具体的实现。

接下来开始重构前面的例子使其遵循依赖反转准则：

```
<?php
namespace Database;

class Database
{
    protected $adapter;

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

interface AdapterInterface {}

class MysqlAdapter implements AdapterInterface {}
```

遵循依赖反转准则的实现（例如 Database 类）依赖于接口，相比依赖于具体实现有更多的优势。例如，假设工作团队中的一位同事负责设计适配器，那么在第一个例子中，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，于是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

依赖反转的一个更大的好处是代码扩展性变得更高。例如，如果一段时间之后我们决定要迁移到一种不同的数据库，接下来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，因此不再需要额外的重构。

1.11 Framework Container

依赖注入容器和依赖注入不是相同的概念。容器是帮助用户更方便地实现依赖注入的工具，但是它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，允许通过配置将依赖绑定在一起，这样就可以写出和框架层同样干净、解耦的应用层代码。

1.12 Commandline Interface

PHP 提供的 CLI SAPI (Server Application Programming Interface, 服务端应用编程端口) 允许使用 PHP 开发命令行应用。

- CLI SAPI 和其它 CLI SAPI 模块相比有很多的不同之处;
- CLI 和 CGI 是不同的 SAPI, 尽管它们之间有很多共同的行为。

PHP 的命令行 (CLI) 脚本接口可以帮助用户完成自动化的任务 (例如测试、部署和应用管理), 而且可以直接调用程序代码而无需创建 Web 图形界面。

用户执行 `php -a` 选项就可以进入和 Ruby 的 IRB 或 python 的交互式 shell 相似的交互式 shell。例如, 一个简单的 “Hello, \$name” CLI 程序:

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

PHP 会在脚本运行时根据参数设置两个特殊的变量, `$argc` 是一个表示参数个数的整数, `$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 (例如这里就是 `hello.php`)。

```
$ php -a
Interactive mode enabled

> php hello.php
Usage: php hello.php [name]
> php hello.php world
Hello, world
```

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。

以下为 CLI SAPI 和其它 CLI SAPI 模块相比的显著区别:

1. 与 CGI SAPI 不同, 其输出没有任何头信息。

尽管 CGI SAPI 提供了取消 HTTP 头信息的方法, 但在 CLI SAPI 中并不存在类似的方法以开启 HTTP 头信息的输出。

CLI 默认以安静模式开始，但为了保证兼容性，`-q` 和 `--no-header` 参数为了向后兼容仍然保留，使得可以使用旧的 CGI 脚本。

在运行时，PHP CLI 不会把工作目录改为脚本的当前目录(可以使用 `-C` 和 `--no-chdir` 参数来兼容 CGI 模式)。

出错时输出纯文本的错误信息（非 HTML 格式）。

2. CLI SAPI 强制覆盖了 `php.ini` 中的某些设置，这些设置在命令行环境下是没有意义的。

CLI SAPI 覆盖的 `php.ini` 设置选项包括 `html_errors`, `implicit_flush`, `max_execution_time` 和 `register_argc_argv`。

3. CLI SAPI 不会将当前目录改为已运行的脚本所在的目录。

可以在命令行运行时给该 CGI SAPI 加上 `-C` 参数，使其支持 CLI SAPI 的功能。

表 1.3: CLI SAPI 强制覆盖的设置选项

设置选项	CLI SAPI 默认值	备注
<code>html_errors</code>	<code>FALSE</code>	无意义的 HTML 标记符使出错信息很凌乱，所以在命令行下阅读报错信息十分困难，因此将该选项的默认值改为 <code>FALSE</code> 。
<code>implicit_flush</code>	<code>TRUE</code>	命令行模式所有来自 <code>print</code> 和 <code>echo</code> 的输出将被立即写到输出端，而不作任何地缓冲操作。如果希望延缓或控制标准输出，仍然可以使用 <code>output buffering</code> 设置项。
<code>max_execution_time</code>	<code>0</code> (无限值)	命令行环境下使用 PHP 时取消了最大运行时间的限制，命令行应用程序可以在后台持久运行。
<code>register_argc_argv</code>	<code>TRUE</code>	设置为 <code>TRUE</code> 就总是可以在 CLI SAPI 中访问到 <code>argc</code> (传送给应用程序参数的个数) 和 <code>argv</code> (包含有实际参数的数组)。

上述这些设置无法在 `php.ini` 或任何指定的其它文件中被初始化为其它值，这些默认值被限制在所有其它的设置文件被解析后改变，不过这些设置的值可以在程序运行的过程中被改变（尽管对于该运行过程来说，这些设置项是没有意义的）。

CLI/CGI 二进制执行文件的文件名、位置和是否存在会根据 PHP 在系统上的安装而不同，在命令行下运行 `php -v` 就可以输出该 php 是 CGI 还是 CLI 的相关信息。

在默认情况下，当运行 `make` 时，CGI 和 CLI 都会被编译并且分别放置在 PHP 源文件目录的 `sapi/cgi/php` 和 `sapi/cli/php` 下，而且这两个文件都被命名为 `php`。

在 `make install` 的过程中会发生的行为取决于配置行。

- 如果在配置的时候选择了一个 SAPI 模块（例如 `apxs`），或者使用了 `--disable-cgi` 参数，则在 `make install` 的过程中，CLI 将被拷贝到 `{PREFIX}/bin/php`，除非 CGI 已经被放置在了那个位置。
- 如果希望撤销 CGI 执行文件的安装，需要在 `make install` 之后运行 `make install-cli`，或者在配置行中加上 `--disable-cgi` 参数。

`--enable-cli` 和 `--enable-cgi` 同时默认有效，因此不需要再配置行中加上 `--enable-cli` 来使得 CLI 在 `make install` 过程中被拷贝到 `{PREFIX}/bin/php`。

CLI SAPI 提供了命令行环境下专用的常量，例如 `STDIN`、`STDOUT` 和 `STDERR`，这样用户就无需自己建立指向诸如 `stderr` 的流，只需简单的使用这些常量来代替流指向即可，同时无需用户自己来关闭这些流，PHP 会自动完成这些操作。

```
$ php -r 'fwrite(STDERR, "stderr\n");'
```

- `STDIN` 是一个已打开的指向 `stdin` 的流。可以用如下方法来调用：

```
<?php
$stdin = fopen('php://stdin', 'r');
```

- `STDOUT` 是一个已打开的指向 `stdout` 的流。可以用如下方式来调用：

```
<?php
$stdout = fopen('php://stdout', 'w');
```

- `STDERR` 是一个已打开的指向 `stderr` 的流。可以用如下方式来调用：

```
<?php
$stderr = fopen('php://stderr', 'w');
```

如果需要从 `stdin` 读取一行内容，可以使用如下方式：

```
<?php
$line = trim(fgets(STDIN)); // 从 STDIN 读取一行
fscanf(STDIN, "%d\n", $number); // 从 STDIN 读取数字
```

1.12.1 \$argv

除了 `STDIN`、`STDOUT` 和 `STDERR` 之外，PHP CLI 没有限制传送给脚本程序的参数的个数（`shell` 程序对命令行的字符数有限制，但通常都不会超过该限制）。

命令行中传递给脚本的参数可在全局变量 `$argv` 中获取。该数组中下标为零的成员为脚本的名称（当 PHP 代码来自标准输入获直接用 `-r` 参数以命令行方式运行时，该名称为 “-”）。

1.12.2 \$argv

全局变量 `$argc` 存有 `$argv` 数组中成员变量的个数（而非传送给脚本程序的参数的个数）。

向脚本传送以 - 开头的参数会导致错误，因为 PHP 会认为应该由它自身来处理这些参数。可以用参数列表分隔符 `--` 来解决这个问题。在 PHP 解析完参数后，该符号后所有的参数将会被原样传送给脚本程序。

- 以下命令将不会运行 PHP 代码，而只显示 PHP 命令行模式的使用说明：

```
$ php -r 'var_dump($argv);' -h
Usage: php [options] [-f] <file> [args...]
[...]
```

- 以下命令将会把 “-h” 参数传送给脚本程序，PHP 不会显示命令行模式的使用说明：

```
$ php -r 'var_dump($argv);' -- -h
array(2) {
  [0]=>
    string(1) "-"
  [1]=>
    string(2) "-h"
}
```

1.12.3 shibang

PHP 的命令行模式能使得 PHP 脚本能完全独立于 web 服务器单独运行。

- 如果使用 Unix 系统，需要在 PHP 脚本的最前面加上一行特殊的代码，使得它能够被执行，这样系统就能知道用哪个程序去运行该脚本。
- 如果使用 Windows 系统，可以将 `php.exe` 和 `.php` 文件的双击属性相关联，也可以编写一个批处理文件来用 PHP 执行脚本。

为 Unix 系统增加的第一行代码不会影响该脚本在 Windows 下的运行，因此也可以用该方法编写跨平台的脚本程序。具体来说，在 PHP 脚本的第一行以 `#!/usr/bin/php` 开头，在其后加上以 PHP 开始和结尾标记符包含的正常的 PHP 代码，然后为该文件设置正确的运行属性（例如 `-r-xr-xr-x`），就可以使得该文件能够像 shell 脚本或 PERL 脚本一样被直接执行。


```
#!/usr/bin/php
<?php
var_dump($argv);
```

在向上述该脚本传送以 - 开头的参数时，脚本仍然能够正常运行。

```
$ chmod +x test
$ ./test -h -- foo
array(4) {
    [0]=>
        string(6) "./test"
    [1]=>
        string(2) "-h"
    [2]=>
        string(2) "--"
    [3]=>
        string(3) "foo"
}
```

下面是一个以命令行方式运行的 PHP 脚本的示例。

```
#!/usr/bin/php
<?php
if ($argc != 2 || in_array($argv[1], array('--help', '-help', '-h', '-?'))) {
    ?>

    This is a command line PHP script with one option.

    Usage:
    <?php echo $argv[0]; ?> <option>

    <option> can be some word you would like
    to print out. With the --help, -help, -h,
    or -? options, you can get this help.

    <?php
} else {
    echo $argv[1];
}
?>
```

这个示例程序检查了参数的个数是大于 1 个还是小于 1 个，而且如果参数是 `--help`，`-help`，`-h` 或 `-?` 时，打印出帮助信息，并同时动态输出脚本的名称，最后如果还收到了其它参数则显示其内容。

在第一行中用特殊的代码来指明该文件应该由 PHP 来执行。在这里使用 CLI 的版本，因此不会有 HTTP 头信息输出。

在用 PHP 编写命令行应用程序时，可以使用两个参数 `$argc` 和 `$argv`。其中，前面一个的值是比参数个数大 1 的整数（运行的脚本本身的名称也被当作一个参数），第二个是包含有参数的数组，其第一个元素为脚本的名称，下标为数字 0 (`$argv[0]`)。

如果希望在 Unix 下运行以上脚本，需要使其属性为可执行文件，然后简单的运行 `script.php echothis` 或 `script.php -h`。在 Windows 下，可以为此编写一个批处理文件：

```
@C:\php\php.exe script.php %1 %2 %3 %
```

上述这个批处理文件会帮助将附加的参数传给脚本程序：`script.bat echothis` 或 `script.bat -h`。

Chapter 2

Syntax

2.1 Constructs

PHP 的语法和语义说明了 PHP 代码中的文本和符号的意义，它们定义了 PHP 程序如何编写和解析，以及 PHP 代码如何嵌入到 HTML 文档中。

= 用于变量赋值操作，每条 PHP 语句以逗号结束，可以使用 `echo` 语句来输出 PHP 变量（以 `$` 开头）的内容，同时 PHP 命名是大小写敏感的。

用户可以使用 `function` 和 `class` 来进行代码模块化，而且 PHP 支持 `if`、`while`、`for`、`foreach`、`switch` 等控制结构，同时 PHP 也支持类似 `end*` 的控制代码结构。

```
if ($x==0) :  
    echo "zero";  
endif;
```

PHP 原生的模板引擎功能支持把 PHP 代码直接嵌入到 HTML 中，例如：

```
<!DOCTYPE html>  
<html>  
<meta charset="utf-8">  
<head>  
    <title>PHP Test</title>  
</head>  
<body>  
    <?php  
        echo 'Hello World';  
    ?>  
</body>  
</html>
```

2.2 Delimiters

PHP 引擎根据 `<?php` 和 `?>` 分隔符来解析 PHP 代码。

PHP 会在输出时自动删除其结束符 `?>` 后的一个换行，主要是针对在一个页面中嵌入多段 PHP 代码或者包含了无实质性输出的 PHP 文件而设计。如果确实需要在 PHP 结束符 `?>` 之后输出换行，可以在其后加一个空格，或者在最后的一个 `echo/print` 语句中加入一个换行。

2.3 Variables

PHP 中，变量以 “\$” 后接变量名称来表示，而且变量名称区分大小写。

- 变量是程序运行时用于保存信息或数据的临时“仓库”，可以改变或删除变量。
- 变量和常量可以理解为程序中的基本数据存储单元，可以用来存储不同类型的数据。
- PHP 的变量和常量的数据类型由程序的上下文决定。

有效的变量名称应以字母或下划线开头，后可以接任意数目的字母、数字或下划线，PHP 也支持使用多字节文字作为变量名。

PHP 解释器在解析变量时，将会尽可能多地取得 “\$” 后面的字符来组成一个合法的变量名，可以使用花括号把变量名括起来以明确表示一个变量。

2.4 Structures

PHP 使用 `{...}` 来放置控制结构中的代码，或者也可以使用类似如下的结构来进行逻辑控制。

```
if (condition) :  
    // code here  
elseif (condition) :  
    // code here  
else :  
    // code here  
endif;
```

PHP 的语法参考了 Perl、C 语言，而且可以内嵌在 HTML 之中。

```
<!DOCTYPE html>  
<meta charset="utf-8">  
<html>
```

```
<head>
  <title>PHP Test</title>
</head>
<body>
  <?php
    echo 'Hello World';
  ?>
</body>
</html>
```

PHP 解析引擎只解析 `<?php` 到 `?>` 之间的代码，不包含在 `<?php` 到 `?>` 之间的内容会直接输出。或者说，`<?php` 和 `?>` 可以让源代码不断地进出 PHP 模式，并最终生成解析结果。

```
<?php
  //-PHP代码
?>
html内容
<?php
  //-PHP代码
?>
```

在 HTML 中嵌入 PHP 时，比如需要单独输出某个变量，除了正常采用 `echo` 语句外，可以直接采用 `<?=$title ?>`

在判断语句中的 HTML 代码并不会被直接输出，而且根据 PHP 代码的执行来决定，例如：

```
<?php
  if (false) {
  ?>
HTML Code
<?php
  }
?>
```

PHP 可以用三种注解的形式：C 与 C++ 所使用的 `/*...*/` 与 `//`，和 Perl 的 `#`。

```
<?php
/* 多行注释的第一行
多行注释的另一行 */

// 单行注释
```

```
# 单行注释
```

```
?>
```

下面的示例说明了 PHP 的变量、常量、控制流程、函数、程序标记和注释等基本元素。

```
<html>
<head><title>Test</title></head>
<body>
<?php
define("MAX_LINE_NUM",4);
$title="<h1>Hello,php</h1>\n";
echo $title;

echo "<pre>\n";
for($i=1;$i<MAX_LINE_NUM;$i++){
    echo print_star($i);
    echo "\n";
}
echo "</pre>\n";

function print_star($num){
    return str_repeat("*",$num);
}

?>
</body>
</html>
```

在对函数或类的定义进行注释说明时，应该包含功能说明、参数列表和返回值等信息，也可以加入应用范例。

Chapter 3

Encoding

PHP 没有在底层实现对 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，但是这并不是很简单，通常需要对 Web 应用进行全方面的检查（从 HTML 到 SQL 再到 PHP）。

3.1 String

最基本的字符串操作（例如连结两个字符串或将字符串赋值给变量）往往并不需要对 UTF-8 做特别的处理，然而大多数字符串的函数（例如 `strpos()` 和 `strlen()`）确实需要特别的处理。

字符串函数名中通常包含 `mb_*`（例如 `mb_strpos()` 和 `mb_strlen()`），这些 `mb_*` 字符串是由 `mbstring`（Multibyte String Extension）提供支持并专门为操作 Unicode 字符串而特别设计。

在操作 Unicode 字符串时务必使用 `mb_*` 函数。例如，如果对一个 UTF-8 字符串使用 `substr()`，那返回的结果中会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，哪怕只有一次忘记了使用，Unicode 字符串就有在接下来的过程中变成乱码的风险。

事实上，不是所有的字符串函数都有一个对应的 `mb_*` 函数，应该在所有的 PHP 脚本（或全局包含的脚本）的开头使用 `mb_internal_encoding()` 函数，然后紧接着在可能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避开风险。

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，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，应该始终将其设为 UTF-8¹。

最后，如果分布式的应用程序中不能确定 `mbstring` 扩展一定开启的话，可以考虑使用 `patchwork/utf8` 包，这样就会在 `mbstring` 可用时自动使用，否则自动切换回非 UTF-8 函数。

3.1.1 `strpos()`

3.1.2 `strlen()`

3.1.3 `substr()`

3.1.4 `mb_strpos()`

3.1.5 `mb_strlen()`

3.1.6 `mb_substr()`

3.2 Browser

3.2.1 `htmlentities()`

3.2.2 `htmlspecialchars()`

3.2.3 `mb_http_output()`

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。

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，现在更好的做法是在 `Content-Type` 响应头中进行设置，可以让处理速度更快。

```
<?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.';
```

¹从 PHP 5.4.0 开始, `htmlentities()` 和 `htmlspecialchars()` 的编码都被默认设为了 UTF-8。


```
// 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>
```

3.2.4 mb_internal_encoding()

3.3 Database

如果使用 PHP 来操作 MySQL，即使做到了上面的每一点，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。

为了确保字符串从 PHP 到 MySQL 都使用 UTF-8，需要检查确认数据库和数据表都设定为 `utf8mb4` 字符集和整理，并且确保 PDO 连接请求也使用了 `utf8mb4` 字符集。

在实践中，为了完整的 UTF-8 支持必须使用 `utf8mb4` 而不是 `utf8`。

Chapter 4

Data types

PHP 主要有四种标量类型，标量可以认为是数据结构中最基本的单元，只能存储一个固定的数据。

- 整型 (integer)
- 浮点型 (float)
- 布尔型 (boolean)
- 字符串 (string)

在定义字符串数据类型时，单引号和双引号在功能上有明显的区别。

- 双引号字符串支持变量的解析和转义字符。
- 单引号字符串不支持变量解析以及转义字符，如果单引号字符串中包含 “'”，那么需要在前面加一个 “\” 进行转义。

单引号在定义简单的字符串时是更加高效的处理方式，双引号则需要花费额外的开销来处理字符串的转义和变量的解析，因此在没有特别需求时可以使用单引号。

另外，PHP 还支持使用定界符 <<< 来定义字符串，使用上字符串必须包含在一组定界标识符中，而且定界结束符必须顶格书写。

```
<?php
$string=<<<EOD
    Are you ok?
EOD;

echo <<< EOT
    $string
    I'm ok.
EOT;
?>
```

定界符“<<<”后面紧跟的就是定界标识符，同样可以由字母、数字或下划线组成，并且不能以数字开始。

定界符中的字符串可以被解析，可以使用转义字符，但是不必转义引号，定界符可以用于定义或输出大量的文本的情况。

PHP 支持的两种复合类型分别是数组（array）和对象（object），相对于标量只能存放一个数据，数组可以存放任何类型的多个数据，可以是标量数据、数组、对象、资源以及其他语法结构等。

- 数组中的每一个数据称为一个元素，每个元素包括索引（键名）和值两部分。
- 元素的索引只能由数字或字符串组成，使用整型数字表示索引时类似于 C 语言的数组“下标（index）”。
- 元素的值可以是多种类型。

PHP 提供了多种方法来构造一个数组，其中最简单的是使用数组函数 `array()`，而且在定义数组时可以省略索引。

```
<?php
$key = 7;
$my_stuff = array(
    1,
    'abc',
    'book'=>'PHP',
    13=>'number',
    "Children's Day"=>'1st June',
    $ky=>'a dynamic day',
    'what is the key here', // 14=>'what is the key here'
);
?>
```

省略的索引值将按当前下标最大索引值加 1 计算，因此如果数组中当前的最大索引值为负数，那么下一个索引值变为 0。

数组允许分别对元素赋值，如果有同名元素，那么只保留最后一次的定义，而且所有整数字符的键都被看作是整数。

4.1 Object

对象是一种高级的数据类型，任何事物都可以看作是对象，每一个对象都是由一组属性值和一组方法（或称操作）构成，其中属性表明对象的一种状态，方法通常用来获取或改变对象的属性。

PHP 支持的两种特殊类型分别是 `NULL` 和资源类型，其中 `NULL` 类型表示“空”、“没有”、“不存在”或“未被初始化”等概念，它只有一个值（即 `NULL`）。

`NULL` 和 `False` 可以表达相同的概念，但是二者有着本质的区别。实际上，任何一种类型的数据，如果没有初始化或者被清空，都可以被认为是 `NULL`。

- 布尔型变量可以转换为整数：`True` 转化为 1，`False` 转换为 0。
- 布尔型变量可以转换为字符串：`True` 转化为字符“1”，`False` 转换为字符“0”。

`NULL` 转换为整型后的值为 0，转换为字符串后的值为空字符“”。

4.2 String

PHP 提供了针对字符串和数组的完善的支持，并且 PHP 还提供了自己独特的功能扩展。

4.2.1 curl

PHP 利用 `curl` 库可以对 URL 地址进行解码或编码，不需要自己手动实现。

4.2.2 pcre

PHP 实用 `PCRE` 来实现对正则表达式的支持。

4.2.3 pspell

PHP 利用 `aspell` 库 (<http://aspell.net>) 可以支持 `pspell` 扩展，从而实现单词拼写检查和建议。

4.3 Array

PHP 通过 `serialize()` 和 `unserialize()` 等函数，可以将数组、对象等数据进行序列化编码，从而方便其在数据库或会话周期中的存储。

4.3.1 wddx

PHP 利用 `WDDX` (Web 分布式数据格式) 可以进行数据串行化，从而在应用环境中交换更为复杂的数据结构。

4.4 Resource

资源是由专门的函数建立和使用的，资源本身是一种特殊的数据类型，可以被用户创建、使用和释放。

PHP 实现了自动垃圾回收机制来释放不再需要的资源，一般发生在 PHP 脚本运行结束之后。

按照封装的层次，用户无法获知某个资源的细节，通常包含打开文件、数据库连接、图形画布区域等特殊句柄等。

Chapter 5

Functions

PHP 函数提供了各种不同的功能，例如文件处理、FTP、字符串处理等，而且这些函数的使用方法和 C 语言相近（例如 `printf`）。

PHP 的函数库系统除了提供了大量的内置函数之外，还允许用户通过扩展来引入自己的函数。例如，用户可以通过下面的代码来定义自己的函数 `myFunction()`。

```
function myFunction() { // declares a function, this is named myFunction
    return 'John Doe'; // returns the value 'John Doe'
}

echo 'My name is ' . myFunction() . '!';
// outputs the text concatenated with the return value of myFunction.
// myFunction is called as a result of this syntax.
// The result of the output will be 'My name is John Doe!'
```

在 Zend 引擎的支持和优化下，现在 PHP 可以通过不同的扩展模块来支持各种各样的 Web 应用（例如网络爬虫等）。

```
<?php
function hello(){
    echo "Hello World!\n";
}

hello();
?>
```

Prior to version 5.3, PHP supports quasi-anonymous functions through the `create_function()` function, although they are not true anonymous functions because anonymous functions are nameless, but functions can only be referenced by name, or indirectly through a variable `$function_name()`;

in PHP. As of version 5.3, PHP supports true anonymous functions.

Function calls may be made via variables, where the value of a variable contains the name of the function to call. This is illustrated in the following example:

```
<?php
function hello(){
    return 'Hello';
}
function world(){
    return "World!";
}

$function1 = 'hello';
$function2 = 'world';

echo $function1() . ' ' . $function2();
?>
```

PHP does not support named parameters or parameter skipping. Some core PHP developers have publicly expressed disappointment with this decision. Others have suggested workarounds for this limitation.

PHP gained support for closures in PHP 5.3. True anonymous functions are supported using the following syntax:

```
function getAdder($x) {
    return function($y) use ($x) {
        return $x + $y;
    };
}

$adder = getAdder(8);
echo $adder(2); // prints "10"
```

Here, the `getAdder()` function creates a closure using the parameter `$x` (the keyword `use` imports a variable from the lexical context), which takes an additional argument `$y` and returns it to the caller. Such a function is a first class object, meaning that it can be stored in a variable, passed as a parameter to other functions, etc. For more details see [Lambda functions and closures RFC](#).

The `goto` flow control statement is used as follows:

```
function lock() {
```

```
$file = fopen('file.txt', 'r+');
retry:
if (!flock($file, LOCK_EX | LOCK_NB)) {
    goto retry;
}
fwrite($file, 'Success!');
fclose($file);
}
```

When `flock()` is called, PHP opens a file and tries to lock it. The target label `retry:` defines the point to which execution should return if `flock()` is unsuccessful and `goto retry;` is called. The `goto` statement is restricted and requires that the target label be in the same file and context.

The `goto` statement has been supported since PHP 5.3.

Chapter 6

Objects

Basic object-oriented programming functionality was added in PHP 3 and improved in PHP 4. Object handling was completely rewritten for PHP 5, expanding the feature set and enhancing performance.

In previous versions of PHP, objects were handled like value types (or primitive types). The drawback of this method was that the whole object was copied when a variable was assigned or passed as a parameter to a method. In the new approach, objects are referenced by handle, and not by value.

PHP 5 introduced private and protected member variables and methods, along with `abstract` classes, `final` classes, `abstract` methods, and `final` methods. It also introduced a standard way of declaring constructors and destructors, similar to that of other object-oriented languages such as C++, and a standard exception handling model.

Furthermore, PHP 5 added interfaces and allowed for multiple interfaces to be implemented. There are special interfaces that allow objects to interact with the runtime system. Objects implementing `ArrayAccess` can be used with array syntax and objects implementing `Iterator` or `IteratorAggregate` can be used with the `foreach` language construct.

The static method and class variable features in Zend Engine 2 do not work the way some would expect. There is no virtual table feature in the engine, so static variables are bound with a name instead of a reference at compile time.

This example shows how to define a class, `foo`, that inherits from class `bar`. The function `mystaticfunc` is a public static function that can be called with `foo::mystaticfunc()`;

```
class foo extends bar{  
    function __construct(){
```

```
$doo = "wah dee dee";  
}  
public static function mystaticfunc(){  
    $dee = "dee dee dum";  
}  
}
```

If the developer creates a copy of an object using the reserved word `clone`, the Zend engine will check if a `__clone()` method has been defined or not. If not, it will call a default `__clone()` which will copy the object's properties. If a `__clone()` method is defined, then it will be responsible for setting the necessary properties in the created object. For convenience, the engine will supply a function that imports the properties of the source object, so that the programmer can start with a by-value replica of the source object and only override properties that need to be changed.

The following is a basic example of object-oriented programming in PHP:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>testobject</title>  
  </head>  
  <body>  
    <?php  
      class Person{  
        public $firstName;  
        public $lastName;  
  
        public function __construct($firstName,$lastName= ''){  
          $this->firstName = $firstName;  
          $this->lastName = $lastName;  
        }  
  
        public function greet(){  
          return "Hello, my name is " . $this->firstName . " " . $this->lastName . ".";  
        }  
  
        public static function staticGreet($firstName,$lastName){  
          return "Hello, my name is " . $firstName . " " . $lastName . ".";  
        }  
      }  
    </?php>  
  </body>  
</html>
```

```
$he = new Person('Jim','Green');
$she = new Person('Meimei','Han');
$other = new Person('LiPing');

echo $he->greet();
echo '<br />';
echo $she->greet();
echo '<br />';
echo $other->greet();
echo '<br />';
echo Person::staticGreet('LiPing');
?>
</body>
</html>
```

The visibility of PHP properties and methods is defined using the keywords `public`, `private`, and `protected`. The default is public, if only `var` is used; `var` is a synonym for public. Items declared `public` can be accessed everywhere. `protected` limits access to inherited classes (and to the class that defines the item). `private` limits visibility only to the class that defines the item. Objects of the same type have access to each other's private and protected members even though they are not the same instance. PHP's member visibility features have sometimes been described as "highly useful." However, they have also sometimes been described as "at best irrelevant and at worst positively harmful."

Chapter 7

Filesystem

文件系统是操作系统的重要组成部分，可以将其理解为数据存储的基本单元。

7.1 Directory

PHP 提供的文件和目录函数等可以对本地文件系统进行直接操作，例如创建、读取和删除文件或目录，也可以对文件系统的所有者权限、用户组等信息进行查询和修改等。

在 `php.ini` 中激活 “`allow_url_open`” 选项后，可以使用 PHP 的 `fopen()` 或 `fwrite()` 等函数对远程文件进行操作。

- 读取或下载远程 Web 服务器的文件；
- 登录远程 FTP 服务器进行数据上传和下载；

7.2 Journal

用户使用 PHP 的 `syslog()` 函数还可以实现对分布式日志的管理和维护等。

7.3 File

除了可以针对 HTML 和文本文件进行处理之外，PHP 还可以对符合特定格式的文件进行快速处理。

- CVS
- INI
- XML

另外，PHP 也可以通过 `zlib` 来读取 ZIP 文件等。

在文件输出方面，PHP 支持输出 XHTML、XML、PDF、Flash 和图形文件等。

- PDF: `PDFLib` 库
- Flash: `LibSWF` 或 `Ming` 库
- 图像: `GD` 库

在 Windows 系统中，PHP 通过 `Printer` 扩展可以在服务器端开辟动态缓存空间来直接打印文件。

Chapter 8

Protocol

PHP 支持 FTP、LDAP、IMAP、SNMP、NNTP、POP3、HTTP 和 COM 等通信协议。

另外，PHP 还可以开放原始网络端口来使得任何其他的协议能够协同工作。

PHP 支持和所有 Web 开发语言之间的 WDDX 复杂数据交换，而且 PHP 支持对 Java 对象的使用，这样可以将它们自由地用作 PH 对象。

Chapter 9

Development

9.1 Interface

PHP 对 Apache 和 IIS 等服务器提供了直接的模块接口 (SAPI)，如果 PHP 不能作为模块支持 Web 服务器，还可以作为 CGI 或 FastCGI 处理器来使用，因此可以使用 PHP 的 CGI 可执行程序来处理所有服务器上的 PHP 文件请求。

- SAPI
- CGI

9.2 Kernel

PHP 允许开发者使用 C 语言开发新的功能模块，并且可以编译为 PHP 核心模块，这样就可以在运行时动态加载。

从本质上来说，PHP 的核心功能（例如字符串和数组处理）也是作为核心扩展来实现的，而且 PECL 提供了大量的第三方功能扩展。

9.3 PEAR

PEAR (PHP Extension and Application Repository) 由 Stig S. Bakken 于 2000 年在 PHP 开发者会议上提出，目的是实做可以重复使用的库来提供 PHP 社区使用。

PEAR 库从 PHP4 开始投入开发，其目标为：

- 提供有架构的代码。
- 提供社区可重复使用的库。

- 创建 PHP 编码风格标准。

PEAR 库是 PHP 面向对象的应用和实践的最好例证，成熟的 PEAR 库包括：

表 9.1: PEAR 计划

Authentication	File Formats	Mail	Semantic Web
Benchmarking	File System	Math	Streams
Caching	Gtk Components	Networking	Structures
Configuration	Gtk2 Components	Numbers	System
Console	HTML	Payment	Text
Database	HTTP	PEAR	Tools and Utilities
Date & Time	Images	PHP	Validate
Encryption	Internationalization	Processing	Web Services
Event	Logging	Science	XML

9.4 PECL

除了内置的函数之外，PHP 也可以通过扩展库（extension）来提供数据库连接函数、数据压缩函数或图形处理等。

从 PHP5 开始引入了对全新的 PECL（PHP Extension Community Library）模块的支持，PECL 是 PEAR 打包系统的升级。

用户必须经过编译、安装和加载才能使用 PECL 扩展。例如，可以通过在 `php.ini` 中的 `extension` 指令加载扩展，或者用 `dlopen()` 函数。

9.4.1 pecl

PECL 网站中列出的 PECL 扩展库的发行版本可以用 `pear` 命令来下载和安装，也可以指明具体的版本号。

- `pecl download extname`

```
$ pecl download extname
```

- `pecl install extname`

```
$ pecl install extname
```

pecl 可以下载 `extname` 的源代码并进行编译, 然后将 `extname.so` 安装到 `extension_dir` 中, 这样就可以通过 `php.ini` 加载 `extname.so`¹。

默认情况下, `pecl` 命令不会安装标记为 `alpha` 或 `beta` 状态的包。如果没有 `stable` 包可用, 也可以用以下命令安装一个 `beta` 包:

```
$ pecl install extname-beta
```

也可以用此命令安装一个指定的版本:

```
$ pecl install extname-0.1
```

9.4.2 phpize

在手工编译 PECL 扩展时, `phpize` 命令用于准备 PHP 扩展库的编译环境。

在下面的例子中, 扩展库的源程序位于 `extname` 目录中:

```
$ cd extname
$ phpize
$ ./configure
$ make
# make install
```

如果系统中没有 `phpize` 命令并且使用了预编译的包 (例如 RPM), 那要安装 PHP 包相应的开发版本, 此版本通常包含了 `phpize` 命令以及相应的用于编译 PHP 及其扩展库的头文件, 通过 `phpize --help` 命令可以显示此命令用法。

9.4.3 php-config

`php-config` 是一个简单的命令行脚本用于获取所安装的 PHP 配置的信息。在编译扩展时, 如果安装有多个 PHP 版本, 可以在配置时用 `--with-php-config` 选项来指定使用哪一个版本编译, 该选项指定了相对应的 `php-config` 脚本的路径。

`php-config` 脚本在命令行所能使用的选项可以通过 `-h` 选项来显示:

```
Usage: /usr/local/bin/php-config [OPTION]
Options:
  --prefix          [...]
  --includes        [...]
  --ldflags         [...]
  --libs            [...]
  --extension-dir   [...]
```

¹在 `php.ini` 中激活扩展之后, 需要重新启动 `web` 服务以使更改生效。

```
--include-dir    [...]
--php-binary     [...]
--php-sapis      [...]
--configure-options [...]
--version        [...]
--vernum         [...]
```

表 9.2: php-config 脚本命令行选项

选项	说明
<code>--prefix</code>	PHP 所安装的路径前缀，例如 <code>/usr/local</code>
<code>--includes</code>	列出用 <code>-I</code> 选项包含的所有文件
<code>--ldflags</code>	PHP 编译时所使用的 LD 标志
<code>--libs</code>	PHP 编译时所附加的库
<code>--extension-dir</code>	扩展库的默认路径
<code>--include-dir</code>	头文件的默认路径前缀
<code>--php-binary</code>	PHP CLI 或者 CGI 可执行文件的完整路径
<code>--php-sapis</code>	列出所有可用的 SAPI 模块
<code>--configure-options</code>	重现当前 PHP 在编译时的配置选项
<code>--version</code>	PHP 版本号
<code>--vernum</code>	PHP 版本号，以整数表示

用户的扩展设定应该与所使用的 PHP 可执行文件的设定都保持一致，有些 Web 服务器就会搞混，因为其并不一定使用和 PHP 可执行文件处于同一目录下的 `php.ini` 文件。

要搞清楚具体使用了哪一个 `php.ini` 文件，在 `phpinfo()` 的输出中查看：

```
Configuration File (php.ini) Path C:\WINDOWS
Loaded Configuration File C:\Program Files\PHP\5.2\php.ini
```

激活一个扩展后，保存 `php.ini` 文件并重启动 web 服务器，然后用 `phpinfo()` 再次查看确定，此时新的扩展应该有其自己的一节。

或者在命令行运行：

```
drive:\\path\to\php\executable\php.exe -i
```

如果某扩展并未在 `phpinfo()` 中显示，应该查看日志以确定问题出在哪里。

- 如果是在命令行使用 PHP (CLI)，扩展加载出错信息会直接在屏幕显示。
- 如果在 Web 服务器中使用 PHP，则日志文件的位置与格式各不相同。

最常见的问题是 DLL 文件的位置，`php.ini` 中 “`extension_dir`” 设定的值，以及编译时的设置不匹配。如果问题出在编译时设置不匹配，那可能所下载的 DLL 文件不对。可以尝试重新下载一个设置匹配的扩展。

阅读所使用的 Web 服务器之文档以确定日志文件的位置，这与 PHP 本身并无关系。此外，`phpinfo()` 可以起到很大帮助。

有时可能需要将扩展库静态编译到 PHP 中，此时就需要将扩展库源程序放入 `php-src/ext/` 目录中去并告诉 PHP 编译系统来生成其配置脚本。

```
$ cd /your/phpsrcdir/ext
$ pecl download extname
$ gzip -d < extname.tgz | tar -xvf -
$ mv extname-x.x.x extname
```

这将产生以下目录：

```
/your/phpsrcdir/ext/extname
```

此时强制 PHP 重新生成配置脚本²，然后正常编译 PHP：

```
$ cd /your/phpsrcdir
$ rm configure
$ ./buildconf --force
$ ./configure --help
$ ./configure --with-extname --enable-someotherext --with-foobar
$ make
$ make install
```

是否用 `--enable-extname` 或 `--with-extname` 取决于扩展库，不需要外部库文件的扩展库使用 `--enable`。如果要确认的话，在 `buildconf` 之后运行：

```
$ ./configure -\--help | grep extname
```

9.5 CGI

多线程 Web 服务器接收的所有请求都运行在相同的进程空间（Web 服务器自己的空间）中，该空间仅有一套环境变量。如果想获得 CGI 变量，例如 `PATH_INFO`、`HTTP_HOST` 等，使用原有的 PHP 3.x 的方式（`getenv()`），或使用类似的方式（注册全局变量到 `$_ENV` 环境变量）都是不可行的，只能获得运行中的 Web 服务器的环境变量，而不能获得任何有效的 CGI 变量。

²要运行 “`buildconf`” 脚本，需要 `autoconf 2.13` 和 `automake 1.4+`（更新版本的 `autoconf` 也许能工作，但不被支持）。

Web 服务器的启动脚本事实上是一个 CGI 脚本，因此启动脚本在启动 Web 服务器进程的同时就会包含一些 CGI 变量，但是如果从命令行手动启动 Web 服务器就不会产生这些 CGI 形式的变量。

PHP 默认情况下被编译为 CLI 和 CGI 程序，可以建立一个命令行解释器来处理 CGI 以及非 Web 相关的操作。如果需要为 Web 服务器提供 PHP 支持，那么通常为性能考虑应该使用模块方式。

CGI 可以使 Apache 用户用不同的用户 ID 运行不同的 PHP 页面，不过使用 CGI 方式进行服务器部署可能存在缺陷。

某些服务器提供的环境变量可能没有定义在当前的 CGI/1.1 标准中，只有下列变量定义在其中，其它的变量均作为“供应商扩展 (vendor extensions)”来对待。

- AUTH_TYPE
- CONTENT_LENGTH
- CONTENT_TYPE
- GATEWAY_INTERFACE
- PATH_INFO
- PATH_TRANSLATED
- QUERY_STRING
- REMOTE_ADDR
- REMOTE_HOST
- REMOTE_IDENT
- REMOTE_USER
- REQUEST_METHOD
- SCRIPT_NAME
- SERVER_NAME
- SERVER_PORT
- SERVER_PROTOCOL
- SERVER_SOFTWARE

Chapter 10

Installation

在配置 PHP 应用服务器时，一般有两个方法可以将 PHP 连接到服务器上。

- SAPI
- CGI

PHP 向 Apache 和 IIS 等 Web 服务器提供了一个直接的模块接口 SAPI，其中 IIS 服务器支持微软的 ISAPI，可以极大地提高 IIS 的性能。

ISAPI 是驻留在 Web 服务器中的代码，而且和 Apache 模块的原理类似，ISAPI 在首次调用时就会被载入内存，这样在以后的使用中就不会启动新的进程。

如果 PHP 无法作为 Web 服务器模块，总是可以将其作为 CGI 或 FastCGI 处理器来使用，这样就可以使用 PHP 的 CGI 可执行程序来处理所有服务器上的 PHP 文件请求。

PHP 命令行脚本可以执行能够在离线状态下根据传递给脚本的参数，自动生成一些图片，或处理一些文本文件等工作。

10.1 UNIX

在 Unix 平台下安装 PHP 时可以使用配置和编译过程，或是使用各种预编译的包。

如果用户需要与标准配置不同的功能（例如安全服务器，或者不同的数据库驱动扩展模块），可能需要编译 PHP 和/或 Web 服务器。

为了直接从 Git 源文件或者自己修改过的包进行编译，可能需要某些辅助工具。

- autoconf: 2.13+ (PHP < 5.4.0), 2.59+ (PHP >= 5.4.0)
- automake: 1.4+
- libtool: 1.4.x+ (除了 1.4.2)
- re2c: 版本 0.13.4 或更高

- flex: 版本 2.5.4 (PHP <= 5.2)
- bison: 版本 1.28 (建议), 1.35 或 1.75

Apache 初始的配置和安装过程由 `configure` 脚本中的一系列命令行选项控制, 可以通过 `./configure --help` 命令了解 Apache 所有可用的编译选项及简短解释。

在配置好 Apache 后, 下一步就可以使用 `make` 命令来编译模块和/或可执行文件。

```
# cd /src
# gzip -d httpd-2.4.12.tar.gz
# tar xvf httpd-2.4.12.tar
# cd httpd-2.4.12
# ./configure --help
```

一般情况下, 可以直接使用 Apache 的默认配置来进行编译。例如, 这里将允许 Apache 加载 DSO (Dynamic Shared Object) 模块:

```
# ./configure --enable-dso
# make
# make install
# /usr/local/apache2/bin/apachectl start
```

为了继续安装 PHP 和配置 Apache, 可以暂时停止 Apache 服务。

```
# /usr/local/apache2/bin/apachectl stop
```

在编译安装 PHP 之前, 首先需要释放 PHP 的官方压缩包, 然后进行 PHP 的配置。

```
# cd /src
# gzip -d php-5.6.10.tar.gz
# tar xvf php-5.6.10.tar
```

Apache 和 PHP 都被设计为模块化结构, 因此可以配合各种不同的功能模块来进行工作, 并且获得更多的扩展功能。

在编译 PHP 时可以手动配置编译选项来控制对应模块的使用, 而且编译选项指定的模块必须事先已经安装, 否则配置脚本将会报错。例如, 下面的示例中说明了编译 PHP 时的一些常见选项:

- `--prefix` 指定 PHP 的安装目录;
- `--with-apxs2` 用于提供对 Apache2 的动态共享模块的支持。

```
# cd php-5.6.10
# ./configure --prefix=/usr/local/php \
--with-apxs2=/usr/local/apache2/bin/apxs \
--with-zlib-dir=/usr/local/zlib2 \
--with-mysql=/usr/local/mysql \
--with-pgsql=/usr/local/pgsql \
```

在配置脚本检测通过后，就可以进行编译和安装，同样也可以重复上述步骤来改变编译选项。

```
# make
# make install
```

在编译并安装 PHP 之后，接下来将建立配置 `php.ini`，从而可以进一步修改 PHP 的选项。

默认情况下，配置阶段在 `/usr/local/lib` 目录下建立 `php.ini`，因此可以使用 `configure` 命令的 `--with-config-file-path=/path` 选项来手动指定。

```
# cp php.ini-production /usr/local/lib/php.ini
```

在 Apache 的 `httpd.conf` 中可以配置 PHP，从而确保 Apache 可以调用 PHP 模块和解析 PHP 脚本文件。

```
# cat >> httpd.conf
## 载入PHP5模块
LoadModule php5_module modules/libphp5.so
## 载入PHP4模块
LoadModule php4_module modules/libphp4.so

## 添加.php扩展名并进行解析
AddType application/x-httpd-php .php

## 添加.phps扩展来显示高亮的PHP源文件
AddType application/x-httpd-php-source .phps
```

在 Windows 操作系统中安装 MySQL 等数据库时，为了保持目录结构一致，同样也可以安装到 `c:\usr\local\mysql`，并且使用相应的配置工具来管理数据库服务。

在 MySQL 配置工具中，可以了解当前系统的运行状况，并提供系统环境、启动检测、服务器参数、配置文件、错误文件、系统变量、服务器进程、数据库和表结构以及综合报告等信息。

用户可以通过 MySQL 的配置文件 `my.ini` 对 MySQL 数据库服务器进行手动配置，并且在配置完毕后重启 MySQL 服务使其生效。

```
#This file was made using the WinMySQLAdmin Tool

[mysqld]
port=3306
basedir=c:/usr/local/mysql
#bind-address=192.168.1.10
```

```
datadir=c:/usr/local/mysql/data
#tmpdir=
#set-variable=key_buffer=16M

[WinMySQLAdmin]
Server=c:/usr/local/mysql/bin/mysql-nt.exe
user=root
password=p@ss
```

在 UNIX/Linux 操作系统中安装 MySQL 时, 可以考虑使用 RPM 自动安装包, 或者从二进制文件进行安装, 不需要自己手动编译, 只需要将安全文件包释放到 MySQL 目录中。

为了提高 MySQL 的灵活性, 在安装之前可以为其建立对应的用户和用户组。

```
# addgroup mysql
# adduser -g mysql mysql
# cd src/
# gunzip < mysql-5.7.6-linux-glibc2.5-x86_64.tar.gz | tar xvf -
# ln -s mysql-5.7.6-linux-glibc2.5-x86_64 mysql
```

在 MySQL 的安装目录中包括程序文件和子目录等, 而且必须将 MySQL 主目录添加到环境变量 PATH 中。

另外, 在 MySQL 的子目录 `scripts` 中包含一个数据库初始化脚本, 可以用来进行数据库初始化。

在使用 `root` 身份执行数据库初始化脚本 `mysql_install_db` 时, 应该显式地使用“`--user`”指定 MySQL 的访问用户帐号。

```
# cd mysql
# ./scripts/mysql_install_db --user=mysql
```

除此之外, 还需要让 `root` 用户和 `mysql` 用户对 MySQL 目录获得适当的权限。

```
# chown -R root /usr/local/mysql
# chown -R root /usr/local/mysql/bin
# chown -R mysql /usr/local/mysql
# chgrp -R mysql /usr/local/mysql
```

在完成对 MySQL 的安装和设置后, 可以启动 MySQL 数据库服务。

```
# /usr/local/mysql/bin/mysql_safe --user=mysql &
```

为了在操作系统启动时自动启动 MySQL 服务, 可以将“`support/mysql.server`”文件复制到系统的启动目录中。

10.2 Mac OSX

Mac OS X 有内建支持的 PHP，编译安装类似于 Unix 系统下的安装，也可以使用预编译和打包的 PHP 版本来安装标准配置的 PHP。

如果需要不同的功能集（比如安全服务器或不同的数据库驱动程序），可能需要自己编译 PHP 和/或 Web 服务器。

10.2.1 Homebrew

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。

10.2.2 MacPorts

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
```

也可以执行 `select` 命令来切换当前的 `php` 版本：

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

10.2.3 PHPBrew

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

另外，`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`)。

最后一个可以用来控制安装 PHP 版本的选择就是自行编译。如果使用这种方法，必须先确认是否已经通过 Apple's Mac Developer Center 下载、安装了 Xcode 或者“Command Line Tools for XCode”。

上述列出的解决方案主要是针对 PHP 本身，并不包含 Apache、Nginx 或者 SQL 服务器，使用 MAMP 或 XAMPP 等集成包可以安装这些软件并且将他们绑在一起，只是易于安装的背后也牺牲了一定的弹性。

在默认的 Web 服务器中启用 PHP 时，只需将 Apache 配置文件 `httpd.conf` 中的若干配置指令前面的注释符号去掉，这样 CGI 或 CLI 默认都可以使用（可以很容易的被终端程序使用）。

在搭建本地 PHP 开发环境时，标准的安装类型为 `mod_php`，接下来在 Apache Web 服务器中启用 PHP 时包含以下的步骤：

1. 找到并打开 Apache 的配置文件。

默认情况下，这个配置文件的位置¹是：`/private/etc/apache2/httpd.conf`。

2. 使用文本的编辑器取消注释。

```
# LoadModule php5_module libexec/httpd/libphp5.so
...
# AddModule mod_php5.c
```

3. 确保将所需要的文件扩展名解析为 PHP。

确保将所需要的文件扩展名解析为 PHP（例如 `.php`、`.html` 以及 `.inc`），否则不能正常运行。

自 Mac Panther 开始，将以下的配置写入 `httpd.conf` 后，只要 PHP 被启用就可以将 `.php` 文件自动解析为 PHP 脚本。

```
<IfModule mod_php5.c>
    # If php is turned on, we respect .php and .phps files.
    AddType application/x-httpd-php .php
    AddType application/x-httpd-php-source .phps

    # Since most users will want index.php to work we
    # also automatically enable index.php
    <IfModule mod_dir.c>
        DirectoryIndex index.html index.php
    </IfModule>
</IfModule>
```

¹在 Mac OS X 10.5 之前的版本中捆绑的是旧版本的 PHP 和 Apache。因此在旧的计算机中 Apache 配置文件的位置可能是 `/etc/httpd/httpd.conf`。

4. 确保 `DirectoryIndex` 加载了所需的默认索引文件。

通常情况下，在 `httpd.conf` 中将 `DirectoryIndex` 设置为 `index.php` 和 `index.html`，可以根据实际情况进行相应的调整。

5. 设置 `php.ini` 的位置或者使用默认的位置。

Mac OS X 上通常默认的位置是 `/usr/local/php/php.ini`，如果没有使用 `php.ini`，PHP 将使用所有的默认值。

6. 定位或者设置 `DocumentRoot`。

`DocumentRoot` 是网站所有文件的根目录，该目录中的文件由 Web 服务器提供服务，从而使得 PHP 文件将在输出到浏览器之前解析为 PHP 脚本。

通常情况下，Web 服务器的默认的路径是 `/Library/WebServer/Documents`，可以根据需要在 `httpd.conf` 中设置为任何其他目录。

另外，用户自己的缺省 `DocumentRoot` 是 `/Users/yourusername/Sites`。

7. 创建 `phpinfo()` 文件。

`phpinfo()` 将会显示 PHP 的相关系统信息。例如，可以在 `DocumentRoot` 下创建一个 PHP 文件，其代码如下：

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

8. 重启 Apache，然后从浏览器访问 `phpinfo` 文件。

要重启 Apache，可以在 shell 中执行 `sudo apachectl graceful`，也可以停止/启动 OS X 系统首选项中的“Personal Web Server”选项。

默认情况下，从浏览器访问本地文件的 URL 一般类似于 `http://localhost/info.php`，或者使用 `http://localhost/~yourusername/info.php` 来访问用户自己 `DocumentRoot` 中的文件。

CLI（或者旧版本中的 CGI）一般文件名为 `php`，其路径可能是 `/usr/bin/php`。

用户可以打开一个终端并执行 `php -v` 来检查当前运行的 PHP 的版本，在脚本中调用 `phpinfo()` 也会显示相关的信息。

10.3 Windows

用户可以下载 PHP 的 Windows 二进制包，如果为 PHP 所在的根目录（`php.exe` 所在的文件夹）设置好环境变量（例如 `PATH`）就可以从命令行中直接执行 PHP。不过，在安装 PHP 之前需要确保系统中已经安装好 HTTP（Web）服务器，并确认其正常工作，而且不同的 PHP 版本的目录结构可能是不同的。

如果需要将生产环境部署在 Windows 上，那么 IIS7 可以提供最稳定和最佳的性能，

表 10.1: PHP5 和 PHP4 的文件对应关系

PHP5	PHP4	备注
php5ts.dll	php4ts.dll	核心动态库
ext/php_*.dll	extensions/php_*.dll	PHP 的扩展库文件
.dll	dlls/.dll	其他扩展文件
php5apache.dll	sapi/php4apache.dll	Apache1.3.x 模块
php5apache2.dll	sapi/php4apache2.dll	Apache 2.0.x 模块
php5isapi.dll	sapi/php4isapi.dll	ISAPI 模块, 支持 IIS
php.exe、php-win.exe	cli/php.exe	CLI 可执行文件
php-cgi.exe	php.exe	CGI 可执行文件

用户可以使用 IIS7 的图形化插件 `phpmanager` 来设置并管理 PHP。

IIS7 有内置的 FastCGI，只需要将 PHP 配置为它的处理器就可以使用。

10.3.1 Compile

如果有开发环境（例如 Microsoft Visual Studio），那么也可以编译 PHP 的源程序来进行安装。

- PHP 5.5.22/5.6.6 使用 Microsoft VC11 编译，不支持 Windows XP；
- PHP 5.4.38 使用 Microsoft VC9 编译，支持 Windows XP。

为了和 Linux 下的 PHP 安装目录匹配，可以将 PHP 的安装目录指定为 `c:\usr\local\php`，同时也需要将 PHP 的安装目录加入系统的 Path 环境变量中。

```
Windows c:\usr\local\php
UNIX/Linux /usr/local/php
```

在 Windows 系统下安装好 PHP 之后，可能还需要加载各种扩展库以实现更多的功能。

另外，虽然互联网上还有一些多合一的安装程序，但是它们没有一个是被 PHP 官方认可的，因此从<http://www.php.net/downloads.php>下载官方 Windows 安装包是系统安全和优化的最好选择，然后就可以为 IIS、PWS 或 Xitami 安装 CGI 版本的 PHP，并配置好 Web 服务器。

10.3.2 Install

PHP 官方安装程序支持两种安装类型（标准和高级），并且安装向导在安装过程中会收集足够的信息来设置 `php.ini` 文件，然后配置好 Web 服务器来使用 PHP。

- 标准：使用合理的默认配置进行安装；
- 高级：在配置中询问相关问题。

一般情况下，在 PHP 安装完成后会提示重新启动系统，并重启服务器，或直接使用 PHP。

10.3.3 Manual

虽然 Windows 安装程序是让 PHP 工作的最容易的方法，但是它有很多限制。例如，它不支持自动安装 PHP 扩展，因此使用安装程序安装 PHP 不是最佳方式，而且 PHP 官方安装程序不包含任何外部的 PHP 扩展（`PHP_*.dll`）。如果需要，可以在 Windows ZIP 包和 PECL 中找到。

旧版本的 PHP 安装程序不能自动配置 Apache，因此需要手动进行配置，而且使用 Windows 安装程序来安装的 PHP 不是安全的。

如果需要一个安全的 PHP 设置，最好使用手动方式安装，并手动设置好每个选项，这样就可以将 PHP 用于在线的生产服务器中。

PHP 的 Windows 安装程序是用 Wix 工具包 (<http://wix.sourceforge.net/>) 基于 MSI 建立的，使用这种方式安装 PHP 时，首先会提示选择要配置的 Web 服务器，以及所需的任何配置细节，然后它将安装并配置 PHP 和所有内置以及 PECL 扩展库，并且配置相关的 Web 服务器（例如 IIS、Apache 以及 Xitami）。

在安装并激活相关特性和扩展时，每个项目的下拉菜单中需要选择“Will be installed on local hard drive”来激活安装该单个项目。

如果选择“Entire feature will be installed on local hard drive”，将会安装所有该项目的子项（例如对“PDO”选择了此选项，则会安装所有的 PDO 驱动）²。

在接下来设定 Windows 使用的 PHP 和 `php.ini` 文件时，可以配置特定的 Web 服务器使用 PHP，也可以通过手工配置来支持其他的 Web 服务器。

新版本 PHP 的安装程序支持无声模式，有助于系统管理员更容易发布 PHP。例如，可以使用下面的命令来使用无声模式：

```
msiexec.exe /i php-VERSION-win32-install.msi /q
```

²事实上，并不推荐安装所有的扩展库，因为其中许多库还需要 PHP 之外的支持才能正常工作。不过，在安装好之后，可以从控制面板的程序和功能里可以调出更改模式来激活或关闭某扩展和功能。

在不同的服务器环境中，可以根据实际需要安装目录作为命令行参数传递给安装程序，例如下面的命令将 PHP 安装到 c:\php：

```
msiexec.exe /i php-VERSION-win32-install.msi /q INSTALLDIR=c:\php
```

另外，可以用同样的语法指定 Apache 配置目录 (APACHEDIR)、Sambar 服务器目录 (SAMBARDIR) 以及 Xitami 服务器目录 (XITAMIDIR)，而且还可以指定安装哪些特性。例如，下面的命令说明要安装 mysqli 扩展和 CGI 可执行程序：

```
msiexec.exe /i php-VERSION-win32-install.msi /q ADDLOCAL=cgi,ext_php_mysql
```

在 Windows 中，可安装的 PHP 特性列表如下：

- MainExecutable - php.exe 可执行文件（默认已包含）
- ScriptExecutable - php-win.exe 可执行文件
- ext_php_* - 扩展库（例如 MySQL 是 ext_php_mysql）
- apache13 - Apache 1.3 模块
- apache20 - Apache 2.0 模块
- apache22 - Apache 2.2 模块
- apacheCGI - Apache CGI 可执行文件
- iis4ISAPI - IIS ISAPI 模块
- iis4CGI - IIS CGI 可执行文件
- iis4FastCGI - IIS CGI 可执行文件
- NSAPI - Sun/iPlanet/Netscape 服务器模块
- netserve - NetServe Web 服务器之 CGI 可执行文件
- Xitami - Xitami CGI 可执行文件
- Sambar - Sambar 服务器 ISAPI 模块
- CGI - php-cgi.exe 可执行文件
- PEAR - PEAR 安装程序
- Manual - CHM 格式的 PHP 手册

10.3.4 Update

如果要升级 PHP，可以正常使用图形模式或者从命令行运行安装程序，可以看到安装程序会读取当前安装的选项，移除旧的安装，并用同样选项重新安装 PHP，因此建议用这种方式更新 PHP 而不是手工替换安装目录下的文件。

10.3.5 Extend

在 Windows 下安装好 PHP 和 Web 服务器之后，可能想要安装一些扩展库来获得更多功能。

PHP 扩展库的 DLL 文件都具有 `php_` 前缀，可以通过修改 `php.ini` 来选择当 PHP 启动时加载哪些扩展库，也可以在脚本中通过使用 `dl()` 来动态加载。

很多扩展库都内置于 Windows 版的 PHP 之中，这意味着要加载这些扩展库，额外的 DLL 文件和 `extension` 配置指令都不需要。

Windows 下的 PHP 扩展库列表列出了需要或曾经需要额外 PHP DLL 文件的扩展库，而且 PHP 搜索扩展库的默认位置在不同的 PHP 版本中是不同的。

- PHP 4 中是 `drive:\php4\extensions`;
- PHP 5 中是 `drive:\php5`。

在 `php.ini` 文件³可以根据用户自己的 PHP 设置来修改扩展目录。

- 修改 `extension_dir` 设置以指向用户放置扩展库的目录或者说放置 `php_*.dll` 文件的位置。

```
extension_dir = drive:\php\extensions
```

- 启用指定的扩展库，需要去掉 `php.ini` 中该行 `extension=php_*.dll` 前的注释符号 (`;`)。

```
// 将这一行
;extension=php_bz2.dll

// 改成这样
extension=php_bz2.dll
```

- 有些扩展库需要额外的 DLL 才能工作。
对于 Oracle (`php_oci8.dll`) 等所需要的 DLL 没有绑定在发行包里的情况，可以将绑定的 DLL 从 `drive:\php\dlls` 拷贝到主目录 `drive:\php` 中，并且需要将 `drive:\php` 放到系统路径 `PATH` 中去。
- 某些 DLL 没有绑定在 PHP 发行包中，例如 PECL 提供的 PHP 扩展库需要单独下载。
下表说明了哪些扩展库需要额外的 DLL。

表 10.2: PHP 扩展库

扩展库	说明	注解
<code>php_bz2.dll</code>	<code>bzip2</code> 压缩函数	无

³如果运行服务器模块版的 PHP，在修改了 `php.ini` 之后别忘了重新启动 web 服务器以使其改动生效。

扩展库	说明	注解
php_calendar.dll	Calendar 日历转换函数	自 PHP 4.0.3 起内置
php_crack.dll	Crack 密码破解函数	无
php_ctype.dll	ctype 家族函数	自 PHP 4.3.0 起内置
php_curl.dll	CURL, 客户端 URL 库函数	需要: libeay32.dll, sslseay32.dll (已附带)
php_dba.dll	DBA: 数据库 (dbm 风格) 抽象层函数	无
php_dbase.dll	dBase 函数	无
php_dbx.dll	dbx 函数	
php_domxml.dll	PHP 4 DOM XML 函数	PHP <= 4.2.0 需要: libxml2.dll (已附带), PHP >= 4.3.0 需要: iconv.dll (已附带)
php_dotnet.dll	.NET 函数	PHP <= 4.1.1
php_exif.dll	EXIF 函数	需要 php_mbstring.dll。并且在 php.ini 中, php_exif.dll 必须在 php_mbstring.dll 之后加载
php_fbsql.dll	FrontBase 函数	PHP <= 4.2.0
php_fdf.dll	FDF: 表单数据格式化函数	需要: fdfk.dll (已附带)
php_filepro.dll	filePro 函数	只读访问
php_ftp.dll	FTP 函数	自 PHP 4.0.3 起内置
php_gd.dll	GD 库图像函数	在 PHP 4.3.2 中移除。此外注意在 GD1 中不能用真彩色函数, 用 php_gd2.dll 替代。
php_gd2.dll	GD 库图像函数	GD2
php_gettext.dll	Gettext 函数	PHP <= 4.2.0 需要 gnu_gettext.dll (已附带), PHP >= 4.2.3 需要 libintl-1.dll, iconv.dll (已附带)
php_hyperwave.dll	HyperWave 函数	无
php_iconv.dll	ICONV 字符集转换	需要: iconv-1.3.dll (已附带), PHP >= 4.2.1 需要 iconv.dll
php_ifx.dll	Informix 函数	需要: Informix 库
php_iisfunc.dll	IIS 管理函数库	无
php_imap.dll	IMAP, POP3 和 NNTP 函数	无
php_ingres.dll	Ingres II 函数	需要: Ingres II 库
php_interbase.dll	InterBase 函数	需要: gds32.dll (已附带)
php_java.dll	Java 函数	PHP <= 4.0.6 需要: jvm.dll (已附带)
php_ldap.dll	LDAP 函数	PHP <= 4.2.0 需要 libsasl.dll (已附带), PHP >= 4.3.0 需要 libeay32.dll, sslseay32.dll (已附带)
php_mbstring.dll	Multi-Byte String 多字节字符串函数	无
php_mcrypt.dll	Mcrypt 加密函数	需要: libmcrypt.dll
php_mhash.dll	Mhash 函数	PHP >= 4.3.0 需要: libmhash.dll (已附带)
php_mime_magic.dll	Mimetype 函数	需要: magic.mime (已附带)
php_ming.dll	Ming 函数 (Flash)	无
php_msql.dll	mSQL 函数	需要: msql.dll (已附带)
php_mssql.dll	MSSQL 函数	需要: ntwdblib.dll (已附带)
php_mysql.dll	MySQL 函数	PHP >= 5.0.0 需要 libmysql.dll (已附带)
php_mysqli.dll	MySQLi 函数	PHP >= 5.0.0 需要 libmysql.dll (PHP <= 5.0.2 中是 lib-mysqli.dll) (已附带)
php_oci8.dll	Oracle 8 函数	需要: Oracle 8.1+ 客户端库
php_openssl.dll	OpenSSL 函数	需要: libeay32.dll (已附带)
php_overload.dll	PHP 4 对象过载函数	自 PHP 4.3.0 起内置, 自 PHP 5.0.0 起移除

扩展库	说明	注解
php_pdf.dll	PDF 函数	无
php_pgsql.dll	PostgreSQL 函数	无
php_printer.dll	Printer 打印机函数	无
php_shmop.dll	Shared Memory 共享内存函数	无
php_snmp.dll	SNMP 函数	仅用于 Windows NT!
php_soap.dll	SOAP 函数	PHP >= 5.0.0
php_sockets.dll	Socket 函数	无
php_sybase_ct.dll	Sybase 函数	需要: Sybase 客户端库
php_tidy.dll	Tidy 函数	PHP >= 5.0.0
php_tokenizer.dll	Tokenizer 函数	自 PHP 4.3.0 起内置
php_w32api.dll	W32api 函数	无
php_xmlrpc.dll	XML-RPC 函数	PHP >= 4.2.1 需要 iconv.dll (已附带)
php_xslt.dll	XSLT 函数	PHP <= 4.2.0 需要 sablot.dll, expat.dll (已附带)。PHP >= 4.2.1 需要 sablot.dll, expat.dll, iconv.dll (已附带)。
php_yaz.dll	YAZ 函数	需要: yaz.dll (已附带)
php_zip.dll	Zip 文件函数	只读访问
php_zlib.dll	ZLib 压缩函数	自 PHP 4.3.0 起内置

10.3.6 Config

Apache 支持以 CGI 或 Apache 模块的方式运行 PHP，它们是基于对 Apache 不同的配置方案来完成的。

对 Apache 服务器的配置主要是通过修改 httpd.conf 来实现的。例如，如果需要以 CGI 方式运行 PHP5，那么需要向 httpd.conf 中添加如下的代码：

```
# 设置PHP目录的别名
ScriptAlias /php/ "c:/usr/local/php/"

# 关联特定的扩展名，以解析PHP文件
AddType application/x-httpd-php .php

# 设置使用PHP5执行CGI脚本的程序
Action application/x-httpd-php "/php/php-cgi.exe"
```

对于 PHP4，则需要在 httpd.conf 中添加下面的代码：

```
ScriptAlias /php/ "c:/usr/local/php/"
AddType application/x-httpd-php .php

# 设置使用PHP4执行CGI脚本的顺序
Action application/x-httpd-ph "/php/php.exe"
```

如果以 Apache 模块的方式来运行 PHP5，则需要在 httpd.conf 中添加下面的代码：

```
# 载入PHP5模块
LoadModule php5_module "c:/usr/local/php/php5apache2.dll"
AddType application/x-httpd-php .php

# 配置php.ini的路径
PHPIniDir "c:/usr/local/php"
```

如果 Apache 模块的方式来运行 PHP4, 则需要将 php4apache2.dll 从 sapi 目录中拷贝到 PHP 安装目录下, 并且在 httpd.conf 中添加下面的代码:

```
# 载入PHP4模块
LoadModule php4_module "c:/usr/local/php/php4apache2.dll"
AddType application/x-httpd-php .php

# 配置php.ini的路径
PHPIniDir "c:/usr/local/php"
```

默认情况下, Apache 提供的 Web 虚拟目录为 Apache 安装目录下的 htdocs 目录。

为了测试 PHP 在 Apache 服务器中运行是否正常, 可以在 Web 虚拟目录 (即 htdocs) 中创建下面的测试文件, 并且在浏览器中通过<http://localhost/phpinfo.php>进行访问。

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

IIS 支持以 CGI 或 ISAPI 模块的方式集成 PHP。

- 如果以 CGI 方式集成 PHP, 可以将 PHP 映射到 CGI 执行程序 (例如 php-cgi.exe)。
- 如果以 ISAPI 方式集成 PHP, 可以将 PHP 映射到 ISAPI 模块 (例如 php5isapi.dll)。

另外, 如果用户需要使用 PHP 的 HTTP 认证功能, 则需要在 ISAPI 筛选器中设置 PHP 的筛选器 (例如 php5isapi.dll)。

10.3.7 Regedit

在 Windows 服务器上进行设置时也可以通过注册表来对 PHP 进行配置。

为了保护注册表安全, 可以使用注册表文件的方式来向注册表中添加新内容。例如, 下面的注册表文件可以用来以 ISAPI 方式配置 IIS 服务器:

```
Windows Registry Editor Version 5.0
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="c:\\usr\\local\\php\\php5isapi.dll"
```

如果以 CGI 方式配置 PHP, 则需要使用下面的注册表文件:

```
Windows Registry Editor Version 5.0
```

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\w3svc\parameters\Script Map]
".php"="c:\\usr\\local\\php\\php-cgi.exe"
```

在对 IIS 进行设置后，需要重启 IIS 服务器来使修改生效。

10.4 Docker

Chapter 11

Accelerator

11.1 Overview

通常情况下，PHP 都是解释执行的，不过也可以通过特定的工具（例如加速器）来将 PHP 代码编译为中间字节码来获得更好的性能。

在 Web 应用系统中，加速器本身是一个加快访问速度的代理服务器，使用硬件或软件来实现。

Web 加速器可以安装在客户端、移动设备以及 ISP 服务器上，并且对数据进行压缩等操作后传输给用户。

PHP 加速器可以被设计为 PHP 扩展来增强性能，而且大多数 PHP 加速器都是将 PHP 程序和模块编译为可直接执行的 `opcode/bytecode` 并缓存，这样在需要调用 PHP 来处理客户端请求可以直接读取来加快处理速度。

11.1.1 eAccelerator

为了更进一步提高性能，可以将 `opcode/bytecode` 载入内存缓冲区，这样就可以从内存直接执行，从而减小在运行时从硬盘读取和从内存拷贝的限制。

例如，从 Turck MMCache 衍生的 `eAccelerator` 可以将 PHP 程序、PHP 核心和相关库函数预先编译后缓存在共享内存中，在需要时可以被重复使用，从而实现加速 PHP 程序的目的。

```
# tar -jxvf eaccelerator.tar.gz
# cd
eaccelerator
# phpize
# ./configure--enable-eaccelerator=shared --with-php-config=/usr/bin/php-config
# make
# make install
```

默认情况下，新编译安装的模块位于/usr/lib64/php/modules/中。

为了预先加载扩展 PHP 模块，可以在/etc/ld.so.conf.d/中创建 ls.so.conf 文件，并新增如下配置：

```
# vim /etc/ld.so.conf.d/ld.so.conf
/usr/lib64/php/modules
# ldconfig
```

eAccelerator 根据当前版本的 PHP 核心进行编译，因此在更新 PHP 后需要手动更新 eAccelerator，并且每次都要在/etc/php.ini 中添加 eAccelerator 的配置信息，或者在/etc/php.d/中创建 eAccelerator 的配置文件。

```
;;;;;;;;;;;;;
; http://eaccelerator.net/
; date:
;;;;;;;;;;;;;
extension="eaccelerator.so"
eaccelerator.shm_size="16"
eaccelerator.cache_dir="/tmp/eaccelerator"
eaccelerator.enable="1"
eaccelerator.optimizer="1"
eaccelerator.check_mtime="1"
eaccelerator.debug="0"
eaccelerator.filter=""
eaccelerator.shm_max="0"
eaccelerator.shm_ttl="0"
eaccelerator.shm_prune_period="0"
eaccelerator.shm_only="0"
eaccelerator.compress="1"
eaccelerator.compress_level="9"
```

如果要检查 eAccelerator 模块的运行状态，可以使用 phpinfo() 函数或 php -i 进行查询。

对于存放 eAccelerator 的临时文件的目录要注意设置适当的权限，并且在重启服务器才能使用 eAccelerator。

```
# mkdir /tmp/eaccelerator
# chmod 777 /tmp/eaccelerator
# apachectl restart
```

除了可以把将 PHP 编译产生的 bytecode 暂存在共享内存内重复使用来提高执行效率，Alternative PHP Cache (APC) 还可以对中间码进行优化来和所有版本的 PHP 协同使用。

```
# pecl install APC
downloading APC-3.1.13.tgz ...
```

```
Starting to download APC-3.1.13.tgz (171, 591 bytes)
....done: 171,591 bytes
55 source files, building
running: phpize
Configuring for:
PHP Api Version: 20121113
Zend Module Api No: 20121212
Zend Extension Api No: 220121212
Enable internal debugging in APC [no] : yes
Enable per request file info about files used from the APC cache [no] : yes
Enable spin locks (EXPERIMENTAL) [no] : yes
Enable memory protection (EXPERIMENTAL) [no] : yes
Enable pthread mutexes (default) [no] : yes
Enable pthread read/write locks (EXPERIMENTAL) [yes] : yes
...
```

由 Lighttpd 计划提供的 XCache 解决了其他 opcache 存在的问题（比如可以支持新的 PHP 版本），并且在 Linux 下可以支撑高负载状况，另外还支持 ThreadSafe/Windows。

11.2 Encoder

默认情况下，PHP 源代码是可以从服务器上直接读取的，这样虽然可以提高 PHP 的弹性，但是也相对会造成安全危机和性能下降等问题。

PHP 编码器可以对 PHP 代码进行编码来保护 PHP 源代码不被读取，也可以提升运行的性能，因此使用 PHP 编码器（例如 Zend 提供的脚本优化器）将 PHP 程序编译成字节码 (byte code)，然后再通过服务器上安装对应的程序来运行 PHP 脚本。

除了通过编码器加速之外，PHP 还可以通过动态的高速缓存机制来提升速度。

11.3 Engine

模板引擎让 PHP 应用程序可以实现逻辑和使用界面上的分离，让程序开发更容易进行。例如，Smarty 是使用 PHP 实现的模板系统，可以简化 PHP 应用程序的开发过程，其核心模板引擎可以通过标签来创建 Web 组件。

不过模板引擎存在性能方面的争议，因为 PHP 本身就是一个模板引擎，使用模板引擎反而变成“重新发明了轮子”。

模板引擎最主要的好处就是让不懂 PHP 代码的人也可以参与使用界面的开发，因为模板引擎的语言远比 PHP 简单。例如，MVC 模式就是对模板引擎的最好应用，从而可以让 PHP 编程人员可以和 HTML 前端程序员分工合作。

11.4 Compiler

Facebook 在 2010 年发布的 HipHop 编译器可以把 PHP 源代码编译成 C++ 来提高速度，并且降低 CPU 的负载。

11.5 WSGI

WSGI (Python Web Server Gateway Interface) 是为 Python 语言定义的 Web 服务器和 Web 应用程序或框架之间的一种简单而通用的接口，现在许多其它语言中也出现了类似接口。

一般而言，Web 应用框架的选择将限制可用的 Web 服务器的选择，反之亦然。

最初，Python 应用程序通常是 CGI、FastCGI、mod_python，或者特定 Web 服务器的自定义的 API 接口而设计的。

基于现存的 CGI 标准而设计的 WSGI 可以作为 Web 服务器与 Web 应用程序或应用框架之间的一种低级别的接口来方便可移植 Web 应用开发。具体来说，WSGI 可以划分两个部分，分别为“服务器”（或“网关”）和“应用程序”（或“应用框架”）。

在处理一个 WSGI 请求时，服务器会为应用程序提供环境信息及一个回调函数 (Callback Function)。当应用程序完成处理请求后，通过前述的回调函数将结果回传给服务器。

WSGI 中间件同时实现了 API 的两方，因此可以在 WSGI 服务和 WSGI 应用之间起调和作用，因此从 WSGI 服务器的角度来说，中间件扮演应用程序，而从应用程序的角度来说，中间件扮演服务器。

WSGI “中间件” 组件可以执行以下功能：

- 重写环境变量后，根据目标 URL 将请求消息路由到不同的应用对象。
- 允许在一个进程中同时运行多个应用程序或应用框架。
- 负载均衡和远程处理，通过在网络上转发请求和响应消息。
- 进行内容后处理（例如应用 XSLT 样式表）。

下面是用 Python 语言写的一个符合 WSGI 的 “Hello World” 应用程序示例。

```
def app (environ, start_response) :  
    start_response ('200 OK', [ ('Content-Type', 'text/plain') ])  
    yield "Hello world!\n"
```

- 第一行定义了一个名为 app 的 callable 并接受两个参数 (environ 和 start_response)，其中 environ 是一个字典包含了 CGI 中的环境变量，start_response 也是一个 callable，并接受两个必须的参数——status (HTTP 状态) 和 response_headers (响应消息的头)。
- 第二行调用了 start_response，状态指定为“200 OK”，消息头指定为内容类型是“text/plain”。
- 第三行将响应消息的消息体返回。

下面是一个调用一个程序并获取它的应答消息的例子。

```
def call_application (app, environ) :
```

```
body = []
status_headers = [None, None]
def start_response (status, headers) :
    status_headers[:] = [status, headers]
    return body.append
app_iter = app (environ, start_response)
try:
    for item in app_iter:
        body.append (item)
finally:
    if hasattr (app_iter, 'close') :
        app_iter.close ()
    return status_headers[0], status_headers[1], ''.join (body)
status, headers, body = call_application (app, {...environ...})
```


Chapter 12

Runtime

12.1 php.ini

PHP 可以基于内置的配置选项进行配置，也可以在运行时动态指定。

PHP 的配置文件 (`php.ini`) 在 PHP 启动时被读取，在 `php.ini` 中可以使用环境变量。

- 对于服务器模块版本的 PHP，仅在 Web 服务器启动时读取一次。

例如，对于 Apache 服务器模块版本的 PHP，仅在 Web 服务器启动时读取一次¹。

- 对于 CGI 和 CLI 版本，每次调用都会读取。

PHP 的配置文件 `php.ini` 包含 PHP 运行时所需要的相关参数，并且在重启 Web 服务器后才能使新的 `php.ini` 设置生效。

`php.ini` 使用 Windows 的 INI 文件的书写风格，使用 “;” 作为注释。

另外，如果某些选项允许有多个值，那么应该使用系统列表分隔符（在 Windows 下是分号 “;”，在 UNIX 下是冒号 “:”）。

`php.ini` 的搜索路径如下（按顺序）：

- SAPI 模块所指定的位置（Apache 2 中的 `PHPIniDir` 指令，CGI 和 CLI 中的 `-c` 命令行选项，NSAPI 中的 `php_ini` 参数，THTTPD 中的 `PHP_INI_PATH` 环境变量）。
- `PHPRC` 环境变量。
- 如果为不同版本的 PHP 指定了不同的 `php.ini` 文件位置，将以下面的顺序检查注册表目录：`[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y.z]`，`[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x.y]` 和 `[HKEY_LOCAL_MACHINE\SOFTWARE\PHP\x]`，其中的 `x`、`y` 和 `z` 指的是 PHP 主版本号，次版本号和发行批次。如果在其中任何目录下的 `IniFilePath` 有键值，则第一个值将被用作 `php.ini` 的位置（仅适用于 Windows）。
- `[HKEY_LOCAL_MACHINE\SOFTWARE\PHP]` 内 `IniFilePath` 的值（Windows 注册表位置）。

¹Apache 服务器在启动时会把目录转到根目录，这将导致 PHP 尝试在根目录下读取 `php.ini`。

- 当前工作目录 (对于 CLI)。
- Web 服务器目录 (对于 SAPI 模块) 或 PHP 所在目录 (Windows 下其它情况)。
- Windows 目录 (C:\Windows 或 C:\Winnt), 或 `--with-config-file-path` 编译时选项指定的位置。

12.2 php-SAPI.ini

如果存在 `php-SAPI.ini` (SAPI 是当前所用的 SAPI 名称, 因此实际文件名为 `php-cli.ini` 或 `php-apache.ini` 等), 则会用它替代 `php.ini`, SAPI 的名称可以用 `php_sapi_name()` 来测定。

由扩展库处理的 `php.ini` 指令的文档由扩展库提供, 但是并非所有的 PHP 指令都有文档说明。

```
; any text on a line after an unquoted semicolon (;) is ignored
[php] ; section markers (text within square brackets) are also ignored
; Boolean values can be set to either:
;   true, on, yes
; or false, off, no, none
register_globals = off
track_errors = yes

; you can enclose strings in double-quotes
include_path = "./usr/local/lib/php"

; backslashes are treated the same as any other character
include_path = ".;c:\php\lib"
```

12.3 .user.ini

- 自 PHP 5.1.0 起, 有可能在 `.ini` 文件内引用已存在的 `.ini` 变量, 例如 `open_basedir = ${open_basedir} ":/new/dir"`。
- 自 PHP 5.3.0 起, PHP 支持基于每个目录的 `.htaccess` 风格的 INI 文件, 不过仅被 CGI/-FastCGI SAPI 支持, 并且导致 PECL 的 `htscanner` 扩展作废。如果使用 Apache, 使用 `.htaccess` 文件有同样效果。

除了主 `php.ini` 之外, PHP 还会在每个目录下扫描 INI 文件, 从被执行的 PHP 文件所在目录开始一直上升到 Web 根目录 (`$_SERVER['DOCUMENT_ROOT']` 所指定的)。如果被执行的 PHP 文件在 Web 根目录之外, 则只扫描该目录。

在 `.user.ini` 风格的 INI 文件中, 只有具有 `PHP_INI_PERDIR` 和 `PHP_INI_USER` 模式的 INI 设置可被识别, 其中 `user_ini.filename` 和 `user_ini.cache_ttl` 控制着用户 INI 文件的

使用。

- `user_ini.filename` 设定了 PHP 会在每个目录下搜寻的文件名，如果设定为空字符串则 PHP 不会搜寻，默认值是 `.user.ini`。
- `user_ini.cache_ttl` 控制着重重新读取用户 INI 文件的间隔时间，默认是 300 秒（5 分钟）。

这些模式决定着一个 PHP 的指令在何时何地，是否能够被设定，而且每个指令都有其所属的模式。

- 有些指令可以在 PHP 脚本中用 `ini_set()` 来设定；
- 有些指令只能在 `php.ini` 或 `httpd.conf` 中进行设定。

例如，`output_buffering` 指令是属于 `PHP_INI_PERDIR`，因而就不能用 `ini_set()` 来设定，但是 `display_errors` 指令是属于 `PHP_INI_ALL`，因而可以在任何地方被设定，包括 `ini_set()`。

表 12.1: `PHP_INI_*` 模式的定义

模式	含义
<code>PHP_INI_USER</code>	可在用户脚本（例如 <code>ini_set()</code> ）或 Windows 注册表（自 PHP 5.3 起）以及 <code>.user.ini</code> 中设定
<code>PHP_INI_PERDIR</code>	可在 <code>php.ini</code> ， <code>.htaccess</code> 或 <code>httpd.conf</code> 中设定
<code>PHP_INI_SYSTEM</code>	可在 <code>php.ini</code> 或 <code>httpd.conf</code> 中设定
<code>PHP_INI_ALL</code>	可在任何地方设定

- PHP 运行于 Apache 模块方式

当使用 PHP 作为 Apache 模块时，也可以用 Apache 的配置文件（例如 `httpd.conf`）和 `htaccess` 文件中的指令来修改 PHP 的配置设定，需要有“`AllowOverride Options`”或“`AllowOverride All`”权限才可以。

有几个 Apache 指令可以使用户在 Apache 配置文件内部修改 PHP 的配置，至于哪些指令属于 `PHP_INI_ALL`，`PHP_INI_PERDIR` 或 `PHP_INI_SYSTEM` 中的哪一个，需要参考 `php.ini` 配置选项列表。

- `php_value name value`
设定指定的值。只能用于 `PHP_INI_ALL` 或 `PHP_INI_PERDIR` 类型的指令。要清除先前设定的值，把 `value` 设为 `none`。不要用 `php_value` 设定布尔值，应该用 `php_flag`。
- `php_flag name on|off`
用来设定布尔值的配置指令。仅能用于 `PHP_INI_ALL` 和 `PHP_INI_PERDIR` 类型的指令。
- `php_admin_value name value`

设定指定的指令的值。不能用于.htaccess 文件。任何用 `php_admin_value` 设定的指令都不能被.htaccess 或 `virtualhost` 中的指令覆盖。要清除先前设定的值，把 `value` 设为 `none`。

- `php_admin_flag name on|off`

用来设定布尔值的配置指令。不能用于.htaccess 文件。任何用 `php_admin_flag` 设定的指令都不能被.htaccess 或 `virtualhost` 中的指令覆盖。

```
<IfModule mod_php5.c>
  php_value include_path ".:usr/local/lib/php"
  php_admin_flag engine on
</IfModule>
<IfModule mod_php4.c>
  php_value include_path ".:usr/local/lib/php"
  php_admin_flag engine on
</IfModule>
```

PHP 常量不存在于 PHP 之外。例如在 `httpd.conf` 中不能使用 PHP 常量如 `E_ALL` 或 `E_NOTICE` 来设定 `error_reporting` 指令，因为其无意义，实际等于 0。应该用相应的掩码值来替代。这些常量可以在 `php.ini` 中使用。

- 通过 Windows 注册表修改 PHP 配置

在 Windows 下运行 PHP 时，可以用 Windows 注册表以目录为单位来修改配置。

PHP 配置值存放于注册表项 `HKLM\SOFTWARE\PHP\Per Directory Values` 下面，子项对应于路径名。例如，对于目录 `c:\inetpub\wwwroot` 的配置值会存放于 `HKLM\SOFTWARE\PHP\Per Directory Values\c\inetpub\wwwroot` 项下面。其中的设定对于任何位于此目录及其任何子目录的脚本都有效。项中的值的名称是 PHP 配置指令的名字，值的数据是字符串格式的指令值。值中的 PHP 常量不被解析。不过只有可修改范围是 `PHP_INI_USER` 的配置值可以用此方法设定，`PHP_INI_PERDIR` 的值就不行。

- 其它接口下的 PHP

无论怎样运行 PHP，都可以在脚本中通过 `ini_set()` 而在运行时修改某个值。如果对自己系统中的配置设定及其当前值的完整列表感兴趣，可以运行 `phpinfo()` 函数并查看其结果的页面。也可以在运行时用 `ini_get()` 或 `get_cfg_var()` 取得个别配置指令的值。

12.4 directive

当 Web 服务器使用 CGI 方式运行 PHP 时，可以开启 `cgi.force_redirect` 来增强系统安全性，但是在 IIS 和 `OmniHTTPD` 等服务器时必须关闭。

PHP 超全局变量的解析顺序可以通过 `variables_order` 进行设定，默认设定为 `EGPCS`，依

次为 `$_ENV`、`$_GET`、`$_POST`、`$_COOKIE` 和 `$_SERVER`。

基于安全的考虑，默认将 `register_globals` 设置为 `Off` 来禁止将 `$_ENV`、`$_GET`、`$_POST`、`$_COOKIE` 和 `$_SERVER` 变量同时设置为全局变量 `$GLOBALS` 的引用。

为了查找文件，可以使用 `include_path` 来指定一组目录以用于 `require` 或 `include`，以及 `fopen_with_path()` 函数。

```
## UNIX
include_path=".:usr/local/php/pear"
## Windows
include_path=".;c:/usr/local/php/pear"
```

PHP 的扩展库可以使用 `extension_dir` 进行指定，通常是 PHP 安装目录下的 `ext` 目录。

为了指定 PHP 启动时加载的扩展库，可以使用 `extension` 进行指定。例如：

```
# UNIX/Linux
extension=curl.so
extension=ftp.so
# Windows
extension=php_curl.dll
extension=php_ftp.dll
```


Chapter 13

Security

攻击者无时无刻不在准备对 Web 应用程序进行攻击，因此提高 Web 应用程序的安全性是非常有必要的。

13.1 Remote files

只要在 php.ini 文件中激活了 `allow_url_fopen` 选项，就可以在大多数需要用文件名作为参数的函数中使用 HTTP 和 FTP 的 URL 来代替文件名，同时也可以使用 `include`、`include_once`、`require` 及 `require_once` 语句中使用 URL。

例如，可以用以下示例的方式来打开远程 web 服务器上的文件，解析需要的输出数据，然后将这些数据用在数据库的检索中，或者简单地以和自己网站其它页面相同的风格输出其内容。

Example 18 获取远程文件的标题

```
<?php
$file = fopen ("http://www.example.com/", "r");
if (!$file) {
    echo "<p>Unable to open remote file.\n";
    exit;
}
while (!feof ($file)) {
    $line = fgets ($file, 1024);
    /* This only works if the title and its tags are on one line */
    if (eregi("<title>(.*?)</title>", $line, $out)) {
        $title = $out[1];
        break;
    }
}
```

```
}  
fclose($file);
```

如果有合法的访问权限，以一个用户的身份和某 FTP 服务器建立了链接，还可以向该 FTP 服务器端的文件进行写操作，不过仅可以使用该方法来创建新的文件，如果尝试覆盖已经存在的文件，`fopen()` 函数的调用将会失败。

要以“anonymous”以外的用户名连接服务器，需要指明用户名以及密码（例如 `ftp://user:password@ftp.example.com/path/to/file`），也可以在通过需要 Basic 认证的 HTTP 协议访问远程文件时使用相同的语法。

Example 19 上传数据到远程服务器

```
<?php  
$file = fopen ("ftp://ftp.example.com/incoming/outputfile", "w");  
if (!$file) {  
    echo "<p>Unable to open remote file for writing.\n";  
    exit;  
}  
/* Write the data here. */  
fwrite ($file, $_SERVER['HTTP_USER_AGENT'] . "\n");  
fclose ($file);
```

PHP 对远程文件的操作可以用来存储远程日志文件等，但是 `fopen()` 方式打开的 URL 中，仅能对新文件进行写操作。如果远程文件已经存在则 `fopen()` 函数的操作将会失败。

实际上，如果需要实现类似于分布式日志系统，需要使用 `syslog()` 函数等。

13.2 Authentication

PHP 的 HTTP 认证机制仅在 PHP 以 Apache 模块方式运行时才有效，因此该功能不适用于 CGI 版本。

在 Apache 模块的 PHP 脚本中，可以用 `header()` 函数来向客户端浏览器发送“Authentication Required”信息，使其弹出一个用户名/密码输入窗口。

当用户输入用户名和密码后，包含有 URL 的 PHP 脚本将会加上预定义变量 `PHP_AUTH_USER`，`PHP_AUTH_PW` 和 `AUTH_TYPE` 被再次调用，这三个变量分别被设定为用户名，密码和认证类型。预定义变量保存在 `$_SERVER` 数组中，支持“Basic”和“Digest”认证方法。

以下是在页面上强制客户端认证的脚本范例：

Example 20 Basic HTTP 认证

```

<?php
if(!isset($_SERVER['PHP_AUTH_USER'])){
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}else{
    echo '<p>Hello {$_SERVER['PHP_AUTH_USER']}</p>';
    echo '<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password';
}

```

Example 21 Digest HTTP 认证

```

<?php
$realm = 'Restricted area';

//user => password
$users = array('admin' => 'mypass', 'guest' => 'guest');

if (empty($_SERVER['PHP_AUTH_DIGEST'])) {
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Digest realm="'.$realm.
        '" qop="auth" nonce="'.uniqid().'" opaque="'.md5($realm).'"');

    die('Text to send if user hits Cancel button');
}

// analyze the PHP_AUTH_DIGEST variable
if (!(($data = http_digest_parse($_SERVER['PHP_AUTH_DIGEST'])) ||
    !isset($users[$data['username']]))
    die('Wrong Credentials!');

// generate the valid response
$A1 = md5($data['username'] . ':' . $realm . ':' . $users[$data['username']
    ]);
$A2 = md5($_SERVER['REQUEST_METHOD'] . ':' . $data['uri']);
$valid_response = md5($A1 . ':' . $data['nonce'] . ':' . $data['nc'] . ':' . $data['
    cnonce'] . ':' . $data['qop'] . ':' . $A2);

```

```

if ($data['response'] != $valid_response)
    die('Wrong Credentials!');

// ok, valid username & password
echo 'Your are logged in as: ' . $data['username'];

// function to parse the http auth header
function http_digest_parse($txt)
{
    // protect against missing data
    $needed_parts = array('nonce'=>1, 'nc'=>1, 'cnonce'=>1, 'qop'=>1, '
        username'=>1, 'uri'=>1, 'response'=>1);
    $data = array();

    preg_match_all('@(\w+)=(\[\"'\"]?)([a-zA-Z0-9=.\/_-]+\2@', $txt, $matches,
        PREG_SET_ORDER);

    foreach ($matches as $m) {
        $data[$m[1]] = $m[3];
        unset($needed_parts[$m[1]]);
    }

    return $needed_parts ? false : $data;
}

```

在编写 HTTP 标头代码时要格外小心。为了对所有的客户端保证兼容性，关键字“Basic”的第一个字母必须大写为“B”，分界字符串必须用双引号（不是单引号）引用，并且在标头行 HTTP/1.0 401 中，在 401 前必须有且仅有一个空格。

在实际运用中，可能需要对用户名和密码的合法性进行检查。或许进行数据库的查询，或许从 dbm 文件中检索。

有些 Internet Explorer 浏览器对标头的顺序要求严格，可以在发送 HTTP/1.0 401 之前首先发送 WWW-Authenticate 标头来可以解决该问题。

为了防止恶意用户通过编写脚本来从传统外部机制认证的页面上获取密码，当外部认证对特定页面有效且安全模式被开启时，PHP_AUTH 变量将不会被设置，不过 REMOTE_USER 可以被用来辨认外部认证的用户，因此可以用 \$_SERVER['REMOTE_USER'] 变量来获取用户名。

PHP 内部通过是否有 AuthType 指令来判断外部认证机制是否有效，不过仍然不能防止有人通过未认证的 URL 来从同一服务器上认证的 URL 上偷取密码。

Netscape Navigator 和 Internet Explorer 浏览器都会在收到 401 的服务端返回信息时清空所有的本地浏览器整个域的 Windows 认证缓存，从而可以有效地注销一个用户，并强制用户重新输入用户名和密码，因此可以使用这种方法来使登录状态“过期”，或者作为“注销”按钮的响应行为。

Example 22 强制重新输入用户名和密码的 HTTP 认证

```
<?php
function authenticate() {
    header('WWW-Authenticate: Basic realm="Test Authentication System"');
    header('HTTP/1.0 401 Unauthorized');
    echo "You must enter a valid login ID and password to access this
        resource\n";
    exit;
}

if (!isset($_SERVER['PHP_AUTH_USER']) ||
    ($_POST['SeenBefore'] == 1 && $_POST['OldAuth'] == $_SERVER['
        PHP_AUTH_USER']))
) {
    authenticate();
} else {
    echo "<p>Welcome: {$_SERVER['PHP_AUTH_USER']}<br />";
    echo "Old: {$_REQUEST['OldAuth']}";
    echo "<form action='{$_SERVER['PHP_SELF']}' METHOD='post'>\n";
    echo "<input type='hidden' name='SeenBefore' value='1' />\n";
    echo "<input type='hidden' name='OldAuth' value='{$_SERVER['PHP_AUTH_USER
       ']}' />\n";
    echo "<input type='submit' value='Re Authenticate' />\n";
    echo "</form></p>\n";
}
```

该行为对于 HTTP 的 Basic 认证标准来说并不是必须的，因此不能依靠这种方法。例如，对 Lynx 浏览器的测试表明 Lynx 在收到 401 的服务端返回信息时不会清空认证文件，因此只要对认证文件的检查要求没有变化，只要用户点击“后退”按钮，再点击“前进”按钮，其原有资源仍然能够被访问。不过，用户可以通过按“_”键来清空他们的认证信息。

HTTP 认证无法工作在 IIS 服务器的 CGI 模式下，需要编辑 IIS 的设置“目录安全”，点击“编辑”并且只选择“匿名访问”，其它所有的复选框都应该留空。

在 IIS 的 ISAPI 模式下使用 PHP 4 的时候，无法使用 PHP_AUTH_* 变量，只能使用 HTTP_AUTHORIZATION。

```
<?php
list($user, $pw) = explode(':', base64_decode(substr($_SERVER['HTTP_AUTHORIZATION'], 6))
);
```

如果要 HTTP 认证能够在 IIS 下工作，PHP 配置选项 `cgi.rfc2616_headers` 必须设置成 0 (默认值)。

如果安全模式被激活，脚本的 UID 会被加到 WWW-Authenticate 标头的 `realm` 部分。

```
Cache-Control:no-store, no-cache, must-revalidate
Connection:keep-alive
Content-Security-Policy:script-src 'self' 'nonce-pagejs' 'nonce-global' 'unsafe-inline'
'unsafe-eval'
Content-Type:text/html; charset=UTF-8
Date:Fri, 09 Dec 2016 09:26:43 GMT
Expires:Thu, 19 Nov 1981 08:52:00 GMT
Pragma:no-cache
Server:nginx/1.11.1
Strict-Transport-Security:max-age=31536000; includeSubdomains; preload
Transfer-Encoding:chunked
WWW-Authenticate:Basic realm='%E8%AF%B7%E8%BE%93%E5%85%A5%E8%B4%A6%E6%88%B7%E5%AF%86%E7%A0%81'
```

13.3 Password Hash

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。

`password_hash` 函数现在使用的是目前 PHP 所支持的最强大的加密算法 `BCrypt`，而且该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。

```
<?php
require 'password.php';
```

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

if($password_verify('bad-password',$passwordHash)) {
    // correct password
} else {
    // wrong password
}
```

13.4 Variable filter

永远不要信任外部输入，务必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。

外部输入的定义并不局限于用户通过表单提交的数据，其他诸如上传和下载的文档、`session` 值、`cookie` 数据，还有来自第三方 Web 服务的数据等，这些都是外部输入。

虽然外部输入可以被存储、组合并在以后继续使用，但是它依旧是外部输入。每次处理、输出、连接或在代码中包含时，都需要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，它可以在站点上执行 HTML 和 JavaScript 脚本（即跨站脚本攻击（XSS））。

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），不过可以使用自带的 `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。

13.5 Variable clean

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，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。当使用 PDO 中的限制参数功能时，它可以用来自动完成过滤的工作。

有些时候，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。

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。

13.6 Validity Verify

有效性验证是来确保外部输入的是应用程序实际所想要的内容。比如，用户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。

13.7 Configure file

在应用程序创建配置文件时，最好的选择时参照以下的做法：

- 推荐将配置信息存储在无法被直接读取和上传的位置上。
- 如果一定要存储配置文件在根目录下，那么使用.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。

13.8 Global Variable

register_globals 选项已经被移除并不再使用，同时也在提醒如果正在升级旧的应用程序的话，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，这样就很容易使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。如果使用的 PHP 版本低于 5.4.0，务必确保 register_globals 是被设置为 off。

13.9 Error Reporting

虽然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。

为了有效的保护应用程序不受到由此而引发的问题，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。

为了在开发环境中显示所有可能的错误，将 php.ini 进行如下配置：

```
display_errors = On
display_startup_errors = On
```

```
error_reporting = -1
log_errors = On
```

将 `error_reporting` 的值设为 `-1` 将会显示出所有的错误，也包括在未来的 PHP 版本中新增加的类型和参数，`-1` 和最初的 `E_ALL` 是相同的。

`E_STRICT` 类型的错误已经被包含在了 `E_ALL` 中，如果需要在 5.3 中显示所有的错误信息，就需要使用 `-1` 或者 `E_ALL | E_STRICT`。

下面的示例说明了如何在不同版本的 PHP 中开启全部错误：

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

为了在生产环境中隐藏错误显示，将 `php.ini` 进行如下配置：

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

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。

13.10 Upload Handling

PHP 支持使用 POST 和 PUT 方法来上传文本和二进制文件，并且提供了认证和文件操作函数来完全控制允许哪些用户上传以及文件上传后怎样处理。

PHP 能够接受任何来自符合 RFC-1867 标准的 HTTP 客户端（包括浏览器）上传的文件。

`php.ini` 的 `file_uploads`, `upload_max_filesize`, `upload_tmp_dir`, `post_max_size` 以及 `max_input_time` 设置选项。

Example 23 文件上传表单

```
<!-- The data encoding type, enctype, MUST be specified as below -->
<form enctype="multipart/form-data" action="__URL__" method="POST">
  <!-- MAX_FILE_SIZE must precede the file input field -->
  <input type="hidden" name="MAX_FILE_SIZE" value="30000" />
  <!-- Name of input element determines name in $_FILES array -->
  Send this file: <input name="userfile" type="file" />
```

```
<input type="submit" value="Send File" />
</form>
```

13.10.1 memory_limit

如果内存限制设置被激活,可能需要将 `memory_limit` 设置的更大,需要确认 `memory_limit` 的设置的是足够大。

13.10.2 post_max_size

如果 `post_max_size` 设置的值太小,则较大的文件会无法被上传,因此需要保证 `post_max_size` 的值足够大。

文件上传表单的属性必须是 `enctype="multipart/form-data"`, 否则文件无法上传。

PHP 的 `$_FILES` 全局变量包含了所有上传的文件信息, 以上范例产生的 `$_FILES` 数组的内容如下所示:

- `$_FILES['userfile']['name']` 客户端机器文件的原名称。
- `$_FILES['userfile']['type']` 文件的 MIME 类型 (如果浏览器提供该信息, 例如 “image/gif”), PHP 默认不检查 MIME 类型。
- `$_FILES['userfile']['size']` 已上传文件的大小 (单位为字节)。
- `$_FILES['userfile']['tmp_name']` 文件被上传后在服务端储存的临时文件名。
- `$_FILES['userfile']['error']` 和该文件上传相关的错误代码。
 1. `UPLOAD_ERR_OK` 值为 0 表示没有错误发生, 文件上传成功。
 2. `UPLOAD_ERR_INI_SIZE` 值为 1 表示上传的文件超过了 `php.ini` 中 `upload_max_filesize` 选项限制的值。
 3. `UPLOAD_ERR_FORM_SIZE` 值为 2 表示上传文件的大小超过了 HTML 表单中 `MAX_FILE_SIZE` 选项指定的值。
 4. `UPLOAD_ERR_PARTIAL` 值为 3 表示文件只有部分被上传。
 5. `UPLOAD_ERR_NO_FILE` 值为 4 表示没有文件被上传。
 6. `UPLOAD_ERR_NO_TMP_DIR` 值为 6 表示找不到临时文件夹。
 7. `UPLOAD_ERR_CANT_WRITE` 值为 7 表示文件写入失败。

13.10.3 upload_tmp_dir

上传的文件默认会被储存到服务端的默认临时目录中, 除非 `php.ini` 中的 `upload_tmp_dir` 设置为其它的路径。

服务端的默认临时目录可以通过更改 PHP 运行环境的环境变量 `TMPDIR` 来重新设置, 但是在 PHP 脚本内部通过运行 `putenv()` 函数来设置是不起作用的。该环境变量也可以用来确认其它的操作是否也是在上传的文件上进行的。

Example 24 完成文件上传

```
<?php
$uploadaddir = '/var/www/uploads/';
$uploadfile = $uploadaddir . basename($_FILES['userfile']['name']);

echo '<pre>';
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile)) {
    echo "File is valid, and was successfully uploaded.\n";
} else {
    echo "Possible file upload attack!\n";
}

echo 'Here is some more debugging info: ';
print_r($_FILES);

print "</pre>";
```

文件路径的表示方法有很多种，PHP 无法确保使用生僻外语的文件名（尤其是包含空格的）能够被正确的处理。

为了决定接下来要对该文件进行哪些操作，接受上传文件的 PHP 脚本应该实现任何逻辑上必要的检查，否则不对正在操作的文件进行验证就意味着用户能够访问其它目录下的敏感信息。

- `$_FILES['userfile']['size']` 变量可以排除过大或过小的文件；
- `$_FILES['userfile']['type']` 变量可以排除文件类型和某种标准不相符合的文件；
- `$_FILES['userfile']['error']` 变量可以根据不同的错误代码来计划下一步如何处理。

不过只能把这些当作一系列检查中的第一步，因为有些值完全由客户端控制而在 PHP 端并不检查，PHP 脚本要么将上传的文件从临时目录中删除，要么将其移动到其它的地方（例如上传到云存储）。

如果上传的文件没有被移动到其它地方也没有被改名，则该文件将在表单请求结束时被删除。

如果表单中没有选择上传的文件，则 `$_FILES['userfile']['size']` 的值将为 0，`$_FILES['userfile']['tmp_name']` 将为空。

13.10.4 max_input_time

`max_input_time` 以秒为单位设定了脚本接收输入（例如文件上传）的最大时间。对于较大或多个文件或者用户的网速较慢时，可能会超过默认的 60 秒。

13.10.5 upload_max_filesize

MAX_FILE_SIZE 隐藏字段（单位为字节）必须放在文件输入字段之前，其值为接收文件的最大尺寸。这是对浏览器的一个建议，而且 PHP 也会检查此项，不过用户可以在 HTTP 客户端端简单绕过此设置，因此不要指望用此特性来阻挡大文件。

虽然 PHP 设置中的上传文件最大值不会失效，但是最好还是在表单中加上 MAX_FILE_SIZE 项目，这样可以避免用户在等待上传大文件上传之后才发现文件过大上传失败的错误。

MAX_FILE_SIZE 值不能大于 ini 设置中 upload_max_filesize 选项设置的值（默认为 2M 字节）。

13.10.6 max_execution_time

max_execution_time 设置的值太小时，如果脚本运行的时间超过该设置就会自动退出，因此需要保证 max_execution_time 设置的足够大。

max_execution_time 仅仅只影响脚本本身运行的时间，任何其它花费在脚本运行之外的时间（例如 system() 函数对系统的调用、sleep() 函数的使用、数据库查询、文件上传等），在计算脚本运行的最大时间时都不包括在内。

httpd 可能会丢弃从客户端获得的 content-type mime 头信息中第一个空格后所有的内容，因此 httpd 可能不支持文件上传特性。

13.11 Batch Uploading

PHP 的 HTML 数组特性支持文件类型，因此用户可以批量上传文件，例如：

Example 25 批量文件上传

```
<form enctype="multipart/form-data" action="__URL__" method="post">
<p>Pictures:
<input type="file" name="pictures[]" />
<input type="file" name="pictures[]" />
<input type="file" name="pictures[]" />
<input type="submit" value="Send" />
</p>
</form>
```

服务器端处理批量文件上传的逻辑如下：

```
<?php
foreach ($_FILES["pictures"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {
```



```

    $tmp_name = $_FILES["pictures"]["tmp_name"][$key];
    $name = $_FILES["pictures"]["name"][$key];
    move_uploaded_file($tmp_name, "data/$name");
}
}

```

PHP 支持同时上传多个文件并将它们的信息自动以数组的形式组织，因此可以对 input 域使用不同的 name 来上传多个文件。

为了实现批量上传，需要在 HTML 表单中对文件上传域使用和多选框与复选框相同的数组式提交语法，注意不应该将普通的输入字段和文件上传的字段混用同一个表单变量（例如都用 foo[]）。

Example 26 多文件批量上传

```

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

```

在以上表单被提交后，数组 \$_FILES['userfile'], \$_FILES['userfile']['name'] 和 \$_FILES['userfile']['size'] 将被初始化，所有这些提交的信息都将被储存到以数字为索引的数组中。

例如，假设名为 /home/me/review.html 和 /home/me/xwp.out 的文件被提交，那么 \$_FILES['userfile']['name'] 的值将是 review.html, \$_FILES['userfile']['name'][1] 的值将是 xwp.out。类似的，\$_FILES['userfile']['size'] 将包含文件 review.html 的大小，依此类推。

此外，也同时设置了 \$_FILES['userfile']['name'][0], \$_FILES['userfile']['tmp_name'][0], \$_FILES['userfile']['size'][0] 以及 \$_FILES['userfile']['type'][0]。

13.12 Request Method

13.12.1 GET

13.12.2 POST

13.12.3 PUT

PHP4 必须使用标准的输入流来读取一个 HTTP PUT 的内容。

Example 27 PHP4 使用 PUT 保存文件

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://stdin", "r");

/* Open a file for writing */
$fp = fopen("myputfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
    fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
```

PUT 请求比文件上传要简单的多，一般的形式为：

```
PUT /path/filename.text HTTP/1.1
```

远程客户端会将其中的 `/path/filename.text` 存储到 web 目录树，让 Apache 或者 PHP 自动允许所有人覆盖 web 目录树下的任何文件显然是很不明智的，因此要处理类似 PUT 的请求时必须先告诉 web 服务器需要用特定的 PHP 脚本来处理该请求。

在 Apache 下，可以用 Script 选项来设置，通常在 `<Directory>` 区域或者 `<Virtualhost>` 区域进行设置。

```
Script PUT /put.php
```

上述设置指示 Apache 将所有对 URI 的 PUT 请求全部发送到 `put.php` 脚本，这些 URI 必须和 PUT 命令中的内容相匹配。

在 `put.php` 文件中，可以作如下操作：

```
<?php
copy($PHP_UPLOADED_FILE_NAME, $DOCUMENT_ROOT . $REQUEST_URI);
```

把文件拷贝到远程客户端请求的位置时，可能希望在文件拷贝之前进行一些检查或者对用户认证之类的操作。

这里唯一的问题是，当 PHP 接受到 PUT 方法的请求时，它将会把上传的文件储存到和其它用 POST 方法处理过的文件相同的临时目录。在请求结束时，临时文件将被删除，因此用来处理 PUT 的 PHP 脚本必须将该文件拷贝到其它的地方。

PUT 上传的临时文件的文件名被储存在变量 `$PHP_PUT_FILENAME` 中，也可以通过 `$REQUEST_URI` 变量获得建议的目标文件名（在非 Apache web 服务器上可能会有较大的变

化)。该目标文件名是由远程客户端指定的，也可以不遵循客户端提供的信息，把所有上传的文件存储到一个特殊的上传目录下。

13.13 Connection handling

在 PHP 内部，系统维护着连接状态，有三种可能的状态：

- 0 - NORMAL (正常)
- 1 - ABORTED (异常退出)
- 2 - TIMEOUT (超时)

当 PHP 脚本正常地运行 NORMAL 状态时，连接为有效。当远程客户端中断连接时，ABORTED 状态的标记将会被打开。远程客户端连接的中断通常是由用户点击 STOP 按钮导致的。当连接时间超过 PHP 的时限，TIMEOUT 状态的标记将被打开。

用户可以决定脚本是否需要在客户端中断连接时退出，不过有时候让脚本完整地运行会带来很多方便，即使没有远程浏览器接受脚本的输出。

默认的情况是当远程客户端连接中断时脚本将会退出，该处理过程可由 `php.ini` 的 `ignore_user_abort` 或由 `httpd.conf` 设置中对应的“`php_value ignore_user_abort`”以及 `ignore_user_abort()` 函数来控制。

如果没有告诉 PHP 忽略用户的中断，脚本将会被中断，除非通过 `register_shutdown_function()` 设置了关闭触发函数。

通过该关闭触发函数，当远程用户点击 STOP 按钮后，脚本再次尝试输出数据时，PHP 将会检测到连接已被中断，并调用关闭触发函数。

脚本也有可能被内置的脚本计时器中断，默认的超时限制为 30 秒。同样地，这个值可以通过设置 `php.ini` 的 `max_execution_time` 或 `httpd.conf` 设置中对应的“`php_value max_execution_time`”参数或者 `set_time_limit()` 函数来更改。当计数器超时的时候，脚本将会类似于以上连接中断的情况退出，先前被注册过的关闭触发函数也将在这时被执行。

在关闭触发函数中，可以通过调用 `connection_status()` 函数来检查超时是否导致关闭触发函数被调用。如果超时导致了关闭触发函数的调用，该函数将返回 2。

需要注意的一点是，ABORTED 和 TIMEOUT 状态可以同时有效，这样的设置在告诉 PHP 忽略用户的退出操作时是可能的，从而让 PHP 仍然注意用户已经中断了连接但脚本仍然在运行的情况。如果到了运行的时间限制，脚本将被退出，设置过的关闭触发函数也将被执行。在这时会发现函数 `connection_status()` 返回 3。

13.14 Persistent connections

持久的数据库连接是指在脚本结束运行时不关闭的连接。当收到一个持久连接的请求时，PHP 将检查是否已经存在一个（前面已经开启的）相同的持久连接。

- 如果存在，将直接使用这个连接；
- 如果不存在，则建立一个新的连接。

这里，所谓“相同”的连接是指用相同的用户名和密码到相同主机的连接。

没有完全理解 Web 服务器的工作原理和分布式负载的用户可能会错误地理解持久连接的作用，持久连接没有在相同的连接上提供建立“用户会话”的能力，也不提供有效建立事务的能力。

实际上，从严格意义上来讲，持久连接不会提供任何非持久连接无法提供的特殊功能，原因和 Web 服务器工作的方式有关。

Web 服务器可以用三种方法来利用 PHP 生成 web 页面。

第一种方法是將 PHP 用作一个单独运行的语言解释器 (CGI Wrapper)。以这种方法运行，PHP 会为向 web 服务器提出的每个 PHP 页面请求生成并结束一个 PHP 解释器线程。由于该线程会随每个请求的结束而结束，因此任何在这个线程中利用的任何资源（例如指向 SQL 数据库服务器的连接）都会随线程的结束而关闭。在这种情况下，使用持久连接和非持久连接没有任何区别——因为 PHP 脚本本身的执行不是持久的。

第二种方法是把 PHP 用作多进程 web 服务器的一个模块（只适用于 Apache）。对于一个多进程的服务器，其典型特征是有一个父进程和一组子进程协调运行，其中实际生成 web 页面的是子进程。每当客户端向父进程提出请求时，该请求会被传递给还没有被其它的客户端请求占用的子进程。这也就是说当相同的客户端第二次向服务端提出请求时，它将被一个不同的子进程来处理。在开启了一个持久连接后，所有请求 SQL 服务的后继页面都能够重用这个已经建立的 SQL Server 连接。

第三种方法是將 PHP 用作多线程 web 服务器的一个插件，PHP 对 ISAPI、WSAPI 和 NSAPI 的支持使得 PHP 可以被用作多线程 Web 服务器的插件，持久连接的行为和前面所描述的多过程模型在本质上是相同的。

持久连接所能提供的好处仅仅是效率，当 Web Server 创建到 SQL 服务器的连接耗费 (Overhead) 较高（如耗时较久，消耗临时内存较多）时，持久连接将更加高效。Overhead 高低取决于很多因素（例如数据库的种类，数据库服务和 web 服务是否在同一台服务器上以及 SQL 服务器负载状况等）。当 Overhead 较高，每次创建数据库连接成本较高时，持久连接将显著地提高效率。

持久连接使得每个子进程在其生命周期中只做一次连接操作，而非每次在处理一个页面时都要向 SQL 服务器提出连接请求。这也就是说，每个子进程将对服务器建立各自独立的持久连接。例如，如果有 20 个不同的子进程运行某脚本建立了持久的 SQL 服务器持久连接，那么实际上向该 SQL 服务器建立了 20 个不同的持久连接，每个进程占有一个。

如果持久连接的子进程数目超过了设定的数据库连接数限制，系统将会产生一些问题。如果数据库的同时连接数限制为 16，而在繁忙会话的情况下，有 17 个线程试图连接，那么有一个线程将无法连接。如果这个时候，在脚本中出现了使得连接无法关闭的错误（例如无限循环），则该数据库的 16 个连接将迅速地受到影响，因此需要即时处理已放弃的以及闲置

的连接。

在使用持久连接时还有一些特别的问题需要注意。例如，

在持久连接中使用数据表锁时，如果脚本不管什么原因无法释放该数据表锁，其随后使用相同连接的脚本将会被持久的阻塞，这种情况下需要重新启动 `httpd` 服务或者数据库服务。

在持久连接中使用事务处理时，如果脚本在事务阻塞产生前结束，则该阻塞也会影响到使用相同连接的下一个脚本，这种情况下同样需要重新启动 `httpd` 服务或者数据库服务。

不管在什么情况下，都可以通过使用 `register_shutdown_function()` 函数来注册一个简单的清理函数来打开数据表锁，或者回滚事务，实际上更好的处理方法是不在使用数据表锁或者事务处理的脚本中使用持久连接，这样就可以从根本上解决这个问题（在其它地方仍然可以使用持久连接）。

持久连接与常规的非持久连接应该是可以互相替换的，即将持久连接替换为非持久连接时，脚本的行为也不应该发生改变，持久连接只应该改变脚本的效率而非其行为。

13.15 Thread Safe

Windows 版的 PHP 从版本 5.2.1 开始有 Thread Safe（线程安全）和 None Thread Safe（NTS，非线程安全）之分，这两者不同在于何处[?]？到底应该用哪种？

从 2000 年 10 月 20 日发布的第一个 Windows 版的 PHP3.0.17 开始的都是线程安全的版本。

与 Linux/Unix 系统是采用多进程的工作方式不同的是，Windows 系统的是多线程的工作方式，因此在 IIS 下以 CGI 方式运行 PHP 就会非常慢，原因就是 CGI 模式是建立在多进程的基础上的，而非多线程。

一般情况下，在 Windows 平台上需要把 PHP 配置成以 ISAPI 的方式来运行，ISAPI 是多线程的方式，可以加快运行速度。

不过，仍然存在一个问题，那就是很多常用的 PHP 扩展是以 Linux/Unix 的多进程思想来开发的，这些扩展在 ISAPI 的方式运行时就会出错并拖垮 IIS，因此在 IIS 下 CGI 模式才是 PHP 运行的最安全方式。

CGI 模式的不足就是对于每个 HTTP 请求都需要重新加载和卸载整个 PHP 环境，其资源消耗是巨大的。

为了兼顾 IIS 下 PHP 的效率和安全性，微软给出了 FastCGI 的解决方案来让 PHP 的进程重复利用而不是每一个新的请求就新建一个进程，同时 FastCGI 也可以允许几个进程同时执行，从而既解决了 CGI 进程模式消耗太大的问题，又利用了 CGI 进程模式不存在线程安全问题的优势。

综合来说，如果使用 ISAPI 的方式来运行 PHP 就必须用 Thread Safe（线程安全）的版本，FastCGI 模式运行 PHP 没有必要进行线程安全检查，None Thread Safe（非线程安全）的

版本能够更好的提高效率。

- 在 Windows 版本的 Apache 下运行 PHP 时需要 VC6 版本的 PHP;
- 在 IIS 下运行 PHP 则需要 VC9 版本的 PHP。

13.16 Safe Mode

PHP 的安全模式是为了试图解决共享服务器 (shared-server) 安全问题而设立的, 现在已经废弃 (强制开启会报告 E_CODE_ERROR 级别的错误)。

虽然在结构上试图从 PHP 层上解决这个问题是不合理的, 但是修改 web 服务器层和操作系统层显得非常不现实, 因此 ISP 大量地使用安全模式。

表 13.1: 保安措施和安全模式配置指令

名字	默认	可修改范围	说明
safe_mode	"0"	PHP_INI_SYSTEM	已废弃
safe_mode_gid	"0"	PHP_INI_SYSTEM	已废弃
safe_mode_include_dir	"/usr/lib"	PHP_INI_SYSTEM	已废弃
safe_mode_exec_dir	"/usr/bin"	PHP_INI_SYSTEM	已废弃
safe_mode_allow_localhost_ips	"0"	PHP_INI_SYSTEM	已废弃
safe_mode_protected_path	"/usr/lib/php"	PHP_INI_SYSTEM	已废弃

PHP 默认不会运行在安全模式下, PHP 安全模式设置的限制不适用于可执行文件。

13.16.1 safe_mode

php.ini 中的 safe_mode 设置是否启用安全模式。

13.16.2 safe_mode_gid

php.ini 中的 safe_mode_gid 配置是否把 UID 比较检查放宽到 GID 比较检查。

默认情况下, 安全模式在打开文件时会进行 UID 比较检查, 打开 safe_mode_gid 可以放宽到 GID 比较。是否在文件访问时使用 UID (FALSE) 或者 GID (TRUE) 来做检查。

13.16.3 safe_mode_exec_dir

在安全模式下, 可以通过 safe_mode_exec_dir 设置具有系统程序执行功能的目录, 而且 exec()、system() 或 popen() 等函数只能调用执行该目录下的系统程序。

如果 PHP 使用了安全模式，`system()` 和其它程序执行函数将拒绝启动不在此目录中的程序，必须使用 `/` 作为目录分隔符（包括 Windows 中）。

13.16.4 `safe_mode_include_dir`

在安全模式下，`safe_mode_include_dir` 可以设置严格限制进行文件包含的目录，如果一个 PHP 程序使用 `include` 或 `require` 包含该目录（及子目录）中的另一个文件时，必须保证彼此具有相同的 UID 或 GID。

当从此目录及其子目录（目录必须在 `include_path` 中或者用完整路径来包含）包含文件时越过 UID/GID 检查，而且 `safe_mode_include_dir` 指令可以接受和 `include_path` 指令类似的风格用冒号（Windows 中是分号）隔开的路径，而不只是一个目录。

指定的限制实际上是一个前缀，而非一个目录名。也就是说，“`safe_mode_include_dir = /dir/incl`”将允许访问“`/dir/include`”和“`/dir/incls`”，如果它们存在的话。如果希望将访问控制在一个指定的目录，那么在结尾加上一个斜线（例如“`safe_mode_include_dir = /dir/incl/`”。如果本指令的值为空，具有不同 UID/GID 的文件将不能被包含。

13.16.5 `safe_mode_allowed_env_vars`

设置某些环境变量可能是潜在的安全缺口。本指令包含有一个逗号分隔的前缀列表。

在安全模式下，用户只能改变那些名字具有在这里提供的前缀的环境变量。默认情况下，用户只能设置以 `PHP_` 开头的环境变量（例如 `PHP_FOO = BAR`）。

如果本指令为空，PHP 将使用户可以修改任何环境变量

13.16.6 `safe_mode_protected_env_vars`

`safe_mode_protected_env_vars` 指令包含有一个逗号分隔的环境变量的列表，最终用户不能用 `putenv()` 来改变这些环境变量，在 `safe_mode_allowed_env_vars` 中设置了允许修改时也不能改变这些变量。

如果 `safe_mode` 设置为 `on`，PHP 将通过文件函数或其目录检查当前脚本的拥有者是否和将被操作的文件的拥有者相匹配。

```
-rw-rw-r-- 1 rasmus rasmus    33 Jul 1 19:20 script.php
-rw-r--r-- 1 root   root      1116 May 26 18:01 /etc/passwd
```

如果安全模式被激活，则执行 `readfile('/etc/passwd');` 将会导致以下错误：

```
Warning: SAFE MODE Restriction in effect. The script whose uid is 500 is not
allowed to access /etc/passwd owned by uid 0 in /docroot/script.php on line 2
```

在某些环境下，宽松的 GID 检查已经足够，严格的 UID 检查反而是不适合的，因此可以用 `safe_mode_gid` 选项来控制这种检查。如果设置为 On 则进行宽松的 GID 检查，设置为 Off（默认值）则进行 UID 检查。

13.16.7 disable_functions

PHP 安全模式允许单独屏蔽某些函数，但是 `disable_functions` 选项不能在 `php.ini` 文件外部使用，也就是说无法在 `httpd.conf` 文件的按不同虚拟主机或不同目录的方式来屏蔽函数。例如，如果将如下内容加入到 `php.ini` 文件：

```
disable_functions readfile,system
```

如果在设置了 `disable_functions` 选项后运行同样的 `script.php`，则其结果如下：

```
Warning: readfile() has been disabled for security reasons in
/docroot/script.php on line 2
```

13.16.8 open_basedir

除了 `safe_mode` 以外，如果设置了 `open_basedir` 选项，则所有的文件操作将被限制在指定的目录下。例如：

```
<Directory /docroot>
  php_admin_value open_basedir /docroot
</Directory>
```

如果在设置了 `open_basedir` 选项后运行同样的 `script.php`，则其结果如下：

```
Warning: open_basedir restriction in effect. File is in wrong directory in
/docroot/script.php on line 2
```

无论是否打开安全模式，`open_basedir` 都可以设置能够进行的文件操作的目录或文件的前缀，可以使用系统分隔符（“;”或“:”）添加多个列表。

当一个脚本试图使用 `fopen()` 或 `gzopen()` 打开一个文件时，那么该文家的位置将被检查，而且这个限制是前缀，不是目录名。

如果基于安全原因需要禁用某些函数/类时，可以使用 `disable_functions/disable_classes`，并且接受逗号分隔的函数/类名列表作为参数，不受安全模式的影响。