第一章 准备工作和背景知识

**千里之行，始于足下。**

在开始进入PHP的内核实现之前，需要做一些准备工作，也需要了解一些背景知识。本章主要涉及PHP源码的获取， PHP源码的编译，从而得到我们的调试环境。

接下来，我们将简单描述PHP源码的结构以及在\*nix环境和Windows环境下如何阅读源码。 最后我们介绍在阅读PHP源码过程中经常会遇到的一些语句。

如果你没有接触过PHP，或者对PHP的历史不太了解，我们推荐你先移步[百度百科 PHP](http://baike.baidu.com/view/99.htm)， 这里有PHP非常详细的历史介绍，它包括PHP的诞生，PHP的发展，PHP的应用，PHP现有三大版本的介绍以及对于PHP6的展望等。

目前PHP6已经停止开发了，PHP6的设计初衷是向后不兼容以及Unicode支持等。目前 很多特性已经在PHP5.3和PHP5.4中实现了:比如5.4中的traits，支持C#类似的getter&setter语法（目前处在实现阶段）， 基本类型的类型提示等。

me：当前PHP开发组已经略过PHP 6，直接发了PHP 7.

下面，我们将介绍源码阅读环境的搭建。

# 第一节 环境搭建

在开始学习PHP实现之前，我们需要一个实验和学习的环境。下面介绍一下怎样在\*nix环境下准备和搭建PHP环境。

(\*nix指的是类Unix环境，比如各种Linux发行版，FreeBSD， OpenSolaris， Mac OS X等操作系统)

## 1.获取PHP源码

为了学习PHP的实现，首先需要下载PHP的源代码。下载源码首选是去[PHP官方网站http://php.net/downloads.php](http://php.net/downloads.php)下载， 如果你喜欢使用svn/git等版本控制软件，也可以使用svn/git来获取最新的源代码。

# git 官方地址

git clone https://git.php.net/repository/php-src.git

# 也可以访问github官方镜像

git clone git://github.com/php/php-src.git

cd php-src && git checkout PHP-5.3 # 签出5.3分支

笔者比较喜欢用版本控制软件签出代码，这样做的好处是能看到PHP每次修改的内容及日志信息， 如果自己修改了其中的某些内容也能快速的查看到，如果你想修复PHP的某个Bug或者提交新功能的话， 有版本控制也会容易的多，更多信息可以参考附录：[怎样为PHP做贡献](http://tina.reeze.cn/book/?p=D-how-to-contribute)。

目前PHP已经[迁移到Git](http://www.php.net/archive/2012.php#id2012-03-20-1)了，PHP的wiki上有关于 [迁移到Git的说明](https://wiki.php.net/vcs/gitfaq)，以及[使用Git的流程](https://wiki.php.net/vcs/gitfaq)   
~~在笔者编写这些内容的时候PHP版本控制是还基于SVN的，上面提到的github镜像地址目前已经没有同步更新了， 由于把svn同步到git会对系统性能造成明显影响，加上社区还没有就到底是否迁移到git达成一致，所以也就停止了更新。 目前很多开源软件都开始转向了分布式版本控制系统(~~[~~DVCS~~](http://en.wikipedia.org/wiki/Distributed_revision_control)~~)， 例如Python语言在转向DVCS时对目前的分布式版本控制系统做了一个~~[~~详细的对比~~](http://www.python.org/dev/peps/pep-0374/)~~， 如果以前没有接触过，笔者强烈建议试试这些版本控制软件。~~现在Github的同步是实时的。 所以习惯Github的话，基本上可以把Github当做官方版本库了。

## 2.准备编译环境

在\*nix环境下，需要安装编译构建环境。如果你用的是Ubuntu或者是用apt做为包管理的系统，可以通过如下命令快速安装：

sudo apt-get install build-essential

如果你使用的是Mac OS X，则需要安装Xcode。Xcode可以在Mac OS X的安装盘中找到，如果你有Apple ID的话， 也可以登陆苹果开发者网站<http://developer.apple.com/>下载。

如果你不愿意下载庞大的Xcode，也可以去<https://github.com/kennethreitz/osx-gcc-installer>下载安装包， 只安装所需的命令行工具。

## 3. 编译

下一步可以开始编译了，本文只简单介绍基本的编译过程，不包含Apache的PHP支持以及Mysql等模块的编译。 相关资料请自行查阅相关文档。 如果你是从svn/git签出的代码则需要执行代码根目录的buildconf脚本（me:编译配置脚本生成文件）以生成所需要的构建脚本。

cd ~/php-src

./buildconf

执行完以后就可以开始configure了，configure有很多的参数，比如指定安装目录，是否开启相关模块等选项：

有的系统自带的autoconf程序版本会有Bug，可能导致扩展的配置无法更新，如果在执行./buildconf时 报错，可以根据出错信息安装合适版本的autoconf工具。

./configure --help # 查看可用参数

为了尽快得到可以测试的环境，我们仅编译一个最精简的PHP。通过执行 ./configure --disable-all来进行配置。 以后如果需要其他功能可以重新编译。如果configure命令出现错误，可能是缺少PHP所依赖的库，各个系统的环境可能不一样。 出现错误可根据出错信息上网搜索。 直到完成configure。configure完成后我们就可以开始编译了。

./configure --disable-all

make

在\*nix下编译过程序的读者应该都熟悉经典的configure make，make install吧。执行make之后是否需要make install就取决于你了。 如果install的话最好在configure的时候是用prefix参数指定安装目录， 不建议安装到系统目录， 避免和系统原有的PHP版本冲突。 在make 完以后，在sapi/cli目录里就已经有了php的可以执行文件. 执行一下命令：

./sapi/cli/php -v

-v参数表示输出版本号，如果命令执行完后看到输出php版本信息则说明编译成功。 如果是make install的话可以执行$prefix/bin/php这个路径的php。 当然如果是安装在系统目录或者你的prefix目录在$PATH环境变量里的话，直接执行php就行了。

在只进行make而不make install时，只是编译为可执行二进制文件，所以在终端下执行的php-cli所在路径就是php-src/sapi/cli/php。

后续的学习中可能会需要重复configure make 或者 make && make install 这几个步骤。

## Windows环境的编译

Windows环境下的编译可以参考官方Wiki的说明：<https://wiki.php.net/internals/windows/stepbystepbuild>

## 推荐书籍和参考

* [Gnu Make中文书册](http://www.yayu.org/book/gnu_make/index.html)
* 《Autotools A Practioner's Guide》

# 第二节 源码结构、阅读代码方法

## PHP源码目录结构

俗话讲：重剑无锋，大巧不工。PHP的源码在结构上非常清晰。下面先简单介绍一下PHP源码的目录结构。

* **根目录: /** 这个目录包含的东西比较多，主要包含一些说明文件以及设计方案。 其实项目中的这些README文件是非常值得阅读的例如：
  + /README.PHP4-TO-PHP5-THIN-CHANGES 这个文件就详细列举了PHP4和PHP5的一些差异。
  + 还有一个比较重要的文件/CODING\_STANDARDS（me：标准规范），如果要想写PHP扩展的话，这个文件一定要阅读一下， 不管你个人的代码风格是什么样，怎么样使用缩进和花括号，既然来到了这样一个团体里就应该去适应这样的规范，这样在阅读代码或者别人阅读你的 代码是都会更轻松。
* **build** 顾名思义，这里主要放置一些和源码编译相关的一些文件，比如开始构建之前的buildconf脚本等文件，还有一些检查环境的脚本等。
* **ext** 官方扩展目录，包括了绝大多数PHP的函数的定义和实现，如array系列，pdo系列，spl系列等函数的实现，都在这个目录中。个人写的扩展在测试时也可以放到这个目录，方便测试和调试。
* **main** 这里存放的就是PHP最为核心的文件了，主要实现PHP的基本设施，这里和Zend引擎不一样，Zend引擎主要实现语言最核心的语言运行环境（me:如buildcheck.sh等）。
* **Zend** Zend引擎的实现目录，比如脚本的词法语法解析，opcode的执行以及扩展机制的实现等等。
* **pear** “PHP 扩展与应用仓库(me:PHP extension and application repository)”，包含PEAR的核心文件。
* **sapi** 包含了各种服务器抽象层的代码(me:service api)，例如apache的mod\_php，cgi，fastcgi以及fpm等等接口。
* **TSRM** PHP的线程安全是构建在TSRM库之上的，PHP实现中常见的\*G宏通常是对TSRM的封装，TSRM(Thread Safe Resource Manager)线程安全资源管理器。
* **tests** PHP的测试脚本集合，包含PHP各项功能的测试文件
* **win32** 这个目录主要包括Windows平台相关的一些实现，比如sokcet的实现在Windows下和\*Nix平台就不太一样，同时也包括了Windows下编译PHP相关的脚本。

PHP的测试比较有意思，它使用PHP来测试PHP，测试php脚本在/run-tests.php，这个脚本读取tests目录中phpt文件。 读者可以打开这些看看，php定义了一套简单的规则来测试，例如以下的这个测试脚本/tests/basic/001.phpt：

--TEST--

Trivial "Hello World" test

--FILE--

<?php [echo](http://www.php.net/echo) "Hello World"?>

--EXPECT--

Hello World

这段测试脚本很容易看懂，执行--FILE--下面的PHP文件，如果最终的输出是--EXPECT--所期望的结果则表示这个测试通过， 可能会有读者会想，如果测试的脚本不小心触发Fatal Error，或者抛出未被捕获的异常了，因为如果在同一个进程中执行， 测试就会停止，后面的测试也将无法执行，php中有很多将脚本隔离的方法比如： system()，exec()等函数，这样可以使用主测试进程服务调度被测脚本和检测测试结果，通过这些外部调用执行测试。 php测试使用了[proc\_open()函数](http://www.php.net/manual/en/function.proc-open.php)， 这样就可以保证测试脚本和被测试脚本之间能隔离开。phpt文件的编写详细信息可参考 附录[E phpt文件的编写](http://tina.reeze.cn/book/?p=E-phpt-file)。 如果你真的那么感兴趣，那么研究下$PHP\_SRC/run-tests.php脚本的实现也是不错的选择。这个测试框架刚开始 由PHP的发明者Rasmus Lerdorf编写，后来进行了很多的改进。后面可能会引入[并行测试](http://marc.info/?l=php-internals&m=133609491526265)的支持。

## PHP源码阅读工具

使用VIM + Ctags

通常在Linux或其他\*Nix环境我们都使用[VIM](http://www.vim.org/)作为代码编辑工具，在纯命令终端下，它几乎是无可替代的。 它具有非常强大的扩展机制，在文字编辑方面基本上无所不能。 不过Emacs用户请不要激动，笔者还没有真正使用Emacs，虽然我知道它甚至可以[煮咖啡](http://people.ku.edu/~syliu/shredderyin/emacs_power.html)， 还是等笔者有时间了或许会试试煮杯咖啡边喝边写。

推荐在Linux下编写代码的读者或多或少的试一试[ctags](http://ctags.sourceforge.net/)。 ctags支持非常多的语言，可以将源代码中的各种符号（如:函数、宏类等信息）抽取出来做上标记并保存到一个文件中， 供其他文本编辑工具（VIM，EMACS等）进行检索。 它保存的文件格式符合[UNIX的哲学（小即是美）](http://zh.wikipedia.org/zh/Unix%E5%93%B2%E5%AD%A6)， 使用也比较简洁：

#在PHP源码目录(假定为/server/php-src)执行:

$ cd /server/php-src

$ ctags -R

#小技巧：在当前目录生成的tags文件中使用的是相对路径，

#若改用 ctags -R /server/ ，可以生成包含完整路径的ctags，就可以随意放到任意文件夹中了。

#在~/.vimrc中添加:

set tags+=/server/php-src/tags

#或者在vim中运行命令:

:set tags+=/server/php-src/tags

上面代码会在/sever/php-src目录下生成一个名为tags的文件，这个文件的[格式如下](http://ctags.sourceforge.net/FORMAT)：

{tagname}<Tab>{tagfile}<Tab>{tagaddress}

EG Zend/zend\_globals\_macros.h /^# define EG(/;" d

它的每行是上面的这样一个格式，第一列是符号名（如上例的EG宏），第二列是该符号的文件位置以及这个符号所在的位置。 VIM可以读取tags文件，当我们在符号上（可以是变量名之类）使用**CTRL+]**时VIM将尝试从tags文件中检索这个符号。 如果找到则根据该符号所在的文件以及该符号的位置打开该文件， 并将光标定位到符号定义所在的位置。 这样我们就能快速的寻找到符号的定义。

使用 **Ctrl+]** 就可以自动跳转至定义，**Ctrl+t** 可以返回上一次查看位置。这样就可以快速的在代码之间“游动”了。

习惯这种浏览代码的方式之后，大家会感觉很方便的。不过若你不习惯使用VIM这类编辑器，也可以看看下面介绍的[IDE](http://zh.wikipedia.org/wiki/%E9%9B%86%E6%88%90%E5%BC%80%E5%8F%91%E7%8E%AF%E5%A2%83)。

如果你使用的Mac OS X，运行ctags程序可能会出错，因为Mac OS X自带的ctags程序有些[问题](http://adamyoung.net/Exuberant-Ctags-OS-X)， 所以需要自己下载安装ctags，笔者推荐使用[homebrew](https://github.com/mxcl/homebrew)来安装。 如果执行还是会出错，请执行下ctags -v 或着 which ctags确保你执行的是新安装的ctags。

### 使用IDE查看代码

如果不习惯使用VIM来看代码，也可以使用一些功能较丰富的IDE，比如Windows下可以使用Visual Studio 2010 Express。 或者使用跨平台的[Netbeans](http://www.netbeans.org/)、[Eclipse](http://www.eclipse.org/)来查看代码， 当然，这些工具都相对较**重量级**一些，不过这些工具不管是调试还是查看代码都相对较方便，

在Eclipse及Netbeans下查看符号定义的方式通常是将鼠标移到符号上，同时按住**CTRL**，然后单击，将会跳转到符号定义的位置。

而如果使用VS的话， 在win32目录下已经存在了可以直接打开的工程文件，如果由于版本原因无法打开， 可以在此源码目录上新建一个基于现有文件的Win32 Console Application工程。

**常用快捷键**：

F12 转到定义

CTRL + F12转到声明

F3: 查找下一个

Shift+F3: 查找上一个

Ctrl+G: 转到指定行

CTRL + -向后定位

CTRL + SHIFT + -向前定位

对于一些搜索类型的操作，可以考虑使用Editplus或其它文本编辑工具进行，这样的搜索速度相对来说会快一些。 如果使用Editplus进行搜索，一般是选择 【搜索】 中的 【在文件中查找...】

# 第三节 常用代码

在PHP的源码中经常会看到的一些很常见的宏，或者有些对于才开始接触源码的读者比较难懂的代码。 这些代码在PHP的源码中出现的频率极高，基本在每个模块都会有他们的身影。本小节我们提取中间的一些进行说明。

## 1. "##"和"#"

宏是C/C++是非常强大，使用也很多的一个功能，有时用来实现类似函数内联的效果， 或者将复杂的代码进行简单封装，提高可读性或可移植性等。在PHP的宏定义中经常使用双井号。 下面对"##"及"#"进行详细介绍。

### 双井号(##)

在C语言的宏中，"##"被称为 连接符（concatenator），它是一种预处理运算符， 用来把两个语言符号(Token)组合成单个语言符号。 这里的语言符号不一定是宏的变量。并且双井号不能作为第一个或最后一个元素存在。如下所示源码：

*#define PHP\_FUNCTION ZEND\_FUNCTION*

*#define ZEND\_FUNCTION(name) ZEND\_NAMED\_FUNCTION(ZEND\_FN(name))*

*#define ZEND\_FN(name) zif\_##name*

*#define ZEND\_NAMED\_FUNCTION(name) void name(INTERNAL\_FUNCTION\_PARAMETERS)*

*#define INTERNAL\_FUNCTION\_PARAMETERS int ht, zval \*return\_value, zval \*\*return\_value\_ptr, \*

zval \*this\_ptr, int return\_value\_used TSRMLS\_DC

PHP\_FUNCTION(count);

*// 预处理器处理以后， PHP\_FUCNTION(count);就展开为如下代码*

void zif\_count(int ht, zval \*return\_value, zval \*\*return\_value\_ptr,

zval \*this\_ptr, int return\_value\_used TSRMLS\_DC)

宏ZEND\_FN(name)中有一个"##"，它的作用一如之前所说，是一个连接符，将zif和宏的变量name的值连接起来。 以这种连接的方式为基础，多次使用这种宏形式，可以将它当作一个代码生成器，这样可以在一定程度上减少代码密度， 我们也可以将它理解为一种代码重用的手段，间接地减少不小心所造成的错误。

### 单井号(#)

"#"是一种预处理运算符，它的功能是将其后面的宏参数进行 **字符串化操作（me:c语言定义）** ， 简单说就是在对它所引用的宏变量通过替换后在其左右各加上一个双引号， 用比较官方的话说就是将语言符号(Token)转化为字符串。 例如:

*#define STR(x) #x //会将x左右两侧自动加上双引号*

int main(int argc char\*\* argv)

{

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%s\n", STR(It's a long string)); // 输出 It's a long string

return 0;

}

如前文所说，It's a long string 是宏STR的参数，在展开后被包裹成一个字符串了（me：因为使用了#，它会将参数的左右加上双引号）。所以printf函数能直接输出这个字符串， 当然这个使用场景并不是很适合，因为这种用法并没有实际的意义，实际中在宏中可能会包裹其他的逻辑，比如对字符串进行封装等等。

## 2. 关于宏定义中的do-while循环

PHP源码中大量使用了宏操作，比如PHP5.3新增加的垃圾收集机制中的一段代码：

*#define ALLOC\_ZVAL(z) \*

do { \

(z) = (zval\*)emalloc(sizeof(zval\_gc\_info)); \

GC\_ZVAL\_INIT(z); \

} while (0)

这段代码，在宏定义中使用了 **do{ }while(0)** 语句格式。如果我们搜索整个PHP的源码目录，会发现这样的语句还有很多。 在其他使用C/C++编写的程序中也会有很多这种编写宏的代码，多行宏的这种格式已经是一种公认的编写方式了。 为什么在宏定义时需要使用do-while语句呢? 我们知道do-while循环语句是先执行循环体再判断条件是否成立， 所以说至少会执行一次。当使用do{ }while(0)时由于条件肯定为false，代码也肯定只执行一次， 肯定只执行一次的代码为什么要放在do-while语句里呢? 这种方式适用于宏定义中存在多语句的情况。 如下所示代码：

*#define TEST(a, b) a++;b++;*

if (expr)

TEST(a, b);

else

do\_else();

代码进行预处理后，会变成：

if (expr)

a++;b++;

else

do\_else();

这样if-else的结构就被破坏了if后面有两个语句，这样是无法编译通过的，那为什么非要do-while而不是简单的用{}括起来呢。 这样也能保证if后面只有一个语句。例如上面的例子，在调用宏TEST的时候后面加了一个分号， 虽然这个分号可有可无， 但是出于习惯我们一般都会写上。 那如果是把宏里的代码用{}括起来，加上最后的那个分号。 还是不能通过编译。 所以一般的多表达式宏定义中都采用do-while(0)的方式。

了解了do-while循环在宏中的作用，再来看"空操作"的定义。由于PHP需要考虑到平台的移植性和不同的系统配置， 所以需要在某些时候把一些宏的操作定义为空操作（me：为了跨平台）。例如在sapi\thttpd\thttpd.c文件中的VEC\_FREE():

*#ifdef SERIALIZE\_HEADERS*

*# define VEC\_FREE() smart\_str\_free(&vec\_str)*

*#else*

*# define VEC\_FREE() do {} while (0)*

*#endif*

这里涉及到条件编译，在定义了SERIALIZE\_HEADERS宏的时候将VEC\_FREE()定义为如上的内容，而没有定义时， 不需要做任何操作，所以后面的宏将VEC\_FREE()定义为一个空操作，不做任何操作，通常这样来保证一致性， 或者充分利用系统提供的功能。

有时也会使用如下的方式来定义“空操作”，这里的空操作和上面的还是不一样，例如很常见的Debug日志打印宏：

*#ifdef DEBUG*

*# define LOG\_MSG printf*

*#else*

*# define LOG\_MSG(...)*

*#endif*

在编译时如果定义了DEBUG则将LOG\_MSG当做printf使用（me：调用printf的时候实际执行的是LOG\_MSG函数），而不需要调试，正式发布时则将LOG\_MSG()宏定义为空， 由于宏是在预编译阶段进行处理的，所以上面的宏相当于从代码中删除了。

上面提到了两种将宏定义为空的定义方式（me:do{}while(0)和define 宏名(…)），看上去一样，实际上只要明白了宏只是简单的代码替换就知道该如何选择了。

## 3. #line 预处理

*#line 838 "Zend/zend\_language\_scanner.c"*

[#line](http://www.cppreference.com/wiki/preprocessor/line)预处理用于改变当前的行号（\_\_LINE\_\_）和文件名（\_\_FILE\_\_）。 如上所示代码，将当前的行号改变为838，文件名Zend/zend\_language\_scanner.c 它的作用体现在编译器的编写中，我们知道编译器对C 源码编译过程中会产生一些中间文件，通过这条指令， 可以保证文件名是固定的，不会被这些中间文件代替，有利于进行调试分析。

## 4.PHP中的全局变量宏

在PHP代码中经常能看到一些类似PG()， EG()之类的**函数**，他们都是PHP中定义的宏，这系列宏主要的作用是解决线程安全所写的全局变量包裹宏， 如$PHP\_SRC/main/php\_globals.h文件中就包含了很多这类的宏。例如PG这个PHP的核心全局变量的宏。 如下所示代码为其定义。

*#ifdef ZTS // 编译时开启了线程安全则使用线程安全库*

*# define PG(v) TSRMG(core\_globals\_id, php\_core\_globals \*, v)*

extern PHPAPI int core\_globals\_id;

*#else*

*# define PG(v) (core\_globals.v) // 否则这其实就是一个普通的全局变量*

extern ZEND\_API struct \_php\_core\_globals core\_globals;

*#endif*

如上，ZTS是线程安全的标记，这个在以后的章节会详细介绍，这里就不再说明。下面简单说说，PHP运行时的一些全局参数(me：很多是php.ini中的配置项，实际在php.ini中修改的配置就是修改这个全局变量)， 这个全局变量为如下的一个结构体，各字段的意义如字段后的注释：

struct \_php\_core\_globals {

zend\_bool magic\_quotes\_gpc; *// 是否对输入的GET/POST/Cookie数据使用自动字符串转义。*

zend\_bool magic\_quotes\_runtime; *//是否对运行时从外部资源产生的数据使用自动字符串转义*

zend\_bool magic\_quotes\_sybase; *// 是否采用Sybase形式的自动字符串转义*

zend\_bool safe\_mode; *// 是否启用安全模式*

zend\_bool allow\_call\_time\_pass\_reference; *//是否强迫在函数调用时按引用传递参数*

zend\_bool implicit\_flush; *//是否要求PHP输出层在每个输出块之后自动刷新数据*

long output\_buffering; *//输出缓冲区大小(字节)*

char \*safe\_mode\_include\_dir; *//在安全模式下，该组目录和其子目录下的文件被包含时，将跳过UID/GID检查。*

zend\_bool safe\_mode\_gid; *//在安全模式下，默认在访问文件时会做UID比较检查*

zend\_bool sql\_safe\_mode;

zend\_bool enable\_dl; *//是否允许使用dl()函数。dl()函数仅在将PHP作为apache模块安装时才有效。*

char \*output\_handler; *// 将所有脚本的输出重定向到一个输出处理函数。*

char \*unserialize\_callback\_func; *// 如果解序列化处理器需要实例化一个未定义的类，这里指定的回调函数将以该未定义类的名字作为参数被unserialize()调用，*

long serialize\_precision; *//将浮点型和双精度型数据序列化存储时的精度(有效位数)。*

char \*safe\_mode\_exec\_dir; *//在安全模式下，只有该目录下的可执行程序才允许被执行系统程序的函数执行。*

long memory\_limit; *//一个脚本所能够申请到的最大内存字节数(可以使用K和M作为单位)。*

long max\_input\_time; *// 每个脚本解析输入数据(POST, GET, upload)的最大允许时间(秒)。*

zend\_bool track\_errors; *//是否在变量$php\_errormsg中保存最近一个错误或警告消息。*

zend\_bool display\_errors; *//是否将错误信息作为输出的一部分显示。*

zend\_bool display\_startup\_errors; *//是否显示PHP启动时的错误。*

zend\_bool log\_errors; *// 是否在日志文件里记录错误，具体在哪里记录取决于error\_log指令*

long log\_errors\_max\_len; *//设置错误日志中附加的与错误信息相关联的错误源的最大长度。*

zend\_bool ignore\_repeated\_errors; *// 记录错误日志时是否忽略重复的错误信息。*

zend\_bool ignore\_repeated\_source; *//是否在忽略重复的错误信息时忽略重复的错误源。*

zend\_bool report\_memleaks; *//是否报告内存泄漏。*

char \*error\_log; *//将错误日志记录到哪个文件中。*

char \*doc\_root; *//PHP的”根目录”。*

char \*user\_dir; *//告诉php在使用 /~username 打开脚本时到哪个目录下去找*

char \*include\_path; *//指定一组目录用于require(), include(), fopen\_with\_path()函数寻找文件。*

char \*open\_basedir; *// 将PHP允许操作的所有文件(包括文件自身)都限制在此组目录列表下。*

char \*extension\_dir; *//存放扩展库(模块)的目录，也就是PHP用来寻找动态扩展模块的目录。*

char \*upload\_tmp\_dir; *// 文件上传时存放文件的临时目录*

long upload\_max\_filesize; *// 允许上传的文件的最大尺寸。*

char \*error\_append\_string; *// 用于错误信息后输出的字符串*

char \*error\_prepend\_string; *//用于错误信息前输出的字符串*

char \*auto\_prepend\_file; *//指定在主文件之前自动解析的文件名。*

char \*auto\_append\_file; *//指定在主文件之后自动解析的文件名。*

arg\_separators arg\_separator; *//PHP所产生的URL中用来分隔参数的分隔符。*

char \*variables\_order; *// PHP注册 Environment, GET, POST, Cookie, Server 变量的顺序。*

HashTable rfc1867\_protected\_variables; *// RFC1867保护的变量名，在main/rfc1867.c文件中有用到此变量*

short connection\_status; *// 连接状态，有三个状态，正常，中断，超时*

short ignore\_user\_abort; *// 是否即使在用户中止请求后也坚持完成整个请求。*

unsigned char header\_is\_being\_sent; *// 是否头信息正在发送*

zend\_llist tick\_functions; *// 仅在main目录下的php\_ticks.c文件中有用到，此处定义的函数在register\_tick\_function等函数中有用到。*

zval \*http\_globals[6]; *// 存放GET、POST、SERVER等信息*

zend\_bool expose\_php; *// 是否展示php的信息*

zend\_bool register\_globals; *// 是否将 E, G, P, C, S 变量注册为全局变量。*

zend\_bool register\_long\_arrays; *// 是否启用旧式的长式数组(HTTP\_\*\_VARS)。*

zend\_bool register\_argc\_argv; *// 是否声明$argv和$argc全局变量(包含用GET方法的信息)。*

zend\_bool auto\_globals\_jit; *// 是否仅在使用到$\_SERVER和$\_ENV变量时才创建(而不是在脚本一启动时就自动创建)。*

zend\_bool y2k\_compliance; *//是否强制打开2000年适应(可能在非Y2K适应的浏览器中导致问题)。*

char \*docref\_root; *// 如果打开了html\_errors指令，PHP将会在出错信息上显示超连接，*

char \*docref\_ext; *//指定文件的扩展名(必须含有’.')。*

zend\_bool html\_errors; *//是否在出错信息中使用HTML标记。*

zend\_bool xmlrpc\_errors;

long xmlrpc\_error\_number;

zend\_bool activated\_auto\_globals[8];

zend\_bool modules\_activated; *// 是否已经激活模块*

zend\_bool file\_uploads; *//是否允许HTTP文件上传。*

zend\_bool during\_request\_startup; *//是否在请求初始化过程中*

zend\_bool allow\_url\_fopen; *//是否允许打开远程文件*

zend\_bool always\_populate\_raw\_post\_data; *//是否总是生成$HTTP\_RAW\_POST\_DATA变量(原始POST数据)。*

zend\_bool report\_zend\_debug; *// 是否打开zend debug，仅在main/main.c文件中有使用。*

int last\_error\_type; *// 最后的错误类型*

char \*last\_error\_message; *// 最后的错误信息*

char \*last\_error\_file; *// 最后的错误文件*

int last\_error\_lineno; *// 最后的错误行*

char \*disable\_functions; *//该指令接受一个用逗号分隔的函数名列表，以禁用特定的函数。*

char \*disable\_classes; *//该指令接受一个用逗号分隔的类名列表，以禁用特定的类。*

zend\_bool allow\_url\_include; *//是否允许include/require远程文件。*

zend\_bool exit\_on\_timeout; *// 超时则退出*

*#ifdef PHP\_WIN32*

zend\_bool com\_initialized;

*#endif*

long max\_input\_nesting\_level; *//最大的嵌套层数*

zend\_bool in\_user\_include; *//是否在用户包含空间*

char \*user\_ini\_filename; *// 用户的ini文件名*

long user\_ini\_cache\_ttl; *// ini缓存过期限制*

char \*request\_order; *// 优先级比variables\_order高，在request变量生成时用到，个人觉得是历史遗留问题*

zend\_bool mail\_x\_header; *// 仅在ext/standard/mail.c文件中使用，*

char \*mail\_log;

zend\_bool in\_error\_log;

};

上面的字段很大一部分是与php.ini文件中的配置项对应的。 在PHP启动并读取php.ini文件时就会对这些字段进行赋值， 而用户空间的ini\_get()及ini\_set()函数操作的一些配置也是对这个全局变量进行操作的。

在PHP代码的其他地方也存在很多类似的宏，这些宏和PG宏(me:PHP的Global宏)一样，都是为了将线程安全进行封装，同时通过约定的 **G** 命名来表明这是全局的， 一般都是个缩写，因为这些全局变量在代码的各处都会使用到，这也算是减少了键盘输入。 我们都应该[尽可能的**懒**](http://blogoscoped.com/archive/2005-08-24-n14.html)不是么？

如果你阅读过一些PHP扩展话应该也见过类似的宏，这也算是一种代码规范，在编写扩展时全局变量最好也使用这种方式命名和包裹， 因为我们不能对用户的PHP编译条件做任何假设。

# 第四节 小结

**不积跬步，无以至千里。**

完成这章后，我们就可以开始我们的千里之行了，我们完成的第一步：起步。

在本章，我们开始搭建了PHP源码阅读环境，探讨了PHP的源码结构和阅读PHP源码的方法， 并且对于一些常用的代码有了一定的了解。我们希望所有的这些能为源码阅读减轻一些难度， 可以更好的关注PHP源码本身在功能上的实现。

好了，下一步我们从宏观上来看看PHP的实现：概览。