# 目录结构

|  |  |
| --- | --- |
| appveyor | 都是一些bat脚本，为PHP的Windows构建提供持续集成。 |
| build | linux下编译相关的目录。 |
| ext | 即extension，php下的扩展包源码，如我们熟知的mysqli、gd、curl、mbstring等等。 |
| main | 为php的主要代码，主要是输入/输出、web通信，以及php框架初始化操作等，比如fastcgi协议的解析、扩展的加载、php配置的解析等工作都是由它来完成的。 |
| netware | 网络目录，以前的版本没有此目录，里面有2个文件sendmail\_nw.h、start.c，分别定义socket通信所需要的头文件和具体实现。 |
| pear | 扩展包目录，PEAR即：PHP Extension and Application Repository。它是一个PHP扩展及应用的一个代码仓库，简单地说，pear就是PHP的cpan。具体用户可以查阅相关资料。 |
| sapi | 是php的应用接口层。可以在不同环境中应用php，比如命令行下、web环境中、嵌入其他应用。为此，php提供一个SAPI层以适配不同的应用环境，SAPI可以认为是php的宿主环境，也是整个php框架最外层的一部分，它主要负责php框架的初始化工作。  经常用到的3个SAPI是cgi、cli、fpm。 |
| scripts | 脚本目录。 |
| tests | 测试目录，Helper for simple tests to check return-value。 |
| travis | compile.sh |
| TSRM | 线程安全相关的实现。 |
| win32 | Windows下编译PHP有关的脚本。 |
| Zend | 是php解析器的主要实现，即ZendVM，它是php语言的核心实现，php代码的解释、执行就是由zend完成的。它介于应用与实际计算机中间，主要分为：编译器、执行器两个部分。  其中编译器负责将php代码解释为执行器可以识别的指令，执行器负责执行指令，ZendVM相当于java中的JVM，都是抽象出来的虚拟计算机。 |

# PHP的生命周期

　(命令执行) 扩展模块初始化 -> (请求脚本) 扩展请求初始化 -> (执行脚本) 扩展请求关闭 -> (完成请求) 扩展模块关闭

　这些都在你的扩展主文件 myext.c 中。

PHP\_MINIT\_FUNCTION(myext)

{

　　# 注册常量或类等初始化操作

return SUCCESS;

}

PHP\_RINIT\_FUNCTION(myext)

{

　　# 例如记录请求开始时间

return SUCCESS;

}

PHP\_RSHUTDOWN\_FUNCTION(myext)

{

　　# 例如记录请求结束时间，记录日志

return SUCCESS;

}

PHP\_MSHUTDOWN\_FUNCTION(myext)

{

　　# 注销一些持久化资源

　　return SUCCESS;

}

# 扩展框架

第一步：生成代码

PHP提供了生成基本代码的工具 ext\_skel。

这个工具在PHP源代码的./ext目录下。

$ cd php\_src/ext/

$ ./ext\_skel --extname=say

extname参数的值就是扩展名称。

执行ext\_skel命令后，这样在当前目录下会生成一个与扩展名一样的目录。

第二步，修改config.m4配置文件

config.m4的作用就是配合phpize工具生成configure文件。

configure文件是用于环境检测的。

检测扩展编译运行所需的环境是否满足。

现在我们开始修改config.m4文件。

打开，config.m4文件后，你会发现这样一段文字。

$ cd ./say

$ vim ./config.m4

dnl If your extension references something external, use with:

dnl PHP\_ARG\_WITH(say, for say support,

dnl Make sure that the comment is aligned:

dnl [ --with-say Include say support])

dnl Otherwise use enable:

dnl PHP\_ARG\_ENABLE(say, whether to enable say support,

dnl Make sure that the comment is aligned:

dnl [ --enable-say Enable say support])

其中，dnl 是注释符号。

上面代码说，如果所编写的扩展依赖其它扩展或lib库，需要去掉PHP\_ARG\_WITH相关代码的注释。

否则，去掉 PHP\_ARG\_ENABLE 相关代码段的注释。

我们编写的扩展不需要依赖其他的扩展和lib库。

因此，去掉PHP\_ARG\_ENABLE前面的注释。

去掉注释后的代码如下：

PHP\_ARG\_ENABLE(say, whether to enable say support,

Make sure that the comment is aligned:

[ --enable-say Enable say support])

第三步，代码实现

修改say.c文件。

实现say方法。

找到PHP\_FUNCTION(confirm\_say\_compiled)，增加如下代码：

PHP\_FUNCTION(say)

{

zend\_string \*strg;

strg = strpprintf(0, "hello word");

RETURN\_STR(strg);

}

找到 PHP\_FE(confirm\_say\_compiled, 增加如下代码：

PHP\_FE(say, NULL)

修改后的代码如下：

const zend\_function\_entry say\_functions[] = {

PHP\_FE(say, NULL) /\* For testing, remove later. \*/

PHP\_FE(confirm\_say\_compiled, NULL) /\* For testing, remove later. \*/

PHP\_FE\_END /\* Must be the last line in say\_functions[] \*/

};

/\* }}} \*/

第四步，编译安装

$ phpize

$ ./configure

$ make && make install

修改php.ini文件，增加如下代码：

[say]

extension = say.so

然后执行，php -m 命令。

在输出的内容中，你会看到say字样。

第五步，调用测试

自己写一个脚本，调用say方法。看输出的内容是否符合预期。

## 扩展文件中的.c文件

最最重要的myfirstext.c文件究竟哪里重要？

首先，我们先将myfirstext.c文件划分一下区块儿，这样更加直观(因为充满了大量的注释，我在这里并没有列出注释的部分)

/\*\*

\* 头文件部分

\*/

#ifdef HAVE\_CONFIG\_H

#include "config.h"

#endif

#include "php.h"

#include "php\_ini.h"

#include "ext/standard/info.h"

#include "php\_myfirstext.h"

static int le\_myfirstext;

/\*\*

\* 自定义函数部分，看到该函数的参数还熟吗？这里就是我们上面自定义函数的实现部分！

\*/

PHP\_FUNCTION(confirm\_myfirstext\_compiled)

{

char \*arg = NULL;

int arg\_len, len;

char \*strg;

if (zend\_parse\_parameters(ZEND\_NUM\_ARGS() TSRMLS\_CC, "s", &arg, &arg\_len) == FAILURE) {

return;

}

len = spprintf(&strg, 0, "Congratulations! You have successfully modified ext/%.78s/config.m4. Module %.78s is now compiled into PHP.", "myfirstext", arg);

RETURN\_STRINGL(strg, len, 0);

}

/\*\*

\* Module初始化和Shutdown部分

\*/

PHP\_MINIT\_FUNCTION(myfirstext)

{

return SUCCESS;

}

PHP\_MSHUTDOWN\_FUNCTION(myfirstext)

{

return SUCCESS;

}

/\*\*

\* Request初始化和shutdown部分

\*/

PHP\_RINIT\_FUNCTION(myfirstext)

{

return SUCCESS;

}

PHP\_RSHUTDOWN\_FUNCTION(myfirstext)

{

return SUCCESS;

}

/\*\*

\* Module Info部分，这里主要控制将扩展信息打印到phpinfo()中

\*/

PHP\_MINFO\_FUNCTION(myfirstext)

{

php\_info\_print\_table\_start();

php\_info\_print\_table\_header(2, "myfirstext support", "enabled");

php\_info\_print\_table\_end();

/\* Remove comments if you have entries in php.ini

DISPLAY\_INI\_ENTRIES();

\*/

}

/\*\*

\* function\_entry部分，这里主要对我们前面自定义的confirm\_myfirstext\_compiled函数做一个封装

\*/

const zend\_function\_entry myfirstext\_functions[] = {

PHP\_FE(confirm\_myfirstext\_compiled, NULL) /\* For testing, remove later. \*/

{NULL, NULL, NULL}

/\* Must be the last line in myfirstext\_functions[] \*/

};

/\*\*

\* module\_entry部分，这里应该算是整个文件最重要的部分了吧，属于我们扩展的CPU，这里将会告诉PHP如何初始化我们的扩展。

\*/

zend\_module\_entry myfirstext\_module\_entry = {

STANDARD\_MODULE\_HEADER,

"myfirstext",

myfirstext\_functions,

PHP\_MINIT(myfirstext),

PHP\_MSHUTDOWN(myfirstext),

PHP\_RINIT(myfirstext), /\* Replace with NULL if there's nothing to do at request start \*/

PHP\_RSHUTDOWN(myfirstext), /\* Replace with NULL if there's nothing to do at request end \*/

PHP\_MINFO(myfirstext),

PHP\_MYFIRSTEXT\_VERSION,

STANDARD\_MODULE\_PROPERTIES

};

#ifdef COMPILE\_DL\_MYFIRSTEXT

ZEND\_GET\_MODULE(myfirstext)

#endif

# 目录&文件信息

## 函数定义宏

### ZEND\_FUNCTION()宏函数与PHP\_FUNCTION

function PHP zend

今天翻看同事的PHP扩展，很多地方都是使用的PHP\_FUNCTION，然而PHP\_FUNCTION、PHP\_FE这样的宏都是为了向下兼容才留下来的。

作为专为PHP 5.X而开发的扩展，这样的宏使用得越少越好，取而代之的是ZEND\_打头的系列。

在PHP 5.3中，ZEND engine更是做了很多改进，所以为了避免今后的可能出现的兼容性问题，程序员们应该“喜新厌旧”，要拥抱变化。

### FAST\_ZPP

　PHP内部函数使用 zend\_parse\_parameters() API 接受参数，将输入参数转换成c变量，

这个函数使用类似scanf()函数的参数解释方法，

通过一个字符串来定义输入参数的个数和类型，

例如(“s” - for string, “l” for long)，

不幸的是，每次调用这个函数时都要对这个这个字符串进行解析，这会加重性能开销。

例如，在一下wordpress的首页，

zend\_parse\_parameters()函数占用6%的cpu运行时间，

在一些简单的函数 is\_string() 或者 ord() zend\_parse\_parameters() 几乎占用将近90%的开销。

zend\_parse\_parameters（）参数说明：

第一个参数，参数个数。一般就使用ZEND\_NUM\_ARGS()，不需要改变。

第二个参数，格式化字符串。作用是指定传入参数与PHP内核类型的转换关系。

if (zend\_parse\_parameters(ZEND\_NUM\_ARGS() TSRMLS\_CC, "al|zb", &input, &offset, &z\_length, &preserve\_keys) == FAILURE) {

return;

}

FAST ZPP

在PHP7中新提供的方式。

为了提高参数解析的性能。

对应经常使用的方法，建议使用FAST ZPP方式。

使用方式：

以ZEND\_PARSE\_PARAMETERS\_START(1, 2)开头。

第一个参数表示必传的参数个数，

第二个参数表示最多传入的参数个数。

以ZEND\_PARSE\_PARAMETERS\_END();结束。

中间是传入参数的解析。

ZEND\_PARSE\_PARAMETERS\_START(2, 4)

Z\_PARAM\_ARRAY(input)

Z\_PARAM\_LONG(offset)

Z\_PARAM\_OPTIONAL

Z\_PARAM\_ZVAL(z\_length)

Z\_PARAM\_BOOL(preserve\_keys)

ZEND\_PARSE\_PARAMETERS\_END();

新API涵盖了现有API的所有可能性。

下表显示了老说明符和新宏之间的对应关系。

specifier Fast ZPP API macro args

| Z\_PARAM\_OPTIONAL

a Z\_PARAM\_ARRAY(dest) dest - zval\*

A Z\_PARAM\_ARRAY\_OR\_OBJECT(dest) dest - zval\*

b Z\_PARAM\_BOOL(dest) dest - zend\_bool

C Z\_PARAM\_CLASS(dest) dest - zend\_class\_entry\*

d Z\_PARAM\_DOUBLE(dest) dest - double

f Z\_PARAM\_FUNC(fci, fcc) fci - zend\_fcall\_info, fcc - zend\_fcall\_info\_cache

h Z\_PARAM\_ARRAY\_HT(dest) dest - HashTable\*

H Z\_PARAM\_ARRAY\_OR\_OBJECT\_HT(dest) dest - HashTable\*

l Z\_PARAM\_LONG(dest) dest - long

L Z\_PARAM\_STRICT\_LONG(dest) dest - long

o Z\_PARAM\_OBJECT(dest) dest - zval\*

O Z\_PARAM\_OBJECT\_OF\_CLASS(dest, ce) dest - zval\*

p Z\_PARAM\_PATH(dest, dest\_len) dest - char\*, dest\_len - int

P Z\_PARAM\_PATH\_STR(dest) dest - zend\_string\*

r Z\_PARAM\_RESOURCE(dest) dest - zval\*

s Z\_PARAM\_STRING(dest, dest\_len) dest - char\*, dest\_len - int

S Z\_PARAM\_STR(dest) dest - zend\_string\*

z Z\_PARAM\_ZVAL(dest) dest - zval\*

Z\_PARAM\_ZVAL\_DEREF(dest) dest - zval\*

+ Z\_PARAM\_VARIADIC('+', dest, num) dest - zval\*, num int

\* Z\_PARAM\_VARIADIC('\*', dest, num) dest - zval\*, num int

### PHP\_FUNCTION宏定义

宏定义，在源码的php-5.6.22-src\main\php.h文件里面，大概在347行

接着搜索ZEND\_FUNCTION宏定义，在php-5.6.22-src\Zend\zend\_API.h文件，大概在68行

还可以在这个文件里面找到接下来的ZEND\_NAMED\_FUNCTION宏定义和ZEND\_FN宏定义

接着其实就是INTERNAL\_FUNCTION\_PARAMETERS宏定义，可以在php-5.6.22-src\Zend\zend.h文件找到，大概是290行，它其实是一些固定的参数这些定义串联的步骤如下（以函数strpos为例子）：观察PHP\_FUNCTION的展开形式。

还可以接着展开TSRMLS\_DC宏定义，可以在php-5.6.22-src\TSRM\TSRM.h文件166行找到，展开后为

void zif\_name(

int ht, zval \*return\_value, zval \*\*return\_value\_ptr,

zval \*this\_ptr, intreturn\_value\_used, void \*\*\*tsrm\_ls

)

### INIT\_CLASS\_ENTRY@

### struct \_zend\_class\_entry {类的结构体定义

zend\_class\_entry \*myclass\_ce;

static zend\_function\_entry myclass\_method[] = {

{ NULL, NULL, NULL }

};

ZEND\_MINIT\_FUNCTION(sample3)

{

zend\_class\_entry ce;

//"myclass"是这个类的名称。

INIT\_CLASS\_ENTRY(ce, "myclass",myclass\_method);

myclass\_ce = zend\_register\_internal\_class(&ce TSRMLS\_CC);

return SUCCESS;

}

### ZEND\_MINIT\_FUNCTION@

### zend\_class\_entry

Zend/zend\_API.h

zend\_class\_entry 是内核中定义的一个结构体（typedef struct \_zend\_function\_entry），

是内核实现PHP语言中类与对象的一个非常基础、关键的结构类型。

相当于定义的类的原型。

如果想获得一个名字为myclass的类该怎么做呢？

首先定义一个zend\_class\_entry变量，并为它设置名字，最后注册到runtime中去。

zend\_class\_entry \*myclass\_ce;

static zend\_function\_entry myclass\_method[] = {

{ NULL, NULL, NULL }

};

ZEND\_MINIT\_FUNCTION(sample3)

{

zend\_class\_entry ce;

//"myclass"是这个类的名称。

INIT\_CLASS\_ENTRY(ce, "myclass",myclass\_method);

myclass\_ce = zend\_register\_internal\_class(&ce TSRMLS\_CC);

return SUCCESS;

}

这样便定义了一个类myclass，而且我们可以正常的在PHP语言中使用它，比如：

<?php

$obj = new myclass();

我们上面还定义了一个myclass\_ce指针，他是干什么用的呢？当我们在扩展中对这个类进行操作，比如生成实例的时候，会使用到它，它的作用就类似于打开文件的操作句柄。

PHP\_MINIT\_FUNCTION(myExt);

PHP\_MSHUTDOWN\_FUNCTION(myExt);

PHP\_RINIT\_FUNCTION(myExt);

PHP\_RSHUTDOWN\_FUNCTION(myExt);

PHP\_MINFO\_FUNCTION(myExt);

### ZEND\_API

定义都是类似于

#if defined(\_\_GNUC\_\_) && \_\_GNUC\_\_ >= 4

# define ZEND\_API \_\_attribute\_\_ ((visibility("default")))

#else

# define ZEND\_API

#endif

一、预定义\_\_GNUC\_\_宏

1 \_\_GNUC\_\_ 是gcc编译器编译代码时预定义的一个宏。需要针对gcc编写代码时， 可以使用该宏进行条件编译。

2 \_\_GNUC\_\_ 的值表示gcc的版本。需要针对gcc特定版本编写代码时，也可以使用该宏进行条件编译。

3 \_\_GNUC\_\_ 的类型是“int”，该宏被扩展后， 得到的是整数字面值。可以通过仅预处理，查看宏扩展后的文本。

知道ZEND\_API定义为：

如果编译器使用的是gcc且GNUC的版本大于等于4,则定义ZEND\_API为 \_\_attribute\_\_ ((visibility("default")))

那\_\_attribute\_\_到底是干嘛的呢？

查阅关于C的相关资料得出结论:

\_\_attribute\_\_ ((visibility("default")))定义的函数都是可见的

## /zend目录

### zend\_types.h数据类型

#### zend\_string

#### zval

#### HashTable

#### 数据类型判断的属性

##### IS\_UNDEF

##### IS\_NULL

##### IS\_FALSE

##### IS\_TRUE

##### IS\_LONG

##### IS\_DOUBLE

##### IS\_STRING

##### IS\_ARRAY

##### IS\_OBJECT

##### IS\_RESOURCE

##### IS\_REFERENCE

##### IS\_CONSTANT

##### IS\_CONSTANT\_AST

\_IS\_BOOL

IS\_CALLABLE

IS\_ITERABLE

IS\_VOID

IS\_INDIRECT

IS\_PTR

\_IS\_ERROR

#### 字符串属性宏

##### ZSTR\_LEN字符串长度

#define ZSTR\_LEN(zstr) (zstr)->len

##### ZSTR\_VAL字符串的值

#define ZSTR\_VAL(zstr) (zstr)->val

##### ZSTR\_HASH

#define ZSTR\_HASH(zstr) zend\_string\_hash\_val(zstr)

##### ZSTR\_H

#define ZSTR\_H(zstr) (zstr)->h

### zend\_hash.h哈希处理

宏方法用于遍历哈希和对哈希进行操作

#### zend\_hash\_exists

检测指定的key在哈希中是否存在。key为字符串。

#### zend\_hash\_index\_exists

检测指定的key在哈希中是否存在。key为数字。

#### zend\_hash\_find

根据key查找指定的值。key为字符串。

#### zend\_hash\_index\_find

根据key查找指定的值。key为数字。

#### zend\_hash\_update

更新指定key的值。key为字符串。

#### zend\_hash\_index\_update

更新指定key的值。key为数字。

### zend\_hash.c类型相关的方法

#### Z\_TYPE\_P(zval \*)

#### Z\_STRLEN\_P(zval \*)

#### 数组相关处理

##### zend\_array\_count(HashTable \*)

获取数组的元素个数

### zend\_API.h

#### 返回值

方法的返回值是使用`RETURN\_`开头的宏方法进行返回的。

常用的宏方法有：

##### RETURN\_NULL()

返回null

##### RETURN\_LONG(l)

返回整型

##### RETURN\_DOUBLE(d)

返回浮点型

##### RETURN\_STR(s)

返回一个字符串。参数是一个zend\_string \* 指针

##### RETURN\_STRING(s)

返回一个字符串。参数是一个char \* 指针

##### RETURN\_STRINGL(s, l)

返回一个字符串。第二个参数是字符串长度。

##### RETURN\_EMPTY\_STRING()

返回一个空字符串。

##### RETURN\_ARR(r)

返回一个数组。参数是zend\_array \*指针。

##### RETURN\_OBJ(r)

返回一个对象。参数是zend\_object \*指针。

##### RETURN\_ZVAL(zv, copy, dtor)

返回任意类型。

参数是 zval \*指针。

##### RETURN\_FALSE

返回false

##### RETURN\_TRUE

返回true

##### RETURN\_STR

#### ZEND\_BEGIN\_MODULE\_GLOBALS& ZEND\_END\_MODULE\_GLOBALS

### zend\_long.h

zend\_long = int32\_t| int64\_t

zend\_ulong = uint32\_t| int64\_t

zend\_off\_t = uint32\_t| int64\_t

### zend\_ini.h

ZEND\_INI\_BEGIN& ZEND\_INI\_END

\_zend\_ini\_entry\_def

## /sapi目录

## main目录

### php\_stdint.h

### php\_config.h

### php\_ini.h