中软国际公司内部技术规范

PHP 语言编程规范

中软国际科技服务有限公司

修订声明

参考:华为编程规范，PHP 开发规范，PHP Coding Standard 等。

日期	修订版本	修订描述	作者
2011-11-2	1.00	初稿完成	郑敦庄
2013-8-27	1.01	汇总整理	覃晓元(WX153018) 施玉洲(KF70940)
2017-7-11	1.1	汇总整理，调整格式，删除第 6 章	陈丽佳（wx435329）

目录

目录

前言：

编程规范目的

1. 统一编程风格
2. 提高代码的可阅读性
3. 减少错误产生
4. 减少性能漏洞
5. 提高代码可靠性
6. 减少错误的编码设计
7. 作为代码检查的依据
8. 建立可维护的 php　语言编程规范

1 排版规则

1.1 缩进

对齐只使用空格键，不使用 TAB 键。 缩进的单位为 4 个空格。

说明：以免用不同的编辑器阅读程序时，因 TAB 键所设置的空格数目不同而造成程序布局变化。 虽然使用空格会增加文件的大小,但在局域网中几乎可以忽略,且在编译过程中也可被消除掉。

1.2 空格规则

空格应在以下情况时使用:

跟在((左括号)后面的关键字应被一个空格隔开。

```
while (true)
```

函数名与左括号之间不应该有空格。这能帮助区分关键字和函数调用。 所有的二元操作符,除了左括号和左方括号应用空格将其与操作数隔开。一元操作符与其操作数 之间不应有空格,除非操作符是个单词,比如 typeof。

每个在控制部分,比如 for 语句中的; (分号)后须跟一个空格。每个, (逗号)后应跟一个空格。

1.2.1 逻辑运算符前后必须加空格

正确 $a == $b;

错误

$a==$b;

$a ==$b;

正确 $a++; $a--;

错误 $a ++; $a --;

自加自减运算符不能加空格。

1.2.2 多个参数分隔时必须加空格

正确 $g_pro , $g_user , $g_show;

```
getDbInfo($host, $user, $passwd);
```

错误 $g_pro,$g_user,$g_show;

getDbInfo($host,$user,$passwd);

1.2.3 语法关键字后必须加空格

if, for , while, switch 等关键
例如:
正确 for ($i = 0; $i < 10; $i++)

错误 for($i = 0; $i < 10; $i++)

1.2.4 字符串和变量连接规则

字符串与变量连接使用’.’号时，必须在’.’前后加空格，使用”号自动转义变量时必须在变 量前后加” {}”。

正确 $myName = 'file_' . $var1;

$myName ="file_{$var1}";

错误 $myName = 'file_'.$var1;

$myName = "file_$var1";

1.3 换行

每行只写一个语句。 用空行来将逻辑相关的代码 块分割开可以提高程序的可读性。相对独立 的程序块之间、变量说明之后必须加空行。

1.3.1 较长的语句（>80 字符）

要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的 新行要进 行适当的缩进，使排版整齐，语句可读。

示例:

```
for ($i = 0, $j = 0; ($i < $bufferKeyword[wordIndex]->wordLength)
```

```
    && ($j < $NewKeyword->wordLength); $i++, $j++)
{
    ... // program code
}

for ($i = 0, $j = 0;
        ($i < $firstWordLength) && ($j < $secondWordLength); $i++, $j++)
{
   ... // code
}
```

2 命名规范

约定类命名使用 Pascal 命名法，即用英文的大小写来分隔单词，所有单词的首字母大写；变量 和方法都使用 Camel 命名法，即第一个英文首字母小写，其余单词首字母大写。

特殊情况会在下面特殊说明，如类的私有成员应以’_’开头，控制器的 Action 方法除 Action 外全部小写。

如：getnamelistAction。

2.1 变量命名

2.1.1 局部变量/成员变量

所有变量都要预先声明，并注明其意义,在函数的首部定义所有的变量。不要使用一个声明一个； 最好把每个变量的声明语句单独放到一行,并加上注释说明。所有变量按照字母排序。

局部变量、成员变量、函数参数都使用 Camel 命名法，即第一个英文首字母小写，其余单词首字母大写。如：

```
$currentEntry;  // 当前选择项

$le    // 缩进程
$s     // 表格大
```

类的私有属性以 _ 开头，如 private $_privateValue;

2.1.2 全局变量命名

全局变量使用 g 前缀。 例:

```
global $gLog;
global &$gLog;
```

尽量减少全局变量的使用,不要让局部变量覆盖全局变量。

2.1.3 静态变量命名

静态变量使用 s 前缀。 如:$sValue。

2.1.4 命名/定义全局常量

全局常量用'_'分隔每个单词，并且全部使用大写。
理由:
这是命名全局常量的传统。你要注意不要与其它的定义相冲突。

例如:

```
define("A_GLOBAL_CONSTANT", "Hello world!");
```

2.2 类命名

在类中，方法放到属性定义前边、公用方法放到专用方法前边。

在为类命名前首先要知道它是什么。如果通过类名的提供的线索，你还是想不起这个类是什么 的话，那 么你的设计就还做的不够好。

超过三个词组成的混合名是容易造成系统各个实体间的混淆，再看看你的设计，尝试使用(CRC Session card)看看该命名所对应的实体是否有着那么多的功用。

对于派生类的命名应该避免带其父类名的诱惑，一个类的名字只与它自身有关，和它的父类叫 什么无关。 有时后缀名是有用的，例如：如果你的系统使用了代理（agent），那么就把某个部件命名为“下载代理” （DownloadAgent）用以真正的传送信息。

Zend 框架下的 Controller 的命名需要特殊处理，它的命名规则 ModuleName_XxxController

例如：Console_ElasticIpController 它表示 Console 模块下弹性 Ip 控制器。

Zend 框架下 Model 的命名建议统一加上 Model 后缀。 这样做好处：与控制器命名对应，区分实体对象。 如：ElasticIpModel，有时候可能会有 ElasticIp 这个实体类。

2.2.1 接口命名原则

接口名字在类的原则上加前缀 I。

例如：IComponent(描述性名词)、ICustomAttributeProvider (名词短语)、IPersistable (形容词)等。

有的类，必须用字母 I 作为类名前缀，而又不是一个接口。这是可以接受的，因为有的类名就 是 I 开头的， 例如： IdentityStore。这种情况和接口的区别在于其第二个字母是小写的。

有的时候，定义完一个接口之后，也会定义一个类作为接口的标准实现。该类和该接口应该有 类似的名字， 唯一的区别就是接口名称前缀为字母 I。

如：IComponent 和它的标准实现 class Component {}。

2.2.2 Exception 命名原则

异常的命名以“Exception”为后缀。

2.3 函数/成员方法命名

函数和类的成员方法，全部采用 Camel 命名法。 例如：

```
function someBloodyFunction() {}
```

控制器的 Action 方法采用特殊的 Camel 命名法，除 Action 外全部小写：

如：getnamelistAction。

构造函数统一用 construct。 私有方法加’_’前缀：

如：private function _initialize(){...};

2.4 方法/函数命名修饰

通常每个方法和函数都是执行一个动作的，所以对它们的命名应该清楚的说明它们是做什么的：用 checkForErrors()代替 errorCheck()，用 dumpDataToFile()代替 dataFile()。这么做也可以使 功能和 数据成为更可区分的物体。

有时后缀名是有用的：

Max - 含义为某实体所能赋予的最大值。

Cnt　　- 一个运行中的计数变量的当前值。

Key　　- 键值。

例如：retryMax 表示最多重试次数，retryCnt 表示当前重试次数。 有时前缀名是有用的：

Is　　- 含义为问一个关于某样事物的问题。无论何时，当人们看到 is 就会知道这是一个问题。

get　　- 含义为取得一个数值。

set　　- 含义为设定一个数值

例如：

```
$isHitRetryLimit。 function &status() {};
function doSomething(&$status) {};
```

使用常用的缩写时，只大写首字母，如 getHtml()。

2.5 缩写词不要全部使用大写字母

无论如何，当遇到以下情况，你可以用首字母大写其余字母小写来代替全部使用大写字母的方 法来表示缩写词。

使用：getHtmlStatistic. 不使用：getHTMLStatistic.

理由:当命名含有缩略词时，人们似乎有着非常不同的直觉。统一规定是最好，这样一来，命名的含 义就完 全可以预知了。

举个 NetworkABCKey 的例子，注意 C 是应该是 ABC 里面的 C 还是 key 里面的 C，这个是很 令人费解的。 有些人不在意这些，其他人却很讨厌这样。所以你会在不同的代码里看到不同的 规则，使得你不知道怎么 去叫它。

例如：

```
class FluidOz  // 不要写成 FluidOZ
function  getHtmlStatistic  // 不要写成 getHTMLStatistic
```

ps:不过有时也要是情况而定，如 PDO 类库本身全大写，那么 class MyPDO extends PDO{...}也就说得过去了，如果 PDO 小写有可能起反作用，主要目的在于容易理解，而不是做样子。

3 版式规则

3.1 语义分隔

各个函数、方法之间应该采用空行间隔；

ps:}结束大括号要有空行 同一个函数中联系紧密的语句之间可以不换行，其他情况需要换行。

3.2 圆括号规则

函数名与括号之间不需要加空格、语法关键字后的括号必须加空格。

正确

```
for ($i = 0; $i < 10; $i++)
strlen($my_name);
```

错误

```
for($i = 0; $i < 10; $i++ )
strlen ($my_name);
```

3.3 大括号{}规则

将大括号放置在关键词下方的同列处： 如：

```
if ($a)
{
        $b = $a;
}
```

从一个程序块移动到另一个程序块只需要用光标和你的括号匹配键就可以了，不需要来回的移 动到行末去找匹配的括号。

3.4 方法/函数

函数名与((左括号)之间不应该有空格。) {(左大括号)独占一行。 函数程序体应缩进四个空格。}(右大括号)与声明函数的那一行代码头部对齐。

```
function outer($c, $d)
{
        $e = $c * $d;
```

```
        return inner(0, 1,
    }

    function inner($a, $b, $e)
    {
        return ($e * $a) + $b;
    }
```

如果函数是匿名函数,则在 function 和((左括号)之间应有一个空格。如果省略了空格,否则会 让人感觉函 数名叫作 function。

```
    $a = function ($e)
    {
        return $e;
    };
```

3.5 关键字

if、for、do、while、case、switch、default 等语句自占一行，且 if、for、do、while 等语句的 执行

语句部分无论多少都要加括号{}。

示例：如下例子不符合规范：

```
if (NULL == $rUserCR) return;
```

应如下书写：

```
if (NULL == $rUserCR)
{
   return;
}
```

3.5.1 if 语句

if 语句应如以下格式：

```
if (condition)
{
        statements;
}
```

```
if (condition)
{
        statements;
}
else (condition)
{
        statements;
}

if (condition)
{
        statements;
}
else if(condition)
{
        statements;

}
else
{
      statements;
}
```

3.5.2 for 语句

for 语句应如以下格式:

```
for (initialization;condition; update)
{
        statements;
}
```

3.5.3 while 语句

while 语句应如以下格式:

```
while (condition)
{
        statements;
}
```

3.5.4 do 语句

do 语句应如以下格式:

```
do
```

```
{
        statements;
} while(condition);
```

不像别的复合语句,do 语句总是以;(分号)结尾。

3.5.5 switch 语句

switch 语句应如以下格式:

```
switch(expression)
{
        case expression: statements; break;
        default:
        statements;
}
```

每一组 statements(除了 default 应以 break,return,或者 throw 结尾。不要让它顺次往下执行。

3.5.6 try 语句

try 语句应如以下格式:

```
try
{
   statements;
}
catch (variable)
{
   statements;
}
```

3.5.7 return 语句

一条有返回值的 return 语句不要使用( )(括号)来括住返回值。如果返回表达式,则表达式应与 return 关键字在同一行,以避免误加分号错误。

4 编程规范

系统统一使用时间戳 time()作为时间标志，写入 mysql 时使用 INT(10)类型写入，读取时可以使用公共 函数库中的 getdate()将时间戳转换为标准时间格式；

引号统一使用'单引号，只有当引号重叠时才使用"双引号，这样每进程可以节省几百 K 内存； 统一使用<?php ?>，禁止使用<? ?>格式。

4.1 数组定义规则

数组定义和使用时中 key 值前后必须加单引号。

```
//正确
array(
        'name' => 'hws',
        'gender' => 'php'
);

//错误
array(
        name => 'hws',
        gender => 'php'
);
```

4.2 不要采用缺省方法测试非零值

不要采用缺省值测试非零值，也就是使用: if (FAIL != f())

比下面的方法好: if (f())

即使 FAIL 可以含有 0 值 ，也就是 PHP 认为 false 的表示。在某人决定用-1 代替 0 作为失 败返回值的时候一个显式的测试就可以帮助你了。就算是比较值不会变化也应该用显式的比较。 if (!($bufsize % strlen($str)))

应该写成: if (($bufsize % strlen($str)) == 0)以表示测试的数值（不是布尔）型。 一

个经常出问题的地方就是使用 strcmp 来测试一个字符等式，结果永远也不会等于缺省值。 非零测试采用基于缺省值的做法，那么其他函数或表达式就会受到以下的限制:

只能返回 0 表示失败，不能为/有其他的值。

命名以便让一个真(true)的返回值是绝对显然的，调用函数 IsValid()而不是 Checkvalid()。 Ps:最直接的体现就是 PHP 的 ip2long 函数，在 5.2.10 以前他返回-1 表示出错，之后他返 回 false 表示出错.

4.3 通常避免嵌入式的赋值

有时候在某些地方我们可以看到嵌入式赋值的语句，那些结构不是一个少冗余，可读性强的好 方法。

```
while ($a != ($c = getchar()))
{
        process the character
}
```

++和--操作符类似于赋值语句。因此，出于许多的目的，在使用函数的时候会产生副作用。使 用嵌入式赋值提高运行时性能是可能的。

无论怎样，程序员在使用嵌入式赋值语句时需要考虑在增长的速度和减少的可维护性两者间加 以权衡。 例如:

```
$a = $b+$c;
$d = $a+$r;
```

不要写成:

```
$d = ($a = $b+$c)+$r;
```

虽然后者可以节省一个周期。但在长远来看，随着程序的维护费用渐渐增长，程序的编写者对代码渐渐遗忘，就会减少在成熟期的最优化所得。

4.4 布尔逻辑类型

大部分函数在 FALSE 的时候返回 0，但是发挥非 0 值就代表 TRUE，因而不要用 1（TRUE，YES， 诸如此类）等式检测一个布尔值，应该用 0（FALSE，NO，诸如此类）的不等式来代替:

```
if (TRUE == func())
```

应该写成:

```
if (FALSE != func())
```

4.5 别在对象架构函数中做实际的工作

别在对象架构构造函数中做实际的工作， 构造函数应该包含变量的初始化和（或）不会发生失 败的操作。

构造不能返回错误。 例如:

```
class Device
{
        function device()
        {
             /* initialize and other stuff */

        }
        function open()
        {
            return FAIL;
        }
};

$dev = new Device;

if (FAIL == $dev->open())
{
        exit(1);
}
```

ps:通过例子清晰表明，问题所在就是构造函数不能 return. 所以不需要 return 的错误就没问题...(出错就自动挂掉全部程序)。

4.6 switch 格式

当一个 case 块处理后，直接转到下一个 case 块处理，在这个 case 块的最后应该加上注释。default case 总应该存在，它应该不被到达，然而如果到达了就会触发一个错误。 如果你要创立一个变量，那就把所有的代码放在块中。

例如：

```
switch (...)
{
      case 1:
       // FALL THROUGH
      case 2:
      {
          $v = get_week_number();
      }
```

```
        break;
        default;
    }
```

4.7 Continue 和 Break

Continue 和 break 其实是变相的隐蔽的 goto 方法。

Continue 和 break 像 goto 一样，它们在代码中是有魔力的，所以要节俭（尽可能少）的使用 它们。 使用了这一简单的魔法，由于一些未公开的原因，读者将会被定向到只有上帝才知道的 地方去。

Continue 有两个主要的问题：它可以绕过测试条件。 它可以绕过等/不等表达式。

看看下面的例子，考虑一下问题都在哪儿发生：

```
while (TRUE)
{
    // A lot of code
    if (/* some condition */)
    {
        continue;
    }
    // A lot of code
    if ($i ++ > STOP_VALUE)
    {
        break;
    }
}
```

注意："A lot of code"是必须的，这是为了让程序员们不能那么容易的找出错误。 通过以上的例子，我们可以得出更进一步的规则：continue 和 break 混合使用是引起灾难的正确方法。

ps:刚看还真没发现问题(悲剧),运行后发现问题所在...

4.8 ？

麻烦在于人们往往试着在 ？ 和 ： 之间塞满了许多的代码。以下的是一些清晰的连接规则：把条件放在括号内以使它和其他的代码相分离。

如果可能的话，动作可以用简单的函数。 把所做的动作，“?”，“:”放在不同的行，除非他们可以清楚的放在同一行。 例如：

```
(condition) ? funct1() : func2();
```

或

```
(condition)
? long statement
 : another long statement;
```

4.9 其他杂项

PHP 标签统一使用<?php ?>，禁止使用<? ?>格式。

使用完毕后的数组变量、对象变量、查询集合必须马上使用 unset()、free_result()释放资源。 PHP 文件中绝不能出现 html 语句，html 文件中尽可能避免出现 PHP 语句。

4.9.1 类定义文件中，定义体之外不得出现 echo、print 等输出语句

理由:

出现这样的语句，应该当做出现 bug 来看。

4.9.2 纯 php 文件不要加 ?>标签，只要<?php

4.9.3 在 HTML 网页中尽量不要穿插 PHP 代码

循环代码和纯粹变量输出（类似于）除外。 需要说明的是我们工作的上游，页面设计者的工作，假如在页面中穿插代码，将破坏结构，这 应当是我们需要避免的。

在这里的 PHP 代码只负责显示，多余的代码显然是不应该的。

4.9.4 没有含义的数字

一个在源代码中使用了的赤裸裸的数字是不可思议的数字，因为包括作者，在三个月内，没人 它的含义。 例如:

```
if (22 == $foo)
{
        start_thermo_nuclear_war();
 }
else if (19 == $foo)
```

```
{
        refund_lotso_money();
}
else if (16 == $foo)
{
          infinite_loop();
}

else
{
          cry_cause_im_lost();
}
```

在上例中 22 和 19 的含义是什么呢？如果一个数字改变了，或者这些数字只是简单的错误，你会怎么想？ 使用不可思议的数字是该程序员是业余运动员的重要标志.

你应该用 define()来给你想表示某样东西的数值一个真正的名字，而不是采用赤裸裸的数字， 例如:

```
define("PRESIDENT_WENT_CRAZY", "22");
define("WE_GOOFED", "19");
define("THEY_DIDNT_PAY", "16");

if (PRESIDENT_WENT_CRAZY == $foo)
{
        start_thermo_nuclear_war();
 }
else if (WE_GOOFED == $foo)
{
        refund_lotso_money();
}
else if (THEY_DIDNT_PAY == $foo)
{
          infinite_loop();
}
else
{
          happy_days_i_know_why_im_here();
}
```

现在不是变得更好了么？

4.9.5 PHP 文件扩展名

常见的 PHP 文件的扩展名有：html,.php,.php3,.php4,.phtml,.inc,.class... 这里我们约定：

1.所有浏览者可见页面使用.html(phtml)

2.所有类、函数库文件使用.php 扩展名描述的是那种数据是用户将会收到的。 PHP 是解释为 HTML 的。

4.9.6 总是将恒量放在等号/不等号的左边，

例如：

if (6 == $errorNum) ... 一个原因是假如你在等式中漏了一个等号，语法检查器会为你报错。第二个原因是你能立刻找 到数值而不 是在你的表达式的末端找到它。需要一点时间来习惯这个格式，但是它确实很有用。

4.9.7 包含调用

包含调用程序文件，请全部使用 require_once，以避免可能的重复包含问题； 包含调用缓存文件，由于缓存文件无法保证100%正确打开，请使用include_once 或include。

在必要时，可以使用@include_once 或@include 的方式，以忽略错误提示； 包含和调用代码中除 soap 接口自动生成代码外以“./”开头，应避免直接写程序文件名(例如：require_once ‘x.php’ ;)的做法。

4.10 SQL 规则

所有 SQL 语句中，除了表名、字段名称以外，全部语句和函数均需大写，应当杜绝小写方式或大小写混杂的写法。例如 select * from cdb_members;是不符合规范的写法。

很长的 SQL 语句应当有适当的断行，依据 JOIN、FROM、ORDER BY 等关键字进行界定。 通常情下，在对多表进行操作时，要根据不同表名称，对每个表指定一个 1~2 个字母的缩写，以利于语句简洁和可读性。

如下的语句范例，是符合规范的：

```
$query = $db->query("SELECT s.*, m.*
      FROM {$tablepre}sessions s, {$tablepre}members m
      WHERE m.uid=s.uid AND s.sid='$sid');
```

数据值两边用单引号”包括，并且应确保数据值中的单引号已经转义以防止 SQL 注入。能使用多表查询一次查询的数据要一次读完，避免多次查询数据库，数据库使用完要关闭连接；

4.10.1 输出网页的页面不出现 SQL 语句

这是 MVC 结构的编程思想所致，每层的任务不同，不允许越权使用。

4.10.2 进行 SQL 执行的数据必须进行有效性检测

特殊符号:

对于 MS SQL Server，’ %_[ ] 这些符号都是在书写 SQL 语句中的特殊含义字符，在 SQL 执 行前需要对这些 字符进行处理。

脚本符号:

对于 PHP 脚本标记，如，在进入数据库前需要检测处理。 这是数据库编程的一个约定，很多参考书上也是这么说，这里需要强调一下。

4.10.3 查询优化

MySQL 中并没有提供针对查询条件的优化功能，因此需要开发者在程序中对查询条件的先后顺序人工进行优化。例如如下的 SQL 语句：

```
SELECT * FROM table WHERE a>’0’ AND b<’1’ ORDER BY c LIMIT 10;
```

事实上无论 a>’ 0’ 还是 b<’ 1’ 哪个条件在前，得到的结果都是一样的，但查询速度就大不 相同，尤其在对大表进行操作时。

开发者需要牢记这个原则：最先出现的条件，一定是过滤和排除掉更多结果的条件；第二出现的次之；以此类推。因而，表中不同字段的值的分布，对查询速度有着很大影响。而 ORDER BY 中的条件，只与索引有关，与条件顺序无关。

除了条件顺序优化以外，针对固定或相对固定的 SQL 查询语句，还可以通过对索引结构进行优化，进而实现相当高的查询速度。原则是：在大多数情况下，根据 WHERE 条件的先后顺序和 ORDER BY 的排序字段的先后顺序而建立的联合索引，就是与这条 SQL 语句匹配的最优索引

结 构。尽管，事实的产品中不能只考虑一条 SQL 语句，也不能不考虑空间占用而建立太多的索引。

同样以上面的 SQL 语句为例，最优的当 table 表的记录达到百万甚至千万级后，可以明显的看到 索引优化带来的速度提升。

5 注释规则

5.1 一般规则

程序中要保证　20%的注释，边写程序边注释。不要吝啬注释。给以后需要理解你的代码的人们(或许就是你 自己)留下信息是非常有用的。注释应该和它们所注释的代码一样是书写良好且清 晰明了。偶尔的小幽默就 更不错了。记得要避免冗长或者情绪化。

及时地更新注释也很重要。错误的注释会让程序更加难以阅读和理解。 让注释有意义。重点在解释那些不容易立即明白的逻辑上。不要把读者的时间浪费在阅读类似 于: i= 0; //让 i 等于 0。

使用单行注释。块注释用于注释正式文档和无用代码。 把注释看成程序的一部分，在编写/维护代码时同时编写/维护注释；

注释完全采用 PHPDocumentor 的规范，以方便用其生成 API 级文档。详细规则请参见 PHPDocumentor 手册。

5.2 方法/函数注释

方法声明时要在开头说明其实现功能、各参数、返回值意义，复杂逻辑要在声明时说明其实现 思想，并在 关键步骤做出注释；

调用方法时也要指出其目的；如：

```
/**
 *绑定弹性 ip。
 *
 * @param $instanceId string 设备 Id
 * @param $publicIp string        弹性 Ip
 * @return SoapResponse          soap 响应
 *
 */
public function associateAddress($instanceId, $publicIp)
{
        $params = array ("instanceId" => $instanceId,
                                "publicIp" => $publicIp );
        return $this->callWebService(__FUNCTION__, $params);
}
```

5.3 类注释

类声明时要在头部注明类作用、作者、时间；修改时要注明修改人，修改说明； 示例：

```
/**
 * 弹性 IpModel 类，负责调用底层的远程方法
 * @author x0012345
 * @version 1.0
 * @created 08-十二月-2011 16:31:17
 */
class ElasticIpModel
{
    /**
     *
     * ec2 的服务类对象
     * @var Ec2Service
     */
      private $_Ec2ser;

       /**
        * 绑定弹性 Ip
           *     @param ip String
           *     @param instanceId String
        */
       public function associateIp($ip, $instanceId)
       {

       }
}
```

5.4 记录所有的空语句

总是记录下 for 或者是 while 的空块语句，以便清楚的知道该段代码是漏掉了，还是故意不写 的。

```
  while ($dest++ = $src++)


  ; // VOID
```

5.5 if (0)来注释外部代码块

有时需要注释大段的测试代码，最简单的方法就是使用 if (0)块:

```
function example()
{
       great looking
```

```
    code if (0)
        {
            lots of code
        }
        more code
}
```

你不能使用/**/，因为注释内部不能包含注释，而大段的程序中可以包含注释。

5.6 版权信息

```
// +----------------------------------+
// | huawei |
// +----------------------------------+
// | Copyright (c) 2011-2011 huawei |
// | Email huawei@huawei.com |
 // | Web http://www. huawei.com |
// +----------------------------------+
// | This source file is subject to PHP License |
// +----------------------------------+
```

备注使用//来标示版权信息，以免和 PHPDocumentor 的 page-level DocBlock 发生冲突。