PHP 代码规范

Author：张金龙

Version：1.10.11

2014/07/06

好居 haoju.cn 让看房更省心

目录

为什么我们需要编码规范.....3

规范的代码可以促进团队合作.....3

规范的代码可以减少 BUG 处理.....3

规范的代码可以降低维护成本.....4

规范的代码有助于代码审查.....5

养成代码规范的习惯，有助于程序员自身的成长.....5

编码规范.....7

文件.....8

排版.....10

注释.....14

命名规范.....18

类，属性，方法.....20

控制结构.....24

为什么我们需要编码规范

当了这么多年码畜，相信大家都听说过“代码规范”这个词。如果仅仅强制大家执行编码规范，而不告诉大家为什么使用编码规范，这是作为管理者的失责。书归正传，编码规范的重要性主要体现在以下几个方面：

规范的代码可以促进团队合作

一个项目大多都是由一个团队来完成，如果没有统一的代码规范，那么每个人的代码必定会风格迥异。且不说会存在多个人同时开发同一模块的情况，即使是分工十分明晰的，等到要整合代码的时候也有够头疼的了。大多数情况下，并非程序中有复杂的算法或是复杂的逻辑，而是去读别人的代码实在是一件痛苦的事情。统一的风格使得代码可读性大大提高了，人们看到任何一段代码都会觉得异常熟悉。显然的，规范的代码在团队的合作开发中是非常有益而且必要的。

规范的代码可以减少 BUG 处理

很多 IT 人士将程序员比做民工，这也的确非常的形象。就像刚才提到的，复杂的算法或逻辑只占项目中很小的比例，大多仅仅是垒代码的工作。可是越是简单，测试的 BUG 反而是越多，而且是无穷无尽的 BUG。这里很大的程度上是由于代码不规范所致。

没有对输入输出参数的约定，没有统一的异常处理，没有一致的日志

输出，不仅仅会导致一些低级错误的发生，而且还很难找到引起 BUG 的原因。

规范不是对开发的制约，而确实是有助于提高开发效率的。

规范的代码可以降低维护成本

随着我们项目经验的累积，会越来越重视后期维护的成本。而开发过程中的代码质量直接影响着维护的成本。因此，我们不得不从开发时便小心翼翼。

在第一点中曾提到，规范的代码大大提高了程序的可读性，几乎所有的程序员都曾做过维护的工作，不用多说，可读性高的代码维护成本必然会大大降低。

但是，维护工作不仅仅是读懂原有代码，而是需要在原有代码基础上作出修改。我们可以先想像没有统一风格的情况下，A 完成开发以后，B 进行维护加一段代码，过一段时间 C 又加一段代码……直到有一天 X 看到那一大堆乱码想死的心都有了，维护也就进行不下去了。因此，统一的风格有利于长期的维护。

另外，好的代码规范会对方法的度量、类的度量以及程序耦合性作出约束。这样不会出现需要修改一个上千行的方法或者去扩展一个没有接口的类的情况。规范的代码对程序的扩展性提高，无疑也是对维护人员的一个奖励。

规范的代码有助于代码审查

我个人是比较赞同进行代码审查的，这样可以及时纠正一些错误，而且可以对开发人员的代码规范作出监督。团队的代码审查同时也是一个很好的学习机会，对成员的进步也是很有益的。但是，开发随意，加重的代码审查的工作量及难度，并且使得代码审查工作没有根据，浪费了大量的时间却收效甚微。

代码规范不仅使得开发统一，减少审查难度，而且让代码审查有据可查，大大提高了审查效率和效果，同时代码审查也有助于代码规范的实施。一举多得，何乐而不为呢。

养成代码规范的习惯，有助于程序员自身的成长

即使明白代码规范的好处，但是有的迫于项目压力，有的因为繁琐的规范作出很多额外的工作，更有的不重视维护的问题，而很难贯彻代码规范。

那么，我们需要了解，规范开发最大的受益人其实是自己！

你有没有花费很多的时候查找自己的代码呢？尤其是出现 BUG 的时候需要逐行的 DEBUG？自己写的代码乱了头绪的确实也见了不少。我们应该做的就是规范开发，减少自己出现的错误。很多时候项目的压力一部分也是由于前期开发中遗留的众多的问题。

还有的人觉得自己可以完成高难度的算法，就认为自己能力很强，不把规范放在眼里。很多人确实是这样，追求个性，大概让别人看他的代码一头雾水更觉得得意。殊不知复杂的算法确实可以体现你个人的

逻辑能力，但是绝不代表你的开发水平。我们知道一些开源项目，一些大师级人物写得程序都是极其规范的。并非规范了就代表高水平，实际上是规范的代码更有利于帮助你理解开发语言、理解模式、理解架构，能够帮助你快速提升开发水平。不明白这点，即使你写的再高明的算法，没准哪天也被当作乱码处理掉。

记住！每天垒乱码（或许你不觉得，但是大多时候在别人眼中确实就是乱码）并不能使你获得更多的进步，相反要达到高水平的程序员，养成良好的开发习惯是绝对必需的。

不要沉迷表面的得失，看似无用的东西要经过慢慢的累积由量变达到质变的时候，你才能感受到其价值所在。

编码规范

本文档将从 PHP 源码文件、代码排版、注释等多个方面对好居的代码规范进行说明。

文件

一、 所有的 PHP 源文件必须使用 Unix LF（”\n”）作为行结束符。

二、 源文件中 PHP 编码格式必须只使用不带字节顺序标记（BOM）的 UTF-8。

三、 源文件只允许使用“<?php”作为 PHP 代码开始标签。

四、 纯 PHP 代码源文件的关闭标签“?>”必须省略。

五、 一个源文件建议只用来做声明（类，函数，常量等）或者只用来做一些引起副作用的操作（例如：输出信息，修改.ini 配置等），但不建议同时做这两件事。

下面是一个既包含声明又有副作用的示例文件：即应避免的例子：

```php
ini_set('error_reporting', E_ALL);    // 副作用：修改了ini配置

include 'file.php';                   // 副作用：载入了文件

echo "<html>\n";                      // 副作用：产生了输出

/* 申明一个函数 */
function foo()
{
    // 函数体
}
```

下面是一个仅包含声明的示例文件：即应提倡的例子：

```php
/* 申明一个函数 */
function foo()
{
    // 函数体
}

/* 条件式声明不算做是副作用 */
if (!function_exists('bar'))
{
    function bar()
    {
        // 函数体
    }
}
```

排版

一、 程序块要采用缩进风格编写，缩进只能使用空格，不能使用TAB符。缩进的空格书为 4 个。

二、 PHP 关键字（不包括 return）必须大写，例如：TRUE，FALSE，NULL。

三、 相对独立的程序块之间，必须加空格。

```
4
5   $condition = NULL;
6
7   /* 和上面的变量定义空一行 */
8   if ($condition != NULL)
9   {
10      # code...
11  }
12
13  /* 和上面的IF块空一行 */
14  if ($condition)
15  {
16      # code...
17  }
18
19  /* 和上面的IF块空一行 */
20  for ($i = 0; $i < ; $i++)
21  {
22      # code...
23  }
24
25  /* 和上面的FOR块空一行 */
26  $condition = FALSE;
27
```

独立的程序块包含但不限于：IF 块，FOR 块，FOREACH 块，WHILE 块，CLASS 块，FUNCTION 块。

四、 较长的语句（> 120 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，操作符放在新行首。划分的新行要进

行适当的缩进，一般是较之前行，保持四个空格的缩进，以使排版整齐，语句可读。

```
28
29  /* 设备编号是每次请求必须携带的参数 */
30  if (is_null($this->deviceId))
31  {
32      throw new Exception('The deviceid can not be found in the request header.'
33          , ErrorCodeConstants::REQUEST_HEADER_INVALID);
34  }
35
36  /* 设备编号是每次请求必须携带的参数 */
37  if (is_null($this->deviceType))
38  {
39      throw new UserAuthException('The device type is not ios or android.'
40          , ErrorCodeConstants::REQUEST_HEADER_INVALID);
41  }
42
```

五、 循环、判断等语句中若有较长的表达式和语句，则要进行适当的划分。长表达式要在低优先级操作符处划分新行，操作符放在新行首。

```
44
45  if (
46      Yii::app()->session['uid'] != $uid            // 距离行首7个空格
47      || Yii::app()->session['deviceId'] != $deviceId   // 距离行首4个空格
48      || Yii::app()->session['deviceType'] != $deviceType
49  )
50  {
51      throw new UserAuthException('The request token\'s param is not equal to session value.'
52          , UserAuthException::USER_AUTH_REQUEST_TOKEN_INVALID);
53  }
54
```

六、 不允许把多个短语句放在一行，每行只能写一条语句。

七、 IF， FOR，FOREACH，WHILE，SWITCH，CASE，BREAK，DEFAULT 等语句自占一行，且 IF，FOR，FOREACH，WHILE，SWITCH 等语句的执行语句部分无论多少，都要加大括号{}。

```php

/* 下面的写法是不符合规范的 */
if ($condition === TRUE) return;

/* 下面的写法是符合规范的 */
if ($condition === TRUE)
{
    return;
}

```

八、 程序块的分界符（例如 { 和 }） 应独占一行，并且位于同一列，同时与引用它们的语句左对齐。在类的定义、方法（函数）的定义，以及 FOR、FOREACH、WHILE、DO、IF、SWITCH 语句中的程序都要采用如上的缩进方式。

```php

/* 下面的写法是不符合规范的 */
for (...) {

}

if (...)
    {

    }

/* 下面的写法是符合规范的 */
for (...)
{

}

if (...)
{
    
}

```

九、 在两个以上的关键字，变量，常量进行对等操作时，它们之间的操作符之前、之后或者前后都要空格；进行非对等操作时，如果是关系密切的立即操作符（例如 ->），应不加空格。

```php

/**
 * 比较操作符，赋值操作符，算术操作符，逻辑操作符，
 * 位域操作符等双目操作的前后要加空格
 */
if ($condition === TRUE)
$toalNumber = $numberFirst + $numSecond;
$totalNumber *= $numberThird;
$totalNumber = $numberFourth ^ 2;

/**
 * “！”，“++”，“--”，“&”等单目操作符前后不加空格
 */
$condition = !$condition;
$condition= &$condition;
--$totalNumber++;

/**
 * “->”，“.”前后不加空格
 */
$instance->property = $totalNumberl

/**
 * IF，FOR，FOREACH，WHILE，SWITCH等与后面括号间
 * 应加空格，以使关键字更加突出
 */
if ($condition === TRUE)
{

}

for ($i = 0; $i < 100; $i++)
{

}

```

注释

一、 一般情况下，源代码有效注释量须在 20%以上。注释的原则是帮助程序员对程序员的阅读理解，注释不宜太多或太少，注释语言必须准确、简洁、易懂。

二、 边写代码边注释，修改代码的同时应相应的修改注释，以保证代码和注释的一致性。不再有用的注释应该删除。

三、 避免在注释中使用缩写，特别是不常用的缩写。

四、 程序块注释使用“/**/”，但行代码注释使用“// “。使用单行注释 ” //“时，应距离被注释语句四个空格。

```
26
27  /* 我是程序块的注释，注意注释文字两边各有一个空格哦 */
28  if (...)
29  {
30      // do something
31  }
32
33  $condition = TRUE;    // 我是单行注释，与左边语句有四个空格
34
```

五、 注释应与所注释的代码在位置上相近，对代码的注释应放在其上方或者右方（单行注释），不可放在下方。如果注释放在上方，则需要与其上面的代码用空行隔开。

```
38
39  下面的写法是不符合规范的
40
41  /* 我是下面FOR语句的注释 */
42
43  foreach (...)
44  {
45
46  }
47  /* 我是上面FOR语句的注释 */
48
49  /* 我是下面IF的注释 */
50  if (...)
51  {
52
53  }
54  /* 我也是下面IF的注释 */
55  if (...)
56  {
57
58  }
59
60
61  下面的写法是符合规范的
62
63  /* 我是下面FOR语句的注释 */
64  foreach (...)
65  {
66
67  }
68
69  /* 我是下面IF的注释 */
70  if (...)
71  {
72
73  }
74
```

六、 每个 PHP 文件的头部都应该有注释，注释必须列出文件名称、作者、相关链接、版权，修改者，修改时间。

```
3
4   /**
5    * Controller class file.
6    *
7    * @author Zhang Jinlong <zjl@haoju.cn>
8    * @link http://api.haoju.cn
9    * @copyright Copyright &copy; 2014 AnHui WinStar Technology Co., Ltd
10   */
11
```

七、 类的属性必须要有注释，包含属性名称、属性类型。

```php

class HJJsonRPCAuthentication
{
    /**
     * 服务标识
     * @var string
     */
    private $_serviceId;
}

```

八、 类的方法和函数必须要有注释，包含方法（函数）名称，输入参数说明，返回值说明。

```php

class HJJsonRPCAuthentication
{
    /**
     * 根据IP地址找到对应的客户数据
     * @param string $ip
     * @return boolean
     */
    protected function getClients($ip)
    {
        $ip = trim($ip);

        $withParam = array(
            'clientIp' => array(
                'select' => FALSE,
                'joinType' => 'INNER JOIN',
                'condition' => 'ip = :ip',
                'params' => array(':ip' => $ip),
            ),
        );

        return TRUE;
    }
}

```

九、 对于所有有具体含义的变量、常量，如果其命名不是充分自注释的，在声明时必须加以注释，说明其具体含义。

十、 注释应与所描述的内容具有同样的缩进。

十一、　　对分支语句（条件分支，循环语句等），每个分支必须要有注释。

十二、　　通过对变量、常量、函数、类等正确的命名，以及合理的组织代码结构，是代码成为自注释的。

命名规范

一、 标识符的命名应该清晰明了，有明确含义，同时使用完整的单词或者大家基本可以理解的缩写，避免使人误解。

二、 类、方法、属性、函数、变量必须使用驼峰法。其中，类名首字母要大写，其他首字母要求小写。

```
110
111 class HJJsonRPCAuthentication
112 {
113     /**
114      * 服务标识
115      * @var string
116      */
117     private $_serviceId;
118
119     /**
120      * 根据IP地址找到对应的客户数据
121      * @param string $ip
122      * @return boolean
123      */
124     protected function getClients($ip)
125     {
126         $ip = trim($ip);
127
128         $withParam = array(
129             'clientIp' => array(
130                 'select' => FALSE,
131                 'joinType' => 'INNER JOIN',
132                 'condition' => 'ip = :ip',
133                 'params' => array(':ip' => $ip),
134             ),
135         );
136
137         return TRUE;
138     }
139 }
140
```

三、 类的属性为私有（private）属性时，以单个下划线开始。

```
110
111  class HJJsonRPCAuthentication
112  {
113      /**
114       * 服务标识
115       * @var string
116       */
117      private $_serviceId;
118  }
119
```

四、 常量命名要求全部大写，各单词间以下划线连接。

```
 9
10  class HJJsonRPCConstants
11  {
12      /**
13       * SECRET键值对分隔符
14       * @var string
15       */
16      const SECRET_SEPRATOR = ';';
17
18      /**
19       * 可用状态
20       * @var number
21       */
22      const AVAILABLE_YES = 1;
23
24      /**
25       * 资源需要验证权限
26       * @var number
27       */
28      const CHECK_PERMISSION_REQUIRED = 1;
29  }
```

五、 对于变量命名，禁止使用单个字符（例如：i，j，k），但是作为局部循环变量是允许的。

类，属性，方法

术语“类”指所有的类（class），接口（interface）和特性（trait）。

一、 扩展（extend）和实现（implement）

一个类的扩展（extend）和实现（implement）关键词必须和类名在同一行。类的左花括号必须放在下面自成一行；右花括号必须放在类主体的后面自成一行。

```
136
137  namespace Vendor\Package;
138
139  use FooClass;
140  use BarClass as Bar;
141  use OtherVendor\OtherPackage\BazClass;
142
143  class ClassName extends ParentClass implements \ArrayAccess
144  {
145      // 常量、属性、方法
146  }
147
```

实现（implement）列表可以被拆分为多个缩进了一次的子行。如果要拆成多个子行，列表的第一项必须要放在下一行，并且每行必须只有一个接口（interface）。

```
150
151  namespace Vendor\Package;
152
153  use FooClass;
154  use BarClass as Bar;
155  use OtherVendor\OtherPackage\BazClass;
156
157  class ClassName extends ParentClass implements
158      \ArrayAccess,
159      \Countable,
160      \Serializable
161  {
162      // 常量、属性、方法
163  }
164
```

二、 属性（property）

所有的属性都必须声明其可见性。变量“var”关键字不可用来声明一个属性。一条语句不可声明多个属性。一个属性声明看起来应该像下面这样：

```
167
168   namespace Vendor\Package;
169
170   class ClassName
171   {
172       /**
173        * 属性描述
174        * @var NULL
175        */
176       public $foo = NULL;
177   }
178
```

三、 方法（method）

所有的方法都必须声明其可见性。方法名不推荐用单个下划线作为前缀来表明其保护或私有的可见性。

方法名在其声明后面不可有空格跟随。其左花括号必须放在下面自成一行，且右花括号必须放在方法主体的下面自成一行。左括号后面不可有空格，且右括号前面也不可有空格。

一个方法声明看来应该像下面这样。 注意括号，逗号，空格和花括号的位置：

```
181
182   class ClassName
183   {
184       public function fooBarBaz($arg1, &$arg2, $arg3 = [])
185       {
186           // 方法主体部分
187       }
188   }
189
```

四、 方法的参数

在参数列表中，逗号之前不可有空格，而逗号之后则必须要有一个空格。方法中有默认值的参数必须放在参数列表的最后面。

```
class ClassName
{
    public function foo($arg1, &$arg2, $arg3 = [])
    {
        // 方法主体部分
    }
}
```

参数列表可以被拆分为多个缩进了一次的子行。如果要拆分成多个子行，参数列表的第一项必须放在下一行，并且每行必须只有一个参数。

```
class ClassName
{
    public function aVeryLongMethodName(
        ClassTypeHint $arg1,
        &$arg2,
        array $arg3 = []
    )
    {
        // 方法主体部分
    }
}
```

五、 抽象（abstract）终结（final）和 静态（static）

当用到抽象和终结来做类声明时，它们必须放在可见性声明的前面。

而当用到静态(static)来做类声明时，则必须放在可见性声明的后面。

```php

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

    final public static function bar()
    {
        // 方法主体部分
    }
}

```

六、 调用方法和函数

调用一个方法或函数时，在方法名或者函数名和左括号之间不可有空格，左括号之后不可有空格，右括号之前也不可有空格。参数列表中，逗号之前不可有空格，逗号之后则必须有一个空格。

```php

bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);

```

参数列表可以被拆分成多个缩进了一次的子行。如果拆分成子行，列表中的第一项必须放在下一行，并且每一行必须只能有一个参数。

```php

$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);

```

控制结构

在上面的排版中，我们已经对代码的版式做了具体的要求。但是控制结构是一个程序中最常遇到的结构之一，下面单独针对控制结构进行更详细的说明。

下面是对于控制结构代码风格的概括：

1. 控制结构的关键词之后必须有一个空格。
2. 控制结构的左括号之后不可有空格。
3. 控制结构的右括号之前不可有空格。
4. 控制结构的代码主体必须进行一次缩进。
5. 控制结构的花括号必须主体的下一行。

每个控制结构的代码主体必须被括在花括号里。这样可是使代码看上去更加标准化，并且加入新代码的时候还可以因此而减少引入错误的可能性。

一、 if，else if，else

下面是一个 if 条件控制结构的示例，注意其中括号，空格和花括号的位置。

```php

if ($expr1)
{
    // if body
}
else if ($expr2)
{
    // else if body
}
else
{
    // else body;
}

```

二、 switch，case

下面是一个 switch 条件控制结构的示例，注意其中括号，空格和花括号的位置。case 语句必须要缩进一级，而 break 关键字（或其他中止关键字） 必须和 case 结构的代码主体在同一个缩进层级。如果一个有主体代码的 case 结构故意的继续向下执行则必须要有一个类似于“// no break”的注释。

```php

switch ($expr)
{
    case 0:
        echo 'First case, with a break';
        break;

    case 1:
        echo 'Second case, which falls through';
        // no break
    
    case 2:
    case 3:
    case 4:
        echo 'Third case, return instead of break';
        return;

    default:
        echo 'Default case';
        break;
}

```

三、 while，do while

下面是一个 while 循环控制结构的示例，注意其中括号，空格和花括号的位置。

```
283
284  while·($expr)
285  {
286  ····//·structure·body
287  }
288
```

下面是一个 do while 循环控制结构的示例，注意其中括号，空格和花括号的位置。

```
291  do
292  {
293  ····//·structure·body;
294  }·while
295
```

四、 for

下面是一个 for 循环控制结构的示例，注意其中括号，空格和花括号的位置。

```
298
299  for·($i·=·0;·$i·<·10;·$i++)
300  {
301  ····//·for·body
302  }
303
```

五、 foreach

下面是一个 foreach 循环控制结构的示例，注意其中括号，空格和花括号的位置。

```
304
305 foreach ($iterable as $key => $value)
306 {
307     // foreach body
308 }
309
```

六、 try, catch

下面是一个 try catch 异常处理控制结构的示例，注意其中括号，空格和花括号的位置。

```
310
311 try
312 {
313     // try body
314 }
315 catch (FirstExceptionType $e)
316 {
317     // catch body
318 }
319 catch (OtherExceptionType $e)
320 {
321     // catch body
322 }
323
```