PHP 编码规范

基于 Lge_CodeSniffer 标准

目录

1. 引言

本规范由编程原则组成，融合并提炼了开发人员长时间积累下来的成熟经验，意在帮助形成良好一致的编程风格。以达到事半功倍的效果，如果有需要本文档会不定期更新。

本规范基于 PSR-2，对于少部分编写风格做了自定义调整，PSR-2 规范请参考：http://www.php-fig.org/psr/psr-2/

2. 标准化的重要性和好处

- 在一致的环境下，人们可以减少犯错的机会；
- 新加入项目的开发人员可以很快的适应环境；
- 代码易于阅读和维护，开发人员可以快速了解代码，弄清程序状况；
- 防止新接触PHP的人出于节省时间的需要，自创一套风格并养成终生的习惯；

当一个软件项目尝试着遵守公共一致的标准时，可以使参与项目的开发人员更容易了解项目中的代码、弄清程序的状况。使新的参与者可以很快的适应环境，防止部分参与者出于节省时间的需要，自创一套风格并养成终生的习惯，导致其它人在阅读时浪费过多的时间和精力。而且在一致的环境下，也可以减少编码出错的机会。缺陷是由于每个人的标准不同，所以需要一段时间来适应和改变自己的编码风格，暂时性的降底了工作效率。从使项目长远健康的发展以及后期更高的团队工作效率来考虑暂时的工作效率降低是值得的，也是必须要经过的一个过程。标准不是项目成功的关键，但可以帮助我们在团队协作中有更高的效率并且更加顺利的完成既定的任务。

3. 代码标签

PHP 代码标签可以使用或来界定 PHP 代码， PHP 代码标签统一使用<?php ?>这样的完整的封闭式标签形式，不建议使用短标签例如<? ?>、<% %>这样的形式。如果是纯 PHP 文件那么建议不闭合标签，以避免文件末尾包含空格或者空行(?>末尾的空行或者空行)时，文件被包含时这些空行或者换行会被展示出来，有时会引起意想不到的错误(特别是使用 header 函数时)。

4. 注释

注释采用标准的 phpDocumentor 注释方式，phpDocumentor 官方网站地址：http://www.phpdoc.org

另外需要强调的是，

1. 开发人员对自己建立的PHP文件顶端写上文件注释，包括：文件功能、维护人员(@author)、修改记录；
2. 类方法、函数必须注释写上完整的输入和返回值说明(@param, @return)；
3. 配置变量，特别是全局的配置变量必须要写上变量说明；

5. 命名原则

命名是程序规划的核心。古人相信只要知道一个人真正的名字就会获得凌驾于那个人之上的不可思议的力量。
只要你给事物想到正确的名字，就会给你以及后来的人带来比代码更强的力量。名字就是事物在它所处的生态环境中一个长久而深远的结果。总的来说，只有了解系统的程序员才能为系统取出最合适的名字。如果所有的命名都与其自然相适合，则关系清晰，含义可以推导得出，一般人的推想也能在意料之中。
就一般约定而言，类、函数和变量的名字应该总是能够描述让代码阅读者能够容易的知道这些代码的作用。形式越简单、越有规则，就越容易让人感知和理解。应该避免使用模棱两可，晦涩不标准的命名。

5.1. 变量、函数、类对象、类成员名

5.1.1. 变量名

变量名均采用英文字母大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写(驼峰式)。
变量名应简短且富于描述、易于记忆，即，能够指出其用途。尽量避免单个字符的变量名，除非是一次性的临时变量。
例如：
$isMember
$userDetail

5.1.2. 函数名

方法名是一个动词，采用大小写混合的方式，第一个单词的首字母小写，其后单词的首字母大写(驼峰式)。
function run();
function runFast();
function getBackground();

通常每个方法都是执行一个动作的，所以对它的命名应该清楚的说明它是做什么的：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。

另外(供参考的命名习惯)：
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
getUserInfo
setNickName

5.1.3. 类及实例

类名称由英文单词组成，且所有英文单词首字母大写。
例如：

```
class User{ }
class Account{ }
class OlineInfo{ }
```

实例(object)命名规范同变量命名规范。
例如：

```
$user = new User();
$account = new Account();
```

5.1.4. 类成员

类成员都必须使用关键字 public、protected 或 private 进行定义。
类成员命名根据其作用域不同，命名规则不同。private / protected 成员、以下划线“_”为前缀，这样能更好地标示作用域，其他同变量命名。
public 成员无需前缀，同变量命名。

例如：

```
class NameOneTwo
{
    private $_varAbc;
    protected $_errorNumber;
    public $memberID;
}
```

5.1.5. 类方法

类中所有方法命名规范如下，
public 方法同函数命名规范。
private / protected 方法名在函数命名规范基础上，都要加下划线’_’前缀。对于维护一个大型项目时(特别是代码重构)，一个类方法是否可以删除或者修改定义，如果有一个“_”前缀的规范标示私有或者保护方法，将会显得特别有益处。
例如：

```
public function getUserName($userid)
private function _parseIsoToDate($token, $locale)
protected function _toToken($part, $locale)
```

5.1.6. 缩写词

缩写词全部使用大写字母，以标示该词为缩写词。
使用: getHTMLStatistic.
不使用: getHtmlStatistic.

5.1.7. 全局变量

全局变量所有的单词首字母大写，以便于和局部变量进行区分。
例如：

```
$Log;
```

```
$SystemConfig;
```

5.2. 常量

常量应该总是全部使用大写字母命名，少数特别必要的情况下，可使用下划线来分隔单词。
如：ROOTPATH、IMGROOT、DB_HOST、DB_PASS、DB_USER、DB_PORT

5.3. 目录及文件命名

可能在 URL 中出现的目录以及文件名称尽量采用小写字母，长度一般不超过 20 个字符，命名采用小写字母，因为 Linux 的文件系统是大小写敏感的，将文件名称统一成小写，有利于提高可靠性。

类文件命名格式，ClassName.class.php，ClassName 所有单词首字母大写。

6. 书写规则

6.1. 项目编码

项目编码统一使用 UTF-8 编码防止文字乱码以及编码不统一对项目造成的错误.

6.2. 变量的初始化与逻辑检查

任何变量在进行累加、直接显示或存储前必需进行初使化，例如：

```
$number = 0; //  数值型初始化
$string = ''; //  字符串初始化
$array = array(); //  数组初始化
```

类成员都必须使用关键字 public、protected 或 private 进行定义，推荐一行定义一个类成员，因为这样以利于写注释。亦即，

```
public $level; //  缩进的程度
public $size; //  由制表符决定
```

要优于，

```
public $level, $size;
```

6.3. 缩进

每个缩进的单位约定是一个 TAB(4 个空白字符宽度)，需每个参与项目的开发人员在编辑器(UltraEdit、EditPlus、Zend Studio 等)中进行强制设定，以防在编写代码时遗忘而造成格式上的不规范。
本缩进规范适用于 PHP、JavaScript 中的函数、类、逻辑结构、循环等。

6.4. 大括号{}、if和switch

if 结构中，if 和 elseif 与前后两个圆括号同行，左右各一个空格，所有的大括号与关键词同行，尾括号与关键字同列。另外，即便 if 后只有一行语句，仍然需要加入大括号，以保证结构清晰(紧凑型编码风格)。
避免使用如下容易引起错误的格式：

```
if (condition) //避免这种写法，他忽略了“{ }”
statement;
```

switch 结构中，通常当一个 case 块处理后，将跳过之后的 case 块处理，因此大多数情况下需要添加 break。break 的位置视程序逻辑，与 case 同在一行，或新起一行均可，但同一 switch 体中，break 的位置格式应当保持一致。
以下是符合上述规范的例子：

```
if ($condition) {
    switch ($var) {
        case 1: echo 'var is 1'; break;
        case 2: echo 'var is 2'; break;
        default: echo 'var is neither 1 or 2'; break;
    }
} else {
    switch ($str) {
        case 'abc':
            $result = 'abc';
            break;
        default:
            $result = 'unknown';
            break;
    }
}
```

6.5. for语句

一个 for 语句应该具有如下格式：

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

一个空的 for 语句(所有工作都在初始化，条件判断，更新子句中完成)应该具有如下格式：

```
for (initialization; condition; update);
```

当在 for 语句的初始化或更新子句中使用逗号时，避免因使用三个以上变量，而导致复杂度提高。若需要，可以在 for 循环之前(为初始化子句)或 for 循环末尾(为更新子句)使用单独的语句。

6.6. while语句

一个 while 语句应该具有如下格式

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

一个空的 while 语句应该具有如下格式：

```
while (condition);
```

6.7. 运算符、小括号、空格、关键词和函数

每个运算符与两边参与运算的值或表达式中间要有一个空格，唯一的特例是字符连接运算符号两边不加空格；
左括号“(” 应和函数关键词紧贴在一起，除此以外应当使用空格将“(”同前面内容分开；
右括号“)”除后面是“)”或者“;”以外，其他一律用空格隔开它们；
除字符串中特意需要，一般情况下，在程序以及 HTML 中不出现两个连续的空格；
任何情况下，PHP 程序中不能出现空白的带有 TAB 或空格的行，即：这类空白行应当不包含任何 TAB 或空格。同时，任何程序行尾也不能出现多余的 TAB 或空格。多数编辑器具有自动去除行尾空格的功能，如果习惯养成不好，可临时使用它，避免多余空格产生；
每段较大的程序体，上、下应当加入空白行，两个程序块之间只使用 1 个空行，禁止使用多行。
程序块划分尽量合理，过大或者过小的分割都会影响他人对代码的阅读和理解。一般可以以较大函数定义、逻辑结构、功能结构来进行划分。少于 15 行的程序块，可不加上下空白行；
说明或显示部分中，内容如含有中文、数字、英文单词混杂，应当在数字或者英文单词的前后加入空格。

根据上述原则，以下举例说明正确的书写格式：

```
$result = (($a + 1) * 3 / 2 + $num)).'Test';
$condition ? func1($var) : func2($var);
$condition ? $long_statement : $another_long_statement;
if ($flag) {
    // Statements
    // More than 15 lines
}
showMessage('Successfully!');
```

6.8. 函数定义

- 函数定义中的左小括号，与函数名紧挨，中间无需空格；
- 开始的左大括号与函数在同一行；
- 具有默认值的参数应该位于参数列表的后面；
- 函数调用与定义的时候参数与参数之间加入一个空格；
- 必须仔细检查并切实杜绝函数起始缩进位置与结束缩进位置不同的现象。

例如，符合标准的定义：

```
function authcode($string, $operation, $key = '') {
    If ($flag) {
        // Statements
    }
    // Statements
}
```

不符合标准的定义：

```
function authcode($string,$operation,$key = '')
{
    // Statements
}
```

另外，确保函数的返回值类型是与定义一样，比如：

```
function getArray() {
    $returnArray = array();
    // Statements
    return $returnArray;
}
```

一个函数返回值应该是一个数组类型的值，但是在定义部分有可能使返回值的类型发生改变或者说不能确保返回值的类型是预期的，那么我们可以在返回值的时候做一次强制类型转换。

```
function getArray() {
    $returnArray = array();
    // Statements
    return (array)$returnArray;
}
```

在能够确保返回值类型为预期格式的情况下，尽量无需这么做，各位在编写函数或者类方法的时候注意一下该情况。

6.9. 引号

PHP 中单引号和双引号具有不同的含义，最大的几项区别如下：

单引号中，任何变量($var)、特殊转义字符(如“\t \r \n”等)不会被解析，因此 PHP 的解析速度更快，转义字符仅仅支持“\’”和“\\”这样对单引号和反斜杠本身的转义；

双引号中，变量($var)值会代入字符串中，特殊转义字符也会被解析成特定的单个字符，还有一些专门针对上述两项特性的特殊功能性转义，例如“\$”和“{$array[‘key’]}”。这样虽然程序编写更加方便，但同时 PHP 的解析也很慢；

数组中，如果下标不是整型，而是字符串类型，请务必用单引号将下标括起，正确的写法为$array[‘key’]，而不是$array[key]，因为不正确的写法会使 PHP 解析器认为 key 是一个常量，进而先判断常量是否存在，不存在时才以“key”作为下标带入表达式中，同时出发错误事件，产生一条 Notice 级错误。

因此，在绝大多数可以使用单引号的场合，禁止使用双引号。依据上述分析，可以或必须使用单引号的情况包括但不限于下述：

- 字符串为固定值，不包含“\t”等特殊转义字符；
- 数组的固定下标，例如$array[‘key’]；
- 表达式中不需要带入变量，例如$string = ‘test’;，而非$string = “test{$var}”；

例外的，在正则表达式(用于 preg_系列函数和 ereg 系列函数)中，全部使用双引号，这是为了人工分析和编写的方便，并保持正则表达式的统一，减少不必要的分析混淆。

数据库 SQL 语句中，所有数字数据都不得加单引号，但是在进行 sql 查询之前都必须经过 intval 或者 floatval 函数处理；所有字符串都必须加单引号，以避免可能的注入漏洞和 SQL 错误。正确的写法为：

```
$catid = intval($catid);
"SELECT * FROM member WHERE `username` = ’{$_username}’ AND `catid` = {$catid}; "
```

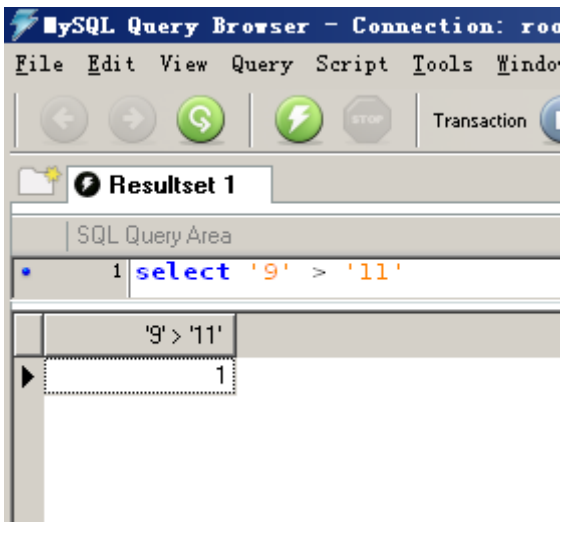
在数字类型的字段比较中(例如排序),禁止使用单引号，因为数据库的字符串比较与数字比较方式是不一样的，例如:

SELET * from files where `size` > '1024' - 这样的语句就是错误的，其中 size 字段是 double 类型的字段，表示文件大小。

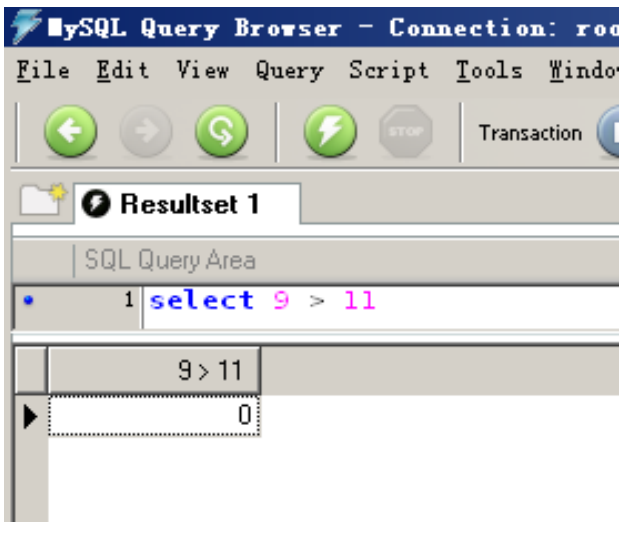
应该改为 SELET * from files where `size` > 1024

可以在 SQL 中执行以下语句检测异同:

SELECT ‘9’ > ‘11’ != SELECT 9 > 11



VS



另外，所有数据在插入数据库之前，均需要进行转义处理，以免特殊字符未经转义在插入数据库的时候出现错误，可以对变量使用 addslashes() 或者 mysql_escape_string() 进行。关于 addslashes()函数需要注意的是，对于外部文件提交的 GET, POST, COOKIE 数组值，应判断是否服务器运行环境中是否开启了魔法引用，对于需要加入 SQL 语句中的变量应采用以下的形式进行处理，防止双重转义:

```
$str = get_magic_quotes_gpc() ? $str: addslashes($str);
```

我们可以定义一个公共的函数对 GET, POST, COOKIE 的数据进行检测处理，例如：

```
function dealSlashes($str) {
    return get_magic_quotes_gpc() ? $str: addslashes($str);
}
```