|  |
| --- |
| 南京智精灵教育科技有限公司 |
| [编码规范] |
| [PHP版本] |
|  |
| **王轶平** |
| **11/11 Wednesday** |

|  |
| --- |
| [基于 PHP互操作性框架制定小组（PHP-FIG :PHP Framework Interoperability Group）制定的PHP编码规范] |

目录

[RFC 2119 4](#_Toc435794537)

[1.MUST 必须/需要/将会 4](#_Toc435794538)

[2.MUST NOT一定不可/一定不能/不会 4](#_Toc435794539)

[3.SHOULD应该/推荐 4](#_Toc435794540)

[4.SHOULD NOT不该 4](#_Toc435794541)

[5.MAY可以/可选 4](#_Toc435794542)

[基本代码规范PSR-1 5](#_Toc435794543)

[1. 概览 5](#_Toc435794544)

[2. 文件 5](#_Toc435794545)

[2.1. PHP标签 5](#_Toc435794546)

[2.2. 字符编码 6](#_Toc435794547)

[2.3. 从属效应（副作用） 6](#_Toc435794548)

[3. 命名空间和类 7](#_Toc435794549)

[4. 类的常量、属性和方法 8](#_Toc435794550)

[4.1. 常量 8](#_Toc435794551)

[4.2. 属性 8](#_Toc435794552)

[4.3. 方法 9](#_Toc435794553)

[代码风格规范PSR-2 10](#_Toc435794554)

[1. 概览 10](#_Toc435794555)

[1.1. 例子 11](#_Toc435794556)

[2. 通则 11](#_Toc435794557)

[2.1 基本编码准则 12](#_Toc435794558)

[2.2 文件 12](#_Toc435794559)

[2.3. 行 12](#_Toc435794560)

[2.4. 缩进 12](#_Toc435794561)

[2.5. 关键字 以及 True/False/Null 13](#_Toc435794562)

[3. namespace 以及 use 声明 13](#_Toc435794563)

[3.1 - 多行参数 13](#_Toc435794564)

[4. 类、属性和方法 14](#_Toc435794565)

[4.1. 扩展与继承 14](#_Toc435794566)

[4.2. 属性 15](#_Toc435794567)

[4.3. 方法 15](#_Toc435794568)

[4.4. 方法的参数 16](#_Toc435794569)

[4.5. abstract 、 final 、 以及 static 17](#_Toc435794570)

[4.6. 方法及函数调用 18](#_Toc435794571)

[5. 控制结构 18](#_Toc435794572)

[5.1. if 、 elseif 和 else 18](#_Toc435794573)

[5.2. switch 和 case 19](#_Toc435794574)

[5.3. while 和 do while 20](#_Toc435794575)

[5.4. for 20](#_Toc435794576)

[5.5. foreach 20](#_Toc435794577)

[5.6. try, catch 21](#_Toc435794578)

[6. 闭包 21](#_Toc435794579)

[7. 注释 24](#_Toc435794580)

[单行注释 24](#_Toc435794581)

[单行注释块 24](#_Toc435794582)

[多行注释块 24](#_Toc435794583)

[临时屏蔽代码 24](#_Toc435794584)

[8. 总结 25](#_Toc435794585)

[Autoloader PSR-4 27](#_Toc435794586)

[1. 概述 27](#_Toc435794587)

[2. 详细说明 27](#_Toc435794588)

[3. 例子 28](#_Toc435794589)

[日志接口规范 PSR-3 29](#_Toc435794590)

[1. 规范说明 29](#_Toc435794591)

[1.1 基本规范 29](#_Toc435794592)

[1.2 记录信息 29](#_Toc435794593)

[1.3 上下文 30](#_Toc435794594)

[1.4 助手类和接口 31](#_Toc435794595)

[2. 包 31](#_Toc435794596)

[3. Psr\Log\LoggerInterface 31](#_Toc435794597)

[4. Psr\Log\LoggerAwareInterface 34](#_Toc435794598)

[5. Psr\Log\LogLevel 34](#_Toc435794599)

# RFC 2119

## 1.MUST 必须/需要/将会

这个关键字，或是术语"REQUIRED"或"SHALL"，意味着他们的定义是一个绝对的规范的必要条件。

## 2.MUST NOT一定不可/一定不能/不会

这个词组，或是词组"SHALL NOT"，意味着他们的定义是一个绝对的规范的禁止的条件。

## 3.SHOULD应该/推荐

这个关键字，或是形容词" RECOMMENDED "，意味着在特殊的环境下可能存在正当的理由忽略一个特殊的项目，但是完整的含义必须能被理解并且要仔细的考虑在重新选择一条不同的路径之前。

## 4.SHOULD NOT不该

这个词组，或是词组"NOT RECOMMENDED"，意味着在特殊的环境下可能存在正当的理由当特殊的行为可接受或是甚至有效，但是完整的含义应该被理解并且这种情况应该被仔细考虑当暗含某种行为用标签描述前。

## 5.MAY可以/可选

这个关键字，或是形容词"OPTIONAL"，意味着一个项目是真正可选的。一个买主可以选择包含项目因为一个特殊的市场要求或是因为买主觉得它能够增强产品当另一个买主可能遗漏了同样的项目时。一个没有包含一个特殊对象MUST的工具准备用来与另一个不包含这个对象的工具间相互起作用，因此或许有简化的功能。在同样的脉络里一个包含了特殊对象MUST的工具准备用来和另一个没有包含这个对象的工具间相互作用。

# 基本代码规范PSR-1

本篇规范制定了代码基本元素的相关标准， 以确保共享的PHP代码间具有较高程度的技术互通性。

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 RFC 2119 。

## 1. 概览

PHP代码文件必须以 <?php 或 <?= 标签开始；

PHP代码文件必须以 不带BOM的 UTF-8 编码；

PHP代码中应该只定义类、函数、常量等声明，或其他会产生 从属效应 的操作（如：生成文件输出以及修改.ini配置文件等），二者只能选其一；

命名空间以及类必须符合 PSR 的自动加载规范： PSR-4（即本文Autoload部分）；

类的命名必须遵循 StudlyCaps 大写开头的驼峰命名规范；

类中的常量所有字母都必须大写，单词间用下划线分隔；

方法名称必须符合 camelCase 式的小写开头驼峰命名规范。

## 2. 文件

### 2.1. PHP标签

PHP代码必须使用 <?php ?> 长标签 或 <?= ?> 短输出标签； 一定不可使用其它自定义标签。

<? 短标签（非短输出标签）一定不可使用。

### 2.2. 字符编码

PHP代码必须且只可使用不带BOM的UTF-8编码。

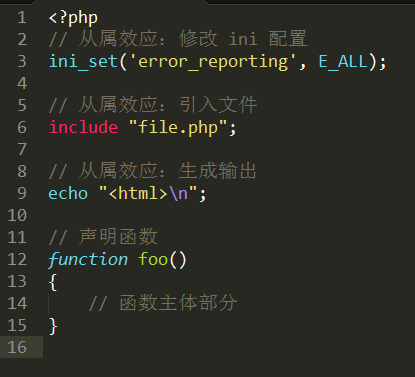
### 2.3. 从属效应（副作用）

一份PHP文件中应该要不就只定义新的声明，如类、函数或常量等不产生从属效应的操作，要不就只有会产生从属效应的逻辑操作，但不该同时具有两者。

“从属效应”(side effects)一词的意思是，仅仅通过包含文件，不直接声明类、 函数和常量等，而执行的逻辑操作。

“从属效应”包含却不仅限于：生成输出、直接的 require 或 include、连接外部服务、修改 ini 配置、抛出错误或异常、修改全局或静态变量、读或写文件等。

以下是一个反例，一份包含声明以及产生从属效应的代码：



下面是一个范例，一份只包含声明不产生从属效应的代码：



## 3. 命名空间和类

命名空间以及类的命名必须遵循 PSR-4.

根据规范，每个类都独立为一个文件，且命名空间至少有一个层次：顶级的组织名称（vendor name）。

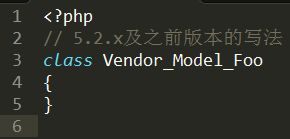
类的命名必须 遵循 StudlyCaps 大写开头的驼峰命名规范。

PHP 5.3及以后版本的代码必须使用正式的命名空间。

例如：



5.2.x及之前的版本应该使用伪命名空间的写法，约定俗成使用顶级的组织名称（vendor name）如 Vendor\_ 为类前缀。

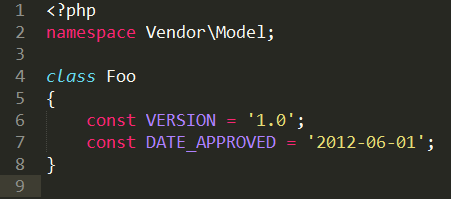


## 4. 类的常量、属性和方法

此处的“类”指代所有的类、接口以及可复用代码块（traits）

### 4.1. 常量

类的常量中所有字母都必须大写，词间以下划线分隔。 参照以下代码：



### 4.2. 属性

~~类的属性命名可以遵循 大写开头的驼峰式 ($StudlyCaps)、小写开头的驼峰式 ($camelCase) 又或者是 下划线分隔式 ($under\_score)，本规范不做强制要求，但无论遵循哪种命名方式，都应该在一定的范围内保持一致。这个范围可以是整个团队、整个包、整个类或整个方法。~~

类的属性命名必须使用小写开头的驼峰式 ($camelCase)。

### 4.3. 方法

方法名称必须符合 camelCase() 式的小写开头驼峰命名规范。

# 代码风格规范PSR-2

本篇规范是 PSR-1 基本代码规范的继承与扩展。

本规范希望通过制定一系列规范化PHP代码的规则，以减少在浏览不同作者的代码时，因代码风格的不同而造成不便。

当多名程序员在多个项目中合作时，就需要一个共同的编码规范， 而本文中的风格规范源自于多个不同项目代码风格的共同特性， 因此，本规范的价值在于我们都遵循这个编码风格，而不是在于它本身。

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 RFC 2119 。

## 1. 概览

代码必须遵循 PSR-1 中的编码规范 。

代码必须使用4个空格符而不是 tab键 进行缩进。

每行的字符数应该软性保持在80个之内， 理论上一定不可多于120个， 但一定不能有硬性限制。

每个 namespace 命名空间声明语句和 use 声明语句块后面，必须插入一个空白行。

类的开始花括号 { 必须写在函数声明后自成一行，结束花括号 } 也必须写在函数主体后自成一行。

方法的开始花括号 { 必须写在函数声明后自成一行，结束花括号 } 也必须写在函数主体后自成一行。

类的属性和方法必须添加访问修饰符（private、protected 以及 public）， abstract 以及 final 必须声明在访问修饰符之前，而 static 必须声明在访问修饰符之后。

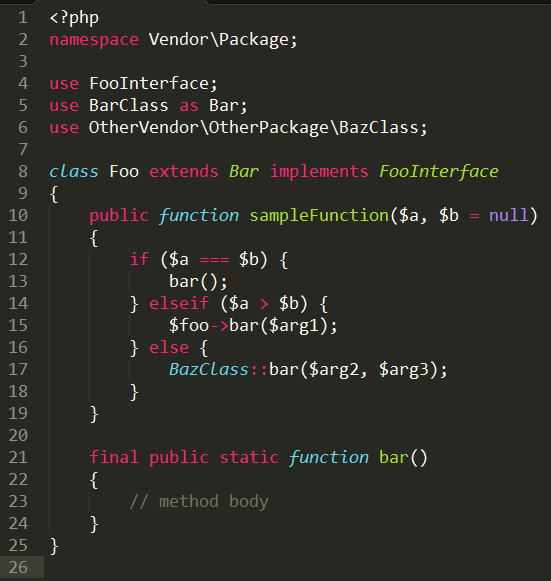
控制结构的关键字后必须要有一个空格符，而调用方法或函数时则一定不能有。

控制结构的开始花括号 { 必须写在声明的同一行，而结束花括号 } 必须写在主体后自成一行。

控制结构的开始左括号后和结束右括号前，都一定不能有空格符。

### 1.1. 例子

以下例子程序简单地展示了以上大部分规范：



## 2. 通则

### 2.1 基本编码准则

代码必须符合 PSR-1 中的所有规范。

### 2.2 文件

所有PHP文件必须使用Unix LF (linefeed)作为行的结束符。

所有PHP文件必须以一个空白行作为结束。

纯PHP代码文件必须省略最后的 ?> 结束标签。

### 2.3. 行

行的长度一定不能有硬性的约束。

软性的长度约束一定要限制在120个字符以内，若超过此长度，带代码规范检查的编辑器一定要发出警告，不过一定不可发出错误提示。

每行不应该多于80个字符，大于80字符的行应该折成多行。

非空行后一定不能有多余的空格符。

空行可以使得阅读代码更加方便以及有助于代码的分块。

每行一定不能存在多于一条语句。

### 2.4. 缩进

代码必须使用4个空格符的缩进，一定不能用 tab键 。

备注: 使用空格而不是tab键缩进的好处在于， 避免在比较代码差异、打补丁、重阅代码以及注释时产生混淆。 并且，使用空格缩进，让对齐变得更方便。

### 2.5. 关键字 以及 True/False/Null

PHP所有 关键字必须全部小写。

常量 true 、false 和 null 也必须全部小写。

## 3. namespace 以及 use 声明

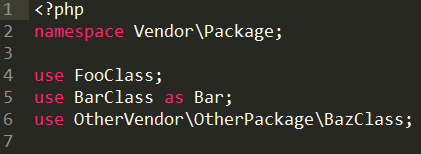
namespace 声明后 必须 插入一个空白行。

所有 use 必须 在 namespace 后声明。

每条 use 声明语句 必须 只有一个 use 关键词。

use 声明语句块后 必须 要有一个空白行。

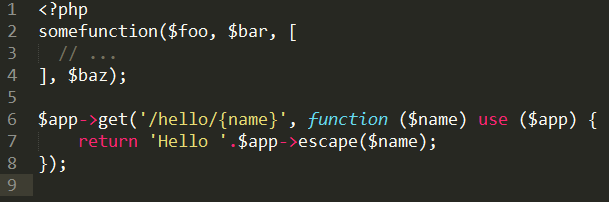
例如：



### 3.1 - 多行参数

使用一个或多个跨行的参数（如数组和匿名函数）并不需要触发 4.6 节中关于参数列表的单行规定， 因此，在参数表中的数组和匿名函数是可以单独分列成多行的。

以下的例子是符合 PSR-2 规范的：



当需要扩展多个接口时，extends 的相关规范与 4.1 节中 implements 的规范一致。

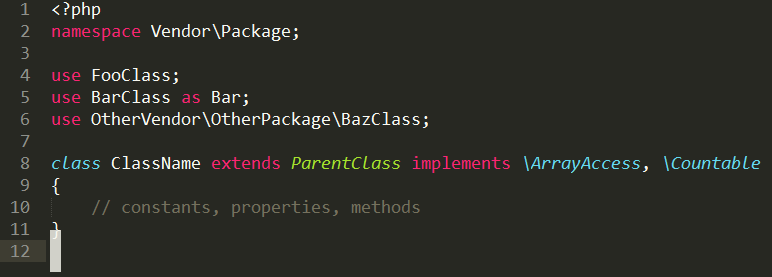
## 4. 类、属性和方法

此处的“类”泛指所有的class类、接口以及traits可复用代码块。

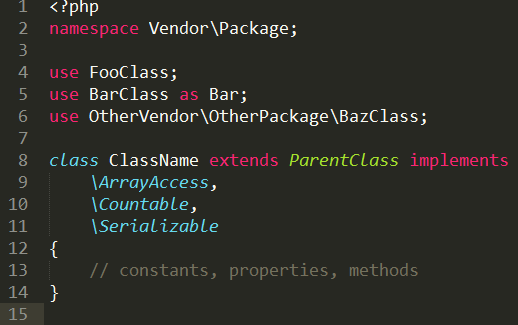
### 4.1. 扩展与继承

关键词 extends 和 implements必须写在类名称的同一行。

类的开始花括号必须独占一行，结束花括号也必须在类主体后独占一行。



implements 的继承列表也可以分成多行，这样的话，每个继承接口名称都必须分开独立成行，包括第一个。



### 4.2. 属性

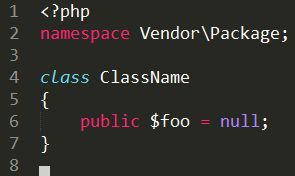
每个属性都必须添加访问修饰符。

一定不可使用关键字 var 声明一个属性。

每条语句一定不可定义超过一个属性。

不要使用下划线作为前缀，来区分属性是 protected 或 private。

以下是属性声明的一个范例：



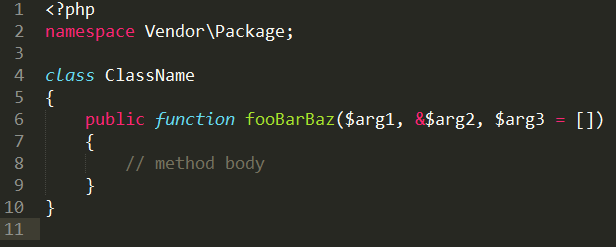
### 4.3. 方法

所有方法都必须添加访问修饰符。

不要使用下划线作为前缀，来区分方法是 protected 或 private。

方法名称后一定不能有空格符，其开始花括号必须独占一行，结束花括号也必须在方法主体后单独成一行。参数左括号后和右括号前一定不能有空格。

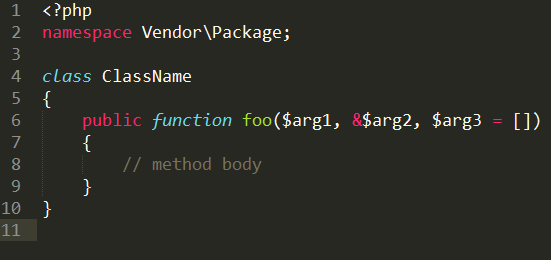
一个标准的方法声明可参照以下范例，留意其括号、逗号、空格以及花括号的位置。



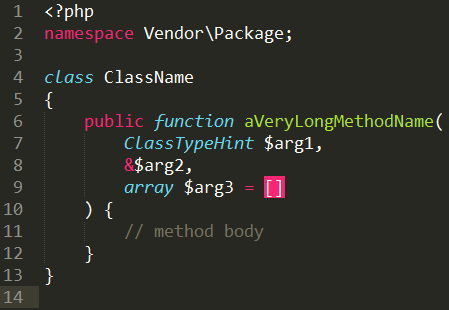
### 4.4. 方法的参数

参数列表中，每个逗号后面必须要有一个空格，而逗号前面一定不能有空格。

有默认值的参数，必须放到参数列表的末尾。

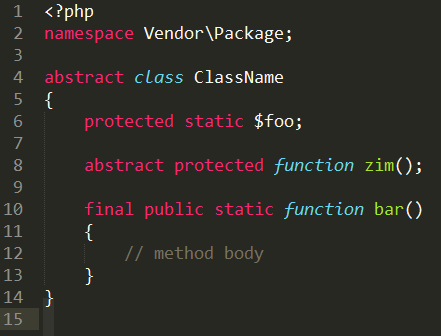
参数列表可以分列成多行，这样，包括第一个参数在内的每个参数都必须单独成行。

拆分成多行的参数列表后，结束括号以及方法开始花括号 必须 写在同一行，中间用一个空格分隔。



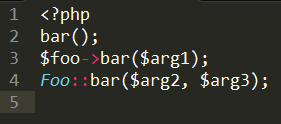
### 4.5. abstract 、 final 、 以及 static

需要添加 abstract 或 final 声明时， 必须写在访问修饰符前，而 static 则必须写在其后。

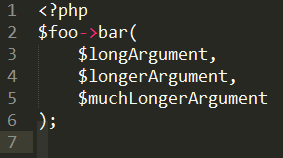


### 4.6. 方法及函数调用

方法及函数调用时，方法名或函数名与参数左括号之间一定不能有空格，参数右括号前也 一定不能有空格。每个参数前一定不能有空格，但其后必须有一个空格。



参数可以分列成多行，此时包括第一个参数在内的每个参数都必须单独成行。



## 5. 控制结构

控制结构的基本规范如下：

控制结构关键词后必须有一个空格。

左括号 ( 后一定不能有空格。

右括号 ) 前也一定不能有空格。

右括号 ) 与开始花括号 { 间一定有一个空格。

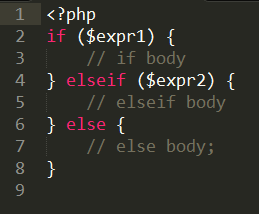
结构体主体一定要有一次缩进。

结束花括号 } 一定在结构体主体后单独成行。

每个结构体的主体都必须被包含在成对的花括号之中， 这能让结构体更加结构话，以及减少加入新行时，出错的可能性。

### 5.1. if 、 elseif 和 else

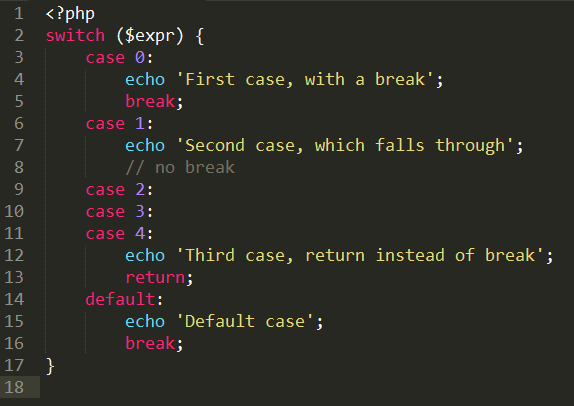
标准的 if 结构如下代码所示，留意 括号、空格以及花括号的位置， 注意 else 和 elseif 都与前面的结束花括号在同一行。



应该使用关键词 elseif 代替所有 else if ，以使得所有的控制关键字都像是单独的一个词。

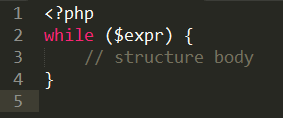
### 5.2. switch 和 case

标准的 switch 结构如下代码所示，留意括号、空格以及花括号的位置。 case 语句必须相对 switch 进行一次缩进，而 break 语句以及 case 内的其它语句都 必须 相对 case 进行一次缩进。 如果存在非空的 case 直穿语句，主体里必须有类似 // no break 的注释。

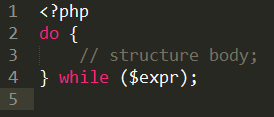


### 5.3. while 和 do while

一个规范的 while 语句应该如下所示，注意其 括号、空格以及花括号的位置。

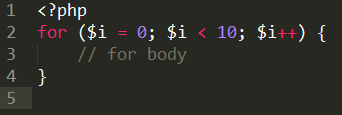


标准的 do while 语句如下所示，同样的，注意其 括号、空格以及花括号的位置。



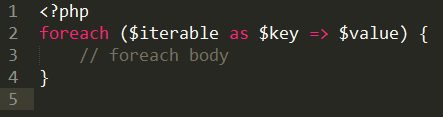
### 5.4. for

标准的 for 语句如下所示，注意其 括号、空格以及花括号的位置。



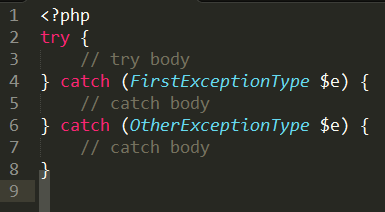
### 5.5. foreach

标准的 foreach 语句如下所示，注意其 括号、空格以及花括号的位置。



### 5.6. try, catch

标准的 try catch 语句如下所示，注意其 括号、空格以及花括号的位置。



## 6. 闭包

闭包声明时，关键词 function 后以及关键词 use 的前后都必须要有一个空格。

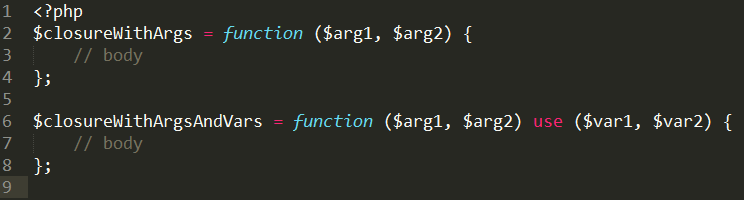
开始花括号必须写在声明的同一行，结束花括号必须紧跟主体结束的下一行。

参数列表和变量列表的左括号后以及右括号前，必须不能有空格。

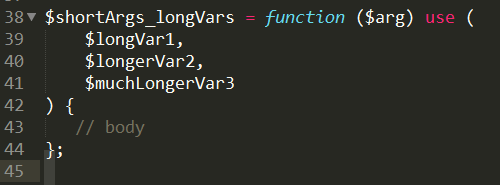
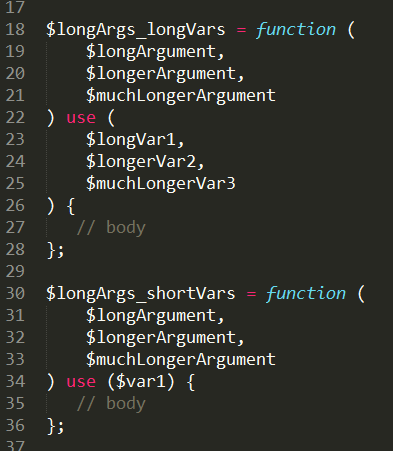
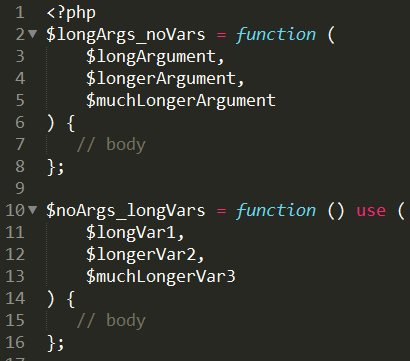
参数和变量列表中，逗号前必须不能有空格，而逗号后必须要有空格。

闭包中有默认值的参数必须放到列表的后面。

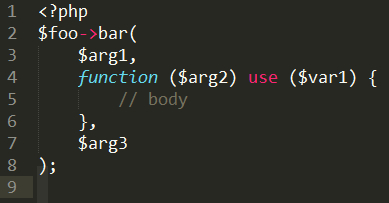
标准的闭包声明语句如下所示，注意其 括号、逗号、空格以及花括号的位置。

参数列表以及变量列表可以分成多行，这样，包括第一个在内的每个参数或变量都必须单独成行，而列表的右括号与闭包的开始花括号必须放在同一行。

以下几个例子，包含了参数和变量列表被分成多行的多情况。



注意，闭包被直接用作函数或方法调用的参数时，以上规则仍然适用。



## 7. 注释

不要注释坏代码——重写吧。

基于phpDocument的标准编写，详细内容参考 <http://www.phpdoc.org/docs/latest/index.html>

常用方法如下：

### 单行注释

//注释内容

必须在需要注释代码的上方。

### 单行注释块

/\*\*注释内容\*/

用于类的属性说明，必须在需要注释代码的上方。

### 多行注释块

/\*\*

\* 注释内容

\*/

用于命名空间、类、方法、函数的说明，必须在需要注释代码的上方。

块注释开始的/必须与类、方法、函数的第一个字符对齐，其下每行开头的\*，与第一行的\*对齐，然后空一格输入内容。

### 临时屏蔽代码

采用/\*……\*/注释块。并在/\*的下方采用单行注释说明屏蔽的时间、屏蔽人、原因。



## 8. 总结

以上规范难免有疏忽，其中包括但不仅限于：

全局变量和常量的定义

函数的定义

操作符和赋值

行内对齐

注释和文档描述块

类名的前缀及后缀

最佳实践

本规范之后的修订与扩展将弥补以上不足。

# Autoloader PSR-4

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 RFC 2119 。

## 1. 概述

本 PSR 是关于由文件路径 自动载入 对应类的相关规范， 本规范是可互操作的，可以作为任一自动载入规范的补充，其中包括 PSR-0，此外， 本 PSR 还包括自动载入的类对应的文件存放路径规范。

## 2. 详细说明

此处的“类”泛指所有的class类、接口、traits可复用代码块以及其它类似结构。

一个完整的类名需具有以下结构:

\<命名空间>(\<子命名空间>)\*\<类名>

完整的类名必须要有一个顶级命名空间，被称为 "vendor namespace"；

完整的类名可以有一个或多个子命名空间；

完整的类名必须有一个最终的类名；

完整的类名中任意一部分中的下滑线都是没有特殊含义的；

完整的类名可以由任意大小写字母组成；

所有类名都必须是大小写敏感的。

当根据完整的类名载入相应的文件……

完整的类名中，去掉最前面的命名空间分隔符，前面连续的一个或多个命名空间和子命名空间，作为“命名空间前缀”，其必须与至少一个“文件基目录”相对应；

紧接命名空间前缀后的子命名空间必须与相应的”文件基目录“相匹配，其中的命名空间分隔符将作为目录分隔符。

末尾的类名必须与对应的以 .php 为后缀的文件同名。

自动加载器（autoloader）的实现一定不能抛出异常、一定不能触发任一级别的错误信息以及不应该有返回值。

## 3. 例子

下表展示了符合规范完整类名、命名空间前缀和文件基目录所对应的文件路径。

|  |  |  |  |
| --- | --- | --- | --- |
| 完整类名 | 命名空间前缀 | 文件基目录 | 文件路径 |
| \Acme\Log\Writer\File\_Writer | Acme\Log\Writer | ./acme-log-writer/lib/ | ./acme-log-writer/lib/File\_Writer.php |
| \Aura\Web\Response\Status | Aura\Web | /path/to/aura-web/src/ | /path/to/aura-web/src/Response/Status.php |
| \Symfony\Core\Request | Symfony\Core | ./vendor/Symfony/Core/ | ./vendor/Symfony/Core/Request.php |
| \Zend\Acl | Zend | /usr/includes/Zend/ | /usr/includes/Zend/Acl.php |

关于本规范的实现，可参阅 相关实例

注意：实例并不属于规范的一部分，且随时会有所变动。

# 日志接口规范 PSR-3

本文制定了日志类库的通用接口规范。

本规范的主要目的，是为了让日志类库以简单通用的方式，通过接收一个 Psr\Log\LoggerInterface 对象，来记录日志信息。 框架以及CMS内容管理系统如有需要，可以对此接口进行扩展，但需遵循本规范， 这才能保证在使用第三方的类库文件时，日志接口仍能正常对接。

关键词 “必须”("MUST")、“一定不可/一定不能”("MUST NOT")、“需要”("REQUIRED")、 “将会”("SHALL")、“不会”("SHALL NOT")、“应该”("SHOULD")、“不该”("SHOULD NOT")、 “推荐”("RECOMMENDED")、“可以”("MAY")和”可选“("OPTIONAL")的详细描述可参见 RFC 2119 。

本文中的 实现者 指的是实现了 LoggerInterface 接口的类库或者框架，反过来讲，他们就是 LoggerInterface 的 使用者。

## 1. 规范说明

### 1.1 基本规范

LoggerInterface 接口对外定义了八个方法，分别用来记录 RFC 5424 中定义的八个等级的日志：debug、 info、 notice、 warning、 error、 critical、 alert 以及 emergency 。

第九个方法 —— log，其第一个参数为记录的等级。可使用一个预先定义的等级常量作为参数来调用此方法，必须与直接调用以上八个方法具有相同的效果。如果传入的等级常量参数没有预先定义，则必须抛出 Psr\Log\InvalidArgumentException 类型的异常。在不确定的情况下，使用者不该使用未支持的等级常量来调用此方法。

### 1.2 记录信息

以上每个方法都接受一个字符串类型或者是有 \_\_toString() 方法的对象作为记录信息参数，这样，实现者就能把它当成字符串来处理，否则实现者必须自己把它转换成字符串。

记录信息参数可以携带占位符，实现者可以根据上下文将其它替换成相应的值。

其中占位符必须与上下文数组中的键名保持一致。

占位符的名称必须由一个左花括号 { 以及一个右括号 } 包含。但花括号与名称之间一定不能有空格符。

占位符的名称应该只由 A-Z、 a-z,0-9、下划线 \_、以及英文的句号 .组成，其它字符作为将来占位符规范的保留。

实现者可以通过对占位符采用不同的转义和转换策略，来生成最终的日志。 而使用者在不知道上下文的前提下，不该提前转义占位符。

以下是一个占位符使用的例子：



### 1.3 上下文

每个记录函数都接受一个上下文数组参数，用来装载字符串类型无法表示的信息。它可以装载任何信息，所以实现者必须确保能正确处理其装载的信息，对于其装载的数据，一定不能 抛出异常，或产生PHP出错、警告或提醒信息（error、warning、notice）。

如需通过上下文参数传入了一个 Exception 对象， 必须以 'exception' 作为键名。 记录异常信息是很普遍的，所以如果它能够在记录类库的底层实现，就能够让实现者从异常信息中抽丝剥茧。 当然，实现者在使用它时，必须确保键名为 'exception' 的键值是否真的是一个 Exception，毕竟它可以装载任何信息。

### 1.4 助手类和接口

Psr\Log\AbstractLogger 类使得只需继承它和实现其中的 log 方法，就能够很轻易地实现 LoggerInterface 接口，而另外八个方法就能够把记录信息和上下文信息传给它。

同样地，使用 Psr\Log\LoggerTrait 也只需实现其中的 log 方法。不过，需要特别注意的是，在traits可复用代码块还不能实现接口前，还需要 implement LoggerInterface。

在没有可用的日志记录器时， Psr\Log\NullLogger 接口可以为使用者提供一个备用的日志“黑洞”。不过，当上下文的构建非常消耗资源时，带条件检查的日志记录或许是更好的办法。

Psr\Log\LoggerAwareInterface 接口仅包括一个 setLogger(LoggerInterface $logger) 方法，框架可以使用它实现自动连接任意的日志记录实例。

Psr\Log\LoggerAwareTrait trait可复用代码块可以在任何的类里面使用，只需通过它提供的 $this->logger，就可以轻松地实现等同的接口。

Psr\Log\LogLevel 类装载了八个记录等级常量。

## 2. 包

上述的接口、类和相关的异常类，以及一系列的实现检测文件，都包含在 psr/log 文件包中。

## 3. Psr\Log\LoggerInterface

<?php

namespace Psr\Log;

/\*\*

\* 日志记录实例

\*

\* 日志信息变量 —— message， \*\*必须\*\*是一个字符串或是实现了 \_\_toString() 方法的对象。

\*

\* 日志信息变量中\*\*可以\*\*包含格式如 “{foo}” (代表foo) 的占位符，

\* 它将会由上下文数组中键名为 "foo" 的键值替代。

\*

\* 上下文数组可以携带任意的数据，唯一的限制是，当它携带的是一个 exception 对象时，它的键名 必须 是 "exception"。

\*

\* 详情可参阅： https://github.com/PizzaLiu/PHP-FIG/blob/master/PSR-3-logger-interface-cn.md

\*/

interface LoggerInterface

{

/\*\*

\* 系统不可用

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function emergency($message, array $context = array());

/\*\*

\* \*\*必须\*\*立刻采取行动

\*

\* 例如：在整个网站都垮掉了、数据库不可用了或者其他的情况下，\*\*应该\*\*发送一条警报短信把你叫醒。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function alert($message, array $context = array());

/\*\*

\* 紧急情况

\*

\* 例如：程序组件不可用或者出现非预期的异常。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function critical($message, array $context = array());

/\*\*

\* 运行时出现的错误，不需要立刻采取行动，但必须记录下来以备检测。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function error($message, array $context = array());

/\*\*

\* 出现非错误性的异常。

\*

\* 例如：使用了被弃用的API、错误地使用了API或者非预想的不必要错误。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function warning($message, array $context = array());

/\*\*

\* 一般性重要的事件。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function notice($message, array $context = array());

/\*\*

\* 重要事件

\*

\* 例如：用户登录和SQL记录。

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function info($message, array $context = array());

/\*\*

\* debug 详情

\*

\* @param string $message

\* @param array $context

\* @return null

\*/

public function debug($message, array $context = array());

/\*\*

\* 任意等级的日志记录

\*

\* @param mixed $level

\* @param string $message

\* @param array $context

\* @return null

\*/

public function log($level, $message, array $context = array());

}

## 4. Psr\Log\LoggerAwareInterface

<?php

namespace Psr\Log;

/\*\*

\* logger-aware 定义实例

\*/

interface LoggerAwareInterface

{

/\*\*

\* 设置一个日志记录实例

\*

\* @param LoggerInterface $logger

\* @return null

\*/

public function setLogger(LoggerInterface $logger);

}

## 5. Psr\Log\LogLevel

<?php

namespace Psr\Log;

/\*\*

\* 日志等级常量定义

\*/

class LogLevel

{

const EMERGENCY = 'emergency';

const ALERT = 'alert';

const CRITICAL = 'critical';

const ERROR = 'error';

const WARNING = 'warning';

const NOTICE = 'notice';

const INFO = 'info';

const DEBUG = 'debug';

}