# PHP

# 编码规范

# Introduction

# PHP编码规范（中文版）导读

本文档是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（PSR:Proposing a Standards Recommendation）中译版。

翻译过程中参照了 莫希爾(Mosil)手札 的繁体中文版，以及 Corrie Zhao 组织翻译的简体中文版， 译文中为了让语句通顺，便于理解，没有对原文逐字翻译，个别语句与原文原意可能略有偏差，希望告知指正。

目前官方已制定的规范包括以下六份文件：

- PSR-0 (已弃用)
- PSR-1
- PSR-2
- PSR-2补充
- PSR-3
- PSR-4
- 2014/04/25 添加 `PSR-2补充` 文件以及修改之前版本中的翻译不当与错误。
- 2014/07/31 添加 `PSR-4` 。

以下是原版的导读：

# PHP互操作性框架制定小组

组建本小组的目的是，通过在各项目的代表之间讨论他们共同的编码规范，以制定一个协作标准。本规范的主要面向对象是本小组的各个组成成员，当然，同时也欢迎关注本规范的其它PHP社区采用本规范。

## 提交规范建议

可以通过以下方式给本规范提交建议:

- fork PSR代码库，创建并检出一个分支，在 `proposed/` 下添加 规范建议，然后 push 分支到 Github，最后给我们发送一个 pull request；又或者
- 在 Github 下新建一个讨论 ticket；又或者
- 在 邮件列表 中提交建议。

# 成为投票成员

注意，你 不需要 成为投票成员才能在 邮件列表 中发表言论。

想要成为投票成员，你必须发送一封邮件到 邮件列表 中。

- 邮件主题格式如下: `Membership Request: {你的名字} ({参与的项目名称})`
- 邮件内容应包括你的名字、你参与的项目名称、项目的地址以及其它相关信息。

目前的成员会对你的加入请求进行投票。

请不要在一份申请中提交多个加入请求，每份申请只能提交一份请求。

# 目前的成员及其代表项目列表

1. Nate Abele: Lithium
2. Nils Adermann: phpBB
3. Brett Bieber: PEAR, PEAR2
4. Guilherme Blanco: Doctrine, Doctrine2, et al.
5. Jordi Boggiano: Composer, Packagist
6. Pádraic Brady: Zend Framework
7. Karma Dordrak: Zikula
8. Paul Dragoonis: PPI, PPI2
9. William Durand: Propel, Propel 2
10. Don Gilbert: Joomla
11. Cal Evans: the community at large
12. Larry Garfield: Drupal
13. Ivan Habunek: Apache log4php
14. Paul M. Jones: Solar Framework, Aura Project
15. Karsten Dambekalns: TYPO3 Flow, TYPO3 Neos
16. Larry Masters: CakePHP, CakePHP 2
17. John Mertic: SugarCRM

18. Taylor Otwell: Laravel
19. Ryan Parman: Amazon Web Services SDK
20. Evert Pot: SabreDAV
21. Fabien Potencier: Symfony, Symfony2
22. Mike van Riel: phpDocumentor
23. Andre Romcke: eZ Publish
24. Phil Sturgeon: PyroCMS
25. Lukas Smith: Jackalope
26. Kris Wallsmith: Assetic, Buzz
27. David Zülke: Agavi

# PSR-1 基本代码规范

# 基本代码规范

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-0 或 PSR-4 中的一个；
- 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。

### 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：

```
<?php
// 从属效应：修改 ini 配置
ini_set('error_reporting', E_ALL);

// 从属效应：引入文件
include "file.php";

// 从属效应：生成输出
echo "<html>\n";

// 声明函数
functionfoo(){
    // 函数主体部分
}
```

下面是一个范例，一份只包含声明不产生从属效应的代码：

```
<?php
// 声明函数
functionfoo(){
    // 函数主体部分
}

// 条件声明**不**属于从属效应
if (! function_exists('bar')) {
    functionbar(){
        // 函数主体部分
    }
}
```

# 3. 命名空间和类

命名空间以及类的命名必须遵循 PSR-0.

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：

```
<?php
// PHP 5.3及以后版本的写法
namespace Vendor\Model;

class Foo{
}
```

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。

```
<?php
// 5.2.x及之前版本的写法
class Vendor_Model_Foo{
}
```

# 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：

```
<?php
namespace Vendor\Model;

class Foo{
    const VERSION = '1.0';
    const DATE_APPROVED = '2012-06-01';
}
```

## 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。

## 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。

# PSR-2 代码风格规范

# 代码风格规范

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：

```
<?php
namespace Vendor\Package;

use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class Foo extends Bar implements FooInterface
{
    public function sampleFunction($a, $b = null)
    {
        if ($a === $b) {
            bar();
        } elseif ($a > $b) {
            $foo->bar($arg1);
        } else {
            BazClass::bar($arg2, $arg3);
        }
    }

    final public static functionbar()
    {
        // method body
    }
}
```

# 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：

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

// ... additional PHP code ...
```

# 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。

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // constants, properties, methods
}
```

`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。

```
<?php
namespace Vendor\Package;

use FooClass;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class ClassName extends ParentClass implements
\ArrayAccess,
\Countable,
\Serializable
{
    // constants, properties, methods
}
```

## 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：

```
<?php
namespace Vendor\Package;

classClassName
{
    public $foo = null;
}
```

## 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。

```
<?php
namespace Vendor\Package;

class ClassName
{
    public function fooBarBaz($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

## 4.4. 方法的参数

参数列表中，每个参数后面必须要有一个空格，而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。

```
<?php
namespace Vendor\Package;

class ClassName
{
    public functionfoo($arg1, &$arg2, $arg3 = [])
    {
        // method body
    }
}
```

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。

```
<?php
namespace Vendor\Package;

class ClassName
{
    public function aVeryLongMethodName( ClassTypeHint $arg1, &$arg2, array $arg3 = [] )
    {
        // method body
    }
}
```

## 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。

```
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected functionzim();

    final public static functionbar()
    {
        // method body
    }
}
```

## 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。

```
<?php
bar();
$foo->bar($arg1);
Foo::bar($arg2, $arg3);
```

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。

```
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

# 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。

```
<?php
if ($expr1) {
    // if body
} elseif ($expr2) {
    // elseif body
} else {
    // else body;
}
```

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。

```
<?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;
}
```

## 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。

```
<?php
while ($expr) {
    // structure body
}
```

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。

```
<?php
do {
    // structure body;
} while ($expr);
```

## 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。

```
<?php
for ($i = 0; $i < 10; $i++) {
    // for body
}
```

## 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。

```
<?php
foreach ($iterable as $key => $value) {
    // foreach body
}
```

## 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。

```
<?php
try {
    // try body
} catch (FirstExceptionType $e) {
    // catch body
} catch (OtherExceptionType $e) {
    // catch body
}
```

# 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。

```
<?php
$closureWithArgs = function($arg1, $arg2){
    // body
};

$closureWithArgsAndVars = function($arg1, $arg2)use($var1, $var2){
    // body
};
```

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。

```
<?php
$longArgs_noVars = function( $longArgument, $longerArgument, $muchLongerArgument ){
  // body
};

$noArgs_longVars = function()use( $longVar1, $longerVar2, $muchLongerVar3 ){
  // body
};

$longArgs_longVars = function( $longArgument, $longerArgument, $muchLongerArgument )use( $lo
ngVar1, $longerVar2, $muchLongerVar3 ){
  // body
};

$longArgs_shortVars = function( $longArgument, $longerArgument, $muchLongerArgument )use($va
r1){
  // body
};

$shortArgs_longVars = function($arg)use( $longVar1, $longerVar2, $muchLongerVar3 ){
  // body
};
```

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。

```
<?php
$foo->bar(
    $arg1,
    function($arg2)use($var1){
        // body
    },
    $arg3
);
```

# 7. 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。

# 附录 A. 问卷调查

为了编写本规范，小组制定了调查问卷，用来统计各成员项目的共同规范。 以下是此问卷调查的数据，在此供查阅。

## A.1. 问卷数据

url,http://www.horde.org/apps/horde/docs/CODING_STANDARDS,http://pear.php.net/manual/en/standards.php,http://solarphp.com/manual/appendix-standards.style,http://framework.zend.com/manual/en/coding-standard.html,http://symfony.com/doc/2.0/contributing/code/standards.html,http://www.ppi.io/docs/coding-standards.html,https://github.com/ezsystems/ezp-next/wiki/codingstandards,http://book.cakephp.org/2.0/en/contributing/cakephp-coding-conventions.html,https://github.com/UnionOfRAD/lithium/wiki/Spec%3A-Coding,http://drupal.org/coding-standards,http://code.google.com/p/sabredav/,http://area51.phpbb.com/docs/31x/coding-guidelines.html,https://docs.google.com/a/zikula.org/document/edit?authkey=CPCU0Us&hgd=1&id=1fcqb93Sn-hR9c0mkN6m_tyWnmEvoswKBtSc0tKkZmJA,http://www.chisimba.com,n/a,https://github.com/Respect/project-info/blob/master/coding-standards-sample.php,n/a,Object Calisthenics for PHP,http://doc.nette.org/en/coding-standard,http://flow3.typo3.org,https://github.com/propelorm/Propel2/wiki/Coding-Standards,http://developer.joomla.org/coding-standards.html
voting,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,no,no,no,?,yes,no,yes
indent_type,4,4,4,4,4,tab,4,tab,tab,2,4,tab,4,4,4,4,4,4,tab,tab,4,tab
line_length_limit_soft,75,75,75,75,no,85,120,120,80,80,80,no,100,80,80,?,?,120,80,120,no,150
line_length_limit_hard,85,85,85,85,no,no,no,no,100,?,no,no,no,100,100,?,120,120,no,no,no,no
class_names,studly,studly,studly,studly,studly,studly,studly,studly,studly,studly,studly,lower_under,studly,lower,studly,studly,studly,studly,?,studly,studly,studly
class_brace_line,next,next,next,next,next,same,next,same,same,same,same,next,next,next,next,next,next,next,next,next,same,next,next
constant_names,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper
true_false_null,lower,lower,lower,lower,lower,lower,lower,lower,lower,upper,lower,lower,lower,upper,lower,lower,lower,lower,lower,upper,lower,lower
method_names,camel,camel,camel,camel,camel,camel,camel,camel,camel,camel,camel,lower_under,camel,camel,camel,camel,camel,camel,camel,camel,camel,camel,camel
method_brace_line,next,next,next,next,next,same,next,same,same,same,same,next,next,same,next,next,next,next,next,next,same,next,next
control_brace_line,same,same,same,same,same,same,next,same,same,same,same,next,same,same,next,same,same,same,same,same,same,next
control_space_after,yes,yes,yes,yes,yes,no,yes,yes,yes,yes,no,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes,yes
always_use_control_braces,yes,yes,yes,yes,yes,yes,no,yes,yes,yes,no,yes,yes,yes,yes,no,yes,yes,yes,yes,yes,yes
else_elseif_line,same,same,same,same,same,same,next,same,same,next,same,next,same,next,next,same,same,same,same,same,next
case_break_indent_from_switch,0/1,0/1,0/1,1/2,1/2,1/2,1/2,1/1,1/1,1/1,1/2,1/2,1/1,1/2,1/2,1/2,1/2,1/2,1/2,1/2,0/1,1/1,1/2,1/2
function_space_after,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no,no
closing_php_tag_required,no,no,no,no,no,no,no,no,yes,no,no,no,no,yes,no,no,no,no,no,yes,no,no
line_endings,LF,LF,LF,LF,LF,LF,LF,LF,?,LF,?,LF,LF,LF,LF,?,,LF,?,LF,LF,LF
static_or_visibility_first,static,?,static,either,either,either,visibility,visibility,visibility,either,static,either,?,visibility,?,?,either,either,visibility,visibility,static,?
control_space_parens,no,no,no,no,no,no,yes,no,no,no,no,no,no,yes,?,no,no,no,no,no,no,no
blank_line_after_php,no,no,no,no,yes,no,no,no,no,yes,yes,no,no,yes,?,yes,yes,no,yes,no,yes,no
class_method_control_brace,next/next/same,next/next/same,next/next/same,next/next/same,next/next/same,next/next/same,same/same/same,next/next/next,same/same/same,same/same/same,same/same/same,same/same/same,next/next/next,next/next/same,next/same/same,next/next/next,next/next/same,next/next/same,next/next/same,next/next/same,next/next/same,next/next/same,same/same/same,next/next/same,next/next/next

## A.2. 问卷说明

`indent_type` : 缩进类型. `tab` = "使用 tab 键一次", `2` or `4` = "空格的数量"

`line_length_limit_soft`：每行字符数量的“软”限制. `?` = 不可辩或无作答, `no` 表示无限制.

`line_length_limit_hard`：每行字符数量的“硬”限制. `?` = 不可辩或无作答, `no` 表示无限制.

`class_names`：类名称的命名. `lower` = 只允许小写字母, `lower_under` = 下滑线分隔的小写字母, `studly` = StudlyCase 的驼峰风格.

`class_brace_line`：类的开始花括号是与 class 关键字在同一行或是在其的下一行？

`constant_names`：类的常量如何命名? `upper` = 下划线分隔的大写字母.

`true_false_null`：关键字 `true`、`false` 以及 `null` 是全部小写 `lower` 还是全部大写 `upper`？

`method_names`：方法名称如何命名? `camel` = `camelCase`, `lower_under` = 下划线分隔的小写字母.

`method_brace_line`：方法的开始花括号是与方法名在同一行还是在其的下一行？

`control_brace_line`：控制结构的开始花括号是与声明在同一行还是在其的下一行？

`control_space_after`：控制结构关键词后是否有空格？

`always_use_control_braces`：控制结构体是否都要被包含在花括号内？

`else_elseif_line`： `else` 或 `elseif` 与前面的结束花括号在同一行还是在其的下一行？

`case_break_indent_from_switch`： `switch` 语句中的 `case` 和 `break` 需要相对 `switch` 缩进多少次？

`function_space_after`：函数调用语句中，函数名称与变量列表的左括号间是否有空格？

`closing_php_tag_required`：纯 PHP 代码的文件，是否需要 `?>` 结束标签？

`line_endings`：选择哪种类型的行结束符？

`static_or_visibility_first`：声明一个静态方法时， `static` 是写访问修饰符前还是后？

`control_space_parens`：控制结构里，左括号后以及右括号前是否有空格？ `yes` = `if ( $expr )`, `no` = `if ($expr)`.

`blank_line_after_php`：PHP 开始标签后，是否需要一个空行？

`class_method_control_brace`：开始花括号在类、方法和控制结构的位置统计。

## A.3. 问卷统计结果

```
indent_type:
    tab: 7
    2: 1
    4: 14
```

```
  line_length_limit_soft:
    ?: 2
    no: 3
    75: 4
    80: 6
    85: 1
    100: 1
    120: 4
    150: 1
  line_length_limit_hard:
    ?: 2
    no: 11
    85: 4
    100: 3
    120: 2
  class_names:
    ?: 1
    lower: 1
    lower_under: 1
    studly: 19
  class_brace_line:
    next: 16
    same: 6
  constant_names:
    upper: 22
  true_false_null:
    lower: 19
    upper: 3
  method_names:
    camel: 21
    lower_under: 1
  method_brace_line:
    next: 15
    same: 7
  control_brace_line:
    next: 4
    same: 18
  control_space_after:
    no: 2
    yes: 20
  always_use_control_braces:
    no: 3
    yes: 19
  else_elseif_line:
    next: 6
    same: 16
  case_break_indent_from_switch:
    0/1: 4
    1/1: 4
    1/2: 14
  function_space_after:
    no: 22
  closing_php_tag_required:
    no: 19
    yes: 3
```

```
line_endings:
    ?: 5
    LF: 17
static_or_visibility_first:
    ?: 5
    either: 7
    static: 4
    visibility: 6
control_space_parens:
    ?: 1
    no: 19
    yes: 2
blank_line_after_php:
    ?: 1
    no: 13
    yes: 8
class_method_control_brace:
    next/next/next: 4
    next/next/same: 11
    next/same/same: 1
    same/same/same: 6
```

# PSR-2-1 补充文档

# PSR-2 补充文档

## 1. 摘要

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。

## 2. 投票

- 投票点:ML

## 3. 勘误

### 3.1 - 多行参数 (09/08/2013)

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：

```
<?php
somefunction($foo, $bar, [
  // ...
], $baz);

$app->get('/hello/{name}', function($name)use($app){
    return 'Hello '.$app->escape($name);
});
```

### 3.2 - 多行参数 (10/17/2013)

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。

# PSR-3 日志接口规范

# 日志接口规范

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：

```
/** * 用上下文信息替换记录信息中的占位符 */
function interpolate($message, array $context = array()){
    // 构建一个花括号包含的键名的替换数组
    $replace = array();
    foreach ($context as $key => $val) {
        $replace['{' . $key . '}'] = $val;
    }

    // 替换记录信息中的占位符，最后返回修改后的记录信息。
    return strtr($message, $replace);
}

// 含有带花括号占位符的记录信息。
$message = "User {username} created";

// 带有替换信息的上下文数组，键名为占位符名称，键值为替换值。
$context = array('username' => 'bolivar');

// 输出 "Username bolivar created"
echo interpolate($message, $context);
```

# 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。
- 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 $context * @return null */
    public functionemergency($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 $context * @return null */
    public functionalert($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 $context * @return null */
    public functioncritical($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 $context * @return null */
    public functionerror($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 $context * @return null */
    public functionwarning($message, array $context = array());

    /** * 一般性重要的事件。 * * @param string $message * @param array $context * @return null */
    public functionnotice($message, array $context = array());

    /** * 重要事件 * * 例如：用户登录和SQL记录。 * * @param string $message * @param array $context * @return null */
    public functioninfo($message, array $context = array());

    /** * debug 详情 * * @param string $message * @param array $context * @return null */
    public functiondebug($message, array $context = array());

    /** * 任意等级的日志记录 * * @param mixed $level * @param string $message * @param array $context * @return null */
    public functionlog($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 functionsetLogger(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';
}
```

# PSR-4 自动载入

# Autoloader

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][http://tools.ietf.org/html/rfc2119] 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

1. 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。
2. 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>
```

1. 完整的类名必须要有一个顶级命名空间，被称为 "vendor namespace"；
2. 完整的类名可以有一个或多个子命名空间；
3. 完整的类名必须有一个最终的类名；
4. 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；
5. 完整的类名可以由任意大小写字母组成；
6. 所有类名都必须是大小写敏感的。
7. 当根据完整的类名载入相应的文件……
8. 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；
9. 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。
10. 末尾的类名必须与对应的以 `.php` 为后缀的文件同名。

L1. 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-wr |
| \Aura\Web\Response\Status | Aura\Web | /path/to/aura-web/src/ | /path/to/aura-web/src/Respc |
| \Symfony\Core\Request | Symfony\Core | ./vendor/Symfony/Core/ | ./vendor/Symf |
| \Zend\Acl | Zend | /usr/includes/Zend/ | /usr/includes/Z |

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。