PSR-1

基本代码规范

本节我们将会讨论一些基本的代码规范问题，以此作为将来讨论更高级别的代码分享和技术互用的基础。

RFC 2119中的必须(MUST)，不可(MUST NOT)，建议(SHOULD)，不建议(SHOULD NOT)，可以/可能(MAY)等关键词将在本节用来做一些解释性的描述。

1. 概述

- 源文件必须只使用 <?php 和 <?= 这两种标签。
- 源文件中php代码的编码格式必须只使用不带字节顺序标记(BOM)的UTF-8。
- 一个源文件建议只用来做声明（类(class)，函数(function)，常量(constant)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。
- 命名空间(namespace)和类(class) 必须遵守PSR-0标准。
- 类名(class name) 必须使用骆驼式(StudlyCaps)写法 (译者注：驼峰式(cameCase)的一种变种，后文将直接用StudlyCaps表示)。
- 类(class)中的常量必须只由大写字母和下划线(_)组成。
- 方法名(method name) 必须使用驼峰式(cameCase)写法(译者注：后文将直接用camelCase表示)。

2. 文件

2.1. PHP标签

PHP代码必须只使用长标签(<?php ?>)或者短输出式标签(<?= ?>)；而不可使用其他标签。

2.2. 字符编码

PHP代码的编码格式必须只使用不带字节顺序标记(BOM)的UTF-8。

2.3. 副作用

一个源文件建议只用来做声明（类(class)，函数(function)，常量(constant)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。

短语副作用(side effects)的意思是 *在包含文件时* 所执行的逻辑与所声明的类(class)，函数(function)，常量(constant)等没有直接的关系。

副作用(side effects)包含但不局限于：产生输出，显式地使用require或include，连接外部服务，修改ini配置，触发错误或异常，修改全局或者静态变量，读取或修改文件等等

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

// 副作用：载入了文件
include "file.php";

// 副作用：产生了输出
echo "<html>\n";

// 声明
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
<?php
// 声明
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
    }
}
```

3. 空间名(namespace)和类名(class name)

命名空间(namespace)和类(class)必须遵守 PSR-0.

这意味着一个源文件中只能有一个类(class)，并且每个类(class)至少要有一级空间名（namespace）：即一个顶级的组织名(vendor name)。

类名(class name) 必须使用StudlyCaps写法。

PHP5.3之后的代码必须使用正式的命名空间(namespace) 例子：

```php
<?php
// PHP 5.3 及之后:
namespace Vendor\Model;

class Foo
{
}
```

PHP5.2.x之前的代码建议用伪命名空间Vendor_作为类名(class name)的前缀

```php
<?php
// PHP 5.2.x 及之前:
class Vendor_Model_Foo
{
}
```

4. 类的常量、属性和方法

术语类(class)指所有的类(class)，接口(interface)和特性(trait)

4.1. 常量

类常量必须只由大写字母和下划线(_)组成。 例子：

```php
<?php
namespace Vendor\Model;

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

4.2. 属性

本指南中故意不对$StulyCaps，$camelCase或者$unser_score中的某一种风格作特别推荐，完全由读者依据个人喜好决定属性名的命名风格。

但是不管你如何定义属性名，建议在一个合理的范围内保持一致。这个范围可能是组织(vendor)级别的，包(package)级别的，类(class)级别的，或者方法(method)级别的。

4.3. 方法

方法名则必须使用camelCase()风格来声明。

PSR-2

代码风格指南

本手册是基础代码规范(PSR-1)的继承和扩展。

为了尽可能的提升阅读其他人代码时的效率，下面例举了一系列的通用规则，特别是有关于PHP代码风格的。

各个成员项目间的共性组成了这组代码规范。当开发者们在多个项目中合作时，本指南将会成为所有这些项目中共用的一组代码规范。 因此，本指南的益处不在于这些规则本身，而在于在所有项目中共用这些规则。

RFC 2119中的必须(MUST)，不可(MUST NOT)，建议(SHOULD)，不建议(SHOULD NOT)，可以/可能(MAY)等关键词将在本节用来做一些解释性的描述。

1. 概述

- 代码必须遵守 PSR-1。
- 代码必须使用4个空格来进行缩进，而不是用制表符。
- 一行代码的长度不建议有硬限制；软限制必须为120个字符，建议每行代码80个字符或者更少。
- 在命名空间(namespace)的声明下面必须有一行空行，并且在导入(use)的声明下面也必须有一行空行。
- 类(class)的左花括号必须放到其声明下面自成一行，右花括号则必须放到类主体下面自成一行。
- 方法(method)的左花括号必须放到其声明下面自成一行，右花括号则必须放到方法主体的下一行。
- 所有的属性(property)和方法(method) 必须有可见性声明；抽象(abstract)和终结(final)声明必须在可见性声明之前；而静态(static)声明必须在可见性声明之后。
- 在控制结构关键字的后面必须有一个空格；而方法(method)和函数(function)的关键字的后面不可有空格。
- 控制结构的左花括号必须跟其放在同一行，右花括号必须放在该控制结构代码主体的下一行。
- 控制结构的左括号之后不可有空格，右括号之前也不可有空格。

1.1. 示例

这个示例中简单展示了上文中提到的一些规则：

```php
<?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 function bar()
    {
        // 方法主体
    }
}
```

2. 通则

2.1 基础代码规范

代码必须遵守 PSR-1 中的所有规则。

2.2 源文件

所有的PHP源文件必须使用Unix LF(换行)作为行结束符。

所有PHP源文件必须以一个空行结束。

纯PHP代码源文件的关闭标签?> 必须省略。

2.3. 行

行长度不可有硬限制。

行长度的软限制必须是120个字符；对于软限制，代码风格检查器必须警告但不可报错。

一行代码的长度不建议超过80个字符；较长的行建议拆分成多个不超过80个字符的子行。

在非空行后面不可有空格。

空行可以用来增强可读性和区分相关代码块。

一行不可多于一个语句。

2.4. 缩进

代码必须使用4个空格，且不可使用制表符来作为缩进。

> 注意：代码中只使用空格，且不和制表符混合使用，将会对避免代码差异，补丁，历史和注解中的一些问题有帮助。空格的使用还可以使通过调整细微的缩进来改进行间对齐变得更加的简单。

2.5. 关键字和 True/False/Null

PHP关键字(keywords)必须使用小写字母。

PHP常量true, false和null 必须使用小写字母。

3. 命名空间(Namespace)和导入(Use)声明

命名空间(namespace)的声明后面必须有一行空行。

所有的导入(use)声明必须放在命名空间(namespace)声明的下面。

一句声明中，必须只有一个导入(use)关键字。

在导入(use)声明代码块后面必须有一行空行。

示例：

```php
<?php
namespace Vendor\Package;

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

// ... 其它PHP代码 ...

```

4. 类(class)，属性(property)和方法(method)

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

4.1. 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(class name)在同一行。

类(class)的左花括号必须放在下面自成一行；右花括号必须放在类(class)主体的后面自成一行。

```php
<?php
namespace Vendor\Package;

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

class ClassName extends ParentClass implements \ArrayAccess, \Countable
{
    // 常量、属性、方法
}
```

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)。

```php
<?php
namespace Vendor\Package;

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

class ClassName extends ParentClass implements
    \ArrayAccess,
    \Countable,
    \Serializable
{
    // 常量、属性、方法
}
```

4.2. 属性(property)

所有的属性(property)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(property)。

一条语句不可声明多个属性(property)。

属性名(property name)不推荐用单个下划线作为前缀来表明其保护(protected)或私有(private)的可见性。

一个属性(property)声明看起来应该像下面这样。

```php
<?php
namespace Vendor\Package;

class ClassName
{
    public $foo = null;
}
```

4.3. 方法(method)

所有的方法(method)都必须声明其可见性。

方法名(method name) 不推荐用单个下划线作为前缀来表明其保护(protected)或私有(private)的可见性。

方法名(method name)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(method)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：

```php
<?php
namespace Vendor\Package;

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

4.4. 方法(method)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(method)中有默认值的参数必须放在参数列表的最后面。

```php
<?php
namespace Vendor\Package;

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。

当参数列表被拆分成多个子行，右括号和左花括号之间必须有一个空格并且自成一行。

```php
<?php
namespace Vendor\Package;

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

4.5. 抽象(abstract)，终结(final)和 静态(static)

当用到抽象(abstract)和终结(final)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
<?php
namespace Vendor\Package;

abstract class ClassName
{
    protected static $foo;

    abstract protected function zim();

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

4.6. 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
<?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
<?php
$foo->bar(
    $longArgument,
    $longerArgument,
    $muchLongerArgument
);
```

5. 控制结构

下面是对于控制结构代码风格的概括：

- 控制结构的关键词之后必须有一个空格。
- 控制结构的左括号之后不可有空格。
- 控制结构的右括号之前不可有空格。
- 控制结构的右括号和左花括号之间必须有一个空格。
- 控制结构的代码主体必须进行一次缩进。
- 控制结构的右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。

5.1. if，elseif，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。同时注意else和elseif要和前一个条件控制结构的右花括号在同一行。

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

推荐用elseif来替代else if，以保持所有的条件控制关键字看起来像是一个单词。

5.2. 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
<?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，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。

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

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。

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

5.4. 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。

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

5.5. 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。

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

5.6. 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。

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

6. 闭包

声明闭包时所用的function关键字之后必须要有一个空格，而use关键字的前后都要有一个空格。

闭包的左花括号必须跟其在同一行，而右花括号必须在闭包主体的下一行。

闭包的参数列表和变量列表的左括号后面不可有空格，右括号的前面也不可有空格。

闭包的参数列表和变量列表中逗号前面不可有空格，而逗号后面则必须有空格。

闭包的参数列表中带默认值的参数必须放在参数列表的结尾部分。

下面是一个闭包的示例。注意括号，空格和花括号的位置。

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

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

参数列表和变量列表可以被拆分成多个缩进了一级的子行。如果要拆分成多个子行，列表中的第一项必须放在下一行，并且每一行必须只放一个参数或变量。

当列表（不管是参数还是变量）最终被拆分成多个子行，右括号和左花括号之间必须要有一个空格并且自成一行。

下面是一个参数列表和变量列表被拆分成多个子行的示例。

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

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

$longArgs_longVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use (
    $longVar1,
    $longerVar2,
    $muchLongerVar3
) {
   // body
};

$longArgs_shortVars = function (
    $longArgument,
    $longerArgument,
    $muchLongerArgument
) use ($var1) {
   // body
};

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

把闭包作为一个参数在函数或者方法中调用时，依然要遵守上述规则。

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

7. 结论

本指南有意的省略了许多元素的代码风格。主要包括：

- 全局变量和全局常量的声明
- 函数声明
- 操作符和赋值
- 行间对齐
- 注释和文档块
- 类名的前缀和后缀
- 最佳实践

以后的代码规范中可能会修正或扩展本指南中规定的代码风格。

附录A 调查

为了写这个风格指南，我们调查了各个项目以最终确定通用的代码风格。并把这次调查在这里公布出来。

A.1. 调查数据

```
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,n
ext,next,next,next,next,same,next,next
constant_names,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,upper,u
pper,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_und
er,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,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,ye
s,yes,yes,yes,yes
always_use_control_braces,yes,yes,yes,yes,yes,yes,no,yes,yes,yes,no,yes,yes,yes,yes,no,y
es,yes,yes,yes,yes,yes,yes
else_elseif_line,same,same,same,same,same,same,next,same,same,next,same,next,same,next,n
ext,same,same,same,same,same,same,next
case_break_indent_from_switch,0/1,0/1,0/1,1/2,1/2,1/2,1/2,1/1,1/1,1/2,1/2,1/1,1/2,1/2,1/
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
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,vi
sibility,either,static,either,?,visibility,?,?,either,either,visibility,visibility,stati
c,?
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,y
es,no
class_method_control_brace,next/next/same,next/next/same,next/next/same,next/next/same,n
ext/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/sa
me,next/next/same,next/next/same,next/next/same,same/same/same,next/next/same,next/next/
next
```

A.2. 调查说明

`indent_type`: 缩进类型。 `tab` = "使用制表符"， `2` or `4` = "空格数量"

`line_length_limit_soft`: 行长度的"软"限制，用字符。 `?` = 不表示或者数字 `no` 意为不限制.

`line_length_limit_hard`: 行长度的"硬"限制，用字符。 `?` = 不表示或者数字, `no` 意为不限制.

`class_names`: 类名如何命名 `lower` = 只是小写, `lower_under` = 小写加下划线, `studly` = 骆驼型.

`class_brace_line`: 类的左花括号是放在同(`same`)一行还是在下(`next`)一行？

`constant_names`: 类常量如何命名？ `upper` = 大写加下划线分隔符。

`true_false_null`: 全小写或者全大写？

method_names: 方法名如何命名？camel = 驼峰式, lower_under = 小写加下划线分隔符。

method_brace_line: 方法的左花括号在同(same)一行还是在下(next)一行？

control_brace_line: 控制结构的左花括号在同(same)一行还是在下(next)一行？

control_space_after: 控制结构关键词后是否有空格？

always_use_control_braces: 控制结构总是使用花括号？

else_elseif_line: 当使用else和elseif，是否放在同(same)一行还是在下(next)一行？

case_break_indent_from_switch: case和break分别从swith语句处缩进多少次？

function_space_after: 函数调用的函数名和左括号是否有空格？

closing_php_tag_required: 如过是纯PHP文件，关闭标签?>是否需要？

line_endings: 使用何种的行结束符？

static_or_visibility_first: 在定义方法的时候static和可见性谁在前面？

control_space_parens: 在控制结构表达式中，左括号后面和右括号前面是否要有一个空格？yes = if ($expr), no = if ($expr).

blank_line_after_php: PHP的开始标签后面是否需要一个空行？

class_method_control_brace: 左花括号在类，方法和控制结构中的位置。

A.3. 调查结果

```
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
```

```
    next/same/same: 1
```

Generated by *VNote*.

注释规范

```

1.文件的注释，介绍文件名，功能以及作者版本号等信息

/**
 * 文件名简单介绍
 *
 * 文件功能
 * @author  作者
 * @version  版本号
 * @date  2020-02-02
 */

2.类的注释，类名及介绍

/**
 * 类的介绍
 *
 * 类的详细介绍（可选）
 * @author  作者
 * @version  版本号
 * @date  2020-02-02
 */

3.函数的注释，函数的作用，参数介绍以及返回类型

/**
 * 函数的含义说明
 *
 * @access  public
 * @author  作者
 * @param mixed $arg1  参数一的说明
 * @param mixed $arg2  参数二的说明
 * @return array  返回类型
 * @date  2020-02-02
 */

```

命名规范

```
1. 【强制】代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。
反例： _name / __name / $name / name_ / name$ / name__

2. 【强制】代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。
说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式
也要避免采用。
正例： alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。
反例： DaZhePromotion [ 打折 ] / getPingfenByName() [ 评分 ] / int 某变量 = 3

3. 【强制】类名使用 UpperCamelCase 风格，但以下情形例外： DO / BO / DTO / VO / AO /
PO / UID 等。
正例： MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion
反例： macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

4. 【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从
驼峰形式。
正例： localValue / getHttpMessage() / inputUserId

5. 【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。
正例： MAX _ STOCK _ COUNT
反例： MAX _ COUNT

6. 【强制】抽象类命名使用 Abstract 或 Base 开头 ； 异常类命名使用 Exception 结尾 ； 测试类
命名以它要测试的类的名称开始，以 Test 结尾。

```

ThinkPHP 基础规范

1.目录和文件

目录使用小写+下划线
类库，函数文件统一以.php为后缀
类的文件名均以命名空间定义，并且命名空间的路径和类库文件所在路径一致
类文件采用驼峰法命名（首字母大写），其他文件采用小写+下划线命名
类名和类文件名保持一致，统一采用驼峰法（首字母大写）

2.函数和类，属性命名

类的命名采用驼峰法（首字母大写），例如 User、UserType，默认不需要添加后缀，例如UserController应该直接命名为User
函数的命名使用小写字母和下划线（小写字母开头）的方式，例如 get_client_ip
方法的命名使用驼峰法（首字母小写），例如 getUserName(如果方法有返回值，那么目前习惯上将首字母用小写的属性类型，如s(string)，i(int)，f(float)，b(boolean)，a(array)等)
属性的命名使用驼峰法（首字母小写），例如 tableName、instance（目前习惯上将首字母用小写的属性类型，如s(string)，i(int)，f(float)，b(boolean)，a(array)等）
以双下划线“__”打头的函数或方法作为魔法方法，例如 __call 和 __autoload
3.常量和配置

常量以大写字母和下划线命名，例如 APP_PATH和 THINK_PATH
配置参数以小写字母和下划线命名，例如 url_route_on 和url_convert
4.数据表盒字段

数据表和字段采用小写加下划线方式命名，并注意字段名不要以下划线开头，例如 think_user 表和 user_name字段，不建议使用驼峰和中文作为数据表字段命名。

MySQL 数据库规格

建表规约

1. 【强制】表达是与否概念的字段，必须使用 is_xxx 的方式命名，数据类型是 unsigned tinyint（1 表示是，0 表示否）。
说明：任何字段如果为非负数，必须是 unsigned 。
注意：数据库表示是与否的值，使用 tinyint 类型，坚持 is_xxx 的命名方式是为了明确其取值含义与取值范围。
正例：表达逻辑删除的字段名 is_deleted ，1 表示删除，0 表示未删除。

2. 【强制】表名、字段名必须使用小写字母或数字 ， 禁止出现数字开头，禁止两个下划线中间只出现数字。数据库字段名的修改代价很大，因为无法进行预发布，所以字段名称需要慎重考虑。
说明： MySQL 在 Windows 下不区分大小写，但在 Linux 下默认是区分大小写。因此，数据库名、表名、字段名，都不允许出现任何大写字母，避免节外生枝。
正例： aliyun _ admin ， rdc _ config ， level 3_ name
反例： AliyunAdmin ， rdcConfig ， level _3_ name

3. 【强制】表名不使用复数名词。
说明：表名应该仅仅表示表里面的实体内容，不应该表示实体数量，对应于 DO 类名也是单数形式，符合表达习惯。

4. 【强制】禁用保留字，如 desc 、 range 、 match 、 delayed,order,field 等，请参考 MySQL 官方保留字。

5. 【强制】主键索引名为 pk_ 字段名；唯一索引名为 uk _字段名 ； 普通索引名则为 idx _字段名。
说明： pk_ 即 primary key；uk _ 即 unique key；idx _ 即 index 的简称。

6. 【强制】小数类型为 decimal ，禁止使用 float 和 double 。
说明： float 和 double 在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确的结果。如果存储的数据范围超过 decimal 的范围，建议将数据拆成整数和小数分开存储。

7. 【强制】如果存储的字符串长度几乎相等，使用 char 定长字符串类型。

8. 【强制】 varchar 是可变长字符串，不预先分配存储空间，长度不要超过 5000，如果存储长度大于此值，定义字段类型为 text ，独立出来一张表，用主键来对应，避免影响其它字段索引效率。

9. 【强制】表必备三字段： id ， gmt _ create ， gmt _ modified 。
说明：其中 id 必为主键，类型为 bigint unsigned 、单表时自增、步长为 1。 gmt_create, gmt_modified 的类型均为 datetime 类型，前者现在时表示主动创建，后者过去分词表示被动更新。

10. 【推荐】表的命名最好是加上“业务名称_表的作用”。
正例： alipay _ task / force _ project / trade _ config

11. 【推荐】库名与应用名称尽量一致。

12. 【推荐】如果修改字段含义或对字段表示的状态追加时，需要及时更新字段注释。

13. 【推荐】字段允许适当冗余，以提高查询性能，但必须考虑数据一致。冗余字段应遵循：

1 ） 不是频繁修改的字段。
2 ） 不是 varchar 超长字段，更不能是 text 字段。
正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

14. 【推荐】单表行数超过 500 万行或者单表容量超过 2 GB ，才推荐进行分库分表。
说明：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

15. 【参考】合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。
正例：如下表，其中无符号值可以避免误存负数，且扩大了表示范围

索引规约

1. 【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。
说明：不要以为唯一索引影响了 insert 速度，这个速度损耗可以忽略，但提高查找速度是明显的 ； 另外，即使在应用层做了非常完善的校验控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

2. 【强制】超过三个表禁止 join 。需要 join 的字段，数据类型必须绝对一致 ； 多表关联查询时，保证被关联的字段需要有索引。
说明：即使双表 join 也要注意表索引、 SQL 性能。

3. 【强制】在 varchar 字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度即可。
说明：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为 20 的索引，区分度会高达 90%以上，可以使用 count(distinct left(列名, 索引长度)) / count(*) 的区分度来确定。

4. 【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。
说明：索引文件具有 B - Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

5. 【推荐】如果有 order by 的场景，请注意利用索引的有序性。 order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现 file _ sort 的情况，影响查询性能。
正例： where a =? and b =? order by c; 索引： a _ b _ c
反例：索引中有范围查找，那么索引有序性无法利用，如： WHERE a >10 ORDER BY b; 索引 a _ b 无法排序。

6. 【推荐】利用覆盖索引来进行查询操作，避免回表。
说明：如果一本书需要知道第 11 章是什么标题，会翻开第 11 章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。
正例：能够建立索引的种类分为主键索引、唯一索引、普通索引三种，而覆盖索引只是一种查询的一种效果，用 explain 的结果， extra 列会出现： using index 。

7. 【推荐】利用延迟关联或者子查询优化超多分页场景。
说明： MySQL 并不是跳过 offset 行，而是取 offset + N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。
正例：先快速定位需要获取的 id 段，然后再关联：

```
SELECT a.* FROM 表 1 a, (select id from 表 1 where 条件 LIMIT 100000,20 ) b where a.id=b
.id

8. 【推荐】 SQL 性能优化的目标：至少要达到 range 级别，要求是 ref 级别，如果可以是 consts
最好。
说明:
1 ）consts 单表中最多只有一个匹配行 （ 主键或者唯一索引 ） ，在优化阶段即可读取到数据。
2 ）ref 指的是使用普通的索引 （normal index） 。
3 ）range 对索引进行范围检索。
反例： explain 表的结果， type = index ，索引物理文件全扫描，速度非常慢，这个 index 级
别比较 range 还低，与全表扫描是小巫见大巫。

9. 【推荐】建组合索引的时候，区分度最高的在最左边。
正例：如果 where a =? and b =? ，如果 a 列的几乎接近于唯一值，那么只需要单建 idx _ a
索引即可。
说明：存在非等号和等号混合时，在建索引时，请把等号条件的列前置。如： where c >? and d =? 那么即
使 c 的区分度更高，也必须把 d 放在索引的最前列，即索引 idx_d_c。

10. 【推荐】防止因字段类型不同造成的隐式转换，导致索引失效。

11. 【参考】创建索引时避免有如下极端误解：
1 ） 宁滥勿缺。认为一个查询就需要建一个索引。
2 ） 宁缺勿滥。认为索引会消耗空间、严重拖慢更新和新增速度。
3 ） 抵制惟一索引。认为业务的惟一性一律需要在应用层通过“先查后插”方式解决
```

SQL 语句

1. 【强制】不要使用 count(列名) 或 count(常量) 来替代 count(*) ， count(*) 是 SQL 92 定义的
标准统计行数的语法，跟数据库无关，跟 NULL 和非 NULL 无关。
说明： count(*) 会统计值为 NULL 的行，而 count(列名) 不会统计此列为 NULL 值的行。

2. 【强制】 count(distinct col) 计算该列除 NULL 之外的不重复行数，注意 count(distinct col 1， col 2) 如果其中一列全为 NULL ，那么即使另一列有不同的值，也返回为 0。

3. 【强制】当某一列的值全是 NULL 时， count(col) 的返回结果为 0，但 sum(col) 的返回结果为 NULL ，因此使用 sum() 时需注意 NPE 问题。
正例：可以使用如下方式来避免 sum 的 NPE 问题： SELECT IF(ISNULL(SUM(g)) ,0, SUM(g)) FROM table;

4. 【强制】使用 ISNULL() 来判断是否为 NULL 值。
说明： NULL 与任何值的直接比较都为 NULL。
1 ） NULL<>NULL 的返回结果是 NULL ，而不是 false 。
2 ） NULL=NULL 的返回结果是 NULL ，而不是 true 。
3 ） NULL<>1 的返回结果是 NULL ，而不是 true 。
5. 【强制】 在代码中写分页查询逻辑时，若 count 为 0 应直接返回，避免执行后面的分页语句。
6. 【强制】不得使用外键与级联，一切外键概念必须在应用层解决。
说明：以学生和成绩的关系为例，学生表中的 student _ id 是主键，那么成绩表中的 student _ id 则为外键。如果更新学生表中的 student _ id ，同时触发成绩表中的 student _ id 更新，即为级联更新。外键与级联更新适用于单机低并发，不适合分布式、高并发集群 ； 级联更新是强阻塞，存在数据库更新风暴的风险 ； 外键影响数据库的插入速度。
7. 【建议】不使用存储过程，存储过程难以调试和扩展，更没有移植性。

8. 【强制】数据订正（特别是删除、修改记录操作）时，要先 select ，避免出现误删除，确认无误才能执行更新语句。

9. 【推荐】 in 操作能避免则避免，若实在避免不了，需要仔细评估 in 后边的集合元素数量，控制在 1000 个之内。

10. 【参考】如果有国际化需要，所有的字符存储与表示，均以 utf -8 编码，注意字符统计函数的区别。
说明：
SELECT LENGTH("轻松工作")； 返回为 12
SELECT CHARACTER _ LENGTH("轻松工作")； 返回为 4
如果需要存储表情，那么选择 utf8mb 4 来进行存储，注意它与 utf -8 编码的区别。

11. 【参考】 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统和事务日志资源少，但 TRUNCATE 无事务且不触发 trigger ，有可能造成事故，故不建议在开发代码中使用此语句。
说明： TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同