PHP之道---PHP基础知识（译）

Mar 08, 2013

原文：PHP: The Right Way - The Basics

译者：youngsterxyf

比较操作符

比较操作符往往是PHP的一个被忽视的方面，这会导致很多意想不到的结果。其中的一个问题源于严格比较（布尔值为整数的比较）。

```
<?php
$a = 5;    // 5为一个整数

var_dump($a == 5);     //比较值；返回true
var_dump($a == '5');       //比较值（忽略类型）；返回true
var_dump($a === 5);        //比较类型/值（整数 vs. 整数）；返回true
var_dump($a === '5');      //比较类型/值（整数 vs. 整数）；返回false

/**
  * 严格比较
  */
if (strpos('testing', 'test')) {    // 在位置0找到'test'，0被解释为布尔值'false'
    // code...
}

vs.

if (strpos('testing', 'test') !== false) {  // true，因为做了严格比较（0 !== false）
    // code...
}
?>
```

- 比较操作符
- 比较列表

条件语句

If语句

在函数或类中使用'if/else'之时，有个常见的误解—'else'必须一起使用以声明潜在的结果。然而，如果，结果是定义返回值，则'else'是不需要的，因为'return'会结束函数，使得'else'变得毫无意义。

```
<?php
function test($a)
{
    if ($a) {
        return true;
    } else {
```

```
        return false;
    }
}

vs.

function test($a)
{
    if ($a) {
        return true;
    }
    return false;       // 不需要else分支
}
?>
```

- If语句

Switch语句

Switch语句是一种避免输入无穷尽的if和elseif的绝妙方式，但需要注意几点：

- Switch语句仅比较值，并不关心类型（等价于'==' ）
- 逐个分支地迭代直到找到一个匹配项。如果没找到匹配项，则使用缺省(default)分支（如果定义了）
- 若匹配项的代码体没有'break'语句，则会继续执行接下来的每个分支，直到遇到一个break/return语句
- 在函数内，使用'return'可以减少'break'的使用，因为'return'能够结束函数

```
<?php
$answer = test(2);      // 'case 2'和'case3'的代码体会得到执行

function test($a)
{
    switch ($a) {
        case 1:
            // code...
            break;      // break用于结束switch语句
        case 2:
            // code...  // 没有break，继续比较'case 3'
        case 3:
            // code...
            return $result;     // 当前位置在函数内，'return'会结束函数
        default:
            // code...
            return $error;
    }
}
?>
```

- Switch语句
- PHP switch

全局命名空间

使用命名空间之时，你可能发现内置函数被你所写的函数覆盖了。推荐在全局函数的函数名之前添加一个反斜杠来修正这个问题。

```
<?php
```

```php
namespace phptherightway;

function fopen()
{
    $file = \fopen();      // 我们的函数名与内部函数的函数名相同。
                           // 通过添加'\'指定从全局命名空间执行函数
}

function array()
{
    $iterator = new \ArrayIterator();      // ArrayIterator是一个内置类。
                                           // 若类名之前没有一个反斜杠，解释器会试图在你的命名空间中解析它
}
?>
```

- 全局命名空间
- 全局规则

字符串

拼接

* 如果代码行超过了推荐的行长度（120个字符），那么应该考虑拼接代码行

* 为了便于阅读，最好使用拼接操作符而不是拼接赋值操作符

* 在变量原本的命名空间内，当拼接使用了新行，则应该缩进

```php
<?php
$a = 'Multi-line example';     // 拼接赋值操作符(.=)
$a .= "\n";
$a .= 'of what not to do';

vs.

$a = 'Multi-line example'      // 拼接操作符(.)
   . "\n"                      // 缩进新行
   . 'of what to do';
?>
```

- 字符串操作符

字符串类型

字符串类型在PHP社区内是个不变的特性，但希望本节内容能够解释清楚字符串类型之间的区别以及各自的好处/用法。

单引号

单引号是创建字符串最简单的方式，并且通常执行速度也是最快的，因为PHP不会解析这种字符串（不解析其中是否存在变量），所以单引号最适用于：

* 不需要解析的字符串

* 将变量写为纯文本值（Writing of a variable into plain text）

```php
<?php
```

```
echo 'This is my string, look at how pretty it is.';       //不需要解析一个简单的字符串

/**
 * 输出:
 *
 * This is my string, look at how pretty it is.
 */
?>
```

- 单引号

双引号

双引号是字符串处理的瑞士军刀，但执行速度比较慢，因为字符串要经过解析。双引号最适用于：

* 转义字符串

* 内含多个变量和纯文本的字符串

* 压缩多行拼接，提高可读性

```
<?php
echo 'phptherightway is ' . $adjective . '.'    // 一个单引号的使用示例，
   . "\n"                                       // 为变量和转义字符串使用了多行拼接
   . 'I love learning ' . $code . '!';

vs.

echo "phptherightway is $adjective.\n I love learning $code!";      // 没有使用多行拼接，
?>                                                                  // 双引号允许我们使用可解析的字符串
```

使用双引号创建的字符串中包含变量时，经常出现变量名与后面另一个字符相接触的情况，从而导致PHP不解析该变量，因为它被”伪装”起来了。为了解决这个问题，可以使用一对大括号把变量包围起来。

```
<?php
$juice = 'plum';
echo "I drank some juice made of $juices";      // $juice得不到解析

vs.

$juice = 'plum';
echo "I drank some juice made of{$juice}s";     // $juice得到解析

/**
 * 大括号内的复杂变量也能得到解析
 */

$juice = array('apple', 'orange', 'plum');
echo "I drank some juice made of {$juice[1]}s";     // $juice[1]将得到解析
?>
```

- 双引号

Nowdoc语法

PHP 5.3引入了Nowdoc语法，其行为与单引号相同，除了她适用于多行字符串的使用，而不需要拼接。

```
<?php
$str = <<<'EOD'        // 通过<<<初始化
```

```
Example of string
spanning multiple lines
using nowdoc syntax.
$a does not parse.
EOD;                    // 'EOD'关闭符必须单独一行，并且处于最左边

/**
  * 输出:
  *
  * Example of string
  * spanning multiple lines
  * using nowdoc syntax.
  * $a does not parse.
  */
?>
```

- Nowdoc语法

Heredoc语法

Heredoc语法的行为与双引号相同，除了它适用于多行字符串的使用，而不需要拼接。

```
<?php
$a = 'Variables';

$str = <<<EOD       // 使用<<<初始化
Example of string
spanning multiple lines
using heredoc syntax.
$a are parsed.
EOD;                    // 'EOD'关闭符必须单独一行，且处于最左边位置

/**
  * 输出：
  *
  * Example of string
  * spanning multiple lines
  * using heredoc syntax.
  * Variables are parsed.
  */
?>
```

- Heredoc语法

三元操作符

三元操作符是一种压缩代码的好方式，但经常被滥用。当需要多层或嵌套使用三元操作符时，建议一行代码仅使用一次三元操作符以提高代码可读性。

```
<?php
$a = 5;
echo ($a == 5) ? 'yay' : 'nay';

vs.

// 嵌套三元操作符
$b = 10;
echo ($a) ? ($a == 5) ? 'yay' : 'nay' : ($b == 10) ? 'excessive' : ':(';    // 过度嵌套，牺牲了可读性
?>
```

使用三元操作符'返回'一个值需使用正确的语法。

```
<?php
$a = 5;
echo ($a == 5) ? return true : return false;       //这个例子会抛出错误

vs.

$a = 5;
return ($a == 5) ? 'yay' : 'nope';          //这个例子会返回'yay'
?>
```

- 三元操作符

变量声明

有时，程序员会试图通过将预定义变量声明为一个不同的名字使得代码更加"干净"。事实上，这样会让脚本的内存消耗加倍。如下例子，我们假设一个示例文本字符串包含1MB的数据，通过拷贝这个变量，脚本执行时就会增加到2MB。

```
<?php
$about = 'A very long string of text';      //使用了2MB内存
echo $about;

vs.

echo 'A very long string of text';          //仅使用1MB内存
?>
```

- 性能技巧

PHP 翻译

← JavaScript：继承和原型链（译） 按 j 或 k 翻页 电脑重装记 →