PHP 教程

PHP 是一种创建动态交互性站点的强有力的服务器端脚本语言。

PHP 是免费的，并且使用非常广泛。同时，对于像微软 ASP 这样的竞争者来说，PHP 无疑是另一种高效率的选项。

适用于PHP初学者的学习线路和建议

PHP 开发工具推荐

PHP 在线工具

通过实例学习 PHP

我们的 PHP 在线实例让您能够更简单的学习 PHP，实例中包含了 PHP 的源码及运行结果。

实例

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "Hello World!";
?>

</body>
</html>
```

运行实例 »

点击"运行实例"按钮查看在线实例运行结果。

PHP 入门视频教程

您的浏览器不支持Video标签。

PHP 参考手册

在菜鸟教程中，您会发现所有 PHP 函数的完整参考手册：

- Array 函数
- Calendar 函数
- cURL 函数
- Date 函数
- Directory 函数
- Error 函数
- Filesystem 函数
- Filter 函数
- FTP 函数
- HTTP 函数
- LibXML 函数
- Mail 函数

PHP Zip File 函数 PHP 简介

点我分享笔记

反馈/建议

PHP 教程 PHP 安装

PHP 简介

PHP 是服务器端脚本语言。

您应当具备的基础知识

在继续学习之前，您需要对以下知识有基本的了解：

- HTML
- CSS

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

PHP 是什么？

PHP（全称：PHP：Hypertext Preprocessor，即"PHP：超文本预处理器"）是一种通用开源脚本语言。

PHP 脚本在服务器上执行。

PHP 可免费下载使用。

PHP 对初学者而言简单易学。

PHP 也为专业的程序员提供了许多先进的功能。

PHP 文件是什么？

PHP 文件可包含文本、HTML、JavaScript代码和 PHP 代码

PHP 代码在服务器上执行，结果以纯 HTML 形式返回给浏览器

PHP 文件的默认文件扩展名是 ".php"

PHP 能做什么？

PHP 可以生成动态页面内容

PHP 可以创建、打开、读取、写入、关闭服务器上的文件

PHP 可以收集表单数据

PHP 可以发送和接收 cookies

PHP 可以添加、删除、修改您的数据库中的数据

PHP 可以限制用户访问您的网站上的一些页面

PHP 可以加密数据

通过 PHP，您不再限于输出 HTML。您可以输出图像、PDF 文件，甚至 Flash 电影。您还可以输出任意的文本，比如 XHTML 和 XML。

为什么使用 PHP?

PHP 可在不同的平台上运行（Windows、Linux、Unix、Mac OS X 等）

PHP 与目前几乎所有的正在被使用的服务器相兼容（Apache、IIS 等）

PHP 提供了广泛的数据库支持

PHP 是免费的，可从官方的 PHP 资源下载它： www.php.net

PHP 易于学习，并可高效地运行在服务器端

PHP 教程 PHP 安装

点我分享笔记

反馈/建议

PHP 简介 PHP 语法

PHP 安装

您需要做什么？

为了开始使用 PHP，您可以：

找一个支持 PHP 和 MySQL 的 Web 主机

在您自己的 PC 机上安装 Web 服务器，然后安装 PHP 和 MySQL

使用支持 PHP 的 Web 主机

如果您的服务器支持 PHP，那么您不需要做任何事情。

只要在您的 web 目录中创建 .php 文件即可，服务器将自动为您解析这些文件。

您不需要编译任何软件，或安装额外的工具。

由于 PHP 是免费的，大多数的 Web 主机都提供对 PHP 的支持。

在您自己的 PC 机上建立 PHP

然而，如果您的服务器不支持 PHP，您必须：

安装 Web 服务器

安装 PHP

安装数据库，比如 MySQL

官方 PHP 网站（PHP.net）有 PHP 的安装说明： http://php.net/manual/en/install.php

---

PHP 服务器组件

对于初学者建议使用集成的服务器组件，它已经包含了 PHP、Apache、Mysql 等服务,免去了开发人员将时间花费在繁琐的配置环境过程。

WampServer

Windows 系统可以使用 WampServer，下载地址：http://www.wampserver.com/，支持32位和64位系统，根据自己的系统选择版本。

WampServer 安装也简单，你只需要一直点击 "**Next**" 就可以完成安装了。

XAMPP

XAMPP 支持 Mac OS 和 Windows 系统，下载地址：https://www.apachefriends.org/zh_cn/index.html。

IDE (Integrated Development Environment,集成开发环境)

Eclipse for PHP（免费）

Eclipse 是一个开放源代码的、基于Java的可扩展开发平台(如果未安装JDK，则需要先 下载 JDK 安装)。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。幸运的是，Eclipse 附带了一个标准的插件集，包括Java开发工具（Java Development Kit，JDK）。

支持 Windows、Linux 和 Mac OS 平台。

Eclipse for PHP 官方下载地址：http://www.eclipse.org/downloads/packages/eclipse-php-developers/heliosr

PhpStorm（收费）

PhpStorm是一个轻量级且便捷的PHP IDE，其旨在提供用户效率，可深刻理解用户的编码，提供智能代码补全，快速导航以及即时错误检查。

PhpStorm 非常适合于PHP开发人员及前端工程师。提供诸于：智能HTML/CSS/JavaScript/PHP编辑、代码质量分析、版本控制集成（SVN、GIT）、调试和测试等功能。

支持 Windows、Linux 和 Mac OS 平台。

PhpStorm 官方下载地址：http://www.jetbrains.com/phpstorm/download/

在 Cloud Studio 中运行 PHP 程序

> *PHP 是一种运行在服务端的开源，解释和面向对象的脚本语言。由于是解释型语言、不需要编译，因此比其他语言编写脚本更快。并且具有开源、独立平台、兼容性、嵌入式脚本等特点，越来越多的被用于 Web 开发。大多数情况下，运行在服务端的 PHP 脚本要依赖 Apache、Nginx 这样的 Web 服务器来运行。但在这里我向你推荐一个更方便的开发环境：运用基于腾讯云主机的在线云端开发工具 **Cloud Studio + PHP 内置的 Web 服务器** 来快速开始你的 PHP 项目。*

step1：访问 Cloud Studio，注册/登录账户。

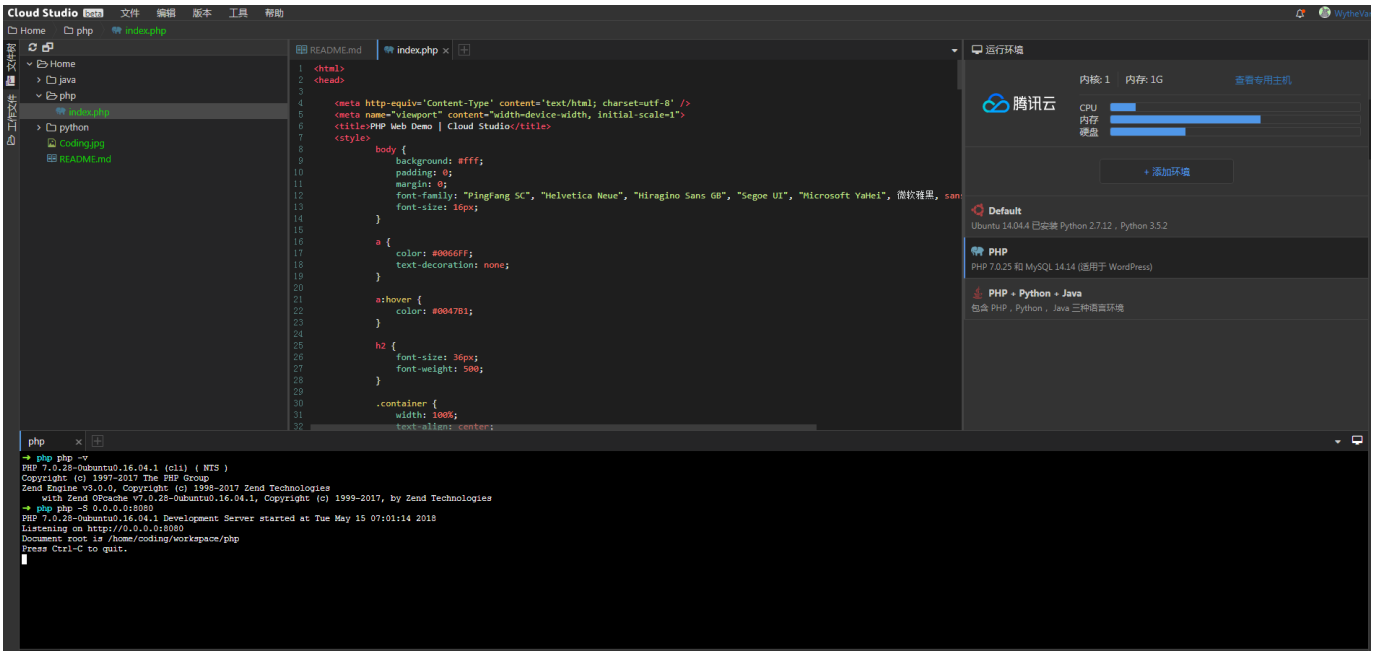
step2：在右侧的运行环境菜单选择："PHP 运行环境"

step3：在左侧代码目录中新建 PHP 代码目录编写PHP代码

step4：进入 PHP 代码目录运行 `php -S 0.0.0.0:8080` 启动 PHP 内置的 web server 服务器监听8080端口。

step5：点击最右侧的【访问链接】选项卡，在访问链接面板中填写端口号为：8080，点击创建链接，即可点击生成的链接访问我们的 PHP 项目

Tips: 从终端中输入命令可以看出 **Cloud Studio** 为我们集成了 **Ubuntu16.04.1 + PHP7.0.28(NTS)** 的开发环境：



有任何疑问，可以查阅帮助文档

□ PHP 简介 PHP 语法 □

□点我分享笔记

反馈/建议

□

□ PHP 安装 PHP 变量 □

PHP 语法

PHP 脚本在服务器上执行，然后将纯 HTML 结果发送回浏览器。

基本的 PHP 语法

PHP 脚本可以放在文档中的任何位置。

PHP 脚本以 **<?php** 开始，以 **?>** 结束：

```
<?php
// PHP 代码
?>
```

PHP 文件的默认文件扩展名是 ".php"。

PHP 文件通常包含 HTML 标签和一些 PHP 脚本代码。

下面，我们提供了一个简单的 PHP 文件实例，它可以向浏览器输出文本 "Hello World!"：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

运行实例 »

PHP 中的每个代码行都必须以分号结束。分号是一种分隔符，用于把指令集区分开来。

通过 PHP，有两种在浏览器输出文本的基础指令：**echo** 和 **print**。

PHP 中的注释

实例

```
<!DOCTYPE html>
<html>
<body>

<?php
// 这是 PHP 单行注释

/*
这是
PHP 多行
注释
*/
?>

</body>
</html>
```

运行实例 »

□ PHP 安装　　PHP 变量 □

□点我分享笔记

反馈/建议

PHP 变量

变量是用于存储信息的"容器"：

实例

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

运行实例 »

与代数类似

x=5

y=6

z=x+y

在代数中，我们使用字母（如 x），并给它赋值（如 5）。

从上面的表达式 z=x+y，我们可以计算出 z 的值为 11。

在 PHP 中，这些字母被称为**变量**。

变量是用于存储数据的容器。

PHP 变量

与代数类似，可以给 PHP 变量赋予某个值（x=5）或者表达式（z=x+y）。

变量可以是很短的名称（如 x 和 y）或者更具描述性的名称（如 age、carname、totalvolume）。

PHP 变量规则：

- 变量以 $ 符号开始，后面跟着变量的名称
- 变量名必须以字母或者下划线字符开始
- 变量名只能包含字母数字字符以及下划线（A-z、0-9 和 _ ）
- 变量名不能包含空格
- 变量名是区分大小写的（$y 和 $Y 是两个不同的变量）

PHP 语句和 PHP 变量都是区分大小写的。

创建（声明）PHP 变量

PHP 没有声明变量的命令。

变量在您第一次赋值给它的时候被创建：

实例

```
<?php
$txt="Hello world!";
$x=5;
$y=10.5;
```

```
?>
```

运行实例 »

在上面的语句执行中，变量 **txt** 将保存值 **Hello world!**，且变量 **x** 将保存值 **5**。

注释： 当您赋一个文本值给变量时，请在文本值两侧加上引号。

PHP 是一门弱类型语言

在上面的实例中，我们注意到，不必向 PHP 声明该变量的数据类型。

PHP 会根据变量的值，自动把变量转换为正确的数据类型。

在强类型的编程语言中，我们必须在使用变量前先声明（定义）变量的类型和名称。

PHP 变量作用域

变量的作用域是脚本中变量可被引用/使用的部分。

PHP 有四种不同的变量作用域：

- local
- global
- static
- parameter

局部和全局作用域

在所有函数外部定义的变量，拥有全局作用域。除了函数外，全局变量可以被脚本中的任何部分访问，要在一个函数中访问一个全局变量，需要使用 global 关键字。

在 PHP 函数内部声明的变量是局部变量，仅能在函数内部访问：

实例

```php
<?php
$x=5; // 全局变量

function myTest()
{
    $y=10; // 局部变量
    echo "<p>测试函数内变量:<p>";
    echo "变量 x 为: $x";
    echo "<br>";
    echo "变量 y 为: $y";
}

myTest();

echo "<p>测试函数外变量:<p>";
echo "变量 x 为: $x";
echo "<br>";
echo "变量 y 为: $y";
?>
```

运行实例 »

在以上实例中 myTest() 函数定义了 $x 和 $y 变量。 $x 变量在函数外声明，所以它是全局变量， $y 变量在函数内声明所以它是局部变量。

当我们调用myTest()函数并输出两个变量的值, 函数将会输出局部变量 $y 的值，但是不能输出 $x 的值，因为 $x 变量在函数外定义，无法在函数内使用，如果要在一个函数中访问一个全局变量，需要使用 global 关键字。

然后我们在myTest()函数外输出两个变量的值，函数将会输出全局部变量 $x 的值，但是不能输出 $y 的值，因为 $y 变量在函数中定义，属于局部变量。

Note 你可以在不同函数中使用相同的变量名称，因为这些函数内定义的变量名是局部变量，只作用于该函数内。

PHP global 关键字

global 关键字用于函数内访问全局变量。

在函数内调用函数外定义的全局变量，我们需要在函数中的变量前加上 global 关键字：

实例

```php
<?php
$x=5;
$y=10;
function myTest()
{
global $x,$y;
$y=$x+$y;
}
myTest();
echo $y; // 输出 15
?>
```

运行实例 »

PHP 将所有全局变量存储在一个名为 $GLOBALS[*index*] 的数组中。 *index* 保存变量的名称。这个数组可以在函数内部访问，也可以直接用来更新全局变量。

上面的实例可以写成这样：

实例

```php
<?php
$x=5;
$y=10;
function myTest()
{
$GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
}
myTest();
echo $y;
?>
```

运行实例 »

Static 作用域

当一个函数完成时，它的所有变量通常都会被删除。然而，有时候您希望某个局部变量不要被删除。

要做到这一点，请在您第一次声明变量时使用 **static** 关键字：

实例

```php
<?php
function myTest()
{
static $x=0;
echo $x;
$x++;
}
myTest();
myTest();
myTest();
?>
```

运行实例 »

然后，每次调用该函数时，该变量将会保留着函数前一次被调用时的值。

注释： 该变量仍然是函数的局部变量。

参数作用域

参数是通过调用代码将值传递给函数的局部变量。

参数是在参数列表中声明的，作为函数声明的一部分：

实例

```
<?php
function myTest($x)
{
echo $x;
}
myTest(5);
?>
```

我们将在 PHP 函数 章节对它做更详细的讨论。

PHP 语法　　PHP 字符串变量

1 篇笔记

#1

写笔记

1、定义在函数外部的就是全局变量，它的作用域从定义处一直到文件结尾。
2、函数内定义的变量就是局部变量，它的作用域为函数定义范围内。
3、函数之间存在作用域互不影响。
4、函数内访问全局变量需要 `global` 关键字或者使用 `$GLOBALS[index]` 数组
在 php 中函数是有独立的作用域，所以局部变量会覆盖全局变量，即使局部变量中没有全局变量相同的变量，也会被覆盖。如下：

```
<?php

$a=5;

$b=3;

function t()

{

    echo $a-$b; // 输出 0

}

t();

?>
```

要想在函数中直接使用全局变量可以通过 `global` 关键字声明或者通过 php 中自定义的 `$GLOBALS` 数组获取:

```
<?php

$a=5;

$b=3;

function t1()

{

    global $a,$b;

    echo $a-$b;  // 输出 2
```

```
}

t1();



echo PHP_EOL;



function t2()

{

    echo $GLOBALS['a']-$GLOBALS['b'];  // 输出 2

}

t2();

?>
```

佛系coder5个月前 (04-26)

反馈/建议

PHP 5 echo 和 print 语句

在 PHP 中有两个基本的输出方式： echo 和 print。

本章节中我们会详细讨论两个语句的用法，并在实例中演示如何使用 echo 和 print。

PHP echo 和 print 语句

echo 和 print 区别:

- echo - 可以输出一个或多个字符串
- print - 只允许输出一个字符串，返回值总为 1

提示：echo 输出的速度比 print 快， echo 没有返回值，print有返回值1。

PHP echo 语句

echo 是一个语言结构，使用的时候可以不用加括号，也可以加上括号： echo 或 echo()。

显示字符串

下面的实例演示了如何使用 echo 命令输出字符串（字符串可以包含 HTML 标签）：

实例

```php
<?php
echo "<h2>PHP 很有趣!</h2>";
echo "Hello world!<br>";
echo "我要学 PHP!<br>";
echo "这是一个", "字符串，", "使用了", "多个", "参数。";
?>
```

尝试一下 »

显示变量

下面的实例演示了如何使用 echo 命令输出变量和字符串：

实例

```php
<?php
$txt1="学习 PHP";
$txt2="RUNOOB.COM";
$cars=array("Volvo","BMW","Toyota");
echo $txt1;
echo "<br>";
echo "在 $txt2 学习 PHP ";
echo "<br>";
echo "我车的品牌是 {$cars[0]}";
?>
```

尝试一下 »

PHP print 语句

print 同样是一个语言结构，可以使用括号，也可以不使用括号： print 或 print()。

显示字符串

下面的实例演示了如何使用 print 命令输出字符串（字符串可以包含 HTML 标签）：

实例

```php
<?php
print "<h2>PHP 很有趣!</h2>";
print "Hello world!<br>";
print "我要学习 PHP!";
?>
```

尝试一下 »

显示变量

下面的实例演示了如何使用 print 命令输出变量和字符串：

实例

```php
<?php
$txt1="学习 PHP";
$txt2="RUNOOB.COM";
$cars=array("Volvo","BMW","Toyota");
print $txt1;
print "<br>";
print "在 $txt2 学习 PHP ";
print "<br>";
print "我车的品牌是 {$cars[0]}";
?>
```

尝试一下 »

PHP EOF(heredoc) 使用说明

PHP EOF(heredoc)是一种在命令行shell（如sh、csh、ksh、bash、PowerShell和zsh）和程序语言（像Perl、PHP、Python和Ruby）里定义一个字串的方法。

使用概述：

1. 必须后接分号，否则编译通不过。
2. **EOF** 可以用任意其它字符代替，只需保证结束标识与开始标识一致。
3. **结束标识必须顶格独自占一行(即必须从行首开始，前后不能衔接任何空白和字符)。**
4. 开始标识可以不带引号或带单双引号，不带引号与带双引号效果一致，解释内嵌的变量和转义符号，带单引号则不解释内嵌的变量和转义符号。
5. 当内容需要内嵌引号（单引号或双引号）时，不需要加转义符，本身对单双引号转义，此处相当与q和qq的用法。

实例

```
<?php
echo <<<EOF
<h1>我的第一个标题</h1>
<p>我的第一个段落。</p>
EOF;
// 结束需要独立一行且前后不能空格
?>
```

注意：

1.以 `<<<EOF` 开始标记开始，以 `EOF` 结束标记结束，结束标记必须顶头写，不能有缩进和空格，且在结束标记末尾要有分号 。

2.开始标记和结束标记相同，比如常用大写的 `EOT、EOD、EOF` 来表示，但是不只限于那几个(也可以用：JSON、HTML等)，只要保证开始标记和结束标记不在正文中出现即可。

3.位于开始标记和结束标记之间的变量可以被正常解析，但是函数则不可以。在 heredoc 中，变量不需要用连接符 `.` 或 `,` 来拼接，如下：

实例

```
<?php
$name="runoob";
$a= <<<EOF
"abc"$name
"123"
EOF;
// 结束需要独立一行且前后不能空格
echo $a;
?>
```

1 篇笔记
#1

写笔记

1.PHP 定界符 `EOF` 的作用就是按照原样，包括换行格式什么的，输出在其内部的东西；
2.在 PHP 定界符 `EOF` 中的任何特殊字符都不需要转义；
3.PHP 定界符 `EOF`

马鹿7个月前 (03-11)

反馈/建议

PHP 5 echo/print 语句 PHP 5 常量

PHP 5 数据类型

String（字符串），Integer（整型），Float（浮点型），Boolean（布尔型），Array（数组），Object（对象），NULL（空值）。

PHP 字符串

一个字符串是一串字符的序列，就像 "Hello world!"。

你可以将任何文本放在单引号和双引号中：

实例

```
<?php
$x = "Hello world!";
echo $x;
echo "<br>";
$x = 'Hello world!';
echo $x;
?>
```

尝试一下 »

PHP 整型

整数是一个没有小数的数字。

整数规则:

- 整数必须至少有一个数字 (0-9)
- 整数不能包含逗号或空格
- 整数是没有小数点的
- 整数可以是正数或负数
- 整型可以用三种格式来指定：十进制， 十六进制（ 以 0x 为前缀）或八进制（前缀为 0）。

在以下实例中我们将测试不同的数字。

PHP var_dump() 函数返回变量的数据类型和值：

实例

```php
<?php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // 负数
var_dump($x);
echo "<br>";
$x = 0x8C; // 十六进制数
var_dump($x);
echo "<br>";
$x = 047; // 八进制数
var_dump($x);
?>
```

尝试一下 »

PHP 浮点型

浮点数是带小数部分的数字，或是指数形式。

在以下实例中我们将测试不同的数字。 PHP var_dump() 函数返回变量的数据类型和值：

实例

```php
<?php
$x = 10.365;
var_dump($x);
echo "<br>";
$x = 2.4e3;
var_dump($x);
echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

尝试一下 »

PHP 布尔型

布尔型可以是 TRUE 或 FALSE。

```php
$x=true;
$y=false;
```

布尔型通常用于条件判断。在接下来的章节中你会学到更多关于条件控制的教程。

PHP 数组

数组可以在一个变量中存储多个值。

在以下实例中创建了一个数组， 然后使用 PHP var_dump() 函数返回数组的数据类型和值：

实例

```php
<?php
$cars=array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

尝试一下 »

在接下来的章节中你将学到更多关于数组的知识。

PHP 对象

对象数据类型也可以用于存储数据。

在 PHP 中，对象必须声明。

首先，你必须使用class关键字声明类对象。类是可以包含属性和方法的结构。

然后我们在类中定义数据类型，然后在实例化的类中使用数据类型：

实例

```
<?php
class Car
{
var $color;
function __construct($color="green") {
$this->color = $color;
}
function what_color() {
return $this->color;
}
}
?>
```

尝试一下 »

以上实例中PHP关键字this就是指向当前对象实例的指针，不指向任何其他对象或类。

你将会在接下来的章节中学会更多关于对象的知识。

PHP NULL 值

NULL 值表示变量没有值。NULL 是数据类型为 NULL 的值。

NULL 值指明一个变量是否为空值。 同样可用于数据空值和NULL值的区别。

可以通过设置变量值为 NULL 来清空变量数据：

实例

```
<?php
$x="Hello world!";
$x=null;
var_dump($x);
?>
```

尝试一下 »

□ PHP 5 echo/print 语句　　PHP 5 常量 □

□

2 篇笔记

#2

□ 写笔记

□

说明： var_dump() 方法，判断一个变量的类型与长度，并输出变量的数值，如果变量有值，则输出是变量的值，并返回数据类型。显示关于一个或多个表达式的结构信息，包括表达式的类型与值。数组将递归展开值，通过缩进显示其结构。

格式：

```
var_dump ( mixed expression [, mixed expression [, ...]] )
```

注意：用保证 var_dump 中的变量必须是存在的，如果变量不存在则返回 NULL。该函数有输出的功能，因此不必加其它的输出函数。

马鹿7个月前 (03-11)

#1

□

echo，print，print_r，var_dump 的区别

1.echo

输出一个或者多个字符串。

2.print

和 echo 一样，速度 比 echo 慢。

3.print_r

打印关于变量的易于理解的信息,如果给出的是 string、integer 或 float，将打印变量值本身。如果给出的是 array，将会按照一定格式显示键和元素。object 与数组类似。

记住，print_r() 将把数组的指针移到最后边。使用 reset() 可让指针回到开始处。

4.var_dump

此函数显示关于一个或多个表达式的结构信息，包括表达式的类型与值。数组将递归展开值，通过缩进显示其结构。

5.var_dump 和 print_r 的区别

var_dump 返回表达式的类型与值而 print_r 仅返回结果，相比调试代码使用 var_dump 更便于阅读。

Xavier1个月前 (08-31)

反馈/建议

PHP 5 常量

常量值被定义后，在脚本的其他任何地方都不能被改变。

PHP 常量

常量是一个简单值的标识符。该值在脚本中不能改变。

一个常量由英文字母、下划线、和数字组成,但数字不能作为首字母出现。 (常量名不需要加 $ 修饰符)。

注意： 常量在整个脚本中都可以使用。

设置 PHP 常量

设置常量，使用 define() 函数，函数语法如下：

```
bool define ( string $name , mixed $value [, bool $case_insensitive = false ] )
```

该函数有三个参数:

name：必选参数，常量名称，即标志符。

value：必选参数，常量的值。

case_insensitive ：可选参数，如果设置为 TRUE，该常量则大小写不敏感。默认是大小写敏感的。

以下实例我们创建一个 **区分大小写的常量**, 常量值为 "欢迎访问 Runoob.com"：

实例

```
<?php
// 区分大小写的常量名
define("GREETING", "欢迎访问 Runoob.com");
echo GREETING; // 输出 "欢迎访问 Runoob.com"
echo '<br>';
echo greeting; // 输出 "greeting"
?>
```

以下实例我们创建一个 **不区分大小写的常量**, 常量值为 "欢迎访问 Runoob.com"：

实例

```
<?php
```

```php
// 不区分大小写的常量名
define("GREETING", "欢迎访问 Runoob.com", true);
echo greeting; // 输出 "欢迎访问 Runoob.com"
?>
```

常量是全局的

常量在定义后，默认是全局变量，可以在整个运行的脚本的任何地方使用。

以下实例演示了在函数内使用常量，即便常量定义在函数外也可以正常使用常量。

实例

```php
<?php
define("GREETING", "欢迎访问 Runoob.com");
function myTest() {
echo GREETING;
}
myTest(); // 输出 "欢迎访问 Runoob.com"
?>
```

PHP 5 数据类型 PHP JSON

点我分享笔记

反馈/建议

PHP 字符串变量

字符串变量用于存储并处理文本。

PHP 中的字符串变量

字符串变量用于包含有字符的值。

在创建字符串之后，我们就可以对它进行操作了。您可以直接在函数中使用字符串，或者把它存储在变量中。

在下面的实例中，我们创建一个名为 txt 的字符串变量，并赋值为 "Hello world!" 。然后我们输出 txt 变量的值：

实例

```php
<?php
$txt="Hello world!";
echo $txt;
?>
```

运行实例 »

注释： 当您赋一个文本值给变量时，请记得给文本值加上单引号或者双引号。

现在，让我们来看看一些常用的操作字符串的函数和运算符。

PHP 并置运算符

在 PHP 中，只有一个字符串运算符。

并置运算符 (.) 用于把两个字符串值连接起来。

下面的实例演示了如何将两个字符串变量连接在一起：

实例

```
<?php
$txt1="Hello world!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

上面的代码将输出：Hello world! What a nice day!

提示：在上面的代码中，我们已经使用了两次并置运算符。这是由于我们需要在两个字符串之间插入一个空格。

PHP strlen() 函数

有时知道字符串值的长度是很有用的。

strlen() 函数返回字符串的长度（字符数）。

下面的实例返回字符串 "Hello world!" 的长度：

实例

```
<?php
echo strlen("Hello world!");
?>
```

运行实例 »

上面的代码将输出：12

提示：strlen() 常常用在循环和其他函数中，因为那时确定字符串何时结束是很重要的。（例如，在循环中，我们需要在字符串中的最后一个字符之后结束循环。）

PHP strpos() 函数

strpos() 函数用于在字符串内查找一个字符或一段指定的文本。

如果在字符串中找到匹配，该函数会返回第一个匹配的字符位置。如果未找到匹配，则返回 FALSE。

下面的实例在字符串 "Hello world!" 中查找文本 "world"：

实例

```
<?php
echo strpos("Hello world!","world");
?>
```

运行实例 »

上面的代码将输出：6

提示：在上面的实例中，字符串 "world" 的位置是 6。之所以是 6 而不是 7 的原因是，字符串中第一个字符的位置是 0，而不是 1。

完整的 PHP String 参考手册

如需查看所有字符串函数的完整参考手册，请访问我们的 PHP String 参考手册。

该参考手册提供了每个函数的简要描述和应用实例！

PHP 运算符

本章节我们将讨论 PHP 中不同运算符的应用。

在 PHP 中，赋值运算符 = 用于给变量赋值。

在 PHP 中，算术运算符 + 用于把值加在一起。

PHP 算术运算符

运算符	名称	描述	实例	结果
x + y	加	x 和 y 的和	2 + 2	4
x - y	减	x 和 y 的差	5 - 2	3
x * y	乘	x 和 y 的积	5 * 2	10
x / y	除	x 和 y 的商	15 / 5	3
x % y	模（除法的余数）	x 除以 y 的余数	5 % 2 10 % 8 10 % 2	1 2 0
- x	取反	x 取反	-2	-2
a . b	并置	连接两个字符串	"Hi" . "Ha"	HiHa

以下实例演示了使用不同算术运算符得到的不同结果：

实例

```php
<?php
$x=10;
$y=6;
echo ($x + $y); // 输出16
echo '<br>'; // 换行
echo ($x - $y); // 输出4
echo '<br>'; // 换行
echo ($x * $y); // 输出60
echo '<br>'; // 换行
echo ($x / $y); // 输出1.6666666666667
echo '<br>'; // 换行
echo ($x % $y); // 输出4
echo '<br>'; // 换行
echo -$x;
?>
```

尝试一下 »

PHP7+ 版本新增整除运算符 **intdiv()**,使用实例：

实例

```php
<?php
var_dump(intdiv(10, 3));
?>
```

以上实例会输出：

```
int(3)
```

PHP 赋值运算符

在 PHP 中，基本的赋值运算符是 "="。它意味着左操作数被设置为右侧表达式的值。也就是说，"$x = 5" 的值是 5。

运算符	等同于	描述
x = y	x = y	左操作数被设置为右侧表达式的值
x += y	x = x + y	加
x -= y	x = x - y	减
x *= y	x = x * y	乘
x /= y	x = x / y	除
x %= y	x = x % y	模（除法的余数）
a .= b	a = a . b	连接两个字符串

以下实例演示了使用不同赋值运算符得到的不同结果：

实例

```php
<?php
$x=10;
echo $x; // 输出10
$y=20;
$y += 100;
echo $y; // 输出120
$z=50;
$z -= 25;
echo $z; // 输出25
$i=5;
$i *= 6;
echo $i; // 输出30
$j=10;
$j /= 5;
echo $j; // 输出2
$k=15;
$k %= 4;
echo $k; // 输出3
?>
```

尝试一下 »

以下实例演示了使用不同字符串运算符得到的相同结果：

实例

```php
<?php
$a = "Hello";
$b = $a . " world!";
echo $b; // 输出Hello world!
```

```php
$x="Hello";
$x .= " world!";
echo $x; // 输出Hello world!
?>
```

尝试一下 »

PHP 递增/递减运算符

运算符	名称	描述
++ x	预递增	x 加 1，然后返回 x
x ++	后递增	返回 x，然后 x 加 1
– x	预递减	x 减 1，然后返回 x
x –	后递减	返回 x，然后 x 减 1

以下实例演示了使用递增/递减运算符得到的结果：

实例

```php
<?php
$x=10;
echo ++$x; // 输出11
$y=10;
echo $y++; // 输出10
$z=5;
echo --$z; // 输出4
$i=5;
echo $i--; // 输出5
?>
```

尝试一下 »

PHP 比较运算符

比较操作符可以让您比较两个值：

运算符	名称	描述	实例
x == y	等于	如果 x 等于 y，则返回 true	5==8 返回 false
x === y	绝对等于	如果 x 等于 y，且它们类型相同，则返回 true	5==="5" 返回 false
x != y	不等于	如果 x 不等于 y，则返回 true	5!=8 返回 true
x <> y	不等于	如果 x 不等于 y，则返回 true	5<>8 返回 true
x !== y	绝对不等于	如果 x 不等于 y，或它们类型不相同，则返回 true	5!=="5" 返回 true
x > y	大于	如果 x 大于 y，则返回 true	5>8 返回 false
x < y	小于	如果 x 小于 y，则返回 true	5<8 返回 true
x >= y	大于等于	如果 x 大于或者等于 y，则返回 true	5>=8 返回 false
x <= y	小于等于	如果 x 小于或者等于 y，则返回 true	5<=8 返回 true

以下实例演示了使用一些比较运算符得到的不同结果：

实例

```php
<?php
$x=100;
$y="100";
var_dump($x == $y);
```

```php
echo "<br>";
var_dump($x === $y);
echo "<br>";
var_dump($x != $y);
echo "<br>";
var_dump($x !== $y);
echo "<br>";
$a=50;
$b=90;
var_dump($a > $b);
echo "<br>";
var_dump($a < $b);
?>
```

尝试一下 »

PHP 逻辑运算符

运算符	名称	描述	实例
x and y	与	如果 x 和 y 都为 true，则返回 true	x=6 y=3 (x < 10 and y > 1) 返回 true
x or y	或	如果 x 和 y 至少有一个为 true，则返回 true	x=6 y=3 (x==6 or y==5) 返回 true
x xor y	异或	如果 x 和 y 有且仅有一个为 true，则返回 true	x=6 y=3 (x==6 xor y==3) 返回 false
x && y	与	如果 x 和 y 都为 true，则返回 true	x=6 y=3 (x < 10 && y > 1) 返回 true
x || y	或	如果 x 和 y 至少有一个为 true，则返回 true	x=6 y=3 (x==5 || y==5) 返回 false
! x	非	如果 x 不为 true，则返回 true	x=6 y=3 !(x==y) 返回 true

PHP 数组运算符

运算符	名称	描述
x + y	集合	x 和 y 的集合
x == y	相等	如果 x 和 y 具有相同的键/值对，则返回 true
x === y	恒等	如果 x 和 y 具有相同的键/值对，且顺序相同类型相同，则返回 true
x != y	不相等	如果 x 不等于 y，则返回 true
x <> y	不相等	如果 x 不等于 y，则返回 true
x !== y	不恒等	如果 x 不等于 y，则返回 true

以下实例演示了使用一些数组运算符得到的不同结果：

实例

```php
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");
$z = $x + $y; // $x 和 $y 数组合并
var_dump($z);
var_dump($x == $y);
var_dump($x === $y);
var_dump($x != $y);
var_dump($x <> $y);
var_dump($x !== $y);
?>
```

尝试一下 »

三元运算符

另一个条件运算符是"?:"（或三元）运算符 。

语法格式

```
(expr1) ? (expr2) : (expr3)
```

对 expr1 求值为 TRUE 时的值为 expr2，在 expr1 求值为 FALSE 时的值为 expr3。

自 PHP 5.3 起，可以省略三元运算符中间那部分。表达式 expr1 ?: expr3 在 expr1 求值为 TRUE 时返回 expr1，否则返回 expr3。

实例

以下实例中通过判断 $_GET 请求中含有 user 值，如果有返回 $_GET['user']，否则返回 nobody：

实例

```php
<?php
$test = '菜鸟教程';
// 普通写法
$username = isset($test) ? $test : 'nobody';
echo $username, PHP_EOL;
// PHP 5.3+ 版本写法
$username = $test ?: 'nobody';
echo $username, PHP_EOL;
?>
```

```
菜鸟教程

菜鸟教程
```

注意：PHP_EOL 是一个换行符，兼容更大平台。

在 PHP7+ 版本多了一个 NULL 合并运算符 `??`，实例如下：

实例

```php
<?php
// 如果 $_GET['user'] 不存在返回 'nobody'，否则返回 $_GET['user'] 的值
$username = $_GET['user'] ?? 'nobody';
// 类似的三元运算符
$username = isset($_GET['user']) ? $_GET['user'] : 'nobody';
?>
```

组合比较符(PHP7+)

PHP7+ 支持组合比较符，实例如下：

实例

```php
<?php
// 整型
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
// 浮点型
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1
// 字符串
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
?>
```

运算符优先级

下表按照优先级从高到低列出了运算符。同一行中的运算符具有相同优先级，此时它们的结合方向决定求值顺序。

说明：左 = 从左到右，右 = 从右到左。

结合方向	运算符	附加信息
无	clone new	clone 和 new
左	[	array()
右	++ -- ~ (int) (float) (string) (array) (object) (bool) @	类型和递增 / 递减
无	instanceof	类型
右	!	逻辑运算符
左	* / %	算术运算符
左	+ - .	算术运算符和字符串运算符
左	<< >>	位运算符
无	== != === !== <>	比较运算符
左	&	位运算符和引用
左	^	位运算符
左	|	位运算符
左	&&	逻辑运算符
左	||	逻辑运算符
左	? :	三元运算符
右	= += -= *= /= .= %= &= |= ^= <<= >>= =>	赋值运算符
左	and	逻辑运算符
左	xor	逻辑运算符
左	or	逻辑运算符
左	,	多处用到

运算符优先级中，or 和 ||，&& 和 and 都是逻辑运算符，效果一样，但是其优先级却不一样。

实例

```
<?php
// 优先级： && > = > and
// 优先级： || > = > or
$a = 3;
$b = false;
$c = $a or $b;
var_dump($c); // 这里的 $c 为 int 值3，而不是 boolean 值 true
$d = $a || $b;
var_dump($d); //这里的 $d 就是 boolean 值 true
?>
```

以上实例输出结果为：

```
int(3)

bool(true)
```

括号的使用

我们通过括号的配对来明确标明运算顺序，而非靠运算符优先级和结合性来决定，通常能够增加代码的可读性。

实例

```
<?php
// 括号优先运算
$a = 1;
$b = 2;
$c = 3;
$d = $a + $b * $c;
echo $d;
echo "\n";
$e = ($a + $b) * $c; // 使用括号
echo $e;
echo "\n";
?>
```

以上实例输出结果为：

```
7

9
```

□ PHP 字符串变量　　PHP If...Else 语句 □

PHP If...Else 语句

条件语句用于根据不同条件执行不同动作。

PHP 条件语句

当您编写代码时，您常常需要为不同的判断执行不同的动作。您可以在代码中使用条件语句来完成此任务。

在 PHP 中，提供了下列条件语句：

- **if 语句** - 在条件成立时执行代码
- **if...else 语句** - 在条件成立时执行一块代码，条件不成立时执行另一块代码
- **if...elseif....else 语句** - 在若干条件之一成立时执行一个代码块
- **switch 语句** - 在若干条件之一成立时执行一个代码块

PHP - if 语句

if 语句用于**仅当指定条件成立时执行代码**。

语法

```
if (条件)
{
    条件成立时要执行的代码;
}
```

如果当前时间小于 20，下面的实例将输出 "Have a good day!"：

实例

```
<?php
$t=date("H");
if ($t<"20")
{
echo "Have a good day!";
}
?>
```

运行实例 »

PHP - if...else 语句

在条件成立时执行一块代码，条件不成立时执行另一块代码，请使用 if....else 语句。

语法

```
if (条件)
{
条件成立时执行的代码;
}
else
{
条件不成立时执行的代码;
}
```

如果当前时间小于 20，下面的实例将输出 "Have a good day!"，否则输出 "Have a good night!"：

实例

```php
<?php
$t=date("H");
if ($t<"20")
{
echo "Have a good day!";
}
else
{
echo "Have a good night!";
}
?>
```

运行实例 »

PHP - if...elseif....else 语句

在若干条件之一成立时执行一个代码块，请使用 if....elseif...else 语句。.

语法

```
if (条件)
{
if 条件成立时执行的代码;
}
elseif (条件)
{
elseif 条件成立时执行的代码;
}
else
{
条件不成立时执行的代码;
}
```

如果当前时间小于 10，下面的实例将输出 "Have a good morning!"，如果当前时间不小于 10 且小于 20，则输出 "Have a good day!"，否则输出 "Have a good night!"：

实例

```php
<?php
$t=date("H");
if ($t<"10")
{
echo "Have a good morning!";
}
elseif ($t<"20")
{
echo "Have a good day!";
}
else
{
echo "Have a good night!";
}
?>
```

运行实例 »

PHP - switch 语句

switch 语句将在下一章进行讲解。

1 篇笔记
#1

□写笔记

`elseif` 和 `else if` 完全同效果，**elseif** 是 PHP 为 **else if** 专门做到容错版。更准确更严格到写法为后者: **else if**

```
<?php

$t=date("H");

if ($t<"10")

{

    echo "Have a good morning!";

}

elseif ($t<"20")

{

    echo "Have a good day!";

}

else

{

    echo "Have a good night!";

}

?>
```

等同于:

```
<?php

$t=date("H");

if ($t<"10")

{

    echo "Have a good morning!";

}

else if ($t<"20") //此处有空格

{

    echo "Have a good day!";

}

else

{

    echo "Have a good night!";

}
```

```
?>
```

stinkaroo11个月前 (11-01)

反馈/建议

PHP Switch 语句

switch 语句用于根据多个不同条件执行不同动作。

PHP Switch 语句

如果您希望**有选择地执行若干代码块之一**，请使用 switch 语句。

语法

```
<?php
switch (n)
{
case label1:
如果 n=label1，此处代码将执行;
break;
case label2:
如果 n=label2，此处代码将执行;
break;
default:
如果 n 既不等于 label1 也不等于 label2，此处代码将执行;
}
?>
```

工作原理：首先对一个简单的表达式 *n*（通常是变量）进行一次计算。将表达式的值与结构中每个 case 的值进行比较。如果存在匹配，则执行与 case 关联的代码。代码执行后，使用 **break** 来阻止代码跳入下一个 case 中继续执行。**default** 语句用于不存在匹配（即没有 case 为真）时执行。

实例

```
<?php
$favcolor="red";
switch ($favcolor)
{
case "red":
echo "你喜欢的颜色是红色!";
break;
case "blue":
echo "你喜欢的颜色是蓝色!";
break;
case "green":
echo "你喜欢的颜色是绿色!";
break;
default:
echo "你喜欢的颜色不是 红，蓝，或绿色!";
}
?>
```

运行实例 »

PHP If...Else 语句 PHP 数组

1 篇笔记

写笔记

#1

在 switch 语句中漏写 break, 可能会使你的输出在你意料之外，下面我来和你们仔细讲解 break 的作用：
下
面的代码是正确示范，结果将是只会输出：**这里是a**。
倘若你忘敲了 **case 'a'，case 'b',case 'c'** 后的 **break**，结果将是将代码中的每一条输出语句都输出。
倘若你只敲了 **case 'c'** 后的 **break**, 结果将会是输出包含 **case 'c'** 之前的所有输出语句。
讲到这里大家应该明白了：原来 **switch** 语句不遇到 **break** 将不会自己"拐弯"，希望这些将会帮助到才接触 php 的菜鸟们！

```
<?php

$x='a';

switch ($x){

case 'a':                       //变量$x的值和该种情况匹配，将从此处开始执行。

    echo "这里是a"."<br>";

    break;

case 'b':

    echo "这里是b"."<br>";

    break;

case 'c':

    echo "这里是c"."<br>";

    break;

default:

    echo "这里是default";

}

?>
```

月色真美 11个月前 (10-23)

反馈/建议

PHP Switch 语句 PHP 数组排序

PHP 数组

数组能够在单个变量中存储多个值：

实例

```php
<?php
$cars=array("Volvo","BMW","Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

运行实例 »

数组是什么？

数组是一个能在单个变量中存储多个值的特殊变量。

如果您有一个项目清单（例如：车名字的清单），将其存储到单个变量中，如下所示：

```
$cars1="Volvo";
$cars2="BMW";
$cars3="Toyota";
```

然而，如果您想要遍历数组并找出特定的一个呢？如果数组的项不只 3 个而是 300 个呢？

解决办法是创建一个数组！

数组可以在单个变量中存储多个值，并且您可以根据键访问其中的值。

在 PHP 中创建数组

在 PHP 中，array() 函数用于创建数组：

```
array();
```

在 PHP 中，有三种类型的数组：

- **数值数组** - 带有数字 ID 键的数组
- **关联数组** - 带有指定的键的数组，每个键关联一个值
- **多维数组** - 包含一个或多个数组的数组

PHP 数值数组

这里有两种创建数值数组的方法：

自动分配 ID 键（ID 键总是从 0 开始）：

```
$cars=array("Volvo","BMW","Toyota");
```

人工分配 ID 键：

```
$cars[0]="Volvo";
$cars[1]="BMW";
$cars[2]="Toyota";
```

下面的实例创建一个名为 $cars 的数值数组，并给数组分配三个元素,然后打印一段包含数组值的文本：

实例

```php
<?php
$cars=array("Volvo","BMW","Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

运行实例 »

获取数组的长度 - count() 函数

count() 函数用于返回数组的长度（元素的数量）：

实例

```php
<?php
$cars=array("Volvo","BMW","Toyota");
echo count($cars);
?>
```

运行实例 »

遍历数值数组

遍历并打印数值数组中的所有值，您可以使用 for 循环，如下所示：

实例

```php
<?php
$cars=array("Volvo","BMW","Toyota");
$arrlength=count($cars);
for($x=0;$x<$arrlength;$x++)
{
echo $cars[$x];
echo "<br>";
}
?>
```

运行实例 »

PHP 关联数组

关联数组是使用您分配给数组的指定的键的数组。

这里有两种创建关联数组的方法：

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
```

or:

```
$age['Peter']="35";
$age['Ben']="37";
$age['Joe']="43";
```

随后可以在脚本中使用指定的键：

实例

```php
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

运行实例 »

遍历关联数组

遍历并打印关联数组中的所有值，您可以使用 foreach 循环，如下所示：

实例

```php
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
foreach($age as $x=>$x_value)
{
echo "Key=" . $x . ", Value=" . $x_value;
echo "<br>";
```

```
}
?>
```

运行实例 »

多维数组

多维数组 将在 PHP 高级教程部分做详细介绍。

完整的 PHP Array 参考手册

如需查看所有数组函数的完整参考手册，请访问我们的 PHP Array 参考手册。

该参考手册提供了每个函数的简要描述和应用实例！

PHP Switch 语句 PHP 数组排序

PHP 数组排序

数组中的元素可以按字母或数字顺序进行降序或升序排列。

PHP - 数组排序函数

在本章中，我们将一一介绍下列 PHP 数组排序函数：

- sort() - 对数组进行升序排列
- rsort() - 对数组进行降序排列
- asort() - 根据关联数组的值，对数组进行升序排列
- ksort() - 根据关联数组的键，对数组进行升序排列
- arsort() - 根据关联数组的值，对数组进行降序排列
- krsort() - 根据关联数组的键，对数组进行降序排列

sort() - 对数组进行升序排列

下面的实例将 $cars 数组中的元素按照字母升序排列：

实例

```
<?php
$cars=array("Volvo","BMW","Toyota");
sort($cars);
```

```
?>
```

运行实例 »

下面的实例将 $numbers 数组中的元素按照数字升序排列：

实例

```
<?php
$numbers=array(4,6,2,22,11);
sort($numbers);
?>
```

运行实例 »

rsort() - 对数组进行降序排列

下面的实例将 $cars 数组中的元素按照字母降序排列：

实例

```
<?php
$cars=array("Volvo","BMW","Toyota");
rsort($cars);
?>
```

运行实例 »

下面的实例将 $numbers 数组中的元素按照数字降序排列：

实例

```
<?php
$numbers=array(4,6,2,22,11);
rsort($numbers);
?>
```

运行实例 »

asort() - 根据数组的值，对数组进行升序排列

下面的实例根据数组的值，对关联数组进行升序排列：

实例

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
asort($age);
?>
```

运行实例 »

ksort() - 根据数组的键，对数组进行升序排列

下面的实例根据数组的键，对关联数组进行升序排列：

实例

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
ksort($age);
?>
```

运行实例 »

arsort() - 根据数组的值，对数组进行降序排列

下面的实例根据数组的值，对关联数组进行降序排列：

实例

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
arsort($age);
?>
```

运行实例 »

krsort() - 根据数组的键，对数组进行降序排列

下面的实例根据数组的键，对关联数组进行降序排列：

实例

```
<?php
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
krsort($age);
?>
```

运行实例 »

完整的 PHP Array 参考手册

如需查看所有数组函数的完整参考手册，请访问我们的 PHP Array 参考手册。

该参考手册提供了每个函数的简要描述和应用实例！

□ PHP 数组　　PHP While 循环 □

1 篇笔记

#1

□ 写笔记

使用 PHP 写一个冒泡排序算法:

```
<?php

// 从大到小排序

$numArray =array(3,2,6,5,8,10);

$numCount = count($numArray);

for($i=$numCount-1;$i>=0;$i--){

    for($j=0;$j<$i;$j++){

        if($numArray[$j]< $numArray[$j+1]){

            $aa = $numArray[$j+1];

            $numArray[$j+1]=$numArray[$j];

            $numArray[$j]=$aa;
```

```
        }

    }

}


print_r($numArray);

?>
```

ccccccc7个月前 **(03-17)**

反馈/建议

PHP 超级全局变量

超级全局变量在PHP 4.1.0之后被启用，是PHP系统中自带的变量，在一个脚本的全部作用域中都可用。

PHP 超级全局变量

PHP中预定义了几个超级全局变量（superglobals），这意味着它们在一个脚本的全部作用域中都可用。 你不需要特别说明，就可以在函数及类中使用。

PHP 超级全局变量列表:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

本章节我们将讲解几个常用的超级全局变量,其余变量我们在接下来几个章节会介绍到。

PHP $GLOBALS

$GLOBALS 是PHP的一个超级全局变量组，在一个PHP脚本的全部作用域中都可以访问。

$GLOBALS 是一个包含了全部变量的全局组合数组。变量的名字就是数组的键。

以下实例介绍了如何使用超级全局变量 $GLOBALS:

实例

```php
<?php
$x = 75;
$y = 25;
function addition()
{
$GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>
```

运行实例 »

以上实例中 z 是一个$GLOBALS数组中的超级全局变量，该变量同样可以在函数外访问。

PHP $_SERVER

$_SERVER 是一个包含了诸如头信息(header)、路径(path)、以及脚本位置(script locations)等等信息的数组。这个数组中的项目由 Web 服务器创建。不能保证每个服务器都提供全部项目；服务器可能会忽略一些，或者提供一些没有在这里列举出来的项目。

以下实例中展示了如何使用$_SERVER中的元素:

实例

```php
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

运行实例 »

下表列出了所有 $_SERVER 变量中的重要元素:

元素/代码	描述
$_SERVER['PHP_SELF']	当前执行脚本的文件名，与 document root 有关。例如，在地址为 http://example.com/test.php/foo.bar 的脚本中使用 $_SERVER['PHP_SELF'] 将得到 /test.php/foo.bar。__FILE__ 常量包含当前(例如包含)文件的完整路径和文件名。 从 PHP 4.3.0 版本开始，如果 PHP 以命令行模式运行，这个变量将包含脚本名。之前的版本该变量不可用。
$_SERVER['GATEWAY_INTERFACE']	服务器使用的 CGI 规范的版本；例如，"CGI/1.1"。
$_SERVER['SERVER_ADDR']	当前运行脚本所在的服务器的 IP 地址。
$_SERVER['SERVER_NAME']	当前运行脚本所在的服务器的主机名。如果脚本运行于虚拟主机中，该名称是由那个虚拟主机所设置的值决定。(如: www.runoob.com)
$_SERVER['SERVER_SOFTWARE']	服务器标识字符串，在响应请求时的头信息中给出。 (如：Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	请求页面时通信协议的名称和版本。例如，"HTTP/1.0"。
$_SERVER['REQUEST_METHOD']	访问页面使用的请求方法；例如，"GET", "HEAD"，"POST"，"PUT"。
$_SERVER['REQUEST_TIME']	请求开始时的时间戳。从 PHP 5.1.0 起可用。 (如：1377687496)
$_SERVER['QUERY_STRING']	query string（查询字符串），如果有的话，通过它进行页面访问。

$_SERVER['HTTP_ACCEPT']	当前请求头中 Accept: 项的内容，如果存在的话。
$_SERVER['HTTP_ACCEPT_CHARSET']	当前请求头中 Accept-Charset: 项的内容，如果存在的话。例如："iso-8859-1,*,utf-8"。
$_SERVER['HTTP_HOST']	当前请求头中 Host: 项的内容，如果存在的话。
$_SERVER['HTTP_REFERER']	引导用户代理到当前页的前一页的地址（如果存在）。由 user agent 设置决定。并不是所有的用户代理都会设置该项，有的还提供了修改 HTTP_REFERER 的功能。简言之，该值并不可信。)
$_SERVER['HTTPS']	如果脚本是通过 HTTPS 协议被访问，则被设为一个非空的值。
$_SERVER['REMOTE_ADDR']	浏览当前页面的用户的 IP 地址。
$_SERVER['REMOTE_HOST']	浏览当前页面的用户的主机名。DNS 反向解析不依赖于用户的 REMOTE_ADDR。
$_SERVER['REMOTE_PORT']	用户机器上连接到 Web 服务器所使用的端口号。
$_SERVER['SCRIPT_FILENAME']	当前执行脚本的绝对路径。
$_SERVER['SERVER_ADMIN']	该值指明了 Apache 服务器配置文件中的 SERVER_ADMIN 参数。如果脚本运行在一个虚拟主机上，则该值是那个虚拟主机的值。(如：someone@runoob.com)
$_SERVER['SERVER_PORT']	Web 服务器使用的端口。默认值为 "80"。如果使用 SSL 安全连接，则这个值为用户设置的 HTTP 端口。
$_SERVER['SERVER_SIGNATURE']	包含了服务器版本和虚拟主机名的字符串。
$_SERVER['PATH_TRANSLATED']	当前脚本所在文件系统（非文档根目录）的基本路径。这是在服务器进行虚拟到真实路径的映像后的结果。
$_SERVER['SCRIPT_NAME']	包含当前脚本的路径。这在页面需要指向自己时非常有用。__FILE__ 常量包含当前脚本(例如包含文件)的完整路径和文件名。
$_SERVER['SCRIPT_URI']	URI 用来指定要访问的页面。例如 "/index.html"。

PHP $_REQUEST

PHP $_REQUEST 用于收集HTML表单提交的数据。

以下实例显示了一个输入字段（input）及提交按钮(submit)的表单(form)。 当用户通过点击 "Submit" 按钮提交表单数据时, 表单数据将发送至<form> 标签中 action 属性中指定的脚本文件。 在这个实例中，我们指定文件来处理表单数据。如果你希望其他的PHP文件来处理该数据，你可以修改该指定的脚本文件名。 然后，我们可以使用超级全局变量 $_REQUEST 来收集表单中的 input 字段数据:

实例

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_REQUEST['fname'];
echo $name;
?>

</body>
</html>
```

运行实例 »

PHP $_POST

PHP $_POST 被广泛应用于收集表单数据，在HTML form标签的指定该属性："method="post"。

以下实例显示了一个输入字段（input）及提交按钮(submit)的表单(form)。 当用户通过点击 "Submit" 按钮提交表单数据时, 表单数据将发送至<form>标签中 action 属性中指定的脚本文件。 在这个实例中，我们指定文件来处理表单数据。如果你希望其他的PHP文件来处理该数据，你可以修改该指定的脚本文件名。 然后，我们可以使用超级全局变量 $_POST 来收集表单中的 input 字段数据:

实例

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_POST['fname'];
echo $name;
?>

</body>
</html>
```

运行实例 »

PHP $_GET

PHP $_GET 同样被广泛应用于收集表单数据，在HTML form标签的指定该属性："method="get"。

$_GET 也可以收集URL中发送的数据。

假定我们有一个包含参数的超链接HTML页面：

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=runoob.com">Test $GET</a>

</body>
</html>
```

当用户点击链接 "Test $GET", 参数 "subject" 和 "web" 将发送至"test_get.php",你可以在 "test_get.php" 文件中使用 $_GET 变量来获取这些数据。

以下实例显示了 "test_get.php" 文件的代码:

实例

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

运行实例 »

提示： 你如果想学习更多关于 $_POST 和 $_GET 的知识，请访问我们的 PHP 表单 章节。

PHP 循环 - While 循环

循环执行代码块指定的次数，或者当指定的条件为真时循环执行代码块。

PHP 循环

在您编写代码时，您经常需要让相同的代码块一次又一次地重复运行。我们可以在代码中使用循环语句来完成这个任务。

在 PHP 中，提供了下列循环语句：

- **while** - 只要指定的条件成立，则循环执行代码块
- **do...while** - 首先执行一次代码块，然后在指定的条件成立时重复这个循环
- **for** - 循环执行代码块指定的次数
- **foreach** - 根据数组中每个元素来循环代码块

while 循环

while 循环将重复执行代码块，直到指定的条件不成立。

语法

```
while (条件)
{
    要执行的代码;
}
```

实例

下面的实例首先设置变量 *i* 的值为 1 ($i=1;)。

然后，只要 *i* 小于或者等于 5，while 循环将继续运行。循环每运行一次，*i* 就会递增 1：

```
<html>
<body>

<?php
```

```
$i=1;

while($i<=5)

{

    echo "The number is " . $i . "<br>";

    $i++;

}

?>



</body>

</html>
```

输出：

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

---

do...while 语句

do...while 语句会至少执行一次代码，然后检查条件，只要条件成立，就会重复进行循环。

语法

```
do

{

    要执行的代码;

}

while (条件);
```

实例

下面的实例首先设置变量 *i* 的值为 1 ($i=1;)。

然后，开始 do...while 循环。循环将变量 *i* 的值递增 1，然后输出。先检查条件（*i* 小于或者等于 5），只要 *i* 小于或者等于 5，循环将继续运行：

```
<html>

<body>



<?php

$i=1;

do

{
```

```
    $i++;

    echo "The number is " . $i . "<br>";

}

while ($i<=5);

?>



</body>

</html>
```

输出：

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

for 循环和 foreach 循环将在下一章进行讲解。

□ PHP 数组排序 PHP For 循环 □

PHP 循环 - For 循环

循环执行代码块指定的次数，或者当指定的条件为真时循环执行代码块。

for 循环

for 循环用于您预先知道脚本需要运行的次数的情况。

语法

```
for (初始值; 条件; 增量)

{

    要执行的代码;
```

```
}
```

参数：

初始值：主要是初始化一个变量值，用于设置一个计数器（但可以是任何在循环的开始被执行一次的代码）。

条件：循环执行的限制条件。如果为 TRUE，则循环继续。如果为 FALSE，则循环结束。

增量：主要用于递增计数器（但可以是任何在循环的结束被执行的代码）。

注释：上面的*初始值*和*增量*参数可为空，或者有多个表达式（用逗号分隔）。

实例

下面的实例定义一个初始值为 i=1 的循环。只要变量 *i* 小于或者等于 5，循环将继续运行。循环每运行一次，变量 *i* 就会递增 1：

实例

```
<?php
for ($i=1; $i<=5; $i++)
{
echo "The number is " . $i . "<br>";
}
?>
```

输出：

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

---

foreach 循环

foreach 循环用于遍历数组。

语法

```
foreach ($array as $value)

{

    要执行代码;

}
```

每进行一次循环，当前数组元素的值就会被赋值给 $value 变量（数组指针会逐一地移动），在进行下一次循环时，您将看到数组中的下一个值。

实例

下面的实例演示了一个输出给定数组的值的循环：

实例

```
<?php
$x=array("one","two","three");
foreach ($x as $value)
{
echo $value . "<br>";
}
?>
```

输出：

```
one
two
three
```

1 篇笔记
#1

写笔记

使用for循环，实现冒泡排序：

```
<?php
$arr = array(5,3,6,2,8,10);
for($i = count($arr)-1;$i>=0;$i--){
    for($j = 0 ; $j < $i ; $j++){
        if($arr[$j+1] > $arr[$j] ){
            $aa = $arr[$j+1];
            $arr[$j+1] = $arr[$j];
            $arr[$j] = $aa;
        }
    }
}
print_r($arr);
?>
```

freedom1年前 (2017-09-19)

反馈/建议

PHP 函数

PHP 的真正威力源自于它的函数。

在 PHP 中，提供了超过 1000 个内建的函数。

PHP 内建函数

如需查看所有数组函数的完整参考手册和实例，请访问我们的 PHP 参考手册。

PHP 函数

在本章中，我们将为您讲解如何创建自己的函数。

如要在页面加载时执行脚本，您可以把它放到函数里。

函数是通过调用函数来执行的。

你可以在页面的任何位置调用函数。

创建 PHP 函数

函数是通过调用函数来执行的。

语法

```
<?php
function functionName()
{
// 要执行的代码
}
?>
```

PHP 函数准则：

函数的名称应该提示出它的功能

函数名称以字母或下划线开头（不能以数字开头）

实例

一个简单的函数，在其被调用时能输出我的名称：

实例

```
<?php
function writeName()
{
echo "Kai Jim Refsnes";
}
echo "My name is ";
writeName();
?>
```

输出：

```
My name is Kai Jim Refsnes
```

PHP 函数 - 添加参数

为了给函数添加更多的功能，我们可以添加参数。参数类似变量。

参数就在函数名称后面有一个括号内指定。

实例 1

下面的实例将输出不同的名字，但姓是相同的：

实例

```
<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br>";
}
echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
```

```
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>
```

输出：

```
My name is Kai Jim Refsnes.

My sister's name is Hege Refsnes.

My brother's name is Stale Refsnes.
```

实例 2

下面的函数有两个参数：

实例

```
<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br>";
}
echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>
```

输出：

```
My name is Kai Jim Refsnes.

My sister's name is Hege Refsnes!

My brother's name is Ståle Refsnes?
```

PHP 函数 - 返回值

如需让函数返回一个值，请使用 return 语句。

实例

```
<?php
function add($x,$y)
{
$total=$x+$y;
return $total;
}
echo "1 + 16 = " . add(1,16);
?>
```

输出：

```
1 + 16 = 17
```

1 篇笔记

#1

写笔记

关于函数的定义和使用：

```
<?php

//计算两个数的和

function add($a,$b){

    $count = $a + $b;

    return $count;

}



//计算小明的数学成绩和语文成绩的和（这个也可以使用在从数据库中读取数据并且做加运算）

function count_score(){

    $m = 96;//数学成绩

    $y = 99;//语文成绩

    $sum = add($m,$y);

    echo "小明的总成绩是".$sum;

}

count_score();

?>
```

freedom1年前 (2017-09-19)

反馈/建议

PHP array_replace_recursive() 函数

PHP 命名空间(namespace)

PHP 魔术常量

PHP 向它运行的任何脚本提供了大量的预定义常量。

不过很多常量都是由不同的扩展库定义的，只有在加载了这些扩展库时才会出现，或者动态加载后，或者在编译时已经包括进去了。

有八个魔术常量它们的值随着它们在代码中的位置改变而改变。

例如 __LINE__ 的值就依赖于它在脚本中所处的行来决定。这些特殊的常量不区分大小写，如下：

__LINE__

文件中的当前行号。

实例

```
<?php
echo '这是第 " ' . __LINE__ . ' " 行';
?>
```

以上实例输出结果为：

```
这是第 “ 2 ” 行
```

__FILE__

文件的完整路径和文件名。如果用在被包含文件中，则返回被包含的文件名。

自 PHP 4.0.2 起，__FILE__ 总是包含一个绝对路径（如果是符号连接，则是解析后的绝对路径），而在此之前的版本有时会包含一个相对路径。

实例:

实例

```
<?php
echo '该文件位于 " ' . __FILE__ . ' " ';
?>
```

以上实例输出结果为：

```
该文件位于 “ E:\wamp\www\test\index.php ”
```

__DIR__

文件所在的目录。如果用在被包括文件中，则返回被包括的文件所在的目录。

它等价于 dirname(__FILE__)。除非是根目录，否则目录中名不包括末尾的斜杠。（PHP 5.3.0中新增）

实例

```
<?php
echo '该文件位于 " ' . __DIR__ . ' " ';
?>
```

以上实例输出结果为：

```
该文件位于 “ E:\wamp\www\test ”
```

__FUNCTION__

函数名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该函数被定义时的名字（区分大小写）。在 PHP 4 中该值总是小写字母的。

实例

```
<?php
function test() {
echo '函数名为: ' . __FUNCTION__ ;
```

```
}
test();
?>
```

以上实例输出结果为：

```
函数名为：test
```

__CLASS__

类的名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该类被定义时的名字（区分大小写）。

在 PHP 4 中该值总是小写字母的。类名包括其被声明的作用区域（例如 Foo\Bar）。注意自 PHP 5.4 起 __CLASS__ 对 trait 也起作用。当用在 trait 方法中时，__CLASS__ 是调用 trait 方法的类的名字。

实例

```
<?php
class test {
function _print() {
echo '类名为：' . __CLASS__ . "<br>";
echo '函数名为：' . __FUNCTION__ ;
}
}
$t = new test();
$t->_print();
?>
```

以上实例输出结果为：

```
类名为：test

函数名为：_print
```

__TRAIT__

Trait 的名字（PHP 5.4.0 新加）。自 PHP 5.4.0 起，PHP 实现了代码复用的一个方法，称为 traits。

Trait 名包括其被声明的作用区域（例如 Foo\Bar）。

从基类继承的成员被插入的 SayWorld Trait 中的 MyHelloWorld 方法所覆盖。其行为 MyHelloWorld 类中定义的方法一致。优先顺序是当前类中的方法会覆盖 trait 方法，而 trait 方法又覆盖了基类中的方法。

实例

```
<?php
class Base {
public function sayHello() {
echo 'Hello ';
}
}
trait SayWorld {
public function sayHello() {
parent::sayHello();
echo 'World!';
}
}
class MyHelloWorld extends Base {
use SayWorld;
}
$o = new MyHelloWorld();
$o->sayHello();
?>
```

以上例程会输出：

```
Hello World!
```

__METHOD__

类的方法名（PHP 5.0.0 新加）。返回该方法被定义时的名字（区分大小写）。

实例:

实例

```php
<?php
function test() {
echo '函数名为: ' . __METHOD__ ;
}
test();
?>
```

以上实例输出结果为：

```
函数名为: test
```

__NAMESPACE__

当前命名空间的名称（区分大小写）。此常量是在编译时定义的（PHP 5.3.0 新增）。

实例:

实例

```php
<?php
namespace MyProject;
echo '命名空间为: "', __NAMESPACE__, '"'; // 输出 "MyProject"
?>
```

以上实例输出结果为：

```
命名空间为: "MyProject"
```

□ PHP array_replace_recursive() 函数　　PHP 命名空间(namespace) □

□

1 篇笔记

#1

□写笔记

□

多个 trait 的情况：

通过逗号分隔，在 use 声明列出多个 trait，可以都插入到一个类中。示例代码如下：

```
<?php

trait Hello {

    public function sayHello() {

        echo 'Hello ';
```

```
    }
}
trait World {
    public function sayWorld() {
        echo 'World';
    }
}
class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}
$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
?>
```

最终输出：`Hello World!`

马鹿7个月前 (03-13)

反馈/建议

PHP 命名空间(namespace)

PHP 命名空间(namespace)是在PHP 5.3中加入的，如果你学过C#和Java，那命名空间就不算什么新事物。 不过在PHP当中还是有着相当重要的意义。

PHP 命名空间可以解决以下两类问题：

1. 用户编写的代码与PHP内部的类/函数/常量或第三方类/函数/常量之间的名字冲突。
2. 为很长的标识符名称(通常是为了缓解第一类问题而定义的)创建一个别名（或简短）的名称，提高源代码的可读性。

定义命名空间

默认情况下，所有常量、类和函数名都放在全局空间下，就和PHP支持命名空间之前一样。

命名空间通过关键字namespace 来声明。如果一个文件中包含命名空间，它必须在其它所有代码之前声明命名空间。语法格式如下：

```
<?php

// 定义代码在 'MyProject' 命名空间中

namespace MyProject;



// ... 代码 ...
```

你也可以在同一个文件中定义不同的命名空间代码，如：

```
<?php

namespace MyProject;



const CONNECT_OK = 1;

class Connection { /* ... */ }

function connect() { /* ... */  }



namespace AnotherProject;



const CONNECT_OK = 1;

class Connection { /* ... */ }

function connect() { /* ... */  }

?>
```

不建议使用这种语法在单个文件中定义多个命名空间。建议使用下面的大括号形式的语法。

```
<?php

namespace MyProject {

    const CONNECT_OK = 1;

    class Connection { /* ... */ }

    function connect() { /* ... */  }

}
```

```
namespace AnotherProject {

    const CONNECT_OK = 1;

    class Connection { /* ... */ }

    function connect() { /* ... */  }

}

?>
```

将全局的非命名空间中的代码与命名空间中的代码组合在一起，只能使用大括号形式的语法。全局代码必须用一个不带名称的 namespace 语句加上大括号括起来，例如：

```
<?php

namespace MyProject {



const CONNECT_OK = 1;

class Connection { /* ... */ }

function connect() { /* ... */  }

}



namespace { // 全局代码

session_start();

$a = MyProject\connect();

echo MyProject\Connection::start();

}

?>
```

在声明命名空间之前唯一合法的代码是用于定义源文件编码方式的 declare 语句。所有非 PHP 代码包括空白符都不能出现在命名空间的声明之前。

```
<?php

declare(encoding='UTF-8'); //定义多个命名空间和不包含在命名空间中的代码

namespace MyProject {



const CONNECT_OK = 1;

class Connection { /* ... */ }

function connect() { /* ... */  }
```

```
}


namespace { // 全局代码

session_start();

$a = MyProject\connect();

echo MyProject\Connection::start();

}

?>
```

以下代码会出现语法错误：

```
<html>

<?php

namespace MyProject; // 命名空间前出现了“<html>”会致命错误 -  命名空间必须是程序脚本的第一条语句

?>
```

子命名空间

与目录和文件的关系很像，PHP 命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义：

```
<?php

namespace MyProject\Sub\Level;  //声明分层次的单个命名空间



const CONNECT_OK = 1;

class Connection { /* ... */ }

function Connect() { /* ... */  }



?>
```

上面的例子创建了常量 MyProject\Sub\Level\CONNECT_OK，类 MyProject\Sub\Level\Connection 和函数 MyProject\Sub\Level\Connect。

命名空间使用

PHP 命名空间中的类名可以通过三种方式引用：

1. **非限定名称，或不包含前缀的类名称，**例如 $a=new foo(); 或 foo::staticmethod();。如果当前命名空间是 currentnamespace，foo 将被解析为 currentnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，则 foo 会被解析为foo。 警告：如果命名空间中的函数或常量未定义，则该非限定的函数名称或常量名称会被解析为全局函数名称或常量名称。
2. **限定名称,或包含前缀的名称，**例如 $a = new subnamespace\foo(); 或 subnamespace\foo::staticmethod();。如果当前的命名空间是 currentna

mespace，则 foo 会被解析为 currentnamespace\subnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，foo 会被解析为subnamespace\foo。

3. **完全限定名称，或包含了全局前缀操作符的名称，**例如， $a = new \currentnamespace\foo(); 或 \currentnamespace\foo::staticmethod();。在这种情况下，foo 总是被解析为代码中的文字名(literal name)currentnamespace\foo。

下面是一个使用这三种方式的实例：

file1.php 文件代码

```
<?php

namespace Foo\Bar\subnamespace;



const FOO = 1;

function foo() {}

class foo

{

    static function staticmethod() {}

}

?>
```

file2.php 文件代码

```
<?php

namespace Foo\Bar;

include 'file1.php';



const FOO = 2;

function foo() {}

class foo

{

    static function staticmethod() {}

}



/* 非限定名称 */

foo(); // 解析为函数 Foo\Bar\foo

foo::staticmethod(); // 解析为类 Foo\Bar\foo ，方法为 staticmethod

echo FOO; // 解析为常量 Foo\Bar\FOO
```

```
/* 限定名称 */
subnamespace\foo(); // 解析为函数 Foo\Bar\subnamespace\foo
subnamespace\foo::staticmethod(); // 解析为类 Foo\Bar\subnamespace\foo,
                                  // 以及类的方法 staticmethod
echo subnamespace\FOO; // 解析为常量 Foo\Bar\subnamespace\FOO

/* 完全限定名称 */
\Foo\Bar\foo(); // 解析为函数 Foo\Bar\foo
\Foo\Bar\foo::staticmethod(); // 解析为类 Foo\Bar\foo, 以及类的方法 staticmethod
echo \Foo\Bar\FOO; // 解析为常量 Foo\Bar\FOO
?>
```

注意访问任意全局类、函数或常量，都可以使用完全限定名称，例如 \strlen() 或 \Exception 或 \INI_ALL。
在命名空间内部访问全局类、函数和常量：

```
<?php
namespace Foo;

function strlen() {}
const INI_ALL = 3;
class Exception {}

$a = \strlen('hi'); // 调用全局函数strlen
$b = \INI_ALL; // 访问全局常量 INI_ALL
$c = new \Exception('error'); // 实例化全局类 Exception
?>
```

命名空间和动态语言特征

PHP 命名空间的实现受到其语言自身的动态特征的影响。因此，如果要将下面的代码转换到命名空间中，动态访问元素。
example1.php 文件代码：

```
<?php
class classname
```

```
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "global";

$a = 'classname';
$obj = new $a; // prints classname::__construct
$b = 'funcname';
$b(); // prints funcname
echo constant('constname'), "\n"; // prints global
?>
```

必须使用完全限定名称（包括命名空间前缀的类名称）。注意因为在动态的类名称、函数名称或常量名称中，限定名称和完全限定名称没有区别，因此其前导的反斜杠是不必要的。
动态访问命名空间的元素

```
<?php
namespace namespacename;
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
```

```
    echo __FUNCTION__,"\n";
}
const constname = "namespaced";

include 'example1.php';

$a = 'classname';
$obj = new $a; // 输出 classname::__construct
$b = 'funcname';
$b(); // 输出函数名
echo constant('constname'), "\n"; // 输出 global

/* 如果使用双引号，使用方法为 "\\namespacename\\classname"*/
$a = '\namespacename\classname';
$obj = new $a; // 输出 namespacename\classname::__construct
$a = 'namespacename\classname';
$obj = new $a; // 输出 namespacename\classname::__construct
$b = 'namespacename\funcname';
$b(); // 输出 namespacename\funcname
$b = '\namespacename\funcname';
$b(); // 输出 namespacename\funcname
echo constant('\namespacename\constname'), "\n"; // 输出 namespaced
echo constant('namespacename\constname'), "\n"; // 输出 namespaced
?>
```

namespace关键字和__NAMESPACE__常量

PHP支持两种抽象的访问当前命名空间内部元素的方法，__NAMESPACE__ 魔术常量和namespace关键字。

常量__NAMESPACE__的值是包含当前命名空间名称的字符串。在全局的，不包括在任何命名空间中的代码，它包含一个空的字符串。

__NAMESPACE__ 示例, 在命名空间中的代码

```
<?php
namespace MyProject;
```

```
echo '"', __NAMESPACE__, '"'; // 输出 "MyProject"

?>
```

__NAMESPACE__ 示例，全局代码

```
<?php



echo '"', __NAMESPACE__, '"'; // 输出 ""

?>
```

常量 __NAMESPACE__ 在动态创建名称时很有用，例如：
使用__NAMESPACE__动态创建名称

```
<?php

namespace MyProject;



function get($classname)

{

    $a = __NAMESPACE__ . '\\' . $classname;

    return new $a;

}

?>
```

关键字 namespace 可用来显式访问当前命名空间或子命名空间中的元素。它等价于类中的 self 操作符。
namespace操作符，命名空间中的代码

```
<?php

namespace MyProject;



use blah\blah as mine; // see "Using namespaces: importing/aliasing"



blah\mine(); // calls function blah\blah\mine()

namespace\blah\mine(); // calls function MyProject\blah\mine()



namespace\func(); // calls function MyProject\func()
```

```
namespace\sub\func(); // calls function MyProject\sub\func()

namespace\cname::method(); // calls static method "method" of class MyProject\cname

$a = new namespace\sub\cname(); // instantiates object of class MyProject\sub\cname

$b = namespace\CONSTANT; // assigns value of constant MyProject\CONSTANT to $b

?>
```

namespace操作符, 全局代码

```
<?php



namespace\func(); // calls function func()

namespace\sub\func(); // calls function sub\func()

namespace\cname::method(); // calls static method "method" of class cname

$a = new namespace\sub\cname(); // instantiates object of class sub\cname

$b = namespace\CONSTANT; // assigns value of constant CONSTANT to $b

?>
```

使用命名空间：别名/导入

PHP 命名空间支持 有两种使用别名或导入方式：为类名称使用别名，或为命名空间名称使用别名。

在PHP中，别名是通过操作符 use 来实现的. 下面是一个使用所有可能的三种导入方式的例子：

1、使用use操作符导入/使用别名

```
<?php

namespace foo;

use My\Full\Classname as Another;



// 下面的例子与 use My\Full\NSname as NSname 相同

use My\Full\NSname;



// 导入一个全局类

use \ArrayObject;



$obj = new namespace\Another; // 实例化 foo\Another 对象

$obj = new Another; // 实例化 My\Full\Classname  对象

NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func
```

```
$a = new ArrayObject(array(1)); // 实例化 ArrayObject 对象

// 如果不使用 "use \ArrayObject" ，则实例化一个 foo\ArrayObject 对象

?>
```

2、 一行中包含多个use语句

```
<?php

use My\Full\Classname as Another, My\Full\NSname;



$obj = new Another; // 实例化 My\Full\Classname 对象

NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func

?>
```

导入操作是在编译执行的，但动态的类名称、函数名称或常量名称则不是。

3、导入和动态名称

```
<?php

use My\Full\Classname as Another, My\Full\NSname;



$obj = new Another; // 实例化一个 My\Full\Classname 对象

$a = 'Another';

$obj = new $a;      // 实际化一个 Another 对象

?>
```

另外，导入操作只影响非限定名称和限定名称。完全限定名称由于是确定的，故不受导入的影响。

4、导入和完全限定名称

```
<?php

use My\Full\Classname as Another, My\Full\NSname;



$obj = new Another; // 实例化 My\Full\Classname 类

$obj = new \Another; // 实例化 Another 类

$obj = new Another\thing; // 实例化 My\Full\Classname\thing 类

$obj = new \Another\thing; // 实例化 Another\thing 类

?>
```

使用命名空间：后备全局函数/常量

在一个命名空间中，当 PHP 遇到一个非限定的类、函数或常量名称时，它使用不同的优先策略来解析该名称。类名称总是解析到当前命名空间中的名称。因此在访问系统内部或不包含在命名空间中的类名称时，必须使用完全限定名称，例如：

1、在命名空间中访问全局类

```
<?php

namespace A\B\C;

class Exception extends \Exception {}



$a = new Exception('hi'); // $a 是类 A\B\C\Exception 的一个对象

$b = new \Exception('hi'); // $b 是类 Exception 的一个对象



$c = new ArrayObject; // 致命错误, 找不到 A\B\C\ArrayObject 类

?>
```

对于函数和常量来说，如果当前命名空间中不存在该函数或常量，PHP 会退而使用全局空间中的函数或常量。

2、 命名空间中后备的全局函数/常量

```
<?php

namespace A\B\C;



const E_ERROR = 45;

function strlen($str)

{

    return \strlen($str) - 1;

}



echo E_ERROR, "\n"; // 输出 "45"

echo INI_ALL, "\n"; // 输出 "7" - 使用全局常量 INI_ALL



echo strlen('hi'), "\n"; // 输出 "2"

if (is_array('hi')) { // 输出 "is not array"

    echo "is array\n";
```

```
} else {

    echo "is not array\n";

}

?>
```

全局空间

如果没有定义任何命名空间，所有的类与函数的定义都是在全局空间，与 PHP 引入命名空间概念前一样。在名称前加上前缀 \ 表示该名称是全局空间中的名称，即使该名称位于其它的命名空间中时也是如此。
使用全局空间说明

```
<?php

namespace A\B\C;



/* 这个函数是 A\B\C\fopen */

function fopen() {

     /* ... */

     $f = \fopen(...); // 调用全局的fopen函数

     return $f;

}

?>
```

命名空间的顺序

自从有了命名空间之后，最容易出错的该是使用类的时候，这个类的寻找路径是什么样的了。

```
<?php

namespace A;

use B\D, C\E as F;



// 函数调用



foo();      // 首先尝试调用定义在命名空间"A"中的函数foo()

            // 再尝试调用全局函数 "foo"



\foo();     // 调用全局空间函数 "foo"
```

```
my\foo();   // 调用定义在命名空间"A\my"中函数 "foo"

F();        // 首先尝试调用定义在命名空间"A"中的函数 "F"
            // 再尝试调用全局函数 "F"

// 类引用

new B();    // 创建命名空间 "A" 中定义的类 "B" 的一个对象
            // 如果未找到，则尝试自动装载类 "A\B"

new D();    // 使用导入规则，创建命名空间 "B" 中定义的类 "D" 的一个对象
            // 如果未找到，则尝试自动装载类 "B\D"

new F();    // 使用导入规则，创建命名空间 "C" 中定义的类 "E" 的一个对象
            // 如果未找到，则尝试自动装载类 "C\E"

new \B();   // 创建定义在全局空间中的类 "B" 的一个对象
            // 如果未发现，则尝试自动装载类 "B"

new \D();   // 创建定义在全局空间中的类 "D" 的一个对象
            // 如果未发现，则尝试自动装载类 "D"

new \F();   // 创建定义在全局空间中的类 "F" 的一个对象
            // 如果未发现，则尝试自动装载类 "F"

// 调用另一个命名空间中的静态方法或命名空间函数

B\foo();    // 调用命名空间 "A\B" 中函数 "foo"

B::foo();   // 调用命名空间 "A" 中定义的类 "B" 的 "foo" 方法
            // 如果未找到类 "A\B" ，则尝试自动装载类 "A\B"
```

```
D::foo();   // 使用导入规则，调用命名空间 "B" 中定义的类 "D" 的 "foo" 方法
            // 如果类 "B\D" 未找到，则尝试自动装载类 "B\D"

\B\foo();   // 调用命名空间 "B" 中的函数 "foo"

\B::foo();  // 调用全局空间中的类 "B" 的 "foo" 方法
            // 如果类 "B" 未找到，则尝试自动装载类 "B"

// 当前命名空间中的静态方法或函数

A\B::foo();   // 调用命名空间 "A\A" 中定义的类 "B" 的 "foo" 方法
              // 如果类 "A\A\B" 未找到，则尝试自动装载类 "A\A\B"

\A\B::foo();  // 调用命名空间 "A" 中定义的类 "B" 的 "foo" 方法
              // 如果类 "A\B" 未找到，则尝试自动装载类 "A\B"
?>
```

名称解析遵循下列规则：

1. 对完全限定名称的函数，类和常量的调用在编译时解析。例如 *new \A\B* 解析为类 *A\B*。
2. 所有的非限定名称和限定名称（非完全限定名称）根据当前的导入规则在编译时进行转换。例如，如果命名空间 *A\B\C* 被导入为 *C*，那么对 *C\D\e()* 的调用就会被转换为 *A\B\C\D\e()*。
3. 在命名空间内部，所有的没有根据导入规则转换的限定名称均会在其前面加上当前的命名空间名称。例如，在命名空间 *A\B* 内部调用 *C\D\e()*，则 *C\D\e()* 会被转换为 *A\B\C\D\e()* 。
4. 非限定类名根据当前的导入规则在编译时转换（用全名代替短的导入名称）。例如，如果命名空间 *A\B\C* 导入为*C*，则 *new C()* 被转换为 *new A\B\C()* 。
5. 在命名空间内部（例如A\B），对非限定名称的函数调用是在运行时解析的。例如对函数 *foo()* 的调用是这样解析的：
    1. 在当前命名空间中查找名为 *A\B\foo()* 的函数
    2. 尝试查找并调用 *全局(global)* 空间中的函数 *foo()*。
6. 在命名空间（例如*A\B*）内部对非限定名称或限定名称类（非完全限定名称）的调用是在运行时解析的。下面是调用 *new C()* 及 *new D\E()* 的解析过程： *new C()*的解析:
    1. 在当前命名空间中查找*A\B\C*类。
    2. 尝试自动装载类*A\B\C*。

    *new D\E()*的解析:
    1. 在类名称前面加上当前命名空间名称变成：*A\B\D\E*，然后查找该类。
    2. 尝试自动装载类 *A\B\D\E*。

    为了引用全局命名空间中的全局类，必须使用完全限定名称 *new \C()*。

1 篇笔记

#1

写笔记

可以把非限定名称类比为文件名（例如 comment.php）、.限定名称类比为相对路径名（例如 ./article/comment.php）、完全限定名称类比为绝对路径名（例如 /blog/article/comment.php），这样可能会更容易理解。

再添一例：

```
<?php

//创建空间Blog

namespace Blog;

class Comment { }

//非限定名称，表示当前Blog空间

//这个调用将被解析成 Blog\Comment();

$blog_comment = new Comment();

//限定名称，表示相对于Blog空间

//这个调用将被解析成 Blog\Article\Comment();

$article_comment = new Article\Comment(); //类前面没有反斜杆\

//完全限定名称，表示绝对于Blog空间

//这个调用将被解析成 Blog\Comment();

$article_comment = new \Blog\Comment(); //类前面有反斜杆\

//完全限定名称，表示绝对于Blog空间

//这个调用将被解析成 Blog\Article\Comment();

$article_comment = new \Blog\Article\Comment(); //类前面有反斜杆\



//创建Blog的子空间Article

namespace Blog\Article;

class Comment { }

?>
```

更多内容可参考：PHP命名空间(Namespace)的使用详解

Alex Gump 2个月前 (08-06)

反馈/建议

PHP 面向对象

在面向对象的程序设计（英语：Object-oriented programming，缩写：OOP）中，对象是一个由信息及对信息进行处理的描述所组成的整体，是对现实世界的抽象。

在现实世界里我们所面对的事情都是对象，如计算机、电视机、自行车等。

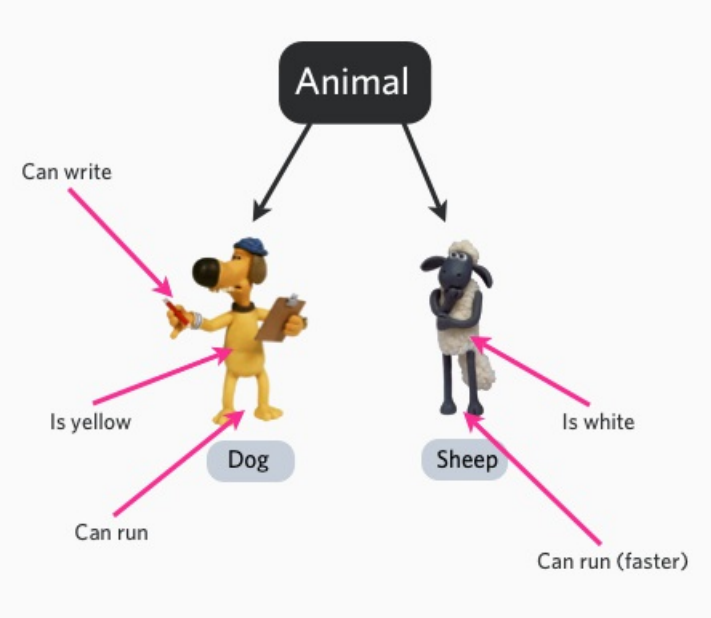
对象的主要三个特性：

对象的行为：可以对 对象施加那些操作，开灯，关灯就是行为。

对象的形态：当施加那些方法是对象如何响应，颜色，尺寸，外型。

对象的表示：对象的表示就相当于身份证，具体区分在相同的行为与状态下有什么不同。

比如 Animal(动物) 是一个抽象类，我们可以具体到一只狗跟一只羊，而狗跟羊就是具体的对象，他们有颜色属性，可以写，可以跑等行为状态。



面向对象内容

类 - 定义了一件事物的抽象特点。类的定义包含了数据的形式以及对数据的操作。

对象 - 是类的实例。

成员变量 - 定义在类内部的变量。该变量的值对外是不可见的，但是可以通过成员函数访问，在类被实例化为对象后，该变量即可称为对象的属性。

成员函数 - 定义在类的内部，可用于访问对象的数据。

继承 - 继承性是子类自动共享父类数据结构和方法的机制，这是类之间的一种关系。在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并加入若干新的内容。

父类 - 一个类被其他类继承，可将该类称为父类，或基类，或超类。

子类 - 一个类继承其他类称为子类，也可称为派生类。

多态 - 多态性是指相同的函数或方法可作用于多种类型的对象上并获得不同的结果。不同的对象，收到同一消息可以产生不同的结果，这种现象称为多态性。

重载 – 简单说，就是函数或者方法有同样的名称，但是参数列表不相同的情形，这样的同名不同参数的函数或者方法之间，互相称之为重载函数或者方法。

抽象性 – 抽象性是指将具有一致的数据结构（属性）和行为（操作）的对象抽象成类。一个类就是这样一种抽象，它反映了与应用有关的重要性质，而忽略其他一些无关内容。任何类的划分都是主观的，但必须与具体的应用有关。

封装 – 封装是指将现实世界中存在的某个客体的属性与行为绑定在一起，并放置在一个逻辑单元内。

构造函数 – 主要用来在创建对象时初始化对象， 即为对象成员变量赋初始值，总与new运算符一起使用在创建对象的语句中。

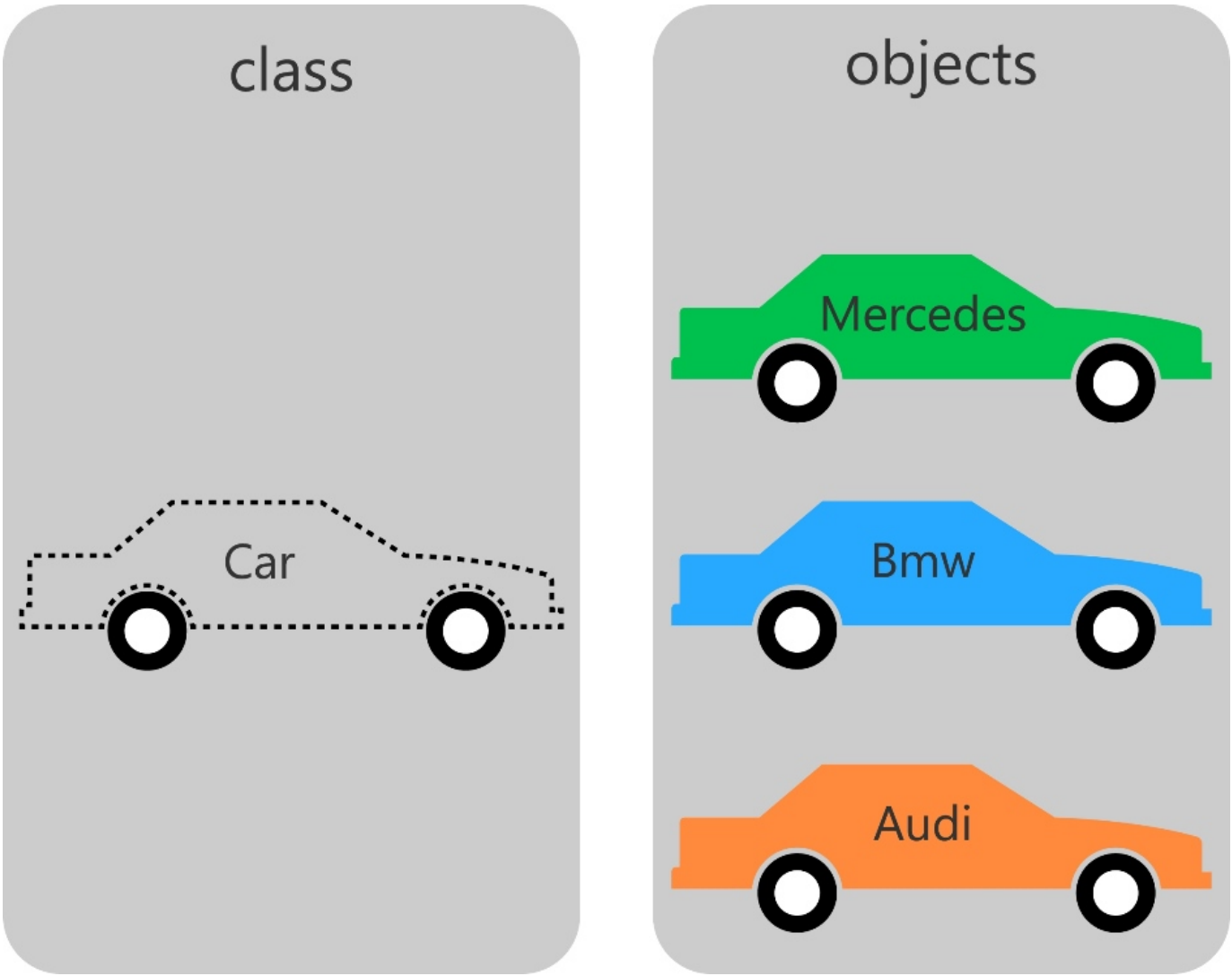
析构函数 – 析构函数(destructor) 与构造函数相反，当对象结束其生命周期时（例如对象所在的函数已调用完毕），系统自动执行析构函数。析构函数往往用来做"清理善后" 的工作（例如在建立对象时用new开辟了一片内存空间，应在退出前在析构函数中用delete释放）。

下图中我们通过 Car 类 创建了三个对象：Mercedes, Bmw, 和 Audi。

```
$mercedes = new Car ();

$bmw = new Car ();

$audi = new Car ();
```



PHP 类定义

PHP 定义类通常语法格式如下：

```
<?php

class phpClass {

  var $var1;

  var $var2 = "constant string";



  function myfunc ($arg1, $arg2) {

     [..]

  }

  [..]

}

?>
```

解析如下：

类使用 **class** 关键字后加上类名定义。

类名后的一对大括号({})内可以定义变量和方法。

类的变量使用 **var** 来声明, 变量也可以初始化值。

函数定义类似 PHP 函数的定义，但函数只能通过该类及其实例化的对象访问。

实例

```
<?php

class Site {

  /* 成员变量 */

  var $url;

  var $title;



  /* 成员函数 */

  function setUrl($par){

     $this->url = $par;

  }



  function getUrl(){

     echo $this->url . PHP_EOL;

  }
```

```
  function setTitle($par){
     $this->title = $par;
  }

  function getTitle(){
     echo $this->title . PHP_EOL;
  }
}
?>
```

变量 **$this** 代表自身的对象。
PHP_EOL 为换行符。

PHP 中创建对象

类创建后，我们可以使用 **new** 运算符来实例化该类的对象：

```
$runoob = new Site;
$taobao = new Site;
$google = new Site;
```

以上代码我们创建了三个对象，三个对象各自都是独立的，接下来我们来看看如何访问成员方法与成员变量。

调用成员方法

在实例化对象后，我们可以使用该对象调用成员方法，该对象的成员方法只能操作该对象的成员变量：

```
// 调用成员函数，设置标题和URL
$runoob->setTitle( "菜鸟教程" );
$taobao->setTitle( "淘宝" );
$google->setTitle( "Google 搜索" );

$runoob->setUrl( 'www.runoob.com' );
$taobao->setUrl( 'www.taobao.com' );
$google->setUrl( 'www.google.com' );

// 调用成员函数，获取标题和URL
$runoob->getTitle();
```

```
$taobao->getTitle();

$google->getTitle();



$runoob->getUrl();

$taobao->getUrl();

$google->getUrl();
```

完整代码如下：

实例

```php
<?php
class Site {
  /* 成员变量 */
  var $url;
  var $title;

  /* 成员函数 */
  function setUrl($par){
     $this->url = $par;
  }

  function getUrl(){
     echo $this->url . PHP_EOL;
  }

  function setTitle($par){
     $this->title = $par;
  }

  function getTitle(){
     echo $this->title . PHP_EOL;
  }
}

$runoob = new Site;
$taobao = new Site;
$google = new Site;

// 调用成员函数，设置标题和URL
$runoob->setTitle( "菜鸟教程" );
$taobao->setTitle( "淘宝" );
$google->setTitle( "Google 搜索" );

$runoob->setUrl( 'www.runoob.com' );
$taobao->setUrl( 'www.taobao.com' );
$google->setUrl( 'www.google.com' );

// 调用成员函数，获取标题和URL
$runoob->getTitle();
$taobao->getTitle();
$google->getTitle();

$runoob->getUrl();
$taobao->getUrl();
$google->getUrl();
?>
```

运行实例 »

执行以上代码，输出结果为：

```
```

```
菜鸟教程

淘宝

Google 搜索

www.runoob.com

www.taobao.com

www.google.com
```

PHP 构造函数

构造函数是一种特殊的方法。主要用来在创建对象时初始化对象， 即为对象成员变量赋初始值，在创建对象的语句中与 `new` 运算符一起使用。

PHP 5 允许开发者在一个类中定义一个方法作为构造函数，语法格式如下：

```
void __construct ([ mixed $args [, $... ]] )
```

在上面的例子中我们就可以通过构造方法来初始化 $url 和 $title 变量：

```
function __construct( $par1, $par2 ) {

   $this->url = $par1;

   $this->title = $par2;

}
```

现在我们就不需要再调用 setTitle 和 setUrl 方法了：

实例

```
$runoob = new Site('www.runoob.com', '菜鸟教程');
$taobao = new Site('www.taobao.com', '淘宝');
$google = new Site('www.google.com', 'Google 搜索');

// 调用成员函数，获取标题和URL
$runoob->getTitle();
$taobao->getTitle();
$google->getTitle();

$runoob->getUrl();
$taobao->getUrl();
$google->getUrl();
```

运行实例 »

析构函数

析构函数(destructor) 与构造函数相反，当对象结束其生命周期时（例如对象所在的函数已调用完毕），系统自动执行析构函数。

PHP 5 引入了析构函数的概念，这类似于其它面向对象的语言，其语法格式如下：

```
void __destruct ( void )
```

实例

```
<?php

class MyDestructableClass {

   function __construct() {

       print "构造函数\n";

       $this->name = "MyDestructableClass";

   }



   function __destruct() {

       print "销毁 " . $this->name . "\n";

   }

}



$obj = new MyDestructableClass();

?>
```

执行以上代码，输出结果为：

```
构造函数

销毁 MyDestructableClass
```

继承

PHP 使用关键字 **extends** 来继承一个类，PHP 不支持多继承，格式如下：

```
class Child extends Parent {

   // 代码部分

}
```

实例

实例中 Child_Site 类继承了 Site 类，并扩展了功能：

```
<?php

// 子类扩展站点类别

class Child_Site extends Site {
```

```
   var $category;



    function setCate($par){

        $this->category = $par;

    }



    function getCate(){

        echo $this->category . PHP_EOL;

    }

}
```

方法重写

如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。

实例中重写了 getUrl 与 getTitle 方法：

```
function getUrl() {

   echo $this->url . PHP_EOL;

   return $this->url;

}



function getTitle(){

   echo $this->title . PHP_EOL;

   return $this->title;

}
```

访问控制

PHP 对属性或方法的访问控制，是通过在前面添加关键字 public（公有），protected（受保护）或 private（私有）来实现的。

public（公有）： 公有的类成员可以在任何地方被访问。

protected（受保护）： 受保护的类成员则可以被其自身以及其子类和父类访问。

private（私有）： 私有的类成员则只能被其定义所在的类访问。

属性的访问控制

类属性必须定义为公有，受保护，私有之一。如果用 var 定义，则被视为公有。

```
<?php
```

```
/**
 * Define MyClass
 */
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // 这行能被正常执行
echo $obj->protected; // 这行会产生一个致命错误
echo $obj->private; // 这行也会产生一个致命错误
$obj->printHello(); // 输出 Public、Protected 和 Private



/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // 可以对 public 和 protected 进行重定义，但 private 而不能
    protected $protected = 'Protected2';

    function printHello()
```

```
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // 这行能被正常执行
echo $obj2->private; // 未定义 private
echo $obj2->protected; // 这行会产生一个致命错误
$obj2->printHello(); // 输出 Public、Protected2 和 Undefined

?>
```

方法的访问控制

类中的方法可以被定义为公有，私有或受保护。如果没有设置这些关键字，则该方法默认为公有。

```
<?php
/**
 * Define MyClass
 */
class MyClass
{
    // 声明一个公有的构造函数
    public function __construct() { }

    // 声明一个公有的方法
    public function MyPublic() { }

    // 声明一个受保护的方法
    protected function MyProtected() { }
```

```
    // 声明一个私有的方法
    private function MyPrivate() { }

    // 此方法为公有
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // 这行能被正常执行
$myclass->MyProtected(); // 这行会产生一个致命错误
$myclass->MyPrivate(); // 这行会产生一个致命错误
$myclass->Foo(); // 公有，受保护，私有都可以执行


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // 此方法为公有
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // 这行会产生一个致命错误
    }
}
```

```
$myclass2 = new MyClass2;

$myclass2->MyPublic(); // 这行能被正常执行

$myclass2->Foo2(); // 公有的和受保护的都可执行，但私有的不行



class Bar

{

    public function test() {

        $this->testPrivate();

        $this->testPublic();

    }



    public function testPublic() {

        echo "Bar::testPublic\n";

    }



    private function testPrivate() {

        echo "Bar::testPrivate\n";

    }

}



class Foo extends Bar

{

    public function testPublic() {

        echo "Foo::testPublic\n";

    }



    private function testPrivate() {

        echo "Foo::testPrivate\n";

    }

}



$myFoo = new foo();
```

```
$myFoo->test(); // Bar::testPrivate

                // Foo::testPublic

?>
```

接口

使用接口（interface），可以指定某个类必须实现哪些方法，但不需要定义这些方法的具体内容。

接口是通过 **interface** 关键字来定义的，就像定义一个标准的类一样，但其中定义所有的方法都是空的。

接口中定义的所有方法都必须是公有，这是接口的特性。

要实现一个接口，使用 **implements** 操作符。类中必须实现接口中定义的所有方法，否则会报一个致命错误。类可以实现多个接口，用逗号来分隔多个接口的名称。

```
<?php



// 声明一个'iTemplate'接口

interface iTemplate

{

    public function setVariable($name, $var);

    public function getHtml($template);

}




// 实现接口

class Template implements iTemplate

{

    private $vars = array();



    public function setVariable($name, $var)

    {

        $this->vars[$name] = $var;

    }



    public function getHtml($template)

    {

        foreach($this->vars as $name => $value) {

            $template = str_replace('{' . $name . '}', $value, $template);
```

```
        }


        return $template;
    }
}
```

常量

可以把在类中始终保持不变的值定义为常量。在定义和使用常量的时候不需要使用 $ 符号。

常量的值必须是一个定值，不能是变量，类属性，数学运算的结果或函数调用。

自 PHP 5.3.0 起，可以用一个变量来动态调用类。但该变量的值不能为关键字（如 self，parent 或 static）。

实例

```
<?php
class MyClass
{
    const constant = '常量值';

    function showConstant() {
        echo  self::constant . PHP_EOL;
    }
}

echo MyClass::constant . PHP_EOL;

$classname = "MyClass";
echo $classname::constant . PHP_EOL; // 自 5.3.0 起

$class = new MyClass();
$class->showConstant();

echo $class::constant . PHP_EOL; // 自 PHP 5.3.0 起
?>
```

抽象类

任何一个类，如果它里面至少有一个方法是被声明为抽象的，那么这个类就必须被声明为抽象的。
定义为抽象的类不能被实例化。
被定义为抽象的方法只是声明了其调用方式（参数），不能定义其具体的功能实现。
继承一个抽象类的时候，子类必须定义父类中的所有抽象方法；另外，这些方法的访问控制必须和父类中一样（或者更为宽松）。例如某个抽象方法被声明为受保护的，那么子类中实现的方法就应该声明为受保护的或者公有的，而不能定义为私有的。

```
<?php
abstract class AbstractClass
{
 // 强制要求子类定义这些方法
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // 普通方法（非抽象方法）
    public function printOut() {
        print $this->getValue() . PHP_EOL;
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}

class ConcreteClass2 extends AbstractClass
{
    public function getValue() {
        return "ConcreteClass2";
```

```
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass2";
    }
}

$class1 = new ConcreteClass1;
$class1->printOut();
echo $class1->prefixValue('FOO_') . PHP_EOL;

$class2 = new ConcreteClass2;
$class2->printOut();
echo $class2->prefixValue('FOO_') . PHP_EOL;
?>
```

执行以上代码，输出结果为：

```
ConcreteClass1
FOO_ConcreteClass1
ConcreteClass2
FOO_ConcreteClass2
```

此外，子类方法可以包含父类抽象方法中不存在的可选参数。

例如，子类定义了一个可选参数，而父类抽象方法的声明里没有，则也是可以正常运行的。

```
<?php
abstract class AbstractClass
{
    // 我们的抽象方法仅需要定义需要的参数
    abstract protected function prefixName($name);

}

class ConcreteClass extends AbstractClass
```

```
{



    // 我们的子类可以定义父类签名中不存在的可选参数

    public function prefixName($name, $separator = ".") {

        if ($name == "Pacman") {

            $prefix = "Mr";

        } elseif ($name == "Pacwoman") {

            $prefix = "Mrs";

        } else {

            $prefix = "";

        }

        return "{$prefix}{$separator} {$name}";

    }

}



$class = new ConcreteClass;

echo $class->prefixName("Pacman"), "\n";

echo $class->prefixName("Pacwoman"), "\n";

?>
```

输出结果为：

```
Mr. Pacman

Mrs. Pacwoman
```

Static 关键字

声明类属性或方法为 static(静态)，就可以不实例化类而直接访问。

静态属性不能通过一个类已实例化的对象来访问（但静态方法可以）。

由于静态方法不需要通过对象即可调用，所以伪变量 $this 在静态方法中不可用。

静态属性不可以由对象通过 -> 操作符来访问。

自 PHP 5.3.0 起，可以用一个变量来动态调用类。但该变量的值不能为关键字 self，parent 或 static。

```
<?php

class Foo {

  public static $my_static = 'foo';
```

```
  public function staticValue() {
     return self::$my_static;
  }
}

print Foo::$my_static . PHP_EOL;
$foo = new Foo();

print $foo->staticValue() . PHP_EOL;
?>
```

执行以上程序，输出结果为：

```
foo
foo
```

Final 关键字

PHP 5 新增了一个 final 关键字。如果父类中的方法被声明为 final，则子类无法覆盖该方法。如果一个类被声明为 final，则不能被继承。
以下代码执行会报错：

```
<?php
class BaseClass {
   public function test() {
       echo "BaseClass::test() called" . PHP_EOL;
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() called"  . PHP_EOL;
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called"  . PHP_EOL;
```

```
    }
}
// 报错信息 Fatal error: Cannot override final method BaseClass::moreTesting()
?>
```

调用父类构造方法

PHP 不会在子类的构造方法中自动的调用父类的构造方法。要执行父类的构造方法，需要在子类的构造方法中调用 **parent::__construct()** 。

```
<?php
class BaseClass {
   function __construct() {
       print "BaseClass 类中构造方法" . PHP_EOL;
   }
}
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();  // 子类构造方法不能自动调用父类的构造方法
       print "SubClass 类中构造方法" . PHP_EOL;
   }
}
class OtherSubClass extends BaseClass {
    // 继承 BaseClass 的构造方法
}

// 调用 BaseClass 构造方法
$obj = new BaseClass();

// 调用 BaseClass、SubClass 构造方法
$obj = new SubClass();

// 调用 BaseClass 构造方法
$obj = new OtherSubClass();
?>
```

执行以上程序，输出结果为：

```
BaseClass 类中构造方法
BaseClass 类中构造方法
SubClass 类中构造方法
BaseClass 类中构造方法
```

PHP 7 新特性 PHP 正则表达式(PCRE)

点我分享笔记

反馈/建议

PHP 表单和用户输入

PHP 中的 $_GET 和 $_POST 变量用于检索表单中的信息，比如用户输入。

PHP 表单处理

有一点很重要的事情值得注意，当处理 HTML 表单时，PHP 能把来自 HTML 页面中的表单元素自动变成可供 PHP 脚本使用。

实例

下面的实例包含了一个 HTML 表单，带有两个输入框和一个提交按钮。

form.html 文件代码：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
<form action="welcome.php" method="post">
名字: <input type="text" name="fname">
年龄: <input type="text" name="age">
<input type="submit" value="提交">
</form>
</body>
</html>
```

当用户填写完上面的表单并点击提交按钮时，表单的数据会被送往名为 "welcome.php" 的 PHP 文件：

welcome.php 文件代码：

```
欢迎<?php echo $_POST["fname"]; ?>!<br>
你的年龄是 <?php echo $_POST["age"]; ?> 岁。
```

通过浏览器访问演示如下：

我们将在下一章中讲解 PHP 中的 $_GET 和 $_POST 变量。

PHP 获取下拉菜单的数据

PHP 下拉菜单单选

以下实例我们设置了下拉菜单三个选项，表单使用 GET 方式获取数据，action 属性值为空表示提交到当前脚本，我们可以通过 select 的 name 属性获取下拉菜单的值：

php_form_select.php 文件代码：

```
<?php
$q = isset($_GET['q'])? htmlspecialchars($_GET['q']) : '';
if($q) {
if($q =='RUNOOB') {
echo '菜鸟教程<br>http://www.runoob.com';
} else if($q =='GOOGLE') {
echo 'Google 搜索<br>http://www.google.com';
} else if($q =='TAOBAO') {
echo '淘宝<br>http://www.taobao.com';
}
} else {
?>
<form action="" method="get">
<select name="q">
<option value="">选择一个站点:</option>
<option value="RUNOOB">Runoob</option>
<option value="GOOGLE">Google</option>
<option value="TAOBAO">Taobao</option>
</select>
<input type="submit" value="提交">
</form>
<?php
}
?>
```

PHP 下拉菜单多选

如果下拉菜单是多选的（ multiple="multiple"），我们可以通过将设置 select **name="q[]"** 以数组的方式获取，以下使用 POST 方式提交，代码如下所示：

php_form_select_mul.php 文件代码：

```
<?php
$q = isset($_POST['q'])? $_POST['q'] : '';
if(is_array($q)) {
$sites = array(
'RUNOOB' => '菜鸟教程: http://www.runoob.com',
'GOOGLE' => 'Google 搜索: http://www.google.com',
'TAOBAO' => '淘宝: http://www.taobao.com',
);
foreach($q as $val) {
// PHP_EOL 为常量，用于换行
echo $sites[$val] . PHP_EOL;
}
} else {
?>
<form action="" method="post">
<select multiple="multiple" name="q[]">
<option value="">选择一个站点:</option>
<option value="RUNOOB">Runoob</option>
<option value="GOOGLE">Google</option>
<option value="TAOBAO">Taobao</option>
</select>
<input type="submit" value="提交">
</form>
<?php
}
?>
```

单选按钮表单

PHP 单选按钮表单中 name 属性的值是一致的，value 值是不同的，代码如下所示：

php_form_radio.php 文件代码：

```
<?php
$q = isset($_GET['q'])? htmlspecialchars($_GET['q']) : '';
if($q) {
if($q =='RUNOOB') {
echo '菜鸟教程<br>http://www.runoob.com';
} else if($q =='GOOGLE') {
echo 'Google 搜索<br>http://www.google.com';
} else if($q =='TAOBAO') {
echo '淘宝<br>http://www.taobao.com';
}
} else {
?><form action="" method="get">
<input type="radio" name="q" value="RUNOOB" />Runoob
<input type="radio" name="q" value="GOOGLE" />Google
<input type="radio" name="q" value="TAOBAO" />Taobao
<input type="submit" value="提交">
</form>
<?php
}
?>
```

checkbox 复选框

PHP checkbox 复选框可以选择多个值：

php_form_select_checkbox.php 文件代码：

```
<?php
$q = isset($_POST['q'])? $_POST['q'] : '';
if(is_array($q)) {
```

```php
$sites = array(
'RUNOOB' => '菜鸟教程: http://www.runoob.com',
'GOOGLE' => 'Google 搜索: http://www.google.com',
'TAOBAO' => '淘宝: http://www.taobao.com',
);
foreach($q as $val) {
// PHP_EOL 为常量，用于换行
echo $sites[$val] . PHP_EOL;
}
} else {
?><form action="" method="post">
<input type="checkbox" name="q[]" value="RUNOOB"> Runoob<br>
<input type="checkbox" name="q[]" value="GOOGLE"> Google<br>
<input type="checkbox" name="q[]" value="TAOBAO"> Taobao<br>
<input type="submit" value="提交">
</form>
<?php
}
?>
```

表单验证

我们应该尽可能的对用户的输入进行验证（通过客户端脚本）。浏览器验证速度更快，并且可以减轻服务器的压力。

如果用户输入需要插入数据库，您应该考虑使用服务器验证。在服务器验证表单的一种好的方式是，把表单的数据传给当前页面（异步提交的方式更好），而不是跳转到不同的页面。这样用户就可以在同一张表单页面得到错误信息。用户也就更容易发现错误了。

□ PHP 函数　　PHP $_GET 变量 □

□ PHP 超级全局变量　　PHP array_column() 函数 □

PHP 表单验证

本章节我们将介绍如何使用PHP验证客户端提交的表单数据。

PHP 表单验证

Note **在处理PHP表单时我们需要考虑安全性。**
本章节我们将展示PHP表单数据安全处理，为了防止黑客及垃圾信息我们需要对表单进行数据安全验证。

在本章节介绍的HTML表单中包含以下输入字段： 必须与可选文本字段，单选按钮，及提交按钮：

Error response

Error code 404.

Message: File not found.

Error code explanation: 404 = Nothing matches the given URI.

查看代码 »

上述表单验证规则如下：

字段	验证规则
名字	必须。 +只能包含字母和空格
E-mail	必须。 + 必须是一个有效的电子邮件地址（包含'@'和'.'）
网址	可选。如果存在，它必须包含一个有效的URL
备注	可选。多行输入字段（文本域）
性别	必须。 必须选择一个

首先让我们先看看纯HTML的表单代码：

文本字段

"名字", "E-mail", 及"网址"字段为文本输入元素，"备注"字段是 textarea。HTML代码如下所示:

```
“名字”: <input type="text" name="name">

E-mail: <input type="text" name="email">

网址: <input type="text" name="website">

备注: <textarea name="comment" rows="5" cols="40"></textarea>
```

单选按钮

"性别"字段是单选按钮，HTML代码如下所示：

```
性别:
<input type="radio" name="gender" value="female">女
<input type="radio" name="gender" value="male">男
```

表单元素

HTML 表单代码如下所示:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

该表单使用 **method="post"** 方法来提交数据。

Note **什么是 $_SERVER["PHP_SELF"] 变量?**

$_SERVER["PHP_SELF"]是超级全局变量，返回当前正在执行脚本的文件名，与 document root相关。

所以， $_SERVER["PHP_SELF"] 会发送表单数据到当前页面，而不是跳转到不同的页面。

Note **什么是 htmlspecialchars()方法?**

htmlspecialchars() 函数把一些预定义的字符转换为 HTML 实体。
预定义的字符是：

- & （和号） 成为 &
- " （双引号） 成为 "
- ' （单引号） 成为 '
- < （小于） 成为 <
- > （大于） 成为 >

PHP表单中需引起注重的地方？

$_SERVER["PHP_SELF"] 变量有可能会被黑客使用！

当黑客使用跨网站脚本的HTTP链接来攻击时，$_SERVER["PHP_SELF"]服务器变量也会被植入脚本。原因就是跨网站脚本是附在执行文件的路径后面的，因此$_SERVER["PHP_SELF"]的字符串就会包含HTTP链接后面的JavaScript程序代码。

Note **XSS又叫 CSS (Cross-Site Script) ,跨站脚本攻击。恶意攻击者往Web页面里插入恶意html代码，当用户浏览该页之时，嵌入其中Web里面的html代码会被执行，从而达到恶意用户的特殊目的。**

指定以下表单文件名为 "test_form.php":

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

现在，我们使用URL来指定提交地址 "test_form.php",以上代码修改为如下所示:

```
<form method="post" action="test_form.php">
```

这样做就很好了。

但是，考虑到用户会在浏览器地址栏中输入以下地址:

```
http://www.runoob.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E
```

以上的 URL 中，将被解析为如下代码并执行：

```
<form method="post" action="test_form.php/"><script>alert('hacked')</script>
```

代码中添加了 script 标签，并添加了alert命令。 当页面载入时会执行该Javascript代码（用户会看到弹出框）。 这仅仅只是一个简单的实例来说明PHP_SELF变量会被黑客利用。

请注意， **任何JavaScript代码可以添加在<script>标签中！** 黑客可以利用这点重定向页面到另外一台服务器的页面上，页面 代码文件中可以保护恶意代码，代码可以修改全局变量或者获取用户的表单数据。

如何避免 $_SERVER["PHP_SELF"] 被利用?

$_SERVER["PHP_SELF"] 可以通过 htmlspecialchars() 函数来避免被利用。

form 代码如下所示：

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

htmlspecialchars() 把一些预定义的字符转换为 HTML 实体。现在如果用户想利用 PHP_SELF 变量, 结果将输出如下所示：

```
<form method="post" action="test_form.php/&quot;&gt;&lt;script&gt;alert('hacked')&lt;/script&gt;">
```

尝试该漏洞失败!

使用 PHP 验证表单数据

首先我们对用户所有提交的数据都通过 PHP 的 htmlspecialchars() 函数处理。

当我们使用 htmlspecialchars() 函数时，在用户尝试提交以下文本域:

```
<script>location.href('http://www.runoob.com')</script>
```

该代码将不会被执行，因为它会被保存为HTML转义代码，如下所示：

```
&lt;script&gt;location.href('http://www.runoob.com')&lt;/script&gt;
```

以上代码是安全的，可以正常在页面显示或者插入邮件中。

当用户提交表单时，我们将做以下两件事情：

1. 使用 PHP trim() 函数去除用户输入数据中不必要的字符 (如：空格，tab，换行)。
2. 使用PHP stripslashes()函数去除用户输入数据中的反斜杠 (\)

接下来让我们将这些过滤的函数写在一个我们自己定义的函数中，这样可以大大提高代码的复用性。

将函数命名为 test_input()。

现在，我们可以通过test_input()函数来检测 $_POST 中的所有变量, 脚本代码如下所示：

实例

```php
<?php
// 定义变量并默认设置为空值
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST")
{
$name = test_input($_POST["name"]);
$email = test_input($_POST["email"]);
$website = test_input($_POST["website"]);
$comment = test_input($_POST["comment"]);
$gender = test_input($_POST["gender"]);
}
function test_input($data)
{
$data = trim($data);
$data = stripslashes($data);
$data = htmlspecialchars($data);
return $data;
}
?>
```

运行实例 »

注意我们在执行以上脚本时，会通过$_SERVER["REQUEST_METHOD"]来检测表单是否被提交 。如果 REQUEST_METHOD 是 POST, 表单将被提交 - 数据将被验证。如果表单未提交将跳过验证并显示空白。

在以上实例中使用输入项都是可选的，即使用户不输入任何数据也可以正常显示。

在接下来的章节中我们将介绍如何对用户输入的数据进行验证。

□ PHP 超级全局变量　　PHP array_column() 函数 □

□ PHP curl_version函数　　PHP 表单 – 验证邮件和URL □

PHP 表单 - 必需字段

本章节我们将介绍如何设置表单必需字段及错误信息。

PHP - 必需字段

在上一章节我们已经介绍了表的验证规则，我们可以看到"名字", "E-mail", 和 "性别" 字段是必需的，各字段不能为空。

字段	验证规则
名字	必需。 + 只能包含字母和空格

E-mail	必需。 + 必需包含一个有效的电子邮件地址（包含"@"和"."）
网址	可选。 如果存在，它必需包含一个有效的URL
备注	可选。多行字段（文本域）。
性别	必需。必需选择一个。

如果在前面的章节中，所有输入字段都是可选的。

在以下代码中我们加入了一些新的变量: $nameErr, $emailErr, $genderErr, 和 $websiteErr.。这些错误变量将显示在必需字段上。 我们还为每个$_POST变量增加了一个if else语句。 这些语句将检查 $_POST 变量是 否为空（使用php的 empty() 函数）。如果为空，将显示对应的错误信息。 如果不为空，数据将传递给test_input() 函数：

```
<?php

// 定义变量并默认设为空值

$nameErr = $emailErr = $genderErr = $websiteErr = "";

$name = $email = $gender = $comment = $website = "";



if ($_SERVER["REQUEST_METHOD"] == "POST") {

  if (empty($_POST["name"])) {

    $nameErr = "名字是必需的。";

  } else {

    $name = test_input($_POST["name"]);

  }



  if (empty($_POST["email"])) {

    $emailErr = "邮箱是必需的。";

  } else {

    $email = test_input($_POST["email"]);

  }



  if (empty($_POST["website"])) {

    $website = "";

  } else {

    $website = test_input($_POST["website"]);

  }



  if (empty($_POST["comment"])) {
```

```
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "性别是必需的。";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - 显示错误信息

在以下的HTML实例表单中，我们为每个字段中添加了一些脚本， 各个脚本会在信息输入错误时显示错误信息。(如果用户未填写信息就提交表单则会输出错误信息):

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']);?>">
   名字: <input type="text" name="name">
   <span class="error">* <?php echo $nameErr;?></span>
   <br><br>
   E-mail: <input type="text" name="email">
   <span class="error">* <?php echo $emailErr;?></span>
   <br><br>
   网址: <input type="text" name="website">
   <span class="error"><?php echo $websiteErr;?></span>
   <br><br>
   备注: <textarea name="comment" rows="5" cols="40"></textarea>
   <br><br>
   性别:
   <input type="radio" name="gender" value="female">女
   <input type="radio" name="gender" value="male">男
   <span class="error">* <?php echo $genderErr;?></span>
```

```
    <br><br>

    <input type="submit" name="submit" value="Submit">

</form>
```

查看代码 »

PHP curl_version函数 PHP 表单 – 验证邮件和URL

点我分享笔记

反馈/建议

PHP 表单 – 必需字段 PHPC完整表单实例

PHP 表单 - 验证邮件和URL

本章节我们将介绍如何验证 names(名称), e-mails（邮件），和 URLs。

PHP - 验证名称

以下代码将通过简单的方式来检测 name 字段是否包含字母和空格，如果 name 字段值不合法，将输出错误信息：

```
$name = test_input($_POST["name"]);

if (!preg_match("/^[a-zA-Z ]*$/",$name)) {

  $nameErr = "只允许字母和空格";

}
```

preg_match — 进行正则表达式匹配。

Note 语法：

int preg_match (string $pattern , string $subject [, array $matches [, int $flags]])

在 subject 字符串中搜索与 pattern 给出的正则表达式相匹配的内容。如果提供了 matches ，则其会被搜索的结果所填充。$matches[0] 将包含与整个模式匹配的文本，$matches[1] 将包含与第一个捕获的括号中的子模式所匹配的文本，以此类推。

PHP - 验证邮件

以下代码将通过简单的方式来检测 e-mail 地址是否合法。如果 e-mail 地址不合法，将输出错误信息：

```
$email = test_input($_POST["email"]);

if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email)) {

  $emailErr = "非法邮箱格式";

}
```

PHP - 验证 URL

以下代码将检测URL地址是否合法 (以下正则表达式运行URL中含有破折号:"-"), 如果 URL 地址不合法，将输出错误信息：

```
$website = test_input($_POST["website"]);

if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {

  $websiteErr = "非法的 URL 的地址";

}
```

PHP - 验证 Name, E-mail, 和 URL

代码如下所示:

```
<?php

// 定义变量并默认设置为空值

$nameErr = $emailErr = $genderErr = $websiteErr = "";

$name = $email = $gender = $comment = $website = "";



if ($_SERVER["REQUEST_METHOD"] == "POST") {

   if (empty($_POST["name"])) {

      $nameErr = "Name is required";

      } else {

         $name = test_input($_POST["name"]);

         // 检测名字是否只包含字母跟空格

         if (!preg_match("/^[a-zA-Z ]*$/",$name)) {

         $nameErr = "只允许字母和空格";

         }

     }



   if (empty($_POST["email"])) {

      $emailErr = "Email is required";
```

```
    } else {
       $email = test_input($_POST["email"]);
       // 检测邮箱是否合法
       if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email)) {
          $emailErr = "非法邮箱格式";
       }
    }

    if (empty($_POST["website"])) {
       $website = "";
    } else {
       $website = test_input($_POST["website"]);
       // 检测 URL 地址是否合法
      if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
          $websiteErr = "非法的 URL 的地址";
       }
    }

    if (empty($_POST["comment"])) {
       $comment = "";
    } else {
       $comment = test_input($_POST["comment"]);
    }

    if (empty($_POST["gender"])) {
       $genderErr = "性别是必需的";
    } else {
       $gender = test_input($_POST["gender"]);
    }
}
?>
```

运行实例 »

□ PHP 表单 – 必需字段 PHP完整表单实例 □

□ PHP 表单 – 验证邮件和URL PHP 超级全局变量 □

PHP 完整表单实例

本章节将介绍如何让用户在点击"提交（submit）"按钮提交数据前保证所有字段正确输入。

PHP - 在表单中确保输入值

在用户点击提交按钮后，为确保字段值是否输入正确，我们在HTML的input元素中插添加PHP脚本， 各字段名为: name, email, 和 website。 在备注中的 textarea 字段中，我们将脚本放于 <textarea> 和 </textarea> 标签之间。

PHP脚本输出值为： $name, $email, $website, 和 $comment 变量。

然后，我们同样需要检查被选中的单选按钮， 对于这一点，我们 必须设置好checked属性(不是radio按钮的 value 属性)：

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

   名字: <input type="text" name="name" value="<?php echo $name;?>">

   <span class="error">* <?php echo $nameErr;?></span>

   <br><br>

   E-mail: <input type="text" name="email" value="<?php echo $email;?>">

   <span class="error">* <?php echo $emailErr;?></span>

   <br><br>

   网址: <input type="text" name="website" value="<?php echo $website;?>">

   <span class="error"><?php echo $websiteErr;?></span>

   <br><br>

   备注: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>

   <br><br>

   性别:
```

```
   <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female") echo "checked";?>  value="female"
>女

   <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male") echo "checked";?>  value="male">男

   <span class="error">* <?php echo $genderErr;?></span>

   <br><br>

   <input type="submit" name="submit" value="Submit">

</form>
```

PHP - 完整表单实例

以下是完整的PHP表单验证实例代码：

实例

Error response

Error code 404.

Message: File not found.

Error code explanation: 404 = Nothing matches the given URI.

运行实例 »

实例中执行结果类似如下图所示：

PHP 表单验证实例

* 必须字段。

名字: Runoob *

E-mail: test@runoob.com *

网址: http://www.runoob.cor

备注: 菜鸟教程

性别: 女 男 *

Submit

您输入的内容是:

Runoob
test@runoob.com
http://www.runoob.com
菜鸟教程
male

PHP 表单 – 验证邮件和URL

PHP 超级全局变量

点我分享笔记

反馈/建议

PHP 表单

PHP $_POST 变量

PHP $_GET 变量

在 PHP 中，预定义的 $_GET 变量用于收集来自 method="get" 的表单中的值。

$_GET 变量

预定义的 $_GET 变量用于收集来自 method="get" 的表单中的值。

从带有 GET 方法的表单发送的信息，对任何人都是可见的（会显示在浏览器的地址栏），并且对发送信息的量也有限制。

实例

form.html 文件代码如下：

```
<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>



<form action="welcome.php" method="get">

名字: <input type="text" name="fname">

年龄: <input type="text" name="age">

<input type="submit" value="提交">

</form>



</body>

</html>
```

当用户点击 "Submit" 按钮时，发送到服务器的 URL 如下所示：

```
http://www.runoob.com/welcome.php?fname=Runoob&amp;age=3
```

"welcome.php" 文件现在可以通过 $_GET 变量来收集表单数据了（请注意，表单域的名称会自动成为 $_GET 数组中的键）：

```
欢迎 <?php echo $_GET["fname"]; ?>!<br>

你的年龄是 <?php echo $_GET["age"]; ?>  岁。
```

以上表单执行演示：

何时使用 method="get"?

在 HTML 表单中使用 method="get" 时，所有的变量名和值都会显示在 URL 中。

注释：所以在发送密码或其他敏感信息时，不应该使用这个方法！

然而，正因为变量显示在 URL 中，因此可以在收藏夹中收藏该页面。在某些情况下，这是很有用的。

注释：HTTP GET 方法不适合大型的变量值。它的值是不能超过 2000 个字符的。

PHP 表单 PHP $_POST 变量

点我分享笔记

反馈/建议

PHP $_GET 变量 PHP 多维数组

PHP $_POST 变量

在 PHP 中，预定义的 $_POST 变量用于收集来自 method="post" 的表单中的值。

$_POST 变量

预定义的 $_POST 变量用于收集来自 method="post" 的表单中的值。

从带有 POST 方法的表单发送的信息，对任何人都是不可见的（不会显示在浏览器的地址栏），并且对发送信息的量也没有限制。

注释：然而，默认情况下，POST 方法的发送信息的量最大值为 8 MB（可通过设置 php.ini 文件中的 post_max_size 进行更改）。

实例

form.html 文件代码如下：

```
<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>



<form action="welcome.php" method="post">

名字: <input type="text" name="fname">

年龄: <input type="text" name="age">

<input type="submit" value="提交">

</form>



</body>

</html>
```

当用户点击 "提交" 按钮时，URL 类似如下所示：

```
http://www.runoob.com/welcome.php
```

"welcome.php" 文件现在可以通过 $_POST 变量来收集表单数据了（请注意，表单域的名称会自动成为 $_POST 数组中的键）：

```
欢迎 <?php echo $_POST["fname"]; ?>!<br>

你的年龄是 <?php echo $_POST["age"]; ?>  岁。
```

通过浏览器访问演示如下：

何时使用 method="post"?

从带有 POST 方法的表单发送的信息，对任何人都是不可见的，并且对发送信息的量也没有限制。

然而，由于变量不显示在 URL 中，所以无法把页面加入书签。

PHP $_REQUEST 变量

预定义的 $_REQUEST 变量包含了 $_GET、$_POST 和 $_COOKIE 的内容。

$_REQUEST 变量可用来收集通过 GET 和 POST 方法发送的表单数据。

实例

你可以将 "welcome.php" 文件修改为如下代码，它可以接受 $_GET、$_POST等数据。

```
欢迎 <?php echo $_REQUEST["fname"]; ?>!<br>

你的年龄是 <?php echo $_REQUEST["age"]; ?>  岁。
```

□ PHP $_GET 变量　　PHP 多维数组 □

□点我分享笔记

反馈/建议

PHP 多维数组

一个数组中的值可以是另一个数组，另一个数组的值也可以是一个数组。依照这种方式，我们可以创建二维或者三维数组：

实例

```php
<?php
// 二维数组:
$cars = array
(
array("Volvo",100,96),
array("BMW",60,59),
array("Toyota",110,100)
);
?>
```

运行实例 »

PHP - 多维数组

多维数组是包含一个或多个数组的数组。

在多维数组中，主数组中的每一个元素也可以是一个数组，子数组中的每一个元素也可以是一个数组。

实例

在这个实例中，我们创建了一个自动分配 ID 键的多维数组：

实例

```php
<?php
$sites = array
(
    "runoob"=>array
    (
        "菜鸟教程",
        "http://www.runoob.com"
    ),
    "google"=>array
    (
        "Google 搜索",
        "http://www.google.com"
    ),
    "taobao"=>array
    (
        "淘宝",
        "http://www.taobao.com"
    )
);
print("<pre>"); // 格式化输出数组
print_r($sites);
print("</pre>");
?>
```

上面的数组将输出如下：

```
Array
(
    [runoob] => Array
        (
            [0] => 菜鸟教程
            [1] => http://www.runoob.com
        )

    [google] => Array
        (
            [0] => Google 搜索
            [1] => http://www.google.com
        )

    [taobao] => Array
        (
            [0] => 淘宝
            [1] => http://www.taobao.com
        )

)
```

实例 2

让我们试着显示上面数组中的某个值：

```
echo $sites['runoob'][0] . '地址为：' . $sites['runoob'][1];
```

上面的代码将输出：

菜鸟教程地址为：http://www.runoob.com

□ PHP $_POST 变量　　PHP date() 函数 □

□点我分享笔记

反馈/建议

PHP date() 函数

PHP date() 函数用于格式化时间/日期。

PHP date() 函数

PHP date() 函数可把时间戳格式化为可读性更好的日期和时间。

时间戳是一个字符序列，表示一定的事件发生的日期/时间。

语法

```
string date ( string $format [, int $timestamp ] )
```

参数	描述
format	必需。规定时间戳的格式。
timestamp	可选。规定时间戳。默认是当前的日期和时间。

PHP Date() - 格式化日期

date() 函数的第一个必需参数 *format* 规定了如何格式化日期/时间。

这里列出了一些可用的字符：

d - 代表月中的天 (01 - 31)

m - 代表月 (01 - 12)

Y - 代表年 (四位数)

如需了解 *format* 参数中可用的所有字符列表，请查阅我们的 PHP Date 参考手册，date() 函数。

可以在字母之间插入其他字符，比如 "/"、"." 或者 "-"，这样就可以增加附加格式了：

```
<?php
echo date("Y/m/d") . "<br>";
echo date("Y.m.d") . "<br>";
echo date("Y-m-d");
?>
```

上面代码的输出如下所示：

```
2016/10/21
2016.10.21
2016-10-21
```

格式字串可以识别以下 format 参数的字符串

format 字符	说明	返回值例子
日	---	---
d	月份中的第几天，有前导零的 2 位数字	*01* 到 *31*
D	星期中的第几天，文本表示，3 个字母	*Mon* 到 *Sun*
j	月份中的第几天，没有前导零	*1* 到 *31*

format 字符	说明	返回值例子
l（"L"的小写字母）	星期几，完整的文本格式	*Sunday* 到 *Saturday*
N	ISO-8601 格式数字表示的星期中的第几天（PHP 5.1.0 新加）	*1*（表示星期一）到 *7*（表示星期天）
S	每月天数后面的英文后缀，2 个字符	*st*，*nd*，*rd* 或者 *th*。可以和 *j* 一起用
w	星期中的第几天，数字表示	*0*（表示星期天）到 *6*（表示星期六）
z	年份中的第几天	*0* 到 *365*
星期	—	—
W	ISO-8601 格式年份中的第几周，每周从星期一开始（PHP 4.1.0 新加的）	例如：*42*（当年的第 42 周）
月	—	—
F	月份，完整的文本格式，例如 January 或者 March	*January* 到 *December*
m	数字表示的月份，有前导零	*01* 到 *12*
M	三个字母缩写表示的月份	*Jan* 到 *Dec*
n	数字表示的月份，没有前导零	*1* 到 *12*
t	给定月份所应有的天数	*28* 到 *31*
年	—	—
L	是否为闰年	如果是闰年为 *1*，否则为 *0*
o	ISO-8601 格式年份数字。这和 *Y* 的值相同，只除了如果 ISO 的星期数（*W*）属于前一年或下一年，则用那一年。（PHP 5.1.0 新加）	Examples: *1999* or *2003*
Y	4 位数字完整表示的年份	例如：*1999* 或 *2003*
y	2 位数字表示的年份	例如：*99* 或 *03*
时间	—	—
a	小写的上午和下午值	*am* 或 *pm*
A	大写的上午和下午值	*AM* 或 *PM*
B	Swatch Internet 标准时	*000* 到 *999*
g	小时，12 小时格式，没有前导零	*1* 到 *12*
G	小时，24 小时格式，没有前导零	*0* 到 *23*
h	小时，12 小时格式，有前导零	*01* 到 *12*
H	小时，24 小时格式，有前导零	*00* 到 *23*
i	有前导零的分钟数	*00* 到 *59>*
s	秒数，有前导零	*00* 到 *59>*

format 字符	说明	返回值例子
u	毫秒 （PHP 5.2.2 新加）。需要注意的是 **date()** 函数总是返回 *000000* 因为它只接受 integer 参数， 而 DateTime::format() 才支持毫秒。	示例: *654321*
时区	—	—
e	时区标识（PHP 5.1.0 新加）	例如：*UTC*，*GMT*，*Atlantic/Azores*
I	是否为夏令时	如果是夏令时为 *1*，否则为 *0*
O	与格林威治时间相差的小时数	例如：*+0200*
P	与格林威治时间（GMT）的差别，小时和分钟之间有冒号分隔（PHP 5.1.3 新加）	例如：*+02:00*
T	本机所在的时区	例如：*EST*，*MDT*（【译者注】在 Windows 下为完整文本格式，例如"Eastern Standard Time"，中文版会显示"中国标准时间"）。
Z	时差偏移量的秒数。UTC 西边的时区偏移量总是负的，UTC 东边的时区偏移量总是正的。	*-43200* 到 *43200*
完整的日期／时间	—	—
c	ISO 8601 格式的日期（PHP 5 新加）	2004-02-12T15:19:21+00:00
r	RFC 822 格式的日期	例如：*Thu, 21 Dec 2000 16:01:07 +0200*
U	从 Unix 纪元（January 1 1970 00:00:00 GMT）开始至今的秒数	参见 time()

完整的 PHP Date 参考手册

如需查看所有日期函数的完整参考手册，请访问我们的 完整的 PHP Date 参考手册。

该参考手册提供了每个函数的简要描述和应用实例！

□ PHP 多维数组　　PHP include 和 require □

PHP 包含文件

PHP include 和 require 语句

在 PHP 中，您可以在服务器执行 PHP 文件之前在该文件中插入一个文件的内容。

include 和 require 语句用于在执行流中插入写在其他文件中的有用的代码。

include 和 require 除了处理错误的方式不同之外，在其他方面都是相同的：

- require 生成一个致命错误（E_COMPILE_ERROR），在错误发生后脚本会停止执行。
- include 生成一个警告（E_WARNING），在错误发生后脚本会继续执行。

因此，如果您希望继续执行，并向用户输出结果，即使包含文件已丢失，那么请使用 include。否则，在框架、CMS 或者复杂的 PHP 应用程序编程中，请始终使用 require 向执行流引用关键文件。这有助于提高应用程序的安全性和完整性，在某个关键文件意外丢失的情况下。

包含文件省去了大量的工作。这意味着您可以为所有网页创建标准页头、页脚或者菜单文件。然后，在页头需要更新时，您只需更新这个页头包含文件即可。

语法

```
include 'filename';

或者

require 'filename';
```

PHP include 和 require 语句

基础实例

假设您有一个标准的页头文件，名为 "header.php"。如需在页面中引用这个页头文件，请使用 include/require：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<?php include 'header.php'; ?>
<h1>欢迎来到我的主页!</h1>
<p>一些文本。</p>

</body>
```

```
</html>
```

实例 2

假设我们有一个在所有页面中使用的标准菜单文件。

"menu.php":

```
echo '<a href="/">主页</a>

<a href="/html">HTML 教程</a>

<a href="/php">PHP 教程</a>';
```

网站中的所有页面均应引用该菜单文件。以下是具体的做法：

```
<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>

<body>



<div class="leftmenu">

<?php include 'menu.php'; ?>

</div>

<h1>欢迎来到我的主页!</h1>

<p>一些文本。</p>



</body>

</html>
```

实例 3

假设我们有一个定义变量的包含文件（"vars.php"）：

```
<?php

$color='red';

$car='BMW';

?>
```

这些变量可用在调用文件中：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<h1>欢迎来到我的主页!</h1>
<?php
include 'vars.php';
echo "I have a $color $car"; // I have a red BMW
?>

</body>
</html>
```

PHP date() 函数 PHP 文件

1 篇笔记

#1

写笔记

include 和 require 的区别

require 一般放在 PHP 文件的最前面，程序在执行前就会先导入要引用的文件；
include 一般放在程序的流程控制中，当程序执行时碰到才会引用，简化程序的执行流程。
require 引入的文件有错误时，执行会中断，并返回一个致命错误；
include 引入的文件有错误时，会继续执行，并返回一个警告。
更多内容可参考：PHP 中 include 和 require 的区别详解

1966vng 2个月前 (07-19)

反馈/建议

PHP 文件处理

fopen() 函数用于在 PHP 中打开文件。

打开文件

fopen() 函数用于在 PHP 中打开文件。

此函数的第一个参数含有要打开的文件的名称，第二个参数规定了使用哪种模式来打开文件：

```
<html>
<body>

<?php
$file=fopen("welcome.txt","r");
?>

</body>
</html>
```

文件可能通过下列模式来打开：

模式	描述
r	只读。在文件的开头开始。
r+	读/写。在文件的开头开始。
w	只写。打开并清空文件的内容；如果文件不存在，则创建新文件。
w+	读/写。打开并清空文件的内容；如果文件不存在，则创建新文件。
a	追加。打开并向文件末尾进行写操作，如果文件不存在，则创建新文件。
a+	读/追加。通过向文件末尾写内容，来保持文件内容。
x	只写。创建新文件。如果文件已存在，则返回 FALSE 和一个错误。
x+	读/写。创建新文件。如果文件已存在，则返回 FALSE 和一个错误。

注释：如果 fopen() 函数无法打开指定文件，则返回 0 (false)。

实例

如果 fopen() 函数不能打开指定的文件，下面的实例会生成一段消息：

```
<html>
<body>

<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
?>

</body>
</html>
```

关闭文件

fclose() 函数用于关闭打开的文件：

```
<?php
$file = fopen("test.txt","r");

//执行一些代码

fclose($file);
?>
```

检测文件末尾（EOF）

feof() 函数检测是否已到达文件末尾（EOF）。

在循环遍历未知长度的数据时，feof() 函数很有用。

注释：在 w、a 和 x 模式下，您无法读取打开的文件！

```
if (feof($file)) echo "文件结尾";
```

逐行读取文件

fgets() 函数用于从文件中逐行读取文件。

注释：在调用该函数之后，文件指针会移动到下一行。

实例

下面的实例逐行读取文件，直到文件末尾为止：

```
<?php
$file = fopen("welcome.txt", "r") or exit("无法打开文件!");
// 读取文件每一行，直到文件结尾
while(!feof($file))
{
    echo fgets($file). "<br>";
}
fclose($file);
?>
```

逐字符读取文件

fgetc() 函数用于从文件中逐字符地读取文件。

注释：在调用该函数之后，文件指针会移动到下一个字符。

实例

下面的实例逐字符地读取文件，直到文件末尾为止：

```
<?php
$file=fopen("welcome.txt","r") or exit("无法打开文件!");
while (!feof($file))
{
    echo fgetc($file);
}
fclose($file);
?>
```

PHP Filesystem 参考手册

如需查看 PHP 文件系统函数的完整参考手册，请访问我们的PHP Filesystem 参考手册。

□ PHP include 和 require　　PHP 文件上传 □

PHP对逗号分隔符文件（*.csv）的处理。
当如果你需要处理的数据比较少时可以使用csv文件（这是一类文本文件）存储数据更加便利。
比如在php代码同目录下有一个a.csv文件，内容如下：(注意逗号是半角英文)
小王,小红,小明,小凡
php代码如何：

```
<?php

$fh=fopen("a.csv","r");//这里我们只是读取数据，所以采用只读打开文件流

$arr=fgetcsv($fh);//这个函数就是读取CSV文件的函数，他把文本读入后转为数组存储在$arr中

fclose($fh);
```

```
foreach($arr as $key=>$value){echo $value;}//循环输出所有的值

?>
```

注意：CSV文本编码必须和HTML的编码相同，否则用php写到HTML中，用户会看到乱码。也可以使用 iconv 转码函数进行转码。

Vladimir1年前 (2017-07-17)

反馈/建议

PHP 文件上传

通过 PHP，可以把文件上传到服务器。

本章节实例在 test 项目下完成，目录结构为：

```
test

|-----upload             # 文件上传的目录

|-----form.html          # 表单文件

|-----upload_file.php    # php 上传代码
```

源码下载

创建一个文件上传表单

允许用户从表单上传文件是非常有用的。

请看下面这个供上传文件的 HTML 表单：

```
<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

</head>
```

```
<body>



<form action="upload_file.php" method="post" enctype="multipart/form-data">

    <label for="file">文件名：</label>

    <input type="file" name="file" id="file"><br>

    <input type="submit" name="submit" value="提交">

</form>



</body>

</html>
```

将以上代码保存到 form.html 文件中。

有关上面的 HTML 表单的一些注意项列举如下：

> **<form>** 标签的 **enctype** 属性规定了在提交表单时要使用哪种内容类型。在表单需要二进制数据时，比如文件内容，请使用 **"multipart/form-data"**。
>
> **<input>** 标签的 **type="file"** 属性规定了应该把输入作为文件来处理。举例来说，当在浏览器中预览时，会看到输入框旁边有一个浏览按钮。

注释： 允许用户上传文件是一个巨大的安全风险。请仅仅允许可信的用户执行文件上传操作。

创建上传脚本

"upload_file.php" 文件含有供上传文件的代码：

```
<?php

if ($_FILES["file"]["error"] > 0)

{

    echo "错误：" . $_FILES["file"]["error"] . "<br>";

}

else

{

    echo "上传文件名: " . $_FILES["file"]["name"] . "<br>";

    echo "文件类型: " . $_FILES["file"]["type"] . "<br>";

    echo "文件大小: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";

    echo "文件临时存储的位置: " . $_FILES["file"]["tmp_name"];

}

?>
```

通过使用 PHP 的全局数组 $_FILES，你可以从客户计算机向远程服务器上传文件。

第一个参数是表单的 input name，第二个下标可以是 "name"、"type"、"size"、"tmp_name" 或 "error"。如下所示：

$_FILES["file"]["name"] - 上传文件的名称

$_FILES["file"]["type"] - 上传文件的类型

$_FILES["file"]["size"] - 上传文件的大小，以字节计

$_FILES["file"]["tmp_name"] - 存储在服务器的文件的临时副本的名称

$_FILES["file"]["error"] - 由文件上传导致的错误代码

这是一种非常简单文件上传方式。基于安全方面的考虑，您应当增加有关允许哪些用户上传文件的限制。

上传限制

在这个脚本中，我们增加了对文件上传的限制。用户只能上传 .gif、.jpeg、.jpg、.png 文件，文件大小必须小于 200 kB：

```
<?php

// 允许上传的图片后缀

$allowedExts = array("gif", "jpeg", "jpg", "png");

$temp = explode(".", $_FILES["file"]["name"]);

$extension = end($temp);        // 获取文件后缀名

if ((($_FILES["file"]["type"] == "image/gif")

|| ($_FILES["file"]["type"] == "image/jpeg")

|| ($_FILES["file"]["type"] == "image/jpg")

|| ($_FILES["file"]["type"] == "image/pjpeg")

|| ($_FILES["file"]["type"] == "image/x-png")

|| ($_FILES["file"]["type"] == "image/png"))

&& ($_FILES["file"]["size"] < 204800)    // 小于 200 kb

&& in_array($extension, $allowedExts))

{

    if ($_FILES["file"]["error"] > 0)

    {

        echo "错误：: " . $_FILES["file"]["error"] . "<br>";

    }

    else

    {

        echo "上传文件名: " . $_FILES["file"]["name"] . "<br>";

        echo "文件类型: " . $_FILES["file"]["type"] . "<br>";

        echo "文件大小: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";

        echo "文件临时存储的位置: " . $_FILES["file"]["tmp_name"];
```

```
    }
}
else
{
    echo "非法的文件格式";
}
?>
```

保存被上传的文件

上面的实例在服务器的 PHP 临时文件夹中创建了一个被上传文件的临时副本。
这个临时的副本文件会在脚本结束时消失。要保存被上传的文件，我们需要把它拷贝到另外的位置：

```
<?php
// 允许上传的图片后缀
$allowedExts = array("gif", "jpeg", "jpg", "png");
$temp = explode(".", $_FILES["file"]["name"]);
echo $_FILES["file"]["size"];
$extension = end($temp);     // 获取文件后缀名
if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/jpg")
|| ($_FILES["file"]["type"] == "image/pjpeg")
|| ($_FILES["file"]["type"] == "image/x-png")
|| ($_FILES["file"]["type"] == "image/png"))
&& ($_FILES["file"]["size"] < 204800)   // 小于 200 kb
&& in_array($extension, $allowedExts))
{
    if ($_FILES["file"]["error"] > 0)
    {
        echo "错误：: " . $_FILES["file"]["error"] . "<br>";
    }
    else
    {
```

```
        echo "上传文件名: " . $_FILES["file"]["name"] . "<br>";
        echo "文件类型: " . $_FILES["file"]["type"] . "<br>";
        echo "文件大小: " . ($_FILES["file"]["size"] / 1024) . " kB<br>";
        echo "文件临时存储的位置: " . $_FILES["file"]["tmp_name"] . "<br>";

        // 判断当期目录下的 upload 目录是否存在该文件
        // 如果没有 upload 目录，你需要创建它，upload 目录权限为 777
        if (file_exists("upload/" . $_FILES["file"]["name"]))
        {
            echo $_FILES["file"]["name"] . " 文件已经存在。 ";
        }
        else
        {
            // 如果 upload 目录不存在该文件则将文件上传到 upload 目录下
            move_uploaded_file($_FILES["file"]["tmp_name"], "upload/" . $_FILES["file"]["name"]);
            echo "文件存储在: " . "upload/" . $_FILES["file"]["name"];
        }
    }
}
else
{
    echo "非法的文件格式";
}
?>
```

上面的脚本检测了文件是否已存在，如果不存在，则把文件拷贝到名为 "upload" 的目录下。
文件上传演示操作如下所示：

PHP 文件 PHP Cookie

点我分享笔记

反馈/建议

PHP Cookie

cookie 常用于识别用户。

Cookie 是什么？

cookie 常用于识别用户。cookie 是一种服务器留在用户计算机上的小文件。每当同一台计算机通过浏览器请求页面时，这台计算机将会发送 cookie。通过 PHP，您能够创建并取回 cookie 的值。

如何创建 Cookie?

setcookie() 函数用于设置 cookie。

注释： setcookie() 函数必须位于 <html> 标签之前。

语法

```
setcookie(name, value, expire, path, domain);
```

实例 1

在下面的例子中，我们将创建名为 "user" 的 cookie，并为它赋值 "runoob"。我们也规定了此 cookie 在一小时后过期：

```
<?php
setcookie("user", "runoob", time()+3600);
?>

<html>
.....
```

注释：在发送 cookie 时，cookie 的值会自动进行 URL 编码，在取回时进行自动解码。（为防止 URL 编码，请使用 setrawcookie() 取而代之。）

实例 2

您还可以通过另一种方式设置 cookie 的过期时间。这也许比使用秒表示的方式简单。

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "runoob", $expire);
?>

<html>
.....
```

在上面的实例中，过期时间被设置为一个月（*60 秒 * 60 分 * 24 小时 * 30 天*）。

如何取回 Cookie 的值？

PHP 的 $_COOKIE 变量用于取回 cookie 的值。

在下面的实例中，我们取回了名为 "user" 的 cookie 的值，并把它显示在了页面上：

```
<?php
// 输出 cookie 值
echo $_COOKIE["user"];

// 查看所有 cookie
print_r($_COOKIE);
```

```
?>
```

在下面的实例中，我们使用 isset() 函数来确认是否已设置了 cookie：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<?php
if (isset($_COOKIE["user"]))
    echo "欢迎 " . $_COOKIE["user"] . "!<br>";
else
    echo "普通访客!<br>";
?>

</body>
</html>
```

如何删除 Cookie?

当删除 cookie 时，您应当使过期日期变更为过去的时间点。

删除的实例：

```
<?php
// 设置 cookie 过期时间为过去 1 小时
setcookie("user", "", time()-3600);
?>
```

如果浏览器不支持 Cookie 该怎么办？

如果您的应用程序需要与不支持 cookie 的浏览器打交道，那么您不得不使用其他的办法在您的应用程序中的页面之间传递信息。一种方式是通过表单传递数据（有关表单和用户输入的内容，在本教程的前面章节中我们已经介绍过了）。

下面的表单在用户单点击 "Submit" 按钮时，向 "welcome.php" 提交了用户输入：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<form action="welcome.php" method="post">
名字: <input type="text" name="name">
年龄: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

取回 "welcome.php" 文件中的值，如下所示：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

欢迎 <?php echo $_POST["name"]; ?>.<br>
你 <?php echo $_POST["age"]; ?> 岁了。

</body>
</html>
```

点我分享笔记

反馈/建议

PHP Session

PHP session 变量用于存储关于用户会话（session）的信息，或者更改用户会话（session）的设置。Session 变量存储单一用户的信息，并且对于应用程序中的所有页面都是可用的。

PHP Session 变量

您在计算机上操作某个应用程序时，您打开它，做些更改，然后关闭它。这很像一次对话（Session）。计算机知道您是谁。它清楚您在何时打开和关闭应用程序。然而，在因特网上问题出现了：由于 HTTP 地址无法保持状态，Web 服务器并不知道您是谁以及您做了什么。

PHP session 解决了这个问题，它通过在服务器上存储用户信息以便随后使用（比如用户名称、购买商品等）。然而，会话信息是临时的，在用户离开网站后将被删除。如果您需要永久存储信息，可以把数据存储在数据库中。

Session 的工作机制是：为每个访客创建一个唯一的 id (UID)，并基于这个 UID 来存储变量。UID 存储在 cookie 中，或者通过 URL 进行传导。

开始 PHP Session

在您把用户信息存储到 PHP session 中之前，首先必须启动会话。

注释：session_start() 函数必须位于 <html> 标签之前：

```
<?php session_start(); ?>



<html>

<body>



</body>

</html>
```

上面的代码会向服务器注册用户的会话，以便您可以开始保存用户信息，同时会为用户会话分配一个 UID。

存储 Session 变量

存储和取回 session 变量的正确方法是使用 PHP $_SESSION 变量：

```
<?php
session_start();
// 存储 session 数据
$_SESSION['views']=1;
?>

<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<?php
// 检索 session 数据
echo "浏览量：". $_SESSION['views'];
?>

</body>
</html>
```

输出：

```
浏览量：1
```

在下面的实例中，我们创建了一个简单的 page-view 计数器。isset() 函数检测是否已设置 "views" 变量。如果已设置 "views" 变量，我们累加计数器。如果 "views" 不存在，则创建 "views" 变量，并把它设置为 1：

```
<?php
session_start();

if(isset($_SESSION['views']))
{
```

```
    $_SESSION['views']=$_SESSION['views']+1;
}
else
{
    $_SESSION['views']=1;
}
echo "浏览量：". $_SESSION['views'];
?>
```

销毁 Session

如果您希望删除某些 session 数据，可以使用 unset() 或 session_destroy() 函数。

unset() 函数用于释放指定的 session 变量：

```
<?php
session_start();
if(isset($_SESSION['views']))
{
    unset($_SESSION['views']);
}
?>
```

您也可以通过调用 session_destroy() 函数彻底销毁 session：

```
<?php
session_destroy();
?>
```

注释：session_destroy() 将重置 session，您将失去所有已存储的 session 数据。

PHP Cookie　　PHP 邮件

PHP 发送电子邮件

PHP 允许您从脚本直接发送电子邮件。

PHP mail() 函数

PHP mail() 函数用于从脚本中发送电子邮件。

语法

```
mail(to,subject,message,headers,parameters)
```

参数	描述
to	必需。规定 email 接收者。
subject	必需。规定 email 的主题。**注释：**该参数不能包含任何新行字符。
message	必需。定义要发送的消息。应使用 LF (\n) 来分隔各行。每行应该限制在 70 个字符内。
headers	可选。规定附加的标题，比如 From、Cc 和 Bcc。应当使用 CRLF (\r\n) 分隔附加的标题。
parameters	可选。对邮件发送程序规定额外的参数。

注释：PHP 运行邮件函数需要一个已安装且正在运行的邮件系统(如：sendmail、postfix、qmail等)。所用的程序通过在 php.ini 文件中的配置设置进行定义。请在我们的 PHP Mail 参考手册 阅读更多内容。

PHP 简易 E-Mail

通过 PHP 发送电子邮件的最简单的方式是发送一封文本 email。

在下面的实例中，我们首先声明变量($to, $subject, $message, $from, $headers)，然后我们在 mail() 函数中使用这些变量来发送了一封 E-mail：

```
<?php

$to = "someone@example.com";          // 邮件接收者

$subject = "参数邮件";                 // 邮件标题

$message = "Hello! 这是邮件的内容。";  // 邮件正文

$from = "someonelse@example.com";    // 邮件发送者

$headers = "From:" . $from;          // 头部信息设置

mail($to,$subject,$message,$headers);

echo "邮件已发送";

?>
```

PHP Mail 表单

通过 PHP，您能够在自己的站点制作一个反馈表单。下面的实例向指定的 e-mail 地址发送了一条文本消息：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<?php
if (isset($_REQUEST['email'])) { // 如果接收到邮箱参数则发送邮件
    // 发送邮件
    $email = $_REQUEST['email'] ;
    $subject = $_REQUEST['subject'] ;
    $message = $_REQUEST['message'] ;
    mail("someone@example.com", $subject,
    $message, "From:" . $email);
    echo "邮件发送成功";
} else { // 如果没有邮箱参数则显示表单
    echo "<form method='post' action='mailform.php'>
    Email: <input name='email' type='text'><br>
    Subject: <input name='subject' type='text'><br>
    Message:<br>
    <textarea name='message' rows='15' cols='40'>
    </textarea><br>
    <input type='submit'>
    </form>";
}
?>

</body>
</html>
```

实例解释：

首先，检查是否填写了邮件输入框

如果未填写（比如在页面被首次访问时），输出 HTML 表单

如果已填写（在表单被填写后），从表单发送电子邮件

当填写完表单点击提交按钮后，页面重新载入，可以看到邮件输入被重置，同时显示邮件发送成功的消息

注释：这个简易发送 e-mail 不安全，在本教程的下一章中，您将阅读到更多关于电子邮件脚本中的安全隐患，我们将为您讲解如何验证用户输入使它更安全。

PHP Mail 参考手册

如需查看更多关于 PHP mail() 函数的信息，请访问我们的 PHP Mail 参考手册。

PHP Session　　PHP 安全 E-mail

PHP 邮件　　PHP 错误处理

PHP Secure E-mails

在上一节中的 PHP e-mail 脚本中，存在着一个漏洞。

PHP E-mail 注入

首先，请看上一章中的 PHP 代码：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>

<?php
```

```
if (isset($_REQUEST['email'])) { // 如果接收到邮箱参数则发送邮件

    // 发送邮件

    $email = $_REQUEST['email'] ;

    $subject = $_REQUEST['subject'] ;

    $message = $_REQUEST['message'] ;

    mail("someone@example.com", $subject,

    $message, "From:" . $email);

    echo "邮件发送成功";

} else { // 如果没有邮箱参数则显示表单

    echo "<form method='post' action='mailform.php'>

    Email: <input name='email' type='text'><br>

    Subject: <input name='subject' type='text'><br>

    Message:<br>

    <textarea name='message' rows='15' cols='40'>

    </textarea><br>

    <input type='submit'>

    </form>";

}

?>



</body>

</html>
```

以上代码存在的问题是，未经授权的用户可通过输入表单在邮件头部插入数据。
假如用户在表单中的输入框内加入如下文本到电子邮件中，会出现什么情况呢？

```
someone@example.com%0ACc:person2@example.com

%0ABcc:person3@example.com,person3@example.com,

anotherperson4@example.com,person5@example.com

%0ABTo:person6@example.com
```

与往常一样，mail() 函数把上面的文本放入邮件头部，那么现在头部有了额外的 Cc:、Bcc: 和 To: 字段。当用户点击提交按钮时，这封 e-mail 会被发送到上面所有的地址！

PHP 防止 E-mail 注入

防止 e-mail 注入的最好方法是对输入进行验证。

下面的代码与上一章中的类似，不过这里我们已经增加了检测表单中 email 字段的输入验证程序：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
<?php
function spamcheck($field)
{
    // filter_var() 过滤 e-mail
    // 使用 FILTER_SANITIZE_EMAIL
    $field=filter_var($field, FILTER_SANITIZE_EMAIL);

    //filter_var() 过滤 e-mail
    // 使用 FILTER_VALIDATE_EMAIL
    if(filter_var($field, FILTER_VALIDATE_EMAIL))
    {
        return TRUE;
    }
    else
    {
        return FALSE;
    }
}

if (isset($_REQUEST['email']))
{
    // 如果接收到邮箱参数则发送邮件

    // 判断邮箱是否合法
```

```
    $mailcheck = spamcheck($_REQUEST['email']);
    if ($mailcheck==FALSE)
    {
        echo "非法输入";
    }
    else
    {
        // 发送邮件
        $email = $_REQUEST['email'] ;
        $subject = $_REQUEST['subject'] ;
        $message = $_REQUEST['message'] ;
        mail("someone@example.com", "Subject: $subject",
        $message, "From: $email" );
        echo "Thank you for using our mail form";
    }
}
else
{
    // 如果没有邮箱参数则显示表单
    echo "<form method='post' action='mailform.php'>
    Email: <input name='email' type='text'><br>
    Subject: <input name='subject' type='text'><br>
    Message:<br>
    <textarea name='message' rows='15' cols='40'>
    </textarea><br>
    <input type='submit'>
    </form>";
}
?>

</body>
</html>
```

在上面的代码中，我们使用了 PHP 过滤器来对输入进行验证：

FILTER_SANITIZE_EMAIL 过滤器从字符串中删除电子邮件的非法字符

FILTER_VALIDATE_EMAIL 过滤器验证电子邮件地址的值

您可以在我们的 PHP Filter 中阅读更多关于过滤器的知识。

□ PHP 邮件 PHP 错误处理 □

□ PHP 安全 E-mail PHP 异常处理 □

PHP 错误处理

在 PHP 中，默认的错误处理很简单。一条错误消息会被发送到浏览器，这条消息带有文件名、行号以及描述错误的消息。

PHP 错误处理

在创建脚本和 Web 应用程序时，错误处理是一个重要的部分。如果您的代码缺少错误检测编码，那么程序看上去很不专业，也为安全风险敞开了大门。

本教程介绍了 PHP 中一些最为重要的错误检测方法。

我们将为您讲解不同的错误处理方法：

简单的 "die()" 语句

自定义错误和错误触发器

错误报告

基本的错误处理：使用 die() 函数

第一个实例展示了一个打开文本文件的简单脚本：

```
<?php

$file=fopen("welcome.txt","r");

?>
```

如果文件不存在，您会得到类似这样的错误：

```
Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
```

```
No such file or directory in /www/runoob/test/test.php on line 2
```

为了避免用户得到类似上面的错误消息，我们在访问文件之前检测该文件是否存在：

```
<?php
if(!file_exists("welcome.txt"))
{
    die("文件不存在");
}
else
{
    $file=fopen("welcome.txt","r");
}
?>
```

现在，如果文件不存在，您会得到类似这样的错误消息：

```
文件不存在
```

相比之前的代码，上面的代码更有效，这是由于它采用了一个简单的错误处理机制在错误之后终止了脚本。

然而，简单地终止脚本并不总是恰当的方式。让我们研究一下用于处理错误的备选的 PHP 函数。

创建自定义错误处理器

创建一个自定义的错误处理器非常简单。我们很简单地创建了一个专用函数，可以在 PHP 中发生错误时调用该函数。

该函数必须有能力处理至少两个参数 (error level 和 error message)，但是可以接受最多五个参数（可选的：file, line-number 和 error context）：

语法

```
error_function(error_level,error_message,
error_file,error_line,error_context)
```

参数	描述
error_level	必需。为用户定义的错误规定错误报告级别。必须是一个数字。参见下面的表格：错误报告级别。
error_message	必需。为用户定义的错误规定错误消息。
error_file	可选。规定错误发生的文件名。
error_line	可选。规定错误发生的行号。
error_context	可选。规定一个数组，包含了当错误发生时在用的每个变量以及它们的值。

错误报告级别

这些错误报告级别是用户自定义的错误处理程序处理的不同类型的错误：

值	常量	描述
2	E_WARNING	非致命的 run-time 错误。不暂停脚本执行。
8	E_NOTICE	run-time 通知。在脚本发现可能有错误时发生，但也可能在脚本正常运行时发生。
256	E_USER_ERROR	致命的用户生成的错误。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_ERROR。
512	E_USER_WARNING	非致命的用户生成的警告。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_WARNING。
1024	E_USER_NOTICE	用户生成的通知。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_NOTICE。
4096	E_RECOVERABLE_ERROR	可捕获的致命错误。类似 E_ERROR，但可被用户定义的处理程序捕获。（参见 set_error_handler()）
8191	E_ALL	所有错误和警告。（在 PHP 5.4 中，E_STRICT 成为 E_ALL 的一部分）

现在，让我们创建一个处理错误的函数：

```
function customError($errno, $errstr)

{

    echo "<b>Error:</b> [$errno] $errstr<br>";

    echo "脚本结束";

    die();

}
```

上面的代码是一个简单的错误处理函数。当它被触发时，它会取得错误级别和错误消息。然后它会输出错误级别和消息，并终止脚本。
现在，我们已经创建了一个错误处理函数，我们需要确定在何时触发该函数。

设置错误处理程序

PHP 的默认错误处理程序是内建的错误处理程序。我们打算把上面的函数改造为脚本运行期间的默认错误处理程序。
可以修改错误处理程序，使其仅应用到某些错误，这样脚本就能以不同的方式来处理不同的错误。然而，在本例中，我们打算针对所有错误来使用我们自定义的错误处理程序：

```
set_error_handler("customError");
```

由于我们希望我们的自定义函数能处理所有错误，set_error_handler() 仅需要一个参数，可以添加第二个参数来规定错误级别。

实例

通过尝试输出不存在的变量，来测试这个错误处理程序：

```
<?php

// 错误处理函数

function customError($errno, $errstr)

{
```

```
    echo "<b>Error:</b> [$errno] $errstr";
}

// 设置错误处理函数
set_error_handler("customError");

// 触发错误
echo($test);
?>
```

以上代码的输出如下所示：

```
Error: [8] Undefined variable: test
```

触发错误

在脚本中用户输入数据的位置，当用户的输入无效时触发错误是很有用的。在 PHP 中，这个任务由 trigger_error() 函数完成。

实例

在本例中，如果 "test" 变量大于 "1"，就会发生错误：

```
<?php
$test=2;
if ($test>1)
{
    trigger_error("变量值必须小于等于 1");
}
?>
```

以上代码的输出如下所示：

```
Notice: 变量值必须小于等于 1
in /www/test/runoob.php on line 5
```

您可以在脚本中任何位置触发错误，通过添加的第二个参数，您能够规定所触发的错误级别。

可能的错误类型：

E_USER_ERROR - 致命的用户生成的 run-time 错误。错误无法恢复。脚本执行被中断。

E_USER_WARNING - 非致命的用户生成的 run-time 警告。脚本执行不被中断。

E_USER_NOTICE - 默认。用户生成的 run-time 通知。在脚本发现可能有错误时发生，但也可能在脚本正常运行时发生。

实例

在本例中，如果 "test" 变量大于 "1"，则发生 E_USER_WARNING 错误。如果发生了 E_USER_WARNING，我们将使用我们自定义的错误处理程序并结束脚本：

```
<?php

// 错误处理函数

function customError($errno, $errstr)

{

    echo "<b>Error:</b> [$errno] $errstr<br>";

    echo "脚本结束";

    die();

}



// 设置错误处理函数

set_error_handler("customError",E_USER_WARNING);



// 触发错误

$test=2;

if ($test>1)

{

    trigger_error("变量值必须小于等于 1",E_USER_WARNING);

}

?>
```

以上代码的输出如下所示：

```
Error: [512] 变量值必须小于等于 1

脚本结束
```

现在，我们已经学习了如何创建自己的 error，以及如何触发它们，接下来我们研究一下错误记录。

错误记录

在默认的情况下，根据在 php.ini 中的 error_log 配置，PHP 向服务器的记录系统或文件发送错误记录。通过使用 error_log() 函数，您可以向指定的文

件或远程目的地发送错误记录。

通过电子邮件向您自己发送错误消息，是一种获得指定错误的通知的好办法。

通过 E-Mail 发送错误消息

在下面的例子中，如果特定的错误发生，我们将发送带有错误消息的电子邮件，并结束脚本：

```
<?php

// 错误处理函数

function customError($errno, $errstr)

{

    echo "<b>Error:</b> [$errno] $errstr<br>";

    echo "已通知网站管理员";

    error_log("Error: [$errno] $errstr",1,

    "someone@example.com","From: webmaster@example.com");

}



// 设置错误处理函数

set_error_handler("customError",E_USER_WARNING);



// 触发错误

$test=2;

if ($test>1)

{

    trigger_error("变量值必须小于等于 1",E_USER_WARNING);

}

?>
```

以上代码的输出如下所示：

```
Error: [512] 变量值必须小于等于 1

已通知网站管理员
```

接收自以上代码的邮件如下所示：

```
Error: [512] 变量值必须小于等于 1
```

这个方法不适合所有的错误。常规错误应当通过使用默认的 PHP 记录系统在服务器上进行记录。

PHP 安全 E-mail　　PHP 异常处理

PHP 异常处理

异常用于在指定的错误发生时改变脚本的正常流程。

异常是什么

PHP 5 提供了一种新的面向对象的错误处理方法。

异常处理用于在指定的错误（异常）情况发生时改变脚本的正常流程。这种情况称为异常。

当异常被触发时，通常会发生：

- 当前代码状态被保存
- 代码执行被切换到预定义（自定义）的异常处理器函数
- 根据情况，处理器也许会从保存的代码状态重新开始执行代码，终止脚本执行，或从代码中另外的位置继续执行脚本

我们将展示不同的错误处理方法：

- 异常的基本使用
- 创建自定义的异常处理器
- 多个异常
- 重新抛出异常
- 设置顶层异常处理器

注释：异常应该仅仅在错误情况下使用，而不应该用于在一个指定的点跳转到代码的另一个位置。

异常的基本使用

当异常被抛出时，其后的代码不会继续执行，PHP 会尝试查找匹配的 "catch" 代码块。

如果异常没有被捕获，而且又没用使用 set_exception_handler() 作相应的处理的话，那么将发生一个严重的错误（致命错误），并且输出 "Uncaught Exception" （未捕获异常）的错误消息。

让我们尝试抛出一个异常，同时不去捕获它：

```
<?php
// 创建一个有异常处理的函数
function checkNum($number)
{
if($number>1)
{
```

```
throw new Exception("Value must be 1 or below");
}
return true;
}
// 触发异常
checkNum(2);
?>
```

上面的代码会得到类似这样的一个错误：

```
Fatal error: Uncaught exception 'Exception' with message 'Value must be 1 or below' in /www/runoob/test/test.php:7 St
ack trace: #0 /www/runoob/test/test.php(13): checkNum(2) #1 {main} thrown in /www/runoob/test/test.php on line 7
```

Try、throw 和 catch

要避免上面实例中出现的错误，我们需要创建适当的代码来处理异常。

适当的处理异常代码应该包括：

1. Try - 使用异常的函数应该位于 "try" 代码块内。如果没有触发异常，则代码将照常继续执行。但是如果异常被触发，会抛出一个异常。
2. Throw - 里规定如何触发异常。每一个 "throw" 必须对应至少一个 "catch"。
3. Catch - "catch" 代码块会捕获异常，并创建一个包含异常信息的对象。

让我们触发一个异常：

```
<?php
// 创建一个有异常处理的函数
function checkNum($number)
{
if($number>1)
{
throw new Exception("变量值必须小于等于 1");
}
return true;
}
// 在 try 块 触发异常
try
{
checkNum(2);
// 如果抛出异常，以下文本不会输出
echo '如果输出该内容，说明 $number 变量';
}
// 捕获异常
catch(Exception $e)
{
echo 'Message: ' .$e->getMessage();
}
?>
```

上面代码将得到类似这样一个错误：

```
Message: 变量值必须小于等于 1
```

实例解释：

上面的代码抛出了一个异常，并捕获了它：

1. 创建 checkNum() 函数。它检测数字是否大于 1。如果是，则抛出一个异常。
2. 在 "try" 代码块中调用 checkNum() 函数。
3. checkNum() 函数中的异常被抛出。
4. "catch" 代码块接收到该异常，并创建一个包含异常信息的对象 ($e)。

5. 通过从这个 exception 对象调用 $e->getMessage()，输出来自该异常的错误消息。

然而，为了遵循 "每个 throw 必须对应一个 catch" 的原则，可以设置一个顶层的异常处理器来处理漏掉的错误。

创建一个自定义的 Exception 类

创建自定义的异常处理程序非常简单。我们简单地创建了一个专门的类，当 PHP 中发生异常时，可调用其函数。该类必须是 exception 类的一个扩展。

这个自定义的 customException 类继承了 PHP 的 exception 类的所有属性，您可向其添加自定义的函数。

我们开始创建 customException 类：

```php
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = '错误行号 '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> 不是一个合法的 E-Mail 地址';
return $errorMsg;
}
}
$email = "someone@example...com";
try
{
// 检测邮箱
if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new customException($email);
}
}
catch (customException $e)
{
//display custom message
echo $e->errorMessage();
}
?>
```

这个新的类是旧的 exception 类的副本，外加 errorMessage() 函数。正因为它是旧类的副本，因此它从旧类继承了属性和方法，我们可以使用 exception 类的方法，比如 getLine()、getFile() 和 getMessage()。

实例解释：

上面的代码抛出了一个异常，并通过一个自定义的 exception 类来捕获它：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一条错误消息。
3. 把 $email 变量设置为不合法的 e-mail 地址字符串。
4. 执行 "try" 代码块，由于 e-mail 地址不合法，因此抛出一个异常。
5. "catch" 代码块捕获异常，并显示错误消息。

多个异常

可以为一段脚本使用多个异常，来检测多种情况。

可以使用多个 if..else 代码块，或一个 switch 代码块，或者嵌套多个异常。这些异常能够使用不同的 exception 类，并返回不同的错误消息：

```php
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = '错误行号 '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> 不是一个合法的 E-Mail 地址';
return $errorMsg;
}
```

```
}
$email = "someone@example.com";
try
{
// 检测邮箱
if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new customException($email);
}
// 检测 "example" 是否在邮箱地址中
if(strpos($email, "example") !== FALSE)
{
throw new Exception("$email 是 example 邮箱");
}
}
catch (customException $e)
{
echo $e->errorMessage();
}
catch(Exception $e)
{
echo $e->getMessage();
}
?>
```

实例解释：

上面的代码测试了两种条件，如果其中任何一个条件不成立，则抛出一个异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 把 $email 变量设置为一个字符串，该字符串是一个有效的 e-mail 地址，但包含字符串 "example"。
4. 执行 "try" 代码块，在第一个条件下，不会抛出异常。
5. 由于 e-mail 含有字符串 "example"，第二个条件会触发异常。
6. "catch" 代码块会捕获异常，并显示恰当的错误消息。

如果 customException 类抛出了异常，但没有捕获 customException，仅仅捕获了 base exception，则在那里处理异常。

重新抛出异常

有时，当异常被抛出时，您也许希望以不同于标准的方式对它进行处理。可以在一个 "catch" 代码块中再次抛出异常。

脚本应该对用户隐藏系统错误。对程序员来说，系统错误也许很重要，但是用户对它们并不感兴趣。为了让用户更容易使用，您可以再次抛出带有对用户比较友好的消息的异常：

```
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = $this->getMessage().' 不是一个合法的 E-Mail 地址。';
return $errorMsg;
}
}
$email = "someone@example.com";
try
{
try
{
// 检测 "example" 是否在邮箱地址中
if(strpos($email, "example") !== FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new Exception($email);
}
}
```

```
catch(Exception $e)
{
// 重新抛出异常
throw new customException($email);
}
}
catch (customException $e)
{
// 显示自定义信息
echo $e->errorMessage();
}
?>
```

实例解释：

上面的代码检测在邮件地址中是否含有字符串 "example"。如果有，则再次抛出异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 把 $email 变量设置为一个字符串，该字符串是一个有效的 e-mail 地址，但包含字符串 "example"。
4. "try" 代码块包含另一个 "try" 代码块，这样就可以再次抛出异常。
5. 由于 e-mail 包含字符串 "example"，因此触发异常。
6. "catch" 代码块捕获到该异常，并重新抛出 "customException"。
7. 捕获到 "customException"，并显示一条错误消息。

如果在当前的 "try" 代码块中异常没有被捕获，则它将在更高层级上查找 catch 代码块。

设置顶层异常处理器

set_exception_handler() 函数可设置处理所有未捕获异常的用户定义函数。

```
<?php
function myException($exception)
{
echo "<b>Exception:</b> " , $exception->getMessage();
}
set_exception_handler('myException');
throw new Exception('Uncaught Exception occurred');
?>
```

以上代码的输出如下所示：

```
Exception: Uncaught Exception occurred
```

在上面的代码中，不存在 "catch" 代码块，而是触发顶层的异常处理程序。应该使用此函数来捕获所有未被捕获的异常。

异常的规则

需要进行异常处理的代码应该放入 try 代码块内，以便捕获潜在的异常。

每个 try 或 throw 代码块必须至少拥有一个对应的 catch 代码块。

使用多个 catch 代码块可以捕获不同种类的异常。

可以在 try 代码块内的 catch 代码块中抛出（再次抛出）异常。

简而言之：如果抛出了异常，就必须捕获它。

PHP 过滤器

PHP 过滤器用于验证和过滤来自非安全来源的数据，比如用户的输入。

什么是 PHP 过滤器？

PHP 过滤器用于验证和过滤来自非安全来源的数据。

测试、验证和过滤用户输入或自定义数据是任何 Web 应用程序的重要组成部分。

PHP 的过滤器扩展的设计目的是使数据过滤更轻松快捷。

为什么使用过滤器？

几乎所有的 Web 应用程序都依赖外部的输入。这些数据通常来自用户或其他应用程序（比如 web 服务）。通过使用过滤器，您能够确保应用程序获得正确的输入类型。

您应该始终对外部数据进行过滤！

输入过滤是最重要的应用程序安全课题之一。

什么是外部数据？

- 来自表单的输入数据
- Cookies
- Web services data
- 服务器变量
- 数据库查询结果

函数和过滤器

如需过滤变量，请使用下面的过滤器函数之一：

- filter_var() - 通过一个指定的过滤器来过滤单一的变量
- filter_var_array() - 通过相同的或不同的过滤器来过滤多个变量
- filter_input - 获取一个输入变量，并对它进行过滤
- filter_input_array - 获取多个输入变量，并通过相同的或不同的过滤器对它们进行过滤

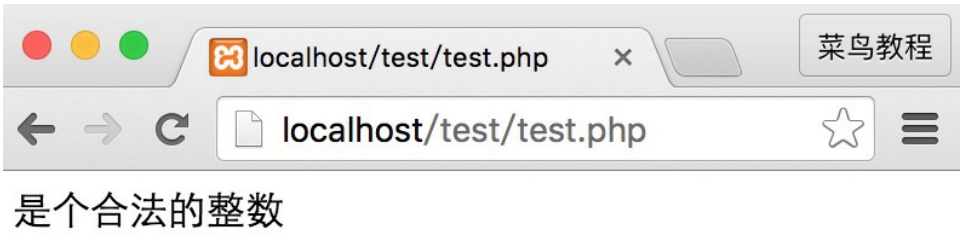
在下面的实例中，我们用 filter_var() 函数验证了一个整数：

实例

```php
<?php
$int = 123;
if(!filter_var($int, FILTER_VALIDATE_INT))
{
echo("不是一个合法的整数");
}
```

```
else
{
echo("是个合法的整数");
}
?>
```

上面的代码使用了 "FILTER_VALIDATE_INT" 过滤器来过滤变量。由于这个整数是合法的，因此上面的代码将输出：



如果我们尝试使用一个非整数的变量（比如 "123abc"），则将输出："Integer is not valid"。

如需查看完整的函数和过滤器列表，请访问我们的 PHP Filter 参考手册。

Validating 和 Sanitizing

有两种过滤器：

Validating 过滤器：

- 用于验证用户输入
- 严格的格式规则（比如 URL 或 E-Mail 验证）
- 如果成功则返回预期的类型，如果失败则返回 FALSE

Sanitizing 过滤器：

- 用于允许或禁止字符串中指定的字符
- 无数据格式规则
- 始终返回字符串

选项和标志

选项和标志用于向指定的过滤器添加额外的过滤选项。

不同的过滤器有不同的选项和标志。

在下面的实例中，我们用 filter_var() 和 "min_range" 以及 "max_range" 选项验证了一个整数：

实例

```
<?php
$var=300;
$int_options = array(
"options"=>array
(
"min_range"=>0,
"max_range"=>256
)
);
if(!filter_var($var, FILTER_VALIDATE_INT, $int_options))
{
echo("不是一个合法的整数");
}
else
{
echo("是个合法的整数");
}
?>
```

就像上面的代码一样，选项必须放入一个名为 "options" 的相关数组中。如果使用标志，则不需在数组内。

由于整数是 "300"，它不在指定的范围内，以上代码的输出将是：

```
不是一个合法的整数
```

如需查看完整的函数和过滤器列表，请访问我们的 PHP Filter 参考手册。您可以看到每个过滤器的可用选项和标志。

验证输入

让我们试着验证来自表单的输入。

我们需要做的第一件事情是确认是否存在我们正在查找的输入数据。

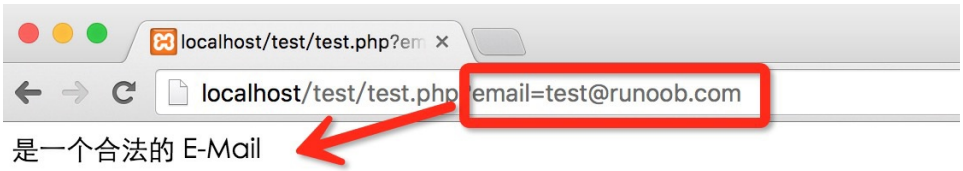
然后我们用 filter_input() 函数过滤输入的数据。

在下面的实例中，输入变量 "email" 被传到 PHP 页面：

实例

```php
<?php
if(!filter_has_var(INPUT_GET, "email"))
{
echo("没有 email 参数");
}
else
{
if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL))
{
echo "不是一个合法的 E-Mail";
}
else
{
echo "是一个合法的 E-Mail";
}
}
?>
```

以上实例测试结果如下：



实例解释

上面的实例有一个通过 "GET" 方法传送的输入变量 (email)：

1. 检测是否存在 "GET" 类型的 "email" 输入变量
2. 如果存在输入变量，检测它是否是有效的 e-mail 地址

净化输入

让我们试着清理一下从表单传来的 URL。

首先，我们要确认是否存在我们正在查找的输入数据。

然后，我们用 filter_input() 函数来净化输入数据。

在下面的实例中，输入变量 "url" 被传到 PHP 页面：

```
<?php
```

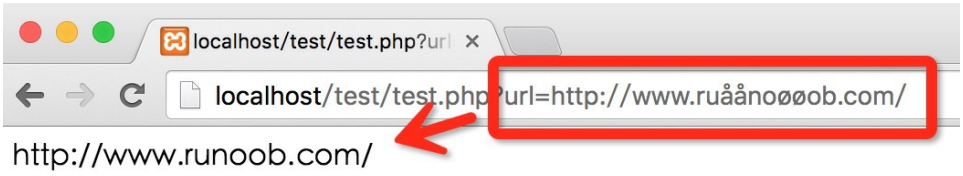
```
if(!filter_has_var(INPUT_GET, "url"))
{
    echo("没有 url 参数");
}
else
{
    $url = filter_input(INPUT_GET,
    "url", FILTER_SANITIZE_URL);
    echo $url;
}
?>
```

实例解释

上面的实例有一个通过 "GET" 方法传送的输入变量 (url):

1. 检测是否存在 "GET" 类型的 "url" 输入变量
2. 如果存在此输入变量，对其进行净化（删除非法字符），并将其存储在 $url 变量中

假如输入变量是一个类似这样的字符串："http://www.ruåånoøøob.com/"，则净化后的 $url 变量如下所示：



过滤多个输入

表单通常由多个输入字段组成。为了避免对 filter_var 或 filter_input 函数重复调用，我们可以使用 filter_var_array 或 the filter_input_array 函数。

在本例中，我们使用 filter_input_array() 函数来过滤三个 GET 变量。接收到的 GET 变量是一个名字、一个年龄以及一个 e-mail 地址：

实例

```
<?php
$filters = array
(
"name" => array
(
"filter"=>FILTER_SANITIZE_STRING
),
"age" => array
(
"filter"=>FILTER_VALIDATE_INT,
"options"=>array
(
"min_range"=>1,
"max_range"=>120
)
),
"email"=> FILTER_VALIDATE_EMAIL
);
```

```php
$result = filter_input_array(INPUT_GET, $filters);
if (!$result["age"])
{
echo("年龄必须在 1 到 120 之间。<br>");
}
elseif(!$result["email"])
{
echo("E-Mail 不合法<br>");
}
else
{
echo("输入正确");
}
?>
```

实例解释

上面的实例有三个通过 "GET" 方法传送的输入变量 (name、age 和 email)：

1. 设置一个数组，其中包含了输入变量的名称和用于指定的输入变量的过滤器
2. 调用 filter_input_array() 函数，参数包括 GET 输入变量及刚才设置的数组
3. 检测 $result 变量中的 "age" 和 "email" 变量是否有非法的输入。（如果存在非法输入，在使用 filter_input_array() 函数之后，输入变量为 FALSE。）

filter_input_array() 函数的第二个参数可以是数组或单一过滤器的 ID。

如果该参数是单一过滤器的 ID，那么这个指定的过滤器会过滤输入数组中所有的值。

如果该参数是一个数组，那么此数组必须遵循下面的规则：

必须是一个关联数组，其中包含的输入变量是数组的键（比如 "age" 输入变量）

此数组的值必须是过滤器的 ID，或者是规定了过滤器、标志和选项的数组

使用 Filter Callback

通过使用 FILTER_CALLBACK 过滤器，可以调用自定义的函数，把它作为一个过滤器来使用。这样，我们就拥有了数据过滤的完全控制权。

您可以创建自己的自定义函数，也可以使用已存在的 PHP 函数。

将您准备用到的过滤器的函数，按指定选项的规定方法进行规定。在关联数组中，带有名称 "options"。

在下面的实例中，我们使用了一个自定义的函数把所有 "_" 转换为 "."：

实例

```php
<?php
function convertSpace($string)
{
return str_replace("_", ".", $string);
}
$string = "www_runoob_com!";
echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
?>
```

上面代码的结果如下所示：

```
www.runoob.com!
```

实例解释

上面的实例把所有 "_" 转换成 "." ：

1. 创建一个把 "_" 替换为 "." 的函数
2. 调用 filter_var() 函数，它的参数是 FILTER_CALLBACK 过滤器以及包含我们的函数的数组

PHP 高级过滤器

检测一个数字是否在一个范围内

以下实例使用了 filter_var() 函数来检测一个 INT 型的变量是否在 1 到 200 内:

实例

```
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int, FILTER_VALIDATE_INT, array("options" => array("min_range"=>$min, "max_range"=>$max))) === fal
se) {
    echo("变量值不在合法范围内");
} else {
    echo("变量值在合法范围内");
}
?>
```

尝试一下 »

检测 IPv6 地址

以下实例使用了 filter_var() 函数来检测一个 $ip 变量是否是 IPv6 地址:

实例

```
<?php
$ip = "2001:0db8:85a3:08d3:1319:8a2e:0370:7334";

if (!filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6) === false) {
    echo("$ip 是一个 IPv6 地址");
} else {
    echo("$ip 不是一个 IPv6 地址");
}
?>
```

尝试一下 »

检测 URL - 必须包含QUERY_STRING（查询字符串）

以下实例使用了 filter_var() 函数来检测 $url 是否包含查询字符串:

实例

```
<?php
$url = "http://www.runoob.com";

if (!filter_var($url, FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED) === false) {
    echo("$url 是一个合法的 URL");
} else {
    echo("$url 不是一个合法的 URL");
}
?>
```

尝试一下 »

移除 ASCII 值大于 127 的字符

以下实例使用了 filter_var() 函数来移除字符串中 ASCII 值大于 127 的字符，同样它也能移除 HTML 标签：

实例

```
<?php
$str = "<h1>Hello WorldÆØÅ!</h1>";

$newstr = filter_var($str, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);
echo $newstr;
?>
```

尝试一下 »

PHP 过滤器参考手册

你也可以通过访问本站的 PHP 过滤器参考手册 来查看过滤器的具体应用。

参考手册中包含了过滤器参数的简要说明和使用例子！

PHP imagecolortransparent – 将某个颜色定义为透明色 PHP RESTful

点我分享笔记

反馈/建议

PHP 5 常量 PHP MySQL 插入多条数据

PHP JSON

本章节我们将为大家介绍如何使用 PHP 语言来编码和解码 JSON 对象。

环境配置

在 php5.2.0 及以上版本已经内置 JSON 扩展。

JSON 函数

函数	描述
json_encode	对变量进行 JSON 编码
json_decode	对 JSON 格式的字符串进行解码，转换为 PHP 变量
json_last_error	返回最后发生的错误

json_encode

PHP json_encode() 用于对变量进行 JSON 编码，该函数如果执行成功返回 JSON 数据，否则返回 FALSE 。

语法

```
string json_encode ( $value [, $options = 0 ] )
```

参数

value: 要编码的值。该函数只对 UTF-8 编码的数据有效。

options:由以下常量组成的二进制掩码：JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK,JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT

实例

以下实例演示了如何将 PHP 数组转换为 JSON 格式数据：

```
<?php
   $arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
   echo json_encode($arr);
?>
```

以上代码执行结果为：

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

以下实例演示了如何将 PHP 对象转换为 JSON 格式数据：

```
<?php
   class Emp {
       public $name = "";
       public $hobbies  = "";
       public $birthdate = "";
   }
   $e = new Emp();
   $e->name = "sachin";
```

```
    $e->hobbies  = "sports";

    $e->birthdate = date('m/d/Y h:i:s a', "8/5/1974 12:20:03 p");

    $e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 12:20:03"));



    echo json_encode($e);

?>
```

以上代码执行结果为：

```
{"name":"sachin","hobbies":"sports","birthdate":"08\/05\/1974 12:20:03 pm"}
```

---

json_decode

PHP json_decode() 函数用于对 JSON 格式的字符串进行解码，并转换为 PHP 变量。

语法

```
mixed json_decode ($json_string [,$assoc = false [, $depth = 512 [, $options = 0 ]]])
```

参数

json_string: 待解码的 JSON 字符串，必须是 UTF-8 编码数据

assoc: 当该参数为 TRUE 时，将返回数组，FALSE 时返回对象。

depth: 整数类型的参数，它指定递归深度

options: 二进制掩码，目前只支持 JSON_BIGINT_AS_STRING 。

实例

以下实例演示了如何解码 JSON 数据：

```
<?php

    $json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';



    var_dump(json_decode($json));

    var_dump(json_decode($json, true));

?>
```

以上代码执行结果为：

```
object(stdClass)#1 (5) {

    ["a"] => int(1)
```

```
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}

array(5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}
```

PHP 5 常量 PHP MySQL 插入多条数据

点我分享笔记

反馈/建议

PHP 7 移除的 SAPI PHP 面向对象

PHP 7 新特性

PHP 7+ 版本极大地改进了性能，在一些WordPress基准测试当中，性能可以达到PHP 5.6的3倍。

PHP 7+ 版本新加特性如下表所示：

序号	内容
1	PHP 标量类型与返回值类型声明
2	PHP NULL 合并运算符
3	PHP 太空船运算符（组合比较符）

4	PHP 常量数组
5	PHP 匿名类
6	PHP Closure::call()
7	PHP 过滤 unserialize()
8	PHP IntlChar()
9	PHP CSPRNG
10	PHP 7 异常
11	PHP 7 use 语句
12	PHP 7 错误处理
13	PHP intdiv() 函数
14	PHP 7 Session 选项
15	PHP 7 废弃特性
16	PHP 7 移除的扩展
17	PHP 7 移除的 SAPI

PHP 7 移除的 SAPI
PHP 面向对象

点我分享笔记

反馈/建议

PHP 过滤器
PHP 连接 MySQL

PHP MySQL 简介

通过 PHP，您可以连接和操作数据库。

MySQL 是跟 PHP 配套使用的最流行的开源数据库系统。

如果想学习更多 MySQL 知识可以查看本站 MySQL 教程。

MySQL 是什么？

MySQL 是一种在 Web 上使用的数据库系统。

MySQL 是一种在服务器上运行的数据库系统。

MySQL 不管在小型还是大型应用程序中，都是理想的选择。

MySQL 是非常快速，可靠，且易于使用的。

MySQL 支持标准的 SQL。

MySQL 在一些平台上编译。

MySQL 是免费下载使用的。

MySQL 是由 Oracle 公司开发、发布和支持的。

MySQL 是以公司创始人 Monty Widenius's daughter: My 命名的。

MySQL 中的数据存储在表中。表格是一个相关数据的集合，它包含了列和行。
在分类存储信息时，数据库非常有用。一个公司的数据库可能拥有以下表：

Employees

Products

Customers

Orders

PHP + MySQL

PHP 与 MySQL 结合是跨平台的。（您可以在 Windows 上开发，在 Unix 平台上应用。）

查询

查询是一种询问或请求。
通过 MySQL，我们可以向数据库查询具体的信息，并得到返回的记录集。
请看下面的查询（使用标准 SQL）：

```
mysql> set names utf8;

mysql> SELECT name FROM websites;

+---------------+

| name          |

+---------------+

| Google        |

| 淘宝        |

| 菜鸟教程  |

| 微博        |

| Facebook      |

| stackoverflow |

+---------------+

6 rows in set (0.00 sec)
```

语句 **set names utf8;**用于设定数据库编码，让中文可以正常显示。
上面的查询选取了 "websites" 表中 "name" 列的所有数据。
如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

下载 MySQL 数据库

如果您的 PHP 服务器没有 MySQL 数据库，可以在此免费下载 MySQL：http://www.mysql.com。

关于 MySQL 数据库的事实

关于 MySQL 的一点很棒的特性是，可以对它进行缩减，来支持嵌入的数据库应用程序。也许正因为如此，许多人认为 MySQL 仅仅能处理中小型的系统。

事实上，对于那些支持巨大数据和访问量的网站（比如 Friendster、Yahoo、Google），MySQL 是事实上的标准数据库。

这个地址提供了使用 MySQL 的公司的概览：http://www.mysql.com/customers/。

PHP 过滤器 PHP 连接 MySQL

PHP MySQL 简介 PHP MySQL 创建数据库

PHP 连接 MySQL

PHP 5 及以上版本建议使用以下方式连接 MySQL :

- **MySQLi extension** ("i" 意为 improved)
- **PDO (PHP Data Objects)**

在 PHP 早期版本中我们使用 MySQL 扩展。但该扩展在 2012 年开始不建议使用。

我是该用 MySQLi，还是 PDO?

如果你需要一个简短的回答，即 "你习惯哪个就用哪个"。

MySQLi 和 PDO 有它们自己的优势：

PDO 应用在 12 种不同数据库中， MySQLi 只针对 MySQL 数据库。

所以，如果你的项目需要在多种数据库中切换，建议使用 PDO，这样你只需要修改连接字符串和部分查询语句即可。 使用 MySQLi, 如果不同数据库，你需要重新编写所有代码，包括查询。

两者都是面向对象, 但 MySQLi 还提供了 API 接口。

两者都支持预处理语句。 预处理语句可以防止 SQL 注入，对于 web 项目的安全性是非常重要的。

MySQLi 和 PDO 连接 MySQL 实例

在本章节及接下来的章节中，我们会使用以下三种方式来演示 PHP 操作 MySQL:

- MySQLi (面向对象)
- MySQLi (面向过程)
- PDO

MySQLi 安装

Linux 和 Windows: 在 php5 mysql 包安装时 MySQLi 扩展多数情况下是自动安装的。

安装详细信息，请查看： http://php.net/manual/en/mysqli.installation.php

可以通过 phpinfo() 查看是否安装成功：

mysqli

Mysqli Support	enabled
Client API library version	mysqlnd 5.0.11-dev - 20120503 - $Id: f373ea5dd5538761406a8022a4b8a374418b240e $
Active Persistent Links	0
Inactive Persistent Links	0
Active Links	0

Directive	Local Value	Master Value
mysqli.allow_local_infile	On	On
mysqli.allow_persistent	On	On

PDO 安装

For 安装详细信息，请查看： http://php.net/manual/en/pdo.installation.php

可以通过 phpinfo() 查看是否安装成功：

PDO

PDO support	enabled
PDO drivers	mysql, pgsql, sqlite

pdo_mysql

PDO Driver for MySQL	enabled
Client API version	mysqlnd 5.0.11-dev - 20120503 - $Id: f373ea5dd5538761406a8022a4b8a374418b240e $

连接 MySQL

在我们访问 MySQL 数据库前，我们需要先连接到数据库服务器：

实例 (MySQLi - 面向对象)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// 创建连接
$conn = new mysqli($servername, $username, $password);
// 检测连接
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
}
echo "连接成功";
?>
```

Note

注意在以上面向对象的实例中 $connect_error 是在 PHP 5.2.9 和 5.3.0 中添加的。如果你需要兼容更早版本 请使用以下代码替换:

```
// 检测连接
if (mysqli_connect_error()) {
    die("数据库连接失败: " . mysqli_connect_error());
}
```

实例 (MySQLi - 面向过程)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// 创建连接
$conn = mysqli_connect($servername, $username, $password);
// 检测连接
if (!$conn) {
die("Connection failed: " . mysqli_connect_error());
}
echo "连接成功";
```

```
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
try {
$conn = new PDO("mysql:host=$servername;", $username, $password);
echo "连接成功";
}
catch(PDOException $e)
{
echo $e->getMessage();
}
?>
```

Note 注意在以上 PDO 实例中我们已经指定了数据库 (myDB)。PDO 在连接过程需要设置数据库名。如果没有指定，则会抛出异常。

关闭连接

连接在脚本执行完后会自动关闭。你也可以使用以下代码来关闭连接：

实例 (MySQLi - 面向对象)

```
$conn->close();
```

实例 (MySQLi - 面向过程)

```
mysqli_close($conn);
```

实例 (PDO)

```
$conn = null;
```

□ PHP MySQL 简介　　PHP MySQL 创建数据库 □

我们可以用一下办法来测试PHP的MySQL数据库的连接。
使用函数 parse_ini_file() 解析配置文件 config.ini 来获得数据库连接参数，然后使用 new 关键字对 mysqli 类进行实例化，最后使用函数 mysqli_connect_errno() 来判断是否成功连接上了 MySQL 数据库，实现该过程的代码如下所示：

```
try{

    //解析config.ini文件

    $config = parse_ini_file(realpath(dirname(__FILE__) . '/config/config.ini'));

    //对mysqli类进行实例化

    $mysqli = new mysqli($config['host'], $config['username'], $config['password'], $config['dbname']);
```

```
    if(mysqli_connect_errno()){    //判断是否成功连接上MySQL数据库
        throw new Exception('数据库连接错误! ');  //如果连接错误，则抛出异常
    }else{
        echo '数据库连接成功! ';    //打印连接成功的提示
    }
}catch (Exception $e){        //捕获异常
    echo $e->getMessage();    //打印异常信息
}
```

kindyear10个月前 (12-15)

反馈/建议

PHP MySQL 创建数据库

数据库存有一个或多个表。

你需要 CREATE 权限来创建或删除 MySQL 数据库。

使用 MySQLi 和 PDO 创建 MySQL 数据库

CREATE DATABASE 语句用于在 MySQL 中创建数据库。

在下面的实例中，创建了一个名为 "myDB" 的数据库：

实例 (MySQLi - 面向对象)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// 创建连接
$conn = new mysqli($servername, $username, $password);
// 检测连接
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
}
// 创建数据库
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
echo "数据库创建成功";
} else {
echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

Note **注意：** 当你创建一个新的数据库时，你必须为 mysqli 对象指定三个参数 (servername, username 和 password)。

Tip: 如果你使用其他端口（默认为3306），为数据库参数添加空字符串，如: new mysqli("localhost", "username", "password", "", port)

实例 (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// 创建连接
$conn = mysqli_connect($servername, $username, $password);
// 检测连接
if (!$conn) {
die("连接失败: " . mysqli_connect_error());
}
// 创建数据库
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
echo "数据库创建成功";
} else {
echo "Error creating database: " . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

注意： 以下使用 PDO 实例创建数据库 **myDBPDO**：

实例

使用 PDO:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername", $username, $password);

    // 设置 PDO 错误模式为异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "CREATE DATABASE myDBPDO";

    // 使用 exec() ，因为没有结果返回
    $conn->exec($sql);

    echo "数据库创建成功<br>";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

提示： 使用 PDO 的最大好处是在数据库查询过程出现问题时可以使用异常类来 处理问题。如果 try{ } 代码块出现异常，脚本会停止执行并会跳到第一个 catch(){ } 代码块执行代码。 在以上捕获的代码块中我们输出了 SQL 语句并生成错误信息。

PHP 创建 MySQL 表

一个数据表有一个唯一名称，并有行和列组成。

使用 MySQLi 和 PDO 创建 MySQL 表

CREATE TABLE 语句用于创建 MySQL 表。

创建表前，我们需要使用 `use myDB` 来选择要操作的数据库：

```
use myDB;
```

我们将创建一个名为 "MyGuests" 的表，有 5 个列： "id", "firstname", "lastname", "email" 和 "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

上表中的注意事项：

数据类型指定列可以存储什么类型的数据。完整的数据类型请参考我们的 数据类型参考手册。

在设置了数据类型后，你可以为每个列指定其他选项的属性：

NOT NULL - 每一行都必须含有值（不能为空），null 值是不允许的。

DEFAULT value - 设置默认值

UNSIGNED - 使用无符号数值类型，0 及正数

AUTO INCREMENT - 设置 MySQL 字段的值在新增记录时每次自动增长 1

PRIMARY KEY - 设置数据表中每条记录的唯一标识。 通常列的 PRIMARY KEY 设置为 ID 数值，与 AUTO_INCREMENT 一起使用。

每个表都应该有一个主键(本列为 "id" 列)，主键必须包含唯一的值。

以下实例展示了如何在 PHP 中创建表：

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}
```

```
// 使用 sql 创建数据表
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "创建数据表错误: " . $conn->error;
}

$conn->close();
?>
```

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检测连接
if (!$conn) {
    die("连接失败: " . mysqli_connect_error());
}

// 使用 sql 创建数据表
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "数据表 MyGuests 创建成功";
} else {
    echo "创建数据表错误: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // 设置 PDO 错误模式，用于抛出异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // 使用 sql 创建数据表
    $sql = "CREATE TABLE MyGuests (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
```

```
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP
    )";

    // 使用 exec() ，没有结果返回
    $conn->exec($sql);
    echo "数据表 MyGuests 创建成功";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

□ PHP MySQL 插入多条数据　　PHP MySQL 预处理语句 □

□点我分享笔记

反馈/建议

PHP MySQL 插入数据

使用 MySQLi 和 PDO 向 MySQL 插入数据

在创建完数据库和表后，我们可以向表中添加数据。

以下为一些语法规则：

- PHP 中 SQL 查询语句必须使用引号
- 在 SQL 查询语句中的字符串值必须加引号
- 数值的值不需要引号
- NULL 值不需要引号

INSERT INTO 语句通常用于向 MySQL 表添加新的记录：

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

学习更多关于 SQL 知识，请查看我们的 SQL 教程。

在前面的几个章节中我们已经创建了表 "MyGuests"，表字段有: "id", "firstname", "lastname", "email" 和 "reg_date"。 现在，让我们开始向表填充数据。

> **Note** **注意：** 如果列设置 AUTO_INCREMENT (如 "id" 列) 或 TIMESTAMP (如 "reg_date" 列)，我们就不需要在 SQL 查询语句中指定值；MySQL 会自动为该列添加值。

以下实例向 "MyGuests" 表添加了新的记录:

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
    die("连接失败: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "新记录插入成功";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检测连接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "新记录插入成功";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // 设置 PDO 错误模式，用于抛出异常
```

```
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')";
    // 使用 exec() ，没有结果返回
    $conn->exec($sql);
    echo "新记录插入成功";
}
catch(PDOException $e)
{
    echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL 创建数据库 PHP MySQL 读取数据

PHP JSON PHP MySQL 创建表

PHP MySQL 插入多条数据

使用 MySQLi 和 PDO 向 MySQL 插入多条数据

mysqli_multi_query() 函数可用来执行多条SQL语句。

以下实例向 "MyGuests" 表添加了三条新的记录:

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建链接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检查链接
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";
if ($conn->multi_query($sql) === TRUE) {
echo "新记录插入成功";
} else {
```

```
echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

Note 请注意，每个SQL语句必须用分号隔开。

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建链接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检查链接
if (!$conn) {
die("连接失败: " . mysqli_connect_error());
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";
if (mysqli_multi_query($conn, $sql)) {
echo "新记录插入成功";
} else {
echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// 开始事务
$conn->beginTransaction();
// SQL 语句
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com')");
$conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')");
// 提交事务
$conn->commit();
echo "新记录插入成功";
}
catch(PDOException $e)
{
// 如果执行失败回滚
$conn->rollback();
echo $sql . "<br>" . $e->getMessage();
}
$conn = null;
```

```
?>
```

使用预处理语句

mysqli 扩展提供了第二种方式用于插入语句。

我们可以预处理语句及绑定参数。

mysql 扩展可以不带数据发送语句或查询到mysql数据库。 你可以向列关联或 "绑定" 变量。

实例 (MySQLi 使用预处理语句)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
} else {
$sql = "INSERT INTO MyGuests(firstname, lastname, email) VALUES(?, ?, ?)";
// 为 mysqli_stmt_prepare() 初始化 statement 对象
$stmt = mysqli_stmt_init($conn);
//预处理语句
if (mysqli_stmt_prepare($stmt, $sql)) {
// 绑定参数
mysqli_stmt_bind_param($stmt, 'sss', $firstname, $lastname, $email);
// 设置参数并执行
$firstname = 'John';
$lastname = 'Doe';
$email = 'john@example.com';
mysqli_stmt_execute($stmt);
$firstname = 'Mary';
$lastname = 'Moe';
$email = 'mary@example.com';
mysqli_stmt_execute($stmt);
$firstname = 'Julie';
$lastname = 'Dooley';
$email = 'julie@example.com';
mysqli_stmt_execute($stmt);
}
}
?>
```

我们可以看到以上实例中使用模块化来处理问题。我们可以通过创建代码块实现更简单的读取和管理。

注意参数的绑定。让我们看下 mysqli_stmt_bind_param() 中的代码：

```
mysqli_stmt_bind_param($stmt, 'sss', $firstname, $lastname, $email);
```

该函数绑定参数查询并将参数传递给数据库。第二个参数是 "sss" 。以下列表展示了参数的类型。 s 字符告诉 mysql 参数是字符串。

可以是以下四种参数:

- i - 整数
- d - 双精度浮点数
- s - 字符串
- b - 布尔值

每个参数必须指定类型，来保证数据的安全性。通过类型的判断可以减少SQL注入漏洞带来的风险。

PHP MySQL 预处理语句

预处理语句对于防止 MySQL 注入是非常有用的。

预处理语句及绑定参数

预处理语句用于执行多个相同的 SQL 语句，并且执行效率更高。

预处理语句的工作原理如下：

1. 预处理：创建 SQL 语句模板并发送到数据库。预留的值使用参数 "?" 标记 。例如：

```
INSERT INTO MyGuests (firstname, lastname, email) VALUES(?, ?, ?)
```

2. 数据库解析，编译，对SQL语句模板执行查询优化，并存储结果不输出。

3. 执行：最后，将应用绑定的值传递给参数（"?" 标记），数据库执行语句。应用可以多次执行语句，如果参数的值不一样。

相比于直接执行SQL语句，预处理语句有两个主要优点：

预处理语句大大减少了分析时间，只做了一次查询（虽然语句多次执行）。

绑定参数减少了服务器带宽，你只需要发送查询的参数，而不是整个语句。

预处理语句针对SQL注入是非常有用的，因为参数值发送后使用不同的协议，保证了数据的合法性。

MySQLi 预处理语句

以下实例在 MySQLi 中使用了预处理语句，并绑定了相应的参数:

实例 (MySQLi 使用预处理语句)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
}
// 预处理及绑定
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);
// 设置参数并执行
```

```php
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();
$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();
$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();
echo "新记录插入成功";
$stmt->close();
$conn->close();
?>
```

解析以下实例的每行代码:

```php
"INSERT INTO MyGuests (firstname, lastname, email) VALUES(?, ?, ?)"
```

在 SQL 语句中，我们使用了问号 (?)，在此我们可以将问号替换为整型，字符串，双精度浮点型和布尔值。

接下来，让我们来看下 bind_param() 函数：

```php
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

该函数绑定了 SQL 的参数，且告诉数据库参数的值。 "sss" 参数列处理其余参数的数据类型。s 字符告诉数据库该参数为字符串。

参数有以下四种类型:

- i - integer（整型）
- d - double（双精度浮点型）
- s - string（字符串）
- b - BLOB（binary large object:二进制大对象）

每个参数都需要指定类型。

通过告诉数据库参数的数据类型，可以降低 SQL 注入的风险。

> **Note** **注意：** 如果你想插入其他数据（用户输入），对数据的验证是非常重要的。

PDO 中的预处理语句

以下实例我们在 PDO 中使用了预处理语句并绑定参数:

实例 (PDO 使用预处理语句)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
// 设置 PDO 错误模式为异常
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// 预处理 SQL 并绑定参数
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (:firstname, :lastname, :email)");
$stmt->bindParam(':firstname', $firstname);
$stmt->bindParam(':lastname', $lastname);
$stmt->bindParam(':email', $email);
// 插入行
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();
```

```php
// 插入其他行
$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();
// 插入其他行
$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();
echo "新记录插入成功";
}
catch(PDOException $e)
{
echo "Error: " . $e->getMessage();
}
$conn = null;
?>
```

PHP MySQL 创建表 PHP 图像处理

PHP MySQL 插入数据 PHP MySQL Where 子句

PHP MySQL 读取数据

从 MySQL 数据库读取数据

SELECT 语句用于从数据表中读取数据:

```
SELECT column_name(s) FROM table_name
```

我们可以使用 * 号来读取所有数据表中的字段：

```
SELECT * FROM table_name
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

使用 MySQLi

以下实例中我们从 myDB 数据库的 MyGuests 表读取了 id, firstname 和 lastname 列的数据并显示在页面上：

实例 (MySQLi - 面向对象)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
die("连接失败: " . $conn->connect_error);
}
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);
if ($result->num_rows > 0) {
// 输出数据
while($row = $result->fetch_assoc()) {
echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
}
} else {
echo "0 结果";
}
$conn->close();
?>
```

以上代码解析如下:

首先，我们设置了 SQL 语句从 MyGuests数据表中读取 id, firstname 和 lastname 三个字段。之后我们使用改 SQL 语句从数据库中取出结果集并赋给复制给变量 $result。

函数 num_rows() 判断返回的数据。

如果返回的是多条数据，函数 fetch_assoc() 将结合集放入到关联数组并循环输出。 while() 循环出结果集，并输出 id, firstname 和 lastname 三个字段值。

以下实例使用 MySQLi 面向过程的方式，效果类似以上代码:

实例 (MySQLi - 面向过程)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// 创建连接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
die("连接失败: " . mysqli_connect_error());
}
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
// 输出数据
while($row = mysqli_fetch_assoc($result)) {
echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
}
} else {
echo "0 结果";
}
mysqli_close($conn);
?>
```

使用 PDO (+ 预处理)

以下实例使用了预处理语句。

选取了 MyGuests 表中的 id, firstname 和 lastname 字段，并放到 HTML 表格中：

实例 (PDO)

```php
<?php
```

```php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";
class TableRows extends RecursiveIteratorIterator {
function __construct($it) {
parent::__construct($it, self::LEAVES_ONLY);
}
function current() {
return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
}
function beginChildren() {
echo "<tr>";
}
function endChildren() {
echo "</tr>" . "\n";
}
}
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
$conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
$stmt->execute();
// 设置结果集为关联数组
$result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
echo $v;
}
}
catch(PDOException $e) {
echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

□ PHP MySQL 插入数据 PHP MySQL Where 子句 □

□点我分享笔记

反馈/建议

PHP MySQL Where 子句

WHERE 子句用于过滤记录。

WHERE 子句

WHERE 子句用于提取满足指定标准的的记录。

语法

```
SELECT column_name(s)

FROM table_name

WHERE column_name operator value
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

下面的实例将从 "Persons" 表中选取所有 FirstName='Peter' 的行：

```
<?php

$con=mysqli_connect("localhost","username","password","database");

// 检测连接

if (mysqli_connect_errno())

{

    echo "连接失败: " . mysqli_connect_error();

}



$result = mysqli_query($con,"SELECT * FROM Persons

WHERE FirstName='Peter'");



while($row = mysqli_fetch_array($result))

{

    echo $row['FirstName'] . " " . $row['LastName'];

    echo "<br>";

}

?>
```

以上代码将输出：

```
Peter Griffin
```

PHP MySQL Order By 关键词

ORDER BY 关键词用于对记录集中的数据进行排序。

ORDER BY 关键词

ORDER BY 关键词用于对记录集中的数据进行排序。

ORDER BY 关键词默认对记录进行升序排序。

如果你想降序排序，请使用 DESC 关键字。

语法

```
SELECT column_name(s)

FROM table_name

ORDER BY column_name(s) ASC|DESC
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

实例

下面的实例选取 "Persons" 表中存储的所有数据，并根据 "Age" 列对结果进行排序：

```
<?php

$con=mysqli_connect("localhost","username","password","database");

// 检测连接

if (mysqli_connect_errno())

{

    echo "连接失败: " . mysqli_connect_error();

}
```

```
$result = mysqli_query($con,"SELECT * FROM Persons ORDER BY age");

while($row = mysqli_fetch_array($result))
{
    echo $row['FirstName'];
    echo " " . $row['LastName'];
    echo " " . $row['Age'];
    echo "<br>";
}

mysqli_close($con);
?>
```

以上结果将输出：

```
Glenn Quagmire 33
Peter Griffin 35
```

根据两列进行排序

可以根据多个列进行排序。当按照多个列进行排序时，只有第一列的值相同时才使用第二列：

```
SELECT column_name(s)
FROM table_name
ORDER BY column1, column2
```

PHP MySQL Where 子句 PHP MySQL Update

PHP MySQL Update

UPDATE 语句用于中修改数据库表中的数据。

更新数据库中的数据

UPDATE 语句用于更新数据库表中已存在的记录。

语法

```
UPDATE table_name

SET column1=value, column2=value2,...

WHERE some_column=some_value
```

注释：请注意 UPDATE 语法中的 WHERE 子句。WHERE 子句规定了哪些记录需要更新。如果您想省去 WHERE 子句，所有的记录都会被更新！

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

在本教程的前面章节中，我们创建了一个名为 "Persons" 的表，如下所示：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的例子更新 "Persons" 表的一些数据：

```
<?php

$con=mysqli_connect("localhost","username","password","database");

// 检测连接

if (mysqli_connect_errno())

{

    echo "连接失败: " . mysqli_connect_error();

}



mysqli_query($con,"UPDATE Persons SET Age=36

WHERE FirstName='Peter' AND LastName='Griffin'");



mysqli_close($con);
```

```
?>
```

在这次更新后，"Persons" 表如下所示：

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

PHP MySQL Order By 关键词　　PHP MySQL Delete

PHP MySQL Update　　PHP 数据库 ODBC

PHP MySQL Delete

DELETE 语句用于从数据库表中删除行。

删除数据库中的数据

DELETE FROM 语句用于从数据库表中删除记录。

语法

```
DELETE FROM table_name

WHERE some_column = some_value
```

注释：请注意 DELETE 语法中的 WHERE 子句。WHERE 子句规定了哪些记录需要删除。如果您想省去 WHERE 子句，所有的记录都会被删除！

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

请看下面的 "Persons" 表：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的实例删除 "Persons" 表中所有 LastName='Griffin' 的记录：

```
<?php
$con=mysqli_connect("localhost","username","password","database");
// 检测连接
if (mysqli_connect_errno())
{
    echo "连接失败: " . mysqli_connect_error();
}

mysqli_query($con,"DELETE FROM Persons WHERE LastName='Griffin'");

mysqli_close($con);
?>
```

在这次删除后，"Persons" 表如下所示：

FirstName	LastName	Age
Glenn	Quagmire	33

PHP MySQL Update　　PHP 数据库 ODBC

点我分享笔记

反馈/建议

PHP MySQL Delete　　PHP XML Expat 解析器

PHP 数据库 ODBC

ODBC 是一种应用程序编程接口（Application Programming Interface，API），使我们有能力连接到某个数据源（比如一个 MS Access 数据库）。

创建 ODBC 连接

通过一个 ODBC 连接，您可以连接到您的网络中的任何计算机上的任何数据库，只要 ODBC 连接是可用的。

这是创建到达 MS Access 数据库的 ODBC 连接的方法：

1. 在控制面板中打开**管理工具**图标。
2. 双击其中的**数据源(ODBC)**图标。
3. 选择**系统 DSN** 选项卡。
4. 点击系统 DSN 选项卡中的**添加**。
5. 选择**Microsoft Access Driver**。点击**完成**。
6. 在下一个界面，点击**选择**来定位数据库。
7. 为数据库起一个**数据源名(DSN)**。
8. 点击**确定**。

请注意，必须在您的网站所在的计算机上完成这个配置。如果您的计算机上正在运行 Internet 信息服务(IIS)，上面的指令将会生效，但是如果您的网站位于远程服务器，您必须拥有对该服务器的物理访问权限，或者请您的主机提供商为您建立 DSN。

连接到 ODBC

odbc_connect() 函数用于连接到 ODBC 数据源。该函数有四个参数：数据源名、用户名、密码以及可选的指针类型。

odbc_exec() 函数用于执行 SQL 语句。

实例

下面的实例创建了到达名为 northwind 的 DSN 的连接，没有用户名和密码。然后创建并执行一条 SQL 语句：

```
$conn=odbc_connect('northwind','','');

$sql="SELECT * FROM customers";

$rs=odbc_exec($conn,$sql);
```

取回记录

odbc_fetch_row() 函数用于从结果集中返回记录。如果能够返回行，则函数返回 true，否则返回 false。

该函数有两个参数：ODBC 结果标识符和可选的行号：

```
odbc_fetch_row($rs)
```

从记录中取回字段

odbc_result() 函数用于从记录中读取字段。该函数有两个参数：ODBC 结果标识符和字段编号或名称。

下面的代码行从记录中返回第一个字段的值：

```
$compname=odbc_result($rs,1);
```

下面的代码行返回名为 "CompanyName" 的字段的值：

```
$compname=odbc_result($rs,"CompanyName");
```

关闭 ODBC 连接

odbc_close() 函数用于关闭 ODBC 连接。

```
odbc_close($conn);
```

ODBC 实例

下面的实例展示了如何首先创建一个数据库连接，接着创建一个结果集，然后在 HTML 表格中显示数据。

```
<html>
<body>

<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
{
    exit("连接失败: " . $conn);
}

$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);

if (!$rs)
{
    exit("SQL 语句错误");
}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";

while (odbc_fetch_row($rs))
{
    $compname=odbc_result($rs,"CompanyName");
    $conname=odbc_result($rs,"ContactName");
```

```
    echo "<tr><td>$compname</td>";
    echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>

</body>
</html>
```

PHP MySQL Delete　　PHP XML Expat 解析器

PHP 数据库 ODBC　　PHP XML DOM

PHP XML Expat 解析器

内建的 Expat 解析器使在 PHP 中处理 XML 文档成为可能。

XML 是什么？

XML 用于描述数据，其焦点是数据是什么。XML 文件描述了数据的结构。

在 XML 中，没有预定义的标签。您必须定义自己的标签。

如需学习更多关于 XML 的知识，请访问我们的 XML 教程。

Expat 是什么？

如需读取和更新 - 创建和处理 - 一个 XML 文档，您需要 XML 解析器。

有两种基本的 XML 解析器类型：

基于树的解析器：这种解析器把 XML 文档转换为树型结构。它分析整篇文档，并提供了对树中元素的访问，例如文档对象模型 (DOM)。

基于事件的解析器：将 XML 文档视为一系列的事件。当某个具体的事件发生时，解析器会调用函数来处理。

Expat 解析器是基于事件的解析器。

基于事件的解析器集中在 XML 文档的内容，而不是它们的结构。正因为如此，基于事件的解析器能够比基于树的解析器更快地访问数据。

请看下面的 XML 片段：

```
<from>Jani</from>
```

基于事件的解析器把上面的 XML 报告为一连串的三个事件：

开始元素：from

开始 CDATA 部分，值：Jani

关闭元素：from

上面的 XML 实例包含了形式良好的 XML。不过这个实例是无效的 XML，因为没有与它关联的文档类型声明 (DTD)。

然而，在使用 Expat 解析器时，这没有区别。Expat 是不检查有效性的解析器，忽略任何 DTD。

作为一款基于事件、非验证的 XML 解析器，Expat 快速且轻巧，十分适合 PHP 的 Web 应用程序。

注释：XML 文档必须形式良好，否则 Expat 会生成错误。

安装

XML Expat 解析器函数是 PHP 核心的组成部分。无需安装就可以使用这些函数。

XML 文件

下面的 XML 文件将应用在我们的实例中：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

初始化 XML 解析器

我们要在 PHP 中初始化 XML 解析器，为不同的 XML 事件定义处理器，然后解析这个 XML 文件。

实例

```
<?php
//Initialize the XML parser
$parser=xml_parser_create();

//Function to use at the start of an element
function start($parser,$element_name,$element_attrs)
{
switch($element_name)
{
case "NOTE":
echo "-- Note --<br>";
break;
case "TO":
echo "To: ";
break;
case "FROM":
echo "From: ";
break;
case "HEADING":
echo "Heading: ";
break;
case "BODY":
echo "Message: ";
}
}

//Function to use at the end of an element
function stop($parser,$element_name)
{
echo "<br>";
}

//Function to use when finding character data
function char($parser,$data)
{
echo $data;
}
```

```
//Specify element handler
xml_set_element_handler($parser,"start","stop");

//Specify data handler
xml_set_character_data_handler($parser,"char");

//Open XML file
$fp=fopen("test.xml","r");

//Read data
while ($data=fread($fp,4096))
{
xml_parse($parser,$data,feof($fp)) or
die (sprintf("XML Error: %s at line %d",
xml_error_string(xml_get_error_code($parser)),
xml_get_current_line_number($parser)));
}

//Free the XML parser
xml_parser_free($parser);
?>
```

以上代码将输出：

```
-- Note --
To: Tove
From: Jani
Heading: Reminder
Message: Don't forget me this weekend!
```

工作原理：

1. 通过 xml_parser_create() 函数初始化 XML 解析器
2. 创建配合不同事件处理程序的的函数
3. 添加 xml_set_element_handler() 函数来定义，当解析器遇到开始和结束标签时执行哪个函数
4. 添加 xml_set_character_data_handler() 函数来定义，当解析器遇到字符数据时执行哪个函数
5. 通过 xml_parse() 函数来解析文件 "test.xml"
6. 万一有错误的话，添加 xml_error_string() 函数把 XML 错误转换为文本说明
7. 调用 xml_parser_free() 函数来释放分配给 xml_parser_create() 函数的内存

更多 PHP Expat 解析器的信息

如需了解更多关于 PHP Expat 函数的信息，请访问我们的 PHP XML Parser 参考手册。

□ PHP 数据库 ODBC　　PHP XML DOM □

□点我分享笔记

反馈/建议

PHP XML DOM

内建的 DOM 解析器使在 PHP 中处理 XML 文档成为可能。

DOM 是什么？

W3C DOM 提供了针对 HTML 和 XML 文档的标准对象集，以及用于访问和操作这些文档的标准接口。

W3C DOM 被分为不同的部分（Core, XML 和 HTML）和不同的级别（DOM Level 1/2/3）：

* Core DOM - 为任何结构化文档定义标准的对象集
* XML DOM - 为 XML 文档定义标准的对象集
* HTML DOM - 为 HTML 文档定义标准的对象集

如需学习更多关于 XML DOM 的知识，请访问我们的 XML DOM 教程。

XML 解析

如需读取和更新 - 创建和处理 - 一个 XML 文档，您需要 XML 解析器。

有两种基本的 XML 解析器类型：

基于树的解析器：这种解析器把 XML 文档转换为树型结构。它分析整篇文档，并提供了对树中元素的访问，例如文档对象模型 (DOM)。

基于事件的解析器：将 XML 文档视为一系列的事件。当某个具体的事件发生时，解析器会调用函数来处理。

DOM 解析器是基于树的解析器。

请看下面的 XML 文档片段：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<from>Jani</from>
```

XML DOM 把上面的 XML 视为一个树形结构：

Level 1: XML 文档

Level 2: 根元素： <from>

Level 3: 文本元素： "Jani"

安装

DOM XML 解析器函数是 PHP 核心的组成部分。无需安装就可以使用这些函数。

XML 文件

下面的 XML 文件将应用在我们的实例中：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

加载和输出 XML

我们需要初始化 XML 解析器，加载 XML，并把它输出：

实例

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

以上代码将输出：

```
ToveJaniReminder Don't forget me this weekend!
```

如果您在浏览器窗口中查看源代码，会看到下面的 HTML：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

上面的实例创建了一个 DOMDocument-Object，并把 "note.xml" 中的 XML 载入这个文档对象中。

saveXML() 函数把内部 XML 文档放入一个字符串，这样我们就可以输出它。

遍历 XML

我们要初始化 XML 解析器，加载 XML，并遍历 <note> 元素的所有元素：

实例

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item)
{
print $item->nodeName . " = " . $item->nodeValue . "<br>";
}
?>
```

以上代码将输出：

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

在上面的实例中，您看到了每个元素之间存在空的文本节点。

当 XML 生成时，它通常会在节点之间包含空白。XML DOM 解析器把它们当作普通的元素，如果您不注意它们，有时会产生问题。

如需学习更多关于 XML DOM 的知识，请访问我们的 XML DOM 教程。

□ PHP XML Expat 解析器　　PHP XML SimpleXML □

首页 HTML CSS JS 本地书签

□ PHP XML DOM　　AJAX 简介 □

PHP SimpleXML

PHP SimpleXML 处理最普通的 XML 任务，其余的任务则交由其它扩展处理。

什么是 PHP SimpleXML?

SimpleXML 是 PHP 5 中的新特性。

SimpleXML 扩展提供了一种获取 XML 元素的名称和文本的简单方式。

与 DOM 或 Expat 解析器相比，SimpleXML 仅仅用几行代码就可以从 XML 元素中读取文本数据。

SimpleXML 可把 XML 文档（或 XML 字符串）转换为对象，比如：

元素被转换为 SimpleXMLElement 对象的单一属性。当同一级别上存在多个元素时，它们会被置于数组中。

属性通过使用关联数组进行访问，其中的索引对应属性名称。

元素内部的文本被转换为字符串。如果一个元素拥有多个文本节点，则按照它们被找到的顺序进行排列。

当执行类似下列的基础任务时，SimpleXML 使用起来非常快捷：

读取/提取 XML 文件/字符串的数据

编辑文本节点或属性

然而，在处理高级 XML 时，比如命名空间，最好使用 Expat 解析器或 XML DOM。

安装

从 PHP 5 开始，SimpleXML 函数是 PHP 核心的组成部分。无需安装就可以使用这些函数。

PHP SimpleXML 实例

假设我们有如下的 XML 文件，"note.xml"：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

现在我们想要输出上面的 XML 文件的不同信息：

实例 1

输出 $xml 变量（是 SimpleXMLElement 对象）的键和元素：

```
<?php
$xml=simplexml_load_file("note.xml");
print_r($xml);
?>
```

运行实例 »

以上代码将输出：

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder [body] => Don't forget me this weekend! )
```

实例 2

输出 XML 文件中每个元素的数据：

```
<?php
$xml=simplexml_load_file("note.xml");
echo $xml->to . "<br>";
```

```php
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

运行实例 »

以上代码将输出：

```
Tove

Jani

Reminder

Don't forget me this weekend!
```

实例 3

输出每个子节点的元素名称和数据：

```php
<?php
$xml=simplexml_load_file("note.xml");
echo $xml->getName() . "<br>";
foreach($xml->children() as $child)
{
echo $child->getName() . ": " . $child . "<br>";
}
?>
```

运行实例 »

以上代码将输出：

```
note

to: Tove

from: Jani

heading: Reminder

body: Don't forget me this weekend!
```

更多 PHP SimpleXML 的信息

如需了解更多关于 PHP SimpleXML 函数的信息，请访问我们的 PHP SimpleXML 参考手册。

PHP XML DOM　　AJAX 简介

AJAX 简介

AJAX 是一种在无需重新加载整个网页的情况下，能够更新部分网页的技术。

AJAX 是什么？

AJAX = Asynchronous JavaScript and XML.

AJAX 是一种用于创建快速动态网页的技术。

AJAX 通过在后台与服务器进行少量数据交换，使网页实现异步更新。这意味着可以在不重载整个页面的情况下，对网页的某些部分进行更新。

传统的网页（不使用 AJAX）如果需要更新内容，必须重载整个页面。

有很多使用 AJAX 的应用程序案例：Google Maps、Gmail、Youtube 和 Facebook。

AJAX 如何工作

AJAX

AJAX 基于因特网标准

AJAX 基于因特网标准，并使用以下技术组合：

- XMLHttpRequest 对象（与服务器异步交互数据）
- JavaScript/DOM（显示/取回信息）
- CSS（设置数据的样式）
- XML（常用作数据传输的格式）

AJAX 应用程序与浏览器和平台无关的！

谷歌搜索建议（Google Suggest）

随着谷歌搜索建议功能在 2005 的发布，AJAX 开始流行起来。

谷歌搜索建议（Google Suggest） 使用 AJAX 创造出动态性极强的 web 界面：当您在谷歌的搜索框中键入内容时，JavaScript 会把字符发送到服务器，服务器则会返回建议列表。

今天就开始使用 AJAX

在我们的 PHP 教程中，我们将演示 AJAX 如何在不重载整个页面的情况下对网页的某些部分进行更新。服务器脚本我们将采用 PHP 来编写。

如果您想要学习更多关于 AJAX 的知识，请访问我们的 AJAX 教程。

PHP - AJAX 与 PHP

AJAX被用于创建交互性更强的应用程序。

AJAX PHP 实例

下面的实例将演示当用户在输入框中键入字符时，网页如何与 Web 服务器进行通信：

实例

尝试在输入框中输入一个名字，如：Anna:

姓名:

返回值:

实例解释 - HTML 页面

当用户在上面的输入框中键入字符时，会执行 "showHint()" 函数。该函数由 "onkeyup" 事件触发：

```
<html>
<head>
<script>
function showHint(str)
{
    if (str.length==0)
    {
        document.getElementById("txtHint").innerHTML="";
        return;
    }
    if (window.XMLHttpRequest)
    {
        // IE7+, Firefox, Chrome, Opera, Safari 浏览器执行的代码
        xmlhttp=new XMLHttpRequest();
    }
    else
```

```
    {
        //IE6, IE5 浏览器执行的代码
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","gethint.php?q="+str,true);
    xmlhttp.send();
}
</script>
</head>
<body>

<p><b>在输入框中输入一个姓名:</b></p>
<form>
姓名: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>返回值: <span id="txtHint"></span></p>

</body>
</html>
```

源代码解释：

如果输入框是空的（str.length==0），该函数会清空 txtHint 占位符的内容，并退出该函数。

如果输入框不是空的，那么 showHint() 会执行以下步骤：

- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含输入框的内容）

PHP 文件

上面这段通过 JavaScript 调用的服务器页面是名为 "gethint.php" 的 PHP 文件。

"gethint.php" 中的源代码会检查姓名数组，然后向浏览器返回对应的姓名：

```
<?php

// 将姓名填充到数组中

$a[]="Anna";

$a[]="Brittany";

$a[]="Cinderella";

$a[]="Diana";

$a[]="Eva";

$a[]="Fiona";

$a[]="Gunda";

$a[]="Hege";

$a[]="Inga";

$a[]="Johanna";

$a[]="Kitty";

$a[]="Linda";

$a[]="Nina";

$a[]="Ophelia";

$a[]="Petunia";

$a[]="Amanda";

$a[]="Raquel";

$a[]="Cindy";

$a[]="Doris";

$a[]="Eve";

$a[]="Evita";

$a[]="Sunniva";

$a[]="Tove";

$a[]="Unni";

$a[]="Violet";

$a[]="Liza";

$a[]="Elizabeth";
```

```
$a[]="Ellen";

$a[]="Wenche";

$a[]="Vicky";



//从请求URL地址中获取 q 参数

$q=$_GET["q"];



//查找是否由匹配值， 如果 q>0

if (strlen($q) > 0)

{

    $hint="";

    for($i=0; $i<count($a); $i++)

    {

        if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))

        {

            if ($hint=="")

            {

                $hint=$a[$i];

            }

            else

            {

                $hint=$hint." , ".$a[$i];

            }

        }

    }

}



// 如果没有匹配值设置输出为 "no suggestion"

if ($hint == "")

{

    $response="no suggestion";

}

else
```

```
{
    $response=$hint;
}

//输出返回值
echo $response;
?>
```

解释：如果 JavaScript 发送了任何文本（即 strlen($q) > 0），则会发生：

1. 查找匹配 JavaScript 发送的字符的姓名
2. 如果未找到匹配，则将响应字符串设置为 "no suggestion"
3. 如果找到一个或多个匹配姓名，则用所有姓名设置响应字符串
4. 把响应发送到 "txtHint" 占位符

PHP Ajax 跨域问题解决方案

如果你的异步请求需要跨域可以查看：PHP Ajax 跨域问题解决方案。

AJAX 简介 PHP 实例 AJAX 与 MySQL

点我分享笔记

反馈/建议

PHP – AJAX 与 PHP PHP 实例 AJAX 与 XML

PHP - AJAX 与 MySQL

AJAX可用来与数据库进行交互式通信。

AJAX 数据库实例

下面的实例将演示网页如何通过 AJAX 从数据库读取信息：

本教程使用到的 Websites 表 SQL 文件：websites.sql。

实例

选择一个网站:

选择对应选项，用户信息会显示在这……

实例解释 - MySQL 数据库

在上面的实例中，我们使用的数据库表如下所示：

```
mysql> select * from websites;
+----+--------------+---------------------------+-------+---------+
| id | name         | url                       | alexa | country |
+----+--------------+---------------------------+-------+---------+
| 1  | Google       | https://www.google.cm/    | 1     | USA     |
| 2  | 淘宝         | https://www.taobao.com/   | 13    | CN      |
| 3  | 菜鸟教程 | http://www.runoob.com/    | 4689  | CN      |
| 4  | 微博         | http://weibo.com/         | 20    | CN      |
| 5  | Facebook     | https://www.facebook.com/ | 3     | USA     |
+----+--------------+---------------------------+-------+---------+
5 rows in set (0.01 sec)
```

实例解释 - HTML 页面

当用户在上面的下拉列表中选择某位用户时，会执行名为 "showSite()" 的函数。该函数由 "onchange" 事件触发：

test.html 文件代码：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<script>
function showSite(str)
{
if (str=="")
{
document.getElementById("txtHint").innerHTML="";
return;
}
if (window.XMLHttpRequest)
{
// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
xmlhttp=new XMLHttpRequest();
}
else
{
// IE6, IE5 浏览器执行代码
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
{
document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
}
}
xmlhttp.open("GET","getsite_mysql.php?q="+str,true);
```

```
xmlhttp.send();
}
</script>
</head>
<body>
<form>
<select name="users" onchange="showSite(this.value)">
<option value="">选择一个网站:</option>
<option value="1">Google</option>
<option value="2">淘宝</option>
<option value="3">菜鸟教程</option>
<option value="4">微博</option>
<option value="5">Facebook</option>
</select>
</form>
<br>
<div id="txtHint"><b>网站信息显示在这里……</b></div>
</body>
</html>
```

showSite() 函数会执行以下步骤：

- 检查是否有网站被选择
- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含下拉列表的内容）

PHP 文件

上面这段通过 JavaScript 调用的服务器页面是名为 "getsite_mysql.php" 的 PHP 文件。

"getsite_mysql.php" 中的源代码会运行一次针对 MySQL 数据库的查询，然后在 HTML 表格中返回结果：

getsite_mysql.php 文件代码：

```
<?php
$q = isset($_GET["q"]) ? intval($_GET["q"]) : '';
if(empty($q)) {
echo '请选择一个网站';
exit;
}
$con = mysqli_connect('localhost','root','123456');
if (!$con)
{
die('Could not connect: ' . mysqli_error($con));
}
// 选择数据库
mysqli_select_db($con,"test");
// 设置编码，防止中文乱码
mysqli_set_charset($con, "utf8");
$sql="SELECT * FROM Websites WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);
echo "<table border='1'>
<tr>
<th>ID</th>
<th>网站名</th>
<th>网站 URL</th>
<th>Alexa 排名</th>
<th>国家</th>
</tr>";
while($row = mysqli_fetch_array($result))
{
echo "<tr>";
echo "<td>" . $row['id'] . "</td>";
echo "<td>" . $row['name'] . "</td>";
echo "<td>" . $row['url'] . "</td>";
echo "<td>" . $row['alexa'] . "</td>";
echo "<td>" . $row['country'] . "</td>";
```

```
echo "</tr>";
}
echo "</table>";
mysqli_close($con);
?>
```

解释：当查询从 JavaScript 发送到 PHP 文件时，将发生：

1. PHP 打开一个到 MySQL 数据库的连接
2. 找到选中的用户
3. 创建 HTML 表格，填充数据，并发送回 "txtHint" 占位符

PHP – AJAX 与 PHP　　PHP 实例 AJAX 与 XML

PHP 实例 AJAX 与 MySQL　　PHP 实例 AJAX 实时搜索

PHP 实例 - AJAX 与 XML

AJAX 可用来与 XML 文件进行交互式通信。

AJAX XML 实例

下面的实例将演示网页如何通过 AJAX 从 XML 文件读取信息：

实例

Select a CD:

CD info will be listed here...

实例解释 - HTML 页面

当用户在上面的下拉列表中选择某张 CD 时，会执行名为 "showCD()" 的函数。该函数由 "onchange" 事件触发：

```
<html>
<head>
<script>
function showCD(str)
```

```
{
    if (str=="")
    {
        document.getElementById("txtHint").innerHTML="";
        return;
    }
    if (window.XMLHttpRequest)
    {
        // IE7+, Firefox, Chrome, Opera, Safari 浏览器执行
        xmlhttp=new XMLHttpRequest();
    }
    else
    {
        // IE6, IE5 浏览器执行
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
        }
    }
    xmlhttp.open("GET","getcd.php?q="+str,true);
    xmlhttp.send();
}
</script>
</head>
<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
```

```
<option value="">Select a CD:</option>

<option value="Bob Dylan">Bob Dylan</option>

<option value="Bonnie Tyler">Bonnie Tyler</option>

<option value="Dolly Parton">Dolly Parton</option>

</select>

</form>

<div id="txtHint"><b>CD info will be listed here...</b></div>



</body>

</html>
```

showCD() 函数会执行以下步骤：

- 检查是否有 CD 被选择
- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含下拉列表的内容）

PHP 文件

上面这段通过 JavaScript 调用的服务器页面是名为 "getcd.php" 的 PHP 文件。

PHP 脚本加载 XML 文档，"cd_catalog.xml"，运行针对 XML 文件的查询，并以 HTML 返回结果：

```
<?php

$q=$_GET["q"];



$xmlDoc = new DOMDocument();

$xmlDoc->load("cd_catalog.xml");



$x=$xmlDoc->getElementsByTagName('ARTIST');



for ($i=0; $i<=$x->length-1; $i++)

{

    // 处理元素节点

    if ($x->item($i)->nodeType==1)

    {
```

```
        if ($x->item($i)->childNodes->item(0)->nodeValue == $q)
        {
            $y=($x->item($i)->parentNode);
        }
    }
}

$cd=($y->childNodes);

for ($i=0;$i<$cd->length;$i++)
{
    // 处理元素节点
    if ($cd->item($i)->nodeType==1)
    {
        echo("<b>" . $cd->item($i)->nodeName . ":</b> ");
        echo($cd->item($i)->childNodes->item(0)->nodeValue);
        echo("<br>");
    }
}
?>
```

当 CD 查询从 JavaScript 发送到 PHP 页面时，将发生：

1. PHP 创建 XML DOM 对象
2. 查找所有 <artist> 元素中与 JavaScript 所传数据相匹配的名字
3. 输出 album 的信息，并发送回 "txtHint" 占位符

□ PHP 实例 AJAX 与 MySQL　　PHP 实例 AJAX 实时搜索 □

PHP 实例 AJAX 与 XML

PHP 实例 AJAX RSS 阅读器

PHP 实例 - AJAX 实时搜索

AJAX 可为用户提供更友好、交互性更强的搜索体验。

AJAX Live Search

在下面的实例中，我们将演示一个实时的搜索，在您键入数据的同时即可得到搜索结果。

实时的搜索与传统的搜索相比，具有很多优势：

当键入数据时，就会显示出匹配的结果

当继续键入数据时，对结果进行过滤

如果结果太少，删除字符就可以获得更宽的范围

在下面的文本框中输入 "HTML"，搜索包含 HTML 的页面：

上面实例中的结果在一个 XML 文件（links.xml）中进行查找。为了让这个例子小而简单，我们只提供 6 个结果。

实例解释 - HTML 页面

当用户在上面的输入框中键入字符时，会执行 "showResult()" 函数。该函数由 "onkeyup" 事件触发：

```
<html>
<head>
<script>
function showResult(str)
{
    if (str.length==0)
    {
        document.getElementById("livesearch").innerHTML="";
        document.getElementById("livesearch").style.border="0px";
        return;
    }
    if (window.XMLHttpRequest)
    {// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行
        xmlhttp=new XMLHttpRequest();
    }
    else
    {// IE6, IE5 浏览器执行
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

```
    }
    xmlhttp.onreadystatechange=function()
    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("livesearch").innerHTML=xmlhttp.responseText;
            document.getElementById("livesearch").style.border="1px solid #A5ACB2";
        }
    }
    xmlhttp.open("GET","livesearch.php?q="+str,true);
    xmlhttp.send();
}
</script>
</head>
<body>

<form>
<input type="text" size="30" onkeyup="showResult(this.value)">
<div id="livesearch"></div>
</form>

</body>
</html>
```

源代码解释：

如果输入框是空的（str.length==0），该函数会清空 livesearch 占位符的内容，并退出该函数。

如果输入框不是空的，那么 showResult() 会执行以下步骤：

- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含输入框的内容）

PHP 文件

上面这段通过 JavaScript 调用的服务器页面是名为 "livesearch.php" 的 PHP 文件。

"livesearch.php" 中的源代码会搜索 XML 文件中匹配搜索字符串的标题，并返回结果：

```php
<?php
$xmlDoc=new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

// 从 URL 中获取参数 q 的值
$q=$_GET["q"];

// 如果 q 参数存在则从 xml 文件中查找数据
if (strlen($q)>0)
{
    $hint="";
    for($i=0; $i<($x->length); $i++)
    {
        $y=$x->item($i)->getElementsByTagName('title');
        $z=$x->item($i)->getElementsByTagName('url');
        if ($y->item(0)->nodeType==1)
        {
            // 找到匹配搜索的链接
            if (stristr($y->item(0)->childNodes->item(0)->nodeValue,$q))
            {
                if ($hint=="")
                {
                    $hint="<a href='" .
                    $z->item(0)->childNodes->item(0)->nodeValue .
                    "' target='_blank'>" .
                    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
                }
                else
                {
                    $hint=$hint . "<br /><a href='" .
```

```
                    $z->item(0)->childNodes->item(0)->nodeValue .
                    "' target='_blank'>" .
                    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
                }
            }
        }
    }
}

// 如果没找到则返回 "no suggestion"
if ($hint=="")
{
    $response="no suggestion";
}
else
{
    $response=$hint;
}

// 输出结果
echo $response;
?>
```

如果 JavaScript 发送了任何文本（即 strlen($q) > 0），则会发生：

加载 XML 文件到新的 XML DOM 对象

遍历所有的 <title> 元素，以便找到匹配 JavaScript 所传文本

在 "$response" 变量中设置正确的 URL 和标题。如果找到多于一个匹配，所有的匹配都会添加到变量。

如果没有找到匹配，则把 $response 变量设置为 "no suggestion"。

□ PHP 实例 AJAX 与 XML　　PHP 实例 AJAX RSS 阅读器 □

PHP 实例 - AJAX RSS 阅读器

RSS 阅读器用于阅读 RSS Feed。

AJAX RSS 阅读器

在下面的实例中，我们将演示一个 RSS 阅读器，通过它，来自 RSS 的内容在网页不进行刷新的情况下被载入：

选择一个 RSS-feed:

RSS-feed 数据列表...

实例解释 - HTML 页面

当用户在上面的下拉列表中选择某个 RSS-feed 时，会执行名为 "showRSS()" 的函数。该函数由 "onchange" 事件触发：

实例

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<script>
function showRSS(str)
{
if (str.length==0)
{
document.getElementById("rssOutput").innerHTML="";
return;
}
if (window.XMLHttpRequest)
{
// IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
xmlhttp=new XMLHttpRequest();
}
else
{
// IE6, IE5 浏览器执行代码
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
{
document.getElementById("rssOutput").innerHTML=xmlhttp.responseText;
}
}
xmlhttp.open("GET","getrss.php?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>
<form>
<select onchange="showRSS(this.value)">
<option value="">选择一个 RSS-feed:</option>
<option value="rss">读取 RSS 数据</option>
</select>
</form>
```

```
<br>
<div id="rssOutput">RSS-feed 数据列表...</div>
</body>
</html>
```

showRSS() 函数会执行以下步骤：

检查是否有 RSS-feed 被选择

创建 XMLHttpRequest 对象

创建在服务器响应就绪时执行的函数

向服务器上的文件发送请求

请注意添加到 URL 末端的参数（q）（包含下拉列表的内容）

PHP 文件

文件 rss_demo.xml。

上面这段通过 JavaScript 调用的服务器页面是名为 "getrss.php" 的 PHP 文件：

实例

```
<?php
// rss 文件
$xml="rss_demo.xml";
$xmlDoc = new DOMDocument();
$xmlDoc->load($xml);
// 从 "<channel>" 中读取元素
$channel=$xmlDoc->getElementsByTagName('channel')->item(0);
$channel_title = $channel->getElementsByTagName('title')
->item(0)->childNodes->item(0)->nodeValue;
$channel_link = $channel->getElementsByTagName('link')
->item(0)->childNodes->item(0)->nodeValue;
$channel_desc = $channel->getElementsByTagName('description')
->item(0)->childNodes->item(0)->nodeValue;
// 输出 "<channel>" 中的元素
echo("<p><a href='" . $channel_link
. "'>" . $channel_title . "</a>");
echo("<br>");
echo($channel_desc . "</p>");
// 输出 "<item>" 中的元素
$x=$xmlDoc->getElementsByTagName('item');
for ($i=0; $i<=1; $i++) {
$item_title=$x->item($i)->getElementsByTagName('title')
->item(0)->childNodes->item(0)->nodeValue;
$item_link=$x->item($i)->getElementsByTagName('link')
->item(0)->childNodes->item(0)->nodeValue;
$item_desc=$x->item($i)->getElementsByTagName('description')
->item(0)->childNodes->item(0)->nodeValue;
echo ("<p><a href='" . $item_link
. "'>" . $item_title . "</a>");
echo ("<br>");
echo ($item_desc . "</p>");
}
?>
```

当 RSS feed 的请求从 JavaScript 发送到 PHP 文件时，将发生：

检查哪个 RSS feed 被选中

创建一个新的 XML DOM 对象

在 xml 变量中加载 RSS 文档

从 channel 元素中提取并输出元素

从 item 元素中提取并输出元素

PHP 实例 - AJAX 投票

AJAX 投票

在下面的实例中，我们将演示一个投票程序，通过它，投票结果在网页不进行刷新的情况下被显示。

你喜欢 PHP 和 AJAX 吗?

是:
否:

实例解释 - HTML 页面

当用户选择上面的某个选项时，会执行名为 "getVote()" 的函数。该函数由 "onclick" 事件触发。

poll.html 文件代码如下：

```
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
<script>
function getVote(int) {
  if (window.XMLHttpRequest) {
    // IE7+, Firefox, Chrome, Opera, Safari 执行代码
    xmlhttp=new XMLHttpRequest();
  } else {
    // IE6, IE5 执行代码
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
```

```
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
      document.getElementById("poll").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","poll_vote.php?vote="+int,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<div id="poll">
<h3>你喜欢 PHP 和 AJAX 吗?</h3>
<form>
是:
<input type="radio" name="vote" value="0" onclick="getVote(this.value)">
<br>否:
<input type="radio" name="vote" value="1" onclick="getVote(this.value)">
</form>
</div>

</body>
</html>
```

getVote() 函数会执行以下步骤：

- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含下拉列表的内容）

PHP 文件

上面这段通过 JavaScript 调用的服务器页面是名为 **"poll_vote.php"** 的 PHP 文件：

```
<?php
```

```
$vote = htmlspecialchars($_REQUEST['vote']);



// 获取文件中存储的数据

$filename = "poll_result.txt";

$content = file($filename);



// 将数据分割到数组中

$array = explode("||", $content[0]);

$yes = $array[0];

$no = $array[1];



if ($vote == 0)

{

  $yes = $yes + 1;

}



if ($vote == 1)

{

  $no = $no + 1;

}



// 插入投票数据

$insertvote = $yes."||".$no;

$fp = fopen($filename,"w");

fputs($fp,$insertvote);

fclose($fp);

?>



<h2>结果:</h2>

<table>

  <tr>

  <td>是:</td>

  <td>
```

```
  <span style="display: inline-block; background-color:green;
      width:<?php echo(100*round($yes/($no+$yes),2)); ?>px;
      height:20px;" ></span>
  <?php echo(100*round($yes/($no+$yes),2)); ?>%
  </td>
  </tr>
  <tr>
  <td>否:</td>
  <td>
  <span style="display: inline-block; background-color:red;
      width:<?php echo(100*round($no/($no+$yes),2)); ?>px;
      height:20px;"></span>
  <?php echo(100*round($no/($no+$yes),2)); ?>%
  </td>
  </tr>
</table>
```

当所选的值从 JavaScript 发送到 PHP 文件时，将发生：

1. 获取 "poll_result.txt" 文件的内容
2. 把文件内容放入变量，并向被选变量累加 1
3. 把结果写入 "poll_result.txt" 文件
4. 输出图形化的投票结果

文本文件

文本文件（**poll_result.txt**）中存储来自投票程序的数据。

它存储的数据如下所示：

```
3||4
```

第一个数字表示 "Yes" 的投票数，第二个数字表示 "No" 的投票数。

注释：请记得只允许您的 Web 服务器来编辑该文本文件。**不要**让其他人获得访问权，除了 Web 服务器 (PHP)。

□ PHP 实例 AJAX RSS 阅读器　　PHP 5 Timezones □

```
//无限循环脚本

var Vote = 0;//你的票。

setInterval(function(){

    getVote(Vote);

},2000);
```

怎样更安全？可以用 Cookies 记录下投票，这样攻击者还需清理 Cookies。

```
if(empty($_COOKIE["voted"])) {

    setcookie("voted","yes!",ime()+60*60*24*365);

} else {

    die("您已经投过票！");

}
```

学神之女5个月前 (05-08)

反馈/建议

PHP 5 Array 函数

PHP Array 简介

PHP Array 函数允许您访问并操作数组。

支持简单的数组和多维数组。

安装

PHP Array 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP 5 Array 函数

函数	描述
array()	创建数组。
array_change_key_case()	返回其键均为大写或小写的数组。
array_chunk()	把一个数组分割为新的数组块。
array_column()	返回输入数组中某个单一列的值。

array_combine()	通过合并两个数组（一个为键名数组，一个为键值数组）来创建一个新数组。
array_count_values()	用于统计数组中所有值出现的次数。
array_diff()	比较数组，返回两个数组的差集（只比较键值）。
array_diff_assoc()	比较数组，返回两个数组的差集（比较键名和键值）。
array_diff_key()	比较数组，返回两个数组的差集（只比较键名）。
array_diff_uassoc()	比较数组，返回两个数组的差集（比较键名和键值，使用用户自定义的键名比较函数）。
array_diff_ukey()	比较数组，返回两个数组的差集（只比较键名，使用用户自定义的键名比较函数）。
array_fill()	用给定的键值填充数组。
array_fill_keys()	用给定的指定键名的键值填充数组。
array_filter()	用回调函数过滤数组中的元素。
array_flip()	反转/交换数组中的键名和对应关联的键值。
array_intersect()	比较数组，返回两个数组的交集（只比较键值）。
array_intersect_assoc()	比较数组，返回两个数组的交集（比较键名和键值）。
array_intersect_key()	比较数组，返回两个数组的交集（只比较键名）。
array_intersect_uassoc()	比较数组，返回两个数组的交集（比较键名和键值，使用用户自定义的键名比较函数）。
array_intersect_ukey()	比较数组，返回两个数组的交集（只比较键名，使用用户自定义的键名比较函数）。
array_key_exists()	检查指定的键名是否存在于数组中。
array_keys()	返回数组中所有的键名。
array_map()	将用户自定义函数作用到给定数组的每个值上，返回新的值。
array_merge()	把一个或多个数组合并为一个数组。
array_merge_recursive()	递归地把一个或多个数组合并为一个数组。
array_multisort()	对多个数组或多维数组进行排序。
array_pad()	将指定数量的带有指定值的元素插入到数组中。
array_pop()	删除数组中的最后一个元素（出栈）。
array_product()	计算数组中所有值的乘积。
array_push()	将一个或多个元素插入数组的末尾（入栈）。
array_rand()	从数组中随机选出一个或多个元素，返回键名。
array_reduce()	通过使用用户自定义函数，迭代地将数组简化为一个字符串，并返回。
array_replace()	使用后面数组的值替换第一个数组的值。
array_replace_recursive()	递归地使用后面数组的值替换第一个数组的值。
array_reverse()	将原数组中的元素顺序翻转，创建新的数组并返回。

array_search()	在数组中搜索给定的值，如果成功则返回相应的键名。
array_shift()	删除数组中的第一个元素，并返回被删除元素的值。
array_slice()	返回数组中的选定部分。
array_splice()	把数组中的指定元素去掉并用其它值取代。
array_sum()	返回数组中所有值的和。
array_udiff()	比较数组，返回两个数组的差集（只比较键值，使用一个用户自定义的键名比较函数）。
array_udiff_assoc()	比较数组，返回两个数组的差集（比较键名和键值，使用内建函数比较键名，使用用户自定义函数比较键值）。
array_udiff_uassoc()	比较数组，返回两个数组的差集（比较键名和键值，使用两个用户自定义的键名比较函数）。
array_uintersect()	比较数组，返回两个数组的交集（只比较键值，使用一个用户自定义的键名比较函数）。
array_uintersect_assoc()	比较数组，返回两个数组的交集（比较键名和键值，使用内建函数比较键名，使用用户自定义函数比较键值）。
array_uintersect_uassoc()	比较数组，返回两个数组的交集（比较键名和键值，使用两个用户自定义的键名比较函数）。
array_unique()	删除数组中重复的值。
array_unshift()	在数组开头插入一个或多个元素。
array_values()	返回数组中所有的值。
array_walk()	对数组中的每个成员应用用户函数。
array_walk_recursive()	对数组中的每个成员递归地应用用户函数。
arsort()	对关联数组按照键值进行降序排序。
asort()	对关联数组按照键值进行升序排序。
compact()	创建一个包含变量名和它们的值的数组。
count()	返回数组中元素的数目。
current()	返回数组中的当前元素。
each()	返回数组中当前的键 / 值对。
end()	将数组的内部指针指向最后一个元素。
extract()	从数组中将变量导入到当前的符号表。
in_array()	检查数组中是否存在指定的值。
key()	从关联数组中取得键名。
krsort()	对关联数组按照键名降序排序。
ksort()	对关联数组按照键名升序排序。
list()	把数组中的值赋给一些数组变量。
natcasesort()	用"自然排序"算法对数组进行不区分大小写字母的排序。

natsort()	用"自然排序"算法对数组排序。
next()	将数组中的内部指针向后移动一位。
pos()	current() 的别名。
prev()	将数组的内部指针倒回一位。
range()	创建一个包含指定范围的元素的数组。
reset()	将数组的内部指针指向第一个元素。
rsort()	对数值数组进行降序排序。
shuffle()	把数组中的元素按随机顺序重新排列。
sizeof()	count() 的别名。
sort()	对数值数组进行升序排序。
uasort()	使用用户自定义的比较函数对数组中的键值进行排序。
uksort()	使用用户自定义的比较函数对数组中的键名进行排序。
usort()	使用用户自定义的比较函数对数组进行排序。

PHP 5 Calendar 函数 □

□ PHP 5 Array 函数 PHP 5 Date/Time 函数 □

PHP 5 Calendar 函数

PHP Calendar 简介

日历扩展包含了简化不同日历格式间的转换的函数。

它是基于 Julian Day Count（儒略日计数），是从公元前 4713 年 1 月 1 日开始计算的。

注释：如需在日历格式之间转换，必须首先转换为 Julian Day Count，然后再转换为您需要的日历格式。

注释：Julian Day Count（儒略日计数）与 Julian Calendar（儒略历法） 不是一回事！

安装

为了让这些函数能够工作，您必须通过 –enable-calendar 编译 PHP。

PHP 的 Windows 版本已内建了对日历扩展的支持。因此，Calendar 函数会自动工作。

PHP 5 Calendar 函数

函数	描述
cal_days_in_month()	针对指定的年份和历法，返回一个月中的天数。
cal_from_jd()	把儒略日计数转换为指定历法的日期。
cal_info()	返回有关指定历法的信息。
cal_to_jd()	把指定历法的日期转换为儒略日计数。
easter_date()	返回指定年份的复活节午夜的 Unix 时间戳。
easter_days()	返回指定年份的复活节与 3 月 21 日之间的天数。
frenchtojd()	把法国共和历法的日期转换成为儒略日计数。
gregoriantojd()	把格利高里历法的日期转换成为儒略日计数。
jddayofweek()	返回日期在周几。
jdmonthname()	返回月的名称。
jdtofrench()	把儒略日计数转换为法国共和历法的日期。
jdtogregorian()	把儒略日计数转换为格利高里历法的日期。
jdtojewish()	把儒略日计数转换为犹太历法的日期。
jdtojulian()	把儒略日计数转换为儒略历法的日期。
jdtounix()	把儒略日计数转换为 Unix 时间戳。
jewishtojd()	把犹太历法的日期转换为儒略日计数。
juliantojd()	把儒略历法的日期转换为儒略日计数。
unixtojd()	把 Unix 时间戳转换为儒略日计数。

PHP 5 预定义的 Calendar 常量

常量	类型	PHP 版本
CAL_GREGORIAN	Integer	PHP 4
CAL_JULIAN	Integer	PHP 4
CAL_JEWISH	Integer	PHP 4
CAL_FRENCH	Integer	PHP 4
CAL_NUM_CALS	Integer	PHP 4
CAL_DOW_DAYNO	Integer	PHP 4
CAL_DOW_SHORT	Integer	PHP 4
CAL_DOW_LONG	Integer	PHP 4
CAL_MONTH_GREGORIAN_SHORT	Integer	PHP 4
CAL_MONTH_GREGORIAN_LONG	Integer	PHP 4

CAL_MONTH_JULIAN_SHORT	Integer	PHP 4
CAL_MONTH_JULIAN_LONG	Integer	PHP 4
CAL_MONTH_JEWISH	Integer	PHP 4
CAL_MONTH_FRENCH	Integer	PHP 4
CAL_EASTER_DEFAULT	Integer	PHP 4.3
CAL_EASTER_ROMAN	Integer	PHP 4.3
CAL_EASTER_ALWAYS_GREGORIAN	Integer	PHP 4.3
CAL_EASTER_ALWAYS_JULIAN	Integer	PHP 4.3
CAL_JEWISH_ADD_ALAFIM_GERESH	Integer	PHP 5.0
CAL_JEWISH_ADD_ALAFIM	Integer	PHP 5.0
CAL_JEWISH_ADD_GERESHAYIM	Integer	PHP 5.0

PHP 5 Array 函数 PHP 5 Date/Time 函数

PHP curl_init函数 PHP curl_multi_add_handle函数

PHP cURL 函数

概述

PHP支持的由Daniel Stenberg创建的libcurl库允许你与各种的服务器使用各种类型的协议进行连接和通讯。

libcurl目前支持http、https、ftp、gopher、telnet、dict、file和ldap协议。libcurl同时也支持HTTPS认证、HTTP POST、HTTP PUT、 FTP 上传(这个也能通过PHP的FTP扩展完成)、HTTP 基于表单的上传、代理、cookies和用户名+密码的认证。

PHP中使用cURL实现Get和Post请求的方法

这些函数在PHP 4.0.2中被引入。

需求

为了使用PHP的cURL函数，你需要安装 » libcurl包。

PHP需要使用libcurl 7.0.2-beta 或者更高版本。在PHP 4.2.3 里使用cURL，你需要安装7.9.0或更高版本的libcurl。从PHP 4.3.0开始你需要安装7.9.0或更高版本的libcurl。从PHP 5.0.0开始你需要安装7.10.5或更高版本的libcurl。

安装

要使用PHP的cURL支持你必须在编译PHP时加上--with-curl[=DIR] 选项，DIR为包含lib和include的目录路径。在include目录中必须有一个名为curl，包含了easy.h和curl.h的文件夹。lib文件夹里应该有一个名为libcurl.a的文件。对于PHP 4.3.0你可以配置--with-curlwrappers 使cURL使用URL流。

注意: Win32用户注意 要在Windows环境下使用这个模块，libeay32.dll和ssleay32.dll必须放到PATH环境变量包含的目录下。 不用cURL网站上的libcurl.dll。

资源类型

这个扩展定义了2中资源：cURL句柄和cURL批处理句柄。

PHP cURL 函数

以下包含了PHP cURL函数列表：

函数	描述
curl_close()	关闭一个cURL会话。
curl_copy_handle()	复制一个cURL句柄和它的所有选项。
curl_errno()	返回最后一次的错误号。
curl_error()	返回一个保护当前会话最近一次错误的字符串。
curl_escape()	返回转义字符串，对给定的字符串进行URL编码。
curl_exec()	执行一个cURL会话。
curl_file_create()	创建一个 CURLFile 对象。
curl_getinfo()	获取一个cURL连接资源句柄的信息。
curl_init()	初始化一个cURL会话。
curl_multi_add_handle()	向curl批处理会话中添加单独的curl句柄。
curl_multi_close()	关闭一组cURL句柄。
curl_multi_exec()	运行当前 cURL 句柄的子连接。
curl_multi_getcontent()	如果设置了CURLOPT_RETURNTRANSFER，则返回获取的输出的文本流。
curl_multi_info_read()	获取当前解析的cURL的相关传输信息。
curl_multi_init()	返回一个新cURL批处理句柄。
curl_multi_remove_handle()	移除curl批处理句柄资源中的某个句柄资源。
curl_multi_select()	等待所有cURL批处理中的活动连接。
curl_multi_setopt()	设置一个批处理cURL传输选项。
curl_multi_strerror()	返回描述错误码的字符串文本。
curl_pause()	暂停及恢复连接。
curl_reset()	重置libcurl的会话句柄的所有选项。
curl_setopt_array()	为cURL传输会话批量设置选项。
curl_setopt()	设置一个cURL传输选项。

curl_share_close()	关闭cURL共享句柄。
curl_share_init()	初始化cURL共享句柄。
curl_share_setopt()	设置一个共享句柄的cURL传输选项。
curl_strerror()	返回错误代码的字符串描述。
curl_unescape()	解码URL编码后的字符串。
curl_version()	获取cURL版本信息。

PHP curl_init函数 PHP curl_multi_add_handle函数

点我分享笔记

反馈/建议

PHP 5 Date/Time 函数

PHP Date/Time 简介

Date/Time 函数允许您从 PHP 脚本运行的服务器上获取日期和时间。您可以使用 Date/Time 函数通过不同的方式来格式化日期和时间。

注释：这些函数依赖于服务器的本地设置。使用这些函数时请记住要考虑夏令时和闰年。

安装

PHP Date/Time 函数是PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

Date/Time 函数的行为受到 php.ini 中设置的影响：

名称	描述	默认	PHP 版本
date.timezone	规定默认时区（所有的 Date/Time 函数使用该选项）	""	PHP 5.1
date.default_latitude	规定默认纬度（date_sunrise() 和 date_sunset() 使用该选项）	"31.7667"	PHP 5.0
date.default_longitude	规定默认经度（date_sunrise() 和 date_sunset() 使用该选项）	"35.2333"	PHP 5.0
date.sunrise_zenith	规定默认日出天顶（date_sunrise() 和 date_sunset() 使用该选项）	"90.83"	PHP 5.0
date.sunset_zenith	规定默认日落天顶（date_sunrise() 和 date_sunset() 使用该选项）	"90.83"	PHP 5.0

PHP 5 Date/Time 函数

函数	描述
checkdate()	验证格利高里日期。
date_add()	添加日、月、年、时、分和秒到一个日期。
date_create_from_format()	返回一个根据指定格式进行格式化的新的 DateTime 对象。
date_create()	返回一个新的 DateTime 对象。
date_date_set()	设置一个新的日期。
date_default_timezone_get()	返回默认时区，被所有的 Date/Time 函数使用。
date_default_timezone_set()	设置默认时区，被所有的 Date/Time 函数使用。
date_diff()	返回两个日期间的差值。
date_format()	返回根据指定格式进行格式化的日期。
date_get_last_errors()	返回日期字符串中的警告/错误。
date_interval_create_from_date_string()	从字符串的相关部分建立一个 DateInterval。
date_interval_format()	格式化时间间隔。
date_isodate_set()	设置 ISO 日期。
date_modify()	修改时间戳。
date_offset_get()	返回时区偏移。
date_parse_from_format()	根据指定的格式返回一个带有指定日期的详细信息的关联数组。
date_parse()	返回一个带有指定日期的详细信息的关联数组。
date_sub()	从指定日期减去日、月、年、时、分和秒。
date_sun_info()	返回一个包含有关指定日期与地点的日出/日落和黄昏开始/黄昏结束的信息的数组。
date_sunrise()	返回指定日期与地点的日出时间。
date_sunset()	返回指定日期与地点的日落时间。
date_time_set()	设置时间。
date_timestamp_get()	返回 Unix 时间戳。
date_timestamp_set()	设置基于 Unix 时间戳的日期和时间。
date_timezone_get()	返回给定 DateTime 对象的时区。
date_timezone_set()	设置 DateTime 对象的时区。
date()	格式化本地日期和时间。
getdate()	返回某个时间戳或者当前本地的日期/时间的日期/时间信息。
gettimeofday()	返回当前时间。
gmdate()	格式化 GMT/UTC 日期和时间。
gmmktime()	返回 GMT 日期的 UNIX 时间戳。

gmstrftime()	根据区域设置格式化 GMT/UTC 日期和时间。
idate()	格式化本地时间/日期为整数。
localtime()	返回本地时间。
microtime()	返回当前 Unix 时间戳的微秒数。
mktime()	返回一个日期的 Unix 时间戳。
strftime()	根据区域设置格式化本地时间/日期。
strptime()	解析由 strftime() 生成的时间/日期。
strtotime()	将任何英文文本的日期或时间描述解析为 Unix 时间戳。
time()	返回当前时间的 Unix 时间戳。
timezone_abbreviations_list()	返回包含夏令时、偏移量和时区名称的关联数组。
timezone_identifiers_list()	返回带有所有时区标识符的数值数组。
timezone_location_get()	返回指定时区的位置信息。
timezone_name_from_abbr()	根据时区缩略语返回时区名称。
timezone_name_get()	返回时区的名称。
timezone_offset_get()	返回相对于 GMT 的时区偏移。
timezone_open()	创建一个新的 DateTimeZone 对象。
timezone_transitions_get()	返回时区的所有转换。
timezone_version_get()	返回时区数据库的版本。

PHP 5 预定义的 Date/Time 常量

常量	描述
DATE_ATOM	Atom (例如：2005-08-15T16:13:03+0000)
DATE_COOKIE	HTTP Cookies (例如：Sun, 14 Aug 2005 16:13:03 UTC)
DATE_ISO8601	ISO-8601 (例如：2005-08-14T16:13:03+0000)
DATE_RFC822	RFC 822 (例如：Sun, 14 Aug 2005 16:13:03 UTC)
DATE_RFC850	RFC 850 (例如：Sunday, 14-Aug-05 16:13:03 UTC)
DATE_RFC1036	RFC 1036 (例如：Sunday, 14-Aug-05 16:13:03 UTC)
DATE_RFC1123	RFC 1123 (例如：Sun, 14 Aug 2005 16:13:03 UTC)
DATE_RFC2822	RFC 2822 (例如：Sun, 14 Aug 2005 16:13:03 +0000)
DATE_RSS	RSS (例如：Sun, 14 Aug 2005 16:13:03 UTC)
DATE_W3C	万维网联盟 (例如：2005-08-14T16:13:03+0000)

PHP 5 Directory 函数

PHP Directory 简介

Directory 函数允许您获得关于目录及其内容的信息。

安装

PHP Directory 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP 5 Directory 函数

函数	描述
chdir()	改变当前的目录。
chroot()	改变根目录。
closedir()	关闭目录句柄。
dir()	返回 Directory 类的实例。
getcwd()	返回当前工作目录。
opendir()	打开目录句柄。
readdir()	返回目录句柄中的条目。
rewinddir()	重置目录句柄。
scandir()	返回指定目录中的文件和目录的数组。

PHP 5 Date/Time 函数 PHP Error 和 Logging 函数

点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

PHP Error 和 Logging 函数

PHP Error 和 Logging 简介

Error 和 Logging 函数允许您对错误进行处理和记录。

Error 函数允许用户定义错误处理规则，并修改记录错误的方式。

Logging 函数允许用户对应用程序进行日志记录，并把日志消息发送到电子邮件、系统日志或其他的机器。

执行配置

error 函数受 php.ini 配置文件影响。

错误和日志配置选项：

参数	默认值	描述	可修改范围
error_reporting	NULL	设置 PHP 的报错级别并返回当前级别(数字或常量)。	PHP_INI_ALL
display_errors	"1"	该选项设置是否将错误信息作为输出的一部分显示到屏幕，或者对用户隐藏而不显示。 **注意：** 该特性不要在上线生产环境中使用 (在开发测试过程中使用)	PHP_INI_ALL
display_startup_errors	"0"	即使 display_errors 设置为开启, PHP 启动过程中的错误信息也不会被显示。强烈建议除了调试目的以外，将 display_startup_errors 设置为关闭。	PHP_INI_ALL
log_errors	"0"	设置是否将脚本运行的错误信息记录到服务器错误日志或者error_log之中。注意，这是与服务器相关的特定配置项。	PHP_INI_ALL
log_errors_max_len	"1024"	设置 log_errors 的最大字节数. 在 error_log 会添加有关错误源的信息。默认值为1024，如果设置为0表示不限长度。该长度设置对记录的错误，显示的错误，以及 $php_errormsg都会有限制作用。	PHP_INI_ALL
ignore_repeated_errors	"0"	不记录重复的信息。重复的错误必须出现在同一个文件中的同一行代码上，除非 ignore_repeated_source 设置为true。	PHP_INI_ALL
ignore_repeated_source	"0"	忽略重复消息时，也忽略消息的来源。当该设置开启时，重复信息将不会记录它是由不同的文件还是不同的源代码行产生的。	PHP_INI_ALL
report_memleaks	"1"	如果这个参数设置为Off，则内存泄露信息不会显示 (在 stdout 或者日志中)。	PHP_INI_ALL
track_errors	"0"	如果开启，最后的一个错误将永远存在于变量 $php_errormsg 中。	PHP_INI_ALL
html_errors	"1"	在错误信息中关闭HTML标签。	PHP_INI_ALL PHP_INI_SYSTEM in PHP <= 4.2.3.
xmlrpc_errors	"0"	关闭正常的错误报告，并将错误的格式设置为XML-RPC错误信息的格式。	PHP_INI_SYSTEM

xmlrpc_error_number	"0"	用作 XML-RPC faultCode 元素的值。	PHP_INI_ALL
docref_root	""	新的错误信息格式包含了对应的参考页面，该页面对错误进行具体描述，或者描述了导致该错误发生的函数。 为了提供手册的页面，你可以在PHP官方站点下载对应语言的手册，并在ini中设置网址到本地对应的地址。 如果你的本地手册拷贝可以使用"/manual/" 访问，你就可以简单的设置 docref_root=/manual/。 另外你还需要设置 docref_ext 匹配你本地文件的后缀名 docref_ext=.html。当然也可以设置一个外部的参考地址。 例如你可以设置 docref_root=http://manual/en/ 或者 docref_root="http://landonize.it/?how=url&theme=classic&filter=Landon&url=http%3A%2F%2Fwww.php.net%2F"	PHP_INI_ALL
docref_ext	""	参见 docref_root.	PHP_INI_ALL
error_prepend_string	NULL	错误信息之前输出的内容。	PHP_INI_ALL
error_append_string	NULL	错误信息之后输出的内容。	PHP_INI_ALL
error_log	NULL	设置脚本错误将被记录到的文件。该文件必须是web服务器用户可写的。	PHP_INI_ALL

安装

Error 和 Logging 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Error 和 Logging 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
debug_backtrace()	生成 backtrace。	4
debug_print_backtrace()	打印 backtrace。	5
error_get_last()	获得最后发生的错误。	5
error_log()	向服务器错误记录、文件或远程目标发送一个错误。	4
error_reporting()	规定报告哪个错误。	4
restore_error_handler()	恢复之前的错误处理程序。	4
restore_exception_handler()	恢复之前的异常处理程序。	5
set_error_handler()	设置用户自定义的错误处理函数。	4
set_exception_handler()	设置用户自定义的异常处理函数。	5
trigger_error()	创建用户自定义的错误消息。	4
user_error()	trigger_error() 的别名。	4

PHP Error 和 Logging 常量

PHP：指示支持该常量的最早的 PHP 版本。

值	常量	描述	PHP
1	E_ERROR	运行时致命的错误。不能修复的错误。停止执行脚本。	
2	E_WARNING	运行时非致命的错误。没有停止执行脚本。	
4	E_PARSE	编译时的解析错误。解析错误应该只由解析器生成。	
8	E_NOTICE	运行时的通知。脚本发现可能是一个错误，但也可能在正常运行脚本时发生。	
16	E_CORE_ERROR	PHP 启动时的致命错误。这就如同 PHP 核心的 E_ERROR。	4
32	E_CORE_WARNING	PHP 启动时的非致命错误。这就如同 PHP 核心的 E_WARNING。	4
64	E_COMPILE_ERROR	编译时致命的错误。这就如同由 Zend 脚本引擎生成的 E_ERROR。	4
128	E_COMPILE_WARNING	编译时非致命的错误。这就如同由 Zend 脚本引擎生成的 E_WARNING。	4
256	E_USER_ERROR	用户生成的致命错误。这就如同由程序员使用 PHP 函数 trigger_error() 生成的 E_ERROR。	4
512	E_USER_WARNING	用户生成的非致命错误。这就如同由程序员使用 PHP 函数 trigger_error() 生成的 E_WARNING。	4
1024	E_USER_NOTICE	用户生成的通知。这就如同由程序员使用 PHP 函数 trigger_error() 生成的 E_NOTICE。	4
2048	E_STRICT	运行时的通知。PHP 建议您改变代码，以提高代码的互用性和兼容性。	5
4096	E_RECOVERABLE_ERROR	可捕获的致命错误。这就如同一个可以由用户定义的句柄捕获的 E_ERROR（见 set_error_handler()）。	5
6143	E_ALL	所有的错误和警告的级别，除了 E_STRICT（自 PHP 6.0 起，E_STRICT 将作为 E_ALL的一部分）。	5

□ PHP 5 Directory 函数　　PHP 5 Filesystem 函数 □

□ PHP Error 和 Logging 函数　　PHP Filter 函数 □

PHP 5 Filesystem 函数

PHP Filesystem 简介

Filesystem 函数允许您访问和操作文件系统。

安装

Filesystem 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

Filesystem 函数的行为受到 php.ini 中设置的影响。

Filesystem 配置选项：

名称	默认	描述	可改变
allow_url_fopen	"1"	允许 fopen()-type 函数使用 URL。（PHP 4.0.4 版以后可用）	PHP_INI_SYSTEM
user_agent	NULL	定义 PHP 发送的用户代理。（PHP 4.3 版以后可用）	PHP_INI_ALL
default_socket_timeout	"60"	设置基于 socket 流的默认的超时时间（秒）。（PHP 4.3 版以后可用）	PHP_INI_ALL
from	""	定义匿名 FTP 的密码（您的 email 地址）。	PHP_INI_ALL
auto_detect_line_endings	"0"	当设置为 "1" 时，PHP 将检查通过 fgets() 和 file() 取得的数据中的行结束符号是符合 Unix、MS-Dos 还是 Mac 的习惯。（PHP 4.3 版以后可用）	PHP_INI_ALL

Unix / Windows 兼容性

当在 Unix 平台上规定路径时，正斜杠 (/) 用作目录分隔符。而在 Windows 平台上，正斜杠 (/) 和反斜杠 (\) 均可使用。

PHP 5 Filesystem 函数

函数	描述
basename()	返回路径中的文件名部分。
chgrp()	改变文件组。
chmod()	改变文件模式。
chown()	改变文件所有者。
clearstatcache()	清除文件状态缓存。
copy()	复制文件。
delete()	参见 unlink() 或 unset()
dirname()	返回路径中的目录名称部分。
disk_free_space()	返回目录的可用空间。
disk_total_space()	返回一个目录的磁盘总容量。
diskfreespace()	disk_free_space() 的别名。
fclose()	关闭打开的文件。
feof()	测试文件指针是否到了文件末尾。
fflush()	向打开的文件刷新缓冲输出。
fgetc()	从打开的文件中返回字符。

fgetcsv()	从打开的文件中解析一行，校验 CSV 字段。
fgets()	从打开的文件中返回一行。
fgetss()	从打开的文件中返回一行，并过滤掉 HTML 和 PHP 标签。
file()	把文件读入一个数组中。
file_exists()	检查文件或目录是否存在。
file_get_contents()	把文件读入字符串。
file_put_contents()	把字符串写入文件。
fileatime()	返回文件的上次访问时间。
filectime()	返回文件的上次修改时间。
filegroup()	返回文件的组 ID。
fileinode()	返回文件的 inode 编号。
filemtime()	返回文件内容的上次修改时间。
fileowner()	返回文件的用户 ID （所有者）。
fileperms()	返回文件的权限。
filesize()	返回文件大小。
filetype()	返回文件类型。
flock()	锁定或释放文件。
fnmatch()	根据指定的模式来匹配文件名或字符串。
fopen()	打开一个文件或 URL。
fpassthru()	从打开的文件中读数据，直到文件末尾（EOF），并向输出缓冲写结果。
fputcsv()	把行格式化为 CSV 并写入一个打开的文件中。
fputs()	fwrite() 的别名。
fread()	读取打开的文件。
fscanf()	根据指定的格式对输入进行解析。
fseek()	在打开的文件中定位。
fstat()	返回关于一个打开的文件的信息。
ftell()	返回在打开文件中的当前位置。
ftruncate()	把打开文件截断到指定的长度。
fwrite()	写入打开的文件。
glob()	返回一个包含匹配指定模式的文件名/目录的数组。
is_dir()	判断文件是否是一个目录。

is_executable()	判断文件是否可执行。
is_file()	判断文件是否是常规的文件。
is_link()	判断文件是否是连接。
is_readable()	判断文件是否可读。
is_uploaded_file()	判断文件是否是通过 HTTP POST 上传的。
is_writable()	判断文件是否可写。
is_writeable()	is_writable() 的别名。
lchgrp()	改变符号连接的组所有权。
lchown()	改变符号连接的用户所有权。
link()	创建一个硬连接。
linkinfo()	返回有关一个硬连接的信息。
lstat()	返回关于文件或符号连接的信息。
mkdir()	创建目录。
move_uploaded_file()	把上传的文件移动到新位置。
parse_ini_file()	解析一个配置文件。
parse_ini_string()	解析一个配置字符串。
pathinfo()	返回关于文件路径的信息。
pclose()	关闭由 popen() 打开的进程。
popen()	打开一个进程。
readfile()	读取一个文件，并写入到输出缓冲。
readlink()	返回符号连接的目标。
realpath()	返回绝对路径名。
realpath_cache_get()	返回高速缓存条目。
realpath_cache_size()	返回高速缓存大小。
rename()	重命名文件或目录。
rewind()	倒回文件指针的位置。
rmdir()	删除空的目录。
set_file_buffer()	设置已打开文件的缓冲大小。
stat()	返回关于文件的信息。
symlink()	创建符号连接。
tempnam()	创建唯一的临时文件。

tmpfile()	创建唯一的临时文件。
touch()	设置文件的访问和修改时间。
umask()	改变文件的文件权限。
unlink()	删除文件。

□ PHP Error 和 Logging 函数 PHP Filter 函数 □

PHP Filter 函数

PHP Filter 简介

PHP 过滤器用于对来自非安全来源的数据（比如用户输入）进行验证和过滤。

安装

Filter 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP Filter 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
filter_has_var()	检查是否存在指定输入类型的变量。	5
filter_id()	返回指定过滤器的 ID 号。	5
filter_input()	从脚本外部获取输入，并进行过滤。	5
filter_input_array()	从脚本外部获取多项输入，并进行过滤。	5
filter_list()	返回包含所有得到支持的过滤器的一个数组。	5
filter_var_array()	获取多个变量，并进行过滤。	5
filter_var()	获取一个变量，并进行过滤。	5

PHP Filters

ID 名称	描述
FILTER_CALLBACK	调用用户自定义函数来过滤数据。
FILTER_SANITIZE_STRING	去除标签，去除或编码特殊字符。
FILTER_SANITIZE_STRIPPED	"string" 过滤器的别名。
FILTER_SANITIZE_ENCODED	URL-encode 字符串，去除或编码特殊字符。
FILTER_SANITIZE_SPECIAL_CHARS	HTML 转义字符 '"<>& 以及 ASCII 值小于 32 的字符。
FILTER_SANITIZE_EMAIL	删除所有字符，除了字母、数字以及 !#$%&'*+-/=?^_`{|}~@.[]
FILTER_SANITIZE_URL	删除所有字符，除了字母、数字以及 $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=
FILTER_SANITIZE_NUMBER_INT	删除所有字符，除了数字和 +-
FILTER_SANITIZE_NUMBER_FLOAT	删除所有字符，除了数字、+- 以及 .,eE
FILTER_SANITIZE_MAGIC_QUOTES	应用 addslashes()。
FILTER_UNSAFE_RAW	不进行任何过滤，去除或编码特殊字符。
FILTER_VALIDATE_INT	把值作为整数来验证。
FILTER_VALIDATE_BOOLEAN	把值作为布尔选项来验证。如果是 "1"、"true"、"on" 和 "yes"，则返回 TRUE。如果是 "0"、"false"、"off"、"no" 和 ""，则返回 FALSE。否则返回 NULL。
FILTER_VALIDATE_FLOAT	把值作为浮点数来验证。
FILTER_VALIDATE_REGEXP	根据 regexp（一种兼容 Perl 的正则表达式）来验证值。
FILTER_VALIDATE_URL	把值作为 URL 来验证。
FILTER_VALIDATE_EMAIL	把值作为 e-mail 地址来验证。
FILTER_VALIDATE_IP	把值作为 IP 地址来验证，只限 IPv4 或 IPv6 或 不是来自私有或者保留的范围。

□ PHP 5 Filesystem 函数　　PHP FTP 函数 □

PHP FTP 函数

PHP FTP 简介

FTP 函数通过文件传输协议 (FTP) 提供对文件服务器的客户端访问。

FTP 函数用于打开、登录以及关闭连接，同时用于上传、下载、重命名、删除及获取文件服务器上的文件信息。不是所有的 FTP 函数对每个服务器都起作用或返回相同的结果。自 PHP 3 起，FTP 函数可用。

这些函数用于对 FTP 服务器进行细致的访问。如果您仅仅需要对 FTP 服务器进行读写操作，建议使用 Filesystem 函数中的 ftp:// wrapper。

安装

PHP 的 Windows 版本内置了对 FTP 扩展的支持。无需加载任何附加扩展库即可使用 FTP 函数。

然而，如果您运行的是 PHP 的 Linux 版本，在编译 PHP 的时候请添加 *--enable-ftp* 选项（PHP4 或以上版本）或者 *--with-ftp* 选项（PHP3 版本）。

PHP FTP 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
ftp_alloc()	为要上传到 FTP 服务器的文件分配空间。	5
ftp_cdup()	把当前目录改变为 FTP 服务器上的父目录。	3
ftp_chdir()	改变 FTP 服务器上的当前目录。	3
ftp_chmod()	通过 FTP 设置文件上的权限。	5
ftp_close()	关闭 FTP 连接。	4
ftp_connect()	打开 FTP 连接。	3
ftp_delete()	删除 FTP 服务器上的一个文件。	3
ftp_exec()	在 FTP 服务器上执行一个程序/命令。	4
ftp_fget()	从 FTP 服务器上下载一个文件并保存到本地一个已经打开的文件中。	3
ftp_fput()	上传一个已经打开的文件，并在 FTP 服务器上把它保存为一个文件。	3
ftp_get_option()	返回 FTP 连接的各种运行时选项。	4
ftp_get()	从 FTP 服务器上下载文件。	3
ftp_login()	登录 FTP 服务器。	3
ftp_mdtm()	返回指定文件的最后修改时间。	3
ftp_mkdir()	在 FTP 服务器上创建一个新目录。	3
ftp_nb_continue()	连续获取/发送文件。（无阻塞）	4
ftp_nb_fget()	从 FTP 服务器上下载一个文件并保存到本地一个已经打开的文件中。（无阻塞）	4
ftp_nb_fput()	上传一个已经打开的文件，并在 FTP 服务器上把它保存为一个文件。（无阻塞）	4
ftp_nb_get()	从 FTP 服务器上下载文件。（无阻塞）	4
ftp_nb_put()	把文件上传到 FTP 服务器上。（无阻塞）	4
ftp_nlist()	返回 FTP 服务器上指定目录的文件列表。	3

ftp_pasv()	把被动模式设置为打开或关闭。	3
ftp_put()	把文件上传到 FTP 服务器上。	3
ftp_pwd()	返回当前目录名称。	3
ftp_quit()	ftp_close() 的别名。	3
ftp_raw()	向 FTP 服务器发送一个 raw 命令。	5
ftp_rawlist()	返回指定目录中文件的详细列表。	3
ftp_rename()	重命名 FTP 服务器上的文件或目录。	3
ftp_rmdir()	删除 FTP 服务器上的一个目录。	3
ftp_set_option()	设置 FTP 连接的各种运行时选项。	4
ftp_site()	向服务器发送 SITE 命令。	3
ftp_size()	返回指定文件的大小。	3
ftp_ssl_connect()	打开一个安全的 SSL-FTP 连接。	4
ftp_systype()	返回 FTP 服务器的系统类型标识符。	3

PHP FTP 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
FTP_ASCII		3
FTP_TEXT		3
FTP_BINARY		3
FTP_IMAGE		3
FTP_TIMEOUT_SEC		3
FTP_AUTOSEEK		4
FTP_AUTORESUME	为 GET 和 PUT 请求自动决定恢复和开始的位置	4
FTP_FAILED	异步传输失败	4
FTP_FINISHED	异步传输成功	4
FTP_MOREDATA	异步传输是活动状态的	4

□ PHP Filter 函数　　PHP HTTP 函数 □

PHP HTTP 函数

PHP HTTP 简介

HTTP 函数允许您在其他输出被发送之前，对由 Web 服务器发送到浏览器的信息进行操作。

安装

HTTP 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP HTTP 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
header()	向客户端发送原始的 HTTP 报头。	3
headers_list()	返回已发送的（或待发送的）响应头部的一个列表。	5
headers_sent()	检查 HTTP 报头是否发送/已发送到何处。	3
setcookie()	向客户端发送一个 HTTP cookie。	3
setrawcookie()	不对 cookie 值进行 URL 编码，发送一个 HTTP cookie。	5

PHP HTTP 常量

无。

Copyright © 2013-2018 菜鸟教程 runoob.com All Rights Reserved. 备案号：闽ICP备15012807号-1

首页 HTML CSS JS 本地书签

PHP Libxml 函数

PHP Libxml 简介

Libxml 函数和常量与 SimpleXML、XSLT 以及 DOM 函数一起使用。

安装

这些函数需要 Libxml 程序包。 在 xmlsoft.org 下载

PHP Libxml 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
libxml_clear_errors()	清空 Libxml 错误缓冲。	5
libxml_get_errors()	检索错误数组。	5
libxml_get_last_error()	从 Libxml 检索最后的错误。	5
libxml_set_streams_context()	为下一次 Libxml 文档加载或写入设置流环境。	5
libxml_use_internal_errors()	禁用 Libxml 错误，允许用户按需读取错误信息。	5

PHP Libxml 常量

函数	描述	PHP
LIBXML_COMPACT	设置小型节点分配优化。会改善应用程序的性能。	5
LIBXML_DTDATTR	设置默认 DTD 属性。	5
LIBXML_DTDLOAD	加载外部子集。	5
LIBXML_DTDVALID	通过 DTD 进行验证。	5
LIBXML_NOBLANKS	删除空节点。	5
LIBXML_NOCDATA	把 CDATA 设置为文本节点。	5
LIBXML_NOEMPTYTAG	更改空标签（比如 改为 </br>）。仅在 DOMDocument->save() 和 DOMDocument->saveXML() 函数中可用。	5
LIBXML_NOENT	替代实体。	5
LIBXML_NOERROR	不显示错误报告。	5
LIBXML_NONET	在加载文档时停止网络访问。	5
LIBXML_NOWARNING	不显示警告报告。	5
LIBXML_NOXMLDECL	在保存文档时，撤销 XML 声明。	5
LIBXML_NSCLEAN	删除额外的命名空间声明。	5
LIBXML_XINCLUDE	使用 XInclude 置换。	5
LIBXML_ERR_ERROR	获得可恢复的错误。	5
LIBXML_ERR_FATAL	获得致命的错误。	5

LIBXML_ERR_NONE	获得无错误。	5
LIBXML_ERR_WARNING	获得简单警告。	5
LIBXML_VERSION	获得 Libxml 版本（例如：20605 或 20617）Get libxml version (e.g. 20605 or 20617)	5
LIBXML_DOTTED_VERSION	获得有点号的 Libxml 版本（例如：2.6.5 或 2.6.17）。	5

□ PHP HTTP 函数 PHP Mail 函数 □

□ PHP Libxml 函数 PHP 5 Math 函数 □

PHP Mail 函数

PHP Mail 简介

mail() 函数允许您从脚本中直接发送电子邮件。

需求

要使邮件函数可用，PHP 需要已安装且正在运行的邮件系统。要使用的程序是由 php.ini 文件中的配置设置定义的。

安装

Mail 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

Mail 函数的行为受 php.ini 文件中的设置的影响。

Mail 配置选项：

名称	默认	描述	可更改
SMTP	"localhost"	Windows 专用：SMTP 服务器的 DNS 名称或 IP 地址。	PHP_INI_ALL
smtp_port	"25"	Windows 专用：SMTP 端口号。自 PHP 4.3 起可用。	PHP_INI_ALL
sendmail_from	NULL	Windows 专用：规定在由 PHP 发送的电子邮件中使用的 "from" 地址。	PHP_INI_ALL
sendmail_path	NULL	Unix 系统专用：规定 sendmail 程序的路径（通常 /usr/sbin/sendmail 或 /usr/lib/sendmail）。	PHP_INI_SYSTEM

PHP Mail 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
ezmlm_hash()	计算 EZMLM 邮件列表系统所需的散列值。	3
mail()	允许您从脚本中直接发送电子邮件。	3

PHP Mail 常量

无。

PHP Libxml 函数 PHP 5 Math 函数

PHP Mail 函数 PHP Misc. 函数

PHP 5 Math 函数

PHP Math 简介

Math 函数能处理 integer 和 float 范围内的值。

安装

PHP Math 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP 5 Math 函数

函数	描述
abs()	返回一个数的绝对值。
acos()	返回一个数的反余弦。
acosh()	返回一个数的反双曲余弦。
asin()	返回一个数的反正弦。
asinh()	返回一个数的反双曲正弦。
atan()	返回一个数的反正切。
atan2()	返回两个变量 x 和 y 的反正切。

atanh()	返回一个数的反双曲正切。
base_convert()	在任意进制之间转换数字。
bindec()	把二进制数转换为十进制数。
ceil()	向上舍入为最接近的整数。
cos()	返回一个数的余弦。
cosh()	返回一个数的双曲余弦。
decbin()	把十进制数转换为二进制数。
dechex()	把十进制数转换为十六进制数。
decoct()	把十进制数转换为八进制数。
deg2rad()	将角度值转换为弧度值。
exp()	返回 Ex 的值。
expm1()	返回 Ex - 1 的值。
floor()	向下舍入为最接近的整数。
fmod()	返回 x/y 的浮点数余数。
getrandmax()	返回通过调用 rand() 函数显示的随机数的最大可能值。
hexdec()	把十六进制数转换为十进制数。
hypot()	计算直角三角形的斜边长度。
is_finite()	判断是否为有限值。
is_infinite()	判断是否为无限值。
is_nan()	判断是否为非数值。
lcg_value()	返回范围为 (0, 1) 的一个伪随机数。
log()	返回一个数的自然对数（以 E 为底）。
log10()	返回一个数的以 10 为底的对数。
log1p()	返回 log(1+number)
max()	返回一个数组中的最大值，或者几个指定值中的最大值。
min()	返回一个数组中的最小值，或者几个指定值中的最小值。
mt_getrandmax()	返回通过调用 mt_rand() 函数显示的随机数的最大可能值。
mt_rand()	使用 Mersenne Twister 算法生成随机整数。
mt_srand()	播种 Mersenne Twister 随机数生成器。
octdec()	把八进制数转换为十进制数。
pi()	返回圆周率 PI 的值。

pow()	返回 x 的 y 次方。
rad2deg()	把弧度值转换为角度值。
rand()	返回随机整数。
round()	对浮点数进行四舍五入。
sin()	返回一个数的正弦。
sinh()	返回一个数的双曲正弦。
sqrt()	返回一个数的平方根。
srand()	播种随机数生成器。
tan()	返回一个数的正切。
tanh()	返回一个数的双曲正切。

PHP 5 预定义的 Math 常量

常量	值	描述	PHP 版本
INF	INF	无限	PHP 4
M_E	2.7182818284590452354	返回 e	PHP 4
M_EULER	0.57721566490153286061	返回 Euler 常量	PHP 4
M_LNPI	1.14472988584940017414	返回圆周率 PI 的自然对数：log_e(pi)	PHP 5.2
M_LN2	0.69314718055994530942	返回 2 的自然对数：log_e 2	PHP 4
M_LN10	2.30258509299404568402	返回 10 的自然对数：log_e 10	PHP 4
M_LOG2E	1.4426950408889634074	返回 E 的以 2 为底的对数：log_2 e	PHP 4
M_LOG10E	0.43429448190325182765	返回 E 的以 10 为底的对数：log_10 e	PHP 4
M_PI	3.14159265358979323846	返回 Pi	PHP 4
M_PI_2	1.57079632679489661923	返回 Pi/2	PHP 4
M_PI_4	0.78539816339744830962	返回 Pi/4	PHP 4
M_1_PI	0.31830988618379067154	返回 1/Pi	PHP 4
M_2_PI	0.63661977236758134308	返回 2/Pi	PHP 4
M_SQRTPI	1.77245385090551602729	返回圆周率 PI 的平方根：sqrt(pi)	PHP 5.2
M_2_SQRTPI	1.12837916709551257390	返回圆周率 PI 的 2/平方根：2/sqrt(pi)	PHP 4
M_SQRT1_2	0.70710678118654752440	返回 1/2 的平方根：1/sqrt(2)	PHP 4
M_SQRT2	1.41421356237309504880	返回 2 的平方根：sqrt(2)	PHP 4
M_SQRT3	1.73205080756887729352	返回 3 的平方根：sqrt(3)	PHP 5.2
NAN	NAN	不是一个数字	PHP 4

PHP_ROUND_HALF_UP	1	遇到 .5 的情况时向上舍入	PHP 5.3
PHP_ROUND_HALF_DOWN	2	遇到 .5 的情况时向下舍入	PHP 5.3
PHP_ROUND_HALF_EVEN	3	遇到 .5 的情况时取偶数舍入	PHP 5.3
PHP_ROUND_HALF_ODD	4	遇到 .5 的情况时取奇数舍入	PHP 5.3

PHP Mail 函数 PHP Misc. 函数

PHP 5 Math 函数 PHP 5 MySQLi 函数

PHP 杂项 函数

PHP 杂项函数简介

我们把不属于其他类别的函数归纳到杂项函数类别。

安装

杂项函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

Runtime 配置

杂项函数的行为受 php.ini 文件中的设置的影响。

杂项配置选项：

名称	默认	描述	可更改
ignore_user_abort	"0"	FALSE 指示只要脚本在客户机终止连接后尝试进行输出，脚本将被终止。	PHP_INI_ALL
highlight.string	"#DD0000"	供突出显示符合 PHP 语法的字符串而使用的颜色。	PHP_INI_ALL
highlight.comment	"#FF8000"	供突出显示 PHP 注释而使用的颜色。	PHP_INI_ALL
highlight.keyword	"#007700"	供语法高亮显示 PHP 关键词而使用的颜色（比如圆括号和分号）。	PHP_INI_ALL
highlight.bg	"#FFFFFF"	背景颜色。	PHP_INI_ALL
highlight.default	"#0000BB"	PHP 语法的默认颜色。	PHP_INI_ALL

highlight.html	"#000000"	HTML 代码的颜色。	PHP_INI_ALL
browscap	NULL	浏览器性能文件（例如：browscap.ini）的名称和位置。	PHP_INI_SYSTEM

PHP 杂项函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
connection_aborted()	检查是否断开客户机。	3
connection_status()	返回当前的连接状态。	3
connection_timeout()	在 PHP 4.0.5 中不赞成使用。检查脚本是否超时。	3
constant()	返回一个常量的值。	4
define()	定义一个常量。	3
defined()	检查某常量是否存在。	3
die()	输出一条消息，并退出当前脚本。	3
eval()	把字符串当成 PHP 代码来计算。	3
exit()	输出一条消息，并退出当前脚本。	3
get_browser()	返回用户浏览器的性能。	3
highlight_file()	对文件进行 PHP 语法高亮显示。	4
highlight_string()	对字符串进行 PHP 语法高亮显示。	4
ignore_user_abort()	设置与远程客户机断开是否会终止脚本的执行。	3
pack()	把数据装入一个二进制字符串。	3
php_check_syntax()	在 PHP 5.0.5 中不赞成使用。	5
php_strip_whitespace()	返回已删除 PHP 注释以及空白字符的源代码文件。	5
show_source()	highlight_file() 的别名。	4
sleep()	延迟代码执行若干秒。	3
time_nanosleep()	延迟代码执行若干秒和纳秒。	5
time_sleep_until()	延迟代码执行直到指定的时间。	5
uniqid()	生成唯一的 ID。	3
unpack()	从二进制字符串对数据进行解包。	3
usleep()	延迟代码执行若干微秒。	3

PHP 杂项常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP

CONNECTION_ABORTED		
CONNECTION_NORMAL		
CONNECTION_TIMEOUT		
__COMPILER_HALT_OFFSET__		5

□ PHP 5 Math 函数 PHP 5 MySQLi 函数 □

□ PHP Misc. 函数 PHP 5 SimpleXML 函数 □

PHP 5 MySQLi 函数

PHP MySQLi 简介

PHP MySQLi = PHP MySQL Improved!

MySQLi 函数允许您访问 MySQL 数据库服务器。

注释：MySQLi 扩展被设计用于 MySQL 4.1.13 版本或更新的版本。

安装 / Runtime 配置

为了能够顺利使用 MySQLi 函数，您必须在编译 PHP 时添加对 MySQLi 扩展的支持。

MySQLi 扩展是在 PHP 5.0.0 版本中引进的。MySQL Native Driver 包含在 PHP 5.3.0 版本。

有关安装的详细信息，请访问： http://www.php.net/manual/en/mysqli.installation.php

有关运行配置的详细信息，请访问： http://www.php.net/manual/en/mysqli.configuration.php

PHP 5 MySQLi 函数

函数	描述
mysqli_affected_rows()	返回前一次 MySQL 操作所影响的记录行数。
mysqli_autocommit()	打开或关闭自动提交数据库修改。
mysqli_change_user()	更改指定数据库连接的用户。
mysqli_character_set_name()	返回数据库连接的默认字符集。
mysqli_close()	关闭先前打开的数据库连接。
mysqli_commit()	提交当前事务。

mysqli_connect_errno()	返回上一次连接错误的错误代码。
mysqli_connect_error()	返回上一次连接错误的错误描述。
mysqli_connect()	打开一个到 MySQL 服务器的新的连接。
mysqli_data_seek()	调整结果指针到结果集中的一个任意行。
mysqli_debug()	执行调试操作。
mysqli_dump_debug_info()	转储调试信息到日志中。
mysqli_errno()	返回最近调用函数的最后一个错误代码。
mysqli_error_list()	返回最近调用函数的错误列表。
mysqli_error()	返回最近调用函数的最后一个错误描述。
mysqli_fetch_all()	从结果集中取得所有行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_array()	从结果集中取得一行作为关联数组，或数字数组，或二者兼有。
mysqli_fetch_assoc()	从结果集中取得一行作为关联数组。
mysqli_fetch_field_direct()	从结果集中取得某个单一字段的 meta-data，并作为对象返回。
mysqli_fetch_field()	从结果集中取得下一字段，并作为对象返回。
mysqli_fetch_fields()	返回结果中代表字段的对象的数组。
mysqli_fetch_lengths()	返回结果集中当前行的每个列的长度。
mysqli_fetch_object()	从结果集中取得当前行，并作为对象返回。
mysqli_fetch_row()	从结果集中取得一行，并作为枚举数组返回。
mysqli_field_count()	返回最近查询的列数。
mysqli_field_seek()	把结果集中的指针设置为指定字段的偏移量。
mysqli_field_tell()	返回结果集中的指针的位置。
mysqli_free_result()	释放结果内存。
mysqli_get_charset()	返回字符集对象。
mysqli_get_client_info()	返回 MySQL 客户端库版本。
mysqli_get_client_stats()	返回有关客户端每个进程的统计。
mysqli_get_client_version()	将 MySQL 客户端库版本作为整数返回。
mysqli_get_connection_stats()	返回有关客户端连接的统计。
mysqli_get_host_info()	返回 MySQL 服务器主机名和连接类型。
mysqli_get_proto_info()	返回 MySQL 协议版本。
mysqli_get_server_info()	返回 MySQL 服务器版本。
mysqli_get_server_version()	将 MySQL 服务器版本作为整数返回。

mysqli_info()	返回有关最近执行查询的信息。
mysqli_init()	初始化 MySQLi 并返回 mysqli_real_connect() 使用的资源。
mysqli_insert_id()	返回最后一个查询中自动生成的 ID。
mysql_kill()	请求服务器杀死一个 MySQL 线程。
mysqli_more_results()	检查一个多查询是否有更多的结果。
mysqli_multi_query()	执行一个或多个针对数据库的查询。
mysqli_next_result()	为 mysqli_multi_query() 准备下一个结果集。
mysqli_num_fields()	返回结果集中字段的数量。
mysqli_num_rows()	返回结果集中行的数量。
mysqli_options()	设置额外的连接选项，用于影响连接行为。
mysqli_ping()	进行一个服务器连接，如果连接已断开则尝试重新连接。
mysqli_prepare()	准备执行一个 SQL 语句。
mysqli_query()	执行某个针对数据库的查询。
mysqli_real_connect()	打开一个到 MySQL 服务器的新的链接。
mysqli_real_escape_string()	转义在 SQL 语句中使用的字符串中的特殊字符。
mysqli_real_query()	执行 SQL 查询
mysqli_reap_async_query()	返回异步查询的结果。
mysqli_refresh()	刷新表或缓存，或者重置复制服务器信息。
mysqli_rollback()	回滚数据库中的当前事务。
mysqli_select_db()	更改连接的默认数据库。
mysqli_set_charset()	设置默认客户端字符集。
mysqli_set_local_infile_default()	撤销用于 load local infile 命令的用户自定义句柄。
mysqli_set_local_infile_handler()	设置用于 LOAD DATA LOCAL INFILE 命令的回滚函数。
mysqli_sqlstate()	返回最后一个 MySQL 操作的 SQLSTATE 错误代码。
mysqli_ssl_set()	用于创建 SSL 安全连接。
mysqli_stat()	返回当前系统状态。
mysqli_stmt_init()	初始化声明并返回 mysqli_stmt_prepare() 使用的对象。
mysqli_store_result()	传输最后一个查询的结果集。
mysqli_thread_id()	返回当前连接的线程 ID。
mysqli_thread_safe()	返回是否将客户端库编译成 thread-safe。
mysqli_use_result()	从上次使用 mysqli_real_query() 执行的查询中初始化结果集的检索。

mysqli_warning_count()	返回连接中的最后一个查询的警告数量。

PHP Misc. 函数 PHP 5 SimpleXML 函数

PHP PDO

PHP 数据对象 （PDO） 扩展为PHP访问数据库定义了一个轻量级的一致接口。

PDO 提供了一个数据访问抽象层，这意味着，不管使用哪种数据库，都可以用相同的函数（方法）来查询和获取数据。

PDO随PHP5.1发行，在PHP5.0的PECL扩展中也可以使用，无法运行于之前的PHP版本。

PDO 安装

你可以通过 PHP 的 phpinfo() 函数来查看是否安装了PDO扩展。

在 Unix 系统上安装 PDO

在Unix上或Linux上你需要添加以下扩展：

```
extension=pdo.so
```

Windows 用户

PDO 和所有主要的驱动作为共享扩展随 PHP 一起发布，要激活它们只需简单地编辑 php.ini 文件，并添加以下扩展：

```
extension=php_pdo.dll
```

除此之外还有以下对应的各种数据库扩展：

```
;extension=php_pdo_firebird.dll
;extension=php_pdo_informix.dll
;extension=php_pdo_mssql.dll
;extension=php_pdo_mysql.dll
```

```
;extension=php_pdo_oci.dll

;extension=php_pdo_oci8.dll

;extension=php_pdo_odbc.dll

;extension=php_pdo_pgsql.dll

;extension=php_pdo_sqlite.dll
```

在设定好这些配置后，我们需要重启PHP 或 Web服务器。
接下来我们们来看下具体的实例，以下为使用PDO连接MySql数据库的实例：

```
<?php

$dbms='mysql';     //数据库类型

$host='localhost'; //数据库主机名

$dbName='test';    //使用的数据库

$user='root';      //数据库连接用户名

$pass='';          //对应的密码

$dsn="$dbms:host=$host;dbname=$dbName";




try {

    $dbh = new PDO($dsn, $user, $pass); //初始化一个PDO对象

    echo "连接成功<br/>";

    /*你还可以进行一次搜索操作

    foreach ($dbh->query('SELECT * from FOO') as $row) {

        print_r($row); //你可以用 echo($GLOBAL); 来看到这些值

    }

    */

    $dbh = null;

} catch (PDOException $e) {

    die ("Error!: " . $e->getMessage() . "<br/>");

}

//默认这个不是长连接，如果需要数据库长连接，需要最后加一个参数：array(PDO::ATTR_PERSISTENT => true) 变成这样：

$db = new PDO($dsn, $user, $pass, array(PDO::ATTR_PERSISTENT => true));
```

```
?>
```

很简单吧，接下来就让我们来具体看下PHP PDO具体说明：

预定义常量

PHP PDO连接连接管理

PHP PDO 事务与自动提交

PHP PDO 预处理语句与存储过程

PHP PDO 错误与错误处理

PHP PDO 大对象 (LOBs)

PDO 类：

PDO::beginTransaction — 启动一个事务

PDO::commit — 提交一个事务

PDO::__construct — 创建一个表示数据库连接的 PDO 实例

PDO::errorCode — 获取跟数据库句柄上一次操作相关的 SQLSTATE

PDO::errorInfo — 返回最后一次操作数据库的错误信息

PDO::exec — 执行一条 SQL 语句，并返回受影响的行数

PDO::getAttribute — 取回一个数据库连接的属性

PDO::getAvailableDrivers — 返回一个可用驱动的数组

PDO::inTransaction — 检查是否在一个事务内

PDO::lastInsertId — 返回最后插入行的ID或序列值

PDO::prepare — 备要执行的SQL语句并返回一个 PDOStatement 对象

PDO::query — 执行 SQL 语句，返回PDOStatement对象,可以理解为结果集

PDO::quote — 为SQL语句中的字符串添加引号。

PDO::rollBack — 回滚一个事务

PDO::setAttribute — 设置属性

PDOStatement 类：

PDOStatement::bindColumn — 绑定一列到一个 PHP 变量

PDOStatement::bindParam — 绑定一个参数到指定的变量名

PDOStatement::bindValue — 把一个值绑定到一个参数

PDOStatement::closeCursor — 关闭游标，使语句能再次被执行。

PDOStatement::columnCount — 返回结果集中的列数

PDOStatement::debugDumpParams — 打印一条 SQL 预处理命令

PDOStatement::errorCode — 获取跟上一次语句句柄操作相关的 SQLSTATE

PDOStatement::errorInfo — 获取跟上一次语句句柄操作相关的扩展错误信息

PDOStatement::execute — 执行一条预处理语句

PDOStatement::fetch — 从结果集中获取下一行

PDOStatement::fetchAll — 返回一个包含结果集中所有行的数组

PDOStatement::fetchColumn — 从结果集中的下一行返回单独的一列。

PDOStatement::fetchObject — 获取下一行并作为一个对象返回。

PDOStatement::getAttribute — 检索一个语句属性

PDOStatement::getColumnMeta — 返回结果集中一列的元数据

PDOStatement::nextRowset — 在一个多行集语句句柄中推进到下一个行集

PDOStatement::rowCount — 返回受上一个 SQL 语句影响的行数

PDOStatement::setAttribute — 设置一个语句属性

PDOStatement::setFetchMode — 为语句设置默认的获取模式。

□ PHP 命名空间(namespace) PHP PDO预定义常量 □

PHP 5 SimpleXML 函数

PHP SimpleXML 简介

SimpleXML 扩展提供了一种获取 XML 元素的名称和文本的简单方式，只要您知道 XML 文档的布局。

SimpleXML 转换 XML 文档到 SimpleXMLElement 对象。

通过正常的属性选择器和数组迭代器，这个对象能够像其他对象一样被处理。

提示：与 DOM 或者 Expat 解析器比较，SimpleXML 只需要几行代码就能读取元素中的文本数据。

安装

SimpleXML 扩展需要 PHP 5 支持。

自 PHP 5 起，SimpleXML 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP 5 SimpleXML 函数

函数	描述
__construct()	创建一个新的 SimpleXMLElement 对象。
addAttribute()	给 SimpleXML 元素添加一个属性。
addChild()	给 SimpleXML 元素添加一个子元素。
asXML()	格式化 XML（版本 1.0）中的 SimpleXML 对象的数据。
attributes()	返回 XML 标签的属性和值。
children()	查找指定节点的子节点。
count()	计算指定节点的子节点个数。
getDocNamespaces()	返回文档中的声明的命名空间。

getName()	返回 SimpleXML 元素引用的 XML 标签的名称。
getNamespaces()	返回文档中使用的命名空间。
registerXPathNamespace()	为下一个 XPath 查询创建命名空间上下文。
saveXML()	asXML() 的别名。
simplexml_import_dom()	从 DOM 节点返回 SimpleXMLElement 对象。
simplexml_load_file()	转换 XML 文件为 SimpleXMLElement 对象。
simplexml_load_string()	转换 XML 字符串为 SimpleXMLElement 对象。
xpath()	运行对 XML 数据的 XPath 查询。

PHP 5 SimpleXML 迭代函数

函数	描述
current()	返回当前元素。
getChildren()	返回当前元素的子元素。
hasChildren()	检查当前元素是否有子元素。
key()	返回当前键。
next()	移动到下一个元素。
rewind()	倒回到第一个元素。
valid()	检查当前元素是否有效。

□ PHP 5 MySQLi 函数 PHP 5 String 函数 □

□ PHP 5 SimpleXML 函数 PHP XML 函数 □

PHP 5 String 函数

PHP 5 String 函数

PHP String 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

函数	描述

addcslashes()	返回在指定的字符前添加反斜杠的字符串。
addslashes()	返回在预定义的字符前添加反斜杠的字符串。
bin2hex()	把 ASCII 字符的字符串转换为十六进制值。
chop()	移除字符串右侧的空白字符或其他字符。
chr()	从指定 ASCII 值返回字符。
chunk_split()	把字符串分割为一连串更小的部分。
convert_cyr_string()	把字符串由一种 Cyrillic 字符集转换成另一种。
convert_uudecode()	对 uuencode 编码的字符串进行解码。
convert_uuencode()	使用 uuencode 算法对字符串进行编码。
count_chars()	返回字符串所用字符的信息。
crc32()	计算一个字符串的 32 位 CRC（循环冗余校验）。
crypt()	单向的字符串加密法（hashing）。
echo()	输出一个或多个字符串。
explode()	把字符串打散为数组。
fprintf()	把格式化的字符串写入到指定的输出流。
get_html_translation_table()	返回 htmlspecialchars() 和 htmlentities() 使用的翻译表。
hebrev()	把希伯来（Hebrew）文本转换为可见文本。
hebrevc()	把希伯来（Hebrew）文本转换为可见文本，并把新行（\n）转换为 。
hex2bin()	把十六进制值的字符串转换为 ASCII 字符。
html_entity_decode()	把 HTML 实体转换为字符。
htmlentities()	把字符转换为 HTML 实体。
htmlspecialchars_decode()	把一些预定义的 HTML 实体转换为字符。
htmlspecialchars()	把一些预定义的字符转换为 HTML 实体。
implode()	返回一个由数组元素组合成的字符串。
join()	implode() 的别名。
lcfirst()	把字符串中的首字符转换为小写。
levenshtein()	返回两个字符串之间的 Levenshtein 距离。
localeconv()	返回本地数字及货币格式信息。
ltrim()	移除字符串左侧的空白字符或其他字符。
md5()	计算字符串的 MD5 散列。
md5_file()	计算文件的 MD5 散列。

metaphone()	计算字符串的 metaphone 键。
money_format()	返回格式化为货币字符串的字符串。
nl_langinfo()	返回指定的本地信息。
nl2br()	在字符串中的每个新行之前插入 HTML 换行符。
number_format()	通过千位分组来格式化数字。
ord()	返回字符串中第一个字符的 ASCII 值。
parse_str()	把查询字符串解析到变量中。
print()	输出一个或多个字符串。
printf()	输出格式化的字符串。
quoted_printable_decode()	把 quoted-printable 字符串转换为 8 位字符串。
quoted_printable_encode()	把 8 位字符串转换为 quoted-printable 字符串。
quotemeta()	引用元字符。
rtrim()	移除字符串右侧的空白字符或其他字符。
setlocale()	设置地区信息（地域信息）。
sha1()	计算字符串的 SHA-1 散列。
sha1_file()	计算文件的 SHA-1 散列。
similar_text()	计算两个字符串的相似度。
soundex()	计算字符串的 soundex 键。
sprintf()	把格式化的字符串写入一个变量中。
sscanf()	根据指定的格式解析来自一个字符串的输入。
str_getcsv()	把 CSV 字符串解析到数组中。
str_ireplace()	替换字符串中的一些字符（大小写不敏感）。
str_pad()	把字符串填充为新的长度。
str_repeat()	把字符串重复指定的次数。
str_replace()	替换字符串中的一些字符（大小写敏感）。
str_rot13()	对字符串执行 ROT13 编码。
str_shuffle()	随机地打乱字符串中的所有字符。
str_split()	把字符串分割到数组中。
str_word_count()	计算字符串中的单词数。
strcasecmp()	比较两个字符串（大小写不敏感）。
strchr()	查找字符串在另一字符串中的第一次出现。（strstr() 的别名。）

strcmp()	比较两个字符串（大小写敏感）。
strcoll()	比较两个字符串（根据本地设置）。
strcspn()	返回在找到任何指定的字符之前，在字符串查找的字符数。
strip_tags()	剥去字符串中的 HTML 和 PHP 标签。
stripcslashes()	删除由 addcslashes() 函数添加的反斜杠。
stripslashes()	删除由 addslashes() 函数添加的反斜杠。
stripos()	返回字符串在另一字符串中第一次出现的位置（大小写不敏感）。
stristr()	查找字符串在另一字符串中第一次出现的位置（大小写不敏感）。
strlen()	返回字符串的长度。
strnatcasecmp()	使用一种"自然排序"算法来比较两个字符串（大小写不敏感）。
strnatcmp()	使用一种"自然排序"算法来比较两个字符串（大小写敏感）。
strncasecmp()	前 n 个字符的字符串比较（大小写不敏感）。
strncmp()	前 n 个字符的字符串比较（大小写敏感）。
strpbrk()	在字符串中搜索指定字符中的任意一个。
strpos()	返回字符串在另一字符串中第一次出现的位置（大小写敏感）。
strrchr()	查找字符串在另一个字符串中最后一次出现。
strrev()	反转字符串。
strripos()	查找字符串在另一字符串中最后一次出现的位置(大小写不敏感)。
strrpos()	查找字符串在另一字符串中最后一次出现的位置(大小写敏感)。
strspn()	返回在字符串中包含的特定字符的数目。
strstr()	查找字符串在另一字符串中的第一次出现（大小写敏感）。
strtok()	把字符串分割为更小的字符串。
strtolower()	把字符串转换为小写字母。
strtoupper()	把字符串转换为大写字母。
strtr()	转换字符串中特定的字符。
substr()	返回字符串的一部分。
substr_compare()	从指定的开始位置（二进制安全和选择性区分大小写）比较两个字符串。
substr_count()	计算子串在字符串中出现的次数。
substr_replace()	把字符串的一部分替换为另一个字符串。
trim()	移除字符串两侧的空白字符和其他字符。
ucfirst()	把字符串中的首字符转换为大写。

ucwords()	把字符串中每个单词的首字符转换为大写。
vfprintf()	把格式化的字符串写到指定的输出流。
vprintf()	输出格式化的字符串。
vsprintf()	把格式化字符串写入变量中。
wordwrap()	按照指定长度对字符串进行折行处理。

PHP 5 SimpleXML 函数 PHP XML 函数

PHP XML Parser 函数

PHP XML Parser 简介

XML 函数允许您解析 XML 文档，但无法对其进行验证。

XML 是一种用于标准结构化文档交换的数据格式。您可以在我们的 XML 教程 中找到更多有关 XML 的信息。

该扩展使用 Expat XML 解析器。

Expat 是一种基于事件的解析器，它把 XML 文档视为一系列事件。当某个事件发生时，它调用一个指定的函数处理它。

Expat 是无验证的解析器，忽略任何链接到文档的 DTD。但是，如果文档的形式不好，则会以一个错误消息结束。

由于它是一种基于事件，且无验证的解析器，Expat 具有快速并适合 Web 应用程序的特性。

XML 解析器函数允许您创建 XML 解析器，并为 XML 事件定义句柄。

安装

XML Parser 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP XML Parser 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
utf8_decode()	把 UTF-8 字符串解码为 ISO-8859-1。	3
utf8_encode()	把 ISO-8859-1 字符串编码为 UTF-8。	3
xml_error_string()	获取 XML 解析器的错误字符串。	3
xml_get_current_byte_index()	获取 XML 解析器的当前字节索引。	3

xml_get_current_column_number()	获取 XML 解析器的当前列号。	3
xml_get_current_line_number()	获取 XML 解析器的当前行号。	3
xml_get_error_code()	获取 XML 解析器的错误代码。	3
xml_parse()	解析 XML 文档。	3
xml_parse_into_struct()	把 XML 数据解析到数组中。	3
xml_parser_create_ns()	创建带有命名空间支持的 XML 解析器。	4
xml_parser_create()	创建 XML 解析器。	3
xml_parser_free()	释放 XML 解析器。	3
xml_parser_get_option()	从 XML 解析器获取选项。	3
xml_parser_set_option()	为 XML 解析器设置选项。	3
xml_set_character_data_handler()	建立字符数据处理器。	3
xml_set_default_handler()	建立默认处理器。	3
xml_set_element_handler()	建立起始和终止元素处理器。	3
xml_set_end_namespace_decl_handler()	建立终止命名空间声明处理器。	4
xml_set_external_entity_ref_handler()	建立外部实体处理器。	3
xml_set_notation_decl_handler()	建立符号声明处理器。	3
xml_set_object()	在对象中使用 XML 解析器。	4
xml_set_processing_instruction_handler()	建立处理指令（PI）处理器。	3
xml_set_start_namespace_decl_handler()	建立起始命名空间声明处理器。	4
xml_set_unparsed_entity_decl_handler()	建立未解析实体声明处理器。	3

PHP XML Parser 常量

常量
XML_ERROR_NONE (integer)
XML_ERROR_NO_MEMORY (integer)
XML_ERROR_SYNTAX (integer)
XML_ERROR_NO_ELEMENTS (integer)
XML_ERROR_INVALID_TOKEN (integer)
XML_ERROR_UNCLOSED_TOKEN (integer)
XML_ERROR_PARTIAL_CHAR (integer)
XML_ERROR_TAG_MISMATCH (integer)
XML_ERROR_DUPLICATE_ATTRIBUTE (integer)

XML_ERROR_JUNK_AFTER_DOC_ELEMENT (integer)
XML_ERROR_PARAM_ENTITY_REF (integer)
XML_ERROR_UNDEFINED_ENTITY (integer)
XML_ERROR_RECURSIVE_ENTITY_REF (integer)
XML_ERROR_ASYNC_ENTITY (integer)
XML_ERROR_BAD_CHAR_REF (integer)
XML_ERROR_BINARY_ENTITY_REF (integer)
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF (integer)
XML_ERROR_MISPLACED_XML_PI (integer)
XML_ERROR_UNKNOWN_ENCODING (integer)
XML_ERROR_INCORRECT_ENCODING (integer)
XML_ERROR_UNCLOSED_CDATA_SECTION (integer)
XML_ERROR_EXTERNAL_ENTITY_HANDLING (integer)
XML_OPTION_CASE_FOLDING (integer)
XML_OPTION_TARGET_ENCODING (integer)
XML_OPTION_SKIP_TAGSTART (integer)
XML_OPTION_SKIP_WHITE (integer)

□ PHP 5 String 函数 PHP Zip File 函数 □

□ PHP XML 函数 PHP 教程 □

PHP Zip File 函数

PHP Zip File 简介

Zip File 函数允许您读取压缩文件。

安装

如需在服务器上运行 Zip File 函数，必须安装这些库：

Guido Draheim 的 ZZIPlib 库： 下载 ZZIPlib 库

Zip PELC 扩展：下载 Zip PELC 扩展

在 Linux 系统上安装

PHP 5+： Zip 函数和 Zip 库默认不会启用，必须从上面的链接下载。请使用 *--with-zip=DIR* 配置选项来包含 Zip 支持。

在 Windows 系统上安装

PHP 5+： Zip 函数默认不会启用，必须从上面的链接下载 php_zip.dll 和 ZZIPlib 库。必须在 php.ini 中启用 php_zip.dll。

如需启用任何 PHP 扩展，PHP extension_dir 设置（在 php.ini 文件中）应该设置为该 PHP 扩展所在的目录。举例 extension_dir 的值可能是 c:\php\ext。

PHP Zip File 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
zip_close()	关闭 ZIP 文件。	4
zip_entry_close()	关闭 ZIP 文件中的一个项目。	4
zip_entry_compressedsize()	返回 ZIP 文件中的一个项目的被压缩尺寸。	4
zip_entry_compressionmethod()	返回 ZIP 文件中的一个项目的压缩方法。	4
zip_entry_filesize()	返回 ZIP 文件中的一个项目的实际文件尺寸。	4
zip_entry_name()	返回 ZIP 文件中的一个项目的名称。	4
zip_entry_open()	打开 ZIP 文件中的一个项目以供读取。	4
zip_entry_read()	读取 ZIP 文件中的一个打开的项目。	4
zip_open()	打开 ZIP 文件。	4
zip_read()	读取 ZIP 文件中的下一个项目。	4

PHP Zip File 常量

无。

PHP XML 函数 PHP 教程

点我分享笔记

反馈/建议

PHP 5 时区

PHP 支持的时区

下面是 PHP 支持的时区的完整列表，这些对一些 PHP 日期函数很有用。

- 非洲
- 美洲
- 南极洲
- 北冰洋
- 亚洲
- 大西洋
- 大洋洲
- 欧洲
- 印度洋
- 太平洋

非洲

Africa/Abidjan	Africa/Accra	Africa/Addis_Ababa	Africa/Algiers	Africa/Asmara
Africa/Asmera	Africa/Bamako	Africa/Bangui	Africa/Banjul	Africa/Bissau
Africa/Blantyre	Africa/Brazzaville	Africa/Bujumbura	Africa/Cairo	Africa/Casablanca
Africa/Ceuta	Africa/Conakry	Africa/Dakar	Africa/Dar_es_Salaam	Africa/Djibouti
Africa/Douala	Africa/El_Aaiun	Africa/Freetown	Africa/Gaborone	Africa/Harare
Africa/Johannesburg	Africa/Juba	Africa/Kampala	Africa/Khartoum	Africa/Kigali
Africa/Kinshasa	Africa/Lagos	Africa/Libreville	Africa/Lome	Africa/Luanda
Africa/Lubumbashi	Africa/Lusaka	Africa/Malabo	Africa/Maputo	Africa/Maseru
Africa/Mbabane	Africa/Mogadishu	Africa/Monrovia	Africa/Nairobi	Africa/Ndjamena
Africa/Niamey	Africa/Nouakchott	Africa/Ouagadougou	Africa/Porto-Novo	Africa/Sao_Tome
Africa/Timbuktu	Africa/Tripoli	Africa/Tunis	Africa/Windhoek	

美洲

America/Adak	America/Anchorage	America/Anguilla
America/Antigua	America/Araguaina	America/Argentina/Buenos_Aires
America/Argentina/Catamarca	America/Argentina/ComodRivadavia	America/Argentina/Cordoba
America/Argentina/Jujuy	America/Argentina/La_Rioja	America/Argentina/Mendoza
America/Argentina/Rio_Gallegos	America/Argentina/Salta	America/Argentina/San_Juan
America/Argentina/San_Luis	America/Argentina/Tucuman	America/Argentina/Ushuaia

America/Aruba	America/Asuncion	America/Atikokan
America/Atka	America/Bahia	America/Bahia_Banderas
America/Barbados	America/Belem	America/Belize
America/Blanc-Sablon	America/Boa_Vista	America/Bogota
America/Boise	America/Buenos_Aires	America/Cambridge_Bay
America/Campo_Grande	America/Cancun	America/Caracas
America/Catamarca	America/Cayenne	America/Cayman
America/Chicago	America/Chihuahua	America/Coral_Harbour
America/Cordoba	America/Costa_Rica	America/Creston
America/Cuiaba	America/Curacao	America/Danmarkshavn
America/Dawson	America/Dawson_Creek	America/Denver
America/Detroit	America/Dominica	America/Edmonton
America/Eirunepe	America/El_Salvador	America/Ensenada
America/Fort_Wayne	America/Fortaleza	America/Glace_Bay
America/Godthab	America/Goose_Bay	America/Grand_Turk
America/Grenada	America/Guadeloupe	America/Guatemala
America/Guayaquil	America/Guyana	America/Halifax
America/Havana	America/Hermosillo	America/Indiana/Indianapolis
America/Indiana/Knox	America/Indiana/Marengo	America/Indiana/Petersburg
America/Indiana/Tell_City	America/Indiana/Vevay	America/Indiana/Vincennes
America/Indiana/Winamac	America/Indianapolis	America/Inuvik
America/Iqaluit	America/Jamaica	America/Jujuy
America/Juneau	America/Kentucky/Louisville	America/Kentucky/Monticello
America/Knox_IN	America/Kralendijk	America/La_Paz
America/Lima	America/Los_Angeles	America/Louisville
America/Lower_Princes	America/Maceio	America/Managua
America/Manaus	America/Marigot	America/Martinique
America/Matamoros	America/Mazatlan	America/Mendoza
America/Menominee	America/Merida	America/Metlakatla
America/Mexico_City	America/Miquelon	America/Moncton
America/Monterrey	America/Montevideo	America/Montreal

America/Montserrat	America/Nassau	America/New_York
America/Nipigon	America/Nome	America/Noronha
America/North_Dakota/Beulah	America/North_Dakota/Center	America/North_Dakota/New_Salem
America/Ojinaga	America/Panama	America/Pangnirtung
America/Paramaribo	America/Phoenix	America/Port-au-Prince
America/Port_of_Spain	America/Porto_Acre	America/Porto_Velho
America/Puerto_Rico	America/Rainy_River	America/Rankin_Inlet
America/Recife	America/Regina	America/Resolute
America/Rio_Branco	America/Rosario	America/Santa_Isabel
America/Santarem	America/Santiago	America/Santo_Domingo
America/Sao_Paulo	America/Scoresbysund	America/Shiprock
America/Sitka	America/St_Barthelemy	America/St_Johns
America/St_Kitts	America/St_Lucia	America/St_Thomas
America/St_Vincent	America/Swift_Current	America/Tegucigalpa
America/Thule	America/Thunder_Bay	America/Tijuana
America/Toronto	America/Tortola	America/Vancouver
America/Virgin	America/Whitehorse	America/Winnipeg
America/Yakutat	America/Yellowknife	

南极洲

Antarctica/Casey	Antarctica/Davis	Antarctica/DumontDUrville	Antarctica/Macquarie	Antarctica/Mawson
Antarctica/McMurdo	Antarctica/Palmer	Antarctica/Rothera	Antarctica/South_Pole	Antarctica/Syowa
Antarctica/Vostok				

北冰洋

Arctic/Longyearbyen

亚洲

Asia/Aden	Asia/Almaty	Asia/Amman	Asia/Anadyr	Asia/Aqtau
Asia/Aqtobe	Asia/Ashgabat	Asia/Ashkhabad	Asia/Baghdad	Asia/Bahrain
Asia/Baku	Asia/Bangkok	Asia/Beirut	Asia/Bishkek	Asia/Brunei
Asia/Calcutta	Asia/Choibalsan	Asia/Chongqing	Asia/Chungking	Asia/Colombo
Asia/Dacca	Asia/Damascus	Asia/Dhaka	Asia/Dili	Asia/Dubai
Asia/Dushanbe	Asia/Gaza	Asia/Harbin	Asia/Hebron	Asia/Ho_Chi_Minh

Asia/Hong_Kong	Asia/Hovd	Asia/Irkutsk	Asia/Istanbul	Asia/Jakarta
Asia/Jayapura	Asia/Jerusalem	Asia/Kabul	Asia/Kamchatka	Asia/Karachi
Asia/Kashgar	Asia/Kathmandu	Asia/Katmandu	Asia/Khandyga	Asia/Kolkata
Asia/Krasnoyarsk	Asia/Kuala_Lumpur	Asia/Kuching	Asia/Kuwait	Asia/Macao
Asia/Macau	Asia/Magadan	Asia/Makassar	Asia/Manila	Asia/Muscat
Asia/Nicosia	Asia/Novokuznetsk	Asia/Novosibirsk	Asia/Omsk	Asia/Oral
Asia/Phnom_Penh	Asia/Pontianak	Asia/Pyongyang	Asia/Qatar	Asia/Qyzylorda
Asia/Rangoon	Asia/Riyadh	Asia/Saigon	Asia/Sakhalin	Asia/Samarkand
Asia/Seoul	Asia/Shanghai	Asia/Singapore	Asia/Taipei	Asia/Tashkent
Asia/Tbilisi	Asia/Tehran	Asia/Tel_Aviv	Asia/Thimbu	Asia/Thimphu
Asia/Tokyo	Asia/Ujung_Pandang	Asia/Ulaanbaatar	Asia/Ulan_Bator	Asia/Urumqi
Asia/Ust-Nera	Asia/Vientiane	Asia/Vladivostok	Asia/Yakutsk	Asia/Yekaterinburg
Asia/Yerevan				

大西洋

Atlantic/Azores	Atlantic/Bermuda	Atlantic/Canary	Atlantic/Cape_Verde	Atlantic/Faeroe
Atlantic/Faroe	Atlantic/Jan_Mayen	Atlantic/Madeira	Atlantic/Reykjavik	Atlantic/South_Georgia
Atlantic/St_Helena	Atlantic/Stanley			

大洋洲

Australia/ACT	Australia/Adelaide	Australia/Brisbane	Australia/Broken_Hill	Australia/Canberra
Australia/Currie	Australia/Darwin	Australia/Eucla	Australia/Hobart	Australia/LHI
Australia/Lindeman	Australia/Lord_Howe	Australia/Melbourne	Australia/North	Australia/NSW
Australia/Perth	Australia/Queensland	Australia/South	Australia/Sydney	Australia/Tasmania
Australia/Victoria	Australia/West	Australia/Yancowinna		

欧洲

Europe/Amsterdam	Europe/Andorra	Europe/Athens	Europe/Belfast	Europe/Belgrade
Europe/Berlin	Europe/Bratislava	Europe/Brussels	Europe/Bucharest	Europe/Budapest
Europe/Busingen	Europe/Chisinau	Europe/Copenhagen	Europe/Dublin	Europe/Gibraltar
Europe/Guernsey	Europe/Helsinki	Europe/Isle_of_Man	Europe/Istanbul	Europe/Jersey
Europe/Kaliningrad	Europe/Kiev	Europe/Lisbon	Europe/Ljubljana	Europe/London
Europe/Luxembourg	Europe/Madrid	Europe/Malta	Europe/Mariehamn	Europe/Minsk
Europe/Monaco	Europe/Moscow	Europe/Nicosia	Europe/Oslo	Europe/Paris

Europe/Podgorica	Europe/Prague	Europe/Riga	Europe/Rome	Europe/Samara
Europe/San_Marino	Europe/Sarajevo	Europe/Simferopol	Europe/Skopje	Europe/Sofia
Europe/Stockholm	Europe/Tallinn	Europe/Tirane	Europe/Tiraspol	Europe/Uzhgorod
Europe/Vaduz	Europe/Vatican	Europe/Vienna	Europe/Vilnius	Europe/Volgograd
Europe/Warsaw	Europe/Zagreb	Europe/Zaporozhye	Europe/Zurich	

印度洋

Indian/Antananarivo	Indian/Chagos	Indian/Christmas	Indian/Cocos	Indian/Comoro
Indian/Kerguelen	Indian/Mahe	Indian/Maldives	Indian/Mauritius	Indian/Mayotte
Indian/Reunion				

太平洋

Pacific/Apia	Pacific/Auckland	Pacific/Chatham	Pacific/Chuuk	Pacific/Easter
Pacific/Efate	Pacific/Enderbury	Pacific/Fakaofo	Pacific/Fiji	Pacific/Funafuti
Pacific/Galapagos	Pacific/Gambier	Pacific/Guadalcanal	Pacific/Guam	Pacific/Honolulu
Pacific/Johnston	Pacific/Kiritimati	Pacific/Kosrae	Pacific/Kwajalein	Pacific/Majuro
Pacific/Marquesas	Pacific/Midway	Pacific/Nauru	Pacific/Niue	Pacific/Norfolk
Pacific/Noumea	Pacific/Pago_Pago	Pacific/Palau	Pacific/Pitcairn	Pacific/Pohnpei
Pacific/Ponape	Pacific/Port_Moresby	Pacific/Rarotonga	Pacific/Saipan	Pacific/Samoa
Pacific/Tahiti	Pacific/Tarawa	Pacific/Tongatapu	Pacific/Truk	Pacific/Wake
Pacific/Wallis	Pacific/Yap			

□ PHP 实例 AJAX 投票

PHP array() 函数 □

□点我分享笔记

反馈/建议

□

□ PHP MySQL 预处理语句

php getimagesize 函数 – 获取图像信息 □

PHP 图像处理

PHP 提供了丰富的图像处理函数，主要包括：

函数	描述
gd_info()	取得当前安装的 GD 库的信息
getimagesize()	获取图像信息
getimagesizefromstring()	获取图像信息
image_type_to_extension()	获取图片后缀
image_type_to_mime_type()	返回图像的 MIME 类型
image2wbmp()	输出WBMP图片
imageaffine()	返回经过仿射变换后的图像
imageaffinematrixconcat()	连接两个矩阵
imageaffinematrixget()	获取矩阵
imagealphablending()	设定图像的混色模式
imageantialias()	是否使用抗锯齿（antialias）功能
imagearc()	画椭圆弧
imagechar()	写出横向字符
imagecharup()	垂直地画一个字符
imagecolorallocate()	为一幅图像分配颜色
imagecolorallocatealpha()	为一幅图像分配颜色和透明度
imagecolorat()	取得某像素的颜色索引值
imagecolorclosest()	取得与指定的颜色最接近的颜色的索引值
imagecolorclosestalpha()	取得与指定的颜色加透明度最接近的颜色的索引
imagecolorclosesthwb()	取得与指定的颜色最接近的色度的黑白色的索引
imagesx()、imagesy()	获取图像宽度与高度

GD 库

使用 PHP 图像处理函数，需要加载 GD 支持库。请确定 php.ini 加载了 GD 库：

Window 服务器上：

```
extension = php_gd2.dll
```

Linux 和 Mac 系统上:

```
extension = php_gd2.so
```

使用 gd_info() 函数可以查看当前安装的 GD 库的信息：

```
<?php
```

```
var_dump(gd_info());

?>
```

输出大致如下：

```
array(12) {

  ["GD Version"]=>

  string(26) "bundled (2.1.0 compatible)"

  ["FreeType Support"]=>

  bool(true)

  ["FreeType Linkage"]=>

  string(13) "with freetype"

  ["T1Lib Support"]=>

  bool(false)

  ["GIF Read Support"]=>

  bool(true)

  ["GIF Create Support"]=>

  bool(true)

  ["JPEG Support"]=>

  bool(true)

  ["PNG Support"]=>

  bool(true)

  ["WBMP Support"]=>

  bool(true)

  ["XPM Support"]=>

  bool(false)

  ["XBM Support"]=>

  bool(true)

  ["JIS-mapped Japanese Font Support"]=>

  bool(false)

}
```

PHP RESTful

REST（英文：Representational State Transfer，简称REST），指的是一组架构约束条件和原则。

符合REST设计风格的Web API称为RESTful API。它从以下三个方面资源进行定义：

直观简短的资源地址：URI，比如：`http://example.com/resources/`。

传输的资源：Web服务接受与返回的互联网媒体类型，比如：JSON，XML，YAM等。

对资源的操作：Web服务在该资源上所支持的一系列请求方法（比如：POST，GET，PUT或DELETE）。

本教程我们将使用 PHP(不用框架) 来创建一个 RESTful web service，在文章末尾你可以下载本章节使用到的代码。

通过本教程你将学习到以下内容：

创建一个 RESTful Webservice。

使用原生 PHP, 不依赖任何框架。

URI 模式需要遵循 REST 规则。

RESTful service 接受与返回的格式可以是 JSON, XML等。

根据不同情况响应对应的 HTTP 状态码。

演示请求头的使用。

使用 REST 客户端来测试 RESTful web service。

RESTful Webservice 实例

以下代码是 RESTful 服务类 **Site.php**：

实例

```
<?php
/*
* 菜鸟教程 RESTful 演示实例
* RESTful 服务类
*/
Class Site {
private $sites = array(
1 => 'TaoBao',
2 => 'Google',
3 => 'Runoob',
4 => 'Baidu',
5 => 'Weibo',
6 => 'Sina'
);
public function getAllSite(){
return $this->sites;
```

```
}
public function getSite($id){
$site = array($id => ($this->sites[$id]) ? $this->sites[$id] : $this->sites[1]);
return $site;
}
}
?>
```

RESTful Services URI 映射

RESTful Services URI 应该设置为一个直观简短的资源地址。Apache 服务器的 .htaccess 应设置好对应的 Rewrite 规则。

本实例我们将使用两个 URI 规则：

1、获取所有站点列表：

```
http://localhost/restexample/site/list/
```

2、使用 id 获取指定的站点，以下 URI 为获取 id 为 3 的站点：

```
http://localhost/restexample/site/list/3/
```

项目的 **.htaccess** 文件配置规则如下所示：

```
# 开启 rewrite 功能

Options +FollowSymlinks

RewriteEngine on



# 重写规则

RewriteRule ^site/list/$   RestController.php?view=all [nc,qsa]

RewriteRule ^site/list/([0-9]+)/$   RestController.php?view=single&id=$1 [nc,qsa]
```

RESTful Web Service 控制器

在 **.htaccess** 文件中，我们通过设置参数 'view' 来获取 RestController.php 文件中对应的请求，通过获取 'view' 不同的参数来分发到不同的方法上。**RestController.php** 文件代码如下：

实例

```
<?php
require_once("SiteRestHandler.php");
$view = "";
if(isset($_GET["view"]))
$view = $_GET["view"];
/*
* RESTful service 控制器
* URL 映射
*/
switch($view){
case "all":
// 处理 REST Url /site/list/
$siteRestHandler = new SiteRestHandler();
$siteRestHandler->getAllSites();
```

```
break;
case "single":
// 处理 REST Url /site/show/<id>/
$siteRestHandler = new SiteRestHandler();
$siteRestHandler->getSite($_GET["id"]);
break;
case "" :
//404 - not found;
break;
}
?>
```

简单的 RESTful 基础类

以下提供了 RESTful 的一个基类，用于处理响应请求的 HTTP 状态码，**SimpleRest.php** 文件代码如下：

实例

```
<?php
/*
* 一个简单的 RESTful web services 基类
* 我们可以基于这个类来扩展需求
*/
class SimpleRest {
private $httpVersion = "HTTP/1.1";
public function setHttpHeaders($contentType, $statusCode){
$statusMessage = $this -> getHttpStatusMessage($statusCode);
header($this->httpVersion. " ". $statusCode ." ". $statusMessage);
header("Content-Type:". $contentType);
}
public function getHttpStatusMessage($statusCode){
$httpStatus = array(
100 => 'Continue',
101 => 'Switching Protocols',
200 => 'OK',
201 => 'Created',
202 => 'Accepted',
203 => 'Non-Authoritative Information',
204 => 'No Content',
205 => 'Reset Content',
206 => 'Partial Content',
300 => 'Multiple Choices',
301 => 'Moved Permanently',
302 => 'Found',
303 => 'See Other',
304 => 'Not Modified',
305 => 'Use Proxy',
306 => '(Unused)',
307 => 'Temporary Redirect',
400 => 'Bad Request',
401 => 'Unauthorized',
402 => 'Payment Required',
403 => 'Forbidden',
404 => 'Not Found',
405 => 'Method Not Allowed',
406 => 'Not Acceptable',
407 => 'Proxy Authentication Required',
408 => 'Request Timeout',
409 => 'Conflict',
410 => 'Gone',
411 => 'Length Required',
412 => 'Precondition Failed',
413 => 'Request Entity Too Large',
414 => 'Request-URI Too Long',
415 => 'Unsupported Media Type',
416 => 'Requested Range Not Satisfiable',
417 => 'Expectation Failed',
500 => 'Internal Server Error',
501 => 'Not Implemented',
502 => 'Bad Gateway',
503 => 'Service Unavailable',
504 => 'Gateway Timeout',
```

```
505 => 'HTTP Version Not Supported');
return ($httpStatus[$statusCode]) ? $httpStatus[$statusCode] : $status[500];
}
}
?>
```

RESTful Web Service 处理类

以下是一个 RESTful Web Service 处理类 SiteRestHandler.php，继承了上面我们提供的 RESTful 基类，类中通过判断请求的参数来决定返回的 HTTP 状态码及数据格式，实例中我们提供了三种数据格式： "application/json" 、 "application/xml" 或 "text/html"：

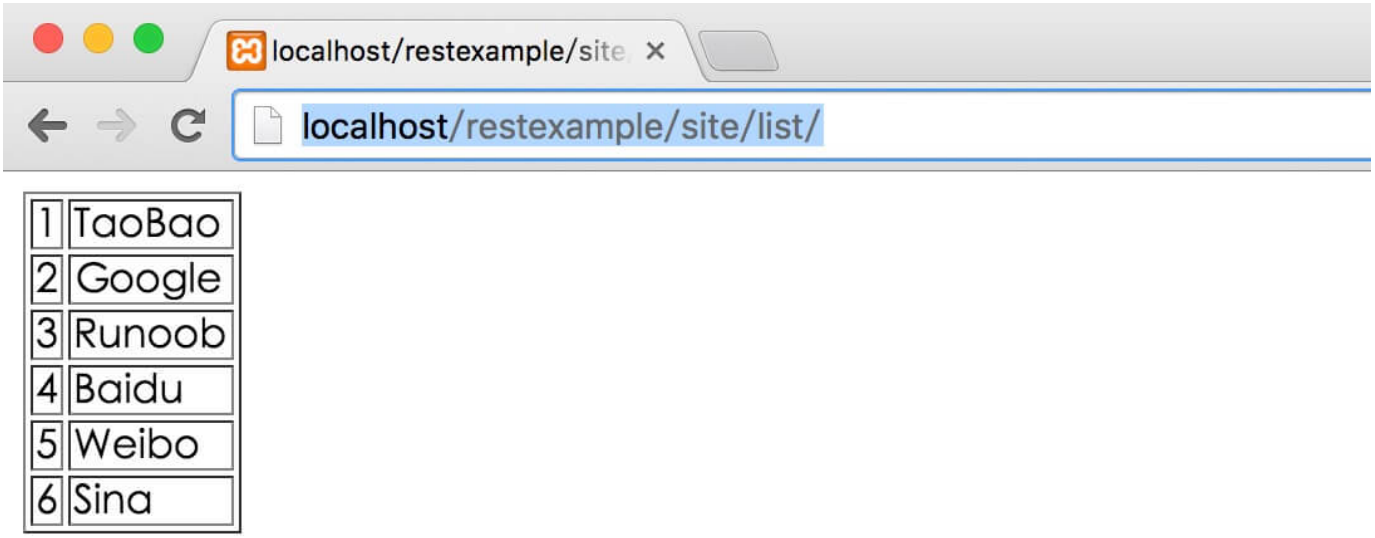
SiteRestHandler.php 文件代码如下:

实例

```
<?php
require_once("SimpleRest.php");
require_once("Site.php");
class SiteRestHandler extends SimpleRest {
function getAllSites() {
$site = new Site();
$rawData = $site->getAllSite();
if(empty($rawData)) {
$statusCode = 404;
$rawData = array('error' => 'No sites found!');
} else {
$statusCode = 200;
}
$requestContentType = $_SERVER['HTTP_ACCEPT'];
$this ->setHttpHeaders($requestContentType, $statusCode);
if(strpos($requestContentType,'application/json') !== false){
$response = $this->encodeJson($rawData);
echo $response;
} else if(strpos($requestContentType,'text/html') !== false){
$response = $this->encodeHtml($rawData);
echo $response;
} else if(strpos($requestContentType,'application/xml') !== false){
$response = $this->encodeXml($rawData);
echo $response;
}
}
public function encodeHtml($responseData) {
$htmlResponse = "<table border='1'>";
foreach($responseData as $key=>$value) {
$htmlResponse .= "<tr><td>". $key. "</td><td>". $value. "</td></tr>";
}
$htmlResponse .= "</table>";
return $htmlResponse;
}
public function encodeJson($responseData) {
$jsonResponse = json_encode($responseData);
return $jsonResponse;
}
public function encodeXml($responseData) {
// 创建 SimpleXMLElement 对象
$xml = new SimpleXMLElement('<?xml version="1.0"?><site></site>');
foreach($responseData as $key=>$value) {
$xml->addChild($key, $value);
}
return $xml->asXML();
}
public function getSite($id) {
$site = new Site();
$rawData = $site->getSite($id);
if(empty($rawData)) {
$statusCode = 404;
$rawData = array('error' => 'No sites found!');
} else {
$statusCode = 200;
}
$requestContentType = $_SERVER['HTTP_ACCEPT'];
$this ->setHttpHeaders($requestContentType, $statusCode);
```

```
if(strpos($requestContentType,'application/json') !== false){
$response = $this->encodeJson($rawData);
echo $response;
} else if(strpos($requestContentType,'text/html') !== false){
$response = $this->encodeHtml($rawData);
echo $response;
} else if(strpos($requestContentType,'application/xml') !== false){
$response = $this->encodeXml($rawData);
echo $response;
}
}
}
?>
```

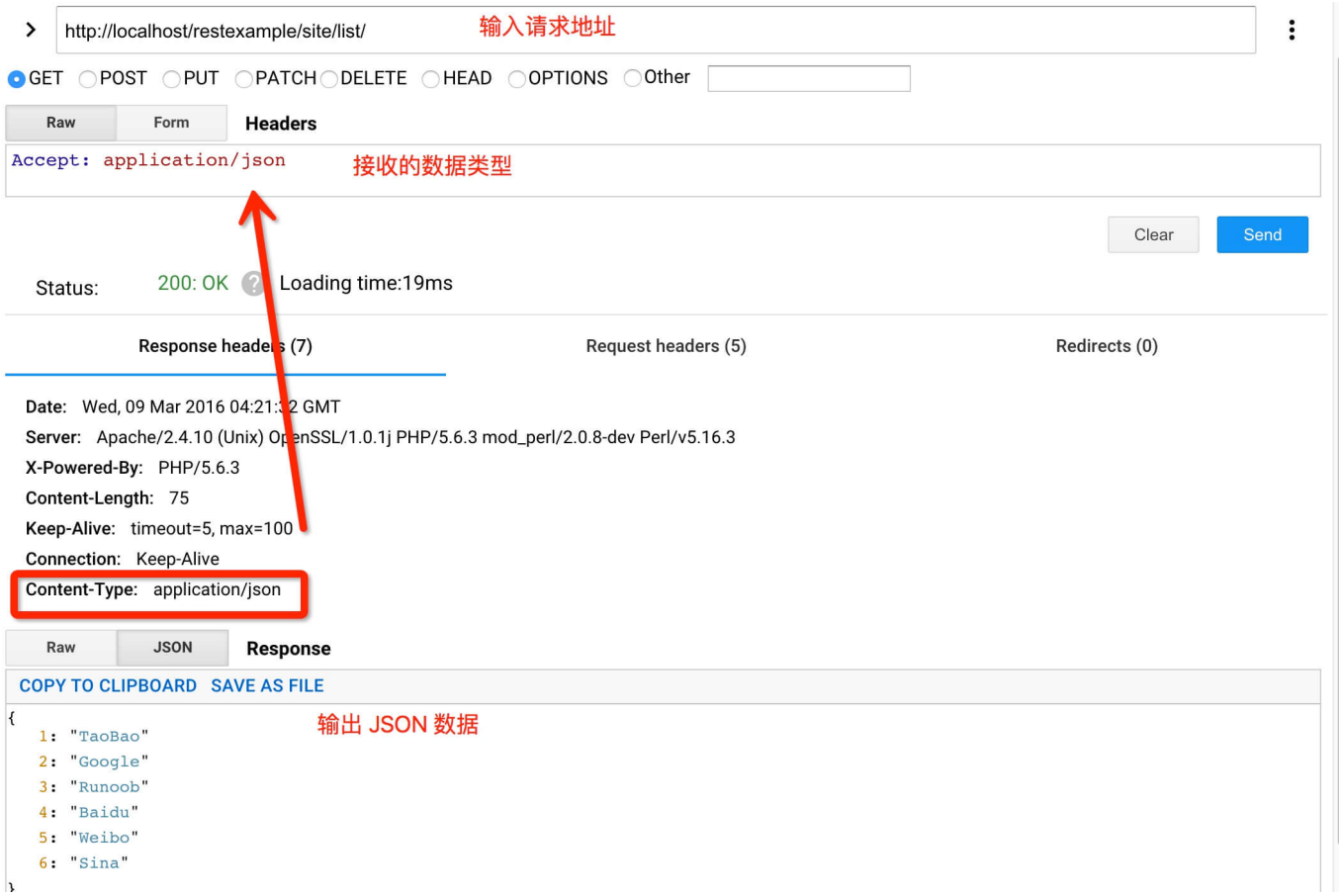
接下来我们通过 http://localhost/restexample/site/list/ 访问，输出结果如下：



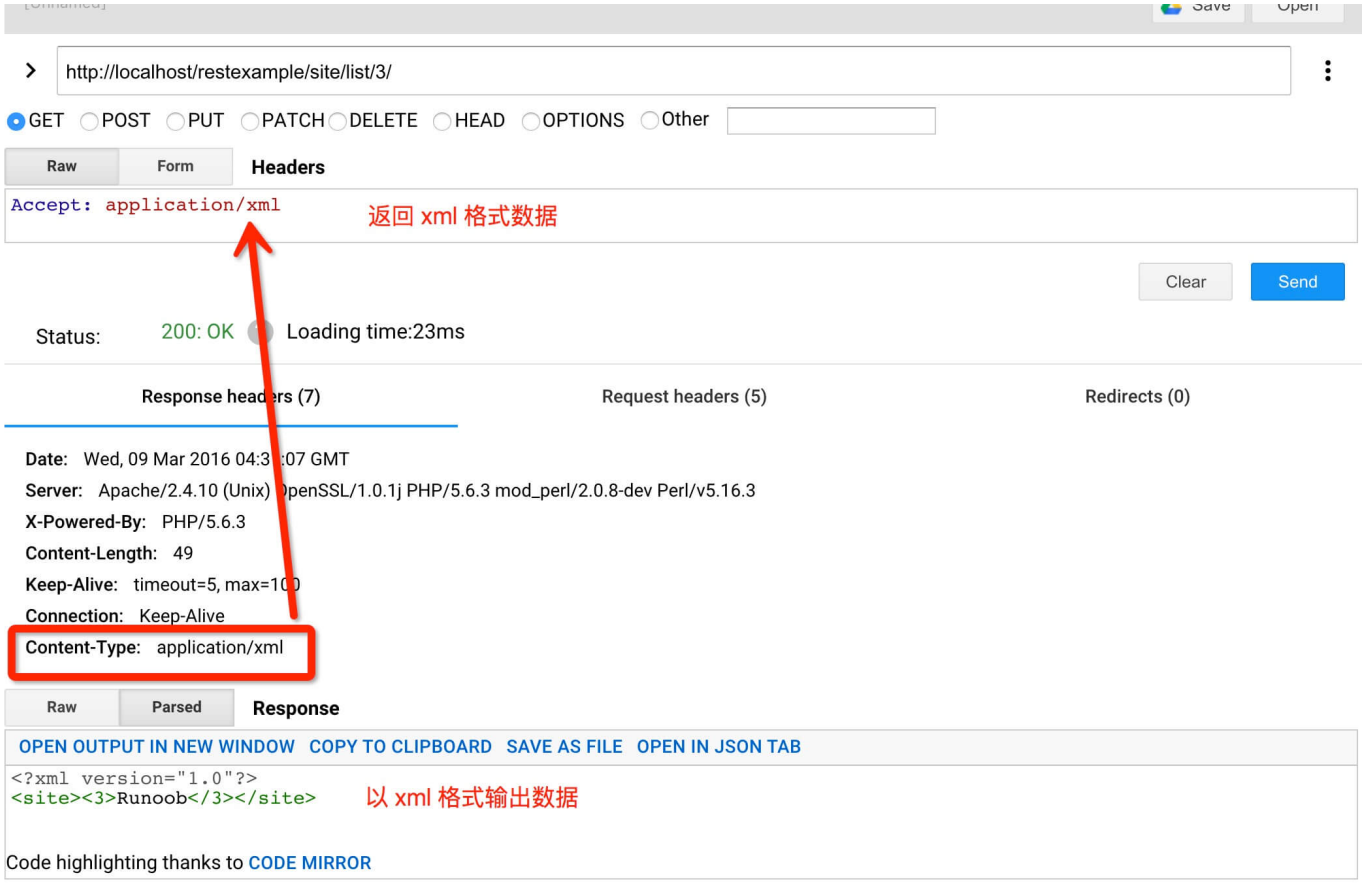
RESTful Web Service 客户端

接下来我们可以使用 Google Chrome 浏览器的 "Advance Rest Client" 作为 RESTful Web Service 客户端来请求我们的服务。

实例中请求 http://localhost/restexample/site/list/ 地址，接收数据类似为 **Accept: application/json**



请求 id 为 3 的站点 Runoob(菜鸟教程)，访问地址为 http://localhost/restexample/site/list/3/，



源码下载

实例中使用到的代码可点击以下按钮下载：

源码下载

PHP 正则表达式(PCRE)

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。

更多正则表达式的内容可参考我们的：正则表达式 - 教程。

PHP 中我们可以使用 PCRE 扩展来匹配字符串的模式。

PCRE 函数

函数	描述
preg_filter	执行一个正则表达式搜索和替换
preg_grep	返回匹配模式的数组条目
preg_last_error	返回最后一个PCRE正则执行产生的错误代码
preg_match_all	执行一个全局正则表达式匹配
preg_match	执行一个正则表达式匹配
preg_quote	转义正则表达式字符
preg_replace_callback_array	执行一个正则表达式搜索并且使用一个回调进行替换
preg_replace_callback	执行一个正则表达式搜索并且使用一个回调进行替换
preg_replace	执行一个正则表达式的搜索和替换
preg_split	通过一个正则表达式分隔字符串

PREG 常量

常量	描述	自哪个版本起
PREG_PATTERN_ORDER	结果按照"规则"排序，仅用于preg_match_all()， 即$matches[0]是完整规则的匹配结果，$matches[1]是第一个子组匹配的结果，等等。	since

常量	描述	自哪个版本起
PREG_SET_ORDER	结果按照"集合"排序，仅用于preg_match_all()， 即$matches[0]保存第一次匹配结果的所有结果(包含子组)信息, $matches[1]保存第二次的结果信息，等等。	
PREG_OFFSET_CAPTURE	查看**PREG_SPLIT_OFFSET_CAPTURE**的描述。	4.3.0
PREG_SPLIT_NO_EMPTY	这个标记告诉 preg_split() 进返回非空部分。	
PREG_SPLIT_DELIM_CAPTURE	这个标记告诉 preg_split() 同时捕获括号表达式匹配到的内容。	4.0.5
PREG_SPLIT_OFFSET_CAPTURE	如果设置了这个标记，每次出现的匹配子串的偏移量也会被返回。注意，这会改变返回数组中的值，每个元素都是由匹配子串作为第0个元素，它相对目标字符串的偏移量作为第1个元素的数组。这个 标记只能用于 preg_split()。	4.3.0
PREG_NO_ERROR	没有匹配错误时调用 preg_last_error() 返回。	5.2.0
PREG_INTERNAL_ERROR	如果有PCRE内部错误时调用 preg_last_error() 返回。	5.2.0
PREG_BACKTRACK_LIMIT_ERROR	如果调用回溯限制超出，调用preg_last_error()时返回。	5.2.0
PREG_RECURSION_LIMIT_ERROR	如果递归限制超出，调用preg_last_error()时返回。	5.2.0
PREG_BAD_UTF8_ERROR	如果最后一个错误时由于异常的utf-8数据(仅在运行在 UTF-8 模式正则表达式下可用)。 导致的，调用preg_last_error()返回。	5.2.0
PREG_BAD_UTF8_OFFSET_ERROR	如果偏移量与合法的urf-8代码不匹配(仅在运行在 UTF-8 模式正则表达式下可用)。 调用preg_last_error()返回。	5.3.0
PCRE_VERSION	PCRE版本号和发布日期(比如： "*7.0 18-Dec-2006*")。	5.2.4

□ PHP 面向对象　　PHP preg_filter() 函数 □

□ PHP EOF(heredoc) 使用说明　　PHP boolval() 函数 □

PHP 可用的函数

PHP 提供了很多可用的标准函数，下表列出了常用的几个：

函数	描述
boolval	获取变量的布尔值

debug_zval_dump	查看一个变量在zend引擎中的引用计数、类型信息
doubleval	floatval 的别名
empty	检查一个变量是否为空
floatval	获取变量的浮点值
get_defined_vars	返回由所有已定义变量所组成的数组
get_resource_type	返回资源（resource）类型
gettype	获取变量的类型
import_request_variables	将 GET / POST / Cookie 变量导入到全局作用域中
intval	获取变量的整数值
is_array	检测变量是否是数组
is_bool	检测变量是否是布尔型
is_callable	检测参数是否为合法的可调用结构
is_double	is_float 的别名
is_float	检测变量是否是浮点型
is_int	检测变量是否是整数
is_integer	is_int 的别名
is_iterable	检测变量的内容是否是一个可迭代的值
is_long	is_int 的别名
is_null	检测变量是否为 NULL
is_numeric	检测变量是否为数字或数字字符串
is_object	检测变量是否是一个对象
is_real	is_float 的别名
is_resource	检测变量是否为资源类型
is_scalar	检测变量是否是一个标量
is_string	检测变量是否是字符串
isset	检测变量是否已设置并且非 NULL
print_r	打印变量，输出易于阅读的信息。
serialize	序列化对象
settype	设置变量的类型
strval	获取变量的字符串值
unserialize	从已存储的表示中创建 PHP 的值

unset	释放给定的变量
var_dump	打印变量的相关信息
var_export	输出或返回一个变量，以字符串形式表示

密码散列算法

函数	描述
password_get_info	返回指定散列（hash）的相关信息
password_hash	创建密码的散列（hash）
password_needs_rehash	检测散列值是否匹配指定的选项
password_verify	验证密码是否和散列值匹配

PHP EOF(heredoc) 使用说明 PHP boolval() 函数

点我分享笔记

反馈/建议