PHP 语法

您无法在浏览器中通过查看源文档的方式来查看 PHP 的源代码 - 您只能看到 PHP 文件的输出，即纯粹的 HTML。这是因为在结果返回浏览器之前，脚本就已经在服务器执行了。

基本的 PHP 语法

PHP 的脚本块以 *<?php* 开始，以 *?>* 结束。您可以把 PHP 的脚本块放置在文档中的任何位置。

当然，在支持简写的服务器上，您可以使用 <? 和 ?> 来开始和结束脚本块。

不过，为了达到最好的兼容性，我们推荐您使用标准形式 (<?php)，而不是简写形式。

```
<?php

?>
```

PHP 文件通常会包含 HTML 标签，就像一个 HTML 文件，以及一些 PHP 脚本代码。

在下面，我们提供了一段简单的 PHP 脚本，它可以向浏览器输出文本 "Hello World"：

```
<html>
<body>

<?php
echo "Hello World";
?>

</body>
</html>
```

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。

有两种通过 PHP 来输出文本的基础指令：*echo* 和 *print*。在上面的例子中，我们使用了 echo 语句来输出文本 "Hello World"。

PHP 中的注释

在 PHP 中，我们使用 // 来编写单行注释，或者使用 /* 和 */ 来编写大的注释块。

```
<html>
<body>

<?php

//This is a comment

/*
This is
a comment
block
*/

?>

</body>
</html>
```

PHP 变量

变量用于存储值，比如数字、字符串或函数的结果，这样我们就可以在脚本中多次使用它们了。

PHP 中的变量

变量用于存储值，比如数字、文本字符串或数组。

一旦设置了某个变量，我们就可以在脚本中重复地使用它。

PHP 中的所有变量都是以 $ 符号开始的。

在 PHP 中设置变量的正确方法是：

```
$var_name = value;
```

PHP 的入门者往往会忘记在变量的前面的 $ 符号。如果那样做的话，变量将是无效的。

让我们试着创建一个存有字符串的变量，和一个存有数值的变量：

```
<?php
$txt = "Hello World!";
$number = 16;
?>
```

PHP 是一门松散类型的语言（Loosely Typed Language）

在 PHP 中，不需要在设置变量之前声明该变量。

在上面的例子中，您看到了，不必向 PHP 声明该变量的数据类型。

根据变量被设置的方式，PHP 会自动地把变量转换为正确的数据类型。

在强类型的编程语言中，您必须在使用前声明变量的类型和名称。

在 PHP 中，变量会在使用时被自动声明。

变量的命名规则

- 变量名必须以字母或下划线 "_" 开头。
- 变量名只能包含字母数字字符以及下划线。
- 变量名不能包含空格。如果变量名由多个单词组成，那么应该使用下划线进行分隔（比如 $my_string），或者以大写字母开头（比如 $myString）。

PHP 字符串

字符串变量用于存储并处理文本片段。

PHP 中的字符串

字符串变量用于包含字符串的值。

在本教程中，我们打算介绍几个在 PHP 中用于操作字符串的最常用的函数和运算符。

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，PHP 脚本把字符串 "Hello World" 赋值给名为 $txt 的字符串变量：

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

以上代码的输出：

```
Hello World
```

现在，让我们试着使用不同的函数和运算符来操作我们的字符串。

并置运算符（Concatenation Operator）

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。

要把两个变量连接在一起，请使用这个点运算符 (.)：

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

以上代码的输出：

```
Hello World 1234
```

您可以看到，我们在上面的例子中使用了两次并置运算符。这是由于我们需要插入第三个字符串。

为了分隔这两个变量，我们在 $txt1 与 $txt2 之间插入了一个空格。

使用 strlen() 函数

strlen() 函数用于计算字符串的长度。

让我们算出字符串 "Hello world!" 的长度：

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

以上代码的输出：

```
12
```

字符串的长度信息常常用在循环或其他函数中，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）。

使用 strpos() 函数

strpos() 函数用于在字符串内检索一段字符串或一个字符。

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

让我们试一下，是不是能在字符串中找到子字符串 "world"：

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

以上代码的输出是：

```
6
```

正如您看到的，在我们的字符串中，字符串 "world" 的位置是 6。返回 6 而不是 7，是由于字符串中的首个位置的 0，而不是 1。

完整的 PHP String 参考手册

如需完整的字符串函数参考手册，请访问我们的 PHP String 参考手册。

这个手册提供了每个函数的简要描述和实例！

PHP 运算符

运算符用于对值进行运算.

PHP 运算符

本部分列出了在 PHP 中使用的各种运算符：

算数运算符

运算符	说明	例子	结果
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

赋值运算符

运算符	说明	例子
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

比较运算符

运算符	说明	例子
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

逻辑运算符

运算符	说明	例子
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

PHP If...Else 语句

if、elseif 以及 else 语句用于执行基于不同条件的不同动作。

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。

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

elseif 语句

与 if...else 配合使用，在若干条件之一成立时执行一个代码块

If...Else 语句

如果您希望在某个条件成立时执行一些代码，在条件不成立时执行另一些代码，请使用 if....else 语句。

语法

```
if (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

实例

如果当前日期是周五，下面的代码将输出 "Have a nice weekend!"，否则会输出 "Have a nice day!"：

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

如果需要在条件成立或不成立时执行多行代码，应该把这些代码行包括在花括号中：

```
<html>
<body>
```

```
<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

ElseIf 语句

如果希望在多个条件之一成立时执行代码，请使用 elseif 语句：

语法

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

实例

如果当前日期是周五，下面的例子会输出 "Have a nice weekend!"，如果是周日，则输出 "Have a nice Sunday!"，否则输出 "Have a nice day!"：

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Switch 语句

PHP 中的 Switch 语句用于执行基于多个不同条件的不同动作。

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。

使用 Switch 语句可以避免冗长的 if..elseif..else 代码块。

语法

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

实例

工作原理：

1. 对表达式（通常是变量）进行一次计算
2. 把表达式的值与结构中 case 的值进行比较
3. 如果存在匹配，则执行与 case 关联的代码
4. 代码执行后，*break 语句*阻止代码跳入下一个 case 中继续执行
5. 如果没有 case 为真，则使用 default 语句

```
<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

PHP 数组

数组能够在单独的变量名中存储一个或多个值。

什么是数组？

在使用 PHP 进行开发的过程中，或早或晚，您会需要创建许多相似的变量。

无需很多相似的变量，你可以把数据作为元素存储在数组中。

数组中的元素都有自己的 ID，因此可以方便地访问它们。

有三种数组类型：

数值数组
　　带有数字 ID 键的数组
关联数组
　　数组中的每个 ID 键关联一个值
多维数组
　　包含一个或多个数组的数组

数值数组

数值数组存储的每个元素都带有一个数字 ID 键。

可以使用不同的方法来创建数值数组：

例子 1

在这个例子中，会自动分配 ID 键：

```
$names = array("Peter","Quagmire","Joe");
```

例子 2

在这个例子中，我们人工分配的 ID 键：

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

可以在脚本中使用这些 ID 键：

```
<?php

$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";

echo $names[1] . " and " . $names[2] . " are ". $names[0] . "'s neighbors";
?>
```

以上代码的输出：

```
Quagmire and Joe are Peter's neighbors
```

关联数组

关联数组，它的每个 ID 键都关联一个值。

在存储有关具体命名的值的数据时，使用数值数组不是最好的做法。

通过关联数组，我们可以把值作为键，并向它们赋值。

例子 1

在本例中，我们使用一个数组把年龄分配给不同的人：

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

例子 2

本例与例子 1 相同，不过展示了另一种创建数组的方法：

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

可以在脚本中使用 ID 键：

```
<?php

$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

以上脚本的输出：

```
Peter is 32 years old.
```

多维数组

在多维数组中，主数组中的每个元素也是一个数组。在子数组中的每个元素也可以是数组，以此类推。

例子 1

在本例中，我们创建了一个带有自动分配的 ID 键的多维数组：

```
$families = array
(
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
```

```
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
);
```

如果输出这个数组的话，应该类似这样：

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

例子 2

让我们试着显示上面的数组中的一个单一的值：

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

以上代码的输出：

```
Is Megan a part of the Griffin family?
```

PHP 循环

PHP 中的循环语句用于执行相同的代码块指定的次数。

循环

在您编写代码时，您经常需要让相同的代码块运行很多次。您可以在代码中使用循环语句来完成这个任务。

在 PHP 中，我们可以使用下列循环语句：

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 语句

只要指定的条件成立，while 语句将重复执行代码块。

语法

```
while (condition)
code to be executed;
```

例子

下面的例子示范了一个循环，只要变量 i 小于或等于 5，代码就会一直循环执行下去。循环每循环一次，变量就会递增 1：

```
<html>
<body>

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

</body>
</html>
```

do...while 语句

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

语法

```
do
{
```

```
code to be executed;
}
while (condition);
```

例子

下面的例子将对 i 的值进行一次累加，然后，只要 i 小于 5 的条件成立，就会继续累加下去：

```
<html>
<body>

<?php
$i=0;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<5);
?>

</body>
</html>
```

for 语句

如果您已经确定了代码块的重复执行次数，则可以使用 for 语句。

语法

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

注释：for 语句有三个参数。第一个参数初始化变量，第二个参数保存条件，第三个参数包含执行循环所需的增量。如果 initialization 或 increment 参数中包括了多个变量，需要用逗号进行分隔。而条件必须计算为 true 或者 false。

例子

下面的例子会把文本 "Hello World!" 显示 5 次：

```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!<br />";
}
?>

</body>
</html>
```

foreach 语句

foreach 语句用于循环遍历数组。

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） - 以此类推。

语法

```
foreach (array as value)
{
    code to be executed;
}
```

例子

下面的例子示范了一个循环，这个循环可以输出给定数组的值：

```
<html>
<body>

<?php
$arr=array("one", "two", "three");

foreach ($arr as $value)
{
  echo "Value: " . $value . "<br />";
}
?>

</body>
</html>
```

PHP 函数

PHP 的真正威力源自于它的函数。

在 PHP 中，提供了超过 700 个内建的函数。

PHP 函数

在本教程中，我们将为您讲解如何创建自己的函数。

如需内建函数的参考和实例，请访问我们的 PHP 参考手册。

创建 PHP 函数

函数是一种可以在任何被需要的时候执行的代码块。

创建 PHP 函数：

1. 所有的函数都使用关键词 "function()" 来开始
2. 命名函数 - 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。
3. 添加 "{" - 开口的花括号之后的部分是函数的代码。
4. 插入函数代码
5. 添加一个 "}" - 函数通过关闭花括号来结束。

例子

一个简单的函数，在其被调用时能输出我的名称的：

```
<html>
<body>

<?php
function writeMyName()
  {
  echo "David Yang";
  }

writeMyName();
?>

</body>
</html>
```

使用 PHP 函数

现在，我们要在 PHP 脚本中使用这个函数了：

```
<html>
<body>

<?php
function writeMyName()
  {
  echo "David Yang";
  }

echo "Hello world!<br />";
echo "My name is ";
writeMyName();
echo ".<br />That's right, ";
writeMyName();
echo " is my name.";
```

```
?>

</body>
</html>
```

以上代码的输出：

```
Hello world!
My name is David Yang.
That's right, David Yang is my name.
```

PHP 函数 - 添加参数

我们的第一个函数是一个非常简单的函数。它只能输出一个静态的字符串。

通过可以添加参数，我们向函数添加更多的功能。参数类似一个变量。

您可能注意到了，函数名称后面有一个括号，比如 writeMyName()。参数就是在括号中规定的。

例子 1

下面的例子讲输出不同的名字，但姓是相同的：

```
<html>
<body>

<?php
function writeMyName($fname)
  {
  echo $fname . " Yang.<br />";
  }

echo "My name is ";
writeMyName("David");

echo "My name is ";
writeMyName("Mike");

echo "My name is ";
writeMyName("John");
?>

</body>
</html>
```

上面的代码的输出：

```
My name is David Yang.
My name is Mike Yang.
My name is John Yang.
```

例子 2

下面的函数有两个参数：

```
<html>
<body>
```

```
<?php
function writeMyName($fname,$punctuation)
  {
  echo $fname . " Yang" . $punctuation . "<br />";
  }

echo "My name is ";
writeMyName("David",".");

echo "My name is ";
writeMyName("Mike","!");

echo "My name is ";
writeMyName("John","...");
?>

</body>
</html>
```

上面的代码的输出：

```
My name is David Yang.
My name is Mike Yang!
My name is John Yang...
```

PHP 函数 - 返回值

函数也能用于返回值。

例子

```
<html>
<body>

<?php
function add($x,$y)
  {
  $total = $x + $y;
  return $total;
  }

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

以上代码的输出：

```
1 + 16 = 17
```

PHP 表单和用户输入

PHP 的 $_GET 和 $_POST 用于检索表单中的值，比如用户输入。

PHP 表单处理

表单实例：

```
<html>
<body>

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

</body>
</html>
```

上面的 HTML 页面实例包含了两个输入框和一个提交按钮。当用户填写该表单并单击提交按钮时，表单的数据会被送往 "welcome.php" 这个文件。

"welcome.php" 文件类似这样：

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

上面这个脚本的输出样本类似这样：

```
Welcome John.
You are 28 years old.
```

我们将在下一章解释 PHP $_GET 和 $_POST。

表单验证

应该在任何可能的时候对用户输入进行验证。客户端的验证速度更快，并且可以减轻服务器的负载。

不过，任何流量很高以至于不得不担心服务器资源的站点，也有必要担心站点的安全性。如果表单访问的是数据库，就非常有必要采用服务器端的验证。

在服务器验证表单的一种好的方式是，把表单传给它自己，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 $_GET

$_GET 变量用于收集来自 method="get" 的表单中的值。

$_GET 变量

$_GET 变量是一个数组，内容是由 HTTP GET 方法发送的变量名称和值。

$_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），并且对发送的信息量也有限制（最多 100 个字符）。

例子

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

当用户点击提交按钮时，发送的 URL 会类似这样：

```
http://www.w3school.com.cn/welcome.php?name=Peter&age=37
```

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

```
Welcome <?php echo $_GET["name"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

为什么使用 $_GET?

注释：在使用 $_GET 变量时，所有的变量名和值都会显示在 URL 中。所以在发送密码或其他敏感信息时，不应该使用这个方法。不过，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；值是不能超过 100 个字符的。

$_REQUEST 变量

PHP 的 $_REQUEST 变量包含了 $_GET, $_POST 以及 $_COOKIE 的内容。

PHP 的 $_REQUEST 变量可用来取得通过 GET 和 POST 方法发送的表单数据的结果。

例子

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
You are <?php echo $_REQUEST["age"]; ?> years old!
```

PHP $_POST

$_POST 变量用于收集来自 method="post" 的表单中的值。

$_POST 变量

$_POST 变量是一个数组，内容是由 HTTP POST 方法发送的变量名称和值。

$_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（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。

例子

```
<form action="welcome.php" method="post">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age" />
<input type="submit" />
</form>
```

当用户点击提交按钮，URL 不会含有任何表单数据，看上去类似这样：

```
http://www.w3school.com.cn/welcome.php
```

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

```
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old!
```

为什么使用 $_POST？

- 通过 HTTP POST 发送的变量不会显示在 URL 中。
- 变量没有长度限制。

不过，由于变量不显示在 URL 中，所有无法把页面加入书签。

$_REQUEST 变量

PHP 的 $_REQUEST 变量包含了 $_GET, $_POST 以及 $_COOKIE 的内容。

PHP 的 $_REQUEST 变量可用来取得通过 GET 和 POST 方法发送的表单数据的结果。

例子

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
You are <?php echo $_REQUEST["age"]; ?> years old!
```

PHP Date()

PHP 的 date() 函数用于格式化时间或日期。

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。

语法

```
date(format,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 日期 - 什么是时间戳（Timestamp）？

时间戳是自 1970 年 1 月 1 日（00:00:00 GMT）以来的秒数。它也被称为 Unix 时间戳（Unix Timestam）。

PHP 日期 - 格式化日期

date() 函数的第一个参数规定了如何格式化日期/时间。它使用字母来表示日期和时间的格式。这里列出了一些可用的字母：

- d - 月中的天 (01-31)
- m - 当前月，以数字计 (01-12)
- Y - 当前的年（四位数）

您可以在我们的 PHP Date 参考手册中，找到格式参数中可以使用的所有字母。

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

以上代码的输出类似这样：

```
2006/07/11
2006.07.11
2006-07-11
```

PHP 日期 - 添加时间戳

date() 函数的第二个参数规定了一个时间戳。此参数是可选的。如果您没有提供时间戳，当前的时间将被使用。

在我们的例子中，我们将使用 mktime() 函数为明天创建一个时间戳。

mktime() 函数可为指定的日期返回 Unix 时间戳。

语法

```
mktime(hour,minute,second,month,day,year,is_dst)
```

如需获得某一天的时间戳，我们只要设置 mktime() 函数的 day 参数就可以了：

```
<?php
$tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));
echo "Tomorrow is ".date("Y/m/d", $tomorrow);
?>
```

以上代码的输出类似这样：

```
明天是 2006/07/12
```

PHP 日期 - 参考手册

如需更多有关 PHP 日期函数的信息，请访问 W3School 为您提供的 PHP Date 参考手册。

PHP 引用文件

服务器端引用 (SSI) 用于创建可在多个页面重复使用的函数、页眉、页脚或元素。

服务器端引用（Server Side Includes）

通过 include() 或 require() 函数，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。除了它们处理错误的方式不同之外，这两个函数在其他方面都是相同的。include() 函数会生成一个警告（但是脚本会继续执行），而 require() 函数会生成一个致命错误（fatal error）（在错误发生后脚本会停止执行）。

这两个函数用于创建可在多个页面重复使用的函数、页眉、页脚或元素。

这会为开发者节省大量的时间。这意味着您可以创建供所有网页引用的标准页眉或菜单文件。当页眉需要更新时，您只更新一个包含文件就可以了，或者当您向网站添加一张新页面时，仅仅需要修改一下菜单文件（而不是更新所有网页中的链接）。

include() 函数

include() 函数可获得指定文件中的所有文本，并把文本拷贝到使用 include 函数的文件中。

例子 1

假设您拥有一个标准的页眉文件，名为 "header.php"。如需在页面中引用这个页眉文件，请使用 include() 函数，就像这样：

```
<html>
<body>

<?php include("header.php"); ?>

<h1>Welcome to my home page</h1>

<p>Some text</p>

</body>
</html>
```

例子 2

现在，假设我们有一个在所有页面上使用的标准菜单文件。请看下面这个 "menu.php"：

```
<html>
<body>

<a href="http://www.w3school.com.cn/default.php">Home</a> |
<a href="http://www.w3school.com.cn/about.php">About Us</a> |
<a href="http://www.w3school.com.cn/contact.php">Contact Us</a>
```

三个文件，"default.php"、"about.php" 以及 "contact.php" 都引用了 "menu.php" 文件。这是 "default.php" 中的代码：

```
<?php include("menu.php"); ?>
```

```
<h1>Welcome to my home page</h1>

<p>Some text</p>

</body>
</html>
```

如果您在浏览器中查看 "default.php" 的源代码，应该类似这样：

```
<html>
<body>

<a href="default.php">Home</a> |
<a href="about.php">About Us</a> |
<a href="contact.php">Contact Us</a>

<h1>Welcome to my home page</h1>
<p>Some text</p>

</body>
</html>
```

同时，当然，我们也将用相同的方法处理 "about.php" 和 "contact.php"。通过使用引用文件，在您需要重命名链接、更改链接顺序或向站点添加另一张网页时，只要简单地更新 "menu.php" 文件中的文本即可。

require() 函数

require() 函数与 include() 相同，不同的是它对错误的处理方式。

include() 函数会生成一个警告（但是脚本会继续执行），而 require() 函数会生成一个致命错误（fatal error）（在错误发生后脚本会停止执行）。

如果在您通过 include() 引用文件时发生了错误，会得到类似下面这样的错误消息：

PHP 代码：

```
<html>
<body>

<?php
include("wrongFile.php");
echo "Hello World!";
?>

</body>
</html>
```

错误消息：

```
Warning: include(wrongFile.php) [function.include]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Warning: include() [function.include]:
Failed opening 'wrongFile.php' for inclusion
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```

```
Hello World!
```

请注意，echo 语句依然被执行了！这是因为警告不会中止脚本的执行。

现在，让我们使用 require() 函数运行相同的例子。

PHP 代码：

```
<html>
<body>

<?php
require("wrongFile.php");
echo "Hello World!";
?>

</body>
</html>
```

错误消息：

```
Warning: require(wrongFile.php) [function.require]:
failed to open stream:
No such file or directory in C:\home\website\test.php on line 5

Fatal error: require() [function.require]:
Failed opening required 'wrongFile.php'
(include_path='.;C:\php5\pear')
in C:\home\website\test.php on line 5
```

由于在致命错误发生后终止了脚本的执行，因此 echo 语句不会执行。

正因为在文件不存在或被重命名后脚本不会继续执行，因此我们推荐使用 require() 而不是 include()。

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。打开并向文件文件的末端进行写操作，如果文件不存在，则创建新文件。
a+	读/追加。通过向文件末端写内容，来保持文件内容。
x	只写。创建新文件。如果文件以存在，则返回 FALSE。
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() 无法打开指定文件，则返回 0 (false)。

例子

如果 fopen() 不能打开指定的文件，下面的例子会生成一段消息：

```
<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");

//some code to be executed

fclose($file);
?>
```

检测 End-of-file

feof() 函数检测是否已达到文件的末端 (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 以及 x 模式，您无法读取打开的文件!

```
if (feof($file)) echo "End of file";
```

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。

例子

下面的例子逐行读取文件，直到文件末端为止：

```
<?php
$file = fopen("welcome.txt", "r") or exit("Unable to open file!");
//Output a line of the file until the end is reached
while(!feof($file))
  {
  echo fgets($file). "<br />";
  }
fclose($file);
?>
```

逐字符读取文件

fgetc() 函数用于从文件逐字符地读取文件。

注释：在调用该函数之后，文件指针会移动到下一个字符。

例子

下面的例子逐字符地读取文件，直到文件末端为止：

```
<?php
$file=fopen("welcome.txt","r") or exit("Unable to open file!");
while (!feof($file))
  {
  echo fgetc($file);
  }
fclose($file);
?>
```

PHP Filesystem 参考手册

如需完整的 PHP 文件系统参考手册，请访问 W3School 提供的 PHP Filesystem 参考手册。

PHP 文件上传

通过 PHP，可以把文件上传到服务器。

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>
<body>

<form action="upload_file.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" />
<br />
<input type="submit" name="submit" value="Submit" />
</form>

</body>
</html>
```

请留意如下有关此表单的信息：

<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 "Error: " . $_FILES["file"]["error"] . "<br />";
  }
else
  {
  echo "Upload: " . $_FILES["file"]["name"] . "<br />";
  echo "Type: " . $_FILES["file"]["type"] . "<br />";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
  echo "Stored in: " . $_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"。就像这样：

- $_FILES["file"]["name"] - 被上传文件的名称

- $_FILES["file"]["type"] - 被上传文件的类型
- $_FILES["file"]["size"] - 被上传文件的大小，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，您应当增加有关什么用户有权上传文件的限制。

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 或 .jpeg 文件，文件大小必须小于 20 kb：

```
<?php

if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
  {
  if ($_FILES["file"]["error"] > 0)
    {
    echo "Error: " . $_FILES["file"]["error"] . "<br />";
    }
  else
    {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
    }
  }
else
  {
  echo "Invalid file";
  }

?>
```

注释：对于 IE，识别 jpg 文件的类型必须是 pjpeg，对于 FireFox，必须是 jpeg。

保存被上传的文件

上面的例子在服务器的 PHP 临时文件夹创建了一个被上传文件的临时副本。

这个临时的复制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
if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
  {
  if ($_FILES["file"]["error"] > 0)
    {
    echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
    }
```

```
  else
    {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
    echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";

    if (file_exists("upload/" . $_FILES["file"]["name"]))
      {
      echo $_FILES["file"]["name"] . " already exists. ";
      }
    else
      {
      move_uploaded_file($_FILES["file"]["tmp_name"],
      "upload/" . $_FILES["file"]["name"]);
      echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
      }
    }
  }
else
  {
  echo "Invalid file";
  }
?>
```

上面的脚本检测了是否已存在此文件，如果不存在，则把文件拷贝到指定的文件夹。

注释：这个例子把文件保存到了名为 "upload" 的新文件夹。

PHP Cookies

cookie 常用于识别用户。

什么是 Cookie？

cookie 常用于识别用户。cookie 是服务器留在用户计算机中的小文件。每当相同的计算机通

过浏览器请求页面时，它同时会发送 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);
```

例子

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

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

<html>
<body>

</body>
</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）。

如何取回 Cookie 的值？

PHP 的 $_COOKIE 变量用于取回 cookie 的值。

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

```
<?php
// Print a cookie
echo $_COOKIE["user"];

// A way to view all cookies
print_r($_COOKIE);
?>
```

在下面的例子中，我们使用 isset() 函数来确认是否已设置了 cookie：

```
<html>
<body>

<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<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。

删除的例子：

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time()-3600);
?>
```

如果浏览器不支持 cookie 该怎么办？

如果您的应用程序涉及不支持 cookie 的浏览器，您就不得不采取其他方法在应用程序中从一张页面向另一张页面传递信息。一种方式是从表单传递数据（有关表单和用户输入的内容，稍早前我们已经在本教程中介绍过了）。

下面的表单在用户单击提交按钮时向 "welcome.php" 提交了用户输入：

```
<html>
<body>

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

</body>
</html>
```

取回 "welcome.php" 中的值，就像这样：

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

PHP Sessions

PHP session 变量用于存储有关用户会话的信息，或更改用户会话的设置。Session 变量保存的信息是单一用户的，并且可供应用程序中的所有页面使用。

PHP Session 变量

当您运行一个应用程序时，您会打开它，做些更改，然后关闭它。这很像一次会话。计算机清楚你是谁。它知道你何时启动应用程序，并在何时终止。但是在因特网上，存在一个问题：服务器不知道你是谁以及你做什么，这是由于 HTTP 地址不能维持状态。

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

Session 的工作机制是：为每个访问者创建一个唯一的 id (UID)，并基于这个 UID 来存储变量。UID 存储在 cookie 中，亦或通过 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();
// store session data
$_SESSION['views']=1;
?>

<html>
<body>

<?php
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>

</body>
</html>
```

输出：

```
Pageviews=1
```

在下面的例子中，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，则我们

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 "Views=". $_SESSION['views'];
?>
```

终结 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
unset($_SESSION['views']);
?>
```

您也可以通过 session_destroy() 函数彻底终结 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 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。
headers	可选。规定附加的标题，比如 From、Cc 以及 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 需要一个已安装且正在运行的邮件系统，以便使邮件函数可用。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。

在下面的例子中，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";
$subject = "Test mail";
$message = "Hello! This is a simple email message.";
$from = "someonelse@example.com";
$headers = "From: $from";
mail($to,$subject,$message,$headers);
echo "Mail Sent.";

?>
```

PHP Mail Form

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

```
<html>
<body>

<?php
if (isset($_REQUEST['email']))
//if "email" is filled out, send email
  {
  //send email
  $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
//if "email" is not filled out, display the form
  {
  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>
```

例子解释：

1. 首先，检查是否填写了邮件输入框
2. 如果未填写（比如在页面被首次访问时），输出 HTML 表单
3. 如果已填写（在表单被填写后），从表单发送邮件
4. 当点击提交按钮后，重新载入页面，显示邮件发送成功的消息

PHP Mail 参考手册

如需更多有关 PHP mail() 函数的信息，请访问我们的 PHP Mail 参考手册。

PHP 安全的电子邮件

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，请看上一节中的 PHP 代码：

```
<html>
<body>

<?php
if (isset($_REQUEST['email']))
//if "email" is filled out, send email
  {
  //send email
  $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
//if "email" is not filled out, display the form
  {
  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这些文本，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: 以及 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。

下面的代码与上一节类似，不过我们已经增加了检测表单中 email 字段的输入验证程序：

```
<html>
<body>
<?php
function spamcheck($field)
  {
  //filter_var() sanitizes the e-mail
  //address using FILTER_SANITIZE_EMAIL
```

```
  $field=filter_var($field, FILTER_SANITIZE_EMAIL);

  //filter_var() validates the e-mail
  //address using FILTER_VALIDATE_EMAIL
  if(filter_var($field, FILTER_VALIDATE_EMAIL))
    {
    return TRUE;
    }
  else
    {
    return FALSE;
    }
  }

if (isset($_REQUEST['email']))
  {//if "email" is filled out, proceed

  //check if the email address is invalid
  $mailcheck = spamcheck($_REQUEST['email']);
  if ($mailcheck==FALSE)
    {
    echo "Invalid input";
    }
  else
    {//send email
    $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
  {//if "email" is not filled out, display the form
  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 从字符串中删除电子邮件的非法字符
- FILTER_VALIDATE_EMAIL 验证电子邮件地址

您可以在我们的 PHP 过滤器这一节中阅读更多有关过滤器的内容。

PHP 错误处理

在 PHP 中，默认的错误处理很简单。一条消息会被发送到浏览器，这条消息带有文件名、行号以及一条描述错误的消息。

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

第一个例子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，您会获得类似这样的错误：

```
Warning: fopen(welcome.txt) [function.fopen]: failed to open stream:
No such file or directory in C:\webfolder\test.php on line 2
```

为了避免用户获得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("File not found");
 }
else
 {
 $file=fopen("welcome.txt","r");
 }
?>
```

现在，假如文件不存在，您会得到类似这样的错误消息：

```
File not found
```

比起之前的代码，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。

不过，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 以及 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。必须是一个值数。 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。规定错误在其中发生的文件名。
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错误处理程序旨在处理的错误的不同的类型：

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。 脚本发现可能有错误发生，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，除级别 E_STRICT 以外。 （在 PHP 6.0，E_STRICT 是 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 "Ending Script";
 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。

Set Error Handler

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，这样脚本就可以不同的方式来处理不同的错误。不过，在本例中，我们打算针对所有错误来使用我们的自定义错误处理程序：

```
set_error_handler("customError");
```

由于我们希望我们的自定义函数来处理所有错误，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
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr";
 }

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

以上代码的输出应该类似这样：

```
Custom 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的很有用的。在 PHP 中，这个任务由 trigger_error() 完成。

例子

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("Value must be 1 or below");
}
?>
```

以上代码的输出应该类似这样：

```
Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6
```

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。脚本发现了可能的错误，也有可能在脚本运行正常时发生。

例子

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，我们将使用我们的自定义错误处理程序并结束脚本：

```
<?php
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr<br />";
 echo "Ending Script";
 die();
 }

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
 {
 trigger_error("Value must be 1 or below",E_USER_WARNING);
 }
?>
```

以上代码的输出应该类似这样：

```
Error: [512] Value must be 1 or below
Ending Script
```

现在，我们已经学习了如何创建自己的 error，以及如何处罚它们，现在我们研究一下错误记录。

错误记录

默认地，根据在 php.ini 中的 error_log 配置，PHP 向服务器的错误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
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr<br />";
 echo "Webmaster has been notified";
 error_log("Error: [$errno] $errstr",1,
 "someone@example.com","From: webmaster@example.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
```

```
 {
 trigger_error("Value must be 1 or below",E_USER_WARNING);
 }
?>
```

以上代码的输出应该类似这样：

```
Error: [512] Value must be 1 or below
Webmaster has been notified
```

接收自以上代码的邮件类似这样：

```
Error: [512] Value must be 1 or below
```

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 异常处理

异常（Exception）用于在指定的错误发生时改变脚本的正常流程。

什么是异常？

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

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
//create function with an exception
function checkNum($number)
 {
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//trigger exception
checkNum(2);
?>
```

上面的代码会获得类似这样的一个错误：

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

Try, throw 和 catch

要避免上面例子出现的错误，我们需要创建适当的代码来处理异常。

处理处理程序应当包括：

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 - 这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
//创建可抛出一个异常的函数
function checkNum($number)
 {
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//在 "try" 代码块中触发异常
try
 {
 checkNum(2);
```

```
 //If the exception is thrown, this text will not be shown
 echo 'If you see this, the number is 1 or below';
 }

//捕获异常
catch(Exception $e)
 {
 echo 'Message: ' .$e->getMessage();
 }
?>
```

上面代码将获得类似这样一个错误：

```
Message: Value must be 1 or below
```

例子解释：

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

不过，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 exception 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 exception 类：

```
<?php
class customException extends Exception
 {
 public function errorMessage()
  {
  //error message
  $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
  .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
  return $errorMsg;
  }
 }

$email = "someone@example...com";

try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
```

```
  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() 以及 getMessage()。

例子解释：

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
class customException extends Exception
{
public function errorMessage()
{
//error message
$errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
return $errorMsg;
}
}

$email = "someone@example.com";

try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
  throw new customException($email);
  }
 //check for "example" in mail address
 if(strpos($email, "example") !== FALSE)
  {
```

```
  throw new Exception("$email is an example e-mail");
  }
 }

catch (customException $e)
 {
 echo $e->errorMessage();
 }

catch(Exception $e)
 {
 echo $e->getMessage();
 }
?>
```

例子解释：

上面的代码测试了两种条件，如何任何条件不成立，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. 执行 "try" 代码块，在第一个条件下，不会抛出异常。
4. 由于 e-mail 含有字符串 "example"，第二个条件会触发异常。
5. "catch" 代码块会捕获异常，并显示恰当的错误消息

如果没有捕获 customException，紧紧捕获了 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()
  {
  //error message
  $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
  return $errorMsg;
  }
 }

$email = "someone@example.com";

try
 {
 try
  {
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE)
   {
   //throw exception if email is not valid
   throw new Exception($email);
```

```
   }
  }
 catch(Exception $e)
  {
  //re-throw exception
  throw new customException($email);
  }
 }

catch (customException $e)
 {
 //display custom message
 echo $e->errorMessage();
 }
?>
```

例子解释：

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 变量设置为一个有效的邮件地址，但含有字符串 "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" 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。

如果在其目前的 "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 （Top Level Exception Handler）

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');
?>
```

以上代码的输出应该类似这样：

```
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 代码块中再次抛出（re-thrown）异常。

简而言之：如果抛出了异常，就必须捕获它。

PHP 过滤器（Filter）

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。

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。

设计 PHP 的过滤器扩展的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 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，您能够确保应有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
- 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

在下面的例子中，我们用 filter_var() 函数验证了一个整数：

```
<?php
```

```
$int = 123;

if(!filter_var($int, FILTER_VALIDATE_INT))
 {
 echo("Integer is not valid");
 }
else
 {
 echo("Integer is valid");
 }
?>
```

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，因此代码的输出是："Integer is valid"。

假如我们尝试使用一个非整数的变量，则输出是："Integer is not valid"。

如需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）
- 返回若成功预期的类型，否则返回 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。

在下面的例子中，我们用 filter_var() 和 "min_range" 以及 "max_range" 选项验证了一个整数：

```
<?php
$var=300;

$int_options = array(
"options"=>array
 (
 "min_range"=>0,
 "max_range"=>256
 )
);

if(!filter_var($var, FILTER_VALIDATE_INT, $int_options))
 {
 echo("Integer is not valid");
```

```
 }
else
 {
 echo("Integer is valid");
 }
?>
```

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"，它不在指定的氛围内，以上代码的输出将是 "Integer is not valid"。

如需完整的函数及过滤器列表，请访问 W3School 提供的 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。

我们需要作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。

在下面的例子中，输入变量 "email" 被传到 PHP 页面：

```
<?php
if(!filter_has_var(INPUT_GET, "email"))
 {
 echo("Input type does not exist");
 }
else
 {
 if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL))
  {
  echo "E-Mail is not valid";
  }
 else
  {
  echo "E-Mail is valid";
  }
 }
?>
```

例子解释：

上面的例子有一个通过 "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邮件地址

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。

在下面的例子中，输入变量 "url" 被传到 PHP 页面：

```
<?php
if(!filter_has_var(INPUT_POST, "url"))
 {
 echo("Input type does not exist");
 }
else
 {
 $url = filter_input(INPUT_POST,
 "url", FILTER_SANITIZE_URL);
 }
?>
```

例子解释：

上面的例子有一个通过 "POST" 方法传送的输入变量 (url)：

1. 检测是否存在 "POST" 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

假如输入变量类似这样："http://www.W3#$%S^%$#ool.com.cn/"，则净化后的 $url 变量应该是这样的：

```
http://www.W3School.com.cn/
```

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 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 变量是一个名称、一个年龄以及一个邮件地址：

```
<?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,
 );

$result = filter_input_array(INPUT_GET, $filters);

if (!$result["age"])
 {
 echo("Age must be a number between 1 and 120.<br />");
 }
elseif(!$result["email"])
 {
 echo("E-Mail is not valid.<br />");
```

```
 }
else
 {
 echo("User input is valid");
 }
?>
```

例子解释：

上面的例子有三个通过 "GET" 方法传送的输入变量 (name, age and email)

1. 设置一个数组，其中包含了输入变量的名称，以及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，）

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、标志以及选项的数组

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，也可以使用已有的 PHP 函数。

规定您准备用到过滤器的函数，与规定选项的方法相同。

在下面的例子中，我们使用了一个自定义的函数把所有 "_" 转换为空格：

```
<?php
function convertSpace($string)
{
return str_replace("_", " ", $string);
}

$string = "Peter_is_a_great_guy!";

echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
?>
```

以上代码的结果是这样的：

```
Peter is a great guy!
```

例子解释：

上面的例子把所有 "_" 转换成空格：

1. 创建一个把 "_" 替换为空格的函数
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 MySQL 简介

MySQL 是最流行的开源数据库服务器。

什么是 MySQL？

MySQL 是一种数据库。数据库定义了存储信息的结构。

在数据库中，存在着一些表。类似 HTML 表格，数据库表含有行、列以及单元。

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这些表："Employees", "Products", "Customers" 以及 "Orders"。

数据库表

数据库通常包含一个或多个表。每个表都一个名称（比如 "Customers" 或 "Orders"）。每个表包含带有数据的记录（行）。

下面是一个名为 "Persons" 的表的例子：

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

上面的表含有三个记录（每个记录是一个人）和四个列（LastName, FirstName, Address 以及 City）。

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：

```
SELECT LastName FROM Persons
```

上面的查询选取了 Persons 表中 LastName 列的所有数据，并返回类似这样的记录集：

LastName
Hansen
Svendson
Pettersen

下载 MySQL 数据库

如果您的 PHP 服务器没有 MySQL 数据库，可以在此下载 MySQL：http://www.mysql.com/downloads/index.html

Facts About MySQL Database

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，MySQL 是事实上的标准数据库（比如 Friendster, Yahoo, Google）。这个地址提供了使用 MySQL 的公司的概览：http://www.mysql.com/customers/。

PHP MySQL 连接数据库

免费的 MySQL 数据库通常是通过 PHP 来使用的。

连接到一个 MySQL 数据库

在您能够访问并处理数据库中的数据之前，您必须创建到达数据库的连接。

在 PHP 中，这个任务通过 mysql_connect() 函数完成。

语法

```
mysql_connect(servername,username,password);
```

参数	描述
servername	可选。规定要连接的服务器。默认是 "localhost:3306"。
username	可选。规定登录所使用的用户名。默认值是拥有服务器进程的用户的名称。
password	可选。规定登录所用的密码。默认是 ""。

注释：虽然还存在其他的参数，但上面列出了最重要的参数。请访问 W3School 提供的 PHP MySQL 参考手册，获得更多的细节信息。

例子

在下面的例子中，我们在一个变量中 ($con) 存放了在脚本中供稍后使用的连接。如果连接失败，将执行 "die" 部分：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code

?>
```

关闭连接

脚本一结束，就会关闭连接。如需提前关闭连接，请使用 mysql_close() 函数。

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code

mysql_close($con);
?>
```

PHP MySQL 创建数据库和表

数据库存有一个或多个表。

创建数据库

CREATE DATABASE 语句用于在 MySQL 中创建数据库。

语法

```
CREATE DATABASE database_name
```

为了让 PHP 执行上面的语句，我们必须使用 mysql_query() 函数。此函数用于向 MySQL 连接发送查询或命令。

例子

在下面的例子中，我们创建了一个名为 "my_db" 的数据库：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

mysql_close($con);
?>
```

创建表

CREATE TABLE 用于在 MySQL 中创建数据库表。

语法

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
.......
)
```

为了执行此命令，我必须向 mysql_query() 函数添加 CREATE TABLE 语句。

例子

下面的例子展示了如何创建一个名为 "person" 的表，此表有三列。列名是 "FirstName", "LastName" 以及 "Age"：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

// Create table in my_db database
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE person
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($sql,$con);

mysql_close($con);
?>
```

重要事项：在创建表之前，必须首先选择数据库。通过 mysql_select_db() 函数选取数据库。

注释：当您创建 varchar 类型的数据库字段时，必须规定该字段的最大长度，例如：varchar(15)。

MySQL 数据类型

下面的可使用的各种 MySQL 数据类型：

数值类型	描述
• int(size) • smallint(size) • tinyint(size) • mediumint(size) • bigint(size)	仅支持整数。在 size 参数中规定数字的最大值。
• decimal(size,d) • double(size,d) • float(size,d)	支持带有小数的数字。 在 size 参数中规定数字的最大值。在 d 参数中规定小数点右侧的数字的最大值。

文本数据类型	描述
char(size)	支持固定长度的字符串。（可包含字母、数字以及特殊符号）。 在 size 参数中规定固定长度。
varchar(size)	支持可变长度的字符串。（可包含字母、数字以及特殊符号）。 在 size 参数中规定最大长度。
tinytext	支持可变长度的字符串，最大长度是 255 个字符。
• text • blob	支持可变长度的字符串，最大长度是 65535 个字符。
• mediumtext • mediumblob	支持可变长度的字符串，最大长度是 16777215 个字符。
• longtext • longblob	支持可变长度的字符串，最大长度是 4294967295 个字符。

日期数据类型	描述
• date(yyyy-mm-dd) • datetime(yyyy-mm-dd hh:mm:ss) • timestamp(yyyymmddhhmmss) • time(hh:mm:ss)	支持日期或时间

杂项数据类型	描述
enum(value1,value2, ect)	ENUM 是 ENUMERATED 列表的缩写。可以在括号中存放最多 65535 个值。
set	SET 与 ENUM 相似。但是，SET 可拥有最多 64 个列表项目，并可存放不止一个 choice

主键和自动递增字段

每个表都应有一个主键字段。

主键用于对表中的行进行唯一标识。每个主键值在表中必须是唯一的。此外，主键字段不能为空，这是由于数据库引擎需要一个值来对记录进行定位。

主键字段永远要被编入索引。这条规则没有例外。你必须对主键字段进行索引，这样数据库引擎才能快速定位给予该键值的行。

下面的例子把 personID 字段设置为主键字段。主键字段通常是 ID 号，且通常使用 AUTO_INCREMENT 设置。AUTO_INCREMENT 会在新纪录被添加时逐一增加该字段的值。要确保主键字段不为空，我们必须向该字段添加 NOT NULL 设置。

例子

```
$sql = "CREATE TABLE person
```

```
(
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";

mysql_query($sql,$con);
```

PHP MySQL Insert Into

INSERT INTO 语句用于向数据库表中插入新纪录。

向数据库表插入数据

INSERT INTO 语句用于向数据库表添加新纪录。

语法

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

您还可以规定希望在其中插入数据的列：

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

注释：SQL 语句对大小写不敏感。INSERT INTO 与 insert into 相同。

为了让 PHP 执行该语句，我们必须使用 mysql_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

例子

在前面的章节，我们创建了一个名为 "Person" 的表，有三个列："Firstname", "Lastname" 以及 "Age"。我们将在本例中使用同样的表。下面的例子向 "Person" 表添加了两个新纪录：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("INSERT INTO person (FirstName, LastName, Age)
VALUES ('Peter', 'Griffin', '35')");

mysql_query("INSERT INTO person (FirstName, LastName, Age)
VALUES ('Glenn', 'Quagmire', '33')");

mysql_close($con);
?>
```

把来自表单的数据插入数据库

现在，我们创建一个 HTML 表单，这个表单可把新纪录插入 "Person" 表。

这是这个 HTML 表单：

```
<html>
<body>

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

```
Firstname: <input type="text" name="firstname" />
Lastname: <input type="text" name="lastname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

当用户点击上例中 HTML 表单中的提交按钮时，表单数据被发送到 "insert.php"。"insert.php" 文件连接数据库，并通过 $_POST 变量从表单取回值。然后，mysql_query() 函数执行 INSERT INTO 语句，一条新的记录会添加到数据库表中。

下面是 "insert.php" 页面的代码：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$sql="INSERT INTO person (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";

if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";

mysql_close($con)
?>
```

PHP MySQL Select

SELECT 语句用于从数据库中选取数据。

从数据库表中选取数据

SELECT 语句用于从数据库中选取数据。

语法

```
SELECT column_name(s) FROM table_name
```

注释：SQL 语句对大小写不敏感。SELECT 与 select 等效。

为了让 PHP 执行上面的语句，我们必须使用 mysql_query() 函数。该函数用于向 MySQL 发送查询或命令。

例子

下面的例子选取存储在 "Person" 表中的所有数据（* 字符选取表中所有数据）：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

mysql_close($con);
?>
```

上面这个例子在 $result 变量中存放由 mysql_query() 函数返回的数据。接下来，我们使用 mysql_fetch_array() 函数以数组的形式从记录集返回第一行。每个随后对 mysql_fetch_array() 函数的调用都会返回记录集中的下一行。 while loop 语句会循环记录集中的所有记录。为了输出每行的值，我们使用了 PHP 的 $row 变量 ($row['FirstName'] 和 $row['LastName'])。

以上代码的输出：

```
Peter Griffin
Glenn Quagmire
```

在 HTML 表格中显示结果

下面的例子选取的数据与上面的例子相同，但是将把数据显示在一个 HTML 表格中：

```
<?php
```

```
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person");

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "</tr>";
  }
echo "</table>";

mysql_close($con);
?>
```

以上代码的输出：

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

PHP MySQL Where 子句

如需选取匹配指定条件的数据，请向 SELECT 语句添加 WHERE 子句。

WHERE 子句

如需选取匹配指定条件的数据，请向 SELECT 语句添加 WHERE 子句。

语法

```
SELECT column FROM table
WHERE column operator value
```

下面的运算符可与 WHERE 子句一起使用：

运算符	说明
=	等于
!=	不等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
BETWEEN	介于一个包含范围内
LIKE	搜索匹配的模式

注释：SQL 语句对大小写不敏感。WHERE 与 where 等效。

为了让 PHP 执行上面的语句，我们必须使用 mysql_query() 函数。该函数用于向 SQL 连接发送查询和命令。

例子

下面的例子将从 "Person" 表中选取所有 FirstName='Peter' 的行：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person
WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

?>
```

以上代码的输出：

```
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。

语法

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name
```

注释：SQL 对大小写不敏感。ORDER BY 与 order by 等效。

例子

下面的例子选取 "Person" 表中的存储的所有数据，并根据 "Age" 列对结果进行排序：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM person ORDER BY age");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br />";
  }

mysql_close($con);
?>
```

以上代码的输出：

```
Glenn Quagmire 33
Peter Griffin 35
```

升序或降序的排序

如果您使用 ORDER BY 关键词，记录集的排序顺序默认是升序（1 在 9 之前，"a" 在 "p" 之前）。

请使用 DESC 关键词来设定降序排序（9 在 1 之前，"p" 在 "a" 之前）：

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name DESC
```

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，只有第一列相同时才使用第二列：

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name1, column_name2
```

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 语句用于在数据库表中修改数据。

语法

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

注释：SQL 对大小写不敏感。UPDATE 与 update 等效。

为了让 PHP 执行上面的语句，我们必须使用 mysql_query(函数。该函数用于向 SQL 连接发送查询和命令。

例子

稍早时，我们在本教程中创建了一个名为 "Person" 的表。它看起来类似这样：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的例子更新 "Person" 表的一些数据：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("UPDATE Person SET Age = '36'
WHERE FirstName = 'Peter' AND LastName = 'Griffin'");

mysql_close($con);
?>
```

在这次更新后，"Person" 表格是这样的：

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

PHP MySQL Delete From

DELETE FROM 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 column_name = some_value
```

注释：SQL 对大小写不敏感。DELETE FROM 与 delete from 等效。

为了让 PHP 执行上面的语句，我们必须使用 mysql_query(函数。该函数用于向 SQL 连接发送查询和命令。

例子

稍早时，我们在本教程中创建了一个名为 "Person" 的表。它看起来类似这样：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的例子删除 "Person" 表中所有 LastName='Griffin' 的记录：

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("DELETE FROM Person WHERE LastName='Griffin'");

mysql_close($con);
?>
```

在这次删除之后，表是这样的：

FirstName	LastName	Age
Peter	Griffin	35

PHP Database ODBC

ODBC 指的是（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 数据的 ODBC 连接的方法：

1. 在控制面板中打开*管理工具*
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*”按钮
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. 为这个数据库取一个*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 信息服务器 (IIS)，上面的指令会生效，但是假如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参数。

odbc_exec() 函数用于执行 SQL 语句。

例子

下面的例子创建了到达名为 northwind 的 DSN 的连接，不没有用户名和密码。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);
```

The code line below returns the value of a field called "CompanyName":

```
$compname=odbc_result($rs,"CompanyName");
```

关闭 ODBC 连接

odbc_close()函数用于关闭 ODBC 连接。

```
odbc_close($conn);
```

ODBC 实例

下面的例子展示了如何首先创建一个数据库连接，然后是结果集，然后在 HTML 表格中显示数据。

```
<html>
<body>

<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
  {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs)
  {exit("Error in SQL");}
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 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。

什么是 XML？

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。

如果希望学习更多有关 XML 的内容，请访问我们的 XML 教程。

什么是 Expat？

如需读取和更新 - 创建创建并处理 - 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 API 来访问树种的元素，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，而不是它们的结果。正因如此，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>John</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：John
- 关闭元素： from

上面的 XML 范例包含了形式良好的 XML。不过这个例子是无效的 XML，因为没有与它关联的文档类型声明 (DTD)，也没有内嵌的 DTD。

不过，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

将在我们的例子中使用下面的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
```

```
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</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。

例子

```
<?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);

?>
```

以上代码的输出：

```
-- Note --
To: George
From: John
Heading: Reminder
Message: Don't forget the meeting!
```

工作原理解释：

- 通过 xml_parser_create() 函数初始化 XML 解析器
- 创建配合不同事件处理程序的的函数
- 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
- 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
- 通过 xml_parse() 函数来解析文件 "test.xml"
- 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
- 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

如需更多有关 PHP Expat 函数的信息，请访问我们的 PHP XML Parser 参考手册。

PHP XML DOM

内建的 DOM 解析器使在 PHP 中处理 XML 文档成为可能。

什么是 DOM？

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

如果您希望学习更多有关 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 - 创建创建并处理 - 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 API 来访问树种的元素，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>John</from>
```

XML DOM 把 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: "John"

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

将在我们的例子中使用下面的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</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：

例子

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

以上代码的输出：

```
George John Reminder Don't forget the meeting!
```

假如您在浏览器窗口中查看源代码，会看到下面这些 HTML：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

上面的例子创建了一个 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。

循环 XML

我们要初始化 XML 解析器，加载 XML，并循环 <note> 元素的所有元素：

例子

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item)
  {
  print $item->nodeName . " = " . $item->nodeValue . "<br />";
  }
?>
```

以上代码的输出：

```
#text =
to = George
#text =
from = John
#text =
heading = Reminder
#text =
body = Don't forget the meeting!
#text =
```

在上面的例子中，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。

如果您希望学习更多有关 XML DOM 的知识，请访问我们的 XML DOM 教程。

PHP SimpleXML

SimpleXML 处理最普通的 XML 任务，其余的任务则交由其它扩展。

什么是 SimpleXML？

SimpleXML 是 PHP 5 中的新特性。在了解 XML 文档 layout 的情况下，它是一种取得元素属性和文本的便利途径。

与 DOM 或 Expat 解析器相比，SimpleXML 仅仅用几行代码就可以从元素中读取文本数据。

SimpleXML 可把 XML 文档转换为对象，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，其中的下标对应属性名称。
- 元素数据 - 来自元素的文本数据被转换为字符串。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 XML 文件
- 从 XML 字符串中提取数据
- 编辑文本节点或属性

不过，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 5.0 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。

使用 SimpleXML

下面是 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

我们打算从上面的 XML 文件输出元素的名称和数据。

这是需要做的事情：

1. 加载 XML 文件
2. 取得第一个元素的名称
3. 使用 children() 函数创建在每个子节点上触发的循环
4. 输出每个子节点的元素名称和数据

例子

```
<?php
$xml = simplexml_load_file("test.xml");

echo $xml->getName() . "<br />";

foreach($xml->children() as $child)
```

```
  {
  echo $child->getName() . ": " . $child . "<br />";
  }
?>
```

以上代码的输出：

```
note
to: George
from: John
heading: Reminder
body: Don't forget the meeting!
```

更多有关 PHP SimpleXML 的信息

如需更多有关 PHP SimpleXML 的信息，请访问我们的 PHP SimpleXML 参考手册。

AJAX 简介

AJAX = Asynchronous JavaScript And XML（异步 JavaScript 及 XML）

AJAX 是 *A*synchronous *J*avaScript *A*nd *X*ML 的首字母缩写。

AJAX 并不是一种新的编程语言，而仅仅是一种新的技术，它可以创建更好、更快且交互性更强的 web 应用程序。

AJAX 使用 JavaScript 在 web 浏览器与 web 服务器之间来发送和接收数据。

通过在幕后与 web 服务器交换数据，而不是每当用户作出改变时重载整个 web 页面，AJAX 技术可以使网页更迅速地响应。

AJAX 基于开放的标准

AJAX 基于以下开放的标准：

- *JavaScript*
- *XML*
- *HTML*
- *CSS*

在 AJAX 中使用的开放标准被良好地定义，并得到所有主要浏览器的支持。AJAX 应用程序独立于浏览器和平台。（可以说，它是一种跨平台跨浏览器的技术）。

AJAX 事关更好的 Internet 应用程序

与桌面应用程序相比，Web 应用程序有很多优势：

- 可拥有更多用户
- 更容易安装和维护
- 更容易开发

但是， 应用程序不总是象传统应用程序那样强大和友好。

通过 AJAX，可以使 Internet 应用程序更加强大（更轻巧、更快速，且更易使用）。

今天您就可以开始使用 AJAX

没有什么新知识需要学习。

AJAX 基于开放的标准。而这些标准已被大多数开发者使用多年。

大多数 web 应用程序可通过使用 AJAX 技术进行重写，来替代传统的 HTML 表单。

AJAX 使用 XML 和 HTTP 请求

传统的 web 应用程序会把数据提交到 web 服务器（使用 HTML 表单）。在 web 服务器把数据处理完毕之后，会向用户返回一张完整的新网页。

由于每当用户提交输入，服务器就会返回新网页，传统的 web 应用程序往往运行缓慢，且越来越不友好。

通过 AJAX，web 应用程序无需重载网页，就可以发送并取回数据。完成这项工作，需要通过向服务器发送 HTTP 请求（在幕后），并通过当服务器返回数据时使用 JavaScript 仅仅修改网页的某部分。

一般使用 XML 作为接收服务器数据的格式，尽管可以使用任何格式，包括纯文本。

您将在本教程接下来的章节学习到如何完成这些工作。

PHP 和 AJAX

不存在什么 AJAX 服务器。

AJAX 是一种在浏览器运行的技术。它使用浏览器与 web 服务器之间的异步数据传输，使网页从服务器请求少量的信息，而不是整张页面。

AJAX 是一种独立于 web 服务器软件的 web 浏览器技术。

但是，在本教程中，我们将集中在运行在 PHP 服务器上的实际案例，而不是 AJAX 的*工作原理*。

如需阅读更多有关 AJAX 如何工作的知识，请访问我们的 AJAX 教程。

AJAX XMLHttpRequest

XMLHttpRequest 对象使 AJAX 成为可能。

XMLHttpRequest

XMLHttpRequest 对象是 AJAX 的关键。

该对象在 Internet Explorer 5.5 与 2000 年 7 月发布之后就已经可用了，但是在 2005 人们开始讨论 AJAX 和 Web 2.0 之前，这个对象并没有得到充分的认识。

创建 XMLHttpRequest 对象

不同的浏览器使用不同的方法来创建 *XMLHttpRequest* 对象。

Internet Explorer 使用 *ActiveXObject*。

其他浏览器使用名为 *XMLHttpRequest* 的 JavaScript 内建对象。

要克服这个问题，可以使用这段简单的代码：

```
var XMLHttp=null
if (window.XMLHttpRequest)
  {
  XMLHttp=new XMLHttpRequest()
  }
else if (window.ActiveXObject)
  {
  XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
```

代码解释：

1. 首先创建一个作为 XMLHttpRequest 对象使用的 *XMLHttp* 变量。把它的值设置为 null。
2. 然后测试 *window.XMLHttpRequest* 对象是否可用。在新版本的 Firefox, Mozilla, Opera 以及 Safari 浏览器中，该对象是可用的。
3. 如果可用，则用它创建一个新对象：*XMLHttp=new XMLHttpRequest()*
4. 如果不可用，则检测 *window.ActiveXObject* 是否可用。在 Internet Explorer version 5.5 及更高的版本中，该对象是可用的。
5. 如果可用，使用它来创建一个新对象：*XMLHttp=new ActiveXObject()*

改进的例子

一些程序员喜欢使用最新最快的版本的 XMLHttpRequest 对象。

下面的例子试图加载微软最新版本的 "Msxml2.XMLHTTP"，在 Internet Explorer 6 中可用，如果无法加载，则后退到 "Microsoft.XMLHTTP"，在 Internet Explorer 5.5 及其后版本中可用。

```
function GetXmlHttpObject()
{
var xmlHttp=null;
```

```
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

代码解释：

1. 首先创建用作 XMLHttpRequest 对象的 *XMLHttp* 变量。把它的值设置为 null。
2. 按照 web 标准创建对象 (Mozilla, Opera 以及 Safari)：*XMLHttp=new XMLHttpRequest()*
3. 按照微软的方式创建对象，在 Internet Explorer 6 及更高的版本可用：*XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")*
4. 如果捕获错误，则尝试更老的方法 (Internet Explorer 5.5)：*XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")*

更多有关 XMLHttpRequest 对象的信息

如果您希望阅读更多有关 XMLHttpRequest 的内容，请访问我们的 AJAX 教程。

PHP 和 AJAX 请求

AJAX 请求

在下面的 AJAX 例子中，我们将演示当用户向 web 表单中输入数据时，网页如何与在线的 web 服务器进行通信。

在下面的文本框中输入一个名字（测试说明：该实例功能未实现）

First Name:

Suggestions:

这个例子包括三张页面：

- a simple HTML form
- a JavaScript
- a PHP page

HTML 表单

这是 HTML 表单。它包含一个简单的 HTML 表单和指向 JavaScript 的链接：

```
<html>
<head>
<script src="clienthint.js"></script>
</head>

<body>

<form>
First Name:
<input type="text" id="txt1"
onkeyup="showHint(this.value)">
</form>

<p>Suggestions: <span id="txtHint"></span></p>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面含有一个简单的 HTML 表单，其中带有一个名为 "txt1" 的输入字段。

该表单是这样工作的：

1. 当用户在输入域中按下并松开按键时，会触发一个事件
2. 当该事件被触发时，执行名为 showHint() 的函数
3. 表单的下面是一个名为 "txtHint" 的 。它用作 showHint() 函数所返回数据的占

位符。

JavaScript

JavaScript 代码存储在 "clienthint.js" 文件中，它被链接到 HTML 文档：

```
var xmlHttp

function showHint(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML=""
  return
  }
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
var url="gethint.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

showHint() 函数

每当在输入域中输入一个字符，该函数就会被执行一次。

如果文本框中有内容 (str.length > 0)，该函数这样执行：

1. 定义要发送到服务器的 URL（文件名）
2. 把带有输入域内容的参数 (q) 添加到这个 URL
3. 添加一个随机数，以防服务器使用缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并在事件被触发时告知该对象执行名为 stateChanged 的函数
5. 用给定的 URL 来打开打开这个 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

如果输入域为空，则函数简单地清空 txtHint 占位符的内容。

stateChanged() 函数

每当 XMLHTTP 对象的状态发生改变，则执行该函数。

在状态变成 4（或 "complete"）时，用响应文本填充 txtHint 占位符 txtHint 的内容。

GetXmlHttpObject() 函数

AJAX 应用程序只能运行在完整支持 XML 的 web 浏览器中。

上面的代码调用了名为 GetXmlHttpObject() 的函数。

该函数的作用是解决为不同浏览器创建不同 XMLHTTP 对象的问题。

这一点在上一节中已经解释过了。

PHP 页面

被 JavaScript 代码调用的服务器页面是一个名为 "gethint.php" 的简单服务器页面。

"gethint.php" 中的代码会检查名字数组，然后向客户端返回对应的名字：

```
<?php
// Fill up array with names
$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";

//get the q parameter from URL
$q=$_GET["q"];

//lookup all hints from array if length of 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];
      }
    }
  }
}

//Set output to "no suggestion" if no hint were found
//or to the correct values
if ($hint == "")
{
$response="no suggestion";
}
else
{
$response=$hint;
}

//output the response
echo $response;
?>
```

如果存在从 JavaScript 送来的文本 (strlen($q) > 0)，则：

1. 找到与 JavaScript 所传送的字符相匹配的名字
2. 如果找到多个名字，把所有名字包含在 response 字符串中
3. 如果没有找到匹配的名字，把 response 设置为 "no suggestion"
4. 如果找到一个或多个名字，把 response 设置为这些名字
5. 把 response 发送到 "txtHint" 占位符

PHP 和 AJAX XML 实例

AJAX 可与 XML 文件进行交互式通信。

AJAX XML 实例

在下面的 AJAX 实例中，我们将演示网页如何使用 AJAX 技术从 XML 文件中读取信息。

在列表中选择一张 CD （测试说明：该实例功能未实现）

Select a CD:

在此列出 CD 信息。

本例包括三张页面：

- 一个简单 HTML 表单
- 一个 XML 文件
- 一个 JavaScript 文件
- 一张 PHP 页面

HTML 表单

上面的例子包含了一张简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="selectcd.js"></script>
</head>

<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bee Gees">Bee Gees</option>
<option value="Cat Stevens">Cat Stevens</option>
</select>
</form>

<p>
<div id="txtHint"><b>CD info will be listed here.</b></div>
</p>

</body>
</html>
```

例子解释：

正如您看到的，它仅仅是一张简单的 HTML 表单，其中带有名为 "cds" 的下拉列表。

表单下面的段落包含了一个名为 "txtHint" 的 div。这个 div 用作从 web 服务器检索到的数据的占位符。

当用户选择数据时，会执行名为 "showCD" 的函数。这个函数的执行是由 "onchange" 事件触发的。

换句话说，每当用户改变了下拉列表中的值，就会调用 showCD 函数。

XML 文件

XML 文件是 "cd_catalog.xml"。该文件中包含了有关 CD 收藏的数据。

JavaScript

这是存储在 "selectcd.js" 文件中的 JavaScript 代码：

```
var xmlHttp

function showCD(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getcd.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;

try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与上一节中的相同，您可以参阅上一页中的相馆解释。

showCD() 函数

假如选择了下拉列表中的某个项目，则函数执行：

1. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象
2. 定义发送到服务器的 URL（文件名）
3. 向 URL 添加带有下拉列表内容的参数 (q)
4. 添加一个随机数，以防服务器使用缓存的文件
5. 当触发事件时调用 stateChanged
6. 通过给定的 URL 打开 XMLHTTP 对象
7. 向服务器发送 HTTP 请求

PHP 页面

这个被 JavaScript 调用的服务器页面，是一个名为 "getcd.php" 的简单 PHP 文件。

这张页面是用 PHP 编写的，使用 XML DOM 来加载 XML 文档 "cd_catalog.xml"。

代码运行针对 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; $i++)
{
//Process only element nodes
if ($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++)
{
//Process only element nodes
if ($cd->item($i)->nodeType==1)
  {
  echo($cd->item($i)->nodeName);
  echo(": ");
  echo($cd->item($i)->childNodes->item(0)->nodeValue);
  echo("<br />");
  }
}
?>
```

例子解释

当请求从 JavaScript 发送到 PHP 页面时，发生：

1. PHP 创建 "cd_catalog.xml" 文件的 XML DOM 对象
2. 循环所有 "artist" 元素 (nodetypes = 1)，查找与 JavaScript 所传数据向匹配的名字
3. 找到 CD 包含的正确 artist
4. 输出 album 的信息，并发送到 "txtHint" 占位符

PHP 和 AJAX MySQL 数据库实例

AJAX 可用来与数据库进行交互式通信。

AJAX 数据库实例

在下面的 AJAX 实例中，我们将演示网页如何使用 AJAX 技术从 MySQL 数据库中读取信息。

在下拉列表中选择一个名字 （测试说明：该实例功能未实现）

Select a User:

在此列出用户信息。
此列由四个元素组成：

- MySQL 数据库
- 简单的 HTML 表单
- JavaScript
- PHP 页面

数据库

将在本例中使用的数据库看起来类似这样：

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

HTML 表单

上面的例子包含了一个简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="selectuser.js"></script>
</head>
<body>

<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>

<p>
<div id="txtHint"><b>User info will be listed here.</b></div>
</p>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，它仅仅是一个简单的 HTML 表单，其中带有名为 "users" 的下拉列表，这个列表包含了姓名，以及与数据库的 "id" 对应的选项值。

表单下面的段落包含了名为 "txtHint" 的 div。这个 div 用作从 web 服务器检索到的信息的占位符。

当用户选择数据时，执行名为 "showUser()" 的函数。该函数的执行由 "onchange" 事件触发。

换句话说：每当用户改变下拉列表中的值，就会调用 showUser() 函数。

JavaScript

这是存储在 "selectuser.js" 文件中的 JavaScript 代码：

```
var xmlHttp

function showUser(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getuser.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 //Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
```

```
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP AJAX 请求 那一节中的相同，您可以参阅其中的相关解释。

showUser() 函数

假如下拉列表中的项目被选择，函数执行：

1. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象
2. 定义发送到服务器的 URL（文件名）
3. 向 URL 添加带有下拉列表内容的参数 (q)
4. 添加一个随机数，以防服务器使用缓存的文件
5. 当触发事件时调用 stateChanged
6. 通过给定的 URL 打开 XMLHTTP 对象
7. 向服务器发送 HTTP 请求

PHP 页面

由 JavaScript 调用的服务器页面，是名为 "getuser.php" 的简单 PHP 文件。

该页面用 PHP 编写，并使用 MySQL 数据库。

其中的代码执行针对数据库的 SQL 查询，并以 HTML 表格返回结果：

```
<?php
$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysql_fetch_array($result))
 {
 echo "<tr>";
 echo "<td>" . $row['FirstName'] . "</td>";
 echo "<td>" . $row['LastName'] . "</td>";
 echo "<td>" . $row['Age'] . "</td>";
```

```
 echo "<td>" . $row['Hometown'] . "</td>";
 echo "<td>" . $row['Job'] . "</td>";
 echo "</tr>";
 }
echo "</table>";

mysql_close($con);
?>
```

例子解释：

当查询从 JavaScript 被发送到这个 PHP 页面，会发生：

1. PHP 打开到达 MySQL 服务器的连接
2. 找到拥有指定姓名的 "user"
3. 创建表格，插入数据，然后将其发送到 "txtHint" 占位符

PHP 和 AJAX responseXML 实例

AJAX 可用于以 XML 返回数据库信息。

AJAX Database 转 XML 实例 （测试说明：该实例功能未实现）

在下面的 AJAX 实例中，我们将演示网页如何从 MySQL 数据库中读取信息，把数据转换为 XML 文档，并在不同的地方使用这个文档来显示信息。

本例与上一节中的 "PHP AJAX Database" 这个例子很相似，不过有一个很大的不同：在本例

中，我们通过使用 responseXML 函数从 PHP 页面得到的是 XML 形式的数据。

把 XML 文档作为响应来接收，使我们有能力更新页面的多个位置，而不仅仅是接收一个 PHP 输出并显示出来。

在本例中，我们将使用从数据库接收到的信息来更新多个 元素。

在下拉列表中选择一个名字

Select a User:

此列由四个元素组成：

- MySQL 数据库
- 简单的 HTML 表单
- JavaScript
- PHP 页面

数据库

将在本例中使用的数据库看起来类似这样：

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

HTML 表单

上面的例子包含了一个简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="responsexml.js"></script>
</head>
<body>

<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>

<h2>
<span id="firstname"></span> <span id="lastname"></span>
</h2>
```

```
<span id="job"></span>

<div style="text-align: right">
<span id="age_text"></span>
<span id="age"></span>
<span id="hometown_text"></span>
<span id="hometown"></span>
</div>

</body>
</html>
```

例子解释 - HTML 表单

- HTML 表单是一个下拉列表，其 name 属性的值是 "users"，可选项的值与数据库的 id 字段相对应
- 表单下面有几个 元素，它们用作我们所接收到的不同的值的占位符
- 当用户选择了具体的选项，函数 "showUser()" 就会执行。该函数的执行由 "onchange" 事件触发

换句话说，每当用户在下拉列表中改变了值，函数 showUser() 就会执行，并在指定的 元素中输出结果。

JavaScript

这是存储在文件 "responsexml.js" 中的 JavaScript 代码：

```
var xmlHttp

function showUser(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="responsexml.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
 xmlDoc=xmlHttp.responseXML;
 document.getElementById("firstname").innerHTML=
 xmlDoc.getElementsByTagName("firstname")[0].childNodes[0].nodeValue;
 document.getElementById("lastname").innerHTML=
 xmlDoc.getElementsByTagName("lastname")[0].childNodes[0].nodeValue;
 document.getElementById("job").innerHTML=
 xmlDoc.getElementsByTagName("job")[0].childNodes[0].nodeValue;
 document.getElementById("age_text").innerHTML="Age: ";
 document.getElementById("age").innerHTML=
 xmlDoc.getElementsByTagName("age")[0].childNodes[0].nodeValue;
```

```
 document.getElementById("hometown_text").innerHTML="<br/>From: ";
 document.getElementById("hometown").innerHTML=
 xmlDoc.getElementsByTagName("hometown")[0].childNodes[0].nodeValue;
 }
}

function GetXmlHttpObject()
 {
 var objXMLHttp=null
 if (window.XMLHttpRequest)
  {
  objXMLHttp=new XMLHttpRequest()
  }
 else if (window.ActiveXObject)
  {
  objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
 return objXMLHttp
 }
```

例子解释：

showUser() 与 GetXmlHttpObject 函数与 PHP 和 AJAX MySQL 数据库实例 这一节中的例子是相同的。您可以参阅其中的相关解释。

stateChanged() 函数

如果选择了下拉列表中的项目，该函数执行：

1. 通过使用 responseXML 函数，把 "xmlDoc" 变量定义为一个 XML 文档
2. 从这个 XML 文档中取回数据，把它们放在正确的 "span" 元素中

PHP 页面

这个由 JavaScript 调用的服务器页面，是一个名为 "responsexml.php" 的简单的 PHP 文件。

该页面由 PHP 编写，并使用 MySQL 数据库。

代码会运行一段针对数据库的 SQL 查询，并以 XML 文档返回结果：

```
<?php
header('Content-Type: text/xml');
header("Cache-Control: no-cache, must-revalidate");
//A date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");

$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = ".$q."";

$result = mysql_query($sql);
```

```
echo '<?xml version="1.0" encoding="ISO-8859-1"?>
<person>';
while($row = mysql_fetch_array($result))
 {
 echo "<firstname>" . $row['FirstName'] . "</firstname>";
 echo "<lastname>" . $row['LastName'] . "</lastname>";
 echo "<age>" . $row['Age'] . "</age>";
 echo "<hometown>" . $row['Hometown'] . "</hometown>";
 echo "<job>" . $row['Job'] . "</job>";
 }
echo "</person>";

mysql_close($con);
?>
```

例子解释：

当查询从 JavaScript 送达 PHP 页面时，会发生：

- PHP 文档的 content-type 被设置为 "text/xml"
- PHP 文档被设置为 "no-cache"，以防止缓存
- 用 HTML 页面送来的数据设置 $q 变量
- PHP 打开与 MySQL 服务器的连接
- 找到带有指定 id 的 "user"
- 以 XML 文档输出数据

PHP 和 AJAX Live Search

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 AJAX 例子中，我们将演示一个实时的搜索。

实时的搜索与传统搜索相比，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

在下面的文本框中搜索 W3School 的页面

本例包括四个元素：

- 简单的 HTML 表单
- JavaScript
- PHP 页面
- XML 文档

在本例中，结果在一个 XML 文档 (links.xml) 中进行查找。为了让这个例子小而简单，我们只提供 8 个结果。

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单，针对此表单的 CSS 样式，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="livesearch.js"></script>
<style type="text/css">
#livesearch
  {
  margin:0px;
  width:194px;
  }
#txt1
  {
  margin:0px;
  }
</style>
</head>
<body>

<form>
<input type="text" id="txt1" size="30"
onkeyup="showResult(this.value)">

<div id="livesearch"></div>
</form>

</body>
</html>
```

例子解释 - HTML 表单

正如你看到的，HTML 页面包含一个简单的 HTML 表单，其中的文本框名为 "txt1"。

表单是这样工作的：

1. 当用户在文本框中按键并松开按键时，会触发一个事件
2. 当事件触发时，会执行名为 showResult() 的函数
3. 表单下面是名为 "livesearch" 的 元素。它用作 showResult() 所返回数据的占位符

JavaScript

JavaScript 代码存储在与 HTML 文档连接的 "livesearch.js" 中：

```
var xmlHttp

function showResult(str)
{
if (str.length==0)
 {
 document.getElementById("livesearch").
 innerHTML="";
 document.getElementById("livesearch").
 style.border="0px";
 return
 }

xmlHttp=GetXmlHttpObject()

if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }

var url="livesearch.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("livesearch").
 innerHTML=xmlHttp.responseText;
 document.getElementById("livesearch").
 style.border="1px solid #A5ACB2";
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

GetXmlHttpObject 与 PHP 和 AJAX 请求 中的例子相同。

showResult() 函数

该函数每当一个字符输入文本框就会执行一次。

如果文本域中没有输入 (str.length == 0)，该函数把返回字段设置为空，并删除周围的任何边框。

不过，如果文本域中存在输入，则函数执行：

1. 定义发送到服务器的 url （文件名）
2. 把带有输入框内容的参数 (q) 添加到 url
3. 添加一个随机数，以防止服务器使用缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并在触发一个变化时告知此函数执行名为 stateChanged 的一个函数
5. 使用给定的 url 来打开 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

stateChanged() 函数

每当 XMLHTTP 对象的状态发生变化时，该函数就会执行。

当状态变为 4 (或 "complete") 时，就会使用响应文本来填充 txtHint 占位符的内容，并在返回字段周围设置一个边框。

PHP 页面

由 JavaScript 代码调用的服务器页面是名为 "livesearch.php" 的 PHP 文件。

"livesearch.php" 中的代码检查那个 XML 文档 "links.xml"。该文档 w3school.com.cn 上的一些页面的标题和 URL。

这些代码会搜索 XML 文件中匹配搜索字符串的标题，并以 HTML 返回结果：

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');

//get the q parameter from URL
$q=$_GET["q"];

//lookup all links from the xml file if length of q>0
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)
  {
  //find a link matching the search text
```

```
  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>";
    }
   }
  }
 }
}

// Set output to "no suggestion" if no hint were found
// or to the correct values
if ($hint == "")
 {
 $response="no suggestion";
 }
else
 {
 $response=$hint;
 }

//output the response
echo $response;
?>
```

例子解释：

如果从 JavaScript 送来了任何文本 (strlen($q) > 0)，会发生：

1. PHP 创建 "links.xml" 文件的一个 XML DOM 对象
2. 遍历所有 "title" 元素 (nodetypes = 1)，以便找到匹配 JavaScript 所传数据的 name
3. 找到包含正确 title 的 link，并设置为 "$response" 变量。如果找到多于一个匹配，所有的匹配都会添加到变量
4. 如果没有找到匹配，则把 $response 变量设置为 "no suggestion"
5. $result 是送往 "livesearch" 占位符的输出

PHP 和 AJAX RSS 阅读器

RSS 阅读器用于阅读 RSS Feed。

RSS 允许对新闻和更新进行快速浏览。

AJAX RSS 阅读器

在下面的 AJAX 实例中，我们将演示一个 RSS 阅读器，通过它，来自 RSS 的内容在不进行刷新的情况下载入网页。

在下面的列表框中选择一个 RSS 新闻订阅

Select an RSS-Feed:

在此列出 RSS Feed。

本例包括三个元素：

- 简单的 HTML 表单
- JavaScript
- PHP 页面

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单和执行一个 JavaScript 文件的链接：

```
<html>
```

```
<head>
<script type="text/javascript" src="getrss.js"></script>
</head>
<body>

<form>
Select an RSS-Feed:
<select onchange="showRSS(this.value)">
<option value="Google">Google News</option>
<option value="MSNBC">MSNBC News</option>
</select>
</form>

<p><div id="rssOutput">
<b>RSS Feed will be listed here.</b></div></p>
</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面包含一个简单的 HTML 表单，其中带有一个下拉列表框。

表单是这样工作的：

1. 当用户选择下拉框中的选项时，会触发一个事件
2. 当事件触发时，执行 showRSS() 函数

表单下面是名为 "rssOutput" 的一个 。它用作 showRSS() 函数所返回的数据的占位符。

JavaScript

JavaScript 代码存储在 "getrss.js" 中，它与 HTML 文档相连接：

```
var xmlHttp

function showRSS(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="getrss.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }

function stateChanged()
 {
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
  document.getElementById("rssOutput")
  .innerHTML=xmlHttp.responseText
  }
 }

function GetXmlHttpObject()
```

```
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP 和 AJAX 请求 这一节中的例子相同。

showRSS() 函数

每当在下拉框中选择选择时，该函数就会执行：

1. 定义发送到服务器的 url （文件名）
2. 把参数 (q) 添加到 url，参数内容是下拉框中的被选项
3. 添加一个随机数，以防止服务器缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并告知该对象在触发一个改变时去执行 stateChanged 函数
5. 通过给定的 url 来打开 XMLHTTP
6. 把 HTTP 请求发动到服务器

PHP 页面

调用 JavaScript 代码的服务器页面是名为 "getrss.php" 的 PHP 文件：

```
<?php
//get the q parameter from URL
$q=$_GET["q"];

//find out which feed was selected
if($q=="Google")
 {
 $xml=("http://news.google.com/news?ned=us&topic=h&output=rss");
 }
elseif($q=="MSNBC")
 {
 $xml=("http://rss.msnbc.msn.com/id/3032091/device/rss/rss.xml");
 }

$xmlDoc = new DOMDocument();
$xmlDoc->load($xml);
```

```
//get elements from "<channel>"
$channel=$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;

//output elements from "<channel>"
echo("<p><a href='" . $channel_link
 . "'>" . $channel_title . "</a>");
echo("<br />");
echo($channel_desc . "</p>");

//get and output "<item>" elements
$x=$xmlDoc->getElementsByTagName('item');
for ($i=0; $i<=2; $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>");
 }
?>
```

例子解释：

当一个选项从 JavaScript 发送时，会发生：

1. PHP 找出哪个 RSS feed 被选中
2. 为选中的 RSS feed 创建 XML DOM 对象
3. 找到并输出来自 RSS 频道的元素
4. 遍历前三个 RSS 项目中的元素，并进行输出

PHP 和 AJAX 投票

AJAX 投票

在这个 AJAX 实例中，我们将演示一个投票程序，网页在不重新加载的情况下，就可以获得结果。

到目前为止，您喜欢 PHP 和 AJXA 吗？

Yes: ○

No: ○

本例包括四个元素：

- HTML 表单
- JavaScript
- PHP 页面
- 存放结果的文本文件

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单，以及一个与 JavaScript 文件的连接：

```
<html>
<head>
<script src="poll.js"></script>
</head>
<body>

<div id="poll">
<h2>Do you like PHP and AJAX so far?</h2>

<form>
Yes:
<input type="radio" name="vote"
value="0" onclick="getVote(this.value)">
<br />
No:
```

```
<input type="radio" name="vote"
value="1" onclick="getVote(this.value)">
</form>
</div>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面包含一个简单的 HTML 表单，其中的 元素带有两个单选按钮。

表单这样工作：

- 当用户选择 "yes" 或 "no" 时，会触发一个事件
- 当事件触发时，执行 getVote() 函数
- 围绕该表单的是名为 "poll" 的 。当数据从 getVote() 函数返回时，返回的数据会替代该表单。

文本文件

文本文件 (poll_result.txt) 中存储来自投票程序的数据。

它类似这样：

```
0||0
```

第一个数字表示 "Yes" 投票，第二个数字表示 "No" 投票。

注释：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)。

JavaScript

JavaScript 代码存储在 "poll.js" 中，并于 HTML 文档相连接：

```
var xmlHttp

function getVote(int)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="poll_vote.php"
url=url+"?vote="+int
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
```

```
 {
 document.getElementById("poll").
 innerHTML=xmlHttp.responseText;
 }
}

function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
 {
 objXMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
 objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
return objXMLHttp
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP 和 AJAX 请求 这一节中的例子相同。

getVote() 函数

当用户在 HTML 表单中选择 "yes" 或 "no" 时，该函数就会执行。

1. 定义发送到服务器的 url （文件名）
2. 向 url 添加参数 (vote)，参数中带有输入字段的内容
3. 添加一个随机数，以防止服务器使用缓存的文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并告知该对象当触发一个变化时执行 stateChanged 函数
5. 用给定的 url 来打开 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

PHP 页面

由 JavaScript 代码调用的服务器页面是名为 "poll_vote.php" 的一个简单的 PHP 文件。

```
<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0)
 {
 $yes = $yes + 1;
 }
if ($vote == 1)
 {
```

```
 $no = $no + 1;
 }

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>

<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

例子解释：

所选的值从 JavaScript 传来，然后会发生：

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

PHP Array 函数

PHP Array 简介

array 函数允许您对数组进行操作。

PHP 支持单维和多维的数组。同时提供了用数据库查询结果来构造数组的函数。

安装

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 Array 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
array()	创建数组。	3
array_change_key_case()	返回其键均为大写或小写的数组。	4
array_chunk()	把一个数组分割为新的数组块。	4
array_combine()	通过合并两个数组来创建一个新数组。	5
array_count_values()	用于统计数组中所有值出现的次数。	4
array_diff()	返回两个数组的差集数组。	4
array_diff_assoc()	比较键名和键值，并返回两个数组的差集数组。	4
array_diff_key()	比较键名，并返回两个数组的差集数组。	5
array_diff_uassoc()	通过用户提供的回调函数做索引检查来计算数组的差集。	5
array_diff_ukey()	用回调函数对键名比较计算数组的差集。	5
array_fill()	用给定的值填充数组。	4
array_filter()	用回调函数过滤数组中的元素。	4
array_flip()	交换数组中的键和值。	4
array_intersect()	计算数组的交集。	4
array_intersect_assoc()	比较键名和键值，并返回两个数组的交集数组。	4
array_intersect_key()	使用键名比较计算数组的交集。	5

array_intersect_uassoc()	带索引检查计算数组的交集，用回调函数比较索引。	5
array_intersect_ukey()	用回调函数比较键名来计算数组的交集。	5
array_key_exists()	检查给定的键名或索引是否存在于数组中。	4
array_keys()	返回数组中所有的键名。	4
array_map()	将回调函数作用到给定数组的单元上。	4
array_merge()	把一个或多个数组合并为一个数组。	4
array_merge_recursive()	递归地合并一个或多个数组。	4
array_multisort()	对多个数组或多维数组进行排序。	4
array_pad()	用值将数组填补到指定长度。	4
array_pop()	将数组最后一个单元弹出（出栈）。	4
array_product()	计算数组中所有值的乘积。	5
array_push()	将一个或多个单元（元素）压入数组的末尾（入栈）。	4
array_rand()	从数组中随机选出一个或多个元素，并返回。	4
array_reduce()	用回调函数迭代地将数组简化为单一的值。	4
array_reverse()	将原数组中的元素顺序翻转，创建新的数组并返回。	4
array_search()	在数组中搜索给定的值，如果成功则返回相应的键名。	4
array_shift()	删除数组中的第一个元素，并返回被删除元素的值。	4
array_slice()	在数组中根据条件取出一段值，并返回。	4
array_splice()	把数组中的一部分去掉并用其它值取代。	4
array_sum()	计算数组中所有值的和。	4
array_udiff()	用回调函数比较数据来计算数组的差集。	5
array_udiff_assoc()	带索引检查计算数组的差集，用回调函数比较数据。	5
array_udiff_uassoc()	带索引检查计算数组的差集，用回调函数比较数据和索引。	5
array_uintersect()	计算数组的交集，用回调函数比较数据。	5
array_uintersect_assoc()	带索引检查计算数组的交集，用回调函数比较数据。	5
array_uintersect_uassoc()	带索引检查计算数组的交集，用回调函数比较数据和索引。	5
array_unique()	删除数组中重复的值。	4
array_unshift()	在数组开头插入一个或多个元素。	4
array_values()	返回数组中所有的值。	4
array_walk()	对数组中的每个成员应用用户函数。	3
array_walk_recursive()	对数组中的每个成员递归地应用用户函数。	5
arsort()	对数组进行逆向排序并保持索引关系。	3
asort()	对数组进行排序并保持索引关系。	3
compact()	建立一个数组，包括变量名和它们的值。	4
count()	计算数组中的元素数目或对象中的属性个数。	3
current()	返回数组中的当前元素。	3
each()	返回数组中当前的键／值对并将数组指针向前移动一步。	3
end()	将数组的内部指针指向最后一个元素。	3

extract()	从数组中将变量导入到当前的符号表。	3
in_array()	检查数组中是否存在指定的值。	4
key()	从关联数组中取得键名。	3
krsort()	对数组按照键名逆向排序。	3
ksort()	对数组按照键名排序。	3
list()	把数组中的值赋给一些变量。	3
natcasesort()	用“自然排序”算法对数组进行不区分大小写字母的排序。	4
natsort()	用“自然排序”算法对数组排序。	4
next()	将数组中的内部指针向前移动一位。	3
pos()	current() 的别名。	3
prev()	将数组的内部指针倒回一位。	3
range()	建立一个包含指定范围的元素的数组。	3
reset()	将数组的内部指针指向第一个元素。	3
rsort()	对数组逆向排序。	3
shuffle()	把数组中的元素按随机顺序重新排列。	3
sizeof()	count() 的别名。	3
sort()	对数组排序。	3
uasort()	使用用户自定义的比较函数对数组中的值进行排序并保持索引关联。	3
uksort()	使用用户自定义的比较函数对数组中的键名进行排序。	3
usort()	使用用户自定义的比较函数对数组中的值进行排序。	3

PHP Array 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CASE_LOWER	用在 array_change_key_case() 中将数组键名转换成小写字母。	
CASE_UPPER	用在 array_change_key_case() 中将数组键名转换成大写字母。	
SORT_ASC	用在 array_multisort() 函数中，使其升序排列。	
SORT_DESC	用在 array_multisort() 函数中，使其降序排列。	
SORT_REGULAR	用于对对象进行通常比较。	
SORT_NUMERIC	用于对对象进行数值比较。	
SORT_STRING	用于对对象进行字符串比较。	
SORT_LOCALE_STRING	基于当前区域来对对象进行字符串比较。	4
COUNT_NORMAL		
COUNT_RECURSIVE		
EXTR_OVERWRITE		
EXTR_SKIP		

EXTR_PREFIX_SAME
EXTR_PREFIX_ALL
EXTR_PREFIX_INVALID
EXTR_PREFIX_IF_EXIST
S
EXTR_IF_EXISTS
EXTR_REFS

PHP Calendar 函数

PHP Calendar 简介

当使用不同的历法格式时，calendar 函数很有用。它所基于的标准是儒略日计数 (Julian day count)。

编者注：Julian day count 是从 January 1, 4713 B.C. 开始计算的，中文译为儒略日计数或恺撒日计数。

请注意，Julian day count（儒略日计数） 与 Julian calendar（儒略历） 不是一回事。

注释：如需在日历格式之间转换，必须首先转换为 Julian day count，然后再转换为日历格式。

安装

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。

不过，如果您运行的是 PHP 的 Linux 版本，就不得不通过 --enable-calendar 编译 PHP，这样日历函数才能工作。

PHP Calendar 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
cal_days_in_month()	针对指定的年份和日历，返回一个月中的天数。	4
cal_from_jd()	把儒略日计数转换为指定日历的日期。	4
cal_info()	返回有关给定日历的信息。	4
cal_to_jd()	把日期转换为儒略日计数。	4
easter_date()	返回指定年份的复活节午夜的 Unix 时间戳。	3
easter_days()	返回指定年份的复活节与 3 月 21 日之间的天数。	3
FrenchToJD()	将法国共和历法转换成为儒略日计数。	3
GregorianToJD()	将格利高里历法转换成为儒略日计数。	3
JDDayOfWeek()	返回日期在周几。	3
JDMonthName()	返回月的名称。	3

JDToFrench()	把儒略日计数转换为法国共和国历法。	3
JDToGregorian()	把儒略日计数转换为格利高里历法。	3
jdtojewish()	把儒略日计数转换为犹太历法。	3
JDToJulian()	把儒略日计数转换为儒略历。	3
jdtounix()	把儒略日计数转换为 Unix 时间戳。	4
JewishToJD()	把犹太历法转换为儒略日计数。	3
JulianToJD()	把儒略历转换为儒略日计数。	3
unixtojd()	把 Unix 时间戳转换为儒略日计数。	4

PHP Calendar 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CAL_GREGORIAN	Gregorian calendar	3
CAL_JULIAN	Julian calendar	3
CAL_JEWISH	Jewish calendar	3
CAL_FRENCH	French Republican calendar	3
CAL_NUM_CALS		3
CAL_DOW_DAYNO		3
CAL_DOW_SHORT		3
CAL_DOW_LONG		3
CAL_MONTH_GREGORIAN_SHORT		3
CAL_MONTH_GREGORIAN_LONG		3
CAL_MONTH_JULIAN_SHORT		3
CAL_MONTH_JULIAN_LONG		3
CAL_MONTH_JEWISH		3
CAL_MONTH_FRENCH		3
CAL_EASTER_DEFAULT		4
CAL_EASTER_DEFAULT		4
CAL_EASTER_ROMAN		4
CAL_EASTER_ALWAYS_GREGORIAN		4
CAL_EASTER_ALWAYS_JULIAN		4
CAL_JEWISH_ADD_ALAFIM_GERESH		5
CAL_JEWISH_ADD_ALAFIM		5
CAL_JEWISH_ADD_GERESHAYIM		5

PHP Date / Time 函数

PHP Date / Time 简介

date/time 函数允许您提取并格式化服务器上的日期和时间。

注释：这些函数依赖于服务器的本地设置。

安装

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

日期/时间函数的行为受到 php.ini 中设置的影响。

Date/Time 配置选项：

名称	默认	描述	可改变
date.default_latitude	"31.7667"	规定默认纬度（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.default_longitude	"35.2333"	规定默认经度（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.sunrise_zenith	"90.83"	规定日出天顶（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.sunset_zenith	"90.83"	规定日落天顶（从 PHP 5 开始可用）。date_sunrise() 和 date_sunset() 使用该选项。	PHP_INI_ALL
date.timezone	""	规定默认时区（从 PHP 5.1 开始可用）。	PHP_INI_ALL

PHP Date / Time 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
checkdate()	验证格利高里日期。	3

date_default_timezone_get()	返回默认时区。	5
date_default_timezone_set()	设置默认时区。	5
date_sunrise()	返回给定的日期与地点的日出时间。	5
date_sunset()	返回给定的日期与地点的日落时间。	5
date()	格式化本地时间 / 日期。	3
getdate()	返回日期 / 时间信息。	3
gettimeofday()	返回当前时间信息。	3
gmdate()	格式化 GMT/UTC 日期/时间。	3
gmmktime()	取得 GMT 日期的 UNIX 时间戳。	3
gmstrftime()	根据本地区域设置格式化 GMT/UTC 时间 / 日期。	3
idate()	将本地时间/日期格式化为整数	5
localtime()	返回本地时间。	4
microtime()	返回当前时间的微秒数。	3
mktime()	返回一个日期的 Unix 时间戳。	3
strftime()	根据区域设置格式化本地时间 / 日期。	3
strptime()	解析由 strftime 生成的日期 / 时间。	5
strtotime()	将任何英文文本的日期或时间描述解析为 Unix 时间戳。	3
time()	返回当前时间的 Unix 时间戳。	3

PHP Date / Time 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
DATE_ATOM	原子钟格式 (如: 2005-08-15T16:13:03+0000)	
DATE_COOKIE	HTTP Cookies 格式 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_ISO8601	ISO-8601 (如: 2005-08-14T16:13:03+0000)	
DATE_RFC822	RFC 822 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC850	RFC 850 (如: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1036	RFC 1036 (如: Sunday, 14-Aug-05 16:13:03 UTC)	
DATE_RFC1123	RFC 1123 (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_RFC2822	RFC 2822 (如: Sun, 14 Aug 2005 16:13:03 +0000)	
DATE_RSS	RSS (如: Sun, 14 Aug 2005 16:13:03 UTC)	
DATE_W3C	World Wide Web Consortium (如: 2005-08-14T16:13:03+0000)	

PHP 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

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 Directory 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
chdir()	改变当前的目录。	3
chroot()	改变当前进程的根目录。	4
dir()	打开一个目录句柄，并返回一个对象。	3
closedir()	关闭目录句柄。	3
getcwd()	返回当前目录。	4
opendir()	打开目录句柄。	3
readdir()	返回目录句柄中的条目。	3
rewinddir()	重置目录句柄。	3
scandir()	列出指定路径中的文件和目录。	5

PHP Directory 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
DIRECTORY_SEPARATOR		3
PATH_SEPARATOR		4

PHP Error 和 Logging 函数

PHP Error 和 Logging 简介

error 和 logging 函数允许你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。

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()	输出 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	致命的运行时错误。错误无法恢复。脚本的执行被中断。	
2	E_WARNING	非致命的运行时错误。脚本的执行不会中断。	
4	E_PARSE	编译时语法解析错误。解析错误只应该由解析器生成。	
8	E_NOTICE	运行时提示。可能是错误，也可能在正常运行脚本时发生。	
16	E_CORE_ERROR	由 PHP 内部生成的错误。	4
32	E_CORE_WARNING	由 PHP 内部生成的警告。	4
64	E_COMPILE_ERROR	由 Zend 脚本引擎内部生成的错误。	4
128	E_COMPILE_WARNING	由 Zend 脚本引擎内部生成的警告。	4
256	E_USER_ERROR	由于调用 trigger_error() 函数生成的运行时错误。	4
512	E_USER_WARNING	由于调用 trigger_error() 函数生成的运行时警告。	4
1024	E_USER_NOTICE	由于调用 trigger_error() 函数生成的运行时提示。	4
2048	E_STRICT	运行时提示。对增强代码的互用性和兼容性有益。	5
4096	E_RECOVERABLE_ERROR	可捕获的致命错误。（参阅 set_error_handler()）	5
8191	E_ALL	所有的错误和警告，除了 E_STRICT。	5

PHP 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

文件系统函数的行为受到 php.ini 中设置的影响。

文件系统配置选项：

名称	默认	描述	可改变
allow_url_fopen	"1"	本选项激活了 URL 形式的 fopen 封装协议使得可以访问 URL 对象例如文件。默认的封装协议提供用 ftp 和 http 协议来访问远程文件，一些扩展库例如 zlib 可能会注册更多的封装协议。 （PHP 4.0.4 版以后可用。）	PHP_INI_SYSTEM
user_agent	NULL	定义 PHP 发送的 User-Agent。 （PHP 4.3.0 版以后可用。）	PHP_INI_ALL
default_socket_timeout	"60"	基于 socket 的流的默认超时时间(秒)。 （PHP 4.3.0 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"	当设为 On 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 Macintosh 的习惯。 这使得 PHP 可以和 Macintosh 系统交互操作，但是默认值是 Off，因为在检测第一行的 EOL 习惯时会有很小的性能损失，而且在 Unix 系统下使用回车符号作为项目分隔符的人们会遭遇向下不兼容的行为。 （PHP 4.3.0 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 Filesystem 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
basename()	返回路径中的文件名部分。	3
chgrp()	改变文件组。	3
chmod()	改变文件模式。	3
chown()	改变文件所有者。	3
clearstatcache()	清除文件状态缓存。	3
copy()	复制文件。	3
delete()	参见 unlink() 或 unset()。	
dirname()	返回路径中的目录名称部分。	3
disk_free_space()	返回目录的可用空间。	4
disk_total_space()	返回一个目录的磁盘总容量。	4
diskfreespace()	disk_free_space() 的别名。	3
fclose()	关闭打开的文件。	3
feof()	测试文件指针是否到了文件结束的位置。	3
fflush()	向打开的文件输出缓冲内容。	4
fgetc()	从打开的文件中返回字符。	3
fgetcsv()	从打开的文件中解析一行，校验 CSV 字段。	3
fgets()	从打开的文件中返回一行。	3
fgetss()	从打开的文件中读取一行并过滤掉 HTML 和 PHP 标记。	3
file()	把文件读入一个数组中。	3

file_exists()	检查文件或目录是否存在。	3
file_get_contents()	将文件读入字符串。	4
file_put_contents	将字符串写入文件。	5
fileatime()	返回文件的上次访问时间。	3
filectime()	返回文件的上次改变时间。	3
filegroup()	返回文件的组 ID。	3
fileinode()	返回文件的 inode 编号。	3
filemtime()	返回文件的上次修改时间。	3
fileowner()	文件的 user ID （所有者）。	3
fileperms()	返回文件的权限。	3
filesize()	返回文件大小。	3
filetype()	返回文件类型。	3
flock()	锁定或释放文件。	3
fnmatch()	根据指定的模式来匹配文件名或字符串。	4
fopen()	打开一个文件或 URL。	3
fpassthru()	从打开的文件中读数据，直到 EOF，并向输出缓冲写结果。	3
fputcsv()	将行格式化为 CSV 并写入一个打开的文件中。	5
fputs()	fwrite() 的别名。	3
fread()	读取打开的文件。	3
fscanf()	根据指定的格式对输入进行解析。	4
fseek()	在打开的文件中定位。	3
fstat()	返回关于一个打开的文件的信息。	4
ftell()	返回文件指针的读/写位置	3
ftruncate()	将文件截断到指定的长度。	4
fwrite()	写入文件。	3
glob()	返回一个包含匹配指定模式的文件名/目录的数组。	4
is_dir()	判断指定的文件名是否是一个目录。	3
is_executable()	判断文件是否可执行。	3
is_file()	判断指定文件是否为常规的文件。	3
is_link()	判断指定的文件是否是连接。	3
is_readable()	判断文件是否可读。	3
is_uploaded_file()	判断文件是否是通过 HTTP POST 上传的。	3
is_writable()	判断文件是否可写。	4
is_writeable()	is_writable() 的别名。	3
link()	创建一个硬连接。	3
linkinfo()	返回有关一个硬连接的信息。	3
lstat()	返回关于文件或符号连接的信息。	3
mkdir()	创建目录。	3
move_uploaded_file()	将上传的文件移动到新位置。	4

parse_ini_file()	解析一个配置文件。	4
pathinfo()	返回关于文件路径的信息。	4
pclose()	关闭有 popen() 打开的进程。	3
popen()	打开一个进程。	3
readfile()	读取一个文件，并输出到输出缓冲。	3
readlink()	返回符号连接的目标。	3
realpath()	返回绝对路径名。	4
rename()	重名名文件或目录。	3
rewind()	倒回文件指针的位置。	3
rmdir()	删除空的目录。	3
set_file_buffer()	设置已打开文件的缓冲大小。	3
stat()	返回关于文件的信息。	3
symlink()	创建符号连接。	3
tempnam()	创建唯一的临时文件。	3
tmpfile()	建立临时文件。	3
touch()	设置文件的访问和修改时间。	3
umask()	改变文件的文件权限。	3
unlink()	删除文件。	3

PHP Filesystem 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
GLOB_BRACE		
GLOB_ONLYDIR		
GLOB_MARK		
GLOB_NOSORT		
GLOB_NOCHECK		
GLOB_NOESCAPE		
PATHINFO_DIRNAME		
PATHINFO_BASENAME		
PATHINFO_EXTENSION		
FILE_USE_INCLUDE_PATH		
FILE_APPEND		
FILE_IGNORE_NEW_LINES		
FILE_SKIP_EMPTY_LINES		

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()	获取多项变量，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	在指定的范围以整数验证值。
FILTER_VALIDATE_BOOLEAN	如果是 "1", "true", "on" 以及 "yes"，则返回 true，如果是 "0", "false", "off", "no" 以及 ""，则返回 false。否则返回 NULL。
FILTER_VALIDATE_FLOAT	以浮点数验证值。
FILTER_VALIDATE_REGEXP	根据 regexp，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 来验证。
FILTER_VALIDATE_IP	把值作为 IP 地址来验证。

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、重名命、删除及获取文件服务器上的文件信息。不是所有 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 版本已经内置该 FTP 扩展模块的支持。无需加载任何附加扩展库即可使用这些函数。

不过，如果您运行的是 PHP 的 Linux 版本，在编译的时候请添加 --enable-ftp 选项 (PHP4 或以上版本) 或者 --with-ftp (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 服务器上的文件。	3
ftp_exec()	在 FTP 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()	返回当前 FTP 连接的各种不同的选项设置。	4
ftp_get()	从 FTP 服务器下载文件。	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 服务器创建一个新目录。	3
ftp_nb_continue()	连续获取 / 发送文件 (non-blocking)。	4
ftp_nb_fget()	从 FTP 服务器上下载文件并保存到本地已经打开的文件中(non-blocking)	4
ftp_nb_fput()	上传已打开的文件，并在 FTP 服务器上把它保存为文件(non-blocking)。	4
ftp_nb_get()	从 FTP 服务器下载文件 (non-blocking)。	4
ftp_nb_put()	把文件上传到服务器 (non-blocking)。	4
ftp_nlist()	返回指定目录的文件列表。	3
ftp_pasv()	返回当前 FTP 被动模式是否打开。	3
ftp_put()	把文件上传到服务器。	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 服务器上的目录。	3
ftp_set_option()	设置各种 FTP 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()	返回远程 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
	为 GET 和 PUT 请求自动决定恢复和开始的位置	
FTP_AUTORESUME	只能工作在 FTP_AUTOSEEK 打开的情况下	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 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 libxml 函数

PHP libxml 简介

libxml 函数和常量与 SimpleXML, XSLT 以及 DOM 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
	更改空标签（比如 改为 </br>）。	
LIBXML_NOEMPTYTAG	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）。	5
LIBXML_DOTTED_VERSION	获得有点号的 libxml 版本（例如：2.6.5 或 2.6.17）。	5

PHP Mail 函数

PHP Mail 简介

HTTP 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

邮件函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

运行时配置

邮件函数的行为受 php.ini 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 段口号。自 PHP 4.3 起可用。	PHP_INI_ALL
sendmail_from	NULL	Windows 专用：规定从 PHP 发送的邮件中使用的 "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 Math 函数

PHP Math 简介

数学 (Math) 函数能处理 integer 和 float 范围内的值。

安装

数学 (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 Math 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
abs()	绝对值。	3
acos()	反余弦。	3
acosh()	反双曲余弦。	4
asin()	反正弦。	3
asinh()	反双曲正弦。	4
atan()	反正切。	3
atan2()	两个参数的反正切。	3
atanh()	反双曲正切。	4
base_convert()	在任意进制之间转换数字。	3
bindec()	把二进制转换为十进制。	3

ceil()	向上舍入为最接近的整数。	3
cos()	余弦。	3
cosh()	双曲余弦。	4
decbin()	把十进制转换为二进制。	3
dechex()	把十进制转换为十六进制。	3
decoct()	把十进制转换为八进制。	3
deg2rad()	将角度转换为弧度。	3
exp()	返回 E^x 的值。	3
expm1()	返回 E^x - 1 的值。	4
floor()	向下舍入为最接近的整数。	3
fmod()	返回除法的浮点数余数。	4
getrandmax()	显示随机数最大的可能值。	3
hexdec()	把十六进制转换为十进制。	3
hypot()	计算直角三角形的斜边长度。	4
is_finite()	判断是否为有限值。	4
is_infinite()	判断是否为无限值。	4
is_nan()	判断是否为合法数值。	4
lcg_value()	返回范围为 (0, 1) 的一个伪随机数。	4
log()	自然对数。	3
log10()	以 10 为底的对数。	3
log1p()	返回 log(1 + number)。	4
max()	返回最大值。	3
min()	返回最小值。	3
mt_getrandmax()	显示随机数的最大可能值。	3
mt_rand()	使用 Mersenne Twister 算法返回随机整数。	3
mt_srand()	播种 Mersenne Twister 随机数生成器。	3
octdec()	把八进制转换为十进制。	3
pi()	返回圆周率的值。	3
pow()	返回 x 的 y 次方。	3
rad2deg()	把弧度数转换为角度数。	3
rand()	返回随机整数。	3
round()	对浮点数进行四舍五入。	3
sin()	正弦。	3
sinh()	双曲正弦。	4
sqrt()	平方根。	3
srand()	播下随机数发生器种子。	3
tan()	正切。	3
tanh()	双曲正切。	4

PHP Math 常量

常量名	常量名	常量值	PHP
M_E	e	2.7182818284590452354	4
M_EULER	Euler 常量	0.57721566490153286061	5.2.0
M_LNPI	log_e(pi)	1.14472988584940017414	5.2.0
M_LN2	log_e 2	0.69314718055994530942	4
M_LN10	log_e 10	2.30258509299404568402	4
M_LOG2E	log_2 e	1.4426950408889634074	4
M_LOG10E	log_10 e	0.43429448190325182765	4
M_PI	Pi	3.14159265358979323846	3
M_PI_2	pi/2	1.57079632679489661923	4
M_PI_4	pi/4	0.78539816339744830962	4
M_1_PI	1/pi	0.31830988618379067154	4
M_2_PI	2/pi	0.63661977236758134308	4
M_SQRTPI	sqrt(pi)	1.77245385090551602729	5.2.0
M_2_SQRTPI	2/sqrt(pi)	1.12837916709551257390	4
M_SQRT1_2	1/sqrt(2)	0.70710678118654752440	4
M_SQRT2	sqrt(2)	1.41421356237309504880	4
M_SQRT3	sqrt(3)	1.73205080756887729352	5.2.0

PHP MySQL 函数

PHP MySQL 简介

MySQL 函数允许您访问 MySQL 数据库服务器。

安装

为了能够顺利的使用本类函数，必须在编译 PHP 时添加对 MySQL 的支持。

编译时，只要使用 --with-mysql[=DIR] 配置选项即可，其中可选的 [DIR] 指向 MySQL 的安装目录。

虽然本 MySQL 扩展库兼容 MySQL 4.1.0 及其以后版本，但是它不支持这些版本提供的额外功能。要使用这些功能，请使用 MySQLi 扩展库。

如果要同时安装 mysql 扩展库和 mysqli 扩展库，必须使用同一个客户端库以避免任何冲突。

在 Linux 系统上安装

PHP 4

默认开启了 --with-mysql 选项。此默认行为可以用 --without-mysql 配置选项来禁止。如果启用 MySQL 而不指定安装目录的话，PHP 将使用绑定的 MySQL 客户端连接库。

还有其它应用程序使用 MySQL（例如 auth-mysql）的用户不要用绑定的库，而要指定 MySQL 的安装目录，如这样：--with-mysql=/path/to/mysql。这将强制 PHP 使用随 MySQL

安装的客户端连接库，就可以避免任何冲突。

PHP 5+

MySQL 默认未启用，也没有绑定的 MySQL 库。使用 --with-mysql[=DIR] 配置选项来加入 MySQL 的支持。可以从 MySQL 下载头文件和库。

在 Windows 系统上安装

PHP 4

PHP MySQL 扩展已经编译入 PHP。

PHP 5+

MySQL 默认未启用，因此必须在 php.ini 中激活 php_mysql.dll 动态连接库。此外，PHP 还需要访问 MySQL 客户端连接库。PHP 的 Windows 发行版包括了一个 libmysql.dll，为了让 PHP 能和 MySQL 对话，此文件必须放在 Windows 的系统路径 PATH 中。

要激活任何 PHP 扩展库（例如 php_mysql.dll），PHP 指令 extension_dir 要被设为 PHP 扩展库所在的目录。PHP 5 下 extension_dir 取值的一个例子是 c:\php\ext。

注释：如果启动 web 服务器时出现类似如下的错误："Unable to load dynamic library './php_mysql.dll'"，这是因为系统找不到 php_mysql.dll 和 / 或 libmysql.dll。

Runtime 配置

MySQL 函数的行为受到 php.ini 中设置的影响。

MySQL 配置选项：

名称	默认	描述	可更改
mysql.allow_persistent	"1"	是否允许 MySQL 的持久连接。	PHP_INI_SYSTEM
mysql.max_persistent	"-1"	每个进程中最大的持久连接数目。	PHP_INI_SYSTEM
mysql.max_links	"-1"	每个进程中最大的连接数，包括持久连接。	PHP_INI_SYSTEM
mysql.trace_mode	"0"	跟踪模式。从 PHP 4.3.0 起可用。	PHP_INI_ALL
mysql.default_port	NULL	指定默认连接数据库的 TCP 端口号。	PHP_INI_ALL
mysql.default_socket	NULL	默认的 socket 名称。PHP 4.0.1 起可用。	PHP_INI_ALL
mysql.default_host	NULL	默认的服务器地址。不适用于 SQL 安全模式。	PHP_INI_ALL
mysql.default_user	NULL	默认使用的用户名。不适用于 SQL 安全模式。	PHP_INI_ALL
mysql.default_password	NULL	默认使用的密码。不适用于 SQL 安全模式。	PHP_INI_ALL
mysql.connect_timeout	"60"	连接超时秒数。	PHP_INI_ALL

资源类型

在 MySQL 模块中使用了两种资源类型。第一种是数据库的连接句柄，第二种是 SQL 查询返回的结果集。

PHP MySQL 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
mysql_affected_rows()	取得前一次 MySQL 操作所影响的记录行数。	3
mysql_change_user()	不赞成。改变活动连接中登录的用户	3
mysql_client_encoding()	返回当前连接的字符集的名称	4
mysql_close()	关闭非持久的 MySQL 连接。	3
mysql_connect()	打开非持久的 MySQL 连接。	3
mysql_create_db()	不赞成。新建 MySQL 数据库。使用 mysql_query() 代替。	3
mysql_data_seek()	移动记录指针。	3
mysql_db_name()	从对 mysql_list_dbs() 的调用返回数据库名称。	3
mysql_db_query()	不赞成。发送一条 MySQL 查询。 使用 mysql_select_db() 和 mysql_query() 代替。	3
mysql_drop_db()	不赞成。丢弃（删除）一个 MySQL 数据库。 使用 mysql_query() 代替。	3
mysql_errno()	返回上一个 MySQL 操作中的错误信息的数字编码。	3
mysql_error()	返回上一个 MySQL 操作产生的文本错误信息。	3
mysql_escape_string()	不赞成。转义一个字符串用于 mysql_query。 使用 mysql_real_escape_string() 代替。	4
mysql_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。	3
mysql_fetch_assoc()	从结果集中取得一行作为关联数组。	4
mysql_fetch_field()	从结果集中取得列信息并作为对象返回。	3
mysql_fetch_lengths()	取得结果集中每个字段的内容的长度。	3
mysql_fetch_object()	从结果集中取得一行作为对象。	3
mysql_fetch_row()	从结果集中取得一行作为数字数组。	3
mysql_field_flags()	从结果中取得和指定字段关联的标志。	3
mysql_field_len()	返回指定字段的长度。	3
mysql_field_name()	取得结果中指定字段的字段名。	3
mysql_field_seek()	将结果集中的指针设定为指定的字段偏移量。	3

mysql_field_table()	取得指定字段所在的表名。	3
mysql_field_type()	取得结果集中指定字段的类型。	3
mysql_free_result()	释放结果内存。	3
mysql_get_client_info()	取得 MySQL 客户端信息。	4
mysql_get_host_info()	取得 MySQL 主机信息。	4
mysql_get_proto_info()	取得 MySQL 协议信息。	4
mysql_get_server_info()	取得 MySQL 服务器信息。	4
mysql_info()	取得最近一条查询的信息。	4
mysql_insert_id()	取得上一步 INSERT 操作产生的 ID。	3
mysql_list_dbs()	列出 MySQL 服务器中所有的数据库。	3
mysql_list_fields()	不赞成。列出 MySQL 结果中的字段。使用 mysql_query() 代替。	3
mysql_list_processes()	列出 MySQL 进程。	4
mysql_list_tables()	不赞成。列出 MySQL 数据库中的表。使用 Use mysql_query() 代替。	3
mysql_num_fields()	取得结果集中字段的数目。	3
mysql_num_rows()	取得结果集中行的数目。	3
mysql_pconnect()	打开一个到 MySQL 服务器的持久连接。	3
mysql_ping()	Ping 一个服务器连接，如果没有连接则重新连接。	4
mysql_query()	发送一条 MySQL 查询。	3
mysql_real_escape_string()	转义 SQL 语句中使用的字符串中的特殊字符。	4
mysql_result()	取得结果数据。	3
mysql_select_db()	选择 MySQL 数据库。	3
mysql_stat()	取得当前系统状态。	4
mysql_tablename()	不赞成。取得表名。使用 mysql_query() 代替。	3
mysql_thread_id()	返回当前线程的 ID。	4
mysql_unbuffered_query()	向 MySQL 发送一条 SQL 查询（不获取 / 缓存结果）。	4

PHP MySQL 常量

在 PHP 4.3.0 以后的版本中，允许在 mysql_connect() 函数和 mysql_pconnect() 函数中指定更多的客户端标记：

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
MYSQL_CLIENT_COMPRESS	使用压缩的通讯协议。	4.3
MYSQL_CLIENT_IGNORE_SP	允许在函数名后留空格位。	4.3

ACE		
MYSQL_CLIENT_INTERACTIVE	在关闭连接前所允许的交互超时非活动时间。	4.3
MYSQL_CLIENT_SSL	使用 SSL 加密（仅在 MySQL 客户端库版本为 4+ 时可用）。	4.3

mysql_fetch_array() 函数使用一个常量来表示所返回数组的类型：

常量	描述	PHP
MYSQL_ASSOC	返回的数据列使用字段名作为数组的索引名。	
MYSQL_BOTH	返回的数据列使用字段名及数字索引作为数组的索引名。	
MYSQL_NUM	返回的数据列使用数字索引作为数组的索引名。 索引从 0 开始，表示返回结果的第一个字段。	

PHP SimpleXML 函数

PHP SimpleXML 简介

SimpleXML 函数允许您把 XML 转换为对象。

通过普通的属性选择器或数组迭代器，可以处理这个对象，就像处理任何其他对象一样。

其中的一些函数需要最新的 PHP 版本。

安装

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 SimpleXML 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
__construct()	创建一个新的 SimpleXMLElement 对象。	5
addAttribute()	给 SimpleXML 元素添加一个属性。	5
addChild()	给 SimpleXML 元素添加一个子元素。	5
asXML()	从 SimpleXML 元素获取 XML 字符串。	5
attributes()	获取 SimpleXML 元素的属性。	5
children()	获取指定节点的子。	5

getDocNamespaces()	获取 XML 文档的命名空间。	5
getName()	获取 SimpleXML 元素的名称。	5
getNamespaces()	从 XML 数据获取命名空间。	5
registerXPathNamespace()	为下一次 XPath 查询创建命名空间语境。	5
simplexml_import_dom()	从 DOM 节点获取 SimpleXMLElement 对象。	5
simplexml_load_file()	从 XML 文档获取 SimpleXMLElement 对象。	5
simplexml_load_string()	从 XML 字符串获取 SimpleXMLElement 对象。	5
xpath()	对 XML 数据运行 XPath 查询。	5

PHP SimpleXML 常量

无。

- Previous Page
- Next Page

Search:

Go

PHP 参考手册

PHP 测验

PHP String 函数

PHP String 简介

String 字符串函数允许您对字符串进行操作。

安装

String 函数是 PHP 核心的组成部分。无需安装即可使用这些函数。

PHP String 函数

PHP：指示支持该函数的最早的 PHP 版本。

函数	描述	PHP
addcslashes()	在指定的字符前添加反斜杠。	4
addslashes()	在指定的预定义字符前添加反斜杠。	3
bin2hex()	把 ASCII 字符的字符串转换为十六进制值。	3
chop()	rtrim() 的别名。	3
chr()	从指定的 ASCII 值返回字符。	3
chunk_split()	把字符串分割为一连串更小的部分。	3

convert_cyr_string()	把字符由一种 Cyrillic 字符转换成另一种。	3
convert_uudecode()	对 uuencode 编码的字符串进行解码。	5
convert_uuencode()	使用 uuencode 算法对字符串进行编码。	5
count_chars()	返回字符串所用字符的信息。	4
crc32()	计算一个字符串的 32-bit CRC。	4
crypt()	单向的字符串加密法 (hashing)。	3
echo()	输出字符串。	3
explode()	把字符串打散为数组。	3
fprintf()	把格式化的字符串写到指定的输出流。	5
get_html_translation_table()	返回翻译表。	4
hebrev()	把希伯来文本从右至左的流转换为左至右的流。	3
hebrevc()	同上，同时把(\n) 转为 。	3
html_entity_decode()	把 HTML 实体转换为字符。	4
htmlentities()	把字符转换为 HTML 实体。	3
htmlspecialchars_decode()	把一些预定义的 HTML 实体转换为字符。	5
htmlspecialchars()	把一些预定义的字符转换为 HTML 实体。	3
implode()	把数组元素组合为一个字符串。	3
join()	implode() 的别名。	3
levenshtein()	返回两个字符串之间的 Levenshtein 距离。	3
localeconv()	返回包含本地数字及货币信息格式的数组。	4
ltrim()	从字符串左侧删除空格或其他预定义字符。	3
md5()	计算字符串的 MD5 散列。	3
md5_file()	计算文件的 MD5 散列。	4
metaphone()	计算字符串的 metaphone 键。	4
money_format()	把字符串格式化为货币字符串。	4
nl_langinfo()	返回指定的本地信息。	4
nl2br()	在字符串中的每个新行之前插入 HTML 换行符。	3
number_format()	通过千位分组来格式化数字。	3
ord()	返回字符串第一个字符的 ASCII 值。	3
parse_str()	把查询字符串解析到变量中。	3
print()	输出一个或多个字符串。	3
printf()	输出格式化的字符串。	3
quoted_printable_decode()	解码 quoted-printable 字符串。	3
quotemeta()	在字符串中某些预定义的字符前添加反斜杠。	3
rtrim()	从字符串的末端开始删除空白字符或其他预定义字符。	3
setlocale()	设置地区信息（地域信息）。	3
sha1()	计算字符串的 SHA-1 散列。	4
sha1_file()	计算文件的 SHA-1 散列。	4

similar_text()	计算两个字符串的匹配字符的数目。	3
soundex()	计算字符串的 soundex 键。	3
sprintf()	把格式化的字符串写写入一个变量中。	3
sscanf()	根据指定的格式解析来自一个字符串的输入。	4
str_ireplace()	替换字符串中的一些字符。（对大小写不敏感）	5
str_pad()	把字符串填充为新的长度。	4
str_repeat()	把字符串重复指定的次数。	4
str_replace()	替换字符串中的一些字符。（对大小写敏感）	3
str_rot13()	对字符串执行 ROT13 编码。	4
str_shuffle()	随机地打乱字符串中的所有字符。	4
str_split()	把字符串分割到数组中。	5
str_word_count()	计算字符串中的单词数。	4
strcasecmp()	比较两个字符串。（对大小写不敏感）	3
strchr()	搜索字符串在另一字符串中的第一次出现。strstr() 的别名	3
strcmp()	比较两个字符串。（对大小写敏感）	3
strcoll()	比较两个字符串（根据本地设置）。	4
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。	3
strip_tags()	剥去 HTML、XML 以及 PHP 的标签。	3
stripcslashes()	删除由 addcslashes() 函数添加的反斜杠。	4
stripslashes()	删除由 addslashes() 函数添加的反斜杠。	3
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)	5
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)	3
strlen()	返回字符串的长度。	3
strnatcasecmp()	使用一种“自然”算法来比较两个字符串（对大小写不敏感）	4
strnatcmp()	使用一种“自然”算法来比较两个字符串（对大小写敏感）	4
strncasecmp()	前 n 个字符的字符串比较（对大小写不敏感）。	4
strncmp()	前 n 个字符的字符串比较（对大小写敏感）。	4
strpbrk()	在字符串中搜索指定字符中的任意一个。	5
strpos()	返回字符串在另一字符串中首次出现的位置（对大小写敏感）	3
strrchr()	查找字符串在另一个字符串中最后一次出现的位置。	3
strrev()	反转字符串。	3
strripos()	查找字符串在另一字符串中最后出现的位置(对大小写不敏感)	5
strrpos()	查找字符串在另一字符串中最后出现的位置(对大小写敏感)	3
strspn()	返回在字符串中包含的特定字符的数目。	3
strstr()	搜索字符串在另一字符串中的首次出现（对大小写敏感）	3

strtok()	把字符串分割为更小的字符串。	3
strtolower()	把字符串转换为小写。	3
strtoupper()	把字符串转换为大写。	3
strtr()	转换字符串中特定的字符。	3
substr()	返回字符串的一部分。	3
substr_compare()	从指定的开始长度比较两个字符串。	5
substr_count()	计算子串在字符串中出现的次数。	4
substr_replace()	把字符串的一部分替换为另一个字符串。	4
trim()	从字符串的两端删除空白字符和其他预定义字符。	3
ucfirst()	把字符串中的首字符转换为大写。	3
ucwords()	把字符串中每个单词的首字符转换为大写。	3
vfprintf()	把格式化的字符串写到指定的输出流。	5
vprintf()	输出格式化的字符串。	4
vsprintf()	把格式化字符串写入变量中。	4
wordwrap()	按照指定长度对字符串进行折行处理。	4

PHP String 常量

PHP：指示支持该常量的最早的 PHP 版本。

常量	描述	PHP
CRYPT_SALT_LENGTH	包含系统默认加密方法的长度。对于标准 DES 加密，长度是 2。	
CRYPT_STD_DES	如果支持 2 字符 salt 的 DES 加密，则设置为 1，否则为 0。	
CRYPT_EXT_DES	如果支持 9 字符 salt 的 DES 加密，则设置为 1，否则为 0。	
CRYPT_MD5	如果支持以1开始的 12 字符 salt 的 MD5 加密，则设置为 1，否则为 0。	
CRYPT_BLOWFISH	如果支持以 2 或 $2a$ 开始的 16 字符 salt 的 Blowfish 加密，则设置为 1，否则为 0。	
HTML_SPECIALCHARS		
HTML_ENTITIES		
ENT_COMPAT		
ENT_QUOTES		
ENT_NOQUOTES		
CHAR_MAX		
LC_CTYPE		
LC_NUMERIC		
LC_TIME		
LC_COLLATE		
LC_MONETARY		

LC_ALL
LC_MESSAGES
STR_PAD_LEFT
STR_PAD_RIGHT
STR_PAD_BOTH

PHP XML Parser 函数

PHP XML Parser 简介

XML 函数允许我们解析 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。

由于它基于事件，且无验证，Expat 具有快速并适合 web 应用程序的特性。

XML 解析器函数允许我们创建 XML 解析器，并为 XML 事件定义句柄。

安装

XML 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 解析器的错误描述。	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 解析器获取选项设置信息。	3
xml_parser_set_option()	为 XML 解析进行选项设置。	3
xml_set_character_data_handler()	建立字符数据处理器。	3
xml_set_default_handler()	建立默认的数据处理器。	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()	建立注释声明处理器。	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()	建立未解析实体定义声明处理器。	3

PHP XML Parser 常量

Constant
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 Zip File 函数

PHP Zip File 简介

压缩文件函数允许我们读取压缩文件。

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 之内启用 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 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这个页面。

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

杂项函数函数的行为受到 php.ini 中设置的影响。

杂项函数配置选项：

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"	供突出显示 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	浏览器性能文件的名称和位置（例如：browscap.ini）。	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。	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()	把字符串按照 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()	对文件进行语法高亮显示。	4
highlight_string()	对字符串进行语法高亮显示。	4
ignore_user_abort()	设置与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 Date / Time 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