[**什么是 PHP 文件？** 3](#_Toc278751384)

[**PHP 中的注释** 4](#_Toc278751385)

[**PHP 是一门松散类型的语言（Loosely Typed Language）** 4](#_Toc278751386)

[**变量的命名规则** 5](#_Toc278751387)

[ 并置运算符（Concatenation Operator） 5](#_Toc278751388)

[ 使用 strlen() 函数 5](#_Toc278751389)

[ 使用 strpos() 函数 6](#_Toc278751390)

[**PHP 运算符** 6](#_Toc278751391)

[PHP If...Else 语句 7](#_Toc278751392)

[PHP Switch 语句 8](#_Toc278751393)

[PHP 数组 8](#_Toc278751394)

[数值数组 9](#_Toc278751395)

[关联数组 9](#_Toc278751396)

[多维数组 10](#_Toc278751397)

[PHP 循环 10](#_Toc278751398)

[while 语句 11](#_Toc278751399)

[do...while 语句 12](#_Toc278751400)

[for 语句 12](#_Toc278751401)

[foreach 语句 13](#_Toc278751402)

[PHP 函数 14](#_Toc278751403)

[创建 PHP 函数 14](#_Toc278751404)

[PHP 函数 - 返回值 16](#_Toc278751405)

[PHP 表单和用户输入 17](#_Toc278751406)

[PHP $\_GET 17](#_Toc278751407)

[$\_GET 变量 17](#_Toc278751408)

[$\_REQUEST 变量 18](#_Toc278751409)

[PHP $\_POST 18](#_Toc278751410)

[$\_POST 变量 18](#_Toc278751411)

[为什么使用 $\_POST？ 19](#_Toc278751412)

[PHP Date() 19](#_Toc278751413)

[PHP Date() 函数 19](#_Toc278751414)

[PHP 日期 - 什么是时间戳（Timestamp）？ 19](#_Toc278751415)

[PHP 日期 - 格式化日期 19](#_Toc278751416)

[PHP 日期 - 添加时间戳 20](#_Toc278751417)

[PHP 日期 - 参考手册 20](#_Toc278751418)

[PHP 引用文件 21](#_Toc278751419)

[服务器端引用（Server Side Includes） 21](#_Toc278751420)

[include() 函数 21](#_Toc278751421)

[require() 函数 23](#_Toc278751422)

[PHP 文件处理 23](#_Toc278751423)

[打开文件 23](#_Toc278751424)

[关闭文件 24](#_Toc278751425)

[检测 End-of-file 25](#_Toc278751426)

[逐行 读取文件 25](#_Toc278751427)

[逐字符 读取文件 25](#_Toc278751428)

[PHP Filesystem 参考手册 26](#_Toc278751429)

[PHP 文件上传 26](#_Toc278751430)

[创建一个文件上传表单 26](#_Toc278751431)

[创建上传脚本 27](#_Toc278751432)

[上传限制 27](#_Toc278751433)

[保存 被上传的文件 28](#_Toc278751434)

[PHP Cookies 29](#_Toc278751435)

[如何创建 cookie？ 29](#_Toc278751436)

[如何取回 Cookie 的值？ 30](#_Toc278751437)

[如何删除 cookie？ 31](#_Toc278751438)

[如果浏览器不支持 cookie 该怎么办？ 31](#_Toc278751439)

[PHP Sessions 32](#_Toc278751440)

[PHP Session 变量 32](#_Toc278751441)

[开始 PHP Session 32](#_Toc278751442)

[存储 Session 变量 32](#_Toc278751443)

[终结 Session 33](#_Toc278751444)

[PHP 发送电子邮件 34](#_Toc278751445)

[PHP mail() 函数 34](#_Toc278751446)

[PHP 简易 E-Mail 34](#_Toc278751447)

[PHP Mail Form 35](#_Toc278751448)

[PHP Mail 参考手册 36](#_Toc278751449)

[PHP 安全的电子邮件 36](#_Toc278751450)

[PHP E-mail 注入 36](#_Toc278751451)

[PHP 防止 E-mail 注入 37](#_Toc278751452)

[PHP 错误处理 39](#_Toc278751453)

[PHP 错误处理 39](#_Toc278751454)

[基本的错误处理：使用 die() 函数 39](#_Toc278751455)

[创建自定义错误处理器 40](#_Toc278751456)

[Set Error Handler 41](#_Toc278751457)

[触发错误 42](#_Toc278751458)

[错误记录 44](#_Toc278751459)

[PHP 异常处理 45](#_Toc278751460)

[什么是异常？ 45](#_Toc278751461)

[异常的基本使用 45](#_Toc278751462)

[创建一个自定义的 Exception 类 47](#_Toc278751463)

[多个异常 49](#_Toc278751464)

[重新抛出异常 50](#_Toc278751465)

[设置顶层异常处理器 （Top Level Exception Handler） 51](#_Toc278751466)

[异常的规则 52](#_Toc278751467)

[PHP 过滤器（Filter） 52](#_Toc278751468)

[什么是 PHP 过滤器？ 52](#_Toc278751469)

[为什么使用过滤器？ 52](#_Toc278751470)

[函数和过滤器 53](#_Toc278751471)

[Validating 和 Sanitizing 53](#_Toc278751472)

[选项和标志 54](#_Toc278751473)

[验证输入 55](#_Toc278751474)

[净化输入 55](#_Toc278751475)

[过滤多个输入 56](#_Toc278751476)

[使用 Filter Callback 57](#_Toc278751477)

**什么是 PHP 文件？**

* PHP 文件可包含文本、HTML 标签以及脚本
* PHP 文件向浏览器返回纯粹的 HTML
* PHP 文件的文件后缀是 ".php"、".php3" 或 ".phtml"

最佳组合:

Apache

PHP

MySQL

下面的链接提供了来自 PHP.net 的优秀教程，您可以从中学习到如何安装 PHP5：

<http://www.php.net/manual/en/install.php>

### 下载 PHP

从此处下载免费的 PHP：<http://www.php.net/downloads.php>

### 下载 MySQL Database

从此处下载免费的 MySQL：<http://www.mysql.com/downloads/index.html>

### 下载 Apache Server

从此处下载免费的 Apache：<http://httpd.apache.org/download.cgi>

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

?>

有两种通过 PHP 来输出文本的基础指令：*echo* 和 *print*。

**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。

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

$txt = "Hello World!";

$number = 16;

?>

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

**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）。

## 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

## 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

## 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。

**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

<html>

<body>

<?php

$d=date("D");

if ($d=="Fri"){

echo "Have a nice weekend!";

}else{

echo "Have a nice day!";

}

?>

</body>

</html>

# PHP Switch 语句

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

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;

}

# PHP 数组

### 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（脚标是数值）

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：

|  |
| --- |
| 一次给数组赋值  $names = array("Peter","Quagmire","Joe"); |
| <?php  $names[0] = "Peter";  $names[1] = "Quagmire";  $names[2] = "Joe";  //一个个的给数组赋值  echo $names[1] . " and " . $names[2] . " are ". $names[0] . "'s neighbors";  ?> |

## 关联数组(脚标是字符串)

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。

|  |
| --- |
| $ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34); |
| <?php  $ages['Peter'] = "32";  $ages['Quagmire'] = "30";  $ages['Joe'] = "34";  echo "Peter is " . $ages['Peter'] . " years old.";  ?> |

## 多维数组(用=>符号)

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。

|  |
| --- |
| $families = array(  "Griffin"=>array("Peter", "Lois", "Megan"),  "Quagmire"=>array("Glenn"),  "Brown"=>array("Cleveland", "Loretta", "Junior")  ); |
| Array(  [Griffin] => Array( [0] => Peter [1] => Lois [2] => Megan )  [Quagmire] => Array( [0] => Glenn )  [Brown] => Array( [0] => Cleveland [1] => Loretta [2] => Junior )  ) |

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?";

# 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。**

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

**在 PHP 中，提供了超过 700 个内建的函数。**

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。
6. <html>
7. <body>
8. <?php
9. function writeMyName(){
10. echo "David Yang";
11. }
12. writeMyName();
13. ?>
14. </body>
15. </html>

<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>

<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>

<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>

## PHP 函数 - 返回值

<html>

<body>

<?php

function add($x,$y){

$total = $x + $y;

return $total;

}

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

?>

</body>

</html>

# 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。**

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

# 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）。

|  |
| --- |
| <form action="welcome.php" method="get">  Name: <input type="text" name="name" />  Age: <input type="text" name="age" />  <input type="submit" />  </form> |
| Welcome <?php echo $\_GET["name"]; ?>.<br />  You are <?php echo $\_GET["age"]; ?> years old! |

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。

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（不会显示在浏览器的地址栏），并且对发送信息的量也没有限制。

<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>

"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，所以无法把页面加入书签。

# 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。

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 Timestamp）。

## 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）

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。

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);

?>

## 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](http://www.w3school.com.cn/php/php_ref_date.asp)。

# 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。

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>

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>

**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)。 |

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](http://www.w3school.com.cn/php/php_ref_filesystem.asp)。

# 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//如果存在错误, 则错误代码大于0

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，则把文件拷贝到名为 "upload"的文件夹。

# PHP Cookies

## 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？用:表单传递数据

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。

Warning: mail() [function.mail]: Failed to connect to mailserver at "localhost" port 25, verify your "SMTP" and "smtp\_port" setting in php.ini or use ini\_set() in D:\www\email.php on line 11

Mail Sent.

### 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](http://www.w3school.com.cn/php/php_ref_mail.asp)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'])){  $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{  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\_SANITIZE\_EMAIL 从字符串中删除电子邮件的非法字符

//filter\_var() validates the e-mail

//address using FILTER\_VALIDATE\_EMAIL

if(filter\_var($field, FILTER\_VALIDATE\_EMAIL)){ // FILTER\_VALIDATE\_EMAIL 验证电子邮件地址

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：

您可以在我们的 [PHP 过滤器](http://www.w3school.com.cn/php/php_filter.asp)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：

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](http://www.w3school.com.cn/php/php_ref_filter.asp)。

## 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