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：

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)	

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()。

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

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	PHP_INI_ALL

		时时间(秒)。 （PHP 4.3.0 版以后可用。）	
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		

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。

WebjxCom 提示：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。

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>
```

WebjxCom 提示：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。

WebjxCom 提示：PHP 允许您从脚本直接发送电子邮件。

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。

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.
3. 如果未填写（比如在页面被首次访问时），输出 HTML 表单
4.
5. 如果已填写（在表单被填写后），从表单发送邮件
6.
7. 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
8.

PHP Mail 参考手册

如需更多有关 PHP mail() 函数的信息：

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

WebjxCom 提示：在上一节中的 PHP e-mail 脚本中，存在着一个漏洞。

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

> WebjxCom 提示：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。

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。

> WebjxCom 提示：异常（Exception）用于在指定的错误发生时改变脚本的正常流程。

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。

WebjxCom 提示：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。

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"。

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