PHP 错误处理 PHP 过滤器

PHP 异常处理

异常用于在指定的错误发生时改变脚本的正常流程。

异常是什么

PHP 5 提供了一种新的面向对象的错误处理方法。

异常处理用于在指定的错误（异常）情况发生时改变脚本的正常流程。这种情况称为异常。

当异常被触发时，通常会发生：

- 当前代码状态被保存
- 代码执行被切换到预定义（自定义）的异常处理器函数
- 根据情况，处理器也许会从保存的代码状态重新开始执行代码，终止脚本执行，或从代码中另外的位置继续执行脚本

我们将展示不同的错误处理方法：

- 异常的基本使用
- 创建自定义的异常处理器
- 多个异常
- 重新抛出异常
- 设置顶层异常处理器

注释：异常应该仅仅在错误情况下使用，而不应该用于在一个指定的点跳转到代码的另一个位置。

异常的基本使用

当异常被抛出时，其后的代码不会继续执行，PHP 会尝试查找匹配的 "catch" 代码块。

如果异常没有被捕获，而且又没用使用 set_exception_handler() 作相应的处理的话，那么将发生一个严重的错误（致命错误），并且输出 "Uncaught Exception" （未捕获异常）的错误消息。

让我们尝试抛出一个异常，同时不去捕获它：

```
<?php
// 创建一个有异常处理的函数
function checkNum($number)
{
if($number>1)
{
throw new Exception("Value must be 1 or below");
}
return true;
}
// 触发异常
```

```
checkNum(2);
?>
```

上面的代码会得到类似这样的一个错误：

```
Fatal error: Uncaught exception 'Exception' with message 'Value must be 1 or below' in /www/runoob/test/test.php:7 Stack trace: #0 /www/runoob/test/test.php(13): checkNum(2) #1 {main} thrown in /www/runoob/test/test.php on line 7
```

Try、throw 和 catch

要避免上面实例中出现的错误，我们需要创建适当的代码来处理异常。

适当的处理异常代码应该包括：

1. Try - 使用异常的函数应该位于 "try" 代码块内。如果没有触发异常，则代码将照常继续执行。但是如果异常被触发，会抛出一个异常。
2. Throw - 里规定如何触发异常。每一个 "throw" 必须对应至少一个 "catch"。
3. Catch - "catch" 代码块会捕获异常，并创建一个包含异常信息的对象。

让我们触发一个异常：

```
<?php
// 创建一个有异常处理的函数
function checkNum($number)
{
if($number>1)
{
throw new Exception("变量值必须小于等于 1");
}
return true;
}
// 在 try 块 触发异常
try
{
checkNum(2);
// 如果抛出异常，以下文本不会输出
echo '如果输出该内容，说明 $number 变量';
}
// 捕获异常
catch(Exception $e)
{
echo 'Message: ' .$e->getMessage();
}
?>
```

上面代码将得到类似这样一个错误：

```
Message: 变量值必须小于等于 1
```

实例解释：

上面的代码抛出了一个异常，并捕获了它：

1. 创建 checkNum() 函数。它检测数字是否大于 1。如果是，则抛出一个异常。
2. 在 "try" 代码块中调用 checkNum() 函数。
3. checkNum() 函数中的异常被抛出。
4. "catch" 代码块接收到该异常，并创建一个包含异常信息的对象 ($e)。
5. 通过从这个 exception 对象调用 $e->getMessage()，输出来自该异常的错误消息。

然而，为了遵循 "每个 throw 必须对应一个 catch" 的原则，可以设置一个顶层的异常处理器来处理漏掉的错误。

创建一个自定义的 Exception 类

创建自定义的异常处理程序非常简单。我们简单地创建了一个专门的类，当 PHP 中发生异常时，可调用其函数。该类必须是 exception 类的一个扩展。

这个自定义的 customException 类继承了 PHP 的 exception 类的所有属性，您可向其添加自定义的函数。

我们开始创建 customException 类：

```php
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = '错误行号 '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> 不是一个合法的 E-Mail 地址';
return $errorMsg;
}
}
$email = "someone@example...com";
try
{
// 检测邮箱
if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new customException($email);
}
}
catch (customException $e)
{
//display custom message
echo $e->errorMessage();
}
?>
```

这个新的类是旧的 exception 类的副本，外加 errorMessage() 函数。正因为它是旧类的副本，因此它从旧类继承了属性和方法，我们可以使用 exception 类的方法，比如 getLine()、getFile() 和 getMessage()。

实例解释：

上面的代码抛出了一个异常，并通过一个自定义的 exception 类来捕获它：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一条错误消息。
3. 把 $email 变量设置为不合法的 e-mail 地址字符串。
4. 执行 "try" 代码块，由于 e-mail 地址不合法，因此抛出一个异常。
5. "catch" 代码块捕获异常，并显示错误消息。

多个异常

可以为一段脚本使用多个异常，来检测多种情况。

可以使用多个 if..else 代码块，或一个 switch 代码块，或者嵌套多个异常。这些异常能够使用不同的 exception 类，并返回不同的错误消息：

```
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = '错误行号 '.$this->getLine().' in '.$this->getFile()
.': <b>'.$this->getMessage().'</b> 不是一个合法的 E-Mail 地址';
return $errorMsg;
}
}
$email = "someone@example.com";
try
{
// 检测邮箱
if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new customException($email);
}
// 检测 "example" 是否在邮箱地址中
if(strpos($email, "example") !== FALSE)
{
throw new Exception("$email 是 example 邮箱");
}
}
catch (customException $e)
{
echo $e->errorMessage();
```

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

实例解释：

上面的代码测试了两种条件，如果其中任何一个条件不成立，则抛出一个异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 把 $email 变量设置为一个字符串，该字符串是一个有效的 e-mail 地址，但包含字符串 "example"。
4. 执行 "try" 代码块，在第一个条件下，不会抛出异常。
5. 由于 e-mail 含有字符串 "example"，第二个条件会触发异常。
6. "catch" 代码块会捕获异常，并显示恰当的错误消息。

如果 customException 类抛出了异常，但没有捕获 customException，仅仅捕获了 base exception，则在那里处理异常。

重新抛出异常

有时，当异常被抛出时，您也许希望以不同于标准的方式对它进行处理。可以在一个 "catch" 代码块中再次抛出异常。

脚本应该对用户隐藏系统错误。对程序员来说，系统错误也许很重要，但是用户对它们并不感兴趣。为了让用户更容易使用，您可以再次抛出带有对用户比较友好的消息的异常：

```
<?php
class customException extends Exception
{
public function errorMessage()
{
// 错误信息
$errorMsg = $this->getMessage().' 不是一个合法的 E-Mail 地址。';
return $errorMsg;
}
}
$email = "someone@example.com";
try
{
try
{
// 检测 "example" 是否在邮箱地址中
if(strpos($email, "example") !== FALSE)
{
// 如果是个不合法的邮箱地址，抛出异常
throw new Exception($email);
}
}
```

```
catch(Exception $e)
{
// 重新抛出异常
throw new customException($email);
}
}
catch (customException $e)
{
// 显示自定义信息
echo $e->errorMessage();
}
?>
```

实例解释：

上面的代码检测在邮件地址中是否含有字符串 "example"。如果有，则再次抛出异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧的 exception 类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 把 $email 变量设置为一个字符串，该字符串是一个有效的 e-mail 地址，但包含字符串 "example"。
4. "try" 代码块包含另一个 "try" 代码块，这样就可以再次抛出异常。
5. 由于 e-mail 包含字符串 "example"，因此触发异常。
6. "catch" 代码块捕获到该异常，并重新抛出 "customException"。
7. 捕获到 "customException"，并显示一条错误消息。

如果在当前的 "try" 代码块中异常没有被捕获，则它将在更高层级上查找 catch 代码块。

设置顶层异常处理器

set_exception_handler() 函数可设置处理所有未捕获异常的用户定义函数。

```
<?php
function myException($exception)
{
echo "<b>Exception:</b> " , $exception->getMessage();
}
set_exception_handler('myException');
throw new Exception('Uncaught Exception occurred');
?>
```

以上代码的输出如下所示：

```
Exception: Uncaught Exception occurred
```

在上面的代码中，不存在 "catch" 代码块，而是触发顶层的异常处理程序。应该使用此函数来捕获所有未被捕获的异常。

异常的规则

- 需要进行异常处理的代码应该放入 try 代码块内，以便捕获潜在的异常。
- 每个 try 或 throw 代码块必须至少拥有一个对应的 catch 代码块。
- 使用多个 catch 代码块可以捕获不同种类的异常。
- 可以在 try 代码块内的 catch 代码块中抛出（再次抛出）异常。

简而言之：如果抛出了异常，就必须捕获它。

PHP 错误处理 PHP 过滤器