第 7 章　错误和异常处理

在本章中，我们将介绍异常处理的概念以及PHP实现异常处理的机制。异常为以一种可扩展、可维护和面向对象的方式处理错误提供了统一的机制。

在本章中，我们将主要介绍以下内容：

- 异常处理的概念
- 异常控制结构：try...throw...catch
- Exception类
- 用户自定义异常
- Bob的汽车零部件商店应用程序的异常
- 异常和PHP的其他错误处理机制

7.1　异常处理的概念

异常处理的基本思想是代码在*try*代码块被调用执行。如下代码段所示的就是一个示例：

```
try
{
  // code goes here
}
```

如果try代码块出现某些错误，我们可以执行一个抛出异常的操作。某些编程语言，例如Java，在特定情况下将自动抛出异常。在PHP中，异常必须手动抛出。可以使用如下方式抛出一个异常：

```
throw new Exception( message', code);
```

throw关键字将触发异常处理机制。它是一个语言结构，而不是一个函数，但是必须给它传递一个值。它要求接收一个对象。在最简单的情况下，可以实例化一个内置的Exception类，就像以上代码所示。

这个类的构造函数需要两个参数：一个消息和一个代码。它们分别表示一个错误消息和错误代码号。这两个参数都是可选的。

最后，在try代码块之后，必须至少给出一个catch代码块。catch代码块可以如下所示：

```
catch ( typehint exception)
{
  // handle exception
}
```

可以将多个catch代码块与一个try代码块进行关联。如果每个catch代码块可以捕获一个不同类型的异常，那么使用多个catch代码块是有意义的。例如，如果希望捕获Exception类的异常，catch代码块可以如下所示：

```
catch (Exception $e)
{
  // handle exception
}
```

传递给catch代码块的对象（也是被catch代码块捕获的）就是导致异常并传递给throw语句的对象（被throw语句抛出）。该异常可以是任何类型的，但是使用Exception类的实例，或从Exception类继承过来并由用户定义的异常类实例，都是不错的选择（在本章的稍后内容中，我们将了解如何定义自己的异常）。

当产生一个异常时，PHP将查询一个匹配的catch代码块。如果有多个catch代码块，传递给每一个catch代码块的对象必须具有不同的类型，这样PHP可以找到需要进入哪一个catch代码块。

需要注意的另外一点是，还可以在一个catch代码块产生新的异常。

要更清楚地介绍这一点，我们来了解一个示例。如程序清单7-1所示的就是一个简单的异常处理示例。

程序清单7-1 basic_exception.php——抛出并捕获一个异常

```
<?php

try {
  throw new Exception('A terrible error has occurred', 42);
}
catch (Exception $e) {
  echo 'Exception '. $e->getCode(). ': '. $e->getMessage()."<br />".
  " in ". $e->getFile(). " on line ". $e->getLine(). '<br />';
}

?>
```

在程序清单7-1中，可以看到我们使用了Exception类的一些方法，稍后将简单介绍。程序清单7-1的运行结果如图7-1所示。

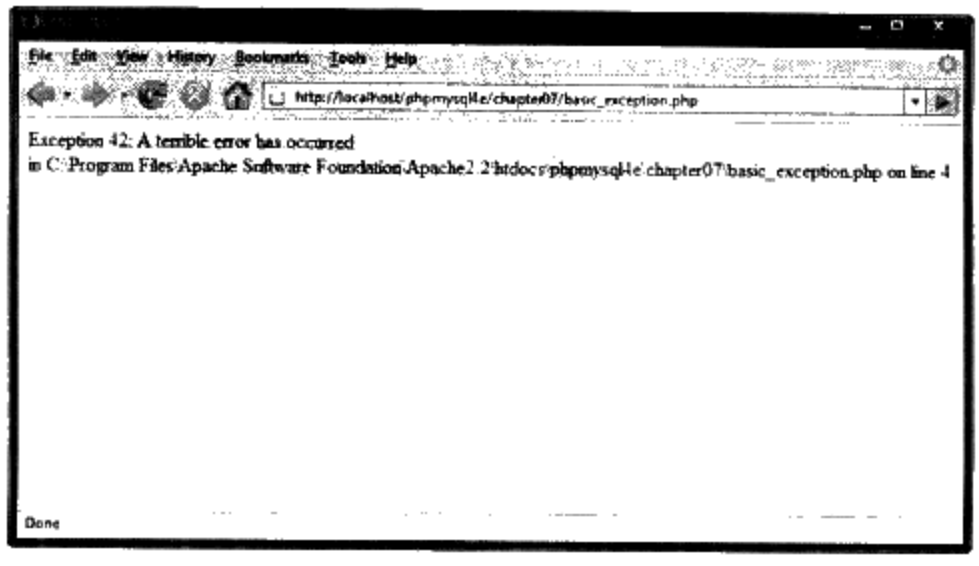

图7-1 catch代码块报告了异常错误消息以及发生错误位置的说明

在以上的示例代码中，可以看到我们生成了一个Exception类的异常。这个内置类具有一些可以在catch代码块中用来报告有用错误消息的方法。

7.2 Exception类

PHP为异常处理提供了内置类——Exception。其构造函数需要两个参数，正如我们在前面讨论的：一个错误消息和一个错误代码。

除了构造函数外，该类还提供了如下所示的内置方法：

- getCode()——返回传递给构造函数的代码。
- getMessage()——返回传递给构造函数的消息。
- getFile()——返回产生异常的代码文件的完整路径。
- getLine()——返回代码文件中产生异常的代码行号。
- getTrace()——返回一个包含了产生异常的代码回退路径的数组。
- getTraceAsString()——返回与getTrace()方向相同的信息，该信息将被格式化成一个字符串。
- _toString()——允许简单地显示一个Exception对象，并且给出以上所有方法可以提供的信息。

在程序清单7-1中，可以看到，我们使用了前4个方法。通过执行以下命令，可以获得相同的异常信息（以及代码回退路径）：

```
echo $e;
```

回退路径显示了在发生异常时所执行的函数。

7.3 用户自定义异常

除了可以实例化并传递Exception基类的实例外，还可以传递任何希望的对象。在大多数情况下，可以扩展Exception类来创建自己的异常类。

我们可以在throw子句中传递任何其他对象。如果在使用特定对象时出现问题，并且希望将其用于调试用途，可以传递其他对象。

然而，在大多数情况下，我们可以扩展Exception基类。PHP手册提供了显示Exception类结构的代码。这段代码如程序清单7-2所示，也可以在http://us.php.net/manual/en/language.oop5.php位置找到。请注意，这并不是真正的代码，它只表示你可能希望继承的代码。

程序清单7-2　Exception类——这是你可能希望继承的代码

```
<?php
class Exception {
    function __construct(string $message=NULL, int $code=0) {
        if (func_num_args()) {
            $this->message = $message;
        }
        $this->code = $code;
        $this->file = __FILE__; // of throw clause
```

```
        $this->line = __LINE__; // of throw clause
        $this->trace = debug_backtrace();
        $this->string = StringFormat($this);
    }

    protected $message = 'Unknown exception '; // exception message
    protected $code = 0; // user defined exception code
    protected $file; // source filename of exception
    protected $line; // source line of exception

    private $trace; // backtrace of exception
    private $string; // internal only!!

    final function getMessage() {
        return $this->message;
    }
    final function getCode() {
        return $this->code;
    }
    final function getFile() {
        return $this->file;
    }
    final function getTrace() {
        return $this->trace;
    }
    final function getTraceAsString() {
        return self::TraceFormat($this);
    }
    function __toString() {
        return $this->string;
    }
    static private function StringFormat(Exception $exception) {
        // ... a function not available in PHP scripts
        // that returns all relevant information as a string
    }
    static private function TraceFormat(Exception $exception) {
        // ... a function not available in PHP scripts
        // that returns the backtrace as a string
    }
}
?>
```

这里给出该类定义的主要原因是希望读者注意到该类的大多数公有方法都是final的：这就意味着不能重载这些方法。我们可以创建自己的Exception子类，但是不能改变这些基本方法的行为。请注意，_toString()函数可以重载，因此我们可以改变异常的显示方式。也可以添加自己的方法。

用户自定义的Exception类示例如程序清单7-3所示。

程序清单7-3　user_defined_exception.php——用户定义Exception类的示例

```
<?php

class myException extends Exception
{
  function __toString()
  {
        return "<table border=\"1\">
        <tr>
        <td><strong>Exception ".$this->getCode()."
        </strong>: ".$this->getMessage()."<br />"."
        in ".$this->getFile()." on line ".$this->getLine()."
        </td>
        </tr>
        </table><br />";
    }
}

try
{
  throw new myException("A terrible error has occurred", 42);
}
catch (myException $m)
{
    echo $m;
}

?>
```

在以上代码中，我们声明了一个新的异常类myException，该类扩展了Exception基类。该类与Exception类之间的差异在于重载了_toString()方法，从而为打印异常提供了一个更好的方法。执行以上代码的输出结果如图7-2所示。

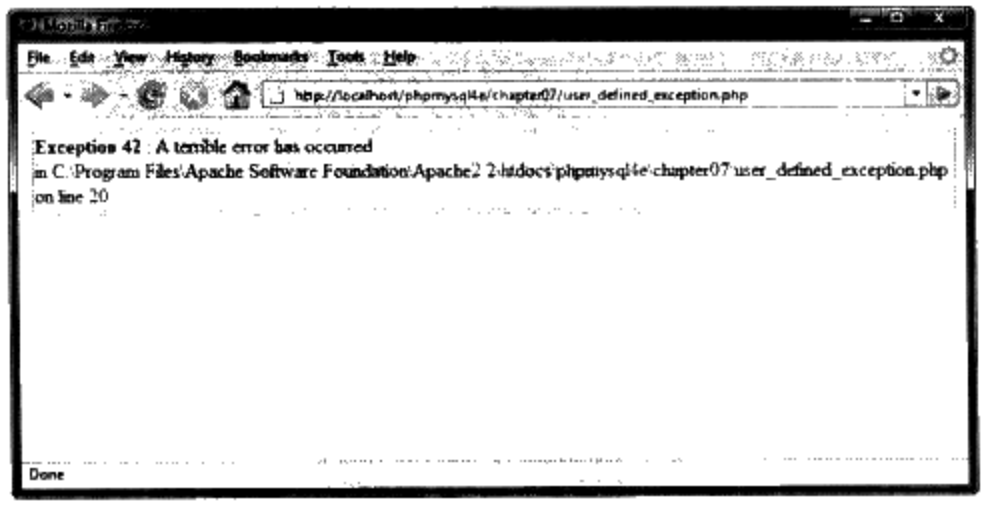

图7-2　myException类为异常提供更好的“打印”格式

这个示例非常简单。在下一节中，我们将介绍创建能够处理不同类型错误的不同异常。

7.4 Bob的汽车零部件商店应用程序的异常

在第2章中，我们介绍了如何以普通文件的格式保存Bob汽车零部件商店的订单数据。我们知道，文件I/O（事实上，任何类型的I/O）是程序经常出现错误的地方。这就使得它成为应用异常处理的合理位置。

回顾初始代码，可以看到，写文件时可能会出现三种情况的错误：文件无法打开、无法获得写锁或者文件无法写入。我们为每一种可能性都创建了一个异常类。这些异常类的代码如程序清单7-4所示。

程序清单7-4 file_exceptions.php——文件I/O相关的异常

```
<?php

class fileOpenException extends Exception
{
  function __toString()
  {
       return "fileOpenException ". $this->getCode()
              . ": ". $this->getMessage()."<br />".' in "
              . $this->getFile(). " on line ". $this->getLine()
              . "<br />";
  }
}

class fileWriteException extends Exception
{
  function __toString()
  {
      return "fileWriteException ". $this->getCode()
             . ': '. $this->getMessage()."<br />".' in "
             . $this->getFile(). " on line ". $this->getLine()
             . "<br />";
  }
}

class fileLockException extends Exception
{
  function __toString()
  {
       return "fileLockException ". $this->getCode()
              . ": ". $this->getMessage()."<br />".' in "
              . $this->getFile(). " on line ". $this->getLine()
              . "<br />";
  }
}


?>
```

Exception类的这些子类并没有执行任何特别的操作。事实上，对于这个应用程序的作用来说，可以让它们成为空的子类或者使用PHP所提供的Exception类。然而，我们为每一个子类提供了_toString()方法，从而可以解释所发生的异常类型。

我们重新编写了第2章的processorder.php文件，在其中使用了异常。该文件的新版本如程序清单7-5所示。

程序清单7-5　processorder.php——Bob汽车零部件商店程序的订单处理脚本，该脚本已经包括了异常处理

```
<?php

  require_once("file_exceptions.php");

  // create short variable names
  $tireqty = $_POST['tireqty'];
  $oilqty = $_POST['oilqty'];
  $sparkqty = $_POST['sparkqty'];
  $address = $_POST['address'];

  $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head>
  <title>Bob's Auto Parts - Order Results</title>
</head>
<body>
<h1>Bob's Auto Parts</h1>
<h2>Order Results</h2>
<?php
$date = date('H:i, jS F');

echo "<p>Order processed at ".$date."</p>";

echo '<p>Your order is as follows: </p>';

$totalqty = 0;
$totalqty = $tireqty + $oilqty + $sparkqty;
echo "Items ordered: ".$totalqty.'<br />';

if( $totalqty == 0) {
  echo "You did not order anything on the previous page!<br />";
} else {
  if ( $tireqty > 0 ) {
    echo $tireqty." tires<br />";
  }
  if ( $oilqty > 0 ) {
    echo $oilqty." bottles of oil<br />";
```

```
    }
    if ( $sparkqty > 0 ) {
      echo $sparkqty.' spark plugs<br />";
    }
}

    $totalamount = 0.00;

    define('TIREPRICE', 100);
    define('OILPRICE', 10);
    define('SPARKPRICE', 4);

    $totalamount = $tireqty * TIREPRICE
                 + $oilqty * OILPRICE
                 + $sparkqty * SPARKPRICE;

    $totalamount=number_format($totalamount, 2, '.', ' ');

    echo "<p>Total of order is ".$totalamount."</p>";
    echo "<p>Address to ship to is ".$address."</p>";

    $outputstring = $date."\t".$tireqty." tires \t".$oilqty." oil\t"
                    .$sparkqty." spark plugs\t\$".$totalamount
                    ."\t". $address."\n";

    // open file for appending
    try
    {
      if (!($fp = @fopen("$DOCUMENT_ROOT/../orders/orders.txt", 'ab')))
          throw new fileOpenException();

    if (!flock($fp, LOCK_EX))
        throw new fileLockException();

    if (!fwrite($fp, $outputstring, strlen($outputstring)))
       throw new fileWriteException();
    flock($fp, LOCK_UN);
    fclose($fp);
    echo "<p>Order written.</p>";
}
catch (fileOpenException $foe)
{
   echo '<p><strong>Orders file could not be opened.
         Please contact our webmaster for help.</strong></p>';
}
catch (Exception $e)
{
```

```
    echo '<p><strong>Your order could not be processed at this time.
          Please try again later.</strong></p>';
}

?>
</body>
</html>
```

可以看到，以上脚本的文件I/O部分被封装在一个try代码块中。通常，良好的编码习惯要求try代码块的代码量较少，并且在代码块的结束处捕获相关异常。这使得异常处理代码更容易编写和维护，因为可以看到所处理的内容。

如果无法打开文件，将抛出一个fileOpenException异常；如果无法锁定该文件，将抛出一个fileLockException异常；而如果无法写这个文件，将抛出一个fileWriteException异常。

分析catch代码块。要说明这一点，我们只给出了两个catch代码块：一个用来处理fileOpen-Exception异常，而另一个用来处理Exception。由于其他异常都是从Exception继承过来的，它们将被第二个catch代码块捕获。catch代码块与每一个instanceof操作符相匹配。这就是为每一个类扩展自己异常类的理由。

一个重要警告：如果异常没有匹配的catch语句块，PHP将报告一个致命错误。

7.5　异常和PHP的其他错误处理机制

除了本章所讨论的异常处理机制，PHP还提供了复杂的错误处理支持，这将在第26章详细介绍。请注意，产生和处理异常的过程并不会影响或禁止这种错误处理机制的运行。

在程序清单7-5中，请注意fopen()函数的调用仍然使用了@错误抑制操作符前缀。如果该函数调用失败，PHP将发出一个警告，根据php.ini中的错误报告设置不同，该警告可能会被报告或者记录。这些设置将在第26章详细介绍，但我们必须知道，无论是否产生一个异常，这个警告仍然会发出。

7.6　进一步学习

由于异常处理对PHP来说是全新的，因此关于该内容并没有更多的介绍。但是，关于异常处理的基本信息还是足够了。Sun提供了一个非常不错的关于异常以及使用异常的原因的介绍（当然，是从Java角度出发而编写的）：

http://java.sun.com/docs/books/tutorial/essential/exceptions/handling.html。

7.7　下一章

本书的下一篇将介绍MySQL。我们将介绍如何创建和操作一个MySQL数据库，学习PHP的有用链接资源，这样我们就可以从Web对数据库进行访问。