第26章 调 试

在本章中，我们将主要讨论如何调试PHP脚本。完成了本书前面所介绍的一些例子或者以前曾经用过PHP，我们已经掌握了一些关于调试的技巧和技术。随着项目更加复杂，调试也变得更加困难。虽然技能提高了，但错误也可能牵涉到更多的文件，或者在多个程序员编写的代码之间相互作用。

在本章中，我们将主要介绍以下内容：

- 编程语法，运行时以及逻辑错误
- 出错信息
- 错误级别
- 触发自定义错误
- 巧妙地处理错误

26.1 编程错误

无论使用什么语言进行编程，通常，都会遇到如下所示的3种错误类型：

- 语法错误
- 运行时错误
- 逻辑错误

在开始讨论检查、处理、避免和解决错误的策略之前，我们先简要地介绍这3种类型。

26.1.1 语法错误

语言有一系列规则，称之为*语法*，语句只有符合语法才是有效的。自然语言（例如，英语）和编程语言（例如，PHP）都是如此。如果语句不符合这些语言规则，就被称为有*语法错误*。通常，语法错误在讨论解释性语言（例如，PHP）的时候，也经常称为*解析错误*；在讨论编译性语言（如C或Java语言）的时候也称为*编译错误*。

如果我们打破英语的语法规则，人们仍然很可能知道我们的意思是什么。但在编程语言中，情况就不是这样。如果脚本不遵守PHP的语法规则——也就是说，如果它含有语法错误——PHP解析器将不能处理其中的一部分或者全部。人们善于从片面或相互冲突的数据中推测出信息，但是计算机却不能。

在许多其他的规则中，PHP语法要求语句以分号结束，字符串包含在引号内，传给函数的参数用逗号分开并且包括在圆括号内。如果不遵守这些规则，PHP脚本就不可能正常工作，并在我们首次运行这些脚本的时候产生出错信息。

PHP的最大的一个优点就是在出错的时候会给出出错信息，而且其出错信息十分有用。通常，一个PHP出错信息将告诉我们出现了什么错误，在哪个文件出错，以及错误在哪一行。

如下所示的就是一个示例的出错信息：

```
Parse error: parse error, unexpected ' ' in
/home/book/public_html/phpmysql4e/chapter26/error.php on line 2
```

该错误由下列脚本产生：

```
<?php
  $date = date(m.d.y );
?>
```

可以看到，我们试图将一个字符串传递给date()函数，但是忘了在字符串开始前添加引号。

通常，这样的简单语法错误最容易发现。但是，我们可能还会产生一个类似的却更难发现的错误——忘记了字符串结束的后引号，例如：

```
<?php
  $date = date('m.d.y);
?>
```

以上代码将产生如下所示的错误：

```
Parse error: parse error, unexpected $end in
/home/book/public_html/phpmysql4e/chapter26/error.php on line 2
```

很明显，脚本代码只有3行，错误不可能真的出现在第4行。当我们打开了什么，却没有将它关闭时出现的错误可能就会像上述错误信息一样。在使用单引号和双引号，以及使用不同形式的括号时，很容易遇到这个问题。

如下所示的脚本将产生类似的语法错误：

```
<?php
  if (true) {
    echo 'error here';
?>
```

如果是多个文件的组合产生的这些错误，那么它们就很难找出来。错误出现在一个大文件里也很难查找。当遇到一个1000行的文件出现了“parse error on line 1001”的错误，恐怕要花一天才能找出错误来。作为一个提醒，我们应该尽量编写模块化的代码。

然而一般来说，语法错误是最容易找出的错误类型。如果出现语法错误并且尝试执行一个代码块，PHP会给出错信息并告诉我们在哪里查找错误。

26.1.2　运行时错误

运行时错误更难检测到，也更难修改。一个脚本可能包含语法错误，也可能没有语法错误。如果脚本包含一个语法错误，解析器能够检测到它。而运行时错误则不会只由脚本的内容导致。它们可能出现在脚本交互的过程中或者其他的事件或条件下。

如下所示的语句：

```
require ('filename.php );
```

这是一个完全有效的PHP语句，它没有任何语法错误。

然而，上述语句却可能产生运行时错误。如果执行该语句时，filename.php并不存在或者运行脚本的用户对该文件没有“读”权限，就可能出现如下所示的出错信息：

```
Fatal error: main() [function.require]: Failed opening required 'filename.php'
(include_path='.:/usr/local/lib/php') in
/home/book/public_html/phpmysql4e/chapter26/error.php on line 1
```

尽管代码没有任何错误，但是由于运行代码所依赖的文件在代码运行的不同时刻可能存在，也可能不存在，所以就可能产生运行时的错误。

如下所示的3个语句都是有效的PHP语句。不幸的是，当3个语句结合到一起的时候，它们是不可能得到运行结果的——因为出现了被0除的现象：

```
$i = 10;
$j = 0;
$k = $i/$j;
```

该代码将产生如下所示的警告：

```
Warning: Division by zero in
/home/book/public_html/phpmysql4e/chapter26/div0.php on line 3
```

这种错误很容易修改。没有人会故意编写被0除的代码，但是忽略检查用户输入却经常会导致这种错误类型。

如下所示的代码有时候将产生相同的错误，但是可能会很难隔离和修改，因为它只发生在某些时候：

```
$i = 10;
$k = $i/$_REQUEST['input'];
```

这是在测试代码时最经常看到的运行时错误之一。

通常，以下操作易导致运行时的错误：

- 调用不存在的函数
- 读写文件
- 与MySQL或其他数据库的交互
- 连接到网络服务
- 检查输入数据失败

在接下来的内容中，我们将简单介绍以上每一种容易导致错误的原因。

1. 调用不存在的函数

在偶然的情况下，调用不存在的函数的操作比较容易发生。此外，内置函数也经常会出现命名不一致的情况。为什么strip_tags()函数名称中间有一个下划线，但是stripslashes()却没有呢?

不但在当前脚本里容易调用自己编写的、不存在的函数，而且在其他地方也容易这样调用。如果代码包含了对一个不存在函数的调用，例如：

```
nonexistent_function();
```

或

```
mispeled_function();
```

将出现如下所示的出错信息：

```
Fatal error: Call to undefined function: nonexistent_function()
in /home/book/public_html/phpmysql4e/chapter26/error.php on line 1
```

相似地，如果调用一个存在的函数，但是使用的参数个数不对，也将遇到一个警告。

函数strstr()要求输入两个字符串：haystack用来搜索，needle用来查找。如果我们按如下方式调用它：

```
strstr();
```

将得到如下所示警告：

```
Warning: Wrong parameter count for strstr() in
/home/book/public_html/phpmysql4e/chapter26/error.php on line 1
```

如下所示的语句同样是错误的：

```
<?php
  if($var == 4) {
    strstr();
  }
?>
```

除了极少的情况（也就是，变量$var值为4）外，对strstr()的调用将不会发生，警告也不会出现。PHP解释器不会将时间花费在解析那些不必要在当前的执行脚本中执行的代码。我们必须确认进行了仔细的测试。

不正确的函数调用很容易发生，但是因为导致的出错信息可以确切地识别出错误行和导致错误的函数，所以它们都比较容易纠正。只有在测试过程非常糟糕并且没有测试所有条件执行的代码时，才很难查出错误。进行测试的时候，测试的目标之一是代码的每一行都要执行一次。另一个目标是测试所有的临界条件和各种类型的输入。

2. 读写文件

尽管在一定程度上说，程序的使用周期里什么错误都可能出现，但是一些错误还是比其他错误更容易出现。由于访问文件错误是经常出现的，因此非常有必要巧妙地处理它们。硬盘驱动器出错或写满，人为操作错误导致目录权限改变等。

通常，与fopen()函数一样，可能偶尔失败的函数都有一个返回值指示出现的错误。对于fopen()函数来说，返回值为false表示失败。

对提供失败指示的函数，我们需要小心检查每次调用的返回值，并根据返回值进行接下来的操作。

3. 与MySQL或其他数据库的交互

连接和使用MySQL可能产生许多错误。函数mysqli_connect()就可能至少产生下列一些错误：

- **Warning:** mysqli_connect() [function.mysqli-connect]: Can't connect to MySQL server on 'localhost'(10061)（无法连接到“localhost”

上的MySQL服务器）

- **Warning:** mysqli_connect() [function.mysqli-connect]:Unknown MySQL Server Host 'hostname'(11001)（未知的MySQL服务器主机“hostname”）
- **Warning:** mysqli_connect() [function.mysqli-connect]:Access denied for user: 'username'@ 'localhost' (Using password: YES)（用户username@localhost的访问被拒绝，使用密码：YES）

mysqli_connect()函数在出现错误的时候将返回值false。这就意味着用户可以很容易地检测和处理这类常见的错误。

如果希望不终止脚本的正常执行而解决这些错误，脚本将继续与数据库交互。没有一个有效的MySQL连接就试图运行查询和得到结果，将导致访问者看到满屏幕的有失专业水准的错误信息。

许多其他常用的、与MySQL相关的PHP函数，例如mysqli_query()，也将返回false来表示函数调用出现错误。

如果出现错误，可以通过调用函数mysqli_error()来获得错误信息文本，或者调用函数mysqli_errno()获得函数调用的错误代码。如果MySQL函数的调用没有出现错误，mysqli_error()将返回空字符串，mysqli_errno()将返回0。

例如，假设我们已经连接到服务器，并且选择了一个要使用的数据库，如下所示的代码段：

```
$result = mysqli_query($db, 'select * from does_not_exist');
echo mysqli_errno($db);
echo '<br />';
echo mysqli_error($db);
```

的可能输出结果是：

```
1146
Table 'dbname.does_not_exist'doesn't exist
```

请注意，这些函数的输出是指最后执行的MySQL函数（而不是mysqli_error()或mysqli_errno()的结果）。如需要知道命令的结果，需确认在运行其他命令之前检查它。

就像文件交互错误一样，数据库交互造成的错误也会时常发生。即使在完成某个服务的开发和测试之后，也会偶尔发现MySQL后台程序（mysqld）崩溃或不能正常连接问题。如果服务器运行于另一台物理机器，那么它还要依赖于另一套硬件和软件组件，而这些组件可能出错——Web服务器和数据库服务器之间的网络连接、网卡、路由器等。

请记住，在试图使用这些结果之前需检查一下数据库请求是否成功。连接数据库失败之后试图检索结果是没有意义的，而检索失败后试图提取和处理检索的结果也是没有意义的。

在这里，值得注意的是，检索失败与仅仅是没有返回任何数据，也没有对任何记录行产生影响的检索是有区别的。

一个包含SQL语法错误的SQL检索，或者对不存在的数据库、表和列的访问也可能会导致错误。如下所示的查询：

```
select * from does_not_exist;
```

也可能会失败，因为表名称并不存在，这样将产生一个错误号和错误消息，可以通过mysqli_errno()函数或mysqli_error()函数获得。

通常，一个SQL查询如果只是在语法上是合法的，而且只对存在的数据库、表和列进行访问，它是不会产生错误的。但是，如果它检索一个空的数据库表或检索一个不存在的数据，它将返回空值。假设已经成功地连接了数据库，并且存在表t1和列c1，如下所示的检索：

```
select * from t1 where c1 = 'not in database';
```

该代码将成功执行但是不返回任何结果。

在使用检索结果之前，要检查检索是否失败和返回空结果。

4. 连接到网络服务

尽管系统中的设备和其他程序偶尔可能出错，但如果质量不是很差，它们应该很少出错。当在网络上连接其他机器和这些机器上的软件时，应该意识到系统的某些部分会经常出错。从一台机器连接到另一机器依赖于许多我们无法控制的设备和服务。

尽管操作可能重复，但我们还是要仔细检查试图与网络服务连接的函数的返回值。

如下所示的函数调用：

```
$sp = fsockopen('localhost', 5000);
```

如果它无法正确连接localhost机器的端口5000，该函数将给出一个警告信息。但是该警告信息将以默认的格式给出，同时不会为代码提供能够巧妙处理该警告的选择。

将以上函数调用重写为：

```
$sp = @fsockopen ('localhost', 5000, &$errorno, &$errorstr);
if(!$sp) {
  echo "ERROR: ".$errorno.": ".$errorstr;
}
```

以上代码将抑制内置错误消息，并且通过函数的返回值检查函数调用是否出现错误，同时使用自己的代码来处理错误消息。运行以上代码后，它将显示一个可以帮助解决问题的错误消息。在这个例子中，将产生如下所示的输出：

```
ERROR: 10035: A non-blocking socket operation could not be completed immediately.
```

与语法错误相比较，运行时错误更难消除，因为在代码首次运行的时候解析器并不能指示出错误所在。因为运行时错误通常出现在一连串的事件响应中，因此很难检查到，也很难解决。解析器并不能自动指出哪一行会产生错误。测试的时候需要提供一个能够产生该类错误的情形。

解决运行时错误需要一定程度的预见性；不仅要检查所有可能出现的不同类型的错误，然后采取适当的措施；也需要仔细检测和模拟可能出现的每一类运行时错误。

这并不表示必须模拟每一个可能出现的错误。例如，MySQL可以提供大约200种不同的错误编号和错误信息，我们需要的是在每一个可能导致错误的每个函数调用中模拟函数错误，每种不同类型的错误将由特定的代码来解决。

5. 检查输入数据失败

通常，我们会对用户可能输入的数据进行一些假设。如果输入的数据不是原先所预料的数据类型，它就可能会导致错误，该错误或者是运行时错误，或者是逻辑错误（将在下一节里详

细介绍)。

典型的运行时错误的例子是在处理用户输入数据的时候忘记对该数据执行addslashes()函数。这样，如果有一个叫O'Grady的用户，名字中包含一个撇号，当在数据库插入语句中通过单引号使用这个输入，那么数据库函数将产生一个错误。

在接下来的一节中，我们将介绍由于对用户输入数据不同假设所导致的错误。

26.1.3 逻辑错误

逻辑错误是最难发现和清除的错误类型。这类错误类型的代码是完全正确的，而且它也是按照正确的程序逻辑执行的，但是这种工作却不是编程人员所期望的。

逻辑错误可能出现于一个简单的键盘输入错误，例如：

```
for ( $i = 0; $i < 10; $i++);
{
  echo 'doing something<br />';
}
```

以上这一小段代码是完全合法的。它遵循PHP语法。也不依赖任何外部服务，因此也不可能出现运行时错误。但是如果不仔细检查，它的运行结果可能就不是我们所期望的或程序员打算要的。

粗看起来，它好像是用for语句循环10次，每次执行都回显“doing something”。在第一行for语句结尾后的分号表示循环对下面的语句行没有任何影响。for语句将循环10次，没有任何输出，然后echo语句只执行一次。

因为这段代码完全合法，但是却效率低下，这样的代码将导致无法预见的结果，解析器也不会有任何警告或出错信息。计算机擅长做某些事情，但是它没有任何常识和智能。计算机只能做我们让它所做的事情。必须确认它所做的就是我们希望的。

逻辑错误不是由任何一种代码失败所导致的，而是程序员没有正确地编写代码以命令计算机做人们所期望的事情。结果，错误就不能自动检查到。计算机也不能告诉编程者出现了错误，更不会给他一段代码告诉他问题出在什么地方。逻辑错误只有通过适当的测试才能检测到。

如上所述的微小逻辑错误我们容易犯，也容易纠正，因为在首次运行源代码看到的输出结果并不是我们所期望的输出。但是，大多数逻辑错误的隐蔽性更强一些。

通常，麻烦的逻辑错误产生于开发人员的错误假设。在第25章“在大型项目中使用PHP和MySQL”中，我们已经建议请其他开发人员来审查代码并提出他们的测试方案，并请目标用户而不是开发人员作为测试人员进行测试。用户只输入某几种类型数据是很容易的，但是如果自己做测试容易漏掉一些没有检测到的情况。

举个例子，在某商务网站有一个Order Quantity文本框。开发人员是不是会假设用户只输入正数呢？如果一个用户输入-10，软件是否会将10倍于该物品价格的钱退还到用户信用卡里呢？

假设有一个文本输入框用来输入美元总数。它允许用户输入的总数后面带美元符号吗？允许满1000就用逗号分开吗？一些错误可以在客户端（例如，使用JavaScript）检查出来而无须由服务器来检查。

如果正在将信息传送到另一页面，可能有一些字符在一个URL中有特殊重要的意义，例如传输的字符串中包含空格。你遇到过这样的情况吗？

逻辑错误有许多。没有自动检测的方法。唯一的办法是，首先要消除暗含在脚本中的假设；其次是彻底地测试每种可能有效或无效的输入，确认这些输入都得到了期望结果。

26.2 使用变量帮助调试

当项目变得更复杂的时候，使用一些实用程序代码帮助确定错误的原因就很有意义了。程序清单26-1所示的代码就可能是很有用的。该代码回显传到页面的变量的值。

程序清单26-1　dump_variables.php——该代码可以包含在页面中，它可以将特定变量的内容打印出来，有助于调试

```
<?php
  // these lines format the output as HTML comments
  // and call dump_array repeatedly

  echo "\n<!-- BEGIN VARIABLE DUMP -->\n\n";

  echo "<!-- BEGIN GET VARS -->\n";
  echo "<!-- ".dump_array($_GET)." -->\n";

  echo "<!-- BEGIN POST VARS -->\n";
  echo "<!-- ".dump_array($_POST)." -->\n";

  echo "<!-- BEGIN SESSION VARS -->\n";
  echo "<!-- ".dump_array($_SESSION)." -->\n";

  echo "<!-- BEGIN COOKIE VARS -->\n";
  echo "<!-- ".dump_array($_COOKIE)." -->\n";

  echo "\n<!-- END VARIABLE DUMP -->\n";

// dump_array() takes one array as a parameter
// It iterates through that array, creating a single
// line string to represent the array as a set

function dump_array($array) {

  if(is_array($array)) {

    $size = count($array);
    $string = "";
    if($size) {

      $count = 0;
      $string .= " ";
```

```
        // add each element's key and value to the string
        foreach($array as $var => $value) {

          $string .= $var." - ".$value;
          if($count++ < ($size-1)) {
            $string .= ", ";
          }
        }
        $string .= " )";
      }
      return $string;
    } else {
      // if it is not an array, just return it
      return $array;
    }
}
?>
```

该代码将页面所接收到的4个数组变量的元素内容通过循环的方式打印出来。如果某页面由GET变量、POST变量、cookie或会话变量来调用，就将打印出这些变量的内容。

在这里，我们已经将其输出加入到HTML注释，这样可以看到它们，而且不会影响浏览器对可见页面元素的显示方式。这是创建调试信息的一种好方法。就像程序清单26-1所示，在注释中隐藏调试信息将允许在最后一分钟以前仍然保留调试代码。我们使用dump_array()函数作为print_r()函数的封装器。dump_array()函数只是过滤了任何HTML的注释结束字符。

确切的输出取决于传给页面的变量，但是如果将以上代码加入到程序清单23-4中（第23章“在PHP中使用会话控制”中的一个身份验证的例子），它将下面这些行添加到脚本生成的HTML中：

```
<!-- BEGIN VARIABLE DUMP -->

<!-- BEGIN GET VARS -->
<!-- Array
(
)
  -->
<!-- BEGIN POST VARS -->
<!-- Array
(
    [userid] => testuser
    [password] => password
)
  -->
<!-- BEGIN SESSION VARS -->
<!-- Array
(
)
```

```
    -->
<!-- BEGIN COOKIE VARS -->
<!    Array
(
    [PHPSESSID] => b2b5f56fad986dd73a133f470f3c1865
)

    >

<!-- END VARIABLE DUMP -->
```

可以看到，以上脚本显示了POST变量的内容——userid和password，该变量是前一页面中的登录表单发送过来的。此外，它也显示了会话变量，该变量用来隐藏用户的名字：valid_user。正如第23章中所述，PHP用一个cookie将会话与特定的用户相关联。我们的脚本显示了伪随机码：PHPSESSID，该码将保存在cookie里，从而识别特定的用户。

26.3　错误报告级别

PHP允许设置对错误的重视程度。用户可以修改以确定哪类事件将产生错误消息。在默认情况下，PHP将报告所有除了通知之外的错误。

错误报告级别是通过一些预定义的常量来设置的，如表26-1所示。

表26-1　错误报告常量

值	名　称	意　义
1	E_ERROR	报告在运行时的致命错误
2	E_WARNING	报告运行时的非致命错误
4	E_PARSE	报告解析错误
8	E_NOTICE	报告通告、注意、表示所做的事情可能是错误的
16	E_CORE_ERROR	报告PHP引擎启动失败
32	E_CORE_WARNING	报告PHP引擎启动时非致命错误
64	E_COMPILE_ERROR	报告编译错误
128	E_COMPILE_WARNING	报告编译时出现的非致命错误
256	E_USER_ERROR	报告用户触发的错误
512	E_USER_WARNING	报告用户触发的警告
1024	E_USER_NOTICE	报告用户触发的通告
6143	E_ALL	报告所有的错误和警告（不包括E_STRICT所报告的）
2048	E_STRICT	报告不赞成的用法和不推荐的行为；不包括在E_ALL中，但对代码重构很有帮助。为交互方式提供修改意见
4096	E_RECOVERABLE_ERROR	报告所有可以捕获的非致命错误

每个常量都表示一种错误类型，该错误可以被报告也可以被忽略。例如，如果指定错误报告级别为E_ERROR，那么只有出现致命错误的时候才报告。这些常量可以用二进制算法将其结合起来，产生不同的错误级别。

默认的错误级别是报告除了通知之外的所有错误，由如下所示语句指定：

```
E_ALL & ~E_NOTICE
```

以上表达式由前面说明的两个常量组成，它们通过位算法操作符“&”结合。“&”操作符表示位操作符AND，而符号“~”表示操作符NOT。上述表达式可读作E_ALL AND NOT E_NOTICE。

E_ALL本身就是除E_STRICT以外的所有错误类型的有效结合。它可以用其他错误级别通过位操作符“|”相“或”得到，例如：

```
E_ERROR | E_WARNING | E_PARSE | E_NOTICE | E_CORE_ERROR | E_CORE_WARNING |
E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING |
E_USER_NOTICE
```

类似地，默认的错误报告级别也可通过除了“E_NOTICE”之外的所有错误级别相“或”而得到：

```
E_ERROR | E_WARNING | E_PARSE | E_CORE_ERROR | E_CORE_WARNING | E_COMPILE_ERROR |
E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE
```

26.4 改变错误报告设置

可以通过php.ini文件或在每个脚本中进行错误报告设置。

在默认情况下，要改变所有脚本的错误报告，可以修改php.ini文件里的下面4行。

```
error_reporting =        E_ALL & ~E_NOTICE
display_errors =         On
log_errors =             Off
track_errors =           Off
```

默认的全局设置如下：

- 报告除了通知之外的所有错误
- 以HTML形式将出错信息输出到标准输出接口
- 不将错误信息作为日志记录到磁盘
- 不跟踪错误，将错误保存在变量$php_errormsg中

最可能要改变的是将错误报告级别提高到E_ALL|E_STRICT。这将导致报告脚本中可能出现的所有通知，这些通知可能指示一个错误，也可能是因为程序员想利用PHP弱类型的特性，还可能因为自动将变量初始化为0这样的事实。

在调试过程中，我们会发现提高error_reporting的级别是有意义的。在产品代码中，如果为自己提供了有用的错误信息，在将error_reporting的级别提高的同时，需将display_errors选项关闭，并将log_errors选项开启，这样会使代码看起来更加专业一些。如果出现错误，可以在日志里看到错误的详细信息。

将track_errors选项打开可以帮助解决用户自己代码中的错误，而不是让 PHP提供其默认的功能。尽管 PHP提供了实用的出错信息，但当出错的时候，它的默认行为却非常糟糕。

在默认情况下，当脚本出现一个致命错误的时候，PHP将输出如下所示的信息：

```
<br />
<b> Error Type </b>: error message in <b> path/file.php</b>
```

```
on line <b> lineNumber</b><br>
```

并且停止执行脚本。对于非致命错误，也会输出同样的信息，但是脚本正常执行。

该HTML的输出将直接发送给错误的标准输出，但是看起来很乏味。这样的出错信息不可能与网站其他部分的外观相协调。如果页面的整个内容显示在一个表格里的话，它也可能导致Netscape用户根本看不到输出。这是因为打开了的HTML没有关闭表格元素，例如：

```
<table>
<tr><td>
<br>
<b> Error Type </b>: error message in <b> path/file.php</b>
on line <b> lineNumber </b><br>
```

以上HTML代码将在某些浏览器中显示为一个空屏。

我们没有必要保留PHP的默认错误处理行为，甚至对所有的文件使用同样的设置。要改变当前脚本的错误报告级别，可以调用error_reporting()函数。

传递一个错误报告常量或几个错误报告常量的结合，与php.ini中类似的指令一样，可以用来设置错误级别。error_reporting()函数将返回前一个错误级别。该函数的常见使用方式如下所示：

```
// turn off error reporting
$old_level = error_reporting(0);
// here, put code that will generate warnings
// turn error reporting back on
error_reporting($old_level);
```

这小段代码将关闭错误报告，允许我们执行一些可能产生警告的代码，而我们并不希望看到这些警告。

永久关闭出错报告是非常糟糕的想法，因为这样很难发现错误并纠正它。

26.5　触发自定义错误

函数trigger_error()可以用来触发错误。用这种方式创建的错误可以像PHP的常规错误那样处理它。

该函数要求给出出错信息，此外，错误类型是该函数的可选参数。错误类型应该是E_USER_ERROR、E_USER_WARNING或E_USER_NOTICE中的一种。如果没有指定这些类型，默认值是E_USER_NOTICE。

如下所示的例子说明了如何使用trigger_error()函数：

```
trigger_error( This computer will self destruct in 15 seconds', E_USER_WARNING);
```

26.6　巧妙地处理错误

对于有C++或Java编程经历的人来说，可能会很习惯使用异常处理。异常处理允许函数发出信号通知错误出现，并将错误处理交给一个专门的异常处理程序。异常机制是在大型项目中

处理错误的好方法。关于异常处理，我们已经在第7章“错误和异常处理”中详细介绍，这里就不再重复。

你已经了解如何触发自己的错误。你也可以提供错误处理程序来捕获错误。

当用户级别的错误、警告和通知发生时，set_error_handler()函数可以提供一个可供调用的函数，可以将用作错误处理程序的函数名称作为set_error_handler()的参数。

错误处理函数必须带有两个参数：错误类型和错误信息。基于这两个参数变量，函数可以决定如何解决这个错误。其中，错误类型应该是已经定义好的错误类型常量。错误信息则是一个描述性字符串。

如下所示的是set_error_handler()函数的调用：

```
set_error_handler('my_error_handler');
```

以上调用将告诉PHP调用一个名为my_error_handler()的函数来进行错误处理操作，此外，还要提供该函数的具体实现方法。该函数原型如下所示：

```
My_error_handler(int error_type, string error_msg
                 [, string errfile [, int errline [, array errcontext]]]))
```

但是，该函数的具体实现可以由自己来设定。

传递给处理程序函数的参数如下：

- 错误类型
- 错误消息
- 发生错误的文件
- 发生错误的行号
- 符号表——也就是，发生错误时所有变量及其值的集合

该函数的逻辑行为可能包括：

- 显示系统提供的错误信息
- 将信息存储到日志文件
- 将错误以电子邮件形式发送到一个邮箱
- 调用一个exit，中断脚本执行

程序清单26-2所示的脚本声明了一个错误处理程序，并调用函数set_error_handler()设置它，然后生成了一些错误。

程序清单26-2 handle.php——该脚本声明和定制了一个错误处理程序并生成不同的错误

```
<?php
  // The error handler function
  function myErrorHandler ($errno, $errstr, $errfile, $errline) {
    echo "<br /><table bgcolor=\"#cccccc\"><tr><td>
          <p><strong>ERROR:</strong> ".$errstr."</p>
          <p>Please try again, or contact us and tell us
          that the error occurred in line '.$errline.' of
          file '.$errfile."</p>';
```

```
    if ((Serrno == E_USER_ERROR) || ($errno == E_ERROR)) {
      echo "<p>This error was fata', program ending</p>
            </td></tr></table> ;

      //close open resources, include page footer, etc
      exit;
    }
    echo '</td></tr></table>';
  }
  // Set the error handler
  set_error_handler( myErrorHandler );

  //trigger different levels of error
  trigger_error( Trigger function called , E_USER_NOTICE);
  fopen('nofile', 'r');
  trigger_error('This computer is beige , E_USER_WARNING);
  include ( nofile');
  trigger_error('This computer will self destruct in 15 seconds', E_USER_ERROR);
?>
```

以上脚本的输出结果如图26-1所示。

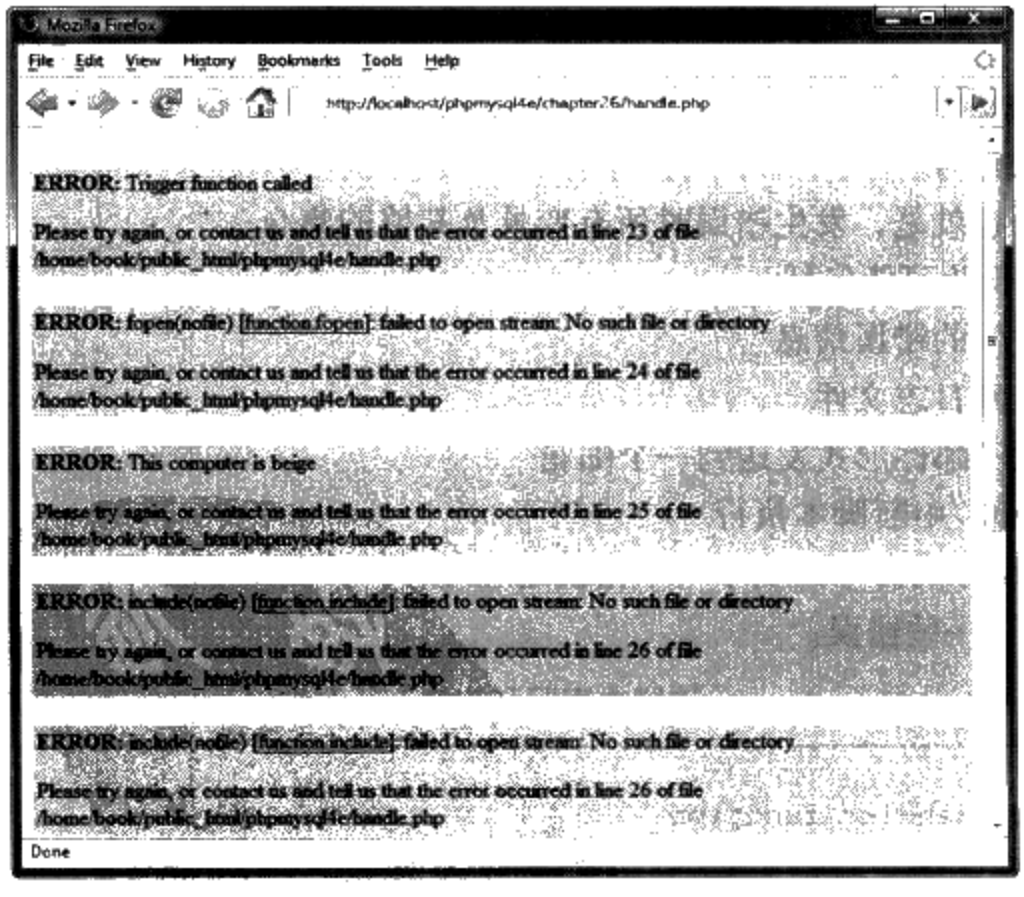

图26-1　如果使用自己的错误处理程序，可以设置比PHP更友好的错误信息

以上自定义的错误处理程序的功能并不比默认的错误处理程序强很多。因为这段代码是由我们自己实现的，所以可以让它做任何事情。通过它可以在出错的时候选择性地告诉访问者一些有用的信息，以及如何显示该信息，这样就可以与网站其他部分的内容相协调。更重要的是，

我们可以灵活地决定发生什么事情：脚本是否应该继续执行？信息是否应该记录到日志或将它显示出来吗？要对技术支持发出警告吗？

需要注意的一点是，自定义错误处理函数将无法处理所有的错误类型。有些错误，例如，解析错误和致命的运行时错误，还是会触发默认的错误处理函数。如果需要考虑到这种情况，需确认在将参数传递给能够产生致命错误，或触发自定义的E_USER_ERROR级别错误之前，仔细检查这些参数。

这里再介绍一个有用的新特征：如果错误处理程序返回了false值，PHP的内置错误处理程序将被调用。这样，可以处理E_USER_*错误，并由内置处理程序处理常见错误。

26.7 下一章

在第27章“建立用户身份验证机制和个性化设置”中，我们将开始本书的第一个项目。在这个项目中，我们将讨论如何识别再次访问网站的用户，以及如何对网站内容进行适当的调整。