# 第 8 章　错误、异常处理与调试

在程序开发中，难免会因为某种原因而产生错误。如何去避免、调试、修复错误并对程序可能发生的异常进行处理是一个程序员必备的能力。PHP 提供了良好的错误提示，在进行程序调试时可根据提示信息对错误进行排除。本章主要内容如下。

- PHP 错误。
- 异常处理。
- 程序调试。

通过对本章的学习，读者将对 PHP 中出现的错误有一个新的认识，并能够正确地理解并避免这些错误，避免程序可能出现的异常，出现错误时能够根据 PHP 的提示信息排除错误。

## 8.1　PHP 错误类型

在 PHP 程序开发中，通常会出现以下 5 种错误。

- 语法错误：在程序中使用了错误的语法而导致的错误。
- 语义错误：在程序中正确地使用了 PHP 的语法，但是没有任何意义，程序达不到预想的效果。
- 逻辑错误：在程序中使用的逻辑与实际上需要的逻辑不符。
- 注释错误：在程序中写的注释与该程序代码的意义不符。
- 运行错误：由于运行环境等原因而导致的错误。

在这几种错误中，除最后一种是由于 PHP 所运行的环境原因等造成的以外，前面的 4 种均是由程序开发人员造成的，因而这 4 种错误应该在程序开发中尽量避免。

### 8.1.1　语法错误

每一种语言都有属于自己的语法。在程序开发中如果使用了错误的语法，就会导致一个语法错误。

【示例 8-1】语法错误的示例。代码如下所示。

```
<?php
$num_1 = 101;
$num_2 = 202;
$num_3 = $num_1 -/ $num_2;                    //使用了错误的语法
```

```
echo $num_3;
?>
```

分析：在上述程序中，本来是将变量 num_1 减去变量 num_2 的差赋给变量 num_3，但是因为第 4 行输入错误而产生了一个错误，PHP 在解析这条语句时认为是错误的，便会抛出如下错误信息。

```
Parse error: syntax error, unexpected '/' in D:\xampplite\htdocs\book\source\8\8.1.php on line 4
```

该错误为解析错误，具体为脚本文件第 4 行出现了不可预知的“/”符号而产生的语法错误。读者在看到该错误信息时，可直接检查该行程序，一般情况下，错误就发生在该行。

### 8.1.2　语义错误

语义错误是在使用了正确语法的基础上，使用了错误的格式而导致的。

【示例 8-2】语义错误的示例。代码如下所示。

```
<?php
$str_1 = "Hello ";
$str_2 = "World.";
$str_3 = $str_1 + $str_2;                    //使用了错误的字符串连接符
echo $str_3;
?>
```

分析：在上述程序中，程序本意是想将两个字符串连接起来，但是程序第 4 行却错误地使用了“+”作为字符连接符。因为 PHP 能够自动进行隐式变量类型转换，PHP 在解析以上代码时认为它是符合 PHP 的语法的，并不会提示出错。

### 8.1.3　逻辑错误

逻辑错误对于 PHP 来讲不是错误，因为语法、语义上没有任何问题。但是因为程序代码存在着逻辑着问题，进而导致程序得不到所期望的结果。

【示例 8-3】逻辑错误的示例。代码如下所示。

```
<?php
$score = 80;
if($score < 80){                             //判断分数
    echo "良好.";
}else{
    echo "中等.";
}
?>
```

分析：在上述程序中，程序本身从语法上和语义上讲都没有任何问题，并且能够得到结果。但是程序逻辑有问题，本来应该是 80 分及其以上为良好，以下为中等，但是程序的结果却是 80 分为中等，这就产生了逻辑错误。这种错误在开发时应尽量避免。

### 8.1.4 注释错误

注释对于程序来讲是必不可少的。因为在分布式开发中，随时都有可能去读其他程序员的代码，如果没有注释，将会花费大量的时间读懂别人的代码。另外对于后期的维护也是相当困难的。对于注释错误，比没有更加可怕，因为开发人员往往会只看注释不会再花时间去看代码。

【示例 8-4】重写【示例 8-3】，构成了一个注释错误。代码如下所示。

```
<?php
$score = 80;
if($score >= 80){                                    //大于 80 分的为良好
    echo "良好.";
}else{
    echo "中等.";
}
?>
```

分析：在上述程序中，显示注释和程序本身的逻辑不统一，程序的逻辑是大于等于 80 分的为良好，而注释却只是大于 80 分为良好。虽然注释错误对于程序本没有任何影响，但是却影响到以后对代码的维护与修改。

### 8.1.5 运行错误

运行错误与程序代码无关，它是由脚本运行的环境等因素造成的。比如在 Linux 系统中，文件权限不对等而引发的错误。

【示例 8-5】因文件不存在而引发的运行错误。代码如下所示。

```
<?php
$filename = "something.txt";                         //定义文件位置
$handle = fopen($filename, "r");                     //打开文件
$contents = fread($handle, filesize ($filename));    //读取文件内容
print $contents;                                     //输出文件内容
fclose($handle);                                     //关闭文件
?>
```

分析：在上述程序中，打开一个文本文件，读取其内容并将内容输出。但是由于当前工作目录下没有指定的文件，因而引发一个运行错误。其结果如下所示。

```
Warning: fopen(something.txt) [function.fopen]: failed to open stream: No such file or directory in D:\xampplite\htdocs\book\source\8\8.5.php on line 3
Warning: filesize() [function.filesize]: stat failed for something.txt in D:\xampplite\htdocs\book\source\8\8.5.php on line 4
Warning: fread(): supplied argument is not a valid stream resource in D:\xampplite\htdocs\book\source\8\8.5.php on line 4
Warning: fclose(): supplied argument is not a valid stream resource in D:\xampplite\htdocs\book\source\8\8.5.php on line 6
```

## 8.2　错误处理

不管是程序引发的错误，还是环境因素引发的错误，默认情况下，PHP 都会给出提示信息。这些提示信息包含有服务器的运行环境信息。在实际的 Web 环境中，将这些信息显示出来，必然给服务器带来安全隐患。因此，必须对可能出现的错误进行相应的处理。

### 8.2.1　错误级别

PHP 中的错误是通过一个错误级别来进行划分的。从最基本的通告到最严重的错误，错误级别标识着所产生的错误的严重性。错误级别包含以下几种。

- E_ERROR：这是一个严重错误，不可恢复，如位置异常、内存不足等。
- E_WARNING：警告，最一般的错误，如函数的参数错误等。
- E_PARSE：解析错误，在解析 PHP 文件时产生，并强制 PHP 在执行前退出。
- E_STRICT：这个错误级别是唯一不包含在 E_ALL 常量中的，主要是为了便于兼容 PHP 的低版本。
- E_NOTICE：通告表示可能在操作一些未知的变量等。在开发时可开启通告，以保证程序是“通告安全”的；而在正式系统中，应关闭通告。
- E_CORE_ERROE：这个内部错误是由于 PHP 加载扩展失败而导致的，并且会导致 PHP 停止运行并退出。
- E_COMPILE_ERROR：编译错误是在编译时发生，这个错误将导致 PHP 运行退出。
- E_COMPILE_WARNING：编译警告用于告诉用户一些不推荐的语法信息。
- E_USER_ERROR：用户定义的错误将导致 PHP 退出执行。它不是来自 PHP 本身，而是来自脚本文件中。
- E_USER_WARNING：脚本使用它来通知一个执行失败，同时 PHP 也会用 E_WARNING 通知。
- E_USER_NOTICE：用户定义的通告用于在脚本中表示可能存在的错误。

### 8.2.2　php.ini 对错误处理的设置

在前面的章节中曾讲到了 php.ini 文件，PHP 的环境几乎都是在这个文件中进行设置。这里有两项关于错误处理的设置：一个是 display_errors，一个是 error_reporting。前一变量是用来告诉 PHP 是否显示错误，它的默认值为 Off，即不显示错误信息，如果设置为 true，将显示错误信息。后一变量是告知 PHP 如何显示提示信息，默认值为 E_ALL&~E_NOTICE，即显示除注意信息外的所有提示信息。

对于在开发环境，为了便于程序的调试，可将 php.ini 的相应项改成如下值。

```
display_errors = On
error_reporting = E_ALL & ~E_NOTICE
```

对于实际的 Web 环境，可将相应项设为如下值。

```
display_errors = Off
error_reporting = E_ALL
```

修改以后，重启 apache 服务器即可。除了可在 php.ini 文件进行设置以外，还可以在程序开头使用 error_reporting()函数进行设置。该函数的参数与 php.ini 文件中的一样。

技巧：为了让程序能够更好地移植，尽量在程序中对错误信息进行设置。

【示例 8-6】演示了当出现错误时，如何显示错误。代码如下所示。

```
<?php
$str_1 = "this is a string.";
echo $str_1;
error_reporting(E_ALL);                                    //显示所有信息
prinr($str_1);                                             //输入错误
?>
```

分析：在上述程序中，因输入错误，误将 print 输入为 prinr，PHP 在解析程序时将会报错，并给出相应的错误提示信息。程序运行结果如下。

```
this is a string.
Fatal error: Call to undefined function priny() in D:\xampplite\htdocs\book\source\8\8.6.php on line 5
```

### 8.2.3 错误处理

在程序中，因各种原因可能导致的错误，PHP 都会给出相应的提示信息。而对于错误信息的处理，除了可采用在 php.ini 文件中进行设置的方法外，还可以直接在程序中进行设置。

1．错误信息的隐藏

对于将整个系统可能产生的错误信息进行隐藏，可采用 error_reporting()函数进行设置。

【示例 8-7】重写【示例 8-5】，演示了如何隐藏所有错误信息。代码如下所示。

```
<?php
error_reporting(0);                                        //隐藏所有提示信息
$filename = "something.txt";
$handle = fopen($filename, "r");                           //打开文本文件
$contents = fread($handle, filesize ($filename));          //读取文本文件内容
print $contents;
fclose($handle);                                           //关闭文件
?>
```

分析：在上述程序中，因不存在指定的文本文件，程序应像【示例 8-5】一样给出相应的错误提示信息。但是由于在第二行采用了 error_reporting(0)函数隐藏了所有提示信息，因此就算没有当前工作目录没有该指定的文本文件，程序也不会显示出错信息。

对于单条语句可能产生的错误信息的隐藏，可采用在语句前加“@”符号进行隐藏。

【示例 8-8】采用“@”符号隐藏单条语句可能产生的错误。代码如下所示。

```
<?php
$filename = "something.txt";
```

```
$handle = @fopen($filename, "r");                          //打开给定的文本文件
$contents = @fread($handle, filesize ($filename));         //读取文件内容
print $contents;                                           //输出文件内容
fclose($handle);                                           //关闭文件
?>
```

分析：在上述程序中，因不存在该指定文件，应像【示例 8-5】一样给出相应的错误提示信息。但由于在第 3 行和第 4 行语句的前面添加了“@”符号，该行产生错误信息将不会被显示。而倒数第 2 行因没有添加“@”符号隐藏可能产生的错误信息，程序将可能显示出错信息。程序运行结果如下。

```
Warning: fclose(): supplied argument is not a valid stream resource in D:\xampplite\htdocs\book\source\8\8.8.php on line 6
```

2．错误信息的定制

在实际的应用中，显示所有的错误提示信息，将会给服务器带来安全隐患。如果隐藏所有的提示信息，在发生错误时，用户就会感觉很困惑，因为用户无法知道当前的情况，因此最好的方法就是在隐藏错误信息的同时定制错误信息。

通常采用的方法就是使用 if 语句来检测错误，并根据错误结果输出不同的信息。

【示例 8-9】采用 if 语句进行错误信息的定制处理。代码如下所示。

```
<?php
$filename = "something.txt";
if($handle = @fopen($filename, "r")){                      //使用 if 语句判断文件是否存在
    $contents = @fread($handle, filesize ($filename));
    print $contents;
    fclose($handle);
}else{
    echo "文件: ".$filename."  不存在.";
}
?>
```

分析：在上述程序中，使用 if 语句判断指定文件是否存在，不存在则给出提示信息。这样就避免了当指定文件不存在时，程序出现类似【示例 8-5】的提示信息。

以上方法虽然定制了出错信息，但是对于实际的系统，就不足以满足要求，并且页面也不够美观。为了能够让所有出错信息实现共用，通常使用 header 函数进行页面的跳转。其语法格式如下所示。

```
header("location: url")
```

其中，location 为特定格式，url 为要跳转的地址。

【示例 8-10】采用 header 函数进行错误的定制。代码如下所示。

```
<?php
$filename = "something.txt";
if($handle = @fopen($filename, "r")){                      //使用 if 语句进行判断
    $contents = @fread($handle, filesize ($filename));
    print $contents;
    fclose($handle);
}else{
    header("location: error.html");                        //使用 header 重定向错误信息页面
```

```
}
?>
```

分析：在上述程序中，当指定文件不存在时，页面将被重定向至错误信息页面 error.html。错误信息页面代码如下所示。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="zh-CN" xml:lang="zh-CN">
<head>
<meta http-equiv="content-type" content="text/html; charset=gb2312"/>
<meta http-equiv="content-language" content="zh-CN">
<title>错误信息</title>
</head>
<body style="text-align:center;">
<h1>出错啦!</h1>
<table width="500px" align="center">
<tr><td>错误信息：</td><td>文件不存在或不能用！</td></tr>
</table>
</body>
</html>
```

### 3．超时错误的处理

对于 Web 系统而言，很多时候可能因为网速太慢或者进行了过多的操作，处理时间超过了 apache 服务器所设置的最高超时时间。此时 apache 服务器将中断连接，并抛出类似如下的错误信息。

```
Fatal error: Maximum execution time of 1 second exceeded in D:\xampplite\htdocs\book\source\8\8.11.php on line 3
```

出现了超时错误后，其后的代码将不会执行。因此，可以在一些可能会超时的脚本中进行处理，让程序能够正常运行，通常采用的方法就是使用 set_time_limit 函数来延长脚本的运行时间。其语法格式如下所示。

```
void set_time_limit(int $seconds)
```

其中，参数 seconds 为要延长的秒数。该函数没有返回值。

【示例 8-11】set_time_limit 函数的应用。代码如下所示。

```
<?php
set_time_limit(1);                          //设置脚本运行时间为 1 秒
for($i=0;;$i++){
    if($i % 50000 == 0){                    //每增加 50000，输出变量 i，并且增加 1 秒运行时间
        echo $i."<br>";
        set_time_limit(1);
    }
    if($i >= 5000000) exit();               //设置跳出循环条件
}
?>
```

分析：在上述程序中，首先设置该脚本的运行时间，然后变量 i 自加 1，每增加 50000，输出变量 i 并且增加一秒的运行时间。读者可自行将第二个 set_time_limit 函数注释掉，然后运行程序，就可能会出现前面所讲的超时错误信息。对于程序倒数第 3 行，设置 for 循环的结束条件，也可以将结束条件写在 for 语句中。这里只是灵活的使用了 for 循环。

# 8.3　PHP 异常

在实际的系统运行中，有可能存在一些不可预知的错误，如文件权限不对、文件不存在等，虽然可以采用前面所讲的使用 if 语句进行错误检测，但是，PHP 提供了一种更好的异常处理方法，它能更好地解决因环境等因素而引发的异常。这一节将对 PHP 的异常及其处理做详细介绍。

## 8.3.1　异常处理原理

在 PHP 中，异常处理是使用关键字 try、catch 和 throw 来实现的。将需要进行异常处理的代码放入 try 代码块内，以便捕获可能存在的异常。每一个 try 语句必须至少有一个 catch 语句与之对应，catch 语句用于捕获异常。使用多个 catch 语句可以捕获不同类所产生的异常。throw 语句用于抛出异常。

当 try 语句不再抛出异常或找不到 catch 语句进行匹配而引发的异常时，PHP 就会转到最后一个 catch 语句的后面执行。当然，在 catch 语句中允许再次抛出（throw）异常。当一个异常发生后，该处以后的代码将不会再执行，PHP 就会尝试寻找与之匹配的 catch 语句。如果所抛出的异常没有被捕获或者也没有使用 set_exception_handler( )函数设置异常处理的话，PHP 将会产生一个严重的错误，并且输出类似 Uncaught Exception…的提示信息。

## 8.3.2　异常处理

前面已讲到，异常处理是由关键字 try、catch 和 throw 实现的。在 PHP 中，异常处理关键字的语法格式如下所示。

```
try{
        …
        throw new Exception($error);
        …
} catch (FirstException $exception){
        …
} catch(SecondException $exception){
        …
}
```

其中，在 try 和 catch 之间放置可能发生异常的代码，当检测到异常时，便使用 throw 关键字抛出异常，catch 语句用于捕获所抛出的异常。可以看出，每一个 try 语句至少有一个 catch 语句与之对应。实际上，在应用中可以使用多个 catch 关键字与之对应，以捕获不同类所引发的异常。多个 catch 关键字会按顺序执行，直到所有异常捕获完成。

throw 语句只能够抛出对象。它不能抛出任何其他的基础数据类型，如字符串或数组。PHP 内置了一个异常处理类 Exception，所有自己使用异常类都必须从它继承出来。如果尝试抛出一个不是由 Exception 类继承的类，将得到一个运行错误。

1．异常类 Excetion

异常类是 PHP 内置的异常处理类。该类的定义如下所示。

```
<?php
class Exception
{
    protected $message = 'Unknown exception';   //异常信息
    protected $code = 0;                        //用户自定义异常代码
    protected $file;                            //发生异常的文件名
    protected $line;                            //发生异常的代码行号

    function __construct($message = null, $code = 0);

    final function getMessage();                //返回异常信息
    final function getCode();                   //返回异常代码
    final function getFile();                   //返回发生异常的文件名
    final function getLine();                   //返回发生异常的代码行号
    final function getTrace();                  //backtrace()数组
    final function getTraceAsString();          //已格成化成字符串的 getTrace()信息

   function __toString();                       //可重载的方法可输出的字符串
}
?>
```

异常处理类用于在脚本发生异常时建立异常对象，该对象将用于存储异常信息并用于抛出和捕获。在准备抛出异常时，需要建立异常对象。其语法格式如下所示。

```
$except = new Exception([string $errmsg[, int $errcode]]);
```

其中，参数 errmsg 为用户自定义的错误信息；参数 errcode 为表示用户自定义的异常代码。

【示例 8-12】应用异常类进行异常处理的实例。代码如下所示。

```
<?php
class NullHandleException extends Exception{        //新建一个类，继承自 Exception 类
    function __construct($msg){
        parent::__construct($msg);
    }
}
function printObj($obj) {                           //定义函数
    if($obj == null){
        throw new NullHandleException("printObj received NULL object.");//调用自定义异常类
    }
    print $obj . "\n";
}
class MyName{                                       //定义一个类
    private $name;
    function __construct($name) {                   //构造函数
        $this->name = $name;
    }
    function __toString(){
        return $this->name;
    }
}
```

```
try{
        printObj(new MyName("Tom"));                    //调用函数
        printObj(NULL);
        printObj(new MyName("Lily"));
} catch(NullHandleException $exception){                //捕获异常
        echo $exception->getMessage();
} catch(Exception $exception){
                                                        //此处不会执行
}
?>
```

分析：在上述程序中，首先定义了一个继承自 Exception 类的自定义异常类，再定义了一个调用异常类的函数，使用 try-catch 捕获抛出的自定义异常。因为程序中只抛出了自定义异常，因此第二个 catch 将捕获不到任何异常。因调用了一个空值，程序在抛出异常后就结束了。程序运行结果如下。

```
Tom
printObj received NULL object.
```

2．设置异常处理函数 set_exception_handler

在实际的应用中，除了捕获可能发生的异常外，还可以设置一个函数来处理一些可能没有被获取的异常。PHP 提供了 set_exception_handler 函数设置异常处理函数，其语法格式如下所示。

```
string set_exception_handler(callback $exception_handler)
```

其中，参数 exception_handler 是用于处理未被捕获异常的函数名。函数设置成功返回先前定义的异常处理函数名，失败时返回 null，如果未定义用于捕获异常的函数，也会返回 null。

注意：用于处理未被捕获的异常的函数必须在使用 set_exception_handler 函数前定义。

用于处理未被捕获的异常的函数的语法格式如下所示。

```
function exception_handler($exception){
//异常处理代码
}
```

其中，参数 exception 为异常对象。

【示例 8-13】演示了如何设置异常处理函数。代码如下所示。

```
<?php
class ExceptionTest {
        public function __construct() {
                @set_exception_handler(array($this, 'exception_handler'));
                throw new Exception(__CLASS__);
        }
        public function exception_handler($exception) {
                print "Exception Caught: ". $exception->getMessage() ."\n";
        }
}
$e = new ExceptionTest;
?>
```

分析：在上述程序中，定义了一个类，用于测试异常处理函数设置成功与否。在抛出

错误信息时，使用了魔术常量__CLASS__，即抛出异常的消息为类名称。程序运行结果如下所示。

```
Exception Caught: ExceptionTest
```

3．完整的异常信息

在实际的应用中，有时为了便于调试，需要显示完整的异常信息。显示完整的异常信息只需要调用异常类的一个现有方法即可实现。

【示例 8-14】重写【示例 8-13】来显示完整的异常信息。代码如下所示。

```
<?php
class ExceptionTest {
    public function __construct() {                                    //构造函数
        @set_exception_handler(array($this, 'exception_handler'));     //设置异常处理函数
        throw new Exception(__CLASS__);
    }
    public function exception_handler($exception) {                    //定义异常处理函数
        print "异常信息: ". $exception->getMessage() ."\n";
        print "异常代码：" . $exception->getCode() . "\n";
        print "文件名：" . $exception->getFile() . "\n";
        print "异常所在行：" . $exception->getLine() . "\n";
        print "追踪路线：";print_r($exception->getTrace());;
        print $exception->getTraceAsString() . "\n";
    }
}
$e = new ExceptionTest;
?>
```

分析：在上述程序中，用户自定义异常处理函数中展示了完整的异常信息。在实际的系统开发中，可采用此种方式显示所有异常信息，对于程序的调试大有益处。

### 8.3.3 扩展的异常处理类

通常，在应用中可能存在着各种异常。如果所有异常都使用同一个异常类或者异常处理函数去获取异常，将不利于问题的解决。为此，通常是根据异常的类型不同而定义不同的自定义异常处理类。这些自定义异常处理类都是继承自 PHP 自带的异常处理类 Exception。

【示例 8-15】自定义不同类型的异常处理类。代码如下所示。

```
<?php
class FileExistsException extends Exception{}                  //定义文件存在异常类
class FileOpenException extends Exception{}                    //定义文件打开异常类

function openFile($file){                                      //定义文件检测函数
    if(!file_exsits($file)){
        throw new FileExistsException("文件不存在！请确认文件是否存在！\n",1);
    }
    if(!fopen($file,"r")){
        throw new FileOpenException("文件无法打开！请确认文件是否有可读权限！\n",2);
    }
```

```
}
$file = "test.html";
try{
        openFile($file);                                                    //调用文件检测函数
}catch(FileExistsException $exception){                                     //捕获文件存在异常
        echo "程序异常：" . $exception->getMessage();
}catch(FileOpenException $exception){                                       //捕获文件打开异常
        echo "程序异常：" . $exception->getMessage();
}catch(Exception $exception){                                               //捕获其他程序异常
        print "异常信息: ". $exception->getMessage() ."\n";
        print "异常代码：" . $exception->getCode() . "\n";
        print "文件名：" . $exception->getFile() . "\n";
        print "异常所在行：" . $exception->getLine() . "\n";
        print "追踪路线：";print_r($exception->getTrace());;
        print $exception->getTraceAsString() . "\n";
}
?>
```

分析：在上述程序中，定义两个文件操作异常类，并于 try 语句内调用文件检测函数进行检测，在 catch 语句内抛出可能发生的异常，并针对不同的异常获取不同的异常信息。程序运行结果如下。

```
程序异常：文件不存在！请确认文件是否存在！
```

在捕获异常时，针对特定类型的异常应写在前面，因为程序在捕获异常后将终止运行，后面的异常将不再会被捕获。如果将一般异常放在最前面，则只会抛出一般性的异常信息，专属于某类型的异常信息不会被捕获，这将不利于程序的调试。

### 8.3.4　异常的传递与重掷

在进行实际应用中的流程处理时，有可能触发到异常，但却无法马上执行，希望暂时忽略处理异常，在适当的时候再处理异常。对于这种情况有一种方案，将处理异常的责任交回调用当前方法的代码，然后在上一层 catch 语句中再次抛出异常。使异常沿着方法的调用链向上传递，这就叫异常的重掷。

【示例 8-16】重写【示例 8-15】，演示了如何进行异常的重掷。代码如下所示。

```
<?php
class FileExistsException extends Exception{}
class FileOpenException extends Exception{}

function openFile($file){
        try{
                if(!file_exists($file)){
                        throw new FileExistsException("文件不存在！请确认文件是否存在！\n",1);
                }
                if(!fopen($file,"r")){
                        throw new FileOpenException("文件无法打开！请确认文件是否有可读权限！\n",2);
                }
        }catch(Exception $e){                                               //捕获异常
```

```
            echo __FUNCTION__ . "出现异常！ ";
            throw $e;                                          //重掷异常
        }
}
$file = "test.html";
try{
        openFile($file);
}catch(FileExistsException $exception){
        echo "程序异常：" . $exception->getMessage();
}catch(FileOpenException $exception){
        echo "程序异常：" . $exception->getMessage();
}catch(Exception $exception){
        print "异常信息: ". $exception->getMessage() ."\n";
        print "异常代码：" . $exception->getCode() . "\n";
        print "文件名：" . $exception->getFile() . "\n";
        print "异常所在行：" . $exception->getLine() . "\n";
        print "追踪路线：";print_r($exception->getTrace());;
        print $exception->getTraceAsString() . "\n";
}
?>
```

分析：在上述程序中，在检测函数中使用 try 语句捕获异常，并输出错误信息。然后在 catch 语句中将异常进行重掷，将异常对象交由上级 catch 语句进行处理，最后在外层的语句中捕获到该异常。程序运行结果如下。

```
openFile 出现异常！ 程序异常：文件不存在！ 请确认文件是否存在！
```

## 8.4 PHP 程序的调试

在 PHP 的程序开发中，不可避免地会对程序进行调试。要进行调试，首先必须打开错误报告，再由相应的语句输出信息至屏幕，并根据这些信息找出并修复程序中可能存在的错误。

### 8.4.1 打开错误报告

在前面的章节已讲到了 PHP 的配置文件 php.ini，要能够显示错误信息需要修改该文件两变量的值。但在实际的应用中，所开发的程序可能会进行移植。如果每一次移植都去修改 PHP 配置文件 php.ini，会非常麻烦。PHP 提供了另一种不需要直接修改配置文件而达到修改该变量值的方法，那就是使用函数 ini_set 实现配置文件变量值的修改。该函数的语法格式如下所示。

```
string ini_set(string $varname, string $newvalue)
```

其中，参数 varname 为要进行修改的变量名；参数 newvalue 为要修改的变量的值。该函数实现将 PHP 配置文件 php.ini 中的 varname 变量的值修改为 newvalue。函数修改配置成功，返回该变量的旧值，修改失败则返回 false。该配置选项将在脚本执行期间保持修改的值，直到脚本执行结束。脚本执行结束后，配置选项将恢复原来的值。

注意：不是所有配置选项都可以使用 ini_set 函数修改，对于具体有哪些选项可以使用该函数进行修改，读者可参考相关文档。

对于使用 ini_set 函数修改配置选项是否成功，可使用 ini_get 函数获取该变量的值，然后显示获取的值，即可以知道是否修改成功。该函数的语法格式如下所示。

```
string ini_get($varname)
```

其中，参数 varname 为获取其变量值的变量名。函数成功获取配置选项的值时返回该值，失败时返回空字符串或 null。

【示例 8-17】在脚本中直接打开系统错误报告。代码如下所示。

```
<?php
@ini_set("display_errors","on");                          //显示错误信息
@ini_set("error_reporting",E_ALL & ~E_NOTICE);            //显示除通告外的所有信息

echo "display_errors: " . ini_get("display_errors") . "\n";
echo "error_reporting: " . ini_get("error_reporting") . "\n";

print("this is a string.\n");                             //输出字符串
prinr("this is a string.\n");                             //书写错误
?>
```

分析：在上述程序中，直接在脚本中修改配置选项，让 PHP 显示错误信息，并且显示除通告外的所有信息。倒数第 2 行程序因输入错误，PHP 将会给出类似以下的提示信息。

```
Fatal error: Call to undefined function prinr() in D:\xampplite\htdocs\book\source\8\8.17.php on line 9
```

类似的，为了在开发中方便调试，可直接在脚本打开错误显示，这样就不会因为环境问题看不到错误信息而无法调试。当然，在正式系统中，应在脚本中关闭错误显示。

### 8.4.2　使用 ECHO 进行调试

在开发中，当程序出现警告或错误时，打开错误报告查看错误信息是很好的调试方式。但是，当程序没有语法错误时，程序由于某种原因而得不到期望的结果，采用查看错误报告的方法就不能进行调试了。此时就需要查看中间变量的值，一步步进行跟踪调试。

【示例 8-18】通过查看中间变量的值进行调试。代码如下所示。

```
<?php
$num_1 = 109;
echo '$num_1 = ' . $num_1 . "<br>";
$num_2 = 300;
echo '$num_2 = ' . $num_2 . "<br>";
$sum = $num_1 + $num_2;
echo '100 + 300 = ' . $sum;
?>
```

分析：在上述程序中，采用逐行输出变量值的方式进行调试。通过程序输出内容，可以很容易地看出程序的错误在第一个变量。程序运行结果如下。

```
$num_1 = 109
$num_2 = 300
100 + 300 = 409
```

这个程序很简单，几乎可以直接看出程序错在哪里。但如果是逻辑复杂的程序，就不容易直接看出，也不可能一步步输出变量值进行追踪调试，这时可采用下面的方法进行调试。

### 8.4.3 使用 DIE 进行调试

对于一些逻辑相对简单的程序，可首先粗略地判断错误出在哪一段，再从该段输出相关变量的值，根据输出的变量值一步步找出程序中存在的问题。但是对于一些逻辑复杂的程序而言，采用这种方法不足以满足要求。此时可使用 die( )语句进行调试，die( )语句会中止程序执行，并在浏览器上显示文本。如果不想注释掉代码，只想显示到出错之前的信息和出错信息，那么 die()语句特别有用。

【示例 8-19】使用 die()进行程序的调试。代码如下所示。

```
<?php
$num = 100;
$mod = 5;
for ($i = 0; $i < $num; $i++){
        if ($i % $mod == 3) {
                die("Error in " . " File: " . __FILE__ . " on line: " . __LINE__);
        }
}
?>
```

分析：在上述程序中，使用 die( )语句对程序进行调试。当发生错误时，程序将在错误发生处中止执行，并在浏览器显示出错信息。根据所提示的错误信息就可以快速地定位于错误行，并进行修改。

程序运行结果如下。

```
Error on File: D:\xampplite\htdocs\book\source\8\8.19.php on line: 7
```

## 8.5 使用 ZendStudio 进行调试

前面讲到了直接在程序中输出中间变量的值或输出错误信息的方法进行 PHP 程序的调试。其实除了这种直接的方法以外，还可以借助于第三方的工具进行调试。例如，PHPEclipse、ZendStudio 等专业的 IDE 开发工具，都具有实时调试的功能。这一节将对 Eclipse+ZendStudio 这一组合开发工具进行详细地介绍。

### 8.5.1 使用 Eclipse 开发 PHP 的优点

Eclipse 平台是一个开放的平台，用户可以免费地获取。而且可跨平台支持多种操作系统，

如 Windows、Linux、Mac 等。Eclipse 可展性好，读者需要某项功能，可下载相应的扩展包即可。Eclipse 安装方便，只需要将相应的文件解压至文件夹即可，省去了安装的步骤。

ZendStudio 是 PHP 商业公司推出的一个屡获大奖的专业 PHP 的集成开发环境。它具备功能强大的专业编辑工具和调试工具，支持 PHP 语法高亮显示、语法自动填充功能、书签功能、语法自动缩排和代码复制功能，内置一个强大的 PHP 代码调试工具，支持本地和远程两种调试模式，支持多种高级调试功能。

### 8.5.2　Eclipse 和 ZendStudio 的安装

对于 Eclipse 和 ZendStudio 的安装，读者可参考以下步骤进行下载安装。

（1）下载 Eclipse。可从 Eclipse 官方网站“http://www.eclipse.org/download/”下载 Eclipse 的最新版本，目前 Eclipse 的最新版本为 3.4。因为 Eclipse 运行需要 Java 2 SDK 支持，读者在下载时还需下载包含 Java 2 的安装包。

（2）喜欢中文界面的读者可从 Eclipse 的官方网站“https://babel.eclipse.org/babel/”下载中文语言包。

（3）下载后，将 Eclipse 解压至某指定的文件夹，如 D:\eclipse。解压完成后，将下载的中文语言包解压至 Eclipse 文件夹，双击该文件夹中的 eclipse.exe 启动 Eclipse。

（4）第一次启动 Eclipse，Eclipse 会要求输入一个路径作为工作区（Workspace）。这个工作区读者可随便设置，设置完成后，读者就可以看到 Eclipse 的中文界面。

（5）依次单击【Help】→【Software Updates】→【Find and Install】命令，此时会弹出【Install/Update】窗口，点选【Search for new features to install】单选按钮，单击【下一步】按钮，此时将显示可升级的服务器列表。

（6）单击【New Remote Site】按钮，在弹出窗口的【name】文本框中输入“Zend Update Site”，在【URL】文本框中输入“http://downloads.zend.com/studio-eclipse/updates/6_1”，单击【ok】按钮。

（7）在【Sites to include in search】选项组中选择【Ganymede Discovery Site】和【Zend Update Site】选项，单击【finish】按钮。

注意：【Ganymede Discovery Site】选项中包含了 ZendStudio 所必需的所有第三方组合。

（8）升级管理器将从所选择的站点搜索可能的升级包，然后显示可升级的镜像网站列表，选择最近下载的服务器，并单击【OK】按钮。从搜索结果窗口中选择【Zend Update Site】和【Zend Debugger】选项。

（9）单击【Select Required】按钮，升级管理器将自动选择所必需的安装包，这可能需要几分钟点时间。单击【Next】按钮开始进行安装。

（10）此时将显示所选择安装包的安装协议，点选【I accept the terms in the license agreements】单选按钮，单击【Next】按钮。

（11）此时将显示所有已选择要进行安装的列表。单击【Next】按钮，显示列表中的所有扩展将被安装。

（12）单击【Finish】按钮。安装所有的扩展，被安装的扩展将被下载并显示一个进度条。这可能需要几分钟的时间。此时可能会显示一个包含未被通过的安全验证的窗口，单击【Install All】按钮，安装所有扩展。

（13）当所有安装完成时，弹出一个需要重新启动 Eclipse 的窗口，单击【Yes】按钮重启 Eclipse。至此，所有 ZendStudio 所必需的扩展安装完成。

说明：ZendStudio 只提供 30 天的试用，30 天后 ZendStudio 的所有功能就不能再使用了。读者可从 Zend 官方网站“http://www.zend.com/en/store/software/studio”购买许可证。

### 8.5.3 Zend Debugger 的安装

本书以 xampp 为例进行安装。从网上下载 xampp 压缩包，一定要解压至盘符的根目录下，否则将可能无法启动。到 Zend 官方网站上下载 Zend Debugger，“http://downloads.zend.com/pdt/server-debugger/ZendDebugger-5.2.12-cygwin_nt-i386.zip”。将压缩包里面的 ZendDebugger.dll 复制到 xampp 文件夹下的 php/ext 文件夹中。

在 xampp 目录下的 htdocs 文件夹中建立一个 phpinfo.php 文件，其内容如下所示。

```
<?php
phpinfo();
?>
```

在浏览器中输入“http://127.0.0.1/phpinfo.php”，查看 php.ini 文件的位置。一般该文件是在“xamppite\apache\bin\”目录下。

注意：在 xampplite\php 目录下也有一个 php.ini，此时应根据 phpinfo 所看到的 php.ini 文件位置为准。

打开 php.ini，找到如下内容。

```
[Zend]
;zend_extension_ts = "\xampplite\php\zendOptimizer\lib\ZendExtensionManager.dll"
;zend_extension_manager.optimizer_ts = "\xampplite\php\zendOptimizer\lib\Optimizer"
;zend_optimizer.enable_loader = 0
;zend_optimizer.optimization_level=15
;zend_optimizer.license_path =
```

将以上内容替换成如下内容。

```
[Zend]
zend_extension_ts="e:\xampp\php\zenddebugger.dll"
zend_debugger.allow_hosts=127.0.0.1/32,192.168.0.0/24
zend_debugger.expose_remotely=always
```

注意：zend_debugger.allow_hosts 的值改为你相应的 IP 地址

重启 xampp，再在浏览器中输入“http://127.0.0.1/phpinfo.php”，如果出现如图 8-1 所示的界面，说明安装成功。如未安装成功，请重新按安装步骤进行安装。

在 Eclipse 菜单栏中单击【Window】→【Preference】命令，在弹出的【Preference】窗口中选择【PHP】→【Debug】命令，单击【PHP Excuteables】链接，将先前解压的 xampp 的 PHP 路径加进来，如图 8-2 所示。

添加完成后，单击【确定】按钮。最后单击【应用】按钮，完成 Zend Debugger 的安装。

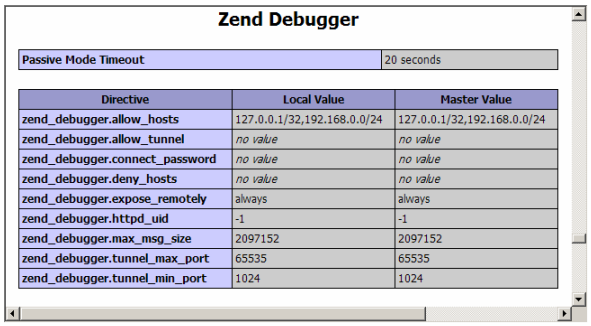


图 8-1　Zend Debugger 安装确认

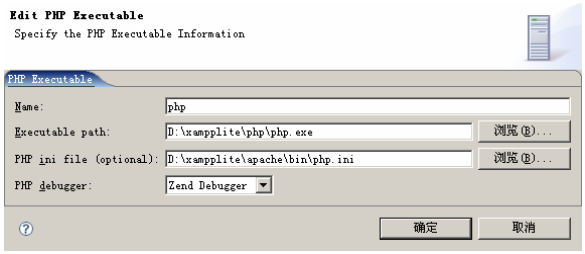


图 8-2　设置 PHP 路径

### 8.5.4　使用 ZendStudio 进行调试

单击 Eclipse 菜单栏中的【文件】→【新建】→【PHP Project】命令，在弹出的窗口中输入项目名称，单击【完成】按钮，完成 PHP 项目的创建。

单击 Eclipse 菜单栏中单击的【文件】→【新建】→【PHP File】命令，在弹出的窗口中输入文件夹的位置及新建的 PHP 文件的文件名。单击【完成】按钮，完成新文件的创建。

创建项目和文件完成后，在 Eclipse 编辑器输入如下代码。

```
<?php
$num_1 = 100;
echo '$num_1 = ' . $num_1 . "<br>";
$num_2 = 300;
echo '$num_2 = ' . $num_2 . "<br>";
$sum = $num_1 + $num_2;
echo '100 + 300 = ' . $sum;
?>
```

在 Eclipse 中调试的步骤如下。

（1）在需要进行调试的地方设置断点。设置断点是通过将光标移动到要设置断点的行，再单击菜单栏中的【运行】→【Toggle Line Breakpoint】命令实现，也可双击要设置断点的行的左侧。设置后该行左侧将出现一个蓝色小圆点。

（2）单击【运行】→【Debug】命令开始进行调试。

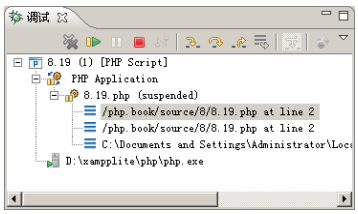


图 8-3　【调试】标签

（3）在【调试】标签下可以看到进程在第二行就暂停了。此时可以通过单步执行进行程序的调试，如图 8-3 所示。

（4）选择【变量】标签可实时地看到当前环境的所有变量，包括所有的系统环境变量和脚本中的变量，系统变量排在脚本变量的前面，如图 8-4 所示。在【Browser Output】标签下可以实时看到程序的输出结果，如图 8-5 所示。

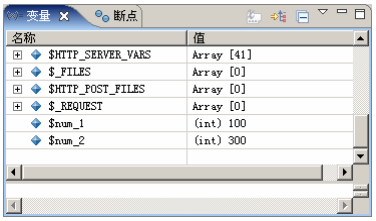


图 8-4　【变量】标签

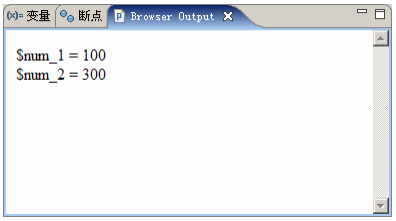


图 8-5　【Browser Output】标签

到此，ZendStudio 进行调试的过程就讲完了，读者可在实际操作中体会 ZendStudio 的强大功能。

## 8.6　本章实例

在 Web 系统中，常常会因某种原因而产生异常。需要对于可能会产生异常的地方进行捕获、分析、处理。

【示例 8-20】异常的捕获、处理。代码如下所示。

```
<?php
    /* PDO 连接 mysql 数据库*/
    $dsn = 'mysql:host=localhost;dbname=testdb';
    $user = 'dbuser';
    $password = 'dbpass';

    try {
        $dbh = new PDO($dsn, $user, $password); //创建数据库连接对象容易出现异常
        echo '如果上面出现异常就不能显示';
    } catch (PDOException $e) {
        echo 'Connection failed: ' . $e->__toString();
    }

    //try..cathc 和 throw 一起用
    try {
        $error = '我抛出异常信息，并且跳出 try 块';
        if(is_dir('./tests')){
            echo 'do sth.';
        }else{
            throw new Exception($error,12345);
        }
```

```
        echo '如果上面出现异常就不能显示！～<br />',"\n";
    } catch (Exception $e) {
        echo '捕获异常: ', $e->getMessage(),$e->getCode(), "\n<br />"; //
    }
    echo '继续执行';
?>
```

分析：在上述程序执行时，有可能因某种原因而引发异常，此时就需要对可能产生的异常进行捕获和处理，如采用 PDO 进行数据表的连接。有时也可根据条件手动抛出异常，并采用 catch 语句捕获该异常，并进行异常信息的输出，如本例中判断文件夹是否存在。

## 8.7　小结

本章介绍了 PHP 中的错误、异常处理及 PHP 程序的调试。在 PHP 中，包含几种错误，对于错误的处理也有不同的方式，读者应根据具体情况做不同的处理。

在实际的 PHP 项目中，常常会用到异常处理。采用异常处理机制，可将所有的异常处理放到一个统一的位置，并可以指定异常信息和异常代码。

对于 PHP 的开发，常常需要对 PHP 代码进行调试。可通过 print 函数和 echo 语句抛出变量内容做最简单的调试，也可以采用专用的集成开发工具对 PHP 代码进行更加专业的调试。读者可根据项目选择合适的程序调试方式。