第五章 PHP5 异常处理

目录

5.1 PHP中的错误信息 3
5.1.1 php.ini中配置错误消息 3
5.1.2 php中错误消息处理 4
5.1.3 php代码中调整错误级别 5
5.1.4 自定义错误处理 6
5.2 PHP5 中的SPL模块 7
5.2.1 什么是SPL 7
5.2.2 spl.php中的异常处理类 8
5.2.3 spl.php中的其它异常类 11
5.3 PHP5 异常捕获 14
5.3.1 异常实例 14
5.3.2 抛出异常 15
5.3.3 在代码中捕获异常 16
5.3.4 在代码中捕获异常(2) 18
5.3.5 一个catch块处理多种异常 19
5.3.6 多个catch块处理异常 20
5.3.7 异常处理块嵌套 23
5.3.8 异常向外抛出 24
5.4 PHP5 自定义异常 25
5.4.1 自定义异常 25
5.5 异常处理实例 26
5.5.1 验证实例 26
5.5.2 验证实例代码 29
5.6 小结 34

5.1 PHP 中的错误信息

5.1.1 php.ini 中配置错误消息

在 PHP4 中，没有异常 **Exception** 这个概念，只有 错误 **Error**。我们可以通过修改 php.ini 文件来配置用户端输出的错误信息。

在 php.ini 中，一个分号 **；** 表示注释。

Php.ini 将能够显示的错误类型分为如下种类。

```
; E_ALL                -所有的错误和警告，（不包含 E_STRICT）.
; E_ERROR              -致命的运行时错误
; E_RECOVERABLE_ERROR  - 几乎致命的运行时错误
; E_WARNING            - 运行时的警告（非致命错误）
; E_PARSE              -编译时解析错误
; E_NOTICE             - 运行时的提示，这些提示常常是代码中的 bug 引起的，也许是故意的（如使用一
                         个未初始化的变量，事实上它被自动初始化成一个空字符串） 。
; E_STRICT             - 运行时提示，能够给于 PHP 建议，以改变你的代码，以获得最好的协同性，并完
善代码的兼容性。
; E_CORE_ERROR         - PHP 初始化启动过程中的致命错误。
; E_CORE_WARNING       - PHP 初始化启动过程中的非致命错误。
; E_COMPILE_ERROR      - 致命的编译错误。
; E_COMPILE_WARNING    - 编译错误（非致命的错误）。
; E_USER_ERROR         - 用户错误信息。
; E_USER_WARNING       - 用户警告信息。
; E_USER_NOTICE        -用户提示信息。；
```

在 php.ini 中 error_reporting 控制输出到用户端的消息种类。

以下几种是 php.ini 中推荐的几种配置。

```
error_reporting  = E_ALL
表示输出所有的信息。
```

```
error_reporting  = E_ALL & ~E_NOTICE 表示输出所有的错误，除了提示。
```

```
error_reporting  = E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR
表示输出所有的 ERROR 信息。
```

在 php.ini 中，display_errors 可以设置是否将以上设置的错误信息输出到用户端。

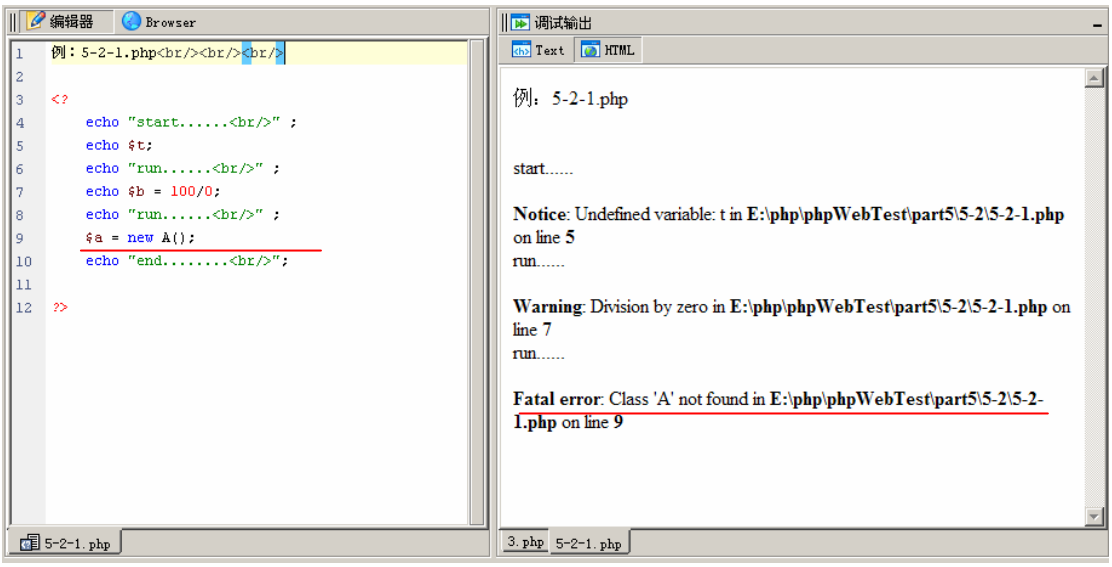
```
display_errors = On   输出到用户端（调试代码时候，打开这项更方便）
```

```
display_errors = OFF 消息将不会输出到用户端（最终发布给用户时记得改成 off）
```

5.1.2 php 中错误消息处理

在 php 中，对于错误处理非常的宽松。php 系统会尽量让程序运行下去，除非遇到致命错误。

例 5-2-1.php



第 5 行，直接打印一个未赋值变量$t 时候，系统报出一个 Notice，未定义变量。

第 7 行，做除以 0 的运算时，系统报出一个 Warning，提示有除以 0 这样的警告，程序依然在运行。

第 9 行，当实例化一个不存在的类的时候，发生致命错误，程序终止运行。

再次提示：如果不想显示错误信息给用户看到，设置 php.ini 中
display_errors = OFF

5.1.3 php 代码中调整错误级别

除了在 php.ini 文件中可以调整错误消息的显示级别外，在 php 代码中也可以自定义消息显示的级别。

PHP 提供了一个方便的调整函数。

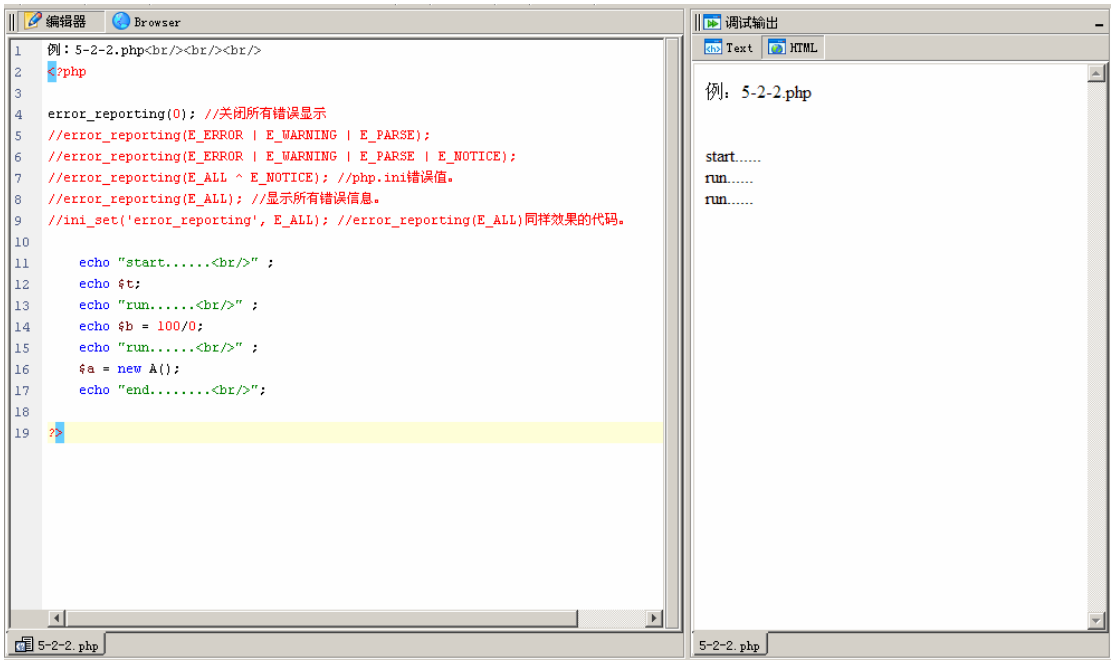
```
int error_reporting ( [int level] )
```

使用这个函数可以定义当前 php 页面中错误消息的显示级别。

参数 level 使用了二进制掩码组合的方式。

value	constant	value	constant
1	E_ERROR	2	E_WARNING
4	E_PARSE	8	E_NOTICE
16	E_CORE_ERROR	32	E_CORE_WARNING
64	E_COMPILE_ERROR	128	E_COMPILE_WARNING
256	E_USER_ERROR	512	E_USER_WARNING
1024	E_USER_NOTICE	2047	E_ALL
2048	E_STRICT	4096	E_RECOVERABLE_ERROR

例 5-2-2.php



5.1.4 自定义错误处理

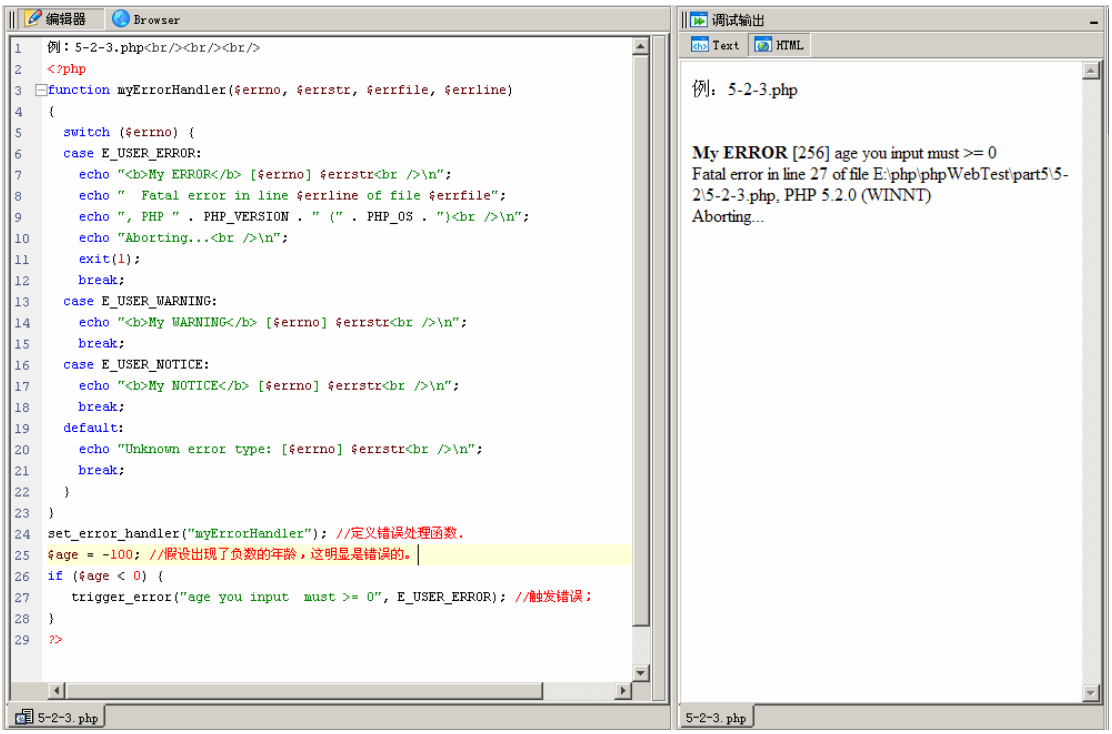
在 php 中，可以自定义对错误处理的方式。

首先要自定义一个错误处理函数，然后使用 **set_error_handler()**函数向系统声明错误处理函数。代码中产生的错误就会使用这个错误处理函数了。

使用 trigger_error()函数，可以触发一个 error。

例如 trigger_error("age you input　must >= 0", E_USER_ERROR)，触发自己的错误信息。

例：5-2-3.php



5.2 PHP5 中的 SPL 模块

5.2.1 什么是 SPL

在 PHP5 中有一个独特的部分 **SPL** - **StandardPHPLibrary Modules** （PHP 标准库）。

在 SPL 文档中这样说明的：

SPL – PHP 标准库是用来解决标准问题并实现一些高效数据访问的接口和类的集合。你会发现这些类用 PHP 代码编写在 sql.php 文件中，或在对应例子、内核的 .inc 文件中。基于这些内核的实现或在示例目录下的，也是一些 .php 文件。

在 SPL 中主要包含了以下内容：

1. **Iterators** : SPL offers some advanced iterator algorithms。
   迭代器: SPL 提供了一些高级的迭代器运算法则。
2. **Directories and Files** : SPL offers two advanced directory and file handling classes:
   目录和文件：SPL 提供了两个高级路径和文件处理类。
3. **XML** : SPL offers an advanced XML handling class:
   XML ： SPL 提供了一个高级 XML 处理类。
4. **Array Overloading** : SPL offers advanced Array overloading:
   数组重载 ：SPL 提供了高级数组重载。
5. **Counting** : interface Countable allows to hook into the standard array function count().
   计数：接口 Countable 允许勾住标准数组方法 count().
6. **Exceptions** : SPL provides a set of standard Exception classes each meant to indicate a certain problem type.
   异常：SPL 规定了一套标准异常类，每个类都标识了一类确定的问题。
7. **Observer**: SPL suggests a standard way of implementing the observer pattern.
   观察者：SPL 提出了实现观察者模式的标准。

关于SPL更多资料，请看 http://www.php.net/~helly/php/ext/spl 。

而后面我们讲只介绍 SPL 中的异常处理部分。

5.2.2 spl.php 中的异常处理类

从 PHP5.0 开始，在 SPL 中引入了异常处理类。

Notice：异常与错误在 PHP 中是两个完全不同的概念。

在 PHP 源码包中能找到这个文件 spl.php，在 spl.php 中定义了一个异常类 Exception。在这个类中，定义了一些属性如下：

protected $message ；
存储异常信息的变量。

private $string;
格式化过以后的异常信息。

protected $code;
通过构造函数传递的 异常代码。

protected $file;
产生异常的 php 文件的文件名。

protected $line;
引起异常的代码在 php 文件中所在的行数。

private $trace;
引起异常后，包含相关信息的一个数组。

构造函数如下：

```
function __construct($message = NULL, $code = 0) {
    if (func_num_args()) {    // func_num_args()返回参数数量
        $this->message = $message;
    }
    $this->code = $code; //错误代码默认是 0；
    $this->file = __FILE__; //  文件名
    $this->line = __LINE__; //  行号
    $this->trace = debug_backtrace(); //返回一个包含多个元素
    $this->string = StringFormat($this); //格式化字符串
}
```

其中还包含了__clone() 方法和对应这些属性的 geter 方法。

spl.php 源代码如下：

编辑器 Browser

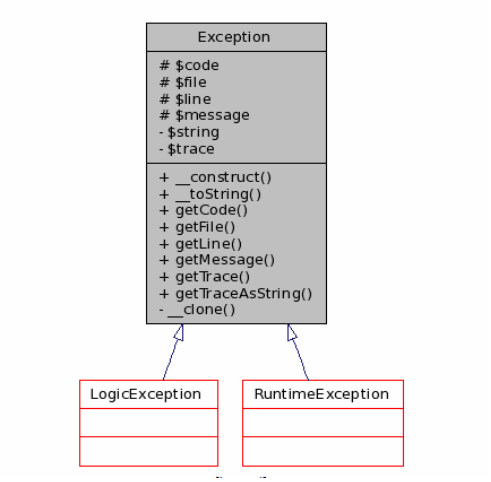
```php
 */
class Exception
{
    /** The exception message */
    protected $message;

    /** The string represenations as generated during construction */
    private $string;

    /** The code passed to the constructor */
    protected $code;

    /** The file name where the exception was instantiated */
    protected $file;

    /** The line number where the exception was instantiated */
    protected $line;

    /** The stack trace */
    private $trace;

    /** Prevent clone
     */
    final private function __clone() {}

    /** Construct an exception
     *
     * @param $message Some text describing the exception
     * @param $code    Some code describing the exception
     */
    function __construct($message = NULL, $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);
    }

    /** @return the message passed to the constructor
     */
    final public function getMessage()
    {
        return $this->message;
    }

    /** @return the code passed to the constructor
     */
    final public function getCode()
    {
        return $this->code;
    }

    /** @return the name of the file where the exception was thrown
     */
    final public function getFile()
    {
        return $this->file;
    }

    /** @return the line number where the exception was thrown
     */
    final public function getLine()
    {
        return $this->line;
    }
```

```
308     /** @return the stack trace as array
309      */
310     final public function getTrace()
311     {
312         return $this->trace;
313     }
314
315     /** @return the stack trace as string
316      */
317     final public function getTraceAsString()
318     {
319     }
320
321     /** @return string represenation of exception
322      */
323     public function __toString()
324     {
325         return $this->string;
326     }
327 }
328
```

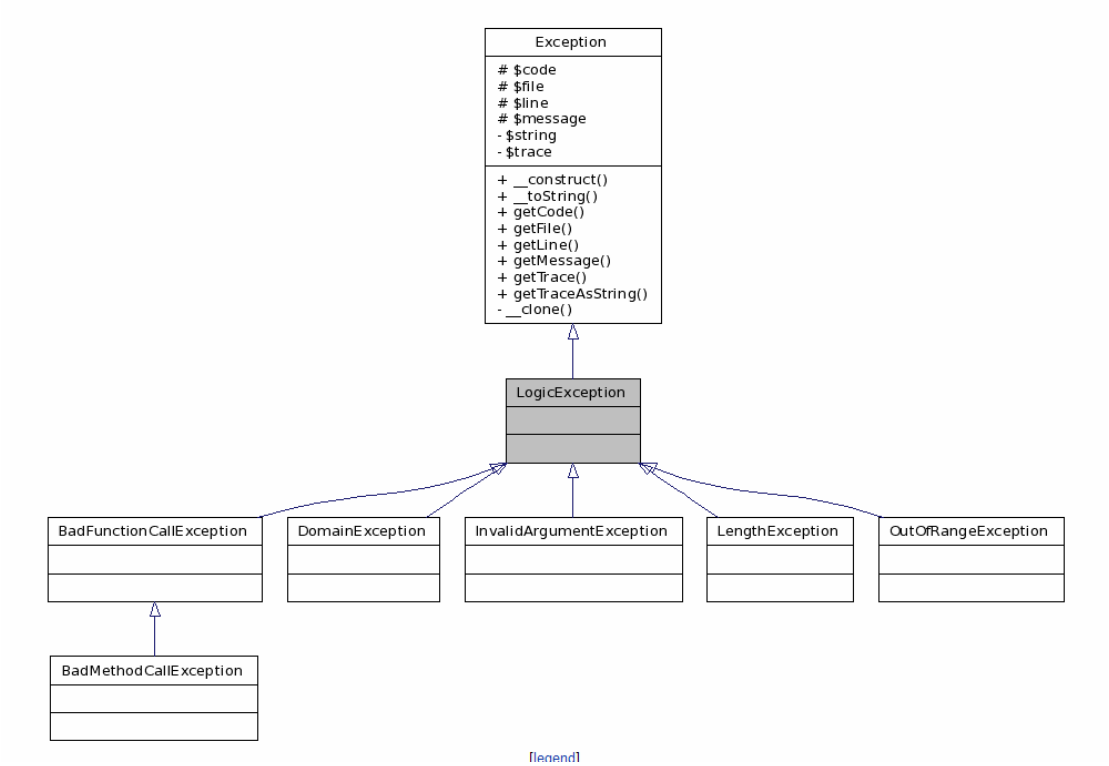
5.2.3 spl.php 中的其它异常类

在 SPL 中还定义了其它的异常类，以对应不同的异常类型。这些异常类都是 Exception 类的子类。

在 Exception 类有两个直接子类 LogicException 和 RuntimeException，分别表示逻辑异常和执行异常。



LogicException 又衍生出其它的逻辑异常子类。



class LogicException extends Exception
　　程序中的逻辑错误的异常类，它是 Exception 类的直接子类。

class BadFunctionCallException extends LogicException
　　当不合法的函数被调用产生的异常类。

class BadMethodCallException extends BadFunctionCallException
　　当不合法的方法被调用产生的异常类。

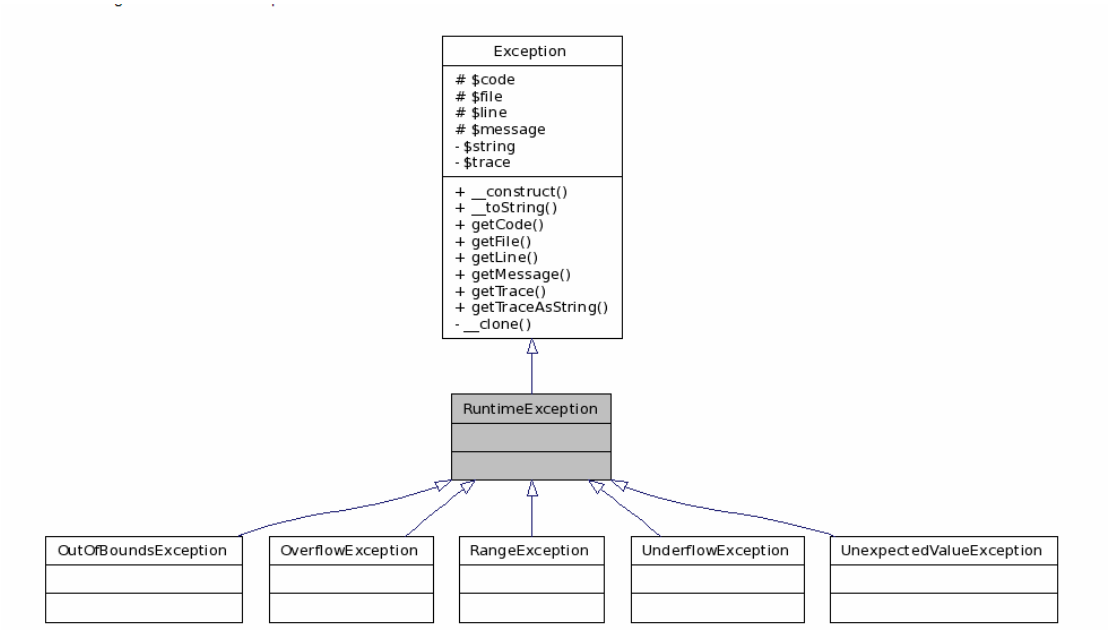
class DomainException extends LogicException
　　表示一个值不在有效范围内的异常。

class InvalidArgumentException extends LogicException
　　表示传递了无效的参数产生的异常。

class LengthException extends LogicException
　　表示一个参数超过了许可的长度的异常。

class OutOfRangeException extends LogicException
　　表示请求检索超越了数组等容器最大长度的异常。

RuntimeException 衍生出其它运行异常子类。



class RuntimeException extends Exception
只有在执行时才能发现的异常，是 Exception 的直接子类。

class OutOfBoundsException extends RuntimeException
表示请求检索超越了数组等容器最大长度的异常。

class OverflowException extends RuntimeException
表示算法/缓存溢出异常

class RangeException extends RuntimeException
运行期间的范围异常

class UnderflowException extends RuntimeException
运行期间的算法/缓存的向下溢出异常。

在 spl.php 中所有 Exception 子类的代码都仅仅是类的定义和简单的父类继承。
而方法内部没有任何扩展、重写。
如：LogicException 的定义。

```
329 /** @ingroup SPL
330  * @brief Exception that represents error in the program logic.
331  * @since PHP 5.1
332  *
333  * This kind of exceptions should directly leed to a fix in your code.
334  */
335 class LogicException extends Exception
336 {
337 }
```

又如：LengthException 的定义部分

```
378 /** @ingroup SPL
379  * @brief Exception thrown when a parameter exceeds the allowed length.
380  * @since PHP 5.1
381  *
382  * This can be used for strings length, array size, file size, number of
383  * elements read from an Iterator and so on.
384  */
385 class LengthException extends LogicException
386 {
387 }
```

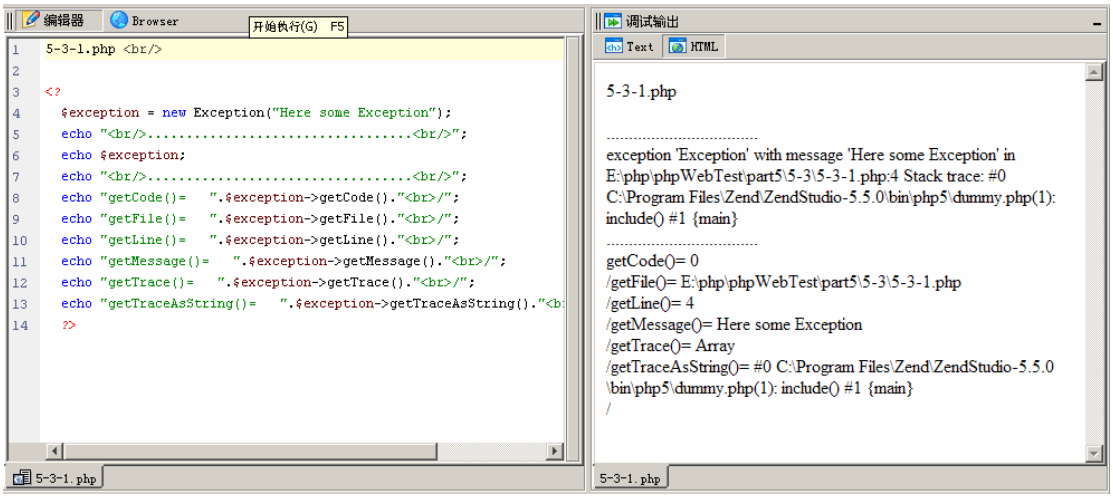
5.3 PHP5 异常捕获

5.3.1 异常实例

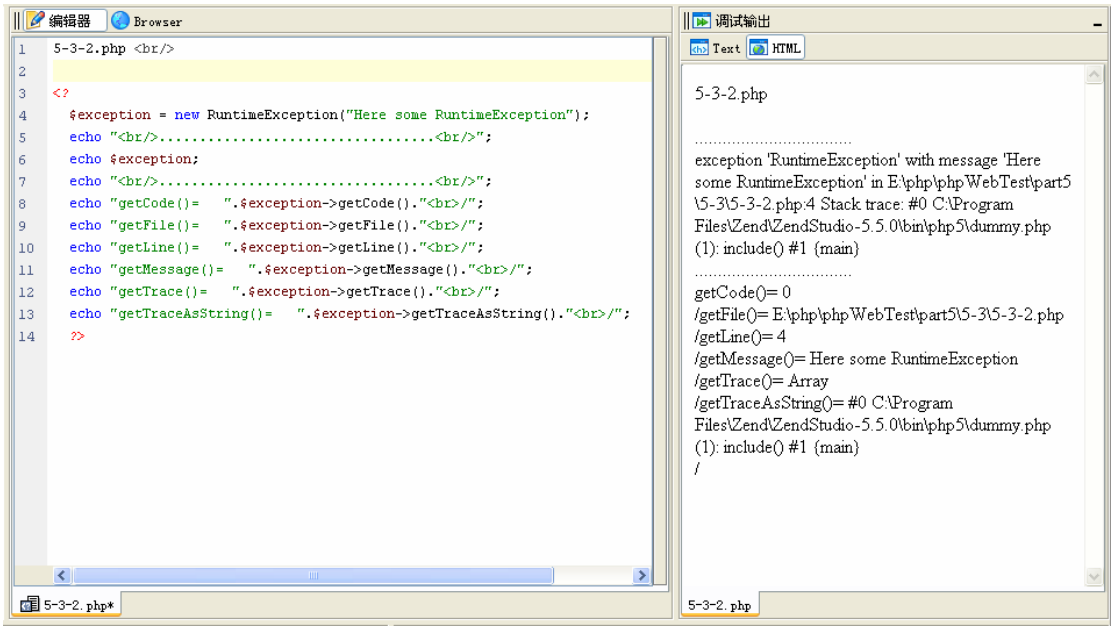
可以通过实例化 Exception 类或者它的子类来创建一个异常实例。

有了这个异常实例，就可以通过 Exception 中定义好的 getter 方法，获得相应的属性值。

例 5-3-1.php



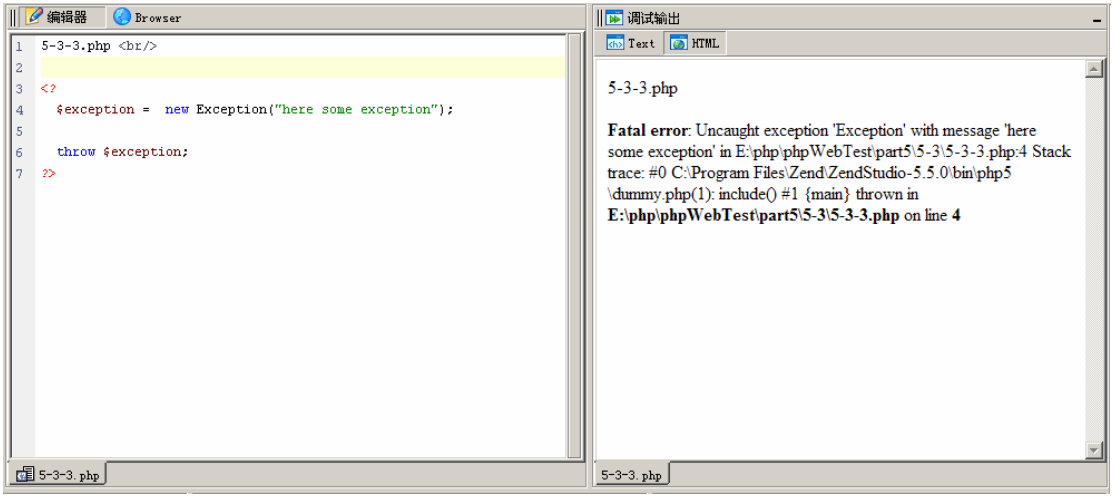
例 5-3-2.php



5.3.2 抛出异常

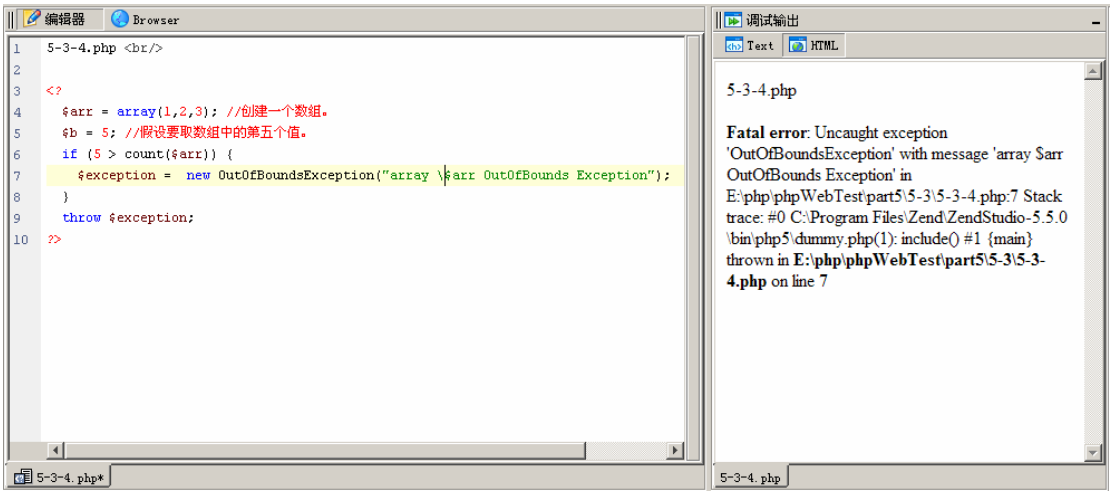
在 PHP5 中使用 throw 关键字，向外抛出一个异常实例。
如果这个异常如果未经处理，将会导致系统产生致命错误，而使代码终止。

例 5-3-3.php



根据需求，我们可以向系统抛出不同的异常。
在 php 中数组越界是不会产生知名错误的，而下面的代码抛出一个数组越界的异常，导致代码运行终止。

例 5-3-4.php



5.3.3 在代码中捕获异常

可以通过 PHP5 支持的 **try catch** 语句捕获并处理异常。

语法如下：

```
try{
    //可能引发异常的语句
}catch(异常类型  异常实例){
    //异常处理语句
}
```

在 try 块中，放置可能产生异常的代码片段。在 catch 块中放置对这个异常处理的代码。

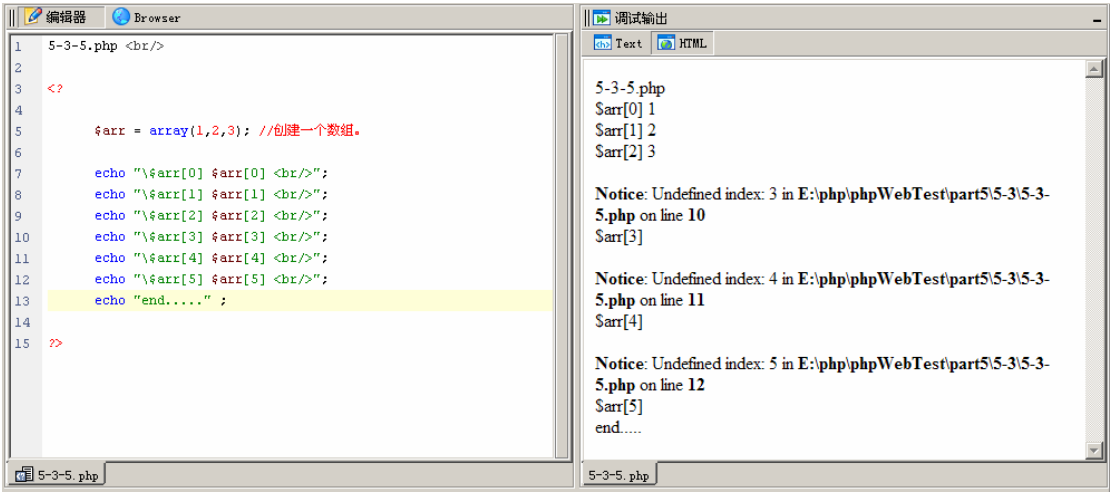
如：

```
try{
    throw new Exception("new Exception");   // 引发抛出异常
}catch(Exception  $ex){
    echo $ex; //打印这个异常对象
}
```

- 如果 try 块未产生任何异常，try 块将运行完毕，catch 块内容不会被执行。
- try 块如果抛出了异常，会立刻在 catch 中寻找可以捕获改异常的 catch 块，并运行相应的 catch 块代码，然后跳出 try catch 块继续运行。 而 try 块中抛出异常后面的代码将被跳过。
- 如果 try 块中的异常不能被 catch 块捕获，将抛向系统引发系统致命错误，代码终止运行。
- 在 catch 中，异常类型后面跟的是一个变量，这个变量将指向内存中被捕获的异常实例。

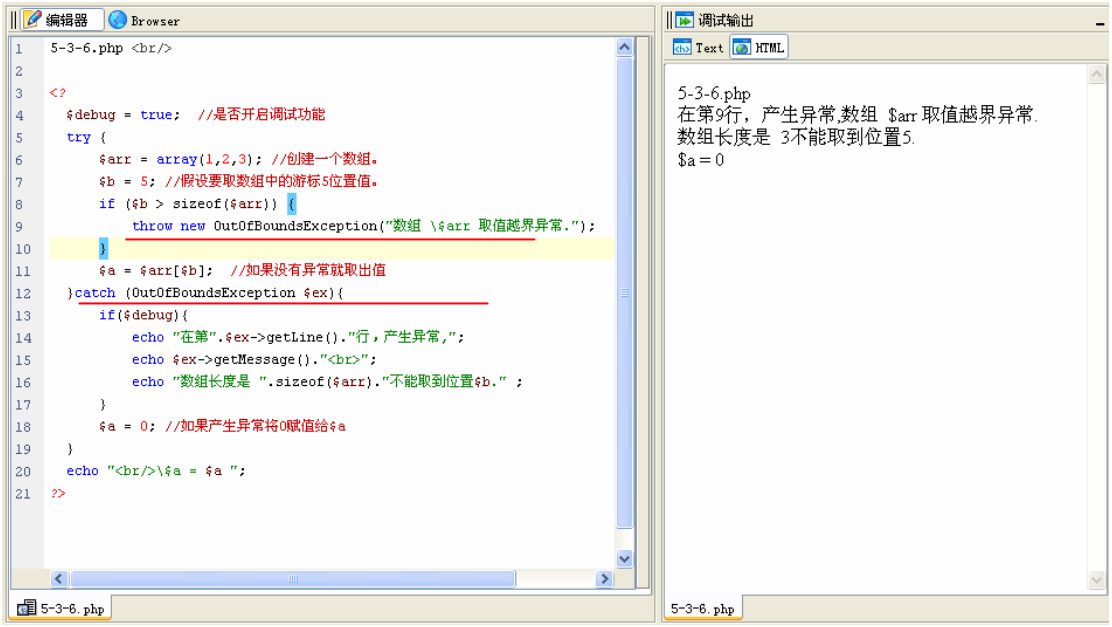
未使用异常处理时，我们从一个数组中取值，如果数组越界，PHP 只会报出一个错误 Notice，我们无法对这些错误做任何的处理。

例 5-3-5.php



在下面例子中，取值超过了数组边界，于是引发了自定义异常。在 catch 块做了处理。

例：5-3-6.php

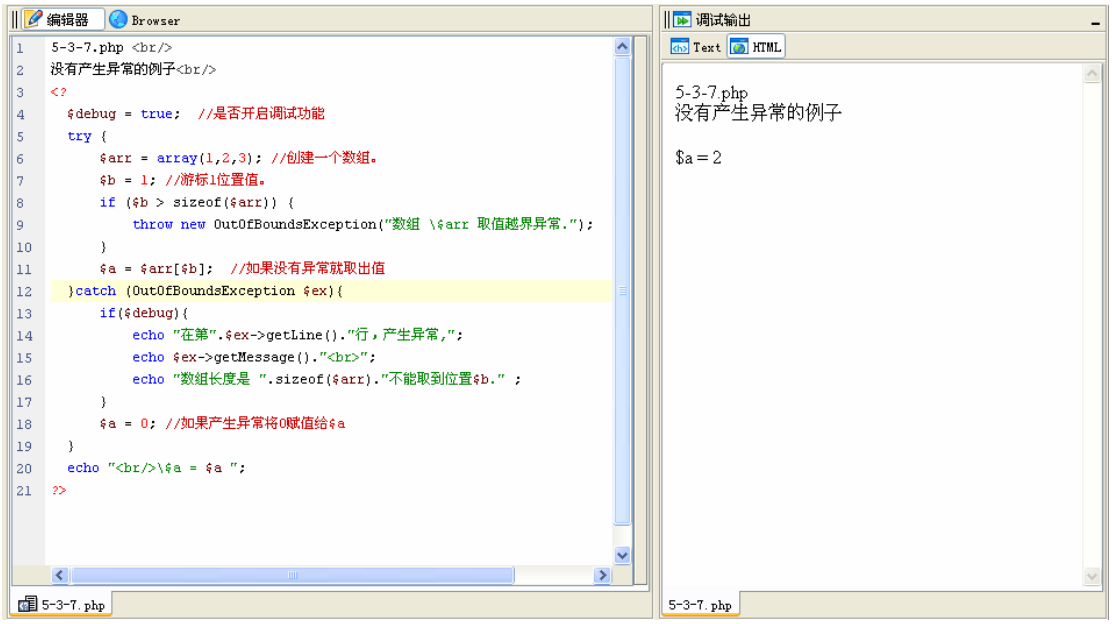


```
5-3-6.php <br/>

<?
  $debug = true;  //是否开启调试功能
  try {
      $arr = array(1,2,3); //创建一个数组。
      $b = 5; //假设要取数组中的游标5位置值。
      if ($b > sizeof($arr)) {
          throw new OutOfBoundsException("数组 \$arr 取值越界异常.");
      }
      $a = $arr[$b];  //如果没有异常就取出值
  }catch (OutOfBoundsException $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常,";
          echo $ex->getMessage()."<br>";
          echo "数组长度是 ".sizeof($arr)."不能取到位置$b." ;
      }
      $a = 0; //如果产生异常将0赋值给$a
  }
  echo "<br/>\$a = $a ";
?>
```

```
5-3-6.php
在第9行，产生异常,数组 $arr 取值越界异常.
数组长度是 3不能取到位置5.
$a = 0
```

取值没有引发异常的代码。

例 5-3-7.php



```
5-3-7.php <br/>
没有产生异常的例子<br/>
<?
  $debug = true;  //是否开启调试功能
  try {
      $arr = array(1,2,3); //创建一个数组。
      $b = 1; //游标1位置值。
      if ($b > sizeof($arr)) {
          throw new OutOfBoundsException("数组 \$arr 取值越界异常.");
      }
      $a = $arr[$b];  //如果没有异常就取出值
  }catch (OutOfBoundsException $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常,";
          echo $ex->getMessage()."<br>";
          echo "数组长度是 ".sizeof($arr)."不能取到位置$b." ;
      }
      $a = 0; //如果产生异常将0赋值给$a
  }
  echo "<br/>\$a = $a ";
?>
```

```
5-3-7.php
没有产生异常的例子

$a = 2
```

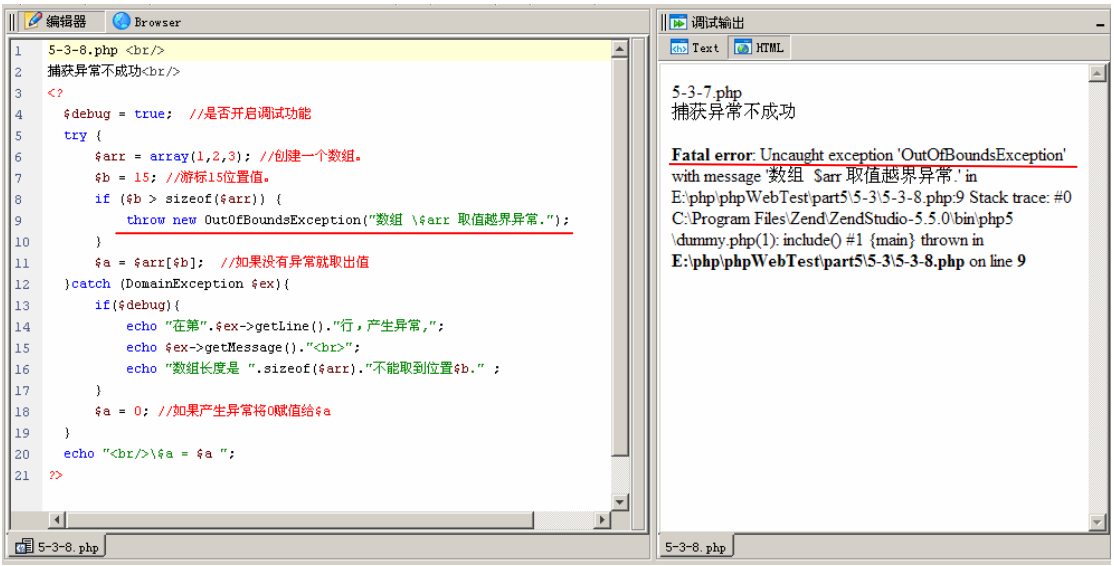
5.3.4 在代码中捕获异常(2)

大家注意到 catch(Exception $ex) 中 Exception 这个类名，下面解释它的具体意义。

- 在 catch 块中能捕获在 catch()块中声明的捕获的异常和其子类类型实例。

下面的例子，抛出一个 OutOfBoundsException 的异常，而 catch 语句捕获 DomainException 异常。这个异常不会被 catch 语句捕获，而直接抛向了系统，引发了一个致命错误，程序被终止了。

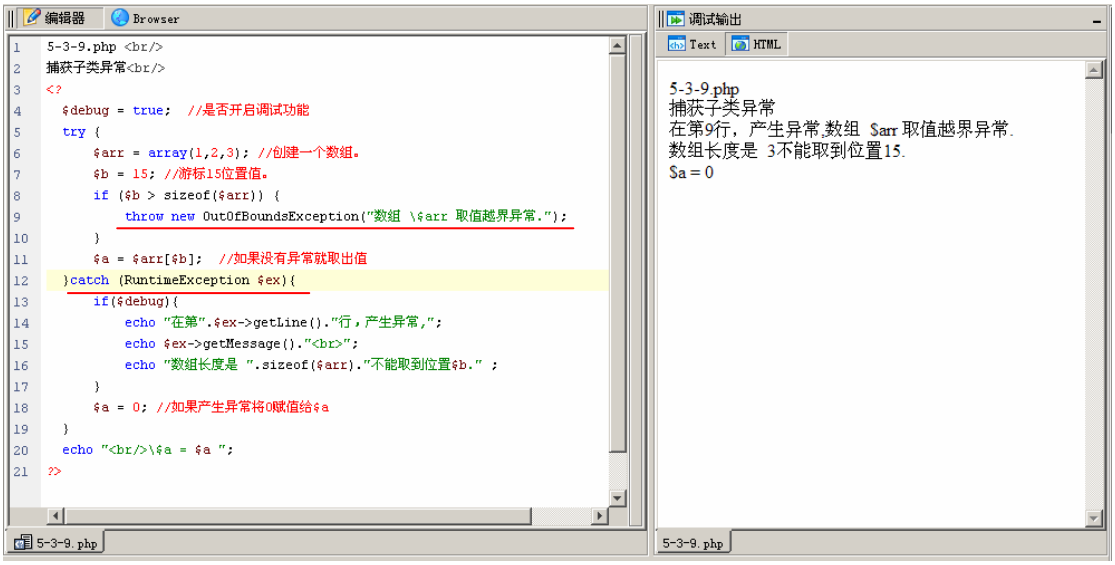
例 5-3-8.php



- 在 catch 块中能捕获 catch()块里声明的异常的子类异常。

下面的例子抛出了 OutOfBoundsException 异常，由它的父类 RuntimeException 捕获。

例 5-3-9.php

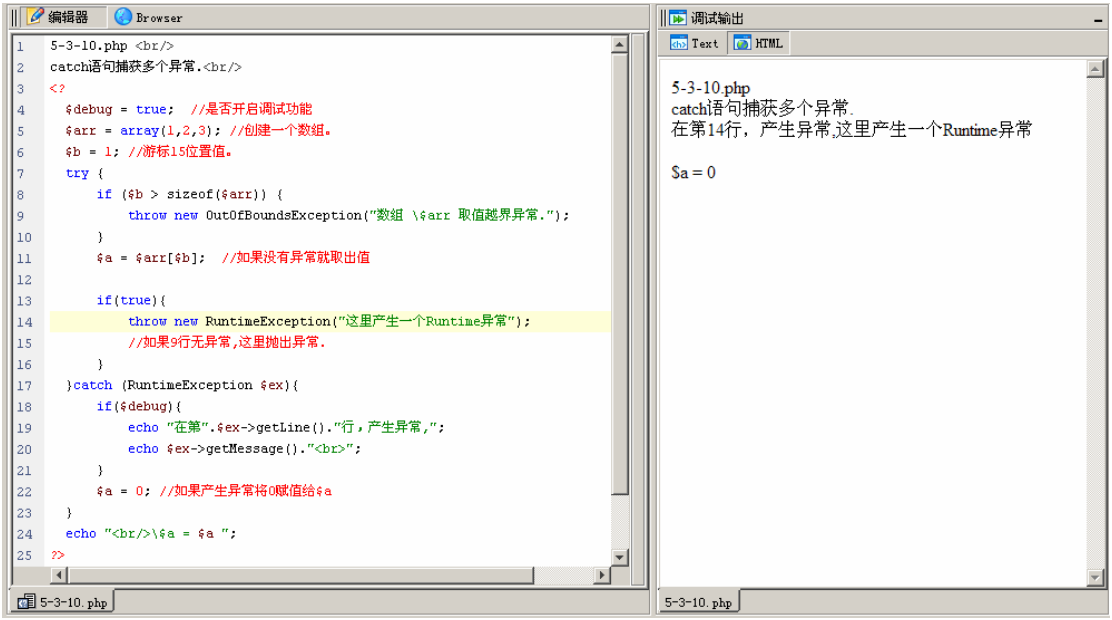


5.3.5 一个 catch 块处理多种异常

- 在 catch 块中能捕获在 catch()块中声明的捕获的异常和其子类类型实例。

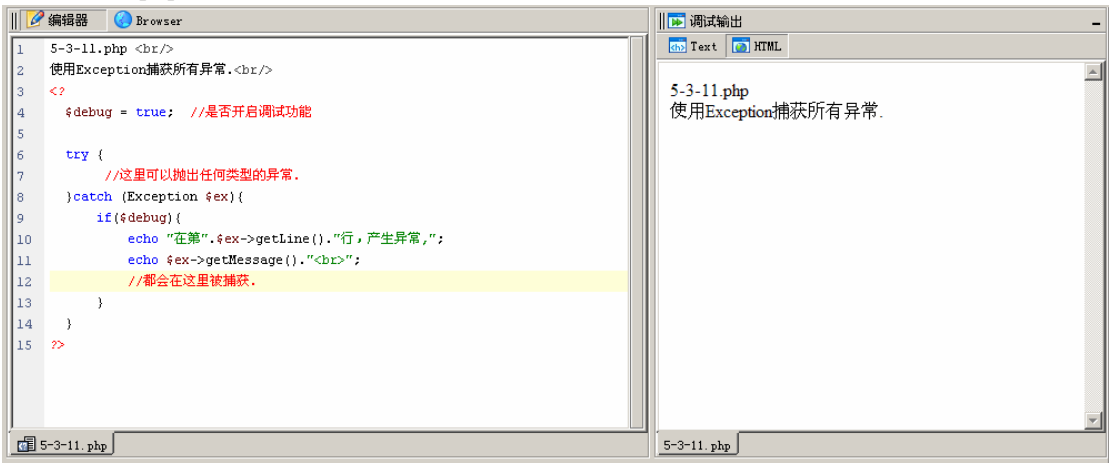
在下面的代码中，catch 块即可以捕获第 9 行的异常，也可以捕获第 14 行的异常。实现了一个异常处理块捕获多种异常。

例 5-3-10.php



Exception 是所有异常类的父类，catch(Exception $ex)可以捕获 try 块中的任何异常。

例：5-3-11.php



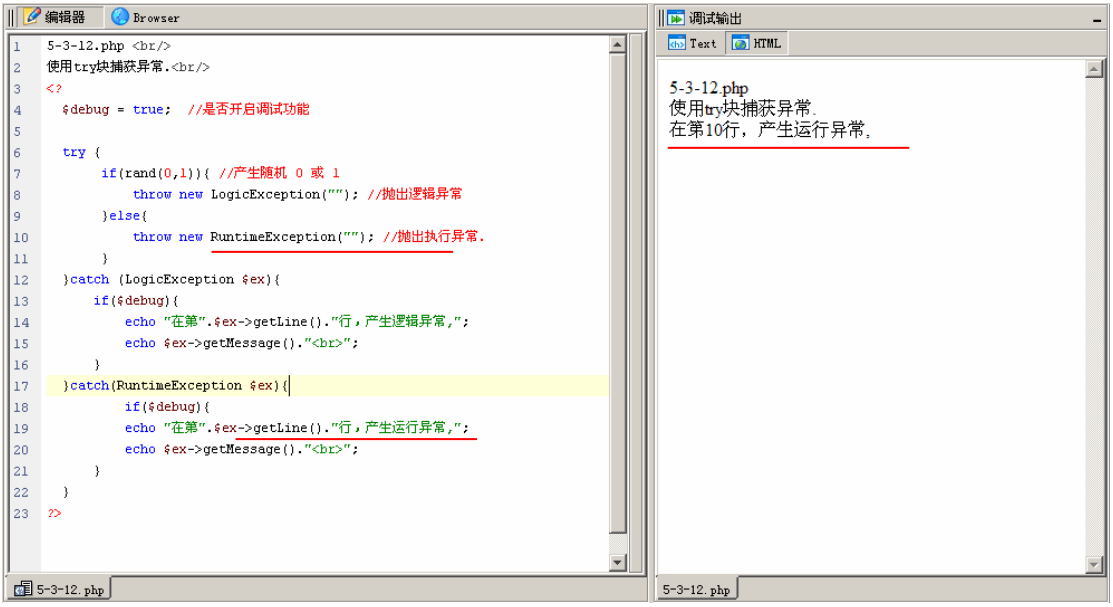
5.3.6 多个 catch 块处理异常

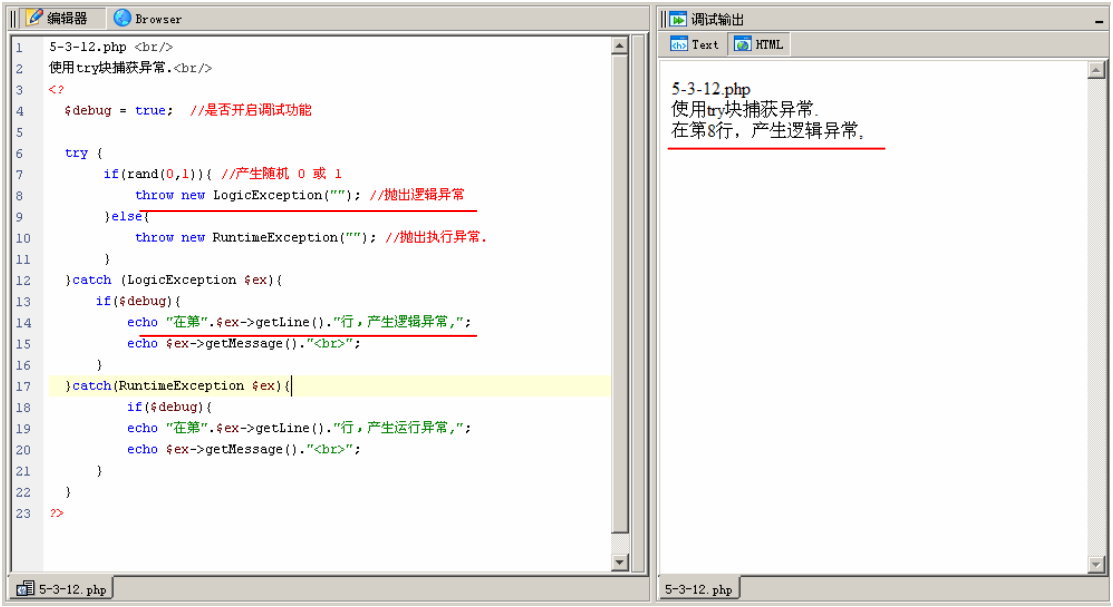
一个 try 块可以跟随多个 catch 块，每个 catch 块捕获不同的异常。

下面例子的第 7 行，使用 rand 函数产生了一个 0 或 1 的随机数，反复运行这个代码会随机抛出逻辑异常或执行异常。

在 try 块后，有两个 catch 块，分别捕获并处理对应的异常，注意查看不同。

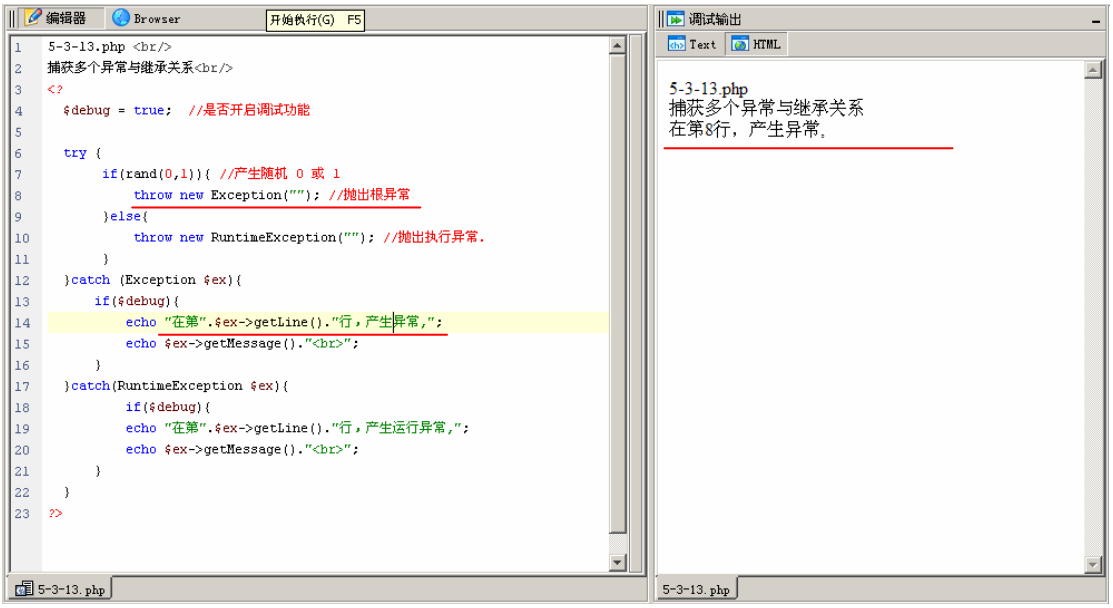
例 5-3-12.php

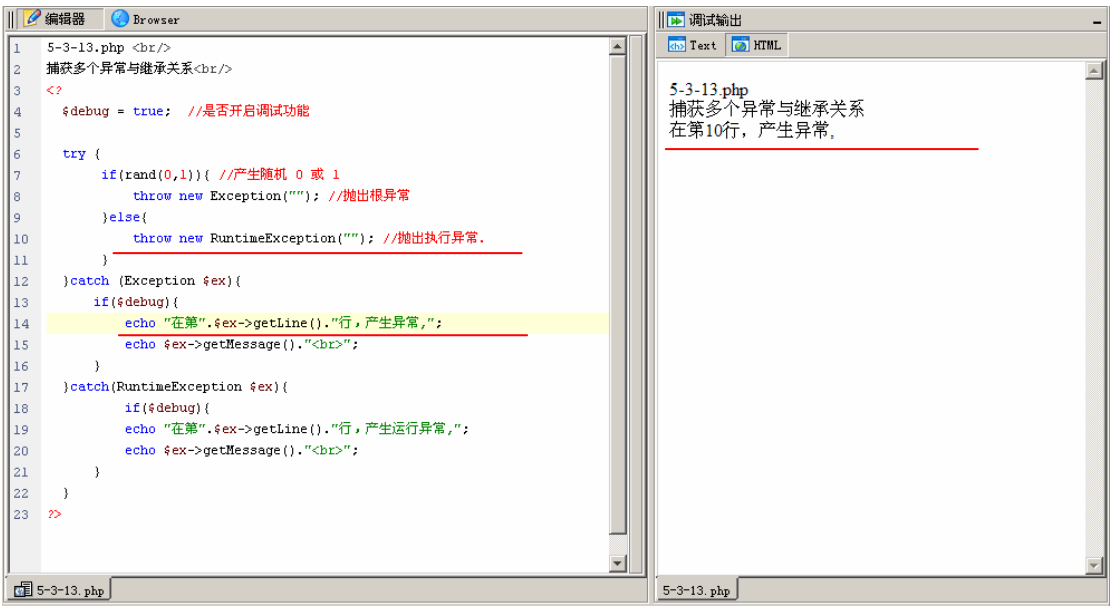




当有异常被抛出，系统从 catch 片段中寻找能处理这个异常的 catch 块，
前面说过一个父类的 catch 块可以捕获子类的异常实例。
在 catch 片段中，如果父类的 catch 块在前面，将屏蔽掉后面对应的子类 catch 块的功能。

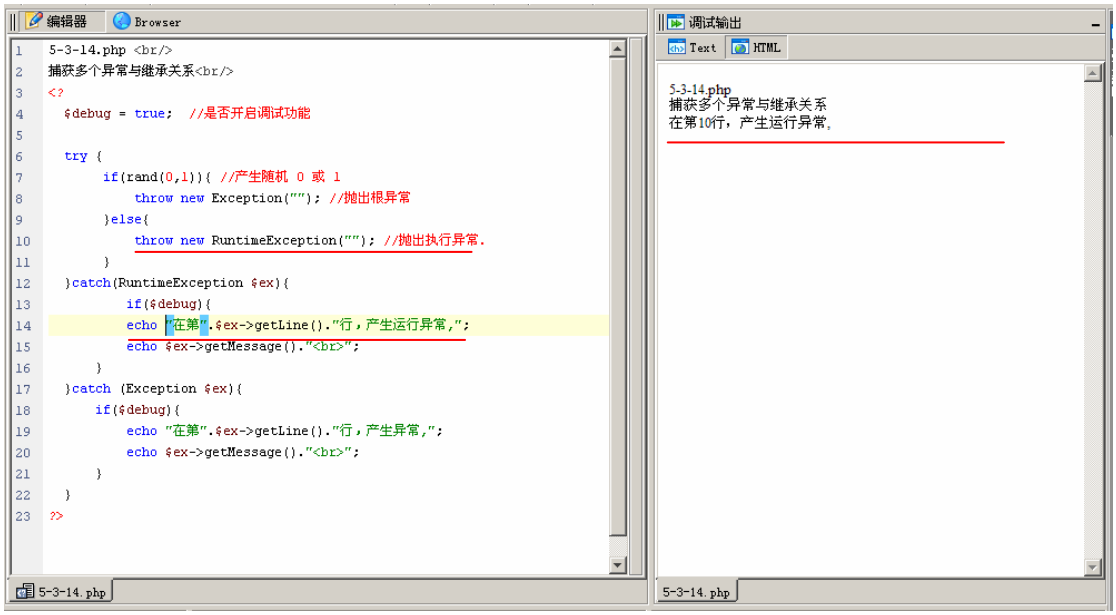
下面例子中引发的两种异常，都被第一个 Exception 类型的 catch 块捕获了。
例 5-3-13.php



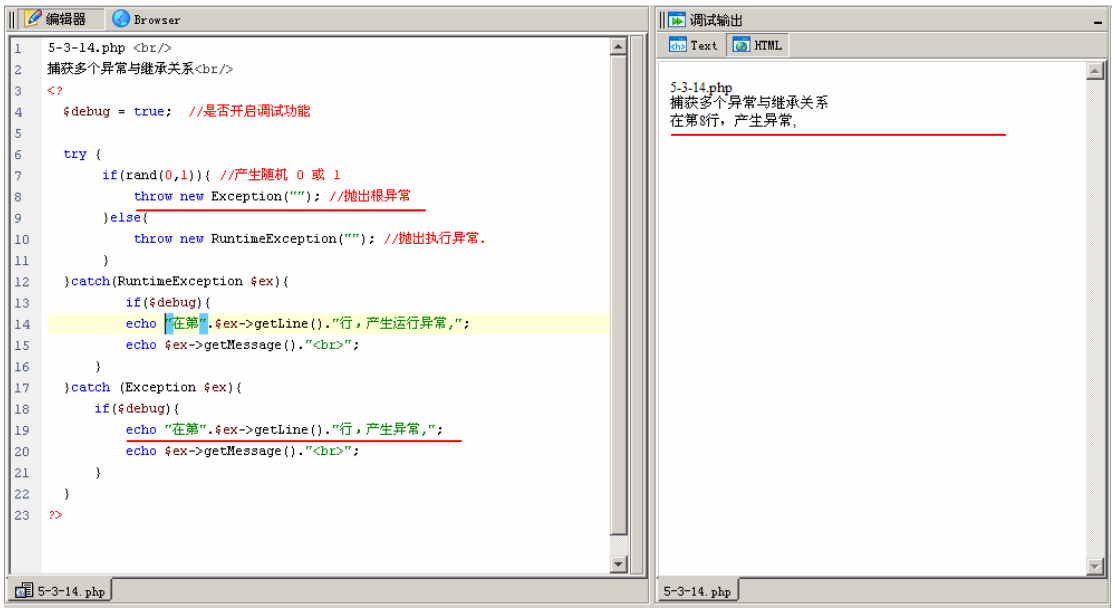


把两个 catch 块的顺序调换，将子类的异常捕获放在前面，父类的异常捕获放在后面。就会看到两个 catch 块能各司其职了。

例：5-3-14.php




```
5-3-14.php <br/>
捕获多个异常与继承关系<br/>
<?
  $debug = true;  //是否开启调试功能

  try {
       if(rand(0,1)){ //产生随机 0 或 1
           throw new Exception(""); //抛出根异常
       }else{
           throw new RuntimeException(""); //抛出执行异常.
       }
  }catch(RuntimeException $ex){
         if($debug){
         echo "在第".$ex->getLine()."行，产生运行异常，";
         echo $ex->getMessage()."<br>";
     }
  }catch (Exception $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常，";
          echo $ex->getMessage()."<br>";
      }
  }
?>
```

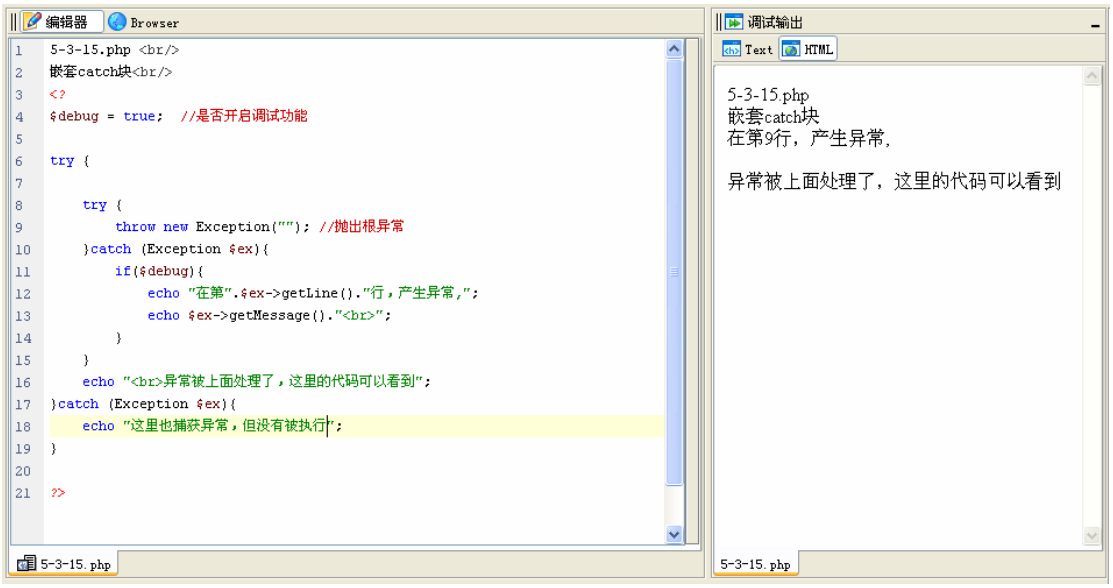



```
5-3-14.php <br/>
捕获多个异常与继承关系<br/>
<?
  $debug = true;  //是否开启调试功能

  try {
       if(rand(0,1)){ //产生随机 0 或 1
           throw new Exception(""); //抛出根异常
       }else{
           throw new RuntimeException(""); //抛出执行异常.
       }
  }catch(RuntimeException $ex){
         if($debug){
         echo "在第".$ex->getLine()."行，产生运行异常，";
         echo $ex->getMessage()."<br>";
     }
  }catch (Exception $ex){
      if($debug){
          echo "在第".$ex->getLine()."行，产生异常，";
          echo $ex->getMessage()."<br>";
      }
  }
?>
```

5.3.7 异常处理块嵌套

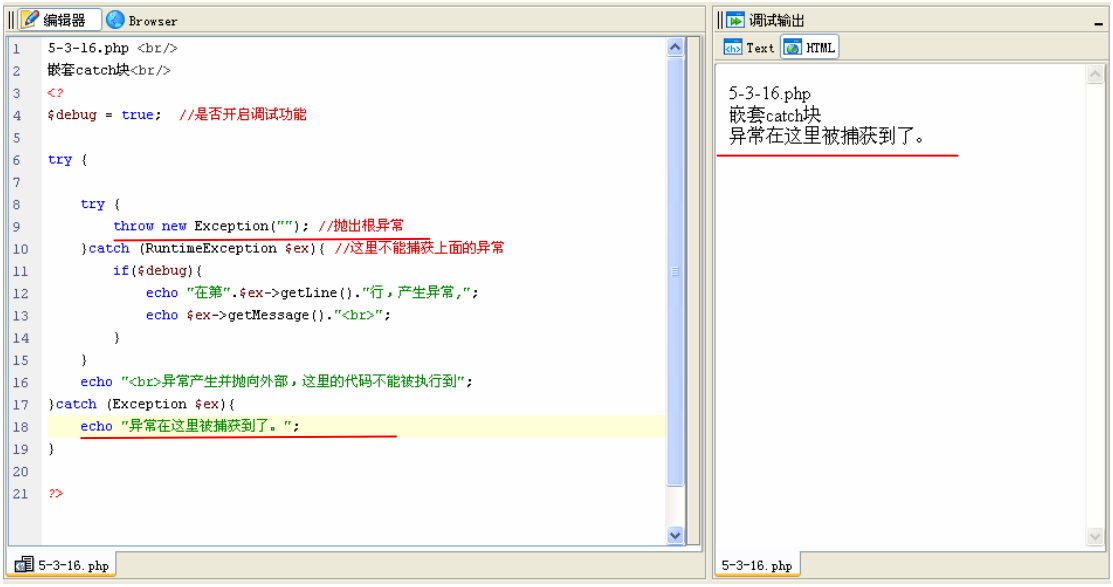
异常处理块只能处理自己 catch 块中的异常，已经处理过的异常将不会向外抛出。

例 5-3-15.php



如果内部块不能捕获异常，这个异常将继续向外抛出。直到能够被捕获为止。

例 5-3-16.php

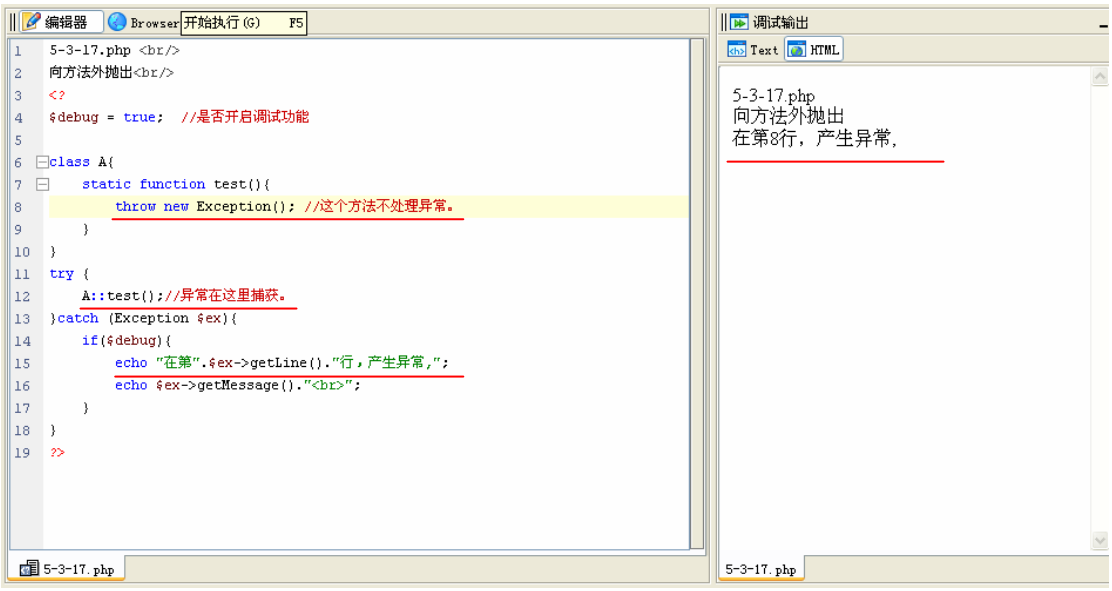


5.3.8 异常向外抛出

代码中一旦 throw 一个异常实例，系统会寻找能够处理这个异常的 try catch 块，如果当前方法不能够处理这个异常，就会向外抛出。抛向调用这个方法的代码，一直向外抛出，如果抛到最外层都无法处理这个异常，会引发致命错误，代码终止。

我们可以在方法引用的任何一个环节，根据业务需求决定捕获异常的位置。

例 5-3-17.php

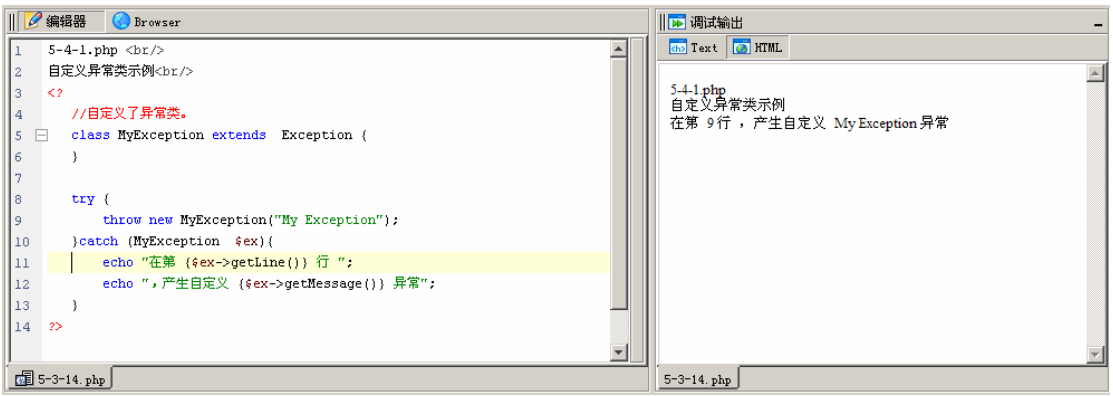


5.4 PHP5 自定义异常

5.4.1 自定义异常

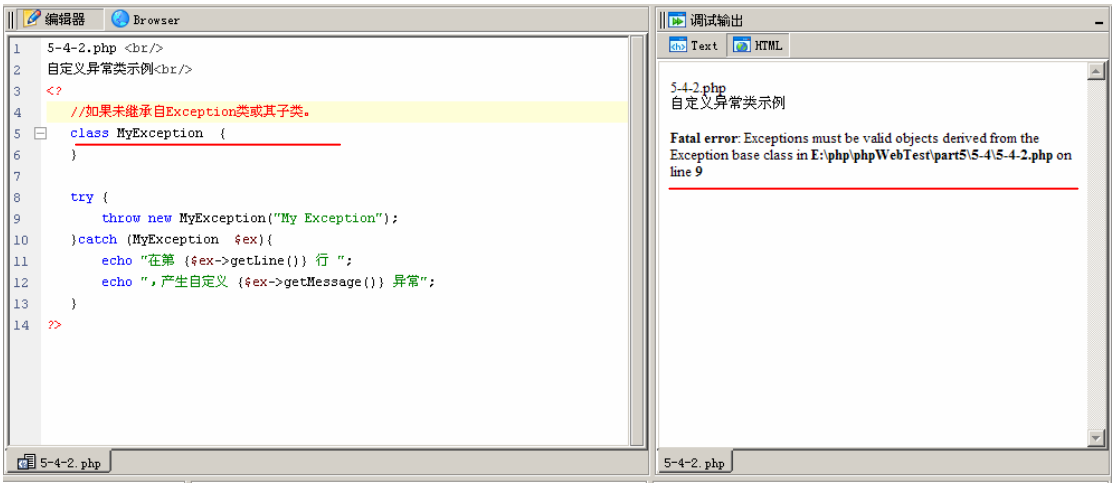
在 PHP5 中可以自定义异常，自定义的异常必须继承自 Exception 类或者它的子类。

例 5-4-1.php



如果自定义的类未继承自 Exception 或者其子类，就会出现不能运行的错误。

例 5-4-2.php



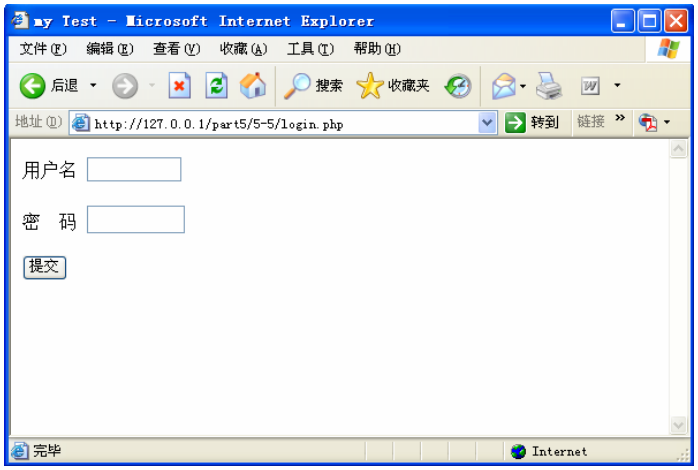
5.5 异常处理实例

5.5.1 验证实例

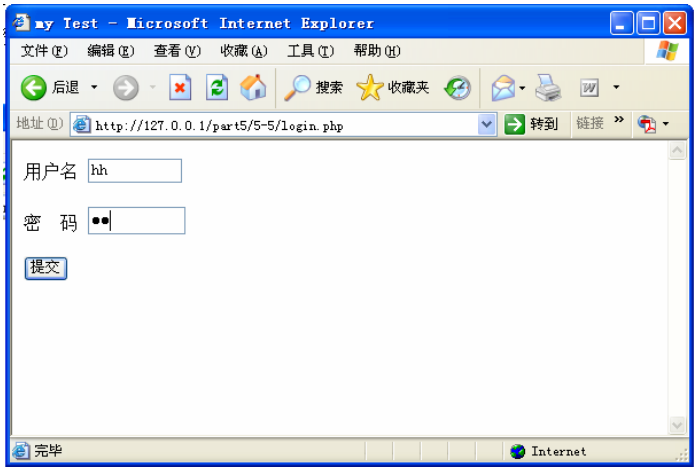
本例将写用户名和密码长度验证并对用户名和密码设定为只能是字符，配合正则表达式完成。

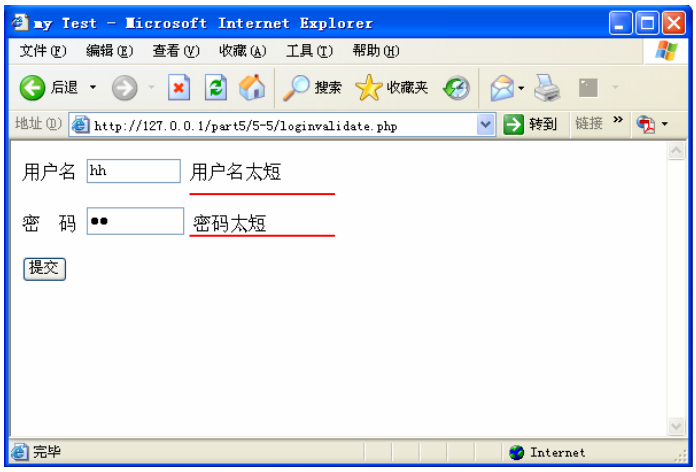
设定用户名和密码长度必须在 3-8 个之间，并且只能是字符与数字

如何与数据库链接，进行数据库验证，就不在本例表达。

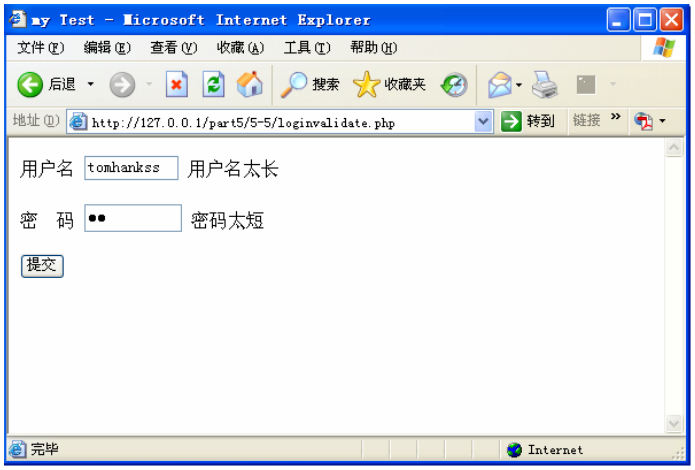


如果用户名和密码长度不够，提交后会提示。

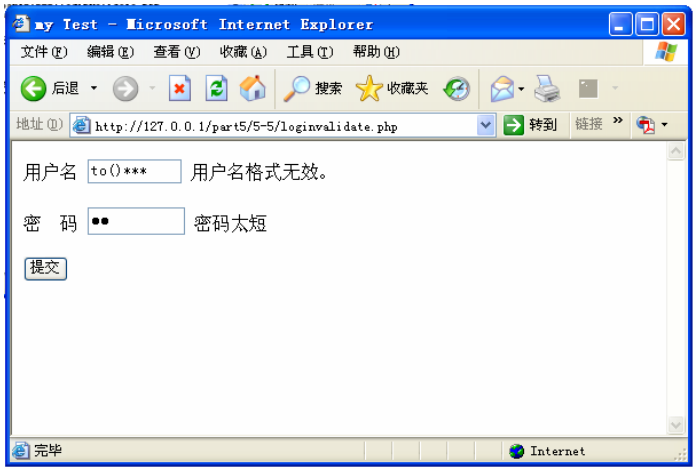




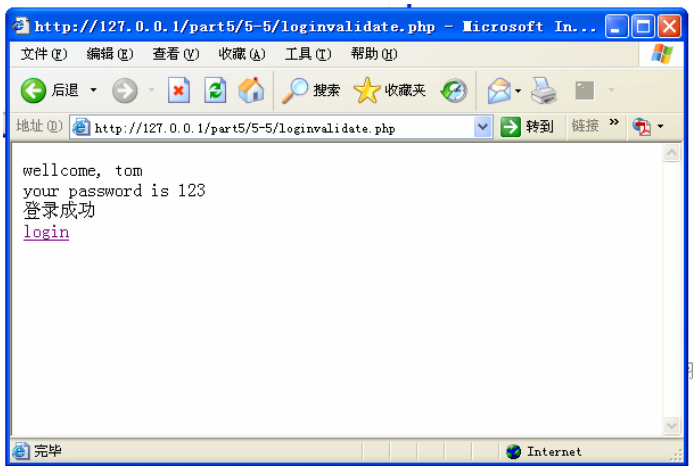
如果过长会提示。



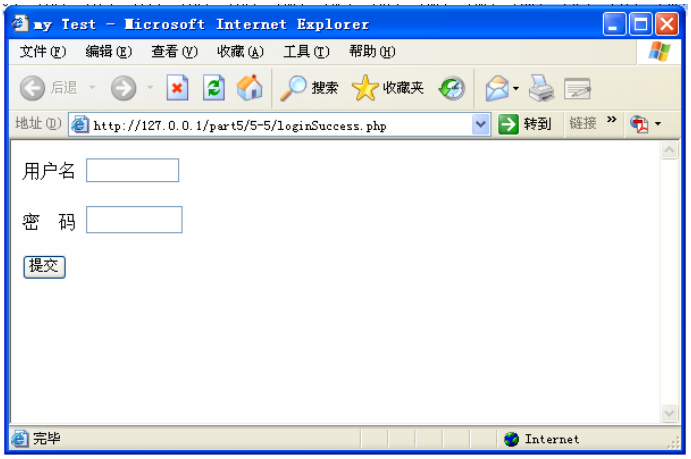
如果输入字符有 （*等符号不合法。



如果验证成功进入登录成功页面
用户输入的用户名和密码可以显示在用户界面。



如果用户直接输入 登录成功 loginSuccess.php 页面，会转到登录页面。



5.5.2 验证实例代码

User.php 实现功能：

- 定义 User 接口。
- 实现存放用户信息的类。
- 通过工厂模式返回用户类实例。

编辑器 Browser

```php
<?
//User.php
//这是一个简单工厂模式的实现,用来获得一个与表单对应的User实例.
interface User{ //定义的接口
    function setUsername($username);
    function getUsername();
    function setPassword($password);
    function getPassword();
}
abstract  class LoginUser implements User { //抽象类
    private $username;
    private $password;
    function setUsername($username){
        $this->username =  $username;
    }
    function getUsername(){
        return $this->username;
    }
    function setPassword($password){
        $this->password =  $password;
    }
    function getPassword(){
        return $this->password;
    }
    //返回子类的实例.
    static function getNormalUser($username,$password){
        $user = new NormalUser();
        $user->setUsername($username);
        $user->setPassword($password);
        return $user;
    }
}

class NormalUser extends LoginUser{
}
?>
```

定义一些参数的类，Properties.php
在这个类中集中定义了一些参数，方便进行后期一些与程序逻辑关系不大的维护。

编辑器 Browser

```php
<?
// Properties.php
//定义了一些参数,集中修改更方便,有些不会在程序中改变的变量可以设置为const类型.
class Properties{
    const User = "User";
    static $title = "my Test";

    static $UsernameLengthMin = 3;
    static $UsernameLengthMax = 8;
    static $username = "username";
    static $validate_Username = "erro_Username";
    static $validate_UsernameInfo = "用户名格式无效。";
    static $validate_UsernameLengthMin = "用户名太短";
    static $validate_UsernameLengthMax = "用户名太长";

    static $PasswordLengthMin = 3;
    static $PasswordLengthMax = 8;
    static $password="password";
    static $validate_Password = "erro_Password";
    static $validate_PasswordInfo="用户格式无效";
    static $validate_PasswordLengthMin="密码太短";
    static $validate_PasswordLengthMax="密码太长";

    static $loginFailes =  "login.php";
    static $loginSuccess = "loginSuccess.php";
}
?>
```

自定义类， MyException.php
自定义了用户名和密码两个独立的异常类，以及其它子类。

编辑器 Browser

```php
<?
//MyException.php
//这里定义了相关异常.为以后的扩展准备好了丰富的异常分类.
class UsernameException extends  Exception {}
class UsernameInvalidateException extends UsernameException {}
class UsernameLengthMinException extends UsernameException {}
class UsernameLengthMaxException extends UsernameException {}

class PasswordException extends Exception {}
class PasswordInvalidateException extends PasswordException {}
class PasswordLengthMinException extends PasswordException  {}
class PasswordLengthMaxException extends PasswordException  {}
?>
```

验证用户名和密码长度合法性的类 Validate.php
在 validateName 方法和 validatePassword 两个方法中，分别对长度做了判断和验证。并通过正则表达式，对用户输入的合法性做了验证。
虽然可以通过正则表达式一次验证用户字符合法性和长度， 本例没有这样写是为了向方法外抛出长度不同的异常，以及显示给用户的信息。

编辑器 Browser

```php
<?
//Validate.php 验证类
include_once("User.php");
include_once("Properties.php");
include_once("MyException.php");

class Validate{

    public static  function validateUser(User $user){

        try{
            self::validateName($user->getUsername()); //用户名验证
        }catch(UsernameException $ex){
            //错误信息放入session.
            $_SESSION[Properties::$validate_Username] = $ex->getMessage();
        }

        try{
            self::validatePassword($user->getPassword());
        }catch(PasswordException  $ex){
            //错误信息放入session.
            $_SESSION[Properties::$validate_Password] =$ex->getMessage();
        }
        if($ex){
            //将用户名和密码放入session
            $_SESSION[Properties::$username] = $user->getUsername();
            $_SESSION[Properties::$password] = $user->getPassword();
            throw $ex; //向外抛出异常实例.
        }//如果没有异常抛出,就天下太平. 没有问题.


    }

    //验证用户名长度和是否符合正则表达式.
    //没有完全使用正则表达式,而先使用了长度判断,是为了提供更确切的错误信息.
    private static function  validateName($username){

        if(strlen($username) < Properties::$UsernameLengthMin){
            throw new UsernameLengthMinException(Properties::$validate_UsernameLengthMin);
        }elseif (strlen($username) > Properties::$UsernameLengthMax){
            throw new UsernameLengthMaxException(Properties::$validate_UsernameLengthMax);
        }elseif(! ereg("^[a-zA-Z0-9]*$",$username)){
            throw new UsernameInvalidateException(Properties::$validate_UsernameInfo);
        }
    }
    //验证密码.
    private static function  validatePassword($password){
        if(strlen($password) < Properties::$PasswordLengthMin){
            throw new PasswordLengthMinException(Properties::$validate_PasswordLengthMin);
        }elseif (strlen($password) > Properties::$UsernameLengthMax){
            throw new PasswordLengthMaxException(Properties::$validate_PasswordLengthMax);
        }elseif(! ereg("^[a-zA-Z0-9]*$",$password)){
            throw new PasswordInvalidateException(Properties::$validate_PasswordInfo);
        }
    }
}
?>
```

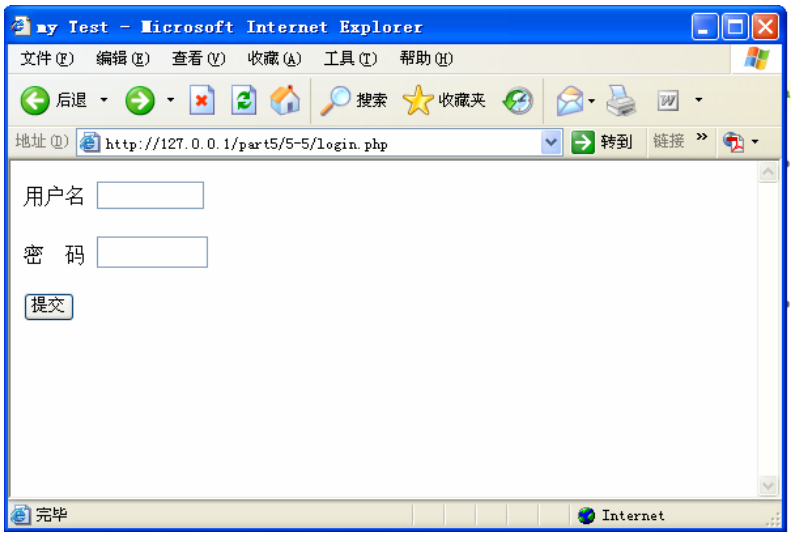
登录页面 login.php。

在第 12 行和第 18 行，分别取出 session 中的用户名密码，以及错误信息。

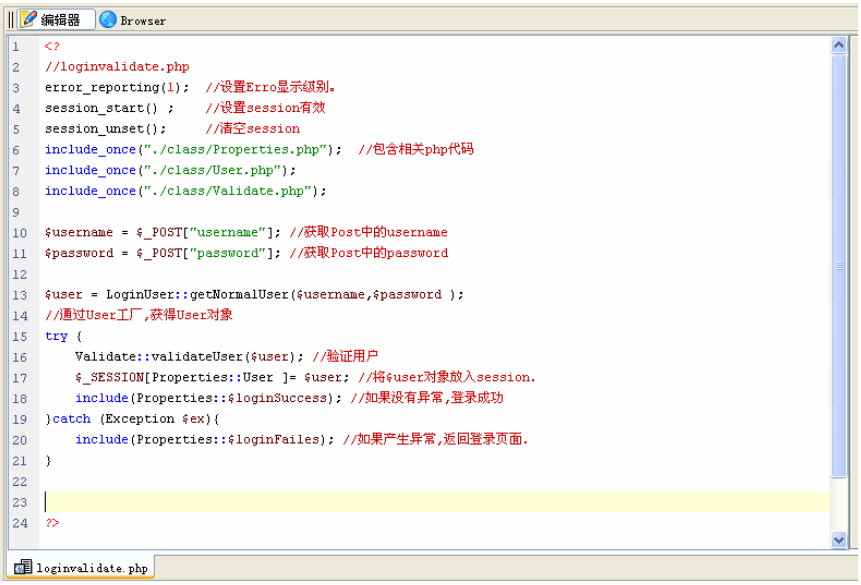
编辑器 Browser

```
1  <? require_once("./class/Properties.php"); ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml">
4  <head>
5  <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
6  <title><?=Properties::$title?></title>
7  </head>
8  <body>
9  <form id="form1" name="form1" method="post" action="loginvalidate.php">
10   <label>用户名
11   <input name="username" type="text"
12 id="username" size="10" maxlength="10" value="<?=$_SESSION[Properties::$username]?>"/>
13   <?=$_SESSION[Properties::$validate_Username] ?>
14   </label>
15   <p>
16     <label>密&amp;nbsp;&amp;nbsp;码
17     <input name="password" type="password"
18 id="password" size="10" maxlength="10" value="<?=$_SESSION[Properties::$password] ?>" />
19      <?=$_SESSION[Properties::$validate_Password] ?>
20     </label>
21     <br />
22     <br />
23     <label>
24     <input type="submit" name="Submit" value="提交" />
25     </label>
26   </p>
27 </form>
28 </body>
```

界面如下

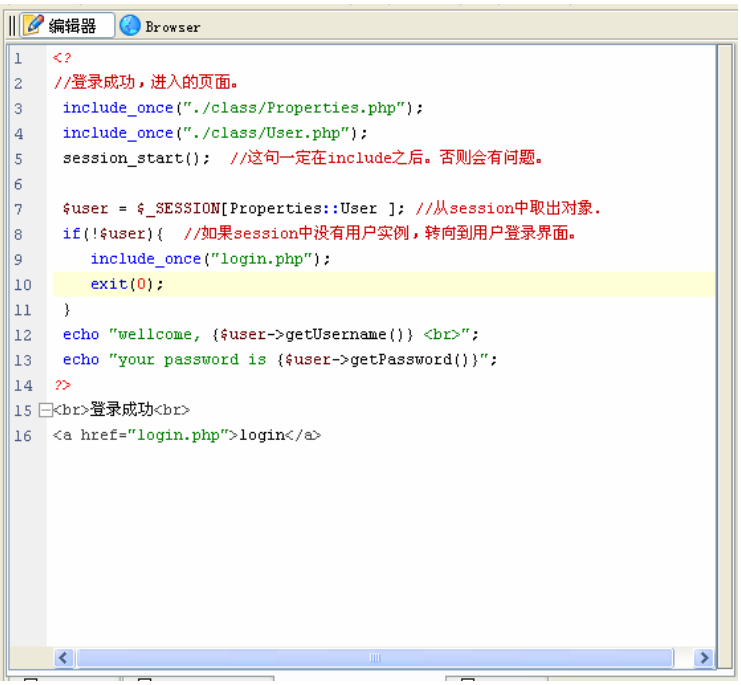


登录会提交给 loginvalidate.php
这段代码相当于 MVC 中的 controller 控制器。



```php
<?
//loginvalidate.php
error_reporting(1);  //设置Erro显示级别。
session_start() ;    //设置session有效
session_unset();     //清空session
include_once("./class/Properties.php");  //包含相关php代码
include_once("./class/User.php");
include_once("./class/Validate.php");

$username = $_POST["username"]; //获取Post中的username
$password = $_POST["password"]; //获取Post中的password

$user = LoginUser::getNormalUser($username,$password );
//通过User工厂,获得User对象
try {
    Validate::validateUser($user); //验证用户
    $_SESSION[Properties::User ]= $user; //将$user对象放入session.
    include(Properties::$loginSuccess); //如果没有异常,登录成功
}catch (Exception $ex){
    include(Properties::$loginFailes); //如果产生异常,返回登录页面.
}


?>
```

登录成功页面，loginSuccess.php
显示用户登录信息，判断 session。



```php
<?
//登录成功，进入的页面。
 include_once("./class/Properties.php");
 include_once("./class/User.php");
 session_start();  //这句一定在include之后。否则会有问题。

 $user = $_SESSION[Properties::User ]; //从session中取出对象.
 if(!$user){  //如果session中没有用户实例，转向到用户登录界面。
    include_once("login.php");
    exit(0);
 }
 echo "wellcome, {$user->getUsername()} <br>";
 echo "your password is {$user->getPassword()}";
?>
<br>登录成功<br>
<a href="login.php">login</a>
```

5.6 小结