7

Обработка ошибок и исключений

В этой главе рассмотрены концепции обработки исключений и описан способ реализации этой обработки в PHP. Исключения — это унифицированный механизм расширяемой, удобной для обслуживания и объектно-ориентированной обработки ошибок.

В главе рассматриваются следующие темы.

- Концепции обработки исключений.
- Структуры управления исключениями: `try...throw...catch`.
- Класс `Exception`.
- Исключения, определяемые пользователем.
- Исключения в приложении "Автозапчасти от Вована".
- Исключения и другие механизмы обработки ошибок PHP.

Концепции обработки ошибок

Основная идея обработки ошибок состоит в выполнении кода внутри так называемого *блока* `try`. Этот блок представляет собой раздел кода следующего вида:

```
try {
  // здесь находится необходимый код
}
```

В случае возникновения непредвиденных ситуаций внутри блока `try` можно выполнить так называемую *генерацию* (throwing) *исключения*. Некоторые языки, такие как Java, в определенных случаях генерируют исключения автоматически. В PHP исключения нужно генерировать вручную. Это выполняется следующим образом:

```
throw new Exception('сообщение', код);
```

Ключевое слово `throw` запускает механизм обработки исключений. Оно представляет собой конструкцию языка, а не функцию, но ему необходимо передать значение. Эта конструкция ожидает получения объекта. В простейшем случае ее можно использовать для создания экземпляра встроенного класса `Exception`, как и было сделано в приведенном примере.

Конструктор этого класса принимает два параметра: сообщение и код. Они служат для представления сообщения об ошибке и номера ошибки. Оба эти параметра необязательны.

И, наконец, за блоком `try` должен следовать как минимум один блок `catch`, который выглядит так:

```
catch (указание_типа исключение) {
  // обработка исключения
}
```

С одним блоком `try` может быть связано несколько блоков `catch`. Использование нескольких блоков `catch` имеет смысл, если каждый из них ожидает перехвата отдельного типа исключения. Например, если требуется перехватывать исключения класса `Exception`, блок `catch` может выглядеть следующим образом:

```
catch (Exception $e) {
  // обработка исключения
}
```

Объект, передаваемый в блок `catch` (и перехватываемый им), является тем, который передается (и генерируется) оператором `throw`, генерирующим исключение. Исключение может быть любого типа, однако удобнее всего использовать либо класс `Exception`, либо экземпляры собственных пользовательских исключений, унаследованных от класса `Exception`. (Определение пользовательских исключений рассматривается далее в этой главе.)

В случае возникновения исключения код PHP ищет соответствующий блок `catch`. При наличии более одного блока `catch` передаваемые в них объекты должны иметь различные типы, чтобы PHP мог определить, какой именно блок `catch` соответствует конкретному случаю.

И еще один момент: внутри блока `catch` тоже можно генерировать исключения.

Для большей наглядности рассмотрим пример. Простой пример обработки исключения показан в листинге 7.1.

Листинг 7.1. `basic_exception.php` — генерирование и перехват исключения

```
<?php

try {
  throw new Exception("Возникла очень серьезная ошибка", 42);
}
catch (Exception $e) {
  echo "Исключение ".$e->getCode().": ".$e->getMessage()."<br />".
       " в ".$e->getFile()."", строка ".$e->getLine()."<br />";
}

?>
```

В листинге 7.1 видно, что мы воспользовались несколькими методами класса `Exception`, которые будут рассмотрены несколько позже. Результат выполнения этого кода показан на рис. 7.1.

В приведенном примере кода было сгенерировано исключение класса `Exception`. Методы этого встроенного класса можно использовать в блоке `catch` для вывода полезного сообщения об ошибке.

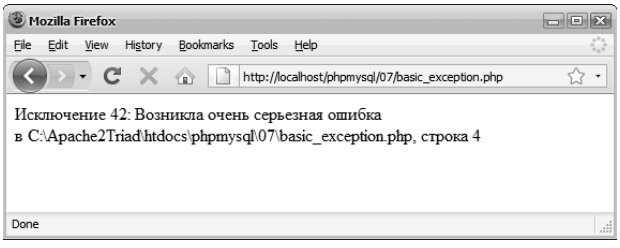

Рис. 7.1. Этот блок `catch` выводит сообщение об ошибке с указанием места ее возникновения

Класс `Exception`

В PHP имеется встроенный класс `Exception`. Как уже упоминалось, конструктор этого класса принимает два параметра: сообщение об ошибке и номер ошибки.

Кроме конструктора, этот класс содержит следующие встроенные методы.

- `getCode()` — возвращает код, переданный конструктору.
- `getMessage()` — возвращает сообщение, переданное конструктору.
- `getFile()` — возвращает полный путь файла кода, в котором возникло исключение.
- `getLine()` — возвращает номер строки в файле кода, где возникло исключение.
- `getTrace()` — возвращает массив, содержащий стек вызовов в месте возникновения исключения.
- `getTraceAsString()` — возвращает ту же информацию, что и метод `getTrace`, но сформатированную в виде строки.
- `__toString()` — позволяет упростить вывод объекта `Exception` с помощью оператора `echo`, предоставляя всю информацию, полученную из перечисленных методов.

Как видите, в коде листинга 7.1 были использованы четыре из этих методов. Эту же информацию (плюс стек вызовов) можно было бы получить с помощью следующего оператора:

```
echo $e;
```

Стек вызовов (backtrace) указывает, какие функции выполнялись в момент возникновения исключения.

Исключения, определяемые пользователем

Вместо создания и передачи экземпляра базового класса `Exception` можно передавать любой другой объект. В большинстве случаев вы будете расширять класс `Exception` для создания своих собственных классов исключений.

Конструкция `throw` позволяет передавать любые другие объекты. Иногда такая потребность может возникать при наличии проблем, связанных с каким-то конкретным объектом, и необходимости его передачи в целях отладки.

Но, как уже говорилось, в большинстве случаев приходится расширять базовый класс `Exception`. В руководстве по PHP показан код, который демонстрирует скелет класса `Exception`. Этот код, доступный по адресу `http://www.php.net/zend-engine-2.php`, воспроизведен в листинге 7.2. Обратите внимание, что это не весь код, а лишь те компоненты, которые можно наследовать.

Листинг 7.2. Класс `Exception` — компоненты класса, которые можно наследовать

```
<?php
class Exception {
   function __construct(string $message=NULL, int $code=0) {
      if (func_num_args()) {
         $this->message = $message;
      }
      $this->code = $code;
      $this->file = __FILE__;                  // из конструкции throw
      $this->line = __LINE__;                  // из конструкции throw
      $this->trace = debug_backtrace();
      $this->string = StringFormat($this);
   }

   protected $message = 'Unknown exception';   // сообщение исключения
   protected $code = 0;                        // пользовательский код исключения
   protected $file;                            // исходное имя файла исключения
   protected $line;                            // исходная строка исключения

   private $trace;                             // стек вызовов исключения
   private $string;                            // только для внутреннего пользования!!

   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) {
      // ... недоступная в PHP-сценариях функция,
      // которая возвращает всю необходимую информацию в виде строки
   }
   static private function TraceFormat(Exception $exception) {
      // ... недоступная в PHP-сценариях функция,
      // которая возвращает весь стек вызовов в виде строки
   }
}
? >
```

Основная причина, по которой было приведено определение этого класса, состоит в том, что большинство общедоступных методов являются финальными: т.е. их нельзя переопределить. Можно создать собственный подкласс `Exceptions`, но нельзя менять поведение базовых методов. Однако можно переопределять функцию `__toString()`, чтобы изменять способ отображения исключения. Можно также добавлять собственные методы.

Пример определенного пользователем класса `Exception` показан в листинге 7.3.

Листинг 7.3. `user_defined_exception.php` — пример определяемого пользователем класса `Exception`

```
<?php

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

try {
  throw new myException("Произошла очень серьезная ошибка", 42);
}

catch (myException $m) {
   echo $m;
}

?>
```

В этом коде объявляется новый класс исключения с именем `myException`, расширяющий базовый класс `Exception`. Различие между этим классом и классом `Exception` состоит в переопределении метода `__toString()` для обеспечения более "изящного" вывода сообщения об исключении. Результат выполнения этого кода показан на рис. 7.2.

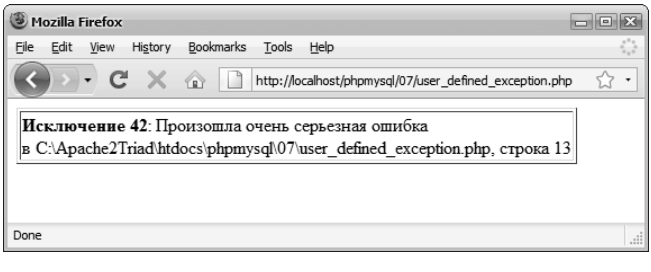

Рис. 7.2. Класс `myException` обеспечивает "изящный" вывод сообщений об исключениях

Приведенный пример очень прост. В следующем разделе мы рассмотрим способы создания различных исключений, связанных с различными категориями ошибок.

Исключения в приложении “Автозапчасти от Вована”

В главе 2 было описано, как данные заказа Вована можно сохранять в плоском файле. Известно, что файловый ввод-вывод (фактически, любой вид ввода-вывода) — это та область программ, в которой часто возникают ошибки. Поэтому имеет смысл применить к нему механизм обработки исключений.

Если вернуться к исходному коду, несложно заметить, что в процессе записи в файл могут возникнуть три проблемы: невозможность открытия файла, невозможность получения блокировки или невозможность записи в файл. Для каждой из этих ситуаций мы создали класс исключения. Код этих классов исключений представлен в листинге 7.4.

Листинг 7.4. `file_exceptions.php` — исключения, связанные с файловым вводом-выводом

```
<?php

class fileOpenException extends Exception {
  function __toString() {
    return "fileOpenException ".$this->getCode().": ".
          $this->getMessage()."<br />в ".$this->getFile().
          ", строка ".$this->getLine()."<br />";
   }
}

class fileWriteException extends Exception {
  function __toString() {
    return "fileWriteException ".$this->getCode().":".
          $this->getMessage()."<br />в ".$this->getFile().
          ", строка ".$this->getLine()."<br />";
   }
}

class fileLockException extends Exception {
  function __toString() {
    return "fileLockException ".$this->getCode().":".
          $this->getMessage()."<br />в ".$this->getFile().
          ", строка ".$this->getLine()."<br />";
   }
}

?>
```

Эти подклассы `Exception` не выполняют никаких действий, представляющих особый интерес. Фактически в этом приложении можно было бы оставить их пустыми или воспользоваться базовым классом `Exception`. Тем не менее, мы включили в каждый подкласс метод `__toString()`, который выводит сообщение о типе возникшего исключения.

Чтобы внедрить обработку исключений, мы переписали файл `processorder.php`, рассмотренный в главе 2. Новая версия этого файла показана в листинге 7.5.

Листинг 7.5. `processorder.php` — сценарий обработки заказов Вована с добавленной обработкой исключений

```
<?php
  require_once("file_exceptions.php");

  // создание коротких имен переменных
  $tireqty = $_POST['tireqty'];
  $oilqty = $_POST['oilqty'];
  $sparkqty = $_POST['sparkqty'];
  $address = $_POST['address'];

  $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>

<html>
<head>
  <title>Автозапчасти от Вована - Результаты заказа</title>
</head>
<body>
<h1>Автозапчасти от Вована</h1>
<h2>Результаты заказа</h2>

<?php

$date = date('H:i, jS F');
echo "<p>Заказ обработан в ".$date."</p>";

$totalqty = $tireqty + $oilqty + $sparkqty;
echo "Заказано товаров: ".$totalqty."<br />";

if($totalqty == 0) {
  echo "Вы ничего не заказали на предыдущей странице!<br />";
} else {
  if ($tireqty>0) {
    echo $tireqty." покрышек<br />";
  }
  if ($oilqty>0) {
    echo $oilqty." бутылок масла<br />";
  }
  if ( $sparkqty>0 ) {
    echo $sparkqty." свечей зажигания<br />";
  }
}

define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);
$totalamount = $tireqty * TIREPRICE
             + $oilqty * OILPRICE
             + $sparkqty * SPARKPRICE;
$totalamount = number_format($totalamount, 2, '.', ' ');

echo "<p>Итого по заказу: ".$totalamount."</p>";
echo "<p>Адрес доставки: ".$address."</p>";
```

```
$outputstring = $date."\t".$tireqty." покрышек\t"
                .$oilqty." бутылок масла\t".$sparkqty." свечей зажигания\t\$"
                .$totalamount."\t".$address."\n";

// открываем файл для дозаписи
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>Заказ записан.</p>";
}

catch (fileOpenException $foe) {
  echo "<p><strong>Невозможно открыть файл заказов."
      ." Обратитесь к Web-мастеру.</strong></p>";
}

catch (Exception $e) {
  echo "<p><strong>В данный момент мы не можем обработать ваш заказ."
      ." Попробуйте повторить его позже.</strong></p>";
}

?>

</body>
</html>
```

Как видите, раздел кода сценария, реализующий файловый ввод-вывод, помещен в блок `try`. В общем случае использование небольших блоков `try`, в конце которых выполняется перехват важных исключений, считается правильным подходом к программированию. Это упрощает написание и сопровождение кода обработки исключений, поскольку легко понять, с чем приходится иметь дело.

В случае невозможности открытия файла код генерирует исключение `fileOpenException`; невозможность блокирования файла ведет к генерированию исключения `fileLockException`, а невозможность записи в файл — к генерированию исключения `fileWriteException`.

Взгляните на блоки `catch`. Для целей иллюстрации мы показали только два таких блока: один для обработки объектов `fileOpenException` и второй для обработки объектов `Exception`. Поскольку остальные исключения наследуют свойства и методы класса `Exception`, они будут перехватываться вторым блоком `catch`. Сопоставление блоков `catch` выполняется в соответствии с теми же основными правилами, что и применяемые в операторе `instanceof`. Это обстоятельство является веским основанием для создания пользовательских классов исключений за счет расширения одного единственного класса.

Важное предупреждение: при генерировании исключения, для которого соответствующий блок `catch` не был создан, PHP сообщит о фатальной ошибке.

Исключения и другие механизмы обработки ошибок PHP

Помимо механизма обработки исключений, рассмотренного в этой главе, PHP обладает развитой поддержкой обработки ошибок, которая будет описана в главе 26. Процесс генерирования и обработки исключений не влияет и не мешает работе данного механизма обработки ошибок.

Обратите внимание, что в листинге 7.5 вызов метода `fopen()` предварен символом операции подавления ошибки (`@`). В случае неудачного выполнения этого метода PHP сгенерирует предупреждение, вывод которого на экран или запись в журнал зависит от настроек отчета об ошибках, которые определены в файле `php.ini`. Эти параметры рассматриваются в главе 26, но следует знать, что данное предупреждение будет сгенерировано независимо от того, выполняется ли генерирование исключения.

Дополнительные источники информации

Поскольку обработка исключений появилась в PHP относительно недавно, по этой теме написано не очень много. Однако существует достаточно большой объем общей информации по обработке исключений. Компания Sun на своей странице `http://java.sun.com/docs/books/tutorial/essential/exceptions/handling.html` предлагает хорошее учебное пособие по вопросу о том, что собой представляют исключения и почему возникает необходимость в их использовании (разумеется, представленная в пособии информация относится к языку Java).

Что дальше

Следующая часть этой книги посвящена MySQL. В ней поясняются создание и заполнение базы данных MySQL, а также технология ее связывания с PHP-сценариями, что дает возможность получать доступ к базе данных из Интернета.