高等教育应用型人才培养系列教材
——科瑞国际校企合作指定教材

Java 程序设计教程（进阶篇）

杨　涛　冯锡炜　主编　贾宗福　主审

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

高等教育应用型人才培养系列教材

——科瑞国际校企合作指定教材

Java 程序设计教程

（进阶篇）

主　编　杨　涛　冯锡炜

副主编　金　英　付　伟　乔　付

参　编　青巴图　马英瑞　赵　松

周　波　栗庆吉　王知非

任向民　苗世迪　于　波

周　洋

主　审　贾宗福

中国铁道出版社

CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书为 Java 程序设计语言教程的应用技术篇，通过书中的大量经典案例，可帮助读者达到快速理解知识点的目的。第 1 章介绍了 Java 的异常处理机制，包括异常的基本概念，如何进行异常处理以及自定义异常等。第 2 章介绍了包的概念及使用、类成员的访问权限等。第 3 章介绍了类集框架的使用，包括常用的 List、Set、Map 等。第 4 章介绍了 Java 输入/输出流的概念及使用，包括 Java 输入/输出流、文件的读写、文件管理等。第 5 章介绍了 Java 中多线程的概念与基本操作方法，以及线程的并发控制、线程同步等。第 6 章介绍了有关网络通信的基础知识以及 Java 对网络通信的支持，包括 Java 基于 URL 的 WWW 资源访问技术以及基于底层 Socket 的有关连接和无连接的网络通信方法等。第 7 章介绍了开发中常用的一些类库及工具，包括 API 的使用、包装类、日期类、算数类等。本书主要介绍编程人员在开发时需要用到的高级应用技术。

本书层次清晰，结构严谨，便于理解，着重应用，既可作为高等院校相关专业的教材，也可作为从事软件开发工作的专业技术人员的参考书。

图书在版编目（CIP）数据

Java 程序设计教程：进阶篇 / 杨涛，冯锡炜主编
. --北京：中国铁道出版社，2010.8
高等教育应用型人才培养系列教材　科瑞国际校企合作指定教材
ISBN 978-7-113-11792-4

Ⅰ.①J…　Ⅱ.①杨…②冯…　Ⅲ.①JAVA 语言－程序设计－高等学校－教材　Ⅳ.①TP312

中国版本图书馆 CIP 数据核字（2010）第 169724 号

书　　名：Java 程序设计教程（进阶篇）
作　　者：杨　涛　冯锡炜　主编

策划编辑：秦绪好　刘彦会
责任编辑：秦绪好　　**读者热线电话：400-668-0820**
封面设计：付　巍　　**封面制作：李　路**
责任印制：李　佳

出版发行：中国铁道出版社（北京市宣武区右安门西街 8 号　　邮政编码：100054）
印　　刷：北京市兴顺印刷厂
版　　次：2010 年 8 月第 1 版　　2010 年 8 月第 1 次印刷
开　　本：787mm×1092mm　1/16　　印张：8.75　　字数：206 千
书　　号：ISBN 978-7-113-11792-4
定　　价：17.00 元

FOREWORD

前 言

Java，是由 Sun Microsystems 公司于 1995 年 5 月推出的 Java 程序设计语言和 Java 平台的总称。用 Java 实现的 HotJava 浏览器（支持 Java applet）显示了 Java 的魅力：跨平台、动态的 Web、Internet 计算。从此，Java 被广泛接受并推动了 Web 的迅速发展，常用的浏览器现在均支持 Java applet。2003 年 5.5 亿台计算机上运行 Java 程序，75%的开发人员将 Java 作为主要的开发工具。至今还保持着编程语言第一的市场份额，可以毫不扩张的说，Java 在软件界占据半壁江山。

每一种语言都会有出生、成长、壮大、衰落几个阶段，Java 也不例外，既然有它的黄金时期，就有它的衰落时期。不过我们大可放心，因为 Java 不只是一门语言更是一门技术，Java 所代表的这一类语言将不会消失。人们需要跨平台、开源的技术，人们需要功能强大但使用简单的技术。我认为未来高级编程语言将会更清晰的分为三大类：一类适合底层开发，追求效率，就像 C 语言；一类适合傻瓜式的开发，目标追求人人都能编程，就像 Delphi；第三类则是追求在效率与简单之间寻找平衡，就像 Java。

本书主要内容包括：第 1 章介绍了 Java 的异常处理机制，包括异常的基本概念，如何进行异常处理以及自定义异常等；第 2 章介绍了包的概念以及使用、类成员的访问权限等；第 3 章介绍了类集框架的使用，包括常用的 List、Set、Map 等；第 4 章介绍了 Java 输入/输出流的概念及使用，包括 Java 输入/输出流、文件的读写、文件管理等；第 5 章介绍了 Java 中多线程的概念与基本操作方法，以及线程的并发控制、线程同步等；第 6 章介绍了有关网络通信的基础知识以及 Java 对网络通信的支持，包括 Java 基于 URL 的 WWW 资源访问技术以及基于底层 Socket 的有关连接和无连接的网络通信方法等；第 7 章介绍了开发中常用的一些类库及工具，包括 API 的使用、包装类、日期类、算数类等。

该书是作者多年的教学经验及项目实战经验的积累，是对 Java 应用技术及学生项目实战演练中出现的问题进行分析、总结的成果。分析结果表明：学生在学习 Java 技术中出现问题根本原因是由于找不到一本便于理解的、由浅入深的教材，本套书有两册：《Java 程序设计教程（入门篇）》和《Java 程序设计教程（进阶篇）》。“入门篇”主要介绍了 Java 核心基础技术，使读者了解 Java 语言的一些关键特性；而“进阶篇”主要介绍编程人员在开发时需要用到的高级应用技术。因此“入门篇”与“进阶篇”搭配使用能够使读者快速的掌握使用 Java 语言进行软件开发的技术。在这里我们将竭诚为 Java 爱好者提供帮助。如果您是一位有着丰富经验的开发人员，并且能够灵活应用像枚举和泛型这样的高级语言特性，那么您就不必学习完“入门篇”再学习“进阶篇”，不过在“进阶篇”中根据内容需要适当的参考“入门篇”中的有关内容。两本书配合使用能满足读者学习 Java 语言由浅入深的需求，使得读者不必再徘徊在 Java 技术的大门外，使其能够走进 Java 语言的美妙世界。

作者根据 Java 技术知识的掌握及运用情况，系统地重新规划了 Java 基础课程中所必须掌握的知识点，而且对于理解能力较好的读者，本书也提供了较高层次知识的讲解。另外本教材结合案例教学方式，对 Java 实用技术进行全面讲解，使读者能在最短的时间内熟练掌握 Java 技术成为可能。

本书第1、2章由杨涛编写，第3、4章由冯锡炜编写，第5章由金英编写，第6章由付伟编写，第7章由乔付编写，青巴图、马英瑞、赵松、周波、栗庆吉、王知非、任向民、苗世迪、于波、周洋也参加了本书的编写工作，全书由贾宗福负责审稿。

由于编者水平、时间有限，书中难免有疏漏和不妥之处，敬请各位专家、老师和广大读者不吝指正。

编 者

2010年6月

CONTENTS

目 录

第1章 异常处理机制

即使在编译时没有错误信息产生，但在程序运行时，经常会出现一些运行时的错误，这种错误对 Java 而言是一种异常。有了异常就要有相应的处理方式。本章将介绍异常的基本概念以及相关的处理方式。

1.1 异常的基本概念

异常也称为例外，是在程序运行过程中发生的、会打断程序正常执行的事件，下面是几种常见的异常：

- 算术异常（ArithmeticException）。
- 没有给对象开辟内存空间时会出现空指针异常（NullPointerException）。
- 找不到文件异常（FileNotFoundException）。

所以在程序设计时，必须考虑到可能发生的异常事件，并做出相应的处理。这样才能保证程序可以正常运行。

Java 的异常处理机制也秉承着面向对象的基本思想。在 Java 中，所有的异常都是以类的类型存在，除了内置的异常类之外，Java 也可以自定义异常类。此外，Java 的异常处理机制也允许自定义抛出异常。关于这些概念，将在后面介绍。

1.1.1 为何需要异常处理

在没有异常处理的语言中，就必须使用 if 或 switch 等语句，配合所想得到的错误状况来捕捉程序里所有可能发生的错误。但为了捕捉这些错误，编写出来的程序代码经常有很多的 if 语句，有时候这样也未必能捕捉到所有的错误，而且这样做势必导致程序运行效率的降低。

Java 的异常处理机制恰好改进了这一点。它具有易于使用、可自行定义异常类，处理抛出的异常同时又不会降低程序运行的速度等优点。因而在 Java 程序设计时，应充分地利用 Java 的异常处理机制，以增进程序的稳定性及效率。

1.1.2 简单的异常范例

Java 本身已有相当好的机制来处理异常的发生。本节先来看看 Java 是如何处理异常的。TestException1_1 是一个错误的程序，它在访问数组时，下标值已超过了数组下标所容许的最大值，因此会有异常发生。

```
范例: TestException1_1.java
01  public class TestException1_1
02  {
03        public static void main(String args[])
04        {
05           int arr[]=new int[5];  // 容许 5 个元素
06           arr[10]=7;             // 下标值超出所容许的范围
07           System.out.println("end of main() method !!");
08        }
09  }
```

在编译的时候程序不会发生任何错误，但是在执行到第 6 行时，会产生下列的错误信息：

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at TestException1_1.main(TestException1_1.java:6)
```

错误的原因在于数组的下标值超出了最大允许的范围。Java 发现这个错误之后，便由系统抛出“ArrayIndexOutOfBoundsException”这个种类的异常，用来表示错误的原因，并停止运行程序。如果没有编写相应的处理异常的程序代码，则 Java 的默认异常处理机制会先抛出异常、然后停止程序运行。

1.1.3 异常的处理

TestException1_1 的异常发生后，Java 便把这个异常抛了出来，可是抛出来之后没有程序代码去捕捉它，所以程序到第 6 行便结束，因此根本不会执行到第 7 行。如果加上捕捉异常的程序代码，则可针对不同的异常做妥善的处理。这种处理的方式称为异常处理。

异常处理是由 try、catch 与 finally 三个关键字所组成的程序块，其语法如下：

【格式 1-1　异常处理的语法】

```
try
{
    要检查的程序语句 ;          } try 语句块
    …
}
catch(异常类  对象名称)
{                               } catch 语句块
    异常发生时的处理语句 ;
}
finally
{                               } finally 语句块
    一定会运行到的程序代码 ;
}
```

格式 1-1 的语法是依据下列的顺序来处理异常：

try 程序块若是有异常发生时，程序的运行便中断，并抛出“异常类所产生的对象”。

抛出的对象如果属于 catch()括号内欲捕获的异常类，则 catch 会捕捉此异常，然后进到 catch 的块里继续运行。

无论 try 程序块是否有捕捉到异常，或者捕捉到的异常是否与 catch()括号里的异常相同，最后一定会运行 finally 块里的程序代码。

finally 的程序代码块运行结束后，程序再回到 try-catch-finally 块之后继续执行。

由上述的过程可知，异常捕捉的过程中做了两个判断：第一个是 try 程序块是否有异常产生，第二个是产生的异常是否和 catch()括号内欲捕捉的异常相同。

值得一提的是，finally 块是可以省略的。如果省略了 finally 块不写，则在 catch()块运行结束后，程序跳到 try-catch 块之后继续执行。

根据这些基本概念与运行的步骤，可以绘制出如图 1-1 所示的流程图。

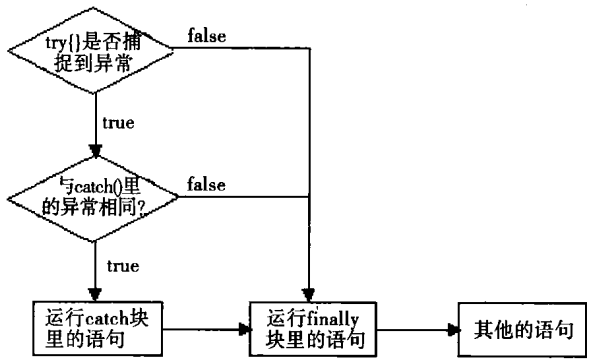

图 1-1 异常处理的流程图

在格式 1-1 中，“异常类”指的是由程序抛出的对象所属的类，例如，TestException1_1 中出现的“ArrayIndexOutOfBoundsException”就是属于异常类的一种。至于有哪些异常类以及它们之间的继承关系，稍后本书将会做更进一步的探讨。下面的程序代码加入了 try 与 catch，使得程序本身具有捕捉异常与处理异常的能力。

范例：TestException1_2.java

```
01  public class TestException1_2
02  {
03      public static void main(String args[])
04      {
05          try                    // 检查这个程序块的代码
06          {
07              int arr[]=new int[5];
08              arr[10]=7;         // 在这里会出现异常
09          }
10          catch(ArrayIndexOutOfBoundsException e)
11          {
12              System.out.println("数组超出绑定范围！");
13          }
14          finally                // 这个块的程序代码一定会执行
15          {
16              System.out.println("这里一定会被执行！");
17          }
18          System.out.println("main()方法结束！");
19      }
20  }
```

输出结果：

```
数组超出绑定范围!
这里一定会被执行!
main()方法结束!
```

程序说明：

1. 程序第 7 行声明一个 arr 的整型数组，并开辟了 5 个数据空间。

2. 程序第 8 行为数组中的第 10 个元素赋值，此时已经超出了该数组本身的范围，所以会出现异常。发生异常之后，程序语句转到 catch 语句中去处理，程序通过 finally 代码块统一结束。

上面程序的第 5~9 行的 try 块是用来检查是否会有异常发生。若有异常发生，且抛出的异常是属于 ArrayIndexOutOfBoundsException 类，则会运行第 10~13 行的代码块。因为第 8 行所抛出的异常正是 ArrayIndexOutOfBoundsException 类，因此第 12 行会输出“数组超出绑定范围!”字符串。由本例可看出，通过异常的机制，即使程序运行时发生问题，只要能捕捉到异常，程序便能顺利地运行到最后，且还能适时地加入对错误信息的提示。

程序 TestException1_2 里的第 10 行，如果程序捕捉到了异常，则在 catch 括号内的异常类 ArrayIndexOutOfBoundsException 之后生成一个对象 e，利用此对象可以得到异常的相关信息，下例说明了对象 e 的应用。

范例: TestException1_3.java

```
01  public class TestException1_3
02  {
03     public static void main(String args[])
04     {
05          try
06          {
07               int arr[]=new int[5];
08               arr[10]=7;
09          }
10          catch(ArrayIndexOutOfBoundsException e){
11               System.out.println("数组超出绑定范围! ");
12               System.out.println("异常: "+e);    // 显示异常对象e的内容
13          }
14          System.out.println("main()方法结束! ");
15     }
16  }
```

输出结果：

```
数组超出绑定范围!
异常: java.lang.ArrayIndexOutOfBoundsException: 10
main()方法结束!
```

范例 TestException1_3 省略了 finally 块，但程序依然可以运行。在第 10 行中，把 catch() 括号内的内容想象成是方法的参数，而 e 就是 ArrayIndexOutOfBoundsException 类的对象。对

象 e 接收到由异常类所产生的对象之后，就进到第 11 行，输出“数组超出绑定范围!”这一字符串，而第 12 行则是输出异常所属的种类，也就是 java.lang. ArrayIndexOutOfBoundsException。而 java.lang 正是 ArrayIndexOutOfBoundsException 类所属的包。

1.1.4 异常处理机制的回顾

当异常发生时，通常可以用两种方法来处理，一种是交由 Java 默认的异常处理机制做处理。但这种处理方式，Java 通常只能输出异常信息，接着便终止程序的运行。如 TestException1_1 的异常发生后，Java 默认的异常处理机制会显示出：

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at TestException1_1.main(TestException1_1.java:6)
```

接着结束 TestException1_1 的运行。

另一种处理方式是自行编写 try-catch-finally 块来捕捉异常，如 TestException1_2 与 TestException1_3。自行编写程序代码来捕捉异常最大的好处是：可以灵活操控程序的流程，且可做出最适当的处理。图 1-2 绘出了异常处理机制的选择流程。

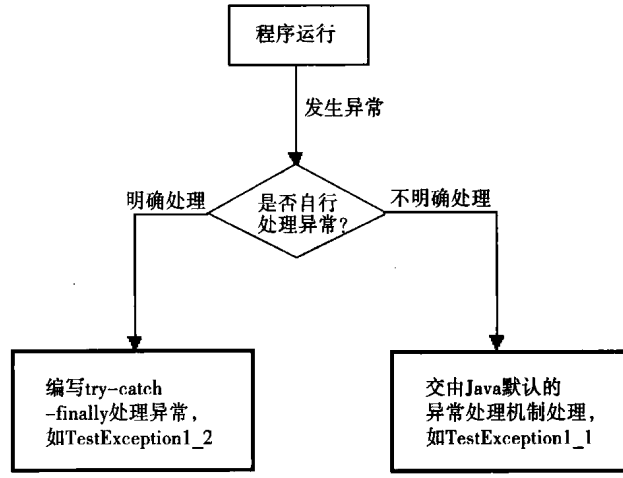

图 1-2 异常处理的方法

1.2 异常类的继承架构

异常可分为两大类：java.lang.Exception 类与 java.lang.Error 类。这两个类均继承自 java.lang.Throwable 类。图 1-3 为 Throwable 类的继承关系图。

习惯上将 Error 与 Exception 类统称为异常类，但这两者本质上还是有不同的：Error 类专门用来处理严重影响程序运行的错误，可是通常程序设计者不会设计程序代码去捕捉这种错误，其原因在于即使捕捉到它，也无法给予适当的处理，如 Java 虚拟机出错就属于一种 Error，不同于 Error 类，Exception 类包含了一般性的异常，这些异常通常在捕捉到之后便可做妥善的处理，以确保程序继续运行，如 TestException1_2 里所捕捉到的 ArrayIndexOutOfBoundsException 就是属于这种异常。

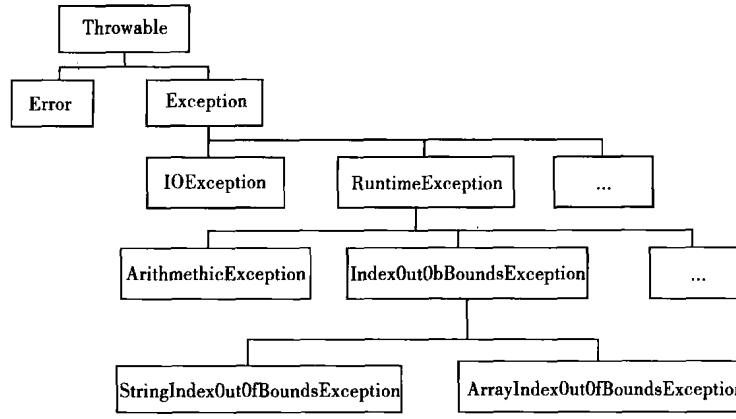

图 1-3 Throwable 类的继承关系图

从异常类的继承架构图中可以看出：Exception 类扩展出数个子类，其中 IOException、RuntimeException 是较常用的两种。RuntimeException 即使不编写异常处理的程序代码，依然可以编译成功，而这种异常必须是在程序运行时才有可能发生，例如，数组的索引值超出了范围。与 RuntimeException 不同的是，IOException 一定要编写异常处理的程序代码才行，它通常用来处理与输入/输出相关的操作，如文件的访问、网络的连接等。

当异常发生时，发生异常的语句代码会抛出一个异常类的实例化对象，之后，此对象与 catch 语句中的类的类型进行匹配，然后在相应的 catch 中进行处理。

1.3 抛出异常

前两节介绍了 try-catch-finally 程序块的编写方法，本节将介绍如何抛出（throw）异常，以及如何由 try-catch 来接收所抛出的异常。抛出异常有下列两种方式：

- 程序中抛出异常。
- 指定方法抛出异常。

以下两小节将介绍如何在程序中抛出异常以及如何指定方法抛出异常。

1.3.1 在程序中抛出异常

在程序中抛出异常时，一定要用到 throw 这个关键字，其语法如下：

【格式 1-2 抛出异常的语法】

```
throw 异常类实例对象;
```

从格式 1-2 中可以发现在 throw 后面抛出的是一个异常类的实例对象，下面来看一个实例：

```
范例: TestException1_4.java
01  public class TestException1_4
02  {
03      public static void main(String args[])
04      {
05          int a=4,b=0;
06          try
```

```
07              {
08                   if(b==0)
09                     throw new ArithmeticException("一个算术异常");
                                        // 抛出异常
10                   else
11                     System.out.println(a+"/"+b+"="+a/b);
                                        // 若抛出异常，则执行此行
12              }
13              catch(ArithmeticException e)
14              {
15                   System.out.println("抛出异常为: "+e);
16              }
17         }
18   }
```

输出结果：

```
抛出异常为: java.lang.ArithmeticException: 一个算术异常
```

程序说明：

程序 TestException1_4 是要计算 a/b 的值。因 b 是除数，不能为 0。若 b 为 0，则系统会抛出 ArithmeticException 异常，代表除到 0 这个数。

在 try 块里，利用第 8 行来判断除数 b 是否为 0。如果 b=0，则运行第 9 行的 throw 语句，抛出 ArithmeticException 异常。如果 b 不为 0，则输出 a/b 的值。在此例中强制把 b 设为 0，因此 try 块的第 9 行会抛出异常，并由第 13 行的 catch()捕捉到异常。

抛出异常时，throw 关键字所抛出的是异常类的实例对象，因此第 9 行的 throw 语句必须使用 new 关键字来产生对象。

1.3.2 指定方法抛出异常

如果方法内的程序代码可能会发生异常，且方法内又没有使用任何的代码块来捕捉这些异常时，则必须在声明方法时一并指明所有可能发生的异常，以便让调用此方法的程序得以做好准备来捕捉异常。也就是说，如果方法会抛出异常，则可将处理此异常的 try-catch-finally 块写在调用此方法的程序代码内。

如果要声明方法抛出异常，则使用如下语法：

【格式 1-3 声明方法抛出异常】

```
方法名称（参数…） throws 异常类 1，异常类 2，…
```

范例 TestException1_5 是指定由方法来抛出异常的，注意此处把 main()方法与 add()方法编写在同一个类内，如下所示：

```
范例: TestException1_5.java
01   class Test
02   {
03                        // throws 在指定方法中不处理异常，在调用此方法的地方处理
04        void add(int a,int b) throws Exception
```

```
05        {
06            int c;
07            c=a/b;
08            System.out.println(a+"/"+b+"="+c);
09        }
10    }
11    public class TestException1_5
12    {
13        public static void main(String args[])
14        {
15            Test t=new Test() ;
16            t.add(4,0);
17        }
18    }
```

编译结果：

```
TestException1_5.java:16: unreported exception java.lang.Exception;
must be caught
or declared to be thrown
                t.add(4,0);
                    ^
1 error
```

程序说明：

1. 程序 1～10 行声明一 Test 类，此类中有一 add(int a,int b)方法，但在此方法后用 throws 关键字抛出一 Exception 异常。

2. 程序第 15 行实例化一 Test 对象 t，在第 16 行调用 Test 类中的 add()方法。

从上面的编译结果中可以发现，如果在类的方法中用 throws 抛出一个异常，则在调用它的地方就必须明确地用 try-catch 来捕捉。

!小提示：在 TestExeption1_5 程序之中，如果在 main()方法后再用 throws Exception 声明的话，那么程序也是依然可以编译通过的。也就是说在调用用 throws 抛出异常的方法时，可以将此异常在方法中再向上传递，而 main()方法是整个程序的起点，所以如果在 main()方法处再用 throws 抛出异常，则此异常就将交由 JVM 进行处理了。

1.4 编写自己的异常类

为了处理各种异常，Java 可通过继承的方式编写自己的异常类。因为所有可处理的异常类均继承自 Exception 类，所以自定义异常类也必须继承这个类。自己编写异常类的语法如下：

【格式 1-4 编写自定义异常类】

```
class 异常名称 extends Exception
{
   ...
}
```

读者可以在自定义异常类里编写方法来处理相关的事件，甚至可以不编写任何语句也可正常地工作，这是因为父类 Exception 已提供相当丰富的方法，通过继承，子类均可使用它们。

接下来以一个范例来说明如何定义自己的异常类以及如何使用它们。

范例: TestException1_6.java

```
01  class DefaultException extends Exception
02  {
03      public DefaultException(String msg)
04      {
05              // 调用 Exception 类的构造方法，存入异常信息
06          super(msg) ;
07      }
08  }
09  public class TestException1_6
10  {
11      public static void main(String[] args)
12      {
13          try
14          {
15              // 在这里用 throw 直接抛出一个 DefaultException 类的实例对象
16                throw new DefaultException("自定义异常! ") ;
17          }
18          catch(Exception e)
19          {
20                System.out.println(e) ;
21          }
22      }
23  }
```

输出结果：

```
DefaultException: 自定义异常!
```

程序说明：

1. 程序 1～8 行声明一 DefaultException 类，此类继承自 Exception 类，所以此类为自定义异常类。

2. 程序第 6 行调用 super 关键字，调用父类（Exception）的有一个参数的构造方法，传入的为异常信息。

Exception 构造方法：

```
public Exception(String message)
```

3. 程序第 16 行用 throw 抛出一 DefaultException 异常类的实例化对象。

在 JDK 中提供的大量 API 方法之中含有大量的异常类，但这些类在实际开发中往往并不能完全满足设计者对程序异常处理的需要，在这个时候就需要用户自己去定义所需的异常类了，用一个类清楚地写出所需要处理的异常。