第6章

异常处理

面向对象程序设计

6.1 异常概述

为了保证程序的正确执行，准确地检测到程序运行过程中可能出现的各种异常，并进行有效地控制是十分关键的。

传统异常处理：

1. 处理异常的代码量大
2. 影响程序的可读性
3. 缺乏异常处理的规范性

6.1.1 异常的概念

影响程序正常运行的主要原因来自两个方面：

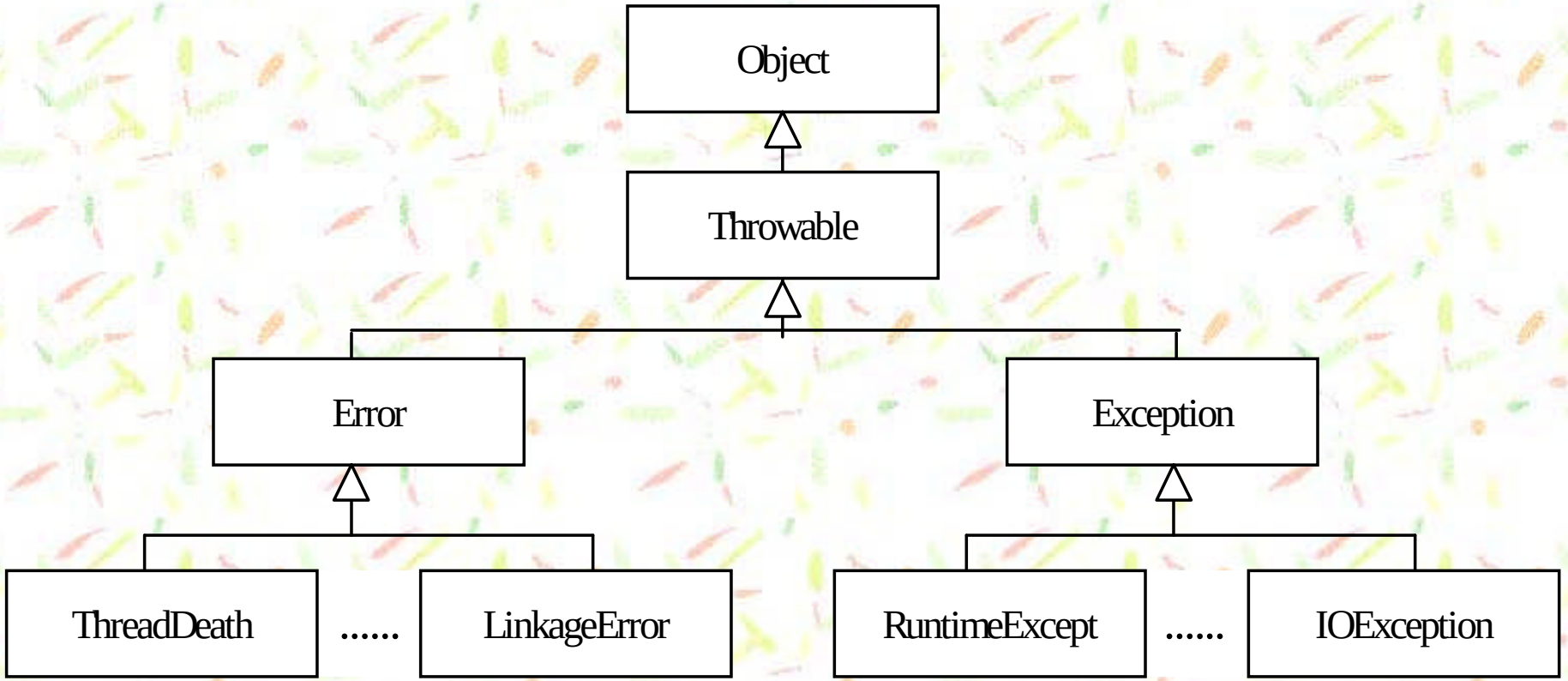
一方面是运行程序的系统出现了异常。例如，系统软、硬件发生的故障，资源短缺等；另一方面是程序本身存在的问题。例如，语法错误、逻辑错误和运行错误等。

在Java语言中，所说的异常是指那些影响程序正常运行的错误，而并不包含导致程序运行结果不正确的那些逻辑错误。

6.1.2 Java语言中的异常类

在Java语言中，对很多可能出现的异常进行了标准化，并将它们封装成了各种各样的类，我们将统称为异常类。一旦在程序运行过程中发生异常，Java虚拟机就会自动地创建一个相应的异常类对象，并将该对象作为参数抛给处理异常的方法。在这些异常类中，主要包含了有关异常的属性信息，跟踪信息等。

JAVA异常类结构



面向对象程序设计

Exception类

Exception类标识的异常通常是由应用程序本身所致的，因此，一旦出现这些异常，应用程序需要做出必要的反映。Exception中包含两个比较重要的子类，一个是IOException类，它包含了有关输入输出的异常；另外一个是RuntimeException，它又包含了很多子类，这些子类分别标识了程序运行期间可能出现的各种异常错误。

java.lang的RuntimeException类的异常子类

类名	描述
ArithmeticExecption	如果进行非法的算术运算就会产生这类异常。
IndexutOfBoundsException	在访问String或Vector对象的内容时，如果出现了下标越界将会产生这类异常。
NegativeArraySizeException	如果在创建数组时，将数组的维数指定为负值就会产生这类异常。
NullPointerException	如果试图访问null对象的成员变量或成员方法就会产生这类异常。
ArrayStroeException	如果试图在数组中存入一个数组元素类型不允许的对象就会产生这类异常。
ClassCastException	如果无法将一个对象转换成指定类型的变量就会产生这类异常。
IllegalArgumentException	传递给成员方法的实际参数的类型与形式参数的类型不一致

SecurityException	如果程序执行了一个有可能破坏安全的非法操作就会产生这类异常。
IllegalStateException	如果非法地调用成员方法就会产生这类异常。
UnsupportedOperationException	如果请求执行一个不支持的操作就会产生这类异常。

6.2 异常处理机制

Java程序中，处理异常要经历三个主要阶段：抛出异常，捕获异常和处理异常。当一个异常被抛出并捕获后，既可以就地自行处理，也可以调用相应异常类的成员方法加以处理，还可以抛给调用该方法的成员方法处理。

6.2.1 抛出异常

所谓抛出异常是指在程序的运行过程中，一旦发生了一个可识别的错误，就立即创建一个与该错误相对应的异常类对象，将其作为参数抛给处理该异常的代码块。如果产生的异常是系统可标识的标准异常，则抛出异常的工作就由系统自动地完成；如果产生的异常是用户自定义的异常，就需要应用程序自行地创建异常类对象,并借助throw语句将其抛出。

6.2.2 捕获异常

在Java程序中，捕获异常用try-catch-finally语句实现，该语句可以被用来捕获一个或多个异常，基本语法格式为：

```
try
{
   Java statements
}
catch (ExceptionType1 ExceptionObject){
   handler for this exception type
}
catch (ExceptionType2 ExceptionObject){
   handler for this exception type
}
.....
```

面向对象程序设计

简单的例子

```
public class TestTryCatch     //测试异常捕获类
{
 public static void main(String[] args)
 {
    int i=1;
    int j=0;
    try          //捕获异常语句
    {
      System.out.println("Try block entered "+"i= "+i+" j= "+j);
      System.out.println(i/j);        //产生ArithmeticException异常
      System.out.println("Ending try block");
    }
    catch(ArithmeticException e){
      System.out.println("Arithmetic exception caught");
    }
    System.out.println("After try block");
 }
}
```

面向对象程序设计

为了更好地控制程序的执行过程，使得程序能够在任何情况下都具有统一的结束方式，可以在try语句块的最后一个catch子句之后增加一个finally子句，其基本的语法格式为：

```
try
{
  Java statements
}
catch (ExceptionType1 ExceptionObject){
     handler for this exception type
}
catch (ExceptionType2 ExceptionObject){
     handler for this exception type
}
.......
finally{
     handler for finally
}
```

面向对象程序设计

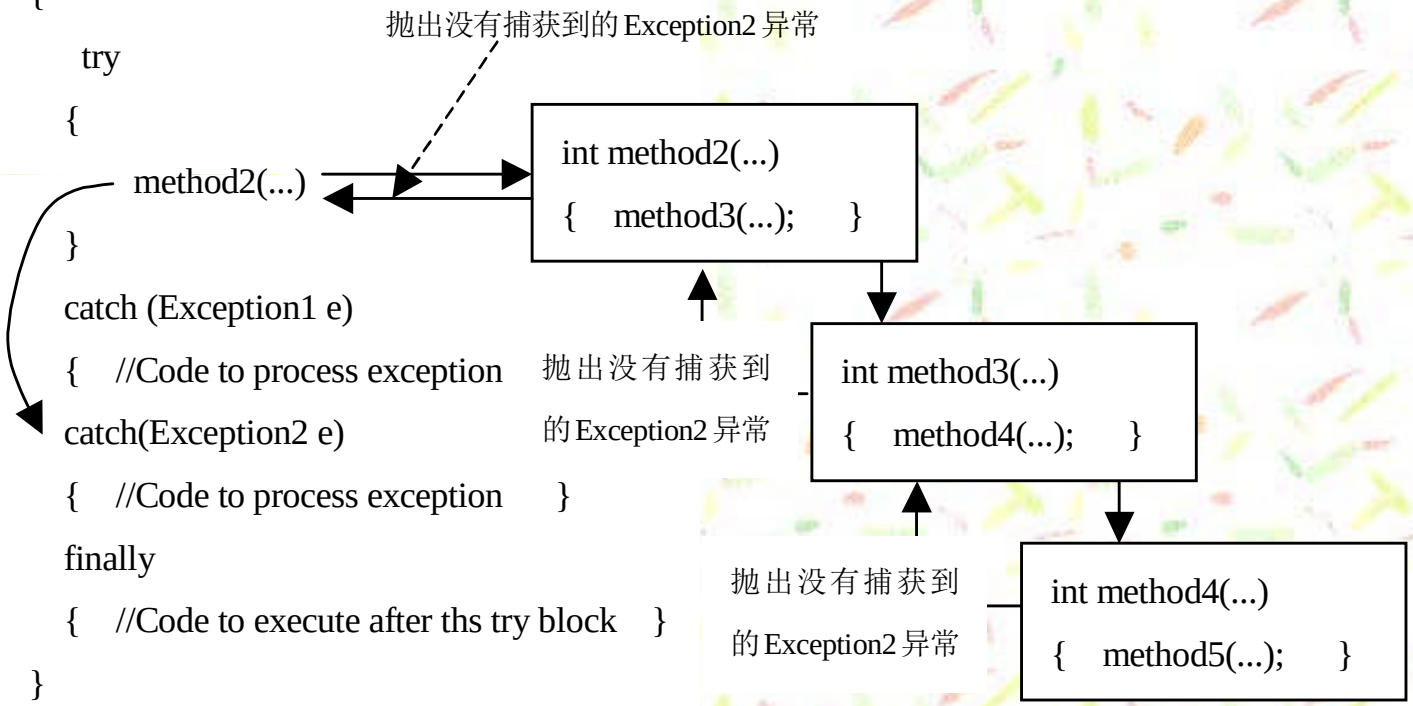
如果在一个try语句块中有可能发生多种不同类型的异常，就需要在try之后放置多个catch子句。

```
try
{
  System.out.println("\nFirst try block in divide() entered");
  array[index+2]=array[index]/array[index+1];
  System.out.println("Code at end of first try block in divide()");
}
catch (ArithmeticException e){
  System.out.println("Arithmetic exception caught in divide()");
}
catch (ArrayIndexOutOfBoundsException e)
{
  System.out.println("Index-out-of-bounds exception caught"}
finally
{
   System.out.println("finally block in divide()");
}
```

捕获异常的基本过程

在一个方法中，对于没有捕获到的那些标准异常，系统将会把它们抛向调用这个方法的方法中去，并且这个过程会不断地向上层方法延伸

```
int method1(...)
{
    try
    {
        method2(...)
    }
    catch (Exception1 e)
    {   //Code to process exception
    catch(Exception2 e)
    {   //Code to process exception    }
    finally
    {   //Code to execute after ths try block   }
}
```



6.2.3 处理异常

在Java语言中，处理异常主要有两种方式：

在产生异常的方法中处理异常

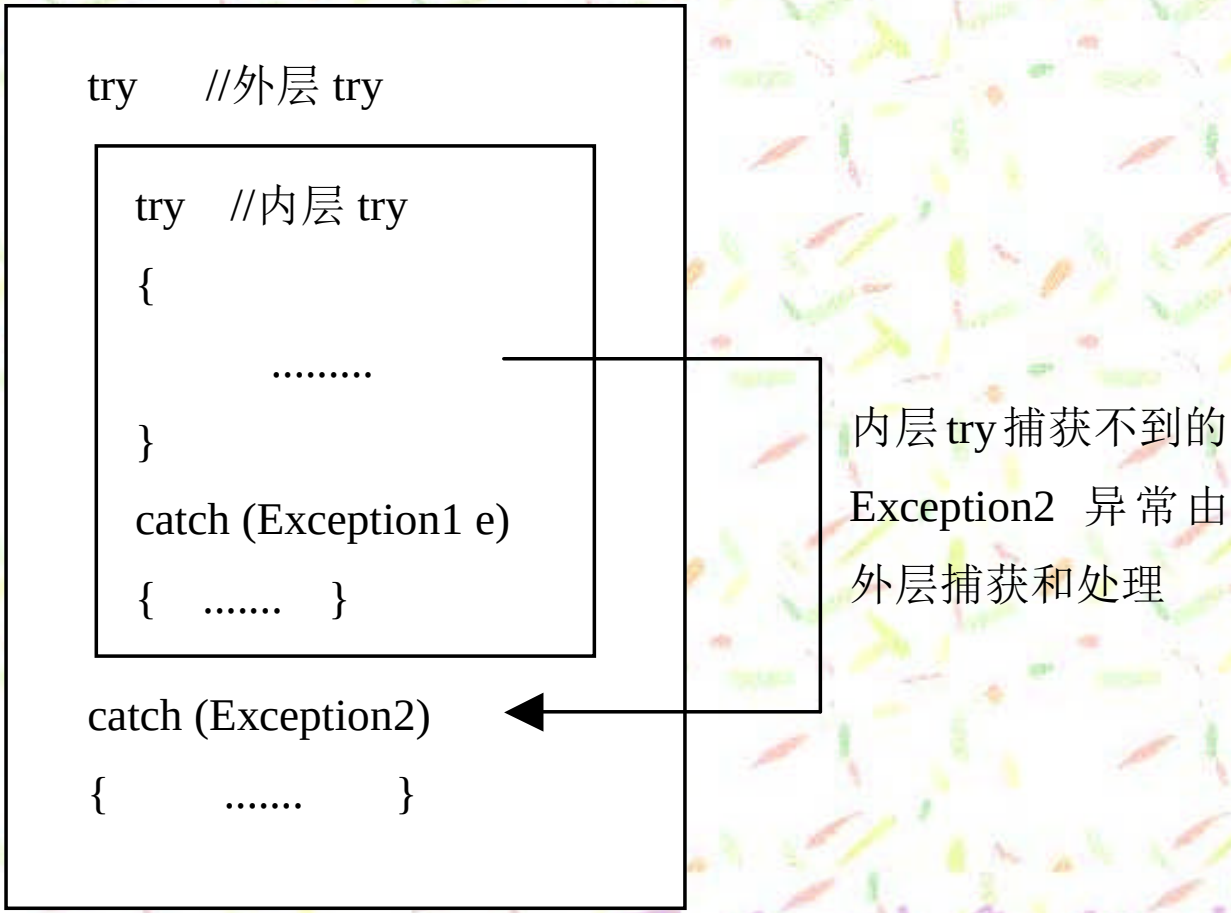
try-catch-finally语句

将异常抛给调用该方法的代码段

Throws语句

嵌套的try语句

如果try语句相互嵌套，内层没有捕获到的异常将由外层捕获。



面向对象程序设计

用thows抛出异常

如果发生异常的方法不清楚具体应该如何处理这个异常，就可以将其抛出，由调用该方法的方法进行捕获和处理，或将其继续向上传递，直到某个方法将其捕获到为止。

```
Modifiers ResultType  MethodName(ParameterList)  throws exceptions
{
  MethodBody
}
```

thows关键字后面列出的是该方法可能抛出的所有异常类名，每个异常类之间用逗号隔开。

面向对象程序设计

异常抛出的简单例子

```
public class ThrowsException
{
    public static void main( String[] args)
    {
      try
      {
            Method( 0 );
            Method( 1 );
      }
      catch(NumberFormatException  e ){
            System.out.println("\t捕获异常："+e);
      }
      catch( ArrayIndexOutOfBoundsException e ){
            System.out.println("\t捕获异常："+e);
      }
```

```
        finally{
            System.out.println("在任何状态下，finally 语句块都将被执行。");
        }
    }

    static void Method( int i )
                        throws ArithmeticException,NumberFormatException
    {
      System.out.println("调用方法Method("+i+")");
       if ( i==0 ){
          System.out.println("\t没有发生异常事件。");
       }
       else  if( i==1 ) {
          String str = "xyz";
         int c=Integer.parseInt(str);
       }
    }
}
```

6.2.4 用户自定义异常类

为了更加方便应用程序捕获和处理异常，Java语言将很多异常进行了标准化，并组成了类层次结构。不仅如此，它还为用户提供了自定义异常类的能力，使得用户可以根据自己的需求，定义符合自己需求的异常类。

Throwable类

Java语言要求，任何异常类都必须是Throwable类的子类，Throwable是所有异常类的公共父类。

Throwable类中主要包含了由构造方法初始化的异常描述性信息和创建异常对象时堆栈的记录情况，它记载了调用每个成员方法的全部过程。

如果希望访问这些内容，可以通过Throwable类中的public成员方法实现。

Throwable类的public成员方法

成员方法	描述
getMessage()	返回当前异常的描述性信息，其中主要包括异常类的名称及有关异常的简短描述。
printStackTrace()	将堆栈的跟踪信息输出到标准的输出流中。在控制台方式下,标准输出流指屏幕。
printStackTrack(PrintStream s)	将堆栈的跟踪信息通过参数s返回。
fillInstackTrack()	填写跟踪信息。

面向对象程序设计

定义异常类

除了异常类必须是Throwable类的子类之外，建议最好将定义的异常类作为Exception的子类，这样Java编译器才能跟踪程序中抛出的异常位置。

定义异常类的基本格式：

```
class TestException extends Exception
{
   TestException(){super();}
   TestException(String s){ super(s);}
}
```

应用自定义的异常类

```
//filename：Test.java
public class Test
{
   public static void main(String[] args)
   {
      for (int i=0;i<args.length;i++) {
            try
            {
                  thrower(args[i]);
                  System.out.println("Test\""+args[i]+"\"      didn't      throw      an
                     exception");
            }
            catch(Exception e){
                  System.out.println("Test\""+args[i]+"\" threw a "
                        +e.getClass()+"\n with message:"+e.getMessage());
            }
      }
   }
```

面向对象程序设计

```
static int thrower(String s) throws TestException
{
    try
    {
        if (s.equals("divide")){
            int i=0;
            return i/i;
        }
        if (s.equals("null")) {
            s=null;
            return s.length();
        }
        if (s.equals("test"))
            throw new TestException("Test message");
    }
    finally{
        System.out.println("[thrower(\""+s+"\")done]");
    }
}
}//end of  class Test
```

面向对象程序设计

运行这个程序后，应该得下面所示的结果：

```
[thrower("divide")done]
Test"divide" threw a class java.lang.ArithmeticException
 with message:/ by zero
[thrower("null")done]
Test"null" threw a classs java.lang.NullPointerException
 with message:null
[thrower("not")done]
Test"not" didn't throw an exception
[thrower("test")done]
Test"test" threw a class TestException
 with message:Test message
```