Java 程序设计

第11章 异常处理

田英鑫 tyx@hit.edu.cn
哈尔滨工业大学软件学院

第11章 异常处理

& 本章导读

- 11.1 处理错误
- 11.2 捕获异常
- 11.3 使用异常机制的建议

第11章 异常处理

& 本章重点

- 11.1 处理错误
- 11.2 捕获异常

& 本章难点

- 11.2 捕获异常

11.1 处理错误

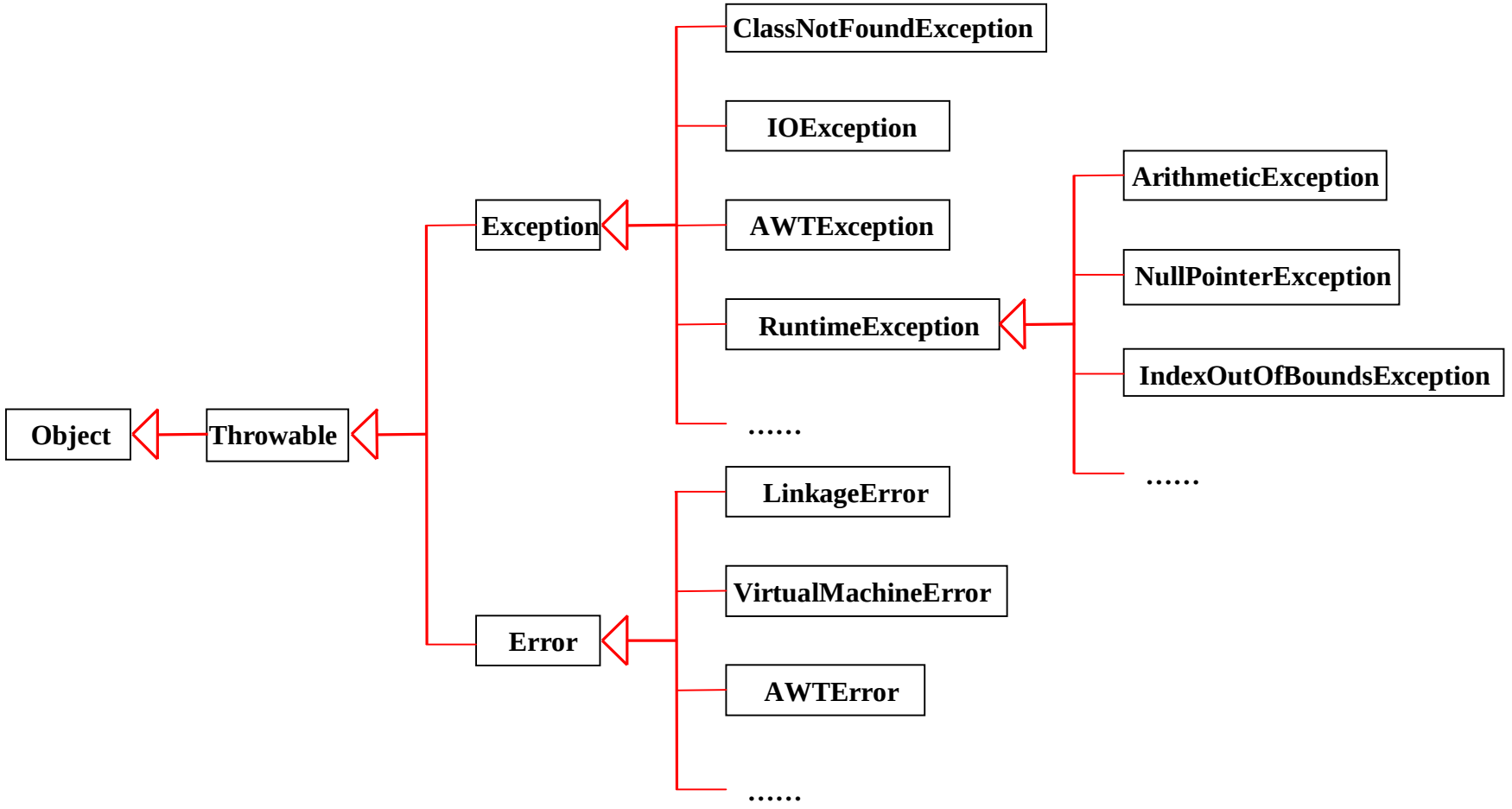
- 错误处理
  - 当程序在运行过程中出现错误而不能完成其操作时，该程序应该
    - 通知用户程序出现了一个错误
    - 返回到一种安全状态，并允许用户执行另外一些命令
    - 允许用户保存其操作结果，并以适当的形式终止程序
  - 异常处理
    - Java使用一种被称为“异常处理（Exception Handling）”的面向对象的错误捕获机制处理程序运行时出现的错误
    - 当方法在运行过程中出现异常时会抛出异常对象

11.1 处理错误

- 错误的分类
  - 用户输入错误
    - 例如：用户可能要求连接到一个语法错误的URL
  - 设备错误
    - 例如：打印机电源被关闭或者缺少打印纸
  - 物理限制
    - 内存或磁盘空间不足
  - 代码错误
    - 使用无效的数组索引
    - 访问哈希表中不存在的条目
    - 对一个空栈进行弹出操作
    - 用零作除数

11.1 处理错误

n 异常的分类



11.1 处理错误

- 异常的分类
  - 在Java语言中，异常对象都是派生于Throwable类的一个实例
  - 所有的异常都由Throwable继承而来，但在下一层立即分解为两个分支：Error和Exception
  - Error类层次结构描述了Java运行时系统的内部错误和资源耗尽错误。应用程序不应该抛出这种类型的对象
  - 在编写Java程序的时候，需要关注Exception层次结构。这个层次结构又分解为两个分支
    - 一个分支是派生于RuntimeException的异常
    - 另一个分支包含其他异常

11.1 处理错误

- 异常的分类
  - 未检查异常
    - 任何RuntimeException类的子类以及Error的子类称为未检查异常
    - 如果出现RuntimeException类型异常应该检查你的程序，而不应该把异常抛给其他调用者
    - 例如：数组访问越界异常、访问空指针异常等
  - 已检查异常
    - 其他的异常被称为已检查异常
    - 如果方法中可能抛出已检查异常，则该方法应该捕获异常或在其方法头中声明它可能会抛出的已检查异常，即把异常抛给下一个调用者
    - 例如：试图打开一个不存在的文件或错误格式的URL等

11.1 处理错误

- 声明已检查异常
  - 如果一个Java方法可能碰到自己无法处理的情况，就应该声明该方法将抛出一个异常
  - 声明异常

```
public FileInputStream(String name)
    throws FileNotFoundException
```

  - 声明多个异常

```
public Image loadImage(String s)
    throws EOFException,MalformedURLException
```

  - 不需要声明的错误和异常
    - 从Error类继承来的错误
    - 从RuntimeException类继承来的未检查异常
  - 方法必须声明它可能抛出的全部“已检查异常”或者在方法内部捕获“已检查异常”

11.1 处理错误

- 如何抛出异常
  - 使用throw语句抛出异常对象

```
String readData(Scanner in) throws EOFException
{
   ...
   while(...) {
      if(!in.hasNext()) {
         if(n < len)
            throw new EOFException();
      }
      ...
   }
   return s;
}
```

11.1 处理错误

- 如何抛出异常
  - 一个方法只能抛出在方法声明中声明的异常以及Error、RuntimeException和它们的子类类型的异常
  - 例如
    - 如果没有在方法声明中声明IOException，那么这个方法不能抛出它
    - 即使方法没有声明RuntimeException或它的子类，方法总能抛出它们
  - 一旦方法抛出异常，该方法就不可能返回到调用者处

11.1 处理错误

- 创建异常类
  - 如果程序可能抛出标准异常类不能描述的异常情况，可以创建自己的异常类
  - 创建自己的异常类应扩展自Exception类并提供一个默认的构造器和一个包含详细信息的构造器

```
class FileFormatException extends IOException
{
   public FileFormatException() {}
   public FileFormatException(String gripe)
   {
      super(gripe);
   }
}
```

11.1 处理错误

- 创建异常类
  - 抛出自己定义的异常

```
String readData(BufferedReader in)
  throws FileFormatException {
   ...
   while(...) {
      if(ch == -1) {
         if(n < len)
            throw new FileFormatException();
      }
      ...
   }
   return s;
}
```

11.2 捕获异常

- 使用 try-catch 语句捕获异常
  - 当调用一个显式声明异常的方法时，应该使用try-catch语句捕获异常，尽量避免把异常抛出
  - 当try语句块中的代码产生异常，程序跳过try块中的其他语句，而转去执行catch块中的代码

```
try
{
    //可能产生异常的代码
}
catch(ExceptionType e)
{
    //对捕获到的异常进行处理
}
```

11.2 捕获异常

- 使用 try-catch 语句捕获异常
  - 如果在一个catch子句捕获了一个父类的异常对象，它就能捕获其子类对象
  - 当子类的方法覆盖了父类的方法时，如果父类的方法抛出了异常，那么子类抛出的异常必须是父类抛出的异常类型或是其子类异常
  - 子类不能抛出超出父类所抛出异常的范围

11.2 捕获异常

- 捕获多个异常
  - try语句块中的代码可能产生多种类型的异常，可以用多个catch语句块分别捕获并处理

```
try {
   //可能产生异常的代码
}
catch(MalformedURLException e1) {
   //处理MalformedURLException异常
}
catch(UnKnownHostException e2) {
   //处理UnKnownHostException异常
}
catch(IOException e3) {
   //处理IOException异常
}
```

11.2 捕获异常

- 捕获多个异常
  - 当捕获多个异常时，如果这些异常存在继承关系，则catch子句应该按照一定的顺序捕获

```
try {
    //可能产生异常的代码
}
catch(Exception e1) {
    //可以捕获任何Exception及其子类的实例
}
catch(IOException e2) {
    //没有机会执行
}
catch(FileNotFoundException e3) {
    //没有机会执行
}
```

11.2 捕获异常

- 再次抛出异常
  - 当try语句块中的代码产生异常时，可以在相应的catch语句中进行处理，处理后还可以把该异常抛出给其他调用者

```
try
{
   access the database
}
catch(SQLException e)
{
   Throwable se = new ServletException("database error");
   se.setCause(e);
   throw se;
}
```

11.2 捕获异常

- finally 子句
  - 有时，不论异常是否出现或是否被捕获，都希望执行某些代码，可以通过finally语句

```
try {
   statements;
}
catch(Exception e) {
   handling e;
}
finally {
   finalStatements;
}
```

11.2 捕获异常

- 异常对象相关方法
  - public String getMessage()
    - 返回异常对象的详细信息
  - public String toString()
    - 返回异常对象的简要描述
  - public String getLocalizedMessage()
    - 返回异常对象的本地化信息
  - public void printStackTrace()
    - 在控制台上打印异常对象和它的追踪信息

参见例11-2:ExceptTest.java

11.3 使用异常机制的建议

- 异常处理不能代替简单的测试
- 不要过分地细化异常
- 利用异常层次结构
- 不要压制异常
- 在检测错误时，“苛刻”要比放任更好
- 不要羞于传递异常

参见例11-3:ExceptionalTest.java

Any Question?