



第12章 异常

导入

- 什么是 exception
- exception 的捕获和处理
- exception 的分类
- 自定义 exception

结构不佳的代码不能运行

- 发现错误的时间
 - 编译
 - 调试
 - 运行
- 使用异常的好处
 - 节省代码
 - 代码分离
 - 便于阅读、编写、调试

什么是 exception?

- divide by zero
- 试图打开的文件不存在
- 网络连接被中断
- 操作数组越界
- 正在装载的类丢失时;
-

出现exception怎么办?

- 创建一个exception对象
 - new 方法在堆上创建exception对象
- 中断正常执行
 - 异常处理机制接管程序执行
- 由exception handler处理exception
 - 寻找到恰当的代码来处理

抛出一个异常

```
try{
```

```
    if(t == null){
```

```
        throw new NullPointerException( "t==null" );
```

```
    }
```

```
}catch(NullPointerException e){
```

```
    System.out.println(e);
```

```
    //t = new TTT( );
```

```
}
```

异常参数
从堆上构造
抛出异常

捕获异常

终止模型
恢复模型

捕获和处理

```
try{
```

```
    // 可能产生异常的代码
```

```
} catch(Type1 id1) {
```

```
    // Handle exceptions of Type1
```

```
} catch(Type2 id2) {
```

```
    // Handle exceptions of Type2
```

```
}  finally{// .....}
```

自定义异常

- 从已有定义的exception类继承,一般继承Exception类

```
class SimpleException extends Exception{}
```

- 带参数的构造方法

```
class MyException extends Exception {
```

```
    MyException() {}
```

```
    MyException(String msg) { super(msg); }
```

```
}
```


异常与日志记录

```
import java.util.logging.*;
import java.io.*;
class LoggingException extends Exception {
    private static Logger logger = Logger.getLogger("LoggingException");
    LoggingException() {
        StringWriter trace = new StringWriter();
        printStackTrace(new PrintWriter(trace));
        logger.severe(trace.toString());
    }
}
```

异常说明

- 程序可能会抛出异常
 - 自己不处理
- 程序调用者需要知道有哪些异常

```
void f() throws TooBig, TooSmall, DivZero{
```

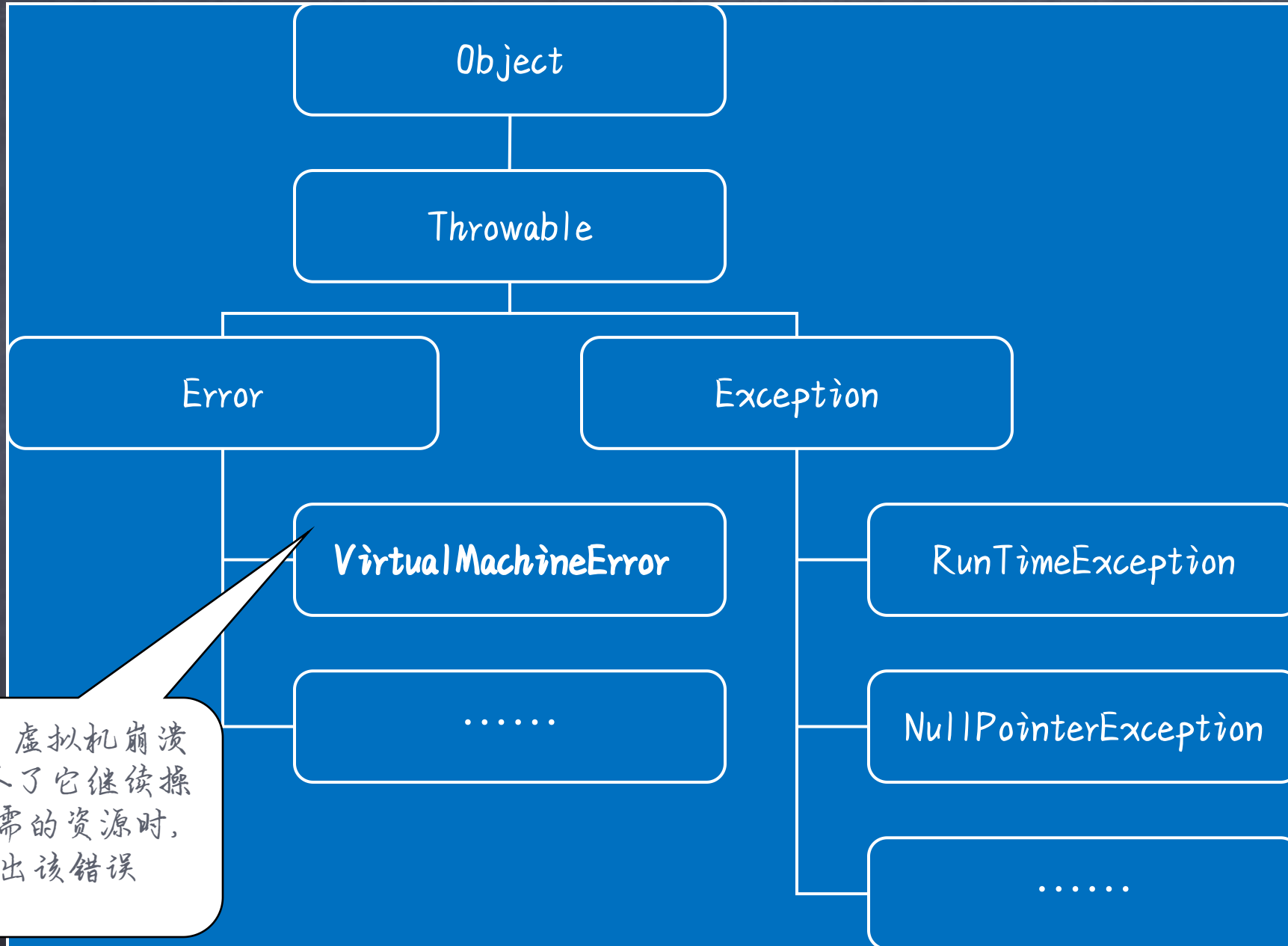
```
...//可能有异常抛出
```

```
}
```

- 可能:有可能不抛出
- 可能出现的异常必须声明抛出

捕获所有异常

```
catch(Exception e){  
    System.err.println( "Caught an exception" );  
}
```



当 Java 虚拟机崩溃
或用尽了它继续操
作所需的资源时，
抛出该错误

Throwable

- `toString()`
- `getMessage()`
- `getLocalizedMessage()`
- `printStackTrace()`
- `printStackTrace(PrintStreams)`
- `printStackTrace(PrintWriters)`
- `fillInStackTrace()`
- 例: `ExceptionMethods.java`

重新抛出异常

```
Catch (Exception e){
```

```
    System.err.println( "A exception was thrown" );
```

```
    throw e;
```

```
}
```

- 将异常处理交给上级处理
- 可以通过 `fillInStackTrace()` 更新状态
- 例: `Rethrowing.java`
- 例: `RethrowNew.java`

finally

- 无论是否有exception都要执行

- 例: FinallyWorks.java


OnOffSwitch.java

WithFinally.java

- 缺陷: finally中抛出的exception未被捕获

- 例: LostMessage.java

重载与异常

- 父类方法声明抛出某种异常
- 子类只能声明抛出这些异常或其子类 
- 子类可以抛出部分异常
- 子类不能抛出新的异常

父类: `throw IOException`

子类: `throw FileNotFoundException`

`throw Exception`

- 例: `StormyInning.java`

构造器与异常

- 子类的构造器可以抛出更多异常
- 子类构造方法中不能捕获父类构造方法中抛出的exception

匹配规则

- exception 与 catch 参数类型相同, 或者为 catch 参数类的子类
- 按先后顺序, 只捕获一次
——先具体、后一般
- 例: Human.java

```
try {  
    //.....  
} catch (Type1 id1) {  
    //.....  
} catch (Type2 id2) {  
    //.....  
} finally {  
    //.....  
}
```


吞食异常

```
try {  
    // ... to do something useful  
} catch (Exception e) {  
    // 什么都不做  
}
```

发生异常时,外界无法感知
隐瞒了问题,并导致程序带病运行
被检查的异常,强制处理

不要隐瞒,让异常尽早暴露

把异常传递给控制台

- `public static void main(String[] args) throws Exception`
- 把“被检查的异常”转换为“不检查的异常”

```
try {
```

```
    // ... to do something useful
```

```
} catch (IDontKnowWhatToDoWithThisCheckedException e) {
```

```
    throw new RuntimeException(e);
```

```
}
```

总结

- 异常能够将商业逻辑与错误处理分开
- 能够报告出现问题的地方与时间
- 不要吞食异常
- 尽早暴露异常

课堂练习

练习1: (2) 编写一个类, 在其`main()`方法的`try`块里抛出一个`Exception`类的对象。传递一个字符串参数给`Exception`的构造器。在`catch`子句里捕获此异常对象, 并且打印字符串参数。添加一个`finally`子句, 打印一条信息以证明这里确实得到了执行。

练习2: (1) 定义一个对象引用并初始化为`null`, 尝试用此引用调用方法。把这个调用放在`try-catch`子句里以捕获异常。

练习3: (1) 编写能产生并能捕获`ArrayIndexOutOfBoundsException`异常的代码。

练习4: (2) 使用`extends`关键字建立一个自定义异常类。为这个类写一个接受字符串参数的构造器, 把此参数保存在对象内部的字符串引用中。写一个方法显示此字符串。写一个`try-catch`子句, 对这个新异常进行测试。

练习5: (3) 使用`while`循环建立类似“恢复模型”的异常处理行为, 它将不断重复, 直到异常不再抛出。

练习6: (1) 创建两个异常类, 每一个都自动记录它们自己的日志, 演示它们都可以正常运行。

练习7: (1) 修改练习3, 使得`catch`子句可以将结果作为日志记录。

练习8: (1) 定义一个类, 令其方法抛出在练习2里定义的异常。不用异常说明, 看看能否通过编译。然后加上异常说明, 用`try-catch`子句测试该类和异常。

练习9: (2) 定义三种新的异常类型。写一个类, 在一个方法中抛出这三种异常。在`main()`里调用这个方法, 仅用一个`catch`子句捕获这三种异常。

练习12: (3) 修改`innerclasses/Sequence.java`, 使其在你试图向其中放置过多地元素时, 抛出一个合适的异常。

练习13: (2) 修改练习9, 加一个`finally`子句。验证一下, 即便是抛出`NullPointerException`异常, `finally`子句也会得到执行。

练习14: (2) 试说明, 在`OnOffSwitch.java`的`try`块内部抛出`RuntimeException`, 程序可能会出现错误。

练习15: (2) 试说明, 在`WithFinally.java`的`try`块内部抛出`RuntimeException`, 程序不会出现错误。

作业

- 提交练习₄



提问