6. 异常机制

Contents

- 异常引入
- 异常处理
  try, catch, finally
- 异常分类
- 异常处理
  throws, throw
- 自定义异常

本章技能点列表

技能点名称	难易程度	认知程度	重要程度
异常定义	易	理解	**
try-catch-finally	中	应用	***
异常类型	易	记忆	**
抛出异常throw	中	应用	**
声明异常throws	中	应用	**
自定义异常	易	应用	**
异常链	中	了解	*

异常引入

- 生活中的异常
  - 正常情况下，小王每日开车去上班，耗时大约30分钟

一路畅通

  - 但是，异常情况迟早要发生！

堵车！

撞车！

异常引入

- 生活中的异常
  - 面对异常该怎么办呢？
  - 生活中，我们会根据不同的异常进行相应的处理，而不会就此中断我们的生活

堵车！

绕行或者等待

撞车！

请求交警解决

异常引入

- 程序中的异常
  - 示例1：给出除数和被除数，求商
    - 如果除数为0，出异常
    - 如果除数或者被除数不是数字，出异常
  - 示例2：将d:/a.txt复制到e:/a.txt
    - 如果d:/a.txt不存在
    - 如果e:/存在a.txt
    - 如果e盘空间不足
    - 如果复制过程中出错

```
if(“d:/a.txt”这个文件存在){
    if(e盘的空间大于a.txt文件长度){
        if(文件复制一半IO流断掉){
                停止copy，输出：IO流出问题！
        }else{
                copyFile("d:/a.txt","e:/a.txt");
        }
    }else{
        输出：e盘空间不够存放a.txt！
    }
}else{
    输出：a.txt不存在！
}
```

真正的代码，只有一行！其余都是用于处理例外情况的代码！

异常引入

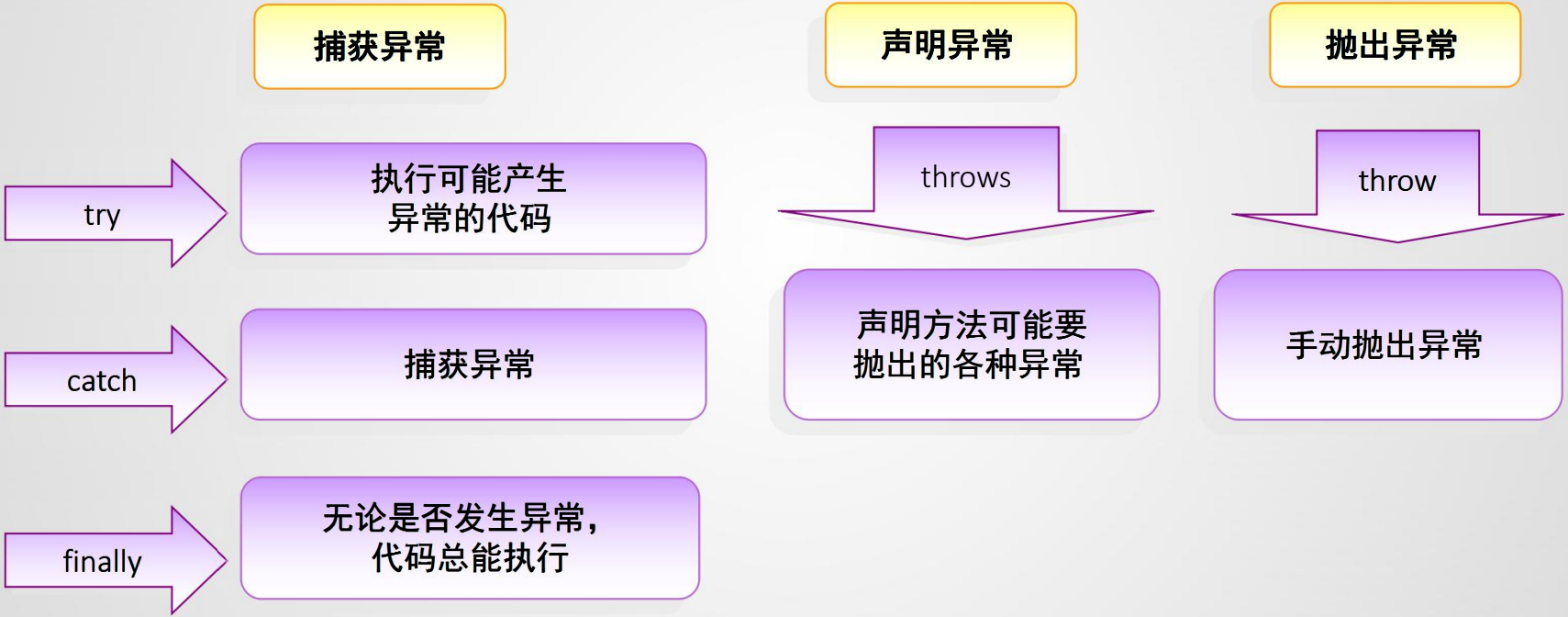
- 程序中的异常
  - 面对异常该怎么办呢？
  - 方式1：由开发者通过if-else来解决异常问题
    - 代码臃肿：业务代码和异常处理代码放一起
    - 程序员要花很大精力"堵漏洞“
    - 程序员很难堵住所有“漏洞”，对程序员本身要求较高
  - 方式2：开发者不需要通过if-else来解决异常问题，而是Java提供异常处理机制。它将异常处理代码和和业务代码分离，使程序更优雅，更好的容错性，高键壮性

异常引入

- 异常（ Exception 也称例外）就是在程序的运行过程中所发生的不正常的事件，它会中断正在运行的程序
  - 所需文件找不到
  - 网络连接不通或中断
  - 算术运算错 (被零除...)
  - 数组下标越界
  - 装载一个不存在的类或者对null对象操作
  - 类型转换异常
  - ……
- 当Java程序出现以上的异常时，就会在所处的方法中产生一个异常对象。这个异常对象包括异常的类型，异常出现时程序的运行状态以及对该异常的详细描述。

异常引入

- Java的异常处理是通过5个关键字来实现的：try、catch、 finally、throw、throws



异常处理

- try-catch
  - 情况1：try块中代码没有出现异常
    - 不执行catch块代码，执行catch块后边的代码
  - 情况2：try块中代码出现异常，catch中异常类型匹配（相同或者父类）
    - 执行catch块代码，执行catch块后边的代码
  - 情况3：try块中代码出现异常， catch中异常类型不匹配
    - 不执行catch块代码，不执行catch块后边的代码，程序会中断运行
  - 注意
    - 出现异常后，Java会生成相应的异常对象，Java系统,寻找匹配的catch块，找到后将异常对象付给catch块异常参数
    - 出现异常后，try块中尚未执行的语句不会执行
    - 出现异常后并处理后，catch块后面的语句还会执行

异常处理

- try-catch
  - catch块中如何处理异常
    - 输出用户自定义异常信息
      - System.err.println("除数不能为零。");
      - System.err.println("被除数和除数必须是整数。");
    - 调用异常对象的方法输出异常信息
      - toString ( )方法，显示异常的类名和产生异常的原因
      - void printStackTrace() 输出异常的堆栈信息
      - String getMessage()返回异常信息描述字符串，是printStackTrace()输出信息的一部分
    - 继续向上抛出异常
      - throw e

异常处理

- try-catch
  - 异常类型

方　法　名	说　　明
Exception	异常层次结构的根类
ArithmeticException	算术错误情形，如以零作除数
ArrayIndexOutOfBoundsException	数组下标越界
NullPointerException	尝试访问 null 对象成员
ClassNotFoundException	不能加载所需的类
InputMismatchException	欲得到数据类型与实际输入类型不匹配
IllegalArgumentException	方法接收到非法参数
ClassCastException	对象强制类型转换出错
NumberFormatException	数字格式转换异常，如把"ab"转换成数字

异常处理

- try-catch-finally
  - 在try-catch块后加入finally块，可以确保无论是否发生异常，finally块中的代码总能被执行
    - 无异常 try-finally
    - 有异常 try-catch-finally
  - 通常在finally中关闭程序块已打开的资源，比如：文件流、释放数据库连接等。

  - finally块中语句不执行的唯一情况
    - 异常处理代码中执行System.exit(1)退出Java虚拟机
  - Finally块的具体执行过程
    - 执行try或catch中代码
    - 遇到return/throw，先执行finally中语句块
    - 执行return/throw

异常处理

- 多重catch
  - 一段代码可能会引发多种类型的异常
  - 当引发异常时，会按顺序来查看每个 catch 语句，并执行第一个与异常类型匹配的catch语句
  - 执行其中一条 catch 语句后，其后 catch 语句将被忽略

  - 在安排catch语句的顺序时，首先应该捕获最特殊的异常， 然后再逐渐一般化，即先子类后父类

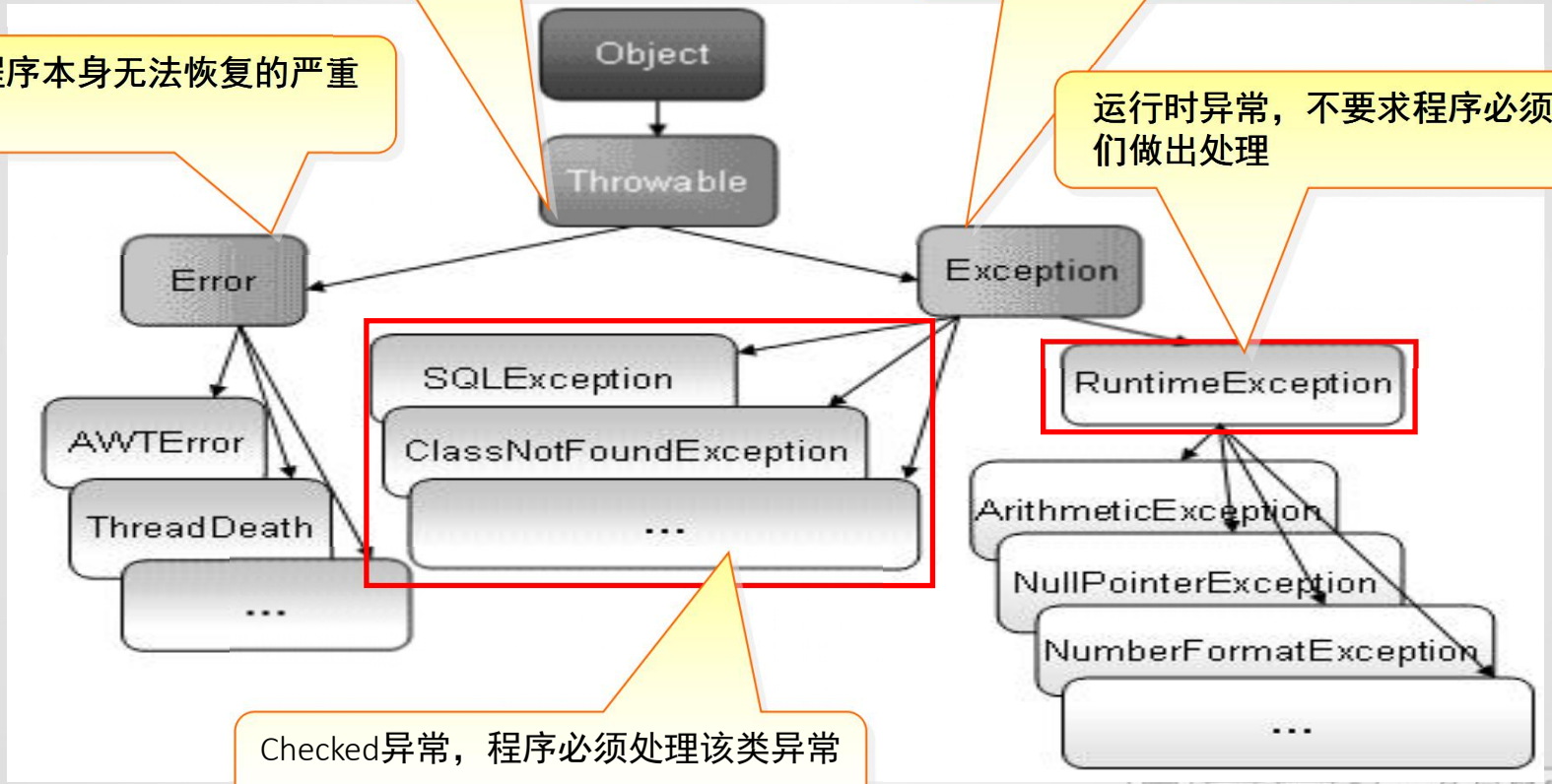
异常分类

Exception和Error类的父类

由Java应用程序抛出和处理的非严重错误

仅靠程序本身无法恢复的严重错误

运行时异常，不要求程序必须对它们做出处理



Checked异常，程序必须处理该类异常

异常分类

- Error
  - Error类层次描述了Java运行时系统内部错误和资源耗尽错误，一般指与JVM或动态加载等相关的问题，如虚拟机错误，动态链接失败，系统崩溃等。
  - 这类错误是我们无法控制的，同时也是非常罕见的错误。所以在编程中，不去处理这类错误。
  - 打开JDK包：java.lang.Error，查看他的所有子类
  - 注：我们不需要管理Error！

异常分类

- Exception
  - 所有异常类的父类，其子类对应了各种各样可能出现的异常事件。
- Exception分类
  - 运行时异常Runtime Exception（unchecked Exception）
  - 可不必对其处理，系统自动检测处理
  - 一类特殊的异常，如被 0 除、数组下标超范围等，其产生比较频繁，处理麻烦，如果显式的声明或捕获将会对程序可读性和运行效率影响很大
  - 检查异常 Checked Exception
  - 必须捕获进行处理，否则会出现编译错误

  - 注意：只有Java提供了Checked异常，体现了Java的严谨性，提高了Java的健壮性。同时也是一个备受争议的问题。

异常处理

- 声明异常throws
  - 当Checked Exception产生时，不一定立刻处理它，可以再把异常Throws出去
  - 如果一个方法抛出多个已检查异常，就必须在方法的首部列出所有的异常，之间以逗号隔开
- 子类声明的异常范围不能超过父类声明范围
  - 父类没有声明异常，子类也不能
  - 不可抛出原有方法抛出异常类的父类或上层类

异常处理

- 手动抛出异常throw
  - Java异常类对象除在程序执行过程中出现异常时由系统自动生成并抛出，也可根据需要手工创建并抛出。
  - 在捕获一个异常前，必须有一段代码先生成异常对象并把它抛出。这个过程我们可以手工做，也可以由JRE来实现，但是他们调用的都是throw子句。
  - 注意抛出运行时异常和Checked异常的区别
    - 抛出Checked异常，该throw语句要么处于try块中，要么方法签名中石油throws抛出
    - 抛出运行时异常，没有以上要求

自定义异常

- 在程序中，可能会遇到任何标准异常类都没有充分的描述清楚的问题，这种情况下可以创建自己的异常类

- 从Exception类或者它的子类派生一个子类即可

- 习惯上，定义的类应该包含2个构造器：一个是默认构造器，另一个是带有详细信息的构造器

总结

- 异常类型
  - 一张图
- 异常处理
  - 五个关键字（try, catch, finally, throws, throw）
  - 先逮小的，再逮大的
  - 自定义异常
- 注意
  - try-finally也可以呀
  - 遇到了return和throw还会执行finally语句；遇到System.exit()就不会执行finally语句了
  - throw了运行时异常，则方法声明中不需要throws