异常

优逸客 让天下没有难学的技术

官网：http://www.sxuek.com/

# 第一章 异常概述与异常体系结构

在使用计算机语言进行项目开发的过程中，即使程序员把代码写得尽善尽美，在系统的运行过程中仍然会遇到一些问题，因为很多问题不是靠代码能够避免的，比如：客户输入数据的格式，读取文件是否存在，网络是否始终保持通畅等等。

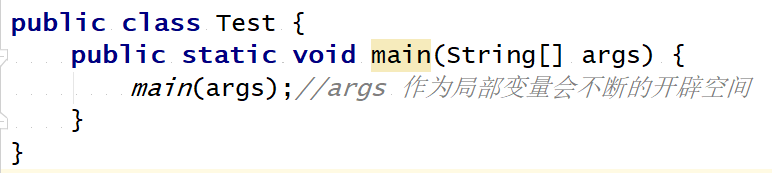
概念：在Java语言中，将程序执行中发生的不正常情况称为“异常”。(开发过程中的语法错误和逻辑错误不是异常)；

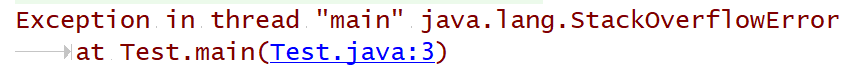
Java程序在执行过程中所发生的异常事件可分为两类：

1. Error:

Java虚拟机无法解决的严重问题。

如：JVM系统内部错误、资源耗尽等严重情况。比如：StackOverflowError和OOM。一般不编写针对性的代码进行处理。





1. Exception:

其它因编程错误或偶然的外在因素导致的一般性问题，可以使用针对性的代码进行处理。

例如：空指针访问、试图读取不存在的文件、网络连接中断、数组角标越界；

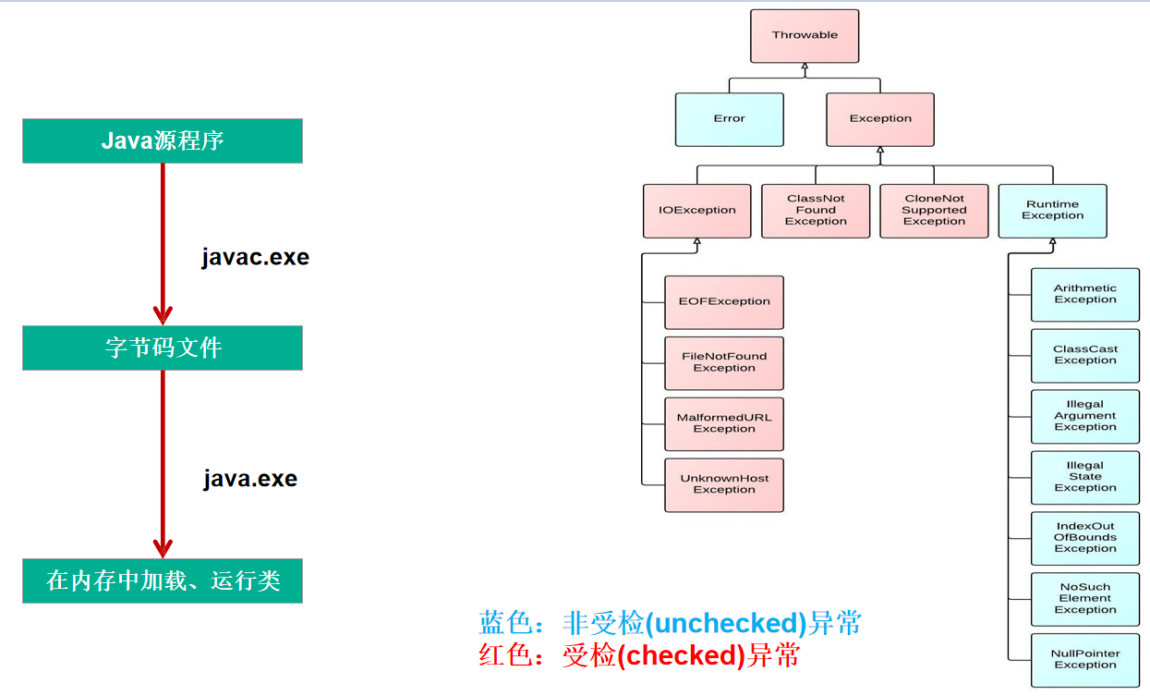
对于这些错误，一般有两种解决方法：

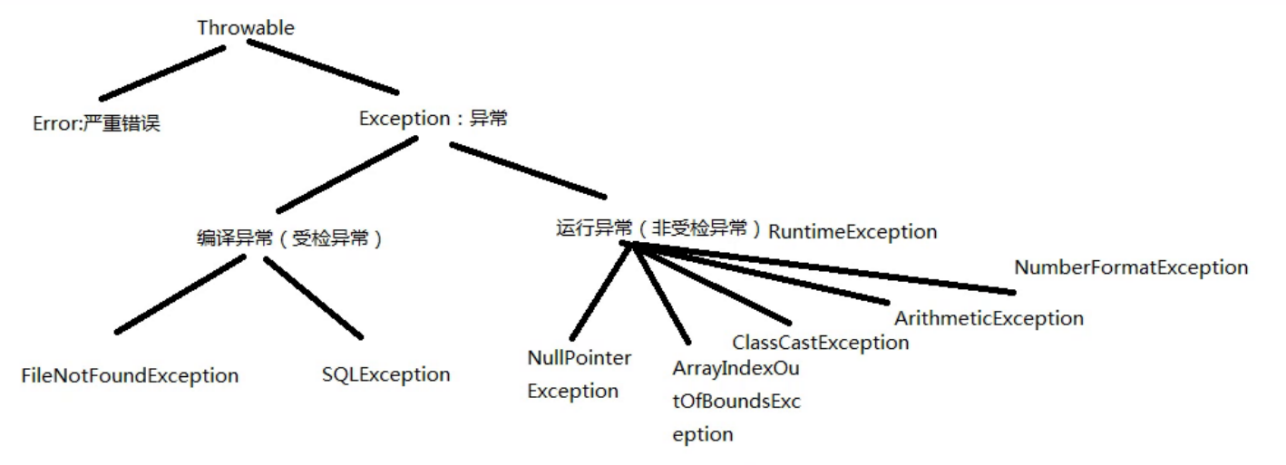
1. 遇到错误就终止程序的运行。
2. 由程序员在编写程序时，就考虑到错误的检测、错误消息的提示，以及错误的处理。

捕获错误最理想的是在编译期间，但有的错误只有在运行时才会发生。比如：除数为0，数组下标越界等。

## 异常分类：

编译时异常和运行时异常。





|  |
| --- |
| *public class TestException1 {*  *public static void main(String[] args) throws FileNotFoundException {*  *// Scanner input = new Scanner(System.in);*  *// int i=input.nextInt(); //运行时异常*  *// System.out.println(i);*  *FileInputStream fis = new FileInputStream("src\\student.txt"); // 编译异常*  *}*  *}* |

1）编译时异常：

是指编译器要求必须处置的异常。即程序在运行时由于外界因素造成的一般性异常。编译器要求java程序必须捕获或声明所有编译时异常。

对于这类异常，如果程序不处理，可能会带来意想不到的结果。

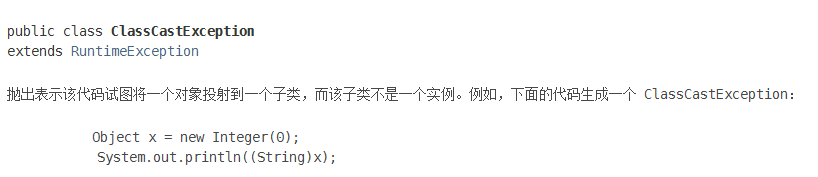
2）运行时异常：

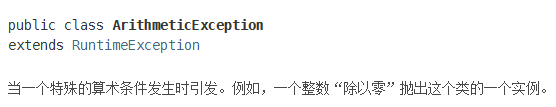
是指编译器不要求强制处置的异常。一般是指编程时的逻辑错误，是程序员应该积极避免其出现的异常。*java.lang.RuntimeException*类及它的子类都是运行时异常。

对于这类异常，可以不作处理，因为这类异常很普遍，若全处理可能会对程序的可读性和运行效率产生影响。

# 第二章 常见运行时异常









|  |
| --- |
| */\*\**  *\* 此类用于演示常见的运行异常*  *\*/*  *class Duck extends Animal{}*  *class Cat extends Animal{}*  *class Animal{}*  *class Boy{*  *Girl g;*  *// public String toString(){ return g.name; }*  *}*  *class Girl{*  *String name;*  *public Girl(String name){this.name=name;}*  *}*  *public class TestException2 {*  *public static void main(String[] args) {*  *//1.空指针异常 ; 当试图使用null对象的属性或方法时；*  *//示例1：*  *// Boy b = new Boy();*  *// System.out.println(b);*  *//示例2：*  *// Boy[] boys = new Boy[3];*  *// for(int i=0;i<boys.length;i++){*  *// boys[i]=new Boy();*  *// boys[i].g=new Girl("柳岩");*  *// }*  *// System.out.println(boys[0].g.name.length());*  *// System.out.println(boys[0]);*  *// System.out.println(boys[0].g);*  *// System.out.println(boys[0].g.name);*  *//2.数组下标越界 ：当试图使用数组的索引超出范围；*  *// System.out.println(args[0]);*  *//3.类型转换异常 ： 当试图将不是该类型的实例强转成该类型；*  *// Animal a = new Duck();*  *// Cat c = (Cat)a;*  *//4.数学运算异常 ： 除数为0时；*  *// int i=1/0;*  *//5.数字格式不正确异常 ： 当试图将字符串转成数值时，如果该字符串不是有效的数值格式；*  *int a=Integer.parseInt(args[0]);*  *}*  *}* |

# 第三章 异常处理机制

在编写程序时，经常要在可能出现错误的地方加上检测的代码，如进行x/y运算时，要检测分母为0，数据为空，输入的不是数据而是字符等。过多的if-else分支会导致程序的代码加长、臃肿，可读性差，因此采用异常处理机制。

概念：Java采用异常处理机制，将异常处理的程序代码集中在一起，与正常的程序代码分开，使得程序简洁，并易于维护。（处理程序中发生的不正常现象，称为异常处理）

## 3.1 Java异常处理机制的方式：

### 3.1.1 try-catch-finally 捕获异常：

Java提供的是异常处理的抓抛模型。

Java程序的执行过程中如出现异常，会生成一个异常类对象，该异常对象将被提交给Java运行时系统（JVM），这个过程称为抛出(throw)异常。

异常对象的生成：

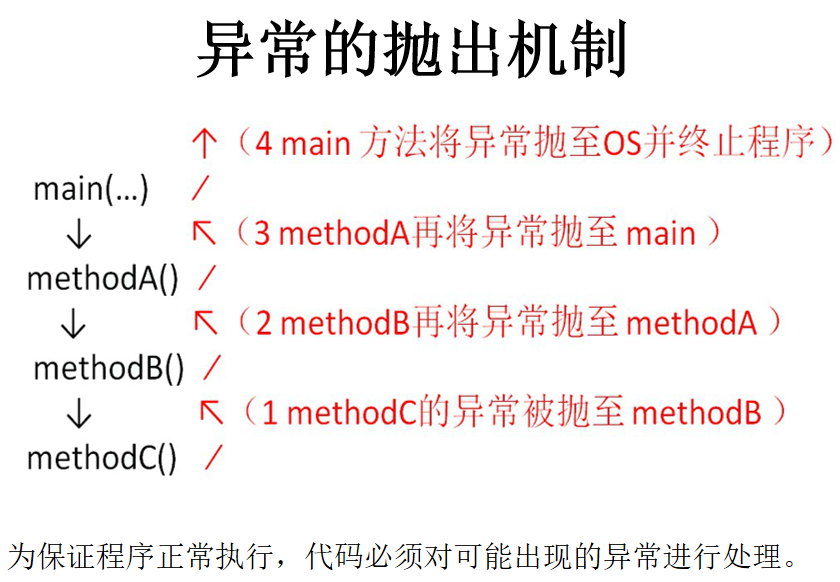
1. 由虚拟机自动生成：

程序运行过程中，虚拟机检测到程序发生了问题，如果在当前代码中没有找到相应的处理程序，就会在后台自动创建一个对应异常类的实例对象并抛出——自动抛出；

1. 由开发人员手动创建：

*Exception exception = new ClassCastException();*

创建好的异常对象不抛出对程序没有任何影响，和创建一个普通对象一样；



如果一个方法内抛出异常，该异常对象会被抛给调用者方法中处理。如果异常没有在调用者方法中处理，它继续被抛给这个调用方法的上层方法。这个过程将一直继续下去，直到异常被处理。 这一过程称为捕获(catch)异常。

如果一个异常回到main()方法，并且main()也不处理，则程序运行终止。

程序员通常只能处理Exception，而对Error无能为力。

1. 语法：

|  |
| --- |
| *try{*  *...... //可能产生异常的代码*  *}*  *catch( ExceptionName1 e ){*  *...... //当产生ExceptionName1型异常时的处置措施*  *}*  *catch( ExceptionName2 e ){*  *...... //当产生ExceptionName2型异常时的处置措施*  *}*  *[ finally{*  *...... //无论是否发生异常，都无条件执行的语句*  *} ]* |

2）解释：

（1）Try：捕获异常的第一步是用try{…}语句块选定捕获异常的范围，将可能出现异常的代码放在try语句块中。

（2）catch (Exceptiontype e) ： 在catch语句块中是对异常对象进行处理的代码。每个try语句块可以伴随一个或多个catch语句，用于处理可能产生的不同类型的异常对象。

如果明确知道产生的是何种异常，可以用该异常类作为catch的参数；也可以用其父类作为catch的参数。比 如 ： 可 以 用 ArithmeticException 类 作 为 参 数 的 地 方 ， 就 可 以 用 RuntimeException类作为参数，或者用所有异常的父类Exception类作为参数。 但不能是与ArithmeticException类无关的异常，如NullPointerException（catch中的语句将不会执行）。

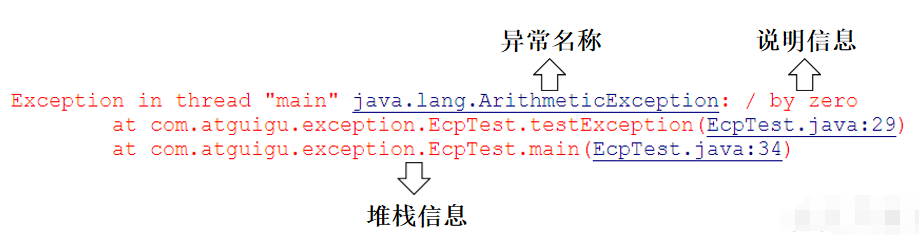
|  |
| --- |
| *public class TestException4 {*  *public static void main(String[] args) {*  *Scanner input = new Scanner(System.in);*  *try{*  *System.out.println("请输入a:");*  *String a = input.next();*  *System.out.println("请输入b:");*  *String b= input.next();*  *//判断两个字符串是否相等*  *// a=null; 空指针*  *if(a.equals(b)){*  *System.out.println("相等");*  *}*  *//将字符串转换成int类型*  *int num1 = Integer.parseInt(a);*  *int num2 = Integer.parseInt(b);*  *//数学运算*  *System.out.println("div="+num1/num2);//容易出现异常*  *System.out.println("哈哈");*  *System.out.println(args[0]);*  *}catch(NumberFormatException e){*  *System.out.println("数字格式不正确吧");*  *}catch(ArithmeticException e){*  *System.out.println("除数为0了吧！");*  *Int i = 1 / 0; 当在catch中出现异常时，整个程序就终止执行*  *}catch(NullPointerException e){*  *System.out.println("空指针了吧");*  *}catch(Exception e){*  *System.out.println("下标越界吧");*  *}finally{//最后一定要执行的语句*  *System.out.println("try catch块后面的语句");*  *}*  *}*  *}* |

（3）捕获异常的有关信息：

与其它对象一样，可以访问一个异常对象的成员变量或调用它的方法。

getMessage() ：获取异常信息，返回字符串；

printStackTrace() ： 获取异常类名和异常信息，以及异常出现在程序中的位置。返回值void。

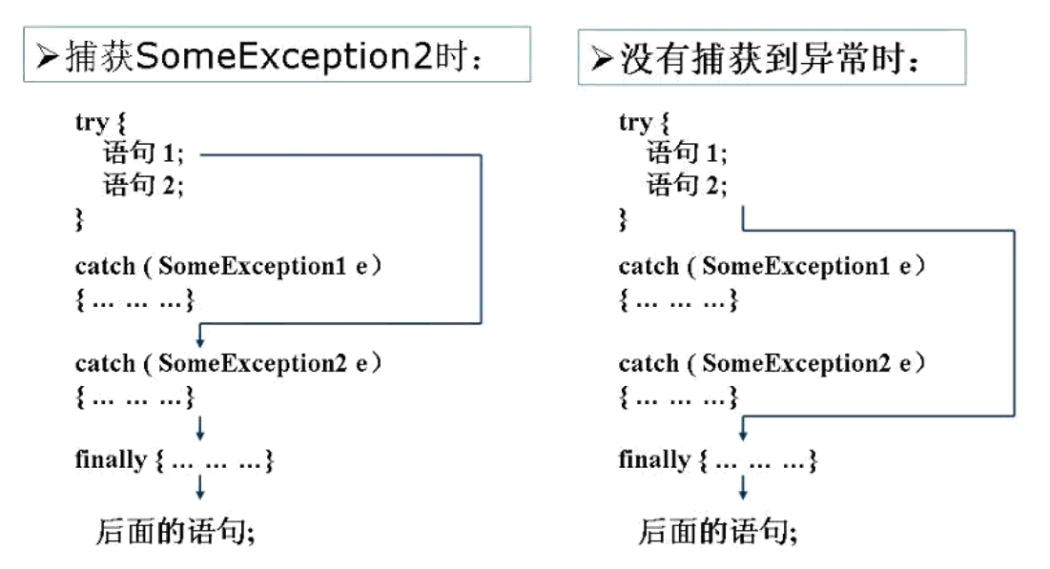


|  |
| --- |
| *public class TestException3 {*  *public static void main(String[] args) {*  *// try{*  *// int age = Integer.parseInt(args[0]);*  *// }catch(NumberFormatException e){ //多态*  *// //具体的处理语句*  *// //方式一：打印错误追踪日志*  *//// e.printStackTrace();*  *// //方式二：打印友好提示*  *//// System.out.println("没有传入有效的数值");*  *// //方式三：打印错误信息*  *// System.out.println("出错了，原因是："+e.getMessage());*  *// }*  *}*  *}* |

（4）Finally：捕获异常的最后一步是通过finally语句为异常处理提供一个统一的出口，使得在控制流转到程序的其它部分以前，能够对程序的状态作统一的管理。

不论在try代码块中是否发生了异常事件，catch语句是否执行，catch语句是否有异常，catch语句中是否有return，finally块中的语句都会被执行。

finally语句和catch语句是任选的。



（5）执行顺序总结：

如果try块出现了异常，则try块中下面的语句将不再执行，而是指向catch块的语句，最后执行finally块的语句；

如果try块没有出现异常，则try块中下面的语句要执行，catch块的语句不再执行，最后执行finally里面的语句；

3）不捕获异常时的情况：

前面使用的异常都是RuntimeException类或是它的子类，这些类的异常的特点是：即使没有使用try和catch捕获，Java自己也能捕获，并且编译通过(但运行时会发生异常使得程序运行终止)。

如果抛出的异常是IOException等类型的非运行时异常，则必须捕获，否则编译错误。也就是说，我们必须处理编译时异常，将异常进行捕捉，转化为运行时异常。

4）捕获异常的好处：

（1）拦截异常，让try catch块下面的语句可以正常执行；

（2）提高用户的体验感；

（3）让正常代码和容易出错的代码进行了有效的分离，从一定程度上，提高语义性和维护性；

1. 捕获异常总结：

并不是try块中的语句越多越好， 因为try块的语句过多，会影响程序执行效率（程序会不停的检查）；

6）练习：

|  |
| --- |
| *public static int method(){*  *int i=10;*  *try{*  *String[] names=new String[3];*  *i++;//11*  *if (names[1].equals("john")) {*  *System.out.println(names[1]);*  *} else {*  *names[3]="lucy";*  *}*  *return i;*  *}catch(ArrayIndexOutOfBoundsException e){*  *return ++i;*  *}catch(NullPointerException e){*  *return ++i;//12*  *}*  *finally{*  *i=999;*  *//return i++;*  *}*  *}*  *public static void main(String[] args){*  *System.out.println(method()); // 12*  *}*  *public static String output="";*  *public static void foo(int i){*  *try{*  *if(i==1)*  *throw new Exception();*  *output+="1";*  *}*  *catch(Exception e){*  *output+="2";*  *return;*  *}*  *finally{*  *output+="3";*  *}*  *output+="4";*  *}*    *public static void main(String args[]){*  *foo(0);*  *System.out.println(output);//134 134*  *foo(1);*  *System.out.println(output);//134234 13423*  *}* |

|  |
| --- |
| *import java.io.\*;*  *public class Test7\_3{*  *public static void main(String[] args){*  *FileInputStream in=new FileInputStream("myfile.txt");*  *int b;*  *b = in.read();*  *while(b!= -1) {*  *System.out.print((char)b);*  *b = in.read();*  *}*  *in.close();*  *}*  *}*  */\**  *\* 需求：编写应用程序EcmDef.java，接收命令行的两个参数，计算两数相除。*  *对数据类型不一致(NumberFormatException)、缺少命令行参数(ArrayIndexOutOfBoundsException、两个数不能是负数、除0(ArithmeticException) 进行异常处理。*  *提示：*  *(1)在主类(EcmDef)中定义异常方法(ecm)完成两数相除功能。*  *(2)在main()方法中使用异常处理语句进行异常处理。*  *(3)运行时接受参数 java EcmDef 20 10*  *//args[0]=“20” args[1]=“10”*  *(4)Interger类的static方法parseInt(String s)将s转换成对应的int值。如int a=Interger.parseInt(“314”); //a=314;*  *\*/*  *public class TestException3 {*  *public static void main(String[] args) {*  *try {*  *double a = Double.parseDouble(args[0]);*  *double b = Double.parseDouble(args[1]);*  *if(a<0||b<0){*  *//手动生成异常对象*  *throw new NegativeException("参数不能是负数！");*  *}*  *double ecm = ecm(a,b);*  *System.out.println(ecm);*  *} catch (NumberFormatException e) {*  *System.out.println("参数必须为有效的 数值型！");*  *}catch(ArithmeticException e){*  *System.out.println("第二个参数不能是0！");*  *}catch(ArrayIndexOutOfBoundsException e){*  *System.out.println("必须传入至少两个参数！");*  *}catch(NegativeException e){*  *System.out.println(e.getMessage());*  *}*  *}*  *public static double ecm(double a,double b){*  *return a/b;*  *}*  *}*  *class NegativeException extends RuntimeException{*  *private static final long serialVersionUID = 1L;*  *public NegativeException() {*  *super();*  *}*  *public NegativeException(String message) {*  *super(message);*  *}*  *}* |

## 3.2 throws + 异常类型 抛出异常：

如果一个方法中的语句执行时可能生成某种异常，但是并不能确定如何处理这种异常，则此方法应显示地声明抛出异常，表明该方法将不对这些异常进行处理，而由该方法的调用者负责处理。

在方法声明中用throws语句可以声明抛出异常的列表，throws后面的异常类型可 以是方法中产生的异常类型，也可以是它的父类。

如果程序中出现了异常，而没有做任何处理，默认的处理方式是throws抛异常；

1）语法：

|  |
| --- |
| *[修饰符] 返回值类型 方法名(参数列表) throws 异常类型{*  *//可能会出现异常的代码块；*  *}* |

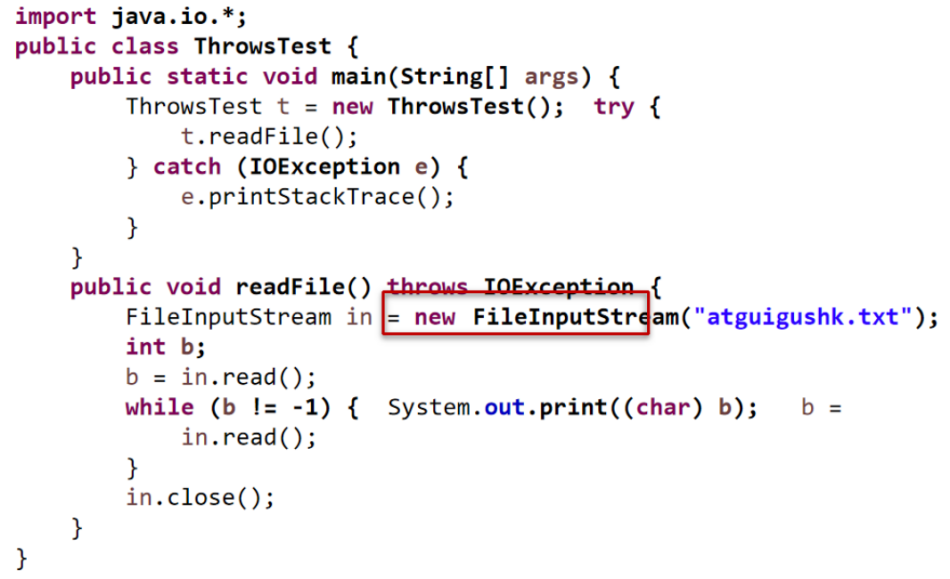
举例：

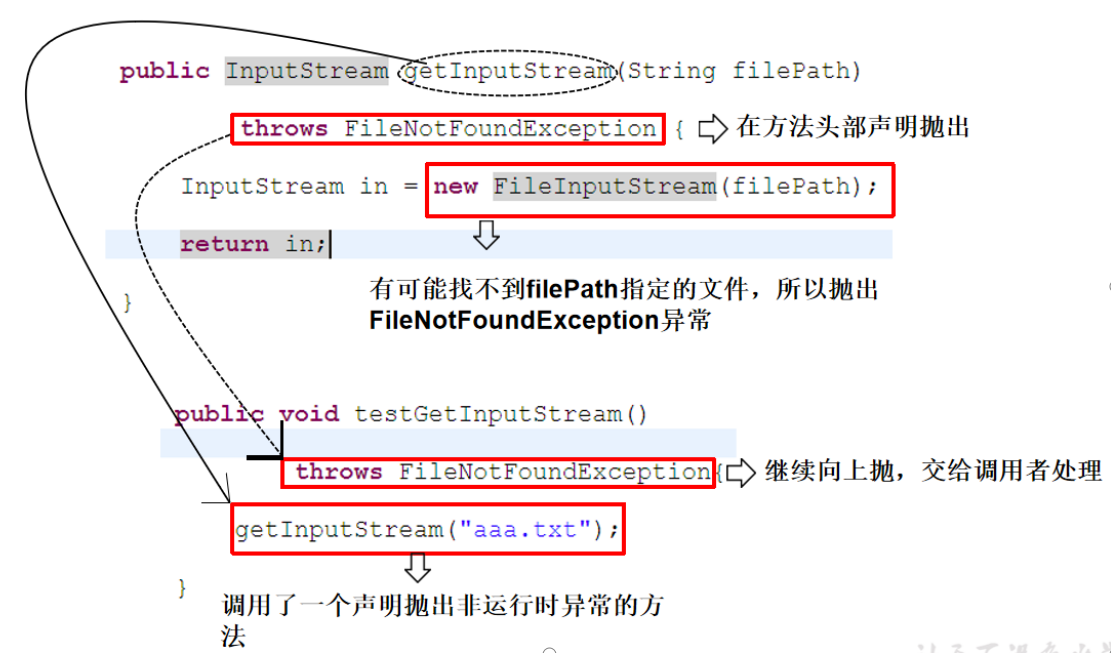
|  |
| --- |
| *public void readFile(String file) throws FileNotFoundException {*  *……*  *// 读文件的操作可能产生FileNotFoundException类型的异常*  *FileInputStream fis = new FileInputStream(file);*  *..……*  *}* |

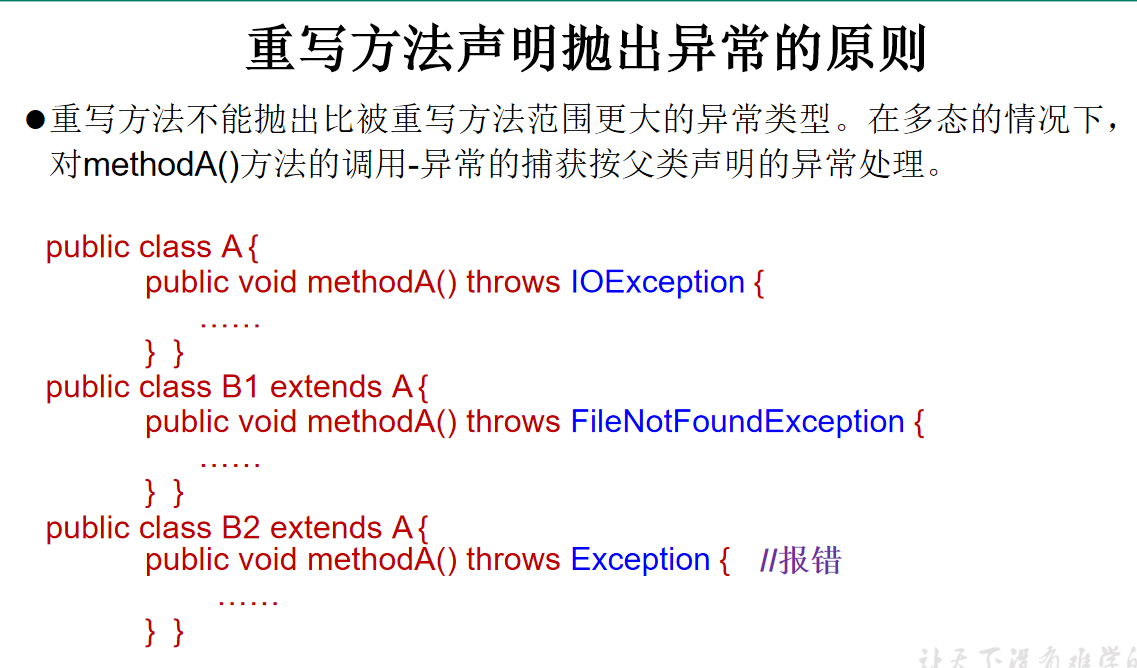
注意：

抛出的异常类型应该写方法容易出现的具体的异常类型或其父类类型；

将方法中的异常抛给调用方，调用方可以有两种处理异常的方式，如果继续往上抛，则可以一直抛到JVM，则JVM会采用默认的处理方式（打印错误的堆栈信息）；







2）案例：

|  |
| --- |
| *public class TestException5 {*  *public static void main(String[] args ){*  *method1();*  *}*  *private static void method1() {*  *method2();*  *}*  *private static void method2() {*  *method3();*  *}*  *private static void method3() {*  *method4();*  *}*  *private static void method4(){*  *int[] arr = {1,2,3};*  *System.out.println(arr[3]);//下标越界异常*  *}*  *}* |
| */\*\**  *\* 此类用于演示子类重写父类方法时对异常类型的要求*  *\* 要求：子类方法抛出的异常类型要么相同，要么为子类类型*  *\*/*  *public class TestException6 {}*  *class Son extends Father{*  *@Override*  *public void method() throws NullPointerException {*  *super.method();*  *}*  *}*  *class Father{*  *public void method() throws RuntimeException{*  *}*  *}* |

## 3.3 手动抛出异常

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，也可根据需要使用人工创建并抛出。

首先要生成异常类对象，然后通过throw语句实现抛出操作(提交给Java运行环境)。

|  |
| --- |
| *IOException e = new IOException();*  *throw e;* |

可以抛出的异常必须是Throwable或其子类的实例。下面的语句在编译时将会产生语法错误：

|  |
| --- |
| *throw new String("want to throw");* |

## 3.4 用户自定义异常类：

一般地，用户自定义异常类都是RuntimeException的子类。自定义异常类通常需要编写几个重载的构造器。自定义异常需要提供serialVersionUID；自定义的异常通过throw抛出。自定义异常最重要的是异常类的名字，当异常出现时，可以根据名字判断异常类型。

|  |
| --- |
| */\*\**  *\* 此类用于演示自定义异常类*  *\* 定义步骤：*  *\* 1、定义类XXException 继承RuntimeException（运行异常）或Exception（编译异常）*  *\* 2、*  *\* 3、类体中，设计无参和有参构造器*  *\* 生成异常对象：*  *\* 1、手动生成：throw new 异常类型(参数); 针对于任何异常类型*  *\* 2、系统自动生成的仅仅针对于系统定义好的异常类型*  *\*/*  *class AgeIllegalException extends RuntimeException{*  *private static final long serialVersionUID = 1L;*  *public AgeIllegalException(String msg){*  *super(msg);*  *}*  *public AgeIllegalException(){}*  *}*  *public class TestException7 {*  *public static void main(String[] args) {*  *Scanner input = new Scanner(System.in);*  *System.out.println("年龄：");*  *int age = input.nextInt();*  *if(age>120 || age<0){*  *//生成一个异常对象*  *//第一种方式：*  *// AgeIllegalException ae = new AgeIllegalException("年龄不合法！");*  *// thorw ae; 必须抛出，否则没有意义*  *//第二种方式：*  *try {*  *throw new RuntimeException("年龄不合法！");*  *} catch (Exception e) {*  *// e.printStackTrace();*  *System.out.println("出错了，信息："+e.getMessage());*  *}*  *}*  *System.out.println("年龄是："+age);*  *}*  *}*    */\*\**  *\* 需求：实现登录验证*  *输入用户名 和密码，判断是否 为“john”和“000”*  *如果用户名输入错误，生成错误信息为“用户名不存在”的异常对象*  *如果用户名正确，密码输入错误，生成错误信息为“密码不正确”的异常对象*  *\*/*  *public class TestException4 {*  *public static void main(String[] args) {*  *Scanner input = new Scanner(System.in);*  *System.out.print("请输入用户名：");*  *String name = input.next();*  *System.out.print("请输入密码：");*  *String pwd = input.next();*  *try {*  *verifyUser(name, pwd);*  *System.out.println("登录成功");*  *} catch (Exception e) {*  *System.out.println(e.getMessage());*  *}*  *}*  */\*\**  *\* 功能：验证是否登录成功*  *\* @param name*  *\* @param pwd*  *\* @return 如果登录成功，则返回true，否则返回 异常信息*  *\*/*  *public static void verifyUser(String name,String pwd) throws LoginException{*  *if(!name.equals("john"))*  *//throw new runtimeException("用户名不存在！");*  *throw new LoginException("用户名不存在！");*  *if(!pwd.equals("000"))*  *//throw new runtimeException("密码不正确！");*  *throw new LoginException("密码不正确！");*  *}*  *}*  *class LoginException extends Exception{*  *private static final long serialVersionUID = 1L;*  *public LoginException() {*  *super();*  *}*  *public LoginException(String message) {*  *super(message);*  *}*  *}* |

