**异常**

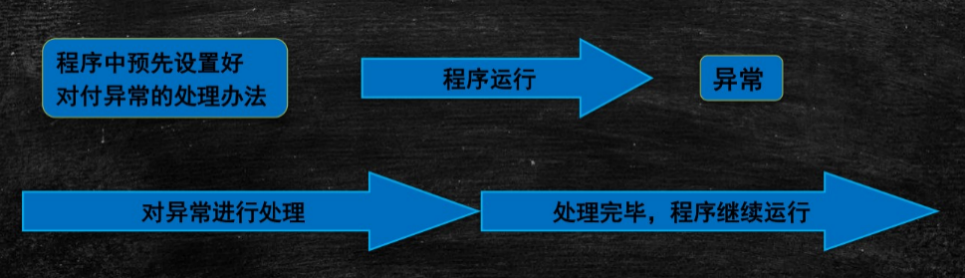
在程序运行过程中出现的不正常情况

相同的代码在运行的时候，根据输入的参数或者操作的不同，有可能不发生异常，有可能会发生异常。应该在写代码的过程中尽可能地保证代码的正确性，不要到处都是bug

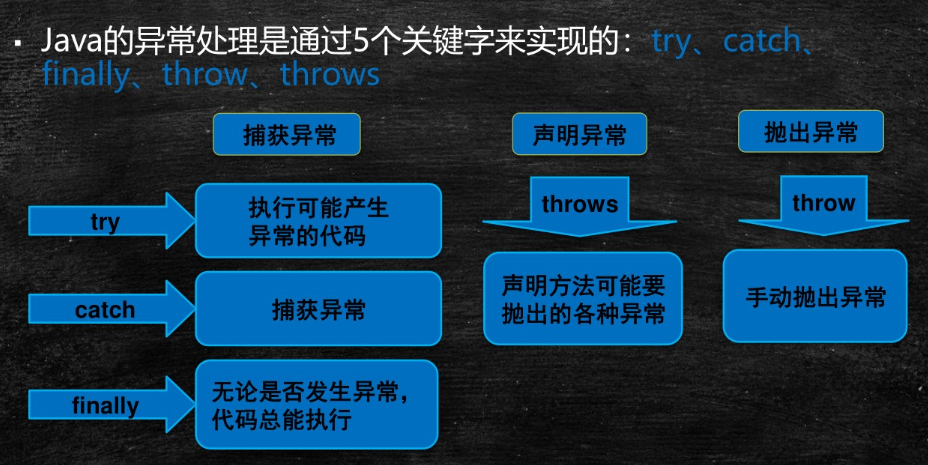
**public class** TestException {  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
 System.***out***.print(**"请输入被除数:"**);  
 **int** num1 = in.nextInt();  
 System.***out***.print(**"请输入除数:"**);  
 **int** num2 = in.nextInt();  
 System.***out***.println(String.*format*(**"%d / %d = %d"**,  
 num1, num2, num1/ num2));  
 System.***out***.println(**"感谢使用本程序！"**);  
 }  
}

如果输入的被除数为0，就会报错：Exception in thread "main" java.lang.ArithmeticException: / by zero。（并且最后的“感谢使用本程序”不会被打印，即出现报错之后程序被中断，后面的代码不会被实行。）可以使用if-else来进行判断，给出报错信息，但是这样要判断的东西多，会显得代码臃肿，而且不利于维护修改，可读性比较差，也很难堵住所有漏洞。因此推荐使用异常机制来处理程序运行中出现的问题

异常处理机制



Java中如何进行异常处理



程序在运行过程中如果出现了问题，会导致后面的代码无法正常执行，而使用异常机制之后可以对异常情况进行处理，同时后续代码会继续执行，不会中断整个程序

**异常处理的方式：**

* **捕获异常**

try{代码逻辑}catch(Exception e){异常处理}

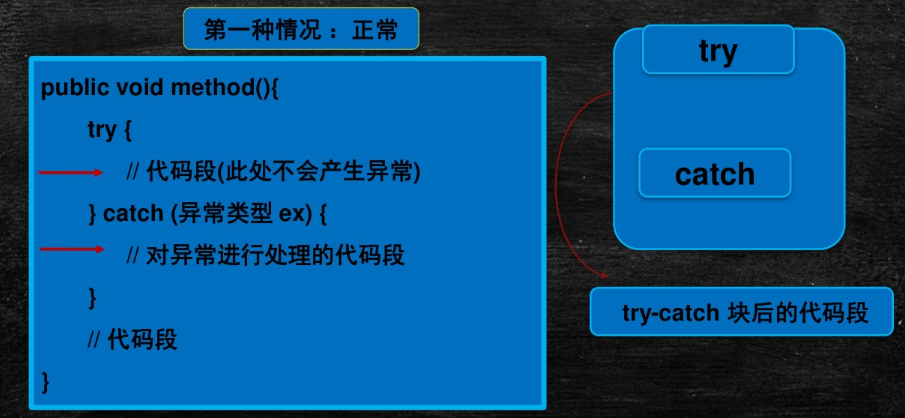
try{代码逻辑}catch(具体的异常Exception e){异常处理}catch(具体的异常);

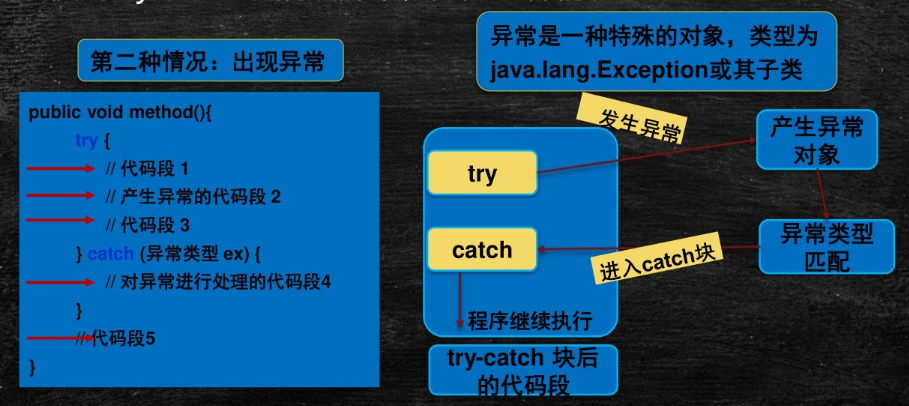
可以针对每一种具体的异常做相应的更丰富的处理

**public class** TestException {  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
 **try** {  
 System.***out***.print(**"请输入被除数:"**);  
 **int** num1 = in.nextInt();  
 System.***out***.print(**"请输入除数:"**);  
 **int** num2 = in.nextInt();  
 System.***out***.println(String.*format*(**"%d / %d = %d"**,  
 num1, num2, num1 / num2));  
 }**catch** (Exception e){  
 System.***out***.println(**"出现异常"**);  
 }  
 System.***out***.println(**"感谢使用本程序！"**);  
 }  
}

写了如上有双下划线的部分以后如果除数写成0或者输入的不是数字的情况之下就会报错：出现异常，并且最后的“感谢使用本程序”还是会被打印

使用try-catch块捕获异常分为三种情况：





**public class** TestException {  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
 **try** {  
 System.***out***.print(**"请输入被除数:"**);  
 **int** num1 = in.nextInt();  
 System.***out***.print(**"请输入除数:"**);  
 **int** num2 = in.nextInt();  
 System.***out***.println(String.*format*(**"%d / %d = %d"**,  
 num1, num2, num1 / num2));  
 System.***out***.println(**"前面没有出现异常"**);  
 }**catch** (Exception e){  
 System.***out***.println(**"出现异常"**);  
 }  
 System.***out***.println(**"感谢使用本程序！"**);  
 }  
}

比如我现在除数写成0，或者写非数值型类型的数据，那么try{ }里出现异常的代码后面的内容将不执行，即划横线的部分没有被执行，执行结果为：

出现异常

感谢使用本程序！



**public class** TestException {  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
 **try** {  
 System.***out***.print(**"请输入被除数:"**);  
 **int** num1 = in.nextInt();  
 System.***out***.print(**"请输入除数:"**);  
 **int** num2 = in.nextInt();  
 System.***out***.println(String.*format*(**"%d / %d = %d"**,  
 num1, num2, num1 / num2));  
 System.***out***.println(**"前面没有出现异常"**);  
 }**catch** (Exception e){  
 System.***out***.println(**"出现异常"**);  
 e.printStackTrace();  
 }  
 System.***out***.println(**"感谢使用本程序！"**);  
 }  
}

加上双下划线的代码后，如果哪一行出现什么样的异常代码执行后会显示出来，并且后面的代码也会实行，比如除数输入0，执行结果为：

出现异常

java.lang.ArithmeticException: / by zero

at com.mashibing.TestException.main(TestException.java:15)

感谢使用本程序！

比如输入非数值类型的信息后的执行结果为：

出现异常

java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:939)

at java.base/java.util.Scanner.next(Scanner.java:1594)

at java.base/java.util.Scanner.nextInt(Scanner.java:2258)

at java.base/java.util.Scanner.nextInt(Scanner.java:2212)

at com.mashibing.TestException.main(**TestException.java:11**)

感谢使用本程序！

在异常的处理过程中，不要只是简单的输出错误，要尽可能地将详细的异常信息进行输出。

**e.printStackTrace();** 打印异常的堆栈信息，可以从异常信息的最后一行开始追踪，寻找自己编写的java类，上面例子中粗体的java:11行的部分

**public class** TestException {  
 **public static void** main(String[] args) {  
 Scanner in = **new** Scanner(System.***in***);  
 **try** {  
 System.***out***.print(**"请输入被除数:"**);  
 **int** num1 = in.nextInt();  
 System.***out***.print(**"请输入除数:"**);  
 **int** num2 = in.nextInt();  
 System.***out***.println(String.*format*(**"%d / %d = %d"**,  
 num1, num2, num1 / num2));  
 System.***out***.println(**"前面没有出现异常"**);  
}**catch** (ArithmeticException e){  
 System.out.println("数字异常，除数不能是0");  
 e.printStackTrace();  
 }catch (InputMismatchException e){  
 System.out.println("输入参数值类型不匹配");  
 e.printStackTrace();  
 }  
 System.***out***.println(**"感谢使用本程序！"**);  
 }  
}

这时除数输入0的时候执行结果为：

数字异常，除数不能是0

java.lang.ArithmeticException: / by zero

at com.mashibing.TestException.main(TestException.java:16)

感谢使用本程序！

输入非数值类型的内容时执行结果为：

输入参数值类型不匹配

java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:939)

at java.base/java.util.Scanner.next(Scanner.java:1594)

at java.base/java.util.Scanner.nextInt(Scanner.java:2258)

at java.base/java.util.Scanner.nextInt(Scanner.java:2212)

at com.mashibing.TestException.main(TestException.java:12)

感谢使用本程序！

注意当使用多重catch的时候一定要注意相关异常的顺序，将子类放在最前面的catch，父类放在最后面的catch

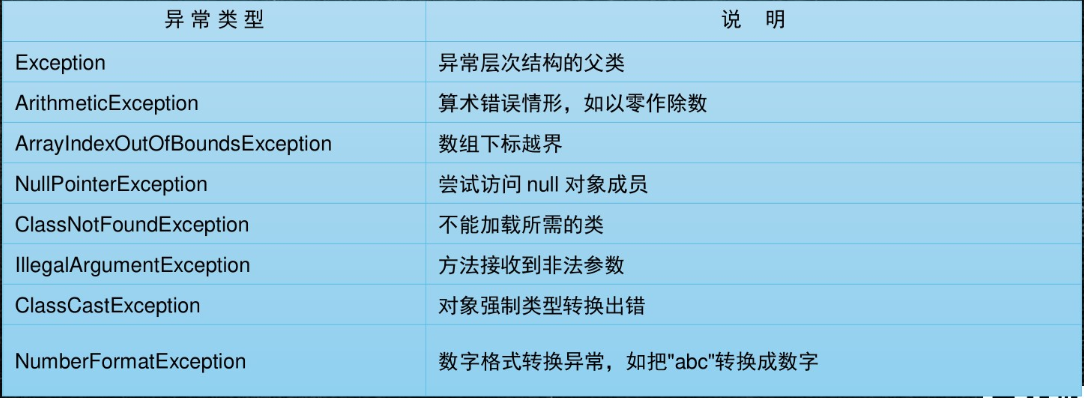
}**catch** (Exception e){  
 System.***out***.println(**"出现异常"**);  
}**catch** (ArithmeticException e){  
 System.***out***.println(**"数字异常，除数不能是0"**);  
 e.printStackTrace();  
}**catch** (InputMismatchException e){  
 System.***out***.println(**"输入参数值类型不匹配"**);  
 e.printStackTrace();  
}   
System.***out***.println(**"感谢使用本程序！"**);

波浪线的部分报错，因为Exception是ArithmeticException和Input Mismatch Exception的父类。如果将这个父类的catch放在最后一个catch的位置就不会报错，正常执行

总结执行过程中可能出现的情况：

1. 正常执行，只执行try中的代码
2. 遇到异常情况，会处理try中异常代码之前的逻辑，后面的逻辑不会执行，最后会执行catch中的代码
3. 使用多重catch的时候，会遇到异常子类不匹配的情况，此时依然会报错，因此建议在catch的最后将所有异常的父类写上

常见的异常类型：



**public class** FinallyDemo {  
 **public static void** main(String[] args) {  
 **try** {  
 System.***out***.println(1 / 0);  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }}  
}

执行结果：

java.lang.ArithmeticException: / by zero

at FinallyDemo.main(FinallyDemo.java:4)

**public class** FinallyDemo {  
 **public static void** main(String[] args) {  
 **try** {  
 System.***out***.println(1 / 0);  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }**finally** {  
 System.***out***.println(**"我是finally处理块"**);  
 }  
 }  
}

执行结果：

java.lang.ArithmeticException: / by zero

at FinallyDemo.main(FinallyDemo.java:4)

我是finally处理块

**public class** FinallyDemo {  
 **public static void** main(String[] args) {  
 **try** {  
 System.***out***.println(1 / 10);  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }**finally** {  
 System.***out***.println(**"我是finally处理块"**);  
 }  
 }  
}

执行结果：

0

我是finally处理块

在程序运行过程中，如果处理异常的部分包含finally的处理，那么无论代码是否发生异常，finally中的代码都会执行

finally中包含哪些处理逻辑：

1. IO流的关闭操作一般设置在finally中
2. 数据库的链接关闭操作设置在finally中

* **情况一**（try中有return，finally中没有return）：

**public class** FinallyDemo {  
 **public static void** main(String[] args){  
 System.***out***.println(*test*());  
 }  
  
 **private static int** test(){  
 **int** num = 10;  
 **try**{  
 System.***out***.println(**"try"**);  
 **return** num += 80;  
 }**catch**(Exception e){  
 System.***out***.println(**"error"**);  
 }**finally**{  
 **if** (num > 20){  
 System.***out***.println(**"num>20 : "** + num);  
 }  
 System.***out***.println(**"finally"**);  
 }  
 **return** num;  
 }  
}

执行结果：

try

num>20 : 90

finally

90

* **情况二**（try和finally中均有return）：

**public class** FinallyDemo {  
 **public static void** main(String[] args){  
 System.***out***.println(*test*());  
 }  
  
 **private static int** test(){  
 **int** num = 10;  
 **try**{  
 System.***out***.println(**"try"**);  
 **return** num += 80;  
 }**catch**(Exception e){  
 System.***out***.println(**"error"**);  
 }**finally**{  
 **if** (num > 20){  
 System.***out***.println(**"num>20 : "** + num);  
 }  
 System.***out***.println(**"finally"**);  
 num = 100;  
 **return** num;  
 }  
 }  
}

执行结果：

try

num>20 : 90

finally

100

分析：try中的return语句同样被拆分了，finally中的return语句先于try中的return语句执行，因而try中的return被”覆盖“掉了，不再执行

* **情况三**（finally中改变返回值num）：
* **public class** FinallyDemo {  
   **public static void** main(String[] args){  
   System.***out***.println(*test*());  
   }  
    
   **private static int** test(){  
   **int** num = 10;  
   **try**{  
   System.***out***.println(**"try"**);  
   **return** num;  
   }**catch**(Exception e){  
   System.***out***.println(**"error"**);  
   }**finally**{  
   **if** (num > 20){  
   System.***out***.println(**"num>20 : "** + num);  
   }  
   System.***out***.println(**"finally"**);  
   num = 100;  
   }  
   **return** num;  
   }  
  }

执行结果：

try

finally

10

分析：虽然在finally中改变了返回值num，但因为finally中没有return该num的值，

因此在执行完finally中的语句后，test（）函数会得到try中返回的num的值，而try中的num的值依然是程序进入finally代码块前保留下来的值，因此得到的返回值为10

* **情况四**（将num的值包装在Num类中）：
* **public class** FinallyDemo {  
   **public static void** main(String[] args){  
   System.***out***.println(*test*().**num**);  
   }  
    
   **private static** Num test(){  
   Num number = **new** Num();  
   **try**{  
   System.***out***.println(**"try"**);  
   **return** number;  
   }**catch**(Exception e){  
   System.***out***.println(**"error"**);  
   }**finally**{  
   **if** (number.**num** > 20){  
   System.***out***.println(**"number.num>20 : "** + number.**num**);  
   }  
   System.***out***.println(**"finally"**);  
   number.**num** = 100;  
   }  
   **return** number;  
   }  
  }  
    
  **class** Num{  
   **public int num** = 10;  
  }

执行结果：

try

finally

100

分析：同样是在finally中改变了返回值num的值，在情况三中，并没有被try中的return返回（test（）方法得到的不是100），但在这里却被try中的return语句返回了

**总结：**

try语句在返回前，将其他所有的操作执行完，保留好要返回的值，而后转入执行finally中的语句，而后分为以下三种情况：

    情况一：如果finally中有return语句，则会将try中的return语句”覆盖“掉，直接执行finally中的return语句，得到返回值，这样便无法得到try之前保留好的返回值。

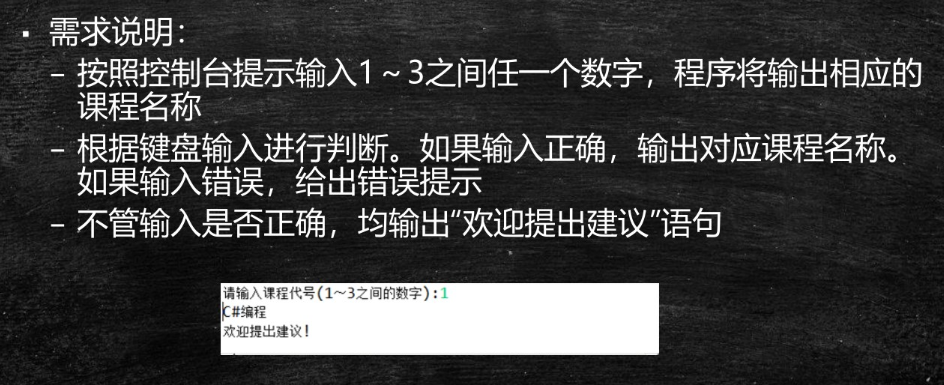
    情况二：如果finally中没有return语句，也没有改变要返回值，则执行完finally中的语句后，会接着执行try中的return语句，返回之前保留的值。

    情况三：如果finally中没有return语句，但是改变了要返回的值，这里有点类似与引用传递和值传递的区别，分以下两种情况，：

        1）如果return的数据是基本数据类型或文本字符串，则在finally中对该基本数据的改变不起作用，try中的return语句依然会返回进入finally块之前保留的值。

        2）如果return的数据是引用数据类型，而在finally中对该引用数据类型的属性值的改变起作用，try中的return语句返回的就是在finally中改变后的该属性的值。

* 课堂练习：根据编号输出课程名称



**public class** CourseDemo {  
 **public static void** main(String[] args) {  
 Scanner scanner = **new** Scanner(System.***in***);  
 System.***out***.println(**"请输入课程编号："**);  
 **try**{  
 **int** i = scanner.nextInt();  
 **if**(i==0){  
 System.***out***.println(**"java"**);  
 }**else if**(i==1){  
 System.***out***.println(**"scala"**);  
 }**else if**(i==2){  
 System.***out***.println(**"python"**);  
 }**else**{  
 System.***out***.println(**"请输入0或1或2"**);  
 }  
 }**catch** (InputMismatchException e){  
 e.printStackTrace();  
 }**finally** {  
 System.***out***.println(**"欢迎选课"**);  
 }  
 }  
}

输入数字以外的内容（比如abc）时提示：

java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:939)

at java.base/java.util.Scanner.next(Scanner.java:1594)

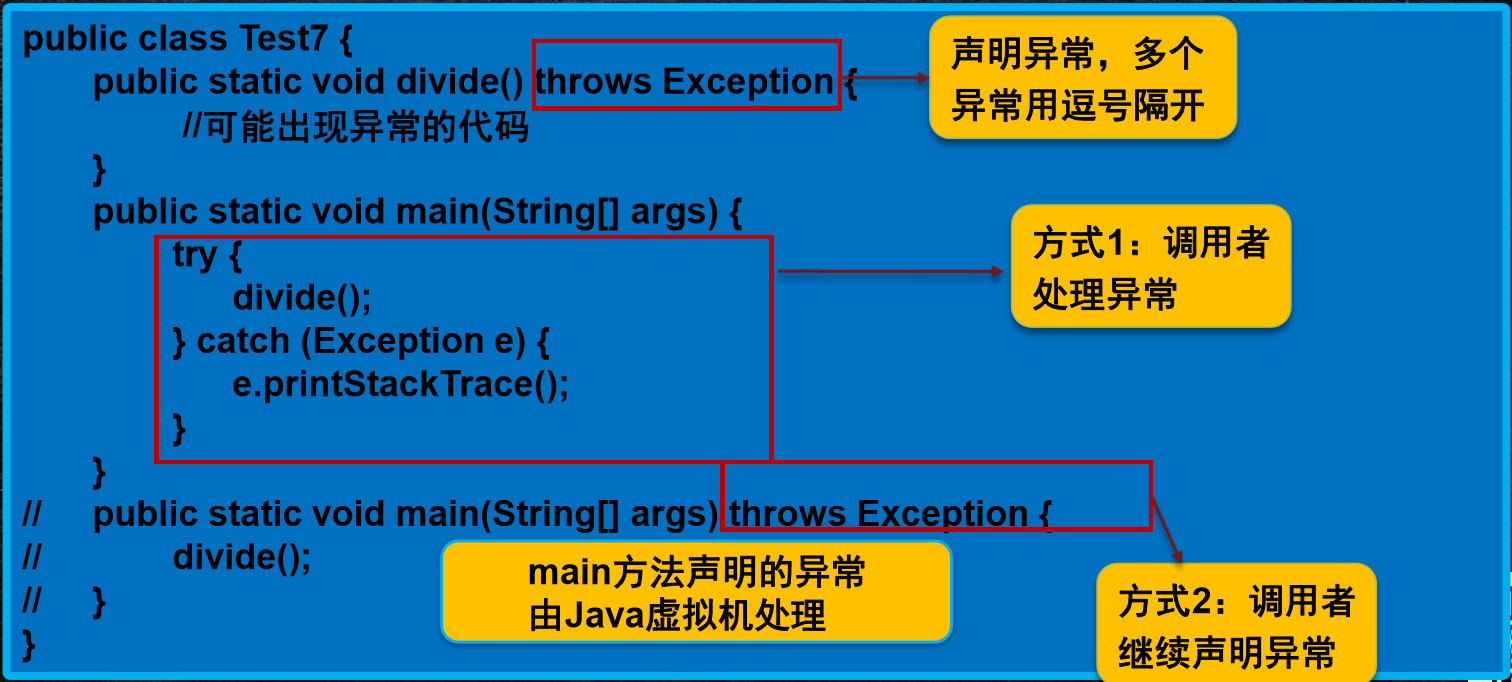
at java.base/java.util.Scanner.nextInt(Scanner.java:2258)

at java.base/java.util.Scanner.nextInt(Scanner.java:2212)

at CourseDemo.main(CourseDemo.java:9)

欢迎选课

* **声明异常throws**



**public class** Exception2 {  
 **public static void** main(String[] args) **throws** Exception{  
 *test*();  
 System.***out***.println(**"hehe"**);  
 }  
  
 **public static void** test()**throws** Exception{  
 System.***out***.println(1/0);  
 }  
}

执行结果：

Exception in thread "main" java.lang.ArithmeticException: / by zero

at com.mashibing.Exception2.test(Exception2.java:10)

at com.mashibing.Exception2.main(Exception2.java:5)

**public class** Exception2 {  
 **public static void** main(String[] args){  
 **try** {  
 *test*();  
 }**catch** (Exception e){  
 e.printStackTrace();  
 }  
 System.***out***.println(**"hehe"**);  
 }  
  
 **public static void** test()**throws** Exception{  
 System.***out***.println(1/0);  
 }  
}

执行结果：

java.lang.ArithmeticException: / by zero

at com.mashibing.Exception2.test(Exception2.java:14)

at com.mashibing.Exception2.main(Exception2.java:6)

hehe

在异常情况出现的时候，可以使用try-catch-finally的方式对异常进行处理，除此之外，可以将异常向外抛出，由外部进行处理

在方法调用过程中，可能存在N多个方法之间的调用，此时假如每个方法中都包含了异常情况，那么就需要在每个方法中都进行try-catch，另外一种比较简单的方式就是在方法的最外层调用处理一次即可，因此使用throws的方式对所有执行过程中的方法出现的异常进行统一集中处理

如何判断是使用throws还是使用try-catch？

最稳妥的方式是在每个方法中都进行异常的处理，偷懒的方式是判断在整个调用的过程中，外层的调用方法是否有对异常的处理，如果有，直接使用throws，如果没有就使用try-catch

**public class** Exception2 {  
 **public static void** main(String[] args){  
 **try** {  
 *test4*();  
 }**catch** (Exception e){  
 e.printStackTrace();  
 System.***out***.println(**"出现异常"**);  
 }  
 }  
  
 **public static void** test1() **throws** Exception{  
 System.***out***.println(1/0);  
 }  
 **public static void** test2() **throws** Exception{  
 *test1*();  
 System.***out***.println(100/0);  
 }  
 **public static void** test3() **throws** Exception{  
 *test2*();  
 }  
 **public static void** test4() **throws** Exception{  
 *test3*();  
 }  
}

执行结果：

java.lang.ArithmeticException: / by zero

at com.mashibing.Exception2.test1(Exception2.java:14)

at com.mashibing.Exception2.test2(Exception2.java:17)

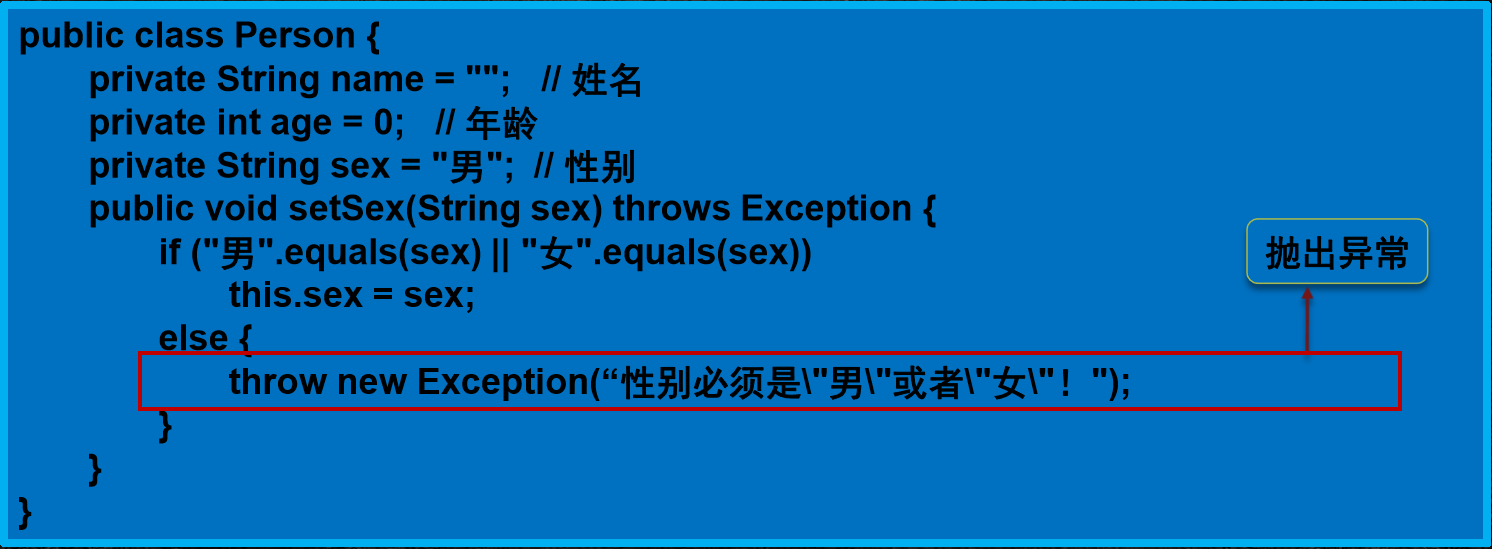
at com.mashibing.Exception2.test3(Exception2.java:21)

at com.mashibing.Exception2.test4(Exception2.java:24)

at com.mashibing.Exception2.main(Exception2.java:6)

出现异常

* **抛出异常throw**



**public class** Exception2 {  
 **public static void** main(String[] args){  
 **try** {  
 *show*();  
 }**catch** (Exception e){  
 e.printStackTrace();

System.***out***.println(**"出现错误"**);  
 }  
 }  
  
 **public static void** show() **throws** Exception{  
 String gender = **"1234"**;  
 **if**(gender.equals(**"man"**)){  
 System.***out***.println(**"man"**);  
 }**else if**(gender.equals(**"woman"**)){  
 System.***out***.println(**"woman"**);  
 }**else**{  
 **throw new** Exception(**"性别出现错误"**);  
 }  
 }

}

执行结果：

java.lang.Exception: 性别出现错误

at com.mashibing.Exception2.show(Exception2.java:19)

at com.mashibing.Exception2.main(Exception2.java:6)

出现错误

但是↑写抛出的异常时Exception（异常父类），不太好。这时可以自定义异常↓

新建的异常类：

**public class** GenderException **extends** Exception{  
  
 **public** GenderException(){  
 System.***out***.println(**"性别异常"**);  
 }  
  
 **public** GenderException(String msg){  
 System.***out***.println(msg);  
 }  
}

情况1：

**public class** Exception2 {  
 **public static void** main(String[] args){  
 **try** {  
 *show*();  
 }**catch** (GenderException e){  
 e.printStackTrace();  
 System.***out***.println(**"出现错误"**);  
 }  
 }  
  
 **public static void** show() **throws** GenderException{  
 String gender = **"1234"**;  
 **if**(gender.equals(**"man"**)){  
 System.***out***.println(**"man"**);  
 }**else if**(gender.equals(**"woman"**)){  
 System.***out***.println(**"woman"**);  
 }**else**{  
 **throw new** GenderException();  
 }  
 }  
}

执行结果：

性别异常

com.mashibing.GenderException

at com.mashibing.Exception2.show(Exception2.java:20)

at com.mashibing.Exception2.main(Exception2.java:6)

出现错误

情况2：

**public class** Exception2 {  
 **public static void** main(String[] args){  
 **try** {  
 *show*();  
 }**catch** (GenderException e){  
 e.printStackTrace();  
 System.***out***.println(**"出现错误"**);  
 }  
 }  
  
 **public static void** show() **throws** GenderException{  
 String gender = **"1234"**;  
 **if**(gender.equals(**"man"**)){  
 System.***out***.println(**"man"**);  
 }**else if**(gender.equals(**"woman"**)){  
 System.***out***.println(**"woman"**);  
 }**else**{  
 **throw new** GenderException(**"gender is wrong"**);  
 }  
 }  
}

执行结果：

gender is wrong

com.mashibing.GenderException

at com.mashibing.Exception2.show(Exception2.java:20)

at com.mashibing.Exception2.main(Exception2.java:6)

出现错误

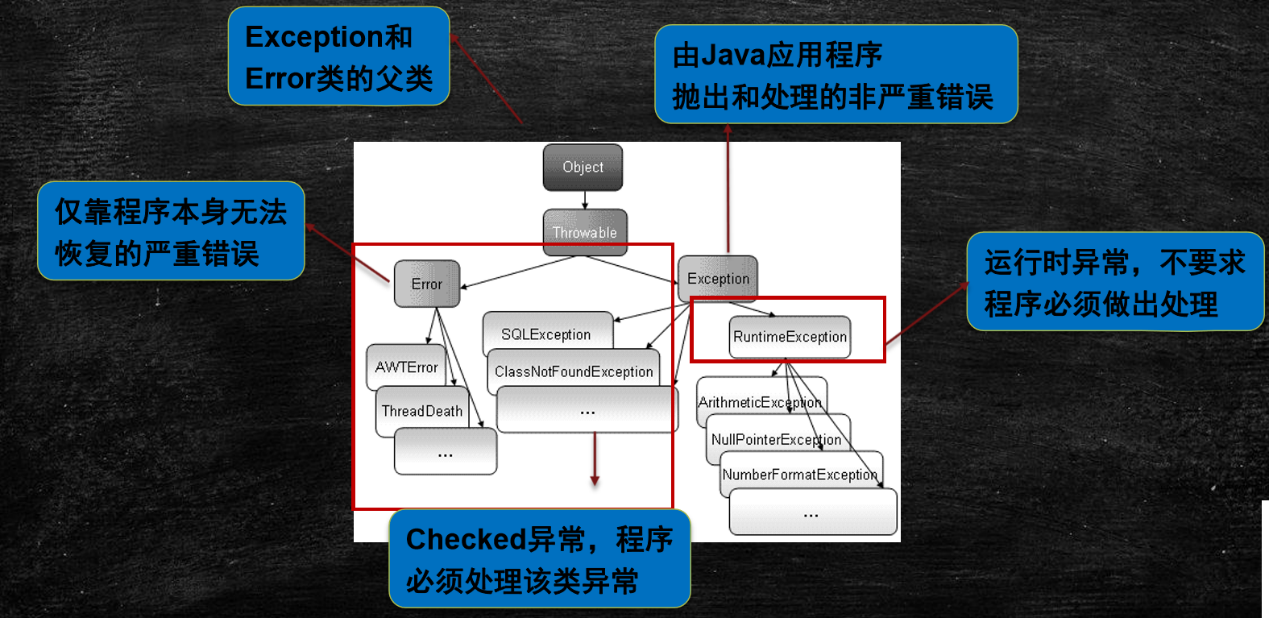
在java的API中提供了非常丰富的异常类，但是在某种情况下不太满足我们的需求，此时需要自定义异常，步骤如下：

继承Exception类 → 自定义实现构造方法 → 需要使用的时候，使用throw new自定义异常的名称

什么时候需要自定义异常？

一般情况下不需要，但是在公司要求明确，或者要求异常格式规范统一的时候时必须要自己实现的

异常的分类：



Checked异常即编译异常，比如编译过程中的红色波浪线标注的部分

RuntimeException异常，编译时没有提示错误，但是运行中可能会出现错误的部分