异常

今日学习内容和目标:

章节	知识点	掌握程度
异常	异常概念	理解
	异常处理机制	理解
	异常处理	熟练掌握
异常分类	检查时异常和运行时异常	掌握
声明异常	throws	熟练掌握
抛出异常	throw	熟练掌握
自定义异常	自定义异常	熟练掌握

1 传统处理异常的方式(了解)

```
public class ExceptionDemo {
    public static void main(String[] args) {
        // 需求：控制台输入两个整数，求两个数的商
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入被除数:");
        if (sc.hasNextInt()) {
            int num1 = sc.nextInt();
            System.out.println("请输入除数:");
            int num2 = 0;
            if (sc.hasNextInt()) {
                num2 = sc.nextInt();
                if (0 == num2) {
                    System.out.println("除数不能为 0");
                } else {
                    int r = num1 / num2;
                    System.out.println(r);
                }
            } else {
                System.out.println("除数输入有误！");
            }
        } else {
            System.out.println("被除数输入有误！");
        }
    }
}
```

以上是传统异常的处理方式！你发现什么问题?

2 异常概述和异常处理机制（理解）

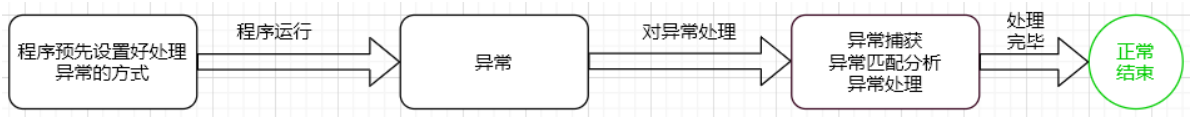
2.1 异常(exception)概述

异常就是程序在运行时出现的意外的，不正常的情况。

若异常产生后没有正确的处理，会导致**程序的中断**，程序不继续执行,以致造成损失。

2.2 异常处理机制

所以我们在开发中要一套机制来处理各种可能会发生的异常，并对其作出正确的处理，确保程序的正常执行。这种机制称为**异常处理机制**，java语言是最先提供异常处理机制的。



异常处理机制可以引导程序向正确的方向运行，不至于崩溃。

3 异常处理(掌握)

java异常处理包含两种代码块。一种是try...catch,一种是try...catch...finally.

3.1 try...catch(掌握)

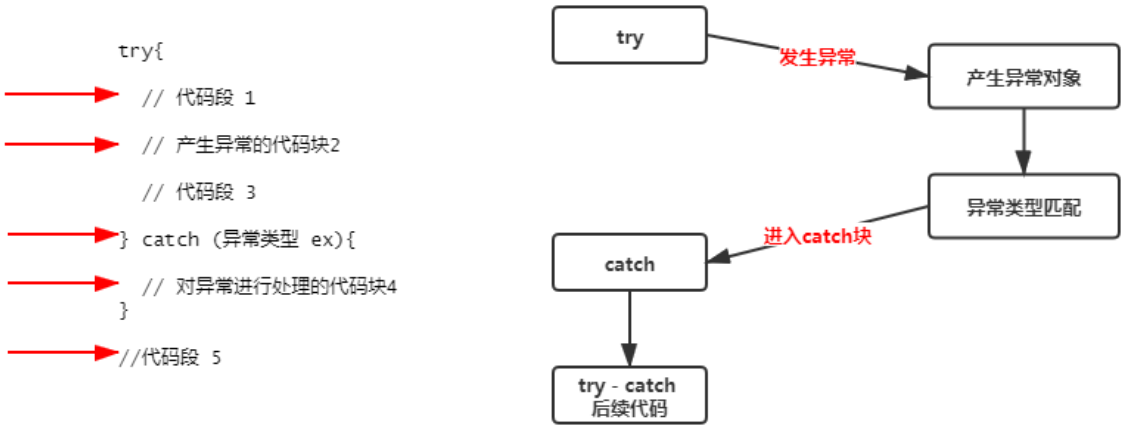
把有可能产生异常的代码放到try中，如果产生异常由catch代码块处理。语法

```
try {
    // 可能产生异常的代码块
} catch(异常类型 e){
    // 异常处理
}
// 后面如果还有代码，无论try中的代码有没有异常，这里都会继续执行
```

try...catch执行有三种情况：

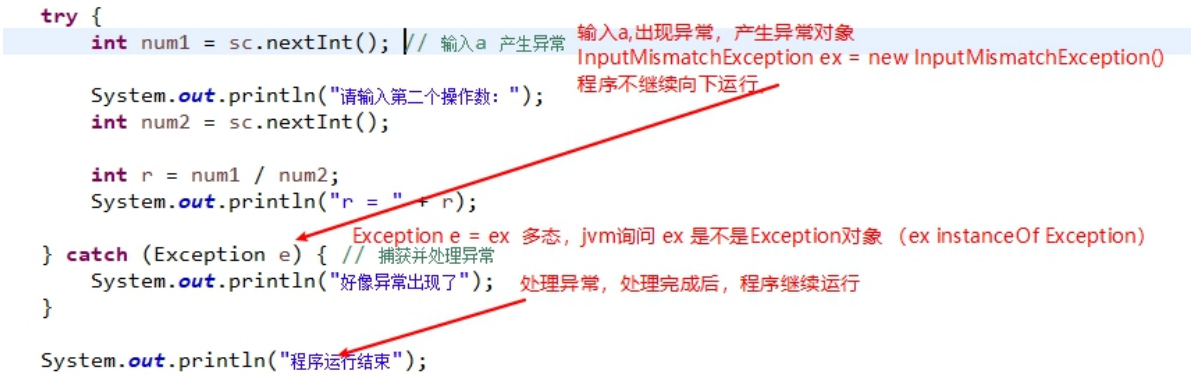
情况1：没有发生异常，正常执行

情况2：出现异常，catch代码块捕获异常，并进行异常处理。处理完成后程序继续执行。



需求1：从控制台输入两个数并做除法

```java
import java.util.Scanner;

public class TryCatchDemo {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("请输入第一个操作数:");

        try{
            int num1 = sc.nextInt();

            System.out.println("请输入第二个操作数:");
            int num2 = sc.nextInt();

            int r = num1 / num2;
            System.out.println("r = " + r);
        }catch (Exception e){
            // 负责处理异常
            System.out.println("出现输入不匹配异常，然后处理");
        }
        System.out.println("程序结束");
    }
}
```



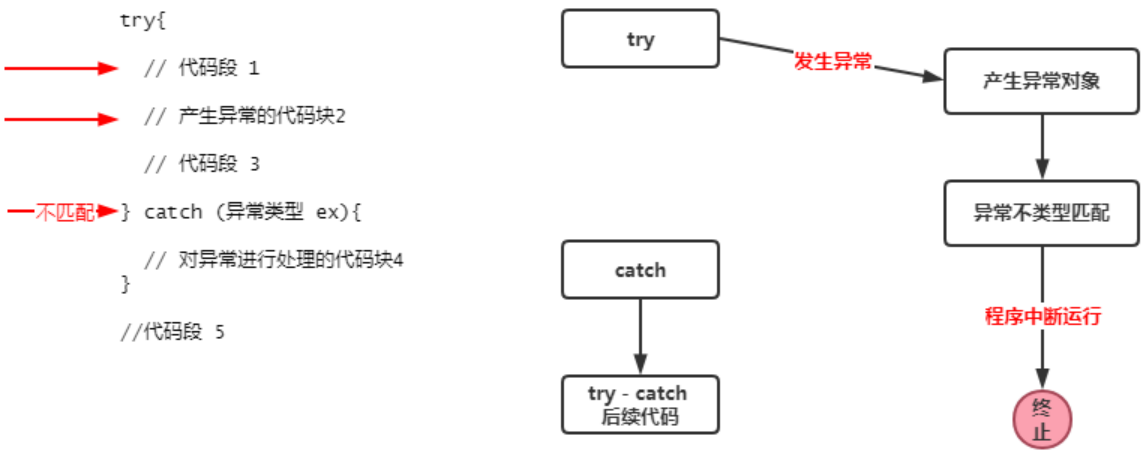
异常发生后，从异常发生的那句代码开始，程序不继续向下运行，立即转入异常处理。

需求2：定义一个两个数的除法方法，并在方法中进行异常处理

```java
public class CatchDemo {
    public static void main(String[] args) {
        System.out.println("begin");
        //divide方法会出现错误，但是因为divide方法已经处理，不会影响到调用方法的继续向下执行。
        divide(17, 0);
        System.out.println("ending");//能够执行到ending
    }

    public static void divide(int a, int b) {
        try {
            System.out.println(a / b);
        } catch (ArithmeticException e) {  //捕获ArithmeticException类型的异常
            System.out.println("除法运算数有错误");
        }
    }
```

```
}
```

情况3：异常不匹配




```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class Test01 {
    public static void main(String[] args) {

        // 需求:从控制台输入两个数并做除法
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入第一个操作数:");

        try{
            int num1 = sc.nextInt();

            System.out.println("请输入第二个操作数:");
            int num2 = sc.nextInt();

            int r = num1 / num2;
            System.out.println("r = " + r);
        }catch (InputMismatchException e){
            // 负责处理异常
            System.out.println("出现输入不匹配异常，然后处理");
        }

        System.out.println("程序结束");
    }
}
```

异常不匹配，程序中断，此时我们就需要多重catch了。

3.2 多重catch(掌握)

可以为try代码书写多个catch用于捕获多个具体的异常。

```
try {
    //可能出现异常的代码
} catch(异常类型A 变量){
    //处理A类型异常的代码
} catch(异常类型B 变量){
    //处理B类型异常的代码
}
...
```

书写时：子类在上，父类在下。

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class MutiCatchDemo {
  public static void main(String[] args) {
         System.out.println("begin");
         divide("17", "0");
         System.out.println("ending");
    }

    public static void divide(String a, String b) {
      try {
        int x = Integer.parseInt(a);
        int y = Integer.parseInt(b);
        System.out.println(x / y);
      } catch (NumberFormatException e) {
        //处理数字格式化异常的代码
        System.out.println("字符串不包含可解析的整数");
      } catch (ArithmeticException e) {
        //处理算术异常的代码
        System.out.println("除数不能为0");
      } catch (Exception e) {
        //处理其他未知的异常
        System.out.println("未知异常");
      }
  }
}
```

3.3 异常对象(掌握)

异常对象是出现异常时的那条语句产生的(jvm 自动创建)。

异常在java类中通过Exception或其具体子类创建，大多数情况下都是由其子类（命名方式:异常类型+Exception）创建，**Exception是所有异常类的父类。**

常用方法	方法介绍
toString	返回异常类型和异常信息
getMessage	返回异常信息
printStackTrace	打印堆栈信息(红色)。包含了异常信息，异常类型，异常位置，方便程序开发阶段的调试（一般要打开），也是JVM默认的异常处理机制

```
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:864)
    at java.util.Scanner.next(Scanner.java:1485)
    at java.util.Scanner.nextInt(Scanner.java:2117)
    at java.util.Scanner.nextInt(Scanner.java:2076)
    at cn.wolfcode._04_exceptionobj.ExObjectDemo.main(ExObjectDemo.java:13)
```

异常堆栈信息：
第一句:表示异常类型和异常的Message构成
最后一句:包含具体发生异常的全路径类,方法,产生异常语句的行数。

```
at cn.wolfcode._04_exceptionobj.ExObjectDemo.main(ExObjectDemo.java:13)
```

3.4 try...catch...finally(掌握)

try...catch 和之前一样用于捕获并处理异常，finally代码块用于处理异常后的收尾工作。

不管是否发生异常，finally总执行。

finally的收尾工作包含释放内存、**关闭文件、关闭网络连接、关闭数据库**、关闭...

```
step 1: 打开文件
step 2: 操作文件
step 3: 关闭文件

try {
    step 1: 打开文件
    step 2: 操作文件
} catch ( Exception e ) {
    // 处理异常
} finally {
    step 3: 关闭文件
}
```

```
 public class TryCatchFinallyDemo {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("请输入被除数:");
        try {
            int num1 = sc.nextInt();

            System.out.println("请输入除数:");
            int num2 = sc.nextInt();

            int r = num1 / num2;
            System.out.println(r);

        } catch (Exception e) {
            // e.printStackTrace();
            System.out.println(e.toString());
        } finally {
            System.out.println("我是finally");
        }

        System.out.println("程序正常结束! ");
    }
```

```
}
```

存在return的try...catch...finally

```
public class TryCatchFinallyDemo {

    public static int div(int a,int b) {
        try {
            int result = a / b;
            return result;
        } catch (Exception e) {
            System.out.println(e.toString());
            return result;
        } finally {
            System.out.println("我是finally");
        }
    }

    public static void main(String[] args) {
        System.out.println(div(10,0));;
        System.out.println("程序正常结束！");
    }
}
```

总结：

- 存在return的try...catch...finally块，finally先执行，然后执行return，**return 总是最后执行的。**

> try-catch / try-catch-finally 快捷键
>
> IDEA : Ctrl + Alt + T

4 抛出异常(掌握)

一旦出现异常，程序会立即终止，所以在开发中我们一定要处理异常，处理异常有两种方式：

一种是直接使用try-catch处理异常（已讲），一种是自身抛出异常，不处理，而抛出异常，有两种：

- 方法里面可能会产生异常，自身不想处理，**提醒**调用该方法的方法做需要处理，使用throws关键字。
- 方法里面会出现异常，但方法不想处理这个异常，使用throw抛出异常对象。

4.1 throws(掌握)

在Java语言中通过throws声明某个方法可能抛出的各种异常。

当方法的定义者在**定义方法时知道调用该方法时可能出现异常，定义者又不知道如何处理时**，此时方法定义者可以选择声明异常，使用throws关键字，**提醒调用该方法的方法做需要处理**，可以声明多个异常，用(,)号分隔。形式：

```
[修饰符] 返回值类型 方法名(形参列表) throws 异常类型1，异常类型2,,, {

}
```

需求1：定义一个两个整数的除法方法，方法声明抛出异常

```
 public class ThrowsDemo {

    // divide方法可能有异常，但divide处理不了该异常，就通过throws声明异常，让divide方法的调用者来处理
    public static void divide(int a, int b) throws Exception {
        System.out.println(a / b);
    }

    public static void main(String[] args) {
        try {
            ThrowsDemo.divide(3, 1);
            ThrowsDemo.divide(1, 0);// 调用divide方法，调用者必须处理或再次抛出
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

声明抛出异常的原因：divide方法自身处理不了该异常，只能使用throws提醒该方法的调用者(main方法)进行处理异常。当然调用者也有两种处理方式： 自己捕获处理或再次声明（要么try...catch ,要么也throws）。

需求2：异常继续上抛

```
public class ThrowsDemo {
    public static void main(String[] args) throws Exception{
        // 调用divide方法，调用者必须处理或再次抛出
        divide(1, 0);
    }

    //divide方法可能有异常，但divide处理不了该异常，就抛出，让divide方法的调用者来处理
    public static void divide(int a, int b) throws Exception {
        System.out.println(a / b);
    }
}
```

在异常声明抛出或者上抛出的过程中，应该遵循以下原则：能在调用处明确处理优先处理，否则继续上抛。

4.2 throw (掌握)

当方法内，需要抛出一个异常对象给调用者时，一般使用throw关键字在方法内手动抛出一个具体的异常对象。

在实际开发过程中，开发者也可以根据程序的需要，手动抛出异常，通过throw关键字。语法

```
XxException ex = new XxException();
throw ex;
或者
throw new XxException();
```

需求：模拟提示用户名xx已经注册

```java
public class ThrowDemo {
    public static void main(String[] args) {
        try {
            ThrowDemo.isExist("will");
        } catch (Exception e) {
            System.out.println(e.getMessage());//对不起，用户名will已经存在
        }
    }

    public static boolean isExist(String userName) throws Exception {
        // 模拟已经注册的用户名
        String[] data = { "will", "lucy", "lily" };

        if (userName != null && userName.length() > 0) {
            for (String name : data) {
                // 用户名相同，证明该用户已经存在
                if (name.equals(userName)) {
                    // 手动抛出一个异常表示存在的不正常情况
                    throw new Exception("对不起，用户名" + userName + "已经存在");
                }
            }
        }
        return false;
    }
}
```

上述代码中throws和throw并不冲突，他们各自的作用是不一样的：

throw：当传入的用户名已经存在，就返回一个结果给isExist方法的调用者。

throws：用于提醒isExist方法的调用者，需要处理异常。

实际开发中，当程序出现某种逻辑问题时由程序员主动抛出某种具有特定详细消息的异常对象。

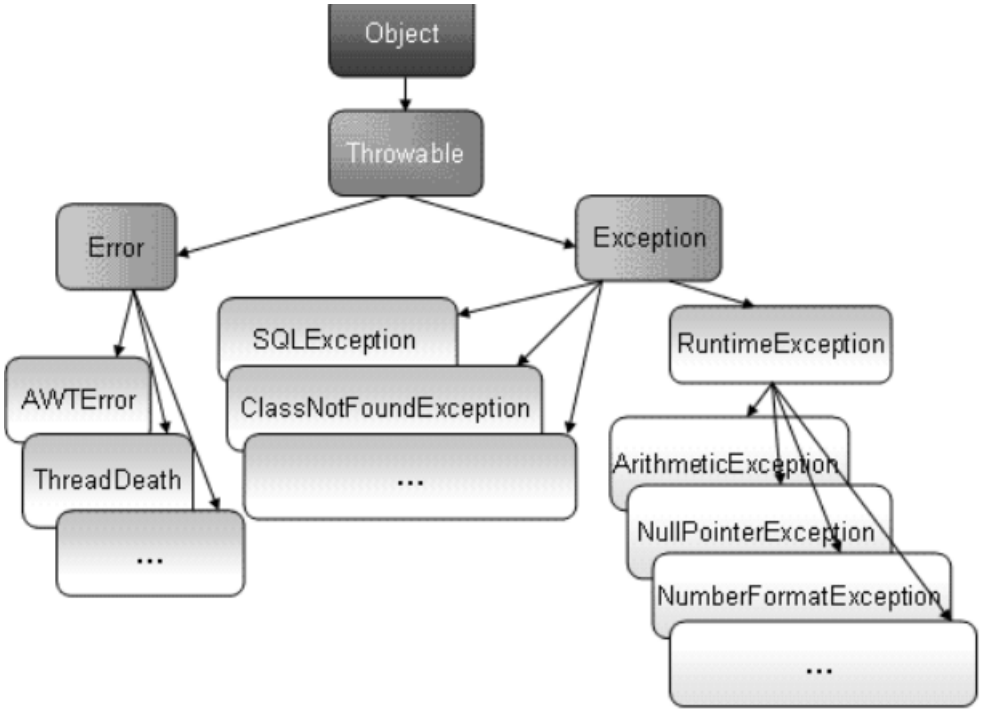
throws和throw的区别

throws 用于方法声明上，表示该方法不需要处理某种类型异常，也在提醒该方法调用者需要处理异常。

throw 用在方法体内，用于抛出具体异常类的对象给调用者使用。

5 异常分类（掌握）

5.1 异常的继承体系（掌握）



Throwable类有两个子类Error和Exception，分别表示错误和异常。

Exception 和Error的子类大都是以Error或Exception作为类名后缀。

5.2 Error（了解）

Error，表示代码运行时 JVM（Java 虚拟机）出现的问题。如系统崩溃或内存溢出等，不需要处理Error，

常见的Error：

- StackOverflowError：当应用程序递归太深而发生堆栈溢出时，抛出该错误。比如死循环或者没有出口的递归调用。
- OutOfMemoryError：因为内存溢出或没有可用的内存提供给垃圾回收器时，Java 虚拟机无法分配一个对象，这时抛出该错误。比如new了非常庞大数量的对象而没释放。

5.3 Exception（掌握）

Exception，表示程序在运行时出现的一些不正常情况，一般大多数表示轻度到中度的问题，属于可预测、可恢复问题。如除数为0，数组索引越界等，这种情况下，程序员通过合理的异常处理，确保程序的正常运行直到结束，常见的Exception

- ArrayIndexOutOfBoundsException：用非法索引访问数组时抛出的异常。如果索引为负或大于等于数组大小，则该索引为非法索引。
- NullPointerException：当应用程序试图在需要对象的地方使用 null 时，抛出该异常。这种情况包括：

异常体系分成：checked（编译时）异常和runtime（运行时）异常。

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
```

划分规则是，**RuntimeException和其子类属于运行时异常**，异常除了运行时异常，其他都是编译时异常。

5.4 运行时异常（了解）

Runtime 异常，顾名思义在编译时期不被检测，只有在运行时期才会被检查出来。

运行异常可以不使用try...catch处理，但一旦出现异常就将由JVM处理（打印堆栈信息）。RuntimeException（运行时异常）通常是指因设计或实现方式不当而导致的问题。程序员小心谨慎是可以避免的异常。如：事先判断对象是否为null就可以避免NullPointerException异常，事先检查除数不为0就可以避免ArithmeticException异常。

运行时异常特点：

在编译阶段，Java编译器检查不出来。一般的，程序可以不用使用try-catch和throws处理运行异常。

简而言之：**对于运行时异常程序可处理(使用try-catch或throws处理)，可不处理。**

5.5 编译时异常（了解）

编译时异常，顾名思义就是在编译时期就会被检测到的异常。除了RuntimeException以及子类以外，其他的Exception及其子类都是编译异常，有时候也称之为**检查时异常。**

编译时异常特点：

在编译阶段，Java编译器会检查出异常，也就说程序中一旦出现这类异常，要么使用try-catch语句捕获，要么使用throws语句声明抛出它，否则编译就不会通过。

简而言之：**程序要求必须处理编译异常，使用try-catch或throws处理。**

> 意识
>
> 遇到不懂的异常类时，首先要分辨它属于编译时异常还是运行时异常。通过快捷键（ctrl+h）查看

6 自定义异常(掌握)

一个异常类只表示某一种特定的异常类型，在项目开发中，可能会出现特定的逻辑问题，此时开发者可以对这些问题进行封装成异常。比如说未来我们会定义一个LogicException用于表示业务逻辑异常。

自定义异常的两种方式，**可以继承Exception类或RuntimeException类。一般推荐继承RuntimeException类。**

自定义异常的步骤：

[1] 确定异常类型（编译时异常、**运行时异常**）。

[2] 继承异常的父类(编译时异常Exception、**运行时异常RuntimeException**)

[3] 定义构造方法。继承异常类之后，一般的，需要提供无参构造方法和带一个String类型参数的构造器。

需求：定义一个年龄的异常，当年龄值不在[1,100]这个范围，程序抛出年龄异常。

自定义异常，继承RuntimeException

```
public class AgeException extends RuntimeException {
    public AgeException() { }

    public AgeException(String message) {
        super(message); // 通过父类的构造器把形参detailMessage赋值给父类的
detailMessage 字段
    }
}
```

定义一个学生类

```
 public class Student {
     private String name;
     private int age;

     // 省略构造器
     // 省略name字段的设置器和访问器

     public int getAge() {
         return age;
     }

     public void setAge(int age) throws AgeException{
         if( age < 0 || age > 120 ) {
             AgeException ex = new AgeException("年龄值不能小于0或者不能大于120");
             throw ex;
         } else {
             this.age = age;
         }
     }
 }
```

测试

```
public class CustomExceptionDemo {
    public static void main(String[] args) {
        Student s1 = new Student("will",17);
        try {
            s1.setAge(130);
        } catch (AgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```