1、课程名称：异常处理

2、知识点

2.1、本节预计讲解的知识点

1、 掌握一下异常的产生及问题
2、 异常的基本处理格式
3、 throw、throws 关键字的使用
4、 通过本章学习可以完整的建立起异常处理操作的标准结构
5、 可以建自定义的异常类
6、 assert 关键字的使用

3、具体内容

3.1、认识异常（重点）

异常是在程序中导致程序中断运行的一种指令流，例如，现在有如下的操作代码：

```
public class ExceptionDemo01{
    public static void main(String argsp[]){
        int i = 10 ;
        int j = 0 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = i / j ;  // 进行除法运算
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

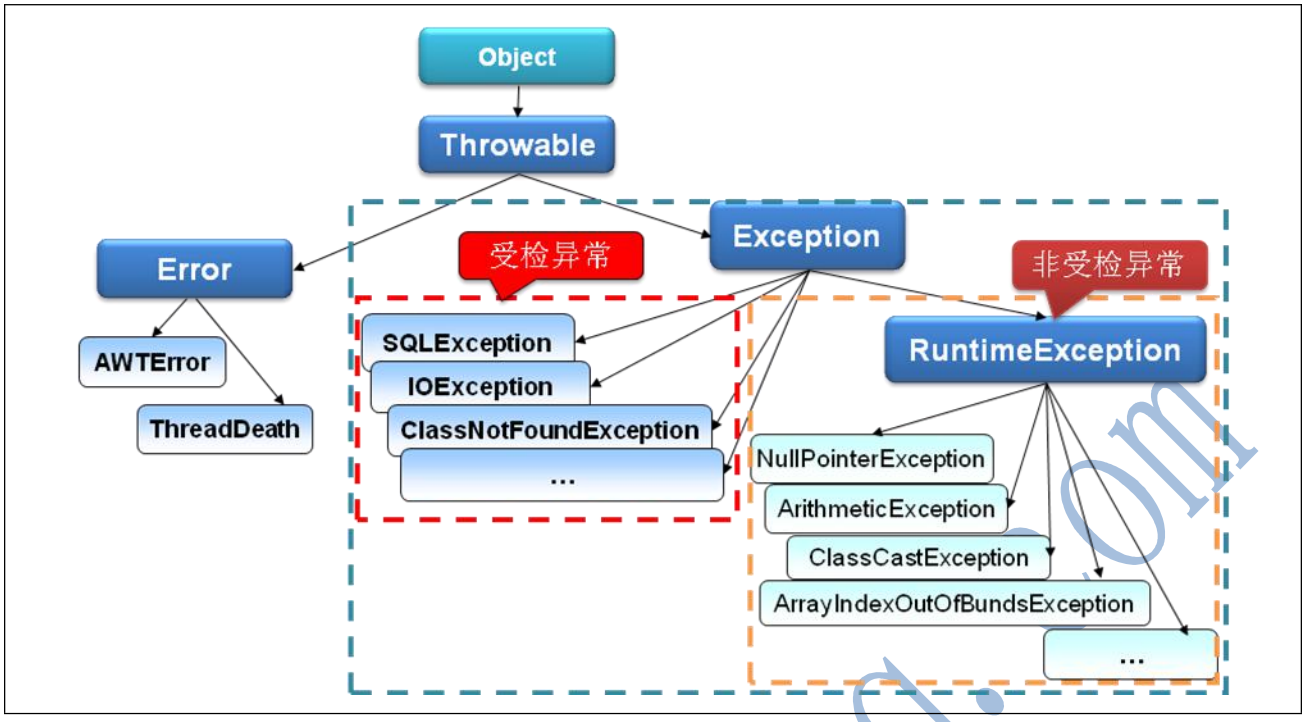
程序运行的结果：

```
============== 计算开始 ==============
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo01.main(ExceptionDemo01.java:6)
```

以上的代码在“int temp = i / j ;”位置处产生了异常，一旦产生异常之后，异常之后的语句将不再执行了，所以现在的程序并没有正确的执行完毕之后就退出了。

那么，为了保证程序出现异常之后仍然可以正确的执行完毕，所以要采用异常的处理机制。

Java 异常结构图：



3.2、处理异常（重点）

如果要想对异常进行处理，则必须采用标准的处理格式，处理格式语法如下：

```
try{
    // 有可能发生异常的代码段
}catch(异常类型 对象){
    // 异常的处理操作
}catch(异常类型 对象){
    // 异常的处理操作
} ...
finally{
    // 异常的统一出口
}
```

但是，以上的格式编写的时候可以直接使用 try…catch 或者 try…catch…finally

范例：对程序进行异常处理

```
public class ExceptionDemo02{
    public static void main(String argsp[]){
        int i = 10 ;
        int j = 0 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            temp = i / j ;      // 进行除法运算
        }catch(ArithmeticException e){
            System.out.println("发生了异常：" + e) ;     // 打印异常对象，调用 toString()方法
        }
        System.out.println("temp = " + temp) ;
```

```
            System.out.println("============= 计算结束 =============") ;
        }
};
```

编写的时候容易不小心犯的错误：

```
public class ExceptionDemo02{
    public static void main(String argsp[]){
        int i = 10 ;
        int j = 0 ;
        System.out.println("============= 计算开始 =============") ;
        // int temp = 0 ;  // 要在外部定义
        try{
            int temp = i / j ;  // 进行除法运算
        }catch(ArithmeticException e){
            System.out.println("发生了异常：" + e) ;     // 打印异常对象，调用 toString()方法
        }
        System.out.println("temp = " + temp) ;        //在此处会报空指向异常，因为现在的 temp 是 try 子句的局部变量
        System.out.println("============= 计算结束 =============") ;
    }
};
```

程序执行结果：

```
============= 计算开始 =============
发生了异常：java.lang.ArithmeticException: / by zero
temp = 0
============= 计算结束 =============
```

以上的操作中，所有的异常进行了正确的处理，那么保证程序的正常执行完毕，并正常退出。

也就是说，一旦出现了异常之后，try 语句中捕获到的异常交给 catch 语句进行处理。

一旦有异常产生，在 try 语句之中的异常产生之后的代码也将不再执行。

```
public class ExceptionDemo03{
    public static void main(String argsp[]){
        int i =1 0 ;
        int j = 0 ;
        System.out.println("============= 计算开始 =============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            System.out.println("********************************") ;
            temp = i / j ;      // 进行除法运算
            System.out.println("--------------------------------") ;       //当上句发生异常时此语句不被执行
        }catch(ArithmeticException e){
            System.out.println("发生了异常：" + e) ;     // 打印异常对象，调用 toString()方法
        }
        System.out.println("temp = " + temp) ;
```

```
            System.out.println("============= 计算结束 =============") ;
        }
};
```

但是，现在要求，对以上的程序进行进一步的扩充，要求使用初始化参数的方式输入两个运算的数字。

```
public class ExceptionDemo03{
    public static void main(String argsp[]){
        int i = 0 ;
        int j = 0 ;
        System.out.println("============= 计算开始 =============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            i = Integer.parseInt(argsp[0]) ;    // 将第一个输入参数变为 int 型
            j = Integer.parseInt(argsp[1]) ;    // 将第二个输入参数变为 int 型
            System.out.println("********************************") ;
            temp = i / j ;    // 进行除法运算
            System.out.println("--------------------------------") ;
        }catch(ArithmeticException e){
            System.out.println("发生了异常：" + e) ;    // 打印异常对象，调用 toString()方法
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============= 计算结束 =============") ;
    }
};
```

但是，以上的程序一旦交给了普通用户使用，则就不那么简单了。

如果不输入数据，则出现以下的问题：

```
============= 计算开始 =============
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at ExceptionDemo03.main(ExceptionDemo03.java:8)
```

如果输入的数据不是数字，则出现以下的问题：

```
============= 计算开始 =============
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
```

那么，此时的程序只是对一个问题进行了处理，如果现在要对多个问题进行处理的话，则就必须加入多个 catch。

范例：代码修改如下

```
public class ExceptionDemo04{
    public static void main(String argsp[]){
        int i = 0 ;
        int j = 0 ;
        System.out.println("============= 计算开始 =============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            i = Integer.parseInt(argsp[0]) ;    // 将第一个输入参数变为 int 型
            j = Integer.parseInt(argsp[1]) ;    // 将第二个输入参数变为 int 型
            System.out.println("********************************") ;
            temp = i / j ;    // 进行除法运算
```

```
            System.out.println("--------------------------------") ;
        }catch(ArithmeticException e){
            System.out.println("发生了算术异常：" + e) ;      // 打印异常对象，调用 toString()方法
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("输入错误参数异常：" + e) ;
        }catch(NumberFormatException e){
            System.out.println("输入的参数不是数字异常：" + e) ;
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

那么，证明，以上的程序中同时处理了多个异常。可以想象一下，现在的程序只处理了三个异常，会不会有更多的异常呢？

3.3、异常的处理流程（重点）

那么一个程序中产生异常之后到底是如何处理的呢？

1、 一旦产生异常，则系统会自动产生一个异常类的实例化对象。

2、 那么，此时如果存在了 try 语句，则会自动找到匹配的 catch 语句执行，如果没有异常处理，则程序将退出，并由系统报告错误。

3、 所有的 catch 根据方法的参数匹配异常类的实例化对象，如果匹配成功，则表示由此 catch 进行处理。

明白以上的操作之后，就可以得出这样的一个假设，如果现在在异常中直接将父类进行处理，则肯定会非常的方便，因为符合于对象向上转型操作。

从之前的三个异常（ArithmeticException、ArrayIndexOutOfBoundsException、NumberFormatException）可以发现：所有类的命名格式都是 XxxException。证明 Exception 表示的是所有异常的父类。

3.4、异常类的继承关系（重点）

从 Java DOC 文档中可以发现，Exception 本身还是存在父类的。父类是 Throwable（可能的抛出），此类下存在两个子类：

- Error：表示的是错误，是 JVM 发出的错误操作。
- Exception：一般表示所有程序中的错误，所以一般在程序中将进行 try…catch 的处理。

范例：将以上的程序进行修改

```
public class ExceptionDemo05{
    public static void main(String argsp[]){
        int i = 0 ;
        int j = 0 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            i = Integer.parseInt(argsp[0]) ;      // 将第一个输入参数变为 int 型
            j = Integer.parseInt(argsp[1]) ;      // 将第二个输入参数变为 int 型
            System.out.println("********************************") ;
```

```
            temp = i / j ;       // 进行除法运算
            System.out.println("--------------------------------") ;
        }catch(ArithmeticException e){
            System.out.println("发生了算术异常：" + e) ;       // 打印异常对象，调用 toString()方法
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("输入错误参数异常：" + e) ;
        }catch(NumberFormatException e){
            System.out.println("输入的参数不是数字异常：" + e) ;
        }catch(Exception e){
            System.out.println("其他异常：" + e) ;
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

以上的程序中如果之前的三个 catch 都无法直接匹配的话，则最后将由 Exception 完成。但是，在进行异常捕获的时候还有以下的几个注意点：

1、 捕获更粗的异常不能放在捕获更细的异常之前。

2、 在进行异常捕获及处理的时候，不要使用 Throwable 作为捕获的类型，因为范围太大了。

3、 如果为了方便，则可以将所有的异常都使用 Exception 进行捕获。

```
public class ExceptionDemo06{
    public static void main(String argsp[]){
        int i = 0 ;
        int j = 0 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = 0 ;     // 要在外部定义
        try{
            i = Integer.parseInt(argsp[0]) ;     // 将第一个输入参数变为 int 型
            j = Integer.parseInt(argsp[1]) ;     // 将第二个输入参数变为 int 型
            System.out.println("********************************") ;
            temp = i / j ;       // 进行除法运算
            System.out.println("--------------------------------") ;
        }catch(Exception e){
            System.out.println("程序产生了异常：" + e) ;
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

4、 但是为了操作的方便起见，所以的异常最好进行分别的处理，即：以上程序中会产生的三个异常最好分别使用三个 try…catch 块进行处理。因为这样做的话会比较容易确认异常的位置。

3.5、异常的统一出口

在进行异常的处理之后，在异常的处理格式中还有一个 finally 语句，那么此语句将作为异常的统一出口，不管是否

产生了异常，最终都要执行此段代码。

```
public class ExceptionDemo07{
    public static void main(String argsp[]){
        int i = 0 ;
        int j = 0 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            i = Integer.parseInt(argsp[0]) ;    // 将第一个输入参数变为 int 型
            j = Integer.parseInt(argsp[1]) ;    // 将第二个输入参数变为 int 型
            System.out.println("********************************") ;
            temp = i / j ;    // 进行除法运算
            System.out.println("--------------------------------") ;
        }catch(Exception e){
            System.out.println("程序产生了异常：" + e) ;
        }finally{
            System.out.println("** 不管是否出现异常，此代码都将执行！") ;
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

如果要想进一步深入的研究以上代码的用处，则必须结合后面的异常标准格式，此处先了解其基本语法即可。

随着 JDK 的发展，实际上对于异常的处理格式，也出现了许多的变态做法。从最早的 Java 开始对于异常的处理结构就分为两种：

- try…catch
- try…catch…finaly

但是最新的 JDK 出现了一种根本就没有任何意义的操作：try…finally

```
public class ExceptionDemo08{
    public static void main(String argsp[]){
        int i = 1 ;
        int j = 2 ;
        System.out.println("============== 计算开始 ==============") ;
        int temp = 0 ;    // 要在外部定义
        try{
            temp = i / j ;    // 进行除法运算
        }finally{
            System.out.println("** 不管是否出现异常，此代码都将执行！") ;
        }
        System.out.println("temp = " + temp) ;
        System.out.println("============== 计算结束 ==============") ;
    }
};
```

以上的代码根本就没有任何的意义，因为一旦发生异常之后，程序仍然会从 finally 块退出。

3.6、throws 关键字（重点）

在程序中异常的基本处理已经掌握了，但是随异常一起的还有一个称为 throws 关键字，此关键字主要在方法的声明上使用，表示方法中不处理异常，而交给调用处处理。

```
class Math{
    public int div(int i,int j) throws Exception{    // 此方法有可能出现异常
        int temp = 0 ;
        temp = i / j ;
        return temp ;
    }
};
public class ExceptionDemo09{
    public static void main(String argsp[]){
        try{
            System.out.println(new Math().div(10,2)) ;    // 此方法不管是否有异常，都需要进行处理
        }catch(Exception e){
            e.printStackTrace() ;  // 打印异常信息，这种方式打印出的信息是最全的
        }
    }
};
```

因为程序中的 div()方法本身使用了 throws 关键字声明，所以在调用此方法的位置处就必须明确的使用 try..catch 进行异常的捕获处理，而不管是否真的会发生异常。

既然 throws 关键字可以在普通方法上使用，那么能不能在主方法上使用呢？

如果在主方法上使用了 thorws 关键字声明的话，表示所有的异常将由 JVM 进行处理。

```
class Math{
    public int div(int i,int j) throws Exception{    // 此方法有可能出现异常
        int temp = 0 ;
        temp = i / j ;
        return temp ;
    }
};
public class ExceptionDemo10{
    public static void main(String argsp[]) throws Exception{ // 所有的异常交由 JVM 进行处理
        System.out.println(new Math().div(10,0)) ;    // 此方法不管是否有异常，都需要进行处理
    }
};
```

此时一旦发生异常之后，所有的异常将交由 JVM 去处理，实际上对于 Java 程序来讲，如果没有加入任何的异常处理，则默认就将由 JVM 进行异常的处理操作。

3.7、throw 关键字（重点）

throw 关键字表示在程序中人为的抛出一个异常，因为从异常处理机制来看，所有的异常一旦产生之后，实际上抛出的就是一个异常类的实例化对象，那么此对象也可以由 throw 直接抛出。

```
public class ExceptionDemo11{
```

```
    public static void main(String argsp[]){ // 所有的异常交由 JVM 进行处理
        try{
            throw new Exception("抛着玩的。") ;  // 人为的抛出异常
        }catch(Exception e){
            e.printStackTrace() ;
        }
    }
};
```

3.8、异常的标准处理结构（绝对重点）

学习完了 try…catch…finally、throw、throws 关键字，那么这些格式和关键字到底该如何应用呢？

- 在异常处理中，不管是否加上 finally 最终结果都会执行异常之后的操作？
- 在程序中都会尽量的避免异常的产生，为什么还要存在 throw 抛异常呢？

所以，在实际的开发中，以上的五个关键字是要一起联合使用的。

现在要求完成以下的一个操作方法：

- 定义一个除法操作，但是在进行除法操作之前，必须打印“计算开始”，在整个操作的最后必须打印“计算结束”
- 如果中间产生了异常，则必须返回到调用处进行处理

```
class Math{
    public int div(int i,int j) throws Exception{    // 此方法有可能出现异常
        System.out.println("====== 计算开始 ======") ;
        int temp = 0 ;
        try{
            temp = i / j ;
        }catch(Exception e){
            throw e ;  // 所有异常抛出
        }
        finally{
            System.out.println("====== 计算结束 ======") ;
        }
        return temp ;
    }
};
public class ExceptionDemo12{
    public static void main(String argsp[]){
        try{
            System.out.println(new Math().div(10,0)) ;    // 此方法不管是否有异常，都需要进行处理
        }catch(Exception e){
            e.printStackTrace() ;  // 打印异常信息，这种方式打印出的信息是最全的
        }
    }
};
```

以上程序实际上就体现出了异常处理的标准结构。

3.9、RuntimeExcepion 与 Exception 的区别（重点）

只要程序的方法中使用了 throws 关键字，是不是就意味着，程序一定要使用 try…catch 进行处理。但是观察一下 Integer 类中的以下方法：

- public static int parseInt(String s) **throws NumberFormatException**

此方法的功能是用于将字符串变为基本的 int 型数据，但是此方法抛出了一个异常。

```
public class ExceptionDemo13{
    public static void main(String argsp[]){
        int i = Integer.parseInt("123") ;
    }
};
```

以上的操作，按照正常的思路来讲，parseInt()方法必须使用 try…catch 进行捕获，如果不捕获那么肯定连编译都无法通过。

因为 NumberFormatException 并不是 Exception 的直接子类，而是 RuntimeException 的子类，只要是 RuntimeException 的子类，则表示程序在操作的时候可以不必使用 try…catch 进行处理，如果有异常发生，则由 JVM 进行处理。当然，如果加上了异常处理格式，程序也不会有任何的错误发生。

3.10、自定义异常类（理解）

在 Java 中，已经提供了很多的异常类的定义，但是如果在开发一些个人的系统中可能需要使用一些自己的异常类的操作，那么此时就可以通过继承 Exception 类的方式完成一个自定义异常类的操作。

```
class MyException extends Exception{  // 继承 Exception，表示一个自定义异常类
    public MyException(String msg){
        super(msg) ;      // 调用 Exception 中有一个参数的构造
    }
};
public class ExceptionDemo14{
    public static void main(String argsp[]){
        try{
            throw new MyException("自己定义，自己抛着玩的。") ;
        }catch(Exception e){
            e.printStackTrace() ;
        }
    }
};
```

3.11、assert 关键字（了解）

在 JDK 1.4 的时候增加了 assert 关键字，表示断言。

当程序执行到某个固定位置的时候，程序中的某个变量的取值肯定是预期的结果，那么这种操作可以使用断言完成。断言的操作语法：

```
assert 表达式 ;
```

范例：使用断言

```
public class AssertDemo{
    public static void main(String args[]){
        int x = 90 ;
        assert x==10 ;    // 断言 x 的取值肯定是 10
    }
};
```

以上程序中好象并没有任何的断言检查，因为如果要想启动断言，则必须在运行的时候加上参数：java -ea AssertDemo

以上的断言操作，如果出现了错误，将由系统进行提示，当然，用户也可以自己定义自己的提示信息，语法如下：

```
assert 表达式 : 错误信息 ;
```

范例：加入错误信息

```
public class AssertDemo{
    public static void main(String args[]){
        int x = 30 ;
        assert x==10 : "操作的结果不正确" ;  // 断言 x 的取值肯定是 10
    }
};
```

3.12、Eclipse debug

debug：调试是程序员编码过程中找逻辑错误的一个很重要的手段

断点：遇到断点，暂挂，等候命令

debug as —> Java Application

快捷键

F5：单步跳入。进入本行代码中执行

F6：单步跳过。执行本行代码，跳到下一行

F7：单步返回。跳出方法

F8：继续。执行到下一个断点，如果没有断点了，就执行到结束

Ctrl+R：执行到光标所在的这一行

4、总结

1、 异常的出现，如果没有进行合理的处理，则有可能造成程序的非正常结束。

2、 使用 try…catch、try…catch…finally 完成异常的基本处理

- 在 try 中捕获异常
- 交给 catch 进行匹配
- 不管是否发生异常，最终结果都要执行 finally 语句

3、 在程序中可以使用 throws 在方法的声明处声明，表示此方法不处理任何的异常。使用 throw 人为的抛出一个异常。

4、 try、catch、finally、throw、throws 联合使用才能形成一个异常的标准处理格式

5、 RuntimeException 的子类在程序中可以不用进行处理，如果有异常产生则交给 JVM 进行处理，当然，如果有需要也可以加上 try…catch 进行处理。

6、 一个类只要继承了 Exception 类，那么此类就表示异常类。

7、 throwable 分为两个子类

- Error：表示 JVM 出错，一般无法处理
- Exception：表示程序出错，一般由开发人员处理

5、作业

1、 创建类：ExceptionTest.java

arithmeticException()方法 构造一个零除异常并抛出 在主方法中处理

outOfBounds()方法构造一个数组越界异常并用 try 处理

主方法事例如下：

```
ExceptionTest t = new ExceptionTest();
        t.outOfBounds();
        try {
            t.arithmeticException();
        } catch (Exception e) {
            System.out.println("除数不能为零！");
        }
```

控制台信息如下（参考）：

```
数组越界了!
处理完毕!
3/0
除数不能为零!
```

2、 创建类：MyException.java

包括一个属性：name

构造函数事例如下：

```
    public MyException(String message) {
        super(message);
        this.name = "自定义异常一";
    }
```

主函数事例如下：

```
try {
        throw new MyException("抛出自定义异常");
} catch (MyException e) {
        System.out.println(e.getName());
}
```