

# 异常处理

## 目标

1. 明确什么是异常 （重点）
2. 能辨识出常见的异常及其含义。 （熟悉+）
3. 理解异常产生的原理 （了解）
4. 能处理异常 （重点）
5. 能够自定义异常类型 （熟悉）

## 什么是异常？

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 =====") ;
    }
};
```

运行结果：

```
===== 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。

## 处理异常

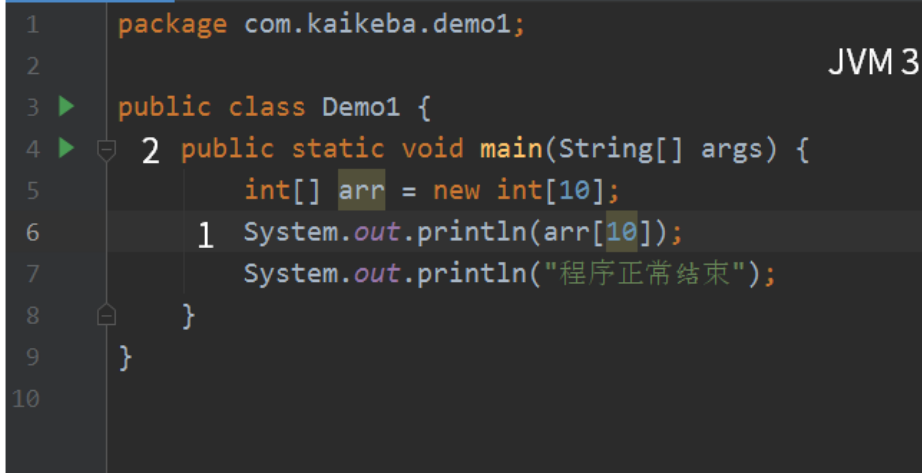
如果要想对异常进行处理，则必须采用标准的处理格式，处理格式语法如下：

```
try{
    // 有可能发生异常的代码段
}catch(异常类型1 对象名1){
    // 异常的处理操作
}catch(异常类型2 对象名2){
```

```
// 异常的处理操作
} ...
finally{
    // 异常的统一出口
}
```

## try+catch的处理流程

- 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，此时如果异常发生在try语句，则会自动找到匹配的catch语句执行，如果没有在try语句中，则会将异常抛出。
- 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。



```
1 package com.kaikeba.demo1;
2
3 public class Demo1 {
4     2 public static void main(String[] args) {
5         int[] arr = new int[10];
6         1 System.out.println(arr[10]);
7         System.out.println("程序正常结束");
8     }
9 }
10
```

1. 发生了异常（JVM根据异常的情况，创建了一个异常对象 - 包含了异常信息）
2. main未处理，自动将异常抛给了main的调用者JVM
3. JVM对异常信息进行了响应（将异常信息显示到控制台，中断处理）

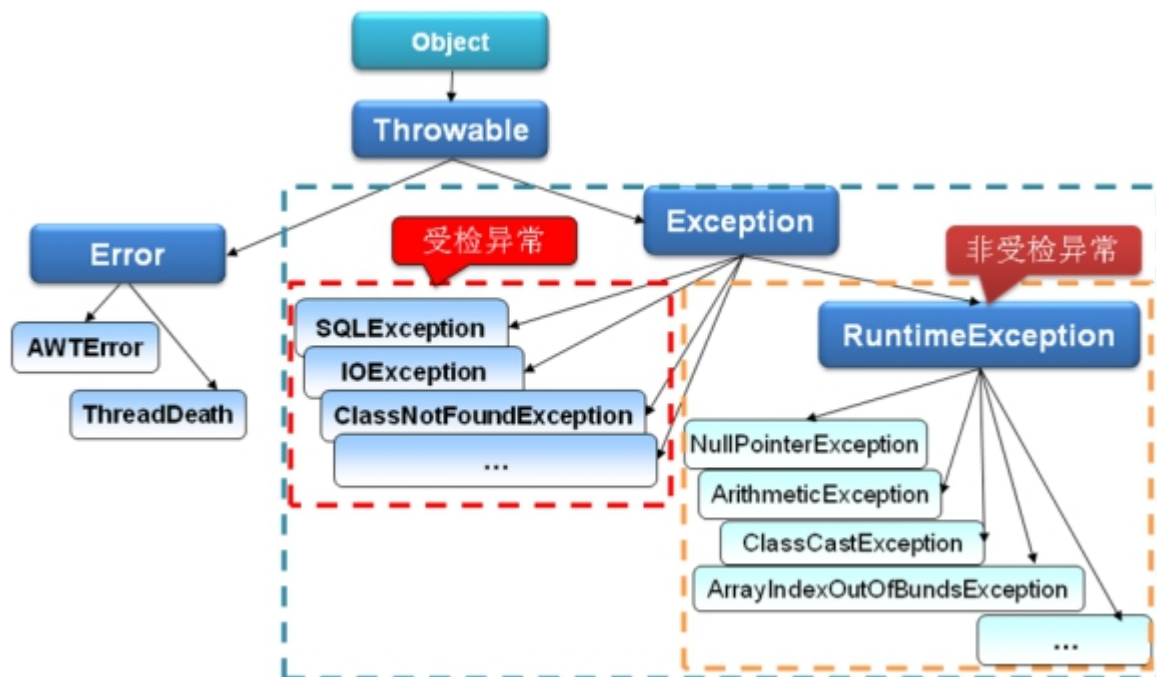
## finally

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。

## 异常体系结构

异常指的是Exception，Exception类，在Java中存在一个父类Throwable（可能的抛出）Throwable存在两个子类：

- 1.Error：表示的是错误，是JVM发出的错误操作，只能尽量避免，无法用代码处理。
- 2.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。



多异常捕获的注意点：

- 1、 捕获更粗的异常不能放在捕获更细的异常之前。
- 2、 如果为了方便，则可以将所有的异常都使用Exception进行捕获。

特殊的多异常捕获写法：

```
catch(异常类型1 | 异常类型2 对象名){  
    //表示此块用于处理异常类型1 和 异常类型2 的异常信息  
}
```

## throws关键字

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。

格式：

```
返回值 方法名称()throws Exception{  
  
}
```

## throw关键字

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。

代码： `throw new Exception("抛着玩的。");`

## RuntimeException与Exception的区别

注意观察如下方法的源码：

Integer类： `public static int parseInt(String text)throws NumberFormatException`

此方法抛出了异常，但是使用时却不需要进行try。。。catch捕获处理，原因：

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，也可以通过try catch处理。

## 自定义异常类 了解

编写一个类，继承Exception，并重写一参构造方法 即可完成自定义受检异常类型。

编写一个类，继承RuntimeException，并重写一参构造方法 即可完成自定义运行时异常类型。

例如：

```
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
    }
};
```

自定义异常可以做很多事情，例如：

```
class MyException extends Exception{
    public MyException(String msg){
        super(msg) ;
        //在这里给维护人员发短信或邮件，告知程序出现了BUG。
    }
};
```

## 异常处理常见面试题

1. try-catch-finally 中哪个部分可以省略？

答： catch和finally可以省略其中一个， catch和finally不能同时省略

注意:格式上允许省略catch块，但是发生异常时就不会捕获异常了,我们在开发中也不会这样去写代码。

2. try-catch-finally 中, 如果 catch 中 return 了, finally 还会执行吗?

答: finally中的代码会执行

详解:

执行流程:

1. 先计算返回值, 并将返回值存储起来, 等待返回
2. 执行finally代码块
3. 将之前存储的返回值, 返回出去;

需注意:

1. 返回值是在finally运算之前就确定了, 并且缓存了, 不管finally对该值做任何的改变, 返回的值都不会改变

2. finally代码中不建议包含return, 因为程序会在上述的流程中提前退出, 也就是说返回的值不是try或catch中的值

3. 如果在try或catch中停止了JVM, 则finally不会执行. 例如停电- -, 或通过如下代码退出  
JVM: System.exit(0);

