03 异常

JAVAEE高级

一：异常

> 1、在Java语言中，将程序执行中发生的不正常情况称为《异常》
> (开发过程中的逻辑错误不是异常)
>
> 2、Throwable类是java语言中的所有错误和异常的基类
>
> 3、Java程序在执行过程中所发生的异常事件可分为两类

1、Error

> 1、Java虚拟机无法解决的严重问题
>
> 2、如：JVM系统内部错误、资源耗尽等严重情况

2、Exception

> 1、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;
>
> 2、如：空指针访问，网络连接中断，下标越界.....

```
    @Test
    public  void getVoid(){
       int [] arr={1,2,3,4,5};
        arr=null;
//报异常:java.lang.NullPointerException
     System.out.println(arr[0]);
   }

```

```
    @Test
    public  void getVoid(){
       int [] arr={1,2,3,4,5};
 //报异常:java.lang.ArrayIndexOutOfBoundsException
     System.out.println(arr[10]);
    }
```

二、异常解决方法

> 1、对于这些异常，一般有两种解决方法：
>
> 2、一是遇到错误就终止程序的运行
>
> 3、另一种方法是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
>
> 4、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
>
> 5、如：除数为0，数组下标越界.....

1、JVM默认的处理异常

> 1、main函数收到这个问题时、有两种处理方式
> 2、一种：自己将该问题处理、然后继续运行
> 3、二种：自己没有针对的处理方式、只有交给调用main的jvm来处理
> 4、jvm有一个默认的异常处理机制、就将该异常进行处理
> 5、并将该异常的名称、异常的信息、异常出现的位置打印在了控制台上、同时将程序停止运行

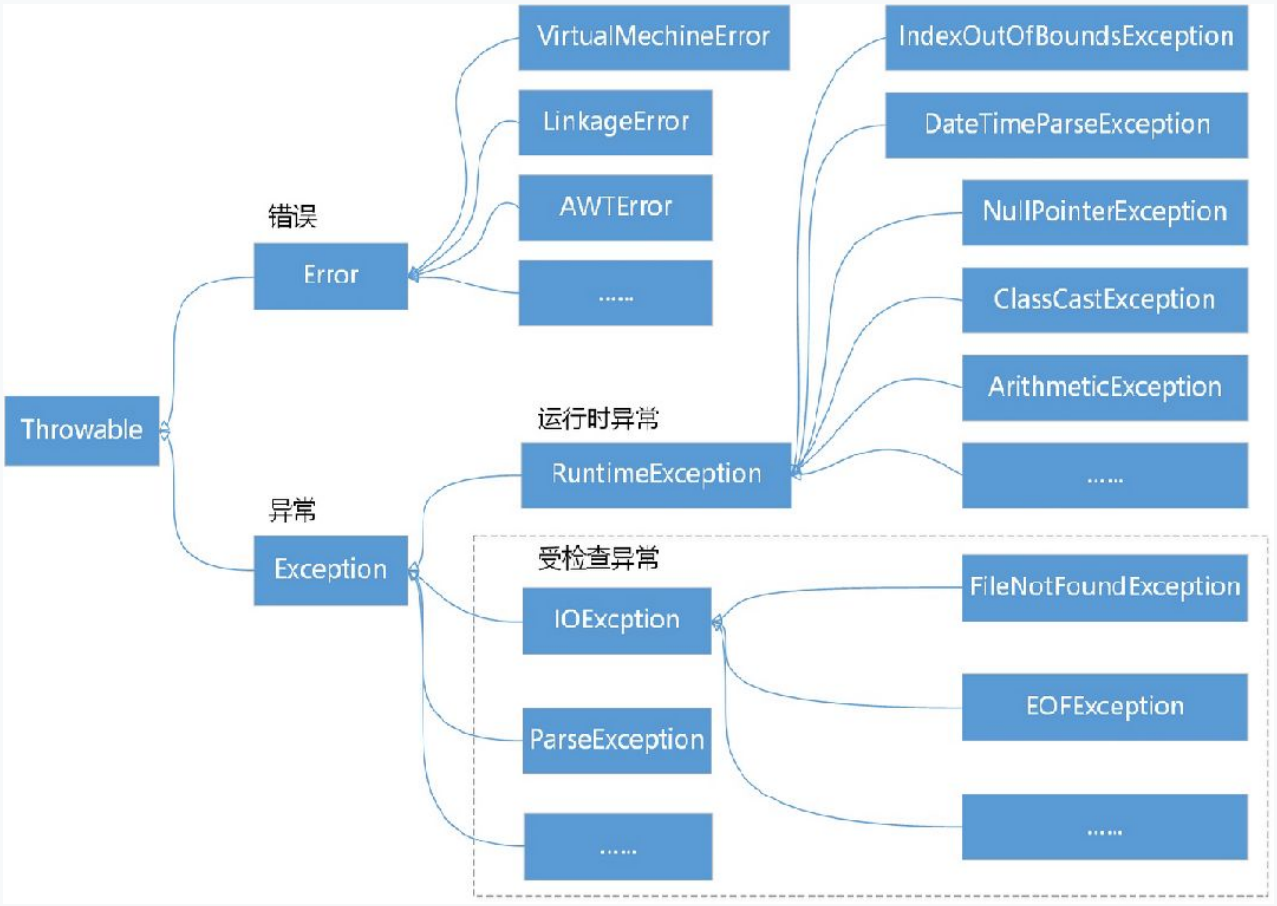
```
    @Test
    public void getVoid1() {
        div(10, 0);
    }

    public int div(int i, int j) {
        return i / j;
```

```
8.          }
```

三、异常分类

1、异常的分类图



1、编译时异常

1、是指编译器要求必须处置的异常

2、即程序在运行时由于外界因素造成的一般性异常

3、编译器要求java程序必须捕获或声明所有编译时异常

如：IOException；SQLException；ClassNotFoundException；.....

异常	描述
ClassNotFoundException	应用程序试图加载类时，找不到相应的类，抛出该异常。
CloneNotSupportedException	当调用 Object 类中的 clone 方法克隆对象，但该对象的类无法实现 Cloneable 接口时，抛出该异常。
IllegalAccessException	拒绝访问一个类的时候，抛出该异常。
InstantiationException	当试图使用 Class 类中的 newInstance 方法创建一个类的实例，而指定的类对象因为是一个接口或是一个抽象类而无法实例化时，抛出该异常。
InterruptedException	一个线程被另一个线程中断，抛出该异常。
NoSuchFieldException	请求的变量不存在
NoSuchMethodException	请求的方法不存在

2、运行时异常

1、是指编译器不要求强制处置的异常

2、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

3、java.lang.RuntimeException类及它的子类都是运行时异常

如：java.lang.ArrayIndexOutOfBoundsException；
java.lang.NullPointerException；.....

异常	描述
ArithmeticException	当出现异常的运算条件时，抛出此异常。例如，一个整数"除以零"时，抛出此类的一个实例。
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。
ArrayStoreException	试图将错误类型的对象存储到一个对象数组时抛出的异常。
ClassCastException	当试图将对象强制转换为不是实例的子类时，抛出该异常。
IllegalArgumentException	抛出的异常表明向方法传递了一个不合法或不正确的参数。
IllegalMonitorStateException	抛出的异常表明某一线程已经试图等待对象的监视器，或者试图通知其他正在等待对象的监视器而本身没有指定监视器的线程。
IllegalStateException	在非法或不适当的时间调用方法时产生的信号。换句话说，即 Java 环境或 Java 应用程序没有处于请求操作所要求的适当状态下。
IllegalThreadStateException	线程没有处于请求操作所要求的适当状态时抛出的异常。
IndexOutOfBoundsException	指示某排序索引（例如对数组、字符串或向量的排序）超出范围时抛出。
NegativeArraySizeException	如果应用程序试图创建大小为负的数组，则抛出该异常。
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
NumberFormatException	当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常。
SecurityException	由安全管理器抛出的异常，指示存在安全侵犯。
StringIndexOutOfBoundsException	此异常由 String 方法抛出，指示索引或者为负，或者超出字符串的大小。
UnsupportedOperationException	当不支持请求的操作时，抛出该异常。

四、异常处理机制

> 1、Java提供的是异常处理的抓与抛模型

1、抛出(throw)异常

> 1、Java程序的执行过程中如出现异常，会生成一个异常类对象
>
> 2、该异常对象将被提交给Java运行时系统，这个过程称为抛出(throw)异常

```
public void getDays(String date1, String date2) throws ParseException {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
        Date parse1 = simpleDateFormat.parse(date1);
    }
```

2、异常对象的生成

①、由虚拟机自动生成

> 1、程序运行过程中，虚拟机检测到程序发生了问题
>
> 2、如果在当前代码中没有找到相应的处理程序
>
> 3、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）
>
> 4、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

②、手动创建

> 1、创建好的异常对象不抛出对程序没有任何影响
>
> 2、和创建一个普通对象一样

```
    @Test
    public void getVoid1() {
        Exception exception = new ClassCastException();
    }
```

2、捕获(catch)异常

> 1、直到异常被处理，这个过程称为捕获(catch)异常
>
> 2、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

> 3、程序员通常只能处理Exception，而对Error无能为力

```
public void getDays(String date1, String date2) {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date parse1 = simpleDateFormat.parse(date1);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
```

五、捕获异常

1、语法

> 1、try catch

```
public void getEx() {
        try {

        } catch (RuntimeException e) {

            e.printStackTrace();
        }
    }
```

> 2、try finally

```
    public void getEx() {
        try {

        } finally {

        }
```

```
    }
```

> 3、try catch .... finally

```
    public void getEx() {
        try {

        } catch (RuntimeException e) {

            e.printStackTrace();
        } catch (Exception e) {

            e.printStackTrace();
        } finally {

        }
    }
```

2、解释

①：try

> 1、捕获异常的第一步是用try{...}语句块选定捕获异常的范围
>
> 2、将可能出现异常的代码放在try语句块中

②：catch (Exceptiontype e)

> 1、在catch语句块中是对异常对象进行处理的代码
>
> 2、每个try语句块可以伴随一个或多个catch语句(1.7以后)
>
> 3、用于处理可能产生的不同类型的异常对象

```
    @Test
    public void getDays() {
        try {
            System.out.println(1 / 0);
            int[] arr = { 1, 2, 3, 4, 5 };
            System.out.println(arr[10]);
```

```
            } catch (ArithmeticException e) {
                e.printStackTrace();
            } catch (ArrayIndexOutOfBoundsException e) {
                e.printStackTrace();
            }
        }
```

> 4、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;但是不能使用它的子类异常类来当作catch的参数
>
> 5、捕获异常的对象，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
>
> toString()
>
> 获取异常类名和异常信息，返回字符串

```
public int div(int i, int j) {
        return i / j;
    }

    @Test
    public void getDays() {
        try {
            div(10, 0);
        } catch (RuntimeException e) {
            String message = e.getMessage();
            System.out.println(message);//打印异常信息
            System.out.println(message);//调用toString方法，打印异常信息和异常类名
            e.printStackTrace();//JVM默认的打印的方式
        }
    }
```

③：finally

> 1、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
>
> 2、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

```
@Test
    public void getDays() {
        try {
            div(10, 0);
        } catch (RuntimeException e) {
            String message = e.getMessage();
            System.out.println(message);
            e.printStackTrace();
        }finally{
            System.out.println("我执行了,哈哈哈");
        }
    }
```

④：总结解释：

> 1、茫茫人海之中，与你相遇；你一直在try我总是catch，无论你发什么脾气我都是静静的接受，默默的处理，但是最终你归宿却是finally

六、案列

1、案列一

> 1、需求：在try块中，编写被零除的代码；catch块中捕获被零除所产生的异常，并且打印异常信息；finally块中，打印一条语句。

```
    @Test
    public void getVoid() {
        try {
            System.out.println(1 / 0);
        } catch (ArithmeticException e) {
            String message = e.getMessage();
            System.out.println("message:" + message);
            e.printStackTrace();
        } finally {
            System.out.println(" 当前除数不能为零");
        }
}
```

2、面试扩展

> 1、final
> 可以修饰类,不能被继承
> 修饰方法,不能被重写
> 修饰变量,只能赋值一次
>
> 2、finally
> try语句中的一个语句体,不能单独使用,用来释放资源
>
> 3、finalize
> 是在垃圾收集器删除对象之前对这个对象进行调用
> 从内存中清除出去前，做必要的清理工作

```
 public class TestInt {
    @Override
    protected void finalize() throws Throwable {
        System.out.println("同学没有交班费！");
    }
    @Test
    public void getVoid() {
        new TestInt();
        System.gc();
    }
}
```

3、如果catch里面有return语句，请问finally的代码还会执行吗?如果会，请问是在return前还是return后

```
@Test
    public void getVoid() {
        int void01 = getVoid01();
        System.out.println(void01);
    }

    public int getVoid01() {
        int i = 10;
        try {
```

```
            i = 20;
            System.out.println(1 / 0);
            return i;
        } catch (Exception e) {
            i = 30;
            return i;
        } finally {
            i = 40;
            return i;
        }
    }
```

七、自定义异常

> 1、用户自定义异常类都是RuntimeException的子类
>
> 2、自定义异常类通常需要编写几个重载的构造器
>
> 3、自定义的异常类对象通过throw抛出
>
> 4、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
>
> 5、用户自己的异常类必须继承现有的异常类

```
public class MyRuntimeException extends RuntimeException {
    public MyRuntimeException() {
        super();
        // TODO Auto-generated constructor stub
    }
    public MyRuntimeException(String message, Throwable cause) {
        super(message, cause);
        // TODO Auto-generated constructor stub
    }
    public MyRuntimeException(String message) {
        super(message);
        // TODO Auto-generated constructor stub
    }
}
```

```
            int i = 0;
            if (i == 0) {
                throw new MyRuntimeException("除数不能为0");
            } else {
                System.out.println("能正常使用");
            }
        }
```

1、throws和throw的区别

> 1、throws
> 用在方法声明后面，跟的是异常类名
> 可以跟多个异常类名，用逗号隔开
> 表示抛出异常，由该方法的调用者来处理
> 2、throw
> 用在方法体内，跟的是异常对象名
> 只能抛出一个异常对象名
> 表示抛出异常，由方法体内的语句处理

2、案列<断言>

> 1、断言方便代码的调用与维护

```
    public  static void getThisExcepton(final  String message )throws ThisExcepton {
        throw   new ThisExcepton(message);
    }

    public  static void getIsNotNull(String str,final  String message)throws ThisExcepton {
          if(str==null){
              getThisExcepton(message);
          }
    }
```

八、异常注意事项

> 1、子类重写父类方法时，子类的方法必须抛出相同的异常或父类异常的子类
>
> 2、如果父类抛出了多个异常,子类重写父类时,只能抛出相同的异常或者是他的子集,子类不能抛出父类没有的异常
>
> 3、如果被重写的方法没有异常抛出,那么子类的方法绝对不可以抛出异常,如果子类方法内有异常发生,那么子类只能try,不能throws