Java异常机制

一、什么叫异常

现实生活中，什么叫异常？

出现了不正常的事件，导致事物原有的秩序被破坏。

异常状况。

一旦异常状况出现，需要处理这个情况，比如：交通事故。一旦交通事故出现，势必交警就得闪亮登

场，处理掉交通事故，以恢复交通秩序。

编程中的异常

在你的程序中，可能会出现你的逻辑之外的事件，这些事件会导致你的程序出现或大或小的问题，问题

严重的，会导致系统崩溃，程序终端，问题小的，只是出现一个异常报错，但是不会终止你的程序运行。

一旦程序中出现这些情况，我们就得处理这些情况。

这就是异常处理机制！

二、异常处理

实例：

```
    private static int div(int i, int j) {
        int result = 0;
        try {
            result = i / j;
        } catch (Exception e) {
            System.out.println("出错啦，除数不能为0");
        }

        return result;
    }
```

在该例子中，try...catch包含的位置，就是java的异常处理。

什么叫做异常处理？

异常处理机制能让程序在异常发生时，按照代码的预先设定的异常处理逻辑，针对性地处理异常，让程序尽最大可能恢复正常并继续执行，且保持代码的清晰。

java中异常的分类

第一个维度：

问题的严重度

Error

错误。一种非常严重的问题，同时也是一种比较少见的问题。代表了JVM本身的错误，错误不能被程序员通过代码处理，一旦Error出现，就意味这程序终止。

Exception

异常。

第二个维度

从异常的处理角度

检查异常（非运行时异常，Exception）

checked exception

编译器能够检测出来的异常。

如果异常存在，编译就通不过。

必须要在编译前修正。如何修正？要么自己try。。catch，要么使用throw往外抛。甩锅！

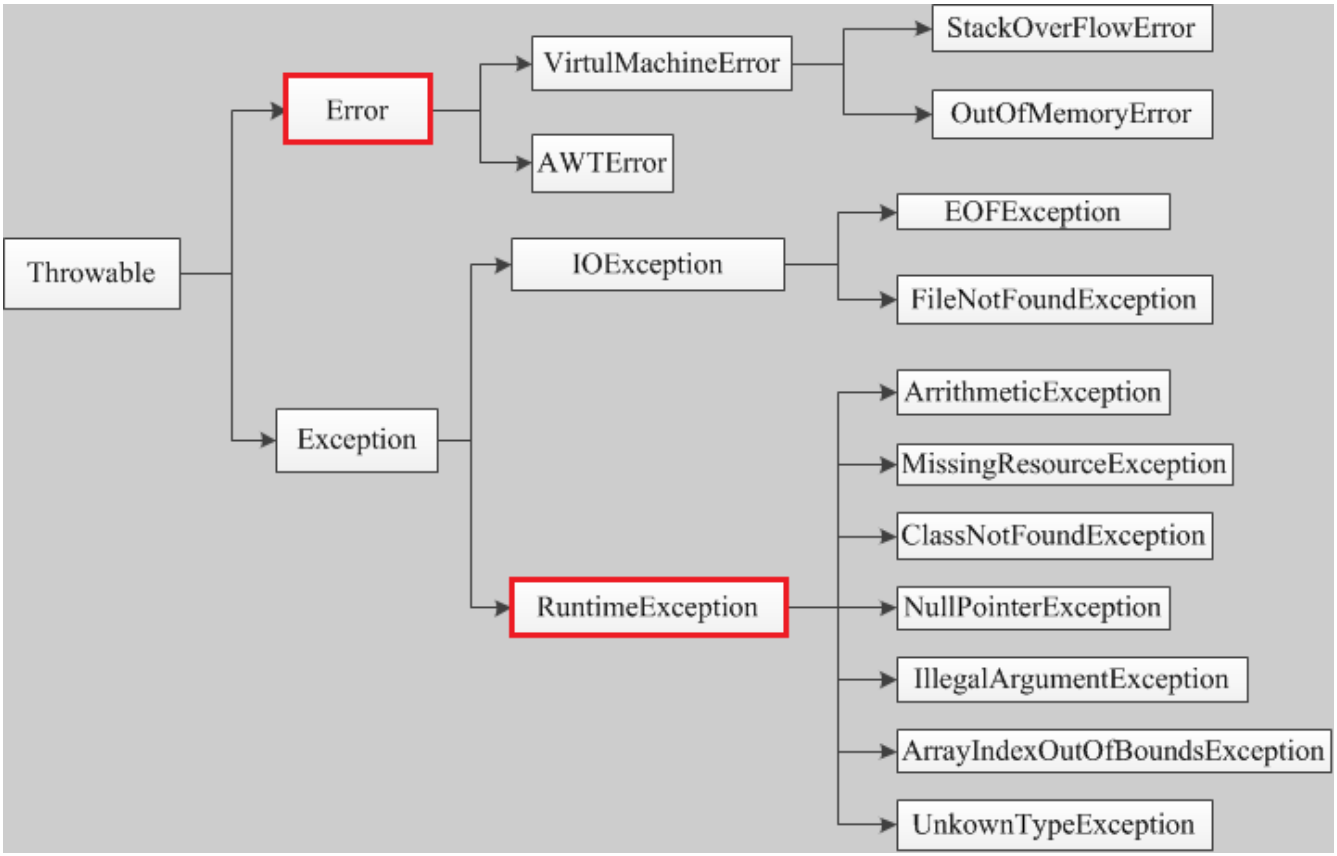
非检查异常（运行时异常、RuntimeException）

unckecked exception

包括：Error（错误）和RuntimeException（运行时异常）

编译器检测不出来的异常，必须是在运行的时候才会出现的异常。

此时，编译器检测不出来，不能提前处理。必须要使用java的异常处理机制。



在java中，异常的顶级父类是Throwable，该类有两个子类，分别代表错误Error和异常Exception。

所有的错误的父类都是Error，所有的异常的父类都是Exception

```
try{
      //try块中放可能发生异常的代码。
      //如果执行完try且不发生异常，则接着去执行finally块和finally后面的代码（如果有的话）。
      //如果发生异常，则尝试去匹配catch块。

}catch(SQLException SQLexception){
     //每一个catch块用于捕获并处理一个特定的异常，或者这异常类型的子类。Java7中可以将多个异常声明在一个catch中。（ catch(ArithmeticException | NumberFormatException e) ）
     //catch后面的括号定义了异常类型和异常参数。如果异常与之匹配且是最先匹配到的，则虚拟机将使用这个catch块来处理异常。
     //在catch块中可以使用这个块的异常参数来获取异常的相关信息。异常参数是这个catch块中的局部变量，其它块不能访问。
     //如果当前try块中发生的异常在后续的所有catch中都没捕获到，则先去执行finally，然后到这个函数的外部caller中去匹配异常处理器。
     //如果try中没有发生异常，则所有的catch块将被忽略。

}catch(Exception exception){
     //...
}finally{

     //finally块通常是可选的。
    //无论异常是否发生，异常是否匹配被处理，finally都会执行。
```

```
    //一个try至少要有一个catch块，否则， 至少要有1个finally块。但是finally不是用来处理异常的，finally不会捕获异常。
   //finally主要做一些清理工作，如流的关闭，数据库连接的关闭等。
}
```

注意事项：

1、try块中的局部变量和catch块中的局部变量（包括异常变量），以及finally中的局部变量，他们之间不可共享使用。

2、finally块没有处理异常的能力。处理异常的只能是catch块。

举例

```
private static int test() {
        System.out.println("test");

        int x = 3;
        int y = 2;
        int result = 0;

        try {
            result = x / y;
            Integer.parseInt("a");
            return 100;
        } catch(ArithmeticException | NumberFormatException e) {
            System.out.println("出错了，异常被抓住了");
        }  finally {
            System.out.println("finally");
            return 88;
            //当try中有一个return，finally也有一个return，其最终结果是：finally
        }

        System.out.println(123123123);
        return result;
    }
```

throw和throws

throw：用在catch中，往外抛出指定的异常

throws：用于方法的签名中（声明方法），表示方法内部的符合给定的异常的种类，往外抛出

往上一层抛出异常。

所谓的上一层？方法的调用者。方法的调用时一个链结构，如果抛到最上层，还不处理，只能抛出到JRE，程序会被终止。

要在异常抛到JRE之前，必须要处理掉。

throws

举例代码：

```
public static void main(String[] args) {
        System.out.println("start");
        test();
}

private static void test() {
        Scanner scanner = new Scanner(new FileInputStream(new File("")));
        //编译是不通过的。原因是：IOException是一个检查异常，编译器要求必须要先处理才能编译通过。
}
```

有两种方式来处理

1、在test方法内部，处理掉。不往外抛

```
    private static void test() {
        try {
            Scanner scanner = new Scanner(new FileInputStream(new File("")));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

    }
```

2、往外抛出异常

抛出异常就是将异常交给调用者来处理。

```
    public static void main(String[] args) {
        System.out.println("start");

        try {
            test();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

    private static void test() throws FileNotFoundException {
        Scanner scanner = new Scanner(new FileInputStream(new File("")));

    }
```

在我们的例子中，test方法将异常往外抛，其调用者main接到这个异常，此时：main方法也有两个选择：

1、处理掉

2、再往上抛

但是，因为main已经是最顶层，如果再往外抛，就只能交给JRE，JRE会选择终止你的程序。

throw

```
public class StringDemo {
    public static void main(String[] args) {
        System.out.println("start");
```

```
        try {
            test();
        } catch (Exception e) {
            System.out.println("出异常啦，搞基一般");
        }
    }

    private static void test() {
        // Scanner scanner = new Scanner(new FileInputStream(new File("")));
        throw new NullPointerException("搞基一班");
        //通过throw语句手动显式的抛出一个异常。throw语句的后面必须是一个异常对象
        //throw 语句必须写在函数中，执行throw 语句的地方就是一个异常抛出点，它和由JRE自动形成的异常抛出点没有任何差别。
    }

}
```

三、自定义的异常

所谓的自定义的异常，就是：异常类不是API提供的，而是自己编写的。

通过查看IOException和NullPointerException的源码，我们发现：

1、如果你要定义一个非运行时异常（检查异常，比如：IOException），继承：Exception类

2、如果你要定义一个运行时异常（非检查异常，比如：NullPointerException），继承：RuntimeException

我们通过查看Throwable的API，发现：自定义的异常可以包含如下的构造函数：

- 一个无参构造函数
- 一个带有String参数的构造函数，并传递给父类的构造函数。
- 一个带有String参数和Throwable参数，并都传递给父类构造函数
- 一个带有Throwable 参数的构造函数，并传递给父类的构造函数。

实例:

```
//自定义的异常类，用于当除数为0或者负数的时候，抛出
package com.psfd.api.exception;

public class DevideException extends RuntimeException {

    private static final long serialVersionUID = 9671818970968320 81L;

    public DevideException() {
        super();
    }

    public DevideException(String message) {
```

```
        super(message);
    }

}
```

```
//自定义异常的使用，
public class StringDemo {
    public static void main(String[] args) {
        System.out.println("start");

        try {
            test();
        } catch (NullPointerException e) {
            System.out.println("空指针");
        } catch (DevideException e) {
            e.printStackTrace();
        }
    }

    private static void test() {
        int x = 9;
        int y = -9999;

        //判断除数，当除数<=0，抛出异常。
        if (y <= 0) {
            throw new DevideException("非法的除数 :" + y);
        }

        System.out.println(x / y);
    }

}
```

注意事项：

1、如果想用throw抛出自定义的异常，那么该异常一定要是：运行时异常（非检查异常）。

为什么检查异常不能throw？因为检查异常在编译的时候就不通过，无法执行。

2、通过查看Throwable的API，两个重要的方法：

1）、getMessage 自定义异常的构造方法传入的String，通过getMessage方法得到

2）、printStackTrace 打印错误的堆栈信息。

比如：

```
com.psfd.api.exception.DevideException: 非法的除数 :-9999
    at com.psfd.api.StringDemo.test(StringDemo.java:33)
    at com.psfd.api.StringDemo.main(StringDemo.java:20)
```