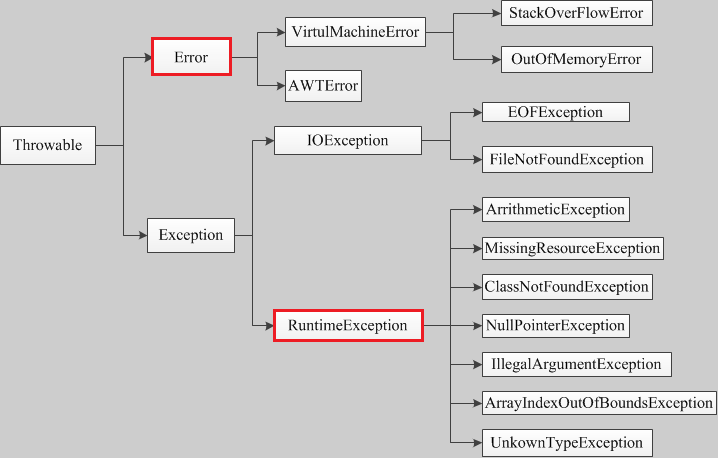
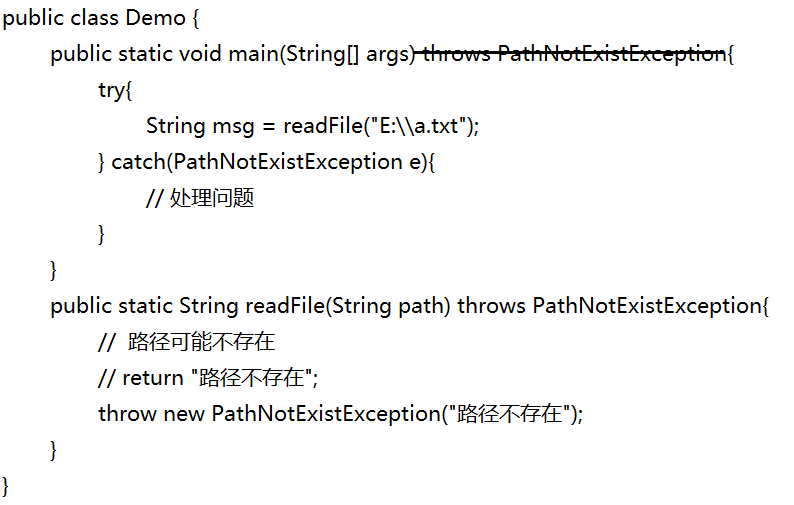
# 1.异常

## 介绍

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中的异常可以是函数中的语句执行时引发的，也可以是程序员通过throw语句手动抛出的，只要在Java程序中产生了异常，就会用一个对应类型的异常对象来封装异常，JRE就会试图寻找异常处理程序来处理异常。





异常是java中一套用于处理和反馈问题的机制。

例如：

ArithmeticException —— 算数异常

ArrayIndexOutOfBoundsException —— 数组下标越界异常

NullPointerException —— 空指针异常

ClassPointerException —— 类型转换异常。

NumberFormatException —— 数字格式化异常。

CloneNotSupportedException —— 克隆异常。

ParseException —— 解析异常

## 分类

### 非检查异常（unckecked exception）

Error 和 RuntimeException 以及他们的子类。

javac在编译时，不会提示和发现这样的异常，不要求在程序处理这些异常。所以如果愿意，我们可以编写代码处理（使用try…catch…finally）这样的异常，也可以不处理。对于这些异常，我们应该修正代码，而不是去通过异常处理器处理 。这样的异常发生的原因多半是代码写的有问题。

如：除0错误ArithmeticException，错误的强制类型转换错误ClassCastException，数组索引越界ArrayIndexOutOfBoundsException，使用了空对象NullPointerException等等。

### 检查异常（checked exception）

除了Error 和 RuntimeException的其它异常。

javac强制要求程序员为这样的异常做预备处理工作（使用try…catch…finally或者throws）。在方法中要么用try-catch语句捕获它并处理，要么用throws子句声明抛出它，否则编译不会通过。这样的异常一般是由程序的运行环境导致的。因为程序可能被运行在各种未知的环境下，而程序员无法干预用户如何使用他编写的程序，于是程序员就应该为这样的异常时刻准备着。

如SQLException 、 IOException、ClassNotFoundException 等。

## 关键字

### throw

在方法内部进行异常声明。

### throws

在方法上进行异常声明。

### try

用来包裹有异常的代码。

### catch

catch结合try使用，catch代码块中一般是解决异常的语句。用来解决有异常的问题。

### finally

finally结合try来使用，无论try是否成功，finally代码块无论如何都会执行一次。经常使用finally来进行善后操作。

#### 坑例

注意：

i++，在return的时候++在后依然不参与。

public static void main(String[] args) {

System.out.println(m()); //结果为4。

}

public static int m() {

int i = 3;

try {

return i++;

} finally {

return i++;

}

}

无论什么时候，都要保证函数有正确的返回值

@SuppressWarnings("finally")

public static int m() {

try {

return 10;

} catch (Exception e) {

e.printStackTrace();

// return 30;

} finally {

return 40;

}

// return 20;

}

Finally中对数据的操作是否影响到return的结果

public static String m() {

Person p = new Person();

try {

p.setName("张三");

// return "张三";

return p.getName();

} catch (Exception e) {

p.setName("李四");

return p.getName();

} finally {

p.setName("王五");

}

}

## 异常类

### Throwable class（异常的顶级父类）

Java标准裤内建了一些通用的异常，这些类以Throwable为顶层父类。Throwable又派生出Error类和Exception类。

#### 构造方法

Throwable();

Throwable(String message);

#### 重要方法

##### printStackTrace();

打印栈轨迹。

##### getMessage();

返回关于发生的异常的详细信息。这个消息在Throwable 类的构造函数中初始化了。

##### getCause();

返回一个Throwable 对象代表异常原因。

##### toString();

使用getMessage()的结果返回类的串级名字。

##### getStackTrace();

返回一个包含堆栈层次的数组。下标为0的元素代表栈顶，最后一个元素代表方法调用堆栈的栈底。返回的是一个StackTraceElement []数组。

##### fillInStackTrace();

用当前的调用栈层次填充Throwable 对象栈层次，添加到栈层次任何先前信息中。

### Error class（错误）

错误，合理的应用程序不应该试图捕获的严重问题。无法处理。

例如：StackOverFlowError

### Exception class（异常）

异常，合理的应用程序想要捕获的条件。可以处理。

根据异常出现的时期不同分为两类：编译时异常、运行市异常。

#### 编译时异常

在编译时期就已经出现了，由于在编译的时候已经出现了，所以要求必须处理。要么抛出，要么捕获。

CloneNotSupportedException —— 克隆不支持异常。

ParseException ——

#### 运行时异常（RuntimeException）

编译的时候不报错的，但是在运行的时候出现。由于在编译的时候不报错，所以可以不处理，也可以处理。往往语法没有错误。

ArithmeticException —— 算术异常。

ArrayIndexOutOfBoundsException —— 数组下标越界异常。

NullPointerException —— 空指针异常。

ClassCastException —— 类型转换异常。

PatternSyntaxException —— 正则格式异常。

NumberFormatException —— 数字格式化异常。

#### 自定义异常

1. 让自己创建的异常类继承Exception。默认是编译时异常，运行时异常需要继承RuntimeException。
2. 定义异常信息属性。可以自定义，也可以使用super调用父类的异常信息属性。
3. 定义构造方法。有参、无参都可以。
4. 定义获取异常信息属性的方法。

##### 例一

@SuppressWarnings("serial")

class FileCannotReadException extends Exception {

public FileCannotReadException() {

}

public FileCannotReadException(String message) {

super(message); //调用父类信息属性。

}

}

##### 例二

@SuppressWarnings("serial")

class PathNotExistException extends Exception {

private String message; //自定义属性。

public PathNotExistException() {

}

public PathNotExistException(String message) {

this.message = message;

}

public String getMessage() {

return message;

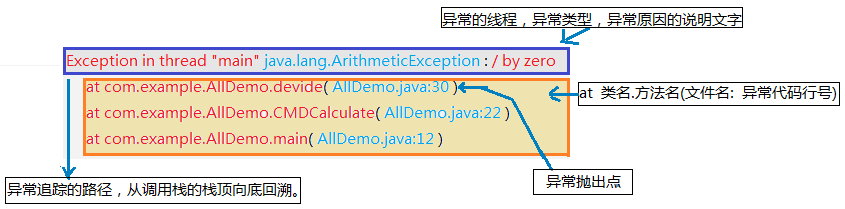
}

}

## 异常的产生

异常是在执行某个函数时引发的，而函数又是层级调用，形成调用栈的，因为，只要一个函数发生了异常，那么他的所有的caller都会被异常影响。当这些被影响的函数以异常信息输出时，就形成的了异常追踪栈。

异常最先发生的地方，叫做异常抛出点。

[](http://www.importnew.com/26613.html/858860-20170911121451422-233079767)

## 异常捕获的方式

### 格式

格式一：

try{

}catch(Exception e){

}

格式二：

try{

}catch(Exception e){

}finally{

}

### 方式

1. 如果出现多个异常的话，可以使用多个catch来分别捕获不同的异常来分别处理。

2. 如果所有异常的处理的方式都一样，可以捕获一个父类异常对象，做统一处理。

3. 如果多个异常需要进行分组处理，同一组异常之间可以使用**|**分隔不同的异常来做分组的处理。这个方式是从JDK1.7开始。

### 特点

1.对于运行时异常而言，可以随意抛出随意捕获；对于编译时异常，只有在抛出的时候才能捕获。

2.如果方法声明throws了一个父类异常，那么在处理的时候必须处理这个父类异常。

3.捕获异常的时候需要先捕获子类异常再捕获父类异常。

4.异常对方法的重载没有任何的影响。方法的重载只和方法签名有关，方法名一致而参数列表不同。

5.子类重写的方法的编译时异常必须是父类方法异常的子类或者本身。子类不能抛出比父类更多的异常，范围不能更大，而且指的是编译时异常。对于运行时异常不做限定。