一、异常

指的是程序在执行过程中，出现的非正常的情况，最终会导致JVM的非正常停止。

1.导致机制其实是帮助我们找到程序中的问题，异常的根类是java.lang.Throwable，其下有两个子类：

java.lang.Error与java.lang.Exception，平常所说的异常指java.lang.Exception。

2.Throwable体系：

Error:严重错误Error,无法处理的错误，只能事先避免，好比绝症

Excepton:表示异常，一场场圣后程序员可以通过代码的方式纠正，使程序继续运行，是必须要处理好的，好比感冒

3.Throwable中的常用方法：

public void printStackTrace():打印异常的详细信息。

(包含了异常的类型,异常的原因,还包括异常出现的位置,在开发和调试阶段,都得使用printStackTrace。)

public String getMessage():获取发生异常的原因

（提示给用户的时候，就提示错误原因）

public String toString:获取异常的类型和异常描述信息（不用）

出现异常不用紧张，把异常的简单类名，拷贝到API中去查

4.异常的分类：

编译时期异常：checked异常。在编译时期，就去检查，如果没有处理异常，则编译失败（如日期格式化异常）

运行时期异常：runtime（RuntimeException）异常。在运行时期，检查异常，运行异常不会编译器检测（不报错）。（如数学异常）

二、异常的处理

1.抛出异常throw

(1)创建一个异常对象，封装一些提示信息（信息可以自己编写）

(2)需要将这个异常对象告知给调用者，怎么告知呢？怎么讲这个异常对象传递到调用者处呢？通过关键字throw就可以完成

throw用在方法内，用来泡出一个异常对象，将这个异常对象传递到调用者处，并结束当前方法的执行。

使用格式：

throw new 异常类名（参数）

例如：

throw new NullPointerExcption("要访问的arr数组不存在")；

throw new ArrayIndexOutOfBoundsExcption("该索引在数组中不存在，已超出范围")；

throw的使用：

public static void main(String[] args) {

int arr[] = {2,4,52,2};

int index = 4;

int element = getElement(arr, index);

System.out.println(element);

System.out.println("over");

}

public static int getElement(int[] arr,int index){

if(index < 0 || index > arr.length-1){

throw new ArithmeticException("哥们角标越界了~~");

}

int element = arr[index];

return element;

}

2.Objects非空判断

public static <T> T requireNonNull(T obj):查看制定引用对象不是null。

public static <T> T requireNonNull(T obj) {

if (obj == null)

throw new NullPointerException();

return obj;

}

3.声明异常throws

修饰符 返回值类型 方法名（参数） throws 异常类名1，异常类名2...{}

public static void main(String[] args) throws FileNotFoundException{

read("a.txt");

}

public static void read (String path) throws FileNotFoundException{

if(!path.equals("a.txt")){

throw new FileNotFoundException("文件不存在哦");

}

}

throws用于进行异常类的声明，若该方法可能有多种异常情况产生，那么在throws后面可以写多个异常类，用逗号隔开。

public static void main(String[] args) throws IOException{

read("a.txt");

}

public static void read(String path)throws FileAlreadyExistsException,IOException{

if(!path.equals("a.txt")){

throw new FileAlreadyExistsException("文件不存在");

}

if(!path.equals("b.txt")){

throw new IOException();

}

}

4.捕获异常try...catch

(1) 该方法不处理,而是声明抛出,由该方法的调用者来处理(throws)。

(2)在方法中使用try-catch的语句块来处理异常。

捕获异常：Java中对异常有针对性的语句进行捕获，可以对出现的异常进行指定方式的处理。

try{

编写可能会出现异常的代码

}catch(异常类型 e){

处理异常的代码

}

try：改代码块中编写可能产生异常的代码

catch：用来进行某种异常的捕获，实现对捕获到的异常进行处理

注意：try和catch都不能单独使用必须连用

public static void main(String[] args) {

try{

read("b.txt");

}catch (FileAlreadyExistsException e){

System.out.println("e");

}

System.out.println("over");

}

public static void read(String path) throws FileAlreadyExistsException{

if(!path.equals("a.txt")){

throw new FileAlreadyExistsException("文件不存在");

}

}

Throwable类中定义了一些查看方法：

public String getMessage():获取异常的描述信息，原因

public String toString():获取异常的类型和异常描述信息

public void printStackTrace():打印异常的跟踪站信息并输出到控制台

5.finally代码块

finally：有一些特定的代码无论异常是否发生，都需要执行。另外，因为异常会引发程序跳转，导致有些语句执行不到。而finally就是解决这个问题的，在finally代码块中存放的代码都是一定会被执行的。

6.异常注意事项

运行时异常被抛出可以不处理。即不捕获也不声明抛出。

如果父类抛出了多个异常，子类覆盖父类方法时，只能抛出相同的异常或者是他的子集。

父类方法没有抛出异常，子类覆盖父类该方法时也不可抛出异常。此时子类产生该异常，只能捕获处理，不能声明抛出

当多异常处理时，捕获处理，前边的类不能是后边类的父类

在try/catch后可以追加finally代码块，其中的代码一定会被执行，通常用于资源回收。

如果finally有return语句,永远返回finally中的结果,避免该情况。