# Java中的异常：java对不正常情况进行描述后的对象体现。

在Java编程过程中，出现问题，要把问题封装成对象，在可能出现问题的地方将异常抛出。而不是用if else 等语句进行处理。

好处是 1．将问题进行封装。

2．将正常流程代码和问题处理代码相分离，方便于阅读。

Error :对于Error一般不编写针对性的代码对其进行处理。

Throwable

(万恶之源) 检查异常(需要处理的异常)：Exception及其子类

Exception

非检查异常(不需要处理)：RuntimeException及其子类

Throwable类中的方法。

* getMessage()

获取异常信息，返回字符串。

* toString()

获取异常类名和异常信息，返回字符串。

* printStackTrace()

打印异常类名、异常信息、以及异常出现的位置。返回值void。

对于运行中出现的RuntimeException，系统自动调用该方法。

* printStackTrace(PrintStream s)

通常用该方法将异常内容保存在日志文件中，以便查阅。

1. 异常处理：try….catch….finally

* 声明异常时，要声明更为具体的异常，调用者在捕捉是可以更为具体的处理
* 声明几个异常，就应对应几个代码块。如果多个catch块出现继承关系，父类异常块应该放在最下面。
* 捕获到的异常时，本层处理不了时，可以继续在catch中抛出。throw e;

catch捕获到异常，处理完后，又产生了新的异常，可在catch中抛出行的异常。

throw new XXXXException(“异常名称”)

* 实际开发中，不要只是简单调用e.printStackTrace()打印异常。
* finally(System.exit(0)除外)一定会执行的语句，通常用于关闭资源。(关闭数据库连接)

1. 自定义异常。

项目中可能出现Java没有封装的异常，这是就需要自定义异常。自动以到了利用Java的面向对象思想，将问题进行封装。

1. throws 和throw

throws使用在函数上。throw使用在函数内。

throws后面跟的异常类。可以跟多个。用逗号隔开。

throw后跟的是异常对象。

1. 异常在子父类覆盖中的体现；

注意事项

1，子类抛出的异常必须是父类的异常的子类或者子集。

1. 如果父类或者接口没有异常抛出时，子类覆盖出现异常，只能try不能抛。
2. RuntimeException (非检查异常)

* 如果在函数内容抛出该异常，函数上可以不用声明，编译一样通过。
* 如果在函数上声明了该异常。调用者可以不用进行处理。编译一样通过。
* 自定义异常时：如果该异常的发生，无法在继续进行运算，就让自定义异常继RuntimeException。

实例。自定义除数不能负数RuntimeException，当除数为负数时，让程序停下来。

//抛出非运行时异常，必须有抛出动作，和抛出声明。调用者必须捕获异常，或抛给更高层次//的调用者,main方法中抛出就抛给JVM………程序停止运行。所以最终调用者一定要去捕获。

class Test{

public void div(int a, int b) throws FuShuException {

if (b<0) { //声明抛出异常，并传递异常名称

throw new FuShuException ("新法则，除数不能为负");

}

}

}

class FuShuException extends Exception{

public FuShuException(String msg) {

//调用父类的构造方法，将异常名称传递进去。

super(msg);

}

}

练习

/\*

毕老师用电脑上课。开始思考上课中出现的问题。

比如问题是(上课过程中出现的异常)

电脑蓝屏。

电脑冒烟。

可是当冒烟发生后，出现讲课进度无法继续。

出现了讲师的问题：课时计划无法完成。

\*/

//定义蓝屏异常

**class** LanPingException **extends** Exception{

LanPingException(String message){

**super**(message);

}

}

//定义冒烟异常

**class** MaoYanException **extends** Exception{

MaoYanException(String message){

**super**(message);

}

}

//定义不能完成计划异常

**class** NoPlanException **extends** Exception{

NoPlanException(String msg){

**super**(msg);

}

}

**class** Computer{

**private** **int** state = 3;

**public** Computer(**int** state) {

**this**.state= state;

}

**public** **void** run()**throws** LanPingException,MaoYanException

{

**if**(state==2)

**throw** **new** LanPingException("蓝屏了");

**if**(state==3)

**throw** **new** MaoYanException("冒烟了");

System.***out***.println("电脑运行");

}

**public** **void** reset(){

state = 1;

System.***out***.println("电脑重启");

}

}

**class** Teacher{

**private** String name;

**private** Computer cmpt;

Teacher(String name){

**this**.name = name;

cmpt = **new** Computer(3);

}

**public** **void** prelect()**throws** NoPlanException

{

**try**{

cmpt.run();

}

**catch** (LanPingException e){

cmpt.reset();

}

**catch** (MaoYanException e){

test();

**throw** **new** NoPlanException("课时无法继续"+e.getMessage());

}

System.***out***.println("讲课");

}

**public** **void** test(){

System.***out***.println("练习");

}

}

**public** **class** ExceptionTest {

**public** **static** **void** main(String[] args)

{

Teacher t = **new** Teacher("毕老师");

**try**{

t.prelect();

}

**catch** (NoPlanException e){

System.***out***.println(e.toString());

System.***out***.println("换老师或者放假");

}

}

}

关于分层思想

1. 数据库访问的分层思想。

//数据提供者，查询数据库，并返回查询数据。并对异常处理

try {

//连接数据库

//数据库操作 如果出现了异常，并被catch

//关闭数据库

} catch (SQLException e) {

e.printStackTrace();

//抛出异常给数据访问者，请求数据不成功

throw new NoDataException();

} finally {

//关闭数据库，这布是必须要有的。

}

//数据请求者

try {

//发送请求

//请求成功

} catch (NoDataException e) {

//请求不成功，并抛出了异常。请求者可以更好的做出处理

e.printStackTrace();

} finally {

//关闭请求，如果是联网的话

}

一个比较经典的面试练习

写出程序结果

class Test{

public static String output="";

public static void foo(int i){

try{

if(i==1)

throw new Exception();

output+="1";

}

catch(Exception e){

output+="2";

return; //return语句执行后，仍然会执行finally

}

finally{

output+="3";

}

output+="4";

}

public static void main(String args[]){

foo(0);

System.out.println(output);//134

foo(1);

System.out.println(output); //13423

}

}