**异常处理**

在进行程序设计时，错误的产生是不可避免的，如何处理错误？把错误交给谁去处理？程序又该如何从错误中恢复？这是任何程序设计语言都要解决的问题。

所谓错误，是在程序运行过程中发生的异常事件，例如除0、数组越界、文件找不到等，这些事件的发生将阻止程序的正常运行。

为了提高程序的健壮性（robust），程序设计时，必须考虑到可能发生的异常事件并做出相应的处理。

**错误的分类：**编译错误、运行错误、逻辑错误。

Java系统根据错误的严重程度不同，将错误分为两类：

* 错误：是致命性的，即程序遇到了非常严重的不正常状态，不能简单地恢复运行。
* 异常：是非致命性的，通过某种修正后程序还能正常运行。

程序运行中的错误，如果都由程序员发现和处理，会增加程序员的负担。而且由于程序员考虑不周、程序运行环境不同等诸多原因，运行程序时还有可能出现各种各样的错误。

开发项目过程中，开发核心业务代码只占20-30%的时间，而开发容错代码的时间却高达70-80%，大大降低了开发效率。Java提供的异常处理机制，可以在一定程度上解决这个问题。

**Java异常处理机制**：Java中声明了很多异常类，每个异常类代表了一种类型的运行错误，类中包含了该种错误的信息和处理错误的方法等内容。每当Java程序运行中发生了一个可识别的运行错误时，系统就为该异常类产生一个异常对象。而一旦有异常对象产生，就可以用相应的机制处理异常，即该错误有一个异常类与之对应时，系统都会产生一个相应的该异常类的对象，即产生一个异常。一旦异常产生了，系统中就一定有相应的机制来处理它，确保不会死机、死循环或其他对操作系统的损害，从而保证了整个程序能够安全运行。

使用Java的异常处理机制有以下优点：

* 将错误处理代码从常规代码中分离出来；
* 按错误类型和差别分组；
* 对无法预测的错误也捕获和处理；
* 克服了传统方法的错误信息有限的问题；
* 把错误传播给调用堆栈。

由于大多Java程序运行在网络环境中，因此安全是首要问题。

Java通过异常类和异常对象处理程序运行中的错误，防止错误对程序造成影响，确保程序能够安全运行。

**异常（Exception）**又称例外/差错/违例，是特殊的运行错误对象，对应着Java语言特定的运行错误处理机制。

异常与其他语言要素一样，是面向对象规范的一部分，是异常类的对象。

下面通过一个例子说明什么是程序运行错误——异常：

public class ExceptionDemo{

public static void main(String[] args){

int a=4,b=0,c[]={1,2,3,4,5};

System.out.println(a/b); //用0作除数

System.out.println(c[a+1]); } //数组下标越界

//throw ArithmeticException e；

System.out.println("end");}

}

该程序中，由于没有语法错误，因此程序通过编译。但是由于变量b的值为0，因此a/b运算存在错误，这个错误在程序运行时暴露出来。

当运行该程序时，Java运行系统发现了用0作为除数这个错误，产生一个类型为ArithmeticException的异常。又由于程序中没有说明如何处理这种错误，因此程序被迫停止运行。

上述只是程序运行异常的一种情况。实际上，程序运行过程中会出现各种各样的异常，任何一种异常，都会影响程序的正常运行。

**Throwable类**属于java.lang包，是所有异常类的父类。在Throwable类中定义了所有异常类共同需要的内容。

Throwable类有两个构造方法：

public Throwable() //无参构造方法

public Throwable(String s) //参数s可以接收传入的字符串信息，该信息通常是对发生的异常的描述。

Throwable类还定义了以下方法：

public String getmessage() //获得传入构造方法的参数，如使用的是无参构造方法，getmessage()获得空值。

public String toString() //获得描述当前异常对象信息的字符串，该字符串中包括异常对象所属的异常类名以及由getmessage()方法获得的信息。

public void printStackTrace() //该方法没有返回值，功能是完成打印操作。使用该方法可以在标准输出设备上打印当前异常对象的堆栈使用轨迹。

Throwable类派生自Object类，它也有两个子类：

**Error类**下的错误都是严重的JVM的错误，用户程序无法进行处理。也就是说，Error类型的错误不允许用户插手处理，由Java系统自行处理。

**Exception类**是Java中所有异常类的父类。

分为检查型/受检型/可控式异常类和非检查型/运行时异常/RuntimeException异常类两种。

一些因编译错误导致的异常，或不期望程序捕获的异常，被命名为非检查型异常并且不需要进行声明，编译器对继承自RuntimeException的异常不做检查。发生运行异常的原因通常是程序对问题考虑不细致、不全面，即程序不够健壮。

其他类型的异常称为检查型异常/受检型/可控式异常类，Java类必须在方法的签名后声明它所抛出的任何检查型异常。对于任何方法，如果它调用的方法抛出一个类型为E的检查型异常，那么调用者就必须捕获E或者也声明抛出E（或者E的一个父类），对此编译器要进行检查。

Java预定义了一些常见的异常：

ArithmeticException

//算术异常类，如除数为0。

NullPointException

//如一个对象还没有被实例化，那么访问该对象或调用它的方法将导致该异常。

NegativeArraySizeException

//创建数组时，如果元素个数为负数，则会引发该异常。

ArrayIndexOutOfBoundsException

//数组下标越界

ArrayStoreException

//数组试图存取数组中错误的数据类型

FileNotFoundException

//试图存取一个并不存在的文件

IOException

//通常的I/O错误

其他参见教材表格。

Exception类的构造方法和常用方法如下：

构造方法：

public Exception()

public Exception(String message) //字符串s是对该异常的描述信息

其常用方法都是从父类Throwable中继承过来的：

public String toString()//返回当前异常对象信息的描述

public String getmessage() //获得当前异常对象的详细描述

getcause()

public void printStackTrace()//用来跟踪异常事件发生时执行堆栈的内容

Java对异常的处理涉及两方面内容：

一是抛出（throw）异常，二是捕获（catch）并处理异常。

如果程序在运行过程中出现了运行错误，就会产生异常对象，我们把生成异常对象并提交给运行时系统的这个过程称为**抛出异常**。

当有异常对象抛出时，运行时系统将在方法的调用栈中查找，从生成异常的方法进行回溯，直到找到包含处理该异常的代码，把异常对象交给该处理代码进行处理，这个过程称为**捕获异常**。

如果异常没有被程序捕获，即程序中没有给出处理异常的代码，则把异常交给Java运行系统默认的处理代码进行处理。默认的处理方式是：首先显示描述异常信息的字符串（java.lang.ArthmeticException），然后终止程序的运行。

抛出异常有两种方式：一种是系统自动抛出(如上述异常交给Java运行系统默认的处理代码进行处理)，另一种是人为抛出异常。

**系统自动抛出异常：**在程序运行过程中，如果出现了可以被Java运行系统识别的异常，系统会自动产生与该异常相对应的异常类对象，即自动抛出异常。

public class ArithmeticExceptionDemo{

public static void main(String[] args){

int a=4,b=0;

System.out.println(a/b); //用0作除数

System.out.println("end");}

}

运行该程序时会自动抛出算术异常类ArithmeticException的一个异常对象，该异常对象被Java运行系统默认处理器捕捉并处理。处理结果是：首先显示描述该异常信息的字符串java.lang. ArithmeticException，然后终止运行。

对于检查型异常，Java强迫程序必须进行处理。处理方法有两种：

* 声明抛出异常：不在当前方法内处理异常，而是把异常抛给调用方法。
* 捕获并处理异常：使用try…catch…捕获到所发生的异常，并进行相应的处理。

**人为抛出异常：**有两种方式；一种是在方法签名后用throws声明需要抛出的异常，另一种是在方法体内用throw语句抛出异常。

**Throws**抛出异常

修饰符 返回值类型 方法名([形参列])throws 异常类1,异常类2,...

{…}

运行该方法过程中，如果出现了由throws列出的异常，则可以抛出异常，并在程序中寻找处理这个异常的代码；如果程序中没有给出处理异常的代码，则把异常交给Java运行系统默认的处理代码进行处理。例如：

public class ThrowsDemo{

public static void main(String[] args)

throws ArithmeticException,ArrayIndexOutOfBoundsException{

int a=4,b=0,c[]={1,2,3,4,5};

System.out.println(a/b); //用0作除数

System.out.println(c[a+1]); //数组下标越界

System.out.println("end");}

}

该程序中，main()方法可以抛出ArithmeticException和ArrayIndexOutOfBoundsException两种异常，前者为算术异常，后者为数组下标越界异常。由于程序中没有给出处理异常的代码，因此出现异常后，直接由Java运行系统进行处理。

**throw**抛出异常

前面提到的异常或者是由JVM生成，或者是由Java类库中的类生成。事实上，在程序中也可以生成自己的异常对象，也就是异常可以不是出错产生，而是人为的产生。这种需要在方法体内某个位置抛出异常，可以使用：

throw new 异常类();

或者：

异常类 e= new 异常类();

throw e;

执行throw语句时，程序终止执行后面的语句，在程序中寻找处理异常的代码，如没有给出处理代码，则把异常交给Java运行系统处理。例如：

public class ThrowDemo1{

public static void main(String[] args){

ArithmeticException e=new ArithmeticException();

//定义算术异常对象e

int a=4,b=0;

System.out.println("Before ArithmeticException");

If (b=0) throw e; //抛出算术异常对象e

System.out.println(a/b); //该语句不被执行 }

}

通常将throw语句与if语句结合起来使用，即只有满足某个条件时，才执行throw语句。上述程序中，通过if语句检查变量b是否为0，如果为0，则执行throw语句，即抛出异常对象e。抛出e后，程序不再继续往下执行，而是在程序中寻找处理异常的代码。又由于程序中没有给出处理异常的代码，因此交给Java运行系统处理。

public class ThrowDemo2{

public static void main(String[] args){

int c[]={1,2,3,4,5},a=5;

System.out.println("Before throw ArrayIndexOutOfBoundsException");

if (a>4) throw (new ArrayIndexOutOfBoundsException());

//抛出数组下标越界异常

System.out.println("After throw ArrayIndexOutOfBoundsException");

//不执行该语句}

}

上述程序通过if语句检查变量a是否大于4，如果大于4，则产生一个ArrayIndexOutOfBoundsException类的对象，并抛出该异常对象，交给Java运行系统处理。

public class ThrowDemo3{

public static void main(String[] args){

int a=4,b=0,c[]={1,2,3,4,5};

System.out.println("Before throw");

if (b==0) throw (new ArithmeticException()); //抛出算术异常

System.out.println(a/b); //该语句被执行

if (a>4) throw (new ArrayIndexOutOfBoundsException());

//抛出数组下标越界异常

System.out.println("After throw");//不执行该语句 }

}

上述程序给出两个throw语句。由于第一个if语句不满足条件，因此没有抛出算术异常，程序可以继续往下执行。由于第二个if语句满足条件，因此抛出数组下标越界异常，程序不再往下执行，又由于程序中没有处理异常的代码，将异常交给Java运行系统处理。

在前面的例子中，由于程序中都没有给出处理异常的代码，因此抛出的异常都被Java运行系统捕获并进行相应的处理。一般在设计程序时，如果能够预测程序中可能发生的异常，则应在程序中给出处理代码，除非对于那些不能预测的异常，才交给Java运行系统处理。

**try…catch…** 程序自己捕获和处理异常，格式如下：

修饰符 返回值类型 方法名([形参列表])

{不发生异常的程序段1

try{可能发生异常的程序段}

catch(异常类1 对象1){处理异常语句组1}

catch(异常类2 对象2){处理异常语句组2}

…

catch(异常类n 对象n){处理异常语句组n}

不发生异常的程序段2

}

说明：

1.每个catch块用于处理一种类型的异常，多个catch块时，特殊的、具体的子类异常catch块写在前，一般的、普通的父类异常catch块写在后。

2.按顺序执行try块中的每条语句。如中途发生异常，则抛出异常并跳出try块，在后面的catch块中自上而下寻找与该异常相匹配的catch块，找到第一个相匹配的catch块，则停止寻找，进入该catch块中执行语句。其他catch块都被忽略。之后执行其后的不发生异常的程序段2，如果所有catch块都不与之匹配，则将异常交给Java运行系统处理。

public class Try\_catchDemo1{

public static void main(String[] args){

int a,b=0;

try{

System.out.println("抛出异常之前");

a=5/b; //在此抛出异常对象

System.out.println("之前有异常抛出，该语句不被执行");

}

catch(ArithmeticException e)//被选中，处理算术异常catch块

{

System.out.println("处理算术异常的catch块捕获了异常");

System.out.println("捕获的异常类型是"+e);

}

catch(ArrayIndexOutOfBoundsException e)//没选中

{

System.out.println("处理数组下标越界异常的catch块捕获了异常");

System.out.println("捕获的异常类型是"+e);

}

System.out.println("catch块后面的语句");}

}

**注意：**无论给出多少个catch块，最多只能选中一个执行，如果多个catch块都与该异常对象的类型匹配，则只选中写在前面的catch块。

public class Try\_catchDemo2{

public static void main(String[] args){

int a=5,c[]={1,2,3,4,5};

try{System.out.println("抛出异常之前");

if (a>4) throw (new ArrayIndexOutOfBoundsException());

System.out.println("抛出异常之后");}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("处理数组下标越界异常的catch块捕获了异常");

System.out.println("捕获的异常类型是"+e);}

System.out.println("catch块后面的语句");}

}

**finally**

如果对方法中发生的各种异常有统一的处理，可以将完成统一处理的程序段放在finally块中。

说明：

1.finally块紧跟最后一个catch块后，是每个catch块的共用部分，程序必须无条件执行其中的程序段。即无论try块是否抛出异常，也无论catch块的异常类型是否与抛出的异常类型匹配，finally块中的程序段都要执行。通常把对各种异常的共同处理部分放在finally块中，如：输出统一信息、释放资源、清理内存、关闭已打开的文件等。

2. finally块特别适用于对文件的统一处理。编写处理文件的方法时，通常采用下面的格式：

public void 方法名([形参列表]){

try{对文件做处理的语句}

catch(FileNotFoundException e){处理文件未找到的异常}

catch(IOException e){处理输入输出的异常}

finally{关闭文件}

}

使用finall块举例：

public class FinallyDemo{

public static void main(String[] args){

int a,b=0;

try{System.out.println("抛出异常之前");

a=5/b; //在此抛出异常对象

System.out.println("之前有异常抛出，该语句不被执行");}

catch(ArithmeticException e)//被选中，处理算术异常catch块

{

System.out.println("处理算术异常的catch块捕获了异常");

System.out.println("捕获的异常类型是"+e);

}

catch(ArrayIndexOutOfBoundsException e)//没选中

{

System.out.println("处理数组下标越界异常的catch块捕获了异常");

System.out.println("捕获的异常类型是"+e);

}

finally{System.out.println("这是所有catch块共有的部分");

System.out.println("还可以在此给出其他语句");}

System.out.println("catch块后面的语句");}

}

**自定义异常类**

尽管Java提供了很多异常类，但用户还是可以根据需要定义自己的异常类，即创建自定义异常类。自定义异常类必须是Throwable类或Exception类的子类。

格式如下：

public class 自定义异常类名 extends 异常类名

{…}

public class TestNewException{

public static void main(String[] args){

try{

System.out.println("2+3="+add(2,3));

System.out.println("-8+10="+add(-8,10));}

catch(Exception e){

System.out.println("异常类是"+e);}

static int add(int n,int m) throws NewException{

if (n<0||m<0) throw new UserException();//抛出NewException异常

return n+m;}

}

//自定义异常类

class NewException extends Exception{

UserException(){super("数据为负数");}

}

**自定义异常类步骤**：

1.创建自定义异常类（extends Exception）

2.在方法体中通过Throw关键字抛出异常对象（throw new Exception()）

3.如果在当前抛出异常的方法中处理异常，可以使用try-catch语句捕获并处理；否则在方法名后通过Throws关键字声明要将该类型的异常抛给方法调用者，继续进行下一步操作。

4.在其调用者中使用try-catch捕获并处理异常。(多层调用可能会涉及再次抛出异常，直到顶层调用者必须提供捕获并处理异常的代码)

**重抛异常**及异常链接：

对于异常，不仅要进行捕获并处理，有时还需要将此异常进一步传递给调用者，以便让调用者也能感受到这种异常。这时可以在catch或finally语句组中采取以下三种方式：

1.将当前捕获的异常对象再次抛出,如：throw e;

2.重新生成一个异常对象，并抛出，如：throw new Exception("message");

3.重新生成并抛出一个新异常对象，该异常对象中包含了当前异常的信息，如：throw new Exception("message",e);然后可以用e.getCause()来得到内部异常。

本章补充的异常类：

表10-1可控式异常（checked Exception），也称受检的异常。

FileNotFoundException//文件未找到异常

可控式异常要求明确进行语法处理：

* 要么捕获(catch)
* 要么抛出(throws)

用throws声明来抛出异常：

在子类中，如果要重写父类的一个方法，假如该父类的这个方法用throws声明抛出了某种类型的异常，则子类重写的方法也可以用throws抛出异常。可以抛出子类异常（即更具体、更少或更特殊的异常），但不能抛出更一般、更多或更普通的异常。

表10-2 运行时异常：可以不明确处理，也可以用if语句去处理。

IndexOutOfBoundsException分为ArrayIndexOutOfBoundsException和StringIndexOutOfBoundsException

NumberFormatException //数字格式化异常

IllegalAccessException //非法访问异常