**异常处理使得程序可以处理非预期的情景，并且继续正常的处理。**

**在程序运行过程中，如果JVM检测出一个不可能执行的操作，就会出现运行时错误（runtime error）。**

**在java中，运行时错误会作为异常抛出。异常就是一种对象，表示阻止正常进行程序执行的错误或者情况。如果异常没有被处理，那么程序将会非正常终止。**

**异常处理概述**

**异常是从方法抛出的。方法的调用者可以捕获以及处理该异常。**

**throw语句的执行称为抛出一个异常。异常就是一个从异常类创建的对象。**

**当异常被抛出时，正常的执行流程就被打断。“抛出异常”就是将异常从一个地方传递到另一个地方。调用方法的语句包含在一个try块和一个catch块中。try块包含了正常情况下执行的代码。异常被catch块所捕获。catch块中的代码执行以处理异常。之后，catch块之后的语句被执行。**

**throw语句类似于方法的调用，但不同于调用方法的是，它调用的是catch块。从某种意义来讲，catch块就像带参数的方法定义，这些参数匹配抛出的值的类型。但是，它不像方法，在执行完catch块之后，程序控制不反回throw语句；而是执行catch块之后的下一条语句。**

**catch块的头部**

**catch(ArithmeticException ex)**

**标识符ex的作用很像是方法中的参数。所以，这个参数称为catch块的参数。ex之前的类型指定了catch块可以捕获的异常类型。一旦捕获该异常，就能从catch块体重的参数访问这个抛出的值。**

**一个异常可能是通过try块中的throw语句直接抛出，或者调用一个可能会抛出异常的方法而抛出。**

**异常类型**

**异常是对象，而对象都采用类来定义。异常的根类是java.lang.Throwable。**

**Throwable类是所有异常类的根类。所有的java异常类都直接或者间接地继承自Throwable。可以通过继承Exception或者Exception的子类来创建自己的异常类。**

**这些异常类可以分为三种主要类型：系统错误、异常和运行时异常。**

**1）系统错误是由java虚拟机抛出的，用Error类表示。Error类描述的是内部系统错误。这样的错误很少发生。如果发生，除了通知用户以及尽量稳妥地终止程序外，几乎什么也不能做。**

**2）异常是用Exception类表示的，它描述的是由程序和外部环境所引起的错误，这些错误能被程序捕获和处理。**

**3）运行时异常是用RuntimeException类表示的，它描述的是程序设计错误，例如，错误的类型转换、访问一个越界数组或数值错误。运行时异常通常是由java虚拟机抛出的。**

**RuntimeException、Error以及它们的子类都称为免检异常（unchecked Exception）。所有其他异常都称为必检异常（checked Exception），意思是指编译器会强制程序员检查并通过try-catch块处理它们，或者在方法头进行声明。**

**在大多数情况下，免检异常都会反映出程序设计上不可恢复的逻辑错误。例如，如果通过一个引用变量访问一个对象前并未将一个对象赋值给它，就会抛出NullPointerException异常；如果访问一个数组的越界元素，就会抛出IndexOutOfBoundsException异常。这些都是程序必须纠正的逻辑错误。免检异常可能在程序的任何一个地方出现。为避免过多的使用try-catch块，java语言不强制要求编写代码捕获或声明免检异常。**

**关于异常处理的更多知识**

**异常的处理器是通过从当前的方法开始，沿着方法调用链，按照异常的反向传播方向找到的。**

**java的异常处理模型基于三种操作：声明一个异常、抛出一个异常和捕获一个异常。**

**声明异常**

**java解释器调用main方法开始执行一个程序。每个方法都必须声明它可能抛出的必检异常的类型。这称为声明异常。因为任何代码都可能发生系统错误和运行时错误，因此，java不要求在方法中显式声明Error和RuntimeException（免检异常）。但是，方法要抛出的其他异常都必须在方法头中显式声明，这样，方法的调用者会被告知有异常。**

**为了在方法中声明一个异常，就要在方法头中使用关键字throws，如下**

**public void myMethod() throws IOException**

**关键字throws声明myMethod方法可能会抛出异常IOException。如果方法可能会抛出多个异常，就可以在关键字throws后面添加一个用逗号分隔的异常列表。**

**如果方法没有在父类中声明异常，那么就不能在子类中对其进行继承来声明异常。**

**抛出异常**

**检测到错误的程序可以创建一个合适的异常类型的实例并抛出它，这就称为抛出一个异常。例如**

**假如程序发现传递给方法的参数与方法的合约不符，这个程序就可以创建IllegalArgumentException的一个实例并抛出它，如下**

**IllegalArgumentException ex=new IllegalArgumentException("Wrong Argument");**

**throw ex;**

**或者**

**throw new IllegalArgumentException("Wrong Argument");**

**IllegalArgumentException是JavaAPI中的一个异常类。通常，JavaAPI中的每个异常类至少有两个构造方法：一个无参构造方法和一个带可描述这个异常的String参数的构造方法。该参数称为异常消息，它可以用getMessage()获取。**

**声明异常的关键字是throws，抛出异常的关键字是throw。**

**捕获异常**

**当抛出一个异常时，可以在try-catch块中捕获和处理它。**

**如果在执行try块的过程中没有出现异常，则跳过catch子句。**

**如果try块中的某条语句抛出一个异常，java就会跳过try块中剩余的语句，然后开始查找处理这个异常的代码的过程。**

**处理这个异常的代码称为异常处理器。可以从当前的方法开始，沿着方法调用链，按照异常的反向传播方向找到这个处理器。从第一个到最后一个逐个检查catch块，判断在catch块中的异常类实例是否是该异常对象的类型。如果是，就将该异常对象赋值给所声明的变量，然后执行catch块中的代码。如果没有发现异常处理器，java会推出这个方法，把异常传递给调用这个方法的方法，继续同样的过程来查找处理器。如果在调用的方法链中找不到处理器，程序就会终止并且在控制台上打印错误信息。寻找处理器的过程称为捕获一个异常。**

**从一个通用的父类可以派生出各种异常类。如果一个catch块可以捕获一个父类的异常对象，它就能捕获那个父类的所有子类的异常对象。**

**在catch块中异常被指定的顺序是非常重要的。如果父类的catch块出现在子类的catch块之前，就会导致编译错误。**

**java强迫程序员处理必检异常。如果方法声明了一个必检异常（即Error或Runtime Exception之外的异常），就必须在try-catch块中调用它，或者在调用方法中声明要抛出异常。**

**从异常中获取信息**

**异常对象包含关于异常的有价值的信息。可以利用下面这些java.lang.Trhowable类中的实例方法获取有关异常的信息。**

**finally子句**

**无论异常是否产生，finally子句总是会被执行的。**

**三种情况：**

**1）如果try块中没有出现异常，执行finalstatements，然后执行try语句的下一条语句。**

**2）如果try块中有一条语句引起异常，并被catch块捕获，然后跳过try块的其他语句，执行catch块和finally子句。执行try语句之后的下一条语句。**

**3）如果try块中有一条语句引起异常，但是没有被任何catch块捕获，就会跳过try块中的其他语句，执行finally子句，并且将异常传递给这个方法的调用者。**

**即使在到达finally块之前有一个return语句，finally块还是会执行。**

**使用finally块时可以省略catch块。**

**何时使用异常**

**当错误需要被方法的调用者处理的时候，方法应该抛出一个异常。**

**异常处理将错误处理代码从正常的程序设计任务中分离出来，这样，可以使程序更易读、更易修改。但是，由于异常处理需要初始化新的异常对象，需要从调用栈返回，而且还需要沿着方法调用链来传播异常以便找到它的异常处理器，所以，异常处理通常需要更多的时间和资源。**

**异常出现在方法中。如果想让该方法的调用者处理异常，应该创建一个异常对象并将其抛出。如果能在发生异常的方法中处理异常，那么久不需要抛出或使用异常。**

**一般来说，一个项目中多个类都会发生共同异常应该考虑作为一种异常类。对于发生在个别方法中的简单错误最好进行局部处理，无须抛出异常。**

**重新抛出异常**

**如果异常处理器不能处理一个异常，或者只是简单地希望它的调用者注意到该异常，java允许该异常处理器重新抛出异常。**

**try{**

**statements；**

**}**

**catch（TheException ex）{**

**perform operations before exits；**

**throw ex；**

**}**

**语句throw ex重新抛出异常给调用者，以便调用者的其他处理器获得处理异常ex的机会。**

**创建自定义异常类**

**可以通过派生java.lang.Exception类来定义一个自定义异常类。**

**java提供相当多的异常类，尽量使用它们而不要创建自己的异常类。然而，有时候可以通过派生Exception类或其子类，例如，IOException，来创建自己的异常类。**

**File类**

**File类包含了获得一个文件/目录的属性，以及对文件/目录进行改名和删除的方法。**

**File类意图提供了一种抽象，这种抽象是指以不依赖机器的方式来处理很多以来与机器的文件和路径名的复杂性。File类包含许多获取文件属性的方法，以及重命名和删除文件和目录的方法，但是File类不包含读写文件内容的方法。**

**文件名是一个字符串。File类是文件名及其目录路径的一个包装类。例如，**

**在windows中，语句new File("c:\\book")在目录c:\\book下创建一个File对象，而语句new File("c:\\book\\test.dat")为文件c:\book\test.dat创建一个File对象。可以用File类的isDirectory()方法来判断这个对象是否表示一个目录，还可以用isFile()方法来判断这个对象是否表示一个文件名。**

**在Windows中目录的分隔符是反斜杠\，但是在java中，反斜杠是一个特殊的字符，应该写成 \\ 的形式。**

**构建一个File实例并不会在机器上创建一个文件。不管文件是否存在，都可以创建任意文件名的File实例。可以调用File实例上的exists()方法来判断这个文件是否存在。**

**在程序中，不要直接使用绝对文件名。有可能在Windows上能工作，而不能在其他平台上工作。应该使用与目前目录相关的文件名。例如，new File("image/us.gif")为在当前目录下的images目录下的us.gif创建一个File对象，斜杠是java目录分隔符，这点和unix是一样的。**

**文件输入和输出**

**使用Scanner类从文件中读取文本数据，使用PrintWriter类向文本写入数据。**

**File对象封装了文件或路径的属性，但是它既不包括创建文件的方法，也不包括从/向文件读/写数据（称为数据输入输出，简称I/O）的方法。**

**使用PrintWriter写数据**

**java.io.PrintWriter类可用来创建一个文件并向文本文件写入数据。首先，必须为一个文本文件创建一个PrintWriter对象，如下**

**PrintWriter output=new PrintWriter(filename);**

**然后，可以调用PrintWriter对象上的print、println和printf方法向文件写入数据。**

**PrintWriter(file:File)**

**PrintWriter(filename:String)**

**print(s:String):void**

**print(c:char):void**

**print(cArray:char[]):void**

**print(i:int):void**

**print(l:long):void**

**print(f:float):void**

**print(d:double):void**

**print(b:boolean):void**

**....**

**如下创建一个PrintWriter实例，并且向文件score.txt中写入两行数据的例子。每行都包括名字（字符串）、中间名字的首字母（字符）、姓（字符串）和分数（整数）。**

public class WriteData {

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

java.io.File file = new java.io.File("scores.txt");

if (file.exists()) {

System.out.println("File already exists");

System.exit(0);

}

// Create a file

java.io.PrintWriter output = new java.io.PrintWriter(file);

// Write formatted output to the file

output.print("John T Smith ");

output.println(90);

output.print("Eric K Jones ");

output.println(85);

// Close the file

output.close();

}

}

调用PrintWriter的构造方法可能会抛出某种I/O异常。java强制要求编写代码来处理这类异常。

我们已经使用过System.out.print方法像控制台输出文本。System.out是控制台的标准java对象。可以创建对象，然后使用print向文件中写入文本。

必须使用close()方法关闭文件。如果没有调用该方法，数据就不能正确保存再文件中。

使用try-with-resources自动关闭资源

语法：

try（声明和创建资源）{

使用资源来处理文件;

}

重写上述代码

public class WriteDataWithAutoClose {

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

java.io.File file = new java.io.File("scores.txt");

if (file.exists()) {

System.out.println("File already exists");

System.exit(0);

}

try (

// Create a file

java.io.PrintWriter output = new java.io.PrintWriter(file);

) {

// Write formatted output to the file

output.print("John T Smith ");

output.println(90);

output.print("Eric K Jones ");

output.println(85);

}

}

}

**关键字try后声明和创建了一个资源。资源放在括号中。资源必须是AutoCloseable的子类型，比如PrinterWriter，具有一个close()方法。**

**资源的声明和创建必须在同一行语句中，可以在括号中进行多个资源的声明和创建。**

**使用try-with-resourse不仅可以避免错误，而且可以简化代码。**

**使用Scanner读数据**

**java.util.Scanner类用来从控制台读取字符串和基本类型数值。Scanner可以将输入分为由空白字符分隔的标记。为了能从键盘读取，需要为System.in创建一个Scanner，如下**

**Scanner input=new Scanner(System.in);**

**为了从文件中读取，为文件创建一个Scanner,如下**

**Scanner input=new Scanner(new File(filename));**

**以下代码创建了一个Scanner实例，并从文件scores.txt中读取数据。**

**public class ReadData {**

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

**// Create a File instance**

**java.io.File file = new java.io.File("scores.txt");**

**// Create a Scanner for the file**

**Scanner input = new Scanner(file);**

**// Read data from a file**

**while (input.hasNext()) {**

**String firstName = input.next();**

**String mi = input.next();**

**String lastName = input.next();**

**int score = input.nextInt();**

**System.out.println(**

**firstName + " " + mi + " " + lastName + " " + score);**

**}**

**// Close the file**

**input.close();**

**}**

**}**

**Scanner如何工作**

**方法nextByte()、nextShort()、nextInt()、nextLong()、nextDouble()和next()等都被称为标记读取方法（token-reading method），因为它们会读取用分隔符分隔开的标记。默认情况下，分隔符是空格。可以使用useDelimiter(String regex)方法设置新的分隔符模式。**

**一个标记读取方法首先跳过任意分隔符（默认情况下是空格），然后读取一个以分隔符结束的标记。然后，对应于nextByte()、nextShort()、nextInt()、nextLong()、nextFloat()和nextDouble()，这个标记就分别被自动地转换为一个byte、short、int、long、long和double型的值。对于next()方法而言是无须转换的。如果标记和期望的类型不匹配，就会抛出一个运行异常java.util.InputMismatchException。**

**方法next()和nextLine()都会读取一个字符串。next()方法读取一个由分隔符分隔的字符串，但是nextLine()读取一个以换行符结束的行。**

**行分隔符字符串是由系统定义的，在Windows平台上是\r\n，而在UNIX平台上是\n。为了得到特定平台上的行分隔符，使用**

**String lineSeparator=System.getProperty("line.separator");**

**如果从键盘输入，每行就以回车键结束，它对应于\n字符。**

**标记读取方法不能读取标记后面的分隔符。如果在标记读取方法之后调用nextLine()，该方法读取从这个分隔符开始，到这行的行分隔符结束的字符。这个行分隔符也被读取，但是它不是nextLine()返回的字符串部分。**

**从Web上读取数据**

**如同从电脑中的文件中读取数据一样，也可以从Web上的文件中读取数据。**

**除开从电脑中的本地文件或者文件服务器中读取数据，如果知道Web上文件的URL，也可以从Web上访问数据。**

**为了读取一个文件，首先要使用java.net.URL类的这个构造方法，为该文件创建一个URL对象。**

**public URL(String spec)throws MalformedURLException**

**例如**

**try{**

**URL url=new URL("http://www.google.com/index.html");**

**}**

**catch(MalformedURLException ex){**

**ex.printStackTrace();**

**}**

**如果URL字符串出现语法错误的话，将会有一个MalformedURLException被抛出。**

**创建一个URL对象后，可以使用URL类中定义的openStream()方法来打开输入流和用输入流创建如下Scanner对象。**

**Scanner input=new Scanner(url.openStream());**

**现在可以从输入流中读取数据了，如同从本地文件中读取一样。**

**下面代码提示用户输入一个URL，然后显示文件的大小**

import java.util.Scanner;

public class ReadFileFromURL {

public static void main(String[] args) {

System.out.print("Enter a URL: ");

String URLString = new Scanner(System.in).next();

try {

java.net.URL url = new java.net.URL(URLString);

int count = 0;

Scanner input = new Scanner(url.openStream());

while (input.hasNext()) {

String line = input.nextLine();

count += line.length();

}

System.out.println("The file size is " + count + " characters");

}

catch (java.net.MalformedURLException ex) {

System.out.println("Invalid URL");

}

catch (java.io.IOException ex) {

System.out.println("IO Errors");

}

}

}