第08天 异常

今日内容介绍

* 异常体系&异常处理
* Throwable常用方法&自定义异常
* 递归

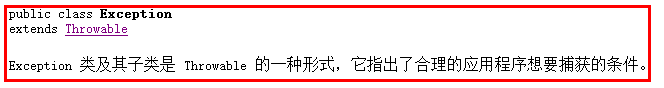
# 异常产生&异常处理

## 异常概述

什么是异常？Java代码在运行时期发生的问题就是异常。

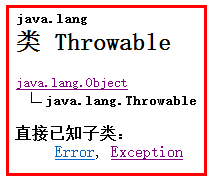
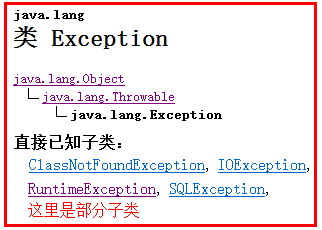
在Java中，把异常信息封装成了一个类。当出现了问题时，就会创建异常类对象并抛出异常相关的信息（如异常出现的位置、原因等）。

在Java中使用Exception类来描述异常。

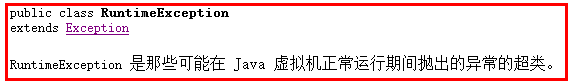


查看API中Exception的描述，Exception 类及其子类是 Throwable 的一种形式，它用来表示java程序中可能会产生的异常，并要求对产生的异常进行合理的异常处理。

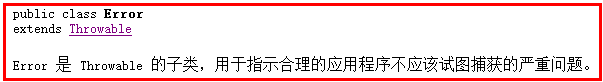
继续观察，我们可以发现Exception有继承关系，它的父类是Throwable。Throwable是Java 语言中所有错误或异常的超类，即祖宗类。



另外，在异常Exception类中，有一个子类要特殊说明一下，RuntimeException子类，RuntimeException及其它的子类只能在Java程序运行过程中出现。



我们再来观察Throwable类，能够发现与异常Exception平级的有一个Error，它是Throwable的子类，它用来表示java程序中可能会产生的严重错误。解决办法只有一个，修改代码避免Error错误的产生。



### 案例代码一:

**package** com.itheima\_01;

**import** java.io.FileWriter;

/\*

\* Exception in thread "main" java.lang.ArithmeticException: / by zero

at com.itheima\_01.ExceptionDemo.main(ExceptionDemo.java:5)

我们在写代码的时候，经常的出现一些小问题，那么为了方便我们处理这些问题，java为我们提供了异常机制

异常包含了错误的类型、原因以及位置

异常：不正常，我们在代码的时候出现的编译或者运行时的错误

异常的体系结构：

Throwable（最顶层）

Error:出现的不能够处理的严重问题

Exception:可以处理的问题

电脑坏了：

系统中毒：重装系统就可以了

主板坏了：买一台新的

\*

\*/

**public** **class** ExceptionDemo {

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

//int a = 10 / 0;

//System.out.println(a);

//FileWriter fw = new FileWriter("a.txt");

}

}

## 异常处理

### JVM默认处理方式

如果出现异常我们没有处理，jvm会帮我们进行处理，他会把异常的类型，原因还有位置显示在命令行并且还终止了程序，异常后面的代码将不在执行

#### 案例代码二:

pa**ckage** com.itheima\_01;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*

\* 异常的处理方式：

\*

\*

\* jvm处理异常的方式：

\* 如果出现异常我们没有处理，jvm会帮我们进行处理，他会把异常的类型，原因还有位置显示在命令行

\* 并且还终止了程序，异常后面的代码将不在执行

\*/

**public** **class** ExceptionDemo2 {

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

System.*out*.println(2/0);

System.*out*.println("hello");

}

}

### try…catch方式处理异常

捕获：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捕获异常格式：

try {

//需要被检测的语句。

}

catch(异常类 变量) { //参数。

//异常的处理语句。

}

finally {

//一定会被执行的语句。

}

**try：**该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。

#### 案例代码三:

**package** com.itheima\_01;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*

\* 异常的处理方式：

\* 捕获处理

\* try...catch语句

\*

\* try {

\* 有可能出现问题的代码;

\* } catch(ArithmeticException ae) {

\* 处理异常;

\* }

\*

\* try...catch的执行顺序：

\* 首先执行try语句

\* 如果发现异常，异常下面的代码不在执行，直接跳入catch语句中，catch语句结束后，整个try...catch结束

\* 如果没有发现异常，try语句执行结束后，try...catch直接结束， 不在执行catch语句

\*

\*

\*

\*

\* jvm处理异常的方式：

\* 如果出现异常我们没有处理，jvm会帮我们进行处理，他会把异常的类型，原因还有位置显示在命令行

\* 并且还终止了程序，异常后面的代码将不在执行

\*/

**public** **class** ExceptionDemo2 {

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

**try** {

System.*out*.println(1);

//System.out.println(2 / 0);

System.*out*.println(2);

} **catch**(ArithmeticException ae) {

System.*out*.println("除数不能为0");

}

System.*out*.println(3);

}

}

### throws方式处理异常

A:throws使用:

权限修饰符 返回值类型 方法名(形参列表) **throws 异常类型1,异常类型2….**{

}

#### 案例代码四:

**package** com.itheima\_01;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*

\* 异常的处理方式：

\* 捕获处理

\* try...catch语句

\*

\* try {

\* 有可能出现问题的代码;

\* } catch(ArithmeticException ae) {

\* 处理异常;

\* }

\*

\* try...catch的执行顺序：

\* 首先执行try语句

\* 如果发现异常，异常下面的代码不在执行，直接跳入catch语句中，catch语句结束后，整个try...catch结束

\* 如果没有发现异常，try语句执行结束后，try...catch直接结束， 不在执行catch语句

\*

\*

\* 抛出去

\* 当我们不想处理异常，或者没有能力处理的时候，我们可以选择抛出异常，谁调用方法谁处理异常

\* 使用关键字throws在方法的声明出抛出异常

\*

\*

\* jvm处理异常的方式：

\* 如果出现异常我们没有处理，jvm会帮我们进行处理，他会把异常的类型，原因还有位置显示在命令行

\* 并且还终止了程序，异常后面的代码将不在执行

\*/

**public** **class** ExceptionDemo2 {

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

//method();

*function*();

}

**public** **static** **void** function() **throws** Exception {

FileWriter fw = **new** FileWriter("a.txt");

}

**private** **static** **void** method() {

**try** {

System.*out*.println(1);

//System.out.println(2 / 0);

System.*out*.println(2);

} **catch**(ArithmeticException ae) {

System.*out*.println("除数不能为0");

}

System.*out*.println(3);

}

}

### 多异常处理

A:对代码进行异常检测，并对检测的异常传递给catch处理。对每种异常信息进行不同的捕获处理。

void show(){ //不用throws

try{

throw new Exception();//产生异常，直接捕获处理

}catch(XxxException e){

//处理方式

}catch(YyyException e){

//处理方式

}catch(ZzzException e){

//处理方式

}

}

注意:这种异常处理方式，要求多个catch中的异常不能相同，并且若catch中的多个异常之间有子父类异常的关系，那么子类异常要求在上面的catch处理，父类异常在下面的catch处理。

#### 案例代码五:

**package** com.itheima\_01;

/\*

\* 如何处理多个异常：

\* 可以使用多个try...catch语句

\* 使用一个try和多个catch

\*

\* 多个catch之间的顺序：

\* 多个catch之间可以有子父类

\* 平级之间没有顺序关系

\* 如果有子父类，父类异常必须放在后面

\*

\*

\*/

**public** **class** ExceptionDemo3 {

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

**try** {

String s = **null**;

System.*out*.println(s.length());

//int[] arr = new int[5];

//System.out.println(arr[8]);

//System.out.println(2 / 0);

}

**catch**(ArrayIndexOutOfBoundsException e) {

System.*out*.println("出现数组越界了");

}

**catch**(NullPointerException e) {

System.*out*.println("出现空指针了");

}

**catch**(Exception e) {

System.*out*.println("出现异常了");

}

/\*try {

} catch(ArrayIndexOutOfBoundsException e) {

System.out.println("出现数组越界了");

}\*/

}

**private** **static** **void** method() {

**try** {

String s = **null**;

System.*out*.println(s.length());

} **catch**(NullPointerException e) {

System.*out*.println("出现空指针了");

}

**try** {

**int**[] arr = **new** **int**[5];

System.*out*.println(arr[8]);

} **catch**(ArrayIndexOutOfBoundsException e) {

System.*out*.println("出现数组越界了");

}

}

}

# Throwable常用方法&自定义异常

## Throwable常用方法

String getMessage() 返回此 throwable 的详细消息字符串

String toString() 返回此 throwable 的简短描述

void printStackTrace() 打印异常的堆栈的跟踪信息

### 案例代码六:

**package** com.itheima\_01;

/\*

\* Throwable的常用方法：

String getMessage()

String toString()

void printStackTrace()

\*

\*/

**public** **class** ExceptionDemo4 {

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

**try** {

System.*out*.println(2 / 0);

} **catch** (ArithmeticException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**private** **static** **void** method() {

**try** {

System.*out*.println(2 / 0);

} **catch**(ArithmeticException e) {

//String getMessage() : 原因

//System.out.println(e.getMessage());

//String toString() 类型和原因

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

//void printStackTrace():类型原因和位置

e.printStackTrace();

}

//System.out.println("hello");

}

}

## finally的概述和应用场景

finally使用格式:

try{

}catch(异常类型 异常变量){

}finally{

//释放资源的代码

}

### 案例代码七:

**package** com.itheima\_01;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*

\* finally:组合try...catch使用，用于释放资源等收尾工作，无论try...catch语句如何执行，finally的代码一定会执行

\*

\* try {

\* 有可能出现问题的代码;

\*

\* } catch(异常对象) {

\* 处理异常;

\* } finally {

\* 释放资源;

\* 清理垃圾;

\* }

\*

\*/

**public** **class** ExceptionDemo5 {

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

//method();

FileWriter fw = **null**;

**try** {

System.*out*.println(2 / 0);

fw = **new** FileWriter("a.txt");

fw.write("hello");

fw.write("world");

//System.out.println(2 / 0);

fw.write("java");

//fw.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **finally** {

//释放资源

**try** {

**if**(fw != **null**) {

fw.close();

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**private** **static** **void** method() {

**try** {

System.*out*.println(2 / 1);

} **catch**(ArithmeticException e) {

System.*out*.println("除数不能为0");

} **finally** {

System.*out*.println("清理垃圾");

}

}

}

## 编译时异常&运行时异常

A: 编译时期异常:是Exception的子类，非RuntimeExcpetion的子类，在编译时期必须处理

B:RuntimeException和他的所有子类异常,都属于运行时期异常。NullPointerException,ArrayIndexOutOfBoundsException等都属于运行时期异常.

运行时期异常的特点:

方法中抛出运行时期异常,方法定义中无需throws声明,调用者也无需处理此异常

运行时期异常一旦发生,需要程序人员修改源代码.

### 案例代码八:

**package** com.itheima\_01;

**import** java.io.FileWriter;

**import** java.io.IOException;

/\*

\* 异常的分类：

运行时期异常:RuntimeException的子类就是运行时期异常，在编译时期可以自由选择处理或者不处理

编译时期异常:是Exception的子类，非RuntimeExcpetion的子类，在编译时期必须处理

\*/

**public** **class** ExceptionDemo6 {

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

//System.out.println(2 / 0);

//String s = null;

//System.out.println(s.length());

**try** {

FileWriter fw = **new** FileWriter("a.txt");

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

## 自定义异常

需求：写一个方法，接受考试成绩，如果考试成绩的范围在0-100之间则属于正常，否则属于异常

### 案例代码九:

当成绩不在0~100范围内,抛出一个运行时异常或者编译时异常,阻止程序继续向下执行

**package** com.itheima\_01;

/\*

\* 需求：写一个方法，接受考试成绩，如果考试成绩的范围在0-100之间则属于正常，否则属于异常

\*

\* throws:处理异常的一种方式，把异常抛出，由调用者来处理

\* throw:制造异常的方式,并且结束方法

\*

\* 注意：如果抛出（throw）的是编译时期异常，必须在方法声明处抛出（throws）

\*

\* 如何自定义一个异常类呢？

\* 非常简单，写一个类去继承Exception或者RuntimeException，然后实现多个构造即可

\*

\* \*/

**public** **class** ExceptionDemo7 {

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

/\*boolean flag = checkScore(-10);

System.out.println(flag);\*/

**try** {

*checkScore*(110);

} **catch** (Exception e) {

//System.out.println(e.getMessage());

e.printStackTrace();

}

//checkScore(110);

}

/\* public static boolean checkScore(int score) {

//判断考试成绩是否符合范围，如果不符合则返回false

if(score < 0 || score > 100) {

return false;

}

//符合

return true;

}\*/

**public** **static** **void** checkScore(**int** score) **throws** Exception {

**if**(score < 0 || score > 100) {

**throw** **new** RuntimeException("考试成绩不符合要求");

//throw new Exception("考试成绩不符合要求");

}

System.*out*.println("考试成绩符合要求");

}

}

### 案例代码十:

我们也可以自定义一个编译时异常或者运行时异常来抛出:

**package** com.itheima\_01;

**public** **class** MyException **extends** /\*RuntimeException\*/ Exception{

**public** MyException() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** MyException(String message) {

**super**(message);

// **TODO** Auto-generated constructor stub

}

/\*public MyException() {

super();

}

public MyException(String s) {

super(s);

}\*/

}

**package** com.itheima\_01;

/\*

\* 需求：写一个方法，接受考试成绩，如果考试成绩的范围在0-100之间则属于正常，否则属于异常

\*

\* throws:处理异常的一种方式，把异常抛出，由调用者来处理

\* throw:制造异常的方式,并且结束方法

\*

\* 注意：如果抛出（throw）的是编译时期异常，必须在方法声明处抛出（throws）

\*

\* 如何自定义一个异常类呢？

\* 非常简单，写一个类去继承Exception或者RuntimeException，然后实现多个构造即可

\*

\* \*/

**public** **class** ExceptionDemo7 {

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

/\*boolean flag = checkScore(-10);

System.out.println(flag);\*/

**try** {

*checkScore*(110);

} **catch** (Exception e) {

//System.out.println(e.getMessage());

e.printStackTrace();

}

//checkScore(110);

}

**public** **static** **void** checkScore(**int** score) **throws** Exception {

**if**(score < 0 || score > 100) {

**throw** **new** MyException("考试成绩不符合要求");

}

System.*out*.println("考试成绩符合要求");

}

}

# 递归

## 递归

递归，指在当前方法内调用自己的这种现象

public void method(){

System.out.println(“递归的演示”);

//在当前方法内调用自己

method();

}

## 递归练习

### 递归求5的阶乘

利用递归求出5!的结果

#### 案例代码十一:

**package** com.itheima\_01;

/\*

\* 需求：求5的阶乘

\* 5! = 5 \* 4 \* 3 \* 2 \* 1; //120

\* 5! = 5 \* 4!; //120

\* 4! = 4 \* 3!; //24

\* 3! = 3 \* 2!; //6

\* 2! = 2 \* 1!; //2

\* 1! = 1; //1

n! = n \* (n - 1)!

递归：把大问题拆成很多小问题，然后再把小问题拆成更多的小问题，

当我们把更多小问题解决了，小问题也解决了

随着小问题的解决，大问题也随之解决了

在方法本身不断的调用方法自己

递归注意事项：

递归一定要有出口，内存溢出

递归次数不宜过多，内存溢出

public void show(int n) {//5

//出口

if(n == 0) {

return;

}

show(n - 1);

}

从前有座山，山里有个庙，庙里有个老和尚，老和尚在给小和尚讲故事

\* 从前有座山，山里有个庙，庙里有个老和尚，老和尚在给小和尚讲故事

\* 从前有座山，山里有个庙，庙里有个老和尚，老和尚在给小和尚讲故事

\* 从前有座山，山里有个庙，庙里有个老和尚，老和尚在给小和尚讲故事

\* 从前有座山，山里有个庙，庙里有个老和尚，老和尚在给小和尚讲故事

\* ...

\*

\* 学习编程...找工作...赚钱...娶媳妇...生娃娃

学习编程...找工作...赚钱...娶媳妇...生娃娃

学习编程...找工作...赚钱...娶媳妇...生娃娃

学习编程...找工作...赚钱...娶媳妇...生娃娃

学习编程...找工作...赚钱...娶媳妇...生娃娃

...

\*

\*/

**public** **class** RecurrenceDemo {

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

**int** result = *jC*(5);

System.*out*.println(result);//120

}

//求一个数的阶乘

**public** **static** **int** jC(**int** n) {

//必须要有出口

**if**(n == 1) {

**return** 1;

}

**else** {

**return** n \* *jC*(n - 1);

}

}

}

### 斐波纳挈数列

有一对兔子，从出生后第3个月起每个月都生一对兔子，

小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，

问第二十个月的兔子对数为多少？

#### 案例代码十二:

**package** com.itheima\_01;

/\*

\* 古典问题：有一对兔子，从出生后第3个月起每个月都生一对兔子，

\* 小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，

\* 问第二十个月的兔子对数为多少？

\*

\*

\* 1

\* 1

\* 2

\* 3

\* 5

\* 8

\* 13

\*

\* 规律：除了第一个月和第二月以外，其余每个月都是前两个月之和

\* 斐波那契列数

\*

\*/

**public** **class** RecurrenceDemo2 {

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

**int** result = *method*(20);//6765

System.*out*.println(result);

}

**public** **static** **int** method(**int** n) {

//如果是第一个月，只有一对兔子

**if**(n == 1) {

**return** 1;

}

//如果是第二个月，也只有一对兔子

**else** **if**(n == 2) {

**return** 1;

}

**else** {

//如果不是第一个月和第二个月，则兔子的数量是前两个月之和

**return** *method*(n - 1) + *method*(n - 2);

}

}

}