# 异常

## 异常的概述和继承体系

### 1.1.1 异常详解

异常的概念:

异常：就是程序出现了不正常的情况。

例如：ArithmeticException:当出现异常的运算条件时，抛出此异常。例如，一个整数“除以零”时，抛出此类的一个实例。

常见的异常有:

IndexOutOfBoundsException、NullPointerException、StringIndexOutOfBoundsException

异常的体系的介绍：

Throwable 类是 Java 语言中所有错误或异常的超类。

Error 是 Throwable 的子类，用于指示合理的应用程序不应该试图捕获的严重问题。

也就是说针对程序发生了Error的情况，Java程序本身是无能为力的，比如说：硬件层面的问题，内存不足等。

所以，针对Error的问题我们不处理。

Exception 类及其子类是 Throwable 的一种形式，它指出了合理的应用程序想要捕获的条件。

也就是说针对程序发生了Exception的情况，是我们需要处理的问题。

Exception的分类：

运行期的异常（RunTimeException）：在编译期是不处理的，在程序运行时候出现了问题，需要我们回来修改代码。

编译期的异常（非RunTimeException）：在编译期就必须处理，否则程序不能通过编译，就更不能正常的执行了。

### 1.1.2 案例代码

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

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

*method*();

}

**public** **static** **void** method() {

**int** a = 10;

**int** b = 5;

//b = 0;

System.***out***.println(a/b);

}

}

## JVM针对异常的默认处理方式

### JVM默认如何处理异常？

处理方案：

A:把异常的名称，异常的原因，异常出现的位置等信息在控制台输出

B:让程序停止执行

### 案例代码

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

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

System.***out***.println("程序开始执行");

*method*();

System.***out***.println("程序结束执行");

}

**public** **static** **void** method() {

**int** a = 10;

**int** b = 0;

System.***out***.println(a/b);

}

}

控制台：

java.lang.ArithmeticException:异常的类名，包括包名

/ by zero:异常的原因，被0除

at com.itheima.ExceptionDemo2.method(ExceptionDemo2.java:17):异常的位置

## 1.3异常处理方案try\_catch

### 1.3.1 try..catch的格式和执行流程为

格式：

try {

可能出现异常的代码;

}catch(异常类名 变量名) {

异常的处理代码;

}

执行流程：

程序从try开始执行，执行到哪里出现了问题，就会跳转到catch里面执行。

执行完毕后，程序还能继续往下执行。

### 1.3.2 捕获异常后, 如何处理?

public void printStackTrace():把异常的错误信息输出在了控制台。

在实际开发中，我们遇见了异常，会给出一个页面进行提示，而我们目前做不了，

所以，就用异常对象调用printStackTrace()就可以了。

### 1.3.3 try..catch处理方式&JVM的默认处理方式有什么不同

首先要明确try..cathc处理方式的特点, 产生了问题, 是自己将问题处理掉, 不影响后续代码的运行.

JVM默认处理方式是将程序终止, 并将异常信息打印在控制台.

但这种方式很显然用户体验度不佳, 所以这时候就可以考虑使用try..cathc将问题捕获并处理掉.

这样就不会影响程序的继续执行了

## 1.4编译时异常和运行时异常的区别

## 1.4.1 异常的两大分类指的是?

编译时异常

运行时异常

### 1.4.2 两种异常的区别是?

Java中的异常被分为两大类：编译时异常和运行时异常。

所有的RuntimeException类及其子类的实例被称为运行时异常，其他的异常都是编译时异常

编译时异常：Java程序必须显示处理，否则程序就会发生错误的一个提示，无法通过编译

运行时异常：Java程序无需显示处理，也可以和编译时异常一样处理

### 1.4.3 案例代码

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

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

System.***out***.println("程序开始执行");

//method();

*method2*();

System.***out***.println("程序结束执行");

}

//编译时异常ParseException

**public** **static** **void** method() {

**try**{

//String s = "2088-08-08";

String s = "abcd";

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");

Date d = sdf.parse(s);

System.***out***.println(d);

}**catch**(ParseException e) {

e.printStackTrace();

}

}

//运行时异常

**public** **static** **void** method2() {

**try**{

**int** a = 10;

**int** b = 0;

System.***out***.println(a/b);

}**catch**(ArithmeticException e) {

e.printStackTrace();

}

}

}

## 1.5异常处理方案throws

### 1.5.1 为什么要有throws处理方式?

我们通过try...catch可以对异常进行处理了，但是并不是所有的时候我们都有权限进行异常的处理。

也就是说，有些时候我们处理不了，但是，这个时候异常时存在的，不处理也不行，怎么办?

这个时候，Java就提供了throws的处理方案。

### 1.5.2 throws使用的格式&注意事项?

格式：

throws 异常类名

注意：这个格式必须跟在方法的括号的后面

注意：

编译时异常时必须要进行处理的，两种处理方案：try...catch...或者throws

如果你采用了throws这种方案，将来谁调用，还得进行处理。

运行时异常可以不用处理，出现问题后我们需要回来修改代码。

### 1.5.3 案例代码

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

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

System.***out***.println("程序开始执行");

**try** {

*method*();

} **catch** (ParseException e) {

e.printStackTrace();

}

*method2*();

System.***out***.println("程序结束执行");

}

//编译时异常

**public** **static** **void** method() **throws** ParseException {

String s = "2088-08-08";

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd");

Date d = sdf.parse(s);

System.***out***.println(d);

}

//运行时异常

**public** **static** **void** method2() **throws** ArithmeticException {

**int** a = 10;

**int** b = 0;

System.***out***.println(a/b);

}

}

# File类

## 2.1 File类的概述和构造方法

### 2.1.1 File类的概述

File:文件和目录路径名的抽象表示形式

也就是说文件和目录是可以通过File封装成对象的

目录：其实就是文件夹

### 2.1.2 File类的构造方法

file构造方法：

File(String pathname):通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。

File(String parent, String child):根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

File(File parent, String child):根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实

### 2.2.3 案例代码

**public** **class** FileDemo {

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

//File(String pathname)

File f1 = **new** File("d:\\aa\\b.txt");

//File(String parent, String child)

File f2 = **new** File("d:\\aa","b.txt");

//File(File parent, String child)

File f3 = **new** File("d:\\aa");

File f4 = **new** File(f3,"b.txt");

//上面的f1,f2,f4其实做的是同样的事情，就是把d:\\aa\\b.txt转换为了一个File对象

}

}

## 2.2 File类的创建功能

### 2.2.1 方法摘要

创建功能

public boolean createNewFile():创建文件

如果文件不存在，创建文件并返回true

如果文件存在，创建文件失败并返回false

public boolean mkdir():创建目录

如果目录不存在，创建目录并返回true

如果目录存在，创建目录失败并返回false

public boolean mkdirs():创建多级目录

### 2.2.2 案例代码

**public** **class** FileDemo {

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

//需求1：我要在d盘目录下创建一个文件a.txt

File f1 = **new** File("d:\\a.txt");

System.***out***.println("createNewFile:"+f1.createNewFile());

//需求2：我要在d盘目录下创建一个目录bb

File f2 = **new** File("d:\\bb");

System.***out***.println("mkdir:"+f2.mkdir());

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

//需求3：我要在d盘目录下创建一个多级目录cc\\dd

// File f3 = new File("d:\\cc\\dd");

// System.out.println("mkdir:"+f3.mkdir());

//一步一步的实现

// File f3 = new File("d:\\cc");

// File f4 = new File("d:\\cc\\dd");

// System.out.println("mkdir:"+f3.mkdir());

// System.out.println("mkdir:"+f4.mkdir());

//一步操作

File f3 = **new** File("d:\\cc\\dd");

System.***out***.println("mkdirs:"+f3.mkdirs());

//需求4：我要在d盘目录下创建一个文件ee\\f.txt

File f4 = **new** File("d:\\ee");

File f5 = **new** File("d:\\ee\\f.txt");

System.***out***.println("mkdir:"+f4.mkdir());

System.***out***.println("createNewFile:"+f5.createNewFile());

}

}

## 2.3 File类的删除功能

### 2.3.1 相对路径&绝对路径

路径的问题：

绝对路径：是以盘符开始的路径。d:\\aa\\b.txt

相对路径：不以盘符开始。相对于当前的项目而言，在项目的目录下。如何显示出来呢？刷新项目就可以了。

### 2.3.2 方法摘要&注意事项

删除功能

public boolean delete():删除文件和目录

注意：

如果一个目录中有内容(目录，文件)，就不能直接删除。

应该先删除目录中的内容，最后才能删除目录。

### 2.3.4 案例代码

**public** **class** FileDemo {

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

// //需求1：我要创建一个文件a.txt

// File f1 = new File("a.txt");

// System.out.println("createNewFile:"+f1.createNewFile());

//

// //需求2：我要创建一个目录bb

// File f2 = new File("bb");

// System.out.println("mkdir:"+f2.mkdir());

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

//

// //需求3：我要创建一个文件cc\\d.txt

// File f3 = new File("cc");

// File f4 = new File("cc\\d.txt");

// System.out.println("mkdir:"+f3.mkdir());

// System.out.println("createNewFile:"+f4.createNewFile());

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

// public boolean delete():删除文件和目录

// 需求1：我要删除a.txt这个文件

File f1 = **new** File("a.txt");

System.***out***.println("delete:" + f1.delete());

//需求2：我要删除bb这个目录

File f2 = **new** File("bb");

System.***out***.println("delete:"+f2.delete());

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

//需求3：我要删除cc这个目录

File f3 = **new** File("cc");

System.***out***.println("delete:"+f3.delete());

//思路：先删除d.txt这个文件，再删除cc这个目录

}

}

## 2.4 File类的判断和获取功能

### 2.4.1 方法摘要(以下方法是File类当中必须要掌握的)

判断功能

public boolean isDirectory():判断是否是目录

public boolean isFile():判断是否是文件

public boolean exists():判断是否存在

获取功能

public String getAbsolutePath():获取绝对路径

public String getPath():获取相对路径

public String getName():获取名称

### 2.4.2 案例代码

**public** **class** FileDemo {

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

//创建File对象

File f = **new** File("aaa\\bbb.txt");

//判断功能

System.***out***.println("isDirectory:"+f.isDirectory());

System.***out***.println("isFile:"+f.isFile());

System.***out***.println("exists:"+f.exists());

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

//获取功能

System.***out***.println("getAbsolutePath:"+f.getAbsolutePath());

System.***out***.println("getPath:"+f.getPath());

System.***out***.println("getName:"+f.getName());

}

}

# IO(字节流)

## 3.1 IO流的概述和分类

### 3.1.1 什么是IO流,其作用为?

IO可以拆开来理解

I ---- Input -> 输入 🡪 读取

O ---- Output -> 输出 🡪 写出

读取和写出都是针对数据而言的, 所以, IO流就是用来处理设备之间的数据传输

常见应用:

文件复制

文件上传

文件下载

### 3.1.2 IO流的分类

按照类型分:

字节流

字符流 (字符流数据通过Windows自带的记事本软件打开是可以读懂里面内容的)

按照流向分:

输入流 : 用来读取数据的:

输出流 : 用来写出数据的

## 3.2 FileOutputStream写数据

### 3.2.1 字节流&字符流的抽象父类

字节流：

InputStream 字节输入流

OutputStream 字节输出流

字符流：

Reader 字符输入流

Writer 字符输出流

### 3.2.2 字节流写出数据

字节流写数据

OutputStream:此抽象类是表示输出字节流的所有类的超类

FileOutputStream:文件输出流是用于将数据写入 File

构造方法：

FileOutputStream(String name):创建一个向具有指定名称的文件中写入数据的输出文件流。

字节流写数据的步骤：

A:创建字节输出流对象

B:调用写数据的方法

C:释放资源

### 3.2.3 案例代码

**public** **class** FileOutputStreamDemo {

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

//创建字节输出流对象

FileOutputStream fos = **new** FileOutputStream("a.txt");

/\*

\* 创建字节输出流对象做了这样的三件事情：

\* A:调用系统功能创建了文件

\* B:创建字节输出流对象

\* C:让fos这个对象指向a.txt这个文件

\*/

//write(int b)

fos.write(65);

fos.write(66);

//最后我们还要做一个事情

//close() 关闭此文件输出流并释放与此流有关的所有系统资源。

fos.close();

}

}

## 3.3 FileOutputStream写数据的三种方式.

### 3.3.1 方法摘要

写出数据的三个方法:

\* public void write(int b):一次写一个字节

\* public void write(byte[] b):一次写一个字节数组

\* public void write(byte[] b,int off,int len):一次写一个字节数组的一部分

String类中的方法

\* byte[] getBytes() 将字符串转换为字节数组

### 3.3.2 案例代码

**public** **class** FileOutputStreamDemo2 {

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

//创建字节输出流对象

//FileOutputStream(String name)

FileOutputStream fos = **new** FileOutputStream("b.txt");

//new File(name)

// FileOutputStream fos = new FileOutputStream(new File("b.txt"));

//FileOutputStream(File file)

// File file = new File("b.txt");

// FileOutputStream fos = new FileOutputStream(file);

// FileOutputStream fos = new FileOutputStream(new File("b.txt"));

//public void write(int b):一次写一个字节

// fos.write(65);

//public void write(byte[] b):一次写一个字节数组

// byte[] bys = {65,66,67,68,69};

// fos.write(bys);

//需求：我如果是一个字符串的数据，能写吗?

//String -- byte[]

//String类中有一个方法：public byte[] getBytes()

// byte[] bys = "ABCDE".getBytes();

// fos.write(bys);

// fos.write("ABCDE".getBytes());

//public void write(byte[] b,int off,int len):一次写一个字节数组的一部分

fos.write("ABCDE".getBytes(),0,3);

//释放资源

fos.close();

}

}

## 3.4 FileOutputStream如何实现换行和追加写数据

### 3.4.1 如何实现数据的换行

不同的操作系统，针对换行的符号识别是不一样的。

windows:\r\n

linux:\n

mac:\r

### 3.4.2 如何实现数据的追加写入?

用构造方法带第二个参数是true的情况即可

### 3.4.3 案例代码

**public** **class** FileOutputStreamDemo3 {

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

//创建字节输出流对象

//FileOutputStream fos = new FileOutputStream("c.txt");

//FileOutputStream(String name, boolean append)

//如果第二个参数为 true，则将字节写入文件末尾处，而不是写入文件开始处

FileOutputStream fos = **new** FileOutputStream("c.txt",**true**);

//调用写数据的方法

**for**(**int** x=0; x<10; x++) {

fos.write("hello".getBytes());

//加入换行符号

fos.write("\r\n".getBytes());

}

//释放资源

fos.close();

}

}

## 3.5 FileOutputStream写数据加入异常处理

### 3.5.1 try..catch.finally

格式:

try{

可能发生问题的代码

}catch(){

处理异常代码

}finally{

一定会被执行的代码. // 通常用于释放资源, 做善后的动作

}

### 3.5.2 案例代码

**public** **class** FileOutputStreamDemo4 {

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

FileOutputStream fos = **null**;

**try**{

//FileOutputStream fos = new FileOutputStream("d.txt");

// fos = new FileOutputStream("z:\\d.txt");

fos = **new** FileOutputStream("d.txt");

fos.write("hello".getBytes());

}**catch**(IOException e) {

e.printStackTrace();

}**finally** {

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

//释放资源

**try** {

fos.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

}

## 3.6 FileInputStream读数据方式1一次读取一个字节

### 3.6.1 字节流读取数据的三个步骤

字节流读数据的步骤：

A:创建字节输入流对象

B:调用读数据的方法

C:释放资源

### 3.6.2 案例代码

**public** **class** FileInputStreamDemo {

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

//创建字节输入流对象

FileInputStream fis = **new** FileInputStream("a.txt");

**int** by;

// 用by不断的记录读取到的每一个数据

**while**((by=fis.read())!=-1) {

System.***out***.print((**char**)by);

}

//释放资源

fis.close();

}

}

## 3.7 FileInputStream读数据方式2一次读取一个字节数组

### 3.7.1 方法摘要

public int read(byte[] b):

从此输入流中将最多 b.length 个字节的数据读入一个 byte 数组中

返回值是读入缓冲区的字节总数，也就是实际的读取个数

如果因为已经到达文件末尾而没有更多的数据，则返回 -1。

### 3.7.2 案例代码

**public** **class** FileInputStreamDemo2 {

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

//创建字节输入流对象

FileInputStream fis = **new** FileInputStream("b.txt");

**byte**[] bys = **new** **byte**[1024]; //1024或者1024的整数倍

**int** len;

//将数据读取到数组中, 并用len记录读取到的有效字节个数

**while**((len=fis.read(bys))!=-1) {

System.***out***.print(**new** String(bys,0,len));

}

//释放资源

fis.close();

}

}

## 3.8 字节流练习之复制文本文件

需求： 拷贝文本文件

分析：

第一步: 创建输入输出流对象关联数据源和数据目的

第二步: 定义字节数组,为了提高效率

第三步: 将数据通过while循环不断读取到字节数组中

第四步: 将数据从字节数组中取出并写出

第五步: 释放资源

### 3.8.1 案例代码

**public** **class** CopyTxtTest {

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

//封装数据源

FileInputStream fis = **new** FileInputStream("d:\\窗里窗外.txt");

//封装目的地

FileOutputStream fos = **new** FileOutputStream("林青霞.txt");

//读写数据

//方式1:一次读取一个字节

// int by;

// while((by=fis.read())!=-1) {

// fos.write(by);

// }

//方式2:一次读取一个字节数组

**byte**[] bys = **new** **byte**[1024];

**int** len;

**while**((len=fis.read(bys))!=-1) {

fos.write(bys, 0, len);

}

//释放资源

fos.close();

fis.close();

}

}

## 3.9 字节流练习之复制图片

思路:

同理3.8案例

### 3.9.1案例代码

**public** **class** CopyJpgTest {

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

// 封装数据源

FileInputStream fis = **new** FileInputStream("d:\\mn.jpg");

// 封装目的地

FileOutputStream fos = **new** FileOutputStream("mn.jpg");

// 读写数据

// 方式1：一次读取一个字节，一次写一个字节(自己练习)

// 方式2：一次读取一个字节数组，一次写一个字节数组的一部分

**byte**[] bys = **new** **byte**[1024];

**int** len;

**while** ((len = fis.read(bys)) != -1) {

fos.write(bys, 0, len);

}

// 释放资源

fos.close();

fis.close();

}

}