# 一、异常

程序中产生的意外情况叫做异常。

本质上来说,异常

是一种由被调用者向调用者传递信息的方式

是调用者根据产生的意外情况控制程序流转的机制

throws：声明抛出异常

throw：抛出异常

try：检查代码，试图捕获

catch：捕获指定异常

finally：无论是否有异常产生都一定会执行

Throwable

|--Error 错误 严重的问题 不希望调用者处理时抛出 不需要声明抛出，调用者可以处理也可以不处理，如果不处理就自动向上抛出最终抛给虚拟机，打印错误停止当前线程。

|--Exception 异常 程序产生的意外情况

|--普通Exception 抛出必须声明 调用者必须处理 要么接着向上抛 要么当场处理

|--RuntimeException 抛出不需要声明 调用者可以处理也可以不处理，如果不处理默认接着往上抛，如果一直都没人处理最终抛出给虚拟机停止当前线程。

# !!!二、程序的错误调试

异常堆栈

异常产生的线程

异常的名称推断异常产生的原因

异常的描述信息进一步推测原因

异常的堆栈信息

断点调试

# 三、IO流 -- Input Output流 -- 输入输出数据流

## 1.概述

IO流就是输入/输出程序的数据流，通过IO流技术可以实现从程序中读取外部数据或者将数据从程序中输出到外部设备中。

java.io

输入流 输出流

字节流 InputStream OutputStream

字符流 Reader Writer

\*\*四大基本流都是抽象类，不能直接使用，在他们的基础上，派生出了很多具有具体功能的子孙流。

## 2.字符流

实验1：输出字符数据到文件中 -- Writer -- OutputStreamWriter -- FileWriter

流在读取文件时，如果文件不存在会自动生成一个新的文件。如果已经存在默认情况下会覆盖该文件，如果不想覆盖，需要在构造流时明确的指定。

FileWriter(String path)

FileWriter(String path,boolean append)

writer(String str)

flush() //刷新缓冲区，可以多次写入多次刷新，刷新后可以再次写入数据

close() //关闭流，在关闭流之前，此方法首先会刷新，防止有数据死在缓冲区中 ，流关闭后就不能在写入数据了

实验：从文件中读取数据1 -- Reader -- InputStreamReader -- FileReader

FileReader(String fileName)

int read() 读取单个字符。 由于读取过程中需要标记结束位置，所以返回值并不是char而是int，如果返回值不是-1则可以强转为char获取字符数据，如果是-1表示文件读到了结尾，文件结束。

实验：从文件中读取数据2

int read(char []) //使用char类型的数组作为缓冲，减少读取的次数从而提高程序的性能，在读取的内容比较多时，推荐使用这种方法。

异常的处理：

FileReader reader = null;

FileWriter writer = null;

try{

}catch(IOException e){

e.printStackTrace();

throw new RuntimeException("拷贝文件出错了！");

}finally{

if(reader != null){

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

} finally{

reader = null;

}

}

if(writer != null){

try {

writer.close();

} catch (IOException e) {

e.printStackTrace();

} finally{

writer = null;

}

}

}

实验：文本文件的复制

## 3.BufferedReader、BufferedWriter

自带缓冲区的Reader和Writer，可以提高性能，还额外提供了按行读写的方法

BufferedReader

BufferedReader(Reader in)

BufferedReader(Reader in, int sz)

readeline()

BufferedWriter

BufferedWriter(Writer out)

BufferedWriter(Writer out, int sz)

void newLine();// 写入一个行分隔符

## 4.LineNumberReader

BufferedReader

|--LineNumberReader

int getLineNumber()

setLineNumber(int num)

# 四、装饰设计模式

改造一个对象中不喜欢的方法，可以彻底改造不喜欢的方法或在方法执行之前或之后做一些额外的操作。

案例：狗

案例：MyBufferedReader

改造不喜欢的方法的三种方式：

继承

不能改造已有对象身上的方法，只能是继承过后再重新创建继承类的对象使用

装饰

可以改造已有对象身上的方法

动态代理

//TODO 以后再讲

# 五、字节流

OutputStream

InputStream

案例1：使用字节流写出字符到文件中

案例2：使用字节流读取文件中的字符数据

案例3：使用字节流复制图片

BufferedInputStream

BufferedOutputStream

int available() //获知流中的可能的字节数

装饰设计模式，在被装饰的类的基础上增加了缓冲功能 另外还额外的提供了一些其他功能。

# 六、系统流

System.in

\*\*BufferedReader(new InputStreamReader(System.in))

\*\*Scanner(System.in)

System.out

System.err

# 七、转换流

转换流是字节流通向字符流的桥梁，可以将字节流转换为字符流，原理其实就是在字节流的基础上增加了编解码的操作。

InputStreamReader

OutputStreamWriter

FileReader、FileWriter其实底层都是字节流，只不过额外的配置一个编码器、解码器，底层让然用字节读数据通过解码器转换成字符，或通过编码器将字符转换为字节再输出。

而FileReader和FileWriter使用的都是系统默认的编码，如果使用它读取写出非系统码的文件，可能会产生乱码的问题。

此时可以通过转换流自己构建指定编码的字符输入流或输出流。

# 八、总结

判断流操作的是字符还是字节 -- 确定出使用字节流还是字符流

判断是读操作还是写操作 -- 确定出使用四大基本流中的哪一个体系

判断操作的目标是什么 -- 从基本流的派生体系统选则对应功能的流

判断是否需要转换流 -- 需要字符流但是只有字节流，或者虽然有字符流但是编码不正确

判断是否需要提高性能 -- BufferedXXXX 包装提高性能

## 场景1：读取一个UTF-8的字符文件

读操作的字符流 -- Reader

文件 -- FileReader -- InputStreamReader 构造一个UTF-8的Reader

性能 -- BufferedReader

## 场景2:想从控制台中输入字符数据到程序中

读操作字符流--Reader

系统控制台 -- System.in

字节流转换为字符流 -- InputStreamReader

性能 -- BufferedReader

# 九、File

代表文件或文件夹的类。

提供了获取文件/文件夹的基本信息和基本操作的方法。

File(String pathname)

File(String parent, String child)

File(File parent, String child) //只是在内存中创建对象代表硬盘中的文件或文件夹，并不会真的去检查该文件或文件夹是否存在，也不会创建该文件或文件夹。

separator

pathSeparator

## 创建：

createNewFile();//只在文件不存在时创建，创建成功返回true，如果文件已经存在不创建，返回false.创建文件时只能在已有的路径下创建。此方法只能创建文件不能创建文件夹。

mkdir()//创建文件夹，此方法只能创建一级目录，如果有多级目录无法创建。

mkdirs()//创建文件夹，此方法可以创建多级目录，会将路径中须有的所有目录都依次创建出来。

删除：

delete()//既可以删除文件也可以删除文件夹，如果要删除的是一个文件夹，则该文件夹必须位空的才能删除

## 判断：

boolean canRead() //判断该文件是否可读

boolean canWrite() //判断文件是否可写

boolean exists() //判断文件或文件夹是否存在

boolean isFile() //判断是否是一个文件，如果File表示的是一个文件且文件存在则返回true，如果File表示的文件不存在或者是一个文件夹则返回false

boolean isDirectory()//判断是否是一个文件夹，只有这个File代表的是一个文件夹且文件夹存在返回true，否则返回false

boolean isHidden() //判断是否是一个隐藏文件

## 获取信息：

String getName();//获取名称

String getAbsolutePath() //获取文件的绝对路径

String getPath() //获取路径

long length() //获取文件的大小，如果文件不存在返回0L,如果是个目录，则返回值不确定

long lastModified() //返回文件的最后修改时间

String getParent() //获取父目录的字符串形式

File getParentFile() //获取父目录的File对象形式

## 设置方法：

boolean setLastModified(long time) //设置最后修改时间

boolean setReadable(boolean readable) //设置文件是否可读

boolean setWritable(boolean writable) //设置文件是否可写

renameTo(File f);//重命名方法，除了可以修改文件名字，还可以实现类似剪切的操作

获取文件：

static File[] listRoots() //获取当前系统的所有盘符

String[] list() //获取目录中的所有内容（包括文件和文件夹）的名字组成的字符串数组

String[] list(FilenameFilter filter)

File[] listFiles() //获取目录中的所有内容（包括文件和文件夹）的File对象的数组

File[] listFiles(FileFilter filter)

案例：文件过滤器

案例：打印文件夹的目录树结构

案例：递归：开发一个程序，能够计算从1到指定的n的加和

1+2+3

1+2+3+4+5

案例：递归删除非空文件夹

\*\*递归其实就是方法自己调用自己

\*\*递归应该注意：递归必须有结束条件 递归的深度不能太深，太深可能会产生java.lang.StackOverflowError问题，原因是堆栈内存溢出了。

==========================================

# 路径专题：

相对路径：不以\开头的路径称为相对路径，相对路径是基于当前路径计算一个新的路径

相对路径也可以以./开头，也可以省略./直接书写

计算规则是：

用相对路径替换掉当前路径的最后一级.

也可以使用../表示替换上一级，一个../表示一级目录，可以写多个

绝对路径：以\开头的路径称为绝对路径，绝对路径在一个基础路径（在文件系统中通常指的是盘符根目录）上进行拼接得到一个新的路径

以盘符开始的绝对路径（硬盘路径）: 从盘符开始指定，说是哪就是哪

==========================================

# 十、IO中的其他重要的流

## 1.Properties

来自集合体系

HashTable

|

java.util.Properties -- 可以从文件中加载，也可以写出到文件中 -- 经常用做程序的配置文件

Properties信息也可以保存在.properties格式的文件中。properties文件中"#"之后的内容是被注释掉了。

Properties中不允许存储中文数据，如果需要保存非ISO8859-1的数据，需要保存成\uxxxx格式

Properties()

Object setProperty(String key, String value)

String getProperty(String key) 用指定的键在此属性列表中搜索属性。

String getProperty(String key, String defaultValue) 用指定的键在属性列表中搜索属性。

load(InputStream in)

load(Reader reader)

store(OutputStream String)

store(Writer String)

## 2.打印流

Writer

|

PrintWriter

通过String路径

通过File对象

通过包装其他Writer

通过包装其他OutputStream

在调用 println、printf 或 format 方法时底层会自动刷新不需要我们刷新

写数据的方法没有异常抛出

提供了对各种类型支持的print方法和println方法，底层就是将传入的内容输出到流中

OutputStream

|

FilterOutputStream

|

PrintStream

在调用 println、printf 或 format 方法时底层会自动刷新不需要我们刷新

写数据的方法没有异常抛出

提供了对各种类型支持的print方法和println方法，底层就是将传入的内容输出到流中

## 3.SequenceInputStream

序列流，可以将多个字节输入流包装成要给流，依次从中获取数据。

案例：文件的切割和合并

## 4.RandomAccessFile

随机读写文件的流。既可以读取也可以写入。在此类看来一个被操作的文件就像一个非常大的字节数组一样，并在该字节数组内部维系了一个指针，这个指针可以被设置到指定位置，从而从指定位置开始进行读写的操作。

RandomAccessFile提供了读写文件指定位置部分字节信息的能力。

RandomAccessFile(File file, String mode)

RandomAccessFile(String name, String mode)

如果文件不存在就创建该文件。如果文件已经存在就直接使用而不会覆盖。

int skipBytes(int n) //让指针跳过指定字节指向后续位置

void seek(long pos) //移动指针到指定位置，其中的参数表示从文件的开始位置计算的字节偏移量。

案例：利用RandomAccessFile修改文件中指定位置的数据

# 5.对象持久化流

ObjectOutputStream(OutputStream out)

ObjectInputStream(InputStream in)

利用这两个流可以将对象持久化到设备中，可以从设备总将之前持久化的对象再读取回来。

想要被序列化的对象的类必须实现Serializable接口。

静态成员变量属于类，不能被序列化

某个想要被序列化的对象类中的某个属性如果没有实现Serializable接口，则在序列化过程中，该属性不能被序列化。

如果想要手动指定某个属性不被序列化，可以使用java中的transient关键字

序列化和反序列化时，会根据serialVersionUID来判断序列化之前的类和要反序列化的列是否是同一个。

serialVersionUID默认自动取值，每次编译值都不同。

我们可以通过在类中指定private static final long serialVersionUID = 999888777l;手动指定一个serialVerwsionUID，实现即使类发生轻微变化仍然可以反序列化、实现远程对象传递时的序列化反序列化问题。

# 6.基本数据流

DataInputStream DateOutputStream

-- 操作基本数据类型的流

ByteArrayInputStream ByteArrayOutputStream

-- 数据来源/目的地 是内存中的字节数组的流

ByteArrayOutputStream()

ByteArrayOutputStream(int size)

byte[] toByteArray() //将ByteArrayOutputStream中的保存的字节转换位字节数组返回。

ByteArrayInputStream(byte[] buf)

ByteArrayInputStream(byte[] buf, int offset, int length)

CharArrayReader CharArrayWriter

-- 数据来源/目的地 是内存中的字符数组的流

StringReader StringWrtier

-- 数据来源/目的地 是字符串

=======================================================================================================================================

# 总结：

IO概念：输入输出数据流 -- 程序输入数据或输出数据的技术

根据方向分为：输入流 和 输出流

根据操作的数据单位： 字节流 和 字符流

四大基本流： Reader Writer InputStream OutputStream -- 抽象类，不能直接使用，各自维系了一套继承体系，在其继承体系中提供了很多具体功能的流

字符流：

读数据

写数据

刷新流

关闭流

异常的处理

使用缓冲数组提升效率

FileReader FileWriter BufferedReader BufferedWriter LineNumberReader

字节流：

读数据

写数据

刷新流

关闭流

异常的处理

使用缓冲数组提升效率

FileOutputStream FileInputStream BufferedInputStream BufferedOutputStream

转化流：

字节流和字符流的桥梁，由于字符在计算机内部也是字节，所以可以将字节流通过增加编码器解码器转换为字符流。

其实FileReader、FileWriter底层也是字节流，只是在字节流的基础上增加了编码解码的机制。并且FileReader、FileWriter使用的编码器解码器默认的编码集是系统默认编码且无法修改。

我们可以通过转换流自己将字节流包装为字符流，在包装的过程中可以手动指定编码，从而可以包装出指定编码的字符流。

其他重要的流：

打印流：

PrintWriter

PrintStream

在调用 println、printf 或 format 方法时底层会自动刷新不需要我们刷新

写数据的方法没有异常抛出

提供了对各种类型支持的print方法和println方法，底层就是将传入的内容输出到流中

系统流： -- 转换流、Scanner

System.in

System.out

System.err

Properties:

不是流而是集合体系中的一员。

此集合也可以保存键值信息，特殊的是可以从流中加载或写出到流中，也可以以.properties格式的文件的形式的存在。

常常被用作配置文件来使用。

RandomAccdessFile

随机读写文件的流。可以有不同的模式，既可以写也可以读。将文件视作一个大的字节数组，并提供指针，可以通过操作指针在指定位置进行读写操作。

ObjectInputStream ObjectOutputStream -- Serializable -- transient -- serialVersionUID

可以实现对象的序列化、反序列化

ByteArrayInputStream ByteArrayOutputStream

CharArrayReader CharArrayWriter

StringReader StringWrtier

SequenceInputStream

序列流，可以将多个流进行合并

装饰设计模式：

在原有对象的基础上改变、增加 、增强 原有的方法。

写一个类实现和被装饰者相同的接口（继承） 提供构造方法将被装饰者传入保存在类的内部 对于不想改变的方法直接调用原有方法，对于想改造的方法从新写，对于想增加的方法增加，对于想改造的方法在调用原有方法之前或之后写额外的代码。

# File

代表文件或文件夹

静态成员常量：

separator

pathSeparator

提供了对文件进行 创建、删除、判断、获取信息、设置信息、获取文件的方法

递归：方法自己调用自己。递归要注意，不能递归太深，递归必须有结束条件。

递归案例：遍历文件树、删除非空文件夹