Java-BIO（阻塞式IO）-基于抽象类

包：java.io

核心掌握五个类（File,OutputStream,InputStream,Reader,Writer，Scanner）+ 一个接口（Seriallzable）

1.File文件操作类

即可描述文件还可以描述文件夹🗀

File类是唯一个与文件本身操作（创建，删除，取得信息）有关的程序类

> **文件内容读取修改需要输入输出流**

1.1File类的使用（描述文件）

1.1.1产生File对象

java.io.File类是一个普通的类，直接产生实例化对象即可。如果要实例化对象则需要使用到两个**构造方法**：

I.

```
public File(String pathname) ：根据文件得到绝对路径来产生file对象
```

II.

```
public File(URI uri) ：根据网络产生File对象
```

1.1.2常用操作方法

- 创建一个新文件

```
public boolean createNewFile() throws IOException
```

- **判断一个文件是否存在**

```
public boolean exists()
```

- **删除文件**

```
public boolean delete()
```

以上实现了简化的文件处理操作，但是代码存在两个问题：
实际项目部署环境可能与开发环境不同。那么这个时候路径的问题就很麻烦了。windows下使用的是""，而Unix系统下使用的是"/"。所以在使用路径分隔符时都会采用File类的一个常量"public static final String separator "来描述

Tips:

1.由于Windows与Linux文件路径不同（方式不同）导致不能跨平台
所以产生一个分间分隔符：file.separator

```
// separator由不同操作系统下的JVM来决定到底是哪个杠杠！
 File file = new File(File.separator + "Users" + File.separator + "yuisama" + File.separator + "Desktop"
 + File.separator + "TestIO.java")
```

2.在Java中要进行文件的处理操作是要通过本地操作系统支持的，在这之中如果操作的是同名文件，就可能出现延迟的问题。（开发之中尽可能避免文件重名问题）

1.2目录操作（描述文件夹🗁）

- **取得父路径的File对象**

```
public File getParentFile()
```

- **取得父路径的目录**

```
public String getParent()
```

若想创建父路径，此时好取得父路径的File类对象。

重要方法：创建多级父路径（一次性创建多级不存在的父路径）

```
public boolean mkdirs()
```

- 创建一个目录

```
public boolean mkdir()
```

//1.取得File对象。

2.判断父路径是否存在，不存在创建多级父路径。 ()

3.判断文件是否存在，不存在则创建。

1.3取得文件信息

- **判断目录是否是文件**

```
public boolean isFile()
```

- **判断Filed对象是否是文件夹🗁**

```
public boolean isDirectory()
```

- **取得文件大小 -单位为字节**

```
public long length()
```

- **取得最后一次修改日期**

```
 public long lastModified()
```

//保证文件存在再进行操作

```
 if (file.exists() && file.isFile()) {
```

- **列举一个目录的全部组成- -调用方法的对象是一个：文件夹🗁**

```
public File[]listFiles()
```

//将IO操作放到线程中执行
如果没有的话：

线程阻塞问题：
现在所有代码都是在main线程下完成的，如果listAllFiles()方法没有完成，那么对于main后续的执行将无法完 成。这种耗时的操作让主线程出现了阻塞，而导致后续代码无法正常执行完毕。如果不想让阻塞产生，最好再产生 一个新的线程进行处理

```
long start = System.currentTimeMillis();
long end = System.currentTimeMillis();
System.out.println(end-start);
```

2.字节与字符流

2.1流操作简介

File类不支持文件内容处理，如果要处理文件内容，必须要通过流的操作模式来完成。流分为输入流和输出流
在java.io包中，流分为两种：字节流与字符流

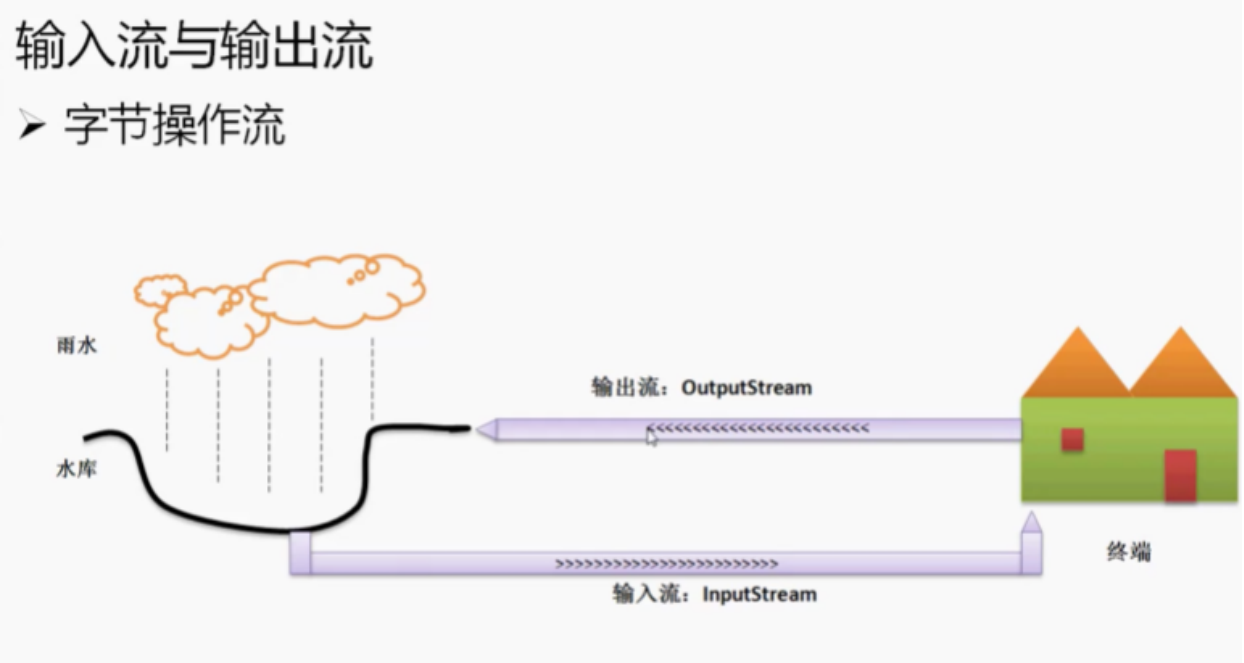
- 字节流（byte）是原生操作，无须转换
    - 可以处理文本文件，图像，音乐，视频等资源

> **1.字节流：InputStream,OutputStream**

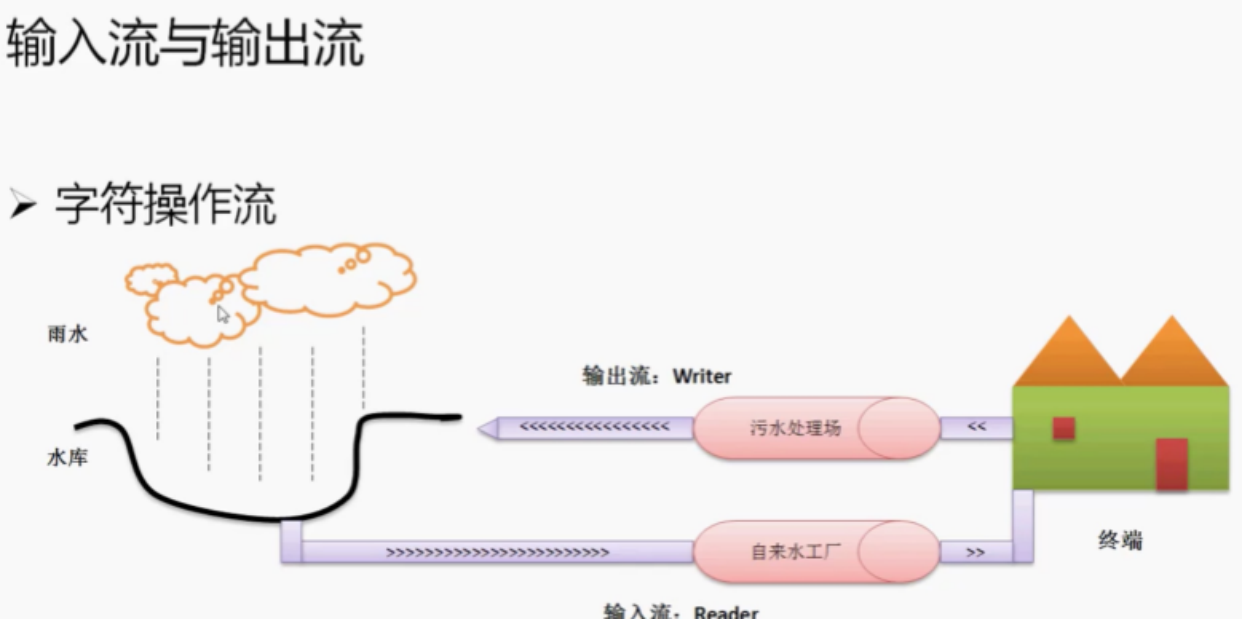
- 字符流（char）是经过处理后的操作
    - 可以处理中文文本

> **字符流：Reader，Writer**

描述如下：



eg：字节流：流媒体文件 音频，视频



有第三方的流

流模型的操作的流程

1.**取得**终端对象
2.根据终端对象**取得输入输出流**
3.根据输入输出流进行**数据的读取和写入**
4.关闭流

IO操作属于资源处理，所有的资源处理（IO操作，数据库操作，网络操作）**在使用后一定要关闭**

2.2字节输出流OutputStream

```
1 public abstract class OutputStream implements Closeable, Flushable {
```

OutputStream类实现了Closeable，Flushable两个接口，这两个接口中的方法：

关闭流：

```
1 Closeable: public void close() throws IOException;
```

刷新流：

```
1  Flushable: public void flush() throws IOException;
```

核心方法：write方法

1.将给定的字节数组全部输出

```
1 public void write(byte b[]) throws IOException : 将给定的字节数组全部输出
```

2.将给定的字节数组以off位置开始输出len长度后停止输出（字节数组的部分数组输出到终端中）

```
1 public void write(byte b[], int off, int len) throws IOException : 将给定的字节数组以off位置开始输出len长度后停止输出
```

3.输出单个字节

```
1 public abstract void write(int b) throws IOException;
```

使用OutputStream输出数据时若指定的文件不存在，FileOutputStream会自动创建文件（不包含创建目录）
使用FileOutputStream输出内容时，默认是文件内容的覆盖操作。

若要进行文件内容的追加使用如下的构造方法

```
public FileOutputStream(File file, boolean append)
```

JDK1.7追加了AutoCloseable自动关闭接口（close（）方法不用人为掉用）
必须使用try-catch{}代码块，推荐显示关闭

2.3字节输入流InputStream

```
public abstract class InputStream implements Closeable {
```

- 将读取的内容放入字节数组中

```
public abstract int read() throws IOException;
```

将读取的内容放入字节数组中

返回值如下三种：
1.返回 b.length :未读取的数据大于存放的缓冲区的大小，返回字节数组的大小。（一次读不完，且剩余大小大于缓冲区大小）

2.返回一个大于0的整数，此整数小于b.length ： 未读取的数据小于存放的缓冲区大小，返回剩余缓冲区大小。（一次读不完，且剩余大小小于缓冲区大小）

3.返回-1（此时数据已经读取完毕反回-1） 标志：此时数据已经读取完毕

- **将文件中的数据读到指定位置off开始的始输出len长度后停止输出**

```
public int read(byte b[], int off, int len) throws IOException {=
```

- **将读取的内容全部放入字节数组中**

```
public int read(byte b[]) throws IOException
```

- 读取单个字节，每次读取一个字节的内容，直到没有数据了返回-1

```
public abstract int read() throws IOException
```

2.4字符输出流Write

字符适合于处理中文数据，Writer是字符输出流的处理类，这个类的定义如下:

```
1 public abstract class Writer implements Appendable, Closeable, Flushable
```

字符输出流可以直接支持字符串的输出

```
1
```

步骤

1. 取得FIle对象
2. 取得字符输出流
3. 数据输出
4. 关闭流

2.5字符输入流

在上面讲到的Writer类中提供有方法直接向目标源写入字符串，而在Reader类中没有方法可以直接读取字符串类 型，这个时候只能通过字符数组进行读取操作

直接接收整个字节数组

```
1 public void write(char cbuf[]) throws IOException
```

2.6字符流和字节流的区别

字符流若未关闭字符流在缓冲区存储，不会输出到目标终端。要想将数据输出，要么将输出流关闭要么使用flush()强制刷新缓冲区。

3转换流（字节流->字符流）

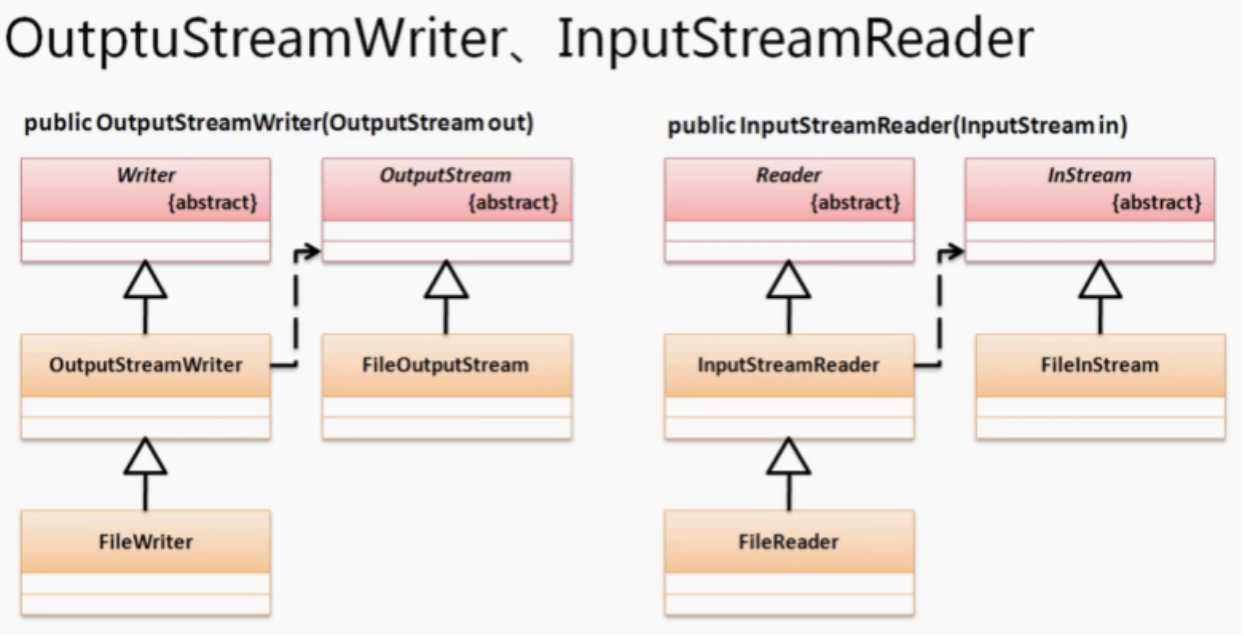
现在为止已经知道了两种数据流：字节流和字符流。实际上这两泪流是可以进行互相转换处理的。

2.7.1OutputStreamWriter(字节输出流->字符输出流)（Writer对于文字的输出要比OutputStream方便)

2.7.2InputStreamReader(字节输入流->字符输出流)（InputStream读取的是字节，不方便中文的处理)

字符流都具体子类大都是通过转换流将字节流转换为字符流FIleWrite继承转换流)

3.1各个输入输出流和转换流转换的框图



3.2文件拷贝（重要）

linux下文件拷贝命令:"cp 源文件路径 目标文件路径"

1. 原文件路径
2. 目标文件路径
3. 单独写一个方法，传入原文路径，目标文件路径
    a. 取得源文件和目标文件的File对象

b. 取得输入输出流

c. 数据输出

4.字符编码

1.GBK,CB2313：**（只描述中文）**
GBK包含繁体与繁体中文，GB2312只包含简体中文
GB：国标

2.UNICODE
java提供的16进制编码，可以描述世界上任意一个语言，但是编码进制数太高，编码体积较大。

3.ISO-8859-1：国际通用编码，不支持中文，浏览器默认编码（**国际通用编码，但是所有的编码都需要进行转换。 ）**

4.UTF编码：结合UNICODE与ISO-8859-1,最长采用的是UTF-8编码。 **（相当于结合了UNICODE、ISO8859-1，也就是说需要使用到16进制文字使用UNICODE，而 如果只是字母就使用ISO8859-1，而常用的就是UTF-8编码形式。）**

乱码产生的原因
编解码不一致。 （95%）
数据丢失（%5） 网络环境交较差（一般不会原因是我们用的是TCP）

网路：TCP(挥手和握手)
算法：数据结构 +二叉树
算法：动归

5.内存流

5.1内存流概念（以内存为终端的输入输出流）

内存：当作临时文件（全局为1：）
字节内存流（*以内存做为终端，或者称之为参考系*）

ByteArrayInputStream、ByteArrayOutputStream

- 将指定的字节数组内容存放到内存中。内存输入

```
public ByteArrayInputStream(byte buf[]) {
```

-

```
public ByteArrayOutputStream() {
```

步骤

1. //取得终端对象以及取得输入输出流
2. 数据的读取与写入
3.

字符内存流

CharArrayReader、CharArrayWriter

5.2实现两个文件的合并处理

6.打印流（输出流的强化版本）

6.1系统提供的专门的打印流

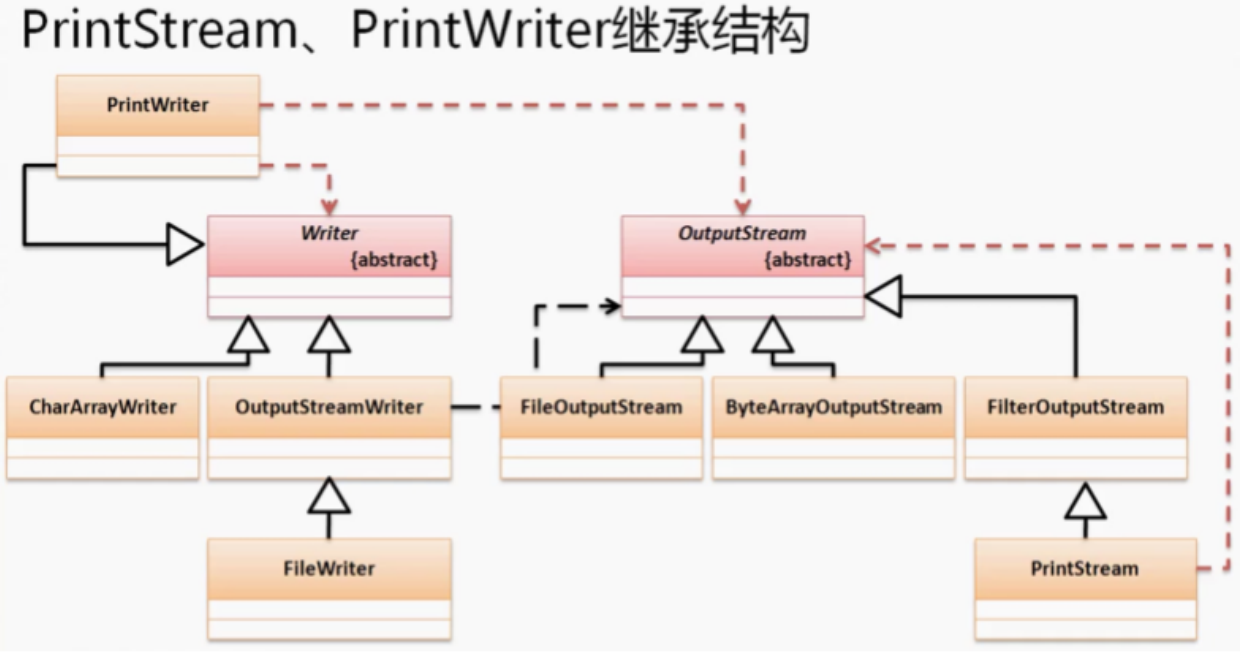
字节打印流：PrintStream

字符打印流：PrintWriter

6.2打印流的结构

使用系统提供的打印流 打印流分为字节打印流：PrintStream、字符打印流:PrintWriter，以后使用PrintWriter几率较高。首先来观察这两 个类的继承结构与构造方法：

PrintStream 类:	PrintWriter 类:
java.lang.Object → |- java.io.OutputStream → → |- java.io.FilterOutputStream → → → |- java.io.PrintStream	java.lang.Object → |- java.io.Writer → → |- java.io.PrintWriter
public PrintStream(OutputStream out)	public PrintWriter(OutputStream out) public PrintWriter (Writer out)



打印流得设置属于我们的装饰设计模式-基于抽象类的

特点：核心依然是某个类（OutputStream提供的write（））的功能,但是为了得到**更好的操作效果**，让其支持的功能更多一些，使用装饰类（PrintStream）

优点：很容易更换装饰类来达到更好的操作效果。

缺点：**由于装饰设计模式的引入造成类（结构）复杂。**

（eg:买服装的例子）

7.System类对IO的支持

标准输出（输出到显示器，颜色为黑色）：System.out （public final static PrintStream out）

标准输入（键盘）：System.in （public final static PrintStream err）

错误输出：System.err(输出到显示器，颜色为红色) (public final static InputStream in)

7.1系统输出

系统提供的out,err对象均是PrintStream的对象
由于System.out是PrintStream的实例化对象，而PrintStream又是OutputStream的子类，所以可以直接使用 System.out直接为OutputStream实例化，这个时候的OutputStream输出的位置将变为屏幕。

以后输出采用日志（Log）-格式化输出

7.2系统输入

System.in 是InputStream的直接子类

8.两种输入流

8.1 BufferedReader、BufferedInputStream（字符输入类）

readline()直接读取一行，默认以回车换行

8.2java.util.Scanner（唯一个不在IO包中的与IO有关的类）

hasNextXX()：判断是否有指定类型的数据输入
nextXX（）：获取只当类型数据
useDelimiter("指定分隔符”)：自定分隔符

支持正则表达式：前端部分