## IO输入流，输出流

# Flie对象表示文件或路径。

常见方法：

1. 创建。 new File(“1.txt”); 只是创建文件对象,并不创建.但是可以直接写.

boolean createNewFile():在指定位置创建文件，如果该文件已经存在，则不创建，返回false。

和输出流不一样，输出流对象一建立创建文件。而且文件已经存在，会覆盖。

boolean mkdir():创建文件夹。

boolean mkdirs():创建多级文件夹。

1. 删除。

boolean delete()：删除失败返回false。如果文件正在被使用，则删除不了返回falsel。

void deleteOnExit();在程序退出时删除指定文件。

1. 判断。

boolean exists() :文件是否存在.

isFile():

isDirectory();

isHidden();

isAbsolute();

1. 获取信息。

getName():

getPath():

getParent():

getAbsolutePath()

long lastModified()

long length() ：返回字节数长度

5. 列出指定目录下文件或者文件夹，包含子目录中的内容。

因为目录中还有目录，只要使用同一个列出目录功能的函数完成即可。在列出过程中出现的还是目录的话，再次调用本功能。也就是函数自身调用自身。这种表现形式，或者编程手法，称为递归。

递归要注意：

限定条件。

要注意递归的次数。尽量避免内存溢出。

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

File file = **new** File("H:\\");

*showDir*(file);

}

**public** **static** **void** showDir(File file) {

File []files = file.listFiles();

**for** (File file2 : files) {

**if** (file2.isDirectory())

*showDir*(file2);

**else**

*sop*(file2);

}

}

# 字节流的两个抽象基类类：

InputStream OutputStream

FileInputStream和FileOutputStream实现文件读写

BufferedInputStream和BufferedOutputStream提供缓冲区功能

public int read() //读取一个字节，返回值int。达到末尾则返回-1

public int read(byte[] b) //读取数据到字节数组中

public int read(byte[] b,int off,int len) //b缓冲数组。返回读取的字节个数，末尾返回-1

public void write(int b) //写int的后八位

public void write(byte[] b)

public void **write**(byte[] b,int off,int len)

字节流的基本实现，实现文件的拷贝：(利用缓冲区。)

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

BufferedInputStream input = new BufferedInputStream (new FileInputStream(file));

BufferedOutputStream output = new BufferedOutputStream(new FileOutputStream("1.txt"));

byte [] buf = new byte[8\*1024];

int len = 0;

while((len = inputStream.read(buf,0,buf.length)) != -1){

outputStream.write(buf,0, len);

}

inputStream.close();

outputStream.close();

# 字符流的两个抽象类：Writer Reader

FileWriter和FileReader可实现文件的读写操作

**new** FileWriter("1.txt"); //创建文件默认覆盖

**new** FileWriter("1.txt",**true**); //不覆盖源文件，并且在已有文件末尾处续写

//创建一个文件读取流对象，和指定名称的文件相关联。

//要保证该文件是已经存在的，如果不存在，会发生异常FileNotFoundException

**new** FileReader ("demo.txt");

# 字符流缓冲区

BufferedWriter和BufferedReader

缓冲区的出现是为了提高流的操作效率而出现的。所以在创建缓冲区之前，必须要先有流对象。

//记住，只要用到缓冲区，就要记得刷新。

bufw.flush();

//其实关闭缓冲区，就是在关闭缓冲区中的流对象。

bufw.close();

BufferedReader :

readLine() //读一行，读到换行符为止!!!但内容不包括换行符。读到末尾返回null

newLine(); //提供了跨平台的写入换行符 windows \r\n linux \n

BufferedWriter

File file = new File("file.txt"); //文件已经存在。

//使用字符流实现文本文件的复制

BufferedReader bufferedReader = new BufferedReader(new FileReader(file)); BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("2.txt")); String buf;

//使用buffer缓冲的时候可以读取和写入一行

while((buf = bufferedReader.readLine()) != null){

bufferedWriter.write(buf);

//写入新的一行

bufferedWriter.newLine();

}

//关闭流，只关闭缓冲期即可。缓冲区操作的就是其包装的流。

bufferedReader.close();

bufferedWriter.close();

# 字符流的基本实现（转换流）

InputStreamReader 完成byte流解析为char流,按照编码解析 //读取字符

OutputStreamWriter 提供char流到byte流，按照编码处理 //写入字符。

//键盘读取的最常见写法。

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

//控制台输出。

BufferedWriter bufw = new BufferedWriter(new OutputStreamWriter(System.out));

//字节流转化字符流

FileInputStream fileInputStream = new FileInputStream("input.txt");

BufferedReader reader = new BufferedReader(new InputStreamReader(fileInputStream));

//将字符流转化为字节流

FileOutputStream fos = new FileOutputStream("output.txt"));

BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(fos));

//实现拷贝文件的操作

String buf;

while((buf = reader.readLine()) != null){

writer.write(buf);

writer.newLine();

System.out.println(buf);

}

//关闭流，只关闭缓冲期即可。缓冲区操作的就是其包装的流。

reader.close();

writer.close();

# 装饰设计模式

当想要对已有的对象进行功能增强时，可以定义类，将已有对象传入，基于已有的功能，并提供加强功能。

那么自定义的该类称为装饰类

**class** Person{

**public** **void** chifan(){

System.***out***.println("吃饭");

}

}

//装饰类，增强person的功能。

**class** SuperPerson {

**private** Person p ;

SuperPerson(Person p){ //通过构造方法接收被装饰的对象。

**this**.p = p;

}

**public** **void** superChifan(){ //基于被装饰的对象的功能，提供更强的功能。

System.***out***.println("开胃酒");

p.chifan();

System.***out***.println("甜点");

}

}

**class** PersonDemo{

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

Person p = **new** Person();

SuperPerson sp = **new** SuperPerson(p);

sp.superChifan();

}

}

MyReader//专门用于读取数据的类。

|--MyTextReader

|--MyBufferTextReader

|--MyMediaReader

|--MyBufferMediaReader

|--MyDataReader

|--MyBufferDataReader

class MyBufferReader{

MyBufferReader(MyTextReader text){

}

MyBufferReader(MyMediaReader media){

}

}

上面这个类扩展性很差。

找到其参数的共同类型。通过多态的形式。可以提高扩展性。

class MyBufferReader extends MyReader{

private MyReader r;

MyBufferReader(MyReader r){

}

}

MyReader//专门用于读取数据的类。

|--MyTextReader

|--MyMediaReader

|--MyDataReader

|--MyBufferReader

以前是通过继承将每一个子类都具备缓冲功能。

那么继承体系会复杂，并不利于扩展。

现在优化思想。单独描述一下缓冲内容。

将需要被缓冲的对象。传递进来。也就是，谁需要被缓冲，谁就作为参数传递给缓冲区。

这样继承体系就变得很简单。优化了体系结构。

装饰模式比继承要灵活。避免了继承体系臃肿。

而且降低了类于类之间的关系。

装饰类因为增强已有对象，具备的功能和已有的是相同的，只不过提供了更强功能。

所以装饰类和被装饰类通常是都属于一个体系中的。

## RandomAccessFile

但是它是IO包中成员。因为它具备读和写功能。

内部封装了一个数组，而且通过指针对数组的元素进行操作。

可以通过getFilePointer获取指针位置，

同时可以通过seek改变指针的位置。

其实完成读写的原理就是内部封装了字节输入流和输出流。

通过构造函数可以看出，该类只能操作文件。

而且操作文件还有模式：只读r，，读写rw等。

如果模式为只读 r。不会创建文件。会去读取一个已存在文件，如果该文件不存在，则会出现异常。

如果模式rw。操作的文件不存在，会自动创建。如果存则不会覆盖。