**File类（重点）：**

**一、对文件的操作：**

File类是位移一个对文件本身进行操作的类，但是要操作文件一定要设置好文件或文件夹路径。使用如下构造方法：

构造方法：public file（String pathname），传入完整的路径。

构造方法：public file（file parent，String child）,传入父路径和子路径。

范例：基本文件操作

.创建文件：public Boolean createNewFile（） throws Exception；：public

删除文件：public boolean delete（）；

首先应该判断文件是否存在：public boolean exists（）；

**public** **class** File\_1 {

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

File file = **new** File("D:\\hello.txt");//使用构造函数创建一个文件。必须指明文件路径。 但是使用“\\”标识路径分割的话，不可以实现平台的移植，所以需要改进

//File file = **new** File("D:" + File.***separator*** +"hello.txt");//使用构造函数创建一个文件。必须指明文件路径。

//使用separator关键字来代替斜线的路径分隔符，实现可移植性

//此种程序只能实现创建了父路径以后建立目录才能创建文件 所以需改进

**if**(file.exists()){

file.delete();

}**else**{

file.createNewFile();

}

}

}

**完整程序：**

**public** **class** File\_1 {

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

File file = **new** File("D:" + File.***separator*** +"hello" + File.***separator*** + "nihao" + File.***separator*** + "haode.txt");//使用构造函数创建一个文件。必须指明文件路径。

//使用separator关键字来代替斜线的路径分隔符，实现可移植性

**if**(!file.getParentFile().exists()){//利用getParentFile()方法得到父路径

file.getParentFile().mkdirs();//利用父文件的，mkdirs()方法实现创建多级目录。

}

**if**(file.exists()){

file.delete();

}**else**{

file.createNewFile();

}

}

}

除了以上基本的文件和文件夹的操作外，还有一些取得文件信息的操作方法：

1. 判断路径是否是文件：pubic boolean isFile（）；
2. 判断路径是否是文件夹：public boolean isDirectory（）；
3. 最后一次修改日期：public long lastModified();
4. 取得文件的大小:pubilc long length();

**public** **class** File\_2 {

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

File file = **new** File("D:" + File.***separator*** + "dog.jpg");

**if**(file.exists()){

System.***out***.println(file.isFile()?"是文件":"不是文件");

System.***out***.println(file.isDirectory()?"是文件夹":"不是文件夹");

System.***out***.println("最后一次修改日期：" + **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS").format(file.lastModified()));

}

System.***out***.println("文件大小：" + (file.length()/(**double**)1024) + "kb");

}

}

**操作文件的方法：**

1. 列出文件夹所有目录里的子目录内容：

基本原则：首先要判断给定的file对象是否是目录，如果是则继续列出，若不是则输出该文件 使用listfiles（）方法

**package** iooparation;

**import** java.awt.image.RescaleOp;

**import** java.io.File;

**import** javax.naming.spi.DirStateFactory.Result;

**import** javax.swing.JPopupMenu.Separator;

**public** **class** File\_3 {

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

File file = **new** File("D:" + File.***separator*** + "ASM");

*print*(file);

}

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

**if**(file.exists() && file.isDirectory()){//如果文件存在且是文件

File result [] = file.listFiles();//列出全部内容

**if**(result != **null**)

**for**(**int** x = 0;x < result.length;x++){

// System.out.println(result[x]);

*print*(result[x]);//递归调用

}//这种方法是输出该文件夹下的所有子目录和文件

}**else**{

System.***out***.println(file);//输出该文件

//file.delete();//删除该文件夹 谨慎使用

}

}

}

1. **对文件内容的操作：**
2. 字节输出流：

在OutputStream类之中存在有三个write（）方法：

1. 输出单个字节：public abstract void write（int b）throws exception;
2. 输出全部字节：public void write(byte [] b) throws Exception;
3. 输出部分字节：public void write(byte [] b,int off,int len) throws Exception

但是outputStream只是一个抽象类，所以要去的实例化对象，必须依靠子类fileoutputstream来实例化。（FileOutputStream）:

构造方法：public FileOutputStream(File file) throws FileNotFountException //覆盖原有文件的内容

构造方法：public FileOutputStream(File file boolean append) throws FileNotFountException 在原有文件内容后面追加

**public** **class** OutputStream {

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

File file = **new** File("D:" + File.***separator*** + "test" + File.***separator*** + "demo.txt");

//通过file类定义输出路径

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}

//通过OutputStream的子类对象为父类对象实例化

FileOutputStream output = **new** FileOutputStream(file);

String msg = "Hello world";//要输出的数据

**byte** data[] = msg.getBytes();//将字符串变为字节数组

output.write(data);//输出全部字节

output.write(1);//输出单个字节

Output.write(data,2,6);//实现部分写入

output.close();

}

}

上面是实现了对原有的文件内容进行覆盖。下面的构造方法是实现对文件内容的追加

FileOutputStream output1 = **new** FileOutputStream(file,**true**);

1. **字节输入流：**

读取文件使用字节输入流InputStream类、

在inputStream类中有三个读取方法read():

1. 读取单个字节：public abstract int read() throws IOException;
   1. 每次使用read()操作读取一个字节数据，此时返回的是数据，如果数据已经读取完了则int返回-1；
2. 读取内容到字节数组：public int read(byte[] b) throws IOException
   1. 将内容读取到字节数组之中，返回读取的个数，如果读取完毕则返回的是-1；
3. 读取内容到部分字节数组：public int read(byte[] b,int off,int len) throws IOException
   1. 将指定长度的内容读取到字节数组之中，返回读取个数，如果读取完毕则返回-1；

但是InputStream是一个抽象类，所以我们应该使用它的子类FileInputStream实例化对象，我们只关心子类的构造方法：

范例：（使用inputstream读取数据）

**public** **class** InputStreamTest\_1 {

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

File file = **new** File("D:" + File.***separator*** + "test" +File.***separator*** +"demo.txt");

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}

**if**(file.exists()){//要操作的文件存在

InputStream input= **new** FileInputStream(file);

**byte** data[] = **new** **byte**[1024];//一个字节数组用来存放从文件中读取的字节内容（全部读取）

**int** len = input.read(data);//得到读取到数组中的字节个数

System.***out***.println("[" + **new** String(data,0,len)+ "]");

input.close();

}

}

**范例：使用while实现从文件单个读取：**

**public** **class** InputStreamTest\_2 {

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

File file = **new** File("D:" + File.***separator*** + "test" + File.***separator*** + "demo.txt");

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}//在io开发中最常用的的while读取

**if**(file.exists()){

InputStream input = **new** FileInputStream(file);

**byte**[] data = **new** **byte**[1024];//用于存放从文件里读取出来的全部内容

**int** len = input.read(data,0,10);//表示只读0-10

**int** temp = 0;//每次读取进来的数据

**int** foot = 0;//定义数组脚标

**while**((temp = input.read()) != -1){

//组成一：temp = input.read(),表示读取单个字节

//组成二：(temp = input.read()) != -1;表示temp的内容是不是-1，如果不是，则继续读取

data[foot++] = (**byte**)temp;

}

System.***out***.println("读取结果：" + **new** String(data));

input.close();

}

}

}

1. **字符输出流：**

在字节输出流之后为了方便又引入了字符输入输出流（writer 和 reader），字符输出流是以char为主，且直接支持String的操作

Writer类提供一系列的操作方法：

）a 输出字符串：public void writer(String str) Throws Exception;

)b 输出字节数组：public void writer(char[] cbuf)Throws Exception;

范例：使用Writer类实现内容的输出

**public** **class** WriteTest\_1 {

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

File file = **new** File("D:" + File.***separator*** + "test" + File.***separator*** + "d.txt");

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}

Writer out = **new** FileWriter(file);//定义一个字符输出流对象

String msg = "nihaoma!wohenhao!";

out.write(msg);//直接用write()方法直接输出对象

out.close();

}

}

**字符输入流：Reader**

Reader 是负责进行字符数据读取的，同时在这个类中可以使用read()方法读取数据

读取数据：public int read(char[] cbuf) throws Exception

范例：读取数据

**public** **class** ReaderTest\_1 {

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

File file = **new** File("D:" + File.***separator*** + "test" +File.***separator*** + "d.txt");

**if**(file.exists()){

Reader read = **new** FileReader(file);

**char** data[] = **new** **char**[1024];

**int** len = read.read(data);

System.***out***.println(**new** String(data,0,len));

read.close();

}

}

}

两种读取的操作本质上区别不大，只是字符操作流操作的是char/String，而字节流只读取字节数据。

**字节流和字符流的区别及如何选择：**

1. 字符流：当程序处理中文的时候，字符流是最方便的；
2. 字节流：当程序处理二进制数据（图片，音乐，电影）或进行网络传输，或则保存到磁盘数据

除了以上的区别以外，字节流在进行操作的时候是直接与操作终端进行交互，而字符流要经过缓冲区的处理以后才可以进行操作，以OutputSteam 和Writer两个类输出文件为例，如果OutputSteam输出的最后可以不关闭输出流，如果是writer类输出，那个保存在缓冲区的数据将无法输出，或则强制性刷新缓冲区。

**public** **class** WriteTest\_1 {

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

File file = **new** File("D:" + File.***separator*** + "test" + File.***separator*** + "d.txt");

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}

Writer out = **new** FileWriter(file);//定义一个字符输出流对象

String msg = "nihaoma!wohenhao!";

out.write(msg);//直接用write()方法直接输出对象

out.flush();//强制刷新缓冲区 没有关闭流

}

}

**文件拷贝程序：（核心，第七个代码模型）：**

**public** **class** FileCopyTest\_1 {

**public** **static** **void** main(String[] args) **throws** Exception {//需要配置参数

//======================================================

//辅助代码

**if**(args.length != 2){//输入参数不足两个 一个是源文件参数 一个是目标文件参数

System.***out***.println("命令语法错误！");

System.*exit*(11);//退出程序

}

File inFile = **new** File(args[0]);//源文件路径

**if**(!inFile.exists()){//如果源文件不存在

System.***out***.println("源文件不存在!");

System.*exit*(1);

}

//======================================================

File outFile = **new** File(args[1]);//目标文件

**if**(!outFile.getParentFile().exists()){

outFile.getParentFile().mkdirs();//创建目录

}

InputStream input = **new** FileInputStream(inFile);//字节输入流

OutputStream output = **new** FileOutputStream(outFile);//字节输出流

//========================================此种方法只适合一个字节一个字节的读入 效率很低

// int temp = 0;//接收每次读取的数据 一个字节一个字节的读入

// while((temp = input.read()) != -1){//有内容读取

// output.write(temp);//输出内容

// }

//========================================此种方法只适合一个字节一个字节的读入 效率很低

**byte**[] tem = **new** **byte**[1024];//

**int** len;

**while**((len = input.read(tem)) != -1){//每次读取1024个字节 读取时返回的是长度

output.write(tem,0,len);//向目标文件中写入读取的数据

}

input.close();

output.close();

**long** end = System.*currentTimeMillis*();

System.***out***.println("文件拷贝所花时间：");

}

}

内存操作流(次重点)：

在之前使用的文件的操作流，可以发现过程中都是以文件为终端进行文件的输入输出。如果说现在需要使用io操作，但是又不希望产生文件的话，可以将我们的输入输出终端改为内存、

内存操作流分为两类：

1. 字节流操作：ByteArrayInputStream、ByteArrayOutputStream
2. 字符流操作：CharArrayReader、CharArrayWriter

对比文件与内存操作的区别：

文件操作流：

|-输出： 程序==>OutputStream==>文件

|-输入 程序<==InputStream<==文件

内存操作流：

|-输出：程序==>ByteArrayInputStream==>内存

|-输入：程序<==ByteArrayOutputStream<==内存

范例：将字母由小写转大写：

**public** **class** MemoryStream {//

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

String str = "hello word";

InputStream input = **new** ByteArrayInputStream(str.getBytes());

ByteArrayOutputStream output = **new** ByteArrayOutputStream();

**int** temp = 0;//保存每次读进来的数据

**while**((temp = input.read()) != -1){

output.write(Character.*toUpperCase*(temp));//所有内容都读取出来了

}

String msg = output.toString();//转换成字符串

input.close();

output.close();

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

}

//此时执行了IO操作但是没有文件的产生 可以理解为临时文件

}

**System对io的支持：**

**package** iooparation;

**import** java.io.InputStream;

**public** **class** SystemIn {

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

// InputStream input = System.in;//由键盘输入而来

// System.out.println("请输入数据：");

// byte data[] = new byte[1024];//保存输入数据

// int len = input.read(data);//接收键盘数据 此时有键盘输入的长度限制

// System.out.println(new String(data,0,len));

// input.close();

//不设置键盘的读取长度

InputStream input = System.***in***;//由键盘输入而来

System.***out***.println("请输入数据：");

StringBuffer buf = **new** StringBuffer();

**int** temp =0;

**while**((temp = input.read()) != -1){//以一个的读取

**if**(temp == '\n')

**break**;//若输入回车则退出

buf.append((**char**)temp); //但是不支持中文输入

}

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

input.close();

}

}

**打印流：PrintStream(字节打印流) 、PrintWriter(字符打印流)**

**第八个代码模型（装饰设计模式）：**

（以后凡是遇见程序输出数据都用printStream,不在用outputstream）

Public String readLine();//读取一行数据 并且返回字符串，意味着可以向任何形式转换，可以使用正则验证。

PrintStream类其实就是将OutputStream类装饰了一下，其实还是基于OutputStream类。

**范例：**

**public** **class** PrintStreamTest\_1 {

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

File file = **new** File("D:" + File.***separator*** + "wen" + File.***separator*** + "my.txt");

**if**(!file.getParentFile().exists()){

file.getParentFile().mkdirs();

}

PrintStream print = **new** PrintStream(file);//定义打印流对象

print.print("姓名：");

print.println("张三");

print.print("年龄：");

print.println(20);

print.print("性别：");

print.println("男");

print.print("爱好：");

print.println("女");

print.close();

}

}

**Scanner类（核心：第八个代码模型）**

在JDK1.5之后增加了一个新类，java.Util.Scanner类，此类专门负责处理输入数据，比BufferedReader更加简单，那么在这个类中主要关心如下几个方法：

1. **构造方法：public Scanner(InputStream source);**
2. **设置读取分割符：public Scanner useDelimiter(String pattern);**
3. **判断是否有数据：public boolean hasNextXxx();**
4. **取数据：public 数据 nextXxx();**

范例：利用scanner实现键盘输入：

**public** **class** ScannerTest\_1 {

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

Scanner scanner = **new** Scanner(System.***in***);

// scanner.useDelimiter("\n");//设置以换行为分割符

// System.out.println("请输入数据：");

System.***out***.println("请输入年龄：");

**if**(scanner.hasNextInt()){//判断是否有数据

System.***out***.println(scanner.nextInt());

}**else**{

System.***out***.println("输入的数据有误！");

}

// if(scanner.hasNext()){

// String data = scanner.next();//有数据

// System.out.println(data);

// }

scanner.close();

}

}

**范例：实现键盘输入日期**

**public** **class** ScannerTest\_2 {

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println("请输入日期：");

**if**(scanner.hasNext("\\d{4}-\\d{2}-\\d{2}")){//利用正则表达式判断日期形式

System.***out***.println(**new** SimpleDateFormat("yyyy-mm-dd").parse(scanner.next()));

}**else**{

System.***out***.println("输入的数据有误！");

}

}

}

**范例：使用scanner实现文件的读取操作：**

**public** **class** ScannerTest\_2 {//第八个代码模型 实现文件的读入

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

Scanner scanner = **new** Scanner(**new** FileInputStream(**new** File("D:"

+ File.***separator*** + "my.txt")));//实现文件的读入

scanner.useDelimiter("\n");//以换行符作为分割符

**while**(scanner.hasNext()){

System.***out***.println(scanner.next());

}

scanner.close();

}

}

**结论：第八个代码模型：**

1. **如果程序输出内容那么使用打印流（PrintStream 或则 PrintWriter）;**
2. **如果程序输入数据使用Scanner（如果有时候Scanner类不好使，使用BufferedReader类）**

**对象序列化（核心）**

**对象序列化指的是将保存在内容中的对象转化为二进制数据的形式，这样就可以将我们的对象保存在文件之中或则网络传输，但如果想要实现对象序列化的操作，有一个要求：对象所在的类一定要实现java.io.serializable接口，此接口没有任何的方法是，是一个标识接口。**

**范例：（定义可以被序列化对象）**

**Class Person implements Serializable{//现在此类对象可以被序列化了**

**}**

**文件输入输出流操作重点：**

1. **File类的使用：**
   1. **构造方法：public File(String path) File.separator表示分隔符；**
   2. **判断文件是否存在：public boolean exists();**
   3. **得到父路径： public File getParentFile();**
   4. **创建目录：public void mkdirs();**
   5. **删除文件：public void delete();**
2. **文件拷贝程序：InputStream,OutputStream操作。**
3. **程序输出数据使用打印流（PrintStream,PrintWrite）,输入数据使用Scanner(BufferedReader);**
4. **对象序列化，实现类java.io.Serializable接口。**