**Java输入输出**

# 1、访问文件和目录

Java中可以通过File类来操作文件和目录，常用方法如下：

## 1.1、访问文件名

getName: 返回文件名和路径名，如果是路径则返回最后一个路径名。

getPath: 返回File对象所指定的路径名。

getParent: 返回父路径名。

## 1.2、文件检测

exists: 判断文件和目录是否存在。

canRead: 判断文件和目录是否可读。

canWrite: 判断文件和目录是否可写。

isFile: 判断是否文件而不是目录。

isDirectory: 判断是否目录。

## 1.3、文件操作

createNewFile: 如果文件不存在就新建一个文件。

delete: 删除对应的文件或目录。

static File createTempFile: 创建临时文件。

deleteOnExit: java虚拟机退出时，删除对应的文件或目录。

## 1.4、目录操作

mkdir: 创建指定目录

String[] list: 列出所有文件名和路径名

File[] listFiles: 列出所有子文件和路径名

static File[] listRoots: 列出所有的根路径

***路径名中建议用斜线（/）作为路径分隔符。***

## 1.5、文件过滤器

java提供了FilenameFilter的接口，其中有accept方法，如果该方法返回true则返回满足条件的文件名或者路径名。

示例代码：

public class FileIOTest1 {

public static void main(String[] args){

try{

File file = new File(".");

String[] fileList = file.list();

String[] fileListByFilter = file.list(new FileIOTest1().new MyFileNameFilter());

for(String fileName : fileList){

System.out.println("fileName = " + fileName);

}

for(String filterFileName : fileListByFilter){

System.out.println("filterFileName = " + filterFileName);

}

}catch(Exception ex){

ex.printStackTrace();

}

}

class MyFileNameFilter implements FilenameFilter{

public boolean accept(File dir,String name){

return name.startsWith("139");

}

}

}

后台输出：

fileName = .classpath

fileName = .project

fileName = .settings

fileName = 1395109967248

fileName = 1395109998289

fileName = bin

fileName = src

fileName = test33.txt

filterFileName = 1395109967248

filterFileName = 1395109998289

# 2、IO流

java中把不同的输入输出抽象成流。通过流的方式java可以用相同的方式访问不同的输入输出源。

## 2.1、流的分类

### 2.1.1、输入流和输出流：

输入流只能读数据，输出流只能写数据，输入输出是相对于内存而已的。

java的输入流主要以InputStream和Reader作为基类，输出流以OutputStream和Writer作为基类，它们都是抽象类不能直接创建实例。

### 2.1.2、字节流和字符流

字节流以InputStream和OutputStream作为基类，字符流以Reader和Writer作为基类。

区别是字节流以字节为单位读取，字符流以字符为单位读取。

一个字符等于二个字节，一个字节8位。

#### 2.1.2.1、InputStream和Reader

InputStream作为字节流输入流的基类，提供了以下常用方法：

***int read():*** 从输入流中读取单个字节的数据

***int read(byte[] b):*** 从输入流中读取b.length长度的数据，读取的数据放在b数组中，返回读取数据的长度。

***int read(byte[] b,int off,int len):*** 从输入流中读取len字节长度的数据，放入b数组中，不是从数组的第一个位置开始存放，而是从off位置开始存放数据。

Reader作为字符流输入流的基类，提供一些常用方法：

***int read():*** 从输入流中读取当个字符长度的数据

***int read(char[] c):*** 从输入流中读取c.length长度的数据放入c数组中，返回获取的数据长度。

***int read(char[] c,int off,int len):*** 从输入流中读取len长度的数据，放入c数组中，不是从数组的第一个位置开始存放数据，而是从off位置开始存放数据。

*当所有数据读取完毕后read方法会返回-1，表面输入流的结束点。*

#### 2.1.2.2、处理文件的输入输出流

InputStream、OutputStream、Reader、Writer都是输入输出流的基类，java提供了处理文件的输入输出流的实现类。分别是FileInputStream、FileOutputStream、FileReader、FileWriter。

代码示例：

public class FileIOTest2 {

public static void main(String[] args){

FileInputStream fileInput = null;

FileOutputStream fileOutput = null;

try {

fileInput = new FileInputStream("./src/com/test/FileIOTest2.java");

fileOutput = new FileOutputStream("FileIOTest2.txt");

byte[] buf = new byte[1024];

int bufIndex = 0;

while((bufIndex = fileInput.read(buf)) > 0){

fileOutput.write(buf, 0, bufIndex);

}

} catch (Exception e) {

e.printStackTrace();

}finally{

try {

fileInput.close();

fileOutput.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

上面代码表示将FileIOTest2.java源代码作为输入流，输出带FileIOTest2.txt文件中。

程序里打开的IO资源不是内存的资源，垃圾回收机制无法回收，所以需要手动关闭IO资源。

#### 2.1.2.3、转换流

转换流的作用就是将字节流转换成字符流。

InputStreamReader：将字节输入流转换成字符输入流

OutputStreamWriter:将字节输出流转换成字符输出流

示例代码：

public class FileIOTest7 {

public static void main(String[] args){

FileInputStream fis = null;

try{

fis = new FileInputStream("test55.txt");

BufferedReader bf = new BufferedReader(new InputStreamReader(fis));

String buffer = null;

while((buffer = bf.readLine()) != null){

System.out.println("buffer = " + buffer);

}

}catch(Exception ex){

ex.printStackTrace();

}finally{

try {

fis.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

上面代码中用到了BufferedReader，它可以读取文本中的一行数据，以换行符结束，

### 2.1.3、节点流和处理流

从指定IO设备读取的是节点流，连接实际的数据源，也被称为低级流。处理流就是对一个已经存在的流进行分装处理，称为高级流，也被称为包装流。

#### 2.1.3.1、PrintStream处理流

PrintStream的处理功能比较强大，一般文本的输出都应该通过printstream输出，因为printstream包装了底层的节点流，所以当调用printstream的close方式时，底层的节点流的close也被调用了，所以不需要再手动调用节点流的close方法了。

示例代码：

public class FileIOTest3 {

public static void main(String[] args){

FileOutputStream fos = null;

PrintStream ps = null;

try{

fos = new FileOutputStream("test55.txt");

ps = new PrintStream(fos);

ps.print("你好printstream");

}catch(Exception ex){

ex.printStackTrace();

}finally{

ps.close();

}

}

}

## 2.2、重定向标准输入输出

java的标准输入输出分别是System.in和System.out，默认情况下代表键盘和显示器。

System类中提供了3个重定向的方法：

static void setErr(PrintStream ps): 重定向标准错误输出流

static void setIn(InputStream is): 重定向标准输入流

static void setOut(PrintStream ps): 重定向标准输出流

示例代码：

public class RedirectOut {

public static void main(String[] args){

PrintStream ps = null;

try{

ps = new PrintStream(new FileOutputStream("outPut.txt"));

System.setOut(ps);

System.out.println("这是一次测试重定向输出");

}catch(Exception ex){

ex.printStackTrace();

}finally{

ps.close();

}

}

}

上面代码是将outPut.txt文件重定向为输出，所以System.out打印出的字符串将输出到outPut.txt文件中。

## 2.3、RandomAccessFile类

该类可以使程序跳转到文件的任意地方进行读写，可以自由访问文件的任意位置。

RandomAccessFile提供了操作文件指针的两个方法：

long getFilePointer(): 返回文件记录指针的当前位置。

void seed(long pos): 将文件记录指针定位到pos位置。

示例代码：

public class FileIOTest6 {

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

RandomAccessFile randomFile = new RandomAccessFile("test33.txt","r");

System.out.println("init index = " + randomFile.getFilePointer());

randomFile.seek(100);

byte[] buff = new byte[1024];

int hasRead = 0;

while((hasRead = randomFile.read(buff)) > 0){

System.out.println(new String(buff,0,hasRead));

}

randomFile.close();

}

}

上面代码中读取test33.txt文件内容，seek方法中设置了100，表示从文件的第100个字节处开始读取数据。

# 3、对象序列化

对象序列化的目的是将对象保存到磁盘中或直接在网络上传输，将内存中的java对象转换成与平台无关的二进制流。可以理解成将java对象转换成IO流，而反序列化是指把IO流转换成java对象。

如果要一个java对象可序列化，那么这个java类需要实现Serializable接口，该接口没有任何方法，只是个标记接口。

java中提供了ObjectOutputStream和ObjectInputStream对象流来处理序列化和反序列化的操作。他们都是处理流，所以初始化的时候需要传入节点流。

示例代码：

public class WriteObject {

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

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("object.txt"));

Person per = new Person("杰克逊","男");

oos.writeObject(per);

oos.close();

}

}

以上代码需要自己创建一个Person类，代码中将person对象序列化后写入到object.txt文件中。

示例代码：

public class ReadObject {

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

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("object.txt"));

Person p = (Person)ois.readObject();

System.out.println("name = " + p.getName());

System.out.println("sex = " + p.getSex());

}

}

上面代码表示从object.txt文件中读取出person对象， 反序列化读取的仅仅是java对象的数据，而不是java类，因此采用反序列化恢复java对象时，必须提供该java对象的class文件。

# 4、NIO

传统的面向流的输入输出系统一个只能处理一个字节，所以效率比较低。从jkd1.4开始改建了输入输出功能，这些功能统称为新IO。新IO采用内存映射的方式来处理输入输出，将文件或文件的一段区域映射到内存中，这样就可以像访问内存一样访问文件了。

Channel（通道）和Buffer（缓冲）是新IO的核心对象，在新IO中所有的数据都需要通过通道传输，Channel与传统的InputStream、OutputStream最大的不同是其提供了map方法，通过map方法直接可以将“一块数据”映射到内存中，传统的输入输出是面向流的，新IO则是面向块的。Buffer可以理解成一个容器，其实就是一个数组。

## 4.1、Buffer

从内部结构上看，Buffer就是一个数组，它是一个抽象类，ByteBuffer是其用的最多的一个子类，ByteBuffer还有一个子类MappedByteBuffer,它用于表示channel将文件或文件的部分映射到内存后得到的结果，通常MappedByteBuffer由channel的map方法得到。

Buffer有三个比较重要的概念：

容量（captcity）: 缓冲区的容量，buffer的最大数据大小。

界限（limit）: 第一个不应该被读出和写入的缓冲区位置索引

位置（position）: 用于指明下一个被写入和读出的缓冲区索引

Buffer的主要作用就是装入数据，然后输出数据。

当Buffer结束装入数据后，调用flip方法，该方法将position所在位置设为limit，position设为0，然后准备输出数据，输出完数据后调用clear方法，将limit设置到capacity所在的位置，position设为0，准备再次装入数据。

## 4.2、Channel

Channel类似于传统的流对象，Channel可以直接将文件或文件的部分映射成Buffer,不能直接从Channel中存取数据，Channel只跟Buffer交互。

所有的Channel对象都不是通过构造器创建的，而是通过InputStream和OutputStream的getChannel方法获得的。