[Java IO（二）——RandomAccessFile](https://www.cnblogs.com/xiaoxi/p/6483390.html)

RandomAccessFile类支持"随机访问"方式，可以跳转到文件的任意位置处读写数据

/\*\*

\* 写文件

\*/

@Test

public void writeFile() throws IOException{

Employee e1 = new Employee("zhangsan", 23);

Employee e2 = new Employee("lisi", 24);

Employee e3 = new Employee("wangwu", 25);

RandomAccessFile raf0 = new RandomAccessFile("D:/Files/employee.txt", "rw");

raf0.writeBytes(e1.getName());

raf0.writeInt(e1.getAge());

raf0.writeBytes(e2.getName());

raf0.writeInt(e2.getAge());

raf0.writeBytes(e3.getName());

raf0.writeInt(e3.getAge());

raf0.close();

}

/\*\*

\* 读文件

\*/

@Test

public void ReadFile() throws IOException{

RandomAccessFile raf1 = new RandomAccessFile("D:/Files/employee.txt", "r");

int len = 8;

raf1.skipBytes(12); // 跳过第一个员工的信息，其姓名8字节，年龄4字节

System.out.println("第二个员工的信息：");

String str = "";

for (int i = 0; i < len; i++)

{

str = str + (char)raf1.readByte();

}

System.out.println("name：" + str);

System.out.println("age：" + raf1.readInt());

System.out.println("第一个员工的信息：");

raf1.seek(0);

str = "";

for (int i = 0; i < len; i++)

{

str = str + (char)raf1.readByte();

}

System.out.println("name：" + str);

System.out.println("age：" + raf1.readInt());

System.out.println("第三个员工的信息：");

raf1.skipBytes(12); // 跳过第二个员工的信息

str = "";

for (int i = 0; i < len; i++)

{

str = str + (char)raf1.readByte();

}

System.out.println("name：" + str.trim());

System.out.println("age：" + raf1.readInt());

raf1.close();

}

**BufferedWriter、BufferedReader**

为了达到最高的效率，避免频繁地进行字符与字节之间的相互转换，最好不要直接使用FileReader和FileWriter这两个类进行读写，而使用BufferedWriter包装OutputStreamWriter，使用BufferedReader包装InputStreamReader。

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

File file = new File("D:/Files/buffered.txt");

Writer writer = new FileWriter(file);

BufferedWriter bw = new BufferedWriter(writer);

bw.write("1234\n");

bw.write("2345\n");

bw.write("3456\n");

bw.write("\n");

bw.write("4567\n");

bw.close();

writer.close();

if (file.exists() && file.getName().endsWith(".txt")){

Reader reader = new FileReader(file);

BufferedReader br = new BufferedReader(reader);

String str = null;

while ((str = br.readLine())!= null){

System.out.println(str);

}

reader.close();

br.close();

}

}

1：什么是 java 序列化，如何实现 java 序列化？

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。

序列化：把Java对象转换为字节序列的过程。  
反序列化：把字节序列恢复为Java对象的过程。

对象的序列化主要有两种用途：  
　　1） 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中；  
　　2） 在网络上传送对象的字节序列。

序列化API

　　java.io.ObjectOutputStream代表对象输出流，它的writeObject(Object obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。只有实现了Serializable和Externalizable接口的类的对象才能被序列化。

java.io.ObjectInputStream代表对象输入流，它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

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

/\*其中的 D:\\objectFile.obj 表示存放序列化对象的文件\*/

//序列化对象

ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("D:\\objectFile.obj"));

Customer customer = new Customer("王麻子", 24);

out.writeObject("你好!"); //写入字面值常量

out.writeObject(new Date()); //写入匿名Date对象

out.writeObject(customer); //写入customer对象

out.close()；

//反序列化对象

ObjectInputStream in = new ObjectInputStream(new FileInputStream("D:\\objectFile.obj"));

System.out.println("obj1 " + (String) in.readObject()); //读取字面值常量

System.out.println("obj2 " + (Date) in.readObject()); //读取匿名Date对象

Customer obj3 = (Customer) in.readObject(); //读取customer对象

System.out.println("obj3 " + obj3);

in.close();

}

}

2:  怎么样把输出字节流转换成输出字符流,说出它的步骤

答案:使用 转换处理流OutputStreamWriter 可以将字节流转为字符流  
New OutputStreamWriter（new FileOutputStream（File file））;

New inputStreamReader(new FileInputStream(File file))

3:15.如果在对象序列化的时候不想给一个字段的数据保存在硬盘上面,采用那个关键字?

答案:transient关键字

有些属性需要序列化，而其他属性不需要被序列化，打个比方，如果一个用户有一些敏感信息（如密码，银行卡号等），为了安全起见，不希望在网络操作（主要涉及到序列化操作，本地序列化缓存也适用）中被传输，这些信息对应的变量就可以加上transient关键字。

4:读取文件

File file = new File("d:/spring.txt");  
try {  
 FileReader reader = new FileReader(file);  
 BufferedReader buffered = new BufferedReader(reader);  
 String data = null;  
while((data = buffered.readLine())!=null){  
 System.out.println(data);  
}   
} catch (FileNotFoundException e) {  
 e.printStackTrace();  
} catch (IOException e) {  
 e.printStackTrace();  
}

5： FileInputStream  & FileOutputStream   协同完成文件复制(不会乱码)

public static void main(String [] args){

File origin = new File("d:/helloWorld.txt");//原始文件

if (!origin.exists()){

try {

origin.createNewFile();

} catch (IOException e) {

e.printStackTrace();

}

}

File destination = new File("d:/helloWorld4.txt");//目的文件

InputStream in = null;

OutputStream out = null;

try {

in = new FileInputStream(origin);

out = new FileOutputStream(destination);

byte [] buff = new byte[1024];

int len = 0;

while ((len=in.read(buff))!=-1){

out.write(buff,0,len);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (in != null){

in.close();

}

if (out != null){

out.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

6：键盘输入，控制台输出功能

public static void main(String[]args){

BufferedReader buff = null;

BufferedWriter bufferedWriter = null;

String line = null;

try {

buff = new BufferedReader(new InputStreamReader(System.in,"utf-8"));

bufferedWriter = new BufferedWriter(new OutputStreamWriter(System.out,"utf-8"));

while ((line=buff.readLine())!=null){

bufferedWriter.write(line);

  bufferedWriter.flush(); //一定要刷新

} catch (IOException e) {

e.printStackTrace();

} finally {

if (buff!=null){

try {

buff.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if (out!=null){

try {

out.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

7：FilterReader

 字符过滤输入流、本事是一个抽象类、为所有装饰类提供一个标准、只是简单重写了父类Reader的所有方法、要求子类必须重写核心方法、和提供具有自己特色的方法

例如现在有这样一个需求：读取一个文件里面的内容，将文件里的内容所有的小写字母都转换成大写的。这个时候我们就可以通过去扩展IO流去做。

10.public class UpCaseReader extends FilterReader {

12. public UppCaseReader(Reader in) {

13. super(in);

14. }

16. @Override

17. public int read(char[] cbuf, int off, int len) throws IOException {

18. int result = super.read(cbuf, off, len);

19. for(int i=off;i<len;i++){

20. if(cbuf[i]>='a'&&cbuf[i]<='z'){

21. cbuf[i] -= 32;

22. }

23. }

24. return result;

25. }

27.// @Override

28.// public int read() throws IOException {

29.// int result = super.read();

30.// if(result>='a'&&result<='z'){

31.// result-=32;

32.// }

33.// return result;

34.// }

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

37. Reader reader = new BufferedReader(new UppCaseReader(new FileReader(new File("src/com/unis/io/UppCaseReader.java"))));

38. int i =0;

39. while((i=reader.read())!=-1){

40. System.out.print((char)i);

41. }

42. }

43.}

是典型的装饰器模式的运用语句。每封装一次，就添加一个新的功能。

装饰模式：给一个类添加一些额外的职责，并且在添加这些额外的职责时不会控制该类的执行逻辑。 装饰模式以对客户端透明的方式扩展对象的功能，是继承关系的一个替代方案。

组成部分：   
抽象构件：原始的功能接口-----相当于i/o流里面InputStream/OutputStream和Reader/Writer。  
具体构件：具体的原始功能类----实现接口的类---------------相当于i/o里面的FileOutputStream和FileInputStream。  
装饰角色：持有具体构件类的对象，以便执行原有功能------接口的实现类，类里调用接口的对象 --------------相当于i/o里面的FilterOutputStream和FilterInputStream。  
具体装饰：具体扩展的功能在这里--------------------------------相当于i/o流里面的BufferedOutputStream和BufferedInputStream以及DataOutputStream和DataInputSrtream。

10：InputStreamReader会包含一个InputStream，从而可以将该输入字节流转换成字符流，代码例子：

InputStream inputStream = new FileInputStream("c:\\data\\input.txt");

Reader reader = new InputStreamReader(inputStream，"UTF-8");

int data = reader.read();

while(data != -1){

char theChar = (char) data;

data = reader.read();

}

reader.close();

OutputStreamWriter会包含一个OutputStream，从而可以将该输出字节流转换成字符流，代码如下：

OutputStream outputStream = new FileOutputStream("c:\\data\\output.txt");

Writer writer = new OutputStreamWriter(outputStream);

writer.write("Hello World");

writer.close();

11：BufferedReader能为字符输入流提供缓冲区，可以提高许多IO处理的速度。你可以一次读取一大块的数据，而不需要每次从网络或者磁盘中一次读取一个字节。特别是在访问大量磁盘数据时，缓冲通常会让IO快上许多。

|  |
| --- |
| Reader input = new BufferedReader(new FileReader("c:\\data\\input-file.txt"), 8 \* 1024); |

除了能够为输入流提供缓冲区以外，其余方面BufferedReader基本与Reader类似。BufferedReader还有一个额外readLine()方法，可以方便地一次性读取一整行字符。

BufferedWriter

|  |
| --- |
| Writer writer = new BufferedWriter(new FileWriter("c:\\data\\output-file.txt"), 8 \* 1024); |

为了更好地使用内置缓冲区的磁盘，同样建议把缓冲区大小设置成1024的整数倍。除了能够为输出流提供缓冲区以外，其余方面BufferedWriter基本与Writer类似。类似地，BufferedWriter也提供了writeLine()方法，能够把一行字符写入到底层的字符输出流中。值得注意是，你需要手动flush()方法确保写入到此输出流的数据真正写入到磁盘或者网络中。

注意：

以下的构造函数取文件名和一个布尔变量作为参数，布尔值表明你是想追加还是覆盖该文件。例子如下：

|  |  |
| --- | --- |
| 1 | Writer writer = new FileWriter("c:\\data\\output.txt", true); //appends to file |

|  |  |
| --- | --- |
| 3 | Writer writer = new FileWriter("c:\\data\\output.txt", false); //overwrites file |

同样，FileWriter不能指定编码，可以通过OutputStreamWriter配合FileOutputStream替代FileWriter