## 1 创建文件夹

当文件夹不存在时，若是只有一级，那么文件对象file.mkdir();若是多级文件夹，那么使用file.mkdirs();

## 2 获取某个路径下的文件

### 2.1 只有一层目录

dir.list():列出当前目录下的所有文件的名字，包括文件夹和文件

//获取文件夹中的文件名字:只有当前子目录  
/\*String[] fileNames = dir.list();  
if(fileNames!=null&&fileNames.length>0){  
 for (String fileName:fileNames) {  
 System.out.println(fileName);  
 }  
}\*/

## 2.2 遍历所有路径下的文件

需要获取所有的文件对象，并判断是否是文件夹，若是进行递归

//使用递归遍历所有路径下的文件：需要File对象  
File[] files = dir.listFiles()**;**if(files!=null&&files.length>**0**){  
 for(File file:files){  
 if(file.isDirectory()){  
 *listDirectory*(file)**;** }else{  
 System.*out*.println(file)**;** }  
 }  
}

## 3 随机读写数据RandomAccessFile

含有文件指针，可以随机的读写文件，也可以通过getFilePointer()方法查看指针的位置，该类的对象每次写入8位的数据，即一个字节，当写入int时，需要写4次，从高8位写道低8位，如：

//int有32位，每次写8位，所以每次得将最高位的8位转移到最低8位  
raf.write(i >>> **24**)**;**raf.write(i >>> **16**)**;**raf.write(i >>> **8**)**;**raf.write(i)**;**

当然，也可以直接写入某种类型

//也可以直接写入int  
raf.writeInt(i)**;**System.*out*.println("指针的位置："+raf.getFilePointer())**;**//可以直接写字节数组  
String str = "见"**;**byte[] gbks = str.getBytes("gbk")**;**raf.write(gbks)**;**System.*out*.println(raf.length())**;**System.*out*.println("指针的位置："+raf.getFilePointer())**;**

通过移动指针进行文件的读取

raf.seek(**0**)**;**//将指针移到0的位置  
byte[] data = new byte[(int)raf.length()]**;**raf.read(data)**;**System.*out*.println("读取的字符数组"+ Arrays.*toString*(data))**;**//转换为16进制  
for (byte b :data)  
 System.*out*.print(Integer.*toHexString*(b & **0xff**))**;**

## 4 编码问题

转换为字节序列时，若是我们不指定使用什么编码，则会使用项目默认的编码：

//转换为字节序列用的是项目的默认编码utf-8:三个字节表示一个中文  
byte[] data = "慕课ABC".getBytes()**;**//把字节转换为int使用16进制的方式，因为前24位不影响，所以使用&0xff舍弃  
for (byte b:data){  
 System.*out*.print(Integer.*toHexString*(b & **0xff**)+" ")**;**}  
System.*out*.println()**;**byte[] data1 = "慕课ABC".getBytes("utf-8")**;**for (byte b:data1){  
 System.*out*.print(Integer.*toHexString*(b & **0xff**)+" ")**;**}  
  
  
//java是双字节编码:不管是中文还是英文都是两个字符表示  
System.*out*.println()**;**byte[] data2 = "慕课ABC".getBytes("utf-16be")**;**for (byte b:data2){  
 System.*out*.print(Integer.*toHexString*(b & **0xff**)+" ")**;**}  
  
System.*out*.println()**;**

当你的字节序列是某种编码时，这个时候想把字节序列变成字符串，也需要用这种编码方式，否则会出现乱码

String str1 = new String(data2)**;**//用项目默认的编码：utf-8  
System.*out*.println(str1)**;**//乱码  
  
String str2 = new String(data2**,**"utf-16be")**;**System.*out*.println(str2)**;**

## 5 文件流读取时返回的值

当使用直接read()时，从该输入流读取一个字节的数据。

当使用read(bytes,0,bytes.length)时，读取的字节内容放在bytes数组中，返回的读取的字节数组的实际大小，bytes.length是用来读取的容器大小

在使用缓冲流写数据时，要记得刷新

## 6 BufferedReader、BufferedWriter和InputStreamReader、OutputStreamWriter的区别

InputStreamReader、OutputStreamWriter可以通过指定输入流的编码读取文件和写入文件，避免乱码

但是，BufferedReader、BufferedWriter只能识别项目中使用的编码，当项目使用的是utf-8，文件是别的编码时，识别出来的是乱码

## 7 序列化

可以通过重写writeObject和readObject自主序列化参数（序列化的类中用transient修饰）

在序列化过程中，虚拟机会试图调用对象类里的writeObject() 和readObject()，  
\* 进行用户自定义的序列化和反序列化，如果没有则调用ObjectOutputStream.defaultWriteObject()  
\* 和ObjectInputStream.defaultReadObject()。同样，在ObjectOutputStream和ObjectInputStream  
\* 中最重要的方法也是writeObject() 和 readObject()，递归地写出/读入byte。

private void writeObject(java.io.ObjectOutputStream s)  
 throws java.io.IOException{  
 s.defaultWriteObject()**;**//把jvm能默认序列化的元素进行序列化操作  
 s.writeInt(stuage)**;**//自己完成stuage的序列化  
}  
private void readObject(java.io.ObjectInputStream s)  
 throws java.io.IOException**,** ClassNotFoundException{  
 s.defaultReadObject()**;**//把jvm能默认反序列化的元素进行反序列化操作  
 this.stuage = s.readInt()**;**//自己完成stuage的反序列化操作  
}

## 8 附录上课笔记

java.io.File类用于表示文件（目录）  
File类只用于表示文件（目录）的信息（名称、大小等），不能用于文件内容的访问  
  
RandomAccessFile java提供的对文件内容的访问，既可以读文件，也可以写文件。  
RandomAccessFile支持随机访问文件，可以访问文件的任意位置  
  
(1)java文件模型  
 在硬盘上的文件是byte byte byte存储的,是数据的集合  
(2)打开文件  
 有两种模式"rw"(读写) "r"（只读)  
 RandomAccessFile raf = new RandomeAccessFile(file,"rw")  
 文件指针，打开文件时指针在开头 pointer = 0;  
(3) 写方法  
 raf.write(int)--->只写一个字节（后8位),同时指针指向下一个位置，准备再次写入  
(4)读方法  
 int b = raf.read()--->读一个字节  
（5）文件读写完成以后一定要关闭（Oracle官方说明）  
  
  
序列化与基本类型序列化  
1）将类型int 转换成4byte或将其他数据类型转换成byte的过程叫序列化  
 数据---->n byte  
2)反序列化  
 将n个byte 转换成一个数据的过程  
 nbyte ---> 数据  
3)RandomAccessFile提供基本类型的读写方法，可以将基本类型数据  
 序列化到文件或者将文件内容反序列化为数据  
 IO流(输入流、输出流)  
 字节流、字符流  
 1.字节流  
 1)InputStream、OutputStream  
 InputStream抽象了应用程序读取数据的方式  
 OutputStream抽象了应用程序写出数据的方式   
 2)EOF = End 读到-1就读到结尾  
 3)输入流基本方法  
 int b = in.read();读取一个字节无符号填充到int低八位.-1是 EOF  
 in.read(byte[] buf)   
 in.read(byte[] buf,int start,int size)  
4)输出流基本方法  
 out.write(int b) 写出一个byte到流，b的低8位  
 out.write(byte[] buf)将buf字节数组都写入到流  
 out.write(byte[] buf,int start,int size)  
   
 5)FileInputStream--->具体实现了在文件上读取数据  
 6)FileOutputStream 实现了向文件中写出byte数据的方法  
 7)DataOutputStream/DataInputStream  
 对"流"功能的扩展，可以更加方面的读取int,long，字符等类型数据  
 DataOutputStream  
 writeInt()/writeDouble()/writeUTF()  
  
 8)BufferedInputStream&BufferedOutputStream  
 这两个流类位IO提供了带缓冲区的操作，一般打开文件进行写入  
 或读取操作时，都会加上缓冲，这种流模式提高了IO的性能  
 从应用程序中把输入放入文件，相当于将一缸水倒入到另一个缸中:  
 FileOutputStream--->write()方法相当于一滴一滴地把水“转移”过去  
 DataOutputStream-->writeXxx()方法会方便一些，相当于一瓢一瓢把水“转移”过去  
 BufferedOutputStream--->write方法更方便，相当于一飘一瓢先放入桶中，再从桶中倒入到另一个缸中，性能提高了  
   
   
 2.字符流  
 1) 编码问题  
 2)认识文本和文本文件  
 java的文本(char)是16位无符号整数，是字符的unicode编码（双字节编码)  
 文件是byte byte byte ...的数据序列  
文本文件是文本(char)序列按照某种编码方案(utf-8,utf-16be,gbk)序列化为byte的存储结果  
3)字符流(Reader Writer)---->操作的是文本文本文件  
字符的处理，一次处理一个字符  
字符的底层任然是基本的字节序列  
字符流的基本实现  
 InputStreamReader 完成byte流解析为char流,按照编码解析  
 OutputStreamWriter 提供char流到byte流，按照编码处理   
   
 FileReader/FileWriter 只能识别项目的编码  
 字符流的过滤器  
 BufferedReader ---->readLine 一次读一行  
 BufferedWriter/PrintWriter ---->写一行   
   
   
3.对象的序列化，反序列化  
1)对象序列化，就是将Object转换成byte序列，反之叫对象的反序列化   
2)序列化流(ObjectOutputStream),是过滤流----writeObject  
 反序列化流(ObjectInputStream)---readObject  
  
3)序列化接口(Serializable)  
 对象必须实现序列化接口 ，才能进行序列化，否则将出现异常  
 这个接口，没有任何方法，只是一个标准  
   
4) transient关键字  
 private void writeObject(java.io.ObjectOutputStream s)  
 throws java.io.IOException  
 private void readObject(java.io.ObjectInputStream s)  
 throws java.io.IOException, ClassNotFoundException  
   
 分析ArrayList源码中序列化和反序列化的问题  
   
5)序列化中 子类和父类构造函数的调用问题