# Java的IO操作

## 1、File类

1）一个File类的对象，表示了磁盘上的文件或目录。

2）File类提供了与平台无关的方法来对磁盘上的文件或目录进行操作。

3）方法：

getName()：返回文件名或目录名。

getPath()：如果是相对路径就返回相对路径，如果是绝对路径就返回绝对路径。

getAbsolutePath()：返回绝对路径。

getParent()：如果是相对路径就返回null，如果是绝对路径就返回上级目录。

deleteOnExit()：在程序退出时将文件或目录删除，常用于创建临时文件。

createTempFile(String prefix, String suffix)：静态方法，用于在缺省的临时文件目录（Temp环境变量指定目录）下创建一个文件并返回该文件，该文件名称以prefix开头以suffix为后缀中间自动添加一些序号，之后调用该文件的deleteOnExit() 方法以便于程序退出时将该临时文件自动删除。

createTempFile(String prefix, String suffix, File directory)：在指定目录下创建一个文件。

4）File类不能处理读写操作。

## 2、IO流的分类

1）按流的方向分：输入流和输出流（站在内存的角度看）

2）按流的数据分：字节流（Inputstream、Outputstream）和字符流（Reader、Writer）（只有纯文本可以使用字符流）

3）按流的功能分：节点流和处理流（节点流是从特定的地方读写的流类，例如：磁盘或一块内存区域；处理流用于包装节点流，用现有的节点流来构造，以优化节点流的某些功能，提高效率；处理流也可以当做节点流）

## 3、节点流

### 1）字节流

（1）FileInputStream和FileOutputStream：读写文件。在构造FileOutputStream时，文件已经存在，则默认覆盖这个文件，可以在构造时选择追加模式。

（2）PipedInputStream和PipedOutputStream：管道流，用于线程间的通信。一个线程的PipedInputStream对象从另一个线程的PipedOutputStream对象读取输入。管道输入流和管道输出流必须关联在一起才能使用，可以通过构造时关联，也可以通过调用connect() 方法关联。

示例代码：

**public class TestPipedStream**

**{**

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

**{**

**PipedOutputStream pos=new PipedOutputStream();**

**PipedInputStream pis=new PipedInputStream();**

**try**

**{**

**pos.connect(pis);**

**new Producer(pos).start();**

**new Consumer(pis).start();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**class Producer extends Thread**

**{**

**private PipedOutputStream pos;**

**public Producer(PipedOutputStream pos)**

**{**

**this.pos=pos;**

**}**

**public void run()**

**{**

**try**

**{**

**pos.write("你好，我是Tom!".getBytes());**

**pos.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

**class Consumer extends Thread**

**{**

**private PipedInputStream pis;**

**public Consumer(PipedInputStream pis)**

**{**

**this.pis=pis;**

**}**

**public void run()**

**{**

**try**

**{**

**Thread.sleep(1); //确保生产者已经写入**

**byte[] buf=new byte[100];**

**int len=pis.read(buf);**

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

**pis.close();**

**}**

**catch(Exception e)**

**{**

**e.printStackTrace();**

**}**

**}**

**}**

3）ByteArrayInputStream和ByteArrayOutputStream：读写内存中的字节数组。ByteArrayInputStream与FileInputStream操作类似，构造时传入一个字节数组，然后调用read() 方法读取即可。而ByteArrayOutputStream与FileOutputStream不同，构造时无需传入字节数组，调用write() 方法时首先写入ByteArrayOutputStream内置的一个字节数组缓存中，然后调用toByteArray() 会返回刚刚写入的内容。（字节数组流可以不关闭，其close() 方法是空实现）

示例代码：

**class Test**

**{**

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

**{**

**ByteArrayInputStream bis = new ByteArrayInputStream("qwe".getBytes());**

**byte[] b = new byte[3];**

**bis.read(b);**

**System.out.println(new String(b)); //qwe**

**ByteArrayOutputStream bos = new ByteArrayOutputStream();**

**bos.write("abc".getBytes());**

**b = bos.toByteArray();**

**System.out.println(new String(b)); //abc**

**bos.write("123".getBytes());**

**b = bos.toByteArray();**

**System.out.println(new String(b)); //abc123**

**}**

**}**

### 2）字符流

FileReader和FileWriter：读写文件，分别继承自InputStreamReader和OutputStreamWriter。

## 4、处理流

### 1）字节流

（1）BufferedInputStream和BufferedOutputStream：提供带缓冲的读写，提高了读写的效率。在使用BufferedOutputStream时，要注意输出缓冲区的问题（缓冲区已满或者调用flush()、close() 方法均可将缓冲区内容写入磁盘）。

示例代码：

**//复制文件**

**class FileUtils**

**{**

**public static void copyFile(String srcPath, String destPath) throws IOException**

**{**

***copyFile*(new File(srcPath), new File(destPath));**

**}**

**public static void copyFile(File srcFile, File destDir) throws IOException**

**{**

**if (!srcFile.exists())**

**throw new FileNotFoundException("srcFile is not exist");**

**File destFile = new File(destDir, srcFile.getName());**

**InputStream in = new BufferedInputStream(new FileInputStream(srcFile));**

**OutputStream out = new BufferedOutputStream(new FileOutputStream(destFile));**

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

**int len;**

**while ((len = in.read(buf)) != -1)**

**{**

**out.write(buf, 0, len);**

**}**

**out.close();**

**in.close();**

**}**

**}**

（2）DataInputStream和DataOutputStream：提供了读写Java中的基本数据类型的功能。读写顺序必须一致。

示例代码：

**class Test**

**{**

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

**{**

**FileOutputStream fos=new FileOutputStream("1.txt");**

**BufferedOutputStream bos=new BufferedOutputStream(fos);**

**DataOutputStream dos=new DataOutputStream(bos);**

**byte b=3;**

**int i=78;**

**String str = "中国";**

**char ch='a';**

**float f=4.5f;**

**dos.writeByte(b);**

**dos.writeInt(i);**

**dos.writeUTF(str);**

**dos.writeChar(ch);**

**dos.writeFloat(f);**

**dos.close();**

**FileInputStream fis=new FileInputStream("1.txt");**

**BufferedInputStream bis=new BufferedInputStream(fis);**

**DataInputStream dis=new DataInputStream(bis);**

**System.out.println(dis.readByte());**

**System.out.println(dis.readInt());**

**System.out.println(dis.readUTF());**

**System.out.println(dis.readChar());**

**System.out.println(dis.readFloat());**

**dis.close();**

**}**

**}**

3）PrintStream：打印流，System.in、System.err都是PrintStream类型。

### 2）字符流

（1）BufferedReader和BufferedWriter：提供带缓冲的读写，提高了读写的效率。

示例代码：

**//复制纯文本文件**

**public class FileUtils**

**{**

**public static void copyTextFile(String srcPath, String destPath) throws IOException**

**{**

**copyTextFile(new File(srcPath), new File(destPath));**

**}**

**public static void copyTextFile(File srcFile, File destDir) throws IOException**

**{**

**if (!srcFile.exists())**

**throw new FileNotFoundException("srcFile is not exist");**

**File destFile = new File(destDir, srcFile.getName());**

**Reader reader = new BufferedReader(new FileReader(srcFile));**

**Writer writer = new BufferedWriter(new FileWriter(destFile));**

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

**int len;**

**while ((len = reader.read(buf)) != -1)**

**{**

**writer.write(buf, 0, len);**

**}**

**writer.close();**

**reader.close();**

**}**

**}**

（2）InputStreamReader和OutputStreamWriter：将字节流转换为字符流。在转换时可以指定字符集，所以常用这种方式弥补FileReader和FileWriter不能修改默认字符集的缺陷。

**Reader reader = new BufferedReader(**

**new InputStreamReader(**

**new FileInputStream(new File("e:/1.txt")), "GBK"));**

## 5、RandomAccessFile类

1）RandomAccessFile类同时实现了DataInput和DataOutput接口，提供了对文件随机存取的功能，利用这个类可以在文件的任何位置读取或写入数据。

2）RandomAccessFile类提供了一个文件指针，用来标志要进行读写操作的下一数据的位置。

3）示例代码：

**public class TestRandomFile**

**{**

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

**{**

**Student s1=new Student(1,"zhangsan",98.5);**

**Student s2=new Student(2,"lisi",96.5);**

**Student s3=new Student(3,"wangwu",78.5);**

**RandomAccessFile raf=new RandomAccessFile("student.txt","rw");**

**s1.writeStudent(raf);**

**s2.writeStudent(raf);**

**s3.writeStudent(raf);**

**Student s=new Student();**

**raf.seek(0);**

**for(long i=0;i<raf.length();i=raf.getFilePointer())**

**{**

**s.readStudent(raf);**

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

**}**

**raf.close();**

**}**

**}**

**class Student**

**{**

**int num;**

**String name;**

**double score;**

**public Student()**

**{**

**}**

**public Student(int num,String name,double score)**

**{**

**this.num=num;**

**this.name=name;**

**this.score=score;**

**}**

**public void writeStudent(RandomAccessFile raf) throws IOException**

**{**

**raf.writeInt(num);**

**raf.writeUTF(name);**

**raf.writeDouble(score);**

**}**

**public void readStudent(RandomAccessFile raf) throws IOException**

**{**

**num=raf.readInt();**

**name=raf.readUTF();**

**score=raf.readDouble();**

**}**

**@Override**

**public String toString()**

**{**

**return "Student [num=" + num + ", name=" + name + ", score=" + score**

**+ "]";**

**}**

**}**

## 6、对象序列化

1）将对象转换为字节流保存起来，并在日后还原这个对象，这种机制叫做对象序列化。

将一个对象保存到永久存储设备上称为持续性。

2）一个对象要想能够实现序列化，必须实现Serializable接口或Externalizable接口。

3）当一个对象被序列化时，只保存对象的非静态成员变量，不能保存任何的成员方法和静态的成员变量。

4）如果一个对象的成员变量是一个对象，那么这个对象的数据成员也会被保存。如果一个可序列化的对象包含对某个不可序列化的对象的引用，那么整个序列化操作将会失败，并且会抛出一个NotSerializableException。我们可以将这个引用标记为transient，那么对象仍然可以序列化，可以使用transient关键字禁止对象的某些成员变量被序列化。

5）在反序列化时并不会调用对象的任何构造方法，仅仅是根据先前保存的对象的状态信息在内存中重新还原该对象。

6）ObjectOutputStream和ObjectInputStream：字节处理流，用于读写对象，完成对象序列化。

7）serialVersionUID：Java的序列化机制是通过判断类的serialVersionUID来验证版本一致性的。在进行反序列化时，JVM会把传来的字节流（也就是序列化后的对象数据）中类的serialVersionUID与本地相应实体类的serialVersionUID进行比较，如果相同就认为是一致的，可以进行反序列化，否则就会出现序列化版本不一致的异常。

总之，如果序列化时对象类的的serialVersionUID与反序列化时类的serialVersionUID不一致，一定会报错；如果序列化时对象类的的serialVersionUID与反序列化时类的serialVersionUID一致，则一定能成功，就算反序列化时类的一些属性减少或增加也能反序列化成功，减少的属性会自动忽略，而增加的属性会进行默认初始化。

注意：若类的serialVersionUID不明确指定的话编译器会根据类名、接口名、成员方法及属性等自动生成，此时在反序列化时若减少或增加类的属性会导致serialVersionUID改变，从而使反序列化失败。

8）示例代码：

**public class TestObjectSerial**

**{**

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

**{**

**Employee e1=new Employee("zhangsan",25,3000.50);**

**Employee e2=new Employee("lisi",24,3200.40);**

**Employee e3=new Employee("wangwu",27,3800.55);**

**FileOutputStream fos=new FileOutputStream("employee.txt");**

**ObjectOutputStream oos=new ObjectOutputStream(fos);**

**oos.writeObject(e1);**

**oos.writeObject(e2);**

**oos.writeObject(e3);**

**oos.close();**

**FileInputStream fis=new FileInputStream("employee.txt");**

**ObjectInputStream ois=new ObjectInputStream(fis);**

**Employee e;**

**for(int i=0;i<3;i++)**

**{**

**e=(Employee)ois.readObject();**

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

**}**

**ois.close();**

**}**

**}**

**class Employee implements Serializable**

**{**

**private static final long serialVersionUID = 1L;**

**String name;**

**int age;**

**transient double salary;**

**public Employee(String name,int age,double salary)**

**{**

**this.name=name;**

**this.age=age;**

**this.salary=salary;**

**}**

**@Override**

**public String toString()**

**{**

**return "Employee [name=" + name + ", age=" + age + ", salary=" + salary**

**+ "]";**

**}**

**}**

## 7、编码与解码

1）计算机中只能存储二进制数，将字符转换为二进制数的过程称为编码，将二进制数转换为字符的过程称为解码。

2）字符集：

返回JVM支持的所有字符集：

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

**{**

**Map<String, Charset> map = Charset.availableCharsets();**

**Set<String> set = map.keySet();**

**Iterator<String> it = set.iterator();**

**while (it.hasNext())**

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

**}**

返回JVM缺省的字符集：

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

**{**

**Properties p = System.getProperties();**

**System.out.println(p.getProperty("file.encoding"));**

**}**

3）Java的字符和字符串在内存中都采用unicode编码。

4）字符（String）→ 字节（byte[]）：编码过程

字节（byte[]）→ 字符（String）：解码过程

**String str = "中国"; //内存中使用unicode编码**

**//采用默认字符集编码**

**byte[] bytes = str.getBytes();**

**//采用默认字符集解码**

**System.out.println(new String(bytes)); //中国**

**//采用指定字符集编码**

**bytes = str.getBytes("gbk");**

**//采用默认字符集解码**

**System.out.println(new String(bytes)); //乱码**

**//采用指定字符集解码**

**System.out.println(new String(bytes, "gbk")); //中国**

## 8、理解

1）JVM默认的字符集是由系统平台决定的，在中国默认为GBK，eclipse工作目录的默认字符集可以在Window 🡪 Properties 🡪 General 🡪 Workspace 🡪 Text file encoding 下进行修改。

2）字符流是由字节流实现的（通过InputStreamReader和OutputStreamWriter转换），将文件字节流（FileInputStream和FileOutputStream）以默认字符集进行编码解码就变成了文件字符流（FileReader和FileWriter）。当用文件字符流对文件进行读写操作时，文件编码必须和JVM默认字符集一致，例如：一个纯文本文件以utf-8编码，将eclipse的工作目录的默认字符集改为utf-8则可以正常读取，但是将源码在DOS窗口中进行编译运行就会出现乱码（此时默认字符集为GBK）。为了弥补这个缺陷，可以采用文件字节流来构造InputStreamReader和OutputStreamWriter，构造时指定字符集即可。

3）Java源文件（.java）的编码方式应该与JVM默认的字符集一致，因为JVM将源文件编译为字节码文件时会使用其默认的字符集。在使用javac命令编译源文件时可以添加 –encoding参数指定字符集，如：javac -encoding utf-8 Test.java。

4）英文字符的编码解码方式在任何字符集下都相同。所以如果一个文本文件是纯英文的就无需考虑字符集问题。