1、File 类的理解

1）File 类定义在 **java.io 包**下。

2）**一个 File 对象**代表硬盘或网络中**可能存在**的**一个文件或者文件目录**（俗称文件夹），与平台无关。

3）File 能**新建、删除、重命名**文件和目录，但 File **不能访问文件内容**本身。如果需要访问文件内容，则**要使用输入/输出流(I/O 流)**，**File 对象**可以**作为参数**传递给**流的构造器**。

4）想要在 Java 程序中表示**一个真实存在的文件或目录**，那么必须有**一个 File 对象**，但是 Java 程序中的**一个 File 对象，可能没有一个真实存在的**文件或目录。

2、构造器（路径，路径拼接，文件对象+路径）

1）**public File(String pathname)**：以 pathname 为路径创建 File 对象，可以是**绝对路径或者相对路径**，如果 pathname 是相对路径，则**默认的当前路径在系统属性 user.dir 中**存储。

```
String pathname = "D:\\aaa\\bbb.txt";
File file = new File(pathname);
```

2）**public File(String parent, String child)**：以 parent 为**父路径** child 为**子路径**创建 File 对象。

```
String parent = "d:\\aaa";
String child = "bbb.txt";
File file3 = new File(parent, child);
```

3）**public File(File parent, String child)**：根据一个**父 File 对象**和**子文件路径**创建 File 对象。

```
File parentDir = new File("d:\\aaa");
String childFile = "bbb.txt";
File file4 = new File(parentDir, childFile);
```

【注意】

1>无论该路径下是否存在文件或者目录，都**不影响** File 对象的创建。

2>windows 的路径分隔符使用“\”，而 Java 程序中的“\”表示转义字符，所以在 Windows 中表示路径，**需要用“\\”**。或者**直接使用“/”**也可以，Java 程序支持将“/”当成平台无关的路径分隔符。或者**直接使用 File.separator 常量值**表示。如：

```
File file2 = new File("d:" + File.separator + "atguigu" + File.separator + "info.txt");
```

3>当构造路径是绝对路径时，那么 getPath 和 getAbsolutePath 结果一样；当构造路径是相对路径时，那么 getAbsolutePath 的路径 = user.dir 的路径 + 构造路径。

4>关于路径：绝对路径：从**盘符(Linux:/)开始的路径**，这是一个完整的路径。相对路径：**相对于项目目录的路径**，这是一个便捷的路径，开发中经常使用。

5>IDEA 中，**main 方法中的文件**的相对路径，是**相对于"当前工程"project**；IDEA 中，**单元测试方法**中的文件的相对路径，是**相对于"当前 module"**。

3、常用方法

1）获取文件和目录基本信息

1>public String **getName()**：获取名称；

2>public String **getPath()**：获取路径；

3>public String **getAbsolutePath()**：获取绝对路径；

4>public File **getAbsoluteFile()**：获取**绝对路径表示的文件**；

5>public String **getParent()**：获取**上层文件目录路径**；若无，返回 null。

6>public long **length()**：获取**文件长度**（即：字节数）；**不能获取目录的长度**，需要做累加。

7>public long **lastModified()**：获取最后一次的修改时间，毫秒值；

如果 File 对象代表的文件或目录**存在**，则 File 对象**实例初始化时**，就**会用硬盘中对应文件或目录的属性信息**（例如，时间、类型等）为 File 对象的属性**赋值**，**否则**除了路径和名称，File 对象的其他属性将会**保留默认值**。

2）列出目录的下一级

1>public String[] **list()**：返回一个**String 数组**，表示该 File 目录中的**所有子文件或目录**。

2>public File[] **listFiles()**：返回一个 **File 数组**，表示该 File 目录中的**所有子文件或目录**。

3）File 类的重命名功能

public boolean **renameTo(File dest)**:把文件重命名为指定的文件路径。

例 file.renameTo(file2)要想返回 true，要求：file 必须存在，且 file2 必须不存在，且 file2 所在目录需要存在。

4）判断功能的方法

1>public boolean **exists()**：此 File 表示的文件或目录**是否实际存在**；

2>public boolean **isDirectory()**：此 File 表示的**是否为目录**；

3>public boolean **isFile()**：此 File 表示的**是否为文件**；

4>public boolean **canRead()**：判断**是否可读**；

5>public boolean **canWrite()**：判断**是否可写**；

6>public boolean **isHidden()**：判断**是否隐藏**。

【注意】不存在则全为 false。

5）创建、删除功能

1>public boolean **createNewFile()**：**创建文件**。若文件存在，则不创建，返回 false。

2>public boolean **mkdir()**：**创建文件目录**。如果**此文件目录存在，就不创建**了。如果此文件目录的**上层目录不存在**，也不创建。

3>public boolean **mkdirs()**：**创建文件目录**。如果**上层文件目录不存在，一并创建**。

4>public boolean **delete()**：删除文件或者文件夹 【注意】：① Java 中的**删除不走回收站**。② 要删除一个文件目录，请注意该文件目录内**不能包含**文件或者文件目录。

【例】判断指定目录下是否有后缀名为.jpg 的文件，如果有，就输出该文件名称。

```
File dir = new File("F:\\10-图片");
//方式 1：
String[] listFiles = dir.list();
for(String s : listFiles){
    if(s.endsWith(".jpg")){
    System.out.println(s);
    }
//方式 2：public String[] list(FilenameFilter filter)
String[] listFiles = dir.list(new FilenameFilter()
{
  @Override
  //name:即为子文件或子文件目录的名称
  public boolean accept(File dir, String name)
{       return name.endsWith(".jpg");
      }
  });
for(String s : listFiles){
   System.out.println(s);
}
```

4、Java IO 原理

1）Java 程序中，对于数据的输入/输出操作**以“流(stream)”的方式**进行，可以看做是一种数据的流动。

2）I/O 流中的 I/O 是 Input/Output 的缩写，用于**处理设备之间的数据传输**。如**读/写**文件，**网络通讯**等。

- 输入 input：读取外部数据（**磁盘、光盘等存储设备**的数据）到程序（**内存**）中。
- 输出 output：将程序（**内存**）数据输出到**磁盘、光盘等存储设备**中。

5、流的分类

1）按**数据的流向**不同分为：**输入流**和**输出流**。

- 输入流：把数据从其他设备上读取到内存中的流；以 **InputStream、Reader** 结尾。
- 输出流：把数据从内存 中写出到其他设备上的流；以 **OutputStream、Writer** 结尾。

2）按**处理数据单位**的不同分为：**字节流**（8bit）和**字符流**（16bit）。

- **字节流**：以字节为单位，读写数据的流；以 **InputStream、OutputStream 结尾**
- **字符流**：以字符为单位，读写数据的流；以 **Reader、Writer 结尾**

3）根据 **IO 流的角色**不同分为：**节点流**和**处理流**。

- **节点流**：直接从**数据源**或**目的地**读写数据
- **处理流**：**不直接连接**到数据源或目的地，而是**“连接”在已存在的流**（节点流或处理流）之上，通过对数据的处理为程序提供更为强大的读写功能。

【常用的节点流】

文件流： FileInputStream\FileOutputStrean、FileReader\FileWriter

字节/字符数组流： ByteArrayInputStream、ByteArrayOutputStream 、CharArrayReader、CharArrayWrite 对数组进行处理的节点流（对应的不再是文件，而是内存中的一个数组）。

【常用处理流】

缓冲流：增加缓冲功能，避免频繁读写硬盘，进而提升读写效率。

BufferedInputStream\BufferedOutputStream 、BufferedReader、BufferedWriter。

转换流：实现字节流和字符流之间的转换；InputStreamReader\OutputStreamReader。

对象流：提供直接读写 Java 对象功能 ObjectInputStream\ObjectOutputStream。

6、FileReader 与 FileWriter

1）java.io.FileReader 类用于读取**字符文件**，构造时使用系统默认字符编码和默认字节缓冲区。

1>**FileReader(File file)**：创建一个新的 FileReader，给定要读取的 **File 对象**。

2>**FileReader(String fileName)**：创建一个新的 FileReader ，给定要读取的**文件的名称**。

2）java.io.FileWriter 类用于写出**字符**到文件，构造时使用系统默认的字符编码和默认字节缓冲区。

1>**FileWriter(File file)**：创建一个新的 FileWriter，给定要读取的 **File 对象**。

2>**FileWriter(String fileName)**：创建一个新的 FileWriter，给定要读取的**文件的名称**。

3>**FileWriter(File file,boolean append)**：创建一个新的 FileWriter，指明是否在现有文件末尾**追加**内容。

3）执行步骤：

第 1 步：创建读取或写出的 File 类的对象；

第 2 步：创建输入流或输出流；

第 3 步：具体的读入或写出的过程；

读入 read(char[] cbuffer)；

写出 write(String str) / write(char[] cbuffer,0,len)

第 4 步：关闭流资源，避免内存泄漏。

4）【注意】

1>因为涉及到流资源的关闭操作，所以出现异常的话，**需要使用 try-catch-finally** 的方式来处理异常；

2>对于输入流来讲，要求 File 类的对象对应的物理磁盘上的**文件必须存在**。否则，**会报 FileNotFoundException**；

3>对于输出流来讲，File 类的对象对应的物理磁盘上的**文件可以不存在**。

> 如果此**文件不存在**，则在输出的过程中，会**自动创建此文件**，并**写出数据**到此文件中。

> 如果**此文件存在**，①使用 FileWriter(File file)或 FileWriter(File file,false)输出数据过程中，**会新建同名文件**对现有的文件**进行覆盖**。②使用 FileWriter(File file,true)输出数据过程中，会在现有文件的**末尾追加写出**内容。

7、关于 flush（刷新）

因为**内置缓冲区的原因**，如果 **FileWriter 不关闭输出流，无法写出字符到**文件中。但**关闭**的流对象**无法继续写出**数据。如果我们既想写出数据又想继续使用流，就需 **flush()方法**。

- **flush()**：刷新缓冲区，流对象可以继续使用。
- **close()**：先刷新缓冲区，然后通知系统释放资源；流对象不可以再被使用了。

注意：即便是 flush()方法写出了数据，操作的最后还是要调用 close 方法，释放系统资源。

8、FileInputStream 与 FileOutputStream

1）java.io.FileInputStream 类是文件输入流，从文件中读取字节。

1>**FileInputStream(File file)**：通过打开与实际文件的连接来创建一个 FileInputStream，该文件由文件系统中的 **File 对象 file 命名**。

2>**FileInputStream(String name)**：通过打开与实际文件的连接创建一个 FileInputStream，该文件由文件系统中的**路径名 name 命名**。

2）java.io.FileOutputStream 类是文件输出流，用于将数据写出到文件。

1>public **FileOutputStream(File file)**：创建文件输出流，写出由**指定 File 对象表示的文件**。

2>public **FileOutputStream(String name)**：创建文件输出流，**指定的名称为写出文件**。

3>public **FileOutputStream(File file, boolean append)**：创建文件输出流，指明是否在现有文件**末尾追加**内容。

3）执行步骤：

第 1 步：创建读取或写出的 File 类的对象；

第 2 步：创建输入流或输出流；

第 3 步：具体的读入或写出的过程；

读入：read(byte[] buffer);

写出：write(byte[] buffer,0,len);

第 4 步：关闭流资源，避免内存泄漏。

4）【注意】

对于**字符流，只能用来操作**文本文件，不能用来处理非文本文件的；

对于**字节流**，通常是**用来处理**非文本文件的。但是，如果涉及到**文本文件的复制**操作，也可以使用字节流。

文本文件：.txt \.java\.c\.cpp\.py 等

非文本文件：.doc\.xls\.jpg\.pdf\.mp3\.avi 等

9、缓冲流

缓冲流的**基本原理**：在创建流对象时，**内部**会创建一个**缓冲区数组**（缺省使用 8192 个字节(8Kb)的缓冲区），通过缓冲区读写，**减少**系统**IO 次数**，从而**提高读写的效率**。

1）缓冲流要“**套接**”在相应的节点流之上，根据数据操作单位可以把缓冲流分为：

处理文本文件的**字符缓冲流**：

BufferedReader：read(char[] cBuffer) / String readLine()

BufferedWriter：write(char[] cBuffer,0,len) / write(String)

处理非文本文件的**字节缓冲流**：

BufferedInputStream：read(byte[] buffer);

BufferedOutputStream：write(byte[] buffer,0,len)

2）构造器

public **BufferedReader(Reader in)**

创建一个 新的字符型的缓冲输入流。

public **BufferedWriter(Writer out)**

创建一个新的字符型的缓冲输出流。

public **BufferedInputStream(InputStream in)**

创建一个新的字节型的缓冲输入流。

public **BufferedOutputStream(OutputStream out)**创建一个新的字节型的缓冲输出流。

3）字符缓冲流特有方法

BufferedReader：public String **readLine()** **读一行文字**，返回字符串。

BufferedWriter：public void **newLine()** **换行**，写一行行分隔符,由系统属性定义符号。

4）实现的步骤

第 1 步：创建 **File 的对象、流的对象**（包括文件流、缓冲流）

第 2 步：使用**缓冲流实现读取数据或写出数据的过程（重点）**

读取：int read(char[] cbuf/byte[] buffer)：每次将数据读入到 cbuf/buffer 数组中，并**返回**读入到数组中的字符的个数；

写出：void **write(String str)/write(char[] cbuf)**：将 str 或 cbuf 写出到文件中；

void write(byte[] buffer) 将 byte[]写到文件中；

第 3 步：关闭资源。

10、转换流（实现字节与字符间的转换）

1）引入原因

情况 1：使用 FileReader 读取项目中的文本文件。由于 IDEA 设置中针对项目设置了 **UTF-8 编码**，当读取 Windows 系统中创建的文本文件时，如果 Windows 系统默认的是 **GBK 编码**，则读入内存中会**出现乱码**。

情况 2：针对文本文件，现在使用**一个字节流**进行数据的**读入**，希望将数据显示在控制台上。此时针对**包含中文的文本数据**，可能会出现乱码。（中断了）

2）InputStreamReader

1>转换流 java.io.InputStreamReader，是 Reader 的子类，是从**字节流**到**字符流**的桥梁。它读取字节，并使用**指定的字符集**将其**解码**为字符。它的字符集可以由名称指定，也可以接受平台的默认字符集。

2>构造器

InputStreamReader(InputStream in):

创建一个使用**默认字符集**的字符流。

InputStreamReader(InputStream in, String charsetName):

创建一个**指定字符集**的字符流。

3）OutputStreamWriter

1> 转换流 java.io.OutputStreamWriter，是 Writer 的子类，是从**字符流**到**字节流**的桥梁。使用**指定的字符集**将字符**编码为**字节。它的字符集可以由名称指定，也可以接受平台的默认字符集。

2>构造器

OutputStreamWriter(OutputStream in):创建一个使用**默认字符集**的字符流。

OutputStreamWriter(OutputStream in,String charsetName): 创建一个**指定字符集**的字符流。

【补充】关于字符集的理解

在**存储的文件中**的字符：

1>**ascii**:主要用来存储 a、b、c 等英文字符和 1、2、3、常用的标点符号。**每个字符占用 1 个字节**。

2>iso-8859-1:了解，每个字符占用 1 个字节。向下兼容 ascii。

3>**gbk**:用来存储中文简体繁体、a、b、c 等英文字符和 1、2、3、常用的标点符号等字符。其中，**中文字符**使用 **2 个字节**存储的。向下兼容 ascii，意味着英文字符、1、2、3、标点符号**仍使用 1 个字节**。

4>**utf-8**:可以用来存储世界范围内主要的语言的所有的字符。使用 1-4 个不等的字节表示一个字符。其中，**中文字符**使用 **3 个字节**存储的。向下兼容 ascii，意味着英文字符、1、2、3、标点符号**仍使用 1 个字节**。

在**内存中**的字符：**一个字符(char)**占用 **2 个字节**；内存中用的字符集称为 **Unicode 字符集**。

11、数据流与对象流

1）数据流：DataOutputStream\DataInputStream

- DataOutputStream：允许应用程序将**基本数据类型、String 类型**的变量**写入输出流**中
- DataInputStream：允许应用程序以与机器无关的方式**从底层输入流中**读取**基本数据类型、String 类型**的变量。

2）数据流 DataInputStream 中的方法：

byte readByte()\short readShort()\int readInt()\ long readLong()\float readFloat()…

3）数据流的弊端：只支持 **Java 基本数据类型**和**字符串**的读写，而**不支持其它 Java 对象**的类型；而 ObjectOutputStream 和 ObjectInputStream **既**支持 Java 基本数据类型的数据读写，**又**支持 Java 对象的读写，所以重点介绍对象流 **ObjectOutputStream 和 ObjectInputStream**。

4）**对象流** ObjectOutputStream 与 ObjectInputStream

1>**ObjectOutputStream**：将 Java 基本数据类型和对象写入字节输出流中。通过在流中使用文件可以实现 Java 各种基本数据类型的数据以及对象的持久存储。

2>**ObjectInputStream**：ObjectInputStream 对以前使用 ObjectOutputStream 写出的基本数据类型的数据和对象进行读入操作，**保存在内存中**。

12、对象的序列化机制是什么

对象序列化机制允许把内存中的 **Java 对象**转换成平台无关的**二进制流**，从而允许把这种二进制流**持久地保存在磁盘**上，或通过**网络**将这种**二进制流传输**到另一个网络节点。当其它程序获取了这种二进制流，就**可以恢复成原来的 Java 对象**。

1>**序列化过程**：用**一个字节序列**可以**表示一个对象**，该字节序列**包含该对象的**类型和对象中存储的属性等**信息**。字节序列写出到文件之后，相当于文件中持久保存了一个对象的信息。

2>**反序列化过程**：该字节序列还可以从文件中读取回来，**重构对象**，对它进行反序列化。对象的数据、对象的类型和对象中存储的数据信息，都可以用来在内存中创建对象。

13、序列化机制的重要性

序列化是 RMI（Remote Method Invoke、**远程方法调用**）**过程的参数和返回值**都**必须实现**的机制，而 RMI 是 JavaEE 的基础。因此序列化机制是 **JavaEE 平台的基础**。

序列化的**好处**，在于可**将任何实现了 Serializable 接口的对象**转化为**字节数据**，使其在保存和传输时**可被还原**。

14、序列化机制实现原理

- 序列化：用 ObjectOutputStream 类保存基本类型数据或对象的机制。方法为：

public final void **writeObject (Object obj)**：将指定的对象写出。

- 反序列化：用 ObjectInputStream 类读取基本类型数据或对象的机制。方法为：

public final Object **readObject ()**:读取一对象。

15、自定义类要想实现序列化机制，需要满足：

1）自定义类需要**实现接口：Serializable**

2）要求自定义类**声明一个全局常量**：

static final long serialVersionUID = 42234234L;

用来**唯一的标识当前的类**。

3）要求自定义类**各个属性必须**是**可序列化**的。

1> 对于**基本数据类型**的属性：**默认**就是可以序列化的；

2> 对于**引用数据类型**的属性：要求**实现 Serializable 接口**。

【注意】

1>如果有一个属性**不需要可序列化**的，则该属性必须注明是瞬态的，使用 **transient 关键字**修饰。

2>**静态（static）变量**的值**不会序列化**。因为静态变量的值不属于某个对象。

16、反序列化失败问题

问题 1：对于 JVM 可以反序列化对象，它必须是能够**找到 class 文件的类**。如果找不到，则抛出 ClassNotFoundException 异常。

问题 2：当 JVM 反序列化对象时，能找到 class 文件，但是 **class 文件在序列化对象之后发生了修改**，那么反序列化操作也会失败，抛出一个 **InvalidClassException 异常**。发生这个异常的原因如下：1>该类的**序列版本号**与从流中读取的类描述符的版本号**不匹配**；2>该类**包含未知数据类型**。

【注意】如果**不声明全局常量 serialVersionUID，系统会自动声明**生成一个针对于当前类的 serialVersionUID。此时如果**修改此类的话，会导致 serialVersionUID 变化**，进而导致反序列化时，出现 InvalidClassException 异常。