本节涉及的相关类，都可以通过查看 API 手册了解！

1. 文件可认为是相关记录或放在一起的数据的集合

java.io.File 类在 java 中用来封装和访问文件和目录名对象，是文件和目录路径名的抽象表示形式。

注意事项

1、File 即可以描述文件也可以描述目录

2、文件的路径分割符号可以是”/”或者”\\”

3、 File 类不是对称的，允许获取文件属性，不允许改变文件属性

如果 Windows 系统中运行 java 代码，总是提示文件找不到，但是文件明明存在，要考虑是不是操作系统隐藏了文件的后缀名！！

文件/目录名操作	判断操作
public String getName() public String getPath() public String getAbsolutePath() public String getParent()	public boolean exists() public boolean canWrite() public boolean canRead() public boolean isFile() public boolean isDirectory() public boolean isAbsolute()
获取常规文件信息操作	设置和修改操作
public long lastModified() public long length()	public boolean delete() public void deleteOnExit() public boolean createNewFile() public setReadOnly() public boolean renameTo(File dest)
目录操作	
public boolean mkdir() public String[] list() public File[] listFiles()	

2. 流是指一连串流动的字符，是以先进先出方式发送信息的通道。

java I/O 流

Java 中把不同的数据源与程序间的数据传输都抽象表述为“流”(Stream)，java.io 包中定义了多种 I/O 流类型实现数据 I/O 功能。

相关术语

I/O(Input/Output)：输入/输出

数据源(Data Source)：数据流入的源头

数据宿(Data Sink)：数据流出的目的地

流的分类：

A. 根据流的方向：

输入流(Input Stream)---流入程序中的数据流，只能从中读取数据，而不能向其写出数据

输出流(Output Stream)---流出程序外的数据流，只能向其写出数据，而不能从中读取数据

B. 根据流的数据单位：

字节流(Byte Stream)---数据单位大小为一个字节(8 位)，字节流以字节为单位进行数据传输

字符流(Character Stream)---数据单位大小为一个字符(16 位)，字符流以字符为单位进行数据传输

C. 根据流与数据源的关系

节点流(Node Stream)---直接关联到数据源，该流是对数据源的直接封装

处理流(Processing Stream)---间接关联到数据源，该流是对已经存在流的连接和封装

3. InputStream/OutputStream

4. Reader/Writer

5. java.io.InputStream 此抽象类是所有**字节输入流类的超类** 。

方法列表	
int read()	读入单个字节
int read(byte[ ] b)	将字节读入数组
int read(byte[ ] b, int offset, int length)	讲字节读入数组的一部分
void close()	关闭该流并相关资源
int available()	返回输入流还能读取的有效字节数
long skip(long n)	跳过和丢弃此流中指定字节数
boolean markSupported()	判断此流是否支持 mark 操作
void mark(int readlimit)	标记流中的当前位置
void reset()	重置该流

java.io.OutputStream 此抽象类是所有**字节输出流类的超类** 。

方法列表	
void write(int b)	写出单个字节
void write(byte[ ] b)	写出字节数组

void write(byte[ ] b, int offset, int length)	写出字节数组一部分
void close()	关闭该流并释放相关资源。但要先刷新它
public void flush()	刷新该流的缓冲区

6. java.io.Reader 此抽象类是所有**字符输入流类的超类** 。

方法列表	
int read()	读入单个字符
int read(char[ ] cbuf)	将字符读入数组
int read(char[ ] cbuf, int offset, int length)	将字符读入数组的某一部分
void close()	关闭该流并释放关联资源
boolean ready()	判断是否准备读取此流
long skip(long n)	跳过指定字符
boolean markSupported()	判断此流是否支持 mark 操作
void mark(int readAheadLimit)	标记流中的当前位置
void reset()	重置该流

java.io.Writer 此抽象类是所有**字符输出流类的超类** 。

方法列表	
void write(int c)	写出单个字符
void write(char[] cbuf)	写出字符数组
void write(char[] cbuf, int offset, int length)	写出字符数组的某一部分
void write(String string)	写出字符串
void write(String string, int offset, int length)	写出字符串的某一部分
void close()	关闭该流并释放关联资源。但要先刷新它
void flush()	刷新该流的缓冲区

7. 操作文件流的步骤：

- 使用 File 类找到一个文件
- 通过 File 的对象去实例化字节流或字符流的子类
- 进行字节(字符)的读写操作
- 关闭文件流

注意事项：

Java.io.IOException ---几乎所有的流方法都会引发 IOException 异常

java.io.FileNotFoundException ----IOException 的子类，代表文件未找到异常。

File 类不会引起这样的异常，但是在基于路径名或 File 对象构建流对象的时候会引发此异常

读取结束 ---从输入流中读取数据，**如果已到达流的末尾，则返回 -1**

8. FileInputStream/FileOutputStream

用于对文件进行读入和写出操作。**操作的文件没有格式的限制，即可以是文本文件也可以是二进制文件。**

FileInputStream 构造方法
public FileInputStream(String name) throws FileNotFoundException
public FileInputStream(File file) throws FileNotFoundException
FileOutputStream 构造方法
public FileOutputStream(String name) throws FileNotFoundException
public FileOutputStream(String name, boolean append) throws FileNotFoundException
public FileOutputStream(File file) throws FileNotFoundException
public FileOutputStream(File file, boolean append) throws FileNotFoundException

FileReader/FileWriter

用于对文件进行读入和写出操作。**只能操作文本文件，不能操作二进制文件。**

FileReader 构造方法
public FileReader(String fileName) throws FileNotFoundException
public FileReader(File file) throws FileNotFoundException
FileWriter 构造方法
public FileWriter(String fileName) throws IOException
public FileWriter(String fileName, boolean append) throws IOException
public FileWriter(File file) throws IOException
public FileWriter(File file, boolean append) throws IOException

9. BufferedReader/BufferedWriter

用于缓冲读取和写入字符数据。

要特别注意：对于有缓冲池概念的类，使用 write 后，如果结果不如自己的预期，务必记得调用 close 方法或者 flush 方法刷新缓冲池，让其将数据写入！

BufferedReader 构造方法
public BufferedReader(Reader in)
public BufferedReader(Reader in, int sz)
BufferedReader 特有方法
public String readLine() throws IOException

用于读取一个文本行。遇到下列字符认为某行已终止：换行（'\n'）、回车（'\r'）或回车后+换行。返回的字符串，不包含任何行终止符，如果已到达流末尾，则返回 null
BufferedWriter 构造方法
public BufferedWriter(Writer out)
public BufferedWriter(Writer out, int sz)
BufferedWriter 特有方法
public void newLine() throws IOException
用于写入一个行分隔符。该符号由系统属性 line.separator 定义，并且不一定是换行符('\n')

10. InputStreamReader/OutputStreamWriter

可以完成字节流到字符流的转换，是字节流通向字符流的桥梁。

它使用指定的 charset 将字节编码和解码。字符集可以直接指定，或者可以接受平台默认的字符集。

InputStreamReader 构造方法
public InputStreamReader(InputStream in)
public InputStreamReader(InputStream in, String charsetName) throws UnsupportedEncodingException
OutputStreamWriter 构造方法
public OutputStreamWriter(OutputStream out)
public OutputStreamWriter(OutputStream out, String charsetName) throws UnsupportedEncodingException
特有方法
public String getEncoding()返回此流使用的字符编码的名称

11. PrintStream

增强的字节输出流，可以方便的输出各种数据类型的数据。

构造方法
public PrintStream(String fileName) throws FileNotFoundException
public PrintStream(String fileName, String encoding) throws FileNotFoundException, UnsupportedEncodingException
public PrintStream(File file) throws FileNotFoundException
public PrintStream(File file, String csn) throws FileNotFoundException, UnsupportedEncodingException
public PrintStream(OutputStream out)
public PrintStream(OutputStream out, boolean autoFlush)
public PrintStream(OutputStream out, boolean autoFlush, String encoding)
注意事项

1、PrintStream 永远不会抛出 IOException
2、指定了 autoFlush 为 true 的 PrintStream 将具备自动刷新缓冲区的功能。写入 byte 数组、调用 println 方法、写入换行符 ('\n') 时都会自动刷新输出缓冲区

PrintWriter

增强的字符输出流，此类实现了在 PrintStream 中的所有 print 方法。

构造方法
public PrintWriter(String fileName) throws FileNotFoundException
public PrintWriter(String fileName, String csn) throws FileNotFoundException,Un...Exception
public PrintWriter(File file) throws FileNotFoundException
public PrintWriter(File file, String csn) throws FileNotFoundException, Un…EncodingException
public PrintWriter(OutputStream out)
public PrintWriter(OutputStream out, boolean autoFlush)
public PrintWriter(Writer out)
public PrintWriter(Writer out, boolean autoFlush)
注意事项
1、PrintWriter 永远不会抛出 IOException
2、如果启用了自动刷新，则只有在调用 println、printf 或 format 的其中一个方法时才可能完成此操作，而不是每当正好输出换行符时才完成。
3、PrintStream 只能封装 OutputStream，而 PrintWriter 既可以封装 OutputStream 和 Writer。

12. DataInputStream/DataOutputStream

分别实现了 DataInput/DataOutput 接口，能够以一种与机器无关的方式，直接以 Java 的默认编码方式读入或者写出 Java 基本类型或 Sring 类型数据。主要应用于跨平台数据通信

构造方法
public DataInputStream(InputStream in)
public DataOutputStream(OutputStream out)
特有方法
public final Type readType() throws IOException
public final void writeType(Type type) throws IOException

13. CharArrayReader/CharArrayWriter

CharArrayReader 实现了一个可用作字符输入流的字符输入缓冲区

CharArrayWriter 实现了一个可用作字符输出流的字符输出缓冲区

构造方法
public CharArrayReader(char[] buf)
public CharArrayReader(char[ ] buf, int offset, int length)

public CharArrayWriter()
public CharArrayWriter(int initialSize)

14. 对象序列化

分析

为什么要序列化对象？

对象的持久化(Object Persistance)

长久保存一个对象的状态并在需要时获取该对象的信息以重新构造一个状态完全相同的对象

对象的序列化(Object Serialization)

通过写出对象的状态数据来记录一个对象的过程即对象的序列化

通常序列化需要：写出对象的状态信息，并遍历该对象对其他对象的引用，递归的序列化所有被引用到的其他对象，从而建立一个完整的序列化流.

序列化用途

A. 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

B. 在网络上传送对象的字节序列。

15. 序列化接口

一个对象需要序列化，其所属的类必须实现以下两种接口之一。

序列化接口没有方法或字段，仅用于标识可序列化的语义。

java.io.Serializable

java.io.Externalizable

serialVersionUID

与每个可序列化类相关联系序列类的序列号，该序列号在反序列化过程中用于验证序列化对象的发送者和接收者是否为该对象加载了兼容的序列化类。使用的时候需要注意：

A. 如果接收者加载的该对象的类的 serialVersionUID 与对应的发送者的类的 serialVersionUID 不同，则反序列化将会导致 InvalidClassException。

B. 序列化类未声明 serialVersionUID，序列化运行时将提供默认 serialVersionUID 值。

计算默认的 serialVersionUID 对类的详细信息具有较高的敏感性，根据编译器实现的不同可能不同，这样在反序列化过程中可能会导致 InvalidClassException。

16. 序列化流类

java.io 包中分别提供了对象的序列化和反序列化功能的流类

序列化：Java.io.ObjectOutputStream

反序列化：Java.io.ObjectInputStream

构造方法
public ObjectInputStream(InputStream in) throws IOException
public ObjectOutputStream(OutputStream out) throws IOException
序列化方法
public final Object readObject() throws IOException, ClassNotFoundException
public final void writeObject(Object obj) throws IOException

Serializable

标记性接口

static 属性

在对象序列化过程中，其所属类的 static 属性和方法代码不会被序列化处理

非 static 属性

对于个别**不希望被序列化的非 static 属性，可以在属性声明时使用 transient 关键字进行标明.**

例如：private transient String pass; // 密码

17. java.util.Scanner

一个可以使用正则表达式来解析基本类型和字符串的简单文本扫描器。它分隔符将其输入分解为标记，然后可以使用 nextXxx() 方法扫描输入。

构造方法	
public Scanner(String source) **public Scanner(InputStream source)** **public Scanner(File source) throws FileNotFoundException**	
功能方法	
public void close()	**关闭扫描器**
public Scanner useDelimiter(String pattern)	**设置分割符号**
public boolean hasNext() **public boolean hasNextLine()** **public boolean hasNextType()**	**判断扫描是否结束**
public String next() **public String nextLine()** **public long nextType()**	**扫描输入**

扫描输入的步骤

A. **使用输入流、文件或者字符串构造 Scanner 对象**

B. **设置分割符号，不设置则以空格作为分割符号**

C. **调用 Scanner 的一个 hasNextXxx()方法判断下一个特定的数据是否存在**

D. **如果 hasNextXxx()方法返回为 true，则调用 nextXxx()方法扫描输入该数据，且扫描标记指向下一个位置**

E. **扫描结束，关闭扫描输入对象.**

18. 标准输入输出

java.lang.System 类提供了对标准输入输出流、标准错误流的支持。该类为这些流提供了相应的静态属性

标准输入

System.in： InputStream 类的实例，此流对应于键盘输入。

可以通过 System 类的 public static void setIn(InputStream in)重写设置

标准输出

System.out： PrintStream 类的实例，此流对应于显示器输出。

可以通过 System 类的 public static void setOut(PrintStream out)重新设置

标准错误

System.err： PrintStream 类的实例，此流对应于显示器输出。

可以通过 System 类的 public static void setErr(PrintStream err)重写设置