## File

File对象可以是一个不存在的文件

可通过exists（） isdirectory（） isfile（）；进行判断

Window中路径使用’\’，在unix/linux 中分隔符用‘/’ File.separatorChar根据系统生成相应的分隔符

如果使用’\’ 需要转义 写为‘\\’, 两个 \ 写为\\\\

转义字符均使用\

常用方法 ：

CreatenewFile（）指定位置创建一个空文件 成功返回true

Mkdir（）指定位置创建目录，

Renameto（file dest） 重命名文件或文件夹 可以操作非空的文件夹，文件不同时相当于剪切，

Delete（） 删除文件或一个空文件夹，文件夹不为空测无法删除。

Getname（） 获取文件或文件夹的名称

getpath（） getabsolutePath（） 获取文件的绝对路径时尽量先判断文件是否存在（exist）

getparent（） 获取抽象路径名父目录的路径名 没有父类就返回null

list（） 返回目录下的文件或者目录名 对于文件操作返回null；

list （filenamefilter filter） 返回指定当前目录中符合过滤条件的子文件或子目录

listfiles（）返回目录下文件或者目录对象

listfiles（fileNameFilter filter） 返回符合过滤条件的子文件或目录 对于文件操作会返回null

## IO流

输入字节流 ：fileinputStream

读方法：read（）一次读一个字节：返回值为读的一个字节数据，读完返回-1；

Read (byte [] b) 返回值为b数组的存储数据长度(字节的长度)； 推荐

//Read（byte [] b,int off, int len）返回值为b数组的存储数据长度 off：起始存储位置 len：存储长度

推荐：

byte [] b = new byte [1024];

int len = 0;

while （ len = read（b）!= -1）{

new String (b, 0 , len);

}

输出字节流： fileoutputStream

写方法：writer（int B） 写一个字节

Writer （byte [] b） 使用缓冲调高效率

Writer （byte [] b，int off， int len） b是容器，off数组起始写位置 len写的长度

a.txt文本文件发现上述程序每运行一次，老的内容就会被覆盖掉。实现追加的功能使用FileOutputStream(File file, boolean append) 第二个参数，append - 如果为 true，则将字节写入文件末尾处，而不是写入文件开始处

推荐：

// 使用字节数组，当做缓冲区

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

**while** ((len = fis.read(byt)) != -1) {

fos.write(byt, 0, len);

}

字节流的异常处理：

在finally中：

**if** (write != **null**) {

**try** {

fis.close();

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

英文占一个字节 中文占两个字节

Java中char包含两个字节，可以存储一个汉字

输入输出字符流

输入字符流：filereader

1，int read()：

读取一个字符。返回的是读到的那个字符。如果读到流的末尾，返回-1.

2，int read(char[])：

将读到的字符存入指定的数组中，返回的是读到的字符个数，也就是往数组里装的元素的个数。如果读到流的末尾，返回-1.

3，close()

输出字符流：filewriter

默认的FileWriter方法新值会覆盖旧值，想要实现追加功能需要

使用如下构造函数创建输出流 append值为true即可。

1，write(ch): 将一个字符写入到流中。

2，write(char[]): 将一个字符数组写入到流中。

3，write(String): 将一个字符串写入到流中。

4，flush():刷新流，将流中的数据刷新到目的地中，流还存在。

5，close():关闭资源：在关闭前会先调用flush()，刷新流中的数据去目的地。然流关闭。

推荐使用：

**int** ch = -1;

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

**while** ((ch = reader.read(arr)) != -1) {

writer.write(arr,0,ch);

}

异常处理：

// 关闭流，注意一定要能执行到close()方法，所以都要放到finally代码块中

**try** {

**if** (reader != **null**) {

reader.close();

}

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

} **finally** {

**try** {

**if** (writer != **null**) {

writer.close();

}

} **catch** (Exception e) {

**throw** **new** RuntimeException(e);

}

}

}

## 序列流：SequenceInputStream

对多个流进行合并。对于其他输入流的逻辑串联，它从输入流的有序集合开始，并从第一个输入流开始读取，直到到达文件末尾，接着从第二个输入流读取，依次类推，直到到达包含的最后一个输入流的文件末尾为止。

## 对象的序列化与反序列化

对象的序列化和反序列化：将对象写入文件设备或在文件设备中读取出来。

ObjectOutput

writeObject(Object obj)

将对象写入底层存储或流。

ObjectInput

readObject()

读取并返回对象。

对象的序列化：objectoutputStream

序列化的对象必须实现serializable接口

序列化时对象包含引用型数据？

1. Java的序列化机制可以解决该类问题，当序列化一个对象时，Java的序列化机制会负责保存对象的所有关联的对象（就是对象图），反序列化时，也会恢复所有的相关内容。但必须对象中的引用类型也要实现serializable 接口；
2. 序列化不适用于静态变量，因为静态变量并不属于对象的实例变量的一部分。静态变量随着类的加载而加载，是类变量。由于序列化只适用于对象。
3. 成员变量标识为transienT，那么在序列化对象时，序列化就会跳过相应的引用类型。

serialVersionUID用于给类指定一个UID。该UID是通过类中的可序列化成员的数字签名运算出来的一个long型的值。

只要是这些成员没有变化，那么该值每次运算都一样。

定义方式：static final long serialVersionUID = 42L;

## Serializable

通过实现 java.io.Serializable 接口以启用其序列化功能

## Properties

可以和流关联的集合对象；不需要泛型，集合中的键值对都是String 类型

Map

|--Hashtable

|--Properties

Properties: 该类是一个Map的子类，提供了可以快速操作配置文件的方法

load() : 将文件设备数据装载为Map集合数据

get(key): 获取Map中的数据

getProperty()获取Map中的数据特有方法

案例：

/\*

\* 将配置文件中的数据通过流加载到集合中。

\*/

**public** **static** **void** loadFile() **throws** IOException {

// 1,创建Properties(Map)对象

Properties prop = **new** Properties();

// 2.使用流加载配置文件。

FileInputStream fis = **new** FileInputStream("c:\\qq.txt");

// 3。使用Properties 对象的load方法将流中数据加载到集合中。

prop.load(fis);

// 遍历该集合

Set<Entry<Object, Object>> entrySet = prop.entrySet();

Iterator<Entry<Object, Object>> it = entrySet.iterator();

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

Entry<Object, Object> next = it.next();

Object key = next.getKey();

Object value = next.getValue();

}

// 通过键获取指定的值

Object object = prop.get("jack");

System.*out*.println(object);

// 通过键修改值

prop.setProperty("jack", "888888");

// 将集合中的数据写入到配置文件中。

FileOutputStream fos = **new** FileOutputStream("c:\\qq.txt");

// 注释：

prop.store(fos, "yes,qq");

fos.close();

fis.close();

}

## 编码与解码

Java中的编码和解码 就是字符串（字符）和字节类型之间的变换；

解码时：

String(byte[],charset)指定码表