1. File类
2. 概述

File类是文件和目录路径名的抽象表示，即文件和目录通过File类封装成对象，可以方便地处理文件和目录。

对于File类而言，封装的并不是一个真实存在的文件，而是一个路径名，该路径名可能存在，也可能不存在。

1. 构造方法

方法名 说明

File(String pathname) 通过将给定的路径名字符串转换为抽象路径名来创建新的 File实例

File(String parent, String child) 从父路径名字符串和子路径名字符串创建新的 File实例

File(File parent, String child) 从父抽象路径名和子路径名字符串创建新的 File实例

1. 文件目录创建方法

方法名 说明

public boolean

createNewFile()

当具有该名称的文件不存在时，创建一个由该抽象路径名命名的新空

文件

public boolean mkdir() 创建由此抽象路径名命名的目录

public boolean mkdirs() 创建由此抽象路径名命名的目录，包括任何必需但不存在的父目录

1. 判断和获取方法
2. 判断方法

方法名 说明

public boolean isDirectory() 测试此抽象路径名表示的File是否为目录

public boolean isFile() 测试此抽象路径名表示的File是否为文件

public boolean exists() 测试此抽象路径名表示的File是否存在

1. 获取方法

方法名 说明

public String getAbsolutePath() 返回此抽象路径名的绝对路径名字符串

public String getPath() 将此抽象路径名转换为路径名字符串

public String getName() 返回由此抽象路径名表示的文件或目录的名称

public String[] list() 返回此抽象路径名表示的目录中的文件和目录的名称字符串数组

public File[] listFiles() 返回此抽象路径名表示的目录中的文件和目录的File对象数组

1. 删除方法

方法名 说明

public boolean delete() 删除由此抽象路径名表示的文件或目录

注意：一个目录中有内容，那么不能直接删除，要先将里面的内容删除，才能删除目录。

1. IO流
2. 概述

IO代表着输入输出，流代表着数据的传输。IO流用于处理设备间的数据传输问题，如文件复制，文件上传，文件下载。

1. 分类
2. 按数据的流向分

输入流：从磁盘输入到内存，即从磁盘读数据。

输出流：从内存输出到磁盘，即写数据到磁盘。

注意：记忆的时候把输入输出都和内存联系起来，输入流是输入到内存，输出流是从内存输出。

1. 按数据类型分

字节流

字符流

1. IO流的使用场景
2. 纯文本文件，优先使用字符流。
3. 图片，视频等二进制文件，优先字节流。
4. 不能确定文件类型，优先字节流。
5. 字节流
6. 字节流抽象基类
7. InputStream

字节输入流的所有类的超类

1. OutputStream

字节输出流的所有类的超类

字节流子类名的特点：子类的名称都是以父类名作为子类名的后缀。

1. 文件字节输出流

FileOutputStream，注意后缀名是OutputStream，是该类的父类名。

使用文件字节输出流写数据的步骤：

1. 创建字节输出流对象 (调用系统功能创建了文件，创建字节输出流对象，让字节输出流对象指向文件)
2. 调用字节输出流对象的写数据方法
3. 释放资源 (关闭此文件输出流并释放与此流相关联的任何系统资源)
4. 文件字节流写数据的问题
5. 字节流写数据实现换行

windows:\r\n

linux:\n

mac:\r

1. 实现追加写入

public FileOutputStream(String name,boolean append)

创建文件输出流以指定的名称写入文件。如果第二个参数为 true ，则字节将写入文件的末尾而不是开头

1. 文件字节输入流读数据
2. FileInputStream

创建文件字节输入流，从磁盘文件中读取数据。

1. 步骤

创建字节输入流对象

调用字节输入流对象的读数据方法

释放资源

1. read(byes[] b)方法

read方法返回的是实际读取的字节个数，当没有读取到数据时，返回-1。

 FileInputStream fis = new FileInputStream("myByteStream\\fos.txt");

    int by;

    /\*

      fis.read()：读数据

      by=fis.read()：把读取到的数据赋值给by

      by != -1：判断读取到的数据是否是-1

    \*/

    while ((by=fis.read())!=-1) {

      System.out.print((char)by);

   }

//释放资源

fis.close();

1. 字节缓冲流
2. 概述
3. BufferedOutputStream

缓冲输出流，当向磁盘中写入数据时，先将数据写入缓冲流中，再将缓冲流中存积的大量数据调用底层输出流写入磁盘。这样，调用底层输出流的次数减少了很多。

1. BufferedInputStream

缓冲输入流，同理，在底层输入流和内存中搭建一个缓冲输入流，减少调用底层输入流的次数。

1. 构造方法
2. BufferedOutputStream(OutputStream out)

创建字节缓冲输出流对象，参数是输出流对象。

1. BufferedInputStream(InputStream in)

创建字节缓冲输入流对象，参数是输入流对象。

1. 示例

 BufferedOutputStream bos = new BufferedOutputStream(new

                  FileOutputStream("myByteStream\\bos.txt"));

    //写数据

    bos.write("hello\r\n".getBytes());

    bos.write("world\r\n".getBytes());

    //释放资源

    bos.close();

    //字节缓冲输入流：BufferedInputStream(InputStream in)

    BufferedInputStream bis = new BufferedInputStream(new

                  FileInputStream("myByteStream\\bos.txt"));

    //一次读取一个字节数据

//    int by;

//    while ((by=bis.read())!=-1) {

//      System.out.print((char)by);

//    }

    //一次读取一个字节数组数据

    byte[] bys = new byte[1024];

    int len;

    while ((len=bis.read(bys))!=-1) {

      System.out.print(new String(bys,0,len));

   }

    //释放资源

bis.close();

1. 字符流
2. 概述

在字节流操作中，中文的表示很不方便，因此，有了字符流来帮助中文等其他字符的表示。

字符流= 字节流 + 编码表

中文的表示也是用几个字节拼成的，通过某种规则将几个字节表示成一个中文字符，这种规则就是编码表，将各种各样的字符用字节表示的规则。

1. 字符集

前面讲了，编码表是一种字符和字节对应的规则，那么，首先我们需要一个庞大的字符集来存储大量的字符。包括各国家文字、标点符号、图形符号、数字等。

1. ASCII 字符集

基于拉丁字母的一套电脑编码系统，用于显示现代英语，主要包括控制字符(回车键、退格、

换行键等)和可显示字符(英文大小写字符、阿拉伯数字和西文符号)

基本的ASCII字符集，使用7位表示一个字符，共128字符。ASCII的扩展字符集使用8位表示一个字符，共

256字符，方便支持欧洲常用字符。是一个系统支持的所有字符的集合，包括各国家文字、标点符号、

图形符号、数字等

1. GBXXX 字符集

GBK：最常用的中文码表。是在GB2312标准基础上的扩展规范，使用了双字节编码方案，共收录了

21003个汉字，完全兼容GB2312标准，同时支持繁体汉字以及日韩汉字等

1. Unicode 字符集

为表达任意语言的任意字符设计，是业界的一种标准，也称为统一码，标准万国码。最多使用4个字节的数字来表示字符，有三种编码方案，UTF-8,UTF-16,UTF-32。

1. 编码表
2. UTF-8编码

用来表示Unicode标准中任意字符，它是电子邮件、网页及其他存储或传送文字的应用

中，优先采用的编码。互联网工程工作小组（IETF）要求所有互联网协议都必须支持UTF-8编码。它使用

一至四个字节为每个字符编码

编码规则：

128个US-ASCII字符，只需一个字节编码

拉丁文等字符，需要二个字节编码

大部分常用字（含中文），使用三个字节编码

其他极少使用的Unicode辅助字符，使用四字节编码

1. 字符的编码与解码方法

方法名 说明

byte[] getBytes() 使用平台的默认字符集将该 String编码为一系列字节

byte[] getBytes(String charsetName) 使用指定的字符集将该 String编码为一系列字节

String(byte[] bytes) 使用平台的默认字符集解码指定的字节数组来创建字符串

String(byte[] bytes, String charsetName) 通过指定的字符集解码指定的字节数组来创建字符串

package File使用;  
  
import java.io.UnsupportedEncodingException;  
import java.util.Arrays;  
  
public class 字符流编码解码 {  
 public static void main(String[] args) throws UnsupportedEncodingException {  
 String s = "中国";  
 byte[] bys = s.getBytes("GBK");  
 String ss = new String(bys, "GBK");  
 System.*out*.println(ss);  
  
 }  
}

1. 字符流的编码与解码
2. InputStreamReader

输入流是为了读取磁盘中的数据，则InputStreamReader就是从字节流到字符流的桥梁。读取字节，使用指定的编码将字节解码为字符。

1. OutputStreamWriter

输出流是为了将数据写入磁盘中，则OutputStreamWriter是字符流到字节流的桥梁。

使用指定的编码将写入的字符编码为字节。

1. 构造方法

方法名 说明

InputStreamReader(InputStream in)

使用默认字符编码创建InputStreamReader对

象

InputStreamReader(InputStream in,String

chatset)

使用指定的字符编码创建InputStreamReader

对象

OutputStreamWriter(OutputStream out)

使用默认字符编码创建OutputStreamWriter对

象

OutputStreamWriter(OutputStream out,String

charset)

使用指定的字符编码创建OutputStreamWriter

对象

1. 刷新和关闭

方法名 说明

flush() 刷新流，之后还可以继续写数据

close() 关闭流，释放资源，但是在关闭之前会先刷新流。一旦关闭，就不能再写数据

1. 字符流写数据

package File使用;  
  
import java.io.FileNotFoundException;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.OutputStreamWriter;  
  
public class 字符流写 {  
 public static void main(String[] args) throws IOException {  
 OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream("Student\\1.txt"));  
  
 char[] chs = {'a', 'b', 'c', 'd'};  
 osw.write(chs);  
  
 osw.write("edf");  
  
 //写字符  
 osw.write(97);  
 osw.write(98);  
 osw.write(99);  
  
 osw.close();  
 }  
  
}

1. 字符流读数据

package File使用;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStreamReader;  
  
public class 字符流读 {  
 public static void main(String[] args) throws IOException {  
 InputStreamReader isr = new InputStreamReader(new FileInputStream("Student\\1.txt"));  
  
// int ch;  
// while((ch = isr.read()) != -1)  
// {  
// System.out.println((char)ch);  
// }  
 char[] chs = new char[1024];  
 int len;  
 while((len=isr.read(chs)) != -1)  
 {  
 System.*out*.println(new String(chs, 0, len));  
 }  
  
 isr.close();  
 }  
}

1. 字符缓冲流
2. 概述

与字节缓冲流一样，都是建立缓冲区。

1. BufferedWriter

设置字符缓冲区，将字符写入字符输出流。

1. BufferedReader

设置字符缓冲区，从字符输入流中读取字符。

1. 构造方法
2. BufferedWriter(Writer out)

创建字符缓冲输出流对象。

1. BufferedReader(Reader in)

创建字符缓冲输入流对象

package File使用;  
  
import java.io.\*;  
  
public class 字符缓冲 {  
 public static void main(String[] args) throws IOException {  
 BufferedWriter bw = new BufferedWriter(new FileWriter("Student\\1.txt"));  
  
 bw.write("hello\r\n");  
 bw.close();  
  
  
 BufferedReader br = new BufferedReader(new FileReader("Student\\1.txt"));  
 char[] chs = new char[1024];  
 int len;  
 while((len=br.read(chs)) != -1)  
 {  
 System.*out*.print(new String(chs, 0, len));  
 }  
 }  
}

1. 字符缓冲流特有方法
2. BufferedWriter中的newLine()

写入一个行分隔符，且该字符的具体定义是由系统的属性决定的。

1. BufferedReader中的readLine()

读一行文字，但不包括行终止符，如果到达末尾，则返回null。

BufferedWriter bw = new BufferedWriter(new

             FileWriter("myCharStream\\bw.txt"));

    //写数据

    for (int i = 0; i < 10; i++) {

      bw.write("hello" + i);

      //bw.write("\r\n");

      bw.newLine();

      bw.flush();

   }

    //释放资源

    bw.close();

    //创建字符缓冲输入流

    BufferedReader br = new BufferedReader(new

             FileReader("myCharStream\\bw.txt"));

    String line;

    while ((line=br.readLine())!=null) {

      System.out.println(line);

   }

br.close();

1. 复制文件的异常处理
2. try-catch-finally

try{

可能出现异常的代码

}catch(异常类名 变量名){

异常处理

}finally{

所有清除操作

}

1. JDK7改进

try(定义流对象){

可能出现异常的代码

}catch(异常类名 变量名){

异常处理

}

1. JDK9改进

定义输入流对象；

定义输出流对象；

try(输入流对象； 输出流对象){

可能出现异常代码

}catch(异常类名 变量名){

异常处理

}

1. 标准输入输出流
2. 标准输入流

System类中的静态成员变量

Public static final InputStream in;

标准输入流，通常该流对应于键盘输入或由主机环境或用户指定的

另一个输入源。

1. 标准输出流

System类中的静态成员变量

Public static final PrintStream out;

标准输出流，通常该流对应于显示输出或由主机环境或用户指定的

另一个输出目标。

通常我们所用的输出语句就是一个标准输出流。

System.out.println();

1. 字节与字符打印流
2. 字节打印流
3. 特点

只用于输出数据，不会抛出IOException异常。

1. 示例

PrintStream ps = new PrintStream(“1.txt”);

Ps.println(97);

Ps.close();

1. 字符打印流
2. 构造方法

方法名 说明

PrintWriter(String

fileName)

使用指定的文件名创建一个新的PrintWriter，而不需要自动执行刷新

PrintWriter(Writer

out, boolean

autoFlush)

创建一个新的PrintWriter out：字符输出流 autoFlush： 一个布尔值，如果

为真，则println ， printf ，或format方法将刷新输出缓冲区

1. 示例

PrintWriter pw = new PrintWriter(“1.txt”);

Ps.write(“hello”);

Pw.write(“\r\n”);

pw.flush();

pw.close();

PrintWriter pw2 = new PrintWriter(new FileWriter(“1.txt”), true);

Pw2.println(“hello”);

Pw2.close();

1. 对象序列化流与反序列化流
2. 对象序列化流
3. 对象序列化

将对象以一个字节序列的方式表示，该序列包含：对象的类型，对象的数据，对象中存储的属性。并将该序列存储到磁盘中或在网络中传输。

1. 对象序列化流：ObjectOutputStream

将数据对象写入OutputStream流中，保存在磁盘中或者网络传输。

1. 构造方法

方法名 说明

ObjectOutputStream(OutputStream

out)

创建一个写入指定的OutputStream的

ObjectOutputStream

1. 序列化对象的方法

方法名 说明

void writeObject(Object obj) 将指定的对象写入ObjectOutputStream

1. 注意事项
2. 对象要实现序列化，所属的类需实现Serializable接口
3. Serializable接口是标记接口，不需重写方法。
4. 示例

package File使用;  
  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
  
public class 对象序列化 {  
 public static void main(String[] args) throws IOException {  
 ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("Student\\1.txt"));  
  
 Student s = new Student("dd", 99, 99, 99);  
  
 oos.writeObject(s);  
  
 oos.close();  
 }  
}

1. 对象反序列化
2. 对象反序列化

将对象被序列化的序列逆向生成原始的对象。

1. 对象反序列化流

ObjectInputStream

1. 构造方法

方法名 说明

ObjectInputStream(InputStream in) 创建从指定的InputStream读取的ObjectInputStream

1. 反序列化对象的方法

方法名 说明

Object readObject() 从ObjectInputStream读取一个对象

1. 示例

package File使用;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
  
public class 对象反序列化 {  
 public static void main(String[] args) throws IOException, ClassNotFoundException {  
 ObjectInputStream ois = new ObjectInputStream(new FileInputStream("Student\\1.txt"));  
 Object obj = ois.readObject();  
 Student s = (Student)obj;  
 System.*out*.println(s.getName() + ":" + s.getSum());  
  
 ois.close();  
 }  
}

1. 序列化对象中的问题
2. serialVersionUID

序列化运行时与每个可序列化的类关联一个版本号，它在反序列化时使用，以验证序列化对象的发送者和接受者是否加载了与序列化兼容的对象的类。一个可序列化的类可以显示声明自己的serialVersionUID，该字段为static,final,long类型的，建议所有可序列化的类都要显示声明serialVersionUID。

Private static final long serialVersionUID;

1. transient

如果不想类中的某个成员变量被序列化，则将该成员声明为transient即可。

1. Properties集合
2. 概述

Properties是一个Map体系的集合类，可以用于将数据保存到流中或从流中加载数据，用键值对保存字符串。

注意，构造方法不是泛型方法，不需要传入类型，而是直接创建新对象。

Properties prop = new Properties();

1. 特有方法

方法名 说明

Object setProperty(String key,

String value)

设置集合的键和值，都是String类型，底层调用 Hashtable方

法 put

String getProperty(String key) 使用此属性列表中指定的键搜索属性

Set stringPropertyNames()

从该属性列表中返回一个不可修改的键集，其中键及其对应的

值是字符串

1. Properties和IO流结合的方法

方法名 说明

void load(InputStream

inStream)

从输入字节流读取属性列表（键和元素对）

void load(Reader reader) 从输入字符流读取属性列表（键和元素对）

void store(OutputStream

out, String comments)

将此属性列表（键和元素对）写入此 Properties表中，以适合于使用

load(InputStream)方法的格式写入输出字节流

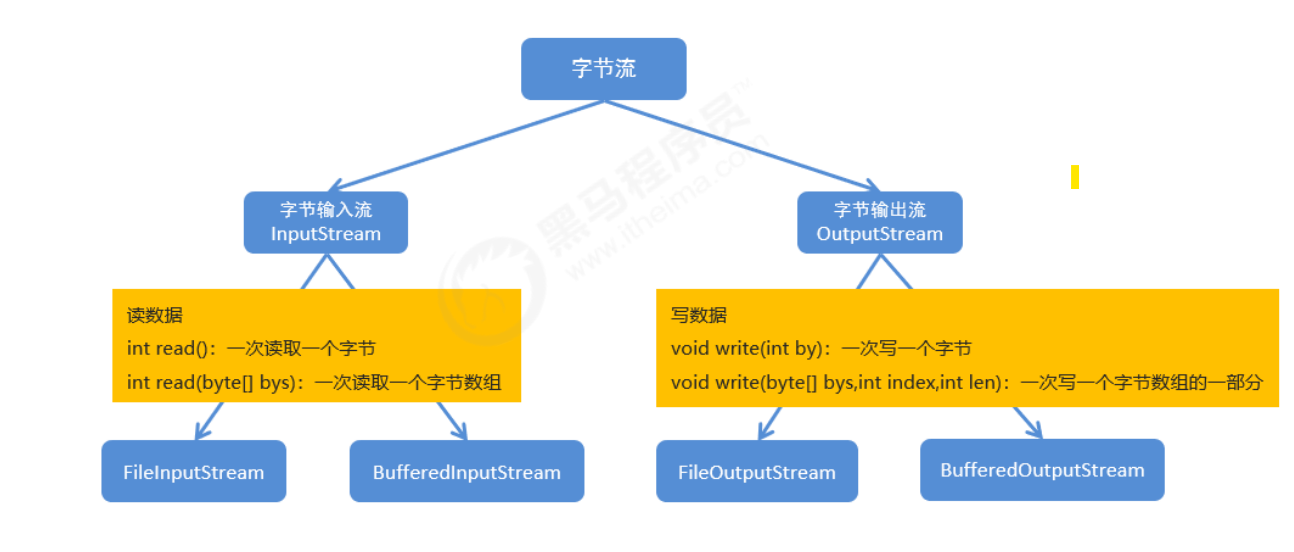
void store(Writer writer,

String comments)

将此属性列表（键和元素对）写入此 Properties表中，以适合使用

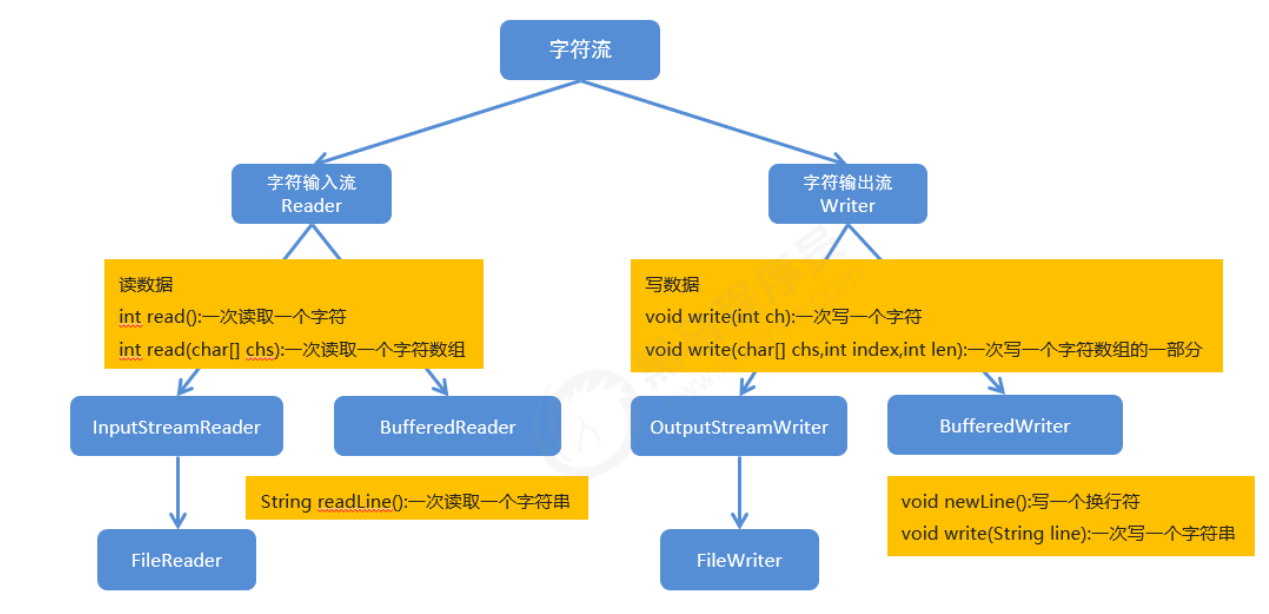
load(Reader)方法的格式写入输出字符流

1. IO流的总结
2. 字节流



一般采用的是字节缓冲流一次读写一个字节数组的方式

1. 字符流



字符流都是用来操作文本数据，一般用字符缓冲流的特有方法。