

**【缓冲流、转换流、序列化流】**

**内容**

转换流

缓冲流

序列化流

打印流

**目标**

能够使用字节缓冲流读取数据到程序

能够使用字节缓冲流写出数据到文件

能够明确字符缓冲流的作用和基本用法

能够使用缓冲流的特殊功能

能够阐述编码表的意义

能够使用转换流读取指定编码的文本文件

能够使用转换流写入指定编码的文本文件

能够说出打印流的特点

能够使用序列化流写出对象到文件

能够使用反序列化流读取文件到程序中

**第一章 缓冲流**

昨天学习了基本的一些流，作为IO流的入门，今天我们要见识一些更强大的流。比如能够高效读写的缓冲流，能够 转换编码的转换流，能够持久化存储对象的序列化流等等。这些功能更为强大的流，都是在基本的流对象基础之上 创建而来的，就像穿上铠甲的武士一样，相当于是对基本流对象的一种增强。

**1.1 概述**

缓冲流,也叫高效流，是对4个基本的 FileXxx 流的增强，所以也是4个流，按照数据类型分类：

**字节缓冲流**：

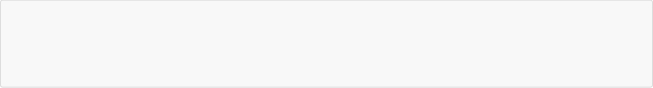
**字符缓冲流**：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| BufferedInputStream | | | ， | BufferedOutputStream | |
| BufferedReader | ， | BufferedWriter | | |  |

缓冲流的基本原理，是在创建流对象时，会创建一个内置的默认大小的缓冲区数组，通过缓冲区读写，减少系统IO 次数，从而提高读写的效率。

**1.2 字节缓冲流**

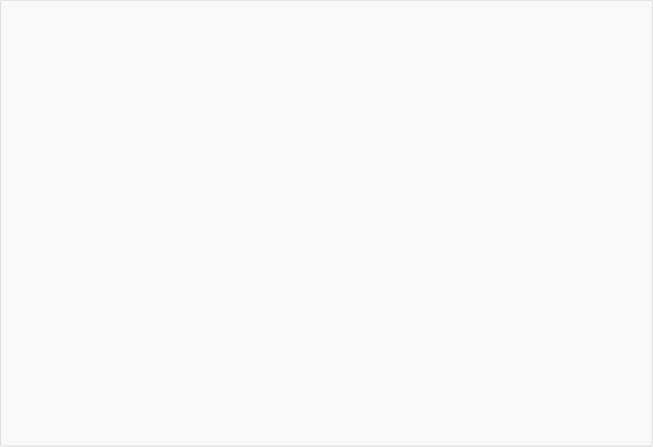
**构造方法**



 // 创建字节缓冲输入流

BufferedInputStream bis = new BufferedInputStream(new FileInputStream ("bis.txt")); // 创建字节缓冲输出流

BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("bos.txt"));



创建一个 新的缓冲输入流。 ： 创建一个新的缓冲输出流。

|  |  |
| --- | --- |
| public BufferedInputStream(InputStream in) | ： |
| public BufferedOutputStream(OutputStream out) | |

构造举例，代码如下：

**效率测试**

查询API，缓冲流读写方法与基本的流是一致的，我们通过复制大文件（375MB），测试它的效率。

1. 基本流，代码如下：

 public class BufferedDemo  {

    public static void main(String[] args) throws FileNotFoundException {

        // 记录开始时间

        long start = System.currentTimeMillis ();

        // 创建流对象

        try (

            FileInputStream  fis = new FileInputStream("jdk9.exe");

            FileOutputStream  fos = new FileOutputStream("copy.exe" )

        ){

            // 读写数据

            int b;

            while ((b = fis.read()) != ‐1) {

                fos.write(b);

            }

        } catch (IOException  e) {

            e.printStackTrace();

        }

        // 记录结束时间

        long end = System.currentTimeMillis ();

        System.out.println ("普通流复制时间:"+(end ‐ start)+" 毫秒");

    }

}

十几分钟过去了...

2. 缓冲流，代码如下：

 public class BufferedDemo  {

    public static void main(String[] args) throws FileNotFoundException {

        // 记录开始时间

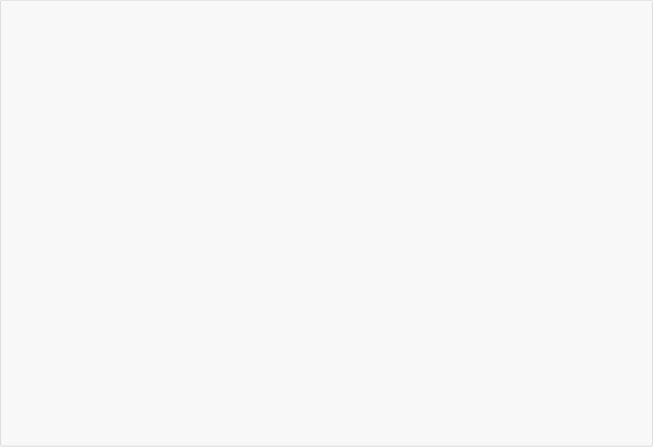
        long start = System.currentTimeMillis ();

        // 创建流对象

        try (

            BufferedInputStream  bis = new BufferedInputStream(new FileInputStream ("jdk9.exe"));          BufferedOutputStream  bos = new BufferedOutputStream(new FileOutputStream("copy.exe" ));

        ){



        // 读写数据

            int b;

            while ((b = bis.read()) != ‐1) {

                bos.write(b);

            }

        } catch (IOException  e) {

            e.printStackTrace();

        }

        // 记录结束时间

        long end = System.currentTimeMillis ();

        System.out.println ("缓冲流复制时间:"+(end ‐ start)+" 毫秒");

    }

}

缓冲流复制时间:8016 毫秒

如何更快呢？

使用数组的方式，代码如下：

 public class BufferedDemo  {

    public static void main(String[] args) throws FileNotFoundException {

        // 记录开始时间

        long start = System.currentTimeMillis ();

        // 创建流对象

        try (

            BufferedInputStream bis = new BufferedInputStream(new FileInputStream ("jdk9.exe"));          BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("copy.exe" ));         ){

            // 读写数据

            int len;

            byte[] bytes = new byte[8\*1024];

            while ((len = bis.read(bytes)) != ‐1) {

                bos.write(bytes, 0 , len);

            }

        } catch (IOException  e) {

            e.printStackTrace();

        }

        // 记录结束时间

        long end = System.currentTimeMillis ();

        System.out.println ("缓冲流使用数组复制时间:"+(end ‐ start)+" 毫秒");

    }

}

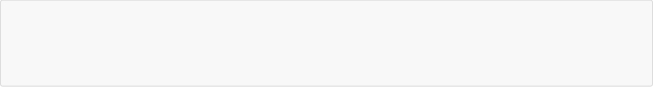
缓冲流使用数组复制时间:666 毫秒

**1.3 字符缓冲流**

**构造方法**

public BufferedReader(Reader in) ：创建一个 新的缓冲输入流。

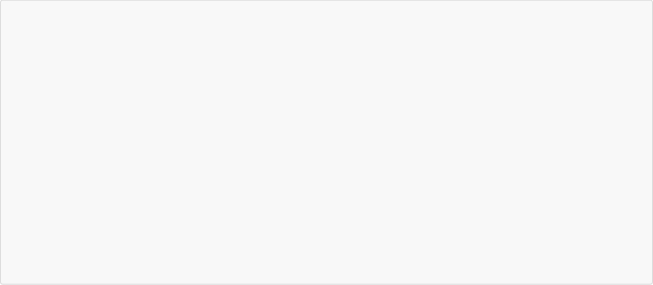
public BufferedWriter(Writer out) ： 创建一个新的缓冲输出流。



 // 创建字符缓冲输入流

BufferedReader br = new BufferedReader(new FileReader("br.txt")); // 创建字符缓冲输出流

BufferedWriter bw = new BufferedWriter(new FileWriter("bw.txt"));



 public class BufferedReaderDemo {

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

         // 创建流对象

        BufferedReader br = new BufferedReader(new FileReader("in.txt"));         // 定义字符串,保存读取的一行文字

        String line  = null;

        // 循环读取,读取到最后返回null

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

            System.out.print(line);

            System.out.println ("‐‐‐‐‐‐");

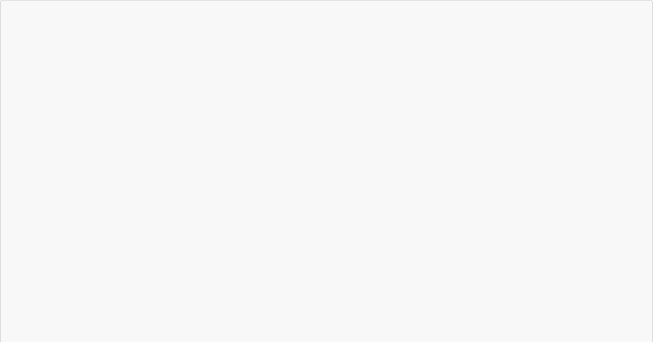
        }

        // 释放资源

        br.close();

    }

}



构造举例，代码如下：

**特有方法**

字符缓冲流的基本方法与普通字符流调用方式一致，不再阐述，我们来看它们具备的特有方法。 BuﬀeredReader： public String readLine() : 读一行文字。

BuﬀeredWriter： public void newLine() : 写一行行分隔符,由系统属性定义符号。 readLine 方法演示，代码如下：

newLine 方法演示，代码如下：

 public class BufferedWriterDemo throws IOException {

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

        // 创建流对象

        BufferedWriter bw = new BufferedWriter(new FileWriter("out.txt"));         // 写出数据

        bw.write("黑马");

        // 写出换行

        bw.newLine();

        bw.write("程序");

        bw.newLine();

        bw.write("员");

        bw.newLine();

        // 释放资源

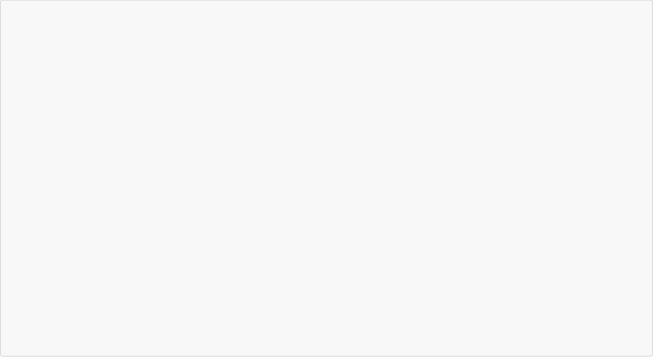
        bw.close();

    }

}

输出效果:

黑马



 3.侍中、侍郎郭攸之、费祎、董允等，此皆良实，志虑忠纯，是以先帝简拔以遗陛下。愚以为宫中之事，事无大小，悉 以咨之，然后施行，必得裨补阙漏，有所广益。

8.愿陛下托臣以讨贼兴复之效，不效，则治臣之罪，以告先帝之灵。若无兴德之言，则责攸之、祎、允等之慢，以彰其 咎；陛下亦宜自谋，以咨诹善道，察纳雅言，深追先帝遗诏，臣不胜受恩感激。

4.将军向宠，性行淑均，晓畅军事，试用之于昔日，先帝称之曰能，是以众议举宠为督。愚以为营中之事，悉以咨之， 必能使行阵和睦，优劣得所。

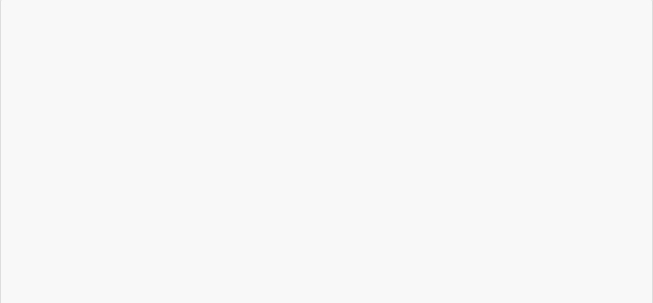
2.宫中府中，俱为一体，陟罚臧否，不宜异同。若有作奸犯科及为忠善者，宜付有司论其刑赏，以昭陛下平明之理，不 宜偏私，使内外异法也。

1.先帝创业未半而中道崩殂，今天下三分，益州疲弊，此诚危急存亡之秋也。然侍卫之臣不懈于内，忠志之士忘身于外 者，盖追先帝之殊遇，欲报之于陛下也。诚宜开张圣听，以光先帝遗德，恢弘志士之气，不宜妄自菲薄，引喻失义，以 塞忠谏之路也。

9.今当远离，临表涕零，不知所言。

6.臣本布衣，躬耕于南阳，苟全性命于乱世，不求闻达于诸侯。先帝不以臣卑鄙，猥自枉屈，三顾臣于草庐之中，咨臣 以当世之事，由是感激，遂许先帝以驱驰。后值倾覆，受任于败军之际，奉命于危难之间，尔来二十有一年矣。 7.先帝知臣谨慎，故临崩寄臣以大事也。受命以来，夙夜忧叹，恐付托不效，以伤先帝之明，故五月渡泸，深入不毛。 今南方已定，兵甲已足，当奖率三军，北定中原，庶竭驽钝，攘除奸凶，兴复汉室，还于旧都。此臣所以报先帝而忠陛 下之职分也。至于斟酌损益，进尽忠言，则攸之、祎、允之任也。

5.亲贤臣，远小人，此先汉所以兴隆也；亲小人，远贤臣，此后汉所以倾颓也。先帝在时，每与臣论此事，未尝不叹息 痛恨于桓、灵也。侍中、尚书、长史、参军，此悉贞良死节之臣，愿陛下亲之信之，则汉室之隆，可计日而待也。



程序

员

**1.4 练习:文本排序**

请将文本信息恢复顺序。

**案例分析**

1. 逐行读取文本信息。
2. 解析文本信息到集合中。
3. 遍历集合，按顺序，写出文本信息。

**案例实现**

 public class BufferedTest  {

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

        // 创建map集合,保存文本数据,键为序号,值为文字

        HashMap<String, String> lineMap = new HashMap<>();

        // 创建流对象

        BufferedReader br = new BufferedReader(new FileReader("in.txt"));         BufferedWriter bw = new BufferedWriter(new FileWriter("out.txt"));

        // 读取数据

        String line  = null;

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

            // 解析文本

            String[] split = line.split("\\.");

            // 保存到集合

            lineMap.put(split[0],split[1]);



        }

        // 释放资源

        br.close();

        // 遍历map集合

        for (int i = 1;i <= lineMap.size(); i++) {

            String key = String.valueOf (i);

            // 获取map中文本

            String value = lineMap.get(key);

            // 写出拼接文本

            bw.write(key+"."+value);

            // 写出换行

            bw.newLine ();

        }

        // 释放资源

        bw.close();

    }

}

**第二章 转换流**

**2.1 字符编码和字符集**

**字符编码**

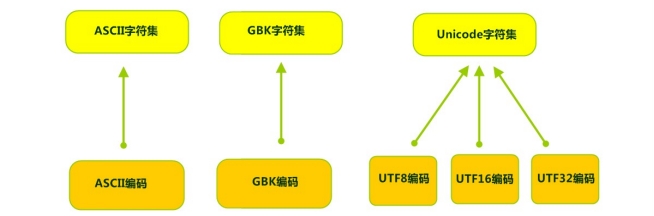
计算机中储存的信息都是用二进制数表示的，而我们在屏幕上看到的数字、英文、标点符号、汉字等字符是二进制 数转换之后的结果。按照某种规则，将字符存储到计算机中，称为**编码** 。反之，将存储在计算机中的二进制数按照 某种规则解析显示出来，称为**解码** 。比如说，按照A规则存储，同样按照A规则解析，那么就能显示正确的文本f符 号。反之，按照A规则存储，再按照B规则解析，就会导致乱码现象。

**字符编码 Character Encoding** : 就是一套自然语言的字符与二进制数之间的对应规则。

**字符集**

**字符集 Charset** ：也叫编码表。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。

计算机要准确的存储和识别各种字符集符号，需要进行字符编码，一套字符集必然至少有一套字符编码。常见字符 集有ASCII字符集、GBK字符集、Unicode 字符集等。



可见，当指定了**编码**，它所对应的**字符集**自然就指定了，所以**编码**才是我们最终要关心的。

**ASCII字符集** ：

ASCII（American Standard Code for Information Interchange ，美国信息交换标准代码）是基于拉丁 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（bits）表示一个字符，共128字符。ASCII的扩展字符集使用8位（bits） 表示一个字符，共256字符，方便支持欧洲常用字符。

**ISO-8859-1字符集**：

拉丁码表，别名Latin-1，用于显示欧洲使用的语言，包括荷兰、丹麦、德语、意大利语、西班牙语等。 ISO-5559-1 使用单字节编码，兼容ASCII编码。

**GBxxx字符集**：

GB就是国标的意思，是为了显示中文而设计的一套字符集。

**GB2312**：简体中文码表。一个小于127的字符的意义与原来相同。但两个大于127的字符连在一起时， 就表示一个汉字，这样大约可以组合了包含7000多个简体汉字，此外数学符号、罗马希腊的字母、日文 的假名们都编进去了，连在ASCII里本来就有的数字、标点、字母都统统重新编了两个字节长的编码，这 就是常说的"全角"字符，而原来在127号以下的那些就叫"半角"字符了。

**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。

**GB18030**：最新的中文码表。收录汉字70244个，采用多字节编码，每个字可以由1个、2个或4个字节 组成。支持中国国内少数民族的文字，同时支持繁体汉字以及日韩汉字等。

**Unicode字符集** ：

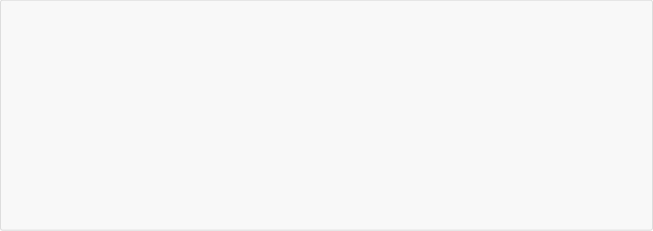
Unicode 编码系统为表达任意语言的任意字符而设计，是业界的一种标准，也称为统一码、标准万国 码。

它最多使用4个字节的数字来表达每个字母、符号，或者文字。有三种编码方案，UTF-8 、UTF-16 和UTF- 32。最为常用的UTF-8编码。

UTF-8 编码，可以用来表示Unicode 标准中任何字符，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。所以， 我们开发Web应用，也要使用UTF-8编码。它使用一至四个字节为每个字符编码，编码规则：

1. 128个US-ASCII字符，只需一个字节编码。
2. 拉丁文等字符，需要二个字节编码。
3. 大部分常用字（含中文），使用三个字节编码。
4. 其他极少使用的Unicode 辅助字符，使用四字节编码。

**2.2 编码引出的问题**



 public class ReaderDemo {

    public static void main(String[] args) throws IOException  {         FileReader fileReader = new FileReader("E:\\File\_GBK.txt");         int read;

        while ((read = fileReader.read()) != ‐1) {

            System.out.print((char)read);

        }

        fileReader.close();

    }

}

输出结果：

���



 InputStreamReader isr = new InputStreamReader(new FileInputStream("in.txt")); InputStreamReader isr2 = new InputStreamReader(new FileInputStream("in.txt") , "GBK");



符集的字符流。

**2.2 编码引出的问题**

在IDEA中，使用 FileReader 读取项目中的文本文件。由于IDEA的设置，都是默认的 UTF-8 编码，所以没有任何 问题。但是，当读取Windows系统中创建的文本文件时，由于Windows系统的默认是GBK编码，就会出现乱码。

那么如何读取GBK编码的文件呢？

**2.3 InputStreamReader类**

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。

**构造方法**

|  |  |
| --- | --- |
| InputStreamReader(InputStream in) | : 创建一个使用默认字 |
| InputStreamReader(InputStream in, String charsetName) | |

: 创建一个指定字符集的字符流。

构造举例，代码如下：

**指定编码读取**

 public class ReaderDemo2 {

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

        // 定义文件路径,文件为gbk编码

        String FileName = "E:\\file\_gbk.txt" ;

        // 创建流对象,默认UTF8编码

        InputStreamReader isr = new InputStreamReader(new FileInputStream(FileName));

        // 创建流对象,指定GBK编码

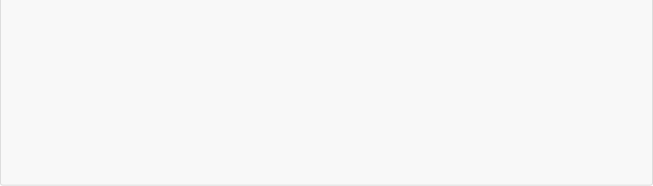
        InputStreamReader isr2 = new InputStreamReader(new FileInputStream(FileName) , "GBK");         // 定义变量,保存字符

        int read;

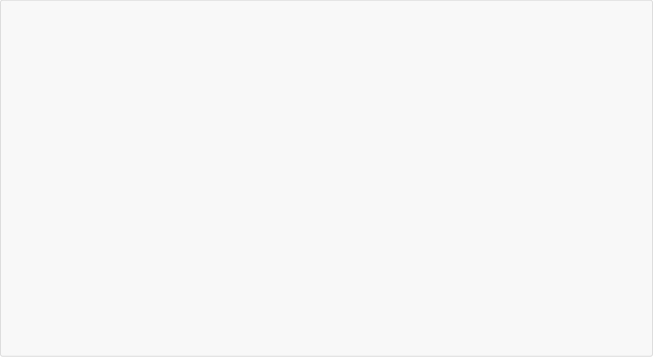
        // 使用默认编码字符流读取,乱码

        while ((read = isr.read()) != ‐1) {

            System.out.print((char)read); // ��Һ�



 OutputStreamWriter isr = new OutputStreamWriter(new FileOutputStream("out.txt")); OutputStreamWriter isr2 = new OutputStreamWriter(new FileOutputStream("out.txt") , "GBK");



 public class OutputDemo {

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

        // 定义文件路径

        String FileName = "E:\\out.txt";

        // 创建流对象,默认UTF8编码

        OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream(FileName ));         // 写出数据

        osw.write("你好"); // 保存为6个字节

        osw.close();

        // 定义文件路径

        String FileName2 = "E:\\out2.txt" ;

        // 创建流对象,指定GBK编码

        OutputStreamWriter osw2 = new OutputStreamWriter(new FileOutputStream(FileName2),"GBK");         // 写出数据

        osw2.write("你好");// 保存为4个字节

        osw2.close();

    }

}

符集的字符流。

        }

        isr.close();

        // 使用指定编码字符流读取,正常解析

        while ((read = isr2.read()) != ‐1) {

            System.out.print((char)read);// 大家好

        }

        isr2.close();

    }

}

**2.4 OutputStreamWriter类**

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。

**构造方法**

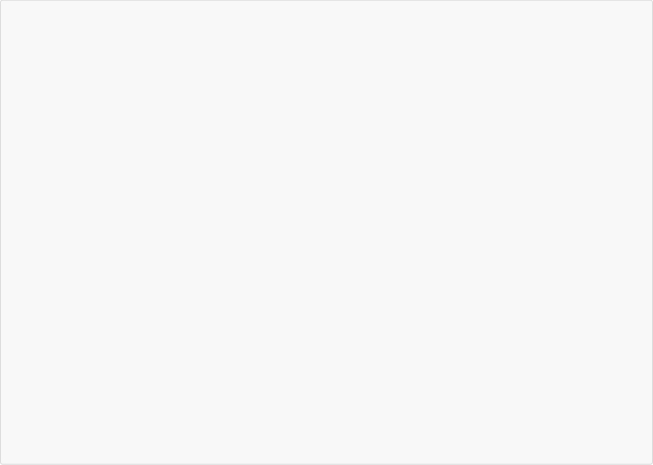
|  |  |
| --- | --- |
| OutputStreamWriter(OutputStream in) | : 创建一个使用默认字 |
| OutputStreamWriter(OutputStream in, String charsetName) | |

: 创建一个指定字符集的字符流。

构造举例，代码如下：

**指定编码写出**

**转换流理解图解**



**转换流是字节与字符间的桥梁！**

**2.5 练习：转换文件编码**

将GBK编码的文本文件，转换为UTF-8 编码的文本文件。

**案例分析**

1. 指定GBK编码的转换流，读取文本文件。
2. 使用UTF-8 编码的转换流，写出文本文件。

**案例实现**

 public class TransDemo {

   public static void main(String[] args) {

        // 1.定义文件路径

        String srcFile = "file\_gbk.txt" ;

        String destFile = "file\_utf8.txt";

        // 2.创建流对象

        // 2.1 转换输入流,指定GBK编码

        InputStreamReader isr = new InputStreamReader(new FileInputStream(srcFile) , "GBK");         // 2.2 转换输出流,默认utf8编码

        OutputStreamWriter osw = new OutputStreamWriter(new FileOutputStream(destFile ));         // 3.读写数据

        // 3.1 定义数组

        char[] cbuf = new char[1024];

        // 3.2 定义长度

        int len;

        // 3.3 循环读取

        while ((len = isr.read(cbuf))!=‐1) {

            // 循环写出

            osw.write(cbuf,0,len);

        }

        // 4.释放资源

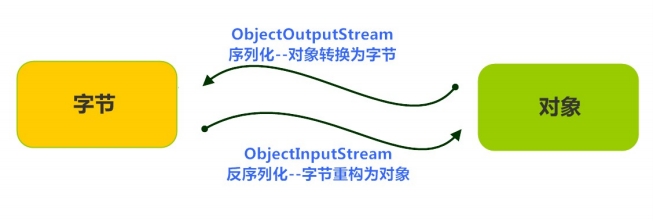
        osw.close();

        isr.close();

    }

}

**第三章 序列化**



 FileOutputStream fileOut = new FileOutputStream("employee.txt"); ObjectOutputStream out = new ObjectOutputStream(fileOut );



**第三章 序列化**

**3.1 概述**

Java 提供了一种对象**序列化**的机制。用一个字节序列可以表示一个对象，该字节序列包含该 对象的数据 、 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。

反之，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。看图理解序列化：

**3.2 ObjectOutputStream类**

java.io.ObjectOutputStream 类，将Java对象的原始数据类型写出到文件,实现对象的持久存储。

**构造方法**

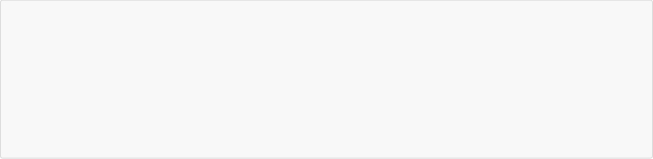
public ObjectOutputStream(OutputStream out) ： 创建一个指定OutputStream的ObjectOutputStream 。 构造举例，代码如下：

**序列化操作**

1. 一个对象要想序列化，必须满足两个条件:

该类必须实现 java.io.Serializable 接口， Serializable 是一个标记接口，不实现此接口的类将不会使任 何状态序列化或反序列化，会抛出 NotSerializableException 。

该类的所有属性必须是可序列化的。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。



 public class Employee implements java.io.Serializable  {

    public String name;

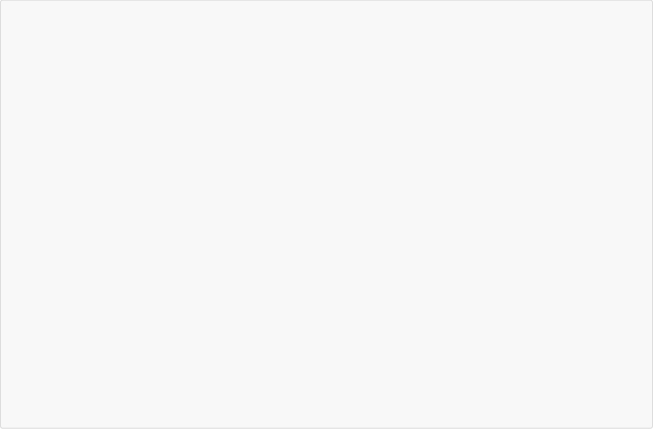
    public String address;

    public transient int age;// transient 瞬态修饰成员,不会被序列化

    public void addressCheck () {

        System.out.println ("Address  check : " + name + "‐‐ " + address);     }

}



2.写出对象方法

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

 public class SerializeDemo{

    public static void main(String [] args)   {

        Employee e = new Employee();

        e.name = "zhangsan";

        e.address = "beiqinglu";

        e.age = 20;

        try {

            // 创建序列化流对象

          ObjectOutputStream  out = new ObjectOutputStream(new FileOutputStream("employee.txt"));             // 写出对象

            out.writeObject(e);

            // 释放资源

            out.close();

            fileOut.close();

            System.out.println ("Serialized data is saved"); // 姓名，地址被序列化，年龄没有被序列 化。

        } catch(IOException i)   {

            i.printStackTrace();

        }

    }

}

输出结果：

Serialized data is saved

**3.3 ObjectInputStream类**

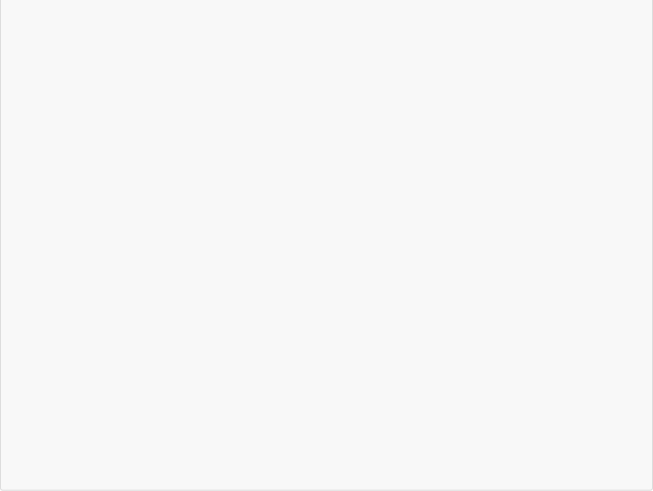
ObjectInputStream 反序列化流，将之前使用ObjectOutputStream 序列化的原始数据恢复为对象。

**构造方法**

public ObjectInputStream(InputStream in) ： 创建一个指定InputStream的ObjectInputStream 。 **反序列化操作1**

如果能找到一个对象的class文件，我们可以进行反序列化操作，调用 ObjectInputStream 读取对象的方法： public final Object readObject () : 读取一个对象。

 public class DeserializeDemo {



   public static void main(String [] args)   {

        Employee e = null;

        try {

             // 创建反序列化流

             FileInputStream fileIn = new FileInputStream("employee.txt");

             ObjectInputStream in = new ObjectInputStream(fileIn);

             // 读取一个对象

             e = (Employee) in.readObject ();

             // 释放资源

             in.close();

             fileIn.close();

        }catch(IOException  i) {

             // 捕获其他异常

             i.printStackTrace();

             return;

        }catch(ClassNotFoundException c)  {

            // 捕获类找不到异常

             System.out.println("Employee class not found");

             c.printStackTrace();

             return;

        }

        // 无异常,直接打印输出

        System.out.println ("Name: "  + e.name);  // zhangsan

        System.out.println ("Address: "  + e.address ); // beiqinglu

        System.out.println ("age: "  + e.age); // 0

    }

}

**对于JVM可以反序列化对象，它必须是能够找到class文件的类。如果找不到该类的class文件，则抛出一个 ClassNotFoundException 异常。**

**反序列化操作2**

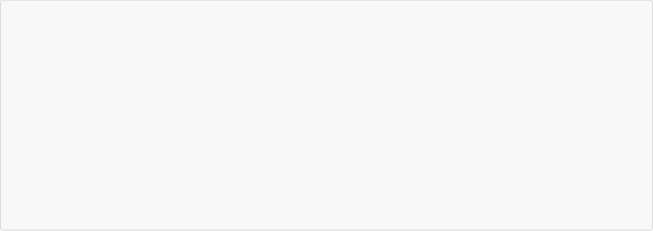
**另外，当JVM反序列化对象时，能找到class文件，但是class文件在序列化对象之后发生了修改，那么反序列化操 作也会失败，抛出一个 InvalidClassException 异常。**发生这个异常的原因如下：

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

该类包含未知数据类型

该类没有可访问的无参数构造方法

Serializable 接口给需要序列化的类，提供了一个序列版本号。 serialVersionUID 该版本号的目的在于验证序 列化的对象和对应类是否版本匹配。



 public class Employee implements java.io.Serializable  {

     // 加入序列版本号

     private static final long serialVersionUID = 1L;

     public String name;

     public String address;

     // 添加新的属性 ,重新编译, 可以反序列化,该属性赋为默认值.

     public int eid;

     public void addressCheck () {

         System.out.println("Address  check : " + name + "‐‐ " + address);      }

}



**3.4 练习：序列化集合**

1. 将存有多个自定义对象的集合序列化操作，保存到 list.txt 文件中。
2. 反序列化 list.txt ，并遍历集合，打印对象信息。

**案例分析**

1. 把若干学生对象 ，保存到集合中。
2. 把集合序列化。
3. 反序列化读取时，只需要读取一次，转换为集合类型。
4. 遍历集合，可以打印所有的学生信息

**案例实现**

 public class SerTest {

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

        // 创建 学生对象

        Student  student = new Student("老王", "laow");

        Student  student2 = new Student("老张", "laoz");

        Student  student3 = new Student("老李", "laol");

        ArrayList <Student > arrayList = new ArrayList<>();

        arrayList .add(student );

        arrayList .add(student2 );

        arrayList .add(student3 );

        // 序列化操作

        // serializ(arrayList);

        // 反序列化

        ObjectInputStream ois  = new ObjectInputStream(new FileInputStream("list.txt"));         // 读取对象,强转为ArrayList 类型

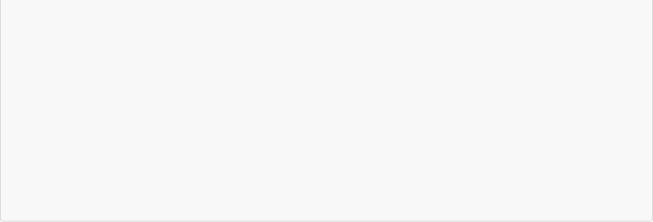
        ArrayList <Student > list  = (ArrayList<Student >)ois.readObject();

        for (int i = 0;i < list.size(); i++ ){

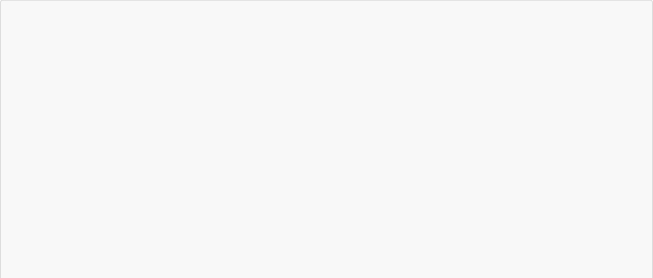
            Student s = list.get(i);

            System.out.println (s.getName()+"‐‐"+ s.getPwd());

      }



 PrintStream ps = new PrintStream("ps.txt")；



        }

    }

    private static void serializ (ArrayList<Student > arrayList) throws Exception  {

        // 创建 序列化流

        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("list.txt" ));         // 写出对象

        oos.writeObject (arrayList);

        // 释放资源

        oos.close();

    }

}

**第四章 打印流**

**4.1 概述**

平时我们在控制台打印输出，是调用 print 方法和 println 方法完成的，这两个方法都来自于 java.io.PrintStream 类，该类能够方便地打印各种数据类型的值，是一种便捷的输出方式。

**4.2 PrintStream类**

**构造方法**

public PrintStream(String fileName) ： 使用指定的文件名创建一个新的打印流。

构造举例，代码如下：

**改变打印流向**

System.out 就是 PrintStream 类型的，只不过它的流向是系统规定的，打印在控制台上。不过，既然是流对象， 我们就可以玩一个"小把戏"，改变它的流向。

 public class PrintDemo {

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

        // 调用系统的打印流,控制台直接输出97

        System.out.println (97);

        // 创建打印流,指定文件的名称

        PrintStream ps = new PrintStream("ps.txt");

        // 设置系统的打印流流向,输出到ps.txt

        System.setOut(ps);

        // 调用系统的打印流,ps.txt中输出97

        System.out.println (97);

    }

}

