day19_IO流02

一 内容回顾（列举前一天重点难点内容）

1.1 教学重点:

```
1.掌握文件的基本操作
2.掌握IO流的分类体系
3.掌握字节流的读写过程实现
4.掌握字节流的读写原理
5.掌握字符流的读写过程实现
6.掌握字符流的读写原理
```

二 教学目标

```
1.掌握缓冲流的基本过程实现
2.掌握缓冲流的实现原理
3.掌握标准流的基本使用
4.掌握转换流的基本使用
5.掌握打印流的基本使用
6.掌握序列化流的基本使用
7.了解装饰设计模式
8.了解编码问题
9.了解Properties的使用
```

三 教学导读

3.1. IO流02

昨天讲解的是IO流的基础实现,今天的IO流02主要是对IO流的升级讲解.包括有一些特殊功能的流的子类,可以高效处理数据读写的缓冲流,还有读写数据时遇到的编码问题.

四 教学内容

4.1. 缓冲流(会)

4.1.1. 缓冲流的简介

- 基本介绍

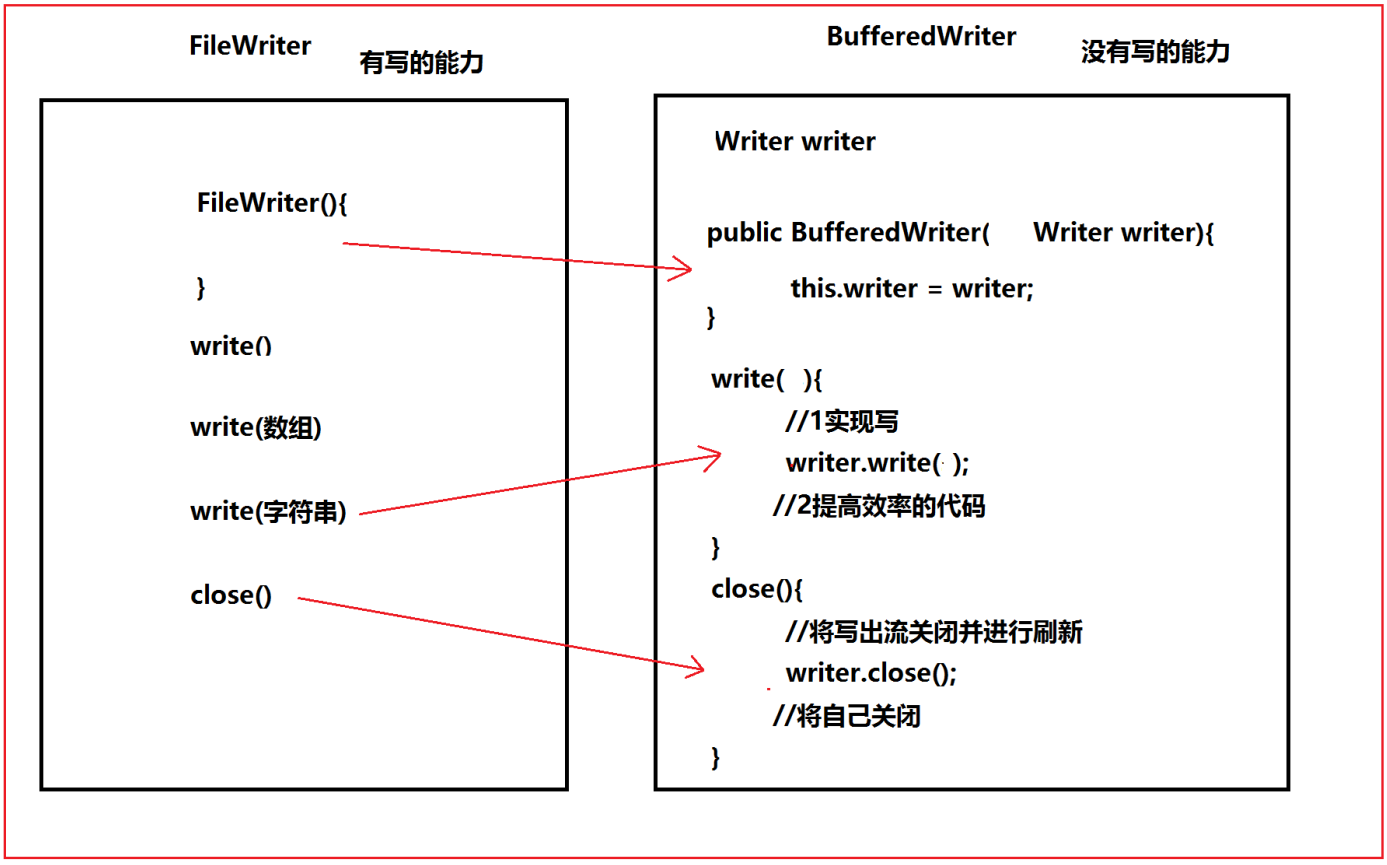
给普通的IO流， 套上一个缓冲区。 所有的使用缓冲流进行的读写操作， 都是和缓冲区进行交互的， 避免了频繁的IO操作。 这样一来， 带来的好处就是可以提高读写的效率。 这个缓冲区， 其实是一个数组。

- 缓冲流的作用

为了提高读写的能力,本身没有读写的能力,要想进行读写就必须借助于字符流/字节流实现.

可以将缓冲流类比于催化剂或者高速的小车

- **常见的缓冲流:**
  - BufferedInputStream : 缓冲字节输入流
  - BufferedOutputStream： 缓冲字节输出流
  - BufferedReader： 缓冲字符输入流
  - BufferedWriter： 缓冲字符输出流
- 字符流和缓冲字符流对比



图示分析:

使用缓冲流实现读写的步骤与字符流一样,只是需要我们先通过构造方法传入一个字符流对象.同时缓冲流可以提高读写效率.

- **总结**:

大家在使用流读写数据时,尽量使用缓冲流,缓冲流中尽量使用缓冲字符流,在字符缓冲流中比缓冲字节流多了readLine()和newLine()方法.

4.1.2. 缓冲字节流

```
1 import java.io.BufferedInputStream;
2 import java.io.FileInputStream;
3 import java.io.IOException;
4
5 /**
6  * @Author 千锋大数据教学团队
7  * @Company 千锋好程序员大数据
```

```java
 * @Description BufferedInputStream使用
 */
public class BufferedInputStreamTest {
    public static void main(String[] args) {
        // 过程和InputStream一模一样的
        // 缓冲字节输入流是需要基于一个字节输入流来进行实例化的
        // 在这里，BufferedInputStream构造方法中的InputStream对象，只是用来做当前的对象的实例化，在使用结束的时候，理论上来讲，是需要关闭的
        // 实际在使用中，使用结束后，只需要关闭BufferedInputStream即可。
        try (BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream("file\\day26\\source"))) {
            // 1. 实例化一个字节数组
            byte[] array = new byte[1024];
            // 2. 声明一个整型变量，用来记录每次读取了多少个字节数据
            int length = 0;
            // 3. 循环读取
            while ((length = bufferedInputStream.read(array)) != -1) {
                // 4. 将读取到的数据转成字符串输出到控制台
                String msg = new String(array, 0, length);
                System.out.println(msg);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description BufferedOutputStream
 */
public class BufferedOutputStreamTest {
    public static void main(String[] args) {
        // 1. 实例化一个缓冲字节输出流对象
        try (BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(new FileOutputStream("file\\day26\\target")))  {
            // 2. 将数据写入到输出流中
            bufferedOutputStream.write("hello world".getBytes());
            bufferedOutputStream.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.1.3. 缓冲字符流

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

```

```java
/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description
 */
public class BufferedReaderTest {
    public static void main(String[] args) {
        // 借助一个字符流，实例化一个缓冲字符输入流
        try (BufferedReader bufferedReader = new BufferedReader(new FileReader("file\\day26\\src"))) {
            // 从流中读取数据
            char[] array = new char[100];
            int length = 0;
            while ((length = bufferedReader.read(array)) != -1) {
                System.out.print(new String(array, 0, length));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description
 */
```

```
public class BufferedWriterTest {
    public static void main(String[] args) {
        // 借助一个字符输出流，实例化一个缓冲字符输出流对象
        try (BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("file\\day26\\dst"))) {
            bufferedWriter.write("hello world");
            bufferedWriter.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.1.4. 缓冲字符流中的特殊方法

BufferedReader 类中多了一个方法 readLine()

- 意义: 读取缓冲流中的一行数据， 可以逐行读取。 一直到读取到的数据是null， 表示数据读取完了， 没有下一行数据了。
- 注意事项: readLine() 是逐行读取， 但是， 只能读取到一行中的内容，并不能读取走换行符。

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Date 2020/4/26
 * @Description
 */
```

```java
public class BufferedReaderSpecial {
    public static void main(String[] args) {
        try (BufferedReader reader = new BufferedReader(new FileReader("file\\day26\\src"))) {
            // 1. 定义一个字符串，用来接收每一行读取到的数据
            String line = "";
            // 2. 循环读取数据
            while ((line = reader.readLine()) != null) {
                // 3. 将读取到的数据输出
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

BufferedWriter 类中多了一个方法 newLine()

- 写换行符 ,不同的系统使用的默认换行符不一样 windows系统 \r\n linux \n
- 意义： 无参的方法， 写一个换行符,支持跨平台(平台无关性)

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description
```

```
 */
public class BufferedWriterSpecial {
    public static void main(String[] args) {
        try (BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter("file\\day26\\dst"))) {
            bufferedWriter.write("hello world");
            bufferedWriter.newLine();
            bufferedWriter.write("你好，世界");
            bufferedWriter.newLine();
            bufferedWriter.write("end");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

课上练习

使用缓冲字符流进行文件的拷贝

```
import java.io.*;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 使用缓冲字符流实现文本文件的拷贝
 */
public class BufferedCopy {
    public static void main(String[] args) {
        try (BufferedReader reader = new
BufferedReader(new FileReader("file\\day26\\src"));
             BufferedWriter writer = new
BufferedWriter(new
FileWriter("file\\day26\\destination"))) {
```

```
            String line = "";
            while ((line = reader.readLine()) != null)
{
                writer.write(line);
                writer.newLine();
            }
            writer.flush();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.1.5. LineNumberReader

是BufferedReader的子类,不能读.但是可以提高效率,特有功能:设置行号,获取行号

示例代码

```
import java.io.FileReader;
import java.io.IOException;
import java.io.LineNumberReader;

public class Demo10{
    public static void main(String[] args) throws
IOException {
        LineNumberReader lineNumberReader = new
LineNumberReader(new
FileReader("BigData2005N18\\src\\com\\qf\\test\\Demo1.j
ava"));
        //设置行号,默认从0开始,从1开始打印
        lineNumberReader.setLineNumber(10);
```

```
10          String data = null;
11          while ((data = lineNumberReader.readLine()) !=
    null) {
12
    System.out.print(lineNumberReader.getLineNumber());//获
    取行号
13              System.out.print(data);
14              System.out.println();
15          }
16          lineNumberReader.close();
17      }
18  }
```

4.1.6. 装饰设计模式(了解)

- 设计模式简介

设计模式， 前人总结出来的对一些常见问题的解决方案,后人直接拿来使用.

常用的设计模式:单例,工厂,代理,适配器,装饰,模板,观察者等,一共有23种

- 装饰设计模式

基于已经实现的功能,提供增强的功能.

- 装饰设计模式特点

装饰设计模式的由来就来自于对缓冲流的实现.

从缓冲流的角度讲解

1.使流原来的继承体更加的简单

2.提高了效率

3.由于是在原有的基础上提高增强的功能,所以他还要属于原来的体系

- 如果自己设计装饰设计模式,怎么处理?

1.原来的类 Test---Reader

2.装饰类 BTest----MyBufferedReader

步骤:

1.让BTest 继承自Test

2.在BTest内有一个Test类型的成员变量

3.通过BTest内一个带参数的构造方法接收外部传入的一个Test类型的对象,交给内部的Test的属性

4.在实现功能的时候,调用传入的Test类型的对象实现原有的功能,自己实现增强的功能.

- 示例代码

```
/*
 * 模拟字符缓冲读入流:BufferedReader
 *
 * 分析:
 * 1.要属于流的体系
 * 2.要有一个Reader类型的成员变量
 * 3.要有一个带参数的构造方法接收外部传入的流对象
 * 4.模拟readLine(),实现读一行的功能
 * 5.关闭流
 */
public class Demo9 {
    public static void main(String[] args) throws
IOException {
```

```
13          MyBufferedReader myBufferedReader = new
    MyBufferedReader(new
    FileReader("BigData2005N18\\src\\com\\qf\\test\\Demo1.j
    ava"));
14          String data = null;
15          while ((data = myBufferedReader.readLine()) !=
    null){
16              System.out.println(data);
17          }
18
19          myBufferedReader.close();
20      }
21  }
22  //1.要属于流的体系
23  class MyBufferedReader extends Reader{
24      //2.要有一个Reader类型的成员变量
25      Reader reader;
26      //3.要有一个带参数的构造方法接收外部传入的流对象
27      public MyBufferedReader(Reader reader) {
28          this.reader = reader;
29      }
30
31      //4.模拟readLine(),实现读一行的功能
32      public String readLine() throws IOException {
33          //用于存储一行的字符
34          StringBuffer stringBuffer = new StringBuffer();
35          int num=0;
36          while ((num = reader.read()) != -1){
37              if (num == '\r'){
38                  continue;
39              }else  if(num == '\n'){
40                  return stringBuffer.toString();
41              }else {
```

```
                stringBuffer.append((char)num);
            }
        }
        //当空文本时的处理
        if (stringBuffer.length() == 0){
            return null;
        }

        //只有一行数据
        return stringBuffer.toString();

        //还有提高效率的代码
    }

    @Override
    public int read(char[] cbuf, int off, int len)
throws IOException {
        return 0;
    }

    //5.关闭流
    @Override
    public void close() throws IOException {
        //把对应的流关闭
        reader.close();
        //把自己关闭
    }
}
```

4.2. Scanner类(会)

4.2.1. 简介

这个类， 并不是一个IO流。 是一个扫描器， 这个类最主要的作用， 是从一个文件中或者从一个流中浏览数据。 在这个类中封装了若干个方法， 方便了数据的读取。

4.2.2. API

方法	描述
next()	读取一个单词，遇到空格或者换行符就不再读取了。
hasNext()	判断是否还有下一个单词可以读取。
nextLine()	读取一行内容，遇到换行符就不再读取了。
hasNextLine()	判断是否还有下一行可以读取。

注意事项

这里nextLine和BufferedReader中的readLine都可以读取一行数据。 但是区别在于： 结束条件不同。

- BufferedReader: 如果读取到的数据是null, 说明没有下一行数据了。
- Scanner: 如果没有下一行了，再去读取，会出现异常。 所以， 此时的结束条件是 hasNextLine() 为false。

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import java.util.regex.Pattern;
```

```
5
6  /**
7   * @Author 千锋大数据教学团队
8   * @Company 千锋好程序员大数据
9   * @Description Scanner类的方法
10  */
11 public class ScannerTest {
12     public static void main(String[] args) {
13         // 其实，Scanner在使用结束之后，也是需要进行关闭的。调用close方法。
14         try (Scanner scanner = new Scanner(new File("file\\day26\\src"))) {
15             // 读取文件中的内容
16             while (scanner.hasNextLine()) {
17 
System.out.println(scanner.hasNextLine());
18             }
19         } catch (FileNotFoundException e) {
20             e.printStackTrace();
21         }
22     }
23 }
```

4.3. 标准输入输出流(会)

4.3.1. 简介

- 标准输入流: System.in : “标准”输入流。此流已打开并准备提供输入数据。通常，此流对应于键盘输入或者由主机环境或用户指定的另一个输入源。

```
输入源:可以发送数据到内存的设备
输出源:可以接收内存的数据的设备

1.当前的流已经打开并关联了输入源--键盘
2.如果不想让键盘充当输入源,可以通过setIn进行更换
3.是一个字节流
```

- 标准输出流: System.out： 标准”输出流。此流已打开并准备接受输出数据。通常，此流对应于显示器输出或者由主机环境或用户指定的另一个输出目标。

4.3.2. 标准输入流

```java
import java.io.BufferedInputStream;
import java.io.IOException;
import java.util.Scanner;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 标准输入流
 */
public class SystemInTest {
    public static void main(String[] args) {
        //创建了标准输入流并关联了键盘(默认的)
        //InputStream inputStream =  System.in;
        //阻塞式方法
        //int num = inputStream.read();
        //System.out.println(num);

       //实例演示
```

```
        try (BufferedInputStream bis = new BufferedInputStream(System.in)) {
            byte[] array = new byte[128];
            int length = 0;
            while ((length = bis.read(array)) != -1) {
                String str = new String(array, 0, length);
                System.out.println(str);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

课上练习

实现从键盘不断接收字符的程序

要求:一行一行的接收

```
public class Demo10 {
    public static void main(String[] args) throws IOException {
        //创建了标准输入流并关联了键盘(默认的)
        InputStream inputStream =  System.in;
        myReadLine(inputStream);
    }

    public static void myReadLine(InputStream inputStream) throws IOException {
        StringBuffer stringBuffer = new StringBuffer();
        while (true) {
            int num = inputStream.read();
```

```
            if (num == '\r') {
                continue;
            }else if (num == '\n') {

System.out.println(stringBuffer.toString());
                //当用户输入over的时候,结束程序
                if
(stringBuffer.toString().equals("over")) {
                    break;
                }

                //将上一次的值清除掉
                stringBuffer.delete(0,
stringBuffer.length());
            }else {
                stringBuffer.append((char)num);
            }
        }
    }
}
```

4.3.3. 标准输出流

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 标准输出流
```

```
 */
public class SystemOutTest {
    public static void main(String[] args) {
        PrintStream original = System.out;
        // PrintStream: 是一个打印流，可以将数据输出到指定位置。
        try (PrintStream ps = new PrintStream(new FileOutputStream("file\\day26\\logs", true))) {
            // ps.println("hello world! ");
            // 重定向标准输出流
            System.setOut(ps);

            System.out.println("123");
        }
        catch (IOException e) {
            e.printStackTrace();
        } finally {
            System.setOut(original);
        }

        System.out.println("你好");

        // System.out;      标准输出流地址
        // System.out -> ps
    }
}
```

4.4. 转换流(了解)

4.4.1. 为什么要用转换流

在进行文件读取的时候， 如果项目采用的字符集和文件的字符集不同，会出现乱码的情况。

4.4.2. 输入流

```
import java.io.*;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 转换流
 *      转换输入流：可以以指定的字符集读取某一个文件中的数据
 *      转换输出流：可以以指定的字符集把数据写入到某一个文件
 */
public class TransforeTest {
    public static void main(String[] args) {
        read();
    }
    private static void read() {
        // 当前的项目是 utf-8, 读取的文件是 GBK
        // 如果需要以指定的字符集进行文件的读取，需要使用 InputStreamReader(InputStream inputStream, String charsetName)
        try (InputStreamReader reader = new InputStreamReader(new FileInputStream("file\\day26\\src"), "GBK")) {
            char[] array = new char[128];
            int length = 0;
            while ((length = reader.read(array)) != -1) {
```

```
                System.out.println(new String(array, 0, length));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

4.4.3. 输出流

```
import java.io.*;

/**
 * @Author 千锋大数据教学团队
 * @Company 千锋好程序员大数据
 * @Description 转换流
 *      转换输入流：可以以指定的字符集读取某一个文件中的数据
 *      转换输出流：可以以指定的字符集把数据写入到某一个文件
 */
public class TransforeTest {
    public static void main(String[] args) {
        write();
    }

    private static void write() {
        // 以指定的字符集写数据
        try (OutputStreamWriter writer = new OutputStreamWriter(new FileOutputStream("file\\day26\\dst", true), "GBK")) {
            writer.write("hello world");
            writer.write("你好，世界");
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

4.5. 打印流(会)

4.5.1. 打印流分类

除了拥有输出流的特点之外,还有打印的功能.

- 字节打印流:PrintStream
- 字符打印流:PrintWriter

4.5.2. 字节打印流

字节打印流支持的设备:

- 1.File类型的文件
- 2.字符串类型的文件
- 3.字节输出流

```
public class Demo14 {
    public static void main(String[] args) throws IOException {
        //设备:File类型的文件
//      PrintStream p1 = new PrintStream(new File("BigData2005N18\\test7.txt"));
        //设备:字节输出流
//      PrintStream p2 = new PrintStream(new FileOutputStream("BigData2005N18\\test7.txt"));
        //设备:字符串类型的文件
//      PrintStream p3 = new PrintStream("BigData2005N18\\test7.txt");
```

```
9
10          //实例
11          PrintStream p3 = new
PrintStream("BigData2005N18\\test7.txt");
12          //直接使用write方法打印,默认只支持一个字节,所以当数据超
出了一个字节的范围,会出现下面的错误.
13          p3.write(97);//00000000 01100001    默认会将高字节
删掉  01100001-----a
14          p3.write(353);//00000001 01100001  ---
-01100001---a
15
16          //一般我们不直接使用write方法,而是使用print方法
17
18          p3.println(353);//直接输出353
19
20          //这是print方法的内部实现原理
21          //先将353转成字符串再转成字节数组
22          p3.write(String.valueOf(353).getBytes());
23          p3.close();
24      }
25  }
```

4.5.3. 字符打印流

- 字符打印流支持的设备:
  - 1.File类型的文件
  - 2.字符串类型的文件
  - 3.字节输出流
  - 4.字符写出流
- 注意点:

  方法:public PrintWriter(Writer out, boolean autoFlush)

autoFlush - boolean 变量；如果为 true，则 println、printf 或 format 方法将自动刷新输出缓冲区

但是执行print方式时需要手动刷新

```
public class Demo11 {
  public static void main(String[] args) throws IOException {
    PrintWriter pWriter = new PrintWriter(new FileWriter("test5.txt"));
    pWriter.write("bingbing");
    pWriter.close();
  }
}
```

4.6. 编码问题(了解)

4.6.1. 开发中的编码

- 常用字符集
  - 中国的字符集:GBK/GB2312
  - 欧洲的:ISO8859-1
  - 通用的:UTF-8
  - 美国的:ASCII
- 对中文的处理
  - 一个汉字:GBK:2个字节 ISO8859-1:1个字节 utf-8:3个字节 unicode:2个字节(内部编码)
  - 说明:GBK,UTF-8是支持中文的,ISO8859-1不支持中文
- 编码:将字符串转化成byte序列的过程

- 解码:是将byte序列转成字符串的过程
- 编码错误:乱码:在执行读与写的时候,由于使用的字符集不同,造成了编码的错误.

解决办法:再编码再解码

- 常用编码解码方法

```
编码:

byte[] getBytes() //对于中文  默认的格式
使用平台的默认字符集将此 String 编码为 byte 序列,
并将结果存储到一个新的 byte 数组中。

byte[] getBytes(Charset charset)
使用给定的 charset 将此 String 编码到 byte 序列，并将结果存储到新的 byte 数组。

解码:

String(byte[] bytes) //对于中文  默认是格式
通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。

String(byte[] bytes, Charset charset)
通过使用指定的 charset 解码指定的 byte 数组，构造一个新的 String。

```

- 示例代码

```

public class Demo10 {

```

```
3       public static void main(String[] args) throws
    UnsupportedEncodingException {
4           //使用GBK编码,GBK解码
5           String str1 = "你好";
6           byte[] str1b = str1.getBytes("GBK");
7           System.out.println(new String(str1b,"GBK"));//
    你好
8
    System.out.println(Arrays.toString(str1b));//[-60, -29,
    -70, -61]
9           //使用utf-8编码,utf-8解码
10          String str2 = "你好";
11          byte[] str2b = str2.getBytes("utf8");
12          System.out.println(new String(str2b,"utf8"));//
    你好
13
    System.out.println(Arrays.toString(str2b));//[-28, -67,
    -96, -27, -91, -67]
14          //使用ISO8859-1编码,解码
15          String str3 = "你好";
16          byte[] str3b = str3.getBytes("ISO8859-1");
17          System.out.println(new String(str3b,"ISO8859-
    1"));//??
18
    System.out.println(Arrays.toString(str3b));//[63, 63]
19      }
20  }
```

4.6.2. 中文乱码处理

- 注意点:乱码解决的办法是再编码再解码

  但是如果是编码出错了,无法解决.

如果是解码出错了,可以利用再编码再解码

- 使用常用字符集GBK,utf8,ISO8859-1,进行编码解码出现乱码的情况分析

```
/* 编码            解码            结果
 * GBK             utf8            不可以(GBK2个字节,utf83个字节)
 * GBK             ISO8859-1       可以
 * utf8            GBK             有时可以
 * utf8            ISO8859-1       可以
 * ISO8859-1         GBK           不可以(编码就出错了)
 * ISO8859-1         utf8          不可以(编码就出错了)
 */
```

- **总结:只有使用GBK或utf8进行编码,使用ISO8859-1进行解码,才能使用再编码再解码解决乱码问题**
- 示例代码

```
public class Demo6 {
    public static void main(String[] args) throws UnsupportedEncodingException {
        //1.GBK编码          utf8解码
//    String s1 = "你好";
//    byte[] s1b = s1.getBytes("GBK");
//    String sr1b = new String(s1b,"utf-8");
//    System.out.println("utf8解码:"+sr1b);//???
//
//    //再编码
//    byte[] s1bb = sr1b.getBytes("utf-8");
//    //再解码
//    System.out.println("GBK再编码:"+new String(s1bb,"GBK"));//锟斤拷锟?


```

```
15          //2.GBK编码          ISO8859-1解码
16  //      String s1 = "你好";
17  //      byte[] s1b = s1.getBytes("GBK");
18  //      String sr1b = new String(s1b,"ISO8859-1");
19  //      System.out.println("ISO8859-1解码:"+sr1b);//????
20  //
21  //      //再编码
22  //      byte[] s1bb = sr1b.getBytes("ISO8859-1");
23  //      //再解码
24  //      System.out.println("GBK再编码:"+new
    String(s1bb,"GBK"));//你好
25
26          //3.utf8编码          GBK解码
27          String s1 = "你好吧";
28          byte[] s1b = s1.getBytes("utf-8");
29          String sr1b = new String(s1b,"GBK");
30          System.out.println("GBK解码:"+sr1b);//浣犲ソ鍚?
31
32          //再编码
33          byte[] s1bb = sr1b.getBytes("GBK");
34          //再解码
35          System.out.println("utf8再编码:"+new
    String(s1bb,"utf-8"));//你好??
36
37
38          //4.utf8编码          ISO8859-1解码
39  //      String s1 = "你好号";
40  //      byte[] s1b = s1.getBytes("utf-8");
41  //      String sr1b = new String(s1b,"ISO8859-1");
42  //      System.out.println("ISO8859-1解
    码:"+sr1b);//????????·
43  //
44  //      //再编码
```

```
45 //     byte[] s1bb = sr1b.getBytes("ISO8859-1");
46 //     //再解码
47 //     System.out.println("utf8再编码:"+new
   String(s1bb,"utf-8"));//你好号
48
49     }
50 }
```

4.6.3. 转换流的编码问题演示

- 由于转换流实现的是字节流和字符流之间的转换,所以会存在编码问题
- 示例代码
  - 分析:写入数据---"冰冰"
  - 首先使用GBK编码,utf-8解码
  - 然后使用utf-8编码,GBK解码
- 代码运行结果

```
readData1:冰冰
readData2:鎰板暘
readData3:����
readData4:冰冰
```

- 代码运行结果分析

这里使用的是utf8和GBK进行的编码解码测试,只有在编码和解码使用同一种字符集时,才是正确的编码方法

```
1 public class Demo4 {
2     public static void main(String[] args) throws
  IOException {
3         //实例:写入数据---"冰冰"
4         //写的时候采用utf-8编码
```

```
5           writeData1();
6           readData1();//使用utf-8字符集
7           readData2();//使用GBK字符集
8
9
10          //写的时候采用GBK编码
11          writeData2();
12          readData3();//使用utf-8字符集
13          readData4();//使用GBK字符集
14      }
15
16      //写出
17      //编码格式是utf8
18      public static void writeData1() throws IOException
    {
19          //创建输出流并关联文件    第一个参数是字节输出流      第
    二个参数是:输出时指定的编码格式
20          OutputStreamWriter outputStreamWriter = new
    OutputStreamWriter(new
    FileOutputStream("BigDataBK2001N21/utf8.txt"),"utf-8");
21          outputStreamWriter.write("冰冰");
22          outputStreamWriter.close();
23      }
24
25      //写出
26      //编码格式是GBK
27      public static void writeData2() throws IOException
    {
28          //创建输出流并关联文件    第一个参数是字节输出流      第
    二个参数是:输出时指定的编码格式
```

```
29          OutputStreamWriter outputStreamWriter = new
OutputStreamWriter(new
FileOutputStream("BigDataBK2001N21/GBK.txt"),"GBK");//
默认GBK
30          outputStreamWriter.write("冰冰");
31          outputStreamWriter.close();
32      }
33
34      //读入
35      //编码格式是utf8
36      public static void readData1() throws IOException {
37          InputStreamReader inputStreamReader = new
InputStreamReader(new
FileInputStream("BigDataBK2001N21/utf8.txt"),"utf-8");
38          char[] arr = new char[100];
39          int num = inputStreamReader.read(arr);
40          System.out.println("readData1:"+new
String(arr,0,num));
41
42          inputStreamReader.close();
43      }
44
45      //读入
46      //编码格式是GBK
47      public static void readData2() throws IOException {
48          InputStreamReader inputStreamReader = new
InputStreamReader(new
FileInputStream("BigDataBK2001N21/utf8.txt"),"GBK");//
默认GBK
49          char[] arr = new char[100];
50          int num = inputStreamReader.read(arr);
51          System.out.println("readData2:"+new
String(arr,0,num));
```

```
52
53          inputStreamReader.close();
54      }
55
56      //读入
57      //编码格式是utf8
58      public static void readData3() throws IOException {
59          InputStreamReader inputStreamReader = new
    InputStreamReader(new
    FileInputStream("BigDataBK2001N21/GBK.txt"),"utf-8");
60          char[] arr = new char[100];
61          int num = inputStreamReader.read(arr);
62          System.out.println("readData3:"+new
    String(arr,0,num));
63
64          inputStreamReader.close();
65      }
66
67      //读入
68      //编码格式是utf8
69      public static void readData4() throws IOException {
70          InputStreamReader inputStreamReader = new
    InputStreamReader(new
    FileInputStream("BigDataBK2001N21/GBK.txt"),"GBK");//默
    认GBK
71          char[] arr = new char[100];
72          int num = inputStreamReader.read(arr);
73          System.out.println("readData4:"+new
    String(arr,0,num));
74
75          inputStreamReader.close();
76      }
77  }
```

```
78
```

4.7. 序列化流(会)

4.7.1. 简介

- 将短期存储的数据实现长期存储,这个过程对应的流就是序列化流
- 数据的存储分成两类:
  - 1.短期存储:存放在内存中,随着程序的关闭而释放---对象,集合,变量,数组
  - 2.长期存储:存储在磁盘中,即使程序关闭了,数据仍然存在------文件
- 序列化:将数据从内存放入磁盘,可以实现数据的长久保存
- 反序列化:将数据从磁盘放回内存

4.7.2. 注意事项

- ObjectInputStream、 ObjectOutputStream， 主要是用来做对象的序列化和反序列化的。
- 序列化、 反序列化， 是对象的持久化存储的一种常用手段。
- 所有的要序列化到本地的类的对象， 类必须实现 `java.io.Serilizable` 接口。

```
实现了Serializable接口的类可以达到的目的:
1.可以进行序列化
2.进行序列化的类的元素都必须支持序列化
3.可序列化类的所有子类型本身都是可序列化的。
4.接口本身没有方法或字段,只是用来表示可序列化的语义
```

- 如果需要序列化多个文件到本地， 尽量不要序列化到一个文件中。 如果需要序列化多个文件到本地， 通常采用的方式， 是存集合。 将多个对象存入一个集合中， 将这个集合序列化到本地。

- 常见问题总结

```
 注意点:
1. ClassNotFoundException:当前的类没有找到
分析:将Person对象进行序列化之后,将Person类删除,再进行反序列化的时候出现了异常
原因:反序列化在执行的时候依赖字节码文件,当类没有了,字节码文件无法创建,反序列化失败

2.java.io.InvalidClassException  无效的类
出现的原因:没有声明自己的serialVersionUID,而使用系统的.在进行反序列化的时候,类被改动了,系统认为现在的类
已经不是原来的类了(在使用系统的id进行识别的时候,重写给Person设置了id),认为此类无效

3.使用系统的serialVersionUID与自定义的ID的区别?
使用系统的,序列化和反序列化,id不能手动设置,使用的是编译器默认生成的,一旦类发生了改动,id会重新赋值
使用自定义的,序列化和反序列化,id不会发生改变,所以当反序列化的时候,即使对Person类进行了一些改动,也能继续反序列化
```

4.7.3. 示例代码

- 创建Person对象,并进行序列化处理

  要想让Person类的对象可以实现序列化,必须让Person类实现Serializable接口

  类通过实现 java.io.Serializable 接口以启用其序列化功能。

  未实现此接口的类将无法使其任何状态序列化或反序列化。

注意:不仅要求当前类可序列化,而且要求当前类的所有元素本身都是可序列化的(比如:ArrayList)

```
1
2  class  Person implements  Serializable{
3      //   /**
4      //     * generated:由编译器自动生成的,后面加L表示long型
5      //     */
6      private static final long serialVersionUID =
   -7224641225172644265L;
7      //    /**
8      //     * default:UID是由用户自己指定的,默认值是1L]
9      //     */
10     //    private static final long serialVersionUID =
   1L;
11     String name;
12     int age;
13     double height;
14
15     public Person(String name, int age) {
16         this.name = name;
17         this.age = age;
18     }
19
20     public  void  show(){
21         System.out.println("show-oo");
22     }
23
24     @Override
25     public String toString() {
26         return "Person{" +
27                 "name='" + name + '\'' +
28                 ", age=" + age +
```

```
                '}';
    }

    public void  run(){
        System.out.println("run");
    }
}
//可序列化类的所有子类型本身都是可序列化的。
class  GoodPerson extends  Person{
    double weight;

    public GoodPerson(String name, int age, double weight) {
        super(name, age);
        this.weight = weight;
    }
}

```

- 将Person对象序列化

```
public class Demo13 {
    static Person p1;
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        //序列化
        //fun1();
        //逆序列化
        fun2();
    }
    //序列化
    public  static  void  fun1() throws IOException {
        //创建序列化流

```

```
12          ObjectOutputStream objectOutputStream = new
    ObjectOutputStream(new
    FileOutputStream("D:\\ideaProgram\\BigDataBK2001N06\\Bi
    gDataBK2001N20\\copyDemo5.java"));
13          //objectOutputStream.writeInt(2);
14
15          //对对象进行序列化
16          p1 = new Person("zhangsan",20);
17          objectOutputStream.writeObject(p1);
18          //可序列化类的所有子类型本身都是可序列化的。
19          //objectOutputStream.writeObject(new
    GoodPerson("hah",20,20));
20          //序列化后要及时关闭流
21          objectOutputStream.close();
22      }
23      //逆序列化
24      public static  void  fun2() throws IOException,
    ClassNotFoundException {
25          ObjectInputStream objectInputStream = new
    ObjectInputStream(new
    FileInputStream("D:\\ideaProgram\\BigDataBK2001N06\\Big
    DataBK2001N20\\copyDemo5.java"));
26  //          int num = objectInputStream.readInt();
27  //          System.out.println(num);
28
29          //反序列化
30          Object o = objectInputStream.readObject();
31          System.out.println(o == p1);//false  说明反序列化
    后的对象与原来的对象是两块儿空间.
32          System.out.println(o);
33          //objectInputStream.close();
34      }
35  }
```

4.7.4. 总结

- 合理使用序列化流和反序列化流,要与输入流与输出流配合使用
- 进行序列化的类一定要实现Serializable接口,只要实现了接口就可以序列化.包括集合,包装类等
- 进行序列化的类要保证当前类与内部的类都要实现Serializable接口

4.8. Properties(了解)

4.8.1. 简介

- 概述

Properties也不是一个IO流， 是一个集合。 是Hashtable的子类。

使用Properties主要是为了描述程序中的属性列表文件。 有时候， 我们会将一些比较简单的项目的配置信息， 以 .properties 格式的文件进行存储。可以使用Properties对象读写 .properties 文件。

- 注意

因为存储的是属性,属性本来就是以键值对的方式存储.这里的键和值都必须是字符串.所以不需要考虑泛型

- 为什么要在这里讲Properties?

因为他的使用与流紧密相关

- Properties作用

1.是HashTable的子类,所以也是键值对形式,保存,操作更容易

2.默认键值都是字符串,所以不需要再考虑泛型

3.提供了一批好用的方法,方便使用(load(),store(),list()等)

4.8.2. 基本使用

```
    public  static  void  fun1(){
        Properties properties = new Properties();
        properties.setProperty("first","java");
        properties.setProperty("second","php");
        properties.setProperty("third","python");
        System.out.println(properties);//重写了toString
方法
        properties.setProperty("first","haha");//key是唯
一的,后面的值会将前面的值覆盖
        System.out.println(properties.get("first"));
        //当前的key在Properties中不存在时,会打印c

System.out.println(properties.getProperty("first1","c")
);
        System.out.println(properties);
    }
```

4.8.3. 获取系统属性

```
public  static  void  fun2(){
      //获取系统属性
        Properties properties = System.getProperties();
        //System.out.println(properties);
      //获取所有键的名字
        Set<String> set =
properties.stringPropertyNames();
       //遍历得到值
        Iterator<String> iterator = set.iterator();
        while (iterator.hasNext()){
            String key = iterator.next();
```

```
            System.out.println("key:"+key+"
value:"+properties.getProperty(key));
        }

        //第一次修改属性信息
        properties.setProperty("user.language","ch");

System.out.println(properties.getProperty("user.languag
e"));//ch

        //再获取一次属性信息
        //思考?经过第一次的修改,我们发现user.language的值变成
了ch,再次获取属性,得到的仍然是ch
       //系统属性可以这样很轻易的更改吗?
        //原理:会先到内存中找属性集合的对象,如果有,直接使用.如果
没有,会重新初始化一个新的对象,并获取属性集合.
        Properties properties1 =
System.getProperties();

System.out.println(properties1.getProperty("user.langua
ge"));//ch
    }
```

4.8.4. 实际应用

加载一个 .properties 文件中的数据,后缀名也可以不是properties.但是一般我们写成properties,方便使用.

- a.properties文件内容

注意:

1.这里是修改过一次的文件

2.Propertes文件中对应的也应该是键值对

3.键和值之间可以是=或者空格或者冒号

4.默认每行只写一个键值对

```
#\u6539\u53D8\u4E86\u51B0\u51B0\u7684\u503C
#Fri Oct 16 17:52:51 IRKT 2020
name=我们
wss|06=
zhaoliu,05=
zhangsan=02
lisi=03
bingbing=buok
zhaoliu=04
```

- 读取数据/遍历数据/修改数据/写回数据

```
 public  static  void  fun3() throws IOException {
        //读取a.txt的内容到Properties对象里面
        Properties properties = new Properties();
        //利用load方法将内容从磁盘读入Properties对象
        properties.load(new
FileReader("D:\\ideaProgram\\BigDataBK2001N06\\a.txt"))
;
        //System.out.println(properties);

        //改变内容
        properties.setProperty("bingbing","buok");
        //使用store方法将数据写入磁盘
        properties.store(new
FileWriter("D:\\ideaProgram\\BigDataBK2001N06\\a.txt"),
"改变了冰冰的值");

```

```
13          //写入控制台---通过list
14          properties.list(System.out);
15      }
```