day11【Properties类、缓冲流、转换流、序列化流、装饰者模式、commons-io工具包】

今日内容

- IO异常处理---->必须掌握
  - jdk1.7之前的IO异常处理
  - jdk1.7之后的IO异常处理
- 属性集--Properties类
  - 结合流加载配置文件------>必须掌握
- 高级流
  - 缓冲流------>必须掌握
  - 转换流------>必须掌握
  - 序列化流和反序列化流
  - 打印流
- 装饰者模式------>必须掌握------>难点
- commons-io工具包------>必须掌握
  - 使用步骤
  - commons-io的api

第一章 IO资源的处理

1.1 JDK7前处理

- 概述:JDK7 之前处理异常的方式是使用 try...catch...finally 的方式。
- 程序:

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

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 8:50
 */
public class Test {
    public static void main(String[] args) {
        // jdk7之前:  try...catch...finally
        // 快捷键:  选中代码--->然后ctrl+alt+t
        FileInputStream fis = null;
        FileOutputStream fos = null;
        try {
            // 1.创建字节输入流对象,关联数据源文件路径
            fis = new FileInputStream("day11\\aaa\\hb.jpg");
```

```
            // 2.创建字节输出流对象,关联目的地文件路径
            fos = new FileOutputStream("day11\\aaa\\hbCopy1.jpg");

            // 3.定义一个byte数组,用来存储读取到的字节数据
            byte[] bys = new byte[8192];

            // 3.定义一个int类型的变量,用来存储读取到的字节个数
            int len;

            // 4.循环读取数据
            while ((len = fis.read(bys)) != -1) {
                // 5.在循环中,写出数据
                fos.write(bys, 0, len);
            }

        } catch (IOException e) {
            System.out.println("出现了异常,异常的信息是:" + e.getMessage());
        } finally {
            // 一般放释放资源的代码
            // 6.释放资源
            try {
                if (fos != null) {
                    fos.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    if (fis != null) {
                        fis.close();
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

    }
}
```

1.2 JDK7后处理

- 概述:JDK7 之后可以使用优化后的 try-with-resource 语句来处理异常，该语句确保了**每个资源在语句结束时自动关闭**。所谓的资源（resource）是指在程序完成后，必须关闭的对象。
- 格式:

```
try(创建流对象的语句,以分号隔开){

}catch(异常类型 变量名){

}
```

- 程序:

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

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 8:50
 */
public class Test {
    public static void main(String[] args) {
        // jdk7之后:  try...with...resource
        try (
                // 1.创建字节输入流对象,关联数据源文件路径
                FileInputStream fis = new
FileInputStream("day11\\aaa\\hb.jpg");

                // 2.创建字节输出流对象,关联目的地文件路径
                FileOutputStream fos = new
FileOutputStream("day11\\aaa\\hbCopy1.jpg");
        ) {

            // 3.定义一个byte数组,用来存储读取到的字节数据
            byte[] bys = new byte[8192];

            // 3.定义一个int类型的变量,用来存储读取到的字节个数
            int len;

            // 4.循环读取数据
            while ((len = fis.read(bys)) != -1) {
                // 5.在循环中,写出数据
                fos.write(bys, 0, len);
            }

        } catch (IOException e) {
            System.out.println("发生了异常,异常的信息:" + e.getMessage());
        }
    }
}
```

第二章 属性集

2.1 Properties类

- 概述:
  - 1.Properties类继承Hashtable,而Hashtable又实现Map接口,所以Properties类其实就是一个双列集合(Map集合),拥有Map集合里面的所有方法,但一般不把Properties当成Map集合使用
  - 2.把Properties类当成属性集使用,结合流去加载配置文件中的数据,属性集中的键和值的类型都是String类型

- 3.要求:
  - 配置文件中的数据是以键值对的形式存储
  - 配置文件中一般没有中文
  - 配置文件后缀一般是`.properties`
- 构造方法:
  - `public Properties();创建一个空的属性集`
- 常用方法:

```
public void load(InputStream is);加载配置文件中的键值对,存储到Properties对象中
public void load(Reader r);加载配置文件中的键值对,存储到Properties对象中
public Set<String> stringPropertyNames() 所有键的名称的集合。--->Map:keySet()
public String getProperty(String key)  使用此属性列表中指定的键搜索属性值。-->Map:
get(K k)

public Object setProperty(String key, String value) 保存一对属性。--->Map:
put(K k,V v) public void store(OutputStream out, String comments) 把
Properties类中的键值对数据写回文件中public void store(Writer writer, String
comments) 把Properties类中的键值对数据写回文件中
```

- 程序1: 加载配置文件中的数据到Properties对象中,并取出所有数据

```
配置文件:
k1=v1
k2=v2
k3=v3
k4=v4
```

```
public class Test1_加载配置文件 {
    public static void main(String[] args) throws Exception {
        // 需求: 加载配置文件中的数据到Properties对象中,并取出所有数据
        // 1.创建Properties对象
        Properties pro = new Properties();

        // 2.调用load()方法加载配置文件中的键值对到Properties对象中
        pro.load(new FileInputStream("day11\\aaa\\a.txt"));// -->如果文件中有
中文,就会乱码
        //pro.load(new FileReader("day11\\aaa\\a.txt"));// --->如果文件中有中
文,不会乱码

        // 3.获取所有的键
        Set<String> keys = pro.stringPropertyNames();

        // 4.循环遍历所有的键
        for (String key : keys) {
            // 5.在循环中,根据键找值,打印输出
            String value = pro.getProperty(key);
            System.out.println(key + " = " + value);
        }
    }
}
```

- 程序2: 将配置文件中k3键对应的值改为value3

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.util.Properties;
import java.util.Set;

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 9:53
 */
public class Test2_修改配置文件 {
    public static void main(String[] args) throws Exception {
        // 需求:将配置文件中k3键对应的值改为value3
        // 1.创建Properties对象
        Properties pro = new Properties();

        // 2.调用load方法加载配置文件
        pro.load(new FileInputStream("day11\\aaa\\b.properties"));

        // 3.获取所有的键
        Set<String> keys = pro.stringPropertyNames();

        // 4.循环遍历所有的键
        for (String key : keys) {
            // 5.在循环中,判断遍历出来的键是否是k3,如果是,就修改对应的值
            if ("k3".equals(key)) {
                pro.setProperty(key, "value3");
            }
        }

        // 6.调用store()方法把Properties对象中所有的键值对写回到文件中-->覆盖之前文件
的所有数据
        //pro.store(new
FileOutputStream("day11\\aaa\\b.properties"),"szitheima113");
        pro.store(new
FileWriter("day11\\aaa\\b.properties"),"szitheima113");
    }
}
```

第三章 缓冲流

3.1 缓冲流的概述

- 概述: 缓冲流,也叫高效流，是对4个基本的 FileXxx 流的增强，所以也有4个流,按照类型分;
  - 字节缓冲流:
    - 字节缓冲输入流: `BufferedInputStream`
    - 字节缓冲输出流: `BufferedOutputStream`
  - 字符缓冲流:

- 字符缓冲输入流: `BufferedReader`
- 字符缓冲输出流: `BufferedWriter`

3.2 字节缓冲流

- 概述: 字节缓冲流指的就是 `BufferedInputStream`, `BufferedOutputStream`,他们没有新的api,依然使用的是`InputStream`, `OutputStream`读写的api,但这些api都增强了,所以他的特殊功能就是读写效率高
- 构造方法:
  - BufferedInputStream: `public BufferedInputStream(InputStream is);`
  - BufferedOutputStream: `public BufferedOutputStream(OutputStream is);`
- 效率测试程序:
  - 普通字节流读写一个字节拷贝文件

```
public class Test1_普通字节流读写一个字节拷贝文件 {
    public static void main(String[] args) throws Exception {
        // 0.获取当前系统时间距离标准基准时间的毫秒值
        long start = System.currentTimeMillis();

        // 1.创建字节输入流对象,关联数据源文件路径
        FileInputStream fis = new
FileInputStream("day11\\aaa\\jdk9.exe");

        // 2.创建字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\aaa\\jdk9Copy1.exe");

        // 3.定义一个int变量,用来存储读取到的字节数据
        int len;

        // 4.循环读取字节数据
        while ((len = fis.read()) != -1) {
            // 5.在循环中,写出字节数据
            fos.write(len);
        }

        // 6.释放资源
        fos.close();
        fis.close();

        // 7.获取当前系统时间距离标准基准时间的毫秒值
        long end = System.currentTimeMillis();
        System.out.println("总共花了:" + (end - start) + "毫秒");// 大概需
要十几分钟
    }
}
```

  - 字节缓冲流读写一个字节拷贝文件

```
public class Test2_字节缓冲流读写一个字节拷贝文件 {
    public static void main(String[] args) throws Exception {
        // 0.获取当前系统时间距离标准基准时间的毫秒值
        long start = System.currentTimeMillis();

        // 1.创建字节缓冲输入流对象,关联数据源文件路径
        FileInputStream fis = new
FileInputStream("day11\\aaa\\jdk9.exe");
        BufferedInputStream bis = new BufferedInputStream(fis);

        // 2.创建字节缓冲输出流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\aaa\\jdk9Copy2.exe");
        BufferedOutputStream bos = new BufferedOutputStream(fos);

        // 3.定义一个int变量,用来存储读取到的字节数据
        int len;

        // 4.循环读取字节数据
        while ((len = bis.read()) != -1) {
            // 5.在循环中,写出字节数据
            bos.write(len);
        }

        // 6.释放资源
        bos.close();
        bis.close();

        // 7.获取当前系统时间距离标准基准时间的毫秒值
        long end = System.currentTimeMillis();
        System.out.println("总共花了:" + (end - start) + "毫秒");// 大概需
要31秒
    }
}
```

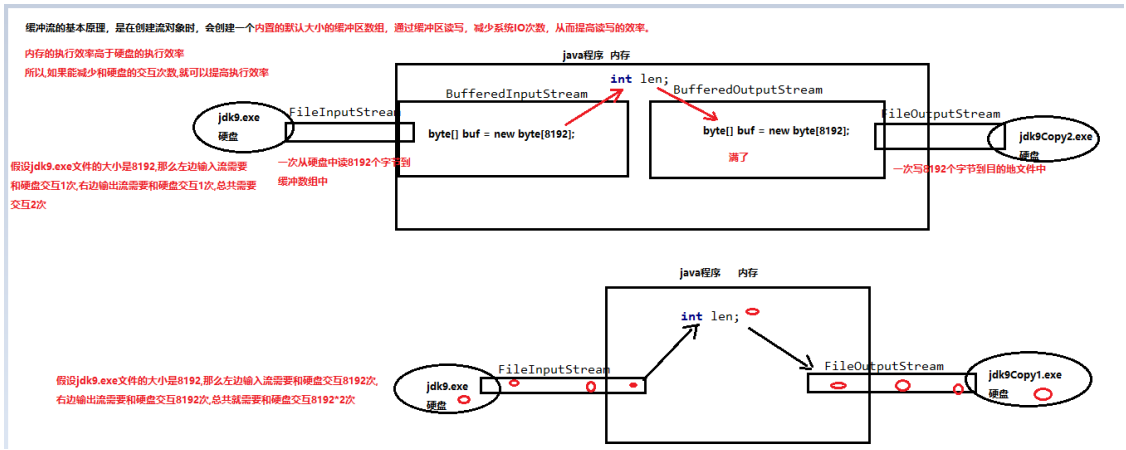
- 字节缓冲流读写一个字节数组拷贝文件

```
public class Test3_字节缓冲流读写一个字节数组拷贝文件 {
    public static void main(String[] args) throws Exception {
        // 0.获取当前系统时间距离标准基准时间的毫秒值
        long start = System.currentTimeMillis();

        // 1.创建字节缓冲输入流对象,关联数据源文件路径
        FileInputStream fis = new
FileInputStream("day11\\aaa\\jdk9.exe");
        BufferedInputStream bis = new BufferedInputStream(fis);

        // 2.创建字节缓冲输出流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\aaa\\jdk9Copy3.exe");
        BufferedOutputStream bos = new BufferedOutputStream(fos);

        // 3.定义一个byte数组,用来存储读取到的字节数据
        byte[] bys = new byte[8192];
        // 3.定义一个int变量,用来存储读取到的字节个数
        int len;
```

```
            // 4.循环读取字节数据
            while ((len = bis.read(bys)) != -1) {
                // 5.在循环中,写出字节数据
                bos.write(bys,0,len);
            }

            // 6.释放资源
            bos.close();
            bis.close();

            // 7.获取当前系统时间距离标准基准时间的毫秒值
            long end = System.currentTimeMillis();
            System.out.println("总共花了:" + (end - start) + "毫秒");// 大概需要3秒
        }
    }
```

3.3 缓冲流读写效率高的基本原理

- 读写高效原理:
  - 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。**
- 画图:



3.4 字符缓冲流

- 概述: 字符缓冲流指的就是 `BufferedReader`,`BufferedWriter`
- 构造方法:
  - BufferedReader: `public BufferedReader(Reader r);`
  - BufferedWriter: `public BufferedWriter(Writer w);`
- 特有方法:
  - BufferedReader: `public String readLine();` 读一行数据,读到文件的末尾返回`null`
  - BufferedWriter: `public void newLine();`写行分隔符,由系统属性定义符号。
- 测试程序:
  - 需求: 每次读一行数据,来读文件中的所有数据

```java
public class Test1_BufferedReader {
    public static void main(String[] args) throws Exception {
        // 1.创建字符缓冲输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day11\\aaa\\b.txt");
        BufferedReader br = new BufferedReader(fr);

        // 2.读数据
        // 2.1 定义一个字符串类型的变量,用来存储读取到的行数据
        String line;

        // 2.2 循环读取行数据
        while ((line = br.readLine()) != null) {
            System.out.println("line:" + line);
        }

        // 3.释放资源
        br.close();

    }
}
```

- 需求: 写一首诗到一个文件中

```java
public class Test2_BufferedWriter {
    public static void main(String[] args) throws Exception{
        // 1.创建字符缓冲输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day11\\aaa\\c.txt");
        BufferedWriter bw = new BufferedWriter(fw);

        // 2.写出数据
        bw.write("今天天气好晴朗");
        bw.newLine();

        bw.write("坐在教室敲代码");
        bw.newLine();

        bw.write("两耳不闻窗外事");
        bw.newLine();

        bw.write("一心只读圣贤书");

        // 3.释放资源
        bw.close();

    }
}
```

3.5 文本排序

需求

请将文本信息恢复顺序。

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。

分析

分析:
1.创建字符缓冲输入流对象,关联数据源文件路径
2.创建ArrayList集合,用来存储每一行字符串数据
3.定义一个String类型的变量,用来存储读取到的行数据
4.循环读取行数据
5.在循环中,把读取到的行数据存储到ArrayList集合中
6.释放资源
7.对ArrayList集合进行升序排序
8.创建字符缓冲输出流对象,关联目的地文件路径
9.循环遍历ArrayList集合
10.在循环中,写出数据到文件中
11.释放资源

实现

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.Collections;

/**
 * @Author：pengzhilin
 * @Date: 2021/4/15 11:26
```

```
 */
public class Test3_文本排序练习 {
    public static void main(String[] args) throws Exception {
        //分析:
        //1.创建字符缓冲输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day11\\aaa\\出师表.txt");
        BufferedReader br = new BufferedReader(fr);

        //2.创建ArrayList集合,用来存储每一行字符串数据
        ArrayList<String> list = new ArrayList<>();

        //3.定义一个String类型的变量,用来存储读取到的行数据
        String line;

        //4.循环读取行数据
        while ((line = br.readLine()) != null) {
            //5.在循环中,把读取到的行数据存储到ArrayList集合中
            list.add(line);
        }

        //6.释放资源
        br.close();


        //7.对ArrayList集合进行升序排序
        // Collections.sort(list)方法对list集合排序,如果list集合中的元素是字符串,会按照字符串的字典顺序进行升序排序
        // eg排序前集合元素: ab,ba,ca,ac,bc
        // eg排序后集合元素: ab,ac,ba,bc,ca
        Collections.sort(list);

        //8.创建字符缓冲输出流对象,关联目的地文件路径
        FileWriter fw = new FileWriter("day11\\aaa\\出师表.txt");
        BufferedWriter bw = new BufferedWriter(fw);

        //9.循环遍历ArrayList集合
        for (String text : list) {
            //10.在循环中,写出数据到文件中
            bw.write(text);
            bw.newLine();
        }
        //11.释放资源
        bw.close();
    }
}
```

第四章 转换流

4.1 字符编码和字符集了解

字符编码的概述

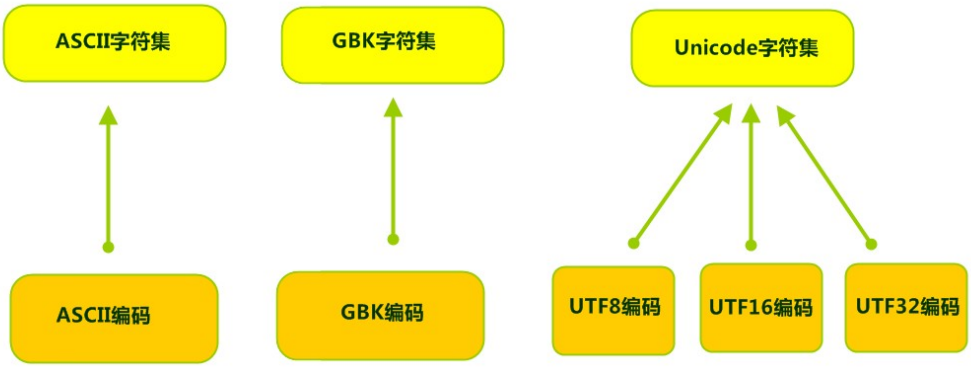
计算机中储存的信息都是用二进制数表示的，而我们在屏幕上看到的数字、英文、标点符号、汉字等字符是二进制数转换之后的结果。按照某种规则，将字符存储到计算机中，称为**编码** 。反之，将存储在计算机中的二进制数按照某种规则解析显示出来，称为**解码** 。比如说，按照A规则存储，同样按照A规则解析，那么就能显示正确的文本f符号。反之，按照A规则存储，再按照B规则解析，就会导致乱码现象。

- **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。**

字符集的概述

- **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。

4.2 编码引出的问题

- 场景: 读文件中的中文
  - 使用字节输入流读中文,一定乱码
  - 使用字符输入流读中文,有可能乱码,有可能不乱码
    - 如果在idea中使用FileReader读utf8编码的中文,不乱码
    - 如果在idea中使用FileReader读gbk编码的中文,就乱码
  - gbk编码下一个中文字符占2个字节
  - utf8编码下一个中文字符占3个字节
  - idea默认编码是utf8编码
- 演示FileReader中文
  - 读gbk编码文件

```
public class Test1_FileReader读gbk编码文件 {
    public static void main(String[] args) throws Exception {
        // 1.创建字符输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day11\\bbb\\gbk.txt");

        // 2.定义一个int类型的变量,用来存储读取到的字符
        int b;

        // 3.循环读取数据
        while ((b = fr.read()) != -1) {
            // 4.在循环中,打印数据
            System.out.println((char) b);// 乱码,因为FileReader在idea中默认utf8编码
        }

        // 5.释放资源
        fr.close();

    }
}
```

  - 读utf8编码文件

```
public class Test1_FileReader读utf8编码文件 {
```

```
    public static void main(String[] args) throws Exception {
        // 1.创建字符输入流对象,关联数据源文件路径
        FileReader fr = new FileReader("day11\\bbb\\utf8.txt");

        // 2.定义一个int类型的变量,用来存储读取到的字符
        int b;

        // 3.循环读取数据
        while ((b = fr.read()) != -1) {
            // 4.在循环中,打印数据
            System.out.println((char) b);// 不乱码,因为FileReader在idea中默认utf8编码
        }

        // 5.释放资源
        fr.close();

    }
}
```

4.3 InputStreamReader类

- 概述: java.io.InputStreamReader类继承Reader类,所以也是字符输入流,可以用来读字符数据
- 作用:
  - 指定编码读数据--->解决编码问题
  - 把字节输入流转换为字符输入流
- 构造方法:
  - `public InputStreamReader(InputStream is);`创建一个转换输入流对象,使用默认字符集
  - `public InputStreamReader(InputStream is,String charsetName);`创建一个转换输入流对象,指定字符集
- 测试案例:
  - 指定gbk编码读gbk编码的文件

```
public class Test1_读gbk编码的文件 {
    public static void main(String[] args) throws Exception{
        // 1.创建转换输入流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day11\\bbb\\gbk.txt");
        InputStreamReader isr = new InputStreamReader(fis,"gbk");

        // 2.定义一个int类型的变量,用来存储读取到的字符
        int b;

        // 3.循环读取数据
        while ((b = isr.read()) != -1) {
            // 4.在循环中,打印数据
            System.out.println((char) b);// 不乱码
        }
```

```
        // 5.释放资源
        isr.close();
    }
}
```

- 指定utf8编码读utf8编码的文件

```
public class Test2_读utf8编码的文件 {
    public static void main(String[] args) throws Exception{
        // 1.创建转换输入流对象,关联数据源文件路径
        FileInputStream fis = new
FileInputStream("day11\\bbb\\utf8.txt");
        InputStreamReader isr = new InputStreamReader(fis,"utf8");

        // 2.定义一个int类型的变量,用来存储读取到的字符
        int b;

        // 3.循环读取数据
        while ((b = isr.read()) != -1) {
            // 4.在循环中,打印数据
            System.out.println((char) b);// 不乱码
        }

        // 5.释放资源
        isr.close();
    }
}
```

4.4 OutputStreamWriter类

- 概述: java.io.OutputStreamWriter类继承Writer类,所以也是字符输出流,可以用来写字符数据
- 作用:
  - 指定编码写数据
  - 把字节输出流转换为字符输出流
- 构造方法:
  - `public OutputStreamWriter(OutputStream os);`创建转换输出流对象,使用默认字符集
  - `public OutputStreamWriter(OutputStream os,String charsetName);`创建转换输出流对象,指定字符集
- 测试程序:
  - 指定gbk编码写出数据

```
public class Test1_指定gbk编码写出数据 {
    public static void main(String[] args) throws Exception{
        // 1.创建转换输出流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\bbb\\gbk_1.txt");
```

```
        OutputStreamWriter osw = new OutputStreamWriter(fos,"gbk");

        // 2.写出数据
        osw.write("中国你好");

        // 3.释放资源
        osw.close();

    }
}
```

- 指定utf8编码写出数据

```
public class Test2_指定utf8编码写出数据 {
    public static void main(String[] args) throws Exception{
        // 1.创建转换输出流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\bbb\\utf8_1.txt");
        OutputStreamWriter osw = new OutputStreamWriter(fos,"utf8");

        // 2.写出数据
        osw.write("中国你好");

        // 3.释放资源
        osw.close();

    }
}
```

4.5 转换文件编码

需求

- 将GBK编码的文本文件，转换为UTF-8编码的文本文件。

分析

1.创建转换输入流对象,指定gbk编码,关联数据源文件路径 2.创建转换输出流对象,指定utf8编码,关联目的地文件路径 3.定义一个int变量,用来存储读取到的字符 4.循环读取 5.在循环中,写出数据 6.释放资源

实现

```
public class Test3_转换文件编码练习 {
    public static void main(String[] args) throws Exception {
        //1.创建转换输入流对象,指定gbk编码,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day11\\bbb\\gbk.txt");
        InputStreamReader isr = new InputStreamReader(fis,"gbk");
```

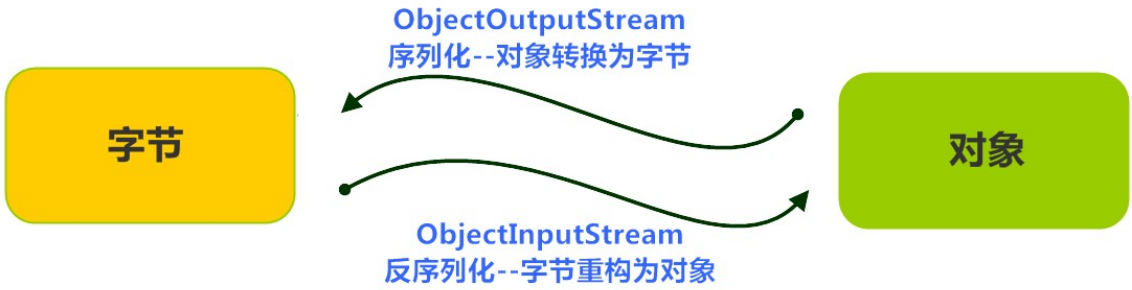
```
        //2.创建转换输出流对象,指定utf8编码,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day11\\bbb\\gbk_utf8.txt");
        OutputStreamWriter osw = new OutputStreamWriter(fos,"utf8");

        //3.定义一个int变量,用来存储读取到的字符
        int c;

        //4.循环读取
        while ((c = isr.read()) != -1) {
            //5.在循环中,写出数据
            osw.write(c);
        }
        //6.释放资源
        osw.close();
        isr.close();

    }
}
```

第五章 序列化

5.1 序列化和反序列化的概念

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：



- **序列化(ObjectOutputStream): 对象---->字节**
- **反序列化(ObjectInputStream): 字节--->对象**

5.2 ObjectOutputStream类

- 概述: java.io.ObjectOutputStream继承OutputStream,所以这个是一个字节输出流
- 作用:
    - 基本功能: 写字节数据到目的地文件中
    - 特殊功能: 写对象到目的地文件中

- 构造方法:
  - `public ObjectOutputStream(OutputStream os);` 创建一个序列化流对象
- 特殊方法:
  - `public void writeObject(Object obj);` 写出一个对象到文件中,持久化保存
- 注意:
  - **被序列化的对象所属的类一定要实现序列化接口(Serializable--->标记接口)**
- 测试程序: 写一个Person对象到a.txt文件中
  - Person类

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Person() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

  - 序列化操作:

```
import com.itheima.bean.Person;

import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
```

```
/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 14:55
 */
public class Test1_写对象 {
    public static void main(String[] args) throws Exception{
        // 需求:写一个Person对象到a.txt文件中
        // 1.创建序列化流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\ccc\\a.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        // 2.创建一个Person对象
        Person p = new Person("张三",18);

        // 3.序列化对象-->把对象写到a.txt文件中
        oos.writeObject(p);// 已经把p对象的所有信息以字节的形式存储到a.txt文件
中

        // 4.释放资源
        oos.close();
    }
}
```

5.3 ObjectInputStream类

- 概述: java.io.ObjectInputStream类继承InputStream,所以是一个字节输入流
- 作用:
  - 普通功能: 读字节数据
  - 特殊功能: 重构对象(把序列化流 序列化的对象 重构出来)
- 构造方法:
  - `public ObjectInputStream(Inputstream is); 创建一个反序列化流对象;`
- 特殊方法:
  - `public Object readObject(); 重构对象`
- 测试程序: 重构上面序列化的Person对象

```
import com.itheima.bean.Person;

import java.io.FileInputStream;
import java.io.ObjectInputStream;

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 15:15
 */
public class Test1_读对象 {
```

```
    public static void main(String[] args) throws Exception{
        // 1.创建反序列化流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day11\\ccc\\a.txt");
        ObjectInputStream ois = new ObjectInputStream(fis);

        // 2.重构对象
        Person p = (Person) ois.readObject();

        // 3.释放资源
        ois.close();

        // 4.打印对象
        System.out.println(p.toString());//Person{name='张三', age=18}
        System.out.println(p);//Person{name='张三', age=18}

    }
}
```

5.4 序列化和反序列化注意事项

- 序列化的注意事项:
  - 被序列化的对象所属的类一定要实现Serializable接口(标记接口)
  - 被序列化的对象所有的属性也是要可以被序列化的
  - 如果被序列化的对象的属性不想被序列化,那么该属性就需要使用transient关键字修饰,表示瞬态
    - Student类

```
public class Student implements Serializable {// 被序列化的对象所属的类一定要实现Serializable接口(标记接口)
    String name;
    // 如果被序列化的对象的属性不想被序列化,那么该属性就需要使用transient关键字修饰,表示瞬态
    transient int age;
    // 被序列化的对象所有的属性也是要可以被序列化的
    Pet pet;

    public Student(String name, int age,Pet pet) {
        this.name = name;
        this.age = age;
        this.pet = pet;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", pet=" + pet +
```

```
                '}';
    }
}
```

- Pet类

```
public class Pet implements Serializable {
    String name;

    public Pet(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Pet{" +
                "name='" + name + '\'' +
                '}';
    }
}
```

- Test类

```
public class Test1_序列化 {
    public static void main(String[] args) throws Exception{
        // 需求:写一个Student对象到b.txt文件中
        // 1.创建序列化流对象,关联目的地文件路径
        FileOutputStream fos = new
FileOutputStream("day11\\ccc\\b.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        // 2.创建一个Pet对象
        Pet p = new Pet("旺财");
        // 2.创建一个Student对象
        Student stu = new Student("李四",19,p);

        // 3.序列化对象-->把对象写到b.txt文件中
        oos.writeObject(stu);// 已经把stu对象的所有信息以字节的形式存储到
b.txt文件中

        // 4.释放资源
        oos.close();
    }
}
```

- 反序列化的注意事项:
  - 1.如果要反序列化成功,一定要能够找到该类的class文件,否则反序列化会失败

- 2.如果能找到该类的class文件,但序列化后又修改了类,这个时候也会反序列化失败-->解决加版本号

```
import java.io.Serializable;

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 15:21
 */
public class Student implements Serializable {// 被序列化的对象所属的类一定要实现Serializable接口(标记接口)
    // 序列化版本号
    static final long serialVersionUID = 2L;

    String name;
    // 如果被序列化的对象的属性不想被序列化,那么该属性就需要使用transient关键字修饰,表示瞬态
    transient int age;
    // 被序列化的对象所有的属性也是要可以被序列化的
    Pet pet;

    // 如果能找到该类的class文件,但序列化后又修改了类,这个时候也会反序列化失败-->解决加版本号
    String sex;

    public Student(String name, int age, Pet pet) {
        this.name = name;
        this.age = age;
        this.pet = pet;
    }

    @Override
    public String toString() {
        return "Student{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", pet=" + pet +
                '}';
    }
}
```

5.5 序列化集合

需求

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。

实现

- 序列化集合:

```
import com.itheima.bean.Person;

import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 15:51
 */
public class Test1_序列化集合 {
    public static void main(String[] args) throws Exception{
        // 1.创建序列化流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day11\\ccc\\list.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        // 2.创建ArrayList集合,限制集合元素的类型为Person类型
        ArrayList<Person> list = new ArrayList<>();

        // 3.往集合中存储多个Person对象
        list.add(new Person("张三",18));
        list.add(new Person("李四",28));
        list.add(new Person("王五",38));
        list.add(new Person("赵六",48));

        // 4.序列化集合
        oos.writeObject(list);// 集合中的元素都会序列化

        // 5.释放资源
        oos.close();

    }
}
```

- 反序列化集合:

```
import com.itheima.bean.Person;

import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.util.ArrayList;

/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 15:54
 */
public class Test2_反序列化集合 {
    public static void main(String[] args) throws Exception {
        // 1.创建反序列化流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day11\\ccc\\list.txt");
```

```
        ObjectInputStream ois = new ObjectInputStream(fis);

        // 2.重构对象
        ArrayList<Person> list = (ArrayList<Person>) ois.readObject();

        // 3.释放资源
        ois.close();

        // 4.循环遍历重构的集合,打印输出
        for (Person p : list) {
            System.out.println(p);
        }

    }
}
```

第六章 打印流

- 概述: java.io.PrintStream类继承OutputStream,也是一个字节输出流
- 作用:
  - 普通功能: 可以用来写出字节数据 write(int len),write(byte[] bys,int off,int len)
  - 特殊功能: 可以方便的打印输出各种类型的数据
- 构造方法:
  - `public PrintStream(String fileName);` 根据文件路径创建打印流对象
- 特殊方法:
  - `public void print(任意类型的数据)` 打印数据到目的地路径中,打印完后不换行
  - `public void println(任意类型的数据)` 打印数据到目的地路径中,打印完后换行
- 测试程序:

```
public class Test1 {
    public static void main(String[] args) throws Exception{
        // 需求: 通过打印流,把数据打印到day11\ddd\a.txt文件中
        // 1.创建打印流对象,关联目的地文件路径
        PrintStream ps = new PrintStream("day11\\ddd\\a.txt");

        // 2.打印数据到文件中
        ps.println(97);
        ps.println(true);
        ps.println(3.14);
        ps.println("黑马程序");
        ps.println('员');

        // 3.释放资源
        ps.close();

        // ----------------------------
        PrintStream ps1 = System.out;
        ps1.println("itcast");
        // 下面这句代码等效于上面2行代码
```

```
        System.out.println("itheima");// 链式编程

    }
}
```

第七章 装饰设计模式

装饰模式概述

装饰模式指的是在不改变原类, 不使用继承的基础上，动态地扩展一个对象的功能。

装饰模式遵循原则:

1. 装饰类和被装饰类必须实现相同的接口
2. 在装饰类中必须传入被装饰类的引用
3. 在装饰类中对需要扩展的方法进行扩展
4. 在装饰类中对不需要扩展的方法调用被装饰类中的同名方法

案例演示

准备环境:

1. 编写一个Star接口, 提供sing 和 dance抽象方法
2. 编写一个LiuDeHua类,实现Star接口,重写抽象方法

```
public interface Star {
    public void sing();
    public void dance();
}
```

```
public class LiuDeHua implements Star {
    @Override
    public void sing() {
        System.out.println("刘德华在唱忘情水...");
    }
    @Override
    public void dance() {
        System.out.println("刘德华在跳街舞...");
    }
}
```

需求:

在不改变原类,不继承重写的基础上对LiuDeHua类的sing方法进行扩展,dace方法不扩展

实现步骤:

```
/**
 * @Author: pengzhilin
 * @Date: 2021/4/15 16:26
```

```
 */
// 装饰类
public class LiuDeHuaWrapper implements Star{

    LiuDeHua ldh;

    public LiuDeHuaWrapper(LiuDeHua ldh) {
        this.ldh = ldh;
    }

    @Override
    public void sing() {
        // 增强
        System.out.println("刘德华在唱忘情水");
        System.out.println("刘德华在唱冰雨");
        System.out.println("刘德华在唱练习");
        System.out.println("刘德华在唱笨小孩...");
        System.out.println("刘德华在...");
    }

    @Override
    public void dance() {
        // 不增强-->执行LiuDeHua里面原有的方法
        ldh.dance();
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        /*
            概述:装饰模式指的是在不改变原类, 不使用继承的基础上, 动态地扩展一个对象的功能。
            使用步骤:
                1.装饰类和被装饰类需要实现同一个接口
                2.装饰类中需要获取被装饰类的引用(被装饰类的对象的地址)
                3.在装饰类中对需要增强的方法进行增强
                4.在装饰类中对不需要增强的方法,就调用被装饰类中原有的方法
         */
        // 创建LiuDeHua对象
        LiuDeHua ldh = new LiuDeHua();
        // 表演
        // ldh.sing();
        // ldh.dance();

        // 创建LiuDeHuaWrapper对象
        LiuDeHuaWrapper ldhW = new LiuDeHuaWrapper(ldh);
        ldhW.sing();
        ldhW.dance();
    }
}
```

第八章 commons-io工具包

- 概述: commons-io 是 apache 开源基金组织提供的一组有关 IO 操作的类库，可以挺提高 IO 功能开发的效率。commons-io 工具包提供了很多有关 io 操作的类。
- api文档:

```
org.apache.commons.io    有关 Streams、Readers、Writers、Files 的工具类。
org.apache.commons.io.input    输入流相关的实现类，包含 Reader 和 InputStream。
org.apache.commons.io.output    输出流相关的实现类，包含 Writer 和 OutputStream。
org.apache.commons.io.serialization    序列化相关的类。
```

- 常用的工具类:

```
IOUtils类:
public static int copy(InputStream in, OutputStream out)
把input输入流中的内容拷贝到output输出流中，返回拷贝的字节个数(适合文件大小为2GB以下)
public static long copyLarge(InputStream in, OutputStream out)
把input输入流中的内容拷贝到output输出流中，返回拷贝的字节个数(适合文件大小为2GB以上)

FileUtils工具类:
public static void copyFileToDirectory( File srcFile,  File destFile)
复制文件到另外一个目录下
public static void copyDirectoryToDirectory(File file1 ,File file2 )
复制 file1 目录到 file2 位置
```

- 使用的步骤:
  - 1.导入commons-io的jar包到模块下
  - 2.把commons-io的jar添加到classpath路径中--->选中jar包,右键,选择add as library
  - 3.使用commons-io的jar包中的工具类
  - ....
- 测试程序:

```
public class Test1_IOUtils工具类 {
    public static void main(String[] args) throws Exception{
        // public static int copy(InputStream in, OutputStream out)
        FileInputStream fis = new FileInputStream("day11\\aaa\\hb.jpg");
        FileOutputStream fos = new
FileOutputStream("day11\\ddd\\hbCopy1.jpg");
        // 使用IOUtils工具类进行拷贝文件
        IOUtils.copy(fis,fos);
        // 释放资源
        fos.close();
        fis.close();

    }
}


public class Test2_FileUtils {
    public static void main(String[] args) throws Exception {
        // public static void copyFileToDirectory( File srcFile,  File
destFile)
```

```
        // 需求: 拷贝文件到指定文件夹下
        File srcF = new File("day11\\aaa\\hb.jpg");// 文件路径
        File destF = new File("day11\\ddd");// 文件夹路径

        FileUtils.copyFileToDirectory(srcF, destF);

    }
}


public class Test3_FileUtils {
    public static void main(String[] args) throws Exception{
        // public static void copyDirectoryToDirectory(File file1 ,File
file2 )
        // 需求: 拷贝文件夹到指定文件夹下
        File srcF = new File("day11\\bbb");
        File destF = new File("day11\\ddd");
        FileUtils.copyDirectoryToDirectory(srcF,destF);

    }
}
```

总结

必须练习:---->要求必须写注释(思路)
    1.IO异常的处理---->jdk7之前;jdk7之后
    2.属性集的使用(Properties)--->课堂里面的2个案例---->3遍以上
    3.使用字节缓冲流一次读写一个字节拷贝文件
    4.使用字节缓冲流一次读写一个字节数组拷贝文件
    5.字符缓冲流: 文本排序案例
    6.转换流: 转换文件编码案例
    7.序列化流: 序列化集合案例
    8.装饰者设计模式的步骤必须能够默写出来
    9.commons-io工具包使用FileUtils拷贝文件和拷贝文件夹的2个案例

- 能够使用Properties的load方法加载文件中配置信息
    构造方法:public Properties();创建一个空的属性集
    常用方法:
public void load(InputStream is);加载配置文件中的键值对,存储到Properties对象中
public void load(Reader r);加载配置文件中的键值对,存储到Properties对象中
public Set<String> stringPropertyNames() 所有键的名称的集合。--->Map:keySet()
public String getProperty(String key) 使用此属性列表中指定的键搜索属性值。-->Map:
get(K k)

public Object setProperty(String key, String value) 保存一对属性。--->Map: put(K
k,V v) public void store(OutputStream out, String comments) 把Properties类中的键值
对数据写回文件中public void store(Writer writer, String comments) 把Properties类中的
键值对数据写回文件中

- 能够使用字节缓冲流读取数据到程序
    BufferedInputStream: public BufferedInputStream(InputStream is);
    特殊功能: 读效率高

- 能够使用字节缓冲流写出数据到文件
  BufferedOutputStream: public BufferedOutputStream(OutputStream is);
  特殊功能: 写效率高

- 能够明确字符缓冲流的作用和基本用法
  - 构造方法:
    - BufferedReader: public BufferedReader(Reader r);
    - BufferedWriter: public BufferedWriter(Writer w);
  - 特有方法:
    - BufferedReader: public String readLine(); 读一行数据,读到文件的末尾返回null
    - BufferedWriter: public void newLine();写行分隔符,由系统属性定义符号。

- 能够使用缓冲流的特殊功能
  字节缓冲流:读写效率高
  字符缓冲流: 读一行,根据系统写行分隔符

- 能够阐述编码表的意义
  定义字符和二进制数之间的对应规则

- 能够使用转换流读取指定编码的文本文件
  public InputStreamReader(InputStream is,String charsetName);创建一个转换输入流对象,指定字符集
  特有功能: 指定编码读数据,可以把字节输入流转换为字符输入流

- 能够使用转换流写入指定编码的文本文件
  public OutputStreamWriter(OutputStream os,String charsetName);创建转换输出流对象,指定字符集
  特有功能: 指定编码写数据,可以把字节输出流转换为字符输出流

- 能够使用序列化流写出对象到文件
  - 构造方法:
    - public ObjectOutputStream(OutputStream os); 创建一个序列化流对象
  - 特殊方法:
    - public void writeObject(Object obj); 写出一个对象到文件中,持久化保存

- 能够使用反序列化流读取文件到程序中
  - 构造方法:
    - public ObjectInputStream(Inputstream is); 创建一个反序列化流对象;
  - 特殊方法:
    - public Object readObject(); 重构对象

- 能够理解装饰模式的实现步骤
  1.装饰类和被装饰类需要实现同一个接口
  2.装饰类中需要获取被装饰类的引用
  3.在装饰类中对需要扩展的方法进行扩展(增强)
  4.在装饰类类中对不需要扩展的方法,就调用被装饰类中同名的方法

- 能够使用commons-io工具包
  1.导入jar包
  2.把jar包添加到classpath路径中--->右键,add as library
  3.使用jar包中的类进行操作...