day13【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
- Properties类
- 缓冲流
- 转换流
- 序列化\反序列化流
- 打印流
- 装饰者模式
- commons-io工具包

教学目标

- [ ] 能够使用Properties的load方法加载文件中配置信息
- [ ] 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
- [ ] 能够理解装饰模式的实现步骤
- [ ] 能够使用commons-io工具包

第一章 IO资源的处理

知识点-- JDK7前处理

目标

- 掌握jdk7之前处理IO异常的方式

路径

- jdk7之前处理IO异常的方式

讲解

之前的入门练习，我们一直把异常抛出，而实际开发中并不能这样处理，建议使用`try...catch...finally`代码块，处理异常部分，代码使用演示：

```
public class Test {
    public static void main(String[] args) {
        // 练习: 拷贝文件
        // jdk7前IO异常处理
        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("day13\\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("day13\\aaa\\hbCopy1.jpg");
            // 3.定义一个字节数组,用来存储读取到的字节数据
            byte[] bys = new byte[8192];
            // 3.定义一个int类型变量,用来存储读取到的字节个数
            int len;
            // 4.循环读取
            while ((len = fis.read(bys)) != -1) {
                // 5.在循环中,写出数据
                fos.write(bys, 0, len);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            // 6.关闭流,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();
                }
            }
        }
    }
}
```

小结

略

知识点-- JDK7的处理

目标

- 掌握jdk7处理IO异常的方式

路径

- jdk7处理IO异常的方式

讲解

还可以使用JDK7优化后的`try-with-resource`语句，该语句确保了每个资源在语句结束时关闭。所谓的资源（resource）是指在程序完成后，必须关闭的对象。

格式：

```
try (创建流对象语句，如果多个,使用';'隔开) {
    // 读写数据
} catch (IOException e) {
    e.printStackTrace();
}
```

代码使用演示：

```
public class HandleException2 {
    public static void main(String[] args) {
        // 创建流对象
        try ( FileWriter fw = new FileWriter("fw.txt"); ) {
            // 写出数据
            fw.write("黑马程序员"); //黑马程序员
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

小结

略

第二章 属性集

知识点-- Properties类

目标

- 掌握Properties类的使用

路径

- Properties类的概述
- Properties类的构造方法
- Properties类存储方法
- Properties类与流相关的方法

讲解

Properties类的概述

`java.util.Properties` 继承于`Hashtable`，来表示一个持久的属性集。它使用键值结构存储数据，每个键及其对应值都是一个字符串。该类也被许多Java类使用，比如获取系统属性时，`System.getProperties` 方法就是返回一个 `Properties` 对象。

Properties类的构造方法

- `public Properties()` :创建一个空的属性列表。

Properties类存储方法

- `public Object setProperty(String key, String value)`： 保存一对属性。
- `public String getProperty(String key)`：使用此属性列表中指定的键搜索属性值。
- `public Set<String> stringPropertyNames()`：所有键的名称的集合。

```
public class ProDemo {
    public static void main(String[] args) throws FileNotFoundException {
        // 创建属性集对象
        Properties properties = new Properties();
        // 添加键值对元素
        properties.setProperty("filename", "a.txt");
        properties.setProperty("length", "209385038");
        properties.setProperty("location", "D:\\a.txt");
        // 打印属性集对象
        System.out.println(properties);
        // 通过键,获取属性值
        System.out.println(properties.getProperty("filename"));
        System.out.println(properties.getProperty("length"));
        System.out.println(properties.getProperty("location"));

        // 遍历属性集,获取所有键的集合
        Set<String> strings = properties.stringPropertyNames();
        // 打印键值对
        for (String key : strings ) {
            System.out.println(key+" -- "+properties.getProperty(key));
        }
    }
}
输出结果:
{filename=a.txt, length=209385038, location=D:\a.txt}
a.txt
209385038
D:\a.txt
filename -- a.txt
length -- 209385038
location -- D:\a.txt
```

Properties类与流相关的方法

- `public void load(InputStream inStream)`： 从字节输入流中读取键值对。
- `public void load(Reader reader)`： 从字节输入流中读取键值对。

参数中使用了字节输入流，通过流对象，可以关联到某文件上，这样就能够加载文本中的数据了。

加载代码演示:

```
public class Test3操作配置文件 {
    public static void main(String[] args) throws Exception{
```

```
        /*
            Properties类操作配置文件:
                1.加载(读取)配置文件中的数据  常见
                    public void load(InputStream inStream):  从字节输入流中读取键值对。
                    public void load(Reader reader):  从字节输入流中读取键值对。

                2.往配置文件中写入数据         不常见
                     void store(OutputStream out, String comments) 把Properties对象中的所有键值对写入配置文件中
                     void store(Writer writer,, String comments) 把Properties对象中的所有键值对写入配置文件中

                注意:
                    1.文本中的数据，必须是键值对形式，可以使用空格、等号、冒号等符号分隔。
                    2.如果配置文件中有中文,那么加载文件文件时,使用字符流,但是开发中一般配置文件中不要写中文
             */
        // 创建Properties对象
        Properties pro = new Properties();

        // 往pro对象中添加键值对
        pro.setProperty("username", "root");
        pro.setProperty("password", "123456");
        pro.setProperty("url", "http://www.itcast.com");

        // 把pro对象中的键值对写入到文件中
        FileOutputStream fos = new FileOutputStream("day13\\aaa\\db2.properties");
        pro.store(fos,"itheima");

        // 关闭流
        fos.close();


    }

    //   1.加载(读取)配置文件中的数据  常见
    private static void method01() throws IOException {
        // 1.加载(读取)配置文件中的数据
        // 创建Properties对象
        Properties pro = new Properties();

        // 调用load方法加载配置文件中的数据
        //FileInputStream fis = new FileInputStream("day13\\aaa\\a.txt");
        FileInputStream fis = new FileInputStream("day13\\aaa\\db.properties");
        pro.load(fis);// 把关联文件中的数据以键值对的形式存储到pro对象中

        // 关闭流,释放资源
        fis.close();

        // 打印pro对象
        System.out.println(pro);

        System.out.println("----------------------------------------------------");
        // 注意事项: 如果配置文件中有中文
        // 创建Properties对象
```

```
        Properties pro2 = new Properties();

        // 调用load方法加载配置文件中的数据
        FileReader fr = new FileReader("day13\\aaa\\a.txt");
        pro2.load(fr);// 把关联文件中的数据以键值对的形式存储到pro对象中

        // 关闭流,释放资源
        fr.close();

        // 打印pro对象
        System.out.println(pro2);
    }
}
```

文件:

```
db.properties:
username=root
password=123456
url=http://www.itcast.com

a.txt:
username=中国
password=123456
url=http://www.itcast.com
```

> 小贴士：文本中的数据，必须是键值对形式，可以使用空格、等号、冒号等符号分隔。

小结

略

第三章 缓冲流

知识点--缓冲流

目标

- 理解缓冲流的概述

路径

- 缓冲流的概述

讲解

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。

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**：`BufferedInputStream`，`BufferedOutputStream`
- **字符缓冲流**：`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。

小结

略

知识点--字节缓冲流

目标

- 掌握字节缓冲流的使用

路径

- 字节缓冲流的构造方法
- 拷贝文件效率测试

讲解

字节缓冲流的构造方法

- `public BufferedInputStream(InputStream in)`：创建一个 新的缓冲输入流。
- `public BufferedOutputStream(OutputStream out)`： 创建一个新的缓冲输出流。

构造举例，代码如下：

```
// 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"));
```

拷贝文件效率测试

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：

```
 // 使用普通字节流读写一个字节拷贝文件
public class BufferedDemo {
    public static void main(String[] args) throws Exception {
        long start = System.currentTimeMillis();
        // 创建字节输入流对象,关联数据源文件路径
        FileInputStream fis = new FileInputStream("day13\\bbb\\jdk9.exe");
        // 创建字节输出流对象,关联目的地文件路径
        FileOutputStream fos = new FileOutputStream("day13\\bbb\\jdk9Copy.exe");
        // 定义一个int类型的变量,用来存储读取到的字节数据
        int len;
        // 循环读取
        while ((len = fis.read()) != -1) {
            // 在循环中,写出数据
            fos.write(len);
        }
        // 关闭流,释放资源
        fos.close();
        fis.close();
        long end = System.currentTimeMillis();
```

```
        System.out.println("总共需要:" + (end - start) + "毫秒");// 至少10几分钟
    }
}
```

2. 缓冲流，代码如下：

```
// 使用高效字节流读写一个字节拷贝文件
public class BufferedDemo {
    public static void main(String[] args) throws Exception {
       long start = System.currentTimeMillis();
        // 创建字节输入流对象,关联数据源文件路径
        BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("day13\\bbb\\jdk9.exe"));
        // 创建字节输出流对象,关联目的地文件路径
        BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("day13\\bbb\\jdk9Copy1.exe"));
        // 定义一个int类型的变量,用来存储读取到的字节数据
        int len;
        // 循环读取
        while ((len = bis.read()) != -1) {
            // 在循环中,写出数据
            bos.write(len);
        }
        // 关闭流,释放资源
        bos.close();
        bis.close();
        long end = System.currentTimeMillis();
        System.out.println("总共需要:" + (end - start) + "毫秒");// 大约32秒
    }
}
```

如何更快呢?

使用数组的方式，代码如下：

```
// 使用高效字节流读写一个数组字节拷贝文件
public class BufferedDemo {
    public static void main(String[] args) throws FileNotFoundException {
        long start = System.currentTimeMillis();
        // 创建字节输入流对象,关联数据源文件路径
        BufferedInputStream bis = new BufferedInputStream(new
FileInputStream("day13\\bbb\\jdk9.exe"));
        // 创建字节输出流对象,关联目的地文件路径
        BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream("day13\\bbb\\jdk9Copy2.exe"));
        // 定义一个字节数组,用来存储读取到的字节数据
        byte[] bys = new byte[8192];
        // 定义一个int类型的变量,用来存储读取到的字节数据
        int len;
        // 循环读取
        while ((len = bis.read(bys)) != -1) {
            // 在循环中,写出数据
            bos.write(bys,0,len);
        }
        // 关闭流,释放资源
        bos.close();
        bis.close();
```

```
        long end = System.currentTimeMillis();
        System.out.println("总共需要:" + (end - start) + "毫秒");// 大约4秒
    }
}
```

小结

略

知识点--字符缓冲流

目标

- 掌握字符缓冲流的使用

路径

- 字符缓冲流的构造方法
- 字符缓冲流的特有方法

讲解

字符缓冲流的构造方法

- `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。

构造举例，代码如下：

```
// 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"));
```

字符缓冲流的特有方法

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：

```
public class BufferedReaderDemo {
    public static void main(String[] args) throws IOException {
         // 创建字符缓冲输入流对象,关联数据源文件路径
        BufferedReader br = new BufferedReader(new
FileReader("day13\\bbb\\b.txt"));

        // 读取一行文字
        //String line1 = br.readLine();
        //System.out.println(line1);

        // 循环读取
        // 定义一个String类型的变量，用来存储读取到的一行文字
```

```
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }

        // 关闭流,释放资源
        br.close();
    }
}
```

`newLine`方法演示，代码如下：

```
public class BufferedWriterDemo throws IOException {
    public static void main(String[] args) throws IOException  {
         // 创建字符缓冲输出流对象，管理目的地文件路径
        BufferedWriter bw = new BufferedWriter(new
FileWriter("day13\\bbb\\c.txt"));
        // 写出数据
        bw.write("戒槟榔");
        bw.newLine();

        bw.write("槟榔加烟，法力无边");
        bw.newLine();

        bw.write("槟榔生吞，道法乾坤");
        bw.newLine();

        bw.write("槟榔配酒，永垂不朽");
        bw.newLine();

        bw.write("槟榔加烟又加酒，阎王在向你招手");
        bw.newLine();

        bw.write("好诗");
        // 关闭流，释放资源
        bw.close();
    }
}
```

小结

略

实操--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

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。

实现

```
public class Test {
    public static void main(String[] args) throws Exception {
        // 需求：请将day13\\bbb\\d.txt文本信息恢复顺序。
        // 实现步骤：
        // 1.创建字符缓冲输入流对象，关联数据源文件路径
        BufferedReader br = new BufferedReader(new
FileReader("day13\\bbb\\d.txt"));

        // 2.创建ArrayList集合，限制集合中元素的类型为String类型
        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.关闭流，释放资源
        br.close();

        for (String s : list) {
            System.out.println(s);
        }

        // 7.使用Collections.sort(List<?> list)方法对ArrayList集合中的元素排序
        // 按照默认规则： String类中的默认规则
```

```
        Collections.sort(list);

        System.out.println("=======排序后================");
        for (String s : list) {
            System.out.println(s);
        }

        // 8.创建字符缓冲输出流对象，关联目的地文件路径
        BufferedWriter bw = new BufferedWriter(new
FileWriter("day13\\bbb\\d.txt"));
        // 9.循环遍历ArrayList集合，获取每一个元素（行数据）
        for (String s : list) {
            // 10.在循环中，把遍历出来的元素（行数据）写入到文件中
            bw.write(s);
            bw.newLine();// 换行
        }
        // 11.关闭流，释放资源
        bw.close();
    }
}
```

小结

略

第四章 转换流

知识点--字符编码和字符集

目标

- 理解字符编码和字符集的概念

路径

- 字符编码的概述
- 字符集的概述

讲解

字符编码的概述

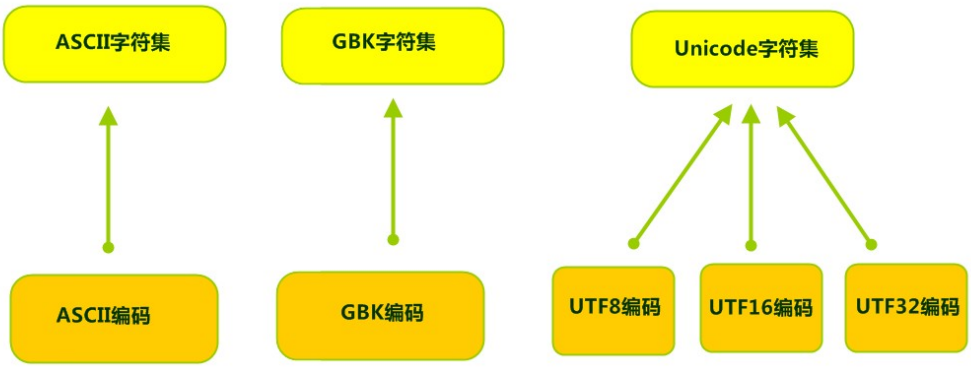
计算机中储存的信息都是用二进制数表示的，而我们在屏幕上看到的数字、英文、标点符号、汉字等字符是二进制数转换之后的结果。按照某种规则，将字符存储到计算机中，称为**编码** 。反之，将存储在计算机中的二进制数按照某种规则解析显示出来，称为**解码** 。比如说，按照A规则存储，同样按照A规则解析，那么就能显示正确的文本符号。反之，按照A规则存储，再按照B规则解析，就会导致乱码现象。

- **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。

小结

略

知识点--编码引出的问题

目标

- 了解编码引出的问题

路径

- 演示编码引出的问题

讲解

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。

```
public class Test {
    public static void main(String[] args) throws Exception{
        /*
            idea默认编码utf8，window有些系统默认的是gbk编码
            问题：  如果使用字符输入流读取含有中文的gbk编码文件,就会出现乱码？
            原因：  因为存的时候使用的是gbk编码，取的时候使用的是utf8编码
         */
        // 创建一个字符输入流对象，关联数据源文件路径
        FileReader fr = new FileReader("day13\\ccc\\gbk.txt");
        // 定义一个int类型的变量，用来存储读取到的字符数据
        int c;
        // 循环读取
        while ((c = fr.read()) != -1){
            System.out.println((char) c);// 乱码
        }
        // 关闭流，释放资源
        fr.close();
    }
}
输出结果：
锟斤拷
```

那么如何读取GBK编码的文件呢？

小结

略

知识点--InputStreamReader类

目标

- 掌握InputStreamReader类 的使用

路径

- InputStreamReader类的概述

- InputStreamReader类的构造方法
- InputStreamReader类指定编码读取

讲解

InputStreamReader类的概述

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。

InputStreamReader类的构造方法

- `InputStreamReader(InputStream in)`：创建一个使用默认字符集的字符流。
- `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：

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

InputStreamReader类指定编码读取

```
public class Test {
    public static void main(String[] args)throws Exception {
        // 创建字符转换输入流对象，关联数据源文件路径  平台默认编码，utf8
        InputStreamReader isr = new InputStreamReader(new
FileInputStream("day13\\ccc\\gbk.txt"));
        // 定义一个int类型的变量，用来存储读取到的字符数据
        int c;
        // 循环读取
        while ((c = isr.read()) != -1){
            System.out.println((char) c);// 乱码
        }
        // 关闭流，释放资源
        isr.close();

        System.out.println("========================================");
        // 创建字符转换输入流对象，关联数据源文件路径  指点编码,gbk
        InputStreamReader isr2 = new InputStreamReader(new
FileInputStream("day13\\ccc\\gbk.txt"),"gbk");
        // 定义一个int类型的变量，用来存储读取到的字符数据
        int c2;
        // 循环读取
        while ((c2 = isr2.read()) != -1){
            System.out.println((char) c2);// 中国
        }
        // 关闭流，释放资源
        isr2.close();
    }
}
```

小结

略

知识点--OutputStreamWriter类

目标

- 掌握OutputStreamWriter类的使用

路径

- OutputStreamWriter类的概述
- OutputStreamWriter类的构造方法
- OutputStreamWriter类指定编码写

讲解

OutputStreamWriter类的概述

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。

OutputStreamWriter类的构造方法

- `OutputStreamWriter(OutputStream in)`: 创建一个使用默认字符集的字符流。 idea默认的是utf8
- `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:

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

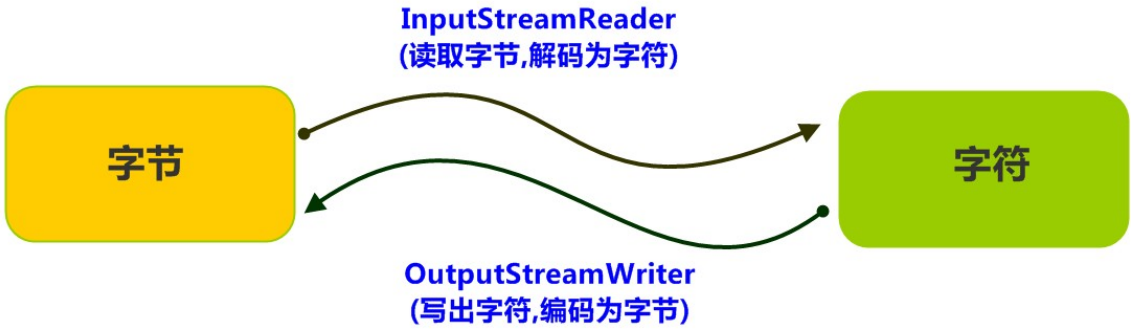
OutputStreamWriter类指定编码读取

```
public class Test {
    public static void main(String[] args) throws Exception{
        // 指点gbk编码写出字符数据
        // 创建字符转换输出流，关联目的地文件路径
        OutputStreamWriter osw = new OutputStreamWriter(new
FileOutputStream("day13\\ccc\\a.txt"),"gbk");
        // 写出数据
        osw.write("中国");// 4个字节
        // 关闭流，释放资源
        osw.close();

        System.out.println("+=======================================");

        // 使用平台默认编码写出字符数据
        // 创建字符转换输出流，关联目的地文件路径
```

```
        OutputStreamWriter osw2 = new OutputStreamWriter(new
FileOutputStream("day13\\ccc\\b.txt"));
        // 写出数据
        osw2.write("中国");// 6个字节
        // 关闭流，释放资源
        osw2.close();
    }
}
```

转换流理解图解

转换流是字节与字符间的桥梁!



小结

略

实操--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. 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。

实现

```
public class Test {
    public static void main(String[] args) throws Exception{
        /*
            需求：将GBK编码的文本文件，转换为UTF-8编码的文本文件。
         */
        // 1.创建转换输入流对象，关联数据源文件路径，指定编码为gbk
        InputStreamReader isr = new InputStreamReader(new
FileInputStream("day13\\ccc\\gbk.txt"),"gbk");
        // 2.创建转换输出流对象，关联目的地文件路径，指点编码为utf8
        OutputStreamWriter osw = new OutputStreamWriter(new
FileOutputStream("day13\\ccc\\utf8.txt"),"utf8");
        // 3.定义一个int类型的变量，用来存放读取到的字符数据
        int len;
        // 4.循环读取
        while ((len = isr.read()) != -1) {
```

```
            // 5.在循环中，写出数据
            osw.write(len);
        }
        // 6.关闭流，释放资源
        osw.close();
        isr.close();
    }
}
```

小结

略

第五章 序列化

知识点--序列化和反序列化的概念

目标

- 理解序列化和反序列化的概念

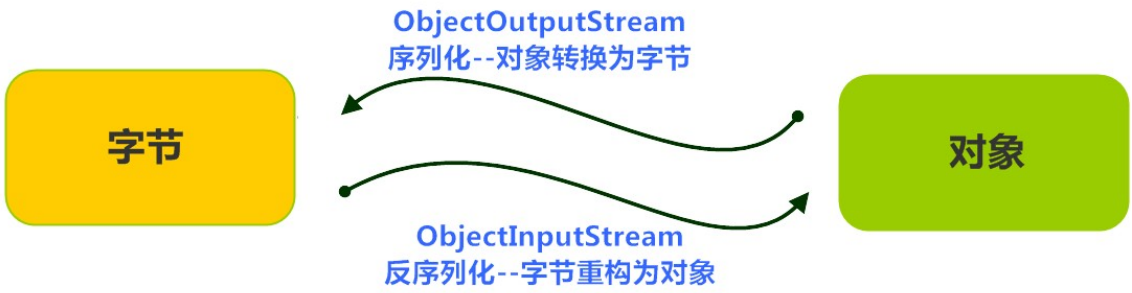
路径

- 序列化
- 反序列化

讲解

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：



小结

略

知识点--ObjectOutputStream类

目标

- 掌握ObjectOutputStream类的使用

路径

- ObjectOutputStream类的概述
- ObjectOutputStream类构造方法
- ObjectOutputStream类序列化操作

讲解

ObjectOutputStream类的概述

`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。

ObjectOutputStream类构造方法

- `public ObjectOutputStream(OutputStream out)`： 创建一个指定OutputStream的ObjectOutputStream。

构造举例，代码如下：

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

ObjectOutputStream类序列化操作

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

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

    public Animal anl;// 宠物

    public Person() {
    }

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

   //...set get ...toString...
}
```

2.写出对象方法

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

```
public class Test {
    public static void main(String[] args)throws Exception {
        // 创建Person对象
        Person p = new Person("张三", 18);
        // 需求：将p对象写入到 day13\\ddd\\a.txt 文件中
        // 创建序列化流对象，关联目的地文件路径
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("day13\\ddd\\a.txt"));
        // 写出对象
        oos.writeObject(p);
        // 关闭流，释放资源
        oos.close();
    }
}
```

小结

略

知识点--ObjectInputStream类

目标

- 掌握ObjectInputStream类的使用

路径

- ObjectInputStream类的概述
- ObjectInputStream类构造方法
- ObjectInputStream类反序列化操作

讲解

ObjectInputStream类的概述

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。

ObjectInputStream类构造方法

- `public ObjectInputStream(InputStream in)`： 创建一个指定InputStream的ObjectInputStream。

ObjectInputStream类反序列化操作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 Test {
    public static void main(String[] args) throws Exception{
        // 反序列化：读取之前序列化的对象
        // 1.创建反序列化流对象，关联数据源文件路径
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("day13\\ddd\\a.txt"));
        // 2.读取一个对象
        Object obj = ois.readObject();
        System.out.println(obj);
        // 3.关闭流，释放资源
        ois.close();
    }
}
```

小结

略

知识点--序列化和反序列化注意事项

目标

- 理解序列化和反序列化注意事项

路径

- 序列化的注意事项
- 反序列化的注意事项

讲解

序列化的注意事项

- 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。

反序列化的注意事项

- 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。
- 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 Person implements Serializable {
```

```
    static final long serialVersionUID = 4L;

    String name;
    transient int age;

    Animal anl;// 宠物

    String sex;

    public Person() {
    }

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

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", anl=" + anl +
                '}';
    }
}
public class Animal implements Serializable {
    String name;

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

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

public class Test1 {
    public static void main(String[] args)throws Exception {
        /*
            序列化的注意事项:
                1.要求序列化的对象所属的类实现Serializable接口，标记该类的对象可以被序列化
                2.要求序列化对象的所有属性值也是可以被序列化的
                3.如果对象的某个属性不想被序列化，那么就得使用transient关键字把该属性表明为
瞬态的
         */
        // 序列化
        // 创建Animal对象
        Animal anl = new Animal("旺财");
        // 创建Person对象
        Person p = new Person("张三", 18,anl);
        // 需求：将p对象写入到 day13\\ddd\\a.txt 文件中
```

```
        // 创建序列化流对象，关联目的地文件路径
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("day13\\ddd\\b.txt"));
        // 写出对象
        oos.writeObject(p);
        // 关闭流，释放资源
        oos.close();
    }
}

public class Test2 {
    public static void main(String[] args) throws Exception{
        /*
            反序列化注意事项：
                1.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。
                2.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
         */
        // 反序列化：
        // 1.创建反序列化流对象，关联数据源文件路径
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("day13\\ddd\\b.txt"));
        // 2.读取一个对象
        Object obj = ois.readObject();
        System.out.println(obj);
        // 3.关闭流，释放资源
        ois.close();
    }
}
```

小结

略

实操--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。

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

实现

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

```
    int age;

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

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

public class Test {
    public static void main(String[] args) throws Exception {
        /*
            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。
         */
        // 创建反序列化流对象,关联数据源文件路径
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("day13\\ddd\\list.txt"));
        // 重构集合对象
        Object obj = ois.readObject();
        // 关闭流,释放资源
        ois.close();

        // 遍历集合，打印对象信息。
        ArrayList<Student> list = (ArrayList<Student>)obj;
        for (Student stu : list) {
            System.out.println(stu);
        }

    }

    // 序列化:
    private static void method01() throws IOException {
        // 1.创建ArrayList集合,限制集合中元素的类型为Student类型
        ArrayList<Student> list = new ArrayList<>();

        // 2.创建多个Student对象
        Student stu1 = new Student("张三", 18);
        Student stu2 = new Student("李四", 12);
        Student stu3 = new Student("王五", 13);
        Student stu4 = new Student("赵六", 16);
        // 3.把Student对象添加到集合中
        list.add(stu1);
        list.add(stu2);
        list.add(stu3);
        list.add(stu4);

        // 4.创建序列化流对象,关联目的地文件路径
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("day13\\ddd\\list.txt"));
```

```
        // 5.序列化集合对象
        oos.writeObject(list);

        // 6.关闭流,释放资源
        oos.close();
    }
}
```

小结

略

第六章 打印流

目标

- 理解打印流的使用

路径

- 打印流的概述
- 打印流的使用

讲解

打印流的概述

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。

打印流的使用

- `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：

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

`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"，将数据输出到指定文本文件中。

```
public class Test {
    public static void main(String[] args) throws Exception{
        /*
            打印流PrintStream:
            - 打印流的概述
                概述: java.io.PrintStream类最终是继承了OutputStream,所以它其实本质也是一个字节输出流
                特点: 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。
            - 打印流的使用
                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。
```

```
                特有方法:
                    void print(任意类型的数据)  不换行打印
                    void println(任意类型的数据) 换行打印
         */
        // 创建打印流对象,关联目的地文件路径
        PrintStream ps = new PrintStream("day13\\eee\\a.txt");

        // 打印数据
        ps.print(100);
        ps.print(2.3);
        ps.print('b');
        ps.print(false);
        ps.println();// 换行
        ps.println(97);
        ps.println(3.14);
        ps.println('a');
        ps.println(true);
        ps.println("jack");

        // 关闭流,释放资源
        ps.close();

        System.out.println("=======================练习--玩一玩
======================");
        // 获取系统的打印流对象:
        System.out.println(100);// 打印到控制台 100
        // 需求: 把System.out.println()打印的目的地从控制台改成day13\\eee\\b.txt
        PrintStream ps2 = System.out;
        ps2.println(100);// 打印到控制台 100


        // 修改系统的打印流对象:
        // 创建打印流对象,关联目的地文件路径
        PrintStream ps3 = new PrintStream("day13\\eee\\b.txt");
        // 修改System.out的值
        System.setOut(ps3);
        System.out.println(100);// 打印b.txt文件中
    }
}
```

小结

略

第七章 装饰设计模式

目标

- 会使用装饰设计模式

路径

- 装饰模式概述
- 案例演示

讲解

装饰模式概述

在我们今天所学的缓冲流中涉及到java的一种设计模式，叫做装饰模式，我们来认识并学习一下这个设计模式。

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:

在不改变原类的基础上对LiuDeHua类的sing方法进行扩展

实现步骤:

1. 编写一个LiuDeHuaWarpper类, 实现Star接口,重写抽象方法
2. 提供LiuDeHuaWarpper类的有参构造, 传入LiuDeHua类对象
3. 在LiuDeHuaWarpper类中对需要增强的sing方法进行增强
4. 在LiuDeHuaWarpper类对不需要增强的方法调用LiuDeHua类中的同名方法

实现代码如下

LiuDeHua类: 被装饰类

LiuDeHuaWarpper类: 我们称之为装饰类

```
/*
    装饰模式遵循原则:
```

```
        装饰类和被装饰类必须实现相同的接口
        在装饰类中必须传入被装饰类的引用
        在装饰类中对需要扩展的方法进行扩展
        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
*/
public class LiuDeHuaWarpper implements Star {
    // 存放被装饰类的引用
    private LiuDeHua liuDeHua;
    // 通过构造器传入被装饰类对象
    public LiuDeHuaWarpper(LiuDeHua liuDeHua){
        this.liuDeHua = liuDeHua;
    }
    @Override
    public void sing() {
        // 对需要扩展的方法进行扩展增强
        System.out.println("刘德华在鸟巢的舞台上演唱忘情水.");
    }
    @Override
    public void dance() {
        // 不需要增强的方法调用被装饰类中的同名方法
        liuDeHua.dance();
    }
}
```

测试结果

```
public static void main(String[] args) {
    // 创建被装饰类对象
    LiuDeHua liuDeHua = new LiuDeHua();
    // 创建装饰类对象,被传入被装饰类
    LiuDeHuaWarpper liuDeHuaWarpper = new LiuDeHuaWarpper(liuDeHua);
    // 调用装饰类的相关方法,完成方法扩展
    liuDeHuaWarpper.sing();
    liuDeHuaWarpper.dance();
}
```

小结

装饰模式可以在不改变原类的基础上对类中的方法进行扩展增强,实现原则为:

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

第八章 commons-io工具包

目标

- 掌握commons-io工具包的使用

路径

- commons-io工具包的概述
- commons-io工具包的使用
- commons-io工具包常用api介绍

讲解

commons-io工具包的概述

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，见下表：

包	功能描述
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

commons-io工具包的使用

步骤：

1. 下载commons-io相关jar包；http://commons.apache.org/proper/commons-io/
2. 把commons-io-2.6.jar包复制到指定的Module的lib目录中
3. 将commons-io-2.6.jar加入到classpath中

commons-io工具包的使用

- commons-io提供了一个工具类 org.apache.commons.io.IOUtils，封装了大量IO读写操作的代码。其中有两个常用方法：

> 1. 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)
> 2. 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)

文件复制案例演示：

```
// IOUtils工具类拷贝文件
    private static void method01() throws IOException {
        // 拷贝一个文件
        // 创建输入流和输出流对象
        FileInputStream fis = new FileInputStream("day13\\aaa\\hb.jpg");
        FileOutputStream fos = new FileOutputStream("day13\\fff\\hb1.jpg");
        // 拷贝
        IOUtils.copy(fis,fos);
        // 关闭流,释放资源
        fos.close();
        fis.close();
    }
```

- commons-io还提供了一个工具类org.apache.commons.io.FileUtils，封装了一些对文件操作的方法：

> 1. public static void copyFileToDirectory(final File srcFile, final File destFile) //复制文件到另外一个目录下。
> 2. public static void copyDirectoryToDirectory(file1 , file2);//复制file1目录到file2位置。

案例演示:

```
public static void main(String[] args) throws IOException {
        // 拷贝文件夹
        /*File file1 = new File("day13\\aaa");
        File file2 = new File("day13\\fff");
        FileUtils.copyDirectoryToDirectory(file1,file2);*/

        // 拷贝文件
        File file3 = new File("day13\\aaa\\hbCopy1.jpg");
        File file4 = new File("day13\\fff");
        FileUtils.copyFileToDirectory(file3,file4);
    }
```

小结

略

总结

练习:

1. IO异常的处理(jdk7前,jdk7)
2. 使用Properties读取配置文件中的数据到程序中
3. 字符缓冲流的读一行数据和写换行
4. 转换文件编码
5. 序列化集合
6. 装饰者设计模式案例
7. commons-io2个例题

```
- 能够使用Properties的load方法加载文件中配置信息
    load(InputStream is)
    load(Reader r)
- 能够使用字节缓冲流读取数据到程序
    高效读写数据
    BufferedInputStream: read() 读一个字节    read(byte[] bys)读字节数组
- 能够使用字节缓冲流写出数据到文件
    BufferedOutputStream: write()写一个字节 write(byte[] b,int off,int len)写指定范围的字节数组
- 能够明确字符缓冲流的作用和基本用法
   字符缓冲流的作用: 高效读写
   基本用法:
      BufferedReader: read() 读一个字符,read(char[] chs)读字符数组
      BufferedWriter: write() 写一个字符 write(char[] chs,int off,int len)写指定范围的字符数组
                      write(String str) 写字符串
- 能够使用缓冲流的特殊功能
      字符缓冲输入流:  readLine()读一行
      字符缓冲输出流:  newLine()根据系统写一个行分隔符(换行)
- 能够阐述编码表的意义
      定义字符和二进制数的对应的规则
```

编码：字符--->二进制数
解码：二进制数--->字符

- 能够使用转换流读取指定编码的文本文件
  InputStreamReader(InputStream is,String charsetName)
- 能够使用转换流写入指定编码的文本文件
  OutputStreamWriter(OutputStream os,String charsetName)
- 能够使用序列化流写出对象到文件
  ObjectOutputStream: writeObject(Object obj)
- 能够使用反序列化流读取文件到程序中
  ObjectInputStream: readObject()
- 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共同的接口
  2.在装饰类中需要获取被装饰类的引用(成员变量)
  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,就使用被装饰类的引用调用被装饰类的原有方法
- 能够使用commons-io工具包
  1.拷贝commons-io工具包到模块下的lib文件夹中
  2.把commons-io工具包添加到classpath路径中
  3.使用commons-io工具包中的常用工具类: IOUtils,FileUtils

```
IOUtils:
 copy(InputStream in, OutputStream out); // 适合2gb以下
 copyLarge(InputStream in, OutputStream out); // 适合2gb以上
FileUtils:
    copyDirectoryToDirectory(File file1 ,File file2 );// 拷贝文件夹到另一个文件夹
    copyFileToDirectory(File file1 ,File file2 );// 拷贝文件到另一个文件夹
```