IO流：字符流等

回顾

今天内容

```
1.转换流
        1.1 InputStreamReader类的使用
        1.2 OutputStreamWriter类的使用
2.缓冲流
        2.1 BufferedInputStream类的使用
        2.2 BufferedOutputStream类的使用
        2.3 BufferedReader类的使用
        2.4 BufferedWriter类的使用
3.内存流
4.标准输入输出流
5.对象流
6.RandomAccessFile类
7.Properties类
8.装饰者设计模式
```

教学目标

```
1.掌握转换流的使用
2.掌握缓冲流的使用
3.了解内存流的使用
4.了解标准输入输出流的使用
5.掌握对象流的使用
6.了解RandomAccessFile类的使用
7.了解Properties的使用
8.了解装饰者设计模式
```

第一节 转换流

```
        作用：

        a.实现字节流到字符流的转换
        b.解决中文乱码的问题
                GBK
                Unicode
                utf-8
        c.只有转换流才能指定读取和写入的字符集
```

1.1 InputStreamReader类

```
InputStreamReader：字节字符转换输入流，将字节输入流转换为字符输入流
```

代码实现：

```
public class InputStreamReaderDemo {
    public static void main(String[] args) throws IOException {
        //1.实例化File的对象
        //File file = new File("file/input1.txt");

        //2.实例化转换输入流的对象
        //注意：当一个流的存在的意义是为了实例化另外一个流，则这个流不需要手动进行关闭
        //InputStream input = new FileInputStream(file);
        //InputStreamReader reader = new InputStreamReader(input);
        //使用默认的字符集【GBK】进行实例化转换流
        //InputStreamReader reader = new InputStreamReader(new FileInputStream(new
File("file/input1.txt")));

        //使用指定字符集进行实例化转换流
        //字符集一般使用字符串直接传参，不区分大小写,但是，如果字符集书写有误的话，则会跑出
java.io.UnsupportedEncodingException
        InputStreamReader reader = new InputStreamReader(new FileInputStream(new
File("file/input1.txt")),"UTF-8");

        //3.读取
        char[] arr = new char[16];
        int len = 0;

        while((len = reader.read(arr)) != -1) {
            String string = new String(arr, 0, len);
            System.out.println(string);
        }

        reader.close();
    }
}
```

1.2 OutputStreamWriter类

OutputStreamWriter：字节字符转换输出流，将字节输出流转换为字符输出流

代码实现：

```
public class OutputStreamWriterDemo {
    public static void main(String[] args) throws IOException {
        //需求：将一段文本以utf-8的格式写入到文件中【注，文件格式为默认格式】
        //1.实例化FIle对象
        //注意：对于所有的输出流而言，文件可以不存在，在进行写入的过程中可以自动进行创建
        //但是，对于所有的输入流而言，文件必须先存在，然后才能操作，否则，会抛出
FileNotFounedException
        File file = new File("file/output1.txt");

        //2.实例化转换输出流
        //如果不想覆盖源文件中的内容时，则在传参的时候，设置一个参数为true
        OutputStreamWriter writer = new OutputStreamWriter(new FileOutputStream(file,true),
"utf-8");

        //3.写入
        writer.write("家客户放假啊刚回家");

        //4.刷新
        writer.flush();

        //5.关闭
        writer.close();
    }
}
```

第二节 缓冲流

作用：主要是为了增强基础流的功能而存在的，提高了流的工作效率【读写效率】

2.1 BufferedInputStream类

```java
public class BufferedInputStreamDemo {
    public static void main(String[] args) throws IOException {
        //实例化一个File对象
        File file = new File("file/test22.txt");

        //实例化一个缓冲字节输入流的对象
        BufferedInputStream input = new BufferedInputStream(new FileInputStream(file));

        /*
        //读取
        byte[] arr = new byte[1024];
        int len = 0;
        while((len = input.read(arr)) != -1) {
            String string = new String(arr, 0, len);
        }
        */

        byte[] arr = new byte[4];
        int len = input.read(arr);
        String string = new String(arr, 0, len);
        System.out.println(string);

        input.mark(66);

        len = input.read(arr);
        string = new String(arr, 0, len);
        System.out.println(string);

        // 实现了效果：覆水可收
        input.reset();

        len = input.read(arr);
        string = new String(arr, 0, len);
        System.out.println(string);

        input.close();
    }
}
```


2.2 BufferedOutputStream类

```
public class BufferedOutputStreamDemo {
    public static void main(String[] args) throws IOException {
        //实例化FIle对象
        File file = new File("test33.txt");

        //实例化换种字节输出流
        BufferedOutputStream output = new BufferedOutputStream(new FileOutputStream(file));

        //写
        output.write("你好的halle".getBytes());

        //刷新
        output.flush();

        //关闭
        output.close();
    }
}
```

2.3 BufferedReader类

```java
public class BufferedReaderDemo {

    public static void main(String[] args) throws IOException {
        //实例化FIle对象
        File file = new File("test33.txt");

        //实例化缓冲字符流的对象
        BufferedReader reader = new BufferedReader(new FileReader(file));

        //方式一：read循环读取
        /*
        //读取
        char[] arr = new char[8];
        int len = 0;

        while((len = reader.read(arr)) != -1) {
            String string = new String(arr, 0, len);
        }
        */

        //方式二：readLine循环读取
        /*
        String result1 = reader.readLine();
        System.out.println(result1);

        String result2 = reader.readLine();
        System.out.println(result2);
        */
        String result = "";
        while((result = reader.readLine()) != null) {
            System.out.println("第一行：" + result);
        }

        reader.close();
    }
}
```


2.4 BufferedWriter类

```java
public class BufferedWriterDemo {
    public static void main(String[] args) throws IOException {
        // 实例化FIle对象
        File file = new File("test33.txt");

        //实例化缓冲字符输出流
        BufferedWriter writer = new BufferedWriter(new FileWriter(file,true));

        // 写
        writer.write("今天天气还可以");

        // 作用：主要就是为了换行
        writer.newLine();

        // 刷新
        writer.flush();

        //关闭
        writer.close();
    }
}
```

第三节 内存流

输入和输出都是从文件中来的，当然，也可将输出的位置设置在内存上，这就需要ByteArrayInputStream和ByteArrayOutputStream

ByteArrayInputStream:将内容写入到内存中，是Inputstream的子类
ByteArrayOutputStream：将内存中数据输出，是OutputStream的子类
此时的操作应该以内存为操作点

> 案例：完成一个字母大小写转换的程序

```
public class TextDemo02 {
    public static void main(String[] args) throws IOException {
        //定义一个字符串，全部由大写字母组成
        String string = "HELLOWORLD";

        //内存输入流
        //向内存中输出内容，注意：跟文件读取不一样，不设置文件路径
        ByteArrayInputStream bis  = new ByteArrayInputStream(string.getBytes());
        //内存输出流
        //准备从内存中读取内容，注意：跟文件读取不一样，不设置文件路径
        ByteArrayOutputStream bos = new ByteArrayOutputStream();

        int temp = 0;
        //read()方法每次只读取一个字符
        while((temp = bis.read()) != -1) {
            //将读取的数字转为字符
            char c = (char)temp;
            //将字符变为大写
            bos.write(Character.toLowerCase(c));
        }
        //循环结束之后，所有的数据都在ByteArrayOutputStream中
        //取出内容，将缓冲区内容转换为字符串
        String newString = bos.toString();

        //关闭流
        bis.close();
        bos.close();
        System.out.println(newString);
    }
}
```

实际上以上操作很好体现了对象的多态。通过实例化其子类不同，完成的功能也不同，也就相当于输出的位置不同，如果是输出文件，则使用FileXxxx类。如果是内存，则使用ByteArrayXxx。

总结：
a.内存操作流的操作对象，一定是以内存为主准，不要以程序为准。
b.实际上此时可以通过向上转型的关系，为OutputStream或InputStream.
c.内存输出流在日后的开发中也是经常使用到，所以一定要重点掌握

第四节 标准输入输出流

Java的标准输入/输出分别通过System.in和System.out实现，默认情况下分别代表是键盘和显示器

```java
public class PrintStreamDemo {
    public static void main(String[] args) throws FileNotFoundException {
        //System.out.println("hello world");

        //创建打印流的对象
        //注意：默认打印到控制台，但是，如果采用setOut方法进行重定向之后，将输出到指定的文件中
        PrintStream print = new PrintStream(new FileOutputStream(new File("test33.txt")));

        /*
         * static void setErr(PrintStream err)
            重新分配“标准”错误输出流。
        static void setIn(InputStream in)
            重新分配“标准”输入流。
        static void setOut(PrintStream out)
            重新分配“标准”输出流。
         * */
        //将标准输出重定向到print的输出流
        System.setOut(print);

        System.out.println("hello world");
    }
}
```

```java
public class InputStreamDemo {
    public static void main(String[] args) throws FileNotFoundException {
        FileInputStream inputStream = new FileInputStream(new File("test33.txt"));

        //setIn
        System.setIn(inputStream);

        //System.out.println("请输入内容:");

        //默认情况下是从控制台进行获取内容
        //但是如果使用setIn方法设置了重定向之后，将从指定文件中获取内容
        Scanner sc = new Scanner(System.in);

        String string = sc.next();

        System.out.println(string);
    }
}
```

第五节 对象流

流中流动的数据是对象

如果将一个对象写入到本地文件中，被称为对象的序列化

如果将一个本地文本中的对象读取出来，被称为对象发序列化

注意：

一个对象流只能操作一个对象，如果试图采用一个对象流操作多个对象的话，会出现EOFException【文件意外达到了文件末尾】

如果向将多个对象序列化到本地，可以借助于集合，【思路：将多个对象添加到集合中，将集合的对象写入到本地文件中，再次读出来，获取到的仍然是集合对象，遍历集合】

```java
public class ObjectStreamDemo {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        //objectOutputStreamUsage();
        objectInputStreamUsage();
    }

    // 写：将对象进行序列化
    public static void objectOutputStreamUsage() {
        //1.实例化一个Person的对象
        Person person =  new Person("张三", 10, 'B');

        //2.实例化一个对象输出流的对象
        ObjectOutputStream output = null;
        try {
            output = new ObjectOutputStream(new FileOutputStream(new File("file/person.txt")));

            //3.将对象写入到流中
            output.writeObject(person);

            //4.刷新
            output.flush();

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        finally {
            try {
                output.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    // 读：反序列化
    public static void objectInputStreamUsage() {
        //1.实例化对象输入流的对象
        try {
            ObjectInputStream input = new ObjectInputStream(new FileInputStream(new
File("file/person.txt")));

            //2.读取
            Object object = input.readObject();
```

```
            //3.对象的向下转型
            if(object instanceof Person) {
                Person p = (Person)object;
                System.out.println(p);
            }

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

注意：在使用对象流的时候，用于初始化对象流的参数只能是字节流（将对象转换为二进制的形式，然后再把二进制写入文件）

第六节 RandomAccessFile类

RandomAccessFile是用来访问那些保存数据记录的文件的，你就可以用seek( )方法来访问记录，并进行读写了。这些记录的大小不必相同；但是其大小和位置必须是可知的。但是该类仅限于操作文件

RandomAccessFile不属于InputStream和OutputStream类系的。实际上，除了实现DataInput和DataOutput接口之外(DataInputStream和DataOutputStream也实现了这两个接口)，它和这两个类系毫不相干，甚至不使用InputStream和OutputStream类中已经存在的任何功能；它是一个完全独立的类，所有方法(绝大多数都只属于它自己)都是从零开始写的。这可能是因为RandomAccessFile能在文件里面前后移动，所以它的行为与其它的I/O类有些根本性的不同。总而言之，它是一个直接继承Object的，独立的类

RandomAccessFile不属于InputStream和OutputStream类系的。实际上，除了实现DataInput和DataOutput接口之外(DataInputStream和DataOutputStream也实现了这两个接口)，它和这两个类系毫不相干，甚至不使用InputStream和OutputStream类中已经存在的任何功能；它是一个完全独立的类，所有方法(绝大多数都只属于它自己)都是从零开始写的。这可能是因为RandomAccessFile能在文件里面前后移动，所以它的行为与其它的I/O类有些根本性的不同。总而言之，它是一个直接继承Object的，独立的类

> 案例一：RandomAccessFile类的应用

```java
public class TextDemo01 {
    public static void main(String[] args) throws Exception {
        RandomAccessFile file = new RandomAccessFile("file.txt", "rw");
        // 以下向file文件中写数据
        file.writeInt(20);// 占4个字节
        file.writeDouble(8.236598);// 占8个字节
        //这个长度写在当前文件指针的前两个字节处，可用readShort()读取
        file.writeUTF("这是一个UTF字符串");
        file.writeBoolean(true);// 占1个字节
        file.writeShort(395);// 占2个字节
        file.writeLong(23254511);// 占8个字节
        file.writeUTF("又是一个UTF字符串");
        file.writeFloat(35.5f);// 占4个字节
        file.writeChar('a');// 占2个字节
        //把文件指针位置设置到文件起始处
        file.seek(0);

        // 以下从file文件中读数据，要注意文件指针的位置
        System.out.println("————从file文件指定位置读数据————");
        System.out.println(file.readInt());
        System.out.println(file.readDouble());
        System.out.println(file.readUTF());

        //将文件指针跳过3个字节，本例中即跳过了一个boolean值和short值。
        file.skipBytes(3);
        System.out.println(file.readLong());

        //跳过文件中“又是一个UTF字符串”所占字节
        //注意readShort()方法会移动文件指针，所以不用写2。
        file.skipBytes(file.readShort());
        System.out.println(file.readFloat());

        // 以下演示文件复制操作
        System.out.println("————文件复制（从file到fileCopy）————");
        file.seek(0);
        RandomAccessFile fileCopy = new RandomAccessFile("fileCopy.txt", "rw");
        int len = (int) file.length();// 取得文件长度（字节数）
        byte[] b = new byte[len];
        //全部读取
        file.readFully(b);
        fileCopy.write(b);
        System.out.println("复制完成！");
    }
}
```

> 案例二：RandomAccessFile 插入写示例

```java
public class TextDemo01 {
    public static void main(String[] args) {

        try {
            insert(3, "java", "file.txt");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    /**
     * @param skip 跳过多少过字节进行插入数据
     * @param str 要插入的字符串
     * @param fileName 文件路径
     * @throws Exception
     */
    public static void insert(long skip, String str, String fileName) throws Exception {

        RandomAccessFile raf = new RandomAccessFile(fileName, "rw");
        if (skip < 0 || skip > raf.length()) {
            System.out.println("跳过字节数无效");
            return;
        }
        byte[] b = str.getBytes();
        raf.setLength(raf.length() + b.length);
        for (long i = raf.length() - 1; i > b.length + skip - 1; i--) {
            raf.seek(i - b.length);
            byte temp = raf.readByte();
            raf.seek(i);
            raf.writeByte(temp);
        }
        raf.seek(skip);
        raf.write(b);
        raf.close();
    }
}
```


第七节 **Properties**类

是Map接口的一个实现类，并且是Hashtable的子类

Properties文件中元素也是以键值对的形式存在的

```
public class PropertiesDemo {
    public static void main(String[] args) {
        //1.实例化一个Properties的对象
        Properties pro = new Properties();
        System.out.println(pro);

        //2.把文件userlist.properties中的键值对同步到集合中
        //实质：读取
        /**
         *  void load(InputStream inStream)
            从输入流中读取属性列表（键和元素对）。
         */
        try {
            pro.load(new BufferedInputStream(new FileInputStream(new
File("file/userlist.properties"))));
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        System.out.println(pro);

        //3.向集合中添加一对键值对
        /*
         *  Object setProperty(String key, String value)
            调用 Hashtable 的方法 put。
         * */
        pro.setProperty("address", "china");

        System.out.println(pro);

        try {
            //4.store
            //实质：写入
            //comments:工作日志
            pro.store(new BufferedOutputStream(new FileOutputStream(new
File("file/userlist.properties"))), "add a pair of key and value");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

第八节 装饰者设计模式

要实现装饰者模式,注意一下几点内容:

a.装饰者类要实现真实类同样的接口
b.装饰者类内有一个真实对象的引用(可以通过装饰者类的构造器传入)
c.装饰类对象在主类中接受请求,将请求发送给真实的对象(相当于已经将引用传递到了装饰类的真实对象)
d.装饰者可以在传入真实对象后,增加一些附加功能(因为装饰对象和真实对象都有同样的方法,装饰对象可以添加一定操作在调用真实对象的方法,或者先调用真实对象的方法,再添加自己的方法)
e.不用继承

假设要制造添加甜蜜素和着色剂的馒头:

a.需要生产一个正常馒头
b.为节省成本(不使用玉米面),使用染色剂加入到正常馒头中
c.和面,最后生产出染色馒头

代码实现:

```
//做面包的接口
public interface IBread {
    void prepair();
    void kneadFlour();
    void steamed();
    void process();
}
```

```java
//制作正常馒头
public class NormalBread implements IBread {

    @Override
    public void prepair() {
        System.out.println("准备面粉,水以及发酵粉...");
    }

    @Override
    public void kneadFlour() {
        System.out.println("和面...");
    }

    @Override
    public void steamed() {
        System.out.println("蒸馒头...香喷喷的馒头出炉了");
    }

    @Override
    public void process() {
        prepair();
        kneadFlour();
        steamed();
    }
}
```

```java
//定义出制作面包的抽象类
public class AbstractBread implements IBread {
    private final IBread bread;
    public AbstractBread(IBread bread) {
        super();
        this.bread = bread;
    }
    @Override
    public void prepair() {
        this.bread.prepair();
    }
    @Override
    public void kneadFlour() {
        this.bread.kneadFlour();
    }
    @Override
    public void steamed() {
        this.bread.steamed();
    }
    @Override
    public void process() {
        prepair();
        kneadFlour();
        steamed();
    }
}
```

```java
//生产有着色剂的"玉米馒头"
public class CornDecorator extends AbstractBread {
    public CornDecorator(IBread bread) {
        super(bread);
    }

    public void paint() {
        System.out.println("添加柠檬黄的着色剂");
    }

    @Override
    public void kneadFlour() {
        //添加着色剂后和面
        this.paint();
        super.kneadFlour();
    }
}
```

```java
//生产有甜蜜素的"甜馒头"
public class SweetDecorator extends AbstractBread {

    public SweetDecorator(IBread bread) {
        super(bread);
    }

    public void paint() {
        System.out.println("添加甜蜜素...");
    }

    @Override
    public void kneadFlour() {
        // 添加甜蜜素后和面
        this.paint();
        super.kneadFlour();
    }
}
```

```
//测试类
public class TextDemo {
    public static void main(String[] args) {
        System.out.println("=======开始装饰馒头");
        IBread normalBread = new NormalBread();
        normalBread = new SweetDecorator(normalBread);
        normalBread = new CornDecorator(normalBread);
        normalBread.process();
        System.out.println("=======装饰馒头结束");
    }
}
```

总结

课前默写

1.使用转换流实现文件内容的拷贝

2.使用字符缓冲流实现文件内容的拷贝

作业

1.在电脑中盘下创建一个文件为HelloWorld.txt文件，判断他是文件还是目录，在创建一个目录FileTest,之后将HelloWorld.txt移动到FileText目录下去

3.在程序中写一个"HelloJavaWorld你好世界"输出到操作系统文件Hello.txt文件中
4.从磁盘读取一个文件到内存中，再打印到控制台
5.统计一个含有英文单词的文本文件的单词个数
6.从磁盘读取一个文件到内存中，再打印到控制台
7.实现将一个文件夹中的一张图片拷贝到另外一个文件夹下
8.将上课关于不同的流实现拷贝的案例敲熟练
9.定义一个类Student，定义属性：学号、姓名和成绩，方法：show（显示个人信息）

实例化几个Student对象，将这几个对象保存到student.txt文件中
然后再将文件中的内容读取出来并且调用show方法

10.从控制台输入一个字符串，统计其中一个字符出现的次数
11.理解装饰者设计模式，并实现类似课堂案例的案例

面试题

1.BufferedReader属于哪种流,它主要是用来做什么的,它里面有那些经典的方法
2.怎么样把输出字节流转换成输出字符流,说出它的步骤
3.流一般需要不需要关闭,如果关闭的话在用什么方法,一般要在那个代码块里面关闭比较好，处理流是怎么关闭的，如果有多个流互相调用传入是怎么关闭的？
4.什么叫对象序列化，什么是反序列化，实现对象序列化需要做哪些工作
5.在实现序列化接口是时候一般要生成一个serialVersionUID字段,它叫做什么,一般有什么用
6.什么是内存流？有什么作用