IO

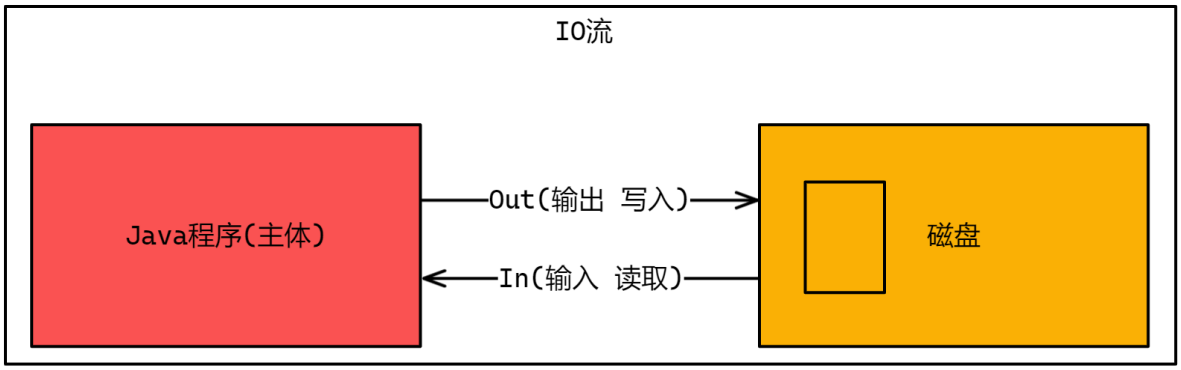
1 IO简介

- In/Out: 相对于程序而言的输入(读取)和输出(写出)的过程。

  即: 通过java程序到磁盘读取数据的过程, 我们称为In的过程, 也就是读取(输入)
- 将java程序中的数据写入磁盘的过程, 我们称为Out的过程, 也就是写出(输出)
- JDK核心类库中提供了IO流相关的类, 这些类都在<java.io>包下

2 流的概念

- 程序中数据的读取和写入, 可以想象成水流在管道中流动。
  - 流只能单方向流动
  - 输入流用来读取in
  - 输出流用来写出Out
  - 数据只能从头到尾顺序的读取一次或写出一次



3 节点流和处理流

- 按照流是否直接与特定的地方(如磁盘,内存,设备等)相连,分为节点流和处理流两类

3.1 节点流

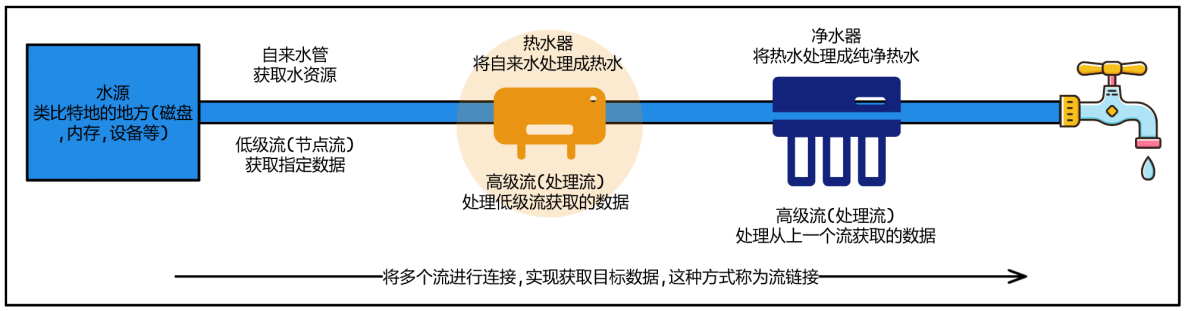
- 可以从或向一个特定的地方(节点)读写数据

3.2 处理流

- 是对一个**已存在的流**的连接和封装,通过所封装的流的功能调用实现数据读写

3.3 处理流特点

- 处理流的构造方法总是要带一个其他的流对象做参数,一个流对象经过其他流的多次包装,成为流的链接.
- 通常节点流也被称之为低级流.处理流也被称之为高级流或者过滤流



4 节点流

4.1 OutputStream

- 此抽象类是表示 **输出字节流** 的所有类的超类。输出流接受输出字节并将这些字节发送到某个接收器。

4.2 FileOutputStream

- 直接插在文件上，直接写出(输出)文件数据

创建对象：

```
FileOutputStream(String name) //创建一个向具有指定名称的文件中写入数据的输出文件流。
FileOutputStream(File file) //创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
FileOutputStream(File file, boolean append) //追加 创建一个向指定 File 对象表示的文件中写入数据的文件输出流。
```

注意: 以上构造方法中, 如果参数指向的文件以及父目录都不存在, 将会抛出FileNotFoundException异常!如果参数指向的文件不存在, 但文件的所在目录存在, 将不会抛异常, 程序会自动创建该文件!

- **代码案例**

```
package cn.tedu.io;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * 此案例学习FileOutputStream文件字节输出流向指定文件写出字节数据
 */
public class FOSDemo {
    public static void main(String[] args) throws IOException {
        //一、创建流对象
        //①第一种方式
        // 创建File实例,绑定目标文件
        //File file = new File("./demo/fos.txt");
        // 创建文件字节输出流,并将file传入到该实例中
        //FileOutputStream fos = new FileOutputStream(file);
        //②第二种方式
        // 创建文件字节流的同时,将目标文件路径传入该实例中
        //fos = new FileOutputStream("./demo/fos.txt");
        //③第三种方式
        // 创建文件字节流时,可以开启追加模式(只需要添加第二个参数为true,即可开启追加模式)
        FileOutputStream fos = new FileOutputStream("./demo/fos.txt", true);
        //二、通过字节输出流向指定的文件中输出字节数据
```

```
        //①输出单字节数据
        /*
         * void write(int b)
         * 参数类型是整数,表示一个字节
         * 该方法,每次调用,都会向绑定的文件中,写出一个字节数据
         */
        fos.write(97);//a
        fos.write(98);//b
        fos.write(99);//c
        fos.write(100);//d
        fos.write(13);//回车符(将光标回归到行首)
        fos.write(10);//换行符(将光标移到下一行)
        //②一次性输出多个字节数据(依赖字节数组)
        /*
         * byte[] getBytes()
         * 将字符串转换为字节数据,存储到字节数组中
         * \r\n 表示回车符和换行符
         */
        fos.write("Hello JSDVN2302!\r\n".getBytes());
        //③输出字节数组中的一部分字节数据
        //ABCDEFGH\r\n → FGH\r\n
        /*
         * void write(byte b[], int off, int len)
         * off: 表示要输出的头字节的下标,下标5的位置就是F
         * len: 表示从要输出的头字节之后的几个字节数据
         */
        fos.write("ABCDEFGH\r\n".getBytes(), 5, 5);
        System.out.println("写出完毕!!");
        //三、关闭流资源,否则会占用资源
        fos.close();
    }
}
```

4.3 InputStream

- 此抽象类是表示字节输入流的所有类的超类/抽象类。

4.4 FileInputStream

- 直接插在文件上，直接读取文件数据。
- **创建对象**

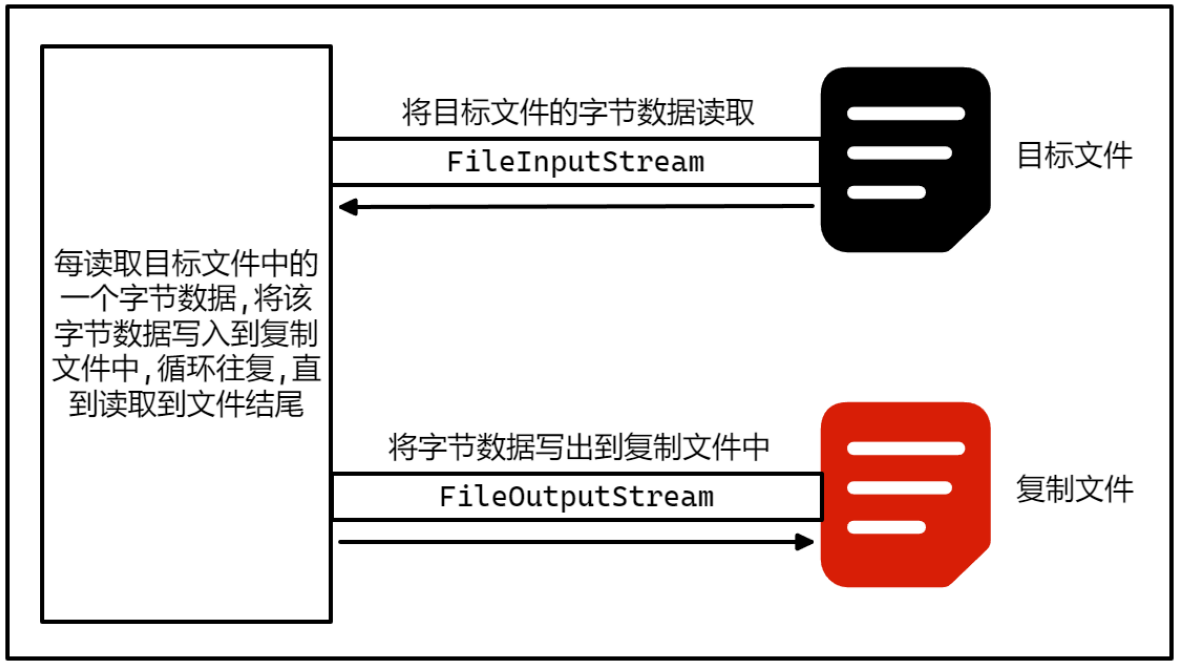
```
FileInputStream(File file)
通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的 File 对象 file 指定。
FileInputStream(String pathname)
通过打开一个到实际文件的连接来创建一个 FileInputStream，该文件通过文件系统中的路径名 name 指定。
```

- **代码案例**

```
package cn.tedu.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

/**
 * 此案例学习FileInputStream文件字节输入流向指定文件读取字节数据
 */
public class FISDemo {
```

```
    public static void main(String[] args) throws IOException {
        //创建文件字节输入流并绑定文件
        FileInputStream fis = new FileInputStream("./demo/fos.txt");
        //读取一个字节
        /*
         * int read()
         * 读取1个字节,并将该字节转换为整数返回,如果读取到了文件的末尾,则返回-1
         * 连续调用read方法时,会连续读取指定文件中的字节数据
         */
//        int d = fis.read();
//        System.out.println((char) d);
//        System.out.println((char) fis.read());
//        System.out.println((char) fis.read());
//        System.out.println((char) fis.read());
//        System.out.println(fis.read());
//        System.out.println(fis.read());
        //使用循环,连续读取文件中的字节数据
        int data;//封装本次读取的结果
        while ((data = fis.read()) != -1) {//说明不是结尾
            System.out.print((char) data);
        }
        //关闭流
        fis.close();
    }
}
```

4.5 复制文件



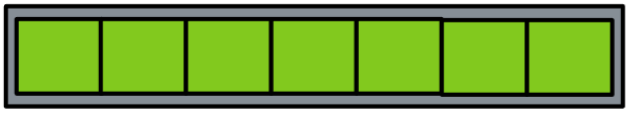

```
package cn.tedu.io;

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

/**
 * 通过此案例学习单字节复制文件
 */
public class CopyDemo01 {
    public static void main(String[] args) throws IOException {
```

```
        //创建文件字节输入流,用于读取目标文件字节数据
        FileInputStream fis = new FileInputStream("./demo/ZhenDe.jpg");
        //创建文件字节输出流,用于写出数据到复制文件
        FileOutputStream fos = new
FileOutputStream("./demo/ZhenDe_01.jpg");
        int data;
        long start = System.currentTimeMillis();
        //循环读取目标文件中的字节数据
        while ((data = fis.read()) != -1) {
            //将本次读取的字节数据,写出到复制文件中
            fos.write(data);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制该图片共耗时:" + (end - start) + "毫秒!");
        //释放资源
        fis.close();
        fos.close();
    }
}
```

4.6 块读写案例

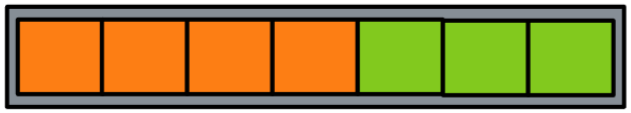
在java.io.InputStream 中定义了读取字节的方法：
1)int read() 读取一个字节数据,将数据转换为整数返回,读取到末尾返回-1
2)int read(byte[] data) 一次性读取给定的字节数组长度字节量,
并将读取的字节数据存储到字节数组中,将本次读取的字节量返回,读取到末尾返回-1

○假定读取的文件共有7个字节,然后4个字节一读



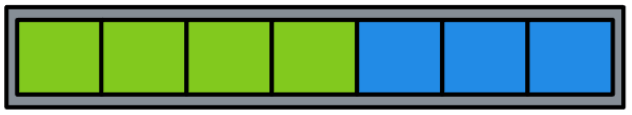
○定义长度为4的字节数组data

①第一次调用read方法,读取4个字节数据
int len = fis.read(data);



将读取的4个字节数据,存储到data数组中

并且返回整数4,表示本次实际读取了4个字节量
②第二次调用read方法,读取4个字节数据

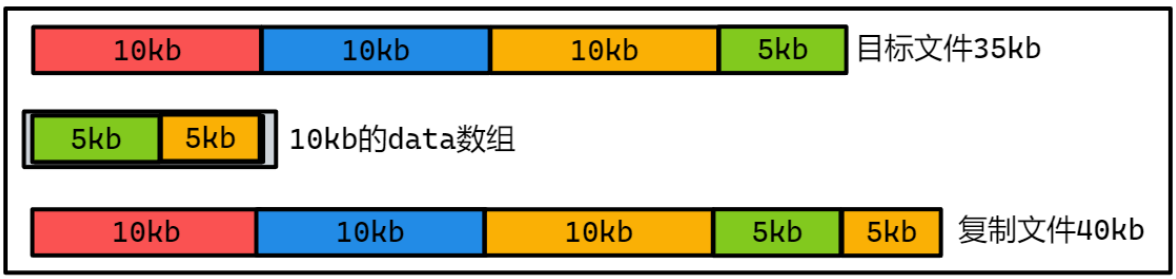


将读取的3个字节数据,存储到data数组中,此时,
data数组中存储本次读取的3个数据,
并且包含1个上次读取的数据
并且返回整数3,表示本次实际读取了3个字节数据
③第三次调用read方法,读取4个字节数据
文件已经读取完毕,此时什么数据也读取不到,
则直接返回-1,表示读取结束

```java
package cn.tedu.io;

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

/**
 * 通过此案例学习多字节复制文件
 * 单字节复制文件,由于需要频繁的内存和硬盘进行交互,导致整体效率极低,
 * 所以可以提高每次交互时读写的字节数量,减少实际交互的次数,进而提高读写效率,
 * 而一组字节一组字节的读写称为: 块读写形式
```

```
 */
public class CopyDemo02 {
    public static void main(String[] args) throws IOException {
        //创建文件字节输入流,用于读取目标文件字节数据
        FileInputStream fis = new FileInputStream("./demo/ZhenDe.jpg");
        //创建文件字节输出流,用于写出数据到复制文件
        FileOutputStream fos = new
FileOutputStream("./demo/ZhenDe_02.jpg");
        long start = System.currentTimeMillis();
        //使用块读的方式复制图片
        //1kb=1024byte
        byte[] data = new byte[10 * 1024];//10kb的字节数组
        //每次读取目标文件10kb的字节数据
        while (fis.read(data) != -1) {
            //每次读取的字节数据,都会存储到data中,所以直接将data写出即可
            fos.write(data);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制该图片共耗时:" + (end - start) + "毫秒!");
        //释放资源
        fis.close();
        fos.close();
    }
}
```

块读复制文件偏大的问题解决




```
package cn.tedu.io;

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

/**
 * 通过此案例学习多字节复制文件时,文件大小保持一致
 * 由于块读时,最后一次如果不满足data数组的长度,则会导致最后一次写入数据时,
 * 会将多余的上次读取的残余数据也一同写入,导致复制的文件偏大
 */
public class CopyDemo03 {
    public static void main(String[] args) throws IOException {
        //创建文件字节输入流,用于读取目标文件字节数据
        FileInputStream fis = new FileInputStream("./demo/ZhenDe.jpg");
        FileOutputStream fos = new
FileOutputStream("./demo/ZhenDe_03.jpg");
        long start = System.currentTimeMillis();
        byte[] data = new byte[10 * 1024];
        int len;//定义len记录本次读取的字节量
        while ((len = fis.read(data)) != -1) {
            //根据len,保证本次写入的字节的个数为实际读取的个数
            fos.write(data, 0, len);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制该图片共耗时:" + (end - start) + "毫秒!");
        fis.close();
```

```
        fos.close();
    }
}
```

4.7 写入字符串

```
package cn.tedu.io;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * 通过此案例向文件中写入字符串
 */
public class WriteStringDemo {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("./demo/string.txt");
        String line = "鹅鹅鹅,曲颈向刀割,拔毛烧开水,铁锅炖大鹅!";
        /*
         * 在使用IO写入中文时,最好指定字符集,防止乱码
         * ASCII: 最早的编码表之一,它包含了128个字符,包括英文,数字,标点符号和一些特殊字符,不包含中文
         * Unicode: 支持超过130000个字符,包括世界各地的语言和符号
         * UTF-8: 是Unicode的传输格式
         */
        fos.write(line.getBytes(StandardCharsets.UTF_8));
        fos.close();
    }
}
```

4.8 简易笔记本

```
package cn.tedu.io;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

/**
 * 此案例实现一个简易的记事本
 */
public class TestNotes01 {
    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);
        System.out.println("请开始输入内容,单独输入exit退出!");
        FileOutputStream fos = new FileOutputStream("./demo/note.txt");
        while (true) {
            //获取在控制台输入的字符串
            String line = scanner.nextLine();
            //判断控制台输入的字符串是否是exit
            if ("exit".equals(line)) {
                //跳出当前循环,结束程序
                break;
            }
            //将控制台输入的内容写入到文件中
            fos.write(line.getBytes(StandardCharsets.UTF_8));
        }
```

```
        fos.close();
    }
}
```

4.9 读取字符串

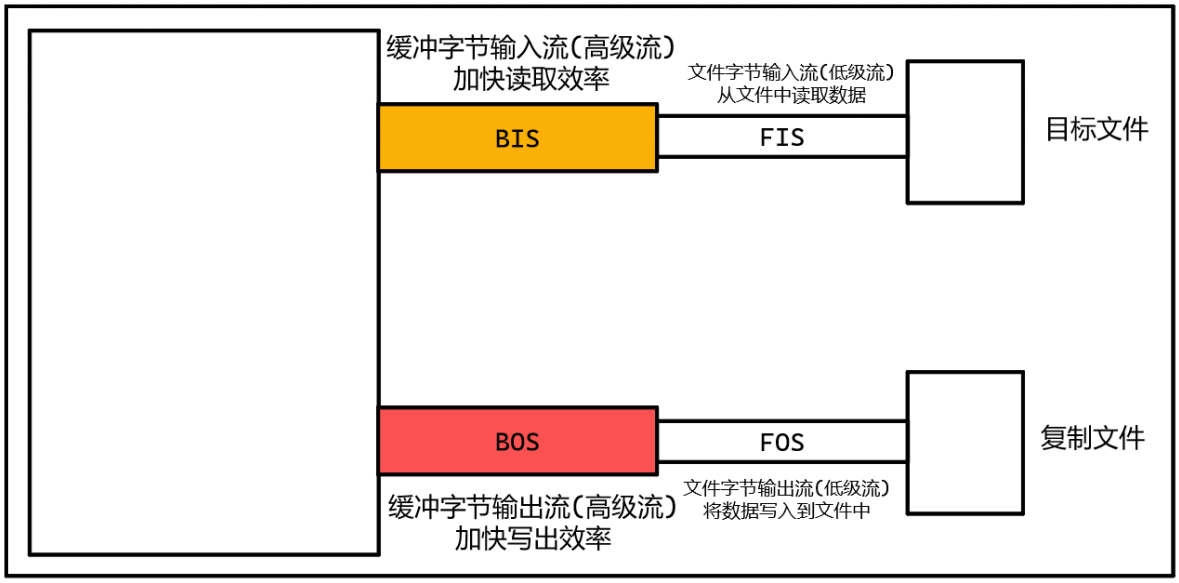
```
package cn.tedu.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;

/**
 * 此案例来学习从文件中读取字符串
 */
public class ReadStringDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("./demo/string.txt");
        /*
         * int available()
         * 预估本次使用输入流读取该文件的字节量,可以利用它表示文件长度
         */
        byte[] data = new byte[fis.available()];
        //一次性读取data大小的内容(文件中所有的数据)
        fis.read(data);
        //利用String的构造方法将data数组按照指定的编码表还原为字符串
        String str = new String(data, StandardCharsets.UTF_8);
        System.out.println(str);
        fis.close();
    }
}
```

5 处理流

5.1 缓冲流

- BufferedOutputStream缓冲输出流
- BufferedInputStream 缓冲输入流

5.1.1 复制文件




```java
package cn.tedu.io;

import java.io.*;

/**
 * 此案例使用缓冲流高效复制文件
 */
public class CopyDemo04 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("./demo/ZhenDe.jpg");
        //创建高级流,缓冲字节输入流,绑定低级流
        BufferedInputStream bis = new BufferedInputStream(fis);
        FileOutputStream fos = new
FileOutputStream("./demo/ZhenDe_04.jpg");
        //创建高级流,缓冲字节输出流,绑定低级流
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        long start = System.currentTimeMillis();
        int data;
        //循环通过高级流单字节读取数据
        while ((data = bis.read()) != -1) {
            //通过高级流写出该字节数据
            bos.write(data);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制该图片共耗时:" + (end - start) + "毫秒!");
        //关闭资源(关闭高级流,会将所连的低级流也一同关闭)
        bis.close();
        bos.close();
    }
}
```

5.1.2 flush

```java
package cn.tedu.io;

import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
```

```
 * 此案例学习缓冲流写出数据的缓冲区问题
 */
public class BOS_flushDemo {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("./demo/flush.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        String line = "唧唧复唧唧,木兰开飞机,开的什么机,波音747!";
        /*
         * 缓冲输出流,内部声明了一个8k的缓冲区,
         * 该缓冲区的特点,是如果读取不满8k的数据,不会将数据写出
         */
        bos.write(line.getBytes(StandardCharsets.UTF_8));
        System.out.println("写出完毕!");
        //故意不写close
        /*
         * 会将缓冲区的数据强制写出
         * close方法的内部会调用flush方法
         */
        bos.flush();
        //bos.close();
    }
}
```

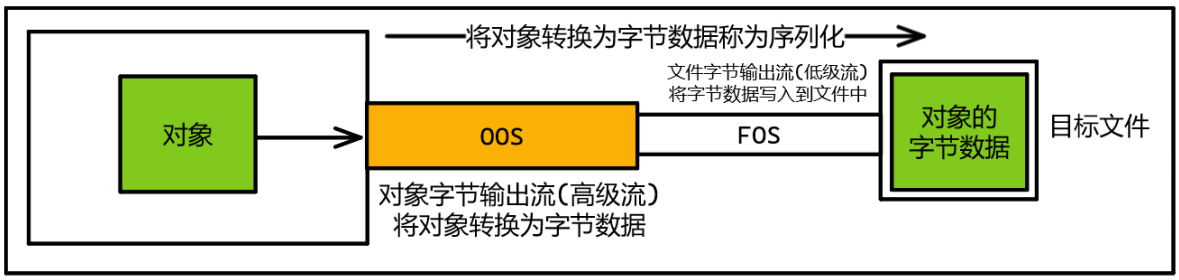
5.2 对象流

5.2.1 Person代码

```
package cn.tedu.io;

import java.io.Serializable;
import java.util.Arrays;

/**
 * 使用该类的实例,测试对象流的内容
 */
public class Person implements Serializable {
    //固定当前类的版本号为42
    static final long serialVersionUID = 42L;
    private String name;
    private int age;
    private String gender;
    /*
     * transient可以将修饰的属性在进行序列化时,忽略该属性的值,
     * 当我们反序列化时,改属性的值将不会被读取,以此达到一个对象瘦身的目的,
     * 从而提高程序的响应速度,减少资源开销
     */
    private transient String[] otherInfo;
    private double salary;
    //生成全参构造
    public Person(String name, int age, String gender, String[] otherInfo)
{
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.otherInfo = otherInfo;
    }

    //生成get和set方法
    public String getName() {
        return name;
    }
```

```
34
35      public void setName(String name) {
36          this.name = name;
37      }
38
39      public int getAge() {
40          return age;
41      }
42
43      public void setAge(int age) {
44          this.age = age;
45      }
46
47      public String getGender() {
48          return gender;
49      }
50
51      public void setGender(String gender) {
52          this.gender = gender;
53      }
54
55      public String[] getOtherInfo() {
56          return otherInfo;
57      }
58
59      public void setOtherInfo(String[] otherInfo) {
60          this.otherInfo = otherInfo;
61      }
62
63      //生成toString方法
64      @Override
65      public String toString() {
66          return "Person{" +
67                  "name='" + name + '\'' +
68                  ", age=" + age +
69                  ", gender='" + gender + '\'' +
70                  ", otherInfo=" + Arrays.toString(otherInfo) +
71                  '}';
72      }
73  }
```

5.2.2 OOSDemo案例



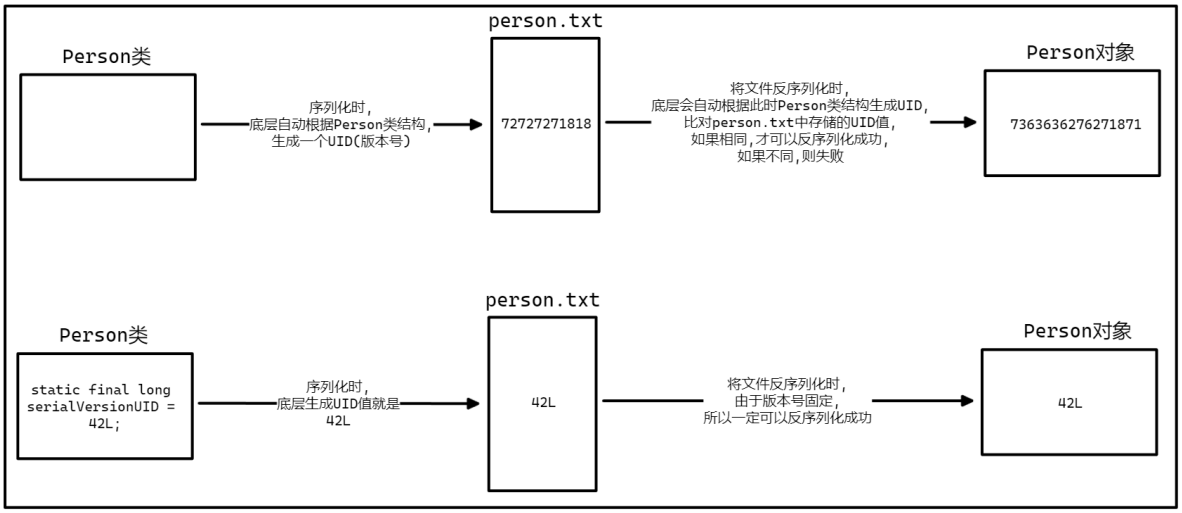

```
1  package cn.tedu.io;
2
3  import java.io.FileNotFoundException;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.ObjectOutputStream;
7
8  /**
9   * 此案例学习对象的序列化流
```

```
 */
public class OOSDemo {
    public static void main(String[] args) throws IOException {
        String name = "薛宏举";
        int age = 18;
        String gender = "男";
        String[] otherInfo = {"是一个帅锅", "来自于天津", "爱好学习", "是广大男性之友"};
        Person person = new Person(name, age, gender, otherInfo);
        System.out.println(person);
        //创建文件字节输出流,绑定要输出的文件
        FileOutputStream fos = new FileOutputStream("./demo/person.txt");
        //创建对象字节输出流,绑定文件字节输出流,将对象转换为字节数据,再将字节数据写入到文件中
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        //将person对象交给对象字节输出流
        //此处注意,写出的对象类必须要实现Serializable接口,否则不能序列化
        oos.writeObject(person);
        System.out.println("写出完毕!");
        //释放资源
        oos.close();
    }
}
```

5.2.3 OISDemo案例

```
package cn.tedu.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

/**
 * 通过此案例学习使用对象输入流将指定文件中的对象还原为对象
 */
public class OISDemo {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        FileInputStream fis = new FileInputStream("./demo/person.txt");
        //创建对象输入流,绑定指定的文件字节输入流,用于将该文件中读取的字节还原为对象
        ObjectInputStream ois = new ObjectInputStream(fis);
        /*
         * Object readObject()
         * 将文件中的字节数据还原为对象,该对象由于程序不知道是什么类型,所以返回的是Object
         *
         */
        Object p = ois.readObject();
        System.out.println(p);
        ois.close();
    }
}
```

5.2.4 版本号冲突



6字节流和字符流

- 在Java中，根据处理的数据单位不同，分为字节流和字符流。
  - 字节流: 一个字节(byte)一个字节的去读取, 或者写出
  - 字符流: 一个字符一个字符的去读取, 或者写出

6.1 字节流

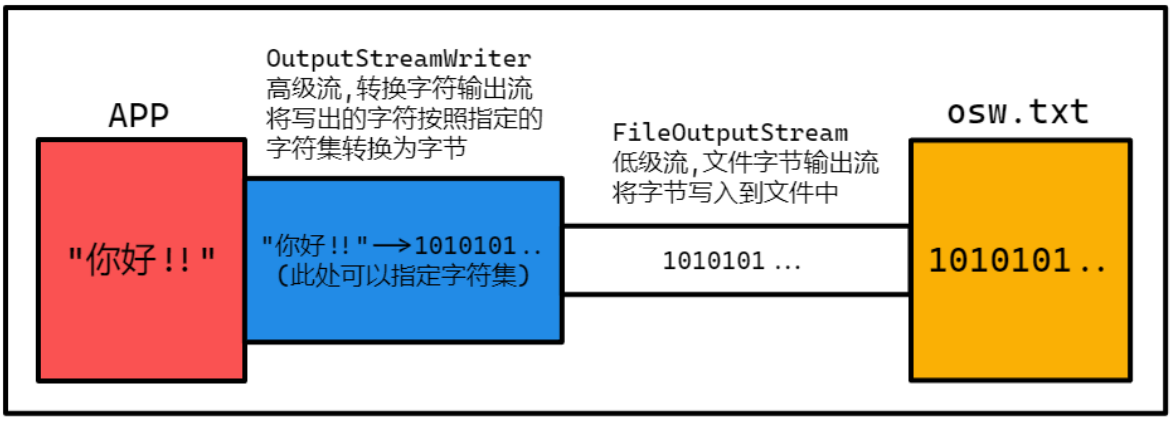
- **字节流(stream)：**针对二进制文件(文本,图片,音频,视频...等)
- InputStream(包含input都是输入流)
  - FileInputStream
  - BufferedInputStream
  - ObjectInputStream
- OutputStream(包含output都是输出流)
  - FileOutputStream
  - BufferedOutputStream
  - ObjectOutputStream

6.2 字符流

- **字符流(Reader,Writer)：**针对文本文件，读写容易发生乱码现象，在读写时最好指定编码集为utf-8
- Reader(Reader结尾的都是字符输入流)
  - BufferedReader
  - InputStreamReader
- Writer(Writer结尾的都是字符输出流)
  - BufferedWriter
  - OutputStreamWriter
  - PrintWriter

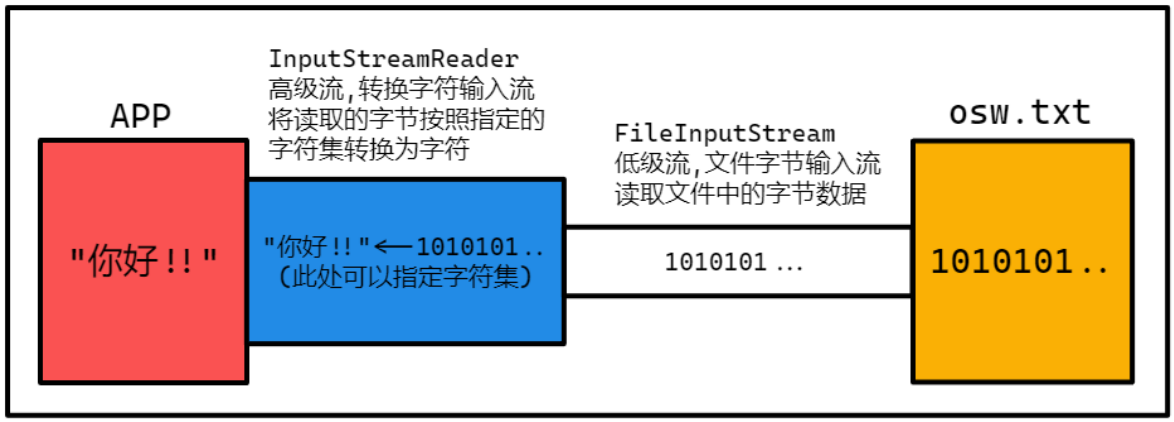
7 转换流

7.1 OutputStreamWriter




```java
package cn.tedu.io;

import java.io.*;
import java.nio.charset.StandardCharsets;

/**
 * 学习字符流时,必须要掌握转换流的内容,原因:
 * 字节流和字符流不能直接相连,需要转换流做协调,
 * 并且转换流具备以下功能:
 * ①在流链接中,衔接其他的高级字符流和下面的字节流
 * ②负责将字符与对应的字节按照指定的字符集进行自动转换方便读写
 */
public class OSWDemo {
    public static void main(String[] args) throws Exception {
        //低级的文件字节输出流
        FileOutputStream fos = new FileOutputStream("./demo/osw.txt");
        //高级的输出字符转换流,指定编码
        OutputStreamWriter osw = new OutputStreamWriter(fos,
StandardCharsets.UTF_8);
        //利用输出字符流,自动将写出的字符串按照编码写出
        osw.write("鹅鹅鹅");
        osw.write("曲颈向刀割");
        osw.write("拔毛烧开水");
        osw.write("铁锅炖大鹅");
        System.out.println("写出完毕!");
        osw.close();
    }
}
```

7.2 InputStreamReader

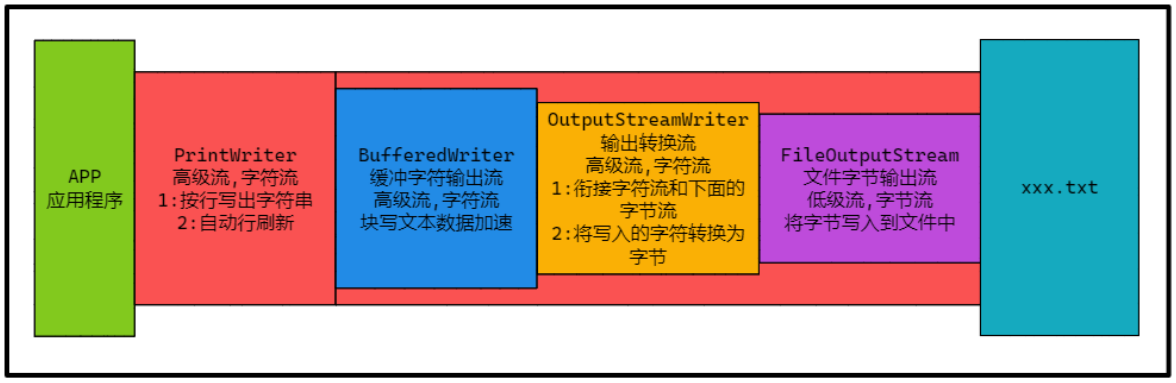



```java
package cn.tedu.io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;

/**
 * 使用输入字符转换流读取文本内容
 */
public class ISRDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("./demo/osw.txt");
        InputStreamReader isr = new InputStreamReader(fis,
StandardCharsets.UTF_8);
        int d;
        //使用字符流读取内容时,是一个字符一个字符读取,比如一次性可以将"鹅"这个字读进来
        //而使用字节流读取内容,是一个字节一个字节读取,在UTF-8中,一个中文由三个字节组成,
所以读取三次才可以将中文"鹅读取出来
        while ((d = isr.read()) != -1) {
            System.out.print((char) d);
        }
        isr.close();
    }
}
```

8 缓冲字符流

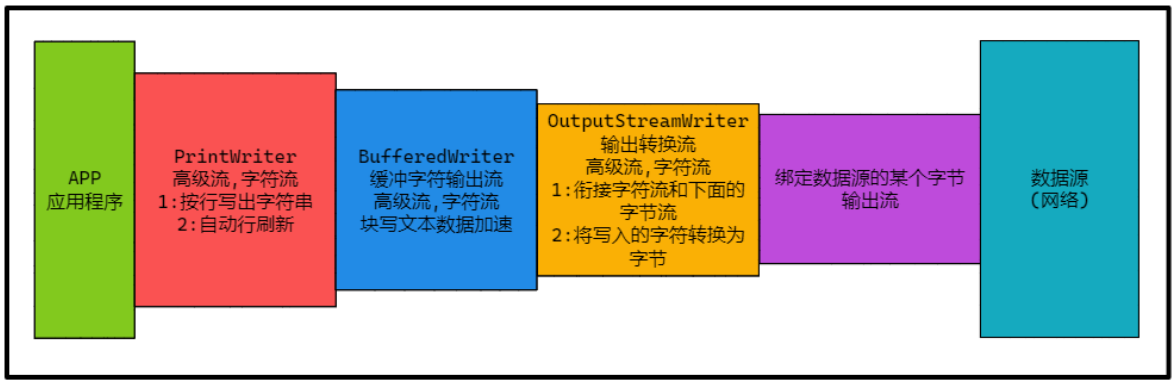
8.1 PrintWriter

- **代码案例**: 连接文件时



```java
package cn.tedu.io;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

/**
 * 缓冲字符流
 * PrintWriter是实际开发中使用的缓冲字符输出流
 * 功能:
 * ①可以提高写出字符的效率
 * ②可以按行写出字符串
 * ③可以自动行刷新
 */
public class PWDemo {
    public static void main(String[] args) throws FileNotFoundException {
        PrintWriter pw = new PrintWriter("./demo/pw.txt");
        //按行写出字符串
        pw.println("无竹令人俗,");
        pw.println("无肉使人瘦.");
        pw.println("不俗又不瘦,");
        pw.println("竹笋焖猪肉.");
        pw.println("出自--<苏轼的竹笋焖猪肉>");
        pw.close();
    }
}
```

- **代码案例**: 模拟连接的不是文件时




```java
package cn.tedu.io;

import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

/**
 * 利用PrintWriter实现建议的记事本,并且可以按行写入
 */
public class TestNote02 {
    public static void main(String[] args) throws FileNotFoundException {
        //创建低级的文件字节输出流 ①绑定文件 ②开启追加模式
        FileOutputStream fos = new FileOutputStream("./demo/note_pw.txt",
true);
        //创建高级的转换输出字符流 ①衔接字节流和字符流 ②将写出的字符自动按照编码集转换为
字节数据
        OutputStreamWriter osw = new OutputStreamWriter(fos,
StandardCharsets.UTF_8);
        //创建高级的缓冲字符输出流 ③提高块写文本数据的效率
        BufferedWriter bw = new BufferedWriter(osw);
        //创建高级的按行刷新字符流 ①按行插入字符串 ②开启自动行刷新(写一行字符串,回车之
后,会自动调用flush方法)
```

```
        PrintWriter pw = new PrintWriter(bw, true);
        //完成简易记事本的录入
        Scanner scanner = new Scanner(System.in);
        while (true) {
            String line = scanner.nextLine();
            if ("exit".equals(line)) {
                break;
            }
            //将控制台录入的字符串写入到文件中
            pw.println(line);
        }
        //释放流资源
        pw.close();
    }
}
```

9 总结

	输入流		输出流	
	字节流 **InputStream**	字符流 **Reader**	字节流 **OutputStream**	字符流 **Writer**
低级流 节点流	文件字节输入流 **File**InputStream 连接程序和文件的"管道", 负责从文件中读取字节		文件字节输出流 **File**OutputStream 连接程序和文件的"管道", 负责将字节写入到文件中	
高级流 处理流	缓冲字节输入流 **Buffered**InputStream 块读字节数据加速	转换输入流 **InputStreamReader** 1:衔接字节流和字符流 2:将读取的字节转换为字符	缓冲字节输出流 **Buffered**OutputStream 块写字节数据加速	转换输出流 **OutputStreamWriter** 1:衔接字节流和字符流 2:将写出的字符转换为字节
	对象字节输入流(反序列化流) **Object**InputStream 将对象2进制数据,还原为对象, 将对象反序列化	缓冲字符输入流 **Buffered**Reader 1:块读文本数据加速 2:按行读取字符串	对象字节输出流(序列化流) **Object**OutputStream 将对象对象转换为2进制数据, 将对象序列化	缓冲字符输出流 **Print**Writer 1:块写文本数据加速 2:按行写出字符串 3:自动的行刷新功能