FOSDemo

```
package io;

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

/*
JAVA IO：
IO：Input、Output，即：输入、输出
JAVA IO使程序可以和外界交换数据，用于对外界数据进行读写操作。

java将输入与输出比喻为“流”：stream
如何理解流：将流想象为一个连接程序和另一端的“管道”，在其中按照同一方向顺序移动的数据。
有点像“水管”中向同一方向流动的水。

输入流：从外界向我们的程序中移动的方向，是用来获取数据的流，作用是：读取操作
输出流：写出操作
注意：流是单向的，输入流永远只用来读，输出流永远只用来写。
将来我们在实际开发中希望与程序交互的另一端互相发送数据时，
只需要创建一个可以连接另一端的“流”，进行读写操作即可。

java定义了两个超类，来规范所有的字节流：
java.io.InputStream：抽象类，所有字节输入流的超类，里面定义了读取字节的相关方法，所有字节输入流都继承自它。
java.io.OutputStream：抽象类，所有字节输出流的超类，里面定义了写出字节的相关方法，所有字节输出流都继承自它。
```

文件流：
java.io.FileInputStream和java.io.FileOutputStream
是真实连接我们程序和文件之间的“管道”。
其中文件输入流用于从文件中读取字节，而文件输出流则用于向文件中写入字节。
文件流是节点流。

JAVA IO将流划分为两类：节点流和处理流
节点流：俗称“低级流”，特点：真实连接我们程序和另一端的“管道”，负责实际读写数据的流。
文件流就是典型的节点流，真实连接我们程序与文件的“管道”，可以读写文件数据。
处理流：俗称"高级流"
特点：
1：不能独立存在（不可以单独实例化进行读写操作）
2：必须连接在其他流上，目的是当数据"流经"当前流时，可以对其做某种加工操作，简化我们的工作。
流的连接：实际开发中经常会串联一组高级流最终到某个低级流上，对数据进行流水线式的加工读写。
 */
```
public class FOSDemo {
    public static void main(String[] args) throws IOException {
        /*
        文件输出流常见的构造器：
        FileOutputStream(String path)
        FileOutputStream(File file)
         */
        //向当前项目目录下的fos.dat文件里写入数据
        FileOutputStream fos = new FileOutputStream("./fos.txt");
        //File file = new File("./fos.dat");
        //FileOutputStream fos = new FileOutputStream(file);
```

```
        /*
        OutputStream上定义了所有输出流都应当具备的写出操作：
        void write(int d):
        该方法的作用是写出1个字节，写出的是给定的int值对应的2进制的“低八位”。
         */
        /*
        int型的“1”对应的2进制：
        00000000 00000000 00000000 00000001

        fos.write(1)
        将整数1的2进制低八位写入到文件中
        00000000 00000000 00000000 00000001
                                   ^^^^^^^^
                                   写入文件的2进制部分

        当write()方法执行完毕后，fos.dat文件中的内容为：
        00000001
        */
        fos.write(48); //0
        fos.write(65); //A
        fos.write(97); //a
        System.out.println("写出完毕！");

        fos.close();
    }
}
```

FISDemo

```
package io;

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

//使用文件输入流从文件中读取字节数据
public class FISDemo {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("./fos.txt");

        /*
        InputStream上定义了所有字节输入流都必须具备的读取操作：
        int read()：
        从流中读取一个字节，返回的int值只有“低八位”有效。
        但是如果返回的int值为整数-1，则表示流读取到了末尾。
        （不能再通过这个流读取到任何数据了）
         */

        /*
        fos.dat文件中的数据：
        00000001 00000010

        当我们第一次调用文件输入流的读取字节操作：
        int d = fis.read();
```

```
00000001 00000010
^^^^^^^^
读取的字节

int d接收的返回值2进制的样子：
00000000 00000000 00000000 00000001
|-----会固定补充24个0------| 读取的字节
 */
int d = fis.read();
System.out.println(d);

//当我们第二次调用文件输入流的读取字节操作：（和第一次相同）
d = fis.read();
System.out.println(d);

/*
当我们第三次调用文件输入流的读取字节操作：
d = fis.read();

00000001 00000010
                  ^^^^^^^^
                  文件末尾

int d接收的返回值2进制的样子：
11111111 11111111 11111111 11111111
-1是一个特殊值，这是用正常读取一个字节永远表达不了的2进制样子来表示文件末尾
 */
```

```
        d = fis.read();
        System.out.println(d);

        fis.close();
    }
}
```

CopyDemo

```
package io;

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

public class CopyDemo {
    public static void main(String[] args) throws IOException {
        //创建一个文件输入流，用来从原文件中读取每个字节
        FileInputStream fis = new FileInputStream("./123.png");

        //创建一个文件输出流，用来向复制的文件中写入每个字节
        FileOutputStream fos = new FileOutputStream("123_copy.png");

        /*
        假设image.jpg中的文件内容:
        11001100 00110011 11110000 00001111 10101010 01010101......

        第一次调用:
        int d = fis.read();

        11001100 00110011 11110000 00001111 10101010 01010101......
        ^^^^^^^^
        读取的字节
```

```
        d的2进制样子：00000000 00000000 00000000 11001100
                                                 ^^^^^^^^
        fos.write(d)                             写出的字节

        写出完毕后，image_cp.jpg文件内容：
        11001100
         */
        int d = 0;
        long start = System.currentTimeMillis();
        while ((d = fis.read()) != -1) {
            fos.write(d);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制完毕！耗时" + (end - start) + "ms");

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

Test

```java
package io;

public class Test {
    public static void main(String[] args) {
        /*
        返回当前系统时间的毫秒值（UTC时间，协调世界时）
        从公元1970年1月1日00:00:00开始的偏移量（单位是毫秒）
         */
        long m = System.currentTimeMillis();
        System.out.println(m);

        long max = Long.MAX_VALUE;
        max = max / 1000 / 60 / 60 / 24 / 365;
        System.out.println("公元: " + (1970 + max));
    }
}
```

CopyDemo2

```
package io;

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

/*
提高每次读写的数据量，减少实际读写的次数，可以提高读写效率。
单字节读写是一种随机读写形式，一组字节的读写是块读写形式。
利用块读写形式可以提高读写效率。
 */
public class CopyDemo2 {
    public static void main(String[] args) throws IOException {
        FileInputStream fis = new FileInputStream("./sea.png");
        FileOutputStream fos = new FileOutputStream("./copy.png");

        /*
        java.io.InputStream上定义了块读取字节的操作：
        int read(byte[] data)：
        一次性从流中读取给定的字节数组data的长度的字节量，并存入该字节数组中。
        返回的int值表达的是本次实际读取到的数据量。
        如果返回的int值为整数-1，则表示本次已经是流的末尾了，没有读取到任何数据。

        假设ppt.pptx中的文件内容：
```

```
11001100 00110011 11110000 00001111 10101010 01010101

int d = 0;
byte[] data = new byte[4]; //长度为4的字节数组
```

背景知识：

byte、short、int、long都表示整数，区别是什么？

byte是1个字节，对应的2进制是00000000（8位2进制）

short是2个字节，对应的2进制是00000000 00000000

int是4个字节，

long是8个字节。

读取前：

d的值是整数：0（10进制的样子）

data数组中每个整数2进制的样子：{00000000,00000000,00000000,00000000} //默认都是0

第一次调用：

```
d = fis.read(data);
```

因为data数组长度为4，意味着read()方法可以一次性读取4个字节

文件内容：

```
11001100 00110011 11110000 00001111 10101010 01010101
^^^^^^^^ ^^^^^^^^ ^^^^^^^^ ^^^^^^^^
```

连续读取的4个字节

读取后会将读取到的4个字节存入到data数组中。因此变量内容的变化：

d的值是整数：4，表示本次实际读取到了4个字节

data数组中每个整数2进制的样子：{11001100,00110011,11110000,00001111}

第二次调用：

d = fis.read(data);
因为data数组长度为4，意味着read()方法可以一次性读取4个字节
文件内容：
11001100 00110011 11110000 00001111 10101010 01010101
^^^^^^^^ ^^^^^^^^ ^^^^^^^^ ^^^^^^^^
实际只能读取2个字节 |---文件末尾了----|
连续读取4个字节，但实际只读取到了2个字节
读取后会将读取到的2个字节存入到data数组中。因此变量内容的变化：
d的值是整数：2，表示本次实际读取到了2个字节
data数组中每个整数2进制的样子：{10101010,01010101,11110000,00001111}
|本次实际读取的字节| |---上次的旧数据---|

第三次调用：
d = fis.read(data);
因为data数组长度为4，意味着read()方法可以一次性读取4个字节
文件内容：
11001100 00110011 11110000 00001111 10101010 01010101
^^^^^^^^ ^^^^^^^^ ^^^^^^^^
^^^^^^^^

文件末尾了
连续读取4个字节，但实际没有读取到任何数据（文件已经读到末尾了）
因此变量内容的变化：
d的值是整数：-1，表示本次已经是文件末尾了
data数组中每个整数2进制的样子：{10101010,01010101,11110000,00001111}
数组没有变化，都是旧数据

java.io.OutputStream上定义了所有字节输出流都必须具备的块写操作：
void write(byte[] data)：

```
        一次性将给定的字节数组的所有字节写出。

        void write(byte[] data,int offset,int len)
        一次性将给定的data数组从下标offset处开始的连续len个字节写出
         */

        int d = 0;
        byte[] data = new byte[1024 * 10]; //10kb
        long start = System.currentTimeMillis();
        while ((d = fis.read(data)) != -1) { //传参：data数组
            fos.write(data, 0, d);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制完毕！耗时" + (end - start) + "ms");

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

WriteStringDemo

```java
package io;

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

//向文件中写入文本数据
public class WriteStringDemo {
    public static void main(String[] args) throws IOException {
        /*
        文件流常用的构造器:
        默认为覆盖模式:
        FileOutputStream(String path)
        FileOutputStream(File file)
        上述两种构造器创建文件流时，如果指定的文件已经存在了，则会将该文件原数据全部抹除。

        FileOutputStream(String path,boolean append)
        FileOutputStream(File file,boolean append)
        如果此时第二个参数为true，则文件流为追加模式，即:
        当指定的文件存在时，文件中原数据都保留，使用当前流写出的内容都会陆续追加到文件中去。
         */
        FileOutputStream fos = new FileOutputStream("fos.txt", true);

        String line = "edgver一二三四五gnnewfewsf";
```

```
        //getBytes：字符串转字节数组
        byte[] data = line.getBytes(StandardCharsets.UTF_8);
        fos.write(data);

        line = "追加内容fewgh";
        data = line.getBytes(StandardCharsets.UTF_8);
        fos.write(data);

        System.out.println("写出完毕！");
        fos.close();
    }
}
```

Test2

```java
package io;

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

/*
完成一个简易记事本工具：
程序启动后，将用户在控制台上输入的每一行字符串都写入到文件note.txt中。
当用户单独输入“exit”时程序退出。
 */
public class Test2 {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("./note.txt");
        Scanner scan = new Scanner(System.in);
        while (true) {
            System.out.println("请输入文字，输入exit退出：");
            String line = scan.nextLine();
            if ("exit".equalsIgnoreCase(line)) {
                //若line放equals前面，当line是null时，会抛出空指针异常，所以line应该放equals后面
                //equalsIgnoreCase：忽略大小写的比较
                System.out.println("程序退出");
                break;
```

```
            }
            line += "\r\n"; //换行
            byte[] data = line.getBytes(StandardCharsets.UTF_8);
            fos.write(data);
            System.out.println("写入成功");
        }
        fos.close();
    }
}
```

ReadStringDemo

```java
package io;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

//使用文件流从文件中读取文本数据
public class ReadStringDemo {
    public static void main(String[] args) throws IOException {
        File file = new File("./src/io/ReadStringDemo.java");
        FileInputStream fis = new FileInputStream(file);

        long len = file.length(); //注意返回值为long!
        System.out.println("文件大小：" + len + "字节");

        //整个文件一次性读取，并存到字节数组中
        byte[] data = new byte[(int) len];
        fis.read(data);

        //字节数组转字符串
        String line = new String(data, StandardCharsets.UTF_8);
        System.out.println(line);
```

```
            fis.close();
        }
}
```

Test3

```java
package io;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

//将io包下的每一个java文件的内容输出到控制台上
public class Test3 {
    public static void main(String[] args) throws IOException {
        String address = "./src/io/";
        File dir = new File(address);

        if (dir.isDirectory()) {
            //获取该目录下的所有java文件
            File[] subs = dir.listFiles(f -> f.getName().endsWith(".java"));

            for (int i = 0; i < subs.length; i++) {
                File file = new File(address + subs[i].getName());
                System.out.println(file.getName());

                FileInputStream fis = new FileInputStream(file);

                long len = file.length(); //获取文件的大小（字节量）
```

```
                System.out.println("文件大小：" + len + "字节");

                byte[] data = new byte[(int) len]; //根据文件大小创建等长的字节数组
                fis.read(data); //块读取，一次性将文件所有数据读取到字节数组中

                //利用String的构造器，将字节数组所有内容按照指定的UTF-8字符集还原为字符串
                String line = new String(data, StandardCharsets.UTF_8);
                System.out.println(line);

                fis.close();
            }
        }
    }
}
```

CopyDemo3

```java
package io;

import java.io.*;

/*
缓冲流：
java.io.BufferedInputStream：缓冲字节输入流
java.io.BufferedOutputStream：缓冲字节输出流
 */
public class CopyDemo3 {
    public static void main(String[] args) throws IOException {
        /*
        缓冲流是一对高级流，在流连接中的作用是提高读写效率，具体是将读写操作统一转换为块读写来保证效率的。
        缓冲流默认的缓冲区大小为8kb（内部维护了一个字节数组），其提供的重载构造器可以修改缓冲区大小。
         */
        FileInputStream fis = new FileInputStream("./ppt.pptx");
        BufferedInputStream bis = new BufferedInputStream(fis, 1024 * 10); //指定缓冲区大小为10k
        FileOutputStream fos = new FileOutputStream("./ppt_copy.pptx");
        BufferedOutputStream bos = new BufferedOutputStream(fos, 1024 * 10);

        int d = 0;
        long start = System.currentTimeMillis();
        while ((d = bis.read()) != -1) {
            bos.write(d);
```

```
            }
            bos.flush();
            //bos.close(); //close()自带flush()
            long end = System.currentTimeMillis();
            System.out.println("复制完毕！耗时" + (end - start) + "ms");

            bis.close();
            bos.close();
            /*
            用不带缓存的输入输出流的话，不调用close()方法也可以复制成功，只是这时删除不了源文件（文件被使用）。
            用带缓存的输入输出流的话，如果自己没有调用flush()方法清空缓存，
            那么最后一次读取的内容还会留在缓冲区中，没有输出，
            如果这时调用close()方法关闭输入输出流，那么会自动调用flush()方法，
            强制清空缓冲区的数据，把数据写入文件或者读入程序。
             */
        }
}
```

BOSDemo

```java
package io;

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

//缓冲输出流写出时的缓冲区问题
public class BOSDemo {
    public static void main(String[] args) throws IOException {
        FileOutputStream fos = new FileOutputStream("./bos.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);

        String line = "vwsdwqfn123";
        byte[] data = line.getBytes(StandardCharsets.UTF_8);

        /*
        flush()方法实际上是在接口Flushable上定义的抽象方法，
        而字节输出流超类java.io.OutputStream实现了该接口，
        这意味着所有字节输出流都具有该方法：flush()。
        但不是所有的高级流的flush()方法都是做写出缓冲操作的，
        而是将flush()方法一级级传递下去，
        最终传递到缓冲输出流上完成真实的flush()方法操作。
         */
```

```
        bos.write(data);
        bos.flush();
        System.out.println("写出完毕");

        bos.close();
    }
}
```

Person

```java
package io;

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

//使用此类测试对象流的对象读写操作
public class Person implements Serializable {
    public static final long serialVersionUID = 42L;

    private String name; //姓名
    private int age; //年龄
    private String gender; //性别
    /*
    transient关键字可以修饰一个属性，被修饰的属性在进行对象序列化时会被忽略。
    忽略不必要的属性可以达到对象序列化瘦身的目的。
     */
    private transient String[] otherInfo; //其他信息

    public Person(String name, int age, String gender, String[] otherInfo) {
        this.name = name;
        this.age = age;
        this.gender = gender;
        this.otherInfo = otherInfo;
    }
```

```
    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;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }

    public String[] getOtherInfo() {
        return otherInfo;
    }
```

```
    public void setOtherInfo(String[] otherInfo) {
        this.otherInfo = otherInfo;
    }

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

OOSDemo

```java
package io;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

/*
对象流：java.io.ObjectOutputStream、java.io.ObjectInputStream
对象流是一对高级流，可以方便我们读写任何java对象
 */
public class OOSDemo {
    public static void main(String[] args) throws IOException {
        //将一个Person对象写入文件person.obj
        String name = "苍老师";
        int age = 55;
        String gender = "男";
        String[] otherInfo = {"你们的技术启蒙老师", "拍片一流", "刘桑"};
        Person p = new Person(name, age, gender, otherInfo);

        FileOutputStream fos = new FileOutputStream("./person.obj");
        ObjectOutputStream oos = new ObjectOutputStream(fos);

        /*
        当进行对象序列化时，可能出现异常：
```

```
        java.io.NotSerializableException
        这是由于写出的对象没有实现序列化接口导致的。

        对象序列化：将一个对象按照其结构转换为一组字节的过程
        数据持久化：将数据写到磁盘上做长久保存
         */
        oos.writeObject(p);
        System.out.println("写出完毕");

        oos.close();
    }
}
```

OISDemo

```java
package io;

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

//使用对象输入流进行反序列化
public class OISDemo {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        FileInputStream fis = new FileInputStream("./person.obj");
        ObjectInputStream ois = new ObjectInputStream(fis);

        /*
        反序列化对象：
        该方法可能会抛出异常：java.io.InvalidClassException
        原因：对象输入流在反序列化对象时，发现该对象的序列化版本号与当前类定义的版本号不一致。
        一个类实现了序列化接口后，会有一个序列化版本号，
        只要类的结构没有发生过改变，那么该版本号的值是不会发生变化的，
        一旦类的结构发生了改变，那么版本号就会发生改变。
         */
        Person p = (Person) ois.readObject();
        //public final Object readObject()方法返回Object类型，所以要向下造型（强转）为Person类型
        System.out.println(p);
        System.out.println(p.getName());
```

```
        ois.close();
    }
}
```

OSWDemo

```
package io;

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

/*
java将流按照读写单位划分为：字节流、字符流
字节流：java.io.InputStream、java.io.OutputStream
字符流：java.io.Reader、java.io.Writer
Reader为所有字符输入流的超类，定义了所有字符输入流都必须具备的读取字符的方法
Writer为所有字符输出流的超类，定义了所有字符输出流都必须具备的写出字符的方法

字符流与字节流的区别:
读写单位不同，
字节流以一个字节（byte，8位2进制）为单位读写数据，
字符流则以一个字符（char）为单位读写数据，
所以字符流天生只适合读写【文本数据】。

转换流：java.io.InputStreamReader、java.io.OutputStreamWriter
转换流是一对字符流（分别继承自Reader和Writer），并且是一对高级流
转换流在实际开发中我们不会直接操作它们，但是在流连接中它们又是不可缺少的（在组件含有字符高级流的流连接中）
 */
```

```java
public class OSWDemo {
    public static void main(String[] args) throws IOException {
        //向文件osw.txt中写入文本数据
        FileOutputStream fos = new FileOutputStream("./osw.txt");
        OutputStreamWriter osw = new OutputStreamWriter(fos, StandardCharsets.UTF_8);

        /*
        Writer（字符输出流超类）上定义了所有字符输出流都必须具备的写出文本数据的相关方法：
        void write(String str)
         */
        String line = "eferwfwedqw";
        osw.write(line);
        osw.write("\r\n");
        osw.write("1234235676658");
        System.out.println("写出完毕");

        osw.close();
    }
}
```

ISRDemo

```java
package io;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

//使用转换输入流读取文本数据
public class ISRDemo {
    public static void main(String[] args) throws IOException {
        //将当前源代码输出到控制台上
        FileInputStream fis = new FileInputStream("./src/io/ISRDemo.java");
        InputStreamReader isr = new InputStreamReader(fis);

        /*
        字符流java.io.Reader上提供了所有字符输入流都应具备的读取字符的方法:
        int read():
        一次读取一个字符，返回的int值实际上表示一个char
        如果返回的int值为-1，则表示流读取到了末尾
         */
        int d = 0;
        while ((d = isr.read()) != -1) {
            System.out.print((char) d);
        }
```

```
            isr.close();
        }
    }
```

PWDemo

```
package io;

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

/*
缓冲字符流:
java.io.BufferedWriter、java.io.BufferedReader
缓冲字符流是一对字符流，且是一对高级流。作用是提高读写文本数据的效率。
内部默认维护一个8k长度的char数组（注意，不是8kb！是一个8192长度的char数组）

java.io.PrintWriter是具有自动行刷新功能的缓冲字符输出流。
其内部总是链接着BufferedWriter，并且可以按行写出字符串。
 */
public class PWDemo {
    public static void main(String[] args) throws FileNotFoundException,
UnsupportedEncodingException {
        /*
        支持直接对文件写操作的构造器:
        PrintWriter(File file)
        PrintWriter(String path)
        上述构造器内部会进行四层流连接完成对文件的写出操作
        详见代码下方图片：PW.png
```

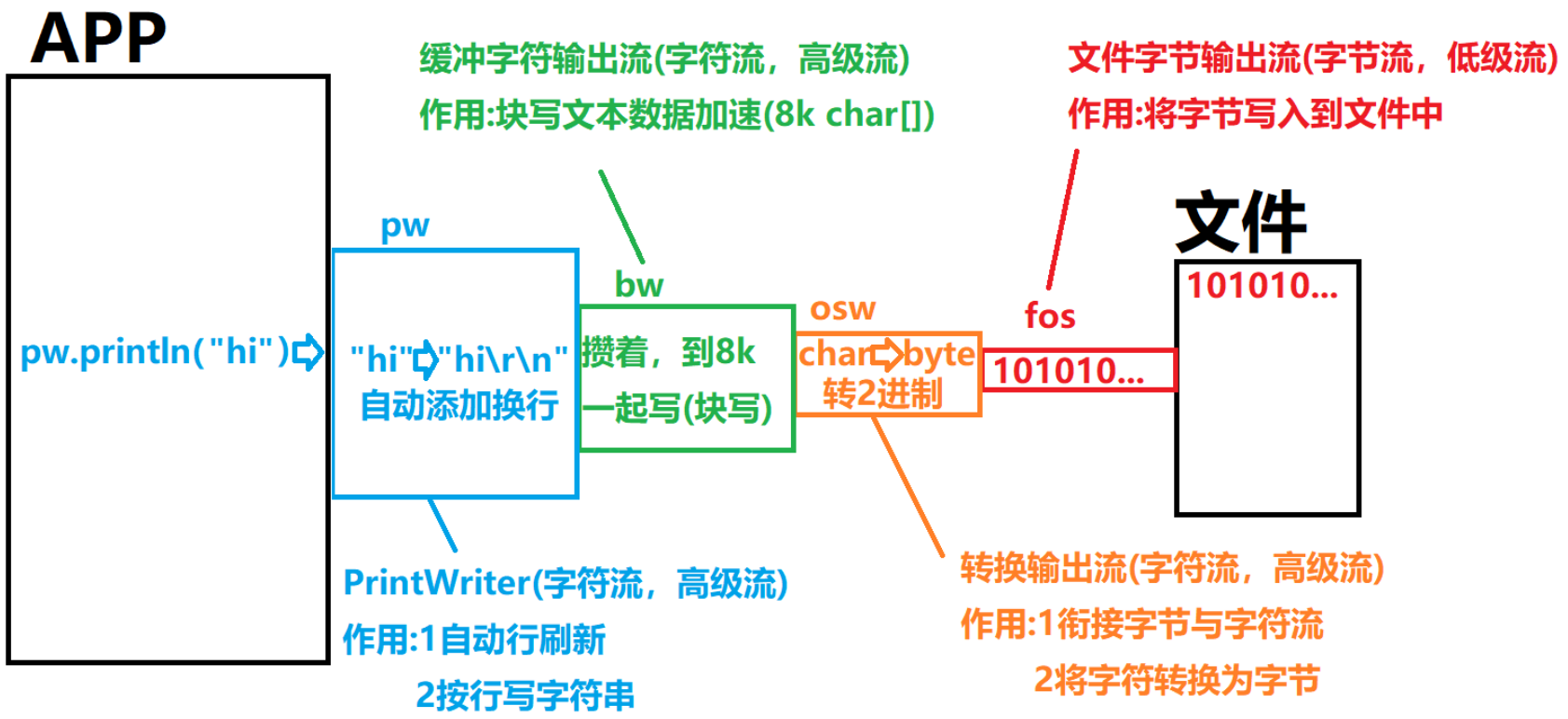
```
        重载构造器（推荐）：
        PrintWriter(File file,String csn)
        PrintWriter(String path,String csn)
        支持用第二个参数来指定字符集的名称
        不理想的地方：字符集的名称需要用字符串自行指定
         */

        //向文件中写入文本数据
        PrintWriter pw = new PrintWriter("./pw.txt", "UTF-8");
        //或者：String.valueOf(StandardCharsets.UTF_8)

        pw.println("test123");
        pw.println("371vdsvc");
        System.out.println("写出完毕");

        pw.close();
    }
}
```

```
this(new BufferedWriter(new OutputStreamWriter(new FileOutputStream(fileName))))
```



PWDemo2

```java
package io;

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

//在流连接中使用PrintWriter
public class PWDemo2 {
    public static void main(String[] args) throws FileNotFoundException {
        //向pw2.txt文件中写入字符串
        FileOutputStream fos = new FileOutputStream("./pw2.txt", true);
        OutputStreamWriter osw = new OutputStreamWriter(fos, StandardCharsets.UTF_8);
        BufferedWriter bw = new BufferedWriter(osw);
        PrintWriter pw = new PrintWriter(bw, true);
        /*
        PrintWriter提供了可以打开自动行刷新功能的构造器，
        当第一个参数为一个流时，就支持再传入一个boolean值，
        如果该值为true，则打开了自动行刷新，
        此时每当我们调用一次println()方法，就会自动调用flush()方法。
        只有println()方法才能自动行刷新，print()方法不行。
         */

        /*
        完成记事本功能：
```

```
        程序启动后，将用户在控制台输入的每一行字符串都按行写入文件中。
        当用户单独输入“exit”时，程序退出。
         */
        Scanner scan = new Scanner(System.in);
        while (true) {
            System.out.println("请输入文字，输入exit退出：");
            String line = scan.nextLine();
            if ("exit".equals(line)) {
                System.out.println("程序退出");
                break;
            }
            pw.println(line);
            System.out.println("保存成功");
        }

        pw.close();
    }
}
```

BRDemo

```
package io;

import java.io.*;

/*
使用缓冲字符输入流读取文本数据

java.io.BufferedReader:
特点:
块读文本数据
可按行读取字符串
 */
public class BRDemo {
    public static void main(String[] args) throws IOException {
        //将当前源代码输出到控制台上
        FileInputStream fis = new FileInputStream("./src/io/BRDemo.java");
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);

        /*
        BufferedReader提供了一个独有的方法:
        String readLine():
        该方法会返回读取到的一行字符串，返回的字符串不含最后的换行符。
        如果单独读取了空行（这一行只有换行符），则返回值为空字符串。
```

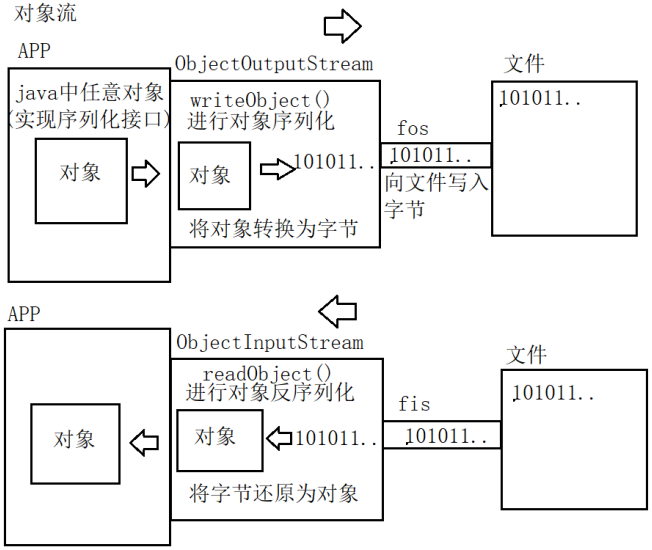
```
        如果返回值为null，则表示流读取到了末尾。

        当第一次调用readLine()读取一行字符串时，
        实际上：缓冲流会一次性读取8192个字符（char）并存入到其内部字符数组中（块读操作），
        再将其中第一行字符串的内容返回。
        当我们第二次调用readLine()时，不会再次读取，而是直接将内部char数组中第二行字符串的内容返回。
         */
        String line;
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }

        br.close();
    }
}
```

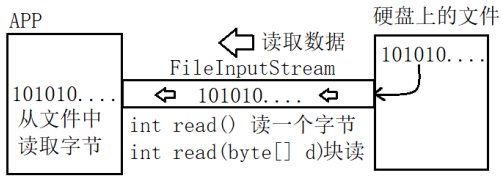
	输入流		输出流	
	字节输入流 java.io.InputStream	字符输入流 java.io.Reader	字节输出流 java.io.OutputStream	字符输出流 java.io.Writer
节点流 低级流	FileInputStream 文件输入流 作用: 实际连接程序与文件的管道 负责从文件中读取字节		FileOutputStream 文件输出流 作用: 实际连接程序与文件的管道 负责将字节写入文件中	
处理流 高级流	BufferedInputStream 缓冲字节输入流 作用: 块读字节数据加速 ObjectInputStream 对象输入流 作用: 进行对象反序列化	InputStreamReader 转换输入流 作用 1:衔接字节与字符流 2:将读取的字节转字符 BufferedReader 缓冲字符输入流 作用 1:块读文本数据加速 2:按行读取字符串	BufferedOutputStream 缓冲字节输出流 作用: 块写字节数据加速的 ObjectOutputStream 对象输出流 作用: 进行对象序列化	OutputStreamWriter 转换输出流 作用: 1:衔接字节与字符流 2:将写出的字符转字节 PrintWriter 缓冲字符输出流 带自动行刷新功能 作用 1:块写文本数据加速 2:按行写出字符串 3:自动行刷新

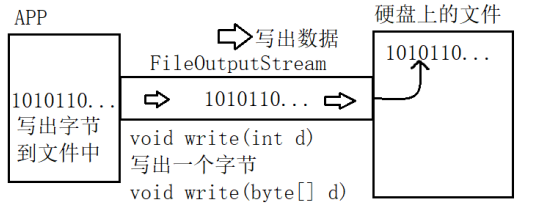
对象流



对象流作用:方便我们读写java对象，不再考虑对象与字节的转换工作

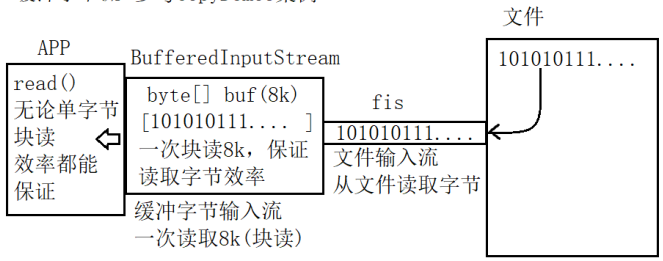
字节流相关图示

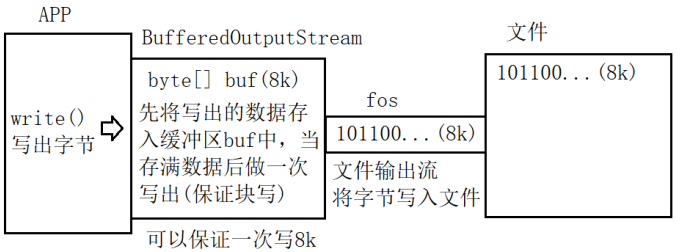




void write(int d)
写出一个字节
void write(byte[] d)
块写字节
void write(byte[] d,int offset,int len)
块写操作(常用在复制文件时的写出操作)

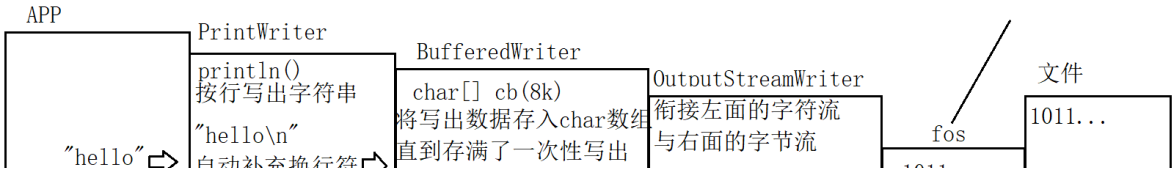
缓冲字节流 参考CopyDemo3案例

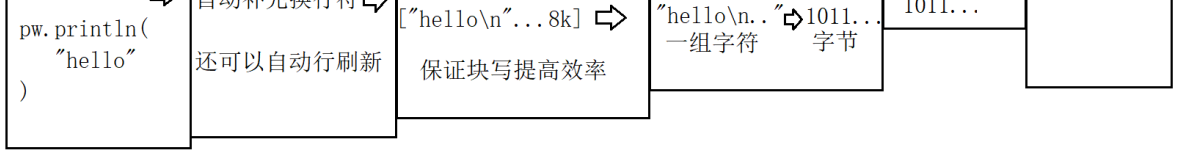




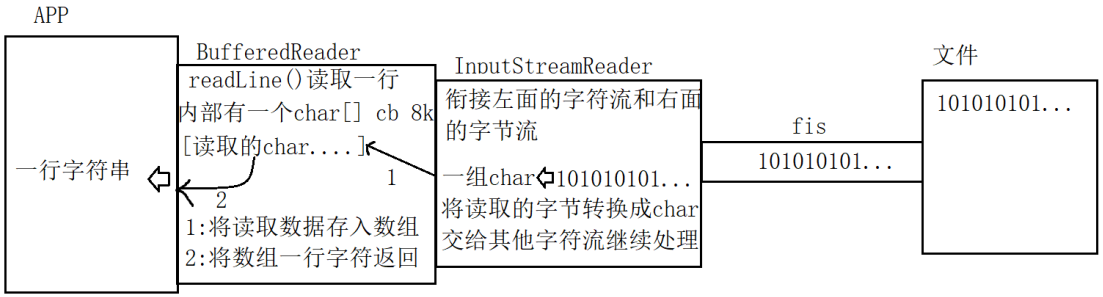
字符流相关图示

文件字节输出流(字节流，低级流)
将字节写入到文件中





使用PrintWriter完成上述一系列流连接目的就是可以方便的按行写出字符串还能保证写出效率



第一次调用readLine就一次性读取8192个字符存入
缓冲区(char数组)然后将其中第一行内容返回。
当再次调用readLine时，再将第二行内容直接返回。
直到数组所有内容均返回后会再次读取8192个字符，
如此往复以块读取保证效率。

使用BufferedReader完成上述流连接目的是可以按行读取字符串并且还可以保证读取效率