

# IO流

## 一、概念

I : input

O : output

输入输出流。

什么叫流？

流水。流是一组有顺序的，有起点和终点的字节集合，是对数据传输的总称或抽象，即数据在两设备间的传输称为流，流的本质是数据传输

## 二、IO流的分类

根据数据流向不同分为：输入流和输出流

输入流：只能从中读取数据，而不能向其写入数据。

输出流：只能向其写入数据，而不能向其读取数据。

java的输入流主要是InputStream和Reader作为基类，而输出流则是主要由OutputStream和Writer作为基类。它们都是一些抽象基类，无法直接创建实例。

根据处理数据类型的不同分为：字符流（Reader、Writer）和字节流（InputStream、OutputStream）

字节流和字符流的用法几乎完成全一样，区别在于字节流和字符流所操作的数据单元不同，字节流操作的单元是数据单元是8位的字节，字符流操作的是数据单元为16位的字符。

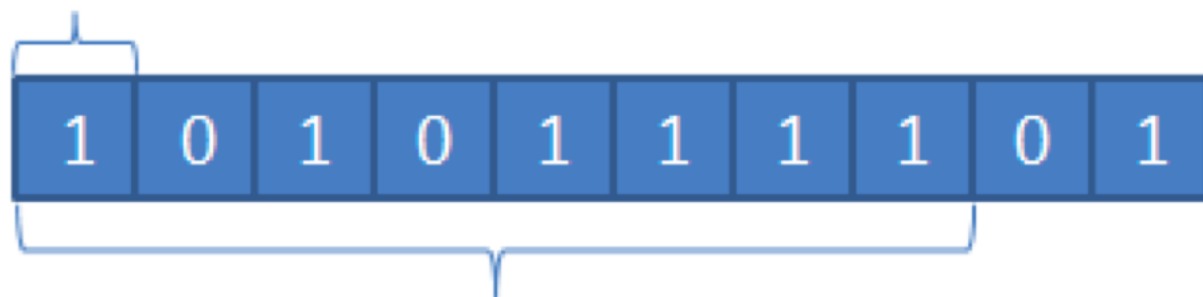
### 2.1 字符和字节的区别

**字节：**

byte

计算机中存储数据的一个单位。比它小的是位（bit，也叫比特），bit是在计算机中数据存储的最小计量单位，每一个bit位，存放的是二进制的数字0和1，如下所示。

1 bit



1 Byte = 8 bits

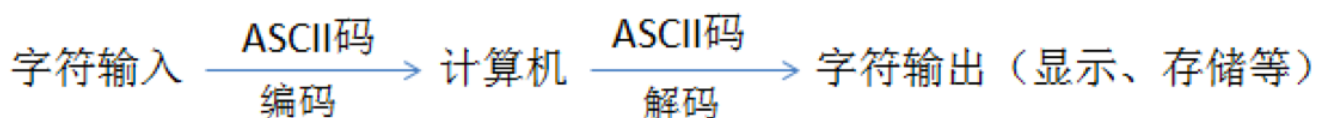
当然比字节更大的是KB（千字节），1KB = 1024B，再到后面就是MB（兆字节），1MB = 1024KB，GB、TB……

Java中有用于表示字节的数据类型——byte。

## 字符

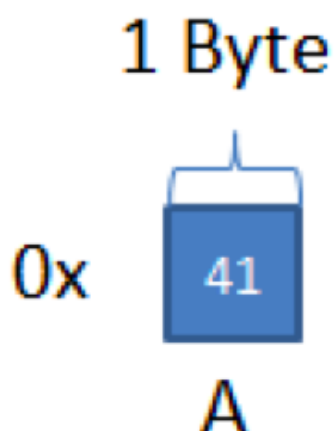
字符表示文字和符号。人与人之间通过人类语言进行沟通，计算机通过二进制来进行沟通，当人-计算机-人，中间多了计算机的媒介过后，中间就需要计算机对我们人类的语言符号“编码”进行传输，而计算机-人这个过程又称之为“解码”。这有点类似“加密”“解密”的过程。

在计算机刚出现的时候只能传输英文字符，这里的传输包括是显示和存储，前面提到要进行编码存储，既然要编码就需要一张表来表示A是什么，B是什么，就好比摩斯密码中的密码本一样。那时的“码表”也就是编码方式叫做ASCII。

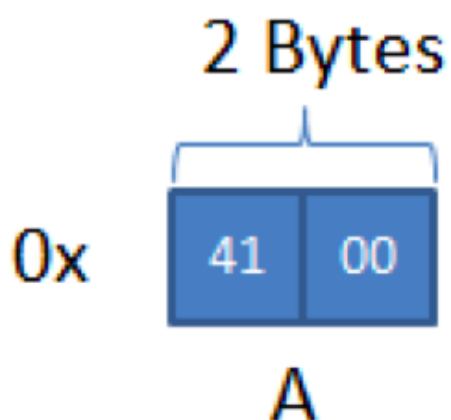


在Java中使用的就是UNICODE编码，这符合Java跨平台的特性，这也就解释了Java中char字符的数据类型占用的是2个字节，因为Java使用UNICODE编码，而UNICODE是2个字节表示1个字符

ASCII



UNICODE



可以看出，这就白白地浪费掉了1个字节的空間，

由于UNICODE给任意字符都是采用的2个字节表示1个字符，会造成空间浪费，所以在UNICODE编码基础上，又出现了可变长编码的UTF-8编码，这种编码方式会灵活地进行字符的空间分配，不同字符所占用的内存空间不相同，在保证兼容性的同时，也保证了空间的最合理使用。

## 2.2 字符流和字节流

###

字符流的由来：因为数据编码的不同，而有了对字符进行高效操作的流对象。本质其实就是基于字节流读取时，去查了指定的码表。字节流和字符流的区别：

（1）读写单位不同：字节流以字节（8bit）为单位，字符流以字符为单位，根据码表映射字符，一次可能读多个字节。

（2）处理对象不同：字节流能处理所有类型的数据（如图片、avi等），而字符流只能处理字符类型的数据。

(3) 字节流在操作的时候本身是不会用到缓冲区的，是文件本身的直接操作的；而字符流在操作的时候下后是会用到缓冲区的，是通过缓冲区来操作文件，我们将在下面验证这一点。

结论：优先选用字节流。首先因为硬盘上的所有文件都是以字节的形式进行传输或者保存的，包括图片等内容。但是字符只是在内存中才会形成的，所以在开发中，字节流使用广泛。

字节流和字符流的处理方式其实很相似，只是它们处理的输入/输出单位不同而已。输入流使用隐式的记录指针来表示当前正准备从哪个“水滴”开始读取，每当程序从InputStream或者Reader里面取出一个或者多个“水滴”后，记录指针自动向后移动；除此之外，InputStream和Reader里面都提供了一些方法来控制记录指针的移动。

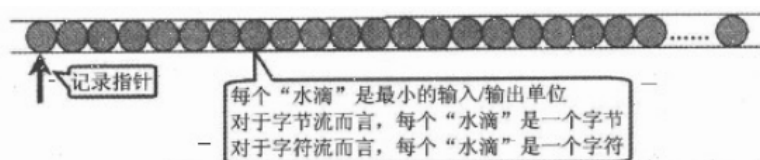


图 15.5 输入流模型图

当执行输出时，程序相当于依次把“水滴”放入到输出流的水管中，输出流同样采用隐式指针来标识当前水滴即将放入的位置，每当程序向OutputStream或者Writer里面输出一个或者多个水滴后，记录指针自动向后移动。

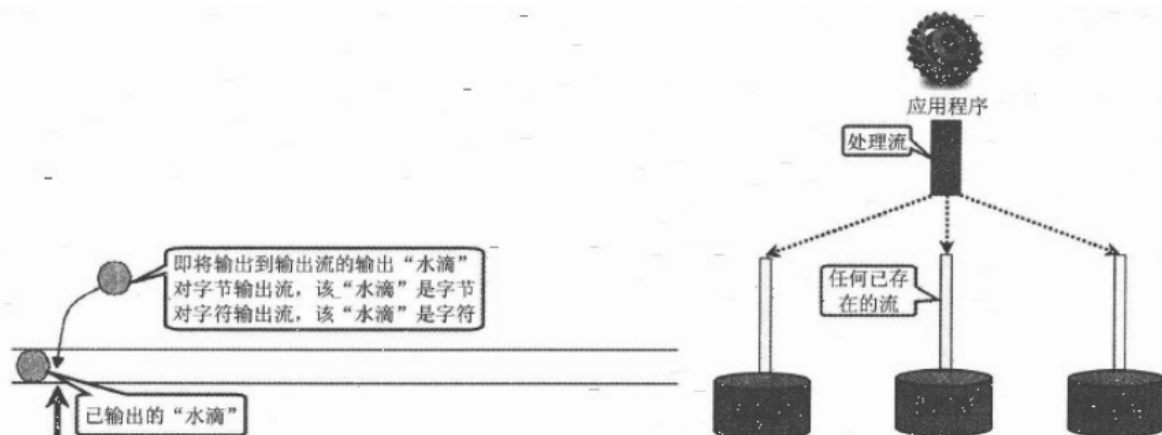


图 15.6 输出流模型图

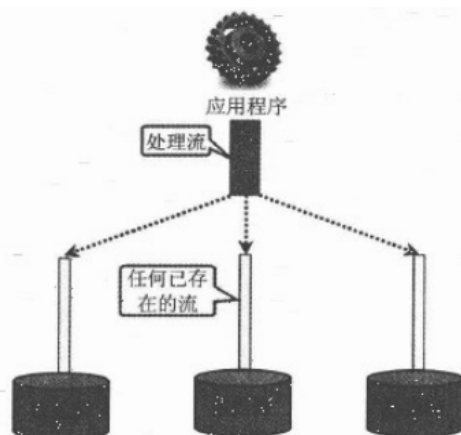


图 15.7 处理流模型图

### 三、java中IO的结构

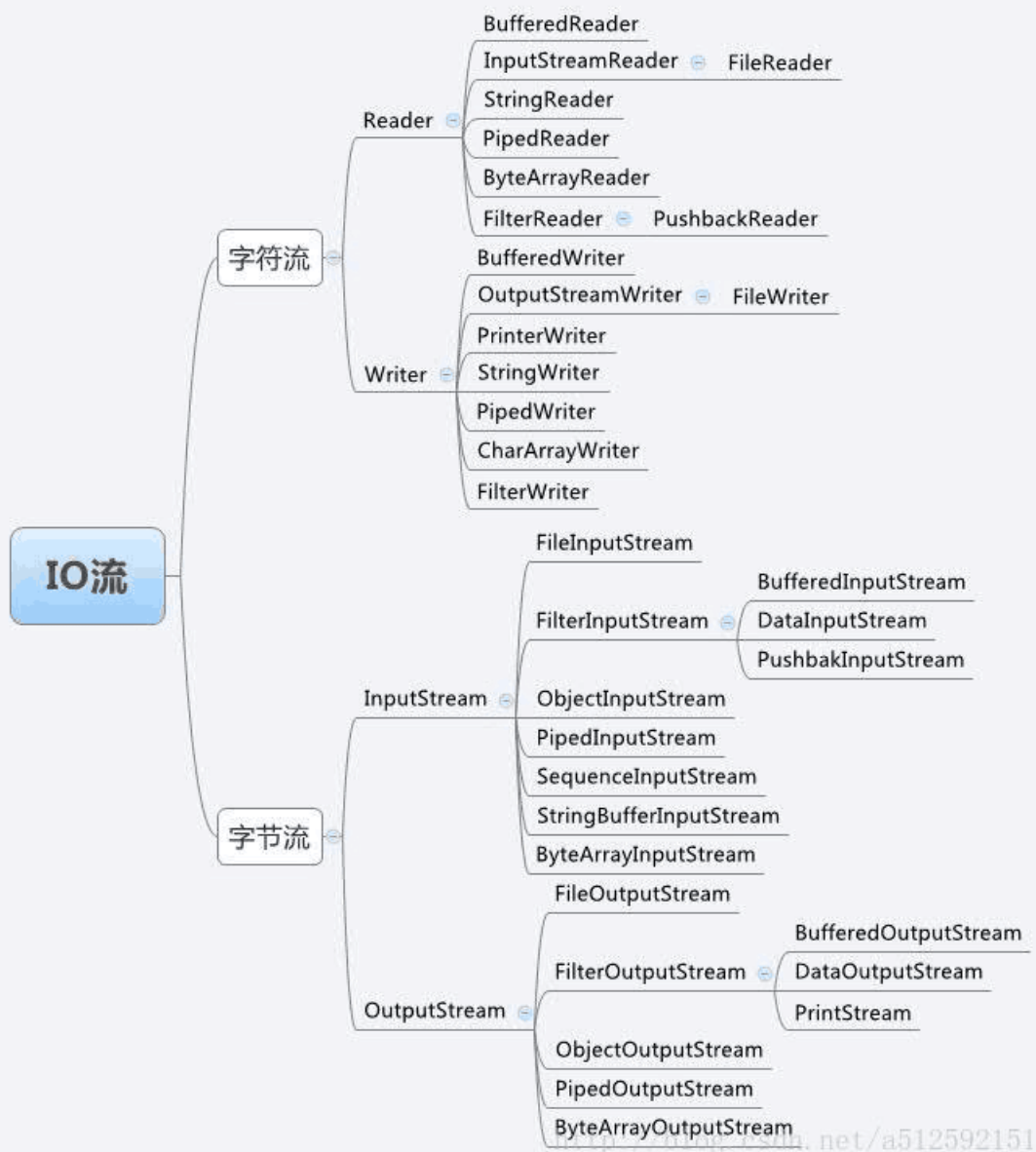
java把所有的传统的流类型都放到在java io包下，用于实现输入和输出功能。

还有新出的java.nio，aio。

传统IO：同步阻塞IO

nio：同步非阻塞

aio：异步非阻塞



java输入/输出流体系中常用的流的分类表

分类	字节输入流	字节输出流	字符输入流	字符输出流
抽象基类	InputStream	OutputStream	Reader	Writer
访问文件	<b>FileInputStream</b>	<b>FileOutputStream</b>	<b>FileReader</b>	<b>FileWriter</b>
访问数组	<b>ByteArrayInputStream</b>	<b>ByteArrayOutputStream</b>	<b>CharArrayReader</b>	<b>CharArrayWriter</b>
访问管道	<b>PipedInputStream</b>	<b>PipedOutputStream</b>	<b>PipedReader</b>	<b>PipedWriter</b>
访问字符串			<b>StringReader</b>	<b>StringWriter</b>
缓冲流	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
转换流			InputStreamReader	OutputStreamWriter
对象流	ObjectInputStream	ObjectOutputStream		
抽象基类	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter

分类	字节输入流	字节输出流	字符输入流	字符输出流
打印流		PrintStream		PrintWriter
推回输入流	PushbackInputStream		PushbackReader	
特殊流	DataInputStream	DataOutputStream		

## 四、文件的读取和写入

要想读取或者写入文件，必须要有文件对象。

读取：FileInputStream

写入：FileOutputStream

文件对象：File

### 4.1 File类

java.io.File

File对象代表磁盘中实际存在的文件（File）和目录（Directory）。

File 类是 [Java.io](#) 包中唯一代表磁盘文件本身的对象

File类定义了一些与平台无关的方法来操作文件，可以通过调用 File 类中的方法，实现创建、删除、重命名文件等操作。

File 类的对象主要用来获取文件本身的一些信息，如文件的路径、文件名、文件所在的目录、文件的长度、文件读写权限、文件是否可以执行、文件是否是隐藏属性等等

常用方法

//获取文件的基本信息

getName() //获取文件名

getParent() //获取上层目录的路径，如果是根目录，返回null

getPath() //获取文件的路径

length() //获取文件内容的长度

renameTo() //重命名或者移动文件

//1、如果目标位置与当前文件所在的位置一样，该方法是重命名的作用

//2、如果目标位置与当前文件所在的位置不一样，该方法的作用是移动文件（剪切）

//如果目标目录下，已经有同名的文件，则不处理。

//修改文件属性

set开头的方法，可以设置文件的：执行、只读、读取、写入等属性。

/\*

can、is、list三组方法

\*/

public static void main(String[] args) {

//File.separator是File提供的字段，获取到路径的分隔符。linux和windows下，分隔符是不一样的  
//这样使用的好处是，跨平台

File file = new File("E:" + File.separator + "abcd123.txt");

String fileName = file.getName();

System.out.println(fileName);

```

//can开头的方法，获取文件是否可以被执行，写入，读取
System.out.println(file.canRead());
System.out.println(file.canWrite());
System.out.println(file.canExecute());

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

//is开头的方法，判断文件的路径是否是绝对路径，是否是目录、是否是文件、是否是隐藏属性
System.out.println(file.isDirectory());
System.out.println(file.isAbsolute());
System.out.println(file.isFile());
System.out.println(file.isHidden());
System.out.println("=====");

//list()方法，返回该目录下所有的文件和子目录的名字，不包括子目录里的内容。
String[] list = file.list();
for (String string : list) {
    System.out.println(string);
}

//listFiles()，返回一个file数组，表示该目录下所有的文件对象。
System.out.println("=====");
File[] listFiles = file.listFiles();
for (File file2 : listFiles) {
    System.out.println(file2.getName());
}

}

```

//文件的操作

```

package com.psf.d.io;

import java.io.File;
import java.io.IOException;

public class Demo {
    public static void main(String[] args) throws IOException {
        //File.separator是File提供的字段，获取到的是路径的分隔符。linux和windows下，分隔符是不一样的
        //这样使用的好处是，跨平台
        File file = new File("E:" + File.separator + "demo234" + File.separator + "abcdtest");
        String fileName = file.getName();
        System.out.println(fileName);
        System.out.println(file.getPath());

        // 创建新文件（仅仅适用于文件，不能适用目录）
        //如果给出的一个目录，使用该方法，会创建一个不带后缀的纯文本的文件
        //如果文件已经存在，不搭理。也不覆盖
        // file.createNewFile();

        //创建目录,如果有同名的文件或者目录，也不会创建。
        System.out.println(file.mkdir());

        //如果创建的目录，其父目录也不存在，使用这个方法，可以将父目录全部创建
        System.out.println(file.mkdirs());
    }
}

```

```

        System.out.println(file.canRead());
    }
}

```

## 4.2 （字节流）文件读取和写入

既然是读取数据，必须使用输入流。（InputStream或者Reader）

```

package com.psfd.io;

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

public class Demo {
    public static void main(String[] args) throws IOException {
        File file = new File("E:/demo234/abc.txt");

        //FileOutputStream用来往文件中写入数据
        //第二个参数，boolean，表示是否是追加。如果为false（默认），覆盖。如果是true，则追加（追加到末尾）
        //如果需要换行，自己加入换行符
        FileOutputStream fileOutputStream = new FileOutputStream(file,true);
        String msg = "Welcome to PSFD!";
        fileOutputStream.write(msg.getBytes());
        fileOutputStream.close();

        //FileInputStream用来读取文件的数据
        FileInputStream fileInputStream = new FileInputStream(file);

        //判断当前流是否支持mark和reset
        System.out.println("isSupport = " + fileInputStream.markSupported());

        int read = fileInputStream.read();
        //在流的当前指针的位置，打上一个标记
        // fileInputStream.mark(read);

        //逐个字节的读取。通过这个例子，告诉我们：流的读取，是指针的指向读取，每次只读一个字节
        // while (read != -1) {
        //     System.out.println(String.valueOf(read));
        //     read = fileInputStream.read();
        // }
        // System.out.println("=====");

        //将流的指针，复位到上次打标记的地方
        // fileInputStream.reset();
        byte[] data = new byte[1024];
        int readLength = fileInputStream.read(data);
        System.out.println(new String(data,0,readLength));

        fileInputStream.close();
    }
}

```



## 4.3 字节流的乱码

字节流在读取中文的时候，可能会出现乱码的问题。

什么是乱码？

造成部分或所有字符无法被阅读的一系列字符。

我们都知道，一个中文占用2个字节，而字节流是逐个字节读取的机制。

当字节流只读取了一个中文的一个字节的时候，那就意味着，中文没有读取完全，无法被解析，只能显示乱码。

```
package com.psfed.io;

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

public class Demo {
    public static void main(String[] args) {
        File file = new File("E:/demo234/abc.txt");
        // FileInputStream用来读取文件的数据
        FileInputStream fileInputStream = null;
        try {
            fileInputStream = new FileInputStream(file);

            int read = fileInputStream.read();
            while (read != -1) {
                System.out.println((char)read);
                read = fileInputStream.read();
            }
            // byte[] data = new byte[1024];
            // int readLength = fileInputStream.read(data);
            // System.out.println(new String(data, 0, readLength));
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

如何解决乱码的问题

在IO流中，使用字符流来读取就可以避免这个问题。

但是，不是说当你出现乱码的时候，就要把字节流换成字符流。

## 4.4 （字符流）文件的读取和写入

```
//单字符读取
```

```

package com.psfed.io;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class Demo {
    public static void main(String[] args) {
        File file = new File("E:/demo234/abc.txt");
        FileReader fileReader = null;
        try {
            fileReader = new FileReader(file);

            int read = fileReader.read();
            System.out.println((char)read);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                fileReader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

//统一读取

```

package com.psfed.io;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.nio.CharBuffer;

public class Demo {
    public static void main(String[] args) {
        File file = new File("E:/demo234/abc.txt");
        FileReader fileReader = null;
        try {
            fileReader = new FileReader(file);

            char cbuf[] = new char[(int)file.length()];
            fileReader.read(cbuf);

            System.out.println(new String(cbuf));
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                fileReader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

## 4.5 IO的性能初步分析

不管是原始的字节流，还是原始的字符流，都是逐个字符或者字节的读取和写入。

这样会有一个问题：

会导致的频繁的操作磁盘（底层系统），当数据量达到一定的程度的时候，性能会出现很严重的问题。

这也是一个软件，在编码完成以后，会进行测试，其中有一项测试叫：性能测试。

性能测试又有一个非常重要的指标：IO消耗。

所以，在我们现在初级阶段，只需要明白，记住一句话：过于频繁的IO操作，会导致性能的急剧下降。

为了避免这个问题，我们要在原始的输入输出流的基础上，再做包装。

如何包装？不要让输入或者输出流，逐个字节或者字符的操作，而是将字节或者字符缓冲在内存中，等到最后，一起写入到磁盘。

## 4.6 缓冲流 Buffer

java缓冲流本身不具IO功能，只是在别的流上加上缓冲提高效率，像是为别的流装上一一种包装。当对文件或其他目标频繁读写或操作效率低，效能差。这时使用缓冲流能够更高效的读写信息。因为缓冲流先将数据缓存起来，然后一起写入或读取出来。所以说，缓冲流还是很重要的，在IO操作时记得加上缓冲流提升性能。

缓冲流分为字节和字符缓冲流

字节缓冲流为：

BufferedInputStream—字节输入缓冲流

BufferedOutputStream—字节输出缓冲流

字符缓冲流为：

BufferedReader—字符输入缓冲流

BufferedWriter—字符输出缓冲流

//BufferedOutputStream类实现缓冲的输出，通过设置这种输出流，应用程序就可以将各个字节写入底层输出流中，而不必每一个字节写入都调用底层系统。

```
package com.psf.d.io;
```

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

```
import java.io.OutputStream;

public class Demo {
    public static void main(String[] args) {
        try {
            // 创建字节输出流实例
            OutputStream out = new FileOutputStream("E:\\test\\test123.txt", true);
            // 根据字节输出流构建字节缓冲流
            BufferedOutputStream buf = new BufferedOutputStream(out);

            String data = "好好学习,天天向上";

            buf.write(data.getBytes()); // 写入缓冲区
            buf.flush(); // 刷新缓冲区, 即把内容写入
            // 关闭流
            buf.close(); // 关闭缓冲流时, 也会刷新一次缓冲区
            out.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

//其他的缓冲流, 参考:
https://www.cnblogs.com/techfox/p/4522734.html
```

## 作业：

在线考试系统，全部写入到文件中保存数据。

不能全部写入到同一个文件。

章节、题目、科目、用户、成绩信息都是独立的文件保存。

文件名以及路径，最好不要使用中文和特殊字符。

