



面向对象与多线程综合实验

结合华为云平台的基于JAVASE数
据挖掘系统设计与实现



主要内容

一

• 实验目标和要求

二

• 实验内容

三

• 知识要点



系统简介

- 基于Java SE 的数据挖掘系统

基于客户端服务器端（Client-Server, C-S）模式，实现日志与物流数据信息的采集、匹配、保存、显示等功能，为数据分析挖掘提供基础支撑。

The screenshot displays the user interface of the data mining system client, which is divided into three main windows:

- 登录 (Login) Window:** Contains fields for "用户名:" (Username) with the value "YYC" and "密码:" (Password) with masked characters "****". It includes "登录" (Login), "重置" (Reset), and "注册" (Register) buttons.
- 用户注册 (User Registration) Window:** Contains fields for "用户名:" (Username), "密码:" (Password), and "确认密码:" (Confirm Password). It also includes a "性别:" (Gender) section with radio buttons for "男" (Male) and "女" (Female), an "爱好:" (Hobbies) section with checkboxes for "阅读" (Reading), "上网" (Internet), "游泳" (Swimming), and "旅游" (Travel), a "地址:" (Address) field, and a "学历:" (Education) dropdown menu currently set to "小学" (Primary School). It includes "确定" (Confirm) and "重置" (Reset) buttons.
- 主界面 (Main Interface) Window:** Titled "欢迎进入数据挖掘系统客户端!", it features a menu bar with "操作" (Operation) and "帮助" (Help). Below the menu bar are six buttons: "采集数据" (Collect Data), "匹配日志数据" (Match Log Data), "匹配物流数据" (Match Logistics Data), "保存数据" (Save Data), "发送数据" (Send Data), and "显示数据" (Display Data). The interface has two tabs: "日志" (Log) and "物流" (Logistics). The "日志" tab is active, showing fields for "日志ID:", "用户名:", "登录地点:", "登录IP:", and "登录状态:" (with radio buttons for "登录" (Login) and "登出" (Logout)). It also includes "确认" (Confirm) and "重置" (Reset) buttons at the bottom.



需求分析

- 系统包括客户端应用程序、服务器端应用程序---C/S模式
- 用户和数据信息的保存---JDBC数据库保存和查询
- 用户能够进行注册和登录，授权后使用系统---GUI登录和注册界面设计与功能实现
- 能够实现日志和物流信息的数据采集（录入），登录登出对匹配、信息保存和数据显示等功能---GUI主界面设计与功能实现
- 系统能够进行数据自动刷新功能，与数据库保持同步---线程
- 客户端与服务器端交互，客户端能够将数据发送到服务器端，服务器端接收客户端发送的日志和物流信息，进行保存和处理---Socket通信
- 系统优化---JAVA高级应用与新特性



任务分配

课时分配	实验任务	任务列表	主要知识点分解
第1-2次课	基于控制台的系统数据 采集、匹配、显示 和 记录 功能实现	<ul style="list-style-type: none">✓ 注册华为软开云账户✓ 搭建数据挖掘系统框架✓ 实现日志和物流数据信息的采集、匹配和显示功能✓ 实现匹配的日志物流数据信息的文件保存和读取记录功能	继承与多态/异常处理/集合/文件存储及IO流/华为软开云
第3次课	基于 JDBC 的控制台系统基本功能实现	<ul style="list-style-type: none">✓ 创建项目所需的数据库表，并搭建数据访问基础环境✓ 实现并测试匹配的日志、物流信息的数据库保存和查询功能	JAVAJDBC/MySQL
第4次课	基于SwingGUI的系统 注册 和 登录界面 设计实现	<ul style="list-style-type: none">✓ 创建用户数据库表、用户实体类和用户业务逻辑类✓ 创建用户注册窗口，并将用户注册信息保存到数据库✓ 创建用户登录窗口，登录成功则进入系统主界面	SwingGUI/事件驱动/WinBuilder
第5次课	基于SwingGUI系统 主界面 设计实现和系统 优化	<ul style="list-style-type: none">✓ 实现主界面中的菜单和工具栏✓ 实现主界面中的日志和物流数据采集、匹配、保存和显示功能✓ GUI系统优化	高级UI组件
第6次课	系统信息 自动刷新 功能实现	<ul style="list-style-type: none">✓ 使用线程实现每隔1分钟(或自定义)日志和物流显示数据表格自动刷新功能，以便与数据库保持同步	线程
第7次课	系统客户端/服务器端数据 发送(交互) 功能实现	<ul style="list-style-type: none">✓ 客户端应用程序：修改主界面发送数据页面响应，即使用Socket实现数据由客户端发送到服务器✓ 服务器端应用程序：使用Server Socket实现接收客户端发送的日志和物流数据信息，并将信息保存到数据库	Socket网络编程
第8次课	系统验收	<ul style="list-style-type: none">✓ 数据挖掘系统的基本功能实现与演示✓ 在华为软开云平台完成系统需求分析和设计反推	PPT演示、系统运行
拓展	Java 高级应用以及Java8新特性	<ul style="list-style-type: none">◆ 使用注解重新迭代升级系统代码◆ 使用格式化将输出的日期进行格式化输出◆ 使用Lambda表达式迭代升级主窗口中“帮助”菜单的事件处理◆ 使用Lambda表达式实现查找指定的匹配信息并显示	增加注解和格式化以及Lambda优化和查询



功能要求

系统基于C/S模式，包括客户端和服务端应用程序

- 1. 用户登录和注册功能：用户验证口令通过后登录系统，新用户进行注册，并将注册信息保存数据库
- 2. 日志、物流数据的采集功能：对日志和物流数据进行采集，并保存到Mysql数据库；
- 3. 日志、物流数据的筛选匹配功能，以日志信息为例：
 - 根据日志的登录、登出状态，对日志进行分类，分别存放到**登录**日志集合（ArrayList<LogRec> logIns）和**登出**日志集合（ArrayList<LogRec> logOuts）中。
 - 在登录日志和登出日志中，根据**用户名**和**IP地址**进行匹配：如果存在相同的用户名和IP地址，则日志信息匹配成功，将匹配的日志数据封装到MatchedLogRec对象，并保存到匹配日志集合（ArrayList<MatchedLogRec> matchLogs）中。
- 4. 日志、物流数据的数据保存功能：在系统主界面中点击“保存数据”按钮时，将匹配的日志数据和物流数据保存到本地文件和数据库中。
- 5. 客户端服务器端交互功能：
 - ✓ 客户端的数据发送功能：在客户端通过Socket技术向服务器端发送匹配的日志数据和物流数据；当数据发送成功后，清空客户端暂时存放数据的集合，然后弹出信息提醒；
 - ✓ 服务器数据查询功能：当点击客户端显示数据功能时，服务器端从数据库中查找符合条件的数据，并发送到客户端；在客户端以表格的形式将数据显示。



实验目标和要求



实验目标

1. 理解与掌握知识点：

文件与IO流：掌握File类，字节流处理和字符流处理，InputStream/OutputStream及其子类，Reader/Writer及其子类；掌握对象串行化的概念和方法

2. 开发控制台程序：

利用文件IO流相关知识，在第一次课搭建系统框架，实现日志和物流数据采集、匹配和显示功能基础上，迭代开发，实现数据挖掘系统中的数据记录功能



实验要求

1. Service包下的LogRecService和TransportService类中，完善日志和物流数据文件存取方法
 - ◆ 日志数据保存和读取方法SaveMatchLog()，readMatchLog()
 - ◆ 物流数据保存和读取方法SaveMatchedTransport()，readMatchedTransport()
2. 在主界面中，调用以上方法，实现数据记录功能，将匹配后的日志和物流数据保存到文件和从文件中读取显示到控制台

★说明：给定的程序有Bug，需要同学们修改，可以改为不追加写入（参考链接：

<http://baijiahao.baidu.com/s?id=1600984799323133994&wfr=spider&for=pc>），或者追加写入（ObjectOutputStream以追加方式写入文件参考以下链接：

<https://blog.csdn.net/u011091632/article/details/23698601>）



实验要求与交付

实验平台上的作业交付清单：

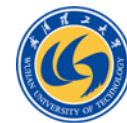
- 本次实验改动的源码
- 本次实验遇到的问题（截图）和解决方法

回答一些问题：

- 介绍下FileOutputStream及其构造方法，说明FileOutputStream(“MatchLogs.txt”, true)和FileOutputStream(“MatchLogs.txt”)区别是什么
- 为什么要进行对象序列化，什么叫对象序列化，对象序列化后写入文件为什么打开是乱码？
- ObjectOutputStream对象输出流如何以追加方式写入文件，至少说明一个方法



本次实验内容



实验内容

1. 做什么？

◆ 实现控制台数据记录功能

```
*****  
* 1、数据采集      2、数据匹配 *  
* 3、数据记录      4、数据显示 *  
* 5、数据发送      0、退出应用 *  
*****
```

请输入菜单项（0~5）：

3

请输入记录数据类型：1.日志 2.物流

1

2. 怎么做？



◆ 完善数据存取方法，将匹配后的日志和物流数据保存到文件和从文件中读取

◆ 将方法嵌入到控制台主框架中实现

3. 需要掌握什么？

◆ 输入输出流
如FileOutputStream
◆ 对象序列化
ObjectOutputStream
◆ File类

◆ 熟悉框架
◆ 调试、单步调试
◆ 类与方法调用



实验内容-系统框架介绍

q_dms_chapter01

src

- com.qst.dms.dos → 控制台主程序
- com.qst.dms.entity → 实体类，日志和物流数据以及匹配后的数据类
- com.qst.dms.exception → 自定义的异常处理类
- com.qst.dms.gather → 集合类，匹配筛选的数据集合
- com.qst.dms.service → 业务类，封装关于日志和物流数据的操作如采集、输出、保存等

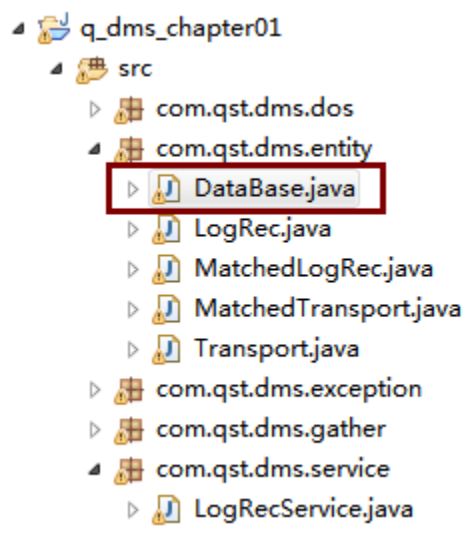
JRE System Library [JavaSE-1.8]



实验内容-对象序列化

1. 将Entity包下的所有实体类升级为可序列化的类，便于后续在文件中保存或在网络中传输

以DataBase.java为例, 类名后增加” implements Serializable”, 即实现Serializable可序列化接口, 则该类可序列化



```
3  
4 import java.io.Serializable;  
5 import java.util.Date;  
6  
7 //数据基础类  
8  
9 public class DataBase implements Serializable {  
10     // ID标识  
11     private int id;  
12     // 时间  
13     private Date time;  
14     // 地点
```



实验内容-日志数据存取功能

2. 匹配后的日志信息保存和读取功能

在日志业务类**LogRecService**类中，新增**saveMatchLog**和**readMatchLog**方法将匹配的日志信息序列化后保存到文本文件，以及从文本文件中读取匹配信息

```
// 匹配日志信息保存，参数是集合，也可自己定义新方法
public void saveMatchLog(ArrayList<MatchedLogRec> matchLogs) {
    // 创建一个ObjectOutputStream对象输出流，并连接文件输出流
    // 以可追加的方式创建文件输出流，数据保存到MatchLogs.txt文件中
    //需补充该方法
    //...
}

// 读匹配日志信息读取，也可自己定义新方法
public ArrayList<MatchedLogRec> readMatchLog() {
    ArrayList<MatchedLogRec> matchLogs = new ArrayList<>();
    // 创建一个ObjectInputStream对象输入流，并连接文件输入流，读MatchLogs.txt文件中
    //需补充该方法
    //...
    return matchLogs;
}
```



实验内容-物流数据存取功能

3.1 匹配后的物流信息保存功能

物流业务类TransportService中，新增saveMatchedTransport方法将匹配后的物流对象序列化后保存到文本文件。参考的saveMatchedTransport()方法如下：**注意，该代码不是正确的，需要修改！**

```
// 匹配物流信息保存，参数是集合
public void saveMatchedTransport(ArrayList<MatchedTransport> matchTrans) {
    // 创建一个ObjectOutputStream对象输出流，并连接文件输出流
    // 以可追加的方式创建文件输出流，数据保存到MatchedTransports.txt文件中
    try (ObjectOutputStream obs = new ObjectOutputStream(
        new FileOutputStream("MatchedTransports.txt", true))) {
        // 循环保存对象数据
        for (MatchedTransport e : matchTrans) {
            if (e != null) {
                // 把对象写入到文件中
                obs.writeObject(e);
                obs.flush();
            }
        }
        // 文件末尾保存一个null对象，代表文件结束
        obs.writeObject(null);
        obs.flush();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```




实验内容-物流数据存取功能

3.2 匹配后的物流数据读取功能

readMatchedTransport方法从已序列化的匹配物流信息文件中读取到（内存/程序中）

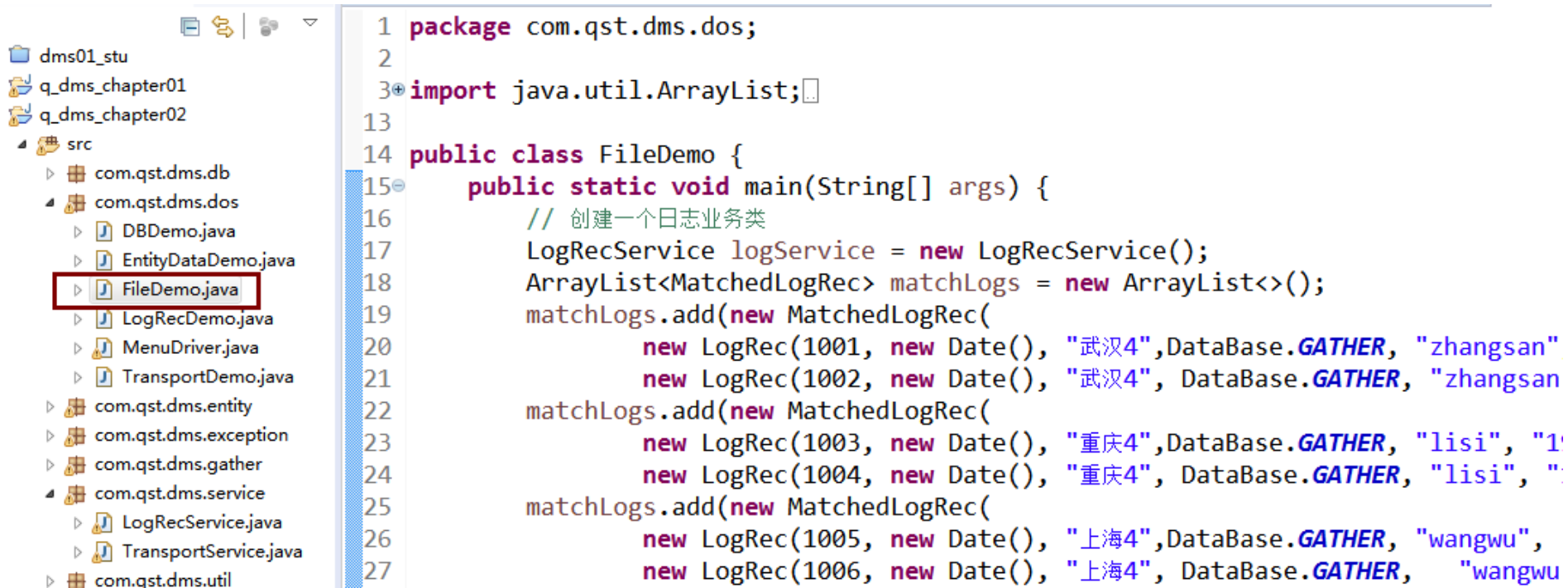
```
// 读匹配物流信息保存, 参数是集合
public ArrayList<MatchedTransport> readMatchedTransport() {
    ArrayList<MatchedTransport> matchTrans = new ArrayList<>();
    // 创建一个ObjectInputStream对象输入流, 并连接文件输入流, 读MatchedTransports.txt文件中
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
        "MatchedTransports.txt"))) {
        MatchedTransport matchTran;
        /*// 循环读文件中的对象
        while ((matchTran = (MatchedTransport) ois.readObject()) != null) {
            // 将对象添加到泛型集合中
            matchTrans.add(matchTran);
        }*/

        /*20191113更新, 解决EOFException问题*/
        while (true) {
            try {
                // 将对象添加到泛型集合中
                matchTran = (MatchedTransport) ois.readObject();
                matchTrans.add(matchTran);
            } catch (EOFException ex) {
                break;
            }
        }
    } catch (Exception ex) {
        //ex.printStackTrace();
    }
    return matchTrans;
}
```



实验内容-测试匹配的日志和物流数据存取

4. 测试代码FileDemo.java, 来测试实现的数据存取功能

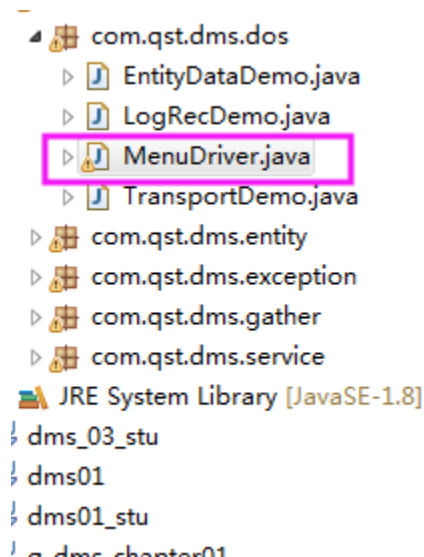


```
1 package com.qst.dms.dos;
2
3 import java.util.ArrayList;
4
5 public class FileDemo {
6     public static void main(String[] args) {
7         // 创建一个日志业务类
8         LogRecService logService = new LogRecService();
9         ArrayList<MatchedLogRec> matchLogs = new ArrayList<>();
10        matchLogs.add(new MatchedLogRec(
11            new LogRec(1001, new Date(), "武汉4", DataBase.GATHER, "zhangsan", "1001", "1001", "1001"),
12            new LogRec(1002, new Date(), "武汉4", DataBase.GATHER, "zhangsan", "1002", "1002", "1002"),
13            new MatchedLogRec(
14                new LogRec(1003, new Date(), "重庆4", DataBase.GATHER, "lisi", "1003", "1003", "1003"),
15                new LogRec(1004, new Date(), "重庆4", DataBase.GATHER, "lisi", "1004", "1004", "1004"),
16                new MatchedLogRec(
17                    new LogRec(1005, new Date(), "上海4", DataBase.GATHER, "wangwu", "1005", "1005", "1005"),
18                    new LogRec(1006, new Date(), "上海4", DataBase.GATHER, "wangwu", "1006", "1006", "1006")
19                )
20            )
21        );
22    }
23 }
```



实验内容主界面调用

5.1 在控制台主界面中完成“数据记录”功能



```
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98

// 物流数据过滤
transAn.doFilter();
// 物流数据分析
matchedTrans = transAn.matchData();
System.out.println("物流数据过滤匹配完成！")
}
}
break;
case 3:
//System.out.println("数据记录 中...");

//在主控制台中完善数据记录功能，需补充
//...

break;
```



实验内容主界面调用

5.2 在控制台主界面中修改数据显示功能，将原来直接从匹配集合中读取修改为先从文件中读取，再调用showMatchLog()方法，显示

```

    case 4: {
        System.out.println("显示匹配的数据：");
        ArrayList<MatchedLogRec> matchLogs = new ArrayList<>();
        matchLogs = logService.readMatchLog();
        logService.showMatchLog(matchLogs);
        /*if (matchedLogs == null || matchedLogs.size() == 0) {
            System.out.println("匹配的日志记录是0条！");
        } else {
            // 输出匹配的日志信息
            logService.showMatchLog(matchedLogs);
        }
        if (matchedTrans == null || matchedTrans.size() == 0) {
            System.out.println("匹配的物流记录是0条！");
        } else {
            // 输出匹配的物流信息
            tranService.showMatchTransport(matchedTrans);
        }*/
    }
}
```



武汉理工大学
WUHAN UNIVERSITY OF TECHNOLOGY

知识点介绍



输入输出

- 输入输出—Input/Output (I/O)

指的是语言提供什么样的手段能够使编程人员操作各种I/O设备

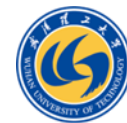
- ✓ Unix-各种I/O设备当成文件，统一操作接口
- ✓ C++-流,所有文件读写简化为流模型
- ✓ **Java**-以系统API类库的方式提供编程人员操作I/O
- ✓ **java.io**





流(Stream)

- 流（Stream）的概念：是一组有序的数据序列，源于UNIX，流是单向的
- 流的分类：
 - 根据流中的基本数据单元，分为字节流和字符流
在Java中，字节流读写以字节byte为单位的流（8位）；而字符流用于读写Unicode字符组成的文本流（16位）。
 - 根据流的角色，分为节点流和处理流
节点/介质流：用于从/向一个特定的IO设备（磁盘/网络）读写数据的流
处理流：对一个已存在的流进行连接和封装，通过封装后的流实现数据读写
 - 根据流的方向可以分输入流和输出流



基本I/O模式输入输出流

★应用程序创建数据流

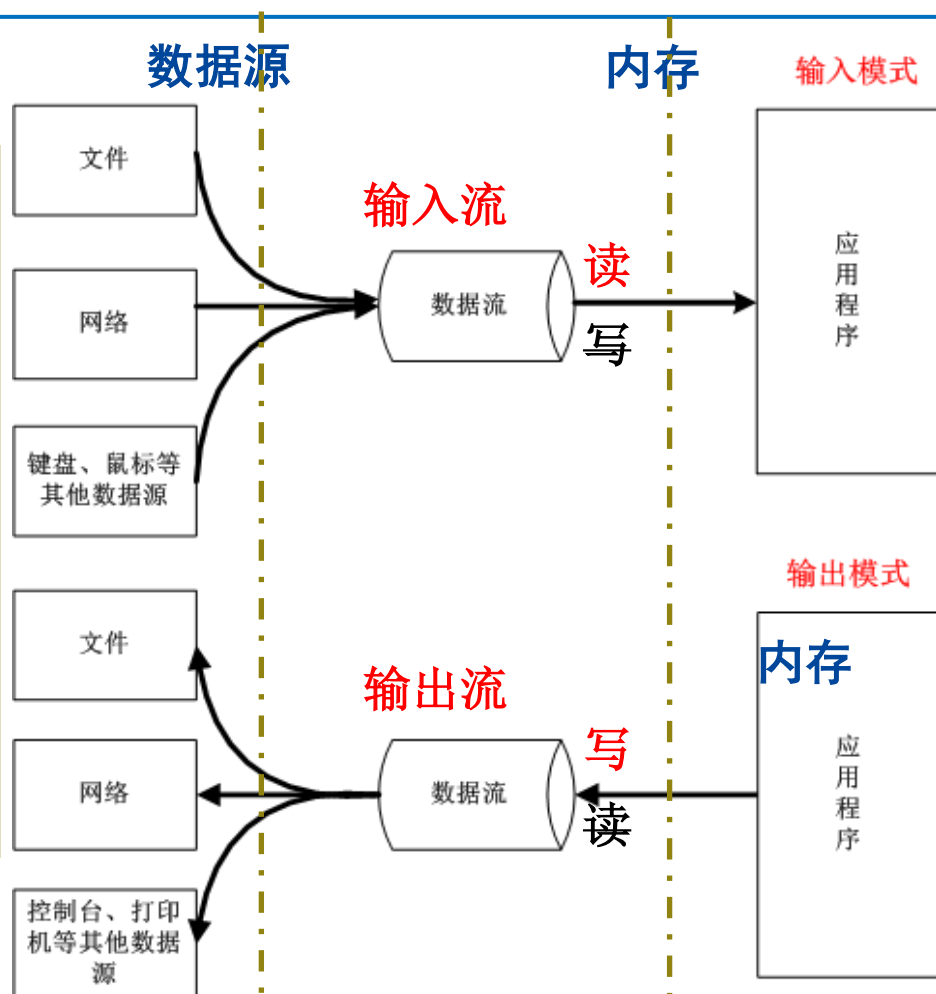
读写数据三步骤：

1. 打开一个流
2. 读/写数据
3. 关闭流

可以从输入流中读取信息，但不能写它；
只能往输出流写入信息，但不能读它。

输入流：应用程序创建某个信息来源的数据流，并打开该数据流获取指定来源的数据

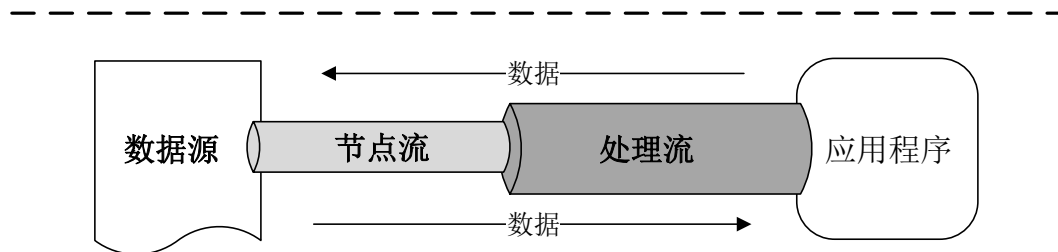
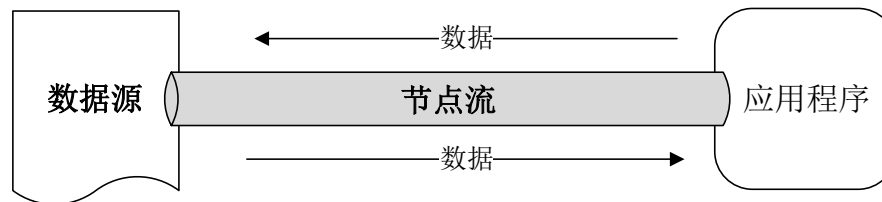
输出流：应用程序创建某个输出对象的数据流，并打开该数据流对象即输出目标。
将数据写入数据流，数据流对象会将数据存储到相应的目标





流(Stream)

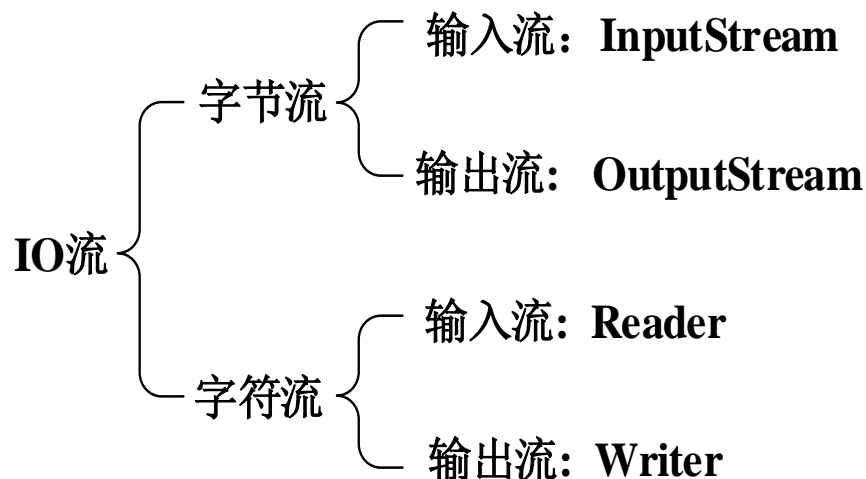
- 使用处理流进行输入/输出时，程序不会直接连接到实际的数据源，而是对节点流进行包装。
- 使用处理流来包装不同的节点流，消除了不同节点流实现的差异，提供了更便利的方法来完成输入/输出功能





流的体系结构

- Java的IO流由4个顶层抽象基类派生

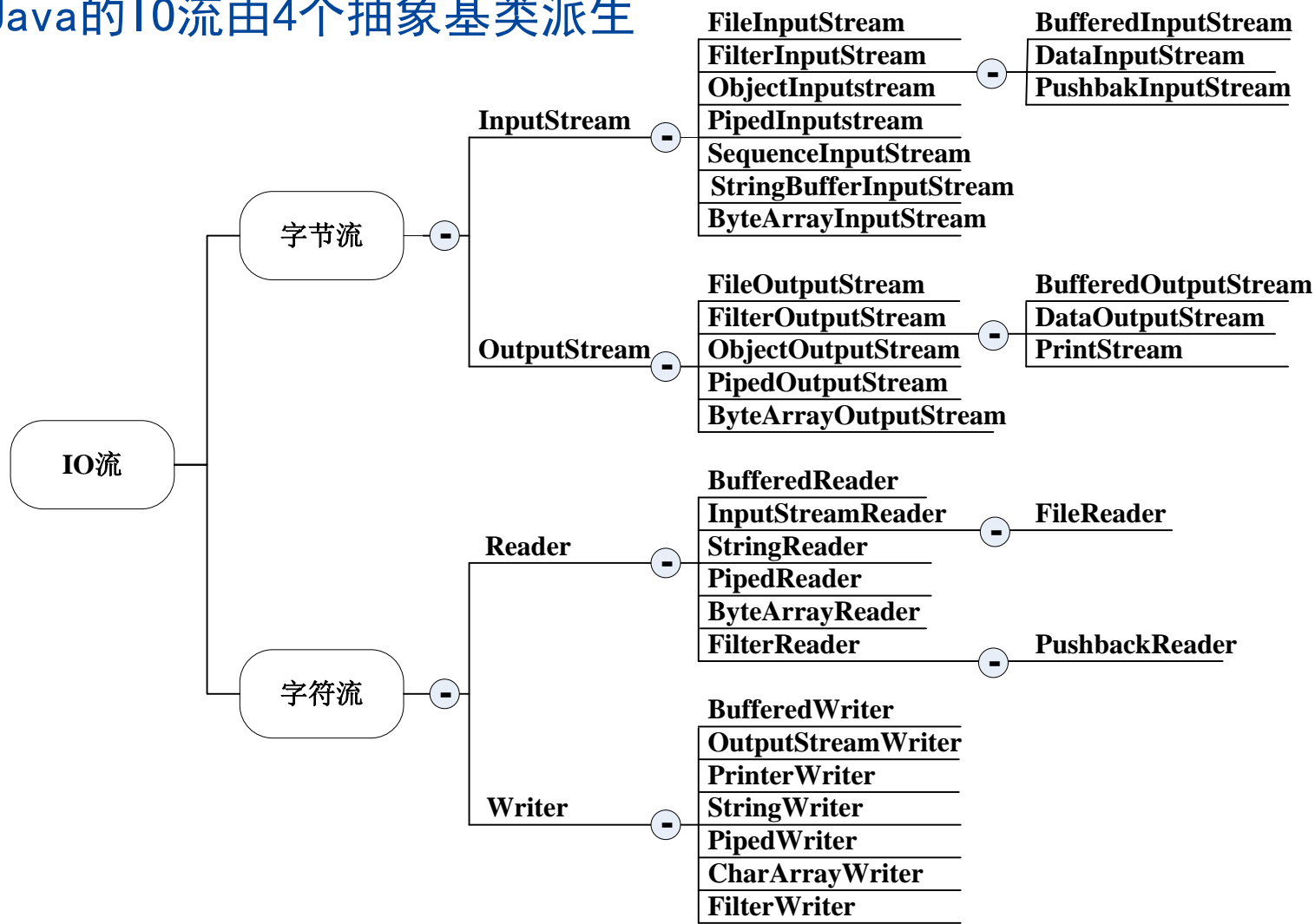


- 针对某个数据源或输出目标的流，由它们的子类具体实现



流的体系结构

■ Java的IO流由4个抽象基类派生





InputStream与OutputStream

- **FileInputStream**和**FileOutputStream**节点流，用于从文件中读取或往文件中写入字节流。如果在构造**FileOutputStream**时，文件已经存在，则覆盖这个文件。
- **BufferedInputStream**和**BufferedOutputStream**过滤流，**需要使用已经存在的节点流来构造**，提供带缓冲的读写，提高了读写的效率。
- **DataInputStream**和**DataOutputStream**过滤流，**需要使用已经存在的节点流来构造**，提供了读写Java中的基本数据类型的功能。



InputStream

■ 三个基本的读方法

`abstract int read()` : 读取一个字节数据, 并返回读到的数据, 如果返回-1, 表示读到了输入流的末尾。

`int read(byte[] b)` : 将数据读入一个字节数组, 同时返回实际读取的字节数。

`int read(byte[] b, int off, int len)` : `off`指定在数组**b**中存放数据的起始偏移位置; `len`指定读取的最大字节数。

• 其它方法

`long skip(long n)` : 在输入流中跳过**n**个字节, 并返回实际跳过的字节数。

`void close()` : 关闭输入流, 释放和这个流相关的系统资源。



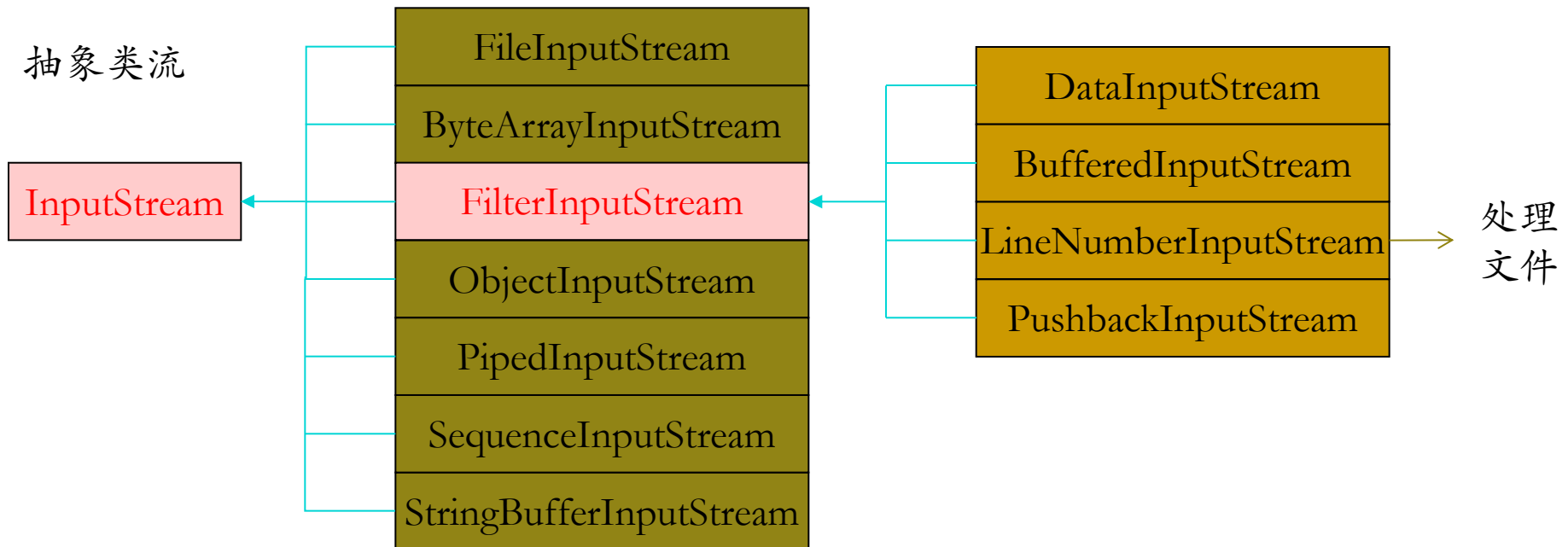
InputStream

■ java.io包中InputStream的类层次

介质流

过滤器流

抽象类流





OutputStream

■ 三个基本的写方法

`abstract void write(int b)` : 往输出流中写入一个字节。

`void write(byte[] b)` : 往输出流中写入数组**b**中的所有字节。

`void write(byte[] b, int off, int len)` : 往输出流中写入数组**b**中从偏移量**off**开始的**len**个字节的数据。

• 其它方法

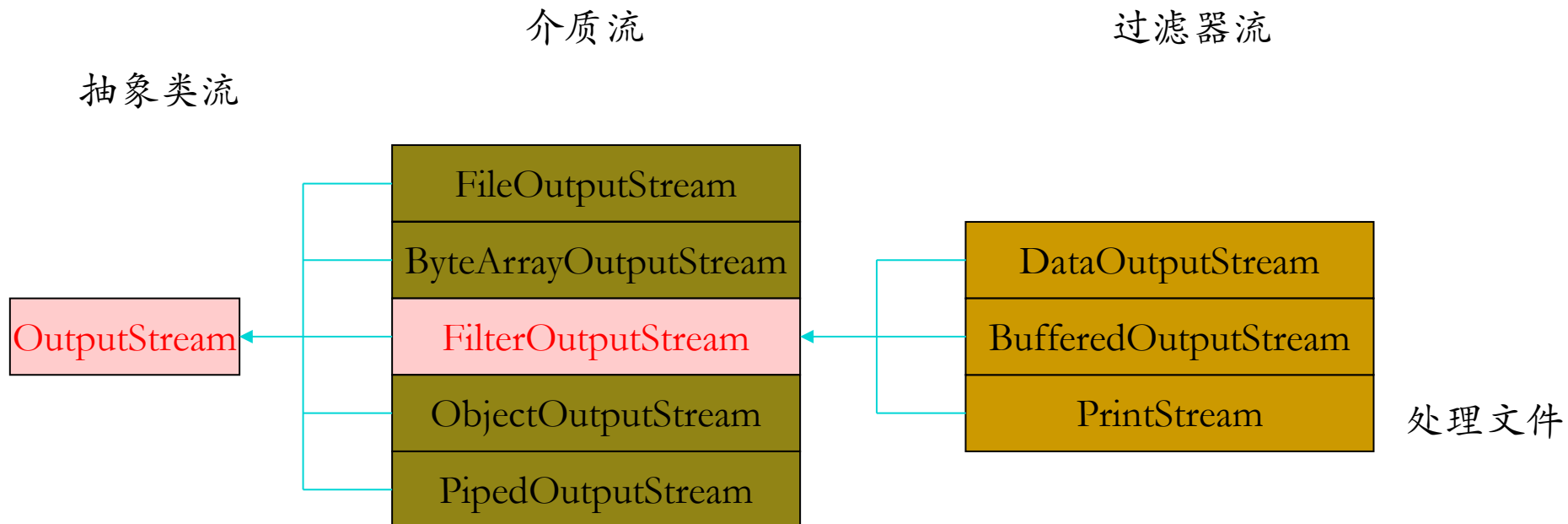
`void flush()` : 刷新输出流，强制缓冲区中的输出字节被写出。

`void close()` : 关闭输出流，释放和这个流相关的系统资源。



OutputStream

■ java.io包中的OutputStream的类层次





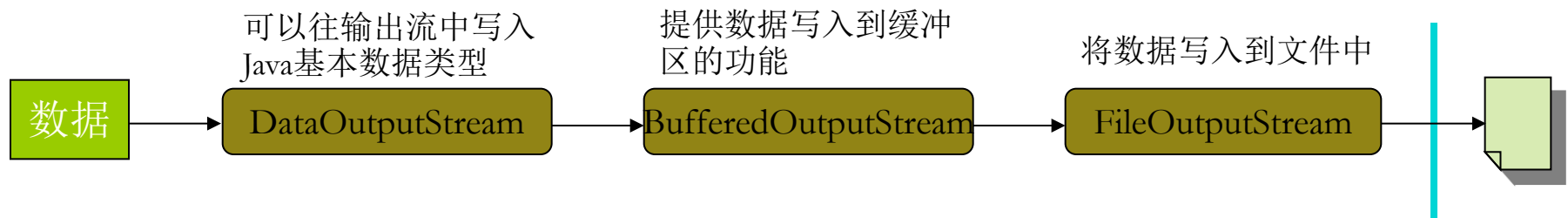
流的链接

Input Stream Chain



```
DataInputStream in=new DataInputStream(new BufferedInputStream(new FileInputStream("yyc.bin")));
```

Output Stream Chain





FileInputStream/FileOutputStream类

文本文件也可用字节流处理，字节流文件的输入输出



FileOutputStream的构造方法：

- 使用File对象打开本地文件，从文件读取数据

```
File file=new File("d://Matchlog.txt");
```

```
FileOutputStream fos=new FileOutputStream(file);//try catch
```

- 不使用File对象，直接传入文件路径

```
FileOutputStream fos=new FileOutputStream("d://Matchlog.txt");//try catch
```

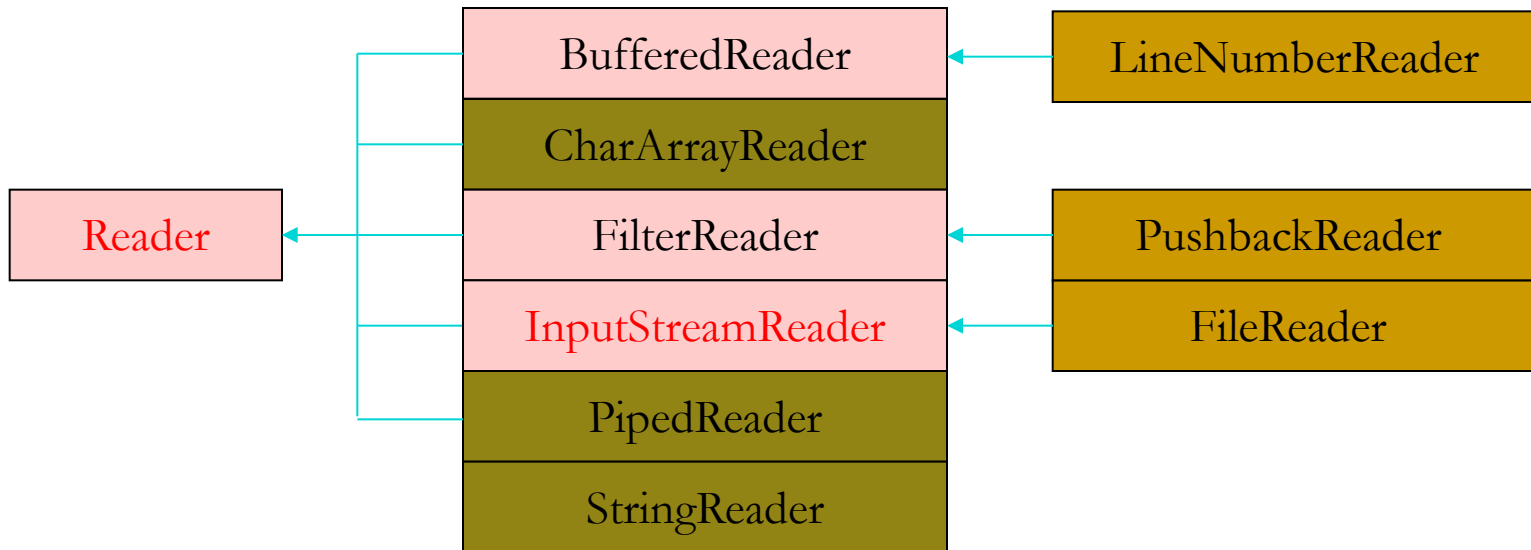
- 打开文件，在文件尾部追加写入数据

```
try (ObjectOutputStream obs = new ObjectOutputStream(new FileOutputStream("MatchLogs.txt,true")))
```

```
catch (Exception ex)
```

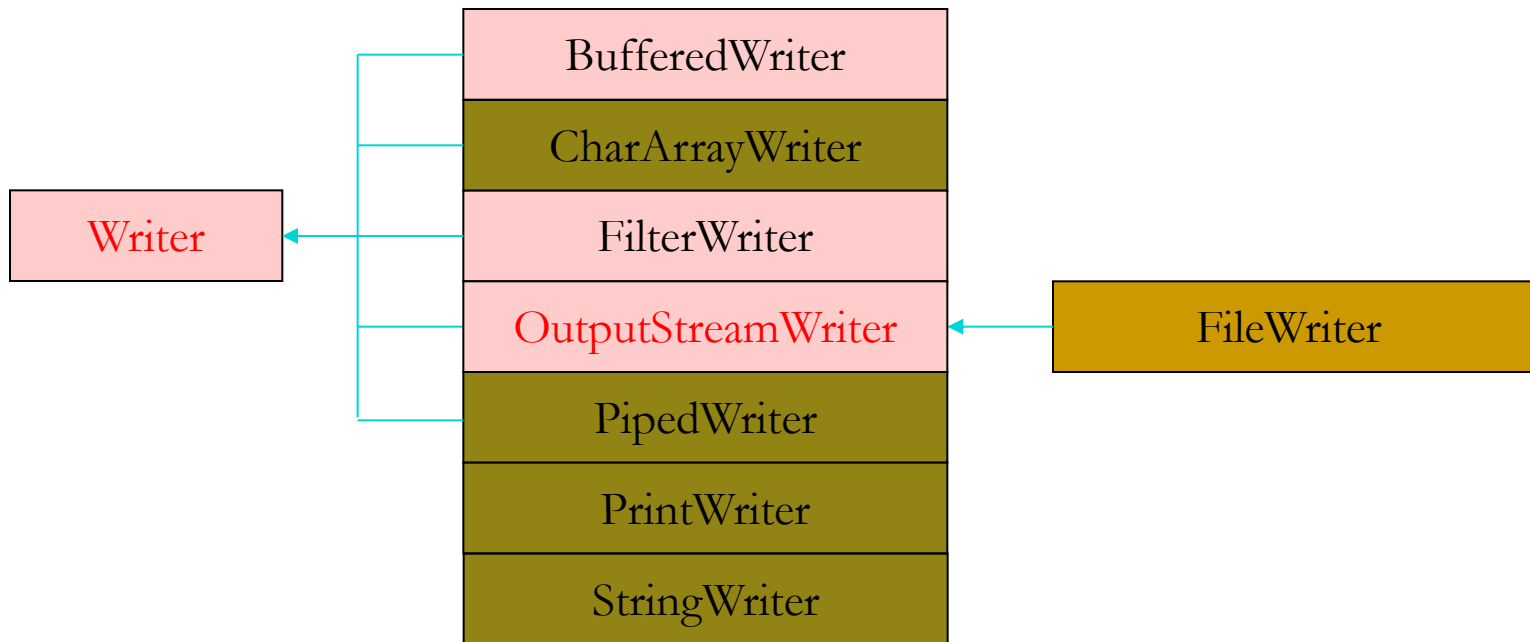


Reader





Writer





I/O处理方式

■ FILE文件类

■ Object对象序列化

序列化是一种用来处理对象流的机制

所谓对象流：就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。

序列化是为了解决在对对象流进行读写操作时所引发的问题



I/O处理方式之对象序列化Serialization

■ 为什么要对象序列化？

对象的序列化

■ 程序运行时保存数据。

1. 对于基本的数据类型，程序在下次启动时，可以读取文件中的数据并初始化程序。
2. 对于复杂和大型的对象类型，如何永久保存数据？
 - ✓ 将对象中的不同属性分解为基本数据类型保存到文件中
 - ✓ 当程序再次运行时，从文件中读取与对象有关的数据，使用这些数据为对象的每个属性初始化



I/O处理方式之对象序列化

- 序列化实现：将需要被序列化的类实现Serializable接口（标记接口）
- Serializable接口中没有任何的方法。当一个类声明要实现Serializable接口（ implements Serializable ）时，只是表明该类参加串行化协议，标注该对象是可被序列化的。
- 序列化和反序列化一个对象之前，该对象的类必须实现了 java.io.Serializable接口。

```
7 //匹配日志记录，“登录登出对” 类型
8
9 public class MatchedLogRec implements Serializable {
10
```



I/O处理方式之对象序列化Serialization

- `ObjectOutputStream`: 对象的序列化流, 作用: 把对象以流的方式写入到文件中保存, 实现对象的持久存储。把对象写入到输出流中, 而不需要每次写入一个字节。

```
1 | ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream("object.data"));
2 |
3 | MyClass object = new MyClass(); output.writeObject(object); //etc.
4 |
5 | output.close();
```

➡ 将指定的对象写入
ObjectOutputStream

- `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, 以输入流方式读取Java对象, 而不需要每次读取一个字节。



I/O处理方式之对象序列化Serialization

■ ObjectOutputStream对象序列化流构造方法

```
public ObjectOutputStream(OutputStream out) throws IOException {  
    verifySubclass();  
    bout = new BlockDataOutputStream(out);  
    handles = new HandleTable(10, (float) 3.00);  
    subs = new ReplaceTable(10, (float) 3.00);  
    enableOverride = false;  
    writeStreamHeader();  
    bout.setBlockDataMode(true);  
    if (extendedDebugInfo) {  
        debugInfoStack = new DebugTraceInfoStack();  
    } else {  
        debugInfoStack = null;  
    }  
}
```

模数

```
protected void writeStreamHeader() throws IOException {  
    bout.writeShort(STREAM_MAGIC);  
    bout.writeShort(STREAM_VERSION);  
}
```

版本

`ObjectOutputStream(OutputStream)`构造方法创建一个`ObjectOutputStream`，此对象流写数据到传入的`OutputStream`流中。
构造方法首先立即写序列化头部到`OutputStream`中



I/O处理方式之对象序列化Serialization

■ ObjectInputStream对象反序列化流构造方法

```
public ObjectInputStream(InputStream in) throws IOException {  
    verifySubclass();  
    bin = new BlockDataInputStream(in);  
    handles = new HandleTable(10);  
    vlist = new ValidationList();  
    enableOverride = false;  
    readStreamHeader();  
    bin.setBlockDataMode(true);  
}
```

readStreamHeader方法用来给子类读取并验证流的头部。头部有两个字段： magic 和 version，与输出流对应。



I/O处理方式之对象序列化Serialization

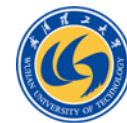
【问题描述】 java.io.StreamCorruptedException: invalid type code: AC

ObjectOutputStream以追加的方式向文件中序列化对象时，每次只想向文件末尾追加对象，而不是覆盖，使用FileOutputStream（文件名，**true**），用writeObject()写入对象，在读取数据的时候第一次会正常读取，不会报错。当关闭流后再打开，追加写入第二个对象，当用ObjectInputStream反序列化读取对象时，就会报出

java.io.StreamCorruptedException: invalid type code: AC的错误。

【原因】：新建一个ObjectOutputStream对象每次都会先写入两个Short类型的即4字节的数据StreamHeader头部数据（构造方法里可以看到调用了一个writeStreamHeader()方法）在反序列化的时候每个ObjectInputStream对象只会读取一个header，那么当遇到第二个的时候就会报错，导致出现异常。



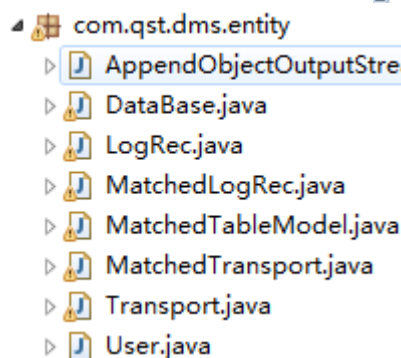


I/O处理方式之对象序列化Serialization

【解决方法】

<https://blog.csdn.net/u011091632/article/details/23698601>

重写ObjectOutputStream中的writeStreamHeader方法，自定义AppendObjectOutputStream写入对象的类



```
public class AppendObjectOutputStream extends ObjectOutputStream {  
    public static File file = null;  
    public AppendObjectOutputStream(File file) throws IOException {  
        super(new FileOutputStream(file, true));  
    }  
    public void writeStreamHeader() throws IOException {  
        if (file != null) {  
            if (file.length() == 0) {  
                super.writeStreamHeader();  
            } else {  
                //System.out.println("文件不为空"); //当文件不为空时，空实现  
            }  
        } else {  
            //System.out.println("文件不存在");  
            super.writeStreamHeader();  
        }  
    }  
}
```



I/O处理方式之对象序列化Serialization

【解决方法】

<https://blog.csdn.net/u011091632/article/details/23698601>

在业务类中新增saveAndAppendMatchLog和saveAndAppandMatchedTransport方法

```
public void saveAndAppandMatchedTransport(ArrayList<MatchedTransport> matchTrans) {  
    AppendObjectOutputStream aoss = null;  
    File file = new File("MatchedTransports.txt");  
    try {  
        AppendObjectOutputStream.file = file; //这句话的作用是什么？把这句话删除看会有什么情况  
        aoss = new AppendObjectOutputStream(file);  
        // 循环保存对象数据  
        for (MatchedTransport e : matchTrans) {  
            if (e != null) {  
                // 把对象写入到文件中  
                aoss.writeObject(e);  
                aoss.flush();  
            }  
        }  
    } catch (Exception ex) {  
    } finally {  
        if(aoss!=null) {  
            try {aoss.close();} catch (IOException e) {e.printStackTrace();}  
        }  
    }  
}
```



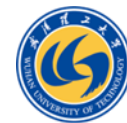
I/O处理方式之对象序列化Serialization

【解决方法】

在FileDemo脚本中进行测试

- com.qst.dms.dos
 - DBDemo.java
 - EntityDataDemo.java
 - FileDemo.java
 - LogRecDemo.java
 - MenuDriver.java
 - TransportDemo.java
- com.qst.dms.entity
 - AppendObjectOutputStre
 - DataBase.java
 - LogRec.java
 - MatchedLogRec.java

```
// 创建一个物流业务类
TransportService tranService = new TransportService();
ArrayList<MatchedTransport> matchTrans = new ArrayList<>();
matchTrans.add(new MatchedTransport(
    new Transport(2001, new Date(), "guangzhou55", DataBase.GATHER, "1"),
    new Transport(2002, new Date(), "guangzhou55", DataBase.GATHER, "1"),
    new Transport(2003, new Date(), "guangzhou55", DataBase.GATHER, "1")
);
matchTrans.add(new MatchedTransport(
    new Transport(2004, new Date(), "guangzhou22", DataBase.GATHER, "1"),
    new Transport(2005, new Date(), "guangzhou22", DataBase.GATHER, "1"),
    new Transport(2006, new Date(), "guangzhou22", DataBase.GATHER, "1")
);
//保存匹配的物流信息到文件中
//tranService.saveMatchedTransport(matchTrans);
tranService.saveAndAppandMatchedTransport(matchTrans);
//保存匹配的物流信息到数据库中
//tranService.saveMatchTransportToDB(matchTrans);
//从文件中读取匹配的物流信息
ArrayList<MatchedTransport> list2 = tranService.readMatchedTransport();
tranService.showMatchTransport(list2);
}
```



I/O处理方式之对象序列化Serialization

【问题描述】 java.io. EOFException

java.io.EOFException

```
at java.io.ObjectInputStream$BlockDataInputStream.peekByte(ObjectInputStream.java:2954)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1535)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:428)
at com.qst.dms.service.LogRecService.readMatchLog(LogRecService.java:118)
at com.qst.dms.dos.FileDemo.main(FileDemo.java:33)
```

【原因】 ObjectInputStream无论是用readObject()读对象还是取int等java基本数据类型，在判断结束时，不是-1和null

【解决办法】

➤ 容器法：将若干个对象装入如ArrayList类似的容器中，然后将容器这一个对象写入，读取时也读取一个容器对象

➤ 非容器法：
EOFException
判断结束
代码结构如右图：

```
public ArrayList<MatchedLogRec> readMatchLog() {
    ArrayList<MatchedLogRec> matchLogs = new ArrayList<>();
    // 创建一个ObjectInputStream对象输入流，并连接文件输入流，读MatchLogs.txt文件中
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
        "MatchLogs.txt"))) {
        MatchedLogRec matchLog;
        /* // 循环读文件中的对象
        while ((matchLog = (MatchedLogRec) ois.readObject()) != null) {
            // 将对象添加到泛型集合中
            matchLogs.add(matchLog);
        }
        */ catch (Exception ex) {
            //ex.printStackTrace();
        }
    }

    //20191113
    while (true) {
        try {
            // 将对象添加到泛型集合中
            matchLog = (MatchedLogRec) ois.readObject();
            matchLogs.add(matchLog);
        } catch (EOFException ex) {
            break;
        }
    }
}
```



I/O处理方式之对象序列化Serialization

【总结】

- ◆ 对象流不同于普通的字节流，当对象流中没有数据时，程序却尝试读取数据，会报EOFException；而字节流就不会出现这种情况，字节流会返回-1
- ◆ ObjectOutputStream写入的数据，在ObjectOutputStream上读取时，应该按照相同的数据类型依次读取，否则数据类型不等会抛出EOFException
- ◆ 最好在实际使用的过程中，先实例化ObjectOutputStream，再实例化ObjectInputStream，这是由这两个类的设计思想所决定的。如此能保证在同一资源的对象流ObjectInputStream能够及时读取到序列化头而不至于阻塞或者引发EOF异常(阻塞对应于Socket IO，EOF异常对应于文件IO)

深入剖析见链接：

https://blog.csdn.net/qq_37206105/article/details/90461717



I/O处理方式之File类

■ File类：

1. 与流无关的类，并不是做文件读写的，来表达一个目录或文件
2. 表达目录Dictionary：列目录上的文件清单，创建或删除目录
3. 表达文件File：查看文件信息或修改文件属性，判断路径等

■ 构造函数：

```
File(String path)、  
File(String path, String FileName)、  
File(File dir, String name)。
```



I/O处理方式之File类

- 通过类File所提供的方法，可以得到文件或目录的描述信息，这主要包括名称、所在路径、可读性、可写性、文件的长度等，还可以生成新的目录、改变文件名、删除文件、列出一个目录中所有的文件等。

方法	含义
String[] list()	列出目录中的文件
Long length()	返回文件的大小
Boolean rename(File dest)	将文件或目录改名
Boolean mkdir()	创建目录
Boolean delete()	删除文件或目录