

课程编号：A0801051060

# 《软件体系结构与设计模式》

## 大作业



姓 名		学 号	
班 级	软件 1901	指 导 教 师	石凯
实 验 名 称	《软件体系结构与设计模式》大作业		
开 设 学 期	2021-2022 春季学期		
开 设 时 间	第 1 周——第 8 周		
报 告 日 期	2022 年 5 月 1 日		
评 定 成 绩		评 定 人	石凯
		评 定 日 期	2022 年 5 月 2 日

东北大学软件学院

## 评价表格

考核标准	得分
(1) 正确理解和掌握实验所涉及的模式结构和原理 (20%) ;	
(2) 按实验要求合理设计类和方法 (20%) ;	
(3) 能设计测试用例，运行结果正确 (20%) ;	
(4) 认真记录实验数据，实验结果分析准确 (10%) ;	
(5) 代码编码规范 (10%) ;	
(6) 实验报告规范 (20%) 。	
合计	

## 大作业

### 1 作业要求

实现一个学生信息管理程序。

其界面原型如下，你需要实现原型中所有的功能。界面仅供参考，你可以在此基础上适当杜撰其他功能性需求。



图 1 作业要求

其他要求：

- 1、图形用户界面首选使用 html 进行表示，当然你也可以选择其他方法实现该图形用户界面。
- 2、全部学生信息需要能持久化到文件中。

3、实现一个网络系统。学生信息存储于服务器端。客户通过 web 端或 html 进行访问。

4、在提升要求基础上，支持多用户同时访问。

## 2 作业展示

本次作业在读取数据库文件的基础上执行的。最后可以输出数据库文件。

student.csv	
1	sid,sname
2	20197581,车开平
3	20195555,张三
4	20196666,李四

图 2 数据库文件

### 2.1 主界面展示

The screenshot shows a web browser window titled "学生信息管理系统" (Student Information Management System) with the URL "localhost:8000". The page displays a table with three rows of student data. Each row contains a red minus button, a text input field, and a text input field. A green plus button is located at the bottom left of the table. The table columns are labeled "学号" (Student ID) and "姓名" (Name). The data is as follows:

学号	姓名
20197581	车开平
20195555	张三
20196666	李四

图 3 首页

### 2.2 添加信息

学号	姓名	撤销
- 20197581	车开平	
- 20195555	张三	
- 20196666	李四	
-		
+ + +		

图 4 添加信息

### 2.3 信息不完整有提醒

学号	姓名	撤销
- 20197581	车开平	
- 20195555	张三	
- 20196666	李四	
- 20192019		空值!
+ + +		

图 5 信息不完整有提醒

### 2.4 信息添加成功，后台会同步更新

学号	姓名	撤销
20197581	车开平	
20195555	张三	
20196666	李四	
20192019	王五	
<b>+</b>		

图 6 添加信息之后

```
GET /mod?id=20192019&name=王五 HTTP/1.1
id:20197581 name:车开平
id:20195555 name:张三
id:20196666 name:李四
id:20192019 name:王五
```

图 7 后台同步展示

## 2.5 信息修改，后台也会同步修改

学号	姓名	撤销
20197581	车开平	
20195555	张三	
20196666	李四	
20192019	王二	
<b>+</b>		

图 8 修改完信息之后

```
GET /mod?id=20192019&name=王二 HTTP/1.1
id:20197581 name:车开平
id:20195555 name:张三
id:20196666 name:李四
id:20192019 name:王二
```

图 9 后台同步展示

## 2.6 信息删除，后台也会同步删除

学号	姓名	撤销
- 20197581	车开平	
- 20195555	张三	
- 20196666	李四	
+		

图 10 前端信息删除

```
GET /del?id=20192019&name=王二 HTTP/1.1  
id:20197581 name:车开平  
id:20195555 name:张三  
id:20196666 name:李四
```

图 11 后台同步删除

## 2.7 撤销操作

学号	姓名	撤销
- 20197581	车开平	
- 20195555	张三	
- 20196666	李四	
- 20192019	王二	
+		

图 12 前端撤销操作

```
GET /undo HTTP/1.1  
id:20197581 name:车开平  
id:20195555 name:张三  
id:20196666 name:李四  
id:20192019 name:王二
```

图 13 后台同步撤销

## 2.8 再一次撤销操作

学号	姓名	撤销
20197581	车开平	
20195555	张三	
20196666	李四	
20192019	王五	
+ (blue)		

图 14 前端再一次撤销

```
GET /undo HTTP/1.1
id:20197581 name:车开平
id:20195555 name:张三
id:20196666 name:李四
id:20192019 name:王五
```

图 15 后台同步撤销操作

## 2.9 多用户数据一致性

--	--

图 16 多用户登陆

### 2.9.1 一个用户增加信息，会同步显示

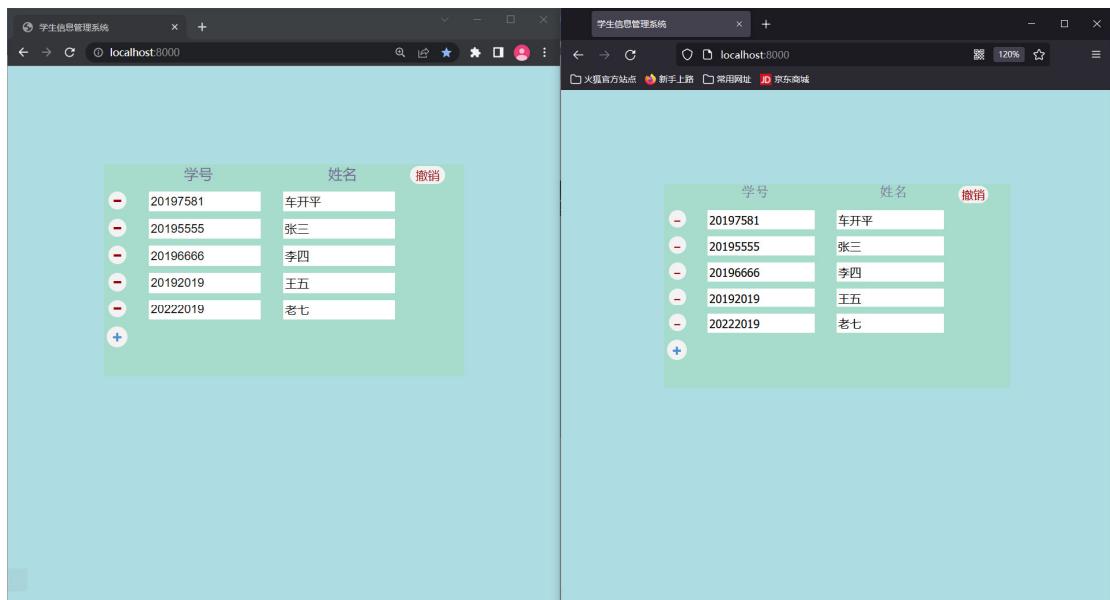


图 17 一个用户增加信息，所有用户同步显示

### 2.9.2 一个用户修改信息，会同步显示

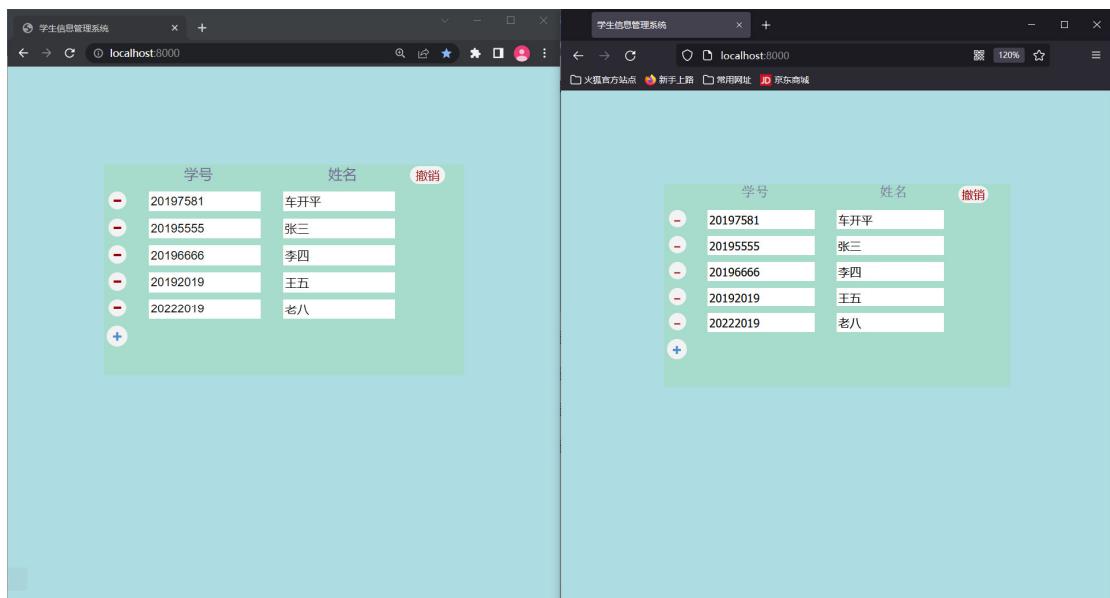


图 18 一个用户修改信息，所有用户同步显示

### 2.9.3 一个用户删除信息，会同步显示

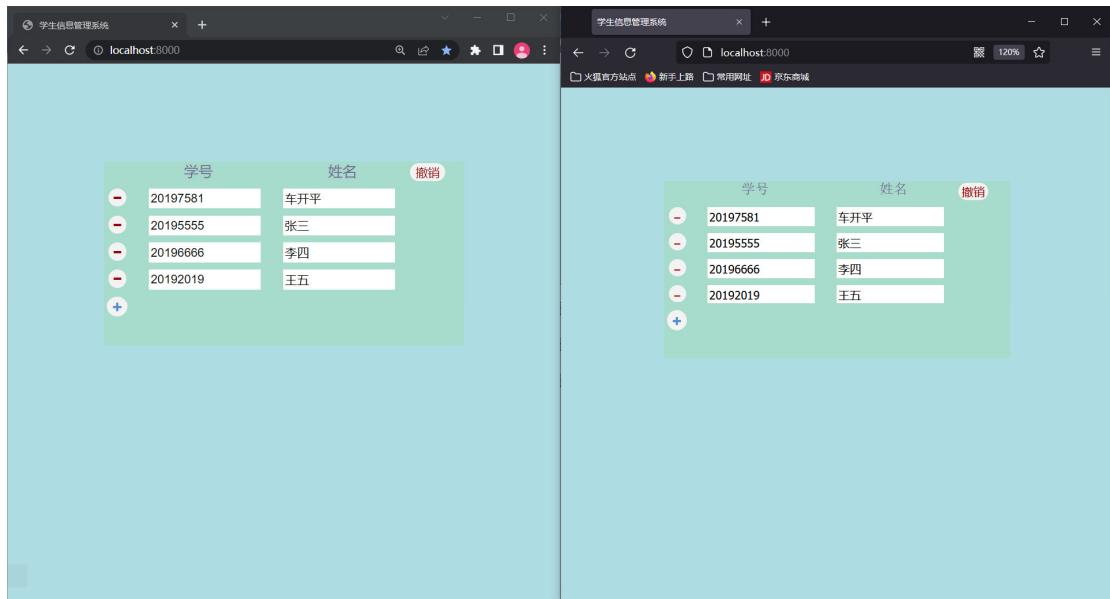


图 19 一个用户删除信息，所有用户同步显示

#### 2.9.4 一个用户撤销操作，会同步显示

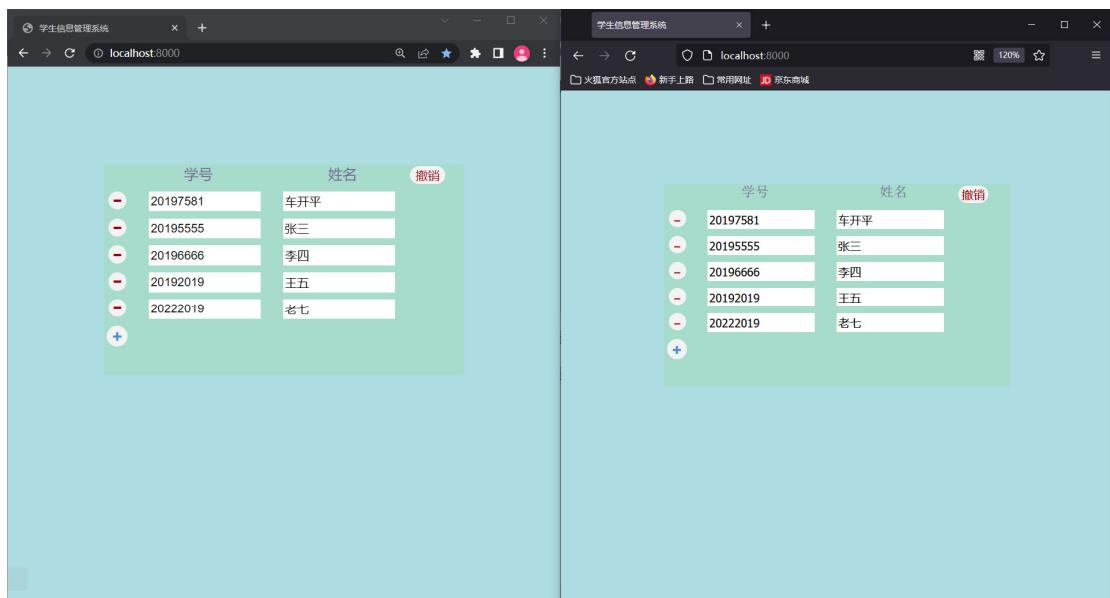


图 20 一个用户撤销操作，所有用户同步显示

#### 2.9.5 一个用户再次撤销，会同步显示

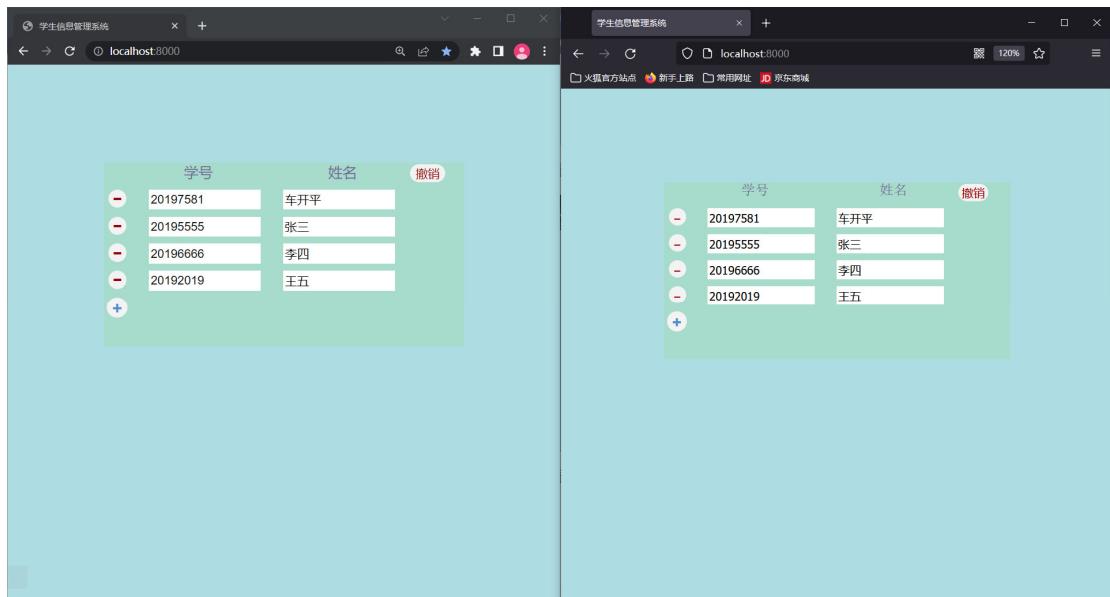


图 21 一个用户再次撤销操作，所有用户同步显示

## 2.10 关闭窗口，提示保存



图 22 用户关闭浏览器

```
-----SAVE-----
filePath:D:\Code\Eclipse\Java\DesignPattern2022\src\com\Ckp\BigProject\
fileName:student2022-04-29-09-12-46
type:csv
finish!
student2022-04-29-09-12-46.csv保存成功
-----SAVE SUCCESSFUL-----
```

图 23 后台服务器持久化数据库文件

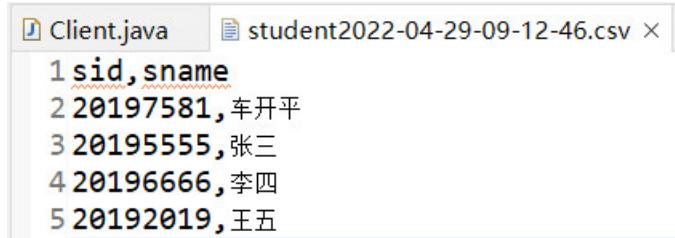


图 24 数据库文件展示

3 总类图

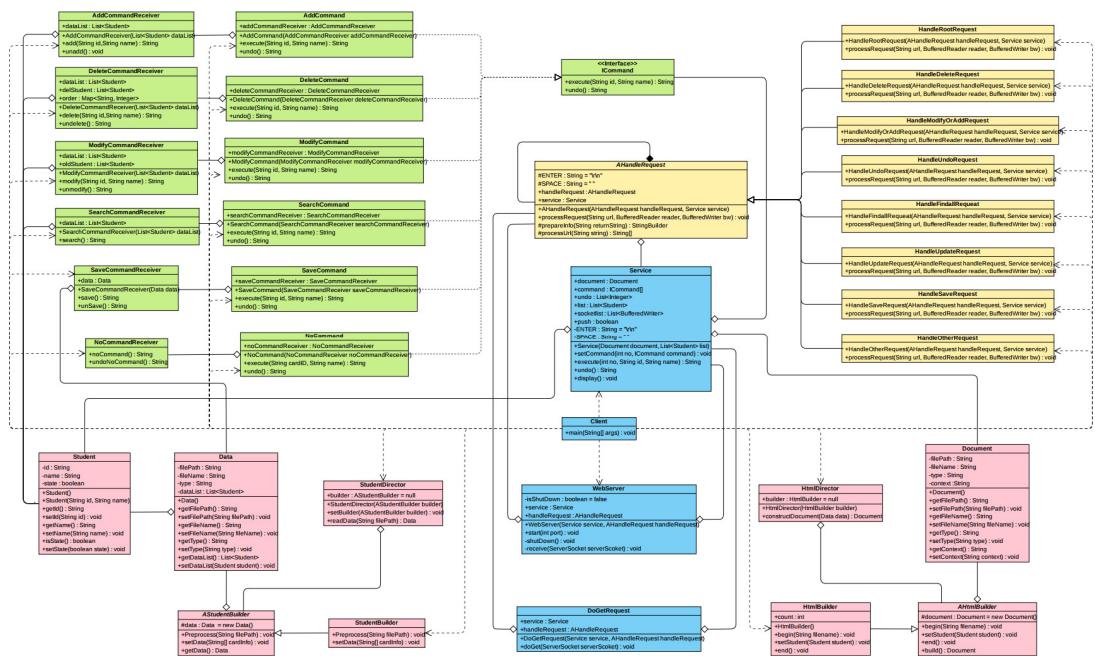


图 25 设计总类图，包含了建造者模式（粉色区域）、命令模式（绿色区域）、职责链模式（黄色区域），清晰的类图在同文件夹的 PDF 中

### 3.1 设计思想

本次作业采用了建造者模式、命令模式、职责链模式这三种设计模式。

因为本作业涉及到文件的输入输出，涉及到的步骤很多。比如读取信息，生成 index.html。为了方便扩展和修改，采用建造者模式，将复杂的步骤逐步分解成一个个小的操作。

又因为涉及到很多 HTTP 协议的请求，为了请求创建接收者对象，然后由这个接收者开始处理，如果不能处理，就传递给下一个处理者。避免了过多的 if else 嵌套，使代码冗余，不易理解。

在职责链的处理器处理的时候，调用的是一个个具体的命令对象，这些命令对象组合了要操作的参数对象，使信息相互独立又紧密相连。同时，能够根据不同的参数执行对象，并且加入到队列中，以便撤销。故采用职责链和命令模式。

### 3.2 建造者部分

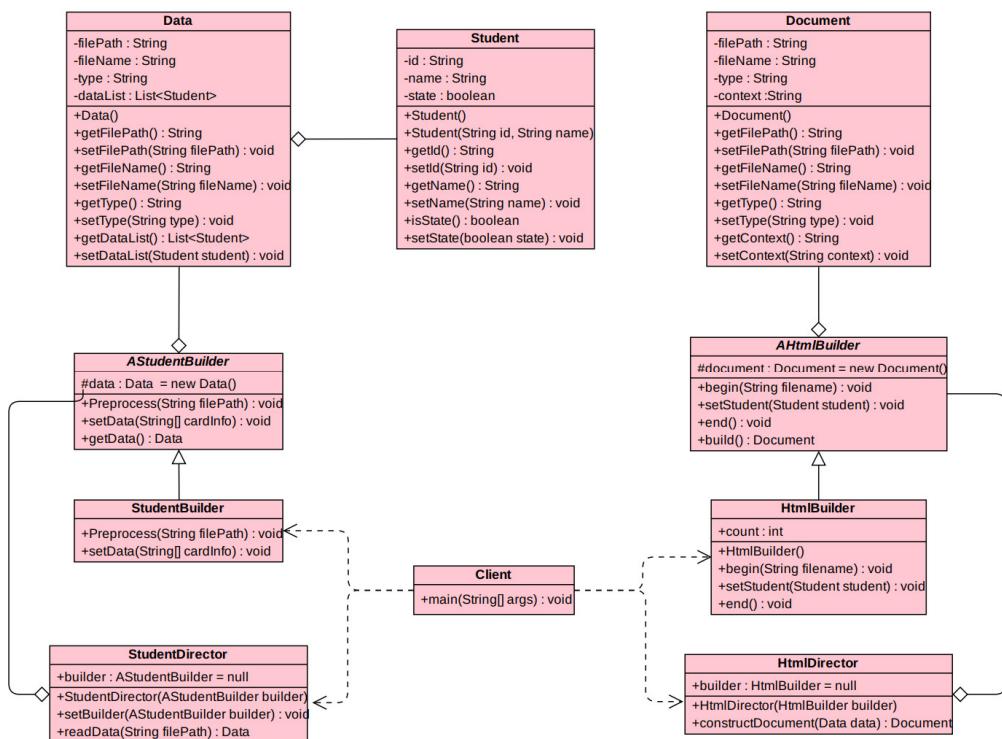


图 26 建造者模式

在这一部分，先读取外部数据库文件，然后生成对应的 Student 对象，最后再生成 index.html 文件，返回到网页请求中去。

#### 3.2.1 Student 类

Student
-id : String
-name : String
-state : boolean
+Student()
+Student(String id, String name)
+getId() : String
+setId(String id) : void
+getName() : String
+setName(String name) : void
+isState() : boolean
+setState(boolean state) : void

图 27 Student 类

定义了一个具体的 Student 对象，包含了 id, name 等信息。提供了所有属性的 get() 和 set() 方法。

### 3. 2. 2 Data 类

Data
-filePath : String
-fileName : String
-type : String
-dataList : List<Student>
+Data()
+getFilePath() : String
+setFilePath(String filePath) : void
+getFileName() : String
+setFileName(String fileName) : void
+getType() : String
+setType(String type) : void
+getDataList() : List<Student>
+setDataList(Student student) : void

图 28 Data 类

这个类定义了一个包含 Student 对象集合的信息类，存储了文件的路径和名称类型等。

### 3. 2. 3 StudentBuilder 抽象建造者类

AStudentBuilder
#data : Data = new Data()
+Preprocess(String filePath) : void
+setData(String[] cardInfo) : void
+getData() : Data

图 29 StudentBuilder 抽象建造者类

这个类组合了一个 Data 信息类，定义了一个抽象建造者，定义了预处理方法、添加 Student 对象方法和 get() Student 对象方法。

### 3. 2. 4 StudentBuilder 具体实现者类

StudentBuilder
+Preprocess(String filePath) : void
+setData(String[] cardInfo) : void

图 30 StudentBuilder 具体实现者类

这个类继承了 StudentBuilder 抽象建造者类，并且实现了具体的方法。

### 3. 2. 5 StudentDirector 指挥者类

StudentDirector
+builder : AStudentBuilder = null
+StudentDirector(AStudentBuilder builder)
+setBuilder(AStudentBuilder builder) : void
+readData(String filePath) : Data

图 31 StudentDirector 指挥者类

这个类组合了 StudentBuilder 类，决定了具体的操作流程。

### 3. 2. 6 Document 文件对象类

Document
-filePath : String
-fileName : String
-type : String
-context :String
+Document()
+getFilePath() : String
+setFilePath(String filePath) : void
+getFileName() : String
+setFileName(String fileName) : void
+getType() : String
+setType(String type) : void
+getContext() : String
+setContext(String context) : void

图 32 Document 文件对象类

这个类定义了一个文件对象，包含了路径、名称、类型和文本内容属性，提供了所有的 get() 和 set() 方法。

### 3. 2. 7 HtmlBuilder 抽象者建造类

AHtmlBuilder
#document : Document = new Document()
+begin(String filename) : void
+setStudent(Student student) : void
+end() : void
+build() : Document

图 33 HtmlBuilder 抽象者建造类

这个类组合了一个 Document 对象，定义了生成 HTML 的过程。

### 3. 2. 8 HtmlBuilder 具体实现类

HtmlBuilder
+count : int
+HtmlBuilder()
+begin(String filename) : void
+setStudent(Student student) : void
+end() : void

图 34 HtmlBuilder 具体实现类

这个类继承了 HtmlBuilder 抽象类，实现了具体的生成方法。

### 3. 2. 9 HtmlDirector 类

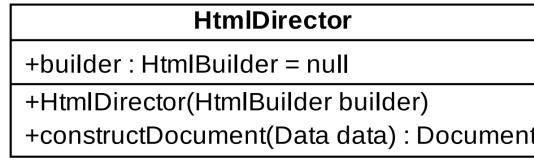


图 35 HtmlDirector 类

这个类组合了 HtmlBuilder 类，决定了具体的操作步骤。

### 3.2.10 Client 调用

```

String filePath =
"D:\\Code\\Eclipse\\Java\\DesignPattern2022\\src\\com\\Ckp\\BigProject\\student
.csv";
StudentBuilder studentBuilder = new StudentBuilder();
StudentDirector director = new StudentDirector(studentBuilder);
Data data = director.readData(filePath);
//
HtmlDirector htmdirector = new HtmlDirector(new HtmlBuilder());
Document document = htmdirector.constructDocument(data);

```

### 3.3 命令模式部分

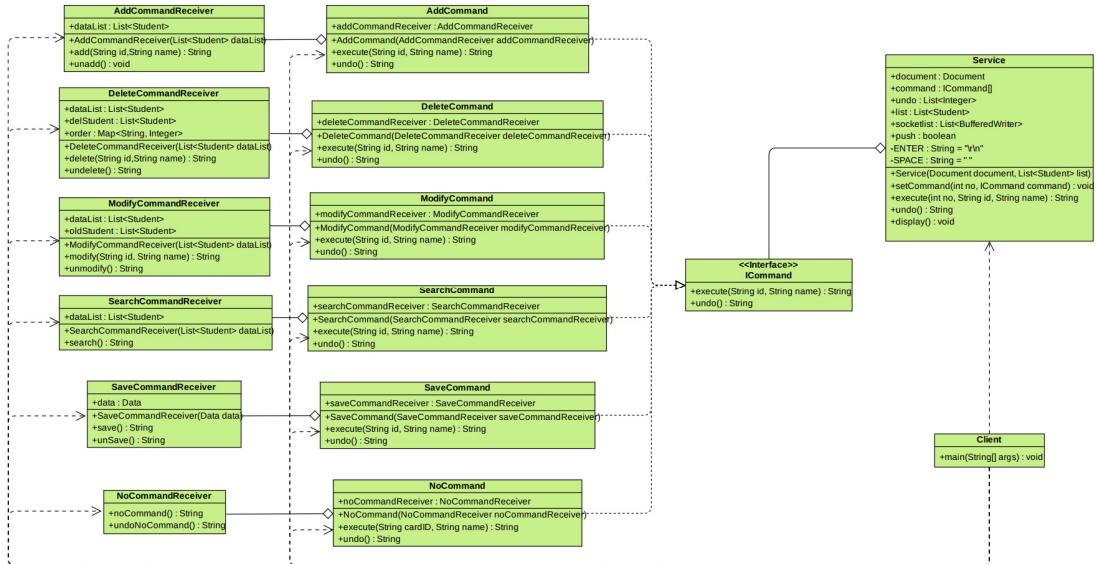


图 36 命令模式类图

在这一部分中，通过定义了不同的命令来实现增删改查等操作。

### 3.3.1 Command 接口

<<Interface>>	
ICommand	
+execute(String id, String name) : String	
+undo() : String	

图 37 Command 接口

这个接口定义了一个命令的操作。

### 3. 3. 1 AddCommand 类

AddCommand	
+addCommandReceiver : AddCommandReceiver	
+AddCommand(AddCommandReceiver addCommandReceiver)	
+execute(String id, String name) : String	
+undo() : String	

图 38 AddCommand 类

这个类组合了一个 AddCommandReceiver 接收者类，通过调用 AddCommandReceiver 接收者类的具体方法，来实现增加操作。

### 3. 3. 2 AddCommandReceiver 接收者类

AddCommandReceiver	
+dataList : List<Student>	
+AddCommandReceiver(List<Student> dataList)	
+add(String id, String name) : String	
+unadd() : void	

图 39 AddCommandReceiver 接收者类

这个类通过组合具体的 Student 信息，来执行增加操作。

### 3. 3. 3 DeleteCommand 类

DeleteCommand	
+deleteCommandReceiver : DeleteCommandReceiver	
+DeleteCommand(DeleteCommandReceiver deleteCommandReceiver)	
+execute(String id, String name) : String	
+undo() : String	

图 40 DeleteCommand 类

这个类组合了一个 DeleteCommandReceiver 接收者类，通过调用 DeleteCommandReceiver 接收者类的具体方法，来实现删除操作。

### 3. 3. 4 DeleteCommandReceiver 接收者类

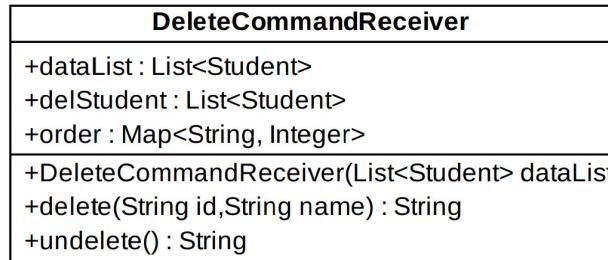


图 41 DeleteCommandReceiver 接收者类

这个类通过组合具体的 Student 信息，来执行删除操作。

### 3. 3. 5 ModifyCommand 类

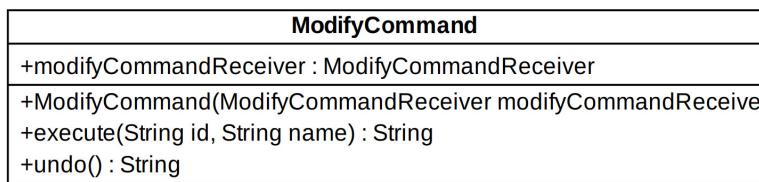


图 42 ModifyCommand 类

这个类组合了一个 ModifyCommandReceiver 接收者类，通过调用 ModifyCommandReceiver 接收者类的具体方法，来实现修改操作。

### 3. 3. 6 ModifyCommandReceiver 接收者类

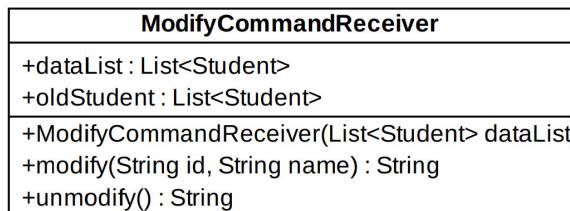


图 43 ModifyCommandReceiver 接收者类

这个类通过组合具体的 Student 信息，来执行修改操作。

### 3. 3. 7 SearchCommand 类

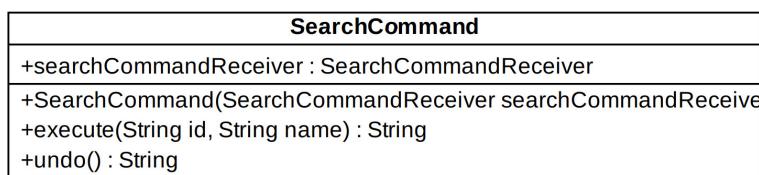


图 44 SearchCommand 类

这个类组合了一个 SearchCommandReceiver 接收者类，通过调用 SearchCommandReceiver 接收者类的具体方法，来实现查找操作。

### 3.3.8 SearchCommandReceiver 接收者类

SearchCommandReceiver
+dataList : List<Student>
+SearchCommandReceiver(List<Student> dataList)
+search() : String

图 45 SearchCommandReceiver 接收者类

这个类通过组合具体的 Student 信息，来执行查找操作。

### 3.3.9 SaveCommand 类

SaveCommand
+saveCommandReceiver : SaveCommandReceiver
+SaveCommand(SaveCommandReceiver saveCommandReceiver)
+execute(String id, String name) : String
+undo() : String

图 46 SaveCommand 类

这个类组合了一个 SaveCommandReceiver 接收者类，通过调用 SaveCommandReceiver 接收者类的具体方法，来实现保存操作。

### 3.3.10 SaveCommandReceiver 接收者类

SaveCommandReceiver
+data : Data
+SaveCommandReceiver(Data data)
+save() : String
+unSave() : String

图 47 SaveCommandReceiver 接收者类

这个类通过组合具体的 Data 信息，来执行保存操作。

### 3.3.11 NoCommand 类

NoCommand
+noCommandReceiver : NoCommandReceiver
+NoCommand(NoCommandReceiver noCommandReceiver)
+execute(String cardID, String name) : String
+undo() : String

图 48 NoCommand 类

这个类组合了一个 NoCommandReceiver 接收者类，通过调用 NoCommandReceiver 接收者类的具体方法，来实现空操作。

### 3.3.12 NoCommandReceiver 接收者类



图 49 NoCommandReceiver 接收者类

这个类不执行任何操作。仅作初始化。

### 3.3.13 Service 类

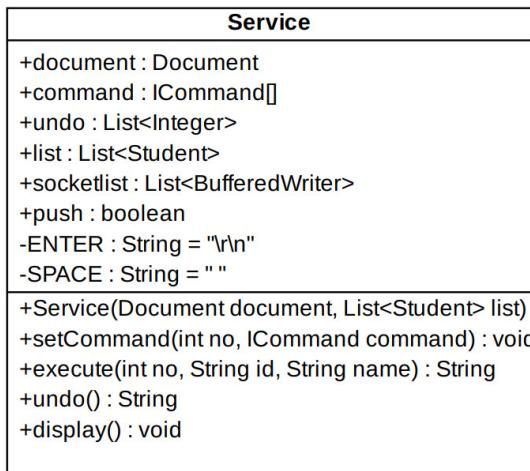


图 50 Service 类

这个类组合了众多 Command 对象，通过 setCommand () 之后，再执行这些操作。

### 3.3.14 Client 调用

```

//设置方法 0增1删2改3查4空5保存
AddCommandReceiver addCommandReceiver = new
AddCommandReceiver(data.getDataList());
service.setCommand(0, new AddCommand(addCommandReceiver));

DeleteCommandReceiver deleteCommandReceiver = new
DeleteCommandReceiver(data.getDataList());
service.setCommand(1, new DeleteCommand(deleteCommandReceiver));

ModifyCommandReceiver modifyCommandReceiver = new
ModifyCommandReceiver(data.getDataList());
service.setCommand(2, new ModifyCommand(modifyCommandReceiver));

SearchCommandReceiver searchCommandReceiver =new
SearchCommandReceiver(data.getDataList());
service.setCommand(3, new SearchCommand(searchCommandReceiver));

```

```

NoCommandReceiver noCommandReceiver = new NoCommandReceiver();
service.setCommand(4, new NoCommand(noCommandReceiver));

SaveCommandReceiver saveCommandReceiver = new SaveCommandReceiver(data);
service.setCommand(5, new SaveCommand(saveCommandReceiver));

//开启服务器
WebServer server = new WebServer(service,handleRootRequest);
server.start(8000);

```

### 3.4 职责链部分

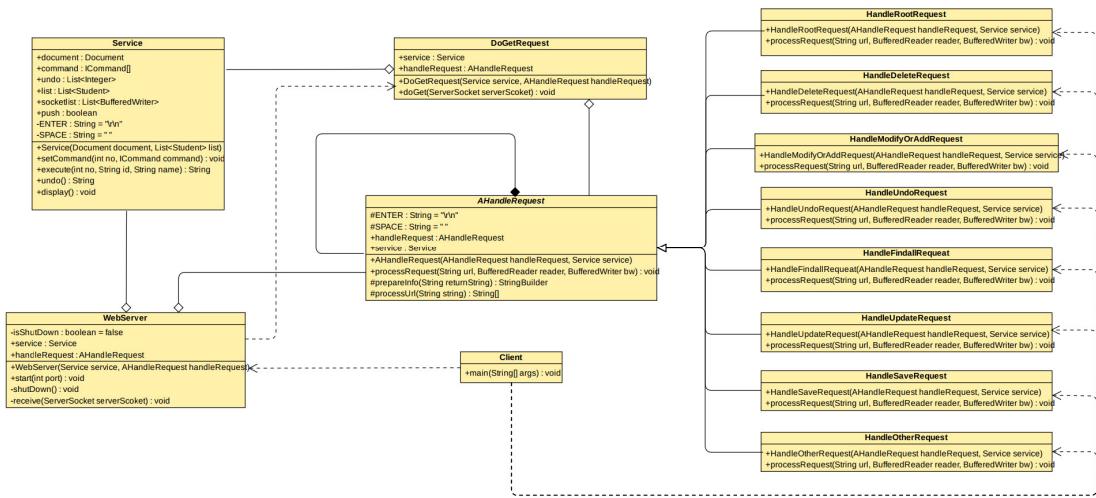


图 51 职责链模式类图

在这个部分中，通过职责链处理，来传递并处理不同的请求，避免了代码冗余，方便扩展。

#### 3.4.1 HandleRequest 抽象类

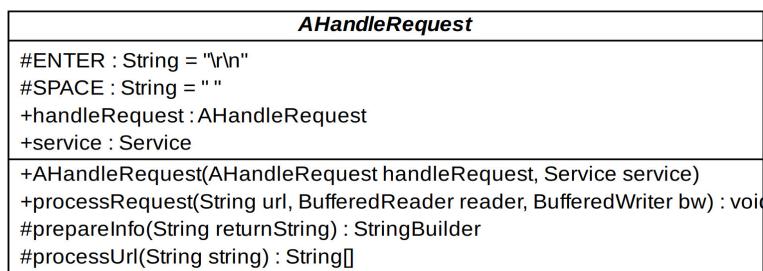


图 52 HandleRequest 抽象类

这个类组合了 `Service`，聚合了自身，定义了公有的处理请求方法和保护权限的处理请求信息的方法。

#### 3.4.2 HandleRootRequest 类

HandleRootRequest
+HandleRootRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 53 HandleRootRequest 类

这个类处理了“/”的请求，也就是首页的加载请求。

### 3. 4. 3 HandleDeleteRequest 类

HandleDeleteRequest
+HandleDeleteRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 54 HandleDeleteRequest 类

这个类处理了“/del”的请求，也就是删除信息请求。

### 3. 4. 4 HandleModifyOrAddRequest 类

HandleModifyOrAddRequest
+HandleModifyOrAddRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 55 HandleModifyOrAddRequest 类

这个类处理了“/mod”的请求，也就是修改信息请求。如果没有对应信息，就是增加信息。

### 3. 4. 5 HandleUndoRequest 类

HandleUndoRequest
+HandleUndoRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 56 HandleUndoRequest 类

这个类处理了“/undo”的请求，也就是撤回操作请求。支持一直撤回操作。

### 3. 4. 6 HandleFindallRequest 类

HandleFindallRequest
+HandleFindallRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 57 HandleFindallRequest 类

这个类处理了“/all”的请求，也就是查询全部信息并返回的请求。

### 3. 4. 7 HandleUpdateRequest 类

HandleUpdateRequest
+HandleUpdateRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 58 HandleUpdateRequest 类

这个类处理了“/update”的请求，也就是当有信息被修改，服务器立即更新全部信息并返回到所有用户的请求。这是 Ajax 的长轮询操作，不同于普通的 Http 协议，是 text/event-stream, keep-alive 的长连接请求。

### 3. 4. 8 HandleSaveRequest 类

HandleSaveRequest
+HandleSaveRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 59 HandleSaveRequest 类

这个类处理了“/save”的请求，也就是持久化全部信息的请求。只有当用户完成修改，关闭浏览器时才会发送此请求。

### 3. 4. 9 HandleOtherRequest 类

HandleOtherRequest
+HandleOtherRequest(AHandleRequest handleRequest, Service service)
+processRequest(String url, BufferedReader reader, BufferedWriter bw) : void

图 60 HandleOtherRequest 类

这个类处理了除了以上所有请求之外的请求，也就是不做任何信息操作，返回默认的提示信息。

### 3. 4. 10 DoGetRequest 类

DoGetRequest
+service : Service
+handleRequest : AHandleRequest
+DoGetRequest(Service service, AHandleRequest handleRequest)
+doGet(Socket serverSocket) : void

图 61 DoGetRequest 类

这个类相当于一个中转请求的类，组合了 Service 和 HandleRequest，传递关键参数给对应的 HandleRequest 来处理。

### 3.4.11 Service 类

Service	
+document : Document	
+command : ICommand[]	
+undo : List<Integer>	
+list : List<Student>	
+socketlist : List<BufferedWriter>	
+push : boolean	
-ENTER : String = "\r\n"	
-SPACE : String = " "	
+Service(Document document, List<Student> list)	
+setCommand(int no, ICommand command) : void	
+execute(int no, String id, String name) : String	
+undo() : String	
+display() : void	

图 62 Service 类

这个类组合了各种命令，相当于中转了各种命令操作。避免了代码冗余。

### 3.4.12 WebServer 类

WebServer	
-isShutdown : boolean	= false
+service : Service	
+handleRequest : AHandleRequest	
+WebServer(Service service, AHandleRequest handleRequest)	
+start(int port) : void	
-shutDown() : void	
-receive(ServerSocket serverScocket) : void	

图 63 WebServer 类

这个类组合了 Service 和 HandleRequest，启动服务器，并且转发请求。

### 3.4.13 Client 调用

```
//在这里设置职责链
HandleOtherRequest handleOtherRequest = new HandleOtherRequest(null, service);
HandleSaveRequest handleSaveRequest = new
HandleSaveRequest(handleOtherRequest, service);
HandleUpdateRequest handleUpdateRequest = new
HandleUpdateRequest(handleSaveRequest, service);
HandleFindallRequest handleFindallRequest = new
HandleFindallRequest(handleUpdateRequest, service);
```

```

HandleUndoRequest handleUndoRequest = new
HandleUndoRequest(handleFindallRequeat,service);
    HandleModifyOrAddRequest handleModifyOrAddRequest = new
HandleModifyOrAddRequest(handleUndoRequest, service);
    HandleDeleteRequest handleDeleteRequest = new
HandleDeleteRequest(handleModifyOrAddRequest,service);
    HandleRootRequest handleRootRequest = new
HandleRootRequest(handleDeleteRequest,service);
    //开启服务器
WebServer server = new WebServer(service,handleRootRequest);
server.start(8000);

```

## 3.5 实现机制

### 3.5.1 基础的增删改查

先读取数据库文件，然后构建出前端 Html，当用户通过浏览器访问主页的时候，html 中的 js 会发出加载操作，执行请求数据的命令。

```

window.onload = function() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', "http://localhost:8000/all");
    xhr.send();
    xhr.onreadystatechange = function() {
        if (xhr.readyState === 4) {
            if (xhr.status >= 200 && xhr.status < 300) {
                clearData();
                insert(JSON.parse(xhr.response));
            }
        }
    };
}

```

图 64 请求数据的 js

网页加载时发送请求，得到数据会清空学生信息的部分 div，然后解析 json 格式的数据，插入到网页中去。

```

function insert(arr) {
    htmlStr = "";
    for (var j = 0; j < arr.length; j++) {
        htmlStr += "<div id='info" +
        (j + 1) +
        "'><button id='del' class='del' type='button' onclick='delStd(" +
        (j + 1) +
        ")'></button><input type='text' name='sid' onchange='check(" +
        (j + 1) +
        ")" value=''" +
        arr[j]["id"] +
        "'><input type='text' name='sname' onchange='check(" +
        (j + 1) +
        ")" value=''" +
        arr[j]["name"] +
        "'><span class='error'></span></div>";
    }
    //插入
    var thisdiv = document.getElementById("bottom");
    thisdiv.insertAdjacentHTML("beforeBegin", htmlStr);
}

```

图 65 插入信息的 js

```

function clearData() {
    var count = document.getElementById("myform").children.length;
    for (var i = 1; i < count; i++) {
        document.getElementById('info' + i).remove();
    }
}

```

图 66 清除 div 的 js

最后网页成功展示。

当页面内容发生变化时，调用不同的请求参数。

```

▼<form action id="myform">
  ▶<div id="info1">...</div>
  ▶<div id="info2">...</div>
  ▶<div id="info3">
    <button id="del" class="del" type="button" onclick="delStd(3)">-</button>
    <input type="text" name="sid" onchange="check(3)" value="20196666">
    <input type="text" name="sname" onchange="check(3)" value="李四">
    <span class="error"></span>
  </div>
  ▶<div id="bottom">
    <button id="add" class="add" type="button" onclick="addStu()">+</button>
  </div>
</form>

```

图 67 触发 js 的增删改操作

```

function delStd(id) {
    var sid = document.getElementById('info' + id).children[1].getAttribute('value');
    var sname = document.getElementById('info' + id).children[2].getAttribute('value');
    document.getElementById('info' + id).remove();
    if (null != sid && null != sname) { //都不为空，再发送请求
        const xhr = new XMLHttpRequest();
        xhr.open('GET', 'http://localhost:8000/del?id=' + sid + '&name=' + sname);
        xhr.send();
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                if (xhr.status >= 200 && xhr.status < 300) {}
            }
        }
    }
}

```

图 68 Js 的删除操作

```

function check(id) {
    var sid = document.getElementById('info' + id).children[1].value;
    var sname = document.getElementById('info' + id).children[2].value;
    var error = document.getElementById('info' + id).children[3];
    if ('' == sid || '' == sname) {
        error.innerHTML = "空值!";
        error.addClass = "error";
    } else if ('' != error.value) {
        error.innerHTML = "";
        error.removeClass = "error";
        const xhr = new XMLHttpRequest();
        xhr.open('GET', "http://localhost:8000/mod?id=" + sid + "&name=" + sname);
        xhr.send();
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                if (xhr.status >= 200 && xhr.status < 300) {
                    if (xhr.responseText == "重复") {
                        error.innerHTML = "重复";
                        error.removeClass = "error";
                    }
                }
            }
        }
    }
}

```

图 69 Js 的检查信息并发送操作

```

function addStu() {
    var aaadiv = document.createElement('div');
    var count = document.getElementById("myform").children.length;
    aaadiv.setAttribute("id", "info" + count);
    var content = "<Button id='del' class='del' type='button' onclick='delStd(" +
        count +
        ")'>-</Button>" +
        "<input type = 'text' name = 'sid' onchange='check(" +
        count +
        ")" +
        "<input type = 'text' name = 'sname' onchange='check(" +
        count +
        ")" +
        "<span class = 'error' > </span>" +
        aaadiv.innerHTML = content;
    var parent = document.getElementById("myform");
    var thisdiv = document.getElementById("bottom");
    parent.insertBefore(aaadiv, thisdiv);
}

```

图 70 js 的添加输入框

```

window.onbeforeunload = function() {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', "http://localhost:8000/save");
    xhr.send();
    var e = window.event || e;
    e.returnValue = ("确定离开当前页面吗？");
}

```

图 71 页面关闭的 js

通过这些操作，前端页面可以向后端发送请求，并且得到回复信息，以便可以正确的展示信息。

### 3.5.2 多用户的数据一致性

通过 Ajax 的长轮询机制，一直向服务器发送更新请求，当信息发生变化时，服务器会立马改变状态，向所有用户发送最新信息，然用户一旦收到“OK”信息，就会执行清空并加载的 js，达到了数据的一致性操作。

```
var sse = new EventSource("http://localhost:8000/update");

sse.addEventListener("update", function(arr) {
    if (arr.data == "OK") {

        const xhr = new XMLHttpRequest();
        xhr.open('GET', "http://localhost:8000/all");
        xhr.send();
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4) {
                if (xhr.status >= 200 && xhr.status < 300) {
                    clearData();
                    insert(JSON.parse(xhr.response));
                    noCommand();
                }
            }
        }
    }
});
```

图 72 Ajax 的长轮询