



中國石油大學 (华东)  
CHINA UNIVERSITY OF PETROLEUM

# 本科毕业设计（论文）

题 目：基于 web 的共享代码写字板设计与实现

学生姓名：张威

学 号：1407010221

专业班级：计算机科学与技术 1402 班

指导教师：张学辉

2018 年 6 月 14 日

# 基于 web 的共享代码写字板设计与实现

## 摘 要

近些年互联网企业用人需求不断增多，远程面试成为诸多公司的选择，这个时候实时协同编辑系统就显得尤为重要。很多类似的在线工具部署在国外，国内用起来不太流畅，所以实现一个简单却不失功能的共享白板，既可以帮助学院算法课堂做实时教学，也可以支持简单的远程面试，本文主要介绍该系统的调研结果，设计，以及实现。

整个系统基于 B/S 架构，同时为了减少实时网络传输成本，解决编辑冲突问题，该系统参考实现了 Google Operational Transformation 算法。本文将详细介绍整个系统的一些难点，基于算法的前后端交互方式，多线程同步问题，节省内存的方法。

**关键词：**协同编辑；Operational Transformation；B/S 架构

# **The design and implementation of Web-based shared white board**

## **Abstract**

In recent years, the demand for Internet companies has been increasing, and remote interviewing has become the choice of many companies. At this time, a real-time collaborative editing system is particularly important. Many similar online tools were deployed abroad, and domestic use was not very smooth. Therefore, Implementing a simple yet functional shared whiteboard could not only help the college algorithm classroom to do real-time teaching, but also support simple The remote interview, this article mainly introduces the system's research results, design, and implementation.

The entire system is based on the B/S architecture. At the same time, in order to reduce the cost of real-time network transmission and solve the problem of editing conflicts, the system references the Google Operational Transformation algorithm. This article will introduce in detail the difficulties of the entire system, based on the algorithm's front and back end interaction mode, multi-thread synchronization problem, and save memory.

**Keywords:** collaborative editing; Operational Transformation; B/S architecture

## 目 录

第 1 章	引言.....	1
第 2 章	相关技术介绍.....	3
2.1	前端技术介绍 .....	3
2.1.1	MVVM 框架 vue.js .....	3
2.1.2	交互技术和样式第三方库.....	3
2.2	后端技术介绍 .....	4
2.2.1	http 服务器 .....	4
2.2.2	websocket 服务器 .....	4
2.2.3	存储.....	4
2.4.4	rest api 库.....	4
第 3 章	操作转移算法.....	5
3.1	概念介绍 .....	5
3.2	操作合并 .....	6
3.3	操作转移 .....	6
第 4 章	系统架构设计.....	9
4.1	架构图总览 .....	9
4.2	编辑流程设计 .....	10
4.3	后端 MVC 设计 .....	11
第 5 章	系统实现.....	12
5.1	后端设计实现 .....	12
5.1.1	后端目录介绍.....	12
5.1.2	jetty websocket session 类介绍 .....	12
5.1.3	EditRoom 类介绍 .....	13
5.1.4	controller 类实现 .....	16
5.1.5	rest api 和 websocket api 的实现.....	16
5.2	前端设计实现 .....	16
5.2.1	前端功能设计.....	16

5.2.2 前端实现原理..... 18

5.3 问题解决和优化 ..... 19

第 6 章 总结和展望..... 21

致谢..... 22

参考文献..... 23

## 第 1 章 引言

近些年互联网企业迅猛发展，用人需求不断增大，软件工程师的需求量也越来越大，从每年 3 月份到 10 月份，各大厂持续招聘，所以面试成为各个公司不可或缺的流程。而考验一个软件工程师是否合格的最好方式就是写代码，但并不是所有的候选人都能参加线下面试，所以一个在线的实时协同编辑系统变得尤为重要。市面上现在出现了很多解决方案，比如牛客网，就是为企业提供一站式招聘流程，其中重要的一环就是实时写代码系统，候选人写代码，面试官可以在另外一个网页实时看到写代码的过程，再比如国外的 Facebook 公司喜欢的 collabedit。作者很喜欢 collabedit 这款产品，唯一的缺点是 collabedit 部署在国外，在国内使用会遇到比较慢的情况，所以本产品功能定位基本和 collabedit 一样，创建一个编辑房间，编辑房间通过 url 标识，协同者通过共享 url 来进入一个编辑房间，进行协作。

作者是在面试过程中对这种实时协同编辑系统产生了很大的兴趣，它有几个显而易见但是值得深思的问题，怎样做到实时通信，怎样减小传输成本，怎样解决编辑冲突，怎样选择存储来保证快速存取。在作者看来，这些问题非常有趣，它们包含了很多 web 技术和工程算法知识，这无疑可以提升工程能力，学习很多对我来说的“新技术”。所以作者的目标，不单单局限于毕业设计，是实现一个简单而不失功能的，具有简洁界面的，原理清晰的，易于部署的共享白板，可用于学院算法课堂的教学，老师直播写代码，或者用于组内面试新人等等，这无疑很有意义。同时，这个项目也会不定期迭代，虚心接受意见，努力做的更好，欢迎同样感兴趣的人一起开发，项目地址：<https://github.com/SGZW/zwedit>。

既然有了目标，剩下的就是要解决问题了，解决问题最好的办法就是逐个击破。刚开始可以定一些解决问题的方向，例如，浏览器服务端通信最常用的是 ajax，是否有更好的选择？减小传输成本也很显然，每次发送和上次编辑的 diff 即可，编辑冲突去调研的已有系统的实现，针对这种场景的存储，显然选用一个内存数据库即可。本文将重点介绍上述问题的解决过程和具体工程实现，主要分为如下章节：

第一章，引言，介绍实时协作编辑系统在当今互联网企业的应用，和设计总体思路。

第二章，相关技术介绍，从前端和后端分别介绍项目使用的技术。

第三章，操作转移算法，讲解操作转移算法的概念，基本操作和解决的问题。

第四章，系统架构设计，介绍整个项目的前后端架构设计，和编辑流程设计。

第五章，具体工程实现，介绍项目具体前后端实现和一些细节处理。

第六章，总结与展望，对项目的总结和期望。

## 第 2 章 相关技术介绍

本章将从前后端分别介绍项目使用的技术，和选用相关技术的原因。

### 2.1 前端技术介绍

传统的前端开发一般是基于 `html5+jquery+css3` 的，后端渲染后返回一个页面，对于复杂交互的网站，显然会造成性能问题，用户体验不好。所以共享白板采用前后端分离的架构，前端是独立于后端的单页应用，后端设计 `api` 采用 `rest api`，把请求抽象成对资源的操作，这样解耦前后端，应用服务器只做数据的处理和交互，不返回静态文件。前端主体语言采用 `es6`<sup>[1]</sup>，因为 `es6` 支持类，打包工具使用 `webpack`<sup>[2]</sup>，开发中使用 `npm` 来管理包，`node` 模拟后端环境。编程白板采用第三方库 `codemirror.js`，这个库有悠久的历史，支持丰富的 `api` 和事件回调。

#### 2.1.1 MVVM 框架 `vue.js`

MVVM 框架主要包括三个部分 `Model`，`View` 和 `ViewModel`，`Model` 指的是数据部分，针对前端来说就是 `js` 对象，`View` 指视图部分指的就是 `html` 中的 `dom` 对象，`ViewModel` 是负责 `View` 和 `Model` 通信的，通常 `ViewModel` 是一个观察者的角色，当数据发生变化，可以监听数据变化，通知视图更新，反之亦然，即双向绑定。

`Vue.js`<sup>[3]</sup>就是这样一套框架，本系统就是基于 `vue.js`，它的核心思想有两个方面，数据驱动和组件化，通过 `directives` 指令去对 `Dom` 做一层封装，同时监听 `dom` 变化，实现数据双向绑定。它的组件设计原则是，每个可视/交互区域都抽象成一个组件，每个组件对应一个工程目录，组件所需要的各种资源就在这个目录就近维护。

`Vue.js` 通过 `Vue Router` 来实现页面跳转，而不需要向后端请求下载新的静态资源文件，当切换路由时，当前的组件被销毁，新的 `url` 的对应的组件被创建，这是构建单页应用的核心部分，实现原理就是监听页面的地址的 `hash` 部分变化，从而实现一个页面可以像一个网站一样工作。

#### 2.1.2 交互技术和样式第三方库

类似于传统的 `ajax`，后端异步发送 `http` 的库使用 `axios`，它是 `vue` 官方推荐的库，和 `vue` 结合的很好，使用类似 `ajax`，就不过多介绍了。

除了 `http` 访问库，因为共享白板要进行实时通信，而用 `axios` 进行轮询肯定不是一个优雅的方法，而 `websocket`<sup>[4]</sup>就是现代浏览器支持的新标准，它是基于 `tcp`<sup>[5]</sup>协议的全



双工通信协议，服务端也可以主动推送数据，保持长连接，充分利用网络连接，而且和普通的 axios 开发并没有什么区别，是事件驱动的，开发很容易。

前端展现样式库，选择使用 iview<sup>[6]</sup>，这个库是基于 vue.js 的，界面清爽。

## 2.2 后端技术介绍

由于实习过程中正在写 java<sup>[7]</sup>，为了使用熟练度，决定使用 java 语言。

### 2.2.1 http 服务器

对于 java 开发者，最常用的服务器就是 tomcat 了，但是 tomcat 总是给人感觉很重的样子，部署麻烦，所以本系统采用 jetty<sup>[8]</sup> http 服务器，jetty server 可以很方便的嵌入到代码中使用，写起来清晰简单，它的生命周期管理类似 tomcat，所以原理认识上也不难。

### 2.2.2 websocket 服务器

共享白板系统除了要实现 http 服务器，也要实现 websocket 服务器，保持长连接，实现服务器推送，这里使用 jetty9 中的 websocket 库，它将 websocket 封装为一个 servlet，当 websocket 连接请求来了，因为 websocket 握手采用 http 协议，它会解析协议，把 http 请求转化为 session 连接对象，用来以后推送数据和接收数据，服务器端编写 websocket 接收信息的方法也是基于事件驱动的，当有消息来到的时候，会调用相应的注解函数进行处理。

### 2.2.3 存储

存储场景是需要选用快速存取的数据库，而且只需要存储 url 和对应的文本信息即可，选用一个 key-value 数据库即可。redis 是一个开源的使用 ANSI C 语言编写，支持网络，可基于内存亦可持久化的日志型，key-value 数据库，支持多种语言的 api，其中 java api 是 Jedis 库。redis 性能高，支持丰富的数据类型，并且操作具有原子性的特点，是应用广泛，经过生产环境检验的 key-value 数据库。所以本系统采用 redis 作为后端存储。

### 2.4.4 rest api 库

后端选用 java jersey 库作为 rest api，它是专门针对 rest 理念设计的 api 库，基于注解定义 url，请求类型和返回类型，把 http 请求抽象成对资源的增删查改，代码阅读性强，可维护性高。

## 第3章 操作转移算法

实时协同编辑系统两个棘手的问题，实时通信通过 websocket 解决，而解决编辑冲突，减小网络传输成本，就需要利用谷歌出品的操作转移算法，全称 Operational Transformation<sup>[9]</sup>，下面简单介绍算法核心过程。

### 3.1 概念介绍

协同者对文档的修改抽象成一个 operation，一个 operation 包含多个 action，action 有三类。Insert(str)，插入一个字符串 str。Delete(n)，删除 n 个字符。Retain(n)，保持 n 个字符不变。其实 operation 就是 action 的列表。

将 operation 应用到文本字符串时，有一个隐形的光标位于字符串起点，retain 向后移动光标，insert 和 delete 在光标所在的位置对字符串进行插入和删除，operation 应用完之后，光标必须处于字符串的末端（这保证了应用 operation 的正确性）。

如图 3-1 插入实例，Operation 包含两个 action:retain(11)和 insert(“ dolor”)，顺序执行这两个 action，光标从左向右移动 11 个字符，然后插入字符串，最终光标到达字符串末端，operation 完成。

```
1 var operation = new ot.Operation()
2   .retain(11)
3   .insert(" dolor");
4
5 operation.apply("lorem ipsum"); // => "lorem ipsum dolor"
```

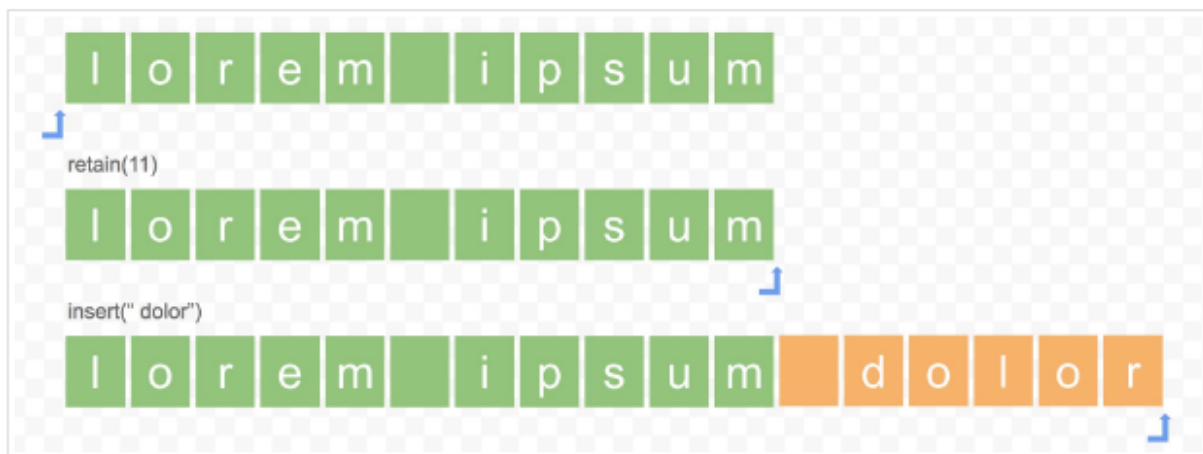


图 3-1 插入实例

## 3.2 操作合并

前面提到一个 operation 是一个 action 列表，这样一个显然的好处是：多个 operation 可以按照时间先后合成一个 operation。例如有两个 operation 要应用到一个字符串上，正常做法是这两个 operation 依次应用，现在可以先合并，然后应用到字符串上。在本系统中，作者分别实现了 js 版，和 java 版本的操作转移库，由于前端部分使用 es6 语法，支持类书写，所以前后端的对应 operation 的类叫 textOperation，后缀分别是 java 和 js，操作合并的方法叫做 compose，接收一个 textOperation 对象，返回一个当前对象和传入对象的合并结果。如果对实现细节感兴趣可以选择看 java 版本或者 js 版本的实现。

## 3.3 操作转移

操作转移算法的核心就是这个 textOperation 类中静态方法 transform，这个方法实现了转换算法，操作转移的思路就是将编辑转换成操作，当多人协同操作时，就需要对这些操作进行转换，使得最终文档的内容一致。

```
1  var str = 'go'
```

Client-A:

```
1  var opA = new ot.Operation()  
2    .retain(2)  
3    .insert("a");  
4  
5  opA.apply(str) // => goa
```

Client-B:

```
1  var opB = new ot.Operation()  
2    .retain(2)  
3    .insert("t");  
4  opB.apply(str) // => got
```

图 3-2 冲突实例

如图 3-2 冲突实例所示，用户对文档的插入操作会立即应用到本地副本，然后将操作 opA 和 opB 上传到服务器；服务器先后接收到这两个操作，然后对服务端的这份文

档进行应用，假设以先 opA 后 opB 的顺序，应用 opA 后，字符串变成 goa，长度变成 3，隐形的光标未到字符串末端，执行会抛出错误。此外，服务端也会将 opA 发送给 ClientB，opB 发给 ClientA，由于此时字符串长度都变成 3，应用过程中隐形的光标未到字符串末端，也会抛出错误。

上述冲突就是 transform 要解决的问题，两个用户得到的文档内容不一致了，这种情况就需要 transform 操作转换了。



图 3-3 transform 过程

多个客户端操作文档，其实可以抽象成，一个客户端和一个服务端操作同一个文档，因为两个客户端之间的影响是通过先同步到服务端，然后再发送给另外一个客户端，所以只要解决单个客户端和服务端编辑冲突问题即可。如图 3-3 transform 过程左，a 表示客户端的 operation，b 表示服务端的 operation，两者相交的顶点表示文档状态相同，此时客户端和服务端分别应用 a 和 b，这时两边的文档内容都发生变化，且不一致；为了客户端和服务端的文档达成一致的状态，我们需要对 a 和 b 进行操作转换  $\text{transform}(a, b) \Rightarrow (a', b')$  得到两个衍生的操作 a' 和 b'。如图 3-3 transform 过程右，a' 在服务端应用，b' 在客户端应用，最终文档内容达成一致。

Transform 算法的细节请看 textOperation 类的 transform 静态方法，过程类似 compose 过程，维护两个光标，遍历两个 action 列表。上面这种问题称之为：one-step diamond problem，客户端和服务端同时对文档执行一个 operation，从上方顶点分裂开两条边，相当于 diamond 图形上面两条边，而两条转化边相当于下面两条边。这样先分后合，最后达成一致。

但是遇到更多的操作是，服务端和客户端都不只一个 operation，比如客户端有一个操作，服务端有两个操作，此时转化方式如图 3-4，展示了 1:2 需要构建两个 diamond 图形，转化两次才能让客户端和服务端的文档收敛到相同的状态，1:N 也是一样，如图 3-5 所示，阐述了 1:N 转化过程，假设客户端一个操作 clientOp，服务端有多个操作

serverOps，那么循环计算出客户端的操作转换，然后再服务端应用操作 clientOpPrime 即可。当碰到 N:M 时，转化成 1:N 或者 N:1 来解决。

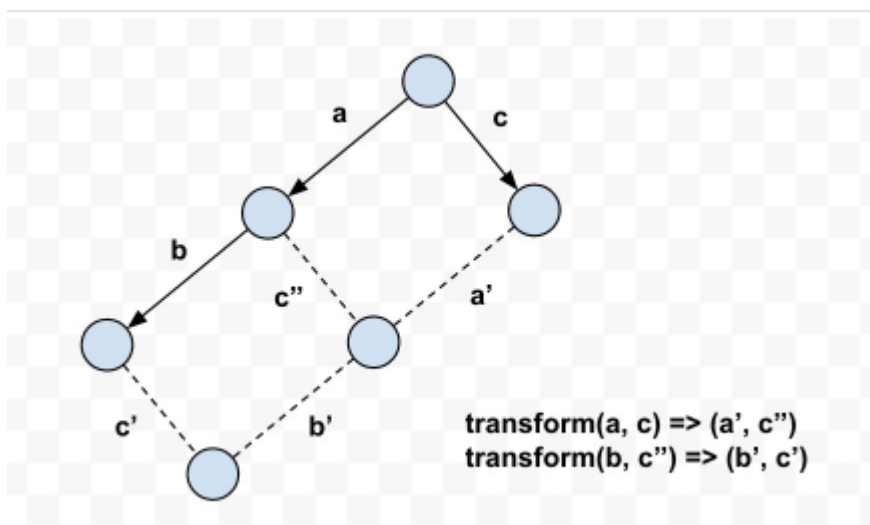


图 3-4 1:2 转化方式

```
1  var clientOp
2  var serverOps = []
3  var clientOpPrime = clientOp
4  for(var i = 0; i < serverOps.length; i++) {
5    clientOpPrime = transform(clientOpPrime, serverOps[i])[0]
6  }
```

图 3-5 1:N 转化过程

## 第4章 系统架构设计

因为共享白板系统是基于 web 技术栈的，所以架构就是传统的 B/S 架构，唯一的不同是前后端分离，前端是 MVVM 框架，后端是简单的 MVC 框架，由于后端业务逻辑没有特别复杂，所以我并没有引入过多的第三方库，针对这样的一个代码量不算多的项目，一个自己实现的 controller 和 model 层明显更加灵活，方便组织。

### 4.1 架构图总览

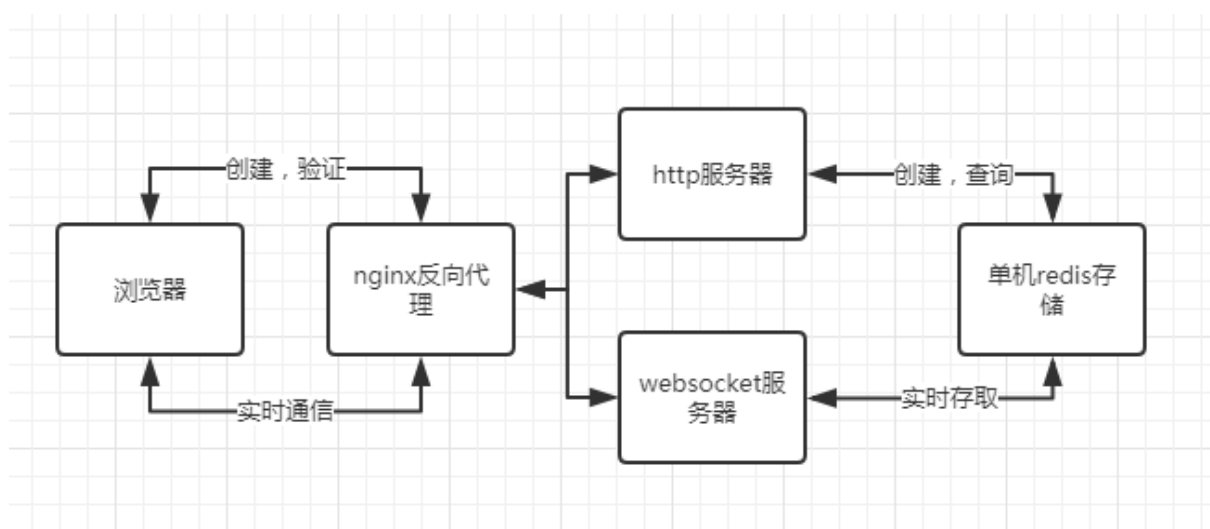


图 4-1 整体架构图

如 4-1 整体架构图所示，从左至右，浏览器在各个用户的 pc 上的，浏览器必须实现两个协议，websocket 协议和 http 协议，整个项目是基于 chrome 浏览器最新正式版做的，未对其它浏览器做兼容，不过用小米六自带的手机浏览器也可使用。当用户打开网站时，由 nginx<sup>[10]</sup>服务器做反向代理，直接返回打包好的静态文件，静态文件包括 html+js+css 部分。

中间部分 nginx 反向代理服务器，可以和应用服务器分开部署，在应用服务器和浏览器直接加一层 nginx 服务器的原因是业界公认的最佳工程实践，首先加一层反向代理服务器可以接收所有的请求，自然可以在这一层做一些安全策略，比如 https 的 ssl 层就可以在这实现，或者做一些访问拦截。第二，直接把静态文件部署在反向代理服务器上，加速静态资源返回。这两点都可以很好的减轻应用服务器的负载，更专注的处理业务逻辑。选用 nginx 的原因是因为 nginx 用 c 编写，系统资源开销小，底层采用 epoll 作为开发模型，支持高并发，并且还支持一些高级功能，例如负载均衡等。当后期如果使用的

人变多，可以考虑后端分布式的架构，多应用服务器参与服务。

当浏览器下载静态资源后，js 代码开始工作，当创建共享白板时，会通过 axios 向后端发送 http 创建白板资源请求。当打开一个共享白板时，会通过 axios 向后端发送 http 获取特定白板资源请求，来验证特定白板资源是否存在。所有的 http 请求作用在 http 服务器，http 服务器通过创建或者获取相应的 redis key-value 数据结构来完成相应的功能。

当协同人员打开特定共享白板开始编辑后，编辑的变化会通过 websocket 协议不断同步到 websocket 服务器，同时 websocket 服务器也会主动推送数据到浏览器来保证达到一致性，websocket 服务器通过 redis 来持久化数据。

如图 4-1，看起来 websocket 服务器和 http 服务器是分离的，其实它们只是单个进程的不同线程，前面讲过，websocket 和 http 协议有良好的兼容性，默认端口也是 80 和 443，握手阶段采用 http 协议，因此握手时不容易被屏蔽，能通过各种 http 代理，所以他们可以共享端口，统称应用服务器。

最后讲一下最右边的 redis，redis 是基于内存的数据库，用来和应用服务器交互，可持久化白板数据，他有良好的特性，高性能，操作原子性等等，他会定期 dump 数据到磁盘，通过合理的配置策略，可以保证数据不失。

## 4.2 编辑流程设计

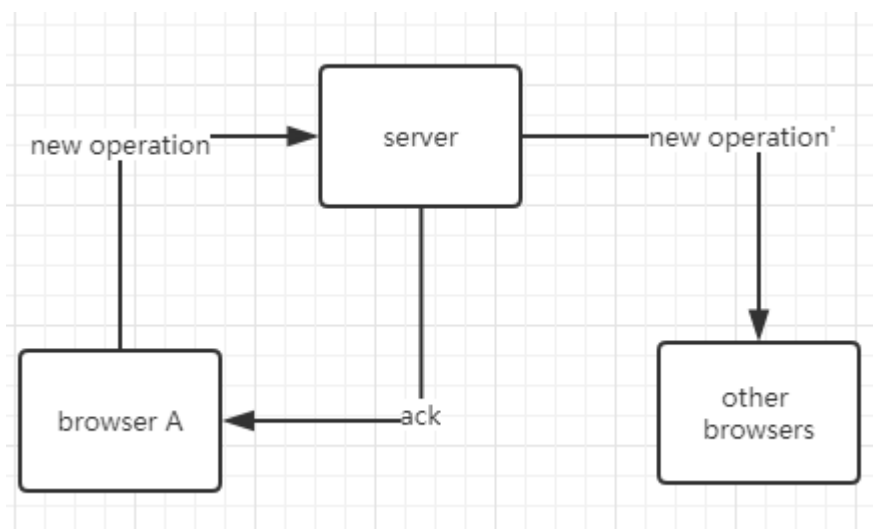


图 4-2 编辑流程

如图 4-2 的编辑流程，首先用户在浏览器 A 做出编辑，通过 codemirror 的 changes 的变化事件，我们可以获得编辑变化，然后把变化转化为一个新的 operation，发送给服务端，服务端接收到新的 operation 后，会做冲突处理，就是操作转移算法中的核心算法，transform，之所以要做 transform，根据上面的抽象思想，解决一个客户端和一个服务端

即可，所以此时服务端可能也做了修改（其实是其它浏览器端做了修改），比浏览器 A 发送这个修改要靠前，这个时候需要做转化。转化过程就是像操作转移算法讲 1:N 模型一样，得到服务器需要应用的 `operation'`，之后应用到服务端保存的文本，持久化到 redis。

把数据持久化到 redis 中，并没有结束，之后默认其他浏览器和服务端保存的副本相同，把 `operation'` 发送到除浏览器 A 外的其它浏览器上，在其它浏览器上接着进行冲突检测和应用。除了发送给别的浏览器，也要给浏览器 A 发送 ACK 消息，代表服务器已经接收到这个新的 `operation`，并且已经应用。ACK 环节很重要，因为每个客户端也需要做冲突处理，会缓存一些 `operation`，服务端已确认的 `operation` 不需要参与冲突处理的 transform 过程，当浏览器从服务器收到一个新的 `operation`，未确认的 `operation` 需要对从服务端接收的 `operation` 进行 transform 才能应用到客户端的文档。

### 4.3 后端 MVC 设计

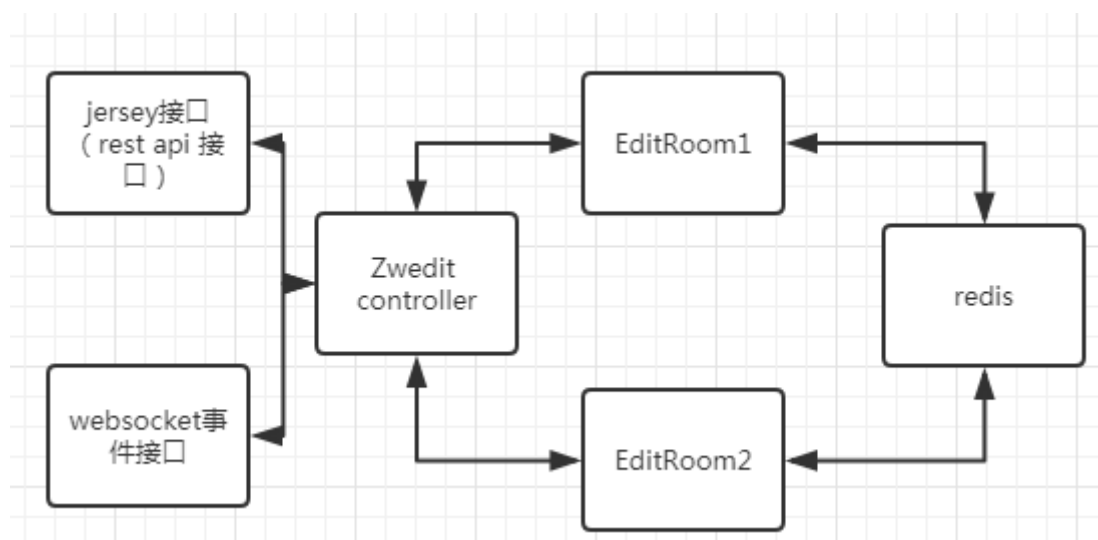


图 4-3 后端 mvc 设计

如图 4-3，展示的后端 mvc 设计，各个接口不直接与 redis 数据库交互，而是通过一个中心 Zwedit controller 来控制。有了整体的设计，就需要设计控制器的包含的数据结构。对于本系统，其实很简单，控制器维护一个 url 到编辑房间（即白板）的 kv 结构，管理多个编辑房间，所以由此引出 EditRoom 类，所有关于具体编辑房间里的数据存取，数据库交互，编辑冲突检测等都交给 EditRoom 类去做，这样可以保证控制器的逻辑清晰，控制器只需要提供白板（即编辑房间）的创建，获取，检测即可。

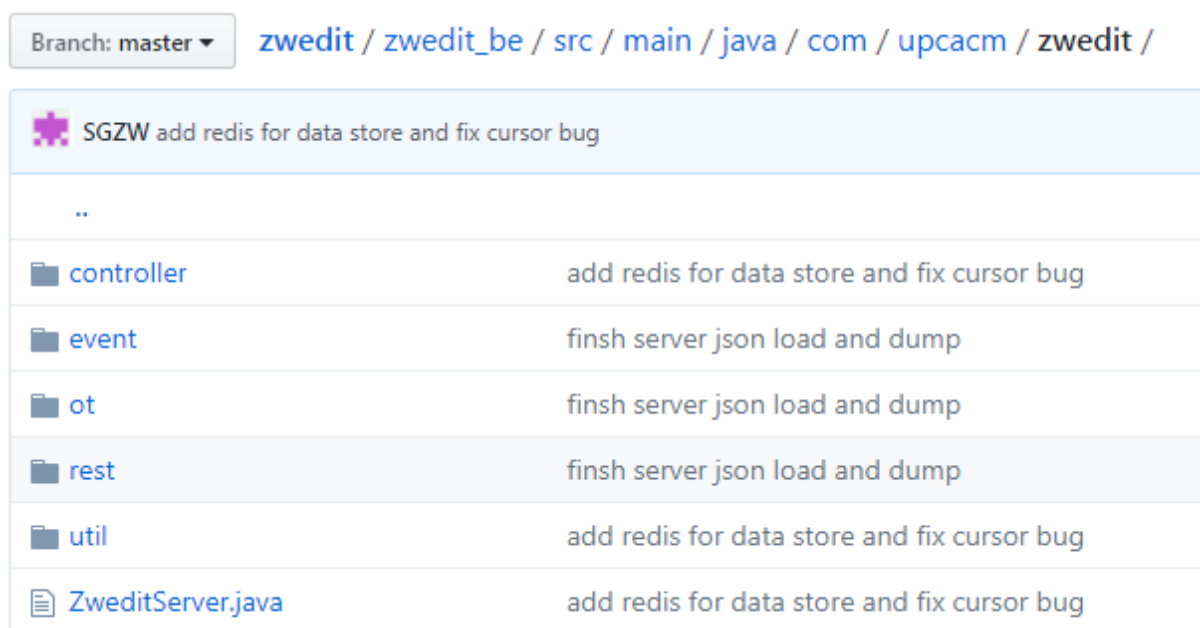


## 第 5 章 系统实现

本章主要讲共享白板系统的具体工程实现，主要包括以下内容：后端设计实现，前端设计实现，问题解决和优化。

### 5.1 后端设计实现

#### 5.1.1 后端目录介绍



Branch: master ▾ zwedit / zwedit_be / src / main / java / com / upcacm / zwedit /	
SGZW add redis for data store and fix cursor bug	
..	
controller	add redis for data store and fix cursor bug
event	finsh server json load and dump
ot	finsh server json load and dump
rest	finsh server json load and dump
util	add redis for data store and fix cursor bug
ZweditServer.java	add redis for data store and fix cursor bug

图 5-1 后端目录

如图 5-1 所示的工程目录图，controller 目录就是 mvc 层的 c 实现，里面主要有 ZweditController 类和 EditRoom 类，event 目录就是 jetty 规定的 Servlet 类，和处理 websocket 事件响应的类，ot 目录是操作转移算法的实现，主要包括 action 类和 operation 类，rest 目录包含定义 http 资源和接口的类，util 目录包含 JSON 解析类，和 redis 连接线程池管理类等工具类，最后的 ZweditServer.java 就是项目的主类了。

#### 5.1.2 jetty websocket session 类介绍

首先我们都知道 websocket 是一个工作在 tcp 协议上的全双工通信协议，做过 socket 编程的同学都知道进行通信需要先获得用来通信的句柄，所以通信双方都需要知道句柄是什么才能发送数据，而 jetty 容器对 websocket 的通信句柄的抽象就是一个 session 类，当客户端发起连接时，jetty 就会生成一个 session 类对象用于后续接收和发送信息。具体的 Session 类对象可以在 websocket 事件相应单例类的方法中获取。

### 5.1.3 EditRoom 类介绍

EditRoom 是核心类，EditRoom 类实例和共享白板的 url 一一对应。下面将具体看 EditRoom 类是怎么写的。

```
19 public class EditRoom {
20
21     private volatile int sessionId = 0;
22
23     private volatile String text = "";
24
25     private HashMap<String, Session> store = null;
26
27     public String roomUrl = "";
28
29     private ArrayList<TextOperation> ops = null;
30
31     private HashMap<String, Integer> lastOp = null;
32
33     private Jedis redis = null;
```

图 5-2 EditRoom 类属性

如图 5-2，展现了 EditRoom 类的属性。sessionId 是 int 类型，是浏览器 id 生成器，每当有一个新的浏览器加入当前的共享白板，sessionId 就会+1，并返回给浏览器用来标记新加入的浏览器，声明为易失类型，强制每次读内存，防止被寄存器优化。这个属性是必要的，因为服务器需要管理所有的协同用户，所以需要给每个用户一个标号。Text 属性就是当前白板保存的文档，并用于持久化到 redis 中。Store 是一个 Map<String, Session>类型，Session 之前说过是句柄，这个维护的是用户标号到用户 websocket 句柄的映射，用来发送数据。roomUrl 属性很简单，就是当前白板对应的 url。Ops 是历史 operation 列表，这个是由于做 transform 转换用的，因为不同用户的编辑请求可能会同时到达，这个时候要串行进行，然后后来的需要和先到的做 transform 转化才可以应用到服务器的文档。lastOp 是一个 Map<String, Integer>类型，key 就是用户标号，value 就是上次这个用户新产生 operation 并且作用到服务端时的文档版本号。Redis 属性是 Jedis 类型，即 java redis 连接，用于向 redis 发送读取命令的，每个 EditRoom 对象都独享一个 redis 连接对象，因为连接线程不安全。

引入版本号的原因：正如前面操作转移算法讲的，抽象一下，需要解决一个客户端和一个服务端进行协作的场景，还是 1:N 的场景，此时客户端产生了一个 Operation，服务端产生了 N 个 operation，这个时候服务端需要把客户端的 operation 转化 n 次才能应

用在文档上。但是这  $n$  个 Operation 需要存下来，所以用一个 ArrayList 记录所有的历史 Operation。除了记录历史的 operation，还是不知道要找哪  $n$  个 Operation，所以引入版本信息，一个 operation 代表一个版本，客户端版本号代表已经在客户端应用了几个 operation（包括自己产生的和服务端发来的），客户端发送新的 operation 时，携带自己的版本号信息，服务端在应用新的 operation 时，会下标为客户端版本号的位置遍历到 ArrayList 的末端进行操作转移转化新的 operation，之后作用到服务端文档。所以通过这样就找到了那  $N$  个 operation。

版本号实现：每一个操作对应一次版本号顺序递增，客户端和服务端各自维持一个版本号，客户端进行一次编辑产生一个 operation 或者从服务端介绍到一个 operation，客户端版本号都会递增 1。服务端把客户端并发编辑操作串行化，保存所有的历史 operation，这样服务端的版本号就是 ArrayList 的长度，不需要另外记录。显然，客户端版本号始终小于等于服务端版本号。

```
public synchronized String getNewSessionId() {
    sessionId += 1;
    return String.valueOf(sessionId);
}

public void refreshRedisClient() throws Exception {
    if(this.redis == null) {
        this.redis = RedisPool.getJedis();
        this.text = this.redis.get(this.roomUrl);
        if(this.text == null) {
            this.redis.set(this.roomUrl, "");
            this.text = "";
        }
    }
}

public synchronized void addSessionAndSendInitResponse(String sid, Session session) throws Exception {
    //every connect monitor alive
    this.refreshRedisClient();

    store.put(sid, session);
    LinkedHashMap<String, Object> ret = new LinkedHashMap<String, Object>();
    ret.put("type", "new");
    ret.put("revision", new Integer(this.ops.size()));
    ret.put("text", this.text);
    String res = JsonUtil.dumpMap(ret);
    session.getRemote().sendString(res, null);
}
```

图 5-3 EditRoom 类的一些方法

如图 5-3 展示了 EditRoom 类的一些方法，首先是 getNewSessionId 方法，当一个新的用户进入协作，会通过 http 接口获取新的用户标号，这个函数同步不可重入，保证

数据准确。refreshRedisClient 方法主要是从 redis 连接线程池获取一个新的连接对象，并进行相关初始化。addSessionAndSendInitResponse 函数在有新的 websocket 连接进入的时候触发，参数是用户标号和 Session 对象，首先初始化 redis 连接对象，然后存储 session 对象，之后向浏览器发送初始化数据，数据形式是 json 格式的，包括数据类型 type，值是 new 代表是初始化，版本号信息，和文档信息，这个方法也是需要同步不可重入，保证串行化。

```
public synchronized void process(String sid, int revision, TextOperation nop) throws Exception {

    Integer last = getLastRevision(sid);
    if (last != null && last.intValue() >= revision) return;
    for (int i = revision; i < ops.size(); ++i) {
        Pair<TextOperation, TextOperation> p = TextOperation.transform(nop, ops.get(i));
        nop = p.getA();
    }
    this.text = nop.apply(this.text);
    this.redis.set(this.roomUrl, this.text);
    this.saveOp(sid, nop);
    ArrayList<String> del = new ArrayList<String>();
    for (Map.Entry<String, Session> entry: store.entrySet()) {
        String id = entry.getKey();
        Session session = entry.getValue();
        if (!session.isOpen()) {
            del.add(id);
            continue;
        }
        if (!id.equals(sid)) session.getRemote().sendString(getResponse(sid, nop.getActionsList()));
        else session.getRemote().sendString(getResponse(sid, new LinkedList()), null);
    }
    for (String d: del) {
        if(store.containsKey(d)) store.remove(d);
        if(lastOp.containsKey(d)) lastOp.remove(d);
    }
    if (store.size() == 0) {
        RedisPool.returnResource(this.redis);
        this.redis = null;
    }
}
```

图 5-4 EditRoom 类的 process 方法

如图 5-4 介绍的 process 方法，当有新 operation 到来被触发，首先肯定也是同步不可重入保证串行化，首先和上次该用户产生新 operation 时的版本号比较，如果小于等于，说明不合法，不做处理，防止重复发送。然后安装 1:N 模型进行操作转化，应用到服务端文档，可持久化到 redis，存下这次的版本号和转化后的 operation。之后把向 open 状态的 websocket 发送 operation，发送的的格式为 json 数据，包括数据类型 type=merge，

代表是新 operation, sid=当前用户标号, 这样浏览器端收到以后, 可以根据 sid 信息判断是 ACK 消息还是新 operation, 还有 actions 列表代表实际的 operation。注意函数末端有个优化, 就在 EditRoom 所管理的 session 对象中剔除不是 open 状态的对象, 这样节省内存, 也加快遍历速度, 当 EditRoom 管理的 session 数量为 0 时, 把当前 EditRoom 持有的 redis 连接对象返回给线程池, 供其他类使用, 减少不必要的浪费, 同时增加并发。

下面介绍的很多类都是基于 EditRoom 类做文章的。

#### 5.1.4 controller 类实现

Controller 类就是管理 EditRoom 对象的, 这是一个单例类, 维护一个 Map<String, EditRoom>的映射, key 是 url, value 是该 url 对应的 EditRoom 对象。方法主要提供, 基于 url 的对象获取, 创建和验证, 总体很简单。

#### 5.1.5 rest api 和 websocket api 的实现

后端 rest api 实现按照功能设计, 提供一个创建白板和获取白板的接口即可, 验证功能可以通过获取接口, 判断是否获取成功即可, 基于 controller 类实现。

Websocket api 实现也很简单, 因为 websocket 的特点, 所以他的方法定义和触发就是很适合事件驱动机制, websocket 规范里包括 4 个事件函数: onConnect, 连接时触发; onMessage, 收到消息触发; onError, 抛出错误时触发; onClose, 关闭连接时触发。在 jetty 中我们通过定义 jetty websocket 注解来定义这样一个事件响应类, 响应时传入的参数是 session 对象和其它必要的参数, 对象是可以复用的, 所以 jetty 只会实例化事件响应类一次, 和 servlet 类的实现基本一致, 初始化一次, 后面直接复用, 节省内存。

## 5.2 前端设计实现

### 5.2.1 前端功能设计

共享白板的分享是基于 url 的, 每个共享白板都有唯一的 url 标识, 参与协作的人通过共享 url 实现协作。之所以选择基于 url 实现, 一是实现成本小, 二足够灵活。

下面将从首页, 共享白板页, 404 页来介绍前端功能设计。

首页功能设计如图 5-5 所示, 点击中间的按钮即可创建并进入一个新的白板, 界面简洁清爽。

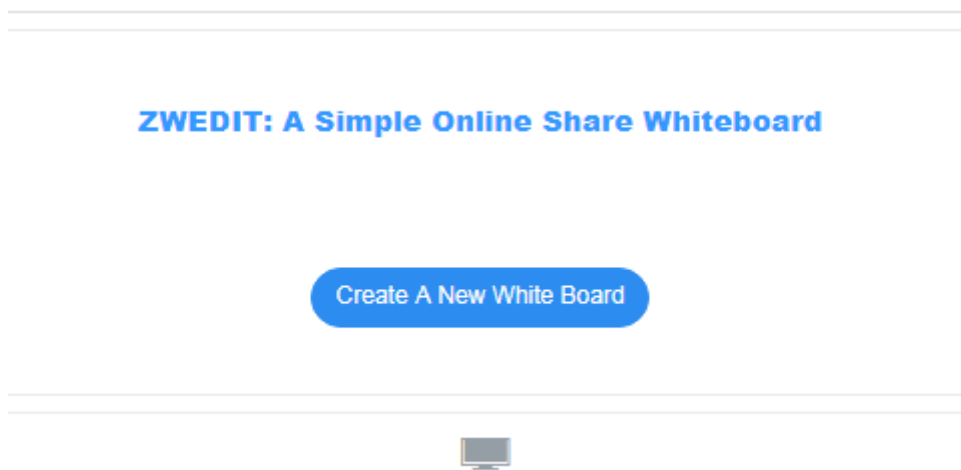


图 5-5 首页功能设计

共享白板页功能设计如图 5-6 所示，页面左侧是写代码区域，按下 **ctrl** 键会有补全提示，支持简单的语言高亮，括号匹配高亮，和代码块折叠。右侧是功能列表和提示区域，最上方提示当前共享白板是否和服务端保持连接，为了节省资源，当用户停止编辑超过一段时间，连接就会自动断开，提示 **normal** 表示连接正常，提示 **error** 表示连接已经断开，需要用户刷新界面，目前超时断开时间设置的是 5 分钟。下方的 **Language** 选择列表可以选择语言的高亮类型，目前支持 **c**, **c#**, **java**, **javascript**, **python**, **golang**。**Theme** 选择列表可以选择代码板的背景主题颜色，支持不同用户的喜好。

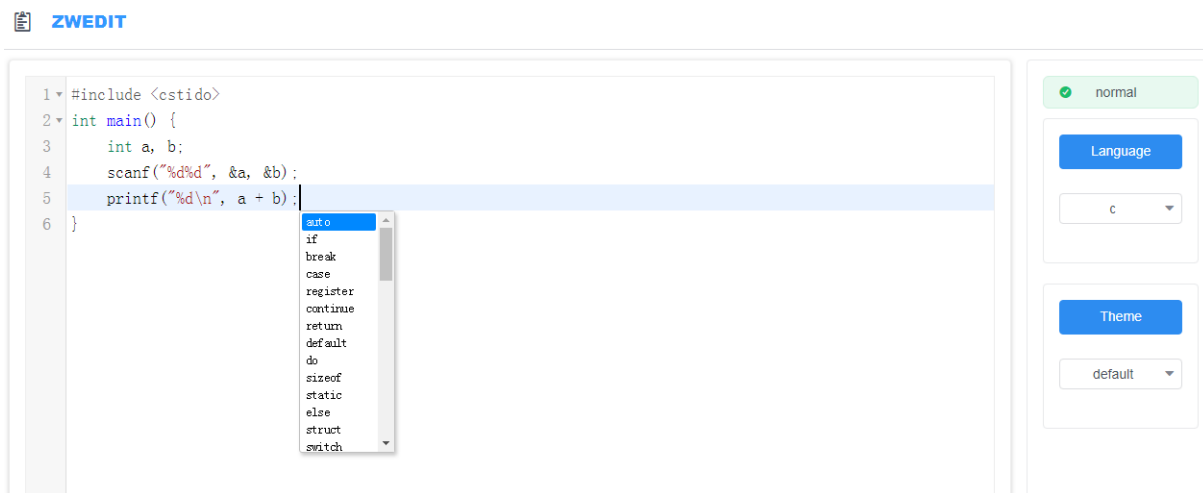


图 5-6 共享白板页功能设计

404 页功能设计如图 5-7 所示，当用户进入一个不存在的共享白板房间时或者输入一个不合法 **url** 时，会进入 404 界面。



图 5-7 404 页功能设计

### 5.2.2 前端实现原理

前端实现是围绕 `codemirror.js` 和 `websocket` 设计实现的，当进入一个贡献白板页面时，会先向后端发出 `http` 请求，验证当前 `url` 是否存在，如果不存在直接跳到 404 界面，存在的话存储当前浏览器的用户标号用于以后的请求。

完成验证后，就会向后端发出 `websocket` 连接请求，这样后端对应的 `EditRoom` 对象会把当前 `websocket` 的句柄加入管理，然后发出 `type=new` 的 `json` 消息。`Websocket` 在浏览器端的实现和服务端没有什么不同，它的 `api` 也是基于事件响应的，也有和后端对应的 4 个方法，我们需要关注的就是 `onMessage` 方法，根据从服务端获取的 `json` 数据来区别应该做什么事情。当正常连接后，当前白板会收到初始化消息，文本和版本号，之后我们用 `codemirror.js` 中的 `api` 填充白板，同时存储下版本号信息。

下面阐述前端如何处理编辑冲突。同样我们用 `es6` 语法实现了操作转移算法，因为 `es6` 语法支持类语法，所以只需要简单的翻译 `java` 代码即可。有了上面服务端的实现流程作为参考，浏览器端也可以维护一个历史 `operation` 序列，类似 1:N 模型进行操作转移。但是经过思考发现，维护历史 `operation` 序列没有必要，还浪费内存。服务端需要维护序列是因为涉及多个用户协作，每个用户的版本号都不一样，所以需要找到的 `N` 个 `operation` 不一样，要从不同的起始下标开始遍历进行转换。客户端的处理场景只涉及两个用户，当服务端有新的 `operation` 到来的时候，客户端已经发生了 `N` 个操作，如何找到这 `N` 个操作呢？在整体架构讲解时说过，用户每发送一个 `operation` 到服务端，需要经过服务端发出 `ACK` 确认才行，`ACK` 的作用就是为了找到这 `N` 个操作进行转移。已确认的 `operation` 已经应用到服务端，不需要再转移，浏览器端丢弃即可，未确认的 `operation` 就是那 `N` 个用来转移的 `operation`，所以浏览器端只需要保存未确认 `operation` 即可，这部分 `operation` 不需要维护一个序列，之前介绍算法时提过，两个相邻的 `operation` 可以

compose，所以只需要把未确认的 operation 按顺序 compose 在一起，抽象成一个操作，就可以解决转移问题。

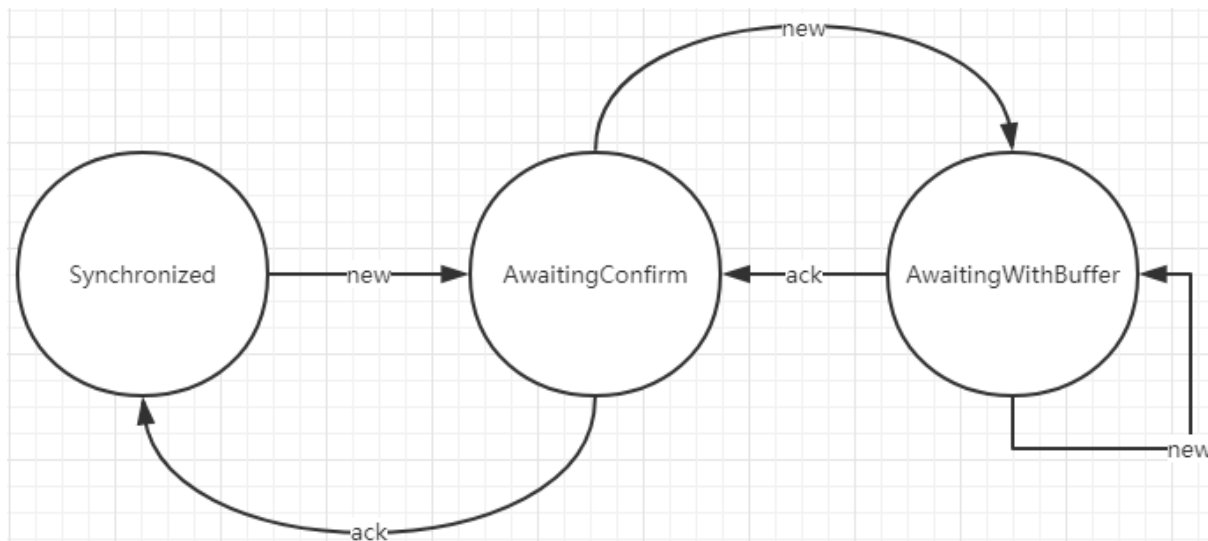


图 5-8 浏览器端状态转移

如上面所说的浏览器端如何处理编辑冲突，处理过程可以抽象成如图 5-8 显示的状态转移过程。当打开一个新的共享白板，接到服务端的初始化数据后，处于 Synchronized 状态；当接到服务端的 operation 时，可以直接应用在文档上；当用户产生新的 operation 时，会存下并发送这个 operation，进入 AwaitingConfirm 状态。AwaitingConfirm 状态代表现在有一个 operation 正在等待服务端 ack：此时如果接到服务端 ack 信号，会重新回到 Synchronized 状态，丢弃等待 ACK 的 operation；当接收到服务端的 operation 时，先用当前等待 ACK 的 operation 对服务端的 operation 进行转移，然后应用在客户端文档上；当用户产生新的 operation 时，存下当前新 operation，进入 AwaitingWithBuffer 状态。AwaitingWithBuffer 状态表示现在有一个操作等待服务端 ACK，一个操作等待被发送：此时如果接到 ACK 信号，等待发送的 operation 会被发送，同时丢弃等待 ACK 的 operation，进入 AwaitingConfirm 状态；当用户产生新的 operation 时，将等待发送的 operation 和新 operation compose 成一个 operation，再次进入 AwaitingWithBuffer 状态；当接到服务端 operation 时，按顺序先用等待 ACK 的 operation 对其进行转化，再用等待发送的 operation 转化，才能应用到客户端文档，因为还没被确认。

综上，前端实现介绍完毕。

### 5.3 问题解决和优化

实时协同编辑的两个问题：实时通信，websocket 可以做的很好；编辑冲突，操作转移算法也做的很好。下面介绍一下工程实现中的一些细节和优化。



首先要知道服务端是工作多线程下的，所以线程同步问题需要解决，例如当两个浏览器的 `operation` 同时到达服务端，并发执行肯定会导致错误，所以这个地方我们需要让他们串行执行，利用 `java` 的 `synchronized` 同步锁，可以保证串行，同时锁的粒度如果很大，就会减小并发，上锁的粒度是一个共享白板一个，不同白板直接的处理和并行执行，互不影响。

还有一个问题是，网络请求到达顺序的问题，通过操作转移算法我们可以发现，`operation` 是严格依赖顺序的，由于网络原因导致一个请求后发先至，也会导致错误，所以加入 `ACK` 机制可以保序，也可以保证最终一致性。

版本号也是必不可少的，版本号是多用户合作的基础。

最后需要细节优化就是本系统利用了 `redis` 连接线程池，当 `EditRoom` 对象进行数据读取时，需要使用 `redis` 连接，但是 `redis` 连接对象线程不安全，所以需要加锁，或者实例化多个对象，第一种减小并发，不考虑，第二种会导致实例化对象过多，创建销毁开销大，所以本系统采用 `redis` 连接线程池，事先创建好一定量的连接线程（可配置），当需要时，从线程池中取，使用完毕后归还这个对象，供其他 `EditRoom` 对象使用。

## 第 6 章 总结和展望

通过毕业设计，作者完成了一个简单，不失功能，界面清爽的共享白板系统，在设计过程中参考了很多经典设计，比如 mvc 设计，状态机设计等。实现过程中，有很多细节处理让我学到很多，例如状态转移怎么写，线程同步，上锁粒度等，同时也学习了对我来说的一些“新技术”，例如 websocket，操作转移算法等，尤其是在学习操作转移算法时，体会到了工程实现的魅力。

同时，正如前言所说，希望这个项目可以不定期迭代，为组里面试，课堂教学做出贡献。项目的功能还有许多需要完善的地方，例如可以添加聊天窗口，支持撤销，支持富文本等，这都是可以驱动改进的动力，欢迎志同道合的同学一起学习。

## 致谢

匆忙中大学四年已过，我要感谢所有帮助过我的老师，同学。尤其感谢张学辉老师带我走上 ACM 竞赛道路，让我在 ACM 竞赛中收获了知识，友谊和工作。同时，我的毕设指导老师也是张老师，让我在毕业设计之际，选择了一个吸引人而且可以提高工程能力项目，真的十分感激。另外，感谢辅导员和很多科任老师在大四实习中给予的鼎力支持，感谢 acm 的队友和班级的同学，在不经意间的谈论中给予我技术上的帮助。大学即将毕业，我将怀揣着这份感动奔赴工作当中，不断学习！

## 参考文献

- [1] 阮一峰.ES6 标准入门（第三版）[M].北京:电子工业出版社,2017.9.
- [2] 吴浩麟.深入浅出 Webpack[M].北京:电子工业出版社,2019.1.
- [3] vue.js v2 guide.介绍[EB/OL]. <https://cn.vuejs.org/v2/guide>.
- [4] Danny Coward.Java WebSocket Programming[M]. America New York:McGraw Hill Education,2013.9.
- [5] W.Richard Stevens. TCP/IP Illustrated [M]. Upper Saddle River: Addison-Wesley Professional,1993.
- [6] iview guide.介绍[EB/OL]. <https://www.iviewui.com/docs/guide/install>.
- [7] Joshua Bloch.Effective Java（第二版）[M].北京:电子工业出版社,2016.3.
- [8] jetty document.documentation[EB/OL]. <http://www.eclipse.org/jetty/documentation/>.
- [9] C.A Ellis,S.J.Gibbs.Concurrency Control in Groupware Systems[J]. ACM SIGMOD international conference,1989.
- [10] 陶辉.深入理解 Nginx:模块开发与架构解析（第 2 版）[M].北京:机械工业出版社,2016.2.