郑州轻工业学院

本科毕业设计（论文）

题　　目	基于 Node.js 的聊天室设计与实现
学生姓名	杨文豪
专业班级	通信工程 14-01
学　　号	541407040153
院　（系）	计算机与通信工程学院
指导教师（职称）	宋国林（讲师）
完成时间	2018 年 6 月 15 日

郑州轻工业学院

毕业设计（论文）任务书

题目 基于 Node.js 的聊天室设计与实现

专业 通信工程 14-01 **学号** 541407040153 **姓名** 杨文豪

主要内容、基本要求、主要参考资料等:

一、主要内容

1. 了解基于 Node.js 的聊天室的背景和意义。
2. 对基于 Node.js 的聊天室进行需求分析，提出设计方案与解决方法。
3. 设计并实现基于 Node.js 的聊天室的各功能模块。

二、基本要求

1. 设计严谨、功能完备、界面清晰流畅。
2. 性能良好、易于维护、管理功能人性化，有较强的交互性。

三、主要参考资料

Nicholas C.Zakas 著，李松峰，曹力译. JavaScript 高级程序设计[M].人民邮电出版社, 2012.

朴灵著. 深入浅出 Node.js[M]. 人民邮电出版社, 2013-12-01.

Kristina Chodorow 著，邓强，王明辉译. MongoDB 权威指南[M]. 人民邮电出版社, 2014.

Adam Freeman 著，谢廷晟，牛化成，刘美英译. HTML5 权威指南[M]. 人民邮电出版社, 2014.

完 成 期 限： 2018 年 6 月 15 日

指导教师签名： _____

专业负责人签名： _____

2018 年 1 月 5 日

基于 Node.js 的聊天室设计与实现

摘　要

基于 Node.js 开发的 web 聊天室，在客户端采用 SPA(单页应用)架构以页面无刷新式渲染来提高用户的使用体验，使用基于 Vue 的组件库 IView 以组件方式搭建前端页面，使用 Vuex 和 localstorage 配合实现数据的初始化和持久化。服务器端采用 RESTful 架构以实现前后端分离使代码结构清晰，使用基于 Node.js 的 web 框架 Koa 来为客户端提供 API 接口。数据库采用非关系型数据库 MongoDB，创建了 user、group、message、private 四张表来分别存储用户、群组、消息、私聊等数据，并使用 mongoose 对数据进行操作。

经使用压力测试工具 ab 对本地服务器进行并发测试，QPS 最大可以达到 3000 次/秒,最大可以支持 300 的并发量并且最多可以容纳 200 人同时在线。

关键词　Node.js；　聊天室；　WebSocket；　Vue；

DESIGN AND IMPLEMENTATION OF CHAT ROOM BASED ON NODE.JS

ABSTRACT

A web chat room developed based on Node.js. The client uses a SPA (single page application) architecture to improve user experience with page-less refresh rendering. Vue based component library IView is used to build front-end pages in a component manner .using Vuex and localstorage to achieve data initialization and persistence. The server adopts a RESTful architecture to achieve separation of the front and back ends to make the code structure clear. The web framework Koa based on Node.js is used to provide API interfaces for clients. The database uses MongoDB, a non-relational database, to create user, group, message, and private four tables to store data such as users, groups, messages, and private chats, and use mongoose to operate on the data.

Using the stress test tool socket-bench for concurrent testing, can support 1000 concurrents and can accommodate up to 200 people simultaneously online

KEY WORDS Node.js; chat room; WebSocket; Vue; Koa

目 录

1 绪论

1.1 项目背景与意义

据 We Are Social 和 Hootsuite 发布的 2018 全球数字报告中指出，全世界已经有 40 亿人在这个互联互通的因特网上畅游。[3]随着网络的进一步普及可以惠及更多的人，使所有人都能享受互联网所带来的便利和全新体验。Web 作为互联网中的重要组成部分也在蓬勃的发展过程中取得长足的进步，其中 HTML5 作为下一代 web 标准的出现，为 web 技术发展注入的新的生机和活力。HTML5 是制订包括 HTML、CSS、JavaScript 在内的诸多 web 技术标准的合集，相较之前的版本而言，HTML5 增加了更多语义化元素、智能化表单控件、绘画元素 Canvas、地理信息、消息通知等更加语义化或人性化的 API，大大提高了浏览器的表现力和创造力。在技术标准日趋成熟的同时，各浏览器厂商也在积极向标准靠拢和尽力提升浏览器引擎的性能。使得许多 web 应用已经拥有了不逊色桌面应用的性能表现而且还兼有免安装、随用随开的特点。由于 HTML5 WebSocket 协议的提出，Web 实时通信聊天室也可以达到类似桌面实时通信应用的效果。

1.2 研究内容和创新点

1.2.1 研究内容

研究内容包括以下几项:

(1) HTML5 中新提出的 WebSocket 协议的原理与使用。HTML5 是下一代 web 应用的标准，在包括 canvas 绘图、本地数据存储、地理信息、消息通知、动画和 WebSocket 协议等方面都制订了新的标准和规范，在仔细学习 HTML5 新标准的同时重点研究 WebSocket 协议，探究 WebSocket 协议的原理和协议建立的过程以及和 HTTP 协议的区别与联系，明晰相对于传统实时通信解决方案的优势。

(2) 对 web 实时通信解决方案的研究。从传统的轮询、长轮询、长连接到如今的 WebSocket 之间的演变历史和各自所解决的问题以及所面临的瓶颈与局限。

(3) 研究服务器端的 Node.js 和其 web 框架 koa。Node.js 从面世至今就一直以高并发、事件驱动的特点而有着广泛的应用场景，在 Node.js 基础上封装的 koa 框架则有着

低耦合、可定制、简洁轻量的优点，研究二者在当前项目场景的使用方法及优化策略。

(4) 研究非关系数据库 MongoDB 的使用方式。学习基于文档的数据模式的设计与操作。

1.2.2 创新点

系统的创新点可以归纳为以下两点：

实时通信方式采用 HTML5 WebSocket。相较于传统的 web 实时通信解决方案需要与服务器之间不断的建立连接或保持长连接来说，在 web 聊天室场景下，WebSocket 作为有状态协议相对于 HTTP 这种无状态协议有着无法比拟的优势，它不仅更加节省网络资源而且还允许服务器与客户端之间可以随时向对方发送数据。基于 WebSocket 协议的实时通信系统在显著提高反应速度和实时性的同时又节约了大量的网络资源。

使用 SPA(单页应用)的方式组织客户端模块。针对传统的多页应用造成的服务端逻辑过重、页面频繁刷新造成的用户体验下降的问题，单页应用通过以下几方面设计解决了上述问题：

(1) 控制器前置，即浏览器通过直接处理请求地址的变化，然后分发到对应的路由以呈现用户希望得到的界面。

(2) 页面模块化，将前端页面划分为一个个相对逻辑独立的功能模块，当路由改变时无须刷新整个页面而是刷新改动的部分。

(3) 数据层前置，即在浏览器维护一个实实在在的数据层，而服务器只负责提供操作数据的 API。

经过上述改变，单页应用拥有无刷新体验、前端完全组件化、API 共享、组件共享等诸多优点，一方面由于组件复用从而提高了工作效率，另一方面也因为页面无刷新而提高页面的操作流畅度。

1.3 论文组织架构

本论文由四个章节组成。

第一章 绪论。介绍了相关市场背景和研究意义以及创新点。

第二章 研究了实时通信的解决方案。细致分析了传统解决方案的实现原理和各自在实际应用中的瓶颈和局限性，介绍了 WebSocket 协议的原理和 WebSocket 在客户端

和服务端的使用方式，对比了它相对于传统解决方案的优劣。

第三章 项目开发的架构与技术选型。这一章分别介绍了客户端和服务端的架构与技术选型，细致介绍了 B/S 架构的相关概念以及其相对与 C/S 架构的优势。介绍了 SPA(单页应用)的架构方式以及其相对于传统多页应用的优势，还介绍了在项目开发中用到的包括 HTML、CSS、JavaScript、vue 及相关生态等前端开发技术的使用，还介绍了服务器端的 Node.js、koa、MongoDB 数据库、mongoose 的使用方法。

第四章 详细的说明了基于 Node.js 的聊天室的设计思路与实现方法。具体介绍了客户端的模块划分以及服务器端的路由接口设计和数据库的表结构的设计，最后对系统的各模块功能进行测试和评估，以保证应用功能的可用性。

2 实时通信解决方案研究

2.1 web 实时通信技术要点

要实现 web 实时通信主要需要克服信息的持久化和信息的传递两个问题。对于第一个问题的解决方案比较多，有很多选择的余地。例如你可以选择各种数据库包括关系型数据库如 MYSQL 和非关系型数据库如 MongoDB 或者文件和缓存来实现聊天信息的保存，可根据项目需求来酌情决定。主要的问题在于如何在消息需要被传递的时候实时的传递给接收方，这其中经历了传统技术解决方案到当前较为先进的 WebSocket 的演进过程。

2.2 传统 web 实时通信解决方案

2.2.1 指定页面全部或局部的刷新

较为初级的解决方案是通过<meta>标签内指定自动刷新的时间间隔，浏览器根据该标签定义的时间间隔自动刷新该页面。现在还有一些手机网页上采用这种方式，但是页面间歇性卡顿造成的用户体验不佳是这种方式最大的弊端。同时还有一种页面局部刷新方式原理与其类似，通过定时器函数来不断刷新内嵌隐藏的 iframe。

2.2.2 Ajax 的轮询

Ajax 技术的诞生，是 web 开发领域的一个极大变革。使 web 应用焕发了更强的生命力。通过使用 Ajax 技术我们对服务器发起的请求就可以采取异步的方式，这样就可以避免整个页面刷新。Ajax 轮询技术就是使用定时器函数在指定时间间隔重复发起 http 请求。同样地，这种方式的缺点也显而易见，在时间间隔的设置上难以把控。过长则达不到实时通信的标准，过短则加重浏览器负担，于性能不利。

2.2.3 HTTP 长连接

HTTP 长连接与 Ajax 轮询的显著不同在于相较于轮询频繁的发送 http 请求而言，HTTP 长连接只发送一次 http 请求而后一直保持此请求为开启状态从而实现周期性的数据互通。在以 webkit 为内核的现代浏览器中，在实现的过程中通过监听 ajax 请求的 redaystatechange 事件查看 readyState 的属性值来判断数据的传输状态。数据保存在请求返回的 responseText 属性上。此种方法劣势在于兼容性较差，在低版本浏览器上无法

兼容且实现过程较为繁琐复杂。

2.3 WebSocket 解决方案

2.3.1 WebSocket 简介

HTML5 作为 HTML 的最新标准由 W3C 于 2014 年 10 月制订完成，其中规定了许多的新标准。WebSocket 协议就是其中非常重要的一项。WebSocket 是一种在单个 TCP 连接上建立的全双工通信协议。可以简化客户端和服务端之间的通信过程和数据交换。WebSocket 赋予了客户端和服务端随时主动向对方推送数据的能力。通过使用 WebSocket，客户端和服务器两者之间就可以直接创建持久性的连接并进行持的续双向数据传输。

2.3.2 WebSocket 握手

在建立 WebSocket 连接的过程中是由客户端首先发起。客户端会向服务器发送一串 Base64 加密的密钥，服务器收到密钥后会将密钥和一个固定的掩码连接后然后经过 SHA-1 算法加密，然后再通过 Base64 加密后返回。如果返回的 HTTP 状态码是 101 证明 WebSocket 连接建立成功，客户端和服务器之间的通信协议也中 HTTP 切换至 WebSocket。此时通信双方都可以向对方推送数据且可以接收对方推送的数据直到一方主动断开连接。

2.3.3 客户端的 WebSocket 使用方式

客户端的 WebSocket 是由全局对象 window 提供的一个构造函数，用于构建 WebSocket 实例。当以 new 操作符调用这个构造函数时，客户端就会尝试与服务器相应端口建立连接，通过监听 WebSocket 实例的 readyState 的属性值，当值为 OPEN 时即为成功建立连接。连接成功建立后即可调用相应的发送和接收信息的 API 来实现数据的传递。

2.3.4 WebSocket 的优势

WebSocket 相对与 HTTP 协议而言有着控制开销少、实时性强的优势。原因在于 WebSocket 连接成功建立后，服务器和客户端之间发生数据交换时，对服务器到客户端的内容来说，用于协议控制的数据包头部的大小只有 2 至 10 字节和额外的 4 字节的掩码。而 HTTP 请求每次请求都会携带完整的头部。所以控制开销大大减小。而且由于

WebSocket 协议是全双工的，所以服务器可以随时主动向客户端推送数据，相对于 HTTP 请求需要等待客户端发起请求服务端才能响应，延迟明显更少。即使是和 Comet 等类似的长轮询比较，其也能在短时间内更多次地传递数据，保持连接状态。与 HTTP 不同的是 WebSocket 是一种有状态的协议，之后通信时可以省略部分状态信息。而 HTTP 请求可能需要在每个请求都携带状态信息。

3 技术架构及选型

3.1 B/S 软件架构

B/S（Browser/Server）软件架构即浏览器和服务器结构。它是随着 Internet 技术的兴起与发展，相对于 C/S 架构的一种变化或者改进的架构。在这种结构下，一个完整的系统由浏览器和服务器组成。其中浏览器负责用户界面的显示，服务器负责事务逻辑的实现，这样一来就大大减轻了作为客户端的压力。减轻了系统维护与升级所需要的成本和工作量。B/S 结构的系统由于前端运行在浏览器上，所以天然具有免安装、跨平台的特性，同时具有很强的扩展性，如需增加业务逻辑，增加页面即可。当系统需要升级时只需要升级服务器即可实现所有用户的同步更新，有着很强的共享性。

3.2 客户端技术架构

3.2.1 单页应用(SPA)

单页应用是一种通过动态重写当前页面来与用户交互而非传统的通过服务器重新生成加载整个新页面的 web 应用程序。这种模式避免了由于页面之间的频繁切换而打断用户的沉浸式体验。且由于后台服务器只需要通过接口提供数据而不需要处理页面所以大大提高了服务器的吞吐量和性能。

3.2.2 前后端分离

前后端分离不仅仅是一种开发模式更是一种 web 应用的架构模式。在开发阶段，前后端约定好数据接口，然后各自实行并行开发和测试。各自开发完成后进行联调，前端通过 http 或其他协议与服务器进行数据交互。联调完成即可部署至公网服务器供用户访问，具体关系如下图所示：

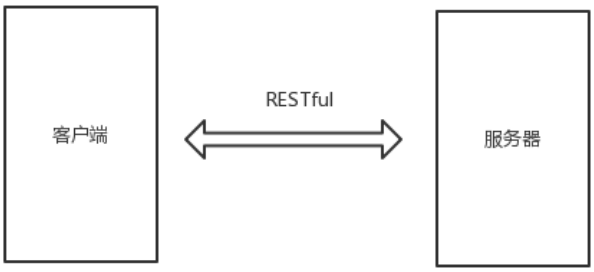

图 3- 1 B/S 架构

根据目前应用软件的发展趋势来看，多终端下用户体验的提升以及云化、微服务化的应用架构越来越需要引起厂商和软件开发者的重视。前后端分离的构架模式则可以提升开发效率和从容应对复杂多变的前端需求，还可以增强代码可维护性以达到提高用户体验的目的。

3.3 客户端技术选型

随着 web 技术的日益发展，前端工程师作为一个新的职业分工被独立出来，前端的知识结构较为庞大，在狭义的传统 web 前端开发中、主要使用 HTML、css、js 来完成 web 界面的布局、样式和逻辑行为的控制。但随着 Node.js 的发展，Node.js 已经成为前端开发的基石，基于 Node.js 的脚手架、自动化工程流等都深深改变了相关从业者的开发理念和开发体验。HTML5 标准的提出和完善及 ECMA 版本的演进使得技术更加规范化和语义化。还有 MVVM 思想的流行和诸如 VUE、React、Angular 等新兴 MVVM 框架的发明和发展，在大幅提高前端从业人员的生产力的同时也在不断地丰富从业者的技能树。

3.3.1 HTML5、CSS3、ES6 等前端基础

HTML 意为超文本标记语言，是整个网页开发的基础技术。HTML 作为一种标记语言通过使用不同的标签来完成页面内容的展示。HTML 通过 img 标签来嵌入图片、通过 form 标签来插入表单等等。HTML5 作为 HTML 的最新标准，除了在基础上增加了许多语义化元素和多类型表单之外还为下一代 web 提供了全新功能主要包括本地音视频播放、动画、地理信息、画布 Canvas、本地存储等。

CSS 是层叠样式表的简称。主要通过多种选择器选择页面元素并为其添加各种样式控制语句用来控制页面 HMTL 元素的视觉展现。CSS3 作为最新的 CSS 标准不仅提

供更先进友好的布局方式还提供诸如变换、动画、阴影、渐变等许多高阶视觉效果、为 web 网页视觉表现增光添彩。

Javascript 是一种基于对象和事件驱动的脚本语言，是一门动态类型的解释型语言。是由网景公司的 LiveScript 发展而来的。经由 ECMA 通过指定 ECMAScript 标准实现语言的规范。完整的 JavaScript 包含三个部分：

(1) ECMAScript(语言核心)，提供了语言的核心功能包括基本类型、关键字、函数、操作符、对象等内容。ECMAScript 发布了不同版本，当前常用版本为 ECMAScript2015 在以前版本的基础上对数组和对象的功能进行了扩展，加入了新的数据类型、提供了许多语法糖等。提高了编码效率的同时又使代码可读性得到一定的提升。

(2) DOM(文档对象模型)，全称为 Document Object Model，是编程语言操控 HTML 文档的标准编程接口。是抽象出来表示和处理一个 HTML 或 XML 文档的常用方法。DOM 不仅仅适用于 JavaScript 还可以用于任何编程语言，不过主要是用来和 JavaScript 进行交互。DOM 将 HTML 结构抽象化为一颗 DOM 树，通过 JavaScript 控制其隐藏或消失、添加或删除。使得页面的交互性大大的增强。

(3) BOM(浏览器对象模型)，全称为 Browser Object Model。是由浏览器提供的可以操作浏览器窗口的对象，其中顶层对象就是代表当前浏览器窗口的 window 对象，其他所有对象都是该对象的属性和方法。通过 JavaScript 与 BOM 的交互可以实现控制浏览器窗口的大小、开闭以及跳转，还可以获取浏览器页面导航对象的详细信息等功能。

3.3.2 Vue 及周边生态

3.3.2.1 Vue.js 的原理和使用方式

Vue 是一个聚焦于页面视图层的前端库，是当前非常流行的基于 mvvm 思想的前端库之一。Vue 底层基于 es6 的 defineProperty 方法和发布订阅模式来实现数据与视图的双向绑定。数据驱动视图的双向绑定使得开发人员将注意力集中于数据。同时 Vue 的组件化思想贯穿整个库。使用 Vue 开发之后所有页面都是由组件组合嵌套组成，通过 props 来定义接受外部数据，使用自定义事件来传递消息，使用 slot API 来将外部动态传入的内容和自身模版进行组合以实现组件间的结合。还具有模块化的特点，使用

webpack 和 vue-loader 对写在一个文件的模版文件、样式文件、脚本文件进行打包整合，具有高内聚、低耦合的优点。可以显著提高开发效率和代码的稳定性及可维护性。

3.3.2.2 vue-router 的原理和使用

路由是浏览器地址栏的 url 和用户所见页面的对应关系。传统的服务端路由是通过客户端请求的网址不同返回不同的页面来实现。这种方式会造成服务器压力过大造成的页面响应延迟。在单页应用中，通过 JavaScript 来处理由 url 改变带来的 UI 的切换，而不需要向服务器请求。前端路由的实现方式有两种，其一是通过监听 hashchange 事件，根据 BOM 对象的 location.hash 值改变来相应切换 UI。另一种方式是根据 HTML5 提供的 History API，该 API 上的 pushState 函数和 replaceState 函数可以修改 url 地址从而触发 UI 的改变。

vue-router 是为 Vue.js 设计的路由库，底层基于上述两种方式，根据浏览器的不同自动选择。由于项目是组件化、模块化开发，所以只需将 url 和组件对应即可。在 router 对象中将路径和组件配置为一个 key-value 形式对应起来。当 url 改变时会因为组件的切换而实现页面的切换。

3.3.2.3 Vuex 的原理和使用

Vuex 的核心概念及运作流程如下图：

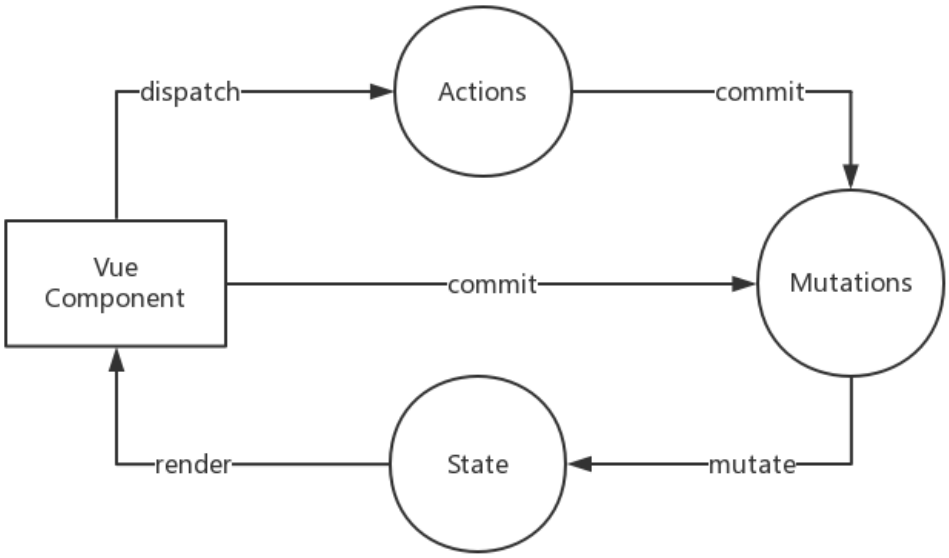

图 3- 2 Vuex 原理

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。由于 Vue 项目是以组件为基本单元构造出来的，而不同级别组件之间的通信方式如果采用 props 传递过于低效和繁琐，而 Vuex 将所有状态提升至全局，并将这个 Vuex 生成的状态对象注册到根组件中。这样一来所有组件都可以直接访问存储在 Vuex 中的状态。以 Vuex 为基础生成一个 store 对象，可以在 store 对象中定义和管理应用所需的状态、定义同步状态改变的 mutations、定义异步状态变更的 Actions。Mutations 和 actions 都是定义的函数，接受全局状态作为参数，经过逻辑处理后返回新的全局状态。具体到本项目中，store 中存储着用户信息、聊天信息、配置信息等状态。当由于用户操作需要同步改变全局状态时，即发起一个 commit 调用 store 中定义的 mutation 来完成同步状态的改变，当由于用户操作需要异步改变全局状态时，则发起一个 dispatch 调用定义在 store 中的 actions 完成异步状态的变更。组件会监听全局状态的变更完成异步更新，这样即可实现页面的刷新。这种声明式的编程方式使得逻辑更加清晰、分工更加明确。

3.3 服务器技术选型

服务器端由于业务的需要，有很多操作文件、网络、数据的场景。可以用于后端开发的由很多语言，如 java、php、python 等，我们采用基于 chromeV8 引擎的 javascript 运行时 Node.js，由于前后端语言相同，可以获取一致的开发体验。数据库当前有两种选择，关系型数据库如 mysql 和非关系型数据库如 MongoDB，非关系数据库由于高性能、易扩展的特性近年来越来越流行。我们采用 MongoDB 存储用户产生的数据，并使用基于 Node.js 的 mongoose 来操作数据库。

3.3.1 Node.js

Node.js 是基于 Google V8 引擎的 JavaScript 运行时环境。底层使用 Libuv 操作系统，使 JavaScript 支持 io、文件等只有服务器语言才有的特性，使得 JavaScript 能够同时具有 DOM 操作(浏览器)和 I/O、文写、操作数据库(服务器端)等能力。其原理图如下所示：

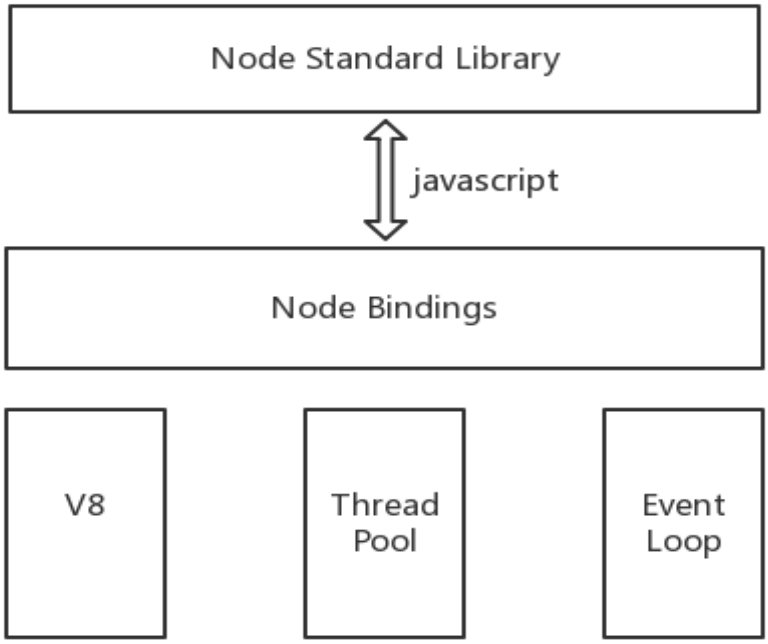

图 3- 3 node 技术架构

基本原理是基于 Chrome V8 引擎构建，由事件循环(Event Loop)分发 I/O 任务，最终工作线程将任务丢到线程池里去执行而事件循环等待执行结果然后执行相应逻辑。由于 Node.js 的事件驱动及非阻塞 I/O 设计使得 Node.js 的并发性能非常突出。

3.3.2 Node.js 的 web 框架 Koa

Koa 是基于最新 JavaScript 语法 async/await 的 Node.js 框架。相较于 Node.js 的回调写法而言，Koa 解决了因多层回调嵌套的回调金字塔问题，提高了代码的可读性和可维护性。另一方面，由于采用了中间件的设计，使得逻辑功能都在中间件中实现从而使主函数更加轻量优雅。

3.3.3 非关系型数据库 MongoDB

MongoDB 是一个高性能、开源、无模式的文档型数据库，由 C++语言编写，旨在为 web 应用提供可扩展的数据存储解决方案，是当前 NoSql 数据库中比较热门的一种。MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富、最像关系数据库的。它支持的数据结构非常松散，是类似与 json 的 bson 格式，因此可存储较复杂的数据类型。MongoDB 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查

询的绝大部分功能，而且支持对数据建立索引，在许多场景下可用于替代传统的关系型数据库或键/值存储方式[7]。

3.3.4 MongoDB 的 ORM 工具库 Mongoose

Mongoose 是 Nodejs 环境下的 MongoDB 对象模型工具，封装了 MongoDB 数据库常用的增删改查等方法，同时提供 plugin、hook、populate 等机制以灵活和方便的操控 MongoDB。Mongoose 主要包括以下内容：

3.3.4.1 Schema

Schema 使用对象的方式定义数据文档模型结构，可以在 Schema 处定义一个文档的字段及字段的数据类型，还可以使用系统预设规则或自定义规则对字段进行校验，同时可以在 Schema 上直接定义静态方法。所有由此 Schema 生成的 Model 都可以直接调用此静态方法，但是 Schema 没有直接操作数据库的能力。

3.3.4.2 Model

模型(Model)是由相应 Schema 构造生成的模型。与 Schema 不同的是 Model 具有直接与数据库交互的能力，可以在 Model 的基础上对数据库进行增删改查的操作，当以 Schema 为基础创建相应的 Model 时。需要指定 Model 的名称和相应的 Schema 的名称。

3.3.4.3 Entity

实体(Entity)是由 Model 创建的实体，当 Entity 创建成功后调用自身的 save 方法即可保存数据至数据库中。Entity 与 Model 相同都有操作数据库的能力，但表现力相比 Model 稍弱。Entity 保存后即作为真实的一条数据存在数据库中，可以在后续操作中对此条数据进行其他操作。

3.3.5 WebSocket 的服务端实现 socket.io

socket.io 是基于 WebSocket 的 C-S 实时通信库，socket.io 底层是 engine.io，这个库实现了跨平台的双向通信。engine.io 使用了 WebSocket 和 XMLHttprequest（或 JSONP） 封装了一套自己的 Socket 协议，在低版本浏览器里面使用长轮询替代 WebSocket。一个完整的 EIO Socket 包括多个 XHR 和 WebSocket 连接。

客户端：在客户端 EIO Socket 通过一个 XHR (XMLHttprequest) 握手通知服务端开始 XHR 长轮询。服务器端返回的数据里面包括一个 open 标志(数字 0 表示), 以

及一个 sid 和 upgrades 字段。sid 是本次 EIO Socket 的会话 ID，因为一次 EIO Socket 包含了多个请求，而后端又会同时连接多个 EIO Socket，sid 的作用就相当于 SESSION ID。另一个字段 upgrades，正常情况下是 ['websocket']，表示可以把连接方式从长轮询升级到 WebSocket。前端在发送第一个 XHR 的时候就开始了 XHR 长轮询，这个时候如果有收发数据的需求，是通过长轮询实现的。所谓长轮询，是指前端发送一个 request，服务端会等到有数据需要返回时再 response。前端收到 response 后马上发送下一次 request。这样就可以实现双向通信。前端收到握手的 upgrades 后，EIO 会检测浏览器是否支持 WebSocket，如果支持，就会启动一个 WebSocket 连接，然后通过这个 WebSocket 往服务器发一条内容为 probe, 类型为 ping 的数据。如果这时服务器返回了内容为 probe, 类型为 pong 的数据，前端就会把前面建立的 HTTP 长轮询停掉，后面只使用 WebSocket 通道进行收发数据。EIO Socket 生命周期内，会间隔一段时间 ping - pong 一次，用来测试网络是否正常。

服务端：服务端使用 ws 库实现 WebSocket 协议。socket.io 服务启动时，会先启动一个 ws 服务。socket.io 会监听 HTTP 服务器的 upgrade 和 request 事件。当 upgrade 事件触发时，说明可能是 WebSocket 握手，先简单校验下，然后把请求交给 ws 服务进行处理，拿到 WebSocket 对象。当 request 事件触发时，根据 url 路径判断是不是 socket.io 的 XHR 请求，拿到 res 和 res 对象。这样就可以正确接收和返回客户端数据了，具体处理过程和前端部分是对应的。

4 系统的设计与实现

本章主要介绍客户端和服务器端的设计与实现。整个系统代码存放在 chat 目录下，chat 目录下有两个子文件夹 client、server 分别用来存放客户端和服务器端的代码。具体代码结构如下图示:

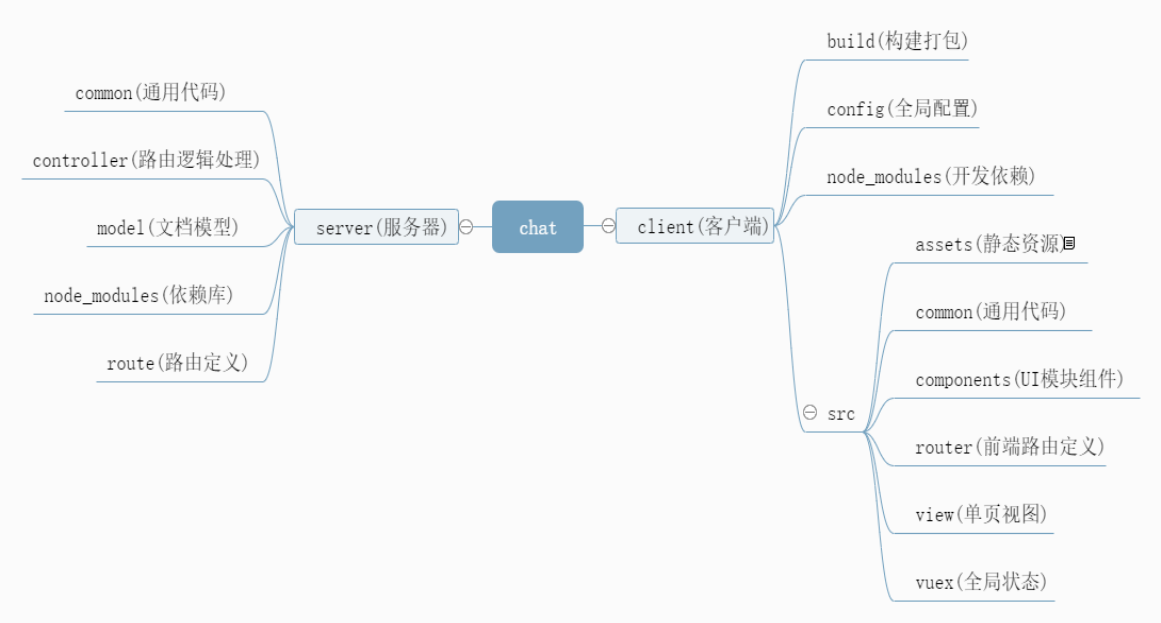

图 4-1 代码结构

4.1 客户端的设计与实现

客户端页面是由 Vue 组件构建而成，组件结构如下图示：

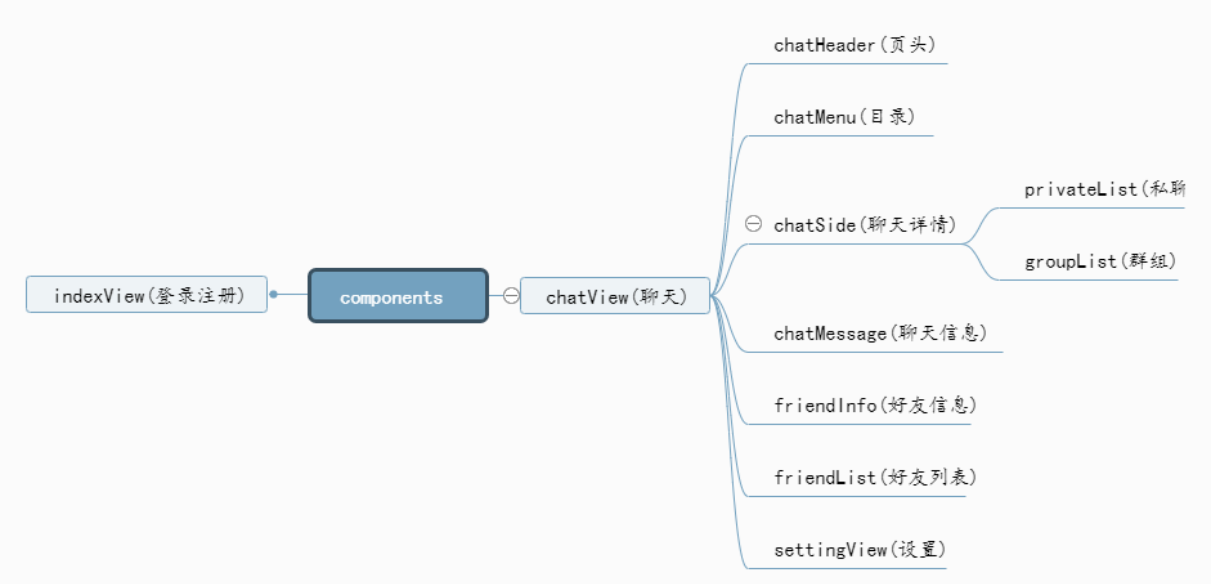

图 4-2 组件构成

客户端整体分为登录注册(indexView)、聊天室(chatView)两个组件。当用户登录或注册成功后，vue-router 会完成由登录注册组件到聊天室的切换。聊天室组件中分为聊天、好友列表、设置等三个子组件，这三个组件是通过在 vuex 中维护的 view 变量来切换，当用户点击不同的功能模块时，会相应改变的改变 vuex 中的 view 变量值，随即触发页面的渲染。

4.1.1 登录注册

登录注册逻辑：

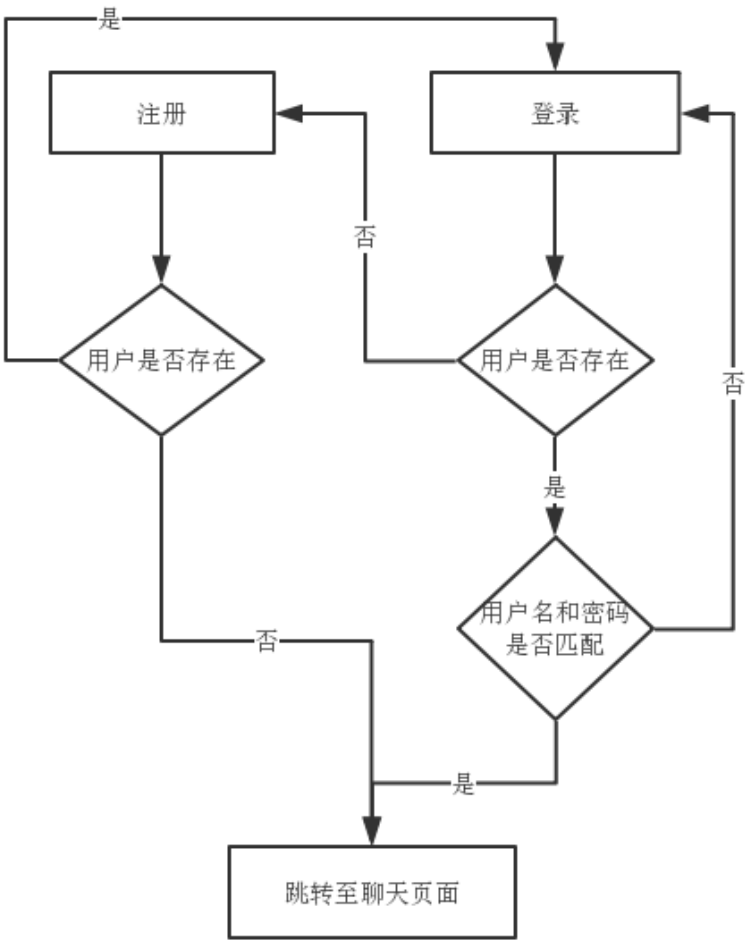

图 4- 3 登录注册逻辑流程图

登录注册页作为整个网站的 index 缺省页，是用户访问的默认跳转页。页面主要包括一个包含名称输入框、密码输入框、两个按钮的表单。当在输入框输入密码时会随着用户的输入作实时校验，当输入内容符合定义的规则方能执行下一步操作。当页面

为登录时，第一个按钮内容是登录，第二个按钮内容为切换为注册页面。当用户输入名称和密码后，点击登录按钮时，会将用户名和密码作为 login 请求的参数发送至服务器。待服务器经过处理并结果后，若登录成功则跳转至聊天页，若发生错误则给予用户一定的文字提示。当页面为注册时，第一个按钮的内容为注册，第二个按钮的内容为切换为登录页面。与登录页大致相同，当用户输入名称和密码后，点击注册则发送 register 请求至服务器，等待服务器返回值，若成功即跳转至聊天页，发生错误后根据错误类型给予用户相应的提示。

4.1.2 聊天页

聊天页是整个应用的核心功能。根据聊天的性质分为群组聊天和私人聊天。前者为一对多的聊天，后者为一对一的聊天。二者之间通过用户操作点击 tab 页完成切换。

4.1.2.1 群组聊天

当用户登录或注册成功后跳转至聊天室时，默认展示的就是聊天页面中的群组聊天。当用户注册成功时，服务器会将用户加入一个容纳所有用户的全体群组中。聊天列表会展示用户所有加入的群组，并根据群组的最后更新时间倒序排列。右侧的消息模块则会根据当前选中的群组，默认选中的为列表中的第一个群组，展示群组的基本信息和所有参与者的聊天消息。聊天消息按照时间倒序排列，并且按照消息的发送者的不同有不同的展示方式。若发送者为用户本人则消息靠右展示，而且背景色和字体颜色会有不同以作区分。

点击新建群组按钮则可弹出新建群组页面。在弹出的页面中用户可以自定义群组的名称和群公告，还可以从所有的注册用户中选择想要邀请的用户加入群组，点击确认后即会发送请求。服务器会创建一个以当前用户为创建者的群组，并邀请所有邀请的用户都加入此群组。之后该用户和所有被邀请的用户即可在此群组中进行聊天。

4.1.2.2 私人聊天

当用户点击 tab 切换至私人聊天时，聊天列表会从服务器根据当前用户 Id 获取所有的私人聊天，并根据最后聊天时间的倒序展示出来。而右侧的聊天信息组件则会根据选中的私人聊天展示聊天对象的用户名和所有的聊天信息。当点击不同的私聊后，聊天信息会相应的改变。

4.1.3 添加好友

添加好友主要有两种方式。其中一种是在群组聊天时，在查看其他用户的聊天内容时，通过将鼠标悬停至其他用户的头像上，会出现一个对方的信息弹层，可以在出现的信息弹层中点击请求添加对方为好友。如果对方不是您的好友则会向对方发送一个添加好友请求，否则会提示已是您的好友。当好友请求成功发送向对方时，若对方同意了您的好友请求，则双方同时进入属于双方的私人聊天中，并向双方的好友列表中同时添加对方，否则提示对方拒绝了您的好友请求。另一种方式是通过搜索添加。在好友搜索框中输入想要添加好友的用户名，搜索支持模糊正则搜索，即输入一个字会展示出所有名字中包含这个字符的已注册用户。您可以点击该用户以向其发送添加好友请求，后续过程与第一种方式相同。

4.1.3 好友列表

当点击左侧目录的好友列表时，将会改变 vuex 中维护的 view 值为 friends。页面即会切换至为好友列表页。当页面完成切换时，会从服务器拉取当前用户的所有好友，并以列表的方式呈现包含该好友的名称、个性签名的简略信息。当点击其中一个好友时，则会在右侧显示出该用户包含注册时间、最后登录时间在内的详细信息。还可以点击开始聊天进入聊天页中的属于用户和该好友之间的私人聊天。

4.1.4 设置及个人信息修改

当点击左侧目录的设置时，会将 vuex 中的 view 值变为 setting，页面即会切换为为设置页。在设置中可以修改用户的包括名称、个人签名的个人信息。当用户将将修改后的信息提交至服务器，若修改成功后会更新当前用户信息。设置中还可以通过开关来设置新消息提醒，新消息提醒默认为开启状态，当前用户接收到新消息时会调用 HTML5 的 Notification API 来对用户发送提醒。

4.2 服务端的设计与实现

服务端使用 koa 框架基于 RESTful 架构实现，后端只提供一系列的接口供客户端调用。

4.2.1 RESTful 架构

RESTful 全称为 Representation State Transfer，意为表现层转化，这里的表现层其

实是”资源的表现层”。资源指的是像文本、图片、音频、视频等存储在网络上的信息实体，可以使用 URI(统一资源标识符)指向它。通过访问这个资源对应的 URI 即可获取相应资源。资源作为一种信息实体可以有多种外在表现形式，例如文本可以以 txt 的格式呈现，也可以使用 HTML、XML、JSON 等格式表现。资源具体呈现出来的形式就叫做表现层。访问一个网站的实质是客户端和服务器的一个交互过程，在这个过程中涉及到数据和状态的变化。作为这二者之间建立联系的桥梁的 HTTP 协议本身是一个无状态协议，这意味着所有的状态都保存在服务器中。如果客户端想要改变数据的状态则需要以某种方式让服务器发生状态转化来达成此目的。这种转化是发生在表现层上所以称作表现层转化，这种方式就是 HTTP 协议中表示操作方式的动词: GET、POST、PUT、DELETE，它们分别对应着客户端对服务器的四种基本操作: GET 用来获取资源、POST 用来新建资源(也可用来更新资源)、PUT 用来更新资源、DELETE 用来删除资源。所以当客户端完成数据状态的转化是通过这四个 HTTP 动词调用后台接口来实现的即为 RESTful 架构。

4.2.2 socket.io 接入

当 koa 服务器启动后，将 socket.io 附着在 koa 服务器上随之启动。当客户端请求此服务器时，即会建立 WebSocket 连接。客户端和服务器端就可以通过触发和相应时间来完成数据的传递和变更，服务器监听的事件如下图示：

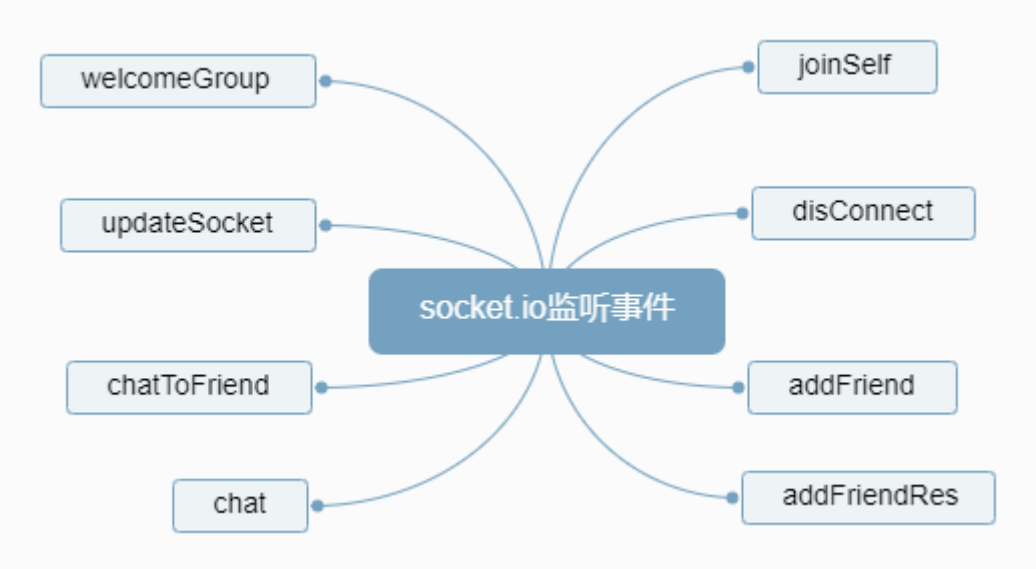

图 4- 4 socket.io 监听事件

(1) joinSelf 事件，当用户成功注册或登录时、用户以登录状态页面刷新时都会触发此

事件，服务器响应此事件后会将该用户添加至名称为该用户 id 的 group 中，如此后续与此用户的通信只需发往此 group 中，该用户即可收到信息。

(2) disconnect 事件，用户主动关闭此应用时触发，服务器监听此事件后会将当前在线人数减一。

(3) addFriend 事件，用户添加好友时触发并携带目标用户的 id，服务器响应此事件后根据用户 id 向响应用户客户端触发事件。

(4) addFriendRes 事件，当用户收到来自其他用户的好友请求时，处理过后触发此事件，并携带参数，同意添加好友为 true，拒绝为 false。当服务器响应此事件后，会根据参数做出不同的处理，若参数 true 则将双方 id 同时加入对方的好友列表中，并创建一条私人聊天数据存入数据库中，然后向客户端双方同时发送一个成功事件。客户端双方同时进入属于双方的私人聊天中。若参数为 false 则向好友请求发送方触发一个失败事件。

(5) chat 事件，当用户发送一条消息时即触发此事件并将聊天类型和聊天内容作为参数一起传递，服务器监听此事件后根据参数中的聊天类型和聊天内容做出处理，若为群组聊天，则向查找相应群组并向群组中的所有用户群发此消息，然后将此消息加如该群组的消息列表中。若为私人聊天，则向目标用户发送这条消息，并将该条信息保存到相应的私人聊天消息列表中。

(6) chatToFriend 事件，当用户向指定好友发起聊天时触发此事件，当服务器响应此事件时即向该目标用户客户端触发好友聊天事件。

(7) welcomeGroup 事件，当用户创建一个群组后触发此事件，并将创建群组的相关信息作为参数传递。服务器响应此事件后，会将所有创建群组中邀请的用户都加入此新群组的成员列表，并将此新群组加入所有被邀请用户的群组列表中。

4.2.3 路由设计

服务器路由设计是为每个来自客户端的 HTTP 请求都指定一个具体的经过逻辑处理后的结果。主要的路由分为 group、user、login、register、friends、private 几个部分，所有路由如下：

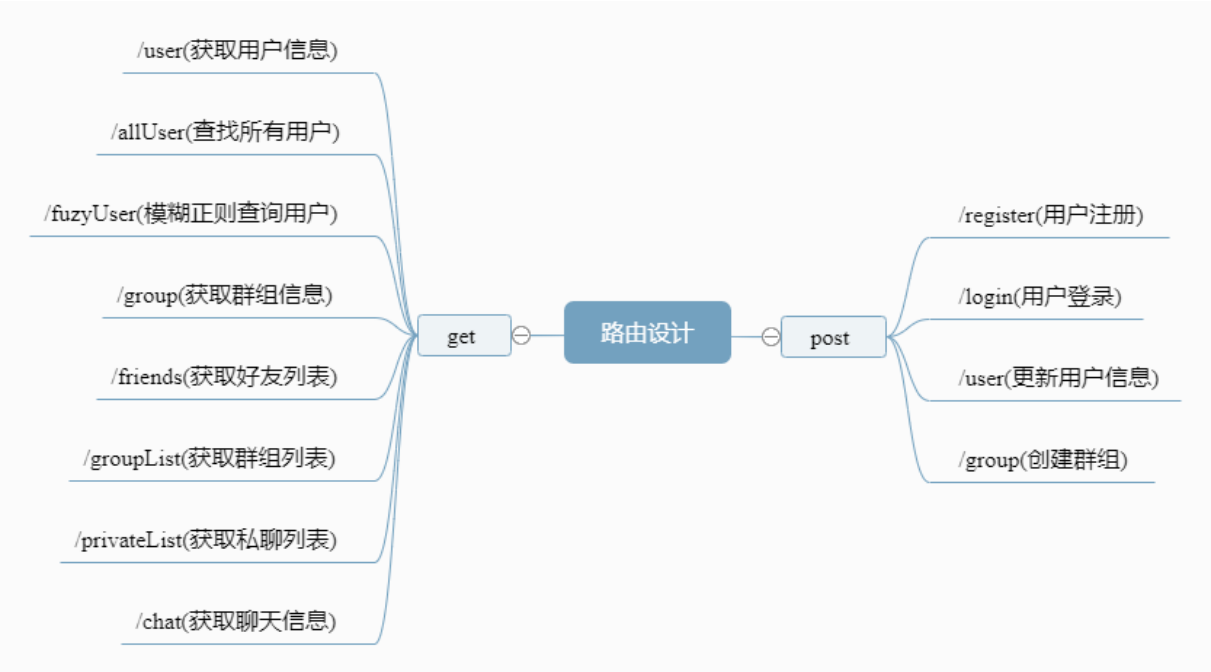

图 4- 5 路由设计

(1) 当客户端以 POST 方法请求 register 接口，服务器会从请求携带的参数中解析出用户名和密码。然后在数据库中查找无重复用户后再以用户名和密码为基础生成一个新用户存在数据库中，同时将此用户加入服务器启动时创建的包含所有注册用户的全体群中。这一系列的操作正确无误后向客户端返回一个返回值。客户端判断返回值后从注册界面跳转至聊天室页面。

(2) 当客户端以 POST 方法请求 login 接口时即为执行登录过程。服务器从请求携带的参数中解析出用户名和密码，然后在数据库查找此用户，当查找成功后返回给客户端一个成功返回值，否则返回该用户没有注册的文字提示。

(3) 当客户端以 POST 方法请求 group 接口时即为创建新群组。服务器从请求中携带的参数中解析出新建一个群组所需的参数，然后根据参数创建一个新的群组保存至数据库中。

(4) 当客户端以 POST 方法请求 user 接口即为更新用户信息。从请求参数中解析出需要更新的参数，然后更新数据库中对应用户的相应参数。

(5) 当客户端以 GET 方法请求 user、group、private、friends 等接口时即为获取用户、群组、私人聊天、好友等操作。从数据库查询出相应数据，然后返回相应的数据至客户端。

4.2.4 数据库设计

Populate(外键引用)是数据库设计中一个关键的知识点，通过 populate 可以在不同的表结构之间建立互相引用，当一个表结构中保存的字段内容是其他表时，不需要保存另一张表的所有内容，只需要将对应表的 ObjectId 存入该字段，查询时只需链式调用 populate 相应字段即可获取相应的详细数据。如此一来，大大降低了数据库的冗余度，提高了数据库查询和保存的性能，同时精简了代码。

根据应用本身的数据处理需求，数据库中共创建了 user、group、message、private 四张表，分别定义了用户、群组、消息、私人聊天的基本信息：

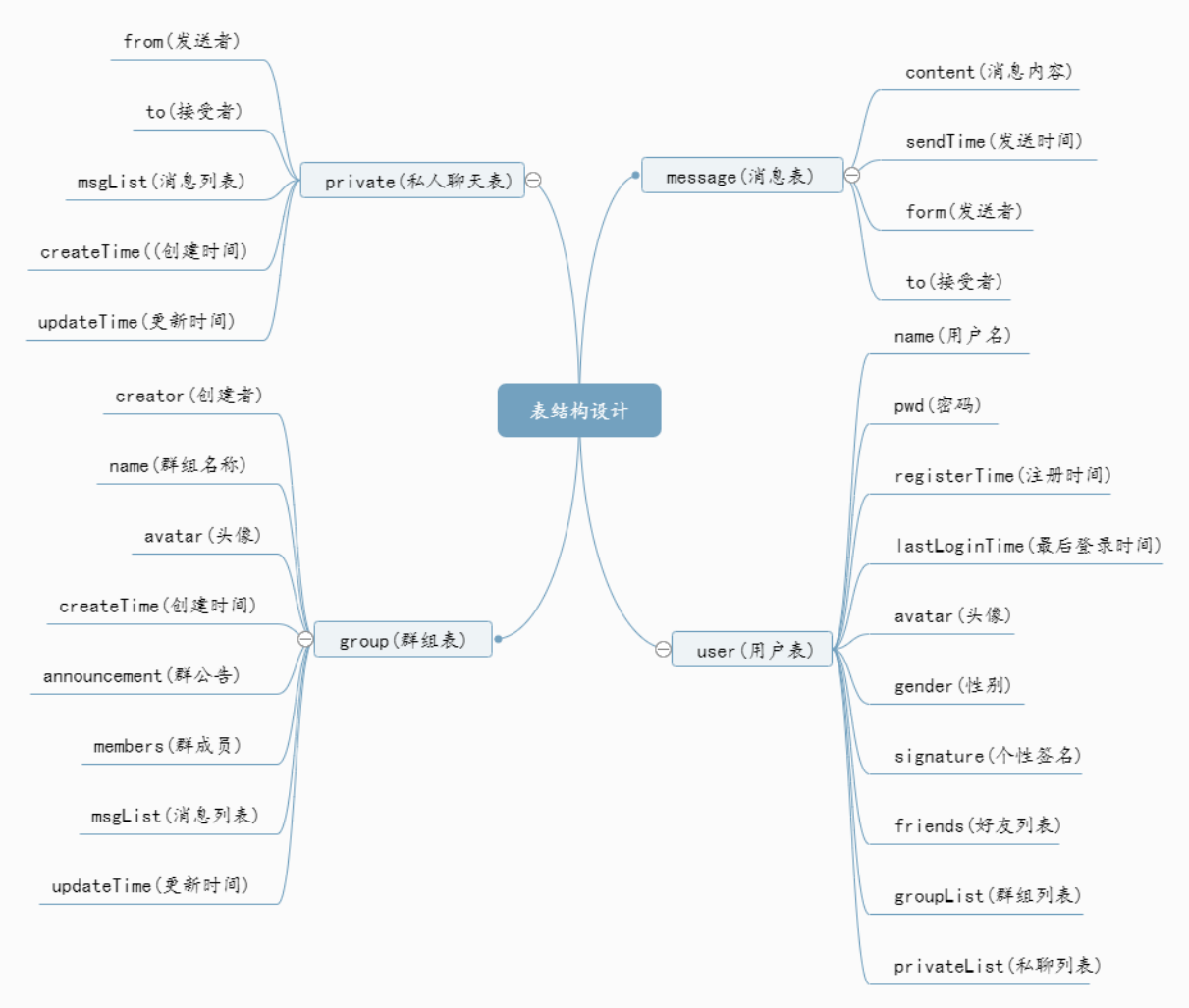

图 4- 6 表结构设计

(1) user(用户表)中主要定义了用户的用户名、密码、头像、注册时间、最后登录时间、好友列表、参与的群组列表、私人聊天列表等信息的类型和默认值。其中对用户名和密码做了必填限制，当接口处理逻辑中新建用户时没有提供确切信息保存会

触发失败。其中好友列表通过 populate 连接至 user 表，当存入用户的好友时只需存入代表该好友的用户 ID，当查询时根据此 id 获取该用户的确切信息。相应的群组列表通过 populate 连接至 group 表、私人聊天通过 populate 连接至 private 表。

(2) group(群组表)主要定义了一个群组的创建者、头像、群公告、创建时间、成员列表、消息列表。其中成员列表和消息列表分别通过 populate 连接至 user 表和 message 表。

(3) message(消息表)主要定义了消息的收发双方、消息内容、发送时间。其中发送方通过 populate 连接至 user 表。收取信息的用户根据群组聊天和私人聊天的区别分别从 group 表和 user 表中查询相应数据。

(4) private(私人聊天)主要定义了该私人聊天的成员和创建时间，收发双方都通过 populate 连接至 user 表中。

4.3 系统测试与评估

根据上述章节描述的原理及实现方式完成的项目进行测试，由于代码的实现较为透明，因此只对项目的设计功能点进行功能测试和评估。

4.3.1 测试方法

为验证系统是否符合事先设计的功能点，故采用真实的使用场景来进行模拟测试，并跟踪记录其中的交互过程并对交互效果做出截图。对比真实情况与设计之间的差异，最终决定是否满足设计需求。

4.3.2 测试流程

整个测试的流程，包括登录注册、群组聊天、私人聊天、新建群组、添加好友、好友列表、信息设置。

测试对象：基于 Node.js 的聊天室的相关功能。

测试目的：测试项目是否满足设计需求。

测试环境：chrome 浏览器 Edge 浏览器。

测试任务：

1) 用户的登录与注册：

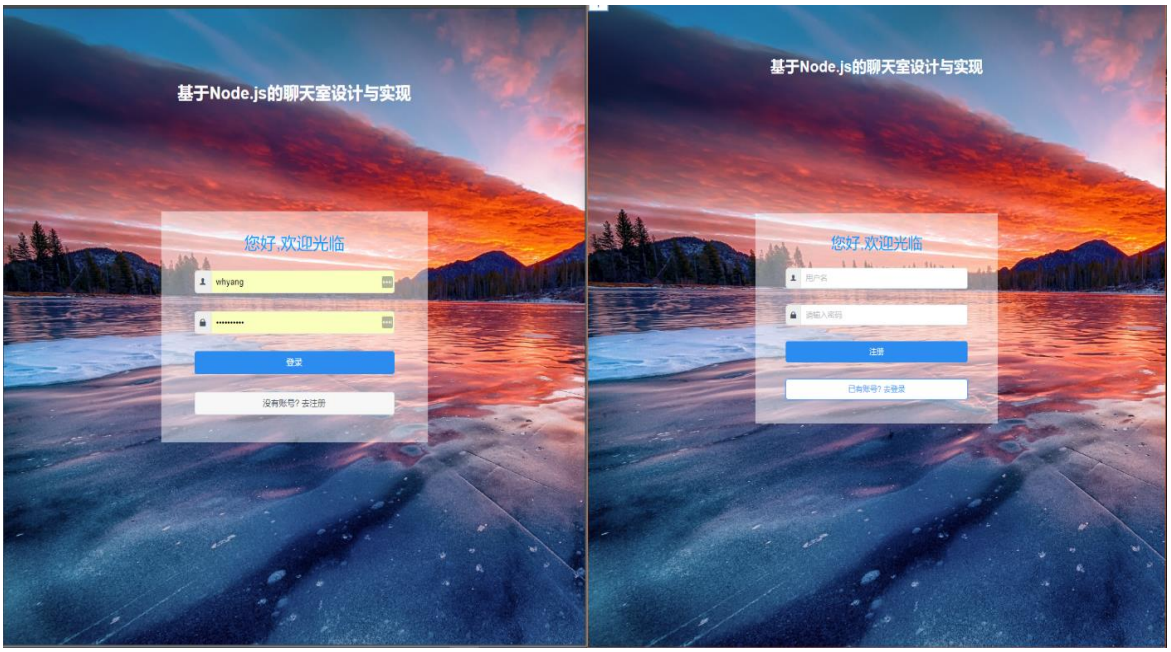

图 4- 7 登录注册页面展示

2) 用户的全体群组聊天：

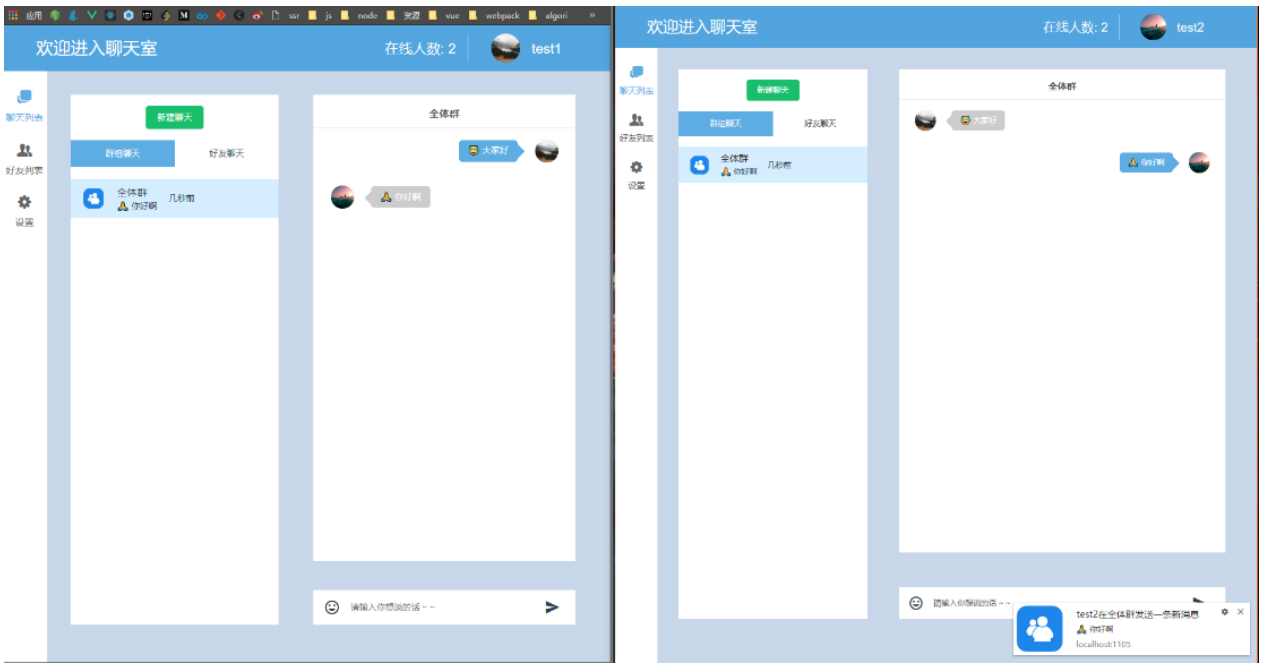

图 4- 8 群组聊天展示

3) 用户添加其他用户为好友：

图 4- 9 添加好友页面展示

4) 用户的私人聊天：

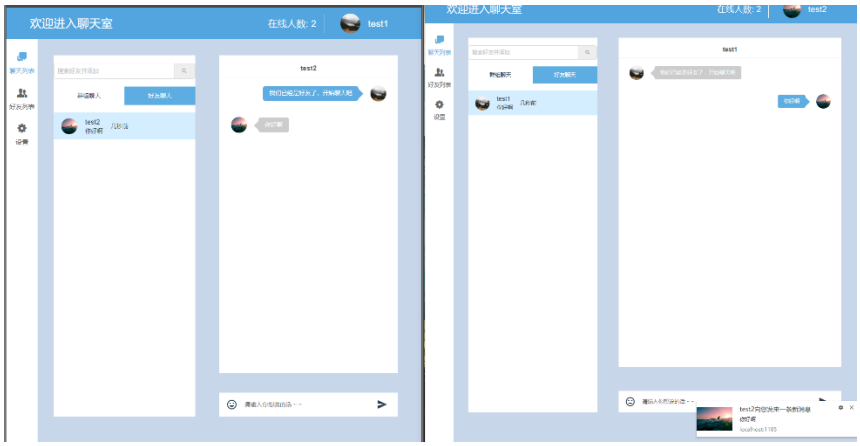

图 4- 10 私人聊天页面展示

5) 用户新建群组：

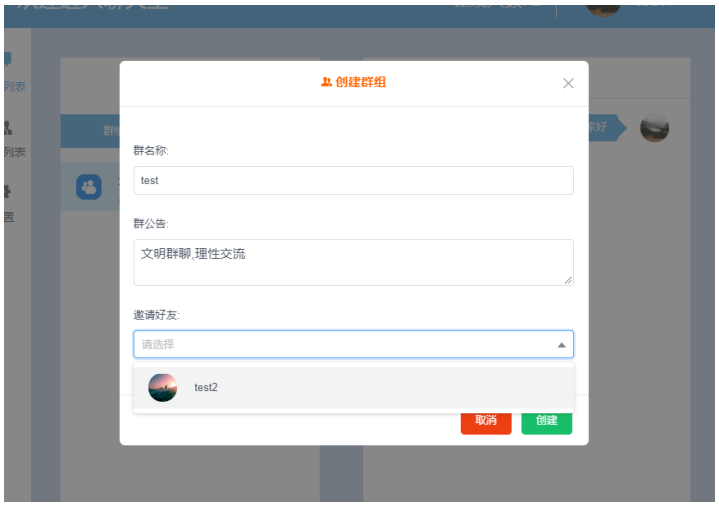

图 4- 11 新建群组

6) 用户的好友列表：

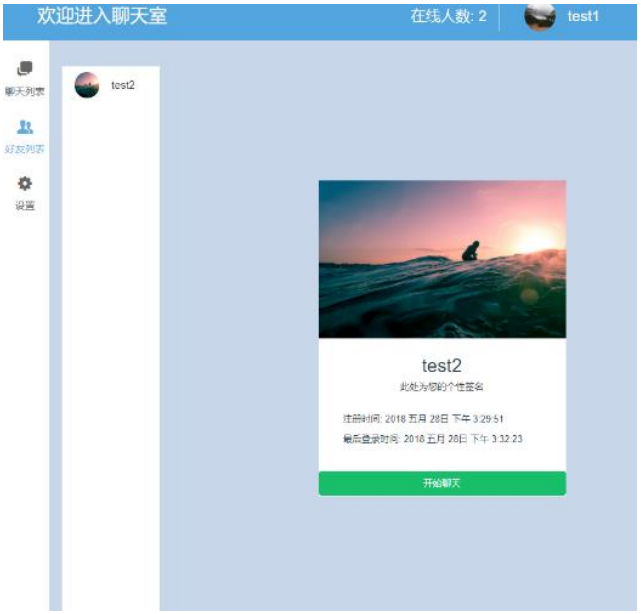

图 4- 12 好友列表页面展示

7）用户的个人信息更新及全局设置：

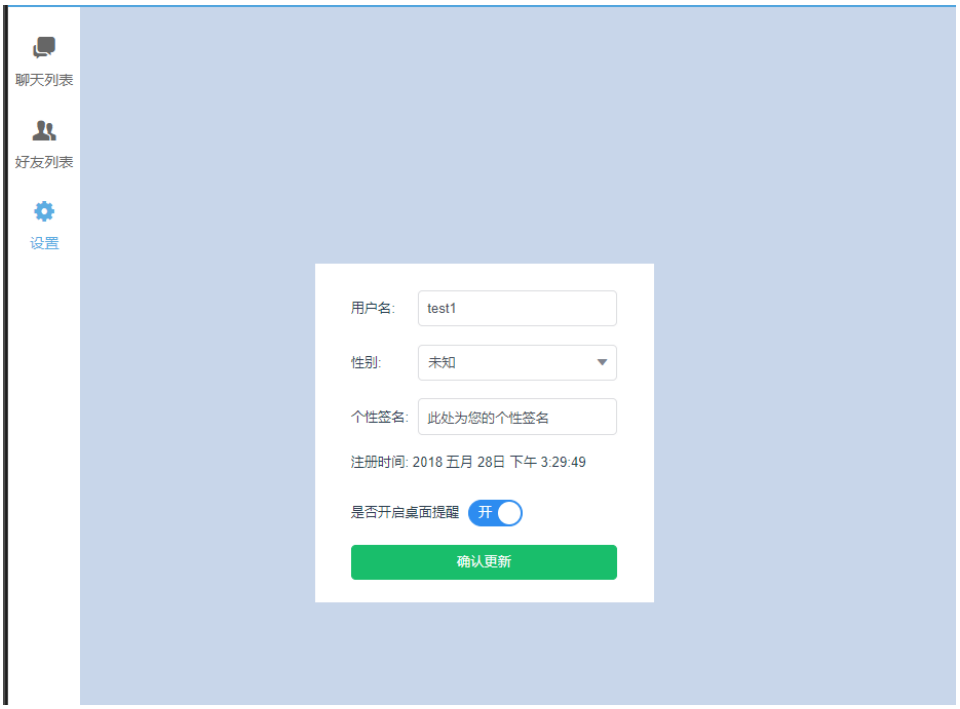

图 4-13 用户信息更新页面展示

测试时间：约半小时。

记录测试结果：对测试过程中关键的交互效果及展现形式进行截图保存。

4.3.3 测试结果评估

由上述流程分析及截图可以得知，与预期设计基本吻合，基本满足了设计需求。聊天室的各部分均可正常使用，测试基本通过，达到了我们的测试目的。经使用网络测试工具对本地服务器进行压力测试时，最大 QPS 可以达到 3000 次/秒，并发数可以达到 300，能同时容纳 200 人在线。

由于个人能力和时间原因，还有很多不足之处和可以优化的点。在 UI 展示上可以更加精细化、人性化。性能上也有很大的提升空间，并发数和同时在线人数，以及消息的延迟时间都还有进一步优化的点。功能上也可以进行拓展，可以向市场上的成熟产品学习，增加更多功能。如此种种，日后慢慢完善。

结束语

根据最新的前端技术编写出的应用依托于现代浏览器提供的更人性化的 API，可以为用户提供更加流畅的使用体验。基于这样的背景下，对当前热门的 vue 框架和 Node.js 和 websocket 的服务器实现 socket.io 进行了研究，并基于以上技术开发完成了基于 Node.js 的聊天室系统。主要工作有：

(1) 仔细研究了 WebSocket 的通信原理以及和传统实时通信的解决方案之间的优劣。

(2) 认真学习了 Node.js 及其 web 框架 koa、MongoDB 数据库及其驱动 mongoose、socket.io、vue 及其周边生态的使用方式及基本原理。

(3) 设计并实现了基于 Node.js 的聊天室系统。设计并实现了该系统的 UI 展示和逻辑功能。设计并实现了用户的登录注册、群聊、私聊、创建群组、直接或搜索添加好友、更新用户信息等功能。最后通过测试各模块的正常使用，确认了方案的可行性。本文从工程实践的角度探索了基于 Node.js 的网络聊天室的可行性。但由于经验不足，系统还存在许多可以优化的点。可以从以下几点进行提升：聊天功能的设计相对简单、支持的信息的格式单一。后续可以在此基础上添加富文本编辑、文件传输、丰富的表情、文件传输等功能。

在性能上还有一定的提升空间，后续可以不断优化包括代码逻辑、数据库设计等方面，降低延迟和提高性能以提升用户体验。

致　谢

本论文是在我的导师宋国林老师的指导下完成的。导师渊博的专业学识、严谨的治学精神和诲人不倦的高尚师德都对我产生了深远的影响。在完成毕业设计的过程中，从课题的选择到项目的最终完成，宋老师都始终给予我耐心且悉心的指导。提出了许多建设性的意见和建议使我深受启发，在此谨向宋国林老师致以我最诚挚的谢意。

此外，本论文的顺利完成还离不开实习单位校外导师的指导和帮助，非常感谢实习期间的同事和经理，帮助我克服了很多技术上的难点。

最后我要感谢在这大学四年中陪伴在我身边的父母、朋友和同学们，感谢他们对我的支持与鼓励、理解与照顾。

参考文献

[1] 于春娜. 基于 HTML5 WebSocket 的智能手机聊天室开发[D].北京邮电大学,2014.

[2] [美] Nicholas C.Zakas 著,李松峰,曹力译. JavaScript 高级程序设计[M]. 人民邮电出版社, 2012.

[3] 2018 全球数字报告[EB/OL], https://www.cnbeta.com/articles/tech/694251.htm.

[4] RFC 6455 - The WebSocket Protocol[EB/OL], https://tools.ietf.org/html/rfc6455.

[5] WebSocket 教程[EB/OL], http://www.ruanyifeng.com/blog/2017/05/websocket.html.

[6] 朴灵著. 深入浅出 Node.js[M]. 人民邮电出版社 2013-12-01.

[7] Kristina Chodorow 著, 邓强, 王明辉译. MongoDB 权威指南. 人民邮电出版社, 2014.

[8] Mongoose ODM v5.1.3[EB/OL], http://mongoosejs.com/.

[9] Vue.js[EB/OL], https://cn.vuejs.org/index.html.

[10] Vuex[EB/OL], https://vuex.vuejs.org/.

[11] Vue Router[EB/OL], https://router.vuejs.org/.

[12] MongoDB 简介[EB/OL], https://www.mongodb.com/cn.

[13] Socket.IO[EB/OL], https://socket.io/.

[14] iView[EB/OL], https://www.iviewui.com/.

[15] axios[EB/OL], https://github.com/axios/axios.

[16] Adam Freeman 著, 谢廷晟, 牛化成, 刘美英译. HTML5 权威指南[M]. 人民邮电出版社, 2014.

[17] websocket 学习(一) [EB/OL], https://blog.csdn.net/binglan520/article/details/55188756.

[18] David Hows, Peter Membrey, Eelco Plugge 等著, 周连科译. MongoDB 大数据处理权威指南[M]. 清华大学出版社, 2017-03.

[19] Azat Mardan. Real-Time Apps with WebSocket, Socket.IO, and DerbyJS[M].Apress:2014-06-15.

[20] Azat Mardan. Socket.IO and Express.js[M].Apress:2014-06-15.

[21] Cory Gackenheimer. Understanding Node.js[M].Apress:2013-06-15.