目 录

[摘要 I](#_Toc514792168)

[ABSTRACT II](#_Toc514792169)

[1.绪论 1](#_Toc514792170)

[1.1项目背景与意义 1](#_Toc514792171)

[1.2研究内容和创新点 1](#_Toc514792172)

[1.3 论文组织架构 1](#_Toc514792173)

[2. 实时通信解决方案研究 2](#_Toc514792174)

[2.1 web实时通信技术要点 2](#_Toc514792175)

[2.2 传统web实时通信解决方案 2](#_Toc514792176)

[2.2.1 指定页面全部或局部的刷新 2](#_Toc514792177)

[2.2.2 Ajax的轮询 2](#_Toc514792178)

[2.2.3 HTTP长连接 3](#_Toc514792179)

[2.3 WebSocket解决方案 3](#_Toc514792180)

[2.3.1 WebSocket简介 3](#_Toc514792181)

[2.3.2 WebSocket握手 4](#_Toc514792182)

[2.3.3 客户端的WebSocket使用方式 5](#_Toc514792183)

[2.3.4 WebSocket的优势 6](#_Toc514792184)

[3技术架构及选型 7](#_Toc514792185)

[3.1 B/S软件架构 7](#_Toc514792186)

[3.2 客户端技术架构 7](#_Toc514792187)

[3.2.1 单页应用(SPA) 7](#_Toc514792188)

[3.2.2 前后端分离 7](#_Toc514792189)

[3.3 客户端技术选型 8](#_Toc514792190)

[3.3.1 HTML5、CSS3、ES6等前端基础 9](#_Toc514792191)

[3.3.2 Vue及周边生态 11](#_Toc514792192)

[3.3 服务端架构及技术选型 11](#_Toc514792193)

[3.3.1 Node.js 11](#_Toc514792194)

[3.3.3非关系型数据库MongoDB 12](#_Toc514792195)

[3.3.4 MongoDB的ORM工具库Mongoose 12](#_Toc514792196)

[3.3.5 WebSocket的服务端实现socket.io 12](#_Toc514792197)

[4 系统的设计与实现 13](#_Toc514792198)

[4.1 客户端的设计与实现 13](#_Toc514792199)

[4.1.1 登录注册 13](#_Toc514792200)

[4.1.2 群组聊天和私人聊天 13](#_Toc514792201)

[4.1.3 好友列表 13](#_Toc514792202)

[4.1.4 设置及个人信息修改 13](#_Toc514792203)

[4.2 服务端的设计与实现 13](#_Toc514792204)

[4.2.1 路由设计 13](#_Toc514792205)

[4.2.2 数据库设计 13](#_Toc514792206)

[5 总结与展望 14](#_Toc514792207)

[5.1 对本文工作的总结 14](#_Toc514792208)

[5.2 对未来工作的展望 14](#_Toc514792209)

[结束语 15](#_Toc514792210)

[致 谢 16](#_Toc514792211)

[参考文献 17](#_Toc514792212)

# 摘要

# ABSTRACT

# 1.绪论

## 1.1项目背景与意义

随着互联网的普及

## 1.2研究内容和创新点

## 1.3 论文组织架构

# 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实时通信主要需要客服两个问题:

* 信息的持久化
* 信息的传递

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

最初为了更新客户端的数据信息采用的方式是通过<meta>内指定自动刷新的时间间隔，这样浏览器会在该时间间隔周期内自动刷新该页面。这是最初的解决更新客户端信息的方式，是最原始的同时也是最蹩脚的方式。现在一些手机网页上还有采用这种方式，这种用户体验让人发指，缺点显而易见。当然还有一种实现原理一样的局部刷新方式，就是内嵌隐藏的iframe来不断刷新。早期很多就是采用的这种方式。

### 2.2.2 Ajax的轮询

Ajax技术的诞生，是一个很大的变革,给web应用注入了新的生命力。就是由于Ajax技术使得我们可以采用异步的方式对服务器发起请求，这对用户体验绝对是一个相当大的改善。Ajax轮询就是结合javascript的setInterval函数在指定时间间隔重复发起http请求。但是这种方式的缺点也是有一定的弊端也是显而易见的,就是时间间隔的设置上,过长则达不到实时通信的标准,果断则加重浏览器负担于性能不利

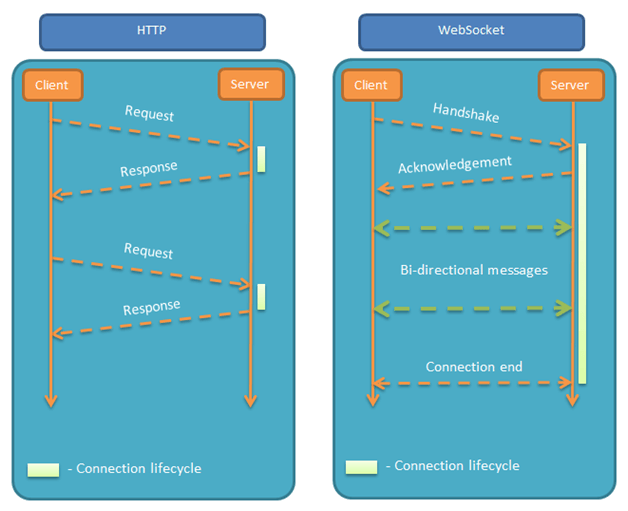
### 2.2.3 HTTP长连接

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事件来查看readyState的属性值来判断数据的传输状态,数据保存在请求的responseText属性上，读取此属性上的数据即可实现HTTP长连接,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月完成标准制订,其中规定的许多新标准引发了许多开发者的关注,其中的WebSocket协议更是吸引眼球,WebSocket是一种在单个TCP连接上的全双工通信协议使得客户端和服务端的通信和数据交换变得简单,WebSocket允许客户端和服务端主动向对方推送数据,在WebSocket API中,浏览器和服务器只需要完成一次握手,两者之间就可以直接创建持久性的连接并持续进行双向数据传输,相较于传统HTTP通信方式见下图



### 2.3.2 WebSocket握手

一个典型的WebSocket请求报文如下所示

GET /chat HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Origin: http://example.com

在这段类似HTTP协议的握手请求中，多了几个东西。

Upgrade: websocket

Connection: Upgrade

此段报文为Websocket的核心,可以告知服务器通信协议需由HTTP升级为WebSocket

Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

Sec-WebSocket-Key 是一个Base64 encode的值，这个是浏览器随机生成的为了向服务器鉴权已确认是否是WebSocket连接。Sec\_WebSocket-Protocol 是一个用户定义的字符串，用来区分同URL下，不同的服务所需要的协议。Sec-WebSocket-Version 是告诉服务器所使用的Websocket Draft（协议版本）,服务器端处理请求成功后会返回一个相应报文

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=

Sec-WebSocket-Protocol: chat

以上报文会通知浏览器升级协议为WebSocket,并在当前套接字上应用此协议,与请求报文相呼应的是Sec-WebSocket-Accept是服务端根据客户端的Sec-WebSocket-Key所生成的加密字符串。Sec-WebSocket-Protocol表示选定的子协议。客户端在校验成功后会成功建立Websocket连接,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

WebSocket对象作为一个构造函数,用于构建WebSocket实例

const ws = new WebSocket(“ws://localhost:8080”)

执行上述语句后客户端就会与服务器进行连接,通过监听WebSocket实例的readyState的属性值当其为OPEN时调用相应的回调函数即可进行通信

ws.onopen = e => {

ws.send(“data”)

}

上述代码即为WebSocket成功建立连接时的回调函数,可在函数内部书写成功建立连接后需要执行的逻辑。

### 2.3.4 WebSocket的优势

较少的控制开销。在连接创建后，服务器和客户端之间交换数据时，用于协议控制的数据包头部相对较小。在不包含扩展的情况下，对于服务器到客户端的内容，此头部大小只有2至10[字节](https://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)（和数据包长度有关）；对于客户端到服务器的内容，此头部还需要加上额外的4字节的[掩码](https://zh.wikipedia.org/wiki/%E6%8E%A9%E7%A0%81" \o "掩码)。相对于HTTP请求每次都要携带完整的头部，此项开销显著减少了。

更强的实时性。由于协议是全双工的，所以服务器可以随时主动给客户端下发数据。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](https://zh.wikipedia.org/w/index.php?title=%E9%95%BF%E8%BD%AE%E8%AF%A2&action=edit&redlink=1" \o "长轮询（页面不存在）)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需要先创建连接，这就使得其成为一种有状态的协议，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（如身份认证等）。更好的二进制支持。WebSocket定义了[二进制](https://zh.wikipedia.org/wiki/%E4%BA%8C%E8%BF%9B%E5%88%B6" \o "二进制)帧，相对HTTP，可以更轻松地处理二进制内容。可以支持扩展。WebSocket定义了扩展，用户可以扩展协议、实现部分自定义的子协议。如部分浏览器支持[压缩](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%8E%8B%E7%BC%A9)等。更好的压缩效果。相对于[HTTP压缩](https://zh.wikipedia.org/wiki/HTTP%E5%8E%8B%E7%BC%A9)，WebSocket在适当的扩展支持下，可以沿用之前内容的[上下文](https://zh.wikipedia.org/wiki/%E4%B8%8A%E4%B8%8B%E6%96%87)，在传递类似的数据时，可以显著地提高压缩率。

# 3技术架构及选型

## 3.1 B/S软件架构

B/S（Browser/Server）结构即浏览器和服务器结构。它是随着Internet技术的兴起，对C/S结构的一种变化或者改进的结构。在这种结构下，用户工作界面是通过WWW浏览器来实现，极少部分事务逻辑在前端实现，但是主要事务逻辑在服务器端实现，形成所谓三层3-tier结构。这样就大大简化了客户端电脑载荷，减轻了系统维护与升级的成本和工作量，降低了用户的总体成本。B/S结构可以看作是瘦客户端，只是把显示的较少的逻辑交给了Web浏览器，事务逻辑数据处理在放在了Server端，这样就避免了庞大的胖客户端，减少了客户端的压力。B/S结构的系统无须特别安装，只有Web浏览器即可。当然AJAX\Flex等等的普遍使用也有富客户端的发展方向。B/S相对于C/S架构有着许多优势如无需安装即客户端不需要安装有浏览器即可。分布性特点，可以随时随地进行查询、浏览等业务处理。业务扩展便捷，通过增加页面即可增加服务器功能。升级维护便捷，无需升级多个客户端，升级服务器即可，就可以实现所有用户的同步更新。共享性强特点，可以直接放在广域网上，通过一定的权限控制实现多客户访问的目的，交互性较强。

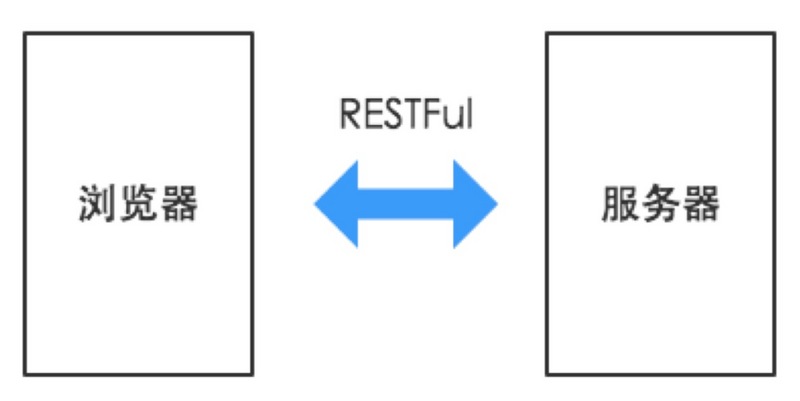
## 3.2 客户端技术架构

### 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加载整个新页面的web应用程序。这种模式避免了页面之间频繁切换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



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和云化、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以及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。但实际上随着Node.js的发展及HTML5标准的提出以及ECMA版本的演进。伴随着MVVM思想的大行其道以及以VUE、React、Angular为代表的MVVM框架的流行。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。下图展示了web前端开发所需要了解的技能



### 3.3.1 HTML5、CSS3、ES6等前端基础

HTML是一种用于创建[网页](https://zh.wikipedia.org/wiki/%E7%BD%91%E9%A1%B5" \o "网页)的标准[标记语言](https://zh.wikipedia.org/wiki/%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80)。HTML是一种基础技术，常与[CSS](https://zh.wikipedia.org/wiki/CSS" \o "CSS)、[JavaScript](https://zh.wikipedia.org/wiki/JavaScript" \o "JavaScript)一起被众多网站用于设计令人赏心悦目的网页、网页应用程序以及移动应用程序的用户界面。[网页浏览器](https://zh.wikipedia.org/wiki/%E7%BD%91%E9%A1%B5%E6%B5%8F%E8%A7%88%E5%99%A8" \o "网页浏览器)可以读取HTML文件，并将其渲染成可视化网页。HTML描述了一个网站的结构语义随着线索的呈现，使之成为一种标记语言而非[编程语言](https://zh.wikipedia.org/wiki/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80" \o "编程语言)。[HTML元素](https://zh.wikipedia.org/wiki/HTML%E5%85%83%E7%B4%A0)是构建网站的基石。HTML允许嵌入图像与对象，并且可以用于创建交互式表单，它被用来结构化信息——例如标题、段落和列表等等，也可用来在一定程度上描述文档的外观和[语义](https://zh.wikipedia.org/wiki/%E8%AF%AD%E4%B9%89)。HTML的语言形式为[尖括号](https://zh.wikipedia.org/wiki/%E6%8B%AC%E5%8F%B7" \o "括号)包围的HTML元素，浏览器使用HTML标签和脚本来诠释网页内容，但不会将它们显示在页面上。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、本地运行、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1995年由网景公司为浏览器开发的一门脚本语言,是一门动态类型、面向对象的解释型语言。经由ECMA通过ECMAScript实现语言的标准化。完整的JavaScript包含三个部分:即

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W3C组织推荐的处理可扩展置标语言的标准编程接口。是表示和处理一个HTML或XML文档的常用方法。DOM可以使用于任何编程语言。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。是浏览器提供的可以操作浏览器窗口的对象结构,BOM提供了多个对象,其中顶层对象就是代表当前浏览器窗口的window对象，其他对象都是该对象的子对象,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

Vue是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与[现代化的工具链](https://cn.vuejs.org/v2/guide/single-file-components.html)以及各种[支持类库](https://github.com/vuejs/awesome-vue#libraries--plugins)结合使用时，Vue 也完全能够为复杂的单页应用提供驱动

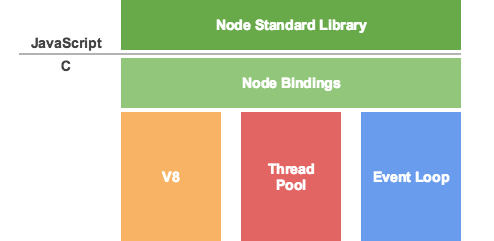
vue-router是在使用Vue构建单页应用时通过路由映射来渲染不同的子组件实现页面的渲染和切换。

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化

Iview是一套以Vue.js为基础开发的一套开源UI框架。可以快速便捷的搭建页面并获得统一的设计风格和表现形式.

## 3.3 服务端架构及技术选型

### 3.3.1 Node.js

Node.js 不是一门语言也不是框架，它只是基于 Google V8 引擎的 JavaScript 运行时环境，同时结合 Libuv 扩展了 JavaScript 功能，使之支持 io、文件等只有语言才有的特性，使得 JavaScript 能够同时具有 DOM 操作(浏览器)和 I/O、文件读写、操作数据库(服务器端)等能力。其原理图如下所示

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地狱问题,而且还提高了代码的可读性和可维护性同时改进了错误处理。另一方面采用中间件的设计，逻辑功能都在中间件中实现使主函数库更加轻量优雅。

### 3.3.3非关系型数据库MongoDB

MongoDB是一个开源的、高性能、模式自由的面向文档的数据库。正因为MongoDB具有面向文档特性所以能以类似JSON的文档集合保存数据。而且数据可以被嵌套到复杂的结构中并保持可查询可索引。这样一来应用程序可以以一种更加自然的方式来为数据建模。MongoDB有着以下优点:

编码类型统一, MongoDB数据采用统一的UTF-8编码,不是UTF-8集合的数据也可以通过使用一种特殊的二进制数据类型来保存和查询。

跨平台支持。MongoDB可以在Windows系统、Linux系统、OSX平台使用。

支持丰富的数据类型包括Date、code、binary data、regular expressions等类型

支持嵌套查询和索引。以及数据聚合来提高数据操作的便捷性。

### 3.3.4 MongoDB的ORM工具库Mongoose

Mongoose是MongoDB的一个对象模型工具,是基于node-mongodb-native开发的MongoDB nodejs驱动,可以在异步的环境下执行。同时封装了MongoDB对文档的一些增删改查的常用方法,让Node.js操作MongoDB数据库变得更加灵活简单。Mongoose主要包括以下几个方面

#### 3.3.4.1 Schema

Schema是一种以文件形式储存的数据库模型骨架。无法直接通往数据库端。它不具备直接操作数据库的能力。仅仅作为数据库模型在程序片段中的一种表现。是对文档结构的定义,定义一个Schema代码如下

UserSchema = new mongoose.Schema({

name: {

type: String,

required: true

}

})

Schema可以规定文档所具有的属性并限制属性值的类型还有丰富的对字段校验的功能。

#### 3.3.4.2 Model

模型(Model)是由Schema构造生成的模型。除了由Schema定义的数据库骨架外,Model还具有操作数据库的能力。可以在Model基础上进行CRUD操作

由Schema创建Model的代码如下

const UserModel = mongoose.model(“User”,UserSchema)

上述代码可以以Schema为基础创建相应的Model。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,使用save方法保存数据。Model和Entity都有操作数据库的能力。但Model比Entity更具有操作性。由Model生成相应Entity代码如下:

let user = new User({name: ”test”})

user.save()

Entity由相应Model生成并调用save方法后就永久保存至数据库中。可在后续的操作中操作此条数据。

### 3.3.5 WebSocket的服务端实现socket.io

#### 3.3.5.1 简介

socket.io 是一个为实时应用提供跨平台实时通信的库。socket.io 旨在使实时应用在每个浏览器和移动设备上成为可能，模糊不同的传输机制之间的差异。  
socket.io 的名字源于它使用了浏览器支持并采用的 HTML5 WebSocket 标准，因为并不是所有的浏览器都支持 WebSocket ，所以该库支持一系列降级功能：

(1). Websocket

(2). AJAX long polling

(3). AJAX multipart streaming

(4). Forever Iframe

#### 3.3.5.2 使用方式

服务端以http为容器启动socket服务器的代码为

var app = require('koa')()

, server = require('http').createServer(app)

, io = require('socket.io').listen(server);

server.listen(1105);

io.sockets.on('connection', function (socket) {

socket.emit('news', { hello: 'world' });

socket.on('my other event', function (data) {

console.log(data);

});

});

其中 io = require('socket.io').listen(server); 将 socket.io 绑定到服务器上，于是任何连接到该服务器的客户端都具备了实时通信功能。 io.sockets.on('connection', function (socket) { ... }) 的作用是服务器监听所有客户端，并返回该新连接对象，接下来我们就可以通过该连接对象（socket）与客户端进行通信了。你可以看到不管是服务器还是客户端都有 emit 和 on 这两个函数，可以说 socket.io 的核心就是这两个函数了，通过 emit 和 on 可以轻松地实现服务器与客户端之间的双向通信。

emit ：用来发射一个事件或者说触发一个事件，第一个参数为事件名，第二个参数为要发送的数据，第三个参数为回调函数（一般省略，如需对方接受到信息后立即得到确认时，则需要用到回调函数）。

on ：用来监听一个 emit 发射的事件，第一个参数为要监听的事件名，第二个参数为一个匿名函数用来接收对方发来的数据，该匿名函数的第一个参数为接收的数据，若有第二个参数，则为要返回的函数。

socket.io 提供了三种默认的事件（客户端和服务器都有）：connect 、message 、disconnect 。当与对方建立连接后自动触发 connect 事件，当收到对方发来的数据后触发 message 事件（通常为 socket.send() 触发），当对方关闭连接后触发 disconnect 事件。

此外，socket.io 还支持自定义事件，毕竟以上三种事件应用范围有限，正是通过这些自定义的事件才实现了丰富多彩的通信。

最后，需要注意的是，在服务器端区分以下三种情况：

socket.emit() ：向建立该连接的客户端广播

socket.broadcast.emit() ：向除去建立该连接的客户端的所有客户端广播

io.sockets.emit() ：向所有客户端广播，等同于上面两个的和

# 4 系统的设计与实现

本章主要介绍整个系统的设计与实现。

## 4.1 客户端的设计与实现

### 4.1.1 登录注册

### 4.1.2 群组聊天和私人聊天

### 4.1.3 好友列表

### 4.1.4 设置及个人信息修改

## 4.2 服务端的设计与实现

### 4.2.1 路由设计

### 4.2.2 数据库设计

# 5 总结与展望

## 5.1 对本文工作的总结

## 5.2 对未来工作的展望

# 结束语

# 致 谢

# 参考文献