**上海电力学院**

WEB2.0技术 大作业



题　　目：　 开源在线聊天室Chatsky

院　　系：　 计算机科学与技术学院

专业年级：　 软件工程2011级

学生姓名：　 凤翔　　学号：　 20113092

2014年11月 6日

目录

[目录 2](#_Toc403068169)

[1 可行性研究 3](#_Toc403068170)

[1.1. 编写目的 3](#_Toc403068171)

[1.2. 背景 3](#_Toc403068172)

[1.1.1. 开发的软件系统的名称 3](#_Toc403068173)

[1.1.2. 项目的开发者 3](#_Toc403068174)

[1.1.3. 项目的用户 3](#_Toc403068175)

[1.1.4. 技术条件方面的可行性： 3](#_Toc403068176)

[2. 需求分析 5](#_Toc403068177)

[2.1. 概述 5](#_Toc403068178)

[2.2. 功能需求 5](#_Toc403068179)

[2.3. 非功能需求 5](#_Toc403068180)

[3. 设计说明 5](#_Toc403068181)

[3.1. 后台设计 5](#_Toc403068182)

[3.2. 前端设计 5](#_Toc403068183)

[3.3. 界面设计 7](#_Toc403068184)

[4. 压力测试 9](#_Toc403068185)

[4.1. 测试环境 9](#_Toc403068186)

[4.2. 参数 9](#_Toc403068187)

[4.3. 结果 9](#_Toc403068188)

[4.4. 结论 11](#_Toc403068189)

[5. 关键代码说明 11](#_Toc403068190)

[5.1. 用户列表数据绑定 11](#_Toc403068191)

[5.2. 聊天信息显示绑定 12](#_Toc403068192)

[5.3. Router路由 12](#_Toc403068193)

[5.4. 服务器端Socket处理 14](#_Toc403068194)

[5.5. 客户端Socket监听 15](#_Toc403068195)

[5.6. 客户端私有函数（AngularJS绑定时使用） 16](#_Toc403068196)

[5.7. 客户端Angular异步请求 19](#_Toc403068197)

[6. 总结 19](#_Toc403068198)

# 可行性研究

## 编写目的

本可行性研究报告旨在分析学生信息管理系统是否有条件能开发成功，用最小的代价在尽可能的时间内确定问题是否能解决。该系统的使用者为管理员，并完成相关的基本操作。

## 背景

### 开发的软件系统的名称

学生信息管理系统

### 项目的开发者

凤翔（Sean Fung）

### 项目的用户

管理员

### 技术条件方面的可行性：

开发环境：

系统：Windows 8.1 Update

服务器环境：nodejs 0.10.29

服务器包管理器：npm

前端js包管理器：bower  
服务器语言：CoffeeScript  
MVC框架：Express 4  
数据库：MongoDB  
ORM框架：Mongoose  
模板引擎：ejs

Web Socket实现解决方案：Socket.IO  
单元测试框架：Karma  
前端页面语言：HTML5  
层叠样式表语言：CSS3、LESS  
页面脚本语言：Javascript、CoffeeScript  
页面开源框架：Angular.js, JQuary, Bootstrap  
版本控制软件：Git  
版本控制服务：Github  
数据库托管平台：Mongohq

项目托管平台：Mopaas云  
IDE：JetBrain Webstorm  
编辑器：Sublime Text 3

* + 1. 可行性分析

本项目作为即时聊天平台，是一个典型的Web App，作为Single-page Application展示可以获得较好的效果。

网页即时通讯主要需要解决的问题是长连接、动态数据绑定以及大量并发访问。

本项目拟使用WebSocket的一个实现Socket.IO以及AngularJS库解决这两大问题，通过这两个工具的搭配使用，可以较好满足SPA6大特点。

Node.js作为单线程异步实现，搭配V8引擎JIT编译的高效，在1000并发条件下满足系统稳定运行是可行的。

# 需求分析

## 概述

即时聊天平台主要满足的是较大规模用户的同时在线互动需求，并且用户需要实时收到全部消息，显示在窗口中。用户发送的消息，也需要实时广播到其他用户的浏览器端。

## 功能需求

网站可以注册和登录；

进入系统后，用户可以修改自己的昵称以及登录密码；

用户可以看到当前在线的所有用户名单；

用户可以搜索过滤当前用户；

用户可以实现和其他用户的即时聊天；

聊天内容是可以被搜索过滤的；

## 非功能需求

界面美观，操作简单，拥有流畅的用户体验；

在大规模并发条件下依然保持高速响应。

# 设计说明

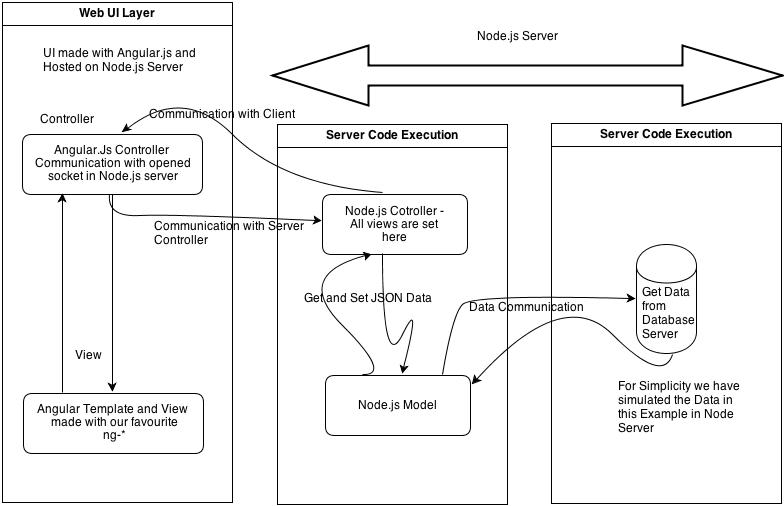
本项目使用AngularJS库作为前端MVVM框架，后端Express以及ejs模板只作为静态服务器以及RESTFUL服务器使用，渲染出基础html页面并给出API接口。

## 后台设计

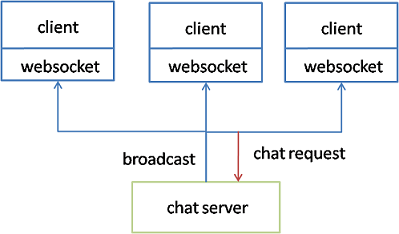
后台数据分两部分，常规AJAX数据使用RESTFUL接口供前端使用。即时长连接数据，即用户聊天相关的用户名单更新、上下线、即时聊天数据等等，则通过Socket.IO发送广播数据触发AngularJS应用更新。

## 前端设计

前端动态数据完全使用AngularJS进行数据绑定，通过后台实时改变的$scope对象，实时通过ng- 标签对前端显示数据进行绑定



**AngularJS – Socket.is – Nodejs服务器 – 数据库结构图**



**Websocket与服务器链接示意图**

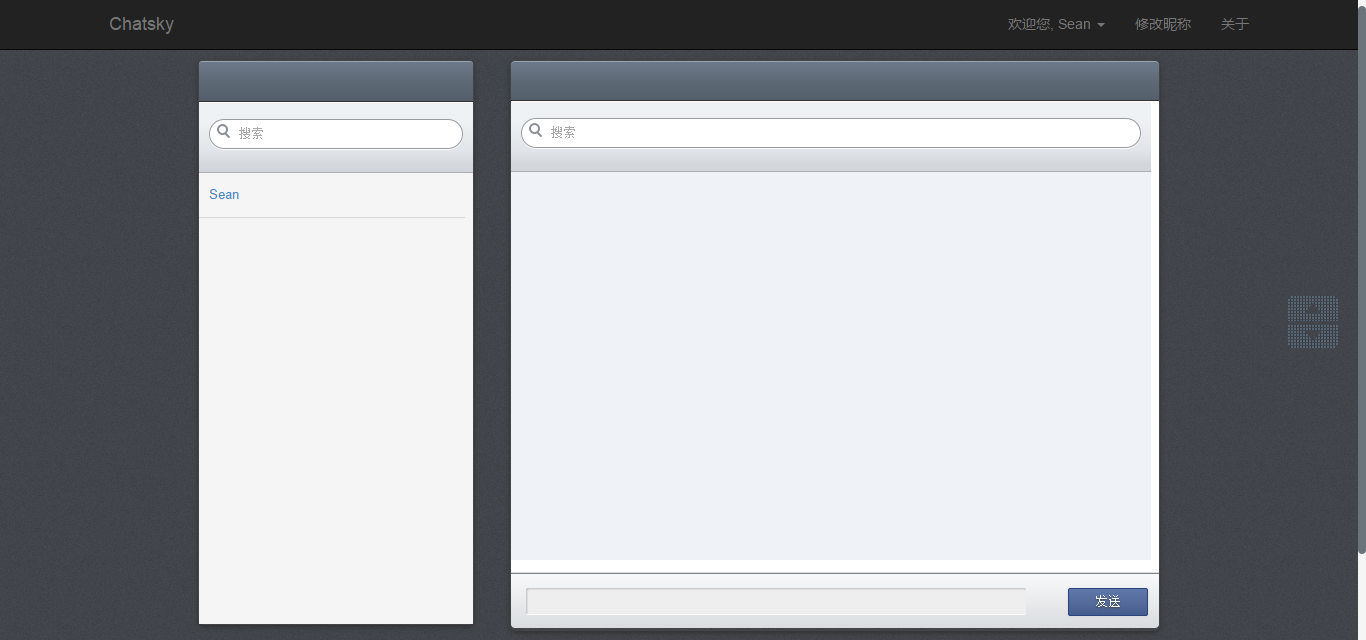
## 界面设计



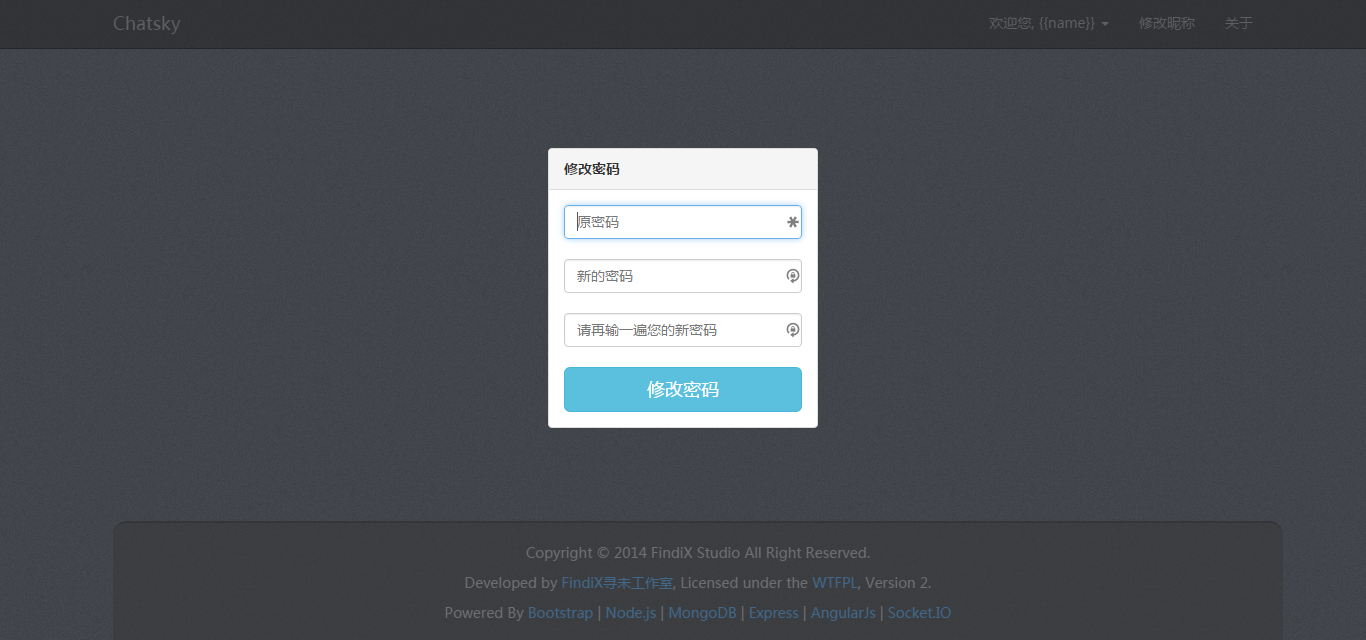
登录界面



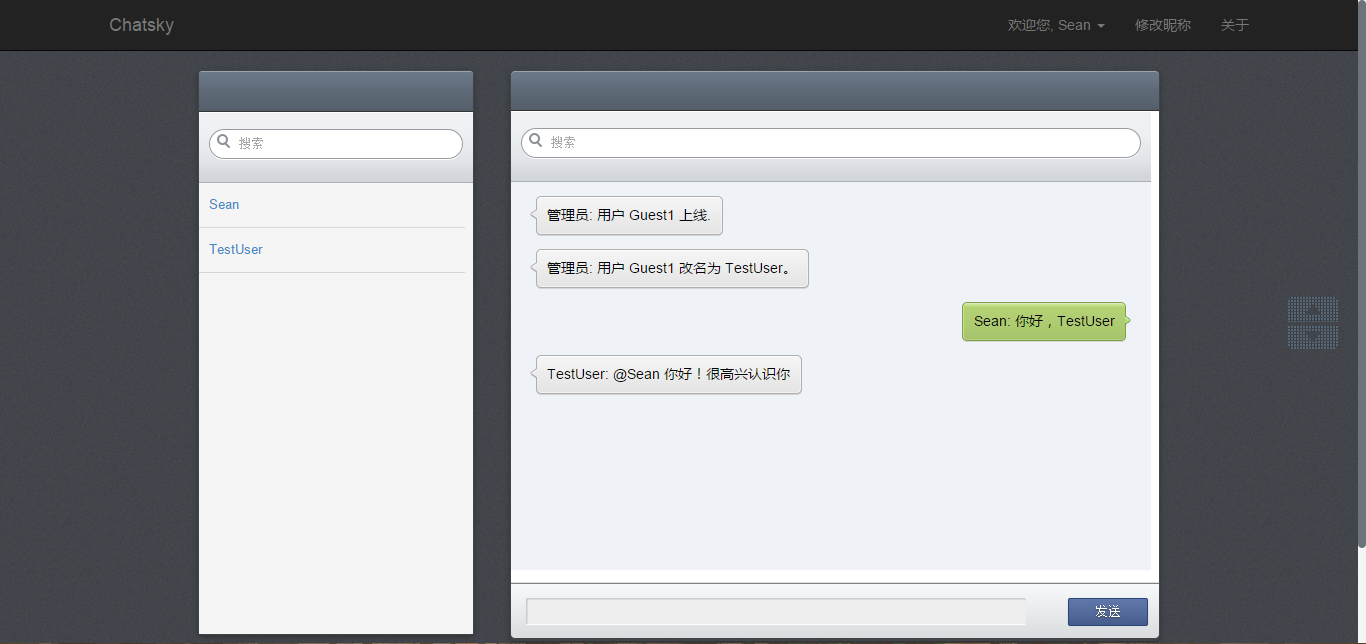
新用户注册界面



应用主界面



修改密码界面



实时聊天界面

# 压力测试

## 测试环境

网络环境：

内网

压力测试服务器：

服务器系统：Linux Ubuntu 14.04(Vmware虚拟机)

服务器配置：Intel(R) Core(TM) CPU i5 2430M 2.40GHz 2 CORES

内存：4GB

发包服务器：

发包工具：apache 2.2.19自带的ab测试工具

服务器系统：Windows(R) 8.1 简体中文企业版 64bit

服务器配置：Intel(R) Core(TM) CPU i5 2430M 2.40GHz 4 CORES

内存：8GB

## 参数

url: /

并发：1000

次数：10

## 结果

(后续测试不再注释)

This is ApacheBench**,** Version 2.3 **<**$Revision: 655654 $**>**

Copyright 1996 Adam Twiss**,** Zeus Technology Ltd**,** http:*//www.zeustech.net/*

Licensed to The Apache Software Foundation**,** http:*//www.apache.org/*

Benchmarking **localhost** **(**be patient**)**

Server Software:

Server Hostname:localhost

Server Port:80

Document Path: **/**

Document Length:4017 bytes

Concurrency Level: 1000

*/\* 整个测试持续的时间 \*/*

Time taken **for** tests: 36.869 seconds

*/\* 完成的请求数量 \*/*

Complete requests: 10000

*/\* 失败的请求数量 \*/*

Failed requests:0

Write errors: 0

*/\* 整个场景中的网络传输量 \*/*

Total transferred: 43235848 bytes

HTML transferred: 40170000 bytes

*/\* 每秒事务数 ，mean 表示这是一个平均值 \*/*

Requests per second:271.23 **[**#**/**sec**]** **(**mean**)**

*/\* 平均事务响应时间 ，mean 表示这是一个平均值 \*/*

Time per request: 3686.857 **[**ms**]** **(**mean**)**

Time per request: 3.687 **[**ms**]** **(**mean**,** across all concurrent requests**)**

*/\* 平均每秒网络上的流量，可以帮助排除是否存在网络流量过大导致响应时间延长的问题 \*/*

Transfer rate: 1145.22 **[**Kbytes**/**sec**]** received

*/\* 网络上消耗的时间的分解 \*/*

Connection **Times** **(**ms**)**

min mean**[+/-**sd**]** median max

Connect:00 10.0 0 508

Processing: 188 3570 649.5 37324442

Waiting: 12 155 447.6 182287

Total:188 3570 649.6 37324442

*/\* 整个场景中所有请求的响应情况。在场景中每个请求都有一个响应时间，其中 50 ％ 的用户响应时间小于 3732 毫秒， 66 ％ 的用户响应时间小于 3094 毫秒，最大的响应时间小于 4442 毫秒 \*/*

Percentage of the requests served within a certain **time** **(**ms**)**

50**%** 3732

66**%** 3782

75**%** 3816

80**%** 3830

90**%** 3926

95**%** 4069

98**%** 4105

99**%** 4125

100**%** 4442 **(**longest request**)**

## 结论

本测试在36.869s内完成了10次1000并发测试，所有响应均全部完成，没有失败请求。其中50%请求在3.7s内完成，最长请求时间为4.4s，考虑到服务器性能以及环境，该结果被认为是可以接受的，所以我们可以认为本系统在1000并发情况下是可用的。

# 关键代码说明

## 用户列表数据绑定

**<div** class="listContent" id="chat\_conversationListContent"**>**

**<div** id="conversationContainer" ng-repeat='user in users | filter:query'**>**

**<div** class="chatListColumn" username="{{user}}" id="{{user}}"**>**

**<div** class="extend"**>**

**</div>**

**<div** class="info"**>**

**<div** class="nickName"**>**

**<a** ng-click="mention(user)"**>**

**<div** class="left name"**>**{{user}}**</div>**

**</a>**

**</div>**

**</div>**

**</div>**

**</div>**

**</div>**

## 聊天信息显示绑定

**<div** id="chat\_chatmsglist" class="chatContent"**>**

**<div** class="chatItem" ng-repeat='message in messages | filter:queryChat' ng-class="{'me':message.user==name, 'you': message.user!=name, 'active':message.me=='active'}"**>**

**<div** class="chatItemContent"**>**

**<div** class="cloud cloudText"**>**

**<div** class="cloudPannel"**>**

**<div** class="cloudBody"**>**

**<div** class="cloudContent"**>**

{{message.user}}: {{message.text}}

**</div>**

**</div>**

**<div** class="cloudArrow "**></div>**

**</div>**

**</div>**

**</div>**

**</div>**

**</div>**

## Router路由

*# GET home page.*

router.get "/", (req, res) ->

**if** !needLogin

req.session.username = 'Chatsky'

**if** req.session.username?

res.render "index"

**else** res.render "login",

status: ''

*# GET login listing.*

router.post '/login', (req, res) ->

\_username = req.body.email

\_password = req.body.password

user.find

username: \_username

password: \_password

, (err, data) ->

**if** data.length is 0

res.end '<div class="option\_result">用户名或密码错误</div>'

**else**

req.session.username = \_username

res.end "SUCCESS"

router.post '/register', (req, res) ->

\_username = req.body.email

\_password = req.body.password

user.find

username: \_username

, (err, data) ->

**if** data.length is 0

userEntity = **new** user

username: \_username

password: \_password

userEntity.save((err)->

**if** err?

console.log err

**else**

req.session.username = \_username

res.end "SUCCESS"

)

**else**

res.end "**#{**\_username**}**已被注册"

router.use '/logout', (req, res) ->

req.session.username = **null**

res.redirect '/'

router.get '/changepasswd', (req, res)->

**unless** req.session.username?

res.redirect '/'

**else** res.render 'changePasswd',

username: req.session.username

status: ''

router.post '/user', (req, res) ->

**if** req.session.username?

res.end req.session.username

**else**

res.end 'ERROR'

## 服务器端Socket处理

*// export function for listening to the socket*

module.exports = **function** (socket) {

**var** name = userNames.getGuestName();

*// send the new user their name and a list of users*

socket.emit('init', {

name: name,

users: userNames.get()

});

*// notify other clients that a new user has joined*

socket.broadcast.emit('user:join', {

name: name

});

*// broadcast a user's message to other users*

socket.on('send:message', **function** (data) {

socket.broadcast.emit('send:message', {

user: name,

text: data.message

});

});

*// validate a user's name change, and broadcast it on success*

socket.on('change:name', **function** (data, fn) {

**if** (userNames.claim(data.name)) {

**var** oldName = name;

userNames.free(oldName);

name = data.name;

socket.broadcast.emit('change:name', {

oldName: oldName,

newName: name

});

fn(**true**);

} **else** {

fn(**false**);

}

});

*// clean up when a user leaves, and broadcast it to other users*

socket.on('disconnect', **function** () {

socket.broadcast.emit('user:left', {

name: name

});

userNames.free(name);

});

};

## 客户端Socket监听

socket.on('init', **function** (data) {

setup\_member();

current\_username = data.name;

$scope.name = data.name;

$scope.users = data.users;

});

socket.on('send:message', **function** (message) {

**var** mention, me;

mention = getMention(message.text);

**if** (mention) {

me = "active";

} **else** {

me = **null**;

}

$scope.messages.push({

user: message.user,

text: message.text,

me: me

});

});

socket.on('change:name', **function** (data) {

changeName(data.oldName, data.newName);

current\_username = data.newName;

});

socket.on('user:join', **function** (data) {

$scope.messages.push({

user: '管理员',

text: '用户 ' + data.name + ' 上线'

});

$scope.users.push(data.name);

});

*// add a message to the conversation when a user disconnects or leaves the room*

socket.on('user:left', **function** (data) {

$scope.messages.push({

user: '管理员',

text: '用户 ' + data.name + ' 下线'

});

**var** i, user;

**for** (i = 0; i < $scope.users.length; i += 1) {

user = $scope.users[i];

**if** (user === data.name) {

$scope.users.splice(i, 1);

**break**;

}

}

});

## 客户端私有函数（AngularJS绑定时使用）

**var** retrieveUsername = **function** () {

**var** username;

username = (localStorage.getItem("username") || **false**);

**if** (!username) {

**return** **false**;

}

**return** username;

};

**var** setup\_member = **function** () {

**var** username;

username = retrieveUsername();

console.log(username);

**if** (username) {

socket.emit('change:name', {

name: username

}, **function** (result) {

**if** (!result) {

alert('修改昵称失败');

} **else** {

changeNameWithoutNotify($scope.name, username);

$scope.name = username;

$scope.newName = '';

}

});

**return**;

}

**return** **false**;

};

*// Check if message has a mention for current user*

**var** getMention = **function** (message) {

**var** text, pattern, mention;

text = message;

pattern = /\B\@([\w\-]+)/gim;

mention = text.match(pattern);

**if** (mention) {

mention = String(mention).split("@")[1];

**if** (mention === current\_username) **return** mention;

}

**return** **false**;

};

**var** changeName = **function** (oldName, newName, member) {

*// rename user in list of users*

**var** i;

**for** (i = 0; i < $scope.users.length; i++) {

**if** ($scope.users[i] === oldName) {

$scope.users[i] = newName;

}

}

**if** (oldName === $scope.name) {

localStorage.setItem("username", newName);

}

current\_username = newName;

$scope.messages.push({

user: '管理员',

text: '用户 ' + oldName + ' 改名为 ' + newName + '。'

});

};

**var** changeNameWithoutNotify = **function** (oldName, newName, member) {

*// rename user in list of users*

**var** i;

**for** (i = 0; i < $scope.users.length; i++) {

**if** ($scope.users[i] === oldName) {

$scope.users[i] = newName;

}

}

current\_username = newName;

};

发布到AngularJS $scope全局变量的函数

$scope.mention = **function**(name) {

$scope.message = '@' + name + ' ';

$('.input-message').focus()

};

$scope.changeName = **function**() {

socket.emit('change:name', {

name: $scope.newName

}, **function**(result) {

**if** (!result) {

alert('修改昵称失败');

} **else** {

changeName($scope.name, $scope.newName);

$scope.name = $scope.newName;

$scope.newName = '';

$('#changeNameModal').modal('hide')

}

});

};

$scope.messages = [];

$scope.sendMessage = **function**() {

socket.emit('send:message', {

message: $scope.message

});

*// add the message to our model locally*

$scope.messages.push({

user: $scope.name,

text: $scope.message

});

*// clear message box*

$scope.message = '';

*// scroll to the bottom*

$('.chatScroll').animate({

scrollTop: $('#chat\_chatmsglist').height()

}, 1000);

};

## 客户端Angular异步请求

$http({

method: 'POST',

url: '/user'

}).success(**function** (data, status) {

$scope.username = data;

}).error(**function** (data, status) {

});

# 总结

根据大作业的要求，需要制作一个满足6点要求的Single-Page应用，实际上也就是实现了一个Web App。我的第一反应是做一个天气应用，因为之前也做过手机端的此类应用。并且天气数据可以通过开放平台获取，十分方便。之所以最终没有选择，其原因有二：一是天气应用不适合使用Socket，因为完全不需要建立长连接，二是天气Web App对于UI表现要求比较多，而我对于CSS方面并不是很擅长。

那么不妨从需求来考虑，既然想完整实现这6点，那么就从使用范围最低的Socket入手，需要Socket的无外乎实时或者大量连续通信。做游戏肯定不行，那么最适合的也就是即时通信了，也就是IM。

确定了需求，下一步就是技术选型。既然一定要做，那不如用点新东西练练手。所以最初后台是准备用Golang或者Python。最终之所以还是选择Node.js其主要原因还是Socket。目前WebSocket最好的实现——Socket.io还是更适合Node平台。此外因为这个项目也主要是前端技术，相对后台不是那么重要。继续尝试对js增加了解也没什么不好。毕竟js这门语言，真的是越了解越觉得它的深奥。

后台OK了以后（当然数据库、ORM框架之类的选择也是顺其自然的，所以不表），就是更加重要的前端框架。UI使用Bootstrap不用说，MVVM以及模板框架才是重头戏。在花了2个小时查阅各种文档后，最终选择了AngularJS。

下面谈谈为什么。首先前端框架主要是解决这几个问题：数据绑定、前端模板、MVVM、路由。既然我都需要用，那么这些能用一个框架解决，就没必要使用多个徒增网络压力。Backbone缺少最重要的模板，Knockout主要是双向数据绑定和模板。这样一来，就不如AngularJS或者Ember.js这样的“一站式”适合了。

AngularJS和Ember.js的选择就比较直接了，功能基本相同，优劣见仁见智，所以就选择了相对较早听说却一直没有使用过的AngularJS。

选型选好了，那么开始前端构建吧。

这时，无意中发现了一个前端包管理器bower，这真是如获至宝，我一直都很喜欢各个平台的包管理器，Node当然是npm，Python的pip，Linux的apt-get以及yum，甚至Windows下我都会使用Chocolaty。如果前端包也可以通过管理器管理，那无疑是非常方便。哪知，去为此花费了大量的时间，得出结论：bower的源不是海底光缆断了就是彻底被墙了，并且国内没有镜像可以使用，总之就是不能用。最终有一部分库通过下载压缩包bower手动安装方式使用，但是方便的管理最终还是没能实现，实在遗憾遗憾，为国内的网络环境。

界面部分主要参考了网页版微信的设计，毕竟是现在用户量最大的IM工具嘛，登录界面自己使用之前项目中的代码，但是修改了大量的验证以及改为通过Ajax方式提交。正如老师所说，一个好的登录都是需要大量细节方面的完善。等到感觉基本满意的时候，发现已经花了4个小时了（当然包括吃饭以及查了其他一些资料）。

下面是主界面实现，这没什么好说的，html和css，最终花费2个多小时，终于有点样子了。

下面是重头戏，关于Socket.IO以及AngularJS的使用，这就是痛苦的开始，这两项技术分开来看都没什么，可是同时使用的时候问题就来了。我原先以为Angular是会自动双向绑定的，但是写了一个Demo之后发现死活没反应，甚至自己写了一个测试，延迟5秒之后变更$scope，竟然页面没有更改，真的是崩溃。再去看文档（英语不好真是受罪）才明白过来，AngularJS只是内置了很少的一部分触发条件，比如按钮点击，按键等等，如果是自己操纵$scope的话，是要自行触发更新的。换言之，我需要在客户端监听到的每个Socket事件后，自行触发$scope更新。只是在明白这一点的时候，已经是早上8点了，因为下午有课程需要答辩，只好git commit代码，赶回学校。基本都还没开始呢，却已经花费了16个小时。主要时间还是在阅读文档和纠结方案上。

当天晚上继续，因为之前已经发现了可能的问题所在，所以实在等不及验证一下。尝试重构服务器和客户端的Socket事件发送监听，把Angular更新代码放在客户端Socket里。这样一来初步尝试成功，接下来就是重复的绑定，debug，功能增加。总计花费3天超过10个小时，终于是在周三晚上熄灯之前完成了一个各方面堪用的版本，部署上线了一个版本，发送地址到群里让同学们帮忙测试一下，从注册到登录聊天，基本功能算是没有问题，想要改进的方面还有很多，不过答辩日期将近，所以我决定把项目开源在Github上，并且在今后逐步完善，因为感觉确实是很有意思。

最后呢，这次的项目主要是对WebSocket以及AngularJS有了初步的了解，对AngularJS的感觉就是实在太帅，今后的Web项目希望都可以使用，当然下一个也想尝试一下Ember.js等等，总之写得很爽，很好玩。连续16小时的编码也是难得的体验，算是成功打破了之前做视频连续14小时的记录了。