**编 号：2014213963**

**审定成绩：**

|  |  |  |  |
| --- | --- | --- | --- |
| **重庆邮电大学**  **毕业设计（论文）** | | | |
|  | | | |
|  | **中文题目** | **基于AI语音识别的websocket聊天室** |  |
|  |  |
| **英文题目** | **The websocket chat room based on AI** |
|  | **voice recognition** |
| **学院名称** | **软件工程学院** |
| **学生姓名** | **陈俊松** |
| **专 业** | **软件工程** |
| **班 级** | **1301415** |
| **学 号** | **2014213963** |
| **指导教师** | **宋琦 讲师** |
| **答 辩 组**  **负 责 人** | **刘歆 副教授** |
|  | |

**二〇一八 年 六 月**

**重庆邮电大学教务处制**

软件工程学院本科毕业设计(论文)诚信承诺书

本人郑重承诺：

我向学院呈交的论文《基于AI语音识别的websocket聊天室》，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明并致谢。本人完全意识到本声明的法律结果由本人承担。

年级 2014级

专业 软件工程

班级 1301415

承诺人签名

年 月 日

学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

|  |  |
| --- | --- |
| 学生签名： | 指导老师签名： |
| 日期： 年 月 日 | 日期： 年 月 日 |

摘要

随着时代的飞速发展，互联网技术日新月异，传统的通信技术已经不能满足我们在日常通信的要求，webSocket即时通信技术应运而生，而基于语音识别的即时通信技术更是将我们的生活丰富化，便捷化。技术的发展，引导我们不断的进步。webSocket协议本质上是一个基于TCP的协议。为了建立一个webSocket连接，客户端浏览器首先要向服务器发起一个HTTP请求，这个请求和通常的HTTP请求不同，包含了一些附加头信息，其中附加头信息“Upgrade: WebSocket”表明这是一个申请协议升级的HTTP请求，服务器端解析这些附加的头信息然后产生应答信息返回给客户端，客户端和服务器端的webSocket便可建立连接。双方就可以通过这个连接通道自由的传递信息，并且这个连接会持续存在直到客户端或者服务器端的某一方主动的关闭连接。语音识别技术，也被称为自动语音识别Automatic Speech Recognition(ASR)，就是让机器通过识别和理解过程把语音信号转变为相应的文本或命令的新技术。主要包括特征提取技术、模式匹配准则及模型训练技术三个方面。其目标是将人类的语音中的词汇内容转换为计算机可读的输入。

基于AI语音识别的websocket聊天室主要是通过语音识别技术将我们的声音进行语音识别并及时将传输到服务端，服务端通过验证进行语音识别，并返回响应，同时机器人进行智能回复，智能查询相关信息，适用于多个应用场景，可流畅的运行在主流windows和ios系统中。

基于AI语音识别的websocket聊天室主要是基于C/S架构，前端通过react+ant.design进行MVVM模式构建，后端通过express+websocket进行服务端实现，数据库通过NO-SQL的MongoDB的进行搭建。

**关键词：**websocket，语音识别，聊天室

**Abstract**

With the rapid development of the times, Internet technology is changing with each passing day, traditional communication technologies can no longer meet the requirements of our daily communications, WebSocket instant messaging technology came into being, and instant messaging technology based on speech recognition enriches our lives. Facilitation. The development of technology leads us to continuous improvement. The WebSocket protocol is essentially a TCP-based protocol. In order to establish a WebSocket connection, the client browser first sends an HTTP request to the server. This request is different from the usual HTTP request and contains some additional header information. The additional header “Upgrade: WebSocket” indicates that this is an application protocol. Upgraded HTTP requests. The server parses these additional headers and generates response information back to the client. WebSockets on the client and server can establish connections. Both parties can freely transmit information through this connection channel, and this connection will continue to exist until either the client or the server side actively closes the connection. Speech recognition technology, also known as Automatic Speech Recognition (ASR), is a high technology that allows machines to convert speech signals into corresponding text or commands by identifying and understanding the process. It mainly includes three aspects: feature extraction technology, pattern matching criteria, and model training technology. The goal is to convert the vocabulary content of human speech into computer-readable input.

The Websocket chat room based on AI voice recognition mainly uses voice recognition technology to carry out voice recognition of our voices and transmit data to the server in time. The server authenticates and recognizes voices and returns responses. It can also reply intelligently through robots. Query related information, suitable for multiple application scenarios, can run smoothly on various systems.

The Websocket chat room based on AI voice recognition is mainly based on the C/S architecture. The front end constructs the MVVM mode through react + ant.design, and the back end implements the server through express + websocket. The database is constructed through NO-SQL MongoDB

**Keywords**: Websocket , voice recognition, chat room

目录

[第1章 引言 1](#_Toc515720542)

[1.1 研究背景和意义 1](#_Toc515720543)

[1.2 国内外研究现状 3](#_Toc515720544)

[1.2.1 国外研究现状 3](#_Toc515720545)

[1.2.2 国内研究现状 4](#_Toc515720546)

[1.3 主要内容和工作安排 5](#_Toc515720547)

[第2章 开发工具及平台兼容性 7](#_Toc515720548)

[2.1 编码工具和调试工具的选择 7](#_Toc515720549)

[2.1.1 sublime的开发优势 7](#_Toc515720550)

[2.1.2 基于chrome扩展程序React Developer Tools的调试 7](#_Toc515720551)

[2.2 Node.js和npm介绍 9](#_Toc515720552)

[2.2.1 Node.js的介绍和开发优势 9](#_Toc515720553)

[2.2.2 npm包管理工具的使用介绍 10](#_Toc515720554)

[2.3 打包工具webpack介绍 11](#_Toc515720555)

[2.4 代码规范及ESLint介绍使用 11](#_Toc515720556)

[2.5 MongoDB数据库介绍 12](#_Toc515720557)

[2.6 本章小结 15](#_Toc515720558)

[第3章 系统分析与设计 17](#_Toc515720559)

[3.1 系统需求分析 17](#_Toc515720560)

[3.2 系统概要设计 18](#_Toc515720561)

[3.2.1 系统结构设计 18](#_Toc515720562)

[3.3 数据库设计 18](#_Toc515720563)

[3.4 本章小结 20](#_Toc515720564)

[第4章 系统的具体实现 23](#_Toc515720565)

[4.1 前端的具体实现 23](#_Toc515720566)

[4.1.1 音频转换模块 23](#_Toc515720567)

[4.1.2 语音识别模块 26](#_Toc515720568)

[4.1.3 语音朗读模块 27](#_Toc515720569)

[4.1.4 登录注册模块 29](#_Toc515720571)

[4.1.5 聊天室模块 31](#_Toc515720572)

[4.2 后端express + websocket设计 31](#_Toc515720573)

[4.2.1 websocket设计模块 31](#_Toc515720574)

[4.2.2 语音识别接口模块 32](#_Toc515720575)

[4.2.3 机器人回复模块 33](#_Toc515720576)

[4.2.4 登录注册模块 35](#_Toc515720577)

[4.2.5 聊天室模块 35](#_Toc515720578)

[4.2.6 数据库连接模块 36](#_Toc515720579)

4.3 本章小结 33

[第5章 系统测试 37](#_Toc515720580)

[5.1 软件测试的目的 39](#_Toc515720581)

[5.2 软件测试的方法 40](#_Toc515720582)

[5.3 测试方案 41](#_Toc515720583)

[5.3.1 单元测试 41](#_Toc515720584)

[5.3.2 白盒测试 41](#_Toc515720585)

[5.3.3 黑盒测试 41](#_Toc515720586)

5.4 测试用例 37

5.5 测试结论 38

5.6 本章小结 38

[第6章 总结与展望 43](#_Toc515720590)

[参考文献 45](#_Toc515720591)

[致谢 47](#_Toc515720592)

附录 49

# 

# 第1章 引言

## 1.1 研究背景和意义

大概在1992年，一家称作Nombas的公司开发了一种叫做C减减（C-minus-minus，简称Cmm）的嵌入式脚本语言。Cmm背后的理念很简单：一个足够强大可以替代宏操作（macro）的脚本语言，同时保持与C（和C++）足够的相似性，以便开发人员能很快学会。这个脚本语言捆绑在一个叫做CEnvi的共享软件中，它首次向开发人员展示了这种语言的威力。Nombas最终把Cmm的名字改成了ScriptEase。当Netscape Navigator崭露头角时，Nombas开发了一个可以嵌入网页中的CEnvi的版本。这些早期的试验被称为Espresso Page（浓咖啡般的页面），它们代表了第一个在万维网上使用的客户端语言。而Nombas丝毫没有料到它的理念将会成为万维网的一块重要基石[1]。

当网上冲浪越来越流行时，对于开发客户端脚本的需求也逐渐增大。此时，大部分因特网用户还仅仅通过28.8kbit/s的调制解调器连接到网络，即便这时网页已经不断地变得更大和更复杂。而更加加剧用户痛苦的是，仅仅为了简单的表单有效性验证，就要与服务器进行多次地往返交互。设想一下，用户填完一个表单，点击提交按钮，等待了30秒的处理后，看到的却是一条告诉你忘记填写一个必要的字段。那时正处于技术革新最前沿的Netscape，开始认真考虑开发一种客户端脚本语言来解决简单的处理问题。

就在Netscape Navigator 2.0即将正式发布前，Netscape将其更名为JavaScript，目的是为了利用Java这个因特网时髦词汇。Netscape的赌注最终得到回报，JavaScript从此变成了因特网的必备组件。

随着JavaScript的迅速发展，其在互联网的角色也越来越鲜明，重要性也越来越高，由此在互联网领域的作用日渐突出。根据许多调查显示，JavaScript连续多年在互联网开发中高居榜首。

“**用JavaScript编写的程序，都终将会以JavaScript编写**”，这句话曾经听起来十分高傲，但是随著Node.js的发展，我们渐渐地明白到这句话的远见性──JavaScript真的越来越强大了。

2009年2月，Ryan Dahl在博客上宣布准备基于V8创建一个轻量级的[Web服务器](https://www.baidu.com/s?wd=Web%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBmW6znywWPy7-nhNWPWm30ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLP1fLPHbdP1mYPW01rHcdrf)并提供一套库。2009年5月，Ryan Dahl在GitHub上发布了最初版本的部分Node.js包，随后几个月里，有人开始使用[Node.js开发](https://www.baidu.com/s?wd=Node.js%E5%BC%80%E5%8F%91&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBmW6znywWPy7-nhNWPWm30ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLP1fLPHbdP1mYPW01rHcdrf)应用。2009年11月和2010年4月，两届JSConf大会都安排了Node.js的讲座。2010年年底，Node.js获得[云计算](https://www.baidu.com/s?wd=%E4%BA%91%E8%AE%A1%E7%AE%97&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBmW6znywWPy7-nhNWPWm30ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHDLP1fLPHbdP1mYPW01rHcdrf)服务商Joyent资助，创始人Ryan Dahl加入Joyent全职负责Node.js的发展。2011年7月，Node.js在微软的支持下发布Windows版本。Node.js是一个非常新兴的开发工具，年龄远不如Python、Ruby、PHP等老大哥，但是它确实有史以来发展最快的开发工具，没有之一。在这短短的几年间，我们看到了Node.js从当初的一无所有到如今的飞速发展，这是没有任何其他开发工具能够媲美的。

HTML5草案的前身名为Web Applications 1.0，于2004年被WHATWG提出，于2007年被W3C接纳，并成立了新的HTML工作团队。HTML5的第一份正式草案已于2008年1月22日公布。HTML5仍处于完善之中。然而，大部分[现代浏览器](https://baike.baidu.com/item/%E7%8E%B0%E4%BB%A3%E6%B5%8F%E8%A7%88%E5%99%A8)已经还仍不具备HTML5支持。

“语音识别”的终极梦想，是真正能够理解人类语言甚至是方言环境的系统。但几十年来，人们并没有一个有效的策略来创建这样一个系统，直到人工智能技术的爆发。在过去几年中，人们在人工智能和深度学习领域的突破，让语音识别的探索跨了一大步。其中的精髓便是对话式AI，对话式AI包含有两个主要类别：人机界面，以及人与人沟通的界面。在人机界面中，人类与机器往往通过语音或文本交互，届时机器会理解人类 (尽管这种理解方式是有限的) 并采取相应的一些措施。

基于HTML5的版本兼容性和JavaScript以及AI语音识别技术的不断演变，为了实现真正的纯JavaScript语言演绎的PC端语音识别系统，由此开发了基于AI语音识别的聊天室。

基于AI语音识别的聊天室摒弃了传统的JSP开发模式，前端页面和后端服务器接口都采用纯JavaScript语言编写，实现技术的一致性，减轻前后端数据连通的复杂性，可以使开发人员清晰的了解数据的流动，实现前后端的无缝式开发。而前端部分通过对HTML5标准进行二次包装，以实现语音格式转换，语音朗读TTS引擎的兼容性，语音识别部分通过封装的AI接口，智能识别并返回机器人响应。完善了HTML5标准的部分接口和AI语音识别的信息响应，实现了纯PC端JavaScript开发的AI语音识别聊天室。

## 1.2 国内外研究现状

### 1.2.1 国外研究现状

语音识别的研究工作可以追溯到20世纪50年代AT&T贝尔实验室的Audry系统，它是第一个可以识别十个英文数字的语音识别系统。但真正取得实质性进展，并将其作为一个重要的课题开展研究则是在60年代末70年代初。实验室语音识别研究的巨大突破产生于20世纪80年代末：人们终于在实验室突破了大词汇量、连续语音和非特定人这三大障碍，第一次把这三个特性都集成在一个系统中，比较典型的是卡耐基梅隆大学（CarnegieMellonUniversity）的Sphinx系统，它是第一个高性能的非特定人、大词汇量连续语音识别系统。

20世纪90年代前期，许多著名的大公司如IBM、苹果、AT&T和NTT都对语音识别系统的实用化研究投以巨资。语音识别技术有一个很好的评估机制，那就是识别的准确率，而这项指标在20世纪90年代中后期实验室研究中得到了不断的提高。比较有代表性的系统有：IBM公司推出的ViaVoice和DragonSystem公司的NaturallySpeaking，Nuance公司的NuanceVoicePlatform语音平台，Microsoft的Whisper，Sun的VoiceTone等。

其中IBM公司于1997年开发出汉语ViaVoice语音识别系统，次年又开发出可以识别上海话、广东话和四川话等地方口音的语音识别系统ViaVoice‘98。它带有一个32，000词的基本词汇表，可以扩展到65，000词，还包括办公常用词条，具有“纠错机制”，其平均识别率可以达到95%。该系统对新闻语音识别具有较高的精度，是目前具有代表性的汉语连续语音识别系统。

2002年8月，W3C建立了第4个HTML工作组（实际上是第3个XHTML工作组），目标是完成HTML到XHTML的转变，包括完成以尽可能多的使用通用的XML技术为设计目标的XHTML2.0。同时，HTML工作组发布了XHTML2.0第一个公开工作草案，并在声明中指出，尽管XHTML2.0来源于HTML4、XHTML1.0和XHTML1.1，XHTML2.0并不向后兼容早期版本。

HTML5在HTML4的基础上新增了较多元素和属性，同时，也废除了部分元素和属性。与HTML4相比，HTML5在文档结构、页面元素、图形支持、多媒体支持、本地存储、离线应用、多线程和地理位置信息等方面有较大的变化，这些变化极大地提升了HTML5对Web应用的开发支持。

浏览器是HTML5的重要解析器，因此，浏览器对HTML5的支持直接影响 HTML5的应用。由于主要浏览器厂商直接参与了HTML5的标准制定工作，因此，多数浏览器厂商发布的新版本浏览器都加强了对HTML5的支持。

### 1.2.2 国内研究现状

我国语音识别研究工作起步于五十年代，但近年来发展很快。研究水平也从实验室逐步走向实用。从1987年开始执行国家863计划后，国家863智能计算机专家组为语音识别技术研究专门立项，每两年滚动一次。我国语音识别技术的研究水平已经基本上与国外同步，在汉语语音识别技术上还有自己的特点与优势，并达到国际先进水平。

中科院自动化所、声学所、清华大学、北京大学、哈尔滨工业大学、上海交通大学、中国科技大学、北京邮电大学、华中科技大学等科研机构都有实验室进行过语音识别方面的研究，其中具有代表性的研究单位为清华大学电子工程系与中科院自动化研究所模式识别国家重点实验室。

HTML5标准涉及浏览器开发商、工具开发商及应用开发商等，技术覆盖面较广，标准制定难度较大。我国为了推动基于HTML5的互联网和移动互联网产业发展，我国标准研制主要聚焦于浏览器兼容性规范、浏览器编程接口及标准符合性测试规范 等基础性标准，并结合HTML5的后续发展，逐步建立起符合我国产业发展需求的标准体系。

目前，HTML5标准由W3C和WHATWG联合制定。我国的百度、腾讯、UC、华为、盛大、360 等互联网企业已于近年加入W3C，并派代表参加HTML工作组[3]。

## 1.3 主要内容和工作安排

本文主要按以下方式组织：

第一章：主要介绍系统的研究目的和意义，研究背景，研究现状，以及标准实现的不同情况。

第二章：主要介绍系统实现的所需工具优势，调试软件的使用，包管理器的开发和使用，打包工具的使用，以及数据库开发的优势

第三章：主要介绍系统的设计方向，包括系统需求分析设计，系统概要设计，系统结构设计，数据库设计方向。

第四章：主要介绍系统的编码实现，包括HTML5标准的实现以及兼容处理，包括对不支持浏览器的容错处理，语音识别的格式转换问题，前端框架架构处理，后端接口数据处理，以及机器人回复处理。

第五章：主要介绍系统测试结果，包括语音识别结果，服务器压力测试，系统黑盒和白盒测试，数据库相关测试。

# 第2章 开发工具及平台兼容性

## 2.1 编码工具和调试工具的选择

### 2.1.1 sublime的开发优势

Sublime Text是一个代码编辑器，也是HTML和散文先进的文本编辑器。Sublime Text是由程序员Jon Skinner于2008年1月份所开发出来，它最初被设计为一个具有丰富扩展功能的[Vim](https://baike.baidu.com/item/Vim)。

Sublime Text具有漂亮的用户界面和强大的功能，例如代码缩略图，[Python](https://baike.baidu.com/item/Python)的插件，代码段等。还可自定义键绑定，菜单和工具栏。Sublime Text的主要功能包括：拼写检查，书签，完整的Python API，Goto功能，即时项目切换，多选择，多窗口等等。Sublime Text是一个跨平台的编辑器，同时支持[Windows](https://baike.baidu.com/item/Windows)、[Linux](https://baike.baidu.com/item/Linux)、[Mac OS X](https://baike.baidu.com/item/Mac%20OS%20X)等操作系统。作为主流前端开发编辑器，它不仅体积较小，运行速度快并且文本功能强大，同时支持编译功能且可在控制台看到输出，内嵌python解释器支持插件开发以达到可扩展目的和内置Package Control——可支持的大量插件可通过其进行管理。

### 2.1.2 基于chrome扩展程序React Developer Tools的调试

React Developer Tools是一款由facebook开发的有用的[Chrome浏览器扩展](http://www.cnplugins.com/)，可以通过Chrome Web存储获取。使用Chrome Devtools进行调试时，可以查看应用程序的React组件分层结构，而不是更加神秘的浏览器DOM表示。添加react developer tools到chrome，是对chrome开发工具的React调试工具。React的开发工具是开源Chrome DevTools延伸反应的JavaScript库。它允许你检查React在Chrome开发者工具组件的层次结构（原名WebKit Web Inspector）。你会得到新的标签要求在你的Chrome DevTools反应。这表明你的根反应组件在页面渲染，以及他们最终渲染组件。

插件安装成功后，浏览器的右上方出现现图示的log，用户如果想要调试的话就点击log就可以了。通过选择树中的一个组件，你可以在右边的面板中检查和编辑它的当前的道具和状态。在面包屑，你可以检查所选组件，创建它的组件，创建一个组件，等等。如果您使用常规元素选项卡检查页面上的一个反应元素，然后切换到“React”选项卡，该元素将在“React树”中自动选择。

如图2.1所示显示了安装了React Developer Tools的example3组件分层结构外观。

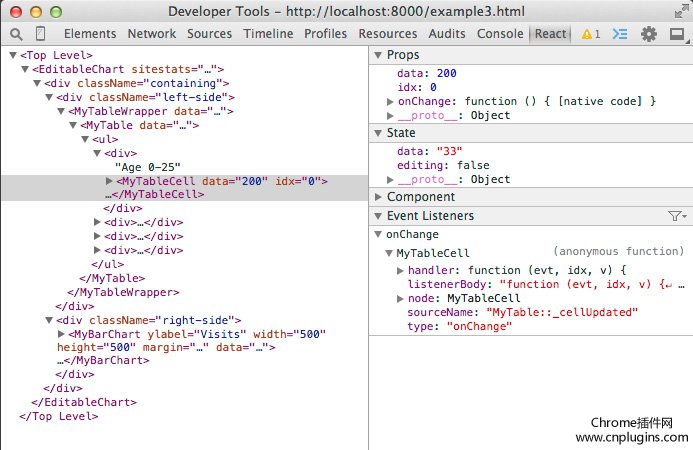


图2.1 React Developer Tools 的 example3 组件分层结构外观图

类似地，如图2.2如果在一个组件的渲染阶段有一个断点，将在“React”选项卡中自动选择。这允许你在渲染树中一步一步看到一个组件如何影响另一个。通过抛出的错误，你可以很容易地找到哪些组件在渲染过程中抛出一个错误，和什么样的原因导致错误的产生。

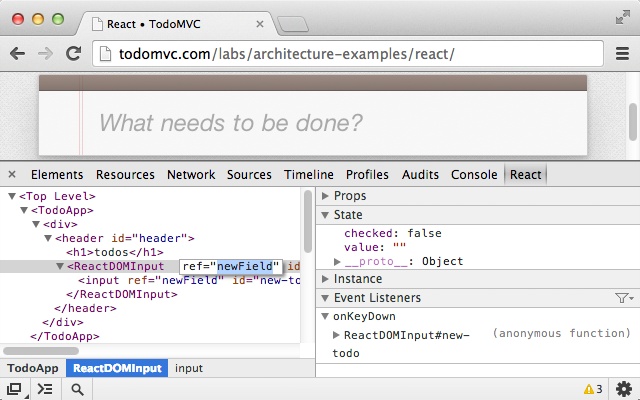


图2.2 错误断点调试

## 2.2 Node.js和npm介绍

### 2.2.1 Node.js的介绍和开发优势

Node.js是一个Javascript运行环境(runtime environment)，发布于2009年5月，由Ryan Dahl开发，实质是对Chrome V8引擎进行了封装。Node.js对一些特殊用例进行优化，提供替代的API，使得V8在非浏览器环境下运行得更好。

V8引擎执行Javascript的速度非常快，性能非常好。Node.js是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用[事件驱动](https://baike.baidu.com/item/%E4%BA%8B%E4%BB%B6%E9%A9%B1%E5%8A%A8)，非阻塞[I/O](https://baike.baidu.com/item/I%2FO/84718)模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。

V8引擎本身使用了一些最新的编译技术。这使得用Javascript这类[脚本语言](https://baike.baidu.com/item/%E8%84%9A%E6%9C%AC%E8%AF%AD%E8%A8%80)编写出来的代码运行速度获得了极大提升，又节省了开发成本。对性能的苛求是Node的一个关键因素。Javascript是一个[事件驱动](https://baike.baidu.com/item/%E4%BA%8B%E4%BB%B6%E9%A9%B1%E5%8A%A8)语言，Node利用了这个优点，编写出可扩展性高的服务器。Node采用了一个称为“事件循环(event loop）”的架构，使得编写可扩展性高的服务器变得既容易又安全。提高服务器性能的技巧有多种多样。Node选择了一种既能提高性能，又能减低开发复杂度的架构。这是一个非常重要的特性。并发编程通常很复杂且布满地雷。Node绕过了这些，但仍提供很好的性能。

Node采用一系列“非阻塞”库来支持事件循环的方式。本质上就是为文件系统、数据库之类的资源提供接口。向文件系统发送一个请求时，无需等待硬盘（[寻址](https://baike.baidu.com/item/%E5%AF%BB%E5%9D%80)并检索文件），硬盘准备好的时候非阻塞接口会通知Node。该模型以可扩展的方式简化了对慢资源的访问，直观，易懂。尤其是对于熟悉[onmouseover](https://baike.baidu.com/item/onmouseover)、onclick等[DOM](https://baike.baidu.com/item/DOM/50288)事件的用户，更有一种似曾相识的感觉。

虽然让Javascript运行于服务器端不是Node的独特之处，但却是其一强大功能。不得不承认，浏览器环境限制了我们选择编程语言的自由。任何服务器与日益复杂的浏览器客户端应用程序间共享代码的愿望只能通过Javascript来实现。虽然还存在其他一些支持Javascript在服务器端 运行的平台，但因为上述特性，Node发展迅猛，成为事实上的平台。

### 2.2.2 npm包管理工具的使用介绍

NPM的全称是Node Package Manager，是一个NodeJS包管理和分发工具，已经成为了非官方的发布Node模块（包）的标准。

Nodejs自身提供了基本的模块，但是开发实际应用过程中仅仅依靠这些基本模块则还需要较多的工作。幸运的是，Nodejs库和框架为我们提供了帮助，让我们减少工作量。但是成百上千的库或者框架管理起来又很麻烦，有了NPM，可以很快的找到特定服务要使用的包，进行下载、安装以及管理已经安装的包。

它是世界上最大的软件注册表，每星期大约有 30 亿次的下载量，包含超过600000个包（package）（即，代码模块）。来自各大洲的开源软件开发者使用npm互相分享和借鉴。包的结构使您能够轻松跟踪依赖项和版本。

npm由三个独立的部分组成:网站，注册表（registry），命令行工具（CLI）。[网站](https://npmjs.com/)是开发者查找包（package）、设置参数以及管理npm使用体验的主要途径。注册表是一个巨大的数据库，保存了每个包（package）的信息。[CLI](https://docs.npmjs.com/cli/npm)通过命令行或终端运行。开发者通过CLI与npm打交道。

## 2.3打包工具webpack介绍

现今的很多网页其实可以看做是功能丰富的应用，它们拥有着复杂的JavaScript代码和一大堆依赖包。为了简化开发的复杂度，前端社区涌现出了很多好的实践方法

模块化，让我们可以把复杂的程序细化为小的文件;

类似于TypeScript这种在JavaScript基础上拓展的开发语言：使我们能够实现目前版本的JavaScript不能直接使用的特性，并且之后还能转换为JavaScript文件使浏览器可以识别；Sass，less等CSS预处理器

这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别,而手动处理又是非常繁琐的，这就为WebPack类的工具的出现提供了需求。

webpack是一个现代JavaScript应用程序的静态模块打包器(module bundler)。当webpack处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

## 2.4 代码规范及ESLint介绍使用

ESLint是一个用来识别ECMAScript并且按照规则给出报告的代码检测工具，使用它可以避免低级错误和统一代码的风格。ESLint被设计为完全可配置的，主要有两种方式来配置ESLint：

在注释中配置：使用JavaScript注释直接把配置嵌入到JS文件中

配置文件：使用下面任一的文件来为全部的目录和它的子目录指定配置信息。

javascript：使用.eslintrc.js文件并导出一个包含配置的对象。

YAML：.eslintrc.yaml或者.eslintrc.yml

JSON：.eslintrc.json，并且此文件允许使用JS形式的注释

废弃的用法：.eslintrc，此文件可以是JSON或者YAML

package.json：在package.json文件中创建eslintConfig属性，所有的配置包含在此属性中。

这些文件的优先级则是按照以上出现的顺序（.eslintrc.js>.eslintrc.yaml>.eslintrc.yml>.eslintrc.json>.eslintrc> package.json）。

可以被配置的信息主要分为3类：

Environments：你的javascript脚本将要运行在什么环境（如：nodejs，browser，commonjs等）中。

Globals：执行代码时脚步需要访问的额外全局变量。

Rules：开启某些规则，也可以设置规则的等级。

## 2.5 MongoDB数据库介绍

随着互联网技术的飞速发展，数据量呈现爆炸性增长趋势，对互联网中应用的数据库提出了高并发读写、高效率存储以及高可扩展性和高可用性的需求，传统数据库越来越显现出力不从心。MongoDB以其独特的优势为WEB应用提供了可扩展的高性能数据存储解决方案。MongoDB的主要特点是开源设计、高性能、易部署、易使用，存储数据方便，它的面向集合存储、模式自由、支持动态查询、支持完全索引等优势正在被越来越多的公司和个人所接受[2]。

作为一个基于分布式文件存储的数据库。其由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。

MongoDB是一个介于[关系数据库](https://baike.baidu.com/item/%E5%85%B3%E7%B3%BB%E6%95%B0%E6%8D%AE%E5%BA%93)和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似[json](https://baike.baidu.com/item/json)的[bson](https://baike.baidu.com/item/bson)格式，因此可以存储比较复杂的数据类型。Mongo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，而且还支持对数据建立[索引](https://baike.baidu.com/item/%E7%B4%A2%E5%BC%95)。

它的特点是高性能、易部署、易使用，存储数据非常方便

主要功能特性有：

面向集合存储，易存储对象类型的数据。

模式自由。

支持动态[查询](https://baike.baidu.com/item/%E6%9F%A5%E8%AF%A2)。

支持完全索引，包含内部对象。

支持查询。

支持复制和故障恢复。

使用高效的二进制数据存储，包括大型对象（如视频等）。

自动处理碎片，以支持云计算层次的扩展性。

支持[RUBY](https://baike.baidu.com/item/RUBY)，[PYTHON](https://baike.baidu.com/item/PYTHON)，[JAVA](https://baike.baidu.com/item/JAVA)，[C++](https://baike.baidu.com/item/C%2B%2B)，[PHP](https://baike.baidu.com/item/PHP)，[C#](https://baike.baidu.com/item/C%23)等多种语言。

文件存储格式为BSON（一种JSON的扩展）。

可通过[网络](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C)访问。

### 2.5.1 MongoDB数据库的部署

32位系统在部署MongoDB时会遇到很多问题，尤其是其默认存储引擎为64位，部署时需要注意一些细节：

1. 建数据库文件的存放位置，比如E:\mongodb\data\db。启动mongodb服务之前需要必须创建数据库文件的存放文件夹，否则命令不会自动创建，而且不能启动成功。
2. 打开cmd（windows键+r输入cmd）命令行，进入到安装目录的bin目录，在安装目录bin文件夹下（shift+右键+在此处打开命令窗口），直接进入到bin目录
3. 输入如下的命令启动mongodb服务：mongod --dbpath E:\mongodb\data\db

注意了，问题来了，storageEngine报错（如图2.3）

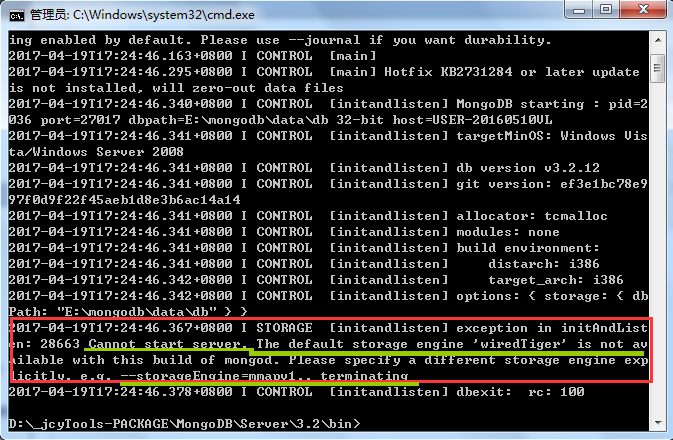


图2.3 数据库部署时存储引擎报错处理

认真读后，发现问题所在 --> storageEngine报错

原因表述：

在2015/3/17以前，MongoDB只有一个存储引擎，叫做MMAP，MongoDB3.0的推出使得MongoDB有了两个引擎：MMAPv1和WiredTiger。

\*MMAPv1：适应于所有MongoDB版本，MongoDB3.0的默认引擎

\*WiredTiger：仅支持64位MongoDB

MongoDB两种引擎可以互相切换，我们可以在创建服务的时候指定引擎类型。

（所以默认的储存引擎并不支持32位）

解决办法：重新运行命令，只不过这回我们要在后边再加一条引擎参数选择

命令如下：mongod –dbpath E:\mongodb\data\db --storageEngine=mmapv1

看到如下信息，即为成功，如图2.2：

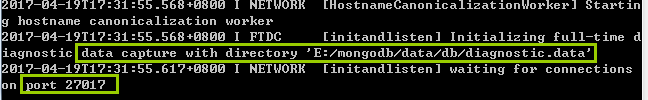


图2.2 部署成功图

当mongod.exe被关闭时，mongo.exe就无法连接到数据库了，因此每次想使用mongodb数据库都要开启mongod.exe程序，所以比较麻烦，此时我们可以将MongoDB安装为windows服务.

为了方便不用每次都自己手动启动MongoDB服务，建议将MongoDB设置成Windows服务，这样，每次开机MongoDB就会自动启动了。

首先，在E:\mongodb目录下，与data同级目录内创建mongodb.config文件，在E:\mongodb\data目录下，与db同级目录内创建log文件夹，并在log文件夹下创建mongodb.log文件：mongodb.config文件内容为：dbpath=E:\mongodb\data\db logpath=E:\mongodb\data\log\mongodb.log

然后用管理员身份打开cmd命令行，进入MongoDB安装目录的bin目录下，输入如下的命令：mongod --config :\mongodb\mongodb.config也可以是命令mongod --config E:\mongodb\mongodb.config --install --serviceName "MongoDB"

结果存放在日志文件中，查看日志发现已经成功。如果失败有可能没有使用管理员身份，遭到拒绝访问。

到此数据库便可成功的部署在本地，并保持开机自启动，可以保证系统随时与数据库的连接。

## 2.6 本章小结

本章通过对基于AI语音识别聊天室系统的开发工具及平台兼容性进行介绍，使我们对整个系统的前端和后端设计大概流程有了一个清晰的理解，“工欲善其事必先利其器”，在系统开发前，了解好系统的一些工具及其兼容性，使用社区支持度广泛，组件开发程度较高的工具能够让我们在开发时事半功倍。通过sublime进行编码，React Developer Tools进行ui界面的调试和数据渲染查看，nodejs进行后端服务器代码编写，webpack实时打包，eslint规范规范和格式矫正，MongoDB数据库存储部署等等，可以让我们对前期的系统编写工作有一个大概的认识，在不断完善中保持稳定持续的代码迭代，使系统一步步完善，直至完成整个系统。

# 第3章 系统分析与设计

## 3.1 系统需求分析

需求分析是软件计划阶段的重要活动，也是软件生存周期中的一个重要环节，该阶段是分析系统在功能上需要“实现什么”，而不是考虑如何去“实现”。需求分析的目标是把用户对待开发软件提出的“要求”或“需要”进行分析与整理，确认后形成描述完整、清晰与规范的文档，确定软件需要实现哪些功能，完成哪些工作。此外，软件的一些非功能性需求(如软件性能、可靠性、响应时间、可扩展性等)，软件设计的约束条件，运行时与其他软件的关系等也是软件需求分析的目标。

本系统是基于AI语音识别的websocket聊天室，所以需求分析的内容是针对待开发软件提供完整、清晰、具体的要求，确定软件必须实现哪些任务。

本系统主要的功能是通过AI语音识别聊天内容并采用websocket传输方式，进行实时数据交流，并使用机器人将语音识别的内容进行智能回复。

主要的业务是:

1. 调用HTML5标准录音接口对接收到的音频进行接收处理，分离，转换，解码成可以进行语音识别的音频内容。
2. 调用语音识别接口，对转换成功的音频文件进行语音识别。
3. 采用websocket进行实时的数据传输，保证数据传输的实时性。
4. 调用机器人接口对语音识别成功的内容进行智能回复。

系统的功能需求是:

1. 登录注册页面的成功实现
2. 录音功能的实现
3. 聊天室功能的实现
4. 系统UI界面友好，交互性好，易于操作

## 3.2系统概要设计

### 3.2.1 系统结构设计

系统结构图，如图3.1所示：

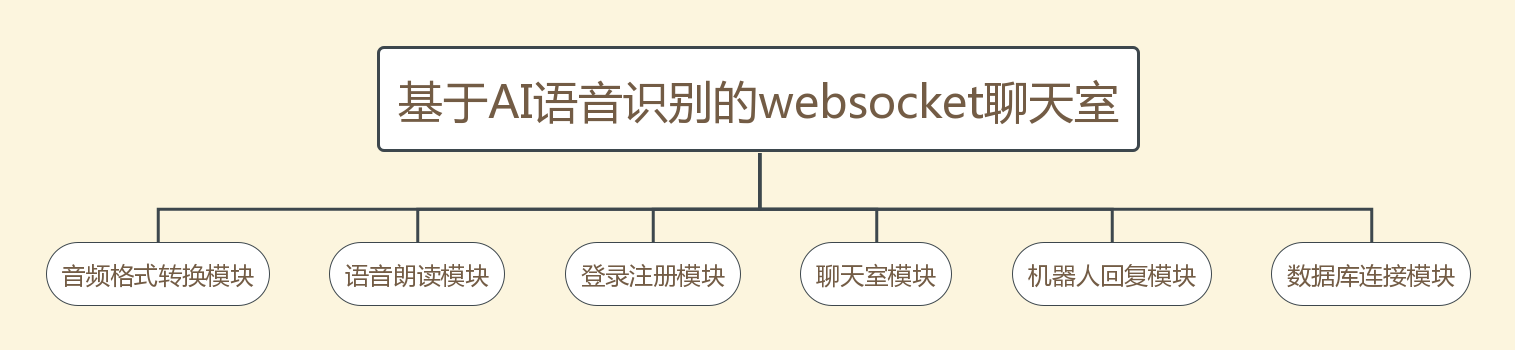


图3.1 系统结构图

## 3.3 数据库设计

MongoDB是一个基于分布式文件存储的数据库。由C++语言编写。旨在为WEB应用提供可扩展的高性能数据存储解决方案。  
MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是它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，而且还支持对数据建立索引。

MongoDB本身不强制使用schema，但是我们在做一个服务应用的时候，应该要知道数据应该怎么组织。

MongoDB的数据库与关系型数据库一样，也存在一对多、多对多等关系。关系型数据库设计的时候有数据库设计范式（1NF，2NF，3NF，BCNF等等），非关系型数据库要思考的问题则不太一样：  
1，关系型数据库的数据基本单位是行和列，MongoDB的数据基本单位是BSON文档；  
2，关系型数据库一般支持adhoc（即席查询）以及连接；MongoDB支持ad hoc，但不支持连接；  
3，关系型数据库允许使用SQL进行复杂的更新，可以在事务中包含多个更新并支持原子性和回滚；MongoDB不支持事务，但支持更新复杂结构的文档数据；  
4，MongoDB无需指定id生成策略，会自动生成\_id；  
5，MongoDB无法与Hibernate之类的ORM框架结合使用；因为文档本身已经是类对象的表示形式了。

MongoDB没有显式地创建数据库的方式，而是会在第一次写入数据的时候创建数据库。创建数据库的时候，MongoDB会在磁盘上分配一系列数据库文件集合，包括所有的集合、索引，以及其他元数据。

数据库文件存储在MongoDB启动时的dbpath参数指定的目录文件夹中；如果不指定dbpath，则会默认在/data/db文件夹下存储。

.ns文件用来表明命名空间；数据库中每个集合和索引的元数据都有自己的命名空间文件，组织形式是哈希表。.ns文件固定大小16MB，意味着数据库中集合和索引的数目之和不能超过26000。

MongoDB还为集合和索引在文件里分配空间，从0开始以递增的方式作为后缀。

在MongoDB中，数据的表示方式有内嵌和引用两种。

“引用”我们比较好理解，是指将不同实体的数据分散不到不同的集合中，而在关系型数据库设计中就是将实体分别建立相应的模型表。如常见的“老师-学生”，“产品-标签”关系，只要实体间存在关系，就可以使用“引用”思想。

“内嵌”是一种反范式化的设计，指的是将每个文档所需的数据都嵌入到文档内部，我想举一个“用户-账户”的关系。我们知道在领域驱动设计中，“用户”是一个聚合根，每个用户对应一个账户，所以是“1对1”的一种关系，在关系型数据库设计中，大部分时候都会将这两者严格区分开来。但是在MongoDB中，却不然，我们可以直接选择将“用户”需要的“账户”数据内嵌到用户文档中，便于我们的增删改查。这是一种反范式化的设计。

设计MongoDB数据模型的时候，我们需要转变以往设计关系型数据模型时的思维。即便是针对一个关系中不同集合的数量规模，我们的模型也将有很大的不同。所以我们需要遵守以下几条准则：

第一条准则：

抛弃关系型数据库设计的范式约束，摒弃关联查询。先考虑内嵌形式，再考虑引用，视使用场景而定。内嵌就是充分利用MongoDB的文档特定，通过嵌套文档的形式，将一组数据统一保存在一个文档下。即一条记录中，这样在列表类的需求中，就不需要多表查询，以及外键关联。

第二条准则：

文档中不是每个字段都必须有值，也就是每行的字段可以不一致。控制字段尽量不插入null值和空值，这样可以节约内存存储，MongoDB中的稀疏索引类型专门为【不是每个文档都有的字段】而设计。

这种特性适合Iot数据采集类似的使用场景，每个文档的字段数目不等，按需插入。

注意这种情况下，切忌文档过宽。那如何避免这种情况，我的方法是预估最大字段数，以20个字段为节点，多于20则采用嵌套document的设计方式组织document。

第三条准则：

时间可以直接定义为格式化的时间，便于识别和查询。不必特意存储时间戳，这样方便可视化的工具查询核对。

"create\_time":ISODate("2017-05-10T15:39:58.000+08:00"),

当然如果系统涉及到不同时区的国际化，最好把原始时间戳记录下来，视情况是否记录源时区。

第四条准则

字段长度尽可能的短，不宜过长。也是考虑到内存优化。MongoDB存储原则中会把key也存储到内存中，所以字段因尽可能的短，这样势必会减低字段的可读性，是的，这里需要牺牲字段的可读性。

## 3.4 本章小结

本章主要对系统进行了详细的分析与设计，通过对系统进行一系列详细的需求分析，深度剖析了系统的主要业务和功能需求。通过系统的概要设计，能使我们比较清晰的把握系统设计的脉络，了解系统的大概设计流程。通过系统结构图可以展示本系统的主要结构，包括其主要的功能模块，例如音频转换模块，语音朗读模块，登录注册模块等等。其次对数据库的设计原则剖析，使系统在开发过程中有了一个完整的设计方向，通过数据库的建模设计，可以优化数据库查询方案，提升开发效率。

# 第4章 系统的具体实现

## 4.1 前端的具体实现

### 4.1.1 音频转换模块

大家都承认是一个数码时代，为追求优良的[音质](https://baike.baidu.com/item/%E9%9F%B3%E8%B4%A8)很多人做出了不懈的努力。随着数码时代的来临，[数字信号](https://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E4%BF%A1%E5%8F%B7)比[模拟信号](https://baike.baidu.com/item/%E6%A8%A1%E6%8B%9F%E4%BF%A1%E5%8F%B7)优越已成为共识。什么是模拟信号？其实任何我们可以听见的声音经过音频线或[话筒](https://baike.baidu.com/item/%E8%AF%9D%E7%AD%92)的传输都是一系列的模拟信号。模拟信号是我们可以听见的。而数字信号就是用一堆数字记号(其实只有[二进制的](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6%E7%9A%84)1和0)来记录声音，而不是用物理手段来保存信号（用普通磁带录音就是一种物理方式）。我们实际上听不到数字信号。

数码音频是我们保存声音信号，传输声音信号的一种方式，它的特点是信号不容易损失。

[数码录音](https://baike.baidu.com/item/%E6%95%B0%E7%A0%81%E5%BD%95%E9%9F%B3)最关键一步就是要把模拟信号转换为数码信号。就电脑而言是把模拟声音信号录制成为wav文件，这个工作HTML5的部分接口也可以做到，但是它的功能十分有限，不能满足我们的需求。

在本次系统开发中，用户通过点击聊天室中的录音按钮可以录入自己当前的声音，可如果是要对其进行语音识别，则对当前的音频质量有着严格要求，音频质量不清晰或者音频声音过高都有可能导致语音识别出错。同时，语音识别对格式也有着一定的严格，而由于HTML5内置的录音API在各个浏览器会出现兼容性问题，录音质量内容和格式都会出现些许偏差，会导致语音识别的不稳定。

所以为了更好的兼容语音识别模块，本次系统设计了音频转换模块，通过JavaScript对其进行封装，可以更好的完善语音识别功能。

录制出来的文件就是wav格式，描述wav格式主要有两个指标，一个是采样频率，或称[采样率](https://baike.baidu.com/item/%E9%87%87%E6%A0%B7%E7%8E%87)、采率，另一个是采样精度也就是[比特率](https://baike.baidu.com/item/%E6%AF%94%E7%89%B9%E7%8E%87)。

对音频进行编码的函数：

encodeWAV: function () {

// 写入采样数据

if (sampleBits === 8) {

for (let i = 0; i < bytes.length; i++ , offset++) {

var s = Math.max(-1, Math.min(1, bytes[i]));

var val = s < 0 ? s \* 0x8000 : s \* 0x7FFF;

val = parseInt(255 / (65535 / (val + 32768)), 10);

data.setInt8(offset, val, true);

}

} else {

for (let i = 0; i < bytes.length; i++ , offset += 2) {

let s = Math.max(-1, Math.min(1, bytes[i]));

data.setInt16(offset, s < 0 ? s \* 0x8000 : s \* 0x7FFF, true);

}

}

return new Blob([data], { type: 'audio/wav' });

}

### 4.1.2 语音识别模块

语音识别模块主要调用百度AI开放平台接口，在聊天室模块中主要通过内部状态设置语音识别的开始和结束，通过模块的状态切换，动态的向后端服务器发送fetch请求（一种基于ajax的异步请求交互技术，HTML5的新型API,）。通过服务器语音识别结果返回，同时异步刷新本地数据，将识别结果展示在前端页面。

在react组件中设置语言识别状态：

setState (status) {

if( status === 'recording') {

this.recording = true

this.recognizing = false

} else if( status === 'recognizing') {

this.recognizing = true

} else {

this.recording = false

this.recognizing = false

}

}

发送语言到后台服务器进行语音识别：

emitMes() {

var \_this = this;

if (\_this.recognizing) {

return

}

if (! \_this.recording) {

window.H5Recorder.init(function (rec) {

\_this.recorder = rec

\_this.recorder.start()

})

\_this.setState('recording')

} else {

let buffer = \_this.recorder.getBlob()

this.socket.emit('chat message', buffer)

\_this.setState('recognizing')

}

### 4.1.3 语音朗读模块

HTML5语音Web Speech API HTML5中和Web Speech相关的API实际上有两类，一类是“语音识别(Speech Recognition)”，另外一个就是“语音合成(Speech Synthesis)”，这两个名词听上去很高大上，实际上指的分别是“语音转文字”，和“文字变语音”。

“语音识别”和“语音合成”看上去像是正反两方面，应该带有镜面气质，实际上，至少从兼容性来看，两者并无法直接对等。

“语音识别(Speech Recognition)”目前的就Chrome浏览器和死忠小弟Opera浏览器默认支持，并且需要webkit私有前缀，如图4.1：（本文不调用此Web Speech API，语音识别有另外的API可寻。）

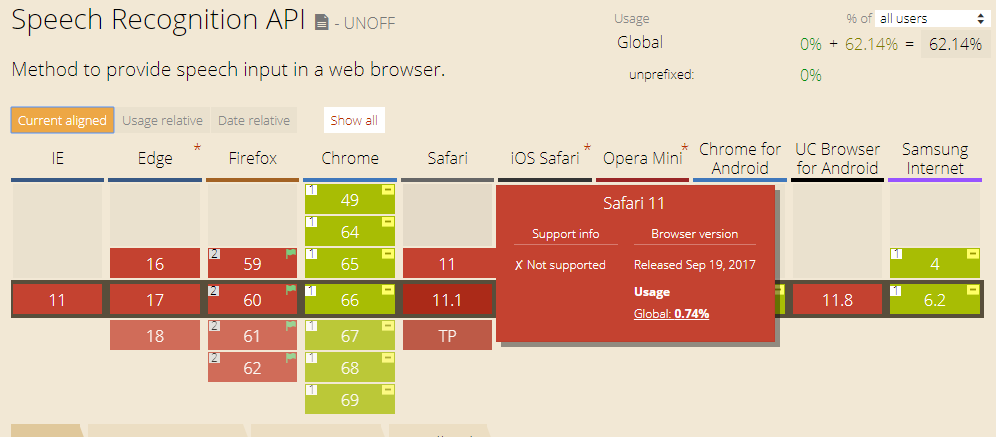


图4.1 Speech Recognition兼容性

“语音合成(Speech Synthesis)”的兼容性要好上太多了，如图4.2：

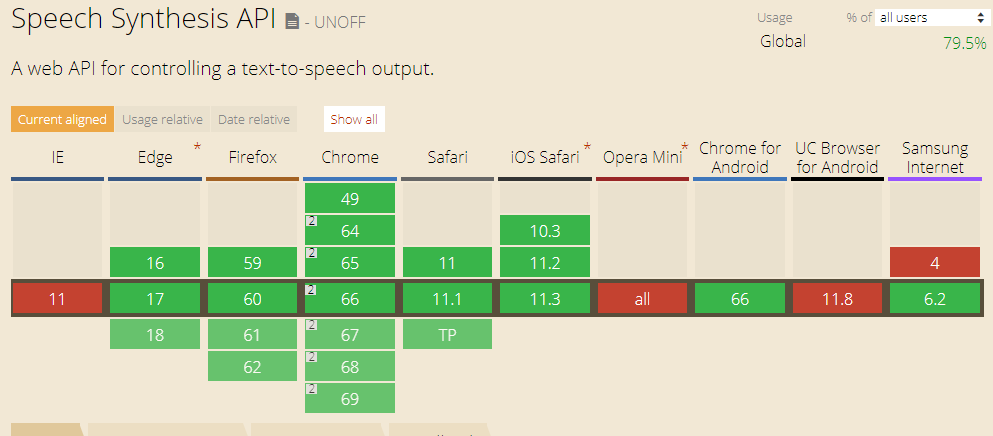


图4.2 Speech Synthesis兼容性

语音朗读模块主要调用Web Speech API的语音合成API，进行语音的朗读：

function synthVoice(text) {

// 声明speechSynthesis对象

const synth = window. speechSynthesis

// 新建语音合成实例

const utterance = new SpeechSynthesisUtterance()

// 设置朗读语音为中文

utterance.lang = 'zh';

utterance.text = text

synth.speak(utterance)

}

并且语音合成需要系统内置TTS模块，如果当前操作系统为精简版操作系统或者系统未安装TTS模块，将导致语音合成失败。

### 4.1.4 登录注册模块

登录注册模块是前端页面的入口，用户通过注册登录进入系统，是整个系统的门面。登录注册模块主要采用ant.design框架进行结构，并采用动画绘制背景。请求发送通过fetch向后端服务器发送，前端收到服务器端验证之后实行react-router的自动跳转到聊天室页面。

登录注册页面背景采用css3技术绘制动画：

@keyframes change-bac {

50% {

background-image: url(../imgs/login1.jpg) ;

background-size:100% 100%;

background-attachment: fixed;

background-position: center center;

background-repeat: no-repeat;

}

100% {

background-image: url(../imgs/login2.jpg) ;

background-size:100% 100%;

background-attachment: fixed;

background-position: center center;

background-repeat: no-repeat;

}

}

登录页面和注册页面通用一个模块：

class Login extends Component {

render() {

return (

<div className="background">

<div className="login">

<LoginForm history = { this.props.history } />

</div>

</div>

);

}

}

export default Login;

登录注册两个页面使用ant.design框架的form表单

采用fetch模块获取数据，并向后台服务器发送数据获取相应：

fetch('http://localhost:8000/login', {

method: "POST",

headers: {

"Content-Type": "application/json"

},

body: JSON.stringify(values)}).then((res) => {

return res.json();

}).then((res) => {

// 返回响应成功 注册成功

if (res.success) {

console.log(res)

openNotification(res.info + '将自动跳转');

setTimeout(() => \_this.props.history.push('/talking'), 2000);

}else {

openNotification(res.info);

setTimeout(() => \_this.props.history.push('/register'), 2000);

}

});

### 4.1.5 聊天室模块

聊天室模块使用react+ant.design设计，前端在模块初始化时向后台获取用户的基本信息，包括聊天室的聊天人数，聊天的历史记录，聊天的搜索内容等等，父组件和子组件之间的通信采用react的props传输，内部状态通过this.state保存：

constructor(props) {

super(props);

this.state = {

talker: ''

}

}

changeTalker(talker) {

console.log(talker);

this.setState({talker: talker})

// this.talker = talker;

}

## 4.2 后端express + websocket设计

### 4.2.1 websocket设计模块

了解计算机网络协议的人，应该都知道：HTTP 协议是一种无状态的、无连接的、单向的应用层协议。它采用了请求/响应模型。通信请求只能由客户端发起，服务端对请求做出应答处理。

这种通信模型有一个弊端：HTTP协议无法实现服务器主动向客户端发起消息。

这种单向请求的特点，注定了如果服务器有连续的状态变化，客户端要获知就非常麻烦。大多数Web应用程序将通过频繁的异步JavaScript和XML（AJAX）请求实现长轮询。轮询的效率低，非常浪费资源（因为必须不停连接，或者HTTP连接始终打开）

Web浏览器和服务器都必须实现WebSockets协议来建立和维护连接。由于 WebSockets连接长期存在，与典型的HTTP连接不同，对服务器有重要的影响。

基于多线程或多进程的服务器无法适用于WebSockets，因为它旨在打开连接，尽可能快地处理请求，然后关闭连接。任何实际的WebSockets服务器端实现都需要一个异步服务器。

websocket模块采用koa-socket进行开发，它是搭载在koa框架上的一款socket模块，使用koa-socket能实现websocket的全部功能。

开始先在koa上装载app:

io.attach(app)

然后开始监听：

io.on('connection', (socket) => {

console.log('a user connected')

})

再对事件进行监听，以便能及时的收到用户的消息：

io.on('chat message', (socket) => {})

### 4.2.2 语音识别接口模块

语音识别使用baidu-ai的SDK，通过调用baidu-ai的接口，可以很方便的进行语音的智能识别并返回可处理的错误信息。

首先使用之前需要在baidu-ai注册当前的app\_id，通过app\_id可以拿到当前app的api\_key和secret\_key，然后在连接语音识别时注册当前app的客户端：

const client = new AipSpeech(APP\_ID, API\_KEY, SECRET\_KEY)

然后上传到baidu-ai进行语音识别：

client.recognize(voiceBuffer, 'wav', 16000, { lan: 'zh' })

.then(function (res) {

console.log('This is what you say')

console.log(res)

if (res.err\_no === 0) {

let text = res.result[0]

SendChat(text)

.then((res) => {

console.log('This is return message')

console.log(res.text)

console.log(res)

io.broadcast('bot reply', { say: text, replay: res.text })

})

.catch((err) => {

console.error(err)

})

### 4.2.3 机器人回复模块

目前聊天机器人根据对话的产生方式，可以分为基于检索的模型（Retrieval-Based Models）和生成式模型（Generative Models）。

生成式模型不依赖预先定义的回答集，而是根据输入的问句及上下文，产生一个新的回答。

聊天机器人的这两条技术路线，从长远的角度看目前技术还都还处在山底，两种技术路线的异同和优势如下：

1. 基于检索的模型的优势
2. 答句可读性好
3. 答句多样性强
4. 出现不相关的答句，容易分析、定位bug

但是它的劣势在于：需要对候选的结果做排序，进行选择基于生成式模型的优势：

1. 端到端的训练，比较容易实现
2. 避免维护一个大的 Q-A 数据集
3. 不需要对每一个模块额外进行调优，避免了各个模块之间的误差级联效应

机器人回复模块主要通过调用图灵机器人接口，向服务器发送语音识别内容，接收到的信息返回给前端服务器，用以实现语音识别并智能回复功能。

基于检索的模型有一个预先定义的回答集，我们需要设计一些启发式规则，这些规则能够根据输入的问句及上下文，挑选出合适的回答。

机器人回复是语音识别聊天室的另一个核心功能，主要调用图灵机器人接口，图灵机器人接口是通过自己在图灵机器实验室自行管理的一个项目实例，语音识别的内容可以通过系统自动回复或者选择自己进行模式训练。

使用机器人接口需要在图灵机器人网站管理自己的项目实例，注册项目所需的一些了方法，key，info，和userid。

首先定义图灵机器人配置文件：

const request = require('request-promise')

function SendChat (text) {

const textTpl = {

'key': '3cadbeb78b354734afc88bd0a13daa26',

'info': text,

'userid': '205216'

}

let options = {

method: 'POST',

uri: 'http://www.tuling123.com/openapi/api',

body: textTpl,

json: true

}

return request(options)

}

module.exports = SendChat

然后在导出的SendChat的方法中使用接口获取机器人实例结果：

SendChat(text)

.then((res) => {

console.log(res.text)

console.log(res)

})

### 4.2.4 登录注册模块

登录注册模块主要定义在express中的route文件夹中，然后在导出到express服务器中进行路由注册。

登录注册后台服务器主要进行用户的注册和登录使用，其中对用户名密码等关键性信息进行加密处理，加密处理主要调用npm的crypto模块进行密码加密，保证用户信息安全。

加密模式采用md5加密：

password : md5.update(req.body.password).digest('base64')；

登录注册使用的方法主要包括：

// 查询用户方法

async function findUser (db, collectionName,query) {

const collection = db.collection(collectionName);

const result = await collection.find(query).count()

return result;

}

// 插入user方法

async function insertUserInfo (db, collectionName, doc) {

const collection = db.collection(collectionName);

const { result } = await collection.insertOne(doc);

return result;

}

### 4.2.5 聊天室模块

聊天室模块是在用户登录注册成功后进行跳转到的页面时需要使用的后台模块，在用户进行聊天室模块后，会自动向后台服务器请求用户信息相关的信息，包括聊天人的消息内容，机器人回复的消息内容的等等。

async function connectDB(url, dbName) {

console.log('ready to connect db');

const client = await MongoClinet.connect(url);

const db = client.db(dbName)

return db;

}

// 插入user方法

async function insertUserInfo (db, collectionName, doc) {

const collection = db.collection(collectionName);

const { result } = await collection.insertOne(doc);

return result;

}

### 4.2.6 数据库连接模块

数据库主要使用NO-SQL式的MongoDB数据库，在express服务框架中连接数据库时主要使用mongodb数据库模块。

首先设置数据库配置：

const url = 'mongodb://localhost:27017';

const dbName = 'talking-room';

const collectionName = 'users'；

初始化数据库：

const MongoClinet = require('mongodb').MongoClient;

进行数据库连接:

MongoClinet.connect(url, async function (err, client) {

console.log('connect db successfully');

const db = await client.db(dbName);

}

关闭数据库操作：

client.close();

console.log('关闭数据库') ;

## 4.3 本章小结

本章主要介绍了系统的具体实现，包括前端核心的音频转换，语音识别，语音朗读，登录注册和聊天室模块。通过对对前端的模块划分，可以使我们很清晰的了解到系统各个模块的分工，同时对各个模块的编码又使整个前端的功能更加完善。

服务器端详细的websocket设计，语音识别接口，机器人回复，服务器端登录注册验证，聊天室和数据库连接模块可以使服务端的处理更加优雅。服务端是整个系统的支柱。保证了数据的可靠和及时有效。通过前端的数据请求，后端服务器接收到响应并对数据进行及时处理，能够实现系统的稳定。

# 第5章 系统测试

系统测试，英文是System Testing。是将已经确认的软件、计算机硬件、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试，系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不符或与之矛盾的地方，从而提出更加完善的方案。系统测试发现问题之后要经过调试找出错误原因和位置，然后进行改正。是基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。对象不仅仅包括需测试的软件，还要包含软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等

## 5.1 软件测试的目的

软件测试是程序的一种执行过程，目的是尽可能发现并改正被测试软件中的错误，提高软件的可靠性。它是软件生命周期中一项非常重要且非常复杂的工作，对软件可靠性保证具有极其重要的意义。在目前形式化方法和程序正确性证明技术还无望成为实用性方法的情况下，软件测试在将来相当一段时间内仍然是软件可靠性保证的有效方法。软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成软件开发项目。不足的测试势必使软件带着一些未揭露的隐藏错误投入运行，这将意味着更大的危险让用户承担。过度测试则会浪费许多宝贵的资源。到测试后期，即使找到了错误，然而付出了过高的代价。E.W.Dijkstra的一句名言说明了这一道理：“程序测试只能表明错误的存在，而不能表明错误不存在。”可见，测试是为了使软件中蕴涵的缺陷低于某一特定值，使产出、投入比达到最大。

## 5.2软件测试的方法

### 5.2.1 可用性测试

英文是Practical Usability Testing。可用性测试是对“用户友好性”的测试。显然这是主观的，且将取决于目标最终用户或客户。用户面谈、调查、用户对话的录象和其他一些技术都可使用。程序员和测试员通常都不宜作可用性测试员。

### 5.2.2 压力测试

英文是Stress Testing。和负载测试差不多。压力测试是一种基本的质量保证行为，它是每个重要软件测试工作的一部分。压力测试的基本思路很简单：不是在常规条件下运行手动或自动测试，而是在计算机数量较少或系统资源匮乏的条件下运行测试。通常要进行压力测试的资源包括内部内存、CPU 可用性、磁盘空间和网络带宽等。一般用并发来做压力测试。

### 5.2.3 负载测试

英文是Load testing。负载测试是测试一个应用在重负荷下的表现。例如测试一个 Web 站点在大量的负荷下，何时系统的响应会退化或失败，以发现设计上的错误或验证系统的负载能力。在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。

负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如，响应时间、事务处理速率和其他与时间相关的方面。

### 5.2.4．健全测试

英文是Sanity testing。健全测试是指一个初始化的测试工作，以决定一个新的软件版本测试是否足以执行下一步大的测试能力。例如，如果一个新版软件每5分钟与系统冲突，使系统陷于泥潭，说明该软件不够“健全”，不具备进一步测试的条件。

## 5.3测试方案

### 5.3.1单元测试

在src中的test文件夹中新建前端及后端测试文件：

1. 新建测试模块，进行单元测试，主要测试模块的连通性和完整性
2. 测试算法的准确性，采用多种数据测试的方式，尽量覆盖多的测试方案

### 5.3.2 白盒测试

主要测试系统相关逻辑：

1. 测试登录注册逻辑判断及其跳转问题
2. 测试数据库写入数据逻辑

### 5.3.3 黑盒测试

测试系统相关功能的具体实现情况：

1. 测试登录注册功能是否完整实现
2. 测试语音识别功能的正确性
3. 测试机器人聊天功能实现
4. 测试语音朗读功能的实现

## 5.4 测试用例

主要列出登录注册模块和聊天室模块的功能测试点（单元测试主要运行在src/test文件中）见表5.1

表5.1登录注册模块和聊天室模块的功能测试点

|  |  |  |  |
| --- | --- | --- | --- |
| 模块 | 测试用例描述 | 操作过程及数据 | 预期结果 |
| 登录注册 | 输入注册数据 | 输入需要注册的用户名密码 | 正常注册 |
| 输入已注册过的用户名 | 输入已注册过的用户名密码 | 提示已经注册过用户名 |
| 输入登录数据 | 输入错误的用户名密码 | 登录错误 |
| 输入未注册的登录数据 | 输入暂未注册的用户名密码 | 提示未注册并跳转注册 |
| 聊天室模块 | 输入声音较低的音频 | 麦克风录音时声音较低 | 返回错误，音频质量太差 |
| 输入声音较高的音频 | 麦克风录音时声音较高 | 返回错误，音频质量太差 |
| 输入正常的音频 | 麦克风录音时正常录音 | 音频质量良好，语音识别正确 |

## 5.5 测试结论

从单元测试和黑盒测试结果中，发现系统能按照预期效果运行成功，语音识别和语音朗读，机器人智能回复等结果对系统的支持良好

## 5.6 本章小结

本章通过一系列测试方法对整个系统进行了完整的测试，其中单元测试，白盒测试和黑盒测试都完整的验证了整个系统的可交付性，通过良好的测试方案，可以表明整个系统在用户使用得当和各个模块都正常运行的情况下，能使系统良好的运行。

# 第6章 总结与展望

基于AI语音识别的websocket聊天室是一套基于C/S模式架构的系统，通过前端react服务+后端express服务进行整个系统的编写，其中用到了很多新兴的技术，包括前端的模块化构建，打包编译工具使用，ServiceWorker前端静态服务缓存，组件化开发，以及后端的路由验证，跨域处理和数据加密等等。通过一系列新兴的技术解决方案，使整个系统的开发步骤可以稳步推进。并且语音识别技术，机器人回复技术都属于现今技术发展的前沿产品，通过在本次系统中加入这些热门技术，可以不断激发自己的学习能力和创造能力。

基于AI语音识别的websocket聊天室核心模块为语音识别功能，机器人回复功能和聊天室功能，采用websocket技术，响应及时，数据交互清晰。前后端的实现都以社区比较稳定的技术框架作为开发基础，保证编码顺利的同时也使结构清楚明了，交互性和可扩展性良好。

没有经过精雕细琢的作品不是好一个作品，在编码后期也发现了本次系统仍然有许多需要完善的部分：

* 1. 增加错误处理解决方案：服务器的突然宕机会使用户的体验性降低，需要增加错误处理机制。
  2. 随着聊天内容的数据量激增，数据库查询操作会带来压力，数据查询缓慢的同时可能会出现请求超时和用户体验性差等情况，需要增加redis缓存机制。
  3. 用户每一次进行系统都需要进行的登录注册操作，这样不仅会增加服务端的查询次数，也会使用户体验性降低。需要增加免登录操作，通过ip识别或者cookie技术，可以使已经登录过的用户再次进行系统时省去登录操作。
  4. 机器人回复内容有时候会略显单一，可以增加关键词查询技术，通过机器人回复的内容中抓取关键词，动态调用其他接口在聊天界面显示动画或者显示用户需要的图片，音乐等等。
  5. PC端对用户的粘性不高，可以开发为微信小程序模式或者APP模式，以便更好的对系统进行使用。

每一个好的产品都是经过不断改进，不断进行版本升级才能更好的进行使用，我们在学习技术的同时也需要注意用户的体验性。不断完善，不断更新才能做出一个好的系统。

# 参考文献

1. 刘智勇.JavaScript 开发技术大全 [M].北京清华大学出版社, 2009.
2. 王光磊.MongoDB数据库的应用研究和方案优化[J].中国科技信息,2011(20):93-94+96.
3. 胡崧. HTML5标准发展现状及我国标准化策略分析[M].北京:中国青年出版社, 2007
4. 吴晓东.[Html5的通信机制及效率的研究](http://www.cnki.com.cn/Article/CJFDTOTAL-CGJM201104045.htm)[J].长春理工大学学报(自然科学版),2011年04期
5. 吴晟.苏庆堂.罗斌.赵莉楠.蔡灿民.[基于Socket和多线程技术的并发服务器的研究](http://www.cnki.com.cn/Article/CJFDTOTAL-KMLG200604009.htm)[J].昆明理工大学学报(理工版),2006年04期
6. 李鑫.[Web服务器性能优化研究与实现](http://cdmd.cnki.com.cn/Article/CDMD-10217-1012266098.htm)[D].哈尔滨工程大学,2011年
7. 薛真真.[基于服务器推送和事件流处理技术的实时Web系统研究](http://cdmd.cnki.com.cn/Article/CDMD-10335-2008066040.htm)[D].浙江大学,2008年
8. 周巍.[基于请求策略和状态切换的服务器推送技术研究](http://cdmd.cnki.com.cn/Article/CDMD-10511-2009159578.htm)[D].华中师范大学,2009年
9. 邱涌.[基于局域网的数据推送](http://www.cnki.com.cn/Article/CJFDTOTAL-GZDN201101030.htm)[J].电脑与电信,2011年01期
10. 罗大晖.陈娟.[基于HTML5的Web离线应用研究与实现](http://www.cnki.com.cn/Article/CJFDTOTAL-JYRJ201212076.htm)[J].计算机应用与软件,2012年12期
11. 郑强.徐国胜.[Websocket在服务器推送中的研究](http://cpfd.cnki.com.cn/Article/CPFDTOTAL-TTTT201208001079.htm)[J].第九届中国通信学会学术年会论文集[C],2012年
12. Li Deng. Dong Yu. “Deep Learning forSignal and Information Processing,” Microsoft Research[J]*, 2013*.
13. RamanArora and Karen Livescu. “Multi-view learning with supervision for transformedbottleneck features,” inProc. ICASSP[J]*, 2014.*
14. Yuan Liu .Tianfan Fu .Yuchen Fan;Yanmin Qian .Kai Yu. Speaker verification with deep features[J], 2014.
15. Improving HMM/DNN in ASR ofunder-resourced languages using probabilistic sampling[J],2015.

# 致谢

光阴似箭，一转眼已是快离开的日子。在校的这几年时间里很感谢学校老师们对我的淳淳教诲，是学校老师们教会了我们勤奋学习，诚实做人，踏实做事，以宽容之心面对生活。指引着我们沿着正确方向前进。在点滴汇聚中使我逐渐形成正确、成熟的人生观、价值观。

值此论文顺利完成之际，特别要感谢我的指导老师宋琦老师，宋琦老师给予我很大的帮助。宋琦老师治学严谨，学识渊博，为我营造了一种良好的精神氛围。在论文的完成过程中，从选题到开题报告，从写作提纲到一遍又一遍地指出每稿中的具体问题，严格把关，循循善诱。同时感谢宋琦老师在我学习期间给予的帮助。

深深的感谢呵护我成长的父母。每当我遇到困难的时候，父母总是第一个给我鼓励的人。回顾20多年来走过的路，每一个脚印都浸满着他们无私的关爱和谆谆教诲，11年的在外求学之路，寄托着父母对我的殷切期望。他们在精神上和物质上的无私支持，坚定了我追求人生理想的信念。父母的爱是天下最无私的最宽厚的爱。大恩无以言报，惟有以永无止境的奋斗，期待将来辉煌的事业让父母为之骄傲。我亦相信自己能达到目标。

另外感谢答辩组的老师，谢谢老师们对我的耐心指导和细心指正。

最后，再次向所有关心我的亲人、师长和朋友们表示深深的谢意。在即将离校之际，祝愿所有老师和同学们健康快乐。

# 附录

## 一：英文原文

## HTML5 WebSocket: A Quantum Leap in Scalability for the Web

Lately there has been a lot of buzz around HTML5 Web Sockets, which defines a full-duplex communication channel that operates through a single socket over the Web. HTML5 Web Sockets is not just another incremental enhancement to conventional HTTP communications; it represents a colossal advance, especially for real-time, event-driven web applications.

HTML5 Web Sockets provides such a dramatic improvement from the old, convoluted "hacks" that are used to simulate a full-duplex connection in a browser that it prompted Google's Ian Hickson—the HTML5 specification lead—to say:

"Reducing kilobytes of data to 2 bytes…and reducing latency from 150ms to 50ms is far more than marginal. In fact, these two factors alone are enough to make Web Sockets seriously interesting to Google."

Let's take a look at how HTML5 Web Sockets can offer such an incredibly dramatic reduction of unnecessary network traffic and latency by comparing it to conventional solutions.

### Polling, Long-Polling, and Streaming—Headache 2.0

Normally when a browser visits a web page, an HTTP request is sent to the web server that hosts that page. The web server acknowledges this request and sends back the response. In many cases—for example, for stock prices, news reports, ticket sales, traffic patterns, medical device readings, and so on—the response could be stale by the time the browser renders the page. If you want to get the most up-to-date "real-time" information, you can constantly refresh that page manually, but that's obviously not a great solution.

Current attempts to provide real-time web applications largely revolve around polling and other server-side push technologies, the most notable of which is Comet, which delays the completion of an HTTP response to deliver messages to the client. Comet-based push is generally implemented in JavaScript and uses connection strategies such as long-polling or streaming.

With polling, the browser sends HTTP requests at regular intervals and immediately receives a response. This technique was the first attempt for the browser to deliver real-time information. Obviously, this is a good solution if the exact interval of message delivery is known, because you can synchronize the client request to occur only when information is available on the server. However, real-time data is often not that predictable, making unnecessary requests inevitable and as a result, many connections are opened and closed needlessly in low-message-rate situations.

With long-polling, the browser sends a request to the server and the server keeps the request open for a set period. If a notification is received within that period, a response containing the message is sent to the client. If a notification is not received within the set time period, the server sends a response to terminate the open request. It is important to understand, however, that when you have a high message volume, long-polling does not provide any substantial performance improvements over traditional polling. In fact, it could be worse, because the long-polling might spin out of control into an unthrottled, continuous loop of immediate polls.

With streaming, the browser sends a complete request, but the server sends and maintains an open response that is continuously updated and kept open indefinitely (or for a set period of time). The response is then updated whenever a message is ready to be sent, but the server never signals to complete the response, thus keeping the connection open to deliver future messages. However, since streaming is still encapsulated in HTTP, intervening firewalls and proxy servers may choose to buffer the response, increasing the latency of the message delivery. Therefore, many streaming Comet solutions fall back to long-polling in case a buffering proxy server is detected. Alternatively, TLS (SSL) connections can be used to shield the response from being buffered, but in that case the setup and tear down of each connection taxes the available server resources more heavily.

Ultimately, all of these methods for providing real-time data involve HTTP request and response headers, which contain lots of additional, unnecessary header data and introduce latency. On top of that, full-duplex connectivity requires more than just the downstream connection from server to client. In an effort to simulate full-duplex communication over half-duplex HTTP, many of today's solutions use two connections: one for the downstream and one for the upstream. The maintenance and coordination of these two connections introduces significant overhead in terms of resource consumption and adds lots of complexity. Simply put, HTTP wasn't designed for real-time, full-duplex communication as you can see in the following figure, which shows the complexities associated with building a Comet web application that displays real-time data from a back-end data source using a publish/subscribe model over half-duplex HTTP.

It gets even worse when you try to scale out those Comet solutions to the masses. Simulating bi-directional browser communication over HTTP is error-prone and complex and all that complexity does not scale. Even though your end users might be enjoying something that looks like a real-time web application, this "real-time" experience has an outrageously high price tag. It's a price that you will pay in additional latency, unnecessary network traffic and a drag on CPU performance.

### HTML5 WebSocket to the Rescue!

bidirectional communications channel that operates through a single socket over the Web. HTML5 Web Sockets provides a true standard that you can use to build scalable, real-time web applications. In addition, since it provides a socket that is native to the browser, it eliminates many of the problems Comet solutions are prone to. Web Sockets removes the overhead and dramatically reduces complexity.

To establish a WebSocket connection, the client and server upgrade from the HTTP protocol to the WebSocket protocol during their initial handshake

Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Both text and binary frames can be sent full-duplex, in either direction at the same time. The data is minimally framed with just two bytes. In the case of text frames, each frame starts with a 0x00 byte, ends with a 0xFF byte, and contains UTF-8 data in between. WebSocket text frames use a terminator, while binary frames use a length prefix.

Note: Although the Web Sockets protocol is ready to support a diverse set of clients, it cannot deliver raw binary data to JavaScript, because JavaScript does not support a byte type. Therefore, binary data is ignored if the client is JavaScript—but it can be delivered to other clients that support it.

### HTML5 WebSocket and the Kaazing WebSocket Gateway

Today, only Google's Chrome browser supports HTML5 Web Sockets natively, but other browsers will soon follow. To work around that limitation, however, Kaazing WebSocket Gateway provides complete WebSocket emulation for all the older browsers (I.E. 5.5+, Firefox 1.5+, Safari 3.0+, and Opera 9.5+), so you can start using the HTML5 WebSocket APIs today.

WebSocket is great, but what you can do once you have a full-duplex socket connection available in your browser is even greater. To leverage the full power of HTML5 Web Sockets, Kaazing provides a ByteSocket library for binary communication and higher-level libraries for protocols like Stomp, AMQP, XMPP, IRC and more, built on top of WebSocket.

### Summary

HTML5 Web Sockets provides an enormous step forward in the scalability of the real-time web. As you have seen in this article, HTML5 Web Sockets can provide a 500:1 or—depending on the size of the HTTP headers—even a 1000:1 reduction in unnecessary HTTP header traffic and 3:1 reduction in latency. That is not just an incremental improvement; that is a revolutionary jump—a quantum leap!

Kaazing WebSocket Gateway makes HTML5 WebSocket code work in all the browsers today, while providing additional protocol libraries that allow you to harness the full power of the full-duplex socket connection that HTML5 Web Sockets provides and communicate directly to back-end services.

## 二：中文翻译

## HTML5 Web Sockets:对于网络扩展性上是一个巨大的飞跃

近来围绕着HTML5 Web Sockets有很大的动静，它定义了一种在通过一个单一的socket在网络上进行全双工通讯的通道。它不仅仅是传统的HTTP通讯的一个增量的提高，尤其对于实时、事件驱动的应用来说是一个飞跃。

HTML5 Web Sockets相对于老的技术（在浏览器中模拟全双工连接的复杂技术）有了如此巨大的提升，以致于谷歌的Ian Hickson—HTML5说明书的总编说：“把数据从kb减少到2b...，延迟时间从150ms减少到50ms，这远远不止是个微调。实际上，仅仅是这两个事实就已经足够让谷歌对Web Sockets产生非常浓厚的兴趣。”

让我们通过对比常规的解决方案来看看HTML5 Web Sockets是如何在非必要的网络传输和延迟性上提供如此难以置信的巨大降低。

### 轮询，长轮询，令人头痛的流式2.0

通常，当浏览器访问web页面时，HTTP请求被发送到承载该页面的web服务器。web服务器确认此请求并发回响应。在许多情况下——例如，股票价格、新闻报道、门票销售、流量模式、医疗设备读数等等——当浏览器呈现页面时，响应可能会变得过时。如果您想获得最新的“实时”信息，您可以不断地手动刷新该页面，但这显然不是一个很好的解决方案。

目前提供实时web应用程序的尝试主要围绕轮询和其他服务器端推送技术展开，其中最值得注意的是Comet，它延迟HTTP响应的完成，以便向客户端发送消息。基于Comet的push通常在JavaScript中实现，并使用长轮询或流等连接策略。

通过轮询，浏览器定期发送HTTP请求，并立即接收响应。这种技术是浏览器提供实时信息的第一次尝试。显然，如果知道消息传递的确切时间间隔，这是一种很好的解决方案，因为只有在服务器上提供信息时才能同步客户机请求。然而，实时数据通常不是那么可预测的，使得不必要的请求不可避免，因此，在低消息率的情况下，许多连接被不必要地打开和关闭。

通过长轮询，浏览器向服务器发送一个请求，服务器将请求打开一个固定的时间段。如果在此期间收到通知，则会将包含消息的响应发送到客户端。如果在设置的时间段内未收到通知，服务器将发送响应以终止开放请求。但是，重要的是要理解，当消息量很大时，长轮询不会比传统轮询提供任何实质性的性能改进。实际上，情况可能更糟，因为长轮询可能会失控，变成一个不受控制的、连续的即时轮询循环。

当你打算从Comet解决方案扩展到更大众方案的时候可能会更糟糕。通过HTTP模拟双向浏览器通信很容易出错，也很复杂，而且所有这些复杂性都无法扩展。尽管您的终端用户可能喜欢看起来像实时web应用程序的东西，但这种“实时”体验的价格高得离谱。您将为此付出额外的延迟、不必要的网络流量和CPU性能的拖累。

### HTML5 WebSocket救星!

通过Web上的单个套接字操作的双向通信通道。HTML5 Web Sockets提供了一个真正的标准，你可以用它来构建可伸缩的、实时的Web应用程序。此外，由于它提供了一个本地浏览器的套接字，它消除了Comet解决方案容易出现的许多问题。Web Sockets消除了开销，并极大地降低了复杂性。

要建立WebSocket连接，客户端和服务器在最初的握手过程中从HTTP协议升级到WebSocket协议

一旦建立，WebSocket数据帧可以在客户端和服务器之间以全双工模式来回发送。文本帧和二进制帧都可以在同一时间以任何方向发送全双工。数据是用最少的两个字节框起来的。在文本帧的情况下，每个帧以0x00字节开始，以0xFF字节结束，中间包含UTF-8数据。WebSocket文本帧使用一个结束符，而二进制帧使用长度前缀。

注意:尽管Web Sockets协议已经准备好支持一组不同的客户端，但是它不能向JavaScript交付原始的二进制数据，因为JavaScript不支持字节类型。因此，如果客户机是javascript，则忽略二进制数据，但是可以将其交付给支持它的其他客户端。

### HTML5 WebSocket和Kaazing WebSocket网关

今天，只有谷歌的Chrome浏览器可以本地支持HTML5 Web Sockets，但是其他浏览器很快就会跟上。为了解决这个限制，Kaazing WebSocket Gateway为所有旧的浏览器提供了完整的WebSocket模拟(例如5.5+、Firefox 1.5+、Safari 3.0+和Opera 9.5+)，所以你现在可以开始使用HTML5 WebSocket api了。

WebSocket很好，但是一旦你的浏览器中有一个全双工套接字连接，你就能做的更大。为了充分利用HTML5 Web Sockets的强大功能，Kaazing提供了一个ByteSocket库用于二进制通信，并为构建在WebSocket之上的Stomp、AMQP、XMPP、IRC等协议提供更高级别的库。

### 最后

HTML5 Web Sockets为实时网络的可扩展性迈出了巨大的一步。正如您在本文中所看到的，HTML5 Web Sockets可以提供500:1——取决于HTTP head的大小——甚至可以减少不必要的HTTP header流量，减少3:1的延迟。这不仅仅是一个渐进的改进;这是一次革命性的跳跃——一次量子飞跃!

Kaazing WebSocket网关使HTML5 WebSocket代码在今天的所有浏览器中都能运行，同时提供了额外的协议库，使您能够充分利用HTML5 Web Sockets提供的全双工socket连接，并直接与后端服务通信。

## 三：关键代码

### 音频转换函数

encodeWAV: function () {

var sampleRate=Math.min(this.inputSampleRate, this.outputSampleRate);

var sampleBits = Math.min(this.inputSampleBits, this.oututSampleBits);

var bytes = this.compress();

var dataLength = bytes.length \* (sampleBits / 8);

var buffer = new ArrayBuffer(44 + dataLength);

var data = new DataView(buffer);

var channelCount = 1;//单声道

var offset = 0;

var writeString = function (str) {

for (var i = 0; i < str.length; i++) {

data.setUint8(offset + i, str.charCodeAt(i));

}

}

// 资源交换文件标识符

writeString('RIFF'); offset += 4;

// 下个地址开始到文件尾总字节数,即文件大小-8

data.setUint32(offset, 36 + dataLength, true); offset += 4;

// WAV文件标志

writeString('WAVE'); offset += 4;

// 波形格式标志

writeString('fmt '); offset += 4;

// 过滤字节,一般为 0x10 = 16

data.setUint32(offset, 16, true); offset += 4;

// 格式类别 (PCM形式采样数据)

data.setUint16(offset, 1, true); offset += 2;

// 通道数

data.setUint16(offset, channelCount, true); offset += 2;

// 采样率,每秒样本数,表示每个通道的播放速度

data.setUint32(offset, sampleRate, true); offset += 4;

// 波形数据传输率 (每秒平均字节数) 单声道×每秒数据位数×每样本数据位/8

data.setUint32(offset,channelCount\*sampleRate \*(sampleBits / 8), true); offset += 4;

// 快数据调整数采样一次占用字节数单声道×每样本的数据位数/8

data.setUint16(offset, channelCount \* (sampleBits / 8), true); offset += 2;

// 每样本数据位数

data.setUint16(offset, sampleBits, true); offset += 2;

// 数据标识符

writeString('data'); offset += 4;

// 采样数据总数,即数据总大小-44

data.setUint32(offset, dataLength, true); offset += 4;

// 写入采样数据

if (sampleBits === 8) {

for (let i = 0; i < bytes.length; i++ , offset++) {

var s = Math.max(-1, Math.min(1, bytes[i]));

var val = s < 0 ? s \* 0x8000 : s \* 0x7FFF;

val = parseInt(255 / (65535 / (val + 32768)), 10);

data.setInt8(offset, val, true);

}

} else {

for (let i = 0; i < bytes.length; i++ , offset += 2) {

let s = Math.max(-1, Math.min(1, bytes[i]));

data.setInt16(offset, s < 0 ? s \* 0x8000 : s \* 0x7FFF, true);

}

}

return new Blob([data], { type: 'audio/wav' });

}

### 机器人配置

const request = require('request-promise')

function SendChat (text) {

const textTpl = {

'key': '3cadbeb78b354734afc88bd0a13daa26',

'info': text,

'userid': '205216'

}

let options = {

method: 'POST',

uri: 'http://www.tuling123.com/openapi/api',

body: textTpl,

json: true

}

return request(options)

}

module.exports = SendChat

### 前端路由转换

class App extends Component {

render() {

return (

<Switch>

<Route path='/index' component={Login} />

<Route path='/talking' component={TalkingRoom} />

<Route path='/recorder' component={Recorder} />

<Route path='/register' component={Register} />

<Redirect to="/index" />

</Switch>

);

}

}

ReactDOM.render(

<BrowserRouter>

<App />

</BrowserRouter>

, document.getElementById('root'));

registerServiceWorker();