华南理工大学
South China University of Technology

本科毕业设计（论文）

基于 Node. js 的社区游戏系统的设计与实现

学　院　　软件学院

专　业　　软件工程

学生姓名　　黄聪

学生学号　　200830630358

指导教师　　彭绍武

提交日期　　年　　月　　日

摘　要

Node.js 是新兴的 JavaScript 虚拟机，基于 V8 引擎实现的事件驱动 IO，性能很好，受众热捧。Node.js 有很多鲜明的特性，包括使用 JavaScript 语言、异步 IO 非阻塞、单线程、事件驱动等。随着 NoSQL 的兴起，MongoDB 也是备受关注。它是功能很丰富，是一个广受好评的最像关系数据库的非关系数据库。本项目是一个基于 MongoDB 和 Node.js 的社区在线游戏，是个类似名叫 Bonbons 桌游的多人在线游戏，让用户选择中央牌池的牌和手牌，图案相同即配对成功，第一个完成所有配对的用户胜利。本项目包括用户管理、游戏大厅、游戏房间三大功能模块。用户管理模块是本项目最基础的一个模块，几乎每一个页面，每一个步骤，都会调用这个模块的接口，这个模块包括用户登录退出和用户信息存储两个小模块。游戏大厅模块是一个相对简单的模块，主要起着衔接的作用，也就是把用户从介绍首页引导到游戏房间的过渡模块，它包括消息同步和引导进入房间两个部分。游戏房间模块是整个游戏最重要的部分，是本项目的灵魂所在，它包含了整个游戏过程的逻辑与实现。该模块同样包含消息同步的部分，还包含会话控制部分以及游戏逻辑部分（各种消息或事件的处理和响应）。

关键词：Node.js；MongoDB；游戏

Abstract

The Node.js powerful new event-driven asynchronous IO JavaScript virtual machine, based on the V8 engine, achieving high performance is focused by a lot of people. Node.js have many distinctive features, including the use of JavaScript language, asynchronous non-blocking IO, single-threaded, event-driven, etc. With the development of the NoSQL, MongoDB also gets many attentions. It is multi-functional and a non-relational database that most like a relational database. It is a very powerful database. This project is a multiplayer online game based on MongoDB and Node.js, similar with a game called Bonbons. Users choose a card from the central pool and a card in his/her pocket to match the picture on the back of the two cards. The first one to complete to match all the cards in the pocket wins. The functional modules in this project include user management, game hall and games room. The user management module is the most basic module of the project. Its interfaces are used in almost every page and every step. This module includes user login, logout and information storage. The game hall module is a relatively simple module and plays the role of a bridge between the index and game rooms. It guides users to game rooms. It has two main parts includes the message synchronization and guide into the room. The game room module is the most important part of the entire game and is the soul of this project. It holds all the logic and the realization of the game. It also contains the message synchronization part, also includes a session control part and the game logic part (handling and reaction of all kinds of messages or event).

Keyword: Node.js, MongoDB, game

目 录

第一章 绪 论

本章主要陈述本项目关键技术的背景和简介，让未接触过 Node.js 和 MongoDB 的读者有大致的了解。

1.1 Node.js 简介

1.1.1 Node.js 的介绍

Node.js ，有时我们将其简称为“Node”，是一种新兴的软件开发平台，它将 JavaScript 从 Web 浏览器移植到常规的服务器端。Github 上 Node.js 项目的网页上有这么一句介绍：Evented I/O for V8 JavaScript。很短，但充分地说出了 Node.js 的特色：基于 V8 引擎的事件驱动 IO。

简单地来说，Node.js 就是把 JavaScript 写在后台，基于 chrome 的 v8 引擎，有好几个明显的特征：异步非阻塞、单线程、事件驱动等等，十分适合用来构建 IO 密集的网站。

1.1.2 Node.js 的发展情况

仅仅诞生三年的 Node.js 发展迅猛，不乏应用案例。微软、eBay、LinkedIn、雅虎、facebook 等国际知名公司及网站均有使用 Node.js 的成功案例。在国内，淘宝是 Node.js 的先锋（淘宝指数、MyFox 数据处理中间件、淘 Job 等），腾讯（朋友网长连接系统等）、新浪等公司及网站也有使用 Node.js 的案例。

1.2 MongoDB 简介

1.2.1 NoSQL 的兴起

NoSQL(Not Only SQL)，意思就是反 SQL，广义定义是非关系型数据库的研究和使用。NoSQL 拥支持者们提倡非关系型的数据存储，区别于现在常用的关系型数据库，这个提倡无疑是一种新思维的注入。非关系型数据库没有需固定的存储结构，一般不存在表连接操作，在大数据存储上在性能方面比关系型数据库有更大的优势。

关系数据库的存储结构是固定的，也就是说每一个元组都有相同的字段，无论那个元组是否填入所有的字段，但是数据库都会给它分配所有字段的空间，因为这样子容易进行表连接等操作，但这是关系型数据性能瓶颈的一大因素。而非关系数据库的结构不

是固定的，也就是说每个元组可以有不一样的属性或者字段，可以根据需要增加一些自己键值对。非关系数据库是以键值对来存储的，存储结构不固定，可以节约时间和空间。

1.2.2 MongoDB 介绍

MongoDB 是最像关系数据库的非关系型数据库，也可以看成是介于关系型与非关系型数据库之间的东西，是一种功能丰富的数据库。MongoDB 具有非关系数据库的优点，例如结构不固定、高性能等等；也具有一般非关系数据库不具有的优点，例如查询语言极其强大，有点像面对对象那样子的查询，几乎可以代替关系数据库的单表查询，而且它还支持索引的建立。

1.3 本章总结

本章介绍了一下几点：

- Node.js 是新兴的后台技术，是一个 JavaScript 的虚拟机，基于 V8 引擎实现的事件驱动 IO，性能很好，受众热捧。
- Node.js 发展迅猛，不乏应用案例。
- MongoDB 是功能丰富，最像关系数据库的非关系数据库。

第二章　项目关键技术介绍

本章将详细的介绍 Node.js 和 MongoDB 以及与其相关的模块组件和托管平台。

2.1 Node.js

2.1.1 Node.js 高并发的性能

Node.js 引起关注的重要原因在于它的吞吐量（以每秒响应的请求数衡量），响应性能好。和同类型产品（例如 Apache）进行基准测试比较，结果显示 Node 的性能提升很大。下面是一个简单的基准测试程序，是一个只返回“Hello World”的 HTTP 服务器。

```
var http = require('http');
http.createServer(function(req, res){
res.writeHead(200, {'Content-Type': 'text/plain'});
res.end('Hello World\n');
}).listen(8124, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8124/');
```

这是一般 Node 介绍网站都会提供的 Hello World 程序，是用 Node 来实现的最简单的 web 服务器。这里 require 的 http 对象封装了 HTTP 协议，createServer 函数能建立一个完整的 web 服务器，而 listen 函数用来监听指定的端口。向这个 web 服务器发的所有请求（包括任何路径的 GET 或者 POST）都会返回一个简单的“Hello World”。正因为它的简单，所以这个程序经常用来测试 Node 请求的最大吞吐量。

Node 作者 Ryan Dahl 曾经演示过一个较为简单的基准测试程序（http://nodejs.org/cinco_de_node.pdf），该程序返回一个 1 MB 的缓冲区（buffer），Node 每秒能处理 822 个请求，而 nginx 每秒处理 708 个。他还指出 nginx 的峰值是 4 MB 内存，而 Node 是 64 MB。

那么目前的服务器程序存在什么问题呢，我们来探讨一下。在 Java 或者 PHP 等语言中，每一次请求都会新建一个线程，而每个新的线程大概需求 2MB 的配套内存。假设这个系统拥有 8GB RAM，理论上支持 4000 个用户的最大并发连接数。若以后你希望你的 web 服务器支持更多的用户，那你就必须添加更多的服务器。当然，这会增加成本，特别是服务器的成本、运输的成本以及人工成本。除了成本提高了以外，在技术上还有个问题：对于每个请求用户可能使用不同的服务器，所以，所有服务器都必须共享所有的共享资源。比如说，在 Java 中，每个服务器上的 JVMs 就必须共享其静态变量和缓存。而这就造成了整个 web 服务框架中吞吐量的瓶颈。

Node.js 用另一种方式解决这个问题，改变连接服务器的方式：为每个连接创建的新进程不需要配套内存块，而不像其他语言或者平台那样为每个连接都创建一个新的 OS 线程（并向其分配一些配套内存）。Node 是单进程、单线程的，声称绝不死锁，因为它根本不需要也不允许使用锁，它不会直接阻塞 IO 的调用。Node 还宣称，用 Node 建立的服务器能支持数万个并发连接，实际上，Node 是通过把系统的瓶颈变成单个系统的流量而不是最大连接数，以这样的方式改变服务器的面貌。

由于本项目和微博接入挂钩，不方便压力测试，所以，在这里，只能附上 CNode 社区中提供的压力测试来说明 Node.js 的高并发量和高吞吐量。

表 1-1 Node.js 压力测试说明

测试环境	测试使用的是 node.js v0.3.1,没有使用任何第三方辅助模块 代理服务和 memcache 部署在不一样的服务器上 系统均为 rhel 5.2, cpu: AMD Opteron 2200, mem: 4g
测试用例	通过这个代理程序，分别采用 memcached 协议以及 http 协议从 memcache 服务中取出一个长度为 100B 的值，并检查最终输出是否正确
压力工具	socket: 由于没有找到合适的工具，所以用 Node.js 实现了一个简单的 socket 压力工具 http: siege 2.70

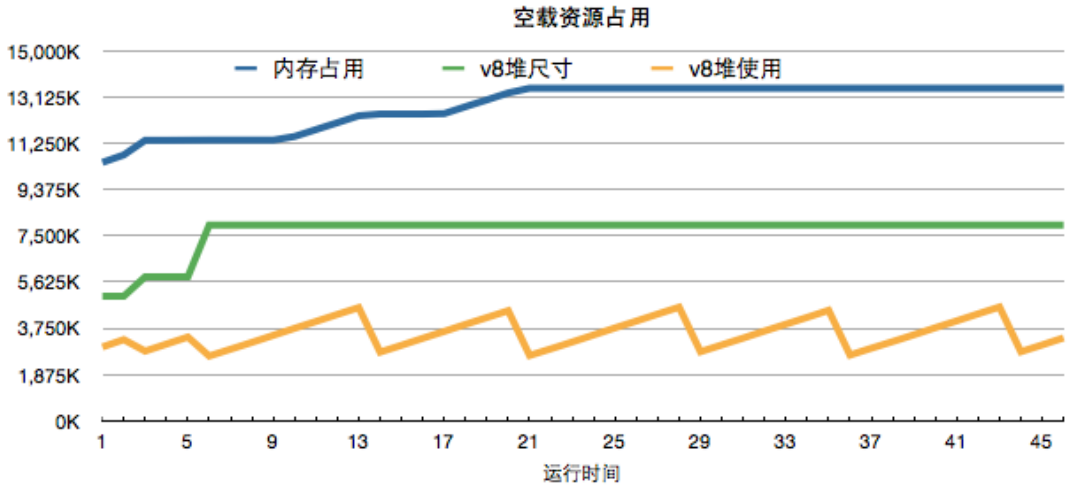

图 1-1 服务运行时间与空载资源的占用

从图 1-1 可知，在程序运行 20 秒后,内存占用达到稳定状态，内存消耗约 13M，v8 堆尺寸约 8M，从堆的使用变化状态可推断 v8 每隔 7 到 8 秒会进行一次 gc 操作。

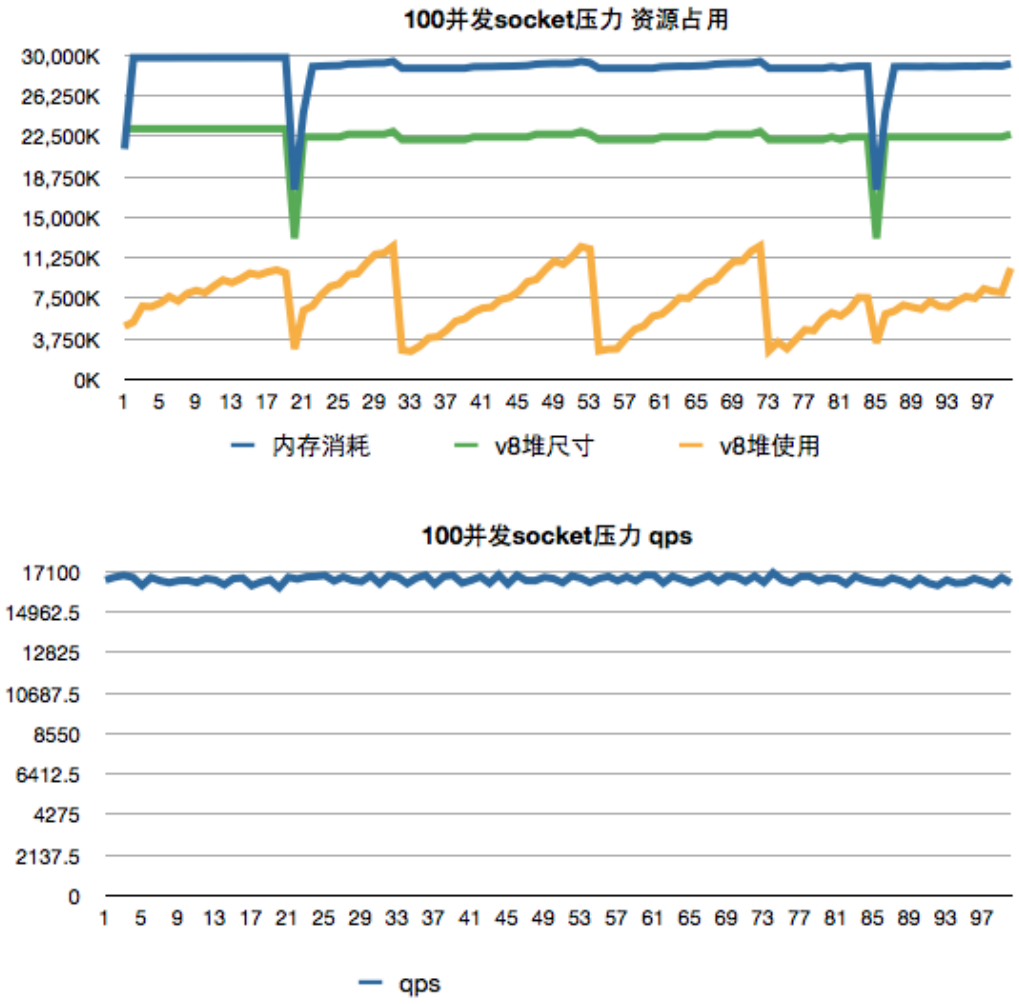

图 1-2 100 并发 100 秒 socket 长连接压力

由图 1-2 可知，压力测试启动后内存占用迅速飙升至 30M，v8 堆尺寸也基本徘徊在 22m 的水平，v8 堆使用率在 20%到 50%之间波动，这个时候 v8 的 gc 操作频率下降到大概 20 秒一次，而 qps 曲线就比较平稳，徘徊在 16700 上下，幅度 400 上下，可见 v8 的 gc 操作对性能影响不大。在整个压力测试的过程中 CPU 占用基本处于满载状态，维持在 95%左右。另外，压力测试结束 20 秒左右，所占用资源就被释放，内存与 v8 堆都恢复到空载水平。

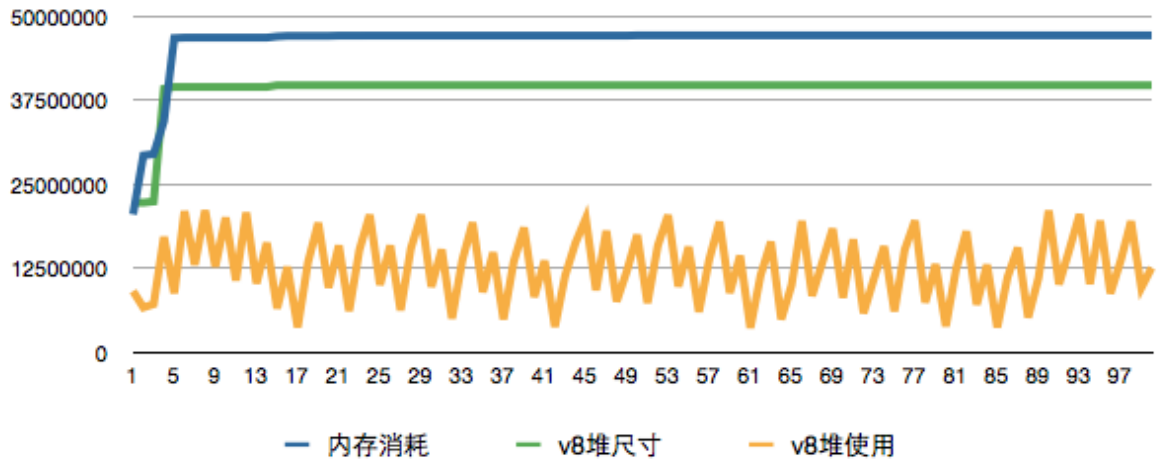

图 1-3 250 并发 100 秒 http 长连接压力

与 socket 相比，http 需要的系统资源大概多出 30%，而且 8v 堆的 gc 操作也比 socket 更加频繁，qps 值约为 4392，gc 操作对 qps 的影响也不大。在整个压力测试过程中 CPU 占用也是基本处于满载状态，维持在 95%左右。和 socket 的结果类似，压力测试结束后 20 秒左右，所占用资源就被释放，内存与 v8 堆都恢复到空载水平。

由压测的结果可知，异步非阻塞 IO 使 Node.js 在系统资源耗用相对低下的情况下表现出高性能和相当出众的负载能力，非常适合 IO 密集型的服务。

2.1.2 Node.js 的语言特性

Node.js 使用的语言是 JavaScript，也就是说，本项目的开发前台和后台都是使用同一种语言 JavaScript。这样的统一，使开发者只需精通一种语言，为前后台都需要一个人完成的庞大工作量省下了不少语言学习熟悉的时间。

目前前端的开发大部分都是关于 JavaScript 的，可以说 JavaScript 无处不在，它拥有现代很多高级语言的概念，并不逊色于其他任何语言。正因为 JavaScript 的流行，软件行业储备了大量有丰富 JavaScript 开发经验的人才。

众所周知，JavaScript 是一门拥有松散类型对象的动态编程语言，而且这种对象还是可动态扩展的，能根据需要非正式地声明。JavaScript 的闭包特性和匿名函数非常适合基于事件驱动的异步的编程。JavaScript 的函数通常作为匿名闭包使用，是一级对象，这使 JavaScript 优于其他编写 Web 应用的语言。理论上，这些特点能使开发者更加高效地工作，但实际上来说，动态与否、是静态行雷还是松散类型语言比较好都没有一个明确的定论，或许永远不会有定论。

JavaScript 最不好的地方可以算是全局对象了，把所有顶级的变量都弄到一个全局对象身上，若使用的模块比较多的时候就会出现很多意外的混乱。由于 web 程序一般都会有很多对象，而且很可能是多个组织或者团队一起编写，或许你会怀疑在 Node 编中，

全局对象的混乱和冲突将会是一个“雷区”。但事实并不是这样的，Node.js 使用的是 CommonJS 模块系统，也就是说模块里面的变量看起来好像是全局变量，但实际上它们却是该模块的局部变量。这样子，模块与模块间就可以很清晰而且自然地区分，避免了全局对象的尴尬。

技术人员们已经盼望很久了，他们一直希望 web 应用的前端（客户端）和后台（服务器）使用同一种编程语言。对于这个梦想甚至可以追溯到很早很早以前的 Java 时代，那时候是用 Java 编写服务器应用前端 Applet，而对于 JavaScript，最初的设想是当作 Applet 的一种比较轻量级的脚本语言。而如今，JavaScript 已经取代了 Java，成为浏览器中使用的唯一一种语言。Node 的出现，是前后台使用同一种编程语言的梦想终于实现了，而这种语言就是 JavaScript。

2.1.3 Node.js 的异步 IO 非阻塞特性

传统模式下的 IO 操作一般是这样的：

```
var data = file.read("file.data");
//等待 io 返回
doSomethingWithData(data);
```

程序逻辑需要等待 IO 操作完成，而在 IO 操作期间，CPU 只能忙等待，资源白白地浪费掉。而所谓 IO 操作，不只是文件读写，还有网络通信等等，这样的忙等待会大大地影响服务器性能，使得吞吐量、处理能力大大下降。

而不同的是，Node.js 的 IO 操作是异步非阻塞的：

```
file.read("file.data", function(data){
    doSomethingAfterRead(data);
});
doSomethingOnReading();
```

从代码可以看到，在文件读取期间，服务器仍在执行 doSomethingOnReading 函数，没有白等待，而在读取文件后的操作则写在读取文件的回调函数中。就是这样的异步非阻塞的 IO 操作，使得 CPU 的利用率大大提高，从而使服务器的性能得到很大的提升。

阻塞型 I/O 程序在执行过程中通常要执行很多 I/O 操作，例如读写文件、输入输出、请求响应等等。I/O 操作时最费时的，至少相对于代码来说，在传统的编程模式中，举个例子，你要读一个文件，整个线程都暂停下来，等待文件读完后继续执行。简单地来说，I/O 操作阻塞了程序的进行，大大地降低了系统使用的效率。

异步 IO 非阻塞是 Node.js 最大的一个特点，也是最受关注的一个特点。这个特点使开发者必须换一个思路去开发项目，编程的时候想的不是先做什么再做什么，更多的是可以同时做什么。

举本项目中的一个例子来说明一下。当某个用户的手牌都配对成功时，系统会告诉数据库该用户胜利的盘数递增，这个更新数据库的操作就是一个异步 IO，也就是说不需要等更新数据库完成才往下走，也就是说添加胜利消息、返回成功 JSON 给用户等操作都不需要等数据库更新的完成。而传统的后台实现，例如让我用 PHP 来开发这个功能，绝对是等数据库更新完成后才返回成功 JSON 给用户，也就是说，让用户白白等待更新数据库的时间，而这段时间里，浏览器和服务器的连接是一直存在的，意味着如果多个这样的操作，同一时间内的连接数目会增加，这样就很可能影响系统的吞吐量和并发性能了。而且在这段时间内，CPU 一直在忙等待，白白地浪费了 CPU 资源。

以后如果要拓展，需要读取数据库中多个表的数据，也可以利用 Node.js 的异步 IO 非阻塞的特性，使用 eventProxy,就可以同时去读多个表，等所有数据都读完以后再触发关于这些数据的操作，这样就充分使用 CPU 资源。

2.1.4 Node.js 的单线程特性

对于并发的请求，用多线程来实现通常会有如下的字眼或者问题，如“开销大”、“容易出错”、“设计困难复杂度高”等等。这里说的复杂性是针对于死锁的预防和避免的策略、共享变量的访问以及线程与线程之间的竞争而言的。

而 Node.js 则是从另一个角度去解决并发的问题，它是以单进程和单线程运行的，或许你会大吃一惊，但你没看错，它的一大特点就是单线程，这个设计很大胆，但是这和 JavaScript 的运行方式是一致的。单线程，使程序逻辑简单，系统资源占用低，而且没有通信与锁开销带来高性能。

或许你会担心，Node.js 是单进程的和单线程的，那么在现在这个多核时代中，单核性能出色的它是如何驾驭多核 CPU 机器的呢？Node.js 的作者 Ryan Dahl 提议，利用某些通信机制来运行多个 Node.js 进程，以协调各项任务。其实目前已经有很多第三方的 Node.js 模块就是在实现多进程支持。一个 Node.js 进程性能已经很好，若能做好负载均衡，充分利用多核 CPU 的话，多个 Node.js 进程必能达到意想不到的良好效果。

2.1.5 Node.js 的事件驱动特性

其实后端 JavaScript 技术不只有 Node.js，而它之所以能够脱衣而出，其中一个原因是，它是基于事件的这个特点。相比于 Rhino（其中一种后端 JavaScript 技术），可以看到其支持的后端 JavaScript 脱离不了其他语言同步执行的影响，以致于 JavaScript 在前端和后端的编程之间有很明显的差别，它们在编程模型上是不统一的，这样前后端用同一种语言就没有意义了。在前端的开发中，事件的使用到处可见，DOM 上都是各式各样的事件。在 Ajax 开始渐渐流行以后，异步的请求得到了广泛的认同，而这个 Ajax 技术也是基于事件机制的。在 Rhino 中，IO 操作（如文件读取等）都是同步操作进行的。在 Node.js 这种单线程的模型下，如果还是采用同步机制，则完全没有

办法和传统的服务端脚本语言（如 PHP）的成熟度媲美，性能也没有什么值得让人兴奋的部分。直到 Ryan Dahl 在 2009 年推出 Node.js 后，后端 JavaScript 才走出其迷局。Node.js 的推出，使前后端 JavaScript 的编程模型得到统一，基于事件机制就可以充分利用异步 IO 去突破单线程编程模型的性能瓶颈，让 JavaScript 在后端起到令人称赞的价值。

举个简单的例子来说明一下 Node.js 的事件驱动机制：

```
var net = require("net");
net.createServer(function(stream){
    stream.on('connect', function(){
        stream.write("Welcome!\r\n");
    });
    stream.on('data', function(data){
        stream.write(data + "\r\n");
    });
}).listen(8080);
```

简单的几行代码就写好一个 TCP 的服务了。这里就是利用了 connect 和 data 的事件，当服务器收到连接信息时则会输出 Welcome!，当接收到数据时则会输出数据信息。事件驱动时后台编程变得很自然很简单，有什么样的事件则做出怎么样的反应。

2.1.6 Express 框架

Express 框架是 Node.js 比较成熟的 web 框架，很方便开发，很多项目都是使用 Express 框架进行开发的。本项目中使用的 web 框架就是 Express 框架。它支持路由设置，重定向辅助、不同渲染模板辅助、会话管理等等。具体可以参考 http://expressjs.com/。

2.1.7 Node App Engine 提供支持

Node App Engine 是专门托管用 Node.js 开发的应用的服务平台，提供 Node.js 应用的在线部署功能。当前版本支持 Node 应用托管、在线开发、多人协作、代码上传、二级域名绑定、mongodb 空间和在线数据管理，每个用户可以支持 10 个应用，支持 git 代码管理、在线调试等，目前还推出了 shell 命令的客户端，让开发者更加便利地开发项目。它是阿里云提供的一块空间，现在由淘宝数据平台与产品部维护。

图 1-4 Node App Engine 首页

图 1-5 Node App Engine 应用信息

Node App Engine(简称 NAE)，地址是 http://cnodejs.net，让本项目可以免费托管在上面，提供了很大的方便。它还提供了代码管理，在线编辑器 IDE，MongoDB 空间，完全可以满足本项目的需要，使本项目能够在网络上让大家可以访问。

2.2 MongoDB

2.2.1 MongoDB 特点

MongoDB 是一个最像关系数据库的非关系数据库，也可以说，它是介于关系与非关系数据库之间的产品。其支持的存储结构是一种很像 json 的名叫 bson 的格式，是一种非常松散的数据结构，这样就可以存储相对复杂的数据类型。MongoDB 最显著的特点是其支持的查询语言，它非常强大，语法比较像面对对象的那种，关系数据库单表查询的绝大部分功能它几乎都可以实现，而且它还支持建立索引。

它的特点是容易使用、容易部署、高性能，存储数据十分方便。主要功能特性有：

存储的结构是一个个集合，很适合对象类型的数据存储。

模式自由。

支持动态查询。

支持索引，甚至对于内部对象。

支持查询。

支持复制和故障恢复。

对于二进制的数据存储十分高效，包括如视频等大型对象。

支持云计算层次的扩展性，自动处理碎片。

支持 RUBY，PYTHON，JAVA，C++，PHP 等多种语言。

文件存储格式是一种 JSON 的扩展，叫 BSON

可通过网络访问。

MongoDB 是面向集合的，也就是说，数据被分组，存储在数据集中，而这个数据集被称为一个集合，概念上就类似关系数据库里面的表，但不同的是，他没有固定的存储结构，不需要定义任何模式。在数据库中每个集合都有属于它自己唯一的标识名，而且可以包含无数的文档。

MongoDB 是模式自由的，也就是说，我们不需要知道存储在 MongoDB 数据库中的数据是什么结构，因为它没有什么结构定义。甚至你可以把不同结构的文件存在一个数据库中，如果有需要的话。

在 MongoDB 的集合中，存储的数据是以键-值对（key-value）的形式去存的。键是一个文档的唯一标识符，是个字符串类型，而值则可使是任意类型，包括字符串、数字或者内部对象等等各种复杂的类型。我们称这种存储形式为 BSON（Binary Serialized document Format）。

2.2.2 MongoDB 储存格式

MongoDB 所使用的储存格式是 BSON。BSON，是 Binary JSON 的缩写，是一种很像 JSON

的二进制形式的储存格式。它和 JSON 一样支持内嵌的数组对象和文档对象，而且它还有一些 JSON 没有的数据类型，例如 BinData 和 Date 类型。BSON 可以用来作为网络传输的一种储存格式，这有点像 Google 的 Protocol Buffer，但 BSON 是一种模式自由的储存格式。BSON 的有点是具有很好的灵活性,但它的空间利用率不是很高,这是它的缺点。

BSON 有三个特点：高效性、可遍历、轻量。

{“hello":"world"}这是一个 BSON 的例子，其中"hello"是 key name，它一般是 cstring 类型，字节表示是 cstring::= (byte*) "/x00" ,其中*表示零个或多个 byte 字节，/x00 表示结束符;后面的"world"是 value 值，它的类型一般是 string、double、array、binarydata 等类型。

MongoDB 使用 BSON 这种格式，去实现数据的存数和网络数据的交换，把存储数据转化成一个个文档，这里所说的一个文档可以理解为关系数据库中的一条记录，但这个文档是模式自由的，没有固定结构，所以它的变化更丰富一些，例如文档可以嵌套。BSON 的可遍历性也是 MongoDB 选择它作为存储结构的一个重要的原因。

前台用 Javascript，后台用 Node.js，数据库用 MongoDB，真是个绝配！因为编码不但都是使用 Javascript，而且数据格式全部是 JSON 系，开发代码减少，逻辑也一致。Node.js 的社区极度火热高效，凡是 Web 开发需要的组件基本上已经全有了。目前，Node.js 关于 MongoDB 的组件，比较出名的有 node-mongodb-native、mongoskin 等等，性能相当稳定。

2.2.3 MongoDB 对数据库设计的影响

MongoDB 是模式自由的，也就是说，我们不需要知道存储在 MongoDB 数据库中的数据是什么结构，因为它没有什么结构定义。甚至你可以把不同结构的文件存在一个数据库中，如果有需要的话。这一点在本项目中没那么明显，但在我开发的另一个项目中体会很深。MongoDB 对于数据库设计的影响，于我而言，就是考虑你要存储什么，而不是像用 mysql 开发那样考虑要怎么存储。这一方面是很便利的，数据库设计起来也不显得那么吃力，想存什么就存什么，就用我们理解的最自然的方式去存就好了，不需要过多地考虑实体与实体间的关系究竟是一对一，多对一，还是麻烦的多对多。

MongoDB 是面向文档的，意味着在这种数据库中主要存储单位是集合 Collection。这里的集合，个人理解，就是一张表，把相关的东西都存进去，重点是选好类似关系数据库中的主键的 key，并建好索引。

2.2.4 MongoDB 与 Node.js 的组合

前台用 Javascript，后台用 Node.js，数据库用 MongoDB，真是个绝配！因为编码不但都是使用 Javascript，而且数据格式全部是 JSON 系，开发代码减少，逻辑也一致。Node.js 的社区极度火热高效，凡是 Web 开发需要的组件基本上已经全有了。目前，

Node.js 关于 MongoDB 的组件，比较出名的有 node-mongodb-native、mongoskin 等等，性能相当稳定。

2.2.5 MongoSkin 的便利

MongoSkin 是让 Node.js 支持 MongoDB 的内嵌访问层，提供了对 MongoDB 的数据插入、删除、更新、查找、建立索引等接口，具体文档可参考这个网址：https://github.com/guileen/node-mongoskin。

2.3 本章总结

本章介绍了一下几点：

- Node.js 的性能、特点以及优势。
- MongoDB 的特点和优势。

第三章　项目的需求分析

本章将详细描述本项目的需求，包括游戏的简介、规则以及主要页面的需求和逻辑。

3.1 游戏总体介绍

3.1.1 游戏简介

实现一个基于 MongoDB 和 Node.js 的社区在线游戏，开发出来的游戏放在 CNode 社区提供的免费的 node app engine 上运行。

3.1.2 游戏规则

本题目特意选了一个类似名叫 Bonbons 的桌游，关于记忆的游戏。游戏规则如下：把背面朝上的牌放在桌面中间，排成 M*M 的正方形，每名玩家（共四名玩家）四张手牌。所有牌的正面都画有图案，总 9 种图案。玩家们轮流翻开自己的牌，并尝试将桌面中间的牌匹配。如果能匹配成功，将两张牌都面朝上放置。第一个把手上的四张牌都匹配成功的玩家获胜。

3.2 游戏流程介绍

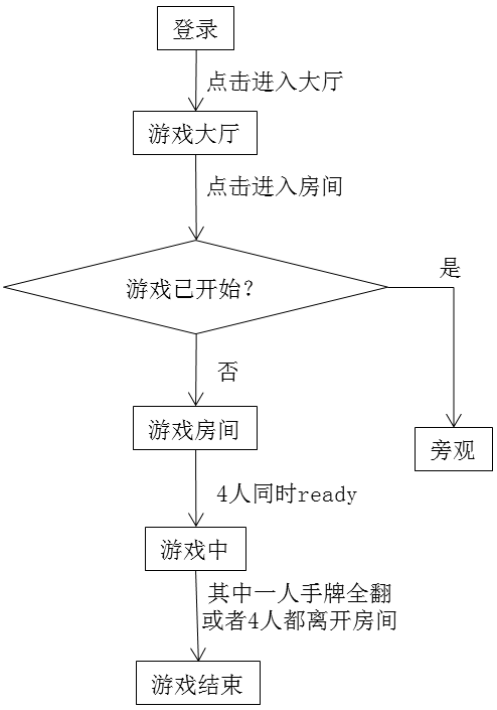

图 3-1 游戏流程介绍

3.3 游戏需求细化

3.3.1 游戏页面

由图 3-1 可知，游戏至少需要三个页面，包括首页、游戏大厅、游戏房间。当然还需要登录页面（决定使用新浪微博接入，登录页面由新浪提供）和错误显示等页面，但并不是本文讨论的重点，所以，下面会分别描述这三个主要页面的需求。

3.3.2 游戏首页

游戏主页是打开这个游戏的第一个页面，所以主要是展示这个游戏以及引导用户。所以，游戏主页主要的需求包括三个，一个需求是游戏介绍，这个可以用图片配上文字显示游戏的操作和规则；一个需求是引导用户用新浪微博登录，就是有个登录按钮，链接到新浪微博的登录页面；最后一个需求是引导用户进入游戏大厅。

3.3.3 游戏大厅

游戏大厅，顾名思义就是能看到游戏的房间，而可以进入游戏房间。所以，用户从首页进入游戏大厅后，游戏大厅的需求主要有两个，一个是实时地显示房间的状态，包括房间的号码（名字）、游戏状态（等待中、游戏中）等；另一个是引导用户进入游戏房间。

3.3.4 游戏房间

游戏开始前：

实时显示进入房间的用户的情况，包括用户的进入、离开、准备、取消准备，以及显示用户的信息。当前用户要显示准备/取消准备的按钮，让其可以改变自己是否准备的状态。

游戏开始后：

房间四人同时准备好时，显示游戏开始。游戏的状态和用户的操作都是实时显示的。游戏四人每个人都会轮着有自己的回合，每个回合最多 30 秒（30 秒内完成操作的，完成操作后即回合结束；若 30 秒内未完成操作的，都会把操作复原，回合结束），每个用户的回合结束后就会按着座位顺序轮到下一个用户的回合。轮到自己回合时，用户需要先按中央牌池的某一张牌，这时会显示这张牌的图案；再让用户按自己的手牌（自己都会坐在右下角的位置上），同样会显示这张手牌的图案；如果显示的这两张牌图案不一样的话，这两张牌的图案会重新盖上。当有用户四张手牌都配对成功时，该用户就胜出，加上 10 分，所有用户重新进入游戏房间。游戏过程中有用户离开时，该用户的位置上显示“该用户已掉线”。

3.3.5 游戏需求汇总

由于上述需求比较零散，所以本人作了个简单的汇总，见表 3-1。

表 3-1 游戏需求汇总

主要页面	需求
游戏主页	游戏介绍、引导登录、引导进入游戏大厅
游戏大厅	实时显示房间状态、引导进入游戏房间
游戏房间	游戏开始前： 实时显示用户状态、显示用户信息 游戏开始后： 实时显示游戏状态、实时显示用户操作、显示用户离开

3.4 本章总结

本章介绍了一下几点：

- 本项目是个基于 mongoDB 和 node.js 的社区在线游戏。
- 本项目的游戏规则。
- 本项目的需求细化说明。

第四章　项目的设计与实现

本章主要陈述整个项目的实现方法，将分别对各个功能模块的实现进行详细的描述，对某些关键的技术（如消息同步等）会作一些方案的讨论。

4.1 项目目录组织

程序目录组织见图 4-1：

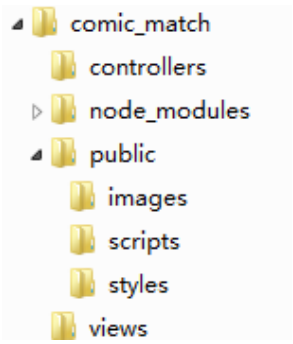

图 4-1 本项目的目录组织

本项目的目录组织体现的是 MVC 的模式。controllers 目录里放的是后台控制的 JavaScript 代码，view 目录里面放的是前端的 HTML 代码，而 public 目录下则放着前端会用到的图片、JavaScript 代码和 CSS，node_modules 内放的是 Node.js 的模块。

4.2 项目功能模块

4.2.1 项目功能模块介绍

图 4-2 为项目的大致功能模块图：

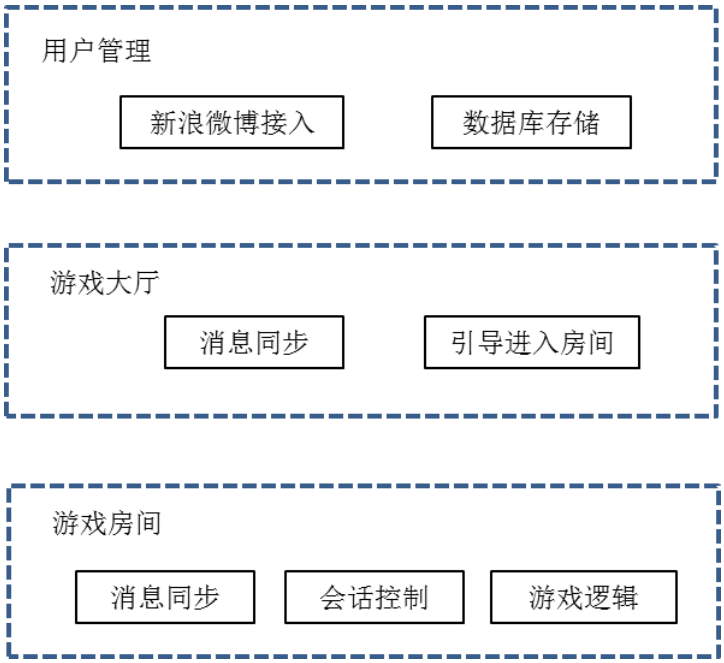

图 4-2 功能模块图

图 4-2 中，虚线框代表功能模块，实线框代表模块里面的组件。

4.2.2 项目模块划分

由上面的项目架构图可以看到，本项目分成用户管理、游戏大厅、游戏房间三大功能模块。用户管理模块主要是用户的登录、退出、检查是否登录，以及用户信息的存储（即数据库部分）。游戏大厅模块顾名思义就是指用户进入游戏大厅的相关功能，衔接这主页和游戏房间两个页面，包括可以实时看到各个房间的状态（等待中还是游戏中），以及引导进入游戏房间。游戏房间模块是整个项目最重要也是最核心的模块，所有游戏规则、逻辑和流程等等都在这个模块中控制，包括消息同步、会话控制和具体游戏流程三个小模块。

4.3 用户管理模块设计

4.3.1 模块的总体设计

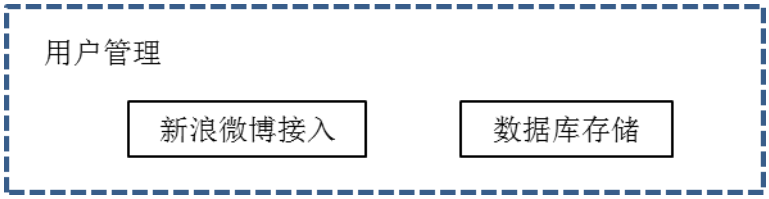

图 4-3 用户管理模块的设计

用户管理模块是本项目最基础的一个模块，几乎每一个页面，每一个步骤，都会调

用这个模块的接口。这个模块包括用户登录退出和用户信息存储两个小模块。

4.3.2 新浪微博的接入

这里的新浪微博接入指的是微连接，也就是用户的登录退出。微连接(Weibo Connect)是新浪微博针对第三方网站提供的社会化网络接入方案。 接入微连接可以让您的网站支持用新浪微博账号登录，让用户方便的分享网站的内容、在网站上关注您的官方微博。使用微连接，可以快速为网站增加用户、流量和官方微博粉丝。

其实本来想直接做个简单的用户登录退出，也就是用数据库存储用户名、密码，然后让用户输入用户名密码，检查其密码是否正确。但相比于微博接入，这是个相当麻烦的选择，因为直接存储用户名密码就要考虑到安全问题，前后端都必需检测用户名和密码是否合法，防止数据库注入，担心泄露个人信息等等的问题。而新浪微博的接入，则可以减少很多这样的麻烦。因为用户的登录信息都由新浪微博统一管理，这样安全风险则可以转移给新浪微博。而且新浪微博的接入，可以使社区游戏的互动增加，在玩游戏的时候知道对方的微博，方便交友。再者，近两年微博的传播效果是有目共睹的，以后宣传可以利用微博这个新兴的平台，对于游戏的推广可以说是成本最低和比较有效的。

新浪微博的接入，本人用的是淘宝苏千@Python 发烧友 开发的 node-weibo，是微博 Node.js 的 SDK。它能方便的完成微博接入，只需要以下的代码：

```
var weibo = require('./node_modules/weibo');
app.use(weibo.oauth({
    login_path: '/login', //指定登录路径
    logout_path: '/logout', //指定登出路径
    blogtype_field: 'type', //微博类型的变量名(除新浪外，node-weibo 还支持其他类型的微博，如腾讯微博、twitter 等)
    callback_path: '/' //登录登出后回调的路径
}));
```

可见，node-weibo 的使用非常方便快捷，而且效果很好。用微博登陆后，还能在 req.session.oauthUser 中得到用户微博相关的信息，十分好用。

4.3.3 用户信息的数据库设计

由于有微博的接入，用户信息的数据库就不需要存储太多的东西了，目前只存储了用户玩游戏的总盘数、赢的总盘数、中途逃跑或掉线的总盘数、分数。

这里使用的数据库是 MongoDB。之所以使用 MongoDB，是因为 MongoDB 是最像关系数据库的非关系数据库。在本文第二章中已经谈过 MongoDB 的好处，这里就不再重复了。

本项目使用的是 MongoSkin 的库，是 Node.js 比较常用的关于 MongoDB 的库，值得强调的是，这个库是中国人开发的。使用 MongoSkin 也非常简单，代码如下：

```
var mongo = require('mongoskin');
var db = mongo.db('[数据库帐号]:[ 数据库密码]@127.0.0.1:20088/[数据库名称]');
```

若要操作一个名为 user 的集合（类似关系数据库中的表），则这样操作：

```
var user = db.collection('user');
//根据 id 查找信息
user.find({id: id}).toArray(function(err, data){
if(err){
        //错误处理
    }
    //操作 data
});
//保存 data 信息
user.save(data, function(err){
    if(err){
        //错误处理
    }
});
//根据 id 更新信息
db.update({id: id}, {$set: newData}, function(err){
    if(err){
        //错误处理
    }
});
```

本项目把数据库的连接信息统一放在 config.js 内，而数据库的相关操作则独立封装出来，放在一个名为 db.js 的文件内，并开放具体操作的接口。目前 db.js 的接口包括：游戏总盘数加一（increasePlay）、赢的总盘数加一（increaseWin），逃跑的总盘数加一（increasePart）、加分（addScore）和读取用户信息（getUserInfo）。

使用 MongoDB 存储信息，更多的是要考虑存什么，而不用像关系数据库那样，要过多地考虑存储的那些数据之间的关系，一对多还是多对多等等，才能进行数据库的设计。所以，用 MongoDB 存储信息是个不错的选择，这也是非关系数据库的优点所在。而且使用 MongoDB 也利于拓展，它没有固定的存储字段或者是模板结构，它是以 BSON 的格式去存储的，提供了很大的灵活性，以后拓展存储信息，直接在数据库上增加数据即可。

4.4 游戏大厅模块设计

4.4.1 模块的总体设计

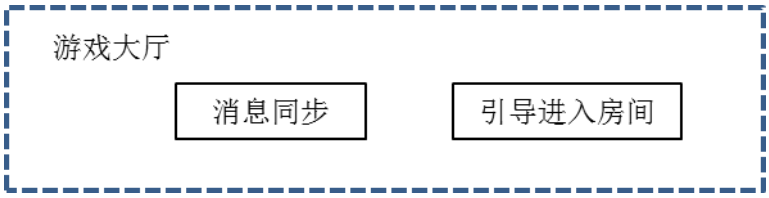

图 4-4 游戏大厅模块设计

游戏大厅模块是一个相对简单的模块，主要起着衔接的作用，也就是把用户从介绍首页引导到游戏房间的过渡模块。它包括消息同步和引导进入房间两个部分。其中难点在消息同步的部分，主要是同步房间的状态，让用户知道该房间是在“等待中”还是“游戏中”。

4.4.2 消息同步的设计

所谓的消息同步，是指服务器和浏览器之间的消息通信方式。而消息同步通常有两种方式，一种是浏览器不断询问服务器要新的消息（称为浏览器“拉”方式，动作发起者是浏览器），另一种是服务器主动把新的消息发送到浏览器（称为服务器“推”方式，动作发起者是服务器）。在《Ajax 推和拉技术的比较》论文中有提到：如果我们想要高的数据一致性和高的网络性能，我们应该选择推送的方法。然而，推送方案带来了一定的可扩展性问题：服务器应用程序 CPU 使用率比拉方案高出 7 倍。对于拉的方案，取得具有很高的网络性能的整体数据一致性是非常困难的。如果拉的时间间隔大于发布时间间隔，一些数据丢失会发生。如果是低于，网络性能将受到影响。拉性能表现好只在当拉的时间间隔等于发布的时间间隔的时候。然而，为了实现这一目标，我们需要事先知道确切的发布时间间隔。但发布时间间隔很少是不变的和可预测的。

推和拉的方案各有其优点和缺点，而反观本项目的游戏大厅消息同步的需求，既要整体数据一致性、实时性，也要高性能，不能使 CPU 负载过重，因为以后用户量有可能会大幅度扩展。于是，本项目中游戏大厅的消息同步采用的是长轮询（long polling）的方案。长轮询（又称异步轮询）是一个纯服务器推送和客户端拉的混合物。订阅通道后，客户端和服务器之间的连接在指定的时间期限保持开放。如果服务器端没有事件发生，超时会产生，此时服务器异步地要求客户端重新连接。如果有事件发生时，服务器将数据发送到客户端并与客户端重新连接。而这个指定的时间期限很小时，长轮询系统表现得更像纯拉的方案，因为客户端（浏览器）会频繁请求服务器，当时间期限大时应更像纯推的方案，服务器更倾向于有新消息则发送到客户端（浏览器）。

本项目中游戏大厅的消息同步参考了 node chat 的代码，node chat 是一个用 Node.js 写的实时多人在线聊天室。下面会分别介绍后台和前端的实现。

后台方面，游戏大厅的消息同步主要写在 hallMessage.js 文件中，而在游戏后台主体 game.js 文件中调用。hallMessage.js 作为一个消息同步的闭包，就像是一个小组件一样。其中里面有两个关键的数组类型变量：messages 和 callbacks。messages 是用来存储消息，算是一个消息队列，里面的消息都是按照时间先后顺序插入到队尾的。消息结构如下：

```
var m = {
    roomId: roomId,
    type: type, //start, over, update
    timestamp: (new Date()).getTime(),
    info: info
};
```

而 callbacks 则是记录用户的回调函数。hallMessage.js 提供两个接口，appendMessage 函数用来添加消息到 messages 的消息队列中，query 函数是查找指定时间以后的消息。appendMessage 函数的步骤如下：先生成一条消息（把参数赋值到消息属性中，再加上系统时间）；然后检查 type 是否符合指定类型，若不符合则返回，否则插入到 messages 消息队列队尾；接着遍历 callbacks 中用户的回调函数，把新消息作为参数调用回调函数，并移除该回调函数；最后若消息队列超过指定长度则移除旧的消息直至符合指定长度。query 函数有两个参数 since（记录用户最后收到的消息的系统时间）和 callback（记录用户的回调函数），其步骤如下：遍历一次消息队列 messages，把时间后于 since 的消息存到 matching 队列中；如果得到的 matching 队列长度不为零，则把 matching 作为参数调用回调函数，否则把回调函数和系统时间一起插入到 callbacks 队列尾部。除了两个接口以外，还会每 3 秒清理一次 callbacks 中超时的回调函数，这里的超时时间为 30 秒。

前台方面，游戏大厅的消息同步主要写在 hall.js 文件中的 longPoll 函数中。该函数有一个 data 参数，是指接收到的数据。这里同样介绍一下 longPoll 函数的步骤：首先检查 transmission_errors 是否超过指定次数，是则返回，否则继续往下走；接下来，如果 data 和 data.message 有定义，则记录下最晚那条消息的系统时间 last_message_time，处理 data 信息（在页面显示 data 中指的房间的状态变化）；最后根据 last_message_time 向服务器发出 Ajax 请求索要新的消息，成功则重新调用 longPoll 函数（也就是把 data 信息的处理延后到后一个 longPoll 函数中），出现错误则记录到 transmission_errors 中。

在大厅页面 document 的 ready 事件中调用 longPoll 函数，也就是长轮询的第一次请求由客户端（浏览器）发起。

本项目中游戏大厅的消息同步可用图表表示，见图 4-5：

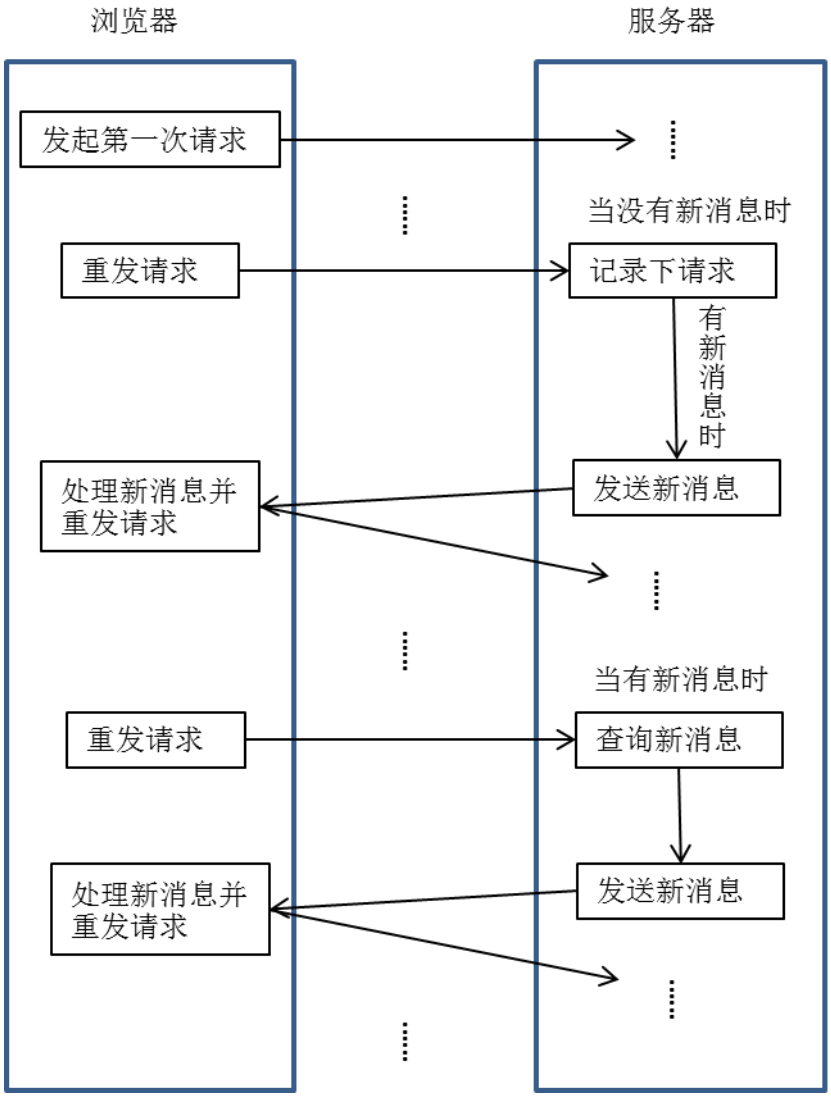

图 4-5 游戏大厅消息同步图解

4.4.3 进入房间的设计

由于每局游戏只能最多只能有 4 个人参加，且游戏允许旁观（即进入游戏房间但不参与游戏），为了更显示人性化，在进入房间之前，本人设计浏览器先询问游戏房间的状态，如果是“等待中”则直接引导进入该游戏房间；如果是“游戏中”则提示用户该房间正在游戏中，是否去旁观，若用户点击确定则引导进入该游戏房间，点击取消则取消进入房间的操作。

进入房间的操作是相当简单的，只是直接把 location.href 改成“/room/:roomId”。

4.5 游戏房间模块设计

4.5.1 模块的总体设计

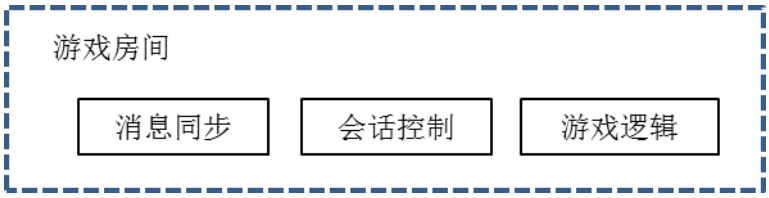

图 4-6 游戏房间模块的设计

游戏房间模块是整个游戏最重要的部分，是本项目的灵魂所在。它包含了整个游戏过程的逻辑与实现。该模块同样包含消息同步的部分，还包含会话控制部分以及游戏逻辑部分（各种消息或事件的处理和响应）。

4.5.2 消息同步的设计

游戏房间的消息同步写在 message.js 文件中，部分同样是参考 node chat 的源码，也是采用长轮询的方式。消息的结构如下：

```
var m = {
    site: site,
    type: type,
    //turn,action,part,join,gameStart,win,ready,cancelReady
    timestamp: (new Date()).getTime(),
    info: info
};
```

一开始是和游戏大厅的消息同步部分（可参考本文中的 2.3.2 小节）很相似的，除了 messages 和 callbacks 变成了下标对应房间的信息数组的集合对象，也就是说，messages[x]和 callbacks[x] 对于房间 x 的关系跟 messages 和 callbacks 对于游戏大厅的关系是一样的。游戏房间与游戏大厅的需求毕竟不一样，在游戏房间中会有系统自动多加消息的需求（例如，用户 0 的回合超时结束后，系统必须添加“轮到用户 1 的回合”的消息），这样会出现个问题，有时候部分用户只能收到前面的消息（即“用户 0 的回合超时结束”的消息），也收不到后面的消息（即“轮到用户 1 的回合”），达不到消息同步的效果。调试的时候发现系统后面自动多加的消息和前面的消息的系统时间是一样的，也就是说收到前面的消息后，浏览器的 last_message_time 就是当前的系统时间，而再用这个时间去查询新消息的时候，服务器就会判断后面多加的那些消息不是新消息，因此而导致上述的问题。更深层次的原因是，因为没新的消息时服务器会记录用户的请求，在 appendMessage 函数中当有一条新消息插入时，会把这条消息发送给

callbacks 中记录的请求，而不是把后面多加的消息一并发送给用户。

对于这个问题，有两种解决方案，一种是每次系统自动多加消息的时候都延后一毫秒（javascript 中时间以毫秒为单位），这样请求新消息就能请求到。在没发现上述问题的深层次原因时，本人就是采用这种解决方案的，实践证明这是可行的。但这种解决方案使代码变得不优雅，而且定时器使用太多会影响性能，使服务器开销增大。所以在发现造成上述问题更深层次的原因后，本人毅然修改代码，选择了另一种解决方案。另一种方案是，在消息同步的文件（即 message.js）中添加一个添加多条消息的接口 appendMessages，把多条消息一并发送给用户；而在游戏过程中需要系统自动多加消息时，不直接调用单条消息添加的接口（即 appendMessage 函数），而是把消息插入到一个 pushMessages 变量中，在最后把整 pushMessages 变量作为参数去调用多条消息添加的接口（即 appendMessages 函数）。

值得一提的是，这里长轮询的时间间隔是 20 秒。

4.5.3 会话部分的设计

会话部分写在一个名为 session.js 的文件中。每当一个用户进入房间时就会创建一个属于该用户的会话（session），用来保存该用户的信息（包括房间号码、用户 ID、用户名称、座位号等），并检测用户是否掉线。

下面介绍一下会话部分的实现：

会话部分的关键变量只有一个，就是 sessions 变量，用来存储所有用户的会话信息，它是以一个房间为单位去存储会话信息的集合对象，简单地来说，sessions[n]代表的是房间编号为 n 的房间的会话信息。而在同一个房间里面的会话信息是按照座位号来存储的，也就是说，sessions[n][2]代表的是房间 n 内座位号为 2 的用户的会话信息。

会话信息的数据结构如下：

```
var session = {
    id: id,
    site: site,
    name: userInfo.screen_name,
    url: userInfo.t_url,
    location: userInfo.location,
    image: userInfo.profile_image_url,
    timestamp: new Date(),
    poke: function(){
      session.timestamp = new Date();
    },
    destroy: function(){
      delete sessions[roomId][session.site];
      messageMgr.appendMessage(roomId, session.site, "part", null);
    }
};
```

它包含了用户的 ID，座位号，用户的名称（即微博昵称），以及最后一次更新的系统时间；包含了两个函数，一个是刷新最后一次更新的系统时间 poke 函数，一个是销毁会话的 destroy 函数。游戏后台可以通过这两个接口来操作该用户的会话信息。

整个会话部分只有两个接口，一个是创建会话 createSession 函数，顾名思义功能就是要创建一个会话，在一系列检查（检查该房间的该作为是否已经有人；检查此人是否已经进入一个房间，为了测试方便目前已屏蔽了此检查）后创建会话；另一个是去会话信息 getSession 函数，是通过房间号码和座位号取出该用户的会话信息。

另外，系统后台还会每隔 3 秒检查一次所有的会话信息，清除超过指定时间期限的会话信息，这里的时间期限是 30 秒，也就是说，检查某个会话时，若它的最后更新时间离现在超出 30 秒的话，就把它清理掉。这个时间期限是有讲究的，不能随便设，设得太小时，当有些用户网速比较慢就会很容易被判断为掉线而清理掉，其实用户并没有掉线，只是网速比较慢而已；设得太大时，若有用户在某一刻掉线了，就要等很长时间才能发现该用户掉线了，这样子游戏的实时性就不准确了。而且这个时间期限必须比消息同步的长轮询时间间隔要长，因为当没有新消息的时候，整个长轮询的时间间隔里，浏览器并没有跟后台通讯，也就没有刷新会话的更新时间，若会话清理时间期限比长轮询的时间间隔要小的话，则服务器会做出“该用户已掉线”的错误判断，即使该用户并未掉线，而且网速也不一定慢。因此，本项目中游戏房间消息同步的长轮询时间间隔为 20 秒，而会话清理时间期限为 30 秒。

4.5.4 游戏逻辑部分的设计

关于游戏的逻辑部分，也就是消息响应的部分（也就是房间等待时和游戏开始后收到什么消息后会有怎么样的反应，有什么样的状态变化等等），比较繁琐，用文字说明的话很难表达清楚，所以下面用两个流程图分别描述前端和后台的游戏逻辑，希望能把整个游戏逻辑表达清楚。详细见图 4-7：

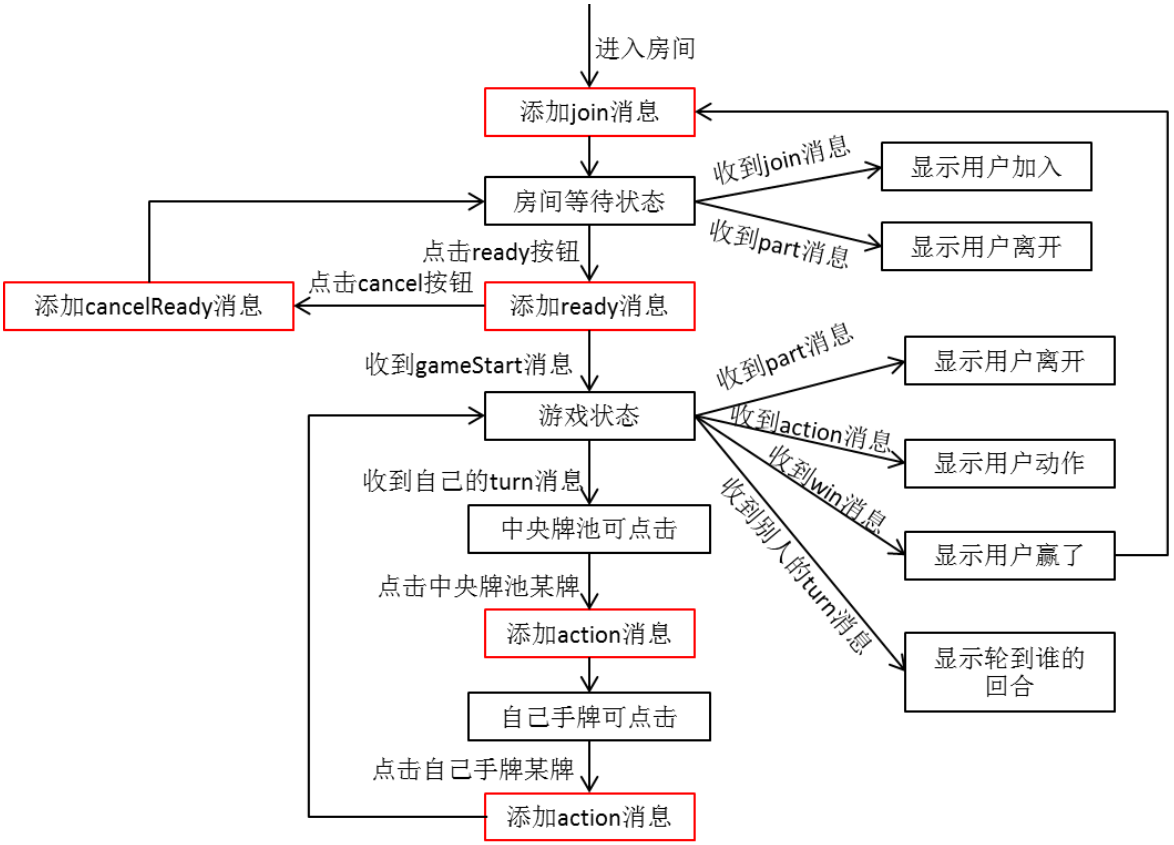

图 4-7 游戏前端的游戏逻辑

图 4-7 中的黑色框代表状态的变化，红色框代表同服务器的通讯。服务器的游戏逻辑比较分散，这里只举了用户加入、准备和动作的例子来表述游戏逻辑，见图 4-8：

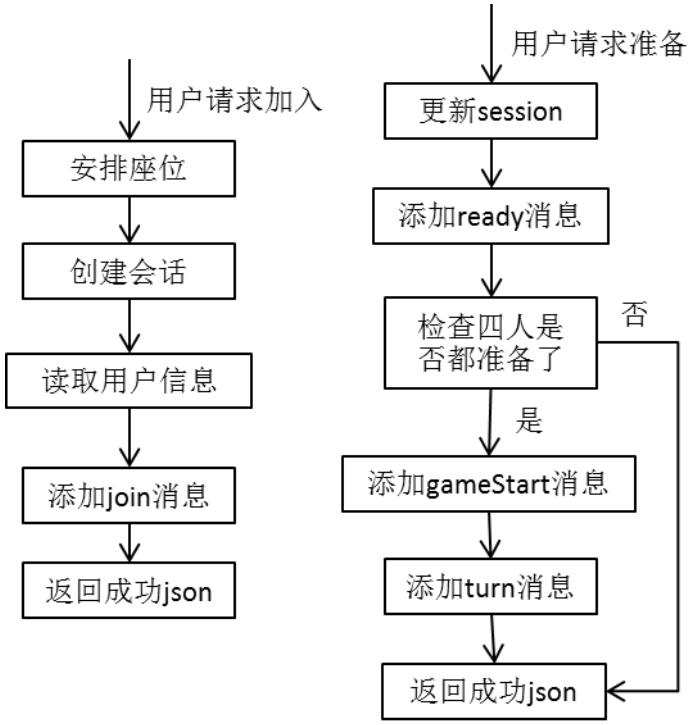

图 4-8 服务器用户加入和准备的游戏逻辑

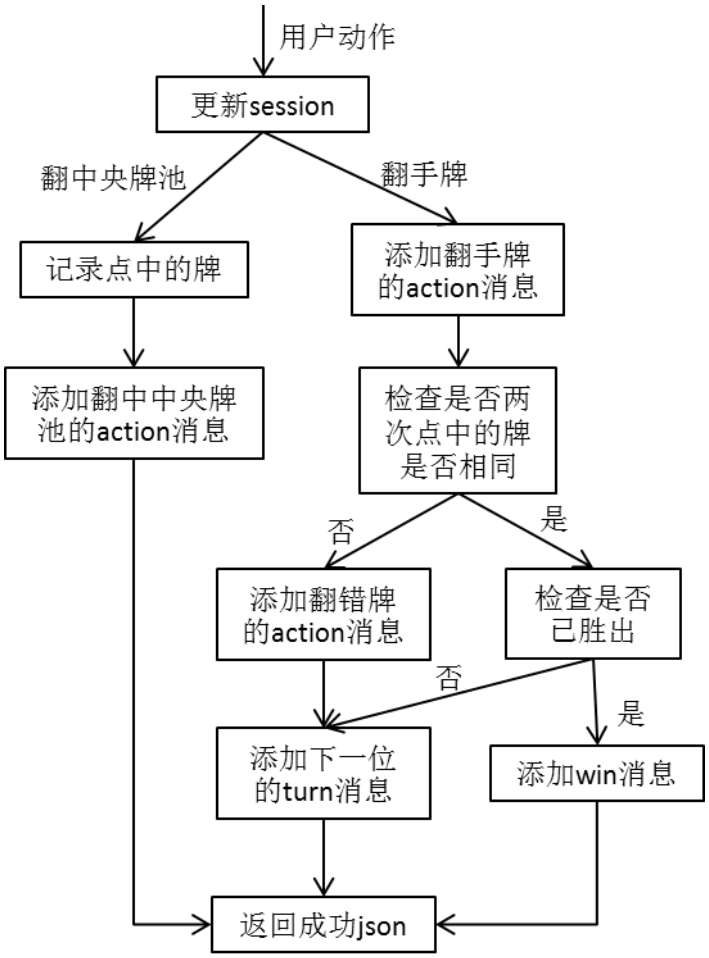

图 2-9 服务器用户动作的游戏逻辑

4.6 本章总结

本章介绍了一下几点：

- 介绍了项目总体的架构设计，分成用户管理、游戏大厅、游戏房间三大功能模块。
- 用户管理模块是本项目最基础的一个模块，几乎每一个页面，每一个步骤，都会调用这个模块的接口。这个模块包括用户登录退出和用户信息存储两个小模块。
- 游戏大厅模块是一个相对简单的模块，主要起着衔接的作用，也就是把用户从介绍首页引导到游戏房间的过渡模块。它包括消息同步和引导进入房间两个部分。
- 游戏房间模块是整个游戏最重要的部分，是本项目的灵魂所在。它包含了整个游戏过程的逻辑与实现。该模块同样包含消息同步的部分，还包含会话控制部分以及游戏逻辑部分（各种消息或事件的处理和响应）。

第五章　系统运行及改进探讨

本章将展示整个游戏系统的进行效果，并探讨其改进方向。

5.1 成果展示

5.1.1 游戏完成情况概述

目前本项目整体已经完成，包括前台和后台部分。当然，美工方面还是稍微欠缺的，毕竟本人也不是搞美工设计的，审美水平有待提高。偶尔还是会出现一些小 bug，但由于很难重现，以及时间有限，还没有一一修复。

本项目的地址是 http://sandy.cnodejs.net/。

5.1.2 游戏运行截图

图 5-1 游戏首页

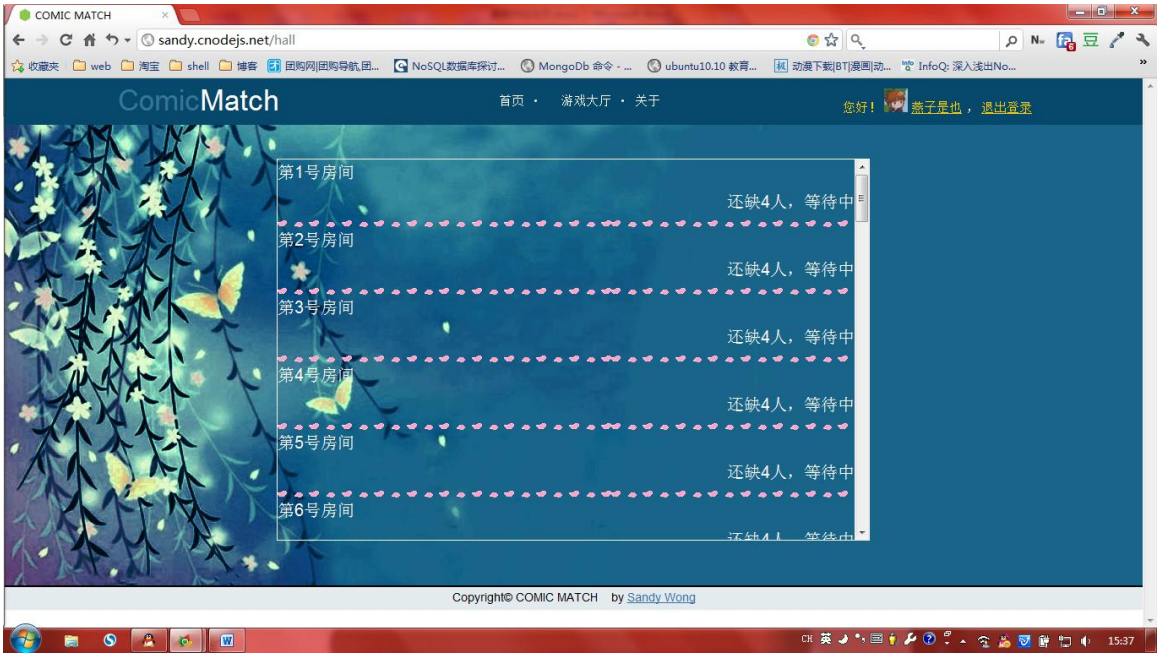

图 5-2 游戏大厅

图 5-3 游戏进行中

5.2 项目改进的方向

5.2.1 系统架构设计的改进

目前，本项目整个游戏的后台游戏逻辑都写在 game.js 里面，大概有 400 多行代码，而且代码比较繁杂，相互依赖比较多。以后改进的话，要把游戏大厅和房间关于游戏逻辑的后台代码分离出来，再进一步解耦合，进一步细化。

5.2.2 消息同步实现的改进

目前，游戏大厅和游戏房间的消息同步都是使用了长轮询的方式。但其实长轮询的方式并没有特别显示出 Node.js 的优势和高性能。以后改进的话，会改用 socket.io 来实现消息同步。用 socket.io 的话会使通讯更加自然，而且 Node.js 异步 IO 和基于事件的特性也将更加明显的表现出来，系统的性能也会进一步提高。

5.2.3 游戏的扩展

目前的游戏逻辑相对还是比较简单的，所以游戏的趣味性可进一步提高。以后拓展游戏时，可以加入道具、在线聊天等功能，增加用户间的互动，从而增加游戏的趣味性。

5.3 本章总结

本章介绍了一下几点：

- 项目完成情况。
- 项目改进的方向。

结束语

这个项目前前后后做了两个多月，从定需求、模块设计到代码实现、优化。这是我第一次开发一个网页游戏，也是第一次用 Node.js 做相对比较大，互动比较强的项目。游戏后台的那些逻辑一开始是比较混乱的，各种操作耦合性很高，很难剥离，后来不断改进，才显得比较清晰。过程中学到很多东西，很有收获。

之所以选择 Node.js，除了对新技术的好奇以外，还以此项目怀念我大三暑假时在淘宝数据平台与产品部（EDP）实习的美好时光，献给那些和我一起打闹和奋斗的同事们。

之所以选择做游戏，是因为我大四在凡趣科技公司实习时做的就是游戏后台，觉得游戏这种 IO 密集型的程序完全可以用 Node.js 实现，并表现出高性能。

我为自己这两个多月来的努力以及成果感到骄傲。

参考文献

〔1〕 David Herron. Node Web 开发〔M〕.北京:人民邮电出版社，2012.4.

〔2〕 Bozdag et al. A Comparison of Push and Pull Techniques for AJAX 〔J〕. Web Site Evolution，2007，15-22

〔3〕 Manuel Kiessling. Node 入门〔DB/OL〕.http://www.nodebeginner.org/index-zh-cn.html

〔4〕 崔康. 什么是 Node.js〔DB/OL〕.http://www.infoq.com/cn/articles/what-is-nodejs

〔5〕 田永强. Node.js 的事件机制〔DB/OL〕. http://www.infoq.com/cn/articles/tyq-nodejs-event

〔6〕 田永强. 初探 Node.js 的异步 I/O 实现〔DB/OL〕. http://www.infoq.com/cn/articles/nodejs-asynchronous-io

〔7〕 Joyent, Inc. Node.js v0.6.18 Manual & Documentation〔DB/OL〕. http://www.nodejs.org/api

〔8〕 廖恺. nodejs 快速入门〔DB/OL〕. http://cnodejs.org/topic/4f16442ccae1f4aa27001151

〔9〕 廖恺. node.js 调研与服务性能测试〔DB/OL〕. http://cnodejs.org/topic/4f16442ccae1f4aa27001153

〔10〕 张轩丞. node chat 源码解读〔DB/OL〕. http://cnodejs.org/topic/4f16442ccae1f4aa2700114b

致 谢

在这里，向在项目开发过程中曾经给我提供帮助的袁锋（淘宝花名：苏千）、黄嘉仪、敦婧瑜和何翊宇以及彭绍武老师表示衷心的感谢。