

论文题目\_\_\_\_\_基于 nodejs 的微博系统的设计与实现\_\_\_\_\_

专业学位类别\_\_\_\_\_工 程 硕 士\_\_\_\_\_

学 号\_\_\_\_\_201192250201\_\_\_\_\_

作 者 姓 名\_\_\_\_\_王 越\_\_\_\_\_

指 导 教 师\_\_\_\_\_白金平 高工\_\_\_\_\_



分类号\_\_\_\_\_密级\_\_\_\_\_

UDC <sup>注1</sup>\_\_\_\_\_

# 学 位 论 文

基于 nodejs 的微博系统的设计与实现

(题名和副题名)

王 越

(作者姓名)

指导教师

白金平

高 工

电子科技大学

成 都

于燕飞

高 工

哈尔滨亚科尔科技有限公司

哈尔滨

(姓名、职称、单位名称)

申请学位级别 硕士 专业学位类别 工程硕士

工程领域名称 软 件 工 程

提交论文日期 2014.03.01 论文答辩日期 2014.05.17

学位授予单位和日期 电子科技大学 2014 年 6 月 25 日

答辩委员会主席 \_\_\_\_\_

评阅人 \_\_\_\_\_

注 1：注明《国际十进分类法 UDC》的类号。



# **DESIGN AND IMPLEMENTATION OF MICRO BLOG SYSTEM BASED ON NODE JS**

A Master Thesis Submitted to  
University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: **Wang Yue**

Advisor: **Bai Jinping**

School : **School of Aeronautics and Astronautics**



## 独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：\_\_\_\_\_ 日期：\_\_\_\_\_ 年 \_\_\_\_\_ 月 \_\_\_\_\_ 日

## 论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：\_\_\_\_\_ 导师签名：\_\_\_\_\_

日期：\_\_\_\_\_ 年 \_\_\_\_\_ 月 \_\_\_\_\_ 日





## 摘要

微博系统作为 WEB 2.0 的最新代表，它是一个基于用户关系的系统，用户能够使用 WEB 和 WAP 等各种客户端，在该系统中组建个人社区，以实现分享信息、发布信息和获取信息的目标。微博作为一种社交网络平台，将媒体、互联网与移动通讯技术聚合为一，凭借内容简短、传播快速、实时性强、互动性强的特性，满足了人们充分分享信息和交流信息的需求。在微博这个社交网络中，你不但可以作为观众，浏览你感兴趣的信息，而且也能够作为发布者，发布信息供别人浏览，也就是说，在微博中，人人都可以发言，人人都能够收听。从本质上说，微博仍是一种传播媒体，其最终目的还是向外界传递消息，获得最大的传播效果。

本文结合实际应用，为满足中小型微博系统的高性能要求，设计并实现一个基于 node js 的微博系统。基于 node js 的微博系统将用户分为普通用户和管理员用户两类，系统的有效用户在通过了身份认证进入系统后，按照用户的类型，拥有对关注、评论、转发、插入图片、插入话题、私信、编辑、视频和音乐、搜索、分页、收藏、推荐、备份等某些功能的操作权限。

为实现数据访问与业务逻辑的分离，使得页面更具动态性，基于 node js 的微博系统的体系结构采用 B/S 风格，包含数据层、业务逻辑层和表示层等三层。系统的后台实现采用 node js 框架，数据库采用 MongoDB 和 Redis 来实现，其前台的逻辑和页面效果主要采用 PHP+HTML+CSS+JavaScript 来实现。

本文的主要工作包括：简要介绍课题背景、微博系统及其发展动态；从微博系统的基本架构出发，介绍本系统涉及的 node js 架构、MongoDB 数据库和 Redis 数据库等相关技术；按照软件工程的思想对于基于 node js 的微博系统进行需求分析、设计、实现和测试。

基于 node js 的微博系统能够实现用户管理、微博管理、用户关系管理、评论和私信、搜索用户、视频分享、图片管理、短链接、敏感词处理、热点话题生成、备份等功能，具有轻量型、易布署的特点，适合于中小规模用户量的微博应用。

**关键词：**社交网络；微博系统；node js

## ABSTRACT

As the newest delegate of WEB 2.0, micro blog is a system based on user relationship. Through WEB, WAP, and other client, the users can build the personal community in order to share information, issue information and get information. Micro blog is a social network which combines medium, internet and mobile communication. With the characteristics of brief content, rapid spread transmission, high real-time and strong interaction, micro blog can meet the requirements of full sharing and exchange information. In micro blog, as a audience, the user can browse the information of his interest, and as a publisher, he can publish the information for others to view. That is to say, in micro blog, everyone can speak, and everyone can listen. In fact, Micro blog is a kind of spread medium, its ultimate goal is carrying the messages to obtain the maximum spread effect.

According to the practical application, A micro blog system based on node js has been designed and implemented in this thesis. The users of the micro blog system based on node js are grouped into two types. They are normal user and admin user. After passing identity test and verify, the valid uses of the system have some operation limits, such as attention, comment, forwarding, inserting pictures, inserting topic, private message, editing, video, music, search, paging, collection, recommendation, bach-up and so on.

In order to separate data access and business logic, and make the pages dynamic, The system uses B/S architecture. It contains three layers. They are data layer, business logic layer and expression layer. The system based on node js uses node js as its framework, MongoDB as the primary storage database, and Redis database as the primary cache. The logics and page effects of foreground are realized through PHP+HTML+CSS+JavaScript.

The main research work includes the following aspects: Briefly introduce the background of the project, micro blog system and its developments; starting from the framework of micro blog system, introduce the related technologies about node js, MongoDB and Redis; according to the thinking in software engineering, complete the task of requirement analysis, design, implement and test for the micro blog system based on node js.

## ABSTRACT

---

The micro blog system based on node js can realize user management, micro blog management, user relationship management, comments, private letter, user search, video sharing, pictures management, short URL, sensitive words, topics, bach-up and other functions. With the characteristics of lightweight and easy to array, the micro blog systemsuit the small and medium-sized application.

**Keywords:** Social network, Micro blog, node js

## 目 录

第一章 绪 论 .....	1
1.1 课题背景 .....	1
1.2 微博系统及其发展动态 .....	1
1.3 论文的主要研究内容和组织结构 .....	6
第二章 相关技术研究 .....	8
2.1 node js 简介 .....	8
2.2 非关系数据库 NoSQL .....	9
2.3 面向文档的分布式数据库 MongoDB .....	11
2.4 基于内存的键值对存储数据库 Redis .....	12
2.5 本章小结 .....	14
第三章 微博系统的需求分析 .....	16
3.1 系统的功能需求 .....	16
3.2 系统的业务流程分析 .....	18
3.3 本章小结 .....	19
第四章 微博系统的设计 .....	20
4.1 系统的设计原则 .....	20
4.2 系统的架构设计 .....	20
4.3 系统的功能结构设计 .....	23
4.4 普通用户的用例 .....	23
4.5 管理员用户的用例 .....	24
4.6 微博系统的运行环境 .....	25
4.7 本章小结 .....	25
第五章 微博系统的实现 .....	26
5.1 系统的实现策略 .....	26
5.2 核心功能模块的实现方案 .....	27
5.2.1 用户管理 .....	27
5.2.2 微博管理 .....	30
5.2.3 用户关系管理 .....	38
5.2.4 评论和私信 .....	39
5.2.5 搜索用户和搜索微博 .....	41

5.2.6 视频分享和发布图片 .....	43
5.2.7 短链接生成 .....	49
5.2.8 敏感词处理 .....	51
5.2.9 分页 .....	53
5.2.10 备份 .....	53
5.2.11 系统缓存 .....	54
5.3 系统实现 .....	55
5.3.1 数据库连接池 .....	55
5.3.2 后台接口的实现 .....	56
5.3.3 前台的实现 .....	57
5.4 本章小结 .....	58
<b>第六章 微博系统的测试</b> .....	<b>60</b>
6.1 功能测试 .....	60
6.2 性能测试 .....	61
6.3 本章小结 .....	63
<b>第七章 结论</b> .....	<b>64</b>
<b>致 谢</b> .....	<b>65</b>
<b>参考文献</b> .....	<b>66</b>



## 第一章 绪 论

### 1.1 课题背景

几百年前，人们读长篇小说、看歌剧、听交响乐，二十世纪，人们读报纸和杂志、看电影和电视、听流行音乐，进入信息时代，人们上网、读博客、看视频。促成这些风气进化的是信息的产量与传播速度的剧增，为处理海量且迅速更新的信息，微博网站就诞生了。

微博，是微博客(Micro blog)的简称，它是一个基于用户关系的系统，用户能够通过万维网(World Wide WEB, WEB)和无线应用协议(Wireless Application Protocol, WAP)等各种客户端，在该系统中组建个人社区，以实现分享信息、发布信息和获取信息的目标。微博将媒体、互联网与移动通讯技术聚合为一，借以内容简短、传播快速、实时性强、互动性强的特性满足人们充分分享信息和交流信息的需求。在微博这个社交网络中，用户不但可以作为观众，阅读自己感兴趣的信息，而且也能够作为信息的发布者，发布信息供其他人阅读。也就是说，在微博中，人人都可以发言，人人都能够收听。2013年6月，中国互联网协会发布的《2012—2013年微博发展研究报告》指出：2012年以来，民众、政府、社会组织等多种角色积极参与微博互动，构建起了微博平台下的新的社会生态。公众、政府和社会组织成为了微博社会中最为重要的三种社会群体<sup>[1]</sup>，因而微博成为了功能全面的网络社区。

微博系统作为WEB 2.0的最新代表，需要采用先进的解决思路和解决方案。如何满足中小型微博系统的高性能要求，正是本文需要解决的问题，因此本文设计并实现一个基于node js的微博系统，实现用户之间相互关注、发布新微博、主页微博内容定时刷新等功能。基于node js的微博系统后台框架采用node js技术，主要存储数据库采用MongoDB，主要缓存数据库采用Redis。基于node js的微博系统具有轻量型、易布署的特点，能够为需要推出自己的微博网站的站长们使用，适合于中小规模用户量的微博应用。

### 1.2 微博系统及其发展动态

#### 1. 微博的定义

“维基百科”对微博的定义为：微博是微博客的简称，是一种允许用户及时更新简短文本并公开发布的博客形式。随着微博的发展，有些微博系统也允许用

户发布包括图片或影音等多媒体<sup>[2]</sup>。微博系统提供了多种发送信息的方式,包括实时消息软件、短信、电子邮件、网页、MP3 等。用户可以允许所有人来阅读他的微博内容,也可以只允许自己选择的某些群组来阅读他的微博内容。

## 2. 国外微博系统的发展动态

美国的 Twitter 是最早也是最著名的微博, Twitter 是最早发明了微博核心概念的。2006 年 3 月, 博客技术先驱 blogger 的创始人 Evan Williams 创建了 Twitter。最初, 用户使用 Twitter 只能向好友的手机发送文本信息, 后来发展为能够使用包括手机短信在内的几百种工具发送文本、图片、影音等多媒体信息<sup>[3]</sup>, 是 Twitter 将大众的视野引入了微博世界。Twitter 利用无线网络、有线网络、通信技术进行即时通讯, 用户可以将自己的最新动态和想法使用最多 140 字的文字表达出来, 通过 Twitter 网站、Twitter 客户端软件 Twitterrific、短信、电子邮件等方式输入, 以短信形式发送给手机和个性化网站。

早期的 Twitter 是采用 Ruby on Rails 框架语言编写的, 由于考虑到执行效率等因素并没有采用 Vanilla 的 Ruby 进行部署, 而是在 Ruby 的企业版上进行了部署。从 2007 年春季到 2008 年, Twitter 的原始消息是通过一个名字叫做 Starling 的持续性数据结构服务器进行处理的, 从 2009 年起, Twitter 开始逐渐使用 Scala 编写的程序进行消息处理, 并且它提供的 API 允许众多网站与 Twitter 进行集成。2011 年, Twitter 的编程语言从 Ruby on Rails 替换为 Java、数据库从 MySQL 替换为到 Lucene, 这使 Twitter 的性能增加了三倍。

Twitter 的主要竞争对手是 Plurk 和 Jaiku。Plurk 是由 A-team 创建的一个提供基于时间轴的可视化微博服务的多种语言支持的社交网站, 它的最大特色就是在一条时间轴上来显示自己和好友的消息更新和回复。Plurk 在港澳台相当流行, 中国大陆地区 2009 年 4 月起暂时无法使用。Jaiku 是由芬兰的 Jyri Engeström 及 Petteri Koponen 开发的一个社交网络, 于 2006 年 7 月正式运行, 它提供了“微博”及“人生转播”服务。Jaiku 是由基于 Python 的 Twisted 架构编写的, 兼容 S60 平台移动电话。Jaiku 有此平台的客户端软件, 可经由此软件, 用手机更新 Jaiku。另外 Jaiku 还有发表 API 供程序员制作第三方软件。Jaiku 的最大特色是“人生转播”功能, 该功能能够将用户的多种网络活动纪录集成起来, 例如, Flickr 相片、Last.fm 音乐及手机更新的位置数据。

## 3. 国内微博系统的发展动态

受 Twitter、Plurk 和 Jaiku 的启发, 中国也开始了对微博的探索。校内网的创始人王兴在 2006 年就开始了对微博系统的开发, 2007 年 5 月他创建了饭否网, 而饭否网也成为了中国第一个微博网站。腾讯作为拥有大量 QQ 用户的企业, 为了



满足用户随时随地分享信息的强烈需求，开发了腾讯滔滔，并于 2007 年 8 月 13 日上线。然而，此时中国的微博网站拥有的用户仅有几十万个，每个月需要处理的信息也只有几千万条。因而，中国的微博将目光转移投向了产品调整与服务完善上<sup>[4-8]</sup>。2009 年 7 月，饭否网和腾讯滔滔等许多微博产品都终止了运营，而 Follow5、9911 等一些新的微博产品开始走进人们的视野，尤其是 2009 年 7 月 19 日 Follow5 在孙楠大连演唱会上亮相，开启了中国将微博引入大型演艺活动的序幕。

2010 年中国的微博迎来了春天，新浪网、腾讯网、网易和搜狐网这四大门户网站都相继推出了微博服务，而新浪微博、腾讯微博、网易微博和搜狐微博也成为了国内用户量最大的微博网站。2012 年 3 月，新浪微博、网易微博、腾讯微博与搜狐微博共同正式实行微博实名制。

新浪网作为第一家提供微博服务的门户网站，于 2009 年 8 月 14 日推出了“新浪微博”内测版，一个月之后，新浪微博又添加了@功能、私信功能、评论功能，以及转发功能，从此微博正式进入中文上网主流人群视野。新浪微博在架构设计及后台实现方面采用经典的 LAMP 架构、MemCached 缓存、CDN 技术，以及异步设计、消息队列的方式。新浪微博就像一个类似于 Twitter 和 Facebook 的混合体，用户可以通过手机短信、手机客户端、网页、WAP 页面等多种方式随时随地发布文本消息、上传图片或链接视频，观众在第一时间就能够浏览到他发布的信息。新浪微博可以直接在一条微博里面发送图片，也可以直接在一条微博下面附加评论，它是最先在微博中添加这两项功能的。同时新浪微博还推出了@功能和私信功能，目前，@已经成为了网络流行语，几乎所有的 SNS 网站都实现了这一功能。Sina App Engine Alpha 版于 2009 年 11 月上线，Sina App Engine Alpha 版的出现使得微博更加多元化，用户发布信息可以通过丰富的 API 使用第三方软件或者插件来实现。新浪微博的群组功能产品—新浪微群于 2010 年 11 月开始内测，新浪微群具有通讯与媒体传播的双重功能，因此人们将它视为网页版的 QQ 群。根据 2010 年官方公布的数据显示，新浪微博已达到 5000 万个用户，每天发布的微博数超过了 2500 万条，新浪微博从此成为了中国最具影响力的、最受瞩目的微博。2011 年 3 月，新浪微博替换掉 sinaURL，启用了短域名 t.cn 服务。2011 年 4 月，新浪微博独立启用了微博拼音域名 weibo.com，2011 年 5 月，新浪微博将原先的链接正式跳转至 weibo.com，用户地址也变成了 weibo.com/####，从此新浪微博正式结束了双域名并存战略局面。2012 年 1 月 5 日，新浪微博的又一个新的功能——悄悄关注正式上线。2013 年 1 月 11 日，新浪微博与网易有道达成战略合作协议，网易有道作为新浪微博首家翻译合作伙伴为用户提供免费的外语微博翻译服务。2013 年 1

月 31 日, 新浪微博客户端推出的 3.3.0 版本新增了“密友”功能, 强化了私密社交圈。截至 2012 年底, 新浪微博注册用户数超过了 5 亿, 日活跃用户数达到了 4620 万。由于目前新浪微博活跃用户中有 75% 的用户使用移动终端, 因此未来新浪微博将继续改善用户体验, 扩大微博的商业化规模, 并将所有产品的聚焦点都放到移动互联网。

腾讯微博有网页版、手机版和客户端版等多个版本, 用户可以通过网页、手机、QQ 客户端、QQ 空间以及电子邮箱等各种途径使用腾讯微博。2010 年 4 月, 腾讯微博开始小规模内测, 只开放了文字形式, 暂时未对外开放图片功能, 字数也限制为 140 字。腾讯微博的特点在于细致的产品功底和庞大的用户群, 5 月 1 日腾讯微博开放了用户邀请注册, 2011 年 2 月, 腾讯对外宣布, 其微博注册用户突破一亿大关。2011 年 4 月, 腾讯微博不断提升功能, 相继推出了邮件分享、本地上传视频、图片版微博、开放式上墙等服务。5 月份, 借助内容一键互通功能, 腾讯微博正式实现了微博与 QQ、QQ 群内容之间的双向互通, 这样用户不仅可以将在 QQ 聊天窗口中的信息一键分享至微博, 而且也能够通过快捷操作将微博平台上有价值的资讯即时分享给 QQ 好友和 QQ 群好友。2011 年 9 月, 腾讯微博的注册用户数超过 3.1 亿, 日活跃用户数超过 5000 万人。2011 年 10 月, 腾讯微博个人开放认证正式上线并增加了独具特色的微博等级服务来反映用户在微博中的活跃情况和受欢迎程度。在拥有了强大的 QQ 平台下, 腾讯公司并未打算把腾讯微博作为战略级产品推出, 而更多的是为了遏制对手, 起到战略防御的作用, 然而腾讯微博依然呈现出了发展迅猛的姿态, 截至 2012 年底, 腾讯微博注册用户数已达到 5.4 亿, 日均活跃用户数超过了 1 亿。未来腾讯微博将从产品和运营角度, 承担更多的社会责任和媒体责任, 致力于推动微博内容的多元化, 挖掘和助推资讯、文化思想交流与生活化价值信息的传递。

网易微博于 2010 年 1 月 20 日正式上线内测, 网易微博的定位是“简单的分享”。在色彩布局方面和整体设计方面, 网易微博都继承了 Twitter 的简约风格。网易微博把用户的建议与意见放到了非常重要的位置。在信息交互方面, 网易微博摒弃了“回复提醒”这一繁琐的功能, 采用了 @ 的方式来实现用户之间的交流。在信息提醒方面, 与新浪微博的右侧小范围提醒不同, 采用了 Twitter 式的 Ajax 免刷新设计的横条, 显著地扩大了可点击范围。在话题搜索快捷插入功能中, 有别于新浪微博的“#话题#”, 使用单个 #, 例如“#话题”的形式, 并将“#意见反馈”放到内容框下更显眼的位置, 实现了用户插入话题的便捷性和易用性。

2010 年 4 月, 搜狐微博上线正式公测, 用户也是通过手机短信、网页、WAP 页面, 以及第三方应用等多种方式发布消息, 但是搜狐微博打破了微博客服务存

在字数限制的传统。2010 年 10 月, 搜狐微博负责人表示搜狐微博是国内唯一一家对于微博消息不设置字数限制的微博服务。2011 年 6 月 28 日, 搜狐微博进行了改版, 其视觉层面全面转向宽版效果, 并在用户体验以及浏览速度等方面进行了一系列优化, 为微博用户提供了更加舒适、快捷的阅读体验。目前, 搜狐微博正在尝试打通诸如博客、社区、搜狐焦点、校友录等各搜狐产品, 试图进一步发挥搜狐的矩阵优势。

### 4. 微博的特点

2010 年末, “微博”成为了各媒体最为关注的一个互联网名词, 微博开启了一场正在爆发的互联网革命, 人们已经进入微博时代。目前, 微博的应用不但渗透到了越来越多的领域, 而且还带来了令人瞩目的发展<sup>[9]</sup>。

相对于论坛、门户系统等传统的 WEB 应用, 微博系统具有如下一些特点:

(1)海量用户: 据中国互联网信息中心(China Internet Network Information Center, CNNIC)、互联网实验室(China Labs)的数据统计, 2013 年第一季度, 新浪微博注册用户数量达到了 5.36 亿, 日活跃用户数达到了 4980 万, 2012 年第三季度, 腾讯微博注册用户数量达到了 5.07 亿, 微博已经成为中国网民上网的主要活动之一。

(2)海量数据: 互联网数据中心(Data Center of China Internet, DCCI)的数据显示: 2012 年, 中国微博用户平均每天发表 2.13 条微博, 并转发 3.12 条微博。

(3)高频率访问: DCCI 对新浪微博等各平台微博用户的访问频率调查发现, 各平台微博用户的主体是平均每天使用一次及以上的高活跃用户, 这类用户达到了 70% 以上。在这些高活跃用户中, 在平均每天使用 6 次及以上的超级活跃用户方面, 新浪微博表现尤为突出, 达到了 19.2%。

(4)多元化媒介: 微博应用不仅可以通过 PC 端访问, 而且也能够通过智能手机和平板电脑等移动终端访问。2013 年第一季度, 通过移动终端登录访问新浪微博的移动终端活跃用户超过了 3800 万, 占总活跃用户的比例大于 76%, 移动化已经成为微博应用的主流。

而微博与博客相比具有如下一些特点:

(1)内容短小精悍: 微博的内容限定为 140 字之内, 强调了内容的微型化与简明性, 能够简明扼要地反映用户的行为、状态和观点。博客的出现将互联网上的社会化媒体向前推进了一大步, 但是博客上博文的创作需要考虑完整的逻辑, 这给博客作者带来很重的负担。微博内容简短、创作门槛较低, 倡导简洁、个性化的自由行为。

(2)随时随地传播信息: 微博网站可以通过手机、互联网等多种渠道发布信息,

用户可以发布文本、图片和视频等多媒体信息，真正实现了随时随地发布信息、接收信息。

(3)信息交互简便快捷：微博网站具有关注、转发、评论、回复、私信等丰富功能，这些功能确保了在用户之间进行时实的信息交互。

(4)信息自传播速度快：博客是靠网站推荐带来流量的，而微博是通过粉丝转发增加阅读数的，利用转发功能使得博客信息快速传播。

(5)创新交互方式：与博客上面对面的表演不同，微博上采用背对脸的交流方式，用户可以选择跟随的对象，在自己的个人空间中就会显示被跟随对象的状态更新，而被跟随对象既可以主动也可以不主动和你交流。跟随可以是一点对多点的，也可以一点对一点的，微博的交互方式真正实现了人际交流的时效性。

(6)原创性：每条微博内容的长度限制在 140 字左右，这使得用户能够更容易、更自由地表达自己的动态和想法，因而在微博上就爆发性地产生了大量的原创内容。

### 1.3 论文的主要研究内容和组织结构

由于 Twitter、新浪微博等平台需要同时支持大量的人数在线，每天的数据量非常大，所以后台架构非常复杂。而基于 node js 的微博系统的目的是设计并实现一个中小型的微博网站，实现对中小用户量的支持，完成用户之间相互关注、发布新微博、主页微博内容定时刷新等功能，并在发布微博时，增加短 URL 地址、表情、@其它人等功能，因此本项目特点是轻量型、易部署，更适合于小网站使用。

本论文主要研究的是：采用 node js 框架，使用 MongoDB 和 Redis 两个 NoSQL 数据库开发微博系统的方法。

本论文的结构为：

第一章绪论：简要介绍课题背景，微博系统及其发展动态，以及本论文的主要工作内容。

第二章相关技术研究：从微博系统的基本架构出发，介绍本系统涉及的几种相关技术，着重介绍 node js 架构、Redis 数据库和 MongoDB 数据库。

第三章微博系统的需求分析：对于微博系统进行了需求分析，阐述了微博系统的功能需求，并给出了微博系统的业务流程。

第四章微博系统的设计：对于微博系统进行了设计，给出了微博系统的设计原则，描述了微博系统的拓扑结构、架构和功能结构，进行了用例设计，确定了系统的运行环境。

第五章微博系统的实现：阐述了微博系统中核心模块的具体实现方案，描述

了微博系统的实现，并展示了实现效果图。

第六章微博系统的测试：对微博系统进行了测试与性能分析，给出了功能测试和性能测试的结果，分析了微博系统的性能。

## 第二章 相关技术研究

### 2.1 node js 简介

node js 是一个能够快速构建网络服务与应用的平台, node js 平台的构建是基于 Chrome's JavaScript Runtime 的<sup>[10]</sup>, 也就是说, node js 实际上是对应用于 Google Chrome 浏览器的 Google V8 引擎进行封装, 而 Google V8 引擎执行 Javascript 具有非常快的速度和非常好的性能。

为了使得 Google V8 在非浏览器环境下运行得更好, node js 优化了一些特殊用例, 提供了一些替代 API。例如, 通常在服务器环境中, 对于二进制数据的处理是必不可少的, 但是 Javascript 对于二进制数据的处理支持不足, 为此, node js 增加了 Buffer 类, 以方便并高效地对于二进制数据进行处理。因此 node js 不仅使用了 Google V8, 而且还对 Google V8 进行了优化, 使其浏览器环境和非浏览器环境下都更加给力<sup>[11]</sup>。Google V8 引擎本身使用了一些先进的编译技术, 这使得用 Javascript 这类脚本语言编写的代码与用 C 这类高级语言编写的程序在性能上相差无几, 而使用 Javascript 这类脚本语言的开发成本却很低。

性能是 node.js 考虑的重要因素, node js 以单进程、单线程模式运行, 它采用事件驱动机制, 使用一种称为“事件循环”(event loop)的架构, 编写出可扩展性高的服务器<sup>[12,13]</sup>。node js 采用一系列“非阻塞”库来支持“事件循环”的方式, 面对大规模的 HTTP 请求, 通过内部单线程高效率地维护事件循环队列, 没有多线程的资源占用和上下文切换, 它本质就是为数据库和文件系统之类的资源提供接口, 以可扩展的方式简化对慢资源的访问。node js 的“事件循环”架构选择不但能够提高性能, 而且还能够减低开发复杂度。

虽然使 Javascript 运行于服务器端不是 node js 的独特之处, 但是它却是其 node js 的一个强大功能。众所周知, 浏览器环境限制了对编程语言的选择, 服务器程序与日益复杂的浏览器客户端应用程序之间的代码共享只能通过 Javascript 来实现。而 node js 已经成为支持 Javascript 在服务器端运行的有效平台。node js 具有以下优点:

1. 在不新增额外线程的情况下, node js 仍然能够对任务进行并行处理。

node js 是单线程的, 它通过事件循环机制来实现并行操作。对此, 程序员应该充分利用这一特点, 尽量多使用非阻塞操作, 而避免使用阻塞操作。

2. 为了简化应用开发, node js 使用 Module 模块划分不同的功能。

Module 模块类似于 C 语言中的类库。node js 的每个模块都包含非常丰富的各

类函数，例如，HTTP 模块包含了和 HTTP 功能相关的很多函数，可以帮助开发者对 HTTP、TCP/UDP 等进行操作，还可以帮助开发者创建 HTTP 和 TCP/UDP 的服务器。

3. 为了实现网络服务，node js 采用事件驱动机制、异步编程风格。

node js 的设计思想是以事件驱动为核心，它提供的绝大多数 API 都是基于事件、异步风格的。例如，在 Net 模块中，net.Socket 对象就有 connect、data、end、timeout、drain、error、close 等事件，使用 node js 的开发者需要根据自己的业务逻辑注册相应的回调函数，这些回调函数都是异步执行的。事件驱动、异步编程的优势在于，执行代码无须阻塞以等待某种操作完成，使得有限的资源能够用于其他的任务，真正实现了系统资源的充分利用。

node js 支持的编程语言是 Javascript，Javascript 的匿名函数和闭包特性特别适合事件驱动、异步编程风格，而且与 Python、Ruby 等动态语言相比，Javascript 在动态语言中性能突出，这也是 node js 的高性能的重要因素之一<sup>[14]</sup>。虽然 node js 刚刚诞生两年多，但是它的发展势头正在赶超 Ruby/Rails，node js 具有丰富的企业应用成功案例，例如，在知名社交网站 LinkedIn 最新发布的移动应用中，其整个移动软件平台都由 node js 构建而成的；著名项目托管网站 GitHub 也尝试了 node js 应用，此 node js 应用是一个称为 nodeLoad 的存档下载服务器；不只在外国，在国内 node 的优点也同样吸引了开发人员的关注，淘宝就是一个成功应用 node js 技术的典范。

## 2.2 非关系数据库 NoSQL

非关系数据库(Not Only SQL, NoSQL)作为一个新兴的领域，是随着 WEB 2.0 应用的兴起而发展起来的。在开发 WEB2.0 网站，特别是在开发超大规模、高并发的 SNS 类型的纯动态网站时，对于数据库的高可扩展性与高可用性、海量数据的高效存储与访问、数据库高度并发读写的需求等问题，传统的关系数据库已经无能为力了，而非关系数据库的出现解决了关系数据库难以克服的问题<sup>[15]</sup>。

1. WEB2.0 网站对数据库的高可扩展性与高可用性的需求：在 WEB 系统架构中，数据库的水平扩展是最难以实现的。当一个应用系统的用户量与日俱增、访问量急剧增大的时候，其数据库却不能与 WEB Server、APP Server 一样通过添加更多硬件和服务节点的方式对性能与负载能力进行简单地扩展，而需要通过停机维护和数据迁移的方式对数据库系统进行升级和扩展。对于那些需要提供 24 小时不间断服务的网站来说，进行这种数据库系统的升级和扩展是非常痛苦的事情。

2. WEB2.0 网站对海量数据的高效存储与访问的需求：对于诸如大型 SNS 网

站的 WEB2.0 网站，每天都要产生海量的用户信息，例如 Friendfeed 平均每个月都有 2.5 亿条用户动态被发布。对于关系数据库而言，如果在一张具有 2.5 亿条记录的表中进行 SQL 查询，那么其查询效率是用户无法忍受的。

3. WEB2.0 网站对数据库高度并发读写的需求：由于 WEB2.0 网站必须根据用户的个性化需求来为用户实时生成动态页面并及时提供动态信息，致使 WEB2.0 网站基本不能使用动态页面静态化技术。WEB2.0 数据库通常需要进行每秒上万次的读写操作，其并发负载是非常高的。关系数据库虽然勉强能够处理每秒上万次的 SQL 查询，但是要响应每秒上万次的读写数据请求，硬盘 I/O 是不能承担的。

在 WEB2.0 网站的开发中，关系数据库的数据库事务的一致性、数据库的写实时性与读实时性、复杂的 SQL 查询，尤其是多表关联查询等许多特性都没有了用武之地。

1. 数据库事务的一致性：许多 WEB2.0 系统对数据库事务的一致性要求并不严格，具体地说，WEB2.0 系统对于读一致性的要求非常低，甚至在有些系统中对于写一致性的要求也很低，因此在 WEB2.0 数据库高负载环境下，数据库事务的一致性管理就变成了沉重的负担。

2. 数据库的写实时性与读实时性：在关系数据库中写入一个数据之后就立刻对其进行查询，是一定能够读出来这个数据的。因此在实时性方面，关系数据库并没有太高的要求。

3. 复杂的 SQL 查询，尤其是多表关联查询：大数据量的 WEB2.0 系统，尤其是 SNS 类型的网站都很忌讳复杂的 SQL 报表查询和多个表间的关联查询，在 WEB2.0 系统的需求分析和设计过程中就首先避免了这种情况的查询，只使用单表的主键查询或者单表的简单条件分页查询，因此在 WEB2.0 系统中，SQL 的查询功能被高度的弱化了。

在 WEB2.0 应用场景下，使用关系数据库进行数据存储就不适合了，而非关系数据库却能够适应 WEB2.0 应用对数据存储的需求。

非关系数据库却与关系型数据库的设计理念是完全不同的<sup>[16]</sup>。关系数据库中的表是格式化的数据结构，在一个表中，每个记录包含的字段都是相同的，尽管不是每个记录都需要所有的字段，但是关系数据库还是会为每个记录来分配所有的字段。关系数据库的表结构有利于表与表之间的连接等操作，但这也是导致关系数据库性能瓶颈的一个重要因素。而非关系数据库存储的是键值对，其结构不是不固定，每个记录都可以根据需要包含一些只属于自己的字段。这样，非关系数据库就能够大大减低数据存储的时间开销和空间开销。

目前比较成熟的 NoSQL 产品包括：Cassandra、Mongodb、CouchDB、Redis、



Riak、Membase、Neo4j 和 HBase。而 Google 的 BigTable 和 Amazon 的 Dynamo 都是商业 NoSQL 实现的成功案例。

非关系数据库具有可以处理超大量的数据、可以运行在便宜的 PC 服务器集群上、没有过多的操作等特点，以及易扩展、大数据量、高性能、灵活的数据模型、高可用等优点。与关系型数据库相比，非关系数据库在大关系数据存取方面具有显著的性能优势<sup>[17,18]</sup>。

## 2.3 面向文档的分布式数据库 MongoDB

MongoDB 是一个基于分布式文件存储的数据库，它是 2007 年 10 月由 10gen 团队开发，2009 年 2 月首度推出的。MongoDB 是使用 C++ 语言编写的，它能够为 WEB 应用提供可扩展的、高新能的数据存储<sup>[19]</sup>。

MongoDB 中的数据能够使用复杂的层次嵌套，而在此情况下依然可以建立索引并进行查询，因此 MongoDB 能够管理大量文档，应用程序能够通过更自然的方式建立模型<sup>[20]</sup>。

MongoDB 是功能最为丰富的、最像关系数据库的一个非关系数据库产品，使用 MongoDB 存储数据是非常方便的。MongoDB 具有高新能、易部署、易使用等特点<sup>[21,22]</sup>。

### 1. MongoDB 的数据模型和文档格式

MongoDB 支持的数据结构非常松散，在 MongoDB 中，每个数据库里可以有多个集合(collection)，集合里存储的是文档(document)。也就是说，在 MongoDB 数据库中，数据被分组存储在集合中。文档具有一组字段(field)，一个字段为一个键值对(key-value)，其中键(key)用于唯一标识一个文档，是字符串类型的，值(value)可以是各种复杂的类型，例如，可以是二进制数据、整数、浮点数、字符串、时间戳等基本类型，也可以是一个文档或者一个数组。MongoDB 的文档的这种存储形式为 BSON (Binary Serialized Document Format) 格式。

BSON 是一种二进制序列化文档格式，BSON 支持内嵌数组和内嵌文档，并且 BSON 具有表示数据类型的扩展，例如，Date 类型和 BinData 类型。BSON 具有模式自由的特性，也就是说，在 MongoDB 数据库中，人们不需要知道数据库中文档的结构定义，如果需要的话，人们可以在同一个数据库中存储不同结构的文档，并且在任意一个或一些文档中新增字段或删除字段都不会对其他的文档产生影响<sup>[23]</sup>。

### 2. MongoDB 的查询与索引

MongoDB 支持非常强大的查询语言，它的语法有些类似于面向对象的查询语

言，它几乎能够实现类似于关系数据库中单表查询的绝大多数功能。与关系数据库查询操作最大的不同是，MongoDB 不支持关联查询，也就是说，MongoDB 不支持 join 运算。但是，在 MongoDB 中能够通过客户端驱动多次访问数据库的方式来模拟实现的关联查询。为了，MongoDB 采用“引用”(References)的概念来表示关联关系，这里的“引用”与关系数据库的外键(Foreign Keys)具有与类似的意义。

MongoDB 不但支持字段匹配查询、正则表达式查询，而且也支持查询内嵌数组。

MongoDB 的索引包括组合索引和唯一索引，索引的建立不仅可以对于字段进行，而且也可以对于内嵌数组进行。MongoDB 还可以删除索引。默认情况下，每个集合都有一个唯一的索引\_id，如果在插入数据时没有指定\_id，服务会自动生成一个\_id。

### 3. MongoDB 的映射框架

Morphia 是一个对象关系映射框架，它对 MongoDB 数据库 Java 版驱动进行了轻量级的对象封装，实现了 MongoDB 数据库的文档和 Java 对象之间的映射，使得 Java 程序员可以十分容易地使用面向对象编程的思维来操作 MongoDB 数据库。Morphia 是一个轻量级的类型安全的 Java 对象 to/fromMongoDB 库。根据 Java 抽象对象和 MongoDB 文档的特性，Morphia 建立了类属性与文档字段的映射、类与集合的映射，以及类间关系与引用 DBRef 及内嵌文档间的映射的实现机制。

## 2.4 基于内存的键值对存储数据库 Redis

### 1. 分布式缓存

传统的缓存是存在于单机机制下的，然而多机器、多进程环境需要的是网络分布式缓存的<sup>[24]</sup>。分布式缓存提供的数据内存缓存可以分布于大量的单独的物理机器中，也就是说，分布式缓存所管理的机器实际上是一个集群，它负责维护这个集群中成员列表的更新一集执行各种操作。分布式缓存可以让不同主机上的多个用户同时访问内存，因此不但解决了共享内存只能单机应用的局限性，而且也不会出现使用数据库时磁盘开销和阻塞的现象<sup>[25-27]</sup>。

在 WEB 1.0 时代，通常采用页面缓存技术来存储那些经常被访问的整个页面，其中的典型代表就是代理缓存。在 WEB 2.0 时代，WEB 应用以个性化数据为主、粒度更细，缓存的真正意义是用来存储动态数据和事务数据，页面缓存技术难以满足应用需求<sup>[28]</sup>。分布式缓存具有过期机制和替换机制两个重要特性。

随着时间的迁移，内存中存储的有些数据不被经常访问甚至从不被访问，这就降低了内存的使用效率。过期机制允许程序员制定动态数据在缓存中保留的时

间, 过期数据就会被删除。过期机制通常有绝对时间过期机制和空闲时间过期机制两种方式。绝对时间过期是指定某个时间后删除数据, 空闲时间过期机制是指某个数据在一个时间段内没有被任何进程访问, 就删除这个数据。

在许多情况下, 缓存的大小是受内存容量的限制。程序员可以使用替换机制来制定最大缓存的大小, 当缓存中存储的数据量达到这个最大缓存的限制时, 就启动一个替换线程运行, 将缓存中的数据删除以便腾出空间给新的数据。

MemCached 是目前最流行的一种分布式缓存。MemCached 是一种分布式的内存对象缓存系统, 用于在动态应用中减少数据库负载, 提高访问速度。MemCached 以键值对(key-value)的方式保存数据, 能够存储包括图像、视频、文件和数据库检索的结果等各种格式的数据, 其原理是通过在内存中维护一个统一的巨大的哈希表。虽然 Memcached 是一种分布式缓存, 但是实际上服务器端没有实现任何分布式的功能, 所有分布式功能都需要客户端实现, 这就是零共享机制。

## 2. 基于内存的键值对存储数据库 Redis

Redis 是一个基于内存的键值对存储数据库, 它是 2010 年 3 月由 VMware 主持开发的, 使用 ANSI C 编写而成, 可以用作分布式缓存。Redis 的出现在很大程度上弥补了 MemCached 这类 key-value 存储的不足, 在某种程度上可以对关系数据库起到很好的补充作用, Java、C、C++、Python、Ruby 等许多语言都包含 Redis 支持<sup>[29]</sup>。

Redis 支持存储的 value 类型比较多, 包括: 字符串(string), 链表(list), 集合(set), 有序集合(sorted set)和哈希表(hash)<sup>[30]</sup>。这些数据类型都支持 push、pop、add、remove, 交集、并集、差集, 以及更加丰富的操作, 而且这些都是原子性操作。在此基础上, Redis 还支持各种不同方式的排序。对于微博系统来说, Redis 中的有序集合具有重要的实用价值, 它可以作为收取消息的信箱。

Redis 是一个数据结构类型的数据库, 不是单纯的 key-value 存储。Redis 存储的内容是二进制安全(Binary Safe)的<sup>[31]</sup>, 二进制安全是指数据在传输过程中采用二进制形式并保证数据的安全性, 包括加密等。

Redis 使用全量数据和增量请求两种文件格式。全量数据格式是指将内存里的数据写入磁盘, 以便下次读取文件进行加载; 而增量请求文件是指将内存里的数据序列化为操作请求, 以用于读取文件进行 replay 得到数据。

Redis 的存储包括内存存储、磁盘存储和 LOG 文件三个部分, 在配置文件中有三个参数进行存储配置, 包括: 按照时间的长短和更新操作次数的多少将数据同步到数据文件, 是否在每次更新操作后进行日志记录, 选择操作系统/手动进行数据缓存同步到磁盘。

为了保证效率, Redis 采用了与 MemCached 相同的方法, 将数据都缓存在内存里, Redis 工作在内存数据集里, 而与 MemCached 不同的是, Redis 可以根据使用情况灵活地选择数据集的持久化方案, 周期性地将更新的数据写入磁盘, 或者将修改操作写入日志记录文件, 并在此基础上, 实现了主从(master-slave)同步。数据能够从主服务器向任意多个从服务器同步, 从服务器也可以是关联其他从服务器的主服务器。主从同步使得 Redis 可以执行单层树复制, 从盘可以有意也可以无意地对数据进行写操作。由于 Redis 实现了发布/订阅机制, 使得从服务器在任何位置同步树时, 都可以订阅一个频道并接受主服务器的完整的消息发布记录。主从同步对读操作的可扩展性和数据冗余的处理具有很大帮助。

客户端与 Redis 服务器之间通过 TCP6379 端口进行连接, 它们之间传送的每个 Redis 命令或者数据都是以“\r\n”(CRLF)结束的。服务器接收命令之后, 执行命令, 然后将一个答复发回客户端。从 Redis 1.2 版本, Redis 就采用了新的统一协议, 而从 Redis 2.0 版本开始就成为了与 Redis Server 交互的标准方式。Redis 的协议采用了一种折中方案, 它主要平衡了容易实现、机器快速解析与容易被人理解等因素。

Redis 的缓存服务需要占用大量的物理内存, 为了提高 Redis 数据库的容量, 除了可以把数据分割到多个 Redis 服务器外, 还可以使用虚拟内存(Virtual Memory)。Redis 的虚拟内存和操作系统的虚拟内存不是一回事, 但是它们的思路 and 目的都是相同的。Redis 的虚拟内存就是暂时将不经常访问的数据从内存交换到磁盘中, 以便腾出一些宝贵的内存空间来存储其他需要经常访问的数据。对于微博网站来说, 确实总是只有少量用户作为活跃用户, 因此只有一部分数据被经常访问, 使用虚拟内存不仅能够提供单台 Redis 服务器数据库的容量, 而且也不会对性能造成太大的影响。

## 2.5 本章小结

本章简要介绍了基于 node js 的微博系统所使用的相关技术。本微博系统所采用 node js 框架, node js 是一个能够快速构建网络服务与应用的平台, 以单进程、单线程模式运行, 采用事件驱动机制, 使用事件循环的架构, 能够编写出可扩展性高的服务器。

本微博系统使用的主要数据库 MongoDB 和缓存数据库 Redis 均为非关系数据库。使用非关系数据库进行数据存储不需要固定的表结构, 而且也没有连接操作, 非关系数据库满足了数据存储的横向伸缩性上的需求, 具有数据模型灵活、易扩展、大数据量、高性能、高可用性等特点, 与关系数据库相比, 非关系数据库在

大数据存取上具有明显的性能优势。

## 第三章 微博系统的需求分析

### 3.1 系统的功能需求

根据基于 node js 的微博系统的轻量化特性要求，本微博系统主要实现微博的发布、删除、转发、评论等功能，对多媒体内容的支持功能，以及权限管理等功能。微博系统的具体功能需求包括：

#### 1. 关注

在微博中，关注与粉丝之间是一种订阅关系。如果一个用户关注了某个人，那么它就订阅了这个人的消息。关注哪种类型的消息，用户是完全自由的。如果用户在添加关注时能够注意筛选来控制所关注的数量，那么不但能够大大减少垃圾信息，而且可以提升自己订阅消息的阅读价值。只有那些有传播或者广告需要的用户，才会大量关注别人，而普通用户只要善于使用“关注”功能，精心选择关注对象，就能够收获更多有趣的信息。

关注和粉丝菜单应能够查看是否双向关注，这样才方便细心的用户及时调整消息发布与订阅目标。例如，如果有一个长期互相关注并经常与其交流的用户突然取消了关注，用户就会分析原因：第一，自己是否发送信息的频率过高而影响了别人的正常阅读？第二，是否自己发布的消息阅读价值有所降低？第三，是否自己对别人的评论过于随意，其中强烈表达了自己的观点而对别人造成困扰？通过分析，找出原因，从而及时纠正自己的偏差来提升消息的阅读价值。有效控制“垃圾”信息才能够让微博这个信息平台更具备思想价值，而尊重别人也可以让自己的使用过程更加愉快。

#### 2. 评论

在微博中，将发表的@回复改为评论。这种评论既有利也有弊。其主要优点包括：在信息内容方面，将发表的@回复改为评论之前，遇到爱聊天的用户时，微博就变成了公众聊天室，其信息内容全部公开化了，而实际上，@回复的内容大多数是用户间的私人交流，这种信息公开化既有可能暴露隐私，又有可能对接收信息的人造成干扰。将发表的@回复改为评论之后，用户的评论不会在公共显示区出现，解决了上述问题。在信息显示方面，将发表的@回复改为评论之前，@回复逐条显示在主页上，这种显示方式非常分散，不方便以后用户查阅最初回复的主题和完整的对话记录。将发表的@回复改为评论之后，评论显示更加集中也更加一目了然。但是评论功能的缺点是：一些自己感兴趣的、有价值的评论非常容易被淹没在海量的灌水评论中。用户阅读大量评论需要花费时间，有些粉丝多的用户可能从来

不看评论，这样就会错过有益的交流，让微博这一交互信息平台变成了单向传播工具，又走回了博客的老路，丢弃了微博的优势。因此微博系统应对评论功能进行优化，为方便用户快速查看自己所关心的评论，节约阅读评论的时间，提高交流的质量和效率，可以将用户自己所关注的人的评论与陌生人的评论分开来显示。

#### 3. 转发

在微博中，转发包括转发消息和转发评论两种，把转发消息和转发评论分开，使得显示内容更加全面。当某条信息再次被转发时，系统会按照“//@用户名+内容”的显示格式自动添加转发者和他的转发评论。这样的转发功能，更加有利于消息传播，有了一种思想接力赛的意味。

如果转发评论不能自动显示在用户的消息后面，就不能提醒用户及时查看转发评论。在微博中，在“@我的”分页，将用户关注的人的转发与陌生人的转发分开显示，更加便于查看用户关注的转发评论，从而实现有效的交流。

#### 4. 插入图片

微博的图片功能就是增加了文字内容外的各种链接的个人图片库，使用这个功能，用户可以写图片日记，也可以发布即时见闻，微博的图片功能支持彩信。

#### 5. 视频和音乐

为了提供多媒体支持，微博系统应该能够通过输入链接直接发送视频、音乐，可以直接点击观看与收听。

#### 6. 插入话题

使用“#话题#”的格式，用户可以自定义话题，这就相当于添加了一个链接入口，点击后能够进入到话题页面，在站内也能够使用关键词搜索到该话题。大多数微博都有着过于繁琐的话题页面，包括：原创的、转发的、含图片的，以及含链接的等等。实际上微博系统可以将话题页面简化为包括：我的、我关注的人的、陌生人的三类。实际上，话题实现了群组及标签的处理，如果用户善于利用添加话题，就能够实现个人信息的分类整理，并能够使用话题标签来查阅自己的消息。

#### 7. 私信

双向关注的用户可以互相发送私信，而对于单向关注，被关注者可以给自己的粉丝发送私信，但不能反向发送，避免了“垃圾”信息。按照用户名将每个用户的私信对话分开显示，就像 MSN 和 QQ 聊天记录一样，使得对于私信的阅读和查阅一目了然。

#### 8. 短链接

由于微博系统中对于每条微博的字数限制，如果链接太长的话，那么微博内

容的字数就会大大减少。短链接的主要目标就是把较长的原始链接压缩为长度很短的地址。当用户点击这个短链接的时候，系统会帮助他跳转到较长的原始地址。

### 9. 敏感词

微博成为了一个能够让人们畅所欲言的场所，然而一些涉暴、涉枪、涉黄等内容也出现在了微博上。微博应能够监控不健康不安全的敏感词，对于用户提交的文字内容进行敏感词过滤，如果其中出现了敏感词就直接删除。

### 10. 搜索

在搜索用户消息时，可以生成信息链接以便直接点击，同样为了方便调用自己的信息，用户也可以搜索自己的消息。

### 11. 收藏

“我的收藏”应该仅对用户自己可见，这是因为用户收藏的消息仅仅是对他自己有用。但是“我的转发”是对外公开的，可以为其设置单独的菜单入口。

### 12. 推荐

用户推荐是指“推荐你喜欢的”功能，用户可以推荐与自己趣味很接近的人，也可以推荐人气用户。如果以用户关注的对象、发送消息的内容等数据为依据，则可以进行更有效的推荐。

### 13. 分页

使用分页功能，用户可以清晰地看到自己以前的消息，使得查阅消息更加方便。

### 14. 备份

对于用户而言，无论是有意义的消息，还是聊天灌水，只要是用心记录的数据都是宝贵的。为了对用户负责，微博系统应该提供备份功能。备份数据包括自己发布的消息、所有的评论、个人收藏，以及私信等等。

## 3.2 系统的业务流程分析

在基于 node js 的微博系统中，将用户分为普通用户和管理员用户两种类型。系统根据用户的类型为其提供相应的功能，即系统为普通用户和管理员用户提供的功能不同。因此系统运行的第一项任务就是在用户登录系统时进行身份认证。

身份认证要经过两个步骤完成，首先，判断要登录的用户是否为系统的有效用户？这决定了要登录的用户是否能够进入微博系统，因为只有系统的有效用户才能进入微博系统。然后，对于系统的有效用户，判断其类型是普通用户，还是管理员用户？系统对于不同类型的用户给予不同的操作权限。本系统的业务流程如图 3-1 所示。



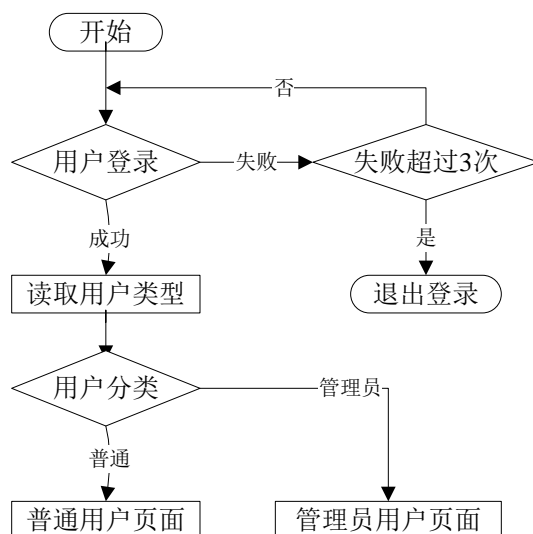


图 3-1 系统业务流程图

系统为每个要登录的用户提供 3 次身份认证的机会。如果要登录的用户 3 次输入的用户名和密码都不能与数据库中存储的用户信息相匹配，就强行退出登录页面。

系统的有效用户在通过了身份认证进入系统后，按照用户的类型，管理员用户和普通用户拥有不同的操作权限，管理员用户拥有能够使用所有功能的一类权限，而普通用户拥有只能使用部分功能的二类权限。

### 3.3 本章小结

本章主要进行了微博系统的需求分析，首先阐述系统的功能需求，然后对于系统进行了业务流程分析。本微博系统主要具有关注、评论、转发、插入图片、插入话题、私信、编辑、视频和音乐、搜索、分页、收藏、推荐、备份等功能。系统将用户分为普通用户和管理员用户两类，要登录系统的用户只有通过身份认证后才能进入系统。

## 第四章 微博系统的设计

### 4.1 系统的设计原则

#### 1. 可用性

微博系统应该能够控制图片的大小和页面大小，使用户快速地得到所需要的信息，不能因为下载时间过长而影响用户体验。系统中页面的使用风格应具有一致性，对于用户可能出错的地方应有预先的分析与防范。系统应提供数字快捷键，用户无需使用滚动菜单，而只需按下菜单所对应的数字键就可以直接进入该菜单项，从而改善用户体验。

#### 2. 开放性

考虑到将来微博系统扩充功能以及提高性能的需要，在设备选择以及联网方案方面应坚持开放性原则，使得微博系统能够实现各种硬件设备的互连互通。微博系统应采用标准接口，在软件方面应支持跨平台和开放数据接口，以便与其它的系统软件集成。微博系统应提供终端适配的能力，可以识别不同的终端并根据各终端的不同能力属性来推送适配的页面。

#### 3. 可扩展性

微博系统的构建不仅要着眼现在，而且更要放眼未来，因此微博系统应具备良好的可扩展性，系统构建不仅要满足当前需求，而且更要具有向未来平滑过渡的能力。例如，当信息量增加、网络规模扩大时，能够方便地对于服务器以及其他设备进行升级服务，从而满足日益增长的业务需求。

### 4.2 系统的架构设计

本微博系统部署于服务器上，为了保证数据的独立性与安全性，将数据与逻辑进行分离，用户通过 Internet 使用微博系统。因此本微博系统的架构如图 4-1 所示。



图 4-1 系统的架构图

在体系结构方面，本微博系统采用浏览器/服务器(Browser/Server, B/S)结构。B/S 结构是三层客户机/服务器(Client/Server, C/S)体系结构的一种实现方式，主要包括：浏览器、WEB 服务器和数据库服务器<sup>[32]</sup>。

与三层 C/S 结构的解决方案相比，B/S 体系结构在客户机上采用了 WWW 浏览器，将 WEB 服务器作为应用服务器。B/S 体系结构的核心是 WEB 服务器，数据请求、网页生成、数据库访问和应用程序执行全部由 WEB 服务器来完成，可以将应用程序以网页的形式存放在 WEB 服务器上。当用户运行某个应用程序时，只需要在客户端浏览器的地址栏中输入相应的 URL(Uniform Resource Locator)，客户端就会向 WEB 服务器发送 HTTP 请求，WEB 服务器在接收到来自客户端的 HTTP 请求后，就去调用相应的应用程序，并向数据库服务器发送数据操作请求，而数据库服务器在接收到来自 WEB 服务器的数据操作请求后，就会对数据操作请求进行响应，去执行数据操作任务，将结果返回给 WEB 服务器的应用程序，WEB 服务器应用程序执行业务处理逻辑，利用 HTML 来封装操作结果，将结果返回给客户端，通过浏览器呈现给用户。

本 B/S 结构的微博系统包括：表示层、业务逻辑层和数据层。其中数据层使用数据库管理系统实现对于数据的读写，它必须能够迅速地执行大量数据的更新和检索操作。业务逻辑层相当于三层 C/S 结构的功能层，负责处理所有的业务逻辑。表示层作为用户和系统之间的接口，负责从终端输入设备读取数据，以及将输出信息显示在终端输出设备上，完成用户与系统之间的对话任务。

本微博系统的整体架构如图 4-2 所示。

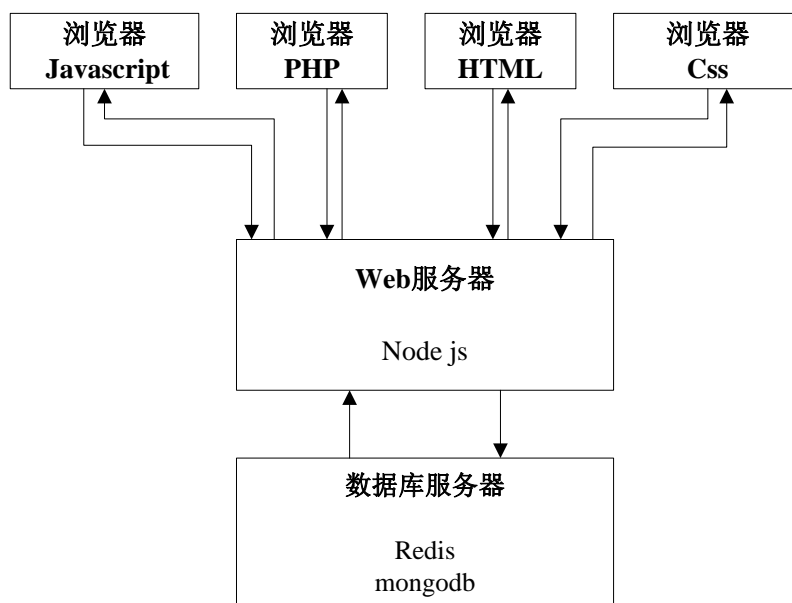


图 4-2 系统架构设计图

基于 node js 的微博系统的前台的逻辑和页面效果主要采用 PHP+HTML+CSS+JavaScript 来实现。PHP 完成动态页面，HTML+CSS 完成页面的排版，JavaScript 主要用于对前端数据的验证，采用 Ajax 技术来保证用户交互的及时响应，这样就能够达到良好的用户交互效果。

系统的后台实现采用 node js 框架。node js 是一个新的 Web Server 解决方案，它已经成为支持 Javascript 在服务器端运行的有效平台。node js 在不新增额外线程的情况下仍然能够对任务进行并行处理，node js 使用 Module 模块划分不同的功能，并采用事件驱动机制、异步编程风格，使得 node js 具有相当高的性能。

数据库采用 MongoDB 和 Redis 来实现。这两个 NoSQL 数据库能够满足数据存储的水平伸缩性上的需求，在处理大流量的数据访问时，比传统的关系型数据库更有优势。特别是，基于 node js 的微博系统使用分布式缓存系统 Redis。Redis 采用将数据和对象缓存在内存中的方式，减少了对于数据库读取的次数，进而减轻了数据库的负载、提高了系统的响应速度。

基于 node js 的微博系统的前台通过 HTTP 请求，从后台或缓存中获取数据。后台通过 node js 处理前台发送过来的 HTTP 请求，并将处理所得的数据存储在 NoSQL 数据库中。

对于备份数据库问题的解决，MongoDB 可以使用“sharding + repliaset”的方式，而 Redis 可以使用传统的“master/slave”方式进行及时备份。

### 4.3 系统的功能结构设计

微博系统的用户主要包括两类：一类是普通用户，另一类是管理员用户，为这两类用户分配的功能有所不同。微博系统的功能结构如图 4-3 所示。

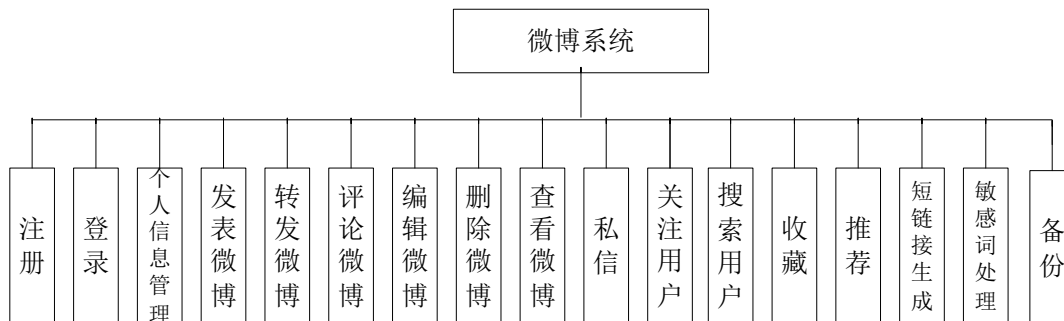


图 4-3 系统的功能结构图

用户的注册、登录、个人信息管理等作为微博系统的通用模块，只是普通用户与管理员用户登录成功后导向的页面是不同的。

发表微博、转发微博、评论微博，既是微博系统的常用功能，也是微博系统的核心功能，可以将这些功能放在一起构成一个核心功能模块。

编辑、删除、查看微博功能是对数据库中的数据进行操作，查看要将搜索结果进行分页显示，删除需要用户进行确认操作。

私信功能就是一个类似于留言板的功能。

搜索用户、关注用户、收藏、推荐等功能中，搜索可以采用模糊搜索的方式从数据库中读取出数据列表并将结果返回到结果页面，关注、收藏、推荐等功能可以通过在数据库中相关字段添加数据来实现。

在用户提交信息时，系统进行短链接生成和敏感词处理。

备份的内容可以包括自己的消息、所有的评论、个人收藏和私信等。如果担心频繁的备份会对服务器造成负担，可以限定每天备份的次数。

### 4.4 普通用户的用例

普通用户是微博系统的主要用户。对于普通用户组，系统为其提供的主要功能包括：注册、登录等权限及信息管理，微博的发布、删除、转发和评论，图片、视频支持，用户推荐、话题推荐，搜索用户、互相关注功能，微博内容的定时刷新等。

普通用户的用例图如图 4-4 所示。

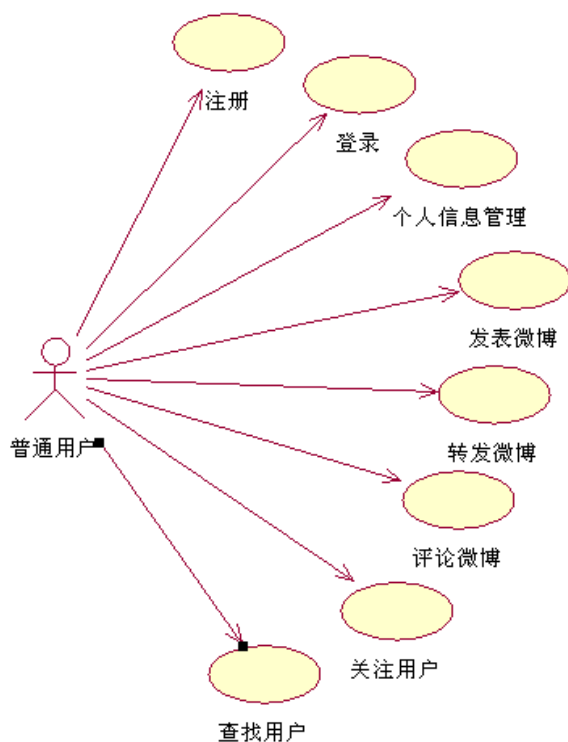


图 4-4 普通用户的用例图

#### 4.5 管理员用户的用例

对于管理员用户组，系统为其提供的主要功能包括：登陆、注册等权限及信息管理，查看用户发布的微博消息，删除微博消息，搜索指定用户。

管理员用户用例图如图 4-5 所示。

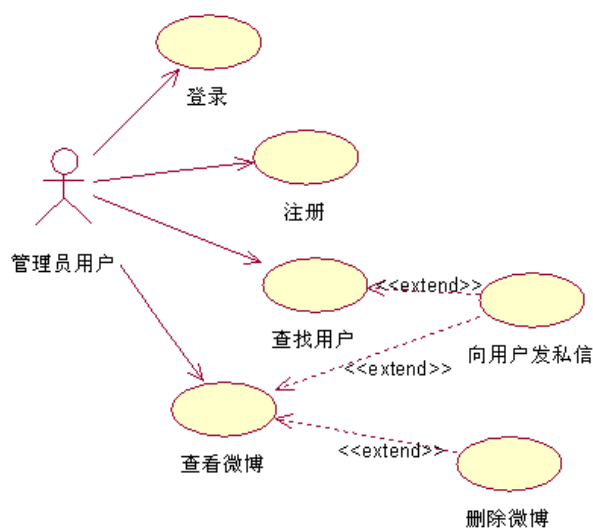


图 4-5 管理员用户用例图

## 4.6 微博系统的运行环境

基于 node.js 的微博系统的运行环境为：

服务器配置包括：CPU 采用 Intel i3 2.6GHz，内存为 4G，硬盘容量是 80G，并装配操作系统 Ubuntu 10.04 LTS、node.js 0.6.9、MongoDB 2.0.6、Redis 2.4.5、PHP 5.4 和 Web 服务器软件 Apache 2.2。

客户端配置包括：CPU 为 Intel T5870 2.0G Hz，内存是 2G，硬盘容量为 250G，装配 Windows 7 或 Windows XP 操作系统，Chrome、IE 或 Firefox 13 浏览器。

网络带宽为 1000Mb。

## 4.7 本章小结

本章主要对于微博系统进行了设计，本微博系统的设计遵循了开放性、可扩展性，以及可用性的基本原则，系统的体系结构采用 B/S 风格，将系统分为数据层、业务逻辑层和表示层，实现了数据访问与业务逻辑的分离，使得页面更具动态性。本章还给出了普通用户和管理员用户的用例，以及微博系统的运行环境。

## 第五章 微博系统的实现

### 5.1 系统的实现策略

基于 node js 的微博系统采用了 B/S 体系结构风格, 为了降低数据层、业务逻辑层和表现层之间耦合度, 系统采用模型-视图-控制器(Model-View-Controller, MVC)模式。View 层使用常见的 HTML+PHP+JavaScript 来实现, Model 层使用 MongoDB 和 Redis 数据库, 以 NoSQL 数据库驱动, 符合微博系统的特点, Controller 层使用 node js 框架实现控制, 完成 View 层和 Model 层的连接, MVC 各层之间的沟通使用 HTTP 连接方式。

基于 node js 的微博系统的 MVC 结构如图 5-1 所示。

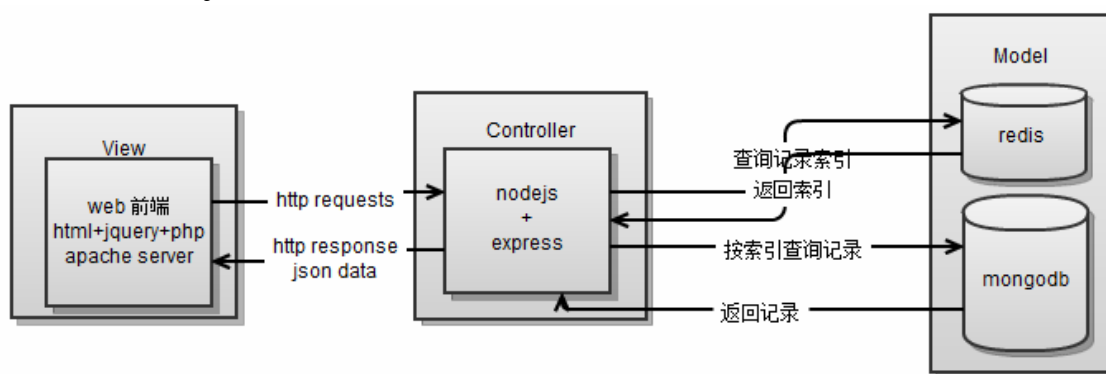


图 5-1 系统的 MVC 结构图

对于后台的开发, 将 node js 框架作为服务器的运行环境。由于 JavaScript 语言对回调函数的支持十分强大, 所以在处理异步的操作方面具有非常高的性能, 而且 JavaScript 能够支持 BSON 格式的数据, 所以大大简化了对于数据的处理。

在数据库的构建方面, 采用 MongoDB+Redis 的方式来实现数据的存储与动态更新。Redis 数据主要存放在内存中, 它的响应速度比 MongoDB 高得多, 但是由于内存空间的限制, 不能将数据都存放在 Redis 中, 因此使用 MongoDB+Redis 的方式。为了确保 Redis 中的数据量不会过大, 不会因为内存资源耗尽而导致系统性能下降, 在 Redis 中只存放数据的 id 做为索引项, 根据 id 到 MongoDB 中去查询具体的记录, 而且 Redis 中只存放较新的数据, 对于较老数据的查询需要到 MongoDB 中进行。实际上, 对于一个微博系统来说, 数据的时效性较高, 用户对老数据的查询需求并不是很多, 因此直接到 MongoDB 中去查询老数据, 牺牲一些响应时间, 用户也是可以接受的。

前台采用 HTML+PHP+JavaScript 的方式来实现, 用户成功登录微博系统后, 系统通过 HTTP 请求去访问基于 node js 的 Server, Server 收到请求后, 根据请求



信息进行数据库操作，首先从 Redis 中获取消息或者用户信息等的 id，然后再去 MongoDB 中查询其的详细内容，最后通过 HTTP 将查询结果返回给用户使用。

## 5.2 核心功能模块的实现方案

### 5.2.1 用户管理

用户信息使用 MongoDB 进行存储。在 MongoDB 中建立一个名为“weibo-user”的数据库用来存储用户信息。数据库“weibo-user”中包含两个 collection，分别是用以存储普通用户数据的“user”和存储管理员用户数据的“admin”。

对于普通用户，用户信息的数据结构定义为：

```
{
  id:1,
  username:'user1',
  password:'xxxxx',
  location:'beijing',
  sex:'m',
  qq:'1234567',
  follows:[1,2,3,4],
  fans:[4,5,6,7]
}
```

其中 id 为用户的唯一标识，是系统自动生成添加到数据库中的。username 和 password 分别为用户名和密码，它们都是必填字段。follows 和 fans 为所关注用户 id 的列表和粉丝 id 的列表。location 是位置信息，sex 是性别信息，qq 是 qq 号码，它们都是选填字段，只有在用户输入信息或者更新信息时才会填入这些信息。

由于 NoSQL 数据库对数据格式的要求不是很严格，所以，对于 follows 和 fans 可以采用数组的处理方式，这是传统的关系数据库所不能提供的便利。

对于管理员用户，用户信息的数据结构定义为：

```
{
  id:1,
  username:'admin',
  password:'xxxxx'
}
```

管理员用户的信息比较简单，只存储了用户的唯一标识、用户名和密码等信

息。

为了防止出现类似 csdn 用户名和密码泄露的悲剧,密码不使用明文进行存储,而是使用 MD5 加密算法将用户密码加密后,把密码的密文存储在数据库中。而且为了提高安全性,在用户登录的时候,可以采用 HTTPS 的方式将用户名和密码传输到服务器上。在用户登录时,首先查看 Redis 中是否存在该用户记录。如果 Redis 中存在该用户记录,则返回用户信息,并将用户记录的 expire 时间重新设置为 604800,也就是七天。如果 Redis 中不存在该用户记录,则到 MongoDB 中去查询。用户成功登录后,在 Redis 中就会添加该用户信息,并将用户记录的 expire 时间设置为七天。如果一个用户超过 7 天没有登录的话,就从 Redis 中删掉其记录,以减少内存的占用。这样做主要是为了减小用户记录所占用的数据库空间。

用户登录的流程图如图 5-2 所示。

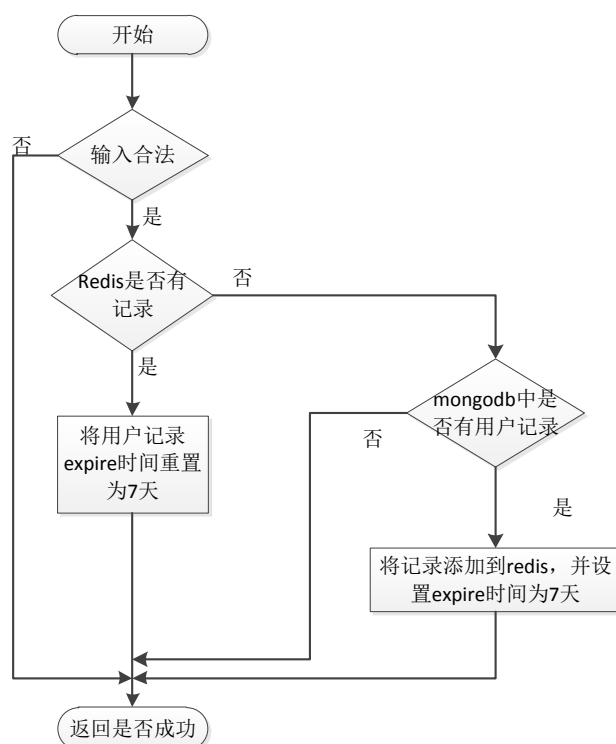


图 5-2 用户登录的流程图

用户登录的主要代码为:

```

<?php
$mongo = new Mongo();
$db = $mongo->selectDB("weibo-user");
?>
<form action="index.php" method="POST">

```

Name:

```
<input type="text" id="user_name" name="user_name" />
```

Password:

```
<input type="password" id="user_password" name="user_password" />
```

```
<input name="submitForm" id="submitForm" type="submit" value="Login" />
```

```
</form>
```

```
<?php
```

```
$succss = "";
```

```
if(isset($_POST) and $_POST['submitForm'] == "Login" ){
```

```
$user_name= _escape_string($_POST['user_name']);
```

```
$user_password = _escape_string($_POST['user_password']);
```

```
$error = array();
```

```
if(empty($user_name) or !filter_var($user_name,FILTER_SANITIZE_NAME)){
```

```
$error[] = "用户名空或无效;
```

```
}
```

```
if(empty($user_password)){
```

```
$error[] = "输入密码";
```

```
}
```

```
if(count($error) == 0){
```

```
$con = new Mongo();
```

```
if($con){
```

```
$db = $con->weibo-user;
```

```
$user = $db->user;
```

```
$qry = array("user" => $user_name,"password" => md5($user_password));
```

```
$result = $user->findOne($qry);
```

```
if($result){
```

```
$success = "登陆成功";
```

```
}
```

```
} else {
```

```
die("Mongo DB not installed");
```

```
}
```

```
}
```

```
}
```

?>

### 5.2.2 微博管理

在用户成功登陆系统后,系统会为每个用户在 Redis 中生成两个 set 和两个 list。其中一个 set 用来保存该用户关注的人的 id, 另一个 set 用来保存该用户的粉丝的 id; 一个 list 用来保存该用户关注的人发布的微博的 id, 另一个 list 用来保存该用户收到的新消息, 如评论、私信、通知等。微博内容存储在 MongoDB 中, 并通过微博的 id 对微博内容进行索引。

用于存储用户关注的人的 id 的 set 的命名规则为:

"weibo:follows:uid"

其中, uid 为用户 id。

用于存储用户粉丝的 id 的 set 的命名规则为:

"weibo:fans:uid"

其中, uid 为用户 id。

用于存储用户关注的人发布的新微博的 id 的 list 的命名规则为:

"weibo:posts:uid"

其中, uid 为用户 id。

每当用户发布一条微博, 系统就会生成一个微博 id, 这个微博 id 是由 Redis 进行维护的。在 Redis 中, 设置一个 key 用来记录系统中最新发布的微博的 id, 记作"weibo:maxid"。如果有某个用户发布一条微博, 首先"weibo:maxid"加 1 得到该条微博的 id, 将微博记录插入到 MongoDB 中, 然后将该微博的 id 分别添加入该用户所有的粉丝用户及其自己的"weibo:posts:uid"list 中。一旦其粉丝成功登录微博, 系统就会从他的"weibo:posts:uid"list 中读取最新的微博 id, 再根据这个微博 id 到 MongoDB 中读取微博内容进行显示。

在 MongoDB 中, 建立一个名为"weibo-posts"的数据库, 这个数据库有三个 collection: 名为 posts 的 collection 用来存储微博内容, 名为 comments 的 collection 用来存储评论内容, 名为 message 的 collection 用来存储私信等内容。

posts 的数据格式为:

```
{
  id:1,
  user_id:1,
  contents:'this is the content of the posts',
  time:1332350422483
```

```
}
```

其中，time 是 JS 机时间。

对于 comments 和 message 这两个 collection 的描述见“评论和私信”的实现方案。

用户发布微博的流程如图 5-3 所示。

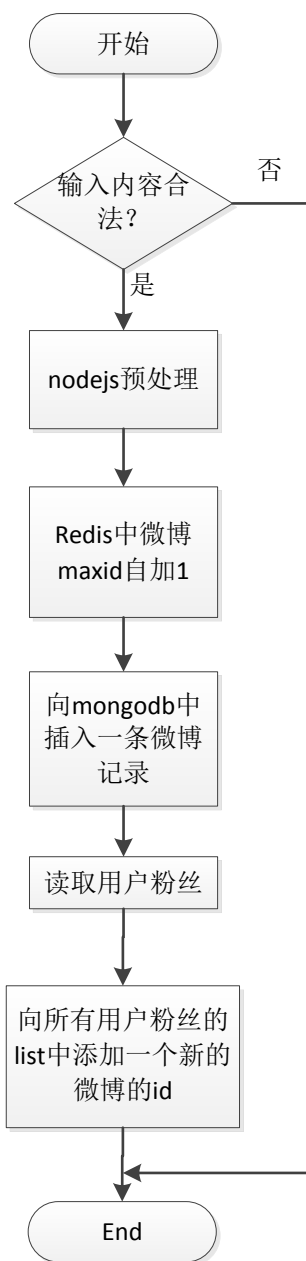


图 5-3 用户发布微博的流程图

其中数据的分发方式如图 5-4 所示。

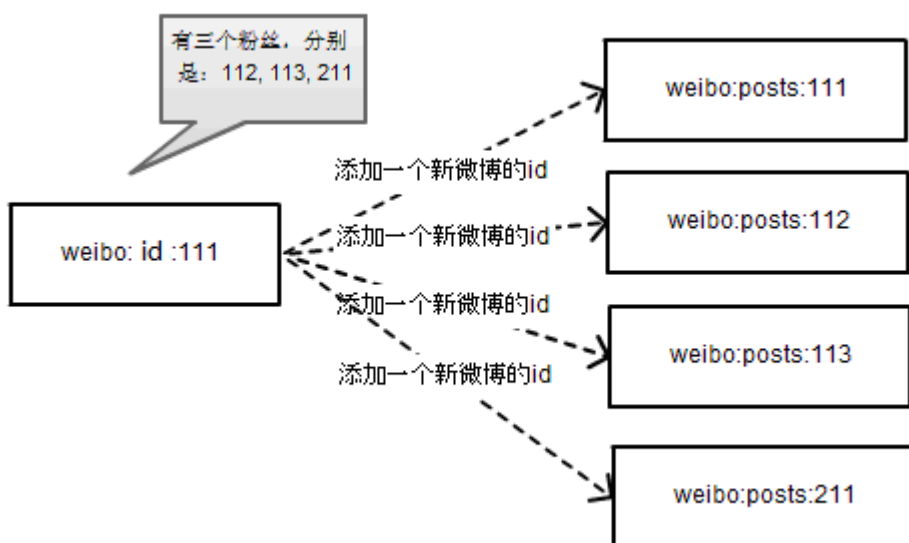


图 5-4 发布微博时的数据分发方式

发布微博功能的主要关键代码如下：

```

function Microblog(options) {
  this.config = {
    maxNum: 140,
    targetElem: .f-text,
    maxNumElem: .maxNum,
    sendBtn: #sendBtn,
    face: #face,
    activeCls: active,
    currentCls: current,
    inputID: #userName,
    textareaId: #conBox,
    list: #list-msg,
    callback: null
  };
  this.cache = {};
  this.init(options);
}
Microblog.prototype = {
  constructor: Microblog,
  init: function(options) {

```

```
this.config = $.extend(this.config,options || {});
var self = this,
    _config = self.config,
    _cache = self.cache;
$(_config.targetElem).each(function(index,item){
    $(item).unbind(focus);
    $(item).bind(focus,function(e){
        !$(this).hasClass(_config.activeCls) && $(this).addClass(_config.activeCls);
    });
    $(item).unbind(blur);
    $(item).bind(blur,function(e){
        $(this).hasClass(_config.activeCls) && $(this).removeClass(_config.activeCls);
    });
});
var faceImg = $(img,$(_config.face));
$(faceImg).each(function(index,item){
    $(item).unbind(click);
    $(item).bind(click,function(e){
        $(this).addClass(_config.currentCls).siblings().removeClass(_config.currentCls);
    });
});
$(_config.sendBtn).hover(function(){
    !$(this).hasClass(hover) && $(this).addClass(hover);
}
function(){
    $(this).hasClass(hover) && $(this).removeClass(hover);
});
self._bindEnv();
}
_countCharacters: function(str) {
    var totalCount = 0;
    for (var i=0; i<str.length; i++) {
        var c = str.charCodeAt(i);
```

```
if((c >= 0x0001 && c <= 0x007e) || (0xff60<=c && c<=0xff9f)) {
    totalCount++;
} else {
    totalCount+=2;
}
}
return totalCount;
}

_bindEnv: function() {
var self = this,
_config = self.config,
_cache = self.cache;
self._keyUp();
self._clickBtn();
}

_keyUp: function() {
var self = this,
_config = self.config,
_cache = self.cache;
$(_config.textareaId).unbind(keyup);
$(_config.textareaId).bind(keyup,function(){
var len = self._countCharacters($(this).val()),
html;
if(_config.maxNum * 1 >= len * 1) {
html = _config.maxNum * 1 - len * 1;
} else {
html = _config.maxNum * 1 - len * 1;
}
$(_config.maxNumElem).html(html);
$(_config.maxNumElem).attr(data-html,html);
});
}

_clickBtn: function() {
```



```
var self = this,
    _config = self.config,
    _cache = self.cache;
var reg = /^\\s*$/g;
$(_config.sendBtn).unbind(click);
(_config.sendBtn).bind(click,function(){
var inputVal = $(_config.inputID).val(),
textVal = $(_config.textareaId).val(),
maxNum = $(_config.maxNumElem).attr(data-html);
if(reg.test(inputVal)) {
alert(输入姓名);
return;
}
else if(reg.test(textVal)) {
alert("说点什么吧!");
return;
}
if(maxNum * 1 < 0) {
alert(字数超过限制，请缩减字数);
return;
}
self._renderHTML(inputVal,textVal);
});
}
_renderHTML: function(inputVal,textVal) {
var self = this,
    _config = self.config,
    _cache = self.cache;
var oLi = document.createElement("li"),
oDate = new Date();
oLi.innerHTML = <div class="userPic"> +
+
</div> +
```

```

<div class="content"> +
<div class="userName"><a href="javascript:;">+inputVal+</a>:</div> +
<div class="msgInfo">+textVal+</div> +
<div class="times">+
  <span>+self._format(oDate.getMonth() + 1) + "\u6708" + self._format
(oDate.getDate()) + "\u65e5 " + self._format(oDate.getHours()) + ":" + self._forma
t(oDate.getMinutes())+</span>+
  <a class="del hidden" href="javascript:;">删除</a>+
</div> +
</div>;
if($_config.list + " li").length > 0) {
  $(oLi).insertBefore($_config.list + " li")[0]);
  self._animate(oLi);
} else {
  $_config.list).append(oLi);
  self._animate(oLi); }
_config.callback && $.isFunction(_config.callback) && _config.callback();
self._clearVal();
self._hover();
}
_format: function(str){
return str.toString().replace(/^(\\d)$/, "0$1");
}
_getSrc: function() {
var self = this,
_config = self.config,
_cache = self.cache;
var faceImg = $(img,$(_config.face));
for(var i = 0; i < faceImg.length; i++) {
if($(faceImg[i]).hasClass(_config.currentCls)) {
return $(faceImg[i]).attr(src);
break;
}
}

```

```

    }
  }
  _clearVal: function() {
    var self = this,
    _config = self.config,
    _cache = self.cache;
    $(_config.inputID) && $(_config.inputID).val();
    $(_config.textareaId) && $(_config.textareaId).val();
  },
  _hover: function() {
    var self = this,
    config = self.config,
    cache = self.cache;
    $(_config.list + li).hover(function(){
      !$(this).hasClass(hover)                                &&
$(this).addClass(hover).siblings().removeClass(hover);
      $(.del,$(this)).hasClass(hidden) && $(.del,$(this)).removeClass(hidden);
      var $that = $(this);
      $(.del,$that).unbind(click);
      $(.del,$that).bind(click,function(){
        $($that).animate({
          opacity : 0
        },500,function(){
          $that.remove();
        });
      });
    }
  }
  function(){
    $(this).hasClass(hover) && $(this).removeClass(hover);
    !$(.del,$(this)).hasClass(hidden) && $(.del,$(this)).addClass(hidden);
  });
  _animate: function(oLi) {
    var self = this;

```

```

var iHeight = $(oLi).height(),
    alphah = 0,
    timer,
    ount = 0;
$(oLi).css({ "opacity" : "0", "height" : "0" });
timer && clearInterval(timer);
timer = setInterval(function () {
    $(oLi).css({ "display" : "block", "opacity" : "0", "height" : (count += 8) + "px" });
    if (count > iHeight) {
        clearInterval(timer);
        $(oLi).css({ "height" : iHeight + "px" });
        timer = setInterval(function () {
            $(oLi).css({ "opacity" : alphah += 10 });
            alphah > 100 && (clearInterval(timer), $(oLi).css({ "opacity":100}));
        },30);
    }
},30);
};
$(function(){
    new Microblog({});
});

```

### 5.2.3 用户关系管理

由于采用了 MongoDB 数据库，关注和粉丝功能的处理就比较简单，只需在用户信息里添加两个属性为数组的字段就能够很好的实现。

如果用户 101 关注了用户 202，系统首先要在用户 101 的用户信息里的 follows 字段里添加一个新的用户 id—202，并在 202 用户的 fans 字段里添加一个新的用户 id—101，然后在 Redis 中更新 101 用户的关注 set: “weibo:follows:101”，并更新 202 用户的粉丝 set: “weibo:fans:202”，分别在其中添加新元素。

用户关注的流程如图 5-5 所示。

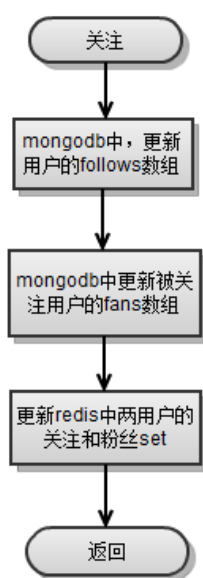


图 5-5 用户关注流程图

#### 5.2.4 评论和私信

如果用户对某条微博进行评论，系统就会首先生成一条消息的 id，并将评论内容存储到 MongoDB 中的名为 comments 的 collection 中，然后，向用于存放发表该条微博的用户的保存消息的 list(即"weibo:msg:uid")中推入一条记录，这条记录中包含评论的 id 和格式，指定它是一条评论。

如果用户对另一个用户发送私信，系统也会首先生成一条私信的 id，并将私信内容存储到 MongoDB 的名为 message 的 collection 中，然后，向目标用户的保存消息的 list(即"weibo:msg:uid")中推入一条记录，这条记录中包含私信的 id 和它的格式，指定它是私信。

用户在成功登录微博后，系统记录用户的登录，一旦"weibo:msg:uid"这个 list 中有新记录，服务器都会向用户推送一条消息来通知用户有新消息。这种服务器推送的方式的优点在于：第一，保证了消息的实时性，第二，保证了用户每次只收到最新的消息，第三，省去了用户对服务器的请求，从而有效地降低了系统压力。

"weibo:msg:uid"中存储的数据记录的格式定义为：

```
{
  id:1,
  type:'comment'
```

```
}
```

其中, id 为记录的 id。当消息为评论的时候, type 为'comment'; 当消息为私信的时候, type 为'msg'。

在 MongoDB 中存储的评论的数据记录的格式定义为:

```
{
  id:1,
  user_id:1,
  posts_id:1,
  contents:'this is the content of the comments',
  time:1332350422483
}
```

其中, user\_id 为发表评论的用户 id, posts\_id 为所评论的微博 id, time 为 JS 机时间。

评论功能的主要代码为:

```
<?php
require 'app.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  $id = $_POST['article_id'];
  $comment = array(
    'name' => $_POST['fName'],
    'comment' => $_POST['fComment'],
    'posted_at' => new MongoDate()
  );
  $status = $db->commentId($id, 'posts', $comment);
  if ($status == TRUE) {
    header('Location: single.php?id=' . $id);
  }
}
?>
```

在 MongoDB 中存储的私信的数据记录的格式定义为:

```
{
  id:1,
  user_id:1,
```

```

        receiver_id:2,
        contents:'this is the content of the msg',
        time:1332350422483
    }

```

其中 user\_id 为私信的发起者的 id, receiver\_id 为私信的接收者的 id, time 为 JS 机时间。

### 5.2.5 搜索用户和搜索微博

由于搜索微博内容的需求不是很明显, 所以本系统只提供搜索用户功能。对于搜索用户的结果, 可以对用户进行排名优化。各类用户的优先级定义为: 最先显示的是用户的已关注用户, 其次显示的是关注用户的粉丝, 然后是微博系统中的活跃用户, 最后是其它的普通用户。

本微博系统除了提供搜索用户功能外, 还提供搜索微博功能。搜索微博是指搜索指定用户微博, 其代码如下:

```

<?php
session_start();
include_once( 'config.php' );
include_once( 'saetv2.ex.class.php' );
function utf_substr($str,$len) {
    for($i=0;$i<$len;$i++){
        $temp_str=substr($str,0,1);
        if(ord($temp_str) > 127){
            $i++;
        }
        if($i<$len){
            $new_str[]=substr($str,0,3);
            $str=substr($str,3);
        }
    }
    else{
        $new_str[]=substr($str,0,1);
        $str=substr($str,1);
    }
}

```

```

return join($new_str);
}
$c=new SaeTClientV2(WB_AKEY,WB_SKEY, $_SESSION['token']
['access_token'] );
$returnstr="opt=query";
if($_POST['name']!=""){
$name=$_POST['name'];
}
else if($_GET['name']!=""){
name=$_GET['name'];
}
else{
$returnstr."&success=false&error=输入用户 ID 或用户名&option=show_user
_by_name";
echo $returnstr;
exit();
}
$name=iconv("gbk","utf-8",$name);
$list=$c->show_user_by_name($name);
if(is_array($list) && count($list)>0){
if($list['error']<>""){
$returnstr."&success=false&error=".$list['error']."&option=get_user_timeline";
}
else{
$uid=number_format($list['id'],0,"","");
$returnstr."&ID=".$list['id'],0,"","");
$returnstr."&username=".$list['screen_name'];
$returnstr."&location=".$list['location'];
$returnstr."&gender=".$list['gender'];
$returnstr."&statuses_count=".$list['statuses_count'],0,"","");
$returnstr."&followers_count=".$list['followers_count'],0,"","");
$returnstr."&friends_count=".$list['friends_count'],0,"","");
$returnstr."&success=true&option=show_user_by_name";

```



```

    $i=0;
    $list=$c->user_timeline_by_id($uid);
    if(is_array($list['statuses'])){
        $i=0;
        for each($list['statuses'] as $item){
            $returnstr."&ID".$i."&".sprintf("%.0f",$item['id'])."&text".$i."&".utf_substr($item
['text'],50);
            $i++;
        }
        $returnstr."&success=true&option=get_user_timeline";
    }
    else{
        $returnstr."&success=false&error=输入用户 ID 或用户名&option=get_user_
timeline";
    }
    echo $returnstr;
?>

```

### 5.2.6 视频分享和发布图片

如果分享一个视频,需要输入视频播放页的地址,目前系统只支持 ku6 与 weiku 两个网站的视频。

用户输入一个视频播放页的地址,系统首先根据 URL 来判断视频来源,接着根据预先设定的方式来抓取网页中 flash 播放器的地址,然后将 flash 播放器地址作为视频分享微博的主要内容存放到 MongoDB 中。在用户查看微博的时候,就将播放器嵌入到网页中去。

由于 node js 框架对网页 dom 树的处理没有 PHP 用起来简单,因此抓取网页中 flash 播放器的地址使用 PHP 来实现。抓取网页中 flash 播放器的地址的具体方法是:首先使用 PHP 获取一个网页的内容,然后使用 sample\_HTML\_dom 来分析 dom 树,从网页中抓取 flash 播放器的地址,最后将网页中 flash 播放器的地址存储到 MongoDB 中。

视频分享的流程如图 5-6 所示。

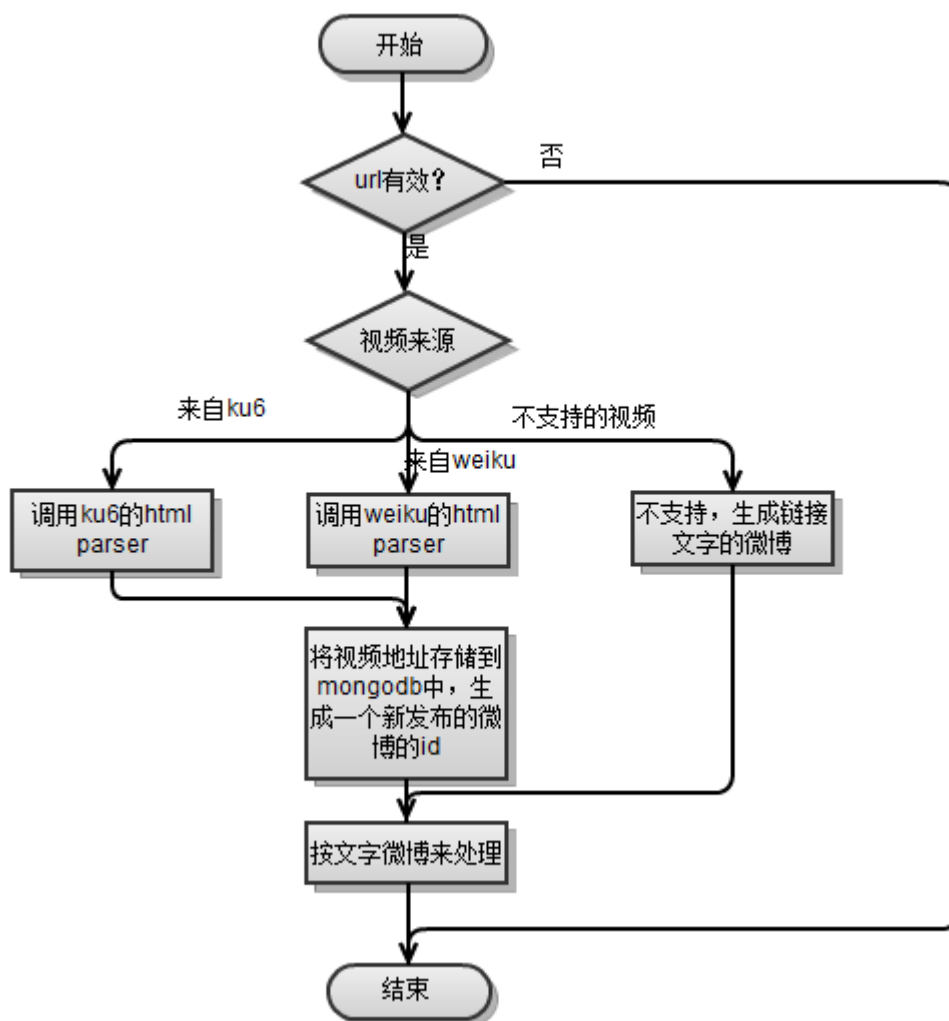


图 5-6 视频分享功能流程图

如果用户发布图片微博，可以上传本地图片，可以通过输入图片的 URL 来引用外站的图片。如果用户使用的是上传本地图片的方式，系统会将图片存储到 MongoDB 中。如果用户通过输入图片的 URL 来引用外站的图片，系统会只存储图片的 URL。

由于图片文件占用的空间较大，因此采用 MongoDB 的 gridfs 技术来存储图片文件，而不是按照传统的方式将图片文件存储在服务器的图片文件夹中。采用 gridfs 存储图片的优点是扩展性强，在服务器存储容量不足的时候，可以通过增加服务器的方式来解决。

图片管理的代码为：

```

<?php
function_construct($akey, $skey=""){
$this->akey=$akey;

```

```
$this->skey=$skey;
$this->base='http://api.t/statuses/upload.xml';
$this->curl=curl_init();
curl_setopt( $this->curl, CURLOPT_RETURNTRANSFER, true);
$this->postInit();
}
function postInit(){
$this->postdata=array('source='. $this->akey);
}
function setUser($name, $pass){
$this->user['oname']=$name;
$this->user['opass']=$pass;
$this->user['name']=$name;
$this->user['pass']=$pass;
curl_setopt( $this->curl, CURLOPT_USERPWD, "$name:$pass");
}
function public_timeline(){
return $this->call_method('statuses', 'public_timeline');
}
function friends_timeline(){
return $this->call_method('statuses', 'friends_timeline');
}
function user_timeline($name){
return
$this->call_method('statuses','user_timeline','?screen_name='.urlencode($name));
}
function mentions($count=10, $page=1){
return $this->call_method('statuses','mentions', '?count='. $count.'&page=', $page);
}
function comments_timeline($count=10, $page=1){
return $this->call_method('statuses','comments_timeline','?count=.$count.
&page=', $page);
}
```

```
function comments_by_me($count=0, $page=1){
    return $this->call_method('statuses','comments_by_me','?count='.$count. '&page=' ,
$page);
}
function comments($tid, $count=10, $page=1){
    if (is_float($tid)){
        $tid=number_format($tid, 0, "", "");
    }
    return $this->call_method('statuses', 'comments', 'id='.$tid.'&count='.$count.
'&page='.$page);
}
function counts($tids){
    if (is_float($tids)){
        $tid=number_format($tid, 0, "", "");
    }
    return $this->call_method('statuses', 'counts' , '?tids='.$tids);
}
function show($tid){
    if (is_float($tid)){
        $tid=number_format($tid, 0, "", "");
    }
    return $this->call_method('statuses', 'show/'.$tid);
}
function destroy($tid){
    if (is_float($tid)){
        $tid=number_format($tid, 0, "", "");
    }
    return $this->call_method('statuses', 'destroy/'.$tid);
}
function repost($tid,$status){
    if (is_float($tid)){
        $tid=number_format($tid, 0, "", "");
    }
}
```

```

$this->postdata[]='id'.'.$tid;
$this->postdata[]='status'.'.urlencode($status);
return $this->call_method('statuses','repost');
}
function update($status){
$this->postdata[]='status'.'.urlencode($status);
return $this->call_method('statuses', 'update');
}
function upload($status,$file){
$boundary=uniqid('-----');
$MPboundary='--'.$boundary;
$endMPboundary=$MPboundary. '--';
$multipartbody= "";
$multipartbody.=$MPboundary . "\r\n";
$multipartbody.='Content-Disposition:form-data;name="pic"; filename="wiki.jpg".
"\r\n";
$multipartbody.='Content-Type: image/jpg'. "\r\n\r\n";
$multipartbody.=$file. "\r\n";
$k="source";
$multipartbody.=$MPboundary . "\r\n";
$multipartbody.='content-disposition: form-data; name="'. $k. "\"\r\n\r\n";
$multipartbody.=$v. "\r\n";
$k="status";
$v=$status;
$multipartbody.=$MPboundary. "\r\n";
$multipartbody.='content-disposition: form-data; name="'. $k. "\"\r\n\r\n";
$multipartbody.=urlencode($v). "\r\n";
$multipartbody.="\r\n". $endMPboundary;
curl_setopt($this->curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($this->curl, CURLOPT_POST, 1 );
curl_setopt($this->curl, CURLOPT_POSTFIELDS, $multipartbody);
$url='http://api.t/statuses/upload.json' ;
curl_setopt($this->curl,CURLOPT_USERPWD,$this->user['oname'].":").

```

```
this->user['opass']);
    $header_array=array("Content-Type: multipart/form-data; boundary=$boundary" ,
"Expect: ");
    curl_setopt($this->curl,CURLOPT_HTTPHEADER,$header_array);
    curl_setopt($this->curl,CURLOPT_URL,$url);
    curl_setopt($this->curl,CURLOPT_HEADER,false);
    curl_setopt($this->curl,CURLINFO_HEADER_OUT,true);
    $info=curl_exec($this->curl);
    return json_decode($info, true);
}
function send_comment($tid,$comment,$cid=""){
    $tid=number_format($tid, 0, "", "");
}
if (is_float($cid)){
    $cid=number_format($cid, 0, "", "");
}
$this->postdata[]='id=' . $tid;
$this->postdata[]='comment='. urlencode($comment);
if($cid>0) $this->postdata[]='cid='.$cid;
return $this->call_method('statuses' , 'comment');
}
function reply($tid, $reply, $cid){
    if (is_float($tid)){
        $tid=number_format($tid, 0, "", "");
    }
    if (is_float($cid)){
        $cid=number_format($cid, 0, "", "");
    }
    $this->postdata[]='id=' . $tid;
    $this->postdata[]='comment='.urlencode($comment);
    if($cid>0)
    $this->postdata[]='cid='.$cid;
    return $this->call_method('statuses' , 'comment');
```

```

    }
    function remove_comment($cid )
    {
        if (is_float($cid)){
            $cid=number_format($cid, 0, "", "");
        }
        return $this->call_method('statuses' , 'comment_destroy/'.$cid);
    }
    function verify_credentials()
    {
        return $this->call_method('account' , 'verify_credentials');
    }
    function call_method($method , $action , $args=""){
        curl_setopt($this->curl, CURLOPT_POSTFIELDS , join('&', $this->postdata) );
        $url=$this->base.$method.'/'.$action.'.json'.$args ;
        curl_setopt($this->curl , CURLOPT_URL, $url );
        $ret=curl_exec( $this->curl );
        $this->postInit();
        return json_decode($ret, true);
    }
    function _destruct (){
        curl_close($this->curl);
    }
}

```

### 5.2.7 短链接生成

由于微博系统中对于每条微博的字数限制，如果链接太长的话，那么微博内容的字数就会大大减少。短链接的主要目标就是把较长的原始链接压缩为长度很短的地址。当用户点击这个短链接的时候，系统会帮助他跳转到较长的原始地址。

目前生成短链接的方法主要有三种：第一种方法是通过对原链接的 URL 进行转码，再压缩生成短链接，这种方法生成的短链接的长度不固定。第二种方法是通过对原链接使用 MD5 算法进行加密，得到一个 32 位的字符串，再对这个 32 位的字符串进行转码，这种方法能够保证短链接的长度固定为 6。第三种方法是直接

生成随机数，并通过遍历已经生成的短链接数据库来保证短链接不重复。

本系统采用第二种方法，算法思想为：

本系统使用 6 位字符串来表示短链接，其中每位字符来自 ASCII 字符表的一个子集  $A = \{a, b, c, \dots, z, 0, 1, 2, 3, 4, 5\}$ ，集合  $A$  中共计 32 个字符。这样 6 位字符串表示的短链接就有  $32^6 = 1073741824$  种。生成短链接的过程如下：

(1)对传入的长 URL 进行使用 MD5 算法进行加密，得到一个 32 位的字符串，这个字符串有  $16^{32}$  种，从而基本上可以保证唯一性。

(2)将上述 32 位的字符串分成 4 份，每份 8 个字符，这 8 个字符的字符串有  $16^8 = 4294967296$  种。

(3)将 8 位的字符串当作 16 进制整数，也就是  $1 * (0x.\$val)$ ，取其 0-30 位，平均分成 6 组，每 5 位 1 组。对于每一组，计算出他的整数值后映射到集合  $A$  中的某个字符上。这样就得到一个 6 位的短链接地址。

(4)在 Redis 中新建一个从短链接到原始链接的映射关系的数据库，命名为 "weibo:dns"，将短链接做为 key 而原链接做为 value 存储在 Redis 数据库中。

当用户访问短链接的时候，系统就从 Redis 中查找到原链接，然后跳转到原链接指向的地址。

短链接生成算法如下：

```
<?php
function shorturl($url="", $prefix="", $suffix='') {
    $base32 = array (
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
        'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
        'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
        'y', 'z', '0', '1', '2', '3', '4', '5' );
    $hex = md5($prefix.$url.$suffix);
    $hexLen = strlen($hex);
    $subHexLen = $hexLen / 8;
    $output = array();
    for ($i = 0; $i < $subHexLen; $i++) {
        $subHex = substr ($hex, $i * 8, 8);
        $int = 0x3FFFFFFF & (1 * ( '0x'.$subHex));
        $out = "";
        for ($j = 0; $j < 6; $j++) {
```



```

$val = 0x0000001F & $int;
$out .= $base32[$val];
$int = $int > 5; }
$output[] = $out; }
return $output; }

$urls = shorturl('http://www.jbxue.com');
var_dump($urls);

```

### 5.2.8 敏感词处理

对于敏感词，系统的前台和后台都需要进行处理，以免用户通过其它方式绕过前台页面直接进行信息的提交<sup>[33]</sup>。在用户提交微博、评论等信息的时候，要将用户提交的内容与敏感词库中的关键词进行比对，具体比对方法是：对于敏感词库中的每个关键词，查看它在用户提交的内容中是否存在，如果用户提交的内容中出现了敏感词库中的关键词，则禁止用户提交该内容，并给出提示和修改建议。由于用户提交的内容不超过 140 个字，而关键词库的长度一般也是固定的，因此比对的时间消耗不会影响系统性能。

敏感词的列表存储在 MongoDB 数据库中，管理员用户可以随时向敏感词数据库中添加新的关键词。在数据库中敏感词存储的数据类型为 set，这样系统就能够自动进行查重操作，如果有重复的关键词插入，数据库会返回错误，并在管理界面给出提醒。

敏感词过滤算法如下：

```

<?php
class FilterTools {
public static $keyword = array();
static function getBadWords($filename){
$file_handle = fopen($filename, "r");
while (!feof($file_handle)) {
$line = trim(fgets($file_handle));
array_push(self::$keyword,$line);
fclose($file_handle);
return self::$keyword; }
static function filterContent($content,$target,$filename,$memconfig){
$mem = new BadWordsMemcache($filename,$memconfig);

```

```
$keyword = $mem->getList();
if(count($keyword) == 0){
    $keyword = self::getBadWords($filename); }
return str($content, array_combine($keyword, array_fill(0, count($keyword),
$target )));

class BadWordsMemcache{
var $memcache;
var $key;
var $list;
var $filename;
function _construct($filename,$memconfig) {
    $this->filename = $filename;
    if(!class_exists("P_Memcache")){
        require_once DIR."lib/memcache.class.php";
        $this->key = "bad_words";
        $this->memcache = new P_Memcache();
        $this->memcache->config = $memconfig;
        $this->memcache->connect();
        print_r($this->memcache);
        $this->init(); }
    function __destruct() {
        $this->memcache->close();}
    function init($isReset = false){
        $this->list = $this->memcache->get($this->key)?$this->memcache->get($this->
key): array();
        if(count($this->list)==0 || $isReset){
            $this->list = filterTools::getBadWords($this->filename);
            $this->memcache->set($this->key, $this->list);
            $log_data = Log::formatData($this->list);
            Log::logWrite($log_data, 'bad.words','init');
            function getList(){
                return $this->list; } }
```

### 5.2.9 分页

为了使得用户更加方便地查阅消息，清晰地看到自己以前的消息，本微博系统提供了分页功能，分页功能能够通过 MongoDB 的游标功能来实现。

分页功能的主要代码为：

```
public function get($page,$collection){
    $currentPage = $page;
    $articlesPerPage = $this->limit;
    $skip = ($currentPage - 1) * $articlesPerPage;
    $table = $this->db->selectCollection($collection);
    $cursor = $table->find();
    $totalArticles = $cursor->count();
    $totalPages = (int) ceil($totalArticles / $articlesPerPage);
    $cursor->sort(array('saved_at' => -1))->skip($skip)->limit($articlesPerPage);
    $data=array($currentPage,$totalPages,$cursor);
    return $data;
}
```

### 5.2.10 备份

对于数据安全性要求比较高的系统，可以使用 Replica Set 方式来实现 MongoDB 数据库的备份。在上层进行 Sharding 操作使数据库完全独立出来，这样只要程序的接口保持不变，数据库就可以任意扩充。

使用 Replica Set 来进行数据的备份，可以防止出现单数据库时当机的情况。一旦主库出现故障，立刻就会有另一个库来充当主库以供系统使用，这就保证了数据的安全性。Sharding 操作的主要目的是不让程序去做路由，它会选择一个可用的数据库来接入数据，而接口程序只要与 Sharding 的 MongoDB 数据库相连，就会采用它所分配的最优方案使用数据库。

Replica Set + Sharding 构建 MongoDB 数据库的方式如图 5-7 所示。

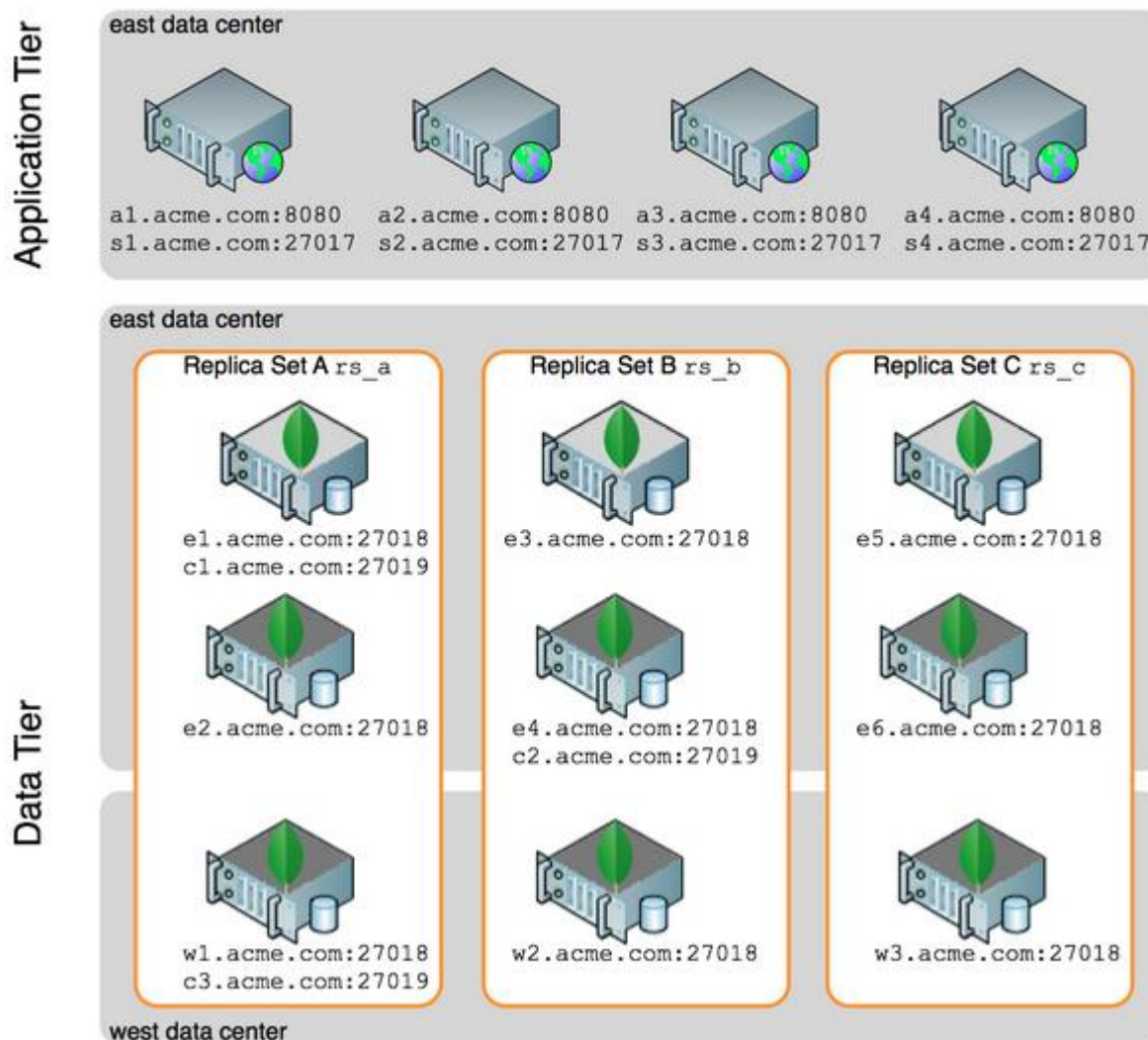


图 5-7 Replica Set + Sharding 方式构建 MongoDB 数据库

### 5.2.11 系统缓存

在微博系统中，需要缓存的主要是微博内容及图片<sup>[34]</sup>。因为微博的 id 部分存放在 Redis 数据库中，而微博的内容存放在 MongoDB 中，微博的图片也存放在 MongoDB 中。如果每次请求都需要进行数据库查询，那么在用户的数量达到一定值的时候，系统的效率就会显著下降。为了提高系统的效率，本系统采用 MemCached 缓存。

MemCached 是一个分布式的内存对象缓存系统，在动态 WEB 应用中用来减轻数据库的负担。它通过在内存中缓存数据和对象来减少对数据库的访问次数，从而提高动态的、数据库驱动的网站响应速度。

建立 MemCached 缓存的方法是：首先在服务器上启动 MemCached 进程以提供缓存服务，接着在 PHP 中进行简单的配置后，就去设置对微博内容及图片进行

缓存。这样在用户频繁登录的时候，就可以只在有新内容的时候访问数据库，有效地减少了数据库查询操作，从而减少系统响应时间。

## 5.3 系统实现

### 5.3.1 数据库连接池

由于本微博系统需要多次、大量地访问数据库，如果每次访问都建立一个新的连接，那么必定会产生巨大的系统开销。为了减少每次建立连接所引起的资源浪费，本系统建立了数据库连接池，每当需要连接数据库的时候，就从数据库连接池中取出一条未关闭的连接来使用。一旦数据库连接池中连接数太少而不够使用，就需要重新建立连接。如果连接数超出了数据库连接池的额定量，在数据库访问结束时，系统就会将多余的连接删除而不放入数据库连接池。

数据库连接池的数据操作流程如图 5-8 和图 5-9 所示。

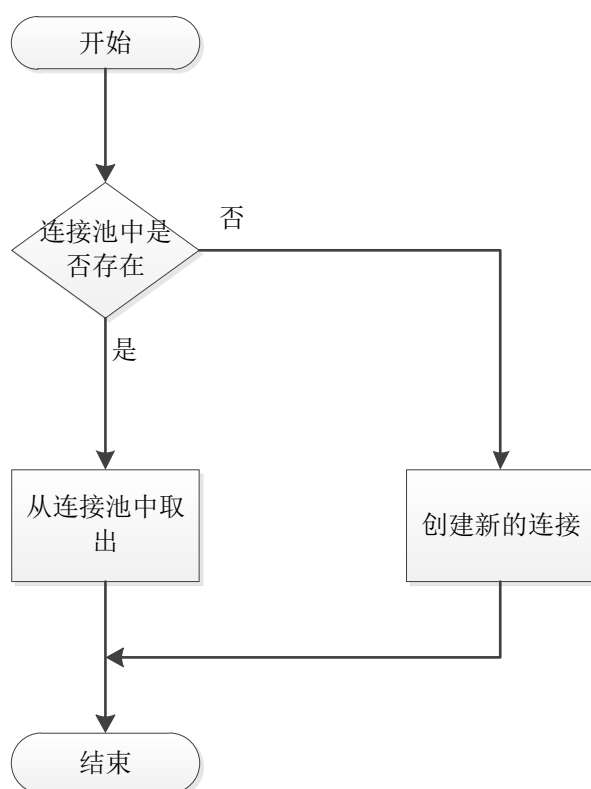


图 5-8 创建连接

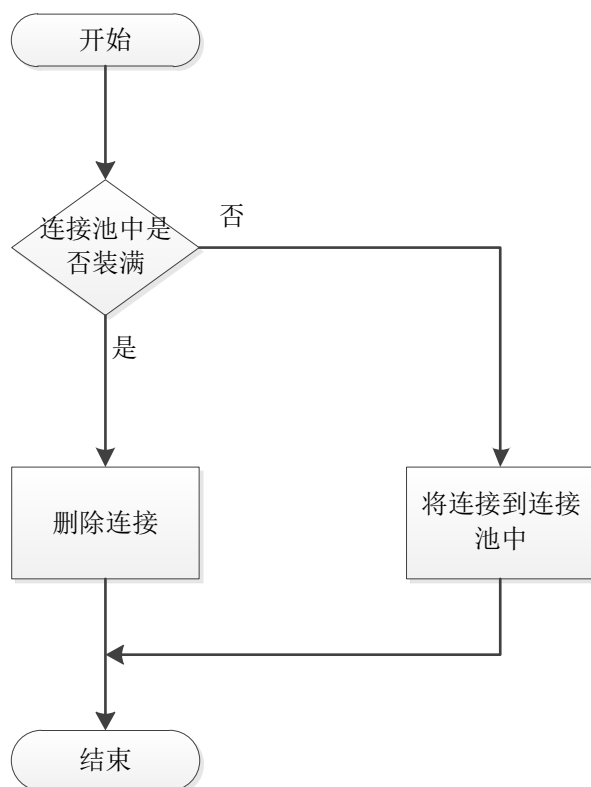


图 5-9 删除连接

### 5.3.2 后台接口的实现

基于 node js 的微博系统采用 MVC 模式，它的数据层、业务逻辑层和表现层三层可以完全分离。前台通过 HTTP 请求与后台进行数据的交互，也就是说，前台与后台进行数据的交互，无论是通过 Ajax 与 node js 进行异步交互，还是通过 curl 与 node js 进行交互，都要通过 HTTP 请求，因而后台只要提供足够完成任务的接口，就可以实现系统功能。

后台接口的设计如表 5-1 所示。

表 5-1 后台接口设计

序号	接口名称	方法	参数描述	接口功能
1	/opt/follow/:from_id/:to_id	Get	from_id:当前用户 id to_id:关注用户 id	关注某用户
2	/opt/del_follow/:from_id/:to_id	Get		取消关注
3	/opt/post	Post	Content:微博内容	发表微博
4	/opt/delete/:id	Get	Id:微博 id	删除微博
5	/opt/msg/:from_id/:to_id	Post	from_id:当前用户 id to_id:关注用户 id content:微博内容	发消息

6	/opt/del_msg/:id	Get	id: 消息 id	删除消息
7	/opt/comment/:user_id	Post	user_id:用户 id content:评论内容	评论
8	/opt/del_comment/:comment_id	Get	comment_id:评论 id	删除评论
9	/user/add	Post	Username:用户名 password:密码	注册新用户
10	/user/login	Post	Username:用户名 password:密码	用户登录
11	/user/logout/:user_id	Get	user_id: 用户 id	注销登录
12	/user/set_userinfo/:user_id	Post	user_id:用户 id object:用户信息	设置用户信息
13	/user/userinfo/:user_id	Get	user_id:用户 id	获取用户信息
14	/user/get_follow/:user_id	Get	user_id:用户 id	获取关注列表
15	/user/get_fans/:user_id	Get	user_id:用户 id	获取粉丝列表
16	/user/get_weibo/:user_id	Get	user_id:用户 id	获取发表过的微博
17	/user/at_me/weibo/:user_id	Get	user_id:用户 id	获取@我的微博
18	/user/at_me/comment/:user_id	Get	user_id:用户 id	获取@我的评论
19	/user/comment/comment_me/:user_id	Get	user_id:用户 id	获取评论我的评论
20	/user/comment/comment_other/:user_id	Get	user_id:用户 id	获取我对其它人的评论
21	/opt/upload_pic	Post	picfile: 图片文件 picname:图片名 user_id:上传图片的用户 id	上传图片
22	/opt/parsevideo/:URL	Get	URL 视频所在网页地址	解析视频接口
23	/opt/get_at_list/:user_id	Get	user_id:用户 id	当@时, 获取用户@列表
24	/opt/get_topics	Get	无	获取热门话题列表
25	/opt/get_new/:user_id	Get	user_id:用户 id	获取新的通知
26	/opt/URL/:URL	Get	URL:原链接地址	获取短链接
27	/usr/delete/:user_id	Get	user_id:用户 id	删除用户

### 5.3.3 前台的实现

#### 1. Ajax 请求的代理转发

本微博系统前台采用 PHP+HTML +JavaScript 的方式来完成。由于 JavaScript 通过 Ajax 请求与 node js Web Server 交互会产生跨域访问的问题, 因此需要使用

PHP 代理，首先 Ajax 向本域内 PHP 发送请求，然后 PHP 通过 curl 向 node js Web Server 发送请求，node js Web Server 响应请求将结果返回给网页。对于前台而言，是不需要知道 node js Web Server 的地址的。

## 2. 网页数据动态更新的实现

服务器推送技术，现有的解决方案包括三种，分别是 HTML refresh、基于插件技术的服务器推送技术、基于 Ajax 的服务器推送技术<sup>[35,36]</sup>，而在基于 Ajax 的服务器推送技术中，主要包括基于 Ajax poll 实现、基于长论询的服务器推送技术、HTTP streaming 实现等三种方式。

根据本微博系统要实现的功能，网页数据动态更新采用基于 Ajax 的服务器推送技术中的第一种—基于 Ajax poll 的方式来实现。由于服务器端提供了“/opt/get\_new/:user\_id”接口，而这个接口访问的数据都存放在 Redis 数据库中，因此网页数据动态更新响应时间是很短的，不会影响系统的性能。

## 3. 前台效果图

本微博系统首页效果如图 5-10 所示。



图 5-10 首页效果图

## 5.4 本章小结

本章对于微博系统进行了实现。首先对用户管理、微博管理、用户关系管理、评论和私信、搜索用户、视频分享、图片管理、短链接、敏感词处理、热点话题



生成、备份、系统缓存等核心模块的阐述了具体的实现方案，然后描述了数据连接池、后台接口和前台的实现，并展示了最终效果图。由于篇幅所限，对于那些辅助系统完成微博特色功能的功能模块的实现方案没有进行阐述。

## 第六章 微博系统的测试

### 6.1 功能测试

#### 1. 基本功能测试

系统基本功能主要包括：微博的发布、转发、评论、删除，查看发布的微博、查看收藏的微博，发布评论、查看评论，@功能、发私信、查看私信，用户登陆、注册等。

基本功能测试主要采用黑盒测试和白盒测试两种技术，其测试范围如表 6-1 所示。

表 6-1 基本功能的测试范围

序号	所属模块	功能点	备注
1	微博管理	微博的发布	
		微博的转发	
		微博的评论	
		微博的删除	只能删除当前用户的微博
		查看已发布微博	
		查看收藏的微博	
2	评论与私信管理	发布评论	
		查看评论	
		@功能	@需要有自动补全的提示
		查看@我的微博	
		发私信	只能查看当前用户自己的私信
		查看私信	
3	用户管理	用户注册	
		用户登录	
		用户注销	
		更改用户信息	

#### 2. 其它功能测试

系统的其它功能主要包括：微博表情的处理、图片微博的处理、视频微博的处理，敏感词的处理，短链接生成，热门话题的生成与推荐，热门用户的推荐等。

其它功能测试主要采用黑盒测试和白盒测试两种技术，其测试范围如表 6-2 所示。

表 6-2 其它功能的测试范围

序号	所属模块	功能点	备注
1	微博管理	微博情绪的发布与展示	测试手动输入表情代码是否能够显示出表情。
		图片微博的处理	分析测试图片链接和本地图片的上传方式
		视频微博的处理	KU6 网和微酷网的视频测试
2	短链接	短链接生成与转义	
3	敏感词	敏感词的检测	
4	热门话题管理	热门话题统计与自动推荐	当输入话题的时候，是否有自动补全功能
5	热门用户管理	热门用户统计与自动推荐	
6	系统新消息推送	系统新消息推送和提醒功能	是否进行了消息分类与统计

### 3. 测试环境

硬件环境为：

服务器配置为：CPU 采用 Intel i3 2.6GHz，内存为 4G，硬盘容量是 80G；客户端配置为：CPU 为 Intel T5870 2.0G Hz，内存是 2G，硬盘容量为 250G。

软件环境为：

服务器端：操作系统 Ubuntu 10.04 lts、node js 0.69、MongoDB 2.0.6、Redis2.4.5、PHP5.4 和 Web 服务器软件 Apache2.2。客户端：操作系统 Windows 7、Windows xp，浏览器 Chrome19、IE8、IE9、FireFox13。

### 4. 测试结果

经过测试得出结论：系统的基本功能和扩展功能均表现正常，很好地达到了预期目标。

## 6.2 性能测试

### 1. 测试目标

由于系统性能主要取决于后台的响应时间，因此性能测试主要是对于系统的后台接口进行，根据测试结果数据对于系统的支撑能力进行分析，进一步判断系统是否能够在多用户并发登陆、以及多用户增删改查等并发操作情况下稳定地运

行。

这里将分别模拟 500 人同时在线、1000 人同时在线、2000 人同时在线、3000 人同时在线的情况，测试用户登陆和用户发布微博两个接口，依次统计系统响应时间。

## 2. 测试环境

服务器配置为：CPU 采用 Intel i3 2.6GHz，内存为 4G，硬盘容量是 80G，操作系统 Ubuntu 10.04 lts、node js 0.6.9、MongoDB 2.0.6、Redis2.4.5、PHP5.4 和 Web 服务器软件 Apache2.2。

客户端配置为：CPU 为 Intel T5870 2.0G Hz，内存是 2G，硬盘容量为 250G，操作系统 Ubuntu 10.04 lts。

局域网带宽 1000Mb，由带宽引起的数据延迟可以忽略。

## 3. 测试结果

(1)500 人并发登陆测试通过，服务器稳定，对增删改查等操作能够正常处理，但是 CPU 和内存的利用率均较低。

(2)1000 人并发登陆测试通过，服务器稳定，对增删改查等操作能够正常处理，但是 CPU 和内存的利用率均较低。

(3)2000 人并发登陆测试通过，服务器稳定，对增删改查等操作能够正常处理，CPU 和内存的利用率均较高。

(4)3000 人并发登陆测试没有通过，系统处理请求的响应时间变慢，大量的处理请求响应时间超高，还有的处理请求未响应，Redis 数据库占用内存量过高，但是 CPU 与内存利用率均较高。

## 4. 测试过程分析

### (1)500 个用户同时在线

系统拥有 4900 个注册用户，其中的 500 个用户来执行测试任务，首先这 500 个用户并发进行登录操作；然后在 500 个用户在线的情况下，这 500 个用户并发提交发布微博任务。

测试结果：所有用户成功登录，所有微博发布成功，且平均响应时间不超过 0.1 秒。

### (2)1000 个用户同时在线

系统拥有 4900 个注册用户，其中的 1000 个用户来执行测试任务，首先这 1000 个用户并发进行登录操作；然后在 1000 个用户在线的情况下，这 1000 个用户并发提交发布微博任务。

测试结果：所有用户成功登录，所有微博发布成功，且平均响应时间不超过

0.5 秒。

(3)2000 个用户同时在线

系统拥有 4900 个注册用户，其中的 2000 个用户来执行测试任务，首先这 2000 个用户并发进行登录操作；然后在 2000 个用户在线的情况下，这 2000 个用户并发提交发布微博任务。

测试结果：98.7%的用户成功登录，96.5%的微博发布成功，平均响应时间不超过 1.5 秒。

(4)3000 个用户同时在线

系统拥有 4900 个注册用户，其中的 3000 个用户来执行测试任务，首先这 3000 个用户并发进行登录操作；然后在 3000 个用户在线的情况下，这 3000 个用户并发提交发布微博任务。

测试结果：仅 46.7%的用户成功登录，38.6%的微博发布任务成功，平均响应时间超过 5 秒，部分请求超时，未进入统计结果。

## 6.3 本章小结

本章对微博系统进行了功能测试与性能分析，给出了功能测试和性能测试的结果。通过分析可知，本微博系统能够支持 2000 人同时在线，能够满足一个中小型微博系统的功能和性能要求。

## 第七章 结论

微博作为一种社交网络平台，借以内容简短、传播快速、实时性强、互动性强的特性满足了人们充分分享信息和交流信息的需求。本文结合实际应用，为满足中小型微博系统的高性能要求，设计并实现一个基于 node js 的微博系统，实现用户之间相互关注、发布新微博、主页微博内容定时刷新等功能。基于 node js 的微博系统后台框架采用 node js 技术，主要存储数据库采用 MongoDB，主要缓存数据库采用 Redis。

本文的主要工作包括以下几个方面：

1. 微博系统的需求分析：分析了微博系统的功能需求，并进行了业务流程分析。基于 node js 的微博系统主要具有关注、评论、转发、插入图片、插入话题、私信、短信、编辑、视频和音乐、搜索、分页、收藏、推荐、移除粉丝、备份等功能。系统将用户分为普通用户和管理员用户两类，管理员用户和普通用户拥有不同的操作权限，要登录的用户只有通过身份认证后才能进入系统。

2. 微博系统的设计：给出了微博系统的拓扑结构，进行了架构设计和功能结构设计。基于 node js 的微博系统遵循可用性、开放性、可扩展性的设计原则，采用 B/S 体系结构风格，将系统分为数据层、业务逻辑层和表现层实现了数据访问与业务逻辑的分离，使得页面更具动态性。

3. 微博系统的实现：给出了用户管理、微博管理、用户关系管理、评论和私信、搜索用户、视频分享、图片管理、短链接、敏感词处理、热点话题生成、备份、系统缓存等核心模块的具体的实现方案，描述了数据连接池、后台接口和前台的实现，并展示了最终效果图。

4. 微博系统的测试：对微博系统进行了测试与性能分析，给出了功能测试和性能测试的结果。

通过分析可知，本微博系统能够能够满一个中小型微博系统的功能和性能要求。

基于 node js 的微博系统具有轻量型、易布署的特点，能够支持 2000 人同时在线，适合于中小规模用户量的微博应用。

## 致 谢

本论文是在我的导师白金平教授的悉心指导下完成的。本论文倾注了导师大量的心血，无论是论文的选题、研究工作，还是论文的撰写工作均是在导师的悉心指导下完成的。白金平老师高尚的品德，缜密的思维，严谨的治学态度，以及可贵的职业道德和敬业精神，是我学习的榜样，尤其是白金平老师对我的科研能力和工程实践能力的培养对我帮助较大，在此表示感谢。

感谢所有热心帮助和耐心指导过我的电子科技大学的老师。



感谢与我共同学习的同学们给予我的帮助和鼓励，感谢我的家人对我学习上的支持和理解，以及生活上的关怀和照顾。

## 参考文献

- [1] 祝阳, 王欢. 微博的政治影响力研究[J]. 重庆邮电大学学报(社会科学版), 2013, 25(4): 84-89
- [2] Yuan Y. Connie, Zhao Xuan, Liao Qinying, et al. The use of different information and communication technologies to support knowledge sharing in organizations: From e-mail to micro-blogging[J]. Journal of the American Society for Information Science and Technology, 2013, 64(8): 1659-1670
- [3] Armentano M. G., Godoy D., Amandi A. A. Followee recommendation based on text analysis of micro-blogging activity[J]. Information Systems, 2013, 38(8): 1116-1127
- [4] 李英乐, 于洪涛, 刘力雄. 基于 SVM 的微博转发规模预测方法[J]. 计算机应用研究, 2013, 30(9): 2594-2597
- [5] 吴凯, 季新生, 刘彩霞. 基于行为预测的微博网络信息传播建模[J]. 计算机应用研究, 2013, 30(6): 1809-1893
- [6] 蔡波斯, 陈翔. 基于行为相似度的微博社区发现研究[J]. 计算机工程, 2013, 39(8): 55-59
- [7] 闫光辉, 赵红运, 任亚缙, 等. 基于时间特性的微博热门话题检测算法研究[J]. 计算机应用研究, 2013, 31(10): 123-128
- [8] 闫光辉, 舒昕, 马志程, 等. 基于主题和链接分析的微博社区发现算法[J]. 计算机应用研究, 2013, 30(7): 1953-1957
- [9] 李开复. 微博: 改变一切[M]. 上海: 上海财经大学出版社, 2011: 76-78
- [10] Michael Abernethy. Just what is node js?[M]. IBM developer works, 2011: 78-92
- [11] Makagonov S. Synchronization between desktop application and web clients provided by node js software system[J]. Computer Modelling and New Technologies, 2012, 16(3): 41-44
- [12] Tilkov Stefan, Vinoski Steve. node js: Using JavaScript to build high-performance network programs[J]. IEEE Internet Computing, 2010, 14(6): 80-83
- [13] Tom Hughes-Croucher, Mike Wilson. node Up and Running[M]. USA: O'Reilly Media, Inc, 2012: 58-67
- [14] 张煜. 一种使用 node.js 构建的分布式数据流日志服务系统[J]. 计算机系统应用, 2013, 22(2): 68-71
- [15] Atzeni Paolo, Bugiotti Francesca, Rossi Luca. Uniform access to NoSQL systems[J]. Information Systems, 2014, 43(6): 117-133
- [16] Pokorny Jaroslav. NoSQL databases: A step to database scalability in web environment[J]. International Journal of Web Information Systems, 2013, 9(1): 69-82



- [17] Tsuyuzaki Kota, Onizuka Makoto. NoSQL database characteristics and benchmark system[J]. NTT Technical Review, 2012, 10(12): 65-72
- [18] 申德荣, 于戈, 王习特, 等.支持大数据管理的 NoSQL 系统研究综述[J]. 软件学报, 2013, 24(8): 1786-1803
- [19] 章文盛, 郑汉华.基于 MongoDB 构建高性能网站技术研究[J]. 吉林师范大学学报( 自然科学版), 2013,(1):123-127
- [20] Kyle Banker. MongoDB in Action[M]. USA: Manning Publications, 2011: 37-44
- [21] Parker Zachary, Poe Scott, Vrbsky, Susan V. Comparing NoSQL MongoDB to an SQL DB[C]. Proceedings of the Annual Southeast Conference, Proceedings of the 51st ACM Southeast Conference, ACMSE 2013, Savannah, GA, United states, April 4, 2013 - April 6, 2013: 54-59
- [22] Boicea Alexandru, Radulescu Florin, Agapin Laura Ioana. MongoDB vs Oracle - Database comparison[C]. Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012, Bucharest, Romania , September 19, 2012 - September 21, 2012: 330-335
- [23] 何胜韬. MongoDB 数据库在网络行为分析与控制系统中的应用[J]. 网络安全技术与应用, 2013(5): 11-13
- [24] 秦秀磊, 张文博, 魏峻, 等.云计算环境下分布式缓存技术的现状与挑战[J]. 软件学报, 2013, 24(10): 34-39
- [25] 李东阳, 刘鹏, 丁科, 等.基于固态硬盘的云存储分布式缓存策略[J]. 计算机工程, 2013, 39(4): 54-58
- [26] 马祥杰, 李晓中, 范兴隆, 等. 基于服务标识的中间级缓存多级多平面分组交换时延保证调度机制研究[J]. 计算机学报, 2013, 36(1): 79-63
- [27] 张艳卿, 李金宝, 郭龙江, 等. Multi-Radio 无线传感器网络中基于缓存和信道切换的数据查询算法的研究[J]. 计算机学报, 2012, 35(11): 68-72
- [28] 刘外喜, 余顺争, 蔡君, 等. ICN 中的一种协作缓存机制[J]. 软件学报, 2013, 24(8): 55-60
- [29] Tiago Macedo , Fred Oliveria. Redis Cookbook[M]. USA: O'Reilly Media, Inc , 2011: 54-62
- [30] Gao Xiaobo, Fang Xianmei. High-performance distributed cache architecture based on redis[J]. Lecture Notes in Electrical Engineering, 2013, 270(1):105-111
- [31] 曾超宇, 李金香. Redis 在高速缓存系统中的应用[J]. 微型机与应用, 2013 , 32 (12): 11-13
- [32] 查修齐, 吴荣泉, 高元钧. C/S 到 B/S 模式转换的技术研究[J]. 计算机工程, 2014, 40(1): 263-267
- [33] Lu Ran, Xue Suzhi, Ren Yuanyuan. A modified approach of hot topics found on micro-blog[J]. Lecture Notes in Electrical Engineering, 2014,(269): 603-614

- [34] 丁兆云, 贾焰, 周斌. 微博数据挖掘研究综述[J]. 计算机研究与发展, 2014, 51(4): 691-706
- [35] 王振兴, 黄静.  基于 PHP 和服务端推送技术的 WEB 即时聊天系统[J]. 计算机系统应用, 2012, 21(12): 64-69
- [36] 陈航, 赵方.  基于服务端推送技术和 XMPP 的 WEB IM 系统实现[J]. 计算机工程与设计, 2010, 31(5): 69-73

# 基于nodejs的微博系统的设计与实现

作者: [王越](#)  
学位授予单位: [电子科技大学](#)

引用本文格式: [王越](#) [基于nodejs的微博系统的设计与实现](#)[学位论文]硕士 2014