**【摘 要】**

互联网和计算机行业发展迅猛，新技术层出不穷，作为计算机行业从业者，为了不被行业淘汰，我们需要不断学习各种新知识，新技术，在学习过程中很多人会收藏别人写的优秀文章，也会撰写自己的原创技术博客，因此，许多人都会开发自己的技术博客网站记录和展示自己的学习成果。但是目前大多数个人开发的博客网站仅能展示博主自己撰写的文章，而不能展示博主在其他博客平台收藏的文章。因此，如果有一个集合博主的原创文章以及博主在其他博客网站收藏的文章的博客平台，将大大方便博主回顾自己学习过的知识。

本文介绍了一个基于postgreSQL数据库，以Node.js搭建后台，前端基于React技术栈的个人博客网站的设计与实现过程。该网站后台系统定时爬取博主在思否、简书两大主流博客平台收藏的文章并展示，支持博主撰写、修改和删除原创文章，同时支持博客浏览者浏览博主收藏和原创的文章，注册并登录该网站后还可以对博主的原创文章点赞、评论，以及回复他人的评论。

**【关键词】：**博客；postgreSQL数据库；Node.js；React技术栈；

[ABSTRACT]

The Internet and computer industries are developing rapidly, and new technologies are emerging. As a practitioner in the computer industry, in order not to be eliminated by the industry, we need to constantly learn all kinds of new knowledge and new technologies, and many people will collect excellent articles from others during the learning process. Also They will write their own original technology blog. Therefore, many people choose to develop their own technology blog sites to record and show their learning outcomes. However, most of the personally developed blog sites can only display bloggers' own articles, but not bloggers' articles on other blog platforms. Therefore, if there is an blog platform which contains blogger’s original articles and the articles collected by the blogger on other blog sites, it will greatly facilitate the blogger to review the knowledge he has learned.

This paper introduces the design and implementation process of a personal blog website based on the PostgreSQL database, with Node.js built in the back-end system and front-end based on React technology stack. The back-end system of the website regularly crawls and displays the articles collected by the bloggers on the two major blog platforms, such as Segment Fault and Jianshu. At the same time, it supports bloggers to compose, modify and delete original articles, and supports visitors to browse bloggers' collections and original articles. After registering and logging in to the site, visitors can also like and comment on bloger’s original articles, and respond to the comments from others.

**[Keywords]:** Blog; postgreSQL; Node.js; React

目 录

第一章 绪论 1

1.1 背景 1

1.2 论文结构简介 1

第二章 系统相关技术概述 3

2.1 PostgreSQL 3

2.2 javaScript 3

2.3 Ajax技术 4

2.4 Node.js 5

2.5 RESTful API 6

2.6 React.js 6

2.7 Redux 7

第三章 系统分析和设计 9

3.1 系统可行性分析 9

3.1.1 技术可行性 9

3.1.2 经济可行性 10

3.1.3 操作可行性 10

3.1.4 法律可行性 10

3.2 需求分析 10

3.2.1 功能需求分析 10

3.2.2 用例分析 11

3.3 系统概要设计 11

3.4 系统数据库设计 14

3.5 后端API设计 18

3.6 前端路由设计 20

3.7 前端Redux store设计 21

第四章 详细设计 23

4.1 爬虫模块设计 23

4.2 后端功能模块设计 25

4.2.1 数据库操作模块设计 25

4.2.2 路由转发模块设计 26

4.3 前端功能模块设计 27

4.3.1 登录模块组件设计 27

4.3.2 文章列表与内容模块组件设计 28

4.3.3 点赞模块组件设计 29

4.3.4 评论模块组件设计 30

4.3.5 回复模块组件设计 31

4.3.6 消息通知模块组件设计 32

第五章 系统测试 34

5.1 测试目的 34

5.2 测试方法 34

5.2.1 单元测试 34

5.2.2 黑盒测试 35

5.3 测试样例 35

5.3.1 单元测试 35

5.3.2 黑盒测试 36

第六章 总结与展望 38

6.1 总结 38

6.2 展望 38

参考文献： 40

致谢 42

# 绪论

## 背景

互联网和计算机行业发展迅猛，新技术层出不穷，作为计算机行业从业者，为了不被行业淘汰，我们需要不断学习各种新知识，新技术，在学习过程中我们会收藏别人写的优秀文章，也会撰写自己的原创技术博客，因此，许多人都有自己的博客以记录和展示自己的学习成果。

通常开设博客有两种形式，一种是在现有的博客平台注册账号撰写博文，这样的好处是没有搭建博客平台的时间和技术成本，同时依赖现有成熟博客平台的大量用户群，我们撰写的文章也能有较高的访问流量，被更多人看到，但是通过这种方式我们就没有办法个性化定制自己博客的页面，每个人的博客页面都是千篇一律，不够个性化。另一种开设博客的方式就是由开发者自己设计个人博客并开发建站，这样我们就可以最大化地个性定制我们自己的博客页面，但是这样的不好处是我们的博客只能展示自己原创的文章，而不能展示我们在其他博客平台收藏的文章。

因此，我开发了一个自己的个人博客，个性化定制了博客页面，支持普通个人博客系统的文章撰写、发布、修改、删除功能，也支持博客访问者点赞和评论文章，同时还通过网络爬虫的方式定时爬取我在思否（SegmentFault）和简书这两大博文平台上收藏的文章，当我在这两个网站收藏了新文章时，该博客系统也能更新显示我添加的新收藏。这样就整合了我的学习资源，方便我回顾学习过的内容。

## 论文结构简介

本论文共分为七章，首先介绍了项目相关的应用背景和技术背景，然后论文就系统的业务需求以及整体架构设计进行了分析，紧接着介绍了具体功能模块的设计与实现。最后介绍了项目测试、部署和运行效果，并对项目进行了总结和展望。

论文具体安排如下：

第一章：介绍了个人博客项目的应用背景，以及实现该系统的意义。

第二章：介绍了项目应用到的数据库、后端技术栈、前端技术栈以及其他相关的技术背景，同时介绍了这些技术的发展现状以及本项目选用这些技术的原因。

第三章：分析了系统功能需求，详细介绍了该个人博客系统的整体架构设计，包括数据库设计，后端模块与接口设计，前端路由和数据状态设计。

第四章：详细介绍了各个功能模块的具体设计和实现。

第五章：介绍了该个人博客系统的系统测试样例和运行结果

第六章：最后论文总结了毕业设计所做的工作，并提出目前项目存在的不足之处，展望改进方向。

# 系统相关技术概述

## PostgreSQL

PostgreSQL 可以说是目前功能最强大、特性最丰富和结构最复杂的开源数据库管理系统，甚至具有某些商业数据库都不具备的特性。该数据库起源于加州大学伯克利分校，已成为一个国际发展项目，拥有庞大的用户群，尤其是海外用户群，并且有越来越多的国内用户。

PostgreSQL 基本上算是见证了整个数据库理论和技术的发展历程，由 UCB 计算机教授 Michael Stonebraker 于 1986 年创建。在此之前，Stonebraker 教授主导了关系数据库 Ingres 研究项目，88 年，提出了 Postgres 的第一个原型设计。

MySQL 号称是使用最广泛的开源数据库，而 PG 则被称为功能最强大的开源数据库。

PostgreSQL的特点如下：

1. PostgreSQL可在所有主要操作系统(即Linux，UNIX(AIX，BSD，HP-UX，SGI IRIX，Mac OS X，Solaris，Tru64)和Windows等)上运行。
2. PostgreSQL支持文本，图像，声音和视频，并包括用于C/C++，Java，Perl，Python，Ruby，Node.js，Tcl和开放数据库连接(ODBC)的编程接口。
3. PostgreSQL支持SQL的许多功能，例如复杂SQL查询，SQL子选择，外键，触发器，视图，事务，多进程并发控制(MVCC)，流式复制(9.0)，热备(9.0))。
4. 在PostgreSQL中，表可以设置为从“父”表继承其特征。
5. 可以安装多个扩展以向PostgreSQL添加附加功能。

## 网络爬虫

网络爬虫也叫网络蜘蛛，即Web Spider，如果把互联网比喻成一个蜘蛛网，那么Spider就是在网上爬来爬去的蜘蛛。网络蜘蛛通过网页的链接地址来寻找网页，从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其它链接地址，然后通过这些链接地址寻找下一个网页，一直循环下去，直到把整个网站所有的网页都抓取完为止。

在网络爬虫的系统框架中，主过程由控制器，解析器，资源库三部分组成。控制器的主要负责给多线程中的各个爬虫线程分配工作任务。解析器的主要工作是下载网页，进行页面的处理，主要是将一些JS脚本标签、CSS代码内容、空格字符、HTML标签等内容处理掉，爬虫的基本工作是由解析器完成。资源库是用来存放下载到的网页资源，一般都采用大型的数据库存储，如Oracle数据库，并对其建立索引。

控制器：控制器是网络爬虫的中央控制器，它主要是负责根据系统传过来的URL链接，分配一线程，然后启动线程调用爬虫爬取网页的过程。

解析器：解析器是负责网络爬虫的主要部分，其负责的工作主要有：下载网页的功能，对网页的文本进行处理，如过滤功能，抽取特殊HTML标签的功能，分析数据功能。

资源库：主要是用来存储网页中下载下来的数据记录的容器，并提供生成索引的目标源。中大型的数据库产品有：Oracle、Sql Server等。

## Ajax技术

Ajax技术的核心是XMLHttpRequest对象（简称XHR），这是由微软首先引入的一个特性，其他浏览器提供商后来都提供了相同的实现。在XHR出现之前，Ajax式的通信必须借助一些hack手段来实现，大多数是使用隐藏的框架或内嵌框架。XHR为向服务器发送请求和解析服务器响应提供了流畅的接口。能够以异步方式从服务器获得更多信息，意味着用户单击后，可以不必刷新页面也能取得新数据。也就是说，可以使用XHR对象取得新数据，然后再通过DOM将新数据插入到页面中。另外，虽然名字中包含XML的成分，但Ajax通信与数据格式无关，现在服务端与客户端通信一般采用JSON格式的数据。这种技术就是无须刷新页面即可从服务器取得数据，但不一定是XML数据。

实际上，在Ajax这个名字被正式提出之前，这种技术已经存在很长时间了，人们过去通常将这种技术叫做远程脚本（remoting scripting），而且早在1998年就有人采用不同的手段实现了这种浏览器与服务端的通信。再往前推，JavaScript需要通过java applet或flash电影等中间层向服务器发送请求。而XHR则将浏览器原生的通信能力提供给了开发人员，简化了实现同样操作的任务。

在被命名为Ajax后，这种浏览器与服务器的通信技术可谓红极一时。人们对JavaScript和web的全新认识，催生了很多使用原有特性的新技术和新模式。就目前来说，熟练使用XHR对象已经成为所有Web开发人员必须掌握的一种技能。也正因为Ajax技术，与客户交互的复杂逻辑才从后端转移到了前端，催生了后来的Angular、React、Vue等前端框架

## Node.js

JavaScript最早是运行在浏览器中，然而浏览器只是提供了一个上下文，它定义了使用JavaScript可以做什么，但并没有“说”太多关于JavaScript语言本身可以做什么。事实上，JavaScript是一门“完整”的语言： 它可以使用在不同的上下文中，其能力与其他同类语言相比有过之而无不及。

Node.js，事实上就是另外一种上下文，它允许在后端（脱离浏览器环境）运行JavaScript代码。

要实现在后台运行JavaScript代码，代码需要先被解释然后正确的执行。Node.js的原理正是如此，它使用了Google的V8虚拟机（Google的Chrome浏览器使用的JavaScript执行环境），来解释和执行JavaScript代码。

除此之外，伴随着Node.js的还有许多有用的模块，它们可以简化很多重复的劳作，比如向终端输出字符串。

因此，Node.js事实上既是一个运行时环境，同时又是一个库。Node.js将前端中广泛运用的异步编程和事件驱动的思想迁移到了服务端，采用单线程，异步I/O，搭配事件循环，使得Node.js非常适合处理大量的并发I/O请求。而本个人博客系统在后端的业务逻辑，主要就是对数据库的增删查改，是一个I/O密集型应用，因此非常适合采用Node.js作为后端开发语言。

## RESTful API

REST的全称是Representational State Transfer，中文含义即表现层状态转化。MVC模式大行其道许多年，一直到RESTful的流行，大家才意识到URL也可以设计得很规范，请求方法也能作为逻辑分发的单元。符合REST规范的设计，我们称为RESTful设计。他的设计哲学主要将服务器端提供的内容实体看作一个资源，并表现在URL上，对资源的操作体现在请求方法上，例如，一个用户资源对应的url可以被设计为/user/jack，而查看，修改，删除，添加该资源则分别是通过以GET、PUT、DELETE、POST方法请求该url来实现的，对于这个资源的表现形态，不再像以前一样表现在url的文件后缀上，而是在请求报头的Accept字段和服务器端的支持情况来决定的。所以REST的设计就是，通过URL设计资源、请求方法定义资源的操作、通过Accept决定资源的表现形式。

## React.js

React.js是Facebook开发的一个开源前端MVVM（Model-View-Viewmodel）框架。 基于 HTML 的前端界面开发正变得越来越复杂，其本质问题基本都可以归结于如何将来自于服务器端或者用户输入的动态数据高效的反映到复杂的用户界面上。而React框架正是完全面向此问题的一个解决方案，按官网描述，其出发点为：用于开发数据不断变化的大型应用程序（Building large applications with data that changes over time）。相比传统型的前端开发，React 开辟了一个相当另类的途径，实现了前端界面的高效率高性能开发。在 Web 开发中，我们总需要将变化的数据实时反应到 UI 上，这时就需要对 DOM 进行操作。而复杂或频繁的 DOM 操作通常是性能瓶颈产生的原因（如何进行高性能的复杂 DOM 操作通常是衡量一个前端开发人员技能的重要指标）。React 为此引入了虚拟 DOM（Virtual DOM）的机制：在浏览器端用 Javascript 实现了一套 DOM API。基于 React 进行开发时所有的 DOM 构造都是通过虚拟 DOM 进行，每当数据变化时，React 都会重新构建整个 DOM 树，然后 React 将当前整个 DOM 树和上一次的 DOM 树进行对比，得到 DOM 结构的区别，然后仅仅将需要变化的部分进行实际的浏览器 DOM 更新。而且 React 能够批处理虚拟 DOM 的刷新，在一个事件循环（Event Loop）内的两次数据变化会被合并，例如你连续的先将节点内容从 A 变成 B，然后又从 B 变成 A，React 会认为 UI 不发生任何变化，而如果通过手动控制，这种逻辑通常是极其复杂的。尽管每一次都需要构造完整的虚拟 DOM 树，但是因为虚拟 DOM 是内存数据，性能是极高的，而对实际 DOM 进行操作的仅仅是 Diff 部分，因而能达到提高性能的目的。这样，在保证性能的同时，开发者将不再需要关注某个数据的变化如何更新到一个或多个具体的 DOM 元素，而只需要关心在任意一个数据状态下，整个界面是如何 Render 的。

## Redux

React 只是 DOM 的一个抽象层，并不是 Web 应用的完整解决方案。有两个方面它没有涉及，那就是代码结构和组件之间的通信，对于大型的复杂应用来说，这两方面恰恰是最关键的。因此，只用 React 没法写大型应用。

为了解决这个问题，2014年 Facebook 提出了 Flux架构的概念，引发了很多的实现。2015年，Redux出现，将 Flux 与函数式编程结合一起，很短时间内就成为了最热门的前端架构。

如果前端项目比较复杂，具有多交互，多数据源的特点，比如：

* 用户的使用方式复杂
* 不同身份的用户有不同的使用方式（比如普通用户和管理员）
* 多个用户之间可以协作
* 与服务器大量交互，或者使用了WebSocket
* View要从多个来源获取数据

那么用Redux管理数据状态就是非常有用的，从组件角度看，如果Web应用有以下场景，可以考虑使用 Redux。

* 某个组件的状态需要共享
* 某个状态需要在任何地方都可以拿到
* 一个组件需要改变全局状态
* 一个组件需要改变另一个组件的状态

发生上面情况时，如果不使用 Redux 或者其他状态管理工具，不按照一定规律处理状态的读写，代码很快就会变成一团乱麻。Redux提供了一种机制，可以在同一个地方查询状态、改变状态、传播状态的变化。

# 系统分析和设计

## 系统可行性分析

可行性研宄是在项目开发前期对项目的一种考察和鉴定，对拟议中的项目进行全面的、综合的调查研宄，其目的是要判断项目可行与否。信息系统技术可行性研宄要从系统开发的计划出发，论述系统开发力量的可行性，同时论证系统方案中所采取的各种技术手段上是否可实现。

由于系统建设是一项投资大、涉及面广、工程复杂的系统工程，因此必须充分的进行可行性论证，以确保投资的准确无误，而且信息系统建设是一项整体工程，必须站在系统的角度论证它的可行性才有说服力，才有意义。可行性研宄的目的是用最小的代价，在尽可能短时间内确定问题是否能够解决，它的目的不是解决问题，而是确定问题是否值得去解决，可行性从以下四个方面来考虑。

### 技术可行性

本个人博客项目是典型的基于B/S架构的web应用，后端业务逻辑主要是对数据库中的数据进行增删查改，实现相对简单。同时前端展示的是博文列表和博文内容，涉及到的用户交互主要是登录，注册，点赞，博主撰写文章，评论和回复的提交，也不是很复杂。再加上已经有较成熟的框架，因此开发这个系统的技术难题不多，具备技术可行性。

### 经济可行性

本项目是一个Web应用，部署好之后只需一个浏览器就可以访问，后期如果给网站申请域名也仅需很少的经济成本，可以忽略不计，同时该系统是个人独立开发的系统，后期维护也不用投入成本，因此经济上基本上零成本的。

### 操作可行性

本系统参照其他博客系统，对于博客访问者来说，涉及到的用户交互都很简单，没有复杂操作，作为设计者与开发者，我也没有加入让用户难以理解和操作的交互方式。

### 法律可行性

本系统不会违反国家法律，爬取和展示的文章是我本人在简书和思否两大网站收藏的免费公开文章，同时展示时也注明了原文作者和出处，不涉及侵权问题

综上所述，该系统在技术上、经济上、可操作性上、法律上都是可行的，而且要求不高，所以该系统的开发是可行的。

## 需求分析

### 功能需求分析

本系统涉及的用户角色有两个，即访客和博主。

对于访客，又分为已注册访客和未注册访客两种，未注册访客仅可以查看博主收藏和原创的文章。对于已注册访客，应实现以下功能：

1. 能查看博主收藏和原创的文章。
2. 能点赞、评论博主的原创文章。
3. 对于其他人在博主原创文章下的评论，可以进行回复讨论。
4. 可以查看博主原创文章下与自己相关的回复（包括其他人对自己的回复，以及自己的评论下的所有回复）。对于自己还没有查看过的相关回复，系统有未读提示。

对于博主，即本系统管理者，除了能使用访客的所有功能外，还应实现如下功能：

1. 在思否、简书两大博客平台收藏了优秀文章后，系统会在某个时间对这两个网站中博主的收藏夹页面进行爬虫，获取博主新收藏的文章后更新本系统中博主的收藏夹数据库。
2. 能用markdown语法撰写并发布原创文章，同时对于已发布的文章，可以修改和删除。
3. 能查看访客对博主原创文章下的评论，点赞，回复，对于还未查看过的相关消息，系统有未读提示。

### 用例分析

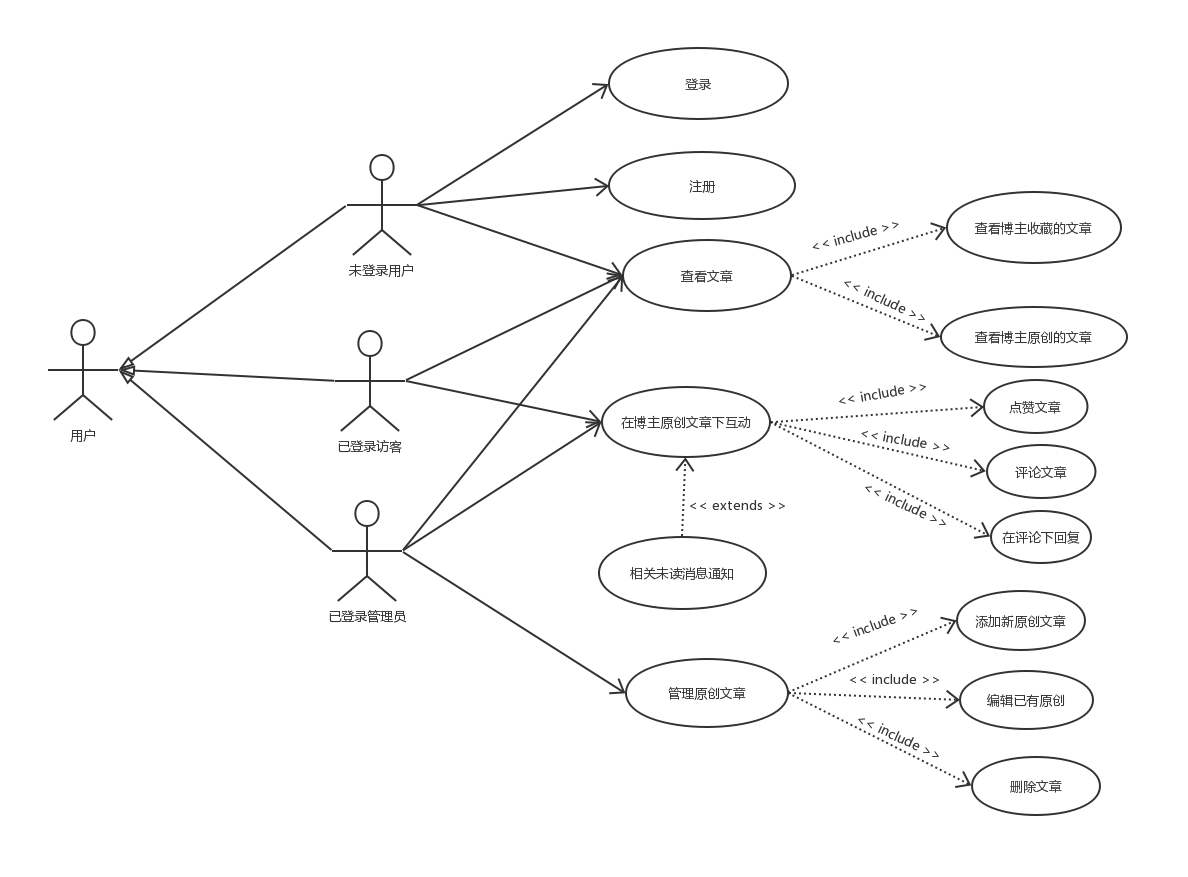


图3-1 系统用例图

## 系统概要设计

要实现该系统，我们需要从三个方面进行设计，即数据库设计，后端设计和前端设计。

本系统使用的数据库是PostgreSQL，是关系型数据库，需要存储用户信息、博主收藏的文章、原创的文章，原创文章下的点赞、评论、回复，以及与用户相关的通知。与后端通过Node.js与PostgreSQL数据库的程序连接接口进行连接。

后端是实现一个响应RESTful API的Web服务，关键主要是实现三大模块，一是爬虫模块，即实现每隔24小时爬取一次我在简书和思否两大博客平台收藏的文章，然后将新收藏的文章存储到数据库中；二是数据库操作模块，即通过Node.js与PostgreSQL数据库的程序连接接口，执行不同的SQL语句，对数据库里的数据进行查询、删除、新增和修改。三是路由模块，即解析前端通过AJAX发送的HTTP请求报文，根据不同的路由请求，运行数据库操作模块对应的处理程序，然后将数据库操作返回的数据通过HTTP响应报文返回给前端处理。

前端需要实现的是一个单页应用，用React-router管理前端路由。页面分为页头，页面主体和页脚三部分，页头展示导航栏和登录/注册组件，页脚展示博主的联系方式等简要信息，页面主体根据前端路由，分别展示以下的内容：

* 个人主页：显示博主的简历信息。
* 注册页：供用户填写注册信息，注册为系统用户。
* 收藏夹文章列表页：展示博主收藏的文章列表，每个列表项包括文章的标题和发布日期，列表内容由收藏夹标签[[1]](#footnote-1)和文章标签[[2]](#footnote-2)决定。
* 收藏夹文章内容页：展示博主收藏的文章的内容，包含文章标题、作者、内容、发布日期。
* 博主原创文章列表页：展示博主原创的文章列表，每个列表项包括文章的标题、发布日期、点赞数、评论数。
* 博主原创文章内容页：展示博主原创文章的内容，包含文章标题、作者、内容、点赞数、回复列表、每条回复下的评论列表。同时包含点赞按钮，评论框和回复框供用户交互。

前端页面以React组件的形式进行组织，通过Ajax技术向后端动态请求数据，

对于多组件公用的数据，通过Redux进行状态管理。

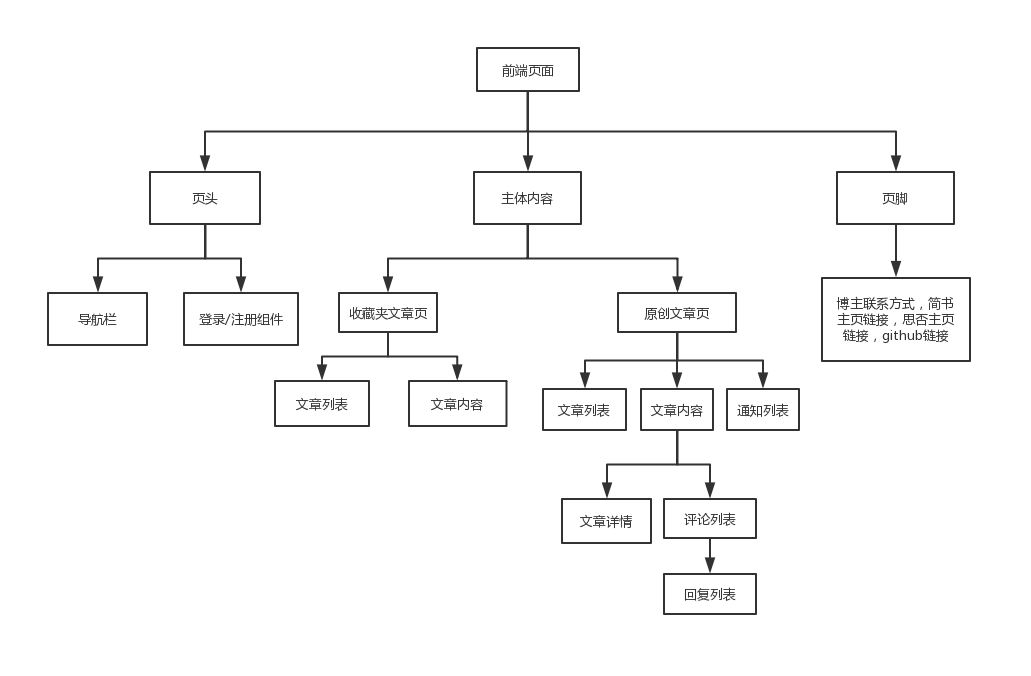


图3-2 前端页面结构图

系统数据流图如下所示：

|  |  |
| --- | --- |
| **图例** | **含义** |
|  | 实体 |
|  | 数据加工 |
|  | 数据存储 |
|  | 数据流 |

表 3-1 数据流图图例

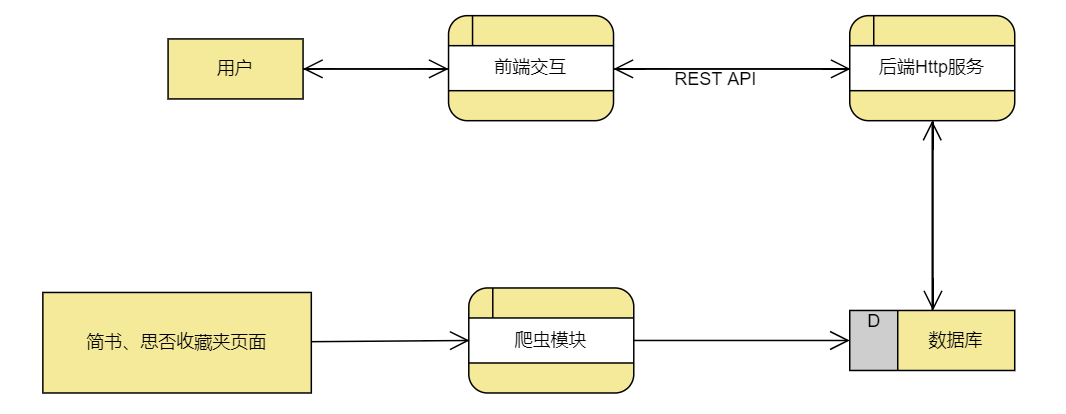


图 3-3 系统数据流图

## 系统数据库设计

本系统数据库采用的是关系数据库postgreSQL，在设计数据库时，通过之前的需求分析，确定了该系统数据库主要包括以下几个实体：用户、收藏的文章、原创文章、评论、回复、点赞、与用户相关的消息。

这些实体之间的关系用E-R图表示如下所示：

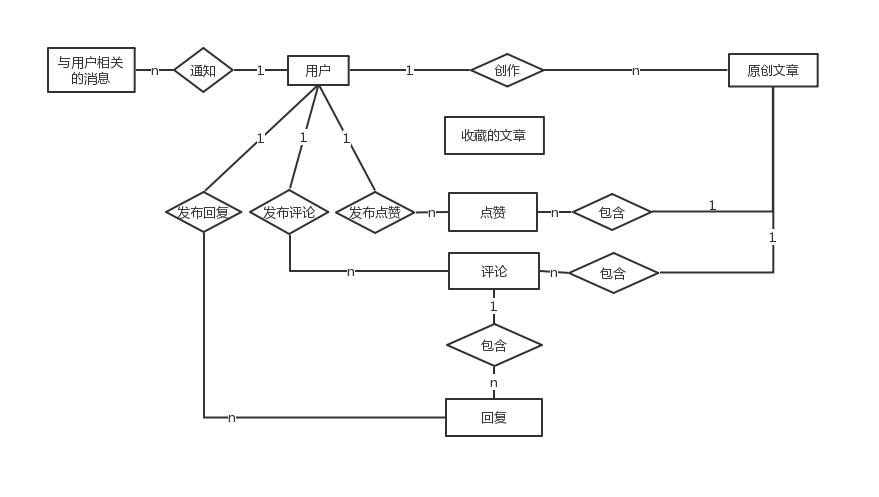


图 3-3 个人博客数据库E-R图

收藏的文章是通过爬虫模块添加到数据库中的，与其他实体之间没有联系。当用户是博主的时候，一个用户可以写多篇原创文章，一篇原创文章的作者只能是一个用户，因此用户与原创文章之间存在一对多（1：n）的联系；一个用户可以发出多个点赞，一个点赞只能属于一个用户，因此用户和点赞之间存在一对多（1：n）的联系；一个用户可以发布多个评论，一个评论的作者只可能是一个用户，因此用户和评论之间存在一对多（1：n）的联系；一个用户可以发布多条回复，一条回复的作者只能是一个用户，因此用户和回复存在一对多（1：n）的联系；一个用户可以收到多条与他相关的消息，一条消息只能发给一个用户，因此用户和消息之间存在一对多（1：n）的联系；一篇原创文章中可以包含多个点赞，而一个点赞只属于一篇原创文章，因此原创文章和点赞存在一对多（1：n）的联系；一篇原创文章可以包含多条评论，一条只能发布在篇原创文章下；因此原创文章和评论之间存在一对多（1：n）的联系；一条评论下可以有多条回复，一条回复只能发布在一条评论下，因此评论和回复存在一对多（1：n）的联系。在创建数据表时，我们需要先创建两个枚举类型，一个是文章种类ARTICLE\_TAG，它标记了收藏的文章是属于什么类型的文章，该类型的取值有：'JS', 'NODE', 'REACT','VUE','WEBPACK','CSS','JAVA','CPP','PYTHON','LINUX','MYSQL','POSTGRESQL','REDIS', 'MONGODB', 'ALGORITHM', 'DATASTRUCTURE','AI'。需要创建的另一个枚举类型是用户角色USER\_ROLE，它标记了用户是博主还是访客，该类型的取值有：'ADMIN','VISITOR'。

经过数据库需求分析和设计后，我们设计出各个实体的数据表如下：

表名：users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| uid | SERIAL | NO | 序列递增值 | 用户id，主键 |
| nickname | VARCHAR(20) | NO |  | 用户名 |
| email | VARCHAR(20) | NO |  | 邮箱 |
| password | VARCHAR(20) | NO |  | 密码 |
| role | USER\_ROLE | NO | ‘VISITOR’ | 用户角色 |

表3-2 用户数据表设计

表名：collected\_articles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| caid | SERIAL | NO | 序列递增值 | 收藏的文章id，主键 |
| tag | ARTICLE\_TAG | YES |  | 文章类别 |
| title | VARCHAR(20) | YES |  | 文章标题 |
| author | VARCHAR(20) | YES |  | 文章作者 |
| content | TEXT | YES |  | 文章内容 |
| original\_link | VARCHAR(50) | YES |  | 原文出处链接 |
| date | DATE | YES | CURRENT\_DATE | 收藏的日期 |

表3-3 收藏的文章数据表设计

表名：original\_articles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| oaid | SERIAL | NO | 序列递增值 | 原创文章id，主键 |
| title | VARCHAR(20) | YES |  | 文章标题 |
| author | INTEGER | YES |  | 文章作者id, 外键 |
| content | TEXT | YES |  | 文章内容 |
| date | DATE | YES | CURRENT\_DATE | 发布的日期 |

表3-4 原创文章数据表设计

表名：comments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| cid | SERIAL | NO | 序列递增值 | 评论id，主键 |
| oaid | VARCHAR(20) | NO |  | 原创文章id, 外键 |
| owner | INTEGER | NO |  | 评论作者id, 外键 |
| content | TEXT | YES |  | 文章内容 |
| date\_time | TIMESTAMP | YES | CURRENT\_TIMESTAMP | 评论发布的时间戳 |

表3-5 评论数据表设计

表名：replys

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| rid | SERIAL | NO | 序列递增值 | 回复id，主键 |
| cid | INTEGER | NO |  | 评论id，外键 |
| owner | INTEGER | NO |  | 回复作者id, 外键 |
| responder | INTEGER | NO |  | 被回复者id，外键 |
| content | TEXT | YES |  | 回复内容 |
| date\_time | TIMESTAMP | YES | CURRENT\_TIMESTAMP | 回复发布的时间戳 |

表3-6 回复数据表设计

表名：likes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| lid | SERIAL | NO | 序列递增值 | 点赞id，主键 |
| oaid | INTEGER | NO |  | 原创文章id，外键 |
| owner | INTEGER | NO |  | 回复作者id, 外键 |
| date\_time | TIMESTAMP | YES | CURRENT\_TIMESTAMP | 回复发布的时间戳 |

表3-7 点赞数据表设计

表名：messages

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 是否可以为空 | 默认值 | 说明 |
| mid | SERIAL | NO | 序列递增值 | 消息id，主键 |
| uid | INTEGER | NO |  | 要通知的用户id，外键 |
| oaid | INTEGER | NO |  | 消息所属的原创文章id, 外键 |
| title | VARCHAR(20) | YES |  | 消息标题 |
| content | TEXT | YES |  | 消息详情 |
| unread | BOOLEAN | NO | TRUE | 是否已读 |
| date\_time | TIMESTAMP | YES | CURRENT\_TIMESTAMP | 消息的时间戳 |

表3-8 通知消息数据表设计

## 后端API设计

因为前端MVVM框架的出现，包括前端页面路由，页面渲染在内的大量的工作可以交给前端完成。在本系统中，后端服务只负责响应RESTfult API，前端和后端通过AJAX进行数据交互，因此API的设计是非常重要的。

本系统的API，传递的数据统一为JSON格式，因此前端发送的http请求需要添加Content-Type：application/json字段。后端返回的数据JSON格式为：

{

succeed: true,

errorCode: -1,

message: null,

data: {

// 响应得到的数据，各个接口各不相同

}

}

API设计满足REST规范，即url代表资源，请求方法代表对资源的操作，其中资源种类包括collected\_articles、original\_articles、users三种，原创文章下评论、点赞、回复资源包含在original\_articles下，与用户相关的消息包含在users下。具体API设计如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **请求方法** | **请求URL** | **查询参数** | **请求内容所需字段** | **说明** |
| GET | /collected\_articles/{tag} | start,num |  | 获取收藏的文章列表 |
| GET | /collected\_articles/a/{caid} |  |  | 获取收藏的文章内容 |
| GET | /original\_articles | start,num |  | 获取原创文章列表 |
| POST |  | title,author,content | 发布原创文章 |
| GET | /original\_articles/{oaid} |  |  | 获取原创文章内容 |
| PUT |  | title,content | 修改原创文章内容 |
| DELETE |  |  | 删除原创文章 |
| GET | /original\_articles/{oaid}/comments | start,num |  | 获取评论列表 |
| POST |  | owner,content | 发布评论 |
| GET | /original\_articles/comments/{cid}/replys | start,num |  | 获取回复列表 |
| POST |  | responder, owner, content | 发表回复 |
| GET | /original\_articles/{oaid}/like |  |  | 获取点赞数 |
| POST |  | owner | 给文章点赞 |
| DELETE |  | owner | 给文章取消点赞 |
| POST | /user/login |  | user, password | 登录接口 |
| POST | /user/register |  | nickname,password,email | 注册接口 |
| GET | /users/{uid}/messages | start,num |  | 获取用户相关消息 |
| POST |  | uid,oaid,title,content, | 添加新消息 |
| PUT | /users/messages/:mid |  |  | 修改消息已读状态 |
| GET | /users{uid}/like/{oaid} |  |  | 获取用户是否给文章点过赞 |

表3-9 后端RESTful API设计

## 前端路由设计

根据不同URL地址来显示不同的内容或页面称之为路由。过去，这一任务交由后端来实现，每次URL变化，后端都要发送一次完整的HTML给前端进行展示。有了AJAX之后，前后端数据交互可以不再刷新页面，路由的映射通常是进行一些DOM的显示和隐藏操作，当访问不同路径的时候，会显示不同的页面组件。这样一来，服务端只用给前端发送一次HTML文件，这样的应用我们称之为SPA(Single Page Application)。

前端路由有两种实现方式，一种是Hash路由。URL中‘#’符号及其后面的部分为Hash，Hash仅仅是客户端的一个状态，也就是说，当向服务器发请求时，Hash部分并不会发过去。通过监听window对象的hashChange事件，就可以实现简单的Hash路由。另一种实现前端路由的方式时利用HTML5的History API，利用它，我们可以在不刷新页面的情况下，直接改变当前URL。

在用React搭建前端页面时，我们可以用官方维护的React-router库实现前端路由，本系统中，我们使用的是基于History Api的路由。

具体设计如下：

|  |  |
| --- | --- |
| **URL** | **说明** |
| / | 首页 |
| /register | 注册页 |
| /front-end/{tag[[3]](#footnote-3)} | 前端收藏夹文章列表页 |
| /front-end/{tag}/{caid} | 前端收藏夹文章内容页 |
| /server/{tag} | 服务端收藏夹文章列表页 |
| /server/{tag}/{caid} | 服务端收藏夹文章内容页 |
| /database/{tag} | 数据库收藏夹文章列表页 |
| /database/{tag}/{caid} | 数据库收藏夹文章内容页 |
| /original/all-articles | 原创文章列表页 |
| /original/all-articles/{oaid} | 原创文章内容页 |
| /original/write-article | 博主编辑原创文章页 |

表3-10 前端路由设计

## 前端Redux store设计

React组件之间的数据传递是单向数据流，即数据只能由上层组件通过props或context属性传递给下层组件。因此要实现两个组件之间的通信，只能通过把公共状态放在这两个组件的同一个祖先组件上实现。结合React的这一特点，React-redux库把各个组件需要通信的数据放在一个顶层组件（Provider组件）的context中，而对这些数据的操作统一由Redux来进行管理。

因此在设计我们的前端工程时，对Redux状态的设计是至关重要的。设计Redux状态时，主要是设计两项，一是存储的数据内容，即state，另一个是对数据可以进行哪些操作，即action。

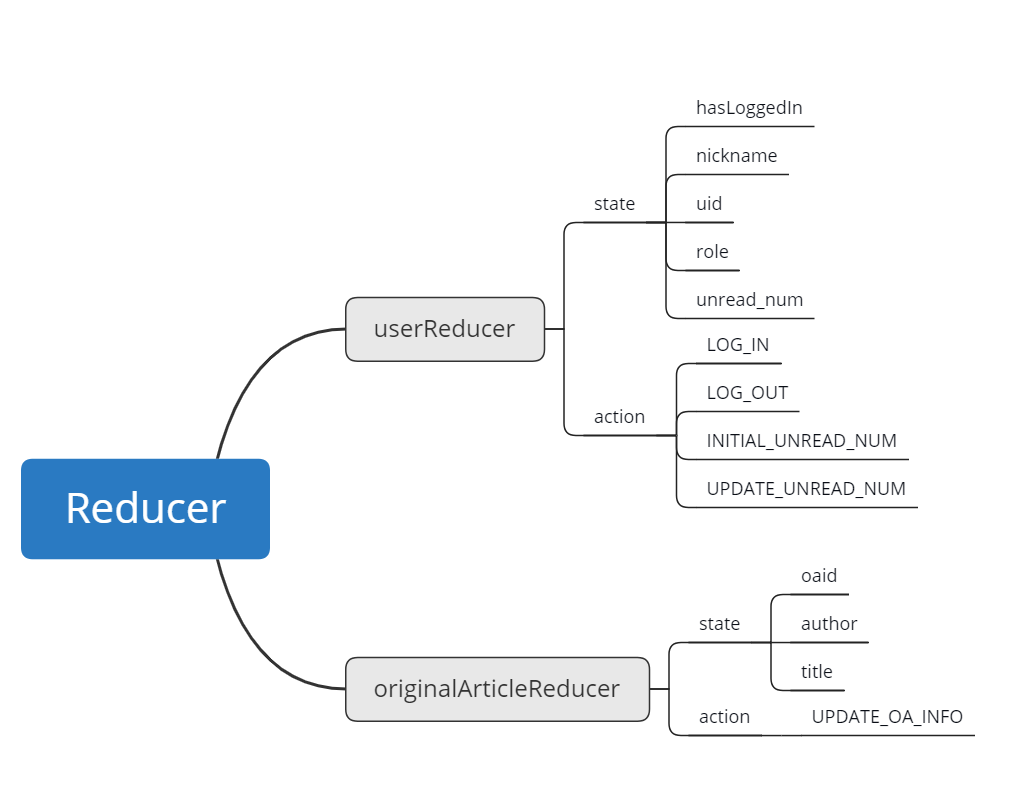
结合前文的需求分析、前端页面架构设计和API设计，我们可以分析出：首先，用户登录后显示用户信息，点赞、发表评论、回复时给后台发送请求以及根据用户是否登录决定是否显示撰写、编辑、删除原创文章的相关组件，实现这三项功能的相关组件需要共享用户id, 用户名，用户角色，用户是否登录这些状态。其次，点赞、发表评论、回复时需要向后台发送添加新通知的请求，请求内容中包含文章标题、文章id，文章作者id，因此点赞、评论、回复组件需要与原创文章组件共享原创文章的这些状态。由此，我们把这些状态放在redux的store中进行管理，设计出如下reducer[[4]](#footnote-4)：

图3-4 前端redux store设计

如图所示，userReducer中包含的状态从上到下依次是：用户的登录状态、用户名、用户id、用户角色、用户的未读消息数；对这些状态的操作有：LOG\_IN(将hasLoggedIn更改为true，同时将nickname、uid、role更改为传入的值，用于登录)、LOG\_OUT(将hasLoggedIn更改为false，用于退出登录)，INITIAL\_UNREAD\_NUM(将unread\_num更改为传入的值，用于在页面初次挂载时初始化用户未读消息数)，UPDATE\_UNREAD\_NUM(将unread\_num减少一，用于在用户点击了未读消息后更新未读消息数)。OriginalArticleReducer包含的状态从上到下依次是：原创文章id、原创文章作者id、原创文章标题；对这些状态的操作有：UPDATE\_OA\_INFO（将这三个状态都修改为传入的新值）。

# 详细设计

## 爬虫模块设计

爬虫模块需要分别爬取我的简书收藏夹和思否收藏夹，然后将爬取到的文章存入数据库中。首先我们需要做的准备工作是，分别在简书和思否两个网站上，都创建如下的收藏夹：JS、NODE、REACT、VUE、WEBPACK、CSS、JAVA、CPP、PYTHON、LINUX、MYSQL、POSTGRESQL、REDIS、MONGODB、ALGORITHM、DATASTRUCTURE、AI。本模块下分三个子模块，分别是简书爬虫模块，思否爬虫模块和日志模块，前两个模块采用request库发起HTTP请求获取页面HTML，然后用cheerio库，用类jQuery的API非常方便地提取页面信息。在爬虫过程中，如果出现错误，日志模块会记录错误信息到日志文件中，有新文章入库成功也会有对应的日志记录，方便日后查看。

简书爬虫模块和思否爬虫模块只有爬取具体页面的时候有所不同，其他流程有重复，因此我采用了面向对象编程，编写了Crawler类，然后JianshuCrawler和SegmentFaultCrawler继承Crawler类并实现相关接口。

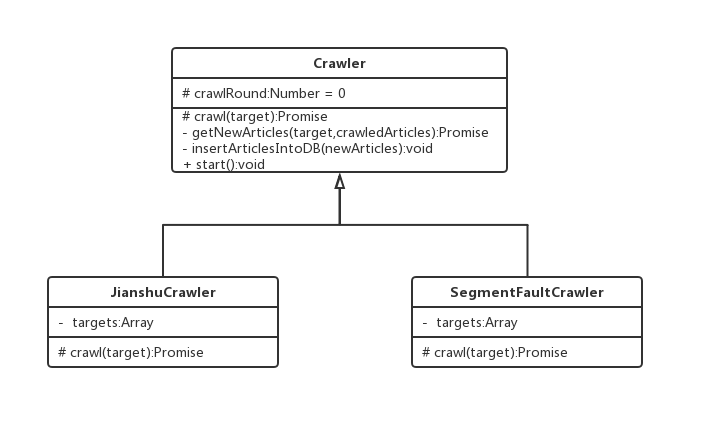


图4-1 爬虫模块UML图

爬虫程序执行流程如下：

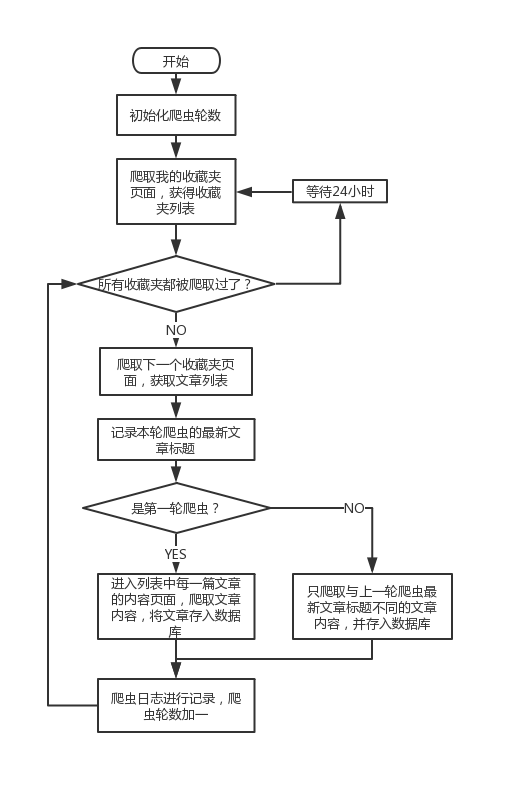


图4-2 爬虫模块流程图

这样，该模块就实现了每隔24小时爬取一次我的简书和思否中所有收藏夹，并将新收藏的文章存入数据库的功能

## 后端功能模块设计

后端需要完成的是数据库操作和路由转发两大功能模块。

### 数据库操作模块设计

本系统后端使用开源的pg-promise库连接PostgreSQL，要返回正确的数据，并不需要我们处理太多的逻辑，大多都是直接执行SQL语句。只有登录和注册功能需要简单的校验处理。

登录校验处理流程：

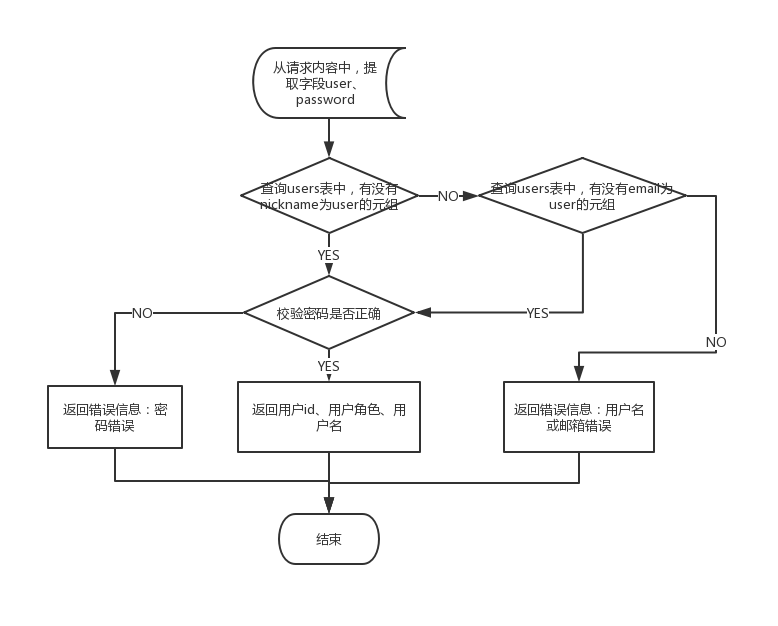


图4-3 登录模块流程图

注册校验处理流程：

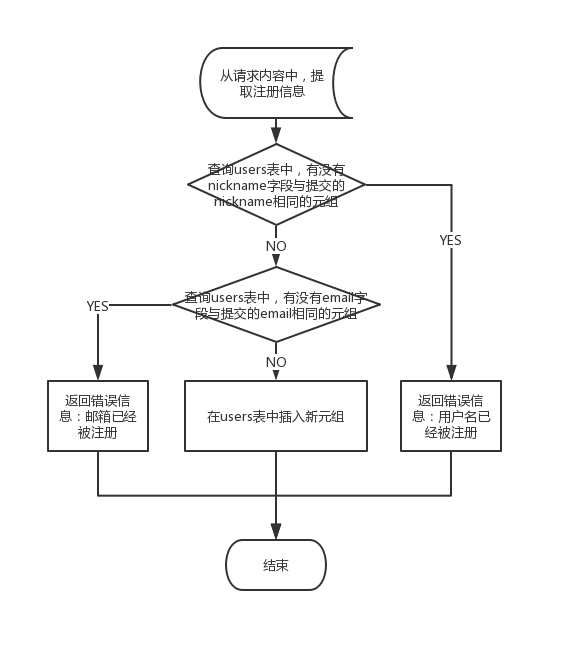


图4-4 注册校验功能

将这些对数据库的操作写成类中的静态方法，即完成了后端数据库操作模块，供路由转发模块调用。

### 路由转发模块设计

后端路由转发就是根据http请求报文的URL，调用相应的处理模块。我们的后端是用Node.js搭建的HTTP服务，Node.js内置了http模块，可以对http报文进行收和发。但是http模块相对底层，因此我采用了Node.js中最流行的express框架，利用express的router模块进行路由转发。根据API设计，路由转发主要转发到三个模块，即collected\_articles，original\_articles，users, 因此我写了三个router，根据设计好的API和数据库操作模块，将URL与对应的数据库操作一一映射即可。

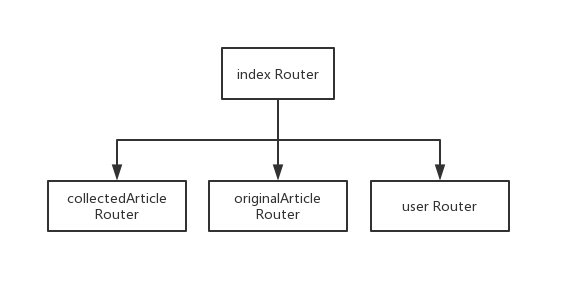


图4-5 路由转发模块划分图

## 前端功能模块设计

### 登录模块组件设计

登录模块总组件下分两个组件，一个是登录信息输入框组件，一个是登录成功后，展示用户信息的组件，显示哪个组件由store中的state.user.hasLoggedIn

决定，登录模块总组件挂载时，会从localStorage中查看是否有用户信息，如果有，则直接更新登录状态为已登录，并显示用户信息，同时向后端发起请求获取最新的未读消息数。用户未登录时，显示登录信息输入框组件。用户在登录信息提示框中输入用户名和密码，点击登录按钮后，向后端发送登录请求，如果登录失败，则显示错误信息，如果登录成功，则将用户信息存入localStorage中，方便用户下一次访问时实现自动登录，同时将store中的用户信息更新为后端返回的用户信息，并更新store中的用户登录状态为已登录，这时就会隐藏登录信息输入框组件，显示用户信息组件，同时向后端发起请求获取用户的最新未读消息数，并更新到store中。



图4-6 登录框组件

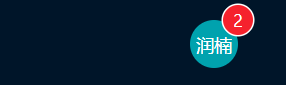


图4-7 用户信息组件

### 文章列表与内容模块组件设计

文章列表组件在挂载阶段向后台发送请求获取起始序号为1，数量为10一组的文章标题列表展示出来，当点击第n页的时候（点击下一页或上一页时，组件内部计算好是第几页），向后台发送请求获取起始序号为n\*10+1，数量为10一组的文章标题列表，更新展示出来。当点击文章标题时，跳转到文章内容页面，如果是收藏的文章，则直接在文章内容组件挂载阶段向后台发起请求获取文章内容并展示，如果是原创文章，则还涉及点赞、评论、回复等组件，将在下文中详细介绍。



图4-8 文章列表组件



图4-9 文章内容组件

### 点赞模块组件设计

当用户未登录时（从store中获取用户是否登录），点赞组件只显示文章点赞数，点击点赞按钮，弹出信息：登录后才可以点赞。当用户已经登录时，原创文章内容组件在挂载阶段向后端发起请求，获取用户是否给文章点过赞，如果点过，则点赞按钮为实心，点击该按钮时，向后台发送取消点赞请求，后台返回取消点赞成功的响应后，点赞数减一，点赞按钮变空心。如果用户没有点过赞，则点赞按钮为空心，再点击按钮时，向后台发送点赞请求，后台返回点赞成功的响应后，点赞数加一，点赞按钮变实心。

图4-10 点赞组件（点赞状态和取消点赞状态）

### 评论模块组件设计

评论模块包含评论列表组件和评论输入框组件，二者均挂载在原创文章内容组件下。评论列表同文章列表组件逻辑相似，只是向后台请求数据的是文章下的评论。评论输入框组件显示已登录用户的用户名和一个输入框，监听输入框的onChange事件，当输入的评论内容超过200字时，显示提示信息：评论内容不能超过两百字。当点击发布评论按钮时，向后台发送两个POST请求，一个是插入新评论的请求，请求报文中所需的原创文章id从store的originalArticle模块获取，评论作者的id，也就是已登录用户的id，从store的user模块获取；另一个是给文章作者添加新消息通知的请求，请求内容中的原创文章标题、文章id、作者id从store的originalArticle模块获取，评论者id从store的user模块获取。从当后台返回评论成功的响应后，显示提示信息：评论成功，刷新页面后可见。



图4-11 评论列表组件

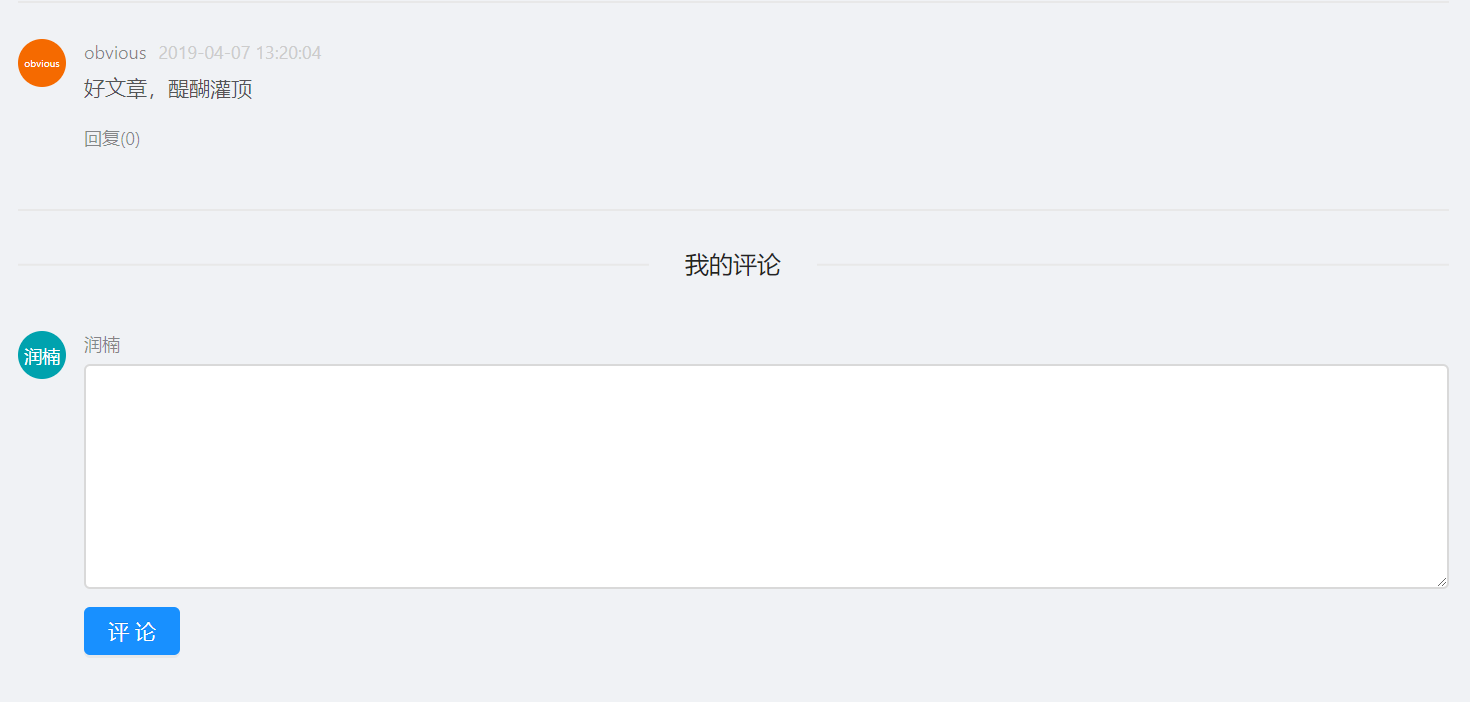


图4-12 评论输入框组件

### 回复模块组件设计

回复模块包含回复列表组件和回复输入框组件，在评论列表的每个评论项中有一个回复按钮，当用户未登录时，单击该按钮，只显示评论列表，不显示评论输入框。评论列表组件的实现逻辑同文章列表和评论列表组件类似，只是向后台请求的是评论下的回复，请求回复时发送的报文需要评论的id，这一数据是评论项组件直接通过props传递给回复列表组件的。当用户已经登录时，单击评论项下的回复按钮，会在评论列表下显示回复输入框组件，同时把评论的作者id通过props传给该组件。当用户直接在回复框内输入回复内容并点击发布回复按钮时，向后台发送插入新回复的请求，请求内容中的responder字段，即被回复者，就是上层组件传入的评论者id。如果在回复列表中点击了列表项的回复按钮，则评论输入框内会自动添加“@”后跟被回复者的用户名，同时在点击发布回复按钮时，向后台发送插入新回复的请求，请求内容中的responder字段，会变成刚才点击的回复项的作者id。发送了插入新回复的请求后，还要分别给文章作者、评论作者、被回复者、发送插入新消息通知的请求，请求报文内容中所需的字段也是从store中获取。当后台返回回复成功的响应后，显示提示信息：回复成功，刷新页面后可见。



图4-13 回复列表和回复输入框组件

### 消息通知模块组件设计

消息通知列表组件主要是显示与用户相关的消息，对访客来说，包括自己评论下的回复和@自己的回复；对于博主，还包括博主所写的原创文章下的评论、回复、点赞。消息通知组件挂载阶段向后台请求消息列表，对于用户未读的消息，会有红点提醒，当点击查看消息的链接后，向后台发送将消息设置为已读的请求，同时将store的user模块中，用户的未读消息数减一。



图4-14 消息列表组件

# 系统测试

## 测试目的

1. 软件测试是为了发现错误而执行程序的过程。
2. 测试是为了证明程序有错，而不是证明程序无错。（发现错误不是唯一目的）
3. 一个好的测试用例在于它发现至今未发现的错误。
4. 一个成功的测试是发现了至今未发现的错误的测试。

## 测试方法

### 单元测试

单元测试是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

在一种传统的结构化编程语言中，比如C，要进行测试的单元一般是函数或者子过程。在像C++这样的面向对象的语言中， 要进行测试的基本单元是类。对Ada语言来说，开发人员可以选择是在独立的过程和函数，还是在Ada包的级别上进行单元测试。单元测试的原则同样被扩展到第四代语言（4GL)的开发中，在这里基本单元被典型地划分为一个菜单或显示界面。

经常与单元测试联系起来的另外一些开发活动包括代码走读（Code review)，静态分析（Static analysis)和动态分析（Dynamic analysis)。静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。动态分析就是通过观察软件运行时的动作，来提供执行跟踪，时间分析，以及测试覆盖度方面的信息。

本系统中，我们主要对后端数据库操作模块的每一个函数进行单元测试，确保查询出的数据是正确的。

### 黑盒测试

黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。在测试中，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

黑盒测试是以用户的角度，从输入数据与输出数据的对应关系出发进行测试的。很明显，如果外部特性本身设计有问题或规格说明的规定有误，用黑盒测试方法是发现不了的。

## 测试样例

### 单元测试

本系统单元测试采用了业界比较流行的javaScript单元测试框架Jest.js，测试的是后端数据库操作模块BlogDB，该模块包含的是一系列操作数据库的静态方法，我们在单元测试中，依次将这些方法放入测试框架中运行即可。

例如，测试函数getUserInfoByNickname，代码如下：

describe('测试函数 getUserInfoByNickname ',()=>{

test('根据用户名查找一个不存在的用户',()=>{

expect.assertions(1); // 确保至少有一个断言被调用，否则测试失败

return db.getUserInfoByNickname('闰南').then(data=>{

expect(data).toBe(null);

})

});

test('根据用户名查找一个存在的用户',()=>{

expect.assertions(1); // 确保至少有一个断言被调用，否则测试失败

const expectedData = {

uid: 1,

nickname: '润楠',

password: 'smielpf1204.',

role: 'ADMIN',

email: '1608272694@qq.com'

};

return db.getUserInfoByNickname('润楠').then(data=>{

expect(JSON.stringify(data)).toBe(JSON.stringify(expectedData));

})

})

})；

如果测试样例通过，则控制台显示如下：

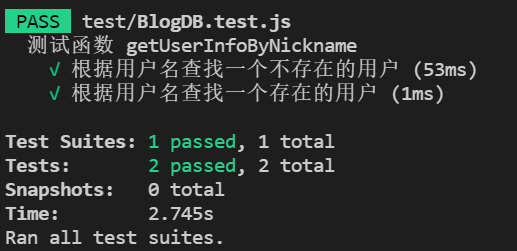


图5-1 单元测试控制台显示结果图

### 黑盒测试

本系统黑盒测试过程就是运行并模仿用户操作使用系统，看是否符合预期效果，例如，登录模块的测试样例设计如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试模块** | **用例描述** | **操作过程** | **预期结果** | **实际结果** | **偏差** |
| 登录模块 | 实现用户登录 | 输入一个已注册用户的用户名/邮箱和正确的密码 | 登录输入框消失，显示用户头像和用户未读消息数，进入原创文章页面，未出现“写文章”菜单项 | 如预期显示 | 无 |
| 实现管理员登录 | 输入一个已注册管理员的用户名/邮箱和正确的密码 | 登录输入框消失，显示用户头像和用户未读消息数，进入原创文章页面，出现“写文章”菜单项 | 如预期显示 | 无 |
| 处理登录信息异常 | 输入一个未注册用户的用户名和密码 | 显示“用户名或密码错误”的气泡提示信息 | 如预期显示 | 无 |
| 输入一个已注册用户的用户名/邮箱和错误密码 | 显示“用户名或密码错误”的气泡提示信息 | 如预期显示 | 无 |

表5-1 登录模块测试用例

# 总结与展望

## 总结

本论文针对我想要定制个人博客页面的同时能查看我在简书和思否上的收藏夹的需求，介绍了一个基于Node.js和React技术栈的个人博客系统的设计与实现。

具体工作内容总结如下：

1. 介绍了当前大多数个人博客系统存在的问题与我的需求之间的冲突，提出了开发该系统的意义。
2. 阐述了项目开发用到的Node.js、React.js、PostgreSQL相关技术栈以及本系统采用这些技术栈的原因。
3. 分析了系统的功能需求，阐述了系统总体架构设计。
4. 详细介绍了各个功能模块的具体设计与实现方法。
5. 阐述了系统测试的目的，方法，并详细介绍了几个典型的测试样例。

## 展望

本系统已经满足了我的基本需求，但是仍然有许多需要改进的地方：

1. 跨浏览器兼容改进：本系统前端部分仅仅实现了在较新版本的PC端Chrome、Firefox等浏览器上的正确显示，还需要对其他多版本浏览器以及移动端浏览器设备进行适配。
2. 安全改进：本系统登录注册功能实现得比较简单，数据库中保存的以及通过HTTP传输的都是用户的明文密码，安全性较低。同时还需要在检查系统是否存在SQL注入，XSS攻击等问题，并加以改进。
3. 部署改进：本系统服务端目前只是运行单个Node.js进程，可靠性较低，能应对的访问量也较少，未来可以考虑通过K8S部署在集群上。
4. SEO改进：撰写了个人博客也需要被更多人看到，但是目前该系统在SEO方面基本没有优化，因此未来应该考虑针对SEO进行优化，让该系统上的文章在搜索引擎上能有相对较好的排名。

# 参考文献：

1. Nicholas C.Zakas. Professional JavaScript for Web Developers [M].3rd edition. 北京：人民邮电出版社. 2012.
2. Nicholas C.Zakas. High Performance JavaScript [M]. 北京：电子工业出版社. 2015.
3. David Gourley, Brain Totty, Marjorie Sayer, Sailu Reedy, Anshu Aggarwal. Http: The Definitive Guide [M]. 北京：人民邮电出版社. 2012.
4. 王珊，萨师煊. 数据库系统概论 [M]. 第五版. 北京：高等教育出版社. 2014.
5. 朴灵. 深入浅出Node.js [M]. 北京：人民邮电出版社. 2013.
6. 骆文亮. Node.js服务器技术初探[J]. 无线互联科技,2014(03):227.

# 致谢

大学本科四年以这次毕业设计画上了圆满的句号，在此，我要感谢大学四年间给予过我帮助与关怀的每一位同学和老师，特别是感谢我的论文指导老师温武少老师，感谢他对我的毕业设计给予的悉心指导，也感谢他对我生活和未来前途提出的建议，让我获益匪浅。

同时也要感谢我实习过的华为、Bigo公司，是在这两家公司实习期间让我学到了很多东西，拓展了视野，让我在技术上，与人交流的技巧上都有了长足的进步。

最后，感谢我的父母为我的学业提供的无条件的支持，是他们在我迷茫无助的时候给予我鼓励和陪伴，他们永远是我最亲最爱的人。

衷心祝愿每一个帮助、支持我的人身体健康，生活顺利。

1. 收藏夹标签：前端、服务端、数据库、其他 [↑](#footnote-ref-1)
2. 文章标签：前端收藏夹下的文章标签有javaScript、Node、React、Vue、Webpack、CSS；后端收藏夹的文章标签有java、cpp、python、linux；数据库收藏夹下的文章标签有mysql、postgreSQL、redis、mongodb；其他收藏夹下的文章标签有：算法、数据结构、人工智能

   [↑](#footnote-ref-2)
3. 这里的tag取值即数据库设计中提及的文章类别标签 [↑](#footnote-ref-3)
4. reducer是redux库的一个函数，用于定义数据和对数据的操作，具体用法参考redux相关资料 [↑](#footnote-ref-4)