

河 南 科 技 学 院

2022 届本科毕业论文（设计）

基于 Golang+Vue 的博客论坛的设计与实现

学 号:	20191544119
姓 名:	胡超
专 业:	通信工程
学 院:	信息工程学院
指导教师:	蔡磊(教授)
完成时间:	2023 年 5 月 4 日

## 摘 要

随着网络时代的快速进步，博客论坛作为一个重要的信息分享与交流平台，已成为许多人学习、工作和生活的一部分。为满足现代博客论坛的需求，本文设计并实现了一个基于 Golang+Vue 的博客论坛系统。系统采用前后端分离架构，后端使用 Golang 编程语言，结合 Gin 框架实现 RESTful API 接口，数据存储方面选择 MySQL 作为关系型数据库，并利用 Redis 进行缓存处理。前端部分则采用 Vue.js 框架，搭配 Element UI 库完成用户界面设计。

本系统包含多个功能模块，如用户登录注册、发表帖子、帖子列表、帖子点赞、社区分类、帖子评论以及实时展示 Github 热门项目等。这些功能模块的设计与实现，使得本博客论坛系统具有较高的可用性和易用性，满足了用户在社区内交流、分享和学习的需求。

关键词：Golang, Vue.js, Gin, MySQL, Redis

## **ABSTRACT**

**In the rapidly progressing era of the internet, blog forums, as an important platform for information sharing and communication, have become an integral part of many people's learning, work, and life. To meet the demands of modern blog forums, this paper designs and implements a blog forum system based on Golang+Vue. The system adopts a front-end and back-end separation architecture. The back-end uses the Golang programming language and combines the Gin framework to implement RESTful API interfaces. MySQL is chosen as the relational database for data storage, and Redis is used for cache processing. The front-end uses the Vue.js framework, combined with the Element UI library to complete the user interface design.**

**The system includes multiple functional modules, such as user login and registration, post publishing, post lists, post likes, community classification, post comments, and real-time display of popular Github projects. The design and implementation of these functional modules make the blog forum system highly available and easy to use, meeting the needs of users for communication, sharing, and learning in the community.**

**Keywords: Golang; Vue.js; Gin; MySQL; Redis**

# 目 录

1 绪论 .....	1
1.1 课题背景及意义 .....	1
1.2 研究现状 .....	1
2 技术支持 .....	2
2.1 Gin 框架 .....	2
2.2 MySQL 数据库 .....	2
2.3 Redis .....	2
2.4 JWT .....	2
2.5 Docker .....	3
2.6 Vue 框架 .....	3
2.7 前后端分离 .....	3
2.8 B/S 架构 .....	错误! 未定义书签。
3 网站设计 .....	4
3.1 网站总体架构 .....	4
3.2 功能模块设计 .....	5
3.3 数据库设计 .....	6
4 论坛前端 .....	7
4.1 注册 .....	7
4.2 登录 .....	8
4.3 首页 .....	9
4.4 博主相关 .....	9
4.4.1 访问量 .....	9
4.4.2 博主信息 .....	10
4.4.3 留言 .....	12
4.4.4 网站运行时间 .....	12
4.5 帖子相关 .....	13
4.5.1 列表 .....	13
4.5.2 发表 .....	16
4.5.3 详情 .....	16

4.5.4 评论 .....	17
4.6 社区相关 .....	18
4.7 三方接口相关 .....	19
4.7.1 每日一言 .....	19
4.7.2 Github 热门项目榜 .....	20
4.7.3 Golang 热门项目榜 .....	21
5 论坛后端 .....	22
5.1 JWT 认证 .....	22
5.2 分布式 ID: 雪花算法 .....	24
5.3 配置文件管理: viper .....	25
5.4 参数校验: validator .....	26
5.5 日志库: zap .....	29
5.6 令牌桶限流 .....	32
5.7 投票排名算法 .....	33
6 网站部署 .....	36
6.1 编写 Dockerfile .....	36
6.2 构建 Docker 镜像 .....	37
6.3 运行 Docker 容器 .....	37
6.4 访问网站应用 .....	37
7 结论 .....	39
参考文献 .....	40
致谢 .....	41

# 1 绪论

## 1.1 课题背景及意义

随着 Internet 技术的进步和移动互联的广泛应用，人们对信息的获取，思想的交流和知识的共享也越来越迫切。博客论坛作为一种典型的网络社区平台，已经成为许多人日常学习、工作和生活的重要组成部分。传统的博客论坛系统在功能、性能和用户体验方面已不能满足现代社交网络的需求，因此，开发一个高效、易用且具备较强扩展性的博客论坛系统具有重要的现实意义。

本课题主要研究如何基于 Golang 和 Vue.js 技术栈设计和实现一个功能完善、性能优越的博客论坛系统。Golang 作为一种现代化的编程语言，具有简洁的语法、高效的性能和强大的并发处理能力，适用于构建高性能的后端服务。Vue.js 是一种轻量级的前端框架，以组件化的开发模式提高了代码的可维护性和可复用性，同时简化了用户界面的设计与实现。通过将这两种技术结合应用在博客论坛系统的开发中，有助于提升系统的性能、安全性和易用性，满足用户对博客论坛的多样化需求。

此外，本课题还涉及到如何利用 Gin 框架构建 RESTful API 接口、如何使用 MySQL 进行数据存储以及如何借助 Redis 实现缓存处理等技术问题。通过对这些技术的深入研究和实践，不仅可以提高通信工程专业的同学们在计算机领域的技能水平，还可以为今后相关领域的研究和开发提供有益的借鉴和参考。

## 1.2 研究现状

随着互联网的普及与发展，博客论坛系统在全球范围内取得了广泛的关注和研究。国内外研究者针对博客论坛系统的设计与实现进行了大量的探讨，提出了许多新颖的理念和技术。

在国外研究方面，一些著名的博客论坛系统如 WordPress、Drupal 和 Joomla 等，已经发展成为具有成熟技术体系和庞大用户群体的开源项目。这些系统在设计理念、功能模块、性能优化等方面具有较高的参考价值。此外，国外研究者还针对博客论坛系统中的各类问题提出了许多解决方案，如搜索引擎优化、个性化推荐算法、内容安全与审查机制等，为博客论坛的持续发展提供了支持。

国内研究方面，随着我国互联网产业的快速崛起，国内研究者在博客论坛系统的设计与实现领域取得了显著的成果。一些知名的国产博客论坛系统如 CSDN、博客园、简书等，已经积累了丰富的实践经验和技术成果。此外，国内研究者还关注博客论坛系统在本土化语境下的特殊需求，如对中文分词技术、网络言论自由与审查制度等方面的研究，为我国博客论坛系统的发展提供了理论支持和技术

基础。

总的来说，无论是国内还是国外，博客论坛系统的研究取得了丰硕的成果。然而，随着技术的不断发展和用户需求的日益多样化，博客论坛系统依然面临着诸多挑战。如何在这个背景下，利用现代化的技术栈如 Golang 和 Vue.js，设计和实现一个具有竞争力的博客论坛系统，仍然具有很高的研究价值和实际意义。

## 2 技术支持

### 2.1 Gin 框架

框架 Gin 框架是一款基于 Golang 编程语言的高性能、轻量级的 Web 应用开发框架。自从发布以来，Gin 框架已经逐渐成为 Golang Web 开发领域中广受欢迎的选择。其优势在于简洁的 API 设计、高性能的处理能力以及对中间件的支持，为开发者提供了一个便捷、灵活的 Web 开发工具。Gin 框架因其简洁、高性能和易于使用而受到许多开发者的欢迎。

### 2.2 MySQL 数据库

MySQL 数据库是一款开源的关系型数据库管理系统，自 1995 年发布以来，已逐渐成为全球广泛使用的数据库解决方案之一。基于其高性能、稳定性、易用性和跨平台支持等特点，MySQL 数据库在 Web 应用、企业信息系统、数据仓库等领域得到了广泛应用。此外，MySQL 数据库提供了多种高可用性解决方案，如主从复制、集群等，以实现数据库的容错和负载均衡。在众多应用领域中，MySQL 数据库已成为值得信赖的数据库解决方案。

### 2.3 Redis

Redis (Remote Dictionary Server, 远程字典服务器) 是一款开源的，基于内存的高性能键值数据库。它支持多种数据结构，如字符串、列表、集合、有序集合、哈希表等。由于其高速的读写能力和丰富的数据结构类型，Redis 在分布式缓存、消息队列、计数器、排行榜等应用场景中具有广泛的应用。Redis 作为一个重要的技术支持手段，负责存储和管理部分数据。例如，利用 Redis 的缓存功能，可以有效减轻数据库的压力，提升网站的访问速度。同时，利用 Redis 的消息队列功能，可以实现异步处理，提高系统的响应能力。此外，Redis 还可以用于实现排行榜、实时计数器等功能，增强博客论坛的交互性和用户体验。

### 2.4 JWT

JSON Web Token (JWT) 是一种在参与方之间进行信息的安全传送与认证的

开放式标准（RFC7519）。JWT 通常应用于身份验证和授权场景，例如，在用户登录一个 Web 应用后，服务器会生成一个 JWT，并将其返回给客户端。之后，客户端在每次访问受保护资源时，都会携带这个 JWT 作为访问凭证。服务器接收到请求后，会验证 JWT 的有效性，并根据其携带的信息决定是否授权访问。在现代 Web 应用和分布式系统中，JWT 作为身份验证和授权的常用手段，具有广泛的实践应用价值。

## 2.5 Docker

Docker 是一种开源的容器化技术，它允许开发人员将应用程序及其依赖项打包到一个轻量级、可移植的容器中。这种容器可以在任何支持 Docker 的系统上无缝运行，消除了因环境差异导致的“在我机器上可以运行”的问题。Docker 的核心概念包括镜像（Image）、容器（Container）和仓库（Repository）。通过编写 Dockerfile，将 Golang 后端服务、Vue 前端应用以及相关依赖项打包成一个 Docker 镜像。之后，可以将该镜像推送到 Docker 仓库，以便在目标服务器上拉取并运行。借助 Docker Compose，还可以同时管理多个服务（如数据库、缓存等），实现一键部署的便捷体验。这种方式不仅大大减少了部署的复杂性，还有助于实现快速迭代和持续集成，提高开发和运维的效率。

## 2.6 Vue 框架

Vue.js 是一款轻量级、易上手的 JavaScript 框架，用于构建现代化的前端应用。它采用数据驱动和组件化的思想，使得开发人员能够以声明式的方式构建用户界面。Vue.js 具有双向数据绑定、虚拟 DOM、模板语法、计算属性、事件处理等特性，帮助开发人员快速构建高性能、可维护的前端应用程序。在项目中，通过将功能模块划分为可复用的组件，开发人员可以更加高效地编写和维护代码。同时，借助 Vue.js 的生态系统（如 Vue Router、Vuex 等），可以轻松地实现前端路由、状态管理等高级功能。

## 2.7 前后端分离

前后端分离是一种现代 Web 应用的架构模式，通过将用户界面（前端）与业务逻辑、数据处理（后端）进行分离，实现开发团队的解耦合与协作，提高开发效率和应用的可维护性。随着 Web 技术的发展，特别是 HTML5、CSS3 和 JavaScript 等前端技术的普及，以及 RESTful API、微服务等后端技术的推广，前后端分离架构逐渐成为主流。总的来说，前后端分离架构为 Web 应用开发带来了诸多优势，有助于提高开发效率、应用性能和可维护性，已成为当今 Web 应用开发的主流模式。



### 3 网站设计

#### 3.1 网站总体架构

本项目采用前后端分离架构体系，基于 Golang 和 Vue 技术为基础。后端使用 Golang 语言，借助 Gin 框架实现 RESTful API，通过 sqlx 库访问数据库。前端使用 Vue 框架搭建，引入 Element UI 库实现界面设计。后端关注业务逻辑处理、数据存储，前端关注用户体验、数据展示。该体系结构使得系统的可维护性、可扩展性和可读性得到了极大的改善，很好地满足网站的需要。如图 3-1 所示。

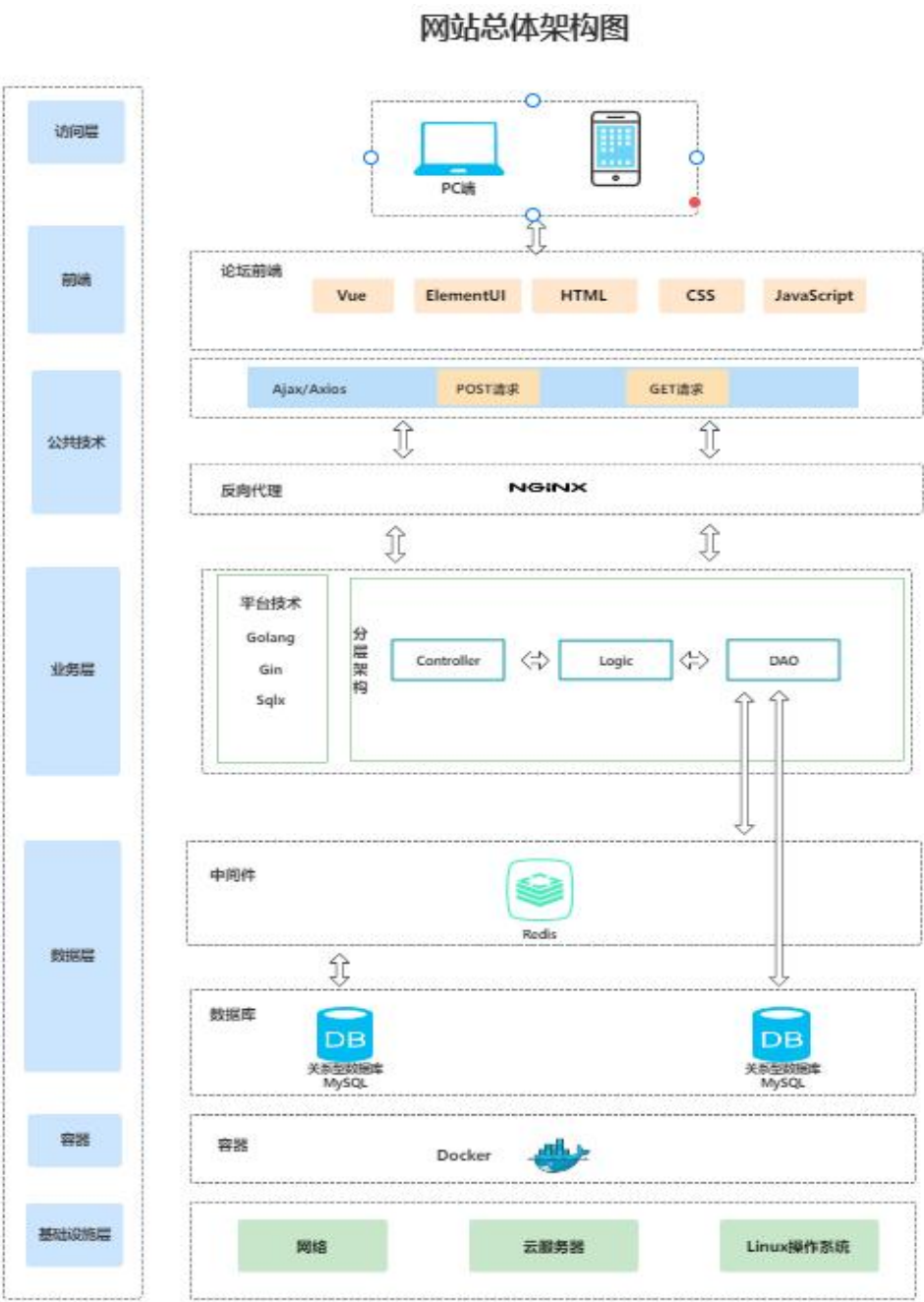


图 3-1 网站总体架构设计图

### 3.2 功能模块设计

在本项目中，设计了以下功能模块以满足不同用户需求：

1. 登录注册：用户可以通过填写用户名、邮箱、密码和性别来完成注册。已注册用户可使用用户名和密码登录系统。
  2. 发表帖子：登录后的用户可以发表文章，包括选择所属社区、输入文章标题和内容。系统提供发帖规范，以确保文章质量。
  3. 帖子列表：系统支持根据时间和分数对帖子进行排序，用户可选择查看最新或最热门的帖子。同时支持分页功能，方便用户浏览。
  4. 帖子点赞：用户可以为感兴趣的帖子点赞，提高帖子在社区内的影响力。
  5. 帖子评论：用户在查看帖子详情时，可以参与到评论区的讨论，在喜欢的帖子下面发表自己的评论。评论功能支持回复他人评论，使得讨论能够更有针对性和深入。
  6. 社区分类：文章按照社区进行分类，用户可以查看特定社区下的文章。社区详情页面展示了社区名称、描述和创建时间等信息。
  7. Github 热榜：网站右侧组件实时展示 Github 热门项目排行榜，根据不同语言分类，如 Golang 热门项目榜。用户可点击加载更多按钮查看更多项目。
- 通过这些功能模块的设计和实现，项目为用户提供了一个便捷、高效的博客论坛平台，满足了用户在社区内交流、分享和学习的需求。
- 如图 3-2 所示。

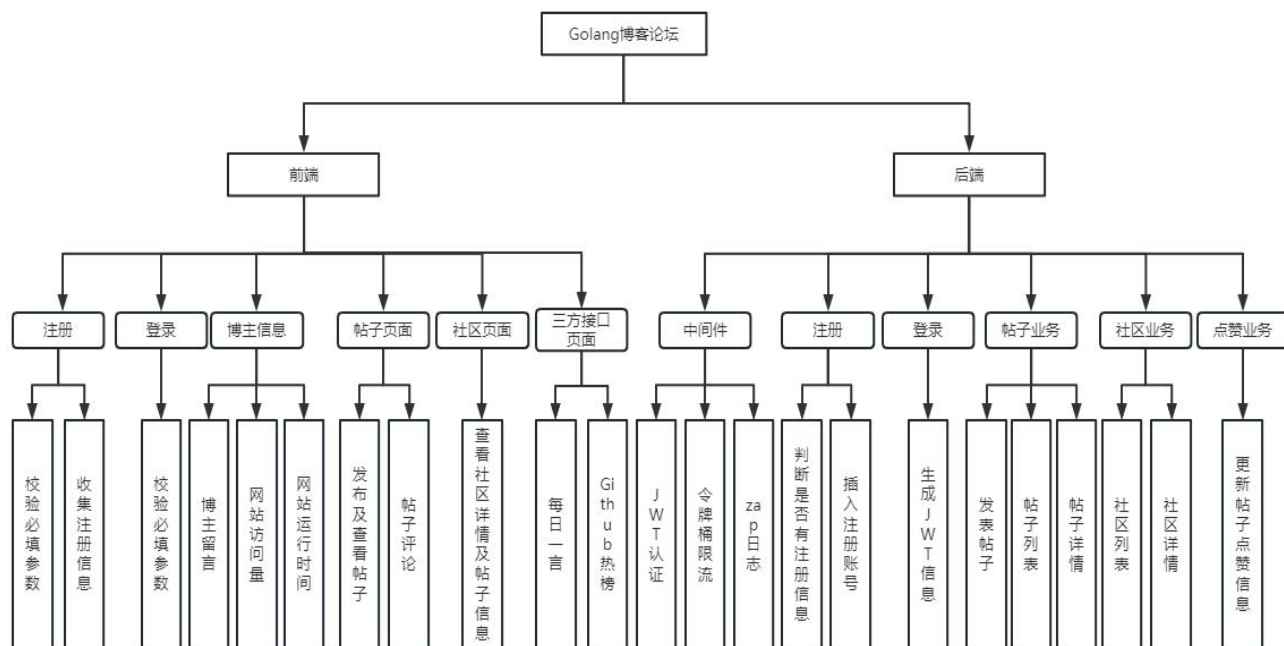


图 3-2 前后端功能模块设计图

### 3.3 数据库设计

数据库是按数据结构来存储和管理数据的计算机系统,是应用程序的根基和软件设计的起点,数据库的功能是组织、存储、管理数据,使信息系统可以方便准确地获取信息。下面列出数据库中的用户表、社区表和帖子表。

用户表的设计信息如表 3.1 所示,在数据库表主要包含主键 id (自增)、用户 id (唯一)、用户名 (唯一)、密码 (加密)、邮箱、性别、创建时间、更新时间等基本情况。

表 3.1 快递详情表 —— user

属性	类型	是否为空	长度	描述
id	int	NOT NULL	11	主键, 自增
user_id	bigint	NOT NULL		用户 id, 唯一
username	varchar	NOT NULL	128	用户名, 唯一
password	varchar	NOT NULL	128	密码, 加密
email	varchar	NULL	128	邮箱
gender	tinyint	Default 0	1	性别
create_time	timestamp	CURRENT_TIMESTAMP		创建时间
update_time	timestamp	CURRENT_TIMESTAMP		更新时间

帖子表的设计信息如表 3.2 所示,在数据库表中主要包含主键 id (自增)、帖子 id (唯一)、标题、内容、作者 id (唯一)、所属社区 id (唯一)、帖子状态、创建时间和更新时间字段等基本信息。

表 3.2 帖子表 —— post

属性	类型	是否为空	长度	描述
id	int	NOT NULL	11	主键, 自增
post_id	bigint	NOT NULL		帖子 id, 唯一
title	varchar	NOT NULL	128	标题
content	varchar	NOT NULL		内容
author_id	bigint	NOT NULL		作者 id, 唯一
community_id	bigint	NOT NULL		社区 id, 唯一
status	tinyint	Default 1	1	帖子状态
create_time	timestamp	CURRENT_TIMESTAMP		创建时间
update_time	timestamp	CURRENT_TIMESTAMP		更新时间

社区表的设计信息如表 3.3 所示,在数据库表中主要包含主键 id (自增)、社区 id (唯一)、社区名称、简介、创建时间和更新时间字段等基本情况。

表 3.3 社区表 —— community

属性	类型	是否为空	长度	描述
id	int	NOT NULL	11	主键，自增
community_id	bigint	NOT NULL		社区 id，唯一
community_name	varchar	NOT NULL	128	社区名称
introduction	varchar	NOT NULL	1024	简介
create_time	timestamp	CURRENT_TIMESTAMP		创建时间
update_time	timestamp	CURRENT_TIMESTAMP		更新时间

## 4 论坛前端

### 4.1 注册

在本次研究中，项目开发了一个具有用户注册功能的前端主页。为了满足现代用户对于数据安全和隐私的需求，为注册过程设置了严格的字段验证，确保收集到的用户信息准确且符合规范。用户可以通过填写用户名、邮箱、密码、确认密码和性别这些必填字段进行注册。

注册表单的设计如下：

- （1）用户名：要求用户输入一个具有唯一性的用户名，以便于识别和管理。
- （2）邮箱：要求用户输入一个有效的电子邮件地址，以便用于接收系统通知、找回密码等功能。
- （3）密码：要求用户设置一个安全的密码，以保护账户信息。
- （4）确认密码：要求用户再次输入密码，以确保两次输入的密码一致。系统会对两次输入的密码进行比较，只有在它们完全一致时，注册过程才会继续进行。
- （5）性别：要求用户选择自己的性别。提供了“男性”、“女性”两个选项，以满足不同用户的需求。在选择性别时，用户只能选择一个选项。

如下图 4-1 所示。

The image shows a web registration form titled "注册" (Register) on the "Golang编程论坛" (Golang Programming Forum) website. The form is centered on a blue background. It includes input fields for "用户名" (Username), "邮箱" (Email), "密码" (Password), and "确认密码" (Confirm Password). There are also radio buttons for "性别" (Gender) with options for "男" (Male) and "女" (Female). A "提交" (Submit) button is at the bottom right of the form. The top of the page features a search bar and buttons for "登录" (Login) and "注册" (Register).

图 4-1 注册页面

在用户完成表单填写并提交后，系统对所有字段进行前端验证。如果发现任何不符合要求的字段，系统会立即提示用户进行修改。只有在所有字段都满足验证规则后，注册请求才会被发送到后端服务器进行进一步处理。

总之，通过设计一个严谨的注册表单和实现相应的前端验证功能，确保了用户注册过程的安全性和有效性。这有助于提升用户体验，同时降低因为错误信息导致的客户支持成本。

## 4.2 登录

登录页面的设计注重易用性，同时兼顾安全性。页面上仅包含两个主要字段：用户名和密码。

用户名字段要求用户输入在注册过程中所创建的唯一用户名，该用户名用于唯一标识用户。密码字段则要求用户输入相应的账户密码。为了确保登录过程的安全性，系统对密码字段进行了掩码处理，以防止他人窥视。用户填写完毕后，只需点击登录按钮，即可提交表单。

当后端对登录申请进行处理时，系统会对用户名和密码的正确性进行验证。只有在用户名和密码匹配的情况下，用户才能成功登录并访问受保护的资源。该登录页面不仅简化了用户登录过程，还为整个应用提供了一道安全屏障。如下图 4-2 所示。



图 4-2 登录页面

## 4.3 首页

首页页面如图 4-3 所示，最上面的是标题 Gophers (Go 语言社区的昵称，代表使用和支持 Go 语言的开发者)，首页左上角是网站名称：Golang 编程论坛。下面从左至右分别为博主信息及网站访问量，帖子排名列表及每日一言，Github 热榜等。用户可以分别点击进入查看详情。



图 4-3 首页

## 4.4 博主相关

### 4.4.1 访问量

访问量是指一个网站或 web 应用在特定时间段内被访问的次数。它是衡量网

站流行度和吸引力的重要指标之一,可以帮助网站拥有者或开发者了解网站的受欢迎程度和用户行为。

本网站使用由 Glitch 提供的计数器服务, Glitch 是一个在线开发平台, 允许用户创建、分享和运行各种类型的应用程序。通过将图片源链接至 Glitch 的计数器服务, 可以轻松地在网站上实现访问量统计功能。如图 4-4 所示。



图 4-4 访问量

上图的图片 src 属性链接到一个外部的计数器服务(Glitch), 用于实时统计访问量。当用户访问包含此段代码的页面时, 计数器服务会自动增加访问次数, 并将更新后的计数值以图形的形式返回。图片将显示当前的访问次数, 从而让网站访问者了解到访问量的情况。

4.4.2 博主信息

本站左侧边栏展示了博主的详细信息, 包括个人头像、日常生活轮播图、博主评级、公众号二维码、QQ 邮箱地址、QQ 交流群地址、联系电话、博主名言以及日历等。如下图 4-5、4-6、4-7、4-8 所示。



图 4-5 博主头像



图 4-6 轮播图





图 4-7 评级、媒体账号及名言



图 4-8 日历



这些元素反映了博主的个性和生活方式，帮助访问者更好地了解博主。通过提供这些联系方式，博主鼓励访问者与其互动、分享知识和建立联系，从而增强社区凝聚力。

### 4.4.3 留言

网站提供了一个留言功能，允许访问者向博主发送留言。点击留言按钮后，将进入留言界面。在此界面上，网站采用了畅言云评作为留言系统，让访问者能够方便地发表评论、提问或分享想法。如图 4-9 所示。

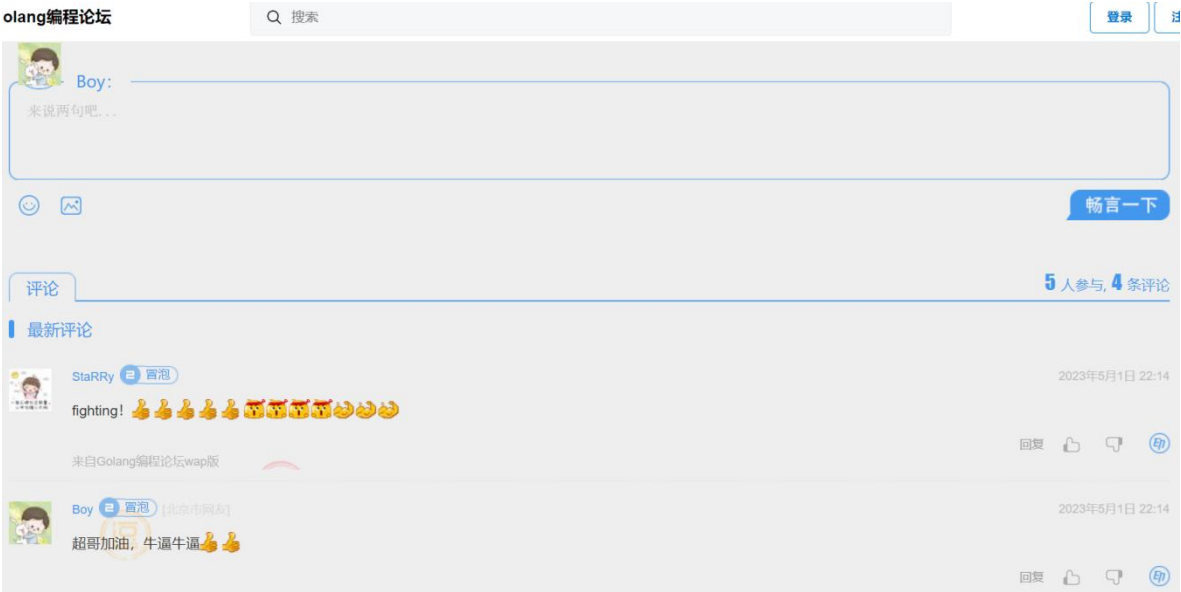


图 4-9 留言页面

畅言云评是一款易于集成的评论插件，为网站提供了强大的留言管理和社交分享功能。这种交互方式有助于建立博主与访问者之间的联系，增强网站的社区氛围。

### 4.4.4 网站运行时间

网站运行时间通常指从网站开始运行至当前时刻的时间长度。它可以帮助访问者了解网站的在线历程和稳定性。通常，网站运行时间会以天、小时、分钟和秒为单位显示在网站首页或其他显眼位置。如图 4-10 所示。



图 4-10 网站运行时间图

网站运行时间计算方法如下：

- (1) 首先，确定网站开始运行的时间，为本网的启动日期。
- (2) 获取当前时间，通过 JavaScript 的 Date 对象获取。
- (3) 计算出两次时间之差，也就是站点的实际运行时间。这可以把两个时间转化成一个时间标记：毫秒为单位的时间戳，然后计算它们的差值来实现。
- (4) 将时间差转换为天、小时、分钟和秒的形式，以便更直观地展示网站运行时间。

在网站上实现运行时间显示，本项目前端使用 Vue 组件实现。通过将网站运行时间展示在首页或其他显眼位置，可以提供给访问者一个直观的了解网站在线历程的方式。

## 4.5 帖子相关

### 4.5.1 列表

前端页面上设有帖子列表，展示了用户发表的帖子。列表中的帖子可以根据时间或分数进行排序，以便让访问者更容易找到感兴趣的内容。时间排序按照帖子的发布时间进行排列，而分数排序则根据帖子的点赞数、发布时间等互动数据进行排序。如图 4-11、4-12 所示。

New

Score

发表

↑  
2  
↓

### Go语言实现的可读性更高的并发神库

前言 哈喽，大家好，我是asong；前几天逛github发现了一个有趣的并发库-conc，其目标是：更难出现goroutine泄漏 处理panic更友好 并发代码可读性高 从简介上看主要封装功能如下：对waitGroup进行封装，避免了产生大量重复代码，并且也封装recover，安全性更高 提供panics.Catcher封装recover逻辑，...

↑  
4  
↓

### Go的ORM也太拉跨了吧，赶紧给他封装一下

背景 去年慢慢开始接触了Go语言，也在公司写了几个Go的生产项目。我是从Java转过来的。（其实也不算转，公司用啥，我用啥）在这个过程中，老是想用Java的思维写Go，在开始的一两个月，那是边写边吐槽。丑陋的错误处理，没有流式处理，还竟然没有泛型，框架生态链不成熟，没有一家独大的类似Sprin...

↑  
4  
↓

### Redis只用来做缓存？来认识一下它其他强大的能力吧。

当今互联网应用中，随着业务的发展，数据量越来越大，查询效率越来越高，对于时序数据的存储、查询和分析需求也越来越强烈，这时候 Redis 就成为了首选的方案之一。Redis 提供了多种数据结构，如字符串、哈希表、列表、集合、有序集合等，每种数据结构都具备不同的特性，可以满足不同的业务需求。其中...

↑  
3  
↓

### 2023最新后端中大厂面经&在面试过程中如何反问？

1. 背景 距离上一次面试已经过去快3年了，又碰上2022年互联网行业大动荡，很多企业都做出了裁员决定。身为互联网人要时刻关注自身成长，所以会对“跟面试官切磋”这种事情比较感兴趣，其实也是一种打探行情的手段：3年间自己的技术是线性成长、指数成长还是对数成长，跟同龄人相比如何？当前求职是...

↑  
4  
↓

### 深入理解 ClickHouse

ClickHouse是一个用于联机分析(OLAP)的列式数据库管理系统(DBMS)。本文将深入探讨ClickHouse的独特特性、高性能查询引擎以及在实际应用场景中的优势。列式数据库和 OLAP 首先我们先来理解下列式数据库和 OLAP (online Analytical Processing)，因为它们构成了 ClickHouse 的基础。在传统的行式...

共 19 条

5条/页

< 1 2 3 4 >

前往 1 页

图 4-11 时间排序

New

Score

发表

↑  
4  
↓

Go的ORM也太拉跨了吧，赶紧给他封装一下

背景 去年慢慢开始接触了Go语言，也在公司写了几个Go的生产项目。我是从Java转过来的。（其实也不算转，公司用啥，我用啥）在这个过程中，老是想用Java的思维写Go，在开始的一两个月，那是边写边吐槽。丑陋的错误处理，没有流式处理，还竟然没有泛型，框架生态链不成熟，没有一家独大的类似Sprin...

↑  
4  
↓

Redis只用来做缓存？来认识一下它其他强大的能力吧。

当今互联网应用中，随着业务的发展，数据量越来越大，查询效率越来越高，对于时序数据的存储、查询和分析需求也越来越强烈，这时候 Redis 就成为了首选的方案之一。Redis 提供了多种数据结构，如字符串、哈希表、列表、集合、有序集合等，每种数据结构都具备不同的特性，可以满足不同的业务需求。其中...

↑  
2  
↓

Go语言实现的可读性更高的并发神库

前言 哈喽，大家好，我是asong；前几天逛github发现了一个有趣的并发库-conc，其目标是： 更难出现goroutine泄漏 处理panic更友好 并发代码可读性高 从简介上看主要封装功能如下： 对waitGroup进行封装，避免了产生大量重复代码，并且也封装recover，安全性更高 提供panics.Catcher封装recover逻辑，...

↑  
4  
↓

深入理解 ClickHouse

ClickHouse是一个用于联机分析(OLAP)的列式数据库管理系统(DBMS)。本文将深入探讨ClickHouse的独特特性、高性能查询引擎以及在实际应用场景中的优势。列式数据库和 OLAP 首先我们先来理解下列式数据库和 OLAP (online Analytical Processing)，因为它们构成了 ClickHouse 的基础。在传统的行式...

↑  
4  
↓

RocketMQ实战一：先写库还是先发消息？

我们以日常开发中的案例来进行分析：下单送积分。用户在下单后，订单系统保存订单数据，然后发送消息到MQ，积分系统订阅这个消息，然后给用户加积分。这就引出了一个问题，从生产者订单系统角度看，到底是先写库还是先发消息呢？那我们接下来就分别看下这两种情况。1. 先写库后发消息 首先，执行...

共 19 条

5条/页

< 1 2 3 4 >

前往 1 页

图 4-12 分数排序

帖子列表还提供了分页功能，使用户能够在多个页面间轻松浏览。这有助于提高页面加载速度，同时也便于用户在查找特定帖子时进行筛选和定位。点击某

15



个帖子后，用户将进入该帖子的详情页面。在详情页面中，访问者可以查看帖子的完整内容，包括文字、评论和其他多媒体元素。此外，用户还可以在此页面上参与评论和点赞等互动，与其他访问者共同讨论和分享观点。

### 4.5.2 发表

发表文章页面为用户提供了一个方便的平台，用于撰写和发布新文章。在页面左上角，用户可以从下拉菜单中选择合适的社区，以便将新文章归档至相应的类别。这有助于对内容进行整理，同时也方便其他访问者按照主题浏览文章。页面右侧展示了发帖规范，引导用户遵循一定的规则和格式发表文章。这些规范可以确保社区内的内容质量，同时也维护了良好的讨论氛围。如图 4-13 所示。

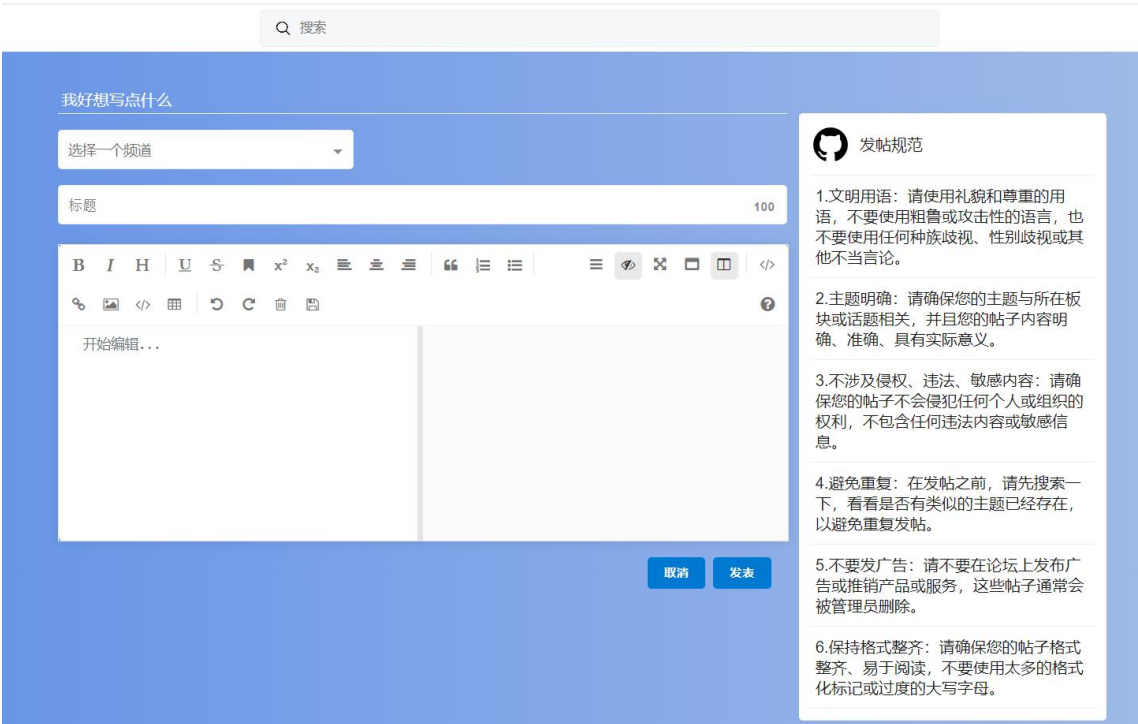


图 4-13 发表文章页面

发表文章时，用户需要输入文章标题和内容。标题应简洁明了，概括文章的主题，而内容则可以包含文字、图片和其他多媒体元素，以展示作者的观点和想法。在完成编辑后，用户可以点击提交按钮，将文章发布到所选社区，与其他访问者共享知识和经验。

### 4.5.3 详情

文章详情页展示了文章的完整信息，为用户提供了一个查看和互动的平台。页面包含以下几个主要部分：

1. 文章详情：展示了文章的全文内容，包括文字、图片和其他多媒体元素。这部分让访问者能够阅读文章并了解作者的观点和想法。

2. 点赞数：显示了该文章获得的点赞数量。这个数字反映了文章的受欢迎程度，帮助其他访问者评估文章的价值和质量。

3. 所属社区：显示了该文章归属于哪个社区，帮助访问者了解文章的主题和分类。点击所属社区，用户可以查看该社区的详情，包括简介、其他文章以及相关讨论。如图 4-14 所示。



图 4-14 文章详情及评论页面

通过这个细致且丰富的详情页设计，网站为用户提供了一个便捷且全面的文章阅读和互动体验，有助于增强社区的凝聚力和活跃度。

#### 4.5.4 评论

在文章详情页面底部设置一个评论区域，用户可以在这个区域内发表自己的观点和想法。评论区通常包括评论列表和发表评论的表单。用户可以在表单中输入评论内容，并选择发布。之后，其他阅读该文章的用户可以看到新发布的评论，这样就形成了一个交流的平台，方便用户分享想法和参与讨论，为社区营造了一个有益的学习环境。如图 4-15 所示。



图 4-15 文章评论页面

## 4.6 社区相关

社区文章列表页为用户展示了特定社区下的文章列表和相关信息。当从文章详情页点击 join 按钮后，用户会被引导至相应的社区页。页面主要包含以下两部分内容：

1. 文章列表：位于页面左侧，显示了该社区下的文章列表。列表按照文章分数进行倒序排序，以便访问者快速找到热门和高质量的内容。同时，文章列表支持分页功能，使用户能够在多个页面间轻松浏览。

2. 社区详细信息：位于页面右侧，展示了该社区的名称、描述以及创建时间。这些信息帮助访问者了解社区的主题和特点，以便更好地参与到相关讨论中。

如图 4-16 所示。



图 4-16 社区文章列表页

在社区页的右上角，提供了一个发表文章按钮。点击此按钮，用户将进入发帖页面，可以在所选社区下发表新的帖子。这个设计鼓励用户积极参与社区讨论，分享知识和经验，从而丰富社区内容和活跃度。

## 4.7 三方接口相关

### 4.7.1 每日一言

网站顶部设有一个名为“每日一言”的功能模块，展示一句精选的格言或名言，为访问者带来鼓舞和启发。点击该模块后，系统会自动刷新并显示一条新的每日一言。这种设计提高了网站的趣味性和互动性，使访问者能够继续关注 and 参与其中，同时为他们的在线体验增添一份惊喜和灵感。如图 4-17 所示。



图 4-17 每日一言



此功能通过 Vue 和 Axios 实现，从指定的 API:

<https://v.api.aal.cn/api/yiyan/index.php>

获取每日一言。这个功能可以为网站增加趣味性和互动性，吸引访问者关注并参与。

#### 4.7.2 Github 热门项目榜

网站右侧组件实时展示了 GitHub 上热门项目的排行榜。该功能通过调用 GitHub API:

[https://api.github.com/search/repositories?q=stars:%3E1&sort=stars&order=desc&page=10&per\\_page=1](https://api.github.com/search/repositories?q=stars:%3E1&sort=stars&order=desc&page=10&per_page=1)

获取项目列表，并按照星数降序排列。在页面加载时，组件会首先展示三条热门项目。当用户点击“加载更多”按钮时，将分页加载更多热门项目，每次新增 3 条。如图 4-18 所示。

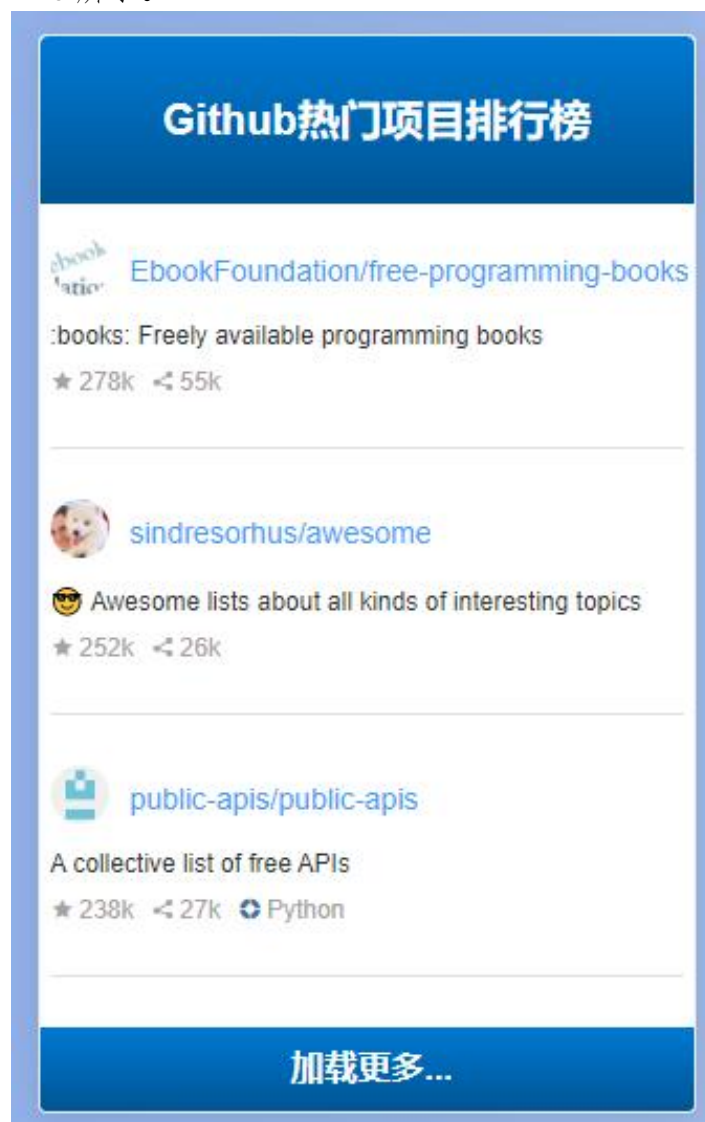


图 4-18 Github 热门项目榜

这个组件为访问者提供了一个方便的途径来查看当前 GitHub 上最受关注的开源项目，有助于了解开源社区的热门趋势和感兴趣的项目。同时，分页加载功能可以避免一次加载太多的数据，从而加快网页的响应速度和用户体验。

#### 4.7.3 Golang 热门项目榜

在网站的右侧组件中，展示了以 Golang 编写的热门项目排行榜。该功能通过调用 GitHub API：

[https://api.github.com/search/repositories?q=stars:%3E1+language:go&sort=stars&order=desc&page=10&per\\_page=1](https://api.github.com/search/repositories?q=stars:%3E1+language:go&sort=stars&order=desc&page=10&per_page=1)

来获取 Go 语言相关的热门项目，并依据项目的星数进行降序排列。API 请求已特别添加了 language:go 参数，以便仅筛选 Go 语言相关项目。如图 4-19 所示。

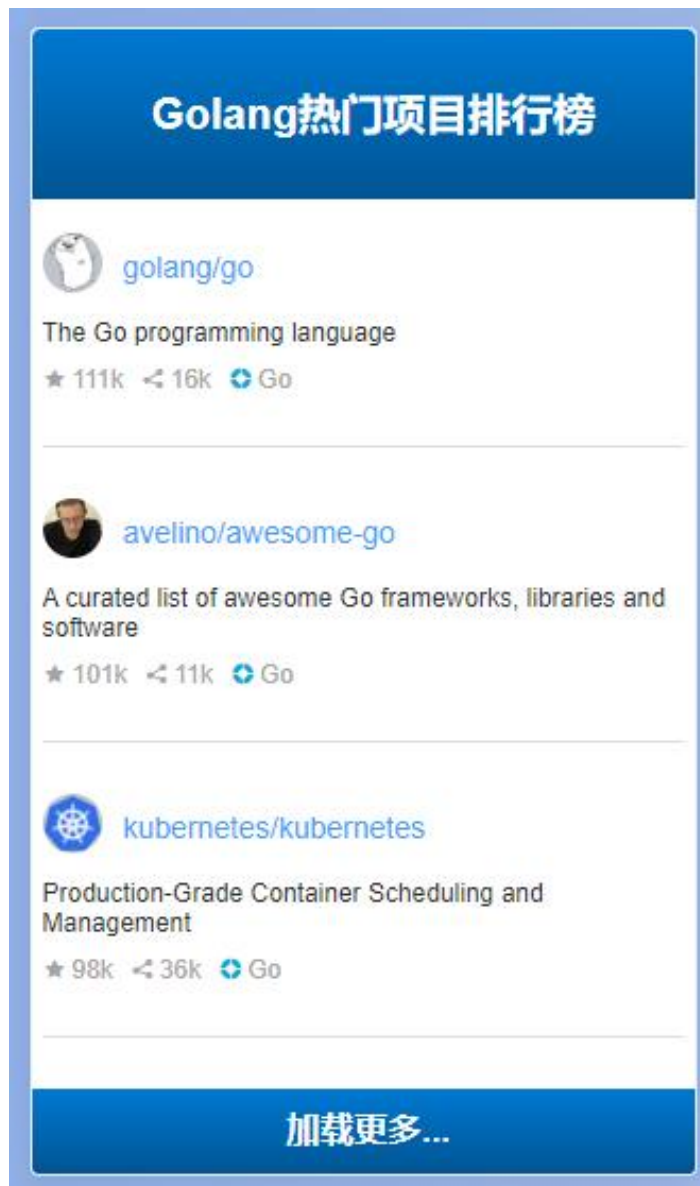


图 4-19 Golang 热门项目榜

在初次访问页面时，组件将展示三个 Go 语言的热门项目。用户可通过点击“加载更多”按钮来查看更多项目，每次点击将额外加载 3 个项目。

这一组件使访问者能够快速了解当前 GitHub 上 Go 语言领域的热门项目和发展趋势，对于关注 Go 语言的开发者来说具有很高的参考价值。

## 5 论坛后端

### 5.1 JWT 认证

JWT 是一种以 Token 为基础的轻量级认证模式，当服务端在验证成功后，会产生一个 JSON 对象，在签名后，会得到一个 Token（令牌），然后再发回给用户，用户在后续的请求中，只要把这个 Token 带上，服务端对其进行解密，就可以获得该用户的相关信息。

下面演示在 Gin 框架中使用 JWT：

首先，在确定在 JWT 中要保留什么数据时，您需要自定义您自己的需求，本项目规定在 JWT 中要存储 user\_id、username 信息，那么就定义一个 MyClaims 结构体，如图 5-1 所示。

```
10 // MyClaims 自定义声明结构体并内嵌jwt.StandardClaims
11 // jwt包自带的jwt.StandardClaims只包含了官方字段
12 // 我们这里需要额外记录一个UserID字段，所以要自定义结构体
13 // 如果想要保存更多信息，都可以添加到这个结构体中
14 type MyClaims struct {
15     UserID    uint64 `json:"user_id"`
16     Username  string `json:"username"`
17     jwt.StandardClaims
18 }
```

图 5-1 自定义 Claims

然后定义 JWT 的过期时间，这里定义为 7 天，如图 5-2 所示。

```
27 // TokenExpireDuration 定义JWT的过期时间
28 const TokenExpireDuration = time.Hour * 24 * 7
```

图 5-2 定义 JWT 的过期时间

接下来还需要定义一个用于签名的字符串，如图 5-3 所示。

```
20 //定义Secret 用于加密的字符串
21 var mySecret = []byte("bluebell")
```

图 5-3 定义用于加密的字符串

据自己的业务需要封装一个生成 token 的函数，如图 5-4 所示。

```

30 // GenToken 生成JWT 生成access token 和 refresh token
31 func GenToken(userID uint64, username string) (aToken, rToken string, err error) {
32     // 创建一个我们自己的声明
33     c := MyClaims{
34         UserID: userID, // 自定义字段
35         Username: username, // 自定义字段
36         jwt.StandardClaims{ // JWT规定的7个官方字段
37             ExpiresAt: time.Now().Add(TokenExpireDuration).Unix(), // 过期时间
38             Issuer: "bluebell", // 签发人
39         },
40     }
41     // 加密并获得完整的编码后的字符串token
42     aToken, err = jwt.NewWithClaims(jwt.SigningMethodHS256, c).SignedString(mySecret)
43
44     // refresh token 不需要存任何自定义数据
45     rToken, err = jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.StandardClaims{
46         ExpiresAt: time.Now().Add(time.Second * 30).Unix(), // 过期时间
47         Issuer: "bluebell", // 签发人
48     }).SignedString(mySecret)
49     // 使用指定的secret签名并获得完整的编码后的字符串token
50     return
51 }

```

图 5-4 生成 access\_token 和 refresh\_token 的函数

根据给定的 JWT 字符串，解析出数据，如图 5-5 所示。

```

75 // ParseToken 解析JWT
76 func ParseToken(tokenString string) (claims *MyClaims, err error) {
77     // 解析token
78     var token *jwt.Token
79     claims = new(MyClaims)
80     token, err = jwt.ParseWithClaims(tokenString, claims, keyFunc)
81     if err != nil {
82         // 校验token
83         if !token.Valid {
84             err = errors.New(text: "invalid token")
85         }
86     }
87     return
88 }

```

图 5-5 解析 JWT 的函数

当 access\_token 过期时，可以选择用 refresh\_token 来进行刷新，函数如图 5-6 所示。

```

90 // RefreshToken 刷新AccessToken
91 func RefreshToken(aToken, rToken string) (newAToken, newRToken string, err error) {
92     // refresh token无效直接返回
93     if _, err = jwt.Parse(rToken, keyFunc); err != nil {
94
95     }
96
97     // 从旧access token中解析出claims数据 解析出payload负载信息
98     var claims MyClaims
99     _, err = jwt.ParseWithClaims(aToken, &claims, keyFunc)
100     v, _ := err.(*jwt.ValidationError)
101
102     // 当access token是过期错误 并且 refresh token没有过期时就创建一个新的access token
103     if v.Errors == jwt.ValidationErrorExpired {
104         return GenToken(claims.UserID, claims.Username)
105     }
106     return
107 }

```

图 5-6 刷新 access\_token 的函数

## 5.2 分布式 ID: 雪花算法

Sonyflake 是一个可扩展的、高性能的、全局唯一 ID 生成算法，旨在满足分布式系统中 ID 生成的需求。该算法基于 Twitter 的雪花算法 (Snowflake) 设计，通过组合时间戳、机器 ID 和序列号来生成 64 位的整数 ID。这些 ID 具有以下特性：

- (1) 全局唯一：确保在分布式环境下生成的 ID 不会发生冲突。
- (2) 时间有序：生成的 ID 随时间单调递增，便于排序和检索。
- (3) 高性能：算法具有低延迟和高吞吐量，可在高并发场景下使用。

首先，在项目中安装 sony/sonyflake 库，执行如下命令。

```
go get -u github.com/sony/sonyflake
```

然后，在需要使用 ID 生成器的 Go 文件中引入库，如图 5-7 所示。

```

3 import (
4     "github.com/sony/sonyflake"
5 )

```

图 5-7 系统结构图

初始化 ID 生成器在代码中创建一个新的 Sonyflake 实例，用于生成全局唯一 ID，如图 5-8 所示。



```

18 // Init 需传入当前的机器ID
19 func Init(machineId uint16) (err error) {
20     sonyMachineID = machineId
21     t, _ := time.Parse( layout: "2006-01-02", value: "2022-02-09") // 初始化一个开始的时间
22     settings := sonyflake.Settings{ // 生成全局配置
23         StartTime: t,
24         MachineID: getMachineID, // 指定机器ID
25     }
26     sonyFlake = sonyflake.NewSonyflake(settings) // 用配置生成sonyflake节点
27     return
28 }

```

图 5-8 生成全局唯一 ID

使用 NextID() 方法从 Sonyflake 实例生成新的 ID，如图 5-9 所示。

```

30 // GetID 返回生成的id值
31 func GetID() (id uint64, err error) { // 拿到sonyFlake节点生成id值
32     if sonyFlake == nil {
33         err = fmt.Errorf( format: "snoy flake not initied")
34         return
35     }
36
37     id, err = sonyFlake.NextID()
38     return
39 }

```

图 5-9 从 Sonyflake 实例生成新的 ID

总结：Sonyflake 是一个适用于分布式环境的高性能全局唯一

### 5.3 配置文件管理：viper

Viper 是一个为 Go 应用提供全面配置解决方案的库。其设计初衷是在应用程序中充分发挥作用，同时能够处理各种类型的配置需求和格式。在开发现代应用程序的过程中，开发者无需过多关注配置文件的格式问题，而应该将重点放在打造优秀的软件上。Viper 库的诞生正是为了在这方面提供支持。

安装，执行如下命令。

```
go get github.com/spf13/viper
```

读取配置文件。下面是一个如何使用 Viper 搜索和读取配置文件的示例。不需要任何特定的路径，但是至少应该提供一个配置文件预期出现的路径。如图 5-10 所示。

```

// 读取配置文件
viper.SetConfigFile( in: "./conf/config.yaml")

```

图 5-10 读取配置文件

当加载配置文件发生错误时，对于无法找到配置文件的特殊情形，可以采用

如下方式，如图 5-11 所示。

```
61 // 查找并读取配置文件
62 err := viper.ReadInConfig()
63 if err != nil {
64     panic(fmt.Errorf("ReadInConfig failed, err: %v", err))
65 }
```

图 5-11 查找并读取配置文件

现如今，不再需要重启服务器以使配置生效，使用 Viper 驱动的应用程序能够在运行过程中实时读取配置文件的更新，不会漏掉任何信息。只需为 Viper 实例设置 watchConfig。此外，还可以为 Viper 设置一个回调函数，以便在配置发生更改时每次都能自动执行。如图 5-12 所示。

```
54 // 读取环境变量
55 viper.WatchConfig()
56 // 监听配置文件变化
57 viper.OnConfigChange(func(in fsnotify.Event) {
58     fmt.Println(a...: "天寿啦~配置文件被人修改啦...")
59     viper.Unmarshal(&Conf)
60 })
```

图 5-12 监控并重新读取配置文件

当需要将 viper 读取的配置反序列到自己定义的结构体变量中时，一定要使用 mapstructure tag，如图 5-13 所示。

```
66 // 把读取到的配置信息反序列化到Conf变量中
67 if err := viper.Unmarshal(&Conf); err != nil {
68     panic(fmt.Errorf("unmarshal to Conf failed, err:%v", err))
69 }
```

图 5-13 把读取到的配置信息反序列化到 Conf 变里中

## 5.4 参数校验: validator

在 Web 开发过程中，请求参数校验是不可避免的一环。通常，会在代码中定义与请求参数相对应的模型（结构体），并利用模型绑定便捷地解析请求中的参数，如 Gin 框架中的 Bind 和 ShouldBind 系列方法。本文将以 Gin 框架的请求参数校验为例，阐述一些 validator 库的实用技巧。

validator 库本身支持多语言特性，通过使用相应的语言包，可以实现校验错误提示信息的自动翻译。以下示例代码展示了如何将错误提示信息转换为中文，将其翻译成其他语言的方法也是类似的。如图 5-14、5-15 所示。

```

18 // 定义一个全局翻译器T
19 var trans ut.Translator
20
21 // InitTrans 初始化翻译器
22 func InitTrans(locale string) (err error) {
23     // 修改gin框架中的Validator引擎属性, 实现自定义
24     if v, ok := binding.Validator.Engine().(*validator.Validate); ok {
25         // 注册一个获取json tag的自定义方法
26         v.RegisterTagNameFunc(func(fld reflect.StructField) string {
27             name := strings.SplitN(fld.Tag.Get("key: json"), sep, 2)[0]
28             if name == "-" : ""
29             return name
30         })
31     }
32     // 为SignUpParam注册自定义校验方法
33     v.RegisterStructValidation(SignUpParamStructLevelValidation, models.RegisterForm{})
34     zhT := zh.New() // 中文翻译器
35     enT := en.New() // 英文翻译器
36     // 第一个参数是备用 (fallback) 的语言环境
37     // 后面的参数是应该支持的语言环境 (支持多个)
38     // uni := ut.New(zhT, zhT) 也是可以的
39     uni := ut.New(enT, zhT, enT)
40     // locale 通常取决于 http 请求头的 'Accept-Language'
41     var ok bool
42     // 也可以使用 uni.FindTranslator(...) 传入多个locale进行查找
43     trans, ok = uni.GetTranslator(locale)
44     if !ok : fmt.Errorf("uni.GetTranslator(%s) failed", locale)
45     // 注册翻译器
46     switch locale {
47     case "en":
48         err = enTranslations.RegisterDefaultTranslations(v, trans)
49     case "zh":
50         err = zhTranslations.RegisterDefaultTranslations(v, trans)
51     default:
52         err = enTranslations.RegisterDefaultTranslations(v, trans)
53     }
54     return
55 }
56
57 return
58
59 }
60

```

图 5-14 初始化翻译器

```

62 // 定义一个去掉结构体名称前缀的自定义方法:
63 func removeTopStruct(fields map[string]string) map[string]string {
64     res := map[string]string{}
65     for field, err := range fields {
66         res[field[strings.Index(field, substr: ".")+1:]] = err
67     }
68     return res
69 }

```

图 5-15 用于移除结构体名称前缀的自定义函数



最后,在代码中使用上述函数将翻译后的 errors 做一下处理即可。如图 5-14 所示。

```
17 // SignUpHandler 注册业务
18 func SignUpHandler(c *gin.Context) {
19     // 1. 获取请求参数
20     var fo *models.RegisterForm
21     // 2. 校验数据有效性
22     if err := c.ShouldBindJSON(&fo); err != nil {
23         // 请求参数有误, 直接返回响应
24         zap.L().Error( msg: "SignUp with invalid param", zap.Error(err))
25         // 判断err是不是 validator.ValidationErrors类型的errors
26         errs, ok := err.(validator.ValidationErrors)
27         if !ok {
28             // 非validator.ValidationErrors类型错误直接返回
29             ResponseError(c, CodeInvalidParams) // 请求参数错误
30             return
31         }
32         // validator.ValidationErrors类型错误则进行翻译
33         ResponseErrorWithMsg(c, CodeInvalidParams, removeTopStruct(errs.Translate(trans)))
34         return // 翻译错误
35     }
36     fmt.Printf("fo: #{fo}\n")
37     // 3. 业务处理 —— 注册用户
38     if err := logic.SignUp(fo); err != nil {
39         zap.L().Error( msg: "logic.signup failed", zap.Error(err))
40         if err.Error() == mysql.ErrorUserExit {
41             ResponseError(c, CodeUserExist)
42             return
43         }
44         ResponseError(c, CodeServerBusy)
45         return
46     }
47     // 返回响应
48     ResponseSuccess(c, data: nil)
49 }
```

图 5-16 翻译 errors

在前面的验证中存在一个小问题,当涉及到一些较为复杂的验证规则时,例如 re\_password 字段需要与 password 字段值相同的情况,之前的自定义错误提示字段名称方法无法很好地解决错误提示信息中的其他字段名称问题。为了实现更优的提示效果,可以将 Password 字段名称也修改为与 JSON 标签一致,并需要为 SignUpParam 自定义一个校验方法。以下是一个自定义校验方法的示例,如图 5-17 所示。

```
71 // SignUpParamStructLevelValidation 自定义SignUpParam结构体校验函数
72 func SignUpParamStructLevelValidation(sl validator.StructLevel) {
73     su := sl.Current().Interface().(models.RegisterForm)
74
75     if su.Password != su.ConfirmPassword {
76         // 输出错误提示信息, 最后一个参数就是传递的param
77         sl.ReportError(su.ConfirmPassword, fieldName: "confirm_password",
78             structFieldName: "ConfirmPassword", tag: "eqfield", param: "password")
79     }
80 }
```

图 5-17 自定义 SignUpParm 结构体校验函数

接下来，在初始化的校验函数中添加您自己定义的校验方法。

## 5.5 日志库：zap

在许多 Go 语言项目中，需要一个好的日志记录器能够提供下面这些功能：

- (1) 可以将事件记录到文件中，而非仅限于应用程序控制台。
- (2) 日志分割 - 根据文件大小、时间或间隔等条件来分割日志文件。支持多种日志级别，如 INFO，DEBUG，ERROR 等。
- (3) 能够显示基本信息，例如调用文件/函数名、行号和日志时间等。

Zap 是一个极速的、结构化的，并支持分日志级别的 Go 日志库。本部分将介绍如何在基于 Gin 框架开发的项目中配置并使用 Zap 来接收和记录 Gin 框架的默认日志，以及如何实现日志归档的配置。

安装，执行如下命令。

```
go get -u go.uber.org/zap
```

创建一个全局的 Logger 对象，如下图 5-18 所示。

```
19 var lg *zap.Logger
```

图 5-18 全局 Logger 对象

然后编写一个初始化日志记录器的函数。它接收一个日志配置对象 (cfg) 和一个模式字符串 (mode)，并返回一个错误。如下图 5-19 所示。

```
21 // Init 初始化lg
22 func Init(cfg *settings.LogConfig, mode string) (err error) {
23     writeSyncer := getLogWriter(cfg.Filename, cfg.MaxSize, cfg.MaxBackups, cfg.MaxAge)
24     encoder := getEncoder()
25     var l = new(zapcore.Level)
26     err = l.UnmarshalText([]byte(cfg.Level))
27     if err != nil {
28         return err
29     }
30     var core zapcore.Core
31     if mode == "dev" {
32         // 进入开发模式，日志输出到终端
33         consoleEncoder := zapcore.NewConsoleEncoder(zap.NewDevelopmentEncoderConfig())
34         core = zapcore.NewTee( // 多个输出
35             zapcore.NewCore(encoder, writeSyncer, l), // 往日志文件里面写
36             zapcore.NewCore(consoleEncoder, zapcore.Lock(os.Stdout), zapcore.DebugLevel), // 终端输出
37         )
38     } else {
39         core = zapcore.NewCore(encoder, writeSyncer, l)
40     }
41
42     lg = zap.New(core, zap.AddCaller())
43     zap.ReplaceGlobals(lg)
44     zap.L().Infof("init logger success")
45     return
46 }
```

图 5-19 初始化日志记录器

这段代码的作用是根据给定的配置和模式创建一个 Logger 对象，并将其设置为全局的默认 Logger。根据不同的模式，日志可以同时输出到终端和文件中，或者只输出到文件中。

接下来创建一个 JSON 编码器，用于格式化日志并创建一个日志写入器。如图 5-20、5-21 所示。

```
48 func getEncoder() zapcore.Encoder {
49     encoderConfig := zap.NewProductionEncoderConfig()
50     encoderConfig.EncodeTime = zapcore.ISO8601TimeEncoder
51     encoderConfig.TimeKey = "time"
52     encoderConfig.EncodeLevel = zapcore.CapitalLevelEncoder
53     encoderConfig.EncodeDuration = zapcore.SecondsDurationEncoder
54     encoderConfig.EncodeCaller = zapcore.ShortCallerEncoder
55     return zapcore.NewJSONEncoder(encoderConfig)
56 }
```

图 5-20 JSON 编码器

它使用 Zap 提供的生产环境编码器配置，并将时间编码为 ISO8601 格式、设置时间键名为 "time"、将日志级别编码为大写形式、将持续时间编码为秒，并使用短格式编码调用者信息。

```
58 func getLogWriter(filename string, maxSize, maxBackup, maxAge int) zapcore.WriteSyncer {
59     lumberjackLogger := &lumberjack.Logger{
60         Filename:   filename,
61         MaxSize:    maxSize,
62         MaxBackups: maxBackup,
63         MaxAge:     maxAge,
64     }
65     return zapcore.AddSync(lumberjackLogger)
66 }
```

图 5-21 日志写入器

它使用 Lumberjack 库创建一个具有文件名、最大大小、最大备份数和最大保留时间的日志写入器，并将其封装为一个 WriteSyncer 对象。

最后，使用 zap 实现两个自定义中间件 GinLogger 用于接收 gin 框架默认日志、GinRecovery 用于 recover 掉项目中可能出现的 panic。如图 5-22、5-23 所示。

```

68 // GinLogger 接收gin框架默认的日志
69 func GinLogger() gin.HandlerFunc {
70     return func(c *gin.Context) {
71         start := time.Now()
72         path := c.Request.URL.Path
73         query := c.Request.URL.RawQuery
74         c.Next()
75
76         cost := time.Since(start)
77         lg.Info(path,
78             zap.Int(key: "status", c.Writer.Status()),
79             zap.String(key: "method", c.Request.Method),
80             zap.String(key: "path", path),
81             zap.String(key: "query", query),
82             zap.String(key: "ip", c.ClientIP()),
83             zap.String(key: "user-agent", c.Request.UserAgent()),
84             zap.String(key: "errors", c.Errors.ByType(gin.ErrorTypePrivate).String()),
85             zap.Duration(key: "cost", cost),
86         )
87     }
88 }

```

图 5-22 自定义中间件 GinLogger

它在处理请求之前记录请求的起始时间，并在请求处理完成后计算请求处理的耗时。然后，使用 Logger 对象记录请求的相关信息，包括请求路径、状态码、请求方法、查询参数、客户端 IP、用户代理、错误信息和处理耗时。

```

90 // GinRecovery recover掉项目可能出现的panic，并使用zap记录相关日志
91 func GinRecovery(stack bool) gin.HandlerFunc {
92     return func(c *gin.Context) {
93         defer func() {
94             if err := recover(); err != nil {
95                 var brokenPipe bool
96                 if ne, ok := err.(*net.OpError); ok {
97                     if se, ok := ne.Err.(*os.SyscallError); ok {
98                         if strings.Contains(strings.ToLower(se.Error()), substr: "broken pipe") ||
99                            strings.Contains(strings.ToLower(se.Error()), substr: "connection reset by peer") {
100                             brokenPipe = true
101                         }
102                     }
103                 }
104                 httpRequest, _ := httputil.DumpRequest(c.Request, body: false)
105                 if brokenPipe {
106                     lg.Error(c.Request.URL.Path,
107                         zap.Any( key: "error", err),
108                         zap.String( key: "request", string(httpRequest)),
109                     )
110                     // If the connection is dead, we can't write a status to it.
111                     c.Error(err.(error)) // nolint: errcheck
112                     c.Abort()
113                     return
114                 }
115                 if stack {
116                     lg.Error( msg: "[Recovery from panic]",
117                         zap.Any( key: "error", err),
118                         zap.String( key: "request", string(httpRequest)),
119                         zap.String( key: "stack", string(debug.Stack())),
120                     )
121                 } else {
122                     lg.Error( msg: "[Recovery from panic]",
123                         zap.Any( key: "error", err),
124                         zap.String( key: "request", string(httpRequest)),
125                     )
126                 }
127                 c.AbortWithStatus(http.StatusInternalServerError)
128             }
129         }()
130         c.Next()

```

图 5-23 自定义中间件 GinRecovery

它使用 `defer` 和 `recover` 捕获可能出现的 `Panic`，并进行处理。如果发生 `Panic`，它会检查是否是由于连接中断导致的，并记录相关的错误信息和请求内容。如果是连接中断导致的 `Panic`，则记录错误并中止请求处理。如果不是连接中断导致的 `Panic`，则根据传入的 `stack` 参数决定是否记录堆栈信息。最后，将请求的处理状态设置为 500，并中止请求处理。

综上，这些函数提供了一个简单而强大的日志记录功能，可以方便地集成到基于 `Gin` 的 `Web` 应用程序中。

## 5.6 令牌桶限流

流量控制（又称限流）通常指限制到达系统的并发请求数。虽然限流可能会对部分用户的体验产生影响，但它在很大程度上保障了系统的稳定性，防止因过大的并发请求导致系统崩溃（这样一来，所有人的用户体验都会受到影响）。

本项目使用令牌桶限流策略来保证系统后端的稳定性。



令牌桶算法的基本原理是：令牌桶以固定的速率往桶中添加令牌，只要能从桶中取出令牌，请求就可以通过。令牌桶算法可以有效地处理突发流量，使得系统能够快速响应这些突然增加的请求。当桶中没有足够的令牌时，有两种处理方式：一种是等待直到桶中有足够的令牌；另一种是直接拒绝请求并返回相应的错误提示。通过这种方式，令牌桶算法为流量控制提供了一种灵活且高效的解决方案。如图 5-24 所示。

```
18 // RateLimitMiddleware 创建指定填充速率和容量大小的令牌桶
19 func RateLimitMiddleware(fillInterval time.Duration, cap int64) func(c *gin.Context) {
20     bucket := ratelimit.NewBucket(fillInterval, cap)
21     return func(c *gin.Context) {
22         // 如果取不到令牌就中断本次请求返回 rate limit...
23         if bucket.TakeAvailable(count: 1) == 0 {
24             c.String(http.StatusOK, format: "rate limit...")
25             c.Abort()
26             return
27         }
28         // 取到令牌就放行
29         c.Next()
30     }
31 }
32
```

图 5-24 自定义令牌桶限流中间件

虽然名为令牌桶，但实际上并无需真正生成令牌并将其放入桶中。每次请求令牌时，只需计算当前是否有足够的令牌。具体计算方法可总结为以下公式：

当前令牌数 = 上次剩余令牌数 + (本次请求令牌时刻 - 上次请求令牌时刻) / 放置令牌时间间隔 \* 每次放置令牌数

针对限流中间件的注册位置，可以根据不同的限流策略将其注册到相应的位置，例如：

- (1) 若需对整个网站进行限流，可将限流中间件注册为全局中间件。
- (2) 若只需对特定路由组进行限流，只需将限流中间件注册到相应的路由组即可。

通过这种方式，可以根据实际需求灵活地设置限流策略，从而保护系统稳定性。

## 5.7 投票排名算法

根据帖子的发布时间以及帖子获得的赞成票数和反对票数实现帖子的实时动态排名功能，是本项目的核心。下面讲解本项目的帖子投票排名算法。

投票分为四种情况：1.投赞成票、2.投反对票、3.取消投票、4.反转投票。

记录文章参与投票的人，谁给哪个帖子投了什么票，将此信息记录至 Redis 数据库。

本项目使用简化版的投票分数，用户点击投票后，更新文章分数：赞成票要加分，432 分、反对票减分，现用  $v$  表示。

当  $v=1$  时，有两种情况：

1. 之前没投过票，现在要投赞成票，更新分数和投票记录，差值的绝对值为：1，帖子+432 分

2. 之前投过反对票，现在要改为赞成票，更新分数和投票记录，差值的绝对值为：2，帖子+432\*2 分

当  $v=0$  时，有两种情况：

1. 之前投过反对票，现在要取消，更新分数和投票记录，差值的绝对值为：1，帖子+432 分

2. 之前投过赞成票，现在要取消，更新分数和投票记录，差值的绝对值为：1，帖子-432 分

当  $v=-1$  时，有两种情况

1. 之前没投过票，现在要投反对票，更新分数和投票记录，差值的绝对值为：1，帖子-432 分

2. 之前投过赞成票，现在要改为反对票，更新分数和投票记录，差值的绝对值为：2，帖子-432\*2 分

投票的限制：

每个帖子发表之日起一个星期之内允许用户投票，超过一个星期就不允许投票了。

1、到期之后将 Redis 中保存的赞成票数及反对票数存储到 Mysql 表中

2、到期之后删除 Redis 记录用户及投票类型的 key：

KeyPostVotedZSetPrefix

核心代码逻辑如图 5-25 所示。

```

46 // VoteForPost 为帖子投票
47 func VoteForPost(userID string, postID string, v float64) (err error) {
48     // 1. 判断投票限制 去redis取帖子发布时间
49     postTime := client.ZScore(KeyPostTimeZSet, postID).Val()
50     if float64(time.Now().Unix())-postTime > OneWeekInSeconds { // 超过一个星期就不允许投票了
51         // 不允许投票了
52         return ErrorVoteTimeExpire
53     }
54     // 2. 更新帖子的分数
55     // 2和3 需要放到一个pipeline事务中操作
56     // 判断是否已经投过票 查当前用户给当前帖子的投票记录
57     key := KeyPostVotedZSetPrefix + postID
58     ov := client.ZScore(key, userID).Val()
59     // 更新: 如果这一次投票的值和之前保存的值一致, 就提示不允许重复投票
60     if v == ov {
61         return ErrVoteRepeated
62     }
63     var op float64
64     if v > ov {
65         op = 1
66     } else {
67         op = -1
68     }
69     diffAbs := math.Abs(ov - v) // 计算两次投票的差值
70     pipeline := client.TxPipeline() // 事务操作
71     incrementScore := VoteScore * diffAbs * op // 计算分数(新增)
72     // ZIncrBy 用于将有序集合中的成员分数增加指定数量
73     _, err = pipeline.ZIncrBy(KeyPostScoreZSet, incrementScore, postID).Result() // 更新分数
74     if err != nil : err
75     // 3. 记录用户为该帖子投票的数据
76     if v == 0 {...} else {
77         pipeline.ZAdd(key, redis.Z{ // 记录已投票
78             Score: v, // 赞成票还是反对票
79             Member: userID,
80         })
81     }
82     // 4. 更新帖子的投票数
83     pipeline.HIncrBy(KeyPostInfoHashPrefix+postID, field: "votes", int64(op))
84     _, err = pipeline.Exec()
85     return err
86 }

```

图 5-25 帖子投票算法函数

这段代码考虑了这样几个因素：

### 1. 帖子的新旧程度 postTime

postTime 的单位为秒，用 unix 时间戳计算。不难看出，一旦帖子发表，postTime 就是固定值，不会随时间改变，而且帖子越新，postTime 值越大。

### 2. 赞成票与反对票的差 diffAbs。diffAbs = 赞成票 - 反对票

表示赞成票与反对票之间差额的绝对值。如果对某个帖子的评价，越是一边倒，diffAbs 就越大。如果赞成票等于反对票，diffAbs 就等于 0。

3. 投票方向 op。op 是一个符号变量，表示对文章的总体看法。如果赞成票居多，op 就是+1；如果反对票居多，op 就是-1；如果赞成票和反对票相等，op 就是 0。



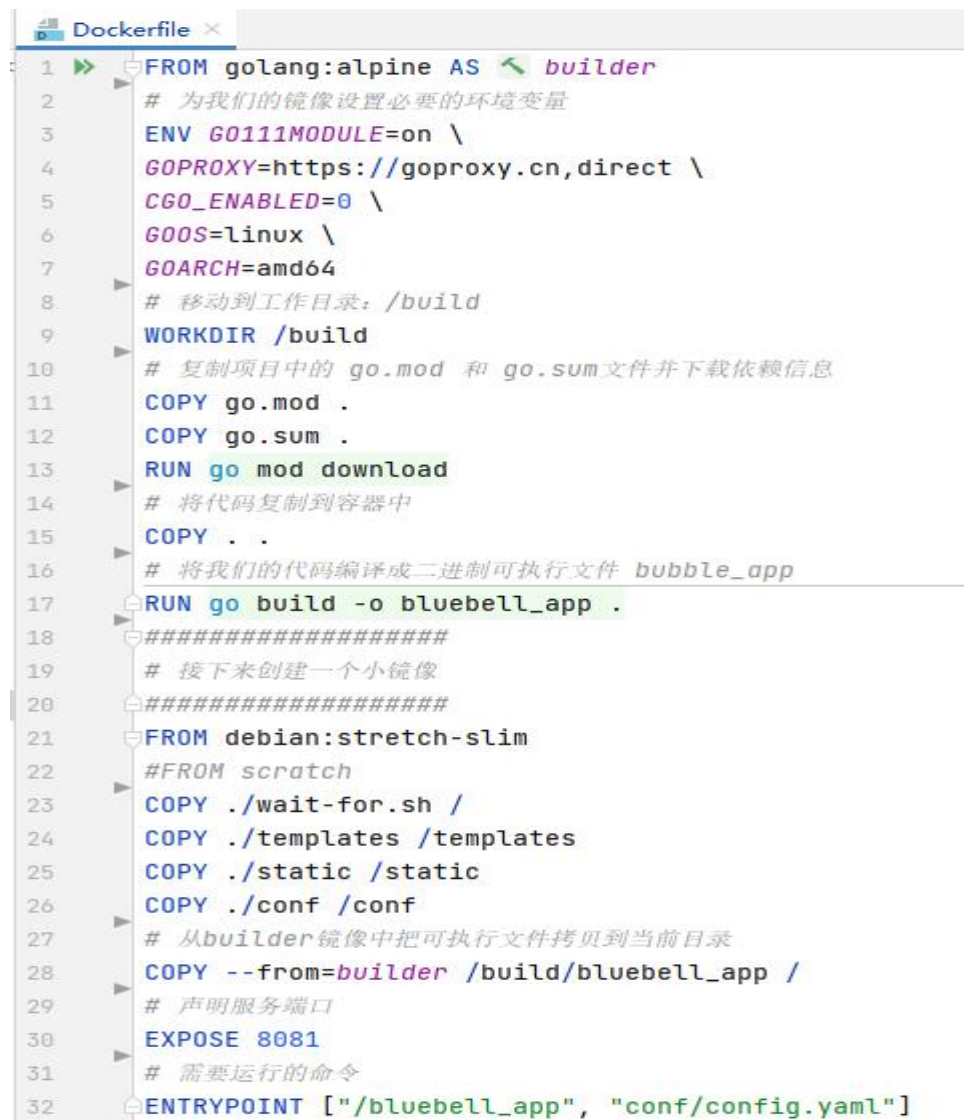
op 的作用是产生加分或减分。当赞成票超过反对票时，这一部分为正，起到加分作用；当赞成票少于反对票时，这一部分为负，起到减分作用；当两者相等，这一部分为 0。这就保证了得到大量净赞成票的文章，会排在前列；赞成票与反对票接近或相等的文章，会排在后面；得到净反对票的文章，会排在最后（因为得分是负值）。

结论就是，本项目的帖子排名，基本上由发帖时间决定，超级受欢迎的文章会排在最前面，一般性受欢迎的文章、有争议的文章都不会很靠前。

## 6 网站部署

### 6.1 编写 Dockerfile

要创建 Docker 镜像（image）必须在配置文件中指定步骤。这个文件通常称之为 Dockerfile。DockerFile 构建镜像文件如图 6-1 所示。



```
1 FROM golang:alpine AS builder
2 # 为我们的镜像设置必要的环境变量
3 ENV GOM11MODULE=on \
4 GOPROXY=https://goproxy.cn,direct \
5 CGO_ENABLED=0 \
6 GOOS=linux \
7 GOARCH=amd64
8 # 移动到工作目录: /build
9 WORKDIR /build
10 # 复制项目中的 go.mod 和 go.sum 文件并下载依赖信息
11 COPY go.mod .
12 COPY go.sum .
13 RUN go mod download
14 # 将代码复制到容器中
15 COPY . .
16 # 将我们的代码编译成二进制可执行文件 bubble_app
17 RUN go build -o bluebell_app .
18 #####
19 # 接下来创建一个小镜像
20 #####
21 FROM debian:stretch-slim
22 #FROM scratch
23 COPY ./wait-for.sh /
24 COPY ./templates /templates
25 COPY ./static /static
26 COPY ./conf /conf
27 # 从builder镜像中把可执行文件拷贝到当前目录
28 COPY --from=builder /build/bluebell_app /
29 # 声明服务端口
30 EXPOSE 8081
31 # 需要运行的命令
32 ENTRYPOINT ["/bluebell_app", "conf/config.yaml"]
```

图 6-1 DockerFile 构建镜像文件示意图

## 6.2 构建 Docker 镜像

在项目目录下，执行下面的命令根据 Dockerfile 构建 Docker 镜像，并指定镜像名称为 bluebell\_app：

```
docker build -t bluebell_app .
```

等待构建过程结束，构建成功后输出如下图 6-2 所示：

```
Successfully built 4d9cbc55779d
Successfully tagged bluebell_app:latest
```

图 6-2 根据 DockerFile 构建镜像成功示意图

## 6.3 运行 Docker 容器

准备好了镜像，但是目前它什么也没做。接下来运行镜像，以便它能够处理请求。运行中的镜像称为容器。执行下面的命令来运行镜像：

```
docker run -d -p 8081:8081 --name bluebell_app bluebell_app
```

-p 标志用于设定端口映射。在此示例中，容器内应用程序使用 8081 端口，因此将其映射到主机的 8081 端口。如果需要将其映射到不同的端口，可以采用 -p \$HOST\_PORT:8081 格式。例如，要将容器的 8081 端口映射到主机的 5000 端口，可以使用 -p 5000:8081。

启动后的容器如图 6-3 所示。

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
56672a50e5ae	bluebell_app	"/bluebell_app conf/..."	8 minutes ago	Up 8 minutes	0.0.0.0:8081->8081/tcp	bluebell_app
dac1eb92dad	redis	"docker-entrypoint.s..."	6 days ago	Up 6 days	0.0.0.0:6379->6379/tcp	myredis
fab81f5cc283	rabbitmq:3	"docker-entrypoint.s..."	9 months ago	Up 9 months		rabbit
3fe87e3340b1	mysql:latest	"docker-entrypoint.s..."	18 months ago	Up 25 hours	0.0.0.0:3306->3306/tcp, 33060/tcp	root_mysql_1
f87cfab76142	nginx:latest	"docker-entrypoint.s..."	18 months ago	Up 18 months	0.0.0.0:80->80/tcp	root_nginx_1
filaa2688039	centos7_mvn_git_java8	"/bin/bash"	18 months ago	Up 18 months		blog-api

图 6-3 Docker 容器示意图

## 6.4 访问网站应用

现在，可以通过访问服务器的 IP 地址和相应的端口来访问应用程序。例如，如果服务器 IP 地址是 47.93.20.204，那么可以通过访问`http://47.93.20.204:8081/`来查看前端页面，如下图 6-4 所示。

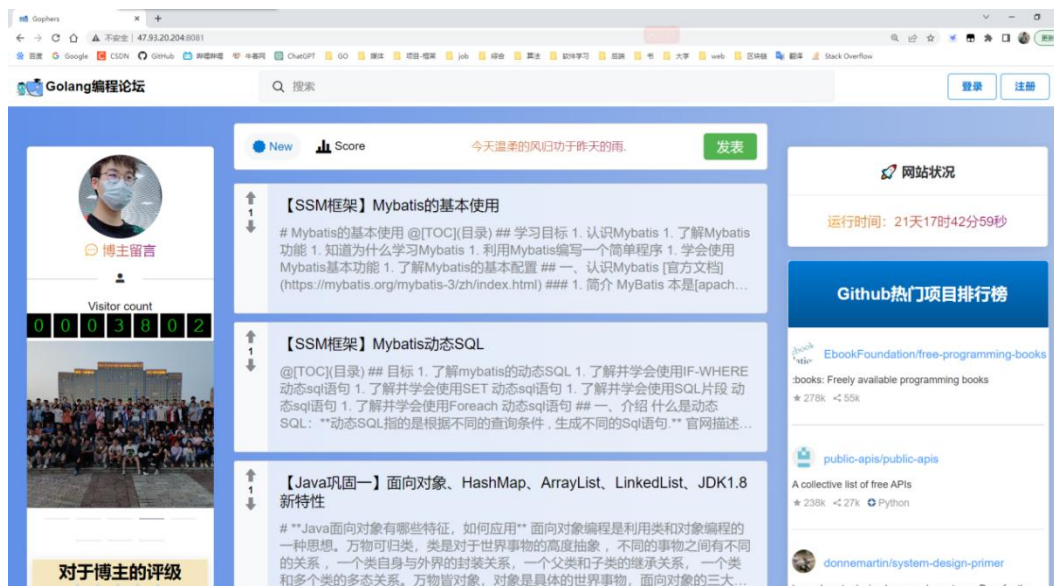


图 6-4 前端网页

通过访问`http://47.93.20.204:8081/api/v1`来访问后端 API,如下图 6-5 所示。

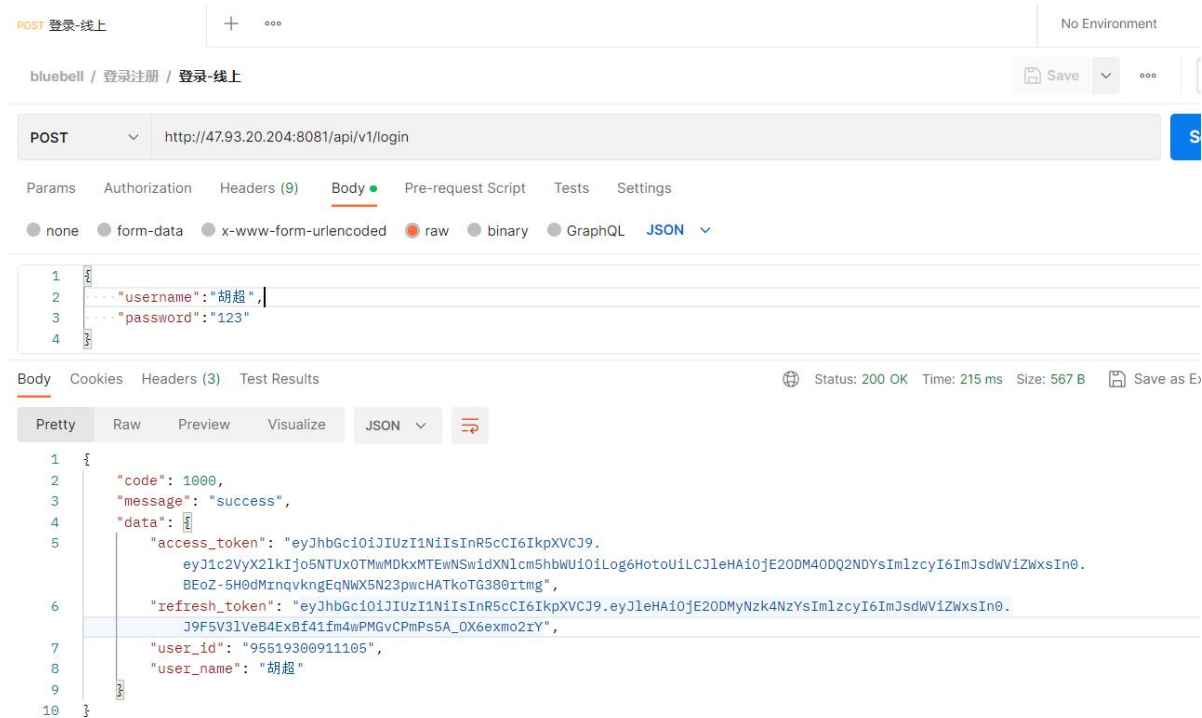


图 6-5 线上接口测试成功示意图

图 6-5 所示，接口返回正常，网站部署成功。

## 7 结论

本文主要研究了基于 Golang 和 Vue.js 的博客论坛设计与实现。通过深入分析了 Golang 的优势、Gin 框架的特点和 Vue.js 的优势，我们成功地设计和实现了一个功能齐全、性能优越的博客论坛系统。

在本文中，详细介绍了后端和前端的设计与实现。后端采用 Golang 语言和 Gin 框架，实现了用户登录注册、文章管理、点赞管理、分类以及全文搜索等功能。前端采用 Vue.js 框架，实现了用户注册与登录、文章列表展示、文章详情展示、发表与编辑文章等页面。

本项目的实现充分体现了 Golang 和 Vue.js 技术的优势。Golang 语言因其高性能、简洁的语法和丰富的库支持，使得后端开发变得高效而稳定。Vue.js 则为前端开发提供了轻量、易学、高效的特点，有助于提升开发速度和用户体验。此外，本项目的实践过程也为开发者提供了宝贵的经验，有助于加深对 Golang 和 Vue.js 技术的理解，为今后的 Web 开发奠定基础。当然，本项目还存在一定的改进空间，例如可以尝试引入更多的功能，例如用户间的私信功能、实时通知功能等；在性能优化方面，可以尝试引入缓存机制、分布式存储等技术，提高系统的性能和可扩展性。

总之，本文通过设计和实现一个基于 Golang 和 Vue.js 的博客论坛系统，充分展示了这两种技术的优势，为 Web 开发者提供了一个实践的范例。我们相信，随着 Golang 和 Vue.js 技术的不断发展和完善，它们将在 Web 开发领域发挥越来越重要的作用。

## 参考文献

- [1] 李洋,刘婷. 基于 MySQL 的家电回收管理系统的数据库设计[J]. 科技与创新,2023(03):141-143+146
- [2] 王福兴,周明辉. 基于 Golang+Gin 的技术运维系统设计与实现[J]. 现代电视技术,2022(10):134-137
- [3] 苏佳旭,白燕,温晓东. 基于 Web 前端与 MySQL 数据库的自然灾害应急信息共享平台设计与实现[J]. 电脑知识与技术,2023,19(05):74-77
- [4] 张辉,李鹏. 基于 Golang 的跨平台蜜罐框架系统的设计与实现[J]. 现代计算机,2022,28(21):87-91
- [5] 顾雅枫,葛静微. 基于 MVC 的实验室低值易耗品管理系统的开发与实现[J]. 现代信息科技,2023,7(06)
- [6] 王晓峰. Golang 语言实现的流水线模型[J]. 电子技术与软件工程,2020(01):53-54
- [7] 齐洋,原变青,刘颖,等. 基于 Gin 和 Vue.js 的作业管理系统的设计[J]. 信息技术与信息化,2022(10):103-105+110
- [8] 徐健. 基于 Go 和 Vue.js 的体育选课系统的设计与实现[J]. 电脑知识与技术,2022(08)
- [9] 卢云霞. 浅谈个人博客网站的设计与实现[J]. 内蒙古科技与经济,2021(17):78-79+81
- [10] 丁岚,范开勇,王英明. 基于 Golang 的网络爬虫系统设计与实现 [J] 电脑编程技巧与维护. 2019,第 006 期
- [11] 王铮清,刘壮峰. 基于 Go 语言的内容管理系统的设计与实现[J]. 电脑知识与技术,2022,18(24)
- [12] 宋云奎,吴文鹏,赵磊,等. 基于 Redis 的分布式数据存储方法[J]. 计算机产品与流通,2020(08)
- [13] 肖睿. 基于 Gin 框架的营销活动公共类库的设计与应用[D]. 武汉:华中科技大学,2019
- [14] 王雄. Golang 或将统治人工智能下一个 10 年[J]. 计算机与网络,2020(20)
- [15] Wu Daiwen. The Application and Management System of Scientific Research Projects Based on PHP and MySQL[J]. Journal of Interconnection Networks,2022,22(Supp02)

## 致谢

在此，我要对所有在《基于 Golang+Vue 的博客论坛的设计与实现》论文创作过程中给予我帮助和支持的人表示衷心的感谢。

首先，我要向我的导师蔡磊教授表达诚挚的感激。在论文的撰写过程中，蔡磊教授以严谨的治学态度和深厚的学术造诣给予我指导和启发。他的悉心教导使我在项目设计和论文撰写上受益匪浅，也使我对计算机技术有了更深的认识和理解。

作为一名通信工程专业的学生，我对计算机领域充满热情。在项目开发过程中，我自学了 Golang、Vue、Gin 框架等技术栈。在这里，我要感谢那些无私分享知识和经验的网络资源和社区成员，正是他们的贡献为我提供了宝贵的学习资料和技术支持，使我能够顺利完成项目的开发和论文的撰写。

同时，我要感谢我的同学和朋友们。在项目开发过程中，他们给予了我许多鼓励与支持，让我在面临困难和挑战时不断追求进步。他们的陪伴让我在学术和生活上不断成长。

最后，我要感谢我的家人。他们在背后默默奉献，给予了我无尽的关爱和支持，让我能够无忧无虑地投入到学术研究中，完成这篇论文。在此，我再次对所有帮助和支持过我的人表示衷心的感谢！