****

**本 科 毕 业 论 文（设计）**

|  |  |
| --- | --- |
| 课题名称 | 基于微服务博客系统后端设计与实现 |
| 学 院 | 计算机科学与教育软件学院 |
| 专 业 | 软件工程 |
| 班级名称 | 软件162 |
| 学生姓名 | 吴晓明 |
| 学 号 | 1607200084 |
| 指导教师 | 吴晓明 |
| 完成日期 | 2020年4月30日 |

教 务 处 制

基于微服务博客系统后端设计与实现

摘要 博客系统专为广大学生用户提供优质的知识分享的平台，广大学生用户可以更方便地在平台上获取优质的知识，也可以分享自己的知识和见解，促进平台健康地发展，以此提高网站的价值。为了满足博客系统的要求，系统采用前后端架构分离，降低耦合，分工明确，前端使用Angular框架开发页面，使用Nginx解析前端的静态资源，后端使用Go语言进行开发，并采用了微服务架构，对模块划分服务，极大的降低耦合并提高拓展性，存储方面使用了Redis作为缓存，使用MySQL用于持久化存储。

论文首先论述博客的历史发展过程和系统开发背景以及设计目标，然后对系统开发中关键技术做了分析介绍，比如Redis，GoMicro框架等，在总体系统设计与实现中，对博客系统的整体架构做了讲解，从接入层到存储层逐一分析介绍如何设计实现的，然后再对后端系统架构讲解，讲解如何使用微服务架构进行设计，并对其中关键技术详细分析，并配以具体代码分析讲解。在后端系统模块设计中，分析后端重要的模块，如何设计和实现的，使用流程图和具体代码分析其中关键的技术，最后在数据库设计中，讲解了用了哪些缓存的数据结构，再讲解数据库表的设计。

关键词 前后端分离；微服务；服务发现与注册；缓存

ABSTRACT The blog system is designed to provide a high-quality knowledge sharing platform for the majority of student users. The majority of student users can more easily obtain high-quality knowledge on the platform, and they can also share their knowledge and insights to promote the healthy development of the platform to improve the website ’s value. In order to meet the requirements of the blog system, the system uses a front-end and back-end architecture to separate, reduce coupling, and make the division of labor clear. The front-end uses the Angular framework to develop pages, the Nginx is used to analyze the static resources of the front-end, the back-end uses Go language for development, and the micro-service architecture is adopted. The module division service greatly reduces coupling and improves scalability. Redis is used as a cache for storage, and MySQL is used for persistent storage.

The thesis first discusses the historical development process and system development background and design goals of the blog, and then analyzes and introduces key technologies in system development, such as Redis, GoMicro framework, etc. In the overall system design and implementation, the overall architecture of the blog system is done After explaining, from the access layer to the storage layer, analyze and introduce how to design and implement, and then explain the back-end system architecture, explain how to use the micro-service architecture for design, and analyze the key technologies in detail, with specific code analysis explain. In the back-end system module design, analyze the important back-end modules, how to design and implement them, use flowcharts and specific codes to analyze the key technologies, and finally in the database design, explain which cached data structures are used Explain the design of database tables.

KEY WORDS Front and rear separation；Microservices；Service discovery and registration；Cache；

目录

[1.前 言 5](#_Toc1955211072)

[1.1 课题背景和意义 5](#_Toc437720710)

[1.2 系统开发的目标 5](#_Toc1640481995)

[1.3 文章组织结构 5](#_Toc38346132)

[2. 项目技术背景和研究工作 7](#_Toc238346424)

[2.1 Go的特性 7](#_Toc831346513)

[2.2 Go Micro微服务框架 7](#_Toc912236609)

[2.3 REDIS 8](#_Toc1074931530)

[2.4 本章小结 8](#_Toc1741786146)

[3.系统分析与设计 9](#_Toc1850163565)

[3.1 系统业务流程分析与后端功能模块划分 9](#_Toc135828395)

[3.2 系统架构设计 10](#_Toc92718004)

[3.3 系统架构具体实现 13](#_Toc1385849153)

[3.3.1 Ningx配置 13](#_Toc359079109)

[3.3.2 前端与后端通信 14](#_Toc613536893)

[3.3.2 注册微服务 15](#_Toc1645571404)

[3.3.3 微服务通信 18](#_Toc1824180962)

[3.4 本章总结 18](#_Toc1532883762)

[4.后端主要模块设计与实现 19](#_Toc1963558522)

[4.1 日志模块 19](#_Toc1146875805)

[4.1.1 配置日志参数 19](#_Toc1875922810)

[4.1.2 日志缓冲 20](#_Toc1427246063)

[4.2 用户模块 21](#_Toc332243851)

[4.2.1 API设计与实现 21](#_Toc564921557)

[4.2.2 用户信息安全与鉴权实现 22](#_Toc611405112)

[4.3 私信模块 25](#_Toc176466489)

[4.3.1 API设计与实现 25](#_Toc197364116)

[4.3.2 点对点聊天实现 26](#_Toc1383946644)

[4.4 点赞模块 26](#_Toc595865051)

[4.5 文章模块 28](#_Toc987666196)

[4.5.1 API设计与实现 28](#_Toc1804648509)

[4.5.2 统计“真实”访问量 29](#_Toc1815944182)

[4.6 定时器模块 30](#_Toc536275710)

[4.7 本章小结 31](#_Toc196991511)

[5. 数据库设计 32](#_Toc1564025350)

[5.1 缓存表设计 32](#_Toc1374218170)

[5.2 数据库表设计 34](#_Toc298159705)

[6.结论 38](#_Toc1090813484)

[参考文献 39](#_Toc234331149)

[致谢 40](#_Toc2066096292)

# 1.前 言

## 1.1 课题背景和意义

写博客是非常高效的学习方式，是把知识当作一种技能来认真学习，更好地让自己的知识和思考通过提炼的方式表达出来，它是一种知识重建的过程，可以对知识有更深刻地理解，非常有效地锻炼抽象思维和逻辑思维，并提高自己的写作水平，所以在本系统中，构建了一个给用户很好地记录和分享知识的信息化管理的平台，用户发表的文章都会及时保存后台数据库中，保证不会丢失数据，用户可以随时随地在不同平台上阅读博客系统上其他用户的文章或者即兴发表文章，相对于书本记录效率会有极大地提高，用户可以通过阅读其他用户分享的知识作为自己的灵感，系统还提供给用户可以发表私有的文章的功能，给用户创作的隐私空间。

## 1.2 系统开发的目标

博客系统的前端是基于Angular进行开发，后端基于Go进行开发，数据库存储使用了Redis和MySQL，整体架构基于前后端分离架构，后端基于微服务架构，使用Etcd作为服务发现与注册，使用了GoMicro框架，将模块变成一个微服务，使用RPC可以实现服务之间互相通信，整体架构极大低提高了系统的灵活性，可维护性，开发效率，并且降低耦合，确保了后端的稳定，高性能，易扩展等特性。

## 1.3 文章组织结构

根据课题研究的具体内容，本论文主要分为五章:

1. 引言，介绍当前互联网博客应用发展情况，博客发展中遇到的问题以及如何设计改进，最后，对本文的基本组织结构等内容作了简要介绍。
2. 相关技术背景和研究工作，主要介绍项目中所采用的架构，所使用的语言技术栈和组件。
3. 系统设计架构设计与实现，主要介绍项目中的架构如何设计，如何实现，业务流程是怎样的，然后后台如何划分模块。
4. 主要功能模块设计与实现，主要介绍后台系统主要的模块是如何实现，流程是怎样，用到了哪些技术。
5. 数据库设计，主要介绍了缓存表如何设计，数据库表如何设计。

# 项目技术背景和研究工作

## 2.1 Go的特性

Go语言也称为Golang，是由Google公司开发的一种强静态类型、编译型、并发型、并具有垃圾回收功能的编程语言。[1]

* 简单易学：Go语言语法非常简单，上手快，Go的关键字很少，但不影响Go语言的表达能力，几乎支持大多数其他语言的特性。
* 丰富的标准库：Go内置大量的库，尤其是网络库非常地强大，几行代码就可以编写出一个HTTP后台服务。
* 垃圾回收：Go内置垃圾回收[2]，实现了有并发标记、三色标记和写屏障等，尽可能降低STW时间，以便不影响用户的逻辑情况下更好地工作。
* 内存分配器：Go使用了tcmalloc，它就是为了并发而设计的高性能内存分配组件，Go的内存分配器保留了tcmalloc的原始架构，每个线程都有一个cache，可以为线程无锁分配内存，central是用于在不同线程间共用的内存单元，heap管理着大块内存，可以切分成不同大小的内存块。
* 并发模型：Go语言层面支持并发，使用Go关键字就可以启动一个协程[3]，协程相对于线程开销更轻量、开销更小、性能更高，而且操作起来非常的简单。

## 2.2 Go MICRO微服务框架

Go Micro是分布式系统开发的框架[4]，提供了分布式系统开发的核心功能，比如服务注册与发现[5]、负载均衡、RPC协议、负载均衡、可插拔接口和支持多种消息编码等功能。

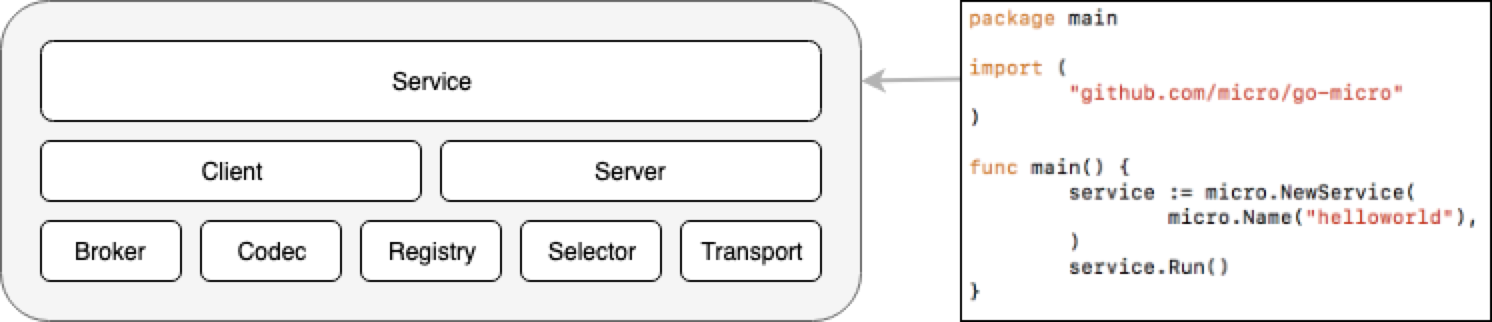


图2.1 Go Micro架构示意图

## 2.3 REDIS

Redis 是开源的，BSD许可的，内存键值存储系统。Redis有许多中数据结构，比如String，Hash，List，Set，Sorted Set，Bitmap和hyperloglog。在这些类型上执行的操作都是原子操作，比如添加一个String类型，增加Hash中的值，计算交集、并集和差集等。根据使用场景，可以通过每个一段时间转储数据到磁盘中，或者通过写命令到日志实现持久化。Redis还有其他的特性，比如事务、订阅和发布、支持Lua脚本，设键值TTL和LRU回收等。[6]

## 2.4 本章小结

本章主要介绍了项目的背景技术，后端所用的语言技术栈是Go，并且使用了Go生态中的GoMicro框架，用于搭建后端的微服务框架，这个框架集成多种微服务中需要的功能，然后还介绍后端使用的缓存组件Redis，它可以显著地提高后端的处理速度。

# 3.系统分析与设计

## 3.1 系统业务流程分析与后端功能模块划分

* 系统业务流程分析

系统只有两个身份：游客和用户，游客只能访问没有受限的功能，如果游客想访问所有功能，那么只能需要注册或登录成为用户，因为前端会进行拦截，并且后端也会进行用户身份校验，无法通过调用接口直接访问资源，游客身份只能简单的浏览，比如文章，评论，用户个人信息的浏览，而用户身份可以对自己文章进行管理，可以分类，可以发布，还可以设置成隐私，不会被其他用户看到，文章可以被其他用户点赞，会实时记录被点赞的数量，文章被访问时会被记录浏览的次数，文章发布出去还可以再次修改发布出去，文章还可以进行分类，有效地管理文章，用户可以修改自己的信息，比如修改头像，昵称等，用户信息会展示用户发表的文章数，获赞数，访问量等信息，用户可以关注其他的用户，然后出现在自己的关注列表内，可以方便快速查找自己关注的用户并跳转到用户的主页，用户可以私信用户，可以与用户实时的聊天，如果用户不在，那么当用户上线，用户可以查看自己未读的消息，也可以查看历史消息，用户可以对文章发表评论，并且可以对评论进行点赞，如果觉得自己评论不恰当，可以删除自己评论，重新发表评论。

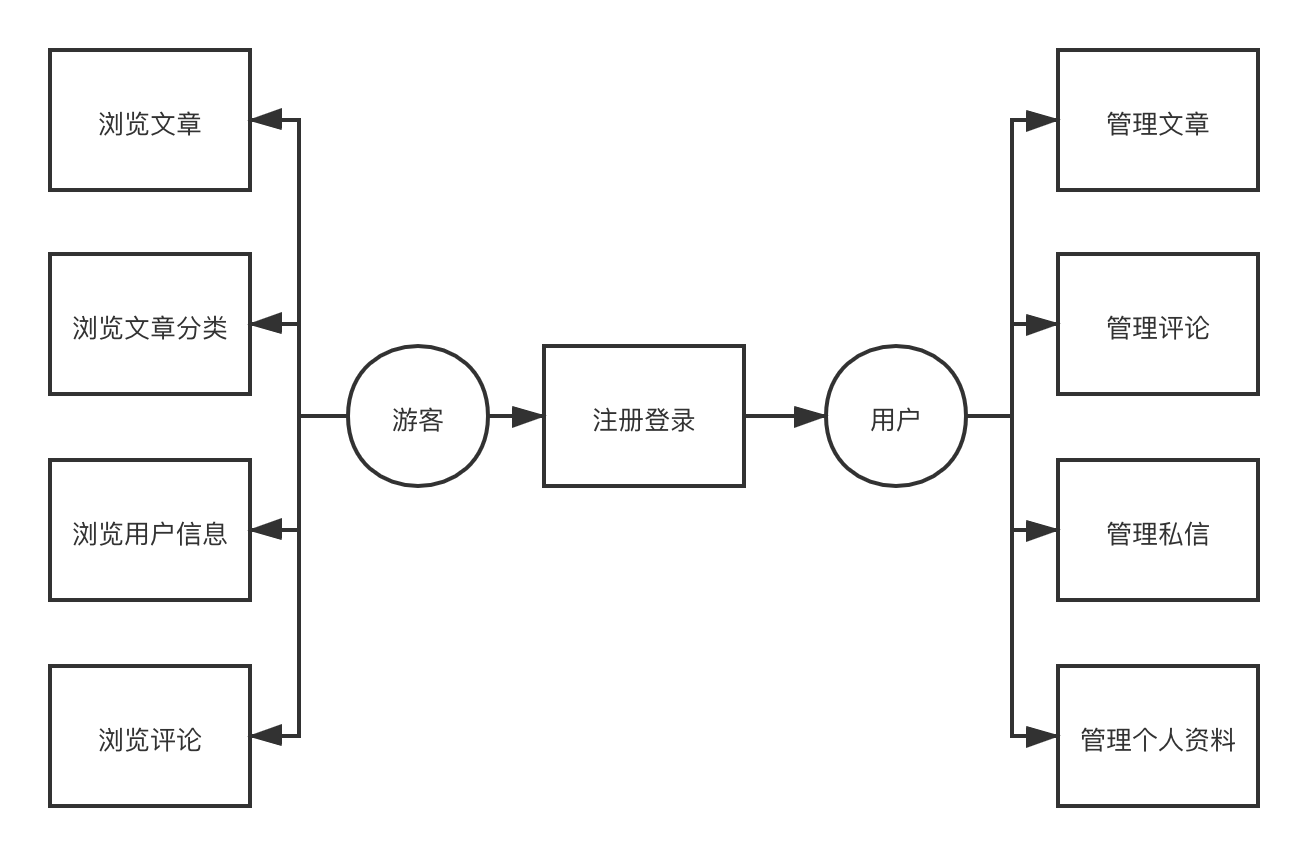


图3.2 业务流程图

* 功能模块划分

根据上述业务流程分析，后端可以划分七个模块：

1. 用户模块：负责用户登陆、注册、基本信息、关注和取关等功能
2. 文章模块：负责文章发布、浏览、管理（增删查改）等功能
3. 评论模块：负责对文章评论，回复评论等功能
4. 聊天模块：负责发送消息，查看未读消息，浏览历史消息等功能
5. 点赞模块：负责对文章进行点赞等功能
6. 文件夹模块：负责对文章进行分类管理
7. 定时器模块：负责将文章访问量、点赞量等数据同步到数据库中

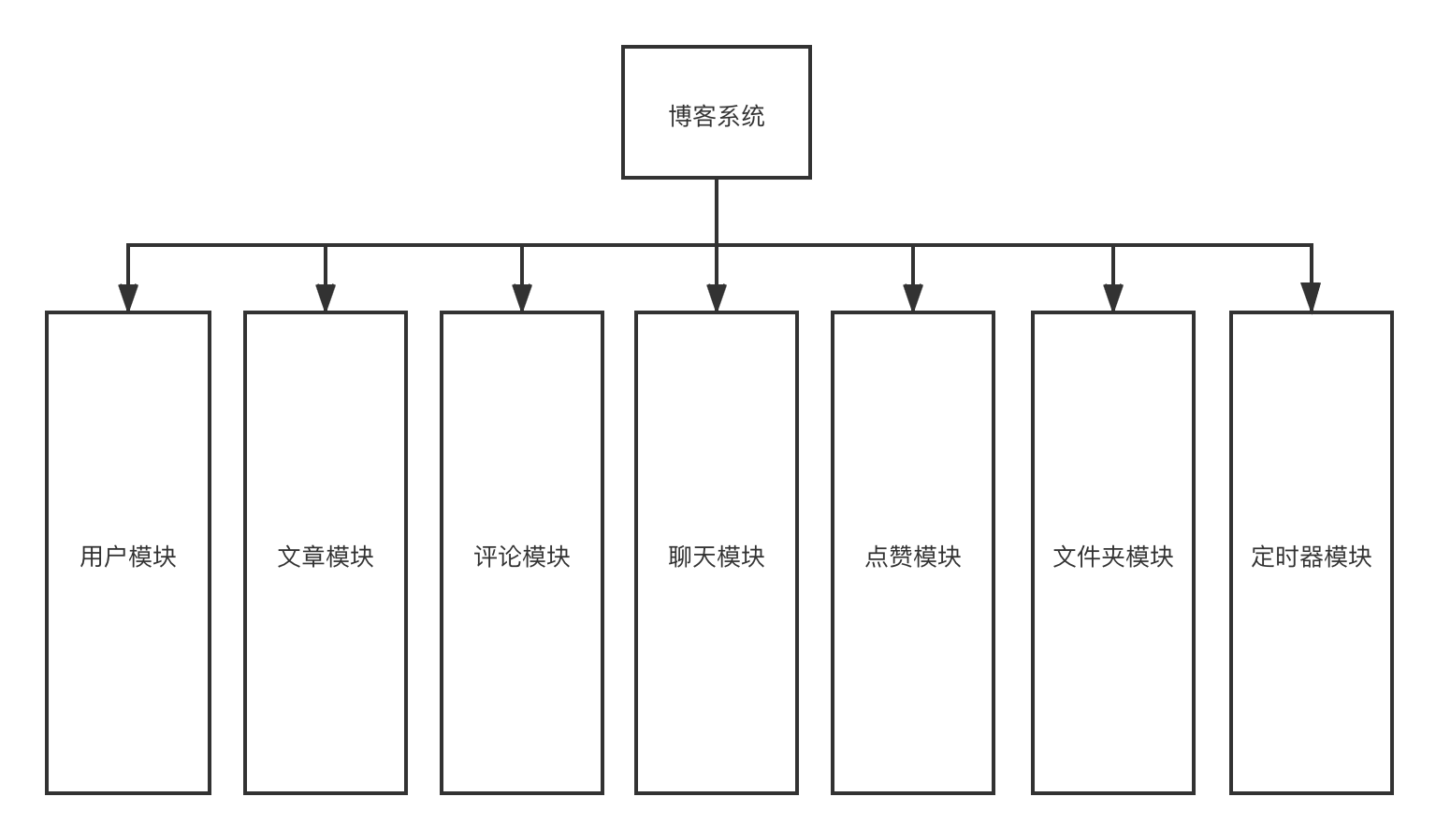


图3.3 后端功能模块划分图

### 3.2 系统架构设计

系统总体架构采用了前后端分离架构，后端系统采用了微服务架构，如下图3.1所示：

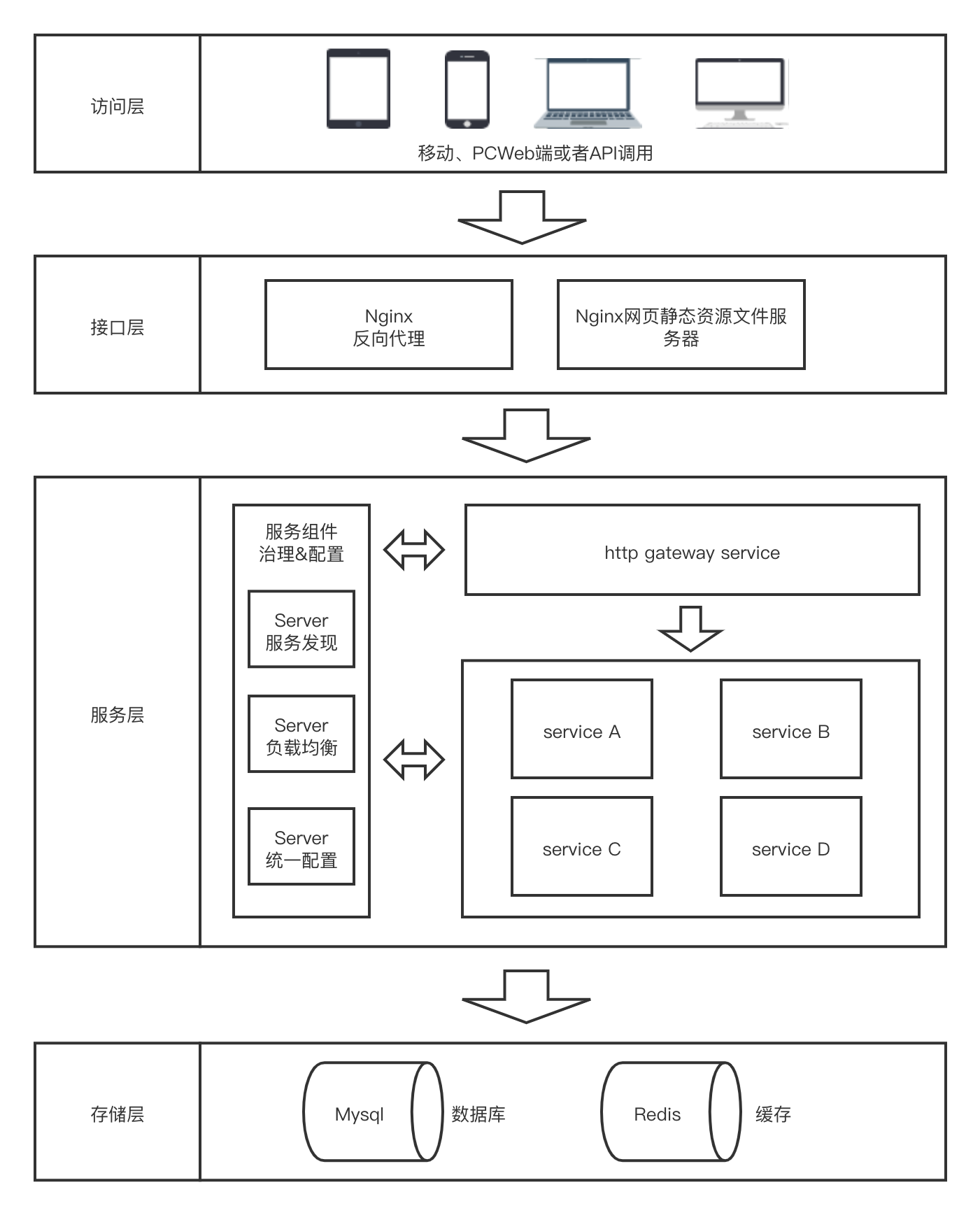


图3.1 系统架构图

* 访问层

访问层既是展示在用户面前的Web页面，即HTML，CSS及JavaScript等静态资源。这些静态资源是由Angular打包编译优化生成的，Angular与其他前端框架语言不同，主要使用的是Typescript语言，这个Typescript语言不能直接被浏览器执行，所以Angular需要将ts文件编译优化转化成js文件。[7]Angular打包生成的静态资源，是放在后台服务器中，由Nginx进行托管，用户打开浏览器，输入URL会请求到接口层Nginx上，Nginx就将静态资源文件返回回去，交给浏览器进行解析，前端页面就展示在用户面前，用户点击页面的交互内容，浏览器会发送一系列请求，请求拉取相应资源，这里也是由接口层Nginx进行负责，如果是请求后端服务，Nginx会进行反向代理，将请求的地址修改成后台服务器真正的地址，使用反向代理的原因主要是为了安全，可以有效屏蔽后台服务的地址，避免受到恶意攻击。

* 接口层

接口层是外网与后台服务交互的唯一入口，这里主要由Nginx负责，Nginx会暴露主要端口，比如HTTP默认的80和HTTPS默认的443端口，Nginx会根据请求的URI前缀进行匹配处理，比如前端页面/posts前缀，Nginx就会拉去静态资源文件给浏览器进行渲染展示，比如后端服务接口/api前缀，Nginx就会反向代理[8]请求后台服务进行处理，Nginx还设置了对回包响应做gzip压缩，这样可以减少回包大小，减少网络传输时间，由其对静态资源文件的压缩比率达到了50%，用户输入URL到展示页面的速度极大地加快。

* 服务层

服务层是后台服务对外提供功能的集合，后台采用了先进的架构思想，即微服务架构，微服务是一种开发软件的架构和组织方法，其中软件由通过明确定义的API进行通信的小型独立服务组成。实现微服务最重要的就是解决通信问题，这里后台使用了RPC协议，每个服务定义了一个结构体和一系列方法（Go语言没有类），通过RPC协议就可以轻松的用“结构体名.方法名”格式调用指定的服务的指定方法，但是使用协议前提是需要知道服务地址，虽然可以在调用RPC协议的时候指定地址，但是当服务数量众多，服务地址变更等因素，这里调用协议的地址需要不停地更改，降低了灵活性，违背了微服务架构的思想，所以这里使用了Etcd作为服务中心，它是分布式可靠键值存储，使用Raft算法保证高可用性，而且Etcd可以进行服务发现与注册，探活，统一下发配置等功能。这里每个服务会去Etcd上报自己的服务名，ip和port等关键信息，Etcd会维护这张表，这样服务调用其他的服务的时候，只要需要向Etcd拉取最新的表，就可以知道调用服务的地址，提高了灵活性，既是服务地址变更，Etcd也会及时更新表的数据，确保服务可用，Etcd会为每个服务维护一个表，这样服务可以快速拓展，发展成集群，然后由调用方自实现负载均衡，调用集群内的服务。

* 存储层

存储层是存储相关数据并可对数据进行操作的集合。存储层主要由两个组件构成一个是Redis和MySQL，这里使用Redis内存数据库，它有丰富的数据结构，可以满足很多需求，并且它对数据操作非常快，因为都是在内存上操作，但是内存相对磁盘的大小来说非常的小，并不能把所有数据都放入内存中去，所以只能将热点数据才放入Redis中，并配上过期时间，这样不仅可以提高数据的处理速度，而且可以避免占满内存，Redis还可以削峰，比如点赞这个频繁的操作，可以在Redis中先对数据进行操作，有效避免MySQL建立连接过多，导致服务不可用，然后为了避免数据的不一致，专门有一个定时器服务，会将Redis缓存中的数据同步到MySQL中，保证数据的一致性。

## 3.3 系统架构具体实现

### 3.3.1 Ningx配置

Nginx是一个轻量级的高性能的网络服务器，也可以用做反向代理、负载均衡器和HTTP缓存，它是一个开箱即用的工具，只需要修改nginx.conf配置文件就可以轻松配置。[9]

nginx.conf

#设定http服务器

http {

...

server{

listen 80; #监听80端口

root /proj/test/dist/; #设置根目录

location /api { # 匹配url的前缀为/api时，进入这个分支处理

proxy\_pass <http://127.0.0.1:8084;> #反向代理，将请求地址修改

}

location /md { # 匹配url的前缀为/md时，进入这个分支处理

alias /proj/NiceFish-Go/BlogServer/file/image/; # 使用alias路径替换url前缀，一般用于拉取图片等文件资源

  }

location / { # 上述前缀都没有匹配到，则进入到该分支，这个分支主要用于解析前端静态资源文件

index  index.html ;

try\_files $uri $uri/ /index.html; # 重定向url，将静态资源文件进行解析并响应

  }

  }

...

}

以上配置，主要讲述了Nginx监听80端口，并对url前缀进行匹配，匹配到的前缀可以设置反向代理，也可以设置成拉取文件资源，还可以对前端静态资源解析获取。

### 3.3.2 前端与后端通信

前后端交互是通过HTTP协议[10]进行交互，使用JSON传输数据，后端首先需要监听某个端口，然后绑定URL前缀和处理函数，这样当HTTP请求过来的时候，匹配URL前缀，回调处理函数，而前端则需要构造HTTP请求包发送请求到后端。

* 后端具体实现

...

r.HandleFunc("/user\_register"，middlware.MiddleWareFunc(handle.HandleRequest, handle.HandleHeader, handle.HandleBody, user.RegisterValid)).Methods(http.MethodPost, http.MethodOptions) // 用户注册API，匹配到/user\_register前缀，则进入右边注册的一些系列处理函数，比如用户鉴权，Header跨域处理等，并且只对外暴露Post和Options请求，其他方法请求无效。[11]

...

// 启动一个HTTP服务器，并监听8084端口，设置了一些读写超时时间，避免请求长时间阻塞消耗系统资源。

server := &http.Server{

Addr: "127.0.0.1:8084"

Handler: newmux(),

ReadTimeout: 30 \* time.Second,

WriteTimeout: 30 \* time.Second,

IdleTimeout: 20 \* time.Minute,

}

server.ListenAndServer();

* 前端调用后端API实现

```

// angular 提供post的API，填写后端URL，请求的body，和headers即可发送一个POST请求。

this.httpclient.post<T>(url, json, {

Headers: header,

Params: params,

})

```

### 3.3.2 注册微服务

在微服务架构中，服务之间要能互相通信，首先服务需要注册，其他服务才能获取被调用服务的地址，才可以进行通信。

* 首先编写pb文件定义方法和结构体

click.proto

Service ClickService {

// GetPostClicks 获取文章点赞数量

rpc GetPostClicks (GetPostClicksReq) returns (GetPostClicksRsp){

}

...

}

...

// GetPostClicksReq 请求的结构体

Message GetPostClicksReq{

Int64 Articleid = 1;

...

}

// GetPostClicksRsp 回包响应的结构体

Message GetPostClicksRsp{

Int64 Likes= 1;

...

}

...

以上pb文件的代码，首先是定义了一系列对外暴露的方法，还有方法需要的结构体类型，然后这些pb文件需要转化为.go文件才能被项目所用。

* 通过插件将proto文件转化为.go文件

proto --proto\_path. --micro\_out=. --go\_out=. click.proto

1. proto\_path 指明proto文件的路径为当前路径。
2. micro\_out 使用go-micro框架的插件，可以生成用于.micro.go代码，并指明路径为当前路径。
3. go\_out 生成.go代码，指明路径为当前路径。

然后会生成两个文件：click.micro.go 和 click.pb.go文件：

click.micro.go文件是go-micro注册服务的代码模版，需要实现它提供的接口，并调用RedisterClickServiceHandler函数进行注册。

```

// ClickService 服务的API，业务需要实现以下方法

type ClickServiceHandler interface {

GetPostClicks(context.Context, \*GetPostClicksReq, \*GetPostClicksRsp) error

...

}

// 将实现的handler与server进行绑定，当有请求过来时可以回调handler里的方法

func RegisterClickServiceHandler(s server.Server, hdlr ClickServiceHandler, opts ...server.HandlerOption) error {

...

return s.Handle(s.NewHandler(&ClickService{h},opts...))

}

```

click.pb.go文件是将pb文件中定义的结构体转化为go语言中定义的结构体。

* 服务注册

首先服务配置相应参数，比如服务名，服务注册中心的地址，与服务注册中心探活间隔，可以开启tls等，然后服务监听指定的端口，调用注册函数进行绑定方法，然后服务启动并保持监听。

```

// new service 结构体

service := micro.NewService(

micro.Name(dconfig.ServiceName\_Click), // 设置服务名

micro.Registry(utils.DefaultRegistry()), // 设置服务中心的地址

micro.RegistryTTL(time.Second\*30), // 探活超时时间

micro.RegistrInterval(time.Second\*10), // 探活间隔

micro.Transport(

transport.NewTransport(

transport.Secure(true), // 设置tls

Transport.TLSConfig(utls.TLSConfig()), // 设置tls相关配置，比如证书的位置

)

),

)

service.Server().Init(

Server.Address("127.0.0.1:8588"), // 服务监听本机的8588端口

)

pb.RegisterClickServiceHandler(service.Server(), controller.NewClickService()) // 注册服务

Service.Run() // 启动服务

```

以上代码简要介绍了如何注册一个服务。

### 3.3.3 微服务通信

首先从服务中心获取注册的服务节点信息，然后通过RPC协议调用该服务暴露的方法，调用的时候也可以配置，比如设置超时时间，重试次数，设置tls连接，服务中心的地址，设置RPC协议底层的协议成TCP或UDP，默认是TCP。

```

// 这里配置了服务中心的地址

cli = Client.NewCLient(client.Registry(utils.DefaultRegistry()))

// 设置tls，建立安全连接

cli.Init(client.Transport(

Transport.NewTransport(transport.Secure(true),transport.TLSConfig(tlsconf)),

))

// 指定调用的服务名，方法名，请求包的结构体

request:=cli.NewRequest(servname, method, req)

// 指定了调用服务的超时时间，回包的结构体

cli.Call(context.Background(), request, rsp, client.WithRequestTimeout(time.Second\*20))

```

以上代码简要介绍了如何调用一个微服务。

## 3.4 本章总结

博客系统总体架构是前后端分离，前后端各司其职，互不干扰，约定接口通过HTTP协议就可以互相通信交互，降低了耦合，后端架构采用了微服务架构，按照模块划分服务，进一步降低耦合，每个服务是某块功能的集合，分工明确，结构清晰，服务之间通信也非常的简单，使用RPC协议就可以轻松调用其他服务的方法。

# 4.后端主要模块设计与实现

## 4.1 日志模块

### 4.1.1 配置日志参数

日志模块是服务的基石，是每个服务的必要组件，负责打印log，这在排查系统问题的时候非常的有用，根据打印的错误信息，快速排查出问题，后端使用uber开源的zap日志库，它是一个结构化、分级、可配置并且高性能的日志库。

* 结构化：它默认输出为JSON格式，JSON格式非常易读，并且结构分明
* 分级：它可以设置日志的级别，控制打印日志的级别，比如只打印ERROR日志，而INFO、DEBUG、WARNING级别日志不打印，这在生产环境非常有用，因为频繁打印日志，会影响程序性能，而且打印太多日志，不易于排查问题。
* 可配置：它可以配置打印日志的时间，配置打印的位置，哪个代码文件，哪一行，在哪个函数，它还可以自定义core，然后后端使用了lumberjack日志库自定义uber的core，lumberjack是一个日志分割器，可以按照天、大小进行分割日志，这样就不会导致日志的大小越来越大。
* 高性能：zap日志库相比其他日志库大幅减少反射和分配内存的过程，所以它的性能非常优异。

```

// new 一个zap log

func newLogger(serviceName string, logpath string, bufferChan int, bufferDuration int, isProduction bool) \*zap.Logger {

 EncoderCfg := zapcore.EncoderConfig{

  TimeKey:    "Time", // 打印输出时间的字段叫Time

  LevelKey:   "Level", // 打印日志级别的字段叫Level

  NameKey:    "Logger",

  CallerKey:  "Caller", // 打印程序调用行的字段叫Caller

  MessageKey: "Message", // 打印程序信息的字段叫Message

  LineEnding: zapcore.DefaultLineEnding, // 打印日志默认换行

  EncodeTime: func(t time.Time, enc zapcore.PrimitiveArrayEncoder) {

   enc.AppendString(t.Format("2006-01-02 15:04:05"))

  },

  EncodeCaller: zapcore.ShortCallerEncoder,

 }

...

 return zap.New(zapcore.NewCore(encoder, writers, dyn), zap.AddCaller())

}

```

以上代码是项目中配置zap日志库的部分代码。

### 4.1.2 日志缓冲

后台还对打印日志做了缓冲处理，当日志条数积累一定数量或者定时器触发才打印输出日志，这样可以有效避免程序因为打印过多日志阻塞，从而提高性能。

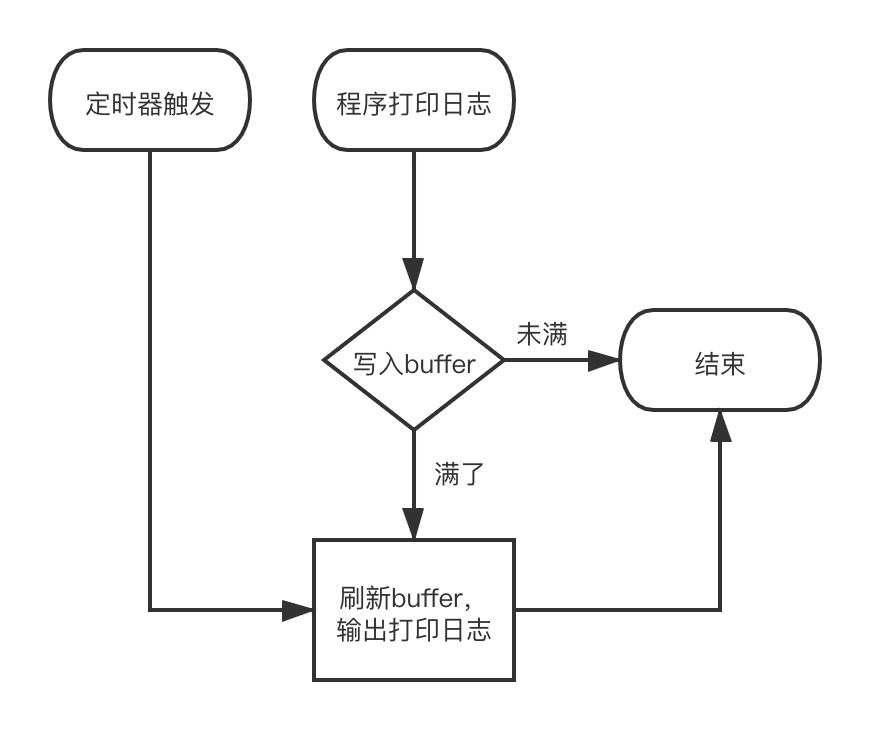


图4.1 日志缓冲流程图

## 4.2 用户模块

### 4.2.1 API设计与实现

用户模块提供了许多功能，比如用户注册、登录，修改密码，查看个人资料，关注用户或取消关注用户等功能，后台将这些功能归类到一个模块内，并划分为一个服务，然后对外提供服务，降低耦合。

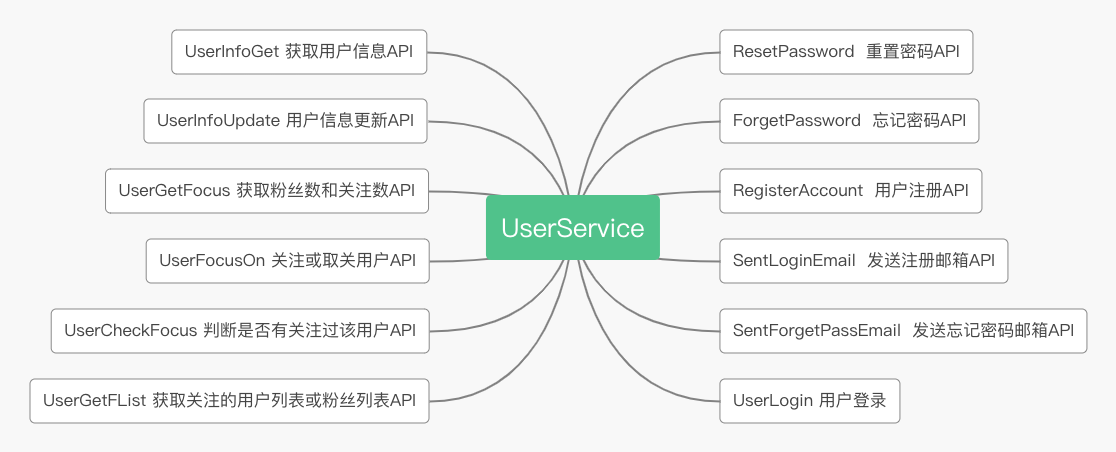


图4.2 用户模块服务接口示意图

* ResetPassword

重置密码的API，用户可以修改自己密码，但是需要提供老密码和新密码，然后新老密码会进行MD5哈希计算，然后根据账号和老密码查询数据库，最后更新用户的密码。

* ForgetPassword

忘记密码并修改密码的API，用户需要提供邮箱账号，后台会下发邮件，用户回填验证码，然后用户填写新密码，最后后台会校验验证码，是否与后台保存的一致，是否过期，然后后台会根据账号查询用户信息，并更新用户的密码。

* RegisterAccount

注册账号的API，用户需要填写邮箱账号和密码，后台会下发邮件，然后用户回填验证码，提交注册，后台会校验验证码，并对密码哈希，然后插入数据库。

* SentLoginEmail

发送注册验证码的API，会下发邮箱，邮件包含注册的验证码，然后验证码会保存在Redis中，并设置过期时间，用于后续在RegisterAccount中校验。

* UserLogin

用户登录的API，根据账号和密码查询数据库，然后生成Token，放入HTTP中的Cookie中。

* UserInfoGet

用户信息获取API，根据用户ID查询数据库用户的信息，比如用户名，头像，发表文章数，点赞数，粉丝数等。

* UserInfoUpdate

用户信息更改API，用户可以更改用户名，头像等信息，需要用户ID，然后修改数据库中用户数据。

* UserGetFocus

获取粉丝数和关注数的API，后台根据用户ID，查询Redis中的缓存数据，如果缓存过期便会拉取数据库信息，并更新到Redis中。

* UserFocusOn

用户关注或取消关注的API，根据用户的ID和关注/取消关注的用户ID，修改数据库中用户关注的状态。

* UserCheckFocus

检查用户是否有关注这个用户的API，根据用户的ID和这个用户的ID，查询数据库中用户关注的状态。

* UserGetFListReq

获取关注的用户列表或者粉丝用户列表，根据用户ID查询数据库关注哪些用户或哪些用户关注他。

### 4.2.2 用户信息安全与鉴权实现

用户模块中最重要的就是如何保证用户信息安全，如何有效校验用户身份，后面就从用户注册、登录和鉴权讲解具体设计和实现。

1. 用户注册

用户注册首先需要填写邮箱账号和密码，后台会对填写的邮箱账号下发一个邮件，用户还需要填写邮件里验证码，后台只对验证码缓存一分钟，过期无效，可以有效避免用户恶意注册账号，当用户填写好所有信息提交注册，后台会验证验证码是否正确，是否有过期，如果都正确，那么后台会往数据库里插入一条用户信息，会保存用户的账号和经过哈希的密码，后台不会保存真实密码，会对密码进行哈希计算，这样当数据库泄露，也无法获知用户的真实密码。

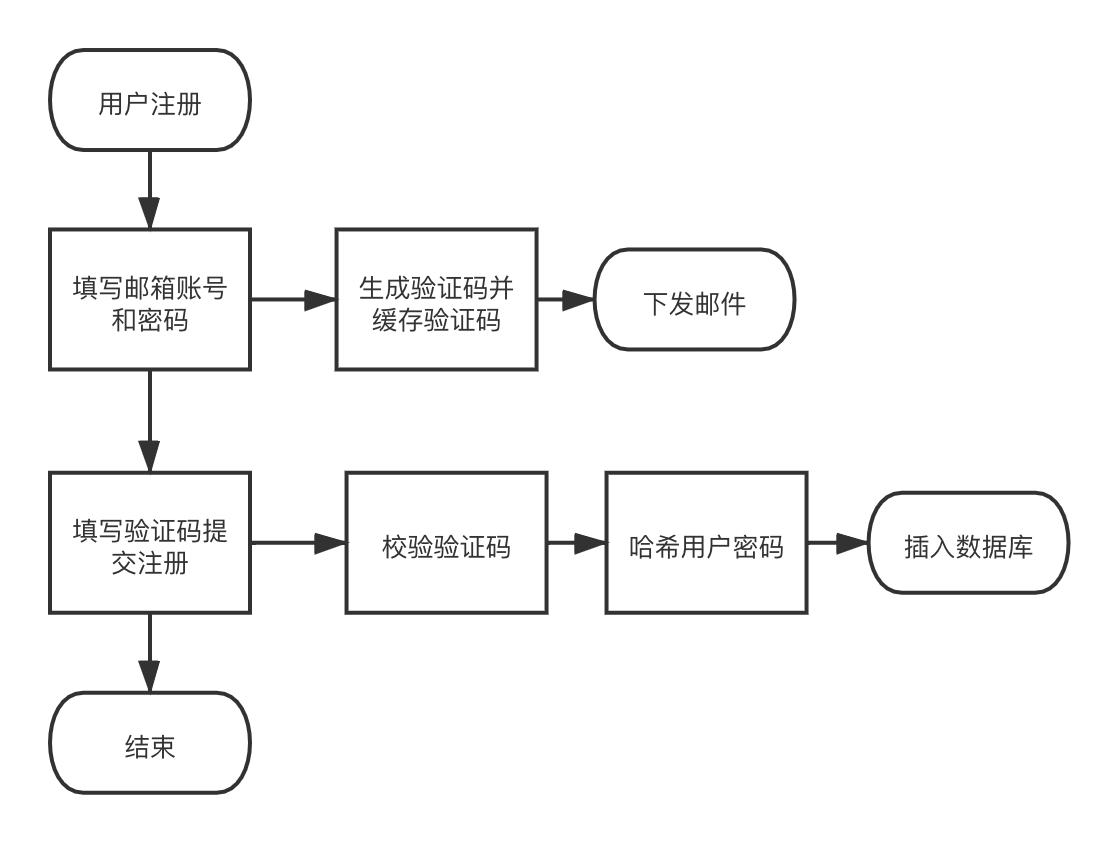


图4.3 用户注册流程图

1. 用户登录

用户登录首先需要填写账号和密码，后台会校验用户的账号和密码，首先会对密码进行哈希，然后对数据库进行查询，如果查询到有该用户，会给用户生成一个token，存放到cookie[12]里面，用于后续鉴权，而且token可以携带一些用户信息，比如级别，用户id等，token主要使用JWT实现，JWT是一个数据结构，[13]它由Header、Payload、Signature三部分组成，Header主要描述JWT的元数据，比如签名的算法，token的类型；Payload用于存放实际需要传递的数据，比如后台存放了用户的id，级别等，Signature顾名思义是签名，是对前两个部分的签名，用于完整性校验，防止数据篡改。

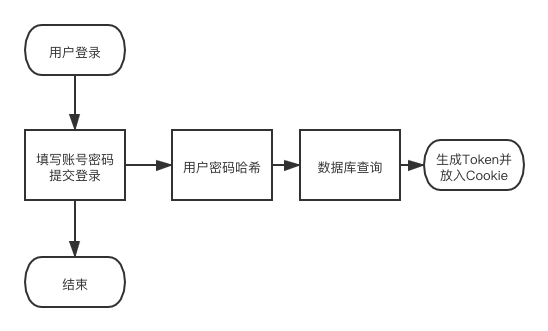


图4.4 用户登录流程图

1. 鉴权

用户访问受限资源的时候，会检查用户身份，此时会从HTTP的Header中获取用户令牌Token，然后会对Token进行解析校验，判断是否合法，是否过期，然后就会校验用户级别，因为Token里会携带用户的级别信息，然后根据级别来访问不同的资源。

```

// 从HTTP Header里获取Token

tokenstring := r.Header.Get("Authorization")

...

// 校验Token

ses, ok := ValidTokenWithClaims(tokenstring)

...

// 校验用户级别

if ses.CheckAll() && (ses.Level)&level == ses.Level {

b.Logger.Info(fmt.Sprintf("userid=%d access to resource", ses.Userid))

b.Ses = ses

return next(w, r, b)

}

```

以上代码是用户鉴权的部分片段。

## 4.3 私信模块

### 4.3.1 API设计与实现

私信模块是提供用户可以与其他用户聊天的功能，用户可以点对点与用户实时聊天，即使用户不在线，消息内容会保存在数据库中，待用户上线既可获取最新的消息，并且用户可以回看历史消息，并且会话窗口会一直保存。

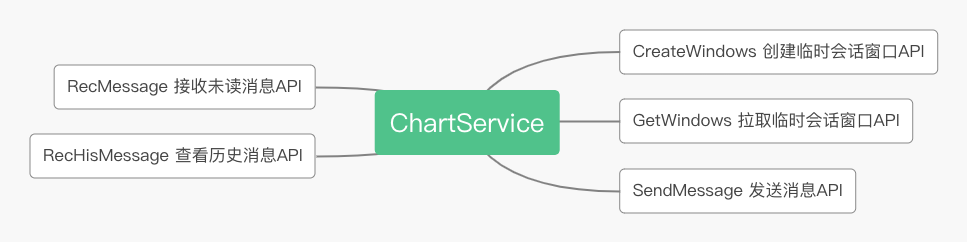


图4.5 私信模块服务接口示意图

* CreateWindows

创建临时会话窗口API，当用户A跟用户B聊天的时候，需要先创建一个窗口ID，用户A和用户B都会保存这个窗口ID，用于后续保存的消息都绑定这个窗口ID，这样可以方便通过窗口ID拉取未读消息，历史消息等。

* GetWindows

拉取临时会话窗口API，根据用户ID查询数据库用户创建的窗口列表，列表内容有最后聊天消息，最后发送时间，未读数等。

* SendMessage

发送消息API，用户之间发送的消息最后都会落地存储保存在了数据库中，用户可以通过查看历史消息，这里会有一个状态标记消息是否未读，这样在拉取未读消息的时候可以通过这个字段进行拉取。

* RecvMessage

接收未读消息API，根据用户ID和窗口ID拉取数据库未读的消息，可以支持分页拉取。

* RecHisMessage

获取历史消息API，根据用户ID和窗口ID拉取数据库中的消息，可以支持分页拉取。

### 4.3.2 点对点聊天实现

用户之间消息发送，需要后端能够有主动推送消息的能力，但是现有HTTP协议无法支持，因为HTTP协议是无状态的，模型是请求与响应，后端无法感知用户的连接是哪个，所以后端使用了WebSocket[14]协议，WebSocket基于TCP协议的应用层协议，可以主动双向数据传输，这样当用户打开一个会话窗口的时候，前端就会与后端建议WebSocket连接，后端会保存WebSocket连接并与用户信息进行绑定，然后当用户发送聊天的时候，后台会查询接收方对的WebSocket连接是否存在，如果存在，则通过连接推送到前端，前端进行展示消息，如果不存在，会给这则消息标记状态并保存到数据库中，待用户上线的时候可以拉取最新的消息。[15]

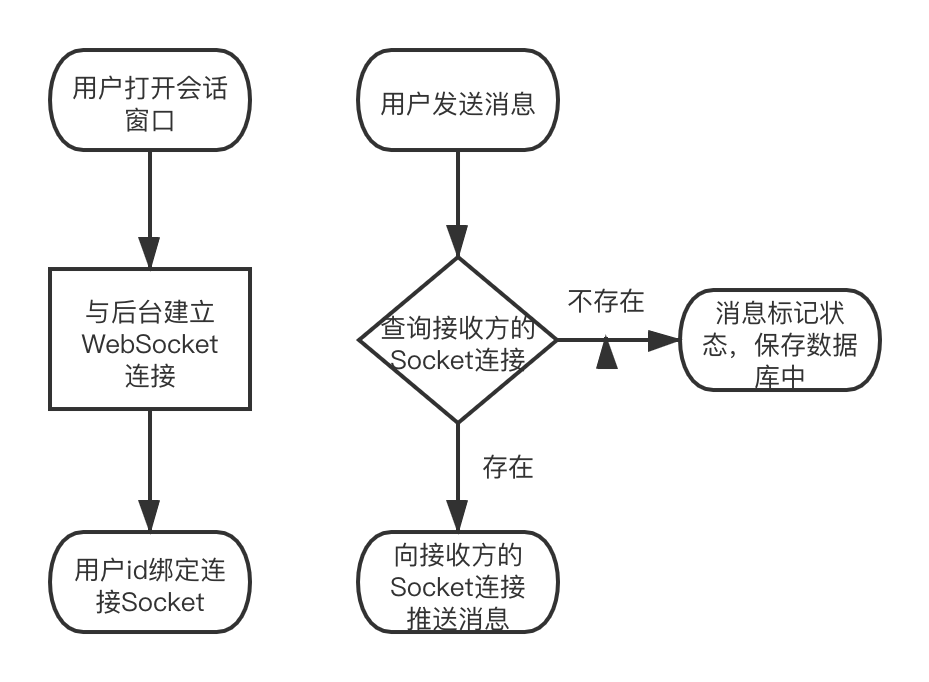


图4.6 消息发送的流程图

## 4.4 点赞模块

4.4.1 API设计与实现

用户可以给文章进行点赞，可以给评论进行点赞，后台会记录这些点赞的数量，其他用户可以看到自己文章被点赞的数量，评论被点赞的数量，在个人资料中也可以看到自己被点赞了多少次，这里后台把这些操作归到点赞的模块里面，划分为一个服务。



图4.7 点赞模块服务接口示意图

* GetPostClicks

获取文章点赞的数量API，根据文章ID和用户ID先拉取缓存中的数据，如果缓存数据过期，那么从数据库中拉取，并写回缓存中，最后将文章点赞数返回。

* OnClick

对文章、评论点赞的API，这个API后台做了通用的数据结构，用Type来标示区分是文章还是评论事件，然后根据TypeID来标示文章ID或者评论ID，然后对缓存数据进行修改，并且将点赞事件推送到缓存的队列中去，等定时任务触发，将点赞事件的数据更新到数据库中。

4.4.2 避免高频点赞导致数据库不可用实现

点赞是非常频繁的操作，如果每次点赞都去修改数据库的数据，那么对数据库压力是非常的大，因为后台会跟数据库建立很多连接，每次点赞就建立一个连接，那么当点赞非常频繁的时候，那么会导致数据库连接过多而不可用，因为数据库有连接数限制，超出这个限制，后续的请求建立连接都会失败，这个会影响整个业务，因为其他业务也需要对数据库进行操作，但是此时数据库连接数过多，导致失败，所以为了避免高频点赞，后台使用Redis作为缓存，点赞的操作均在Redis中完成，这里使用了Redis的哈希表，可以建立多级映射，可以保存很多信息，这里只需要保存用户点赞操作的时间，用户点赞的状态（用户点赞之后，再次点赞就是取消点赞）以及点赞的数量（文章或某个评论的点赞数量）。

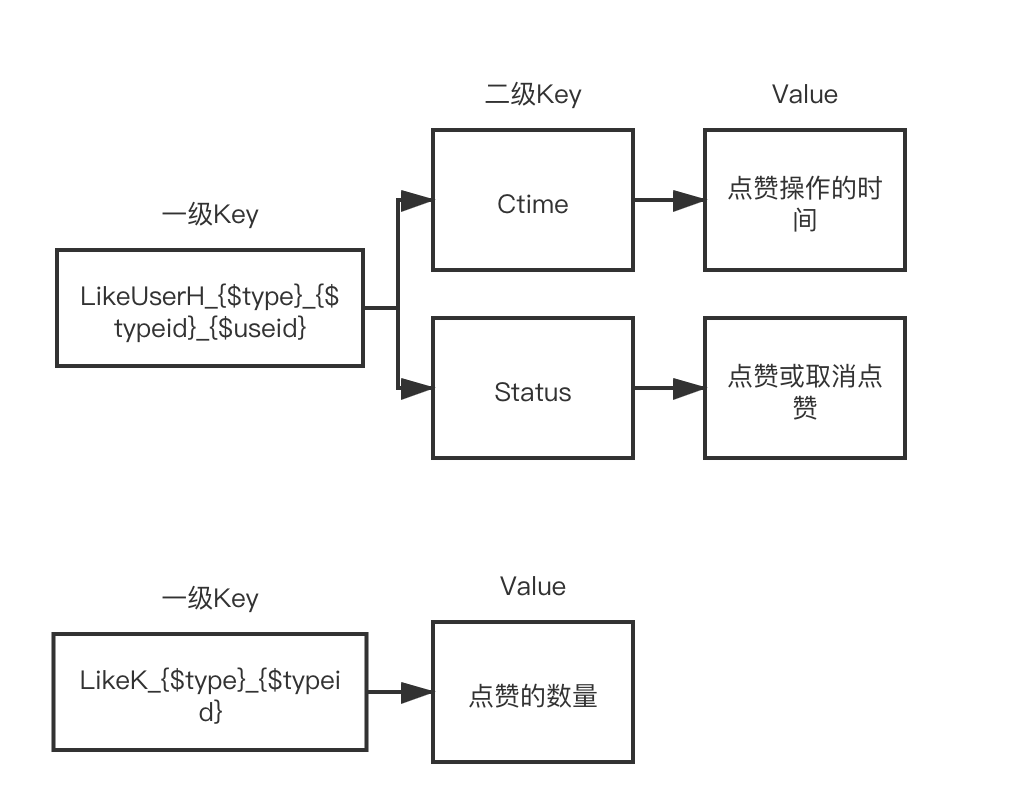


图4.8 Redis中的数据结构示意图

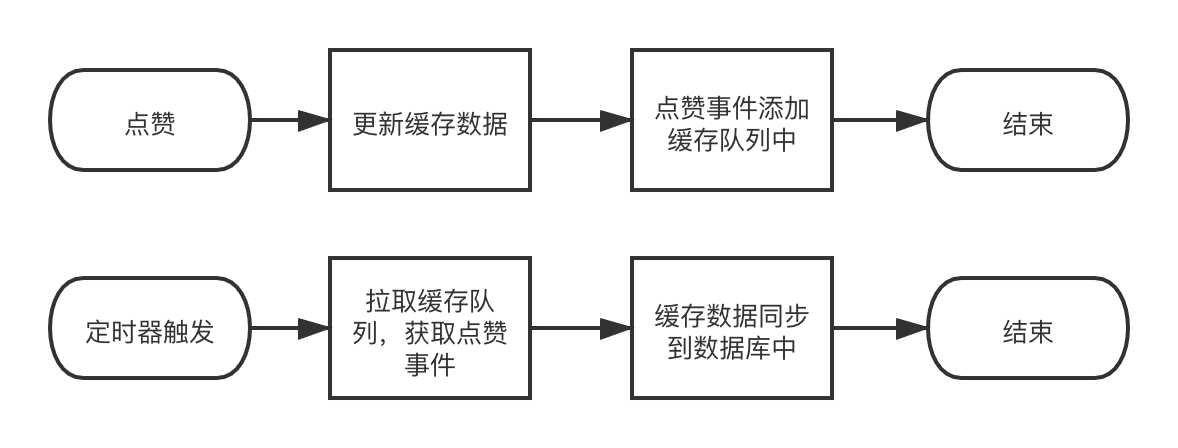


图4.9 后台点赞处理流程图

## 4.5 文章模块

### 4.5.1 API设计与实现

文章模块是与前端主要交互模块之一，它涉及用户发表的文章，在主页拉取文章列表并显示摘要，用户管理自己的文章等，这些功能都统一归纳到文章模块内，文章模块即是一个服务，对外提供文章相关操作的接口。

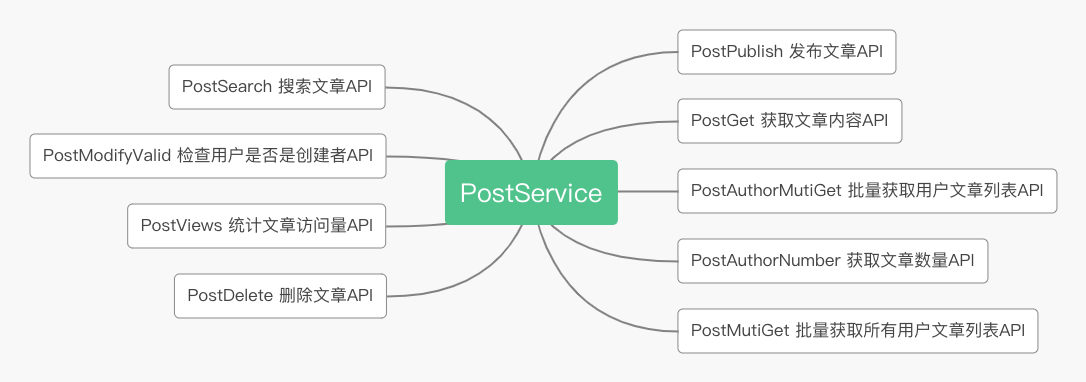


图4.10 文章模块服务接口示意图

* PostPublish

发布文章API，用户发表文章需要调此接口进行发布，将用户发表文章的信息更新到数据库中

* PostGet

获取文章的API，用户浏览文章的时候需要调用此接口，该接口户会根据文章的ID返回文章的内容。

* PostAuthorMutiGet

获取用户发表文章的列表的API，此接口用于专门拉取某个用户的文章列表，该接口根据用户ID查询数据库，将文章列表信息拉取出来。

* PostAuthorNumber

获取用户文章数量的API，此接口会返回指定用户发表文章的数量，这个根据用户ID查询数据库，统计已发表文章的数量

* PostMutiGet

批量获取所有用户文章列表的API，此接口用于主页拉取发表文章，会返回已发表的文章并默认按事件排序。

* PostSearch

搜索文章的API，该接口提供搜索文章标题的能力，通过数据库中的标题进行简单匹配查询。

* PostModifyValid

判断用户是否为文章拥有者API，根据用户ID和文章ID来查询数据库，如果能查到相应数据，则用户为文章的拥护者。

* PostViews

统计文章访问量的API，每当打开一个文章就会请求后台，文章访问量加1，但是后台没有简单做成这样，会对请求筛重，如果是同一个用户多次打开过这个文章，这最终这个文章访问量只会加1。

* PostDelete

删除文章的API，用户可以选择删除文章，后台会对文章做一个已删除的标记，并不会真正删除文章。

### 4.5.2 统计“真实”访问量

后台统计访问量，并没有简单做成一个请求过来，访问量就加1，会对请求进行筛重，因为这样精确统计到底有多少人访问了该文章，而不是打开了多少次这个文章，数据统计的维度不一样，但是要统计有多少人访问了文章不是易事，需要有效的筛重，这样就很大集合，将访问过的用户添加进去，这样用户再次访问，可以查询集合是否有该用户，如果没有该用户，则访问量可以加1，可是集合会越来越大，效率也会越来越低，非常消耗内存，所以后台使用了Redis的HyperLogLog数据结构，Redis的HyperLogLog是用来作基数统计的算法，它的优点是在输入元素的数量或者体积非常非常大时候，计算基数所需的空间总是固定的、并且很小的，非常适合用作统计访问量这种集合组件，所以当用户打开一个文章的时候，会先查询HyperLogLog中是否有该用户存在，如果没有则插入进去，并且访问量加1，当然如果用户没有登录，是游客身份，则会根据IP地址作为游客身份的标示。

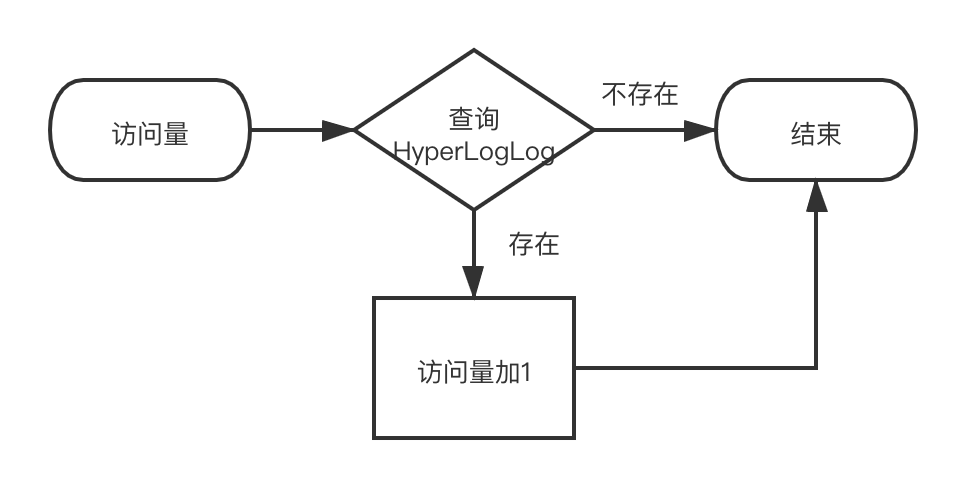


图4.11 访问量请求处理流程图

## 4.6 定时器模块

定时器模块的职责要把文章点赞数据，评论点赞数据，文章访问量数据都同步到数据库中，因为这些数据在业务中都使用缓存保存了最新数据，所以后台需要单独编写个定时器模块将这些数据同步到数据库，保证数据的一致性，这里是用了Redis的队列实现，将需要通过定时器模块更新的事件，那么其他业务则需要往Redis的队列添加事件，而定时器模块会定时拉取队列的事件，然后获取缓存中最新的数据同步到数据库中，这里还使用了Redis支持的Lua脚本语言，可以快速批量的拉取事件并处理数据。

```

redis.replicate\_commands()

local typeid=redis.call("spop", string.format(KEYS[1],ARGV[1]),ARGV[2]) -- 获取事件类型

local array={}

for key,value in ipairs(typeid)

Do

-- 获取事件类型的相关数据

 local clicks=redis.call("get",string.format(KEYS[2],ARGV[1],value))

 array[key]={value,clicks}

end

return array

```

以上代码是项目中使用Lua脚本语言的片段，可以看到在脚本语言中可以批量处理命令，而不需要发送多个请求处理，极大减少了网络传输，提高了处理效率，但是由于Redis处理Lua脚本命令的时候是原子操作，而且Redis是单线程，所以不能让Redis处理Lua脚本命令时间过长，要不然会影响其他业务使用Redis，后台对每次处理事件的次数做了限制，可以很好避免Redis处理某一命令事件过长，而堵塞其他命令的问题。

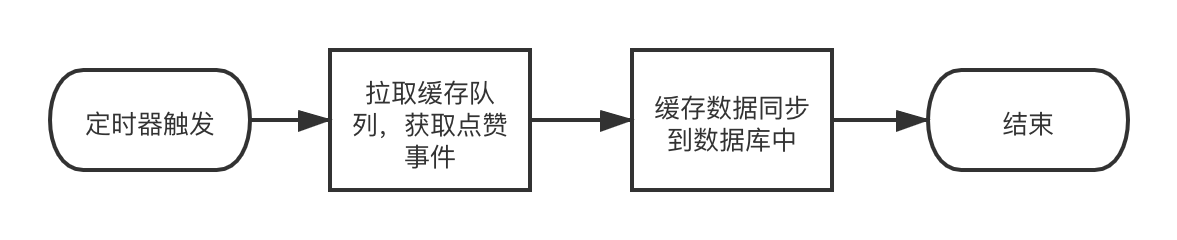


图4.12 定时器触发处理流程

## 4.7 本章小结

本章重点讲解后端主要模块的设计与实现，日志模块不是一个服务，但是它是每个模块服务的基础，定时器模块是单独起的进程不能算是服务，它职责主要帮助保证数据的一致性，其他模块皆为服务，会单独部署，这些服务都是某块功能的集合，极大降低了耦合性而且非常易于扩展。

# 数据库设计

## 5.1 缓存表设计

1. 普通Key-Value键值

表5.1 Key-Value

|  |  |
| --- | --- |
| key | 备注 |
| login\_{$account} | {$account}是用户账号，保存用户注册的验证码，并设置了过期时间为30分钟。 |
| forget\_{$account} | {$account}是用户账号，保存用户忘记密码的验证码，并设置了过期时间为30分钟。 |
| KPV\_{$postid} | {$postid}是文章ID，保存文章的访问量，并设置了过期时间为4天。 |
| LikeK\_{$type}\_{$typeid} | {$type}是点赞的类型，可以是文章，评论等类型，{$typeid}是通用ID，可以是文章ID，评论ID等类型，存储点赞的数量，并设置了过期时间为3天。 |

1. 多级Key-Value键值

表5.2 多级Key-Value

|  |  |  |  |
| --- | --- | --- | --- |
| 一级key | 备注 | 二级key | 备注 |
| huser\_{$userid} | {$userid}是用户id，存储用户相关信息，并设置了过期时间为3天。 | info | 存储了用户的基本信息：头像，昵称，座右铭等，这些数据被序列化成JSON数据保存。 |
| pubpostn | 存储了用户发表的文章数量。 |
| views | 存储了用户文章被访问的次数。 |
| clicks | 存储了用户被点赞的次数。 |
| fans | 存储了用户被多少人关注的数量。 |
| followers | 存储了用户关注多少人的数量。 |
| hpost\_{$articleid}\_{$userid} | {$articleid}是文章ID，{$userid}是用户ID，存储文章相关信息，并设置了过期时间为3天。 | title | 存储了文章的标题。 |
| content | 存储文章的内容。 |
| time | 存储文章发表的时间。 |
| private | 存储文章状态是否为私有。 |
| LikeUserH\_{$type}\_{$typeid}\_{$userid} | {$type}为通用type类型，{$typeid}为通用typeID，{$userid}为用户ID，存储用户点赞的信息，并设置过期时间为5天。 | Ctime | 存储了点赞的操作时间。 |
| Status | 存储点赞状态，true为点赞，false为取消点赞。 |

1. Set集合

表5.3 Set

|  |  |
| --- | --- |
| key | 备注 |
| LikeS\_{$type} | {$type}为通用类型，存放typeid。 |
| LikeUserS\_{$type}\_{$typeid} | {$type}为通用类型，{$typeid}为通用ID，存放用户ID。 |
| SSyncPV | 存放被访问的文章ID。 |
| SSyncPV\_U | 存放被访问文章的创建者ID。 |

1. HyperLogLog

表5.4 HyperLogLog

|  |  |
| --- | --- |
| Key | 备注 |
| HPV\_{$postid} | {$postid}为文章ID，存放用户ID或用户IP，快速筛重，设置过期时间为4天。 |

## 5.2 数据库表设计

1. 用户管理表，此表存储用户管理信息。

表5.5 用户管理表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| adminid | int(11)，主键，自增 | 用户管理ID。 |
| password | varchar(128) | 用户的密码。 |
| level | int(11) | 用户的级别。 |
| registertime | datatime | 用户的注册时间。 |
| email | varchar(128)，唯一 | 用户的账号。 |
| ban | tinyint(1) | 用户的账号的状态，true为被封，false为正常。 |

1. 用户信息表，此表存储用户基本信息。

表5.4 用户信息表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| userid | int(11)，主键，自增 | 用户ID。 |
| adminid | int(11)，唯一 | 用户管理ID。 |
| username | varchar(50) | 用户名。 |
| image | varchar(300) | 用户头像。 |
| motto | varchar(300) | 用户个性签名。 |

1. 用户关注表，此表存储用户关注信息。

表5.5 用户关注表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| attentionid | int(11)，主键，自增 | 关注ID。 |
| userid | int(11)，非唯一 | 用户ID。 |
| parentid | int(11)，非唯一 | 被关注用户ID。 |
| ctime | datatime | 关注时间。 |
| focus | tinyint(1) | 关注状态，true为关注，false为取关。 |

1. 文章表，此表存储文章信息。

表5.6 文章表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| articleid | int(11)，主键，自增 | 文章ID。 |
| title | varchar(300) | 文章标题。 |
| content | text | 文章内容。 |
| summary | text | 文章摘要。 |
| userid | int(11)，非唯一 | 创建者的用户ID。 |
| folderid | int(11)，非唯一 | 文章的文件夹ID。 |
| views | int(11) | 文章访问量。 |
| clicks | int(11) | 文章点赞量。 |
| private | tinyint(1) | 文章私有状态。 |
| publishtime | datatime | 文章发布时间。 |
| ban | tinyint(1) | 文章状态，true为文章已删除，false为正常。 |

1. 点赞表，此表存储用户点赞信息。

表5.7 点赞表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| clickid | int(11)，主键，自增 | 点赞ID |
| typeid | int(11)，非唯一 | 类型ID |
| type | int(11) | 类型 |
| userid | int(11)，非唯一 | 点赞的用户ID |
| clicktime | datatime | 点赞时间 |
| status | tinyint(1) | 点赞状态，true为点赞，false为取消点赞 |

1. 评论表，吃表存储用户评论信息。

表5.8 评论表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| commentid | int(11)，主键，自增 | 评论ID |
| articleid | int(11)，非唯一 | 文章ID |
| content | text | 评论内容 |
| clicks | int(11) | 评论点赞数量 |
| commenttime | timestamp | 评论发表时间 |
| userid | int(11)，非唯一 | 评论的用户ID |
| username | varchar(50) | 评论的用户昵称 |
| image | varchar(50) | 评论的用户头像 |
| to\_commentid | int(11) | 回复评论的评论id |
| to\_userid | int(11) | 回复评论的用户id |
| to\_username | int(11) | 回复评论的用户昵称 |
| ban | varchar(50) | 评论状态，true为删除，false 为正常 |

1. 文件夹表，此表存储文件夹信息。

表5.9 文件夹表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| folderid | int(11)，主键，自增 | 文件夹ID |
| userid | int(11)，非唯一 | 文件夹创建者的用户ID |
| fname | varchar(100) | 文件夹名 |
| folders | int(11) | 文件夹里的文件数量 |
| ban | tinyint(1) | 文件夹状态，true为删除，false为正常 |

1. 房间表，此表存储聊天房间信息。

表5.10 房间表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| rid | int(11)，主键，自增 | 房间ID |
| ctime | datetime | 房间创建时间 |
| ban | tinyint(1) | 房间状态 |

1. 窗口表，此表存储聊天窗口信息。

表5.11 窗口表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| wid | int(11) | 窗口ID |
| rid | int(11) | 房间ID |
| userid | int(11) | 用户ID |
| lastcontent | text | 最后一次聊天信息 |
| time | datetime | 最后一次聊天信息的时间 |
| to\_id | int(11) | 接收消息的用户ID |
| wintype | tinyint(1) | 窗口类型 |
| winimage | varchar(300) | 窗口头像 |
| winname | varchar(50) | 窗口昵称 |

1. 消息表，此表存储用户聊天信息。

表5.12 消息表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 备注 |
| mid | int(11) | 消息ID |
| rid | int(11) | 房间ID |
| content | text | 消息内容 |
| status | tinyint(1) | 消息状态 |
| sendtime | datetime | 消息发送时间 |
| mtype | int(11) | 消息类型 |
| userid | int(11) | 发送者ID |
| to\_userod | int(11) | 接收者ID |

# 6.结论

本文讲述了一个完整的博客系统是如何设计开发的，从总体架构设计中，博客系统采用了前后端分离架构的思想，实现了前端和后端分离，使用HTTP协议前后端通信，前后端各司其职，提高了开发效率，降低代码耦合，这已经是互联网项目开发的业界标准使用方式。然后从后端架构设计中，博客系统后端采用了微服务架构思想[16]，后端把模块划分为微服务，每个微服务是专注于单一责任与功能的小型功能区块，使用了Etcd作为后端服务发现中心，每个微服务都注册到服务发现中心，微服务之间就可以使用RPC通信，由许许多多的微服务组合成复杂的大型应用程序，而且各功能区块使用与语言无关的API集互相通信，微服务相比单体应用更具备易扩展、逻辑清晰、灵活组合、简化部署等多种特性，单体架构是将应用程序的所有功能打包成一个独立的单元，导致代码复杂度高，难以理解，应用程序伸缩性差，可靠性差，可能由于一个BUG就会引起整个应用的奔溃，所以微服务架构已逐渐被许多互联网公司所实践，每个团队可以独立负责某些服务，可以获得更高的生产力，后端也因此采用了微服务架构的思想，实现了分布式架构，掌握了其中的技术要点，明白微服务架构的优劣性，紧跟时代技术的发展，多实践，多了解，多学习。

# 参考文献

1. 曹春晖. Go语言高级编程[M]. 人民邮电出版社，2019-07-09.
2. 理查德·琼斯. 垃圾回收算法手册:自动内存管理的艺术[M]. 机械工业出版社，2016-03.
3. 刘书健. 基于协程的高并发的分析与研究[D].昆明理工大学,2016.
4. 孔海峰. 分布式RPC框架的设计与实现[D].华中科技大学,2018.
5. 易品. 基于DNS的服务发现技术的研究与设计[D].北京邮电大学,2016.
6. 黄健宏. Redis设计与实现[M]. 机械工业出版社，2014-06-01.
7. Carlos Taborda. Angular权威教程[M]. 人民邮电出版社，2017-04.
8. 苏翔宇,朱爱群.CentOS7下基于Nginx的反向代理及负载均衡研究与实现[J].现代计算机(专业版),2018(10):61-64.
9. 杜星. 轻量级Web服务器Nginx的理论与技术研究[D].南京邮电大学,2016.
10. David Gourley，Brian Totty.HTTP权威指南[M]. 人民邮电出版社，2012-09-01.
11. 郑兆雄.Go Web编程[M]. 人民邮电出版社，2017-12-01.
12. 吕俊宏,周江峰.深入解析Cookie技术[J].数字通信世界,2015(06):62-65.
13. 周虎. 一种基于JWT认证token刷新机制研究[J]. 软件工程,2019,22(12):18-20.
14. Andrew Lombardi.Lightweight Client-Server Communications[M]. O'Reilly Media，2015-9-26.
15. 惠苗,赖道健.基于WebSocket协议的即时通讯系统的开发[J].榆林学院学报,2019,29(06):76-79.
16. Nadareishvili.Microservice Architecture: Aligning Principles, Practices, and Culture[M]. O'Reilly Media，2016-8-12.

# 致谢

转眼间，大学四年就这样马上要结束了，会想起自己奋斗四年的过往，在自己大一的时候我毅然决然的选择了计算机专业，我不后悔，我找到了自己兴趣爱好所在，我顺利了加入到了学校的ACM社团，可能自己找不到学习练习ACM的方向，没过多久我就退出了ACM，开始自己自学之路，但是这个过程非常的艰辛，容易找不到方向，不知道做什么学什么，直到当我大二的时候，我参加了学院的创新班社团，这是我大学的转折点，我非常感谢创新班，它引领着我踏入软件开发的世界，我重新提起学习的兴趣，不在盲目没有方向，类似无头苍蝇一般，我在创新班中学到了很多软件开发的经验，还教会了我学习的方法，还了解了企业软件开发的流程，这些都是我后来应聘企业的敲门砖，所以非常感谢金政哲老师带领着创新班同学不断学习探索，也感谢创新班的同学一起陪伴着度过大学四年充实的生活，感谢他们互帮互助，让我明白团队的重要性，还要感谢计算机学院的老师，感谢他们耐心指导，给我们打下坚实的基础，让我们才能在互联网开发中寻求一席之地，最后感谢计算机学院，希望计算机学院越来越好。