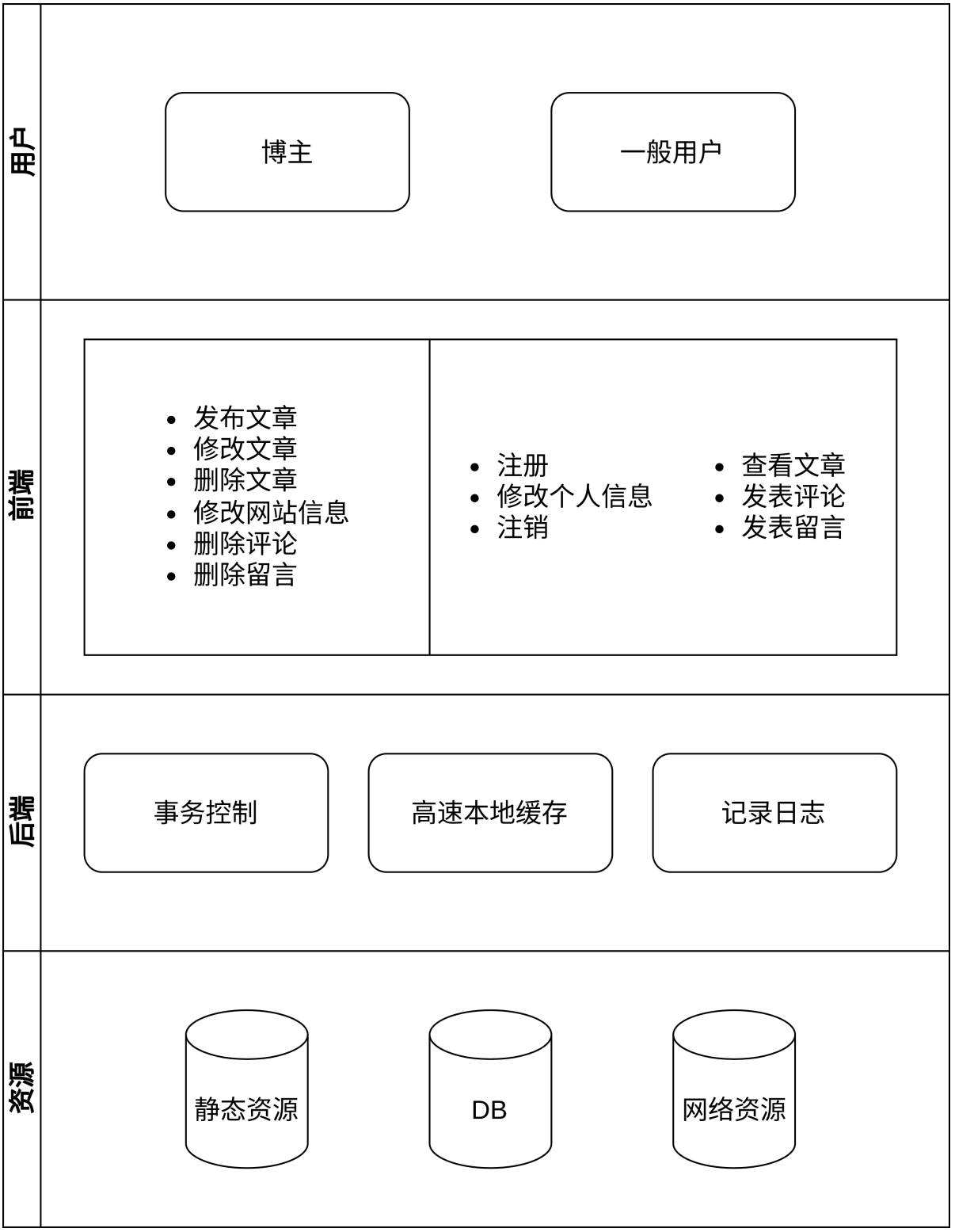
三、系统架构设计 [徐建奇]

整体架构



在设计的时候主要考虑两方面的内容，一方面是用户相关的操作，一方面是文章相关的操作，剩下部分集中作为扩展内容（包括网站的信息相关和留言板相关），后续也能够随着新内容部的引入而从中分离出来。

鉴于在一些静态博客的框架上（如 Hexo）不具备这种实时的互动，用户模块的设计初衷在于让用户间能够进行一些互动，分享心得体会。最初设计的时候用户仅有博主通过后端脚本进行发布文章，其他用户仅有评论和留言的权限，后续为了让该系统更加亲民，任何专业的用户都能体验完整的功能，让发布文章能够通过前端进行，于是在这部分的设计上让所有用户都具有了发布文章的权利，该个人博客也会逐

渐演化为了论坛的形式。当然为了与最初的方案统一，最终还是以个人博客的形式呈现，限制了仅有特定的用户能够发布文章，但为后续向论坛的形式进行改进保留了空间。

文章的检索是一个较为重要的内容，原本计划上设计了标签(tag)以及分类(category)两个属性来作为不同的检索依据，后来认为在个人博客的结构中，这两个属性的重叠度较高，基于设计简单的原则，保留了标签。同样的，在后续的迭代中，可能仍然会重新将分类这个检索依据重新加上。

在资源存储方面，考虑需要对图片等二进制文件进行存储，在建立数据库的基础上建立了图床以便让用户更加方便地构建起头像、相册等功能。

前端

!todo

后端

选择 Rust Axum 作为后端 Web 框架的原因有很多，这些原因包括了主观因素和客观事实，并充分利用了 Rust 语言的优势。下面将对这些原因进行简单阐释：

1. **内存管理严格**：Rust 对内存管理非常严格，通过其独特的所有权系统和借用检查器，避免了内存泄漏问题，提供了更好的内存安全保障。
2. **运行速度快**：Rust 通过将代码编译成机器码来运行，避免了 Java 虚拟机运行和 Python 解释运行带来的开销，同时在无垃圾回收机制（GC）的内存管理体系下，Rust 的执行效率理论上更高。
3. **高并发支持**：Rust 非常适合高并发编程。它具备数据无竞争和安全高效的线程通信等优势。此外，Rust 还拥有像 Tokio 这样的优秀异步编程库，进一步增强了其在并发编程中的表现。

数据库上选择了 MariaDB。主要考虑了其易用以及兼容性，是 MySql 的一个分支，其次由于在 debian 12 系统中默认安装了 MariaDB，而 MySql 不在 apt 源中，因此使用了 MariaDB 作为数据库。

四、API设计 [徐建奇]

该部分在设计的时候采取了类似面向对象的方式，路由的第一级表示要操作的`对象`，第二级表示对该对象采取的`动作`。

参照`三、系统架构设计`中描述的，主要分为两类对象，用户和文章，其他零碎的部分则是仅有一级，当然随着后续功能的加入也会分离出对应的对象。

因此在设计此 API 时，遵循了以下原则：

1. **模块化设计**：将功能划分为独立的模块，以便于管理和扩展。
2. **简洁易用**：API 的设计尽量简洁，参数和返回值清晰明了，便于开发者使用。
3. **安全性**：确保敏感操作（如用户信息更新、文章管理等）需要身份验证，保护用户数据。
4. **高性能**：通过优化数据结构和查询，确保 API 的高性能响应，满足高并发请求的需求。
5. **扩展性**：设计考虑到未来的扩展需求，确保可以方便地增加新功能和模块。

用户模块

包括常规的注册、注销、登陆、修改信息、查询信息操作以及发布文章、评论、留言这些特定的操作。

文章模块

文章模块是系统的核心部分，涵盖了创建、删除、更新、获取文章内容、获取文章信息列表、获取文章的评论列表功能。

扩展

扩展部分包括网站的基本信息管理（查看信息和修改信息）、留言板功能（查看留言板列表）、上传文件和静态文件访问。

> **网站信息**:
> 此功能用于获取和更新网站的基本信息，包括标题、副标题、描述、作者等内容，以及一些社交媒体的链接。

API 表单

路由	说明
`/post/create`	创建一篇文章
`/post/delete`	删除一篇文章
`/post/update`	更新一篇文章
`/post/content`	得到文章内容
`/post/infolist`	得到文章列表
`/post/remarks`	得到文章对应的评论内容
`/user/register`	注册一名用户
`/user/delete`	注销一名用户
`/user/update`	更新用户信息
`/user/login`	用户登陆
`/user/info`	得到用户信息
`/user/remark`	用户发表评论
`/user/chat`	用户留言
`/chatlist`	留言板消息列表
`/upload`	上传文件
`/resouce/{file/path}`	静态文件访问
`/basicinfo`	获取网站信息
`/updateinfo`	更新网站信息

API 使用及测试

Base URL

http://124.223.209.159:8300/api

Routes

`/post`

Create Post

- **Description**: 创建一篇文章
- **URL**: `/post/create`
- **Method**: POST
- **Request Body**:

```
{
   "title": "test",
   "author": 100000,
   "brief": "广告招租位",
   "content": "test",
   "tags": ["广告", "测试"]
}
```

- **Response**:
  - **Status**: 200 OK

Delete Post

- **Description**: 删除一篇文章
- **URL**: `/post/delete`
- **Method**: POST
- **Request Body**:

```
{
  "pid": "f354c332-a2ad-42a6-bdae-99b535395ef1"
}
```

- **Response**:
  - **Status**: 200 OK / 500 Internal Server Error
  - **Error Message**:

    ```
    "Data Not Found in database"
    ```

Update Post

- **Description**: 更新一篇文章
- **URL**: `/post/update`
- **Method**: POST
- **Request Body**:

```
{
  "pid": "dfebd659-9cd2-4c9a-8a42-85fbf561bd9a",
  "title": "update",
  "brief": "广告招租位update",
  "content": "更新！！！！",
  "tags": ["广告", "测试", "更新"]
}
```

- **Response**:
  - **Status**: 200 OK

List Post Information

- **Description**: 得到文章列表
- **URL**: `/post/infolist`
- **Method**: GET
- **Response**:
  - **Status**: 200 OK
  - **Body**:

```
[
  {
    "author": 100000,
    "brief": "广告招租位",
    "date": "2024-06-06T13:08:27Z",
    "pid": "283036ea-afe6-46df-bbb4-f47d10172114",
    "tags": ["广告", "测试"],
    "title": "test"
  },
  ...
]
```

Get Post Content

- **Description**: 得到文章内容
- **URL**: `/post/content`
- **Method**: GET
- **Query Parameters**:
  - `pid`: string
- **Response**:
  - **Status**: 200 OK

- **Body**:

```
Content 内容 String
```

List Post Remarks

- **Description**: 得到文章对应的评论内容
- **URL**: `/post/remarks`
- **Method**: GET
- **Query Parameters**:
  - `pid`: string
- **Response**:
  - **Status**: 200 OK
  - **Body**:

```
[
  {
    "content": "remark",
    "created_at": "2024-06-06T13:08:27Z",
    "uid": 10000000
  },
  ...
]
```

`/user`

Register User

- **Description**: 注册一名用户
- **URL**: `/user/register`
- **Method**: POST
- **Request Body**:

```
{
  "name": "username",
  "passwd": "password"
}
```

- **Response**:
  - **Status**: 200 OK
  - **Body**:

```
uid
```

Delete User

- **Description**: 注销一名用户
- **URL**: `/user/delete`
- **Method**: POST
- **Request Body**:

```
{
  "uid": 10000002
}
```

- **Response**:
  - **Status**: 200 OK

Update User

- **Description**: 更新用户信息
- **URL**: `/user/update`
- **Method**: POST
- **Request Body**:

```
{
  "uid": 10000000,
  "passwd": "root",
  "name": "root",
  "avatar": null,
  "bio": null
}
```

- **Response**:
  - **Status**: 200 OK

User Login

- **Description**: 用户登陆
- **URL**: `/user/login`
- **Method**: POST
- **Request Body**:

```
{
  "uid": 10000000,
  "passwd": "root"
}
```

- **Response**:
  - **Status**: 200 OK / 500 Internal Server Error
  - **Body**:
    - Success:

```
{
  "code": 1,
  "userinfo": {
    "avatar": null,
    "bio": null,
    "created_at": "2024-06-06T07:47:26Z",
    "name": "root",
    "uid": 10000000
  }
}
```


- Failure:


```
{
  "code": 0,
  "userinfo": null
}
```


- Error Message:


```
"User Not Found
```


User Info

- **Description**: 得到用户信息
- **URL**: `/user/info`
- **Method**: GET
- **Query Parameters**:
  - `uid`: int
- **Response**:
  - **Status**: 200 OK
  - **Body**:


```
 {
  "uid": 10000000,
  "name": "root",
  "avatar": null,
  "bio": null,
  "created_at": "2024-06-06T07:47:26Z"
}
```


Add Remark

- **Description**: 用户发表评论
- **URL**: `/user/remark`
- **Method**: POST
- **Request Body**:

```
{
  "uid": 10000000,
  "pid": "dfebd659-9cd2-4c9a-8a42-85fbf561bd9a",
  "content": "remark"
}
```

- **Response**:
  - **Status**: 200 OK

Add Chat

- **Description**: 用户聊天
- **URL**: `/user/chat`
- **Method**: POST
- **Request Body**:

```
{
  "uid": 10000000,
  "content": "chat"
}
```

- **Response**:
  - **Status**: 200 OK

`/chatlist`

- **Description**: 聊天消息列表
- **URL**: `/chatlist`
- **Method**: GET
- **Request Body**:

```
{
  "uid": 10000000,
  "content": "chat"
}
```

- **Response**:
  - **Status**: 200 OK
  - **Body**:

```
[
  {
    "content": "chat",
    "created_at": "2024-06-06T12:55:05Z",
    "uid": 10000000
  },
  ...
]
```

Upload

- **Description**: 上传文件
- **URL**: `/upload`
- **Method**: POST
- **Request Body**:
  - Content-Type: multipart/form-data
  - Fields:
    - file: 文件内容，类型为 file，即要上传的文件。
- **Response**:
  - **Status**: 200 OK

> ps: 传入的文件内容要求小于 2MB

resouce

- **Description**: 静态文件访问
- **URL**: `/resouce/{file/path}`
- **Method**: GET
- **Response**:
  - **Status**: 200 OK

basic_info

- **Description**: 网站信息
- **URL**: `/basicinfo`
- **Method**: GET
- **Response**:
  - **Status**: 200 OK
  - **Body**:

```
{
  "sitebasicinfo": {
    "title": "",
    "subtitle": "",
    "description": "",
    "author": "",
    "favicon": "",
    "avatar": ""
  },
  "mylinks": {
    "github": "",
    "bilibili": "",
    "zhihu": "",
    "qq": "",
    "wechat": null,
    "gitee": ""
  }
```

```
}
```

basic_info_update

- **Description**: 更新网站信息
- **URL**: `/updateinfo`
- **Method**: POST
- **Request Body**:

```
{
  "sitebasicinfo": {
    "title": "",
    "subtitle": "",
    "description": "",
    "author": "",
    "favicon": "",
    "avatar": ""
  },
  "mylinks": {
    "github": "",
    "bilibili": "",
    "zhihu": "",
    "qq": "",
    "wechat": null,
    "gitee": ""
  }
}
```

- **Response**:
  - **Status**: 200 OK

五、数据库设计

本数据库设计旨在支持一个包含用户管理、文章发布与评论、用户互动等功能的全功能应用，包括七张主要的表：`users`、`user_infos`、`posts`、`remarks`、`chat_messages`、`post_tag`、`tag_set`。以下是对数据库表结构及其设计原理的详细解释。

设计原则与优势

在数据库设计过程中，遵循了以下原则和优势：

1. **规范化设计**：通过合适的表结构和关联，确保数据存储的有效性和一致性。
2. **扩展性**：使用标准的多对多关系模式和自增主键，方便未来系统功能的扩展和改进。
3. **性能优化**：合理使用索引和优化查询，提升系统在大规模数据处理和高并发访问时的性能表现。
4. **可维护性**：使用时间戳字段追踪数据修改历史，便于故障排查和数据修复。
5. **不设置外键**：提高了效率、灵活性和扩展性，相应的限制在后端代码中实现。

数据库表描述

用户相关表格

1. `users` 表格

```
CREATE TABLE `users` (
    `uid` INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    `passwd` VARCHAR(255) NOT NULL
) AUTO_INCREMENT=10000000;
```

- **设计目的**：存储用户的基本身份信息，`uid` 为用户唯一标识，自增长；`passwd` 存储用户密码，确保用户数据的安全性。

2. `user_infos` 表格

```
CREATE TABLE `user_infos` (
    `uid` INT UNSIGNED PRIMARY KEY,
    `avatar` VARCHAR(255),
    `name` VARCHAR(255),
    `bio` TEXT DEFAULT NULL,
    `created_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    `updated_at` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- **设计目的**：存储用户的详细信息，包括头像路径、用户名、个人简介等。
- **关联性**：与 `users` 表通过 `uid` 建立一对一关系，确保用户信息的完整性和一致性。
- **时间戳字段**：`created_at` 记录用户创建时间，`updated_at` 记录最后更新时间，用于数据版本控制和审计。

文章及评论相关表格

3. `posts` 表格

```
CREATE TABLE `posts` (
    `pid` uuid NOT NULL DEFAULT uuid() PRIMARY KEY,
    `title` varchar(255) NOT NULL,
    `author` INT UNSIGNED,
    `brief` text NOT NULL DEFAULT '暂无简介',
    `content` longtext DEFAULT NULL,
    `created_at` timestamp NULL DEFAULT current_timestamp(),
    `updated_at` timestamp NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp()
);
```

- **设计目的**：存储用户发布的文章信息，包括标题、作者、简介和内容。
- **关联性**：`author` 字段关联到 `users` 表的 `uid`，建立作者与文章之间的关系。
- **时间戳字段**：`created_at` 记录文章创建时间，`updated_at` 记录最后更新时间，便于文章版本管理和时间排序。

4. `remarks` 表格

```sql
CREATE TABLE `remarks` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `uid` INT UNSIGNED DEFAULT NULL,
  `pid` uuid DEFAULT NULL,
  `content` text DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT current_timestamp(),
  `updated_at` timestamp NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp()
) AUTO_INCREMENT=0;
```

- **设计目的**：存储文章的评论信息，包括评论内容、评论者 ID (`uid`) 和文章 ID (`pid`)。
- **时间戳字段**：`created_at` 记录评论创建时间，`updated_at` 记录最后更新时间，用于评论排序和版本控制。

5. `chat_messages` 表格

```sql
CREATE TABLE `chat_messages` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `uid` INT UNSIGNED DEFAULT NULL,
  `content` text DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT current_timestamp(),
  `updated_at` timestamp NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp()
) AUTO_INCREMENT=0;
```

- **设计目的**：存储留言板信息，包括留言内容、留言者 ID (`uid`)。
- **时间戳字段**：`created_at` 记录留言创建时间，`updated_at` 记录最后更新时间，用于留言排序和版本控制。

辅助表格

6. `post_tag` 表格

```sql
CREATE TABLE `post_tag` (
  `pid` uuid NOT NULL,
  `tag` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`tag`,`pid`)
);
```

- **设计目的**：实现文章与标签的多对多关系，存储每篇文章的标签信息。
- **复合主键**：使用 (`tag`, `pid`) 作为复合主键，确保同一文章不会有重复的标签，同时优化标签的检索和管理效率。

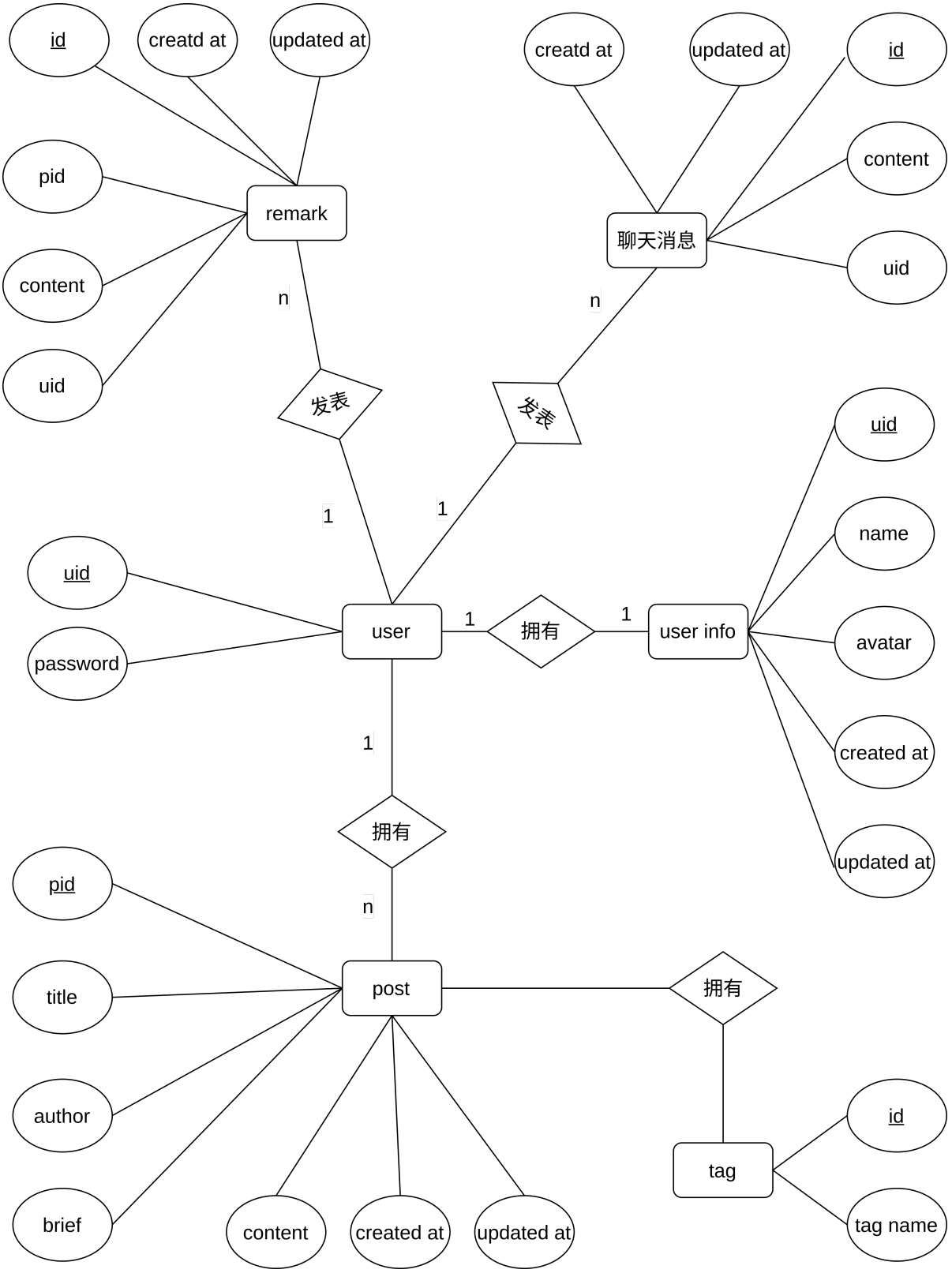
7. `tag_set` 表格

```sql
CREATE TABLE `tag_set` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `tag` varchar(255) NOT NULL
) AUTO_INCREMENT=0;
```

- **设计目的**：存储系统中所有可能的标签，保证标签的唯一性和统一管理。

- **自增主键**：使用 `id` 作为自增长的主键，确保每个标签都有唯一的标识符。

ER 图



七、后端服务器的实现

九、系统部署

十一、清单

十三、参考文献