软件系统分析与设计作业

**学术成果分享平台**

**架构设计说明书**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **项目组成员信息** | | | | |
| **小组编号** | **13** | | | |
| **小组名称** | **MSI** | | | |
| **小组联系人** | **周星达** | | | |
| **学号** | **姓名** | **联系电话** | **本次实践中主要承担的工作内容** | **本次作业占比**  **(95%-105%之间)** |
| 21373339 | 周星达 | 15269992619 | 架构设计PPT制作 | 100% |
| 20101021 | 梁坤 | 18712085680 | 非功能需求架构设计 | 100% |
| 21373330 | 吴浩宇 | 13552318175 | 部署策略 | 100% |
| 21373019 | 曾尔文 | 18979618851 | 文档分析完善 | 100% |
| 21373278 | 赵乾宇 | 18355750747 | 文档分析完善 | 100% |
| 21373326 | 秦茂凯 | 15611759462 | 文档分析完善 | 100% |
| 21373328 | 王陆昊 | 13273673856 | 文档审核与汇总 | 100% |
| 21371220 | 杨硕 | 18098715927 | 技术选型 | 100% |
| 21371134 | 田琦 | 13225685550 | 前后端架构设计 | 100% |
| 71066002 | 温舒盈 | 19658727170 | 文档分析完善 | 100% |

2023年

版本变更历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 主要编制人 | 审核人 | 版本说明 |
| 1.0 | 2023.11.05 | 王陆昊、赵乾宇、曾尔文、田琦、杨硕、温舒盈、周星达、梁坤 | 周星达 | 完成大体部分，合并完成 |
| 1.1 | 2023.10.04 | 周星达 | 周星达 | 进行内容完善 |

目 录

[1. 引言 2](#_Toc150201187)

[1.1 项目概述 2](#_Toc150201188)

[1.2 问题引入 2](#_Toc150201189)

[1.3 文档概述 2](#_Toc150201190)

[2. 总体架构 2](#_Toc150201191)

[2.1 前端架构 2](#_Toc150201192)

[2.1.1 MVVM设计模式 3](#_Toc150201193)

[2.1.2 组件化开发 3](#_Toc150201194)

[2.1.3 Vue CLI工具 3](#_Toc150201195)

[2.1.4 指令系统 3](#_Toc150201196)

[2.1.5 html、css、javascript设计 3](#_Toc150201197)

[2.2 后端架构 4](#_Toc150201198)

[3. 部署策略 5](#_Toc150201199)

[3.1 部署架构 5](#_Toc150201200)

[3.2 部署方案 6](#_Toc150201201)

[3.3 部署设备性能实例 6](#_Toc150201202)

[3.3.1 高性能服务器实例 6](#_Toc150201203)

[3.3.2 普通服务器实例 6](#_Toc150201204)

[4. 技术选型 7](#_Toc150201205)

[4.1 技术框架表格 7](#_Toc150201206)

[4.1.1 后端技术 7](#_Toc150201207)

[4.1.2 前端技术 7](#_Toc150201208)

[4.2 重要技术说明 7](#_Toc150201209)

[4.2.1 Django 7](#_Toc150201210)

[4.2.2 MySQL 8](#_Toc150201211)

[4.2.3 Elasticsearch 8](#_Toc150201212)

[4.2.4 Celery 8](#_Toc150201213)

[4.2.5 Nginx 8](#_Toc150201214)

[4.2.6 Redis 8](#_Toc150201215)

[4.2.7 Vue.js 9](#_Toc150201216)

[5. 质量属性 9](#_Toc150201217)

[5.1 设计质量 9](#_Toc150201218)

[5.1.1 概念完整性 9](#_Toc150201219)

[5.1.2 可移植性 9](#_Toc150201220)

[5.1.3 可维护性 10](#_Toc150201221)

[5.1.4 复用性 10](#_Toc150201222)

[5.2 运行质量 10](#_Toc150201223)

[5.2.1 优化性能 10](#_Toc150201224)

[5.2.2 可延展性 10](#_Toc150201225)

[5.2.3 安全性 10](#_Toc150201226)

[5.3 系统质量 10](#_Toc150201227)

[5.3.1 可支持性 10](#_Toc150201228)

[5.3.2 可测试性 10](#_Toc150201229)

[5.4 用户质量 11](#_Toc150201230)

[5.4.1 可用性 11](#_Toc150201231)

[6. 横切关注点 11](#_Toc150201232)

[6.1 安全性 11](#_Toc150201233)

[6.1.1 安全威胁 11](#_Toc150201234)

[6.1.2 安全模式 11](#_Toc150201235)

[6.2 异常处理 12](#_Toc150201236)

[6.2.1 异常检测 12](#_Toc150201237)

[6.2.2 异常恢复 12](#_Toc150201238)

[6.3 缓存 12](#_Toc150201239)

[6.4 日志 12](#_Toc150201240)

# 引言

## 项目概述

本项目定位建设一套面向广大科研从业人员的学术成果分享平台，包括资源检索，论文分享讨论，管理学术成果等功能。本项目可以为广大科研从业人员提供一个开放、共享的学术环境，促进学术成果的传播与合作，推动学术研究的发展。

## 问题引入

在当今信息时代，学术界的研究成果呈现出爆炸性增长的趋势，因此学术成果的管理涉及到大量数据的处理与存储，具有复杂的数据结构和关联关系。为了实现快速、稳定、安全的数据交互和处理，需要设计高效的后端架构。同时，为了提供用户友好的界面和交互体验，需要设计直观、灵活的前端架构。因此，本项目需要在前后端架构设计上做出合理的规划，以满足用户需求、提高系统性能、保障数据安全。

## 文档概述

本文档聚焦于学术成果分享平台的软件架构设计、部署说明和技术选型等。文档从前后端架构、部署策略、技术选型、非功能需求几个方面详细阐述了该平台的架构设计，在后续开发之前首先完成设计

# 总体架构

## 前端架构

本项目的前端架构将基于用于构建用户界面的 JavaScript 框架Vue来搭建。Vue 是一个框架，也是一个生态，其功能覆盖了大部分前端开发常见的需求。Vue 的设计非常注重灵活性和“可以被逐步集成”这个特点，适合来完成类似本项目的功能较为繁杂的任务。本项目的前端将完全基于Vue框架来完成，需要利用到Vue框架中路由管理、组件系统、视图渲染等模块功能，为了更好的利用Vue框架来进行开发，需要对Vue框架的设计有所了解。

### MVVM设计模式

MVVM（Model-View-ViewModel）是一种设计模式，而Vue.js是一个流行的JavaScript框架，它实现了MVVM模式。Vue.js是基于MVVM架构设计的，可以说Vue.js是MVVM模式的一种具体实现。在Vue.js中，数据绑定是MVVM模式的关键特性。Vue实例中的数据和模板之间建立了双向绑定关系，当数据发生变化时，视图会自动更新；同时，用户在视图上的操作也会影响到数据，实现了数据的双向同步。这种数据绑定机制使得开发者能够更容易地管理数据和用户界面，提高了开发效率。



图2-1 前端MVVM设计模式

### 组件化开发

Vue将页面抽象为组件，每个组件拥有独立的状态和视图。组件可以嵌套，形成复杂的应用界面。组件化开发使得代码更易于维护和扩展，同时也促进了团队协作，不同团队成员可以专注于不同组件的开发。对于本项目，在学术成果分享交流平台的开发中，我们可以将不同功能模块，如资源检索、学术成果管理、学者门户等，抽象成独立的组件。每个组件负责自己的功能，通过组件间的数据传递和事件触发来实现功能的整合和交互。

### Vue CLI工具

Vue提供了Vue CLI脚手架工具，帮助开发者快速搭建项目基础结构，提供了开发、构建和部署的便利性。Vue CLI还集成了各种开发工具和插件，使得开发过程更加顺畅。本项目开发中Vue CLI工具将为项目创建和自定义设置起到重要作用，其提供的路由插件、状态管理插件等插件也是项目不可或缺的。

### 指令系统

Vue提供了丰富的指令系统，例如v-bind、v-model、v-for等，使得开发者可以在模板中直接操作DOM。本项目前端开发中这些指令可帮助开发者大大提高开发效率。

### html、css、javascript设计

本项目使用vue框架搭建单页面应用，因而采用客户端渲染的模式。项目中的HTML主要负责定义页面结构和Vue模板语法的应用，CSS部分使用Less预处理语言和Tailwind CSS框架，增强样式的灵活性和开发效率，JavaScript部分则使Vue框架的响应式数据绑定和组件化开发特性，以及Vue 3的新语法糖，使得前端代码更加清晰、高效。同时，通过使用ESLint插件，保障代码样式的规范性和一致性。采用这些设计能使得项目具备良好的可维护性、可扩展性和开发效率。

## 后端架构

后端架构主要基于Python语言的Django框架进行开发。Djang框架注重代码复用，允许多个组件以"插件"形式服务于整个框架，同时提供丰富的第三方插件和工具包开发支持，赋予了Django极高的可扩展性。该框架遵循MVT设计模式，MVT设计模式中最重要的是视图(view), 因为它同时与模型(model)和模板(templates)进行交互。当用户发来一个请求(request)时，Django会对请求头信息进行解析，解析出用户需要访问的url地址，然后根据路由urls.py中的定义的对应关系把请求转发到相应的视图处理。视图会从数据库读取需要的数据，指定渲染模板，最后返回响应数据。这个过程如下图所示:

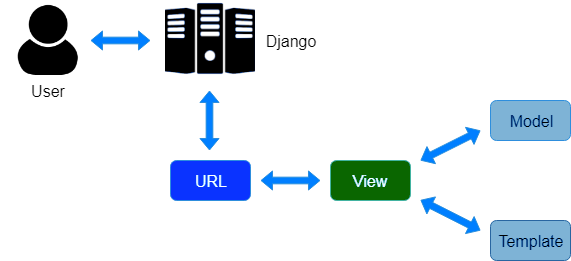


图2-2 MVT模式大致流程图

Django还遵循DRY（Don't Repeat Yourself）原则，强调快速开发，使得开发者能够高效地构建功能强大的Web应用。此外，Django框架的独特之处还在于其高度集成的特性。它内置了许多常用的功能，如认证系统、表单处理、数据库ORM（Object-Relational Mapping）等，使得开发者无需从零开始构建这些常见功能，节省了大量开发时间。

# 部署策略

## 部署架构

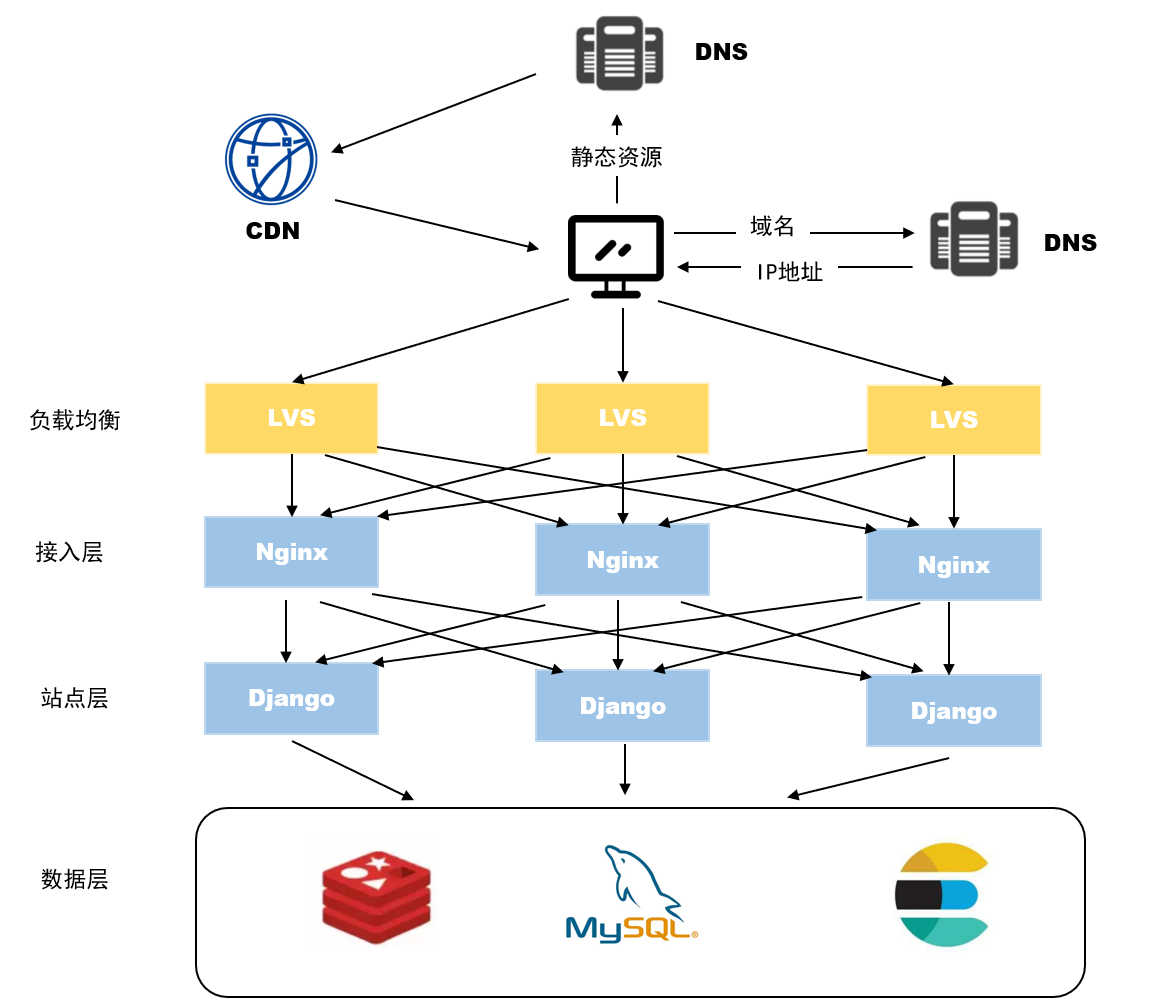


图3-1 部署架构图

为了满足网站的高并发要求，我们选择了分布式部署方式。我们将静态资源部署到了CEN上，并利用CDN返回给用户最近的节点。对于动态资源，我们使用了DNS负载均衡技术。在进行DNS解析域名时，请求会随机跳转到LVS上的一台服务器，然后通过LVS的负载均衡算法将请求分配给某个服务器。

在数据层方面，我们的系统主要以读操作为主。对于大规模的文献类型数据，我们使用了读写速度快的Elasticsearch。而对于其他信息，我们则使用了Mysql数据库和Redis缓存来完成存储和读取。

## 部署方案

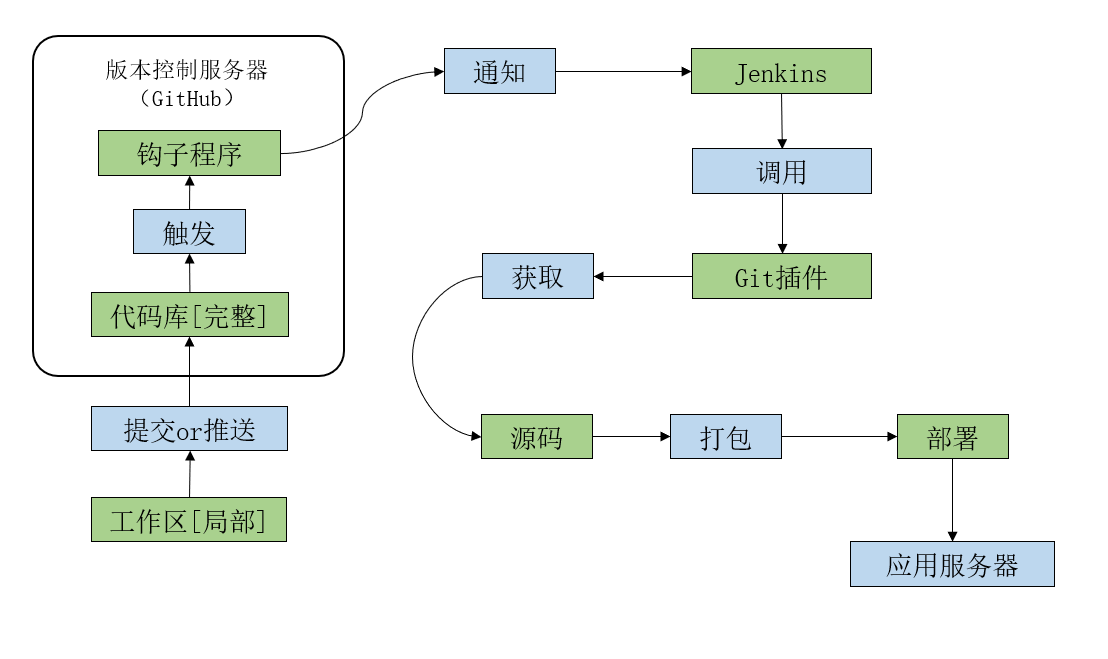


图3-2 部署方案图

在开发过程中，为了解决多个后端逻辑服务器需要维护的问题，我们决定使用Jenkins作为中心服务器来管理整个部署流程，从而实现自动化部署。

具体流程如下：当开发人员提交代码到远程代码仓库后，触发钩子程序会向Jenkins服务器发送通知。Jenkins服务器会调用Git插件来获取源代码，并使用Docker进行打包。最后，打包好的应用程序将部署到多个应用服务器上。

## 部署设备性能实例

### 高性能服务器实例

CPU：8核心

内存：64G

硬盘：一块高IO硬盘700G，一块普通硬盘80G

数量：1台

### 普通服务器实例

CPU：2核心

内存：2G

硬盘：60G

数量：4台

1. 技术选型

## 技术框架表格

### 后端技术

|  |  |
| --- | --- |
| 应用框架 | Python |
| Django |
| 代码管理及版本控制 | Git |
| 后端开发工具 | Python |
| 数据库服务器 | MySQL |
| 搜索框架 | Elasticsearch |
| 服务器部署 | Nginx |
| 缓存 | Redis |
| 消息队列 | Celery |

### 前端技术

|  |  |
| --- | --- |
| 应用框架 | Vue.js |
| 应用层 | Vuex |
| Vue-Router |
| 代码管理及版本控制 | Git |
| 前端开发工具 | VS Code/WebStorm |
| HTTP库 | Axios |
| 服务器部署 | Nginx |

## 重要技术说明

### Django

Django是一个Python语言的Web框架，旨在帮助开发者快速开发网站，它提供了许多内置的组件和工具，使得开发过程更加高效。

Django遵循MVC设计模式（Model（模型）+ View（视图）+ Controller（控制器）），这种模式使得代码组织和程序流程更加清晰和易于维护。同时，Django也提供了一个数据库抽象层，这使得开发者可以使用Python编写数据库查询，而不需要手动编写SQL语句。在扩展性方面，Django支持大规模网站，可以轻松地扩展以满足不断增长的需求。此外，Django在设计和实现时考虑了安全性，它提供了许多安全措施，例如防止SQL注入、跨站点脚本、跨站请求伪造和点击注入等攻击。

使用 Django，开发人员只需要实现很少的代码，就可以轻松地完成一个正式网站所需要的大部分内容，并进一步开发出全功能的 Web 服务，这也正是OpenStack的Horizon组件采用这种架构进行设计的主要原因。

### MySQL

本项目使用MySQL这一关系型数据库管理系统来管理本项目的数据库。

在本项目中，用户数据的量级为十万级，MySQL足以支撑本项目的十万级用户数据。

另外，采用MySQL的索引，可以大大提高数据库的检索以及其他各种操作的效率，加快项目整体运转的效率。MySQL的日志功能也可以有效地帮助我们监控数据库乃至整个项目的运行情况。

### Elasticsearch

Elasticsearch是一个基于Apache Lucene的分布式搜索和分析引擎，能够解决不断涌现出的各种用例。作为Elastic Stack的核心，Elasticsearch会集中存储数据，让用户飞快完成搜索，微调相关性，进行强大的分析，并轻松缩放规模。

Elasticsearch中存储的文档分布在不同的容器中（分片），可以进行复制以提供数据冗余副本，以防发生硬件故障。

此外，Elasticsearch 的分布式特性使得它可以扩展至数百台甚至数千台服务器，并处理PB量级的数据。

在本项目中，学术成果数据是亿级的，如果使用MySQL进行查询效率很低，而且并发量有限，同时为了预防硬件故障进行分库存储，使用起来也比较复杂。为了满足性能要求，简化工作量，我们选择使用Elasticsearch。

### Celery

Celery是一个Python的异步任务队列/作业队列，它允许将耗时的任务放到后台处理，从而不会阻塞Web应用程序的主线程。这使得Celery成为了一个非常适合处理大量消息和进行实时任务处理的工具。

Celery由三个主要部分组成：消息中间件（message broker），任务执行单元（worker）和任务执行结果存储（task result store）。其适用异步处理问题，当发送邮件、或者文件上传, 图像处理等等一些比较耗时的操作，我们可将其异步执行，这样用户不需要等待很久，提高用户体验。

另外，Celery具有高可用性和扩展能力，并且可以与第三方工具集成，使得其十分灵活。

### Nginx

Nginx是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。Nginx将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好。

### Redis

Redis（Remote Dictionary Server）是一个开源的使用ANSI C语言编写、支持网络、可基于内存亦可持久化的日志型Key-Value数据库，并提供多种语言的API。它通常被称为数据结构服务器，因为值可以是 字符串(String), 哈希(Map), 列表(List), 集合(Sets) 和 有序集合(sorted sets)等类型。这些数据类型都支持push/pop、add/remove、取交集并集和差集以及更丰富的操作，并且这些操作都是原子性的。在此基础上，Redis支持各种不同方式的排序。

为了保证效率，Redis数据都是缓存在内存中。Redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave(主从)同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。主从同步对读取操作的可扩展性和数据冗余很有帮助。

### Vue.js

Vue.js是一套用于构建用户界面的渐进式JavaScript框架，其核心库关注视图层，采用自底向上增量开发的设计。Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

使用Vue.js作为学术成果分享平台的前端应用框架，可以更好地支持实时响应式设计。同时，Vue.js操作的简便性也为开发效率的提升做出贡献。除此之外，利用Vue可以将元素集合封装成组件，方便重复使用。

# 质量属性

## 设计质量

### 概念完整性

1. 使用一致的命名规范：JavaScript变量采用camelCase命名规范。例如：userInfo, userSettings；Vue文件使用PascalCase，例如：UserProfile.vue, DashboardPage.vue；事件和组件名称使用kebab-case，例如：user-clicked, custom-button；后端相关变量也采用camelCase命名规范，例如：userProfileData, apiResponseData；前后端数据传输标识符采取snake\_case方式，例如：user\_profile\_data, api\_response\_data。

2. 统一的术语使用：为确保一致性，系统中涉及相同事物的名词应该使用相同的术语。例如，如果系统中使用了术语"用户配置"来描述用户的设置信息，那么所有开发人员和维护人员应该统一使用这个术语，而不要混用其他类似的术语，如"用户设置"或"用户偏好设置"。

3. 组件化与模块化设计：前端项目采用Vue框架。组织项目时，可以采用资源层、组件层、路由层、存储层和视图层的结构。例如，在组件层可以设计全局组件，如一个名为UserAvatar.vue的组件，以便在整个应用中重复使用。在视图层，对于每个页面文件，创建一个同名的子文件夹，并在其中存储局部组件，例如UserProfile页面可以包括一个UserProfileHeader.vue局部组件；后端项目的Django框架采用MVT（Model-View-Template）的软件设计模式，可以将代码分为模型、视图和模板层。例如，一个名为UserModel的模型可以负责用户数据的存储，而一个名为UserProfileView的视图可以负责呈现用户的个人资料页面。

### 可移植性

我们计划采用RESTful接口设计风格，这意味着我们将使用URL来定位资源，并使用HTTP动词（如GET、POST、DELETE、PUT）来描述操作。这种风格的好处在于它提高了请求的自解释性，同时允许我们通过一致的接口为不同平台提供服务，包括Web、iOS和Android，从而确保了系统的可移植性

### 可维护性

1. 前后端代码和接口代码都将进行注释，确保对其功能和使用方法有清晰的文档说明。这有助于减少在不同团队协作或未来维护时可能出现的理解偏差。

2. 除了前述部分的注释，我们还将为项目中的其他代码部分添加适当的注释，以便未来的代码维护工作更加顺利和高效。

### 复用性

前后端项目中的代码采用模块化和组件化的设计理念，有助于实现模块内高内聚，同时将模块间的耦合度降至最低水平。这意味着我们可以更容易地重用现有的代码模块，以提高开发效率和降低重复工作的风险。

## 运行质量

### 优化性能

1. 在内部网络环境中，系统的往返延迟将保持在低于1秒的水平，并且能够支持每秒高达千次的最大并发请求。当系统面对高于千级的每秒并发请求时，系统将容忍平均响应时间略微延长至1秒以上。

2. 我们将使用内容分发网络（CDN）来自动选择距离用户最近的节点来响应用户请求，从而显著降低用户的访问时延，提供更快的响应速度。

### 可延展性

部署策略将采用负载均衡技术，包括Linux Virtual Server（LVS）和Nginx，以确保系统具有出色的可扩展性。这意味着我们可以轻松地扩展系统以应对不断增长的用户和流量需求。

### 安全性

1. 服务器实例将配置专用的硬件防火墙，并利用CDN的特性来进行负载均衡、流量清洗和带宽扩展等操作，以确保系统具备抵御至少10Gbps的DDoS攻击的能力。

2. 配置SSL安全证书，以确保所有连接都使用TLS 1.3协议进行强化的加密通信。此外，还将使用ECDH密钥交换协议，以增强前向加密的安全性。

3. 对于所有敏感信息，将仅存储其经SHA-256加密处理的值，进一步提高数据安全性。

4. 定期备份数据，以确保在意外数据损失的情况下能够快速进行恢复。

## 系统质量

### 可支持性

系统包含全面的日志记录功能，自动捕获和记录日志信息，有助于快速定位和解决错误，并支持审计和监控流程的执行。

### 可测试性

采用前后端分离的架构，实现了系统的模块化和组件化设计。降低了模块间的耦合度，可以轻松地进行分模块测试，确保系统的稳定性和可靠性。

## 用户质量

### 可用性

1. 提供简洁而直观的用户界面，确保所有标识清晰可辨，按钮大小适中，易于点击。这有助于提高用户的满意度和系统的可用性。

2. 前端设计将对不同屏幕大小的设备进行自适应界面设计，以确保用户在各种设备上都能获得最佳的使用体验。

# 横切关注点

## 安全性

### 安全威胁

在系统设计中，我们必须考虑到可能的安全威胁，这些威胁包括网络上的恶意程序攻击以及可能的用户恶意操作，这些威胁可能导致系统崩溃或敏感信息泄露。因此，我们着重关注以下几个方面的安全问题：

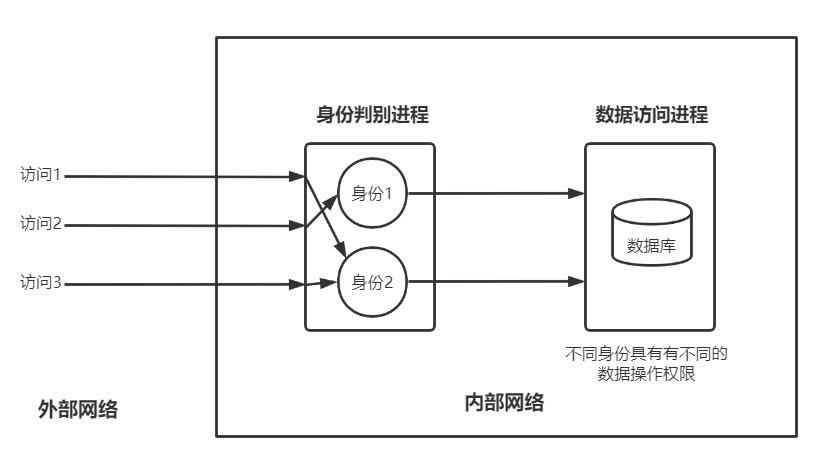
1. 防御分布式拒绝服务攻击DDoS：为了抵御潜在的网络爬虫或用户恶意行为对系统的过大压力，我们将采取一系列措施。其中之一是限制每个IP地址每小时的访问次数，或者为不同类型的访问操作分配不同的权重，然后限制每个IP地址每小时所能产生的总权重。这可以通过在负载均衡分配过程中引入一个Redis哈希表来实现，以统计访问次数或权重。

2. 用户身份认证：对于登录系统的用户，我们会进行身份验证，将用户分为不同角色，包括访客、普通用户、认证用户以及管理员。每个角色将被授予不同的权限，例如，访客只能浏览基础文献信息，而管理员则可以添加、删除文献资源以及处理论文申诉请求。

3. 授权：用户可以通过申请权限来获取对相关数据的访问权限，这些权限将由管理员进行审批。

4. 输入和数据安全：为了防止恶意输入和SQL注入攻击，我们需要对系统中的数据信息进行验证。例如，搜索操作时，不允许出现特殊符号。对于敏感信息，如密码等，需要进行加密传输，并且在数据库中不以明文形式存储。

### 安全模式



**图6-1 安全性模型**

我们的安全性架构如图6-1所示。我们的安全性模型采用了多层次的身份验证和授权机制。我们使用令牌（Token）机制对系统使用者进行身份验证，不同的身份将获得不同的权限。首先，在身份验证进程中对传输的令牌进行验证，一旦身份合法，请求将被发送到数据库操作进程，然后根据请求中携带的身份标识进行二次验证。只有当验证通过时，才允许对数据库进行操作。

## 异常处理

### 异常检测

当服务器检测到无法处理的异常情况时，我们将立即发送通知给报警系统。这个报警系统将进一步通知平台维护者，通过发送警报邮件来提醒他们采取行动。

此外，我们还将设置监测进程，以定期向系统的各个组成部分发送请求。如果在规定的时间内未能收到某个部分的响应，系统将视此部分为故障，自动触发报警通知给维护者。

### 异常恢复

1. 数据库操作是以事务为单位进行的。因此，如果出现数据库操作异常或系统宕机，我们将确保数据能够进行回滚和恢复，以维护数据的完整性。

2. 为了预防媒体损坏或数据丢失，我们将定期进行数据库的增量备份和全量备份，以确保数据的安全性。

3. 明确定义了异常恢复流程，以确保系统宕机后能够在最短时间内恢复服务的可用性。此外，我们还配置了系统，以确保服务器重启后能够自动启动各个服务进程。

## 缓存

1. 前端数据将使用LocalStorage缓存，以减少对服务器的请求，提高页面响应速度。

2. 我们将采用MySQL数据库，并使用Redis来缓存数据库查询结果，以降低数据库负载并提高数据检索速度。

3. 采用页面缓存技术，以加速动态网页的响应时间，提供更快速的用户体验。

## 日志

服务器实例将采用Python内置的logging库来记录关键信息。这个库支持多个日志级别，包括DEBUG、INFO、WARNING、ERROR和CRITICAL，用户可自定义输出配置，包括日志级别输出阈限、时间格式和内容格式等。此外，logging库天然支持线程安全，允许多个线程同时写入同一个日志文件。