# 系统设计文档

#### 技术选型

该项目从前端到后端使用了多种技术实现业务功能，包含前端技术和后端技术。

**前端技术栈：**

Vue.js + 生态 ，Vue 3。使用 Vue 3 的组合式 API 和 <script setup> 语法构建组件。同时模块化开发，如 components、views、layouts 等目录结构清晰分离 UI 组件和页面逻辑。使用TypeScript强类型语言，提升代码可维护性和 IDE 支持。使用Vue Router实现 SPA 内部路由跳转。使用Pinia 进行状态管理，替代 Vuex，轻量高效的状态管理库。同时自定义自定义 Composables（组合函数），用于封装复用逻辑，实现对不同微服务API的交互。

**后端技术栈：**（Spring Boot + 微服务生态）

Spring Boot快速搭建 RESTful Web 服务。自动配置机制、内嵌 Tomcat。Spring Cloud Gateway微服务网关，利用Nacos，统一处理请求转发和权限控制。

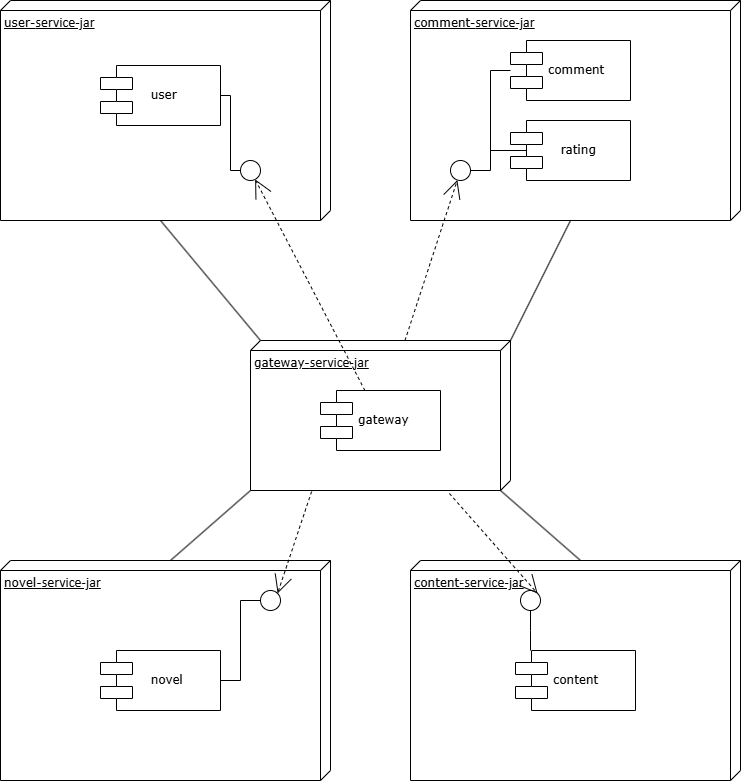
项目划分为多个独立的服务模块（comment-service, content-service, gateway-service, novel-service, user-service），每个服务专注于单一职责，易于扩展和维护。网关服务（gateway-service）负责请求路由、负载均衡、鉴权等功能。各业务服务（如 comment-service, novel-service）处理各自领域的业务逻辑。

数据访问层中使用了MyBatisPlus+ XML 映射提供灵活的数据访问能力。

安全方面，对于Spring Security / JWT，通过拦截器实现了对用户的认证授权。

对于不同的微服务，使用YML 配置管理，实现在多台服务器上部署不同的微服务。每台微服务使用Maven 构建工具进行依赖管理。

#### 系统总体架构概貌

系统组件图如图所示

系统生成5个jar包，每个jar包下属的类如下所示，根据不同服务划分成如下部分：

gateway-service-jar

|  |  |
| --- | --- |
| AuthProperties | 校验属性 |
| JwtProperties | JWT属性 |
| SecurityConfig | 生成密钥 |
| AuthGlobalFilter | 解析token并放行 |
| JwtTool | JWT工具类 |

user-service-jar

|  |  |
| --- | --- |
| register | 用户注册 |
| login | 用户登录 |
| editProfile | 修改头像 |
| editPassword | 修改密码 |
| becomeVIP | 修改VIP状态 |
| getUserById | 根据userId获取用户 |
| getUser | 直接获取用户 |

comment-service-jar

|  |  |
| --- | --- |
| createComment | 新建评论 |
| getOne | 根据id获取评论 |
| comment-service:listByNovelId | 根据小说id获取评论 |
| listByChapterId | 根据章节id获取评论 |
| createRating | 新建评分 |
| getAverageRating | 根据小说id获取平均评分 |
| rating-service:listByNovelId | 根据小说id获取评分 |

content-service-jar

|  |  |
| --- | --- |
| getUserHistory | 获取用户的阅读历史 |
| addHistory | 添加或更新阅读历史 |
| getRecentHistory | 获取用户最近的阅读历史 |
| deleteHistory | 删除特定的阅读历史 |
| getNovelChapters | 获取小说的所有章节 |
| getLatestChapter | 获取小说最新章节 |
| getChapterByOrder | 获取指定章节 |
| getChapterCount | 章节统计 |
| searchChaptersByTitle | 章节标题搜索 |
| getChaptersByPage | 分页查询 |
| getChapterById | 获取单个章节 |
| getAllNovels | 所有小说查询 |
| searchByTitle | 标题搜索 |
| searchByCategory | 分类查询 |
| searchByTag | 标签搜索 |
| searchByAuthor | 作者查询 |
| searchNovels | 组合查询 |
| getNovelById | 单本小说查询 |

novel-service-jar

|  |  |
| --- | --- |
| getNewOrder | 获取最新章节顺序 |
| addChapter | 添加章节 |
| getChapters | 获取章节 |
| addNovel | 添加小说 |
| addNovelWithCover | 添加小说（带封面） |
| getNovelsByAuthorId | 根据作者获取小说 |
| getNovelsByNovelId | 根据小说id获取小说 |

Vue组件和TypeScript组件

content-service

|  |  |
| --- | --- |
| NovelBrowse.vue | 展示小说列表 |
| NovelDetail.vue | 展示小说详情 |
| NovelReader.vue | 展示章节内容 |
| UserLibrary.vue | 展示用户收藏的小说和历史浏览数据 |
| useApiContents.ts | 和阅读小说相关的请求 |

novel-service

|  |  |
| --- | --- |
| AddNovel.vue | 创建新小说 |
| AddChapter.vue | 为小说添加章节 |
| ManageChapter.vue | 查看与管理小说的章节 |
| AuthorHome.vue | 展示作者发布的所有小说 |
| useApiNovel.ts | 作者添加、查询小说的请求 |
| useApiChapter.ts | 作者添加、查询章节的请求 |

user-service

|  |  |
| --- | --- |
| Login.vue | 用户登录 |
| Register.vue | 用户注册 |
| Profile.vue | 修改用户信息 |
| useApiUser.ts | 和用户信息相关的请求 |

comment-service

|  |  |
| --- | --- |
| CommentCard.vue | 展示一条用户评论 |
| CommentInputBox.vue | 输入用户评论 |
| CommentList.vue | 展示用户评论列表 |
| useApiComment.ts | 和评论相关的请求 |
| RatingAverage.vue | 展示平均评分 |
| RatingDistribution.vue | 展示评分分布 |

TypeScript模块

|  |  |
| --- | --- |
| useApi.ts | 对 Axios 请求进行二次封装，设置 baseURL、拦截器 |

组件部署  
 所有-jar包部署在Server端，所有Vue组件和TypeScript组件部署在Client端。  
 使用5台服务器，分别部署5个微服务，服务边界根据“高内聚、低耦合”的软件开发原则，将完成同一功能的方法分别封装在相应服务中。

服务之间的交互通过nacos服务注册中心实现，主要为请求通过gateway分发到不同服务。

系统实现了自动部署，只需将文件上传到github即可完成自动将代码部署到微服务所在服务器上。

版本升级通过自动部署即可实现。

#### 技术关键点

###### 1.转发与登录校验

本项目使用Spring Cloud Gateway 作为微服务转发和登录校验的网关。网关通过在 Spring Cloud Nacos 发现服务，我们为其配置了路由转发以及负载均衡的配置文件。

在实际测试时，我们发现云服务器在 nacos 注册的 IP 地址为服务器内网地址，会造成 403 服务不可达错误，于是我们在不同微服务的配置文件下注明了每个服务器的外网地址，帮助 gateway 成功完成转发。

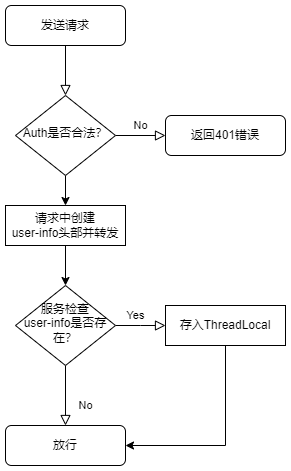
有趣的是，我们在分别服务器上远程部署了每个微服务后，在我们的电脑本地又开启了微服务，我们发现，nacos 上注册了我们本地的服务，但是地址为 192.168.x.x，此时我们部分地遇到了服务请求时间过长问题，这变相地说明我们的负载均衡配置成功了。

在用户 Token 方面，我们使用了 JWT 作为用户 Token 生成器，在 UserService 微服务登录时生成，有效期 10 分钟。

对于登录校验，我们为 gateway 配置了 globalFilter，首先获取请求头中的 Authorization，将其用 JWT 解析，如果解析无效或者校验过期，则不允许放行，返回401错误；反之，如果验证了数据有效，则将其转换为 userId，新建 user-info的请求头部，然后进行转发。

微服务接收到请求后，进入UserInfo拦截器解析user-info，并将其存入 ThreadLocal 中，方便服务内使用。

流程图如下所示：



###### 2.将图片上传至阿里云oss并返回oss链接

本项目有两个功能用到此技术，分别是用户上传头像及作者上传小说封面。  
本功能主要过程为：  
 1.前端获取需要上传的图片

2.前端将图片传给后端

3.后端将图片上传至阿里云oss容器内，并返回图片在阿里云oss容器内的对应链接

4.后端将链接传给前端

5.前端读取链接即可展示对应图片

流程图如下所示



###### 3.自动化构建部署

我们在服务打包和上传的过程中不堪其扰，繁琐的打包上传工作几乎每次commit都要进行，尤其是我们在把微服务配置好之后。

因此，我们在github上做了自动化构建部署工作，通过获取每台服务器的ip地址、用户名、ssh端口号和ssh密钥。

通过使用 scp 文件上传以及 ssh-action ssh执行命令，我们实现了每次 main 分支 commit 时，微服务打包为jar包，自动部署到远端服务器，并终止之前已经运行的程序，重启服务。

流程图如下：

