

SSO统一认证中心 - 项目架构分析文档

一、项目概述

本项目是一个基于Spring Boot + Vue的前后端分离统一认证中心（SSO）系统，用于实现多系统单点登录和统一用户管理。

技术栈

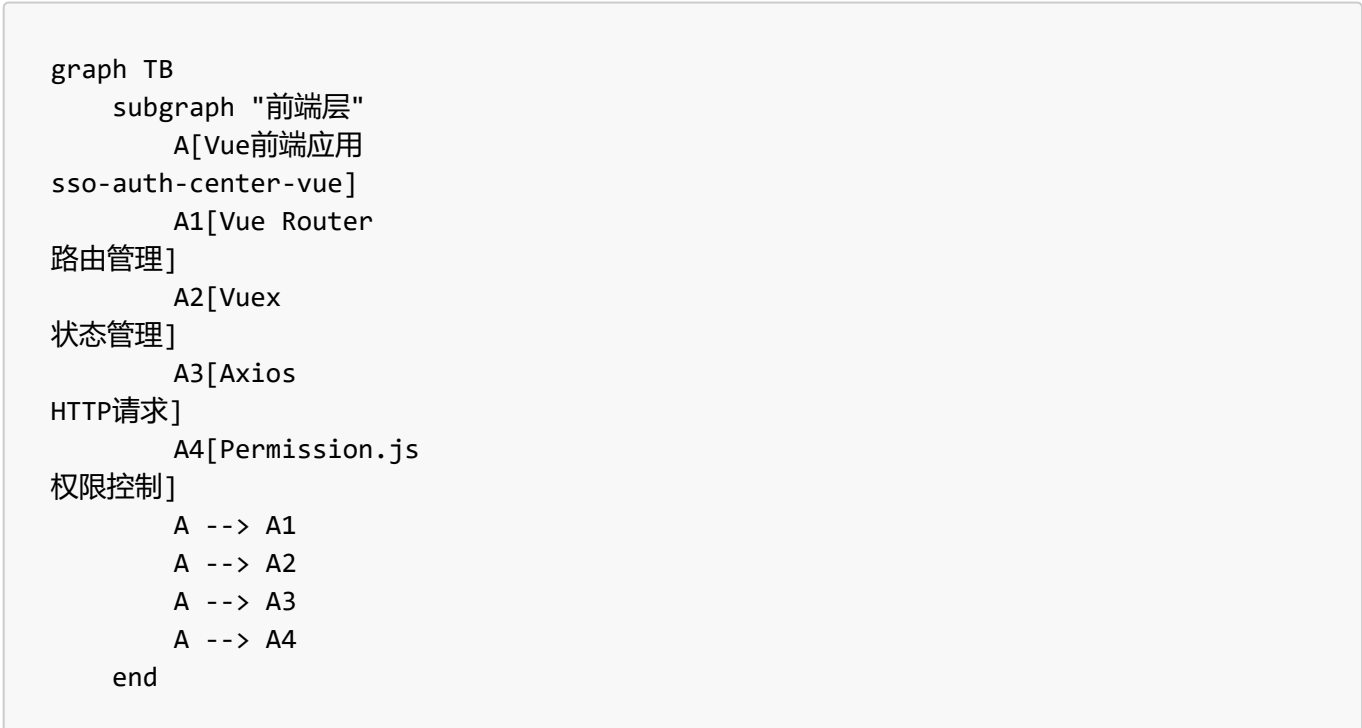
后端技术栈：

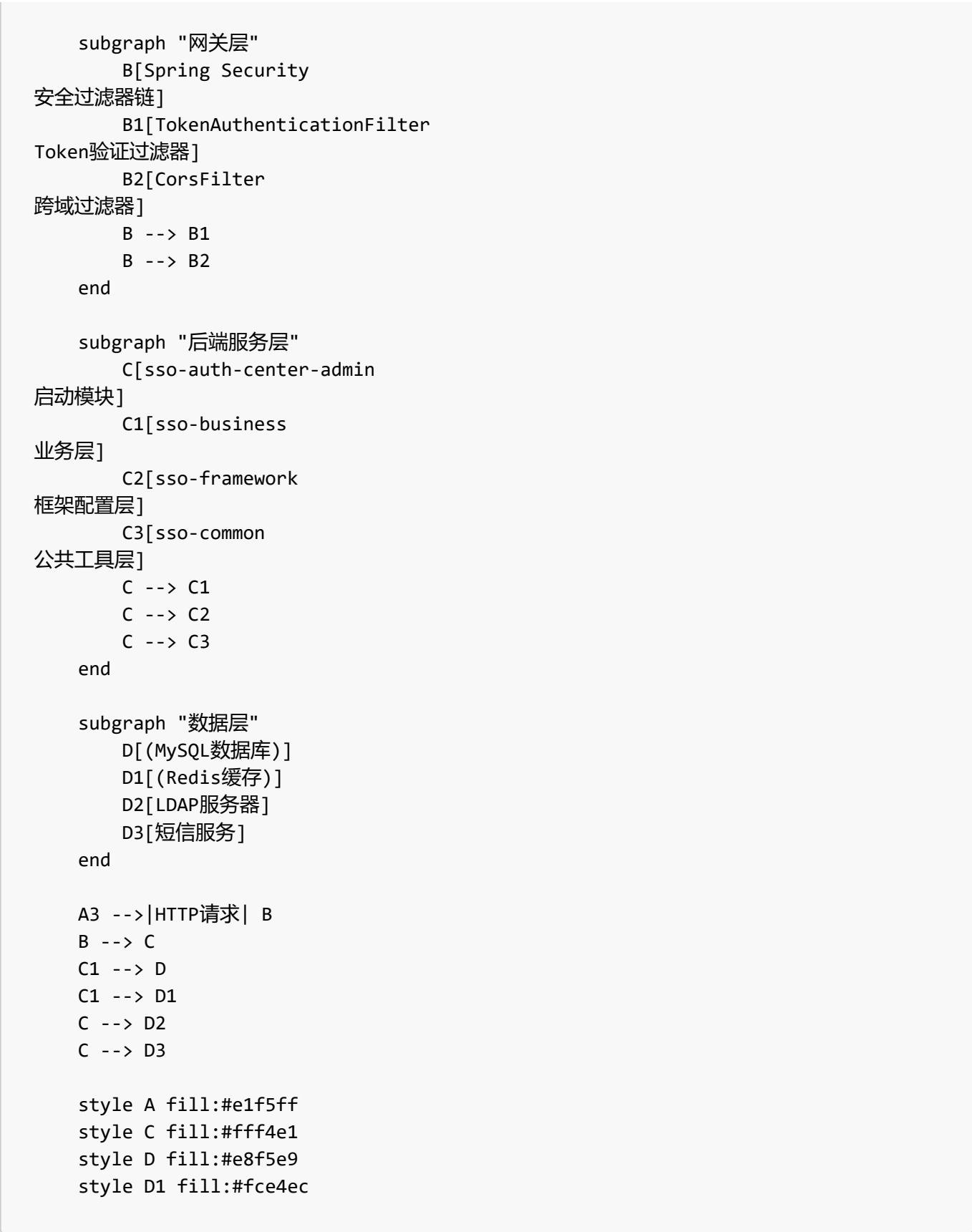
- Spring Boot 2.1.17
- Spring Security（认证授权）
- MyBatis（数据持久化）
- Redis（缓存）
- JWT（Token认证）
- Druid（数据库连接池）
- MySQL（数据库）
- LDAP（可选认证方式）

前端技术栈：

- Vue 2.6.10
- Element UI 2.13.2
- Vue Router 3.0.6
- Vuex 3.1.0
- Axios 0.18.1

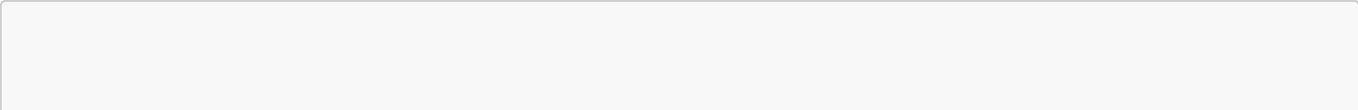
二、整体架构图





三、后端模块架构

3.1 模块分层结构



```

graph LR
    subgraph "sso-auth-center-service 父工程"
        A[sso-auth-center-admin  
启动模块]
        B[sso-business  
业务模块]
        C[sso-framework  
框架模块]
        D[sso-common  
公共模块]
    end

    A -->|依赖| B
    A -->|依赖| C
    A -->|依赖| D
    B -->|依赖| C
    B -->|依赖| D
    C -->|依赖| D

    style A fill:#ffcdd2
    style B fill:#c8e6c9
    style C fill:#fff9c4
    style D fill:#bbdefb

```

3.2 后端模块详细说明

sso-auth-center-admin (启动模块)

- **职责：** Spring Boot应用启动入口，Web层（Controller）
- **主要组件：**
 - `AuthCenterAdminApplication.java` - 启动类
 - `controller/admin/` - 管理后台Controller
 - `LoginController.java` - 登录接口
 - `UserController.java` - 用户管理
 - `RoleController.java` - 角色管理
 - `MenuController.java` - 菜单管理
 - `DeptController.java` - 部门管理
 - `SystemController.java` - 系统管理
 - `controller/getway/` - 网关接口
 - `OpenApiController.java` - 开放API接口
 - `security/` - Spring Security配置
 - `SecurityConfig.java` - 安全配置
 - `TokenAuthenticationFilter.java` - Token验证过滤器
 - `UserAccountAuthServiceImpl.java` - 用户认证服务

sso-business (业务模块)

- **职责：** 业务逻辑层和数据访问层
- **主要组件：**

- dao/ - 数据访问层
 - mapper/ - MyBatis Mapper接口
 - entity/ - 实体类
- service/ - 业务服务层
 - admin/ - 管理后台业务服务
 - SsoLoginService.java - 登录服务
 - UserService.java - 用户服务
 - RoleService.java - 角色服务
 - MenuService.java - 菜单服务
 - SystemService.java - 系统服务
 - base/ - 基础服务
 - SsoTokenService.java - Token服务
- model/ - 数据模型
 - bo/ - 业务对象 (入参)
 - vo/ - 视图对象 (出参)

sso-framework (框架模块)

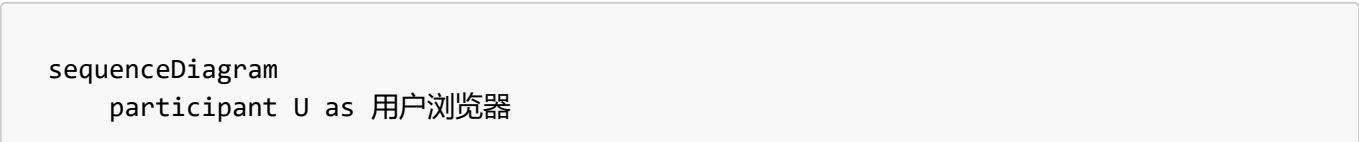
- 职责：框架核心配置
- 主要组件：
 - 数据源配置 (Druid)
 - Redis配置
 - MyBatis配置
 - 验证码配置
 - 统一异常处理
 - 网关API扫描

sso-common (公共模块)

- 职责：公共工具类和常量
- 主要组件：
 - utils/ - 工具类
 - sms/ - 短信工具
 - ip/ - IP工具
 - 其他工具类
 - constant/ - 常量定义
 - enums/ - 枚举类
 - exception/ - 异常类
 - annotation/ - 自定义注解

四、认证授权流程

4.1 登录认证流程图



```

participant V as Vue前端
participant F as Security过滤器
participant C as LoginController
participant L as SsoLoginService
participant A as AuthenticationManager
participant T as TokenService
participant R as Redis
participant D as MySQL数据库

U->>V: 1. 访问登录页面
V->>V: 2. 获取验证码
V->>V: 3. 用户输入账号密码
U->>V: 4. 提交登录表单
V->>F: 5. POST /login (携带验证码)
F->>F: 6. 检查是否为匿名访问路径
F->>C: 7. 转发到LoginController
C->>L: 8. 调用login方法
L->>L: 9. 验证码校验
L->>A: 10. 账号密码验证
A->>D: 11. 查询用户信息 (数据库/LDAP)
D-->>A: 12. 返回用户信息
A-->>L: 13. 认证成功
L->>T: 14. 生成JWT Token
T->>R: 15. 存储Token到Redis
T-->>L: 16. 返回Token
L->>D: 17. 记录登录日志
L->>D: 18. 更新最后登录时间
L->>R: 19. 保存在线用户记录
L-->>C: 20. 返回Token
C-->>V: 21. 返回Token给前端
V->>V: 22. 保存Token到Cookie/LocalStorage
V->>V: 23. 跳转到首页或重定向URL

```

4.2 Token验证流程

```

sequenceDiagram
    participant U as 用户浏览器
    participant V as Vue前端
    participant F as TokenAuthenticationFilter
    participant T as TokenService
    participant R as Redis
    participant C as Controller

    U->>V: 1. 发起业务请求
    V->>F: 2. HTTP请求 (携带Authorization Header)
    F->>F: 3. 提取Token
    F->>T: 4. 验证Token有效性
    T->>R: 5. 从Redis获取Token信息
    R-->>T: 6. 返回Token信息
    alt Token有效
        T-->>F: 7. Token验证通过
    end

```

```
F->>F: 8. 设置SecurityContext
F->>C: 9. 转发到Controller
C-->>V: 10. 返回业务数据
else Token无效/过期
  T-->>F: 7. Token验证失败
  F-->>V: 8. 返回401未授权
  V->>V: 9. 跳转到登录页
end
```

4.3 权限验证流程

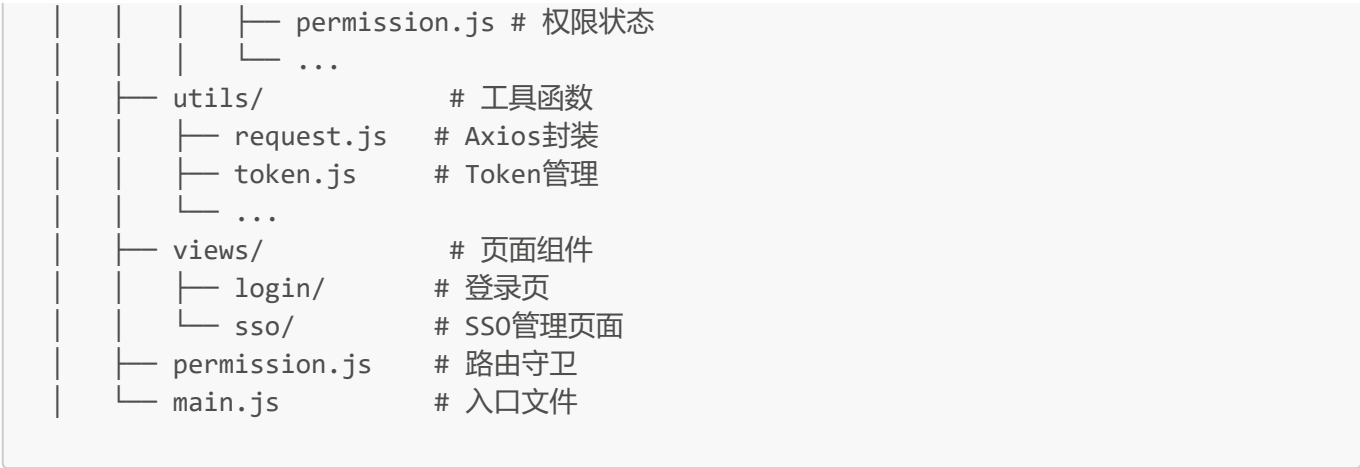
```
graph TD
  A[用户请求资源] --> B{是否有Token?}
  B -->|否| C[返回401,跳转登录页]
  B -->|是| D[Token验证]
  D --> E{Token是否有效?}
  E -->|否| C
  E -->|是| F[获取用户信息]
  F --> G[获取用户角色和权限]
  G --> H{资源是否需要权限?}
  H -->|否| I[允许访问]
  H -->|是| J{用户是否有权限?}
  J -->|是| I
  J -->|否| K[返回403,权限不足]

  style C fill:#ffcdd2
  style K fill:#ffcdd2
  style I fill:#c8e6c9
```

五、前端架构

5.1 前端目录结构

```
sso-auth-center-vue/
├── src/
│   ├── api/                # API接口定义
│   │   ├── login.js        # 登录API
│   │   ├── user.js         # 用户API
│   │   └── ...
│   ├── assets/             # 静态资源
│   ├── components/         # 公共组件
│   ├── directive/          # 自定义指令（权限）
│   ├── layout/             # 布局组件
│   ├── router/             # 路由配置
│   │   └── index.js        # 路由定义
│   ├── store/              # Vuex状态管理
│   │   └── modules/
│   │       └── user.js     # 用户状态
```



5.2 前端请求流程



5.3 前端路由守卫流程



```
style E fill:#ffcdd2
style K fill:#ffcdd2
style M fill:#ffcdd2
style D fill:#c8e6c9
```

六、数据模型关系

6.1 核心数据表关系

```
erDiagram
    SSO_USER ||--o{ SSO_USER_ROLE : has
    SSO_USER ||--o{ SSO_USER_DEPT : belongs_to
    SSO_USER ||--o{ SSO_USER_SYSTEM : can_access
    SSO_USER ||--o{ SSO_LOGIN_LOG : generates
    SSO_USER ||--o{ SSO_ONLINE_USER : online

    SSO_ROLE ||--o{ SSO_USER_ROLE : assigned_to
    SSO_ROLE ||--o{ SSO_ROLE_MENU : has

    SSO_MENU ||--o{ SSO_ROLE_MENU : assigned_to

    SSO_DEPT ||--o{ SSO_USER_DEPT : contains
    SSO_DEPT ||--|| SSO_DEPT_TREE_PATH : tree_path

    SSO_SYSTEM ||--o{ SSO_USER_SYSTEM : accessible_by
    SSO_SYSTEM ||--o{ SSO_SYSTEM_MANAGER : managed_by

    SSO_USER {
        bigint user_id PK
        string username
        string password
        string real_name
        datetime create_time
    }

    SSO_ROLE {
        bigint role_id PK
        string role_name
        string role_code
    }

    SSO_MENU {
        bigint menu_id PK
        string menu_name
        string menu_url
    }

    SSO_DEPT {
        bigint dept_id PK
```



```
        string dept_name
    }

    SSO_SYSTEM {
        bigint system_id PK
        string system_code
        string system_name
    }
```

七、关键技术点说明

7.1 双重认证机制

系统支持两种认证方式：

1. **数据库认证**：通过[UserAccountAuthServiceImpl](#)实现，查询MySQL数据库验证用户
2. **LDAP认证**：通过[CustomLdapUserDetailsMapper](#)实现，连接LDAP服务器验证用户

两种认证方式在Spring Security中同时配置，优先尝试LDAP认证。

7.2 JWT Token机制

- Token生成：使用JWT库生成，包含用户信息、权限等
- Token存储：Token存储在Redis中，用于快速验证和单点登出
- Token刷新：可通过刷新Token机制延长登录有效期

7.3 无状态会话

- 使用JWT Token代替Session，实现无状态认证
- 所有状态信息存储在Token和Redis中
- 支持分布式部署，无需Session共享

7.4 跨域处理

- 通过[CorsFilter](#)处理跨域请求
- 支持前后端分离部署
- 允许携带认证Token的跨域请求

7.5 权限控制

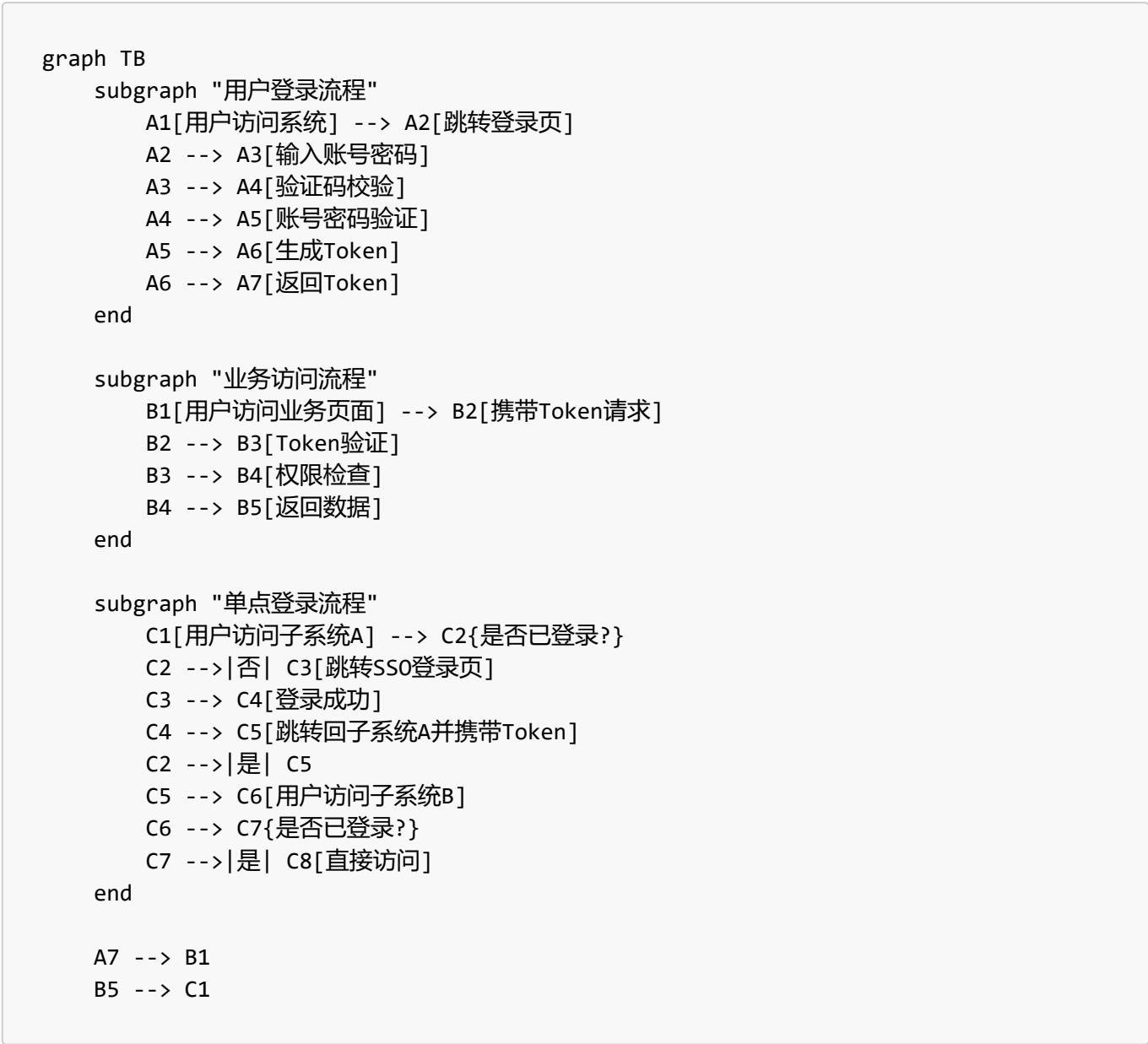
- **后端权限**：使用Spring Security的[@PreAuthorize](#)注解进行方法级权限控制
- **前端权限**：通过路由守卫和权限指令控制页面和按钮的显示

7.6 单点登录 (SSO)

- 支持多个系统接入
- 通过Token传递实现单点登录
- 支持跨域单点登录（通过redirectUrl参数）

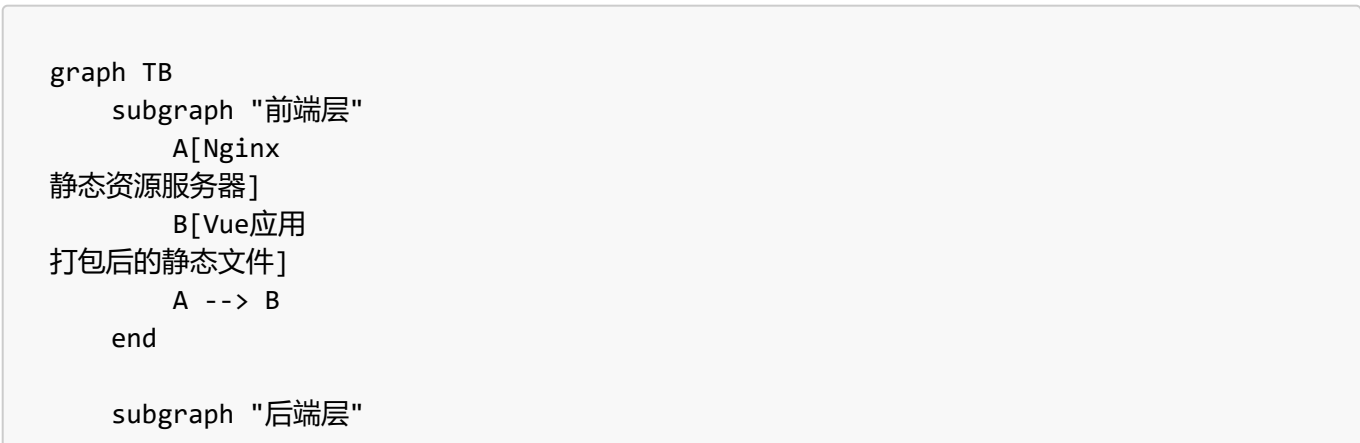
八、系统交互流程

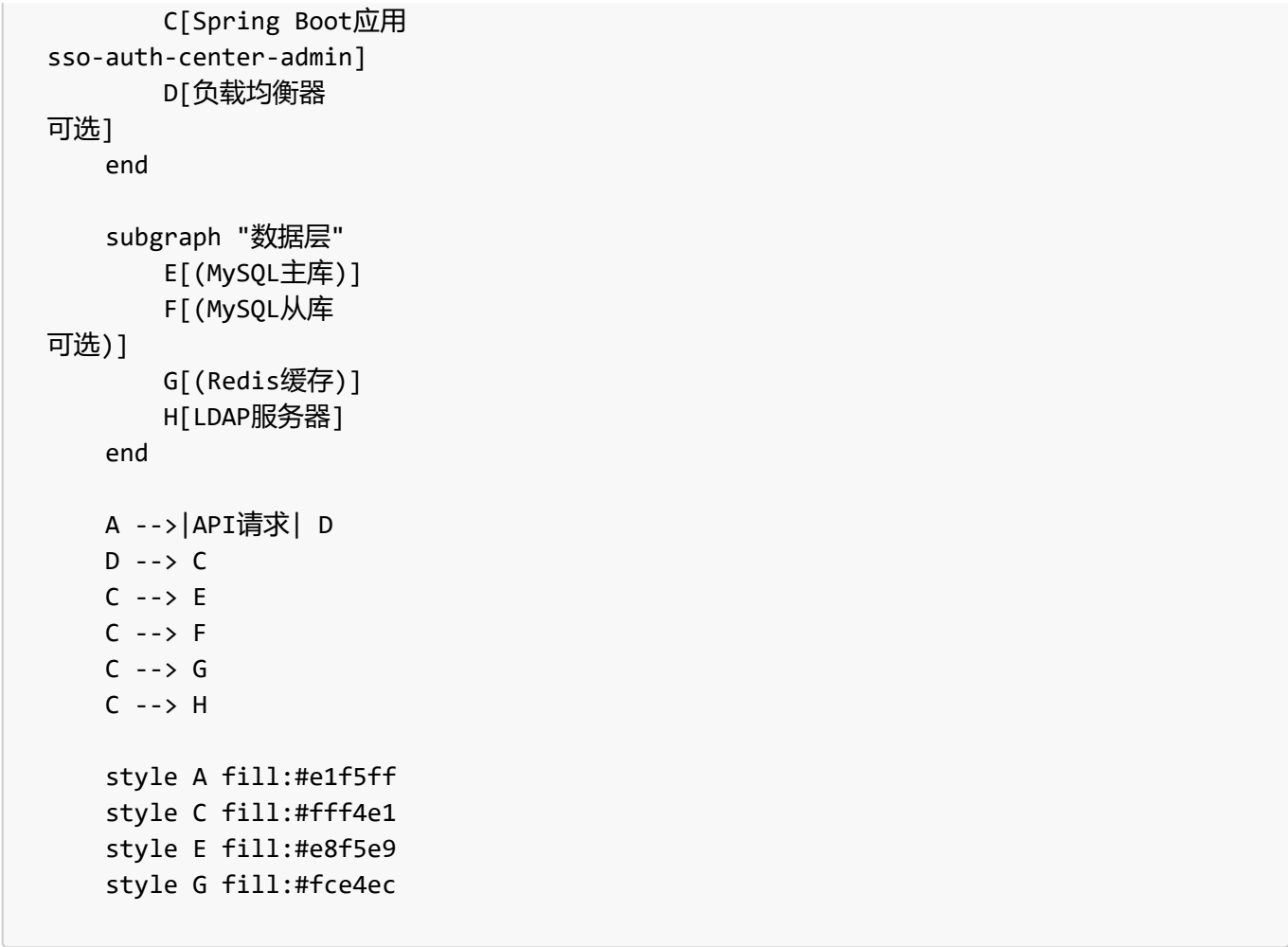
8.1 完整业务流程



九、部署架构

9.1 部署架构图





十、总结

10.1 架构特点

- 1. **前后端分离**: 前端Vue + 后端Spring Boot, 通过RESTful API通信
- 2. **模块化设计**: 后端采用Maven多模块结构, 职责清晰
- 3. **安全可靠**: Spring Security + JWT实现无状态认证
- 4. **灵活扩展**: 支持多种认证方式, 易于接入新系统
- 5. **高可用性**: 支持分布式部署, Redis缓存提升性能

10.2 主要功能模块

- ☒ 用户管理 (增删改查)
- ☒ 角色管理 (角色权限配置)
- ☒ 菜单管理 (动态菜单)
- ☒ 部门管理 (组织架构)
- ☒ 系统管理 (多系统接入)
- ☒ 登录日志 (登录记录)
- ☒ 在线用户 (在线用户监控)
- ☒ 单点登录 (SSO功能)

10.3 技术亮点

-  双重认证 (数据库 + LDAP)
-  JWT无状态认证
-  Redis缓存优化
-  Spring Security安全框架
-  响应式前端设计
-  单点登录支持
-  完整的日志记录
-  在线用户监控