Shiro权限认证设计与应用介绍

本系统采用Apache Shiro作为后端权限认证和会话管理的核心安全框架，配合Spring生态，实现了安全、灵活、可扩展的用户认证与会话托管。以下内容面向技术团队或答辩场景，从Shiro原理、分层过程、配置细节、关键代码、流程模型与典型用例等多维度系统阐述本项目Shiro的使用方式和优势。

一、Shiro原理概述

1.1 核心概念

- **Subject**：当前用户实体（可以是人、进程、服务等）。
- **SecurityManager**：安全管理器，Shiro的核心调度组件，负责用户认证、授权、会话、加密等。
- **Realm**：数据域，负责从业务数据源（如数据库）加载用户账号、密码和权限，完成认证与授权。
- **Token**：认证令牌，包含用户输入的身份凭证（如用户名/密码）。

1.2 认证流程原理

1. 前端提交用户名密码
2. 系统封装为`Token`，交由`Subject.login(token)`
3. `SecurityManager`调用配置好的`Realm`，拉取用户数据进行加密比对
4. 认证通过则Subject变为已登录，否则抛出认证异常

二、Shiro在本项目中的分层与配置

2.1 分层结构

- **配置层（ShiroConfig.java）**
  负责Shiro整体集成、Redis分布式会话、加密算法、过滤规则、主安全管理器等Bean配置
- **安全域层（UserRealm.java）**
  自定义登录认证逻辑，集成持久层（UserMapper），实现Shiro对本地数据库的安全适配
- **业务服务层（UserService.java）**
  提供加密、注册、登录、信息查询等接口，配合Shiro完成认证、加密、登出等操作

2.2 关键配置解读

配置主线（ShiroConfig.java）：

- **Redis分布式会话支持**
  通过`RedisManager`、`RedisSessionDAO`实现横向扩展下的会话共享（含超时/前缀/连接参数）
- **会话管理器**
  `DefaultWebSessionManager`托管所有Subject会话，统一超时与ID策略
- **安全域（Realm）配置**
  注入`UserMapper`，定制`UserRealm`，配置SHA-256+1024次迭代的加密匹配器
- **安全核心管理器**
  `DefaultWebSecurityManager`集成Realm与会话管理，绑定到全局`SecurityUtils`

- **过滤链定义**
  明确哪些URL匿名可访问（如登录/注册），其余全部需认证（`authc`）

UserRealm.java（认证逻辑核心）：

- **doGetAuthenticationInfo**
  - 拉取用户信息（userMapper.findByUsername）
  - 检查用户存在、状态、加密盐与密文
  - 返回`SimpleAuthenticationInfo`（Shiro自动用此对象与Token做加密比对）
- **doGetAuthorizationInfo**
  - 当前项目未做细粒度授权（返回null），后续可扩展角色/权限模型

三、用户认证与会话管理全流程

3.1 登录认证流程

过程模型

```
[前端] -> /user/login -> [UserService.login] -> [Shiro] -> [UserRealm] ->
[UserMapper] -> [数据库]
```

关键代码

```
// UserService.java
public String login(String username,String password) {
    Subject subject = SecurityUtils.getSubject();
    UsernamePasswordToken token = new UsernamePasswordToken(username, password);
    try {
        subject.login(token); // 触发UserRealm#doGetAuthenticationInfo
        return "登录成功";
    } catch (AuthenticationException e) {
        return "登录失败: " + e.getMessage();
    }
}
```

```
// UserRealm.java
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token)
throws AuthenticationException {
    String username = (String) token.getPrincipal();
    User user = userMapper.findByUsername(username);
    if (user == null) throw new UnknownAccountException("用户不存在");
    if ("locked".equals(user.getStatus())) throw new LockedAccountException("账户
已锁定");
    if (user.getSalt() == null || user.getPwd() == null) throw new
```

```
AuthenticationException("凭证不完整");
    return new SimpleAuthenticationInfo(
        username,
        user.getPwd(),
        ByteSource.Util.bytes(user.getSalt()),
        getName()
    );
}
```

- **认证原理**：Shiro自动将Token中的明文密码，与数据库中密文（user.getPwd()）+盐（user.getSalt()）+Hash算法做比对，完全不用手写加密逻辑。

3.2 密码加密与存储

```
// UserService.java
public String encryptPassword(String password, String salt) {
    return new SimpleHash("SHA-256", password, ByteSource.Util.bytes(salt),
1024).toHex();
}
```

- 保证数据库永远不存明文，且每用户独立盐值，抗爆破性强。

3.3 会话与权限控制

- **认证通过后**：Shiro自动生成会话（Session），并托管于Redis，实现分布式与持久化。
- **全局Subject访问**：任何地方都能通过`SecurityUtils.getSubject()`实时获取当前登录用户。
- **登出流程**：`SecurityUtils.getSubject().logout()`即可安全销毁会话。

3.4 过滤链控制

- 关键接口如`/user/login`、`/user/add`允许匿名访问，其余全部需先登录认证（保证安全边界）。

四、典型用例与过程示例

4.1 新用户注册

1. 前端提交用户名/密码
2. Service层用`encryptPassword`加密密码，生成随机盐存库
3. 后续登录自动用相同算法/盐进行比对

4.2 用户登录

1. Controller调用`userService.login`，封装Token并传递给Shiro
2. Shiro自动查找UserRealm拉取用户信息、比对密码，认证通过后生成Session
3. 后续每个请求自动带SessionId，Shiro从Redis恢复会话状态

4.3 权限控制/扩展（可选）

- 目前只做了认证（是否登录）控制
- 若需RBAC/细粒度权限，只需在UserRealm#doGetAuthorizationInfo实现分配角色/权限即可

五、Shiro分层设计优势

5.1 为什么这样分层

- **职责单一，逻辑清晰**：配置、认证逻辑、业务服务、数据访问各司其职，便于维护和扩展
- **数据同步和一致性**：所有认证和会话均由Shiro统一调度和Redis集中托管，保证分布式场景下用户状态一致
- **高安全性**：加密、会话、异常处理全部交由Shiro高强度方案实现
- **可测试性强**：每层可独立Mock，支持自动化测试
- **可扩展性好**：后续如接入OAuth2/第三方登录、细粒度权限、踢人下线等都可方便集成

5.2 分层过程举例

以一次登录请求为例：

- 请求先经Shiro过滤器（配置层），被放行到UserService（业务层）
- UserService调用Shiro SecurityUtils（认证逻辑层），触发UserRealm（安全域层）的认证方法
- UserRealm内部调用UserMapper（数据访问层）拉取用户数据
- 认证通过后，Shiro在Redis中存储用户Session

总结：每一层只负责自己关心的部分，既保证了数据流的清晰和安全，也提升了系统的健壮性和可维护性。

六、总结

通过以上Shiro安全架构，本系统实现了高标准的用户认证与会话管理，保障数据安全与业务稳定。分层设计模式、分布式会话方案与灵活的权限扩展能力，使得系统能够适应多种业务拓展与安全需求，是现代企业级后端开发的推荐实践。