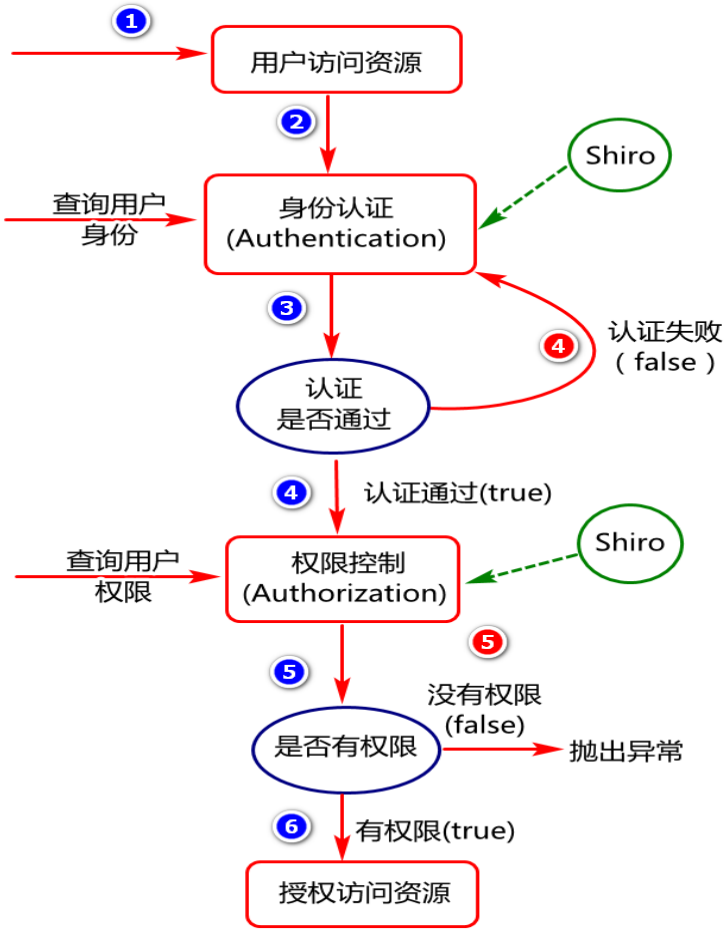
动吧权限管理子系统
Shiro 安全框架

1.Shiro 安全框架简介

1.1. Shiro 概述

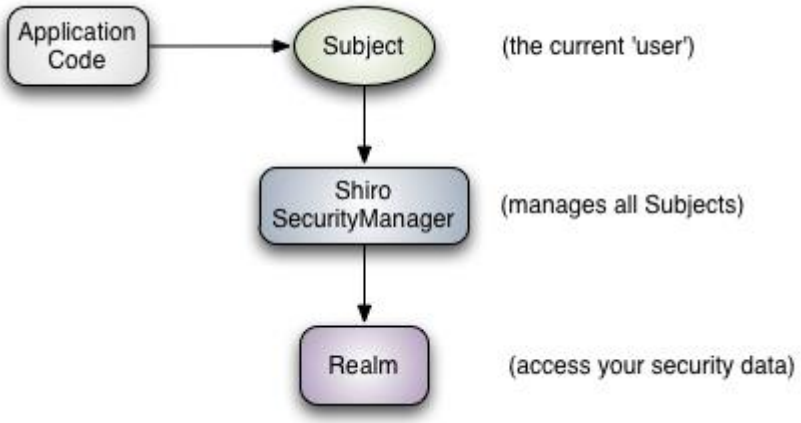
Shiro 是 apache 旗下一个开源安全框架，它将软件系统的安全认证相关的功能抽取出来，实现用户身份认证，权限授权、加密、会话管理等功能，组成了一个通用的安全认证框架，使用 shiro 就可以非常快速的完成认证、授权等功能的开发，降低系统成本。

用户资源访问控制,流程分析：



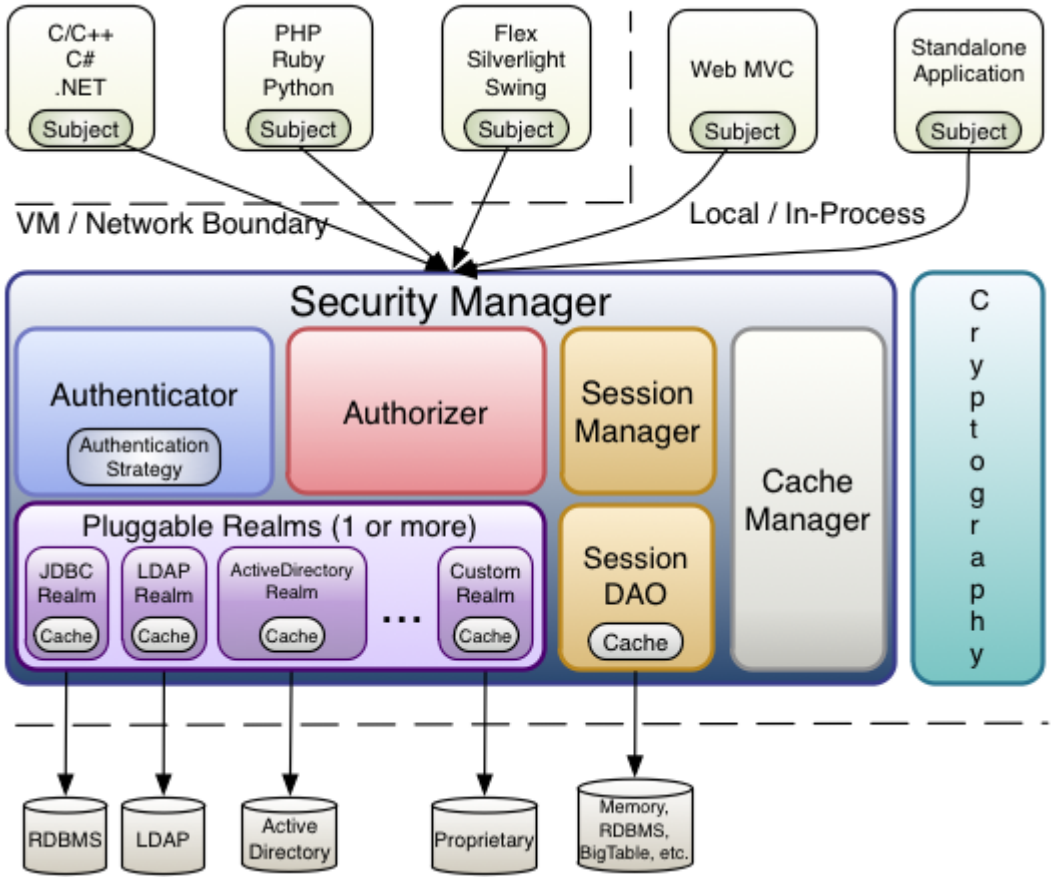
1.2. Shiro 概要架构

在概念层，Shiro 架构包含三个主要的理念：Subject,SecurityManager 和 Realm。



1.3. Shiro 详细架构

Shiro 的核心架构思想如下图所示：



通过 Shiro 框架进行权限管理时，要涉及到的一些核心对象，主要包括：认证管理对象，授权管理对象，会话管理对象，缓存管理对象，加密管理对象以及 Realm 管理对象(领域对象：负责处理认证和授权领域的数据访问题)

1) Subject（主体）:与软件交互的一个特定的实体（用户、第三方服务等）。
2) SecurityManager(安全管理器):Shiro 的核心，用来协调管理组件工作。
3) Authenticator(认证管理器):负责执行认证操作
4) Authorizer(授权管理器):负责授权检测
5) SessionManager(会话管理):负责创建并管理用户 Session 生命周期,提供一个强有力的 Session 体验。
6) SessionDAO:代表 SessionManager 执行 Session 持久（CRUD）动作,它允许任何存储的数据挂接到 session 管理基础上。
7) CacheManager（缓存管理器）:提供创建缓存实例和管理缓存生命周期的功能
8) Cryptography(加密管理器):提供了加密方式的设计及管理。
9) Realms(领域对象):是 shiro 和你的应用程序安全数据之间的桥梁。

2.Shiro 框架认证拦截实现

2.1. 添加 shiro 依赖

添加 shiro 框架依赖（spring 整合 shiro）

```
<dependency>
   <groupId>org.apache.shiro</groupId>
   <artifactId>shiro-spring</artifactId>
   <version>1.3.2</version>
</dependency>
```

2.2. 配置核心对象

创建 spring-shiro.xml 配置文件(tomcat 启动时候要加载此配置文件文件),并添加如下配置:

配置 SecurityManager 对象

```
<bean id="securityManager"
            class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

</bean>
```

配置 ShiroFilterFactoryBean 对象

```
<bean id="shiroFilterFactory"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <property name="SecurityManager" ref="securityManager"/>
        <!-- 设置请求过滤规则 -->
        <property name="FilterChainDefinitionMap">
           <map>
              <entry key="/bower_components/**" value="anon"/>
              <entry key="/build/**" value="anon"/>
              <entry key="/dist/**" value="anon"/>
              <entry key="/plugins/**" value="anon"/>
              <entry key="/**" value="authc"/><!-- 必须认证 -->
           </map>
        </property>
</bean>
```

说明:spring-shiro 配置文件需要在 spring-configs.xml 进行导入.

在项目的 web.xml 文件中添加如下配置:

```
<filter>
    <filter-name>shiroFilter</filter-name>
<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
    class>
    <init-param>
      <param-name>targetBeanName</param-name>
      <param-value>shiroFilterFactory</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

说明:

1) targetBeanName 名字由 DelegatingFilterProxy 对象底层设置并读取。
2) shiroFilterFactory 名字要与 ShiroFilterFactoryBean 配置的 id 相同。

2.3. 服务端实现

在 PageController 中添加一个呈现登录页面的方法：

```
@RequestMapping("doLoginUI")
public String doLoginUI(){
        return "login";
}
```

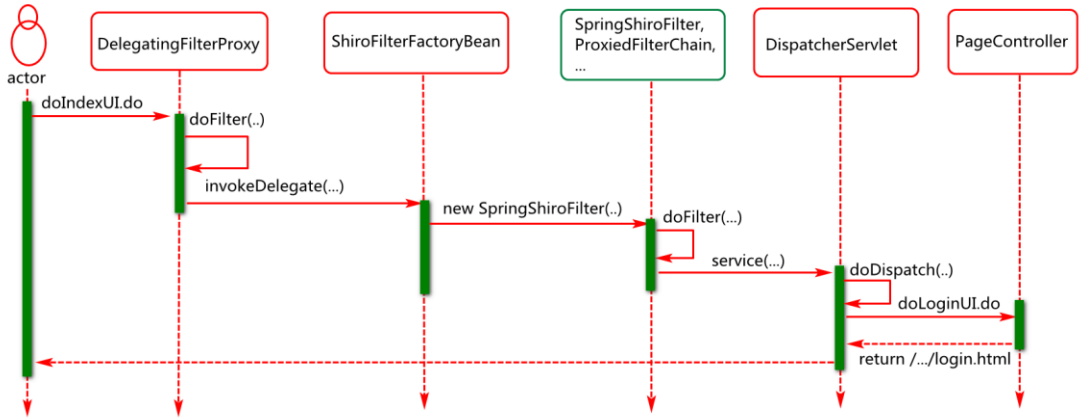
在 spring-shiro.xml 中的 shiroFilterFactorybean 的配置中添加如下配置

```
<property name="LoginUrl" value="/doLoginUI.do"/>
```

2.4. 客户端实现

在/WEB-INF/pages/添加一个 login.html 页面。

页面呈现流程分析：时序图

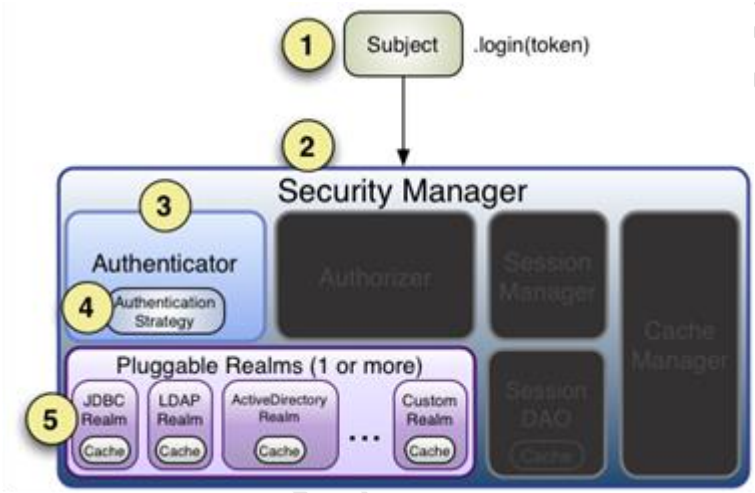


3.Shiro 框架认证过程实现

本讲的 shiro 应用主要讲解 shiro 是集成到 spring 如何实现权限控制

3.1. 认证流程分析

身份认证：判定用户是否是系统的合法用户。
用户访问系统资源时的认证（对用户身份信息的认证）流程如下：



具体流程分析如下：

1)系统调用 subject 的 login 方法将用户信息提交给 SecurityManager
2)SecurityManager 将认证操作委托给认证器对象 Authenticator
3)Authenticator 将身份信息传递给 Realm。
4)Realm 访问数据库获取用户信息然后对信息进行封装并返回。
5)Authenticator 对 realm 返回的信息进行身份认证。

思考：不使用 shiro 框架如何完成认证操作？filter，intercetor

3.2. 认证服务端实现

3.2.1. Dao 接口实现

业务描述

在 SysUserDao 中根据用户名获取用户对象

业务实现(根据用户名查询用户对象的方法定义)

1）返回值 SysUser

2）方法名 findUserByUserName

3）参数列表（String username）

代码实现：

```
SysUser findUserByUserName(String username);
```

3.2.2. Mapper 元素定义

根据 SysUserDao 中定义的方法，添加元素定义

```
<select id="findUserByUserName"
        resultType="com.jt.sys.entity.SysUser">
    select *
    from sys_users
    where username=#{username}
</select>
```

3.2.3. Service 接口实现

业务描述

本模块的 service 可以借助 realm 实现，我们编写 realm 时可以继承 AuthorizingRealm 并重写相关方法完成相关业务的实现。

业务实现：(创建 realm 类并重写相关方法)

1） 包名:com.jt.sys.service.realm

2) 类名:ShiroUserRealm

3) 方法:AuthenticationInfo （完成认证信息的获取与封装）

```
@Service
public class ShiroUserRealm extends AuthorizingRealm {

    @Autowired
    private SysUserDao sysUserDao;

    /**
     * 设置凭证匹配器
     */
    @Override
    public void setCredentialsMatcher(
        CredentialsMatcher credentialsMatcher) {
        //构建凭证匹配对象
        HashedCredentialsMatcher cMatcher=
        new HashedCredentialsMatcher();
        //设置加密算法
        cMatcher.setHashAlgorithmName("MD5");
        //设置加密次数
        cMatcher.setHashIterations(1);
        super.setCredentialsMatcher(cMatcher);
    }
    /**
     * 通过此方法完成认证数据的获取及封装,系统
     * 底层会将认证数据传递认证管理器，由认证
     * 管理器完成认证操作。
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token)
            throws AuthenticationException {
        //1.获取用户名(用户页面输入)
        UsernamePasswordToken upToken=
        (UsernamePasswordToken)token;
        String username=upToken.getUsername();
        //2.基于用户名查询用户信息
        SysUser user=
        sysUserDao.findUserByUserName(username);
        //3.判定用户是否存在
        if(user==null)
        throw new UnknownAccountException();
        //4.判定用户是否已被禁用。
```

```
        if(user.getValid()==0)
        throw new LockedAccountException();

        //5.封装用户信息
        ByteSource credentialsSalt=
        ByteSource.Util.bytes(user.getSalt());
        //记住：构建什么对象要看方法的返回值
        SimpleAuthenticationInfo info=
        new SimpleAuthenticationInfo(
                user,//principal (身份)
                user.getPassword(),//hashedCredentials
                credentialsSalt, //credentialsSalt
                getName());//realName
        //6.返回封装结果
        return info;//返回值会传递给认证管理器(后续
        //认证管理器会通过此信息完成认证操作)
    }
    ....
}
```

对此 realm 在 spring-shiro.xml 文件中以属性的形式注入给 SecurityManager 对象

```
<bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="Realm" ref="shiroUserRealm"/>
</bean>
```

3.2.4. Controller 类实现

业务描述

1）在 SysUserController 添加登录方法 doLogin

2）接收用户名及密码参数，并对其进行有效验证

3）执行登录认证

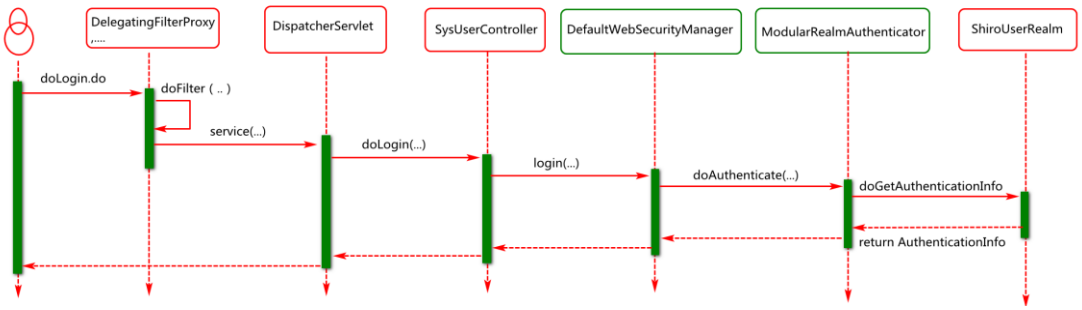
代码实现

```
@RequestMapping("doLogin")
@ResponseBody
public JsonResult doLogin(String username,String password){
    //1.获取Subject对象
    Subject subject=SecurityUtils.getSubject();
    //2.通过Subject提交用户信息,交给shiro框架进行认证操作
    //2.1对用户进行封装
    UsernamePasswordToken token=
    new UsernamePasswordToken(
            username,//身份信息
            password);//凭证信息
    //2.2对用户信息进行身份认证
    subject.login(token);
    //分析:
    //1)token会传给shiro的SecurityManager
    //2)SecurityManager将token传递给认证管理器
    //3)认证管理器会将token传递给realm
    return new JsonResult("login ok");
}
```

说明：此控制层的方法映射必须允许匿名访问。需要在 spring-shiro.xml 配置文件中对/user/doLogin.do 这个路径进行匿名访问的配置，例如

```
<entry key="/user/doLogin.do" value="anon"/>
```

业务流程分析：



3.3. 认证客户端实现

3.3.1. 编写用户登陆页面

在 WEB-INF/pages/目录下添加登陆页面(login.html)

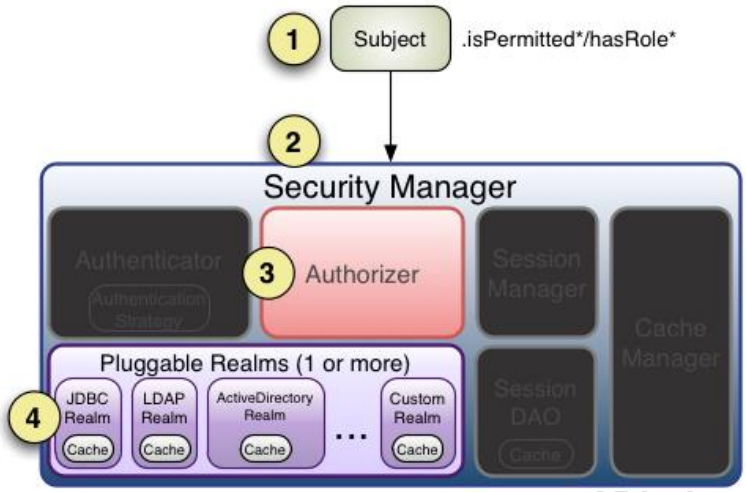
3.3.2. 异步登陆操作实现

```
$(function () {
  $(".login-box-body").on("click",".btn",doLogin);
});
function doLogin(){
    var params={
       username:$("#usernameId").val(),
       password:$("#passwordId").val()
    }
    var url="user/doLogin.do";
    $.post(url,params,function(result){
        if(result.state==1){
          //跳转到indexUI对应的页面
          location.href="doIndexUI.do?t="+Math.random();
        }else{
          $(".login-box-msg").html(result.message);
        }
    });
}
```

4.Shiro 框架授权过程实现

4.1. 授权流程分析

授权：对用户资源访问的授权（是否允许用户访问此资源）
用户访问系统资源时的授权流程如下：



1)系统调用 subject 相关方法将用户信息(例如 isPermitted)递交给 SecurityManager
2)SecurityManager 将权限检测操作委托给 Authorizer 对象
3)Authorizer 将用户信息委托给 realm.
4)Realm 访问数据库获取用户权限信息并封装。
5) Authorizer 对用户授权信息进行判定。

思考：思考不使用 shiro 如何完成授权操作？intercetor，aop

4.2. 添加授权配置

在 spring-shiro.xml 中追加如下配置：

```
    <!-- 配置bean对象的生命周期管理 -->
    <bean id="lifecycleBeanPostProcessor"
class="org.apache.shiro.spring.LifecycleBeanPostProcessor">
    </bean>
```

```
    <!-- 配置Bean对象的代理 -->
    <bean
class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyC
reator"
          depends-on="lifecycleBeanPostProcessor">
    </bean>
```

```
    <!-- 配置授权属性-->
    <bean
class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeS
ourceAdvisor">
        <property name="SecurityManager" ref="securityManager"/>
    </bean>
```

4.3. 授权服务端实现

4.3.1. Dao 实现

业务描述：(核心业务是基于用户 id 获取用户对应的权限)

1) 基于用户 id 查找角色 id 信息

2) 基于角色 id 查找菜单 id 信息

3) 基于菜单 id 查找权限标识信息

业务实现：(在 SysUserDao 中基于用户 id 查找角色 id 信息)

1) 返回值 List<Integer>

2) 方法名 findRoleIdsByUserId

3) 参数列表 (Integer id)

业务实现：(在 SysRoleMenuDao 中基于用户 id 查找菜单 id 信息)

1) 返回值 List<Integer>

2) 方法名 findMenuIdsByRoleIds

3) 参数列表 (Integer[] id)

业务实现：(在 SysMenuDao 中基于菜单 id 查找权限标识信息)

1）返回值 List<String>

2）方法名 findPermisssions

3）参数列表（Integer[] id）

代码实现：

SysUserRoleDao 中方法定义

```
    List<Integer> findRoleIdsByUserId(
            Integer id);
```

SysRoleMenuDao 中方法定义

```
    List<Integer> findMenuIdsByRoleIds(
            @Param("roleIds")Integer[] roleIds);
```

SysMenuDao 中方法定义

```
    List<String> findPermissions(
            @Param("menuIds")
            Integer[] menuIds);
```

4.3.2. Mapper 实现

业务描述

基于 Dao 中方法，定义映射元素

代码实现：

SysUserRoleMapper 中元素定义

```
    <select id="findRoleIdsByUserId"
            resultType="int">
          select role_id
          from sys_user_roles
          where user_id=#{userId}
    </select>
```

SysRoleMenuMapper 中元素定义

```
    <select id="findMenuIdsByRoleIds"
         resultType="int">
```

```
        select menu_id
        from sys_role_menus
        where role_id in
        <foreach collection="roleIds"
                 open="("
                 close=")"
                 separator=","
                 item="item">
             #{item}
        </foreach>
    </select>
```

SysMenuMapper 中元素定义

```
    <select id="findPermissions"
            resultType="string">
       select permission <!-- sys:user:update -->
       from sys_menus
       where id in
       <foreach collection="menuIds"
                open="("
                close=")"
                separator=","
                item="item">
           #{item}
       </foreach>
   </select>
```

4.3.3. Service 实现

业务描述

重写对象 realm 的 doGetAuthorizationInfo 方法，并完成用户权限信息的获取以及封装，最后将信息传递给授权管理器完成授权操作。

```
@Service
public class ShiroUserRealm extends AuthorizingRealm {

    @Autowired
    private SysUserDao sysUserDao;
```

```
@Autowired
private SysUserRoleDao sysUserRoleDao;

@Autowired
private SysRoleMenuDao sysRoleMenuDao;

@Autowired
private SysMenuDao sysMenuDao;

//…
/**通过此方法完成授权信息的获取及封装*/
@Override
protected AuthorizationInfo doGetAuthorizationInfo(
    PrincipalCollection principals) {
    //1.获取登录用户信息，例如用户id
    SysUser user=(SysUser)principals.getPrimaryPrincipal();
    Integer userId=user.getId();
    //2.基于用户id获取用户拥有的角色(sys_user_roles)
    List<Integer> roleIds=
    sysUserRoleDao.findRoleIdsByUserId(userId);
    if(roleIds==null||roleIds.size()==0)
    throw new AuthorizationException();
    //3.基于角色id获取菜单id(sys_role_menus)
    Integer[] array={};
    List<Integer> menuIds=
    sysRoleMenuDao.findMenuIdsByRoleIds(
            roleIds.toArray(array));
    if(menuIds==null||menuIds.size()==0)
    throw new AuthorizationException();
    //4.基于菜单id获取权限标识(sys_menus)
    List<String> permissions=
    sysMenuDao.findPermissions(
    menuIds.toArray(array));
    //5.对权限标识信息进行封装并返回
    Set<String> set=new HashSet<>();
    for(String per:permissions){
    if(!StringUtils.isEmpty(per)){
        set.add(per);
    }
    }
    SimpleAuthorizationInfo info=
    new SimpleAuthorizationInfo();
    info.setStringPermissions(set);
    return info;//返回给授权管理器
```

```
    }

}
```

4.4. 授权拦截实现实现

在需要进行授权访问的方法上添加执行此方法需要的权限标识
例如
@RequiredPermissions(“sys:user:valid”)

5.Shiro 应用增强

5.1.1. Shiro 缓存配置（了解）

Step01：添加 ehcache 依赖

```
  <dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-ehcache</artifactId>
   <version>1.3.2</version>
  </dependency>
```

Step02：添加 ehcache 配置文件

在项目的 src/main/resources 目录下添加 ehcache.xml

Step03： Spring 中配置 ehcache 配置文件

```
<bean id="cacheManager"
      class="org.apache.shiro.cache.ehcache.EhCacheManager">
       <property name="cacheManagerConfigFile"
                 value="classpath:ehcache.xml"/>
</bean>
```

将 cacheManager 添加到 securityManager 中

```
    <bean id="securityManager"
          class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="cacheManager" ref="cacheManager"/>
        <property name="realm" ref="userRealm"></property>
    </bean>
```

5.1.2. Shiro 记住我（了解）

课后可自学

6. 总结

6.1. 重点和难点分析

---

1. shiro 认证过程实现
2. shiro 授权过程实现

6.2. 常见 FAQ

---

1. 说说 shiro 的核心组件
2. 说说 shiro 的认证流程
3. 说说 shiro 的授权流程