

国际物流云商系统第四天

一.知识回顾

1.spring data Jpa 强化训练

2.RBAC 认证方式

3.RBAC 认证下的数据库表结构

4.实现用户模块的 CRUD

5.实现角色模块的 CRUD

6.实现模块的 CRUD

Role 的实体类中多对多 modules 配置

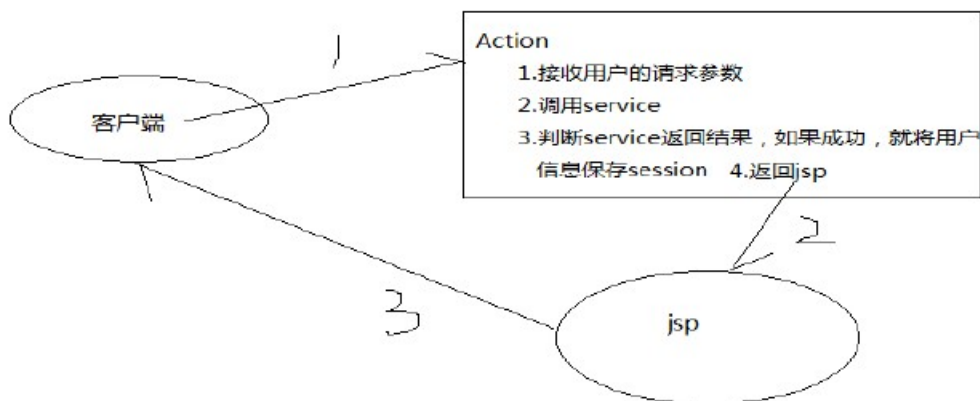
```
@ManyToMany(mappedBy="roles")
@OrderBy("ORDER_NO")
private Set<Module> modules = new HashSet<Module>(0); //角色与模块 多对多
```

Module 的实体类中多对多 roles 配置

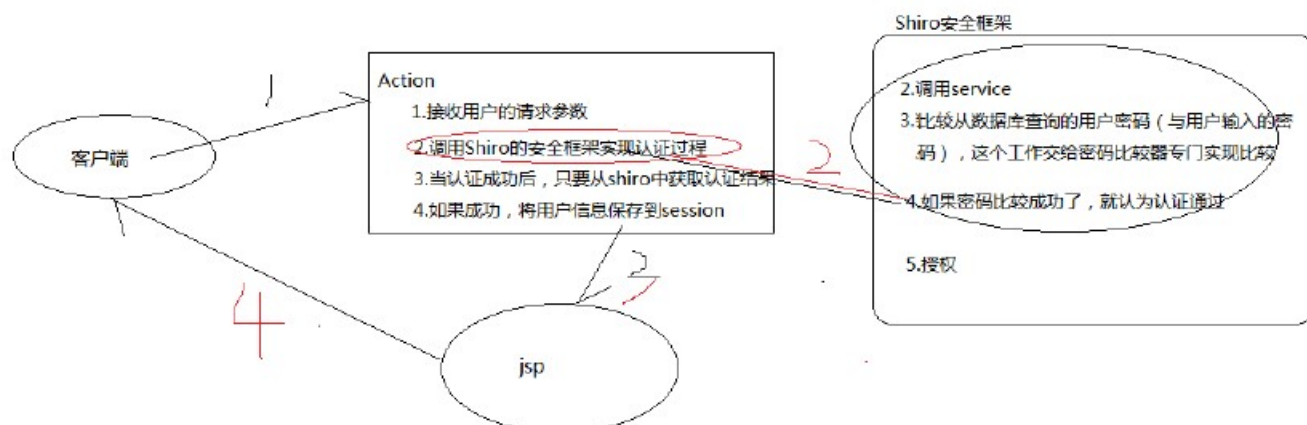
```
@ManyToMany
@JoinTable(name="ROLE_MODULE_P",
    joinColumns={@JoinColumn(name="MODULE_ID",referencedColumnName="MODULE_ID")},
    inverseJoinColumns={@JoinColumn(name="ROLE_ID",referencedColumnName="ROLE_ID")}
)
private Set<Role> roles = new HashSet<Role>(0); //模块与角色 多对多
```

二. 认证与 Shiro 安全框架

1.传统登录方式



2. Shiro 安全框架实现登录

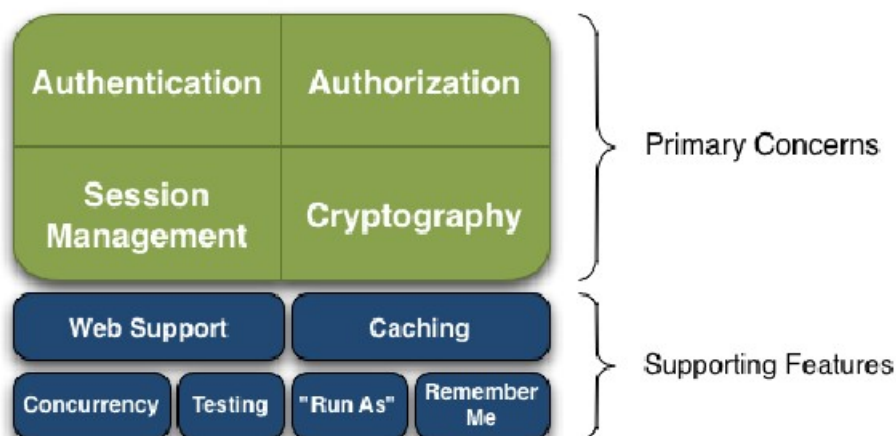


3. 什么是 Shiro?

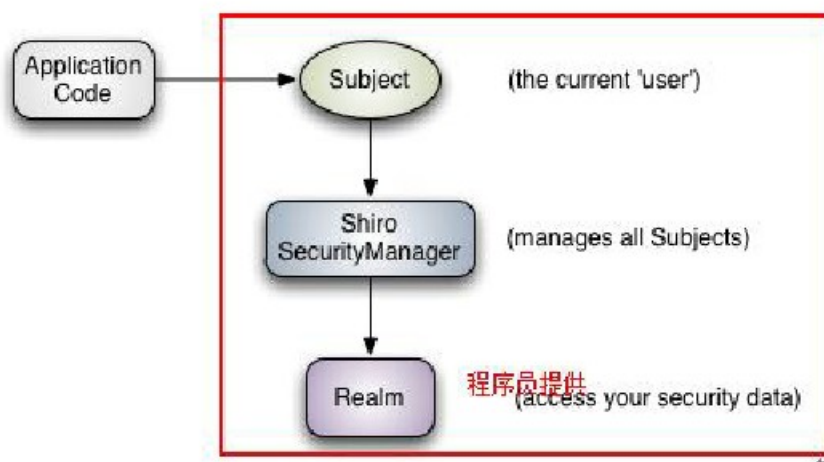
它是一个安全框架，用于解决系统的认证和授权问题，同时提供了会话管理，数据加密机制。

4. Shiro 的内部组织结构?

可以用在 JavaEE 环境。Shiro 可以帮助我们完成：认证、授权、加密、会话管理、与 Web 集成、缓存等。这不就是我们想要的嘛，而且 Shiro 的 API 也是非常简单；其基本功能点如下图所示：



5.应用程序如何使用 Shiro 框架？



可以看出，程序员只关注两部分：

1. 如何获得 **Subject**
2. 如何定义一个符合规定的 **Realm** 域（密码比较器的定义也是程序员干的）

6.Shiro 使用时要配置相关过滤器

过滤器简称	对应的 java 类
anon	org.apache.shiro.web.filter.authc.AnonymousFilter
authc	org.apache.shiro.web.filter.authc.FormAuthenticationFilter
authcBasic	org.apache.shiro.web.filter.authc.BasicHttpAuthenticationFilter
perms	org.apache.shiro.web.filter.authz.PermissionsAuthorizationFilter
port	org.apache.shiro.web.filter.authz.PortFilter
rest	org.apache.shiro.web.filter.authz.HttpMethodPermissionFilter
roles	org.apache.shiro.web.filter.authz.RolesAuthorizationFilter
ssl	org.apache.shiro.web.filter.authz.SslFilter



user	org.apache.shiro.web.filter.authc.UserFilter
logout	org.apache.shiro.web.filter.authc.LogoutFilter

虽然是 10 个过滤器，但使用时只需要配置一个过滤器就可以了

使用时，在 web.xml 文件中配置如下：

核心 filter，一个 filter 相当于 10 个 filter；web.xml

注意：shiro 的 filter 必须在 struts2 的 filter 之前，否则 action 无法创建

<!-- Shiro Security filter filter-name 这个名字的值将来还会在 spring 中用到-->

<filter>

<filter-name>shiroFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

<init-param>

<param-name>targetFilterLifecycle</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>shiroFilter</filter-name>

<url-pattern>/*</url-pattern>

</filter-mapping>

7. 菜单的管理如何实现？



不是每个用户都能看到所有菜单？

可以借助 Shiro 标签来实现

当前用户的权限字符串【“系统首页”、“货运管理”】

```
<!-- shiro:hasPermission 标签 -->
<shiro:hasPermission name="货运管理">
  <span id="topmenu" onclick="toModule('cargo');">货运管理</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="统计分析">
  <span id="topmenu" onclick="toModule('stat');">统计分析</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="基础信息">
  <span id="topmenu" onclick="toModule('baseinfo');">基础信息</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="系统管理">
  <span id="topmenu" onclick="toModule('sysadmin');">系统管理</span>
</shiro:hasPermission> -->
```







三. Shiro 具体使用步骤

1 导入 jar 包

Maven 工程

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-all</artifactId>
    <version>1.2.3</version>
</dependency>
```

Web 工程

 shiro-all-1.2.3.jar	2015/7/18 21:06	360压缩	524 KB
 shiro-core-1.2.3.jar	2015/7/18 21:06	360压缩	359 KB
 shiro-guice-1.2.3.jar	2015/7/18 21:06	360压缩	44 KB

2 过滤器的配置

```
<!--Shiro核心控制器 一定放在struts2过滤器之前-->
<filter>
    <filter-name>shiroFilter</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
        <param-name>targetFilterLifecycle</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>shiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3 产生代理类的方式

下面这一行代码，放在 applicationContext.xml 事务管理器声明之前

```
<aop:aspectj-autoproxy proxy-target-class="true" />

<!-- 产生Shiro控制器的广式，使用cglib生成代理 -->
<aop:aspectj-autoproxy proxy-target-class="true" />
```



4.Shiro 的配置文件

```
<description>Shiro与Spring整合</description>
<bean id="securityManager"
class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
  <!-- Single realm app. If you have multiple realms, use the
'realms' property instead. -->
    <property name="realm" ref="authRealm"/><!-- 引用自定义的
realm -->
  </bean>

  <!-- 自定义Realm域的编写 -->
  <bean id="authRealm" class="cn.itcast.shiro.AuthRealm">
    <!-- 注入自定义的密码比较器 -->
    <property name="credentialsMatcher"
ref="customerCredentialsMatcher" ></property>
  </bean>

  <!-- 自定义的密码比较器 -->
  <bean id="customerCredentialsMatcher"
class="cn.itcast.shiro.CustomerCredentialsMatcher"></bean>

  <!-- filter-name这个名字的值来自于web.xml中filter的名字 -->
  <bean id="shiroFilter"
class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
    <property name="securityManager" ref="securityManager"/>
    <!-- 登录页面 -->
    <property name="loginUrl" value="/index.jsp"></property>

    <!-- 登录成功后 -->
    <!-- <property name="successUrl"
value="/home.action"></property> -->

    <property name="filterChainDefinitions">
      <!-- /**代表下面的多级目录也过滤 -->
      <value>
        /index.jsp* = anon
        /home* = anon
        /sysadmin/login/login.jsp* = anon
        /sysadmin/login/loginAction_logout* = anon
        /login* = anon
        /logout* = anon
        /components/** = anon
      </value>
    </property>
  </bean>
```



```
        /css/** = anon
        /img/** = anon
        /js/** = anon
        /plugins/** = anon
        /images/** = anon
        /js/** = anon
        /make/** = anon
        /skin/** = anon
        /stat/** = anon
        /ufiles/** = anon
        /validator/** = anon
        /resource/** = anon
        /** = authc
        /*.* = authc
    </value>
</property>
</bean>
```

5.在 applicationContext.xml 文件中加载 shiro 配置文件

```
<import resource="classpath:spring/applicationContext-shiro.xml"></import>
```

5 编写密码比较器

1.Md5Hash 算法介绍

```
//高强度加密算法,不可逆
public static String md5(String password, String salt){
    return new Md5Hash(password,salt,2).toString();
}

public static void main(String[] args) {
    System.out.println(new Md5Hash("bb","cgx",3).toString());
}
```

2.密码比较器



```

package cn.itcast.jk.shiro;

import org.apache.shiro.authc.AuthenticationInfo;

/**
 * @description:
 * @author 传智.宋江
 * @date 2015年11月26日
 * @version 1.0
 */
public class CustomCredentialsMatcher extends SimpleCredentialsMatcher {

    public boolean doCredentialsMatch(AuthenticationToken token,
        AuthenticationInfo info) {
        UsernamePasswordToken upToken = (UsernamePasswordToken) token;
        //1.如何得到用户输入的密码（登录界面上输入的密码）
        String cIntInputPwd = Encrypt.md5(upToken.getPassword().toString(), upToken.getUsername()); //将用户输入的明文转化为密文

        //2.得到数据库中的密码
        String dbPwd = (String) info.getCredentials(); //代表用户的认证密码（数据库中的密码）

        //这是父类中的方法，用于比较密码是否一致
        return this.equals(cIntInputPwd, dbPwd);
    }
}

```

3.配置密码比较器

```

<!-- 设置密码加密策略 md5hash -->
<bean id="passwordMatcher" class="cn.itcast.jk.shiro.CustomCredentialsMatcher"/>

```

四. 编写自定义 Realm 域

1.编写自定义的 AuthRealm

自定义的 AuthRealm 需要继承 AuthorizingRealm 类。

```

public class AuthRealm extends AuthorizingRealm {
    @Autowired
    private UserService userService;

    /**
     * 授权
     */
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection pc) {
        //1.从Shiro中取出当前登录的用户信息
        User user = (User) pc.fromRealm(this.getName()).iterator().next();

        List<String> moduleList = new ArrayList<>();
        //2.加载用户的角色信息
        Set<Role> roleSet = user.getRoles();

        //3.遍历每个角色信息
        for (Role role : roleSet) {
            Set<Module> modules = role.getModules();
            for (Module module : modules) {
                moduleList.add(module.getName());
            }
        }

        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        info.addStringPermissions(moduleList);
        return info;
    }
}

```




```
/**
 * 认证
 */
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    //1.用户的信息向下转型
    final UsernamePasswordToken upToken = (UsernamePasswordToken) token;

    //2.调用业务逻辑方法，实现登录
    Specification<User> spec = new Specification<User>() {
        public Predicate toPredicate(Root<User> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            Predicate p1 = cb.equal(root.get("userName").as(String.class), upToken.getUsername());
            return p1;
        }
    };
    List<User> userList = userService.find(spec);

    //3.判断用户的用户名是否可用
    if(userList!=null && userList.size()>0){
        User user = userList.get(0);
        return new SimpleAuthenticationInfo(user,user.getPassword(),this.getName());
    }
    return null;
}
```

2.将编写的 AuthRealm 域配置好

```
<!-- 自定义权限认证 -->
<bean id="authRealm" class="cn.itcast.jk.shiro.AuthRealm">
    <property name="userService" ref="userService"/>
    <!-- 自定义密码加密算法 -->
    <property name="credentialsMatcher" ref="passwordMatcher"/>
</bean>
```

五. 登录操作

1.修改 LoginAction 类 login()方法

```
if(UtilFuns.isEmpty(username)){
    return "login";
}

//使用Shiro实现登录操作

try {
    //1.得到Subject对象
    Subject subject = SecurityUtils.getSubject();
    //2.封装用户数据成一个AuthenticationToken对象
    UsernamePasswordToken uptoken = new
    UsernamePasswordToken(username, password);
    //3.实现登录操作
    subject.login(uptoken); //立即调用AuthRealm域中的认证方法
```



```
//4. 登录成功后，就可以从Shiro中取出用户对象
User user = (User)subject.getPrincipal();
//5. 将用户信息，放入session域中
session.put(SysConstant.CURRENT_USER_INFO, user);
} catch (Exception e) {
    e.printStackTrace();
    //当密码比较失败后，也会抛出异常
    request.put("errorMsg", "对不起，用户名或密码错误，登录失败");
    return "login";
}
return SUCCESS;
}
```

2. 测试认证过程成功，进入后台操作主页面。



3. 测试授权过程

当jsp页面上出现Shiro标签时就会执行AuthRealm域中的授权方法。

1. 引入Shiro标签

```
<%@ taglib uri="http://shiro.apache.org/tags" prefix="shiro" %>
```

2. 使用shiro标签进行授权判断



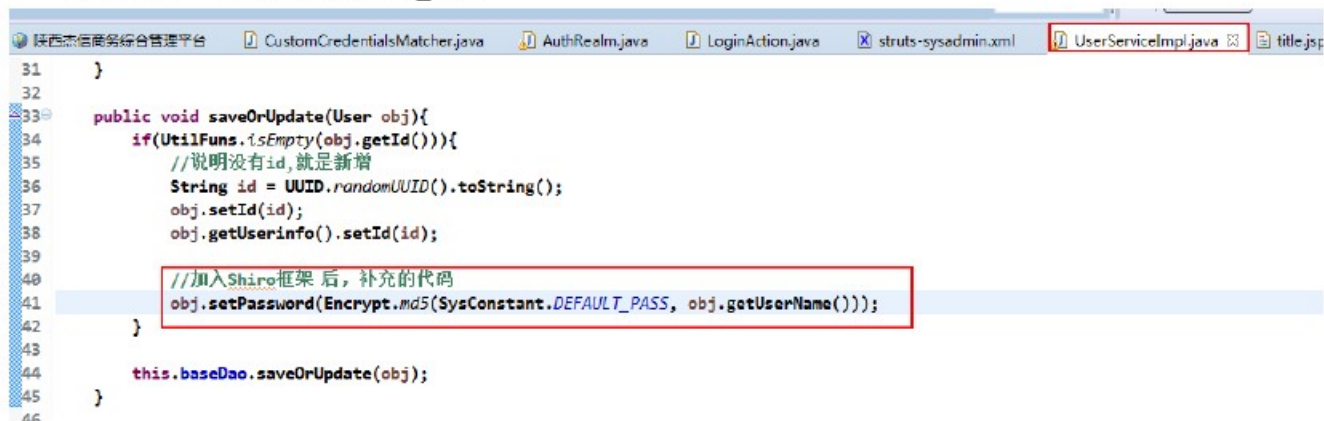
```
<shiro:hasPermission name="系统首页">
<span id="topmenu" onclick="toModule('home');">系统首页</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="系统管理">
    <span id="topmenu" onclick="toModule('cargo');">货运管理</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="统计分析">
<span id="topmenu" onclick="toModule('stat');">统计分析</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="基础信息">
<span id="topmenu" onclick="toModule('baseinfo');">基础信息</span><span id="tm_separator"></span>
</shiro:hasPermission>
<shiro:hasPermission name="系统管理">
<span id="topmenu" onclick="toModule('sysadmin');">系统管理</span>
</shiro:hasPermission>
<shiro:hasPermission name="流程管理">
<span id="topmenu" onclick="toModule('activiti');">流程管理</span>
</shiro:hasPermission>
```

4. 添加用户时的 bug 修正

早期实现添加用户时，没有给密码，所以要补充，这样才可以使用添加的新用户实现登录操作。

SysConstant 是一个常量类，主要配置各种常量的。

密码就是一个常量：DEFAULT_PASS



```
31 }
32
33 public void saveOrUpdate(User obj){
34     if(UtilFuns.isEmpty(obj.getId())){
35         //说明没有id,就是新增
36         String id = UUID.randomUUID().toString();
37         obj.setId(id);
38         obj.getUserInfo().setId(id);
39
40         //加入Shiro框架后,补充的代码
41         obj.setPassword(Encrypt.md5(SysConstant.DEFAULT_PASS, obj.getUserName()));
42     }
43
44     this.baseDao.saveOrUpdate(obj);
45 }
46
```

5. 登出操作

```
//退出
@Action("loginAction_logout")
public String logout(){
    session.remove(SysConstant.CURRENT_USER_INFO); //删除session
    SecurityUtils.getSubject().logout(); //登出
    return "logout";
}
```



六. Shiro 的注解

1.Shiro 配置文件

如果需要加入注解，则需要在 shiro 配置文件中加入下面的配置，这样 shiro 就支持注解开发了。

```
<!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->
<bean id="LifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

<!-- 生成代理，通过代理进行控制 -->
<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
    depends-on="LifecycleBeanPostProcessor">
    <property name="proxyTargetClass" value="true"/>
</bean>

<!-- 安全管理器 -->
<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
    <property name="securityManager" ref="securityManager"/>
</bean>
```

2.注解添加

```
/**
 * 分页查询部门列表
 */
@Action(value = "deptAction_list", results = {
    @Result(name = "list", location = "/WEB-INF/pages/sysadmin/dept/jDeptList.jsp") })
@RequiresPermissions("部门管理")
public String list() throws Exception {
```

测试效果：

错误信息：

Subject does not have permission [部门管理]

[返回](#)

[点击这里查看具体错误消息。](#)

报告以下错误消息给系统管理员,可以更加快速的解决问题；

联系电话：120

七. Shiro 运行流程分析

1. 登录过程

The diagram illustrates the login process flow. It starts with a UI form (用户名, 密码, 登录, 重置). An arrow points from the '登录' button to the `LoginAction` class. Inside `LoginAction`, the `login()` method is shown. An arrow points from `login()` to the `AuthRealm` class, specifically to the `doGetAuthenticationInfo()` method. The `AuthRealm` class contains the logic for authenticating the user, including finding the user by username, comparing passwords, and returning an `AuthenticationInfo` object. The flow is annotated with numbers 1 through 5, corresponding to the steps in the code comments.

```

//SSH传统登录方式
public String login() throws Exception {
    //1. 得到Subject对象
    Subject subject = SecurityUtils.getSubject();
    subject.login(token);

    //2. 当认证成功后, 要将shiro中保存的用户对象取出来
    User user = (User) subject.getPrincipal();

    //3. 将用户对象放入session域中
    session.put(SysConstant.CURRENT_USER_INFO, user);

    //4. 返回true
    return true;
} catch (Exception e) {
    e.printStackTrace();
    request.put("errorMsg", "对不起, 登录失败, 用户名或密码错误!");
    return false;
}
    
```

```

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    //1. 认证
    System.out.println("认证");
    //2. 根据用户名查询数据库, 得到用户对象
    User user = userService.findUserByUsername(upToken.getUsername());

    //3. 向下执行代码, 说明用户名存在 第一个参数: 代表当前用户对象,
    AuthenticationInfo info = new SimpleAuthenticationInfo(user, user.getPassword(), this.getName());
    return info;

    //4. 不为null
    //5. 比较密码的方法
    public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
        //1. 向下转型
        UsernamePasswordToken upToken = (UsernamePasswordToken) token;
        //2. 得到原始密码, 没有的密码,
        String oldPwd = new String(upToken.getPassword());
        //3. 并进行加密
        String newPwd = Encrypt.md5(oldPwd, upToken.getUsername());
        //4. 获取数据库中当前用户的密文
        Object dbPwd = info.getCredentials();
        return equals(newPwd, dbPwd);
    }
    }
    
```

2. 授权的过程

The diagram illustrates the authorization process flow. It starts with a JSP page snippet showing a `shiro:hasPermission` tag. An arrow points from this tag to the `AuthRealm` class, specifically to the `doGetAuthorizationInfo()` method. The `AuthRealm` class contains the logic for authorizing the user, including finding the user by principal, getting roles, and returning an `AuthorizationInfo` object. The flow is annotated with numbers 1 through 2, corresponding to the steps in the code comments.

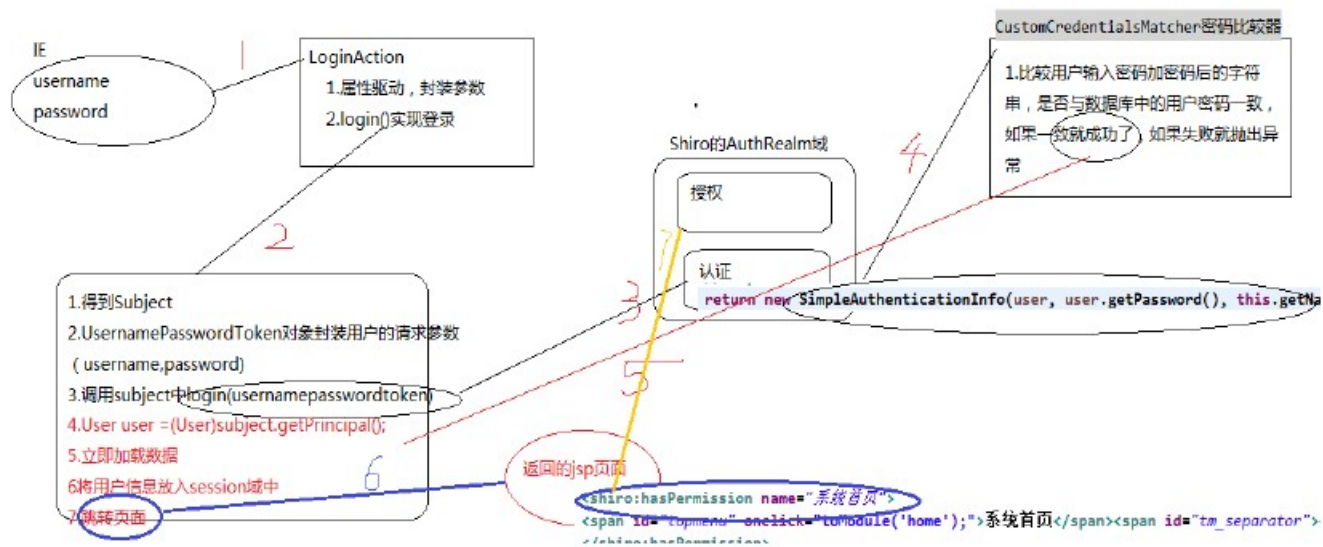
```

title.jsp页面中的shiro标签 <@ taglib uri="http://shiro.apache.org/tags" prefix="shiro" %>
<shiro:hasPermission name="系统首页">
<span id="topmenu" onclick="toModule('home');">系统首页</span><span id="tm_separator"></span>
</shiro:hasPermission>
    
```

```

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    User user = (User) principals.fromRealm(this.getName()).iterator().next();
    Set<Role> roles = user.getRoles();
    List<String> list = new ArrayList<String>(); //用于保存模块的集合
    //3. 遍历出每个角色
    for (Role role : roles) {
        //进一步得到每个角色下的模块列表
        Set<Module> modules = role.getModules();
        //遍历当前角色下的模块列表
        for (Module m : modules) {
            if (m.getCType() == 0) {
                //只加顶部菜单
                list.add(m.getName()); //添加模块名
            }
        }
    }
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
    info.addStringPermissions(list); //添加权限列表, 将来与shiro标签进行比较看是否有权限
    }
    }
    
```

3. 整体分析的图



七. 作业

1. 完成 Shiro 登录
2. 实现三级菜单（要求使用 Shiro 控制查看按钮，新增按钮，修改按钮，删除按钮）
2. 实现用户管理中的角色列表显示（扩展）
3. 实现角色管理中的模块列表显示（扩展）