**shiro在ssm中的使用**

**1、本章面试题**

       shiro在web开发（ssm）的认证和授权的使用

**2、知识点**

**2.1、课程回顾**

                什么shiro

shiro 特点

shiro 核心概念 subject SecurityManager realm

realm 自定义的 realm JdbcRealm

**2.2、本章重点**

         shiro在ssm框架的认证

shiro在ssm框架的授权

**3、具体内容**

**3.1 整合简介**

Apache Shiro是一个强大且易用的Java安全框架,执行身份验证、授权、密码学和会话管理。 Shiro不会去认证用户、维护权限；这些需要我们自己去设计；然后通过相应的接口注入给Shiro即可。

**3.2 整合过程**

**1，创建SSM项目，引入shiro jar**

<shiro.version>1.5.1</shiro.version>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-core</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-ehcache</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>${shiro.version}</version>

</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-web</artifactId>

<version>${shiro.version}</version>

</dependency>

**2，配置shiro过滤器**

<!-- shiro过滤器 开始-->

<filter>

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

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

<init-param>

<!--如果设置"targetFilterLifecycle"为True，则spring来管理Filter.init()和Filter.destroy()；若为false，则这两个方法失效-->

<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>

<!-- shiro过滤器 结束-->

**3，编写shiro配置文件，并在监听器中读取**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<!--shiro 过滤器，名称要和web.xml中一致-->

<bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">

<property name="securityManager" ref="securityManager"/>

<!-- override these for application-specific URLs if you like:-->

<!--登录页面-->

<property name="loginUrl" value="/toLogin.do"/>

<!--successUrl配置只是做为一种附加配置，只有session中没有用户请求地址时才会使用successUrl。系统默认的是认证成功后跳转到上一次请求的路径，如果是首次请求，那shiro就会跳转到默认虚拟路径“/”，也就是跳转到index.jsp。-->

<property name="successUrl" value="/index.jsp"/>

<!--未授权的页面-->

<property name="unauthorizedUrl" value="/unauthorized.jsp"/>

<!-- The 'filters' property is not necessary since any declared javax.servlet.Filter bean -->

<!-- defined will be automatically acquired and available via its beanName in chain -->

<!-- definitions, but you can perform instance overrides or name aliases here if you like: -->

<!-- <property name="filters">

<util:map>

<entry key="anAlias" value-ref="someFilter"/>

</util:map>

</property> -->

<property name="filterChainDefinitions">

<value>

<!-- anon免拦截   authc需要拦截的资源  \*\*代表拦截所有的-->

# some example chain definitions:

/login.jsp=anon

/imgs/\*=anon

/toLogin.do=anon

/userLogin.do=anon

<!--拦截所有放在最下面-->

/\*\* = authc

# more URL-to-FilterChain definitions here

</value>

</property>

</bean>

<!--把shiro的bean交给spring管理-->

<bean id="lifecycleBeanPostProcessor" class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

<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="myRealm"/>

<!-- By default the servlet container sessions will be used. Uncomment this line

to use shiro's native sessions (see the JavaDoc for more): -->

<!-- <property name="sessionMode" value="native"/> -->

</bean>

<!--自定义myRealm-->

<bean id="myRealm" class="com.aaa.ssm.util.MyShiroRealm">

<!--cm放开-->

<property name="credentialsMatcher" ref="credentialsMatcher"></property>

</bean>

<!--加密相关类-->

<bean id="credentialsMatcher" class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">

<!--加密算法-->

<property name="hashAlgorithmName" value="SHA-512"></property>

<!--哈希次数-->

<property name="hashIterations" value="10"></property>

</bean>

</beans>

**4，编写自己的realm(重写认证和授权方法)**

package com.aaa.shiro.util;

import com.aaa.shiro.service.UserService;

import org.apache.shiro.authc.AuthenticationException;

import org.apache.shiro.authc.AuthenticationInfo;

import org.apache.shiro.authc.AuthenticationToken;

import org.apache.shiro.authc.SimpleAuthenticationInfo;

import org.apache.shiro.authz.AuthorizationInfo;

import org.apache.shiro.authz.SimpleAuthorizationInfo;

import org.apache.shiro.realm.AuthorizingRealm;

import org.apache.shiro.subject.PrincipalCollection;

import org.springframework.beans.factory.annotation.Autowired;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

/\*\*

\* fileName:MyShiroRealm

\* description:

\* author:zz

\* createTime:2019/11/12 9:37

\* version:1.0.0

\*/

public class MyShiroRealm extends AuthorizingRealm {

@Autowired

private UserService userService;

@Override

public String getAuthorizationCacheName() {

return super.getAuthorizationCacheName();

}

/\*\*

\* 授权

\* @param principalCollection

\* @return

\*/

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {

//使用方法提供的参数对象获取认证成功后的用户名

r4444444444444444444444444444444444e

//需要根据用户名查询出该用户拥有的所有角色

List<Map> roleMapList = userService.getRoleListByUserName(userName + "");

//根据返回值实例化子类

SimpleAuthorizationInfo simpleAuthorizationInfo = new SimpleAuthorizationInfo();

if(roleMapList!=null&&roleMapList.size()>0){

for (Map map : roleMapList) {

simpleAuthorizationInfo.addRole(map.get("ROLE\_NAME")+"");

}

}

// simpleAuthorizationInfo.A 取出所有权限

return simpleAuthorizationInfo;

}

/\*\*

\* 认证

\* @param authenticationToken

\* @return

\* @throws AuthenticationException

\*/

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {

//获取登录登录方法中收集到的用户名

Object userName = authenticationToken.getPrincipal();

// 根据用户名去查询 用户列表

Map map =new HashMap();

map.put("userName",userName);

List<Map> empMapList = empService.listUserByParam(map);

//如果根据收集到的用户名查不出数据，说明用户名是不对

if(empMapList==null || empMapList.isEmpty()){

throw new AccountException();//如果满足条件，抛出异常，下面代码就不会执行

}

//如果用户名称正确，肯定查询出数据，下面把查询出的信息交给SecurityManger比对密码

//获取当前用户对象

Map userMapInfo = empMapList.get(0);

//不加密不加盐 使用oracle数据库，返回List<map> 获取map中对应的列值时，必须使用大写的列名称

//return new SimpleAuthenticationInfo(userMapInfo,userMapInfo.get("PASSWORD")+"",getName());

//加密加盐时，返回

return new SimpleAuthenticationInfo(userMapInfo,userMapInfo.get("PASSWORD")+"",

ByteSource.Util.bytes(userMapInfo.get("PWSALT")+""),getName());

}

}

**5，编写登录功能**

代码和前面讲课内容重复

**6，测试认证和授权**

1，测试登录功能

2，测试权限标签功能

**4、本章总结**

**4.1 总结本章知识点**

**4.2 面试题答案**

**4.3 预习下一章重点**