**Shiro概述**

**1.什么是Shiro？**

Apache shiro是Java安全框架,执行身份验证、授权、密码和会话管理。

**2.Shiro的主要功能**

Authentication：身份认证 / 登录（确定用户身份）;  
Authorization：授权，即权限验证（对用户访问系统资源的行为做控制）;  
Session Management：会话管理，即用户登录后就是一次会话;  
Cryptography：加密;

更多信息：  
<https://www.w3cschool.cn/shiro/co4m1if2.html>

**3.Shiro是如何工作的？**

Subject：主体，代表了当前 “用户”；

SecurityManager：安全管理器，它管理着所有 Subject，是shiro的核心；

Realm：域，Shiro 从 Realm 获取安全数据（如用户、角色、权限）， Shiro 不提供维护用户 / 权限，而是通过 Realm 让开发人员自己注入；

# Springboot集成Shiro

[Java](http://localhost:9093/itour/work/detail?id=63) [Shiro](http://localhost:9093/itour/work/detail?id=63)

阅读  最后发布于 2022-05-26 19:32

## 1.步基本骤

1）引入依赖  
2）配置Shiro

## 3.代码实现

### 1.引入依赖

<properties>

<shiro-spring-boot-web-starter.version>1.6.0</shiro-spring-boot-web-starter.version>

</properties>

<dependency>

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

<artifactId>shiro-spring-boot-web-starter</artifactId>

<version>${shiro-spring-boot-web-starter.version}</version>

</dependency>

### 2.构建Realm

public class MyRealm extends AuthorizingRealm {

private static final Logger logger = LoggerFactory.getLogger(MyRealm.class);

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// TODO Auto-generated method stub

logger.info("doGetAuthorizationInfo---------------------------->>>");

return null;

}

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// TODO Auto-generated method stub

logger.info("doGetAuthenticationInfo---------------------------->>>");

return null;

}

}

### 3. 配置Shiro

@Configuration

public class ShiroConfig {

//1.Realm 代表系统资源

@Bean

public Realm myRealm() {

MyRealm myRealm = new MyRealm();

return myRealm;

}

//2.SecurityManager 流程控制

@Bean

public DefaultWebSecurityManager mySecurityManager(Realm myRealm) {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

securityManager.setRealm(myRealm);

return securityManager;

}

//3.ShiroFilterFactoryBean 请求过滤

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

return factoryBean;

}

# Shiro的基础认证授权

[Shiro](http://localhost:9093/itour/work/detail?id=65)

阅读  最后发布于 2022-06-20 19:30

## 1.Shiro认证思路

1.获取当前的 Subject，调用SecurityUtils.getSubject()；

2.校验用户是否已经被认证，即是否已经登录，调用Subject 的isAuthenticated()；

3.若没有认证，则根据前端传入的用户名和密码,把用户名和密码封装为UsernamePasswordToken对象；

1）前台创建1个表单；

2）把请求提交springmvc的Handler；

3）在controller中获取用户名和密码；

4.执行登录,调用Subject中的login(AuthenticationToken token)方法；

5.自定义Realm的方法，从数据库中获取对应的记录给Shiro。

1）需要继承org.apache.shiro.realm.AuthorizingRealm类；

2）使用shiro加密密码

a.配置shiro加密方式

b.将加密方式注入Realm

3）实现doGetAuthenticationInfo (AuthenticationToken token)方法；

6.由shiro完成对密码的比对；

## 2.shiroFilter工作原理：

如果请求的url在ShiroFilterFactoryBean中没有配置或者配置了是可以匿名访问的可以直接访问对应的页面，否则会被重定向到loginurl页面。

## 3.如何配置ShiroFilter

3.1配置路径过滤key：是ant路径,支持\*,\*\*,?;value配置shiro的默认过滤器, anon:匿名访问;authc:需要认证(登录)才能访问;

| **key** | **value** | **说明** |
| --- | --- | --- |
| ？ | 匹配一个字符 | /admin?匹配/admin1但是不匹配/admin ,/admin/ |
| \* | 匹配零个或多个字符串 | /admin\*匹配/admin1,/admin,但不匹配/admin/1 |
| \*\* | 匹配路径中的零个或多个路基 | /admin/\*\* 匹配/admin/a,/admin/a/b |

3.2 URL权限采取第一次匹配优先的方式，即从头开始使用第一个匹配的url模式对应的拦截器链；

Map<String,String> filterMap = new HashedMap<String, String>();

filterMap.put("/account/\*\*","anon");

filterMap.put("/account/queryAccout","authc");

filterMap.put("/\*\*","authc");

如果请求的url是“/account/queryAccout”,因为按照声明顺序进行匹配，那么将使用anon进行拦截；

## 4.代码实现

### 1.配置Shiro

@Configuration

public class ShiroConfig {

@Autowired

AccountConnector accountApi;

//1.Realm 代表系统资源

@Bean

public Realm myRealm() {

MyRealm myRealm = new MyRealm();

return myRealm;

}

//2.SecurityManager 流程控制

@Bean

public DefaultWebSecurityManager mySecurityManager(Realm myRealm) {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

securityManager.setRealm(myRealm);

securityManager.setCacheManager(memoryConstrainedCacheManager());

return securityManager;

}

//3.ShiroFilterFactoryBean 请求过滤

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

//设置登录页面,如果不设置会默认去找项目根目录下的login.jsp

factoryBean.setLoginUrl("/login");

//配置路径过滤key：是ant路径,支持\*,\*\*,?;value配置shiro的默认过滤器, anon:匿名访问;authc:需要认证(登录)才能访问

Map<String,String> filterMap = new LinkedHashMap<String, String>();

//放行静态资源

filterMap.put("/css/\*\*","anon");

filterMap.put("/md/\*\*","anon");

filterMap.put("/js/\*\*","anon");

filterMap.put("/img/\*\*","anon");

//从数据库中获取权限

List<Map<String, String>> accountRight = accountApi.getAccountRight();

for (Map<String, String> map : accountRight) {

String url = map.get("url");

String isLogin = map.get("isLongin");

filterMap.put(url, isLogin);

}

filterMap.put("/loginSub", "anon");

//退出

filterMap.put("/shiro/logout", "logout");

filterMap.put("/\*\*", "authc");

factoryBean.setFilterChainDefinitionMap(filterMap);

return factoryBean;

}

//Shiro自带的缓存 (缓存只能用于本机，那么在集群时就无法使)

@Bean

public CacheManager memoryConstrainedCacheManager() {

MemoryConstrainedCacheManager cacheManager = new MemoryConstrainedCacheManager();

return cacheManager;

}

/\*\*

\* 开启Shiro的注解，

\* (如@RequiresRoles,@RequiresPermissions),需借助SpringAOP扫描使用Shiro注解的类,

\* 并在必要时进行安全逻辑验证 \* 配置以下两个bean(DefaultAdvisorAutoProxyCreator(可选)

\* 和AuthorizationAttributeSourceAdvisor)即可实现此功能

\* 与springboot集成时，如果不配置，且使用了相关注解会导致页面404无法访问;

\* @return

\*/

@Bean

public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {

DefaultAdvisorAutoProxyCreator proxyCreator = new DefaultAdvisorAutoProxyCreator();

proxyCreator.setProxyTargetClass(true);

return proxyCreator;

}

/\*\*

\* 开启 shiro aop注解支持.

\*

\* @param securityManager

\* @return

\*/

@Bean

public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(DefaultWebSecurityManager securityManager) {

AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor();

advisor.setSecurityManager(securityManager);

return advisor;

}

}

### 2.编写认证方法

@RequestMapping("/login")

public Map login(@RequestBody JSONObject jsonObject,HttpServletRequest request) {

Map<String,Object> resultMap = new HashMap<String, Object>();

String password = jsonObject.getString("password");

String userName = jsonObject.getString("userName");

try {

Subject subject = SecurityUtils.getSubject();

if(!subject.isAuthenticated()) {

UsernamePasswordToken token = new UsernamePasswordToken(userName, password);

subject.login(token);

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", Constant.SUCESS\_MESSAGE);

}

} catch (UnknownAccountException e) {//用户不存在

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_ACCOUNTINFO);

}catch (IncorrectCredentialsException e) {//用户存在，但密码不匹配

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_ACCOUNTINFO);

}catch (LockedAccountException e) {//用户被锁定

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", ExceptionInfo.EXCEPTION\_STATUS);

}catch (AuthenticationException e) {

// TODO: handle exception

e.printStackTrace();

resultMap.put("code", Constant.FAILED\_CODE);

resultMap.put("msg", Constant.FAILED\_SYSTEM\_ERROR);

}catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

resultMap.put("msg", Constant.FAILED\_SYSTEM\_ERROR);

}

return resultMap;

}

### 3.自定义Realm

认证基本思路

1.把AuthenticationToken转换为ExUsernamePasswordToken

2.从ExUsernamePasswordToken中获取Username

3.调用数据库方法从数据库中查询Username对应的记录

4.若用户不存在则可以抛出UnknownAccountException异常

5.根据用户情况,决定是否抛出其他的AuthenticationException异常

6.根据用户信息来构建AuthenticationInfo并返回，通常使用的是SimpleAuthenticationInfo，并使用盐值加密；

盐值加密

1).如何把一个字符串加密为MD5或SHA-1

/\*\*

\* MD5盐值加密

\* @param credential

\* @param salt

\* @return

\*/

public static String simpleHashMd5(String credential,String salt) {

String result = new SimpleHash("MD5", credential, ByteSource.Util.bytes(salt), 1024).toHex();

return result;

}

/\*\*

\* SHA-1加密

\* @param credential 密码

\* @param salt 盐

\* @return

\*/

public static String simpleHashSHA\_1(String credential,String salt) {

String result = new SimpleHash("SHA-1", credential, ByteSource.Util.bytes(salt), 1024).toHex();

return result;

}

授权思路

1.从PrincipalCollection获取登录用户的信息

2.利用登录的用户信息来获取当前用户的角色或权限(可能需要查询数据库)

3.创建SimpleAuthorizationInfo并设置roles属性和StringPermissions属性

具体实现

public class MyRealm extends AuthorizingRealm {

private static final Logger logger = LoggerFactory.getLogger(MyRealm.class);

@Autowired

AccountConnector accountApi;

//授权

//1.从PrincipalCollection获取登录用户的信息

//2.利用登录的用户信息来获取当前用户的角色或权限(可能需要查询数据库)

//3.创建SimpleAuthorizationInfo并设置roles属性

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

// TODO Auto-generated method stub

logger.info("doGetAuthorizationInfo---------------------------->>>");

//1.从PrincipalCollection获取登录用户的信息

Oauth primaryPrincipal =(Oauth) principals.getPrimaryPrincipal();

Set<String> roles= new HashSet<String>();

Set<String> permissions = new HashSet<String>();

JSONObject jsonObject = new JSONObject();

jsonObject.put("uid", primaryPrincipal.getuId());

List<Role> roles= accountApi.getAccountRoleName(jsonObject, null);

for (Role role : roleList) {

roles.add(role.getRoleName());

}

//2.3.获取当前用户下的权限

List<Rigth> queryAccountRight = accountConnector.getAccountRightDetial(jsonObject, null);

for (Rigth rightDetail : rightDetailList) {

permissions.add(rightDetail.getUrl());

}

//3.创建SimpleAuthorizationInfo

SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();

info.setRoles(roles);

info.setStringPermissions(permissions);

return info;

}

//认证

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {

// TODO Auto-generated method stub

//1.把AuthenticationToken转换为UsernamePasswordToken

UsernamePasswordToken upt = (UsernamePasswordToken)token;

//2.获取前端传入的用户名

String username = upt.getUsername();

//3.从数据库获取对应记录

JSONObject jsonObject = new JSONObject();

jsonObject.put("username", username);

Account account = accountApi.selectOne(jsonObject);

//4.若用户不存在则可以抛出UnknownAccountException异常

if (account == null) {

throw new UnknownAccountException(ExceptionInfo.EXCEPTION\_USRNAME);

}

//5.根据用户情况,决定是否抛出其他的AuthenticationException异常

if ("1".equals(account.getStatus())) {

//如果用户被锁定了抛出LockedAccountException

throw new LockedAccountException(ExceptionInfo.EXCEPTION\_STATUS);

}

//6.使用盐值加密;

//6.1认证的实体信息，可以是username，也可以是数据库表对应的用户的实体对

Object principal = account;

//6.2加密后的密码（从数据库中取的密码）

Object hashedCredentials = account.getCredentials();

//6.3Realm对象的name，调用父类的getName()方法即可

String realmName = getName();

//6.4盐

String salt = username;

ByteSource credentialsSalt = ByteSource.Util.bytes(salt);

7.构建SimpleAuthenticationInfo

SimpleAuthenticationInfo simpleAuthenticationInfo = new SimpleAuthenticationInfo(principal, hashedCredentials, credentialsSalt, realmName);

return simpleAuthenticationInfo;

}

}

**Shiro登出**

[Shiro](http://localhost:9093/itour/work/detail?id=71)

阅读  最后发布于 2022-06-02 17:14

**1.自己实现**

@RequestMapping("/shiro/logout")

public String logout() {

Subject subject = SecurityUtils.getSubject();

subject.logout();

return "/login";

}

**2.使用Shiro过滤器实现**

shiro帮我们实现了退出功能，在ShiroFilter中配置filterMap.put("/shiro/logout", "logout")可以实现退出，跳转登录页面.

@Bean

public ShiroFilterFactoryBean shiroFilterFactoryBean(DefaultWebSecurityManager mySecurityManager) {

ShiroFilterFactoryBean factoryBean = new ShiroFilterFactoryBean();

factoryBean.setSecurityManager(mySecurityManager);

//配置登录页面地址，如果没有配置会去项目根目录下找login.jsp

factoryBean.setLoginUrl("/login");

Map<String,String> filterMap = new HashMap<String, String>();

//放行静态资源

filterMap.put("/css/\*\*","anon");

filterMap.put("/md/\*\*","anon");

filterMap.put("/js/\*\*","anon");

filterMap.put("/img/\*\*","anon");

filterMap.put("/loginSub", "anon");

filterMap.put("/shiro/logout", "logout");

filterMap.put("/\*\*", "authc");

factoryBean.setFilterChainDefinitionMap(filterMap);

return factoryBean;

}