## 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。

## Shiro的主要功能



Authentication：身份认证 / 登录;

Authorization：授权，即权限验证;

Session Management：会话管理，即用户登录后就是一次会话;

Cryptography：加密;

Web Support：Web 支持

Caching：缓存

Concurrency：shiro 支持多线程应用的并发验证

Testing：提供测试支持；

Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

Remember Me：记住我；

更多信息：

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

## shiro是如何完成工作的



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

### 1.添加依赖

<properties>

<shiro-spring.version>1.6.0</shiro-spring.version>

</properties>

<dependency>

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

<artifactId>shiro-spring</artifactId>

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

</dependency>

### 2.配置shiro

#### 1.配置shiroFilter；

**public** **class** ShiroConfig {

/\*\*

\* 1.配置shiro拦截器ShiroFilter

\* **@param** securityManager 安全管理器

\* **@return** ShiroFilterFactoryBean

\*/

@Bean("shiroFilter")

**public** ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {

//1.构建ShiroFilterFactoryBean

ShiroFilterFactoryBean shiroFilterFactoryBean = **new** ShiroFilterFactoryBean();

//2.获取Filters

Map<String, Filter> filters = shiroFilterFactoryBean.getFilters();

//4.设置SecurityManager

shiroFilterFactoryBean.setSecurityManager(securityManager);

//5.设置未授权界面(没有权限);

shiroFilterFactoryBean.setUnauthorizedUrl("/notRole");

//6.设置登录成功后要跳转的链接

shiroFilterFactoryBean.setSuccessUrl("/index");

//7.设置登录页面,如果不设置默认会自动寻找Web工程根目录下的"/login.jsp"页面

shiroFilterFactoryBean.setLoginUrl("/member/login");

/\*8. authc:url必须认证(登录)通过才可以访问; anon:url可以匿名访问 (无需认证)\*/

Map<String, String> filterChainDefinitionMap = **new** LinkedHashMap<String, String>();

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

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

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

filterChainDefinitionMap.put("/easyui/\*\*","anon");

filterChainDefinitionMap.put("/member/login", "anon");

filterChainDefinitionMap.put("/member/loginSub", "anon");

filterChainDefinitionMap.put("/verify/\*\*", "anon");

//8.0实际开发中,需要从数据库获取需要登录url,设置的到filterChainDefinitionMap中;

//8.1配置退出 过滤器,其中的具体的退出代码Shiro已经替我们实现了

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

/\*\*

\* 8.2 put("/\*\*", "authc");必须放在所有权限设置的最后，不然会导致所有 url 都被拦截 剩余的都需要认证 ;

\* 不能访问的情况下shiro会自动跳转到setLoginUrl()的页面;

\*/

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

shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);

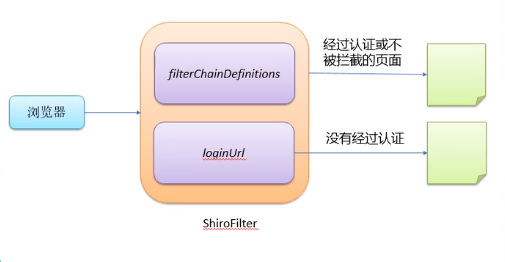
**return** shiroFilterFactoryBean;

}

}

#### 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.URL配置细节

1.URL模式使用Ant风格模式，支持？、\*、\*\*,但需要注意的是通配符匹配不包括

目录分隔符”/”;

|  |  |  |
| --- | --- | --- |
| ? | 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,单不匹配/admin/1 |
| \*\* | 匹配路径中的零个或多个路基 | /admin/\*\* 匹配/admin/a,/admin/a/b |

2.URL的匹配顺序

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；

如：

-/bb/\*\* =filter1

-/bb/aa =filter2

-/\*\* =fiter3

- 如果请求的url是“/bb/aa”,因为按照声明顺序进行匹配，那么将使用filter1进行拦截；

## 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；
4. 前台创建1个表单；
5. 把请求提交springmvc的Handler；
6. 在controller中获取用户名和密码；
7. 执行登录,调用Subject中的login(AuthenticationToken token)方法；
8. 自定义Realm的方法，从数据库中获取对应的记录给Shiro。
9. 需要继承org.apache.shiro.realm.AuthorizingRealm类；
10. 实现doGetAuthenticationInfo (AuthenticationToken token)方法；
11. 由shiro完成对密码的比对；

## Shiro认证代码实现

### 1.Contoller部分

/\*\*

\* 登陆提交

\* **@param** jsonObject

\* **@param** request

\* **@return**

\*/

@RequestMapping("/login")

@ResponseBody

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

**try** {

String username = jsonObject.getString("usrName");

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

//获取当前的 Subject

Subject currentUser = SecurityUtils.*getSubject*();

**if** (!currentUser.isAuthenticated()) {//当前用户是否已经被认证，即是否登录

UsernamePasswordToken token = **new** UsernamePasswordToken(username, password) ;

**try** {

//执行登录

currentUser.login(token);

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

// **TODO**: handle exception

e.printStackTrace();

**return** ResponseMessage.*getFailed*(ExceptionInfo.***EXCEPTION\_ACCOUNTINFO***);

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

// **TODO**: handle exception

e.printStackTrace();

**return** ResponseMessage.*getFailed*(ExceptionInfo.***EXCEPTION\_ACCOUNTINFO***);

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

// **TODO**: handle exception

e.printStackTrace();

**return** ResponseMessage.*getFailed*(ExceptionInfo.***EXCEPTION\_STATUS***);

}**catch** (AuthenticationException e) {

// **TODO**: handle exception

e.printStackTrace();

**return** ResponseMessage.*getFailed*(Constant.***FAILED\_SYSTEM\_ERROR***);

}

}

} **catch** (Exception e) {

// **TODO**: handle exception

e.printStackTrace();

**return** ResponseMessage.*getFailed*(Constant.***FAILED\_SYSTEM\_ERROR***);

}

**return** ResponseMessage.*getSucess*();

}

### 2. Realm部分实现

#### 基本步骤：

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

7.盐值使用MD5盐值加密：

1.如何把一个字符串加密为MD5

2.shiro通过AuthenticatingRealm的credentialsMatcher属性来进行的密码比对

#### 实现：

**public** **class** LoginRealm **extends** AuthorizingRealm {

@Override

**protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

}

@Override

**protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {

//1.把AuthenticationToken转换为ExUsernamePasswordToken

ExUsernamePasswordToken upt = (ExUsernamePasswordToken)token;

//2.从ExUsernamePasswordToken中获取Username(前端输入的用户名)

String username = upt.getUsername();

//3.调用数据库方法从校验用户名和密码，若用户不存在则可以抛出UnknownAccountException异常

JSONObject jsonObject = **new** JSONObject();

jsonObject.put("regName", username);

Account account = accountConnector.loginSub(jsonObject,upt.getRequest());

**if** (account == **null**) {

**throw** **new** UnknownAccountException("用户不存在");

}

//4.根据用户情况,决定是否需要抛出其他的AuthenticationException异常;

**if**("1".equals(account.getLock())) {

**throw** **new** LockedAccountException("用户被锁定")

}

//5.根据用户信息来构建AuthenticationInfo并返回，通常使用的是SimpleAuthenticationInfo

//以下信息是从数据库中获取的

//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;

//credentials密码：加密后的密码（从数据库中取的密码）

Object credentials = account.getCredential();

//Realm对象的name，调用父类的getName()方法即可

String realmName= getName();

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

SimpleAuthenticationInfo simpleAuthenticationInfo = **new** SimpleAuthenticationInfo(principal, credentials, credentialsSalt);

**return** simpleAuthenticationInfo;

}

}