## Shiro.ini文件说明

|  |
| --- |
| 1、操作内置对象时,在[main]里面写东西  [main]  securityManager.属性=值  myobj=com.bjsxt.lei <bean id=myobj class=” com.bjsxt.lei”>  securityManager.对象属性=$myobj <bean id=” securityManager” class=””>  <property name=” 对象属性” ref=” myobj”>  2、[users] :定义用户名和密码  [users]  # 定义用户名为zhangsan 密码为zs  zhangsan=zs  # 定义用户名lisi密码为lisi同时具有role1和role2两个角色  lisi=lisi,role1,role2  3、[roles]: 定义角色  [roles]  role1=权限名1,权限名2  role2=权限3,权限4  [roles]  role1=user:query,user:add,user:update,user:delete,user:export  4、[urls] : 定义哪些内置urls生效.在web应用时使用.  [urls]  #url地址=内置filter或自定义filter  # 访问时出现/login的url必须去认证.支持authc对应的Filter  /login=authc  # 任意的url都不需要进行认证等功能.  /\*\* = anon  # 所有的内容都必须保证用户已经登录.  /\*\*=user  # url abc 访问时必须保证用户具有role1和role2角色.  /abc=roles[“role1,role2”]  authc 代表必须认证之后才能访问的路径  anon 任意的url都不需要进行认证等功能  user 所有的内容都必须保证用户已经登录.  logout 注销 |

## 实现认证和授权

|  |
| --- |
| // 1，创建安全管理器的工厂对象 org.apache.shiro.mgt.SecurityManager; 不能使用java.lang.SecurityManager  Factory<SecurityManager> factory = **new** ~~IniSecurityManagerFactory~~("classpath:shiro.ini");  // 2,使用工厂创建安全管理器  SecurityManager securityManager = factory.getInstance();  // 3,把当前的安全管理器绑定当到线的线程  SecurityUtils.*setSecurityManager*(securityManager);  // 4,使用SecurityUtils.getSubject得到主体对象  Subject subject = SecurityUtils.*getSubject*();  // 5，封装用户名和密码  AuthenticationToken token = **new** UsernamePasswordToken(username, password);  // 6,得到认证  **try** {  subject.login(token);  System.***out***.println("认证通过");  } **catch** (AuthenticationException e) {  System.***out***.println("用户名或密码不正确");  }  /\*} catch (IncorrectCredentialsException e) {  System.out.println("密码不正确");  } catch (UnknownAccountException e) {  System.out.println("用户名不存在");  }\*/  相关方法  subject.hasRole(“”); 判断是否有角色  subject.hashRoles(List);分别判断用户是否具有List中每个内容  subject.hasAllRoles(Collection);返回boolean,要求参数中所有角色用户都需要具有.  subject.isPermitted(“”);判断是否具有权限.  //判断用户是否认证通过  **boolean** authenticated = subject.isAuthenticated();  System.***out***.println("是否认证通过:"+authenticated);  //角色判断  **boolean** hasRole1 = subject.hasRole("role1");  System.***out***.println("是否有role1的角色:"+hasRole1);  //分别判断集合里面的角色 返回数组  List<String> roleIdentifiers=Arrays.*asList*("role1","role2","role3");  **boolean**[] hasRoles = subject.hasRoles(roleIdentifiers);  **for** (**boolean** b : hasRoles) {  System.***out***.println(b);  }  //判断当前用户是否有roleIdentifiers集合里面的所有角色  **boolean** hasAllRoles = subject.hasAllRoles(roleIdentifiers);  System.***out***.println(hasAllRoles);    //权限判断  **boolean** permitted = subject.isPermitted("user:query");  System.***out***.println("判断当前用户是否有user:query的权限 "+permitted);  **boolean**[] permitted2 = subject.isPermitted("user:query","user:add","user:export");  **for** (**boolean** b : permitted2) {  System.***out***.println(b);  }  **boolean** permittedAll = subject.isPermittedAll("user:query","user:add","user:export");  System.***out***.println(permittedAll); |

## 自定义Realm

|  |
| --- |
| **public** **class** UserRealm **extends** AuthorizingRealm {  **private** UserService userService=**new** UserServiceImpl();  **private** RoleService roleService =**new** RoleServiceImpl();  **private** PermissionService permissionService=**new** PermissionServiceImpl();  /\*\*  \* 做认证  \*/  @Override  **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {  String username=token.getPrincipal().toString();  token.getCredentials();  System.***out***.println(username);  /\*\*  \* 以前登陆的逻辑是 把用户和密码全部发到数据库 去匹配  \* 在shrio里面是先根据用户名把用户对象查询出来，再来做密码匹配  \*/  User user=userService.queryUserByUserName(username);  **if**(**null**!=user) {  List<String> roles=roleService.queryRoleByUserName(user.getUsername());  List<String> permissions=permissionService.queryPermissionByUserName(user.getUsername());  ActiverUser activerUser=**new** ActiverUser(user, roles, permissions);  /\*\*  \* 参数说明  \* 参数1:可以传到任意对象  \* 参数2:从数据库里面查询出来的密码  \* 参数3:当前类名  \*/  SimpleAuthenticationInfo info=**new** SimpleAuthenticationInfo(activerUser, user.getPwd(), **this**.getName());  **return** info;  }**else** {  //用户不存在 shiro会抛 UnknowAccountException  **return** **null**;  }  }  /\*\*  \* 作授权  \* 参数说明  \*/  @Override  **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {  ActiverUser activerUser = (ActiverUser) principals.getPrimaryPrincipal();  SimpleAuthorizationInfo info=**new** SimpleAuthorizationInfo();  //添加角色  Collection<String> roles=activerUser.getRoles();  **if**(**null**!=roles&&roles.size()>0) {  info.addRoles(roles);  }  Collection<String> permissions=activerUser.getPermissions();  //添加权限  **if**(**null**!=permissions&&permissions.size()>0) {  info.addStringPermissions(permissions);  }  // if(activerUser.getUser().getType()==0) {  // info.addStringPermission("\*:\*");  // }  **return** info;  }  } |

## 散列算法+凭证配置

|  |
| --- |
| 在Realm里面注入凭证匹配器*credentialsMatcher*，指定加密算法*hashAlgorithmName*和散列次数*hashIterations* |