Shiro技术说明文档

目录

[Shiro技术说明文档 1](#_Toc512189552)

[1 配置过程 Spring 和springmvc 1](#_Toc512189553)

[Spring配置 1](#_Toc512189554)

[spring-servlet.xml配置 2](#_Toc512189555)

[关于control层 3](#_Toc512189556)

[2 Shiro 配置 4](#_Toc512189557)

[Shiro 的认证过程 7](#_Toc512189558)

[认证过程 7](#_Toc512189559)

[加密方式的实现 7](#_Toc512189560)

[授权 处理 （配置） 9](#_Toc512189561)

[Shiro 支持三中授权方式 10](#_Toc512189562)

[授权的realm 如何实现 10](#_Toc512189563)

[Shiro 标签 （用的不是很多） 11](#_Toc512189564)

[权限注解 13](#_Toc512189565)

[Shiro的会话管理 14](#_Toc512189566)

[1，会话相关API 14](#_Toc512189567)

[2，会话监听器(SessionListener接口) 14](#_Toc512189568)

[3 SessionDao 15](#_Toc512189569)

[Remember me 的实现 15](#_Toc512189570)

# 1 配置过程 Spring 和springmvc

Springmvc 的一些简单的配置 和入门使用

一般需要支持包 在spring的required 都需要

Spring 2.5.6：spring.jar、spring-webmvc.jar、commons-logging.jar、cglib-nodep-2.1\_3.jar

SpringMVC是一个基于DispatcherServlet的MVC框架，每一个请求最先访问的都是DispatcherServlet，DispatcherServlet负责转发每一个Request请求给相应的Handler，Handler处理以后再返回相应的视图(View)和模型(Model)，返回的视图和模型都可以不指定，即可以只返回Model或只返回View或都不返回。

DispatcherServlet是继承自HttpServlet的，既然SpringMVC是基于DispatcherServlet的，那么我们先来配置一下DispatcherServlet，好让它能够管理我们希望它管理的内容。HttpServlet是在web.xml文件中声明的

在web.xml 的配置

## Spring配置

<!-- ====================================== -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 可以自定义servlet.xml配置文件的位置和名称，默认为WEB-INF目录下，名称为[<servlet-name>]-servlet.xml，如spring-servlet.xml

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring-servlet.xml</param-value>&nbsp; 默认

</init-param>

-->

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<!-- Spring配置 -->

<!-- ====================================== -->

<listener>

<listenerclass>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<!-- 指定Spring Bean的配置文件所在目录。默认配置在WEB-INF目录下 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:config/applicationContext.xml</param-value>

</context-param>

## spring-servlet.xml配置

　　spring-servlet这个名字是因为上面web.xml中<servlet-name>标签配的值为spring（<servlet-name>spring</servlet-name>），再加上“-servlet”后缀而形成的spring-servlet.xml文件名，如果改为springMVC，对应的文件名则为springMVC-servlet.xml

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

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

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

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

http://www.springframework.org/schema/context <a href="http://www.springframework.org/schema/context/spring-context-3.0.xsd">http://www.springframework.org/schema/context/spring-context-3.0.xsd</a>">

<!-- 启用spring mvc 注解 -->

<context:annotation-config />

<!-- 设置使用注解的类所在的jar包 -->

<context:component-scan base-package="controller"></context:component-scan>

<!-- 完成请求和注解POJO的映射 -->

<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

<!-- 对转向页面的路径解析。prefix：前缀， suffix：后缀 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" p:prefix="/jsp/" p:suffix=".jsp" />

</beans>

关于视图解析 和跳转的处理

DispatcherServlet会利用一些特殊的bean来处理Request请求和生成相应的视图返回。

关于视图的返回，Controller只负责传回来一个值，然后到底返回的是什么视图，是由视图解析器控制的，在jsp中常用的视图解析器是InternalResourceViewResovler，它会要求一个前缀和一个后缀

在上述视图解析器中，如果Controller返回的是blog/index，那么通过视图解析器解析之后的视图就是/jsp/blog/index.jsp

## 关于control层

Springmvc 的详细教程 在 <http://elim.iteye.com/blog/1753271>

一个类的被标记@control 都是control层

package controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import entity.User;

@Controller //类似Struts的Action

public class TestController {

@RequestMapping("test/login.do") // 请求url地址映射，类似Struts的action-mapping

public String testLogin(@RequestParam(value="username")String username, String password, HttpServletRequest request) {

// @RequestParam是指请求url地址映射中必须含有的参数(除非属性required=false)

// @RequestParam可简写为：@RequestParam("username")

if (!"admin".equals(username) || !"admin".equals(password)) {

return "loginError"; // 跳转页面路径（默认为转发），该路径不需要包含spring-servlet配置文件中配置的前缀和后缀

}

return "loginSuccess";

}

@RequestMapping("/test/login2.do")

public ModelAndView testLogin2(String username, String password, int age){

// request和response不必非要出现在方法中，如果用不上的话可以去掉

// 参数的名称是与页面控件的name相匹配，参数类型会自动被转换

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return new ModelAndView("loginError"); // 手动实例化ModelAndView完成跳转页面（转发），效果等同于上面的方法返回字符串

}

return new ModelAndView(new RedirectView("../index.jsp")); // 采用重定向方式跳转页面

// 重定向还有一种简单写法

// return new ModelAndView("redirect:../index.jsp");

}

@RequestMapping("/test/login3.do")

public ModelAndView testLogin3(User user) {

// 同样支持参数为表单对象，类似于Struts的ActionForm，User不需要任何配置，直接写即可

String username = user.getUsername();

String password = user.getPassword();

int age = user.getAge();

if (!"admin".equals(username) || !"admin".equals(password) || age < 5) {

return new ModelAndView("loginError");

}

return new ModelAndView("loginSuccess");

}

@Resource(name = "loginService") // 获取applicationContext.xml中bean的id为loginService的，并注入

private LoginService loginService; //等价于spring传统注入方式写get和set方法，这样的好处是简洁工整，省去了不必要得代码

@RequestMapping("/test/login4.do")

public String testLogin4(User user) {

if (loginService.login(user) == false) {

return "loginError";

}

return "loginSuccess";

}

}

# 2 Shiro 配置

首先是在web.xml 中

<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>

然后在applicationContext.xml 文件

配置cachemanager

<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">

<!-- Set a net.sf.ehcache.CacheManager instance here if you already have one. If not, a new one

will be creaed with a default config:

<property name="cacheManager" ref="ehCacheManager"/> -->

<!-If you don't have a pre-built net.sf.ehcache.CacheManager instance to inject, but you want

a specific Ehcache configuration to be used, specify that here. If you don't, a default

will be used.:

<property name="cacheManagerConfigFile" value="classpath:some/path/to/ehcache.xml"/> -->

</bean>

Eacahe jar包在hibernate中可以找到 然后配置一个eacahe.xml 文件

配置realm （实现了realm的接口）

<bean id="jdbcRealm" class="org.apache.shiro.samples.spring.realm.SaltAwareJdbcRealm">

<property name="name" value="jdbcRealm"/>

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

<property name="credentialsMatcher">

<!-- The 'bootstrapDataPopulator' Sha256 hashes the password

(using the username as the salt) then base64 encodes it: -->

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

<property name="hashAlgorithmName" value="SHA-256"/>

<!-- true means hex encoded, false means base64 encoded -->

<property name="storedCredentialsHexEncoded" value="false"/>

</bean>

</property>

</bean>

配置lifecycleBeanPostProcessor

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

必须在开启spring 的注解过程中才可以使用

<bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"

depends-on="lifecycleBeanPostProcessor"/>

<bean class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">

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

</bean>

<bean id="secureRemoteInvocationExecutor" class="org.apache.shiro.spring.remoting.SecureRemoteInvocationExecutor">

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

</bean>

配置shiroFilter (id必须和配置文件中的)

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

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

<property name="loginUrl" value="/s/login"/>

<property name="successUrl" value="/s/index"/>

<property name="unauthorizedUrl" value="/s/unauthorized"/>

资源是否可以被匿名访问

Authc 必须是在认证之后才可以访问

<property name="filterChainDefinitions">

<value>

/favicon.ico = anon

/logo.png = anon

/shiro.css = anon

/s/login = anon

# allow WebStart to pull the jars for the swing app:

/\*.jar = anon

# everything else requires authentication:

/\*\* = authc

</value>

</property>

</bean>

Realm

LifecycleBeanPost

shiroFilter （和web.xml一致）

# Shiro 的认证过程

Url拦截 有优先级，先匹配的可以 覆盖后面的配置

（outline 查看大纲）

## 认证过程

使用realm获取安全数据

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

2 测试当前的用户是否已经被认证， 调用subject的isAutjenticated()

3 若没有认证，则把用户密码和账号封装为 UsernamePasswordToken 对象

1. 创建一个表单页
2. 把请求提交到Springmvc的handle 中
3. 获取用户账号和密码

4执行登录：调用subject的login(AuthenticationToken)

5 自定义realm方法。从数据库中获取对应的记录 返回给shiro

6 由shiro完成对密码的对比

## 加密方式的实现

进行对realm的额配置 （applicationContext.xml）

<bean id ="shiroRealm" class="com.shiro.test.shirorealms.shiroRealm">

<property name="credentialsMatcher">

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

<property name="hashAlgorithmName" value="MD5"></property>

<property name="hashIterations" value="1024"></property>

</bean>

</property>

</bean>

为什么使用盐值加密 （）

怎样设置盐值

1 在doGetAuthenticationInfo 方法中

创建 SimpleAuthenticationInfo 对象

并返回return info = new SimpleAuthenticationInfo(principal, credentials, credetialSalt, realmName);

2 使用 ByteSource.Util.bytes(); 计算盐值

3 盐值需要唯一，一般使用随机字符或者userid

使用new SimpleHash(hashAlgorithName, credentials, salt, hashIterations); 来计算盐值加密后的值

在实际的加密 的过程中 可能是多个数据库的存储 就是多realm

首先是编写不同的realm 然后在进行配置中添加

<!-- 配置一个认证器 -->

<bean id="authenticator"

class="org.apache.shiro.authc.pam.ModularRealmAuthenticator">

<property name="realms">

<list>

<ref bean="shiroRealm"/>

<ref bean="secondRealm"/>

</list>

</property>

</bean>

<bean id ="shiroRealm" class="com.shiro.test.shirorealms.shiroRealm">

<property name="credentialsMatcher">

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

<property name="hashAlgorithmName" value="MD5"></property>

<property name="hashIterations" value="1024"></property>

</bean>

</property>

</bean>

<bean id ="secondRealm" class="com.shiro.test.shirorealms.secondRealm">

<property name="credentialsMatcher">

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

<property name="hashAlgorithmName" value="SHA1"></property>

<property name="hashIterations" value="1024"></property>

</bean>

</property>

</bean>

两中认证方式交给 认证器 再把认证器加到安全管理器中

bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

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

<!-- Single realm app. If you have multiple realms, use the 'realms' property instead. -->

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

<property name="authenticator" value="authenticator"></property>

</bean>

认证策略 就是多个realm存在时 只有一个认证成功就可以了但是是可以修改的

变成 所有认证 才可以成功使用 （在配置文件中修改）

<!-- 配置一个认证器 -->

<bean id="authenticator"

class="org.apache.shiro.authc.pam.ModularRealmAuthenticator">

<property name="realms">

<list>

<ref bean="shiroRealm"/>

<ref bean="secondRealm"/>

</list>

</property>

<property name="authenticationStrategy">

<bean class="org.apache.shiro.authc.pam.AllSuccessfulStrategy"></bean>

</property>

</bean>

# 授权 处理 （配置）

首先是最好把多realm属性添加到 安全管理器 这是没有问题的

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

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

<!-- Single realm app. If you have multiple realms, use the 'realms' property instead. -->

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

<property name="authenticator" value="authenticator"></property>

<property name="realms">

<list>

<ref bean="shiroRealm"/>

<ref bean="secondRealm"/>

</list>

</property>

</bean>

而且更便于使用 授权的时候需要使用securityManage 的realms 属性

把认证管理器的所有认证变成atleast

<!-- 配置一个认证器 -->

<bean id="authenticator"

class="org.apache.shiro.authc.pam.ModularRealmAuthenticator">

<property name="authenticationStrategy">

<bean class="org.apache.shiro.authc.pam.AtLeastOneSuccessfulStrategy"></bean>

</property>

</bean>

## Shiro 支持三中授权方式

编程式 在if else 使用完成

// 有权限

If(subject.hasRole(“admin”){

}) else{

// 没有权限

}

注解式 在执行的java方法上 放置相应的注解完成，没有权限的抛出异常

@RequestRole（"admin”）

Public void hello(){

}

Jsp/gsp标签 页面完成

1 默认的拦截器 （过滤器）

授权相关的 roles, perms,ssl 等 shiro 的permissions 的

角色的权限配置

/user.jsp = roles[user]

/admin.jsp = roles[admin]

## 授权的realm 如何实现

需要继承 AuthorizingRealm 并实现doGetAuthorizationInfo 的方法

AuthorizingRealm 继承AuthenticatingRealm （自定义realm的实现）

public class testRealm extends AuthorizingRealm {

// 实现授权的方法

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {

return null;

}

// 用户认证的方法

@Override

protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken arg0) throws AuthenticationException {

return null;

}

}

多realm授权的话一个 就可以了 具体的实现方法

// 授权的时候实现的方法 授权会被shiro回调方法

@Override

protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {

//1 从PrincipalCollection 中 获取登录用户的信息

Object principal = principals.getPrimaryPrincipal();

//2 利用登录的用信息 来用户当前用户的角色和权限 （可能需要查询数据库）

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

roles.add("user");

if("admin".equals(principal)){

roles.add("admin");

}

//3创建SimpleAuthenticationInfo 并设置roles属性

SimpleAuthorizationInfo info= new SimpleAuthorizationInfo(roles);

// 返回 SimpleAuthorizationInfo 对象

return info;

}

# Shiro 标签 （用的不是很多）

头部需要导入 <%@ taglib prefix="shiro" uri="http://shiro.apache.org/tags" %>

一个简单的例子

Weclome <shiro:principal></shiro:principal>

<shiro:hasRole name ="admin">

<a href="admin.jsp"> Admin.jsp</a>

</shiro:hasRole>

<shiro:hasRole name ="user">

<a href="user"> User.jsp</a>

</shiro:hasRole>

下面是一些简单的例子

<shiro:guest>

游客访问 <a href = "login.jsp"></a>

</shiro:guest>

user 标签：用户已经通过认证\记住我 登录后显示响应的内容

<shiro:user>

欢迎[<shiro:principal/>]登录 <a href = "logout">退出</a>

</shiro:user>

authenticated标签：用户身份验证通过，即 Subjec.login 登录成功 不是记住我登录的

<shiro:authenticted>

用户[<shiro:principal/>] 已身份验证通过

</shiro:authenticted>

notAuthenticated标签：用户未进行身份验证，即没有调用Subject.login进行登录,包括"记住我"也属于未进行身份验证

<shiro:notAuthenticated>

未身份验证(包括"记住我")

</shiro:notAuthenticated>

principal 标签：显示用户身份信息，默认调用

Subjec.getPrincipal()获取，即Primary Principal

<shiro:principal property = "username"/>

hasRole标签：如果当前Subject有角色将显示body体内的内容

<shiro:hashRole name = "admin">

用户[<shiro:principal/>]拥有角色admin

</shiro:hashRole>

hasAnyRoles标签：如果Subject有任意一个角色(或的关系)将显示body体里的内容

<shiro:hasAnyRoles name = "admin,user">

用户[<shiro:pricipal/>]拥有角色admin 或者 user

</shiro:hasAnyRoles>

lacksRole:如果当前 Subjec没有角色将显示body体内的内容

<shiro:lacksRole name = "admin">

用户[<shiro:pricipal/>]没有角色admin

</shiro:lacksRole>

hashPermission:如果当前Subject有权限将显示body体内容

<shiro:hashPermission name = "user:create">

用户[<shiro:pricipal/>] 拥有权限user:create

</shiro:hashPermission>

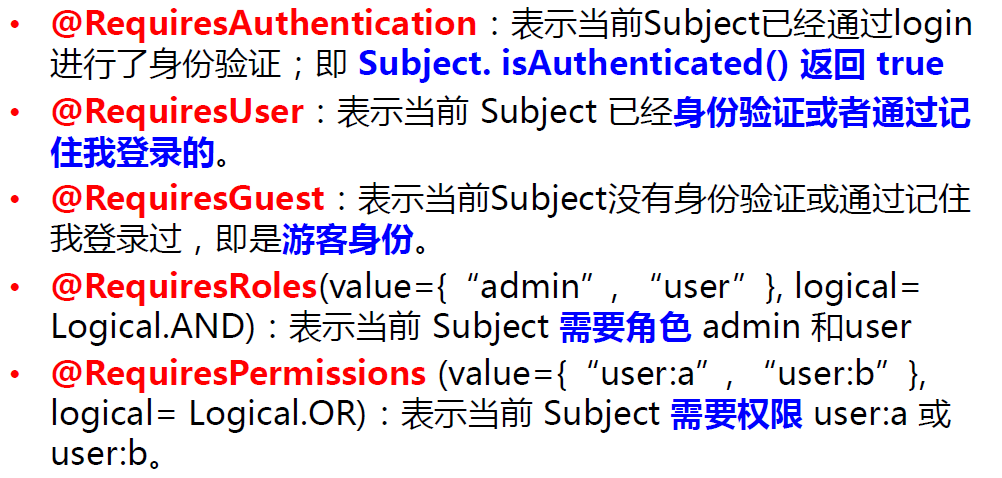
lacksPermission:如果当前Subject没有权限将显示body体内容

<shiro:lacksPermission name = "org:create">

用户[<shiro:pricipal/>] 没有权限org:create

</shiro:lacksPermission>

# 权限注解



编写一个测试类 shiroservices （切入的测试方法）

然后放在IOC容器中 ， 然后在handle层添加 （control层）进行页面跳转的处理

Services的编写

@RequiresRoles({"admin"})

public class shiroservice {

public void testMethod(){

System.out.println("testMethod , time" + new Date());

}

}

交给IOC管理

<bean id="shiroservice" class="com.shiro.services.shiroservice"></bean>

Handle的处理 （相当于strusts 进行页面的跳转）

@Autowired

private shiroservice shiroService;

@RequestMapping("/testShiroAnnotation")

public String testShiroAnnotation() {

shiroService.testMethod();

return "redirect:/list.jsp";

}

前台页面 的对应处理

<a href="login/testShiroAnnotation"> Test for the shiro annoation</a>

初始化资源和权限

# Shiro的会话管理

不依赖于底层的容器 不管是javase 或者是javaee都可以

建议在开发中，Controller层使用原生的HttpSession对象，在Service层中使用Shiro提供的Session对象。如果在Service层中使用HttpSession对象，那么属于侵入式，并不建议这么做。Shiro提供的Session能够很好的解决这个问题

例子中的handle层的是httpsession 对象但是在service使用的是 shiro的session

public String testShiroAnnotation(HttpSession session) {

session.setAttribute("key", "123456");

在 org.apache.shiro.session.Session session =SecurityUtils.getSubject().getSession();

Object val = session.getAttribute("key");

System.out.println("Service SessionVal" + val);

}

## 1，会话相关API

①Subject.getSession()

获取会话，等价于Subject.getSession(true)，即如果当前没有创建session对象会创建一个；Subject.getSession(false)，如果当前没有创建session对象则返回null。

②Subject.getSession(true)

③session.getId()

获取当前会话的唯一标识。

④session.getHost()

获取当前会话的主机地址。

⑤session.getTimeout() & session.setTimeout(毫秒)

设置/获取当前Session的过期时间。

⑥session.getStartTimestamp() & session.getLastAccessTime()

获取会话的启动时间及最后访问时间；如果是J2SE环境需要自己定期调用session.touch()去更新最后访问时间；如果是Web环境，每次进入ShiroFilter都会自动调用session.touch()来更新最后访问时间。

⑦session.touch() & session.stop()

更新会话最后访问时间以及销毁会话；Subject.logout()会自动调用session.stop()。在Web应用中，调用HttpSession.invalidate()也会自动调用session.stop()来销毁shiro的会话。

⑧session.setAttribute(key,val) & session.getAttribute(key) & session.removeAttribute(key)

设置/获取/删除 会话属性

## 2，会话监听器(SessionListener接口)

①onStart(Session)

监听会话创建事件

②onStop(Session)

监听会话销毁事件

③onExpiration(Session)

监听会话过期事件

## 3 SessionDao

详细用的不是很多

详情请查询 <https://www.cnblogs.com/startcaft/p/7456933.html>

Shiro的缓存

<bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">

<property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>

</bean>

在realm配置中可以刚添加指定的缓存文件

<bean id ="shiroRealm" class="com.shiro.test.shirorealms.shiroRealm">

<property name="credentialsMatcher">

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

<property name="hashAlgorithmName" value="MD5"></property>

<property name="hashIterations" value="1024"></property>

</bean>

</property>

**<property name="authenticationCache"></property>**

</bean>

# Remember me 的实现

首先需注意区分的是认证和记住我

1 subject.isAuthenticated() 表示用户进行了身份验证登录的即使有subject.login进行了登录

2 subject.isRemembered() 表示用户通过记录我登陆的 此时可能不是你使用的电脑（比如是你的朋友使用的电脑

或者是你的cookie 被盗用）

3两者二选一 subject.isAuthenticated() == true 则subject.isRemeberd() == false；

具体remember me 的实现

首先是页面的部分

userName<input type="text"id="userName" name="userName"value="<shiro:principal/>" /><br/>

password:<input type="text" id="password" name="password" value="${user.password}"/><br/>

下次自动登录：<inputtype="checkbox"id="rememberMe"name="rememberMe" value="true"/><br/>

<input type="button" id="login" value="login"/><font color="red"/><font>${error}</font>

其次需要配置spring-shiro.xml 没有整合的就在applicationContext.xml配置

<!-- 会话Cookie模板 -->

<bean id="sessionIdCookie"class="org.apache.shiro.web.servlet.SimpleCookie">

<constructor-arg value="sid"/>

<property name="httpOnly" value="true"/>

<property name="maxAge" value="-1"/>

</bean>

<bean id="rememberMeCookie" class="org.apache.shiro.web.servlet.SimpleCookie">

<constructor-arg value="rememberMe"/>

<property name="httpOnly" value="true"/>

<property name="maxAge" value="2592000"/><!-- 30天 -->

</bean>

RememberMe 管理器 , cipherKey是rememberMe 的cookie算法 默认的AES算法

<!-- rememberMe管理器 -->

<bean id="rememberMeManager" class="org.apache.shiro.web.mgt.CookieRememberMeManager">

<property name="cipherKey" value="#{T(org.apache.shiro.codec.Base64).decode('4AvVhmFLUs0KTA3Kprsdag==')}"/>

<property name="cookie" ref="rememberMeCookie"/>

</bean>

设置securityManager安全管理器的rememberMeManager；securityManager可以实现自己的relam

<!-- 安全管理器 -->

<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

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

<property name="realm" ref="myRealm"/>

</bean>

在html里定义，rememberMeParam，即rememberMe请求参数名，请求参数是boolean类型，true表示rememberMe。默认是false

<!-- 基于Form表单的身份验证过滤器 -->

<bean id="formAuthenticationFilter" class="org.apache.shiro.web.filter.authc.FormAuthenticationFilter">

<property name="usernameParam" value="userName"/>

<property name="passwordParam" value="password"/>

<property name="rememberMeParam" value="rememberMe"/>

<property name="loginUrl" value="/login"/>

</bean>

JAVA层，设置rememberMe。

subject.isRemembered()：记住用户后会保存cookie，但是登录的时候可能不是真的你，能用你浏览器的都能打开，像/main=user这种权限都能打开。

subject.isAuthenticated()：表示用用户名密码登录的用户，像/userMange=authc都能打开。

像平常权限的网页 建议设置成=user 但是像牵扯到安全事务的比如修改密码之类的 就要设置成=authc

authc拦截器会判断用户是否是通过Subject.login（isAuthenticated()==true）登录的，如果是才放行，否则会跳转到登录页面叫你重新登录

Subject subject = SecurityUtils.getSubject();

// 用户输入的账号和密码,,存到UsernamePasswordToken对象中..然后由shiro内部认证对比,

// 认证执行者交由ShiroDbRealm中doGetAuthenticationInfo处理

// 当以上认证成功后会向下执行,认证失败会抛出异常

UsernamePasswordToken token = new UsernamePasswordToken(userName, password);

try {

if(rememberMe){

token.setRememberMe(true);

}else {

token.setRememberMe(false);

}

subject.login(token);

} catch (LockedAccountException lae) {

token.clear();

request.setAttribute("error", "用户已经被锁定不能登录，请与管理员联系！");

return "login";

}