**SpringSecurity第二天**

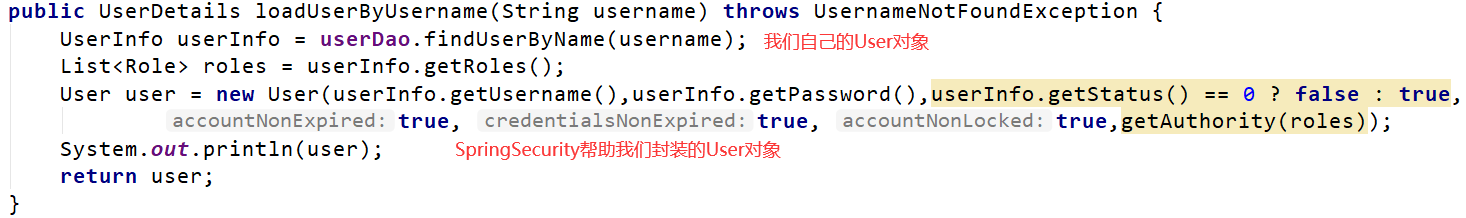
上海千锋 文威

* 设置用户状态
* remember me
* 动态展示菜单
* 方法级别权限控制
* 权限不足异常处理

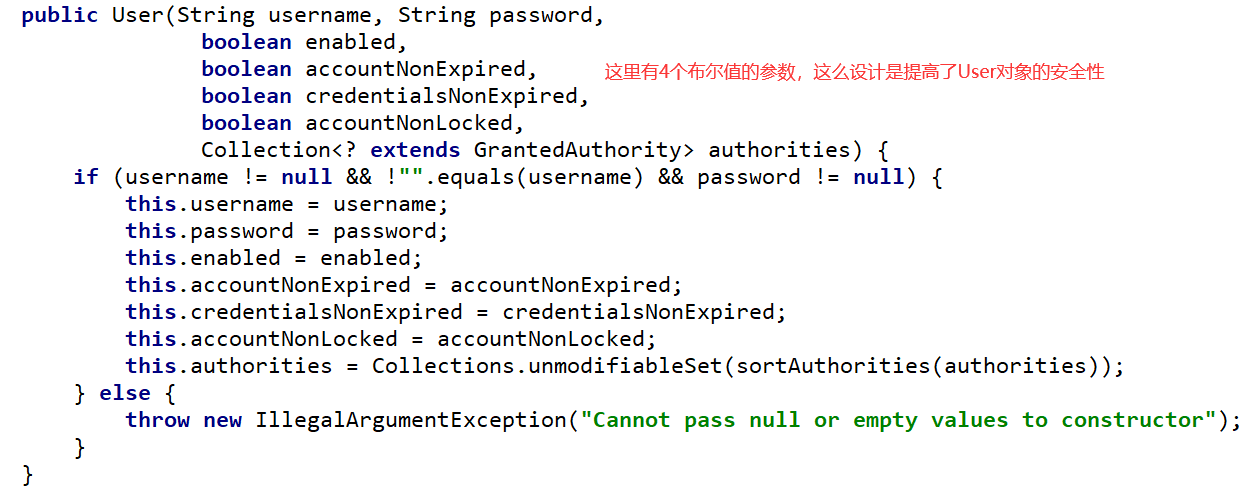
# 1 设置用户状态

## 1.1 源码解析

首先我们回顾自己编写认证的业务代码。



那么疑问来了，我们SpringSecurity到底给我们封装的是什么样子的User对象呢?我们可以点击进入源码查看。



可以看到，这个构造方法里多了四个布尔类型的构造参数，其实我们使用的三个构造参数的构造方法里这四个布尔 值默认都被赋值为了true，那么这四个布尔值到底是何意思呢？

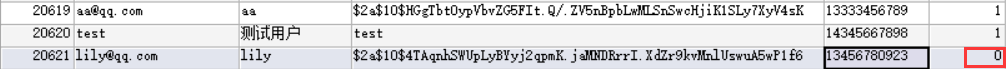
* boolean enabled 是否可用
* boolean accountNonExpired 账户是否失效
* boolean credentialsNonExpired 秘密是否失效
* boolean accountNonLocked 账户是否锁定

## 1.2 判断认证用户的状态

这四个参数必须同时为true认证才可以，为了节省时间，我只用第一个布尔值做个测试。我们新增一个用户，设置用户状态为0(不可用)。然后给它赋予相关的角色信息。



数据表中记录如下:



同时赋予ROLE和ADMIN的角色(设置中间表users\_role即可)



现在我们使用lily这个账户去登录。

经测试，无法访问主页。

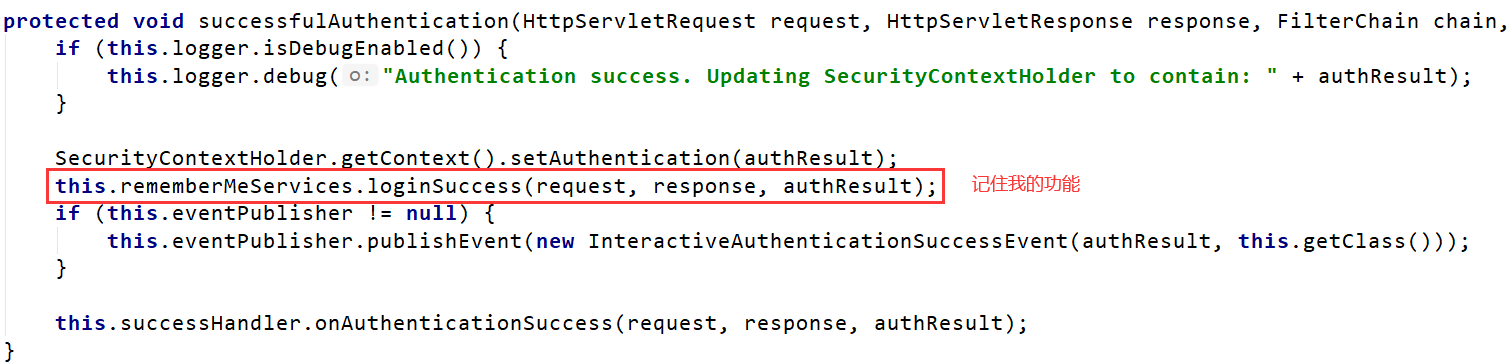


# 2 remember功能

## 2.1 记住我功能原理分析

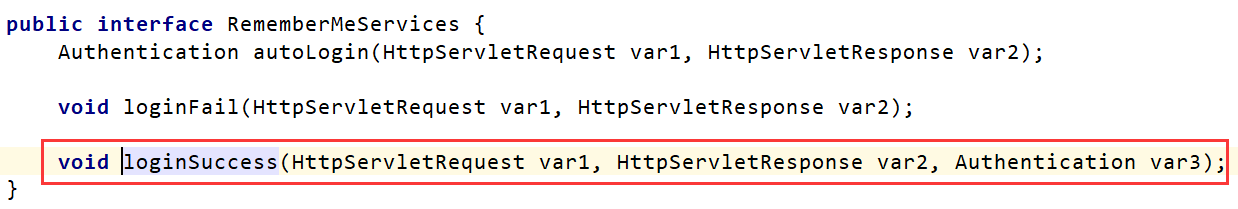
还记得前面咱们分析认证流程时，提到的记住我功能吗？

在UsernamePasswordAuthenticationFilter的父类里面。现在继续跟踪找到AbstractRememberMeServices对象的loginSuccess方法：

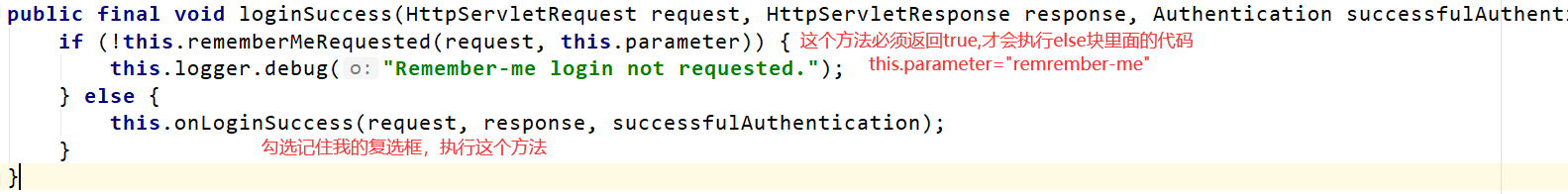


点击进入loginSuccess方法。

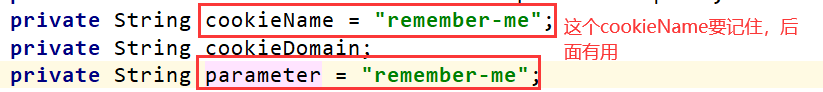
这是一个接口提供的方法：



点击找它的实现类：

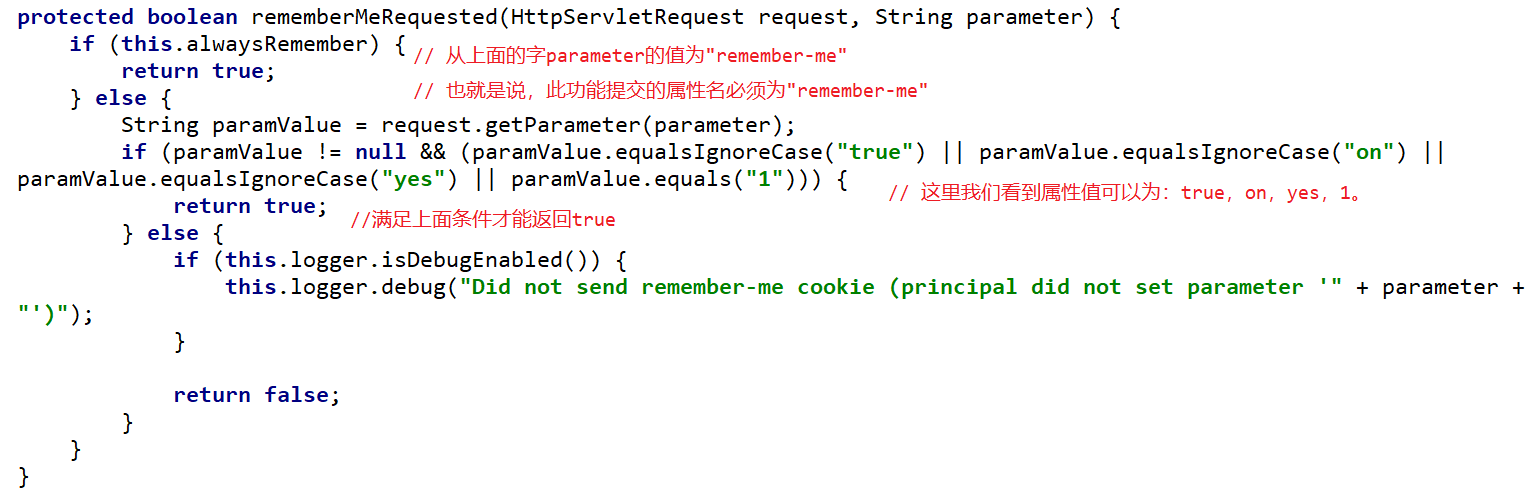


我们点击查看：



我们得出：这个this.parameter的值就是一个字符串”remember-me”。

再点进去上面if判断中的rememberMeRequested方法，还在当前类中：



如果上面方法返回true，就表示页面勾选了记住我选项了。继续顺着调用的方法找到PersistentTokenBasedRememberMeServices的onLoginSuccess方法：



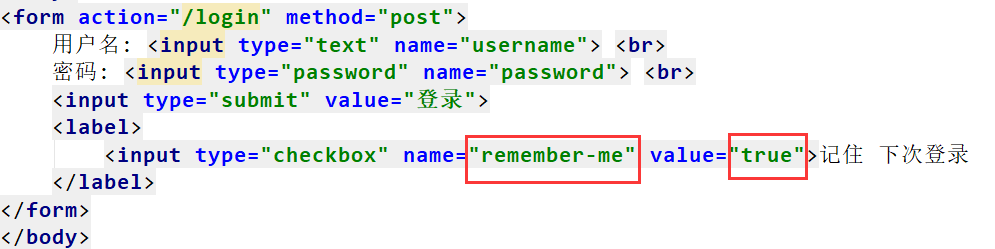
这是一个抽象方法，继续看实现类。



## 2.2 记住我功能页面代码

注意name和value属性的值不要写错哦！

在login.jsp定义记住我的复选框按钮。

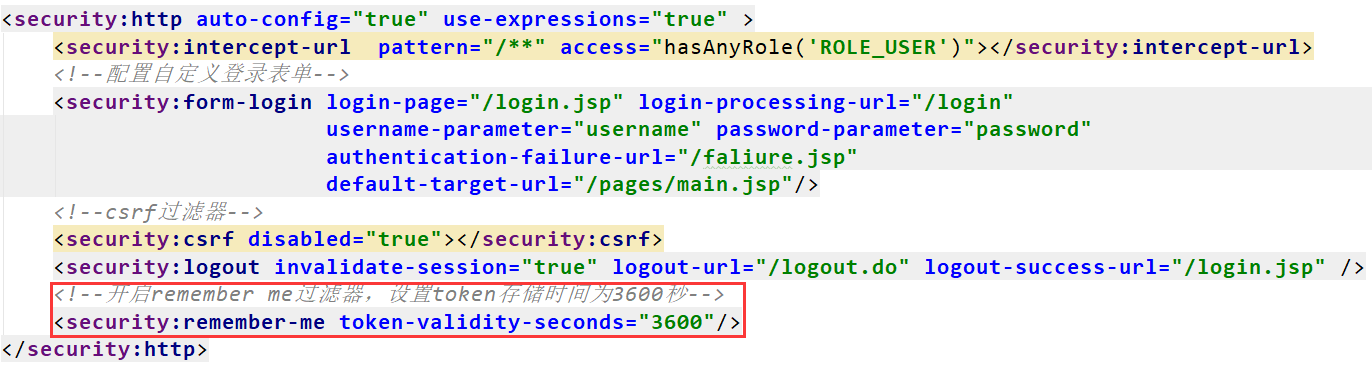


## 2.3 开启remember me过滤器

在springSecurity.xml里面定义remember-me过滤器。

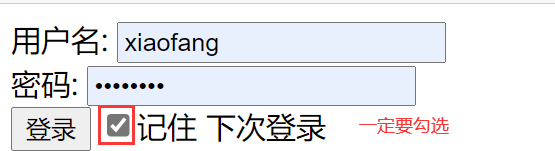
|  |
| --- |
| *<!--开启remember me过滤器，设置token存储时间为3600秒-->* <**security:remember-me token-validity-seconds="3600"**/> |

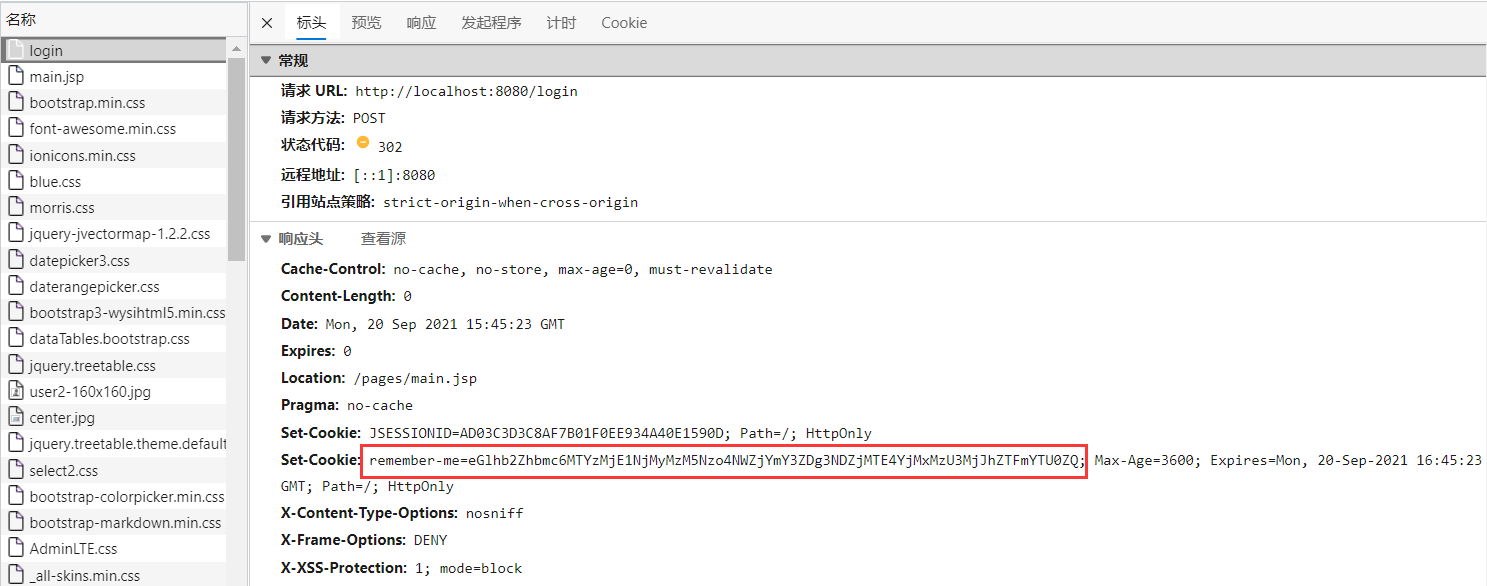
具体定义如下:



说明：RememberMeAuthenticationFilter中功能非常简单，会在打开浏览器时，自动判断是否认证，如果有则调用autoLogin进行自动认证。

我们可以测试一下：





此时我们关闭浏览器直接访问主页:localhost:8080。



可以直接进入主页面。

## 2.4 remember me安全性分析

记住我功能方便是大家看得见的，但是安全性却令人担忧。因为Cookie毕竟是保存在客户端的，很容易盗取，而且 cookie的值还与用户名、密码这些敏感数据相关，虽然加密了，但是将敏感信息存在客户端，还是不太安全。那么 这就要提醒喜欢使用此功能的，用完网站要及时手动退出登录，清空认证信息。 此外，SpringSecurity还提供了remember me的另一种相对更安全的实现机制 :在客户端的cookie中，仅保存一个 无意义的加密串（与用户名、密码等敏感数据无关），然后在db中保存该加密串-用户信息的对应关系，自动登录时，用cookie中的加密串，到db中验证，如果通过，自动登录才算通过。

## 2.5 持久化remember me安全信息

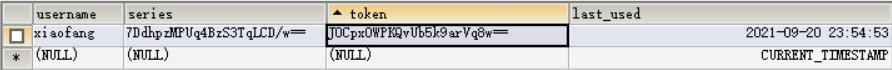
创建一张表，注意这张表的名称和字段都是固定的，不要修改。

|  |
| --- |
| CREATE TABLE `persistent\_logins` (  `username` varchar(64) NOT NULL,  `series` varchar(64) NOT NULL,  `token` varchar(64) NOT NULL,  `last\_used` timestamp NOT NULL,  PRIMARY KEY (`series`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

然后将spring-security.xml中改为：

|  |
| --- |
| <**security:remember-me data-source-ref="dataSource"  token-validity-seconds="60"  remember-me-parameter="remember-me"**/> |

最后测试发现数据库中自动多了一条记录：



# 3动态展示菜单

在aside.jsp对每个菜单通过SpringSecurity标签库指定访问所需角色。





我们做个测试，xiaofang这个用户现在只有普通用户角色ROLE\_USER，用xiaofang登录后，果然只看到了基础数据模块：



那么问题来了，是不是现在已经授权成功了呢？答案是否定的！你可以试试直接去访问用户的http请求地址：localhost:8080/user/findAll.do



我们发现xiaofang其实是可以操作用户模块的，只是没有把用户功能展示给xiaofang而已！

总结一句：页面动态菜单的展示只是为了用户体验，并未真正控制权限！

# 4 授权操作

说明：SpringSecurity可以通过注解的方式来控制类或者方法的访问权限。注解需要对应的注解支持，若注解放在 controller类中，对应注解支持应该放在mvc配置文件中，因为controller类是有mvc配置文件扫描并创建的，同理，注解放在service类中，对应注解支持应该放在spring配置文件中。由于我们现在是模拟业务操作，所以就把注解放在controller类中了。

## 4.1 开启权限使用注解支持

在spring-mvc.xml里面配置

|  |
| --- |
| *<?***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:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xmlns:aop="http://www.springframework.org/schema/aop"  xmlns:security="http://www.springframework.org/schema/security"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/mvc  http://www.springframework.org/schema/mvc/spring-mvc.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop.xsd  http://www.springframework.org/schema/security  http://www.springframework.org/schema/security/spring-security.xsd"**>  *<!--配置包扫描-->   <!-- 扫描controller的注解，别的不扫描 -->* <**context:component-scan base-package="com.qf.controller"**>  </**context:component-scan**>   *<!--配置视图解析器-->* <**bean id="internalResourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  <**property name="prefix" value="/pages/"**></**property**>  <**property name="suffix" value=".jsp"**></**property**>  </**bean**>   *<!--静态资源不过滤-->* <**mvc:resources location="/css/" mapping="/css/\*\*"** />  <**mvc:resources location="/img/" mapping="/img/\*\*"** />  <**mvc:resources location="/js/" mapping="/js/\*\*"** />  <**mvc:resources location="/plugins/" mapping="/plugins/\*\*"** />   *<!--开启使用注解表达式使用支持-->* <**security:global-method-security pre-post-annotations="enabled" proxy-target-class="true"**/>   *<!--  支持AOP的注解支持，AOP底层使用代理技术  JDK动态代理，要求必须有接口  cglib代理，生成子类对象，proxy-target-class="true" 默认使用cglib的方式  -->  <!--<aop:aspectj-autoproxy proxy-target-class="true"/>-->   <!--开启对springmvc注解支持-->* <**mvc:annotation-driven**></**mvc:annotation-driven**> </**beans**> |

## 4.2 在方法上添加对应注解

|  |
| --- |
| */\*\*  \* 查询所有  \** ***@param model*** *\** ***@return*** *\*/* @RequestMapping(**"findAll.do"**) @PreAuthorize(**"hasRole('ROLE\_ADMIN')"**) **public** String findAll(Model model){  List<UserInfo> userList = **userService**.findAll();  model.addAttribute(**"userList"**,userList);  **return "user-list"**; } |

## 4.3 编写异常处理页面

编写403.jsp页面

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>Title</**title**> </**head**> <**body**>  <**h3**>权限不足，无法操作</**h3**> </**body**> </**html**> |

在web.xml里面配置这个错误页面。

|  |
| --- |
| *<!--配置错误页面-->* <**error-page**>  <**error-code**>403</**error-code**>  <**location**>/403.jsp</**location**> </**error-page**> |