一、设置用户状态

1.1 源码分析

用户认证业务里，我们封装User对象时，选择了三个构造参数的构造方法，其实还有另一个构造方法：

```
public User(String username, String password, boolean enabled, boolean accountNonExpired,
boolean credentialsNonExpired, boolean accountNonLocked, Collection<? extends GrantedAuthority>
authorities) {
    if (username != null && !"".equals(username) && password != null) {
        this.username = username;
        this.password = password;
        this.enabled = enabled;
        this.accountNonExpired = accountNonExpired;
        this.credentialsNonExpired = credentialsNonExpired;
        this.accountNonLocked = accountNonLocked;
        this.authorities = Collections.unmodifiableSet(sortAuthorities(authorities));
    } else {
        throw new IllegalArgumentException("Cannot pass null or empty values to constructor");
    }
}
```

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，修改认证业务代码：

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    SysUser sysUser = userDao.findByName(username);
    if(sysUser==null){
        //若用户名不对，直接返回null，表示认证失败。
        return null;
    }
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();
    List<SysRole> roles = sysUser.getRoles();
    for (SysRole role : roles) {
        authorities.add(new SimpleGrantedAuthority(role.getRoleName()));
    }

    //最终需要返回一个SpringSecurity的UserDetails对象，{noop}表示不加密认证。
```

```
    return new User(sysUser.getUsername(),
                    sysUser.getPassword(),
                    sysUser.getStatus()==1,
                    true,
                    true,
                    true, authorities);
}
```

此刻，只有用户状态为1的用户才能成功通过认证！

二、退出登录

注意：一旦开启了csrf防护功能，logout处理器便只支持POST请求方式了！

修改header.jsp中注销请求：

```
<form action="${pageContext.request.contextPath}/logout" method="post">
    <security:csrfInput/>
    <input type="submit" value="注销">
</form>
```

三、remember me

3.1 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？

现在继续跟踪找到AbstractRememberMeServices对象的loginSuccess方法：

```
public abstract class AbstractRememberMeServices implements RememberMeServices,
InitializingBean, LogoutHandler {
    public final void loginSuccess(HttpServletRequest request, HttpServletResponse response,
Authentication successfulAuthentication) {
        // 判断是否勾选记住我
        // 注意：这里this.parameter点进去是上面的private String parameter = "remember-me";
        if (!this.rememberMeRequested(request, this.parameter)) {
            this.logger.debug("Remember-me login not requested.");
        } else {
            //若勾选就调用onLoginSuccess方法
            this.onLoginSuccess(request, response, successfulAuthentication);
        }
    }
}
```

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

```
protected boolean rememberMeRequested(HttpServletRequest request, String parameter) {
    if (this.alwaysRemember) {
```

```
        return true;
    } else {
        // 从上面的字parameter的值为"remember-me"
        // 也就是说，此功能提交的属性名必须为"remember-me"
        String paramValue = request.getParameter(parameter);
        // 这里我们看到属性值可以为：true, on, yes, 1。
        if (paramValue != null && (paramValue.equalsIgnoreCase("true") ||
paramValue.equalsIgnoreCase("on") || paramValue.equalsIgnoreCase("yes") ||
paramValue.equals("1"))) {
            //满足上面条件才能返回true
            return true;
        } else {
            if (this.logger.isDebugEnabled()) {
                this.logger.debug("Did not send remember-me cookie (principal did not set
parameter '" + parameter + "')");
            }
            return false;
        }
    }
}
```

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：

```
public class PersistentTokenBasedRememberMeServices extends AbstractRememberMeServices {
    protected void onLoginSuccess(HttpServletRequest request, HttpServletResponse response,
Authentication successfulAuthentication) {
        // 获取用户名
        String username = successfulAuthentication.getName();
        this.logger.debug("Creating new persistent login for user " + username);

        //创建记住我的token
        PersistentRememberMeToken persistentToken = new PersistentRememberMeToken(username,
this.generateSeriesData(), this.generateTokenData(), new Date());

        try {
            //将token持久化到数据库
            this.tokenRepository.createNewToken(persistentToken);
            //将token写入到浏览器的Cookie中
            this.addCookie(persistentToken, request, response);
        } catch (Exception var7) {
            this.logger.error("Failed to save persistent token ", var7);
        }
    }
}
```

3.2 记住我功能页面代码

注意name和value属性的值不要写错哦！

```
<p class="login-box-msg">登录系统</p>
<form action="${pageContext.request.contextPath}/login" method="post">
    <security:csrfInput/>
    <div class="form-group has-feedback">
        <input type="text" name="username" class="form-control"
               placeholder="用户名"> <span
                                       class="glyphicon glyphicon-envelope form-control-
feedback"></span>
    </div>
    <div class="form-group has-feedback">
        <input type="password" name="password" class="form-control"
               placeholder="密码"> <span
                                      class="glyphicon glyphicon-lock form-control-feedback">
</span>
    </div>
    <div class="row">
        <div class="col-xs-8">
            <div class="checkbox icheck">
                <!-- 注意name和value属性的值不要写错哦 -->
                <label><input type="checkbox" name="remember-me" value="true"> 记住 下次自动登录
</label>
            </div>
        </div>
        <!-- /.col -->
        <div class="col-xs-4">
            <button type="submit" class="btn btn-primary btn-block btn-flat">登录</button>
        </div>
        <!-- /.col -->
    </div>
</form>
```

先测试一下，认证通过后，关掉浏览器，再次打开页面，发现还要认证！为什么没有起作用呢?

这是因为remember me功能使用的过滤器RememberMeAuthenticationFilter默认是不开启的!

3.3 开启remember me过滤器

```
<!--设置可以用spring的el表达式配置Spring Security并自动生成对应配置组件（过滤器）-->
<security:http auto-config="true" use-expressions="true">
    <!--省略其余配置-->
    <!--开启remember me过滤器，设置token存储时间为60秒-->
    <security:remember-me token-validity-seconds="60"/>
</security:http>
```

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，如果没有则调用autoLogin进行自动认证。

3.4 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。

3.5 持久化remember me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：

```
<!--
        开启remember me过滤器，
        data-source-ref="dataSource" 指定数据库连接池
        token-validity-seconds="60" 设置token存储时间为60秒 可省略
        remember-me-parameter="remember-me" 指定记住的参数名 可省略
-->
<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：

username	series	token	last_used
xiaozhi	/SxF3gOpXiAwKImtpW8MpQ==	0x3HXW4rhkSdQjSnOeZc+A==	2019-06-27 20:32:48
(NULL)	(NULL)	(NULL)	(NULL)

四、显示当前认证用户名

在header.jsp中找到页面头部最右侧图片处添加如下信息：

```
<span class="hidden-xs">
    <security:authentication property="principal.username" />
</span>
或者
<span class="hidden-xs">
    <security:authentication property="name" />
</span>
```

五、授权准备工作

为了模拟授权操作，咱们临时编写两个业务功能：

处理器代码：

```
//ProductController
@Controller
@RequestMapping("/product")
public class ProductController {
    @RequestMapping("/findAll")
    public String findAll(){
        return "product-list";
    }
}

//OrderController
@Controller
@RequestMapping("/order")
public class OrderController {
    @RequestMapping("/findAll")
    public String findAll(){
        return "order-list";
    }
}
```

aside.jsp页面：

```
<ul class="treeview-menu">
    <li id="system-setting"><a href="${pageContext.request.contextPath}/product/findAll">
        <i class="fa fa-circle-o"></i> 产品管理</a>
    </li>
    <li id="system-setting"><a href="${pageContext.request.contextPath}/order/findAll">
        <i class="fa fa-circle-o"></i> 订单管理</a>
    </li>
</ul>
```

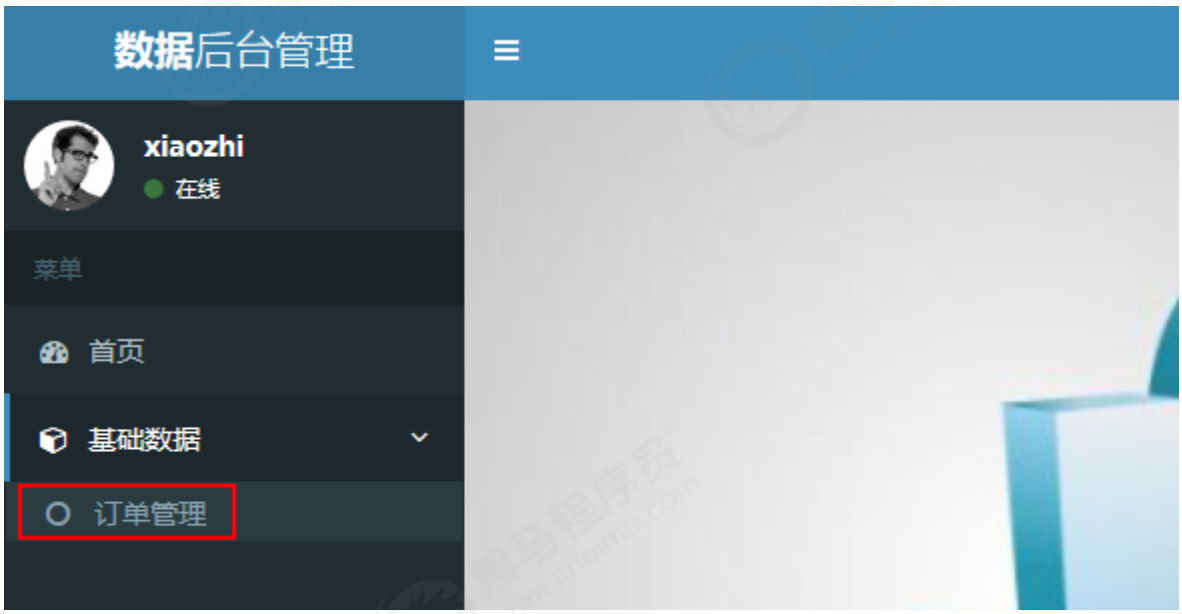
六、动态展示菜单

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

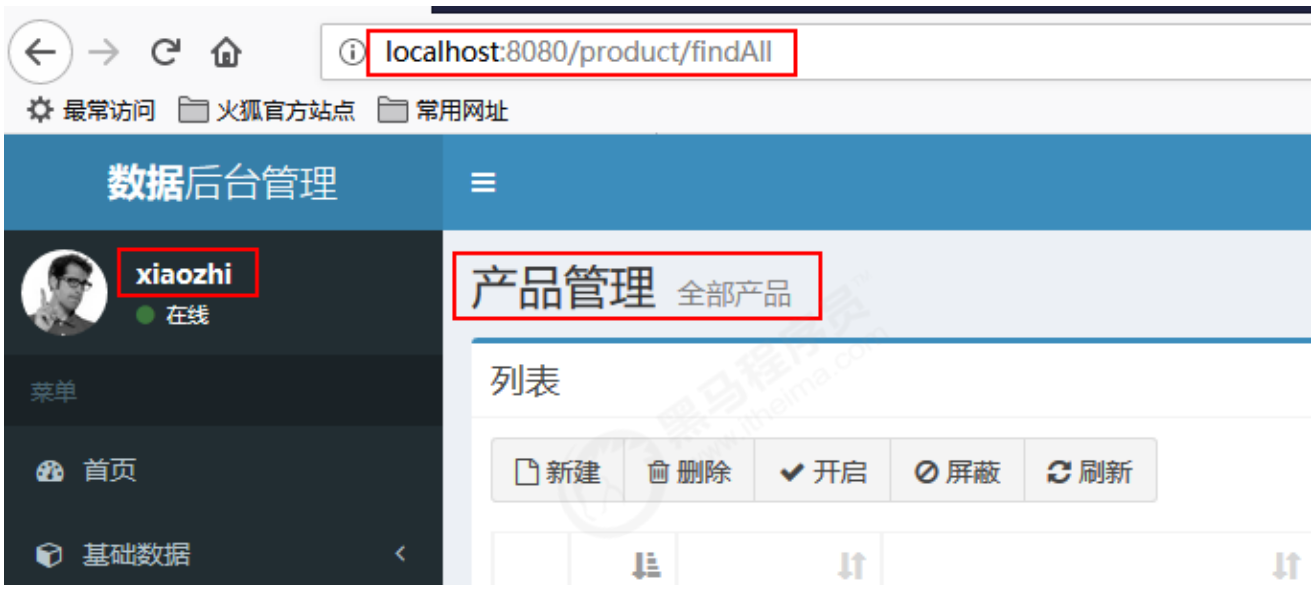
```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<aside class="main-sidebar">
    <!-- sidebar: style can be found in sidebar.less -->
    <section class="sidebar">
        <!-- Sidebar user panel -->
        <div class="user-panel">
            <div class="pull-left image">
                <img src="${pageContext.request.contextPath}/img/user2-160x160.jpg"
                     class="img-circle" alt="User Image">
            </div>
            <div class="pull-left info">
                <p>
                    <security:authentication property="principal.username"/>
                </p>
                <a href="#"><i class="fa fa-circle text-success"></i> 在线</a>
            </div>
        </div>

        <!-- sidebar menu: : style can be found in sidebar.less -->
        <ul class="sidebar-menu">
            <li class="header">菜单</li>
            <li id="admin-index"><a
                    href="${pageContext.request.contextPath}/pages/main.jsp"><i
                    class="fa fa-dashboard"></i> <span>首页</span></a></li>
           <!-- 唯有超级管理员可以操作权限 -->
            <security:authorize access="hasAnyRole('ROLE_ADMIN')">
                <li class="treeview"><a href="#"> <i class="fa fa-cogs"></i>
                <span>系统管理</span> <span class="pull-right-container"> <i
                            class="fa fa-angle-left pull-right"></i>
                </span>
                </a>
                    <ul class="treeview-menu">

                        <li id="system-setting"><a
                                href="${pageContext.request.contextPath}/user/findAll"> <i
                                class="fa fa-circle-o"></i> 用户管理
                        </a></li>
                        <li id="system-setting"><a
                                href="${pageContext.request.contextPath}/role/findAll"> <i
                                class="fa fa-circle-o"></i> 角色管理
                        </a></li>
                        <li id="system-setting"><a
                                href="${pageContext.request.contextPath}/pages/permission-
list.jsp">
                            <i class="fa fa-circle-o"></i> 权限管理
                        </a></li>
                    </ul>
                </li>
            </security:authorize>
            <!-- 基础模块超级管理员和用户权限都可以操作 -->
            <security:authorize access="hasAnyRole('ROLE_ADMIN', 'ROLE_USER')">
                <li class="treeview"><a href="#"> <i class="fa fa-cube"></i>
```

```
                    <span>基础数据</span> <span class="pull-right-container"> <i
                            class="fa fa-angle-left pull-right"></i>
                </span>
                </a>
                    <ul class="treeview-menu">
                        <!-- 产品模块还需要产品角色，注意这里还需再次指定超级管理员 -->
                        <security:authorize access="hasAnyRole('ROLE_ADMIN','ROLE_PRODUCT')">
                            <li id="system-setting"><a
                                    href="${pageContext.request.contextPath}/product/findAll">
                                <i class="fa fa-circle-o"></i> 产品管理
                            </a></li>
                        </security:authorize>
                        <!-- 订单模块，有普通用户角色就可以操作了 -->
                        <li id="system-setting"><a
                                    href="${pageContext.request.contextPath}/order/findAll">
                                <i class="fa fa-circle-o"></i> 订单管理
                        </a></li>
                    </ul>
                </li>
            </security:authorize>
        </ul>
    </section>
    <!-- /.sidebar -->
</aside>
```

我们做个测试，xiaozhi这个用户现在只有普通用户角色ROLE_USER，用xiaozhi登录后，果然只看到了订单管理：



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产品的http请求地址：



我们发现xiaozhi其实是可以操作产品模块的，只是没有把产品功能展示给xiaozhi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！

七、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，并没有service业务代码，所以就把注解放在controller类中了。

7.1 开启授权的注解支持

这里给大家演示三类注解，但实际开发中，用一类即可！

```
<?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:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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/aop
              http://www.springframework.org/schema/aop/spring-aop.xsd
              http://www.springframework.org/schema/tx
              http://www.springframework.org/schema/tx/spring-tx.xsd
              http://www.springframework.org/schema/mvc
              http://www.springframework.org/schema/mvc/spring-mvc.xsd
              http://www.springframework.org/schema/security
              http://www.springframework.org/schema/security/spring-security.xsd">
```

```
    <context:component-scan base-package="com.itheima.controller"/>

    <mvc:annotation-driven/>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <mvc:default-servlet-handler/>

    <!--
    开启权限控制注解支持
    jsr250-annotations="enabled"表示支持jsr250-api的注解，需要jsr250-api的jar包
    pre-post-annotations="enabled"表示支持spring表达式注解
    secured-annotations="enabled"这才是SpringSecurity提供的注解
    -->
    <security:global-method-security jsr250-annotations="enabled"
                                     pre-post-annotations="enabled"
                                     secured-annotations="enabled"/>


</beans>
```

7.2 在注解支持对应类或者方法上添加注解

```
//表示当前类中所有方法都需要ROLE_ADMIN或者ROLE_PRODUCT才能访问
@Controller
@RequestMapping("/product")
@RolesAllowed({"ROLE_ADMIN","ROLE_PRODUCT"})//JSR-250注解
public class ProductController {
    @RequestMapping("/findAll")
    public String findAll(){
        return "product-list";
    }
}

//表示当前类中findAll方法需要ROLE_ADMIN或者ROLE_PRODUCT才能访问
@Controller
@RequestMapping("/product")
public class ProductController {
    @RequestMapping("/findAll")
    @PreAuthorize("hasAnyRole('ROLE_ADMIN','ROLE_PRODUCT')")//spring表达式注解
    public String findAll(){
        return "product-list";
    }
}

//表示当前类中所有方法都需要ROLE_ADMIN或者ROLE_PRODUCT才能访问
@Controller
@RequestMapping("/product")
```

```
@Secured({"ROLE_ADMIN","ROLE_PRODUCT"})//SpringSecurity注解
public class ProductController {
    @RequestMapping("/findAll")
    public String findAll(){
        return "product-list";
    }
}
```

八、权限不足异常处理

大家也发现了，每次权限不足都出现403页面，着实难堪！体会一下：

现在我们立马消灭它！

方式一：在spring-security.xml配置文件中处理

```
<!--设置可以用spring的el表达式配置Spring Security并自动生成对应配置组件（过滤器）-->
<security:http auto-config="true" use-expressions="true">
    <!--省略其它配置-->
    <!--403异常处理-->
    <security:access-denied-handler error-page="/403.jsp"/>
</security:http>
```

方式二：在web.xml中处理

```
<error-page>
    <error-code>403</error-code>
    <location>/403.jsp</location>
</error-page>
```

方式三：编写异常处理器

```
@ControllerAdvice
public class ControllerExceptionAdvice {

    //只有出现AccessDeniedException异常才调转403.jsp页面
    @ExceptionHandler(AccessDeniedException.class)
    public String exceptionAdvice(){
        return "forward:/403.jsp";
    }
}
```