

第7章 系统管理-权限设置

学习目标

- 了解认证和授权的概念
- 掌握Spring Security入门案例开发过程
- 掌握Spring Security实现认证的过程
- 掌握Spring Security实现授权的过程

1. 权限控制、SpringSecurity入门及进阶

【目标】

了解认证和授权的概念

【路径】

1: 认证和授权的概念

- 认证：登录（用户名和密码）
- 授权：访问系统功能的权限

2: 权限模块的数据模型

- 用户表
- 角色表
- 权限表
- 菜单表

用户,角色是多对多

权限,角色是多对多

菜单,角色是多对多

【讲解】

1.1. 认证和授权概念【理解】

前面我们已经完成了传智健康后台管理系统（health_web）的部分功能，例如检查项管理、检查组管理、套餐管理、预约设置等。接下来我们需要思考2个问题：

问题1：在生产环境下我们如果不登录后台系统就可以完成这些功能操作吗？

答案显然是否定的，要操作这些功能必须首先登录到系统才可以。（用户登录系统-->认证）

问题2：是不是所有用户，只要登录成功就都可以操作所有功能呢？

答案是否定的，并不是所有的用户都可以操作这些功能。不同的用户可能拥有不同的权限，这就需要进行授权了。（用户登录之后，对每个用户进行授权，通过授权去访问系统中不同的功能-->授权）

认证：系统提供的用于识别用户身份的功能，通常提供用户名和密码进行登录其实就是在进行认证，认证的目的是让系统知道你是谁。

说白了：登陆系统了，就可以访问，没登陆就不能访问。验证用户名与密码是否存在数据库中 (t_user)

授权：用户认证成功后，需要为用户授权，其实就是指当前用户可以操作哪些功能。

用户登陆后，查询获取用户所拥有的权限绑定到用户，当进行权限验证时就可以取用与登陆用户绑定的权限与访问的资源标定的权限进行匹配

商店里的 矿泉水 3块钱 资源权限标定。标定好了需要访问这个资源时的权限

登陆用户 有多少钱？老婆给予的，老婆授权。

授权是为了做权限验证的。登陆用户身上所拥有的权限集合

/checkitem/findPage.do 查询检查项 只有健康管理师才可以访问

员工A 来登陆 (查看套餐的权限) 来访 查询检查项(/checkitem/findPage.do 健康管理师) 报没有权限

认证与授权的目的，更好的保护资源(使用后台管理功能)

本章节就是要对后台系统进行权限控制，其本质就是对用户进行认证和授权。

1.2. 权限模块数据模型

前面已经分析了认证和授权的概念，要实现最终的权限控制，需要有一套表结构支撑：

用户表t_user、权限表t_permission、角色表t_role、菜单表t_menu、用户角色关系表t_user_role、角色权限关系表t_role_permission、角色菜单关系表t_role_menu。

表之间关系如下图：



通过上图可以看到，权限模块共涉及到7张表。在这7张表中，角色表起到了至关重要的作用，其处于核心位置，我们把基于角色的权限控制叫做RBAC，因为用户、权限、菜单都和角色是多对多关系。



接下来我们可以分析一下在认证和授权过程中分别会使用到哪些表：

认证过程：只需要用户表就可以了，在用户登录时可以查询用户表t_user进行校验，判断用户输入的用户名和密码是否正确。



授权过程：用户必须完成认证之后才可以进行授权，首先可以根据用户查询其角色，再根据角色查询对应的菜单，这样就确定了用户能够看到哪些菜单。然后再根据用户的角色查询对应的权限，这样就确定了用户拥有哪些权限。所以授权过程会用到上面7张表。



【小结】

1. 认证和授权

- 认证: 提供账号和密码进行登录认证, 系统知道你的身份
- 授权: 根据不同的身份, 赋予不同的权限, 不同的权限, 可访问系统不同的功能(系统的功能也要标定访问权限)

2. RBAC权限模块数据模型 (基于角色的权限控制)

- 用户表
- 角色表
- 权限表
- 菜单表

用户,角色是多对多 用户角色表

权限,角色是多对多 角色权限表

菜单,角色是多对多 角色菜单表

一共7张表

1.3. Spring Security简介

【目标】

知道什么是Spring Security

【路径】

1. Spring Security简介
2. Spring Security使用需要的坐标

【讲解】

Spring Security是 Spring提供的安全认证服务的框架。使用Spring Security可以帮助我们简化认证和授权的过程。官网: <https://spring.io/projects/spring-security>



对应的maven坐标:

```
1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-web</artifactId>
4   <version>5.0.5.RELEASE</version>
5 </dependency>
6 <dependency>
7   <groupId>org.springframework.security</groupId>
8   <artifactId>spring-security-config</artifactId>
9   <version>5.0.5.RELEASE</version>
10 </dependency>
```

常用的权限框架除了Spring Security, 还有Apache的shiro框架。

【小结】

1. SpringSecurity是Spring家族的一个安全框架, 简化我们开发里面的认证和授权过程, (登陆, 访问url时的权限控制 security帮我们做了)
2. SpringSecurity内部封装了Filter (只需要在web.xml容器中配置一个过滤器--代理过滤器, 真实的过滤器(12个)在spring的容器中配置)

3. 常见的安全框架

- Spring的 SpringSecurity 配置复杂 有个好老爸(spring) 无缝整合 链式模式
- Apache的Shiro <http://shiro.apache.org/> 比较简单 apache 写整合 外观者模式

1.4. Spring Security入门案例-【重点】

【目标】

【需求】

使用Spring Security进行控制: 网站(一些页面)需要登录才能访问 (认证)

【路径】

1. 创建Maven health_parent的子工程 springsecurity_demo,导入坐标(依赖health_interface), 打包方式为war
2. 配置web.xml(前端控制器加载spring, SpringSecurity代理过滤器)
3. 创建spring-security.xml (核心配置文件, spring要加载这个配置)

【讲解】

1.4.1. 工程搭建

创建maven工程, 打包方式为war, 为了方便起见我们可以让入门案例工程依赖health_interface, 这样相关的依赖都继承过来了。

 image-20200804101015750

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.itheima</groupId>
8     <artifactId>springsecurity_demo</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <packaging>war</packaging>
12
13     <properties>
14         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15         <maven.compiler.source>1.8</maven.compiler.source>
16         <maven.compiler.target>1.8</maven.compiler.target>
17     </properties>
18
19     <dependencies>
20         <dependency>
21             <groupId>com.itheima</groupId>
22             <artifactId>health_interface</artifactId>
23             <version>1.0-SNAPSHOT</version>
24         </dependency>
25     </dependencies>
```

内置提供index.html页面，内容为“登录成功”!!

1.4.2. 配置web.xml

【路径】

1: DelegatingFilterProxy用于整合第三方框架（代理过滤器，非真正的过滤器，真正的过滤器需要在spring的配置文件）

2: springmvc的核心控制器

在web.xml中主要配置SpringMVC的DispatcherServlet和用于整合第三方框架的DelegatingFilterProxy（代理过滤器，真正的过滤器在spring的配置文件），用于整合Spring Security。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns="http://java.sun.com/xml/ns/javaee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5                               http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
6           version="3.0">
7
8      <filter>
9          <!-- filter-name: 固定为springSecurityFilterChain,不能修改。容器启动时,
10             会从spring容器中获取这个filtername的bean对象 (security启动时创建的bean name)
11             可以修改: 必须在spring容器中配置一个bean对象为你修改后的名称
12             -->
13             <filter-name>springSecurityFilterChain</filter-name>
14          <!-- 代理过滤:
15             自己不干活,拦截的请求转给security的过滤器(springSecurityFilterChain)去处
16             理
17             -->
18             <filter-
19 class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
20             </filter>
21             <filter-mapping>
22                 <filter-name>springSecurityFilterChain</filter-name>
23                 <url-pattern>/*</url-pattern>
24             </filter-mapping>
25
26          <!-- 使用DispatcherServlet或者使用contextLoaderListener 都一样,只要启动容器即可
27          -->
28          <servlet>
29              <servlet-name>dispatchServlet</servlet-name>
30              <servlet-
31 class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
32              <init-param>
33                  <param-name>contextConfigLocation</param-name>
34                  <!-- 启动spring容器,要导入spring-security.xml的配置文件 -->
35                  <param-value>classpath:spring-security.xml</param-value>
36              </init-param>
37              <load-on-startup>1</load-on-startup>
38          </servlet>
39          <servlet-mapping>
40              <servlet-name>dispatchServlet</servlet-name>

```

```

37         <url-pattern>*.do</url-pattern>
38     </servlet-mapping>
39 </web-app>

```

1.4.3. 配置spring-security.xml

【路径】

1: 定义哪些链接可以放行

2: 定义哪些链接不可以放行, 即需要有角色、权限才可以放行

3: 认证管理, 定义登录账号名和密码, 并授予访问的角色、权限

在spring-security.xml中主要配置Spring Security的拦截规则和认证管理器。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xmlns:security="http://www.springframework.org/schema/security"
5         xsi:schemaLocation="http://www.springframework.org/schema/beans
6                               http://www.springframework.org/schema/beans/spring-beans.xsd
7                               http://www.springframework.org/schema/security
8                               http://www.springframework.org/schema/security/spring-security.xsd">
9
10     <!--
11     【重要】【重要】【重要】
12     这里面的所有路径必须以/开头, 否则启动报错
13     -->
14
15     <!-- auto-config: 自动配置, 自动生成login页面, login处理, 退出处理
16         use-expressions: 是否使用spel表达式 true: access的值可以填表达式(hasRole,
17         hasAuthority, hasAny....)
18         false: ROEL_角色名(必须是ROLE_打, 否则启动报错), 或
19         security写死的几个常量
20     -->
21
22     <!-- 拦截规则配置 -->
23     <security:http auto-config="true" use-expressions="false">
24     <!-- pattern="/**" 拦截所有的路径 access="ROLE_ADMIN"
25         要访问符合pattern的url时, 登陆用户必须有ROLE_ADMIN的角色, 如果没有则不能访问
26
27         security:intercept-url: 可以配置多个
28     -->
29         <security:intercept-url pattern="/**" access="ROLE_ADMIN"/>
30     </security:http>
31
32     <!-- 认证管理器 -->
33     <security:authentication-manager>
34     <!-- 认证信息提供者, 认证信息的来源
35         提供登陆用户信息 用户名、密码、权限集合
36         user-service-ref 指向spring容器中一个bean对象, 实现这个UserDetailsService
37         的对象, 提供用户的名称、密码、权限集合
38
39         一旦配置了user-service-ref, 就不要配置security:user-service
40     -->
41         <security:authentication-provider>
42     <!-- 登陆用户信息由我们自己来提供 -->
43         <security:user-service>

```

```

37 <!-- security:user 硬编码一个用户对象在内存中，就不需要去查询数据库了
38 将来应该使用从数据库查询
39 name: 登陆的用户名 password:登陆的密码
40 authorities: 指定的权限(既可以是角色名也可以权限名) authorities与上面access
一致才能访问
41
42 {noop}使用的是明文，no operation 不要对密码做处理
43 -->
44 <security:user name="admin" password="{noop}admin"
authorities="ROLE_ADMIN"/>
45 </security:user-service>
46 </security:authentication-provider>
47 </security:authentication-manager>
48 </beans>

```

{noop}: 表示当前使用的密码为明文。表示当前密码不需要加密。security启动后，会默认创建以下10个加密器用于对密码的校验



自动跳转到登录页面（springSecurity自动提供的）



输入错误用户名和密码



输入正确用户名和密码（admin/admin）



此时说明登陆成功了，成功后，security会自动跳转到成功页面，由于我们没配置成功页面，所以会访问favicon.ico。而我们的项目webapp下也没有favicon.ico，所以会报404

如果新建index.html，可以正常访问index.html



【小结】

使用步骤

1. 创建Maven工程, 添加坐标(health_interface)
2. 配置web.xml(前端控制器,springSecurity权限相关的过滤器 DelegatingFilterProxy, 过滤器的名称不能修改，一定是springSecurityFilterChain)
3. 创建spring-security.xml(security:http 自动配置,使用表达式的方式完成授权，只要具有ROLE_ADMIN的角色权限才能访问系统中的所有功能； 授权管理器，指定用户名admin，密码{noop}admin，具有ROLE_ADMIN的角色权限)

注意实现

- 1.在web.xml里面配置的权限相关的过滤器名字不能改（springSecurityFilterChain）

```

1 <filter>
2   <filter-name>springSecurityFilterChain</filter-name>
3   <filter-
4   class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
5 </filter>
6 <filter-mapping>
7   <filter-name>springSecurityFilterChain</filter-name>
8   <url-pattern>/*</url-pattern>
9 </filter-mapping>

```

2.入门案例里面没有指定密码加密方式的. 配置密码的时候的加{noop}

```

1 <security:user-service>
2   <security:user name="admin" password="{noop}admin"
3   authorities="ROLE_ADMIN"/>
4 </security:user-service>

```

1.5. Spring Security进阶

【目标】

前面我们已经完成了Spring Security的入门案例，通过入门案例我们可以看到，Spring Security将我们项目中的所有资源都保护了起来，要访问这些资源必须要完成认证而且需要具有ROLE_ADMIN角色。

但是入门案例中的使用方法离我们真实生产环境还差很远，还存在如下一些问题：

- 1、项目中我们将所有的资源（所有请求URL pattern=/**）都保护起来，实际环境下往往有一些资源不需要认证也可以访问，也就是可以匿名访问(静态资源)。
- 2、登录页面是由框架生成的，而我们的项目往往会使用自己的登录页面。
- 3、直接将用户名和密码配置在了配置文件中，而真实生产环境下的用户名和密码往往保存在数据库中。
- 4、在配置文件中配置的密码使用明文，这非常不安全，而真实生产环境下密码需要进行加密。

本章节需要对这些问题进行改进。

【路径】

- 1：配置可匿名访问的资源(不需要登录,权限 角色 就可以访问的资源)
- 2：使用指定的登录页面（login.html）
- 3：从数据库查询用户信息
- 4：对密码进行加密
- 5：配置多种校验规则（对访问的页面做权限控制）粗颗粒
- 6：注解方式权限控制（对访问的Controller类做权限控制）细颗粒 可的实现方法名控制
- 7：退出登录

【讲解】

1.5.1. 配置可匿名访问的资源

【路径】

1: 在项目中创建js、css目录并在两个目录下提供任意一些测试文件

2: 在spring-security.xml文件中配置，指定哪些资源可以匿名访问

第一步：在项目中创建js、css目录并在两个目录下提供任意一些测试文件



访问<http://localhost:85/js/vue.js>



第二步：在spring-security.xml文件中配置，指定哪些资源可以匿名访问

```
1 <!--
2 http: 用于定义相关权限控制
3 指定哪些资源不需要进行权限校验，可以使用通配符
4 -->
5 <security:http security="none" pattern="/js/**" />
6 <security:http security="none" pattern="/css/**" />
```

通过上面的配置可以发现，js和css目录下的文件可以在没有认证的情况下任意访问。

1.5.2. 使用指定的登录页面

【路径】

1: 提供login.html作为项目的登录页面

2: 修改spring-security.xml文件，指定login.html页面可以匿名访问

3: 修改spring-security.xml文件，加入表单登录信息的配置

4: 修改spring-security.xml文件，关闭csrfFilter过滤器

【讲解】

第一步：提供login.html作为项目的登录页面

1: 用户名是abc

2: 密码是bbb

3: 登录的url是/login

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8 <form name='f' action='/login' method='POST'>
9   <table>
10     <tr><td>用户名:</td><td><input type='text' name='abc' value=''></td>
11   </tr>
12     <tr><td>密码:</td><td><input type='password' name='bbb' /></td></tr>
13     <tr><td colspan='2'><input name="submit" type="submit" value="登
14 陆"/></td></tr>
```

```

13     </table>
14 </form>
15 </body>
16 </html>

```

第二步：修改spring-security.xml文件，指定login.html页面可以匿名访问，否则无法访问。

```

1 <security:http security="none" pattern="/login.html" />

```


第三步：修改spring-security.xml文件，加入表单登录信息的配置

```

1 <!-- 登陆配置 form-login
2 login-page: 登陆页面
3 username-parameter: 前端传过来的用户名的参数名 与form表单中的用户名输入框中的name一致
4 password-parameter: 前端传过来的密码的参数名 与form表单中的密码输入框中的name一致
5 login-processing-url: 处理登陆请求的url, 必须与form表单中的action一致
6 default-target-url: 登陆成功后默认跳转的页面, success.html -> login.html -> success.html
7 always-use-default-target: true不管是从哪个页面转到login.html, 登陆后都跑到
8   success.html -> login.html -> index.html
9 authentication-failure-url: 登陆失败后跳转的页面
10 -->
11     <security:form-login
12         login-page="/login.html"
13         username-parameter="abc"
14         password-parameter="bbb"
15         login-processing-url="/login"
16         default-target-url="/index.html"
17         always-use-default-target="true"
18         authentication-failure-url="/fail.html"
19     ></security:form-login>

```

第四步：修改spring-security.xml文件，关闭CsrfFilter过滤器

 image-20200630170633682

1.5.2.1. 注意1:

如果用户名和密码输入不正确/正确。抛出异常:

 img

分析原因:

 img

Spring-security采用盗链机制，其中_csrf使用token标识和随机字符，每次访问页面都会随机生成，然后和服务器进行比较，成功可以访问，不成功不能访问（403：权限不足）。

解决方案:

```

1 <!--关闭盗链安全请求-->
2 <security:csrf disabled="true" />

```



1.5.2.2. 注意2:

1:创建test.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     我是test页面
9 </body>
10 </html>
```

2: 先访问test.html页面, 跳转到login.html



3: 再使用admin、admin登录, 会跳转到test.html, 怎么办?



分析原因:

登录成功后, 没有始终跳转到成功页面, 而是跳转到了之前访问的页面。

解决方案:

always-use-default-target="true"

```
1 <!--
2     登录页面配置
3     login-page: 登录页面
4     login-processing-url: 处理登录的地址
5     default-target-url: 登录成功后默认跳转地址
6     authentication-failure-url: 登录失败跳转地址
7     always-use-default-target="true": 登录成功后, 始终跳转到default-target-url指
    定的地址, 即登录成功的默认地址
8 -->
9 <security:form-login login-page="/login.html"
10     username-parameter="username"
11     password-parameter="password"
12     login-processing-url="/login.do"
13     default-target-url="/index.html"
14     authentication-failure-url="/login.html"
15     always-use-default-target="true"
16 />
```

1.5.3. 从数据库查询用户信息

【路径】

1: 定义UserService类, 实现UserDetailsService接口。

2: 修改spring-security.xml配置 (注册且注入UserService)

如果我们要从数据库动态查询用户信息，就必须按照spring security框架的要求提供一个实现UserDetailsService接口的实现类，并按照框架的要求进行配置即可。框架会自动调用实现类中的方法并自动进行密码校验。

第一步：定义UserService类，实现UserDetailsService接口。

实现类代码：

```
1 package com.itheima.service;
2
3 import com.itheima.health.pojo.Permission;
4 import com.itheima.health.pojo.Role;
5 import org.springframework.security.core.GrantedAuthority;
6 import org.springframework.security.core.authority.SimpleGrantedAuthority;
7 import org.springframework.security.core.userdetails.User;
8 import org.springframework.security.core.userdetails.UserDetails;
9 import org.springframework.security.core.userdetails.UserDetailsService;
10 import
    org.springframework.security.core.userdetails.UsernameNotFoundException;
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12
13 import java.util.ArrayList;
14 import java.util.HashSet;
15 import java.util.List;
16 import java.util.Set;
17
18 /**
19  * <p>
20  *
21  * </p>
22  *
23  * @author: Eric
24  * @since: 2020/10/31
25  */
26 public class UserService implements UserDetailsService {
27     /**
28      * 获取登陆用户信息
29      *
30      * @param username 从前端传过来的用户名
31      * @return 用户名，密码，权限集合
32      * @throws UsernameNotFoundException
33      */
34     @Override
35     public UserDetails loadUserByUsername(String username) throws
        UsernameNotFoundException {
36         // 假设从数据库查询，从数据库查询到的用户信息（用户名，密码，角色，权限）
37         com.itheima.health.pojo.User userInDb = findByUsername(username);
38         if(null != userInDb){
39             // 授权，userDetail是security需要的
40             //String username, 用户名
41             //String password, 密码，必须是从数据库查询到的密码
42             //Collection<? extends GrantedAuthority> authorities 用户的权限
            集合
43             List<GrantedAuthority> authorities = new
                ArrayList<GrantedAuthority>();
44             // 遍历用户的角色与权限
45             // 用户拥有的角色
46             Set<Role> roles = userInDb.getRoles();
```

```

47         if(null != roles){
48             GrantedAuthority sga = null;
49             for (Role role : roles) {
50                 // 角色名, 授予角色
51                 sga = new SimpleGrantedAuthority(role.getKeyword());
52                 authorities.add(sga);
53                 // 授予权限, 这个角色下所拥有的权限
54                 Set<Permission> permissions = role.getPermissions();
55                 if(null != permissions){
56                     for (Permission permission : permissions) {
57                         // 授予权限
58                         sga = new
SimpleGrantedAuthority(permission.getKeyword());
59                         authorities.add(sga);
60                     }
61                 }
62             }
63         }
64
65         User securityUser = new User(username, "
{noop}" + userInDb.getPassword(), authorities);
66         return securityUser;
67
68     }
69     return null;
70 }
71
72 /**
73  * 这个用户admin/admin, 有ROLE_ADMIN角色, 角色下有ADD_CHECKITEM权限
74  * 假设从数据库查询
75  * @param username
76  * @return
77  */
78 private com.itheima.health.pojo.User findByUsername (String username){
79     if("admin".equals(username)) {
80         com.itheima.health.pojo.User user = new
com.itheima.health.pojo.User();
81         user.setUsername("admin");
82         // 使用密文, 删除{noop}
83         user.setPassword("admin");
84
85         // 角色
86         Role role = new Role();
87         role.setKeyword("ROLE_ADMIN");
88
89         // 权限
90         Permission permission = new Permission();
91         permission.setKeyword("ADD_CHECKITEM");
92
93         // 给角色添加权限
94         role.getPermissions().add(permission);
95
96         // 把角色放进集合
97         Set<Role> roleList = new HashSet<Role>();
98         roleList.add(role);
99
100         role = new Role();
101         role.setKeyword("ABC");

```

```

102         roleList.add(role);
103
104         // 设置用户的角色
105         user.setRoles(roleList);
106         return user;
107     }
108     return null;
109 }
110
111 public static void main(String[] args) {
112     BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
113     // 加密密码
114     //System.out.println(encoder.encode("1234"));
115     // 校验密码，第1个参数为明文，第2个为密文
116     System.out.println(encoder.matches("1234",
117 "$2a$10$c2i8PHwnBtqMJlvKD7DsCuP9Kl4uQT4TIqBTgdaly/Pekp6Tb/4G0"));
118     System.out.println(encoder.matches("1234",
119 "$2a$10$IfpKav5WRkaaODODWPLU9uxQgt3qzfVUj1PxnzNPYiY.C7ycQRvAm"));
120     System.out.println(encoder.matches("1234",
121 "$2a$10$u/BcsUUqZNWUXdmDhbnoeeobjy6IBsL1Gn/S0dMxI2RbSgnMKJ.4a"));
122 }

```

第二步：spring-security.xml：

image-20201031164632828

本章节我们提供了UserService实现类，并且按照框架的要求实现了UserDetailsService接口。在spring配置文件中注册UserService，指定其作为认证过程中根据用户名查询用户信息的处理类。当我们进行登录操作时，spring security框架会调用UserService的loadUserByUsername方法查询用户信息，并根据此方法中提供的密码和用户页面输入的密码进行比对来实现认证操作。

1.5.4. 对密码进行加密

前面我们使用的密码都是明文的，这是非常不安全的。一般情况下用户的密码需要进行加密后再保存到数据库中。

常见的密码加密方式有：

3DES、AES、DES：使用对称加密算法，可以通过解密来还原出原始密码 可逆加密

MD5、SHA1：使用单向HASH算法，无法通过计算还原出原始密码，但是可以建立彩虹表进行查表破解

MD5可进行加盐加密，保证安全

```

1 public class TestMD5 {
2
3     @Test
4     public void testMD5(){
5         // 密码同样是1234却变成了密码不相同
6         System.out.println(MD5Utils.md5("1234xiaowang"));
7         //a8231077b3d5b40ffadee7f4c8f66cb7
8         System.out.println(MD5Utils.md5("1234xiaoli"));
9         //7d5250d8620fcd53b25f96a1c7be591
10    }
11 }

```

同样的密码值，盐值不同，加密的结果不同。

bcrypt: 将salt随机并混入最终加密后的密码，验证时也无需单独提供之前的salt，从而无需单独处理salt问题

spring security中的BCryptPasswordEncoder方法采用SHA-256 +随机盐+密钥对密码进行加密。SHA系列是Hash算法，不是加密算法，使用加密算法意味着可以解密（这个与编码/解码一样），但是采用Hash处理，其过程是不可逆的。

(1) 加密(encode): 注册用户时，使用SHA-256+随机盐+密钥把用户输入的密码进行hash处理，得到密码的hash值，然后将其存入数据库中。

(2) 密码匹配(matches): 用户登录时，密码匹配阶段并没有进行密码解密（因为密码经过Hash处理，是不可逆的），而是使用相同的算法把用户输入的密码进行hash处理，得到密码的hash值，然后将其与从数据库中查询到的密码hash值进行比较。如果两者相同，说明用户输入的密码正确。

这正是为什么处理密码时要用hash算法，而不用加密算法。因为这样处理即使数据库泄漏，黑客也很难破解密码。

建立测试代码

```

1 package com.itheima.security.test;
2
3 import org.junit.Test;
4 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
5
6 public class TestSpringSecurity {
7     // SpringSecurity加盐加密
8     @Test
9     public void testSpringSecurity(){
10         BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
11         // 加密密码
12         //System.out.println(encoder.encode("1234"));
13
14         // 验证密码
15         // 原密码
16         // 加密后的密码
17         System.out.println(encoder.matches("1234",
18 "$2a$10$P7Qx8eKUPX5lNgz9UEstU0aDRldEwrj9Rbyox/ShyeoxvPbEHTvni"));
19         System.out.println(encoder.matches("1234",
20 "$2a$10$5q.0a0F0hRix8TBJxQ4DB.ekwGzPs3e47hoQVNR7cihi/Rob.G3T6"));
21         System.out.println(encoder.matches("1234",
22 "$2a$10$voh.1PjRXqazoi jk72sIooslpMLYPyB.6Ltt7aUrXqUO5G8Aw43we"));
23
24         System.out.println(encoder.matches("1234",
25 "$2a$10$u/BcsUUqZNWUXdmDhbnoeobjy6IBsL1Gn/S0dMxI2RbSgnMKJ.4a"));
26

```

```
22     }
23 }
```

加密后的格式一般为：

```
1 | $2a$10$/bTvVqq1H9UiE0ZJZ7N2Me3RIgUCdgmheyTgV0B4cMCSokPa.6oCa
```

加密后字符串的长度为固定的60位。其中：

\$是分割符，无意义；

2a是bcrypt加密版本号；

10是cost的值；

而后的前22位是salt值；

再然后的字符串就是密码的密文了。

实现步骤：

【路径】

1：在spring-security.xml文件中指定密码加密对象

2：修改UserService实现类

【讲解】

第一步：在spring-security.xml文件中指定密码加密对象

```
1  <!--
2      三：认证管理，定义登录账号名和密码，并授予访问的角色、权限
3      authentication-manager：认证管理器，用于处理认证操作
4  -->
5  <security:authentication-manager>
6      <!--
7          authentication-provider：认证提供者，执行具体的认证逻辑
8      -->
9      <security:authentication-provider user-service-ref="userService">
10         <!--指定密码加密策略-->
11         <security:password-encoder ref="encoder"></security:password-
12 encoder>
13     </security:authentication-provider>
14 </security:authentication-manager>
15
16 <!--配置密码加密对象-->
17 <bean id="encoder"
    class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
```

第二步：修改UserService实现类

```
1  package com.itheima.service;
2
3  import com.itheima.health.pojo.Permission;
4  import com.itheima.health.pojo.Role;
5  import org.springframework.security.core.GrantedAuthority;
6  import org.springframework.security.core.authority.SimpleGrantedAuthority;
```



```

7  import org.springframework.security.core.userdetails.User;
8  import org.springframework.security.core.userdetails.UserDetails;
9  import org.springframework.security.core.userdetails.UserDetailsService;
10 import
    org.springframework.security.core.userdetails.UsernameNotFoundException;
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12
13 import java.util.ArrayList;
14 import java.util.HashSet;
15 import java.util.List;
16 import java.util.Set;
17
18 /**
19  * <p>
20  *
21  * </p>
22  *
23  * @author: Eric
24  * @since: 2020/10/31
25  */
26 public class UserService implements UserDetailsService {
27     /**
28      * 获取登陆用户信息
29      *
30      * @param username 从前端传过来的用户名
31      * @return 用户名, 密码, 权限集合
32      * @throws UsernameNotFoundException
33      */
34     @Override
35     public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
36         // 假设从数据库查询, 从数据库查询到的用户信息 (用户名, 密码, 角色, 权限)
37         com.itheima.health.pojo.User userInDb = findByUsername(username);
38         if(null != userInDb){
39             // 授权, userDetails是security需要的
40             //String username, 用户名
41             //String password, 密码, 必须是从数据库查询到的密码
42             //Collection<? extends GrantedAuthority> authorities 用户的权限
集合
43             List<GrantedAuthority> authorities = new
ArrayList<GrantedAuthority>();
44             // 遍历用户的角色与权限
45             // 用户拥有的角色
46             Set<Role> roles = userInDb.getRoles();
47             if(null != roles){
48                 GrantedAuthority sga = null;
49                 for (Role role : roles) {
50                     // 角色名, 授予角色
51                     sga = new SimpleGrantedAuthority(role.getKeyword());
52                     authorities.add(sga);
53                     // 授予权限, 这个角色下所拥有的权限
54                     Set<Permission> permissions = role.getPermissions();
55                     if(null != permissions){
56                         for (Permission permission : permissions) {
57                             // 授予权限
58                             sga = new
SimpleGrantedAuthority(permission.getKeyword());
59                             authorities.add(sga);

```

```

60         }
61     }
62 }
63 }
64 // 去除{noop}
65 User securityUser = new User(username,
userInDb.getPassword(),authorities);
66     return securityUser;
67 }
68 }
69 return null;
70 }
71
72 /**
73  * 这个用户admin/admin，有ROLE_ADMIN角色，角色下有ADD_CHECKITEM权限
74  * 假设从数据库查询
75  * @param username
76  * @return
77  */
78 private com.itheima.health.pojo.User findByUsername (String username){
79     if("admin".equals(username)) {
80         com.itheima.health.pojo.User user = new
com.itheima.health.pojo.User();
81         user.setUsername("admin");
82         // 使用密文，删除{noop}
83
84         user.setPassword("$2a$10$IfpKav5WrkaaODODWPLU9uxQgt3qzfVUj1PxnzNPYiY.C7yc
QRvAm");
85
86         // 角色
87         Role role = new Role();
88         role.setKeyword("ROLE_ADMIN");
89
90         // 权限
91         Permission permission = new Permission();
92         permission.setKeyword("ADD_CHECKITEM");
93
94         // 给角色添加权限
95         role.getPermissions().add(permission);
96
97         // 把角色放进集合
98         Set<Role> roleList = new HashSet<Role>();
99         roleList.add(role);
100
101         role = new Role();
102         role.setKeyword("ABC");
103         roleList.add(role);
104
105         // 设置用户的角色
106         user.setRoles(roleList);
107         return user;
108     }
109     return null;
110 }
111
112 public static void main(String[] args) {
113     BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
114     // 加密密码

```

```

114         //System.out.println(encoder.encode("1234"));
115         // 校验密码，第1个参数为明文，第2个为密文
116         System.out.println(encoder.matches("1234",
"$2a$10$c2I8PHwnBtqMJlVKD7DsCuP9Kl4uQT4TIqBTgdaly/Pekp6Tb/4Go"));
117         System.out.println(encoder.matches("1234",
"$2a$10$IfpKav5WRkaaODODWPLU9uxQgt3qzfVUj1PxnzNPYiY.C7ycQRvAm"));
118         System.out.println(encoder.matches("1234",
"$2a$10$u/BcsUUqZNWUxdmDhbnoeobJy6IBsL1Gn/S0dMxI2RbSgnMKJ.4a"));
119
120     }
121 }
122

```

使用密码加密器小结:

1. security.xml添加bean 加密器

```

1  <!--定义使用的加密器，做密码校验-->
2  <bean id="encoder"
    class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"
    />

```

2. security.xml使用加密器

```

1  <security:authentication-provider user-service-ref="userService">
2      <!-- 引用密码加密器，用于登陆时密码的校验 -->
3      <security:password-encoder ref="encoder"/>
4  </security:authentication-provider>

```

3. 修改UserService中admin用户的密码为密文, 去{noop}

1.5.5. 配置多种验证规则（对页面）

为了测试方便，首先在项目中创建a.html、b.html、c.html、d.html几个页面

修改spring-security.xml文件：

【前提】：<security:http auto-config="true" use-expressions="true">，开启对表达式的支持

```

1  <!-- use-expressions="true" 就不能使用这种了access=ROLE_ADMIN了，要把它注释掉，否则启动会报错
2  <security:intercept-url pattern="/*" access="ROLE_ADMIN"/>
3  -->
4  <!--只要认证通过就可以访问-->
5  <security:intercept-url pattern="/index.html" access="isAuthenticated()" />
6  <security:intercept-url pattern="/a.html" access="isAuthenticated()" />
7
8  <!--拥有add权限就可以访问b.html页面-->
9  <security:intercept-url pattern="/b.html" access="hasAuthority('add')" />
10
11 <!--拥有ROLE_ADMIN角色就可以访问c.html页面，
12     注意：此处虽然写的是ADMIN角色，框架会自动加上前缀ROLE_-->
13 <security:intercept-url pattern="/c.html" access="hasRole('ADMIN')" />

```

```

14
15 <!--拥有ROLE_ADMIN角色就可以访问d.html页面-->
16 <security:intercept-url pattern="/d.html" access="hasRole('ABC')" />

```

测试:

登录后可以访问a.html,b.html,c.html, 不能访问d.html (抛出403的异常)

hasAuthority不需要ROLE, *hasRole就要ROLE*

1.5.6. 注解方式权限控制 (对类)

Spring Security除了可以在配置文件中配置权限校验规则, 还可以使用注解方式控制类中方法的调用。例如Controller中的某个方法要求必须具有某个权限才可以访问, 此时就可以使用Spring Security框架提供的注解方式进行控制。

【路径】

- 1: 在spring-security.xml文件中配置组件扫描和mvc的注解驱动, 用于扫描Controller
- 2: 在spring-security.xml文件中开启权限注解支持
- 3: 创建Controller类并在Controller的方法上加入注解 (@PreAuthorize) 进行权限控制

实现步骤:

第一步: 在spring-security.xml文件中配置组件扫描, 用于扫描Controller

```

1 <context:component-scan base-package="com.itheima"/>
2 <mvc:annotation-driven></mvc:annotation-driven>

```

第二步: 在spring-security.xml文件中开启权限注解支持

```

1 <!--开启注解方式权限控制-->
2 <security:global-method-security pre-post-annotations="enabled" />

```

第三步: 创建Controller类并在Controller的方法上加入注解 (@PreAuthorize) 进行权限控制

```

1 package com.itheima.controller;
2
3 import org.springframework.security.access.prepost.PreAuthorize;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 @Controller
8 @RequestMapping("/hello")
9 public class HelloController {
10
11     @RequestMapping("/add")
12     @PreAuthorize("hasAuthority('add')")//表示用户必须拥有add权限才能调用当前方法
13     public String add(){
14         System.out.println("add...");
15         return null;
16     }
17
18     @RequestMapping("/update")
19     @PreAuthorize("hasRole('ROLE_ADMIN')")//表示用户必须拥有ROLE_ADMIN角色才能调用当前方法

```

```

20     public String update(){
21         System.out.println("update...");
22         return null;
23     }
24
25     @RequestMapping("/delete")
26     @PreAuthorize("hasRole('ABC')")//表示用户必须拥有ABC角色才能调用当前方法
27     public String delete(){
28         System.out.println("delete...");
29         return null;
30     }
31 }

```

测试delete方法不能访问



小结:

1. 开启注解支持

```

1 | <security:global-method-security pre-post-annotations="enabled"/>

```

2. 使用mvc:annotation-driven
3. 创建controller, 且加入扫包context:component-scan
4. controller方法@PreAuthorize(hasAuthority/hasRole)
5. use-expressions="true" 要注解掉

```

1 | <!--<security:intercept-url pattern="/**"
   |    access="hasRole('ROLE_ADMIN')"/>-->

```

URL拦截权限配置, 粗颗粒

@PreAuthrize 细颗粒 (具体的方法) @PreAuthrize用controller类上

如果类有PreAuthrize, 方法上也有PreAuthrize, 两个条件都满足后才可以访问

推荐: 使用角色来做权限控制

1.5.7. 退出登录

用户完成登录后Spring Security框架会记录当前用户认证状态为已认证状态, 即表示用户登录成功了。那用户如何退出登录呢? 我们可以在spring-security.xml文件中进行如下配置:

【路径】

- 1: index.html定义退出登录链接
- 2: 在spring-security.xml定义

【讲解】

第一步: index.html定义退出登录链接

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     登录成功! <br>
9     <a href="/logout">退出登录</a>
10 </body>
11 </html>

```

第二步：在spring-security.xml定义：

```

1 <!--
2     logout: 退出登录
3     logout-url: 退出登录操作对应的请求路径
4     logout-success-url: 退出登录后的跳转页面
5 -->
6 <security:logout logout-url="/logout"
7                 logout-success-url="/login.html" invalidate-session="true"/>

```

通过上面的配置可以发现，如果用户要退出登录，只需要请求/logout.do这个URL地址就可以，同时会将当前session失效，最后页面会跳转到login.html页面。

【小结】

1：配置可匿名访问的资源(不需要登录,权限 角色 就可以访问)

```

1 <security:http security="none" pattern="/js/**"></security:http>
2 <security:http security="none" pattern="/css/**"></security:http>
3 <security:http security="none" pattern="/login.html"></security:http>

```

2：使用指定的登录页面 (login.html)

```

1 <security:form-login login-page="/login.html"
2                     username-parameter="username"
3                     password-parameter="password"
4                     login-processing-url="/login"
5                     default-target-url="/index.html"
6                     authentication-failure-url="/login.html"
7                     always-use-default-target="true"/>
8 <!--      关闭跨域访问限制      -->
9 <security:csrf disabled="true"/>

```

3：从数据库查询用户信息

添加类实现UserDetailsService接口，实现loadByUsername方法，且要返回UserDetails对象（用户名，数据库中的密码，用户所拥有的权限集合）

```

1 <security:authentication-manager>
2     <security:authentication-provider user-service-ref="userService">
3         <security:password-encoder ref="encoder"></security:password-encoder>
4     </security:authentication-provider>
5 </security:authentication-manager>

```

4: 对密码进行加密

```

1 <bean id="encoder"
  class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder">
  </bean>
2
3 userDetails中密码不再用{noop},把{noop}删除

```

5: 配置多种校验规则 (对访问的页面做权限控制, 多个时, 从上往按顺序, 只要有一个满足, 就会处理)

```

1 use-expressions="true"
2 <security:intercept-url pattern="/index.html" access="isAuthenticated()" />
3 <security:intercept-url pattern="/a.html" access="isAuthenticated()" />
4 <security:intercept-url pattern="/b.html" access="hasAuthority('add')" />
5 <security:intercept-url pattern="/c.html" access="hasRole('ROLE_ADMIN')" />
6 <security:intercept-url pattern="/d.html" access="hasRole('ABC')" />

```

6: 注解方式权限控制 (对访问的Controller类做权限控制)

```

1 <security:global-method-security pre-post-annotations="enabled" />

```

同时使用注解:

在Controller类中的方法上添加: @PreAuthorize(value = "hasRole('ROLE_ADMIN')")

7: 退出登录

```

1 <security:logout logout-url="/logout" logout-success-url="/login.html"
  invalidate-session="true" />

```

2. 项目中使用Spring Security 【重点】

前面我们已经学习了Spring Security框架的使用方法, 本章节我们就需要将Spring Security框架应用到后台系统中进行权限控制, 其本质就是认证和授权。

要进行认证和授权需要前面课程中提到的权限模型涉及的7张表支撑, 因为用户信息、权限信息、菜单信息、角色信息、关联信息等都保存在这7张表中, 也就是这些表中的数据是我们进行认证和授权的依据。所以在真正进行认证和授权之前需要对这些数据进行管理, 即我们需要开发如下一些功能:

- 1、用户数据管理 (增删改查、用户关联角色)
- 2、角色数据管理 (增删改查、角色关联权限、角色关联菜单)
- 3、权限数据管理 (增删改查)
- 4、菜单数据管理 (增删改查)

鉴于时间关系，我们不再实现这些数据管理的代码开发。我们可以直接将数据导入到数据库中即可。

导入用户、角色、权限、菜单的初始数据

在项目第一天时就已经导入了过了，可忽略



【目标】

在传智健康的项目中使用SpringSecurity完成认证和授权

【路径】

1: 导入SpringSecurity环境

- (1) pom.xml中添加依赖
- (2) health_web web.xml添加代理过滤器

2: 实现认证和授权

- (1) 导入login.html,放入health_web工程的webapp目录下
 - (2) 认证: SpringSecurityUserService.java
 - (3) 创建UserService类、UserDao接口类、UserDao映射文件
 - (4) springmvc.xml (dubbo注解扫描范围扩大)
 - (5) spring-security.xml
 - 静态资源
 - 拦截规则，所有都必须登陆后才可访问
 - 登陆页面配置
 - 关闭csrf
 - frame访问策略
 - 退出登陆
 - 开启注解权限控制
 - 认证管理器
 - 认证信息提供者
 - 加密器
 - (6) springmvc.xml (导入spring-security.xml)
 - (7) CheckItemController类 (@PreAuthorize("hasAuthority('CHECKITEM_ADD')"): 完成权限)
 - (8) 捕获异常
- 3: 显示用户名
- 4: 用户退出

【讲解】

2.1. 导入Spring Security环境

【路径】

1: pom.xml导入坐标

2: web.xml添加代理过滤器

2.1.1. 第一步：pom.xml导入坐标

在health_parent父工程的pom.xml中导入Spring Security的依赖版本管理。注意：如果已经引入就不要重复引了

```
1  <!-- 依赖版本管理标签-->
2  <dependencyManagement>
3      <dependencies>
4          <dependency>
5              <groupId>org.springframework.security</groupId>
6              <artifactId>spring-security-web</artifactId>
7              <version>${spring.security.version}</version>
8          </dependency>
9          <dependency>
10             <groupId>org.springframework.security</groupId>
11             <artifactId>spring-security-config</artifactId>
12             <version>${spring.security.version}</version>
13         </dependency>
14         ...
15     </dependencies>
16 </dependencyManagement>
```

在health_common工程中引入security的依赖（已经引入）。注意：如果已经引入就不要重复引了

```
1  <dependencies>
2      <!--springSecurity-->
3      <dependency>
4          <groupId>org.springframework.security</groupId>
5          <artifactId>spring-security-web</artifactId>
6      </dependency>
7      <dependency>
8          <groupId>org.springframework.security</groupId>
9          <artifactId>spring-security-config</artifactId>
10     </dependency>
11     <dependency>
12         <groupId>org.springframework.security</groupId>
13         <artifactId>spring-security-taglibs</artifactId>
14     </dependency>
15     ...
16 </dependencies>
```

2.1.2. 第二步：web.xml添加代理过滤器

在health_web工程的web.xml文件中配置用于整合Spring Security框架的过滤器DelegatingFilterProxy

```

1 <filter>
2     <!--
3         DelegatingFilterProxy用于整合第三方框架（代理过滤器，非真正的过滤器，真正的过滤器需要在spring的配置文件）
4         整合Spring Security时过滤器的名称必须为springSecurityFilterChain,
5         否则会抛出NoSuchBeanDefinitionException异常
6     -->
7     <filter-name>springSecurityFilterChain</filter-name>
8     <filter-
9 class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
9 </filter>
10 <filter-mapping>
11     <filter-name>springSecurityFilterChain</filter-name>
12     <url-pattern>/*</url-pattern>
13 </filter-mapping>

```

2.2. 实现认证和授权

2.2.1. 第一步：导入login.html页面



此时login.html是可放行的页面，而pages下的页面必须认证之后才能访问的页面

2.2.2. 第二步：SpringSecurityUserService.java

在health_web工程中按照Spring Security框架要求提供SpringSecurityUserService，并且实现UserDetailsService接口

```

1 package com.itheima.health.security;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.pojo.Permission;
5 import com.itheima.health.pojo.Role;
6 import com.itheima.health.pojo.User;
7 import com.itheima.health.service.UserService;
8 import org.springframework.security.core.GrantedAuthority;
9 import org.springframework.security.core.authority.SimpleGrantedAuthority;
10 import org.springframework.security.core.userdetails.UserDetails;
11 import org.springframework.security.core.userdetails.UserDetailsService;
12 import
13 org.springframework.security.core.userdetails.UsernameNotFoundException;
14 import org.springframework.stereotype.Component;
15
16 import java.util.ArrayList;
17 import java.util.List;
18 import java.util.Set;
19
20 /**
21  * Description: 登陆用户认证与授权
22  * 记得要把这个类注册到spring容器
23  * User: Eric
24  */
25 @Component

```

```

25 public class SpringSecurityUserService implements UserDetailsService {
26
27     @Reference
28     private UserService userService;
29
30     /**
31      * 提供登陆用户信息  username password 权限集合 authorities
32      * @param username
33      * @return
34      * @throws UsernameNotFoundException
35      */
36     @Override
37     public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
38         //根据登陆用户名称查询用户权限信息
39         //t_user -> t_user_role -> t_role -> t_role_permission ->
t_permission
40         //找出用户所拥有的角色，及角色下所拥有的权限集合
41         //User.roles(角色集合).permissions(权限集合)
42         User user = userService.findByUsername(username);
43         if(null != user){
44
45             // 用户名
46             // 密码
47             String password = user.getPassword();
48             // 权限集合
49             List<GrantedAuthority> authorities = new
ArrayList<GrantedAuthority>();
50             // 授权
51
52             // 用户所拥有的角色
53             SimpleGrantedAuthority sai = null;
54             Set<Role> roles = user.getRoles();
55             if(null != roles){
56                 for (Role role : roles) {
57                     // 角色用关键字，授予角色
58                     sai = new SimpleGrantedAuthority(role.getKeyword());
59                     authorities.add(sai);
60                     // 权限，角色下的所有权限
61                     Set<Permission> permissions = role.getPermissions();
62                     if(null != permissions){
63                         for (Permission permission : permissions) {
64                             // 授予权限
65                             sai = new
SimpleGrantedAuthority(permission.getKeyword());
66                             authorities.add(sai);
67                         }
68                     }
69                 }
70             }
71
72             return new
org.springframework.security.core.userdetails.User(username,password,authori
ties);
73         }
74         // 返回null，限制访问
75         return null;
76     }

```

```
77 | }  
78 |
```

使用debug跟踪调试，查看user。



2.2.3. 第三步：Service、Dao接口、Mapper映射文件

创建UserService服务接口、服务实现类、Dao接口、Mapper映射文件等

【路径】

- 1: UserService.java接口
- 2: UserServiceImpl.java类
- 3: UserDao.java（使用用户名称查询用户）
- 4: RoleDao.java（使用用户id查询角色集合）
- 5: PermissionDao.java（使用角色id查询权限集合）
- 6: UserDao.xml（使用用户名称查询用户）
- 7: RoleDao.xml（使用用户id查询角色集合）
- 8: PermissionDao.xml（使用角色id查询权限集合）

【讲解】

- 1: 服务接口

```
1 package com.itheima.health.service;  
2  
3 import com.itheima.health.pojo.User;  
4  
5 /**  
6  * Description: 用户服务(企业员工)  
7  * User: Eric  
8  */  
9 public interface UserService {  
10     /**  
11      * 根据登陆用户名称查询用户权限信息  
12      * @param username  
13      * @return  
14      */  
15     User findByUsername(String username);  
16 }  
17
```

- 2: 服务实现类

```
1 package com.itheima.health.service.impl;  
2  
3 import com.alibaba.dubbo.config.annotation.Service;  
4 import com.itheima.health.dao.UserDao;
```

```

5  import com.itheima.health.pojo.User;
6  import com.itheima.health.service.UserService;
7  import org.springframework.beans.factory.annotation.Autowired;
8
9  /**
10   * Description: No Description
11   * User: Eric
12   */
13  @Service(interfaceClass = UserService.class)
14  public class UserServiceImpl implements UserService {
15
16      @Autowired
17      private UserDao userDao;
18
19      /**
20       * 根据登陆用户名称查询用户权限信息
21       * @param username
22       * @return
23       */
24      @Override
25      public User findByUsername(String username) {
26          return userDao.findByUsername(username);
27      }
28  }
29

```

3: Dao接口

(1) UserDao

```

1  package com.itheima.health.dao;
2
3  import com.itheima.health.pojo.User;
4
5  /**
6   * Description: No Description
7   * User: Eric
8   */
9  public interface UserDao {
10
11      /**
12       * 根据登陆用户名称查询用户权限信息
13       * @param username
14       * @return
15       */
16      User findByUsername(String username);
17  }
18

```

4: Mapper映射文件

(1) UserDao.xml

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

```

```

4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5
6      <mapper namespace="com.itheima.health.dao.UserDao">
7          <select id="findByUsername" parameterType="string"
8              resultMap="userRolePermissionMap">
9              select u.id,u.username,u.password,
10                  ur.role_id, r.keyword role_keyword, r.name role_name,
11                  rp.permission_id, p.keyword permission_keyword, p.name
12                  permission_name
13              From
14                  t_user u, t_user_role ur, t_role r,
15                  t_role_permission rp, t_permission p
16              where u.id=ur.user_id and ur.role_id=r.id
17                  and r.id=rp.role_id and rp.permission_id=p.id
18                  and u.username=#{username}
19          </select>
20
21          <resultMap id="userRolePermissionMap" type="User">
22              <id property="id" column="id"></id>
23              <result property="username" column="username"/>
24              <result property="password" column="password"/>
25              <collection property="roles" ofType="Role">
26                  <id property="id" column="role_id"/>
27                  <result property="keyword" column="role_keyword"/>
28                  <result property="name" column="role_name"/>
29                  <collection property="permissions" ofType="Permission">
30                      <id property="id" column="permission_id"/>
31                      <result property="keyword" column="permission_keyword"/>
32                      <result property="name" column="permission_name"/>
33                  </collection>
34              </collection>
35          </resultMap>
36      </mapper>

```

2.2.4. 第四步：springmvc.xml

修改health_web工程中的springmvc.xml文件，修改dubbo批量扫描的包路径

之前的包扫描

```

1  <!--批量扫描-->
2  <dubbo:annotation package="com.itheima.health.controller" />

```

现在的包扫描

```

1  <!--批量扫描-->
2  <dubbo:annotation package="com.itheima.health" />

```

注意：此处原来扫描的包为com.itheima.controller，现在改为com.itheima包的目的是需要将我们上面定义的SpringSecurityUserService也扫描到，因为在SpringSecurityUserService的loadUserByUsername方法中需要通过dubbo远程调用名称为UserService的服务。

2.2.5. 第五步：spring-security.xml

【路径】

- 1: 定义哪些链接可以放行
- 2: 定义哪些链接不可以放行, 即需要有角色、权限才可以放行
- 3: 认证管理, 定义登录账号名和密码, 并授予访问的角色、权限
- 4: 设置在页面可以通过iframe访问受保护的页面, 默认为不允许访问, 需要添加security:frame-options policy="SAMEORIGIN"

【讲解】

在health_web工程中提供spring-security.xml配置文件

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:security="http://www.springframework.org/schema/security"
5       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">
6 <!-- 静态资源(css, img, js..., login.html)-->
7     <security:http pattern="/css/**" security="none"/>
8     <security:http pattern="/img/**" security="none"/>
9     <security:http pattern="/js/**" security="none"/>
10    <security:http pattern="/plugins/**" security="none"/>
11    <security:http pattern="/login.html" security="none"/>
12
13    <!--
14    拦截规则
15        autoconfig userexpress
16        interceptor-url pattern access
17        登陆页面
18        退出登陆
19        关闭csrf
20    -->
21    <security:http auto-config="true" use-expressions="true">
22        <security:intercept-url pattern="/**" access="isAuthenticated()"/>
23        <security:form-login login-page="/login.html"
24                            login-processing-url="/login.do"
25                            username-parameter="username"
26                            password-parameter="password"
27                            default-target-url="/pages/main.html"
28                            always-use-default-target="true"/>
29        <security:headers>
30 <!-- frame-options 控制页面中嵌套frame (访问其它页面, 把其它页面的内容展示在这个页面
上)
31            policy 使用的策略:
32                DENY: 不允许访问
33                SAMEORIGIN: 同域可以访问
34                ALLOW-FROM: 指定url可以访问
35        -->
36        <security:frame-options policy="SAMEORIGIN"/>
37    </security:headers>
38    <security:csrf disabled="true"/>
39
```

```

40     <security:logout logout-url="/logout.do" logout-success-
url="/login.html" invalidate-session="true"/>
41     </security:http>
42     <!--
43     认证信息
44     认证管理器
45     提供者 user-service-ref springSecurityUserService implements
UserDetailsService
46     配置加密器
47     -->
48     <security:authentication-manager>
49     <security:authentication-provider user-service-
ref="springSecurityUserService">
50     <security:password-encoder ref="encoder"/>
51     </security:authentication-provider>
52     </security:authentication-manager>
53     <!--注册springSecurityUserService
54     注册密码加密器-->
55     <!--<bean id="springSecurityUserService"
class="com.itheima.health.security.SpringSecurityUserService"/>-->
56     <bean id="encoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"/>
57
58     <!-- 权限控制注解支持 -->
59     <security:global-method-security pre-post-annotations="enabled"/>
60 </beans>

```

这里注意：如果出现以下问题



使用下面的配置，在spring-security.xml中添加。

放置到<security:http auto-config="true" use-expressions="true">里面

```

1 <security:headers>
2     <!--设置在页面可以通过iframe访问受保护的页面，默认为不允许访问-->
3     <security:frame-options policy="SAMEORIGIN"></security:frame-options>
4 </security:headers>

```

是因为我们在main.html中定义：如果不配置springSecurity会认为iframe访问的html页面是受保护的页面，不允许访问。

```

1 <el-container>
2     <iframe name="right" class="el-main" src="checkitem.html" width="100%"
height="580px" frameborder="0"></iframe>
3 </el-container>

```

备注：



2.2.6. 第六步：springmvc.xml

在springmvc.xml文件中引入spring-security.xml文件


```
1 | <import resource="classpath:spring-security.xml"></import>
```

2.2.7. 第七步：CheckItemController类

在Controller的方法上加入权限控制注解，此处以CheckItemController为例

```
1 package com.itheima.health.controller;
2
3 import com.alibaba.dubbo.config.annotation.Reference;
4 import com.itheima.health.constant.MessageConstant;
5 import com.itheima.health.entity.PageResult;
6 import com.itheima.health.entity.QueryPageBean;
7 import com.itheima.health.entity.Result;
8 import com.itheima.health.pojo.CheckItem;
9 import com.itheima.health.service.CheckItemService;
10 import org.springframework.security.access.prepost.PreAuthorize;
11 import org.springframework.web.bind.annotation.*;
12
13 import java.util.List;
14
15 /**
16  * Description: No Description
17  * User: Eric
18  */
19 @RestController
20 @RequestMapping("/checkitem")
21 public class CheckItemController {
22
23     @Reference
24     private CheckItemService checkItemService;
25
26     @GetMapping("/findAll")
27     public Result findAll(){
28         // 调用服务查询所有的检查项
29         List<CheckItem> list = checkItemService.findAll();
30         // 封装返回的结果
31         return new Result(true,
32             MessageConstant.QUERY_CHECKITEM_SUCCESS, list);
33     }
34
35     /**
36      * 新增检查项
37      * @param checkitem
38      * @return
39      */
40     @PostMapping("/add")
41     @PreAuthorize("hasAuthority('CHECKITEM_ADD')")
42     public Result add(@RequestBody CheckItem checkitem){
43         // 调用业务服务
44         checkItemService.add(checkitem);
45         // 响应结果给前端
46         return new Result(true, MessageConstant.ADD_CHECKITEM_SUCCESS);
47     }
48 }
```

```

48     /**
49      * 分页查询
50      */
51     @PostMapping("/findPage")
52     @PreAuthorize("hasAuthority('CHECKITEM_QUERY')")
53     public Result findPage(@RequestBody QueryPageBean queryPageBean){
54         // 调用业务来分页
55         PageResult<CheckItem> pageResult =
checkItemService.findPage(queryPageBean);
56
57         //return pageResult;
58         // 返回给页面，包装到Result，统一风格
59         return new Result(true,
MessageConstant.QUERY_CHECKITEM_SUCCESS,pageResult);
60     }
61
62     /**
63      * 删除
64      */
65     @PostMapping("/deleteById")
66     public Result deleteById(int id){
67         // 调用业务服务
68         //try {
69             checkItemService.deleteById(id);
70         //} catch (Exception e) {
71             // e.printStackTrace();
72         //}
73         // 响应结果
74         return new Result(true, MessageConstant.DELETE_CHECKITEM_SUCCESS);
75     }
76
77     /**
78      * 通过id查询
79      */
80     @GetMapping("/findById")
81     public Result findById(int id){
82         CheckItem checkItem = checkItemService.findById(id);
83         return new Result(true,
MessageConstant.QUERY_CHECKITEM_SUCCESS,checkItem);
84     }
85
86     /**
87      * 修改检查项
88      * @param checkitem
89      * @return
90      */
91     @PostMapping("/update")
92     public Result update(@RequestBody CheckItem checkitem){
93         // 调用业务服务
94         checkItemService.update(checkitem);
95         // 响应结果给前端
96         return new Result(true, MessageConstant.EDIT_CHECKITEM_SUCCESS);
97     }
98 }
99

```

2.2.8. 第八步：异常捕获

修改health_web项目中的HealthExceptionAdvice

```
1 package com.itheima.health.controller;
2
3 import com.itheima.health.entity.Result;
4 import com.itheima.health.exception.HealthException;
5 import org.springframework.security.access.AccessDeniedException;
6 import org.springframework.security.authentication.BadCredentialsException;
7 import
org.springframework.security.authentication.InternalAuthenticationServiceExc
ception;
8 import org.springframework.web.bind.annotation.ExceptionHandler;
9 import org.springframework.web.bind.annotation.RestControllerAdvice;
10
11 /**
12  * Description: No Description
13  * User: Eric
14  */
15 // 与前端约定好的，返回的都是json数据
16 @RestControllerAdvice
17 public class HealExceptionAdvice {
18
19     /**
20      * 自定义招出的异常处理
21      * @param he
22      * @return
23      */
24     @ExceptionHandler(HealthException.class)
25     public Result handleHealthException(HealthException he){
26         return new Result(false, he.getMessage());
27     }
28
29     /**
30      * 所有未知的异常
31      * @param he
32      * @return
33      */
34     @ExceptionHandler(Exception.class)
35     public Result handleException(Exception he){
36         he.printStackTrace();
37         return new Result(false, "发生未知错误，操作失败，请联系管理员");
38     }
39
40     /**
41      * 密码错误
42      * @param he
43      * @return
44      */
45     @ExceptionHandler(BadCredentialsException.class)
46     public Result handBadCredentialsException(BadCredentialsException he){
47         return handleUserPassword();
48     }
49
50     /**
51      * 用户名不存在
```

```

52     * @param he
53     * @return
54     */
55     @ExceptionHandler({InternalAuthenticationServiceException.class})
56     public Result
57     handleInternalAuthenticationServiceException(InternalAuthenticationServiceExce
58     ption he){
59
60         return handleUserPassword();
61     }
62
63     private Result handleUserPassword(){
64         return new Result(false, "用户名或密码错误");
65     }
66
67     /**
68     * 用户名不存在
69     * @param he
70     * @return
71     */
72     @ExceptionHandler({AccessDeniedException.class})
73     public Result handleAccessDeniedException(AccessDeniedException he){
74         return new Result(false, "没有权限");
75     }
76 }

```

2.3. 显示用户名

【路径】

- 1: 引入js
- 2: 定义loginUsername属性
- 3: 使用钩子函数，调用ajax，查询登录用户（从SpringSecurity中获取），赋值username属性
- 4: 修改页面，使用{{loginUsername}}显示用户信息

【讲解】

前面我们已经完成了认证和授权操作，如果用户认证成功后需要在页面展示当前用户的用户名。Spring Security在认证成功后会将用户信息保存到框架提供的上下文对象中，所以此处我们就可以调用Spring Security框架提供的API获取当前用户的username并展示到页面上。

实现步骤：

第一步：在main.html页面中修改，定义username模型数据基于VUE的数据绑定展示用户名，发送ajax请求获取username

(1)：引入js

```
1 <script src="../js/axios-0.18.0.js"></script>
```

(2)：定义loginUsername属性



(3)：使用钩子函数mounted，调用ajax



(4) 显示用户名



页面最终如下

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <!-- 页面meta -->
5     <meta charset="utf-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <title>传智健康</title>
8     <meta name="description" content="传智健康">
9     <meta name="keywords" content="传智健康">
10    <meta content="width=device-width,initial-scale=1,maximum-
scale=1,user-scalable=no" name="viewport">
11    <!-- 引入样式 -->
12    <link rel="stylesheet" href="../plugins/elementui/index.css">
13    <link rel="stylesheet" href="../plugins/font-awesome/css/font-
awesome.min.css">
14    <link rel="stylesheet" href="../css/style.css">
15    <style type="text/css">
16    .el-main{
17        position: absolute;
18        top: 70px;
19        bottom: 0px;
20        left: 200px;
21        right: 10px;
22        padding: 0;
23    }
24    </style>
25 </head>
26 <body class="hold-transition skin-purple sidebar-mini">
27     <div id="app">
28         <el-container>
29             <el-header class="main-header" style="height:70px;">
30                 <nav class="navbar navbar-static-top" :class=''>
31                     <!-- Logo -->
32                     <a href="#" class="logo" style="text-align:center">
33                         <span class="logo-lg">
34                     </a>
35                     <div class="right-menu">
36                         <span class="help"><i class="fa fa-exclamation-
circle" aria-hidden="true"></i>帮助</span>
37                         <el-dropdown class="avatar-container right-menu-
item" trigger="click">
38                             <div class="avatar-wrapper">
39                                 
40                                 {{loginUsername}}
41                             </div>
42                             <el-dropdown-menu slot="dropdown">
```

```

43         <el-dropdown-item divided>
44             <span style="display:block;">修改密码
45         </span>
46     </el-dropdown-item>
47     <el-dropdown-item divided>
48         <span style="display:block;"><a
49 href="/logout.do">退出</a> </span>
50     </el-dropdown-item>
51 </el-dropdown-menu>
52 </el-dropdown>
53 </div>
54 </nav>
55 </el-header>
56 <el-container>
57     <el-aside width="200px">
58         <el-menu>
59             <el-submenu v-for="menu in menuList"
60 :index="menu.path">
61                 <template slot="title">
62                     <i class="fa" :class="menu.icon"></i>
63                     {{menu.title}}
64                 </template>
65                 <template v-for="child in menu.children">
66                     <el-menu-item :index="child.path">
67                         <a :href="child.linkurl"
68 target="right">{{child.title}}</a>
69                     </el-menu-item>
70                 </template>
71             </el-submenu>
72         </el-menu>
73     </el-aside>
74     <el-container>
75         <iframe name="right" class="el-main"
76 src="ordersetting.html" width="100%" height="580px" frameborder="0">
77     </iframe>
78 </el-container>
79 </el-container>
80 </div>
81 </body>
82 <!-- 引入组件库 -->
83 <script src="../js/vue.js"></script>
84 <script src="../plugins/elementui/index.js"></script>
85 <script type="text/javascript" src="../js/jquery.min.js"></script>
86 <script src="../js/axios-0.18.0.js"></script>
87 <script>
88     new Vue({
89         el: '#app',
90         data:{
91             loginUsername:'',
92             menuList:[
93                 {
94                     "path": "1",
95                     "title": "工作台",
96                     "icon": "fa-dashboard",
97                     "children": []
98                 },
99                 {

```

```
95     "path": "2",
96     "title": "会员管理",
97     "icon": "fa-user-md",
98     "children": [
99         {
100             "path": "/2-1",
101             "title": "会员档案",
102             "linkUrl": "member.html",
103             "children": []
104         },
105         {
106             "path": "/2-2",
107             "title": "体检上传",
108             "children": []
109         },
110         {
111             "path": "/2-3",
112             "title": "会员统计",
113             "linkUrl": "all-item-list.html",
114             "children": []
115         },
116     ],
117 },
118 {
119     "path": "3",
120     "title": "预约管理",
121     "icon": "fa-tty",
122     "children": [
123         {
124             "path": "/3-1",
125             "title": "预约列表",
126             "linkUrl": "ordersettinglist.html",
127             "children": []
128         },
129         {
130             "path": "/3-2",
131             "title": "预约设置",
132             "linkUrl": "ordersetting.html",
133             "children": []
134         },
135         {
136             "path": "/3-3",
137             "title": "套餐管理",
138             "linkUrl": "setmeal.html",
139             "children": []
140         },
141         {
142             "path": "/3-4",
143             "title": "检查组管理",
144             "linkUrl": "checkgroup.html",
145             "children": []
146         },
147         {
148             "path": "/3-5",
149             "title": "检查项管理",
150             "linkUrl": "/pages/checkitem.html",
151             "children": []
152         },
153     ],
154 }
```

```

153     ]
154   },
155   {
156     "path": "4",
157     "title": "健康评估",
158     "icon": "fa-stethoscope",
159     "children": [
160       {
161         "path": "/4-1",
162         "title": "中医体质辨识",
163         "linkUrl": "all-medical-list.html",
164         "children": []
165       },
166     ]
167   },
168   {
169     "path": "5", //菜单项所对应的路由路径
170     "title": "统计分析", //菜单项名称
171     "icon": "fa-heartbeat",
172     "children": [//是否有子菜单，若没有，则为[]
173       {
174         "path": "/5-1",
175         "title": "会员数量统计",
176         "linkUrl": "/pages/report_member.html",
177         "children": []
178       },
179       {
180         "path": "/5-2",
181         "title": "预约套餐占比",
182         "linkUrl": "/pages/report_setmeal.html",
183         "children": []
184       },
185       {
186         "path": "/5-3",
187         "title": "运营数据统计",
188         "linkUrl": "/pages/report_business.html",
189         "children": []
190       }
191     ]
192   }
193 ]
194 },
195 mounted(){
196   // 获取登陆用户名
197   axios.get('/user/getUsername.do').then(res => {
198     if(res.data.flag){
199       this.loginUsername = res.data.data;
200     }else{
201       this.$message.error(res.data.message);
202     }
203   })
204 }
205 });
206 $(function() {
207   var wd = 200;
208   $(".el-main").css('width', $('body').width() - wd + 'px');
209 });
210 </script>

```



```
211 </html>
212
```

第二步：创建UserController并提供getUsername方法

```
1 package com.itheima.health.controller;
2
3 import com.itheima.health.constant.MessageConstant;
4 import com.itheima.health.entity.Result;
5 import org.springframework.security.core.context.SecurityContextHolder;
6 import org.springframework.security.core.userdetails.User;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 /**
12  * Description: No Description
13  * User: Eric
14  */
15 @RestController
16 @RequestMapping("/user")
17 public class UserController {
18
19     /**
20      * 获取登陆用户名
21      */
22     @GetMapping("/getUsername")
23     public Result getUsername(){
24         // 获取登陆用户的认证信息
25         User loginUser = (User)
26             SecurityContextHolder.getContext().getAuthentication().getPrincipal();
27         // 登陆用户名
28         String username = loginUser.getUsername();
29         // 返回给前端
30         return new Result(true,
31             MessageConstant.GET_USERNAME_SUCCESS, username);
32     }
33 }
```

通过debug调试可以看到Spring Security框架在其上下文中保存的用户相关信息：



显示当前登录人：



2.4. 用户退出

【路径】

- 1：在main.html中提供的退出菜单上加入超链接
- 2：在spring-security.xml文件中配置

【讲解】

第一步：在main.html中提供的退出菜单上加入超链接

```
1 <el-dropdown-item divided>
2     <a href="/logout.do"><span style="display:block;">退出</span></a>
3 </el-dropdown-item>
```

第二步：在spring-security.xml文件中配置

```
1 <!--
2     logout: 退出登录
3     logout-url: 退出登录操作对应的请求路径
4     logout-success-url: 退出登录后的跳转页面
5 -->
6 <security:logout logout-url="/logout.do"
7                 logout-success-url="/login.html" invalidate-session="true"/>
```

【小结】

1: 导入SpringSecurity环境

- (1) pom.xml中添加依赖
- (2) web.xml添加代理过滤器

2: 实现认证和授权

- (1) 导入login.html(6天资料里)登录页面 webapp目录下
- (2) 认证: SpringSecurityUserService (@Component) , 实现UserDetailsService接口
- (3) 创建UserService类、UserDao接口类、UserDao映射文件 (使用用户名查询当前用户信息, 包括角色集合和权限集合)
- (4) springmvc.xml (dubbo注解扫描范围扩大, 扫到SpringSecurityUserService)
- (5) spring-security.xml (重点 存小抄)
 - 静态资源过滤
 - 拦截的规则 security:http auto-config..., intercept-url, form-login, form-logout, csrf, security:header
 - 开启注解支持
 - 关闭跨域访问限制
 - 认证管理器->提供者user-service-ref->加密器
 - 加密器
- (6) springmvc.xml (导入spring-security.xml)
- (7) CheckItemController类 (@PreAuthorize("hasAuthority('CHECKITEM_ADD')"): 对类中的方法完成权限控制) , hasAuthority 权限校验(t_permission.keyword), hasRole角色校验(t_role.keyword)
- (8) checkitem.html (如果没有权限, 可以提示错误信息)

异常捕获HealthExceptionHandler, AccessDeniedException, return 没有权限的结果

3: 显示用户名

从SecurityContextHolder对象中获取认证的用户信息, 页面定义一个vue的data变量接收, 使用插值表达式在页面显示, 页面加载时发送请求(vue created axios)

```
1 // 获取登陆用户的认证信息
2 User loginUser = (User)
  SecurityContextHolder.getContext().getAuthentication().getPrincipal();
```

在jsp中获取登陆用户信息

```
1 ${sessionScope.SPRING_SECURITY_CONTEXT.authentication.principal.username}
```

4: 用户退出

调用/logout.do security帮我们做好了