

# SpringSecurity

## 1 简介

- 在Web开发中，安全第一！我们之前使用过的安全措施：过滤器，拦截器。。。
- 开发时，安全应该在设计之初就得考虑
- 市面上常用的安全框架：SpringSecurity、Shiro

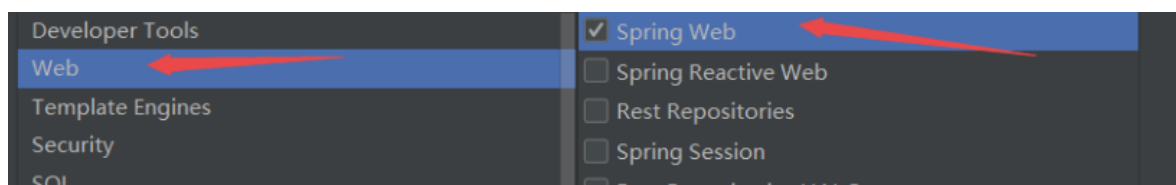
- 主要做两件事：认证（密码验证）、授权（vip1, vip2, vip3。。。）
- 功能相似
- 功能权限
- 访问权限
- 菜单权限

过去用拦截器、过滤器，使用了大量的原生代码，太冗余

- 我们仅仅需要引入spring-boot-starter-security模块，进行少量的配置，即可实现强大的安全管理
- 利用了AOP的思想
- 记住几个强大的类：
  - WebSecurityConfigurerAdapter：自定义Security策略
  - AuthenticationManagerBuilder：自定义认证策略
  - @EnableWebSecurity：开启WebSecurity模式
- 官方文档：<https://docs.spring.io/spring-security/site/docs/5.2.0.RELEASE/reference/htmlsingle/>
- 启动器文档：<https://docs.spring.io/spring-boot/docs/2.0.3.RELEASE/reference/htmlsingle/#using-boot-starter>

## 2 搭建环境

- 添加web支持



- 添加依赖

```
1 <!--SpringSecurity-->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-security</artifactId>
5 </dependency>
```

## 3 基本使用

### 3.1先写一个禁止进入没有权限的页面【授权】

```
1 package com.kuang.config;
2
```

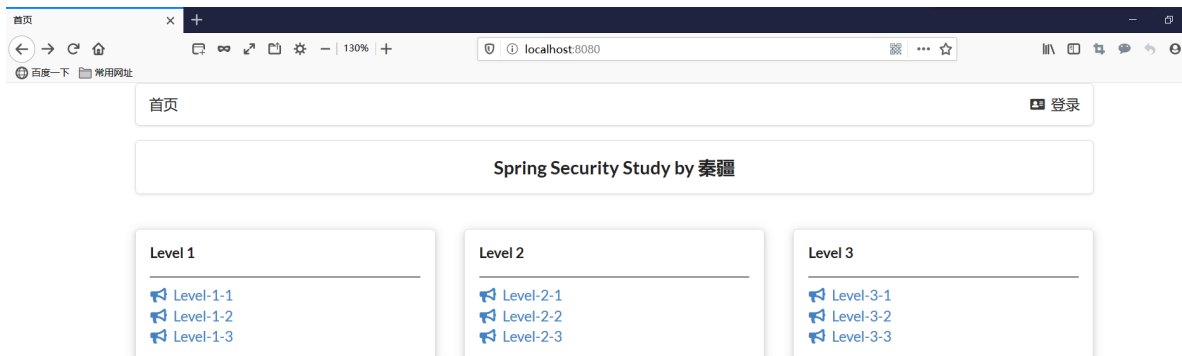
```

3  import
   org.springframework.security.config.annotation.web.builders.HttpSecurity;
4  import
   org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
5  import
   org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
6
7  /**
8   * 自定义一个Security的配置类
9   * <p>
10  * 采用的是AOP的思想，在不改变原有代码的情况下，添加访问权限的功能
11  */
12  @EnableWebSecurity
13  public class SecurityConfig extends WebSecurityConfigurerAdapter {
14
15      /**
16       * 链式编程
17       * <p>
18       * 首页所有人可以访问，功能页只有对应有权限的人才可以访问
19       * "/"代表首页
20       * <p>
21       * antMatchers 匹配请求
22       * permitAll 代表所有人都可以访问
23       * hasRole 拥有某个角色才可以访问
24       *
25       * @param http
26       * @throws Exception
27       */
28      @Override
29      protected void configure(HttpSecurity http) throws Exception {
30
31          http.authorizeRequests()
32              .antMatchers("/").permitAll()
33              .antMatchers("/level1/**").hasRole("vip1")
34              .antMatchers("/level2/**").hasRole("vip2")
35              .antMatchers("/level3/**").hasRole("vip3");
36
37      }
38  }

```

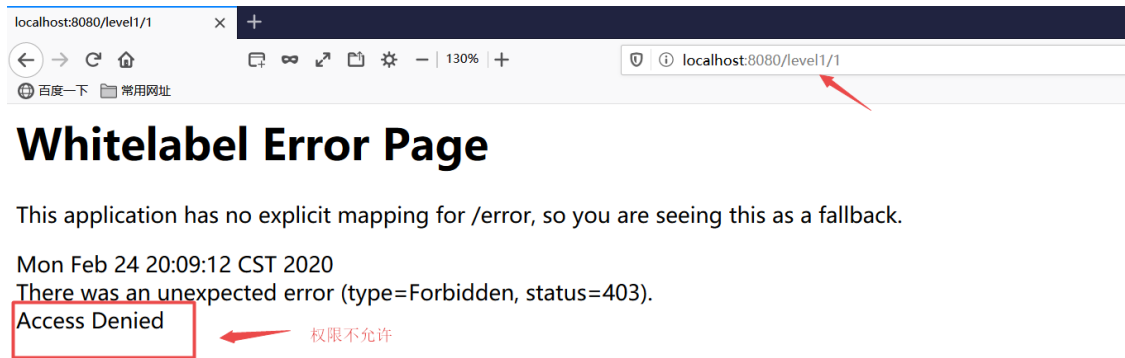
这时候我们再访问一下：

- 进入首页：



可以进入！

- 进入功能页：比如<http://localhost:8080/level1/1>



进不去了！

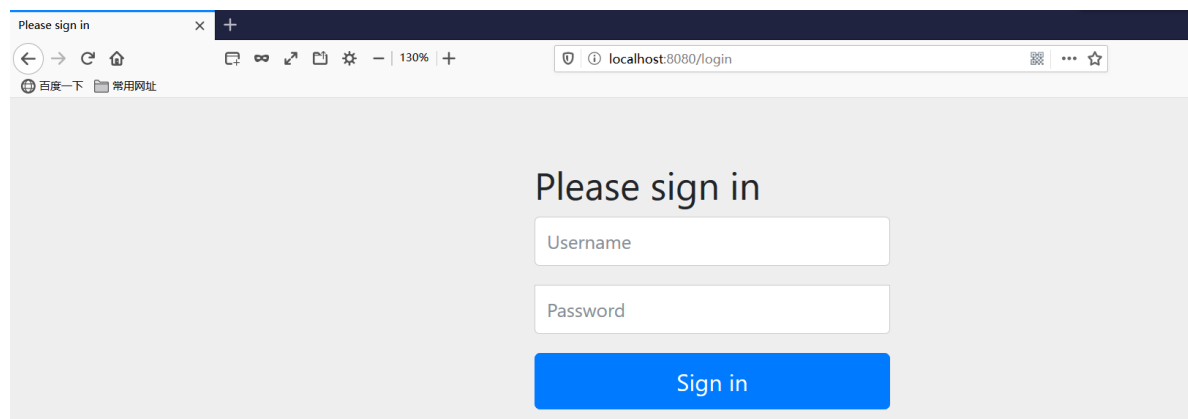
### 3.2 添加功能：没有权限会默认回到登录页面【授权】

一句话搞定：

写在protected void configure(HttpSecurity http) throws Exception里面

```
1 http.formLogin();
```

此时再次测试：



当我们点击功能页的时候，自动跳转到了login请求的界面。

但是，其实我们并没有写这个login请求，也没有写login的这个页面。

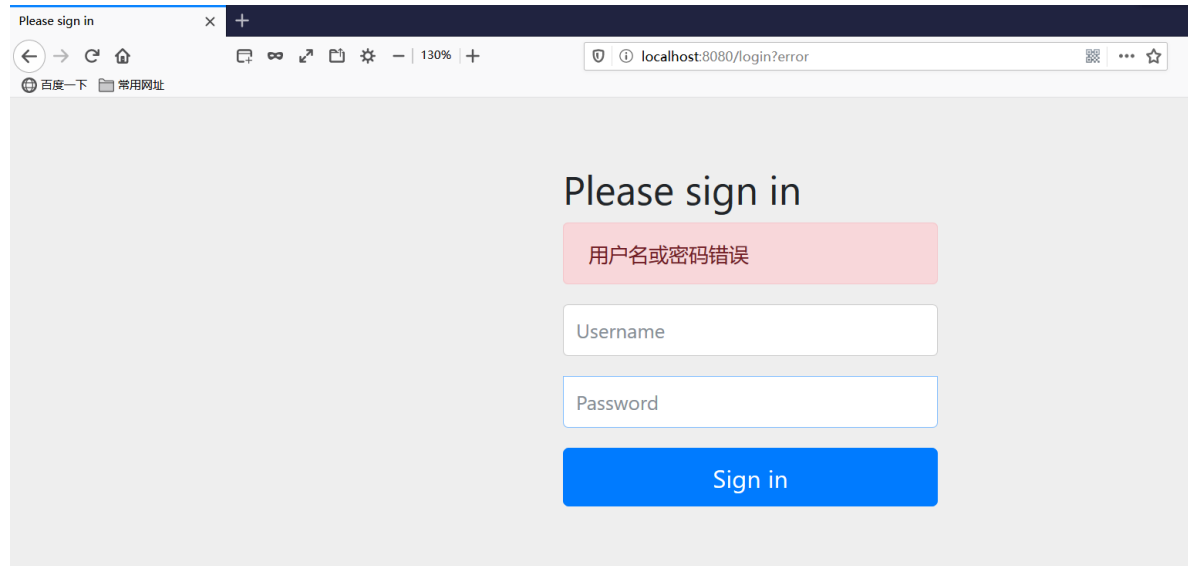
所以，这是SpringSecurity给我们的默认页面。

至于/login请求，看一下源码：

```
1 The most basic configuration defaults to automatically generating a login
  page at
2 the URL "/login", redirecting to "/login?error" for authentication failure.
  The
3 details of the login page can be found on
```

当错误请求的时候，还可以跳到/login?error请求。

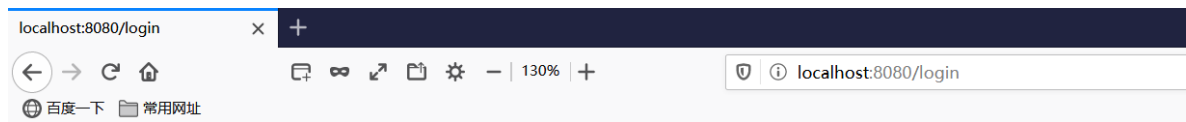
比如登录信息错误的时候：



### 3.3 添加登录信息【认证】

```
1  /**
2   * 《认证》
3   * <p>
4   * 代表我们登陆的信息可以取哪里取（内存。数据库）
5   * <p>
6   * 这些数据一般要到数据库中取到
7   * <p>
8   * inMemoryAuthentication() 代表在内存中进行授权
9   * 授权的三个信息：用户名、密码、角色名
10  * 当要进行多个用户的授权时，之间用and()进行拼接
11  * <p>
12  *
13  * @param auth
14  * @throws Exception
15  */
16 @Override
17 protected void configure(AuthenticationManagerBuilder auth) throws
18 Exception {
19     auth.inMemoryAuthentication()
20         .withUser("Geekst").password("123456").roles("vip1", "vip2")
21         .and()
22         .withUser("root").password("root").roles("vip1", "vip2",
23 "vip3");
24 }
```

测试一下：



# Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Feb 24 20:29:08 CST 2020

There was an unexpected error (type=Internal Server Error, status=500).

There is no PasswordEncoder mapped for the id "null"

发现输入正确登录信息后出现这个问题：

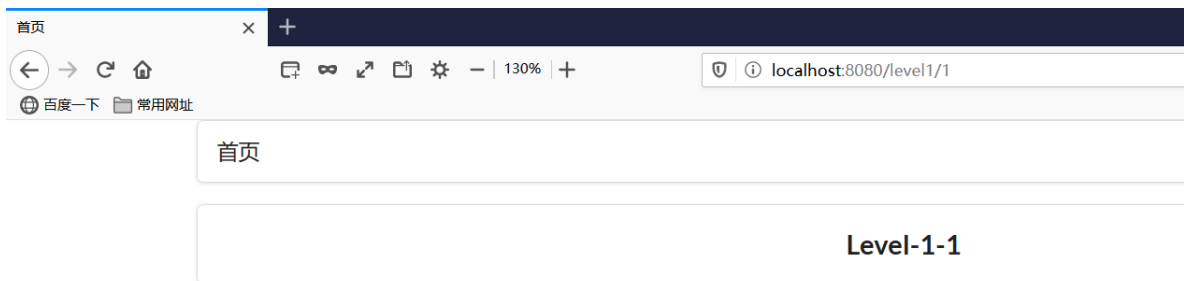
There is no PasswordEncoder mapped for the id "null"

告诉我们没有对密码进行加密。

在SpringBoot2.1.x中是可以直接使用的，但是之后的版本需密码进行加密。

```
1  /**
2   * 《认证》
3   * <p>
4   * 代表我们登陆的信息可以取哪里取（内存。数据库）
5   * <p>
6   * 这些数据一般要到数据库中取到
7   * <p>
8   * inMemoryAuthentication() 代表在内存中进行授权
9   * 授权的三个信息：用户名、密码、角色名
10  * 通过roles进行授权
11  * 当要进行多个用户的授权时，之间用and()进行拼接
12  * <p>
13  * <p>
14  * 注意点：
15  * 在SpringSecurity 5.0+ 新增了很多的加密方法，必须对密码进行加密，否则会报错：
16  * There is no PasswordEncoder mapped for the id "null"
17  * 加密设置：passwordEncoder ===》BCryptPasswordEncoder是比较推荐的方式
18  *
19  * @param auth
20  * @throws Exception
21  */
22  @Override
23  protected void configure(AuthenticationManagerBuilder auth) throws
24  Exception {
25      auth.inMemoryAuthentication().passwordEncoder(new
26  BCryptPasswordEncoder())
27      .withUser("Geekst").password(new
28  BCryptPasswordEncoder().encode("123456")).roles("vip1", "vip2")
29      .and()
30      .withUser("root").password(new
31  BCryptPasswordEncoder().encode("root")).roles("vip1", "vip2", "vip3");
32  }
```

测试：



输入正确的登录信息后，跳转成功了！而且该用户下的具有权限的网页均可进入。

如果要用数据库信息验证：

```
1  @Autowired
2  private DataSource dataSource;
3
4  @Autowired
5  public void configureGlobal(AuthenticationManagerBuilder auth) throws
   Exception {
6      // ensure the passwords are encoded properly
7      UserBuilder users = User.withDefaultPasswordEncoder();
8      auth
9          .jdbcAuthentication()
10             .dataSource(dataSource)
11             .withDefaultSchema()
12
13             .withUser(users.username("user").password("password").roles("USER"))
14             .withUser(users.username("admin").password("password").roles("USER", "ADMIN")
15             );
16 }
```

### 3.4 开启注销功能【授权】

#### 3.4.1 系统注销后默认跳转/login请求【登录页面】

也只需一行代码：【写在protected void configure(HttpSecurity http) throws Exception里面】

```
1  //开启注销功能
2  http.logout();
```

看下源码：

```
1  * Provides logout support. This is automatically applied when using
2  * {@link WebSecurityConfigurerAdapter}. The default is that accessing the URL
3  * "/logout" will log the user out by invalidating the HTTP session, cleaning
   up any
4  * {@link #rememberMe()} authentication that was configured, clearing the
5  * {@link SecurityContextHolder}, and then redirect to "/login?success".
```

这段说了：通过/logout进行注销，而注销成功则会进入/login?success请求。

我们在前端写一个代码：

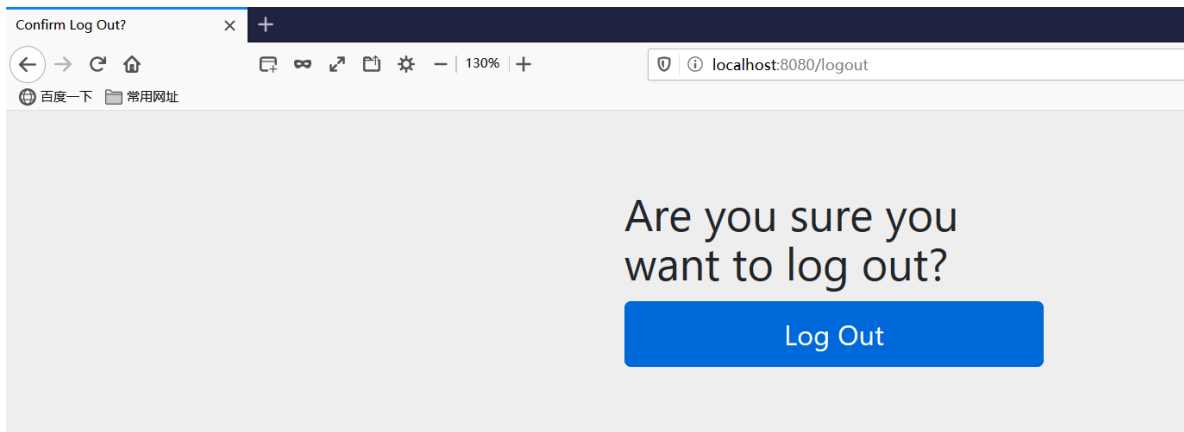
```
1 <!--注销-->
2 <a class="item" th:href="@{/logout}">
3     <i class="sign-out icon"></i> 注销
4 </a>
```

重点在于跳转的那个：th:href="@{/logout}"

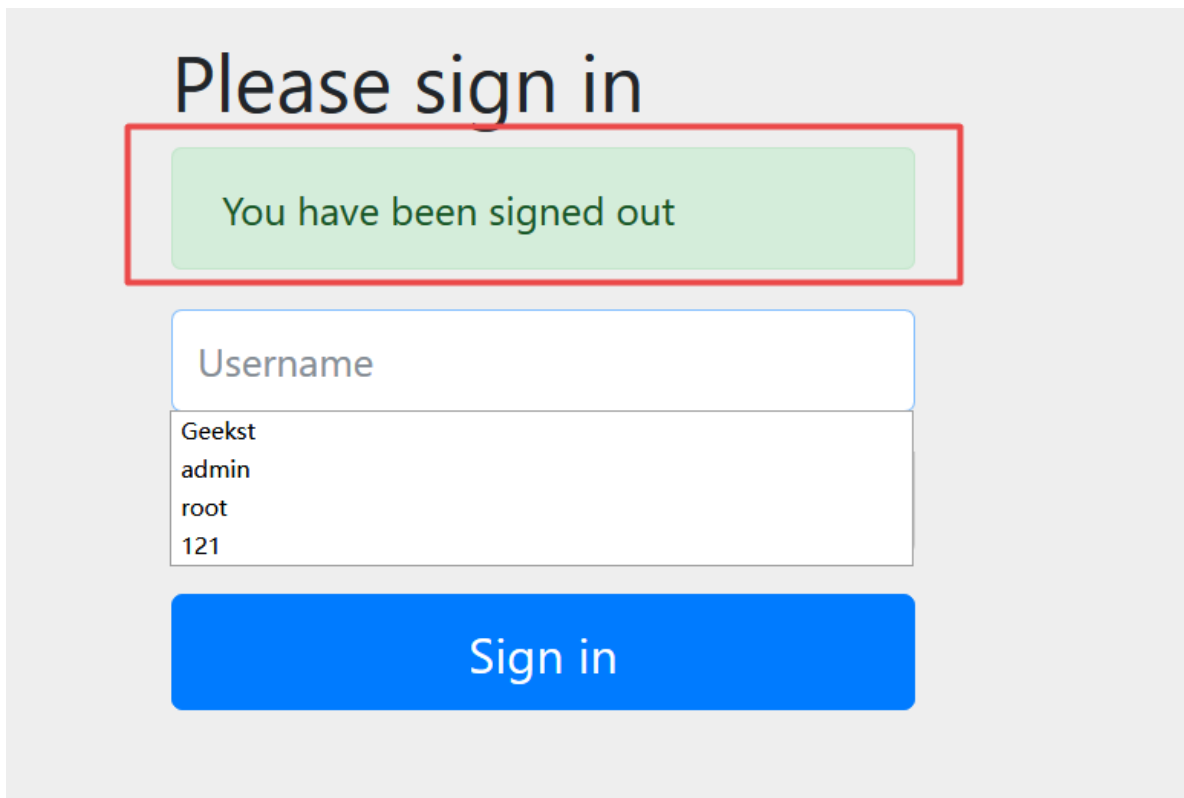
我们输入正确的登录信息，然后点击注销：



弹出这个页面：



点击Log Out后：



这时候我们进入首页，发现没有权限进入功能页了，证明退出成功！

### 3.4.2 自定义注销后的跳转

还是先看一下源码：

```

1  *   &#064;Override
2  *   protected void configure(HttpSecurity http) throws Exception {
3  *
4      http.authorizeRequests().antMatchers("&quot;/**&quot;").hasRole("&quot;USER&quot;
5      ;).and().formLogin()
6      *
7      *           .and()
8      *           // sample logout customization
9      *
10     .logout().deleteCookies("&quot;remove&quot;").invalidateHttpSession(false)
11     *
12     .logoutUrl("&quot;/custom-
13     logout&quot;").logoutSuccessUrl("&quot;/logout-success&quot;");
14 *   }

```

源码告诉我们通过设置logoutUrl进行跳转，而且还有一些设置。

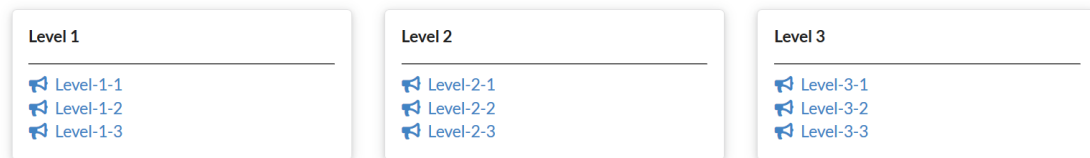
```

1  //开启注销功能
2  //deleteCookies: remove代表移除所有的Cookie
3  //invalidateHttpSession: true代表清空所有的Session
4  //logoutUrl 注销请求发起后，并且执行成功后，发起的请求
5  http.logout()
6      .deleteCookies("remove")
7      .invalidateHttpSession(true)
8      .logoutSuccessUrl("/");

```

【注意：这里用的是logoutSuccessUrl，代表注销成功后的请求，而不是使用logoutUrl。】

### 3.5 不显示没有权限的链接（菜单）



我们不应该将所有的菜单都展示，而是应该有全权限的菜单才可以展示！

这里，我们又要使用到了Thymeleaf了，这也是其强大的地方。==》整合SpringSecurity和Thymeleaf

- 这里我们先说一个最主要的大问题：SpringBoot的版本问题

在高版本的SpringBoot中，不支持以下将要使用的sec: 标签【无法识别】，所以我们将版本降到2.0.9

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" ?>
<modelVersion>4.0.0</modelVersion>
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.9.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.kuang</groupId>

```

#### 3.5.1 导入整合包



```

1 <!--Thymeleaf和SpringSecurity整合包-->
2 <!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-
   extras-springsecurity4 -->
3 <dependency>
4     <groupId>org.thymeleaf.extras</groupId>
5     <artifactId>thymeleaf-extras-springsecurity4</artifactId>
6     <version>3.0.4.RELEASE</version>
7 </dependency>

```

有了这个整合包，就可以在Thymeleaf中写SpringSecurity相关的东西。

html中导入命名空间：xmlns:sec="<http://www.thymeleaf.org/thymeleaf-extras-springsecurity4>"

```

1 <html lang="en" xmlns:th="http://www.thymeleaf.org"
2     xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">

```

### 3.5.2 关闭csrf功能

什么是csrf?

跨站请求伪造（**英语：Cross-site request forgery**），也被称为 one-click attack\*\* 或者 **session riding**，通常缩写为 **CSRF** 或者 **XSRF**，是一种挟制用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法。跟[跨网站脚本](#)（XSS）相比，XSS 利用的是用户对指定网站的信任，CSRF 利用的是网站对用户网页浏览器的信任。

如果不关闭，我们后面在注销的时候会报错。

### 3.5.3 修改主页

三个按钮：

```

1 <!--如果未登录，则显示登陆按钮
2 !isAuthenticated() 代表未登录
3 -->
4 <div sec:authorize="!isAuthenticated()">
5     <a class="item" th:href="@{/toLogin}">
6         <i class="address card icon"></i> 登录
7     </a>
8 </div>
9
10
11 <!--如果已登录，则显示用户名、注销按钮-->
12 <div sec:authorize="isAuthenticated()">
13     <a class="item">
14         用户名: <span sec:authentication="name"></span>
15         角色: <span sec:authentication="principal.authorities"></span>
16     </a>
17 </div>
18 <div sec:authorize="isAuthenticated()">
19     <!--注销-->
20     <a class="item" th:href="@{/logout}">
21         <i class="sign-out icon"></i> 注销
22     </a>
23 </div>

```

菜单栏的动态显示：【这里页面稍简略，不影响理解】

```

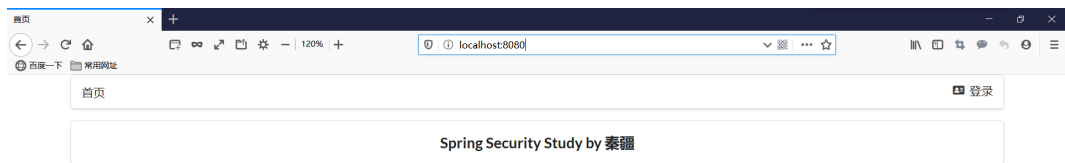
1 <div class="column" sec:authorize="hasRole('vip1')">
2     ... vip1的菜单栏。。。
3 </div>
4
5 <div class="column" sec:authorize="hasRole('vip2')">
6     ... vip2的菜单栏。。。
7 </div>
8
9 <div class="column" sec:authorize="hasRole('vip3')">
10     ... vip3的菜单栏。。。
11 </div>

```

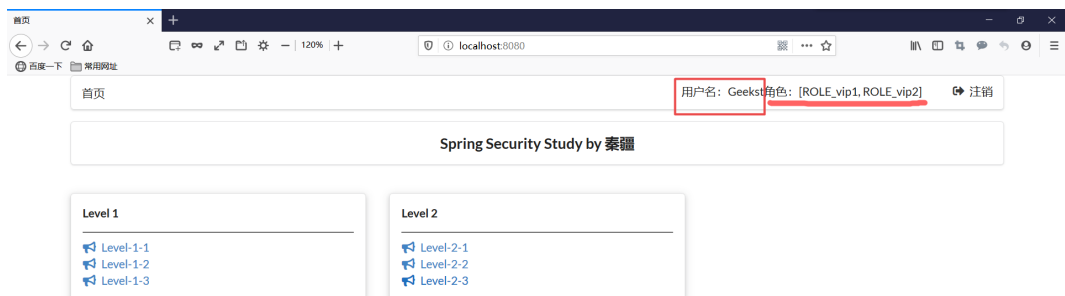
hasRole: 代表拥有某个角色

- 测试

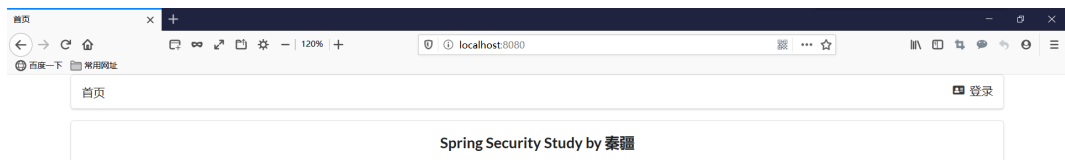
- 未登录的主页



- 登录具有vip1和vip2权限的Geekst用户



- 点击注销



## 3.6 记住我

只需一行代码:

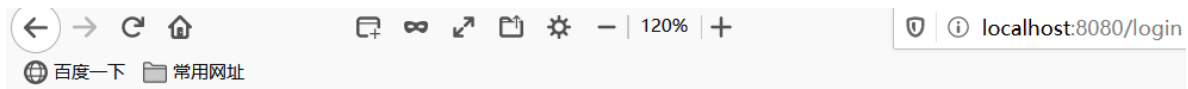
```

1 //开启记住我功能(本质就是cookie)
2 http.rememberMe();

```

- 测试:

马上登陆界面多了一个:



## Login with Username and Password

User:

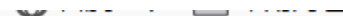
Password:

☐

Remember me on this computer.

Login

输入正确的登录信息，点击Login:



## Login with Username and Password

User:

Password:

☒

Remember me on this computer.

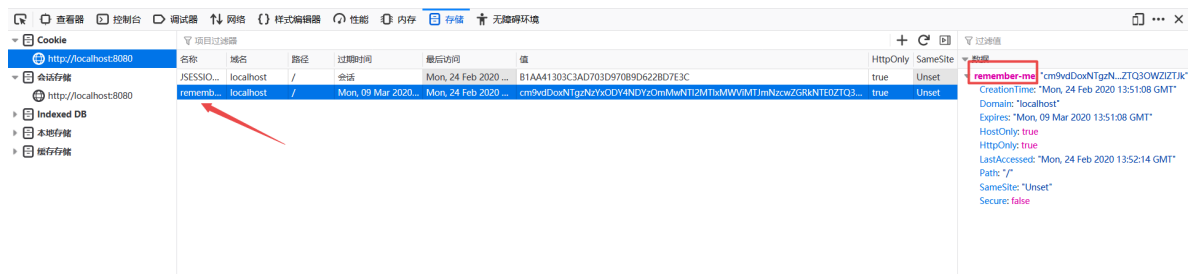
Login

关闭浏览器，再打开浏览器:



发现自动进来了。

审查元素看一下:



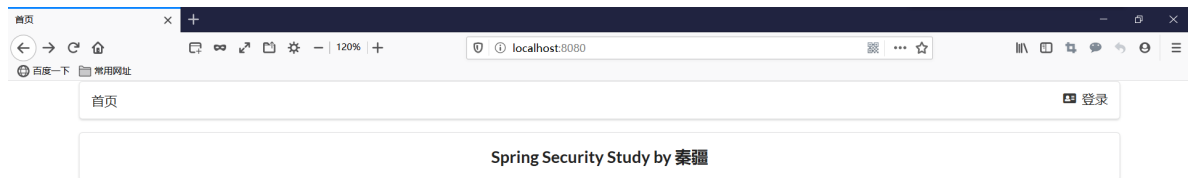
确实存在remember-me这个cookie。

看一下有效日期：Expires=》默认保存两周

清除浏览器的Cookie：



重新打开：



## 3.7 主页定制

### 3.7.1 请求一致

之前的代码修改一下：

```
1 //没有权限会默认回到登录页面
2 http.formLogin().loginPage("/toLogin");
```

让它去走我们的/toLogin请求。

于是，我们表单提交也得和这里一样：

```
1 <form th:action="@{/toLogin}" method="post">
```

测试：


点击登录：

# 登录

Username

 root

Password

 .....

提交查询

注册

blog.kuangstudy.com

## Spring Security Study by 秦疆



登录成功!

### 3.7.2 请求不一致

假如表单跳转的是:

咱们并不存在这个请求, 我们跳转的是/toLogin, 所以我们得在这里修改一下:


```
1 http.formLogin()
2   .loginPage("/toLogin")
3   .loginProcessingUrl("/login");
```

loginProcessingUrl是我们真正走的请求! 【可以理解为是表单提交的请求, 和表单请求一样, 才可进行认证】


测试一下:



Username

 Geekst

Password

 •••••

提交查询

注册

blog.kuangstudy.com

Spring Security Study by 秦疆

百度一下 常用网址

首页 用户名: Geekst角色: [ROLE\_vip1, ROLE\_vip2] 注销

Spring Security Study by 秦疆

Level 1

- Level-1-1
- Level-1-2
- Level-1-3

Level 2

- Level-2-1
- Level-2-2
- Level-2-3

也一样可以!

### 3.7.3 前端的name和后端的属性名不一致

假如前端的usr和pwd:

```
1 <input type="text" placeholder="Username" name="usr">
2 <input type="password" name="pwd">
```

后台是: username 和 password

我们这样子, 在进行登录的时候, 会接收不到验证。

因为底层用的就是username和password (看源码)。

所以, 我们可以这么改动:


```
1 http.formLogin().loginPage("/toLogin")
2     .usernameParameter("usr")
3     .passwordParameter("pwd")
4     .loginProcessingUrl("/login");
```

用usernameParameter和passwordParameter来指定前端的name。

测试：


登录

Username



Geekst

Password



•••••

提交查询

注册

blog.kuangstudy.com

Spring Security Study by 秦疆



还是登录成功！

### 3.7.4 加上“记住我”

在我们自己的页面加上这个按钮：

```
1 | <input type="checkbox" name="remember">记住我
```

## 登录

Username

 Username

Password



☐ 记住我

提交查询

注册

Spring Security Study by 秦疆

代码修改:


```
1 //开启记住我功能(本质就是Cookie)，默认保存两周
2 //自定义接收前端的参数: rememberMeParameter
3 http.rememberMe().rememberMeParameter("remember");
```

名字和前端的name一致!


- 测试:

## 登录

Username

 Geekst

Password

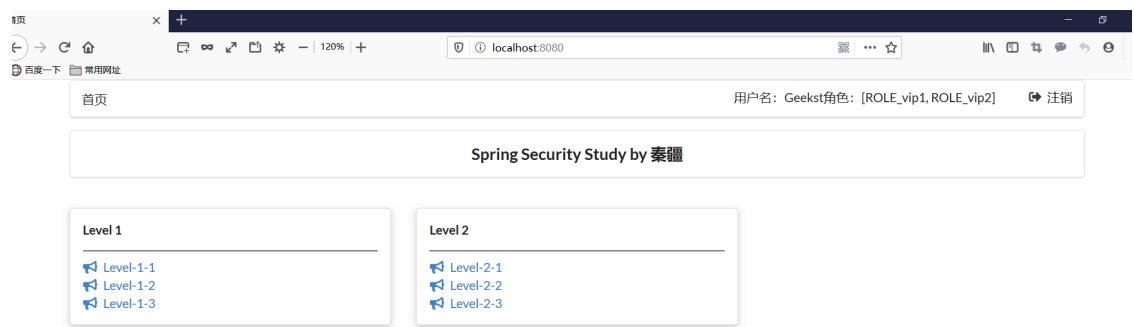
 •••••

☒ 记住我

提交查询

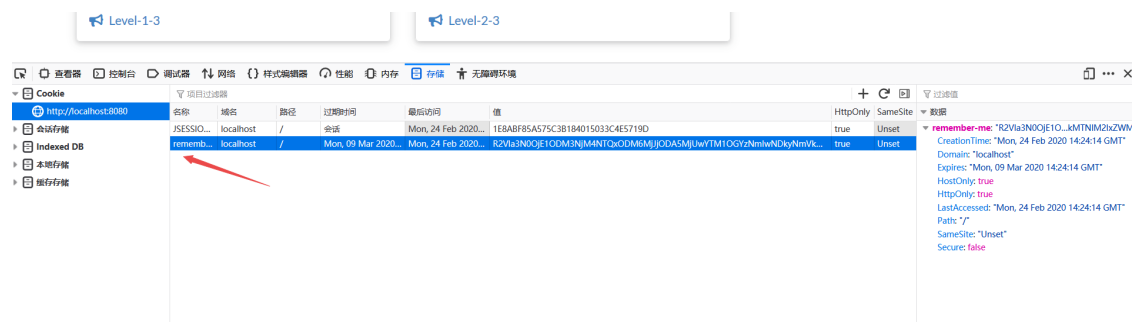
注册



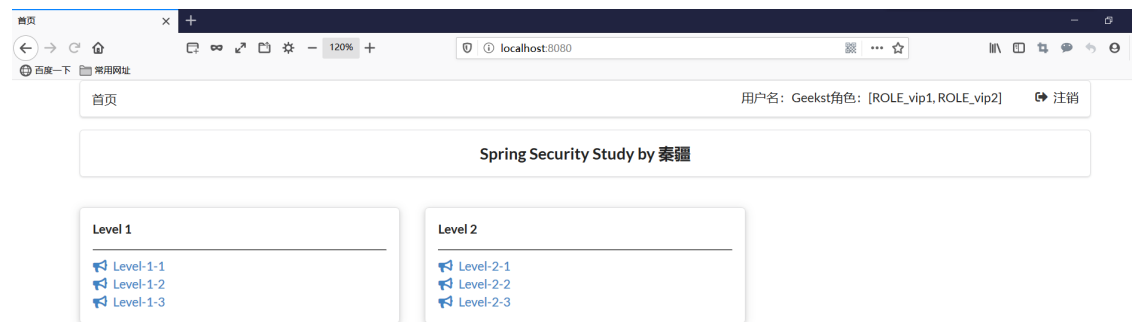


登录成功！

现在来看一下Cookie：



一样的关闭浏览器，再打开浏览器：



依然在。