在 Spring Security 中使用 ThymeLeaf 页面模板引擎

韩博孙

2021 年 2 月 16 日 麻省理工学院

评价我: 5.00/5 (2 票)

关于如何设置应用程序 Spring Security 和 Spring MVC 以及使用 ThymeLeaf 进行安全页面渲染的教程。

在本教程中，在简要了解如何将 Spring Security 添加到 Spring MVC 应用程序之后，您将学习如何将 Spring Security 与 Thymeleaf 模板引擎集成。

下载演示项目 - 401.1 KB

介绍

这是我关于 Thymeleaf 模板引擎的第二个教程。在本教程中，我想展示如何将 Spring Security 与 Thymeleaf 模板引擎集成。我写了一篇关于如何将 Spring Security 添加到 Spring MVC 应用程序的教程。在本教程中，我将对其进行回顾。我学到了一些新概念。最重要的部分是如何使用 Spring Security 标签和 Thymeleaf 模板引擎来保护页面渲染。总的来说，让所有这些都发挥作用的工作一点也不困难。我根据我之前的 Thymeleaf 教程构建了这个示例。我以前的 Thymeleaf 教程可以在这里找到。本教程还大量借鉴了我关于 Spring Security 集成的其他教程。你可以在这里找到这个教程. 您不必检查这两个教程。这里会提到重要的概念。请继续阅读。

应用架构

本教程中的示例应用程序需要用户先登录，否则用户将无法访问应用程序提供的安全资源。Web 应用程序将有一些新配置要添加到启动中。为了添加这样的配置，我不得不将 spring-boot-starter-security jar 添加到项目中。添加此 jar 后，用户必须登录才能访问页面。这意味着我必须做一些更多的配置更改以允许匿名访问某些页面，而其他页面仅限于具有属性访问权限的登录用户。我将解释这是如何完成的，以及用户身份验证和授权检查的工作原理。

本教程中更重要的概念是用于显示或隐藏网页部分的安全标签。网页使用 Thymeleaf 模板引擎呈现。我担心的一件事是这些标签是什么。我一点也不关心如何使用它们，因为我知道如何使用。对于这个示例应用程序，我只使用了基于角色的授权。如何分配角色是安全配置的一部分，将详细说明。然后是安全标签。这些是 ThymeLeaf 特定的标签，与我的其他教程中使用的 JSTL/Spring Security 标签非常相似。

由于本教程混合了 Spring Security 和 ThymeLeaf，让我们从 maven POM 文件开始。关于这个文件，有几件事需要讨论。

Maven POM 文件

对于此示例应用程序，我将其打包为 jar 文件。这与我在之前的 ThymeLeaf 教程中所做的相同。使用 Thymeleaf 模板引擎，我不必担心以 war 格式打包 Web 应用程序。为了支持在 HTML 标记中使用 Spring Security 和安全标记，我必须在 Maven POM 文件中包含这些。首先，我需要为我的项目声明父 POM，即`spring-boot-starter-parent`.

XML 复制代码

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
</parent>
```

接下来，我需要添加所有必要的依赖项。对于 Spring Boot 应用程序，所需的依赖项很少。他们来了：

XML 复制代码

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
  <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
     <groupId>org.thymeleaf.extras</groupId>
     <artifactId>thymeleaf-extras-springsecurity5</artifactId>
     <version>3.0.4.RELEASE</version>
  </dependency>
</dependencies>
```

关于这一点，您需要了解以下几点。要为应用程序启用安全性，我需要添加`spring-boot-starter-security`依赖项。添加后，默认情况下，对应用程序的所有请求都必须经过身份验证。一些配置可以使对请求的访问更加灵活。我将在后面的部分中向您展示如何操作。接下来，我需要添加 ThymeLeaf 核心的依赖项，即`spring-boot-starter-thymeleaf`. 添加此依赖项允许我在 HTML 页面中使用 ThymeLeaf 标签，并允许我创建可重用的组件，称为片段。为了能够对 HTML 页面使用安全标记，我必须添加最后一个依赖项，称为`thymeleaf-extras-springsecurity5`.

这五个依赖项都是让这个应用程序工作所必需的。这是第一步。在下一节中，我将向您展示启动类的源代码，以及安全配置。安全配置是最重要的，因为它决定了哪些请求可以在没有身份验证的情况下访问，哪些不能。

启动和安全配置

启动类非常简单。这是课程：

爪哇 复制代码

```java
package org.hanbo.boot.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class App
{
   public static void main(String[] args)
   {
      SpringApplication.run(App.class, args);
   }
}
```

我所要做的就是声明一个名为“ App”的类。然后使用注解@SpringBootApplication来装饰它。这被传递到 SpringApplication.run(). Spring Boot 使用依赖注入来查找对象类型的相互依赖关系。这是创建依赖关系映射的地方。Spring IOC 容器将扫描包以查找所有类类型。只要类与注释@Configuration，@Repository，@Service或@Components，以及Controller和RestController，春天会发现他们并创建一个查找表这些。然后，对象实例化将使用映射来查找要注入的对象。

我们来看看设置 Spring Security 的配置类。这是此示例应用程序中最难的部分。本教程为基于表单的身份验证与 Web 应用程序的集成提供了最基本的配置。这是该类的完整源代码：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.app.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.
       authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.
       method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.
       web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.security.web.authentication.
       SavedRequestAwareAuthenticationSuccessHandler;
import org.hanbo.boot.app.security.UserAuthenticationService;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter
{
   @Autowired
   private UserAuthenticationService authenticationProvider;

   @Autowired
   private AccessDeniedHandler accessDeniedHandler;

   @Override
   protected void configure(HttpSecurity http) throws Exception
   {
      http
      .authorizeRequests()
          .antMatchers("/assets/**", "/public/**").permitAll()
          .anyRequest().authenticated()
      .and()
      .formLogin()
          .loginPage("/login")
          .permitAll()
          .usernameParameter("username")
          .passwordParameter("userpass")
          .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
          .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")
          .and()
      .logout().logoutSuccessUrl("/public/logout")
          .permitAll()
          .and()
      .exceptionHandling().accessDeniedHandler(accessDeniedHandler);
   }

   @Override
   protected void configure(AuthenticationManagerBuilder authMgrBuilder)
      throws Exception
   {
      authMgrBuilder.authenticationProvider(authenticationProvider);
   }
}
```

这门课有一些有趣的事情。首先，类声明如下：

爪哇 复制代码

```
...
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)
public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter
{
...
}
```

该类使用三种不同的注释进行装饰。第一个将类标记为配置提供程序（`@Configuration`）。第二个为 Web 应用程序启用 Web Security（`@EnableWebSecurity`）。第三个很有趣，它的作用是在类的方法上启用安全注解，尤其是在控制器或`RestController`类的方法上，并锁定对这些方法的访问。

该类从`WebSecurityConfigurerAdapter`. 它可用于提供 Web 安全和 HTTP 安全的定制。当Spring框架得到这个类时，它会调用这个类的`configure()`方法来做web安全配置。在我的`WebAppSecurityConfig`课堂上，有两种`configure()`方法。第一个是安全配置，比如哪些页面可以匿名访问（无需登录），哪些页面只能安全访问（访问前必须登录）。这里是：

爪哇 复制代码

```
@Override
protected void configure(HttpSecurity http) throws Exception
{
   http
   .authorizeRequests()
       .antMatchers("/assets/**", "/public/**").permitAll()
       .anyRequest().authenticated()
   .and()
   .formLogin()
       .loginPage("/login")
       .permitAll()
       .usernameParameter("username")
       .passwordParameter("userpass")
       .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
       .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")
       .and()
   .logout().logoutSuccessUrl("/public/logout")
       .permitAll()
       .and()
   .exceptionHandling().accessDeniedHandler(accessDeniedHandler);
}
```

这看起来是一种可怕的方法。这是一种可怕的方法。如果我没有正确设置它，它会让我以后后悔。让我一次只讨论一个部分。第一个是这样的：

爪哇 复制代码

```
...
http
.authorizeRequests()
    .antMatchers("/assets/**", "/public/**").permitAll()
    .anyRequest().authenticated()
...
```

这部分相当简单，我希望所有传入的请求除了具有“ *.../assets/...* ”和“ *.../public/...* ”部分的请求在访问之前进行身份验证。对于具有这些子路径的请求，将可以公开访问。下一部分是登录页面的配置，以及登录成功和失败时的操作：

爪哇 复制代码

```
...
.formLogin()
    .loginPage("/login")
    .permitAll()
    .usernameParameter("username")
```

```
    .passwordParameter("userpass")
    .successHandler(new SavedRequestAwareAuthenticationSuccessHandler())
    .defaultSuccessUrl("/secure/index", true).failureUrl("/public/authFailed")
...
```

我将登录页面 URL 路径设置为“<Application Base URL>/login”。并且此页面也设置为可公开访问。如果我不这样做（将其设置为可公开访问），则需要对其进行身份验证才能访问，这就造成了小鸡和鸡蛋的悖论。实际上，没有悖论。如果我不将登录页面设置为可公开访问，用户将无法登录，也无法看到受保护的页面。

方法`usernameParameter()`和`passwordParameter()`设置登录表单上可以提供用户名和密码的输入字段是什么。如果不使用这两种方法设置字段名，则有默认的字段名。我认为它们是“用户名”和“密码”。对于我的应用程序，我将它们设置为“`username`”和“`userpass`”。接下来，我使用方法`successHandler()`设置动作以重定向到预期的子 URL（如果有）。例如，如果我想访问：

`authenticationProvider`：

复制代码

```
http://localhost:8080/webapp/secure/page1
```

而且我还没有登录，我会先被引导到登录页面。通过调用`.successHandler(new SavedRequestAwareAuthenticationSuccessHandler())`，在我成功登录后，这会将我重定向回原始 url。如果没有 url 重定向回来，那么我使用 next 方法设置用户成功登录时的默认登陆页面，`.defaultSuccessUrl("/secure/index", true)`。如果用户登录失败，那么我使用最后一种方法设置登录失败页面，`.failureUrl("/public/authFailed")`.

下一部分是注销行为的配置：

爪哇 复制代码

```
...
logout().logoutSuccessUrl("/public/logout")
.permitAll()
...
```

所有这些都是为了指定当用户注销时用户将被重定向到哪里。并且注销目标页面将设置为可供公众访问。同样，如果该页面不可公开访问。然后注销后，注销页面将无法访问，用户将收到 403 错误。

我要做的最后一个配置是如何处理拒绝访问错误。我会将这种类型的错误重定向到另一个页面。这是我的配置方式：

爪哇 复制代码

```
...
.exceptionHandling().accessDeniedHandler(accessDeniedHandler);
...
```

`accessDeniedHandler`object 是自定义对象。它在此类的顶部定义为：

爪哇 复制代码

```
@Autowired
private AccessDeniedHandler accessDeniedHandler;
```

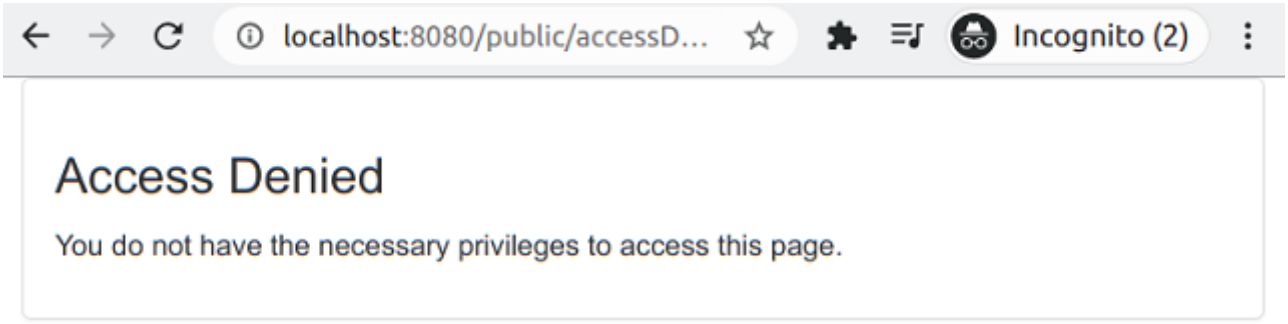
定义的类`accessDeniedHandler`如下所示：

爪哇 缩小▲ 复制代码

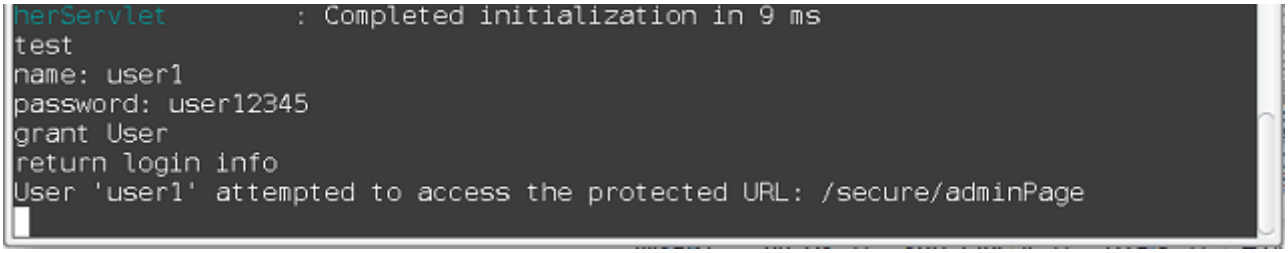
```
package org.hanbo.boot.app.config;

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.access.AccessDeniedHandler;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

// handle 403 page
@Component
public class MyAccessDeniedHandler implements AccessDeniedHandler
{
    @Override
    public void handle(HttpServletRequest httpServletRequest,
                       HttpServletResponse httpServletResponse,
                       AccessDeniedException e) throws IOException, ServletException
    {
        Authentication auth
           = SecurityContextHolder.getContext().getAuthentication();

        if (auth != null)
        {
            System.out.println("User '" + auth.getName()
                    + "' attempted to access the protected URL: "
                    + httpServletRequest.getRequestURI());
        }

        httpServletResponse.sendRedirect(httpServletRequest.getContextPath() +
                                         "/public/accessDenied");
    }
}
```

这个处理程序所做的只是拦截发生的事情`AccessDeniedException`，而不是显示 Spring Boot 错误页面，而是将响应重定向到我的访问被拒绝页面，如下所示：



处理程序拦截访问被拒绝异常时，将向终端输出`string`：“`User '<username>' attempted to access the protected URL: <URL user attempted to access>`”。这是终端上的屏幕截图：



最后，问题仍然存在。我如何授权用户？我所说的授权是指在确认用户身份后，应用程序将确定应该授予用户什么访问权限。大多数情况下，我使用基于角色的授权。这足以满足我的业余项目所需的一切。如果需要不同类型的授权，基于权限的授权，Spring Security 为其提供了必要的机制。但本教程不会介绍它。

在我的`WebAppSecurityConfig`课程中，我添加了一个`authenticationProvider`对象并配置为确定用户的授权。这是声明和配置：

爪哇 复制代码

```
...

   @Autowired
   private UserAuthenticationService authenticationProvider;
```

```
...

   @Override
   protected void configure(AuthenticationManagerBuilder authMgrBuilder)
      throws Exception
   {
      authMgrBuilder.authenticationProvider(authenticationProvider);
   }

...
```

该类UserAuthenticationService定义如下:

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.app.security;

import java.util.ArrayList;
import java.util.List;

import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Service;

@Service
public class UserAuthenticationService
implements AuthenticationProvider
{
   @Override
   public Authentication authenticate(Authentication auth) throws AuthenticationException
   {
      System.out.println("test");
      Authentication retVal = null;
      List<GrantedAuthority> grantedAuths = new ArrayList<GrantedAuthority>();

      if (auth != null)
      {
         String name = auth.getName();
         String password = auth.getCredentials().toString();
         System.out.println("name: " + name);
         System.out.println("password: " + password);

         if (name.equals("admin") && password.equals("admin12345"))
         {
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_ADMIN"));
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));

            retVal = new UsernamePasswordAuthenticationToken(
               name, "", grantedAuths
            );
            System.out.println("grant Admin");
         }
         else if (name.equals("staff1") && password.equals("staff12345"))
         {i
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_STAFF"));
            grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));

            retVal = new UsernamePasswordAuthenticationToken(
               name, "", grantedAuths
            );
            System.out.println("grant Staff");
         }
         else if (name.equals("user1") && password.equals("user12345"))
```

```
            {
               grantedAuths.add(new SimpleGrantedAuthority("ROLE_USER"));

               retVal = new UsernamePasswordAuthenticationToken(
                  name, "", grantedAuths
               );
               System.out.println("grant User");
            }
         }
         else
         {
            System.out.println("invalid login");
            retVal = new UsernamePasswordAuthenticationToken(
               null, null, grantedAuths
            );
            System.out.println("bad Login");
         }

         System.out.println("return login info");
         return retVal;
      }

      @Override
      public boolean supports(Class<?> tokenType)
      {
         return tokenType.equals(UsernamePasswordAuthenticationToken.class);
      }
}
```

从此类实例化的对象可用于验证用户名和密码。确认后，我会将用户角色分配给用户。管理员用户将具有管理员、员工和用户角色。员工级别用户将拥有员工和用户角色。而用户级用户只能拥有用户角色。如果用户无法通过身份验证，则无法分配用户角色。

这就是后端 Web 安全配置的全部内容。我只想再提一个问题。在我之前的教程中，我为表单和 HTTP POST 操作禁用了 CSRF。我认为这是一件坏事。它削弱了应用程序的安全性。在本教程中，我启用了 CSRF。在接下来的几节中，我将展示如何利用此功能。

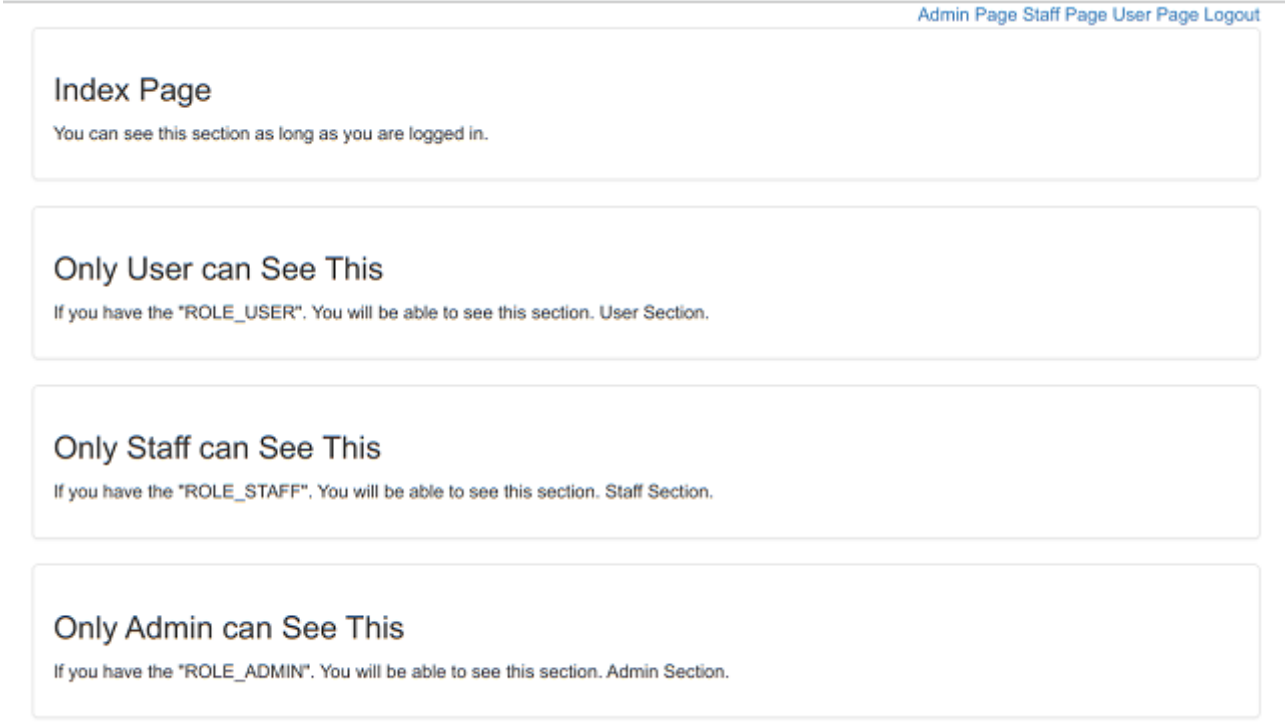
通过 ThymeLeaf 使用 Spring Security

我写这个教程的最大原因是我想知道如何在 ThymeLeaf 中使用安全标签。我想在我的下一个项目中使用 ThymeLeaf，并且集成 Spring Security 对成功至关重要。在我可以有效地使用它之前，我想测试一下。因此创建了本教程。事实证明，将 Spring Security 标签与 ThymeLeaf 集成起来很容易。在关于 Maven POM 的部分中，我已经提到需要依赖`thymeleaf-extras-springsecurity5`。这是为页面提供安全标签的标签。

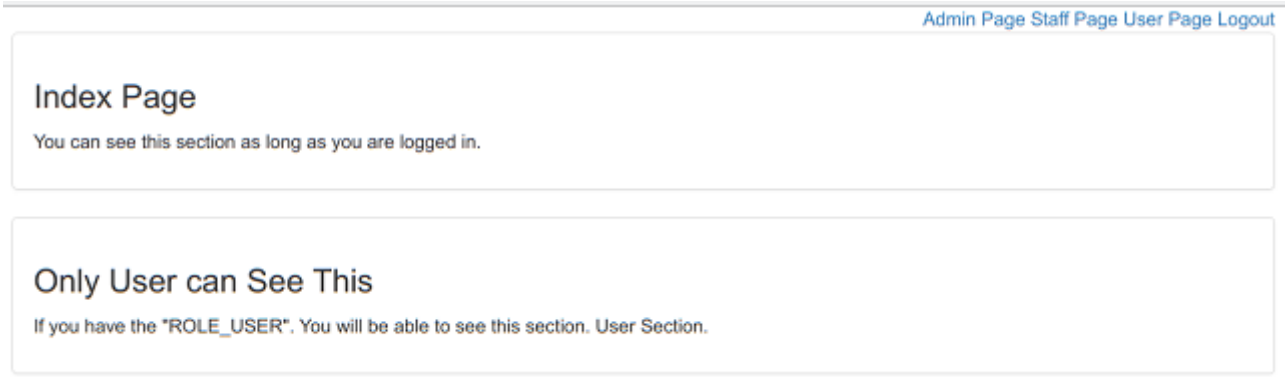
接下来，应为 ThymeLeaf 模板引擎设置文件夹结构。在项目文件夹资源下，我创建了一个名为*"模板"*的文件夹。这是存储页面模板和可重用片段的地方。对于这个项目，我设置了四个不同的页面：

- 每个人都可以看到一页，但每个角色不同的用户看到的页面部分会有所不同。
- 一页只有管理员用户可以访问。
- 只有员工级别和管理员用户可以看到的一页。
- 管理员用户、职员级别和普通用户可以看到一页。但不是匿名用户。

为了设置这些页面模板以确保不同级别的用户可以访问这些页面，我需要在页面模板上使用 Spring Security 标签。让我向您展示所有用户都可以访问的安全索引页面的屏幕截图，但不同的用户会看到不同的页面：



此屏幕截图仅在用户为管理员用户时可见。如果登录的用户是非管理员级别的用户，例如非管理员、非员工级别的用户登录，将看到相同的页面，如下所示：



这两个截图的区别是通过 Spring Security 标签来区分的。这是此页面模板的完整源代码：

HTML 缩小▲ 复制代码

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <title>Login</title>
    <link rel="stylesheet" th:href="@{/assets/bootstrap/css/bootstrap.min.css}"/>
    <link rel="stylesheet" th:href="@{/assets/bootstrap/css/bootstrap-theme.min.css}"/>
    <link rel="stylesheet" th:href="@{/assets/css/index.css}"/>
</head>
<body>
   <div class="container">
      <div th:replace="parts/pieces::logoutForm">
      </div>
        <div th:replace="parts/pieces::header">
        </div>

      <div class="row">
         <div class="col-xs-12">
            <div class="panel panel-default">
               <div class="panel-body">
                  <h3>Index Page</h3>
```

```
                <p>You can see this section as long as you are logged in.</p>
             </div>
          </div>
       </div>

       <div class="row" sec:authorize="hasRole('USER')">
          <div class="col-xs-12">
             <div class="panel panel-default">
                <div class="panel-body">
                   <h3>Only User can See This</h3>
                   <p>If you have the "ROLE_USER".
                   You will be able to see this section. User Section.</p>
                </div>
             </div>
          </div>
       </div>

       <div class="row" sec:authorize="hasRole('STAFF')">
          <div class="col-xs-12">
             <div class="panel panel-default">
                <div class="panel-body">
                   <h3>Only Staff can See This</h3>
                   <p>If you have the "ROLE_STAFF".
                   You will be able to see this section. Staff Section.</p>
                </div>
             </div>
          </div>
       </div>

       <div class="row" sec:authorize="hasRole('ADMIN')">
          <div class="col-xs-12">
             <div class="panel panel-default">
                <div class="panel-body">
                   <h3>Only Admin can See This</h3>
                   <p>If you have the "ROLE_ADMIN".
                   You will be able to see this section. Admin Section.</p>
                </div>
             </div>
          </div>
       </div>

    </div>

    <script type="text/javascript" th:src="@{/assets/jquery/js/jquery.min.js}"></script>
    <script type="text/javascript" th:src="@{/assets/bootstrap/js/bootstrap.min.js}"></script>
</body>
</html>
```

该页面最大的部分是三个部分，它们仅根据用户的安全角色显示。部分显示如下所示：

HTML 复制代码

```
<div class="row" sec:authorize="hasRole('USER')">
...
</div>

<div class="row" sec:authorize="hasRole('STAFF')">
...
</div>

<div class="row" sec:authorize="hasRole('ADMIN')">
...
</div>
```

属性`sec:authorize="hasRole('ADMIN')"`的使用就是Spring Security标签的使用。赋值的表达式（用SpEL写的）很容易理解，当用户具有该表达式定义的角色时，就会使授权为真以显示该部分。这与 Spring Security 与 taglib 一起使用的方式完全相同。惊喜！

这是页面标记的一个有趣部分。另一种方法是将片段添加到页面模板中。这些片段也可以使用 Spring Security 标签。在上述页面标记的顶部，您将看到：

HTML 复制代码

```
<div th:replace="parts/pieces::logoutForm">
</div>
   <div th:replace="parts/pieces::header">
   </div>
```

这些是我从另一个文件插入片段的地方。第一个`<div>`将添加一个新表单。它用于注销用户。另`<div>`一种是在页面顶部插入导航菜单。暂时忘掉表格吧。我稍后会解释。使用fragment添加导航菜单是最常见的操作之一，因为导航菜单可以在多个页面共享，最好一次创建，在所有需要的地方使用。

我定义了名为“ *pieces.html* ”的片段文件。它存储在子文件夹“ *parts* ”中。该文件定义如下：

HTML 缩小▲ 复制代码

```
<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
   <head>
      <meta charset="UTF-8"/>
      <title>Spring Boot Thymeleaf Application - Fragments</title>
   </head>
   <body>
      <div th:fragment="logoutForm">
         <form id="logoutForm" th:action="@{/logout}"
          method="post" th:hidden="true" name="logoutForm">
            <input type="submit" value="Logout" />
         </form>
      </div>
      <div th:fragment="header">
         <div class="row">
            <div class="col-xs-12 text-right">
               <a href="/secure/adminPage">Admin Page</a>
               <a href="/secure/staffPage">Staff Page</a>
               <a href="/secure/userPage">User Page</a>
               <a href="javascript: document.logoutForm.submit();"
                sec:authorize="isAuthenticated()">Logout</a>
            </div>
         </div>
      </div>
      <div th:fragment="logout_header">
         <div class="row">
            <div class="col-xs-12 text-right">
               <a href="javascript: document.logoutForm.submit();"
                sec:authorize="isAuthenticated()">Logout</a>
            </div>
         </div>
      </div>
   </body>
</html>
```

在这个文件中，我创建了三个片段。第一个是将用户注销的表单。再次，请暂时忘记这一点。我将在下一节解释它的作用。另外两个是简单的导航菜单。一个有四个链接。另一个只有一个链接。我想展示的是我使用的另一个 Spring Security 标签，并且经常使用：`sec:authorize="isAuthenticated()"`. 当用户的角色无关紧要时，此标签用于渲染或不渲染元素，只要用户登录，则此标签将`true`始终返回，并允许渲染已标记的元素。

我们来看看`controller`渲染这些页面的类：

爪哇 缩小▲ 复制代码

```
package org.hanbo.boot.app.controllers;

import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

```
import org.springframework.web.servlet.ModelAndView;"

@Controller
public class SecuredPageController
{
   @PreAuthorize("hasRole('USER')")
   @RequestMapping(value="/secure/index", method = RequestMethod.GET)
   public ModelAndView index1()
   {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("indexPage");
        return retVal;
   }

   @PreAuthorize("hasRole('ADMIN')")
   @RequestMapping(value="/secure/adminPage", method = RequestMethod.GET)
   public ModelAndView adminPage()
   {
        ModelAn      <div th:replace="parts/pieces::logoutForm">
      </div>dView retVal = new ModelAndView();
        retVal.setViewName("AdminPage");
        return retVal;
   }

   @PreAuthorize("hasRole('STAFF')")
   @RequestMapping(value="/secure/staffPage", method = RequestMethod.GET)
   public ModelAndView staffPage()
   {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("StaffPage");
        return retVal;
   }

   @PreAuthorize("hasRole('USER')")
   @RequestMapping(value="/secure/userPage", method = RequestMethod.GET)
   public ModelAndView userPage()
   {
        ModelAndView retVal = new ModelAndView();
        retVal.setViewName("UserPage");
        return retVal;
   }
}
```

这些处理用户请求的方法都使用注解`@PreAuthorize("hasRole('<User Role Name>')")`。这意味着，请求必须经过身份验证并具有关联的用户角色才能由后端代码处理。你有没有注意到，在我的`UserAuthenticationService`课中，分配给用户的角色以“ `ROLE_`”为前缀，在 Spring Security 5 及以上版本`ROLE_`中，SPeL`hasRole()`表达式中不再需要前缀“ ”。这是我想透露的最后一点。

最后，我想谈谈CSRF令牌和注销。

CSRF Token 和注销操作

CSRF 是一种攻击方式。您可以在线搜索并阅读所有相关信息。为了防止这种情况，Spring Security 提供了 CSRF 令牌作为缓解机制。也就是说，每次使用页面模板引擎（如 ThymeLeaf）呈现包含表单的页面时，后端 Spring MVC/Spring Security 都会创建一个 CSRF 作为表单上的隐藏字段附加。当表单被提交时，CSRF 将被发送回后端，然后表单将被成功处理。令牌充当前端和后端之间的握手信号。

我相信从 Spring Security 5 开始，启用 CSRF 令牌后，注销只能使用 HTTP 完成`POST`，而不是 HTTP `GET`。这有点棘手，但在网上搜索，我能够找到答案。网上提出的解决方案是创建一个隐藏的表单。它只有一个提交按钮，它调用 HTTP`POST`进行注销。

由于注销表单对所有四个页面都是通用的。我将它定义为一个片段：

HTML 复制代码

```
<div th:fragment="logoutForm">
    <form id="logoutForm" th:action="@{/logout}"
     method="post" th:hidden="true" name="logoutForm">
```

```
    <input type="submit" value="Logout" />
    </form>
</div>
```

它被放置在目标页面上，如下所示：

HTML 复制代码

```
<div th:replace="parts/pieces::logoutForm">
</div>
```

问题是，我如何调用这个表单。它可以通过一行 JavaScript 代码来完成。这里是：

HTML 复制代码

```
<a href="javascript: document.logoutForm.submit();"
sec:authorize="isAuthenticated()">Logout</a>
```

在这里，我调用`document.logoutForm.submit();`. 这将调用表单并执行注销。这是一种非常原始的注销方式。但是对于这个简单的教程，它会做到。问题是，如果我使用 AngularJS 或其他一些 Web 应用程序框架将其替换为单页 Web 应用程序，我会怎么做？对此还没有很好的答案。

无论如何，post 操作不是由我的控制器方法处理的。Spring Security 内部提供了处理方法。但是，如果注销操作成功，则会将页面重定向到*/public/logout*。这个 URL 由我的控制器方法处理，它看起来像这样：

爪哇 复制代码

```
@RequestMapping(value="/public/logout", method = RequestMethod.GET)
public ModelAndView logout()
{
   ModelAndView retVal = new ModelAndView();
   retVal.setViewName("logoutPage");
   return retVal;
}
```

至此，该 Web 应用程序的所有内容都已讨论完毕。接下来，我将讨论如何对此进行测试。

如何测试

下载完整源代码并解压缩后，请将所有*.sj文件重命名回*.js文件。

完成后，请确保已安装 JDK 14 或更高版本。此 Web 应用程序设置为使用 JDK 14 或更高版本编译和打包。使用以下命令编译和打包应用程序：

复制代码

```
mvn clean install
```

构建成功完成后，运行以下命令启动应用程序：

复制代码

```
java -jar target/thymeleaf-security-sample-1.0.0.jar
```

当命令运行时，它会吐出很多日志消息。当您可以确认应用程序已成功启动时，您可以通过导航到以下位置来测试应用程序：

复制代码

```
http://localhost:8080
```

请注意，可能无法形成安全会话。在这种情况下，您可以使用 HTTPS 而不是 HTTP。一些研究是必要的，但很容易做到。祝你好运。

概括

本教程文章讨论了 ThymeLeaf 模板引擎和 Spring Security 的集成。示例应用程序使用基于标准表单的身份验证和基于角色的授权。我为自己写了这个，以便我可以在需要设置使用类似机制的项目时参考。与我之前的一些教程不同，我专注于这个示例应用程序最重要的方面。有些事情在我之前的教程中讨论过。他们很重要。此外，它们是 Spring Security 5 及更高版本的新功能。

在我写这篇文章的时候，我认为用 OAuth 来增强它会很酷，也就是说，使用谷歌身份验证来验证用户登录。这将是一个很好的教程。当我研究这件事时，它似乎很容易解决。困难的部分可能是授权部分。另外，我想知道是否还需要 CSRF 令牌。我不知道。我的另一个问题是我需要做什么才能将 CSRF 令牌添加到 AngularJS 应用程序中。必须有一种方法来获取令牌，然后使用 HTTPPOST方法与后端进行交互。但是这种示例应用程序的设置可能很乏味。所以请继续关注。

除了这两个想法，我还有另一个教程。用于 AngulaJS 应用程序的带有 ES6 语法的 JavaScript。这将是今年的下一个教程。今年我有很多想法。这将是一场精彩的比赛，敬请期待。祝您运行示例应用程序好运。

历史

- 2021 年 2 月15日：初始版本

执照

分享

关于作者

韩博孙

组长 The Judge Group

美国

手表该会员

没有提供传记

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

-- 本论坛暂无消息 --