

Spring Web MVC Security Basic Example Part 2 with Java-based Configuration

Last Updated on 21 January 2016 | [Print](#) [✉ Email](#)

Download Aspose for all your file format manipulation needs

Welcome to part 2 of Spring Web MVC Security tutorial. In the [first part](#), we showed you how to secure a Spring Web MVC application using XML configuration. In this second part, we are going to show you develop the same application as [part 1](#), but use Java configuration instead. No XML at all, even there is no `web.xml` file.

To understand the basics of Spring Security framework and how the sample application works, please refer to part 1:

[Spring Web MVC Security Basic Example Part 1 with XML Configuration](#)

Now, let's discover how to use annotations and Java configuration to secure a Spring Web MVC application.

1. Project Setup

In this tutorial, we use Eclipse IDE to create a dynamic web project, and then convert it to Maven project.

The Spring MVC Security Java Config project is developed using the following pieces of technologies:

- Java 8.
- Tomcat 8 with Servlet 3.1.
- Spring framework 4.2.4.RELEASE.
- Spring Security 4.0.3.RELEASE.
- JSTL 1.2
- Eclipse IDE, Mars Release (4.5.0).

Maven Note: Configuration for Maven dependencies is as same as the [first part](#). Remember the dependencies for Spring Security as below:

```
1  <!-- Spring Security Web -->
2  <dependency>
3      <groupId>org.springframework.security</groupId>
4      <artifactId>spring-security-web</artifactId>
5      <version>${spring.security.version}</version>
6  </dependency>
7
8
9  <!-- Spring Security Config -->
10 <dependency>
11     <groupId>org.springframework.security</groupId>
12     <artifactId>spring-security-config</artifactId>
13     <version>${spring.security.version}</version>
14 </dependency>
```



To see the full Maven dependencies, please refer to part 1: [Spring Web MVC Security Basic Example Part 1 with XML Configuration](#).

2. Coding Index Page

Make a directory called `views` under the `/WEB-INF` directory, then create an `index.jsp` file with the following HTML code:

```
1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2   pageEncoding="ISO-8859-1"%>
3 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
4   "http://www.w3.org/TR/html4/loose.dtd">
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8 <title>Spring Security Basic Demo (Java Config)</title>
9 </head>
10 <body>
11   <div align="center">
12     <h1>Spring Security Basic Demo (Java Config)</h1>
13     <a href="/admin">Go to Administrator Page</a>
14   </div>
15 </body>
16 </html>
```

As you can see, this is very simple page with a heading “*Spring Security Basic Demo (Java Config)*” and a hyperlink to the administrator page which requires authentication to access.

3. Coding Admin Page

Next, create an `admin.jsp` file under the `/WEB-INF/views` directory with the following code:

```
1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
2 <%@ page language="java" session="true" contentType="text/html; charset=ISO-8859-1"
3   pageEncoding="ISO-8859-1"%>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5   "http://www.w3.org/TR/html4/loose.dtd">
6 <html>
7 <head>
8 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
9 <title>Spring Security Basic Demo (Java Config)</title>
10 </head>
11 <body>
12   <div align="center">
13     <h1>${title}</h1>
14     <h2>${message}</h2>
15     <c:if test="${pageContext.request.userPrincipal.name != null}">
16       <h2>Welcome : ${pageContext.request.userPrincipal.name} |
17       <a href="/<c:url value="/logout" />" > Logout</a></h2>
18     </c:if>
19   </div>
20 </body>
21 </html>
```

This is the administrator page which requires authentication and authorization to access. We use JSTL expressions to display the title and message in the model. If the user is logged in, display his username along with a logout link.

4. Coding Spring MVC Controller

Next, we write code for a Spring controller in order to handle requests coming to the application. Create a Java class named `MainController` under the package `net.codejava.spring` with the following code:

```

1 package net.codejava.spring;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RequestMethod;
6 import org.springframework.web.servlet.ModelAndView;
7
8 /**
9  * Spring Web MVC Security Java Config Demo Project
10  * Primary controller of the application.
11  *
12  * @author www.codejava.net
13  *
14  */
15 @Controller
16 public class MainController {
17
18     @RequestMapping(value="/", method = RequestMethod.GET)
19     public ModelAndView visitHome() {
20         return new ModelAndView("index");
21     }
22
23     @RequestMapping(value="/admin", method = RequestMethod.GET)
24     public ModelAndView visitAdmin() {
25         ModelAndView model = new ModelAndView("admin");
26         model.addObject("title", "Administrator Control Panel");
27         model.addObject("message", "This page demonstrates how to use Spring security.");
28
29         return model;
30     }
31 }

```

As you can see, this controller is designed to handle 2 requests:

- “/”: the request to the application’s context root, i.e. the home page.
- “/admin”: the request to the administrator page, which will be secured by Spring security.

The annotations `@Controller` and `@RequestMapping` are used to declare this is a controller which has two HTTP GET handle methods. These annotations will be scanned by Spring as we will configure in the Spring’s application context file.

This book: [Spring in Action](#) helps you learn the latest features, tools, and practices including Spring MVC, REST, Security, Web Flow, and more.

5. Configuring Spring Dispatcher Servlet

Instead of using XML configuration in the `web.xml` file as usual, here we write Java code. Create a Java file called `SpringWebAppInitializer` with the following content:

```

1 package net.codejava.spring;
2
3 import javax.servlet.ServletContext;
4 import javax.servlet.ServletException;
5 import javax.servlet.ServletRegistration;
6
7 import org.springframework.web.WebApplicationInitializer;
8 import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
9 import org.springframework.web.servlet.DispatcherServlet;
10
11 /**
12  * Spring Web MVC Security Java Config Demo Project
13  * Bootstraps Spring Dispatcher Servlet in Servlet 3.0+ environment.
14  *
15  * @author www.codejava.net
16  *
17  */
18 public class SpringWebAppInitializer implements WebApplicationInitializer {
19
20     @Override
21     public void onStartup(ServletContext servletContext) throws ServletException {
22         AnnotationConfigWebApplicationContext appContext = new AnnotationConfigWebApplicationContext();
23         appContext.register(MvcConfig.class);
24
25         ServletRegistration.Dynamic dispatcher = servletContext.addServlet(
26             "SpringDispatcher", new DispatcherServlet(appContext));
27         dispatcher.setLoadOnStartup(1);
28         dispatcher.addMapping("/");
29
30     }
31
32 }

```

Here, the **SpringWebAppInitializer** is a type of **WebApplicationInitializer** which is bootstrapped automatically by any Servlet 3.0+ container (See more: [Bootstrapping a Spring Web MVC application programmatically](#)).

The above code is equivalent to the following XML snippet in the web.xml:

```

1 <servlet>
2   <servlet-name>SpringController</servlet-name>
3   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4   <init-param>
5     <param-name>contextConfigLocation</param-name>
6     <param-value>/WEB-INF/spring-mvc.xml</param-value>
7   </init-param>
8   <load-on-startup>1</load-on-startup>
9 </servlet>
10
11 <servlet-mapping>
12   <servlet-name>SpringController</servlet-name>
13   <url-pattern>/</url-pattern>
14 </servlet-mapping>

```

6. Configuring Spring Security Filter

Spring Security Filter is a servlet filter that intercepts requests for securing the application, so we need to load it upon application's start up.

Instead of using XML, here's Java code to load the Spring Security Filter:

```

1 package net.codejava.spring;
2
3 import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;
4
5 /**
6  * Spring Web MVC Security Java Config Demo Project
7  * Bootstraps Spring Security Filter.
8  *
9  * @author www.codejava.net
10  *
11  */
12 public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {
13
14     public SecurityWebApplicationInitializer() {
15         super(SecurityConfig.class);
16     }
17
18 }

```

It's equivalent to the following XML snippet in the `web.xml` file:

```

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

```

Note that the `SecurityWebApplicationInitializer` class passes `SecurityConfig.class` to its super's constructor, which tells Spring Security looks for configuration in the class named `SecurityConfig` - which is described below.

7. Configuring Authentication and Authorization for the Spring MVC application

Create a class named `SecurityConfig` that extends the `WebSecurityConfigurerAdapter` with the following code:

```

1 package net.codejava.spring;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
6 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
7 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
8 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
9
10 /**
11  * Spring Web MVC Security Java Config Demo Project
12  * Configures authentication and authorization for the application.
13  *
14  * @author www.codejava.net
15  *
16  */
17 @Configuration
18 @EnableWebSecurity
19 public class SecurityConfig extends WebSecurityConfigurerAdapter {
20
21     @Autowired
22     public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
23         auth
24             .inMemoryAuthentication()
25             .withUser("admin").password("nimda").roles("ADMIN");
26     }
27
28     @Override
29     protected void configure(HttpSecurity http) throws Exception {
30
31         http.authorizeRequests()
32             .antMatchers("/").permitAll()
33             .antMatchers("/admin**").access("hasRole('ADMIN')")
34             .and().formLogin();
35
36         http.csrf().disable();
37     }
38 }

```

This is equivalent to the following configuration in XML:

```


1 <http auto-config="true">
2     <intercept-url pattern="/admin**" access="hasRole('ADMIN')" />
3     <csrf disabled="true" />
4 </http>
5
6 <authentication-manager>
7     <authentication-provider>
8         <user-service>
9             <user name="admin" password="nimda" authorities="ADMIN" />
10        </user-service>
11    </authentication-provider>
12 </authentication-manager>

```

Note that we disable CSRF for the sake of simplicity in this tutorial:

```
1 http.csrf().disable();
```

That tells Spring Security to intercept the **/logout** link as an HTTP GET request. In production, you should enable CSRF for best security practice.

 Ads by Google

[▶ Test Java Version](#)

[▶ Java Browser](#)

[▶ Java Source Code](#)

[▶ Spring Security](#)

8. Configuring Spring MVC Application Context

Instead of using XML, we configure the view resolvers in Spring MVC using Java code like this:

```

1 package net.codejava.spring;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.ComponentScan;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.web.servlet.view.InternalResourceViewResolver;
7
8 /**
9  * Spring Web MVC Security Java Config Demo Project
10  * Configures Spring MVC stuffs.
11  *
12  * @author www.codejava.net
13  *
14  */
15 @Configuration
16 @ComponentScan("net.codejava.spring")
17 public class MvcConfig {
18
19     @Bean(name = "viewResolver")
20     public InternalResourceViewResolver getViewResolver() {
21         InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
22         viewResolver.setPrefix("/WEB-INF/views/");
23         viewResolver.setSuffix(".jsp");
24         return viewResolver;
25     }
26 }

```

As you can see, we configure this **InternalResourceViewResolver** bean to resolve logical view names to physical JSP pages. It's equivalent to the following XML:

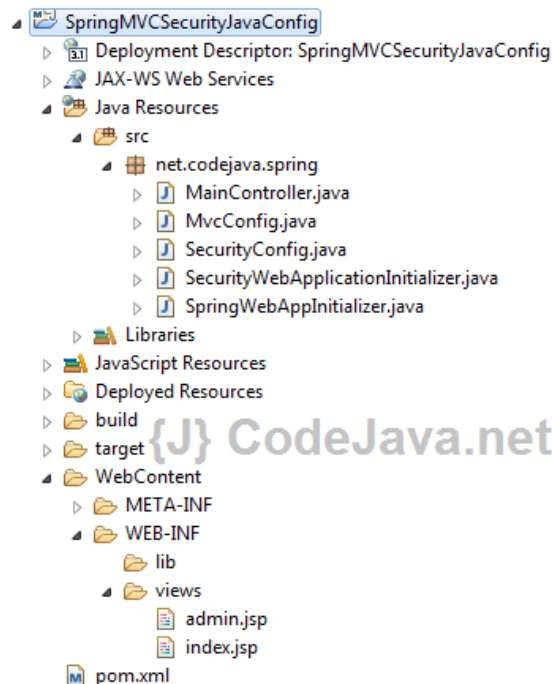
```

1 <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
2     <property name="prefix" value="/WEB-INF/views/" />
3     <property name="suffix" value=".jsp" />
4 </bean>

```

9. The Whole Project Structure

Lastly, the project's structure would look like this in Eclipse IDE:



NOTE: For your convenience, we provide full source code of this Spring MVC Security Java Config Project (Eclipse-Maven) in the Download Attachments section.

Spring Framework 4 Tutorial

Become A Real World Spring Developer Rapidly

10. Test the Spring Security Sample Application

It's time to test the Spring MVC Security application we've built so far. After deploying the project on Tomcat, type the following URL in your browser:

<http://localhost:8080/SpringMVCSecurityJavaConfig>

The home page appears:

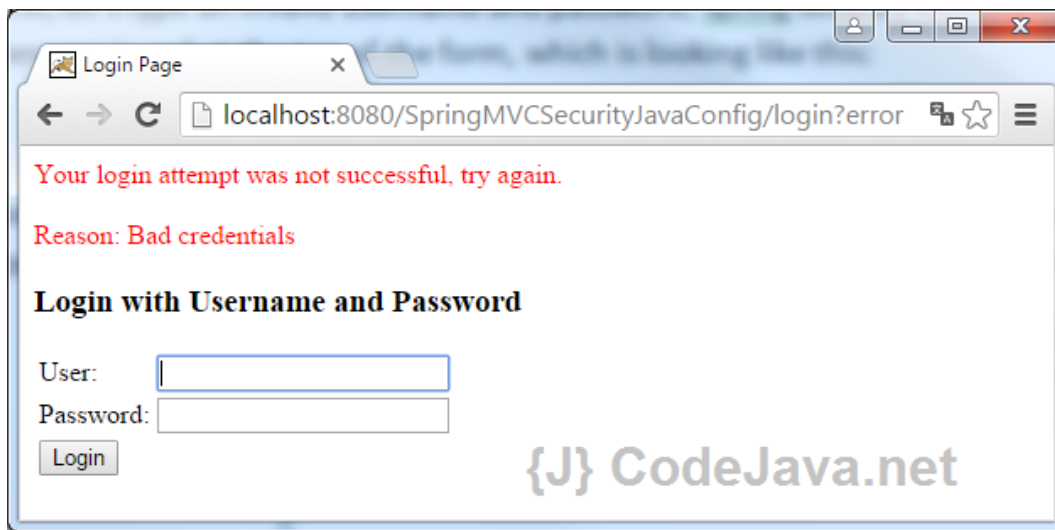


Click on the hyperlink **Go to Administrator Page** (it's equivalent to type the URL:

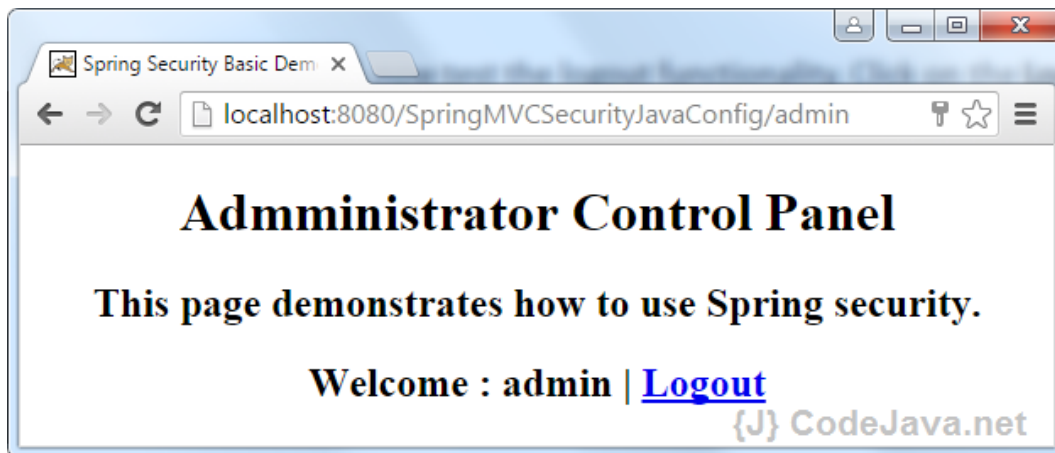
<http://localhost:8080/SpringMVCSecurityJavaConfig/admin>), Spring automatically generates a login page looks like this:



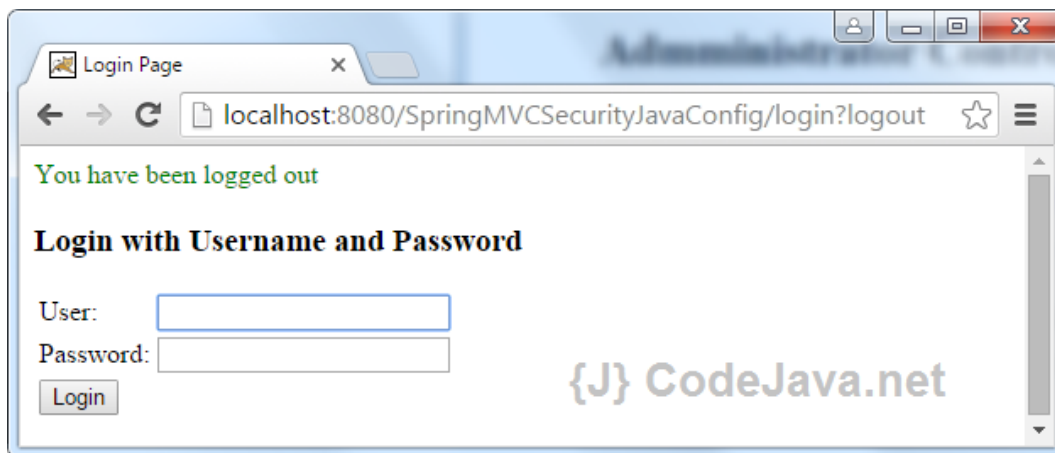
First, let's type an invalid username and password, Spring security automatically generates an error message in red at the top of the form, which is looking like this:



Next, let's type the correct username and password (**admin** and **nimda**, as per configured in this application), Spring security authenticates the login and returns the administrator page as follows:



Awesome! Now, let's test the logout functionality. Click on the **Logout** link, Spring security filter automatically intercepts the **/logout** URL, invalidates the session and take you to the login page again:



Notice the message 'You have been logged out' is displayed in green at the top of the form, and the URL got changed to **http://localhost:8080/ SpringMVCSecurityJavaConfig/login?logout**

To make sure Spring security already invalidated the session, let's try to access the page **http://localhost:8080/SpringMVCSecurityJavaConfig/admin** again, we will see the login page. It's working seamlessly!

Perfect! We've done a basic Spring Web MVC Security project with Java configuration.

References:

- [Spring Security Reference \(4.0.3.RELEASE\)](#)
- [Spring Security Project](#)

You may be also interested in:

- [Spring Web MVC Security Basic Example Part 1 with XML Configuration](#)
- [Understanding the core of Spring framework](#)
- [Understanding Spring MVC](#)
- [Spring MVC beginner tutorial with Spring Tool Suite IDE](#)
- [14 Tips for Writing Spring MVC Controller](#)

Share this article:



THRIVENT
MUTUAL FUNDS

Thrivent Asset Allocation Funds

We make investing easy.

Free Java Beginner Tutorial Videos (8,212+ guys benefited)

EMAIL ADDRESS:

you@domain.com

FIRST NAME:

John

SEND ME

Attachments:

 [SpringMVCSecurityJavaConfig.zip](#) [Spring Web MVC Security Java Config Eclipse-Maven Project] 22 kB