EBook Gratis

APRENDIZAJE spring-security

Free unaffiliated eBook created from
Stack Overflow contributors.

#spring-security

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: spring-security

It is an unofficial and free spring-security ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official spring-security.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Comenzando con la seguridad de primavera

Observaciones

Esta sección proporciona una descripción general de qué es Spring-Security y por qué un desarrollador puede querer usarla.

También debe mencionar cualquier tema importante dentro de la seguridad de primavera, y vincular a los temas relacionados. Dado que la Documentación para Spring-Security es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Versiones

Versión	Fecha de lanzamiento
4.2.2	2017-03-02
3.2.10	2016-12-22
4.2.1	2016-12-21
4.1.4	2016-12-21
4.2.0	2016-11-10

Examples

Instalación o configuración

Instrucciones detalladas para configurar o instalar la seguridad de resorte.

Spring Securitiy para proteger los puntos finales de la API REST

Agregue las siguientes entradas en `pom.xml` .

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>3.1.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>3.1.0.RELEASE</version>
```

```
</dependency>
```

Importante para la versión Spring superior a 3.1:

El error de creación de Bean para `org.springframework.security.filterChains` produce cuando se utiliza la versión Spring superior a 3.1 y no se han agregado dependencias manualmente para `spring-aop` , `spring-jdbc` , `spring-tx` y `spring-expressions` en su `pom.xml` .

Agregue las siguientes entradas en el contexto de Spring. Queremos proteger dos puntos finales REST (helloworld y adiós). Ajustar la versión XSD según la versión Spring.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:security="http://www.springframework.org/schema/security"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
                       http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.1.xsd
                       http://www.springframework.org/schema/security
                       http://www.springframework.org/schema/security/spring-security-
3.1.xsd">

    <security:http auto-config='true' create-session="never">
        <security:intercept-url pattern="/helloworld/**" access="ROLE_USER" />
        <security:intercept-url pattern="/goodbye/**" access="ROLE_ADMIN" />
        <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <security:http-basic />
    </security:http>

    <security:authentication-manager>
        <security:authentication-provider>
            <security:user-service>
                <security:user name="username1" password="password1"
                    authorities="ROLE_USER" />
                <security:user name="username2" password="password2"
                    authorities="ROLE_ADMIN" />
            </security:user-service>
        </security:authentication-provider>
    </security:authentication-manager>
</beans>
```

Agregue las siguientes entradas en `web.xml` .

```
<!-- Spring security-->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
```

```
</listener>

<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:security-context.xml</param-value>
</context-param>
```

Spring-Security usando Spring-boot y JDBC Authentication

Supongamos que quiere evitar que usuarios no autorizados accedan a la página, entonces tiene que ponerles un obstáculo al autorizar el acceso. Podemos hacerlo utilizando Spring-Security, que proporciona autenticación básica al asegurar todos los puntos finales de HTTP. Para eso necesita agregar dependencia de seguridad de primavera a su proyecto, en maven podemos agregar la dependencia como:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Aquí hay una configuración de seguridad que garantiza que solo los usuarios autenticados puedan acceder.

```
@Configuration
@Order(SecurityProperties.ACCESS_OVERRIDE_ORDER)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
DataSource datasource;

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest()
            .fullyAuthenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .failureUrl("/login?error")
            .permitAll()
            .and()
        .logout()
            .logoutUrl("/logout")
            .logoutSuccessUrl("/login?logout")
            .permitAll()
            .and()
        .csrf();
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication().dataSource(datasource).passwordEncoder(passwordEncoder());
}

@Bean
```

```
public PasswordEncoder passwordEncoder() {
    PasswordEncoder encoder = new BCryptPasswordEncoder();
    return encoder;
}
}
```

Configuración	Descripción
`@Configuration`	Indica que el contenedor Spring IoC puede usar la clase como fuente de definiciones de beans.
`@Order (SecurityProperties.ACCESS_OVERRIDE_ORDER)`	Anule las reglas de acceso sin cambiar ninguna otra función configurada automáticamente. Los valores más bajos tienen mayor prioridad.
`WebSecurityConfigurerAdapter`	La clase `SecurityConfig` extiende y anula algunos de sus métodos para establecer algunos detalles específicos de la configuración de seguridad.
`@Autowired` de `DataSource`	Proporcionar fábrica para las conexiones a la fuente de datos físicos.
`configure(HttpSecurity)`	El método anulado define qué rutas de URL se deben proteger y cuáles no.
`.authorizeRequests().anyRequest() .fullyAuthenticated()`	Indica a primavera que toda solicitud a nuestra aplicación requiere ser autenticada.
`.formLogin()`	Configura un inicio de sesión basado en formulario
`.loginPage("/login").failureUrl("/login?error").permitAll()`	Especifica la ubicación de la página de inicio de sesión y todos los usuarios deben tener permiso para acceder a la página.
`.logout().logoutUrl("/logout")` `.logoutSuccessUrl("/login?logout").permitAll()`	La URL a la que redirigir

Configuración	Descripción
	después de que se haya cerrado la sesión. El valor predeterminado es / login? Logout.
`.csrf()`	Usado para prevenir la falsificación de solicitudes entre sitios, la protección CSRF está habilitada (predeterminado).
`configure(AuthenticationManagerBuilder){}`	Método anulado para definir cómo se autentican los usuarios.
`.jdbcAuthentication().dataSource(datasource)`	Indica a la primavera que estamos usando autenticación JDBC
`.passwordEncoder(passwordEncoder())`	Indica que estamos utilizando un codificador de contraseña para codificar nuestras contraseñas. (Se crea un bean para devolver la elección del codificador de contraseña, en este caso estamos utilizando BCrypt)

Tenga en cuenta que no hemos configurado ningún nombre de tabla para utilizar ni ninguna consulta, esto se debe a que la seguridad Spring busca de forma predeterminada las tablas siguientes:

```
create table users (
  username varchar(50) not null primary key,
  password varchar(255) not null,
  enabled boolean not null) ;

create table authorities (
  username varchar(50) not null,
  authority varchar(50) not null,
  foreign key (username) references users (username),
  unique index authorities_idx_1 (username, authority));
```

Inserte las siguientes filas en las tablas anteriores:

```
INSERT INTO authorities(username,authority)
VALUES ('user', 'ROLE_ADMIN');
```

```
INSERT INTO users(username,password,enabled)
VALUES('user', '$2a$10$JvqOtJaDys0yoXPX9w47YOqu9wZr/PkN1dJqjG9HHAzMyu9EV1R4m', '1');
```

El **nombre** de **usuario** en nuestro caso es `user` y la **contraseña** también está cifrada por el `user` con el algoritmo BCrypt

Finalmente, configure una fuente de datos en application.properties para que Spring Boot la use:

```
spring.datasource.url = jdbc:mysql://localhost:3306/spring
spring.datasource.username = root
spring.datasource.password = Welcome123
```

Nota: cree y configure un controlador de inicio de sesión, asígnele un mapa a la ruta `/login` y dirija su página de inicio de sesión a este controlador

Hola seguridad de primavera

> **Nota 1:** Necesita algunos conocimientos previos sobre la ***página del servlet de Java (JSP)*** y ***Apache Maven*** antes de comenzar con estos ejemplos.

Inicie el servidor web (como ***Apache tomcat***) con un proyecto web existente o cree uno.

Visita el ***index.jsp*** .

Cualquiera puede acceder a esa página, es inseguro!

Aplicación de seguridad

1. Actualizar dependencias de Maven

Agregando dependencias a su archivo pom.xml

pom.xml

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>4.0.1.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>4.0.1.RELEASE</version>
</dependency>
```

> **Nota 1:** Si no está utilizando "Spring" en su proyecto anteriormente, no hay dependencia acerca de "Spring-context". Este ejemplo utilizará la configuración xml con "spring-context". Así que añade esta dependencia también.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>4.2.2.RELEASE</version>
</dependency>
```

Nota 2: Si no está utilizando JSTL en su proyecto anteriormente, no hay dependencia al respecto. Este ejemplo utilizará JSTL en la página jsp. Así que añade esta dependencia también.

```
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>javax.servlet.jsp.jstl</artifactId>
  <version>1.2.1</version>
</dependency>
```

---

2. Hacer archivo de configuración de seguridad de Spring

Haga que el nombre de la carpeta sea "spring" dentro de la carpeta "WEB-INF" y cree el archivo security.xml. Copia y pega de los siguientes códigos.

WEB-INF / spring / security.xml

```
<b:beans xmlns="http://www.springframework.org/schema/security"
         xmlns:b="http://www.springframework.org/schema/beans"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         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">

  <http />

  <user-service>
    <user name="stackoverflow" password="pwd" authorities="ROLE_USER" />
  </user-service>

</b:beans>
```

---

3. Actualizar web.xml

Actualice su web.xml dentro de la carpeta "WEB-INF"

WEB-INF / web.xml

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

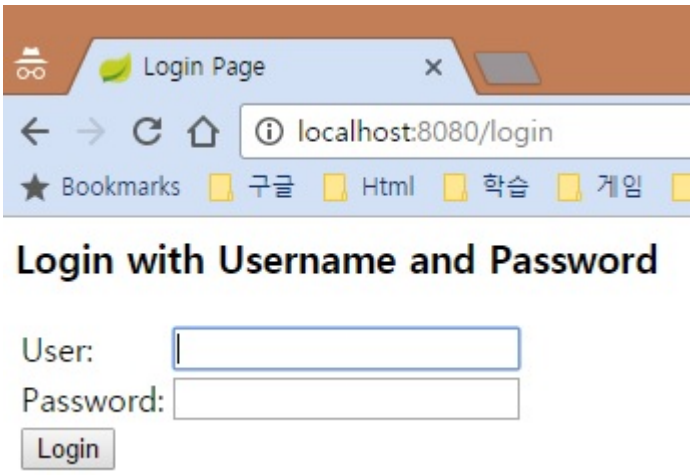
Nota: Si no está utilizando "Spring" en su proyecto anteriormente, no hay configuraciones sobre la carga de los contextos Spring. Así que agrega este parámetro y escucha también.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/spring/*.xml
  </param-value>
</context-param>

<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
```

Ejecutando la aplicación web segura

Después de ejecutar su servidor web y visitar ***index.jsp*** , verá la página de inicio de sesión predeterminada que generó Spring Security. Porque no estás autenticado.



Puedes iniciar sesión

```
username : stackoverflow
password : pwd
```

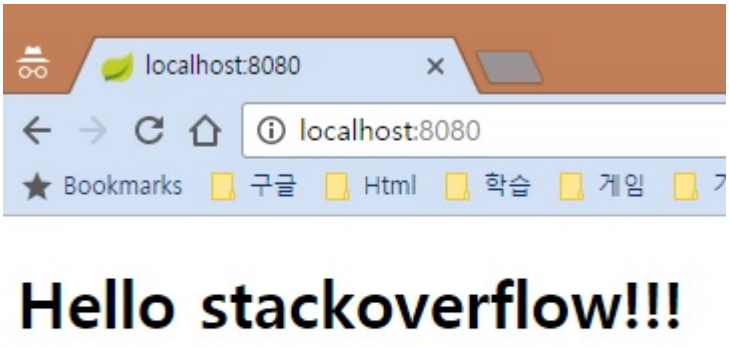
Nota: configuración de nombre de usuario y contraseña en ***WEB-INF / spring / security.xml***

Mostrando el nombre de usuario

Añadiendo etiqueta jstl después del "Hola", que imprime el nombre de usuario

index.jsp

```
<h1>Hello <c:out value="${pageContext.request.remoteUser}" />!!</h1>
```



Hello stackoverflow!!!

Saliendo de tu cuenta

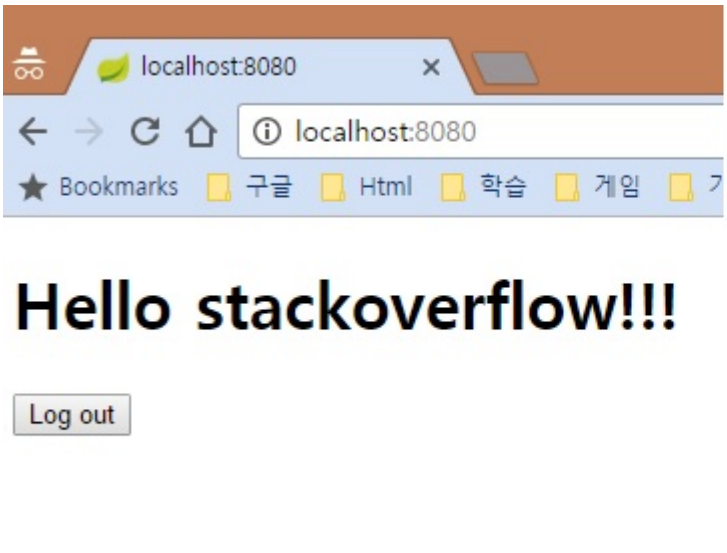
index.jsp

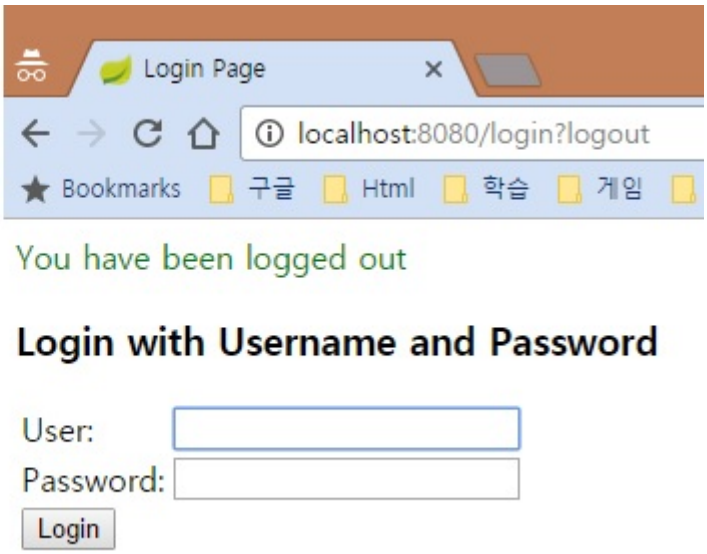
Agregando el formulario, ingrese las etiquetas después de "Hola nombre de usuario", que envía el ***cierre*** de ***sesión*** url ***/ logout*** de Spring Security.

```
<h1>Hello <c:out value="${pageContext.request.remoteUser}" />!!</h1>

<form action="/logout" method="post">
  <input type="submit" value="Log out" />
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
</form>
```

Cuando cierre la sesión correctamente, verá de nuevo la página de inicio de sesión generada automáticamente. Debido a que no están autenticados ahora.





Lea Comenzando con la seguridad de primavera en línea: https://riptutorial.com/es/spring-security/topic/1434/comenzando-con-la-seguridad-de-primavera

Capítulo 2: Configuración de seguridad de primavera con java (no XML)

Introducción

Base de datos típica respaldada, configuración de seguridad de base de anotación base.

Sintaxis

1. configureGlobal () configura el objeto auth.
2. Los dos últimos SQL pueden ser opcionales.
3. El método configure () le dice a spring mvc cómo autenticar la solicitud
4. alguna url no necesitamos autenticar
5. otros redireccionarán a / login si aún no están autenticados.

Examples

Seguridad básica de primavera con anotación, fuente de datos SQL

```
@Configuration
public class AppSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
DataSource dataSource;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
     auth.jdbcAuthentication().dataSource(dataSource)
    .passwordEncoder(new BCryptPasswordEncoder())
    .usersByUsernameQuery("select username,password, enabled from users where username=?")
    .authoritiesByUsernameQuery("select username, role from user_roles where username=?");

}
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.authorizeRequests().antMatchers(".resources/**", "/public/**")
            .permitAll().anyRequest().authenticated().and().formLogin()
            .loginPage("/login").permitAll().and().logout().permitAll();

    }

}
```

Lea Configuración de seguridad de primavera con java (no XML) en línea: https://riptutorial.com/es/spring-security/topic/8700/configuracion-de-seguridad-de-primavera-con-java--no-xml-

Capítulo 3: Configuración de seguridad de Spring

Examples

Configuración

Aquí está la configuración de Java correspondiente:

Agregue esta anotación a una clase `@Configuration` para tener la configuración de Spring Security definida en cualquier `WebSecurityConfigurer` o, más probablemente, extendiendo la clase base `WebSecurityConfigurerAdapter` y anulando métodos individuales:

```
@Configuration
@EnableWebSecurity
@Profile("container")
public class XSecurityConfig extends WebSecurityConfigurerAdapter {
```

enMemoryAuthentication
Define un esquema de autenticación en memoria con un usuario que tiene el nombre de usuario "usuario", la contraseña "contraseña" y el rol "ROLE_USER".

```
  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
     auth
         .inMemoryAuthentication()
             .withUser("user")
             .password("password")
             .roles("ROLE_USER");
  }

  @Override
  public void configure(WebSecurity web) throws Exception {
     web
         .ignoring()
             .antMatchers("/scripts/**","/styles/**","/images/**","/error/**");
  }
```

HttpSecurity
Permite configurar la seguridad basada en web para solicitudes HTTP específicas. De forma predeterminada, se aplicará a todas las solicitudes, pero se puede restringir utilizando `requestMatcher(RequestMatcher)` u otros métodos similares.

```
  @Override
  public void configure(HttpSecurity http) throws Exception {
     http
         .authorizeRequests()
             .antMatchers("/rest/**").authenticated()
             .antMatchers("/**").permitAll()
```

```
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .successHandler(new AuthenticationSuccessHandler() {
                @Override
                public void onAuthenticationSuccess(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    Authentication a) throws IOException, ServletException {
                        // To change body of generated methods,
                        response.setStatus(HttpServletResponse.SC_OK);
                }
            })
            .failureHandler(new AuthenticationFailureHandler() {
                @Override
                public void onAuthenticationFailure(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    AuthenticationException ae) throws IOException, ServletException {
                        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                }
            })
            .loginProcessingUrl("/access/login")
            .and()
        .logout()
            .logoutUrl("/access/logout")
            .logoutSuccessHandler(new LogoutSuccessHandler() {
                @Override
                public void onLogoutSuccess(
                    HttpServletRequest request,
                    HttpServletResponse response,
                    Authentication a) throws IOException, ServletException {
                        response.setStatus(HttpServletResponse.SC_NO_CONTENT);
                }
            })
            .invalidateHttpSession(true)
            .and()
        .exceptionHandling()
        .authenticationEntryPoint(new Http403ForbiddenEntryPoint())
            .and()
        .csrf() //Disabled CSRF protection
            .disable();
  }
}
```

Lea Configuración de seguridad de Spring en línea: https://riptutorial.com/es/spring-security/topic/6600/configuracion-de-seguridad-de-spring

Creditos

S. No	Capítulos	Contributors
1	Comenzando con la seguridad de primavera	Alex78191, AMAN KUMAR, Community, dur, Gnanam, kartik, Panther, sayingu, Xtreme Biker
2	Configuración de seguridad de primavera con java (no XML)	Maxi Wu
3	Configuración de seguridad de Spring	dur, ojus kulkarni