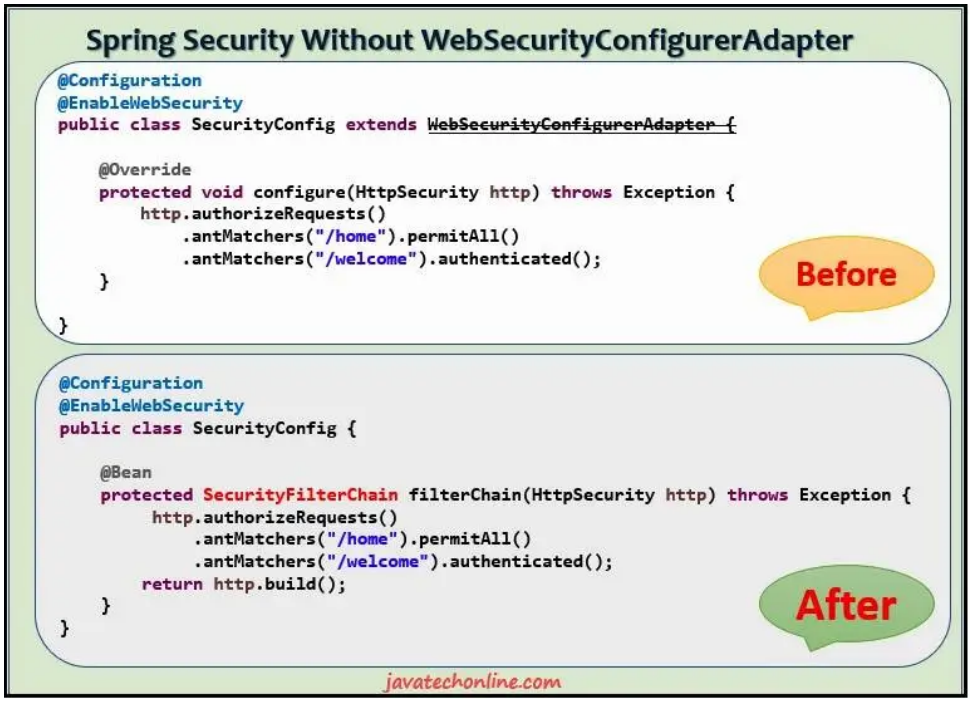
Making Java easy to learn
Java Technology and Beyond

Spring Security Without WebSecurityConfigurerAdapter

java Spring Spring Boot Spring Security *by devs5003 - July 23, 2022* 0

websecurityconfigureradapter deprecated


Spring Security Without WebSecurityConfigurerAdapter
Before
After
javatechonline.com

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends ~~WebSecurityConfigurerAdapter {~~

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/home").permitAll()
            .antMatchers("/welcome").authenticated();
    }
}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    protected SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/home").permitAll()
            .antMatchers("/welcome").authenticated();
        return http.build();
    }
}
```

In the context of the Spring Security module, **WebSecurityConfigurerAdapter** is an abstract class which has been deprecated from Spring Security 5.7.0-M2 as per an announcement posted in the Spring Official website, on 21st Feb, 2022. It was generally used to extend configure() methods by a custom configuration subclass. As a result, it encourages users to move towards a component-based security configuration. To support with the change to this new design of configuration, we will discuss a list of common use-cases and the proposed alternatives going forward. Therefore, we will discuss about the implementation of use cases of Spring Security Without WebSecurityConfigurerAdapter.

It is important to know about this change because sooner or later we will be developing security features using the latest version of Spring Security. Let's discuss the topic 'Spring Security Without WebSecurityConfigurerAdapter' and its related concepts.

Table of Contents

What is WebSecurityConfigurerAdapter?

WebSecurityConfigurerAdapter is an abstract class provided by the Spring Security module. Generally, we use it to override its configure() methods in order to define our security configuration class. Typically, we use two configure() methods with different parameters while implementing Spring Security in our application. One is used to declare authentication related configurations whereas the other one is to declare authorization related configurations. The code looks like below code snippet.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

         // configure Authentication ......
    }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // configure Authorization ......
    }
}
```

Why do we need to learn this change?

If you work with Spring Boot 2.7.0 & maven, it will automatically download Spring Security 5.7.0 or higher version. In that case, you will find WebSecurityConfigurerAdapter deprecated. If you still want to use this class without deprecation, you can change Spring Boot version to lower version(e.g. 2.6.6) in your pom.xml as shown below. It will automatically download Spring Security version lower than 5.7.0 and deprecation warning will disappear. However, if you are not using Spring Boot, but simple Spring Security module, you can even lower the version of Spring Security.

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.6.6</version>
</parent>
```

In this way, we can implement Spring Security Without WebSecurityConfigurerAdapter.

Where do we need to implement this change?

Below are some of the cases where may implement this change.

1) If you are working on Spring Boot 2.7.0 or higher versions

2) If you are working on Spring Security 5.7.0 or higher versions

3) If your project is getting upgraded or migrated to higher versions as described above

4) If you want to customize your Spring Security Configuration using the latest version of Spring Boot

5) If you want to remove annoying warnings of WebSecurityConfigurerAdapter Deprecated

6) If you want to implement Spring Security Without WebSecurityConfigurerAdapter.

How to remove deprecated warning?

Let's learn it step by step given below:

1) If you are using STS as an IDE to develop your project, you need to lower the version of Spring Boot. If you are using Spring Boot version 2.7.0 or higher, make it 2.6.x (e.g. 2.6.6). You can do it by updating pom.xml. Once done, it will download a compatible version of the Spring Security module automatically.

2) Check your configuration class which is extending WebSecurityConfigurerAdapter class. The deprecated warning should disappear. Now you can implement Spring Security Without WebSecurityConfigurerAdapter.

How to work with Spring Security without WebSecurityConfigurerAdapter?

As aforementioned, generally we do inherit configure() methods of WebSecurityConfigurerAdapter in our custom configuration class. Therefore, we need to find out alternative of those methods as WebSecurityConfigurerAdapter deprecated since Spring Security 5.7.0-M2 release.

Let's discuss some examples.

Example#1: With WebSecurityConfigurerAdapter

This example demonstrates the HttpSecurity configuration. Typically, we write it to declare the authorization artifacts.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(HttpSecurity http) throws Exception {
     http.cors().and().csrf().disable()
         .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATEI
         .and().authorizeRequests()
         .antMatchers("/home").permitAll()
         .antMatchers("/welcome").authenticated()
         .antMatchers("/admin").hasAuthority("ADMIN")
         .antMatchers("/emp").hasAuthority("EMPLOYEE")
         .antMatchers("/mgr").hasAuthority("MANAGER")
         .anyRequest().authenticated()
     ;
   }
}
```

Example#1: Without WebSecurityConfigurerAdapter

The below code demonstrates the possible solution of implementing Spring Security Without WebSecurityConfigurerAdapter.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http.cors().and().csrf().disable()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELE
        .and().authorizeRequests()
        .antMatchers("/home").permitAll()
        .antMatchers("/welcome").authenticated()
        .antMatchers("/admin").hasAuthority("ADMIN")
        .antMatchers("/emp").hasAuthority("EMPLOYEE")
        .antMatchers("/mgr").hasAuthority("MANAGER")
        .anyRequest().authenticated();
      return http.build();
   }
}
```

Here, we need to follow below steps:

Step#1: Remove the @override annotation as we are not extending and overriding it from any class.

Step#2: Apply @Bean Annotation on this method

Step#3: Instead of void, now declare the method return type as SecurityFilterChain

Step#4: Update method name whatever you want. Let's say filterChain.

Step#5: At the end of the method body, return http.build(). Here http is the variable of type HttpSecurity.

Example#2: With WebSecurityConfigurerAdapter

In the example below, we have used both methods.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.Authenti
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecur
import org.springframework.security.config.annotation.web.configuration.WebSecurityCon
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

   @Override
   protected void configure(AuthenticationManagerBuilder auth) throws Exception {

      // {noop} => No operation for password encoder (no password encoding needed)
      auth.inMemoryAuthentication()
          .withUser("devs")
          .password ("{noop} devs") //no password encoding needed
          .authorities("ADMIN");

      auth.inMemoryAuthentication().withUser("ns").password("{noop}ns").authorities("E
      auth.inMemoryAuthentication().withUser("vs").password("{noop}vs").authorities("M
   }

   @Override
   protected void configure(HttpSecurity http) throws Exception {

      //declares which Page(URL) will have What access type
      http.authorizeRequests()
          .antMatchers("/home").permitAll()
          .antMatchers("/welcome").authenticated()
          .antMatchers("/admin").hasAuthority("ADMIN")
          .antMatchers("/emp").hasAuthority("EMPLOYEE")
          .antMatchers("/mgr").hasAuthority("MANAGER")
          .antMatchers("/common").hasAnyAuthority("EMPLOYEE","MANAGER")

      // Any other URLs which are not configured in above antMatchers
      // generally declared aunthenticated() in real time
          .anyRequest().authenticated()

      // Login Form Details
         .and()
         .formLogin()
         .defaultSuccessUrl("/welcome", true)

      // Logout Form Details
```

```
            .and()
            .logout()
            .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

          // Exception Details
            .and()
            .exceptionHandling()
            .accessDeniedPage("/accessDenied")
          ;
      }
  }
```

Example#2: Without WebSecurityConfigurerAdapter

The below code demonstrates the possible solution of implementing Spring Security Without WebSecurityConfigurerAdapter.

```
import java.util.ArrayList;
import java.util.List;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecur
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
public class SecurityConfigNew {

    @Bean
    protected InMemoryUserDetailsManager configAuthentication() {

        List<UserDetails> users = new ArrayList<>();
        List<GrantedAuthority> adminAuthority = new ArrayList<>();
        adminAuthority.add(new SimpleGrantedAuthority("ADMIN"));
        UserDetails admin= new User("devs", "{noop}devs", adminAuthority);
        users.add(admin);
        List<GrantedAuthority> employeeAuthority = new ArrayList<>();
        adminAuthority.add(new SimpleGrantedAuthority("EMPLOYEE"));
        UserDetails employee= new User("ns", "{noop}ns", employeeAuthority);
        users.add(employee);

        List<GrantedAuthority> managerAuthority = new ArrayList<>();
        adminAuthority.add(new SimpleGrantedAuthority("MANAGER"));
        UserDetails manager= new User("vs", "{noop}vs", managerAuthority);
        users.add(manager);

        return new InMemoryUserDetailsManager(users);
    }

    @Bean
    protected SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        //declares which Page(URL) will have What access type
        http.authorizeRequests()
            .antMatchers("/home").permitAll()
            .antMatchers("/welcome").authenticated()
            .antMatchers("/admin").hasAuthority("ADMIN")
            .antMatchers("/emp").hasAuthority("EMPLOYEE")
            .antMatchers("/mgr").hasAuthority("MANAGER")
            .antMatchers("/common").hasAnyAuthority("EMPLOYEE","MANAGER")

        // Any other URLs which are not configured in above antMatchers
        // generally declared aunthenticated() in real time
           .anyRequest().authenticated()

        // Login Form Details
           .and()
           .formLogin()
           .defaultSuccessUrl("/welcome", true)

        // Logout Form Details
          .and()
          .logout()
         .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))

        // Exception Details
         .and()
         .exceptionHandling()
        .accessDeniedPage("/accessDenied")
        ;
        return http.build();
    }
}
```

The above code snippet is an example of In-Memory Authentication using Spring Security.

Example#3: With WebSecurityConfigurerAdapter

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowire
    UserDetailsService userDetailsService;

    @Override
    public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) t
        authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEn
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Example#3: Without WebSecurityConfigurerAdapter

The below code demonstrates the possible solution of implementing Spring Security Without WebSecurityConfigurerAdapter.

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    @Bean
    AuthenticationManager authenticationManager(AuthenticationConfiguration authentica
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

Here, in the old version we inject AuthenticationManagerBuilder, set userDetailsService, passwordEncoder and build it. But AuthenticationManager is already created in this step. It is created the way we wanted (with userDetailsService and the passwordEncoder).

Example#4: With WebSecurityConfigurerAdapter

While implementing configuration for web security, the WebSecurityCustomizer is a callback interface that can be used to customize WebSecurity.

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    public void configure(WebSecurity web) {
        web.ignoring().antMatchers("/ignore1", "/ignore2");
    }
}
```

Example#4: Without WebSecurityConfigurerAdapter

Spring Security 5.7.0-M2 onward, the recommended way of doing this is by registering a WebSecurityCustomizer bean. The below code demonstrates the possible solution of implementing Spring Security Without WebSecurityConfigurerAdapter.:

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring().antMatchers("/ignore1", "/ignore2");
    }
}
```

Example#5: With WebSecurityConfigurerAdapter

Below code demonstrates the changes in JDBC authentication in the context of Spring Security.

```
@Configuration
```

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private DataSource dataSource;

    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

       auth.jdbcAuthentication()
           .dataSource(dataSource) //creates database connection
           .usersByUsernameQuery("select user_name,user_pwd,user_enabled from user whe
           .authoritiesByUsernameQuery("select user_name,user_role from user where use
           .passwordEncoder(passwordEncoder);

    }
}
```

Example#5: Without WebSecurityConfigurerAdapter

Going forward, the new code will look like below, if we want to implement Spring Security Without WebSecurityConfigurerAdapter.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

   @Autowired
   private DataSource dataSource;

   @Bean
   public UserDetailsManager authenticateUsers() {

      UserDetails user = User.withUsername("username")
        .password(PasswordEncoderFactories.createDelegatingPasswordEncoder().encode("p
      JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSource);
      users.setAuthoritiesByUsernameQuery("select user_name,user_pwd,user_enabled from
      users.setUsersByUsernameQuery("select user_name,user_role from user where user_n
      users.createUser(user);
```

```
            return users;
        }
    }
```

The new implementation will look like above. It may not be the exact solution.

What are the deprecation updates in Spring Security 6.0.0 or higher?

If you are using Spring Security 6.0.0 or higher (Also Spring Boot 3.0 or higher), you will also see other API deprecations. For example, below are some of the changes need to be made:

authorizeRequests() -> authorizeHttpRequests()

antMatchers() -> requestMatchers()

regexMatchers() -> RegexRequestMatchers()

FAQ

Where to find Complete Examples of 'Spring Security Without WebSecurityConfigurerAdapter?

Below are the links to find complete examples of 'Spring Security Without WebSecurityConfigurerAdapter':

1) How To Implement Security In Spring Boot Project?

2) How to implement Role Based Spring Security Without WebSecurityConfigurerAdapter in Spring Boot using UserDetailsService?

3) How to implement JWT Authentication in Spring Boot Project?

Conclusion

After going through all the theoretical and example part of 'Spring Security Without WebSecurityConfigurerAdapter', finally, we should be ready to handle the warning 'WebSecurityConfigurerAdapter Deprecated' in a real time project. Further, we expect from you to extend the knowledge provided in the article 'Spring Security Without WebSecurityConfigurerAdapter' and implement the concept in your project accordingly. For further learning on Spring Security, you may visit the series of tutorial on Spring Security Using Spring Boot. In addition, If there is any update in the future, we will also update the article accordingly. Moreover, Feel free to provide your comments