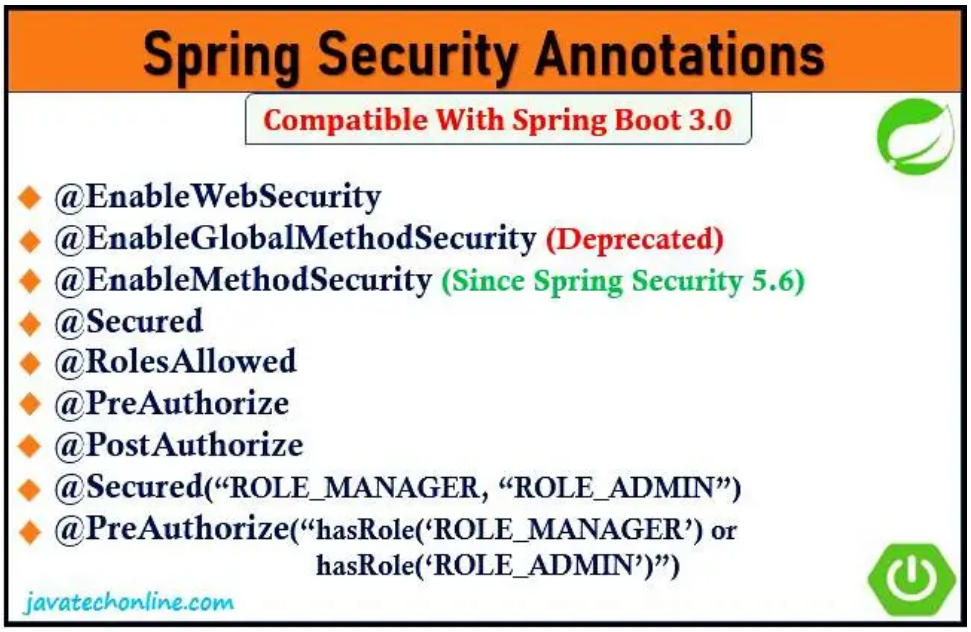
Making Java easy to learn
Java Technology and Beyond

Spring Security Annotations With Examples

java Spring Spring Boot Spring Security *by devs5003 - August 4, 2022* 0



In this article, we will discuss on Spring Security Annotations with examples. Obviously, these annotations can be used in a Spring Boot project as well. These annotations play a crucial role in creating a web application in Spring Boot. If you want to learn all annotations which are generally used in a Spring Boot Project, kindly visit our article 'Spring Boot Annotations with Examples'. Let's discuss about 'Spring Security Annotations with examples' here only.

First of all, in order to use Security related annotations in your Spring Boot project, you need to add security starter dependency. If you created a project using STS (Spring Tool Suite), you have to select 'Spring Security' starter or else add the following dependency in your pom.xml file.

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

@EnableWebSecurity

Spring Boot already provides some basic security configurations. In contrast, while doing customizations on Web Security, we generally create a custom configuration class that extends WebSecurityConfigurerAdapter. That custom configuration class becomes the candidate for applying *@EnableWebSecurity* annotation.

In fact, we use this annotation to enable Spring Security's web security support and provide the Spring MVC integration. Therefore, we can add this annotation to a custom configuration class (in our case, it's MyWebSecurityConfiguration.java) to have the Spring Security configuration defined. We extend our class with any WebSecurityConfigurer or more likely the WebSecurityConfigurerAdapter base class and override individual methods. For example, below code demonstrates the scenario & concept.

Note: Please note that the WebSecurityConfigurerAdapter is deprecated from Spring Security 5.7.0-M2. Hence, we will not be using WebSecurityConfigurerAdapter in our custom configuration class. Instead, there will be a new way to implement the methods in our custom configuration class. Please refer a separate article 'Spring Security Without WebSecurityConfigurerAdapter' on this. However, we will use @EnableWebSecurity as it is.

As in our article we are talking about 'Spring Security Annotations with examples', let's get into one example as below:

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

   @Override
   protected void configure(AuthenticationManagerBuilder auth) {
               // enable in memory based authentication with a user named "user" and "admin"
     auth.inMemoryAuthentication()
          .withUser("user")
          .password("password")
          .roles("USER")
          .and()
          .withUser("admin")
```

```
            .password("password")
            .roles("USER", "ADMIN");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
      http.authorizeRequests().antMatchers("/public/**").permitAll().anyRequest()
            .hasRole("USER").and()
                // Possibly more configuration ...
            .formLogin() // enable form based log in
                // set permitAll for all URLs associated with Form Login
            .permitAll();
    }
              // Possibly more overridden methods ...
}
```

The above example demonstrates the concept of in-Memory database authentication using Spring Security. You may also visit our article for Spring Boot JDBC Authentication example. Moreover, in order to get more details on the role of WebSecurityConfigurerAdapter kindly visit our article Spring Boot Security.

@EnableGlobalMethodSecurity

We can enable annotation-based security using the *@EnableGlobalMethodSecurity* annotation on any *@Configuration* annotated class. By default, global method security is disabled, so if you want to use this functionality, you first need to enable it. Hence, in order to get access of annotations such as *@PreAuthorize, @PostAuthorize, @Secured, @RolesAllowed*, you first need to enable Global Method Security by applying *@EnableGlobalMethodSecurity* annotation to any *@Configuration* annotated java class. For example, below code demonstrates the concept.

```
@EnableWebSecurity
@EnableGlobalMethodSecurity(
      prePostEnabled = true,  // Enables @PreAuthorize and @PostAuthorize
      securedEnabled = true, // Enables @Secured
      jsr250Enabled = true    // Enables @RolesAllowed (Ensures JSR-250 annotations are enabled)
 )
@Configuration
 public class AppSecurityConfig extends WebSecurityConfigurerAdapter {
     @Override
     protected void configure(HttpSecurity http) throws Exception {
```

```
            ....
        }
         // some other overriden methods
    }
```

1) The property ***prePostEnabled=true*** enables support for Spring's *@PreAuthorize* and *@PostAuthorize* annotations. Hence, Spring will ignore this annotation unless you set the flag to true.
2) The property ***securedEnabled=true*** enables support for the *@Secured* annotation. Hence, Spring will ignore this annotation unless you set the flag to true.
3) The property ***jsr250Enabled=true*** enables support for the JSR-250 annotations. One of the most commonly used *@RolesAllowed* annotation comes under this. Hence, Spring will ignore these annotations unless you set the flag to true.

Note that EnableGlobalMethodSecurity still must be included on the class extending GlobalMethodSecurityConfiguration to determine the settings.

Note: If you are using Spring 3.0.0 or a later version, you will get *@EnableGlobalMethodSecurity* as deprecated. You need to use *@EnableMethodSecurity* in place of *@EnableGlobalMethodSecurity*.

@Secured vs @RolesAllowed

We use *@Secured* on a method to specify a list of roles who can access the method. If there are multiple roles, user can access that method if the user has at least one of the specified roles. For example, Let's assume that we created a method getUserDetails() as below:

```
@Secured("ROLE_MANAGER","ROLE_ADMIN")
public String getUserDetails() {
        SecurityContext securityContext = SecurityContextHolder.getContext();
        return securityContext.getAuthentication().getName();
}
```

In the example above, If a user has either Admin or Manager role, that user can access the getUserDetails() method.

We use ***@RolesAllowed*** in a similar way as *@Secured*. The *@RolesAllowed* annotation is the JSR-250's equivalent annotation of the *@Secured* annotation. However, *@Secured* annotation doesn't support SpEL(Spring Expression Language).

@PreAuthorize and @PostAuthorize

If we have a requirement where we want to apply a conditional or expression based access restriction to a method, then we can use *@PreAuthorize* and *@PostAuthorize* annotations. Moreover, we can write expressions using SpEL (Spring Expression Language). The *@PreAuthorize* annotation validates the provided expression before entering into the method. In contrast, the *@PostAuthorize* annotation checks it after the execution of the method and could modify the result.

@PreAuthorize

Here is the example of @PreAuthorize.

```
@PreAuthorize("hasRole('ROLE_MANAGER') or hasRole('ROLE_ADMIN')")
public String getUserDetails() {
        SecurityContext securityContext = SecurityContextHolder.getContext();
        return securityContext.getAuthentication().getName();
}
```

If you compare the example from @Secured annotation, the *@PreAuthorize("hasRole('ROLE_MANAGER') or hasRole('ROLE_ADMIN')")* is another way of writing *@Secured("ROLE_MANAGER","ROLE_ADMIN")*. Consequently,
The *@PreAuthorize("hasRole('ROLE_ADMIN')")* is equivalent to *@Secured("ROLE_ADMIN")* and both have the same meaning.

@PostAuthorize

As aforementioned, *@PostAuthorize* annotation allows the business logic of a method to execute first and only then, the security expression it contains will be evaluated. So, be careful while applying this annotation. It is recommended that do not use this annotation with methods that perform modifying queries like for example Delete User or Update User. The *@PostAuthorize* annotation has access to an object that the method is going to return. We can access the object that the method is going to return in a security expression via the *'returnObject'*. Lets write an example of *@PostAuthorize* using this concept.

```
@PostAuthorize("returnObject.userId == authentication.principal.userId")
public User getUserDetail(String username) {
    return userRepository.getUserByUserName(username);
}
```

In this example, the *getUserDetail( )* method would only execute successfully if the userId of the returned *User* is equal to the current authentication principal's userId.

@EnableMethodSecurity

In Spring Security 5.6, we can enable annotation-based security using the ***@EnableMethodSecurity*** annotation in place of *@EnableGlobalMethodSecurity* on any @Configuration annotated class. It enables *@PreAuthorize, @PostAuthorize, @PreFilter, and @PostFilter* by default and also complies with JSR-250. For example, below code snippets demonstrate the concept of using this annotation.

Example#1: To enable Spring Security's @PreAuthorize annotation

```
@EnableMethodSecurity
@Configuration
public class MySecurityConfig {
     // ...
}
```

We can add an annotation to a method (on a class or interface). It will then limit the access to that method accordingly. For example, observe the below code snippet.

```
public interface PaymentService {

   @PreAuthorize("hasRole('USER')")
   Double readAmount(Long id);

   @PreAuthorize("hasRole('USER')")
   List<Account> findAccount();
 }
```

Example#2: To enable support for Spring Security's @Secured annotation

```
@EnableMethodSecurity(securedEnabled = true)
@Configuration
public class MySecurityConfig {
    // ...
}
```

Example#3: To enable support for JSR-250 annotations

```
@EnableMethodSecurity(jsr250Enabled = true)
@Configuration
public class MySecurityConfig {
    // ...
}
```

This is all about Spring Security Annotations with examples that a developer must know. In order to learn complete security features in Spring Boot, kindly visit our series of articles on Spring Boot Security.

Spring Boot Annotations With Examples
February 20, 2021
In "Spring Boot"

Spring Boot Errors and AOP Annotations With Examples
February 20, 2021
In "Spring Boot"

Spring Boot MVC REST Annotations With Examples
February 20, 2021
In "Spring Boot"

Tagged @enablemethodsecurity @EnableWebSecurity @PostAuthorize @PreAuthorize @preauthorize example @preauthorize vs @secured @RolesAllowed @Secured enableglobalmethodsecurity hasrole spring security preauthorize spring preauthorize spring boot preauthorize spring boot security annotations spring preauthorize spring security annotations spring security preauthorize

‹ Previous article
Spring Boot Interview Questions & Answers

Next article ›
Spring Scheduling Annotations With Examples

Leave a Reply

File Edit View Insert Format Tools Table
Paragraph Font Family Font Sizes Formats