Spring-Security官方文档总结

1. 配置

1.1 SpringBoot集成

1.1.1 Maven

快速引入：

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

如上配置，会直接使用springBoot的BOM中管理的对应版本，若想修改security的版本，可进行如下配置：

```
<properties>
    <spring-security.version>目的版本号</spring-security.version>
</properties>
```

如果不想使用SpringBoot的BOM管理版本，可以使用springSecurity的BOM，来保证项目中security版本一致性：

```
<dependencyManagement>
    <dependencies>
        <!-- ... other dependency elements ... -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-bom</artifactId>
            <version>5.2.0.BUILD-SNAPSHOT</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

1.2 SNAPSHOT仓库

```
<repositories>
    <!-- ... possibly other repository elements ... -->
    <repository>
        <id>spring-snapshot</id>
        <name>Spring Snapshot Repository</name>
        <url>https://repo.spring.io/snapshot</url>
    </repository>
</repositories>
```

1.3 Milestone仓库

```
<repositories>
    <!-- ... possibly other repository elements ... -->
    <repository>
        <id>spring-milestone</id>
        <name>Spring Milestone Repository</name>
        <url>https://repo.spring.io/milestone</url>
    </repository>
</repositories>
```

2. 主要jar包

2.1 spring-security-core.jar

此包，包含了用户认证、权限控制的核心类和接口，提供单例应用、远程客户端、方法权限控制和JDBC用户配置等的支持；
包含的顶级包：

- `org.springframework.security.core`
- `org.springframework.security.access`
- `org.springframework.security.authentication`
- `org.springframework.security.provisioning`

2.2 spring-security-web.jar

任何使用SpringSecurity，基于servelet-api的应用都需要这个包；

2.3 spring-security-config.jar

1. Contains the security namespace parsing code & Java configuration code.

   > You need it if you are using the Spring Security XML namespace for configuration or Spring Security's Java Configuration support.

2. None of the classes are intended for direct use in an application.

   没有一个类是直接使用；

3. 主要类/接口

3.1 springSecurityFilterChain

> `springSecurityFilterChain` which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within your application.

意思：springSecurityFilterChain会拦截一切；

3.2 HttpServletRequest

HttpServletRequest中包含有不少信息，比如用户信息Principal，可自行查看HttpServletRequest源码：

- HttpServletRequest#**getRemoteUser**()
- HttpServletRequest.html#**getUserPrincipal**()
- HttpServletRequest.html#**isUserInRole**(java.lang.String)
- HttpServletRequest.html#**login**(java.lang.String, java.lang.String)
- HttpServletRequest.html#**logout**()

3.3 WebApplicationInitializer

WebApplicationInitializer作用之一就是注册 `springSecurityFilterChain`，为了方便使用，spring提供了一个抽象类**AbstractSecurityWebApplicationInitializer**；在使用与不使用spring框架的情况下，
`AbstractSecurityWebApplicationInitializer` 的运用方式会有所区别；

3.4 HttpSecurity

http认证相关的配置在`WebSecurityConfigurerAdapter`类中的`configure(HttpSecurity http)`方法中配置；代码配置示例：

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .and()
        .httpBasic();
}
```

xml配置示例：

```
<http>
    <intercept-url pattern="/**" access="authenticated"/>
    <form-login />
    <http-basic />
</http>
```

代码配置形式中的`and()`类似于xml中的结束标签（如这里的`</http>`）；

3.4.1 自定义登录页

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .permitAll();
}
```

login.jsp如下：

```
<c:url value="/login" var="loginUrl"/>
<form action="${loginUrl}" method="post">
    <c:if test="${param.error != null}">
        <p>
            Invalid username and password.
        </p>
    </c:if>
    <c:if test="${param.logout != null}">
        <p>
            You have been logged out.
        </p>
    </c:if>
    <p>
        <label for="username">Username</label>
        <input type="text" id="username" name="username"/>
    </p>
    <p>
        <label for="password">Password</label>
        <input type="password" id="password" name="password"/>
    </p>
    <input type="hidden"
        name="${_csrf.parameterName}"
        value="${_csrf.token}"/>
    <button type="submit" class="btn">Log in</button>
</form>
```

关于_csrf：防止跨站请求伪造攻击

> We must the section called “Include the CSRF Token” To learn more read the Section 10.6, “Cross Site Request Forgery (CSRF)” section of the reference

3.4.2 logouts

默认退出应用的URL是`/logout`，它的作用包括：

- Invalidating the HTTP Session
- Cleaning up any RememberMe authentication that was configured
- Clearing the `SecurityContextHolder`
- Redirect to `/login?logout`

3.4.3 自定义退出页面

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .logout()
            .logoutUrl("/my/logout")           // 1
            .logoutSuccessUrl("/my/index")     // 默认/login?logout
            .logoutSuccessHandler(logoutSuccessHandler)  // 2
            .invalidateHttpSession(true)
            .addLogoutHandler(logoutHandler)   // 3
            .deleteCookies(cookieNamesToClear) // 4
            .and()
        ...
}
```

代码解释：

1. 真正触发退出的URL

   > The URL that triggers log out to occur (default is `/logout`). If CSRF protection is enabled (default), then the request must also be a POST. For more information, please consult the JavaDoc.

2. 成功退出后的动作，如果配置了此项，那么`logoutSuccessUrl`就会失效；

   实际上，logoutSuccessUrl是`logoutSuccessHandler(SimpleUrlLogoutSuccessHandler)`的快捷方式。如果是restful风格的应用，最好使用`HttpStatusReturningLogoutSuccessHandler`：

   > The `HttpStatusReturningLogoutSuccessHandler` can be interesting in REST API type scenarios. Instead of redirecting to a URL upon the successful logout, this `LogoutSuccessHandler` allows you to provide a plain HTTP status code to be returned. If not configured a status code 200 will be returned by default.

3. 添加退出操作步骤，默认只有SecurityContextLogoutHandler（处理session等一系列退出操作）：

   > Adds a `LogoutHandler`. `SecurityContextLogoutHandler` is added as the last `LogoutHandler` by default.

4. 清除特定的cookie：

   > Allows specifying the names of cookies to be removed on logout success. This is a shortcut for adding a `CookieClearingLogoutHandler` explicitly.

3.5 LogoutHandler

LogoutHandler代表退出处理，LogoutHandler被设计用来清理退出时数据，因此，不应该再LogoutHandler中抛出异常，它的一些常用实现类：

- PersistentTokenBasedRememberMeServices
- TokenBasedRememberMeServices
- CookieClearingLogoutHandler
- CsrfLogoutHandler
- SecurityContextLogoutHandler

3.6 AuthenticationProvider

可自行实现AuthenticationProvider接口，自行定义用户的鉴定；

```
@Bean
public SpringAuthenticationProvider springAuthenticationProvider() {
    return new SpringAuthenticationProvider();
}
```

3.7 UserDetailsService

自定义用户信息获取方式：

```
@Bean
public SpringDataUserDetailsService springDataUserDetailsService() {
    return new SpringDataUserDetailsService();
}
```

但是，这种方式有条件要求：

> This is only used if the `AuthenticationManagerBuilder` has not been populated and no `AuthenticationProviderBean` is defined.

3.8 PasswordEncoder

自定义密码加密器：

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

3.9 EnableGlobalMethodSecurity

SpringSecurity支持方法（包括service方法）、bean级别的安全策略；

EnableGlobalMethodSecurity要配合@Configuration一起使用：

> We can enable annotation-based security using the `@EnableGlobalMethodSecurity` annotation on any `@Configuration` instance.

```
@EnableGlobalMethodSecurity(securedEnabled = true)
public class MethodSecurityConfig {
// ...
}
```

配置后，可以在方法上使用注解，进行权限控制：

```
public interface BankService {

@Secured("IS_AUTHENTICATED_ANONYMOUSLY")
public Account readAccount(Long id);

@Secured("IS_AUTHENTICATED_ANONYMOUSLY")
public Account[] findAccounts();

@Secured("ROLE_TELLER")
public Account post(Account account, double amount);
}
```

方法上的注解配置，实际进行权限判定的是`AccessDecisionManager`类：

> These will be passed to the AccessDecisionManager for it to make the actual decision.

@EnableGlobalMethodSecurity直接注释在普通类上，作用是开启默认GlobalMethod配置，如果想自行实现GlobalMethod的handler，可以继承GlobalMethodSecurityConfiguration类，重新实现其中的方法，如下，定制化MethodSecurityExpressionHandler：

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {
    @Override
    protected MethodSecurityExpressionHandler createExpressionHandler() {
        // ... create and return custom MethodSecurityExpressionHandler ...
        return expressionHandler;
    }
}
```

4. Oauth 2.0

扩展

专业名词

- JSON Web Token (JWT)
- JSON Web Signature (JWS)
- JSON Web Encryption (JWE)
- JSON Web Key (JWK)
- Pre-Authentication - 一种用户和角色由`容器`来管理的安全策略模式