HttpSecurity工作原理

1. 使用了什么设计模式

2. authorizeRequests()啥意思

3. and()啥意思，跟xml中的闭合标签一致？，有and()与没有有啥区别?

```
https://spring.io/blog/2013/07/11/spring-security-java-config-preview-readability/:
By formatting our Java configuration code it is much easier to read. It can be read similar to the XML namespace equ

// 重点
Each child of authorizeRequests is similar to each <intercept-urls>
```

4. WebSecurity 与 HttpSecurity 区别?

5. antmatcher中多个不同url书写顺序有何影响?

6. ExpressionInterceptUrlRegistry是怎么工作的?

   authorizeRequests() 所创建的 RequestMatcher 是有顺序的？？

7. DefaultSecurityFilterChain ?

8. HttpSecurity 的dsl如何理解？多个authorizeRequests有啥区别？？

```
@Override
    protected void configure(HttpSecurity http) throws Exception {
        http.servletApi()
        .and().csrf().disable()
  .authorizeRequests().filterSecurityInterceptorOncePerRequest(false)
.and()           .anonymous().disable().sessionManagement()
.and() .formLogin().disable()        .authorizeRequests().anyRequest().authenticated()
.and().headers().frameOptions().sameOrigin().and()
.authorizeRequests().antMatchers("/error").permitAll().antMatchers("/**").authenticated()
.and().addFilterBefore(new RestOpenAmSpringFilter(openAmAuthenticationManager, restSsoClient), UsernamePasswordAuthe
}
```

9. 配置Demo

```
public static class ApiWebSecurityConfigurerAdapter extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()
                .addFilterBefore(new RestOpenAmSpringFilter(openAmAuthenticationManager, restSsoClient), UsernamePas
                .exceptionHandling()
                .authenticationEntryPoint((request, response, authException) -> {
                    log.debug("Authentication required.");

                    response.setHeader("X-Authenticate", properties.getLoginUrl());
                    response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Authentication required");
                }).and()

                .antMatcher("/api/**")
                .authorizeRequests().anyRequest()
                .authenticated()
        ;
```

```
        }
    }
```

10. 如何保护请求资源

通过authenticated()和permitAll()来定义该如何保护路径。authenticated()要求在执行该请求时，
必须已经登录了应用。如果用户没有认证的话，Spring Security的Filter将会捕获该请求，并将用户重定向到应用的登录页面。
同时，permitAll()方法允许请求没有任何的安全限制。

除了authenticated()方法和permitAll()方法外,还有一些其他方法用来定义该如何保护请求.

access(String) 如果给定的SpEL表达式计算结果为true，就允许访问
anonymous() 允许匿名用户访问
authenticated() 允许认证的用户进行访问
denyAll() 无条件拒绝所有访问
fullyAuthenticated() 如果用户是完整认证的话（不是通过Remember-me功能认证的），就允许访问
hasAuthority(String) 如果用户具备给定权限的话就允许访问
hasAnyAuthority(String…)如果用户具备给定权限中的某一个的话，就允许访问
hasRole(String) 如果用户具备给定角色(用户组)的话,就允许访问/
hasAnyRole(String…) 如果用户具有给定角色(用户组)中的一个的话,允许访问.
hasIpAddress(String 如果请求来自给定ip地址的话,就允许访问.
not() 对其他访问结果求反.
permitAll() 无条件允许访问
rememberMe() 如果用户是通过Remember-me功能认证的，就允许访问

11. spring 文档解释

For a more concrete example, take a look at the following code:

COPY

```
http
    // #1
    .formLogin()
        // #2
        .loginPage("/login")
        .failureUrl("/login?error")
        // #3
        .and()
    // #4
    .authorizeRequests()
        // #5
        .antMatchers("/signup","/about").permitAll()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated();
```

- **#1** `formLogin` updates the `http` object itself. The indentation of `formLogin` is incremented from that of `http` (much like they way the `<form-login>` is indented from `<http>`)
- **#2** `loginPage` and `failureUrl` update the `formLogin` configuration. For example, `loginPage` determines where Spring Security will redirect if log in is required. For this reason, each is a child of `formLogin`.
- **#3** `and` means we are done configuring the parent (in this case `formLogin`). This also implies that the next line will decrease indentation by one. When looking at the configuration you can read it as `http` is configured with `formLogin` **and** `authorizeRequests`. If we had nothing else to configure, the `and` is not necessary.
- **#4** We decrease the indentation with `authorizeRequests` since it is not related to form based log in. Instead, its intent is to restrict access to various URLs.
- **#5** each `antMatchers` and `anyRequest` modifies the authorization requirements for `authorizeRequests`. This is why each is a child of `authorizeRequests`

```
http
    .formLogin()
        .loginPage("/login")
        .failureUrl("/login?error")
        .and()
    .authorizeRequests()
        .antMatchers("/signup", "/about").permitAll()
        .antMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated();
```

The relevant, but not equivalent, XML configuration can be seen below. Note that the differences between how Spring Security will behave between these configurations is due to the different default values between Java Configuration and XML configuration.

```
<http use-expressions="true">
  <form-login
      login-page="/login"
      authentication-failure-url="/login?error"
  /> <!-- similar to and() -->
  <intercept-url pattern="/signup" access="permitAll"/>
  <intercept-url pattern="/about" access="permitAll"/>
  <intercept-url pattern="/**" access="hasRole('ROLE_USER')"/>
</http>
```

- The first thing to notice is that the `http` and `<http>` are quite similar. One difference is that Java Configuration uses `authorizeRequests` to specify `use-expressions="true"`
- `formLogin` and `<form-login>` are quite similar. Each child of `formLogin` is an XML attribute of `<form-login>`. Based upon our explanation of indentation, the similarities are logical since XML attributes modify XML elements.
- The `and()` under `formLogin` is very similar to ending an XML element.
- Each child of `authorizeRequests` is similar to each `<intercept-urls>`, except that Java Configuration specifies requires-channel differently which helps reduce configuration in many circumstances.

Xml配置与Java-Config配置

```
<http security="none" pattern="/resources/**"/>
<http use-expressions="true">
  <intercept-url pattern="/logout" access="permitAll"/>
  <intercept-url pattern="/login" access="permitAll"/>
  <intercept-url pattern="/signup" access="permitAll"/>
  <intercept-url pattern="/about" access="permitAll"/>
  <intercept-url pattern="/**" access="hasRole('ROLE_USER')"/>
  <logout
      logout-success-url="/login?logout"
      logout-url="/logout"
  />
  <form-login
      authentication-failure-url="/login?error"
      login-page="/login"
      login-processing-url="/login"
      password-parameter="password"
      username-parameter="username"
  />
</http>
<authentication-manager>
  <authentication-provider>
    <user-service>
      <user name="user"
          password="password"
          authorities="ROLE_USER"/>
      <user name="admin"
          password="password"
          authorities="ROLE_USER,ROLE_ADMIN"/>
```

```
    </user-service>
  </authentication-provider>
</authentication-manager>


@EnableWebSecurity
@Configuration
public class CustomWebSecurityConfigurerAdapter extends
   WebSecurityConfigurerAdapter {
  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder auth) {
    auth
      .inMemoryAuthentication()
        .withUser("user")  // #1
          .password("password")
          .roles("USER")
          .and()
        .withUser("admin") // #2
          .password("password")
          .roles("ADMIN","USER");
  }

  @Override
  public void configure(WebSecurity web) throws Exception {
    web
      .ignoring()
         .antMatchers("/resources/**"); // #3
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http
      .authorizeUrls()
        .antMatchers("/signup","/about").permitAll() // #4
        .antMatchers("/admin/**").hasRole("ADMIN") // #6
        .anyRequest().authenticated() // 7
        .and()
    .formLogin()  // #8
        .loginUrl("/login") // #9
        .permitAll(); // #5
  }
}
```