spring-security研究.详细教程

相关注解和类

注解

- `@EnableWebSecurity`
- `@EnableGlobalMethodSecurity`
- `@EnableGlobalAuthentication`

类/接口

- `WebSecurityConfigurerAdapter` 实现 `WebSecurityConfigurer` 继承 `SecurityConfigurer`
- `SecurityBuilder`
- `ProviderManager` 实现 `AuthenticationManager`
- `AccessDecisionManager` 、 `GlobalMethodSecurityConfiguration`
- `ObjectPostProcessor`
- `SecurityContextHolder` 、 `SecurityContext` 、 `Authentication`
- `AbstractSecurityInterceptor`
- `SessionRegistry` 、 `SessionManagementConfigurer` 、 `SessionAuthenticationStrategy`

知识点

- ** `WebSecurityConfiguration` **类的文档说明详细地表明了 `Spring-security` 的定制化配置是如何导入的；
- 只继承 `AbstractSecurityWebApplicationInitializer` 类，便可以继承 `spring-security` （前提是使用了 `spring-mvc` 框架）

```
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

// 不用任何注解,然后SecurityConfig就会被自动注入到应用中，SecurityConfig需要注解
public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {

}
```

`SecurityConfig` 需要注解

```
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

- `security` 的配置方式可以有2种（都需要 `@EnableWebSecurity` 注解）：

```
// spring doc
    Customizations can be made to WebSecurity by extending WebSecurityConfigurerAdapter and
exposing it as a
Configuration or implementing WebSecurityConfigurer and exposing it as a Configuration. This
configuration is
imported when using EnableWebSecurity.
```

- 继承 `WebSecurityConfigurerAdapter` 类
- 继承 `WebSecurityConfigurer` 类

- **过滤器链**
  - `@EnableWebSecurity` 注解中引入了 `WebSecurityConfiguration` 类（ `@Import` ），`WebSecurityConfiguration.springSecurityFilterChain()` 方法生成过滤器链；
  - `WebSecurityConfiguration` 创建 `WebSecurity` ，然后， `WebSecurity` 的 `build()` 方法创建 `FilterChainProxy` (是 `Filter` 的子类)
  - 根据 `servlet3.0` 规范， `servlet容器` 要负责创建 `/META-INF/services/javax.servlet.ServletContainerInitializer` (spring-web包中)中定义的类
  - 过滤器链的注册： `servlet容器` 会加载 `SpringServletContainerInitializer` 类(`ServletContainerInitializer` 的子类）， `SpringServletContainerInitializer` 通过 `@HandlesTypes` 注解会引入 `WebApplicationInitializer` 的实现并在 `onStartup()` 方法中
    调用 `WebApplicationInitializer.onStartup()` 方法，而属于 `security` 的 `WebApplicationInitializer` 类(`AbstractSecurityWebApplicationInitializer`) 就在这里被执行， `security` 的过滤器链就是在 `AbstractSecurityWebApplicationInitializer.onStartup()` 中被注册的；

- **配置类 `WebSecurityConfigurerAdapter` 的加载过程**
  - `SecurityConfigurer` （ `WebSecurityConfigurerAdapter` 的父接口）被加载后，会先执行 `init(securityBuilder)` 方法，然后调用 `configure(SecurityBuilder)` 方法；
    - `WebSecurityConfigurerAdapter` 中的2个方法（ `configure(AuthenticationManagerBuilder)` 和 `configure(HttpSecurity)` ）就是在 `init(securityBuilder)` 中执行的
    - `configure(SecurityBuilder)` 是在 `builder.doBuild()` 调用时执行的；(`doBuild()` 方法时在 `@EnableWebSecurity` → `WebSecurityConfiguration.springSecurityFilterChain()` 中执行)

- `Builder` **和** `Configurer`
  - `SecurityBuilder` 需要用到 `SecurityConfigurer` 来生成过滤器链；
  - `http.formLogin()` 、 `http.httpBasic()` 等，其实返回的就是 `SecurityConfigurer`
  - 每个 `SecurityBuilder` 只能处理部分 `SecurityConfigurer` ，所以会有多种 `config(builder)` 配置（如： `configure(HttpSecurity http)` 、 `configure(AuthenticationManagerBuilder)` 等
  - `HttpSecurity` （builder）能够处理的 `SecurityConfigurer` 都在它的源码中；对于其他的 `SecurityBuilder` 实现类也是类似，其支持的 `SecurityConfiguer` 都定义在自己的源码中

- **FormLogin** 传送门
  - 默认提供的登录页面是 `DefaultLoginPageGeneratingFilter` 类的 `generateLoginPageHtml()` 方法生成的；
  - 是否注册默认登录页面，可参考 `FormLoginConfigurer#initDefaultLoginFilter`

- **注销**(访问 `/logout`)，会有以下默认动作：
  - 使 `HttpSession` 失效
  - 清空已配置的 `RememberMe` 验证
  - 清空 `SecurityContextHolder`
  - 重定向到 `/login`

- ** `ProviderManager` **是管理验证提供者（jdbc、ldap等）；

    - `ProviderManager` 管理所有的 `AuthenticationProvider` （ `ProviderManager` 属性 `List<AuthenticationProvider> providers` ），身份鉴定的时候 `ProviderManager` 中管理的 `provider` 会被挨个的调用，直到有一个 `provider` 可以provide（提供）一个非空的回应（ `non-null response` ，非空就表明 `provider` 已经可以对请求进行身份认证）
    - 如果前一个 `provider` 不能提供 `non-null response` 或抛出 `AuthenticationException` ，流程会继续下一个 `provider` ，直到得到一个 `non-null response`
    - 如果所有 `provider` 都不能提供一个 `non-null response` ，那么就会抛出最近的一个 `AuthenticationException`
    - 如果所有的 `provider` 都没有提供 `non-null response` ，也没有抛出 `AuthenticationException` ，那么 `ProviderManager` 会抛出 `ProviderNotFoundException`
    - `AuthenticationManager` 可以设置父 `manager` ，当前面的几步中没有获取到 `non-null response` ，父 `manager` 会被调用；
    - `ProviderManager` 中有个事件分发器 `AuthenticationEventPublisher` ，默认是个空分发器 `new NullEventPublisher()` ，如果需要，用户必须手动设置事件分发器；
        - 可以使用 `auth.authenticationEventPublisher(authenticationEventPublisher());` 添加自定义的publisher
    - 父 `manager` 中的事件也会被 `AuthenticationEventPublisher` 发布；
- `GlobalMethodSecurityConfiguration` 是全局方法权限鉴定的基本类，如果想实现定制化，可以继承这个类，但是必须保证定制类上有 `@EnableGlobalMethodSecurity` 注解
- 多 `HTTPSecurity` 配置：传送门
- 继承 `GlobalMethodSecurityConfiguration` 可以实现更加复杂的方法级认证。传送门

知识碎片

- `.antMatchers( "/admin/**").hasRole("ADMIN" )` 中的 `hasRole` 不需要加 `ROLE_` 前缀；
- 路径匹配方式有ant(`antMatchers`)和regex(`regexMatchers`)方式：Ant语法
- 触发注销操作的url，默认是 `/logout` 。如果开启了 `CSRF` 保护(默认开启),那么请求必须是 `POST` 方式。
- 在 `Restful` 模式下， `HttpStatusReturningLogoutSuccessHandler` 可以实现非重定向，返回一个纯文本的状态码；
- 获取当前用户

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();

if (principal instanceof UserDetails) {
String username = ((UserDetails)principal).getUsername();
} else {
String username = principal.toString();
}
```

- `spring-security` 关于 `crsf` 的解决方案 传送门
    - 应用中解决方法
- cookie中保存的 `RememberMe` 的值是 `PersistentRememberMeToken` 类中的属性进行base64计算(`Base64(series + ":" + tokenValue)（去掉末尾的等号）`)

遗留问题

- `ObjectPostProcessor`