Spring Security

概述

Spring Security 是一个灵活和强大的身份验证和访问控制框架，以确保基于 Spring 的 Java Web 应用程序的安全。

Spring Security 是一个轻量级的安全框架，它确保基于 Spring 的应用程序提供身份验证和授权支持。它与 Spring MVC 有很好地集成，并配备了流行的安全算法实现捆绑在一起//原文出自【易百教程】,商业转载请联系作者获得授权,非商业请保留原文链接:https://www.yiibai.com/spring-security/

Spring Security 实现了一系列的过滤器链，就按照下面顺序一个一个执行下去。

```
     <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>4.2.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>4.2.2.RELEASE</version>
    </dependency>
```

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

1. shiro: shiro 是一套权限管理框架，包括认证、授权等，在使用时直接写相应的接口（小而简单的 Shiro 就足够）
2. Spring Security: 基于 Spring 的企业应用系统提供声明式的安全访问控制解决方案的安全框架；可以在 Spring 应用上下文中配置的 Bean，充分利用了 Spring IoC，DI
3. jwt: jwt 只是一种生成 token 的机制，所有权限处理逻辑还需要自己写（主要优势在于使用无状态、可扩展的方式处理应用中的用户会话）
4. oauth2: 是一种安全的授权框架，提供了一套详细的授权机制。用户或应用可以通过公开的或私有的设置，授权第三方应用访问特定资源。(如果不介意 API 的使用依赖于外部的第三方认证提供者，你可以简单地把认证工作留给认证服务商去做。)

Apache Shiro 是 Java 的一个安全框架。目前，使用 Apache Shiro 的人越来越多，因为它相当简单，对比 Spring Security，可能没有 Spring Security 做的功能强大，但是在实际工作时可能

并不需要那么复杂的东西，所以使用小而简单的 Shiro 就足够了

Shiro 三个核心组件：Subject，SecurityManager 和 Realms.

Subject：主体，代表了当前“用户”，这个用户不一定是一个具体的人，与当前应用交互的任何东西都是 Subject,如网络爬虫,机器人等;即一个抽象概念;所有 Subject 都绑定到 SecurityManager，与 Subject 的所有交互都会委托给 SecurityManager；可以把 Subject 认为是一个门面；SecurityManager 才是实际的执行者；

SecurityManager：安全管理器；即所有与安全有关的操作都会与 SecurityManager 交互；且它管理着所有 Subject；可以看出它是 Shiro 的核心，它负责与后边介绍的其他组件进行交互，如果学习过 SpringMVC，你可以把它看成 DispatcherServlet 前端控制器；

Realm：域，Shiro 从从 Realm 获取安全数据（如用户、角色、权限），就是说 SecurityManager 要验证用户身份，那么它需要从 Realm 获取相应的用户进行比较以确定用户身份是否合法；也需要从 Realm 得到用户相应的角色/权限进行验证用户是否能进行操作；可以把 Realm 看成 DataSource，即安全数据源。

Apache Shiro 是 Java 的一个安全框架，在轻量级的程序应用更广泛，简化了 Spring security 的功能，提供了处理身份认证，授权，企业会话管理和加密的功能。

应用领域

1. 验证用户
2. 对用户执行访问控制，如：
   判断用户是否拥有角色 admin。
   判断用户是否拥有访问的权限
3. 在任何环境下使用 Session API。例如 CS 程序。
4. 可以使用多个用户数据源。例如一个是 oracle 用户库，另外一个是 mysql 用户库。
5. 单点登录（SSO）功能。
6. “Remember Me”服务 ，类似购物车的功能，shiro 官方建议开启

核心领域：

1. 身份验证 Authentication（重要模块）
2. 授权 Authorization（重要模块）
3. 会话管理 Session Management（重要模块）
4. 加密 Cryptography（重要模块）
5. web support：Web 支持，可以非常容易的集成到 Web 环境(次要)
6. Caching：缓存，比如用户登录后，其用户信息、拥有的角色/权限不必每次去查，这样可以提高效率；(次要)
7. Concurrency：shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；(次要)
8. Testing：提供测试支持；(次要)
9. Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；(次要)
10. Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了(次要)

Spring Security 和 Shiro

相同点：

1：认证功能

2：授权功能

3：加密功能

4：会话管理

5：缓存支持

6：rememberMe 功能.....

不同点：

优点：

1：Spring Security 基于 Spring 开发，项目中如果使用 Spring 作为基础，配合 Spring Security 做权限更加方便，而 Shiro 需要和 Spring 进行整合开发

2：Spring Security 功能比 Shiro 更加丰富些，例如安全防护

3：Spring Security 社区资源比 Shiro 丰富

缺点：

1：Shiro 的配置和使用比较简单，Spring Security 上手复杂

2：Shiro 依赖性低，不需要任何框架和容器，可以独立运行，而 Spring Security 依赖于 Spring 容器

简介

Spring Security 是为基于 Spring 的应用程序提供**声明式安全保护**的安全性框架，它提供了完整的安全性解决方案，能够在 web 请求级别和方法调用级别处理身份证验证和授权。因为基于 Spring 框架，所以 Spring Security 充分利用了依赖注入和面向切面的技术。

Spring Security 主要是从两个方面解决安全性问题：

1. web 请求级别：使用 Servlet 规范中的过滤器（Filter）保护 Web 请求并限制 URL 级别的访问。
2. 方法调用级别：使用 Spring AOP 保护方法调用，确保具有适当权限的用户才能访问安全保护的方法。

DEMO

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

配置类:

```
package com.spring.security.springsecurity.config;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.Authenti
cationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecur
ity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCon
figurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
@EnableWebSecurity  //启用Web安全功能
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                //访问"/"和"/home"路径的请求都允许
                .antMatchers("/", "/home","/staff","/staff/*")
                .permitAll()
                //而其他的请求都需要认证
                .anyRequest()
                .authenticated()
                .and()
                //修改Spring Security默认的登陆界面
                .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
                .logout()
                .permitAll();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception{
        //基于内存来存储用户信息
        auth.inMemoryAuthentication().passwordEncoder(new
BCryptPasswordEncoder())
                .withUser("user").password(new
BCryptPasswordEncoder().encode("123")).roles("USER").and()
                .withUser("admin").password(new
BCryptPasswordEncoder().encode("456")).roles("USER","ADMIN");
    }
}
```

@EnableWebSecurity 注解: 启用Web安全功能

WebSecurityConfigurerAdapter

WebSecurityConfigurerAdapter 类: 可以通过重载该类的三个 configure()方法来制定 Web 安全的细节。

- configure(**WebSecurity**): 通过重载该方法, 可配置 Spring Security 的 **Filter** 链。
- configure(HttpSecurity): 通过重载该方法, 可配置如何通过**拦截器**保护请求。

保护路径的配置方法：

方法	能够做什么
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()	允许认证过的用户访问
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
hasAnyAuthority(String…)	如果用户具备给定权限中的某一个的话，就允许访问
hasAnyRole(String…)	如果用户具备给定角色中的某一个的话，就允许访问
hasAuthority(String)	如果用户具备给定权限的话，就允许访问
hasIpAddress(String)	如果请求来自给定IP地址的话，就允许访问
hasRole(String)	如果用户具备给定角色的话，就允许访问
not()	对其他访问方法的结果求反
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

Spring Security 支持的所有 SpEL 表达式如下::

安全表达式	计算结果
authentication	用户认证对象
denyAll	结果始终为false
hasAnyRole(list of roles)	如果用户被授权指定的任意权限，结果为true
hasRole(role)	如果用户被授予了指定的权限，结果 为true
hasIpAddress(IP Adress)	用户地址
isAnonymous()	是否为匿名用户
isAuthenticated()	不是匿名用户
isFullyAuthenticated	不是匿名也不是remember-me认证
isRemberMe()	remember-me认证
permitAll	始终true
principal	用户主要信息对象

- configure(AuthenticationManagerBuilder):通过重载该方法,可配置 user-detail(用户详细信息）服务。

方法	描述
accountExpired(boolean)	定义账号是否已经过期
accountLocked(boolean)	定义账号是否已经锁定
and()	用来连接配置
authorities(GrantedAuthority…)	授予某个用户一项或多项权限
authorities(List)	授予某个用户一项或多项权限
authorities(String…)	授予某个用户一项或多项权限
credentialsExpired(boolean)	定义凭证是否已经过期
disabled(boolean)	定义账号是否已被禁用
password(String)	定义用户的密码
roles(String…)	授予某个用户一项或多项角色

用户信息存储方式共有三种：

1. 使用**基于内存**的用户存储：通过 inMemoryAuthentication()方法，我们可以启用、配置并任意填充基于内存的用户存储。并且，我们可以调用 withUser()方法为内存用户存储添加新的用户，这个方法的参数是 username。withUser()方法返回的是 UserDetailsManagerConfigurer.UserDetailsBuilder，这个对象提供了多个进一步配置用户的方法，包括设置用户密码的 password()方法以及为给定用户授予一个或多个角色权限的 roles()方法。需要注意的是，roles()方法是 authorities()方法的简写形式。roles()方法所给定的值都会添加一个 ROLE_前缀，并将其作为权限授予给用户。因此上诉代码用户具有的权限为：ROLE_USER,ROLE_ADMIN。而借助 passwordEncoder()方法来指定一个密码转码器（encoder），我们可以对用户密码进行加密存储。

2. 基于**数据库表**进行认证：用户数据通常会存储在关系型数据库中，并通过 JDBC 进行访问。为了配置 Spring Security 使用以 JDBC 为支撑的用户存储，我们可以使用 jdbcAuthentication()方法，并配置他的 DataSource，这样的话，就能访问关系型数据库了。

3. 基于 LDAP 进行认证：为了让 Spring Security 使用基于 LDAP 的认证，我们可以使用 ldapAuthentication()方法。

方法调用级别的安全性

Spring Security 提供了三种不同的安全注解：

1. Spring Security 自带的@Secured 注解；
2. JSR-250 的@RolesAllowed 注解；
3. 表达式驱动的注解，包括@PreAuthorize、@PostAuthorize、@PreFilter 和@PostFilter。

注解	描述
@PreAuthorize	在方法调用之前，基于表达式的计算结果来限制对方法的访问
@PostAuthorize	允许方法调用，但是如果表达式计算结果为false，将抛出一个安全性异常
@PostFilter	允许方法调用，但必须按照表达式来过滤方法的结果
@PreFilter	允许方法调用，但必须在进入方法之前过滤输入值

启用基于注解的方法安全性

在 Spring 中，如果要启用基于注解的方法安全性，关键之处在于要在配置类上使用@EnableGlobalMethodSecurity

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true,jsr250Enabled =
true,prePostEnabled = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {

}
```

这里我们设置了 securedEnabled = true，此时 Spring 将会创建一个切点，并将带有@Secured 注解的方法防入切面中。同理，jsr250Enabled = true 与 prePostEnabled = true，分别表示启用@RolesAllowed 与表达式驱动的注解。

此时配置好 MethodSecurityConfig 类后，我们可以在上诉代码的基础上，在 SecurityController 中添加一个方法：

```
    @GetMapping(value = "/admin")
    @Secured("ROLE_ADMIN")
    public String admin(){
        return "admin";
    }
```

普通用户访问/admin 时将会出现 403 错误

前后端分离

在前后端分离的状态下，传统的 spring security 认证模式也需要做一点改造，以适应 ajax 的前端访问模式

传统的 spring security 安全机制是基于页面跳转的，使用 302 重定向（认证成功跳转至之前访问的页面，认证失败或未认证跳转至系统设置的默认登陆页面）。传统应用这么弄没问题，但现在 vue 一般都是基于 axios 进行 ajax 访问，ajax 请求是没法直接处理 302 跳转的（浏览器会直接处理跳转请求，ajax 的 callback 拿到的是跳转后的返回页面，在 spring security 中就是登陆首页，不符合需求）。幸好 spring security 所有的流程都是可以自定义的，我们可以扩展一下各个环节的流程。

原理

https://www.jianshu.com/p/a41a8f09b811