**Spring Security 调研**

# 简介

Spring Security本质上是一连串的Filter， 然后又以一个独立的Filter的形式插入到Filter Chain里，其名为FilterChainProxy。

实际上FilterChainProxy下面可以有多条Filter Chain，来针对不同的URL做验证，而Filter Chain中所拥有的Filter则会根据定义的服务自动增减。所以无需要显示再定义这些Filter，当想实现自己的逻辑可以自己实现这些Filter，注入到服务中。

如下图所示



## 核心组件

Spring Security主要由以下几部分组成的:

SecurityContextHolder, 提供几种访问 SecurityContext的方式。

SecurityContext, 保存Authentication信息和请求对应的安全信息。

Authentication, Spring Security安全信息的实体。

GrantedAuthority, 在应用程序范围，赋予主体的权限。

UserDetails,通过该bean，提供必要的信息，构建Authentication对象。

UserDetailsService, 通过数据库，获取用户真实数据。

## 目前支持这些技术集成的身份验证

HTTP BASIC 认证头 (基于 IETF RFC-based 标准)

HTTP Digest 认证头 ( IETF RFC-based 标准)

HTTP X.509 客户端证书交换 ( IETF RFC-based 标准)

LDAP (一个非常常见的方法来跨平台认证需要, 尤其是在大型环境)

Form-based authentication (用于简单的用户界面)

OpenID 认证

……

比较多不列了，详细可看官方文档https://springcloud.cc/spring-security-zhcn.html#jc-logout。

## 认证过程

1. 用户使用用户名和密码进行登录。

2.、Spring Security将获取到的用户名和密码封装成一个Authentication接口的实现类，比如常用的UsernamePasswordAuthenticationToken。

3、将上述产生的Authentication对象传递给AuthenticationManager的实现类ProviderManager进行authenticate函数认证。

4、ProviderManager依次轮询各个AuthenticationProvider进行认证，只要有一个认证成功则为认证成功，不在继续轮询。认证成功后返回一个封装了用户权限等信息的Authentication对象。

5、将AuthenticationManager返回的Authentication对象赋予给当前的SecurityContext。

## 验证过程

1、用户名和密码进行组合成一个实例UsernamePasswordAuthenticationToken (一个Authentication接口的实例, 我们之前看到的).

2、令牌传递到AuthenticationManager的实现类ProviderManager进行认证。

3、ProviderManager依次调用各个AuthenticationProvider进行认证，认证成功后返回一个封装了用户权限等信息的Authentication对象。

4、将AuthenticationManager返回的Authentication对象赋予给当前的SecurityContext，通过调用 SecurityContextHolder.getContext().setAuthentication(…​)实现。

## 授权相关

所有的Authentication实现类都保存了一个GrantedAuthority列表，其表示用户所具有的权限。GrantedAuthority是通过AuthenticationManager设置到Authentication对象中的，然后AccessDecisionManager将从Authentication中获取用户所具有的GrantedAuthority来鉴定用户是否具有访问对应资源的权限

# 实践

SpringSecurity有实现好的权限控制，根据自己的业务需求，还可以通过实现相关的接口或继承父类来定制自己的处理逻辑。

## 登录和登出

### 登录配置

可以使用以下配置，使用springsecurity 提供的登录功能，也可以不配置，自己定义登陆相关



### 默认登录认证

认证是由 AuthenticationManager 来管理的，但是真正进行认证的是 AuthenticationManager 中定义的 AuthenticationProvider。AuthenticationManager 中可以定义有多个 AuthenticationProvider。

Spring Security 默认会使用 DaoAuthenticationProvider。DaoAuthenticationProvider 在进行认证的时候需要一个 UserDetailsService 来获取用户的信息 UserDetails，其中包括用户名、密码和所拥有的权限等。所以如果我们需要改变认证的方式，我们可以实现自己的 AuthenticationProvider；如果需要改变认证的用户信息来源，我们可以实现 UserDetailsService

AuthenticationProvider的实现类

### 自定义登录认证

SpringSecurity 有自己的登录验证，你也可以实现自定义验证满足业务。

可以通过实现AuthenticationProvider接口，自定义登录方式和验证逻辑，也可以继承已有的实现类，修改已有登录认证方式的验证逻辑，将其添加进ProviderManager里，并将实现配置到WebSecurityConfigurerAdapter中，从而实现自己的业务。。

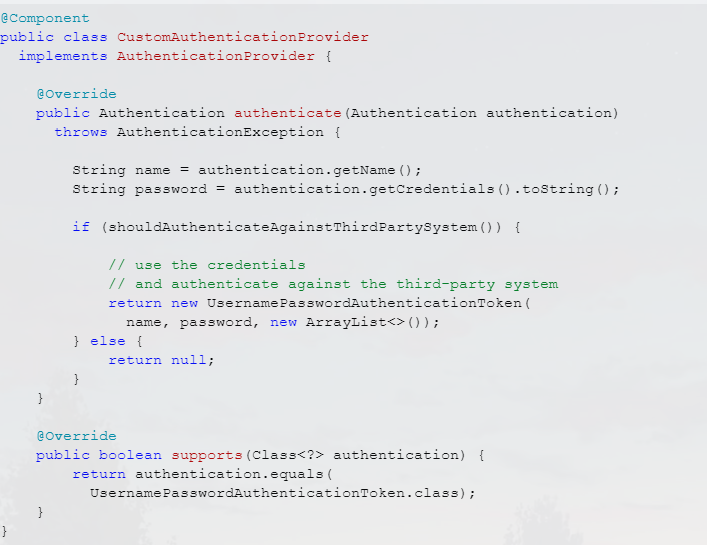
Authenticate 方法的返回结果：

1、验证成功，返回一个带有用户信息的Authentication。

2、验证失败，抛出一个AuthenticationException异常。

3、无法判断，返回null，会执行下一个AuthenticationProvider。

其中的supports()方法接受一个authentication参数，用来判断传进来的authentication是不是该AuthenticationProvider能够处理的类型。



根据springSecurity的认证规则，可以实现多个AuthenticationProvider，定义多种不同方式认证的认证处理逻辑，根据返回的结果来判断

### 登出

可以自定义登出，也可通过配置对登出管理。



### 密码加密

**1、BCryptPasswordEncoder相关知识：**

用户表的密码通常使用MD5等不可逆算法加密后存储，为防止彩虹表破解更会先使用一个特定的字符串（如域名）加密，然后再使用一个随机的salt（盐值）加密。

特定字符串是程序代码中固定的，salt是每个密码单独随机，一般给用户表加一个字段单独存储，比较麻烦。

BCrypt算法将salt随机并混入最终加密后的密码，验证时也无需单独提供之前的salt，从而无需单独处理salt问题。

**2、WebSecurityConfig类配置**

## 

1. **注册时密码加密后存入数据库**

以下类似代码，对用户密码进行加密保存如数据库

private void encryptPassword(UserEntity userEntity){

String password = userEntity.getPassword();

password = new BCryptPasswordEncoder().encode(password);

userEntity.setPassword(password);

}

补充说明：即使不同的用户注册时输入相同的密码，存入数据库的密文密码也会不同。

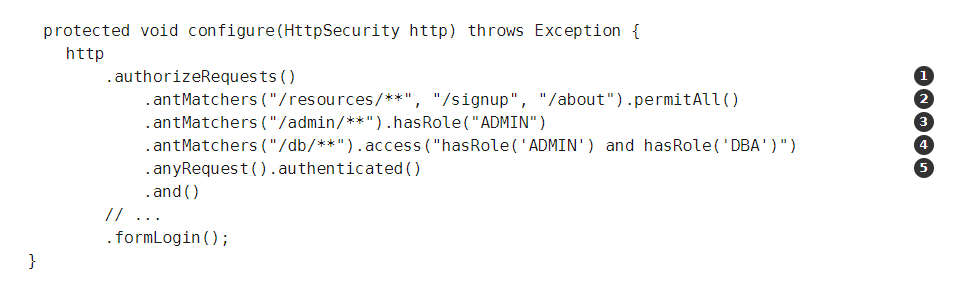
1. **登录时密码加密校验**

通过.passwordEncoder(passwordEncoder()) 配置后，登录时用到了 DaoAuthenticationProvider ，它有一个方法 #additionalAuthenticationChecks(UserDetails userDetails,UsernamePasswordAuthenticationTokenauthentication)，此方法用来校验从数据库取得的用户信息和用户输入的信息是否匹配，类似passwordEncoder.matches(rawPassword, encodedPassword) 的方式。

## 权限验证

### 使用HttpSecurity配置URL权限验证

我们的例子中要求用户进行身份验证并且在我们应用程序的每个URL这样做。我么你可以通过给http.authorizeRequests()添加多个子节点来指定多个定制需求到我们的URL。例如：



1、http.authorizeRequests()方法有多个子节点，每个macher按照他们的声明顺序执行。

2、我们指定任何用户都可以通过访问的多个URL模式。任何用户都可以访问URL以"/resources/", equals "/signup", 或者 "/about"开头的URL。

3、以 "/admin/" 开头的URL只能由拥有 "ROLE\_ADMIN"角色的用户访问。请注意我们使用 hasRole 方法，没有使用 "ROLE\_" 前缀.

4、任何以"/db/" 开头的URL需要用户同时具有 "ROLE\_ADMIN" 和 "ROLE\_DBA"。和上面一样我们的 hasRole 方法也没有使用 "ROLE\_" 前缀.

5、尚未匹配的任何URL要求用户进行身份验证

### 自定义权限鉴定规则

通过httpSecurity 配置.anyRequest().access("@mySecurity.check(authentication,request)") 定义自己的过滤鉴定规则。

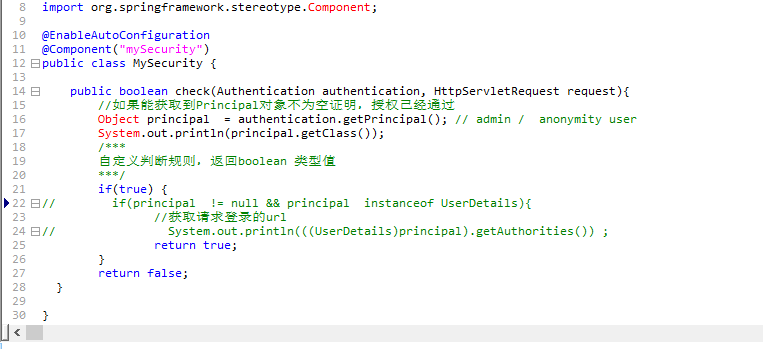
httpSecurity.

authorizeRequests()

.antMatchers("/XXX").permitAll()

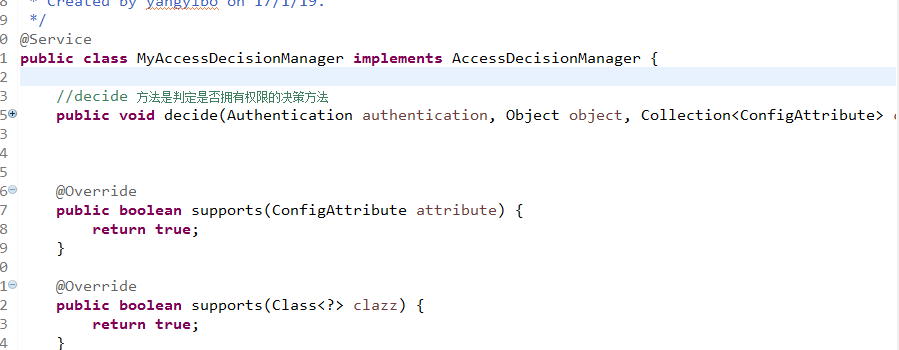
.anyRequest().access("@mySecurity.check(authentication,request)")

通过mySecurity bean的check 方法，来鉴定权限，如下



### 自定义决策管理器AccessDecisionManager

AccessDecisionManager 最终决策Authentication对资源的权限判断，会注册一个默认的，你可以实现该接口重写方法，自定义决策，



通过AbstractSecurityIntercepto继承子类setMyAccessDecisionManager方法注入自定义的决策管理，并在config 中添加该实例：httpSecurity.addFilterBefore(myFilterSecurityInterceptor, FilterSecurityInterceptor.class),即可使该权限决策管理生效。

## 方法级别注解

在集成WebSecurityConfigurerAdapter的配置类上添加@EnableGlobalMethodSecurity(

securedEnabled = true)来控制方法注解的开启

在api 方法上添加注解，如下所示：

@Secured("IS\_AUTHENTICATED\_ANONYMOUSLY")

public Account readAccount(Long id);

## 支持的标签库

官方文档支持jsp标签

支持Thymeleaf 标签

<html xmlns:th="http://www.thymeleaf.org"

xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4">

<div sec:authorize="isAuthenticated()">

This content is only shown to authenticated users.

</div>

<div sec:authorize="hasRole('ROLE\_ADMIN')">

This content is only shown to administrators.

</div>

<div sec:authorize="hasRole('ROLE\_USER')">

This content is only shown to users.

</div>



