Spring Security是一个提供身份验证，授权和保护以防止常见攻击的框架
本质上，Spring Security是一个过滤链

Spring Security过滤器加载的的完整过程

ChannelProcessingFilter
WebAsyncManagerIntegrationFilter
SecurityContextPersistenceFilter
HeaderWriterFilter
CorsFilter
CsrfFilter
LogoutFilter
OAuth2AuthorizationRequestRedirectFilter
Saml2WebSsoAuthenticationRequestFilter
X509AuthenticationFilter
AbstractPreAuthenticatedProcessingFilter
CasAuthenticationFilter
OAuth2LoginAuthenticationFilter
Saml2WebSsoAuthenticationFilter
UsernamePasswordAuthenticationFilter
OpenIDAuthenticationFilter
DefaultLoginPageGeneratingFilter
DefaultLogoutPageGeneratingFilter
ConcurrentSessionFilter
DigestAuthenticationFilter
BearerTokenAuthenticationFilter
BasicAuthenticationFilter
RequestCacheAwareFilter
SecurityContextHolderAwareRequestFilter
JaasApiIntegrationFilter
RememberMeAuthenticationFilter
AnonymousAuthenticationFilter
OAuth2AuthorizationCodeGrantFilter
SessionManagementFilter
ExceptionTranslationFilter
FilterSecurityInterceptor
SwitchUserFilter

SecurityFilterChain

Client

Filter Chain

Filter

DelegatingFilterProxy

FilterChainProxy

Filter

Servlet

Security Filter
Security Filter
Security Filter

/api/**

Filter
Filter
Filter

SecurityFilterChain

Security Filter
Security Filter
Security Filter

/hello/**

在spring-boot-autoconfigure的依赖包下，
找到META-INF/spring.factories文件

找到org.springframework.boot.autoconfigure.EnableAutoConfiguration

发现SpringBoot自动装配
org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration

@Import   @Import   @Import

SpringBootWebSecurityConfiguration

WebSecurityEnablerConfiguration

整合SpringData时配置
SecurityDataConfiguration

自动配置条件

@ConditionalOnDefaultWebSecurity

自动配置一个SecurityFilterChain

defaultSecurityFilterChain(HttpSecurity http)

有EnableWebSecurity类就进行下面配置
@ConditionalOnClass({EnableWebSecurity.class})

@EnableWebSecurity

@Conditional

DefaultWebSecurityCondition

没有配置的话，会自动配置一个
WebSecurityConfigurerAdapter
SecurityFilterChain

@ConditionalOnMissingBean({WebSecurityConfigu
rerAdapter.class, SecurityFilterChain.class})

@EnableGlobalAuthentication

WebSecurityConfiguration

SpringWebMvcImportSelector   OAuth2ImportSelector   HttpSecurityConfiguration

@Import

@Bean(name = AbstractSecurityWebApplicationInitializer.DEFAULT_FILTER_NAME)

AuthenticationConfiguration

springSecurityFilterChain()

setFilterChainProxySecurityConfigurer(objectPostProcessor,
List <SecurityConfigurer<Filter, WebSecurity>>
webSecurityConfigurers)

创建一个WebSecurity

@Import

ObjectPostProcessorConfiguration

建造者模式

this.webSecurity.build();

objectPostProcessor.postProcess(new
WebSecurity(objectPostProcessor));

spring创建一个bean

autowireBeanFactory.initializeBean(obje
ct, object.toString())

创建实例ObjectPostProcessor

@Bean

new AutowireBeanFactoryObjectPostProcessor(beanFactory)

空方法
可扩展

将webSecurityConfigurer添加到webSecurity中

this.webSecurity.apply(webSecurityConfigurer)

autowireBeanFactory.autowireBean(object);

populateBean(bd.getBeanClass().getName(), bd, bw);

@Bean

//GlobalAuthenticationConfigurerAdapter EnableGloba
lAuthenticationAutowiredConfigurer(context)

beforeInit();

判断   判断

@Bean

//InitializeUserDetailsBeanManagerConfigurerInitialize
UserDetailsBeanManagerConfigurer(context)

调用所有SecurityConfigurer的
init()方法

init();

configurer.init((B) this);

result instanceof
DisposableBean

result instanceof
SmartInitializingSinglet
on

@Bean

//InitializeAuthenticationProviderBeanManagerConfigurerInit
ializeAuthenticationProviderBeanManagerConfigurer(context
)

空方法
可扩展

disposableBeans.add((DisposableBean) result)

smartSingletons.add((SmartInitializingSingleton) result)

// AuthenticationManagerBuilderDefaultPasswordEnco
derAuthenticationManagerBuilder

beforeConfigure();

调用所有SecurityConfigurer的
configure()方法

configure();

configurer.configure((B) this)

开始创建

performBuild()

securityFilterChains.add(new
DefaultSecurityFilterChain(ignoredRequest));

httpSecurity在这里build

securityFilterChains.add(securityFilterChainBuilder.build())

FilterChainProxy委托securityFilterChains执行doFilter()

new FilterChainProxy(securityFilterChains)

filterChainProxy.afterPropertiesSet();

运行一个线程

postBuildAction.run();

返回FilterChainProxy

return filterChainProxy