

Spring Security

Java 安全 (Security) 技术选型

Java Security 技术

Java SE - Java Security

- 加密/解密
 - Oracle Java 算法禁运 - Java Security 扩展
 - 古巴
 - 朝鲜
 - 伊朗
 - 叙利亚
- JVM 安全
 - Java 安全沙箱
 - 权限 API - Permission
 - 属性权限 - PropertyPermission
 - 运行时权限 - RuntimePermission
 - ClassLoader 使用 - getClassLoader
 - 反射权限 - ReflectPermission
 -
 - 权限配置
 - java.policy - Java Permission 授权文件
 - java.security - Java 安全配置扩展文件
 - Java 身份表示 - java.security.Principal
 - Java 安全异常 - java.lang.SecurityException (运行时异常)

- Java 安全管理器 - java.lang.SecurityManager
 - Java 安全校验方法 -
checkPermission(java.security.Permission)
- Java 安全入口控制器 - java.security.AccessController
 - Java 安全校验方法 -
checkPermission(java.security.Permission)
 - Java 鉴权方法 -
doPrivileged(java.security.PrivilegedAction) 以及重载

ClassLoader 安全

- Bootstrap ClassLoader (用户 Java 代码无法获取, null) - 无法 re-define Class 结构
 - 每个 ClassLoader 会有自己加载的 Classes -
findLoadedClass
 - java.lang.String 无法被修改
- System ClassLoader (用户 Java 代码可通过
ClassLoader#getSystemClassLoader) - 允许 re-define
Class 结构 (需要 Java Security 授权)
 - App ClassLoader (用户 Java 代码可以自定义
ClassLoader)

Web Security 技术选型

Servlet Security

参考 Servlet 规范 3.1

- Servlet 程序是没有权限去写入（读权限）
- 外部的程序是没有任何权限

HttpServletRequest 安全相关

安全方法

- 认证方法 - authenticate
- 登录/登出 - login 和 logout
- 获取远程用户（名） - getRemoteUser
- 用户角色判断 - isUserInRole
- 用户身份 - Principal

@ServletSecurity 安全相关

Tomcat 对 Servlet Security 实现

认证方式

- BASIC
- FORM
- Client Certificate
- Digest

Apache Shiro

Spring Security

Spring Security 模块

- Remoting - spring-security-remoting.jar
- Web - spring-security-web.jar
- Config - spring-security-config.jar
- OAuth 2.0 Core - spring-security-oauth2-core.jar
- OAuth 2.0 Client - spring-security-oauth2-client.jar
- OAuth 2.0 JOSE - spring-security-oauth2-jose.jar

Spring Security 核心原理

Spring Security 对 Java Web 实现

Spring Security 对 Servlet 实现

传统 Servlet 3.0+ 容器实现

Web 自动装配实现 -

`AbstractSecurityWebApplicationInitializer`

Spring Web `webApplicationInitializer`

- `AbstractSecurityWebApplicationInitializer`

小知识: Servlet 3.0+ `javax.servlet.ServletContainerInitializer` 所有它的实现配置在 `META-INF/services/javax.servlet.ServletContainerInitializer` 后, 均会被 Servlet 3.0+ 容器回调方法 -

`javax.servlet.ServletContainerInitializer#onStartup`

- 代表实现类 - `SpringServletContainerInitializer`
- 当实现类标注 `@javax.servlet.annotation.HandlesTypes` 时, 并且制定类的范围, 被指定类的所有派生类会出现在 `onStartup` 方法的首参 - `Set<Class<?> classes`

"springSecurityFilterChain" `Filter` 执行流程:

`AbstractSecurityWebApplicationInitializer` ->

`DelegatingFilterProxy` -> "springSecurityFilterChain" `Filter`

Spring Security 利用 Servlet 3.0+ Web 自动装配能力来引导 Root `ApplicationContext` 装配, 并且将一个名为 "springSecurityFilterChain" 动态地 (编码) 加入 (注册) 到 `ServletContext` 中。

小知识: "springSecurityFilterChain" 使通过 `org.springframework.web.filter.DelegatingFilterProxy` 动态读取, 并且 `DelegatingFilterProxy` 是一个 `Filter`, 它通过配置形成获取 Spring 应用上下文中的 `Filter Bean`

springSecurityFilterChain 实现 (简版)

1. 注入 `List<SecurityConfigurer<Filter, WebSecurity>>` Bean
- 2.

Spring Security 对 WebFlux 实现

Spring Security Servlet 核心组件

Servlet 核心扩展适配器 - `WebSecurityConfigurerAdapter`

- `configure` 重载方法
 - `configure(WebSecurity)`
 - `configure(HttpSecurity)`

核心配置接口 - `SecurityConfigurer`

Web 安全接口 - `WebSecurity`

与 `FilterChainProxy` 密切相关

设计模式: Builder

HTTP 安全接口 - `HttpSecurity`

与 HTTP 安全特性密切相关

- CSRF
- XSS
- Content Security Policy (CSP)

小结论: `HttpSecurity` 每种功能特性方法被调用时, 它所对应的 `SecurityConfigurer` 实现类被激活

设计模式: Builder

小技巧：如果需要自定义 Filter 添加到 Spring Security Filter Chain 时，需要记住框架内部 Filter 实现类以及它对应顺序，即通过 `HttpSecurity#addFilterBefore`

HTTP 请求匹配器 - `RequestMatcher`

功能特性

通过 `HttpSecurity` 接口的特性方法，如：`csrf()` 构建 `SecurityConfigurer` 对象，从而激活对应 `Filter` 实现

CSRF

核心组件 - `CsrfFilter`

安全 HTTP 响应头

通过 `HttpSecurity#headers()` 方法来激活安全 HTTP 响应头，该方法返回响应的对象 `HeadersConfigurer`，该类提供子特性：

- Cache Control - `cacheControl()`
- HSTS

小总结：`HeadersConfigurer` 子特性方法激活特性配置对象，当调用 `enable()` 方法时，会初始化 `HeaderWriter` 实现类，相反 `disable()` 方法调用时，将 writer 置为 `null`

