Spring Security

# 概述

Spring Security为J2EE应用提供了一个全面的**安全解决方案**。当你在阅读这个教程的时，你会发现，我们尝试向你展示的是一个有用的和**高度可配置**的安全系统。

安全是一个不断演进的目标，追求一个全面的、系统级别的安全方案是非常重要的。站在安全领域的角度，我们鼓励分层的概念，每一层都只管理自己职责范围内的安全问题，每一层的安全机制越严格，我们的应用就越健壮、越安全。

1、在最底层，我们需要处理传输层安全和系统识别，从而避免中间人(man-in-the-middle )攻击。

2、接着我们会使用防火墙，可能会联合VPN或者IP安全机制来保证只有被授权的系统才能进行连接。

3、在企业环境中，我们需要部署一个DMZ( demilitarized zone )服务来隔离对外提供访问的接口的服务器与内部数据库和应用服务器。

4、我们的操作系统也扮演了安全中的一环，例如使用不具有特定权限的用户运行进程，限制用户最大可以操作的文件数量等。操作系统通常也会配置自己的防火墙。

5、我们可能还会尝试使阻止DDOS( Distributed DenialofService)分布式拒绝服务和暴力破解攻击（brute force attacks ）。一个入侵检测系统对于攻击的监控和响应是非常有用的，可以帮助我们实时的拒绝某些TPC/IP地址的访问。

6、从更高的层面即JVM的层面来说，我们可以通过配置最小化一个Java类可以具有的权限(译者注：通过JAVA\_HOME/jre/lib/security/java.policy文件进行配置)

7、最后我们在应用层面添加一些领域特定的安全配置。

**Spring Security可以让最后一点，即应用相关的安全( application security )设置变得更加容易**。

**你可能已经知道，应用级别的安全主要分为“验证( authentication) ”和“(授权) authorization ”两个部分**。这也是Spring Security主要需要处理的两个部分。“ Authentication ”指的是建立规则( principal )的过程。规则可以是一个用户、设备、或者其他可以在我们的应用中执行某种操作的其他系统。" Authorization "指的是判断某个 principal 在我们的应用是否允许执行某个操作。在 进行授权判断之前，要求其所要使用到的规则必须在验证过程中已经建立好了。这些概念是通用的，并不是只针对"Spring Security"。

以前一直分不清 authentication 和 authorization，其实很简单，举个例子来说：你要登机，你需要出示你的身份证和机票，身份证是为了证明你张三确实是你张三，这就是 authentication；而机票是为了证明你张三确实买了票可以上飞机，这就是 authorization。在 computer science 领域再举个例子：你要登陆论坛，输入用户名张三，密码1234，密码正确，证明你张三确实是张三，这就是 authentication；再一check用户张三是个版主，所以有权限加精删别人帖，这就是 authorization。

在验证(Authentication )层面， Spring Security 提供了不同的验证模型。大部分的authentication模型来自于第三方或者权威机构或者由一些相关的标准制定组织(如IETF)开发。此外，Spring Security也提供了一些验证特性。特别的，Spring Security目前支持对以下所有验证方式的整合：

* HTTP BASIC authentication headers (一个基于 IEFT RFC 的标准)
* HTTP Digest authentication headers (一个基于 IEFT RFC 的标准)
* HTTP X.509 client certificate exchange (一个基于 IEFT RFC 的标准)
* LDAP (一个非常常见的跨平台认证需要做法，特别是在大环境)
* Form-based authentication (提供简单用户接口的需求)
* OpenID authentication
* 基于预先建立的请求头进行认证 （比如 Computer Associates Siteminder）
* JA-SIG Central Authentication Service (也被称为 CAS，这是一个流行的开源单点登录系统)
* Transparent authentication context propagation for Remote Method Invocation(RMI) and HttpInvoker (一个 Spring 远程调用协议)
* Automatic "remember-me" authentication (这样你可以设置一段时间，避免在一段时间内还需要重新验证)
* Anonymous authentication (允许任何调用，自动假设一个特定的安全主体)
* Run-as authentication (这在一个会话内使用不同安全身份的时候是非常有用的)
* Java Authentication and Authorization Service (JAAS)
* JEE Container autentication (这样，你可以继续使用容器管理认证，如果想的话)
* Kerberos
* Java Open Source Single Sign On (JOSSO) \*
* OpenNMS Network Management Platform \*
* AppFuse \*
* AndroMDA \*
* Mule ESB \*
* Direct Web Request (DWR) \*
* Grails \*
* Tapestry \*
* JTrac \*
* Jasypt \*
* Roller \*
* Elastic Plath \*
* Atlassian Crowd \*
* 你自己的认证系统

注：\*号标记的部分由第三方提供。

除了验证机制， Spring Security 也提供了一系列的授权能力。主要感兴趣的是以下三个方面：

1、对web请求进行授权

2、授权某个方法是否可以被调用

3、授权访问单个领域对象实例

# Spring Security 模块

在Spring3.0中，Spring Security将代码划分到不同的jar中，这使得不同的功能模块和第三方依赖显得更加清晰。如果你使用maven来构建项目，下面可能是你将会加入到pom.xml中的模块。

* Core - - - - spring-security-core.jar

包含了核心认证和权限控制类和接口， 运程支持和基本供应 API。使用 Spring Security

所必须的。支持单独运行的应用， 远程客户端，方法（服务层）安全和 JDBC 用户供应。

* Web - - - - spring-security-web.jar

包含过滤器和对应的 web 安全架构代码。任何需要依赖 servlet API 的。 你将需要它，如

果 你 需 要 Spring Security Web 认 证 服 务 和 基 于 URL 的 权 限 控 制 。

* Config - - - - spring-security-config.jar

包含安全命名控制解析代码（因此我们不能直接把它用在你的应用中）。你需要它， 如果使

用 了 Spring Security XML 命 名 控 制 来 进 行 配 置 。

* LDAP - - - - spring-security-ldap.jar

LDAP 认证和实现代码，如果你需要使用 LDAP 认证或管理 LDAP 用户实体就是必须的。

* LACL - - - - spring-security-acl.jar

处理领域对象 ACL 实现。用来提供安全给特定的领域对象实例，在你的应用中。

* CAS - - - - spring-security-cas-client.jar

Spring Security 的 CAS 客户端集成。如果你希望使用 Spring Security web 认证 整合

一个 CAS 单点登录服务器。

* OpenID - - - - spring-security-openid.jar

OpenID web 认 证 支 持 。 用 来 认 证 用 户 ， 通 过 一 个 外 部 的 OpenID 服 务 。

# Spring Security使用

## Maven中引入必须依赖

<dependencies>

<!-- ... other dependency elements ... -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

</dependencies>

## 命名空间配置

<beans:beans xmlns="

xmlns:beans="

xmlns:xsi="

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security.xsd">

//配置spring\_security

</beans:beans>

### **web.xml配置**

你第一件要做的事是在你的web.xml文件中声明过滤器的配置。

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

这为Spring Security框架提供了一个钩子函数。 DelegatingFilterProxy 是Spring框架的一个类，其会将过滤操作委派给一个在application context中以spring bean方式定义的Filter。在这个案例中，bean的名字是" springSecurityFilterChain"，这是由命名空间创建的一个用来处理web安全的内部基础bean 。注意，你自己不应该使用这个bean的名称。一旦你将这个添加到你的web.xml，你就可以开始编辑你的application context文件。Web安全服务通过<http>元素进行配置。

### Web应用安全 <http>元素

<http>元素为你的应用程序的 web 层封装了安全性配置。 它创建了一个名为

"springSecurityFilterChain"的 FilterChainProxy bean，这个 bean 维护了一系列的建

立了 web 安全配置的安全过滤器。 一些核心的过滤器总是要被创建的，其他的将根据子元素的配置添加到过滤器队列中。 标准过滤器的位置都是固定的，这避免了之前版本中的一个常见问题，那时候用户必须自己在FilterChainProxy bean 中配置过滤器链。

<http> 命 名 空 间 块 会 创 建 一 个 HttpSessionContextIntegrationFilter ， 一 个ExceptionTranslationFilter 和一个 FilterSecurityInterceptor。 它们是固定的，不能使用其他可选方式替换。

#### 属性

* **access-decision-manager-ref** 指定AccessDecisionManager的实现 用于授权HTTP请求.
* **authentication-manager-ref** 指定当前http元素的FilterChain的AuthenticationManager.
* **auto-config** 自动注册登录表单，基本认证，注销服务。如果设置为“真”，所有这些功能都被添加。如果未指定，默认为“假”。不建议使用此属性。使用明确的配置元素，避免混乱。.
* **create-session**: 控制spring security创建HTPP Session,包括如下选项:

**always** : 如果不存在Spring Security就创建.

**ifRequired** : 如果需要Spring Security就创建(默认值).

**never** : Spring Security从不创建，但会使用已经存在的HTTP Session；

**stateless** : Spring Security不创建session且会忽略( will not create a session and ignore the session for obtaining a Spring Authentication).

* **disable-url-rewriting:** 阻止URL重写，默认是true。
* **entry-point-ref:** Normally the AuthenticationEntryPoint used will be set depending on which authentication mechanisms have been configured. This attribute allows this behaviour to be overridden by defining a customized AuthenticationEntryPoint bean which will start the authentication process.
* **name:** A bean identifier, used for referring to the bean elsewhere in the context.
* **once-per-request**: Corresponds to the observeOncePerRequest property of FilterSecurityInterceptor. Defaults to true.
* **pattern**: 拦截请求的模式配置，如果不配置，默认拦截所有请求
* **realm**: Sets the realm name used for basic authentication (if enabled). Corresponds to the realmName property on BasicAuthenticationEntryPoint.
* **security** 设置为none则匹配的请求不被spring security管理
* **security-context-repository-ref**: 给SecurityContextPersistenceFilter 指定自定义的SecurityContextRepository。
* **use-expressions**: Enables EL-expressions in the access attribute.The default value is true.
* request-matcher：请求匹配规则，默认是[ant](https://blog.csdn.net/wangshfa/article/details/26471641)，也可以是正则表达式

#### <http>子元素

##### <access-denied-handler>

设置默认的AccessDeniedHandler的errorPage属性，以及自定义AccessDeniedHandler实现(AccessDeniedHandler被ExceptionTranslationFilter使用)。

属性：

* error-page：指定用户访问一个未被授权的页面时，跳转的页面！
* ref ：指定自定义的AccessDeniedHandler实现。

##### <cors>

跨域请求配置

属性：

* ref ：指定CorsFilter
* cors-configuration-source-ref 指定CorsConfigurationSource

##### <headers>

设置response头信息

* Cache-Control, Pragma, and Expires - Can be set using the [cache-control](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-cache-control) element. This ensures that the browser does not cache your secured pages.
* Strict-Transport-Security - Can be set using the [hsts](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/" \l "nsa-hsts" \o "41.1.7 <hsts>) element. This ensures that the browser automatically requests HTTPS for future requests.
* X-Frame-Options - Can be set using the [frame-options](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-frame-options) element. The [X-Frame-Options](https://en.wikipedia.org/wiki/Clickjacking#X-Frame-Options) header can be used to prevent [clickjacking attacks.](https://blog.csdn.net/tanzhen1991910/article/details/53007791?locationNum=14&fps=1)
* X-XSS-Protection - Can be set using the [xss-protection](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/" \l "nsa-xss-protection" \o "41.1.14 <xss-protection>) element. The [X-XSS-Protection](https://en.wikipedia.org/wiki/Cross-site_scripting)header can be used by browser to do basic control.
* X-Content-Type-Options - Can be set using the [content-type-options](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-content-type-options) element. The [X-Content-Type-Options](https://blogs.msdn.com/b/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx) header prevents Internet Explorer from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome, when downloading extensions.
* Public-Key-Pinning or Public-Key-Pinning-Report-Only - Can be set using the [hpkp](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/" \l "nsa-hpkp" \o "41.1.8 <hpkp>) element. This allows HTTPS websites to resist impersonation by attackers using mis-issued or otherwise fraudulent certificates.
* Content-Security-Policy or Content-Security-Policy-Report-Only - Can be set using the [content-security-policy](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-content-security-policy) element. [Content Security Policy (CSP)](https://www.w3.org/TR/CSP2/)is a mechanism that web applications can leverage to mitigate content injection vulnerabilities, such as cross-site scripting (XSS).
* Referrer-Policy - Can be set using the [referrer-policy](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-referrer-policy) element, [Referrer-Policy](https://www.w3.org/TR/referrer-policy/) is a mechanism that web applications can leverage to manage the referrer field, which contains the last page the user was on.

属性：

* **defaults-disabled** Optional attribute that specifies to disable the default Spring Security’s HTTP response headers. The default is false (the default headers are included).
* **disabled** Optional attribute that specifies to disable Spring Security’s HTTP response headers. The default is false (the headers are enabled).

##### <anonymous>

添加一个AnonymousAuthenticationFilter 和AnonymousAuthenticationProvider。但我们需要使用匿名认证(IS\_AUTHENTICATED\_ANONYMOUSLY)时。它是必须的。

属性：

* **enabled** 是否允许匿名认证
* **granted-authority**  给匿名请求授予怎样的权限，默认是ROLE\_ANONYMOUS
* **key** The key shared between the provider and filter.
* **username** 设置匿名请求的用户名称，默认是anonymousUser

##### <[csrf](https://blog.csdn.net/stpeace/article/details/53512283)>

This element will add [Cross Site Request Forger (CSRF)](https://en.wikipedia.org/wiki/Cross-site_request_forgery) protection to the application. It also updates the default RequestCache to only replay "GET" requests upon successful authentication.

属性：

disabled 是否禁止Spring Security的CSRF保护。默认false。

token-repository-ref 使用 CsrfTokenRepository. 默认HttpSessionCsrfTokenRepository

##### <custom-filter>

添加过滤器到spring security的过滤器链中

属性：

* after 将过滤器放置在指定过滤器的后面
* before 讲过滤器放置在指定过滤器的前面
* position 将过滤器放置在指定位置
* ref 指定过滤器

##### <expression-handler>

指定一个SecurityExpressionHandler 对象用于启用基于表达式的访问控制。如果不指定使用默认值。

##### <form-login>

Used to add an UsernamePasswordAuthenticationFilter to the filter stack and an LoginUrlAuthenticationEntryPoint to the application context to provide authentication on demand. This will always take precedence over other namespace-created entry points. If no attributes are supplied, a login page will be generated automatically at the URL "/login" ,The behaviour can be customized using the [<form-login> Attributes](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#nsa-form-login-attributes).

属性：

* default-target-url UsernamePasswordAuthenticationFilter的defaultTargetUrl 属性值，默认是”/”,登录成功后会跳转到该页面。
* always-use-default-target 是否总是使用默认的登录成功后跳转url ，UsernamePasswordAuthenticationFilter的alwaysUseDefaultTargetUrl属性值
* login-page 自定义登录页url,默认为/login，LoginUrlAuthenticationEntryPoint的loginFormUrl 属性值
* login-processing-url 登录请求拦截的url,也就是form表单提交时指定的action ，UsernamePasswordAuthenticationFilter的filterProcessesUrl属性值，默认是”/login”;
* password-parameter 密码的请求字段 默认为password
* username-parameter用户名的请求字段 默认为username
* authentication-failure-url 登录失败后跳转的url，UsernamePasswordAuthenticationFilter的authenticationFailureUrl属性值。
* authentication-success-handler-ref 指向一个AuthenticationSuccessHandler用于处理认证成功的请求,不能和default-target-url还有always-use-default-target同时使用
* authentication-success-forward-url 用于authentication-failure-handler-ref
* authentication-failure-handler-ref 指向一个AuthenticationFailureHandler用于处理失败的认证请求，不能和authentication-failure-url同时使用
* authentication-failure-forward-url 用于authentication-failure-handler-ref
* authentication-details-source-ref 指向一个AuthenticationDetailsSource,在认证过滤器中使用

##### <http-basic>

使用HTTP Basic认证，默认添加BasicAuthenticationFilter和BasicAuthenticationEntryPoint 到配置中。

属性：

* authentication-details-source-ref 指定AuthenticationDetailsSource
* entry-point-ref 指定BasicAuthenticationFilter的AuthenticationEntryPoint

##### <intercept-url>

配置一组URL Pattern的访问规则。如果指定的URL需要通过HTTPS访问的话，那么还需要配置ChannelProcessingFilter 。

属性：

* access 设置给定URL Pattern的可访问规则，多个规则(角色)间用逗号隔开。
* pattern 匹配url，默认使用ant表达式，具体取决于http元素的request-matcher属性值。
* requires-channel 指定访问协议，requires-channel支持三个值：http、https和any。any表示http和https都可以访问。
* method 通常我们都会要求某些URL只能通过POST请求，某些URL只能通过GET请求。这些限制Spring Security也已经为我们实现了，通过指定intercept-url的method属性可以限制当前intercept-url适用的请求方式，默认为所有的方式都可以。  method的可选值有GET、POST、DELETE、PUT、HEAD、OPTIONS和TRACE。

##### <logout>

添加LogoutFilter过滤器

属性：

* **delete-cookies 退出登录后需要被删除的cookie名称，多个用逗号分隔**
* **invalidate-session 退出登录后是否session失效，默认是true**
* **logout-success-url 退出登录成功后，跳转的页面**
* **logout-url 退出登录的URL，默认是/logout**
* **success-handler-ref 指定**LogoutSuccessHandler

##### <[openid-login](https://blog.csdn.net/hereweare2009/article/details/4014045)>

配置openid方式登录

属性：

* default-target-url UsernamePasswordAuthenticationFilter的defaultTargetUrl 属性值，默认是”/”,登录成功后会跳转到该页面。
* always-use-default-target 是否总是使用默认的登录成功后跳转url ，UsernamePasswordAuthenticationFilter的alwaysUseDefaultTargetUrl属性值
* login-page 自定义登录页url,默认为/login，LoginUrlAuthenticationEntryPoint的loginFormUrl 属性值
* login-processing-url 登录请求拦截的url,也就是form表单提交时指定的action ，UsernamePasswordAuthenticationFilter的filterProcessesUrl属性值，默认是”/login”;
* password-parameter 密码的请求字段 默认为password
* username-parameter用户名的请求字段 默认为userName
* authentication-failure-url 登录失败后跳转的url，UsernamePasswordAuthenticationFilter的authenticationFailureUrl属性值。
* authentication-success-handler-ref 指向一个AuthenticationSuccessHandler用于处理认证成功的请求,不能和default-target-url还有always-use-default-target同时使用
* authentication-success-forward-url 用于authentication-failure-handler-ref
* authentication-failure-handler-ref 指向一个AuthenticationFailureHandler用于处理失败的认证请求，不能和authentication-failure-url同时使用
* authentication-failure-forward-url 用于authentication-failure-handler-ref
* authentication-details-source-ref 指向一个AuthenticationDetailsSource,在认证过滤器中使用
* user-service-ref A reference to a user-service (or UserDetailsService bean) Id

子元素有：<attribute-exchange>、<openid-attribute> 详见[官方手册](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/)

##### <[remember-me](http://wiki.jikexueyuan.com/project/spring-security/remember-me.html)>

实现remember-me自动登录功能。

属性：

* remember-me-parameter 传递给后台的请求参数名，用于触发remember-me自动登录功能。默认值是"remember-me"
* remember-me-cookie 存储自动登录token的cookie名，默认是"remember-me"
* key 设置用来防止修改 token 的key，不设置将采用随机值
* services-ref 指定自定义的RememberMeServices
* use-secure-cookie 使用安全的cookie(只能在HTTPS下被发送)
* user-service-ref 指定UserDetailsService
* token-validity-seconds 设置token的有效期，默认是14days
* data-source-ref 设置一个数据源，如果设置将会使用PersistentTokenBasedRememberMeServices

##### <[session-management](https://www.cnblogs.com/fenglan/p/5913352.html)>

添加一个默认的SessionManagementFilter 到我们的过滤器链中来实现session的管理

属性：

* invalid-session-url 当提交一个无效的sessionId时会重定向到一个指定的URL
* session-authentication-strategy-ref 指定一个session认证策略SessionAuthenticationStrategy
* session-authentication-error-url 指定当session认证发生异常时，跳转的页面。默认返回401未授权错误码。
* [session-fixation](https://blog.csdn.net/h_mxc/article/details/50542038)-protection 当用户认证时指定会话固定保护。none、newSession、migrateSession、changeSessionId

子元素

<concurrency-control>

* 会话并发控制，允许用户的当前活跃会话数。
* max-sessions 最大会话数。-1表示不限制
* session-registry-ref 指定SessionRegistry实现
* error-if-maximum-exceeded If set to "true" a SessionAuthenticationException will be raised when a user attempts to exceed the maximum allowed number of sessions. The default behaviour is to expire the original session.
* expired-url The URL a user will be redirected to if they attempt to use a session which has been "expired" by the concurrent session controller because the user has exceeded the number of allowed sessions and has logged in again elsewhere. Should be set unless exception-if-maximum-exceeded is set. If no value is supplied, an expiry message will just be written directly back to the response.

### Authentication Services

SpringSecurity的认证服务

#### <authentication-manager>

认证管理器，为应用提供认证服务。

属性：

* erase-credentials If set to true, the AuthenticationManager will attempt to clear any credentials data in the returned Authentication object, once the user has been authenticated. Literally it maps to the eraseCredentialsAfterAuthentication property of the ProviderManager
* alias This attribute allows you to define an alias name for the internal instance for use in your own configuration.
* id

子元素

##### <authentication-provider>

配置认证提供者。可以使用ref属性指定认证提供者。如果没有，默认是[DaoAuthenticationProvider](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/" \l "core-services-dao-provider" \o "10.1.2 DaoAuthenticationProvider). DaoAuthenticationProvider使用UserDetailsService加载用户信息，然后与用户提交的用户/密码比对。

属性：

* ref 指定DaoAuthenticationProvider实现
* user-service-ref 指定UserDetailsService的实现

子元素：

###### <jdbc-user-service>

创建一个JDBC-based UserDetailsService

* authorities-by-username-query 指定一个授权SQL查询用户的授权信息，默认是

select username, authority from authorities where username = ?

* data-source-ref 指定数据源
* group-authorities-by-username-query 指定SQL查询用户的组权限信息，默认是

select

g.id, g.group\_name, ga.authority

from

groups g, group\_members gm, group\_authorities ga

where

gm.username = ? and g.id = ga.group\_id and g.id = gm.group\_id

* role-prefix 用户角色前缀，默认是ROLE\_，"none"表示没有前缀
* users-by-username-query 指定认证SQL，查询用户的用户名/密码/状态信息，默认是

select username, password, enabled from users where username = ?

Security Database Schema

数据库用户权限表模型

create table users(

username varchar\_ignorecase(50) not null primary key,

password varchar\_ignorecase(50) not null,

enabled boolean not null

);

create table authorities (

username varchar\_ignorecase(50) not null,

authority varchar\_ignorecase(50) not null,

constraint fk\_authorities\_users foreign key(username) references users(username)

);

create unique index ix\_auth\_username on authorities (username,authority);

数据库组权限模型

create table groups (

id bigint generated by default as identity(start with 0) primary key,

group\_name varchar\_ignorecase(50) not null

);

create table group\_authorities (

group\_id bigint not null,

authority varchar(50) not null,

constraint fk\_group\_authorities\_group foreign key(group\_id) references groups(id)

);

create table group\_members (

id bigint generated by default as identity(start with 0) primary key,

username varchar(50) not null,

group\_id bigint not null,

constraint fk\_group\_members\_group foreign key(group\_id) references groups(id)

);

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | users | | |  |  |  |  |  |  |
|  |  | 字段 | 类型 | 含义 |  |  |  |  |  |  |
| |  | | --- | |  | |  | username | varchar(50) | 用户名 | |  | | --- | |  | |  | authorities | | |  |
|  |  | password | varchar(50) | 密码 |  |  | 字段 | 类型 | 含义 |  |
|  |  | enabled | boolean | 是否可用 |  |  | username | varchar(50) | 用户名 |  |
|  |  |  |  |  |  |  | authority | varchar(50) | 权限 |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | group\_members | | |  |  |  |  |  |  |
|  |  | 字段 | 类型 | 含义 |  |  |  |  |  |  |
|  |  | id | bigint | 主键 |  |  |  |  |  |  |
|  |  | username | varchar(50) | 用户名 |  |  |  |  |  |  |
| |  | | --- | |  | |  | group\_id | bigint | 组ID |  |  |  |  |  |  |
|  |  |  |  |  |  |  | group\_authorities | | |  |
|  |  |  |  |  |  |  | 字段 | 类型 | 含义 |  |
|  |  | groups | | | |  | | --- | |  | |  | group\_id | bigint | 组ID |  |
|  |  | 字段 | 类型 | 含义 |  |  | authority | varchar\_ignorecase(50) | 权限 |  |
|  |  | id | bigint | 主键 |  |  |  |  |  |  |
|  |  | group\_name | varchar\_ignorecase(50) | 组名称 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

###### <password-encoder>

为认证提供者指定一个密码加密/解密规则。

属性：

* base64 base64加密
* hash 指定一种hash算法用于加密
* ref 指定一个PasswordEncoder实现

子元素 <salt-source> 指定加密的盐

* ref 指定一个spring bean
* system-wide 一个加单的值作为加密的盐
* user-property 使用用户的某个属性字段作为加密的盐，比如usernane

Security Database Schema

###### <user-service>

提供一个基于内存的UserDetailsService实现

属性：

* properties 指定一个配置文件来配置我们的用户信息，配置文件格式如下

username=password,grantedAuthority[,grantedAuthority][,enabled|disabled]

子元素：

<user> 指定用户信息

属性：

* + name 指定用户名
  + password 指定用户密码
  + authorities 指定授予用户的权限，多个权限间用逗号分隔，如"ROLE\_USER,ROLE\_ADMINISTRATOR"
  + disabled 标记当前用户是不安全且不可用的
  + locked 标记当前用户被锁定且不安全的

### [Method Security](https://blog.csdn.net/itguangit/article/details/78937840)

#### 基于XML配置

intercept-methods

<beans:bean class=*"com.itcast.test.service.UserServiceImpl"* id=*"userService"*>

<intercept-methods >

<protect access=*"ROLE\_ADMIN"* method=*"search"*/>

</intercept-methods>

</beans:bean>

protect-pointcut

<beans:bean class=*"com.itcast.test.service.UserServiceImpl"* id=*"userService"*>

</beans:bean>

<global-method-security proxy-target-class=*"false"*>

<protect-pointcut access=*"ROLE\_ADMIN"* expression=*"execution(\* com.itcast.test.service.UserServiceImpl.search(..))"*/>

</global-method-security>

#### 基于注解

##### JSR-250注解

Spring\_security.xml如下配置，开启JSR-250注解：

<security:global-method-security jsr250-annotations="enabled"/>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>jsr250-api</artifactId>

<version>1.0</version>

</dependency>

在对应类的方法上使用注解@RolesAllowed(方法与类上)、@PermitAll(方法与类上)、@DenyAll(方法上)

@RolesAllowed("ROLE\_ADMIN")

**public** String search(){

**return** "hello spring security......";

}

##### @Secured注解

spring\_security.xml如下配置，开启Secured注解：

<security:global-method-security secured-annotations="enabled"/>

在类和方法上使用如下注解

@Secured("ROLE\_ADMIN")

##### 支持表达式的注解

Spring Security中定义了四个支持使用表达式的注解，分别是@PreAuthorize、@PostAuthorize、@PreFilter和@PostFilter。其中前两者可以用来在方法调用前或者调用后进行权限检查，后两者可以用来对集合类型的参数或者返回值进行过滤。要使它们的定义能够对我们的方法的调用产生影响我们需要设置global-method-security元素的pre-post-annotations=”enabled”，默认为disabled。

 <security:global-method-security pre-post-annotations="disabled"/>

**使用@PreAuthorize和@PostAuthorize进行访问控制**

@Service

**public class** UserServiceImpl **implements** UserService {

   @PreAuthorize("hasRole('ROLE\_ADMIN')")

**public void** addUser(User user) {

      System.out.println("addUser................" + user);

   }

   @PreAuthorize("hasRole('ROLE\_USER') or hasRole('ROLE\_ADMIN')")

**public** User find(**int** id) {

      System.out.println("find user by id............." + id);

**return null**;

   }

}

在上面的代码中我们定义了只有拥有角色ROLE\_ADMIN的用户才能访问adduser()方法，而访问find()方法需要有ROLE\_USER角色或ROLE\_ADMIN角色。使用表达式时我们还可以在表达式中使用方法参数。

/\*\*

    \* 限制只能查询Id小于10的用户

    \*/

   @PreAuthorize("#id<10")

**public** User find(**int** id) {

      System.out.println("find user by id........." + id);

**return null**;

   }

   /\*\*

    \* 限制只能查询自己的信息

    \*/

   @PreAuthorize("principal.username.equals(#username)")

**public** User find(String username) {

      System.out.println("find user by username......" + username);

**return null**;

   }

   /\*\*

    \* 限制只能新增用户名称为abc的用户

    \*/

   @PreAuthorize("#user.name.equals('abc')")

**public void** add(User user) {

      System.out.println("addUser............" + user);

   }

}

在上面代码中我们定义了调用find(int id)方法时，只允许参数id小于10的调用；调用find(String username)时只允许username为当前用户的用户名；定义了调用add()方法时只有当参数user的name为abc时才可以调用。

有时候可能你会想在方法调用完之后进行权限检查，这种情况比较少，但是如果你有的话，Spring Security也为我们提供了支持，通过@PostAuthorize可以达到这一效果。使用@PostAuthorize时我们可以使用内置的表达式returnObject表示方法的返回值。我们来看下面这一段示例代码。

@PostAuthorize("returnObject.id%2==0")

**public** User find(**int** id) {

      User user = **new** User();

      user.setId(id);

**return** user;

   }

**使用@PreFilter和@PostFilter进行过滤**

使用@PreFilter和@PostFilter可以对集合类型的参数或返回值进行过滤。使用@PreFilter和@PostFilter时，Spring Security将移除使对应表达式的结果为false的元素。

 @PostFilter("filterObject.id%2==0")

**public** List<User> findAll() {

      List<User> userList = **new** ArrayList<User>();

      User user;

**for** (**int** i=0; i<10; i++) {

         user = **new** User();

         user.setId(i);

         userList.add(user);

      }

**return** userList;

   }

上述代码表示将对返回结果中id不为偶数的user进行移除。filterObject是使用@PreFilter和@PostFilter时的一个内置表达式，表示集合中的当前对象。当@PreFilter标注的方法拥有多个集合类型的参数时，需要通过@PreFilter的filterTarget属性指定当前@PreFilter是针对哪个参数进行过滤的。如下面代码就通过filterTarget指定了当前@PreFilter是用来过滤参数ids的

@PreFilter(filterTarget="ids", value="filterObject%2==0")

**public void** delete(List<Integer> ids, List<String> usernames) {

      ...

   }

## Java配置

Spring Security提供的基于java bean的配置。这里不再描述，详见spring security[官方文档](https://docs.spring.io/spring-security/site/docs/4.2.7.BUILD-SNAPSHOT/reference/htmlsingle/#hello-web-security-java-configuration)

General support for Java Configuration was added to Spring Framework in Spring 3.1. Since Spring Security 3.2 there has been Spring Security Java Configuration support which enables users to easily configure Spring Security without the use of any XML.

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.\*;

**import** org.springframework.security.config.annotation.authentication.builders.\*;

**import** org.springframework.security.config.annotation.web.configuration.\*;

*@EnableWebSecurity*

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

*@Bean*

**public** UserDetailsService userDetailsService() **throws** Exception {

InMemoryUserDetailsManager manager = **new** InMemoryUserDetailsManager();

manager.createUser(User.withUsername("user").password("password").roles("USER").build());

**return** manager;

}

}

# Spring Security的原理解析

比较复杂自己百度去吧！！！

# 问题

**1、说说在你的项目中是怎么使用spring security？**

**2、spring security可以实现类似shiro的方法级别粒度的权限控制吗？**

**3、你们项目中如何实现同一个用户同时只能在一处登录的（挤出登录）？**

**4、你们项目中实现了记住登录功能吗？**

**5、 session劫持？**

**同一解决方案：https httponly（）**

**会话固定攻击（session fixation attack）**

[**https://blog.csdn.net/youanyyou/article/details/79406499**](https://blog.csdn.net/youanyyou/article/details/79406499)