OAuth 2 开发人员指南（Spring security oauth2）

入门

这是支持OAuth2.0的用户指南。对于OAuth1.0，一切都是不同的，所以看它的用户指南。

本用户指南分为两个部分，第一部分是OAuth2.0提供端（OAuth 2.0 Provider），第二部分是OAuth2.0的客户端（OAuth 2.0 Client）

OAuth2.0提供端

OAuth2.0的提供端的用途是负责将受保护的资源暴露出去。配置包括建立一个可以访问受保护的资源的客户端代表。提供端是通过管理和验证可用于访问受保护的资源的OAuth 2令牌来做的。在适当的地方，提供端也必须为用户提供一个用于确认客户端 是否能够访问受保护的资源的接口（也就是一个页面或是一个窗口）。

在 OAuth 2 提供者其实是分成授权服务和资源服务两个角色的，而这两个角色有时是存在于同一个应用程序中的，通过 Spring Security OAuth 你可以有选择的将它们分裂到两个应用程序中，也可以有选择的给授权服务配置多个资源服务。获取令牌（Tokens）的请求是由Spring MVC的控制器端点来处理的，访问受保护的资源是通过标准的Spring Security请求过滤器来处理的。

下面列举的端点是 Spring Security 过滤器链实现 OAuth 2 授权服务器必须的端点：

- AuthorizationEndpoint 是用于授权服务请求的。默认的URL是：/oauth/authrize。
- TokenEndpoint 是用于获取访问令牌（Access Tokens）的请求。默认的URL是：/oauth/token。

下面过滤器是实现一个OAuth2资源服务器所必须的：

- OAuth2AuthenticationProcessingFilter 这个过滤器是用于加载请求提供的一个授权了的访问令牌是否有效。

对于所有的OAuth 2.0的提供端，通过使用Spring OAuth专门的@Configuration适配器来简化配置。也可以通过XML命名空间来配置OAuth，XML的schema存在：http://www.springframework.org/schema/security/spring-security-oauth2.xsd。命令空间是http://www.springframework.org/schema/security/oauth2

授权服务器配置

配置授权服务器时,您必须考虑的grant类型，从终端用户那，客户端会使用获取到的访问令牌(如授权代码、用户凭证、刷新令牌)。服务器的配置是用于提供一些实现，像客户端细节服务和令牌服务还有启用或禁用某些方面的全球机制。但是请注意,每个客户端可以专门配置权限能够使用某些授权机制和访问授权。即仅仅因为您的提供者配置为支持"客户证书"grant类型,并不意味着一个特定的客户端被授权使用grant类型。

@EnableAuthorizationServer 注释用于配置 OAuth 2.0 授权服务器机制，加上任意一个@Beans 去实现 AuthorizationServerConfigurer (这是一个hander适配器实现的空方法)。以下功能委托给由 spring 创造的独立 configurers 而且传递到 AuthorizationServerConfigurer：

- ClientDetailsServiceConfigurer:这个configurer定义了客户端细节服务。客户详细信息可以被初始化,或者你可以参考现有的商店。
- AuthorizationServerSecurityConfigurer:在令牌端点上定义了安全约束。
- AuthorizationServerEndpointsConfigurer:定义了授权和令牌端点和令牌服务

提供端配置中重要的一项是提供给OAuth 客户端的授权码。 OAuth 客户端通过将终端用户导向一个可以输入证书/口令的授权验证页面来获取授权码，然后，将授权码传递给提供端授权服务器，服务器验证后重定向页面。 在 OAuth 2 说明文档中有详细的示例。

在 xml 配置文件中，可以使用<authorization-server/>节点配置 OAuth 2.0 授权服务器。

配置客户端详细步骤

ClientDetailsServiceConfigurer 类（AuthorizationServerConfigurer类中的一个调用类）可以用来定义一个基于内存的或者JDBC的客户端信息服务。

客户端对象重要的属性有：

- clientId：（必须）客户端id。

- secret：（对于可信任的客户端是必须的）客户端的私密信息。
- scope：客户端的作用域。如果scope未定义或者为空（默认值），则客户端作用域不受限制。
- authorizedGrantTypes：授权给客户端使用的权限类型。默认值为空。
- authorities：授权给客户端的权限（Spring普通的安全权限）。

在运行的应用中，可以通过直接访问隐藏的存储文件（如：JdbcClientDetailsService中用到的数据库表）或者通过实现ClientDetailsManager 接口（也可以实现ClientDetailsService 接口，或者实现两个接口）来更新客户端信息。

管理令牌

AuthorizationServerTokenServices 接口里定义了 OAuth 2.0 令牌的操作方法。 注意以下几点：

- 创建一个访问令牌时，必须保存权限信息，这样后续令牌才可以引用它。
- 访问令牌用于加载创建令牌时的授权信息。

创建AuthorizationServerTokenServices 接口的实现类时，你可以考虑使用DefaultTokenServices 类，它使用随机值创建令牌，并处理除永久令牌以外的所有令牌，对于永久令牌，它委托TokenStore类进行处理。 令牌默认采用基于内存实现的存储方式，但也有一些其它的存储方式。 下面是其中一些方式的简介：

- 默认的InMemoryTokenStore 处理类对于单服务器场景非常适用（优点有：低阻塞，宕机时无需热切换到备份服务器）。大部分项目可以在开始时或者在开发模式下使用这种方式，这样比较容易启动一个没有其它依赖的服务器。
- JdbcTokenStore 类是实现存储令牌的JDBC 版本，它将令牌信息保存到关系型数据库中。 如果服务器间共享数据库或者同一个服务器有多个实例或者授权服务器、资源服务器有多个组件，那么可以使用JDBC方式存储令牌。 使用JdbcTokenStore 类时，需要将spring-jdbc组件jar包添加到工程的编译路径中。
- JSON网页令牌（JWT）可以加密所有令牌授权访问的数据（因此不需要在后台存储数据，这就是JWT一个重要的优点）。 缺点是你不能方便地撤销一个已授权的令牌（因此一般它们被授权的有效期较短，撤销授权的操作在刷新令牌中进行）。 另一个缺点是存储的令牌数据会越来越大因为在令牌里面存储了大量的用户证书信息。 JwtTokenStore 类不是一个真正的存储类，它未持久化（保存）任何数据，但是它在传输令牌信息和授权信息（在DefaultTokenServices类中实现）中扮演了同样的角色。 JwtTokenStore（接口）类依赖JwtAccessTokenConverter类，授权服务器和资源服务器都需要接口的实现类（因此他们可以安全地使用相同的数据并进行解码）。 令牌以默认方式进行签名，资源服务器为了能够验证这些签名，它需要有与授权服务器相同的对称密钥（服务器共享对称密钥），或者它需要有可以匹配签名私钥的公钥（公有私有密钥或混合密钥）。 为了使用JwtTokenStore 类，你需要在工程编译路径下添加spring-security-jwt组件jar包（你可以在Spring OAuth的github资源库中找到，但是两者的版本号是不一致的）。

Grant 类型

AuthorizationEndpoint 通过 AuthorizationServerEndpointsConfigurer 可以配置支持 Grant 类型。默认情况下支持所有的 grant 类型，除了密码(有关详细信息,请参阅下面的信息如何开启和关闭)。以下属性影响grant类型：

- authenticationManager：密码授予被注入一个authenticationManager开启。
- authorizationCodeServices：为了身份验证代码授予定义了授权代码服务(org.springframework.security.oauth2.provider.code.AuthorizationCodeServices实例).
- implicitGrantService：在隐式授予管理状态。
- tokenGranter:TokenGranter(完全掌控的授予和忽略上面的其他属性)在XML grant类型包括authorization-server的子元素。

配置端点的URL

AuthorizationServerEndpointsConfigurer有一个pathMapping()方法。该方法有两个参数：

- defaultPath 默认的端点URL
- customPath 自定义的URL

框架自己提供的URL路径是/oauth/authorize（授权端），/oauth/token (令牌端)，/oauth/confirm_access (用户发送确认授权到这里)，还有/oauth/error (用户呈现授权服务器授权出错的请求)。

注意：授权端/oauth/authorize（或者是它的影射）应该是受Spring Security保护的，所以它只能被已授权的用户访问。令牌端默认是通过使用支持使用HTTP基本身份验证客户机的秘密的注解@Configuration，被Spring Oauth保护的，但不是使用XML文件的（所以在这种情况下它被保护是很明确的）。

使用XML的<authorization-server/>元素可以使用一些属性来达到改变默认的端点URL。

自定义错误处理

授权服务器上错误处理使用标准的 Spring MVC 功能,即 @ExceptionHandler 端点本身的方法。用户还可以提供一个 WebResponseExceptionTranslator 端点本身,最好的办法是改变响应的内容而不是他们呈现的方式。异常的呈现代表 HttpMesssageConverters (这个可以添加到MVC的配置中)令牌的端点和OAuth的错误视图(/ OAuth /error)的授权端点。提供一个

whitelabel错误端点,但是用户可能需要提供一个自定义的实现(例如,就添加一个 @Controller，它的请求映射是 @RequestMapping(“/oauth /error”))。

配置资源服务器

资源服务器（可能和授权服务器或者一个单独的应用程序在同一台主机上）提供被OAuth2 令牌保护的资源。 Spring OAuth提供一个Spring Security授权过滤器，它实现保护资源的功能。在@Configuration类中，你可以使用@EnableResourceServer来开启/关闭过滤器，使用ResourceServerConfigurer来配置它。 下面是可配置的属性：

- tokenServices：定义令牌服务的实体（ResourceServerTokenServices类的实例）。
- resourceId：资源ID（可选的，建议配置，如果不为空，授权服务器会对它进行验证）。

@EnableResourceServer注解把一个 OAuth2AuthenticationProcessingFilter 类型过滤器添加到Spring Security 过滤链中。

在 XML中，有一个<resource-server/>元素，它有一个id属性 – 这是servlet过滤器的bean id，它过滤哪些可以被添加到Spring Security链中。

配置OAuth-Aware表达式处理器

你可能想要使用Spring Security的使用表达式语言配置访问控制的优点。 表达式处理器在 @EnableResourceServer配置中以默认方式进行注册。 表达式包括*#oauth2.clientHasRole*,*#oauth2.clientHasAnyRole*, 和 *#oath2.denyClient*，这些可以提供基于oauth客户端角色的访问控制（详细列表见OAuth2SecurityExpressionMethods）。 在XML中，你可以在<http/> 安全配置节点内使用expression-handler元素注册一个oauth-aware表达式处理器。

OAuth 2.0 客户端

OAuth 2.0 客户端控制着 OAuth 2.0保护的其它服务器的资源的访问权限。 配置包括建立相关受保护资源与有权限访问资源的用户之间的连接。 客户端也需要实现存储用户的授权代码和访问令牌的功能。

受保护资源的配置

受保护的资源（或称为远程资源）可以使用OAuth2ProtectedResourceDetails类型的实体bean定义。 一个受保护的资源有以下属性：

- id：资源id。它仅在客户端搜索资源的时候使用；在OAuth协议中它从未被用到。它也被用作bean的id。
- clientId：OAuth客户端id。OAuth提供端依赖这个id来识别客户端。
- clientSecret：与资源有关的秘密。默认情况下，该值不为空。
- accessTokenUri：提供访问口令的OAuth提供者终端的统一资源标识符（URI）。
- scope：以逗号分隔的字符串列表，标识可访问资源的范围。默认情况下，该值为空。
- clientAuthenticationScheme： 客户端对访问的令牌终端授权时使用的机制。 建议值： "http_basic" 和 "form"。 默认值： "http_basic"。 见OAuth 2 帮助文档2.1节。

不同的授权类型有不同的实现OAuth2ProtectedResourceDetails （对于client_credentials授权类型，使用ClientCredentialsResource ）的方式。对于需要进行用户身份验证的授权类型，还有一个属性：

- userAuthorizationUri： 用户访问资源需要身份验证时跳转页面的URI。 注意这个字段不是必填的，它依赖于被支持的OAuth 2的配置文件类型。

在XML中，可以使用<resource/>元素创建一个OAuth2ProtectedResourceDetails类型的实体bean。 它有上面提到的所有的属性。

使用@EnableOAuth2Client对OAuth2.0的客户端的配置比较简单。主要要做两件事情：

- 创建一个过滤bean(用oauth2ClientContextFilter)来存储当前的请求和上下文。在这种情况下需要在请求期间授权，来管理从重定向的位置和OAuth已认证的url。
- 在请求作用域中创建一个AccessTokenRequest类型的bean。这个类型的bean可以在个人用户的冲突中被识别码识别（或隐式的），授权客户端保持授权状态。

这个过滤器需要能够连接到应用（例如，使用Servlet初始化程序或者web.xml配置文件配置 一个和DelegatingFilterProxy相同名称的代理）。

在一个OAuth2RestTemplate中AccessTokenRequest可以这样使用：

```
@Autowired
private OAuth2ClientContext oauth2Context;

@Bean
```

```
public OAuth2RestTemplate sparklrRestTemplate() {
    return new OAuth2RestTemplate(sparklr(), oauth2Context);
}
```

OAuth2ClientContext可以在Session的作用域中保持不同用户不同的状态。如果没有这个功能，你需要在你自己的服务强上管理等效的数据结构，映射不用用户进来的请求并且用相互隔离的OAuth2ClientContext实例连接各个用户。

在XML文件中，有一个带有id属性的<client/>标签用于表示servlet的Filter的bean id，这个必须映射成一个和DelegatingFilterProxy有相同名称的@Configuration注解。

访问受保护的资源

一旦，你已经提供了所有的资源服务器配置时，现在你可以访问这些资源。建议的访问这些资源是通过使用在Spring3中提到的RestTemplate。Spring Security OAuth已经提供了一个扩展的RestTemplate，只需要提供一个OAuth2ProtectedResourceDetails的实例。使用它与user-tokens(授权代码授予)你应该考虑使用@EnableOAuth2Client配置(或XML等价< oauth:rest-template / >),造成了一些请求和会话作用域上下文对象,要求不同的用户在运行时不发生冲突。

客户端持久化访问令牌

客户端不需要持久化令牌,但是它可以很好的用户不需要批准一个新的令牌格兰特每次重新启动客户机应用程序。ClientTokenServices接口定义所需的操作持续OAuth 2.0为特定用户令牌。提供一个JDBC实现,但是如果你喜欢的话你可以实现自己的服务以存储的访问令牌持久数据库和相关的验证实例。如果你想使用此功能,你需要提供一个专门配置TokenProvider theOAuth2RestTemplate如

```
@Bean@Scope(value = "session", proxyMode = ScopedProxyMode.INTERFACES)public OAuth2RestOperations restTemplate() {
    OAuth2RestTemplate template = new OAuth2RestTemplate(resource(), new DefaultOAuth2ClientContext(accessTokenRequest));
    AccessTokenProviderChain provider = new AccessTokenProviderChain(Arrays.asList(new AuthorizationCodeAccessTokenProvider()));
    provider.setClientTokenServices(clientTokenServices());
    return template;}
```

为外部 **OAuth2** 提供者定制客户端

一些外部OAuth2提供商(如Facebook)不完全实现正确规范,否则他们只是困在一个旧版本比Spring Security OAuth的规范。在您的客户机应用程序使用这些供应商可能需要适应客户端基础设施的各个部分。

使用Facebook作为一个例子,有一个Facebook功能tonr2应用程序(您需要更改配置添加你自己的,有效的,客户机id和秘密,他们很容易生成在Facebook网站上)。

Facebook响应令牌也包含一个不一致的、有失效时间的JSON实体(他们使用到期而不是expires_in),所以如果你想在应用程序中使用失效时间你将不得不使用一个自定义OAuth2SerializationService手动解码。

本文地址：http://www.oschina.net/translate/oauth-2-developers-guide
原文地址：https://github.com/spring-projects/spring-security-oauth/blob/master/docs/oauth2.md

---