**OAuth 2.0 Provider**

OAuth 2.0提供程序机制负责公开OAuth 2.0受保护资源。 配置涉及建立可独立或代表用户访问其受保护资源的OAuth 2.0客户端。 提供商通过管理和验证用于访问受保护资源的OAuth 2.0令牌来实现此目的。 在适用的情况下，提供者还必须为用户提供接口，以确认客户端可以被授权访问受保护资源（即确认页面）。

**OAuth 2.0 Provider Implementation**

OAuth 2.0中的提供者角色实际上是在授权服务和资源服务之间分配的，虽然这些角色有时位于同一个应用程序中，但是使用Spring Security OAuth，您可以选择将它们分成两个应用程序，还可以拥有多个共享的资源服务 授权服务。 对令牌的请求由Spring MVC控制器端点处理，对受保护资源的访问由标准Spring Security请求过滤器处理。 Spring Security过滤器链中需要以下端点才能实现OAuth 2.0授权服务器：

AuthorizationEndpoint用于为授权请求提供服务。 默认网址：/ oauth / authorize。

TokenEndpoint用于为访问令牌的请求提供服务。 默认URL：/ oauth / token。

要实现OAuth 2.0资源服务器，需要以下过滤器：

OAuth2AuthenticationProcessingFilter用于在给定经过身份验证的访问令牌的情况下为请求加载身份验证。

对于所有OAuth 2.0提供程序功能，使用特殊的Spring OAuth @Configuration适配器简化了配置。 OAuth配置还有一个XML命名空间，架构位于http://www.springframework.org/schema/security/spring-security-oauth2.xsd。 命名空间是http://www.springframework.org/schema/security/oauth2。

**Authorization Server Configuration**

在配置授权服务器时，您必须考虑客户端用于从最终用户获取访问令牌的授权类型（例如授权代码，用户凭据，刷新令牌）。服务器的配置用于提供客户端详细信息服务和令牌服务的实现，以及全局启用或禁用该机制的某些方面。但请注意，每个客户端都可以专门配置权限，以便能够使用某些授权机制和访问授权。即仅仅因为您的提供程序配置为支持“客户端凭据”授予类型，并不意味着特定客户端有权使用该授权类型。

@EnableAuthorizationServer注释用于配置OAuth 2.0授权服务器机制，以及实现AuthorizationServerConfigurer的任何@Beans（有一个方便的适配器实现，使用空方法）。以下功能委派给由Spring创建并传递到AuthorizationServerConfigurer的独立配置器：

ClientDetailsServiceConfigurer：定义客户端详细信息服务的配置器。可以初始化客户端详细信息，也可以只引用现有商店。

AuthorizationServerSecurityConfigurer：定义令牌端点上的安全约束。

AuthorizationServerEndpointsConfigurer：定义授权和令牌端点以及令牌服务。

提供程序配置的一个重要方面是授权代码提供给OAuth客户端的方式（在授权代码授权中）。 OAuth客户端通过将最终用户引导到授权页面来获取授权代码，在该授权页面中用户可以输入她的凭证，从而导致从提供者授权服务器重定向到具有授权代码的OAuth客户端。 OAuth 2规范中详细说明了这方面的例子。

在XML中，有一个<authorization-server />元素，其使用方式与配置OAuth 2.0授权服务器的方式类似。

**Configuring Client Details**

ClientDetailsServiceConfigurer（来自AuthorizationServerConfigurer的回调）可用于定义客户端详细信息服务的内存或JDBC实现。客户的重要属性是

clientId :(必需）客户端ID。

secret :(可信客户端需要）客户端密钥（如果有）。

范围：客户受限的范围。如果范围未定义或为空（默认值），则客户端不受范围限制。

authorizedGrantTypes：授权客户端使用的授权类型。默认值为空。

权限：授予客户的权限（常规Spring Security权限）。

通过直接访问底层存储（例如，在JdbcClientDetailsService的情况下为数据库表）或通过ClientDetailsManager接口（ClientDetailsService的两个实现也实现），可以在正在运行的应用程序中更新客户端详细信息。

注意：JDBC服务的模式不与库打包在一起（因为您可能希望在实践中使用太多变体），但是有一个示例可以从github中的测试代码开始。

**Managing Tokens**

AuthorizationServerTokenServices接口定义管理OAuth 2.0令牌所需的操作。请注意以下事项：

创建访问令牌时，必须存储身份验证，以便接受访问令牌的资源可以在以后引用它。

访问令牌用于加载用于授权其创建的身份验证。

在创建AuthorizationServerTokenServices实现时，您可能需要考虑使用DefaultTokenServices，它具有许多可插入的策略来更改访问令牌的格式和存储。默认情况下，它通过随机值创建令牌，并处理除委托给TokenStore的令牌持久性之外的所有内容。默认存储是内存实现，但也有一些其他实现可用。这是一个描述，其中每个都有一些讨论

对于单个服务器，默认的InMemoryTokenStore是完全正常的（即，在发生故障的情况下流量较低且不与备份服务器进行热交换）。大多数项目可以从这里开始，也可以在开发模式下以这种方式运行，以便轻松启动没有依赖项的服务器。

JdbcTokenStore是同一个东西的JDBC版本，它将令牌数据存储在关系数据库中。如果可以在服务器之间共享数据库，则使用JDBC版本，如果只有一个服务器，则扩展同一服务器的实例;如果有多个组件，则使用授权和资源服务器。要使用JdbcTokenStore，您需要在类路径上使用“spring-jdbc”。

存储的JSON Web令牌（JWT）版本将有关授权的所有数据编码到令牌本身（因此根本没有后端存储，这是一个重要的优势）。一个缺点是您不能轻易撤销访问令牌，因此它们通常被授予短期到期，并且撤销在刷新令牌处理。另一个缺点是，如果您在其中存储大量用户凭据信息，则令牌可能会变得非常大。 JwtTokenStore实际上不是一个“商店”，因为它不会保留任何数据，但它在DefaultTokenServices中扮演相同的角色，即转换令牌值和身份验证信息。

注意：JDBC服务的模式不与库打包在一起（因为您可能希望在实践中使用太多变体），但是有一个示例可以从github中的测试代码开始。确保@EnableTransactionManagement可以防止在创建令牌时竞争相同行的客户端应用程序之间的冲突。另请注意，示例模式具有显式的PRIMARY KEY声明 - 这些声明在并发环境中也是必需的。

**JWT Tokens**

要使用JWT令牌，您需要在Authorization Server中使用JwtTokenStore。资源服务器还需要能够解码令牌，以便JwtTokenStore依赖于JwtAccessTokenConverter，并且授权服务器和资源服务器都需要相同的实现。默认情况下，令牌是签名的，资源服务器也必须能够验证签名，因此它需要与授权服务器（共享密钥或对称密钥）相同的对称（签名）密钥，或者它需要公共与授权服务器中的私钥（签名密钥）匹配的密钥（验证者密钥）（公共 - 私有或非对称密钥）。授权服务器在/ oauth / token\_key端点上公开公钥（如果可用），默认情况下该访问规则为“denyAll（）”。您可以通过将标准SpEL表达式注入AuthorizationServerSecurityConfigurer来打开它（例如，“permitAll（）”可能就足够了，因为它是一个公钥）。

要使用JwtTokenStore，您需要在类路径上使用“spring-security-jwt”（您可以在与Spring OAuth相同的github存储库中找到它，但具有不同的发布周期）。

**Grant Types**

AuthorizationEndpoint支持的授权类型可以通过AuthorizationServerEndpointsConfigurer进行配置。默认情况下，除密码外都支持所有授权类型（有关如何打开密码的详细信息，请参阅下文）。以下属性会影响授权类型：

authenticationManager：通过注入AuthenticationManager来打开密码授予。

userDetailsService：如果您注入UserDetailsService或者无论如何全局配置（例如在GlobalAuthenticationManagerConfigurer中），则刷新令牌授权将包含对用户详细信息的检查，以确保该帐户仍处于活动状态

authorizationCodeServices：为授权代码授权定义授权代码服务（AuthorizationCodeServices的实例）。

implicitGrantService：在imlpicit授权期间管理状态。

tokenGranter：TokenGranter（完全控制授权并忽略上面的其他属性）

在XML中，授权类型包含在授权服务器的子元素中。

**Configuring the Endpoint URLs**

AuthorizationServerEndpointsConfigurer具有pathMapping（）方法。 它需要两个参数：

端点的默认（框架实现）URL路径

所需的自定义路径（以“/”开头）

框架提供的URL路径是/ oauth / authorize（授权端点），/ oauth / token（令牌端点），/ oauth / confirm\_access（用户在此处发布授权批准），/ oauth / error（用于呈现错误） 在授权服务器中），/ oauth / check\_token（由资源服务器用于解码访问令牌）和/ oauth / token\_key（如果使用JWT令牌，则公开用于令牌验证的公钥）。

注： 应使用Spring Security保护授权端点/ oauth / authorize（或其映射的替代方案），以便只有经过身份验证的用户才能访问它。 例如，使用标准的Spring Security WebSecurityConfigurer：

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests().antMatchers("/login").permitAll().and()

// default protection for all resources (including /oauth/authorize)

.authorizeRequests()

.anyRequest().hasRole("USER")

// ... more configuration, e.g. for form login

}

注意：如果您的授权服务器也是资源服务器，则还有另一个优先级较低的安全过滤器链控制API资源。 对于那些受访问令牌保护的请求，您需要将其路径与主要面向用户的过滤器链中的路径不匹配，因此请确保包含请求匹配器，该请求匹配器仅选取上述WebSecurityConfigurer中的非API资源。

默认情况下，Spring OAuth在@Configuration支持中使用客户机密钥的HTTP基本身份验证为您保护令牌端点。 这不是XML中的情况（因此应该明确地保护它）。

在XML中，<authorization-server />元素具有一些属性，可用于以类似的方式更改默认端点URL。 必须显式启用/ check\_token端点（使用check-token-enabled属性）。

**Customizing the UI**

大多数Authorization Server端点主要由计算机使用，但有一些资源需要UI，而且这些是/ oauth / confirm\_access的GET和来自/ oauth / error的HTML响应。它们是在框架中使用whitelabel实现提供的，因此授权服务器的大多数实际实例都希望提供自己的实例，以便它们可以控制样式和内容。您需要做的就是为这些端点提供带有@RequestMappings的Spring MVC控制器，并且框架默认值在调度程序中的优先级较低。在/ oauth / confirm\_access端点，您可以期望绑定到会话的AuthorizationRequest包含寻求用户批准所需的所有数据（默认实现是WhitelabelApprovalEndpoint，因此请查找复制的起点）。您可以从该请求获取所有数据并根据需要进行渲染，然后用户需要做的就是POST回到/ oauth / authorize，其中包含有关批准或拒绝授权的信息。请求参数直接传递给AuthorizationEndpoint中的UserApprovalHandler，因此您可以根据需要或多或少地解释数据。默认的UserApprovalHandler取决于您是否在AuthorizationServerEndpointsConfigurer中提供了ApprovalStore（在这种情况下，它是ApprovalStoreUserApprovalHandler）（在这种情况下，它是TokenStoreUserApprovalHandler）。标准批准处理程序接受以下内容：

TokenStoreUserApprovalHandler：通过user\_oauth\_approval的简单是/否决定等于“true”或“false”。

ApprovalStoreUserApprovalHandler：一组范围。\*参数键，其中“\*”等于所请求的范围。 参数的值可以是“true”或“approved”（如果用户批准了授权），否则用户被视为拒绝该范围。 如果至少批准一个范围，则授予成功。

注意：不要忘记在为用户呈现的表单中包含CSRF保护。 Spring Security期望默认情况下名为“\_csrf”的请求参数（并且它在请求属性中提供值）。 有关详细信息，请参阅Spring Security用户指南，或查看whitelabel实现以获取指导。

**Enforcing SSL**

普通HTTP适用于测试，但授权服务器只能在生产中通过SSL使用。 您可以在安全容器或代理后面运行应用程序，如果正确设置代理和容器（这与OAuth2无关），它应该可以正常工作。 您可能还希望使用Spring Security requiresChannel（）约束来保护端点。 对于/ authorize端点，您可以将其作为正常应用程序安全性的一部分。 对于/ token端点，AuthorizationServerEndpointsConfigurer中有一个标志，您可以使用sslOnly（）方法设置该标志。 在这两种情况下，安全通道设置都是可选的，但如果它在不安全的通道上检测到请求，则会导致Spring Security重定向到它认为的安全通道。