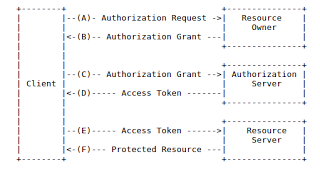
# Spring Security Oauth 入门

本文以Oauth2.0协议为准，讲解基于Spring Security框架的Spring Oauth实现。

参考文献： Pro\_Spring\_Security.pdf

<http://www.tuicool.com/articles/bYZBbe>



<http://localhost:8080/tonr2/sparklr/photos> 流程：

1. OAuth2ClientContextFilter#doFilter
2. SparklrController#photos

OAuth2RestTemplate

令牌实例：OAuth2AccessToken

客户端，令牌将被储存在上下文**DefaultOAuth2ClientContext**对象中，DefaultOAuth2ClientContext对象还包含令牌请求对象**AccessTokenRequest**，通过配置<oauth:rest-template resource="sparklr" />，AccessTokenRequest 和DefaultOAuth2ClientContext将自动在Spring容器启动时自动配置，**RestTemplateBeanDefinitionParser**类将负责这个解析这个xml。

如果上下文对象（DefaultOAuth2ClientContext）中没有令牌，AccessTokenProvider（接口）将负责尝试获取和接收令牌，这时，**AccessTokenProviderChain**将被使用，该对象有List<AccessTokenProvider> chain和**ClientTokenServices** clientTokenServices两个成员属性，当需要获得令牌时，其obtainAccessToken方法将被调用，入参为**OAuth2ProtectedResourceDetails**和AccessTokenRequest实例，OAuth2ProtectedResourceDetails实例可以通过<oauth:rest-template>节点的资源属性<oauth:resource>来配置，例如：

<oauth:resource id=*"sparklr"* type=*"authorization\_code"*

client-id=*"tonr"* client-secret=*"secret"* access-token-uri=*"${accessTokenUri}"*

user-authorization-uri=*"${userAuthorizationUri}"* scope=*"read,write"* />

<oauth:rest-template resource=*"sparklr"* />

**AuthorizationCodeResourceDetails**

**AuthorizationCodeAccessTokenProvider**作为AccessTokenProvider的实现之一，会通过调用<http://localhost:8080/sparklr2/oauth/authorize>（POST请求）来为客户端和用户获取授权码（authorization code）

当请求/sparklr2/oauth/authorize时，服务端的**FilterSecurityInterceptor**将被使用，如果根据配置，用户（客户端）需要ROLE\_USER角色才能访问该地址，如果没有，将抛出**AccessDeniedException**异常，该异常会被**ExceptionTranslationFilter**捕获，并将其重定向到<http://localhost:8080/sparklr2/login.jsp>，这个重定向请求将被客户端的**AuthorizationCodeAccessTokenProvider**给捕获，它将抛出一个**UserRedirectRequiredException**异常，这个异常将被Oauth2ClientContextFilter捕获，Oauth2ClientContextFilter捕获此异常后，将去调用这个重定向的地址，当然此时会带上一些额外的参数，像这样：<http://localhost:8080/sparklr2/login.jsp?response_type=code&client_id=tonr&scope=read+write&state=dWby7l&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Ftonr2%2Fsparklr%2Fphotos>

这时候出现在我们面前的是来自服务端的登录页面，我们输入完用户名和密码后，点击登录，将会发送到/sparklr2/login.do（由<form-login>节点的属性login-processing-url="/login.do"配置），该地址默认被配置成**UsernamePasswordAuthenticationFilter**来处理，该过滤器将调用**AuthenticationManager**接口的实现**ProviderManager**来处理，AuthenticationManager发现**DaoAuthenticationProvider**被user details给配置在内存中，就是如下配置：

<authentication-manager alias=*"authenticationManager"*

xmlns=*"http://www.springframework.org/schema/security"*>

<authentication-provider>

<user-service id=*"userDetailsService"*>

<user name=*"marissa"* password=*"koala"* authorities=*"ROLE\_USER"* />

<user name=*"paul"* password=*"emu"* authorities=*"ROLE\_USER"* />

</user-service>

</authentication-provider>

</authentication-manager>

DaoAuthenticationProvider发现了已经配置了marissa的角色ROLE\_USER，并验证成功，于是将重定向到原来请求/sparklr2/oauth/authorize的URL，将对这个到端点访问授予给刚刚登录的用户。

/oauth/authorize的请求在服务端的被Spring Security Oauth项目的核心类org.springframework.security.oauth2.provider.endpoint.**AuthorizationEndpoint**来处理，使用是只需如下配置即可：

<oauth:authorization-server

client-details-service-ref=*"clientDetails"* token-services-ref=*"tokenServices"*

user-approval-handler-ref=*"userApprovalHandler"*>

<oauth:authorization-code />

<oauth:implicit />

<oauth:refresh-token />

<oauth:client-credentials />

<oauth:password />

</oauth:authorization-server>一旦<oauth:authorization-server>节点被配置，它将被**AuthorizationServerBeanDefinitionParser**类来解析，该类用来解析和处理许多不同的元素和设置。当/oauth/authorize 请求过来，AuthorizationEndpoint做的第一件事情就是检查客户端做的是何种类型的请求，基本上是通过看请求参数response\_type，检查他的值是code还是token，就知道客户端授权码（authorization code）还是访问令牌（access token）正被请求，接下来AuthorizationEndpoint所要做的是创建一个org.springframework.security.oauth2.provider.**AuthorizationRequest**的实例化对象，AuthorizationRequest的主要一部分，是用客户端ID参数来检索连接的客户端的详细信息，在本例，客户端ID是tonr，详细信息由**InMemoryClientDetailsService**来检索，详见如下配置：

<oauth:client-details-service id=*"clientDetails"*>

<oauth:client client-id=*"my-trusted-client"* authorized-grant-types=*"password,authorization\_code,refresh\_token,implicit"*

authorities=*"ROLE\_CLIENT, ROLE\_TRUSTED\_CLIENT"* scope=*"read,write,trust"*

access-token-validity=*"60"* />

<oauth:client client-id=*"my-trusted-client-with-secret"*

authorized-grant-types=*"password,authorization\_code,refresh\_token,implicit"*

secret=*"somesecret"* authorities=*"ROLE\_CLIENT, ROLE\_TRUSTED\_CLIENT"* />

<oauth:client client-id=*"my-client-with-secret"*

authorized-grant-types=*"client\_credentials"* authorities=*"ROLE\_CLIENT"*

scope=*"read"* secret=*"secret"* />

<oauth:client client-id=*"my-less-trusted-client"*

authorized-grant-types=*"authorization\_code,implicit"* authorities=*"ROLE\_CLIENT"* />

<oauth:client client-id=*"my-less-trusted-autoapprove-client"*

authorized-grant-types=*"implicit"* authorities=*"ROLE\_CLIENT"* />

<oauth:client client-id=*"my-client-with-registered-redirect"* authorized-grant-types=*"authorization\_code,client\_credentials"*

authorities=*"ROLE\_CLIENT"* redirect-uri=*"http://anywhere?key=value"*

scope=*"read,trust"* />

<oauth:client client-id=*"my-untrusted-client-with-registered-redirect"*

authorized-grant-types=*"authorization\_code"* authorities=*"ROLE\_CLIENT"*

redirect-uri=*"http://anywhere"* scope=*"read"* />

<oauth:client client-id=*"tonr"* resource-ids=*"sparklr"*

authorized-grant-types=*"authorization\_code,implicit"* authorities=*"ROLE\_CLIENT"*

scope=*"read,write"* secret=*"secret"* />

</oauth:client-details-service>

当InMemoryClientDetailsService来检索上面一段xml是，会给每个client节点生成一个**ClientDetails**对象，AuthorizationRequest对象被创建后，InMemoryClientDetailsService将检查该对象是否已被批准，因为该对象是刚创建的，所以还未被批准，所以InMemoryClientDetailsService将转发该请求到框架默认配置的:/oauth/confirm\_access地址，作为授权服务（authentication server）的应用程序，该url需要被我们来处理，在当前情况下，由Sparklr应用程序来处理，所以，在Sparklr中是由**AccessConfirmationController**来处理到/oauth/confirm\_access的URL.AccessConfirmationController将显示一个包含授权和拒绝的页面。

当用户点击授权按钮时，在Sparklr中将产生一个去/oauth/authorize的 POST调用，这个调用将被AuthorizationEndpoint的approveOrDeny方法来处理，approveOrDeny将参数转换成一个AuthorizationRequest对象，AuthorizationRequest中的redirectUri将被再次进行评审处理，这次这个请求被批准，该请求还会被评估以确认他请求的是授权码（和AuthorizationRequest中的responseTypes）而不是访问令牌，框架将生成一个授权码，approveOrDeny完成后将返回一个RedirectView对象来重定向回客户端应用程序，比如：

<http://localhost:8080/tonr2/sparklr/photos;jsessionid=03B2E814391E010B3D1210241ECF6C0A?code=vqMbuf&state=aTSlVl>

重定向请求到达客户端Tonr后，现在Tonr将再一次处理这个URL，不过这一次，为了继续进行它将尝试获取访问令牌。之前，Tonr将会用OAuth2RestTemplate结合的AuthorizationCodeAccessTokenProvider尝试和检索来自Sparklr访问令牌，特别的是，他们会去带着参数（grant\_type=’authorization\_code’, redirect\_uri=’http://localhost:8080/tonr2/sparklr/photos’, code=xxxx）去/sparklr/oauth/token URL去服务端Sparklr请求。这个请求将被服务端的org.springframework.security.oauth2.provider.endpoint.**TokenEndpoint**来处理，TokenEndpoint将创建一个新的OAuth2AccessToken实例化对象并且生成一个承载令牌（Bearer token）来应答。

这样，客户端Tonr接收到该应答并且现在再一次尝试访问Sparklr，不过这一次将携带接受到的承载访问令牌作为请求中授权头中的一部分。Sparklr需要去验证该请求，**OAuth2AuthenticationProcessingFilter**将从请求中提取令牌并和**Oauth2AuthenticationManager**来一起验证和认证令牌。

当该令牌被验证了，这请求将最终到达服务端的控制器来为客户端请求服务。