# **spring cloud微服务权限认证**

微服务的两套认证：

1.身份认证在zuul加filter拦截获取请求user token，校验user token合法性

2.服务认证为内部服务之间通过server token来校验服务调用的合法性

整个流程的大概逻辑：

1.启动认证服务(auth-server)，此时会生成user的公钥私钥和服务server的公钥私钥

2.启动其他服务，启动后请求认证服务(auth-server)拿到对应的2套公钥，用于对2种token做解密认证用

整个调用请求通过zuul filter时校验token，zuul会通过启动时拿到的公钥对token认证，通过校验后可以在调用权限管理服务（permission-server）查询当前用户是否有访问此url的权限，在访问permission-server时先请求auth-server获取zuul 的server token，带上server token和user token去请求permission-server,在permission-server里的拦截器会对user token和server token做校验(也是启动的时候拿到那2个公钥),调用其他服务都是这个逻辑。

# **使用Spring Cloud Security OAuth2搭建授权服务**

## 添加依赖

添加依赖：spring-cloud-starter-security,spring-cloud-starter-oauth2

## 添加注解和配置

启动类中添加@EnableAuthorizationServer注解；

分配 client\_id, client\_secret。Spring Security OAuth2的配置方法是编写@Configuration类继承AuthorizationServerConfigurerAdapter，重写void configure(ClientDetailsServiceConfigurer clients)方法

@Override

public void configure(ClientDetailsServiceConfigurer clients) throws Exception {

clients.inMemory() // 使用in-memory存储

.withClient("client") // client\_id

.secret("secret") // client\_secret

.authorizedGrantTypes("authorization\_code") // 该client允许的授权类型

.scopes("app"); // 允许的授权范围

}

## 授权流程

# **Spring Cloud云架构 - SSO单点登录之OAuth2.0登录认证**

## oauth中的角色

**client**：调用资源服务器API的应用

Oauth 2.0 Provider：包括Authorization Server和Resource Server

（1）**Authorization Server**：认证服务器，进行认证和授权

（2）**Resource Server**：资源服务器，保护受保护的资源  
**user**：资源的拥有者

### Oauth 2.0 Provider

Authorization Server

（1）[AuthorizationEndpoint](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/endpoint/AuthorizationEndpoint.html" \o "AuthorizationEndpoint" \t "http://blog.csdn.net/qq_40354025/article/details/_blank):进行授权的服务，Default URL: /oauth/authorize

（2）[TokenEndpoint](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/endpoint/TokenEndpoint.html" \o "TokenEndpoint" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)：获取token的服务，Default URL: /oauth/token

**Resource Server**

[OAuth2AuthenticationProcessingFilter](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/authentication/OAuth2AuthenticationProcessingFilter.html" \o "OAuth2AuthenticationProcessingFilter" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)：给带有访问令牌的请求加载认证

### Authorization Server

**两个配置类**

分别继承AuthorizationServerConfigurerAdapter、WebSecurityConfigurerAdapter，再去复写里面的方法。主要出现的两种注解：

**@EnableAuthorizationServer**：声明一个认证服务器，当用此注解后，应用启动后将自动生成几个[Endpoint](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/endpoint/AuthorizationEndpoint.html" \o "AuthorizationEndpoint" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)：（注：其实实现一个认证服务器就是这么简单，加一个注解就搞定，当然真正用到生产环境还是要进行一些配置和复写工作的。）

/oauth/authorize：验证

/oauth/token：获取token

/oauth/confirm\_access：用户授权

/oauth/error：认证失败

/oauth/check\_token：资源服务器用来校验token

/oauth/token\_key：如果jwt模式则可以用此来从认证服务器获取公钥

以上这些endpoint都在源码里的endpoint包里面。

**@Beans**：需要实现AuthorizationServerConfigurer；AuthorizationServerConfigurer包含三种配置：

ClientDetailsServiceConfigurer：client客户端的信息配置，client信息包括：clientId、secret、scope、authorizedGrantTypes、authorities

（1）scope：表示权限范围，可选项，用户授权页面时进行选择

（2）authorizedGrantTypes：有四种授权方式

Authorization Code：用验证获取code，再用code去获取token（用的最多的方式，也是最安全的方式）

Implicit: 隐式授权模式

Client Credentials (用來取得 App Access Token)

Resource Owner Password Credentials

（3）authorities：授予client的权限

这里的具体实现有多种，in-memory、JdbcClientDetailsService、jwt等。

AuthorizationServerSecurityConfigurer：声明安全约束，哪些允许访问，哪些不允许访问

AuthorizationServerEndpointsConfigurer：声明授权和token的端点以及token的服务的一些配置信息，比如采用什么存储方式、token的有效期等

client的信息的读取：在ClientDetailsServiceConfigurer类里面进行配置，可以有in-memory、jdbc等多种读取方式。jdbc需要调用JdbcClientDetailsService类，此类需要传入相应的DataSource.

**管理token**

[AuthorizationServerTokenServices](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/token/AuthorizationServerTokenServices.html" \o "AuthorizationServerTokenServices" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)接口:声明必要的关于token的操作

（1）当token创建后，保存起来，以便之后的接受访问令牌的资源可以引用它。

（2）访问令牌用来加载认证

接口的实现也有多种，[DefaultTokenServices](http://docs.spring.io/spring-security/oauth/apidocs/org/springframework/security/oauth2/provider/token/DefaultTokenServices.html" \o "DefaultTokenServices" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)是其默认实现，他使用了默认的InMemoryTokenStore，不会持久化token；

token存储方式共有三种分别是：

（1）InMemoryTokenStore：存放内存中，不会持久化

（2）JdbcTokenStore：存放数据库中

（3）Jwt: json web token

**授权类型**

可以通过AuthorizationServerEndpointsConfigurer来进行配置，默认情况下，支持除了密码外的所有授权类型。相关授权类型的一些类：

（1）authenticationManager：直接注入一个AuthenticationManager，自动开启密码授权类型

（2）userDetailsService：如果注入UserDetailsService，那么将会启动刷新token授权类型，会判断用户是否还是存活的

（3）authorizationCodeServices：AuthorizationCodeServices的实例，auth code 授权类型的服务

（4）implicitGrantService：imlpicit grant

（5）tokenGranter：

**endpoint的URL的配置**

（1）AuthorizationServerEndpointsConfigurer的pathMapping()方法，有两个参数，第一个是默认的URL路径，第二个是自定义的路径

（2）WebSecurityConfigurer的实例，可以配置哪些路径不需要保护，哪些需要保护。默认全都保护。

自定义UI:

（1）有时候，我们可能需要自定义的登录页面和认证页面。登陆页面的话，只需要创建一个login为前缀名的网页即可，在代码里，设置为允许访问，这样，系统会自动执行你的登陆页。此登陆页的action要注意一下，必须是跳转到认证的地址。

（2）另外一个是授权页，让你勾选选项的页面。此页面可以参考源码里的实现，自己生成一个controller的类，再创建一个对应的web页面即可实现自定义的功能。

**授权获取token流程**

（1）端口号换成你自己的认证服务器的端口号，client\_id也换成你自己的，response\_type类型为code。

 localhost:8080/uaa/oauth/authorize?client\_id=client&response\_type=code&redirect\_uri=[http://www](http://www/" \t "http://blog.csdn.net/qq_40354025/article/details/_blank).baidu.com  
（2）这时候你将获得一个code值：[http://www](http://www/" \t "http://blog.csdn.net/qq_40354025/article/details/_blank).baidu.com/?code=G0C20Z

（3）使用此code值来获取最终的token：

curl -X POST -H "Cant-Type: application/x-www-form-urlencoded" -d 'grant\_type=authorization\_code&code=G0C20Z&redirect\_uri=[http://www.baidu.com](http://www.baidu.com/" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)' "[http://client:secret@localhost:8080/uaa/oauth/token](http://localhost:8080/oauth/token" \t "http://blog.csdn.net/qq_40354025/article/details/_blank)"

返回值：

{"access\_token":"b251b453-cc08-4520-9dd0-9aedf58e6ca3","token\_type":"bearer","expires\_in":2591324,"scope":"app"}

（4）用此token值来调用资源服务器内容（如果资源服务器和认证服务器在同一个应用中，那么资源服务器会自己解析token值，如果不在，那么你要自己去做处理）

curl -H "Authorization: Bearer b251b453-cc08-4520-9dd0-9aedf58e6ca3" "localhost:8081/service2（此处换上你自己的url）"

### Resource Server

保护资源，需要令牌才能访问。在配置类上加上注解@EnableResourceServer即启动。使用ResourceServerConfigurer进行配置：

（1）tokenServices：ResourceServerTokenServices的实例，声明了token的服务

（2）resourceId：资源Id，由auth Server验证。

（3）其它一些扩展点，比如可以从请求中提取token的tokenExtractor

（4）一些自定义的资源保护配置，通过HttpSecurity来设置

使用token的方式也有两种：

（1）Bearer Token（https传输方式保证传输过程的安全）:主流

（2）Mac（http+sign）

**如何访问资源服务器中的API？**

如果资源服务器和授权服务器在同一个应用程序中，并且您使用DefaultTokenServices，那么您不必太考虑这一点，因为它实现所有必要的接口，因此它是自动一致的。如果您的资源服务器是一个单独的应用程序，那么您必须确保您匹配授权服务器的功能，并提供知道如何正确解码令牌的ResourceServerTokenServices。与授权服务器一样，您可以经常使用DefaultTokenServices，并且选项大多通过TokenStore（后端存储或本地编码）表示。

（1）在校验request中的token时，使用RemoteTokenServices去调用AuthServer中的/auth/check\_token。

（2）共享数据库，使用Jdbc存储和校验token，避免再去访问AuthServer。

（3）使用JWT签名的方式，资源服务器自己直接进行校验，不借助任何中间媒介。

### oauth client

在客户端获取到token之后，想去调用下游服务API时，为了能将token进行传递，可以使用RestTemplate.然后使用restTemplate进行调用Api。

scopes和authorities的区别：

scopes是client权限，至少授予一个scope的权限，否则报错。

authorities是用户权限。