授权类型的列表

* authorization\_code：授权码类型。
* implicit：隐式授权类型。
* password：资源所有者（即用户）密码类型。
* client\_credentials：客户端凭据（客户端ID以及Key）类型。
* refresh\_token：通过以上授权获得的刷新令牌来获取新的令牌。

可以用 @EnableAuthorizationServer 注解来配置OAuth2.0 授权服务机制，通过使用@Bean注解的几个方法一起来配置这个授权服务。下面咱们介绍几个配置类，这几个配置是由Spring创建的独立的配置对象，它们会被Spring传入AuthorizationServerConfigurer中：

* ClientDetailsServiceConfigurer：用来配置客户端详情服务（ClientDetailsService），客户端详情信息在这里进行初始化，你能够把客户端详情信息写死在这里或者是通过数据库来存储调取详情信息。
* AuthorizationServerSecurityConfigurer：用来配置令牌端点(Token Endpoint)的安全约束.
* AuthorizationServerEndpointsConfigurer：用来配置授权（authorization）以及令牌（token）的访问端点和令牌服务(token services)。

**配置授权类型（Grant Types）：**

授权是使用 AuthorizationEndpoint 这个端点来进行控制的，你能够使用 AuthorizationServerEndpointsConfigurer 这个对象的实例来进行配置(AuthorizationServerConfigurer 的一个回调配置项，见上的概述) ，如果你不进行设置的话，默认是除了资源所有者密码（password）授权类型以外，支持其余所有标准授权类型的（RFC6749），我们来看一下这个配置对象有哪些属性可以设置吧，如下列表：

* authenticationManager：认证管理器，当你选择了资源所有者密码（password）授权类型的时候，请设置这个属性注入一个 AuthenticationManager 对象。
* userDetailsService：如果啊，你设置了这个属性的话，那说明你有一个自己的 UserDetailsService 接口的实现，或者你可以把这个东西设置到全局域上面去（例如 GlobalAuthenticationManagerConfigurer 这个配置对象），当你设置了这个之后，那么 "refresh\_token" 即刷新令牌授权类型模式的流程中就会包含一个检查，用来确保这个账号是否仍然有效，假如说你禁用了这个账户的话。
* authorizationCodeServices：这个属性是用来设置授权码服务的（即 AuthorizationCodeServices 的实例对象），主要用于 "authorization\_code" 授权码类型模式。
* implicitGrantService：这个属性用于设置隐式授权模式，用来管理隐式授权模式的状态。
* tokenGranter：这个属性就很牛B了，当你设置了这个东西（即 TokenGranter 接口实现），那么授权将会交由你来完全掌控，并且会忽略掉上面的这几个属性，这个属性一般是用作拓展用途的，即标准的四种授权模式已经满足不了你的需求的时候，才会考虑使用这个。

**配置客户端详情信息（Client Details)：**

ClientDetailsServiceConfigurer (AuthorizationServerConfigurer 的一个回调配置项，见上的概述) 能够使用内存或者JDBC来实现客户端详情服务（ClientDetailsService），有几个重要的属性如下列表：

* clientId：（必须的）用来标识客户的Id。
* secret：（需要值得信任的客户端）客户端安全码，如果有的话。
* scope：用来限制客户端的访问范围，如果为空（默认）的话，那么客户端拥有全部的访问范围。
* authorizedGrantTypes：此客户端可以使用的授权类型，默认为空。
* authorities：此客户端可以使用的权限（基于Spring Security authorities）。

**管理令牌（Managing Token）：**

AuthorizationServerTokenServices 接口定义了一些操作使得你可以对令牌进行一些必要的管理，在使用这些操作的时候请注意以下几点：

* 当一个令牌被创建了，你必须对其进行保存，这样当一个客户端使用这个令牌对资源服务进行请求的时候才能够引用这个令牌。
* 当一个令牌是有效的时候，它可以被用来加载身份信息，里面包含了这个令牌的相关权限。

当你自己创建 AuthorizationServerTokenServices 这个接口的实现时，你可能需要考虑一下使用 DefaultTokenServices 这个类，里面包含了一些有用实现，你可以使用它来修改令牌的格式和令牌的存储。默认的，当它尝试创建一个令牌的时候，是使用随机值来进行填充的，除了持久化令牌是委托一个 TokenStore 接口来实现以外，这个类几乎帮你做了所有的事情。并且 TokenStore 这个接口有一个默认的实现，它就是 InMemoryTokenStore ，如其命名，所有的令牌是被保存在了内存中。除了使用这个类以外，你还可以使用一些其他的预定义实现，下面有几个版本，它们都实现了TokenStore接口：

* InMemoryTokenStore：这个版本的实现是被默认采用的，它可以完美的工作在单服务器上（即访问并发量压力不大的情况下，并且它在失败的时候不会进行备份），大多数的项目都可以使用这个版本的实现来进行尝试，你可以在开发的时候使用它来进行管理，因为不会被保存到磁盘中，所以更易于调试。
* JdbcTokenStore：这是一个基于JDBC的实现版本，令牌会被保存进关系型数据库。使用这个版本的实现时，你可以在不同的服务器之间共享令牌信息，使用这个版本的时候请注意把"spring-jdbc"这个依赖加入到你的classpath当中。
* JwtTokenStore：这个版本的全称是 JSON Web Token（JWT），它可以把令牌相关的数据进行编码（因此对于后端服务来说，它不需要进行存储，这将是一个重大优势），但是它有一个缺点，那就是撤销一个已经授权令牌将会非常困难，所以它通常用来处理一个生命周期较短的令牌以及撤销刷新令牌（refresh\_token）。另外一个缺点就是这个令牌占用的空间会比较大，如果你加入了比较多用户凭证信息。JwtTokenStore 不会保存任何数据，但是它在转换令牌值以及授权信息方面与 DefaultTokenServices 所扮演的角色是一样的。

**配置授权端点的URL（Endpoint URLs）：**

AuthorizationServerEndpointsConfigurer 这个配置对象(AuthorizationServerConfigurer 的一个回调配置项，见上的概述) 有一个叫做 pathMapping() 的方法用来配置端点URL链接，它有两个参数：

* 第一个参数：String 类型的，这个端点URL的默认链接。
* 第二个参数：String 类型的，你要进行替代的URL链接。

以上的参数都将以 "/" 字符为开始的字符串，框架的默认URL链接如下列表，可以作为这个 pathMapping() 方法的第一个参数：

* /oauth/authorize：授权端点。
* /oauth/token：令牌端点。
* /oauth/confirm\_access：用户确认授权提交端点。
* /oauth/error：授权服务错误信息端点。
* /oauth/check\_token：用于资源服务访问的令牌解析端点。
* /oauth/token\_key：提供公有密匙的端点，如果你使用JWT令牌的话。

Oauth2数据库表说明

<http://andaily.com/spring-oauth-server/db_table_description.html>

oauth\_client\_details：

在项目中,主要操作oauth\_client\_details表的JdbcClientDetailsService.java. 也可以根据实际的需要,去扩展或修改该类的实现.

oauth\_client\_token：

该表用于在客户端系统中存储从服务端获取的token数据, 在[spring-oauth-server](http://git.oschina.net/shengzhao/spring-oauth-server)项目中未使用到. 对oauth\_client\_token表的主要操作在JdbcClientTokenServices.java类中,。

oauth\_access\_token：

在项目中,主要操作oauth\_access\_token表的对象是JdbcTokenStore.java.

oauth\_refresh\_token：

 在项目中,主要操作oauth\_refresh\_token表的对象是JdbcTokenStore.java. (与操作oauth\_access\_token表的对象一样);更多的细节请参考该类. 如果客户端的grant\_type不支持refresh\_token,则不会使用该表.

oauth\_code：

在项目中,主要操作oauth\_code表的对象是JdbcAuthorizationCodeServices.java.. 只有当grant\_type为"authorization\_code"时,该表中才会有数据产生; 其他的grant\_type没有使用该表.

# [Spring Security TokenStore](https://blog.csdn.net/DuShiWoDeCuo/article/details/78929333)

<https://blog.csdn.net/DuShiWoDeCuo/article/details/78929333>

## 1. InMemoryTokenStore

这个是OAuth2默认采用的实现方式。在单服务上可以体现出很好特效（即并发量不大，并且它在失败的时候不会进行备份），大多项目都可以采用此方法。根据名字就知道了，是存储在内存中，毕竟存在内存，而不是磁盘中，调试简易。

## 2. JdbcTokenStore

这个是基于JDBC的实现，令牌（Access Token）会保存到数据库。这个方式，可以在多个服务之间实现令牌共享。

## 3. JwtTokenStore

jwt全称 JSON Web Token。这个实现方式不用管如何进行存储（内存或磁盘），因为它可以把相关信息数据编码存放在令牌里。JwtTokenStore 不会保存任何数据，但是它在转换令牌值以及授权信息方面与 DefaultTokenServices 所扮演的角色是一样的。

## 4. RedisTokenStore

由于TokenStore作用就是对于OAuth2令牌持久化接口，而我们在实际开发中，对于内存的使用是慎之又慎，而对于存储到数据库也是根据项目需求进行调配。因此就想，可不可以用redis来进行存储持久化我们的OAuth2令牌。OAuth2还有那些实现了TokenStore的，找到了一个RedisTokenStore。