

主讲老师：Fox

学习Spring Security OAuth2前提：掌握Spring Security

## 1. OAuth2.0介绍

OAuth (Open Authorization) 是一个关于授权 (authorization) 的开放网络标准，允许用户授权第三方应用访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方移动应用或分享他们数据的所有内容。OAuth在全世界得到广泛应用，目前的版本是2.0版。

OAuth协议：<https://tools.ietf.org/html/rfc6749>

协议特点：

- 简单：不管是OAuth服务提供者还是应用开发者，都很易于理解与使用；
- 安全：没有涉及到用户密钥等信息，更安全更灵活；
- 开放：任何服务提供商都可以实现OAuth，任何软件开发商都可以使用OAuth；

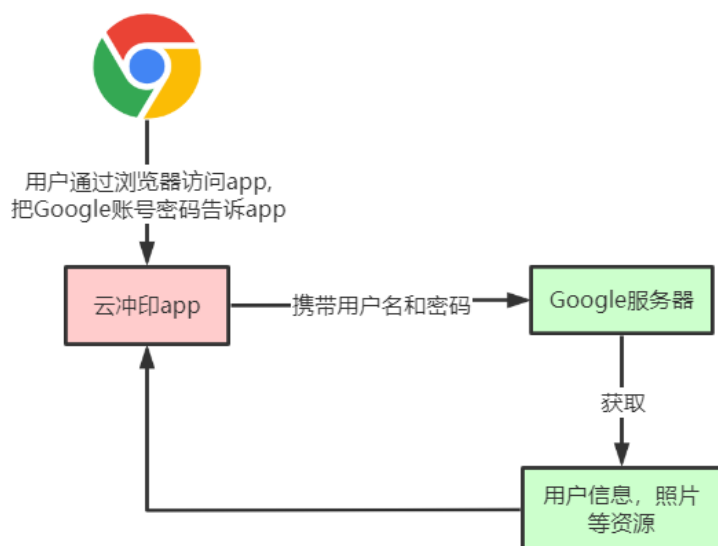
### 1.1 应用场景

- 原生app授权：app登录请求后台接口，为了安全认证，所有请求都带token信息，如果登录验证、请求后台数据。
- 前后端分离单页面应用：前后端分离框架，前端请求后台数据，需要进行oauth2安全认证，比如使用vue、react后者h5开发的app
- 第三方应用授权登录，比如QQ，微博，微信的授权登录。

有一个"云冲印"的网站，可以将用户储存在Google的照片，冲印出来。用户为了使用该服务，必须让"云冲印"读取自己储存在Google上的照片。只有得到用户的授权，Google才会同意"云冲印"读取这些照片。那么，"云冲印"怎样获得用户的授权呢？

传统方法是，用户将自己的Google用户名和密码，告诉"云冲印"，后者就可以读取用户的照片了。这样的做法有以下几个严重的缺点：

- "云冲印"为了后续的服务，会保存用户的密码，这样很不安全。
- Google不得不部署密码登录，而我们知道，单纯的密码登录并不安全。
- "云冲印"拥有了获取用户储存在Google所有资料的权力，用户没法限制"云冲印"获得授权的范围和有效期。
- 用户只有修改密码，才能收回赋予"云冲印"的权力。但是这样做，会使得其他所有获得用户授权的第三方应用程序全部失效。
- 只要有一个第三方应用程序被破解，就会导致用户密码泄漏，以及所有被密码保护的数据泄漏。



生活中常见的oauth2场景，京东商城（<https://www.jd.com/>）接入微信开放平台，可以通过微信登录。



登录流程分析：

<https://www.processon.com/view/link/60a32e7a079129157118740f>

## 1.2 基本概念

- (1) **Third-party application**: 第三方应用程序，又称"客户端" (client)，即例子中的"云冲印"。
- (2) **HTTP service**: HTTP服务提供商，简称"服务提供商"，即例子中的Google。
- (3) **Resource Owner**: 资源所有者，又称"用户" (user)。
- (4) **User Agent**: 用户代理，比如浏览器。
- (5) **Authorization server**: 授权服务器，即服务提供商专门用来处理认证授权的服务器。
- (6) **Resource server**: 资源服务器，即服务提供商存放用户生成的资源的服务器。它与授权服务器，可以是同一台服务器，也可以是不同的服务器。

OAuth的作用就是让"客户端"安全可控地获取"用户"的授权，与"服务提供商"进行交互。

## 1.3 优缺点

优点：

- 更安全，客户端不接触用户密码，服务器端更易集中保护
- 广泛传播并被持续采用
- 短寿命和封装的token
- 资源服务器和授权服务器解耦
- 集中式授权，简化客户端
- HTTP/JSON友好，易于请求和传递token
- 考虑多种客户端架构场景
- 客户可以具有不同的信任级别

缺点：

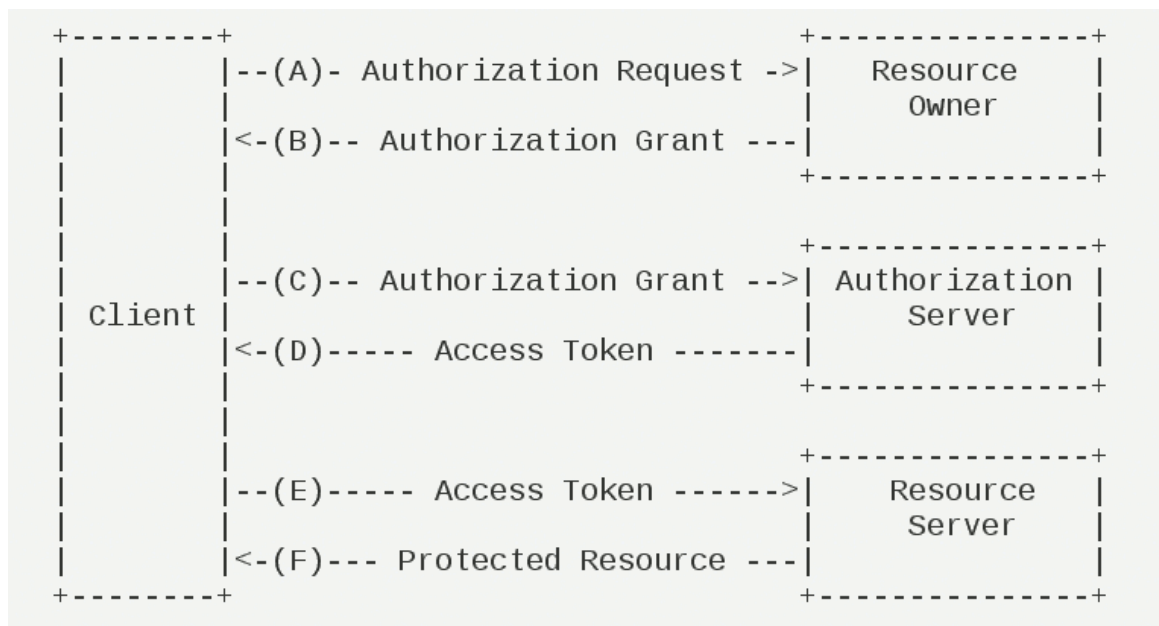
- 协议框架太宽泛，造成各种实现的兼容性和互操作性差
- 不是一个认证协议，本身并不能告诉你任何用户信息。

## 2. OAuth2的设计思路

OAuth在"客户端"与"服务提供商"之间，设置了一个授权层（authorization layer）。"客户端"不能直接登录"服务提供商"，只能登录授权层，以此将用户与客户端区分开来。"客户端"登录授权层所用的令牌

（token），与用户的密码不同。用户可以在登录的时候，指定授权层令牌的权限范围和有效期，"客户端"登录授权层以后，"服务提供商"根据令牌的权限范围和有效期，向"客户端"开放用户储存的资料。

OAuth 2.0的运行流程如下图，摘自RFC 6749：



- 用户打开客户端以后，客户端要求用户给予授权。
- 用户同意给予客户端授权。
- 客户端使用上一步获得的授权，向授权服务器申请令牌。
- 授权服务器对客户端进行认证以后，确认无误，同意发放令牌。
- 客户端使用令牌，向资源服务器申请获取资源。
- 资源服务器确认令牌无误，同意向客户端开放资源。

令牌（token）与密码（password）的作用是一样的，都可以进入系统，但是有三点差异。

- （1）令牌是短期的，到期会自动失效，用户自己无法修改。密码一般长期有效，用户不修改，就不会发生变化。
- （2）令牌可以被数据所有者撤销，会立即失效。密码一般不允许被他人撤销。
- （3）令牌有权限范围（scope）。对于网络服务来说，只读令牌就比读写令牌更安全。密码一般是完整权限。

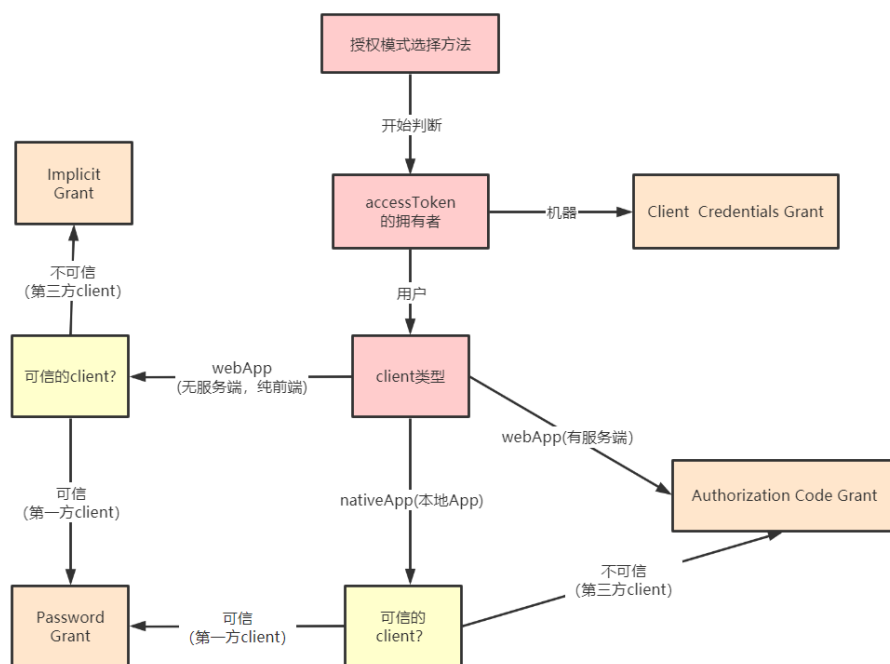
上面这些设计，保证了令牌既可以让第三方应用获得权限，同时又随时可控，不会危及系统安全。这就是 OAuth 2.0 的优点。

## 2.1 客户端授权模式

客户端必须得到用户的授权（authorization grant），才能获得令牌（access token）。OAuth 2.0 对于如何颁发令牌的细节，规定得非常详细。具体来说，一共分成四种授权类型（authorization grant），即四种颁发令牌的方式，适用于不同的互联网场景。

- 授权码模式（authorization code）
- 密码模式（resource owner password credentials）
- 简化(隐式)模式（implicit）
- 客户端模式（client credentials）

不管哪一种授权方式，第三方应用申请令牌之前，都必须先到系统备案，说明自己的身份，然后会拿到两个身份识别码：客户端 ID（client ID）和客户端密钥（client secret）。这是为了防止令牌被滥用，没有备案过的第三方应用，是不会拿到令牌的。

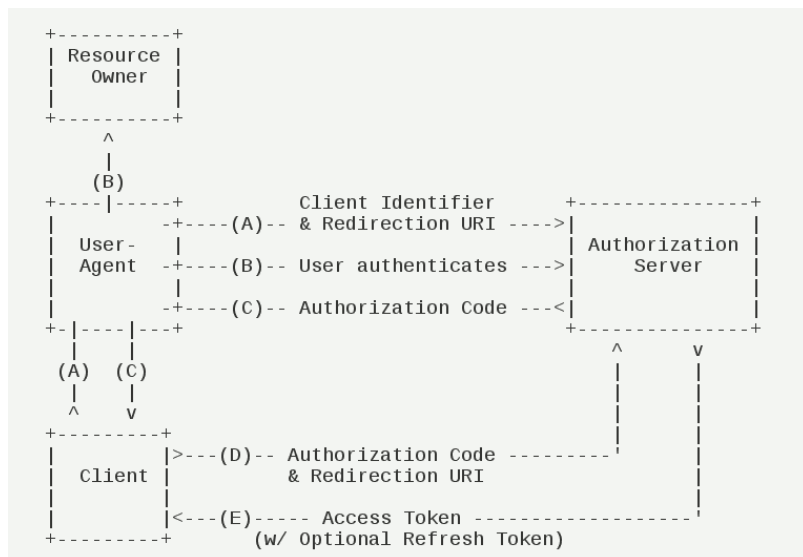


### 授权码模式

**授权码（authorization code）方式，指的是第三方应用先申请一个授权码，然后再用该码获取令牌。**

这种方式是最常用的流程，安全性也最高，它适用于那些有后端的 Web 应用。授权码通过前端传送，令牌则是储存在后端，而且所有与资源服务器的通信都在后端完成。这样的前后端分离，可以避免令牌泄漏。

**适用场景：**目前市面上主流的第三方验证都是采用这种模式



它的步骤如下：

- (A) 用户访问客户端，后者将前者导向授权服务器。
- (B) 用户选择是否给予客户端授权。
- (C) 假设用户给予授权，授权服务器将用户导向客户端事先指定的"重定向URI" (redirection URI)，同时附上一个授权码。
- (D) 客户端收到授权码，附上早先的"重定向URI"，向授权服务器申请令牌。这一步是在客户端的后台的服务器上完成的，对用户不可见。
- (E) 授权服务器核对了授权码和重定向URI，确认无误后，向客户端发送访问令牌 (access token) 和更新令牌 (refresh token)。

1. A网站提供一个链接，用户点击后就会跳转到 B 网站，授权用户数据给 A 网站使用。下面就是 A 网站跳转 B 网站的一个示意链接。

```

1 > https://b.com/oauth/authorize?
2 > response_type=code& #要求返回授权码 (code)
3 > client_id=CLIENT_ID& #让 B 知道是谁在请求
4 > redirect_uri=CALLBACK_URL& #B 接受或拒绝请求后的跳转网址
5 > scope=read # 要求的授权范围 (这里是只读)
6 >
  
```

客户端申请授权的URI，包含以下参数：

- response\_type：表示授权类型，必选项，此处的值固定为"code"
- client\_id：表示客户端的ID，必选项
- redirect\_uri：表示重定向URI，可选项
- scope：表示申请的权限范围，可选项
- state：表示客户端的当前状态，可以指定任意值，授权服务器会原封不动地返回这个值。

2. 用户跳转后，B 网站会要求用户登录，然后询问是否同意给予 A 网站授权。用户表示同意，这时 B 网站就会跳回redirect\_uri参数指定的网址。跳转时，会传回一个授权码，就像下面这样。

```

1 > https://a.com/callback?code=AUTHORIZATION_CODE #code参数就是授权码
2 >
  
```

3. A 网站拿到授权码以后，就可以在后端，向 B 网站请求令牌。用户不可见，服务端行为

```

1 > https://b.com/oauth/token?
2 > client_id=CLIENT_ID&
3 > client_secret=CLIENT_SECRET& # client_id和client_secret用来让 B 确认 A 的身份,client_secret参数是保密的，因此只能在后端发请求
4 > grant_type=authorization_code& # 采用的授权方式是授权码
5 > code=AUTHORIZATION_CODE& # 上一步拿到的授权码
6 > redirect_uri=CALLBACK_URL # 令牌颁发后的回调网址
  
```

4. B 网站收到请求以后，就会颁发令牌。具体做法是向`redirect_uri`指定的网址，发送一段 JSON 数据。

```

1 > {
2 >   "access_token": "ACCESS_TOKEN", # 令牌
3 >   "token_type": "bearer",
4 >   "expires_in": 2592000,
5 >   "refresh_token": "REFRESH_TOKEN",
6 >   "scope": "read",
7 >   "uid": 100101,
8 >   "info": {...}
9 > }

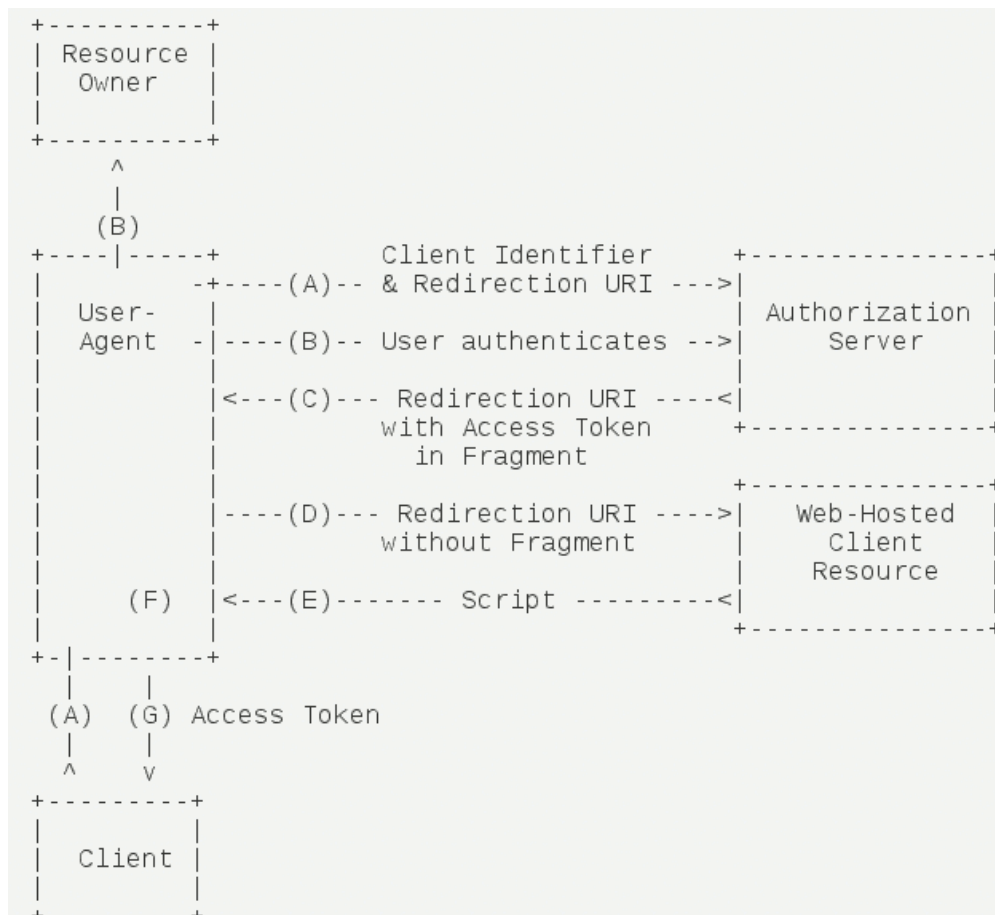
```

### 简化(隐式)模式

有些 Web 应用是纯前端应用，没有后端。这时就不能用上面的方式了，必须将令牌储存在前端。**RFC 6749**就规定了第二种方式，允许直接向前端颁发令牌，这种方式没有授权码这个中间步骤，所以称为（授权码）“**隐藏式**”（implicit）

简化模式不通过第三方应用程序的服务器，直接在浏览器中向授权服务器申请令牌，跳过了“授权码”这个步骤，所有步骤在浏览器中完成，令牌对访问者是可见的，且客户端不需要认证。

这种方式把令牌直接传给前端，是很不安全的。因此，只能用于一些安全要求不高的场景，并且令牌的有效期限必须非常短，通常就是会话期间（session）有效，浏览器关掉，令牌就失效了。



它的步骤如下：

- (A) 客户端将用户导向授权服务器。
- (B) 用户决定是否给予客户端授权。
- (C) 假设用户给予授权，授权服务器将用户导向客户端指定的“重定向URI”，并在URI的Hash部分包含了访问令牌。
- (D) 浏览器向资源服务器发出请求，其中不包括上一步收到的Hash值。
- (E) 资源服务器返回一个网页，其中包含的代码可以获取Hash值中的令牌。
- (F) 浏览器执行上一步获得的脚本，提取出令牌。

(G) 浏览器将令牌发给客户端。

1. A 网站提供一个链接，要求用户跳转到 B 网站，授权用户数据给 A 网站使用。

```
1 > https://b.com/oauth/authorize?
2 > response_type=token& # response_type参数为token，表示要求直接返回令牌
3 > client_id=CLIENT_ID&
4 > redirect_uri=CALLBACK_URL&
5 > scope=read
6 >
```

2. 用户跳转到 B 网站，登录后同意给予 A 网站授权。这时，B 网站就会跳回 `redirect_uri` 参数指定的跳转网址，并且把令牌作为 URL 参数，传给 A 网站。

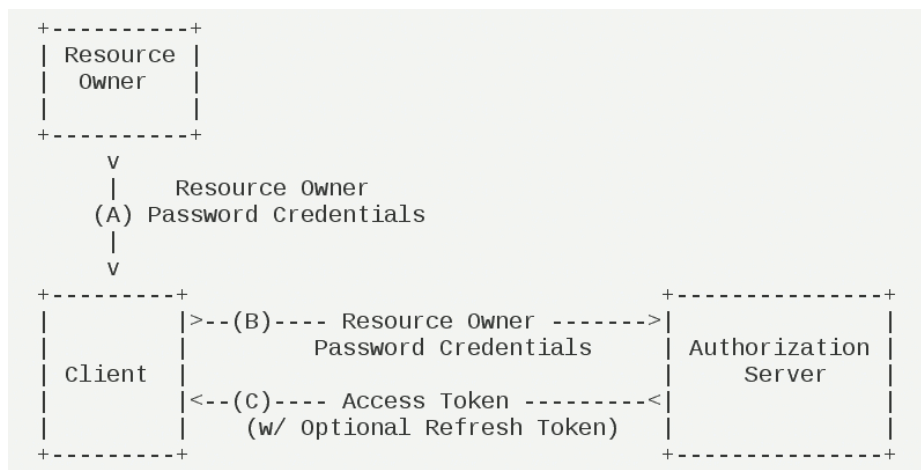
```
1 > https://a.com/callback#token=ACCESS_TOKEN #token参数就是令牌，A 网站直接在前端拿到令牌。
2 >
```

### 密码模式

如果你高度信任某个应用，RFC 6749 也允许用户把用户名和密码，直接告诉该应用。该应用就使用你的密码，申请令牌，这种方式称为“密码式”（password）。

在这种模式中，用户必须把自己的密码给客户端，但是客户端不得储存密码。这通常用在用户对客户端高度信任的情况下，比如客户端是操作系统的一部分，或者由一个著名公司出品。而授权服务器只有在其他授权模式无法执行的情况下，才能考虑使用这种模式。

适用场景：自家公司搭建的授权服务器



它的步骤如下：

- (A) 用户向客户端提供用户名和密码。
- (B) 客户端将用户名和密码发给授权服务器，向后者请求令牌。
- (C) 授权服务器确认无误后，向客户端提供访问令牌。

1. A 网站要求用户提供 B 网站的用户名和密码，拿到以后，A 就直接向 B 请求令牌。整个过程中，客户端不得保存用户的密码。

```
1 > https://oauth.b.com/token?
2 > grant_type=password& # 授权方式是"密码式"
3 > username=USERNAME&
4 > password=PASSWORD&
5 > client_id=CLIENT_ID
6 >
```

2. B 网站验证身份通过后，直接给出令牌。注意，这时不需要跳转，而是把令牌放在 JSON 数据里面，作为 HTTP 回应，A 因此拿到令牌。



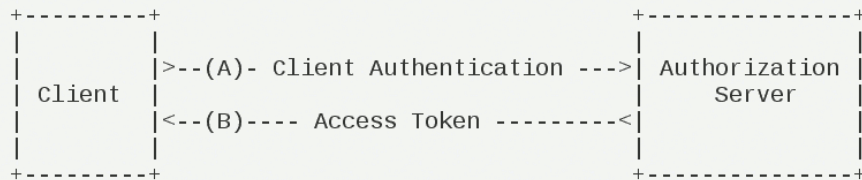
```
localhost:8080/oauth/token?username=fox&password=123456&grant_type=password&scope=select&client_id=client_2&client_secret=123456

{
  access_token: "373059e1-b140-4d5a-a7d4-5b6868dc5e3a",
  token_type: "bearer",
  refresh_token: "e7f1ea37-8175-4d57-beb5-931af185aa9d",
  expires_in: 38389,
  scope: "select"
}
```

## 客户端模式

客户端模式 (Client Credentials Grant) 指客户端以自己的名义, 而不是以用户的名义, 向"服务提供商"进行授权。

适用于没有前端的命令行应用, 即在命令行下请求令牌。一般用来提供给我们完全信任的服务器端服务。



它的步骤如下:

- (A) 客户端向授权服务器进行身份认证, 并要求一个访问令牌。
- (B) 授权服务器确认无误后, 向客户端提供访问令牌。

1. A 应用在命令行向 B 发出请求。

```
1 > https://oauth.b.com/token?
2 > grant_type=client_credentials&
3 > client_id=CLIENT_ID&
4 > client_secret=CLIENT_SECRET
```

2. B 网站验证通过以后, 直接返回令牌。

```
localhost:8080/oauth/token?grant_type=client_credentials&scope=select&client_id=client_1&client_secret=123456

{
  access_token: "a4e4ccb0-9a51-479a-a86c-376410fd0c00",
  token_type: "bearer",
  expires_in: 38086,
  scope: "select"
}
```

## 2.2 令牌的使用

A 网站拿到令牌以后, 就可以向 B 网站的 API 请求数据了。

此时, 每个发到 API 的请求, 都必须带有令牌。具体做法是在请求的头信息, 加上一个Authorization字段, 令牌就放在这个字段里面。

```
1 > curl -H "Authorization: Bearer ACCESS_TOKEN" \
2 > "https://api.b.com"
3 >
```

也可以通过添加请求参数access\_token请求数据。

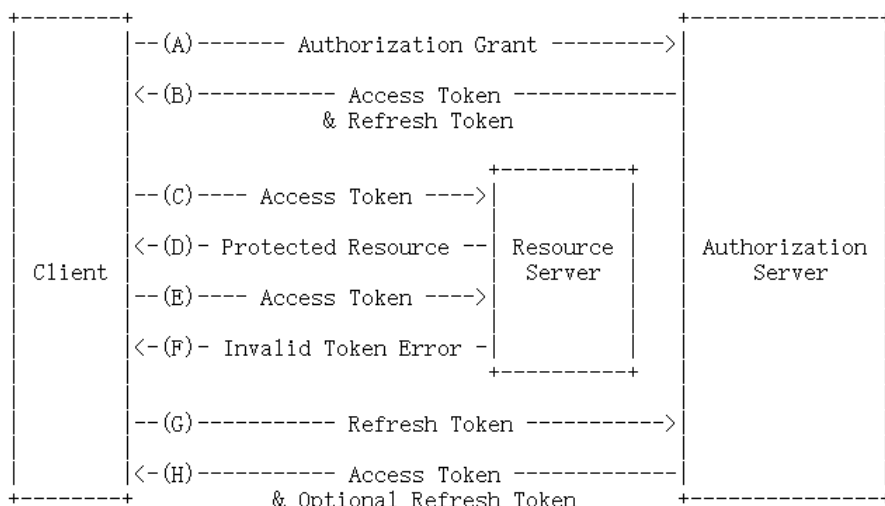
```
localhost:8080/user/getCurrentUser?access_token=124977da-3e18-475b-ad32-af5cf1836625

{
  password: null,
  username: "fox",
  - authorities: [
    - {
      authority: "admin"
    }
  ],
  accountNonExpired: true,
  accountNonLocked: true,
  credentialsNonExpired: true,
  enabled: true
}
```

## 2.3 更新令牌



令牌的有效期到了，如果让用户重新走一遍上面的流程，再申请一个新的令牌，很可能体验不好，而且也没有必要。OAuth 2.0 允许用户自动更新令牌。



具体方法是，B 网站颁发令牌的时候，一次性颁发两个令牌，一个用于获取数据，另一个用于获取新的令牌（refresh token 字段）。令牌到期前，用户使用 refresh token 发一个请求，去更新令牌。

```
1 > https://b.com/oauth/token?
2 > grant_type=refresh_token& # grant_type参数为refresh_token表示要求更新令牌
3 > client_id=CLIENT_ID&
4 > client_secret=CLIENT_SECRET&
5 > refresh_token=REFRESH_TOKEN # 用于更新令牌的令牌
6 >
```

```
← → ↻ local:8080/oauth/token?grant_type=refresh_token&client_id=client&client_secret=123123&refresh_token=dc03bdc2-ca3b-4690-9265-d31a21896d02
{
  access_token: "f9fc9d38-4987-495a-b8d2-710bf63d04f0",
  token_type: "bearer",
  refresh_token: "9b5fea4d-6e05-46c4-8a88-affbd53b072a",
  expires_in: 3599,
  scope: "all"
}
```

### 3. Spring Security OAuth2快速开始

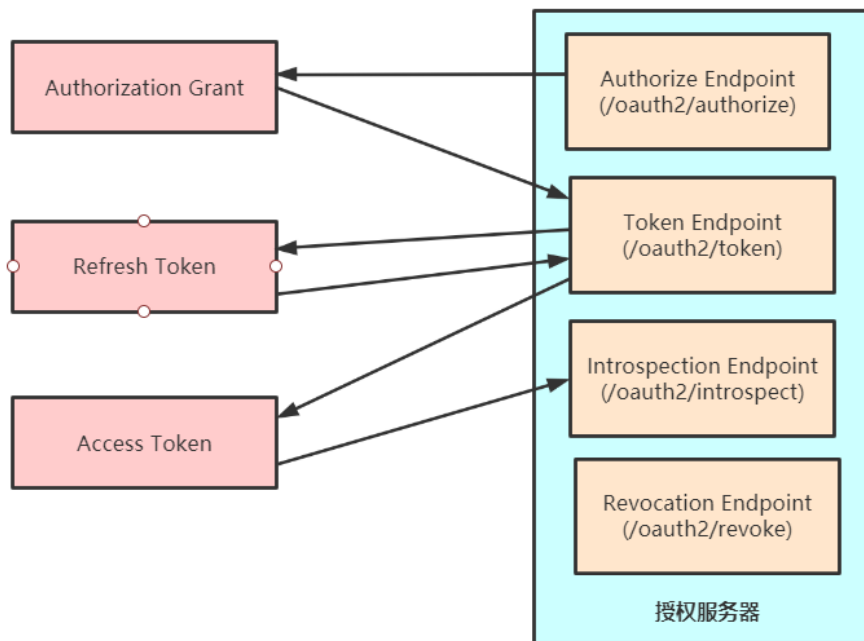
Spring Security是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的**安全框架**。Spring Security 主要实现了**Authentication**（认证，解决who are you?）和 **Access Control**（访问控制，也就是what are you allowed to do?，也称为**Authorization**）。Spring Security在架构上将认证与授权分离，并提供了扩展点。

**认证 (Authentication)**：用户认证就是判断一个用户的身份是否合法的过程，用户去访问系统资源时系统要求验证用户的身份信息，身份合法方可继续访问，不合法则拒绝访问。常见的用户身份认证方式有：用户名密码登录，二维码登录，手机短信登录，指纹认证等方式。

**授权 (Authorization)**：授权是用户认证通过根据用户的权限来控制用户访问资源的过程，拥有资源的访问权限则正常访问，没有权限则拒绝访问。

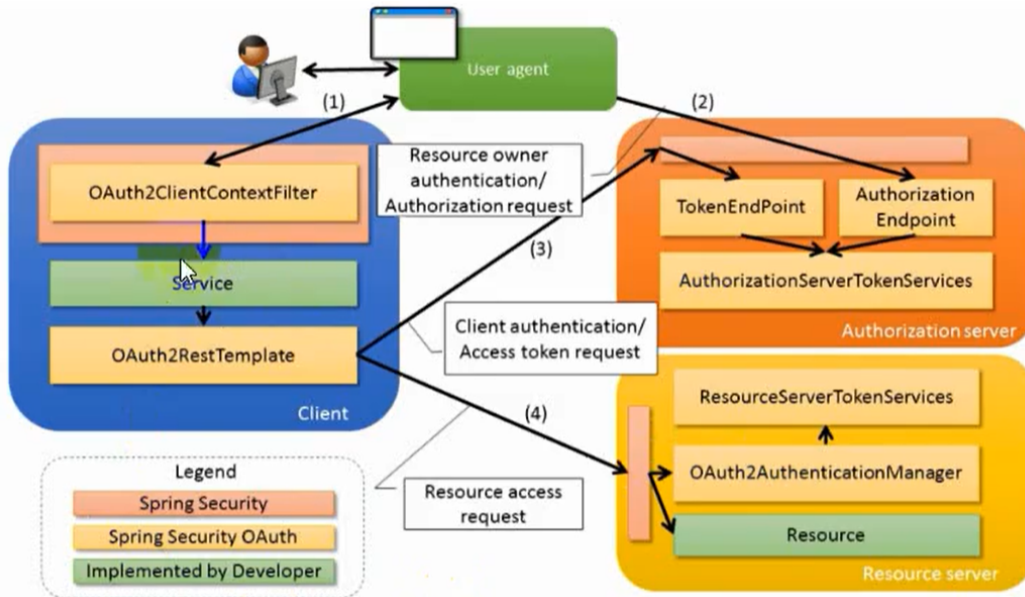
将OAuth2和Spring Security集成，就可以得到一套完整的安全解决方案。我们可以通过Spring Security OAuth2构建一个授权服务器来验证用户身份以提供access\_token，并使用这个access\_token来从资源服务器请求数据。

#### 3.1 授权服务器



- Authorize Endpoint：授权端点，进行授权
- Token Endpoint：令牌端点，经过授权拿到对应的Token
- Introspection Endpoint：校验端点，校验Token的合法性
- Revocation Endpoint：撤销端点，撤销授权

### 3.2 整体架构



流程：

1. 用户访问,此时没有Token。Oauth2RestTemplate会报错，这个报错信息会被Oauth2ClientContextFilter捕获并重定向到授权服务器。
2. 授权服务器通过Authorization Endpoint进行授权，并通过AuthorizationServerTokenServices生成授权码并返回给客户端。
3. 客户端拿到授权码去授权服务器通过Token Endpoint调用AuthorizationServerTokenServices生成Token并返回给客户端

4. 客户端拿到Token去资源服务器访问资源，一般会通过Oauth2AuthenticationManager调用ResourceServerTokenServices进行校验。校验通过可以获取资源。

### 3.3 授权码模式

#### 引入依赖

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-security</artifactId>
4 </dependency>
5
6 <dependency>
7   <groupId>org.springframework.security.oauth</groupId>
8   <artifactId>spring-security-oauth2</artifactId>
9   <version>2.3.4.RELEASE</version>
10 </dependency>
```

#### 或者 引入spring cloud oauth2依赖

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-oauth2</artifactId>
4 </dependency>
5
6 <!-- spring cloud -->
7 <dependencyManagement>
8   <dependencies>
9     <dependency>
10      <groupId>org.springframework.cloud</groupId>
11      <artifactId>spring-cloud-dependencies</artifactId>
12      <version>Hoxton.SR8</version>
13      <type>pom</type>
14      <scope>import</scope>
15    </dependency>
16  </dependencies>
17 </dependencyManagement>
```

#### 配置 spring security

```
1 @Configuration
2 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
3
4   @Bean
5   public PasswordEncoder passwordEncoder(){
6     return new BCryptPasswordEncoder();
7   }
8
9
10  @Override
11  protected void configure(HttpSecurity http) throws Exception {
12    http.formLogin().permitAll()
13      .and().authorizeRequests()
14      .antMatchers("/oauth/**").permitAll()
15      .anyRequest().authenticated()
16      .and().logout().permitAll()
17      .and().csrf().disable();
18  }
```

```

19 }
20
21 @Service
22 public class UserService implements UserDetailsService {
23
24     @Autowired
25     private PasswordEncoder passwordEncoder;
26
27     @Override
28     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
29         String password = passwordEncoder.encode("123456");
30         return new User("fox",password, AuthorityUtils.commaSeparatedStringToAuthorityList("admin"));
31     }
32 }
33
34 @RestController
35 @RequestMapping("/user")
36 public class UserController {
37
38     @RequestMapping("/getCurrentUser")
39     public Object getCurrentUser(Authentication authentication) {
40         return authentication.getPrincipal();
41     }
42 }

```

## 配置授权服务器

```

1 @Configuration
2 @EnableAuthorizationServer
3 public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
4
5     @Autowired
6     private PasswordEncoder passwordEncoder;
7
8     @Override
9     public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
10         clients.inMemory()
11             //配置client_id
12             .withClient("client")
13             //配置client-secret
14             .secret(passwordEncoder.encode("123123"))
15             //配置访问token的有效期
16             .accessTokenValiditySeconds(3600)
17             //配置刷新token的有效期
18             .refreshTokenValiditySeconds(864000)
19             //配置redirect_uri，用于授权成功后跳转
20             .redirectUri("http://www.baidu.com")
21             //配置申请的权限范围
22             .scopes("all")
23             //配置grant_type，表示授权类型
24             .authorizedGrantTypes("authorization_code");
25     }
26 }

```

## 配置资源服务器

```

1 @Configuration
2 @EnableResourceServer

```

```

3 public class ResourceServiceConfig extends ResourceServerConfigurerAdapter {
4
5     @Override
6     public void configure(HttpSecurity http) throws Exception {
7         http.authorizeRequests()
8             .anyRequest().authenticated()
9             .and().requestMatchers().antMatchers("/user/**");
10
11     }
12 }

```

测试

获取授权码

[http://localhost:8080/oauth/authorize?response\\_type=code&client\\_id=client](http://localhost:8080/oauth/authorize?response_type=code&client_id=client)

或者

[http://localhost:8080/oauth/authorize?](http://localhost:8080/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://www.baidu.com&scope=all)

[response\\_type=code&client\\_id=client&redirect\\_uri=http://www.baidu.com&scope=all](http://localhost:8080/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://www.baidu.com&scope=all)

登录之后进入

# OAuth Approval

Do you authorize "client" to access your protected resources?

- scope.all: ☐ Approve ☒ Deny

Authorize

选择Approve, 点击授权获取授权码

[baidu.com/?code=FvIDC2](http://baidu.com/?code=FvIDC2) 授权码

获取令牌

根据授权码通过post请求获取

http://localhost:8080/oauth/token

POST http://localhost:8080/oauth/token

Params Authorization Headers Body Pre-request Script Tests

TYPE

Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure we recommend using variables. [Learn more about variables](#)

Username client

Password 123123

☒ Show Password

http://localhost:8080/oauth/token

POST http://localhost:8080/oauth/token

Params Authorization Headers (2) **Body** Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	authorization_code
<input checked="" type="checkbox"/> code	FvIDC2
<input type="checkbox"/> client_id	client
<input type="checkbox"/> client_secret	123123
<input checked="" type="checkbox"/> redirect_uri	http://www.baidu.com
<input type="checkbox"/> scope	all
Key	Value

Body Cookies (1) Headers (10) Test Results

Pretty Raw Preview JSON

```

1 {
2   "access_token": "124977da-3e18-475b-ad32-af5cf1836625",
3   "token_type": "bearer",
4   "expires_in": 3492,
5   "scope": "all"
6 }

```

或者

GET http://localhost:8080/user/getCurrentUser

Params Authorization Headers (1) Body Pre-request Script Tests

KEY	VALUE
<input checked="" type="checkbox"/> Authorization	bearer 91d32ed5-8bf4-42e9-9a92-e682109104a2
Key	Value

Body Cookies (1) Headers (11) Test Results

Pretty Raw Preview JSON

```

1 {
2   "password": null,
3   "username": "fox",
4   "authorities": [
5     {
6       "authority": "admin"
7     }
8   ],
9   "accountNonExpired": true,
10  "accountNonLocked": true,
11  "credentialsNonExpired": true,
12  "enabled": true
13 }

```

或者

localhost:8080/user/getCurrentUser?access\_token=124977da-3e18-475b-ad32-af5cf1836625

```

{
  password: null,
  username: "fox",
  authorities: [
    {
      authority: "admin"
    }
  ],
  accountNonExpired: true,
  accountNonLocked: true,
  credentialsNonExpired: true,
  enabled: true
}

```

### 3.4 简化模式

authorizedGrantType添加implicit

```

/**
 * 配置grant_type, 表示授权类型
 * authorization_code: 授权码模式
 * implicit: 简化模式
 * password: 密码模式
 * client_credentials: 客户端模式
 * refresh_token: 更新令牌
 */
.authorizedGrantTypes("authorization_code", "implicit", "password",
"client_credentials", "refresh_token");

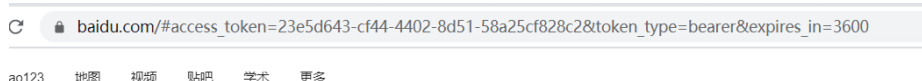
```

测试

<http://localhost:8080/oauth/authorize?>

[client\\_id=client&response\\_type=token&scope=all&redirect\\_uri=http://www.baidu.com](http://localhost:8080/oauth/authorize?client_id=client&response_type=token&scope=all&redirect_uri=http://www.baidu.com)

登录之后进入授权页面，确定授权后浏览器会重定向到指定路径，并以Hash的形式存放在重定向uri的fragment中：

 baidu.com/#access\_token=23e5d643-cf44-4402-8d51-58a25cf828c2&token\_type=bearer&expires\_in=3600

ao123 地图 视频 贴吧 学术 更多

## 3.5 密码模式

修改WebSecurityConfig，增加AuthenticationManager

```

1 @Configuration
2 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
3
4     @Bean
5     public PasswordEncoder passwordEncoder(){
6         return new BCryptPasswordEncoder();
7     }
8
9
10    @Override
11    protected void configure(HttpSecurity http) throws Exception {
12        http.formLogin().permitAll()
13            .and().authorizeRequests()
14            .antMatchers("/oauth/**").permitAll()
15            .anyRequest().authenticated()
16            .and().logout().permitAll()
17            .and().csrf().disable();
18    }
19
20
21    @Bean
22    @Override
23    public AuthenticationManager authenticationManagerBean() throws Exception {
24        return super.authenticationManagerBean();
25    }
26 }

```

修改AuthorizationServerConfig配置

```

1 @Configuration
2 @EnableAuthorizationServer
3 public class AuthorizationServerConfig2 extends AuthorizationServerConfigurerAdapter {
4
5     @Autowired
6     private PasswordEncoder passwordEncoder;
7
8     @Autowired

```



```

9  private AuthenticationManager authenticationManagerBean;
10
11
12  @Override
13  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
14      endpoints.authenticationManager(authenticationManagerBean) //使用密码模式需要配置
15      .allowedTokenEndpointRequestMethods(HttpMethod.GET, HttpMethod.POST); //支持GET, POST请求
16  }
17
18  @Override
19  public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
20      //允许表单认证
21      security.allowFormAuthenticationForClients();
22  }
23
24  @Override
25  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
26
27      /**
28      *授权码模式
29      *http://localhost:8080/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://www.baidu.com&scope=all
30      *http://localhost:8080/oauth/authorize?response_type=code&client_id=client
31      *
32      * password模式
33      * http://localhost:8080/oauth/token?
34      username=fox&password=123456&grant_type=password&client_id=client&client_secret=123123&scope=all
35      *
36      * 客户端模式
37      * http://localhost:8080/oauth/token?grant_type=client_credentials&scope=all&client_id=client&client_secret=123123
38      */
39      clients.inMemory()
40      //配置client_id
41      .withClient("client")
42      //配置client-secret
43      .secret(passwordEncoder.encode("123123"))
44      //配置访问token的有效期
45      .accessTokenValiditySeconds(3600)
46      //配置刷新token的有效期
47      .refreshTokenValiditySeconds(86400)
48      //配置redirect_uri, 用于授权成功后跳转
49      .redirectUri("http://www.baidu.com")
50      //配置申请的权限范围
51      .scopes("all")
52      /**
53      * 配置grant_type, 表示授权类型
54      * authorization_code: 授权码
55      * password: 密码
56      * client_credentials: 客户端
57      */
58      .authorizedGrantTypes("authorization_code", "password", "client_credentials");
59  }

```

## 获取令牌

通过浏览器测试，需要配置支持get请求和表单验证

<http://localhost:8080/oauth/token?>

[username=fox&password=123456&grant\\_type=password&client\\_id=client&client\\_secret=123123&scope=all](http://localhost:8080/oauth/token?username=fox&password=123456&grant_type=password&client_id=client&client_secret=123123&scope=all)

```
← → ↻ 🌐 localhost:8080/oauth/token?username=fox&password=123456&grant_type=password&client_id=client&client_secret=123123&scope=all

{
  access_token: "3f653663-26bb-4e72-9e3d-de1dbab9310e",
  token_type: "bearer",
  expires_in: 3599,
  scope: "all"
}
```

## 通过Postman测试

POST http://localhost:8080/oauth/token

Params Authorization Headers (2) Body Pre-request Script Tests

TYPE  
Basic Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Username client  
Password .....  
☐ Show Password

Heads up! These parameters hold sensitive data. To keep this data secure while variables. [Learn more about variables](#)

POST http://localhost:8080/oauth/token

Params Authorization Headers (2) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary

KEY	VALUE
<input checked="" type="checkbox"/> grant_type	password
<input checked="" type="checkbox"/> username	fox
<input checked="" type="checkbox"/> password	123456
<input type="checkbox"/> scope	all
Key	Value

Body Cookies (1) Headers (10) Test Results

Pretty Raw Preview JSON

```
1 {
2   "access_token": "f16350fd-3661-4c00-832b-278a1c378518",
3   "token_type": "bearer",
4   "expires_in": 3371,
5   "scope": "all"
6 }
```

## 访问资源

```
← → ↻ 🌐 localhost:8080/user/getCurrentUser?access_token=f16350fd-3661-4c00-832b-278a1c378518

{
  password: null,
  username: "fox",
  - authorities: [
    - {
      authority: "admin"
    }
  ],
  accountNonExpired: true,
  accountNonLocked: true,
  credentialsNonExpired: true,
  enabled: true
}
```

## 3.6 客户端模式

### 获取令牌

```
{
  access_token: "c90737cb-56d3-4fa6-9eeb-429b7b968eda",
  token_type: "bearer",
  expires_in: 3428,
  scope: "all"
}
```

### 3.7 更新令牌

使用oauth2时，如果令牌失效了，可以使用刷新令牌通过refresh\_token的授权模式再次获取access\_token。只需修改认证服务器的配置，添加refresh\_token的授权模式即可。

修改授权服务器配置，增加refresh\_token配置

```
1 @Autowired
2 private UserService userService;
3 @Override
4 public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
5     endpoints.authenticationManager(authenticationManagerBean) //使用密码模式需要配置
6     // .tokenStore(tokenStore) //指定token存储到redis
7     .reuseRefreshTokens(false) //refresh_token是否重复使用
8     .userDetailsService(userService) //刷新令牌授权包含对用户信息的检查
9     .allowedTokenEndpointRequestMethods(HttpMethod.GET, HttpMethod.POST); //支持GET, POST请求
10 }
11
12 @Override
13 public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
14
15     /**
16     * 授权码模式
17     * http://localhost:8080/oauth/authorize?response_type=code&client_id=client&redirect_uri=http://www.baidu.com&scope=all
18     * http://localhost:8080/oauth/authorize?response_type=code&client_id=client
19     *
20     * password模式
21     * http://localhost:8080/oauth/token?username=fox&password=123456&grant_type=password&client_id=client&client_secret=123123&scope=all
22     *
23     * 客户端模式
24     * http://localhost:8080/oauth/token?grant_type=client_credentials&scope=all&client_id=client&client_secret=123123
25     */
26     clients.inMemory()
27     //配置client_id
28     .withClient("client")
29     //配置client-secret
30     .secret(passwordEncoder.encode("123123"))
31     //配置访问token的有效期
32     .accessTokenValiditySeconds(3600)
33     //配置刷新token的有效期
34     .refreshTokenValiditySeconds(86400)
35     //配置redirect_uri，用于授权成功后跳转
36     .redirectUri("http://www.baidu.com")
37     //配置申请的权限范围
38     .scopes("all")
39     /**
40     * 配置grant_type，表示授权类型
41     * authorization_code: 授权码
42     * password: 密码
```

```

43 * client_credentials: 客户端
44 * refresh_token: 更新令牌
45 */
46 .authorizedGrantTypes("authorization_code", "password", "client_credentials", "refresh_token");
47 }

```

## 通过密码模式测试

localhost:8080/oauth/token?username=fox&password=123456&grant\_type=password&client\_id=client&client\_secret=123123&scope=all

```

{
  access_token: "298ee823-f4d1-49d9-8f50-9e47826025fc",
  token_type: "bearer",
  refresh_token: "f9ca4556-a04c-41c3-a1a5-c37cb750af97",
  expires_in: 3599,
  scope: "all"
}

```

[http://localhost:8080/oauth/token?](http://localhost:8080/oauth/token?grant_type=refresh_token&client_id=client&client_secret=123123&refresh_token=dc03bdc2-ca3b-4690-9265-d31a21896d02)

[grant\\_type=refresh\\_token&client\\_id=client&client\\_secret=123123&refresh\\_token=dc03bdc2-ca3b-4690-9265-d31a21896d02](http://localhost:8080/oauth/token?grant_type=refresh_token&client_id=client&client_secret=123123&refresh_token=dc03bdc2-ca3b-4690-9265-d31a21896d02)

localhost:8080/oauth/token?grant\_type=refresh\_token&client\_id=client&client\_secret=123123&refresh\_token=dc03bdc2-ca3b-4690-9265-d31a21896d02

```

{
  access_token: "f9fc9d38-4987-495a-b8d2-710bf63d04f0",
  token_type: "bearer",
  refresh_token: "9b5fea4d-6e05-46c4-8a88-affbd53b072a",
  expires_in: 3599,
  scope: "all"
}

```

## 3.8 基于redis存储Token

### 引入依赖

```

1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-data-redis</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>org.apache.commons</groupId>
7   <artifactId>commons-pool2</artifactId>
8 </dependency>

```

### 修改application.yaml

```

1 spring:
2   redis:
3     host: 127.0.0.1
4     database: 0

```

### 编写redis配置类

```

1 @Configuration
2 public class RedisConfig {
3   @Autowired
4   private RedisConnectionFactory redisConnectionFactory;
5   @Bean
6   public TokenStore tokenStore(){
7     return new RedisTokenStore(redisConnectionFactory);
8   }
9 }

```

### 在授权服务器配置中指定令牌的存储策略为Redis

```

1 @Autowired
2 private TokenStore tokenStore;
3
4 @Override
5 public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
6   endpoints.authenticationManager(authenticationManagerBean) //使用密码模式需要配置

```

```
7  .tokenStore(tokenStore) //指定token存储到redis
8  .reuseRefreshTokens(false) //refresh_token是否重复使用
9  .userDetailsService(userService) //刷新令牌授权包含对用户信息的检查
10 .allowedTokenEndpointRequestMethods(HttpMethod.GET,HttpMethod.POST); //支持GET,POST请求
11 }
```

文档: 20 Spring Security OAuth2实战.note

链接: [http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=eaab054e79f362d02340114569806d3a&sub=E1946B307F2C4CD5AD5AC703217B3A46)

id=eaab054e79f362d02340114569806d3a&sub=E1946B307F2C4CD5AD5AC703217B3A46