springboot 使用 OAuth2 认证

来源：中柠檬 阅读：1 时间：2017-03-30 20:48

分享：

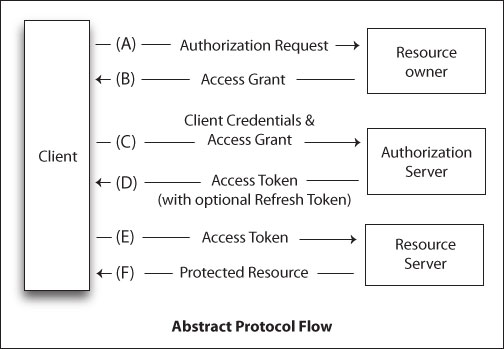
[**【粉丝福利】-《web 前端从入门到放弃系列课程》免费在线直播教学>>>**](https://www.baidu.com/)**Hot**

OAuth2

统一认证

简介

模式



注明:

* Resource Owner：即指需要授权访问的资源，比如用户昵称，头像
* Authorization Server：鉴权服务，核心鉴权逻辑
* Resource Server：资源服务

配合代码中的三个注解：

EnableResourceServer：配置授权资源路径

EnableAuthorizationServer：配置授权服务

EnableOAuth2Client：配置Client信息

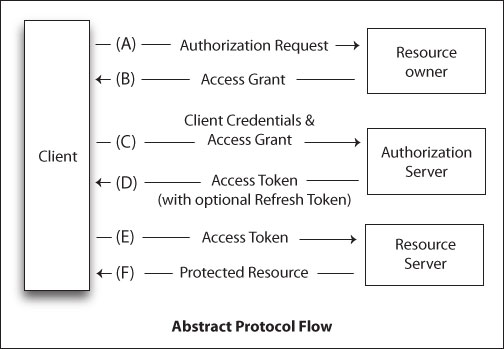
原理

github登录点接入的例子是授权码模式。

自定义的登录使用的是密码模式。

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。



（A）用户打开客户端以后，客户端要求用户给予授权。

（B）用户同意给予客户端授权。

（C）客户端使用上一步获得的授权，向认证服务器申请令牌。

（D）认证服务器对客户端进行认证以后，确认无误，同意发放令牌。

（E）客户端使用令牌，向资源服务器申请获取资源。

（F）资源服务器确认令牌无误，同意向客户端开放资源。

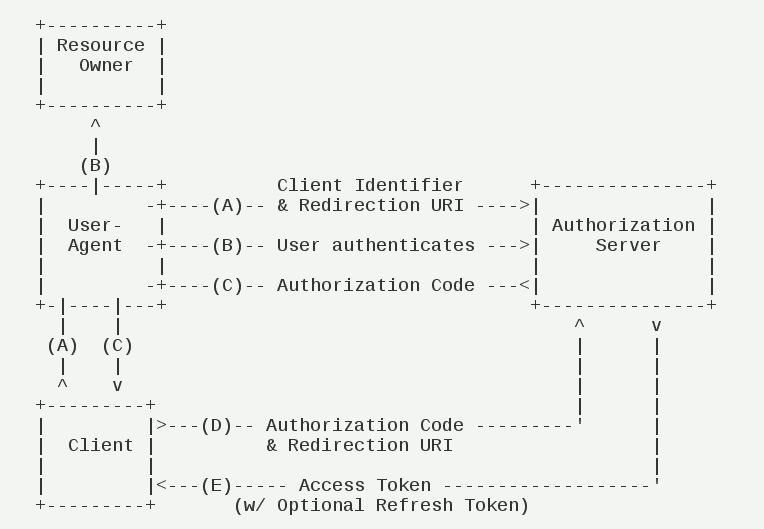
其中，B 是用户给予客户端授权。oauth2 定义了四种方式。分别为

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

授权码模式



上个栏目的流程图只是密码模式。下面看下最复杂的授权码模式：



（A）用户访问客户端，后者将前者导向认证服务器。

（B）用户选择是否给予客户端授权。

（C）假设用户给予授权，认证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"，向认证服务器申请令牌。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）认证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）。

A步骤中，客户端申请认证的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，认证服务器会原封不动地返回这个值。

比如：

GET /authorize?response\_type=code&client\_id=s6BhdRkqt3&state=xyz

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1Host: server.example.com

C步骤中，服务器回应客户端的URI，包含以下参数：

* code：表示授权码，必选项。该码的有效期应该很短，通常设为10分钟，客户端只能使用该码一次，否则会被授权服务器拒绝。该码与客户端ID和重定向URI，是一一对应关系。
* state：如果客户端的请求中包含这个参数，认证服务器的回应也必须一模一样包含这个参数。

比如：

HTTP/1.1 302 FoundLocation: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA

&state=xyz

D步骤中，客户端向认证服务器申请令牌的HTTP请求，包含以下参数：

* grant\_type：表示使用的授权模式，必选项，此处的值固定为"authorization\_code"。
* code：表示上一步获得的授权码，必选项。
* redirect\_uri：表示重定向URI，必选项，且必须与A步骤中的该参数值保持一致。
* client\_id：表示客户端ID，必选项。

比如：

POST /token HTTP/1.1Host: server.example.comAuthorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JWContent-Type: application/x-www-form-urlencoded

grant\_type=authorization\_code&code=SplxlOBeZQQYbYS6WxSbIA

&redirect\_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb

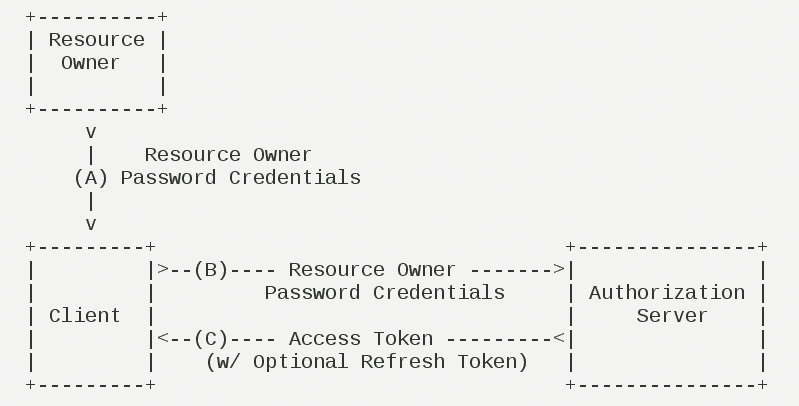
E步骤中，认证服务器发送的HTTP回复，包含以下参数：

* access\_token：表示访问令牌，必选项。
* token\_type：表示令牌类型，该值大小写不敏感，必选项，可以是bearer类型或mac类型。
* expires\_in：表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间。
* refresh\_token：表示更新令牌，用来获取下一次的访问令牌，可选项。
* scope：表示权限范围，如果与客户端申请的范围一致，此项可省略。
* HTTP/1.1 200 OK
* Content-Type: application/json;charset=UTF-8
* Cache-Control: no-store
* Pragma: no-cache
* {
* "access\_token":"2YotnFZFEjr1zCsicMWpAA",
* "token\_type":"example",
* "expires\_in":3600,
* "refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",
* "example\_parameter":"example\_value"
* }

从上面代码可以看到，相关参数使用JSON格式发送（Content-Type: application/json）。此外，HTTP头信息中明确指定不得缓存。

密码模式

密码模式是将授权码模式中的授权码固定为用户名和密码。



（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认证服务器，向后者请求令牌。

（C）认证服务器确认无误后，向客户端提供访问令牌。

B步骤中，客户端发出的HTTP请求，包含以下参数：

* grant\_type：表示授权类型，此处的值固定为"password"，必选项。
* username：表示用户名，必选项。
* password：表示用户的密码，必选项。
* scope：表示权限范围，可选项。

比如：

POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

grant\_type=password&username=johndoe&password=A3ddj3w

C步骤中，认证服务器向客户端发送访问令牌，下面是一个例子。

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access\_token":"2YotnFZFEjr1zCsicMWpAA",

"token\_type":"example",

"expires\_in":3600,

"refresh\_token":"tGzv3JOkF0XG5Qx2TlKWIA",

"example\_parameter":"example\_value"

}

上面代码中，各个参数的含义参见《授权码模式》一节。

整个过程中，客户端不得保存用户的密码。

pom文件

如果添加了spring-security-oauth2依赖，你可以利用自动配置简化认证（Authorization）或资源服务器（Resource Server）的设置。

pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.security.oauth</groupId>

<artifactId>spring-security-oauth2</artifactId>

</dependency>

OAuth2 的配置文件

package club.lemos.sso.config.security;

import javax.annotation.Resource;

import javax.servlet.Filter;

import java.security.Principal;

import java.util.ArrayList;

import java.util.List;

@Configuration

@RestController

@EnableOAuth2Client

@EnableAuthorizationServer

@Order(SecurityProperties.ACCESS\_OVERRIDE\_ORDER)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Resource

private LopUserDetailService userDetailService;

@Resource

private OAuth2ClientContext oauth2ClientContext;

@RequestMapping("/user")

public String user(Principal user) {

return "Hello" + user.getName();

}

@Override

@Bean // share AuthenticationManager for web and oauth

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

/\*\*

\* 主过滤器

\*

\* @param http

\* @throws Exception

\*/

@Override

protected void configure(HttpSecurity http) throws Exception {

http

// .headers()

// .frameOptions()

// .sameOrigin()

// .and()

//.csrf().disable()

// 跨域支持

.cors().and()

.antMatcher("/\*\*")

.authorizeRequests()

.antMatchers("/", "/github", "/login\*\*", "/webjars/\*\*").permitAll()

.anyRequest().authenticated();

http

.exceptionHandling()

.authenticationEntryPoint(new LoginUrlAuthenticationEntryPoint("/login"))

.and()

.formLogin().loginPage("/login").loginProcessingUrl("/login.do").defaultSuccessUrl("/success")

.failureUrl("/login?err=1")

.permitAll();

http

.logout()

.logoutUrl("/logout") //默认只接受post请求处理,需要携带csrf token

.logoutSuccessUrl("/").permitAll()

.invalidateHttpSession(true)

.clearAuthentication(true)

.and().csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()); //csrf for angular

http

.addFilterBefore(ssoFilter(), BasicAuthenticationFilter.class);

}

/\*\*

\* 授权服务器(定义UserDetails类)

\*

\*/

@Override

protected void configure(AuthenticationManagerBuilder auth)

throws Exception {

// Configure spring security's authenticationManager with custom

// user details service

auth.userDetailsService(this.userDetailService);

}

/\*\*

\* 过滤器(第三方,需要注入到主过滤器)

\*

\*/

private Filter ssoFilter() {

CompositeFilter filter = new CompositeFilter();

List<Filter> filters = new ArrayList<>();

filters.add(ssoFilter(github(), "/login/github"));

filter.setFilters(filters);

return filter;

}

@Bean

@ConfigurationProperties("github")

public ClientResources github() {

return new ClientResources();

}

/\*\*

\* 支持从本地重定向到第三方,由异常触发

\*

\*/

@Bean

public FilterRegistrationBean oauth2ClientFilterRegistration(

OAuth2ClientContextFilter filter) {

FilterRegistrationBean registration = new FilterRegistrationBean();

registration.setFilter(filter);

registration.setOrder(-100);

return registration;

}

/\*\*

\* 本地的资源服务器

\*

\*/

@Configuration

@EnableResourceServer

protected static class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

@Override

public void configure(HttpSecurity http) throws Exception {

http.antMatcher("/api/\*\*").authorizeRequests().anyRequest().authenticated();

}

}

private Filter ssoFilter(ClientResources client, String path) {

OAuth2ClientAuthenticationProcessingFilter filter = new OAuth2ClientAuthenticationProcessingFilter(path);

OAuth2RestTemplate template = new OAuth2RestTemplate(client.getClient(), oauth2ClientContext);

filter.setRestTemplate(template);

UserInfoTokenServices tokenServices = new UserInfoTokenServices(

client.getResource().getUserInfoUri(), client.getClient().getClientId());

tokenServices.setRestTemplate(template);

filter.setTokenServices(tokenServices);

return filter;

}

class ClientResources {

@NestedConfigurationProperty

private AuthorizationCodeResourceDetails client = new AuthorizationCodeResourceDetails();

@NestedConfigurationProperty

private ResourceServerProperties resource = new ResourceServerProperties();

public AuthorizationCodeResourceDetails getClient() {

return client;

}

public ResourceServerProperties getResource() {

return resource;

}

}

}

application.properties

github 的相关配置需要手动注入（@ConfigurationProperties("github")

# 服务端 oauth2基本配置

github.client.client-id=dd2bf79a9e6be256f0e8

github.client.client-secret=0e555a2ee5d627e3abdee3f5096de6d8278d3413

# user 手动授权,获得 code

github.client.user-authorization-uri=https://github.com/login/oauth/authorize

# 根据 "code,id和secret",获取 user token

github.client.access-token-uri=https://github.com/login/oauth/access\_token

github.client.token-name=oauth\_token

github.client.authentication-scheme=query

github.client.client-authentication-scheme=form

github.client.scope=user

github.resource.user-info-uri=https://api.github.com/user

#重定向url,必须是 callbackURL的路径或者子路径

#github.client.client.pre-established-redirect-uri=http://localhost:8090/login/github/users

# oauth2 自己的授权服务器基本配置支持

security.oauth2.client.client-id=client

security.oauth2.client.client-secret=secret

security.oauth2.client.scope=read,write

security.oauth2.client.auto-approve-scopes='.\*'

security.oauth2.client.grant-type=password

security.basic.enabled=false

配置说明

一

需要注意一点，@EnableOAuth2Sso 有两个作用

oauth2Client 和 authentication(授权),客户端负责与服务器(github)进行交互,

authentication 负责跟spring security 对齐.

使用 更低级的 @EnableOAuth2Client 注解替换@EnableOAuth2Sso,可以丰富oauth2Client 的功能:

可以注入 OAuth2ClientContext

@Autowired

OAuth2ClientContext oauth2ClientContext;

添加认证 Filter(由注入的 OAuth2ClientContext 创建)

@Override

protected void configure(HttpSecurity http) throws Exception {

http.antMatcher("/\*\*")

...

.addFilterBefore(ssoFilter(), BasicAuthenticationFilter.class);

}

二

如果出现 HTTP URL must be null 这个错误，控制台或者页面都有可能出现这个。

那么，一定是Resource Server 问题。

资源服务器(第三方,需要注册到过滤器中)

@ConfigurationProperties("github")

public ClientResources githubResource() {

return new ClientResources();

}

如果不配置或者属性名称不对，则访问报错

@ConfigurationProperties("github")

或者写错了，将这里的

github.resource.user-info-uri=https://api.github.com/user

写成

github.client.resource.user-info-uri=https://api.github.com/user

参考文档| <https://spring.io/guides/tutorials/spring-boot-oauth2/>

不使用自授权服务器，可以无视

UserDetailServie

由于自己的授权服务器，采用的不是授权码方式，而是password 认证。需要定制一个UserDeatilService

package club.lemos.sso.service.impl;

import club.lemos.sso.domain.UserInfo;

import org.apache.log4j.LogManager;

import org.apache.log4j.Logger;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

/\*\*

\* 自定义 UserDetailsService实现

\* Created by lg on 2017/3/25.

\*/

@Service

public class FavUserDetailService implements UserDetailsService {

private static final Logger logger = LogManager.getLogger(FavUserDetailService.class);

/\*\*

\* 根据用户名获取用户 - 用户的角色、权限等信息

\*/

public UserDetails loadUserByUsername(String username)

throws UsernameNotFoundException {

UserDetails userDetails = null;

try {

UserInfo favUser = new UserInfo();

favUser.setUsername("admin");

favUser.setPassword("admin");

Collection<GrantedAuthority> authList = getAuthorities();

userDetails = new User(username, favUser.getPassword().toLowerCase(),true,true,true,true,authList);

} catch (Exception e) {

e.printStackTrace();

}

return userDetails;

}

/\*\*

\* 获取用户的角色权限,为了降低实验的难度，这里去掉了根据用户名获取角色的步骤

\* @param

\* @return

\*/

private Collection<GrantedAuthority> getAuthorities(){

List<GrantedAuthority> authList = new ArrayList<GrantedAuthority>();

authList.add(new SimpleGrantedAuthority("ROLE\_USER"));

authList.add(new SimpleGrantedAuthority("ROLE\_ADMIN"));

return authList;

}

}

简单访问 自授权服务器 API

之前的Web端（B\S结构），可以正常通信。登录跳转什么的。对于受限资源，需要通过 oauth2授权( C\S 结构)。可以通过Web端发送一个请求进行认证。认证成功，获得 token，可以使用 token访问f服务器受限资源。

普通登录直接通过 security api，就可以获得用户信息。

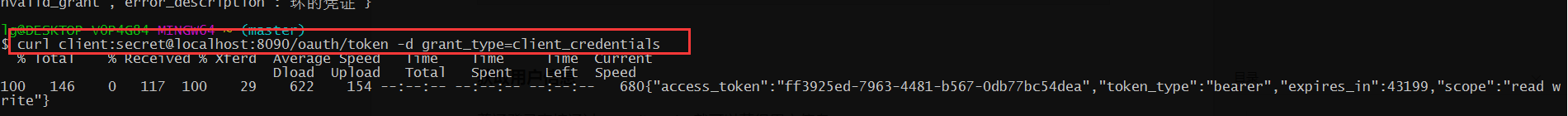
oauth2 登录，需要先获得 token，再用它访问 github api，获取用户信息。

上面那种，看 security文档即可。关于 oauth2，探讨如下。

获取 TOKEN 的两种方式

第一种方式:证书 token

curl client:secret@localhost:8090/oauth/token -d grant\_type=client\_credentials



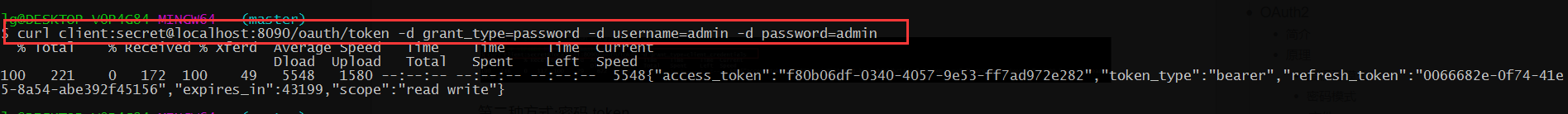
第二种方式:密码 token

当应用启动时，springboot 会创建一个默认的用户，用户id为‘user‘。密码是随机的，但可以从打印的日志中看到。

curl client:secret@localhost:8090/oauth/token -d grant\_type=password -d username=user -d password=...

当然，如果配置了授权信息。可以使用已经定义好的密码。这里我们设置的是 admin admin

curl client:secret@localhost:8090/oauth/token -d grant\_type=password -d username=admin -d password=admin



或者还有一种写法

curl -H "Authorization: Basic Y2xpZW50OnNlY3JldA==" http://localhost:8090/oauth/token -d "username=admin&password=admin&scope=read&grant\_type=password"

使用 密码 Token 获取用户资源

curl http://localhost:8090/api/users -H "Authorization: bearer 7e7b7ced-3747-43a2-8134-c7e6b87c6451"

CURL 说明

获取 token需要设置 Authorization BASIC，设置方法有两种：

在url中添加用户名和密码

http://userName:password@api.minicloud.com.cn/statuses/friends\_timeline.xml

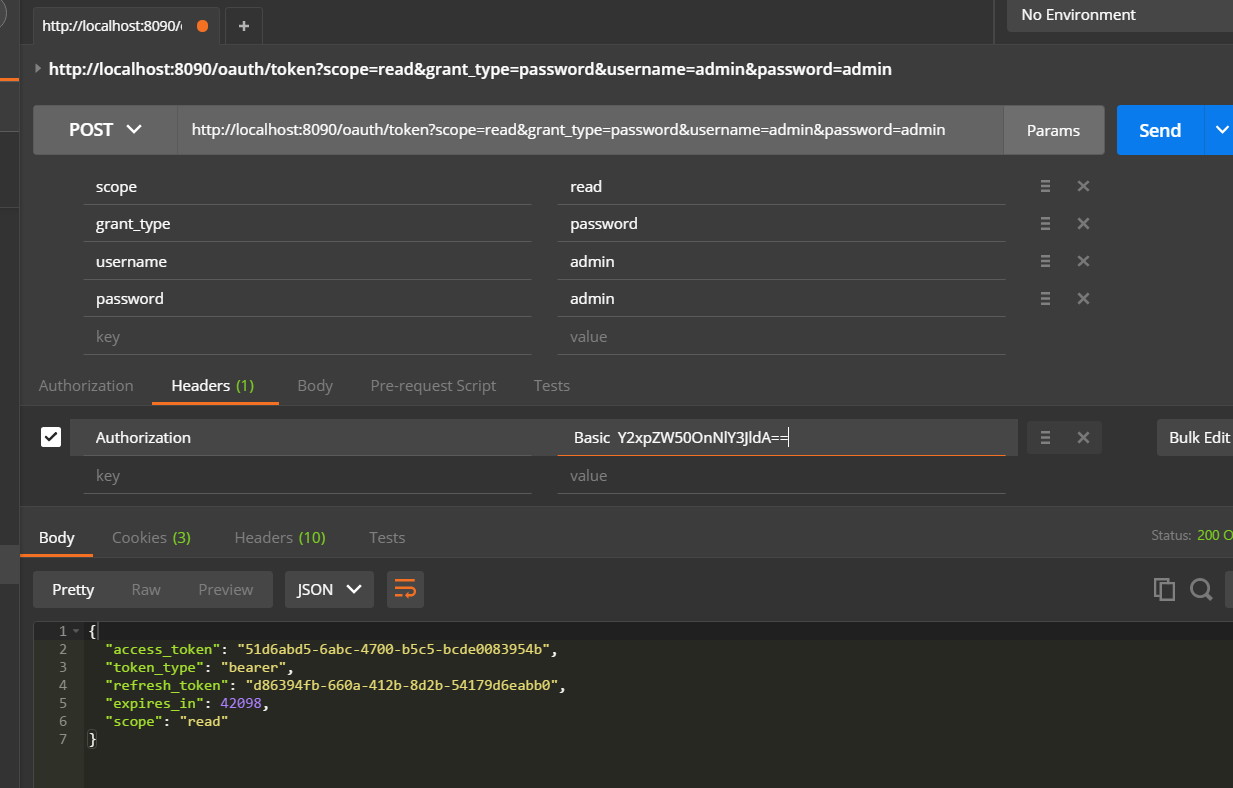
在请求头中添加Authorization

Authorization: "Basic 用户名和密码的base64加密字符串"

js 中的写法

根据上面的说明，使用 Basic认证，访问获取 token

curl -H "Authorization: Basic Y2xpZW50OnNlY3JldA==" http://localhost:8090/oauth/token -d "username=admin&password=admin&scope=read&grant\_type=password"



在 javascript 中可以通过添加头信息的方式发送请求

这里的关键问题在于 Authtication 的 Basic 密钥怎么获得?

$.ajaxSetup({ headers: {

'X-CSRF-Token': 'xxx'

}

});

或者

$.ajax({

type: "GET",

url: "index1.php",

dataType: 'json',

async: false,

username: username,

password: password,

data: '{ "comment" }',

success: function (){

alert('Thanks for your comment!');

}});

再或者

$.ajax({

type: "GET",

url: "index1.php",

dataType: 'json',

async: false,

headers: {

"Authorization": "Basic " + btoa(USERNAME + ":" + PASSWORD)

},

data: '{ "comment" }',

success: function (){

alert('Thanks for your comment!');

}});

Github oauth2 实现

基本配置已经在上面实现，主要操作是把 application.properties 中的相关属性修改为自己的。具体可以从 github上[申请](https://github.com/settings/developers)。

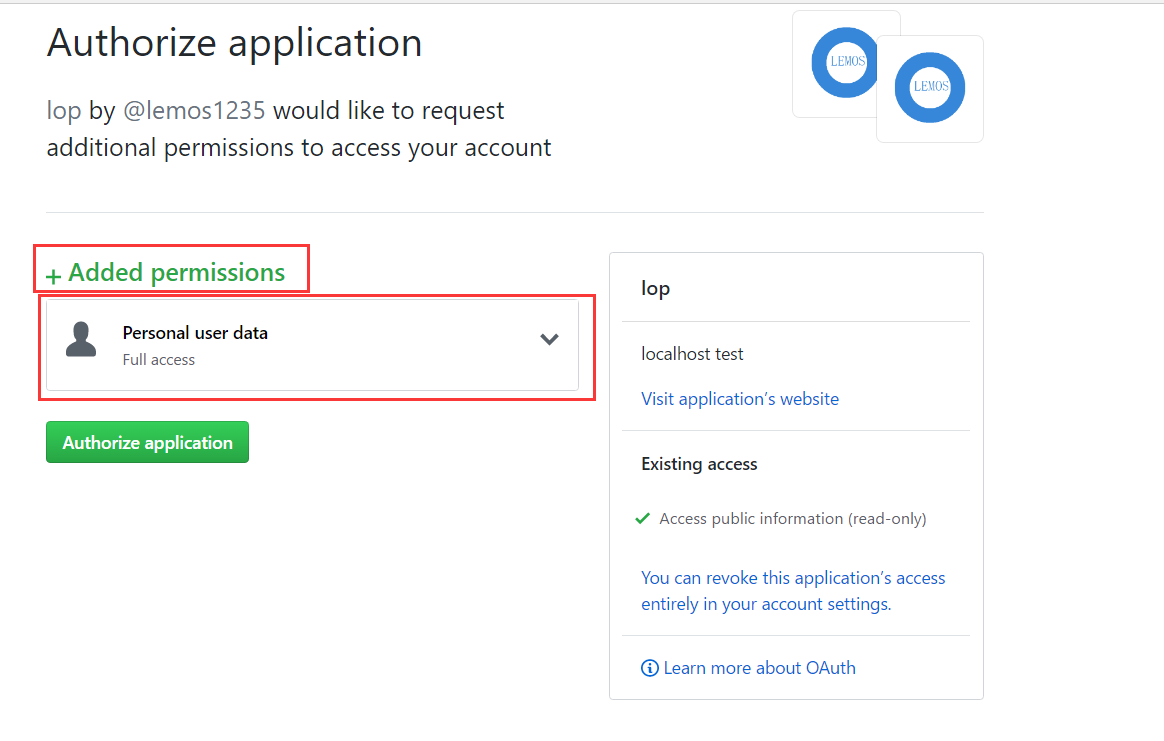
执行流程

一，点击页面的 callback超链接。

callbackURL = http://localhost:8090/login/github  GET

二，重定向到授权页面，用户点击授权

user-authorization-uri = https://github.com/login/oauth/authorize?client\_id=141c0a61de83cf2d9841&redirect\_uri=http://localhost:8090/login/github&response\_type=code&scope=user&state=4jgVT2



三，重定向回自己的页面，并携带一个 code (授权码) 和 前一步中的 state参数，如果 states 匹配，则可以发送一个 POST <https://github.com/login/oauth/access_token>

http://localhost:8090/login/github?code=2c6dcdce82ef1473e148&state=4jgVT2

四，请求 token（授权成功，会自动发送这个请求）

https://github.com/login/oauth/access\_token    POST

响应 token(包含着授权信息，存储在 JSESSION 中）

access\_token=e72e16c7e42f292c6912e7710c838347ae178b4a&scope=user%2Cgist&token\_type=bearer

或者

Accept: application/json {"access\_token":"e72e16c7e42f292c6912e7710c838347ae178b4a", "scope":"repo,gist", "token\_type":"bearer"}

五，访问 Github API

GET https://api.github.com/user?access\_token=...

或者

Authorization: token OAUTH-TOKEN

原文 | <https://developer.github.com/v3/oauth/>

基本信息获取

请看 github v3 api | <https://developer.github.com/v3/>