【Spring Authorization Server 系列】（一）入门篇，快速搭建一个授权服务器

Spring Authorization Server

简介

官方主页：https://spring.io/projects/spring-authorization-server

Spring Authorization Server 是一个框架，提供了 OAuth 2.1 和 OpenID Connect 1.0 规范以及其他相关规范的实现。 它建立在 Spring Security 之上，为构建 OpenID Connect 1.0 Identity Providers 和 OAuth2 Authorization Server 产品提供安全、轻量级和可定制的基础。

Spring实现OAuth2的旧版的框架是 Spring Security OAuth2，该框架已停止维护。Spring Security OAuth2认证/授权服务器的前世今生

功能列表

https://docs.spring.io/spring-authorization-server/docs/0.3.1/reference/html/overview.html

授权方式支持授权码（Authorization Code）、客户凭证（Client Credentials）

令牌支持刷新（Refresh Token）

令牌格式支持自包含（JWT）和引用（Opaque）

…

入门案例

官方案例：https://docs.spring.io/spring-authorization-server/docs/0.3.1/reference/html/getting-started.html

本案例基于官方案例，做了一丝丝改动，便于调试。

1. 创建 SpringBoot 项目

一个 pom，一个 Application 启动类，相信难不倒你，跳过。

2. 引入依赖

本案例使用 0.3.1 版本

<properties>

<spring-authorization-server.version>0.3.1</spring-authorization-server.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<version>${spring-authorization-server.version}</version>

</dependency>

</dependencies>

3. 编写配置

配置类

为了便于调试授权码流程，给客户端多加了个重定向地址 .redirectUri("https://cn.bing.com")

@Configuration

public class SecurityConfiguration {

@Bean

@Order(1)

public SecurityFilterChain authorizationServerSecurityFilterChain(HttpSecurity http)

throws Exception {

OAuth2AuthorizationServerConfiguration.applyDefaultSecurity(http);

http

// Redirect to the login page when not authenticated from the

// authorization endpoint

.exceptionHandling((exceptions) -> exceptions

.authenticationEntryPoint(

new LoginUrlAuthenticationEntryPoint("/login"))

);

return http.build();

}

@Bean

@Order(2)

public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http)

throws Exception {

http

.authorizeHttpRequests((authorize) -> authorize

.anyRequest().authenticated()

)

// Form login handles the redirect to the login page from the

// authorization server filter chain

.formLogin(Customizer.withDefaults());

return http.build();

}

@Bean

public UserDetailsService userDetailsService() {

UserDetails userDetails = User.withDefaultPasswordEncoder()

.username("user")

.password("password")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(userDetails);

}

@Bean

public RegisteredClientRepository registeredClientRepository() {

RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID().toString())

.clientId("messaging-client")

.clientSecret("{noop}secret")

.clientAuthenticationMethod(ClientAuthenticationMethod.CLIENT\_SECRET\_BASIC)

.authorizationGrantType(AuthorizationGrantType.AUTHORIZATION\_CODE)

.authorizationGrantType(AuthorizationGrantType.REFRESH\_TOKEN)

.authorizationGrantType(AuthorizationGrantType.CLIENT\_CREDENTIALS)

.redirectUri("http://127.0.0.1:8080/login/oauth2/code/messaging-client-oidc")

.redirectUri("http://127.0.0.1:8080/authorized")

.redirectUri("https://cn.bing.com") // 便于调试授权码流程

.scope(OidcScopes.OPENID)

.scope("message.read")

.scope("message.write")

.clientSettings(ClientSettings.builder().requireAuthorizationConsent(true).build())

.build();

return new InMemoryRegisteredClientRepository(registeredClient);

}

@Bean

public JWKSource<SecurityContext> jwkSource() {

KeyPair keyPair = generateRsaKey();

RSAPublicKey publicKey = (RSAPublicKey) keyPair.getPublic();

RSAPrivateKey privateKey = (RSAPrivateKey) keyPair.getPrivate();

RSAKey rsaKey = new RSAKey.Builder(publicKey)

.privateKey(privateKey)

.keyID(UUID.randomUUID().toString())

.build();

JWKSet jwkSet = new JWKSet(rsaKey);

return new ImmutableJWKSet<>(jwkSet);

}

private static KeyPair generateRsaKey() {

KeyPair keyPair;

try {

KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");

keyPairGenerator.initialize(2048);

keyPair = keyPairGenerator.generateKeyPair();

}

catch (Exception ex) {

throw new IllegalStateException(ex);

}

return keyPair;

}

@Bean

public ProviderSettings providerSettings() {

return ProviderSettings.builder().build();

}

}

application.yml

服务端口为 9000

框架的异常信息不是很友好，此处增加了两个类的 trace 日志，便于定位问题。

server:

port: 9000

logging:

level:

root: info

# org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping: trace

# org.springframework.security: debug

org.springframework.security.web.FilterChainProxy: trace # 过滤器执行顺序

org.springframework.security.web.access.ExceptionTranslationFilter: trace #异常处理

4. 启动

运行 启动类

5. 测试

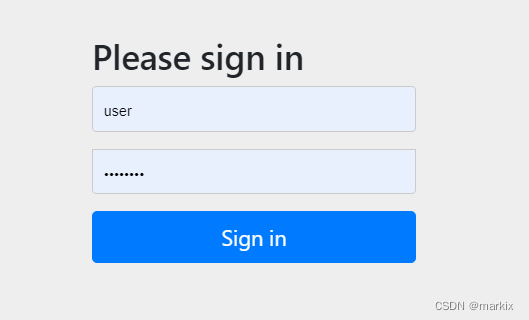
授权码 获取令牌

1. 三方应用访问授权页

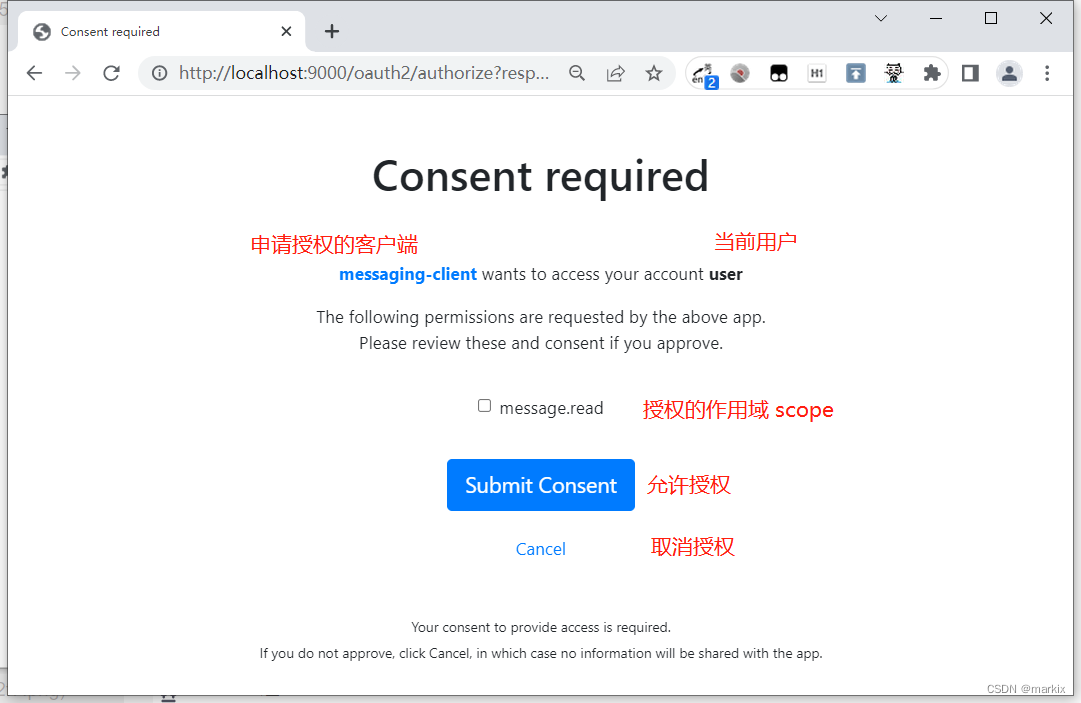
http://localhost:9000/oauth2/authorize?response\_type=code&client\_id=messaging-client&scope=message.read&redirect\_uri=https://cn.bing.com

1.1 若用户未登录，则会重定向登录页

输入配置类中的指定的用户信息 user、password，登录，登录成功后会重定向到授权页



1.2. 授权页



2. 用户授权

勾选授权作用域scope（即message.read），点击 Submit Consent 允许授权

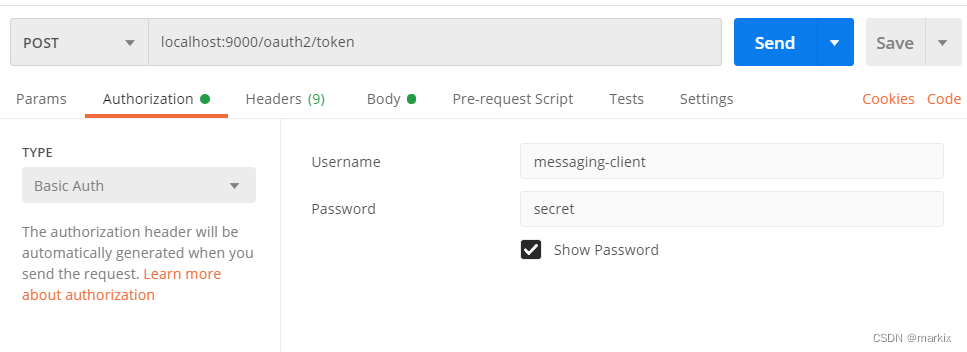
3. 获取code

IMG_256

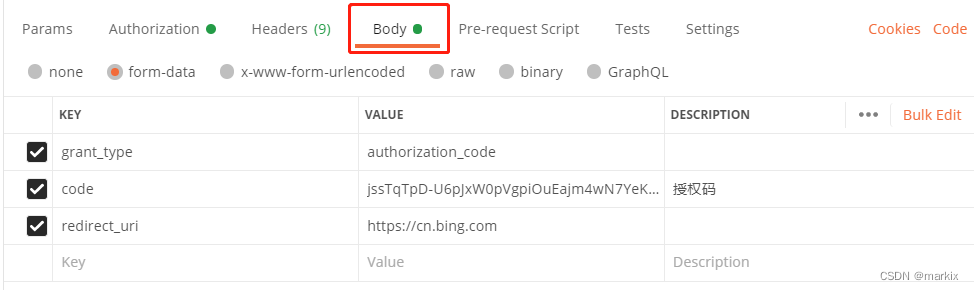
授权成功后，会重定向到上述地址并携带 code 值

4. 通过 code 获取令牌

使用postman发送请求 POST /oauth2/token，需要注意的是客户端的信息以 Basic 方式认证



填入上步骤获取的 code 值



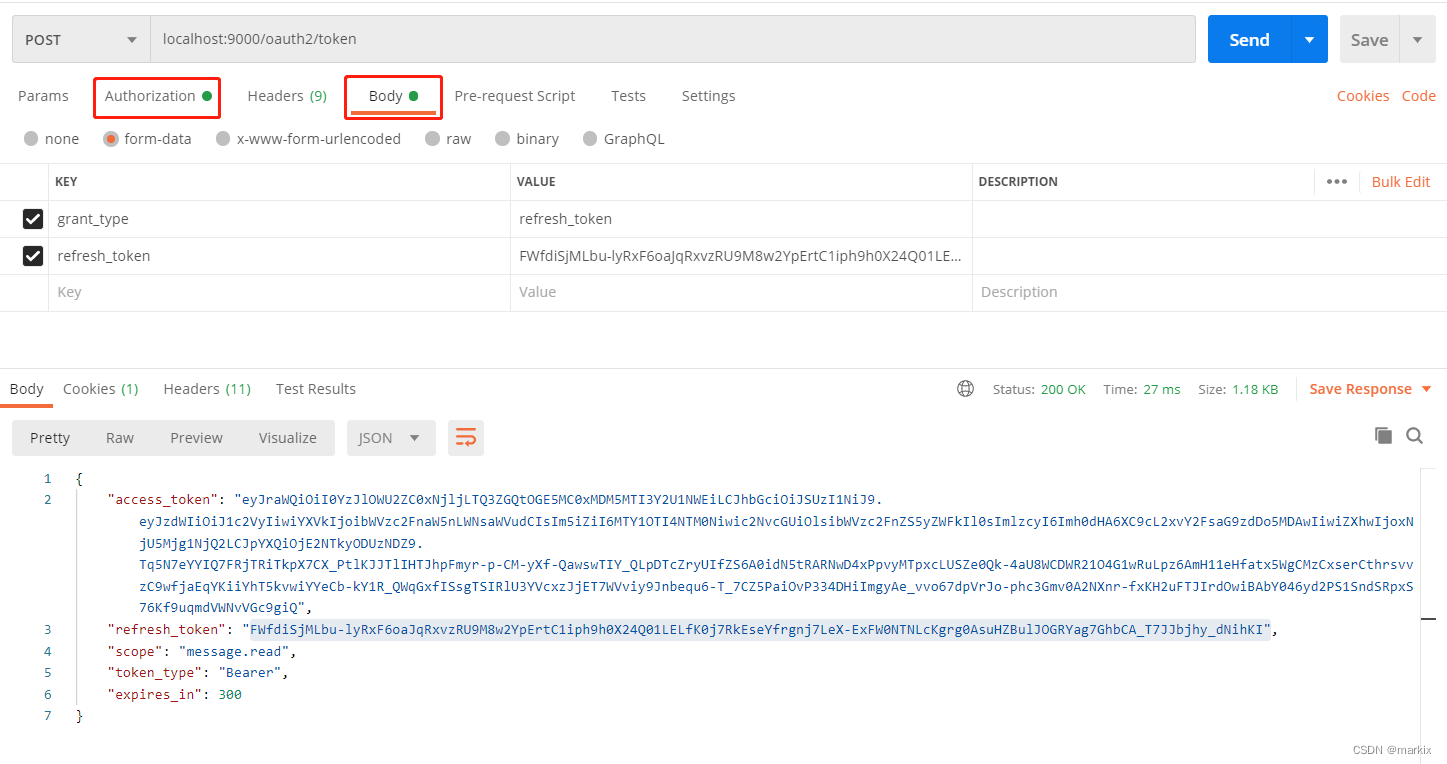
发送请求，得到响应，access\_token 即为令牌。



刷新令牌

同样是 POST /oauth2/token 接口，仅参数不同，客户端的信息同样以 Basic 方式认证

使用上步骤获取的 refresh\_token 测试。



客户端凭证 获取令牌

同样是 POST /oauth2/token 接口，仅参数不同，客户端的信息同样以 Basic 方式认证

end