链滴

SpringCloud Alibaba 微服务实战十四 - SpringCloud Gateway 集成 Oauth2.0

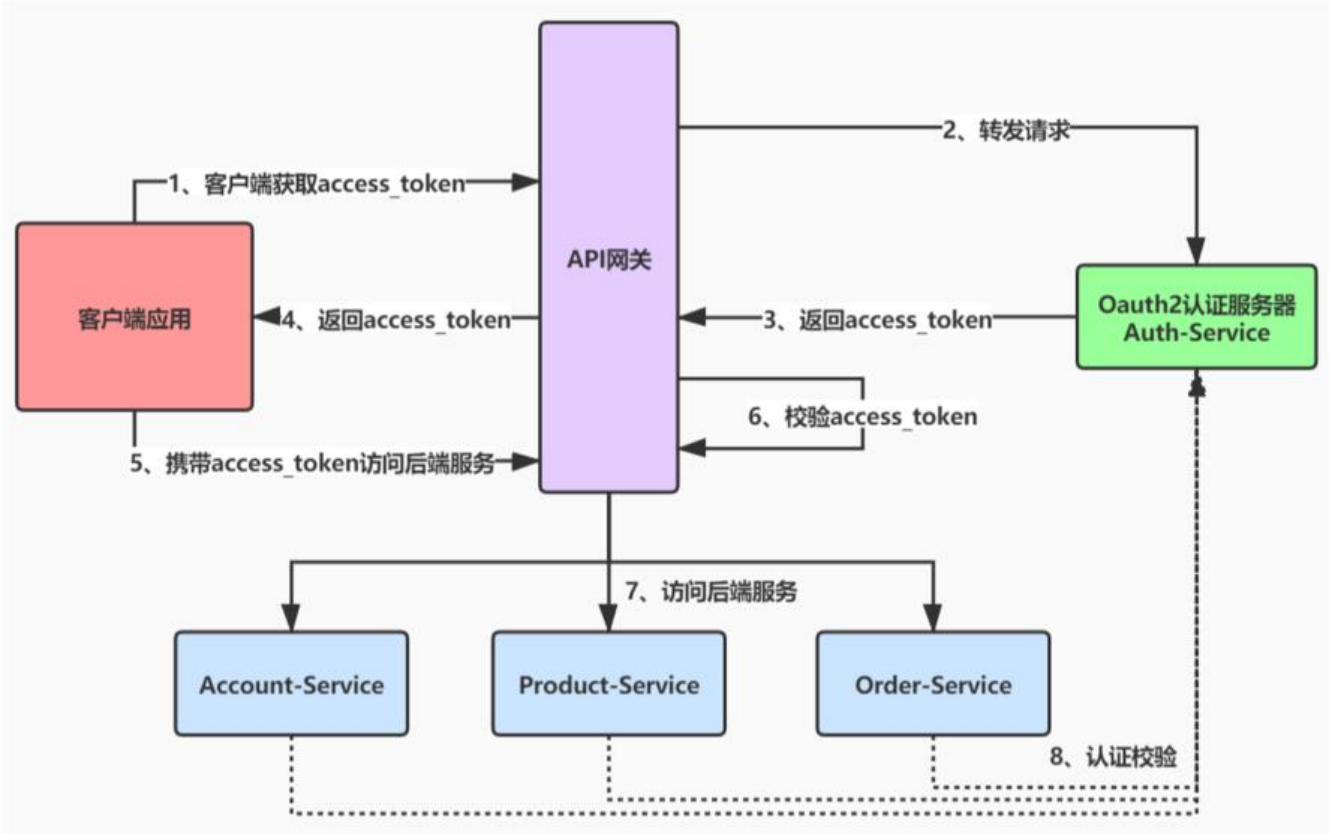
作者：jianzh5

原文链接：https://ld246.com/article/1584427764248

来源网站：链滴

导读：上篇文章我们已经抽取出了单独的认证服务，本章主要内容是让SpringCloud Gateway 集成Oauth2。

概念部分



在网关集成Oauth2.0后，我们的流程架构如上。主要逻辑如下：

1、客户端应用通过api网关请求认证服务器获取access_token http://localhost:8090/auth-service/auth/token

2、认证服务器返回access_token

```
{
  "access_token": "f938d0c1-9633-460d-acdd-f0693a6b5f4c",
  "token_type": "bearer",
  "refresh_token": "4baea735-3c0d-4dfd-b826-91c6772a0962",
  "expires_in": 43199,
  "scope": "web"
}
```

3、客户端携带access_token通过API网关访问后端服务

4、API网关收到access_token后通过 AuthenticationWebFilter 对access_token认证

5、API网关转发后端请求，后端服务请求Oauth2认证服务器获取当前用户

在前面文章中我们搭建好了单独的Oauth2认证授权服务，基本功能框架都实现了，这次主要是来实第四条，SpringCloud 整合 Oauth2 后如何进行access_token过滤校验。

代码示例

引入组件

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-oauth2</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

主要引入跟oauth2相关的jar包，这里还需要引入数据库相关的jar包，因为我们的token是存在数据中，要想在网关层校验token的有效性必须先从数据库取出token。

bootstrap.yml 配置修改

```
spring:
  application:
    name: cloud-gateway
  datasource:
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:mysql://xx.0.xx.xx:3306/oauth2_config?characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=false
    username: xxxxx
    password: xxxxxxx
    driver-class-name: com.mysql.jdbc.Driver
```

主要配置oauth2的数据库连接地址

自定义认证接口管理类

在webFlux环境下通过实现 ReactiveAuthenticationManager 接口 自定义认证接口管理，由于我们token是存在jdbc中所以命名上就叫ReactiveJdbcAuthenticationManager

```
@Slf4j
public class ReactiveJdbcAuthenticationManager implements ReactiveAuthenticationManager
{

    private TokenStore tokenStore;

    public JdbcAuthenticationManager(TokenStore tokenStore){
        this.tokenStore = tokenStore;
    }

    @Override
    public Mono<Authentication> authenticate(Authentication authentication) {
        return Mono.justOrEmpty(authentication)
                .filter(a -> a instanceof BearerTokenAuthenticationToken)
                .cast(BearerTokenAuthenticationToken.class)
                .map(BearerTokenAuthenticationToken::getToken)
                .flatMap((accessToken ->{
                    log.info("accessToken is :{}",accessToken);
                    OAuth2AccessToken oAuth2AccessToken = this.tokenStore.readAccessToken(acc
ssToken);
                    //根据access_token从数据库获取不到OAuth2AccessToken
                    if(oAuth2AccessToken == null){
                        return Mono.error(new InvalidTokenException("invalid access token,please che
k"));
                    }else if(oAuth2AccessToken.isExpired()){
                        return Mono.error(new InvalidTokenException("access token has expired,please
reacquire token"));
                    }

                    OAuth2Authentication oAuth2Authentication =this.tokenStore.readAuthenticatio
(accessToken);
                    if(oAuth2Authentication == null){
                        return Mono.error(new InvalidTokenException("Access Token 无效!"));
                    }else {
                        return Mono.just(oAuth2Authentication);
                    }
                })).cast(Authentication.class);
    }
}
```

网关层的安全配置

```
@Configuration
public class SecurityConfig {
    private static final String MAX_AGE = "18000L";
    @Autowired
    private DataSource dataSource;
    @Autowired
```

```
    private AccessManager accessManager;

    /**
     * 跨域配置
     */
    public WebFilter corsFilter() {
        return (ServerWebExchange ctx, WebFilterChain chain) -> {
            ServerHttpRequest request = ctx.getRequest();
            if (CorsUtils.isCorsRequest(request)) {
                HttpHeaders requestHeaders = request.getHeaders();
                ServerHttpResponse response = ctx.getResponse();
                HttpMethod requestMethod = requestHeaders.getAccessControlRequestMethod();
                HttpHeaders headers = response.getHeaders();
                headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_ORIGIN, requestHeaders.get
rigin());
                headers.addAll(HttpHeaders.ACCESS_CONTROL_ALLOW_HEADERS, requestHeaders
getAccessControlRequestHeaders());
                if (requestMethod != null) {
                    headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_METHODS, requestMethod
name());
                }
                headers.add(HttpHeaders.ACCESS_CONTROL_ALLOW_CREDENTIALS, "true");
                headers.add(HttpHeaders.ACCESS_CONTROL_EXPOSE_HEADERS, "*");
                headers.add(HttpHeaders.ACCESS_CONTROL_MAX_AGE, MAX_AGE);
                if (request.getMethod() == HttpMethod.OPTIONS) {
                    response.setStatusCode(HttpStatus.OK);
                    return Mono.empty();
                }
            }
            return chain.filter(ctx);
        };
    }

    @Bean
    SecurityWebFilterChain webFluxSecurityFilterChain(ServerHttpSecurity http) throws Excepti
n{
        //token管理器
        ReactiveAuthenticationManager tokenAuthenticationManager = new ReactiveJdbcAuthe
ticationManager(new JdbcTokenStore(dataSource));
        //认证过滤器
        AuthenticationWebFilter authenticationWebFilter = new AuthenticationWebFilter(token
uthenticationManager);
        authenticationWebFilter.setServerAuthenticationConverter(new ServerBearerTokenAuth
nticationConverter());

        http
                .httpBasic().disable()
                .csrf().disable()
                .authorizeExchange()
                .pathMatchers(HttpMethod.OPTIONS).permitAll()
                .anyExchange().access(accessManager)
                .and()
                // 跨域过滤器
                .addFilterAt(corsFilter(), SecurityWebFiltersOrder.CORS)
```

```
                //oauth2认证过滤器
                .addFilterAt(authenticationWebFilter, SecurityWebFiltersOrder.AUTHENTICATION);
        return http.build();
    }
}
```

这个类是SpringCloug Gateway 与 Oauth2整合的关键，通过构建认证过滤器 AuthenticationWebFiter 完成Oauth2.0的token校验。

AuthenticationWebFilter 通过我们自定义的 ReactiveJdbcAuthenticationManager 完成token校验。

我们在这里还加入了CORS过滤器，以及权限管理器 AccessManager

权限管理器

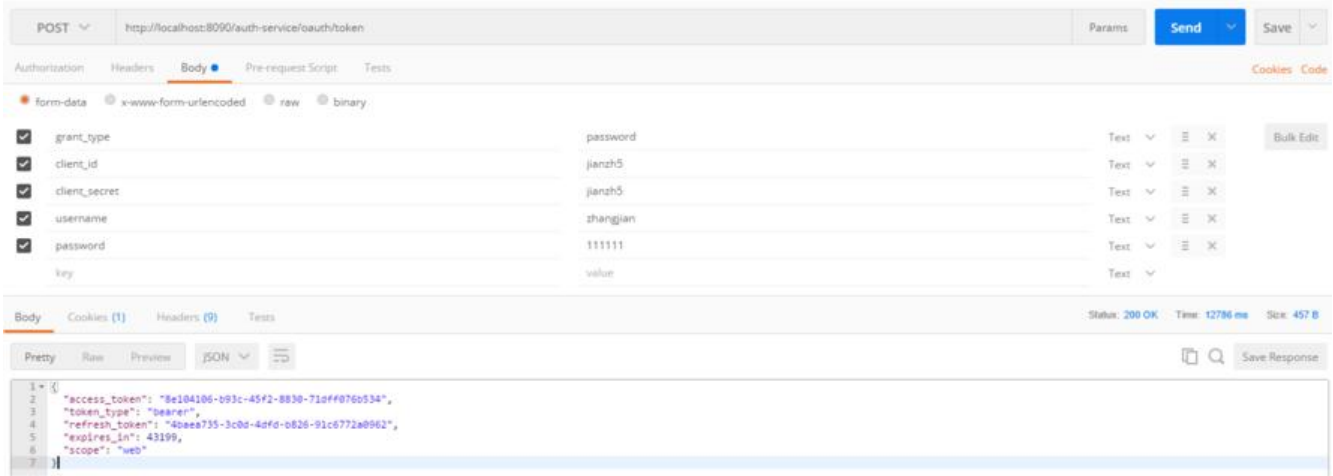
```
@Slf4j
@Component
public class AccessManager implements ReactiveAuthorizationManager<AuthorizationContex
> {
    private Set<String> permitAll = new ConcurrentHashSet<>();
    private static final AntPathMatcher antPathMatcher = new AntPathMatcher();


    public AccessManager (){
        permitAll.add("/");
        permitAll.add("/error");
        permitAll.add("/favicon.ico");
        permitAll.add("/**/v2/api-docs/**");
        permitAll.add("/**/swagger-resources/**");
        permitAll.add("/webjars/**");
        permitAll.add("/doc.html");
        permitAll.add("/swagger-ui.html");
        permitAll.add("/**/oauth/**");
        permitAll.add("/**/current/get");
    }

    /**
     * 实现权限验证判断
     */
    @Override
    public Mono<AuthorizationDecision> check(Mono<Authentication> authenticationMono,
AuthorizationContext authorizationContext) {
        ServerWebExchange exchange = authorizationContext.getExchange();
        //请求资源
        String requestPath = exchange.getRequest().getURI().getPath();
        // 是否直接放行
        if (permitAll(requestPath)) {
            return Mono.just(new AuthorizationDecision(true));
        }

        return authenticationMono.map(auth -> {
            return new AuthorizationDecision(checkAuthorities(exchange, auth, requestPath));
        }).defaultIfEmpty(new AuthorizationDecision(false));
```

```
    }

    /**
     * 校验是否属于静态资源
     * @param requestPath 请求路径
     * @return
     */
    private boolean permitAll(String requestPath) {
        return permitAll.stream()
                .filter(r -> antPathMatcher.match(r, requestPath)).findFirst().isPresent();
    }

    //权限校验
    private boolean checkAuthorities(ServerWebExchange exchange, Authentication auth, Strin
 requestPath) {
        if(auth instanceof OAuth2Authentication){
            OAuth2Authentication athentication = (OAuth2Authentication) auth;
            String clientId = athentication.getOAuth2Request().getClientId();
            log.info("clientId is {}",clientId);
        }

        Object principal = auth.getPrincipal();
        log.info("用户信息:{}",principal.toString());
        return true;
    }
}
```

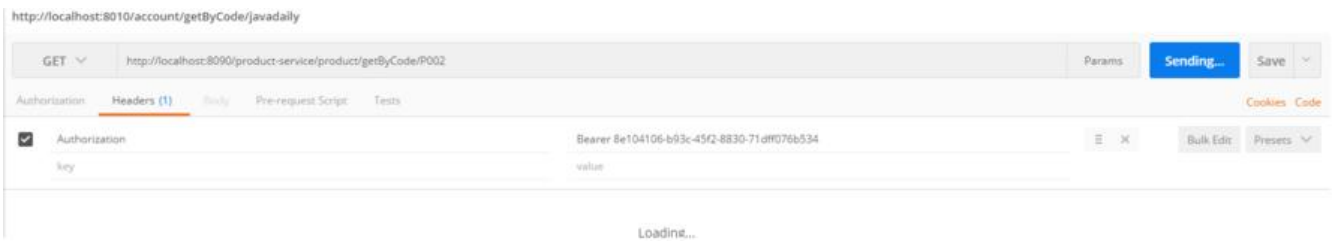
主要是过滤掉静态资源，将来一些接口权限校验也可以放在这里。

测试

- 通过网关调用auth-service获取 access_token



- 在Header上添加认证访问后端服务



- 网关过滤器进行token校验

```
@Override
public Mono<Authentication> authenticate(Authentication authentication) {  authentication: "org.springframework.s
    return Mono.justOrEmpty(authentication) Mono<Authentication>  authentication: "org.springframework.security.oau
            .filter(a -> a instanceof BearerTokenAuthenticationToken)
            .cast(BearerTokenAuthenticationToken.class) Mono<BearerTokenAuthenticationToken>
            .map(BearerTokenAuthenticationToken::getToken) Mono<String>
            .flatMap((accessToken ->{
                log.info("accessToken is :{}",accessToken);
                OAuth2AccessToken oAuth2AccessToken = this.tokenStore.readAccessToken(accessToken);
                //根据access_token从数据库获取不到OAuth2AccessToken
                if(oAuth2AccessToken == null){
                    return Mono.error(new InvalidTokenException("invalid access token,please check"));
                }else if(oAuth2AccessToken.isExpired()){
                    return Mono.error(new InvalidTokenException("access token has expired,please reacquire token"
                }

                OAuth2Authentication oAuth2Authentication =this.tokenStore.readAuthentication(accessToken);
                if(oAuth2Authentication == null){
                    return Mono.error(new InvalidTokenException("Access Token 无效!"));
                }else {
                    return Mono.just(oAuth2Authentication);
                }
            })).cast(Authentication.class);
}
}
```

- 权限管理器校验

```
//权限校验
private boolean checkAuthorities(ServerWebExchange exchange, Authentication auth, String requestPath) {  exch
    if(auth instanceof OAuth2Authentication){
        OAuth2Authentication athentication = (OAuth2Authentication) auth;  athentication: "org.springframewor
        String clientId = athentication.getOAuth2Request().getClientId();  clientId: "jianzh5"  athentication
        log.info("clientId is {}",clientId);  clientId: "jianzh5"
    }

    Object principal = auth.getPrincipal();
    log.info("用户信息:{}",principal.toString());
    return true;
}
}
```

- 去认证服务器校验当前用户

```
     * @return 授权信息
     */
    @GetMapping("current/get")
    public Principal user(Principal prin
        return principal;  principal: "o
    }

}
```

UserController › user()

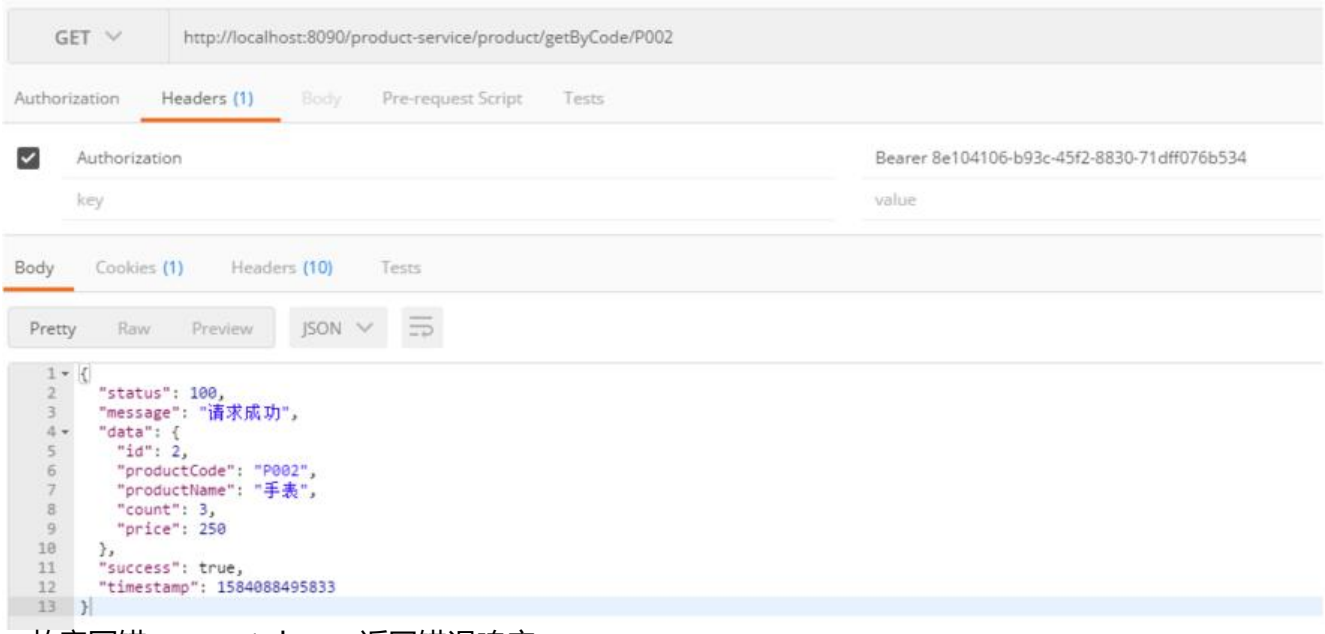
Server | Endpoints | Frames | Threads | Variables | Console

principal

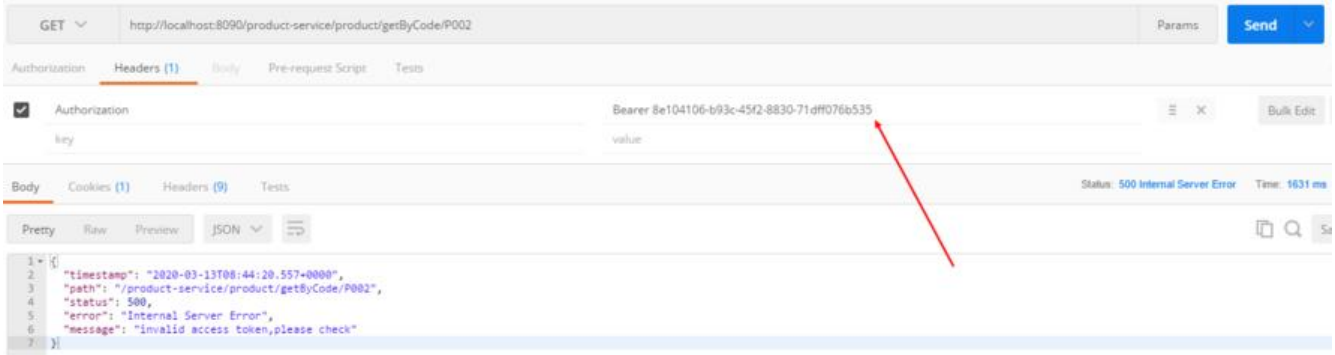
```
principal = {OAuth2Authentication@10405} "org.springframework.security.oauth2.prov... View
  storedRequest = {OAuth2Request@10425}
  userAuthentication = {UsernamePasswordAuthenticationToken@10426} "org.spring... View
    principal = {User@10437} "org.springframework.security.core.userdetails.User@a... View
      password = null
      username = "zhangjian"
      authorities = {Collections$UnmodifiableSet@10442}  size = 1
      accountNonExpired = true
      accountNonLocked = true
      credentialsNonExpired = true
      enabled = true
    credentials = null
    authorities = {Collections$UnmodifiableRandomAccessList@10438}  size = 1
    details = {LinkedHashMap@10439}  size = 4
    authenticated = true
  authorities = {Collections$UnmodifiableRandomAccessList@10427}  size = 1
  details = {OAuth2AuthenticationDetails@10428} "remoteAddress=127.0.0.1, tokenType=Be
  authenticated = true
```

原文链接：SpringCloud Alibaba 微服务实战十四 - SpringCloud Gateway 集成 Oauth2.0

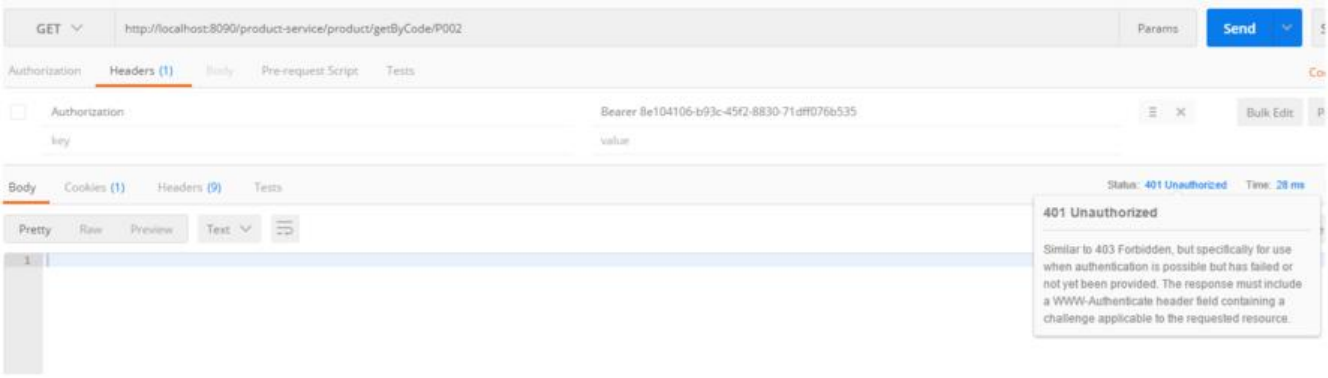
- 返回正常结果



- 故意写错access_token，返回错误响应



- 请求头上去掉access_token，直接返回 401 Unauthorized



总结

通过以上几步我们将SpringCloud Gateway整合好了Oauth2.0，这样我们整个项目也基本完成了，面几期再来对项目进行优化，欢迎持续关注。