链滴

SpringCloud Alibaba 微服务实战十八 - Oauth2.0 自定义授权模式

作者：jianzh5

原文链接：https://ld246.com/article/1595917948936

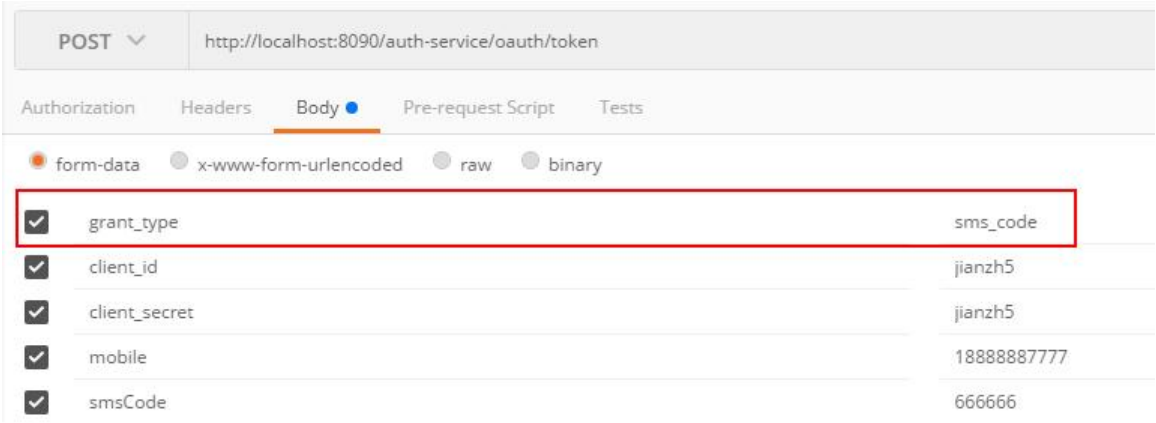
来源网站：链滴

概述

大家都知道在oauth2认证体系中有四种授权模式：

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

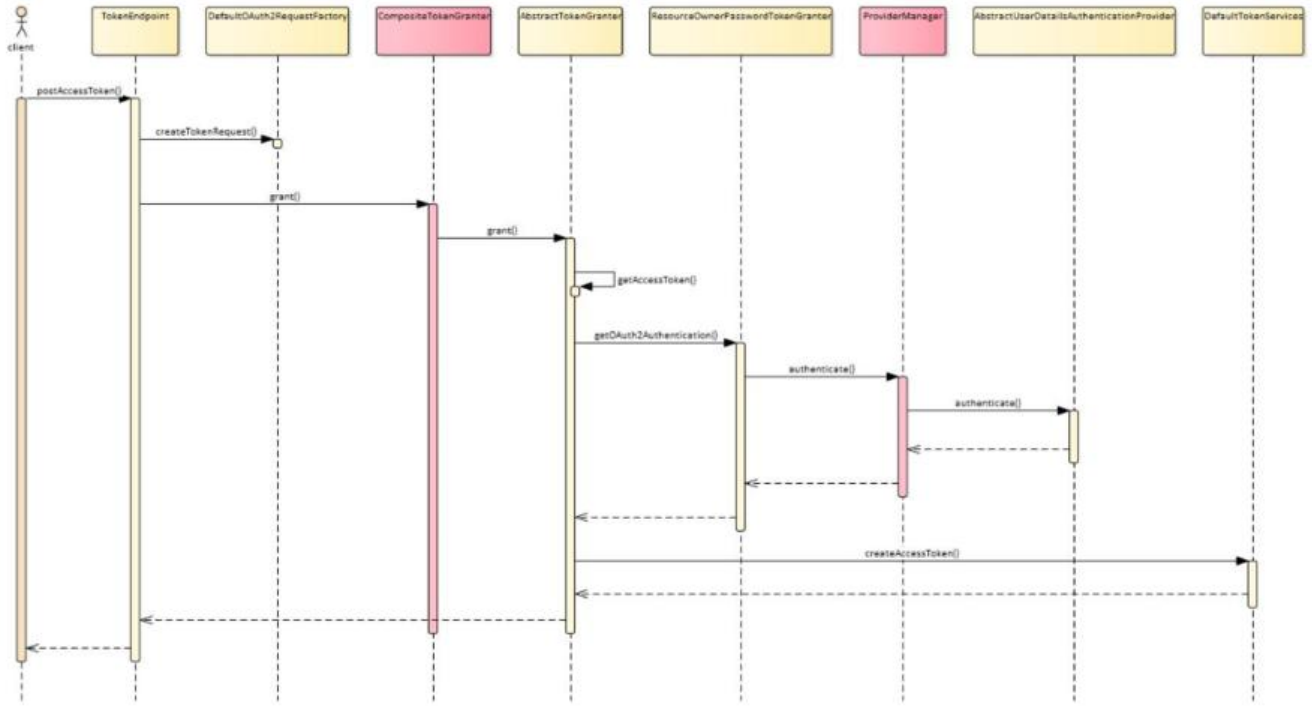
那么如何新增一个自定义的授权模式，比如像下面这样根据手机号和短信验证码进行登录呢？



要自定义授权模式我们得先了解下oauth2.0的整体认证过程，认证入口在 org.springframework.security.oauth2.provider.endpoint.TokenEndpoint#postAccessToken 方法中

```
@RequestMapping(
        value = {"/oauth/token"},
        method = {RequestMethod.POST}
    )
public ResponseEntity<OAuth2AccessToken> postAccessToken(Principal principal, @Requestaram Map<String, String> parameters) throws HttpRequestMethodNotSupportedException {
    ...
}
```

通过阅读源码可以梳理出核心认证逻辑代码的执行顺序（password模式）：



核心源码解读

● TokenEndpoint#postAccessToken(...) 主入口

```
OAuth2AccessToken token =
getTokenGranter().grant(tokenRequest.getGrantType(),
tokenRequest);
```

● CompositeTokenGranter#grant(String grantType,TokenRequest tokenRequest) 负责从所有TokenGranter中根据授权类型找到具体的TokenGranter

```
public class CompositeTokenGranter implements TokenGranter {
    private final List<TokenGranter> tokenGranters;
    ...
    public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
        for (TokenGranter granter : tokenGranters) {
            OAuth2AccessToken grant = granter.grant(grantType, tokenRequest);
            if (grant!=null) {
                return grant;
            }
        }
        return null;
    }
    ...
}
```

那么这里的tokenGranters又是从哪来的呢？答案是oauth2认证服务器端点配置类 AuthorizationServerEndpointsConfigurer

```
public final class AuthorizationServerEndpointsConfigurer {
    ...
    private TokenGranter tokenGranter;
```

```
    public TokenGranter getTokenGranter() {
        return tokenGranter();
    }

    //默认的四种授权模式+Refresh令牌模式
    private List<TokenGranter> getDefaultTokenGranters() {
        ClientDetailsService clientDetails = clientDetailsService();
        AuthorizationServerTokenServices tokenServices = tokenServices();
        AuthorizationCodeServices authorizationCodeServices = authorizationCodeServices();
        OAuth2RequestFactory requestFactory = requestFactory();

        List<TokenGranter> tokenGranters = new ArrayList<TokenGranter>();
        tokenGranters.add(new AuthorizationCodeTokenGranter(tokenServices, authorizationCo
eServices, clientDetails,
                requestFactory));
        tokenGranters.add(new RefreshTokenGranter(tokenServices, clientDetails, requestFactory
);
        ImplicitTokenGranter implicit = new ImplicitTokenGranter(tokenServices, clientDetails, re
uestFactory);
        tokenGranters.add(implicit);
        tokenGranters.add(new ClientCredentialsTokenGranter(tokenServices, clientDetails, requ
stFactory));
        if (authenticationManager != null) {
            tokenGranters.add(new ResourceOwnerPasswordTokenGranter(authenticationManage
, tokenServices,
                    clientDetails, requestFactory));
        }
        return tokenGranters;
    }

    private TokenGranter tokenGranter() {
        if (tokenGranter == null) {
            tokenGranter = new TokenGranter() {
                private CompositeTokenGranter delegate;

                @Override
                public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
                    if (delegate == null) {
                        delegate = new CompositeTokenGranter(getDefaultTokenGranters());
                    }
                    return delegate.grant(grantType, tokenRequest);
                }
            };
        }
        return tokenGranter;
    }
    ...
}
```

可以看到Spring已经把把默认的四种授权模式+刷新令牌的模式的配置在代码中写死了，那么如何让Spring能识别我们自定义的授权模式呢?

我们可以通过配置类覆盖TokenGranter，在里面注入我们自定义的授权模式!

- ProviderManager#authenticate(Authentication authentication)

这个类是提供了认证的实现逻辑和流程，他负责从所有的AuthenticationProvider中找出具体的Provier进行认证

```
public class ProviderManager implements AuthenticationManager, MessageSourceAware,
        InitializingBean {
    ...
    public Authentication authenticate(Authentication authentication)
            throws AuthenticationException {
        Class<? extends Authentication> toTest = authentication.getClass();
        AuthenticationException lastException = null;
        AuthenticationException parentException = null;
        Authentication result = null;
        Authentication parentResult = null;
        boolean debug = logger.isDebugEnabled();
        //遍历所有的providers使用supports方法判断该provider是否支持当前的认证类型
        for (AuthenticationProvider provider : getProviders()) {
            if (!provider.supports(toTest)) {
                continue;
            }

            try {
            //找到具体的provider进行认证
                result = provider.authenticate(authentication);
                if (result != null) {
                    copyDetails(authentication, result);
                    break;
                }
            }
            catch (AccountStatusException | InternalAuthenticationServiceException e) {
                prepareException(e, authentication);
                throw e;
            } catch (AuthenticationException e) {
                lastException = e;
            }
        }
        throw lastException;
    }
    ...
}
```

代码实现（核心代码）

使用手机号登录时先在表单中输入正确的手机号码，请求后端获取验证码。（此时后台服务一般会将

机号码和验证码进行关联，并设置一个较短时间的有效期）

手机获取到验证码后将其输入到表单中即可登录，后端框架将手机号与用户进行关联认证。

短信验证需要两个基础表单数据：手机号码，短信验证码。

本文并没有实现表单登录方式，是使用postman的方式进行认证。使用上图只是让大家对短信认证过有个印象。

SmsCodeAuthenticationToken

```
/**
 * <p>
 * <code>SmsAuthenticationToken</code>
 * </p>
 * Description:
 * 实现手机号登录，参考org.springframework.security.authentication.UsernamePasswordAuthe
ticationToken
 * @author javadaily
 * @date 2020/7/13 8:44
 */
public class SmsCodeAuthenticationToken extends AbstractAuthenticationToken {
    private static final long serialVersionUID = 520L;

    /**
     * 账号主体信息，手机号验证码登录体系中代表 手机号码
     */
    private final Object principal;


    /**
     * 构建未授权的 SmsCodeAuthenticationToken
     * @param mobile 手机号码
     */
    public SmsCodeAuthenticationToken(String mobile) {
        super(null);
        this.principal = mobile;
        setAuthenticated(false);
    }


    /**
     * 构建已经授权的 SmsCodeAuthenticationToken
     */
    public SmsCodeAuthenticationToken(Object principal, Collection<? extends GrantedAuthor
ty> authorities){
        super(authorities);
        this.principal = principal;
        super.setAuthenticated(true);
    }


    @Override
    public Object getCredentials() {
```

```
        return null;
    }

    @Override
    public Object getPrincipal() {
        return this.principal;
    }


    @Override
    public void setAuthenticated(boolean isAuthenticated) {
        if(isAuthenticated){
            throw new IllegalArgumentException("Cannot set this token to trusted - use construct
r which takes a GrantedAuthority list instead");
        }else{
            super.setAuthenticated(false);
        }
    }

    @Override
    public void eraseCredentials() {
        super.eraseCredentials();
    }
}
```

SmsCodeAuthenticationProvider

```
/**
 * Description:
 * 短信登陆鉴权 Provider，要求实现 AuthenticationProvider 接口
 * @author javadaily
 * @date 2020/7/13 13:07
 */
@Log4j2
public class SmsCodeAuthenticationProvider implements AuthenticationProvider{

    private IUserService userService;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationEx
eption {

        SmsCodeAuthenticationToken smsCodeAuthenticationToken = (SmsCodeAuthentication
oken) authentication;
        userService = SpringContextHolder.getBean(IUserService.class);

        String mobile = (String) smsCodeAuthenticationToken.getPrincipal();

        //校验手机号验证码
        checkSmsCode(mobile);

        User user = userService.getUserByMobile(mobile);
        if(null == user){
            throw new BadCredentialsException("Invalid mobile!");
```

```
        }

        //授权通过
        UserDetails userDetails = buildUserDetails(user);
        return new SmsCodeAuthenticationToken(userDetails, userDetails.getAuthorities());
    }

    /**
     * 构建用户认证信息
     * @param user 用户对象
     * @return UserDetails
     */
    private UserDetails buildUserDetails(User user) {
        return new org.springframework.security.core.userdetails.User(
                user.getUsername(),
                user.getPassword(),
                AuthorityUtils.createAuthorityList("ADMIN")) ;
    }

    /**
     * 校验手机号与验证码的绑定关系是否正确
     *  todo 需要根据业务逻辑自行处理
     * @author javadaily
     * @date 2020/7/23 17:31
     * @param mobile 手机号码
     */
    private void checkSmsCode(String mobile) {
        HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.getReq
estAttributes()).getRequest();
        //获取验证码
        String smsCode = request.getParameter("smsCode");
        if(StringUtils.isEmpty(smsCode) || !"666666".equals(smsCode)){
            throw new BadCredentialsException("Incorrect sms code,please check !");
        }
        //todo  手机号与验证码是否匹配
    }

    /**
     * ProviderManager 选择具体Provider时根据此方法判断
     * 判断 authentication 是不是 SmsCodeAuthenticationToken 的子类或子接口
     */
    @Override
    public boolean supports(Class<?> authentication) {
        return SmsCodeAuthenticationToken.class.isAssignableFrom(authentication);
    }
}
```

短信验证码模式认证实现类，需要实现AuthenticationProvider，通过 supports方法会被ProviderManager选中成为具体的认证实现类。

手机号码与短信的关联关系需要根据自己业务场景实现，这里直接先写死。

配置类SmsCodeSecurityConfig

```
@Component
public class SmsCodeSecurityConfig extends SecurityConfigurerAdapter<DefaultSecurityFilte
Chain, HttpSecurity> {
    /**
     * 短信验证码配置器
     *  所有的配置都可以移步到WebSecurityConfig
     *  builder.authenticationProvider() 相当于 auth.authenticationProvider();
     *  使用外部配置必须要在WebSecurityConfig中用http.apply(smsCodeSecurityConfig)将配置
入进去
     * @param builder
     * @throws Exception
     */
    @Override
    public void configure(HttpSecurity builder) throws Exception {
        //注入SmsCodeAuthenticationProvider
        SmsCodeAuthenticationProvider smsCodeAuthenticationProvider = new SmsCodeAuthe
ticationProvider();
        builder.authenticationProvider(smsCodeAuthenticationProvider);
    }
}
```

此类主要实现SmsCodeAuthenticationProvider的注入，否则ProviderManager无法选到SmsCode uthenticationProvider。

SmsCodeTokenGranter

```
/**
 * 扩展认证模式
 * @author javadaily
 * @date 2020/7/14 8:31
 */
public class SmsCodeTokenGranter extends AbstractTokenGranter{

    private static final String GRANT_TYPE = "sms_code";

    private final AuthenticationManager authenticationManager;

    public SmsCodeTokenGranter(AuthenticationManager authenticationManager, Authorizati
nServerTokenServices tokenServices, ClientDetailsService clientDetailsService, OAuth2Request
actory requestFactory) {
        this(authenticationManager, tokenServices, clientDetailsService, requestFactory, GRANT_
YPE);
    }

    protected SmsCodeTokenGranter(AuthenticationManager authenticationManager, Authori
ationServerTokenServices tokenServices, ClientDetailsService clientDetailsService, OAuth2Req
estFactory requestFactory, String grantType) {
        super(tokenServices, clientDetailsService, requestFactory, grantType);
        this.authenticationManager = authenticationManager;
    }

    @Override
    protected OAuth2Authentication getOAuth2Authentication(ClientDetails client, TokenRequ
st tokenRequest) {
```

```
        Map<String, String> parameters = new LinkedHashMap(tokenRequest.getRequestPara
eters());
        String mobile = parameters.get("mobile");

        Authentication userAuth = new SmsCodeAuthenticationToken(mobile);

        ((AbstractAuthenticationToken)userAuth).setDetails(parameters);

        try {
            userAuth = this.authenticationManager.authenticate(userAuth);
        } catch (AccountStatusException ex) {
            throw new InvalidGrantException(ex.getMessage());
        } catch (BadCredentialsException ex) {
            throw new InvalidGrantException(ex.getMessage());
        }

        if (userAuth != null && userAuth.isAuthenticated()) {
            OAuth2Request storedOAuth2Request = this.getRequestFactory().createOAuth2Reque
t(client, tokenRequest);
            return new OAuth2Authentication(storedOAuth2Request, userAuth);
        } else {
            throw new InvalidGrantException("Could not authenticate mobile: " + mobile);
        }
    }
}
```

继承AbstractTokenGranter扩展认证模式sms_code，需要将其添加到Spring中并通过grantType被中。

配置类TokenGranterConfig

通过前面几步自定义认证的基础逻辑都已实现，接下来需要将我们的短信认证模式添加到Spring中，要参考 org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationerverEndpointsConfigurer#getDefaultTokenGranters() 实现。

```
/**
 *参考实现：org.springframework.security.oauth2.config.annotation.web.configurers.Authoriza
ionServerEndpointsConfigurer#getDefaultTokenGranters()
 * @author javadaily
 * @date 2020/7/14 8:38
 */
@Configuration
public class TokenGranterConfig {
    @Autowired
    private ClientDetailsService clientDetailsService;

    private TokenGranter tokenGranter;

    @Autowired
    private TokenStore tokenStore;

    @Autowired
    TokenEnhancer tokenEnhancer;
```

```
    @Autowired
    private AuthenticationManager authenticationManager;

    private AuthorizationServerTokenServices tokenServices;

    private boolean reuseRefreshToken = true;

    private AuthorizationCodeServices authorizationCodeServices;

    @Autowired
    private UserDetailsService userDetailsService;

    @Bean
    public TokenGranter tokenGranter(){
        if(null == tokenGranter){
            tokenGranter = new TokenGranter() {
                private CompositeTokenGranter delegate;

                @Override
                public OAuth2AccessToken grant(String grantType, TokenRequest tokenRequest) {
                    if(delegate == null){
                        delegate = new CompositeTokenGranter(getDefaultTokenGranters());
                    }
                    return delegate.grant(grantType,tokenRequest);
                }
            };
        }
        return tokenGranter;
    }

    private List<TokenGranter> getDefaultTokenGranters() {
        AuthorizationServerTokenServices tokenServices = tokenServices();
        AuthorizationCodeServices authorizationCodeServices = authorizationCodeServices();
        OAuth2RequestFactory requestFactory = requestFactory();

        List<TokenGranter> tokenGranters = new ArrayList();
        //授权码模式
        tokenGranters.add(new AuthorizationCodeTokenGranter(tokenServices, authorizationCo
eServices, clientDetailsService, requestFactory));
        //refresh模式
        tokenGranters.add(new RefreshTokenGranter(tokenServices, clientDetailsService, request
actory));
        //简化模式
        ImplicitTokenGranter implicit = new ImplicitTokenGranter(tokenServices, clientDetailsSer
ice, requestFactory);
        tokenGranters.add(implicit);
        //客户端模式
        tokenGranters.add(new ClientCredentialsTokenGranter(tokenServices, clientDetailsServic
, requestFactory));

        if (authenticationManager != null) {
            //密码模式
            tokenGranters.add(new ResourceOwnerPasswordTokenGranter(authenticationManage
, tokenServices, clientDetailsService, requestFactory));
```

```
            //短信验证码模式
            tokenGranters.add(new SmsCodeTokenGranter(authenticationManager, tokenServices,
clientDetailsService, requestFactory));
        }

        return tokenGranters;
    }

    private AuthorizationServerTokenServices tokenServices() {
        if (tokenServices != null) {
            return tokenServices;
        }
        this.tokenServices = createDefaultTokenServices();
        return tokenServices;
    }

    private AuthorizationServerTokenServices createDefaultTokenServices() {
        DefaultTokenServices tokenServices = new DefaultTokenServices();
        tokenServices.setTokenStore(tokenStore);
        tokenServices.setSupportRefreshToken(true);
        tokenServices.setReuseRefreshToken(reuseRefreshToken);
        tokenServices.setClientDetailsService(clientDetailsService);
        tokenServices.setTokenEnhancer(tokenEnhancer);
        addUserDetailsService(tokenServices, this.userDetailsService);
        return tokenServices;
    }

    /**
     * 添加预身份验证
     * @param tokenServices
     * @param userDetailsService
     */
    private void addUserDetailsService(DefaultTokenServices tokenServices, UserDetailsService
userDetailsService) {
        if (userDetailsService != null) {
            PreAuthenticatedAuthenticationProvider provider = new PreAuthenticatedAuthenticat
onProvider();
            provider.setPreAuthenticatedUserDetailsService(new UserDetailsByNameServiceWrap
er<PreAuthenticatedAuthenticationToken>(userDetailsService));
            tokenServices.setAuthenticationManager(new ProviderManager(Arrays.<Authenticati
nProvider>asList(provider)));
        }
    }

    /**
     * OAuth2RequestFactory的默认实现，它初始化参数映射中的字段，
     * 验证授权类型(grant_type)和范围(scope)，并使用客户端的默认值填充范围(scope)（如果缺少
些值）。
     */
    private OAuth2RequestFactory requestFactory() {
        return new DefaultOAuth2RequestFactory(clientDetailsService);
    }

    /**
```

```
     * 授权码API
     * @return
     */
    private AuthorizationCodeServices authorizationCodeServices() {
        if (this.authorizationCodeServices == null) {
            this.authorizationCodeServices = new InMemoryAuthorizationCodeServices();
        }
        return this.authorizationCodeServices;
    }
}
```

修改认证服务器配置 AuthorizationServerConfig

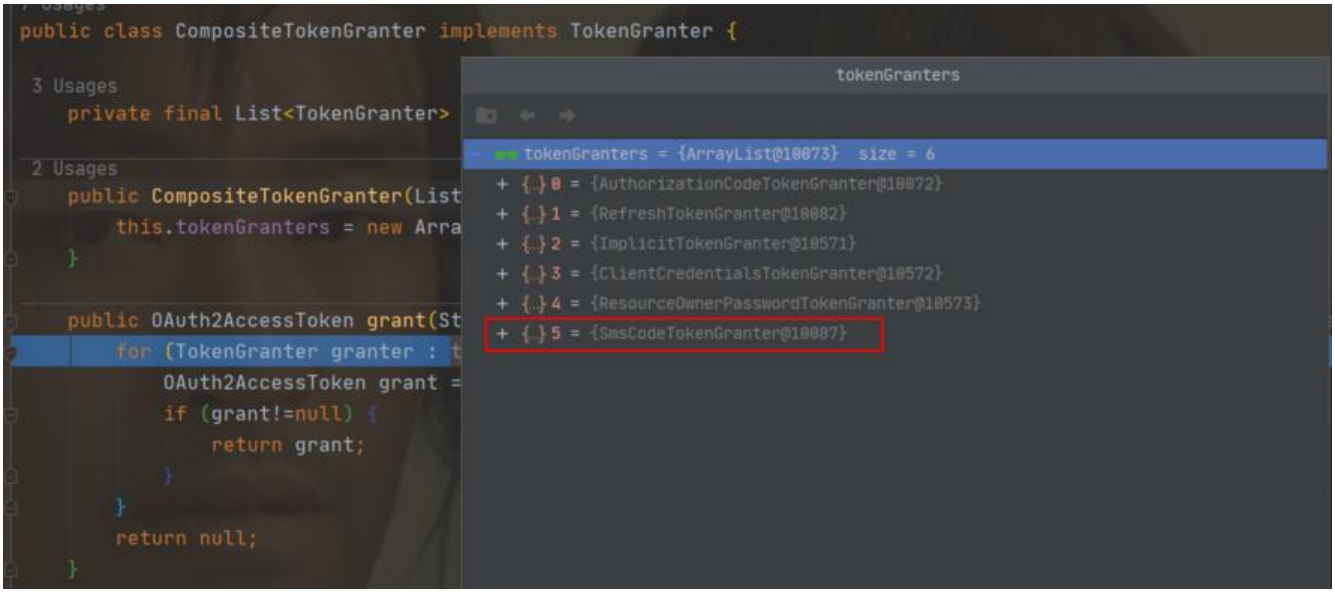
在上面的TokenGranterConfig中已经创建了AuthorizationServerTokenServices，所以我们可以将AthorizationServerConfig中的tokenServices功能删除，然后在方法 configure(AuthorizationServerndpointsConfigurer endpoints) 中注入 tokenGranter 即可

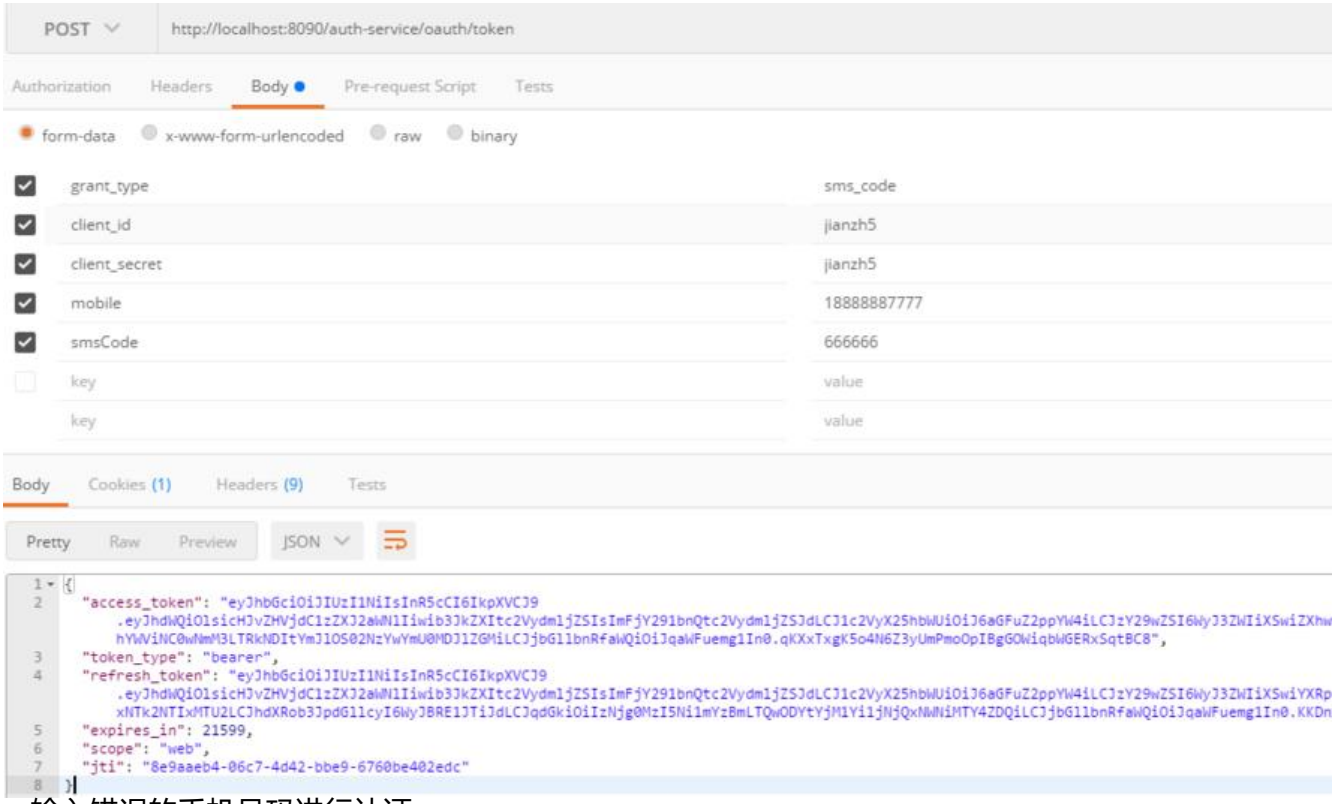
```
@Configuration
@EnableAuthorizationServer
@RequiredArgsConstructor(onConstructor = @__(@Autowired))
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
    @Autowired
    private  TokenGranter tokenGranter;
    ...
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception
{
        endpoints.tokenGranter(tokenGranter);
    }
    ...
}
```

测试

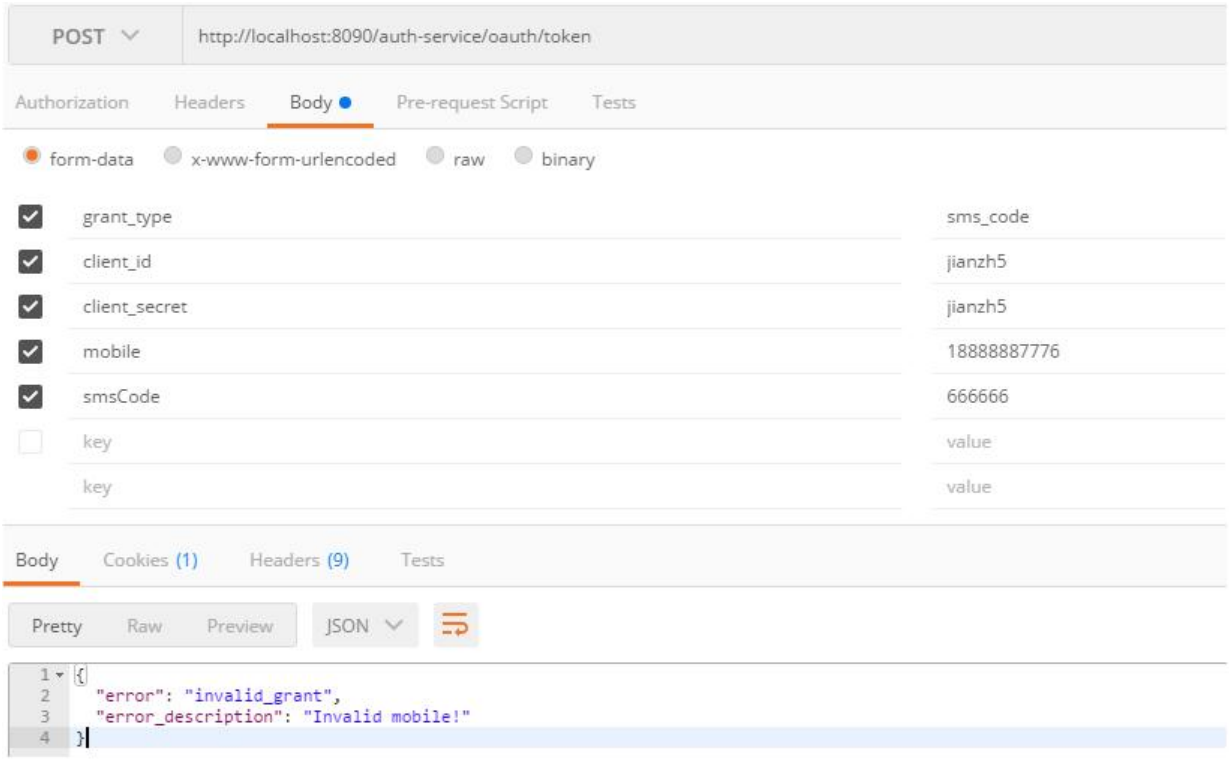
- 正常测试

通过debug模式可以看到SmsCodeTokenGranter已经被加入Spring，并能正常返回jwt token了。

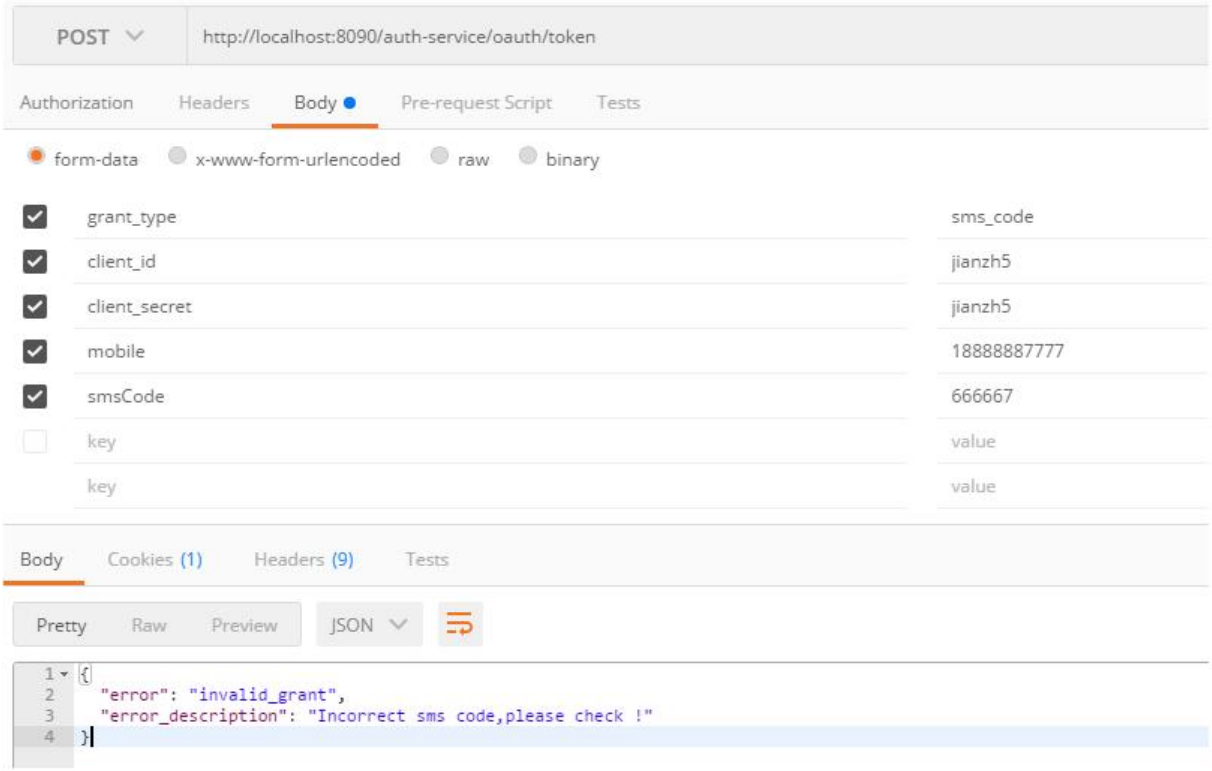




- 输入错误的手机号码进行认证



- 输入错误的短信验证码进行认证



本篇文章是SpringCloud alibaba实战系列文章的第20篇，如果大家对之前的文章感兴趣可以移步 htt://javadaily.cn/tags/SpringCloud 查看

原文链接：SpringCloud Alibaba 微服务实战十八 - Oauth2.0 自定义授权模式