

# 一:微服务 & 微服务架构

## 1: 单体架构 VS 微服务架构

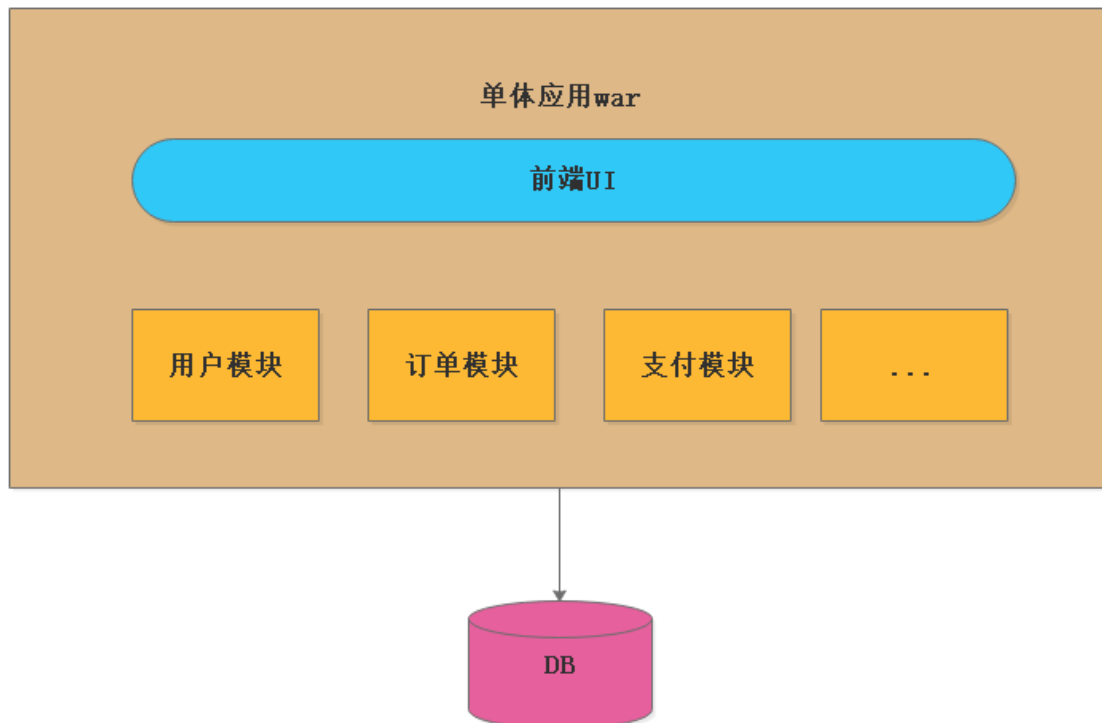
### 1.1)从单体架构说起

一个工程对应一个归档包(war), 这个war包 包含了该工程的所有功能。我们成为这种应用为单体应用, 也就是我们常说的单体架构(**一个war包打天下**)。

具体描述: 就是在我们的一个war包种, 聚集了各种功能以及资源, 比如JSP

JS,CSS等。而业务种包含了我们的用户模块, 订单模块, 支付模块等等.

### 1.2)单体架构图



### 1.3)单体架构优缺点总结

#### 优点:

①: 架构简单明了, 没有“花里胡哨”的问题需要解决。

②:开发，测试，部署简单（尤其是运维人员 睡着都会笑醒）

缺点:

①：随着业务扩展，代码越来越复杂，代码质量参差不齐(开发人员的水平不一),会让你每次提交代码，修改每一个小bug都是心惊胆战的。

②:部署慢(由于单体架构，功能复杂) 能想像下一个来自200W+代码部署的速度  
(15分钟)

③:扩展成本高，根据单体架构图 假设用户模块是一个CPU密集型的模块(涉及到大量的运算)那么我们需要替换更加牛逼的CPU，而我们的订单模块是一个IO密集模块（涉及大量的读写磁盘）,那我们需要替换更加牛逼的内存以及高效的磁盘。但是我们的单体架构上 无法针对单个功能模块进行扩展，那么就需要替换更牛逼的CPU 更牛逼的内存 更牛逼的磁盘 价格蹭蹭的往上涨。

④:阻碍了新技术的发展。。。。。比如我们的web架构模块 从struts2迁移到springboot，那么就会成为灾难性

## 1.4) 微服务以及微服务架构



### 1.4.1)微服务的定义

①：英文:<https://martinfowler.com/articles/microservices.html>

②：中文:<http://blog.cuicc.com/blog/2015/07/22/microservices>

1.4.2) 微服务核心就是把传统的单机应用，**根据业务将单机应用拆分为一个一个的服务**，彻底的解耦，**每一个服务都是提供特定的功能**，一个服务只做一件事,类似进程，每个服务都能够单独部署，甚至可以拥有自己的数据库。这样的一个一个的小服务就是 微服务。

①: 比如传统的单机电商应用, tulingshop 里面有 **订单/支付/库存/物流/积分等模块**(理解为service)

②:我们根据 业务模型来拆分,可以拆分为 **订单服务，支付服务，库存服务，物流服务，积分服务**

**\*③\*若不拆分的时候，我的非核心业务积分模块 出现了重大bug 导致系统内存溢出，导致整个服务宕机。**

**若拆分之后，只是说我的积分微服务不可用，我的整个系统核心功能还是能使用**

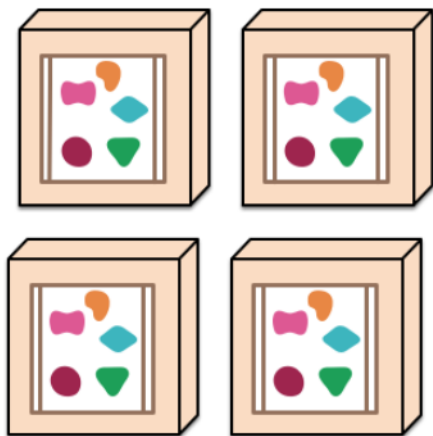
一个单体应用程序把它所有的功能放在一个单一进程中...



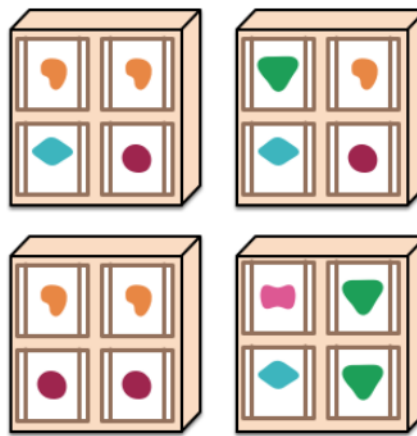
一个微服务架构把每个功能元素放进一个独立的服务中...



...并且通过在多个服务器上复制这个单体进行扩展

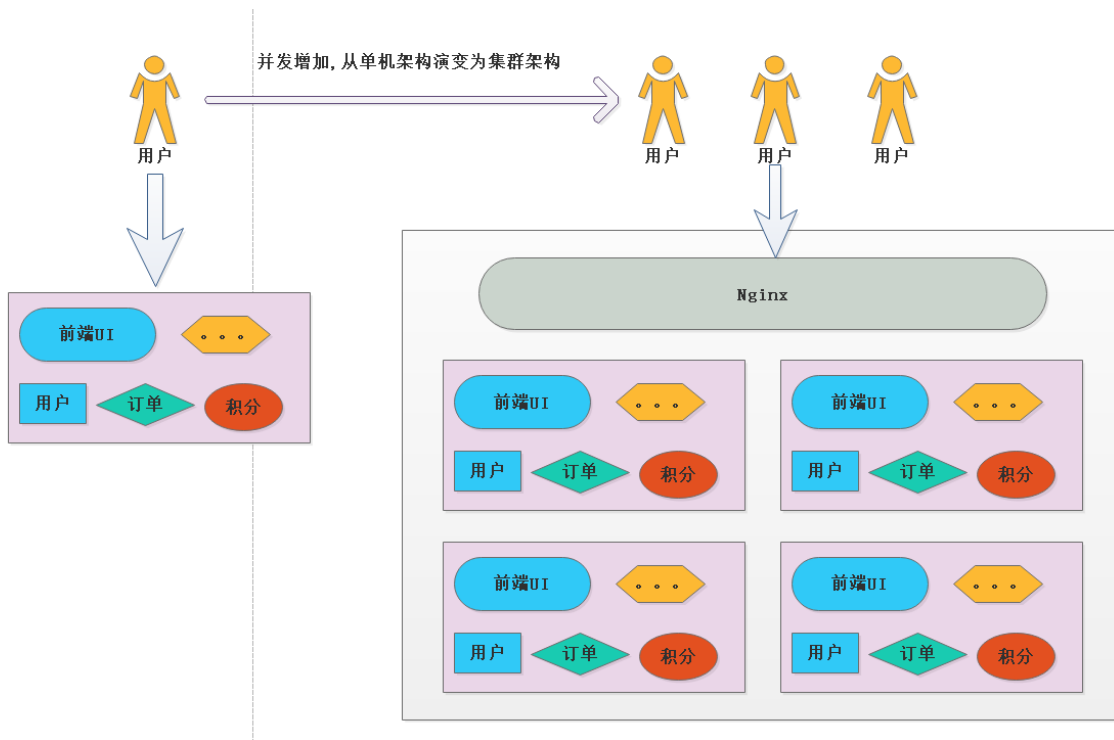


...并且通过跨服务器分发这些服务进行扩展，只在需要时才复制。

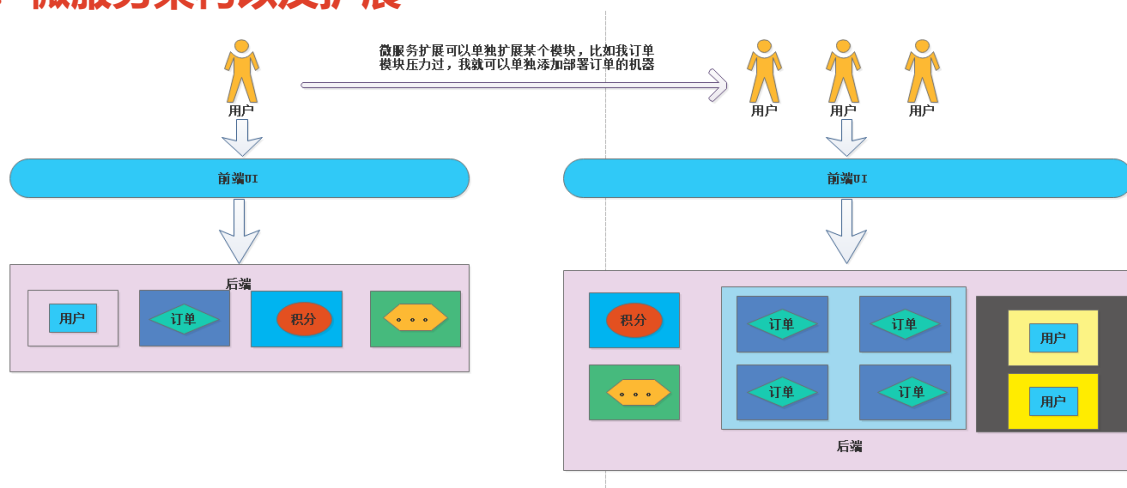


### 1.4.3)单机架构扩展与微服务扩展

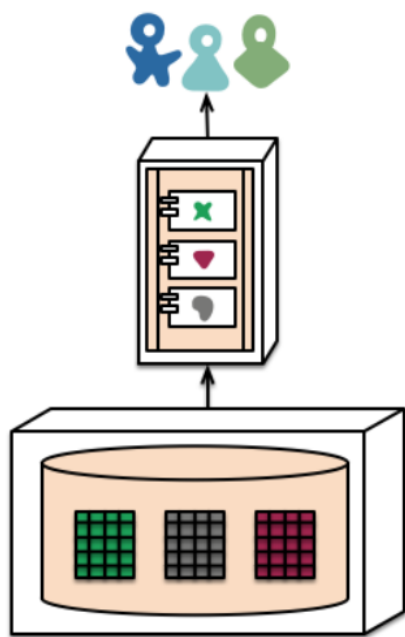
#### ①：单机架构扩展



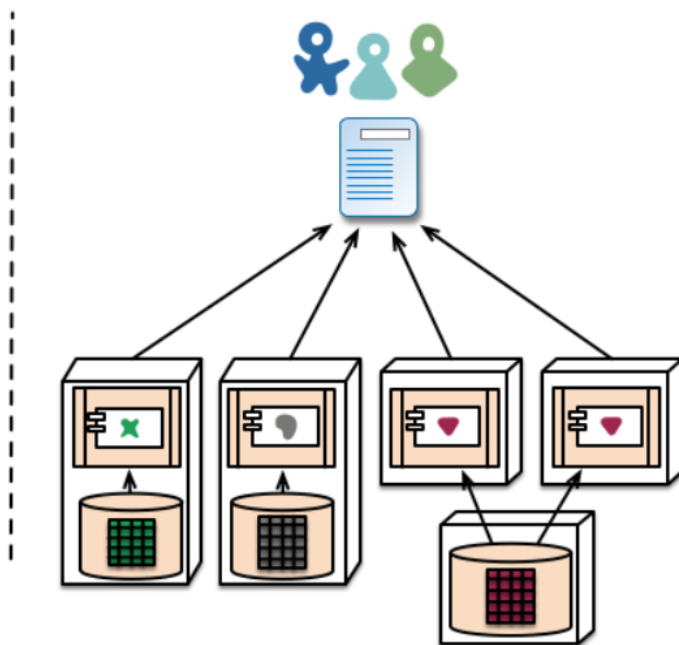
## ②：微服务架构以及扩展



## ③：微服务数据存储



单体 - 单一数据库



微服务 - 应用程序数据库

#### 1.4.4)微服务架构是什么？

**微服务架构是一个架构风格, 提倡**

- ①:将一个单一应用程序开发为一组小型服务.
- ②:每个服务运行在自己的进程中
- ③:服务之间通过轻量级的通信机制(http rest api)
- ④:每个服务都能够独立的部署
- ⑤:每个服务甚至可以拥有自己的数据库

#### 1.4.5) 微服务以及微服务架构的是二个完全不同的概念。

**微服务**强调的是服务的大小和对外提供的单一功能，而**微服务架构**是指把一个一个一个的微服务组合管理起来，对外提供一套完整的服务。

#### 1.4.6)微服务的优缺点

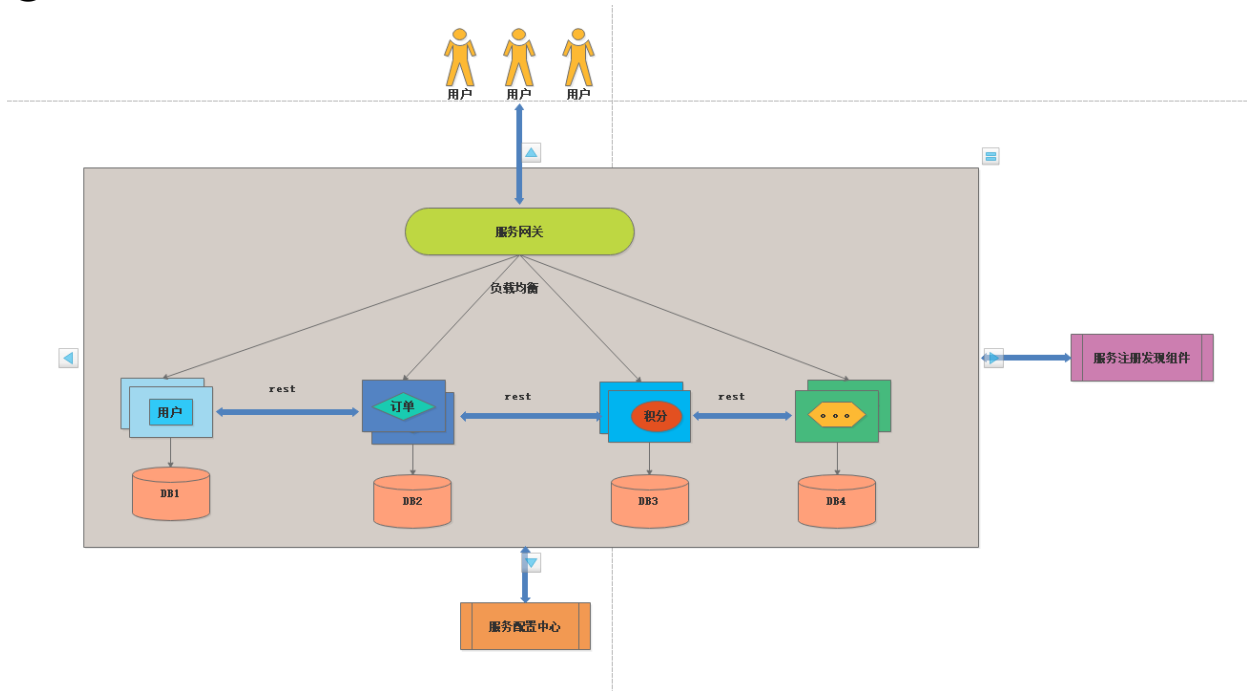
**A:优点:**

- ①: 每个服务足够小,足够内聚, 代码更加容易理解,专注一个业务功能点(对比传统应用, 可能改几行代码 需要了解整个系统)
- ②: 开发简单, 一个服务只干一个事情。(加入你做支付服务, 你只要了解支付相关代码就可以了)
- ③: 微服务能够被2-5个人的小团队开发, 提高效率
- ④: 按需伸缩
- ⑤: 前后段分离, 作为java开发人员, 我们只要关系后端接口的安全性以及性能, 不要去关注页面的人机交互(H5工程师)根据前后端接口协议, 根据入参, 返回json的回参

⑥:一个服务可用拥有自己的数据库。也可以多个服务连接同一个数据库.

### 缺点:

- ①:增加了运维人员的工作量，以前只要部署一个war包，现在可能需要部署成百上千个war包 (k8s+docker+jenkins )
- ②: 服务之间相互调用，增加通信成本
- ③:数据一致性问题(分布式事物问题)
- ④:系统能监控等,问题定位.....



### 1.4.6) 微服务的适用场景

#### A: 合适

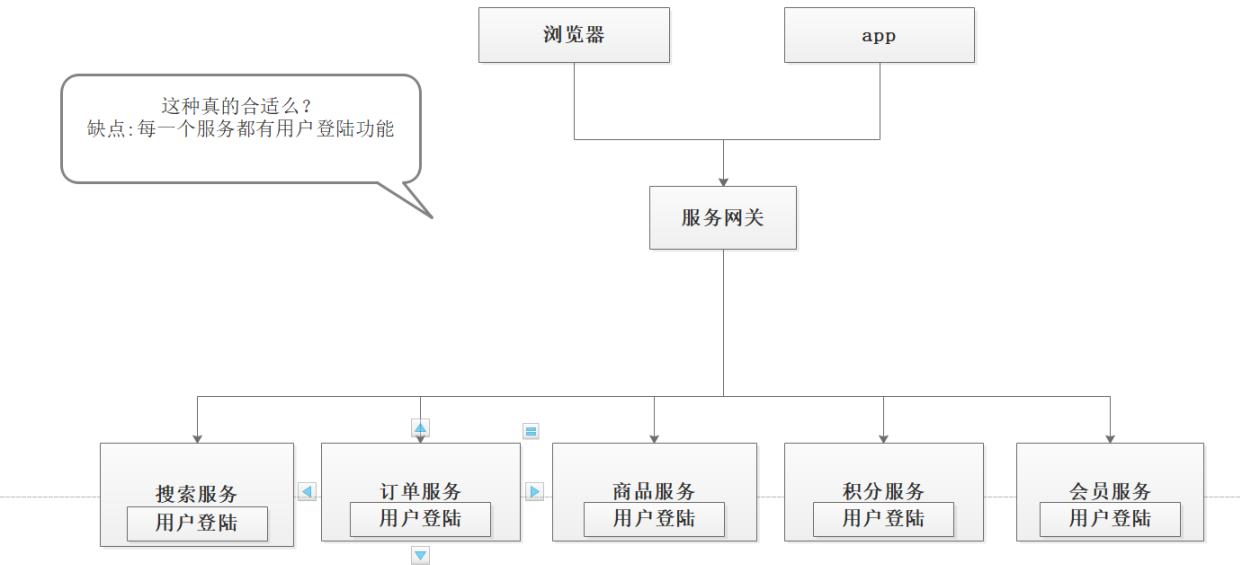
- ①:大型复杂的项目.....(来自单体架构200W行代码的恐惧)
- ②:快速迭代的项目.....(来自一天一版的恐惧)
- ③:并发高的项目.....(考虑弹性伸缩扩容的恐惧)

#### B: 不合适

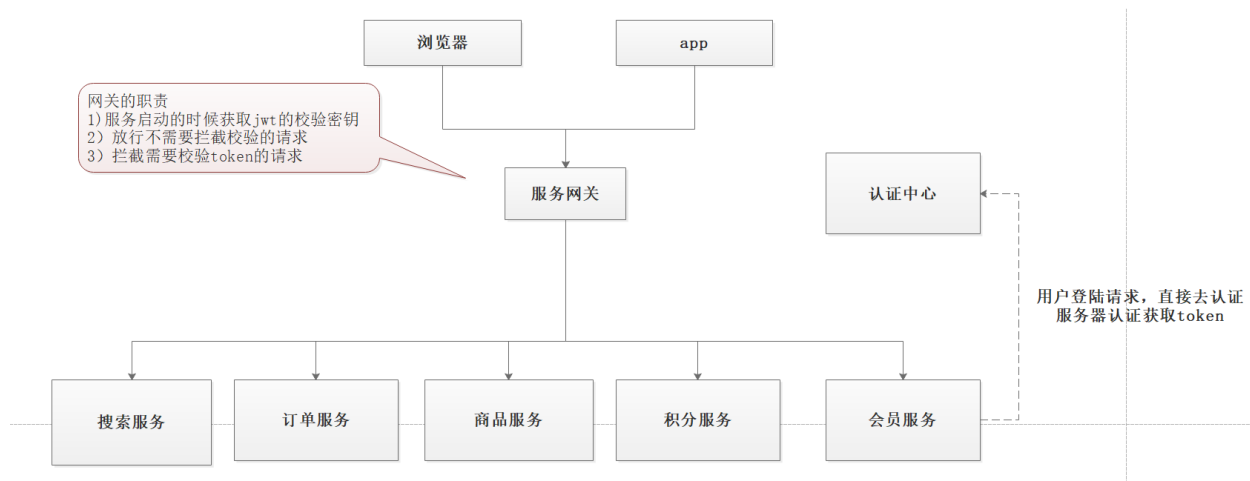
- ①: 业务稳定，就是修修bug，改改数据
- ②: 迭代周期长 发版频率 一二月一次.

## 二:手把手搭建授权中心

名称	服务IP	端口	描述
数据库	192.168.159.8	3306	数据库
注册中心(Nacos)	192.168.158.8	8848	注册中心
配置中心(nacos)	192.168.158.8	8848	配置中心
tulingmall-authcenter	localhost auth.tuling.com	8899	认证中心
tulingmall-gateway	localhost gateway.tuling.com	8888	服务网关
tulingmall-member	localhost member.tuling.com	8877	会员服务
tulingmall-product	localhost product.tuling.com	8866	商品服务
tulingmall-coupons	localhost coupons.tuling.com	8855	积分服务
tulingmall-order	localhost order.tuling.com	8844	订单服务



优化架构:



## 2.1) 首要任务 搭建授权中心 tulingmall-authcenter

### 添加依赖:

```

1  <dependency>
2    <groupId>org.springframework.boot</groupId>
3    <artifactId>spring-boot-starter</artifactId>
4  </dependency>
5
6
7  <dependency>
8    <groupId>org.springframework.boot</groupId>
9    <artifactId>spring-boot-starter-web</artifactId>
10 </dependency>
11
12 <!--nacos的依赖-->
13 <dependency>
14   <groupId>com.alibaba.cloud</groupId>
15   <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
16 </dependency>
17
18 <!--SpringSecurity的依赖-->
19 <dependency>
20   <groupId>org.springframework.cloud</groupId>
21   <artifactId>spring-cloud-starter-oauth2</artifactId>
22 </dependency>
23
24 <dependency>
25   <groupId>com.tuling</groupId>
26   <artifactId>tulingmall-mbg</artifactId>
27 </dependency>
28

```



```

29 <dependency>
30   <groupId>org.springframework.boot</groupId>
31   <artifactId>spring-boot-configuration-processor</artifactId>
32   <optional>true</optional>
33 </dependency>

```

### 2.1.1) 因为我们的授权中心需要去进行用户认证，需要连接数据库的配置

这里我们暂时写到本地(后续会移植到配置中心的)

```

1 spring:
2   datasource:
3     url: jdbc:mysql://192.168.159.8:3306/micromall?
serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=UTF-8
4     username: root
5     password: Zw726515@
6   druid:
7     initial-size: 5 #连接池初始化大小
8     min-idle: 10 #最小空闲连接数
9     max-active: 20 #最大连接数
10  web-stat-filter:
11    exclusions: "*.js,*.gif,*.jpg,*.png,*.css,*.ico,/druid/*" #不统计这些请求
数据
12    stat-view-servlet: #访问监控网页的登录用户名和密码
13    login-username: druid
14    login-password: druid

```

### 2.1.2)授权中心 服务器配置类

**第一步:**添加@EnableAuthorizationServer 标识为认证服务器

**第二步:**继承抽象类AuthorizationServerConfigurerAdapter(里面有配置需要配)

```

1 @Configuration
2 @EnableAuthorizationServer
3 public class TulingAuthServerConfig extends AuthorizationServerConfigurer
Adapter{
4
5   //现在里面什么都没有配置
6 }

```

**第三步:** 认证服务器需要知道 我可以办法token给哪些第三方用户

我们这里是采用基于数据库配置的方式存储第三方客户端的信息 你需要添加一个表

client_id	resource_ids	client_secret	scope	authorized_grant_types
api-gateway	order-service,product-service,member-service	\$2a\$10\$vrjbiA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/BoHtshii5Judp9G	read,write	password
member-service	(Null)	\$2a\$10\$vrjbiA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/BoHtshii5Judp9G	read,write,test	password,refresh_token
order_app	order-service	\$2a\$10\$vrjbiA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/BoHtshii5Judp9G	read	password
portal_app	order-service,product-service,api-gateway,authorize-server	\$2a\$10\$E.QABsfckMmIX2W81iqRbulFnVfJoGD1nWuYZjOk8ZCPuFqFY2Pu	read,write	password,authorization_code
product_app	product-service	\$2a\$10\$vrjbiA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/BoHtshii5Judp9G	read,write	password

```

1 CREATE TABLE `oauth_client_details` (
2   `client_id` varchar(256) CHARACTER SET utf8 NOT NULL,
3   `resource_ids` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
4   `client_secret` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
5   `scope` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
6   `authorized_grant_types` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
7   `web_server_redirect_uri` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
8   `authorities` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
9   `access_token_validity` int(11) DEFAULT NULL,
10  `refresh_token_validity` int(11) DEFAULT NULL,
11  `additional_information` varchar(4096) CHARACTER SET utf8 DEFAULT NULL,
12  `autoapprove` varchar(256) CHARACTER SET utf8 DEFAULT NULL,
13  PRIMARY KEY (`client_id`)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

```

1 @Configuration
2 @EnableAuthorizationServer
3 public class TulingAuthServerConfig extends AuthorizationServerConfigurer
4 Adapter{
5   /**
6    * 方法实现说明:认证服务器能够给哪些 客户端颁发token 我们需要把客户端的配置 存
7    储到
8    * 数据库中 可以基于内存存储和db存储
9    * @author:smlz
10   * @return:
11   * @exception:
12   * @date:2020/1/15 20:18
13   */
14   @Override
15   public void configure(ClientDetailsServiceConfigurer clients) throws Ex
16   ception {
17     clients.withClientDetails(clientDetails());
18   }
19 }
20 /**
21  * 方法实现说明:用于查找我们第三方客户端的组件 主要用于查找 数据库表 oauth_cli
22  ent_details

```

```
19  * @author:smlz
20  * @return:
21  * @exception:
22  * @date:2020/1/15 20:19
23  */
24  @Bean
25  public ClientDetailsService clientDetails() {
26  return new JdbcClientDetailsService(dataSource);
27  }
28 }
```

**第四步: 授权服务器颁发的token 基于什么方式存储的配置**  
**但是我们的token密钥 在生产上不能使用普通的 我们需要生成密钥**  
**我们使用jdk自动的工具生成**

**命令格式**

**keytool**

**-genkey**

**-alias tomcat(别名)**

**-keypass 123456(别名密码)**

**-keyalg RSA(生证书的算法名称, RSA是一种非对称加密算法)**

**-keysize 1024(密钥长度,证书大小)**

**-validity 365(证书有效期, 天单位)**

**-keystore W:/tomcat.keystore(指定生成证书的位置和证书名称)**

**-storepass 123456(获取keystore信息的密码)**

**- storetype (指定密钥仓库类型)**

**命令:**

**keytool -genkeypair -alias jwt -keyalg RSA -keysize 2048 -  
keystore D:/jwt/jwt.jks**

```


C:\Users\zhuwei>keytool -genkeypair -alias jwt -keyalg RSA -keysize 2048 -keystore D:/jwt/jwt.key
输入密钥库口令:
再次输入新口令:
您的名字与姓氏是什么?
[Unknown]: smlz
您的组织单位名称是什么?
[Unknown]: tuling
您的组织名称是什么?
[Unknown]: tuling
您所在的城市或区域名称是什么?
[Unknown]: changsha
您所在的省/市/自治区名称是什么?
[Unknown]: hunan
该单位的双字母国家/地区代码是什么?
[Unknown]: china
CN=smlz, OU=tuling, O=tuling, L=changsha, ST=hunan, C=china是否正确?
[否]: y

输入 <jwt> 的密钥口令
(如果和密钥库口令相同, 按回车):
再次输入新口令:

C:\Users\zhuwei>_

```

> study (D:) > jwt

名称	修改日期	类型	大小
 jwt.key	2020/2/17 17:06	KEY 文件	3 KB

## 把密钥copy到认证中心目录下

### ①:项目中jwt的配置

```

1  tuling:
2  jwt:
3  keyPairName: jwt.jks
4  keyPairAlias: jwt
5  keyPairSecret: 123456
6  keyPairStoreSecret: 123456
7  /**
8  * @vlog: 高于生活, 源于生活
9  * @desc: 类的描述: jwt 证书配置
10 * @author: smlz
11 * @createDate: 2020/1/21 21:56
12 * @version: 1.0
13 */
14 @Data
15 @ConfigurationProperties(prefix = "tuling.jwt")
16 public class JwtCAProperties {
17
18     /**
19     * 证书名称
20     */
21     private String keyPairName;

```

```

22
23
24  /**
25   * 证书别名
26   */
27   private String keyPairAlias;
28
29  /**
30   * 证书私钥
31   */
32   private String keyPairSecret;
33
34  /**
35   * 证书存储密钥
36   */
37   private String keyPairStoreSecret;
38  }

```

## ②:Jwt Token的配置

```

1  @Autowired
2  private JwCProperties jwtCProperties;
3
4
5  /**
6   * 方法实现说明:我们颁发的token通过jwt存储
7   * @author:smlz
8   * @return:
9   * @exception:
10   * @date:2020/1/21 21:49
11   */
12  @Bean
13  public TokenStore tokenStore(){
14      return new JwtTokenStore(jwtAccessTokenConverter());
15  }
16
17  @Bean
18  public JwtAccessTokenConverter jwtAccessTokenConverter() {
19      JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
20      //jwt的密钥
21      converter.setKeyPair(keyPair());

```

```

22     return converter;
23 }
24
25 /**
26  *jwt密钥
27  */
28 @Bean
29 public KeyPair keyPair() {
30     KeyStoreKeyFactory keyStoreKeyFactory = new KeyStoreKeyFactory(new Class
31     sPathResource(jwtCAProperties.getKeyPairName()), jwtCAProperties.getKeyPair
32     Secret().toCharArray());
33
34     return keyStoreKeyFactory.getKeyPair(jwtCAProperties.getKeyPairAlias(),
35     jwtCAProperties.getKeyPairStoreSecret().toCharArray());
36 }
37
38 /**
39  * token的增强器 根据自己业务添加字段到Jwt中
40  */
41 @Bean
42 public TulingTokenEnhancer tulingTokenEnhancer() {
43     return new TulingTokenEnhancer();
44 }
45
46 //需要自己写一个token增强器的类
47 /**
48  * @vlog: 高于生活，源于生活
49  * @desc: 类的描述:jwt自定义增强器(根据自己的业务需求添加非敏感字段)
50  * @author: smlz
51  * @createDate: 2020/1/21 21:51
52  * @version: 1.0
53  */
54 public class TulingTokenEnhancer implements TokenEnhancer {
55     @Override
56     public OAuth2AccessToken enhance(OAuth2AccessToken accessToken, OAuth2A
57     uthentication authentication) {
58         MemberDetails memberDetails = (MemberDetails) authentication.getPrincip
59         al();

```

```

59
60     final Map<String, Object> additionalInfo = new HashMap<>();
61
62     final Map<String, Object> retMap = new HashMap<>();
63
64     //todo 这这里暴露memberId到Jwt的令牌中,后期可以根据自己的业务需要 进行添加字
    段
65     additionalInfo.put("memberId", memberDetails.getUmsMember().getId());
66
    additionalInfo.put("nickName", memberDetails.getUmsMember().getNickname());
67     additionalInfo.put("integration", memberDetails.getUmsMember().getIntegr
    ation());
68
69     retMap.put("additionalInfo", additionalInfo);
70
71     ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(retMa
    p);
72
73     return accessToken;
74 }
75 }

```

## 第五步:认证服务器的配置

### 主要配置是把处理我们的token

```

1  /**
2   * 方法实现说明:授权服务器的配置的配置
3   * @author: smlz
4   * @return:
5   * @exception:
6   * @date: 2020/1/15 20:21
7   */
8  @Override
9  public void configure(AuthorizationServerEndpointsConfigurer endpoints) t
    hrows Exception {
10
11     TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
12     tokenEnhancerChain.setTokenEnhancers(Arrays.asList(tulingTokenEnhancer(), jw
    tAccessTokenConverter()));
13
14     endpoints.tokenStore(tokenStore()) //授权服务器颁发的token 怎么存储的
15     .tokenEnhancer(tokenEnhancerChain)

```

```

16     .userService(tulingUserDetailService) //用户来获取token的时候需要
    进行账号密码
17     .authenticationManager(authenticationManager);
18 }
19

```

## 第六步:认证服务器安全配置

```

1  /**
2   * 方法实现说明:授权服务器安全配置
3   * @author:smlz
4   * @return:
5   * @exception:
6   * @date:2020/1/15 20:23
7   */
8  @Override
9  public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
10     //第三方客户端校验token需要带入 clientId 和clientSecret来校验
11     security .checkTokenAccess("isAuthenticated()")
12     .tokenKeyAccess("isAuthenticated()");//来获取我们的tokenKey需要带入clientId,clientSecret
13
14     security.allowFormAuthenticationForClients();
15 }

```

## 第七步: 认证服务器工程的安全配置

```

1  /**
2   * @vlog: 高于生活，源于生活
3   * @desc: 类的描述:授权中心安全配置
4   * @author: smlz
5   * @createDate: 2019/12/20 16:08
6   * @version: 1.0
7   */
8  @Configuration
9  @EnableWebSecurity
10 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
11
12     @Autowired
13     private TulingUserDetailService tulingUserDetailService;
14
15
16

```



```
17  /**
18  * 方法实现说明:用于构建用户认证组件,需要传递userService和密码加密器
19  * @author:smlz
20  * @param auth
21  * @return:
22  * @exception:
23  * @date:2019/12/25 14:31
24  */
25  @Override
26  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
27      auth.userDetailsService(tulingUserDetailService).passwordEncoder(passwordEncoder());
28  }
29
30
31  /**
32  * 设置前台静态资源不拦截
33  * @param web
34  * @throws Exception
35  */
36  @Override
37  public void configure(WebSecurity web) throws Exception {
38      web.ignoring().antMatchers("/assets/**", "/css/**", "/images/**");
39  }
40
41
42
43  @Bean
44  public PasswordEncoder passwordEncoder() {
45      return new BCryptPasswordEncoder();
46  }
47
48  @Bean
49  public AuthenticationManager authenticationManagerBean() throws Exception {
50      return super.authenticationManagerBean();
51  }
52
53  public static void main(String[] args) {
54      System.out.println(new BCryptPasswordEncoder().encode("test"));
```

```
55 }  
56  
57 }
```

## 第八步：核心类的UserDetails的编写 用于用户登陆的

```
1  /**  
2  * @vlog: 高于生活，源于生活  
3  * @desc: 类的描述:认证服务器加载用户的类  
4  * @author: smlz  
5  * @createDate: 2020/1/21 21:29  
6  * @version: 1.0  
7  */  
8  @Slf4j  
9  @Component  
10 public class TulingUserDetailService implements UserDetailsService {  
11  
12     /**  
13      * 方法实现说明:用户登陆  
14      * @author:smlz  
15      * @param userName 用户名密码  
16      * @return: UserDetails  
17      * @exception:  
18      * @date:2020/1/21 21:30  
19      */  
20  
21     @Autowired  
22     private UmsMemberMapper memberMapper;  
23  
24     @Override  
25     public UserDetails loadUserByUsername(String userName) throws UsernameNot  
otFoundException {  
26  
27         if(StringUtils.isEmpty(userName)) {  
28             log.warn("用户登陆用户名为空:{}",userName);  
29             throw new UsernameNotFoundException("用户名不能为空");  
30         }  
31  
32         UmsMember umsMember = getByUsername(userName);  
33  
34         if(null == umsMember) {  
35             log.warn("根据用户名没有查询到对应的用户信息:{}",userName);
```

```

36     }
37
38     log.info("根据用户名:{0}获取用户登陆信息:{0}", userName, umsMember);
39
40     MemberDetails memberDetails = new MemberDetails(umsMember);
41
42     return memberDetails;
43 }
44
45 /**
46  * 方法实现说明:根据用户名获取用户信息
47  * @author:smlz
48  * @param username:用户名
49  * @return: UmsMember 会员对象
50  * @exception:
51  * @date:2020/1/21 21:34
52  */
53 public UmsMember getByUsername(String username) {
54     UmsMemberExample example = new UmsMemberExample();
55     example.createCriteria().andUsernameEqualTo(username);
56     List<UmsMember> memberList = memberMapper.selectByExample(example);
57     if (!CollectionUtils.isEmpty(memberList)) {
58         return memberList.get(0);
59     }
60     return null;
61 }
62 }

```

## 三:认证服务器自测

### 准备工作:

**3.1)还记得**这个表吧 我们需要创建一个应用 member-server 的应用来访问我们的授权服务器.

client_id	resource_ids	client_secret	scope	authorized_grant_types
api-gateway	order-service,product-service,member-service	\$2a\$10\$vrIjbIA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/iBoHtshii5jUdp9G	read,write	password
member-service	(Null)	\$2a\$10\$vrIjbIA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/iBoHtshii5jUdp9G	read,write,test	password,refresh_token
order_app	order-service	\$2a\$10\$vrIjbIA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/iBoHtshii5jUdp9G	read	password
portal_app	order-service,product-service,api-gateway,authorize-server	\$2a\$10\$E.QABsfckMmiX2W81iqRbulfnVtJoGD1nWuYZ0k8ZCPuFqFY2Pu	read,write	password,authorization_co
product_app	product-service	\$2a\$10\$vrIjbIA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/iBoHtshii5jUdp9G	read,write	password

```

1 INSERT INTO `oauth_client_details`
2 VALUES ('member-service', null, '$2a$10$vrIjbIA99pqEIXR08dbLVuSQSEYHGZ7Pn51P/iBoHtshii5jUdp9G', 'read,write,test', 'password,refresh_token', null, n



```

```
ull, '7200', '9999999', null, 'true');
```

### ①:模拟用户登陆(获取Token的请求)

<http://localhost:8899/oauth/token>

### 参数说明:

▶ BODY  

Form ▼

☒ username [ Text ▼ ] = test


☒ password [ Text ▼ ] = test

☒ grant\_type [ Text ▼ ] = password

☒ scope [ Text ▼ ] = read write test

+ Add form parameter

☒ application/x-www-form-urlencoded ▼



用户的账号密码

授权码模式

申请令牌的作用域权限

The screenshot shows the 'Authorization' dialog box in Postman. The 'Type' is set to 'Authorization'. The 'Username' field is filled with 'member-service' and the 'Password' field is filled with 'smlz'. The 'show password' checkbox is checked. A red box highlights the Username and Password fields, with a red arrow pointing to the text '第三方应用的账号密码' (Account and password of the third-party application). The background shows the Postman interface with the 'getToken' endpoint and headers.

BODY 

pretty 

```
{
  "access token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJ0ZXN0Iiwic2NvcGUiOiJ0ZXN0IiwiaWF0IjoxNjU0MjU0MDA0LCJpc29udGVudCI6ImVudGVudCJ9",
  "token_type": "bearer",
  "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJ0ZXN0Iiwic2NvcGUiOiJ0ZXN0IiwiaWF0IjoxNjU0MjU0MDA0LCJpc29udGVudCI6ImVudGVudCJ9",
  "expires_in": 7199,
  "scope": "read test write",
  "additionalInfo": {
    "nickName": "windir",
    "integration": 5000,
    "memberId": 1
  },
  "jti": "0d27668f-501a-43d4-a037-0dd39f2de9c4"
}
```

这个就是我们想要的token了

我们来看下我们的access\_token里面是啥

copy access\_token的值去<https://jwt.io/> 看看

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJ0ZXN0Iiwic2NvcGU0IjoiIiwiaWF0Ijoi1c2VhbmFhZCIsInRlc3Q3cm10ZSjdLCJhZGRpdGlubmFzSW5mbyI6eyJuaWNrTmFtZSI6IndpbnRpciIsIm1udGVncmF0aW9uIjo1MDAwLCJtZW1iZXJJZCI6MX0sImV4cCI6MTU4MTkzOTk2MCwiYXV0aG9yaXRPZXMiOi01siVEVTVVCjdLCJqdGkiOiIwZDI3NjY4Zi01MDFhLTQzZDQtYTZAzNy0wZGQzOWYyZGU5YzQiLCJjbG1lbnRfaWQiOiJtZW1iZXItc2VydmljZSJ9.Cv41inu2gfD0AhrSFBcIkBETexGUGYvYTLUBSxWgsYRyyeEPfZ4DjHg1LRs_gYTA--dRKfNSDewB0805oxXktuo9ghi2BSDZznIDf09PW8Ki9KCUCgZj0Lf7G8CfYbR7KXWn1eCr-wmJ3CkdIGGViWfAZTZ-i8oY34xKSLQ0IBcrkvEa6HN2ZA6nh3FPECMZtR487KsT5Hx_7r0M3WK0rz7q1iLD7mGdSQ1xVcVEc4oUtW9Hj0LwvBmkLlnkLYzQufgWz-crToAafhwiPjP80zBnQzC8GE-1Vk3Si02TBwhLXrZRubbXukUUhVhjCilULfYtoa05obhiAniFyFQczg
```

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "RS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "user_name": "test",  "scope": [    "read",    "test",    "write"  ],  "additionalInfo": {    "nickName": "windir",    "integration": 5000,    "memberId": 1  },  "exp": 1581939960,  "authorities": [    "TEST"  ],  "jti": "0d27668f-501a-43d4-a037-0dd39f2de9c4",  "client_id": "member-service"}
```

## ②：测试校验token 接口

BODY  

Form 


☒ token [ Text  ] = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbWUiOiJ0ZXN0Iiwic2NvcGU0IjoiIiwiaWF0Ijoi1c2VhbmFhZCIsInRlc3Q3cm10ZSjdLCJhZGRpdGlubmFzSW5mbyI6eyJuaWNrTmFtZSI6IndpbnRpciIsIm1udGVncmF0aW9uIjo1MDAwLCJtZW1iZXJJZCI6MX0sImV4cCI6MTU4MTkzOTk2MCwiYXV0aG9yaXRPZXMiOi01siVEVTVVCjdLCJqdGkiOiIwZDI3NjY4Zi01MDFhLTQzZDQtYTZAzNy0wZGQzOWYyZGU5YzQiLCJjbG1lbnRfaWQiOiJtZW1iZXItc2VydmljZSJ9.Cv41inu2gfD0AhrSFBcIkBETexGUGYvYTLUBSxWgsYRyyeEPfZ4DjHg1LRs\_gYTA--dRKfNSDewB0805oxXktuo9ghi2BSDZznIDf09PW8Ki9KCUCgZj0Lf7G8CfYbR7KXWn1eCr-wmJ3CkdIGGViWfAZTZ-i8oY34xKSLQ0IBcrkvEa6HN2ZA6nh3FPECMZtR487KsT5Hx\_7r0M3WK0rz7q1iLD7mGdSQ1xVcVEc4oUtW9Hj0LwvBmkLlnkLYzQufgWz-crToAafhwiPjP80zBnQzC8GE-1Vk3Si02TBwhLXrZRubbXukUUhVhjCilULfYtoa05obhiAniFyFQczg

+ Add form parameter ☒ application/x-www-form-urlencoded 

tulingmall > 认证中心接口测试



check\_token(校验令牌)

METHOD POST SCHEME http://localhost:8080

HEADERS 

☒ Content-Type : application/json

☒ Authorization : Basic bWVudGVncmF0aW9uIjo1MDAwLCJtZW1iZXJJZCI6MX0sImV4cCI6MTU4MTkzOTk2MCwiYXV0aG9yaXRPZXMiOi01siVEVTVVCjdLCJqdGkiOiIwZDI3NjY4Zi01MDFhLTQzZDQtYTZAzNy0wZGQzOWYyZGU5YzQiLCJjbG1lbnRfaWQiOiJtZW1iZXItc2VydmljZSJ9.Cv41inu2gfD0AhrSFBcIkBETexGUGYvYTLUBSxWgsYRyyeEPfZ4DjHg1LRs\_gYTA--dRKfNSDewB0805oxXktuo9ghi2BSDZznIDf09PW8Ki9KCUCgZj0Lf7G8CfYbR7KXWn1eCr-wmJ3CkdIGGViWfAZTZ-i8oY34xKSLQ0IBcrkvEa6HN2ZA6nh3FPECMZtR487KsT5Hx\_7r0M3WK0rz7q1iLD7mGdSQ1xVcVEc4oUtW9Hj0LwvBmkLlnkLYzQufgWz-crToAafhwiPjP80zBnQzC8GE-1Vk3Si02TBwhLXrZRubbXukUUhVhjCilULfYtoa05obhiAniFyFQczg

+ Add header  Add authorization 

Response

Authorization

Type Authorization

Username member-service

Password smlz ☒ show password

Cancel Set

▶ BODY ?

```
{
  user_name : "test",
  scope : [
    "read",
    "test",
    "write"
  ],
  additionalInfo : {
    nickName : "windir",
    integration : 5000,
    memberId : 1
  },
  active : true,
  exp : 1581939960,
  authorities : [
    "TEST"
  ],
  jti : "0d27668f-501a-43d4-a037-0dd39f2de9c4",
  client_id : "member-service"
}
```

### ③测试刷新token的接口

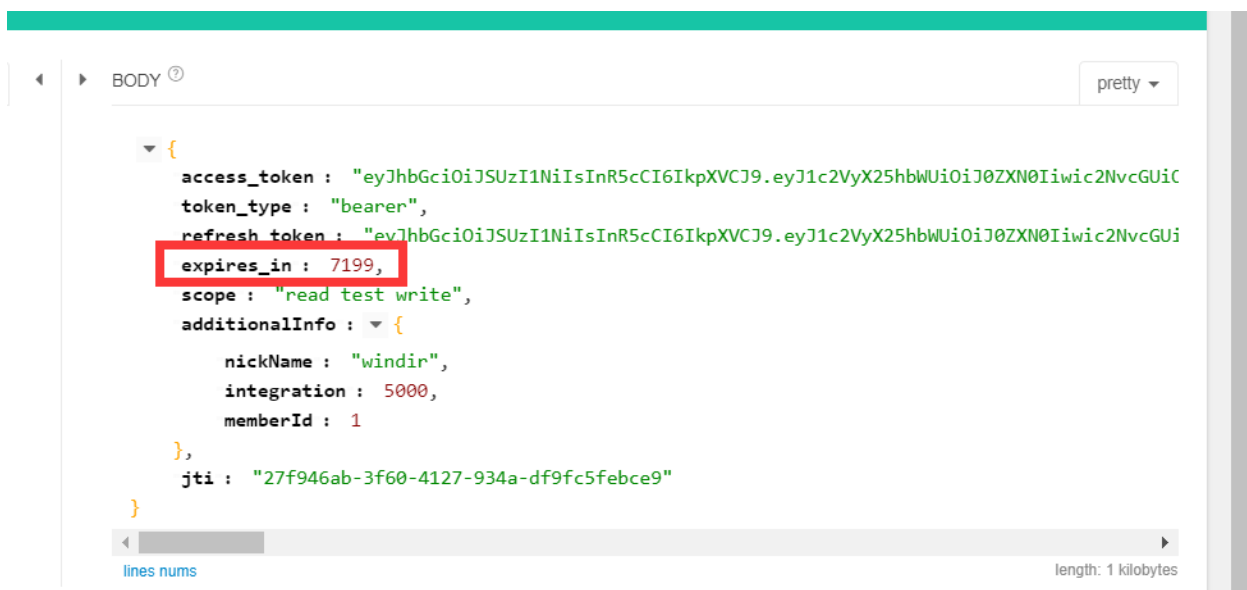
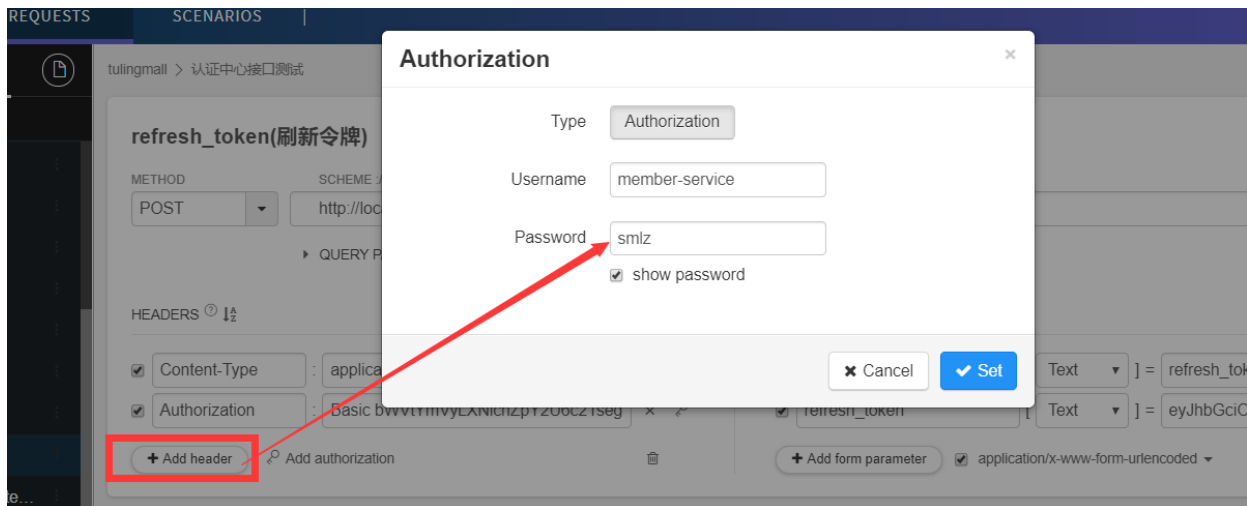
<http://localhost:8899/oauth/token>

length: 33 byte(s) Send

▶ BODY ? Form

<input checked="" type="checkbox"/>	grant_type	[ Text ] =	refresh_token	×
<input checked="" type="checkbox"/>	refresh_token	[ Text ] =	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX25hbGl	×

+ Add form parameter ☒ application/x-www-form-urlencoded refresh\_token的值



四: 经过上面的配置，我们现在搭建网关工程，通过网关工程来测试用户中心授权登陆.

#### 4.1) 创建 网关工程 tulingmall-gateway

添加依赖:(后续还要添加的)

```
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-gateway</artifactId>
4 </dependency>
5
6 <!--加入nacos的依赖-->
7 <dependency>
8   <groupId>com.alibaba.cloud</groupId>
9   <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
10 </dependency>
```

写配置:

```
1 server:
2   port: 8888
3 spring:
4   application:
5     name: tulingmall-gateway
6   cloud:
7     gateway:
8     discovery:
9     locator:
10      lower-case-service-id: true
11     enabled: true
12     routes:
13       - id: tulingmall-authcenter
14         uri: lb://tulingmall-authcenter
15         predicates:
16           - Path=/oauth/**
17     nacos:
18     discovery:
19       server-addr: 192.168.159.8:8848
20
21 management: #开启SpringBoot Admin的监控
22   endpoints:
23     web:
24     exposure:
25       include: '*'
26     endpoint:
27     health:
28     show-details: always
29 logging:
30   level:
31     org.springframework.cloud.gateway: debug
32
33 /**
34  **配置网关不需要校验的配置路径
35  **/
36 tuling:
37   gateway:
38     shouldSkipUrls:
39       - /oauth/**
40       - /sso/**
```



## 4.2)编写GateWay的全局过滤器进行拦截 校验

```
1 @Component
2 @Slf4j
3 @EnableConfigurationProperties(value = NotAuthUrlProperties.class)
4 public class AuthorizationFilter implements GlobalFilter,Ordered {
5
6     /**
7      * 请求各个微服务 不需要用户认证的URL
8      */
9     @Autowired
10    private NotAuthUrlProperties notAuthUrlProperties;
11
12    @Override
13    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
14
15        String currentUrl = exchange.getRequest().getURI().getPath();
16
17        //1:不需要认证的url
18        if(shouldSkip(currentUrl)) {
19            log.info("跳过认证的URL:{}",currentUrl);
20            return chain.filter(exchange);
21        }
22    }
```

到这里我们的网关雏形就完成了，现在我们就来测试功能，通过网关测试功能

1) 我们这里通过挑选一个代表 就是获取token来测试我们的 通过网关能获取到我们的服务的。

<http://localhost:8888/oauth/token>

[illegible]

## 测试通过