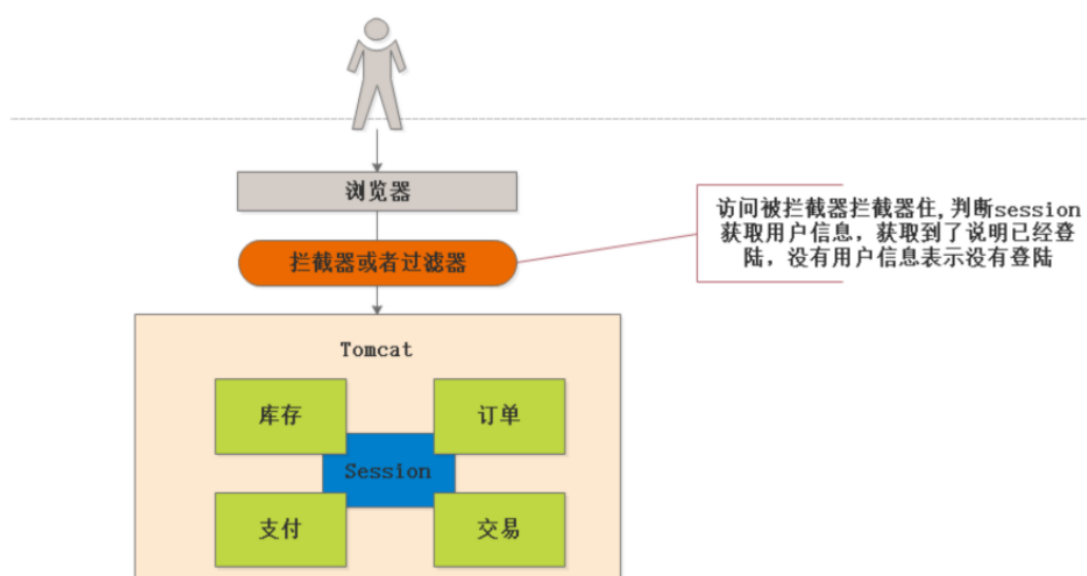


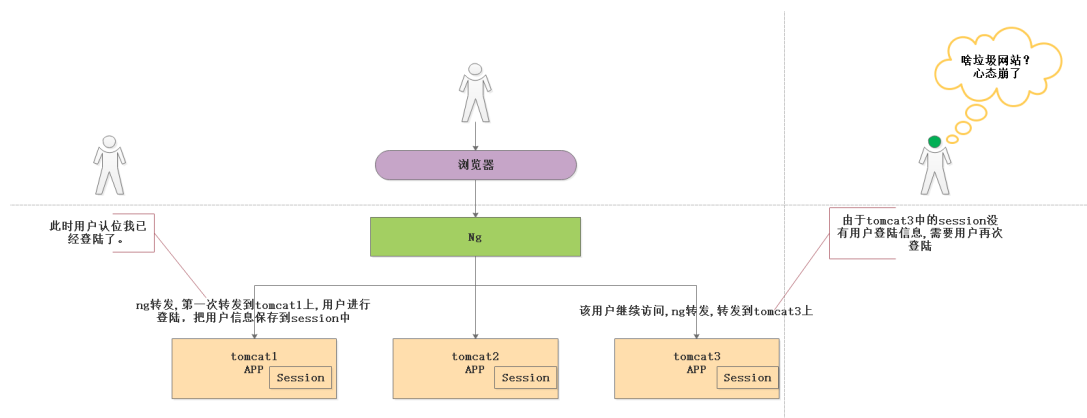
# 单体应用的安全

系统中某些页面只有在正常登录后才可以使用，用户请求这些页面时要检查session中是否有该用户信息

**解决方案：**编写一个用于检测用户是否登录的过滤器，如果用户未登录，则重定向到指定的登录页面



## 集群环境下如何解决登陆问题



## 解决方案一: NG的iphash算法

IP绑定 ip\_hash

每个请求按访问ip的hash结果分配，这样每个访客固定访问一个后端服务器，可以解决session的问题。

```
1 upstream backserver {  
2 ip_hash;
```

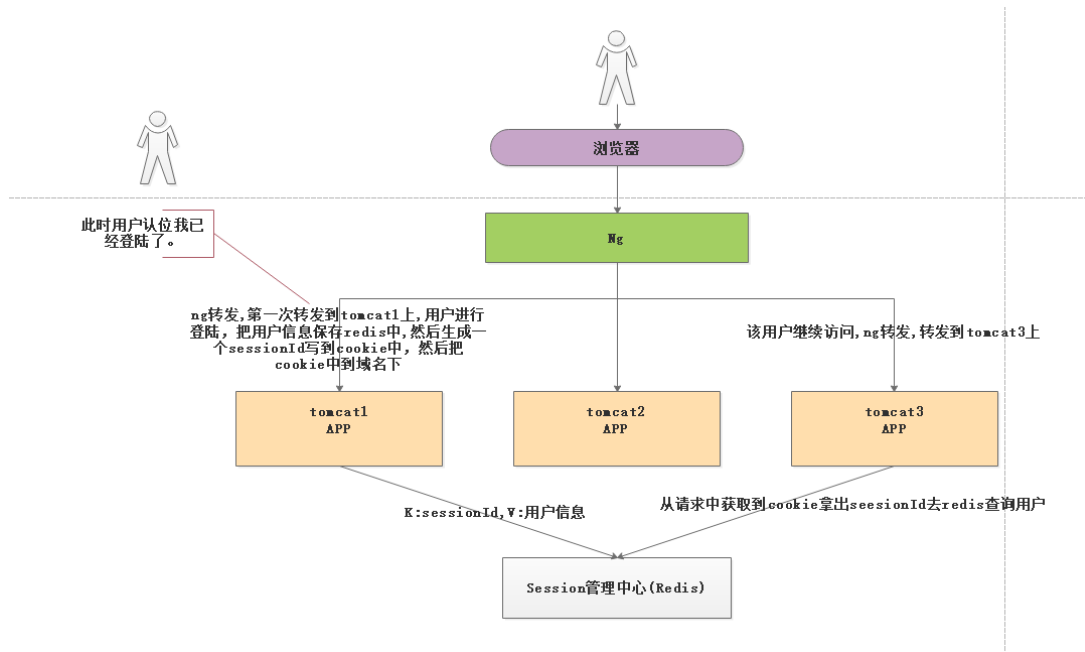
```

3 server 192.168.0.14:88;
4 server 192.168.0.15:80;
5 }

```

**缺点:**不能充分考虑到各个服务器的性能,有可能同一时刻访问通过ip\_hash出来都请求到一台服务器上,而且这台服务器的性能不行???

## 解决方案二:集中式Session。



## 核心代码

```

1 public class CookieUtil {
2
3     private static final String COOKIE_DOMAIN = ".happymmall.com";
4     private static final String COOKIE_NAME = "mmall_login_token";
5
6     public static String readLoginToken(HttpServletRequest request) {
7         Cookie[] cookies = request.getCookies();
8         if (cookies != null) {
9             for (Cookie cookie : cookies) {
10                 log.info("read cookieName:{}, cookieValue:{},", cookie.getName(), cookie.getValue());
11                 if (StringUtils.equals(COOKIE_NAME, cookie.getName())) {
12                     log.info("return cookieName:{}, cookieValue:{},", cookie.getName(),
13                         cookie.getValue());
14                     return cookie.getValue();
15                 }
16             }
17             return null;
18         }
19
20     public static void writeLoginToken(HttpServletResponse response, String token) {
21         Cookie cookie = new Cookie(COOKIE_NAME, token);
22     }
23 }

```

```

22  cookie.setDomain(COOKIE_DOMAIN);
23  cookie.setPath("/");
24  // 防止脚本攻击
25  cookie.setHttpOnly(true);
26  // 单位是秒，如果是 -1，代表永久；
27  // 如果 MaxAge 不设置，cookie 不会写入硬盘，而是在内存，只在当前页面有效
28  cookie.setMaxAge(60 * 60 * 24 * 365);
29  log.info("write cookieName:{}, cookieValue:{}", cookie.getName(),
cookie.getValue());
30  response.addCookie(cookie);
31  }
32
33  public static void delLoginToken(HttpServletRequest request, HttpServletResponse res
ponse) {
34  Cookie[] cookies = request.getCookies();
35  if (cookies != null) {
36  for (Cookie cookie : cookies) {
37  if (StringUtils.equals(COOKIE_NAME, cookie.getName())) {
38  cookie.setDomain(COOKIE_DOMAIN);
39  cookie.setPath("/");
40  // maxAge 设置为 0，表示将其删除
41  cookie.setMaxAge(0);
42  log.info("del cookieName:{}, cookieValue:{}", cookie.getName(), cookie.getValue());
43  response.addCookie(cookie);
44  return;
45  }
46  }
47  }
48  }
49  }
50
51
52  =====登陆成功=====
53  CookieUtil.writeLoginToken(response, session.getId());
54  RedisShardedPoolUtil.setEx(session.getId(),
JsonUtil.obj2Str(serverResponse.getData()),
Const.RedisCacheExtTime.REDIS_SESSION_EXTIME);
55
56
57  =====退出登陆=====
58  String loginToken = CookieUtil.readLoginToken(request);
59  CookieUtil.delLoginToken(request, response);
60  RedisShardedPoolUtil.del(loginToken);
61
62  =====获取用户信息=====
63  String loginToken = CookieUtil.readLoginToken(request);
64  if (StringUtils.isEmpty(loginToken)) {

```

```

65     return ServerResponse.createByErrorMessage("用户未登录，无法获取当前用户信息");
66 }
67 String userJsonStr = RedisShardedPoolUtil.get(loginToken);
68 User user = JsonUtil.str2Obj(userJsonStr, User.class);
69
70 SessionExpireFilter 过滤器
71 另外，在用户登录后，每次操作后，都需要重置 Session 的有效期。可以使用过滤器来实现
72 public class SessionExpireFilter implements Filter {
73
74     @Override
75     public void init(FilterConfig filterConfig) throws ServletException { }
76
77     @Override
78     public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
79         FilterChain filterChain) throws IOException, ServletException {
80         HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
81         String loginToken = CookieUtil.readLoginToken(httpServletRequest);
82         if (StringUtils.isEmpty(loginToken)) {
83             String userJsonStr = RedisShardedPoolUtil.get(loginToken);
84             User user = JsonUtil.str2Obj(userJsonStr, User.class);
85             if (user != null) {
86                 RedisShardedPoolUtil.expire(loginToken,
87                     Const.RedisCacheExtime.REDIS_SESSION_EXTIME);
88             }
89             filterChain.doFilter(servletRequest, servletResponse);
90         }
91     }
92     @Override
93     public void destroy() { }
94 }
95

```

## 传统的SSO

[https://pan.baidu.com/s/1HFIOHZ-QWoZ4rFwOvpT\\_JA](https://pan.baidu.com/s/1HFIOHZ-QWoZ4rFwOvpT_JA)

提取码: to3v

## 微服务安全

### 1) 什么是一个Oauth2协议?

**你真的没有接触过Oatuh2协议么？不，你肯定接触过，老师说的。**

**生活场景:**你玩 王者荣耀登录账号的时候，有一个用QQ登陆的选项，然后你点击该按钮，然后通过QQ登陆，进入游戏。

OAuth（开放授权）是一个开放标准，允许**用户(你)**授权第三方应用(**王者农药**)访问用户存储在另外的服务提供者(**QQ服务器**)上的信息，而不需要将用户名和密码提供给第三方移动应用(**王者农药**)-----这个就是典型的**授权码模式**。

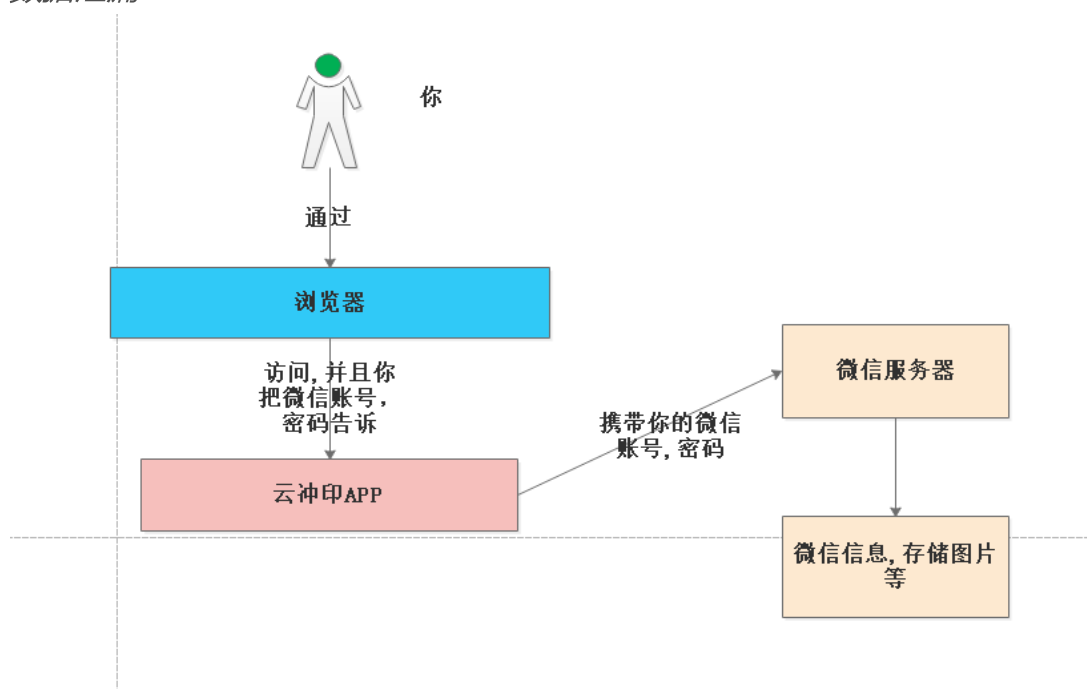
**一例不够，再来一例(密码模式 使用场景 第三方APP是高度被信任的)**

**vip学员张三,通过浏览器访问 老师(司马老师)开发的 "云冲印网站"，去打印你在微信上存储的照片。**

问题是只有得到用户张三的授权，**微信**才会同意"云冲印"读取这些照片。那么，"云冲印"怎样获得用户的授权呢？传统方法是，用户将自己的微信用户名和密码，告诉"云冲印"，后者就可以读取用户的照片了。这样的做法有以下几个严重的缺点。

缺点：

- (1) "云冲印"为了后续的服务，会保存用户的密码，这样很不安全。
- (2) "云冲印"拥有了获取用户储存在Google所有资料的权力，用户没法限制"云冲印"获得授权的范围和有效期。
- (3) 用户只有修改密码，才能收回赋予"云冲印"的权力。但是这样做，会使得其他所有获得用户授权的第三方应用程序全部失效。
- (4) 只要有一个第三方应用程序被破解，就会导致用户密码泄漏，以及所有被密码保护的数据泄漏



**名称解释:**

- (1) **Third-party application**: 第三方应用程序，本文中又称"客户端" (client)，即上一节例子中的"云冲印"。
- (2) **HTTP service**: HTTP服务提供商，本文中简称"服务提供商"，即上一节例子中的微信服务器

(3) **Resource Owner**: 资源所有者, 本文中又称"用户" (user) 。你

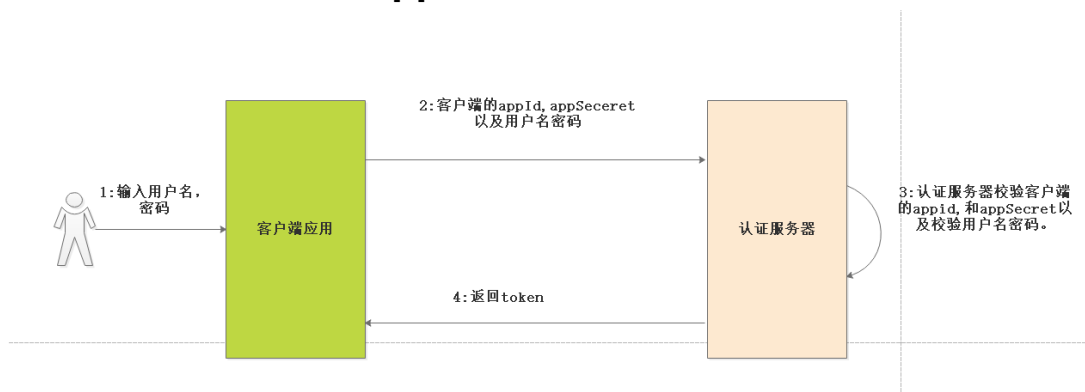
(4) **User Agent**: 用户代理, 本文中就是指浏览器。

(5) **Authorization server**: 认证服务器, 即服务提供商专门用来处理认证的服务器。

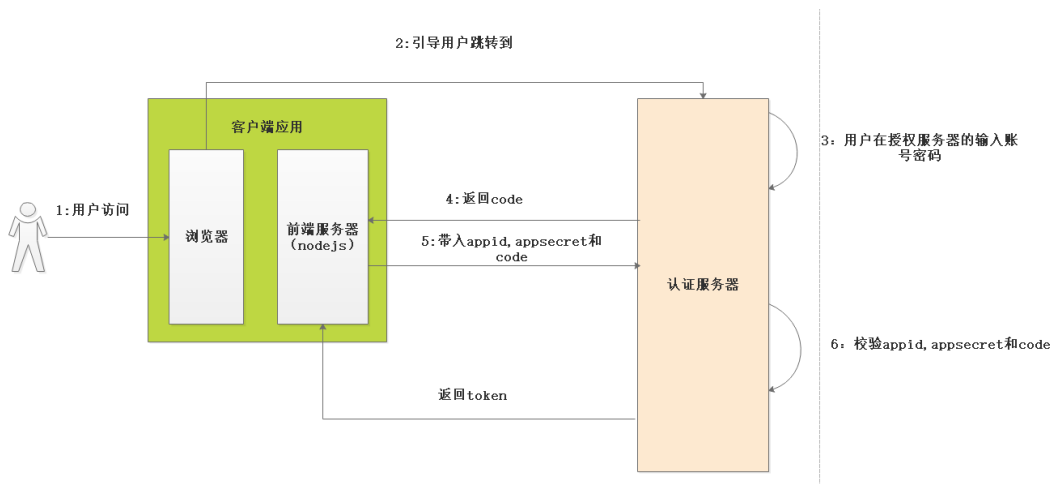
(6) **Resource server**: 资源服务器, 即服务提供商存放用户生成的资源的服务器。它与认证服务器, 可以是同一台服务器, 也可以是不同的服务器。

## 2)Oauth2的四种授权模式

**a)密码模式** 适用的业务场景 客户端应用(手机app) 是高度受信用的, 一般是自己公司开发的app项目。



**b)授权码模式(最安全的模式)** 业务场景 第三方不授信的,搭建自己的开发能力平台。



①:获取授权码[http://localhost:9999/oauth/authorize?](http://localhost:9999/oauth/authorize?response_type=code&client_id=portal_app&redirect_uri=http://www.baidu.com&state=abc)

[response\\_type=code&client\\_id=portal\\_app&redirect\\_uri=http://www.baidu.com&state=abc](http://localhost:9999/oauth/authorize?response_type=code&client_id=portal_app&redirect_uri=http://www.baidu.com&state=abc)

参数说明: client\_id 认证服务器分配给第三方客户端的appid

response\_type:固定格式 值位code

redirect\_uri: 用户在认证服务器上登陆成功了 需要回调到 客户端应用上

state: 你传什么给授权服务器 授权服务器原封不动的返回给你

localhost:9999/login

Please sign in

Sign in

输入你自己的用户名 密码

localhost:9999/oauth/authorize?response\_type=code&client\_id=portal\_app&redirect\_uri=http://www.baidu.com&state=abc

## OAuth Approval

Do you authorize "portal\_app" to access your protected resources?

• scope:read ☒ Approve ☐ Deny

Authorize

拒绝授权

同意授权

baidu.com/code=DOuSfb&state=abc

长沙: 17°C 54 | 换肤 消息

## 2:获取到code, 去换取token

<http://localhost:9999/oauth/token> (Post请求)

getCode(授权码模式-不走网关)

METHOD: POST

SCHEME: //HOST [ ":" PORT ] [ PATH [ "?" QUERY ] ]

http://localhost:9999/oauth/token

length: 33 byte(s)

HEADERS

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

Authorization: Basic cG9ydGFsX2FwcDpw3J0YWxMY:

BODY

code: DOuSfb

grant\_type: authorization\_code

redirect\_uri: http://www.baidu.com

Response

200

Cache Detected - Elapsed Time: 236ms

HEADERS

Pragma: no-cache

Cache-Control: no-store

X-Content-Type-Options: nosniff

X-XSS-Protection: 1; mode=block

X-Frame-Options: DENY

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

Transfer-Encoding: chunked

BODY

```
{  "access_token": "b914f265-051c-40b8-a920-0a49907c602f",  "token_type": "bearer",  "expires_in": 3599,  "scope": "read"}
```

## Authorization



Type Authorization

Username portal\_app

Password portal\_app

☒ show password

✕ Cancel

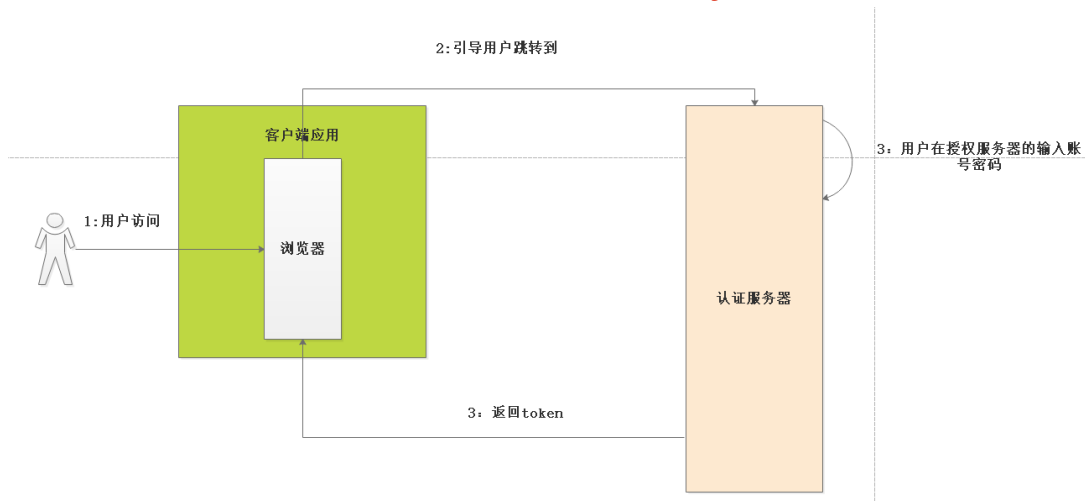
✓ Set

BODY ?

```
{
  access_token : "b914f265-051c-40b8-a920-0a49907c602f",
  token_type : "bearer",
  expires_in : 3599,
  scope : "read"
}
```

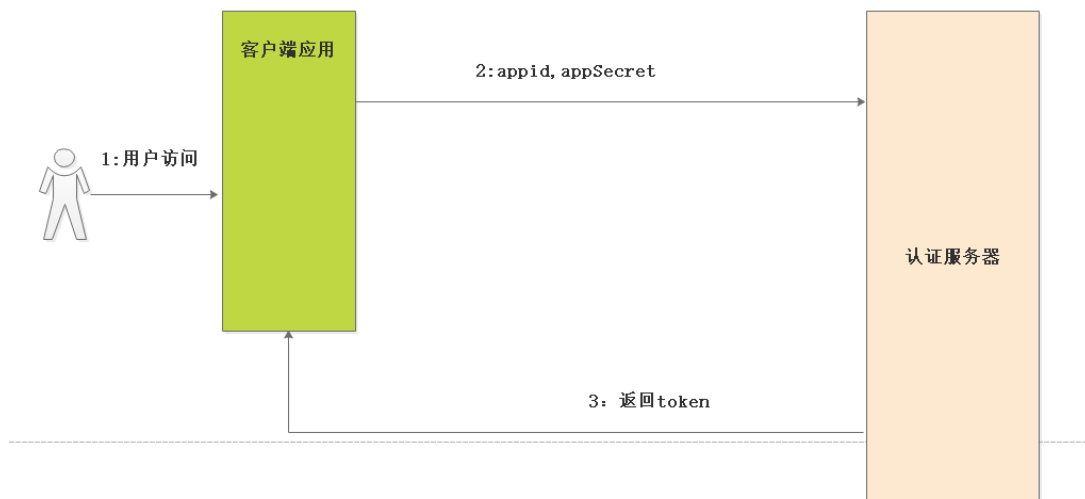
lines nums

**C)简化模式(开发中几乎接触不到 适用于 客户端就是一堆js css html 没有前端服务器)**



**d):客户端模式(开发中几乎用不到,这种模式 用户都没有参与过程)**





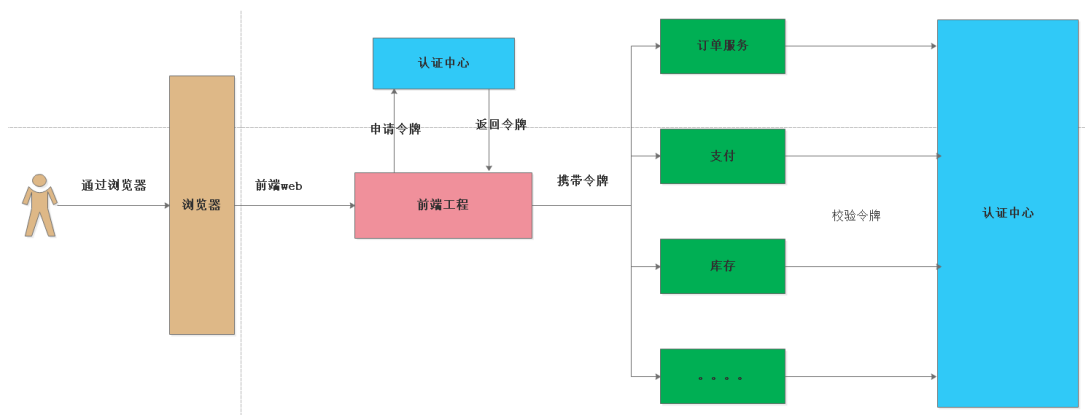
## 动手搭建微服务认证中心实现微服务鉴权角色

认证中心(认证服务器)

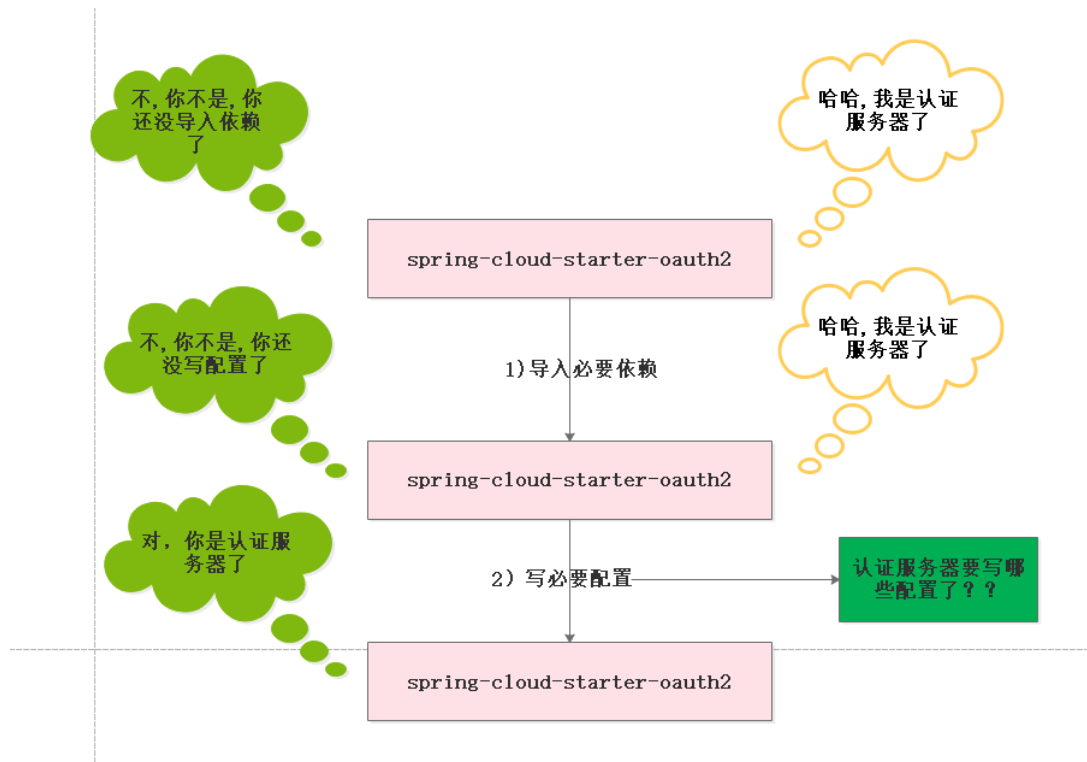
订单，支付，库存等后端微服务（资源）

前端工程 web-portal工程

你.



## 动手搭建认证中心



1)创建工程名称为tulingvip09-ms-auth-server, 加入依赖

主要是添加**spring-cloud-starter-oauth2**

```

1 <dependencies>
2 <dependency>
3 <groupId>org.springframework.boot</groupId>
4 <artifactId>spring-boot-starter</artifactId>
5 </dependency>
6
7 <dependency>
8 <groupId>org.springframework.boot</groupId>
9 <artifactId>spring-boot-starter-web</artifactId>
10 </dependency>
11
12 <dependency>
13 <groupId>com.alibaba.cloud</groupId>
14 <artifactId>spring-cloud-alibaba-nacos-discovery</artifactId>
15 </dependency>
16
17 <!--Oauth2的包-->
18 <dependency>
19 <groupId>org.springframework.cloud</groupId>
20 <artifactId>spring-cloud-starter-oauth2</artifactId>
21 </dependency>
22
23 </dependency>
24 </dependencies>
  
```

## 2)添加注解写配置文件

### 2.1)作为认证服务器,那么就会又认证服务器的配置

写一个配置类 `TulingAuthorizationServerConfig` 实现 `AuthorizationServerConfigurerAdapter` 添加 `@EnableAuthorizationServer`

`AuthorizationServerConfigurerAdapter` 类有三个配置方法, 我们都需要覆盖

①:第三方客户端配置, 配置哪些应用可以来访问我们认证服务器。

他有二种存储模式, 一种是内存模式, 一种是db模式(用于生产),后面讲

下面的配置解释说明:

为第三方客户端 分配一个client\_id为 `portal_app`, 密码为`portal_app`,他的权限访问是read权限的  
我服务器颁发的token 有效期是1一个小时, 我拿着这个token令牌可以访问`order-service`,`product-service`二个微服务。

`order-server`和`product-server`同理, 也是分配了账号。

```
1  @Override
2  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
3      /**
4       * 配置解析 授权服务器指定客户端(第三方应用)能访问授权服务器
5       * 为第三方应用颁发客户端 id为 ,密码为smlz
6       * 支持的授权类型为 密码模式(有四种模式,后面说)
7       * 颁发的令牌的有效期为1小时
8       * 通过该令牌可以访问 哪些资源服务器(order-service) 可以配置多个
9       * 访问资源服务器的read write权限
10     */
11
12     clients.inMemory()
13         .withClient("portal_app")
14         .secret(passwordEncoder.encode("portal_app"))
15         .authorizedGrantTypes("password")
16         .scopes("read")
17         .accessTokenValiditySeconds(3600)
18         .resourceIds("order-service", "product-service")
19         .and()
20         .withClient("order_app")
21         .secret(passwordEncoder.encode("smlz"))
22         .accessTokenValiditySeconds(1800)
23         .scopes("read")
24         .authorizedGrantTypes("password")
25         .resourceIds("order-service")
26         .and()
27         .withClient("product_app")
28         .secret(passwordEncoder.encode("smlz"))
29         .accessTokenValiditySeconds(1800)
```

```

30 .scopes("read")
31 .authorizedGrantTypes("password")
32 .resourceIds("product-service");
33 }

```

②:针对用户的配置，也就是说，第三方客户端带入过来的用户名，密码我认证中心怎么去验证他的正确性？

那么authenticationManager是一个什么东西？学名(认证管理器,用来给用户认证的),那么authenticationManager 怎么来的？？？？？？ 这里是一个疑问，请接下来再看

```

1 public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
2     endpoints.authenticationManager(authenticationManager);
3 }

```

③:针对 资源服务器 来校验令牌的配置。

意思，你资源服务器来校验令牌 需要带入client\_id和client\_secret过来

```

1 public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
2     //获取tokenkey需要登陆
3     security.checkTokenAccess("isAuthenticated()");
4 }

```

## 2.2)认证服务器的安全配置

①:写一个配置类WebSecurityConfig，实现 WebSecurityConfigurerAdapter类，有一个配置方法这个方法就是构建我们的authenticationManager对象,而构建该对象需要配置传递一个userDetailsService,以及一个加密器对象passwordEncoder

```

1 @Override
2 protected void configure(AuthenticationManagerBuilder auth) throws Exception {
3     auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
4 }

```

问题?userDetailsService怎么来？ passwordEncoder怎么来

配置加密器对象

```

1 @Bean
2 public PasswordEncoder passwordEncoder() {
3     return new BCryptPasswordEncoder();
4 }

```

写一个UserDetailService实现 UserDetailsService接口用于用户登陆认证的。

```

1 @Component("userDetailsService")
2 @Slf4j
3 public class TulingUserDetailService implements UserDetailsService {
4
5     @Autowired
6     private PasswordEncoder passwordEncoder;
7

```

```

8  @Override
9  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
10 //需要登录数据库的,这里入门搭建,直接写死
11 log.info("当前登陆用户名为:{},username);
12
13 return User.builder().username(username)
14 .password(passwordEncoder.encode("123456"))
15 .authorities("ROLE_ADMIN")
16 .build();
17 }
18 }

```

现在有了UserDetailsService 和passwordEncoder对象了 现在可以创建**AuthenticationManager**

```

1  @Bean
2  public AuthenticationManager authenticationManagerBean() throws Exception {
3  return super.authenticationManagerBean();
4  }

```

到这里就是配置好了我们的认证服务器了。测试认证中心。

测试地址:<http://localhost:9999/oauth/token>

测试参数:请求头参数

The screenshot shows a REST client interface with the following configuration:

- METHOD:** POST
- URL:** http://localhost:9999/oauth/token
- HEADERS:**
  - Content-Type: application/x-www-form-urlencoded
  - Buttons: + Add header, Add authorization (highlighted with a red box), and a trash icon.
- BODY:**
  - Fields: username (smlz), password (123456), grant\_type (password), scope (read).
  - Buttons: + Add form parameter, application/x-www-form-urlencoded (selected), and a trash icon.

输入上面我们配置的第三方账号密码。

# Authorization

Type

Basic

Username

portal\_app

Password

portal\_app

show password

Cancel

Set

表单内容:

REQUEST

METHOD

POST

SCHEME // HOST [ : PORT ] [ PATH [ "?" QUERY ] ]

http://localhost:9999/oauth/token

Send

length: 33 bytes

QUERY PARAMETERS

HEADERS

1/2

Form

Authori: Basic cG9ydGFsX2FwcDpw

Content: application/x-www-form-url

Add header

Add authorization

BODY

1/2

Form

username [ Text ] = smlz

password [ Text ] = 123456

grant\_type [ Text ] = password

scope [ Text ] = read

Add form parameter

application/x-www-form-urlencoded

你在认证服务器上的账号, 密码

密码模式

令牌的权限, 读权限

测试截图:

POST

http://localhost:9999/oauth/token

Send

length: 33 bytes

QUERY PARAMETERS

HEADERS

1/2

Form

Authorizato Basic cG9ydGFsX2FwcDpw

Content-Tyr application/x-www-form-urlencoded

Add header

Add authorization

BODY

1/2

Form

username [ Text ] = smlz

password [ Text ] = 123456

grant\_type [ Text ] = password

scope [ Text ] = read

Add form parameter

application/x-www-form-urlencoded

RESPONSE

Cache Detected - Elapsed Time: 560ms

200

HEADERS

1/2

pretty

cache-control: no-store

content-type: application/json; charset=UTF-8

date: 2019 Dec 27 21:36:37

pragma: no-cache

transfer-encoding: chunked

x-content-type-opt\_ nosniff

x-frame-options: DENY

x-xss-protection: 1; mode=block

BODY

1/2

pretty

access\_token : "8589154d-759e-4e9d-94af-391989cab93e"

token\_type : "bearer"

expires\_in : 3599

scope : "read"

令牌token

令牌的有效期

看到这个测试页面 恭喜认证中心搭建成功.

## 动手搭建微服务（资源服务器）

### 1)创建工程名称为tulingvip09-ms-alibaba-order，加入依赖

```
1 <dependency>
```

```

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

```

## 2)写配置(资源服务器的配置)

ResourceServerConfig extends ResourceServerConfigurerAdapter

```

1 @Configuration
2 @EnableResourceServer
3 public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
4
5     //标识自己是一个资源服务器 唯一标识为order-service
6     @Override
7     public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
8         resources.resourceId("order-service");
9     }
10
11     springsecurity的配置
12     @Override
13     public void configure(HttpSecurity http) throws Exception {
14         //标识/selectOrderInfoById/** 需要token的scope有read 权限
15         ///saveOrder 有写权限
16         http.authorizeRequests().
17             antMatchers("/selectOrderInfoById/**").access("#oauth2.hasScope('read')")
18             .and()
19             .authorizeRequests().antMatchers("/order/saveOrder").access("#oauth2.hasScope('write')");
20     }
21
22 }

```

## 3)资源服务器的安全配置

我资源服务器拿到令牌，我怎么知道这个令牌是否合法，所以我需要去配置 远程校验token的配置

```

1 @Configuration
2 @EnableWebSecurity
3 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
4
5     @Autowired
6     private RestTemplate restTemplate;
7
8     //远程校验token的配置
9     @Bean
10    public ResourceServerTokenServices resourceServerTokenServices() {
11        RemoteTokenServices remoteTokenServices = new RemoteTokenServices();
12        //client_id和密码
13        remoteTokenServices.setClientId("order_app");
14        remoteTokenServices.setClientSecret("smlz");
15        //认证服务器的校验地址

```

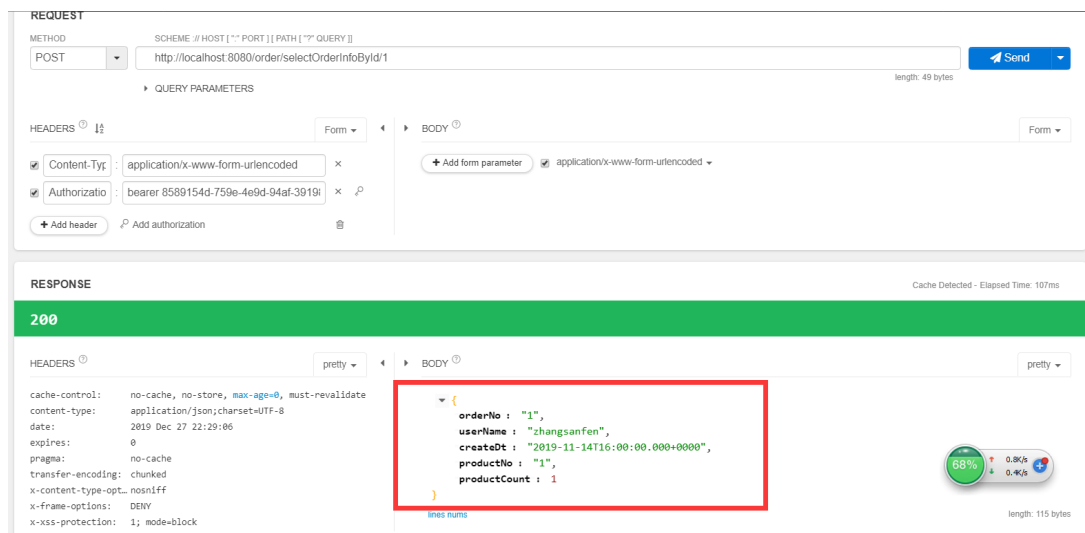
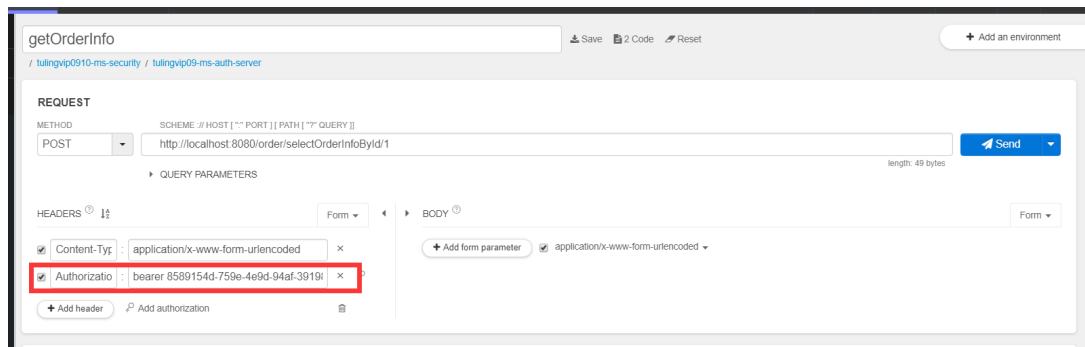
```

16 remoteTokenServices.setCheckTokenEndpointUrl("http://auth-
server/oauth/check_token");
17 remoteTokenServices.setRestTemplate(restTemplate);
18 return remoteTokenServices;
19 }
20
21 @Bean
22 public AuthenticationManager authenticationManagerBean() {
23 OAuth2AuthenticationManager manager = new OAuth2AuthenticationManager();
24 manager.setTokenServices(resourceServerTokenServices());
25 return manager;
26 }
27 }

```

**测试 资源服务器 <http://localhost:8080/order/selectOrderInfoById/1>**

**添加请求头: bearer token(bearer 8589154d-759e-4e9d-94af-391989cab93e)**



**不带请求头:**



getOrderInfo

Save 2 Code Reset + Add an environment

/ tulingvip0910-ms-security / tulingvip09-ms-auth-server

**REQUEST**

METHOD: POST SCHEME: // HOST: PORT: PATH: QUERY: http://localhost:8080/order/selectOrderInfoById/1 length: 49 bytes

QUERY PARAMETERS

HEADERS: Content-Type: application/x-www-form-urlencoded + Add header Add authorization

BODY: + Add form parameter application/x-www-form-urlencoded

**RESPONSE**

Cache Detected - Elapsed Time: 106ms

**401**

HEADERS: cache-control: no-store content-type: application/json;charset=UTF-8 date: 2019 Dec 27 23:26:15 -1s pragma: no-cache transfer-encoding: chunked

BODY: { "error": "unauthorized", "error\_description": "Full authentication is required to access this resource" }

带错误的请求头:

getOrderInfo

Save 2 Code Reset + Add an environment

/ tulingvip0910-ms-security / tulingvip09-ms-auth-server

**REQUEST**

METHOD: POST SCHEME: // HOST: PORT: PATH: QUERY: http://localhost:8080/order/selectOrderInfoById/1 length: 49 bytes

QUERY PARAMETERS

HEADERS: Content-Type: application/x-www-form-urlencoded Authorization: bearer abc + Add header Add authorization

BODY: + Add form parameter application/x-www-form-urlencoded

**RESPONSE**

Cache Detected - Elapsed Time: 1.21s

**500**

带只有读权限的token访问 需要写权限的资源

POST http://localhost:8080/order/saveOrder length: 37 bytes

HEADERS: Content-Type: application/x-www-form-urlencoded Authorization: bearer a4e6e99a-af28-4a4f-a9e8-46ae7 + Add header Add authorization

BODY: orderNo: 12345, userName: smlz, productNo: 1, productCount: 1 + Add form parameter application/x-www-form-urlencoded

**RESPONSE**

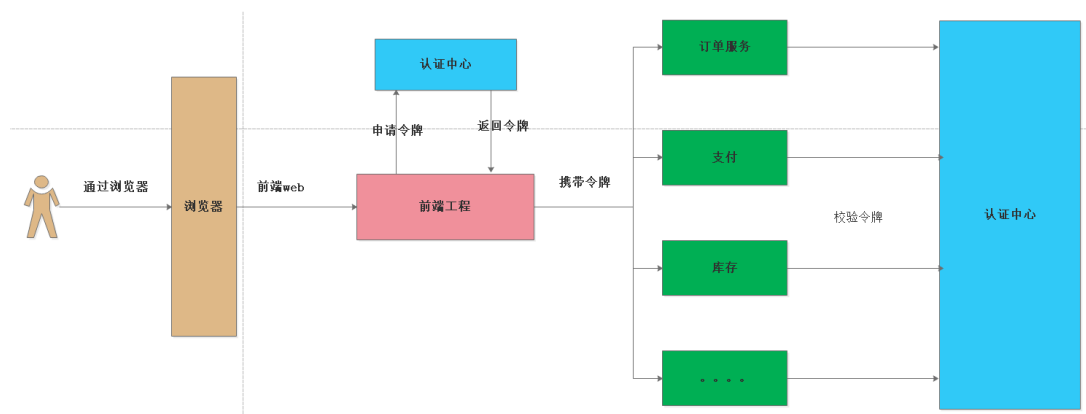
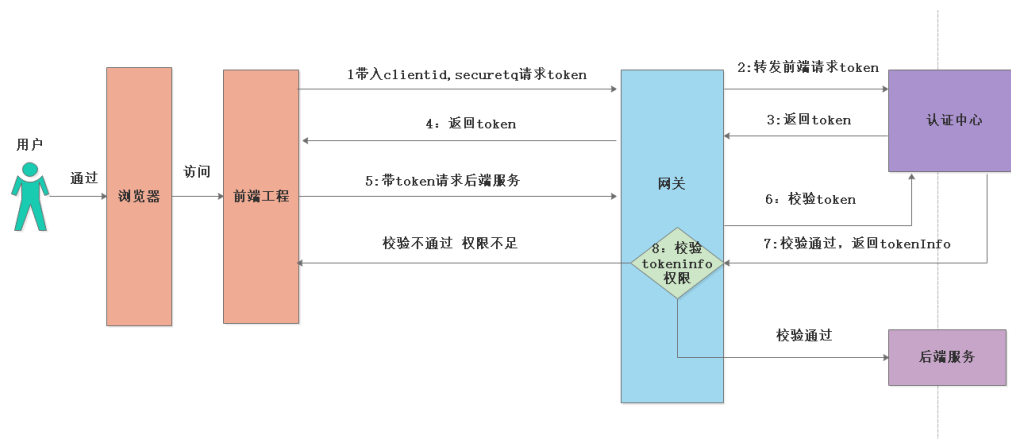
Cache Detected - Elapsed Time: 93ms

**403**

HEADERS: cache-control: no-store content-type: application/json;charset=UTF-8 date: 2019 Dec 27 23:30:20 pragma: no-cache transfer-encoding: chunked

BODY: { "error": "insufficient\_scope", "error\_description": "Insufficient scope for this resource", "scope": "write" }

生产最佳实践



## 改动点:基于上面二个图

第一，我们需要加入网关

第二，我们的微服务不再做权限控制，把权限控制功能放到网关上。

第三，授权服务器引入rbac模型权限控制。

## 第一:创建网关工程。tulingvip09-ms-cloud-gateway

### 网关功能分析

1)对于第三方应用来请求的 token请求，都不需要进行拦截，

2)对于带有token的请求对token进行校验token的正确性

3)从tokenInfo获取中 进行权限控制

4)记录访问日志等.....

图:Apigateway流程图

### 添加依赖:

```

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>

```

## 写网关路由配置

```

1 server:
2   port: 8888
3 spring:
4   application:
5     name: api-gateway
6   cloud:
7     gateway:
8     discovery:
9     locator:
10      lower-case-service-id: true
11     enabled: true
12   routes:
13     - id: product_center
14       uri: lb://product-center
15     predicates:
16       - Path=/product/**
17     - id: order_center
18       uri: lb://order-center
19     predicates:
20       - Path=/order/**
21     - id: auth_center
22       uri: lb://auth-server
23     predicates:
24       - Path=/oauth/**

```

## 编写核心过滤器 认证过滤器AuthorizationFilter

功能)

- 1)判断请求的路径是否需要授权，不需要直接放行
- 2)若需要经授权的才能访问的url 必须请求带有token 的请求，没带直接返回
- 3)去认证服务器 校验token的合法性，返回token
- 4) 把tokeninfo放入到request对象中

```

1 @Component
2 @Slf4j
3 public class AuthorizationFilter implements GlobalFilter,Ordered,InitializingBean {
4
5   @Autowired

```

```

6  private RestTemplate restTemplate;
7
8  /**
9   * 请求各个微服务 不需要用户认证的URL
10  */
11  private static Set<String> shouldSkipUrl = new LinkedHashSet<>();
12
13
14  @Override
15  public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
16
17      String reqPath = exchange.getRequest().getURI().getPath();
18      log.info("网关认证开始URL->:{", reqPath);
19
20      //1:不需要认证的url
21      if(shouldSkip(reqPath)) {
22          log.info("无需认证的路径");
23          return chain.filter(exchange);
24      }
25
26      //获取请求头
27      String authHeader = exchange.getRequest().getHeaders().getFirst("Authorization");
28
29      //请求头为空
30      if(StringUtils.isEmpty(authHeader)) {
31          log.warn("需要认证的url,请求头为空");
32          throw new GatewayException(SystemErrorType.UNAUTHORIZED_HEADER_IS_EMPTY);
33      }
34
35      TokenInfo tokenInfo=null;
36      try {
37          //获取token信息
38          tokenInfo = getTokenInfo(authHeader);
39      }catch (Exception e) {
40          log.warn("校验令牌异常:{", authHeader);
41          throw new GatewayException(SystemErrorType.INVALID_TOKEN);
42      }
43
44      /* //向headers中放文件,记得build
45      ServerHttpRequest request = exchange.getRequest().mutate().header("tokenInfo", (tokenInfo==null?"":tokenInfo).toString()).build();
46      //将现在的request 变成 change对象
47      ServerWebExchange serverWebExchange = exchange.mutate().request(request).build();*/
48
49      exchange.getAttributes().put("tokenInfo", tokenInfo);
50
51      return chain.filter(exchange);

```

```

52
53 }
54
55 private TokenInfo getTokenInfo(String authHeader) {
56     String token = StringUtils.substringAfter(authHeader, "bearer ");
57
58     HttpHeaders headers = new HttpHeaders();
59     headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
60     headers.setBasicAuth(MDA.clientId, MDA.clientSecret);
61
62     MultiValueMap<String, String> params = new LinkedMultiValueMap<>();
63     params.add("token", token);
64
65     HttpEntity<MultiValueMap<String, String>> entity = new HttpEntity<>(params,
66         headers);
67
68     ResponseEntity<TokenInfo> response = restTemplate.exchange(MDA.checkTokenUrl, HttpMethod.POST,
69         entity, TokenInfo.class);
70
71     log.info("token info : " + response.getBody().toString());
72
73     return response.getBody();
74 }
75
76 /**
77  * 方法实现说明:不需要授权的路径
78  * @author:smlz
79  * @param reqPath 当前请求路径
80  * @return:
81  * @exception:
82  * @date:2019/12/26 13:49
83  */
84 private boolean shouldSkip(String reqPath) {
85
86     for(String skipPath:shouldSkipUrl) {
87         if(reqPath.contains(skipPath)) {
88             return true;
89         }
90     }
91     return false;
92 }
93
94
95 @Override
96 public int getOrder() {

```

```

97     return 0;
98 }
99
100 @Override
101 public void afterPropertiesSet() throws Exception {
102     /**
103      * 实际上，这边需要通过去数据库读取 不需要认证的URL,不需要认证的URL是各个微服务
104      * 开发模块的人员提供出来的。我在这里没有去查询数据库了,直接模拟写死
105      */
106     //模仿商品详情接口不需要认证
107     shouldSkipUrl.add("/product/selectProductInfoById");
108     //去认证的请求,本来就不需要拦截
109     shouldSkipUrl.add("/oauth/token");
110 }
111 }

```

## 核心过滤器AuthenticationFilter 鉴权tokenInfo

### 1)不需要鉴权的 直接放行

### 2)tokenInfo是否有效

### 3)判断权限

```

1  @Component
2  @Slf4j
3  public class AuthenticationFilter implements GlobalFilter,Ordered{
4
5      @Override
6      public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
7          //获取当前请求的路径
8          String reqPath = exchange.getRequest().getURI().getPath();
9
10         TokenInfo tokenInfo = exchange.getAttribute("tokenInfo");
11
12         //无需拦截直接放行
13         if(shouldSkipUrl.contains(reqPath)) {
14             return chain.filter(exchange);
15         }
16
17         if(!tokenInfo.isActive()) {
18             log.warn("token过期");
19             throw new GateWayException(SystemErrorType.TOKEN_TIMEOUT);
20         }
21
22         boolean hasPremisson = false;
23         //登陆用户的权限集合判断
24         List<String> authorities = Arrays.asList(tokenInfo.getAuthorities());

```

```

25 for (String url: authorities) {
26     if(reqPath.contains(url)) {
27         hasPremisson = true;
28         break;
29     }
30 }
31 if(!hasPremisson){
32     log.warn("权限不足");
33     throw new GateWayException(SystemErrorType.FORBIDDEN);
34 }
35
36 return chain.filter(exchange);
37 }
38
39 @Override
40 public int getOrder() {
41     return 1;
42 }
43 }

```

**授权服务器引入rbac模型权限控制，以及改变token的存储方式,第三方客户端保存的方式 为了避免干扰，新创建一个工程**  
**tulingvip09-ms-auth-server-gateway**  
**认证服务器配置需要进行少量修改**  
**修改的内容是**

```

1  @Configuration
2  @EnableAuthorizationServer
3  public class AuthServerInDbConfig extends AuthorizationServerConfigurerAdapter {
4
5      @Autowired
6      private DataSource dataSource;
7
8      @Autowired
9      private RedisConnectionFactory redisConnectionFactory;
10
11     @Autowired
12     private AuthenticationManager authenticationManager;
13
14     /**
15      * 把第三方客户端存储到db中
16      * @param clients
17      * @throws Exception
18      */

```

```

19  @Override
20  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
21  //对比tulingvip09-ms-auth-server 我们把第三方客户端进行持久化到
22  //数据库中 对于的表为oauth_client_details
23  clients.jdbc(dataSource);
24  }
25
26  /**
27   * 把token存储到redis中
28   * @return
29   */
30  @Bean
31  public TokenStore tokenStore() {
32  //生产上 需要把token存储到redis中或者使用jwt
33  RedisTokenStore redisTokenStore = new RedisTokenStore(redisConnectionFactory);
34  //这里，我们需要把token存储到数据库或者是redis中
35  //return new JdbcTokenStore(dataSource);
36  return redisTokenStore;
37  }
38
39  /**
40   * 根据SpringSecurity 默认的token的存储模式 改为jwt的
41   * @param endpoints
42   * @throws Exception
43   */
44  @Override
45  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
46
47  //添加token的存储方式
48  endpoints
49  .tokenStore(tokenStore())
50  .authenticationManager(authenticationManager);
51
52  }
53
54  @Override
55  public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
56  security.checkTokenAccess("isAuthenticated()");
57  }
58
59  }

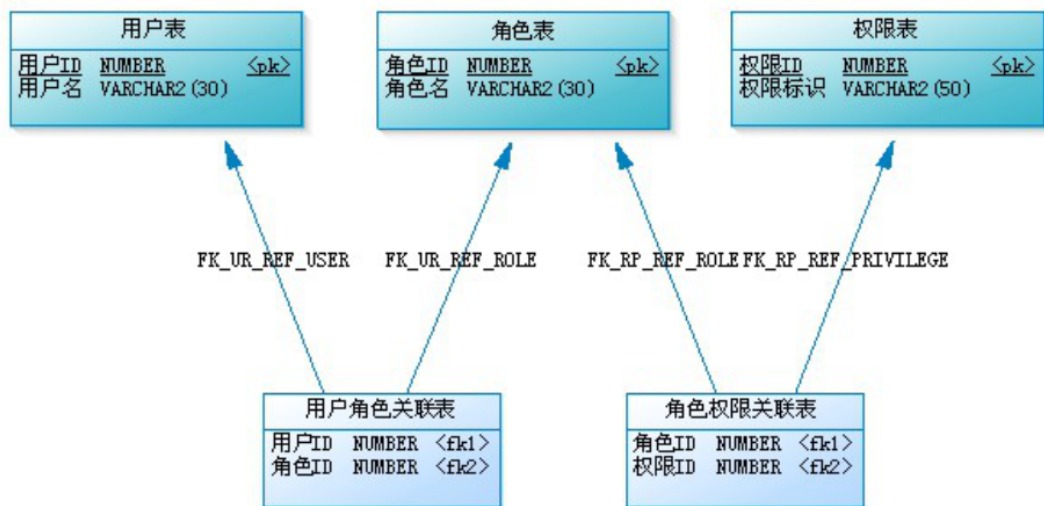
```



## UserDetialsService改造,通过数据库访问, 通过登陆用户, 返回用户的所有权限。

```
1  /**
2   * 该类都是基于内存的 , 后期会改变为db,需要去数据库中查询
3   * Created by smlz on 2019/12/25.
4   */
5  @Component("userDetailsService")
6  @Slf4j
7  public class TulingUserDetailService implements UserDetailsService {
8
9
10     //密码加密组件
11     @Autowired
12     private PasswordEncoder passwordEncoder;
13
14     @Autowired
15     private SysUserMapper sysUserMapper;
16
17     @Autowired
18     private ISysPermissionService sysPermissionService;
19
20     @Override
21     public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
22
23         SysUser sysUser = sysUserMapper.findByUserName(userName);
24
25         if(null == sysUser) {
26             log.warn("根据用户名: {}查询用户信息为空", userName);
27             throw new UsernameNotFoundException(userName);
28         }
29
30         List<SysPermission> sysPermissionList = sysPermissionService.findByUserId(sysUser.getId());
31
32         List<SimpleGrantedAuthority> authorityList = new ArrayList<>();
33         if (!CollectionUtils.isEmpty(sysPermissionList)) {
34             for (SysPermission sysPermission : sysPermissionList) {
35                 authorityList.add(new SimpleGrantedAuthority(sysPermission.getUri()));
36             }
37         }
38
39         TulingUser tulingUser = new
40             TulingUser(sysUser.getUsername(), passwordEncoder.encode(sysUser.getPassword()), authorityList);
41         log.info("用户登陆成功: {}", JSON.toJSONString(tulingUser));
42         return tulingUser;
43     }
44 }
```

```
42 }  
43 }
```



(图：RBAC权限模型)

### 具体的查询脚本:查询用户有哪些权限

```
1  
2 SELECT  
3   su.id,  
4   su.username,  
5   sr.role_name,  
6   sr.role_description,  
7   sr.id AS 'role_id',  
8   sp.uri  
9 FROM  
10  sys_user su,  
11  sys_role sr,  
12  sys_user_role sur,  
13  sys_role_permission srp,  
14  sys_permission sp  
15 WHERE  
16  su.id = sur.user_id  
17  AND sr.id = sur.role_id  
18  AND srp.role_id = sr.id  
19  AND srp.permission_id = sp.id  
20  AND su.username = 'admin';
```

把原来的order服务器中的安全代码 从微服务剥离出来。

网关测试 获取令牌

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

**REQUEST**

METHOD: POST  
SCHEME: // HOST: [": PORT: ] PATH: ["/? QUERY: ]  
http://localhost:8888/oauth/token length: 33 bytes

HEADERS: Authorization: Basic cG9ydGFsX2FwcDpw3J0YWxkYXN5 Content-Type: application/x-www-form-urlencoded

BODY: { username: admin password: admin grant\_type: password scope: read }

**RESPONSE**

200 OK

HEADERS: cache-control: no-store content-type: application/json; charset=UTF-8 date: 2019 Dec 29 15:57:21 -11 pragma: no-cache transfer-encoding: chunked x-content-type-opt: nosniff

BODY: { access\_token: "1a5f6ee7-c7b7-4390-8f2a-1215ef779802", token\_type: "bearer", expires\_in: 3599, scope: "read" }

## 查看admin的权限:

对象: user @sso (本... sys\_role @sso (本... oauth\_client\_details... \*无标题 @sso (本地...)

运行 停止 解释 新建 加载 保存 另存为 美化 SQL 备注 导出

查询创建工具 查询编辑器

```
3 su.username,
4 sr.role_name,
5 sr.role_description,
6 sr.id AS 'role_id',
7 sp.uri
8 FROM
9 sys_user su,
10 sys_role sr,
11 sys_user_role sur,
12 sys_role_permission srp,
13 sys_permission sp
14 WHERE
15 su.id = sur.user_id
16 AND sr.id = sur.role_id
17 AND srp.role_id = sr.id
18 AND srp.permission_id = sp.id
19 AND su.username = 'admin';
```

信息 结果1 概况 状态

id	username	role_name	role_description	role_id	uri
1	admin	部门经理	部门经理	2	/order/saveOrder
1	admin	部门经理	部门经理	2	/order/selectOrderInfoById
1	admin	部门经理	部门经理	2	/order/list

## 通过admin用户登陆的token去测试访问地址

METHOD: POST SCHEME: // HOST: localhost PORT: 8888 PATH: /order/saveOrder

length: 37 bytes

QUERY PARAMETERS

HEADERS

- Content-Type: application/x-www-form-urlencoded
- Authorization: bearer 931206e7-1c7e-4f23-90f1-9eacR

BODY

- orderNo: 12345
- userName: smlz
- productNo: 1
- productCount: 1

RESPONSE

200 OK

HEADERS

- content-type: application/json; charset=UTF-8
- date: 2019 Dec 29 16:14:53 -1s
- transfer-encoding: chunked

BODY

```
{
  orderNo: "12345",
  userName: "smlz",
  createDt: null,
  productNo: "1",
  productCount: 1
}
```

测试张三登陆的权限<http://localhost:8888/oauth/token>

对象: \* 无标题 @sso (本地...)

运行 停止 解释 新建 加载 保存 另存为 美化 SQL 备注 导出

查询创建工具 查询编辑器

```

1 SELECT
2   su.id,
3   su.username,
4   sr.role_name,
5   sr.role_description,
6   sr.id AS 'role_id',
7   sp.uri
8 FROM
9   sys_user su,
10  sys_role sr,
11  sys_user_role sur,
12  sys_role_permission srp,
13  sys_permission sp
14 WHERE
15   su.id = sur.user_id
16 AND sr.id = sur.role_id
17 AND srp.role_id = sr.id
18 AND srp.permission_id = sp.id
19 AND su.username = 'zhangsan';

```

信息 结果1 概况 状态

id	username	role_name	role_description	role_id	uri
2	zhangsan	员工	普通员工	1	/order/selectOrderInfoById

REQUEST

METHOD: POST SCHEME: // HOST: localhost PORT: 8888 PATH: /oauth/token

length: 33 bytes

QUERY PARAMETERS

HEADERS

- Authorization: Basic cG9ydGFsX2FwcDpw3J0YXxY
- Content-Type: application/x-www-form-urlencoded

BODY

- username: zhangsan
- password: 123456
- grant\_type: password
- scope: read

RESPONSE

200 OK

HEADERS

- cache-control: no-store
- content-type: application/json; charset=UTF-8
- date: 2019 Dec 29 16:16:03 -1s
- pragma: no-cache
- transfer-encoding: chunked
- vary: content-type

BODY

```
{
  access_token: "e7eaa9ce-2b36-4e1e-bdf3-08709ad3dd26",
  token_type: "bearer",
  expires_in: 3599,
  scope: "read"
}
```

## 张三token拿到的权限:

▶ BODY ?

```
{
  aud : [
    "api-gateway",
    "product-service",
    "order-service"
  ],
  user_name : "zhangsan",
  scope : [
    "read"
  ],
  active : true,
  exp : 1577610963,
  authorities : [
    "/order/selectOrderInfoById"
  ],
  client_id : "portal_app"
}
```

lines nums

通过张三登陆的去访问不没有权限访问的url 返回403

<http://localhost:8888/order/saveOrder>

POST http://localhost:8888/order/saveOrder length: 37 bytes

▶ QUERY PARAMETERS

HEADERS ?

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

Authorization: bearer e7eaa9ce-2b36-4e1e-bdf3-0870f

+ Add header

+ Add authorization

Form

▶ BODY ?

orderNo: 12345

userName: smlz

productNo: 1

productCount: 1

+ Add form parameter

application/x-www-form-urlencoded

RESPONSE Cache Detected - E

500 Internal Server Error

HEADERS ?

content-length: 70 bytes

content-type: application/json; charset=UTF-8

pretty

▶ BODY ?

```
{
  code : "403",
  msg : "无权访问",
  time : "2019-12-29T08:22:03.282Z"
}
```

lines nums