目录

目录	
简介	
概述	
系统拓扑图	
代码结构	
数据模型	
技术选型	
依赖环境	(
后端技术	
前端技术	
后端部署	
前端部署	(
快速开始	
快速开始	
常见问题	10
项目启动失败?	10
安全认证提示:认证失败,请重新登录!	10
访问接口,提示"权限不足,拒绝访问!",相关操作步骤?	10
访问接口,提示"请求地址,禁止访问!"?	10
访问接口,提示"请求地址,拒绝访问!"?	10
资源服务器 OAuth2身份认证方式	10
集成开发	12
微服务Restful接口开发、部署、调用	12
开发	12
打包上线	15
OAuth2使用	17
OAuth2 获取 AccessToken 并调用接口	17
OAuth2 获取 AccessToken	19
密码模式(password)	19
授权码模式(authorization_code)	20
客户端模式(client_credentials)	20
接口在线调试	25
多用户认证中心	2
多用户认证中心	21
项目示例:	2
Nginx部署前端项目	28
Jenkins持续集成	30
Jenkins持续集成	30
构建后端服务	30
构建前端项目	3:

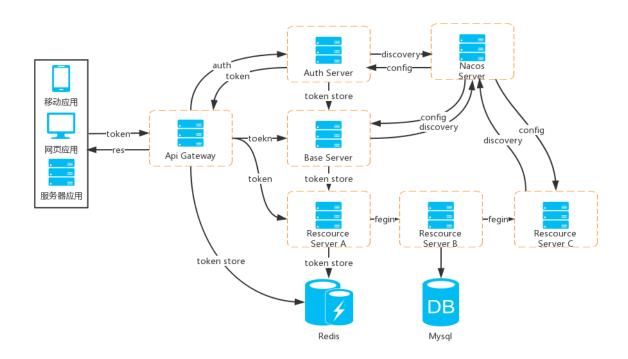
简介

概述

搭建基于OAuth2的开放平台、为APP端、应用服务提供统一接口管控平台、为第三方合作伙伴的业务对接提供授信可控的技术对接平台

- 分布式架构, Nacos(服务注册+配置中心)统一管理、Feign(RPC服务调用)
- 统一API网关(参数验签、身份认证、接口鉴权、接口调试、接口限流、接口状态、接口外网访问)
- 统一Oauth2认证协议

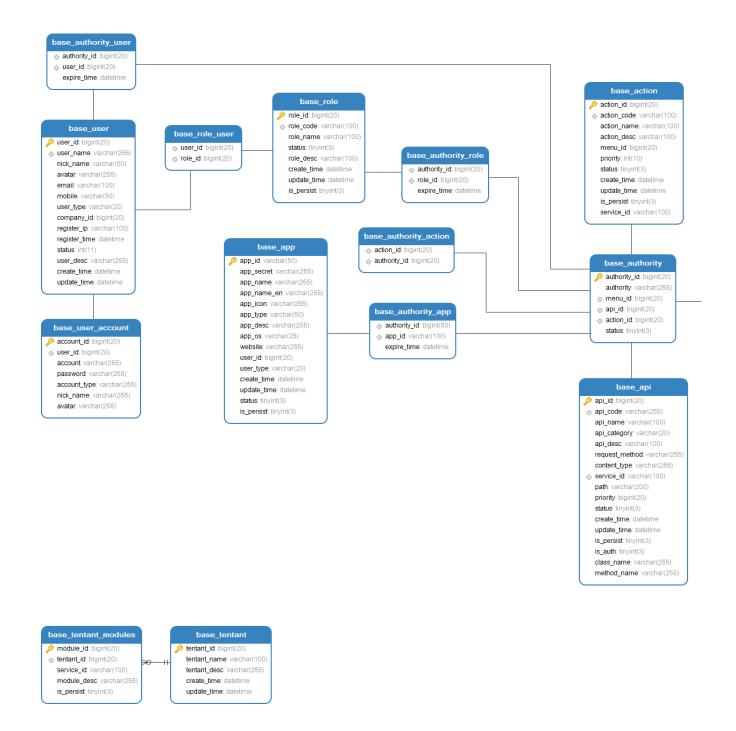
系统拓扑图

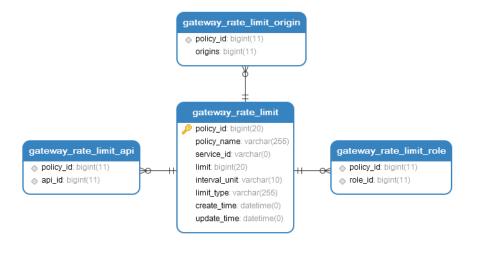


代码结构

```
open-platform
--- docs
                     -- 文档及脚本
  ├— bin
                    -- 执行脚本
                     -- 公共配置,用于导入到nacos配置中心
    - config
  ---- sql
                    -- sql文件
   - data
                     -- 增量数据
                      -- Docker部署脚本
  - docker
  --- elk
                    -- ELK部署脚本
    — influxdb
                     -- InfluxDB部署脚本
   --- monitor
                      -- Grafana部署脚本
  components
                        -- 公共组件
                   -- 提供微服务相关依赖包、工具类、全局异常解析等
   --- common-core
   --- common-starter -- SpringBoot自动配置扫描
                -- 开放平台api集成SDK(完善中...)
   --- java-sdk
  ├── tenant-starter -- 多租户模块,多数据源自动切换(完善中...)
  - migration
                      -- 多环境数据迁移-菜单、操作、路由、异构API
                      -- 平台服务
  platform
  ├── api-spring-server -- API开放网关-基于SpringCloudGateway[port = 8888](推荐)
  ├─ api-zuul-server -- API开放网关-基于Zuul[port = 8888](功能完善)
   --- base-client
               -- 平台基础服务接口
                 -- 平台基础服务器[port=8233]
    – base-server
    - generator-server -- 在线代码生成服务器[port = 5555]
    -- sba-server -- SpringBootAdmin监控服务[port = 8849]
  --- sentinel-dashboard -- Sentinel控制台服务[port = 8080]
```

数据模型





gateway_access_logs

access_id: bigint(20)
path: varchar(255)
params: text
headers: text
ip: varchar(500)
http_status: varchar(100)
method: varchar(50)
request_time: datetime(0)
response_time: datetime(0)
use_time: bigint(20)
use_time: bigint(20)
use_tagent varchar(255)
access_desc varchar(255)
authentication: text
server_ip: varchar(255)

gateway_route

path: varchar(255)
service_id: varchar(255)
service_id: varchar(255)
url: varchar(255)
strip_prefix: tinyint(3)
retryable: tinyint(3)
status: tinyint(3)
route_desc: varchar(255)
is_persist: tinyint(3)

技术选型

依赖环境

- nodejs
- redis(缓存服务)
- rabbitMq(消息服务)
- nacos(服务注册和配置管理)
- sentinel(流量控制、熔断降级、系统负载保护)
- influxdb(时序数据库)
- mysql(数据库)

后端技术

- maven
- spring cloud
- spring-security-oauth2
- mybatis
- mybatis-plus
- swagger2
- ...

前端技术

- vue.js
- iview (前端框架)

后端部署

- jar包形式部署,使用undertow代替原有tomcat
- jenkins持续集成

前端部署

• nginx

快速开始

快速开始

本项目基于SpringCloud打造的分布式快速开发框架. 需要了解SpringCloud,SpringBoot,SpringSecurity,分布式原理。

- 1. 准备环境
 - Java1.8 (v1.8.0_131+)
 - Nacos服务注册和配置中心 (v1.0.0+) 阿里巴巴Nacos
 - Sentinel流量控制、熔断降级、系统负载保护 (v1.6.0+) 阿里巴巴Sentinel
 - Redis (v3.2.00+)
 - RabbitMq (v3.7+) (需安装rabbitmq_delayed_message_exchange插件下载地址)
 - Mysql (v5.5.28+)
 - InfluxDB (v1.7.9+)
 - ELK (v7.4.1+)
 - Maven (v3+)
 - Nodejs (v10.14.2+)
- 2. 执行创建数据库open_platform并执行sql脚本
 - docs/sql/oauth2.sql
 - docs/sql/base.sql
 - docs/sql/gateway.sql
 - docs/sql/msg.sql
 - docs/sql/quartz.sql && scheduler.sql
 - docs/sql/org.sql
 - docs/sql/data/update.sql

执行创建数据库open_platform_user并执行sql脚本

docs/sql/user.sql

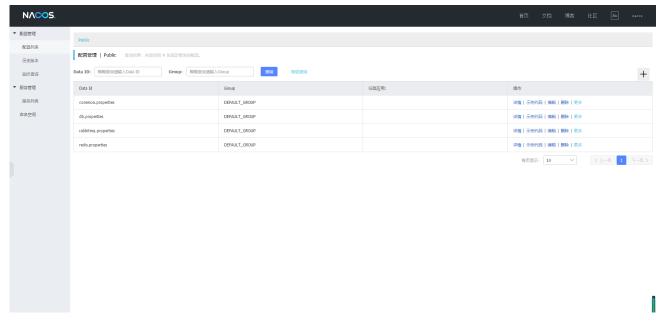
执行创建数据库open_platform_payment并执行sql脚本

docs/sql/payment.sql

执行创建数据库open_platform_comment并执行sql脚本

- docs/sql/comment.sql
- 3. 启动nacos服务发现&配置中心,新建公共配置文件
 - 访问 http://localhost:8848/nacos/index.html
 - 新建配置文件及导入 (根据需要导入docs/config下配置文件)
 - docs/config/db.properties > db.properties
 - docs/config/rabbitmq.properties > rabbitmq.properties
 - docs/config/redis.properties > redis.properties
 - docs/config/common.properties > common.properties

如图:



4. 修改主pom.xml 初始化maven项目

maven clean install

本地启动,默认不用修改

- <!--Nacos配置中心地址-->
- <config.server-addr>127.0.0.1:8848</config.server-addr>
- <!--Nacos配置中心命名空间,用于支持多环境.这里必须使用ID,不能使用名称,默认为空-->
- <config.namespace></config.namespace>
- <!--Nacos服务发现地址-->
- <discovery.server-addr>127.0.0.1:8848</discovery.server-addr>
- <!--Nacos服务发现命名空间,用于支持多环境.这里必须使用ID,不能使用名称,默认为空-->
- <discovery.namespace></discovery.namespace>
- <!--Logstash日志收集地址-->
- logstash.destination>127.0.0.1:5000

5. 本地启动(按顺序启动)

- i. [必需]BaseApplication(平台基础服务)
- ii. [必需]UaaAdminApplication(平台用户认证服务器)
- iii. [必需]GatewaySpringApplication(推荐)或GatewayZuulApplication

访问 http://localhost:8888

4.[非必需]SpringBootAdmin(监控服务器)(非必需)

访问 http://localhost:8849

6. 前端启动

npm install npm run dev

访问 http://localhost:8080

7. 项目打包部署

• maven多环境打包,替换变量

mvn clean install package -P {dev|test|uat|online}

• 项目启动

./docs/bin/startup.sh {start|stop|restart|status} base-server.jar ./docs/bin/startup.sh {start|stop|restart|status} uaa-admin-server.jar ./docs/bin/startup.sh {start|stop|restart|status} api-spring-server.jar

8. docker部署

- 配置DOCKER私服仓库
- maven多环境打包,替换变量.并构建docker镜像

clean install package -P {dev|test|uat|online} dockerfile:build

• 启动docker镜像

docker run -d -e JAVA_OPTS="-Xms128m -Xmx768m" -p 8233:8233 --name base-server platform/base-server:3.0.0 docker run -d -e JAVA_OPTS="-Xms128m -Xmx768m" -p 8211:8211 --name uaa-admin-server platform/uaa-admin-server:3.0.0 docker run -d -e JAVA_OPTS="-Xms128m -Xmx768m" -p 8888:8888 --name api-spring-server platform/api-spring-server:3.0.0

常见问题

项目启动失败?

- 1. 先检查nacos服务注册发现是否启动成功。
- 2. 检查相关公共配置是否完全导入到nacos中。
- 3. 新建项目启动失败, 检查是否引入jar冲突,并排除相关依赖.

安全认证提示:认证失败,请重新登录!

- 1. 网关作为第一道防火墙, 可以通过 管理后台 API网关 API列表 设置接口是否需要"身份认证".
- 2. 业务资源服务器作为最后一道防火墙. 开发时,可以继承 ResourceServerConfigurerAdapter 指定具体资源权限验证或放行策略

```
http.sessionManagement ().sessionCreationPolicy (SessionCreationPolicy.IF\_REQUIRED)\\
        .and()
        .authorizeRequests()
        // 监控端点内部放行
        .requestMatchers(EndpointRequest.toAnyEndpoint()).permitAll()
        // feign访问或无需身份认证
         .antMatchers(
             "/authority/access",
             "/authority/app",
             "/developer/login"
        ).permitAll()
         .anyRequest().authenticated()
        .and()
        //认证鉴权错误处理,为了统一异常处理。每个资源服务器都应该加上。
         .exceptionHandling()
         .accessDeniedHandler(new OpenAccessDeniedHandler())
         .authenticationEntryPoint(new OpenAuthenticationEntryPoint())
         .and()
         .csrf().disable();
```

或开启@EnableWebSecurity使用注解方式配置更细粒度权限验证。 学习更多 spring-security 官网

访问接口,提示"权限不足,拒绝访问!",相关操作步骤?

- 用户权限 (需用户登录)
 - 配置功能菜单与资源权限(开发阶段): 系统管理 > 功能菜单 > 功能按钮 > 接口授权 > 勾选相关接口 > 保存
 - 分配功能菜单(使用阶段): 系统管理 > 角色管理 > 编辑 > 分配权限 > 勾选菜单和功能按钮 > 保存
 - 分配用户角色(使用阶段): 系统管理 > 用户管理 > 编辑 > 分配角色 > 保存
- 应用权限(不需用户登录,仅用于client_credentials模式,直接调用接口) 应用管理 > 编辑 > 开发信息 > 勾选客户端模式 > 功能授权 > 勾选相关接口 > 保存

访问接口,提示"请求地址,禁止访问!"?

1. API网关 - API列表 - 查看API接口信息:状态是否被禁用

访问接口,提示"请求地址,拒绝访问!"?

1. API网关 - API列表 - 查看API接口信息:是否可公开访问

资源服务器 OAuth2身份认证方式

1. (默认)远程校验解析token. 缺点:需创建客户端信息,http方式,性能较差.

 $Resource Server Token Services \ token Service = Open Helper. build Remote Token Services (Open Common Properties) \\$

2. (默认)jwt校验解析token. 缺点:需提供jwt签名或密钥,生存token过长(cookie有时存放不下),且base64加密无法存放敏感数据,不方便拓展.

 $Resource Server Token Services \ token Service = Open Helper. build Jwt Token Services (Open Common Properties) \\ Properties (Open Common Proper$

3. (推荐使用-opencloud)redis校验解析token. 无需创建任何客户端和密钥,读取性能优,可存放复杂的认证信息. 缺点:必须使用同一个redisDbIndex.

ResourceServerTokenServices tokenService = OpenHelper.buildRedisTokenServices(RedisConnectionFactory redisConnectionFactory)

集成开发

微服务Restful接口开发、部署、调用

接口调用方式分为:

- 外部调用 通过网关统一入口调用,通过oauth2协议获取access_token,统一验证调用接口权限,验证参数签名。
- 内部调用 feign+ribbon方式,负载到目标服务,由微服务自身验证权限,无需验证参数签名。

开发

- 1.微服务内部调用, feign方式规范。
- 创建maven项目pom.xml: test-client

```
<?xml version="1.0" encoding="UTF-8"?>
xmlns="http://maven.apache.org/POM/4.0.0"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <parent>
    <groupId>com.opencloud</groupId>
    <artifactld>platform</artifactld>
    <version>3.0.0</version>
  </parent>
 <modelVersion>4.0.0</modelVersion>
 <artifactId>test-client</artifactId>
 <version>1.0.0-SNAPSHOT</version>
 <packaging>jar</packaging>
  <description>测试服务接口</description>
  <dependencies>
    <dependency>
      <groupId>com.opencloud</groupId>
      <artifactId>common-core</artifactId>
      <version>${opencloud.common.version}</version>
    </dependency>
  </dependencies>
  <bul>d
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins/groupId>
        <artifactId>maven-source-plugin</artifactId>
        <version>${maven-source-plugin.version}</version>
        <executions>
          <execution>
             <id>attach-sources</id>
             <goals>
               <goal>jar</goal>
             </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

• 创建资源服务feign接口 ITestClient

```
public interface ITestClient{
    /**
    * 你好
    * @return
    */
    @PostMapping("/say/hi")
    ResultBody<String> sayHi(
        @RequestParam String message
    );
}
```

2.创建test资源服务器

- 创建新maven项目pom.xml (无需建在open-platform目录下,可独立创建项目)
- 引入test-client接口依赖

```
<?xml version="1.0" encoding="UTF-8"?>
project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <!-- 依赖父项目,使用父项目依赖、变量、插件,需(install)到本地仓库或发布(deploy)到私服仓库 -->
    <groupId>com.opencloud</groupId>
    <artifactId>parent</artifactId>
    <version>3.0.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <packaging>jar</packaging>
  <artifactId>test-server</artifactId>
  <description>新建资源服务器</description>
  <dependencies>
     <!-- 引入公共包,需(install)到本地仓库或发布(deploy)到私服仓库 -->
    <dependency>
      <groupId>com.opencloud</groupId>
      <artifactId>common-starter</artifactId>
      <version>${opencloud.common.version}</version>
    </dependency>
    <!-- 引入test客户端接口 -->
     <dependency>
      <groupId>com.opencloud</groupId>
      <artifactId>test-client</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
  <build>
    <!-- 打包名称 -->
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-maven-plugin</artifactId>
         <version>${spring-boot.version}</version>
         <executions>
           <execution>
             <goals>
               <goal>repackage</goal>
             </goals>
           </execution>
         </executions>
      </plugin>
      <!-- docker打包插件 -->
      <plugin>
         <groupId>com.spotify</groupId>
         <artifactId>dockerfile-maven-plugin</artifactId>
         <version>${dockerfile-maven-plugin.version}/version>
         <configuration>
           <repository>${docker.image.prefix}/${project.artifactId}</repository>
             <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
           </buildArgs>
         </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

```
server:
 port: 8266
spring:
  application:
    name: ${artifactId}
  cloud:
    nacos:
       config:
         namespace: ${config.namespace}
         refreshable-dataids: common.properties
         server-addr: ${config.server-addr}
         shared-dataids: common.properties,db.properties,redis.properties,rabbitmq.properties
         server-addr: ${discovery.server-addr}
  main:
    allow-bean-definition-overriding: true
  mvc:
    throw-exception-if-no-handler-found: true
    add-mappings: false
  profiles:
    active: ${profile.name}
management:
  endpoints:
    web:
       exposure:
         include: '*'
opencloud:
  swagger2:
    description: 平台消息服务
    enabled: true
    title: 平台消息服务
```

4.创建启动类 TestApplication.java

```
//开启feign
@EnableFeignClients
// 开启服务发现
@EnableDiscoveryClient
@SpringBootApplication
public class TestApplication {
   public static void main(String[] args) {
        SpringApplication.run(TestApplication.class, args);
    }
}
```

5.创建ResourceServerConfiguration.java 资源服务器安全配置

```
.antMatchers("/say/hi").permitAll()

// 指定监控访问权限
.requestMatchers(EndpointRequest.toAnyEndpoint()).permitAll()
.anyRequest().authenticated()
.and()

//认证鉴权错误处理,为了统一异常处理。每个资源服务器都应该加上。
.exceptionHandling()
.accessDeniedHandler(new OpenAccessDeniedHandler())
.authenticationEntryPoint(new OpenAuthenticationEntryPoint())
.and()
.csrf().disable();

}
```

6.实现test-client feign接口

```
@RestController
@Api(value = "测试", tags = "测试")
public class TestController implements ITestClient {
    @ApiOperation(value = "打招呼",notes = "招呼")
    @PostMapping("/say/hi")
    @Override
    public ResultBody<String> sayHi(@RequestParam String message) {
        return ResultBody.ok().data("你好!");
    }
}
```

- 7.服务消费者-feign调用 在使用方项目中创建feign使用类,继承服务提供方接口
- 继承ITestClient即可.
- @FeignClient(value = "test-server") 指定服务提供者

```
@Component
@FeignClient(value = "test-server")
public interface TestServiceClient extends ITestClient {
}

//feign 接口注入与调用
@Autowired
private TestServiceClient testServiceClient;

public void customer(){
    testServiceClient.sayHi("你还啊");
}
```

打包上线

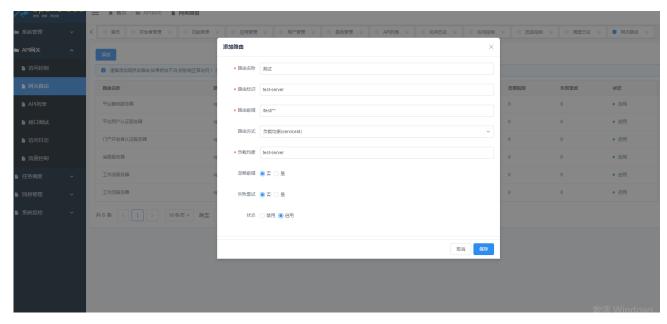
1. 使用maven命令

```
mvn clean install package -P {dev|test|uat|online}
```

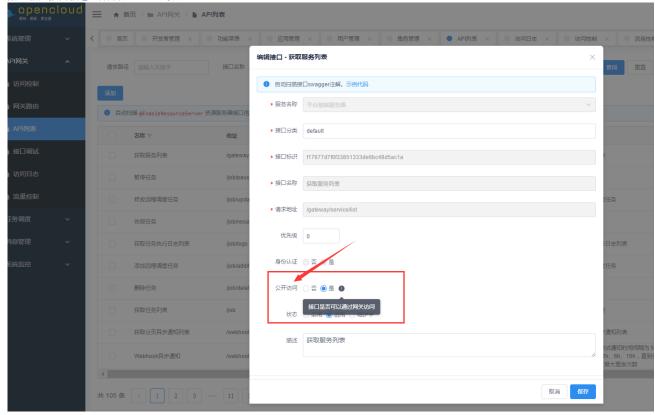
2. 启动服务jar包 将./docs/bin/startup.sh上传到服务器

```
./startup.sh {start|stop|restart|status} server.jar
```

3. 配置网关路由(如果该服务仅作为内部服务使用,就不要配置路由信息了) 后台管理 - API网关 - 网关路由



4. 配置相关接口信息. 后台管理 - API列表

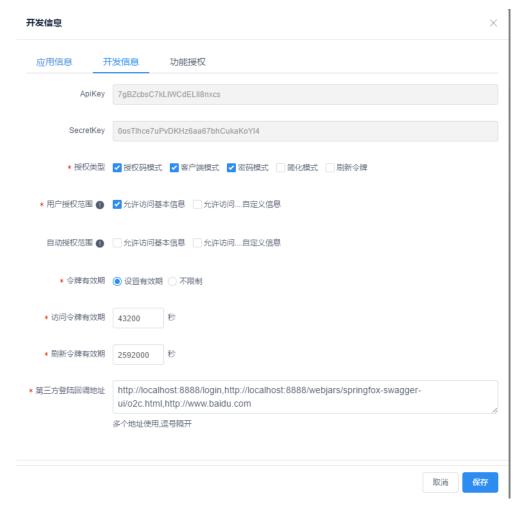


OAuth2使用

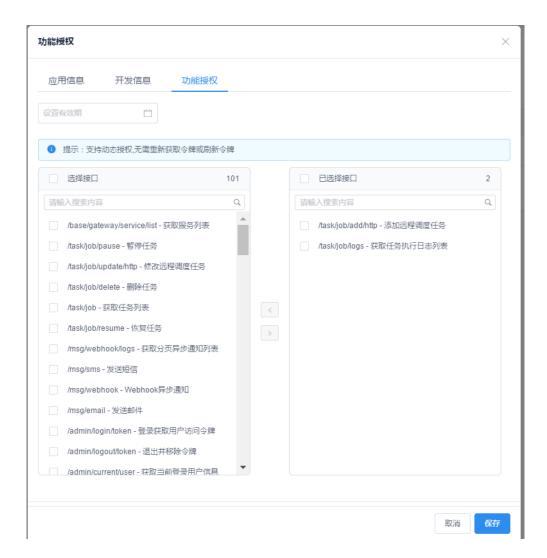
OAuth2 获取 AccessToken 并调用接口



- 创建一个新应用 _
- 配置开发信息 根据实例需求勾选,授权方式



• 接口授权,勾选需要授权的接口 访问api网关时,将验证该权限。 否则提示权限不足



OAuth2 获取 AccessToken

• 使用postman测试

业用信息

 $ApiKey(client_id): \ 7gBZcbsC7kLIWCdELIl8nxcs$

 $SecretKey(client_secret): \ 0 os TIhce 7 u Pv DKHz 6 aa 67 bh Cuka Ko YI4$

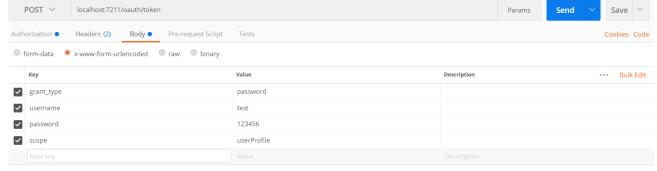
密码模式(password)

http://localhost:8211/oauth/token

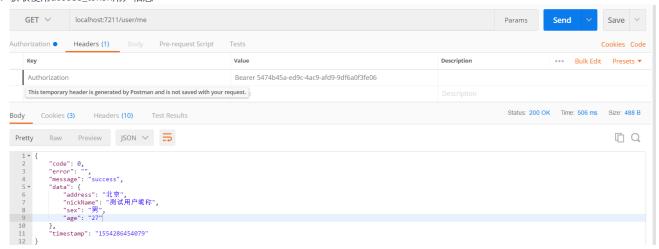
1. 首先设置请求头basic方式配置client_id和client_secret



2. 输入用户username和password



3. 获取使用access_token用户信息

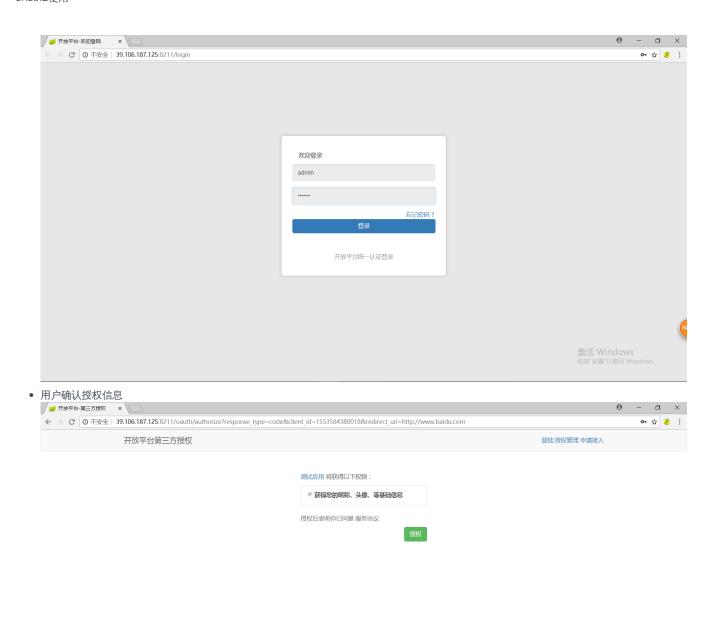


授权码模式(authorization_code)

• 浏览器访问,引导用户跳转到登录页(适合sso单点登录系统接入)

 $http://localhost:8211/oauth/authorize?response_type=code\&client_id=1553588629729\&redirect_uri=http://www.baidu.com.pdf.$

• 跳转到登录页,输入系统用户登录信息



激活 Windows 转到"设置"以激活 Windows。

• 重定向到回调地址,获得code

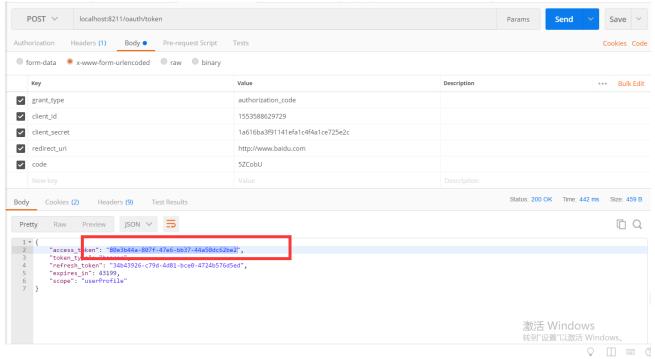




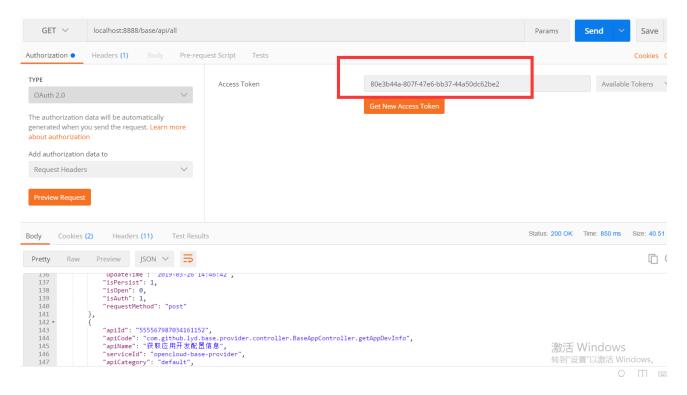




• 使用postman通过code获取access_token,



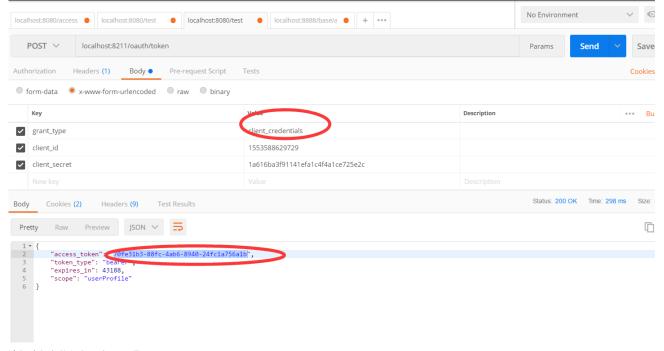
• 使用access_token获取已授权资源



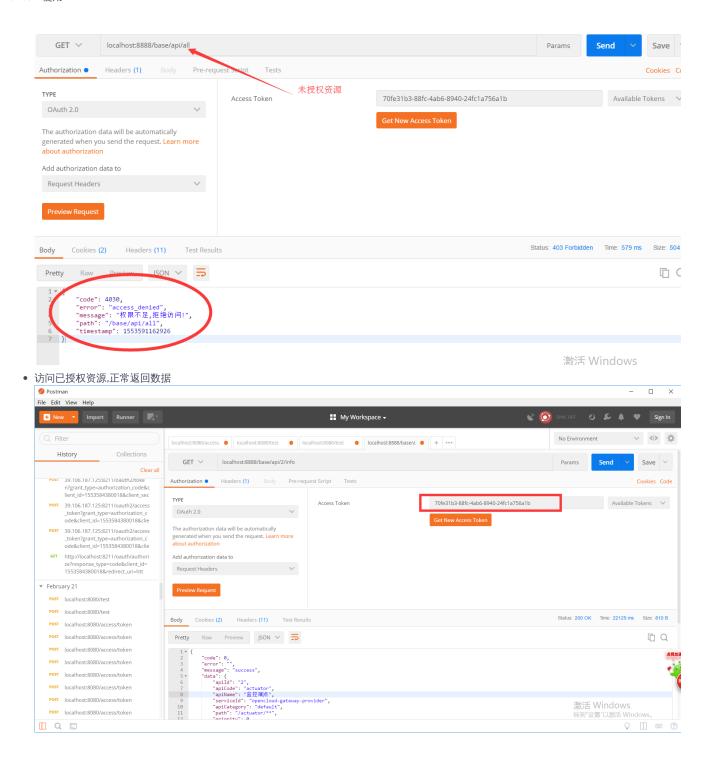
客户端模式(client_credentials)

 $\label{localinear} $$ $ \true = client_credentials & client_id = 1553588629729 & client_secret = 1a616ba3f91141efa1c4f4a1ce725e2c & client_credentials & client_id = 1553588629729 & client_secret = 1a616ba3f91141efa1c4f4a1ce725e2c & client_credentials & client_id = 1553588629729 & client_secret = 1a616ba3f91141efa1c4f4a1ce725e2c & client_credentials & client_id = 1553588629729 & client_credentials & client_cred$

获取客户端token



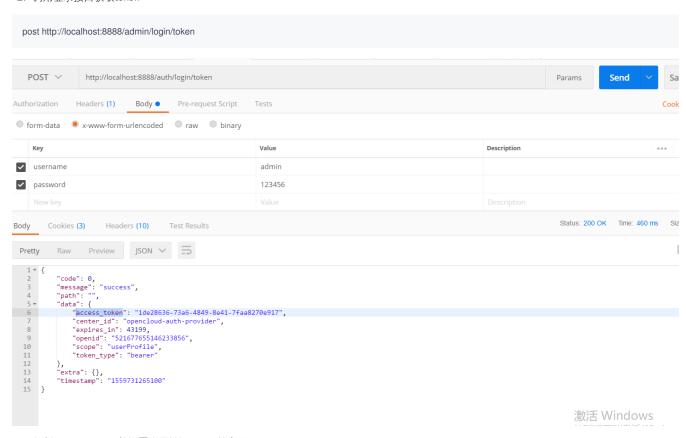
• 访问未授权资源提示权限不足!



接口在线调试

基于swagger2+swagger-bootstrap-ui 官方文档

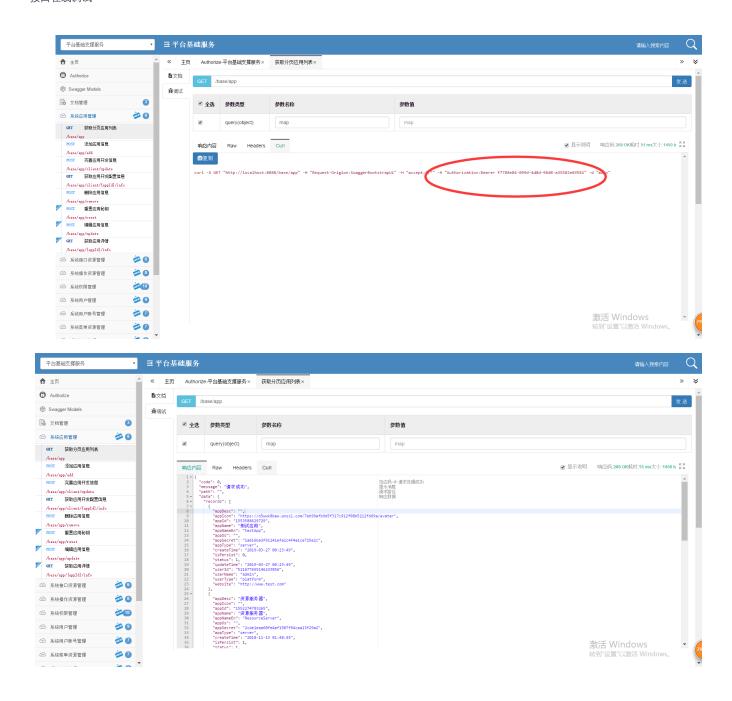
1. 调用登录接口获取token



2. 复制access_token 并设置全局认证token 输入值: Bearer {access_token}



3. 访问接口



多用户认证中心

多用户认证中心

- 不同点:用户表都是完全独立的,登录方式也是多种多样(密码登录,手机验证码登录,等等...)
- 共同点:每个认证中心,共享oauth_client_details(客户端信息)
- 优点:
- 1. 从项目层隔离开,统一token协议,在向下游微服务传递时,使用同一方式读取token信息.
- 2. app应用直接具备第三方授权登录能力,例如:微信,qq等第三方登录.方便后期拓展。
- 3. 拓展更丰富的个人中心功能

项目示例:

- uaa-admin-server 平台用户认证服务器
- uaa-portal-server 门户开发者认证服务器

Nginx部署前端项目

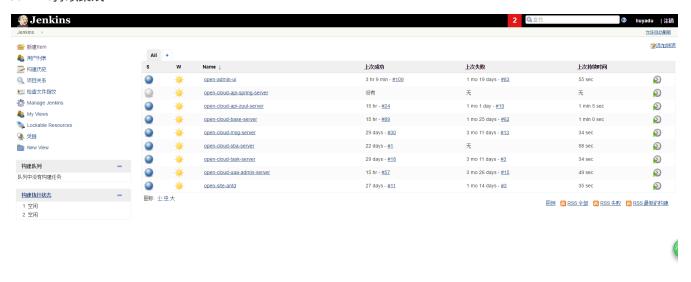
- 安装nginx服务
- 配置nginx.conf

```
user root;
worker_processes 1;
#error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;
      logs/nginx.pid;
#pid
events {
 worker_connections 1024;
http {
  include mime.types;
  default_type application/octet-stream;
  log_format main '$remote_addr - $remote_user [$time_local] "$request" '
            '$status $body_bytes_sent "$http_referer" '
            ""$http_user_agent" "$http_x_forwarded_for"";
  access_log logs/access.log main;
  sendfile
           on;
  #tcp_nopush on;
  #keepalive_timeout 0;
  keepalive_timeout 65;
  #开启或关闭gzip on off
  gzip on;
  #不使用gzip IE6
  gzip_disable "msie6";
  #gzip压缩最小文件大小,超出进行压缩(自行调节)
  gzip_min_length 100k;
  #buffer 不用修改
  gzip_buffers 4 16k;
  #压缩级别:1-10,数字越大压缩的越好,时间也越长
  gzip_comp_level 3;
  # 压缩文件类型
  gzip_types text/plain application/x-javascript text/css application/xml text/javascript application/x-httpd-php image/jpeg image/gif image/png;
  #跟Squid等缓存服务有关,on的话会在Header里增加 "Vary: Accept-Encoding"
  gzip_vary off;
  underscores_in_headers on;
  server {
    listen
           80;
    server_name localhost;
    #charset koi8-r;
    access_log logs/host.access.log main;
    #默认目录
    location / {
      root html;
      index index.html;
```

```
#vue二级目录代理
   location /admin {
    alias /root/www/admin;
    index index.html;
    try_files $uri $uri/ /index.html last;
   #api网关代理
   location /api {
    rewrite ^/api/(.*)$ /$1 break;
    proxy_pass http://127.0.0.1:8888;
    proxy_set_header Host $host;
    proxy_set_header User-Agent $http_user_agent;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header authorization $http_authorization;
   #error_page 404
                          /404.html;
   # redirect server error pages to the static page /50x.html
   error_page 500 502 503 504 /50x.html;
   location = /50x.html {
    root html;
}
```

Jenkins持续集成

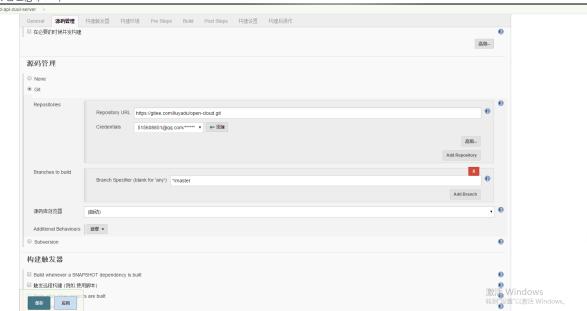
Jenkins持续集成



• 安装相关插件

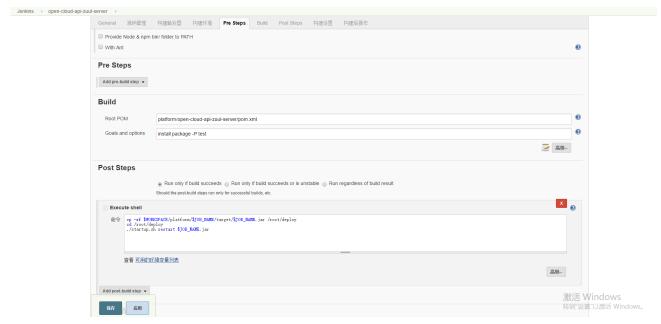
构建后端服务

- 1. 安装maven插件
- 2. 构建一个maven项目,项目名作为任务名称
- 3. 配置代码拉取地址(git/svn)



激活 Windows 转到"设置"以激活 Windows。 生成页面:2019-7-12下午03时09930号 RESTAPI Jenkins ver. 2.178

4. 配置构建具体项目pom.xml路径



5. 配置构建完成后,执行命令(这里执行的本机命令,可使用ssh管道执行远程脚本)构建成功执行shell命令

cp -rf \$WORKSPACE/platform/\$JOB_NAME/target/\$JOB_NAME.jar /root/deploy cd /root/deploy ./startup.sh restart \$JOB_NAME.jar

构建前端项目

- 安装nodejs插件
- 构建一个自由风格的软件项目
- 项目名作为任务名称
- 配置代码拉取地址(git/svn)
- 配置执行命令

