

# 闪聚支付 第2章 讲义-对接SaaS

## 1 对接SaaS

### 1.1 基础概念

#### 1.1.1 SaaS

SaaS是Software-as-a-Service（软件即服务）的简称，它是一种通过互联网提供软件服务的模式，与传统软件相比有如下几点区别：

- 1、SaaS软件不再是用户向软件供应商定制软件或进行二次开发，而是供应商将软件部署在自己的服务器上并通过互联网提供在线服务。
- 2、软件供应商负责搭建一切网络设备、软硬件运行平台等基础设施，并全权负责运营和维护软件。
- 3、用户根据实际需要需要通过互联网订购所需要的软件服务，按照订购服务的多少和时间长短支付费用。
- 4、用户不需要一次性支付很大一笔软件定制费，只需支付一些租用费用就可以使用软件，风险非常低，当发现软件不满足需求或不适合公司管理模式可以停止续租。

对于许多中小型企业来说，SaaS模式是采用先进技术的最好途径，它减少了企业购买、构建和维护基础设施和应用程序的成本，大大降低了软件费用。



云计算的三种模式：

云计算的发展至今有十几年的历史，如今云计算得到了广泛的应用，具体包括三个层次：

IaaS: Infrastructure-as-a-Service（基础设施即服务），也叫Hardware-as-a-Service，将计算机硬件资源打包对外提供服务，比如云主机、云存储等。

PaaS: Platform-as-a-Service（平台即服务），也叫中间件服务，比如：MySQL数据库服务、ElasticSearch搜索服务等。SaaS: Software-as-a-Service（软件即服务），提供具体应用软件服务，比如：CRM系统，电商平台等。

下图是一个云平台的架构图，每层列出了比较典型的服务内容：



### 1.1.2 多租户

当一个使用SaaS模式部署的软件同时有多个企业用户租用时，每一个企业都是独立的租用者，我们通常称他为：租户(**tenant**)；同时有多个租用者，那就是多租户(**multi-tenant**)。多租户 (Multi-tenant)是SaaS最重要的核心概念和关键技术。

什么是租户？

一个“组织”在某软件平台上购买了部分软件服务(权限集合)，这个“组织”就是这个软件平台的“租户”。“组织”就是指人们为实现一定的目标，互相协作结合而成的集体或团体，如某企业、某学校、某机构、某商户、甚至某家庭。

什么是租户类型？

闪聚支付平台作为一个SaaS平台，提供多租户管理，具有相同业务的为同一类租户，比如：“商户”是一类租户，“XX超市”则是一个具体的商户（租户）。

什么是用户？

“用户”是“组织”(租户)内成员，是软件平台的实际使用者，使用者凭身份（用户名）、凭证(密码)登入平台。

“用户”可以在其所在“租户”内新建其它用户，并在“租户”购买的权限集合范围内对其它用户授权，称之为用户、权限“自我管理”。

整个软件平台被很多个租户共同使用，引入“多租户”的用户架构是为了让组织得到用户、权限“自我管理”的支持，并与其它“租户”的数据隔离。

下图是闪聚支付平台的多租户结构图：

闪聚支付平台由支付系统、运营系统、统计分析系统、员工管理系统等模块组成，租户根据自己的需求购买平台的服务。比如：以支付为主的租户，他需要购买支付系统、员工管理系统、统计分析系统；需要运营管理的租户则需要购买运营系统。

如何购买服务呢？

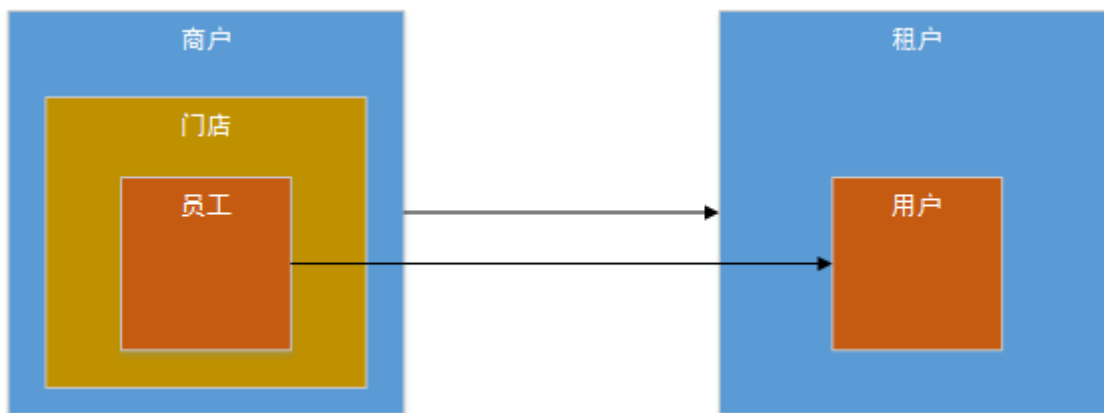
闪聚支付平台将平台的系统功能集合作成一个一个的套餐，比如：“套餐A”包括支付系统、员工管理系统、统计分析系统，以支付为主的租户购买套餐A就拥有了支付系统、员工管理系统、统计分析系统的使用权限。



## 1.2 对接SaaS步骤

### 1.2.1 业务模型

商户平台应用与SaaS平台模型的对应关系如下：



左侧为商户平台，右侧为SaaS平台。

商户平台的商户对应SaaS平台的租户。

商户平台的员工对应SaaS平台的用户。

门店是什么？

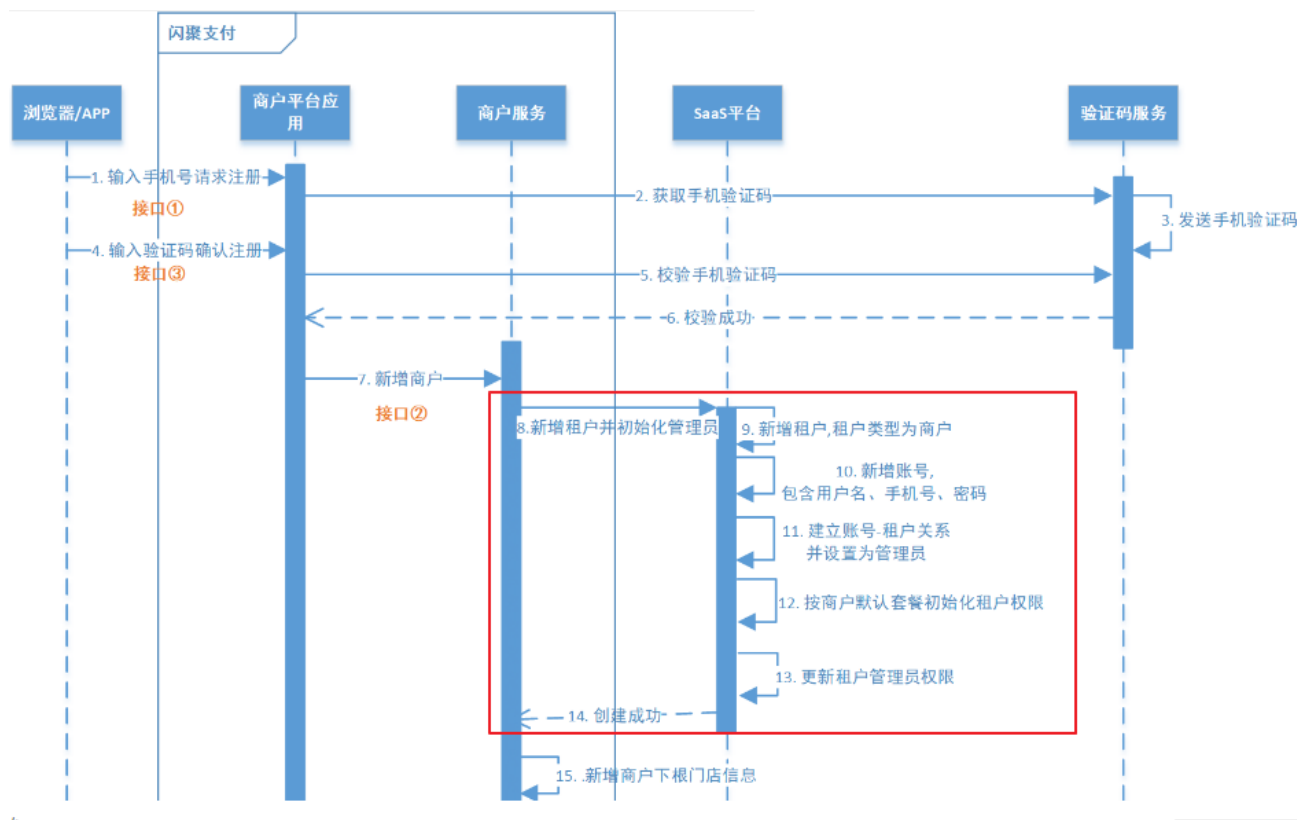
门店是商户平台中员工的组织机构，比如：本商户是一个大型超市，该大型超市在很多地方有自己的连锁店，这个连锁店就是门店，每个连锁店都有自己的员工。

门店和SaaS平台有对应的模型吗？

门店是商户平台由于经营需要所设立的组织机构，在SaaS平台中没有对应模型。

## 1.2.2 接入步骤

一个商户注册相当于一个租户在SaaS平台注册，下图展示了商户注册时商户服务与SaaS的交互流程：



商户注册，调用SaaS平台的新增租户与用户接口，每个步骤如下：

#### 1、新增租户（上图第9步）

在商户平台新增商户。

调用SaaS系统的接口新增租户。

#### 2、新增用户（上图第10-11步）

在商户平台 新增员工。

调用SaaS的接口新增用户，且自动设置用户和租户的绑定关系，并将该用户设置为该租户的管理员。

#### 3、初始化租户权限（上图第12步）

商户注册成功为该租户设置默认权限。

#### 4、更新管理员的权限（上图第13步）

为第2步新增的账号分配管理员权限，管理员权限即是租户的权限。

商户平台 实际接入SaaS开发步骤如下：

#### 1、部署SaaS系统

SaaS系统是闪聚支付平台独立的系统，我们需要部署SaaS系统并实现商户平台与其对接。

#### 2、商户注册后调用SaaS系统的接口完成上边流程的对接

##### 1) 商户平台完成的功能如下：

新增商户

新增员工

新增门店

为门店管理员功能

##### 2) 调用SaaS系统的接口完成的功能如下：

新增租户

新增用户

自动设置用户和租户的绑定关系，并将该用户设置为该租户的管理员

为该租户设置默认权限

分配管理员权限

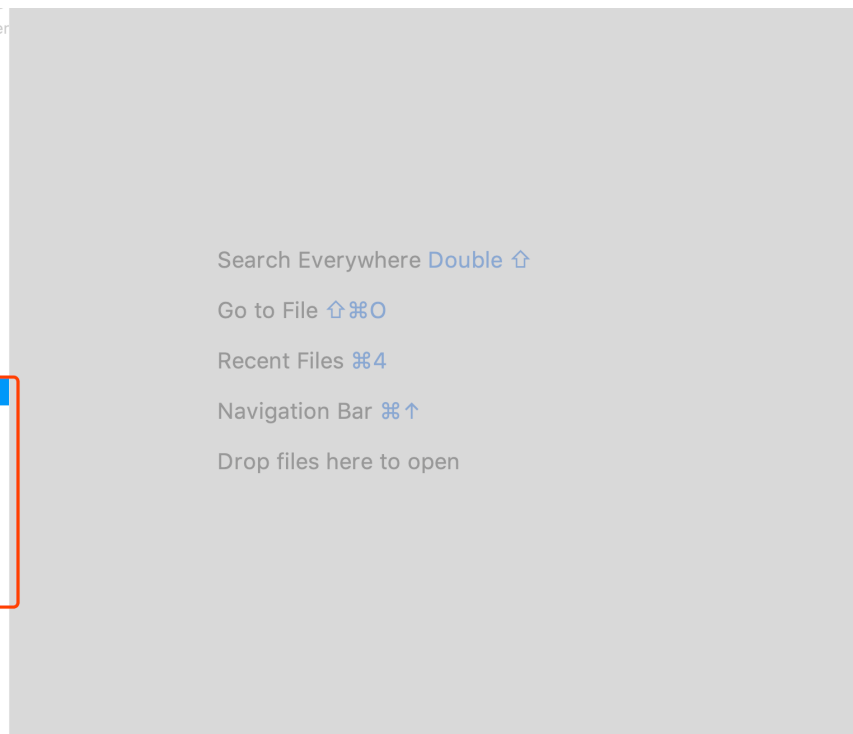
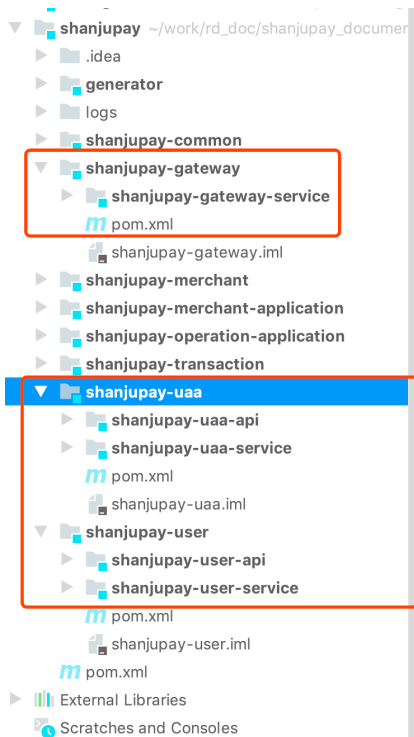
## 1.3 部署SaaS系统

### 1.3.1 初始化数据

执行“资料”下的“shanjupay\_saas.sql”脚本导入SaaS系统数据库。

## 1.3.2 部署服务

1. 复制shanjupay-gateway、shanjupay-uaa、shanjupay-user到shanjupay父工程
2. 添加上述三个Module
3. 完成后，整体目录结构如下：



## 1.3.3 完善配置

### 1.3.3.1 网关服务

1. 在Nacos上添加jwt.yaml配置，Group: COMMON\_GROUP，如下图所示

```
siging-key: shanju123
```

\* Data ID:


\* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 : 

```
1 signing-key: shanju123
```

2. 将授课资料中的gateway-service.yaml配置添加到Nacos上，Group：SHANJUPAY\_GROUP

\* Data ID:


\* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 : 

```
1 zuul:
2   retryable: true
3   add-host-header: true
4   ignoredServices: "*"
5   sensitiveHeaders: "*"
6   routes:
```

配置内容参考“资料”文件夹。

1. 修改shanjupay-gateway-service工程下的bootstrap.yml中namespace为自己Nacos中的namespace id



```
server:
  port: 56010 #启动端口 命令行注入
  max-http-header-size: 100KB

nacos:
  server:
    addr: 127.0.0.1:8848

spring:
  application:
    name: gateway-service
  main:
    allow-bean-definition-overriding: true # Spring Boot 2.1 需要设定
  cloud:
    nacos:
      discovery:
        server-addr: ${nacos.server.addr}
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        cluster-name: DEFAULT
      config:
        server-addr: ${nacos.server.addr} # 配置中心地址
        file-extension: yaml
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        group: SHANJUPAY_GROUP # 聚合支付业务组
        ext-config:
          -
            refresh: true
            data-id: jwt.yaml # jwt配置
            group: COMMON_GROUP # 通用配置组
```

### 1.3.3.2 UAA服务

1. 在Nacos中添加uaa-service.yaml配置，Group：SHANJUPAY\_GROUP

```
# 覆盖spring-boot-http.yaml的项目
server:
  servlet:
    context-path: /uaa

# 覆盖spring-boot-starter-druid.yaml的项目
spring:
  datasource:
    druid:
      url: jdbc:mysql://127.0.0.1:3306/shanjupay_uaa?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&useSSL=false
      username: root
      password: yourpassword
```

2. 修改shanjupay-uaa-service工程下的bootstrap.yml中namespace为自己Nacos中的namespace id



```
server:
  port: 56020 #启动端口 命令行注入
  max-http-header-size: 100KB

nacos:
  server:
    addr: 127.0.0.1:8848

spring:
  application:
    name: uaa-service
  main:
    allow-bean-definition-overriding: true # Spring Boot 2.1 需要设定
  cloud:
    nacos:
      discovery:
        server-addr: ${nacos.server.addr}
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        cluster-name: DEFAULT
      config:
        server-addr: ${nacos.server.addr} # 配置中心地址
        file-extension: yaml
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        group: SHANJUPAY_GROUP # 聚合支付业务组
        ext-config:
          -
            refresh: true
            data-id: spring-boot-http.yaml # spring boot http配置
            group: COMMON_GROUP # 通用配置组
```

### 1.3.3.3 统一账户服务

1. 在Nacos中添加user-service.yaml配置，Group：SHANJUPAY\_GROUP

```
# 覆盖spring-boot-http.yaml的项目
server:
  servlet:
    context-path: /user

# 覆盖spring-boot-starter-druid.yaml的项目
spring:
  datasource:
    druid:
      url: jdbc:mysql://127.0.0.1:3306/shanjupay_user?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&useSSL=false
      username: root
      password: yourpassword

# 覆盖spring-boot-mybatis-plus.yaml的项目
mybatis-plus:
  typeAliasesPackage: com.shanjupay.user.entity
  mapper-locations: classpath:com/shanjupay/*/mapper/xml/*.xml

sms:
  url: "http://localhost:56085/sailing"
  effectiveTime: 6000
```

2. 修改shanjupay-user-service工程下的bootstrap.yaml中namespace为自己Nacos中的namespace id

```
server:
  port: 56030 #启动端口 命令行注入

nacos:
  server:
    addr: 127.0.0.1:8848

spring:
  application:
    name: user-service
  main:
    allow-bean-definition-overriding: true # Spring Boot 2.1 需要设定
  cloud:
    nacos:
      discovery:
        server-addr: ${nacos.server.addr}
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        cluster-name: DEFAULT
      config:
        server-addr: ${nacos.server.addr} # 配置中心地址
        file-extension: yaml
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8 # 默认开发环境
        group: SHANJUPAY_GROUP # 聚合支付业务组
        ext-config:
          refresh: true
          data-id: spring-boot-http.yaml # spring boot http配置
          group: COMMON_GROUP # 通用配置组
```

## 1.4 对接SaaS代码实现

根据前边接口SaaS步骤 的分析，需要在商户平台 完成新增员工、新增门店、设置门店管理员等功能。

### 1.4.1 商户服务新增门店接口

#### 1.4.1.1 接口定义

在商户服务定义新增门店接口

1、接口描述如下：

1) 商户注册的同时新增默认门店

2、接口定义如下：

生成StoreDTO类。

在MerchantService接口类中定义如下接口：

```
/**
 * 商户下新增门店
 * @param storeDTO
 */
StoreDTO createStore(StoreDTO storeDTO) throws BusinessException;
```

#### 1.4.1.2 接口实现

编写StoreConvert。

在MerchantServiceImpl类中实现createStore方法。

```
@Override
public StoreDTO createStore(StoreDTO storeDTO) {
    Store store = StoreConvert.INSTANCE.dto2entity(storeDTO);
    log.info("商户下新增门店"+JSON.toJSONString(store));
    storeMapper.insert(store);
    return StoreConvert.INSTANCE.entity2dto(store);
}
```

## 1.4.2 商户服务新增员工接口

### 1.4.2.1 接口定义

在商户服务定义新增员工接口

1、接口描述如下：

- 1) 接收商户填写的员工数据
- 2) 请求商户服务进行新增员工

员工的账号和手机号需要保持唯一性。

3) 返回结果

2、接口定义如下：

生成StaffDTO类。

在MerchantService接口类中定义如下接口：

```
/**
 * 商户新增员工
 * @param staffDTO
 */
StaffDTO createStaff(StaffDTO staffDTO) throws BusinessException;
```

### 1.4.2.2 接口实现

编写StaffConvert。

在MerchantServiceImpl类中定义createStaff方法：

```
@Override
public StaffDTO createStaff(StaffDTO staffDTO) {
    //1. 校验手机号格式及是否存在
    String mobile = staffDTO.getMobile();

    if(StringUtils.isBlank(mobile)){
```



```
        throw new BusinessException(CommonErrorCode.E_100112);
    }
    //根据商户id和手机号校验唯一性
    if(isExistStaffByMobile(mobile, staffDTO.getMerchantId())){
        throw new BusinessException(CommonErrorCode.E_100113);
    }
    //2.校验用户名是否为空
    String username = staffDTO.getUsername();
    if(StringUtils.isBlank(username)){
        throw new BusinessException(CommonErrorCode.E_100110);
    }
    //根据商户id和账号校验唯一性
    if(isExistStaffByUserName(username, staffDTO.getMerchantId())){
        throw new BusinessException(CommonErrorCode.E_100114);
    }
    Staff entity = StaffConvert.INSTANCE.dto2entity(staffDTO);
    log.info("商户下新增员工");
    staffMapper.insert(entity);

    return StaffConvert.INSTANCE.entity2dto(entity);
}

/**
 * 根据手机号判断员工是否已在指定商户存在
 * @param mobile 手机号
 * @return
 */
private boolean isExistStaffByMobile(String mobile, Long merchantId) {
    LambdaQueryWrapper<Staff> lambdaQueryWrapper = new LambdaQueryWrapper<Staff>();
    lambdaQueryWrapper.eq(Staff::getMobile, mobile).eq(Staff::getMerchantId, merchantId);
    int i = staffMapper.selectCount(lambdaQueryWrapper);
    return i > 0;
}

/**
 * 根据账号判断员工是否已在指定商户存在
 * @param userName
 * @param merchantId
 * @return
 */
private boolean isExistStaffByUserName(String userName, Long merchantId) {
    LambdaQueryWrapper<Staff> lambdaQueryWrapper = new LambdaQueryWrapper<Staff>();
    lambdaQueryWrapper.eq(Staff::getUsername, userName).eq(Staff::getMerchantId,
merchantId);
    int i = staffMapper.selectCount(lambdaQueryWrapper);
    return i > 0;
}
```

### 1.4.3 商户服务门店设置管理员接口

### 1.4.3.1 接口定义

1、接口描述如下：绑定门店和员工对应关系

2、接口定义如下：

在MerchantService接口类中定义如下接口：

```
/**
 * 为门店设置管理员
 * @param storeId
 * @param staffId
 * @throws BusinessException
 */
void bindStaffToStore(Long storeId, Long staffId) throws BusinessException;
```

### 1.4.3.2 接口实现

在MerchantServiceImpl实现bindStaffToStore。

```
@Override
public void bindStaffToStore(Long storeId, Long staffId) {
    StoreStaff storeStaff = new StoreStaff();
    storeStaff.setStoreId(storeId);
    storeStaff.setStaffId(staffId);
    storeStaffMapper.insert(storeStaff);
}
```

## 1.4.4 商户服务商户注册接口修改

### 1.4.4.1 接口说明

SaaS系统的用户服务提供注册租户的接口，如下：

```
/**
 * 创建租户如果已存在租户则返回租户信息，否则新增租户、新增租户管理员，同时初始化权限
 * 1.若管理员用户名已存在，禁止创建
 * 2.手机号已存在，禁止创建
 * 3.创建根租户对应账号时，需要手机号，账号的用户名密码
 * @param createTenantRequest 创建租户请求信息
 * @return
 */
TenantDTO createTenantAndAccount(CreateTenantRequestDTO createTenantRequest);
```

接口参数：

- 1、手机号
- 2、账号
- 3、密码

- 4、租户类型：shanju-merchant
- 5、默认套餐：shanju-merchant
- 6、租户名称，同账号名

### 1.4.4.2 接口实现

- 1、添加SaaS的用户服务API依赖：打开shanjupay-merchant-service工程的pom.xml

```
<!-- SaaS的用户服务API依赖 -->
<dependency>
    <groupId>com.shanjupay</groupId>
    <artifactId>shanjupay-user-api</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

- 2、MerchantServiceImpl，完善createMerchant()方法：

```
@Override
@Transactional
public MerchantDTO createMerchant(MerchantDTO merchantDTO) throws BusinessException{
    // 1.校验
    if (merchantDTO == null) {
        throw new BusinessException(CommonErrorCode.E_100108);
    }
    //手机号非空校验
    if (StringUtils.isBlank(merchantDTO.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100112);
    }
    //校验手机号的合法性
    if (!PhoneUtil.isMatches(merchantDTO.getMobile())) {
        throw new BusinessException(CommonErrorCode.E_100109);
    }
    //联系人非空校验
    if (StringUtils.isBlank(merchantDTO.getUsername())) {
        throw new BusinessException(CommonErrorCode.E_100110);
    }
    //密码非空校验
    if (StringUtils.isBlank(merchantDTO.getPassword())) {
        throw new BusinessException(CommonErrorCode.E_100111);
    }

    //校验商户手机号的唯一性,根据商户的手机号查询商户表,如果存在记录则说明已有相同的手机号重复
    LambdaQueryWrapper<Merchant> lambdaQryWrapper = new LambdaQueryWrapper<Merchant>()
        .eq(Merchant::getMobile, merchantDTO.getMobile());
    Integer count = merchantMapper.selectCount(lambdaQryWrapper);
    if(count>0){
        throw new BusinessException(CommonErrorCode.E_100113);
    }
    //2.添加租户 和账号 并绑定关系
    CreateTenantRequestDTO createTenantRequest = new CreateTenantRequestDTO();
```



```
createTenantRequest.setMobile(merchantDTO.getMobile());
//表示该租户类型是商户
createTenantRequest.setTenantTypeCode("shanju-merchant");
//设置租户套餐为初始化套餐
createTenantRequest.setBundleCode("shanju-merchant");
//租户的账号信息
createTenantRequest.setUsername(merchantDTO.getUsername());
createTenantRequest.setPassword(merchantDTO.getPassword());
//新增租户并设置为管理员
createTenantRequest.setName(merchantDTO.getUsername());
log.info("商户中心调用统一账号服务，新增租户和账号");
TenantDTO tenantDTO = tenantService.createTenantAndAccount(createTenantRequest);
if (tenantDTO == null || tenantDTO.getId() == null) {
    throw new BusinessException(CommonErrorCode.E_200012);
}

//判断租户下是否已经注册过商户
Merchant merchant = merchantMapper
    .selectOne(new QueryWrapper<Merchant>().lambda().eq(Merchant::getTenantId,
tenantDTO.getId()));
if (merchant != null && merchant.getId() != null) {
    throw new BusinessException(CommonErrorCode.E_200017);
}

//3. 设置商户所属租户
merchantDTO.setTenantId(tenantDTO.getId());
//设置审核状态，注册时默认为"0"
merchantDTO.setAuditStatus("0");//审核状态 0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝
Merchant entity = MerchantCovert.INSTANCE.dto2entity(merchantDTO);
//保存商户信息
log.info("保存商户注册信息");
merchantMapper.insert(entity);

//4.新增门店，创建根门店
StoreDTO storeDTO = new StoreDTO();
storeDTO.setMerchantId(entity.getId());
storeDTO.setStoreName("根门店");
storeDTO = createStore(storeDTO);
log.info("门店信息：{" + JSON.toJSONString(storeDTO));

//5.新增员工，并设置归属门店
StaffDTO staffDTO = new StaffDTO();
staffDTO.setMerchantId(entity.getId());
staffDTO.setMobile(merchantDTO.getMobile());
staffDTO.setUsername(merchantDTO.getUsername());
//为员工选择归属门店,此处为根门店
staffDTO.setStoreId(storeDTO.getId());
staffDTO = createStaff(staffDTO);

//6.为门店设置管理员
bindStaffToStore(storeDTO.getId(), staffDTO.getId());

//返回商户注册信息
```

```
        return MerchantCovert.INSTANCE.entity2dto(entity);  
    }
```

#### 1.4.4.2 测试

使用Postman进行商户注册，请求：<http://localhost:57010/merchant/merchants/register>

注册成功后观察shanjupay\_user数据库下的account、tenant、tenant\_account表是否有新注册的账号、租户信息。

## 2 用户认证

SaaS平台提供统一认证的服务，本章节学习SaaS平台的认证功能。

### 2.1 基本概念

#### 2.1.1 什么是认证

进入移动互联网时代，大家每天都在刷手机，常用的软件有微信、支付宝、头条等，下边拿微信来举例子说明认证相关的基本概念，在初次使用微信前需要注册成为微信用户，然后输入账号和密码即可登录微信，输入账号和密码登录微信的过程就是认证。

系统为什么要认证？

认证是为了保护系统的隐私数据与资源，用户的身份合法方可访问该系统的资源。

**认证**：用户认证就是判断一个用户的身份是否合法的过程，用户去访问系统资源时系统要求验证用户的身份信息，身份合法方可继续访问，不合法则拒绝访问。常见的用户身份认证方式有：用户名密码登录，二维码登录，手机短信登录，指纹认证等方式。

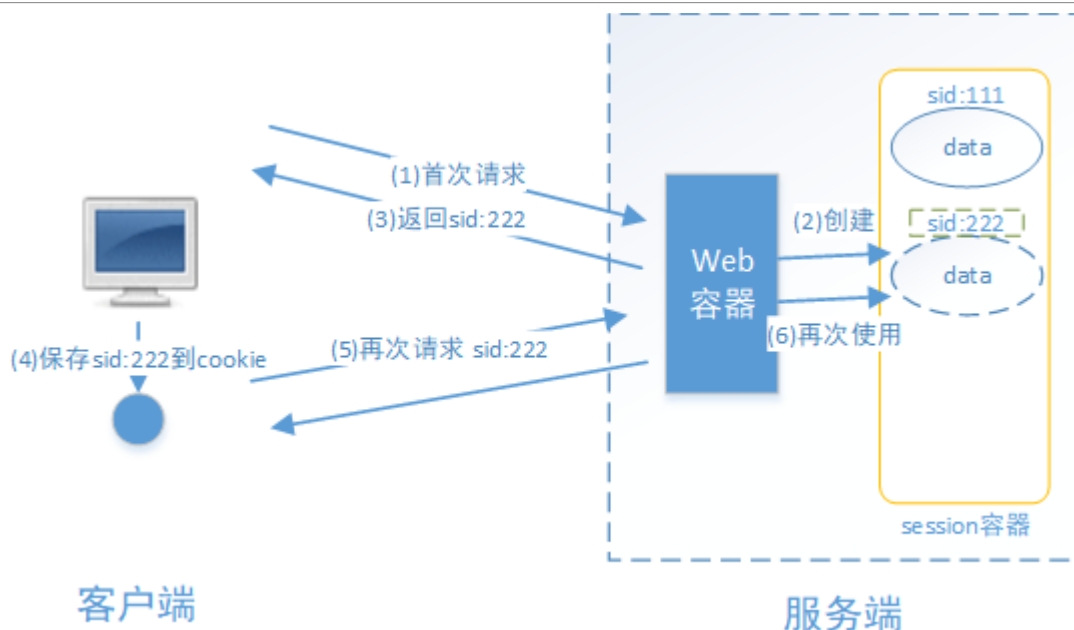
#### 2.1.2 什么是会话

用户认证通过后，为了避免用户的每次操作都进行认证可将用户的信息保证在会话中。会话就是系统为了保持当前用户的登录状态所提供的机制，常见的有基于session方式、基于token方式等。

基于session的认证方式如下图：

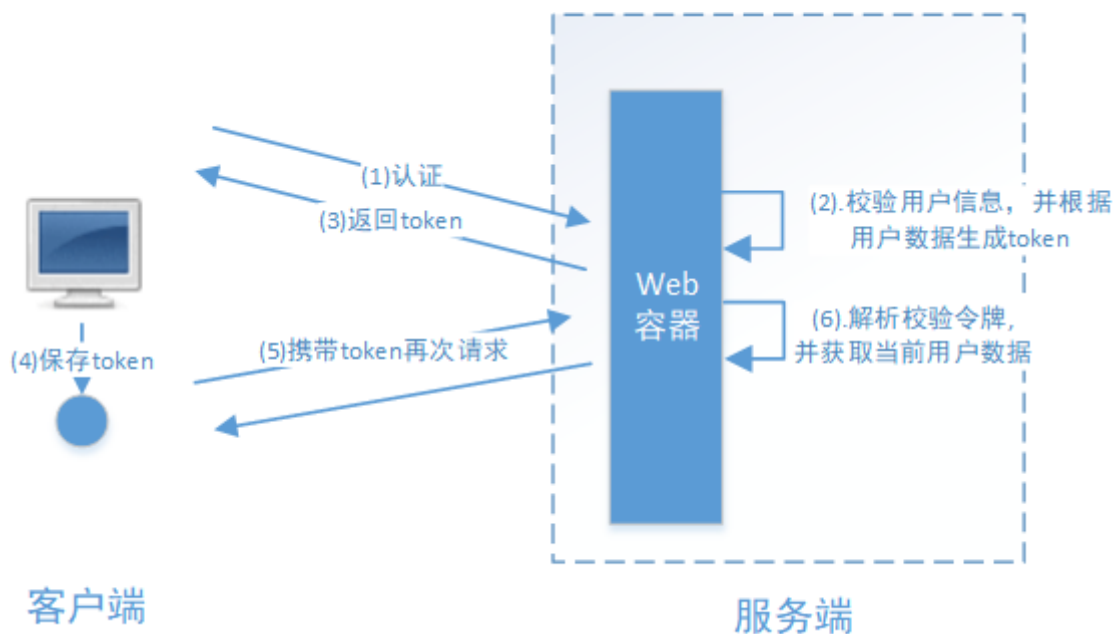
它的交互流程是，用户认证成功后，在服务端生成用户相关的数据保存在session(当前会话)中，发给客户端的session\_id 存放到 cookie 中，这样用户客户端请求时带上 session\_id 就可以验证服务器端是否存在 session 数据，以此完成用户的合法校验，当用户退出系统或session过期销毁时,客户端的session\_id也就无效了。





基于token方式如下图：

它的交互流程是，用户认证成功后，服务端生成一个token发给客户端，客户端存储token，每次请求时带上token，服务端收到token通过验证后即可确认用户身份。



基于session的认证方式由Servlet规范定制，服务端要存储session信息需要占用内存资源，客户端需要支持cookie；基于token的方式则一般不需要服务端存储token，并且不限制客户端的存储方式。如今移动互联网时代更多类型的客户端需要接入系统，系统多是采用前后端分离的架构进行实现，所以基于token的方式更适合。

### 2.1.3 什么是授权

还拿微信来举例子，微信登录成功后用户即可使用微信的功能，比如，发红包、发朋友圈、添加好友等，没有绑定银行卡的用户是无法发送红包的，绑定银行卡的用户才可以发红包，发红包功能、发朋友圈功能都是微信的资源即功能资源，用户拥有发红包功能的权限才可以正常使用发送红包功能，拥有发朋友圈功能的权限才可以使用发朋友圈功能，这个根据用户的权限来控制用户使用资源的过程就是授权。

为什么要授权？

认证是为了保证用户身份的合法性，授权则是为了更细粒度的对隐私数据进行划分，授权是在认证通过后发生的，控制不同的用户能够访问不同的资源。

**授权：**授权是用户认证通过根据用户的权限来控制用户访问资源的过程，拥有资源的访问权限则正常访问，没有权限则拒绝访问。

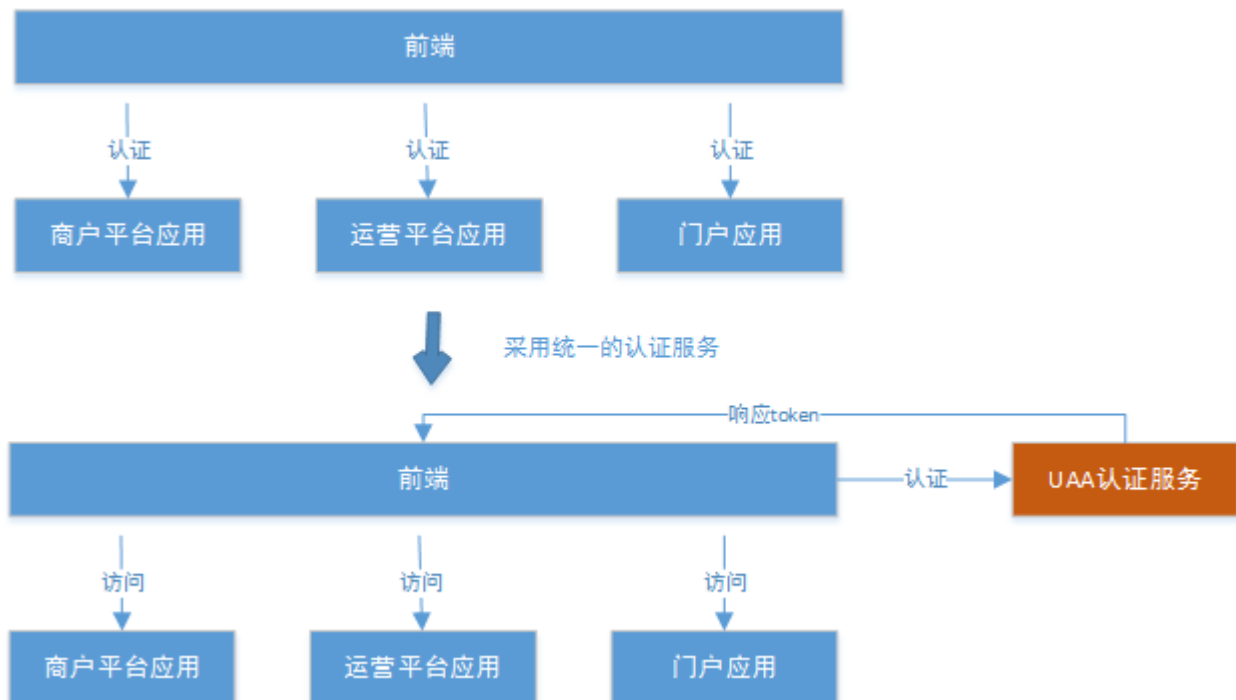
## 2.2.分布式系统认证方案

### 2.2.1 分布式认证需求

#### 1、统一认证授权

分布式系统的每个服务（系统）都会有认证、授权的需求，如果每个服务都实现一套认证授权逻辑会非常冗余，考虑分布式系统共享性的特点，需要由独立的认证服务来处理系统认证授权的请求；

如下图，闪聚支付平台包括：商户平台应用、运营平台应用、门户应用，每个应用都需要身份认证，闪聚支付平台统一由UAA认证服务完成认证。

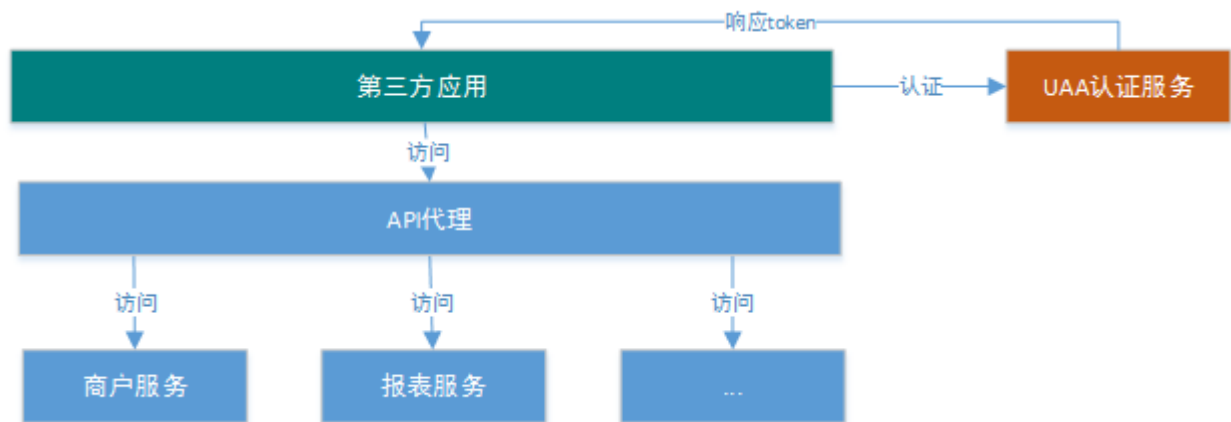


- 1、前端请求UAA认证服务请求认证，认证通过获取 Token
- 2、前端携带Token访问各各应用。

#### 2、开放认证体系

考虑分布式系统开放性的特点，UAA认证服务不仅服务于平台自身，并且对第三方系统也要提供认证，平台应提供扩展和开放的认证机制，以开放API的方式供第三方应用接入，一方应用（内部系统服务）和三方应用（第三方应用）均采用统一机制接入。

下图是第三方的应用接入闪聚支付平台结构图：



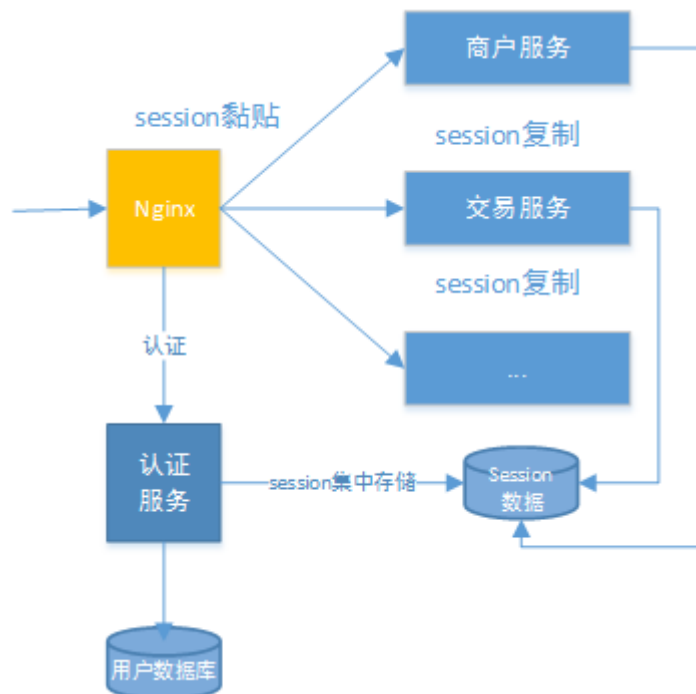
- 1、第三方应用请求UAA认证服务请求认证，认证通过获取Token。
- 2、第三方应用携带Token访问API代理（专门针对第三方应用接入开发的微服务）。
- 3、API代理访问平台微服务向第三方应用返回业务数据。

## 2.2.2 分布式认证方案

### 2.2.2.1 选型分析

#### 1、基于session的认证方式

在分布式的环境下，基于session的认证会出现一个问题，每个应用服务都需要在session中存储用户身份信息，通过负载均衡将本地的请求分配到另一个应用服务需要将session信息带过去，否则会重新认证。



这个时候，通常的做法有下面几种：

**Session复制**：多台应用服务器之间同步session，使session保持一致，对外透明。

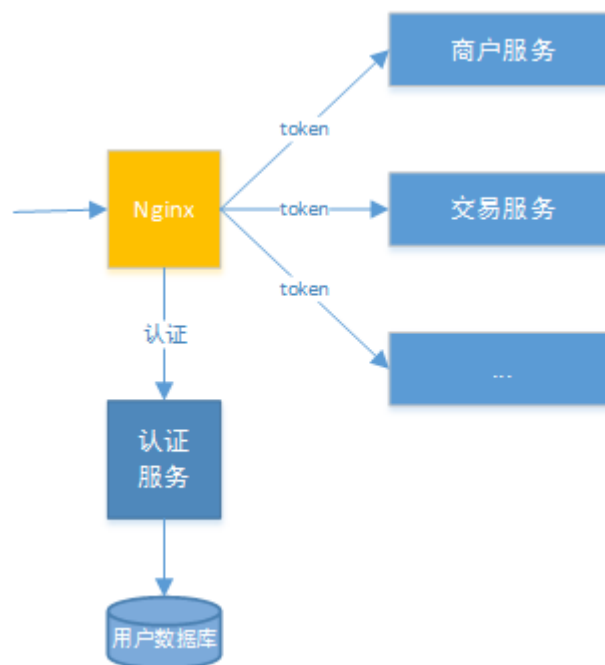
**Session黏贴**：当用户访问集群中某台服务器后，强制指定后续所有请求均落到此机器上。

**Session集中存储：**将Session存入分布式缓存中，所有服务器应用实例统一从分布式缓存中存取Session。

总体来讲，基于session认证的认证方式，可以更好的在服务端对会话进行控制，且安全性较高。但是，session机制方式基于cookie，在复杂多样的移动客户端上不能有效的使用，并且无法跨域，另外随着系统的扩展需提高session的复制、黏贴及存储的容错性。

## 2、基于token的认证方式

基于token的认证方式，服务端不用存储认证数据，易维护扩展性强，客户端可以把token 存在任意地方，并且可以实现web和app统一认证机制。其缺点也很明显，token由于自包含信息，因此一般数据量较大，而且每次请求都需要传递，因此比较占带宽。另外，token的签名验签操作也会给cpu带来额外的处理负担。

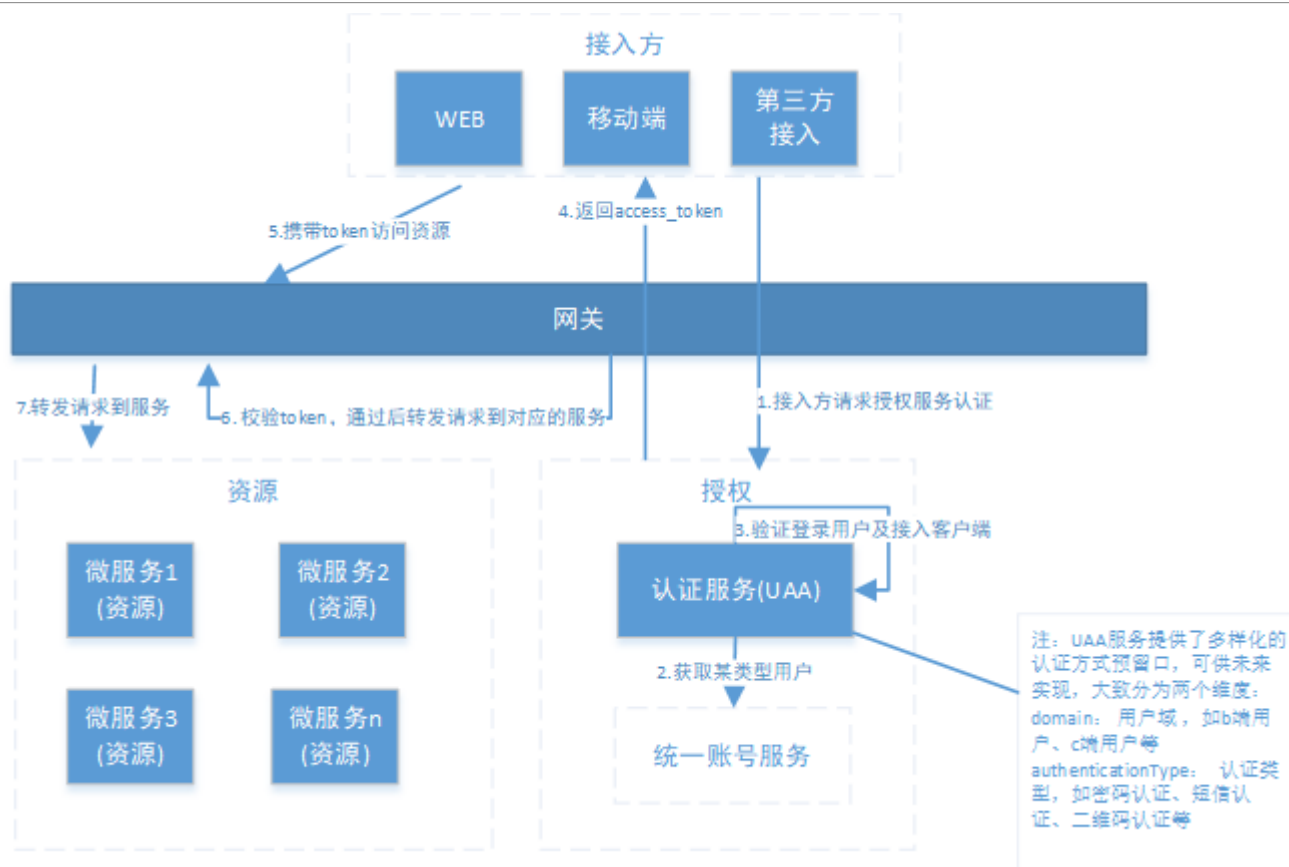


### 2.2.2.2 技术方案

根据 选型的分析，决定采用基于token的认证方式，它的优点是：

- 1、适合统一认证的机制，客户端、一方应用、三方应用都遵循一致的认证机制。
- 2、token认证方式对第三方应用接入更适合，因为它更开放，使用当前有流行的开放协议Oauth2.0、JWT。
- 3、一般情况服务端无需存储会话信息，减轻了服务端的压力。

分布式系统认证技术方案见下图：



流程描述：

- (1) 接入方（需要使用平台资源的统称为接入方）采取OAuth2.0方式请求统一认证服务(UAA)进行认证。
- (2) 认证服务(UAA)调用统一账号服务去查询该用户信息及其权限信息。（第三方应用接入不需要该步骤）
- (3) 认证服务(UAA)验证登录用户及第三方应用合法性。
- (4) 若接入方身份合法，认证服务生成jwt令牌返回给接入方，其中jwt中包含了权限信息。
- (5) 接入方携带jwt令牌对API网关内的微服务资源进行访问。
- (6) API网关对令牌解析、并验证接入方的权限是否能够访问本次请求的微服务。
- (7) 如果接入方的权限没问题，API网关将Token转发至微服务。
- (8) 微服务收到请求，明文token中包含登录用户的身份和权限信息，后续微服务使用用户身份及权限信息。

流程所涉及到的统一账号服务、UAA服务、API网关这三个组件职责如下：

### 1) 统一账号服务

提供商户和平台运营人员的登录账号、密码、角色、权限、资源等系统级信息的管理，不包含用户业务信息。

### 2) 统一认证服务(UAA)

它承载了OAuth2.0接入方认证、登入用户的认证、授权以及生成令牌的职责，完成实际的用户认证、授权功能。

### 3) API网关

作为系统的唯一入口，API网关为接入方提供定制的API集合，它可能还具有其它职责，如身份验证、监控、负载均衡、缓存等。API网关方式的核心要点是，所有的接入方和消费端都通过统一的网关接入微服务，在网关层处理所有的非业务功能。

## 2.3 OAuth2.0

### 2.3.1 OAuth2.0介绍

OAuth（开放授权）是一个开放标准，允许用户授权第三方应用访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方应用或分享他们数据的所有内容。OAuth2.0是OAuth协议的延续版本，但不向后兼容OAuth 1.0即完全废止了OAuth1.0。很多大公司如Google，Yahoo，Microsoft等都提供了OAUTH认证服务，这些都足以说明OAUTH标准逐渐成为开放资源授权的标准。

OAuth协议目前发展到2.0版本，1.0版本过于复杂，2.0版本已得到广泛应用。

参考：<https://baike.baidu.com/item/oauth/7153134?fr=aladdin>

OAuth协议：<https://tools.ietf.org/html/rfc6749>

下边分析一个OAuth2认证的例子，通过例子去理解OAuth2.0协议的认证流程，本例子是黑马程序员网站使用微信认证的过程，这个过程的简要描述如下：

用户借助微信认证登录黑马程序员网站，用户就不用单独在黑马程序员注册用户，怎么样算认证成功吗？黑马程序员网站需要成功从微信获取用户的身份信息则认为用户认证成功，那如何从微信获取用户的身份信息？用户信息的拥有者是用户本人，微信需要经过用户的同意方可为黑马程序员网站生成令牌，黑马程序员网站拿此令牌方可从微信获取用户的信息。

#### 1、客户端请求第三方授权

用户进入黑马程序的登录页面，点击微信的图标以微信账号登录系统，用户是自己在微信里信息的资源拥有者。



点击“微信”出现一个二维码，此时用户扫描二维码，开始给黑马程序员授权。



## 2、资源拥有者同意给客户端授权

资源拥有者扫描二维码表示资源拥有者同意给客户端授权，微信会对资源拥有者的身份进行验证，验证通过后，微信会询问用户是否给授权黑马程序员访问自己的微信数据，用户点击“确认登录”表示同意授权，微信认证服务器会颁发一个授权码，并重定向到黑马程序员的网站。





## 黑马程序员用户中心

即将登录黑马程序员用户中心，请确  
认是本人操作

- 使用你的帐号登录该应用

确认登录

取消

### 3、客户端获取到授权码，请求认证服务器申请令牌

此过程用户看不到，客户端应用程序请求认证服务器，请求携带授权码。

### 4、认证服务器向客户端响应令牌

微信认证服务器验证了客户端请求的授权码，如果合法则给客户端颁发令牌，令牌是客户端访问资源的通行证。

此交互过程用户看不到，当客户端拿到令牌后，用户在黑马程序员看到已经登录成功。

### 5、客户端请求资源服务器的资源

客户端携带令牌访问资源服务器的资源。

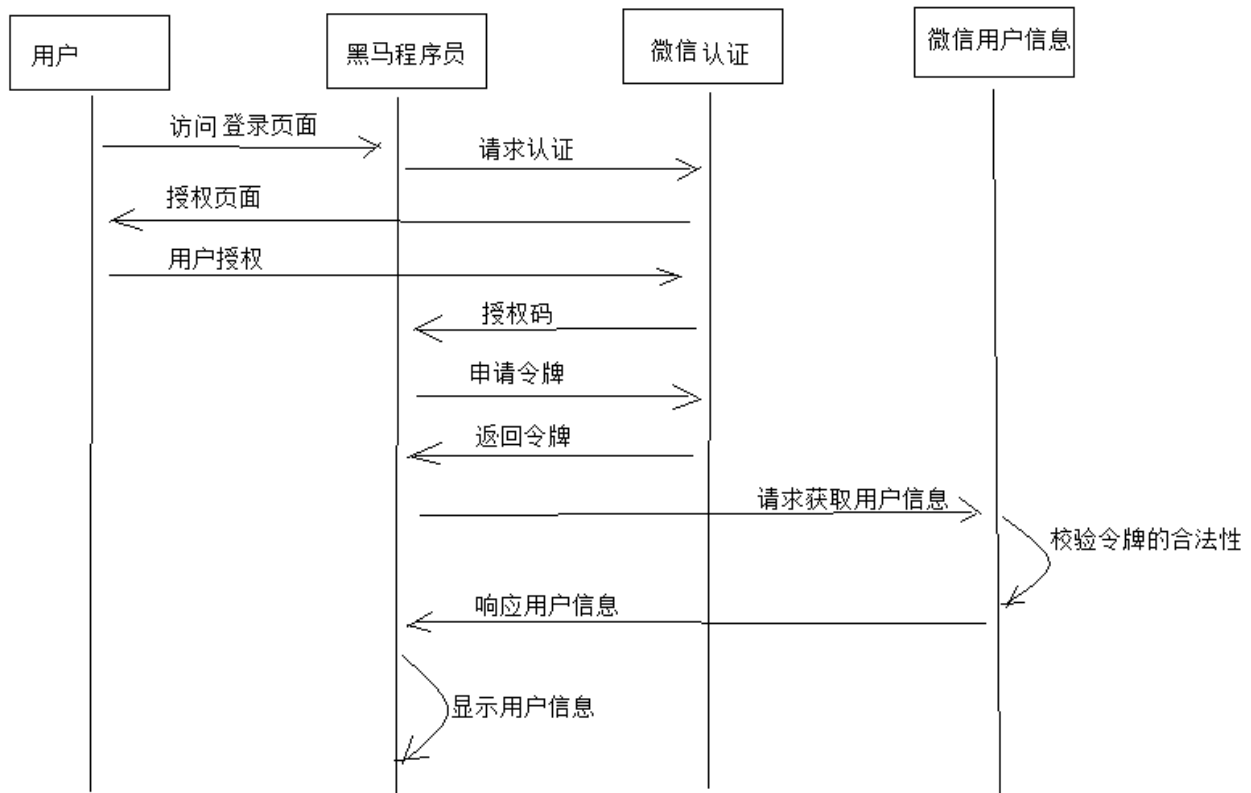
黑马程序员网站携带令牌请求访问微信服务器获取用户的基本信息。

### 6、资源服务器返回受保护资源

资源服务器校验令牌的合法性，如果合法则向用户响应资源信息内容。

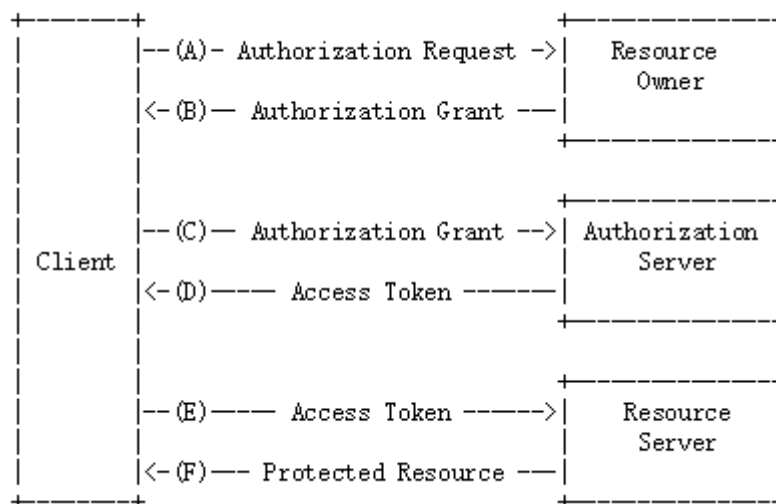
以上认证授权详细的执行流程如下：





通过上边的例子我们大概了解了OAuth2.0的认证过程，下边我们看OAuth2.0认证流程：

引自OAuth2.0协议rfc6749 <https://tools.ietf.org/html/rfc6749>



OAuth2.0包括以下角色：

### 1、客户端

本身不存储资源，需要通过资源拥有者的授权去请求资源服务器的资源，比如：Android客户端、Web客户端（浏览器端）、微信客户端等。

### 2、资源拥有者

通常为用户，也可以是应用程序，即该资源的拥有者。

### 3、授权服务器（也称认证服务器）

用于服务提供商对资源拥有的身份进行认证、对访问资源进行授权，认证成功后会给客户端发放令牌（access\_token），作为客户端访问资源服务器的凭据。本例为微信的认证服务器。

#### 4、资源服务器

存储资源的服务器，本例子为微信存储的用户信息。

现在还有一个问题，服务提供商能允许随便一个**客户端**就接入到它的**授权服务器**吗？答案是否定的，服务提供商会给准入的接入方一个身份，用于接入时的凭据：

**client\_id**：客户端标识 **client\_secret**：客户端密钥

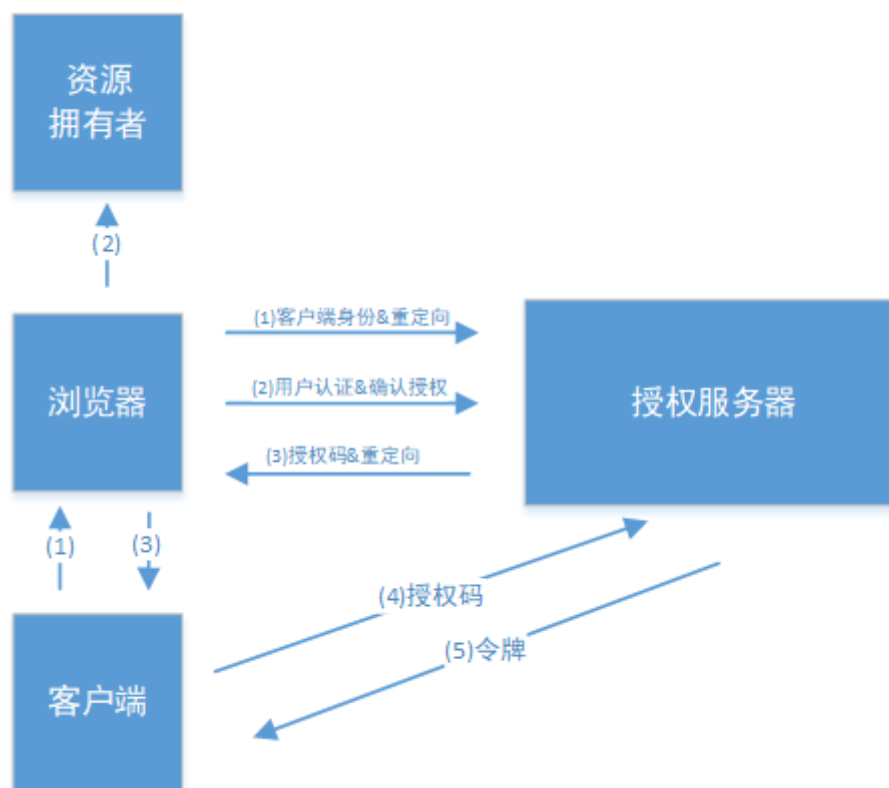
因此，准确来说，**授权服务器**对两种OAuth2.0中的两个角色进行认证授权，分别是**资源所有者**、**客户端**。

### 2.3.2 四种模式

OAuth2.0提供了四种授权(获取令牌)方式，四种方式均采用不同的执行流程，让我们适应不同的场景。

#### 2.3.2.1.授权码模式

授权码模式流程如下：



(1) 资源所有者打开客户端，客户端要求资源所有者给予授权，它将浏览器被重定向到授权服务器，重定向时会附加客户端的身份信息。如：

```
/uaa/oauth/authorize?
client_id=p2pweb&response_type=code&scope=app&redirect_uri=http://xx.xx/notify
```

参数列表如下：

- client\_id：客户端接入标识。
- response\_type：授权码模式固定为code。
- scope：客户端权限。
- redirect\_uri：跳转uri，当授权码申请成功后会跳转到此地址，并在后边带上code参数（授权码）。

**（2）浏览器出现向授权服务器授权页面，之后将用户同意授权。**

**（3）授权服务器将授权码（AuthorizationCode）转经浏览器发送给client(通过redirect\_uri)。**

**（4）客户端拿着授权码向授权服务器索要访问access\_token，请求如下：**

```
/uaa/oauth/token?  
client_id=p2pweb&client_secret=gdjbcd&grant_type=authorization_code&code=5PgfcD&redirect_uri=htt  
p://xx.xx/notify
```

参数列表如下

- client\_id：客户端准入标识。
- client\_secret：客户端密钥。
- grant\_type：授权类型，填写authorization\_code，表示授权码模式
- code：授权码，就是刚刚获取的授权码，注意：授权码只使用一次就无效了，需要重新申请。
- redirect\_uri：申请授权码时的跳转url，一定和申请授权码时用的redirect\_uri一致。

**（4）授权服务器返回令牌(access\_token)**

这种模式是四种模式中最安全的一种模式。一般用于Web服务器端应用或第三方的原生App调用资源服务的时候。因为在这种模式中access\_token不会经过浏览器或移动端的App，而是直接从服务端去交换，这样就最大限度的减小了令牌泄漏的风险。

### 2.3.2.2.密码模式

密码模式使用较多，适应于第一方的单页面应用以及第一方的原生App，比如：闪聚支付平台运营平台用户使用此模式完成用户登录。

密码模式认证流程如下：



(1) 资源拥有者将用户名、密码发送给客户端

(2) 客户端拿着资源拥有者的用户名、密码向授权服务器请求令牌 (access\_token)，请求如下：

```
/uaa/oauth/token?  
client_id=p2pweb&client_secret=fgsdgrf&grant_type=password&username=shangsan&password=123456
```

参数列表如下：

- client\_id：客户端准入标识。
- client\_secret：客户端密钥。
- grant\_type：授权类型，填写password表示密码模式
- username：资源拥有者用户名。
- password：资源拥有者密码。

(3) 授权服务器将令牌 (access\_token) 发送给client

这种模式十分简单，但是却意味着直接将用户敏感信息泄漏给了client，因此这就说明这种模式只能用于client是我们自己开发的情况下。因此密码模式一般用于我们自己开发的，第一方原生App或第一方单页面应用。

### 2.3.2.3.客户端模式



(1) 客户端向授权服务器发送自己的身份信息，并请求令牌 (access\_token)

(2) 确认客户端身份无误后，将令牌 (access\_token) 发送给client，请求如下：

```
/uaa/oauth/token?client_id=p2pweb&client_secret=fdafdag&grant_type=client_credentials
```

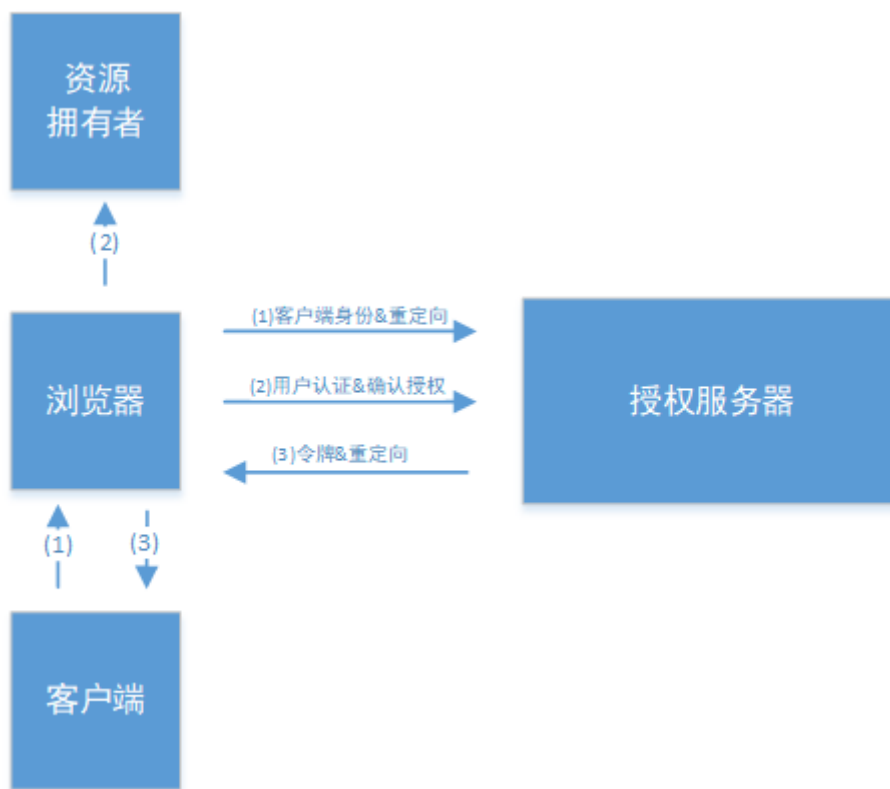
参数列表如下：

- client\_id：客户端准入标识。
- client\_secret：客户端密钥。
- grant\_type：授权类型，填写client\_credentials表示客户端模式

这种模式是最方便但最不安全的模式。因此这就要求我们对client完全的信任，而client本身也是安全的。因此这种模式一般用来提供给我们完全信任的服务器端服务。比如，合作方系统对接，拉取一组用户信息。

客户端模式适应于没有用户参与的，完全信任的一方或合作方服务器端程序接入。

#### 2.3.2.4.简化模式



(1) 资源拥有者打开客户端，客户端要求资源拥有者给予授权，它将浏览器被重定向到授权服务器，重定向时会附加客户端的身份信息。如：

```
/uaa/oauth/authorize?
client_id=p2pweb&response_type=token&scope=app&redirect_uri=http://xx.xx/notify
```

参数描述同**授权码模式**，注意response\_type=token，说明是简化模式。

(2) 浏览器出现向授权服务器授权页面，之后将用户同意授权。

(3) 授权服务器将授权码将令牌 (access\_token) 以Hash的形式存放在重定向uri的fragment中发送给浏览器。

注：fragment 主要是用来标识 URI 所标识资源里的某个资源，在 URI 的末尾通过 (#) 作为 fragment 的开头，其中 # 不属于 fragment 的值。如<https://domain/index#L18>这个 URI 中 L18 就是 fragment 的值。大家只需要知道js通过响应浏览器地址栏变化的方式能获取到fragment 就行了。

一般来说，简化模式用于第三方单页面应用。

## 2.4.统一认证测试

### 2.4.1.认证接口说明

#### 2.4.1.1.登录

**功能说明：**用户登录并返回令牌，该令牌用于访问闪聚支付平台内受保护资源。

**访问路径：**[授权服务地址]/oauth/token

**请求参数：**

- **grant\_type**：授权类型，可以是authorization\_code,implicit,client\_credentials,password
- **client\_id**：接入客户端id
- **client\_secret**：接入客户端密钥
- **username**：登录用户名，认证类型(如密码认证，短信认证，二维码认证等)

```
{"username": "admin", "authenticationType": "password"}
```

- **password**：登录密码

**响应内容：**

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", //令牌
  "token_type": "bearer",
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...", //刷新令牌
  "expires_in": 31535999, //有效期
  "scope": "read",
  "jti": "dfc6f30a-aa8e-4028-a43a-1487e64a2cfb"
}
```

#### 2.4.1.2.解析令牌

**功能说明：**返回令牌的明文内容，描述的是当前登录的用户及接入客户端的信息。

**访问路径：**[授权服务地址]/oauth/check\_token

**请求参数：**

- **token**："eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

**响应内容：**

```
{
  "aud": [
    "shanju-resource"
  ],
  "payload": {
    "1186173819157544962": {
```



```
        "user_authorities": {
            "r_001": [
                "sj_m_staff_list",
                "sj_m_payment",
                "sj_o_member_list",
                "sj_m_console",
                "sj_m_account_list",
                "sj_o_entreprise_list",
                "sj_m_app_list",
                "sj_m_enterprise_auth",
                "sj_o_audit",
                "sj_m_transaction_list",
                "sj_m_store_list",
                "sj_o_service_type",
                "sj_m_account_check"
            ],
            "r_002": [
                "sj_o_member_query"
            ]
        }
    },
    "user_name": "admin",
    "scope": [
        "read"
    ],
    "mobile": "17717771777",
    "exp": 1606354316,
    "client_authorities": [
        "ROLE_MERCHANT",
        "ROLE_USER"
    ],
    "jti": "dfc6f30a-aa8e-4028-a43a-1487e64a2cfb",
    "client_id": "merchant-platform"
}
```

## 2.4.2.接口测试

分别启动shanjupay-gateway、shanjupay-user、shanjupay-uaa三个工程，测试过程若有不清楚可参考前面1.2.3章节OAuth2.0理论部分和上一小节接口说明。

### 2.4.2.1.密码模式认证

```
POST http://localhost:56020/uaa/oauth/token
```

请求参数：

POST

http://localhost:56020/uaa/oauth/token

Params

Send

Key	Value	Description
<input checked="" type="checkbox"/> client_id	merchant-platform	
<input checked="" type="checkbox"/> client_secret	123456	
<input checked="" type="checkbox"/> grant_type	password	
<input checked="" type="checkbox"/> username	{"username":"itcasttest03","authenticationType":"passwo...	
<input checked="" type="checkbox"/> password	123456	
New key	Value	Description

Body

Cookies (4)

Headers (8)

Test Results

Status: 200 OK

Time: 134 ms

Pretty

Raw

Preview

JSON

```

1  {
2    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiIsc2hhbWp1LXJlc291cmNlIi0sInBheWxvYWQiOiJ0nsiMiI6eyJyZXNvdXJjZXMmOm51bGwsInV
3    "token_type": "bearer",
4    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiIsc2hhbWp1LXJlc291cmNlIi0sInBheWxvYWQiOiJ0nsiMiI6eyJyZXNvdXJjZXMmOm51bGwsInV
5    "expires_in": 31535999,
6    "scope": "read",
7    "jti": "c02f77b7-b29f-4280-9c73-02019d28ded5"

```

## 2.4.2.2. 解析token

http://localhost:56020/uaa/oauth/check\_token

请求参数：

检查token有效性

Comments (0)

Examples (0)

POST

http://localhost:56010/uaa/oauth/check\_token

Send

Save

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

Code

none

form-data

☒ x-www-form-urlencoded

raw

binary

GraphQL BETA

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiIsc2hhbWp1LXJlc291cmNlIi0sInBheWxvYWQiOiJ0nsiMiI6eyJyZXNvdXJjZXMmOm51bGwsInV	

## 2.4.3 JWT

### 什么是JWT？

JSON Web Token (JWT) 是一个开放的行业标准 (RFC 7519)，它定义了一种简洁的、自包含的协议格式，用于在通信双方传递json对象，传递的信息经过数字签名可以被验证和信任。JWT可以使用HMAC算法或使用RSA的公钥/私钥对来签名，防止被篡改。

**JWT令牌的优点：** 1、jwt基于json，非常方便解析。 2、可以在令牌中自定义丰富的内容，易扩展。 3、通过非对称加密算法及数字签名技术，JWT防止篡改，安全性高。

### JWT令牌结构：

JWT令牌由Header、Payload、Signature三部分组成，每部分中间使用点 (.) 分隔，比如：xxxxx.yyyyy.zzzzz



# JWT TOKEN



## 1) Header

头部包括令牌的类型（即JWT）及使用的哈希算法（如HMAC SHA256或RSA）。

一个例子：

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

将上边的内容使用Base64Url编码，得到一个字符串就是JWT令牌的第一部分。

HS256就是HMAC-SHA256，加密算法使用HMAC，摘要算法使用SHA256。

测试：

将生成的jwt令牌第一部分使用Base64还原原始内容如下：

```
public static void main(String[] args) {
    byte[] header =
    java.util.Base64.getDecoder().decode("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9");
    System.out.println(new String(header));
}
```

得到的结果是：

```
{"alg":"HS256","typ":"JWT"}
```

## 2) Payload

第二部分是负载，内容也是一个json对象，它是存放有效信息的地方，它可以存放jwt提供的现成字段，比如：  
iss（签发者），exp（过期时间戳），sub（面向的用户）等，也可自定义字段。此部分不建议存放敏感信息，因为此部分可以解码还原原始内容。一个例子：

```
{
  "sub": "1234567890",
  "name": "456",
  "admin": true
}
```

最后将第二部分负载使用Base64Url编码，得到一个字符串就是JWT令牌的第二部分。

### 3 ) Signature

第三部分是签名，此部分用于防止jwt内容被篡改。这个部分使用base64url将前两部分进行编码，编码后使用点 (.) 连接组成字符串，最后使用header中声明 签名算法进行签名。一个例子：

```
HMACSHA256(  
base64UrlEncode(header) + "." +  
base64UrlEncode(payload),  
secret)
```

base64UrlEncode(header) : jwt令牌的第一部分。 base64UrlEncode(payload) : jwt令牌的第二部分。 secret : 签名所使用的密钥。

JWT三个部分只有第三部分是加密的，通过**数字签名**机制，我们既可以保证数据**完整性**，也可以对数据来源进行**身份验证**。

什么是签名？

签名是数字签名，发送方将消息原文使用摘要算法生成摘要，再用私钥对摘要进行加密，生成**数字签名**。

传输数据时为了保证数据的完整性可以使用数字签名技术：

- 1、发送方使用私钥对内容进行数字签名
- 2、将内容附带数字签名发送给对方
- 3、对方收到内容和数字签名，使用公钥进行验签（相当于解密的过程），如果发现内容不一致则说明传输过程被篡改。