**畅购电商系统开发**

**第10天**

传智播客.黑马程序员.深圳

# 学习目标

* 了资源服务器授权配置
* 掌握OAuth2.0认证微服务动态加载数据

授权码模式数据加载

密码模式数据加载

* 掌握购物车流程

用户未登录购物车

用户已登录购物车

* OAuth2.0认证并获取用户令牌数据

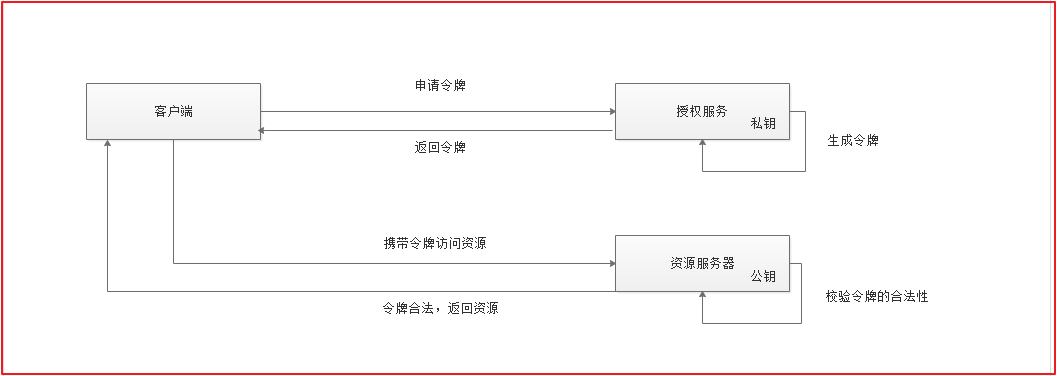
解析令牌，获取用户信息

* 微服务与微服务之间的认证

多个微服务之间令牌携带访问问题

# 资源服务器授权配置(回顾)

## 资源服务授权配置



基本上所有微服务都是资源服务，这里我们在课程管理服务上配置授权控制，当配置了授权控制后如要访问课程信 息则必须提供令牌。

### 配置公钥

认证服务生成令牌采用非对称加密算法，认证服务采用私钥加密生成令牌，对外向资源服务提供公钥，资源服务使 用公钥 来校验令牌的合法性。 将公钥拷贝到 public.key文件中，将此文件拷贝到资源服务工程的classpath下。

### 添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-actuator</**artifactId**>  
</**dependency**>

### 配置Http请求路径安全控制与公钥读取

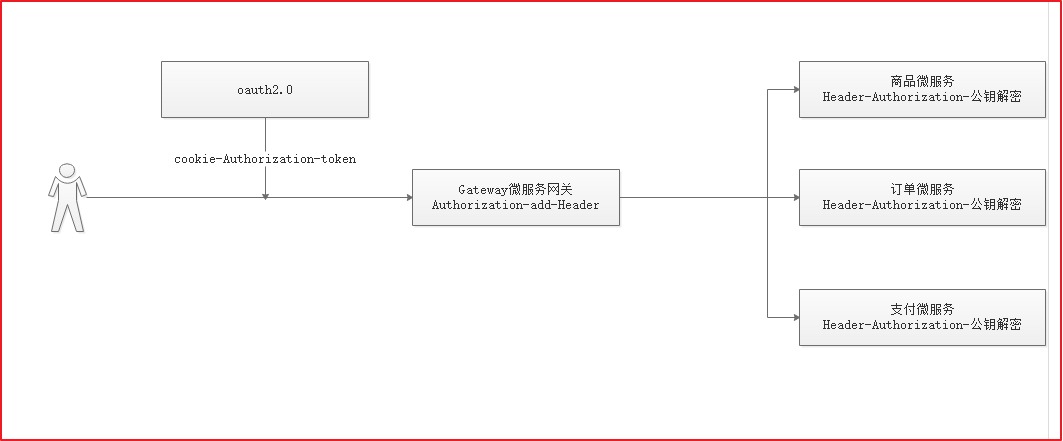
**配置每个系统的Http请求路径安全控制策略以及读取公钥信息识别令牌（类似于网关）**，如下：

*/\*\*  
 \* 实现信息校验  
 \* 1、获取公钥  
 \* 2、使用公钥作为校验密钥  
 \* 3、校验令牌  
 \*/*@Configuration  
@EnableResourceServer *//开启资源授权控制*@EnableGlobalMethodSecurity(prePostEnabled = **true**, securedEnabled = **true**)*//激活方法上的PreAuthorize注解***public class** ResourceServerConfig **extends** ResourceServerConfigurerAdapter {  
  
 *//公钥* **private static final** String ***PUBLIC\_KEY*** = **"public.key"**;  
  
 */\*\*\*  
 \* 定义JwtTokenStore  
 \** ***@param jwtAccessTokenConverter*** *\** ***@return*** *\*/* @Bean  
 **public** TokenStore tokenStore(JwtAccessTokenConverter jwtAccessTokenConverter) {  
 **return new** JwtTokenStore(jwtAccessTokenConverter);  
 }  
  
 */\*\*\*  
 \* 定义JJwtAccessTokenConverter  
 \** ***@return*** *\*/* @Bean  
 **public** JwtAccessTokenConverter jwtAccessTokenConverter() {  
 JwtAccessTokenConverter converter = **new** JwtAccessTokenConverter();  
 converter.setVerifierKey(getPubKey());  
 **return** converter;  
 }  
 */\*\*  
 \* 获取非对称加密公钥 Key  
 \** ***@return*** *公钥 Key  
 \*/* **private** String getPubKey() {  
 Resource resource = **new** ClassPathResource(***PUBLIC\_KEY***);  
 **try** {  
 InputStreamReader inputStreamReader = **new** InputStreamReader(resource.getInputStream());  
 BufferedReader br = **new** BufferedReader(inputStreamReader);  
 **return** br.lines().collect(Collectors.*joining*(**"\n"**));  
 } **catch** (IOException ioe) {  
 **return null**;  
 }  
 }  
  
 */\*\*\*  
 \* Http安全配置，对每个到达系统的http请求链接进行校验  
 \** ***@param http*** *\** ***@throws*** *Exception  
 \*/* @Override  
 **public void** configure(HttpSecurity http) **throws** Exception {  
 *//所有请求必须认证通过* http.authorizeRequests()  
 *//下边的路径放行* .antMatchers(  
 **"/user/add"**). *//配置地址放行* permitAll()  
 .anyRequest().  
 authenticated(); *//其他地址需要认证授权* }  
}

## 用户微服务资源授权

配置过程，我们在上一天课程4.3章节内容已经完成，请参考上一天4.3章节内容，步骤其实就是上面介绍的3步法。

# OAuth对接微服务



用户每次访问微服务的时候，先去oauth2.0服务登录，登录后再访问微服务网关，微服务网关将请求转发给其他微服务处理。

1.用户登录成功后，会将令牌信息存入到cookie中(一般建议存入到头文件中)

2.用户携带Cookie中的令牌访问微服务网关

3.微服务网关先获取头文件中的令牌信息，如果Header中没有Authorization令牌信息，则取参数中找，参数中如果没有，则取Cookie中找Authorization，最后将令牌信息封装到Header中，并调用其他微服务

4.其他微服务会获取头文件中的Authorization令牌信息，然后匹配令牌数据是否能使用公钥解密，如果解密成功说明用户已登录，解密失败，说明用户未登录

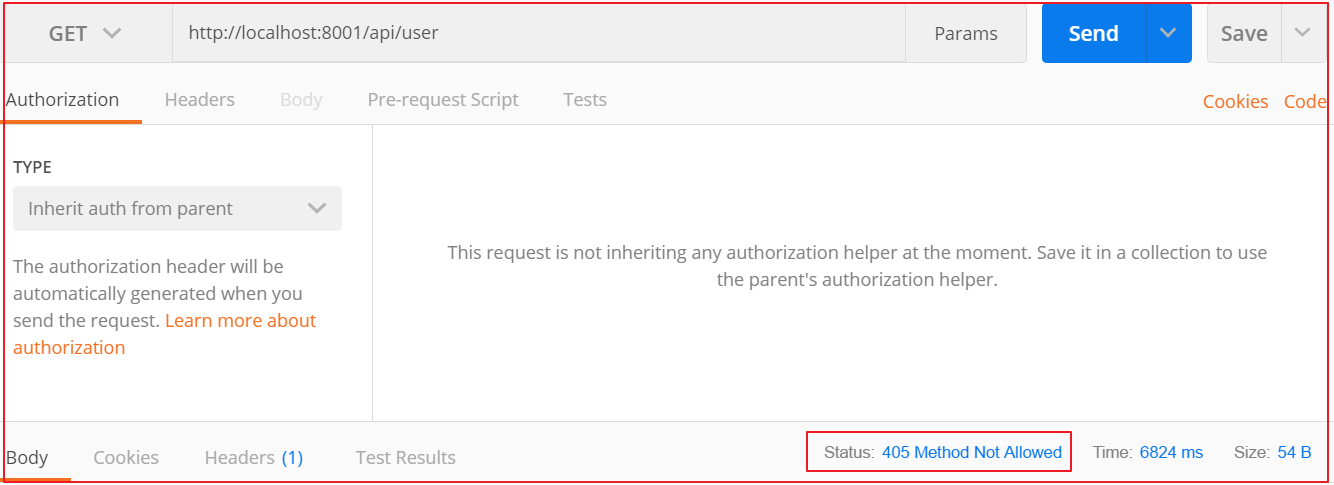
## 令牌加入到Header中

修改changgou-gateway-web的全局过滤器com.changgou.filter.AuthorizeFilter，实现将令牌信息添加到头文件中，代码如下：

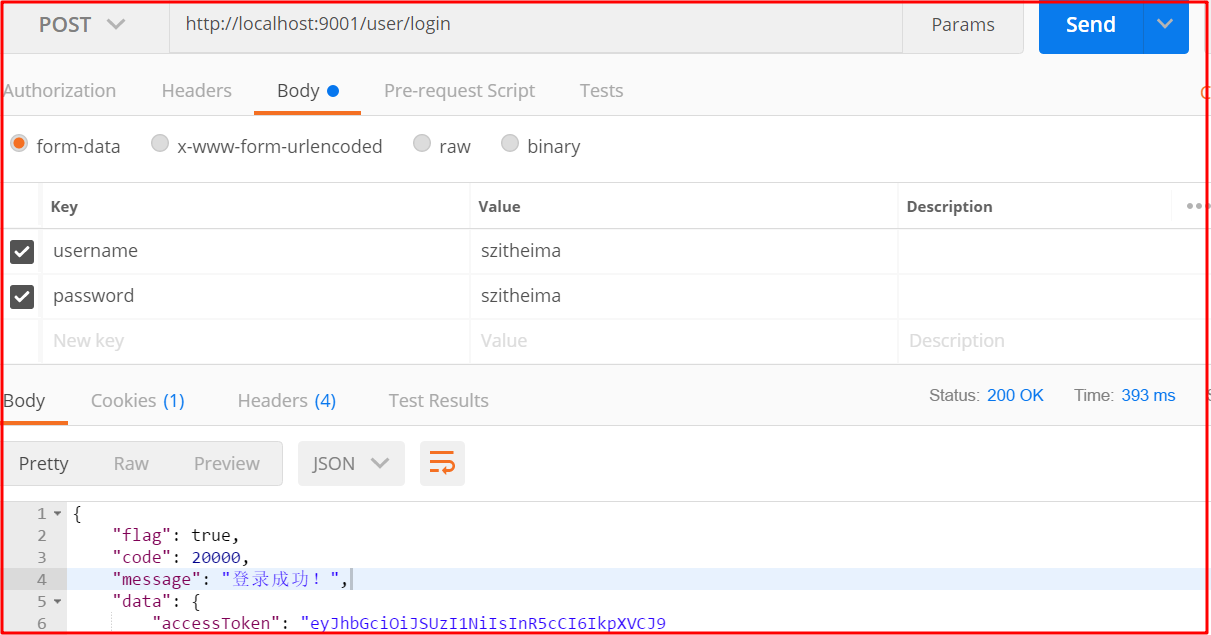
*/\*\*  
 \* 网关全局权限过滤器  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.filter  
 \*/*@Component  
**public class** AuthorizeFilter **implements** GlobalFilter, Ordered {  
  
 *//令牌的key* **private static final** String ***AUTHORIZE\_TOKEN*** = **"Authorization"**;  
  
 */\*\*  
 \* 拦截请求过滤规则逻辑**\** ***@return*** *\*/* @Override  
 **public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  
 *//1、获取Request、Response对象-exchange.get...* ServerHttpRequest request = exchange.getRequest();  
 ServerHttpResponse response = exchange.getResponse();  
 *//2、获取请求的URI-request.getURI().getPath()* String uri = request.getURI().getPath();  
 *//3、如果是登录请求-uri.startsWith，放行-chain.filter* **if**(uri.startsWith(**"/api/user/login"**)){  
 **return** chain.filter(exchange);  
 }**else**{ *//4、如果是非登录请求  
 //4.1 获取前端传入的令牌-从请求头中获取-request.getHeaders().getFirst* String token = request.getHeaders().getFirst(***AUTHORIZE\_TOKEN***);  
 *//4.2 如果头信息中没有，从请求参数中获取-request.getQueryParams().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 token = request.getQueryParams().getFirst(***AUTHORIZE\_TOKEN***);  
 }  
 *//4.3 如果请求参数中没有，从cookie中获取-request.getCookies().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 HttpCookie first = request.getCookies().getFirst(***AUTHORIZE\_TOKEN***);  
 **if**(first != **null**){  
 token = first.getValue();  
 }  
 }  
 *//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete* **if**(StringUtils.*isEmpty*(token)){  
 *//返回405，错误表示方法不允许访问* response.setStatusCode(HttpStatus.***METHOD\_NOT\_ALLOWED***);  
 **return** response.setComplete();  
 }**else**{ *// 4.5 如果获取到了令牌，解析令牌-JwtUtil.parseJWT，放行-chain.filter(exchange)* **try** {  
 *//Claims claims = JwtUtil.parseJWT(token);  
 //4.5.1解析成功-把令牌返回-request.mutate().header(key,value)* request.mutate().header(***AUTHORIZE\_TOKEN***, **"bearer "** + token);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//无效的认证* response.setStatusCode(HttpStatus.***UNAUTHORIZED***);  
 **return** response.setComplete();  
 }  
 **return** chain.filter(exchange);  
 }  
 }  
 }  
 */\*\*  
 \* 过滤器执行顺序设置  
 \** ***@return*** *\*/* @Override  
 **public int** getOrder() {  
 **return** 0;  
 }  
}

## 测试

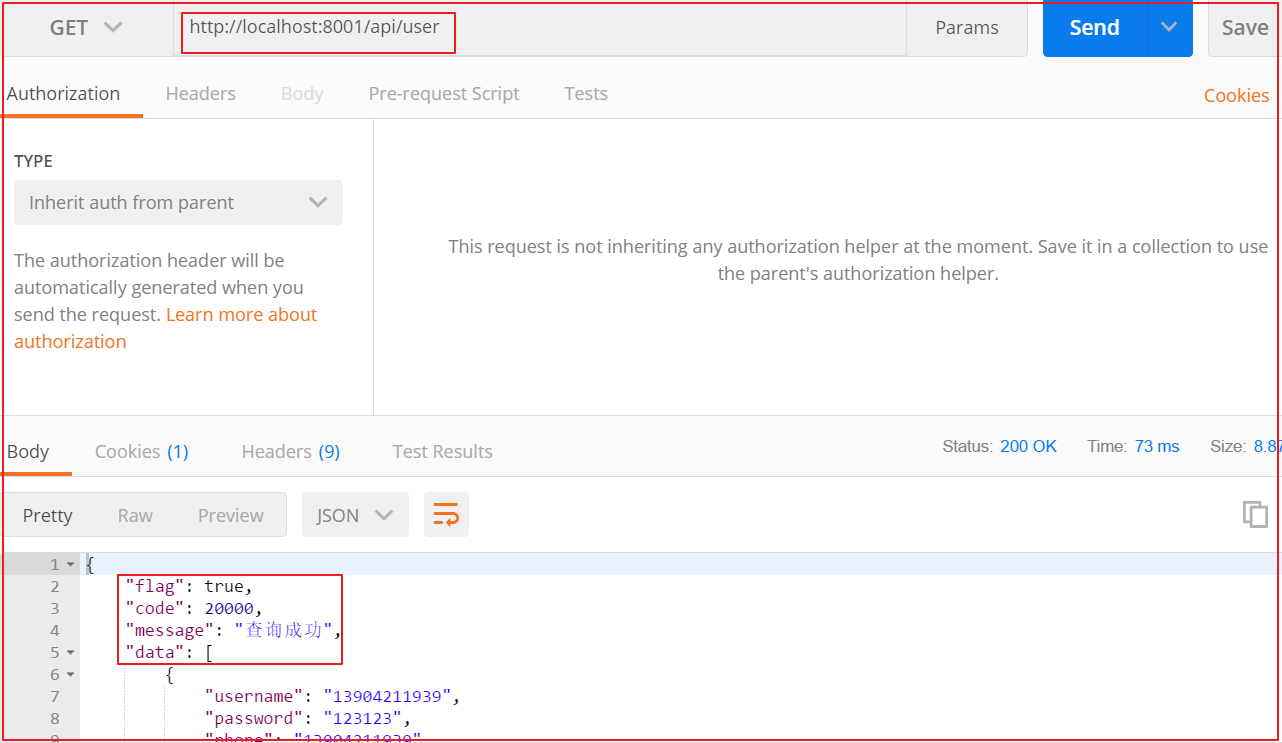
直接访问<http://localhost:8001/api/user>，发现权限不足。



先访问<http://localhost:9001/user/login> ，获取令牌



再次访问<http://localhost:8001/api/user>，发现可以取到数据了



## SpringSecurity权限控制

由于我们项目使用了微服务，任何用户都有可能使用任意微服务，此时我们需要控制相关权限，例如：普通用户角色不能使用用户的删除操作，只有管理员才可以使用,那么这个时候就需要使用到SpringSecurity的权限控制功能了。

### 角色加载

在changgou-user-oauth服务中，com.changgou.oauth.config.UserDetailsServiceImpl该类实现了加载用户相关信息，如下代码：

*/\*\*\*\*  
 \* 自定义授权认证  
 \** ***@param username*** *\** ***@return*** *\** ***@throws*** *UsernameNotFoundException  
 \*/*@Override  
**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  
 *//取出身份，如果身份为空说明没有认证* Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  
 *//没有认证统一采用httpbasic认证，httpbasic中存储了client\_id和client\_secret，开始认证client\_id和client\_secret* **if**(authentication==**null**){  
 ClientDetails clientDetails = **clientDetailsService**.loadClientByClientId(username);  
 **if**(clientDetails!=**null**){  
 *//秘钥* String clientSecret = clientDetails.getClientSecret();  
 *//静态方式* **return new** User(username,**new** BCryptPasswordEncoder().encode(clientSecret), AuthorityUtils.*commaSeparatedStringToAuthorityList*(**""**));  
 *//数据库查找方式  
 //return new User(username,clientSecret, AuthorityUtils.commaSeparatedStringToAuthorityList(""));* }  
 }  
  
 **if** (StringUtils.*isEmpty*(username)) {  
 **return null**;  
 }  
  
 *//根据用户名查询用户信息* String pwd = **new** BCryptPasswordEncoder().encode(**"szitheima"**);  
 *//创建用户的角色名列表：普通用户、销售员用户* String permissions = **"user,salesman"**;  
 *//创建User对象* UserJwt userDetails = **new** UserJwt(username,pwd,AuthorityUtils.*commaSeparatedStringToAuthorityList*(permissions));  
 **return** userDetails;  
}

上述代码给登录用户定义了2个角色，分别为salesman,user，这一块我们目前使用的是硬编码方式将角色写死了，后面会从数据库加载。

### 角色权限控制

在每个微服务中，需要获取用户的角色，然后根据角色识别是否允许操作指定的方法，Spring Security中定义了四个支持权限控制的表达式注解:

@PreAuthorize 在方法调用之前,基于表达式的计算结果来限制对方法的访问

@PostAuthorize 允许方法调用,但是如果表达式计算结果为false,将抛出一个安全性异常

@PostFilter 允许方法调用,但必须按照表达式来过滤方法的结果

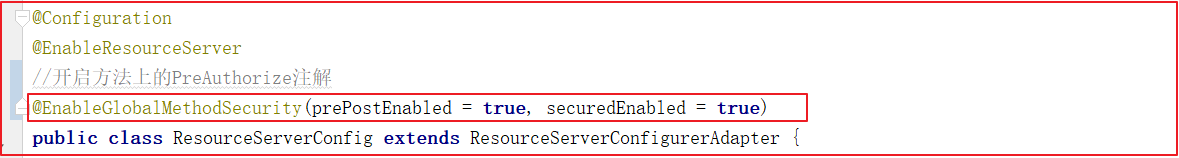
@PreFilter 允许方法调用,但必须在进入方法之前过滤输入值

在需要控制权限的方法上，我们可以添加@PreAuthorize注解，用于方法执行前进行权限检查，校验用户当前角色是否能访问该方法。

令牌加权限控制组合来将用户划定范围

#### 开启@PreAuthorize

在changgou-user-service的com.changgou.user.config.ResourceServerConfig类上添加@EnableGlobalMethodSecurity注解，用于开启@PreAuthorize的支持，代码如下：



#### 方法权限控制

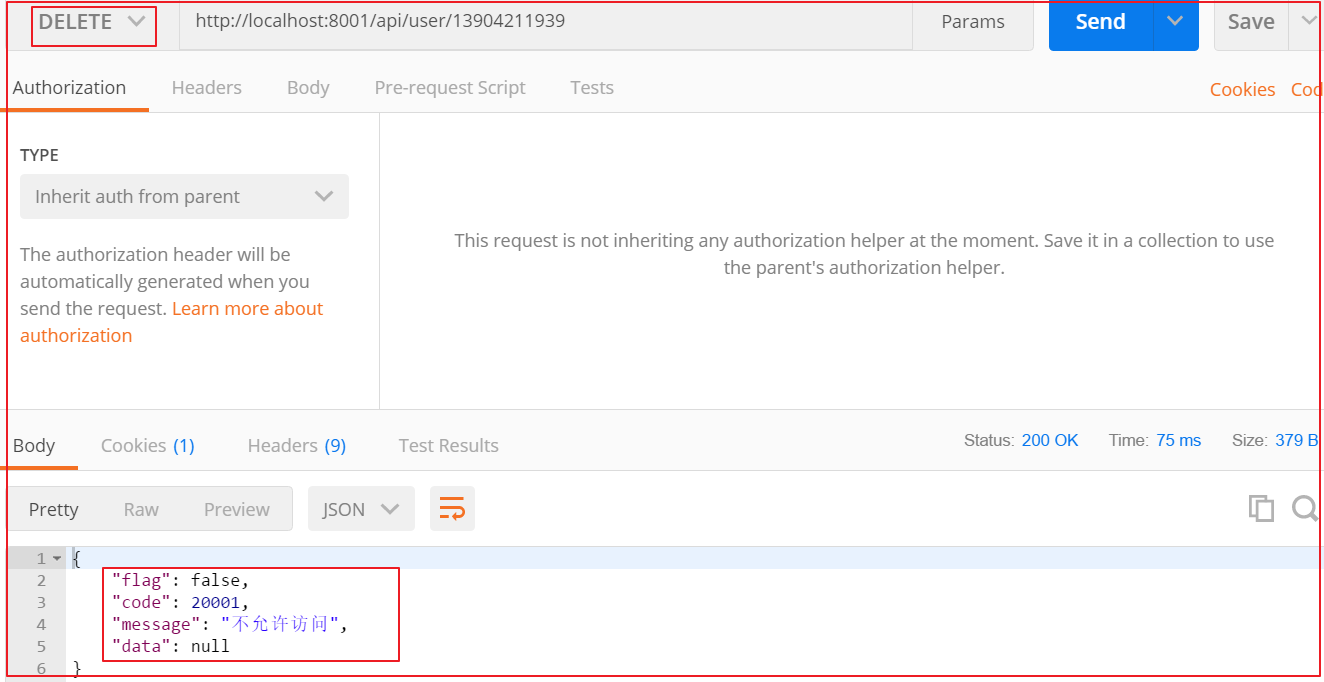
在changgoug-service-user微服务的com.changgou.user.controller.UserController类的delete()方法上添加权限控制注解@PreAuthorize，代码如下：



#### 测试

我们使用Postman测试，先创建令牌，然后访问微服务网关来调用user微服务的delete方法，效果如下：

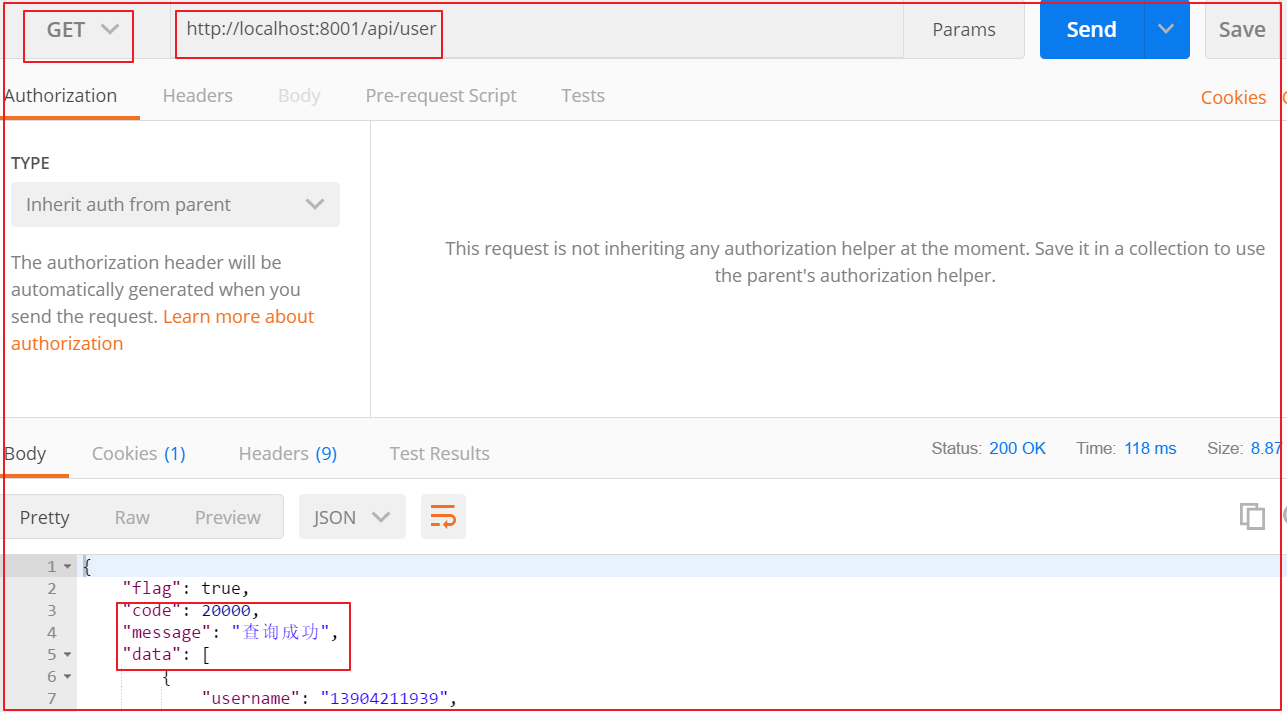
地址：<http://localhost:8001/api/user/13904211939> 提交方式：DELETE



发现上面无法访问，因为用户登录的时候，角色不包含admin角色，而delete方法需要admin角色，所以被拦截了。

我们再测试其他方法，其他方法没有配置拦截，所以用户登录后就会放行。

访问<http://localhost:8001/api/user> 效果如下：



#### 扩展知识点说明

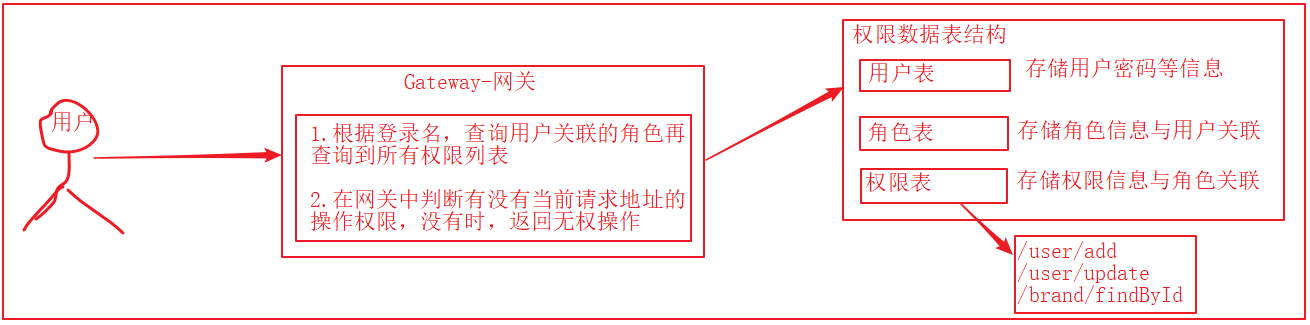
如果希望一个方法能被多个角色访问，在方法上配置:@PreAuthorize("hasAnyAuthority('admin','user')")

如果希望一个类都能被多个角色访问，在类上配置:@PreAuthorize("hasAnyAuthority('admin','user')")

还有一点就是，现在我们权限控制写死在Controller的注解上



大家以后在企业工作后，一般不会这么做，因为真实情况下我们每个用户拥有的角色与权限都不一定一样



# OAuth动态加载数据

前面OAuth我们用的数据都是静态的，在现实工作中，数据都是从数据库加载的，所以我们需要调整一下OAuth服务，从数据库加载相关数据。

* 客户端数据[生成令牌相关数据]
* 用户登录账号密码从数据库加载

## 客户端数据加载

### 数据介绍

**(1)客户端静态数据**

在changgou-user-oauth的com.changgou.oauth.config.AuthorizationServerConfig类中配置了客户端静态数据，主要用于配置客户端数据，代码如下：



**(2)客户端表结构介绍**

创建一个数据库changgou\_oauth,并在数据库中创建一张表，表主要用于记录客户端相关信息，表结构如下：

**CREATE** **TABLE** `oauth\_client\_details` (

`client\_id` varchar(48) **NOT** NULL COMMENT '客户端ID，主要用于标识对应的应用',

`resource\_ids` varchar(256) **DEFAULT** NULL,

`client\_secret` varchar(256) **DEFAULT** NULL COMMENT '客户端秘钥，BCryptPasswordEncoder加密算法加密',

`scope` varchar(256) **DEFAULT** NULL COMMENT '对应的范围',

`authorized\_grant\_types` varchar(256) **DEFAULT** NULL COMMENT '认证模式',

`web\_server\_redirect\_uri` varchar(256) **DEFAULT** NULL COMMENT '认证后重定向地址',

`authorities` varchar(256) **DEFAULT** NULL,

`access\_token\_validity` int(11) **DEFAULT** NULL COMMENT '令牌有效期',

`refresh\_token\_validity` int(11) **DEFAULT** NULL COMMENT '令牌刷新周期',

`additional\_information` varchar(4096) **DEFAULT** NULL,

`autoapprove` varchar(256) **DEFAULT** NULL,

**PRIMARY** **KEY** (`client\_id`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8;

字段说明：

client\_id：客户端id

resource\_ids：资源id（暂时不用）

client\_secret：客户端秘钥

scope：范围

access\_token\_validity：访问**token**的有效期（秒）

refresh\_token\_validity：刷新**token**的有效期（秒）

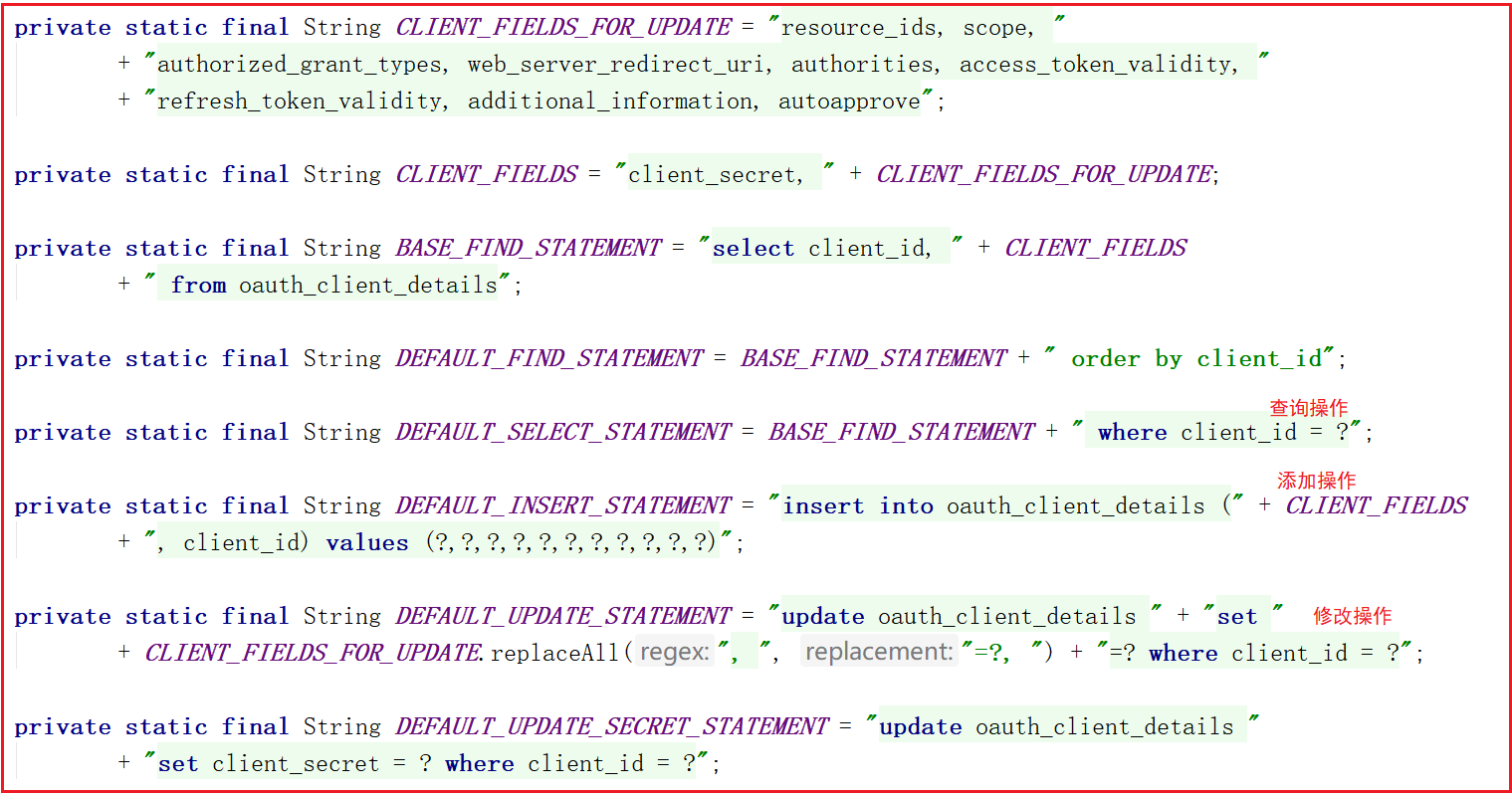
authorized\_grant\_type：授权类型:authorization\_code,password,refresh\_token,client\_credentials

导入2条记录到表中，SQL如下：数据中密文分别为changgou、szitheima

**INSERT** **INTO** `oauth\_client\_details` **VALUES** ('changgou', null, '$2a$10$wZRCFgWnwABfE60igAkBPeuGFuzk74V2jw3/trkdUZpnteCtJ9p9m', 'app', 'authorization\_code,password,refresh\_token,client\_credentials', 'http://localhost', null, '432000000', '432000000', null, null);

**INSERT** **INTO** `oauth\_client\_details` **VALUES** ('szitheima', null, '$2a$10$igxoCZxTbjWx5TrmfWEEpe/WFdwbUhbxik9BKTe9i64ZOSfnu/lqe', 'app', 'authorization\_code,password,refresh\_token,client\_credentials', 'http://localhost', null, '432000000', '432000000', null, null);

上述表结构属于SpringSecurity Oauth2.0所需的一个认证表结构，不能随意更改。相关操作在其他类中有所体现，如：org.springframework.security.oauth2.provider.client.JdbcClientDetailsService中的片段代码如下：



### 授权码模式-加载数据改造

#### 修改连接配置

从数据库加载数据，我们需要先配置数据库连接，在changgou-user-oauth的application.yml中配置连接信息，如下代码：

**datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_oauth?useUnicode=true&characterEncoding=utf-8&useSSL=false&allowMultiQueries=true&serverTimezone=UTC  
 **username**: root  
 **password**: 123456

#### 修改客户端加载源

修改changgou-user-oauth的com.changgou.oauth.config.AuthorizationServerConfig类的configure方法，将之前静态的客户端数据变成从数据库加载，修改如下：

*/\*\*\*  
 \* 客户端信息配置  
 \** ***@param clients*** *\** ***@throws*** *Exception  
 \*/*@Override  
**public void** configure(ClientDetailsServiceConfigurer clients) **throws** Exception {  
 */\*clients.inMemory()  
 .withClient("changgou") //客户端id  
 .secret("changgou") //秘钥  
 .redirectUris("http://localhost") //重定向地址  
 .accessTokenValiditySeconds(60) //访问令牌有效期  
 .refreshTokenValiditySeconds(60) //刷新令牌有效期  
 .authorizedGrantTypes(  
 "authorization\_code", //根据授权码生成令牌  
 "client\_credentials", //客户端认证  
 "refresh\_token", //刷新令牌  
 "password") //密码方式认证  
 .scopes("app"); //客户端范围，名称自定义，必填\*/  
 //从数据库加载信息* clients.jdbc(**dataSource**).clients(**this**.clientDetails());  
}

#### UserDetailsServiceImpl修改

将之前的加密方式去掉即可，代码如下：

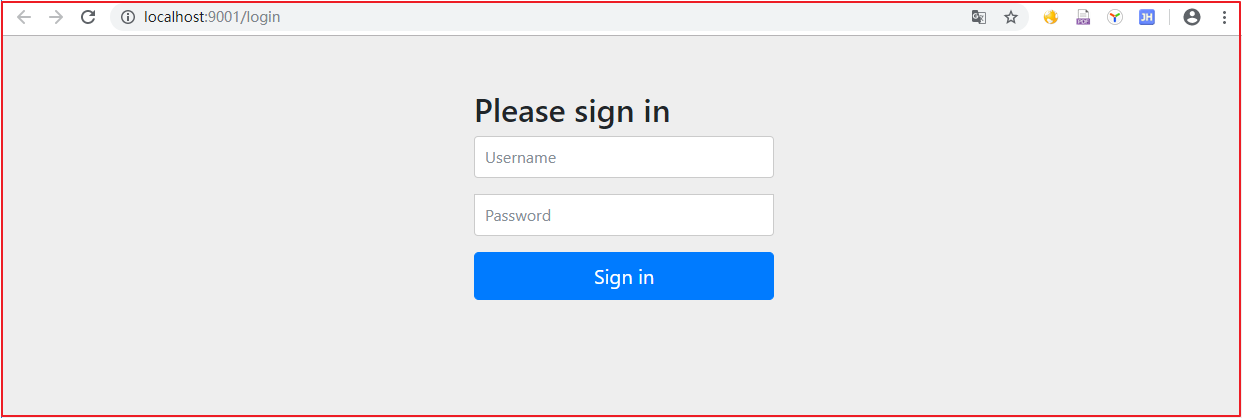
@Override  
**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  
 *//取出身份，如果身份为空说明没有认证* Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  
 *//没有认证统一采用httpbasic认证，httpbasic中存储了client\_id和client\_secret，开始认证client\_id和client\_secret* **if**(authentication==**null**){  
 ClientDetails clientDetails = **clientDetailsService**.loadClientByClientId(username);  
 **if**(clientDetails!=**null**){  
 *//秘钥* String clientSecret = clientDetails.getClientSecret();  
 *//静态方式  
 //return new User(username,new BCryptPasswordEncoder().encode(clientSecret), AuthorityUtils.commaSeparatedStringToAuthorityList(""));  
 //数据库查找方式* **return new** User(username,clientSecret, AuthorityUtils.*commaSeparatedStringToAuthorityList*(**""**));  
 }  
 }  
  
 **if** (StringUtils.*isEmpty*(username)) {  
 **return null**;  
 }  
  
 *//根据用户名查询用户信息* String pwd = **new** BCryptPasswordEncoder().encode(**"szitheima"**);  
 *//创建用户的角色名列表：普通用户、销售员用户* String permissions = **"user,salesman"**;  
 *//创建User对象* UserJwt userDetails = **new** UserJwt(username,pwd,AuthorityUtils.*commaSeparatedStringToAuthorityList*(permissions));  
 **return** userDetails;  
}

#### 测试

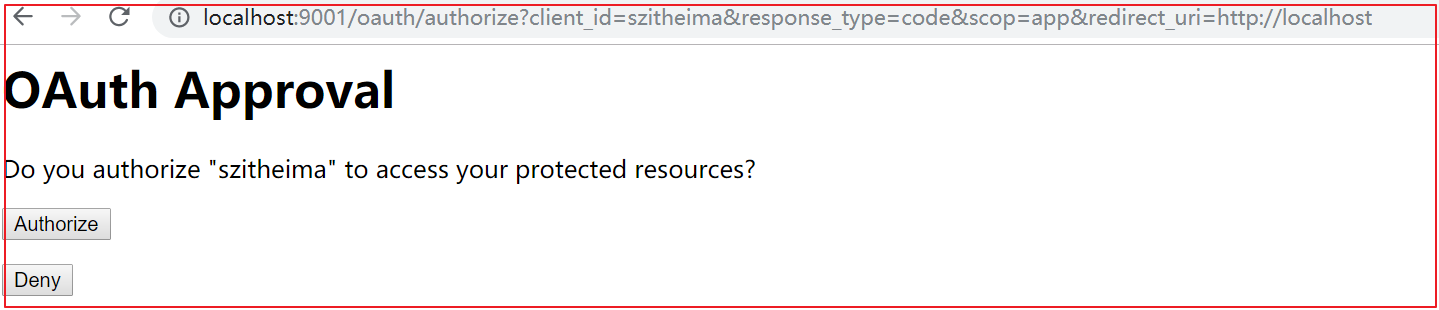
重启认证微服务授权码模式测试

访问：<http://localhost:9001/oauth/authorize?client_id=szitheima&response_type=code&scop=app&redirect_uri=http://localhost>

效果如下：



用户名对应应用id，密码对应秘钥。账号输入：szitheima 密码：szitheima，效果如下：



### 密码模式授权测试

我们之前编写的账号密码登录代码如下，每次都会加载指定的客户端ID和指定的秘钥，所以此时的客户端ID和秘钥固定了，输入的账号密码不再是客户端ID和秘钥了。



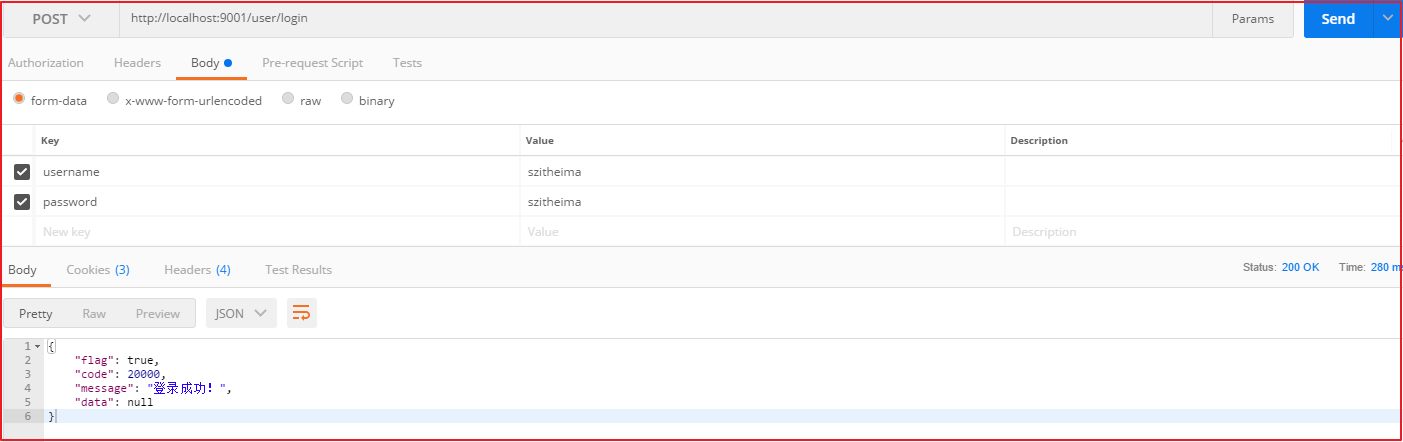
OAuth中的com.changgou.oauth.config.UserDetailsServiceImpl配置如下：



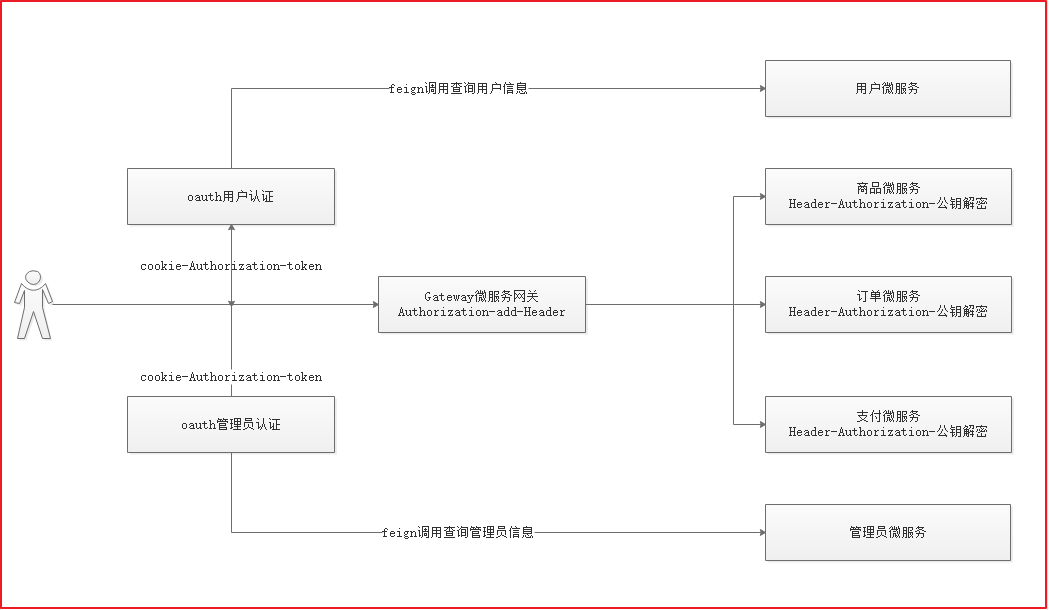
用户每次输入账号和密码，只要密码是szitheima，即可登录成功。

访问地址<http://localhost:9001/user/login>

输入账号密码均为szitheima,效果如下：



## 用户数据加载



因为我们目前整套系统是对内提供登录访问，所以每次用户登录的时候oauth需要调用用户微服务查询用户信息，如上图：

我们需要在用户微服务中提供用户信息查询的方法，并在oauth中使用feign调用即可。

在真实工作中，用户和管理员对应的oauth认证服务器会分开，网关也会分开，我们今天的课堂案例只实现用户相关的认证即可。 就是普通用户只是user权限，管理员是admin权限，会将不同权限的用户分开进入不同的微服务界面

### Feign创建

在changgou-service-user-api中创建com.changgou.user.feign.UserFeign，代码如下：

@FeignClient(name = **"user"**)  
@RequestMapping(**"user"**)  
**public interface** UserFeign {  
 */\*\*\*  
 \* 根据ID查询User数据  
 \*/* @GetMapping(**"load/{id}"**)  
 Result<User> findById(@PathVariable String id);  
}

### 修改UserController

修改changgou-service-user的UserController的findById方法，添加一个新的地址，用于加载用户信息，代码如下：

@GetMapping({**"/{id}"**,**"/load/{id}"**})  
**public** Result<User> findById(@PathVariable String id){  
 *//调用UserService实现根据主键查询User* User user = **userService**.findById(id);  
 **return new** Result<User>(**true**,StatusCode.***OK***,**"查询成功"**,user);  
}

### 放行查询用户方法

因为oauth服务需要调用查询用户信息，需要在changgou-service-user中放行/user/load/{id}方法,修改ResourceServerConfig，添加对/user/load/{id}的放行操作，代码如下：

*/\*\*\*  
 \* Http安全配置，对每个到达系统的http请求链接进行校验  
 \** ***@param http*** *\** ***@throws*** *Exception  
 \*/*@Override  
**public void** configure(HttpSecurity http) **throws** Exception {  
 *//所有请求必须认证通过* http.authorizeRequests()  
 *//下边的路径放行* .antMatchers(  
 **"/user/add"**,**"/user/load/\*"**). *//配置地址放行* permitAll()  
 .anyRequest().  
 authenticated(); *//其他地址需要认证授权*}

### oauth调用查询用户信息

oauth服务引入对user-api的依赖

<**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_user\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
</**dependency**>

修改oauth的com.changgou.oauth.config.UserDetailsServiceImpl的loadUserByUsername方法，调用UserFeign查询用户信息，代码如下：

*/\*\*\*\*  
 \* 自定义授权认证  
 \** ***@param username*** *\** ***@return*** *\** ***@throws*** *UsernameNotFoundException  
 \*/*@Override  
**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {  
 *//取出身份，如果身份为空说明没有认证* Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  
 *//没有认证统一采用httpbasic认证，httpbasic中存储了client\_id和client\_secret，开始认证client\_id和client\_secret* **if**(authentication==**null**){  
 ClientDetails clientDetails = **clientDetailsService**.loadClientByClientId(username);  
 **if**(clientDetails!=**null**){  
 *//秘钥* String clientSecret = clientDetails.getClientSecret();  
 *//静态方式  
 //return new User(username,new BCryptPasswordEncoder().encode(clientSecret), AuthorityUtils.commaSeparatedStringToAuthorityList(""));  
 //数据库查找方式* **return new** User(username,clientSecret, AuthorityUtils.*commaSeparatedStringToAuthorityList*(**""**));  
 }  
 }  
  
 **if** (StringUtils.*isEmpty*(username)) {  
 **return null**;  
 }  
  
 *//根据用户名查询用户信息  
 //String pwd = new BCryptPasswordEncoder().encode("szitheima");* String pwd = **userFeign**.findById(username).getData().getPassword();  
  
 *//创建用户的角色名列表：普通用户、销售员用户* String permissions = **"user,salesman"**;  
 *//创建User对象* UserJwt userDetails = **new** UserJwt(username,pwd,AuthorityUtils.*commaSeparatedStringToAuthorityList*(permissions));  
 **return** userDetails;  
}

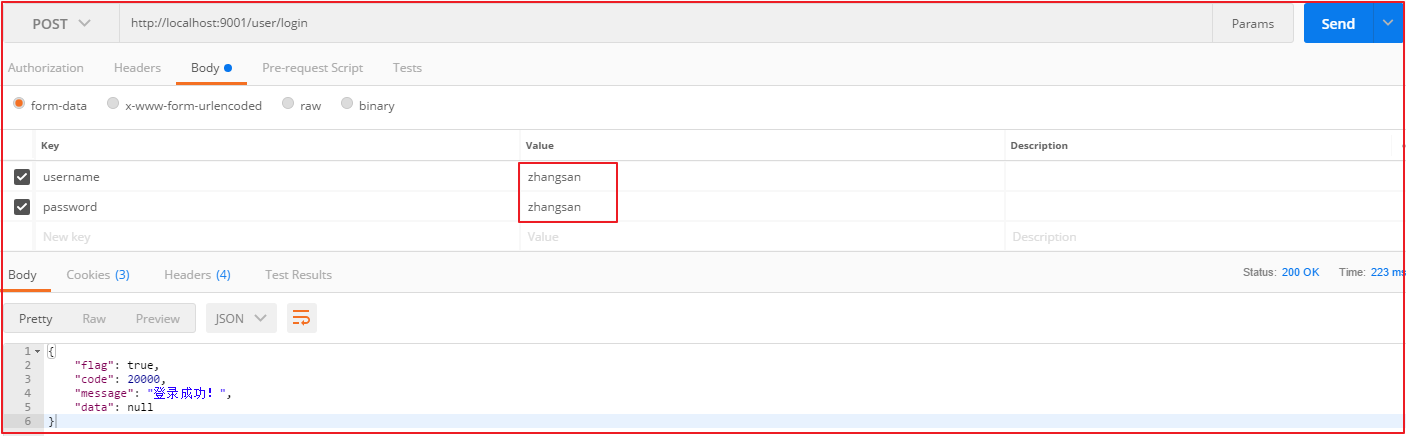
### feign开启

修改com.changgou.OAuthApplication开启Feign客户端功能

@SpringBootApplication  
@EnableDiscoveryClient  
@MapperScan(basePackages = **"com.changgou.auth.dao"**)  
@EnableFeignClients(basePackages = **"com.changgou.user.feign"**)  
**public class** OAuthApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(OAuthApplication.**class**,args);  
 }  
   
 @Bean(name = **"restTemplate"**)  
 **public** RestTemplate restTemplate() {  
 **return new** RestTemplate();  
 }  
  
}

### 测试

我们换个数据库中的账号密码登录，分别输入zhangsan，效果如下：



# 购物车

购物车分为用户登录购物车和未登录购物车操作，国内知名电商京东用户登录和不登录都可以操作购物车，如果用户不登录，操作购物车可以将数据存储到Cookie或者WebSQL或者SessionStorage中，用户登录后购物车数据可以存储到Redis中，再将之前未登录加入的购物车合并到Redis中即可。

淘宝天猫则采用了另外一种实现方案，用户要想将商品加入购物车，必须先登录才能操作购物车。

我们今天实现的购物车是天猫解决方案，即用户必须先登录才能使用购物车功能。

购物车类型：2种

1.用户未登录和已登录都可以使用购物车

不登录,购物车数据存入到客户端(Cookie|WebSQL[H5])

已登录,可以存储到服务端(MySQL、Redis、MongoDB)

状态切换:用户未登录，商品加入购物车->登录（之前可以存着购物车数据）,购物车合并

2.用户未登录不能使用购物车，已登录才能使用购物车

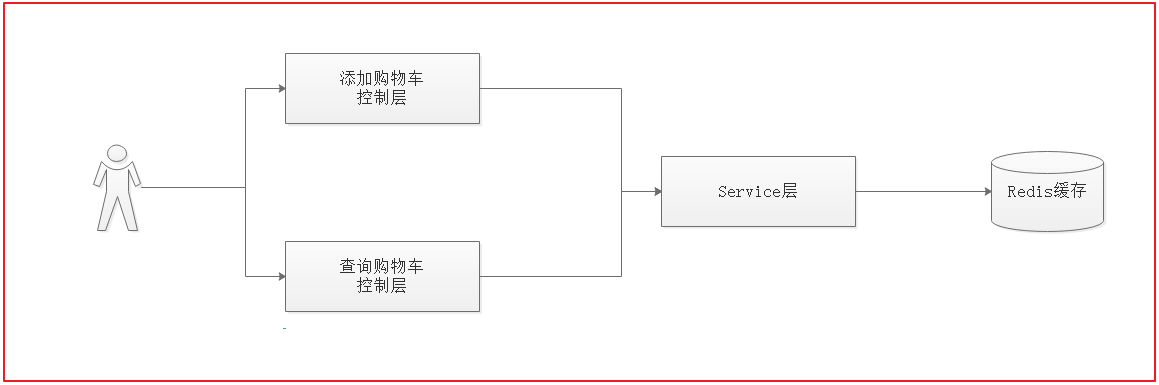
已登录,可以存储到服务端(MySQL、Redis、MongoDB)

## 购物车分析

用户在商品详细页点击加入购物车，提交商品SKU编号和购买数量，添加到购物车。购物车展示页面如下：



### 购物车实现思路



我们实现的是用户登录后的购物车，用户将商品加入购物车的时候，直接将要加入购物车的详情存入到Redis即可。每次查看购物车的时候直接从Redis中获取。

### 表结构分析

用户登录后将商品加入购物车，需要存储商品详情以及购买数量，购物车详情表如下：

changgou\_order数据中tb\_order\_item表：

**CREATE** **TABLE** `tb\_order\_item` (

`id` varchar(20) **COLLATE** utf8\_bin **NOT** NULL COMMENT 'ID',

`category\_id1` int(11) **DEFAULT** NULL COMMENT '1级分类',

`category\_id2` int(11) **DEFAULT** NULL COMMENT '2级分类',

`category\_id3` int(11) **DEFAULT** NULL COMMENT '3级分类',

`spu\_id` varchar(20) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT 'SPU\_ID',

`sku\_id` bigint(20) **NOT** NULL COMMENT 'SKU\_ID',

`order\_id` bigint(20) **NOT** NULL COMMENT '订单ID',

`name` varchar(200) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '商品名称',

`price` int(20) **DEFAULT** NULL COMMENT '单价',

`num` int(10) **DEFAULT** NULL COMMENT '数量',

`money` int(20) **DEFAULT** NULL COMMENT '总金额',

`pay\_money` int(11) **DEFAULT** NULL COMMENT '实付金额',

`image` varchar(200) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '图片地址',

`weight` int(11) **DEFAULT** NULL COMMENT '重量',

`post\_fee` int(11) **DEFAULT** NULL COMMENT '运费',

`is\_return` char(1) **COLLATE** utf8\_bin **DEFAULT** NULL COMMENT '是否退货',

**PRIMARY** **KEY** (`id`),

**KEY** `item\_id` (`sku\_id`),

**KEY** `order\_id` (`order\_id`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8 **COLLATE**=utf8\_bin;

购物车详情表其实就是订单详情表结构，只是目前临时存储数据到Redis，等用户下单后才将数据从Redis取出存入到MySQL中。

## 订单购物车微服务

我们先搭建一个订单购物车微服务工程，按照如下步骤实现即可。

### 导入资源

搭建订单购物车微服务，工程名字changgou\_service\_order并搭建对应的api工程changgou\_service\_order\_api,将生成好的dao和相关文件拷贝到工程中，以及生成好的Pojo拷贝到API工程中。同时在changgou\_service\_order中引入changgou\_service\_order\_api,如下图：

其中，serice\_order要添加api的依赖引入：

<**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_order\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
</**dependency**>

changgou-service-order:



changgou-service-order-api:



### application.yml配置

在changgou-service-order的resources中添加application.yml配置文件，代码如下：

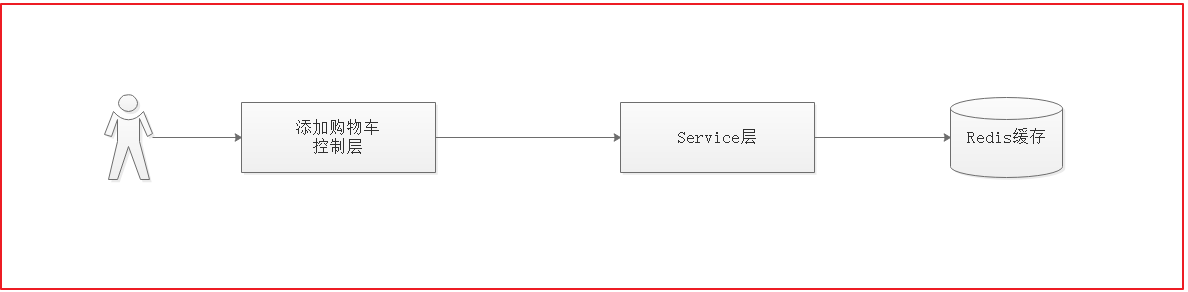
**server**:  
 **port**: 18089  
**spring**:  
 **application**:  
 **name**: order  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_order?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
 **redis**:  
 **host**: 192.168.211.132  
 **port**: 6379  
 **main**:  
 **allow-bean-definition-overriding**: **true  
  
eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true**

### 创建启动类

在changgou-service-order的resources中创建启动类，代码如下：

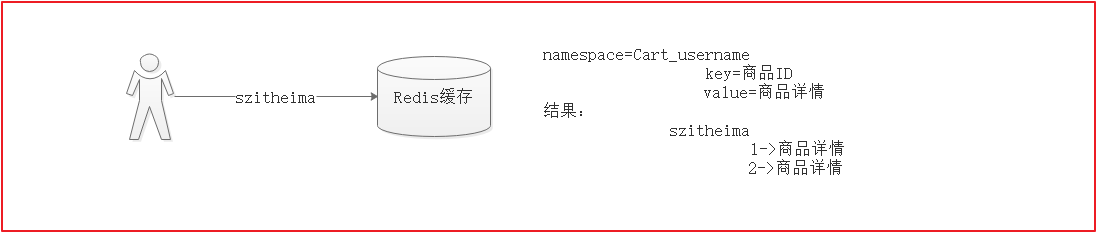
@SpringBootApplication  
@EnableEurekaClient  
@MapperScan(basePackages = {**"com.changgou.order.dao"**})  
**public class** OrderApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(OrderApplication.**class**,args);  
 }  
}

## 添加购物车



用户添加购物车，只需要将要加入购物车的商品存入到Redis中即可。一个用户可以将多件商品加入购物车，存储到Redis中的数据可以采用Hash类型。

选Hash类型可以将用户的用户名作为namespace的一部分，将指定商品加入购物车，则往对应的namespace中增加一个key和value，key是商品ID，value是加入购物车的商品详情，如下图：



### feign创建

下订单需要调用feign查看商品信息，我们先创建feign分别根据ID查询Sku和Spu信息，在changgou-service-goods-api工程中的SkuFeign和SpuFeign根据ID查询方法如下：

com.changgou.goods.feign.SkuFeign

@FeignClient(name = **"goods"**)  
@RequestMapping(**"sku"**)  
**public interface** SkuFeign {  
  
 */\*\*\*  
 \* 根据ID查询Sku数据  
 \** ***@param id*** *\** ***@return*** *\*/* @GetMapping(**"/{id}"**)  
 **public** Result<Sku> findById(@PathVariable Long id);  
}

com.changgou.goods.feign.SpuFeign

@FeignClient(name = **"goods"**)  
@RequestMapping(**"spu"**)  
**public interface** SpuFeign {  
 @GetMapping(**"/{id}"**)  
 **public** Result<Spu> findById(@PathVariable Long id);  
}

### 业务层

**业务层接口**

在changgou-service-order微服务中创建com.changgou.order.service.CartService接口，代码如下：

**public interface** CartService {  
  
 */\*\*\*  
 \* 添加购物车  
 \** ***@param num****:购买商品数量  
 \** ***@param skuId****：购买商品的skuId  
 \** ***@param username****：购买用户  
 \** ***@return*** *\*/* **void** add(Integer num, Long skuId, String username);  
}

**业务层接口实现类**

在changgou-service-order微服务中创建接口实现类com.changgou.order.service.impl.CartServiceImpl,代码如下：

@Service  
**public class** CartServiceImpl **implements** CartService {  
 @Autowired  
 **private** RedisTemplate **redisTemplate**;  
  
 @Autowired  
 **private** SkuFeign **skuFeign**;  
  
 @Autowired  
 **private** SpuFeign **spuFeign**;  
  
 @Override  
 **public void** add(Integer num, Long skuId, String username) {  
 *//查询SKU* Result<Sku> resultSku = **skuFeign**.findById(skuId);  
 **if** (resultSku != **null** && resultSku.isFlag()) {  
 *//获取SKU* Sku sku = resultSku.getData();  
 *//获取SPU* Spu spu = **spuFeign**.findById(sku.getSpuId()).getData().getSpu();  
  
 *//将SKU转换成OrderItem* OrderItem orderItem = **new** OrderItem();  
 orderItem.setSpuId(sku.getSpuId());  
 orderItem.setSkuId(sku.getId());  
 orderItem.setName(sku.getName());  
 orderItem.setPrice(sku.getPrice());  
 orderItem.setNum(num);  
 orderItem.setMoney(num \* orderItem.getPrice()); *//单价\*数量* orderItem.setPayMoney(num \* orderItem.getPrice()); *//实付金额* orderItem.setImage(sku.getImage());  
 orderItem.setWeight(sku.getWeight() \* num); *//重量=单个重量\*数量  
  
 //分类ID设置* orderItem.setCategoryId1(spu.getCategory1Id());  
 orderItem.setCategoryId2(spu.getCategory2Id());  
 orderItem.setCategoryId3(spu.getCategory3Id());  
  
 */\*\*\*\*\*\*  
 \* 购物车数据存入到Redis  
 \* namespace = Cart\_[username]  
 \* key=skuId  
 \* value=OrderItem  
 \*/* **redisTemplate**.boundHashOps(**"Cart\_"** + username).put(skuId, orderItem);  
 }  
 }  
}

### 控制层

在changgou-service-order微服务中创建com.changgou.order.controller.CartController，代码如下：

@RestController  
@CrossOrigin  
@RequestMapping(value = **"/cart"**)  
**public class** CartController {  
  
 @Autowired  
 **private** CartService **cartService**;  
  
 */\*\*\*  
 \* 加入购物车  
 \** ***@param num****:购买的数量  
 \** ***@param id****：购买的商品(SKU)ID  
 \** ***@return*** *\*/* @RequestMapping(value = **"/add"**)  
 **public** Result add(Integer num, Long id){  
 *//用户名* String username=**"szitheima"**;  
 *//将商品加入购物车* **cartService**.add(num,id,username);  
 **return new** Result(**true**, StatusCode.***OK***,**"加入购物车成功！"**);  
 }  
}

### feign配置

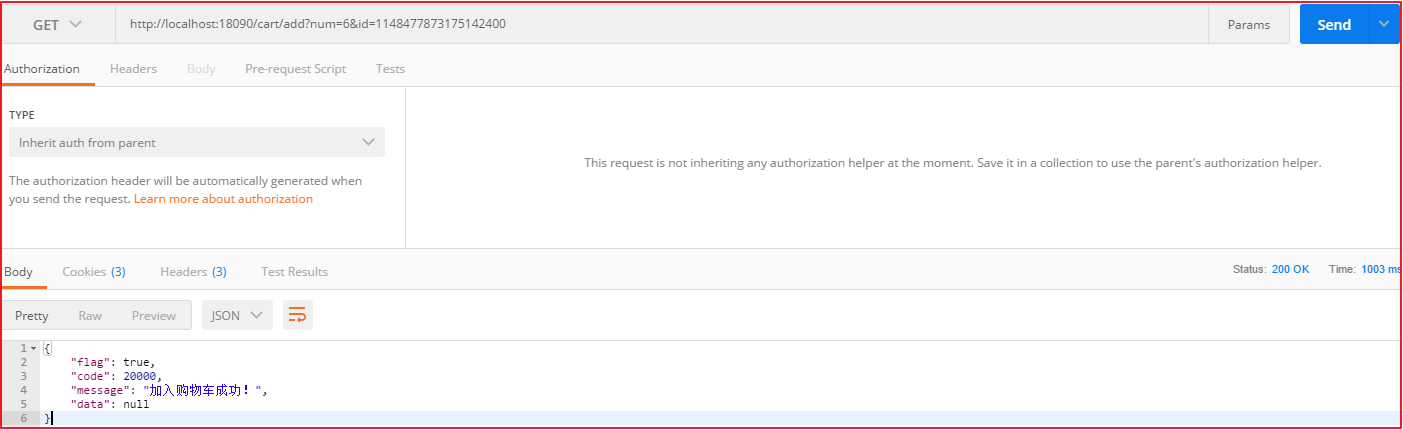
修改com.changgou.OrderApplication开启Feign客户端：

@SpringBootApplication  
@EnableEurekaClient  
@MapperScan(basePackages = {**"com.changgou.order.dao"**})  
@EnableFeignClients(basePackages = {**"com.changgou.goods.feign"**})  
**public class** OrderApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(OrderApplication.**class**,args);  
 }  
}

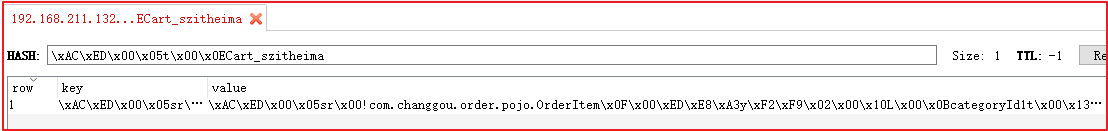
### 测试

添加购物车，效果如下：

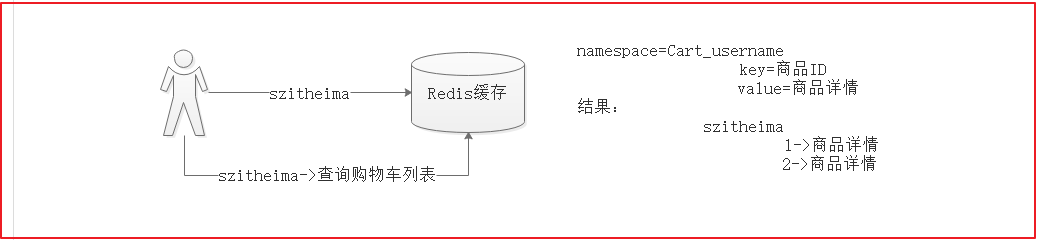
请求地址<http://localhost:18089/cart/add?num=2&id=1148477873175142400>



Redis缓存中的数据



## 购物车列表



接着我们实现一次购物车列表操作。因为存的时候是根据用户名往Redis中存储用户的购物车数据的，所以我们这里可以将用户的名字作为key去Redis中查询对应的数据。

### 业务层

**业务层接口**

修改changgou-service-order微服务的com.changgou.order.service.CartService接口，添加购物车列表方法，代码如下：

*/\*\*\*  
 \* 查询用户的购物车数据  
 \** ***@param username*** *\** ***@return*** *\*/*List<OrderItem> list(String username);

**业务层接口实现类**

修改changgou-service-order微服务的com.changgou.order.service.impl.CartServiceImpl类，添加购物车列表实现方法，代码如下：

*/\*\*\*  
 \* 查询用户购物车数据  
 \** ***@return*** *\*/*@Override  
**public** List<OrderItem> list(String username) {  
 *//查询所有购物车数据* List<OrderItem> orderItems = **redisTemplate**.boundHashOps(**"Cart\_"**+username).values();  
 **return** orderItems;  
}

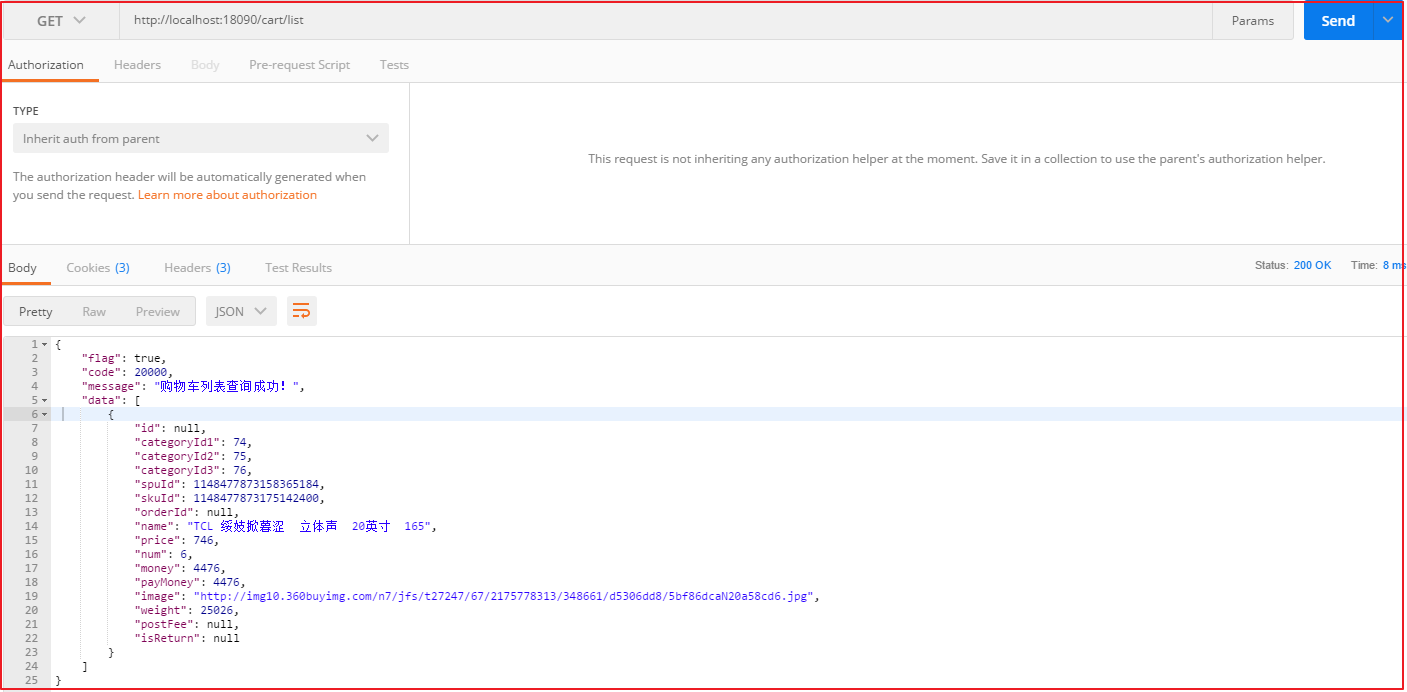
### 控制层

修改changgou-service-order微服务的com.changgou.order.controller.CartController类，添加购物车列表查询方法，代码如下：

*/\*\*\*  
 \* 查询用户购物车列表  
 \** ***@return*** *\*/*@GetMapping(value = **"/list"**)  
**public** Result list() {  
 *//用户名* String username = **"szitheima"**;  
 List<OrderItem> orderItems = **cartService**.list(username);  
 **return new** Result(**true**, StatusCode.***OK***, **"购物车列表查询成功！"**, orderItems);  
}

### 测试

使用Postman访问 GET <http://localhost:18089/cart/list> ,效果如下：



## 问题处理

### 删除商品购物车



我们发现个问题，就是用户将商品加入购物车，无论数量是正负，都会执行添加购物车，如果数量如果<=0，应该移除该商品的。

修改changgou-service-order的com.changgou.order.service.impl.CartServiceImpl的add方法，添加如下代码：

@Override  
**public void** add(Integer num, Long skuId, String username) {  
 **if**(num <= 0){  
 *//如果商品数量小于等于0，删除当前商品* **redisTemplate**.boundHashOps(**"Cart\_"** + username).delete(skuId);  
 **return**;  
 }  
 *//查询SKU* Result<Sku> resultSku = **skuFeign**.findById(skuId);  
 **if** (resultSku != **null** && resultSku.isFlag()) {  
 *//获取SKU* Sku sku = resultSku.getData();  
 *//获取SPU* Spu spu = **spuFeign**.findById(sku.getSpuId()).getData().getSpu();  
  
 *//将SKU转换成OrderItem* OrderItem orderItem = **new** OrderItem();  
 orderItem.setSpuId(sku.getSpuId());  
 orderItem.setSkuId(sku.getId());  
 orderItem.setName(sku.getName());  
 orderItem.setPrice(sku.getPrice());  
 orderItem.setNum(num);  
 orderItem.setMoney(num \* orderItem.getPrice()); *//单价\*数量* orderItem.setPayMoney(num \* orderItem.getPrice()); *//实付金额* orderItem.setImage(sku.getImage());  
 orderItem.setWeight(sku.getWeight() \* num); *//重量=单个重量\*数量  
  
 //分类ID设置* orderItem.setCategoryId1(spu.getCategory1Id());  
 orderItem.setCategoryId2(spu.getCategory2Id());  
 orderItem.setCategoryId3(spu.getCategory3Id());  
  
 */\*\*\*\*\*\*  
 \* 购物车数据存入到Redis  
 \* namespace = Cart\_[username]  
 \* key=skuId  
 \* value=OrderItem  
 \*/* **redisTemplate**.boundHashOps(**"Cart\_"** + username).put(skuId, orderItem);  
 }  
}

### 数据精度丢失问题

SkuId是Long类型，在页面输出的值过大时候会存在精度丢失问题，我们只需要在OrderItem的SkuId上加上字符串序列化类型就可以了，代码如下：



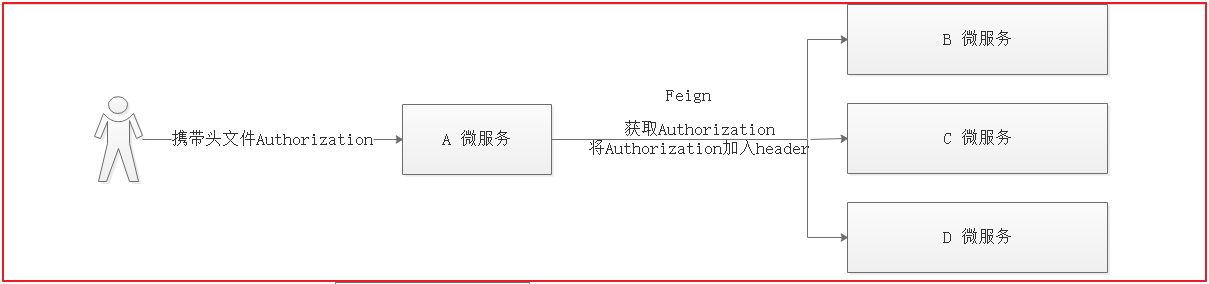
# 用户身份识别

## 购物车需求分析



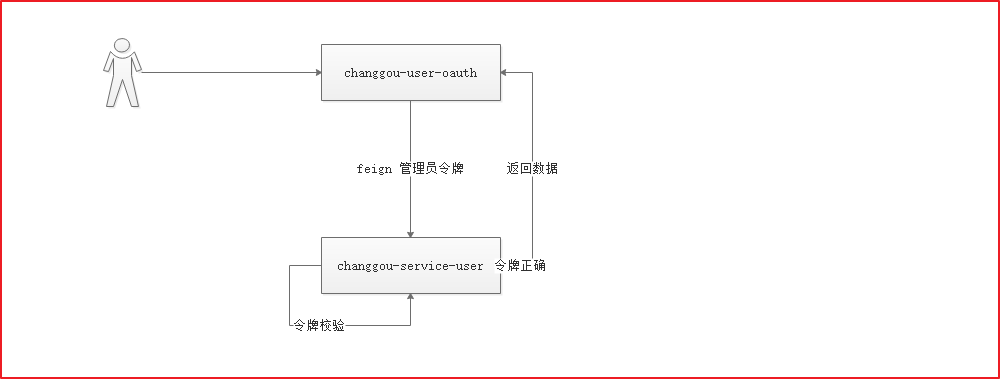
购物车功能已经做完了，但用户我们都是硬编码写死的。用户要想将商品加入购物车，必须得先登录授权，登录授权后再经过微服务网关，微服务网关需要过滤判断用户请求是否存在令牌，如果存在令牌，才能再次访问微服务，此时网关会通过过滤器将令牌数据再次存入到头文件中，将令牌数据传递给购物车订单微服务，到了购物车订单微服务的时候，此时微服务需要校验令牌数据，如果令牌正确，才能使用购物车功能，并解析令牌数据获取用户信息。

## 微服务之间认证



如上图：因为微服务之间并没有传递头文件，所以我们可以定义一个拦截器，每次微服务调用之前都先检查下头文件，将请求的头文件中的令牌数据再放入到header中，再调用其他微服务即可。

oauth认证

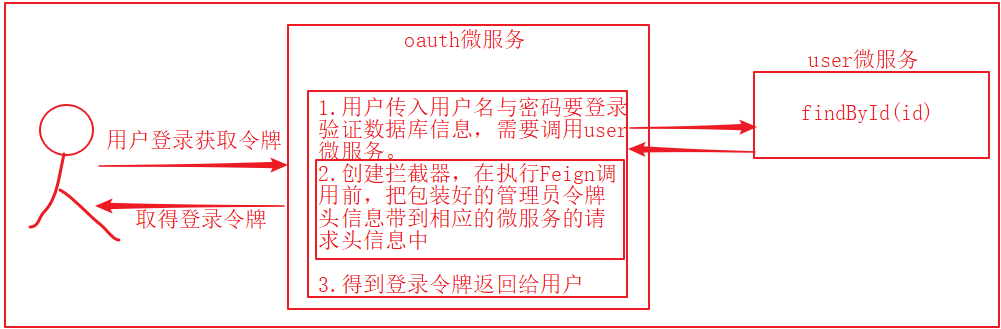


### 令牌创建

因为changgou-user-oauth服务无法自己登录，我们需要手动为它创建管理员令牌，可以创建一个工具类，专门用于生成特殊令牌，在changgou-user-oauth中创建com.changgou.oauth.util.JwtToken代码参考com.changgou.token.CreateJwtTest修改如下：

*/\*\*  
 \* 管理员令牌生成器  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.oauth.util  
 \*/***public class** JwtToken {  
 */\*\*\*  
 \* 获取管理员令牌  
 \** ***@return*** *\*/* **public static** String adminJwt(){  
 *//证书文件路径* String key\_location=**"changgou.jks"**;  
 *//秘钥库密码* String key\_password=**"changgou"**;  
 *//秘钥密码* String keypwd = **"changgou"**;  
 *//秘钥别名* String alias = **"changgou"**;  
  
 *//访问证书路径* ClassPathResource resource = **new** ClassPathResource(key\_location);  
  
 *//创建秘钥工厂* KeyStoreKeyFactory keyStoreKeyFactory = **new** KeyStoreKeyFactory(resource,key\_password.toCharArray());  
  
 *//读取秘钥对(公钥、私钥)* KeyPair keyPair = keyStoreKeyFactory.getKeyPair(alias,keypwd.toCharArray());  
  
 *//获取私钥* RSAPrivateKey rsaPrivate = (RSAPrivateKey) keyPair.getPrivate();  
  
 *//定义Payload* Map<String, Object> tokenMap = **new** HashMap<>();  
 tokenMap.put(**"id"**, **"1"**);  
 tokenMap.put(**"name"**, **"itheima"**);  
 *//角色列表* String[] roles = {**"admin"**,**"user"**};  
 tokenMap.put(**"authorities"**, roles);  
  
 *//生成Jwt令牌-使用私钥* Jwt jwt = JwtHelper.*encode*(JSON.*toJSONString*(tokenMap), **new** RsaSigner(rsaPrivate));  
  
 *//取出令牌* **return** jwt.getEncoded();  
 }  
}

### 拦截器创建



在changgou-user-oauth中创建com.changgou.oauth.interceptor.FeignOauth2RequestInterceptor拦截器，携带管理员令牌，代码如下：

*/\*\*  
 \* Feign调用拦截器-会在调用所有Feign前执行apply方法的逻辑  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.oauth.interceptor  
 \*/*@Configuration  
**public class** FeignOauth2RequestInterceptor **implements** RequestInterceptor {  
 @Override  
 **public void** apply(RequestTemplate requestTemplate) {  
 *//创建管理员令牌* String token = **"bearer "** + JwtToken.*adminJwt*();  
 *//将令牌加入请求头信息中* requestTemplate.header(**"Authorization"**, token);  
 }  
}

当然，如果还学要携带其他已经存在的头文件信息，可以优化一下该拦截器，代码如下：

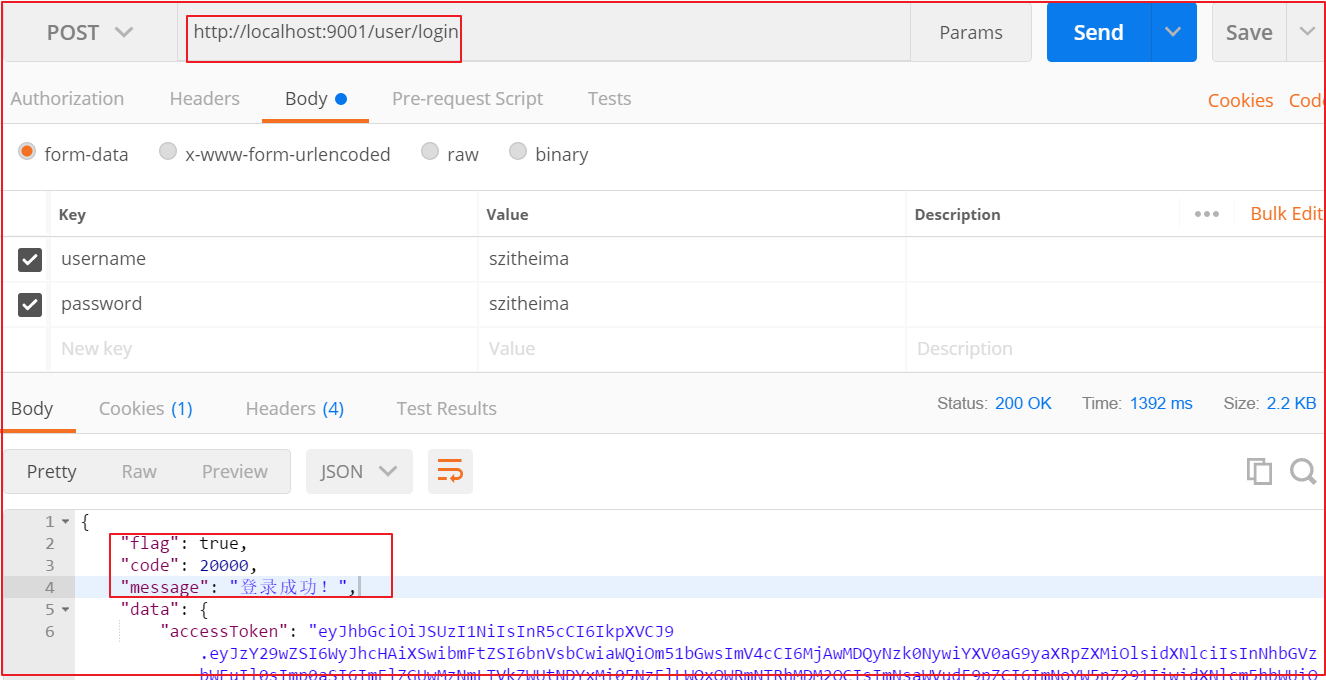
*/\*\*  
 \* Feign调用拦截器-会在调用所有Feign前执行apply方法的逻辑  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.oauth.interceptor  
 \*/*@Configuration  
**public class** FeignOauth2RequestInterceptor **implements** RequestInterceptor {  
 @Override  
 **public void** apply(RequestTemplate requestTemplate) {  
 *//创建管理员令牌* String token = **"bearer "** + JwtToken.*adminJwt*();  
 *//将令牌加入请求头信息中* requestTemplate.header(**"Authorization"**, token);  
  
 *//使用RequestContextHolder工具获取request相关变量* ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
 **if** (attributes != **null**) {  
 *//取出request* HttpServletRequest request = attributes.getRequest();  
 *//获取所有头文件信息的key* Enumeration<String> headerNames = request.getHeaderNames();  
 **if** (headerNames != **null**) {  
 **while** (headerNames.hasMoreElements()) {  
 *//头文件的key* String name = headerNames.nextElement();  
 *//头文件的value* String values = request.getHeader(name);  
 *//将令牌数据添加到头文件中* requestTemplate.header(name, values);  
 }  
 }  
 }  
 }  
}

### 测试

修改changgou\_service\_user工程的com.changgou.user.config.ResourceServerConfig的configure方法，把我们之前放行的/user/load/\*删除掉，代码如下

*/\*\*\*  
 \* Http安全配置，对每个到达系统的http请求链接进行校验  
 \** ***@param http*** *\** ***@throws*** *Exception  
 \*/*@Override  
**public void** configure(HttpSecurity http) **throws** Exception {  
 *//所有请求必须认证通过* http.authorizeRequests()  
 *//下边的路径放行* .antMatchers(  
 *//删除"/user/load/\*"的放行策略* **"/user/add"**). *//配置地址放行* permitAll()  
 .anyRequest().  
 authenticated(); *//其他地址需要认证授权*}

重启service\_user与oauth微服务，访问<http://localhost:9001/user/login> ，再次登录一次，同样可以登录



## 容断器隔离模式介绍

### 介绍

hystrix隔离模式目前有两种方式：信号量模式(SEMAPHORE)和线程池模式(THREAD)。

信号量并不支持超时，当被调服务发生问题时，有少部分用户会长时间无法得到响应。

线程池模式无法传递Header

信号量和线程池对比：

|  |  |  |
| --- | --- | --- |
| 比较项 | 线程池隔离 | 信号量隔离 |
| 线程 | 与调用线程非相同线程 | 与调用线程相同（jetty线程） |
| 开销大小 | 排队、调度、上下文开销等 | 无线程切换，开销低 |
| 异步 | 支持 | 不支持 |
| 并发支持 | 支持（最大线程池大小） | 支持（最大信号量上限） |
| 是否支持超时 | 支持 | 不支持 |

### 测试

微服务之间相互调用，如果开启feign熔断，再使用Feign调用数据，默认采用的是线程池隔离模式，feign调用和请求的线程不属于同一个线程，无法获取请求的线程数据,会造成空指针异常。

开启feign的熔断配置，在changgou-user-oauth的application.yml中添加开启熔断配置：

*#开启容断器***feign**:  
 **hystrix**:  
 **enabled**: **true**

拦截器中出现空指针问题：



我们发现这块的ServletRequestAttributes始终为空，RequestContextHolder.getRequestAttributes()该方法是从ThreadLocal变量里面取得相应信息的，当hystrix断路器的隔离策略为THREAD(线程隔离)时，是无法取得ThreadLocal中的值。

### 解决方案

hystrix隔离策略换为SEMAPHORE(信号量隔离)

修改changgou-user-oauth的application.yml配置文件中添加如下代码，代码如下：

*#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **isolation**:  
 **thread**:  
 *# 熔断超时设置，默认为1秒* **timeoutInMilliseconds**: 10000  
 *# 隔离模式使用信号量隔离SEMAPHORE 默认值是线程隔离THREAD* **strategy**: SEMAPHORE

## 拦截器工具类抽取

微服务之间相互认证的情况非常多，我们可以把上面的拦截器抽取出去，放到changgou-common的entity包中创建FeignInterceptor，其他工程需要用，直接创建一个@Bean对象即可。

*/\*\*  
 \* Feign调用拦截器-会在调用所有Feign前执行apply方法的逻辑  
 \** ***@author*** *Steven  
 \** ***@description*** *entity  
 \*/***public class** FeignInterceptor **implements** RequestInterceptor {  
 @Override  
 **public void** apply(RequestTemplate requestTemplate) {  
 *//使用RequestContextHolder工具获取request相关变量* ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  
 **if** (attributes != **null**) {  
 *//取出request* HttpServletRequest request = attributes.getRequest();  
 *//获取所有头文件信息的key* Enumeration<String> headerNames = request.getHeaderNames();  
 **if** (headerNames != **null**) {  
 **while** (headerNames.hasMoreElements()) {  
 *//头文件的key* String name = headerNames.nextElement();  
 *//头文件的value* String values = request.getHeader(name);  
 *//将令牌数据添加到头文件中* requestTemplate.header(name, values);  
 }  
 }  
 }  
 }  
}

**使用方式：**



## 订单对接网关+oauth

### application.yml配置

修改微服务网关changgou-gateway-web的application.yml配置文件，添加order的路由过滤配置，配置如下：

- **id**: changgou\_order\_route *#路由唯一标识-user* **uri**: lb://order *# 发送给注册服务名字为goods的服务器  
 # 路由断言：路由拦截规则* **predicates**:  
 - Path=/api/cart/\*\*,/api/categoryReport/\*\*,/api/orderConfig/\*\*,/api/order/\*\*,/api/orderItem/\*\*,/api/orderLog/\*\*,/api/preferential/\*\*,/api/returnCause/\*\*,/api/returnOrder/\*\*,/api/returnOrderItem/\*\*  
 **filters**:  
 - StripPrefix=1 *# 去除路径前缀过滤器，一个"/"识别为一个路径*

### 过滤地址识别工具

在微服务网关changgou-gateway-web中添加com.changgou.filter.URLFilter过滤类，用于过滤需要用户登录的地址，代码如下：

*/\*\*  
 \* 用户识别不需要登录的Url地址  
 \** ***@author*** *Steven  
 \** ***@description*** *com.changgou.filter  
 \*/***public class** URLFilter {  
 *//不需要登录的url* **private static final** String[] ***ignore*** = {  
 **"/api/user/login"**,  
 **"/api/user/add"** };  
  
 */\*\*  
 \* 识别传入的uri是否要权限校验  
 \** ***@param uri*** *当前传入的uri  
 \** ***@return*** *true需要 | false不需要  
 \*/* **public static boolean** hasAuthorize(String uri){  
 **for** (String ig : ***ignore***) {  
 *//如果配置到不需要校验地址* **if** (uri.startsWith(ig)) {  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
}

### 全局过滤器修改

修改之前的com.changgou.filter.AuthorizeFilter的过滤方法，将是否需要用户登录过滤也加入其中，代码如下：

@Override  
**public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  
 *//1、获取Request、Response对象-exchange.get...* ServerHttpRequest request = exchange.getRequest();  
 ServerHttpResponse response = exchange.getResponse();  
 *//2、获取请求的URI-request.getURI().getPath()* String uri = request.getURI().getPath();  
 *//3、如果是登录请求-uri.startsWith，放行-chain.filter* **if**(!URLFilter.*hasAuthorize*(uri)){  
 **return** chain.filter(exchange);  
 }**else**{ *//4、如果是非登录请求  
 //4.1 获取前端传入的令牌-从请求头中获取-request.getHeaders().getFirst* String token = request.getHeaders().getFirst(***AUTHORIZE\_TOKEN***);  
 *//4.2 如果头信息中没有，从请求参数中获取-request.getQueryParams().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 token = request.getQueryParams().getFirst(***AUTHORIZE\_TOKEN***);  
 }  
 *//4.3 如果请求参数中没有，从cookie中获取-request.getCookies().getFirst* **if**(StringUtils.*isEmpty*(token)){  
 HttpCookie first = request.getCookies().getFirst(***AUTHORIZE\_TOKEN***);  
 **if**(first != **null**){  
 token = first.getValue();  
 }  
 }  
 *//4.4 如果以上方式都取不到令牌-返回405错误-response.setStatusCode(405)-return response.setComplete* **if**(StringUtils.*isEmpty*(token)){  
 *//返回405，错误表示方法不允许访问* response.setStatusCode(HttpStatus.***METHOD\_NOT\_ALLOWED***);  
 **return** response.setComplete();  
 }**else**{ *// 4.5 如果获取到了令牌，解析令牌-JwtUtil.parseJWT，放行-chain.filter(exchange)* **try** {  
 *//Claims claims = JwtUtil.parseJWT(token);  
 //4.5.1解析成功-把令牌返回-request.mutate().header(key,value)* request.mutate().header(***AUTHORIZE\_TOKEN***, **"bearer "** + token);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//无效的认证* response.setStatusCode(HttpStatus.***UNAUTHORIZED***);  
 **return** response.setComplete();  
 }  
 **return** chain.filter(exchange);  
 }  
 }  
}

### 集成OAuth进行安全校验

修改changgou-service-order的pom.xml，添加oauth的依赖

*<!--oauth依赖-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-oauth2</**artifactId**>  
</**dependency**>

将公钥拷贝到changgou-service-order工程的resources中

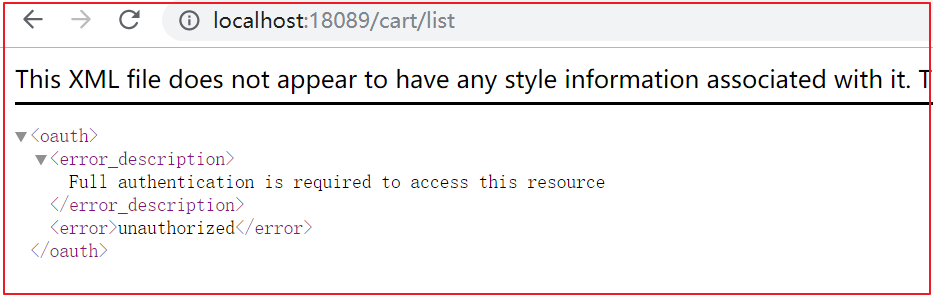


在changgou-service-order工程中创建com.changgou.order.config.ResourceServerConfig,配置需要拦截的路径，这里需要拦截所有请求路径，代码如下：

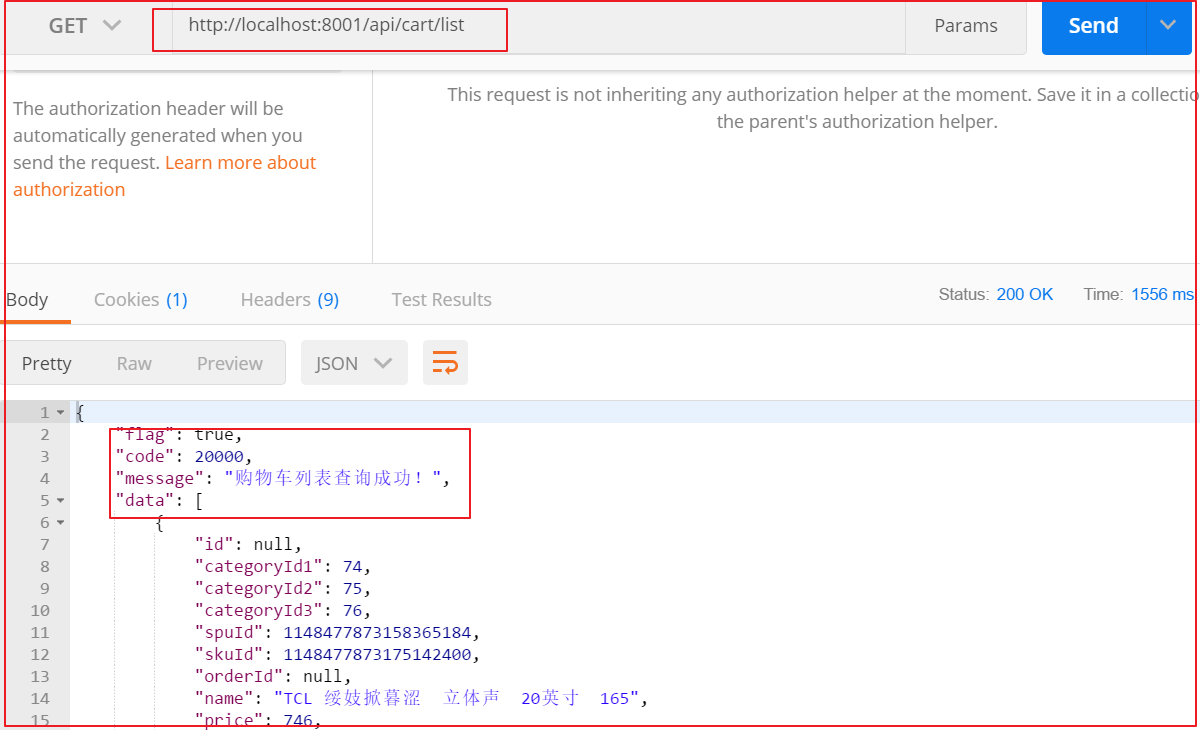
*/\*\*  
 \* 实现信息校验  
 \* 1、获取公钥  
 \* 2、使用公钥作为校验密钥  
 \* 3、校验令牌  
 \*/*@Configuration  
@EnableResourceServer *//开启资源授权控制*@EnableGlobalMethodSecurity(prePostEnabled = **true**, securedEnabled = **true**)*//激活方法上的PreAuthorize注解***public class** ResourceServerConfig **extends** ResourceServerConfigurerAdapter {  
  
 *//公钥* **private static final** String ***PUBLIC\_KEY*** = **"public.key"**;  
  
 */\*\*\*  
 \* 定义JwtTokenStore  
 \** ***@param jwtAccessTokenConverter*** *\** ***@return*** *\*/* @Bean  
 **public** TokenStore tokenStore(JwtAccessTokenConverter jwtAccessTokenConverter) {  
 **return new** JwtTokenStore(jwtAccessTokenConverter);  
 }  
  
 */\*\*\*  
 \* 定义JJwtAccessTokenConverter  
 \** ***@return*** *\*/* @Bean  
 **public** JwtAccessTokenConverter jwtAccessTokenConverter() {  
 JwtAccessTokenConverter converter = **new** JwtAccessTokenConverter();  
 converter.setVerifierKey(getPubKey());  
 **return** converter;  
 }  
 */\*\*  
 \* 获取非对称加密公钥 Key  
 \** ***@return*** *公钥 Key  
 \*/* **private** String getPubKey() {  
 Resource resource = **new** ClassPathResource(***PUBLIC\_KEY***);  
 **try** {  
 InputStreamReader inputStreamReader = **new** InputStreamReader(resource.getInputStream());  
 BufferedReader br = **new** BufferedReader(inputStreamReader);  
 **return** br.lines().collect(Collectors.*joining*(**"\n"**));  
 } **catch** (IOException ioe) {  
 **return null**;  
 }  
 }  
  
 */\*\*\*  
 \* Http安全配置，对每个到达系统的http请求链接进行校验  
 \** ***@param http*** *\** ***@throws*** *Exception  
 \*/* @Override  
 **public void** configure(HttpSecurity http) **throws** Exception {  
 *//所有请求必须认证通过* http.authorizeRequests()  
 */\*//下边的路径放行  
 .antMatchers(  
 //删除"/user/load/\*"的放行策略  
 "/user/add"). //配置地址放行  
 permitAll()\*/* .anyRequest().  
 authenticated(); *//其他地址需要认证授权* }  
}

### 测试

未登录：使用浏览器访问 <http://localhost:8001/api/cart/list> ，效果如下：



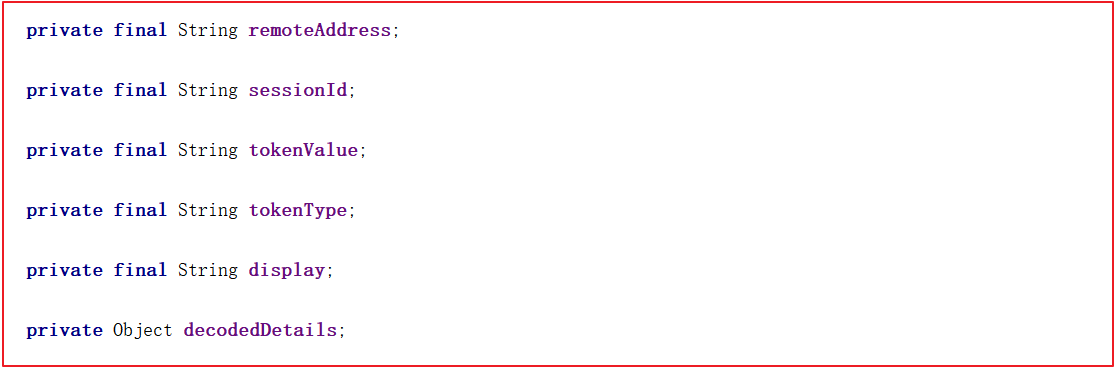
已登录：使用Postman访问 <http://localhost:8001/api/cart/list> ，效果如下：



## 获取用户数据

### 数据分析

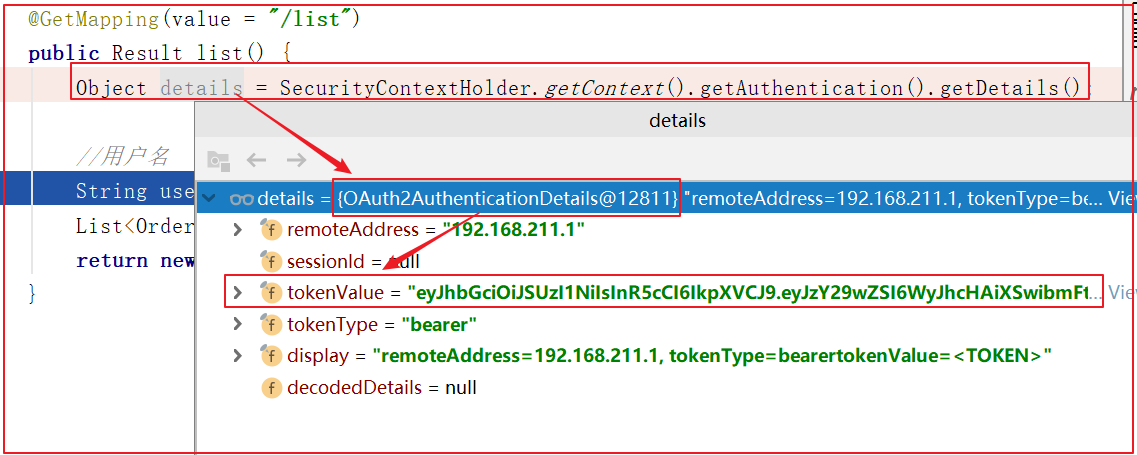
用户登录后，数据会封装到SecurityContextHolder.getContext().getAuthentication()里面，我们可以将数据从这里面取出，然后转换成OAuth2AuthenticationDetails,在这里面可以获取到令牌信息、令牌类型等，代码如下：



这里的tokenValue是加密之后的令牌数据，remoteAddress是用户的IP信息，tokenType是令牌类型。

我们可以获取令牌加密数据后，使用公钥对它进行解密，如果能解密说明说句无误，如果不能解密用户也没法执行到这一步。解密后可以从明文中获取用户信息。

我们可以通过调试，查询SpringSecurity中包装的数据，在CartController的list方法中添加测试代：



### 代码实现

#### 创建校验读取令牌信息工具类

在后续的项目案例中我们经常需要用到获取令牌校验令牌读取用户信息的代码，所以我们在changgou\_common工程中抽取entity.TokenDecode工具类。

**首先在changgou-common工程中引入鉴权包**

*<!--oauth依赖-->*<**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-oauth2</**artifactId**>  
 <**scope**>provided</**scope**>  
</**dependency**>  
*<!--鉴权-->*<**dependency**>  
 <**groupId**>io.jsonwebtoken</**groupId**>  
 <**artifactId**>jjwt</**artifactId**>  
 <**version**>0.9.0</**version**>  
 <**scope**>provided</**scope**>  
</**dependency**>

**工具代码如下，我们可以直接复制，这里的代码与我们每个微服务整合oauth后添加的ResourceServerConfig类似：**

*/\*\*  
 \* 校验读取令牌信息工具类  
 \** ***@author*** *Steven  
 \** ***@description*** *entity  
 \*/***public class** TokenDecode {  
 *//公钥* **private static final** String ***PUBLIC\_KEY*** = **"public.key"**;  
 **private static** String *publickey* = **""**;  
  
 */\*\*  
 \* 获取非对称加密公钥 Key  
 \** ***@return*** *公钥 Key  
 \*/* **public static** String getPubKey() {  
 **if** (StringUtils.*isNotEmpty*(*publickey*)) {  
 **return** *publickey*;  
 }  
 Resource resource = **new** ClassPathResource(***PUBLIC\_KEY***);  
 **try** {  
 InputStreamReader inputStreamReader = **new** InputStreamReader(resource.getInputStream());  
 BufferedReader br = **new** BufferedReader(inputStreamReader);  
 *publickey* = br.lines().collect(Collectors.*joining*(**"\n"**));  
 **return** *publickey*;  
 } **catch** (IOException ioe) {  
 **return null**;  
 }  
 }  
  
 */\*\*\*  
 \* 读取令牌数据  
 \*/* **public static** Map<String,String> dcodeToken(String token){  
 *//校验Jwt* Jwt jwt = JwtHelper.*decodeAndVerify*(token, **new** RsaVerifier(*getPubKey*()));  
 *//获取Jwt原始内容* String claims = jwt.getClaims();  
 **return** JSON.*parseObject*(claims,Map.**class**);  
 }  
 */\*\*\*  
 \* 获取用户信息  
 \** ***@return*** *\*/* **public static** Map<String,String> getUserInfo(){  
 *//获取授权信息* OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails) SecurityContextHolder.*getContext*().getAuthentication().getDetails();  
 *//令牌解码* **return** *dcodeToken*(details.getTokenValue());  
 }  
}

#### 控制层获取用户数据

在CartController中使用TokenDecode的getUserInfo方法获取用户信息，代码如下：

@RestController  
@CrossOrigin  
@RequestMapping(value = **"/cart"**)  
**public class** CartController {  
 @Autowired  
 **private** CartService **cartService**;  
 */\*\*\*  
 \* 加入购物车  
 \** ***@param num****:购买的数量  
 \** ***@param id****：购买的商品(SKU)ID  
 \** ***@return*** *\*/* @RequestMapping(value = **"/add"**)  
 **public** Result add(Integer num, Long id){  
 *//用户名  
 //String username="szitheima";* String username=TokenDecode.*getUserInfo*().get(**"username"**);  
 *//将商品加入购物车* **cartService**.add(num,id,username);  
 **return new** Result(**true**, StatusCode.***OK***,**"加入购物车成功！"**);  
 }  
 */\*\*\*  
 \* 查询用户购物车列表**\*/* @GetMapping(value = **"/list"**)  
 **public** Result list() {  
 *//用户名  
 //String username="szitheima";* Map<String, String> userInfo = TokenDecode.*getUserInfo*();  
 String username=userInfo.get(**"username"**);  
 List<OrderItem> orderItems = **cartService**.list(username);  
 **return new** Result(**true**, StatusCode.***OK***, **"购物车列表查询成功！"**, orderItems);  
 }  
}