

第10章 权限配置和购物车

学习目标

- 资源服务器授权配置
- 掌握OAuth2.0认证微服务动态加载数据

- 1 授权码模式数据加载
- 2 密码模式数据加载

- 掌握购物车流程

- 1 1. 用户未登录购物车(京东)
- 2 2. 用户已登录购物车(天猫/京东)

- OAuth2.0认证并获取用户令牌数据

- 1 解析令牌，获取用户信息

- 微服务与微服务之间的认证

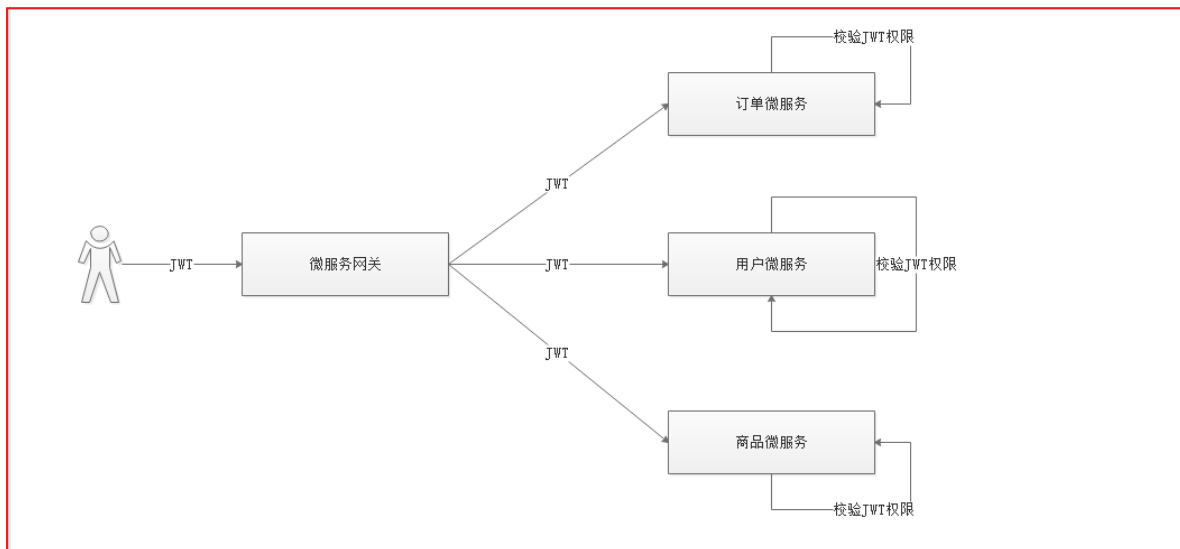
- 1 多个微服务之间令牌携带访问问题

1 鉴权分析

微服务网中鉴权类型有很多，我们可以根据不同的要求选择，我们项目中目前选择了客户端Token解决方案。

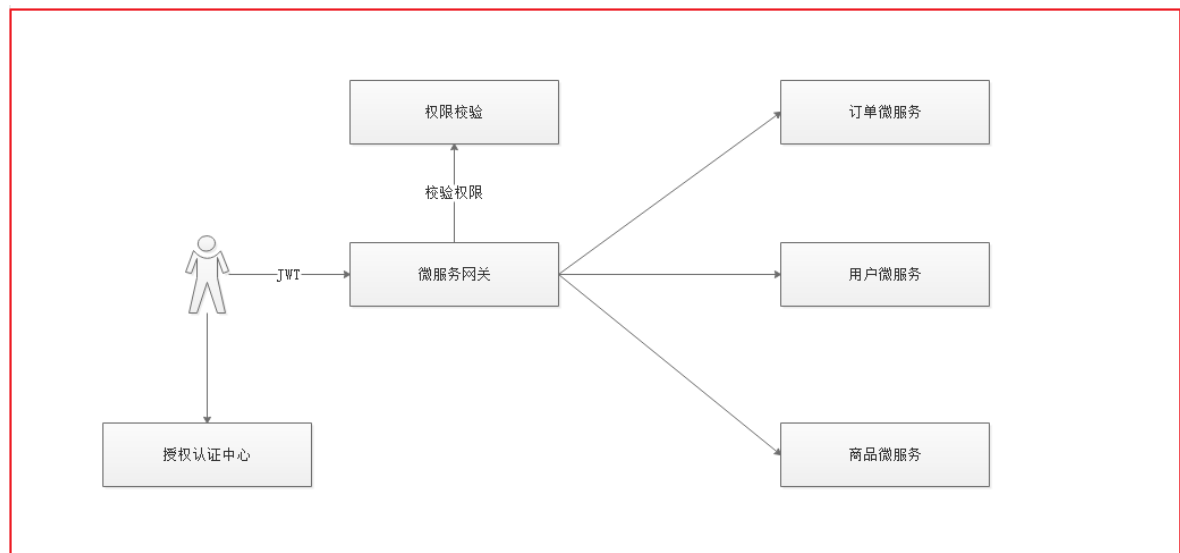
1.1 客户端Token解决方案

项目中我们采用这种方式，这种方式令牌在客户端存储，令牌必须包含足够的信息，以便可以在所有微服务中建立用户身份。令牌会附加到每个请求上，为微服务提供用户身份验证，这种解决方案的安全性相对较好，而且它足够简单且库支持程度也比较好，但校验权限频次较高。



1.2 客户端Token与API网关结合(学习)

这个方案意味着所有请求都通过网关，从而有效地隐藏了微服务,所有的权限校验直接在微服务网关这里完成，这种方式效率更高，这种方案权限认证通常要考虑2个方面权限，分别为功能权限和数据权限。



这里列出一些表结构供大家学习。

系统地址url规则表：

```

1 CREATE TABLE `url_rule` (
2   `id` varchar(60) NOT NULL,
3   `url` varchar(300) DEFAULT NULL COMMENT '跳转地址,数字占位符: [number], 字符占位符: [str]',
4   `name` varchar(30) DEFAULT NULL COMMENT '权限名字',
5   `is_menu` int(1) DEFAULT NULL COMMENT '是否是菜单: 0不是菜单权限, 1: 菜单权限',
6   `pid` varchar(60) DEFAULT NULL COMMENT '父ID, 0: 无父节点',
7   `method` int(1) DEFAULT NULL COMMENT '提交方式, 0: 所有提交方式, 1: GET, 2: POST, 3: PUT, 4: DELETE',
8   `sort` int(4) DEFAULT NULL COMMENT '排序',
9   PRIMARY KEY (`id`)
10 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
  
```

角色表：

```

1 CREATE TABLE `roles` (
2   `id` int(5) NOT NULL,
3   `role_name` varchar(30) DEFAULT NULL,
4   `role_desc` varchar(2000) DEFAULT NULL COMMENT '角色描述',
5   PRIMARY KEY (`id`)
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

角色路径关联表:

```

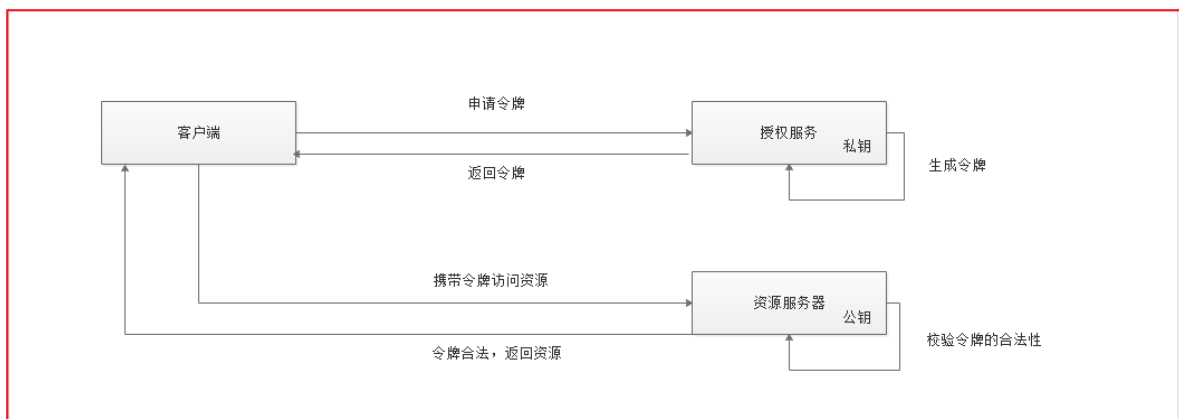
1 CREATE TABLE `url_role_list` (
2   `url_id` varchar(60) NOT NULL COMMENT 'urlID',
3   `role_id` int(5) NOT NULL COMMENT '角色ID',
4   `handler_safe` int(1) DEFAULT '1' COMMENT '安全限制: 1, 不限制, 2: 身份绑定',
5   PRIMARY KEY (`url_id`)
6 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

权限: 2个地方要注意 1)功能权限 2)数据权限

2 资源服务器授权配置

2.1 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。

(1)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

(2)添加依赖

```

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

```

(3)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，如下：

```

1  @Configuration
2  @EnableResourceServer
3  @EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)//
   激活方法上的PreAuthorize注解
4  public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
5
6      //公钥
7      private static final String PUBLIC_KEY = "public.key";
8
9      /**
10       * 定义JwtTokenStore
11       * @param jwtAccessTokenConverter
12       * @return
13       */
14     @Bean
15     public TokenStore tokenStore(JwtAccessTokenConverter
16     jwtAccessTokenConverter) {
17         return new JwtTokenStore(jwtAccessTokenConverter);
18     }
19
20     /**
21      * 定义JJwtAccessTokenConverter
22      * @return
23      */
24     @Bean
25     public JwtAccessTokenConverter
26     () {
27         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
28         converter.setVerifierKey(getPubKey());
29         return converter;
30     }
31
32     /**
33      * 获取非对称加密公钥 Key
34      * @return 公钥 Key
35      */
36     private String getPubKey() {
37         Resource resource = new ClassPathResource(PUBLIC_KEY);
38         try {
39             InputStreamReader inputStreamReader = new
40             InputStreamReader(resource.getInputStream());
41             BufferedReader br = new BufferedReader(inputStreamReader);
42             return br.lines().collect(Collectors.joining("\n"));
43         } catch (IOException ioe) {
44             return null;
45         }
46     }
47
48     /**
49      * Http安全配置，对每个到达系统的http请求链接进行校验
50      * @param http
51      * @throws Exception
52      */
53     @Override
54     public void configure(HttpSecurity http) throws Exception {
55         //所有请求必须认证通过
56         http.authorizeRequests()
57             //下边的路径放行
58             .antMatchers(

```

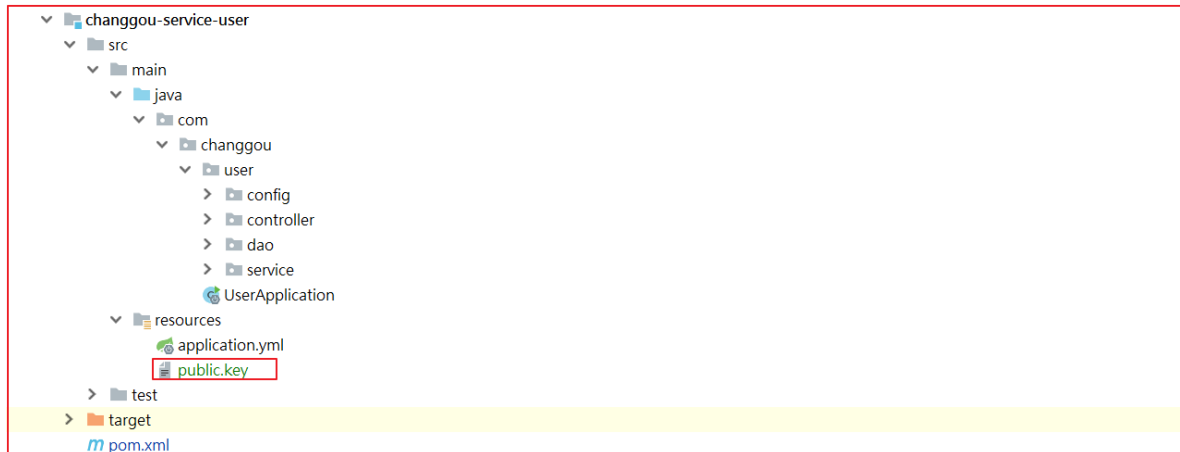
```

56         "/user/add"). //配置地址放行
57         permitAll()
58         .anyRequest().
59         authenticated(); //其他地址需要认证授权
60     }
61 }

```

2.2 用户微服务资源授权

将上面生成的公钥public.key拷贝到 changgou-service-user 微服务工程的resources目录下，如下图：



(1)引入依赖

在changgou-service-user微服务工程pom.xml中引入oauth依赖

```

1  <!--oauth依赖-->
2  <dependency>
3      <groupId>org.springframework.cloud</groupId>
4      <artifactId>spring-cloud-starter-oauth2</artifactId>
5  </dependency>

```

(2)资源授权配置

在changgou-service-user工程中创建com.changgou.user.config.ResourceServerConfig，代码如下：

```

1  @Configuration
2  @EnableResourceServer
3  @EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)//
    激活方法上的PreAuthorize注解
4  public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
5
6      //公钥
7      private static final String PUBLIC_KEY = "public.key";
8
9      /**
10         * 定义JwtTokenStore

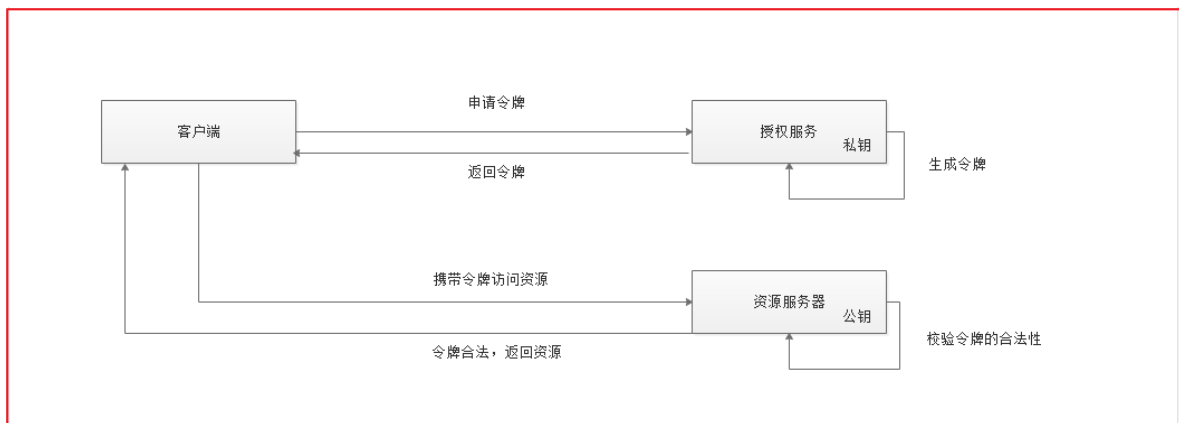
```

```

11     * @param jwtAccessTokenConverter
12     * @return
13     */
14     @Bean
15     public TokenStore tokenStore(JwtAccessTokenConverter
jwtAccessTokenConverter) {
16         return new JwtTokenStore(jwtAccessTokenConverter);
17     }
18
19     /**
20     * 定义JJwtAccessTokenConverter
21     * @return
22     */
23     @Bean
24     public JwtAccessTokenConverter jwtAccessTokenConverter() {
25         JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
26         converter.setVerifierKey(getPubKey());
27         return converter;
28     }
29     /**
30     * 获取非对称加密公钥 key
31     * @return 公钥 Key
32     */
33     private String getPubKey() {
34         Resource resource = new ClassPathResource(PUBLIC_KEY);
35         try {
36             InputStreamReader inputStreamReader = new
InputStreamReader(resource.getInputStream());
37             BufferedReader br = new BufferedReader(inputStreamReader);
38             return br.lines().collect(Collectors.joining("\n"));
39         } catch (IOException ioe) {
40             return null;
41         }
42     }
43
44     /**
45     * Http安全配置，对每个到达系统的http请求链接进行校验
46     * @param http
47     * @throws Exception
48     */
49     @Override
50     public void configure(HttpSecurity http) throws Exception {
51         //所有请求必须认证通过
52         http.authorizeRequests()
53             //下边的路径放行
54             .antMatchers(
55                 "/user/add"). //配置地址放行
56             permitAll()
57             .anyRequest().
58             authenticated(); //其他地址需要认证授权
59     }
60 }

```

3.3 授权测试



用户每次访问微服务的时候，需要先申请令牌，令牌申请后，每次将令牌放到头文件中，才能访问微服务。

头文件中每次需要添加一个 `Authorization` 头信息，头的结果为 `bearer token`。

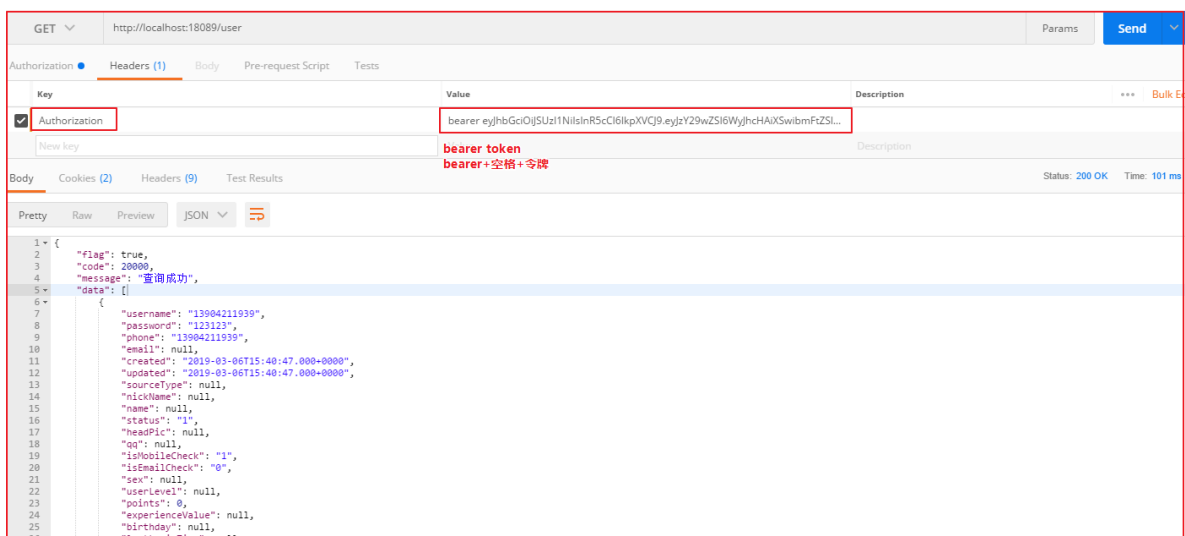
(1)不携带令牌测试

访问<http://localhost:18088/user> 不携带令牌，结果如下：



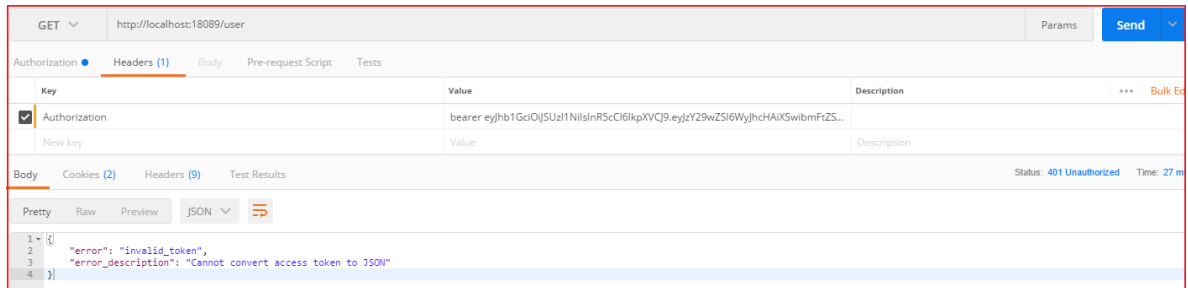
(2)携带正确令牌访问

访问<http://localhost:18088/user> 携带正确令牌，结果如下：

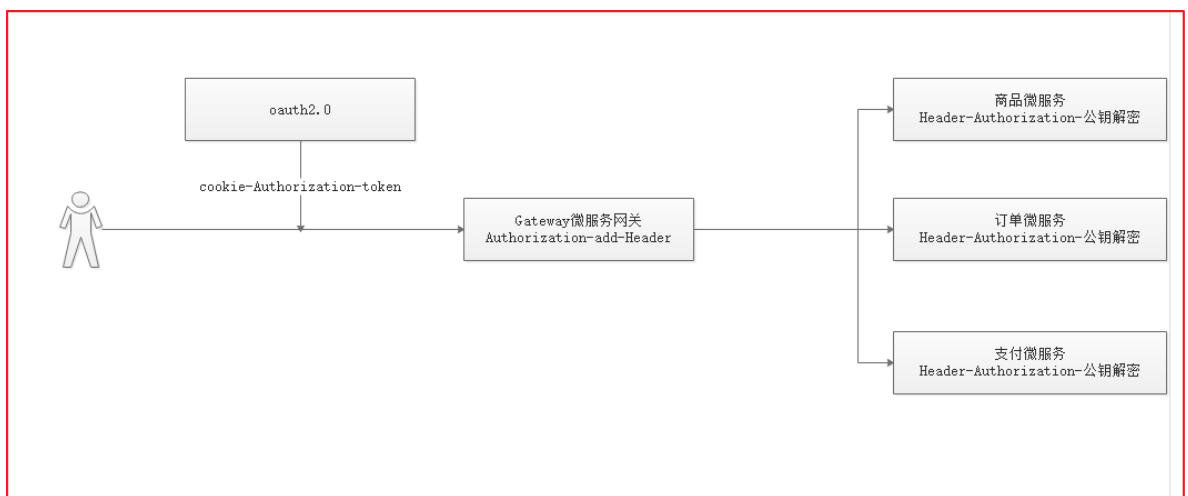


(3)携带错误令牌

访问<http://localhost:18088/user> 携带不正确令牌，结果如下：



3 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头文件中
2. 用户携带头文件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(头文件)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

3.1 令牌加入到Header中

访问 <http://localhost:8001/api/user>，将生成的新令牌放到头文件中，在令牌前面添加 Bearer，这里主要有个空格，效果如下：


```
@Configuration
@EnableResourceServer
//开启方法上的PreAuthorize注解
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
```

(2)方法权限控制

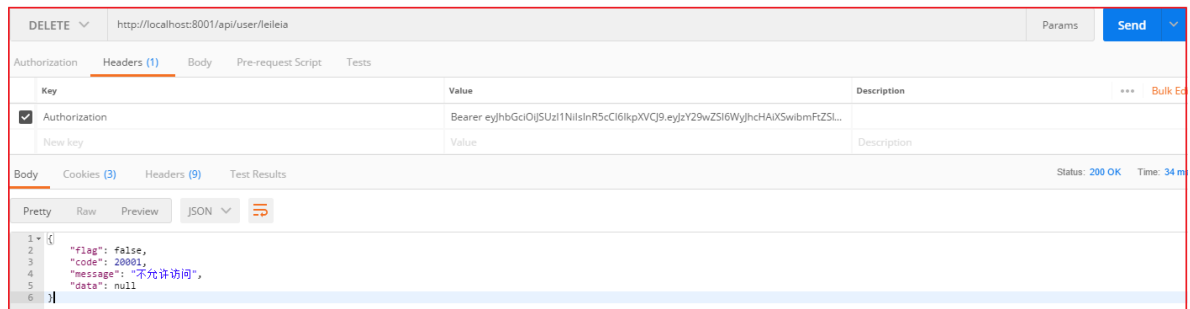
在 changgoug-service-user 微服务的 `com.changgou.user.controller.UserController` 类的 `delete()`方法上添加权限控制注解 `@PreAuthorize`，代码如下：

```
@PreAuthorize("hasAnyAuthority('admin')") 表示只有admin角色才能访问该方法，其他角色无权访问
@DeleteMapping(value =("/{id}") )
public Result delete(@PathVariable String id){
    userService.delete(id);
    return new Result( flag: true, StatusCode.OK, message: "删除成功");
}
```

(3)测试

我们使用Postman测试，先创建令牌，然后将令牌数存放到头文件中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/1e1e1e1e` 提交方式：DELETE

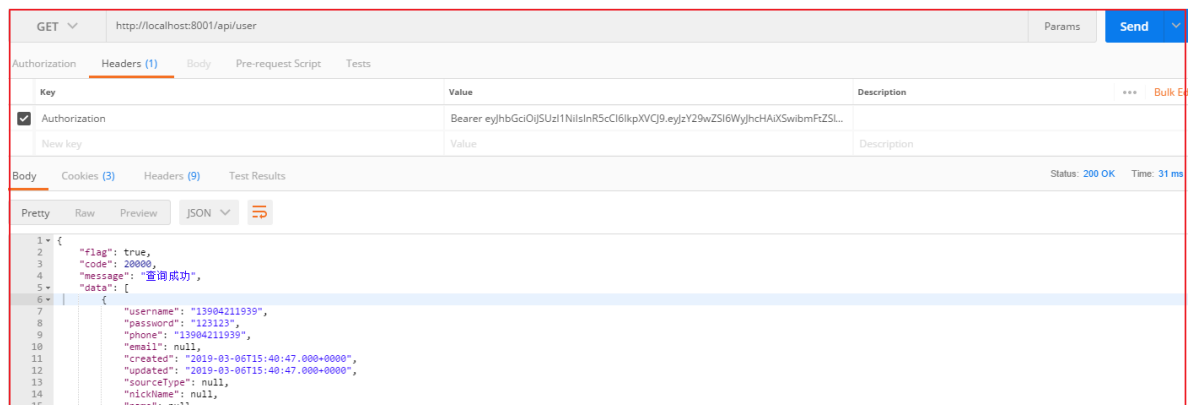


发现上面无法访问，因为用户登录的时候，角色不包含admin角色，而delete方法需要admin角色，所以被拦截了。

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：



知识点说明:

如果希望一个方法能被多个角色访问, 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')")`

4 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

4.1 客户端数据加载

4.1.1 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:

```
/**
 * 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") //密钥
        .redirectUri("http://localhost") //重定向地址
        .accessTokenValiditySeconds(3600) //访问令牌有效期
        .refreshTokenValiditySeconds(3600) //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
}
```

(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:

```
1 CREATE TABLE `oauth_client_details` (
2   `client_id` varchar(48) NOT NULL COMMENT '客户端ID, 主要用于标识对应的应用',
3   `resource_ids` varchar(256) DEFAULT NULL,
4   `client_secret` varchar(256) DEFAULT NULL COMMENT '客户端密钥,
   BCryptPasswordEncoder加密算法加密',
```

```

5   `scope` varchar(256) DEFAULT NULL COMMENT '对应的范围',
6   `authorized_grant_types` varchar(256) DEFAULT NULL COMMENT '认证模式',
7   `web_server_redirect_uri` varchar(256) DEFAULT NULL COMMENT '认证后重定向地址',
8   `authorities` varchar(256) DEFAULT NULL,
9   `access_token_validity` int(11) DEFAULT NULL COMMENT '令牌有效期',
10  `refresh_token_validity` int(11) DEFAULT NULL COMMENT '令牌刷新周期',
11  `additional_information` varchar(4096) DEFAULT NULL,
12  `autoapprove` varchar(256) DEFAULT NULL,
13  PRIMARY KEY (`client_id`)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

字段说明:

```

1  client_id: 客户端id
2  resource_ids: 资源id (暂时不用)
3  client_secret: 客户端密钥
4  scope: 范围
5  access_token_validity: 访问token的有效期 (秒)
6  refresh_token_validity: 刷新token的有效期 (秒)
7  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

```

1  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);
2  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:

```

private static final String CLIENT_FIELDS_FOR_UPDATE = "resource_ids, scope, "
    + "authorized_grant_types, web_server_redirect_uri, authorities, access_token_validity, "
    + "refresh_token_validity, additional_information, autoapprove";

private static final String CLIENT_FIELDS = "client_secret, " + CLIENT_FIELDS_FOR_UPDATE;

private static final String BASE_FIND_STATEMENT = "select client_id, " + CLIENT_FIELDS
    + " from oauth_client_details";

private static final String DEFAULT_FIND_STATEMENT = BASE_FIND_STATEMENT + " order by client_id";

private static final String DEFAULT_SELECT_STATEMENT = BASE_FIND_STATEMENT + " where client_id = ?"; // 查询操作

private static final String DEFAULT_INSERT_STATEMENT = "insert into oauth_client_details (" + CLIENT_FIELDS // 添加操作
    + ", client_id) values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

private static final String DEFAULT_UPDATE_STATEMENT = "update oauth_client_details " + "set " // 修改操作
    + CLIENT_FIELDS_FOR_UPDATE.replaceAll(regex: ",", replacement: "=? , ") + "=? where client_id = ?";

private static final String DEFAULT_UPDATE_SECRET_STATEMENT = "update oauth_client_details "
    + "set client_secret = ? where client_id = ?";

```

4.1.2 加载数据改造

(1)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.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
```

上图代码如下：

```
1  datasource:
2      driver-class-name: com.mysql.jdbc.Driver
3      url: jdbc:mysql://192.168.211.132:3306/changgou_oauth?
4          useUnicode=true&characterEncoding=utf-
5          8&useSSL=false&allowMultiQueries=true&serverTimezone=UTC
6      username: root
7      password: 123456
```

(2)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：

修改前：

```
/**
 * 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") //密钥
        .redirectUri("http://localhost") //重定向地址
        .accessTokenValiditySeconds(3600) //访问令牌有效期
        .refreshTokenValiditySeconds(3600) //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
}
```

修改后：

```
/**
 * 客户端信息配置
 * @param clients
 * @throws Exception
 */
@Override
public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
    clients.jdbc(dataSource).clients(clientDetails());
}
```

(3) UserDetailsServiceImpl 修改

将之前的加密方式去掉即可，代码如下：

修改前：

```
/**
 * 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){
            //密钥
            String clientSecret = clientDetails.getClientSecret();
            //静态方式
            //改前
            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(rawPassword: "szitheima");
    //创建User对象
    String permissions = "salesman,accountant,user"; //指定角色
    UserJwt userDetails = new UserJwt(username, pwd, AuthorityUtils.commaSeparatedStringToAuthorityList(permissions));
    return userDetails;
}
```

修改后：

```
if(authentication==null){
    ClientDetails clientDetails = clientDetailsService.loadClientByClientId(username);
    if(clientDetails!=null){
        //密钥
        String clientSecret = clientDetails.getClientSecret();
        //静态方式
        return new User(username, clientSecret, AuthorityUtils.commaSeparatedStringToAuthorityList(""));
        //数据库查找方式
        //改后
        //return new User(username, clientSecret, AuthorityUtils.commaSeparatedStringToAuthorityList(""));
    }
}
```

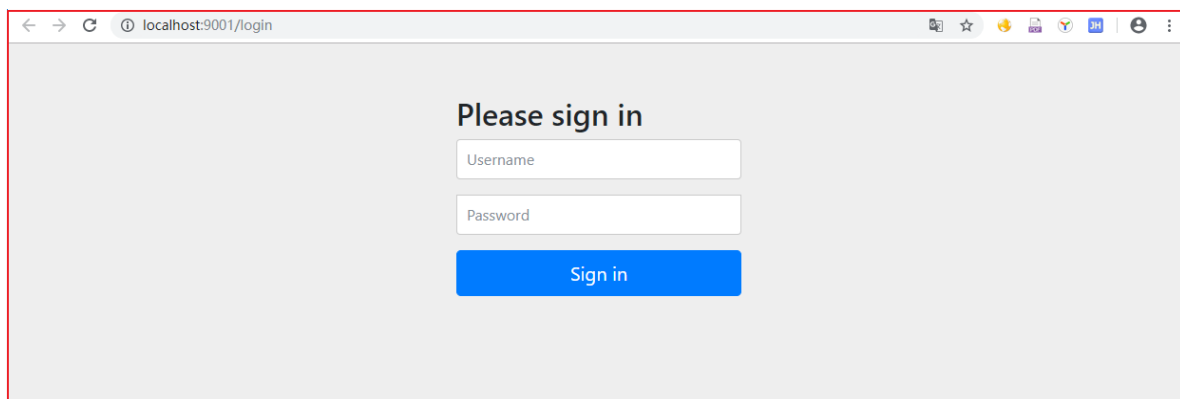
(4) 测试

授权码模式测试

访问：`http://localhost:9001/oauth/authorize?`

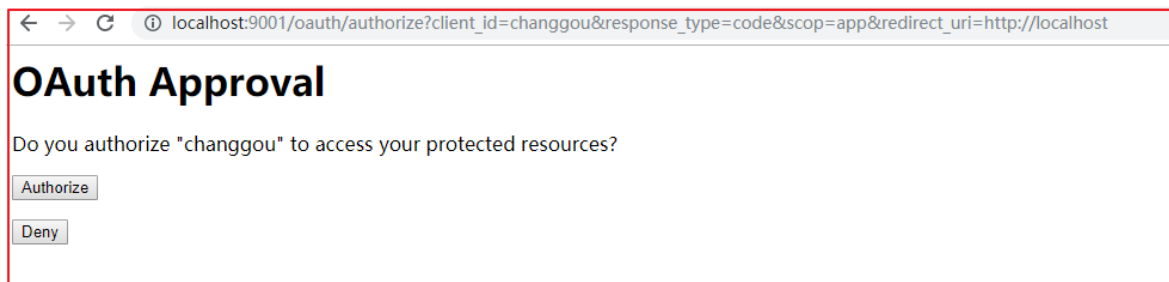
`client_id=szitheima&response_type=code&scope=app&redirect_uri=http://localhost`

效果如下：



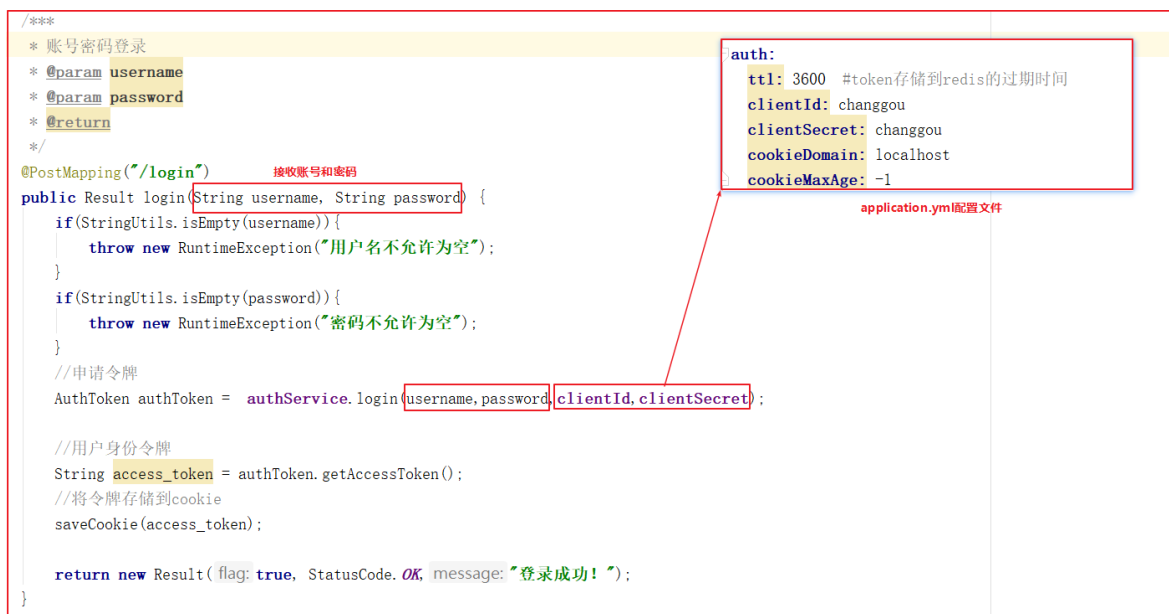
The screenshot shows a web browser window with the address bar displaying 'localhost:9001/login'. The page content is centered and features the text 'Please sign in' in a bold font. Below this text are two input fields: the first is labeled 'Username' and the second is labeled 'Password'. At the bottom of the form is a blue button with the text 'Sign in' in white.

用户名对应应用id，密码对应密钥。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和指定的密钥，所以此时的客户端ID和密钥固定了，输入的账号密码不再是客户端ID和密钥了。



用户加载

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

```

@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){
            //密钥
            String clientSecret = clientDetails.getClientSecret();
            //静态方式
            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( rawPassword: "szitheima");
    //创建User对象
    String permissions = "salesman,accountant,user"; //指定角色
    UserJwt userDetails = new UserJwt(username,pwd, AuthorityUtils.commaSeparatedStringToAuthorityList(permissions));
    return userDetails;
}

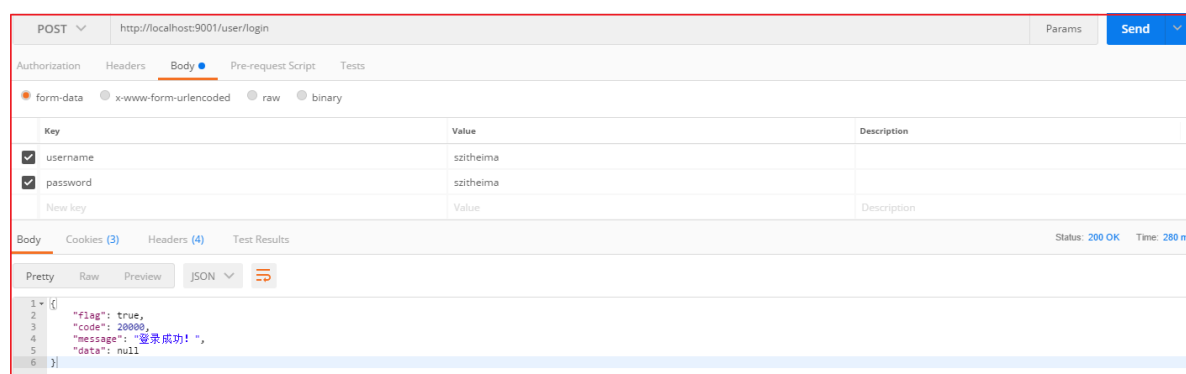
```

这里密码固定写死了
任意账号, 只要密码是szitheima, 则密码正确

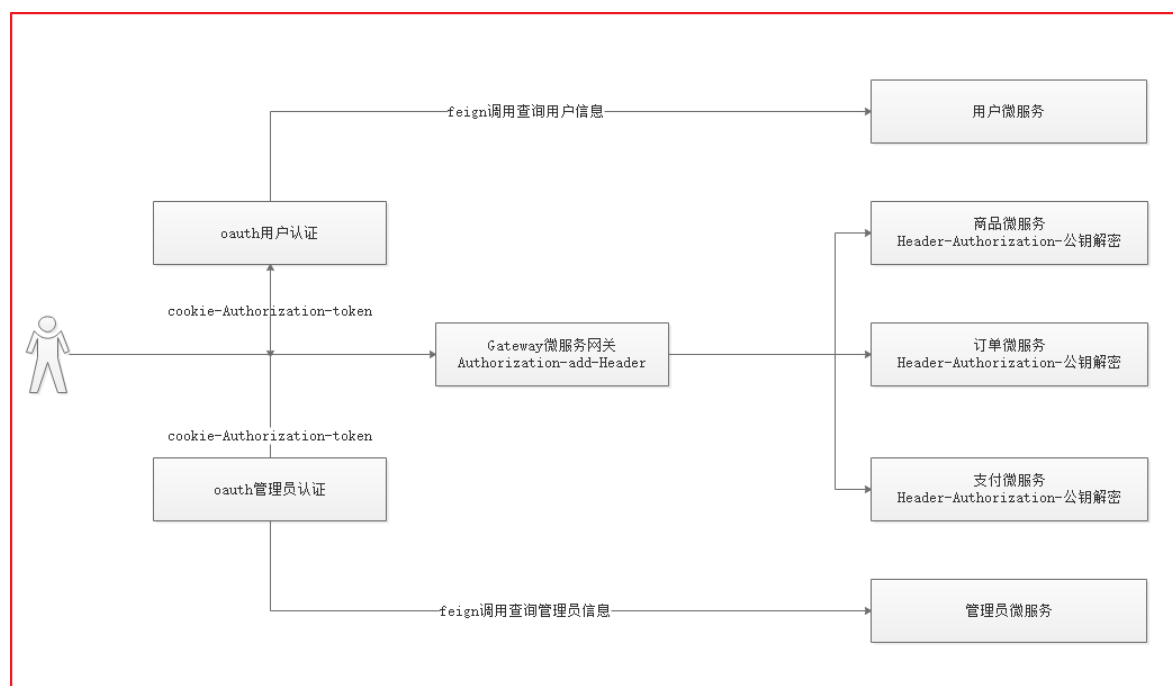
用户每次输入账号和密码, 只要密码是szitheima, 即可登录成功。

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

输入账号密码均为szitheima,效果如下:



4.2 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。

(1)Feign创建

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

```
1  @FeignClient(name="user")
2  @RequestMapping("/user")
3  public interface UserFeign {
4
5      /**
6       * 根据ID查询用户信息
7       * @param id
8       * @return
9       */
10     @GetMapping("/load/{id}")
11     Result<User> findById(@PathVariable String id);
12 }
```

(2)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) {
    //根据ID查询
    User user = userService.findById(id);
    return new Result<User> (flag: true, StatusCode: OK, message: "查询成功", user);
}
```

(3)放行查询用户方法

因为oauth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(
            ...antPatterns: "/user/add", "/user/load/*"). //配置地址放行
        .permitAll()
        .anyRequest().
        authenticated(); //其他地址需要认证授权
}
```

(4)oauth调用查询用户信息

oauth引入对user-api的依赖

```
1 <!-- 依赖用户api -->
2 <dependency>
3     <groupId>com.changgou</groupId>
4     <artifactId>changgou-service-user-api</artifactId>
5     <version>1.0-SNAPSHOT</version>
6 </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){
            //密钥
            String clientSecret = clientDetails.getClientSecret();
            //数据库加载方式
            return new User(username, clientSecret, AuthorityUtils.commaSeparatedStringToAuthorityList("authorityString: ""));
        }
    }

    if (StringUtils.isEmpty(username)) {
        return null;
    }

    //查询数据库
    Result<com.changgou.user.pojo.User> user = userFeign.findById(username);

    //根据用户名查询用户信息
    //String pwd = new BCryptPasswordEncoder().encode("szitheima");
    //创建User对象
    String permissions = "salesman,accountant,user"; //指定角色
    UserJwt userDetails = new UserJwt(username, user.getData().getPassword(), AuthorityUtils.commaSeparatedStringToAuthorityList(permissions));
    return userDetails;
}
```

(5)feign开启

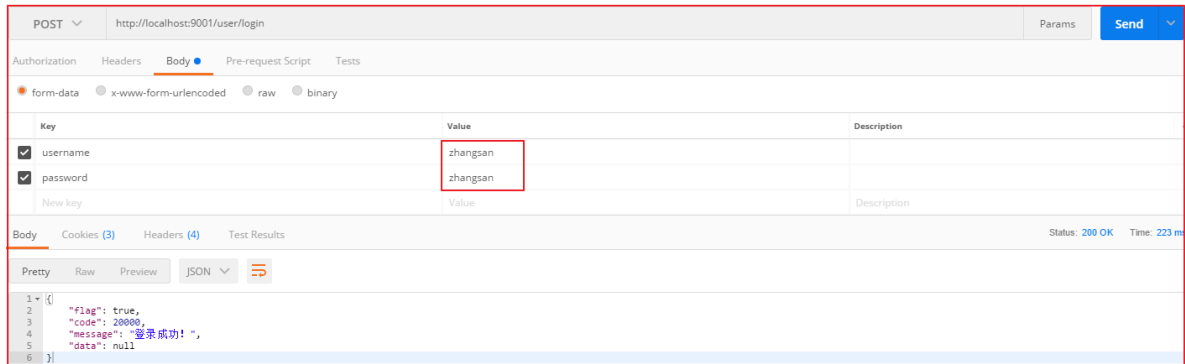
修改 `com.changgou.OAuthApplication` 开启Feign客户端功能

```
1 @SpringBootApplication
2 @EnabledDiscoveryClient
3 @EnableFeignClients(basePackages = {"com.changgou.user.feign"})
4 @MapperScan(basePackages = "com.changgou.auth.dao")
5 public class OAuthApplication {
6
7     public static void main(String[] args) {
8         SpringApplication.run(OAuthApplication.class, args);
9     }
10
11
12     @Bean(name = "restTemplate")
```

```
13     public RestTemplate restTemplate() {
14         return new RestTemplate();
15     }
16 }
```

(6)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：



5 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在Cookie或者WebSQL或者SessionStorage中，用户登录后购物车数据可以存储在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【方便】

- 1 不登录,购物车数据存入到客户端(Cookie|webSQL[H5])
- 2 已登录,可以存储到服务端(MySQL、Redis、MongoDB)
- 3 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【数据更安全】

- 1 已登录,可以存储到服务端(MySQL、Redis、MongoDB)




5.1 购物车分析

B2C->商家只有自己

(1)需求分析

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：

全部商品

| <input type="checkbox"/> 全部 | 商品 | 单价 (元) | 数量 | 小计 (元) | 操作 |
|-----------------------------|---|--------|---|---------|-------------------------------|
| <input type="checkbox"/> | <div></div> <div>米家 (MIJIA) 小米小白智能摄像机增强版 1080p高清360度全景拍摄AI增强</div> | 399.00 | <div><div>-</div><div>1</div><div>+</div></div> | 8848.00 | <div>删除</div> <div>移到收藏</div> |
| <input type="checkbox"/> | <div></div> <div>米家 (MIJIA) 小米小白智能摄像机增强版 1080p高清360度全景拍摄AI增强</div> | 399.00 | <div><div>-</div><div>1</div><div>+</div></div> | 8848.00 | <div>删除</div> <div>移到收藏</div> |
| <input type="checkbox"/> | <div></div> <div>米家 (MIJIA) 小米小白智能摄像机增强版 1080p高清360度全景拍摄AI增强</div> | 399.00 | <div><div>-</div><div>1</div><div>+</div></div> | 8848.00 | <div>删除</div> <div>移到收藏</div> |

☐ 全选

删除选中的商品

移到我的关注

清除下柜商品

已选择 0件商品

总价 (不含运费) : **¥16283.00**

已节省: -¥20.00

结算

猜你喜欢

特惠换购

(2)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。

(3)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：

```
1 CREATE TABLE `tb_order_item` (
2   `id` varchar(20) COLLATE utf8_bin NOT NULL COMMENT 'ID',
3   `category_id1` int(11) DEFAULT NULL COMMENT '1级分类',
4   `category_id2` int(11) DEFAULT NULL COMMENT '2级分类',
5   `category_id3` int(11) DEFAULT NULL COMMENT '3级分类',
6   `spu_id` varchar(20) COLLATE utf8_bin DEFAULT NULL COMMENT 'SPU_ID',
7   `sku_id` bigint(20) NOT NULL COMMENT 'SKU_ID',
8   `order_id` bigint(20) NOT NULL COMMENT '订单ID',
9   `name` varchar(200) COLLATE utf8_bin DEFAULT NULL COMMENT '商品名称',
10  `price` int(20) DEFAULT NULL COMMENT '单价',
11  `num` int(10) DEFAULT NULL COMMENT '数量',
12  `money` int(20) DEFAULT NULL COMMENT '总金额',
13  `pay_money` int(11) DEFAULT NULL COMMENT '实付金额',
14  `image` varchar(200) COLLATE utf8_bin DEFAULT NULL COMMENT '图片地址',
15  `weight` int(11) DEFAULT NULL COMMENT '重量',
```

```

16 `post_fee` int(11) DEFAULT NULL COMMENT '运费',
17 `is_return` char(1) COLLATE utf8_bin DEFAULT NULL COMMENT '是否退货',
18 PRIMARY KEY (`id`),
19 KEY `item_id` (`sku_id`),
20 KEY `order_id` (`order_id`)
21 ) 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。

购物车存储

```

1 1、存储的位置Redis
2 2、存储的结构 key = username
3 value = List<OrderItem>

```

查询

```

1 1、获取用户的登录名 username
2 2、根据登录名username去Redis中查询

```

5.2 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。

(1)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：

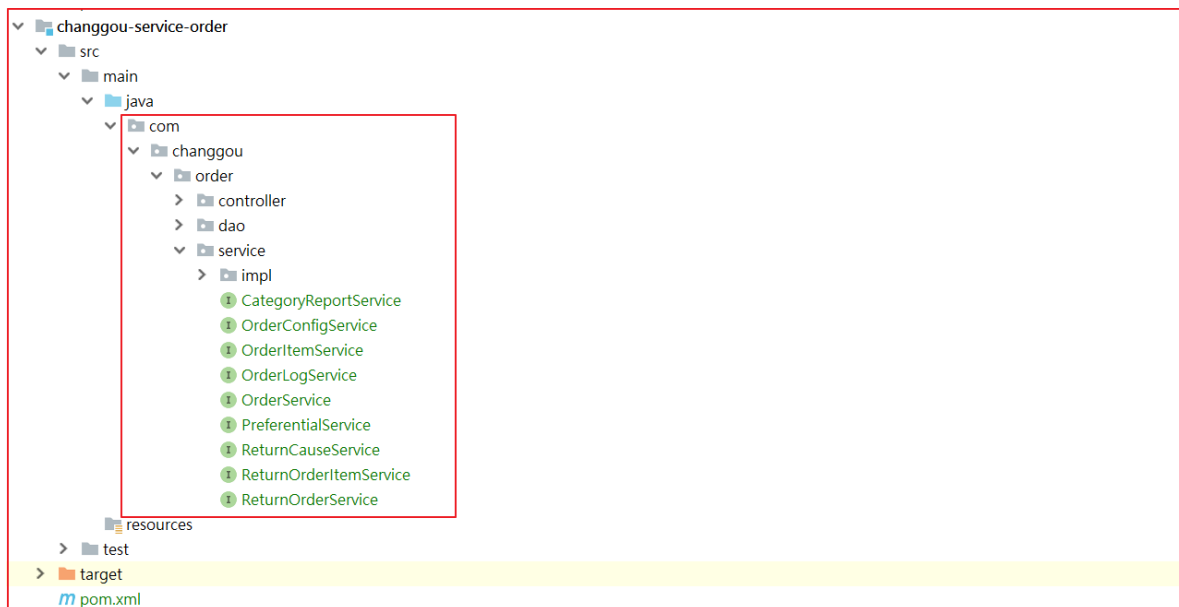
依赖引入：

```

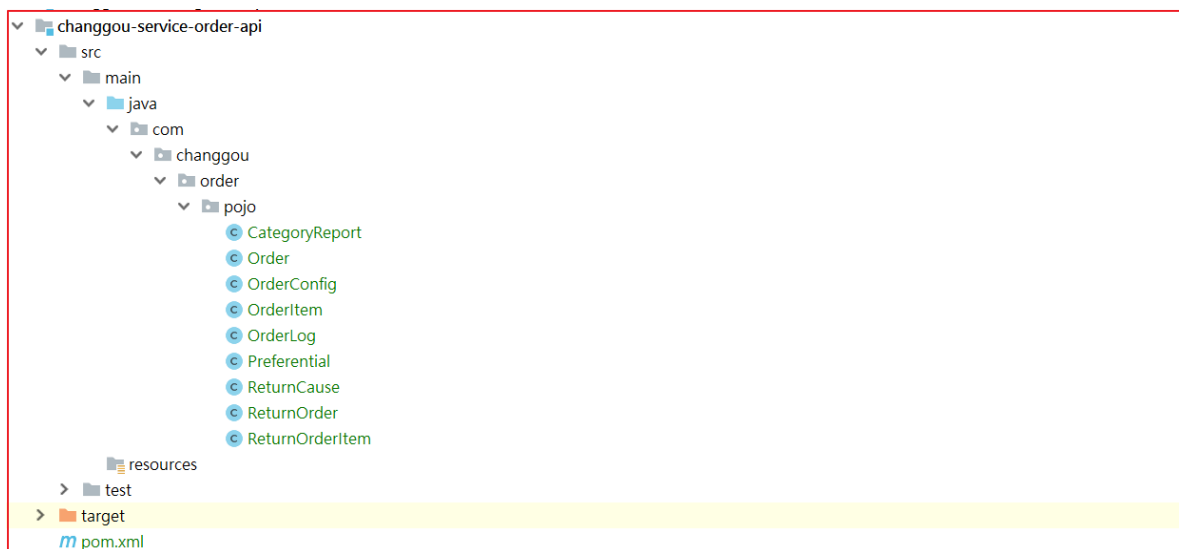
1 <dependency>
2   <groupId>com.changgou</groupId>
3   <artifactId>changgou-service-order-api</artifactId>
4   <version>1.0-SNAPSHOT</version>
5 </dependency>

```

changgou-service-order:



changgou-service-order-api:



(2)application.yml配置

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

```
1 server:
2   port: 18089
3 spring:
4   application:
5     name: order
6   datasource:
7     driver-class-name: com.mysql.jdbc.Driver
8     url: jdbc:mysql://192.168.211.132:3306/changgou_order?
9       useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
10    username: root
11    password: 123456
12  redis:
13    host: 192.168.211.132
14    port: 6379
15  main:
16    allow-bean-definition-overriding: true
17  eureka:
```

```

17 client:
18     service-url:
19         defaultZone: http://127.0.0.1:7001/eureka
20 instance:
21     prefer-ip-address: true

```

(3)创建启动类

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

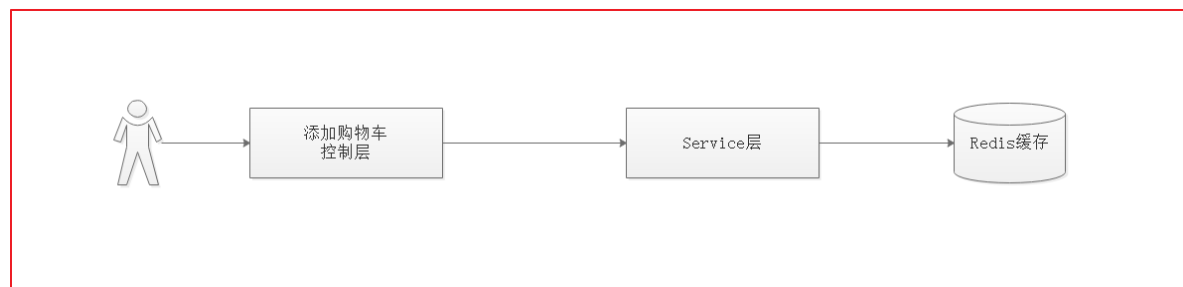
```

1  @SpringBootApplication
2  @EnableEurekaClient
3  @MapperScan(basePackages = {"com.changgou.order.dao"})
4  public class OrderApplication {
5
6      public static void main(String[] args) {
7          SpringApplication.run(OrderApplication.class,args);
8      }
9  }

```

5.3 添加购物车

5.3.1 思路分析



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：



5.3.2 代码实现

(1)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-order-api工程中的SkuFeign和SpuFeign根据ID查询方法如下：

com.changgou.goods.feign.SkuFeign

```
1  /**
2   * 根据ID查询SKU信息
3   * @param id : sku的ID
4   */
5  @GetMapping(value =("/{id}")
6  Result<Sku> findById(@PathVariable(value = "id", required = true) String
  id);
```

com.changgou.goods.feign.SpuFeign

```
1  /**
2   * 根据SpuID查询Spu信息
3   * @param id
4   * @return
5   */
6  @GetMapping("/{id}")
7  Result<Spu> findById(@PathVariable(value = "id") String id);
```

(2)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：

```
1  public interface CartService {
2
3      /**
4       * 添加购物车
5       * @param num:购买商品数量
6       * @param id: 购买ID
7       * @param username: 购买用户
8       * @return
9       */
10     void add(Integer num, String id, String username);
11 }
```

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：

```
1  @Service
2  public class CartServiceImpl implements CartService {
3
4      @Autowired
5      private RedisTemplate redisTemplate;
6
7      @Autowired
8      private SkuFeign skuFeign;
```



```

9
10 @Autowired
11 private SpuFeign spuFeign;
12
13
14 /**
15  * 加入购物车
16  * @param num:购买商品数量
17  * @param id: 购买ID
18  * @param username: 购买用户
19  * @return
20  */
21 @Override
22 public void add(Integer num, String id, String username) {
23     //查询SKU
24     Result<Sku> resultSku = skuFeign.findById(id);
25     if(resultSku!=null && resultSku.isFlag()){
26         //获取SKU
27         Sku sku = resultSku.getData();
28         //获取SPU
29         Result<Spu> resultSpu = spuFeign.findById(sku.getSpuId());
30
31         //将SKU转换成OrderItem
32         OrderItem orderItem = sku2OrderItem(sku,resultSpu.getData(),
num);
33
34         /**
35          * 购物车数据存入到Redis
36          * namespace = Cart_[username]
37          * key=id(sku)
38          * value=OrderItem
39          */
40         redisTemplate.boundHashOps("Cart_"+username).put(id,orderItem);
41     }
42 }
43
44 /**
45  * SKU转成OrderItem
46  * @param sku
47  * @param num
48  * @return
49  */
50 private OrderItem sku2OrderItem(Sku sku,Spu spu,Integer num){
51     OrderItem orderItem = new OrderItem();
52     orderItem.setSpuId(sku.getSpuId());
53     orderItem.setSkuId(sku.getId());
54     orderItem.setName(sku.getName());
55     orderItem.setPrice(sku.getPrice());
56     orderItem.setNum(num);
57     orderItem.setMoney(num*orderItem.getPrice()); //单价*数量
58     orderItem.setPayMoney(num*orderItem.getPrice()); //实付金额
59     orderItem.setImage(sku.getImage());
60     orderItem.setweight(sku.getweight()*num); //重量=单个重量*数
量
61
62     //分类ID设置
63     orderItem.setCategoryId1(spu.getCategory1Id());
64     orderItem.setCategoryId2(spu.getCategory2Id());

```

```

65         orderItem.setCategoryId3(spu.getCategory3Id());
66         return orderItem;
67     }
68 }

```

(3)控制层

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

```

1  @RestController
2  @CrossOrigin
3  @RequestMapping(value = "/cart")
4  public class CartController {
5
6      @Autowired
7      private CartService cartService;
8
9      /**
10       * 加入购物车
11       * @param num:购买的数量
12       * @param id: 购买的商品(SKU)ID
13       * @return
14       */
15      @RequestMapping(value = "/add")
16      public Result add(Integer num, String id){
17          //用户名
18          String username="szitheima";
19          //将商品加入购物车
20          cartService.add(num,id,username);
21          return new Result(true, StatusCode.OK,"加入购物车成功! ");
22      }
23 }

```

(4)feign配置

修改 com.changgou.OrderApplication 开启Feign客户端：

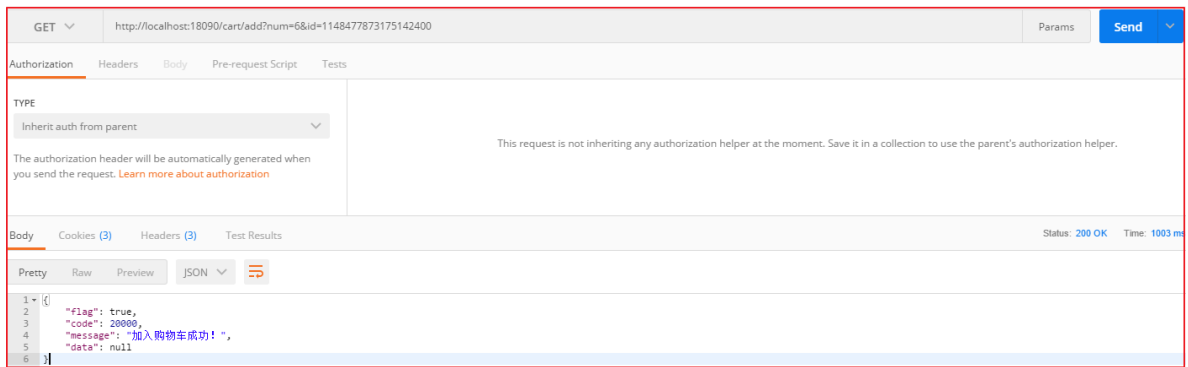
```

1  @SpringBootApplication
2  @EnableEurekaClient
3  @EnableFeignClients(basePackages = {"com.changgou.goods.feign"})
4  @MapperScan(basePackages = {"com.changgou.order.dao"})
5  public class OrderApplication {
6
7      public static void main(String[] args) {
8          SpringApplication.run(OrderApplication.class,args);
9      }
10 }

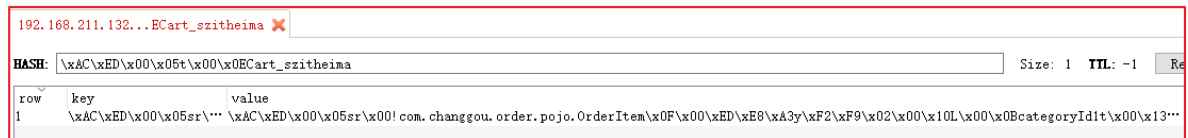
```

测试添加购物车，效果如下：

请求地址 <http://localhost:18089/cart/add?num=6&id=1148477873175142400>

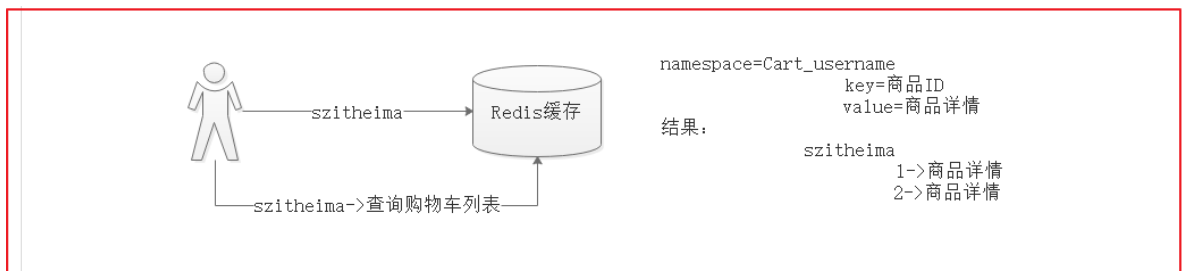


Redis缓存中的数据



5.4 购物车列表

5.4.1 思路分析



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。

5.4.2 代码实现

(1)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：

```
1  /**
2   * 查询用户的购物车数据
3   * @param username
4   * @return
5   */
6  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：

```
1  /**
2   * 查询用户购物车数据
3   * @param username
4   * @return
5   */
6  @Override
7  public List<OrderItem> list(String username) {
8      //查询所有购物车数据
9      List<OrderItem> orderItems =
10         redisTemplate.boundHashOps("Cart_"+username).values();
11         return orderItems;
12     }
13 }
```

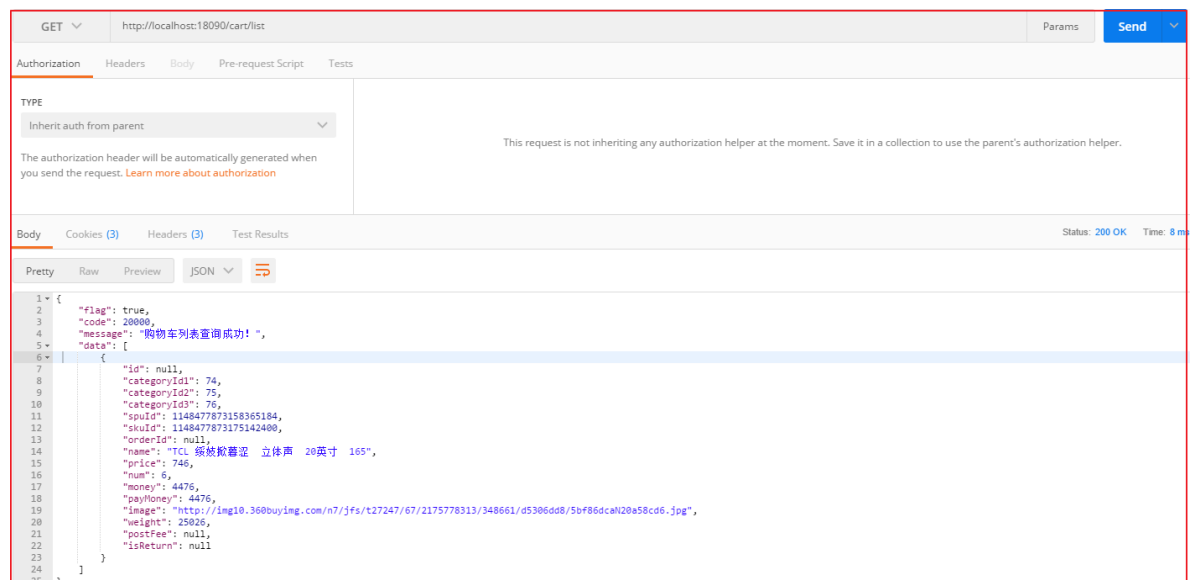
(2)控制层

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：

```
1  /**
2   * 查询用户购物车列表
3   * @return
4   */
5  @GetMapping(value = "/list")
6  public Result list(){
7      //用户名
8      String username="szitheima";
9      List<OrderItem> orderItems = cartService.list(username);
10         return new Result(true,StatusCode.OK,"购物车列表查询成功!",orderItems);
11     }
12 }
```

(3)测试

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



5.4.3 问题处理

(1)删除商品购物车

| 全部商品 | | | | | |
|-----------------------------|---|--------|-------|--------|------------|
| <input type="checkbox"/> 全部 | 商品 | 单价 (元) | 数量 | 小计 (元) | 操作 |
| <input type="checkbox"/> |  海尔 涉笔幻 立体声 20英寸 165 | 619 | - 2 + | 1238 | 删除 移到收藏 |
| <input type="checkbox"/> |  TCL 炫炫摇滚 立体声 20英寸 165 | 746 | - 0 + | 0 | 删除 移到收藏 |

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：

```
public void add(Integer num, Long id, String username) {  
    //删除该商品的购物车数据  
    if(num<=0){  
        redisTemplate.boundHashOps( key: "Cart_"+username).delete(id);  
        return;  
    }  
    //查询SKU  
    Result<Sku> resultSku = skuFeign.findById(id);  
    if(resultSku!=null && resultSku.isFlag()){  
        //获取SKU  
        Sku sku = resultSku.getData();  
        //获取SPU  
        Result<Spu> resultSpu = spuFeign.findById(sku.getSpuId());  
  
        //将SKU转换成OrderItem  
        OrderItem orderItem = sku2OrderItem(sku,resultSpu.getData(), num);  
  
        /****  
        * 购物车数据存入到Redis  
        * namespace = Cart_[username]  
        * key=id(sku)  
        * value=OrderItem  
        */  
        redisTemplate.boundHashOps( key: "Cart_"+username).put(id, orderItem);  
    }  
}
```

如果传入的数量 ≤ 0 ，则删除该商品的购物车数据

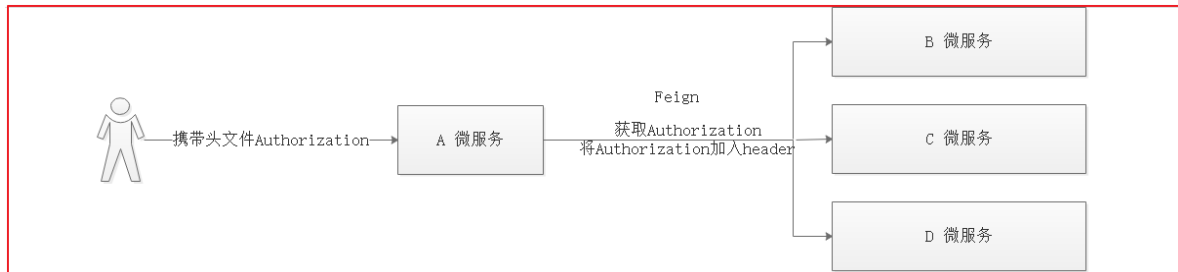
6 用户身份识别

6.1 购物车需求分析



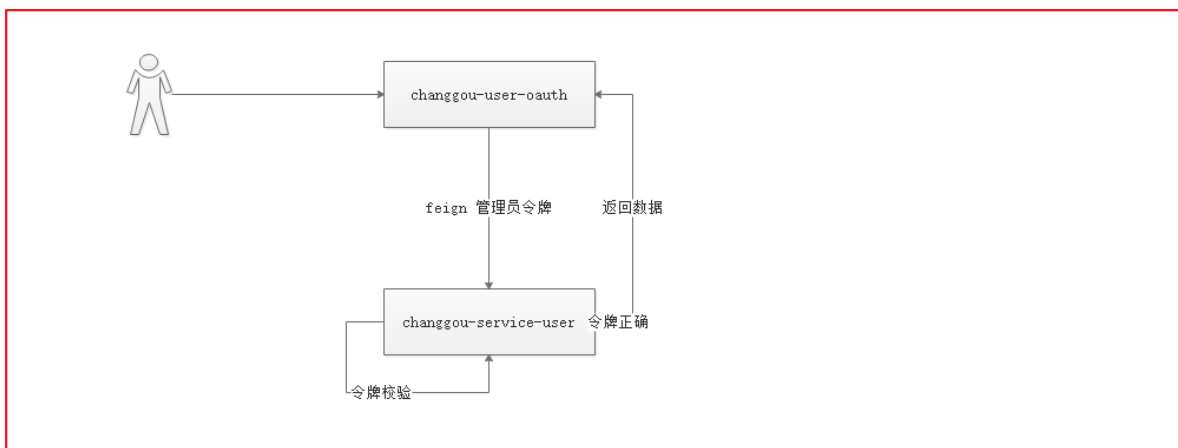
购物车功能已经做完了，但用户我们都是硬编码写死的。用户要想将商品加入购物车，必须先登录授权，登录授权后再经过微服务网关，微服务网关需要过滤判断用户请求是否存在令牌，如果存在令牌，才能再次访问微服务，此时网关会通过过滤器将令牌数据再次存入到头文件中，将令牌数据传递给购物车订单微服务，到了购物车订单微服务的时候，此时微服务需要校验令牌数据，如果令牌正确，才能使用购物车功能，并解析令牌数据获取用户信息。

6.2 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



(1)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` 代码如下：

```
1 public class JwtToken {
2
3     /**
4      * 获取管理员令牌
5      * @return
6      */
7     public static String adminJwt(){
8         //密钥->私钥
9         Resource resource = new ClassPathResource("changgou.jks");
10
11     /**
12      * 加载证书,读取证书数据
13      * 1:证书对象
```

```

14         * 2:证书的密码
15         */
16         KeyStoreKeyFactory keyStoreKeyFactory = new
KeyStoreKeyFactory(resource, "changgou".toCharArray());
17
18         //把私钥信息当做秘钥
19         KeyPair keyPair =
keyStoreKeyFactory.getKeyPair("changgou", "changgou".toCharArray());
20         PrivateKey privateKey = keyPair.getPrivate();
21         RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) privateKey;
22         //并指定加密算法
23         RsaSigner rsaSigner = new RsaSigner(rsaPrivateKey);
24
25         //添加载荷数据 payload
26         Map<String, Object> map = new HashMap<String, Object>();
27         map.put("authorities", new String[]{"admin"});
28
29         //生成令牌 JwtHelper生成令牌/解析令牌
30         Jwt jwt = JwtHelper.encode(JSON.toJSONString(map), rsaSigner);
31         return jwt.getEncoded();
32     }
33 }

```

(2)拦截器创建

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

```

1  @Configuration
2  public class FeignOauth2RequestInterceptor implements RequestInterceptor {
3
4      /**
5       * 自定义操作
6       */
7      @Override
8      public void apply(RequestTemplate requestTemplate) {
9          //创建令牌信息
10         String token = "Bearer " + JwtToken.adminJwt();
11         //将令牌添加到头文件中
12         requestTemplate.header("Authorization", token);
13     }
14 }

```

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：

```

1  @Configuration
2  public class FeignOauth2RequestInterceptor implements RequestInterceptor {
3
4      /**
5       * 自定义操作
6       */
7      @Override
8      public void apply(RequestTemplate requestTemplate) {
9          try {
10             //创建令牌信息

```

```

11         String token = "Bearer " + JwtToken.adminJwt();
12         //将令牌添加到头文件中
13         requestTemplate.header("Authorization", token);
14
15         //使用RequestContextHolder工具获取request相关变量
16         ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
17         if (attributes != null) {
18             //取出request
19             HttpServletRequest request = attributes.getRequest();
20             //获取所有头文件信息的key
21             Enumeration<String> headerNames = request.getHeaderNames();
22             if (headerNames != null) {
23                 while (headerNames.hasMoreElements()) {
24                     //头文件的key
25                     String name = headerNames.nextElement();
26                     //头文件的value
27                     String values = request.getHeader(name);
28                     //将令牌数据添加到头文件中
29                     requestTemplate.header(name, values);
30                 }
31             }
32         }
33     } catch (Exception e) {
34         e.printStackTrace();
35     }
36 }
37 }

```

(1)创建拦截器

在 changgou-user-oauth 服务中创建一个com.changgou.order.interceptor.FeignInterceptor拦截器，并将所有头文件数据再次加入到Feign请求的微服务头文件中，代码如下：

```

1 public class FeignInterceptor implements RequestInterceptor {
2
3     @Override
4     public void apply(RequestTemplate requestTemplate) {
5         try {
6             //使用RequestContextHolder工具获取request相关变量
7             ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
8             if (attributes != null) {
9                 //取出request
10                HttpServletRequest request = attributes.getRequest();
11                //获取所有头文件信息的key
12                Enumeration<String> headerNames = request.getHeaderNames();
13                if (headerNames != null) {
14                    while (headerNames.hasMoreElements()) {
15                        //头文件的key
16                        String name = headerNames.nextElement();
17                        //头文件的value
18                        String values = request.getHeader(name);
19                        //将令牌数据添加到头文件中

```



```

20         requestTemplate.header(name, values);
21     }
22 }
23 }
24 } catch (Exception e) {
25     e.printStackTrace();
26 }
27 }
28 }

```

(2)创建拦截器Bean

在changgou-web-order服务中启动类里创建对象实例

```

1  /**
2   * 创建拦截器Bean对象
3   * @return
4   */
5  @Bean
6  public FeignInterceptor feignInterceptor(){
7      return new FeignInterceptor();
8  }

```

(3)测试

微服务之间相互调用，如果使用Feign调用，如果开启feign熔断，默认采用的是线程，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：

```

1  feign:
2      hystrix:
3          enabled: true

```

拦截器中出现空指针问题：



```

public void apply(RequestTemplate requestTemplate) {
    try {
        //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();

```

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-web-order的application.yml配置文件，在application.yml中添加如下代码，代码如下：

```

1  #hystrix 配置
2  hystrix:
3      command:
4          default:
5              execution:
6                  isolation:
7                      thread:
8                          timeoutInMilliseconds: 10000
9                          strategy: SEMAPHORE

```

隔离区别：

| 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 | 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） |

再次测试，attributes就不为空了。

(4)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，其他工程需要用，直接创建一个@Bean对象即可。

```

1  public class FeignInterceptor implements RequestInterceptor {
2
3      @Override
4      public void apply(RequestTemplate requestTemplate) {
5          try {
6              //使用RequestContextHolder工具获取request相关变量
7              ServletRequestAttributes attributes = (ServletRequestAttributes)
RequestContextHolder.getRequestAttributes();
8              if (attributes != null) {
9                  //取出request
10                 HttpServletRequest request = attributes.getRequest();
11                 //获取所有头文件信息的key
12                 Enumeration<String> headerNames = request.getHeaderNames();
13                 if (headerNames != null) {
14                     while (headerNames.hasMoreElements()) {
15                         //头文件的key
16                         String name = headerNames.nextElement();
17                         //头文件的value
18                         String values = request.getHeader(name);
19                         //将令牌数据添加到头文件中
20                         requestTemplate.header(name, values);
21                     }
22                 }
23             }
24         } catch (Exception e) {

```

```

25         e.printStackTrace();
26     }
27 }
28 }

```

6.3 网关过滤

用户请求，会先经过微服务网关，请求才会到达微服务，所以我们需要先在微服务网关中配置下路由规则。

6.4 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：

```

#订单微服务
- id: changgou_order_route
  uri: lb://order
  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

```

上图代码如下：

```

1      #订单微服务
2      - id: changgou_order_route
3        uri: lb://order
4        predicates:
5          -
6            Path=/api/cart/**,/api/categoryReport/**,/api/orderConfig/**,/api/order/**,/a
7            pi/orderItem/**,/api/orderLog/**,/api/preferential/**,/api/returnCause/**,/ap
            i/returnOrder/**,/api/returnOrderItem/**
6            filters:
7              - StripPrefix=1

```

这里注意使用的是 `yml` 格式，所以上面代码中的空格也一并记得拷贝到 `application.yml` 文件中。

6.5 获取用户数据

6.5.1 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：

```
private final String remoteAddress;

private final String sessionId;

private final String tokenValue;

private final String tokenType;

private final String display;

private Object decodedDetails;
```

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，使用公钥对它进行校验，如果能校验说明说句无误，如果不能校验用户也没法执行到这一步。校验后可以从明文中获取用户信息。

6.5.2 代码实现

在changgou-service-order微服务中创建com.changgou.order.config.TokenDecode类，用于解密令牌信息，在类中读取公钥信息，代码如下：

```
1 public class TokenDecode {
2
3     //公钥
4     private static final String PUBLIC_KEY = "public.key";
5
6     private static String publickey="";
7
8
9     /**
10      * 获取非对称加密公钥 key
11      * @return 公钥 Key
12      */
13     public static String getPubKey() {
14         if(!StringUtils.isEmpty(publickey)){
15             return publickey;
16         }
17         Resource resource = new ClassPathResource(PUBLIC_KEY);
18         try {
19             InputStreamReader inputStreamReader = new
20             InputStreamReader(resource.getInputStream());
21             BufferedReader br = new BufferedReader(inputStreamReader);
22             publickey = br.lines().collect(Collectors.joining("\n"));
23             return publickey;
24         } catch (IOException ioe) {
25             return null;
26         }
27     }
28
29     /**
30      * 读取令牌数据
31      */
32     public static Map<String,String> dcodeToken(String token){
33         //校验Jwt
```

```

33         Jwt jwt = JwtHelper.decodeAndVerify(token, new
Rsaverifier(getPubKey()));
34
35         //获取Jwt原始内容
36         String claims = jwt.getClaims();
37         return JSON.parseObject(claims, Map.class);
38     }
39
40     /**
41      * 获取用户信息
42      * @return
43      */
44     public static Map<String,String> getUserInfo(){
45         //获取授权信息
46         OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails)
SecurityContextHolder.getContext().getAuthentication().getDetails();
47         //令牌解码
48         return dcodeToken(details.getTokenValue());
49     }
50 }

```

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：

```

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

/**
 * 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(flag: true, StatusCode.OK, message: "加入购物车成功!");
}

```

6.6 代码抽取

以后很有可能在很多微服务中都会用到该对象来获取用户信息，我们可以把它抽取出去。

在changgou-common工程中引入鉴权包

```
1 <!--oauth依赖-->
2 <dependency>
3     <groupId>org.springframework.cloud</groupId>
4     <artifactId>spring-cloud-starter-oauth2</artifactId>
5     <scope>provided</scope>
6 </dependency>
```

将TokenDecode拷贝到com.changgou.util包中，并且删除changgou-service-order中的TokenDecode，哪里需要用到，直接使用即可。

课后要求:

- 1.资源服务器授权的配置: 加坐标 copy公钥文件 创建配置文件
- 2.测试一下角色控制controller
- 3.实现授权码模式的数据库数据动态加载,测试一下授权码模式
- 4.实现密码模式的动态数据加载: 用户微服务提供一个feign接口查询用户的信息 放行一个查询的接口

注意:修改配置文件中的clientId和clientSecret

- 5.购物车添加和购物车查询(先将用户名写死在controller中)
- 6.关闭用户微服务中的放行(步骤4中放行的那个接口),添加临时令牌,将浏览器请求的参数全部放入resttemplate的请求头中去,开启feign的熔断器,配置熔断的线程隔离为信号量隔离
- 7.动态加载用户的数据,工具类引入以后,通过工具类getUserInfo, get("username")获取要用户的用户名