尚医通

版本：V1.0

# 登录需求

## 登录效果



## 登录需求

1. 登录采取弹出层的形式
2. 登录方式：
3. 手机号码+手机验证码
4. 微信扫描
5. 无注册界面，第一次登录根据手机号判断系统是否存在，如果不存在则自动注册
6. 微信扫描登录成功必须绑定手机号码，即：第一次扫描成功后绑定手机号，以后登录扫描直接登录成功
7. 网关统一判断登录状态，如何需要登录，页面弹出登录层

# 登录

## 搭建service-user模块

### 搭建service-user模块

搭建过程参考service-hosp模块

### 1.2 修改配置

1、修改pom.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>*  <**project xmlns="http://maven.apache.org/POM/4.0.0"**  **xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**  **xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**parent**>  <**artifactId**>service</**artifactId**>  <**groupId**>com.atguigu</**groupId**>  <**version**>0.0.1-SNAPSHOT</**version**>  </**parent**>  <**modelVersion**>4.0.0</**modelVersion**>  <**artifactId**>service\_user</**artifactId**>  <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu</**groupId**>  <**artifactId**>service\_cmn\_client</**artifactId**>  <**version**>0.0.1-SNAPSHOT</**version**>  </**dependency**>  </**dependencies**>  </**project**> |

2、添加配置文件application.properties

|  |
| --- |
| # 服务端口  **server.port**=**8203**  # 服务名  **spring.application.name**=**service-user**  # 环境设置：dev、test、prod  **spring.profiles.active**=**dev**  # mysql数据库连接  **spring.datasource.driver-class-name**=**com.mysql.jdbc.Driver**  **spring.datasource.url**=**jdbc:mysql://192.168.44.165:3306/yygh\_user?characterEncoding=utf-8&useSSL=false**  **spring.datasource.username**=**root**  **spring.datasource.password**=**root123**  #返回json的全局时间格式  **spring.jackson.date-format**=**yyyy-MM-dd HH:mm:ss**  **spring.jackson.time-zone**=**GMT+8**  # nacos服务地址  **spring.cloud.nacos.discovery.server-addr**=**127.0.0.1:8848**  #配置mapper xml文件的路径  **mybatis-plus.mapper-locations**=**classpath:com/atguigu/yygh/user/mapper/xml/\*.xml** |

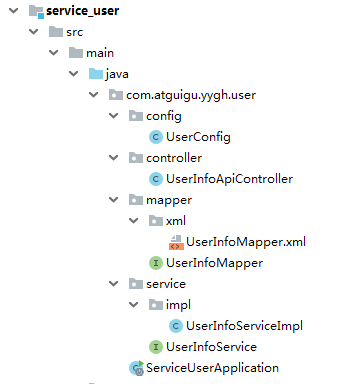
### 1.3 启动类

|  |
| --- |
| @SpringBootApplication  @ComponentScan(basePackages = **"com.atguigu"**)  @EnableDiscoveryClient  @EnableFeignClients(basePackages = **"com.atguigu"**)  **public class** ServiceUserApplication {  **public static void** main(String[] args) {  SpringApplication.*run*(ServiceUserApplication.**class**, args);  }  } |

### 1.4配置网关

|  |
| --- |
| *#设置路由id*  **spring.cloud.gateway.routes[2].id**=**service-user**  *#设置路由的uri*  **spring.cloud.gateway.routes[2].uri**=**lb://service-user**  *#设置路由断言,代理servicerId为auth-service的/auth/路径*  **spring.cloud.gateway.routes[2].predicates**= **Path=/\*/user/\*\*** |

## 2、添加用户基础类



### 2.1 添加model

说明：由于实体对象没有逻辑，我们已经统一导入

com.atguigu.yygh.model.user.UserInfo

### 2.2 添加Mapper

1，添加com.atguigu.yygh.user.UserInfoMapper

|  |
| --- |
| **import** com.atguigu.yygh.model.user.UserInfo;  **import** com.baomidou.mybatisplus.core.mapper.BaseMapper;  **public interface** UserInfoMapper **extends** BaseMapper<UserInfo> {  } |

1. 添加UserInfoMapper.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>*  **<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"**  **"http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>***  <**mapper namespace="com.atguigu.yygh.user.mapper.UserInfoMapper"**>  </**mapper**> |

### 2.3 添加service接口及实现类

1、添加com.atguigu.yygh.user.service.UserInfoService接口

|  |
| --- |
| **public interface** UserInfoService **extends** IService<UserInfo> {  } |

2、添加com.atguigu.yygh.user.service.impl.UserInfoServiceImpl接口实现

|  |
| --- |
| @Service  **public class** UserInfoServiceImpl **extends**  ServiceImpl<UserInfoMapper, UserInfo> **implements** UserInfoService {  } |

### 2.4 添加controller

添加com.atguigu.yygh.user.api.UserInfoApiController类

|  |
| --- |
| @RestController  @RequestMapping(**"/api/user"**)  **public class** UserInfoApiController {  @Autowired  **private** UserInfoService **userInfoService**;  } |

## 3、登录api接口

### 3.1 添加service接口与实现

1、在UserInfoService类添加接口

|  |
| --- |
| *//会员登录*  Map<String, Object> login(LoginVo loginVo); |

2，在UserInfoServiceImpl类实现接口

|  |
| --- |
| @Service  **public class** UserInfoServiceImpl **extends**  ServiceImpl<UserInfoMapper, UserInfo> **implements** UserInfoService {  @Override  **public** Map<String, Object> login(LoginVo loginVo) {  String phone = loginVo.getPhone();  String code = loginVo.getCode();  *//校验参数*  **if**(StringUtils.*isEmpty*(phone) ||  StringUtils.*isEmpty*(code)) {  **throw new** YyghException(ResultCodeEnum.***PARAM\_ERROR***);  }  *//****TODO 校验校验验证码***  *//手机号已被使用*  QueryWrapper<UserInfo> queryWrapper = **new** QueryWrapper<>();  queryWrapper.eq(**"phone"**, phone);  *//获取会员*  UserInfo userInfo = **baseMapper**.selectOne(queryWrapper);  **if**(**null** == userInfo) {  userInfo = **new** UserInfo();  userInfo.setName(**""**);  userInfo.setPhone(phone);  userInfo.setStatus(1);  **this**.save(userInfo);  }  *//校验是否被禁用*  **if**(userInfo.getStatus() == 0) {  **throw new** YyghException(ResultCodeEnum.***LOGIN\_DISABLED\_ERROR***);  }  *//****TODO 记录登录***  *//返回页面显示名称*  Map<String, Object> map = **new** HashMap<>();  String name = userInfo.getName();  **if**(StringUtils.*isEmpty*(name)) {  name = userInfo.getNickName();  }  **if**(StringUtils.*isEmpty*(name)) {  name = userInfo.getPhone();  }  map.put(**"name"**, name);  map.put(**"token"**, **""**);  **return** map;  }  } |

说明：

1. 验证码先注释，后续校验
2. 登录成功生成token，后续讲解

### 3.2 添加controller接口

1、在UserInfoApiController类添加方法

|  |
| --- |
| @ApiOperation(value = **"会员登录"**)  @PostMapping(**"login"**)  **public** Result login(@RequestBody LoginVo loginVo, HttpServletRequest request) {  loginVo.setIp(IpUtil.*getIpAddr*(request));  Map<String, Object> info = **userInfoService**.login(loginVo);  **return** Result.*ok*(info);  } |

2、添加IpUtil工具类

|  |
| --- |
| **public class** IpUtil {  **private static final** String ***UNKNOWN*** = **"unknown"**;  **private static final** String ***LOCALHOST*** = **"127.0.0.1"**;  **private static final** String ***SEPARATOR*** = **","**;  **public static** String getIpAddr(HttpServletRequest request) {  System.***out***.println(request);  String ipAddress;  **try** {  ipAddress = request.getHeader(**"x-forwarded-for"**);  **if** (ipAddress == **null** || ipAddress.length() == 0 || ***UNKNOWN***.equalsIgnoreCase(ipAddress)) {  ipAddress = request.getHeader(**"Proxy-Client-IP"**);  }  **if** (ipAddress == **null** || ipAddress.length() == 0 || ***UNKNOWN***.equalsIgnoreCase(ipAddress)) {  ipAddress = request.getHeader(**"WL-Proxy-Client-IP"**);  }  **if** (ipAddress == **null** || ipAddress.length() == 0 || ***UNKNOWN***.equalsIgnoreCase(ipAddress)) {  ipAddress = request.getRemoteAddr();  **if** (***LOCALHOST***.equals(ipAddress)) {  InetAddress inet = **null**;  **try** {  inet = InetAddress.*getLocalHost*();  } **catch** (UnknownHostException e) {  e.printStackTrace();  }  ipAddress = inet.getHostAddress();  }  }  *// 对于通过多个代理的情况，第一个IP为客户端真实IP,多个IP按照','分割*  *// "\*\*\*.\*\*\*.\*\*\*.\*\*\*".length()*  **if** (ipAddress != **null** && ipAddress.length() >15) {  **if** (ipAddress.indexOf(***SEPARATOR***) >0) {  ipAddress = ipAddress.substring(0, ipAddress.indexOf(**","**));  }  }  } **catch** (Exception e) {  ipAddress = **""**;  }  **return** ipAddress;  }  } |

## 5、生成token

### 5.1 JWT介绍

JWT工具

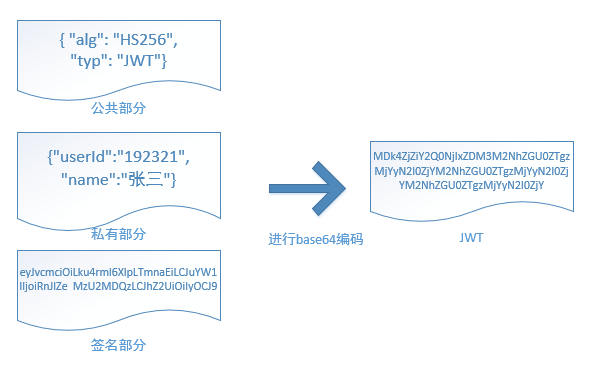
JWT（Json Web Token）是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准。

JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源。比如用在用户登录上

JWT最重要的作用就是对 token信息的**防伪**作用。

JWT的原理，

一个JWT由三个部分组成：公共部分、私有部分、签名部分。最后由这三者组合进行base64编码得到JWT。



1. 公共部分

主要是该JWT的相关配置参数，比如签名的加密算法、格式类型、过期时间等等。

Key=ATGUIGU

1. 私有部分

用户自定义的内容，根据实际需要真正要封装的信息。

userInfo{用户的Id，用户的昵称nickName}

1. 签名部分

SaltiP: 当前服务器的Ip地址!{linux 中配置代理服务器的ip}

主要用户对JWT生成字符串的时候，进行加密{盐值}

最终组成 key+salt+userInfo 🡺 token!

base64编码，并不是加密，只是把明文信息变成了不可见的字符串。但是其实只要用一些工具就可以把base64编码解成明文，所以不要在JWT中放入涉及私密的信息。

### 5.2 集成JWT

1，在common-util模块pom.xml添加依赖

|  |
| --- |
| <**dependency**>  <**groupId**>io.jsonwebtoken</**groupId**>  <**artifactId**>jjwt</**artifactId**>  </**dependency**> |

版本已在yygh-parent父模块pom.xml添加

2，在common-util模块编写JwtHelper类

|  |
| --- |
| **public class** JwtHelper {  **private static long** *tokenExpiration* = 24\*60\*60\*1000;  **private static** String *tokenSignKey* = **"123456"**;  **public static** String createToken(Long userId, String userName) {  String token = Jwts.*builder*()  .setSubject(**"YYGH-USER"**)  .setExpiration(**new** Date(System.*currentTimeMillis*() + *tokenExpiration*))  .claim(**"userId"**, userId)  .claim(**"userName"**, userName)  .signWith(SignatureAlgorithm.***HS512***, *tokenSignKey*)  .compressWith(CompressionCodecs.***GZIP***)  .compact();  **return** token;  }  **public static** Long getUserId(String token) {  **if**(StringUtils.*isEmpty*(token)) **return null**;  Jws<Claims> claimsJws = Jwts.*parser*().setSigningKey(*tokenSignKey*).parseClaimsJws(token);  Claims claims = claimsJws.getBody();  Integer userId = (Integer)claims.get(**"userId"**);  **return** userId.longValue();  }  **public static** String getUserName(String token) {  **if**(StringUtils.*isEmpty*(token)) **return ""**;  Jws<Claims> claimsJws  = Jwts.*parser*().setSigningKey(*tokenSignKey*).parseClaimsJws(token);  Claims claims = claimsJws.getBody();  **return** (String)claims.get(**"userName"**);  }  **public static void** main(String[] args) {  String token = JwtHelper.*createToken*(1L, **"55"**);  System.***out***.println(token);  System.***out***.println(JwtHelper.*getUserId*(token));  System.***out***.println(JwtHelper.*getUserName*(token));  }  } |

说明：执行main方法测试

### 5.3 完善登录service接口

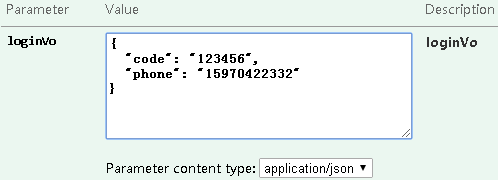
修改UserInfoServiceImpl类登录方法

|  |
| --- |
| **public** Map<String, Object> loginUser(LoginVo loginVo) {  …………  *//jwt生成token字符串*  String token = JwtHelper.*createToken*(userInfo.getId(), name);  map.put(**"token"**,token);  **return** map;  } |

### 5.4 使用Swagger测试接口

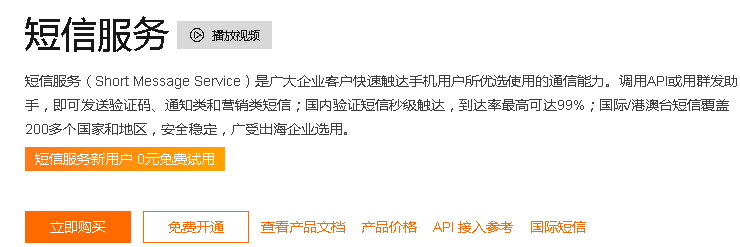
测试多次，第一次注册，以后直接登录

请求参数：



## 阿里云短信

### 6.1阿里云短信介绍



#### 6.1.1开通阿里云短信服务



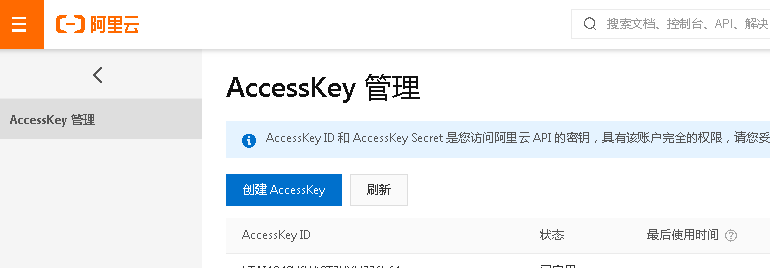
#### 6.1.2添加签名管理与模板管理





注：审批通过后方可使用

#### 6.1.3获取用户AccessKey



### 6.2搭建service-msm模块

#### 6.2.1 搭建service-msm模块

搭建过程参考service-hosp模块

#### 6.2.2修改配置

1、修改pom.xml

|  |
| --- |
| <**dependencies**>  <**dependency**>  <**groupId**>com.aliyun</**groupId**>  <**artifactId**>aliyun-java-sdk-core</**artifactId**>  </**dependency**>  </**dependencies**> |

2、添加配置文件application.properties

|  |
| --- |
| *# 服务端口*  **server.port**=**8204**  *# 服务名*  **spring.application.name**=**service-msm**  *#返回json的全局时间格式*  **spring.jackson.date-format**=**yyyy-MM-dd HH:mm:ss**  **spring.jackson.time-zone**=**GMT+8**  **spring.redis.host**=**192.168.44.165**  **spring.redis.port**=**6379**  **spring.redis.database**= **0**  **spring.redis.timeout**=**1800000**  **spring.redis.lettuce.pool.max-active**=**20**  **spring.redis.lettuce.pool.max-wait**=**-1**  *#最大阻塞等待时间(负数表示没限制)*  **spring.redis.lettuce.pool.max-idle**=**5**  **spring.redis.lettuce.pool.min-idle**=**0**  *# nacos服务地址*  **spring.cloud.nacos.discovery.server-addr**=**127.0.0.1:8848**    **aliyun.sms.regionId**=**default**  **aliyun.sms.accessKeyId**=**LT6I0Y5633pX89qC**  **aliyun.sms.secret**=**jX8D04Dm12I3gGKj345FYSzu0fq8mT** |

#### 6.2.3 启动类

|  |
| --- |
| @SpringBootApplication(exclude = DataSourceAutoConfiguration.**class**)*//取消数据源自动配置*  @EnableDiscoveryClient  **public class** ServiceMsmApplication {  **public static void** main(String[] args) {  SpringApplication.*run*(ServiceMsmApplication.**class**, args);  }  } |

#### 6.2.4 配置网关

|  |
| --- |
| *#设置路由id*  **spring.cloud.gateway.routes[3].id**=**service-msm**  *#设置路由的uri*  **spring.cloud.gateway.routes[3].uri**=**lb://service-msm**  *#设置路由断言,代理servicerId为auth-service的/auth/路径*  **spring.cloud.gateway.routes[3].predicates**= **Path=/\*/msm/\*\*** |

### 6.3封装注册短信验证码接口

#### 6.3.1 添加配置类

|  |
| --- |
| @Component  **public class** ConstantPropertiesUtils **implements** InitializingBean {  @Value(**"${aliyun.sms.regionId}"**)  **private** String **regionId**;  @Value(**"${aliyun.sms.accessKeyId}"**)  **private** String **accessKeyId**;  @Value(**"${aliyun.sms.secret}"**)  **private** String **secret**;  **public static** String *REGION\_Id*;  **public static** String *ACCESS\_KEY\_ID*;  **public static** String *SECRECT*;  @Override  **public void** afterPropertiesSet() **throws** Exception {  *REGION\_Id*=**regionId**;  *ACCESS\_KEY\_ID*=**accessKeyId**;  *SECRECT*=**secret**;  }  } |

#### 6.3.2 封装service接口和实现类

|  |
| --- |
| **public interface** MsmService {  *//发送手机验证码*  **boolean** send(String phone, String code);  } |

|  |
| --- |
| @Service  **public class** MsmServiceImpl **implements** MsmService {  @Override  **public boolean** send(String phone, String code) {  *//判断手机号是否为空*  **if**(StringUtils.*isEmpty*(phone)) {  **return false**;  }  *//整合阿里云短信服务*  *//设置相关参数*  DefaultProfile profile = DefaultProfile.  *getProfile*(ConstantPropertiesUtils.*REGION\_Id*,  ConstantPropertiesUtils.*ACCESS\_KEY\_ID*,  ConstantPropertiesUtils.*SECRECT*);  IAcsClient client = **new** DefaultAcsClient(profile);  CommonRequest request = **new** CommonRequest();  *//request.setProtocol(ProtocolType.HTTPS);*  request.setMethod(MethodType.***POST***);  request.setDomain(**"dysmsapi.aliyuncs.com"**);  request.setVersion(**"2017-05-25"**);  request.setAction(**"SendSms"**);  *//手机号*  request.putQueryParameter(**"PhoneNumbers"**, phone);  *//签名名称*  request.putQueryParameter(**"SignName"**, **"我的谷粒在线教育网站"**);  *//模板code*  request.putQueryParameter(**"TemplateCode"**, **"SMS\_180051135"**);  *//验证码 使用json格式 {"code":"123456"}*  Map<String,Object> param = **new** HashMap();  param.put(**"code"**,code);  request.putQueryParameter(**"TemplateParam"**, JSONObject.*toJSONString*(param));  *//调用方法进行短信发送*  **try** {  CommonResponse response = client.getCommonResponse(request);  System.***out***.println(response.getData());  **return** response.getHttpResponse().isSuccess();  } **catch** (ServerException e) {  e.printStackTrace();  } **catch** (ClientException e) {  e.printStackTrace();  }  **return false**;  }  } |

#### 6.3.3 封装controller接口

|  |
| --- |
| @RestController  @RequestMapping(**"/api/msm"**)  **public class** MsmApiController {  @Autowired  **private** MsmService **msmService**;  @Autowired  **private** RedisTemplate<String,String> **redisTemplate**;  *//发送手机验证码*  @GetMapping(**"send/{phone}"**)  **public** Result sendCode(@PathVariable String phone) {  *//从redis获取验证码，如果获取获取到，返回ok*  *// key 手机号 value 验证码*  String code = **redisTemplate**.opsForValue().get(phone);  **if**(!StringUtils.*isEmpty*(code)) {  **return** Result.*ok*();  }  *//如果从redis获取不到，*  *// 生成验证码，*  code = RandomUtil.*getSixBitRandom*();  *//调用service方法，通过整合短信服务进行发送*  **boolean** isSend = **msmService**.send(phone,code);  *//生成验证码放到redis里面，设置有效时间*  **if**(isSend) {  **redisTemplate**.opsForValue().set(phone,code,2, TimeUnit.***MINUTES***);  **return** Result.*ok*();  } **else** {  **return** Result.*fail*().message(**"发送短信失败"**);  }  }  } |

工具类

|  |
| --- |
| **public class** RandomUtil {  **private static final** Random ***random*** = **new** Random();  **private static final** DecimalFormat ***fourdf*** = **new** DecimalFormat(**"0000"**);  **private static final** DecimalFormat ***sixdf*** = **new** DecimalFormat(**"000000"**);  **public static** String getFourBitRandom() {  **return *fourdf***.format(***random***.nextInt(10000));  }  **public static** String getSixBitRandom() {  **return *sixdf***.format(***random***.nextInt(1000000));  }  */\*\**  *\* 给定数组，抽取n个数据*  *\** ***@param list***  *\** ***@param n***  *\** ***@return***  *\*/*  **public static** ArrayList getRandom(List list, **int** n) {  Random random = **new** Random();  HashMap<Object, Object> hashMap = **new** HashMap<Object, Object>();  *// 生成随机数字并存入HashMap*  **for** (**int** i = 0; i < list.size(); i++) {  **int** number = random.nextInt(100) + 1;  hashMap.put(number, i);  }  *// 从HashMap导入数组*  Object[] robjs = hashMap.values().toArray();  ArrayList r = **new** ArrayList();  *// 遍历数组并打印数据*  **for** (**int** i = 0; i < n; i++) {  r.add(list.get((**int**) robjs[i]));  System.***out***.print(list.get((**int**) robjs[i]) + **"\t"**);  }  System.***out***.print(**"\n"**);  **return** r;  }  } |

#### 6.3.4 完善登录service接口

修改UserInfoServiceImpl类登录方法，编写验证码校验

|  |
| --- |
| *//校验校验验证码*  String mobleCode = redisTemplate.opsForValue().get(phone);  **if**(!code.equals(mobleCode)) {  **throw new** YyghException(ResultCodeEnum.***CODE\_ERROR***);  } |

## 登录前端

### 7.1 封装api请求

创建api文件夹，创建/api/userInfo.js

|  |
| --- |
| import request from '@/utils/request'  const api\_name = `/api/user`  export default {      login(userInfo) {          return request({              url: `${api\_name}/login`,              method: `post`,              data: userInfo          })      }  } |

创建/api/msm.js

|  |
| --- |
| import request from '@/utils/request'  const api\_name = `/api/msm`  export default {      sendCode(mobile) {          return request({              url: `${api\_name}/send/${mobile}`,              method: `get`          })      }  } |

### 7.2安装cookie

登录成功，我们要把用户信息记录在cookie里面

命令行执行：

npm install js-cookie

### 7.3 添加登录组件

登录层是一个公共层，因此我们把它放在头部组件里面

修改layouts/myheader.vue文件

|  |
| --- |
| <!-- 登录弹出层 -->      <el-dialog :visible.sync="dialogUserFormVisible" style="text-align: left;" top="50px" :append-to-body="true"  width="960px" @close="closeDialog()">        <div class="container">          <!-- 手机登录 #start -->          <div class="operate-view" v-if="dialogAtrr.showLoginType === 'phone'">            <div class="wrapper" style="width: 100%">              <div class="mobile-wrapper" style="position: static;width: 70%">                <span class="title">{{ dialogAtrr.labelTips }}</span>                <el-form>                  <el-form-item>                    <el-input v-model="dialogAtrr.inputValue" :placeholder="dialogAtrr.placeholder" :maxlength="dialogAtrr.maxlength" class="input v-input">                      <span slot="suffix" class="sendText v-link" v-if="dialogAtrr.second > 0">{{ dialogAtrr.second }}s </span>                      <span slot="suffix" class="sendText v-link highlight clickable selected" v-if="dialogAtrr.second == 0" @click="getCodeFun()">重新发送 </span>                    </el-input>                  </el-form-item>                </el-form>                <div class="send-button v-button" @click="btnClick()"> {{ dialogAtrr.loginBtn }}</div>              </div>              <div class="bottom">                <div class="wechat-wrapper" @click="weixinLogin()"><span                  class="iconfont icon"></span></div>                <span class="third-text"> 第三方账号登录 </span></div>            </div>          </div>          <!-- 手机登录 #end -->          <!-- 微信登录 #start -->          <div class="operate-view"  v-if="dialogAtrr.showLoginType === 'weixin'" >            <div class="wrapper wechat" style="height: 400px">              <div>                <div id="weixinLogin"></div>              </div>              <div class="bottom wechat" style="margin-top: -80px;">                <div class="phone-container">                  <div class="phone-wrapper"  @click="phoneLogin()"><span                    class="iconfont icon"></span></div>                  <span class="third-text"> 手机短信验证码登录 </span></div>              </div>            </div>          </div>          <!-- 微信登录 #end -->          <div class="info-wrapper">            <div class="code-wrapper">              <div><img src="//img.114yygh.com/static/web/code\_login\_wechat.png" class="code-img">                <div class="code-text"><span class="iconfont icon"></span>微信扫一扫关注                </div>                <div class="code-text"> “快速预约挂号”</div>              </div>              <div class="wechat-code-wrapper"><img                src="//img.114yygh.com/static/web/code\_app.png"                class="code-img">                <div class="code-text"> 扫一扫下载</div>                <div class="code-text"> “预约挂号”APP</div>              </div>            </div>            <div class="slogan">              <div>xxxxxx官方指定平台</div>              <div>快速挂号 安全放心</div>            </div>          </div>        </div>      </el-dialog> |

说明：登录成功用户信息记录cookie

|  |
| --- |
| <script>  import cookie from 'js-cookie'  import Vue from 'vue'  import userInfoApi from '@/api/userInfo'  import smsApi from '@/api/msm'  import hospitalApi from '@/api/hosp'  const defaultDialogAtrr = {    showLoginType: 'phone', // 控制手机登录与微信登录切换    labelTips: '手机号码', // 输入框提示    inputValue: '', // 输入框绑定对象    placeholder: '请输入您的手机号', // 输入框placeholder    maxlength: 11, // 输入框长度控制    loginBtn: '获取验证码', // 登录按钮或获取验证码按钮文本    sending: true,      // 是否可以发送验证码    second: -1,        // 倒计时间  second>0 : 显示倒计时 second=0 ：重新发送 second=-1 ：什么都不显示    clearSmsTime: null  // 倒计时定时任务引用 关闭登录层清除定时任务  }  export default {    data() {      return {        userInfo: {          phone: '',          code: '',          openid: ''        },        dialogUserFormVisible: false,        // 弹出层相关属性        dialogAtrr:defaultDialogAtrr,        name: '' // 用户登录显示的名称      }    },    created() {      this.showInfo()    },    methods: {      // 绑定登录或获取验证码按钮      btnClick() {        // 判断是获取验证码还是登录        if(this.dialogAtrr.loginBtn == '获取验证码') {          this.userInfo.phone = this.dialogAtrr.inputValue          // 获取验证码          this.getCodeFun()        } else {          // 登录          this.login()        }      },      // 绑定登录，点击显示登录层      showLogin() {        this.dialogUserFormVisible = true        // 初始化登录层相关参数        this.dialogAtrr = { ...defaultDialogAtrr }      },      // 登录      login() {        this.userInfo.code = this.dialogAtrr.inputValue        if(this.dialogAtrr.loginBtn == '正在提交...') {          this.$message.error('重复提交')          return;        }        if (this.userInfo.code == '') {          this.$message.error('验证码必须输入')          return;        }        if (this.userInfo.code.length != 6) {          this.$message.error('验证码格式不正确')          return;        }        this.dialogAtrr.loginBtn = '正在提交...'        userInfoApi.login(this.userInfo).then(response => {          console.log(response.data)          // 登录成功 设置cookie          this.setCookies(response.data.name, response.data.token)        }).catch(e => {          this.dialogAtrr.loginBtn = '马上登录'        })      },      setCookies(name, token) {        cookie.set('token', token, { domain: 'localhost' })        cookie.set('name', name, { domain: 'localhost' })        window.location.reload()      },      // 获取验证码      getCodeFun() {        if (!(/^1[34578]\d{9}$/.test(this.userInfo.phone))) {          this.$message.error('手机号码不正确')          return;        }        // 初始化验证码相关属性        this.dialogAtrr.inputValue = ''        this.dialogAtrr.placeholder = '请输入验证码'        this.dialogAtrr.maxlength = 6        this.dialogAtrr.loginBtn = '马上登录'        // 控制重复发送        if (!this.dialogAtrr.sending) return;        // 发送短信验证码        this.timeDown();        this.dialogAtrr.sending = false;        smsApi.sendCode(this.userInfo.phone).then(response => {          this.timeDown();        }).catch(e => {          this.$message.error('发送失败，重新发送')          // 发送失败，回到重新获取验证码界面          this.showLogin()        })      },      // 倒计时      timeDown() {        if(this.clearSmsTime) {          clearInterval(this.clearSmsTime);        }        this.dialogAtrr.second = 60;        this.dialogAtrr.labelTips = '验证码已发送至' + this.userInfo.phone        this.clearSmsTime = setInterval(() => {          --this.dialogAtrr.second;          if (this.dialogAtrr.second < 1) {            clearInterval(this.clearSmsTime);            this.dialogAtrr.sending = true;            this.dialogAtrr.second = 0;          }        }, 1000);      },      // 关闭登录层      closeDialog() {        if(this.clearSmsTime) {          clearInterval(this.clearSmsTime);        }      },      showInfo() {        let token = cookie.get('token')        if (token) {          this.name = cookie.get('name')          console.log(this.name)        }      },      loginMenu(command) {        if('/logout' == command) {          cookie.set('name', '', {domain: 'localhost'})          cookie.set('token', '', {domain: 'localhost'})          //跳转页面          window.location.href = '/'        } else {          window.location.href = command        }      },      handleSelect(item) {        window.location.href = '/hospital/' + item.hoscode      },      weixinLogin() {        this.dialogAtrr.showLoginType = 'weixin'      },      phoneLogin() {        this.dialogAtrr.showLoginType = 'phone'        this.showLogin()      }    }  }  </script> |

### 7.4 头部显示登录状态

#### 7.4.1 获取登录信息

|  |
| --- |
| created() {      this.showInfo()    },      showInfo() {        let token = cookie.get('token')        if (token) {          this.name = cookie.get('name')        }      }, |

#### 7.4.2 登录页面控制

|  |
| --- |
| <!-- 右侧 -->  <div class="right-wrapper">    <span class="v-link clickable">帮助中心</span>    <span v-if="name == ''" class="v-link clickable" @click="showLogin()" id="loginDialog">登录/注册</span>    <el-dropdown v-if="name != ''" @command="loginMenu">          <span class="el-dropdown-link">            {{ name }}<i class="el-icon-arrow-down el-icon--right"></i>          </span>      <el-dropdown-menu class="user-name-wrapper" slot="dropdown">        <el-dropdown-item command="/user">实名认证</el-dropdown-item>        <el-dropdown-item command="/order">挂号订单</el-dropdown-item>        <el-dropdown-item command="/patient">就诊人管理</el-dropdown-item>        <el-dropdown-item command="/logout" divided>退出登录</el-dropdown-item>      </el-dropdown-menu>    </el-dropdown>  </div> |

说明：<**el-dropdown-item command="/user"**>实名认证</**el-dropdown-item**>“command”的值为下拉列表的访问路径，即：实名认证的访问路径，后续会有对应的页面，目前我们将它处理好，方便后续使用

添加下拉列表事件处理：

|  |
| --- |
| loginMenu(command) {    if('/logout' == command) {      cookie.set('name', '', {domain: 'localhost'})      cookie.set('token', '', {domain: 'localhost'})      //跳转页面      window.location.href = '/'    } else {      window.location.href = command    }  }, |

## 登录全局事件

目前登录层在myheader组件里面，登录按钮也在同一个组件里面，我们点击登录，调用showLogin()方法即可

目前的问题是，我们在预约挂号页面，选择科室去挂号时我们需要判断当前是否登录，如果登录可以进入下一个页面；如果没有登录需要显示登录层，那么这个问题怎么解决呢，我们不能直接调用头部登录方法，我们目前的组件是包含在nuxt里面的

问题总是能够解决的，其实很简单，我们可以注册一个全局登录事件，当需要登录层是，我们发送一个登录事件，头部监听登录事件，然后我们触发登录按钮的点击事件即可打开登录层，下面我们来试试

### 8.1 头部注册和监听登录事件

修改myheader.vue组件

1、引入Vue

|  |
| --- |
| **import** Vue **from 'vue'** |

2、注册与监听事件

|  |
| --- |
| mounted() {  *// 注册全局登录事件对象*  ***window***.**loginEvent** = **new** Vue();  *// 监听登录事件*  **loginEvent**.$on(**'loginDialogEvent'**, **function** () {  ***document***.getElementById(**"loginDialog"**).click();  })  *// 触发事件，显示登录层：loginEvent.$emit('loginDialogEvent')*  } |

### 8.2 预约挂号页面调整

修改/pages/hospital/\_hoscode.vue组件

1、引入cookie

|  |
| --- |
| **import cookie from 'js-cookie'** |

2、修改方法

|  |
| --- |
| schedule(depcode) {    // 登录判断    let token = cookie.get('token')    if (!token) {      loginEvent.$emit('loginDialogEvent')      return    }    window.location.href = '/hospital/schedule?hoscode=' + this.hospital.hoscode + "&depcode="+ depcode  }, |

说明：清除cookie，点击科室测试

## myheader.vue完整代码

|  |
| --- |
| <template>      <div class="header-container">          <div class="wrapper">          <!-- logo -->              <div class="left-wrapper v-link selected">                  <img style="width: 50px" width="50" height="50" src="~assets/images/logo.png">                  <span class="text">尚医通 预约挂号统一平台</span>              </div>          <!-- 搜索框 -->          <div class="search-wrapper">              <div class="hospital-search animation-show">                  <el-autocomplete                      class="search-input small"                      prefix-icon="el-icon-search"                      v-model="state"                      :fetch-suggestions="querySearchAsync"                      placeholder="点击输入医院名称"                      @select="handleSelect"                      >                      <span slot="suffix" class="search-btn v-link highlight clickable selected">搜索 </span>                  </el-autocomplete>              </div>          </div>          <!-- 右侧 -->          <!-- 右侧 -->          <div class="right-wrapper">            <span class="v-link clickable">帮助中心</span>            <span v-if="name == ''" class="v-link clickable" @click="showLogin()" id="loginDialog">登录/注册</span>            <el-dropdown v-if="name != ''" @command="loginMenu">                  <span class="el-dropdown-link">                    {{ name }}<i class="el-icon-arrow-down el-icon--right"></i>                  </span>              <el-dropdown-menu class="user-name-wrapper" slot="dropdown">                <el-dropdown-item command="/user">实名认证</el-dropdown-item>                <el-dropdown-item command="/order">挂号订单</el-dropdown-item>                <el-dropdown-item command="/patient">就诊人管理</el-dropdown-item>                <el-dropdown-item command="/logout" divided>退出登录</el-dropdown-item>              </el-dropdown-menu>            </el-dropdown>          </div>          </div>          <!-- 登录弹出层 -->      <el-dialog :visible.sync="dialogUserFormVisible" style="text-align: left;" top="50px" :append-to-body="true"  width="960px" @close="closeDialog()">        <div class="container">          <!-- 手机登录 #start -->          <div class="operate-view" v-if="dialogAtrr.showLoginType === 'phone'">            <div class="wrapper" style="width: 100%">              <div class="mobile-wrapper" style="position: static;width: 70%">                <span class="title">{{ dialogAtrr.labelTips }}</span>                <el-form>                  <el-form-item>                    <el-input v-model="dialogAtrr.inputValue" :placeholder="dialogAtrr.placeholder" :maxlength="dialogAtrr.maxlength" class="input v-input">                      <span slot="suffix" class="sendText v-link" v-if="dialogAtrr.second > 0">{{ dialogAtrr.second }}s </span>                      <span slot="suffix" class="sendText v-link highlight clickable selected" v-if="dialogAtrr.second == 0" @click="getCodeFun()">重新发送 </span>                    </el-input>                  </el-form-item>                </el-form>                <div class="send-button v-button" @click="btnClick()"> {{ dialogAtrr.loginBtn }}</div>              </div>              <div class="bottom">                <div class="wechat-wrapper" @click="weixinLogin()"><span                  class="iconfont icon"></span></div>                <span class="third-text"> 第三方账号登录 </span></div>            </div>          </div>          <!-- 手机登录 #end -->          <!-- 微信登录 #start -->          <div class="operate-view"  v-if="dialogAtrr.showLoginType === 'weixin'" >            <div class="wrapper wechat" style="height: 400px">              <div>                <div id="weixinLogin"></div>              </div>              <div class="bottom wechat" style="margin-top: -80px;">                <div class="phone-container">                  <div class="phone-wrapper"  @click="phoneLogin()"><span                    class="iconfont icon"></span></div>                  <span class="third-text"> 手机短信验证码登录 </span></div>              </div>            </div>          </div>          <!-- 微信登录 #end -->          <div class="info-wrapper">            <div class="code-wrapper">              <div><img src="//img.114yygh.com/static/web/code\_login\_wechat.png" class="code-img">                <div class="code-text"><span class="iconfont icon"></span>微信扫一扫关注                </div>                <div class="code-text"> “快速预约挂号”</div>              </div>              <div class="wechat-code-wrapper"><img                src="//img.114yygh.com/static/web/code\_app.png"                class="code-img">                <div class="code-text"> 扫一扫下载</div>                <div class="code-text"> “预约挂号”APP</div>              </div>            </div>            <div class="slogan">              <div>xxxxxx官方指定平台</div>              <div>快速挂号 安全放心</div>            </div>          </div>        </div>      </el-dialog>      </div>  </template>  <script>  import cookie from 'js-cookie'  import Vue from 'vue'  import userInfoApi from '@/api/userInfo'  import smsApi from '@/api/msm'  import hospitalApi from '@/api/hosp'  const defaultDialogAtrr = {    showLoginType: 'phone', // 控制手机登录与微信登录切换    labelTips: '手机号码', // 输入框提示    inputValue: '', // 输入框绑定对象    placeholder: '请输入您的手机号', // 输入框placeholder    maxlength: 11, // 输入框长度控制    loginBtn: '获取验证码', // 登录按钮或获取验证码按钮文本    sending: true,      // 是否可以发送验证码    second: -1,        // 倒计时间  second>0 : 显示倒计时 second=0 ：重新发送 second=-1 ：什么都不显示    clearSmsTime: null  // 倒计时定时任务引用 关闭登录层清除定时任务  }  export default {    data() {      return {        userInfo: {          phone: '',          code: '',          openid: ''        },        dialogUserFormVisible: false,        // 弹出层相关属性        dialogAtrr:defaultDialogAtrr,        name: '' // 用户登录显示的名称      }    },    created() {      this.showInfo()    },    mounted() {      // 注册全局登录事件对象      window.loginEvent = new Vue();      // 监听登录事件      loginEvent.$on('loginDialogEvent', function () {        document.getElementById("loginDialog").click();      })      // 触发事件，显示登录层：loginEvent.$emit('loginDialogEvent')    },    methods: {      // 绑定登录或获取验证码按钮      btnClick() {        // 判断是获取验证码还是登录        if(this.dialogAtrr.loginBtn == '获取验证码') {          this.userInfo.phone = this.dialogAtrr.inputValue          // 获取验证码          this.getCodeFun()        } else {          // 登录          this.login()        }      },      // 绑定登录，点击显示登录层      showLogin() {        this.dialogUserFormVisible = true        // 初始化登录层相关参数        this.dialogAtrr = { ...defaultDialogAtrr }      },      // 登录      login() {        this.userInfo.code = this.dialogAtrr.inputValue        if(this.dialogAtrr.loginBtn == '正在提交...') {          this.$message.error('重复提交')          return;        }        if (this.userInfo.code == '') {          this.$message.error('验证码必须输入')          return;        }        if (this.userInfo.code.length != 6) {          this.$message.error('验证码格式不正确')          return;        }        this.dialogAtrr.loginBtn = '正在提交...'        userInfoApi.login(this.userInfo).then(response => {          console.log(response.data)          // 登录成功 设置cookie          this.setCookies(response.data.name, response.data.token)        }).catch(e => {          this.dialogAtrr.loginBtn = '马上登录'        })      },      setCookies(name, token) {        cookie.set('token', token, { domain: 'localhost' })        cookie.set('name', name, { domain: 'localhost' })        window.location.reload()      },      // 获取验证码      getCodeFun() {        if (!(/^1[34578]\d{9}$/.test(this.userInfo.phone))) {          this.$message.error('手机号码不正确')          return;        }        // 初始化验证码相关属性        this.dialogAtrr.inputValue = ''        this.dialogAtrr.placeholder = '请输入验证码'        this.dialogAtrr.maxlength = 6        this.dialogAtrr.loginBtn = '马上登录'        // 控制重复发送        if (!this.dialogAtrr.sending) return;        // 发送短信验证码        this.timeDown();        this.dialogAtrr.sending = false;        smsApi.sendCode(this.userInfo.phone).then(response => {          this.timeDown();        }).catch(e => {          this.$message.error('发送失败，重新发送')          // 发送失败，回到重新获取验证码界面          this.showLogin()        })      },      // 倒计时      timeDown() {        if(this.clearSmsTime) {          clearInterval(this.clearSmsTime);        }        this.dialogAtrr.second = 60;        this.dialogAtrr.labelTips = '验证码已发送至' + this.userInfo.phone        this.clearSmsTime = setInterval(() => {          --this.dialogAtrr.second;          if (this.dialogAtrr.second < 1) {            clearInterval(this.clearSmsTime);            this.dialogAtrr.sending = true;            this.dialogAtrr.second = 0;          }        }, 1000);      },      // 关闭登录层      closeDialog() {        if(this.clearSmsTime) {          clearInterval(this.clearSmsTime);        }      },      showInfo() {        let token = cookie.get('token')        if (token) {          this.name = cookie.get('name')          console.log(this.name)        }      },      loginMenu(command) {        if('/logout' == command) {          cookie.set('name', '', {domain: 'localhost'})          cookie.set('token', '', {domain: 'localhost'})          //跳转页面          window.location.href = '/'        } else {          window.location.href = command        }      },      handleSelect(item) {        window.location.href = '/hospital/' + item.hoscode      },      weixinLogin() {        this.dialogAtrr.showLoginType = 'weixin'      },      phoneLogin() {        this.dialogAtrr.showLoginType = 'phone'        this.showLogin()      }    }  }  </script> |

# 用户认证与网关整合

思路：

1. 所有请求都会经过服务网关，服务网关对外暴露服务，在网关进行统一用户认证；
2. 既然要在网关进行用户认证，网关得知道对哪些url进行认证，所以我们得对ur制定规则
3. Api接口异步请求的，我们采取url规则匹配，如：/api/\*\*/auth/\*\*，如凡是满足该规则的都必须用户认证

## 1、调整server-gateway模块

### 1.1在服务网关添加fillter

|  |
| --- |
| @Override  **public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  ServerHttpRequest request = exchange.getRequest();  String path = request.getURI().getPath();  System.***out***.println(**"==="**+path);  *//内部服务接口，不允许外部访问*  **if**(**antPathMatcher**.match(**"/\*\*/inner/\*\*"**, path)) {  ServerHttpResponse response = exchange.getResponse();  **return** out(response, ResultCodeEnum.***PERMISSION***);  }  Long userId = **this**.getUserId(request);  *//api接口，异步请求，校验用户必须登录*  **if**(**antPathMatcher**.match(**"/api/\*\*/auth/\*\*"**, path)) {  **if**(StringUtils.*isEmpty*(userId)) {  ServerHttpResponse response = exchange.getResponse();  **return** out(response, ResultCodeEnum.***LOGIN\_AUTH***);  }  }  **return** chain.filter(exchange);  } |

### 1.2在服务网关中判断用户登录状态

在网关中如何获取用户信息：

1. 我们统一从header头信息中获取

如何判断用户信息合法：

登录时我们返回用户token，在服务网关中获取到token后，我在到redis中去查看用户id，如何用户id存在，则token合法，否则不合法

### 1.3 取用户信息

|  |
| --- |
| */\*\**  *\* 获取当前登录用户id*  *\** ***@param request***  *\** ***@return***  *\*/*  **private** Long getUserId(ServerHttpRequest request) {  String token = **""**;  List<String> tokenList = request.getHeaders().get(**"token"**);  **if**(**null** != tokenList) {  token = tokenList.get(0);  }  **if**(!StringUtils.*isEmpty*(token)) {  **return** JwtHelper.*getUserId*(token);  }  **return null**;  } |

### 1.3 网关filter完整代码

|  |
| --- |
| */\*\**  *\* <p>*  *\* 全局Filter，统一处理会员登录与外部不允许访问的服务*  *\* </p>*  *\**  *\** ***@author*** *qy*  *\** ***@since*** *2019-11-21*  *\*/*  @Component  **public class** AuthGlobalFilter **implements** GlobalFilter, Ordered {  **private** AntPathMatcher **antPathMatcher** = **new** AntPathMatcher();  @Override  **public** Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  ServerHttpRequest request = exchange.getRequest();  String path = request.getURI().getPath();  System.***out***.println(**"==="**+path);  *//内部服务接口，不允许外部访问*  **if**(**antPathMatcher**.match(**"/\*\*/inner/\*\*"**, path)) {  ServerHttpResponse response = exchange.getResponse();  **return** out(response, ResultCodeEnum.***PERMISSION***);  }  Long userId = **this**.getUserId(request);  *//api接口，异步请求，校验用户必须登录*  **if**(**antPathMatcher**.match(**"/api/\*\*/auth/\*\*"**, path)) {  **if**(StringUtils.*isEmpty*(userId)) {  ServerHttpResponse response = exchange.getResponse();  **return** out(response, ResultCodeEnum.***LOGIN\_AUTH***);  }  }  **return** chain.filter(exchange);  }  @Override  **public int** getOrder() {  **return** 0;  }  */\*\**  *\* api接口鉴权失败返回数据*  *\** ***@param response***  *\** ***@return***  *\*/*  **private** Mono<Void> out(ServerHttpResponse response, ResultCodeEnum resultCodeEnum) {  Result result = Result.*build*(**null**, resultCodeEnum);  **byte**[] bits = JSONObject.*toJSONString*(result).getBytes(StandardCharsets.***UTF\_8***);  DataBuffer buffer = response.bufferFactory().wrap(bits);  *//指定编码，否则在浏览器中会中文乱码*  response.getHeaders().add(**"Content-Type"**, **"application/json;charset=UTF-8"**);  **return** response.writeWith(Mono.*just*(buffer));  }  */\*\**  *\* 获取当前登录用户id*  *\** ***@param request***  *\** ***@return***  *\*/*  **private** Long getUserId(ServerHttpRequest request) {  String token = **""**;  List<String> tokenList = request.getHeaders().get(**"token"**);  **if**(**null** != tokenList) {  token = tokenList.get(0);  }  **if**(!StringUtils.*isEmpty*(token)) {  **return** JwtHelper.*getUserId*(token);  }  **return null**;  }  } |

## 2、调整前端yygh-site

请求服务器端接口时我们默认带上token，需要登录的接口如果token没有或者token过期，服务器端会返回208状态，然后发送登录事件打开登录弹出层登录

修改utils/request.js文件

|  |
| --- |
| import axios from 'axios'  import { MessageBox, Message } from 'element-ui'  import cookie from 'js-cookie'  // 创建axios实例  const service = axios.create({      baseURL: 'http://localhost',      timeout: 15000 // 请求超时时间  })  // http request 拦截器  service.interceptors.request.use(      config => {      // token 先不处理，后续使用时在完善      //判断cookie是否有token值      if(cookie.get('token')) {          //token值放到cookie里面          config.headers['token']=cookie.get('token')      }      return config  },    err => {      return Promise.reject(err)  })  // http response 拦截器  service.interceptors.response.use(      response => {          //状态码是208          if(response.data.code === 208) {              //弹出登录输入框              loginEvent.$emit('loginDialogEvent')              return          } else {              if (response.data.code !== 200) {                  Message({                      message: response.data.message,                      type: 'error',                      duration: 5 \* 1000                  })                  return Promise.reject(response.data)              } else {                  return response.data              }          }      },      error => {          return Promise.reject(error.response)  })  export default service |