# EGO前台商城用户登录

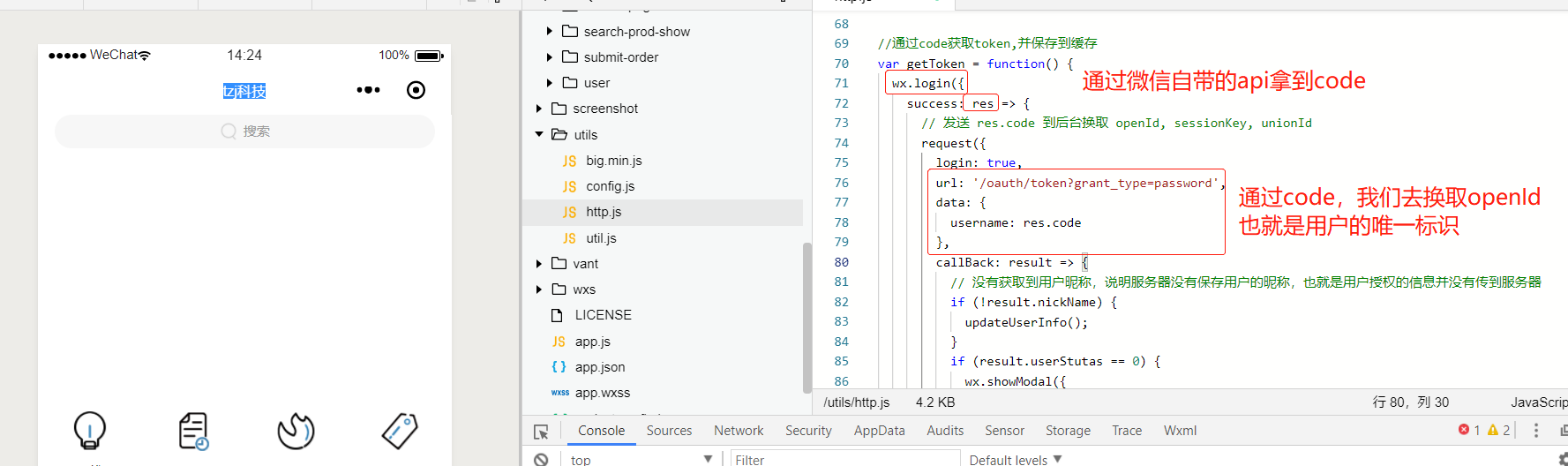
## 商城开发

前台商城是借助微信小程序的载体，所以登录时走微信的那一套流程，这也是我们第一次接触到第三方的集成

## 微信获取openId的流程

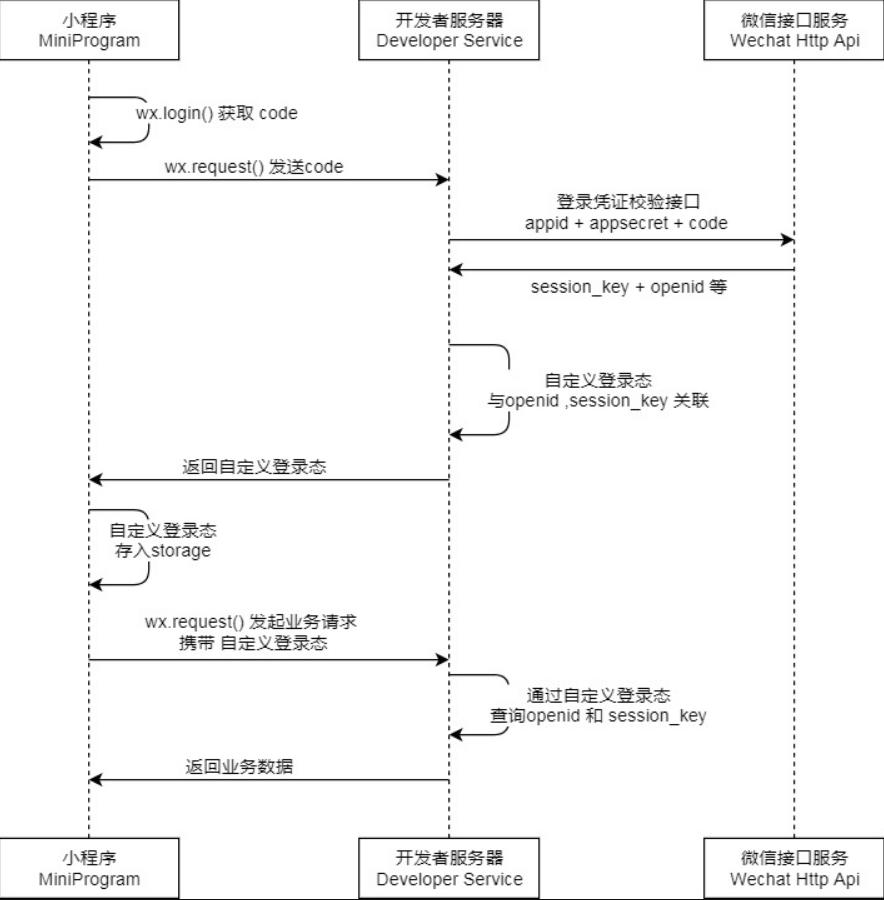
wxd3b90c3acfea33be

在第一天启动前台商城的时候，我们发现wx.login会拿到一个code，在根据code去换取openId，就是用户的唯一标识



具体的小程序登录流程图微信官方给出了

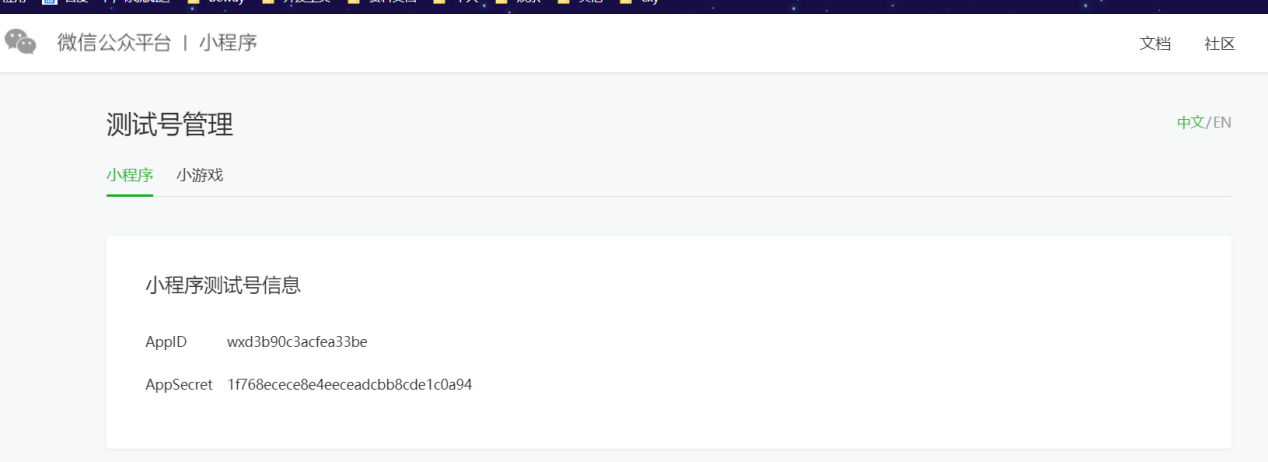
<https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/login.html>



## 申请微信小程序测试号（appID和appsecret）

<https://developers.weixin.qq.com/sandbox>

如果没有获取到测试号，就先点登录社区 再返回刷新



## 查看官网文档，拿到请求地址

<https://developers.weixin.qq.com/miniprogram/dev/api-backend/open-api/login/auth.code2Session.html>



我们在授权服务里面发送请求

GET请求

[https://api.weixin.qq.com/sns/jscode2session?appid=](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[APPID](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[&secret=](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[SECRET](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[&js\_code=](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[JSCODE](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)**[&grant\_type=authorization\_code](https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code)

参数说明：

|  |
| --- |
| appid：你的测试号的appid  secret:你的测试号的appsecret  js\_code:微信小程序通过wx.login拿到的code  grant\_type：固定填写authorization\_code |

## 修改auth-server

### 修改远端auth-server/auth-server-dev.yml的配置文件

|  |
| --- |
| wechat: #测试的appId和appSecret  appid: wxd3b90c3acfea33be  secret: 1f768ecece8e4eeceadcbb8cde1c0a94 |

### 在ego-common中添加User实体类（逆向生成）

### 在ego-common中添加UserMapper（其他微服务也可能用到）

|  |
| --- |
| public interface UserMapper extends BaseMapper<User> {  } |

### 修改auth-server启动类

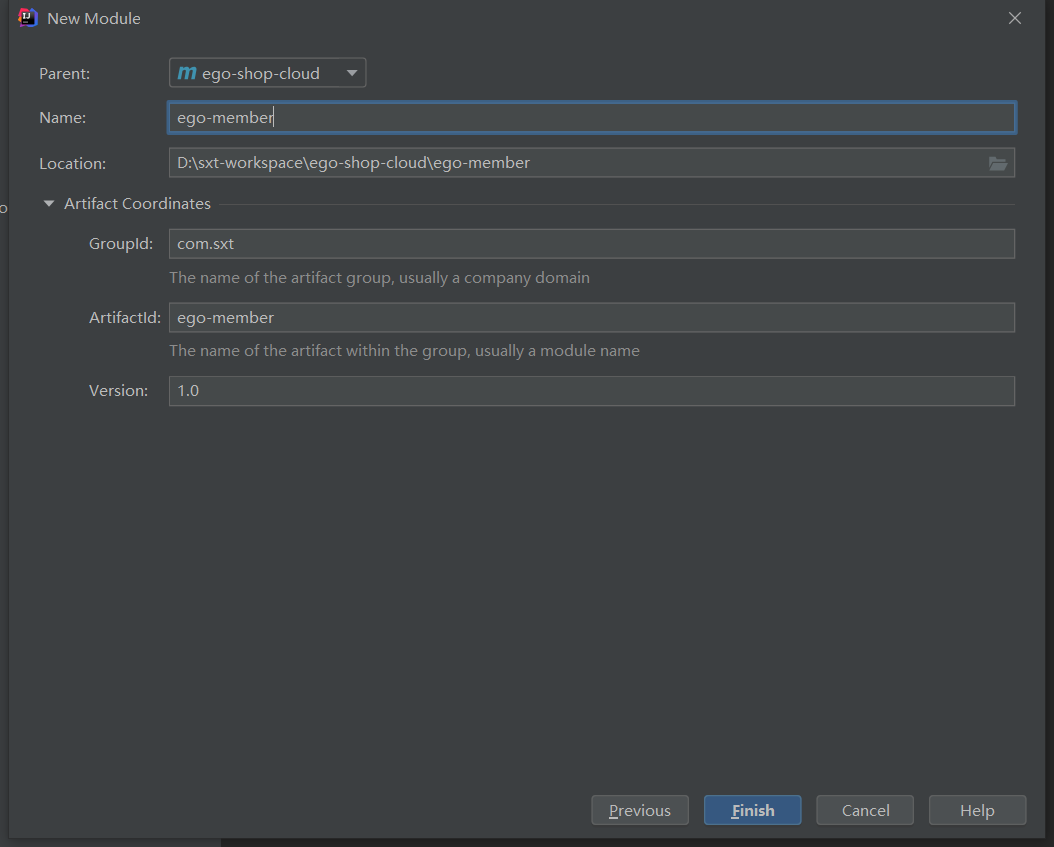
|  |
| --- |
| /\*\*  \* 添加发请求  \*  \* @return  \*/  @Bean  public RestTemplate restTemplate() {  return new RestTemplate();  } |

### 修改auth-server启动类

|  |
| --- |
| @Service  public class UserDetailServiceImpl implements UserDetailsService {  **@Value("${wechat.appid}")**  **private String appId;**  **@Value("${wechat.secret}")**  **private String appSecret;**  **//请求的openId地址**  **private String WECHAT\_AUTH\_URL = "https://api.weixin.qq.com/sns/jscode2session?appid=%s&secret=%s&js\_code=%s&grant\_type=authorization\_code";**  @Autowired  private RestTemplate restTemplate;  @Autowired  private UserDetailMapper userDetailMapper;  @Autowired  private SysUserDetailMapper sysUserDetailMapper;  @Autowired  private UserMapper userMapper;  @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = requestAttributes.getRequest();  String loginType = request.getHeader(AuthConstant.LOGIN\_TYPE);  if (StringUtils.isEmpty(loginType)) {  return null;  }  switch (loginType) {  **case AuthConstant.USER\_TYPE:**  **//如果是前台用户 拿到code请求，发请求**  **String wxStr = restTemplate.getForObject(String.format(WECHAT\_AUTH\_URL, new Object[]{this.appId, this.appSecret, username}), String.class);**  **JSONObject jsonObject = JSONUtil.parseObj(wxStr);**  **//拿到openid**  **String openId = jsonObject.get("openid", String.class);**  **if (StringUtils.isEmpty(openId)) {**  **//如果openId为空 直接返回null**  **return null;**  **}**  **//查询数据库**  **UserDetail userDetail = userDetailMapper.selectOne(new LambdaQueryWrapper<UserDetail>()**  **.eq(UserDetail::getUserId, openId)**  **.eq(UserDetail::getStatus, 1)**  **);**  **if (ObjectUtils.isEmpty(userDetail)) {**  **//如果用户为空，创建用户 插入数据库 返回**  **userDetail = createUserAndSave(openId);**  **}**  **return userDetail;**  case AuthConstant.SYS\_USER\_TYPE:  //如果是后台用户，要查询权限  SysUserDetail sysUserDetail = sysUserDetailMapper.selectOne(new LambdaQueryWrapper<SysUserDetail>()  .eq(SysUserDetail::getUsername, username)  .eq(SysUserDetail::getStatus, 1)  );  if (!ObjectUtils.isEmpty(sysUserDetail)) {  //查询权限  List<String> authorization = getUserPermission(sysUserDetail.getUserId());  if (!ObjectUtils.isEmpty(authorization)) {  sysUserDetail.setPermissions(authorization);  }  }  return sysUserDetail;  default:  }  return null;  }  **private UserDetail createUserAndSave(String openId) {**  **//创建一个用户插入数据库**  **User user = new User();**  **user.setUserId(openId);**  **user.setUserRegtime(new Date());**  **user.setModifyTime(new Date());**  **//设置名称，先随机给一个**  **user.setNickName("sxt" + UUID.randomUUID().toString().substring(0, 11));**  **user.setStatus(1);**  **int insert = userMapper.insert(user);**  **if (insert > 0) {**  **UserDetail userDetail = new UserDetail();**  **userDetail.setUserId(openId);**  **userDetail.setStatus(1);**  **return userDetail;**  **}**  **throw new RuntimeException("插入数据库失败");**  **}**  /\*\*  \* 查询权限的方法  \*  \* @param userId  \* @return  \*/  private List<String> getUserPermission(Long userId) {  HashSet<String> actualPermissions = new HashSet<>();  //得到权限  List<String> permission = sysUserDetailMapper.getPermissionByUserId(userId);  if (ObjectUtils.isEmpty(permission)) {  return Collections.emptyList();  }  //处理逗号分隔  permission.forEach(p -> {  String[] complexPermission = p.split(",");  for (String s : complexPermission) {  actualPermissions.add(s);  }  });  return new ArrayList<>(actualPermissions);  }  } |

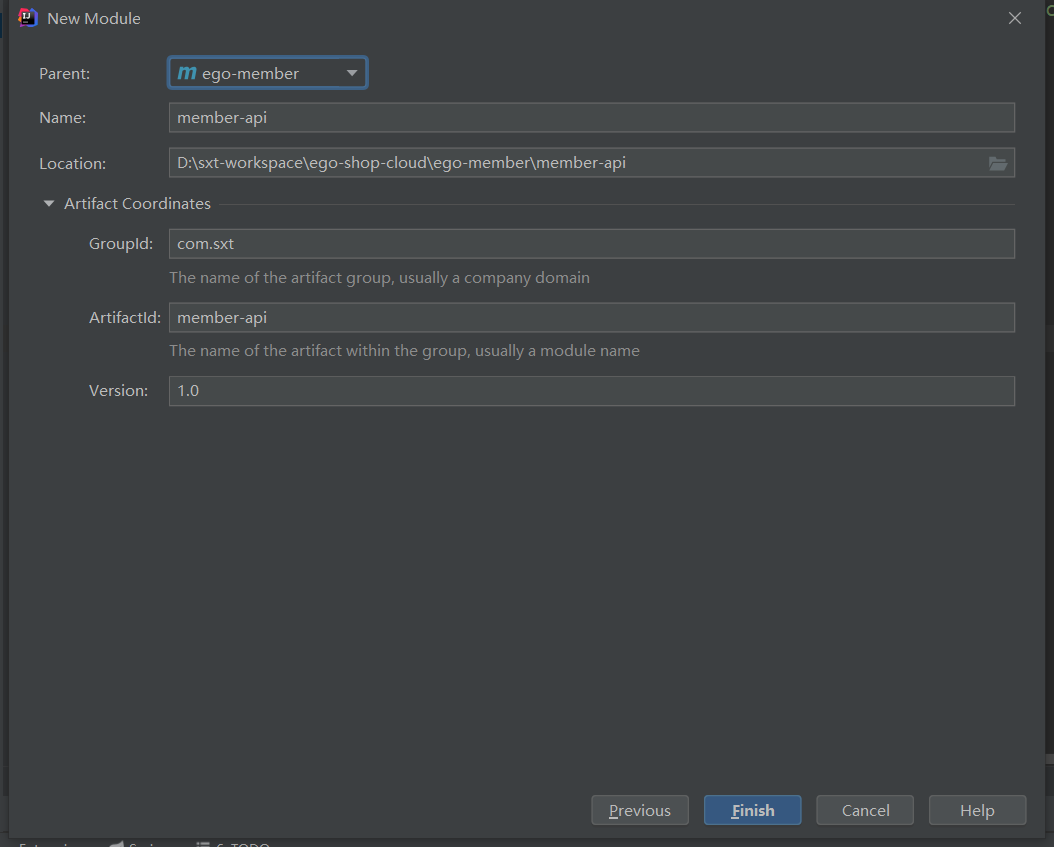
## 在ego-shop-cloud下面新建用户模块

### 修改一个maven模块ego-member



### 在ego-member下面新建member-api

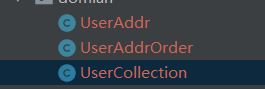
#### 新建模块



#### 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>ego-member</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>member-api</artifactId>  <description>ego商城会员服务api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>product-api</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  </project> |

#### 逆向生成用户相关的表



#### 创建UserCollectFeign

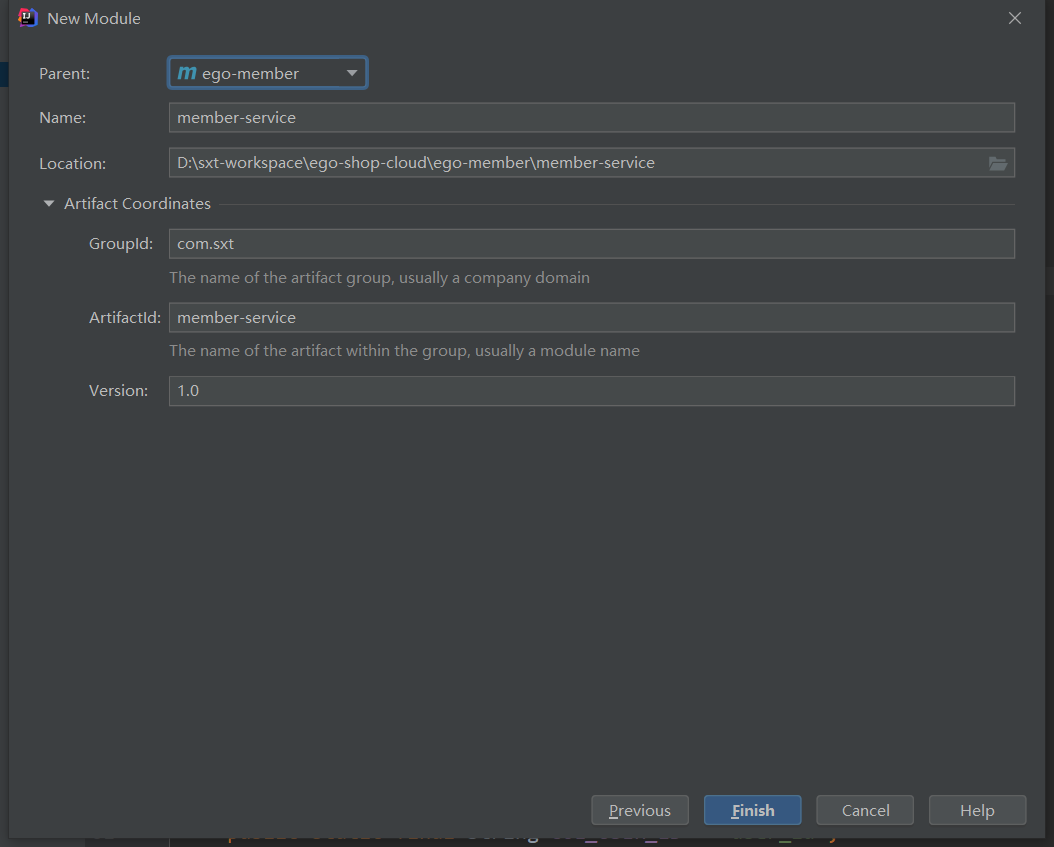
|  |
| --- |
| @FeignClient(value = "search-service", fallback = UserCollectSearchFeignHystrix.class)  public interface UserCollectSearchFeign {  /\*\*  \* 远程的搜索服务 拿到商品对象  \*  \* @param prodIds  \* @return  \*/  @PostMapping("search/getProd")  List<ProductEs> getUserCollectByProdIds(@RequestBody List<Long> prodIds);  } |

#### 创建UserCollectFeignHystrix熔断

|  |
| --- |
| @Component  @Slf4j  public class UserCollectSearchFeignHystrix implements UserCollectSearchFeign {  /\*\*  \* 远程的搜索服务 拿到商品对象  \*  \* @param prodIds  \* @return  \*/  @Override  public List<ProductEs> getUserCollectByProdIds(List<Long> prodIds) {  log.error("远程调用失败了");  return Collections.emptyList();  }  } |

### 在ego-member下面新建member-service

#### 新建模块



#### 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>ego-member</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>member-service</artifactId>  <description>ego商城的会员服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>member-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  <!-- spring的切面编程-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  <!-- 引入druid starter-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 创建启动类

|  |
| --- |
| @SpringBootApplication  @EnableEurekaClient  @EnableFeignClients  @EnableCaching  public class MemberServiceApplication {  public static void main(String[] args) {  SpringApplication.run(MemberServiceApplication.class, args);  }  } |

#### 创建bootstrap.yml文件

|  |
| --- |
| spring:  application:  name: member-service  cloud:  config:  discovery:  service-id: config-server  enabled: true  name: member-service  label: master  profile: dev  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka/}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-renewal-interval-in-seconds: 10  lease-expiration-duration-in-seconds: 30 |

#### 创建远端member-service/member-service-dev.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8085}  spring:  datasource:  druid:  url: jdbc:mysql://localhost:3306/ego-shop?useSSL=false&serverTimezone=GMT%2B8  username: root  password: root  driver-class-name: com.mysql.cj.jdbc.Driver  max-active: 30  min-idle: 3  initial-size: 3  validation-query: select 1  max-wait: 5000  stat-view-servlet:  enabled: true  login-username: admin  login-password: admin  allow:  deny:  url-pattern:  - /druid/\*  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*'  feign:  hystrix:  enabled: false  ribbon:  ReadTimeout: 18000  ConnectTimeout: 18000  hystrix: #hystrix的全局控制  command:  default: #default是全局控制，也可以换成单个方法控制，把default换成方法名即可  fallback:  isolation:  semaphore:  maxConcurrentRequests: 1000  circuitBreaker:  enabled: true #开启断路器  requestVolumeThreshold: 20 #失败次数（阀值）  sleepWindowInMilliseconds: 30000 #窗口时间  errorThresholdPercentage: 60 #失败率  execution:  isolation:  strategy: SEMAPHORE #使用信号量隔离方式  thread:  timeoutInMilliseconds: 5000  semaphore:  maxConcurrentRequests: 1000 #最大并发量  mybatis-plus:  mapper-locations: classpath:mapper/\*\*/\*.xml  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  resources:  url: http://192.168.226.129  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 前台用户管理系统接口  description: 实现前台用户管理系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble |

#### 创建UserController

|  |
| --- |
| @RestController  @Api(tags = "会员的管理")  public class UserController {  @Autowired  private UserService userService;  @Autowired  private StringRedisTemplate redisTemplate;  @Autowired  private RabbitTemplate rabbitTemplate;  @RequestMapping("/p/user/setUserInfo")  @ApiOperation("设置用户信息")  public ResponseEntity<Void> setUserInfo(@RequestBody @Validated User user) {  //拿到user  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  user.setUserId(openId);  userService.updateById(user);  return ResponseEntity.ok().build();  }  @GetMapping("getUserInfoById")  @ApiOperation("获取用户的信息")  public User getUserInfo(String openId) {  return userService.getById(openId);  }  } |

#### 创建UserMapper2

这里因为auth-server中有一个UserMapper了，所以我们改一个名字，只要继承的没问题

|  |
| --- |
| public interface **UserMapper2 extends BaseMapper<User>** {  } |

#### 创建UserAddrController

|  |
| --- |
| @RestController  @Api(tags = "用户地址管理接口")  public class UserAddrController {  @Autowired  private UserAddrService userAddrService;  @GetMapping({"/p/address/list"})  @ApiOperation("分列查询列表")  public ResponseEntity<List<UserAddr>> list() {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  List<UserAddr> userAddrs = this.userAddrService.findUserAdderss(openId);  return ResponseEntity.ok(userAddrs);  }  @PostMapping({"/p/address/addAddr"})  @ApiOperation("新增一个UserAddr")  public ResponseEntity<Void> save(@RequestBody @Validated UserAddr entity) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  entity.setUserId(openId);  this.userAddrService.save(entity);  return ResponseEntity.ok().build();  }  @GetMapping({"/p/address/addrInfo/{id}"})  @ApiOperation("查询UserAddr的详情")  public ResponseEntity<UserAddr> info(@PathVariable("id") Long id) {  UserAddr entity = this.userAddrService.getById(id);  return ResponseEntity.ok(entity);  }  @DeleteMapping({"/p/address/deleteAddr/{id}"})  @ApiOperation("删除UserAddr")  public ResponseEntity<Void> delete(@PathVariable("id") Long id) {  this.userAddrService.removeById(id);  return ResponseEntity.ok().build();  }  @PutMapping({"/p/address/updateAddr"})  @ApiOperation("根据id修改UserAddr")  public ResponseEntity<Void> update(@RequestBody @Validated UserAddr entity) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  entity.setUserId(openId);  this.userAddrService.updateById(entity);  return ResponseEntity.ok().build();  }  @PutMapping({"/p/address/defaultAddr/{id}"})  @ApiOperation("设置默认的地址")  public ResponseEntity<Void> setDefaultAddr(@PathVariable("id") Long id) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  this.userAddrService.setDefaultAddr(openId, id);  return ResponseEntity.ok().build();  }  } |

#### 修改UserAddrService

|  |
| --- |
| public interface UserAddrService extends IService<UserAddr> {  /\*\*  \* 通过用户的id查询用户的地址  \*  \* @param userId  \* @return  \*/  List<UserAddr> findUserAdderss(String userId);  /\*\*  \* 将id设置为user的默认收货地址  \*  \* @param userId  \* @param id  \*/  void setDefaultAddr(String userId, Long id);  /\*\*  \* 设置用户的默认的收货地址  \*  \* @param userId  \* @return  \*/  UserAddr getUserDefaultAddr(String userId);  } |

#### 修改UserAddrServiceImpl

|  |
| --- |
| @Service  @Slf4j  @CacheConfig(cacheNames = "com.sxt.service.impl.UserAddrServiceImpl")  public class UserAddrServiceImpl extends ServiceImpl<UserAddrMapper, UserAddr> implements UserAddrService {  @Autowired  private UserAddrMapper userAddrMapper;  @Autowired  private StringRedisTemplate redisTemplate;  /\*\*  \* 通过用户的id查询用户的地址  \*  \* @param userId  \* @return  \*/  @Override  @Cacheable(key = "#userId")  public List<UserAddr> findUserAdderss(String userId) {  List<UserAddr> userAddrs = userAddrMapper.selectList(new LambdaQueryWrapper<UserAddr>()  .eq(UserAddr::getUserId, userId)  .eq(UserAddr::getStatus, 1)  .orderByAsc(UserAddr::getUpdateTime)  );  return userAddrs;  }  /\*\*  \* 将选中的id设置为user的默认收货地址  \*  \* @param userId  \* @param id  \*/  @Override  @Transactional  @CacheEvict(key = "#userId")  public void setDefaultAddr(String userId, Long id) {  //查询用户的默认地址  UserAddr userAddr = getById(id);  if (ObjectUtils.isEmpty(userAddr) || !userAddr.getUserId().equals(userId)) {  throw new RuntimeException("用户地址不存在，或者用户的id不匹配");  }  log.info("设置用户{}的默认地址,原地址id为{}", userId, id);  //更新地址  UserAddr updateUserAddr = new UserAddr();  //是否为默认地址 0不是 1是  updateUserAddr.setCommonAddr(0);  //把原来的默认地址设为0  userAddrMapper.update(updateUserAddr, new LambdaQueryWrapper<UserAddr>().  eq(UserAddr::getUserId, userId).  eq(UserAddr::getCommonAddr, 1));  //把用户新选的地址 设为默认地址  userAddr.setCommonAddr(1);  userAddr.setUpdateTime(new Date());  updateById(userAddr);  }  /\*\*  \* 设置用户的默认的收货地址  \*  \* @param userId  \* @return  \*/  @Override  public UserAddr getUserDefaultAddr(String userId) {  UserAddr userAddr = userAddrMapper.selectOne(new LambdaQueryWrapper<UserAddr>().eq(UserAddr::getUserId, userId).eq(UserAddr::getCommonAddr, 1));  return userAddr;  }  @Override  @CacheEvict(key = "#entity.userId")  public boolean save(UserAddr entity) {  log.info("新增一个用户地址{}", JSONUtil.toJsonStr(entity));  entity.setCreateTime(new Date());  entity.setUpdateTime(new Date());  entity.setStatus(1);  //查询有没有默认的地址，如果没有，就设置为默认的  Integer count = userAddrMapper.selectCount(new LambdaQueryWrapper<UserAddr>()  .eq(UserAddr::getUserId, entity.getUserId())  .eq(UserAddr::getCommonAddr, 0)  );  if (count == null || count == 0) {  entity.setCommonAddr(0);  }  return super.save(entity);  }  @Override  public boolean removeById(Serializable id) {  log.info("删除用户{}的收货地址", id);  UserAddr userAddr = getById(id);  if (userAddr == null) {  throw new IllegalArgumentException("删除用户的地址为空");  }  String userId = userAddr.getUserId();  redisTemplate.delete("com.sxt.service.impl.UserAddrServiceImpl::" + userId);  return super.removeById(id);  }  @Override  @CacheEvict(key = "#entity.userId")  public boolean updateById(UserAddr entity) {  entity.setUpdateTime(new Date());  return super.updateById(entity);  }  } |

#### 添加UserCollectController

|  |
| --- |
| @RestController  @Api(tags = "会员收藏接口")  public class UserCollectController {  @Autowired  private UserCollectionService userCollectionService;  /\*\*  \* 查询收藏的数量  \*  \* @return  \*/  @GetMapping("/p/user/collection/count")  @ApiOperation("获取用户收藏的数量")  public ResponseEntity<Integer> getCollectionCount() {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  Integer totalCount = userCollectionService.getCollectionCount(openId);  return ResponseEntity.ok(totalCount);  }  /\*\*  \* 查询用户收藏的商品，根据商品id 使用es查询  \*  \* @param page  \* @return  \*/  @GetMapping("/p/user/collection/prods")  @ApiOperation("查询用户收藏的藏品")  public ResponseEntity<Page<ProdVo>> loadCollections(Page<UserCollection> page) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  // 查询商品的ids  Page<ProdVo> prodIdList = userCollectionService.getCollectionProdList(openId, page);  // 利用商品的ids 查询搜索系统得到商品的数据  if (prodIdList.getRecords().isEmpty()) {  page.setTotal(prodIdList.getTotal());  return ResponseEntity.ok(null);  }  return ResponseEntity.ok(prodIdList);  }  @GetMapping({"/p/user/collection/isCollection"})  @ApiOperation("是否收藏")  public ResponseEntity<Boolean> isCollection(Long prodId) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  Boolean isCollection = this.userCollectionService.isCollection(openId, prodId);  return ResponseEntity.ok(isCollection);  }  @PostMapping({"/p/user/collection/addOrCancel"})  @ApiOperation("收藏或取消收藏商品")  public ResponseEntity<Void> addOrCancel(@RequestBody Long prodId) {  String openId = SecurityContextHolder.getContext().getAuthentication().getPrincipal().toString();  this.userCollectionService.addOrCancel(openId, prodId);  return ResponseEntity.ok().build();  }  } |

#### 修改UserCollectService

|  |
| --- |
| public interface UserCollectionService extends IService<UserCollection> {  /\*\*  \* 查询用户收藏的数量  \*  \* @param openId  \* @return  \*/  Integer getCollectionCount(String openId);  /\*\*  \* 分页查询用户收藏  \*  \* @param page  \* @param openId  \* @return  \*/  IPage<ProductEs> loadUserCollectPage(Page<UserCollection> page, String openId);  /\*\*  \* 用户是否收藏过该商品  \*  \* @param openId  \* @param prodId  \* @return  \*/  Boolean isCollection(String openId, Long prodId);  /\*\*  \* 收藏或取消  \*  \* @param openId  \* @param prodId  \*/  void addOrCancel(String openId, Long prodId);  } |

#### 修改UserCollectServiceImpl

|  |
| --- |
| @Service  public class UserCollectionServiceImpl extends ServiceImpl<UserCollectionMapper, UserCollection> implements UserCollectionService {  @Autowired  private UserCollectionMapper userCollectionMapper;  @Autowired  private UserCollectFeign userCollectFeign;  /\*\*  \* 查询用户收藏的数量  \*  \* @param openId  \* @return  \*/  @Override  public Integer getCollectionCount(String openId) {  Integer count = userCollectionMapper.selectCount(new LambdaQueryWrapper<UserCollection>()  .eq(UserCollection::getUserId, openId)  );  return count;  }  /\*\*  \* 分页查询用户收藏  \* 现在有userId了  \* List<Long> prodIds 10  \* 在远程调用搜索服务 查询List<ProductEs>  \* 在组装数据  \*  \* @param page  \* @param openId  \* @return  \*/  @Override  public IPage<ProductEs> loadUserCollectPage(Page<UserCollection> page, String openId) {  page.addOrder(OrderItem.desc("create\_time"));  IPage<ProductEs> productEsIPage = new Page<>();  IPage<UserCollection> userCollectionIPage = userCollectionMapper.selectPage(page, new LambdaQueryWrapper<UserCollection>()  .eq(UserCollection::getUserId, openId)  );  List<UserCollection> records = userCollectionIPage.getRecords();  if (CollectionUtils.isEmpty(records)) {  productEsIPage.setTotal(0L);  return productEsIPage;  }  //如果用户有收藏  List<Long> prodIds = records.stream().map(UserCollection::getProdId).collect(Collectors.toList());  //远程调用搜索服务 查询用户收藏的商品对象  List<ProductEs> userCollectByProdIds = userCollectSearchFeign.getUserCollectByProdIds(prodIds);  productEsIPage.setCurrent(page.getCurrent());  productEsIPage.setSize(page.getSize());  productEsIPage.setTotal(userCollectionIPage.getTotal());  productEsIPage.setRecords(userCollectByProdIds);  return productEsIPage;  }  /\*\*  \* 用户是否收藏过该商品  \*  \* @param openId  \* @param prodId  \* @return  \*/  @Override  public Boolean isCollection(String openId, Long prodId) {  UserCollection userCollection = userCollectionMapper.selectOne(new LambdaQueryWrapper<UserCollection>()  .eq(UserCollection::getUserId, openId)  .eq(UserCollection::getProdId, prodId)  );  return userCollection != null;  }  /\*\*  \* 收藏或取消  \*  \* @param openId  \* @param prodId  \*/  @Override  @Transactional  public void addOrCancel(String openId, Long prodId) {  //先查询用户是否收藏了该商品  UserCollection userCollection = userCollectionMapper.selectOne(new LambdaQueryWrapper<UserCollection>()  .eq(UserCollection::getProdId, prodId)  .eq(UserCollection::getUserId, openId)  );  if (!ObjectUtils.isEmpty(userCollection)) {  //说明收藏了，我们删掉这一行数据  userCollectionMapper.delete(new LambdaQueryWrapper<UserCollection>()  .eq(UserCollection::getProdId, prodId)  .eq(UserCollection::getUserId, openId)  );  } else {  //插入一条收藏  UserCollection userCollection1 = new UserCollection();  userCollection1.setCreateTime(new Date());  userCollection1.setProdId(prodId);  userCollection1.setUserId(openId);  userCollectionMapper.insert(userCollection1);  }  }  } |