# EGO消息模块

## EGO消息模块的定位

主要用于商城中发送消息，短信等功能，实现异步操作，

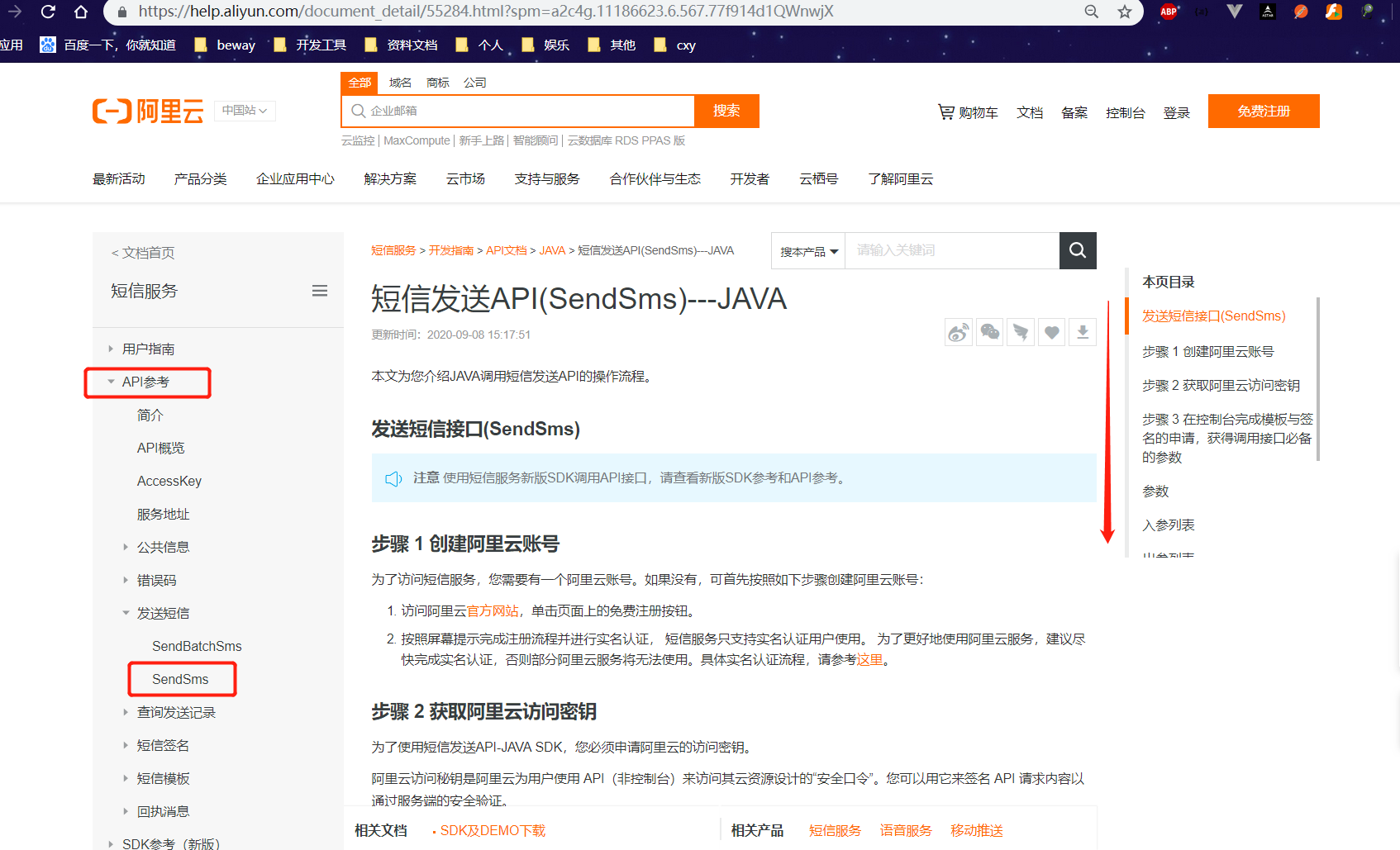
我们的消息主要有两种，第一个是阿里大于的短信，第二个是微信公众号

## 短信消息的集成（阿里大于）

主要用于用户注册，绑定手机号等

### 官网流程文档：

<https://help.aliyun.com/document_detail/55284.html?spm=a2c4g.11186623.6.567.77f914d1QWnwjX>



### 登录阿里大于：

<https://dayu.aliyun.com/>

### 去短信服务





https://dysms.console.aliyun.com/dysms.htm?spm=5176.12818093.0.ddysms.6eac16d0SKkGSC#/domestic/text/template

### 设置签名模板（需要审核）



### 拿到AccessKey





### 申请短信模板



### 总结需要的东西

**短信签名，短信模板id，AccessKey，AccessKey Secret，version（版本信息，有点坑），Action（SendSms）关键的还有一个jar包（依赖）**

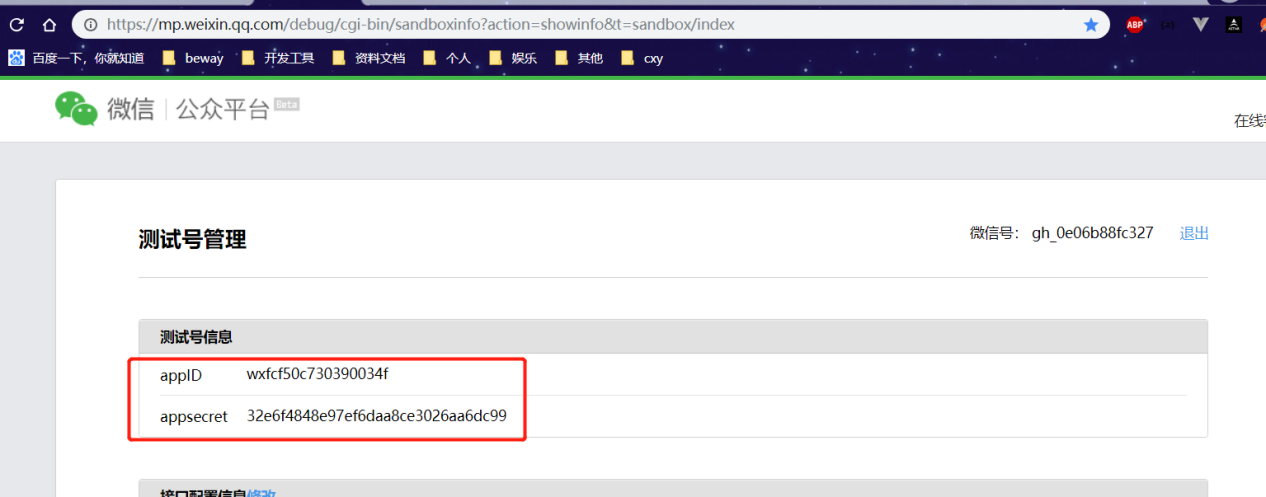
## 微信消息的集成（微信公众号）

主要用于商城购买提示，金额提醒，活动广告等

### 申请微信测试号（和微信小程序有区别）

<https://mp.weixin.qq.com/debug/cgi-bin/sandboxinfo?action=showinfo&t=sandbox/index>

### 拿到信息



### 查看具体文档



### 获取access\_token（这里区别其他的token）

本质就是发送一个请求



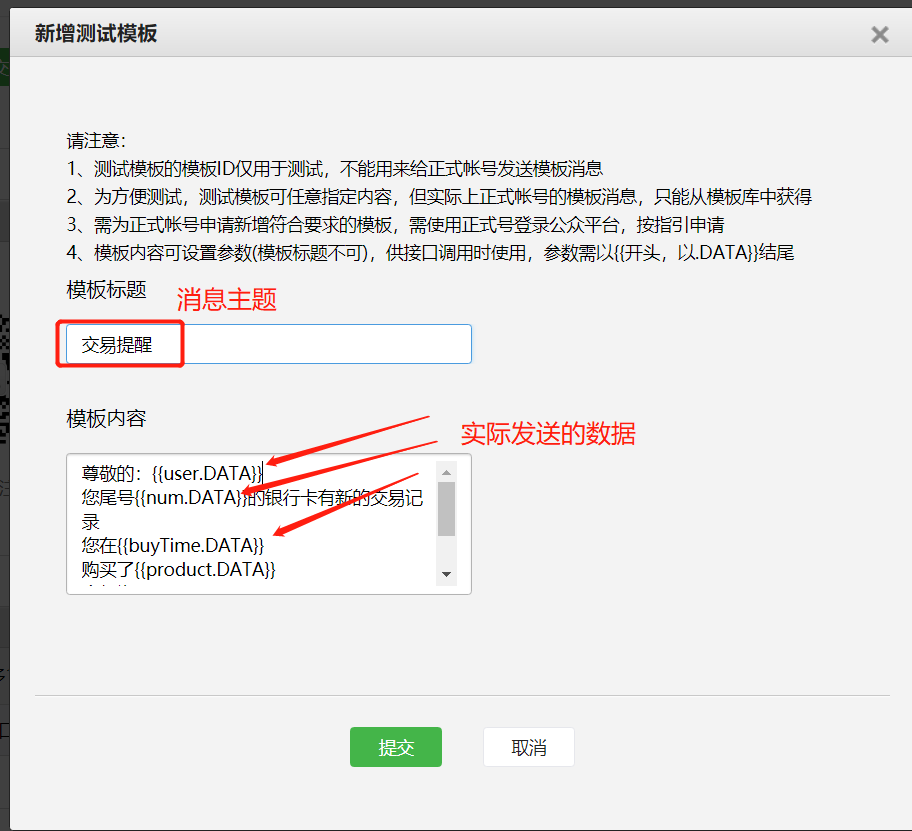
### 发送消息模板文档



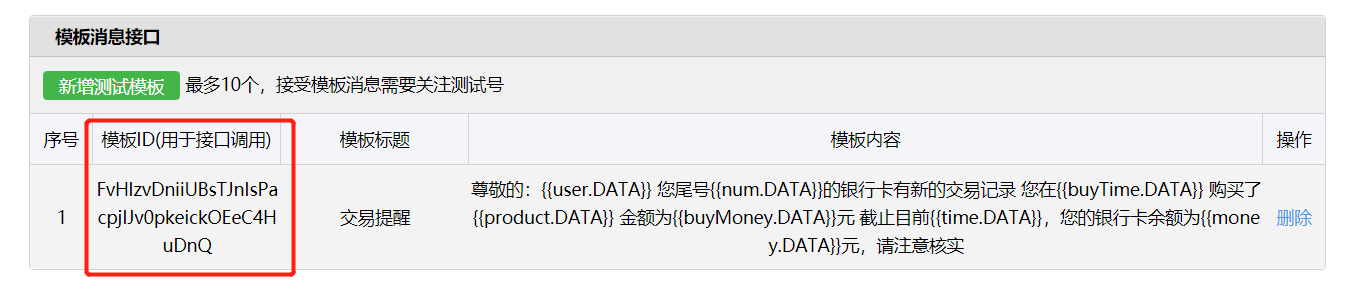


### 消息模板





### 拿到模板ID



### 总结需要的东西

**appID，appsecret，模板ID**

**流程：**

**1.根据appid和appsecret拿到access\_token**

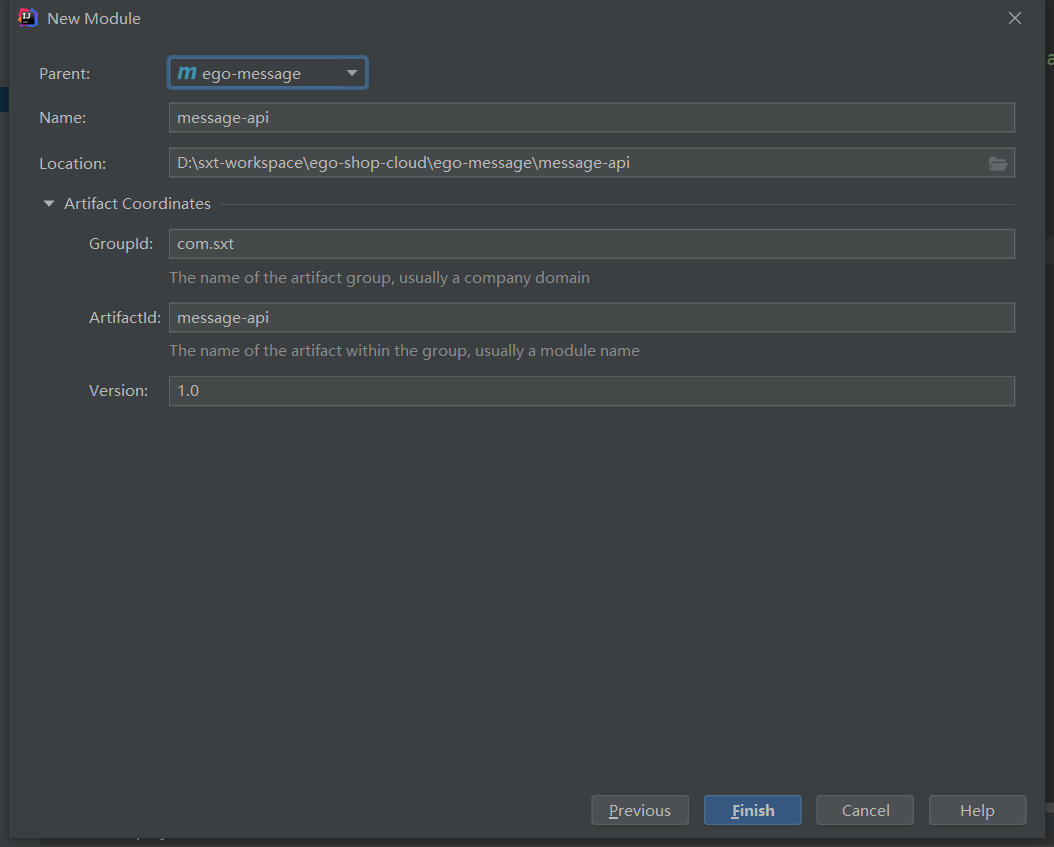
**2.发一个请求（请求中包含模板内容，access\_token）**

## EGO消息**模块**的搭建

### 在ego-shop-cloud下面新建ego-message模块

### 在ego-message模块下新建message-api模块

#### 新建maven模块



#### 添加短信消息对象SmsMessage（具体看阿里大于文档）

|  |
| --- |
| @Data  public class SmsMessage implements Serializable {  private String regionId = "cn-hangzhou";  @ApiModelProperty("发送的手机号")  private String phoneNumbers;  @ApiModelProperty("发送短信的签名")  private String signName;  @ApiModelProperty("发送短信的模板")  private String templateCode;  @ApiModelProperty("发送短信模板的数据")  private String templateParam;  @ApiModelProperty("发送短信的扩展的数据")  private String smsUpExtendCode;  @ApiModelProperty("外呼人")  private String outId;  } |

#### 添加微信消息的对象（具体看微信测试号文档）

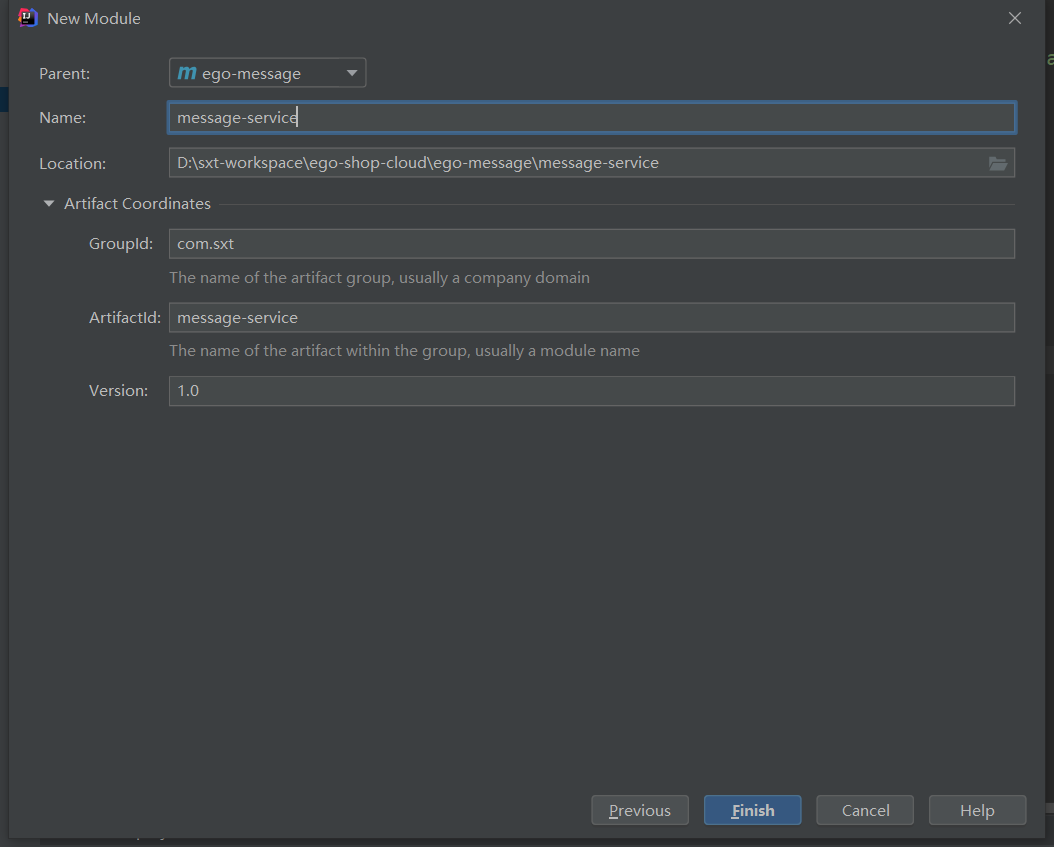
|  |
| --- |
| @Data  public class WechatMessage implements Serializable {  @ApiModelProperty("发送给那个用户")  @JsonProperty("touser")  private String toUser;  @ApiModelProperty("本次发消息的模板是哪一个")  @JsonProperty("template\_id")  private String templateId;  @ApiModelProperty("点击该消息跳转到那个页面")  private String url;  @ApiModelProperty("微信消息的顶层颜色")  @JsonProperty("topcolor")  private String topColor;  @ApiModelProperty("模板消息数据")  private Map<String, Map<String, String>> data;  public static Map<String, String> buildProp(String value, String color) {  HashMap<String, String> prop = new HashMap<>(4);  prop.put("value", value);  prop.put("color", color);  return prop;  }  } |

#### 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-message</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>message-api</artifactId>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  </project> |

### 在ego-message模块下新建message-service模块

#### 新建maven模块



#### 修改pom.xml文件

|  |
| --- |
| <?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-message</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>message-service</artifactId>  <description>ego商城消息服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>message-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>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  **<dependency>**  **<groupId>com.aliyun</groupId>**  **<artifactId>aliyun-java-sdk-core</artifactId>**  **<version>4.0.6</version> <!-- 注：如提示报错，先升级基础包版，无法解决可联系技术支持 -->**  **</dependency>**  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 添加启动类

|  |
| --- |
| @SpringBootApplication(exclude = DataSourceAutoConfiguration.class)  @EnableEurekaClient  @EnableScheduling //开启定时任务  public class MessageServiceApplication {  public static void main(String[] args) {  SpringApplication.run(MessageServiceApplication.class, args);  }  @Bean  public RestTemplate restTemplate() {  return new RestTemplate();  }  } |

#### 添加bootstrap.yml

|  |
| --- |
| spring:  application:  name: message-service  cloud:  config:  discovery:  service-id: config-server  enabled: true  name: message-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 |

#### 添加远端message-service/message-service-dev.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8086}  spring:  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  listener:  simple:  acknowledge-mode: manual #手动签收消息 默认是自动的  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*'  **sms:**  **region-id: cn-hangzhou**  **access-key-id: LTAIkRdZIaLMBGKw**  **access-secret: ZPuDXJKxZ0ZNndDXlWkO8WAcZK26n7**  **sys-domain: dysmsapi.aliyuncs.com**  **version: 2017-05-25**  **wechat:**  **app-id: wxfcf50c730390034f**  **app-secret: 32e6f4848e97ef6daa8ce3026aa6dc99**  **token-url: https://api.weixin.qq.com/cgi-bin/token?grant\_type=client\_credential&appid=%s&secret=%s**  **message-url: https://api.weixin.qq.com/cgi-bin/message/template/send?access\_token=%s** |

#### 修改ego-common中的QueueConstant

|  |
| --- |
| public class QueueConstant {  //商品修改的队列  public static final String PROD\_CHANGE\_QUEUE = "prod.change.queue";  //商品修改队列的交换机  public static final String PROD\_CHANGE\_EXCHANGE = "prodChangeEx";  //商品修改的路由key  public static final String PROD\_ROUTER\_KEY = "prodRouterKey";  **//绑定手机号的队列**  **public static final String PHONE\_MUN\_QUEUE = "phone.num.queue";**  **//绑定手机号的交换机**  **public static final String PHONE\_MUN\_EXCHANGE = "phoneNumEx";**  **//绑定手机号的路由key**  **public static final String PHONE\_MUN\_KEY = "phoneRouterKey";**  **//微信消息的队列**  **public static final String WECHAT\_MSG\_QUEUE = "wechat.msg.queue";**  **//微信消息的交换机**  **public static final String WECHAT\_MSG\_EXCHANGE = "wechatMsgEx";**  **//微信消息的路由key**  **public static final String WECHAT\_MSG\_KEY = "wxRouterKey";**  } |

#### 在member-service模块中添加RabbitConfig

|  |
| --- |
| @Configuration  public class RabbitMqConfig {  @Bean  public Queue phoneQueue() {  return new Queue(QueueConstant.PHONE\_MUN\_QUEUE);  }  @Bean  public DirectExchange phoneDirectEx() {  return new DirectExchange(QueueConstant.PHONE\_MUN\_EXCHANGE);  }  @Bean  public Binding phoneBind() {  return BindingBuilder.bind(phoneQueue()).to(phoneDirectEx()).with(QueueConstant.PHONE\_MUN\_KEY);  }  @Bean  public Queue wxQueue() {  return new Queue(QueueConstant.WECHAT\_MSG\_QUEUE);  }  @Bean  public DirectExchange wxDirectEx() {  return new DirectExchange(QueueConstant.WECHAT\_MSG\_EXCHANGE);  }  @Bean  public Binding wxBind() {  return BindingBuilder.bind(wxQueue()).to(wxDirectEx()).with(QueueConstant.WECHAT\_MSG\_KEY);  }  } |

#### 在message-service中添加阿里大于的配置类

|  |
| --- |
| @ConfigurationProperties(prefix = "sms")  @Component  @Data  public class SmsProperties {  /\*\*  \* 区域id 固定的  \*/  private String regionId;  /\*\*  \* 验证的id  \*/  private String accessKeyId;  /\*\*  \* 验证的密码  \*/  private String accessSecret;  /\*\*  \* jar包的域名 固定的  \*/  private String sysDomain;  /\*\*  \* 版本号  \*/  private String version;  } |

#### 在message-service中添加阿里大于的自动配置

|  |
| --- |
| @EnableConfigurationProperties(SmsProperties.class)  @Configuration  @Slf4j  public class SmsAutoConfiguration {  @Autowired  private SmsProperties smsProperties;  /\*\*  \* 创建一个阿里的客户端对象  \*  \* @return  \*/  @Bean  public IAcsClient iAcsClient() {  DefaultProfile profile = DefaultProfile.getProfile(  smsProperties.getRegionId(),  smsProperties.getAccessKeyId(),  smsProperties.getAccessSecret());  return new DefaultAcsClient(profile);  }  } |

#### 在message-service中添加阿里大于短信的监听

|  |
| --- |
| @Component  @Slf4j  public class SmsListener {  @Autowired  private SmsProperties smsProperties;  @Autowired  private IAcsClient iAcsClient;  /\*\*  \* 监听消息队列  \*  \* @param channel  \* @param message  \* @param tagId  \*/  @RabbitListener(queues = QueueConstant.PHONE\_MUN\_QUEUE, concurrency = "3-5")  public void handlerPhoneSms(Channel channel, Message message, @Header(AmqpHeaders.DELIVERY\_TAG) Long tagId) {  log.info("接受到短信的请求");  try {  SmsMessage smsMessage = JSON.parseObject(new String(message.getBody()), SmsMessage.class);  sendSms(smsMessage);  //手动签收  channel.basicAck(tagId, false);  } catch (Exception e) {  log.error("发送短信失败", e.getMessage());  }  }  /\*\*  \* 请求阿里大于 发短信  \* 组装数据  \*  \* @param smsMessage  \*/  private void sendSms(SmsMessage smsMessage) {  //组装请求对象  CommonRequest commonRequest = new CommonRequest();  commonRequest.setRegionId(smsProperties.getRegionId());  commonRequest.setMethod(MethodType.POST);  commonRequest.setDomain(smsProperties.getSysDomain());  commonRequest.setAction("SendSms");  commonRequest.setVersion(smsProperties.getVersion());  //下面是短信的具体内容  commonRequest.putQueryParameter("PhoneNumbers", smsMessage.getPhoneNumbers());  commonRequest.putQueryParameter("SignName", smsMessage.getSignName());  commonRequest.putQueryParameter("TemplateCode", smsMessage.getTemplateCode());  commonRequest.putQueryParameter("TemplateParam", smsMessage.getTemplateParam());  commonRequest.putQueryParameter("SmsUpExtendCode", smsMessage.getSmsUpExtendCode());  commonRequest.putQueryParameter("OutId", smsMessage.getOutId());  try {  CommonResponse commonResponse = iAcsClient.getCommonResponse(commonRequest);  System.out.println(commonResponse.getData());  } catch (ClientException e) {  log.error("调用阿里大于发短信失败", e.getMessage());  }  }  } |

#### 在message-service中添加微信公众号的配置类

|  |
| --- |
| @ConfigurationProperties(prefix = "wechat")  @Data  @Component  public class WechatProperties {  @ApiModelProperty("微信的应用的id")  private String appId;  @ApiModelProperty("微信的密钥")  private String appSecret;  @ApiModelProperty("获取Tokenurl的地址")  private String tokenUrl;  @ApiModelProperty("发送微信模板消息的地址")  private String messageUrl;  } |

#### 在message-service中添加微信公众号的自动配置

|  |
| --- |
| @EnableConfigurationProperties(WechatProperties.class)  @Configuration  @Slf4j  public class WechatAutoConfiguration {  @Autowired  private WechatProperties wechatProperties;  @Autowired  private RestTemplate restTemplate;  //因为我们需要定时刷新token，为了让其他线程可以看到，我们使用volatile来修饰  private volatile String accessToken;  /\*\*  \* 刷新token的方法 两个小时以内获取一次  \*  \* @return  \*/  @Scheduled(fixedRate = 7000 \* 1000)  public void refresh() {  //拿到请求token 的地址  String tokenUrl = wechatProperties.getTokenUrl();  //发送一个请求  // GET https://api.weixin.qq.com/cgi-bin/token?grant\_type=client\_credential&appid=APPID&secret=APPSECRET  String accessToken = restTemplate  .getForObject(  String.format(  tokenUrl,  wechatProperties.getAppId(),  wechatProperties.getAppSecret()),  String.class);  JSONObject jsonObject = JSON.parseObject(accessToken);  if (jsonObject.get("access\_token") != null) {  //说明本次获取成功  log.info("获取accessToken成功");  this.accessToken = jsonObject.getString("access\_token");  } else {  log.error("获取accessToken失败");  }  }  //给一个get方法  public String getAccessToken() {  return accessToken;  }  } |

#### 在message-service中添加微信公众号消息的监听

|  |
| --- |
| @Component  @Slf4j  public class WechatListener {  @Autowired  private WechatAutoConfiguration wechatAutoConfiguration;  @Autowired  private WechatProperties wechatProperties;  @Autowired  private RestTemplate restTemplate;  /\*\*  \* 监听发微信消息的队列  \*  \* @param channel  \* @param message  \* @param tagId  \*/  @RabbitListener(queues = QueueConstant.WECHAT\_MSG\_QUEUE, concurrency = "3-5")  public void HandlerWechatMessage(Channel channel, Message message, @Header(AmqpHeaders.DELIVERY\_TAG) long tagId) {  //接收到一个消息 json格式的  try {  //转换消息  WechatMessage wechatMessage = JSON.parseObject(new String(message.getBody()), WechatMessage.class);  log.info("真正的发送微信消息");  sendWechatMessage(wechatMessage);  //手动签收  channel.basicAck(tagId, false);  } catch (Exception e) {  log.error("发送微信消息失败", e.getMessage());  }  }  /\*\*  \* 发送微信消息  \*  \* @param msg  \*/  private void sendWechatMessage(WechatMessage msg) {  //就是往一个地址发送一个消息  //POST请求  //https://api.weixin.qq.com/cgi-bin/message/template/send?access\_token=ACCESS\_TOKEN  String response = restTemplate.postForObject(String.format(wechatProperties.getMessageUrl(), wechatAutoConfiguration.getAccessToken()  ), msg  , String.class);  System.out.println(response);  //发送成功 可以远程调用 写数据库  }  } |

### 在member-service中添加message模块依赖

|  |
| --- |
| <dependency>  <groupId>com.sxt</groupId>  <artifactId>message-api</artifactId>  <version>1.0</version>  </dependency> |

#### 修改message-service中的UserController（发短信）

|  |
| --- |
| /\*\*  \* 用户绑定手机号 发短信消息  \*  \* @param phonenum  \* @return  \*/  @PostMapping({"p/sms/send"})  public ResponseEntity<Void> bindPhoneNum(@RequestBody Map<String, String> phonenum) {  userService.bindPhone(phonenum);  return ResponseEntity.ok().build();  } |

#### 修改message-service中的UserService

|  |
| --- |
| public interface UserService extends IService<User> {  /\*\*  \* 用户绑定手机号  \*  \* @param phonenum  \*/  void bindPhone(Map<String, String> phonenum);  } |

#### 修改message-service中的UserServiceImpl

|  |
| --- |
| /\*\*  \* 用户绑定手机号  \* 1生成一个验证码，  \* 2存放redis，3分钟有效  \* 3往消息队列放一个消息  \*  \* @param phonenum  \*/  @Override  public void bindPhone(Map<String, String> phonenum) {  log.info("绑定手机号开始");  //获取验证吗  String code = getValidateCode();  String phone = phonenum.get("phonenum");  //存放redis  redisTemplate.opsForValue().set(PhoneConstant.BIND\_PHONE\_NUM + phone, code, Duration.ofSeconds(180));  //创建一个消息模板  SmsMessage smsMessage = new SmsMessage();  smsMessage.setPhoneNumbers(phone);  smsMessage.setSignName("ego商城");  smsMessage.setTemplateCode("SMS\_203185255");  JSONObject data = new JSONObject();  data.put("code", code);  smsMessage.setTemplateParam(data.toString());  //往mq放一个消息  rabbitTemplate.convertAndSend(QueueConstant.PHONE\_MUN\_EXCHANGE, QueueConstant.PHONE\_MUN\_KEY, JSON.toJSONString(smsMessage));  } |

#### 修改message-service中的UserController（发微信消息）

|  |
| --- |
| /\*\*  \* 测试发微信消息的模板  \*  \* @param openId  \* @param username  \* @return  \*/  @GetMapping("test/wxMsg")  public String testWxMsg(String openId, String username) {  WechatMessage wechatMessage = new WechatMessage();  wechatMessage.setToUser(openId);  wechatMessage.setTemplateId("FvHlzvDniiUBsTJnIsPacpjIJv0pkeickOEeC4HuDnQ");  wechatMessage.setUrl("www.baidu.com");  wechatMessage.setTopColor("#FF0000");  //设置具体的消息内容  HashMap<String, Map<String, String>> data = new HashMap<>();  data.put("user", WechatMessage.buildProp(username, "#173177"));  data.put("num", WechatMessage.buildProp("4793", "#173177"));  data.put("buyTime", WechatMessage.buildProp(new Date().toString(), "#173177"));  data.put("product", WechatMessage.buildProp("女朋友", "#173177"));  data.put("buyMoney", WechatMessage.buildProp("0.99", "#173177"));  data.put("time", WechatMessage.buildProp(new Date().toString(), "#173177"));  data.put("money", WechatMessage.buildProp("9,548,762,130", "#173177"));  wechatMessage.setData(data);  rabbitTemplate.convertAndSend(QueueConstant.WECHAT\_MSG\_EXCHANGE, QueueConstant.WECHAT\_MSG\_KEY, JSON.toJSONString(wechatMessage));  return "发送成功";  } |