微信扫码登录前期准备

1. 微信开放平台注册 <https://open.weixin.qq.com/>
2. 邮箱激活
3. 完善开发者资料
4. 开发者资质凭证（准备营业执照，1-2个工作日审核，300元认证费）
5. 创建网站应用（7个工作日审核）
6. 微信扫码登录的参数：app\_id、app\_secret、redirect\_url

wx.open.app\_id=wxed9954c01bb89b47

wx.open.app\_secret=a7482517235173ddb4083788de60b90e

wx.open.redirect\_url=http://guli.shop/api/ucenter/wx/callback

yygh.baseUrl=http://localhost:3000

1. 微信扫码登录的官方开发文档

<https://open.weixin.qq.com/cgi-bin/showdocument?action=dir_list&t=resource/res_list&verify=1&id=open1419316505&token=e547653f995d8f402704d5cb2945177dc8aa4e7e&lang=zh_CN>

1. 二维码生成（参考开发文档，参数都是微信认证的时候得到的，直接返给前端创建

wxLogin实例化对象将参数带进去就生成二维码了，id对应的dom的id）



1. 微信二维码生成以后用户一扫， 微信就会调用咱们给他准备好的redirect\_uri的路径， 他会给咱们返回一个code值，利用httpClient远程调用接着获取用户信息

4.1、引入httpClient依赖

<dependency>  
 <groupId>org.apache.httpcomponents</groupId>  
 <artifactId>httpclient</artifactId>  
 </dependency>

<dependency>  
 <groupId>commons-io</groupId>  
 <artifactId>commons-io</artifactId>  
 <version>2.7</version>  
</dependency>

<dependency>  
 <groupId>commons-lang</groupId>  
 <artifactId>commons-lang</artifactId>  
 <version>2.6</version>  
</dependency>

4.2、httpClient工具类

import org.apache.commons.io.IOUtils;  
import org.apache.commons.lang.StringUtils;  
import org.apache.http.Consts;  
import org.apache.http.HttpEntity;  
import org.apache.http.HttpResponse;  
import org.apache.http.NameValuePair;  
import org.apache.http.client.HttpClient;  
import org.apache.http.client.config.RequestConfig;  
import org.apache.http.client.entity.UrlEncodedFormEntity;  
import org.apache.http.client.methods.HttpGet;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.conn.ConnectTimeoutException;  
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;  
import org.apache.http.conn.ssl.SSLContextBuilder;  
import org.apache.http.conn.ssl.TrustStrategy;  
import org.apache.http.conn.ssl.X509HostnameVerifier;  
import org.apache.http.entity.ContentType;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.CloseableHttpClient;  
import org.apache.http.impl.client.HttpClients;  
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;  
import org.apache.http.message.BasicNameValuePair;  
  
import javax.net.ssl.SSLContext;  
import javax.net.ssl.SSLException;  
import javax.net.ssl.SSLSession;  
import javax.net.ssl.SSLSocket;  
import java.io.IOException;  
import java.net.SocketTimeoutException;  
import java.security.GeneralSecurityException;  
import java.security.cert.CertificateException;  
import java.security.cert.X509Certificate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
  
public class HttpClientUtils {  
 public static final int *connTimeout*=10000;  
 public static final int *readTimeout*=10000;  
 public static final String *charset*="UTF-8";  
 private static HttpClient *client* = null;  
  
 static {  
 PoolingHttpClientConnectionManager cm = new PoolingHttpClientConnectionManager();  
 cm.setMaxTotal(128);  
 cm.setDefaultMaxPerRoute(128);  
 *client* = HttpClients.*custom*().setConnectionManager(cm).build();  
 }  
  
 public static String postParameters(String url, String parameterStr) throws ConnectTimeoutException, SocketTimeoutException, Exception{  
 return *post*(url,parameterStr,"application/x-www-form-urlencoded",*charset*,*connTimeout*,*readTimeout*);  
 }  
  
 public static String postParameters(String url, String parameterStr,String charset, Integer connTimeout, Integer readTimeout) throws ConnectTimeoutException, SocketTimeoutException, Exception{  
 return *post*(url,parameterStr,"application/x-www-form-urlencoded",charset,connTimeout,readTimeout);  
 }  
  
 public static String postParameters(String url, Map<String, String> params) throws ConnectTimeoutException,  
 SocketTimeoutException, Exception {  
 return *postForm*(url, params, null, *connTimeout*, *readTimeout*);  
 }  
  
 public static String postParameters(String url, Map<String, String> params, Integer connTimeout,Integer readTimeout) throws ConnectTimeoutException,  
 SocketTimeoutException, Exception {  
 return *postForm*(url, params, null, connTimeout, readTimeout);  
 }  
  
 public static String get(String url) throws Exception {  
 return *get*(url, *charset*, null, null);  
 }  
  
 public static String get(String url, String charset) throws Exception {  
 return *get*(url, charset, *connTimeout*, *readTimeout*);  
 }  
  
 */\*\*  
 \* 发送一个 Post 请求, 使用指定的字符集编码.  
 \*  
 \** ***@param*** *url  
 \** ***@param*** *body RequestBody  
 \** ***@param*** *mimeType 例如 application/xml "application/x-www-form-urlencoded" a=1&b=2&c=3  
 \** ***@param*** *charset 编码  
 \** ***@param*** *connTimeout 建立链接超时时间,毫秒.  
 \** ***@param*** *readTimeout 响应超时时间,毫秒.  
 \** ***@return*** *ResponseBody, 使用指定的字符集编码.  
 \** ***@throws*** *ConnectTimeoutException 建立链接超时异常  
 \** ***@throws*** *SocketTimeoutException 响应超时  
 \** ***@throws*** *Exception  
 \*/* public static String post(String url, String body, String mimeType,String charset, Integer connTimeout, Integer readTimeout)  
 throws ConnectTimeoutException, SocketTimeoutException, Exception {  
 HttpClient client = null;  
 HttpPost post = new HttpPost(url);  
 String result = "";  
 try {  
 if (StringUtils.*isNotBlank*(body)) {  
 HttpEntity entity = new StringEntity(body, ContentType.*create*(mimeType, charset));  
 post.setEntity(entity);  
 }  
 // 设置参数  
 RequestConfig.Builder customReqConf = RequestConfig.*custom*();  
 if (connTimeout != null) {  
 customReqConf.setConnectTimeout(connTimeout);  
 }  
 if (readTimeout != null) {  
 customReqConf.setSocketTimeout(readTimeout);  
 }  
 post.setConfig(customReqConf.build());  
  
 HttpResponse res;  
 if (url.startsWith("https")) {  
 // 执行 Https 请求.  
 client = *createSSLInsecureClient*();  
 res = client.execute(post);  
 } else {  
 // 执行 Http 请求.  
 client = HttpClientUtils.*client*;  
 res = client.execute(post);  
 }  
 result = IOUtils.*toString*(res.getEntity().getContent(), charset);  
 } finally {  
 post.releaseConnection();  
 if (url.startsWith("https") && client != null&& client instanceof CloseableHttpClient) {  
 ((CloseableHttpClient) client).close();  
 }  
 }  
 return result;  
 }  
  
  
 */\*\*  
 \* 提交form表单  
 \*  
 \** ***@param*** *url  
 \** ***@param*** *params  
 \** ***@param*** *connTimeout  
 \** ***@param*** *readTimeout  
 \** ***@return*** *\** ***@throws*** *ConnectTimeoutException  
 \** ***@throws*** *SocketTimeoutException  
 \** ***@throws*** *Exception  
 \*/* public static String postForm(String url, Map<String, String> params, Map<String, String> headers, Integer connTimeout,Integer readTimeout) throws ConnectTimeoutException,  
 SocketTimeoutException, Exception {  
  
 HttpClient client = null;  
 HttpPost post = new HttpPost(url);  
 try {  
 if (params != null && !params.isEmpty()) {  
 List<NameValuePair> formParams = new ArrayList<NameValuePair>();  
 Set<Map.Entry<String, String>> entrySet = params.entrySet();  
 for (Map.Entry<String, String> entry : entrySet) {  
 formParams.add(new BasicNameValuePair(entry.getKey(), entry.getValue()));  
 }  
 UrlEncodedFormEntity entity = new UrlEncodedFormEntity(formParams, Consts.*UTF\_8*);  
 post.setEntity(entity);  
 }  
  
 if (headers != null && !headers.isEmpty()) {  
 for (Map.Entry<String, String> entry : headers.entrySet()) {  
 post.addHeader(entry.getKey(), entry.getValue());  
 }  
 }  
 // 设置参数  
 RequestConfig.Builder customReqConf = RequestConfig.*custom*();  
 if (connTimeout != null) {  
 customReqConf.setConnectTimeout(connTimeout);  
 }  
 if (readTimeout != null) {  
 customReqConf.setSocketTimeout(readTimeout);  
 }  
 post.setConfig(customReqConf.build());  
 HttpResponse res = null;  
 if (url.startsWith("https")) {  
 // 执行 Https 请求.  
 client = *createSSLInsecureClient*();  
 res = client.execute(post);  
 } else {  
 // 执行 Http 请求.  
 client = HttpClientUtils.*client*;  
 res = client.execute(post);  
 }  
 return IOUtils.*toString*(res.getEntity().getContent(), "UTF-8");  
 } finally {  
 post.releaseConnection();  
 if (url.startsWith("https") && client != null  
 && client instanceof CloseableHttpClient) {  
 ((CloseableHttpClient) client).close();  
 }  
 }  
 }  
  
 */\*\*  
 \* 发送一个 GET 请求  
 \*/* public static String get(String url, String charset, Integer connTimeout,Integer readTimeout)  
 throws ConnectTimeoutException,SocketTimeoutException, Exception {  
  
 HttpClient client = null;  
 HttpGet get = new HttpGet(url);  
 String result = "";  
 try {  
 // 设置参数  
 RequestConfig.Builder customReqConf = RequestConfig.*custom*();  
 if (connTimeout != null) {  
 customReqConf.setConnectTimeout(connTimeout);  
 }  
 if (readTimeout != null) {  
 customReqConf.setSocketTimeout(readTimeout);  
 }  
 get.setConfig(customReqConf.build());  
  
 HttpResponse res = null;  
  
 if (url.startsWith("https")) {  
 // 执行 Https 请求.  
 client = *createSSLInsecureClient*();  
 res = client.execute(get);  
 } else {  
 // 执行 Http 请求.  
 client = HttpClientUtils.*client*;  
 res = client.execute(get);  
 }  
  
 result = IOUtils.*toString*(res.getEntity().getContent(), charset);  
 } finally {  
 get.releaseConnection();  
 if (url.startsWith("https") && client != null && client instanceof CloseableHttpClient) {  
 ((CloseableHttpClient) client).close();  
 }  
 }  
 return result;  
 }  
  
 */\*\*  
 \* 从 response 里获取 charset  
 \*/* @SuppressWarnings("unused")  
 private static String getCharsetFromResponse(HttpResponse ressponse) {  
 // Content-Type:text/html; charset=GBK  
 if (ressponse.getEntity() != null && ressponse.getEntity().getContentType() != null && ressponse.getEntity().getContentType().getValue() != null) {  
 String contentType = ressponse.getEntity().getContentType().getValue();  
 if (contentType.contains("charset=")) {  
 return contentType.substring(contentType.indexOf("charset=") + 8);  
 }  
 }  
 return null;  
 }  
  
 */\*\*  
 \* 创建 SSL连接  
 \** ***@return*** *\** ***@throws*** *GeneralSecurityException  
 \*/* private static CloseableHttpClient createSSLInsecureClient() throws GeneralSecurityException {  
 try {  
 SSLContext sslContext = new SSLContextBuilder().loadTrustMaterial(null, new TrustStrategy() {  
 public boolean isTrusted(X509Certificate[] chain, String authType) throws CertificateException {  
 return true;  
 }  
 }).build();  
  
 SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(sslContext, new X509HostnameVerifier() {  
  
 @Override  
 public boolean verify(String arg0, SSLSession arg1) {  
 return true;  
 }  
  
 @Override  
 public void verify(String host, SSLSocket ssl)  
 throws IOException {  
 }  
  
 @Override  
 public void verify(String host, X509Certificate cert)  
 throws SSLException {  
 }  
  
 @Override  
 public void verify(String host, String[] cns,  
 String[] subjectAlts) throws SSLException {  
 }  
 });  
 return HttpClients.*custom*().setSSLSocketFactory(sslsf).build();  
  
 } catch (GeneralSecurityException e) {  
 throw e;  
 }  
 }  
}

4.2、httpClient远程调用获取用户信息，存到数据库

// 扫码之后的回调方法， 得到扫描人的信息(接口路径要和 redirect\_uti 的路径保持一致， 微信才会调到)  
@GetMapping("callback")  
public Object callback(String code) {  
 // 用code、appid、appscrect 换 access\_token 、openid  
 StringBuffer stringBuffer = new StringBuffer()  
 .append("https://api.weixin.qq.com/sns/oauth2/access\_token")  
 .append("?appid=%s")  
 .append("&secret=%s")  
 .append("&code=%s")  
 .append("&grant\_type=authorization\_code");  
 String accessTokenUrl = String.*format*(stringBuffer.toString(), appid, app\_secret, code);  
  
 // 用 httpClient 调用微信的接口获取access\_token  
 try {  
 String accTokenInfo = HttpClientUtils.*get*(accessTokenUrl);  
 JSONObject jsonObject = JSONObject.*parseObject*(accTokenInfo);  
 String access\_token = jsonObject.getString("access\_token");  
 String openid = jsonObject.getString("openid");  
  
 // 拿着access\_token和openid获取用户的个人信息  
 String baseUserInfo = "https://api.weixin.qq.com/sns/userinfo" +  
 "?access\_token=%s"+  
 "&openid=%s";  
 String userInfoUrl = String.*format*(baseUserInfo, access\_token, openid);  
 String result = HttpClientUtils.*get*(userInfoUrl);  
 JSONObject jsonObject1 = JSONObject.*parseObject*(result);  
  
 // 用户信息的openid  
 String openid1 = jsonObject1.getString("openid");  
 // 用户昵称  
 String nickname = jsonObject1.getString("nickname");  
 // 用户性别  
 String sex = jsonObject1.getString("sex");  
 //  
 String city = jsonObject1.getString("city");  
 //  
 String country = jsonObject1.getString("country");  
 //  
 String province = jsonObject1.getString("province");  
 // 头像  
 String headimgurl = jsonObject1.getString("headimgurl");  
  
 // 获取到信息之后将用户信息存到数据库就完成扫码登录的操作  
 // ........  
 // 给前端返回信息 name 和 token  
 Map<String, Object> map = new HashMap<>();  
 map.put("name", nickname);  
 String token = JwtHelper.*createToken*(Long.*valueOf*(openid1), nickname);  
 map.put("token", token);  
 return map;  
 } catch (Exception e) {  
 e.printStackTrace();  
 return null;  
 }  
}

至此， 扫码登录成功，后台再准备个接口让前台轮询查数据库直到扫码成功之后前端做相应的处理（当然websocket推送也行）