**Java Spring Boot集成第三方登录 (QQ和微信)**

[](https://www.jianshu.com/u/ce36d64a62a2)

[冷风\_f818](https://www.jianshu.com/u/ce36d64a62a2) 关注

2017.11.30 11:41 字数 562 阅读 536评论 1喜欢 7

第一次发文章,最近写了一个集成第三方登录的Demo

1. QQ互联/微信开发者平台 通过认证,然后创建应用 获得你的APPID 和AppSecret 配置回调函数
2. 微信QQ请求都是https的请求,这里需要一个工具类 HttpClientUtils.java 用来请求QQ或微信的接口,工具类我贴在下面
3. 添加 httpclient的依赖 依赖的jar包有：commons-lang-2.6.jar、httpclient-4.3.2.jar、httpcore-4.3.1.jar、commons-io-2.4.jar
4. 配置Constants类, APPID 以及 AppSecret 都放到yml文件中
5. yml文件中写入你的APPID等信息
6. 按开发文档上拼接请求参数,发送请求(代码在下面)

**maven的依赖**

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.4</version>

</dependency>

<dependency>

<groupId>org.apache.httpcomponents</groupId>

<artifactId>httpclient</artifactId>

<version>4.3.2</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.38</version>

</dependency>

**工具类HttpClientUtils.java**

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.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

import javax.net.ssl.SSLContext;

import javax.net.ssl.SSLException;

import javax.net.ssl.SSLSession;

import javax.net.ssl.SSLSocket;

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.config.RequestConfig.Builder;

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;

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

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<org.apache.http.NameValuePair>();

Set<Entry<String, String>> entrySet = params.entrySet();

for (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 (Entry<String, String> entry : headers.entrySet()) {

post.addHeader(entry.getKey(), entry.getValue());

}

}

// 设置参数

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

\*

\* @param url

\* @param charset

\* @param connTimeout 建立链接超时时间,毫秒.

\* @param readTimeout 响应超时时间,毫秒.

\* @return

\* @throws ConnectTimeoutException 建立链接超时

\* @throws SocketTimeoutException 响应超时

\* @throws Exception

\*/

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

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

\*

\* @param ressponse

\* @return

\*/

@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;

}

}

public static void main(String[] args) {

try {

String str= post("https://localhost:443/ssl/test.shtml","name=12&page=34","application/x-www-form-urlencoded", "UTF-8", 10000, 10000);

//String str= get("https://localhost:443/ssl/test.shtml?name=12&page=34","GBK");

/\*Map<String,String> map = new HashMap<String,String>();

map.put("name", "111");

map.put("page", "222");

String str= postForm("https://localhost:443/ssl/test.shtml",map,null, 10000, 10000);\*/

System.out.println(str);

} catch (ConnectTimeoutException e) {

e.printStackTrace();

} catch (SocketTimeoutException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

}

}

**常量的配置 用来获取yml文件中的APPID等**

import org.hibernate.validator.constraints.NotEmpty;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Configuration;

/\*\*

\* 常量配置类

\*/

@Configuration

@ConfigurationProperties(prefix = "constants")

public class Constants {

@NotEmpty

private String qqAppId;

@NotEmpty

private String qqAppSecret;

@NotEmpty

private String qqRedirectUrl;

@NotEmpty

private String weCatAppId;

@NotEmpty

private String weCatAppSecret;

@NotEmpty

private String weCatRedirectUrl;

public String getQqAppId() {

return qqAppId;

}

public void setQqAppId(String qqAppId) {

this.qqAppId = qqAppId;

}

public String getQqAppSecret() {

return qqAppSecret;

}

public void setQqAppSecret(String qqAppSecret) {

this.qqAppSecret = qqAppSecret;

}

public String getQqRedirectUrl() {

return qqRedirectUrl;

}

public void setQqRedirectUrl(String qqRedirectUrl) {

this.qqRedirectUrl = qqRedirectUrl;

}

public String getWeCatAppId() {

return weCatAppId;

}

public void setWeCatAppId(String weCatAppId) {

this.weCatAppId = weCatAppId;

}

public String getWeCatAppSecret() {

return weCatAppSecret;

}

public void setWeCatAppSecret(String weCatAppSecret) {

this.weCatAppSecret = weCatAppSecret;

}

public String getWeCatRedirectUrl() {

return weCatRedirectUrl;

}

public void setWeCatRedirectUrl(String weCatRedirectUrl) {

this.weCatRedirectUrl = weCatRedirectUrl;

}

}

**yml文件中的配置**

constants:

# QQ

qqAppId: xxxxxxxx

qqAppSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

qqRedirectUrl: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

#WECAT

weCatAppId: xxxxxxxxxx

weCatAppSecret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

weCatRedirectUrl: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**开始编写controller层, 我的demo没有前端的页面 所以只是纯后端的请求方式**

**第一步获取code**

@Autowired

private Constants constants;

@RequestMapping("getCode")

public String getCode() throws Exception {

//拼接url

StringBuilder url = new StringBuilder();

url.append("https://graph.qq.com/oauth2.0/authorize?");

url.append("response\_type=code");

url.append("&client\_id=" + constants.getQqAppId());

//回调地址 ,回调地址要进行Encode转码

String redirect\_uri = constants.getQqRedirectUrl();

//转码

url.append("&redirect\_uri="+ URLEncodeUtil.getURLEncoderString(redirect\_uri));

url.append("&state=ok");

String result = HttpClientUtils.get(url.toString(),"UTF-8");

System.out.println(url.toString());

return url.toString();

}

//上面的请求回返回一个url,然后进入到这个url中 QQ端会把调用你的回调函数,并把code一起传过来

**第二步 通过code取到token**

/\*\*

\* 获取token,该步骤返回的token期限为一个月

\* @param code

\* @return

\* @throws Exception

\*/

@RequestMapping("callback.do")

public String getAccessToken(String code) throws Exception {

if (code != null){

System.out.println(code);

}

StringBuilder url = new StringBuilder();

url.append("https://graph.qq.com/oauth2.0/token?");

url.append("grant\_type=authorization\_code");

url.append("&client\_id=" + constants.getQqAppId());

url.append("&client\_secret=" + constants.getQqAppSecret());

url.append("&code=" + code);

//回调地址

String redirect\_uri = constants.getQqRedirectUrl();

//转码

url.append("&redirect\_uri="+ URLEncodeUtil.getURLEncoderString(redirect\_uri));

String result = HttpClientUtils.get(url.toString(),"UTF-8");

System.out.println("url:" + url.toString());

//把token保存

String[] items = StringUtils.splitByWholeSeparatorPreserveAllTokens(result, "&");

String accessToken = StringUtils.substringAfterLast(items[0], "=");

Long expiresIn = new Long(StringUtils.substringAfterLast(items[1], "="));

String refreshToken = StringUtils.substringAfterLast(items[2], "=");

if (qqProperties.get("accessToken") != null){

qqProperties.remove("accessToken");

}

if (qqProperties.get("expiresIn") != null){

qqProperties.remove("expiresIn");

}

if (qqProperties.get("refreshToken") != null){

qqProperties.remove("refreshToken");

}

qqProperties.put("accessToken",accessToken);

qqProperties.put("expiresIn",expiresIn);

qqProperties.put("refreshToken",refreshToken);

return result;

}

上面这个controller就是在yml文件中配置的回调函数地址,我这边是把token存到一个map中了  
QQ的这个请求返回值是一个字符串 比如  
access\_token=FE04\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CCE2&expires\_in=7776000&refresh\_token=88E4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BE14  
这样子的 要取出来的话 需要做一下拆分,但是微信的是返回的json格式的 可以转为model保存

**第三步(可选) 上一步获取的token是有期限的,过期就会失效,这里提供了刷新token的方法**

/\*\*

\* 刷新token

\* @return

\* @throws Exception

\*/

@RequestMapping("refreshToken")

public String refreshToken() throws Exception {

StringBuilder url = new StringBuilder("https://graph.qq.com/oauth2.0/token?");

url.append("grant\_type=refresh\_token");

url.append("&client\_id=" + constants.getQqAppId());

url.append("&client\_secret=" + constants.getQqAppSecret());

//获取refreshToken

String refreshToken = (String) qqProperties.get("refreshToken");

url.append("&refresh\_token=" + refreshToken); // 该处需要传入上个步骤获取到的refreshToken;

String result = HttpClientUtils.get(url.toString(),"UTF-8");

System.out.println("url:" + url.toString());

//把新获取的token存到map中

String[] items = StringUtils.splitByWholeSeparatorPreserveAllTokens(result, "&");

String accessToken = StringUtils.substringAfterLast(items[0], "=");

Long expiresIn = new Long(StringUtils.substringAfterLast(items[1], "="));

String newRefreshToken = StringUtils.substringAfterLast(items[2], "=");

if (qqProperties.get("accessToken") != null){

qqProperties.remove("accessToken");

}

if (qqProperties.get("expiresIn") != null){

qqProperties.remove("expiresIn");

}

if (qqProperties.get("refreshToken") != null){

qqProperties.remove("refreshToken");

}

qqProperties.put("accessToken",accessToken);

qqProperties.put("expiresIn",expiresIn);

qqProperties.put("refreshToken",newRefreshToken);

return result;

}

**第四步,获取用户openId**

/\*\*

\* 获取用户openId

\* @return

\* @throws Exception

\*/

@RequestMapping("getOpenId")

public String getOpenId() throws Exception {

StringBuilder url = new StringBuilder("https://graph.qq.com/oauth2.0/me?");

//获取保存的用户的token

String accessToken = (String) qqProperties.get("accessToken");

if (!StringUtils.isNotEmpty(accessToken)){

return "未授权";

}

url.append("access\_token=" + accessToken);

String result = HttpClientUtils.get(url.toString(),"UTF-8");

String openId = StringUtils.substringBetween(result, "\"openid\":\"", "\"}");

System.out.println(openId);

//把openId存到map中

if (qqProperties.get("openId") != null) {

qqProperties.remove("openId");

}

qqProperties.put("openId",openId);

return result;

}

//这个步骤的正确返回值是callback( {"client\_id":"YOUR\_APPID","openid":"YOUR\_OPENID"} );  
也是一个字符串,需要进行拆分保存,这里请求传入的就是上次获取到的token

**第五步 获取用户信息**

/\*\*

\* 根据openId获取用户信息

\*/

@RequestMapping("getUserInfo")

public QQUserInfo getUserInfo() throws Exception {

StringBuilder url = new StringBuilder("https://graph.qq.com/user/get\_user\_info?");

//取token

String accessToken = (String) qqProperties.get("accessToken");

String openId = (String) qqProperties.get("openId");

if (!StringUtils.isNotEmpty(accessToken) || !StringUtils.isNotEmpty(openId)){

return null;

}

url.append("access\_token=" + accessToken);

url.append("&oauth\_consumer\_key=" + constants.getQqAppId());

url.append("&openid=" + openId);

String result = HttpClientUtils.get(url.toString(),"UTF-8");

Object json = JSON.parseObject(result,QQUserInfo.class);

QQUserInfo QQUserInfo = (QQUserInfo)json;

return QQUserInfo;

}

//传入token APPID,openId 就可以获取到用户信息 由于用到的这个工具类 返回的是一个字符串,可以转成object类型,再强转成model  
//到此授权就完成了,微信端和QQ不同的地方是微信获取token的时候会把openId一并获取到

**回调函数转码工具类**

import java.io.UnsupportedEncodingException;

public class URLEncodeUtil {

private final static String ENCODE = "UTF-8";

/\*\*

\* URL 解码

\*/

public static String getURLDecoderString(String str) {

String result = "";

if (null == str) {

return "";

}

try {

result = java.net.URLDecoder.decode(str, ENCODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

return result;

}

/\*\*

\* URL 转码

\*/

public static String getURLEncoderString(String str) {

String result = "";

if (null == str) {

return "";

}

try {

result = java.net.URLEncoder.encode(str, ENCODE);

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

return result;

}

}