

## 微信支付接口 For WindowsPhone 上手指南

SDK 的接口和使用请参见[微信开放平台上的上手指南（WindowsPhone）<sup>1</sup>](#)和[SDK Demo<sup>2</sup>](#)。这里主要说明在正常使用微信 SDK 的第三方 app 上使用微信支付接口的步骤。

### 准备工作：

在使用接口之前请先保证持有向微信开放平台申请得到的 appid、appsecret（长度为 32 的字符串，用于获取 access\_token）、appkey（长度为 128 的字符串，用于支付过程中生成 app\_signature）及 partnerkey（微信公众平台商户模块生成的商户密钥）。

**注意：**appsecret、appkey、partnerkey 不应硬编码到客户端程序中，建议需要用到这三个字段的过程都在服务器端完成。

### 一、获取 access\_token

access\_token 是 APP 的全局唯一票据，APP 调用各接口时都需使用 access\_token。正常情况下 access\_token 有效期为 7200 秒，重复获取将导致上次获取的 access\_token 失效。

APP 可以使用 AppID 和 AppSecret 调用本接口来获取 access\_token。AppID 和 AppSecret 可在开放平台后台获得。注意调用接口时需使用 https 协议。

#### 接口调用请求说明

http 请求方式：GET  
[https://api.weixin.qq.com/cgi-bin/token?grant\\_type=client\\_credential&appid=APPID&secret=APPSECRET](https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET)

#### 参数说明：

参数	是否必须	说明
grant_type	是	获取 access_token，此处填写 client_credential
appid	是	APP 唯一凭证
secret	是	应用密钥，在微信开放平台提交应用审核通过后获得

#### 返回说明

正常情况下，微信会返回下述 JSON 数据包给开发者：

{"access_token":"ACCESS_TOKEN","expires_in":7200}	
参数	说明

<sup>1</sup>

<sup>2</sup>

access_token	获取到的凭证
expires_in	凭证有效时间，单位：秒。正常情况下 access_token 有效期为 7200 秒，重复获取将导致上次获取的 access_token 失效。

错误时微信会返回错误码等信息，JSON 数据包示例如下（该示例为 AppID 无效错误）：

```
{"errcode":40013,"errmsg":"invalid appid"}
```

## 二、生成预支付订单

用第一步请求的 access\_token 作为参数，通过微信开放平台接口生成预支付订单。

http 请求方式：POST

[https://api.weixin.qq.com/pay/genprepay?access\\_token=ACCESS\\_TOKEN](https://api.weixin.qq.com/pay/genprepay?access_token=ACCESS_TOKEN)

Url 中的参数只包含目前微信公众平台凭证 access\_token，详细的订单数据放在 PostData 中，格式为 json，示例如下：

```
{
  "appid": "wwwb4f85f3a797777",
  "traceid": "crestxu",
  "noncestr": "11111222233333",
  "package": "bank_type=WX&body=XXX&fee_type=1&input_charset=GBK&notify_url=http%3a%2f%2fwww.qq.com&out_trade_no=16642817866003386000&partner=1900000109&spbill_create_ip=127.0.0.1&total_fee=1&sign=BEEF37AD19575D92E191C1E4B1474CA9",
  "timestamp": 1381405298,
  "app_signature": "53cca9d47b883bd4a5c85a9300df3da0cb48565c",
  "sign_method": "sha1"
}
```

其中，各字段含义如下：

参数	是否必须	说明
appid	是	应用唯一标识，在微信开放平台提交应用审核通过后获得
traceid	否	商家对用户的唯一标识，如果用微信 SS0，此处建议填写授权用户的 openid
noncestr	是	32位内的随机串，防重发
package	是	订单详情（具体生成方法见后文）
timestamp	是	时间戳，为 1970 年 1 月 1 日 00:00 到请求发起时间的秒数
app_signature	是	签名（具体生成方法见后文）

sign_method	是	加密方式，默认为 sha1
-------------	---	---------------

返回结果说明：

正确的 Json 返回示例：

```
{"prepayid":"PREPAY_ID","errcode":0,"errmsg":"Success"}
```

错误的 Json 返回示例：

```
{"errcode":48001,"errmsg":"api unauthorized"}
```

**package 生成方法：**

A) 对所有传入参数按照字段名的 ASCII 码从小到大排序（字典序）后，使用 URL 键值对的格式（即 key1=value1&key2=value2...）拼接成字符串 string1；

B) 在 string1 最后拼接上 key=partnerKey 得到 stringSignTemp 字符串，并对 stringSignTemp 进行 md5 运算，再将得到的字符串所有字符转换为大写，得到 sign 值 signValue。

C) 对 string1 中的所有键值对中的 value 进行 urlencode 转码，按照 a 步骤重新拼接成字符串，得到 string2。对于 js 前端程序，一定要使用函数 encodeURIComponent 进行 urlencode 编码（注意！进行 urlencode 时要将空格转化为%20 而不是+）。

D) 将 sign=signValue 拼接到 string1 后面得到最终的 package 字符串。

代码示例如下：

```
// 构造参数列表
string productName = "千足金箍棒";
string productPrice = "1";
Dictionary<string, string> args = new Dictionary<string, string>();
args.Add("input_charset", "UTF-8");
args.Add("partner", ConstString.PARTNER_ID);
args.Add("spbill_create_ip", "196.168.1.1");
args.Add("notify_url", "http://weixin.qq.com");
args.Add("OutPutType", "JSON");//这个都要大写
args.Add("fee_type", "1");
args.Add("out_trade_no", Util.getNowSeconds().ToString());
args.Add("body", productName);
args.Add("total_fee", productPrice);

string originalString = Util.stringWithFormEncodedComponentsAscending(args, true,
true, "&", false, false);

//【财付通签名原串】
string signText = originalString + "&key=" + partnerKey;
string sign = MD5Core.GetHashString(signText).ToUpper();

string escapeArguments = Util.stringWithFormEncodedComponentsAscending(arguments,
true, true, "&", true, false);
tring package = escapeArguments + "&sign=" + sign;
```

#### app\_signature 生成方法:

A) 参与签名的字段包括: appid、appkey、noncestr、package、timestamp 以及 traceid

B) 对所有待签名参数按照字段名的 ASCII 码从小到大排序 (字典序) 后, 使用 URL 键值对的格式 (即 key1=value1&key2=value2...) 拼接成字符串 string1。

注意: 所有参数名均为小写字符

C) 对 string1 作签名算法, 字段名和字段值都采用原始值, 不进行 URL 转义。具体签名算法为 SHA1

代码示例如下:

```
mReqRict = new Dictionary<string, string>();
mReqRict.Add("package", package);
mReqRict.Add("appid", ConstString.APPID);
mReqRict.Add("traceid", "traceid");
mReqRict.Add("noncestr",
MD5Core.GetHashString(Util.getNowSeconds().ToString()).ToLower()); //这个要小写
mReqRict.Add("timestamp", Util.getNowSeconds().ToString());
mReqRict.Add("appkey", ConstString.SECRET_KEY);

string appOriginalString = Util.stringWithFormEncodedComponentsAscending(mReqRict,
true, true, "&", false, false);
string appSign = Util.Sha1(appOriginalString);
```

### 三、调起微信支付

将第二步生成的 prepayId 作为参数, 调用微信 sdk 发送支付请求到微信。

代码示例如下:

```
//拉起微信支付
private void btnPay_Click(object sender, RoutedEventArgs e)
{
    if (mReqDict == null || mReqDict.Count == 0)
    {
        MessageBox.Show("please get prepayid first");
        return;
    }

    string AppID = mReqDict["appid"];
    string NonceStr = mReqDict["noncestr"];
    string Package = mReqDict["packageforsdk"];
    string PartnerId = ConstString.PARTNER_ID;
    string PrepayId = mReqDict["prepayid"];
    string Sign = mReqDict["signsha1"];
    uint TimeStamp = uint.Parse(mReqDict["timestamp"]);

    SendPay.Reg req = new SendPay.Reg(PartnerId, PrepayId, NonceStr, TimeStamp,
    Package, Sign);
    IWXAPI api = WXAPIFactory.CreateWXAPI(AppID);
    api.SendReq(req);
}
```

注意事项:

1. 调起微信支付 SDK 时, 请求参数中 package 需填写为: **Sign=WXPAY**。

#### 四、接收支付返回结果:

参照微信 SDK\_Demo, 在 WXEntryBasePage 子类中实现 On\_SendPay\_Response 函数, 处理支付结果的通知和下一步界面操作。注意由客户端返回的支付结果不能作为最终支付的可信结果, 应以服务器端的支付结果通知为准。

代码示例如下:

```
/// <param name="response">SendPay的响应数据</param>
public override void On_SendPay_Response(SendPay.Resp response)
{
    if (response != null && response.ErrCode == ErrorCode.ERR_USER_CANCEL)
    {
        MessageBox.Show("支付取消");
    }
    else if (response == null || response.ErrCode != ErrorCode.ERR_OK)
    {
        MessageBox.Show("支付失败");
    }
    else
    {
        MessageBox.Show("payfinished, ReturnKey = " + response.ReturnKey);
    }

    if (NavigationService.CanGoBack)
    {
        NavigationService.GoBack();
    }
    else
    {
        NavigationService.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));
        NavigationService.RemoveBackEntry();
    }
}
```

#### 附:微信 SDK 更新说明

这里主要说明新增加的支付接口。

- 1、增加支付请求数据结构体: PayReq。

```
/// <summary>
/// <div class="en">
/// 商家向财付通申请的商家id
/// </div>
///
///<div class="zh_CN">
///商家向财付通申请的商家id
///</div>
/// </summary>
```

```

public string PartnerId;

/// <summary>
/// <div class="en">
/// 预支付订单
/// </div>
///
/// <div class="zh_CN">
/// 预支付订单
/// </div>
/// </summary>
public string PrepayId;

/// <summary>
/// <div class="en">
/// 随机串，防重发
/// </div>
///
/// <div class="zh_CN">
/// 随机串，防重发
/// </div>
/// </summary>
public string NonceStr;

/// <summary>
/// <div class="en">
/// 时间戳，防重发
/// </div>
///
/// <div class="zh_CN">
/// 时间戳，防重发
/// </div>
/// </summary>
public uint TimeStamp;

/// <summary>
/// <div class="en">
/// 商家根据财付通文档填写的数据和签名
/// </div>
///
/// <div class="zh_CN">
/// 商家根据财付通文档填写的数据和签名
/// </div>
/// </summary>
public string Package;

/// <summary>
/// <div class="en">
/// 商家根据微信开放平台文档对数据做的签名
/// </div>
///
/// <div class="zh_CN">
/// 商家根据微信开放平台文档对数据做的签名
/// </div>
/// </summary>
public string Sign;

```

```
/// <summary>
/// <div class="en">Build an empty request</div>
/// <div class="zh_CN">构造空请求</div>
/// </summary>
```

2、增加支付结果数据结构体：PayResp。

```
/**
 * <div class="en">财付通返回给商家的信息</div>
 * <div class="zh_CN">财付通返回给商家的信息</div>
 */
public string ReturnKey;
```

## 后台通知接口说明

### 通知接口简介

用户在成功完成支付后，微信后台通知（post）商户服务器（notify\_url）支付结果。商户可以使用 notify\_url 的通知结果进行个性化页面的展示。

### 补单机制

对后台通知交互时，如果微信收到商户的应答不是 success 或超时，微信认为通知失败，微信会通过一定的策略（如 30 分钟共 8 次）定期重新发起通知，尽可能提高通知的成功率，但微信不保证通知最终能成功。

由于存在重新发送后台通知的情况，因此同样的通知可能会多次发送给商户系统。商户系统必须能够正确处理重复的通知。

微信推荐的做法是，当收到通知进行处理时，首先检查对应业务数据的状态，判断该通知是否已经处理过，如果没有处理过再进行处理，如果处理过直接返回 success。在对业务数据进行状态检查和处理之前，要采用数据锁进行并发控制，以避免函数重入造成的数据混乱。

目前补单机制的间隔时间为：8s、10s、10s、30s、30s、60s、120s、360s、1000s。

## 通知接口参数

后台通知通过请求中的 notify\_url 进行，采用 post 机制。返回通知中的参数一致，url 包含如下内容：

字段名	变量名	必填	类型	说明
协议参数				
签名方式	sign_type	否	String(8)	签名类型，取值：MD5、RSA，默认：MD5
接口版本	service_version	否	String(8)	版本号，默认为 1.0
字符集	input_charset	否	String(8)	字符编码，取值：GBK、UTF-8，默认：GBK。
签名	sign	是	String(32)	签名
密钥序号	sign_key_index	否	Int	多密钥支持的密钥序号，默认 1
业务参数				
交易模式	trade_mode	是	Int	1-即时到账 其他保留
交易状态	trade_state	是	Int	支付结果： 0-成功 其他保留
支付结果信息	pay_info	否	String(64)	支付结果信息，支付成功时空
商户号	partner	是	String(10)	商户号，也即之前步骤的 partnerid, 由微信统一分配的 10 位正整数 (120XXXXXXX) 号
付款银行	bank_type	是	String(16)	银行类型，在微信中使用 WX
银行订单号	bank_billno	否	String(32)	银行订单号
总金额	total_fee	是	Int	支付金额，单位为分，如果 discount 有值，通知的 total_fee + discount = 请求的 total_fee
币种	fee_type	是	Int	现金支付币种，目前只支持人民币，默认值是 1-人民币
通知 ID	notify_id	是	String(128)	支付结果通知 id，对于某



				些特定商户，只返回通知 id，要求商户据此查询交易结果
订单号	transaction_id	是	String(28)	交易号，28 位长的数值，其中前 10 位为商户号，之后 8 位为订单产生的日期，如 20090415，最后 10 位是流水号。
商户订单号	out_trade_no	是	String(32)	商户系统的订单号，与请求一致。
商家数据包	attach	否	String(127)	商家数据包，原样返回
支付完成时间	time_end	是	String(14)	支付完成时间，格式为 yyyyMMddhhmmss，如 2009 年 12 月 27 日 9 点 10 分 10 秒表示为 20091227091010。时区为 GMT+8 beijing。
物流费用	transport_fee	否	Int	物流费用，单位分，默认 0。如果有值，必须保证 $\text{transport\_fee} + \text{product\_fee} = \text{total\_fee}$
物品费用	product_fee	否	Int	物品费用，单位分。如果有值，必须保证 $\text{transport\_fee} + \text{product\_fee} = \text{total\_fee}$
折扣价格	discount	否	Int	折扣价格，单位分，如果有值，通知的 $\text{total\_fee} + \text{discount} =$ 请求的 $\text{total\_fee}$
买家别名	buyer_alias	否	String(64)	对应买家账号的一个加密串

同时，在 postData 中还将包含 xml 数据。数据如下：

```
<xml>
  <OpenId><![CDATA[111222]]></OpenId>
  <AppId><![CDATA[wwwb4f85f3a79777]]></AppId>
  <IsSubscribe>1</IsSubscribe>
  <TimeStamp> 1369743511</TimeStamp>
  <NonceStr><![CDATA[jAL1dRTHAFd5Tgs5]]></NonceStr>
```

```

    <AppSignature><![CDATA[bafe07f060f22dcda0bfdb4b5ff756f973aecffa]]>
  </AppSignature>
  <SignMethod><![CDATA[sha1]]></ SignMethod >
</xml>

```

各字段定义如下：

参数	必填	说明
AppId	是	字段名称：公众号 id；字段来源：商户注册具有支付权限的公众号成功后即可获得；传入方式：由商户直接传入。
TimeStamp	是	字段名称：时间戳；字段来源：商户生成从 1970 年 1 月 1 日 00:00:00 至今的秒数，即当前的时间；由商户生成后传入。取值范围：32 字符以下
NonceStr	是	字段名称：随机字符串；字段来源：商户生成的随机字符串；取值范围：长度为 32 个字符以下。由商户生成后传入。取值范围：32 字符以下
OpenId	是	支付该笔订单的用户 ID，商户可通过公众号其他接口为付款用户服务。
AppSignature	是	字段名称：签名；字段来源：对前面的其他字段与 appKey 按照字典序排序后，使用 SHA1 算法得到的结果。由商户生成后传入。
IsSubscribe	是	用户是否关注了公众号。1 为关注，0 为未关注。

AppSignature 依然是根据前文 paySign 所述的签名方式生成，参与签名的字段为：appid、appkey、timestamp、noncestr、openid、issubscribe。

从以上信息可以看出，url 参数中携带订单相关信息，postData 中携带该次支付的用户相关信息，这将便于商家拿到 openid，以便后续提供更好的售后服务。

## 后台通知结果返回

微信后台通过 notify\_url 通知商户，商户做业务处理后，需要以字符串的形式反馈处理结果，内容如下：

返回结果	结果说明
success	处理成功，微信系统收到此结果后不再进行后续通知
fail 或其它字	处理不成功，微信收到此结果或者没有收到任何结果，系统通

符	过补单机制再次通知
---	-----------

## 后台通知签名方式

对于 url 中签名原始串按以下方式组装成字符串：

- 除 sign 字段外，所有参数按照字段名的 `ascii` 码从小到大排序后使用 `QueryString` 的格式（即 `key1=value1&key2=value2...`）拼接而成字符串 `string1`，空值不传递，不参与签名组串。
- 在 `string1` 最后拼接上 `key=paternerKey` 得到 `stringSignTemp` 字符串，并对 `stringSignTemp` 进行 `md5` 运算，再将得到的字符串所有字符转换为大写，得到 `sign` 值 `signValue`。
- 对 `string1` 进行 `urlencode` 转码，得到 `string2`。
- 将 `sign=signValue` 拼接到 `string1` 后面得到最终的 `notifyargs` 字符串。
- 所有参数是指通信过程中实际出现的所有非空参数，即使是接口中无描述的字段，也需要参与签名组串。
- 签名原始串中，字段名和字段值都采用原始值，不进行 `URL Encode`。
- 微信通知消息可能会由于升级增加参数，请验证应答签名时注意允许这种情况。

下面定义了一段生成 `notifyargs` 字符串的示范过程：

```

假设以下为 notifyargs 传入参数：
bank_billno=206064184488,
bank_type=0,
discount=0,
fee_type=1,
input_charset=GBK,
notify_id=WE37gwCoFBcAKdkH34YlnW94r_vao2l jmwE3oAHEeAP690xSVhR1eOMfhs
gJwVGdpluT-vdS79kbDbkDnjYg4qsmTdSjuJxl,
out_trade_no=843254536943809900,
partner=1900000109,
product_fee=1,
sign_type=MD5,
time_end=20130606015331,
total_fee=1,
trade_mode=1,
trade_state=0,
transaction_id=1900000109201306060282555397,
transport_fee=0

i: 经过 a 过程 url 键值对字典序排序后的字符串 string1 为：

```

```
bank_billno=206064184488&bank_type=0&discount=0&fee_type=1&input_charset=GBK&notify_id=WE37gwCoFBcAKdkH34Y1nW94r_vao21jmwE3oAHEeAP690xSVhRleOMfhsgjwVGdpluT-  
vdS79kbDbkDnjYg4qsmTdSjuJxl&out_trade_no=843254536943809900&partner=1900000109&product_fee=1&sign_type=MD5&time_end=20130606015331&total_fee=1&trade_mode=1&trade_state=0&transaction_id=1900000109201306060282555397&transport_fee=0
```

ii: 经过 b 过程后得到 sign 为:

```
sign  
=  
md5(string1&key=8934e7d15453e97507ef794cf7b0519d).toUpperCase  
=  
md5(bank_billno=206064184488&bank_type=0&discount=0&fee_type=1&input_charset=GBK&notify_id=WE37gwCoFBcAKdkH34Y1nW94r_vao21jmwE3oAHEeAP690xSVhRleOMfhsgjwVGdpluT-  
vdS79kbDbkDnjYg4qsmTdSjuJxl&out_trade_no=843254536943809900&partner=1900000109&product_fee=1&sign_type=MD5&time_end=20130606015331&total_fee=1&trade_mode=1&trade_state=0&transaction_id=1900000109201306060282555397&transport_fee=0&key=8934e7d15453e97507ef794cf7b0519d).toUpperCase()  
=  
" 8ef1f69d5d9d4ec39d3787526f27924e".toUpperCase()  
=  
" 8EF1F69D5D9D4EC39D3787526F27924E"
```

iii: 再对 string1 经过 c 过程 urlencode 编码后得到:

```
bank_billno=206064184488&bank_type=0&discount=0&fee_type=1&input_charset=GBK&notify_id=WE37gwCoFBcAKdkH34Y1nW94r_vao21jmwE3oAHEeAP690xSVhRleOMfhsgjwVGdpluT-  
vdS79kbDbkDnjYg4qsmTdSjuJxl&out_trade_no=843254536943809900&partner=1900000109&product_fee=1&sign_type=MD5&time_end=20130606015331&total_fee=1&trade_mode=1&trade_state=0&transaction_id=1900000109201306060282555397&transport_fee=0
```

iv: 拼接上 sign 后得到最终 package 结果:

```
bank_billno=206064184488&bank_type=0&discount=0&fee_type=1&input_charset=GBK&notify_id=WE37gwCoFBcAKdkH34Y1nW94r_vao21jmwE3oAHEeAP690xSVhRleOMfhsgjwVGdpluT-  
vdS79kbDbkDnjYg4qsmTdSjuJxl&out_trade_no=843254536943809900&partner=1900000109&product_fee=1&sign_type=MD5&time_end=20130606015331&total_fee=1&
```

trade\_mode=1&trade\_state=0&transaction\_id=1900000109201306060282555397&transport\_fee=0&sign=8EF1F69D5D9D4EC39D3787526F27924E