腾讯慧眼微信支付预授权

1.1 实名认证预授权对外接口

1) 接口

```
https://iauth-sandbox.wecity.qq.com/new/cgi-bin/preauth.php
```

2) 描述

```
实名认证流程预授权接口。
```

3) 方法

```
POST
```

4) HTTP请求格式

```
a. 头部信息
```

要求	参数名	类型	参数说明
必选	signature	String	接口签名，具体见签名算法

```
b. 请求包体
```

要求	参数名	类型	参数说明
必选	appid	String	分配的appid
必选	uid	String	客户流水号（订单号），最后结果中会返回给客户，用于绑定和校验
必选	ID	String	身份证号
必选	name	String	姓名
必选	redirect	String	回调地址 最终验证接口回调的地址

```
c. 包体示例
```

```
{
    "appid": "xxxx",
    "uid" : "xxxxxxxx",
    "ID": "450111199408082135",
    "name": "张三",
    "redirect":"xxxxxxxxxxxx"
}
```

5) 返回值说明

a. 返回主体包内容

字段	类型	说明
errorcode	int	返回状态码,0表示成功，非0值为出错
errormsg	String	返回错误描述
data	json	auth_uri 授权调用的url

b. 示例

```
{
    "errorcode": 0,
    "errormsg": "成功",
    "data": {
    "auth_uri": "https://xxx/cgi-bin/xxx.php?appid=xxx&authcode=xxx”
    }
}
```

6) 回调地址数据说明

a. 头部信息

要求	参数名	类型	参数说明
必选	token	String	用户实名信息凭证
必选	uid	String	请求时传参的uid 客户流水号（订单号），在回调地址中会带回

b. 返回示例

```
redirect?token=xxx&uid=xxx
```

1.2 实名信息拉取接口

1) 接口

```
https://iauth-sandbox.wecity.qq.com/new/cgi-bin/getdetectinfo.php
```

2) 描述

```
拉取实名详细信息接口。回包内容已加密，详细算法参看 6.加解密算法。
```

3) 方法

```
POST
```

4) HTTP请求格式

```
a. 头部信息
```

要求	参数名	类型	参数说明
必选	signature	String	接口签名，具体见签名算法

```
b. 请求包体
```

要求	参数名	类型	参数说明
必选	token	String	回调返回的token序列号
必选	appid	String	分配的appid

```
c. 包体示例
```

```
{
    "token":"{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}",
    "appid":"xxxx"
}
```

5) 返回值说明

```
a. 返回主体包内容
```

字段	类型	说明
errorcode	int	返回状态码,0表示成功，非0值为出错
errormsg	String	返回错误描述
data	String	AES密文

```
b. 示例
```

```
{
    "errorcode": 0,
    "errormsg": "success",
    "data":"AES密文"
}
```

aes解密data后对应的数据如下：

字段	类型	说明
yt_errorcode	String	验证结果错误码 0表示成功，非0值为出错
yt_errormsg	String	验证结果错误信息
uid	String	客户流水号（订单号） 用来校验是否与回调一致

示例

```
{
    "yt_errorcode": "xxx",
    "yt_errormsg": "xxx",
    "uid": "xxxxxxx"
}
```

4 鉴权算法

4.1 签名

上述提供的API接口，通过签名来验证请求的合法性。开发者通过将签名授权给调用方，使其具备使用API接口的能力。密钥的获取及签名的生成方法如下：

Step 1 获取应用appid、密钥secretkey和过期时间expired

```
应用appid：用来标识唯一的应用；
密钥secretkey：使用加密算法时使用的秘钥；
expired：当次请求的时间戳的有效时间；
```

Step 2 拼接有效签名串

```
a=xxxxx&m=xxxxxxx&t=1427786065&e=600
a为appid
m为调用的api名，
t为当前时间戳，是一个符合UNIX Epoch时间戳规范的数值，单位为秒
e为此签名的凭证有效期，是一个符合UNIX Epoch时间戳规范的数值，单位为秒，
同appid和secretkey一样，由API提供方给定。
拼接有效签名串的结果,下文称之为orignal
```

Step 3 生成签名串

(1)API提供方 使用 HMAC-SHA1 算法对请求进行签名。
(2)签名串需要使用 Base64 编码。
根据签名方法signature= Base64(HMAC-SHA1(secretkey, orignal) + original)，其中secretkey为Step1获取，orignal为Step2中拼接好的签名串，对orignal使用HMAC-SHA1算法进行签名，然后将orignal附加到签名结果的末尾，再进行Base64编码，得到最终的sign。

注：此处使用的是标准的Base64编码，不是urlsafe的Base64编码，请注意。 以 JAVA 语言为例,其他语言均有类似算法. JAVA语言的示例代码见附件。

Step 4 使用签名串

将Step 3生成的signature，填充到http请求的head头部的signature字段中即可。

NodeJS参考代码

```
var crypto = require('crypto');
var signExpired = 600;//有效期一般为600s
//生成签名
function getAppSign(apiName, appId, appSecretKey, signExpired) {
    if (!apiName || !appId || !appSecretKey || !signExpired)
        return '';
    var now = parseInt(Date.now() / 1000);
    var plainText = 'a=' + appId + '&m=' + apiName + '&t=' + now + '&e=' + signExpired;
    var data = new Buffer(plainText, 'utf8');
    var res = crypto.createHmac('sha1', appSecretKey).update(data).digest();
    var bin = Buffer.concat([res, data]);
    var sign = bin.toString('base64');
    return sign;
}
```

5 错误码

1) 成功

取值	说明
"0"	SUCESS 成功

2) 错误码说明

取值	说明
"1"	HTTP_UNSET_PARAM 请求不合法，缺少参数
"2"	HTTP_WRONG_PARAM 请求不合法，参数错误
"3"	HTTP_UNAUTHORIZED 权限验证失败
"4"	HTTP_NOT_SIGNATURE 请求不合法，缺少签名
"5"	HTTP_REQUES_ILLEGAL 业务请求不合法
"7"	"未登录"
"8"	"系统错误"
"9"	"未授权接口"
"10"	"超过调用限制"
"11"	"无效请求"
"12"	"token超过有效期"
"13"	"非法转跳URL"

DB错误

取值	说明
"201"	没有找到数据
"202"	数据插入异常
"203"	数据更新异常

实名认证错误

取值	说明
"901"	实名认证失败

6 加解密算法

6.1 AES 256算法

使用项目提供的AES解密秘钥解密

NodeJS 代码参考

AES-256-ECB + PKCS7

```
function encryptAes256ECBPKCS7(data, secretKey) {
    try {
        let iv = "";
        var clearEncoding = 'utf8';
        var cipherEncoding = 'base64';
        var cipherChunks = [];
        var cipher = crypto.createCipheriv('aes-256-ecb', secretKey, iv);
        cipher.setAutoPadding(true);
        cipherChunks.push(cipher.update(data, clearEncoding, cipherEncoding));
        cipherChunks.push(cipher.final(cipherEncoding));
        return cipherChunks.join('');
    } catch (e) {
        console.error(e);
        return "";
        }
}
function decryptAes256ECBPKCS7(data, secretKey) {
    try {
    if (!data) {
        return "";
        }
        let iv = "";
        var clearEncoding = 'utf8';
        var cipherEncoding = 'base64';
        var cipherChunks = [];
        var decipher = crypto.createDecipheriv('aes-256-ecb', secretKey, iv);
        decipher.setAutoPadding(true);
        let buff = data.replace('\r', '').replace('\n', '');
        cipherChunks.push(decipher.update(buff, cipherEncoding,clearEncoding));
        cipherChunks.push(decipher.final(clearEncoding));
        return cipherChunks.join('');
        } catch (e) {
            console.error(e);
            return "";
            }
}
```

PHP 代码参考

AES-256-ECB + PKCS7（由于 PHP 底层的 256 和 Nodejs、Java 的不一样。所以 PHP 使用的是 **128** 长度）

```php
/**
 * 利用mcrypt做AES加密解密
 */
class AES{
    /**
     * 算法,另外还有192和256两种长度
     */
    const CIPHER = MCRYPT_RIJNDAEL_128;
    /**
     * 模式
     * 1. MCRYPT_MODE_ECB(electronic codebook)
     适合对小数量随机数据的加密，比如加密用户的登录密码之类的。
     * 2. MCRYPT_MODE_CBC (cipher block chaining)
     适合加密安全等级较高的重要文件类型。
     * 3. MCRYPT_MODE_CFB (cipher feedback)
     适合于需要对数据流的每一个字节进行加密的场合。
     * 4. MCRYPT_MODE_OFB (output feedback, in 8bit) 和CFB模式兼容，但比C
     FB模式更安全。CFB模式会引起加密的错误扩散，如果一个byte出错，则其后
     续的所有byte都会出错。OFB模式则不会有此问题。但该模式的安全度不是很
     高，不建议使用。
     * 5. MCRYPT_MODE_NOFB (output feedback, in nbit)
     和OFB兼容，由于采用了块操作算法，安全度更高。
     * 6. MCRYPT_MODE_STREAM
     是为了WAKE或者RC4等流加密算法提供的额外模型。
     */
    const MODE = MCRYPT_MODE_ECB;

    /**
     * pkcs7补码
     * @param string $string  明文
     * @param int $blocksize Blocksize , 以 byte 为单位
     * @return String
     */
    private function addPkcs7Padding($string, $blocksize = 16) {
        $len = strlen($string); //取得字符串长度
        $pad = $blocksize - ($len % $blocksize); //取得补码的长度
        $string .= str_repeat(chr($pad), $pad); //用ASCII码为补码长度的字符， 补足最后一段
        return $string;
    }

    /**
     * 加密然后base64转码
     * @param $str
     * @param $key
     * @return string
     */
    function aes256cbcEncrypt($str,$key ) {
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return base64_encode(mcrypt_encrypt(self::CIPHER, $key, $this->addPkcs7Padding($str) ,
self::MODE, $iv));
    }
```

```
    /**
     * 除去pkcs7 padding
     *
     * @param String 解密后的结果
     *
     * @return String
     */
    private function stripPkcs7Padding($string){
        $slast = ord(substr($string, -1));
        $slastc = chr($slast);
        $pcheck = substr($string, -$slast);

        if(preg_match("/$slastc{".$slast."}/", $string)){
            $string = substr($string, 0, strlen($string)-$slast);
            return $string;
        } else {
            return false;
        }
    }
    /**
     * 解密
     * @param String $encryptedText 二进制的密文
     * @param String $key 密钥
     * @return String
     */
    function aes256cbcDecrypt($encryptedText, $key) {
        $encryptedText =base64_decode($encryptedText);
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return $this->stripPkcs7Padding(mcrypt_decrypt(self::CIPHER, $key, $encryptedText,
self::MODE, $iv));
    }
}
```