
WALL·E

Table of Contents

微信NodeJs机器人搭建

本书内容整理自webot以及微信接口

最快速简介的方式 安装 `weixin-robot`

1. weixin-robot

   ```
   weixin-robot 是 webot 和 wechat-mp 的 高级包装。
   webot 负责定义回复规则，wechat-mp 负责与微信服务器通信。
   ```

2. webot

   ```
   基于正则何自定义函数的简单规则，你可以很容易的运行一个网络服务。
   ```

3. wechat-mp

   ```
   校验签名，接受并解析微信消息，处理回复内容为 XML ，并回复给微信。
   ```

4. wechat-api

   ```
   本模块只负责与 `access_token` 有关的高级接口功能
   ```

5. webot-cli

   ```
   weixin-robot 的命令行接口
   ```

6. wechat

   ```
   微信公共平台消息接口服务中间件与API SDK
   ```

7. 接口文档

> node-webot/weixin-robot

微信公共帐号机器人(Weixin Robot)

一个微信的node.js机器人。

微信公众平台提供的开放信息接口的自动回复系统。

`weixin-robot` 是 webot 和 wechat-mp 的高级包装。`webot` 负责定义回复规则，`wechat-mp` 负责与微信服务器通信。

注意

> 注意： 现在如果要启用session支持, `webot.watch` 必须在 `app.use(connect.session())` 之前

功能特色

1. 方便灵活的规则定义，轻松实现文本匹配流程控制
2. 基于正则表达式的对话设定，配置简单，可以给一句话随机回复不同内容
3. 支持等待后续操作模式，如可以提示用户“需要我执行xxx操作吗？”
4. 可直接从 yaml 或 json 文件中载入对话规则

许可证

快速入门

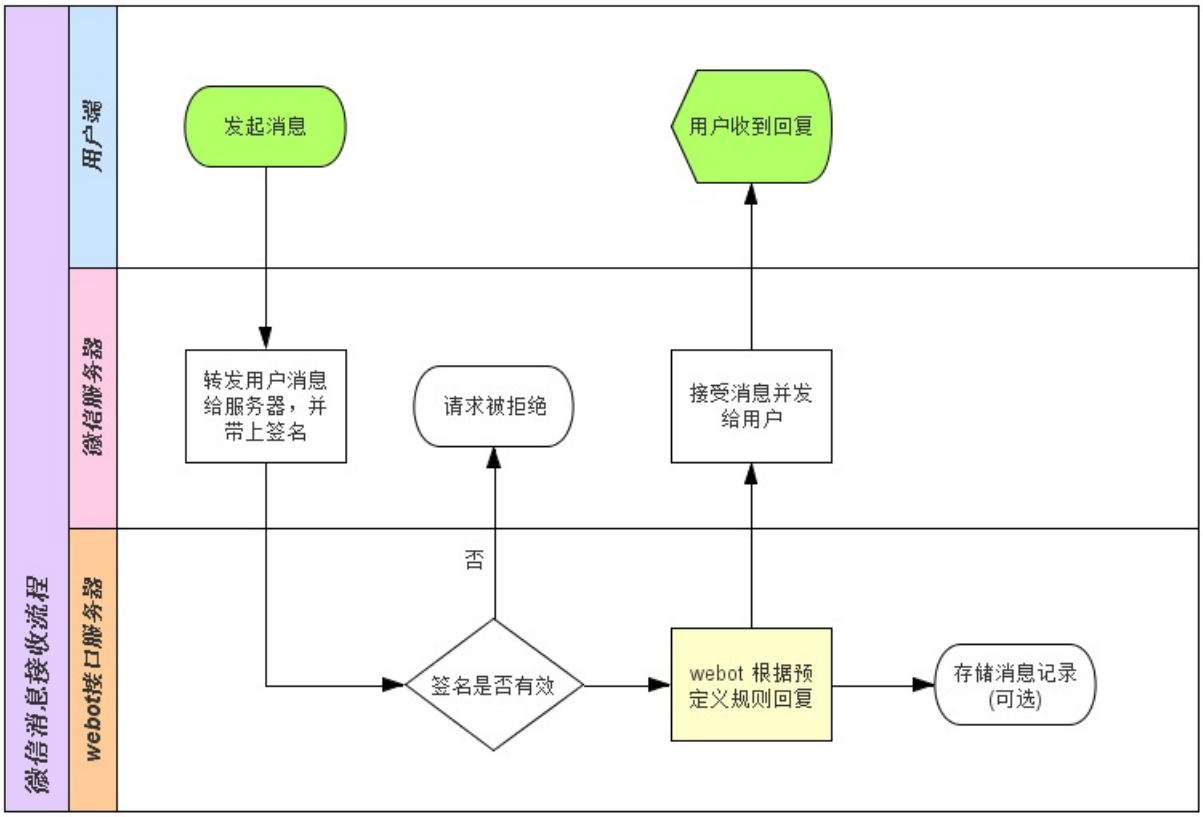
```
var express = require('express');
var webot = require('weixin-robot');

var app = express();

// 指定回复消息
webot.set('hi', '你好');

webot.set('subscribe', {
  pattern: function(info) {
    return info.is('event') && info.param.event === 'subscribe';
  },
  handler: function(info) {
    return '欢迎订阅微信机器人';
  }
});

webot.set('test', {
  pattern: /^test/i,
  handler: function(info, next) {
    next(null, 'roger that!')
  }
})

// 你可以获取已定义的 rule
//
// webot.get('subscribe') ->
//
// {
//   name: 'subscribe',
//   pattern: function(info) {
//     return info.is('event') && info.param.event === 'subscribe';
//   },
//   handler: function(info) {
//     return '欢迎订阅微信机器人';
//   }
// }
//

// 接管消息请求
webot.watch(app, { token: 'your1weixin2token', path: '/wechat' });

// 如果需要多个实例（即为多个微信账号提供不同回复）：
var webot2 = new webot.Webot();
webot2.set({
  '/hi/i': 'Hello',
  '/who (are|r) (you|u)/i': 'I\'m a robot.'
});
webot2.watch(app, {
  token: 'token2',
  path: '/wechat_en', // 这个path不能为之前已经监听过的path的子目录
});

// 启动 Web 服务
// 微信后台只允许 80 端口
app.listen(80);

// 如果你不想让 node 应用直接监听 80 端口
// 可以尝试用 nginx 或 apache 自己做一层 proxy
// app.listen(process.env.PORT);
// app.enable('trust proxy');
```

然后你就可以在微信公众平台后台填入你的接口地址和 token， 或者使用 webot-cli 来调试消息。

如果一切顺利，你也搭建好了自己的机器人，欢迎到此项目的 Wiki 页面添加你的帐号。

API 参考

微信自动回复API流程图



规则定义

> 具体的规则定义部分，请参考 webot 的文档。

主要API:

- webot.set()
- webot.waitRule()
- webot.loads()

info 对象

webot rule 的 handler 接收到的 info 对象，包含请求消息内容和 session 支持。

请求消息属性

`wexin-robot` 的 `info` 把微信的请求内容包装为了更符合 js 命名规则的值，并根据 `MsgType` 的不同， 将额外参数存入了 `info.param` 对象。这样做能保证 `info` 对象的标准化，方便你在 不同平台使用相同的机器人。

你可以通过 `info.raw` 拿到与微信官方文档一致的参数对象。

原始请求参数与 info 属性的对照表：

官方参数名	定义	info对象属性	备注

```
-------------------------------------------------------------------------------------------------------

ToUserName      开发者微信号                    info.sp                     sp means "service provider"
FromUserName    发送方帐号（一个OpenID）        info.uid
CreateTime      消息创建时间 （整型）
MsgId           消息id                          info.id
MsgType         消息类型                        info.type
-------------------------------------------------------------------------------------------------------
Content         文本消息内容                    info.text                   MsgType == text
-------------------------------------------------------------------------------------------------------
PicUrl          图片链接                        info.param.picUrl           MsgType == image
-------------------------------------------------------------------------------------------------------
Location_X      地理位置纬度(lat)               info.param.lat              MsgType == location
Location_Y      地理位置经度(lng)               info.param.lng
Scale           地图缩放大小                    info.param.scale
Label           地点名                          info.param.label            可能为空
-------------------------------------------------------------------------------------------------------
Title           消息标题                        info.param.title            MsgType == link
Description     消息描述                        info.param.description
Url             消息链接                        info.param.url
-------------------------------------------------------------------------------------------------------
Event           事件类型                        info.param.event            MsgType == event
                subscribe(订阅)、
                unsubscribe(取消订阅)、
                CLICK(自定义菜单点击事件)
                LOCATION(上报地理位置事件)

EventKey        事件KEY值，与自定义菜单接       info.param.eventKey
                口中KEY值对应
-------------------------------------------------------------------------------------------------------
MediaId         媒体文件的 id                   info.param.mediaId          MsgType == voice / video
Recognition     语音识别的文本                  info.param.recognition      MsgType == voice
ThumbMediaId    视频消息缩略图的媒体id          info.param.thumbMediaId     MsgType == video
Format          音频文件的格式                  info.param.format
```

注意：

- 大部分属性值只是把首字母大写换成了小写。地理信息的 `Location_X` 和 `Location_Y` 除外。
- recognition 参数需要开通微信的语音识别功能，同时为方便调用，此文本也会直接存到 info.text 也就是说，语音识别消息与普通文本消息都有 `info.text`，只不过 `info.type` 不同

例如，地理位置消息(MsgType === 'location') 会被转化为：

```
{
  uid: 'the_FromUserName',
  sp: 'the_ToUserName',
  id: 'the_MsgId',
  type: 'location',
  param: {
    lat: 'the_Location_X',
    lng: 'the_Location_Y',
    scale: 'the_Scale',
    label: 'the_Label'
  }
}
```

info.reply

大部分时候你并不需要直接给 **info.reply** 赋值。

你只需在 `rule.handler` 的返回值或 callbak 里提供回复消息的内容， `webot.watch` 自带的 express 中间件会自动给 `info.reply` 赋值， 并将其打包成 XML 发送给微信服务器。

`info.reply` 支持的数据类型：

- {String} 直接回复文本消息，不能超过2048字节
- {Object} 单条 图文消息/音乐消息
- {Array} 多条图文消息

回复文本消息

```
info.reply = '收到你的消息了，谢谢'
```

回复图文消息

```
title        消息标题
url          消息网址
description  消息描述
picUrl       消息图片网址
```

```
info.reply = {
  title: '消息标题',
  url: 'http://example.com/...',
  picUrl: 'http://example.com/....a.jpg',
  description: '对消息的描述出现在这里',
}

// or

info.reply = [{
  title: '消息1',
  url: 'http://example.com/...',
  picUrl: 'http://example.com/....a.jpg',
  description: '对消息的描述出现在这里',
}, {
  title: '消息2',
  url: 'http://example.com/...',
  picUrl: 'http://example.com/....a.jpg',
  description: '对消息的描述出现在这里',
}]
```

回复音乐消息

```
title             标题
description       描述
musicUrl          音乐链接
hqMusicUrl        高质量音乐链接，wifi 环境下会优先使用该链接播放音乐
```

需指定 `reply.type` 为 `'music'`：

```
info.reply = {
  type: 'music',
  title: 'Music 101',
  musicUrl: 'http://....x.mp3',
  hqMusicUrl: 'http://....x.m4a'
}
```

Have fun with wechat, and enjoy being a robot!

info.noReply

如果对不想回复的消息，可设置 `info.noReply = true`

```
// 比如对于语音类型的消息不回复
webot.set('ignore', {
  pattern: function(info) {
    return info.is('voice');
  },
  handler: function(info) {
    info.noReply = true;
```

```
      return;
    }
  });
```

命令行工具

提供可执行文件 `webot` 用于发送测试消息。 使用 `npm` 安装 webot-cli：

```
    npm install webot-cli -g
```

自定义菜单

webot-cli 提供处理微信自定义菜单的功能，安装好之后执行：

```
webot help menu
```

FAQ

Q: Info对象可否提供详细的文档，例如包含方法以及变量的说明。之前的说明并不够详细，最近改动API后就更混乱了。谢谢。

A: 在 README 里已经有详细文档。如果觉得不够详细，应该发`issue`，FAQ应该是可以通过文字回答直接解决的疑问才对。

Q: 我使用了info.wait('wait_someaction')这个机制，单个node实例的时候运作正常。为了效率我用pm2做了四个node的cluster，由于每次请求都是由

A：你需要配置 Express sessionStore 的持久化。示例：

```
 app.use(express.cookieParser());
 // 参考 http://expressjs.com/2x/guide.html#session-support
 app.use(express.session({ secret: 'abced111', store: new RedisStore() }));
```

各个 cluster 采用相同的持久化存储连接（也就是相同的数据库配置），和相同的 cookie secret

> webot-example

微信公共帐号机器人示例 

本地运行

```
git clone https://github.com/node-webot/webot-example.git
cd webot-example/
npm install
make start
```

其中，`make start` 命令会调用 `node app.js` 。 建议你 fork 一份自己的版本，这样你就可以任意做出更改和调试了。

消息调试

使用 `webot-cli` 命令行工具来发送测试消息。

安装：

```
npm install webot-cli -g
```

`npm install -g` 代表全局安装 npm 模块，你可能需要 `sudo` 权限。

使用：

```
webot help            # 查看使用帮助
webot send Hello      # 发送一条叫「Hello」的消息
webot send image      # 调试图片消息
webot send location   # 调试地理位置
webot send event      # 调试事件消息
```

`webot-cli` 默认访问的接口地址是 http://127.0.0.1:3000 ，要调试本示例的程序， 你需要指定 `webot send --des http://127.0.0.1:3000/wechat'

在微信上试用此示例

- 微信账号：webot-test

搭建你自己的机器人

1. fork 本仓库，修改 package.json 里的各项属性
2. 修改你自己的 app.js，填写你在微信后台输入的 token
3. 参考 rules/index.js，新建你自己的回复规则

发布到云平台

仓库中的 `Procfile` 为 heroku 的配置文件。 `manifest.yml` 为 cloudfoundry 的示例配置文件。

Credit

weixin-robot 的初始版本由@ktmud实现， @atian重构并扩展为 0.2 版本。目前的测试用例也大部分由他完成。

[weixin-robot] 使用了 @JacksonTian 的 wechat 组件。

weixin-robot 0.5

Breaking change!

- [*] 使用 `wechat-mp` 模块来处理与微信之间的关系，各 middlewares 拆分细化
- [+] 新增 `webot.middleware()` 接口来调用 `webot.reply()`

注意： 现在如果要启用session支持， `webot.watch` 必须在 `app.use(connect.session())` 之前

weixin-robot 0.4.6

- [+] 语音识别消息增加 info.param.recognition
- [*] 升级依赖

webot 0.4.1-1

- 0.4.1 之后的更新请移步 webot/History.md

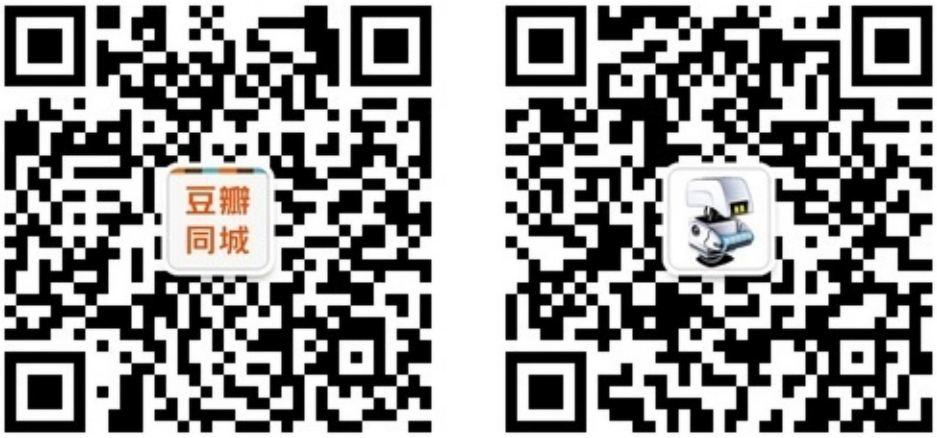
webot 0.4.1

- [+] 增加 `webot.waitRule`， 创建等待回复时专用的规则
- [+] 增加 `webot.use()` 方法，可以在回复每条消息前对请求消息预处理
- [+] 增加 `webot.config.beforeSend`， 回复消息的预处理，替代 mapping 的功能
- [-] 删掉打包消息时的 `mapping` 支持

weixin-robot 0.4

- [*] 规则匹配逻辑拆分为 `webot`， 微信相关逻辑和 session 支持采用 `wechat` 模块
- [*] `webot.watch` 不再支持 `webot.watch(app, path, token)` 这种易混淆的 API， 需要使用 `webot.watch(app, { path: path, token: token })`
- [*] 特殊消息的参数值不再直接附在 `info` 上，必须通过 `info.param` 获取 值得注意的时， `info.pic` 属性被替换为了 `info.param.picUrl`
- [*] `webot.checkSig`，`webot.bodyParser` 等 middleware 不再支持直接调用。 但可以通过 `webot.wechat` 访问到一些 `wechat` 组件的方法。
- [-] `webot.wait` 和 `webot.data` 方法已弃用，请总是使用 `info.wait`
- [*] `info.wait` 需要 session 支持，请使用 connect 的 cookieSession 中间件为请求附加 session
- [-] 去掉 `info.isLocation` 等判断消息类型的 API， 避免新增类型支持时需要穷举
- [+] 增加 `info.is('location')` 替代 `info.isLocation()`
- [-] 去掉 `info.query`， 使用 `info.param` 代替
- [-] 使用 `info.wait` 时不再支持闭包留存变量值，应总是使用 `info.session` 来存取变量
- [+] `info.wait` 只接受 rule name 字符串作为参数

使用此系统的微信帐号



小野助手（帐号：**onaka_ass**）

豆瓣同城（帐号：**douban-event**）

短网址小助手（微信号：**shorturl**）

秀域美容有限公司（帐号：**ishowyu**）

羊圈**[Yjion.com]**（账号：**yjion_**）

鱼山科协（中国海洋大学海洋生命学院科技协会，微信号：**oucoast**）

最美应用（微信号：**nice-app**）

#

>  node-webot/webot

node网路机器人 

一个node.js的网络机器人.

基于正则何自定义函数的简单规则，你可以很容易的运行一个网络服务。

许可证

入门

```
var express = require('express');
var webot = require('webot');

var app = express();

webot.set('hi', "Hi, I'm Webot.");

webot.set('subscribe', {
  pattern: function(info) {
    return info.event === 'subscribe';
  },
  handler: function(info) {
    return 'Thank you for subscribe.';
  }
});

app.get('/webot', function(req, res, next) {
  var message = req.query.message;

  webot.reply({
    text: message,
  }, function(err, info) {
    if (err) return res.json({ r: err });
    res.json({
      r: 0,
      reply: info.reply
    });
  });
});
```

Webot

webot.set(pattern, handler, *[, replies]*)

添加回复规则.

```
webot.set(pattern, handler, replies)
// or
webot.set({
  name: 'rule name',
  pattern: function(info) { ... },
  handler: function(info, next) {
  }
})
```

我们建议你给每条规则都命名，以方便规则之间互相调用。

```
webot.set('rule A', {
  pattern: /ruleA/,
  handler: function() {
  },
});

// webot.get('rule A') 即可获得刚才定义的规则

// 可以省略第二个参数里的 pattern ，
// 则规则名会被转换为一个用于匹配的正则
webot.set('你好', function() {
  // 随机回复一句话
  return ['你也好', '你好', '很高兴认识你'];
});

// 更简单地
webot.set('你好', ['你也好', '你好', '很高兴认识你']);

// 如果 handler 是一个 object ，也会直接作为 reply 返回
webot.set('test music message', {
  type: 'music',
  url: 'http://example.com/a.mp3'
});
```

你甚至还可以直接传入一个 Object ， 其 key 为 pattern ， value 为 handler（只要里面不包括 'handler' 这个 key）：

```
webot.set({
  '你好':  function() {
    // 随机回复一句话
    return ['你也好', '你好', '很高兴认识你'];
  },
  '你是谁': '我是你的小天使呀'
});
```

webot.delete(ruleName);

你可以根据rule的名字动态的删除一个rule,

```
webot.delete('rule1');
```

webot.update(name,{})

你可以根据rule的名字动态的更新rule，参数和set方法的参数一致

```
webot.update('rule1', {
  pattern: 'hello',
  handler: function(info) {
    info.reply = 'hello';
  }
});
```

有关 `replies` 的使用，请参考 rule.replies 。

webot.get(ruleName)

根据它的名字获取规则。 `ruleName` 必须是一个字符串。

webot.waitRule(name, [handler])

给 `info.wait` 设置等规。必须提供一个有效的 `name` .如果处理器未响应，尝试获得那个名字的等待规则.

`等待规则` 即只在等待用户回复时才执行的规则。等待规则，在返回结果后结束。如果不想结束请使用 `info.rewait()` ;

```
// 定义一个 wait rule
webot.waitRule('wait_guess', function(info) {
  var r = Number(info.text);
  // 用户不想玩了...
  if (isNaN(r)) {
    info.resolve();
    return null;
  }
  var num = info.session.guess_answer;
  if (r === num) {
    return '你真聪明!';
  }
  var rewaitCount = info.session.rewait_count || 0;
  if (rewaitCount >= 2) {
    return '怎么这样都猜不出来！答案是 ' + num + ' 啊！';
  }
  //重试
  info.rewait();
  return (r > num ? '大了': '小了') +',还有' + (2 - rewaitCount) + '次机会,再猜.';
});

webot.set('guess number', {
  description: '发送: game , 玩玩猜数字的游戏吧',
  pattern: /(?:game|玩?游戏)\s*(\d*)/,
  handler: function(info){
    //等待下一次回复
    var num = Number(info.param[1]) || _.random(1,9);
    verbose('answer is: ' + num);
    info.session.guess_answer = num;
    info.wait('wait_guess');
    return '玩玩猜数字的游戏吧, 1~9,选一个';
  }
});
```

webot.beforeReply(handler)

添加预处理规则。 每次在检测回复规则之前 `rule.handler` 将被调用。

预处理规则，可以对发来的消息进行预处理，比如从数据库取出用户等。

```
webot.beforeReply(function load_user(info, next) {
  User.get(info.uid, function(err, user) {
    if (err) return next(err);
```

```
    info.user = user; // attach this user object to Info.
    next();
  });
});
```

webot.afterReply()

添加后回复规则。每次获得回复后 `rule.handler` 将被调用。

获得回复内容或对消息再处理，比如简繁转化。

webot.domain(pattern, handler, replies)

添加指定的域规则，规则组的中间件。

实例:

```
webot.domain('domain-1', function require_user(info) {
  if (!info.user) {
    return 'Must login first';
  }
});

webot.set('domain-1 act-1', {
  domain: 'domain-1',
  pattern: /some pattern/,
  handler: function()..
});

webot.set('domain-1 act-2', {
  domain: 'domain-1',
  pattern: /another pattern/,
  handler: function()..
});
```

所以 `/some pattern/` 和 `/another pattern/` 被匹配时, 机器人将首先运行 `domain-1` 中间件,检测用户是否登陆。

webot.loads(file1, *[file2, ...]*)

载入 nodejs 模块作为 webot 的回复规则，以方便你在比较复杂的项目中组织文件。

file1, file2 是相对于调用此方法的文件所在目录的文件名。

```
webot.loads('./rules/a', './rules.b');
```

在文件 `./rules/a.js` 和 `./rules/b.js` 里是:

```
module.exports = function(webot) {
  webot.set('rule_a', function(info, next) {
    // ...
  });
  webot.set('rule_a_1', 'some reply');
};
```

或者使用简的规则定义

```
module.exports = {
  pattern: /some pattern/,
```

```
  handler: function(info, next) {
    // ...
  },
};
```

webot.dialog(file1, *[file2, ...]*)

增加对话规则

```
webot.dialog({
  'hello': '哈哈哈',
  'hi': ['好吧', '你好']
});

// 或者
webot.dialog('./rules/foo.js', './rules/bar.js');
```

在文件 `rules/foo.js` 里:

```
module.exports = {
  'hello': '哈哈哈',
  'hi': ['好吧', '你好']
};
```

使用YAML

你也可以在你的项目中 `require('js-yaml')`， 采用简洁的 yaml 语法来定义纯文本的对话规则：

在 `package.json` 里:

```
   "dependencies": {
       ...
     "js-yaml": "~2.0.3"
       ...
   }
```

在你得 `app.js`:

```
require('js-yaml');

webot.dialog('./rules/abc.yaml');
```

在 `rules/abc.yaml` 里:

直接回复

```
hi: 'hi,I am robot'
```

随机回复一个

```
hello:
  - 你好
  - fine
  - how are you
```

匹配组替换

```
/key (.*)/i: '你输入了: {1}, \{1}这样写就不会被替换'
```

也可以是一个**rule**定义；如果没有定义**pattern**，自动使用**key**

```
yaml:
  name: 'test_yaml_object'
  handler: '这是一个yaml的object配置'
```

webot.reset()

重置所有规则, 当应用在运行中你可以重载规则.

webot.watch(app, *[options]*)

添加标准中间件到express应用中. 包括:

- **options.verify**: 验证请. 默认: 始终通过.
- **options.parser**: 解析请求体. 默认: 使用 `req.body` .
- **options.send**: to send reply. 默认: use `res.json` .
- **options.sessionStore**: 存储机器人对话, 就像express的 cookieSession.
- **options.path**: 监测哪里. 默认: "/".
- **options.prop**: `req` 或者 `res` 的属性名附加分析和应答的数据. 默认: "webot_data".

中间件布局将是:

```
app.get(path, verify);
app.post(path, verify, parser, function(req, res, next) {
  webot.reply(req[prop], function(err, info) {
    res[prop] = info;
    next();
  });
}, send);
```

规则(options)

使用 `webot.set` 和 `webot.wait` 等方法时，会自动新建一条 rule， rule 定义的具体可用参数如下：

options.name

为规则命名，方便使用 `webot.get` 获取规则。

options.pattern

消息匹配规则，用以判断是否对用户发送的消息进行回复。如果为正则表达式和字符串， 则只匹配用户发送的文本消息（也就是 `info.text !== undefined` 的消息）。

所有支持的格式：

- {String} 如果是潜在的 RegExp （如字符串 '/abc/igm' ），会被转为 RegExp，如果以 '=' 打头，则完全匹配，否则模糊匹配
- {RegExp} 正则表达式，匹配到的捕获组会被赋予 `info.param` ， 可通过 `info.param[1]` , `info.param[2]` .. 获取
- {Function} 该函数只接受一个参数 `info` ，返回布尔值
- {NULL} 不指定 pattern ， 则视为通过匹配，此 Rule 的 handler 将总是会执行

示例：

```
// 匹配下列所有消息：
//
//    你是机器人吗
//    难道你是机器人？
//    你是不是机器人？
//    ...
//
webot.set('Blur match', {
  pattern: '是机器人',
  handler: '是的，我就是一名光荣的机器人'
});

// 当字符串 pattern 以 "=" 开头时，则会完全匹配
webot.set('Exact match', {
  pattern: '=a',
  handler: '只有回复「a」时才会看到本消息'
});

// 利用正则来匹配
webot.set('your name', {
  pattern: /^(?:my name is|i am|我(?:的名字)?(?:是|叫)?)\s*(.*)$/i,
  handler: function(info) {
    return '你好，' + info.param[1];
  },
  // 或者直接返回字符串，webot 会自动替换匹配关键字
  // handler: '你好，{1}'
});

// 类正则的字符串会被还原为正则
webot.set('/(good\s*)morning/i', '早上好，先生');

// 可以接受 function
webot.set('pattern as fn', {
  pattern: function(info){
    return info.param.eventKey === 'subscribe';
  },
  handler: '你好，欢迎关注我'
});
```

options.handler

指定如何生成回复消息

当返回非真值(null/false)时继续执行下一个动作，否则返回值会被回复给用户。

支持的定义格式:

- {String} 直接返回字符串
- {Array} 从数组中随机取一个作为 handler
- {Object} 直接返回
- {Function} 执行函数获取返回值，第一个参数为消息请求的 info 对象

支持异步：

```
webot.set('search_database', {
  description: 'Search a keyword from database',
  pattern: /^(?:s\s+)(.+)$/i,
  handler: function(info, next) {
    // assert(this.name == 'search_database');
    // 函数内的 this 变量即此规则

    // 执行一个异步操作..
    query_from_database(info.text, function(err, ret) {
      if (err) return next(500);
      return next(null, ret);
    });
  }
});
```

在函数执行过程中，如果设置 `info.ended = true` ，即使没有返回可用的回复，也不会再继续下一条规则。 此时会 fallback 到预设的 404 错误，参见 **webot.codeReplies**。

注意：

你是否注意到， `handler` 允许异步操作，而 `pattern` 却不可以？

事实上，所有的 pattern 操作，都可以放到 `handler` 里执行：

```
webot.set('test_A', {
  handler: function(info, next) {
    if (info.text == 'A') {
      next(null, 'You said A.');
    }
    next();
  }
});
```

或者更省略一点：

```
webot.set(function test_A(info) {
  if (info.text == 'A') {
    return 'You said A.';
  }
});
```

所以，异步的匹配可以写成：

```
webot.set('test', {
  handler: function(info, next) {
    var uid = info.uid;
    User.findOne(uid, function(err, doc) {
      if (!doc) {
```

```
        // 异步判断失败，进入下一条规则
        return next();
      }
      // 判断成功后需要执行的操作在这里
      return next(null, '欢迎你, ' + doc.name);
    });
  }
});
```

相信你并不会太需要在匹配消息规则时也进行异步。 事实上，很多需求都可以转化为使用 `webot.beforeReply` 。

options.replies

指定如何再次回复用户的回复。即用户回复了根据当前规则回复的消息后，如何继续对话。必须先配置 session支持。

```
webot.set('guess my sex', {
  pattern: /是男.还是女.|你.*男的女的/,
  handler: '你猜猜看呐',
  replies: {
    '/女|girl/i': '人家才不是女人呢',
    '/男|boy/i': '是的，我就是翩翩公子一枚',
    'both|不男不女': '你丫才不男不女呢',
    '不猜': '好的，再见',
    '/.*/': function(info) {
      // 在 replies 的 handler 里可以获得等待回复的重试次数参数
      if (info.rewaitCount < 2) {
        info.rewait();
        return '你到底还猜不猜嘛！';
      }
      return '看来你真的不想猜啊';
    },
  }
```

也可以用一个函数搞定:

```
  replies: function(info){
    return '嘻嘻，不告诉你'
  }
```

也可以是数组格式，每个元素为一条**rule**

```
  replies: [{
    pattern: '/^g(irl)?\\??$/i',
    handler: '猜错'
  },{
    pattern: '/^b(oy)?\\??$/i',
    handler: '猜对了'
  },{
    pattern: 'both',
    handler: '对你无语...'
  }]
});
```

Info

请求和响应在一个地方，带会话支持功能。

info.session

当你在你的 `express` 中间件中为 `info` 加入了 `session` 支持，即可使用等待操作的高级功能。

info.wait(rule)

等待用户回复。并根据 `rule` 定义来回复用户。 `rule` 可以是一个 function 或 object。用法与 `webot.set` 的参数类似。

info.rewait()

重试上次等待操作。一般在 `replies` 的 handler 里调用。

以上两个方法均需要 `session` 支持。具体用法请参看示例。

info.err

每次我们从 `rule.handler` 获取一个错误, `info.err` 将被更新.最后的错误将始终呆在哪.

对话支持

`session` 设置必须放置在 `webot.watch` 后面

```
var session = require('express-session')
  , redis = require("redis")
  , client = redis.createClient()
  , RedisStore = require('connect-redis')(session)
...
app.use(session({
    store: new RedisStore({client:client,host:'10.10.10.10',port:9999}),
    secret: 'webot secret',
     cookie: { secure: true }
}));
```

命令行工具

提供可执行文件 `webot` 用于发送测试消息。 使用 `npm` 安装 webot-cli：

```
npm install webot-cli -g
```

微信公共帐号机器人示例

> node-webot/webot-example

本地运行

```
git clone https://github.com/node-webot/webot-example.git
cd webot-example/
npm install
make start
```

其中，`make start` 命令会调用 `node app.js` 。 建议你 fork 一份自己的版本，这样你就可以任意做出更改和调试了。

目录结构

- lib/
  - support.js

    ```
    高德地图API
    搜索百度
    下载图片
    ```

- rules/
  - index.js #初始化路由规则
  - dialog.yaml #支持多个实例
- test/
  - bootstrap.js
  - mocha.opts
  - rule.js
- app.js #启动服务
- package.json

规则列表

1. help
2. more
3. who
4. 正则匹配后的匹配组存在 info.query 中
5. 简单的纯文本对话，可以用单独的 yaml 文件来定义
6. 支持一次性加多个（方便后台数据库存储规则）
7. 等待下一次回复
8. 定义一个 wait rule
9. 调用已有的action
10. 可以通过回调返回结果
11. 超时处理
12. 实现类似电话客服的自动应答流程
13. 支持location消息 此examples使用的是高德地图的API
14. 回复图文消息
15. 可以指定图文消息的映射关系
16. 所有消息都无法匹配时的fallback

消息调试

使用 `webot-cli` 命令行工具来发送测试消息。

安装：

```
npm install webot-cli -g
```

`npm install -g` 代表全局安装 npm 模块，你可能需要 `sudo` 权限。

使用：

```
webot help            # 查看使用帮助
webot send Hello      # 发送一条叫「Hello」的消息
webot send image      # 调试图片消息
webot send location   # 调试地理位置
webot send event      # 调试事件消息
```

`webot-cli` 默认访问的接口地址是 http://127.0.0.1:3000 ，要调试本示例的程序， 你需要指定 `webot send --des http://127.0.0.1:3000/wechat'

在微信上试用此示例

- 微信账号：webot-test

搭建你自己的机器人

1. fork 本仓库，修改 package.json 里的各项属性
2. 修改你自己的 app.js ，填写你在微信后台输入的 token
3. 参考 rules/index.js ，新建你自己的回复规则

发布到云平台

仓库中的 `Procfile` 为 heroku 的配置文件。 `manifest.yml` 为 cloudfoundry 的示例配置文件。

荣誉

weixin-robot 的初始版本由@ktmud实现， @atian重构并扩展为 0.2 版本。目前的测试用例也大部分由他完成。

weixin-robot 使用了 @JacksonTian 的 wechat 组件。

> node-webot/webot-cli

webot-cli

weixin-robot 的命令行接口.

```
npm install webot-cli -g
```

许可证

(the MIT licence)

命令

```
Usage: webot <command> [options]

Options:

  -h, --help     output usage information
  -V, --version  output the version number

Commands:

  send [image|text|loc|..]   send a message to test host
  menu [create|delete|get]   manipulate wechat menu
  help [command]             view help info for specified command
```

使用 `webot help` 获取更多信息.

发送测试消息

```
Usage: webot send [type] [options] [message]

Types:

  # t, text            Send text messages (default)
  # i, image, pic      Send image messages
  # l, loc, location   Send location messages
  # e, event           Send event messages
  # scan               Send scan QRcode event
  # reportloc          Send report location event

Options:

  --token          API token for wechat, defaults to `process.env.WX_TOKEN`
  --port           The port your service is listening at, defaults to `process.env.PORT`
  --host           Server hostname, defaults to 127.0.0.1
  --route          The subdirectory you are watching
  --des            Request destination, a full url
                   Will override host, port and route
  --user           FromUserName of this message.

Examples:

  webot send --token abc123 --des http://example.com/webot
  webot send t Hello
  webot send loc 20.12 120.33 "Somewhere Out There"
```

在代码里使用

```
var send = require('webot-cli').commands.send
var options = {
  silent: true,
  port: PORT,
  route: media.webotPath(),
  token: media.wx_token,
  input: ['text', 'abcotea']
}

send(options, data)
```

微信自定义菜单

```
webot help menu
```

需要一个 json 文件保存 access_token 等配置信息（默认是当前目录的 **wx_config.json**）。 为安全起见，建议你只在本地使用此文件，不要把它放到代码仓库中。

```
{
  "appid": "abcedfe123456",
  "secret": "abcdefghijklmn1234567890"
}
```

创建或修改

按照微信文档中的[请求示例][1]，新建一个 json 文件（如 `menu.json`），然后利用 stdin 传入 `webot menu create` 命令：

文件 **menu.json** 的内容：

```
 {
     "button":[
     {
          "type":"click",
          "name":"今日歌曲",
          "key":"V1001_TODAY_MUSIC"
      },
      {
           "type":"click",
           "name":"歌手简介",
           "key":"V1001_TODAY_SINGER"
      },
      {
           "name":"菜单",
           "sub_button":[
            {
               "type":"click",
               "name":"hello word",
               "key":"V1001_HELLO_WORLD"
            },
            {
               "type":"click",
               "name":"赞一下我们",
               "key":"V1001_GOOD"
            }]
       }]
 }
```

执行命令：

```
webot menu create --config wx_config.json < menu.json
```

自定义菜单与 **webot** 的结合

发送文字消息或者触发点击事件都可以：

```
var reg_gequ = /(今日歌曲|today music)/i;

webot.set('today_music', {
  pattern: function(info) {
```

```
    return reg_gequ.test(info.text) || info.param.eventKey === 'V1001_TODAY_MUSIC';
  },
  handler: function(info, cb) {
    // 从你的数据库里查找今日歌曲
    cb(null, {
      type: 'music',
      musicUrl: 'http://xxxx...',
      hqMusicUrl: 'http://xxxx...'
    });
  },
});
```

> node-webot/wechat-mp

wechat-mp 微信公众平台消息接口中间件 

校验签名，接受并解析微信消息，处理回复内容为 XML，并回复给微信。

如需使用自定义菜单等高级接口，可使用 wechat-api 模块。

许可证

如何使用Express中间件

本模块主要作为 Connect/Express 框架的中间件使用：

```
var mp = require('wechat-mp')(process.env.WX_TOKEN)
var app = require('express')()

app.use('/wechat', mp.start())
app.post('/wechat', function(req, res, next) {
  console.log(req.body)
  res.body = {
    msgType: 'text',
    content: 'Hi.'
  }
  // or rich media message
  res.body = {
    msgType: 'music',
    content: {
      title: 'A beautiful song',
      musicUrl: 'http://.....'
    },
  }
  next()
}, mp.end())
```

如果要在 koa 里使用，可尝试 koa-wechat 模块。

选项 require('wechat-mp')(*[options]*)

`options` can be either the token string or an object. You can use these options both when initialization(`mp = require('wechat-mp')(options)`) and `mp.start()` .

- token
- tokenProp
- dataProp
- map
- session

options.token

微信令牌用来检测签名。

options.tokenProp

默认: 'wx_token'

将试着获取 `req[tokenProp]` 作为令牌. 适合动态设置令牌。

options.dataProp

默认: 'body'

Will put parsed data on `req[dataProp]` . So you can access wechat request message via `req.body` or `req.wx_data` , etc.

已析数据属性映射

We changed some of the properties' names of Wechat's incoming message, to make it more "JavaScript like", typically, a request datum would be:

```
{
  uid: 'xahfai2oHaf2ka2M41',      // FromUserName
  sp: 'gh_xmfh2b32tmgkgagsagf',   // ToUserName
  type: '',                       // MsgType
  createTime: new Date(2014-12..) // CreateTime
  text: 'Hi.',                    // when it's a text message
  param: {
    lat: '34.193819105',          // for a "LOCATION" message's Location_X
    lng: '120.2393849201',        // Location_Y
  }
}
```

欲了解更多详情，请参阅 `lib/xml.js` .

options.session

Unless `options.session` is set to `false` , the `mp.start()` middleware will set `req.sessionID` and `req.sessionId` to `"wx.#{toUserName}.#{fromUserName}"` . So you can use `req.session` to save information about one specific user.

The `sessionId` cannot be changed by any other following middlewares.

注意 为了使它工作, `mp.start()` 必须放在 express/connect的session中间件前.

```
app.use('/wechat', mp.start())
app.use(connect.cookieParser())
app.use(connect.session({ store: ... }))
```

mp.start()

The starting middleware, to parse a Wechat message request, and set `req.body` as a JS object.

mp.end()

The ending middleware, to response a xml based on `res.body`. For how to set `res.body` for multi media messages, see source code `lib/xml.js`.

If your set `res.body` to a string, will reply a text message. When `res.body` is an object, a `res.body.msgType` is expected, otherwise it will be treated as text message, and `res.body.content` will be replied as it was a string.

典型的响应:

```
{
  msgType: 'news',
  content: [{
    title: 'news 1',
    url: 'http://...',
    picUrl: 'http://...'
  }, {
    title: 'news 2',
    url: 'http://...',
    picUrl: 'http://...'
  }]
}
```

weixin-robot

使用 wexin-robot 模块，更傻瓜化地定义自动回复功能。

调试

使用 webot-cli 调试发送测试消息。

> ktmud/wechat-api

wechat-api 微信公共平台 API 工具

本模块只负责与 `access_token` 有关的高级接口功能， 接收和发送消息请使用 wechat-mp 。

特点

- 统一的 API 调用
- 自动管理 `access_token` 过期时间，请求时如遇过期，将自动刷新。

待办事项

- 接口频率限制

许可证

使用

```
// replace `APP_ID` and `SECRET` with corresponding value
var wechat = require('wechat-api')(APP_ID, SECRET)

// 查询自定义菜单信息
wechat.getMenu(function(err, result) {
  //
  // the `result` will be the json response
  //
  // when error happens:
  //
  // err == {"errcode":40013,"errmsg":"invalid appid"}
  // result == null
  //
})

// 根据 OpenID 获取用户信息
wechat.getUserInfo({
  openid: 'o6_bmjrPTlm6_2sgVt7hMZOPfL2M',
  lang: 'zh_CN'
}, function(err, result) {
  //
  // result === {
  //   "subscribe": 1,
  //   "openid": "o6_bmjrPTlm6_2sgVt7hMZOPfL2M",
  //   "nickname": "Band",
  //   "sex": 1,
  //   "language": "zh_CN",
  //   "city": "广州",
  //    ...
  // }
  //
})
```

API

- createMenu(menu, callback) 创建菜单
- getMenu(callback) 查询自定义菜单
- deleteMenu(callback) 删除自定义菜单
- getUserInfo(args, callback) 获取用户信息 可直接传入 openid 字符串
- getUserList(args, callback) 获取关注者列表
- createQRCode(args, callback) 创建二维码
- createTempQRCode(scene_id, [expires_seconds,] callback) 创建临时二维码
- createPermQRCode(scene_id, callback) 创建永久二维码
- uploadMedia(type, filepath, callback) 上传多媒体文件
- mediaUrl(media_id) 获取文件下载地址
- getMedia(media_id) 下载多媒体文件，返回一个 readable stream ， 可直接 pipe 到本地文件存储

事件

- **ready** 已获取 access_token，可以开始请求，如果你在初始化时传入了之前存储好的 token，`ready` 事件会即时触发
- **refresh** 已获取新的 access_token

```
var client = require('wechat-api')('appid', 'secret', 'the-stored-token')

client.on('ready', function() {
})
client.on('refresh', function(token) {
  // token.expire_date ~= +new Date() + 7200 * 10e3
})
```

> node-webot/wechat

wechat

微信公共平台消息接口服务中间件与API SDK

交流群

QQ群：157964097，使用疑问，开发，贡献代码请加群。

感谢

感谢以下贡献者：

```
project  : wechat
repo age : 1 year, 8 months
active   : 100 days
commits  : 232
files    : 60
authors  :
  195  Jackson Tian            84.1%
   10  yelo                    4.3%
    4  realdog                 1.7%
    4  Bruce Lee               1.7%
    3  Guo Yu                  1.3%
    2  zhongao                 0.9%
    2  Jesse Yang              0.9%
    2  Lu Jun                  0.9%
    2  dan                     0.9%
    1  Chen Wei                0.4%
    1  feit                    0.4%
    1  ifeiteng                0.4%
    1  p13766                  0.4%
    1  Rogerz Zhang            0.4%
    1  xianda                  0.4%
    1  Lance Li                0.4%
    1  feichang.wyl            0.4%
```

捐赠

如果您觉得Wechat对您有帮助，欢迎请作者一杯咖啡

简介

功能列表

- 自动回复（文本、图片、语音、视频、音乐、图文）
- 发送客服消息（文本、图片、语音、视频、音乐、图文）
- 菜单操作（查询、创建、删除）
- 二维码（创建临时、永久二维码，查看二维码URL）
- 分组操作（查询、创建、修改、移动用户到分组）
- 用户信息（查询用户基本信息、获取关注者列表）
- 媒体文件（上传、获取）
- 等待回复（用于调查问卷、问答等场景）
- 会话支持（创新功能）
- OAuth API（授权、获取基本信息）
- 群发消息（文本、图片、语音、视频、图文）
- 客服记录
- 群发消息
- 公众号支付（发货通知、订单查询）
- 微信小店（商品管理、库存管理、邮费模板管理、分组管理、货架管理、订单管理、功能接口）
- 模版消息

详细参见API文档

安装

```
$ npm install wechat
```

与Connect/Express一起使用

```
var wechat = require('wechat');

app.use(connect.query()); // Or app.use(express.query());
app.use('/wechat', wechat('some token', function (req, res, next) {
  // 微信输入信息都在req.weixin上
  var message = req.weixin;
  if (message.FromUserName === 'diaosi') {
    // 回复屌丝(普通回复)
    res.reply('hehe');
  } else if (message.FromUserName === 'text') {
    //你也可以这样回复text类型的信息
    res.reply({
      content: 'text object',
      type: 'text'
    });
  } else if (message.FromUserName === 'hehe') {
    // 回复一段音乐
    res.reply({
      type: "music",
      content: {
        title: "来段音乐吧",
        description: "一无所有",
        musicUrl: "http://mp3.com/xx.mp3",
        hqMusicUrl: "http://mp3.com/xx.mp3"
      }
    });
  } else {
    // 回复高富帅(图文回复)
    res.reply([
      {
        title: '你来我家接我吧',
        description: '这是女神与高富帅之间的对话',
        picurl: 'http://nodeapi.cloudfoundry.com/qrcode.jpg',
        url: 'http://nodeapi.cloudfoundry.com/'
      }
    ]);
  }
}));
```

备注：token在微信平台的开发者中心申请

回复消息

当用户发送消息到微信公众账号，自动回复一条消息。这条消息可以是文本、图片、语音、视频、音乐、图文。详见：官方文档

回复文本

```
res.reply('Hello world!');
// 或者
res.reply({type: "text", content: 'Hello world!'});
```

回复图文

```
res.reply([
  {
    title: '你来我家接我吧',
    description: '这是女神与高富帅之间的对话',
    picurl: 'http://nodeapi.cloudfoundry.com/qrcode.jpg',
    url: 'http://nodeapi.cloudfoundry.com/'
  }
]);
```

回复图片

```
res.reply({
  type: "image",
  content: {
    mediaId: 'mediaId'
  }
});
```

回复语音

```
res.reply({
  type: "voice",
  content: {
    mediaId: 'mediaId'
  }
});
```

回复视频

```
res.reply({
  type: "video",
  content: {
    mediaId: 'mediaId',
    thumbMediaId: 'thumbMediaId'
  }
});
```

回复音乐

```
res.reply({
  title: "来段音乐吧",
  description: "一无所有",
  musicUrl: "http://mp3.com/xx.mp3",
  hqMusicUrl: "http://mp3.com/xx.mp3"
});
```

对话支持[WXSession]

由于公共平台应用的客户端实际上是微信，所以采用传统的Cookie来实现会话并不现实，为此中间件模块在openid的基础上添加了Session支持。一旦服务端启用了 `connect.session` 中间件，在业务中就可以访问 `req.wxsession` 属性。这个属性与 `req.session` 行为类似。

```
app.use(connect.cookieParser());
app.use(connect.session({secret: 'keyboard cat', cookie: {maxAge: 60000}}));
app.use('/wechat', wechat('some token', wechat.text(function (info, req, res, next) {
  if (info.Content === '=') {
    var exp = req.wxsession.text.join('');
    req.wxsession.text = '';
    res.reply(exp);
  } else {
    req.wxsession.text = req.wxsession.text || [];
    req.wxsession.text.push(info.Content);
    res.reply('收到' + info.Content);
  }
})));
```

`req.wxsession` 与 `req.session` 采用相同的存储引擎，这意味着如果采用redis作为存储，这样 `wxsession` 可以实现跨进程共享。

等待回复

等待回复，类似于电话拨号业务。该功能在WXSession的基础上提供。需要为等待回复预置操作，中间件将其抽象为 `List` 对象，在提供服务前需要添加服务。

添加服务[List]

```
var List = require('wechat').List;
List.add('view', [
  ['回复{a}查看我的性别', function (info, req, res) {
    res.reply('我是个妹纸哟');
  }],
  ['回复{b}查看我的年龄', function (info, req, res) {
    res.reply('我今年18岁');
  }],
  ['回复{c}查看我的性取向', '这样的事情怎么好意思告诉你啦- -']
]);
```

触发事务[res.wait()]

然后在业务中触发等待回复事务，如下示例，当收到用户发送 `list` 后，调用 `res.wait('view')` 进入事务 `view` 中。

```
var app = connect();
app.use(connect.query());
app.use(connect.cookieParser());
app.use(connect.session({secret: 'keyboard cat', cookie: {maxAge: 60000}}));
app.use('/wechat', wechat('some token', wechat.text(function (info, req, res, next) {
  if (info.Content === 'list') {
    res.wait('view');
  } else {
    res.reply('hehe');
    // 或者中断等待回复事务
    // res.nowait('hehe');
  }
})));
```

用户将收到如下回复：

```
回复a查看我的性别
回复b查看我的年龄
回复c查看我的性取向
```

用户回复其中的 `a` 、 `b` 、 `c` 将会由注册的方法接管回复。回复可以是一个函数，也可以是一个字符串：

```
List.add('view', [
  ['回复{a}查看我的性别', function (info, req, res, next) {
    res.reply('我是个妹纸哟');
  }],
  // 或者字符串
  ['回复{c}查看我的性取向', '这样的事情怎么好意思告诉你啦- -']
]);
```

中断事务[res.nowait()]

如果用户触发等待回复事务后，没有按照 `{}` 中的进行回复，那么将会由原有的默认函数进行处理。在原有函数中，可以选择调用 `res.nowait()` 中断事务。 `nowait()` 除了能中断事务外，与 `reply` 的行为一致。

案例展示

Node.js API自动回复

欢迎关注。

代码：https://github.com/JacksonTian/api-doc-service

你可以在CloudFoundry、appfog、BAE等搭建自己的机器人。

API

原始API文档请参见：消息接口指南。

目前微信公共平台能接收到7种内容：文字、图片、音频、视频、位置、链接、事件。支持6种回复：纯文本、图文、音乐、音频、图片、视频。 针对目前的业务形态，发布了0.6.x版本，该版本支持六种内容分别处理，以保持业务逻辑的简洁性。

```
app.use('/wechat', wechat('some token', wechat.text(function (message, req, res, next) {
  // message为文本内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125035',
  // MsgType: 'text',
  // Content: 'http',
  // MsgId: '5837397576500011341' }
}).image(function (message, req, res, next) {
  // message为图片内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359124971',
  // MsgType: 'image',
  // PicUrl: 'http://mmsns.qpic.cn/mmsns/bfc815ygvIWcaaZlEXJV7NzhmA3Y2fc4eBOxLjpPI60Q1Q6ibYicwg/0',
  // MediaId: 'media_id',
  // MsgId: '5837397301622104395' }
}).voice(function (message, req, res, next) {
  // message为音频内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125022',
  // MsgType: 'voice',
  // MediaId: 'OMYnpghh8fRfzHL8obuboDN9rmLig4s0xdpoNT6a5BoFZWufbE6srbCKc_bxduzS',
  // Format: 'amr',
  // MsgId: '5837397520665436492' }
}).video(function (message, req, res, next) {
  // message为视频内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125022',
  // MsgType: 'video',
  // MediaId: 'OMYnpghh8fRfzHL8obuboDN9rmLig4s0xdpoNT6a5BoFZWufbE6srbCKc_bxduzS',
  // ThumbMediaId: 'media_id',
  // MsgId: '5837397520665436492' }
}).location(function (message, req, res, next) {
  // message为位置内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125311',
  // MsgType: 'location',
  // Location_X: '30.283950',
  // Location_Y: '120.063139',
  // Scale: '15',
  // Label: {},
  // MsgId: '5837398761910985062' }
}).link(function (message, req, res, next) {
  // message为链接内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125022',
  // MsgType: 'link',
  // Title: '公众平台官网链接',
  // Description: '公众平台官网链接',
  // Url: 'http://1024.com/',
  // MsgId: '5837397520665436492' }
}).event(function (message, req, res, next) {
  // message为事件内容
  // { ToUserName: 'gh_d3e07d51b513',
  // FromUserName: 'oPKu7jgOibOA-De4u8J2RuNKpZRw',
  // CreateTime: '1359125022',
  // MsgType: 'event',
  // Event: 'LOCATION',
  // Latitude: '23.137466',
  // Longitude: '113.352425',
  // Precision: '119.385040',
  // MsgId: '5837397520665436492' }
})));
```

注意： `text`，`image`，`voice`，`video`，`location`，`link`，`event` 方法请至少指定一个。 这六个方法的设计适用于按内容类型区分处理的场景。如果需要更复杂的场景，请使用第一个例子中的API。

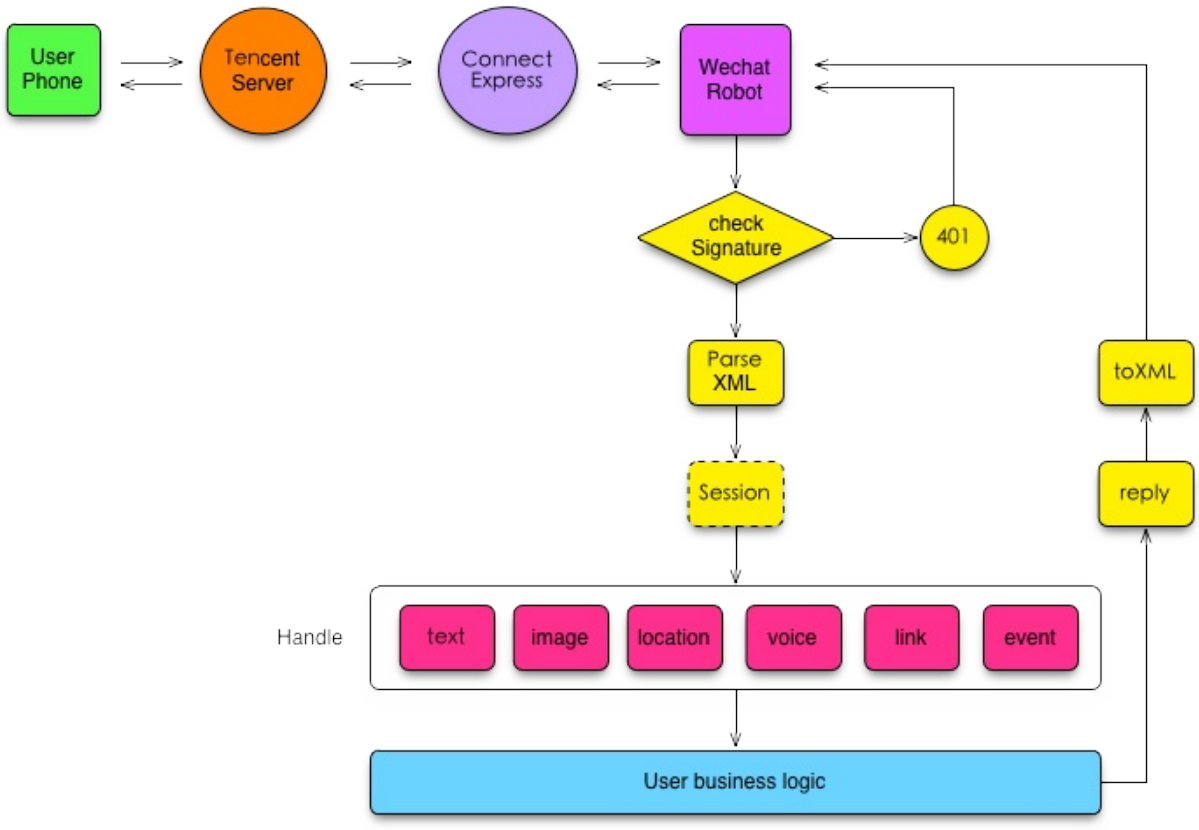
更简化的API设计

示例如下：

```
app.use('/wechat', wechat('some token').text(function (message, req, res, next) {
  // TODO
}).image(function (message, req, res, next) {
  // TODO
}).voice(function (message, req, res, next) {
  // TODO
}).video(function (message, req, res, next) {
  // TODO
}).location(function (message, req, res, next) {
  // TODO
}).link(function (message, req, res, next) {
  // TODO
}).event(function (message, req, res, next) {
  // TODO
}).middlewarify());
```

该接口从0.3.x提供。

流程图



诸多细节由wechat中间件提供，用户只要关注蓝色部分的业务逻辑即可。

> wechat-oauth

OAuth接口[wechat-oauth]

微信公共平台OAuth接口消息接口服务中间件与API SDK

模块状态

- npm package 1.0.0
- build passing
- dependencies up to date
- coverage 91%

功能列表

- OAuth授权
- 获取基本信息

OAuth2.0网页授权，使用此接口须通过微信认证，如果用户在微信中（Web微信除外）访问公众号的第三方网页，公众号开发者可以通过此接口获取当前用户基本信息（包括昵称、性别、城市、国家）。详见：官方文档

详细参见API文档

Installation

```
$ npm install wechat-oauth
```

Usage

初始化

引入OAuth并实例化

```
var OAuth = require('wechat-oauth');
var client = new OAuth('your appid', 'your secret');
```

以上即可满足单进程使用。 当多进程时，token需要全局维护，以下为保存token的接口。

```
var oauthApi = new OAuth('appid', 'secret', function (openid, callback) {
  // 传入一个根据openid获取对应的全局token的方法
  fs.readFile(openid +':access_token.txt', 'utf8', function (err, txt) {
    if (err) {return callback(err);}
    callback(null, JSON.parse(txt));
  });
}, function (openid, token, callback) {
  // 请将token存储到全局，跨进程、跨机器级别的全局，比如写到数据库、redis等
  // 这样才能在cluster模式及多机情况下使用，以下为写入到文件的示例
  // 持久化时请注意，每个openid都对应一个唯一的token!
  fs.writeFile(openid + ':access_token.txt', JSON.stringify(token), callback);
});
```

引导用户

生成引导用户点击的URL

```
var url = client.getAuthorizeURL('redirectUrl', 'state', 'scope');
```

用户点击上步生成的URL后会被重定向到上步设置的 `redirectUrl`，并且会带有 `code` 参数，我们可以使用这个 `code` 换取 `access_token` 和用户的 `openid`

```
client.getAccessToken('code', function (err, result) {
  var accessToken = result.data.access_token;
  var openid = result.data.openid;
});
```

如果我们生成引导用户点击的URL中 `scope` 参数值为 `snsapi_userinfo`，接下来我们就可以使用 `openid` 换取用户详细信息（必须在getAccessToken方法执行完成之后）

```
client.getUser('openid', function (err, result) {
  var userInfo = result;
});
```

捐赠

如果您觉得Wechat OAuth对您有帮助，欢迎请作者一杯咖啡

或者 tips $0/week

交流群

QQ群：157964097，使用疑问，开发，贡献代码请加群。

Contributors

感谢以下贡献者：

```
$ git summary
```

```
project  : wechat-oauth
repo age : 31 minutes
active   : 1 days
commits  : 2
files    : 10
authors  :
    2  Jackson Tian  100.0%
```

License

The MIT license.

企业版[wechat-enterprise]

> node-webot/wechat-enterprise

微信公共平台企业号版SDK

功能列表

- 自动回复
- 主动消息
- 菜单操作
- 部门管理
- 用户管理
- 标签管理
- 媒体文件
- OAuth API（授权、获取基本信息）

创新功能

- 会话支持

详细参见API文档 代码测试覆盖率

安装

```
$ npm install wechat-enterprise
```

实例

Node.js API自动回复

欢迎关注。

代码：https://github.com/JacksonTian/api-doc-service

你可以在CloudFoundry、appfog、BAE等搭建自己的机器人。

证书

The MIT license.

交流群

QQ群：157964097，使用疑问，开发，贡献代码请加群。

感谢

感谢以下贡献者：

```
project  : wechat-enterprise
repo age : 6 minutes
active   : 1 days
commits  : 1
files    : 46
authors  :
    1  Jackson Tian            100.0%
```

捐赠

如果您觉得Wechat企业号版本对您有帮助，欢迎请作者一杯咖啡


请我喝咖啡

wechat enterprise api

微信公共平台企业号版主动调用API

模块状态

- 
- 
- 
- 

功能列表

- 主动消息
- 菜单操作
- 部门管理
- 用户管理
- 标签管理
- 媒体文件
- OAuth API（授权、获取基本信息）

详细文档

- 文档主页
- API文档
- 代码测试覆盖率
- 新手上路

Show cases

Node.js API自动回复

欢迎关注。

代码：https://github.com/JacksonTian/api-doc-service

你可以在CloudFoundry、appfog、BAE等搭建自己的机器人。

License

The MIT license.

交流群

QQ群：157964097，使用疑问，开发，贡献代码请加群。

感谢

感谢以下贡献者：

```
project  : wechat-enterprise-api
repo age : 12 hours
active   : 1 days
commits  : 2
files    : 28
authors  :
    2  Jackson Tian  100.0%
```

捐赠

如果您觉得Wechat企业号版本对您有帮助，欢迎请作者一杯咖啡


请我喝咖啡

微信接口文档

> 微信公众平台开发者文档

接口权限表

服务包	接口内容	接口状态	操作
0:2	1:2	2:2	3:2

基础接口 接收用户消息 已获得 向用户回复消息 接受事件推送 自定义菜单 会话界面自定义菜单 已获得 高级接口 语音识别 未获得 客服接口 高级群发接口 设置用户备注名接口 OAuth2.0网页授权 生成带参数二维码 获取用户地理位置(已关闭) 获取用户基本信息 获取关注者列表 用户分组接口 上传下载多媒体文件 微信小店接口 微信小店接口 未获得 模板消息 模板消息 未获得

基础接口

接收用户消息

向用户回复消息

接受事件推送

自定义菜单

条件 服务号或者订认证号

会话界面自定义菜单

自定义菜单能够帮助公众号丰富界面，让用户更好更快地理解公众号的功能。开启自定义菜单后，公众号界面如图所示：

 目前自定义菜单最多包括3个一级菜单，每个一级菜单最多包含5个二级菜单。一级菜单最多4个汉字，二级菜单最多7个汉字，多出来的部分将会以“...”代替。请注意，创建自定义菜单后，由于微信客户端缓存，需要**24**小时微信客户端才会展现出来。建议测试时可以尝试取消关注公众账号后再次关注，则可以看到创建后的效果。

自定义菜单接口可实现多种类型按钮，如下：

1. click：点击推事件

   ```
   用户点击click类型按钮后，微信服务器会通过消息接口推送消息类型为event
   的结构给开发者（参考消息接口指南），并且带上按钮中开发者填写的key值，
   开发者可以通过自定义的key值与用户进行交互；
   ```

2. view：跳转URL

   ```
   用户点击view类型按钮后，微信客户端将会打开开发者在按钮中填写的网页URL，
   可与网页授权获取用户基本信息接口结合，获得用户基本信息。
   ```

3. scancode_push：扫码推事件

   ```
   用户点击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后显示扫描结果
   （如果是URL，将进入URL），且会将扫码的结果传给开发者，开发者可以下发消息。
   ```

4. scancode_waitmsg：扫码推事件且弹出“消息接收中”提示框

   ```
   用户点击按钮后，微信客户端将调起扫一扫工具，完成扫码操作后，
   将扫码的结果传给开发者，同时收起扫一扫工具，然后弹出“消息接收中”提示框，
   随后可能会收到开发者下发的消息。
   ```

5. pic_sysphoto：弹出系统拍照发图

   ```
   用户点击按钮后，微信客户端将调起系统相机，完成拍照操作后，
   会将拍摄的相片发送给开发者，并推送事件给开发者，同时收起系统相机，
   随后可能会收到开发者下发的消息。
   ```

6. pic_photo_or_album：弹出拍照或者相册发图

   ```
   用户点击按钮后，微信客户端将弹出选择器供用户选择“拍照”或者“从手机相册选择”。
   用户选择后即走其他两种流程。
   ```

7. pic_weixin：弹出微信相册发图器

   ```
   用户点击按钮后，微信客户端将调起微信相册，完成选择操作后，
   将选择的相片发送给开发者的服务器，并推送事件给开发者，同时收起相册，
   随后可能会收到开发者下发的消息。
   ```

8. location_select：弹出地理位置选择器

   ```
   用户点击按钮后，微信客户端将调起地理位置选择工具，完成选择操作后，
   ```

```
将选择的地理位置发送给开发者的服务器，同时收起位置选择工具，
随后可能会收到开发者下发的消息。
```

请注意，3到8的所有事件，仅支持微信iPhone5.4.1以上版本，和Android5.4以上版本的微信用户，旧版本微信用户点击后将没有回应，开发者也不能正常接收到事件推送。

接口调用请求说明

http请求方式：POST（请使用https协议） https://api.weixin.qq.com/cgi-bin/menu/create?access_token=ACCESS_TOKEN

click和view的请求示例

```
{
    "button":[
    {
         "type":"click",
         "name":"今日歌曲",
         "key":"V1001_TODAY_MUSIC"
     },
     {
          "name":"菜单",
          "sub_button":[
          {
              "type":"view",
              "name":"搜索",
              "url":"http://www.soso.com/"
           },
           {
              "type":"view",
              "name":"视频",
              "url":"http://v.qq.com/"
           },
           {
              "type":"click",
              "name":"赞一下我们",
              "key":"V1001_GOOD"
           }]
      }]
}
```

其他新增按钮类型的请求示例

```
{
    "button": [
        {
            "name": "扫码",
            "sub_button": [
                {
                    "type": "scancode_waitmsg",
                    "name": "扫码带提示",
                    "key": "rselfmenu_0_0",
                    "sub_button": [ ]
                },
                {
                    "type": "scancode_push",
                    "name": "扫码推事件",
                    "key": "rselfmenu_0_1",
                    "sub_button": [ ]
                }
            ]
        },
        {
            "name": "发图",
            "sub_button": [
                {
                    "type": "pic_sysphoto",
                    "name": "系统拍照发图",
                    "key": "rselfmenu_1_0",
                   "sub_button": [ ]
                 },
                {
```

```
                    "type": "pic_photo_or_album",
                    "name": "拍照或者相册发图",
                    "key": "rselfmenu_1_1",
                    "sub_button": [ ]
                },
                {
                    "type": "pic_weixin",
                    "name": "微信相册发图",
                    "key": "rselfmenu_1_2",
                    "sub_button": [ ]
                }
            ]
        },
        {
            "name": "发送位置",
            "type": "location_select",
            "key": "rselfmenu_2_0"
        }
    ]
}
```

参数说明

参数	是否必须	说明
button	是	一级菜单数组，个数应为1~3个
sub_button	否	二级菜单数组，个数应为1~5个
type	是	菜单的响应动作类型
name	是	菜单标题，不超过16个字节，子菜单不超过40个字节
key	click等点击类型必须	菜单KEY值，用于消息接口推送，不超过128字节
url	view类型必须	网页链接，用户点击菜单可打开链接，不超过256字节

返回结果

正确时的返回JSON数据包如下：

```
{"errcode":0,"errmsg":"ok"}
```

错误时的返回JSON数据包如下（示例为无效菜单名长度）：

```
{"errcode":40018,"errmsg":"invalid button name size"}
```

推广支持

生成带参数的二维码

长链接转短链接接口

生成带参数的二维码

长链接转短链接接口

将一条长链接转成短链接。

主要使用场景： 开发者用于生成二维码的原链接（商品、支付二维码等）太长导致扫码速度和成功率下降，将原长链接通过此接口转成短链接再生成二维码将大大提升扫码速度和成功率。

接口调用请求说明

开发者可通过OpenID来获取用户基本信息。请使用https协议。

接口调用请求说明

http请求方式: `POST`

```
https://api.weixin.qq.com/cgi-bin/shorturl?access_token=ACCESS_TOKEN
```

参数说明

参数	是否必须	说明
access_token	是	调用接口凭证
action	是	此处填long2short，代表长链接转短链接
long_url	是	需要转换的长链接，支持http://、https://、weixin://wxpay 格式的url

调用举例

```
curl -d "{\"action\":\"long2short\",\"long_url\":\"http://wap.koudaitong.com/v2/showcase/goods?alias=128wi9shh&spm=h560
```

返回说明

正常情况下，微信会返回下述JSON数据包给公众号：

```
{"errcode":0,"errmsg":"ok","short_url":"http:\/\/w.url.cn\/s\/AvCo6Ih"}
```

参数说明

参数	说明
errcode	错误码
errmsg	错误信息
short_url	短链接

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"}

高级接口

条件必须是已`微信认证`的服务号才可获得

语音识别

客服接口

高级群发接口

设置用户备注名接口

OAuth2.0网页授权

生成带参数二维码

获取用户地理位置

获取用户基本信息

获取关注者列表

用户分组接口

上传下载多媒体文件

微信小店接口

条件 必须是已 `微信认证` 、已接入 `微信支付` 的号、且在添加功能插件中申请微信小店功能成功的公众号，才可获得

微信公众平台本次更新增加了微信小店功能，微信小店基于微信支付，包括添加商品、商品管理、订单管理、货架管理、维权等功能。

开发者可以通过小店接口来实现快速开店，目前支持以下接口：

1. 商品管理接口
    - 开发者可通过商品管理接口，来增加商品、删除商品、修改商品信息、查询已有商品，并可通过接口对商品进行上下架等操作管理。
2. 库存管理接口
    - 开发者可通过库存管理接口，来为已有商品增加和减少库存，包括进行与自身系统或其他平台的库存同步。
3. 邮费模板管理接口
    - 对于部分邮费计算复杂的商品，开发者可通过邮费模板管理接口，来生成、修改、删除和查询支持复杂邮费计算的邮费模板。
4. 分组管理接口
    - 对已有商品，开发者可通过分组管理接口，来对商品进行分组管理。接口包括增加、删除、修改和查询分组。
5. 货架管理接口
    - 微信商户除了可以在公众平台网站中自定义货架外，也可通过接口来增加、删除、修改和查询货架。货架也是通过控件来组成的。开发者甚至可以将自己的页面作为货架，通过JS API来调起商品详情页。
6. 订单管理接口
    - 开发者可按订单状态和时间来获取订单，并对订单进行发货。
7. 功能接口
    - 目前功能接口暂时只支持上传图片接口一项。微信商户开发接口中所有需要用到图片的地方，都需先使用上传图片接口来预先获得图片的URL。

请点击下载微信小店接口文档： 下载

模板消息

条件 必须是已 `微信认证` 、已接入 `微信支付` 的服务号才可获得.

介绍

- 模板消息接口让公众号可以向用户发送预设的模板消息。
- 模板消息仅用于公众号向用户发送业务通知。如信用卡刷卡通知，商品购买成功通知等。
- 模板消息只对认证的服务号开放。
- 模板消息在微信客户端的展示如图所示

使用规则

- 请根据运营规范使用模板消息，否则可能会被停止内测资格甚至封号惩罚
- 请勿使用模板发送垃圾广告或造成骚扰
- 请勿使用模板发送营销类消息
- 请在符合模板要求的场景时发送模板

支付开发教程

1. 使用网页授权接口 使用网页授权接口获取用户的基本信息。 【微信公众号】OAuth2.0授权.pdf 点击下载
2. 使用共享收货地址控件 使用共享收货地址控件，获取用户在微信的收货地址。 【微信公众号支付】收货地址共享接口文档V1.6.pdf 点击下载
3. 使用公众号发起支付请求 使用JS API在微信的网页中发起支付请求，详细方法见文档中有关JS API的章节。 使用Native API发起支付请求，详细方法文档中有关Native API的章节。 微信支付接口文档及demo(公众账号).zip 点击下载
4. 接入客服维权系统 所有公众号商户必须接入客服维权系统。接入客服维权系统后，用户可以通过自定义菜单中“维权”菜单，选择订单，向微信发起维权请求。微信将会把维权信息通过payfeedback_url通知到商户，商户必须在规定时间内处理维权请求。 【微信公众号支付】用户维权系统及接口说明V1.6.pdf 点击下载
5. 在第三方应用上发起支付请求 在iOS平台和android平台上的第三方应用，通过微信支付SDK发起支付请求。 微信支付接口文档及demo(APP).zip 点击下载
6. 微信支付退款&对账 使用微信支付退款和对账功能，帮助轻松实现退款和对账。 微信支付(退款&对账接口)文档.zip 点击下载

网页授权获得用户基本信息

如果用户在微信中（Web 微信除外）访问第三方网页，网页的开发者可以通过此接口 获取当前用户身份信息。利用用户信息，可以实现用户来源统计、帐号绑定、用户身份鉴权 等功能。 OAuth2.0 鉴权 关于 OAuth2.0 的详细介绍，可以参考 OAuth2.0 协议标准 第三方开发者进行微信 OAuth2.0 要符合以下条件： a.拥有一个微信公众账号，并获取到 appid、secret b.微信后台配置跳转域名以及授权 scope 类型（请联系对接的商务同事）。 跳转域名配置规范为全域名： 比如需要网页授权的域名为：www.qq.com，配置以后此域名下面的页面 http://www.qq.com/music.html、http://www.qq.com/login.html 都可以进行 OAuth2.0 鉴权。但 http://pay.qq.com、http://music.qq.com、http://qq.com 无法进行 OAuth2.0 鉴权。 授权流程 微信目前支持 Authorization code 授权模式，主要流程分为三步： 1、获取 code 2、通过 code 换取网页授权 accesstoken, 此 access_token 与基础支持的 access_token 不同。 3、通过 accesstoken 换取用户基本信息 流程图：实现方式： 第一步：引导需要授权的用户到如下页面地址，获取 CODE 地址说明 在确保微信公众账号拥有授权作用域（scope 参数）的权限的前提下，使用微信 客户端打开以下链接（严格按照以下格式）： https://open.weixin.qq.com/connect/oauth2/authorize?appid=APPID&redir ect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE#wecha t_redirect 若提示“该链接无法访问”，请检查参数是否填写错误，是否拥有 scope 参数对 应的授权作用域权限。 参考链接(请在微信客户端中打开此链接体验) Scope 为 snsapi_base https://open.weixin.qq.com/connect/oauth2/authorize?appid=wx520c15f417810387&redirect_uri=http%3A%2F%2Fchong.qq.com%2Fphp%2Findex.php%3Fd%3D%26c%3DwxAdapter%26m%3DmobileDeal%26showwxpaytitle%3D1%26vb2ctag%3D4_2030_5_1194_60&response_type=code&scope=snsapi_base&state=123#wecha t_redirectScope 为 snsapi_userinfo https://open.weixin.qq.com/connect/oauth2/authorize?appid=wxf0e81c3bee622d60&redirect_uri=http%3A%2F%2Fnba.bluewebgame.com%2Foauth_response.php&response_type=code&scope=snsapi_userinfo&state=STATE#wechat_red irect

参数说明 参数 是否必 须 说明 appid 是 应用唯一标识 redirect_uri 是 重定向地址 response_type 是 填 code scope 是 应用授权作用域，只能选择一个 snsapi_base （不弹出授权页面，直接跳转，这个只能拿到用户openid） snsapi_userinfo （弹出授权页面，这个可以通过 openid 拿到昵称、性别、所在 地） state 否 重定向后会带上 state 参数，开发者可以填写任意参数值

wechat_redirect 否

直接在微信打开链接，可以不填此参数。做页面 302 重定向时 候，必须带此参数。 获得用户授权结果如果用户同意授权，页面将跳转至 redirect*uri/?code=CODE&state=STATE*。若用户禁止授 权，则重定向后不会带上 *code* 参数，仅会带上 *state* 参数 *redirect_uri?state=STATE code* 说明 ： *code* 作为换取 *access_token* 的票据，每次用户授权带上的 *code* 将不一样，*code* 只能使用一 次，*5* 分钟未被使用自动过期。 通过 *code* 换取 *access_token* 请求方法 获取 *code* 后，请求以下链接获取 *access_token*： *https://api.weixin.qq.com/sns/oauth2/access_token?appid=APPID&secret=SECRET&code=CODE&grant_type=authorization_code* 参数说明 参数 是否必须 说明 *appid* 是 应用唯一标识 *secret* 是 应用密钥 *code* 是 填写第一步获取的 *code* 参数 *grant_type* 是 填 *authorization_code* 返回说明 正确的返回：*{ "access_token":"ACCESS_TOKEN", "expires_in":7200, "refresh_token":"REFRESH_TOKEN", "openid":"OPENID", "scope":"SCOPE" }* 参数 说明 *access_token* 接口调用凭证,注意：此 *access_token* 与基础支持的 *access_token* 不同 *expires_in access_token* 接口调用凭证超时时间，单位（秒） *refresh_token* 用户刷新 *access_token openid* 用户唯一标识*scope* 用户授权的作用域 错误返回样例：*{"errcode":40029,"errmsg":"invalid code"}* 刷新 *access_token* 由于 *access_token* 拥有较短的有效期，当 *access_token* 超时后，可以使用 *refresh_token* 进行 刷新，*refresh_token* 拥有较长的有效期（*7* 天、*30* 天、*60* 天、*90* 天），当 *refresh_token* 失 效的后，需要用户重新授权。 请求方法 获取第二步的 *refresh_token* 后，请求以下链接获取 *access_token*： *https://api.weixin.qq.com/sns/oauth2/refresh_token?appid=APPID&grant*type=refresh_token&refresh_token=REFRESH_TOKEN 参数说明 参数 是否必须 说明 appid 是 应用唯一标识 grant_type 是 填 refresh_token refresh_token 是 填写通过 access_token 获取到的 refresh_token 参数 返回说明 正确的返回：{ "access_token":"ACCESS_TOKEN", "expires_in":7200, "refresh_token":"REFRESH_TOKEN", "openid":"OPENID", "scope":"SCOPE" } 参数说明 参数 说明 access_token 接口调用凭证 expires_in access_token 接口调用凭证超时时间， 单位（秒） refresh_token 用户刷新 access_token openid 授权用户唯一标识 scope 用户授权的作用域，使用逗号（,）分隔 错误返回样例： {"errcode":40029,"errmsg":"invalid code"} 通过 access_token 拉取用户信息(仅限 scope= snsapi_userinfo)： http：GET https://api.weixin.qq.com/sns/userinfo?access_token=ACCESS_TOKEN?access_token=ACCESS_TOKEN&openid=OPENID 参数说明 参数 是否必须 说明 access_token 是 调用凭证 openid 是 普通用户的标识，对当前开发者账号唯一 返回说明 正确的 Json 返回： { "openid":" OPENID", " nickname": NICKNAME, "sex":"1", "province":"PROVINCE" "city":"CITY", "country":"COUNTRY", "privilege":["PRIVILEGE1" "PRIVILEGE2"] } 参数 说明

openid 普通用户的标识，对当前开发者账号唯一nickname 普通用户昵称 sex 普通用户个人资料填写的省份 province 用户授权的作用域，使用逗号（,）分隔 city 普通用户个人资料填写的城市 country 国家，如中国为CN privilege 用户特权信息，json 数组，如微信沃卡用户为（chinaunicom） 错误返回样例：{"errcode":40003,"errmsg":" invalid openid "}

微信公众号支付收货地址共享接口文档 **V1.5**

收货地址共享接口文档 收货地址共享接口文档 收货地址共享接口文档 收货地址共享接口文档V1. V1. V1. V1.6666微信公众号支付收货地址共享接口文档 V1.5 1 收货地址共享简介 收货地址共享简介 收货地址共享简介 收货地址共享简介 1.1 1.1 1.1 1.1 功能简介 功能简介 功能简介 功能简介 微信收货地址共享，是指用户在微信浏览器内打开商品H5 网页，填写过地 址，后续支持快速选择免填写，也可增加和编辑。此地址为用户属性，可在商户 网页中共享使用。支持原生控件填写地址，地址数据会传递到商户后台。

1.
    i.
        i. 1.2222 功能支持 功能支持 功能支持 功能支持 地址共享是基于微信JS API 实现，只能在微信内置浏览器中使用，其他浏 览器调用无效。同时，需要微信5.0 版本才能支持，建议通过user agent 来确 定用户当前的版本号后再调用地址接口。以iPhone 版本为例，可以通过user agent可获取如下微信版本示例信息："Mozilla/5.0(iphone;CPU iphone OS 5_1_1 like Mac OS X) AppleWebKit/534.46(KHTML,like Geocko) Mobile/9B206 MicroMessenger/5.0" 其中5.0 为用户安装的微信版本号，商户可以判定版本号是否高于或者等于5.0。
2.
    i.
        i. 1.3333 地址格式 地址格式 地址格式 地址格式 微信地址共享使用的数据字段包括：
3. 收货人姓名
4. 地区，省市区三级
5. 详细地址
6. 邮编
7. 联系电话 其中，地区对应是国标三级地区码，如“广东省-广州市-天河区”，对应的微信公众号支付收货地址共享接口文档 V1.5 是510630。详情参考链接： http://www.stats.gov.cn/tjsj/tjbz/xzqhdm/201401/t20140116_501070.ht ml 2222 公众号收货地址 公众号收货地址 公众号收货地址 公众号收货地址JSJSJSJS API API API API 接口定义 接口定义 接口定义 接口定义 2.1 2.1 2.1 2.1 编辑并获取收货地址 编辑并获取收货地址 编辑并获取收货地址 编辑并获取收货地址 公众号编辑收货地址 editAddress editAddress editAddress editAddress 参数列表 参数列表 参数列表 参数列表 参数 必填 说明 appId 是 公众号appID scope 是 填写“jsapi_address”，获得编辑地址权限 signType 是 签名方式，目前仅支持SHA1 addrSign 是 签名 timeStamp 是 时间戳 nonceStr 是 随机字符串 说明：addrSign 的生成规则如下。参与addrSign 签名的字段包括：appId、 url（当前网页url）、timestamp、noncestr、accessToken（用户授权凭证， 请参照oauth2.0 协议获取）。这里scope、signType 并不参与签名。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。这里需 要注意的是所有参数名均为小写字符，例如appId 在排序后字符串则为appid；微信公众号支付收货地址共享接口文档 V1.5 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addrSign = SHA1(string1)。这里给出生成 addrSign 的具体示例如 下： 示例： 示例： 示例： 示例： appId=wx17ef1eaef46752cb url=http://open.weixin.qq.com/ timeStamp=1384841012 nonceStr=123456 accessToken=OezXcEiiBSKSxW0eoyIIeBFk1b8VbNtfWALJ5g6aMgZHaqZ wK4euEskSn78Qd5pLsfQtuMdgmhajVM5QDm24W8X3tJ18kz5mhmkUcI 3RoLm7qGgh1cEnCHejWQo8s5L3VvsFAdawhFxUuLmgh5FRA i：经过a过程键值对排序后得到string1 为： accesstoken=OezXcEiiBSKSxW0eoyIIeBFk1b8VbNtfWALJ5g6aMgZHaqZ wK4euEskSn78Qd5pLsfQtuMdgmhajVM5QDm24W8X3tJ18kz5mhmkUcI 3RoLm7qGgh1cEnCHejWQo8s5L3VvsFAdawhFxUuLmgh5FRA&appid=w x17ef1eaef46752cb&noncestr=123456&timestamp=1384841012&url= http://open.weixin.qq.com/ ii：经过b过程签名后可得到： addrSign=SHA1(accesstoken=OezXcEiiBSKSxW0eoyIIeBFk1b8VbNtfWA LJ5g6aMgZHaqZwK4euEskSn78Qd5pLsfQtuMdgmhajVM5QDm24W8X3 tJ18kz5mhmkUcI3RoLm7qGgh1cEnCHejWQo8s5L3VvsFAdawhFxUuLmg h5FRA&appid=wx17ef1eaef46752cb&noncestr=123456&timestamp=1微信公众号支付收货地址共享接口文档 V1.5 384841012&url=http://open.weixin.qq.com/)=ca604c740945587544a9c c25e58dd090f200e6fb 参数返回： 参数返回： 参数返回： 参数返回： 返回值 说明 err_msg edit_address:ok获取编辑收货地址成功 edit_address:fail获取编辑收货地址失败 userName 收货人姓名 telNumber 收货人电话 addressPostalCode 邮编 proviceFirstStageName 国标收货地址第一级地址 addressCitySecondStageName 国标收货地址第二级地址 addressCountiesThirdStageName 国标收货地址第三级地址 addressDetailInfo 详细收货地址信息 nationalCode 收货地址国家码 使用示例 使用示例 使用示例 使用示例 WeixinJSBridge.invoke('editAddress',{ "appId" : getAppId(), "scope" : "jsapi_address", "signType" : "sha1", "addrSign" : "xxxxx", "timeStamp" : "12345", "nonceStr" : "10000",微信公众号支付收货地址共享接口文档 V1.5 },function(res){ //若

res 中所带的返回值不为空，则表示用户选择该返回值作为收货地 址。否则若返回空，则表示用户取消了这一次编辑收货地址。 document.form1.address1.value =res.proviceFirstStageName; document.form1.address2.value =res.addressCitySecondStageName; document.form1.address3.value =res.addressCountiesThirdStageName; document.form1.detail.value = res.addressDetailInfo; document.form1.phone.value = res.telNumber; }); }); 3333 基本交互流程 基本交互流程 基本交互流程 基本交互流程 基本交互流程如下：微信公众号支付收货地址共享接口文档 V1.5 4444 收货地址共享 收货地址共享 收货地址共享 收货地址共享JSJSJSJS API API API API 申请流程简介 申请流程简介 申请流程简介 申请流程简介 （1）若要获取收货地址相关JS API，需要先申请具有商户支付权限的公众号；（2）在获取具有支付权限的appID 后，默认已开启收货地址相关JS API 权限， 但需要商家配置对应使用收货地址功能的域名，域名配置请在公众平台网站 公众平台网站 公众平台网站 公众平台网站----服服服服 务务务务----我的服务 我的服务 我的服务 我的服务-OAuth2.0 -OAuth2.0 -OAuth2.0 -OAuth2.0 网页授权 网页授权 网页授权 网页授权中设置（针对已认证的公众号），如下图： 点击修改，把授权回调页面域名设置为使用收货地址功能的域名即可。微信公众号支付收货地址共享接口文档 V1.5 5555 报错后的调试方向 报错后的调试方向 报错后的调试方向 报错后的调试方向 （1）签名参数必须小写 （2）调用接口参数必须用字符串格式 （3）token需要用oauth获取的token，获取token的scope是snsapi_base （4）用文档验证签名算法正确性 （5）签名使用的url必须是调用时所在页面的url，此url域名要与填写Oauth2.0 授权域名一致 （6）参与签名使用的url必须带上微信服务器返回的code和state参数

用户维权系统接口文档 V1.6

微信公众号支付 微信公众号支付 微信公众号支付 微信公众号支付 用户维权系统接口文档 用户维权系统接口文档 用户维权系统接口文档 用户维权系统接口文档 V1.6 V1.6 V1.6 V1.6用户维权系统接口文档 V1.6 1 微信支付客单投诉简介.....3 1.1 功能简介.....3 1.2 支付账户.....4 2 通知接口说明.....5 2.1 通知接口简介.....5 2.2 通知接口参数.....5 2.4 后台通知签名方式.....8 3 API 接口说明.....8 3.1 API 接口简介.....8 3.2 API 使用方式.....8 3.3 API 列表.....9 4.MP 系统-维权模块指引.....9 4.1 维权单查看&处理.....9 4.2 自定义投诉原因&解决方案.....10 5.商户和用户沟通建议.....10

1. 客服接口.....10用户维权系统接口文档 V1.6 1111 微信支付客单投诉简介 微信支付客单投诉简介 微信支付客单投诉简介 微信支付客单投诉简介 1.1 1.1 1.1 1.1 功能简介 功能简介 功能简介 功能简介 • 背景 用户在公众号进行支付购买行为出现异常时，通常会投诉到腾讯客服，因此微信侧需要 即时了解公众号与用户交易的详情。同时，为了最快效率的解决用户的问题，微信作为连接 用户和商户的桥梁，会通过此客服系统即时将问题同步给商户，并将解决结果反馈至用户。 • 接入方要求 接入微信支付的商户都必须接入此客服系统。 • 接入产品说明 1) 入口：调通维权接口将作为商户全网发布的前提条件之一。成功全网发布的商户，将自 动在 profile 页增加该商户的“交易维权”入口，默认指向维权页面； 2) 发货：商品的发货状态将作为判断维权责任的重要依据，请商户务必调通发货接口，并 确保每一笔订单都如实返回发货状态。虚拟、服务类商品必须 24 小时内发货，实物类 商品必须在 72 小时内发货，详细请查看《微信公众号支付接口文档》。 3) 维权发起时限：用户可对已发货或超时未发货，且支付成功后 45 天内的订单发起维权。 4) 维权处理时限：超时未发货状态下用户发起的维权，商户(虚拟物品交易、服务交易) 必须在 1 天内解决用户的投诉，商户(实体物品交易)必须在 2 天内解决用户的投诉；已用户维权系统接口文档 V1.6 发货状态下用户发起的维权，商户(虚拟物品交易、服务交易)必须在 3 天内解决用户的 投诉，商户(实体物品交易)必须在 15 天内解决用户的投诉。 5) 超时未处理的维权单，或处理结果用户不满意的维权单，用户可要求微信平台介入仲裁。 6) 请务必确认已按照与用户沟通的结果处理完毕后再进行销单，否则用户可对处理结果不 满意的维权单申请微信介入仲裁，届时有可能影响商户信誉。 7) 此版本新增 MP 系统查看、处理维权单的功能，商户在确保调通维权接口后，可以选择 自己搭建系统，或直接在 MP 系统处理维权单。 • 流程示意 1.2 1.2 1.2 1.2 支付账户 支付账户 支付账户 支付账户 商户向微信公众平台提交企业信息以及银行账户资料，审核通过并签约后，可以获得以用户维权系统接口文档 V1.6 下帐户（包含财付通的相关支付资金账户），用于公众号支付。 帐号帐号帐号帐号 作用作用作用作用 appId 公众号身份标识。 appSecret 公众平台 API 的权限获取所需密钥 Key，在使用所有公众平台 API 时，都 需要先用它去换取 access_token,然后再进行调用。 paySignKey 公众号支付请求中用于加密的密钥 Key，可验证商户唯一身份，PaySignKey 对应于支付场景中的 appKey 值。 partnerId 财付通商户身份标识。 partnerKey 财付通商户权限密钥 Key。 注意：appSecret、paySignKey、partnerKey 是验证商户唯一性的安全标识，请妥善保管。 对于 appSecret 和 paySignKey 的区别，可以这样认为：appSecret 是 API 使用时的登录 密码，会在网络中传播的；而 paySignKey 是在所有支付相关数据传输时用于加密并进行身 份校验的密钥，仅保留在第三方后台和微信后台，不会在网络中传播。 2222 通知接口说明 通知接口说明 通知接口说明 通知接口说明 2.1 2.1 2.1 2.1 通知接口简介 通知接口简介 通知接口简介 通知接口简介 用户在新增投诉单后，微信后台通知（post）商户服务器（payfeedback_url）支付结果。 商户可以使用 payfeedback_url 的通知结果进行个性化页面的展示。 注：payfeedback_url 请提交至微信相关接口人，微信侧登记后即可用。用户维权系统接口文档 V1.6 2.2 2.2 2.2 2.2 通知接口参数 通知接口参数 通知接口参数 通知接口参数 后台通知通过请求中的 payfeedback_url 进行，采用 post 机制。postData 中将包含 xml 数据。 数据如下：
(1) 用户新增投诉的 xml <![CDATA[oDF3iY9P32sK_5GgYiRkjsCo45bk]]><![CDATA[wxf 8b4f85f3a794e77]]> 1393400471 <![CDATA[request]]> 7197417460812502768 <![CDATA[1900000109201402143240185685]]> <![CDATA[质量问题]]> <![CDATA[换货]]> <![CDATA[备注 12435321321]]> <![CDATA[d60293982cc7c97a5a9d3383af761db763c07c86]]> <![CDATA[sha1]]> ![CDATA[http://mmbiz.qpic.cn/mmbiz/49ogibiahRNtOk37iaztwmdgFbyFS9FU

rqfodiaUAmxr4hOP34C6R4nGgebMalKuY3H35riaZ5vtzJh25tp7vBUwWxw/0]] ! [CDATA[http://mmbiz.qpic.cn/mmbiz/49ogibiahRNtOk37iaztwmdgFbyFS9FUrqfn3y72eHKRS AwVz1PyIcUSjBrDzXAibTiaAdrTGb4eBFbib9ibFaSeic3OIg/0]] <![CDATA[]]><![CDATA[]]><PicUrl

> <![CDATA[]]> (2) 用户确认处理完毕投诉的 xml <![CDATA[111222]]> <![CDATA[wwwwb4f85f3a797777]]>用户维权系统接口文档 V1.6 1369743511 <![CDATA[confirm/reject]]> <![CDATA[5883726847655944563]]> <![CDATA[商品质量有问题]]> <![CDATA[bafe07f060f22dcda0bfdb4b5ff756f973aecffa]]> <![CDATA[sha1]]></ SignMethod > 各字段定义如下： 参数参数参数参数 必填必填必填必填 说明说明说明说明 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 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。 MsgType 是 通知类型 request 用户提交投诉 confirm 用户确认消除 投诉 reject 用户拒绝消除投诉 FeedBackId 是 投诉单号 TransId 否 交易订单号用户维权系统接口文档 V1.6 Reason 否 用户投诉原因 Solution 否 用户希望解决方案 ExtInfo 否 备注信息+电话 PicUrl 否 用户上传的图片凭证，最多五张 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、openid。 2.4 2.4 2.4 2.4 后台通知签名方式 后台通知签名方式 后台通知签名方式 后台通知签名方式 同微信支付签名方式 3333 API API API API 接口说明 接口说明 接口说明 接口说明 3.1 3.1 3.1 3.1 API API API API 接口简介 接口简介 接口简介 接口简介 商户可以通过 api 调用，标记客户的投诉处理状态。 3.2 3.2 3.2 3.2 API API API API 使用方式 使用方式 使用方式 使用方式 在使用 API 之前，需要拥有 API 使用过程中用到的具有时效性的凭证 access_token。这 个获取方式就是使用前文提到的 appid 和 appsecret 调用 token 这个 api 获取。更详细的文档 请参考：用户维权系统接口文档 V1.6 http://mp.weixin.qq.com/wiki/index.php?title=%E9%80%9A%E7%94%A8%E6%8E%A5% E5%8F%A3%E6%96%87%E6%A1%A3 只有拥有了有效 access_token（即相当于具有了一定时间内的 API 登录态），后续的所 有 API 调用才会成功。 3.3 3.3 3.3 3.3 API API API API 列表列表列表列表 （1）标记客户的投诉处理状态。 updatefeedback Api 的 url 为：https://api.weixin.qq.com/payfeedback/update?access_token=xxxxx&openid=XXXX&fee dbackid=xxxx Url 中的参数包含目前微信公众平台凭证 access_token，和客户投诉对应的单号 feedbackid，以及 openid 微信公众平台在校验 ok 之后，会返回数据表明是否通知成功，例如： {"errcode":0,"errmsg":"ok"} 如果有异常，会在 errcode 和 errmsg 描述出来，如果成功 errcode 就为 0。 4.MP 4.MP 4.MP 4.MP 系统系统系统系统----维权模块指引 维权模块指引 维权模块指引 维权模块指引 4.1 4.1 4.1 4.1 维权单查看 维权单查看 维权单查看 维权单查看&&&&处理处理处理处理 MP 系统新增维权处理界面，让商户更方便地查看、处理维权单。 （1） 入口：登录 MP 系统：（https://mp.weixin.qq.com/），选择 功能---商户功能---维权仲 裁---维权单用户维权系统接口文档 V1.6 （2） 查看维权单：商户可在此查看所有的维权单，并且可按投诉日期排序，按维权单状 态筛选；点击空白处可以展开维权单详情。 （3） 处理维权单：当商户与用户沟通完毕，并按沟通结果完成相应的退款、退货操作后， 可在系统发起销单。在该页面的“操作”列点击“确认”即可。请务必确认已按照 与用户沟通的结果处理完毕后再进行销单，否则用户可对处理结果不满意的维权单 申请微信介入仲裁，届时有可能影响商户信誉。 4.2 4.2 4.2 4.2 自定义投诉原因 自定义投诉原因 自定义投诉原因 自定义投诉原因&&&&解决方案 解决方案 解决方案 解决方案 考虑到每个商户的个性化需求，特提供商户在 MP 系统自定义投诉原因&解决方案的入口， 商户最多可各增加 3 个投诉原因和解决方案。方法如下： 在上述页面，选择“自定义维权原因&解决方案“，选择对应操作后，点击”“添加”，即可 进行编辑。文案限 15 个字以内。

2.
    i.
        i. 5.商户和用户沟通建议 商户和用户沟通建议 商户和用户沟通建议 商户和用户沟通建议 商户在接到投诉后，常常有和用户沟通的需求，以下是商户和用户进行沟通的建议方式： 微信已经提供的用户信息有 openid 和维权单详情（请商户务必存储以上信息，供后续备查）， 若商户有主动下发消息与用户沟通的需求，可以向微信接口人提出申请模版消息接口的权 限，若商户有与用户进行多次一对一沟通的需求，可以向微信接口人提出申请客服消息接口 的权限。 有开发资源的开发者也可以自行开发一套客服系统，与微信维权系统进行对接。用户维权系统接口文档 V1.6
3.
    i.
        i.
            i. 客服接口 客服接口 客服接口 客服接口 当用户主动发消息给公众号的时候（包括发送信息、点击自定义菜单 clike 事件、订阅事 件、扫描二维码事件、支付成功事件、用户维权），微信将会把消息数据推送给开发者， 开发者在一段时间内（目前为 24 小时）可以调用客服消息接口，通过 POST 一个 JSON 数据包来发送消息给普通用户，在 24 小时内不限制发送次数。此接口主要用于客服等有 人工消息处理环节的功能，方便开发者为用户提供更加优质的服务。 接口调用请求说明 http 请求方式: POST

https://api.weixin.qq.com/cgi-bin/message/custom/send?access_token=ACCESS_TOKEN 各消息类型所需的 JSON 数据包如下。

4.
    i.
        i.
            i. 发送文本消息 发送文本消息 发送文本消息 发送文本消息 { "touser":"OPENID", "msgtype":"text", "text": { "content":"Hello World" } } 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid 用户维权系统接口文档 V1.6 msgtype 是 消息类型，text content 是 文本消息内容

5.
    i.
        i.
            i. 发送图片消息 发送图片消息 发送图片消息 发送图片消息 { "touser":"OPENID", "msgtype":"image", "image": { "media_id":"MEDIA_ID" } }

6.
    i.
        i.
            i. 发送语音消息 发送语音消息 发送语音消息 发送语音消息 { "touser":"OPENID", "msgtype":"voice", "voice": { "media_id":"MEDIA_ID" } } 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid msgtype 是 消息类型，image media_id 是 发送的图片的媒体 ID用户维权系统接口文档 V1.6 msgtype 是 消息类型，voice media_id 是 发送的语音的媒体 ID

7.
    i.
        i.
            i. 发送视频消息 发送视频消息 发送视频消息 发送视频消息 { "touser":"OPENID", "msgtype":"video", "video": { "media_id":"MEDIA_ID", "thumb_media_id":"THUMB_MEDIA_ID" } } 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid msgtype 是 消息类型，video media_id 是 发送的视频的媒体 ID thumb_media_id 是 视频缩略图的媒体 ID

8.
    i.
        i.
            i. 发送音乐消息 发送音乐消息 发送音乐消息 发送音乐消息 { "touser":"OPENID", "msgtype":"music", "music": { "title":"MUSIC_TITLE", "description":"MUSIC_DESCRIPTION",用户维权系统接口文档 V1.6 "musicurl":"MUSIC_URL", "hqmusicurl":"HQ_MUSIC_URL", "thumb_media_id":"THUMB_MEDIA_ID" } } 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid msgtype 是 消息类型，music title 否 音乐标题 description 否 音乐描述 musicurl 是 音乐链接 hqmusicurl 是 高品质音乐链接，wifi 环境优先使用该链接播放音乐 thumb_media_id 是 缩略图的媒体 ID

9.
    i.
        i.
            i. 发送图文消息 发送图文消息 发送图文消息 发送图文消息 图文消息条数限制在 10 条以内，注意，如果图文数超过 10，则将会无响应。 { "touser":"OPENID", "msgtype":"news", "news":{ "articles": [{ "title":"Happy Day", "description":"Is Really A Happy Day", "url":"URL",用户维权系统接口文档 V1.6 "picurl":"PIC_URL" }, { "title":"Happy Day", "description":"Is Really A Happy Day", "url":"URL", "picurl":"PIC_URL" }] } } 参数 是否必须 说明 access_token 是 调用接口凭证 touser 是 普通用户 openid msgtype 是 消息类型，news title 否 标题 description 否 描述 url 否 点击后跳转的链接 picurl 否 图文消息的图片链接，支持 JPG、PNG 格式，较好的效果 为大图 640*320*，小图 *80*80

内置浏览器

微信内置浏览器那些坑爹的事

WeixinApi

微信内置浏览器常用接口

zxlie/WeixinApi

angularjs的services

```
'use strict';

angular.module('worldApp')
.factory('Wechat',function($document,$location,$window) {
    var Wechat = function(data){
      var data  = data || {};
      var url   = data.url ?  data.url  :$location.absUrl();
      var title = data.title? data.title:$document[0].title;
      var desc  = data.desc?  data.desc : url;
      var img   = data.img ?  data.img  : 'http://img.***.com/images/logo.png';
      var onBridgeReady = function(){
        WeixinJSBridge.on('menu:share:appmessage', function () {
          WeixinJSBridge.invoke('sendAppMessage', {
            title: title,
            img_url: img,
            img_width: '120',
            img_height: '120',
            desc: desc,
            link: url
          }, function (e) {WeixinJSBridge.log(e.err_msg);});
        });
        WeixinJSBridge.on('menu:share:timeline', function () {
          WeixinJSBridge.invoke('shareTimeline', {
            title: title,
            img_url: img,
            img_width: '120',
            img_height: '120',
            desc: title,
            link: url
          }, function (e) {WeixinJSBridge.log(e.err_msg);});
        });
      };
      if (typeof WeixinJSBridge == "undefined"){
        $document.bind('WeixinJSBridgeReady',onBridgeReady);
      }else{
        onBridgeReady();
      }
    };
    return Wechat;
  })
.factory('userAgent',function(){
    this.isAndroid =  function() {
      return navigator.userAgent.match(/Android/i);
    };
    this.isIOS = function() {
      return navigator.userAgent.match(/iPhone|iPad|iPod/i);
    };
    this.isBlackBerry = function(){
      return navigator.userAgent.match(/BlackBerry/i);
    };
    this.isOpera = function(){
      return navigator.userAgent.match(/Opera Mini/i);
    };
    this.isWindows = function(){
      return navigator.userAgent.match(/IEMobile/i);
    };
    this.mobile = function(){
      return (this.isAndroid || this.isIOS || this.isBlackBerry || this.isOpera || this.isWindows)
```

```
    };
    return this;
  })
```

关于分享按钮问题

如果使用单页js框架，在android的微信内置浏览器中，第一次加载可以用分享按钮，第二页就失效，暂时没找到问题所在，只能用暴力刷新的方式了。

```
'use strict';

angular.module('worldApp')
.controller('GroupCtrlInfo',function($scope,$routeParams,$location,$cookies,Wechat,userAgent) {
    if($cookies.groupId !== $routeParams.groupId&&userAgent.isAndroid()){
        $cookies.groupId = $routeParams.groupId;
        location.reload();
    }
});
```

JS-SDK概述

微信JS-SDK是微信公众平台面向网页开发者提供的基于微信内的网页开发工具包。

通过使用微信JS-SDK，网页开发者可借助微信高效地使用拍照、选图、语音、位置等手机系统的能力，同时可以直接使用微信分享、扫一扫、卡券、支付等微信特有的能力，为微信用户提供更优质的网页体验。

此文档面向网页开发者介绍微信JS-SDK如何使用及相关注意事项。

使用说明

在使用微信JS-SDK对应的JS接口前，需确保公众号已获得使用对应JS接口的权限，可登录微信公众平台进入“开发者中心”查看对应的接口权限。

注意： 所有的JS接口只能在公众号绑定的域名下调用，公众号开发者需要先登录微信公众平台进入“公众号设置”》“功能设置”里填写“JS接口安全域名”。

步骤一：引入JS文件

在需要调用JS接口的页面引入如下JS文件，（支持https）：http://res.wx.qq.com/open/js/jweixin-1.0.0.js

备注：支持使用 AMD/CMD 标准模块加载方法加载

步骤二：通过config接口注入权限验证配置

所有需要使用JS-SDK的页面必须先注入配置信息，否则将无法调用（同一个url仅需调用一次，对于变化url的SPA的web app可在每次url变化时进行调用）。

```
wx.config({
    debug: true, // 开启调试模式,调用的所有api的返回值会在客户端alert出来，若要查看传入的参数，可以在pc端打开，参数信息会通过log打出，仅
    appId: '', // 必填，公众号的唯一标识
    timestamp: , // 必填，生成签名的时间戳
    nonceStr: '', // 必填，生成签名的随机串
    signature: '',// 必填，签名，见附录1
    jsApiList: [] // 必填，需要使用的JS接口列表，所有JS接口列表见附录2
});
```

步骤三：通过ready接口处理成功验证

```
wx.ready(function(){

    // config信息验证后会执行ready方法，所有接口调用都必须在config接口获得结果之后，config是一个客户端的异步操作，所以如果需要在页面加载
});
```

步骤四：通过error接口处理失败验证

```
wx.error(function(res){

    // config信息验证失败会执行error函数，如签名过期导致验证失败，具体错误信息可以打开config的debug模式查看，也可以在返回的res参数中查看

});
```

接口调用说明

所有接口通过wx对象(也可使用jWeixin对象)来调用，参数是一个对象，除了每个接口本身需要传的参数之外，还有以下通用参数：

- success：接口调用成功时执行的回调函数。
- fail：接口调用失败时执行的回调函数。
- complete：接口调用完成时执行的回调函数，无论成功或失败都会执行。
- cancel：用户点击取消时的回调函数，仅部分有用户取消操作的api才会用到。
- trigger: 监听Menu中的按钮点击时触发的方法，该方法仅支持Menu中的相关接口。

以上几个函数都带有一个参数，类型为对象，其中除了每个接口本身返回的数据之外，还有一个通用属性errMsg，其值格式如下：

- 调用成功时："xxx:ok"，其中xxx为调用的接口名
- 用户取消时："xxx:cancel"，其中xxx为调用的接口名 调用失败时：其值为具体错误信息

基础接口

判断当前客户端版本是否支持指定JS接口

```
wx.checkJsApi({
    jsApiList: ['chooseImage'] // 需要检测的JS接口列表，所有JS接口列表见附录2,
    success: function(res) {
        // 以键值对的形式返回，可用的api值true，不可用为false
        // 如：{"checkResult":{"chooseImage":true},"errMsg":"checkJsApi:ok"}
    });
```

备注：checkJsApi接口是客户端6.0.2新引入的一个预留接口，第一期开放的接口均可不使用checkJsApi来检测。

分享接口

请注意不要有诱导分享等违规行为，对于诱导分享行为将永久回收公众号接口权限，详细规则请查看：朋友圈管理常见问题。

获取“分享到朋友圈”按钮点击状态及自定义分享内容接口

```
wx.onMenuShareTimeline({
    title: '', // 分享标题
    link: '', // 分享链接
    imgUrl: '', // 分享图标
    success: function () {
        // 用户确认分享后执行的回调函数
    },
    cancel: function () {
        // 用户取消分享后执行的回调函数
    }
});
```

获取“分享给朋友”按钮点击状态及自定义分享内容接口

```
wx.onMenuShareAppMessage({
    title: '', // 分享标题
    desc: '', // 分享描述
    link: '', // 分享链接
    imgUrl: '', // 分享图标
    type: '', // 分享类型,music、video或link，不填默认为link
    dataUrl: '', // 如果type是music或video，则要提供数据链接，默认为空
    success: function () {
        // 用户确认分享后执行的回调函数
    },
    cancel: function () {
        // 用户取消分享后执行的回调函数
    }
});
```

获取“分享到QQ”按钮点击状态及自定义分享内容接口

```
wx.onMenuShareQQ({
    title: '', // 分享标题
    desc: '', // 分享描述
    link: '', // 分享链接
    imgUrl: '' // 分享图标
    success: function () {
       // 用户确认分享后执行的回调函数
    },
    cancel: function () {
       // 用户取消分享后执行的回调函数
    }
});
```

获取“分享到腾讯微博”按钮点击状态及自定义分享内容接口

```
wx.onMenuShareWeibo({
    title: '', // 分享标题
    desc: '', // 分享描述
    link: '', // 分享链接
    imgUrl: '' // 分享图标
    success: function () {
       // 用户确认分享后执行的回调函数
    },
    cancel: function () {
        // 用户取消分享后执行的回调函数
```

```
        }
```

});

图像接口

拍照或从手机相册中选图接口

```
wx.chooseImage({
    success: function (res) {
        var localIds = res.localIds; // 返回选定照片的本地ID列表，localId可以作为img标签的src属性显示图片
    }
});
```

预览图片接口

```
wx.previewImage({
    current: '', // 当前显示的图片链接
    urls: [] // 需要预览的图片链接列表
});
```

上传图片接口

```
wx.uploadImage({
    localId: '', // 需要上传的图片的本地ID，由chooseImage接口获得
    isShowProgressTips: 1// 默认为1，显示进度提示
    success: function (res) {
        var serverId = res.serverId; // 返回图片的服务器端ID
    }
});
```

备注：可用微信下载多媒体文件接口下载上传的图片，此处获得的 serverId 即 media_id，参考文档上传下载多媒体文件

下载图片接口

```
wx.downloadImage({
    serverId: '', // 需要下载的图片的服务器端ID，由uploadImage接口获得
    isShowProgressTips: 1// 默认为1，显示进度提示
    success: function (res) {
        var localId = res.localId; // 返回图片下载后的本地ID
    }
});
```

音频接口

开始录音接口

```
wx.startRecord();
```

停止录音接口

```
wx.stopRecord({
    success: function (res) {
        var localId = res.localId;
    }
});
```

监听录音自动停止接口

```
wx.onVoiceRecordEnd({
    // 录音时间超过一分钟没有停止的时候会执行 complete 回调
    complete: function (res) {
        var localId = res.localId;
    }
});
```

播放语音接口

```
wx.playVoice({
    localId: '' // 需要播放的音频的本地ID，由stopRecord接口获得
});
```

暂停播放接口

```
wx.pauseVoice({
    localId: '' // 需要暂停的音频的本地ID，由stopRecord接口获得
});
```

停止播放接口

```
wx.stopVoice({
    localId: '' // 需要停止的音频的本地ID，由stopRecord接口获得
});
```

监听语音播放完毕接口

```
wx.onVoicePlayEnd({
    serverId: '', // 需要下载的音频的服务器端ID，由uploadVoice接口获得
    success: function (res) {
        var localId = res.localId; // 返回音频的本地ID
    }
});
```

上传语音接口

```
wx.uploadVoice({
    localId: '', // 需要上传的音频的本地ID，由stopRecord接口获得
    isShowProgressTips: 1// 默认为1，显示进度提示
        success: function (res) {
        var serverId = res.serverId; // 返回音频的服务器端ID
    }
});
```

备注：可用微信下载多媒体文件接口下载上传的语音，此处获得的 serverId 即 media_id，参考文档上传下载多媒体文件

下载语音接口

```
wx.downloadVoice({
    serverId: '', // 需要下载的音频的服务器端ID，由uploadVoice接口获得
    isShowProgressTips: 1// 默认为1，显示进度提示
    success: function (res) {
        var localId = res.localId; // 返回音频的本地ID
    }
});
```

智能接口

识别音频并返回识别结果接口

```
wx.translateVoice({
   localId: '', // 需要识别的音频的本地Id，由录音相关接口获得
    isShowProgressTips: 1, // 默认为1，显示进度提示
    success: function (res) {
        alert(res.translateResult); // 语音识别的结果
    }
});
```

设备信息

获取网络状态接口

```
wx.getNetworkType({
    success: function (res) {
        var networkType = res.networkType; // 返回网络类型2g, 3g, 4g, wifi
    }
});
```

地理位置

使用微信内置地图查看位置接口

```
wx.openLocation({
    latitude: 0, // 纬度，浮点数，范围为90 ~ -90
    longitude: 0, // 经度，浮点数，范围为180 ~ -180。
    name: '', // 位置名
    address: '', // 地址详情说明
    scale: 1, // 地图缩放级别,整形值,范围从1~28。默认为最大
    infoUrl: '' // 在查看位置界面底部显示的超链接,可点击跳转
});
```

获取地理位置接口

```
wx.getLocation({
    timestamp: 0, // 位置签名时间戳，仅当需要兼容6.0.2版本之前时提供
    nonceStr: '', // 位置签名随机串，仅当需要兼容6.0.2版本之前时提供
    addrSign: '', // 位置签名，仅当需要兼容6.0.2版本之前时提供，详见附录4
    success: function (res) {
       var longitude = res.longitude; // 纬度，浮点数，范围为90 ~ -90
       var latitude = res.latitude; // 经度，浮点数，范围为180 ~ -180。
        var speed = res.speed; // 速度，以米/每秒计
        var accuracy = res.accuracy; // 位置精度
    }
});
```

界面操作

隐藏右上角菜单接口

```
wx.hideOptionMenu();
```

显示右上角菜单接口

```
wx.showOptionMenu();
```

关闭当前网页窗口接口

```
wx.closeWindow();
```

批量隐藏功能按钮接口

```
wx.hideMenuItems({
    menuList: [] // 要隐藏的菜单项，所有menu项见附录3
});
```

批量显示功能按钮接口

```
wx.showMenuItems({
    menuList: [] // 要显示的菜单项，所有menu项见附录3
});
```

隐藏所有非基础按钮接口

```
wx.hideAllNonBaseMenuItem();
```

显示所有功能按钮接口

```
wx.showAllNonBaseMenuItem();
```

微信扫一扫

调起微信扫一扫接口

```
wx.scanQRCode({
    desc: 'scanQRCode desc',
    needResult: 0, // 默认为0，扫描结果由微信处理，1则直接返回扫描结果，
    scanType: ["qrCode","barCode"], // 可以指定扫二维码还是一维码，默认二者都有
    success: function (res) {
    var result = res.resultStr; // 当needResult 为 1 时，扫码返回的结果
}
});
```

微信小店

跳转微信商品页接口

```
wx.openProductSpecificView({
    productId: '', // 商品id
    viewType: '' // 0.默认值，普通商品详情页1.扫一扫商品详情页2.小店商品详情页
});
```

微信卡券

调起适用于门店的卡券列表并获取用户选择列表

```
wx.chooseCard({
    shopId: '', // 门店Id
    cardType: '', // 卡券类型
    cardId: '', // 卡券Id
    timeStamp: 0, // 卡券签名时间戳
    nonceStr: '', // 卡券签名随机串
    cardSign: '', // 卡券签名，详见附录6
    success: function (res) {
        var cardList= res.cardList; // 用户选中的卡券列表信息
    }
});
```

批量添加卡券接口

```
wx.addCard({
    cardList: [{
        cardId: '',
        cardExt: ''
    }], // 需要添加的卡券列表
    success: function (res) {
        var cardList = res.cardList; // 添加的卡券列表信息
    }
});
```

查看微信卡包中的卡券接口

```
wx.openCard({
    cardList: [{
        cardId: '',
        code: ''
    }]// 需要打开的卡券列表
});
```

微信支付

发起一个微信支付请求

```
wx.chooseWXPay({
    timestamp: 0, // 支付签名时间戳
    noncestr: '', // 支付签名随机串
    package: '', // 订单详情扩展字符串，详见附录5
    paySign: '', // 支付签名，详见附录5
});
```

附录

附录1-JS-SDK使用权限签名算法

jsapi_ticket

生成签名之前必须先了解一下jsapi_ticket，jsapi_ticket是公众号用于调用微信JS接口的临时票据。正常情况下，jsapi_ticket的有效期为7200秒，通过access_token来获取。由于获取jsapi_ticket的api调用次数非常有限，频繁刷新jsapi_ticket会导致api调用受限，影响自身业务，开发者必须在自己的服务全局缓存jsapi_ticket 。

1. 参考以下文档获取access_token（有效期7200秒，开发者必须在自己的服务全局缓存access_token）：../15/54ce45d8d30b6bf6758f68d2e95bc627.html
2. 用第一步拿到的access_token 采用http GET方式请求获得jsapi_ticket（有效期7200秒，开发者必须在自己的服务全局缓存jsapi_ticket）：https://api.weixin.qq.com/cgi-bin/ticket/getticket?access_token=ACCESS_TOKEN&type=jsapi

成功返回如下JSON：

```
{
"errcode":0,
"errmsg":"ok",
"ticket":"bxLdikRXVbTPdHSM05e5u5sUoXNKd8-41ZO3MhKoyN5OfkWITDGgnr2fwJ0m9E8NYzWKVZvdVtaUgWvsdshFKA",
"expires_in":7200
}
```

获得jsapi_ticket之后，就可以生成JS-SDK权限验证的签名了。

签名算法

签名生成规则如下：参与签名的字段包括noncestr（随机字符串）, 有效的jsapi_ticket, timestamp（时间戳）, url（当前网页的URL，不包含#及其后面部分） 。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。这里需要注意的是所有参数名均为小写字符。对string1作sha1加密，字段名和字段值都采用原始值，不进行URL 转义。

即signature=sha1(string1)。 示例：

- noncestr=Wm3WZYTPz0wzccnW
- jsapi_ticket=sM4AOVdWfPE4DxkXGEs8VMCPGGVi4C3VM0P37wVUCFvkVAy_90u5h9nbSlYy3-Sl-HhTdfl2fzFy1AOcHKP7qg
- timestamp=1414587457
- url=http://mp.weixin.qq.com

步骤**1.** 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**：

```
jsapi_ticket=sM4AOVdWfPE4DxkXGEs8VMCPGGVi4C3VM0P37wVUCFvkVAy_90u5h9nbSlYy3-Sl-HhTdfl2fzFy1AOcHKP7qg&noncestr=Wm3WZYTPz0
```

步骤**2.** 对**string1**进行**sha1**签名，得到**signature**：

```
f4d90daf4b3bca3078ab155816175ba34c443a7b
```

注意事项

签名用的noncestr和timestamp必须与wx.config中的nonceStr和timestamp相同。 签名用的url必须是调用JS接口页面的完整URL。 出于安全考虑，开发者必须在服务器端实现签名的逻辑。

附录2-所有JS接口列表

版本1.0.0接口

- onMenuShareTimeline
- onMenuShareAppMessage
- onMenuShareQQ
- onMenuShareWeibo
- startRecord
- stopRecord
- onVoiceRecordEnd
- playVoice
- pauseVoice
- stopVoice
- onVoicePlayEnd
- uploadVoice
- downloadVoice
- chooseImage
- previewImage
- uploadImage
- downloadImage
- translateVoice
- getNetworkType
- openLocation
- getLocation
- hideOptionMenu
- showOptionMenu
- hideMenuItems
- showMenuItems
- hideAllNonBaseMenuItem
- showAllNonBaseMenuItem
- closeWindow
- scanQRCode
- chooseWXPay
- openProductSpecificView
- addCard
- chooseCard
- openCard

附录3-所有菜单项列表

基本类

- 举报: "menuItem:exposeArticle"
- 调整字体: "menuItem:setFont"
- 日间模式: "menuItem:dayMode"
- 夜间模式: "menuItem:nightMode"
- 刷新: "menuItem:refresh"
- 查看公众号（已添加）: "menuItem:profile"
- 查看公众号（未添加）: "menuItem:addContact"

传播类

- 发送给朋友: "menuItem:share:appMessage"
- 分享到朋友圈: "menuItem:share:timeline"
- 分享到QQ: "menuItem:share:qq"
- 分享到Weibo: "menuItem:share:weiboApp"
- 收藏: "menuItem:favorite"
- 分享到FB: "menuItem:share:facebook"

保护类

- 调试: "menuItem:jsDebug"
- 编辑标签: "menuItem:editTag"
- 删除: "menuItem:delete"
- 复制链接: "menuItem:copyUrl"
- 原网页: "menuItem:originPage"
- 阅读模式: "menuItem:readMode"
- 在QQ浏览器中打开: "menuItem:openWithQQBrowser"
- 在Safari中打开: "menuItem:openWithSafari"
- 邮件: "menuItem:share:email"
- 一些特殊公众号: "menuItem:share:brand"

附录4-位置签名生成算法

addrSign的生成规则与JS-SDK权限验证的签名生成规则相同（参考附录1），只是参与签名参数有所不同。参与addrSign的签名参数有：appId、url（当前网页url）、timestamp、noncestr、accesstoken（用户授权凭证，请参照oauth2.0 协议获取）。

附录5-支付扩展字段及签名生成算法

订单详情（package）扩展字符串定义

在商户调起JS API 时，商户需要此时确定该笔订单详情，并将该订单详情通过一定的方式进行组合放入package。JS API 调用后，微信将通过package 的内容生成预支付单。下 面将定义package 的所需字段列表以及签名方法。 接口需要注意：所有传入参数都是字符串类型！

package 所需字段列表:

参数	名称	是否必填	格式	说明
bank_type	银行通道类型	是	字符串类型，固定为"WX"，注意大写	固定为"WX"；
body	商品描述	是	字符串类型，128字节以下	商品描述；
attach	附加数据	否	字符串类型，128字节以下	附加数据，原样返回；
partner	商户号	是	字符串类型	字符串类型注册时分配的财付通商户号partnerId；
out_trade_no	商户订单号	是	字符串类型，32字节以下	商户系统内部的订单号，32 个字符内、可包含字母；确保在商户系统唯一
	订单			

total_fee	总金额	是	字符串类型	订单总金额，单位为分；
fee_type	支付币种	是	字符串类型，默认值是"1"	取值：1（人民币），暂只支持1；
notify_url	通知URL	是	字符串类型，255字节以下	在支付完成后，接收微信通知支付结果的URL，需给绝对路径， 255 字符内,格式如:http://wap.tenpay.com/tenpay.asp；
spbill_create_ip	订单生成的机器IP	是	字符串类型，15字节以下	指用户浏览器端IP，不是商户服务器IP，格式为IPV4；
time_start	交易起始时间	否	字符串类型，14字节以下	订单生成时间，格式为yyyyMMddHHmmss，如2009 年12 月25 日9 点10 分10 秒表示为20091225091010，时区为GMT+8 beijing；该时间取自商户服务器；
time_expire	交易结束时间	否	字符串类型，14字节以下	订单失效时间，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；该时间取自商户服务器；
transport_fee	物流费用	否	字符串类型	物流费用，单位为分。如果有值，必须保证 transport_fee + product_fee=total_fee；
product_fee	商品费用	否	字符串类型	物流费用，单位为分。如果有值，必须保证 transport_fee + product_fee=total_fee；
goods_tag	商品标记	否	字符串类型	商品标记，优惠券时可能用到；
input_charset	传入参数字符编码	是	字符串类型	取值范围："GBK"、"UTF-8"，默认："GBK"

package 生成方法： 由于package中携带了生成订单的详细信息，因此在微信将对package里面的内容进行鉴 权，确定package携带的信息是真实、有效、合理的。因此，这里将定义生成package字符 串的方法。

1. 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，注意：值为空的参数不参与签名；
2. 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。
3. 对传入参数中所有键值对的value进行urlencode转码后重新拼接成字符串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+）。
4. 将sign=signValue拼接到string2后面得到最终的package字符串。

下面定义了一段生成package字符串的示范过程： 假设以下为package传入参数：

- bank_type=WX
- body=支付测试
- fee_type=1
- input_charset=UTF-8
- notify_url=http://weixin.qq.com
- out_trade_no=7240b65810859cbf2a8d9f76a638c0a3
- partner=1900000109
- spbill_create_ip=196.168.1.1
- total_fee=1

i: 经过a过程URL键值对字典序排序后的字符串string1为：

```
bank_type=WX&body=支付测试&fee_type=1&input_charset=UTF-8&notify_url=http://we
ixin.qq.com&out_trade_no=7240b65810859cbf2a8d9f76a638c0a3&partner=1900000109&spbill_
create_ip=196.168.1.1&total_fee=1
```

ii：经过b过程后得到sign为：

```
sign
=md5(string1&key=8934e7d15453e97507ef794cf7b0519d).toUpperCase
=md5(bank_type=WX&body=支付测试&fee_type=1&input_charset=UTF-8&notify_url=htt
p://weixin.qq.com&out_trade_no=7240b65810859cbf2a8d9f76a638c0a3&partner=1900000109&
spbill_create_ip=196.168.1.1&total_fee=1&key=8934e7d15453e97507ef794cf7b0519d).toUpper
Case()
="7f77b507b755b3262884291517e380f8".toUpperCase()
="7F77B507B755B3262884291517E380F8"
```

iii：再对传入参数中的每一个键值对中的value进行urlencode编码后得到：

```
bank_type=WX&body=%E6%94%AF%E4%BB%98%E6%B5%8B%E8%AF%95&fee_typ
e=1&input_charset=UTF-8&notify_url=http%3A%2F%2Fweixin.qq.com&out_trade_no=7240b6
5810859cbf2a8d9f76a638c0a3&partner=1900000109&spbill_create_ip=196.168.1.1&total_fee=1
```

iv：拼接上sign后得到最终package结果：

```
bank_type=WX&body=%E6%94%AF%E4%BB%98%E6%B5%8B%E8%AF%95&fee_typ
e=1&input_charset=UTF-8&notify_url=http%3A%2F%2Fweixin.qq.com&out_trade_no=7240b6
5810859cbf2a8d9f76a638c0a3&partner=1900000109&spbill_create_ip=196.168.1.1&total_fee=1
&sign=7F77B507B755B3262884291517E380F8
```

支付签名（**paySign**）生成方法

paySign字段是对本次发起JSAPI的行为进行鉴权，只有通过了paySign鉴权，才能继续对package鉴权并生成预支付单。paySign的生成规则与JS-SDK权限验证的签名生成规则相同（参考附录1）。

参与paySign签名的字段包括：appid、timestamp、noncestr、package以及appkey（即 paySignkey）。

附录**6-**卡券扩展字段及签名生成算法

卡券扩展字段cardExt说明

cardExt本身是一个JSON字符串，是商户为该张卡券分配的唯一性信息，包含以下字段：

字段	是否必填	说明
code	否	指定的卡券code码，只能被领一次。use_custom_code字段为true的卡券必须填写，非自定义code不必填写。
openid	否	指定领取者的openid，只有该用户能领取。bind_openid字段为true的卡券必须填写，非自定义openid不必填写。
timestamp	是	时间戳，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,且最终需要转换为字符串形式;由商户生成后传入。
signature	是	签名，商户将接口列表中的参数按照指定方式进行签名,签名方式使用SHA1,具体签名方案参见下文;由商户按照规范签名后传入。
balance	否	红包余额，以分为单位。红包类型必填（LUCKY_MONEY），其他卡券类型不填。

签名说明

1. 将appsecret（第三方用户唯一凭证密）、timestamp、card_id、code、openid、balance的value值进行字符串的字典序排序。
2. 将所有参数字符串拼接成一个字符串进行sha1加密，得到signature。
3. signature中的timestamp和card_ext中的timestamp必须保持一致。
4. 假如数据示例中code=23456，timestamp=141231233，card_id=345667，appsecret=45678则

signature=sha1(1412312332345634566745678 9)=4F76593A4245644FAE4E1BC940F6422A0C3EC03E。

卡券签名cardSign说明

1. 将appsecret、app_id、location_id、times_tamp、nonce_str、card_id、card_type的value值进行字符串的字典序排序。
2. 将所有参数字符串拼接成一个字符串进行sha1加密，得到cardSign。

附录7-常见错误及解决方法

调用config 接口的时候传入参数 debug: true 可以开启debug模式，页面会alert出错误信息。以下为常见错误及解决方法：

1. invalid url domain当前页面所在域名与使用的appid没有绑定（一个appid可以绑定三个有效域名）。
2. invalid signature签名错误。建议按如下顺序检查：
   i. 确认签名算法正确，可用 http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=jsapisign 页面工具进行校验。
   ii. 确认config中noncestr, timestamp与用以签名中的对应noncestr, timestamp一致。
   iii. 确认url是页面完整的url，包括GET参数部分。
   iv. 确认 config 中的 appid 与用来获取 jsapi_ticket 的 appid 一致。
3. the permission value is offline verifying这个错误是因为config没有正确执行，或者是调用的JSAPI没有传入config的jsApiList参数中。建议按如下顺序检查：
   i. 确认config正确通过。
   ii. 如果是在页面加载好时就调用了JSAPI，则必须写在wx.ready的回调中。
   iii. 确认config的jsApiList参数包含了这个JSAPI。
4. permission denied该应用没有权限使用这个JSAPI。

附录8-DEMO页面和示例代码

DEMO页面：

http://demo.open.weixin.qq.com/jssdk

示例代码：

http://demo.open.weixin.qq.com/jssdk/sample.zip

备注：链接中包含`php`、`java`、`nodejs`以及python的示例代码供第三方参考，第三方切记要对获取的`accesstoken`以及`jsapi_ticket`进行缓存以确保不会触发频率限制。

附录9-问题反馈

邮箱地址：weixin-open@qq.com

邮件主题：【微信JS-SDK反馈】

邮件内容说明：

用简明的语言描述问题所在，并交代清楚遇到该问题的场景，可附上截屏图片，微信团队会尽快处理你的反馈。

JS-SDK常见错误

> 随官方持续更新中

调用config 接口的时候传入参数 debug: true 可以开启debug模式，页面会alert出错误信息。以下为常见错误及解决方法：

1. invalid url domain当前页面所在域名与使用的appid没有绑定（一个appid可以绑定三个有效域名，见 目录1.1.1）。
2. invalid signature签名错误。建议按如下顺序检查：
   i. 确认签名算法正确，可用 http://mp.weixin.qq.com/debug/cgi-bin/sandbox?t=jsapisign 页面工具进行校验。
   ii. 确认config中noncestr, timestamp与用以签名中的对应noncestr, timestamp一致。
   iii. 确认url是页面完整的url，包括GET参数部分。
   iv. 确认 config 中的 appid 与用来获取 jsapi_ticket 的 appid 一致。
   v. 确保一定缓存access_token和jsapi_ticket，可以减少两次服务器请求加速体验外，还避免了触发频率限制，提高服务稳定性。
3. the permission value is offline verifying这个错误是因为config没有正确执行，或者是调用的JSAPI没有传入config的jsApiList参数中。建议按如下顺序检查：
   i. 确认config正确通过。
   ii. 如果是在页面加载好时就调用了JSAPI，则必须写在wx.ready的回调中。
   iii. 确认config的jsApiList参数包含了这个JSAPI。
4. permission denied该公众号没有权限使用这个JSAPI（部分接口需要认证之后才能使用）。
5. Android用户已取消分享朋友圈，但仍返回分享成功（微信团队已修复此问题，会在Android6.1版本上线）
6. **Android6.0.2**部分客户端无法使用监听分享接口（Android6.0.2之前以及6.0.2.58以后的版本都不会有问题，请从官网下载最新版本体验）
7. 服务上线之后无法获取jsapi_ticket，自己测试时没问题。因为access_token和jsapi_ticket必须要在自己的服务器缓存，否则上线后会触发频率限制。请确保一定对token和ticket做缓存以减少2次服务器请求，不仅可以避免触发频率限制，还加快你们自己的服务速度。目前为了方便测试提供了1w的获取量，超过阀值后，服务将不再可用，请确保在服务上线前一定缓存access_token和jsapi_ticket。
8. Android部分版本上传图片接口偶尔卡住（早期的Android6.0.2版本存在此问题，官方已修复，6.0.2.58之后的版本都支持，请从官网下载最新版本体验）
9. 下载多媒体相关jsapi只能下载通过jsapi上传的资源，无法下载通过多媒体后台接口上传的资源（微信团队已经确认该问题，正在努力修复中，预计本周修复上线）

Glossary