

# Table of Contents

## Part 1

前言	1.1
----	-----

## Part 2

入门	2.1
简易教程	2.1.1
体验小程序	2.1.2

## Part 3

框架	3.1
目录结构	3.1.1
配置	3.1.2
逻辑层	3.1.3
注册程序	3.1.3.1
注册页面	3.1.3.2
模块化	3.1.3.3
API	3.1.3.4
视图层	3.1.4
WXML	3.1.4.1
数据绑定	3.1.4.1.1
条件渲染	3.1.4.1.2
列表渲染	3.1.4.1.3
模板	3.1.4.1.4
事件	3.1.4.1.5
引用	3.1.4.1.6
WXSS	3.1.4.2
组件	3.1.4.3

## Part 4

组件	4.1
视图容器	4.1.1
view	4.1.1.1
scroll-view	4.1.1.2
swiper	4.1.1.3

基础内容	4.1.2
icon	4.1.2.1
text	4.1.2.2
progress	4.1.2.3
表单组件	4.1.3
button	4.1.3.1
checkbox	4.1.3.2
form	4.1.3.3
input	4.1.3.4
label	4.1.3.5
picker	4.1.3.6
radio	4.1.3.7
slider	4.1.3.8
switch	4.1.3.9
操作反馈	4.1.4
action-sheet	4.1.4.1
modal	4.1.4.2
toast	4.1.4.3
loading	4.1.4.4
导航	4.1.5
navigator	4.1.5.1
媒体组件	4.1.6
audio	4.1.6.1
image	4.1.6.2
video	4.1.6.3
地图	4.1.7
map	4.1.7.1
画布	4.1.8
canvas	4.1.8.1

## Part 5

API	5.1
网络	5.1.1
发起请求	5.1.1.1
上传、下载	5.1.1.2
wx.uploadFile	5.1.1.2.1
wx.downloadFile	5.1.1.2.2
WebSocket	5.1.1.3
wx.connectSocket	5.1.1.3.1
wx.onSocketOpen	5.1.1.3.2

<a href="#">wx.onSocketError</a>	5.1.1.3.3
<a href="#">wx.sendSocketMessage</a>	5.1.1.3.4
<a href="#">wx.onSocketMessage</a>	5.1.1.3.5
<a href="#">wx.closeSocket</a>	5.1.1.3.6
<a href="#">wx.onSocketClose</a>	5.1.1.3.7
<b>媒体</b>	5.1.2
<b>图片</b>	5.1.2.1
<a href="#">wx.chooseImage</a>	5.1.2.1.1
<a href="#">wx.previewImage</a>	5.1.2.1.2
<b>录音</b>	5.1.2.2
<a href="#">wx.startRecord</a>	5.1.2.2.1
<a href="#">wx.stopRecord</a>	5.1.2.2.2
<b>音频播放控制</b>	5.1.2.3
<a href="#">wx.playVoice</a>	5.1.2.3.1
<a href="#">wx.pauseVoice</a>	5.1.2.3.2
<a href="#">wx.stopVoice</a>	5.1.2.3.3
<b>音乐播放控制</b>	5.1.2.4
<a href="#">wx.getBackgroundAudioPlayerState</a>	5.1.2.4.1
<a href="#">wx.playBackgroundAudio</a>	5.1.2.4.2
<a href="#">wx.pauseBackgroundAudio</a>	5.1.2.4.3
<a href="#">wx.seekBackgroundAudio</a>	5.1.2.4.4
<a href="#">wx.stopBackgroundAudio</a>	5.1.2.4.5
<a href="#">wx.onBackgroundAudioPlay</a>	5.1.2.4.6
<a href="#">wx.onBackgroundAudioPause</a>	5.1.2.4.7
<a href="#">wx.onBackgroundAudioStop</a>	5.1.2.4.8
<b>文件</b>	5.1.2.5
<b>视频</b>	5.1.2.6
<b>数据</b>	5.1.3
<b>数据缓存</b>	5.1.3.1
<a href="#">wx.setStorage</a>	5.1.3.1.1
<a href="#">wx.setStorageSync</a>	5.1.3.1.2
<a href="#">wx.getStorage</a>	5.1.3.1.3
<a href="#">wx.getStorageSync</a>	5.1.3.1.4
<a href="#">wx.clearStorage</a>	5.1.3.1.5
<a href="#">wx.clearStorageSync</a>	5.1.3.1.6
<b>位置</b>	5.1.4
<b>获取位置</b>	5.1.4.1
<b>查看位置</b>	5.1.4.2
<b>设备</b>	5.1.5
<b>网络状态</b>	5.1.5.1

系统信息	5.1.5.2
重力感应	5.1.5.3
罗盘	5.1.5.4
界面	5.1.6
设置导航条	5.1.6.1
导航	5.1.6.2
动画	5.1.6.3
绘图	5.1.6.4
其他	5.1.6.5
开放接口	5.1.7
登录	5.1.7.1
签名加密	5.1.7.1.1
用户信息	5.1.7.2
微信支付	5.1.7.3
模板消息	5.1.7.4
使用说明	5.1.7.4.1
接口说明	5.1.7.4.2

## Part 6

工具	6.1
概览	6.1.1
程序调试	6.1.2
模拟器	6.1.2.1
调试工具	6.1.2.2
Wxml Panel	6.1.2.2.1
Sources Panel	6.1.2.2.2
Network Panel	6.1.2.2.3
Appdata Panel	6.1.2.2.4
Storage Panel	6.1.2.2.5
Console Panel	6.1.2.2.6
小程序操作区	6.1.2.3
代码编辑	6.1.3
项目预览	6.1.4
下载	6.1.5

## Part 7

Q&A	7.1
Q&A	7.1.1
联系我们	7.1.2

# 前言

## 微信小程序官方使用手册-**20161015**

本手册为根据微信官方版本内容整理制作的离线版手册，为外网不畅用户所准备。

本手册最初制作于2016年10月15日，此时微信小程序尚未完成内测，其官方文档也在持续更新中。本手册在编辑过程中修复了部分官方文档中的错误。本手册虽力求同步于微信公众平台官方文档的更新，但仍不代表官方文档，如有疑问请遵照官方内容。

手册完整在线版本地址：<https://wecharts.github.io/tinyapp-doc/>

微信官方手册地址（内测）：<https://mp.weixin.qq.com/debug/wxadoc/dev/>

离线手册的持续更新地址为：<http://www.xcxl.com>

微信小程序技术交流QQ群：416592844

感谢小程序联盟的支持！

本文档将带你一步步创建完成一个微信小程序，并可以在手机上体验该小程序的实际效果。这个小程序的首页将会显示欢迎语以及当前用户的微信头像，点击头像，可以在新开的页面中查看当前小程序的启动日志。[下载源码](#)

## 1. 获取微信小程序的 AppID

如果你是收到邀请的开发者，我们会提供一个帐号，利用提供的帐号，登录 <https://mp.weixin.qq.com>，就可以在网站的“设置”-“开发者设置”中，查看到微信小程序的 AppID 了，注意不可直接使用服务号或订阅号的 AppID。

如果没有收到内测邀请，可以跳过本步骤



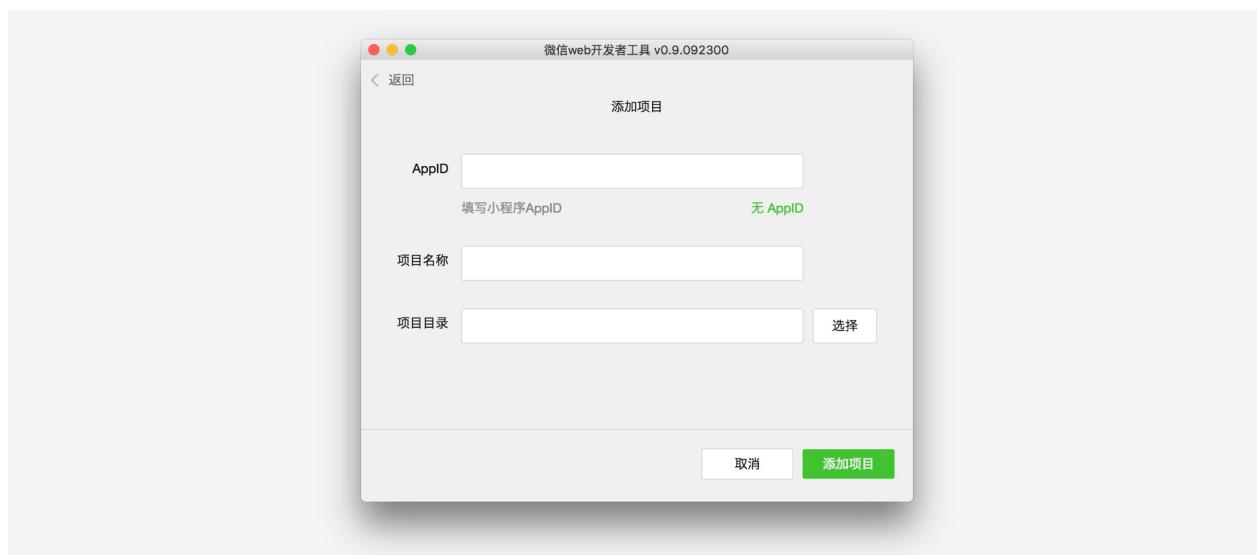
注意：如果要以非管理员微信号在手机上体验该小程序，那么我们还需要操作“绑定开发者”。即在“用户身份”-“开发者”模块，绑定上需要体验该小程序的微信号。本教程默认注册帐号、体验都是使用管理员微信号。

## 2. 创建项目

我们需要通过[开发者工具](#)，来完成小程序创建和代码编辑。

开发者工具安装完成后，打开并使用微信扫码登录。选择创建“项目”，填入上文获取到的 AppID，设置一个本地项目的名称（非小程序名称），比如“我的第一个项目”，并选择一个本地的文件夹作为代码存储的目录，点击“新建项目”就可以了。

为方便初学者了解微信小程序的基本代码结构，在创建过程中，如果选择的本地文件夹是个空文件夹，开发者工具会提示，是否需要创建一个 quick start 项目。选择“是”，开发者工具会帮助我们在开发目录里生成一个简单的 demo。



项目创建成功后，我们就可以点击该项目，进入并看到完整的开发者工具界面，点击左侧导航，在“编辑”里可以查看和编辑我们的代码，在“调试”里可以测试代码并模拟小程序在微信客户端效果，在“项目”里可以发送到手机里预览实际效果。

### 3. 编写代码

#### 创建小程序实例

点击开发者工具左侧导航的“编辑”，我们可以看到这个项目，已经初始化并包含了一些简单的代码文件。最关键也是必不可少的，是 `app.js`、`app.json`、`app.wxss` 这三个。其中，`.js` 后缀的是脚本文件，`.json` 后缀的文件是配置文件，`.wxss` 后缀的是样式表文件。微信小程序会读取这些文件，并生成[小程序实例](#)。

下面我们简单了解这三个文件的功能，方便修改以及从头开发自己的微信小程序。

`app.js` 是小程序的脚本代码。我们可以在这个文件中监听并处理小程序的生命周期函数、声明全局变量。调用框架提供的丰富的 API，如本例的同步存储及同步读取本地数据。想了解更多可用 API，可参考 [API 文档](#)

```
//app.js
App({
  onLaunch: function () {
    //调用API从本地缓存中获取数据
    var logs = wx.getStorageSync('logs') || []
    logs.unshift(Date.now())
    wx.setStorageSync('logs', logs)
  },
  getUserInfo:function(cb){
    var that = this;
    if(this.globalData.userInfo){
      typeof cb == "function" && cb(this.globalData.userInfo)
    }else{
      //调用登录接口
      wx.login({
        success: function () {
          wx.getUserInfo({
            success: function (res) {
              that.globalData.userInfo = res.userInfo;
              typeof cb == "function" && cb(that.globalData.userInfo)
            }
          })
        }
      });
    }
  },
  globalData:{
    userInfo:null
  }
})
```

`app.json` 是对整个小程序的全局配置。我们可以在这个文件中配置小程序是由哪些页面组成，配置小程序的窗口背景色，配置导航条样式，配置默认标题。注意该文件不可添加任何注释。更多可配置项可参考[配置详解](#)

```
{  
  "pages": [  
    "pages/index/index",  
    "pages/logs/logs"  
  ],  
  "window": {  
    "backgroundTextStyle": "light",  
    "navigationBarBackgroundColor": "#fff",  
    "navigationBarTitleText": "WeChat",  
    "navigationBarTextStyle": "black"  
  }  
}
```

**app.wxss** 是整个小程序的公共样式表。我们可以在页面组件的 **class** 属性上直接使用 **app.wxss** 中声明的样式规则。

```
/**app.wxss**/  
.container {  
  height: 100%;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  justify-content: space-between;  
  padding: 200rpx 0;  
  box-sizing: border-box;  
}
```

## 创建页面

在这个教程里，我们有两个页面，**index** 页面和 **logs** 页面，即欢迎页和小程序启动日志的展示页，它们都在 **pages** 目录下。微信小程序中的每一个页面的【路径+页面名】都需要写在 **app.json** 的 **pages** 中，且 **pages** 中的第一个页面是小程序的首页。

每一个[小程序页面](#)是由同路径下同名的四个不同后缀文件的组成，如：**index.js**、**index.wxml**、**index.wxss**、**index.json**。**.js** 后缀的文件是脚本文件，**.json** 后缀的文件是配置文件，**.wxss** 后缀的是样式表文件，**.wxml** 后缀的文件是页面结构文件。

**index.wxml** 是页面的结构文件：

```
<!--index.wxml-->  
<view class="container">  
  <view bindtap="bindViewTap" class="userinfo">  
    <image class="userinfo-avatar" src="{{userInfo.avatarUrl}}" background-size="cover"></image>  
    <text class="userinfo-nickname">{{userInfo.nickName}}</text>  
  </view>  
  <view class="usermotto">  
    <text class="user-motto">{{motto}}</text>  
  </view>  
</view>
```

本例中使用了 **<view/>**、**<image/>**、**<text/>** 来搭建页面结构，绑定数据和交互处理函数。

**index.js** 是页面的脚本文件，在这个文件中我们可以监听并处理页面的生命周期函数、获取小程序实例，声明并处理数据，响应页面交互事件等。

```
//index.js
//获取应用实例
var app = getApp()
Page({
  data: {
    motto: 'Hello World',
    userInfo: {}
  },
  //事件处理函数
  bindViewTap: function() {
    wx.navigateTo({
      url: '../logs/logs'
    })
  },
  onLoad: function () {
    console.log('onLoad')
    var that = this
    //调用应用实例的方法获取全局数据
    app.getUserInfo(function(userInfo){
      //更新数据
      that.setData({
        userInfo:userInfo
      })
    })
  }
})
```

**index.wxss** 是页面的样式表：

```
/**index.wxss*/
.userInfo {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.userInfo-avatar {
  width: 128rpx;
  height: 128rpx;
  margin: 20rpx;
  border-radius: 50%;
}

.userInfo-nickname {
  color: #aaa;
}

.usermotto {
  margin-top: 200px;
}
```

页面的样式表是非必要的。当有页面样式表时，页面的样式表中的样式规则会层叠覆盖 **app.wxss** 中的样式规则。如果不指定页面的样式表，也可以在页面的结构文件中直接使用 **app.wxss** 中指定的样式规则。

**index.json** 是页面的配置文件：

页面的配置文件是非必要的。当有页面的配置文件时，配置项在该页面会覆盖 **app.json** 的 **window** 中相同的配置项。如果没有指定的页面配置文件，则在该页面直接使用 **app.json** 中的默认配置。

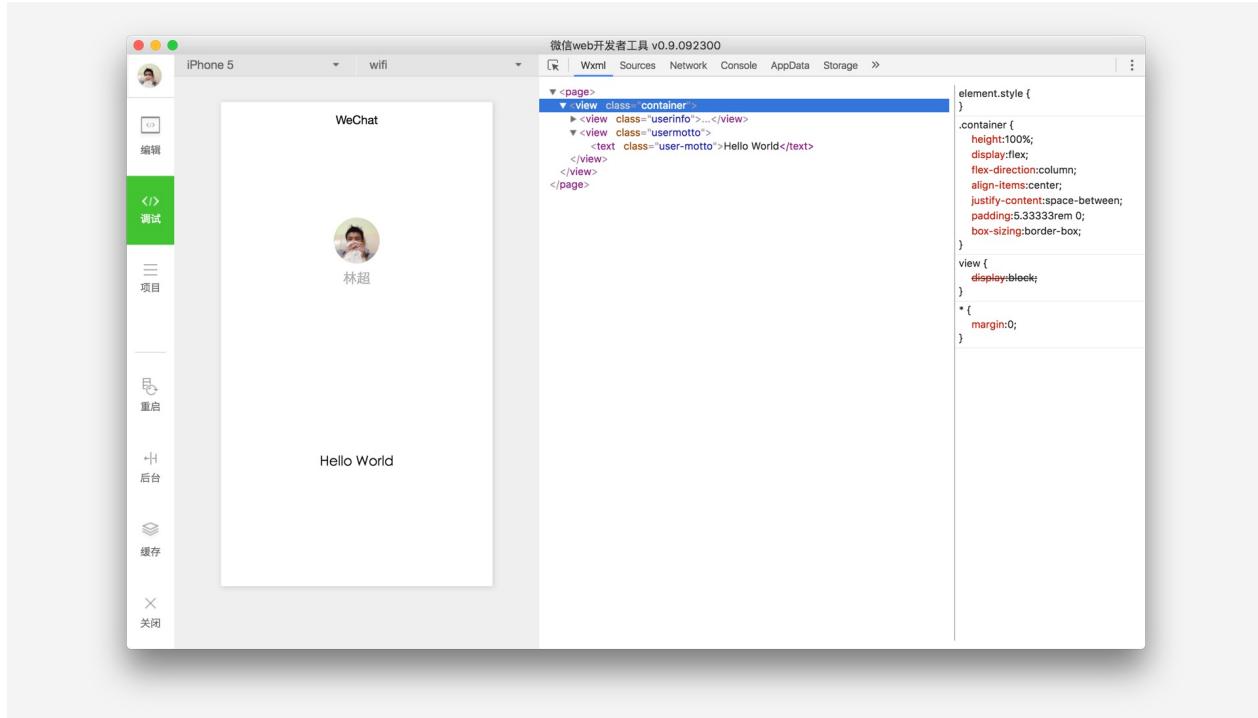
**logs** 的页面结构

```
<!--logs.wxml-->
<view class="container log-list">
  <block wx:for="{{logs}}" wx:for-item="log">
    <text class="log-item">{{index + 1}}. {{log}}</text>
  </block>
</view>
```

logs 页面使用 `<block/>` 控制标签来组织代码，在 `<block/>` 上使用 `wx:for` 绑定 logs 数据，并将 logs 数据循环展开节点

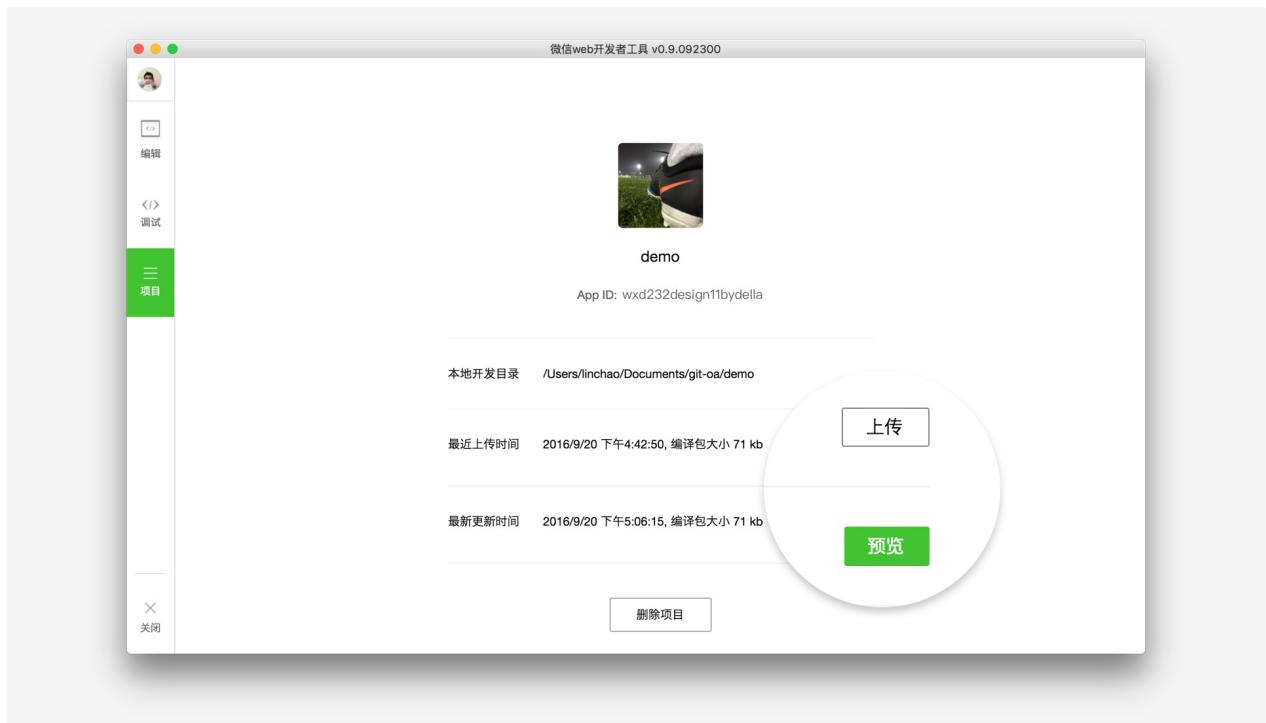
```
//logs.js
var util = require('../utils/util.js')
Page({
  data: {
    logs: []
  },
  onLoad: function () {
    this.setData({
      logs: (wx.getStorageSync('logs') || []).map(function (log) {
        return util.formatTime(new Date(log))
      })
    })
  }
})
```

运行结果如下：



## 4. 手机预览

开发者工具左侧菜单栏选择“项目”，点击“预览”，扫码后即可在微信客户端中体验。



下载微信客户端版本号：6.3.27 及以上，只有小程序绑定的开发者有权限扫码体验。[下载源码 版本20161010](#)





# 小程序组件

这是展示小程序组件的DEMO。

视图容器



基础内容



表单组件



操作反馈



导航



媒体组件



地图



组件



接口

# 框架

小程序开发框架的目标是通过尽可能简单、高效的方式让开发者可以在微信中开发具有原生 APP 体验的服务。

框架提供了自己的视图层描述语言 **WXML** 和 **WXSS**，以及基于 **JavaScript** 的逻辑层框架，并在视图层与逻辑层间提供了数据传输和事件系统，可以让开发者可以方便的聚焦于数据与逻辑上。

## 响应的数据绑定

框架的核心是一个响应的数据绑定系统。

整个系统分为两块视图层（**View**）和逻辑层（**App Service**）

框架可以让数据与视图非常简单地保持同步。当做数据修改的时候，只需要在逻辑层修改数据，视图层就会做相应的更新。

通过这个简单的例子来看：

```
<!-- This is our View -->
<view> Hello {{name}}! </view>
<button bindtap="changeName"> Click me! </button>

// This is our App Service.
// This is our data.
var helloData = {
  name: 'WeChat'
}

// Register a Page.
Page({
  data: helloData,
  changeName: function(e) {
    // sent data change to view
    this.setData({
      name: 'MINA'
    })
}
})
```

- 开发者通过框架将逻辑层数据中的 `name` 与视图层的 `name` 进行了绑定，所以在页面一打开的时候会显示 `Hello WeChat!`
- 当点击按钮的时候，视图层会发送 `changeName` 的事件给逻辑层，逻辑层找到对应的事件处理函数
- 逻辑层执行了 `setData` 的操作，将 `name` 从 `WeChat` 变为 `MINA`，因为该数据和视图层已经绑定了，从而视图层会自动改变为 `Hello MINA!`。

## 页面管理

框架管理了整个小程序的页面路由，可以做到页面间的无缝切换，并给以页面完整的生命周期。开发者需要做的只是将页面的数据，方法，生命周期函数注册进框架中，其他的一切复杂的操作都交由框架处理。

## 基础组件

框架提供了一套基础的组件，这些组件自带微信风格的样式以及特殊的逻辑，开发者可以通过组合基础组件，创建出强大的微信小程序。

## 丰富的 API

框架 提供丰富的微信原生 API，可以方便的调起微信提供的能力，如获取用户信息，本地存储，支付功能等。

# 文件结构

框架程序包含一个描述整体程序的 `app` 和多个描述各自页面的 `page`。

一个框架程序主体部分由三个文件组成，必须放在项目的根目录，如下：

文件	必填	作用
<code>app.js</code>	是	小程序逻辑
<code>app.json</code>	是	小程序公共设置
<code>app.wxss</code>	否	小程序公共样式表

一个框架页面由四个文件组成，分别是：

文件类型	必填	作用
<code>js</code>	是	页面逻辑
<code>wxml</code>	是	页面结构
<code>wxss</code>	否	页面样式表
<code>json</code>	否	页面配置

注意：为了方便开发者减少配置项，我们规定描述页面的这四个文件必须具有相同的路径与文件名。

# 配置

我们使用 `app.json` 文件来对微信小程序进行全局配置，决定页面文件的路径、窗口表现、设置网络超时时间、设置多 `tab` 等。

以下是一个包含了所有配置选项的简单配置 `app.json`：

```
{  
  "pages": [  
    "pages/index/index",  
    "pages/logs/index"  
  ],  
  "window": {  
    "navigationBarTitleText": "Demo"  
  },  
  "tabBar": {  
    "list": [{  
      "pagePath": "pages/index/index",  
      "text": "首页"  
    }, {  
      "pagePath": "pages/logs/logs",  
      "text": "日志"  
    }]  
  },  
  "networkTimeout": {  
    "request": 10000,  
    "downloadFile": 10000  
  },  
  "debug": true  
}
```

## app.json 配置项列表

属性	类型	必填	描述
<code>pages</code>	Array	是	设置页面路径
<code>window</code>	Object	否	设置默认页面的窗口表现
<code>tabBar</code>	Object	否	设置底部 <code>tab</code> 的表现
<code>networkTimeout</code>	Object	否	设置网络超时时间
<code>debug</code>	Boolean	否	设置是否开启 <code>debug</code> 模式

## pages

接受一个数组，每一项都是字符串，来指定小程序由哪些页面组成。每一项代表对应页面的【路径+文件名】信息，数组的第一项代表小程序的初始页面。小程序中新增/减少页面，都需要对 `pages` 数组进行修改。

文件名不需要写文件后缀，因为框架会自动去寻找路径 `.json`，`.js`，`.wxml`，`.wxss` 的四个文件进行整合。

如开发目录为：

```
pages/  
  pages/index/index.wxml  
  pages/index/index.js  
  pages/index/index.wxss  
  pages/logs/logs.wxml  
  pages/logs/logs.js  
app.js  
app.json  
app.wxss
```

则，我们需要在 `app.json` 中写

```
{  
  "pages": [  
    "pages/index/index"  
    "pages/logs/logs"  
  ]  
}
```

## window

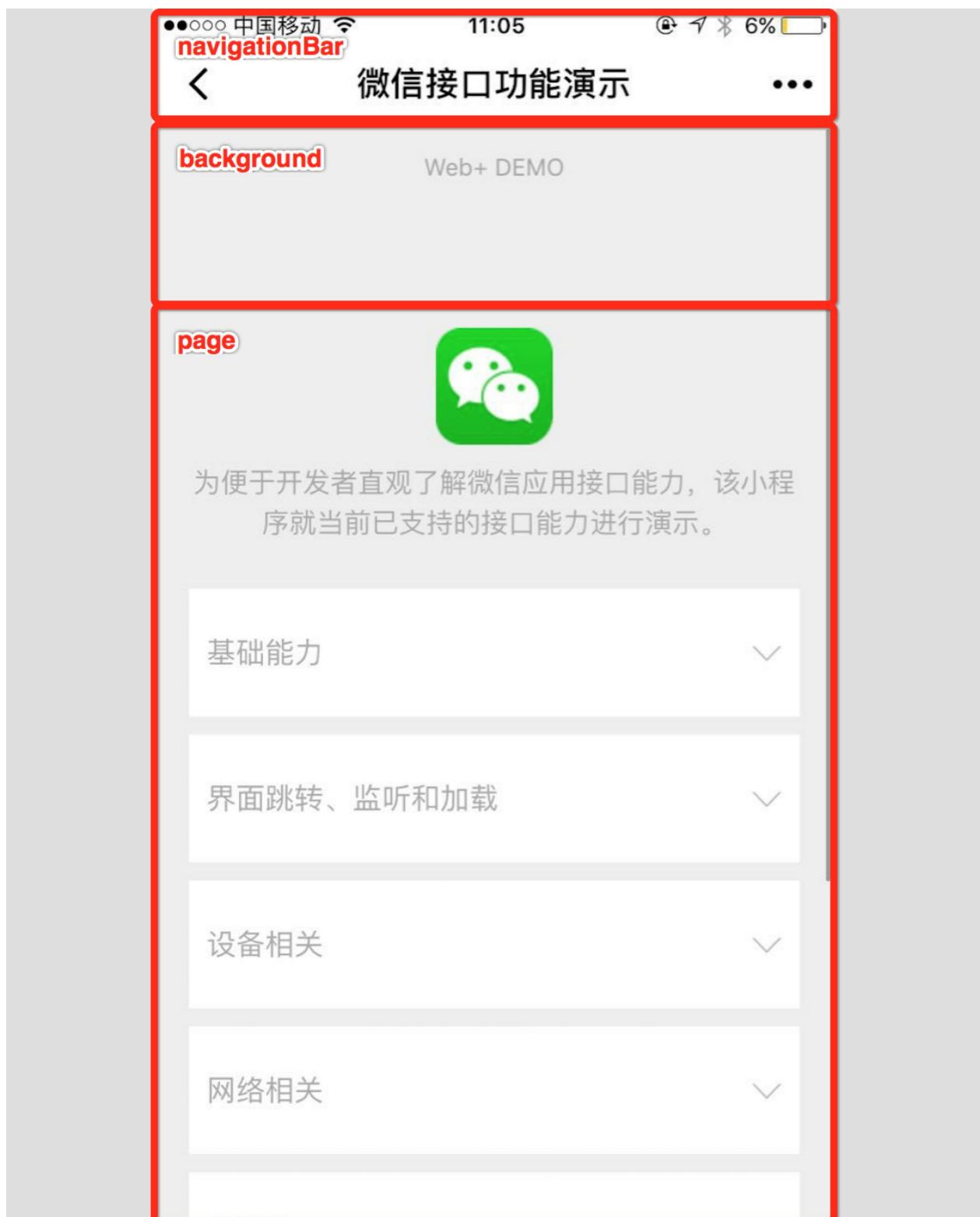
用于设置小程序的状态栏、导航条、标题、窗口背景色。

属性	类型	默认值	描述
<code>navigationBarBackgroundColor</code>	<code>HexColor</code>	<code>#000000</code>	导航栏背景颜色，如 <code>"#000000"</code>
<code>navigationBarTextStyle</code>	<code>String</code>	<code>white</code>	导航栏标题颜色，仅支持 <code>black/white</code>
<code>navigationBarTitleText</code>	<code>String</code>		导航栏标题文字内容
<code>backgroundColor</code>	<code>HexColor</code>	<code>#ffffff</code>	窗口的背景色
<code>backgroundTextStyle</code>	<code>String</code>	<code>dark</code>	下拉背景字体、 <code>loading</code> 图的样式，仅支持 <code>dark/light</code>
<code>enablePullDownRefresh</code>	<code>Boolean</code>	<code>false</code>	是否开启下拉刷新，详见 <a href="#">页面相关事件处理函数</a> 。

注：**HexColor**（十六进制颜色值），如`"#ff00ff"`

如 `app.json`：

```
{  
  "window": {  
    "navigationBarBackgroundColor": "#ffffff",  
    "navigationBarTextStyle": "black",  
    "navigationBarTitleText": "微信接口功能演示",  
    "backgroundColor": "#eeeeee",  
    "backgroundTextStyle": "light"  
  }  
}
```



## tabBar

如果我们的小程序是一个多 tab 应用（客户端窗口的底部有 tabBar 标签可以切换页面），那么我们可以通过 tabBar 配置项指定 tabBar 标的表现，以及 tab 切换时显示的对应页面。

tabBar 是一个数组，只能配置最少 2 个、最多 5 个 tab，tab 按数组的顺序排序。

属性说明：

属性	类型	必填	默认值	描述
color	HexColor	是		tab 上的文字默认颜色
selectedColor	HexColor	是		tab 上的文字选中时的颜色
backgroundColor	HexColor	是		tab 的背景色
borderStyle	String	否	black	tabbar上边框的颜色, 仅支持 black/white
list	Array	是		tab 的列表, 详见 list 属性说明, 最少2个、最多5个 tab

其中 list 接受一个数组, 数组中的每个项都是一个对象, 其属性值如下:

属性	类型	必填	说明
pagePath	String	是	页面路径, 必须在 pages 中先定义
text	String	是	tab 上按钮文字
iconPath	String	是	图片路径, icon 大小限制为40kb
selectedIconPath	String	是	选中时的图片路径, icon 大小限制为40kb



## networkTimeout

可以设置各种网络请求的超时时间。

属性说明:

属性	类型	必填	说明
request	Number	否	wx.request的超时时间，单位毫秒
connectSocket	Number	否	wx.connectSocket的超时时间，单位毫秒
uploadFile	Number	否	wx.uploadFile的超时时间，单位毫秒
downloadFile	Number	否	wx.downloadFile的超时时间，单位毫秒

## debug

可以在开发者工具中开启 `debug` 模式，在开发者工具的控制台面板，调试信息以 `info` 的形式给出，其信息有 `Page` 的注册， 页面路由， 数据更新， 事件触发。可以帮助开发者快速定位一些常见的问题。

## page.json

每一个小程序页面也可以使用 `.json` 文件来对本页面的窗口表现进行配置。页面的配置比 `app.json` 全局配置简单得多，只是设置 `app.json` 中的 `window` 配置项的内容，页面中配置项会覆盖 `app.json` 的 `window` 中相同的配置项。

页面的 `.json` 只能设置 `window` 相关的配置项，以决定本页面的窗口表现，所以无需写 `window` 这个键，如：

```
{
  "navigationBarBackgroundColor": "#ffffff",
  "navigationBarTextStyle": "black",
  "navigationBarTitleText": "微信接口功能演示",
  "backgroundColor": "#eeeeee",
  "backgroundTextStyle": "light"
}
```

# 逻辑层(**App Service**)

小程序开发框架的逻辑层是由**JavaScript**编写。

逻辑层将数据进行处理后发送给视图层，同时接受视图层的事件反馈。在 **JavaScript** 的基础上，我们做了一些修改，以方便地开发小程序。

- 增加 [App](#) 和 [Page](#) 方法，进行程序和页面的注册。
- 提供丰富的 [API](#)，如扫一扫，支付等微信特有能力。
- 每个页面有独立的[作用域](#)，并提供[模块化](#)能力。
- 由于框架并非运行在浏览器中，所以 **JavaScript** 在 **web** 中一些能力都无法使用，如 `document`, `window` 等。
- 开发者写的所有代码最终将会打包成一份 **JavaScript**，并在小程序启动的时候运行，直到小程序销毁。类似 [ServiceWorker](#)，所以逻辑层也称之为 **App Service**。

# App

## App()

`App()` 函数用来注册一个小程序。接受一个 `object` 参数，其指定小程序的生命周期函数等。

**object**参数说明：

属性	类型	描述	触发时机
<code>onLaunch</code>	<code>Function</code>	生命周期函数--监听小程序初始化	当小程序初始化完成时，会触发 <code>onLaunch</code> （全局只触发一次）
<code>onShow</code>	<code>Function</code>	生命周期函数--监听小程序显示	当小程序启动，或从后台进入前台显示，会触发 <code>onShow</code>
<code>onHide</code>	<code>Function</code>	生命周期函数--监听小程序隐藏	当小程序从前台进入后台，会触发 <code>onHide</code>
其他	<code>Any</code>	开发者可以添加任意的函数或数据到 <code>Object</code> 参数中，用 <code>this</code> 可以访问	

前台、后台定义：当用户点击左上角关闭，或者按了设备 `Home` 键离开微信，小程序并没有直接销毁，而是进入了后台；当再次进入微信或再次打开小程序，又会从后台进入前台。

只有当小程序进入后台一定时间，或者系统资源占用过高，才会被真正的销毁。

示例代码：

```
App({
  onLaunch: function() {
    // Do something initial when launch.
  },
  onShow: function() {
    // Do something when show.
  },
  onHide: function() {
    // Do something when hide.
  },
  globalData: 'I am global data'
})
```

## App.prototype.getCurrentPage()

`getCurrentPage()` 函数用户获取当前[页面](#)的实例。

## getApp()

我们提供了全局的 `getApp()` 函数，可以获取到小程序实例。

```
// other.js
var appInstance = getApp()
console.log(appInstance.globalData) // I am global data
```

注意：

`App()` 必须在 `app.js` 中注册，且不能注册多个。

不要在定义于 `App()` 内的函数中调用 `getApp()`，使用 `this` 就可以拿到 `app` 实例。

不要在 `onLaunch` 的时候调用 `getCurrentPage()`，此时 `page` 还没有生成。

通过 `getApp()` 获取实例之后，不要私自调用生命周期函数。

# Page

`Page()` 函数用来注册一个页面。接受一个 `object` 参数，其指定页面的初始数据、生命周期函数、事件处理函数等。

**object** 参数说明：

属性	类型	描述
<code>data</code>	<code>Object</code>	页面的初始数据
<code>onLoad</code>	<code>Function</code>	生命周期函数--监听页面加载
<code>onReady</code>	<code>Function</code>	生命周期函数--监听页面初次渲染完成
<code>onShow</code>	<code>Function</code>	生命周期函数--监听页面显示
<code>onHide</code>	<code>Function</code>	生命周期函数--监听页面隐藏
<code>onUnload</code>	<code>Function</code>	生命周期函数--监听页面卸载
<code>onPullDownRefresh</code>	<code>Function</code>	页面相关事件处理函数--监听用户下拉动作
其他	<code>Any</code>	开发者可以添加任意的函数或数据到 <code>object</code> 参数中，用 <code>this</code> 可以访问

示例代码：

```
//index.js
Page({
  data: {
    text: "This is page data."
  },
  onLoad: function(options) {
    // Do some initialize when page load.
  },
  onReady: function() {
    // Do something when page ready.
  },
  onShow: function() {
    // Do something when page show.
  },
  onHide: function() {
    // Do something when page hide.
  },
  onUnload: function() {
    // Do something when page close.
  },
  onPullDownRefresh: function() {
    // Do something when pull down
  },
  // Event handler.
  viewTap: function() {
    this.setData({
      text: 'Set some data for updating view.'
    })
  }
})
```

## 初始化数据

初始化数据将作为页面的第一次渲染。`data` 将会以 JSON 的形式由逻辑层传至渲染层，所以其数据必须是可以转成 JSON 的格式：字符串，数字，布尔值，对象，数组。

渲染层可以通过 [WXML](#) 对数据进行绑定。

示例代码：

```
<view>{{text}}</view>
<view>{{array[0].msg}}</view>
```

```
Page({
  data: {
    text: 'init data',
    array: [{msg: '1'}, {msg: '2'}]
  }
})
```

## 生命周期函数

- `onLoad`：页面加载
  - 一个页面只会调用一次。
  - 参数可以获取 `wx.navigateTo` 和 `wx.redirectTo` 及 `<navigator/>` 中的 `query`。
- `onShow`：页面显示
  - 每次打开页面都会调用一次。
- `onReady`：页面初次渲染完成
  - 一个页面只会调用一次，代表页面已经准备妥当，可以和视图层进行交互。
  - 对界面的设置如 `wx.setNavigationBarTitle` 请在 `onReady` 之后设置。详见[生命周期](#)
- `onHide`：页面隐藏
  - 当 `navigateTo` 或底部 `tab` 切换时调用。
- `onUnload`：页面卸载
  - 当 `redirectTo` 或 `navigateBack` 的时候调用。

## 页面相关事件处理函数

- `onPullDownRefresh`：下拉刷新
  - 监听用户下拉刷新事件。
  - 需要在 `config` 的 `window` 选项中开启 `enablePullDownRefresh`。
  - 当处理完数据刷新后，`wx.stopPullDownRefresh` 可以停止当前页面的下拉刷新。

## 事件处理函数

除了初始化数据和生命周期函数，`Page` 中还可以定义一些特殊的函数：事件处理函数。在渲染层可以在组件中加入[事件绑定](#)，当达到触发事件时，就会执行 `Page` 中定义的事件处理函数。

示例代码：

```
<view bindtap="viewTap"> click me </view>
```

```
Page({
  viewTap: function() {
    console.log('view tap')
  }
})
```

## Page.prototype.setData()

`setData` 函数用于将数据从逻辑层发送到视图层，同时改变对应的 `this.data` 的值。

注意：

1. 直接修改 `this.data` 无效，无法改变页面的状态，还会造成数据不一致。
2. 单次设置的数据不能超过**1024kB**，请尽量避免一次设置过多的数据。

### setData() 参数格式

接受一个对象，以 `key, value` 的形式表示将 `this.data` 中的 `key` 对应的值改变成 `value`。

其中 `key` 可以非常灵活，以数据路径的形式给出，如 `array[2].message`，`a.b.c.d`，并且不需要在 `this.data` 中预先定义。

示例代码：

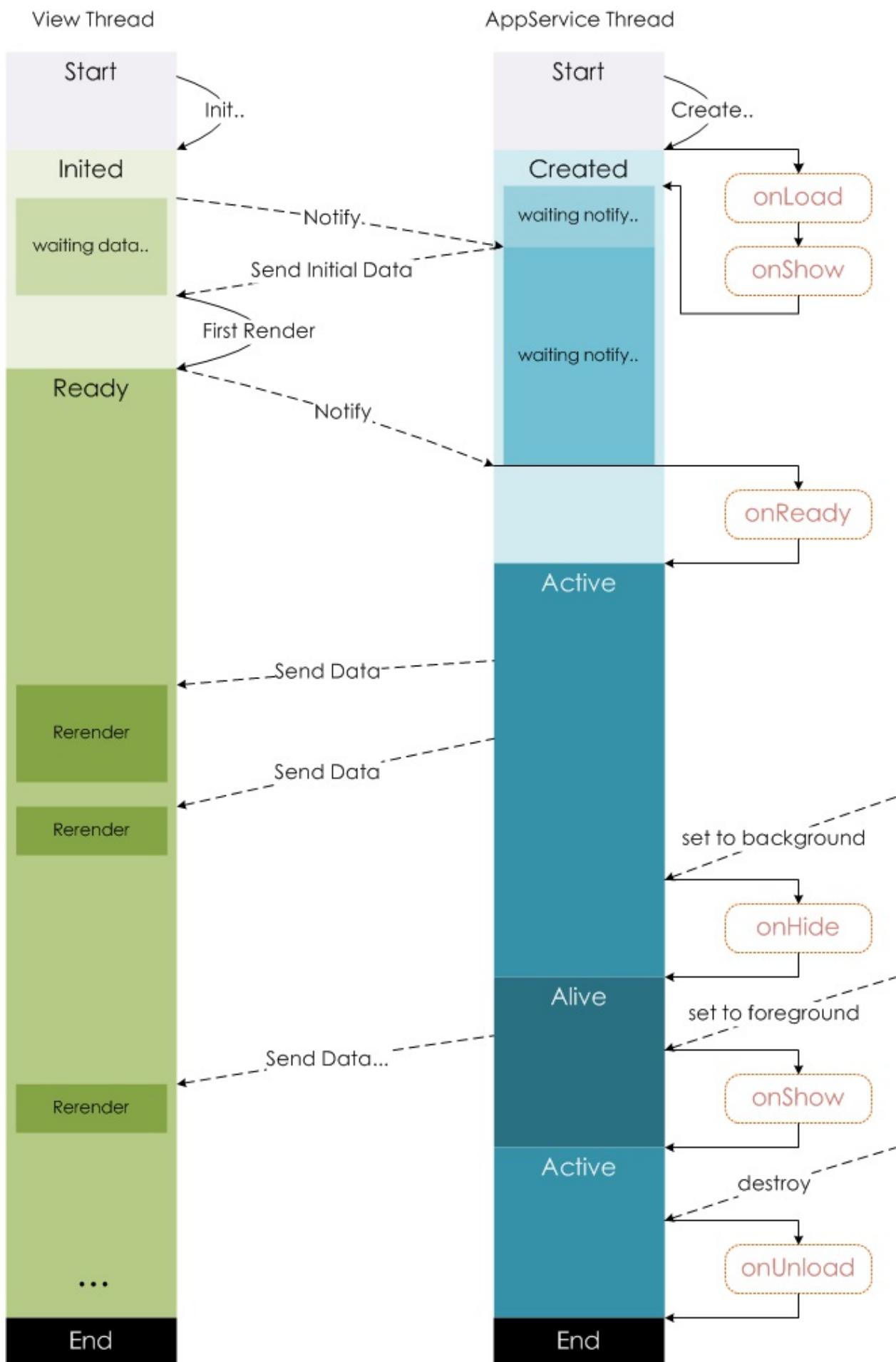
```
<!--index.wxml-->
<view>{{text}}</view>
<button bindtap="changeText"> Change normal data </button>
<view>{{array[0].text}}</view>
<button bindtap="changeItemInArray"> Change Array data </button>
<view>{{obj.text}}</view>
<button bindtap="changeItemInObject"> Change Object data </button>
<view>{{newField.text}}</view>
<button bindtap="addNewField"> Add new data </button>
```

```
//index.js
Page({
  data: {
    text: 'init data',
    array: [{text: 'init data'}],
    object: {
      text: 'init data'
    }
  },
  changeText: function() {
    // this.data.text = 'changed data' // bad, it can not work
    this.setData({
      text: 'changed data'
    })
  },
  changeItemInArray: function() {
    // you can use this way to modify a dynamic data path
    this.setData({
      'array[0].text':'changed data'
    })
  },
  changeItemInObject: function(){
    this.setData({
      'object.text': 'changed data'
    });
  },
  addNewField: function() {
    this.setData({
      'newField.text': 'new data'
    })
  }
})
```

以下内容你不需要立马完全弄明白，不过以后它会有帮助。

## 生命周期

下图说明了 `Page` 实例的生命周期。



## 页面的路由

在小程序中所有页面的路由全部由框架进行管理，对于路由的触发方式以及页面生命周期函数如下：

路由方式	触发时机	路由后页面	路由前页面
初始化	小程序打开的第一个页面	onLoad, onShow	
打开新页面	调用 API <code>wx.navigateTo</code> 或使用组件 <code>&lt;navigator /&gt;</code>	onLoad, onShow	onHide
页面重定向	调用 API <code>wx.redirectTo</code> 或使用组件 <code>&lt;navigator /&gt;</code>	onLoad, onShow	onUnload
页面返回	调用 API <code>wx.navigateBack</code> 或用户按左上角返回按钮	onShow	onUnload
Tab切换	多 Tab 模式下用户切换 Tab	第一次打开 onLoad, onshow; 否则 onShow	onHide

## 文件作用域

在 JavaScript 文件中声明的变量和函数只在该文件中有效；不同的文件中可以声明相同名字的变量和函数，不会互相影响。

通过全局函数 `getApp()` 可以获取全局的应用实例，如果需要全局的数据可以在 `App()` 中设置，如：

```
// app.js
App({
  globalData: 1
})
```

```
// a.js
// The localValue can only be used in file a.js.
var localValue = 'a'
// Get the app instance.
var app = getApp()
// Get the global data and change it.
app.globalData++
```

```
// b.js
// You can redefine localValue in file b.js, without interference with the localValue in a.js.
var localValue = 'b'
// If a.js is run before b.js, now the.globalData should be 2.
console.log(getApp().globalData)
```

## 模块化

我们可以将一些公共的代码抽离成为一个单独的 js 文件，作为一个模块。模块只有通过 `module.exports` 或者 `exports` 才能对外暴露接口。

需要注意的是：

- `exports` 是 `module.exports` 的一个引用，因此在模块里边随意更改 `exports` 的指向会造成未知的错误。所以我们更推荐开发者采用 `module.exports` 来暴露模块接口，除非你已经清晰知道这两者的关系。
- 小程序目前不支持直接引入 `node_modules`，开发者需要使用到 `node_modules` 时候建议拷贝出相关的代码到小程序的目录中。

```
// common.js
function sayHello(name) {
  console.log(`Hello ${name} !`)
}

function sayGoodbye(name) {
  console.log(`Goodbye ${name} !`)
}

module.exports.sayHello = sayHello
exports.sayGoodbye = sayGoodbye
```

在需要使用这些模块的文件中，使用 `require(path)` 将公共代码引入

```
var common = require('common.js')
Page({
  helloMINA: function() {
    common.sayHello('MINA')
  },
  goodbyeMINA: function() {
    common.sayGoodbye('MINA')
  }
})
```

## ES6 语法以及 API 支持

微信小程序运行在三端：iOS、Android 和用于调试的开发者工具

- 在 iOS 上，小程序的 javascript 代码是运行在 JavaScriptCore 中
- 在 Android 上，小程序的 javascript 代码是通过 X5 内核来解析
- 在开发工具上，小程序的 javascript 代码是运行在 nwjs（chrome内核）中

虽然尽管三端的环境是十分相似的，但是至少在目前还是有一些区别的，这给很多开发者带来很大的困扰。

在 0.10.101000 以及之后版本的开发工具中，会默认使用 babel 将开发者代码 ES6 语法转换为三端都能很好支持的 ES5 的代码，帮助开发者解决环境不同所带来的开发问题。开发者可以在项目设置中关闭这个功能。

需要注意的是：

- 这种转换只会帮助开发处理语法上问题，新的 ES6 的 API 例如 Promise 等需要开发者自行引入 Polyfill 或者别的类库。
- 为了提高代码质量，在开启 ES6 转换功能的情况下，默认启用 javascript 严格模式，请参考 "use strict"。

# API

小程序开发框架提供丰富的微信原生 API，可以方便的调起微信提供的能力，如获取用户信息，本地存储，支付功能等。

详细介绍请参考 [API 文档](#)

## 视图层

框架的视图层由 **WXML** 与 **WXSS** 编写，由组件来进行展示。

将逻辑层的数据反应成视图，同时将视图层的事件发送给逻辑层。

**WXML**(WeiXin Markup language)用于描述页面的结构。

**WXSS**(WeiXin Style Sheet)用于描述页面的样式。

组件(**Component**)是视图的基本组成单元。

# WXML

WXML（WeiXin Markup Language）是框架设计的一套标签语言，结合[基础组件](#)、[事件系统](#)，可以构建出页面的结构。

用以下一些简单的例子来看看 WXML 具有什么能力：

## 数据绑定

```
<!--wxml-->
<view>{{message}}</view>
```

```
// page.js
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

## 列表渲染

```
<!--wxml-->
<view wx:for="{{array}}"> {{item}} </view>
```

```
// page.js
Page({
  data: {
    array: [1, 2, 3, 4, 5]
  }
})
```

## 条件渲染

```
<!--wxml-->
<view wx:if="{{view == 'WEBVIEW'}}"> WEBVIEW </view>
<view wx:elif="{{view == 'APP'}}"> APP </view>
<view wx:else="{{view == 'MINA'}}"> MINA </view>
```

```
// page.js
Page({
  data: {
    view: 'MINA'
  }
})
```

## 模板

```
<!--wxml-->
<template name="staffName">
  <view>
    FirstName: {{firstName}}, LastName: {{lastName}}
  </view>
</template>

<template is="staffName" data="{{...staffA}}></template>
<template is="staffName" data="{{...staffB}}></template>
<template is="staffName" data="{{...staffC}}></template>
```

```
// page.js
Page({
  data: {
    staffA: {firstName: 'Hulk', lastName: 'Hu'},
    staffB: {firstName: 'Shang', lastName: 'You'},
    staffC: {firstName: 'Gideon', lastName: 'Lin'}
  }
})
```

## 事件

```
<view bindtap="add"> {{count}} </view>
```

```
Page({
  data: {
    count: 1
  },
  add: function(e) {
    this.setData({
      count: this.data.count + 1
    })
  }
})
```

具体的能力以及使用方式在以下章节查看：

[数据绑定、列表渲染、条件渲染、模板、事件、引用](#)

# 数据绑定

WXML 中的动态数据均来自对应 Page 的 data。

## 简单绑定

数据绑定使用 Mustache 语法（双大括号）将变量包起来，可以作用于：

### 内容

```
<view>{{message}} </view>
```

```
Page({
  data: {
    message: 'Hello MINA!'
  }
})
```

### 组件属性(需要在双引号之内)

```
<view id="item-{{id}}"> </view>
```

```
Page({
  data: {
    id: 0
  }
})
```

### 控制属性(需要在双引号之内)

```
<view wx:if="{{condition}}"> </view>
```

```
Page({
  data: {
    condition: true
  }
})
```

## 运算

可以在 {{ }} 内进行简单的运算，支持的有如下几种方式：

### 三元运算

```
<view hidden="{{flag ? true : false}}> Hidden </view>
```

## 算数运算

```
<view> {{a + b}} + {{c}} + d </view>
```

```
Page({  
  data: {  
    a: 1,  
    b: 2,  
    c: 3  
  }  
)
```

view中的内容为 `3 + 3 + d`。

## 逻辑判断

```
<view wx:if="{{length > 5}}> </view>
```

## 字符串运算

```
<view>{{"hello" + name}}</view>
```

```
Page({  
  data:{  
    name: 'MINA'  
  }  
)
```

## 组合

也可以在 **Mustache** 内直接进行组合，构成新的对象或者数组。

## 数组

```
<view wx:for="{{[zero, 1, 2, 3, 4]}}> {{item}} </view>
```

```
Page({  
  data: {  
    zero: 0  
  }  
)
```

最终组合成数组 `[0, 1, 2, 3, 4]`。

## 对象

```
<template is="objectCombine" data="{{for: a, bar: b}}></template>
```

```
Page({  
  data: {  
    a: 1,  
    b: 2  
  }  
})
```

最终组合成的对象是 `{for: 1, bar: 2}`

也可以用扩展运算符 `...` 来将一个对象展开

```
<template is="objectCombine" data="{{...obj1, ...obj2, e: 5}}"></template>
```

```
Page({  
  data: {  
    obj1: {  
      a: 1,  
      b: 2  
    },  
    obj2: {  
      c: 3,  
      d: 4  
    }  
  }  
})
```

最终组合成的对象是 `{a: 1, b: 2, c: 3, d: 4, e: 5}`。

如果对象的 `key` 和 `value` 相同，也可以间接地表达。

```
<template is="objectCombine" data="{{foo, bar}}"></template>
```

```
Page({  
  data: {  
    foo: 'my-foo',  
    bar: 'my-bar'  
  }  
})
```

最终组合成的对象是 `{foo: 'my-foo', bar:'my-bar'}`。

注意：上述方式可以随意组合，但是如有存在变量名相同的情况，后边的会覆盖前面，如：

```
<template is="objectCombine" data="{{...obj1, ...obj2, a, c: 6}}"></template>
```

```
Page({  
  data: {  
    obj1: {  
      a: 1,  
      b: 2  
    },  
    obj2: {  
      b: 3,  
      c: 4  
    },  
    a: 5  
  }  
})
```

最终组合成的对象是 {a: 5, b: 3, c: 6}。

# 条件渲染

## wx:if

在框架中，我们用 `wx:if="{{condition}}"` 来判断是否需要渲染该代码块：

```
<view wx:if="{{condition}}"> True </view>
```

也可以用 `wx:elif` 和 `wx:else` 来添加一个 `else` 块：

```
<view wx:if="{{length > 5}}> 1 </view>
<view wx:elif="{{length > 2}}> 2 </view>
<view wx:else> 3 </view>
```

## block wx:if

因为 `wx:if` 是一个控制属性，需要将它添加到一个标签上。但是如果我们想一次性判断多个组件标签，我们可以使用一个 `<block/>` 标签将多个组件包装起来，并在上边使用 `wx:if` 控制属性。

```
<block wx:if="{{true}}">
  <view> view1 </view>
  <view> view2 </view>
</block>
```

注意：`<block/>` 并不是一个组件，它仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

## wx:if vs hidden

因为 `wx:if` 之中的模板也可能包含数据绑定，所有当 `wx:if` 的条件值切换时，框架有一个局部渲染的过程，因为它会确保条件块在切换时销毁或重新渲染。

同时 `wx:if` 也是惰性的，如果在初始渲染条件为 `false`，框架什么也不做，在条件第一次变成真的时候才开始局部渲染。

相比之下，`hidden` 就简单的多，组件始终会被渲染，只是简单的控制显示与隐藏。

一般来说，`wx:if` 有更高的切换消耗而 `hidden` 有更高的初始渲染消耗。因此，如果需要频繁切换的情景下，用 `hidden` 更好，如果在运行时条件不大可能改变则 `wx:if` 较好。

# 列表渲染

## wx:for

在组件上使用 `wx:for` 控制属性绑定一个数组，即可使用数组中各项的数据重复渲染该组件。

默认数组的当前项的下标变量名默认为 `index`，数组当前项的变量名默认为 `item`

```
<view wx:for="{{items}}>
  {{index}}: {{item.message}}
</view>
```

```
Page({
  data: {
    items: [
      {
        message: 'foo',
      },
      {
        message: 'bar'
      }
    ]
  }
})
```

使用 `wx:for-item` 可以指定数组当前元素的变量名

使用 `wx:for-index` 可以指定数组当前下标的变量名：

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  {{idx}}: {{itemName.message}}
</view>
```

`wx:for` 也可以嵌套，下边是一个九九乘法表

```
<view wx:for="{{{1, 2, 3, 4, 5, 6, 7, 8, 9}}}" wx:for-item="i">
  <view wx:for="{{{1, 2, 3, 4, 5, 6, 7, 8, 9}}}" wx:for-item="j">
    <view wx:if="{{i <= j}}">
      {{i}} * {{j}} = {{i * j}}
    </view>
  </view>
</view>
```

## block wx:for

类似 `block wx:if`，也可以将 `wx:for` 用在 `<block/>` 标签上，以渲染一个包含多节点的结构块。例如：

```
<block wx:for="{{{1, 2, 3}}}">
  <view> {{index}}: </view>
  <view> {{item}} </view>
</block>
```

# 模板

WXML提供模板（`template`），可以在模板中定义代码片段，然后在不同的地方调用。

## 定义模板

使用`name`属性，作为模板的名字。然后在`<template>`内定义代码片段，如：

```
<!--
  index: int
  msg: string
  time: string
-->
<template name="msgItem">
  <view>
    <text> {{index}}: {{msg}} </text>
    <text> Time: { {time}} </text>
  </view>
</template>
```

## 使用模板

使用`is`属性，声明需要的使用的模板，然后将模板所需要的`data`传入，如：

```
<template is="msgItem" data="{{...item}}"/>
```

```
Page({
  data: {
    item: {
      index: 0,
      msg: 'this is a template',
      time: '2016-09-15'
    }
  }
})
```

`is`属性可以使用 Mustache 语法，来动态决定具体需要渲染哪个模板：

```
<template name="odd">
  <view> odd </view>
</template>
<template name="even">
  <view> even </view>
</template>

<block wx:for="{{[1, 2, 3, 4, 5]}}">
  <template is="{{item % 2 == 0 ? 'even' : 'odd'}}"/>
</block>
```

## 模板的作用域

模板拥有自己的作用域，只能使用`data`传入的数据。

# 事件

## 什么是事件

- 事件是视图层到逻辑层的通讯方式。
- 事件可以将用户的行为反馈到逻辑层进行处理。
- 事件可以绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数。
- 事件对象可以携带额外信息，如id, dataset, touches。

## 事件的使用方式

- 在组件中绑定一个事件处理函数。

如 `bindtap`，当用户点击该组件的时候会在该页面对应的**Page**中找到相应的事件处理函数。

```
<view id="tapTest" data-hi="MINA" bindtap="tapName"> Click me! </view>
```

- 在相应的**Page**定义中写上相应的事件处理函数，参数是**event**。

```
Page({
  tapName: function(event) {
    console.log(event)
  }
})
```

- 可以看到**log**出来的信息大致如下：

```
{
  "type": "tap",
  "timeStamp": 1252,
  "target": {
    "id": "tapTest",
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
      "hi": "MINA"
    }
  },
  "currentTarget": {
    "id": "tapTest",
    "offsetLeft": 0,
    "offsetTop": 0,
    "dataset": {
      "hi": "MINA"
    }
  },
  "touches": [{{
    "pageX": 30,
    "pageY": 12,
    "clientX": 30,
    "clientY": 12,
    "screenX": 112,
    "screenY": 151
}}],
  "detail": {
    "x": 30,
    "y": 12
  }
}
```

## 事件详解

### 事件分类

事件分为冒泡事件和非冒泡事件：

1. 冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。
2. 非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML的冒泡事件列表：

类型	触发条件
touchstart	手指触摸
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断，如来电提醒，弹窗
touchend	手指触摸动作结束
tap	手指触摸后离开
longtap	手指触摸后，超过350ms再离开

注：除上表之外的其他组件自定义事件都是非冒泡事件，如 `<form/>` 的 `submit` 事件，`<input/>` 的 `input` 事件，`<scroll-view/>` 的 `scroll` 事件，(详见各个\*\*组件\*\*)

### 事件绑定

事件绑定的写法同组件的属性，以 **key**、**value** 的形式。

- **key** 以 `bind` 或 `catch` 开头，然后跟上事件的类型，如 `bindtap`，`catchtouchstart`
- **value** 是一个字符串，需要在对应的 `Page` 中定义同名的函数。不然当触发事件的时候会报错。

`bind` 事件绑定不会阻止冒泡事件向上冒泡，`catch` 事件绑定可以阻止冒泡事件向上冒泡。

如在下边这个例子中，点击 `inner view` 会先后触发 `handleTap3` 和 `handleTap2` (因为 `tap` 事件会冒泡到 `middle view`，而 `middle view` 阻止了 `tap` 事件冒泡，不再向父节点传递)，点击 `middle view` 会触发 `handleTap2`，点击 `outer view` 会触发 `handleTap1`。

```
<view id="outer" bindtap="handleTap1">
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">
      inner view
    </view>
  </view>
</view>
```

## 事件对象

如无特殊说明，当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。

事件对象的属性列表：

属性	类型	说明
<code>type</code>	<code>String</code>	事件类型
<code>timeStamp</code>	<code>Integer</code>	事件生成时的时间戳
<code>target</code>	<code>Object</code>	触发事件的组件的一些属性值集合
<code>currentTarget</code>	<code>Object</code>	当前组件的一些属性值集合
<code>touches</code>	<code>Array</code>	触摸事件，触摸点信息的数组
<code>detail</code>	<code>Object</code>	额外的信息

### **type**

通用事件类型

### **timeStamp**

该页面打开到触发事件所经过的毫秒数。

### **target**

触发事件的源组件。

属性	说明
<code>id</code>	事件源组件的id
<code>dataset</code>	事件源组件上由 <code>data-</code> 开头的自定义属性组成的集合
<code>offsetLeft, offsetTop</code>	事件源组件的坐标系统中偏移量

## currentTarget

事件绑定的当前组件。

属性	说明
<code>id</code>	当前组件的 id
<code>dataset</code>	当前组件上由 <code>data-</code> 开头的自定义属性组成的集合
<code>offsetLeft, offsetTop</code>	当前组件的坐标系统中偏移量

说明：`target` 和 `currentTarget` 可以参考上例中，点击 `inner view` 时，`handleTap3` 收到的事件对象 `target` 和 `currentTarget` 都是 `inner`，而 `handleTap2` 收到的事件对象 `target` 就是 `inner`，`currentTarget` 就是 `middle`

## dataset

在组件中可以定义数据，这些数据将会通过事件传递给 SERVICE。书写方式：以 `data-` 开头，多个单词由连字符 - 链接，不能有大写(大写会自动转成小写)如 `data-element-type`，最终在 `event.target.dataset` 中会将连字符转成驼峰 `elementType`。

示例：

```
<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap"> DataSet Test </view>
```

```
Page({
  bindViewTap:function(event){
    event.target.dataset.alphaBeta == 1 // - 会转为驼峰写法
    event.target.dataset.alphabeta == 2 // 大写会转为小写
  }
})
```

## touches

`touches`是一个触摸点的数组，每个触摸点包括以下属性：

属性	说明
<code>pageX,pageY</code>	距离文档左上角的距离，文档的左上角为原点，横向为X轴，纵向为Y轴
<code>clientX,clientY</code>	距离页面可显示区域（屏幕除去导航条）左上角距离，横向为X轴，纵向为Y轴
<code>screenX,screenY</code>	距离屏幕左上角的距离，屏幕左上角为原点，横向为X轴，纵向为Y轴

## detail

特殊事件所携带的数据，如表单组件的提交事件会携带用户的输入，媒体的错误事件会携带错误信息，详见[组件](#)定义中各个事件的定义。

# 引用

WXML 提供两种文件引用方式 `import` 和 `include`。

## import

`import` 可以在该文件中使用目标文件定义的 `template`，如：

在 `item.wxml` 中定义了一个叫 `item` 的 `template`：

```
<!-- item.wxml -->
<template name="item">
  <text>{{text}}</text>
</template>
```

在 `index.wxml` 中引用了 `item.wxml`，就可以使用 `item` 模板：

```
<import src="item.wxml"/>
<template is="item" data="{{text: 'forbar'}}"/>
```

## import 的作用域

`import` 有作用域的概念，即只会 `import` 目标文件中定义的 `template`，而不会 `import` 目标文件 `import` 的 `template`。

如：**C import B, B import A**，在**C**中可以使用**B**定义的 `template`，在**B**中可以使用**A**定义的 `template`，但是**C**不能使用**A**定义的 `template`。

```
<!-- A.wxml -->
<template name="A">
  <text> A template </text>
</template>
```

```
<!-- B.wxml -->
<import src="a.wxml"/>
<template name="B">
  <text> B template </text>
</template>
```

```
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/>  <!-- Error! Can not use template when not import A. -->
<template is="B"/>
```

## include

`include` 可以将目标文件除了 `<template/>` 的整个代码引入，相当于拷贝到 `include` 位置，如：

```
<!-- index.wxml -->
<include src="header.wxml"/>
<view> body </view>
<include src="footer.wxml"/>
```

```
<!-- header.wxml -->
<view> header </view>
```

```
<!-- footer.wxml -->
<view> footer </view>
```

# WXSS

WXSS(WeiXin Style Sheets)是一套样式语言，用于描述 WXML 的组件样式。

WXSS 用来决定 WXML 的组件应该怎么显示。

为了适应广大的前端开发者，我们的 WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，我们对 CSS 进行了扩充以及修改。

与 CSS 相比我们扩展的特性有：

- 尺寸单位
- 样式导入

## 尺寸单位

- **rpx (responsive pixel)** : 可以根据屏幕宽度进行自适应。规定屏幕宽为750rpx。如在 iPhone6 上，屏幕宽度为375px，共有750个物理像素，则 $750\text{rpx} = 375\text{px} = 750\text{物理像素}$ ， $1\text{rpx} = 0.5\text{px} = 1\text{物理像素}$ 。

设备	rpx换算px (屏幕宽度/750)	px换算rpx (750/屏幕宽度)
iPhone5	$1\text{rpx} = 0.42\text{px}$	$1\text{px} = 2.34\text{rpx}$
iPhone6	$1\text{rpx} = 0.5\text{px}$	$1\text{px} = 2\text{rpx}$
iPhone6 Plus	$1\text{rpx} = 0.552\text{px}$	$1\text{px} = 1.81\text{rpx}$

- **rem (root em)** : 规定屏幕宽度为20rem； $1\text{rem} = (750/20)\text{rpx}$ 。

建议：开发微信小程序时设计师可以用 iPhone6 作为视觉稿的标准。注意：在较小的屏幕上不可避免的会有一些毛刺，请在开发时尽量避免这种情况。

## 样式导入

使用 `@import` 语句可以导入外联样式表，`@import` 后跟需要导入的外联样式表的相对路径，用 ; 表示语句结束。

示例代码：

```
/** common.wxss */
.small-p {
  padding:5px;
}
```

```
/** app.wxss */
@import "common.wxss";
.middle-p {
  padding:15px;
}
```

## 内联样式

框架组件上支持使用 `style`、`class` 属性来控制组件的样式。

- **style**: 静态的样式统一写到 `class` 中。`style` 接收动态的样式，在运行时会进行解析，请尽量避免将静态的样式写进 `style` 中，以免影响渲染速度。

```
<view style="color:{{color}};" />
```

- **class**: 用于指定样式规则，其属性值是样式规则中类选择器名(样式类名)的集合，样式类名不需要带上`.`，样式类名之间用空格分隔。

```
<view class="normal_view" />
```

## 选择器

目前支持的选择器有：

选择器	样例	样例描述
.class	.intro	选择所有拥有 <code>class="intro"</code> 的组件
#id	#firstname	选择拥有 <code>id="firstname"</code> 的组件
element	view	选择所有 <code>view</code> 组件
element, element	view checkbox	选择所有文档的 <code>view</code> 组件和所有的 <code>checkbox</code> 组件
::after	view::after	在 <code>view</code> 组件后边插入内容
::before	view::before	在 <code>view</code> 组件前边插入内容

## 全局样式与局部样式

定义在 `app.wxss` 中的样式为全局样式，作用于每一个页面。在 `page` 的 `wxss` 文件中定义的样式为局部样式，只作用在对应的页面，并会覆盖 `app.wxss` 中相同的选择器。

# 基础组件

框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。

详细介绍请参考[组件文档](#)

# 基础组件

框架为开发者提供了一系列基础组件，开发者可以通过组合这些基础组件进行快速开发。

什么是组件：

- 组件是视图层的基本组成单元。
- 组件自带一些功能与微信风格的样式。
- 一个组件通常包括 `开始标签` 和 `结束标签`，`属性` 用来修饰这个组件，`内容` 在两个标签之内。

```
<tagname property="value">  
Content goes here ...  
</tagname>
```

注意：所有组件与属性都是小写，以连字符 - 连接

## 属性类型

类型	描述	注解
<code>Boolean</code>	布尔值	组件写上该属性，不管该属性等于什么，其值都为 <code>true</code> ，只有组件上没有写该属性时，属性值才为 <code>false</code> 。如果属性值为变量，变量的值会被转换为 <code>Boolean</code> 类型
<code>Number</code>	数字	<code>1</code> , <code>2.5</code>
<code>String</code>	字符串	<code>"string"</code>
<code>Array</code>	数组	<code>[ 1, "string" ]</code>
<code>Object</code>	对象	<code>{ key: value }</code>
<code>EventHandler</code>	事件处理函数名	<code>"handlerName"</code> 是 <a href="#">Page</a> 中定义的事件处理函数名
<code>Any</code>	任意属性	-

## 共同属性类型

所有组件都有的属性：

属性名	类型	描述	注解
<code>id</code>	<code>String</code>	组件的唯一标示	保持整个页面唯一
<code>class</code>	<code>String</code>	组件的样式类	在对应的 <code>WXSS</code> 中定义的样式类
<code>style</code>	<code>String</code>	组件的内联样式	可以动态设置的内联样式
<code>hidden</code>	<code>Boolean</code>	组件是否显示	所有组件默认显示
<code>data-*</code>	<code>Any</code>	自定义属性	组件上触发的事件时，会发送给事件处理函数
<code>bind* / catch*</code>	<code>EventHandler</code>	组件的事件	详见 <a href="#">事件</a>

## 特殊属性

几乎所有组件都有各自定义的属性，可以对该组件的功能或样式进行修饰，请参考各个[组件](#)的定义。

## 组件列表

基础组件分为以下八大类：

### 视图容器(**View Container**):

组件名	说明
<code>view</code>	视图容器
<code>scroll-view</code>	可滚动视图容器
<code>swiper</code>	滑块视图容器

### 基础内容(**Basic Content**):

组件名	说明
<code>icon</code>	图标
<code>text</code>	文字
<code>progress</code>	进度条

### 表单(**Form**):

标签名	说明
<code>button</code>	按钮
<code>form</code>	表单
<code>input</code>	输入框
<code>checkbox</code>	多项选择器
<code>radio</code>	单项选择器
<code>picker</code>	列表选择器
<code>slider</code>	滚动选择器
<code>switch</code>	开关选择器
<code>label</code>	标签

**操作反馈(Interaction):**

组件名	说明
action-sheet	上拉菜单
modal	模态弹窗
toast	消息提示框
loading	加载提示符

**导航(Navigation):**

组件名	说明
navigator	应用链接

**多媒体(Media):**

组件名	说明
audio	音频
image	图片
video	视频

**地图(Map):**

组件名	说明
map	地图

**画布(Canvas):**

组件名	说明
canvas	画布

## view

视图容器。

示例：

```
<view class="section">
  <view class="section__title">flex-direction: row</view>
  <view class="flex-wrp" style="flex-direction:row;">
    <view class="flex-item bc_green">1</view>
    <view class="flex-item bc_red">2</view>
    <view class="flex-item bc_blue">3</view>
  </view>
</view>
<view class="section">
  <view class="section__title">flex-direction: column</view>
  <view class="flex-wrp" style="height: 300px;flex-direction:column;">
    <view class="flex-item bc_green">1</view>
    <view class="flex-item bc_red">2</view>
    <view class="flex-item bc_blue">3</view>
  </view>
</view>
```



view

flex-direction: row



flex-direction: column



## scroll-view

可滚动视图区域。

属性名	类型	默认值	说明
scroll-x	Boolean	false	允许横向滚动
scroll-y	Boolean	false	允许纵向滚动
upper-threshold	Number	50	距顶部/左边多远时（单位px），触发 scrolltoupper 事件
lower-threshold	Number	50	距底部/右边多远时（单位px），触发 scrolltolower 事件
scroll-top	Number		设置竖向滚动条位置
scroll-left	Number		设置横向滚动条位置
scroll-into-view	String		值应为某子元素id，则滚动到该元素，元素顶部对齐滚动区域顶部
bindscrolltoupper	EventHandle		滚动到顶部/左边，会触发 scrolltoupper 事件
bindscrolltolower	EventHandle		滚动到底部/右边，会触发 scrolltolower 事件
bindscroll	EventHandle		滚动时触发，event.detail = {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}

使用竖向滚动时，需要给 `<scroll-view/>` 一个固定高度，通过 WXSS 设置 height。

示例代码：

```
<view class="section">
  <view class="section__title">vertical scroll</view>
  <scroll-view scroll-y="true" style="height: 200px;" bindscrolltoupper="upper" bindscrolltolower="lower" bindscroll="scroll" scroll-into-view="{{toView}}" scroll-top="{{scrollTop}}>
    <view id="green" class="scroll-view-item bc_green"></view>
    <view id="red" class="scroll-view-item bc_red"></view>
    <view id="yellow" class="scroll-view-item bc_yellow"></view>
    <view id="blue" class="scroll-view-item bc_blue"></view>
  </scroll-view>

  <view class="btn-area">
    <button size="mini" bindtap="tap">click me to scroll into view </button>
    <button size="mini" bindtap="tapMove">click me to scroll</button>
  </view>
</view>
<view class="section section_gap">
  <view class="section__title">horizontal scroll</view>
  <scroll-view class="scroll-view_H" scroll-x="true" style="width: 100%">
    <view id="green" class="scroll-view-item_H bc_green"></view>
    <view id="red" class="scroll-view-item_H bc_red"></view>
    <view id="yellow" class="scroll-view-item_H bc_yellow"></view>
    <view id="blue" class="scroll-view-item_H bc_blue"></view>
  </scroll-view>
</view>
```

```
var order = ['red', 'yellow', 'blue', 'green', 'red']
Page({
  data: {
    toView: 'red',
    scrollTop: 100
  },
  upper: function(e) {
    console.log(e)
  },
  lower: function(e) {
    console.log(e)
  },
  scroll: function(e) {
    console.log(e)
  },
  tap: function(e) {
    for (var i = 0; i < order.length; ++i) {
      if (order[i] === this.data.toView) {
        this.setData({
          toView: order[i + 1]
        })
        break
      }
    }
  },
  tapMove: function(e) {
    this.setData({
      scrollTop: this.data.scrollTop + 10
    })
  }
})
```

## scroll-view

可滚动视图区域。

### vertical scroll



click me to scroll into view

click me to scroll

### horizontal scroll



## swiper

滑块视图容器。

属性名	类型	默认值	说明
indicator-dots	Boolean	false	是否显示面板指示点
autoplay	Boolean	false	是否自动切换
current	Number	0	当前所在页面的 index
interval	Number	5000	自动切换时间间隔
duration	Number	1000	滑动动画时长
bindchange	EventHandle		current 改变时会触发 change 事件, event.detail = {current: current}

注意： 其中只可放置 `<swiper-item>` 组件，其他节点会被自动删除。

## swiper-item

仅可放置在 `<swiper>` 组件中，宽高自动设置为100%。

示例代码：

```
<swiper indicator-dots="{{indicatorDots}}"
  autoplay="{{autoplay}}" interval="{{interval}}" duration="{{duration}}">
  <block wx:for="{{imgUrls}}">
    <swiper-item>
      <image src="{{item}}" class="slide-image" width="355" height="150"/>
    </swiper-item>
  </block>
</swiper>
<button bindtap="changeIndicatorDots"> indicator-dots </button>
<button bindtap="changeAutoplay"> autoplay </button>
<slider bindchange="intervalChange" show-value min="500" max="2000"/> interval
<slider bindchange="durationChange" show-value min="1000" max="10000"/> duration
```

```
Page({
  data: {
    imgUrls: [
      'http://img02.tooopen.com/images/20150928/tooopen_sy_143912755726.jpg',
      'http://img06.tooopen.com/images/20160818/tooopen_sy_175866434296.jpg',
      'http://img06.tooopen.com/images/20160818/tooopen_sy_175833047715.jpg'
    ],
    indicatorDots: false,
    autoplay: false,
    interval: 5000,
    duration: 1000
  },
  changeIndicatorDots: function(e) {
    this.setData({
      indicatorDots: !this.data.indicatorDots
    })
  },
  changeAutoplay: function(e) {
    this.setData({
      autoplay: !this.data.autoplay
    })
  },
  intervalChange: function(e) {
    this.setData({
      interval: e.detail.value
    })
  },
  durationChange: function(e) {
    this.setData({
      duration: e.detail.value
    })
  }
})
```

## icon

图标。

属性名	类型	默认值	说明
type	String		icon的类型，有效值: success, success_no_circle, info, warn, waiting, cancel, download, search, clear
size	Number	23	icon的大小，单位px
color	Color		icon的颜色，同css的color

示例：

```
<view class="group">
  <block wx:for="{{iconSize}}">
    <icon type="success" size="{{item}}"/>
  </block>
</view>

<view class="group">
  <block wx:for="{{iconType}}">
    <icon type="{{item}}" size="45"/>
  </block>
</view>

<view class="group">
  <block wx:for="{{iconColor}}">
    <icon type="success" size="45" color="{{item}}"/>
  </block>
</view>
```

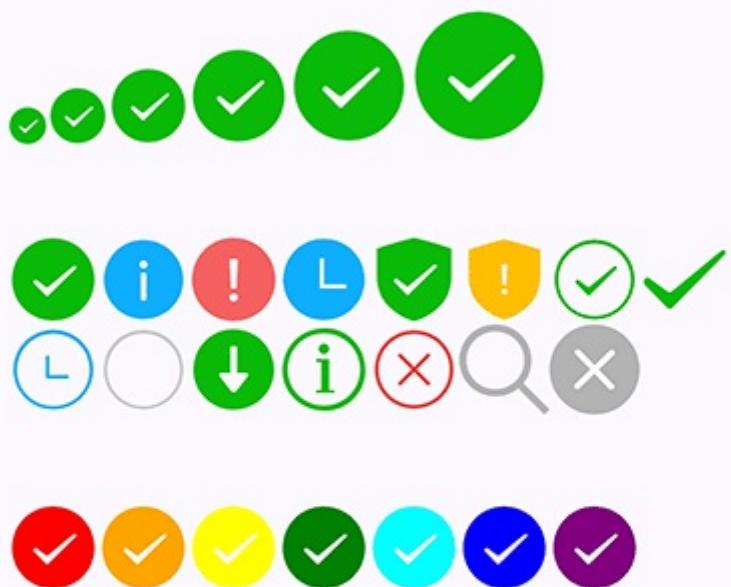
```
Page({
  data: {
    iconSize: [20, 30, 40, 50, 60, 70],
    iconColor: [
      'red', 'orange', 'yellow', 'green', 'rgb(0,255,255)', 'blue', 'purple'
    ],
    iconType: [
      'success', 'info', 'warn', 'waiting', 'safe_success', 'safe_warn',
      'success_circle', 'success_no_circle', 'waiting_circle', 'circle', 'download',
      'info_circle', 'cancel', 'search', 'clear'
    ]
  }
})
```



## 演示



icon



## text

文本。

支持转义符"\\"。

除了文本节点以外的其他节点都无法长按选中。

示例：

```
<view class="btn-area">
  <view class="body-view">
    <text>{{text}}</text>
    <button bindtap="add">add line</button>
    <button bindtap="remove">remove line</button>
  </view>
</view>
```

```
var initData = 'this is first line\nthis is second line'
var extraLine = [];
Page({
  data: {
    text: initData
  },
  add: function(e) {
    extraLine.push('other line')
    this.setData({
      text: initData + '\n' + extraLine.join('\n')
    })
  },
  remove: function(e) {
    if (extraLine.length > 0) {
      extraLine.pop()
      this.setData({
        text: initData + '\n' + extraLine.join('\n')
      })
    }
  }
})
```



## progress

进度条。

属性名	类型	默认值	说明
percent	Float	无	百分比0~100
show-info	Boolean	false	在进度条右侧显示百分比
stroke-width	Number	6	进度条线的宽度，单位px
color	Color	#09BB07	进度条颜色
active	Boolean	false	进度条从左往右的动画

示例：

```
<progress percent="20" show-info />
<progress percent="40" stroke-width="12" />
<progress percent="60" color="pink" />
<progress percent="80" active />
```

.....

中国联通

20:46



61%



## 演示



progress



## button

按钮。

属性名	类型	默认值	说明
size	String	default	有效值 default, mini
type	String	default	按钮的样式类型，有效值 primary, default, warn
plain	Boolean	false	按钮是否镂空，背景色透明
disabled	Boolean	false	是否禁用
loading	Boolean	false	名称前是否带 loading 图标
form-type	String	无	有效值: submit, reset, 用于 <form/> 组件，点击分别会触发 submit/reset 事件
hover-class	String	button-hover	指定按钮按下去的样式类。当 hover-class="none" 时，没有点击态效果

注： button-hover 默认为 {background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}

示例代码：

```
/** wxss */
/** 修改button默认的点击态样式类*/
.button-hover {
  background-color: red;
}
/** 添加自定义button点击态样式类*/
.other-button-hover {
  background-color: blur;
}
```

```
<button type="default" size="{{defaultSize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="default" hover-class="other-button-hover"> default </button>
<button type="primary" size="{{primarySize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="primary"> primary </button>
<button type="warn" size="{{warnSize}}" loading="{{loading}}" plain="{{plain}}"
        disabled="{{disabled}}" bindtap="warn"> warn </button>
<button bindtap="setDisabled">点击设置以上按钮disabled属性</button>
<button bindtap="setPlain">点击设置以上按钮plain属性</button>
<button bindtap=" setLoading">点击设置以上按钮loading属性</button>
```

```

var types = ['default', 'primary', 'warn']
var pageObject = {
  data: {
    defaultSize: 'default',
    primarySize: 'default',
    warnSize: 'default',
    disabled: false,
    plain: false,
    loading: false
  },
  setDisabled: function(e) {
    this.setData({
      disabled: !this.data.disabled
    })
  },
  setPlain: function(e) {
    this.setData({
      plain: !this.data.plain
    })
  },
  setLoading: function(e) {
    this.setData({
      loading: !this.data.loading
    })
  }
}

for (var i = 0; i < types.length; ++i) {
  (function(type) {
    pageObject[type] = function(e) {
      var key = type + 'Size'
      var changedData = {}
      changedData[key] =
        this.data[key] === 'default' ? 'mini' : 'default'
      this.setData(changedData)
    }
  })(types[i])
}

Page(pageObject)

```



## checkbox-group

多项选择器，内部由多个 `checkbox` 组成。

属性名	类型	默认值	说明
bindchange	EventHandle		<checkbox-group/> 中选中项发生改变是触发 change 事件， <code>detail = {value:[选中的checkbox的value的数组]}</code>

## checkbox

多选项目。

属性名	类型	默认值	说明
value	String		<checkbox/> 标识，选中时触发 <checkbox-group/> 的 change 事件，并携带 <checkbox/> 的 value
disabled	Boolean	false	是否禁用
checked	Boolean	false	当前是否选中，可用来设置默认选中

示例：

```
<checkbox-group bindchange="checkboxChange">
  <label class="checkbox" wx:for="{{items}}>
    <checkbox value="{{item.name}}" checked="{{item.checked}}"/>{{item.value}}
  </label>
</checkbox-group>
```

```
Page({
  data: {
    items: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本'},
      {name: 'ENG', value: '英国'},
      {name: 'TUR', value: '法国'}
    ]
  },
  checkboxChange: function(e) {
    console.log('checkbox发生change事件，携带value值为：', e.detail.value)
  }
})
```

## checkbox

多选框

美国

中国

巴西

日本

英国

法国

## form

表单，将组件内的用户输入的 `<switch/>` `<input/>` `<checkbox/>` `<slider/>` `<radio/>` `<picker/>` 提交。

属性名	类型	说明
<code>report-submit</code>	<code>Boolean</code>	是否返回formId用于发送 <a href="#">模板消息</a>
<code>bindsubmit</code>	<code>EventHandle</code>	携带form中的数据触发submit事件， <code>event.detail = {value : {'name': 'value'}, formId: ''}</code>
<code>bindreset</code>	<code>EventHandle</code>	表单重置时会触发reset事件

示例代码：

```
<form bindsubmit="formSubmit" bindreset="formReset">
  <view class="section section_gap">
    <view class="section__title">switch</view>
    <switch name="switch"/>
  </view>
  <view class="section section_gap">
    <view class="section__title">slider</view>
    <slider name="slider" show-value ></slider>
  </view>

  <view class="section">
    <view class="section__title">input</view>
    <input name="input" placeholder="please input here" />
  </view>
  <view class="section section_gap">
    <view class="section__title">radio</view>
    <radio-group name="radio-group">
      <label><radio value="radio1"/>radio1</label>
      <label><radio value="radio2"/>radio2</label>
    </radio-group>
  </view>
  <view class="section section_gap">
    <view class="section__title">checkbox</view>
    <checkbox-group name="checkbox">
      <label><checkbox value="checkbox1"/>checkbox1</label>
      <label><checkbox value="checkbox2"/>checkbox2</label>
    </checkbox-group>
  </view>
  <view class="btn-area">
    <button formType="submit">Submit</button>
    <button formType="reset">Reset</button>
  </view>
</form>
```

```
Page({
  formSubmit: function(e) {
    console.log('form发生了submit事件，携带数据为：', e.detail.value)
  },
  formReset: function() {
    console.log('form发生了reset事件')
  }
})
```

form

表单

switch



slider



input

please input here

radio

- radio1
- radio2

checkbox

- checkbox1
- checkbox2

## input

输入框。

属性名	类型	默认值	说明
value	String		输入框的内容
type	String	text	input 的类型，有效值: text, number, idcard, digit, time, date
password	Boolean	false	是否是密码类型
placeholder	String		输入框为空时占位符
placeholder-style	String		指定 placeholder 的样式
placeholder-class	String	input-placeholder	指定 placeholder 的样式类
disabled	Boolean	false	是否禁用
maxlength	Number	140	最大输入长度，设置为0的时候不限制最大长度
auto-focus	Boolean	false	自动聚焦，拉起键盘。页面中只能有一个 <input/> 设置 auto-focus 属性
focus	Boolean	false	获取焦点（开发工具暂不支持）
bindchange	EventHandle		输入框失去焦点时，触发 bindchange 事件，event.detail = {value: value}
bindinput	EventHandle		除了date/time类型外的输入框，当键盘输入时，触发input 事件，event.detail = {value: value}，处理函数可以直接 return 一个字符串，将替换输入框的内容。
bindfocus	EventHandle		输入框聚焦时触发，event.detail = {value: value}
bindblur	EventHandle		输入框失去焦点时触发，event.detail = {value: value}

示例代码：

```
<!--input.wxml-->
<view class="section">
  <input placeholder="这是一个可以自动聚焦的input" auto-focus/>
</view>
<view class="section">
  <input placeholder="这个只有在按钮点击的时候才聚焦" focus="{{focus}}" />
  <view class="btn-area">
    <button bindtap="bindButtonTap">使得输入框获取焦点</button>
  </view>
</view>
<view class="section">
  <input maxlength="10" placeholder="最大输入长度10" />
</view>
<view class="section">
  <view class="section__title">你输入的是: {{inputValue}}</view>
  <input bindinput="bindKeyInput" placeholder="输入同步到view中"/>
</view>
<view class="section">
  <input bindinput="bindReplaceInput" placeholder="连续的两个1会变成2" />
</view>
<view class="section">
  <input bindinput="bindHideKeyboard" placeholder="输入123自动收起键盘" />
</view>
<view class="section">
  <input password type="number" />
</view>
<view class="section">
  <input password type="text" />
</view>
<view class="section">
  <input type="digit" placeholder="带小数点的数字键盘"/>
</view>
<view class="section">
  <input type="idcard" placeholder="身份证输入键盘" />
</view>
<view class="section">
  <input placeholder-style="color:red" placeholder="占位符字体是红色的" />
</view>
```

```
//input.js
Page({
  data: {
    focus: false,
    inputValue: ''
  },
  bindButtonTap: function() {
    this.setData({
      focus: Date.now()
    })
  },
  bindKeyInput: function(e) {
    this.setData({
      inputValue: e.detail.value
    })
  },
  bindReplaceInput: function(e) {
    var value = e.detail.value
    var pos = e.detail.cursor
    if(pos != -1){
      //光标在中间
      var left = e.detail.value.slice(0,pos)
      //计算光标的位置
      pos = left.replace(/11/g,'2').length
    }
    //直接返回对象，可以对输入进行过滤处理，同时可以控制光标的位置
    return {
      value: value.replace(/11/g,'2'),
      cursor: pos
    }
  },
  //或者直接返回字符串，光标在最后边
  //return value.replace(/11/g,'2'),
},
bindHideKeyboard: function(e) {
  if (e.detail.value === '123') {
    //收起键盘
    wx.hideKeyboard()
  }
})
})
```



## label

用来改进表单组件的可用性，使用 `for` 属性找到对应的 `id`，或者将控件放在该标签下，当点击时，就会触发对应的控件。

`for` 优先级高于内部控件，内部有多个控件的时候默认触发第一个控件。

目前可以绑定的控件有：`<button/>`，`<checkbox/>`，`<radio/>`，`<switch/>`。

属性名	类型	说明
<code>for</code>	String	绑定控件的 id

示例代码：

```
<view class="section section_gap">
<view class="section_title">表单组件在label内</view>
<checkbox-group class="group" bindchange="checkboxChange">
  <view class="label-1" wx:for="{{checkboxItems}}>
    <label>
      <checkbox hidden value="{{item.name}}" checked="{{item.checked}}"/></checkbox>
      <view class="label-1__icon">
        <view class="label-1__icon-checked" style="opacity:{{item.checked ? 1 : 0}}"/>
      </view>
      <text class="label-1__text">{{item.value}}</text>
    </label>
  </view>
</checkbox-group>
</view>

<view class="section section_gap">
<view class="section_title">label用for标识表单组件</view>
<radio-group class="group" bindchange="radioChange">
  <view class="label-2" wx:for="{{radioItems}}>
    <radio id="{{item.name}}" hidden value="{{item.name}}" checked="{{item.checked}}"/></radio>
    <view class="label-2__icon">
      <view class="label-2__icon-checked" style="opacity:{{item.checked ? 1 : 0}}"/>
    </view>
    <label class="label-2__text" for="{{item.name}}><text>{{item.name}}</text></label>
  </view>
</radio-group>
</view>
```

```

Page({
  data: {
    checkboxItems: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本', checked: 'true'},
      {name: 'ENG', value: '英国'},
      {name: 'TUR', value: '法国'},
    ],
    radioItems: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本'},
      {name: 'ENG', value: '英国'},
      {name: 'TUR', value: '法国'},
    ],
    hidden: false
  },
  checkboxChange: function(e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.checkboxItems.length; i++) {
      if (checked.indexOf(this.data.checkboxItems[i].name) !== -1) {
        changed['checkboxItems['+i+'].checked'] = true
      } else {
        changed['checkboxItems['+i+'].checked'] = false
      }
    }
    this.setData(changed)
  },
  radioChange: function(e) {
    var checked = e.detail.value
    var changed = {}
    for (var i = 0; i < this.data.radioItems.length; i++) {
      if (checked.indexOf(this.data.radioItems[i].name) !== -1) {
        changed['radioItems['+i+'].checked'] = true
      } else {
        changed['radioItems['+i+'].checked'] = false
      }
    }
    this.setData(changed)
  }
})

```

```
.label-1, .label-2{
  margin-bottom: 15px;
}
.label-1__text, .label-2__text {
  display: inline-block;
  vertical-align: middle;
}

.label-1__icon {
  position: relative;
  margin-right: 10px;
  display: inline-block;
  vertical-align: middle;
  width: 18px;
  height: 18px;
  background: #fcffff;
}

.label-1__icon-checked {
  position: absolute;
  top: 3px;
  left: 3px;
  width: 12px;
  height: 12px;
  background: #1aad19;
}

.label-2__icon {
  position: relative;
  display: inline-block;
  vertical-align: middle;
  margin-right: 10px;
  width: 18px;
  height: 18px;
  background: #fcffff;
  border-radius: 50px;
}

.label-2__icon-checked {
  position: absolute;
  left: 3px;
  top: 3px;
  width: 12px;
  height: 12px;
  background: #1aad19;
  border-radius: 50%;
}

.label-4_text{
  text-align: center;
  margin-top: 15px;
}
```

英国

法国

label用for标识表单组件

USA

CHN

BRA

JPN

ENG

FRA

绑定button

点击这段文字， button会被选中

按钮

label内有多个时选中第一个

选中我  选不中  选不中  选不中

点我会选中第一个

## picker

滚动选择器，现支持三种选择器，通过mode来区分，分别是普通选择器，时间选择器，日期选择器，默认是普通选择器。

普通选择器： mode = selector

属性名	类型	默认值	说明
range	Array	[]	mode为 selector 时， range 有效
value	Number	0	mode为 selector 时，是数字，表示选择了 range 中的第几个，从0开始。
bindchange	EventHandle		value改变时触发change事件， event.detail = {value: value}

时间选择器： mode = time

属性名	类型	默认值	说明
value	String		表示选中的时间，格式为"hh:mm"
start	String		表示有效时间范围的开始，字符串格式为"hh:mm"
end	String		表示有效时间范围的结束，字符串格式为"hh:mm"
bindchange	EventHandle		value改变时触发change事件， event.detail = {value: value}

日期选择器： mode = date

属性名	类型	默认值	说明
value	String	0	表示选中的日期，格式为"yyyy-MM-dd"
start	String		表示有效日期范围的开始，字符串格式为"yyyy-MM-dd"
end	String		表示有效日期范围的结束，字符串格式为"yyyy-MM-dd"
fields	String	day	有效值year,month,day, 表示选择器的粒度
bindchange	EventHandle		value改变时触发change事件， event.detail = {value: value}

注意：开发工具暂时只支持mode = selector。

示例代码：

```

<view class="section">
    <view class="section__title">地区选择器</view>
    <picker bindchange="bindPickerChange" value="{{index}}" range="{{array}}>
        <view class="picker">
            当前选择:
        </view>
    </picker>
</view>
<view class="section">
    <view class="section__title">时间选择器</view>
    <picker mode="time" value="{{time}}" start="09:01" end="21:01" bindchange="bindTimeChange">
        <view class="picker">
            当前选择:
        </view>
    </picker>
</view>

<view class="section">
    <view class="section__title">日期选择器</view>
    <picker mode="date" value="{{date}}" start="2015-09-01" end="2017-09-01" bindchange="bindDateChange">
        <view class="picker">
            当前选择:
        </view>
    </picker>
</view>

```

```

Page({
  data: {
    array: ['美国', '中国', '巴西', '日本'],
    index: 0,
    date: '2016-09-01',
    time: '12:01'
  },
  bindPickerChange: function(e) {
    console.log('picker发送选择改变，携带值为', e.detail.value)
    this.setData({
      index: e.detail.value
    })
  },
  bindDateChange: function(e) {
    this.setData({
      date: e.detail.value
    })
  },
  bindTimeChange: function(e) {
    this.setData({
      time: e.detail.value
    })
  }
})

```

····· 中国联通 10:42 53%



## 演示

...

picker

地区选择器

当前选择：中国

时间选择器

当前选择：12:01

日期选择器

当前选择：2016-09-01

## radio-group

单项选择器，内部由多个 `<radio/>` 组成。

属性名	类型	默认值	说明
bindchange	EventHandle		<code>&lt;radio-group/&gt;</code> 中的选中项发生变化时触发 <code>change</code> 事件， <code>event.detail = {value: 选中项radio的value}</code>

## radio

单选项目

属性名	类型	默认值	说明
value	String		<code>&lt;radio/&gt;</code> 标识。当该 <code>&lt;radio/&gt;</code> 选中时， <code>&lt;radio-group/&gt;</code> 的 <code>change</code> 事件会携带 <code>&lt;radio/&gt;</code> 的 <code>value</code>
checked	Boolean	false	当前是否选中
disabled	Boolean	false	是否禁用

```
<radio-group class="radio-group" bindchange="radioChange">
  <label class="radio" wx:for="{{items}}>
    <radio value="{{item.name}}" checked="{{item.checked}}"/>
  </label>
</radio-group>
```

```
Page({
  data: {
    items: [
      {name: 'USA', value: '美国'},
      {name: 'CHN', value: '中国', checked: 'true'},
      {name: 'BRA', value: '巴西'},
      {name: 'JPN', value: '日本'},
      {name: 'ENG', value: '英国'},
      {name: 'TUR', value: '法国'},
    ],
    radioChange: function(e) {
      console.log('radio发生change事件，携带value值为：', e.detail.value)
    }
})
```

radio

单选框

- 美国
- 中国
- 巴西
- 日本
- 英国
- 法国

## slider

滑动选择器。

属性名	类型	默认值	说明
min	Number	0	最小值
max	Number	100	最大值
step	Number	1	步长，取值必须大于0，并且可被(max - min)整除
disabled	Boolean	false	是否禁用
value	Number	0	当前取值
show-value	Boolean	false	是否显示当前 value
bindchange	EventHandle		完成一次拖动后触发的事件，event.detail = {value: value}

示例代码：

```
<view class="section section_gap">
  <text class="section_title">设置left/right icon</text>
  <view class="body-view">
    <slider bindchange="slider1change" left-icon="cancel" right-icon="success_no_circle"/>
  </view>
</view>

<view class="section section_gap">
  <text class="section_title">设置step</text>
  <view class="body-view">
    <slider bindchange="slider2change" step="5"/>
  </view>
</view>

<view class="section section_gap">
  <text class="section_title">显示当前value</text>
  <view class="body-view">
    <slider bindchange="slider3change" show-value/>
  </view>
</view>

<view class="section section_gap">
  <text class="section_title">设置最小/最大值</text>
  <view class="body-view">
    <slider bindchange="slider4change" min="50" max="200" show-value/>
  </view>
</view>
```

```
var pageData = {}
for (var i = 1; i < 5; i++) {
  (function (index) {
    pageData['slider' + index + 'change'] = function(e) {
      console.log('slider' + 'index' + '发生 change 事件，携带值为', e.detail.value)
    }
  })(i)
}
Page(pageData)
```



slider

设置step



显示当前value



设置最小/最大值



## switch

开关选择器。

属性名	类型	默认值	说明
checked	Boolean	false	是否选中
type	String	switch	样式, 有效值: switch, checkbox
bindchange	EventHandle		checked改变时触发change事件, event.detail={ value:checked}

```
<view class="body-view">
  <switch checked bindchange="switch1Change"/>
  <switch bindchange="switch2Change"/>
</view>
```

```
Page({
  switch1Change: function (e){
    console.log('switch1 发生 change 事件, 携带值为', e.detail.value)
  },
  switch2Change: function (e){
    console.log('switch2 发生 change 事件, 携带值为', e.detail.value)
  }
})
```

····· 中国联通

10:52

④ 51%



演示

...

switch



## action-sheet

从屏幕底部出现的菜单表。

属性名	类型	默认值	说明
bindchange	EventHandle		点击背景或 action-sheet-cancel 按钮时触发 change 事件，不携带数据

## action-sheet-item

底部菜单表的子选项。

## action-sheet-cancel

底部菜单表的取消按钮，和 `<action-sheet-item>` 的区别是，点击它会触发 `<action-sheet>` 的 `change` 事件，并且外观上会同它上面的内容间隔开来。

示例代码：

```
<button type="default" bindtap="actionSheetTap">弹出action sheet</button>
<action-sheet hidden="{{actionSheetHidden}}" bindchange="actionSheetChange">
  <block wx:for="{{actionSheetItems}}">
    <action-sheet-item class="item" bindtap="bindItemTap" data-name="{{item}}></action-sheet-item>
  </block>
  <action-sheet-cancel class="cancel">取消</action-sheet-cancel>
</action-sheet>
```

```
Page({
  data: {
    actionSheetHidden: true,
    actionSheetItems: ['item1', 'item2', 'item3', 'item4']
  },
  actionSheetTap: function(e) {
    this.setData({
      actionSheetHidden: !this.data.actionSheetHidden
    })
  },
  actionSheetChange: function(e) {
    this.setData({
      actionSheetHidden: !this.data.actionSheetHidden
    })
  },
  bindItemTap: function(e) {
    console.log('tap ' + e.currentTarget.dataset.name)
  }
})
```



## 演示

...

action-sheet

弹出action sheet

item1

item2

item3

item4

取消

## modal

模态弹窗。

属性名	类型	默认值	说明
title	String		标题
no-cancel	Boolean	false	是否隐藏 cancel 按钮
confirm-text	String	确定	confirm 按钮文字
cancel-text	String	取消	cancel 按钮文字
bindconfirm	EventHandle		点击 confirm 触发的回调
bindcancel	EventHandle		点击 cancel 以及蒙层触发的回调

示例：

```
<modal title="标题" confirm-text="confirm" cancel-text="cancel" hidden="{{modalHidden}}" bindconfirm="modalChange" bindcancel="modalChange">
    这是对话框的内容。
</modal>

<modal class="modal" hidden="{{modalHidden2}}" no-cancel bindconfirm="modalChange2">
    <view> 内容可以插入节点 </view>
</modal>

<view class="btn-area">
    <button type="default" bindtap="modalTap">点击弹出modal</button>
    <button type="default" bindtap="modalTap2">点击弹出modal2</button>
</view>
```

```
Page({
  data: {
    modalHidden: true,
    modalHidden2: true
  },
  modalTap: function(e) {
    this.setData({
      modalHidden: false
    })
  },
  modalChange: function(e) {
    this.setData({
      modalHidden: true
    })
  },
  modalTap2: function(e) {
    this.setData({
      modalHidden2: false
    })
  },
  modalChange2: function(e) {
    this.setData({
      modalHidden2: true
    })
  },
})
```





点击弹出modal2

标题

这是对话框的内容。

取消

确定

## toast

消息提示框。

属性名	类型	默认值	说明
duration	Float	1500	hidden 设置 false 后，触发 bindchange 的延时，单位毫秒
bindchange	EventHandle		duration 延时后触发

示例代码：

```
<view class="body-view">
  <toast hidden="{{toast1Hidden}}" bindchange="toast1Change">
    默认
  </toast>
  <button type="default" bindtap="toast1Tap">点击弹出默认toast</button>
</view>
<view class="body-view">
  <toast hidden="{{toast2Hidden}}" duration="3000" bindchange="toast2Change">
    设置duration
  </toast>
  <button type="default" bindtap="toast2Tap">点击弹出设置duration的toast</button>
</view>
```

```
var toastNum = 2
var pageData = {}
pageData.data = {}
for(var i = 0; i <= toastNum; ++i) {
  pageData.data['toast'+i+'Hidden'] = true
  ;(function (index) {
    pageData['toast'+index+'Change'] = function(e) {
      var obj = {}
      obj['toast'+index+'Hidden'] = true
      this.setData(obj)
    }
    pageData['toast'+index+'Tap'] = function(e) {
      var obj = {}
      obj['toast'+index+'Hidden'] = false
      this.setData(obj)
    }
  })(i)
}
Page(pageData)
```

.....

中国联通

20:11

④

55%



## 演示

...

toast

点击弹出默认toast

点击弹出设置duration的toast

默认

## loading

加载提示。

示例代码：

```
<view class="body-view">
  <loading hidden="{{hidden}}" bindchange="loadingChange">
    加载中...
  </loading>
  <button type="default" bindtap="loadingTap">点击弹出loading</button>
</view>
```

```
Page({
  data: {
    hidden: true
  },
  loadingChange: function () {
    this.setData({
      hidden: true
    })
  },
  loadingTap: function () {
    this.setData({
      hidden: false
    })

    var that = this
    setTimeout(function () {
      that.setData({
        hidden: true
      })
    }, 1500)
  }
})
```



演示

...

loading

点击弹出loading



加载中...

## navigator

页面链接。

属性名	类型	默认值	说明
url	String		应用内的跳转链接
redirect	Boolean	false	是否关闭当前页面
hover-class	String	navigator-hover	指定点击时的样式类，当 <code>hover-class="none"</code> 时，没有点击态效果

注：`navigator-hover` 默认为 `{background-color: rgba(0, 0, 0, 0.1); opacity: 0.7;}`，`<navigator/>` 的子节点背景色应为透明色

示例代码：

```
/** wxss */
/** 修改默认的navigator点击态 */
.navigator-hover {
    color:blue;
}
/** 自定义其他点击态样式类 */
.other-navigator-hover {
    color:red;
}

<!-- sample.wxml -->
<view class="btn-area">
    <navigator url="navigate?title=navigate" hover-class="navigator-hover">跳转到新页面</navigator>
    <navigator url="redirect?title=redirect" redirect hover-class="other-navigator-hover">在当前页打开</navigator>
</view>

<!-- navigator.wxml -->
<view style="text-align:center"> {{title}} </view>
<view> 点击左上角返回回到之前页面 </view>
```

```
<!-- redirect.wxml -->
<view style="text-align:center"> {{title}} </view>
<view> 点击左上角返回回到上级页面 </view>
```

```
// redirect.js navigator.js
Page({
    onLoad: function(options) {
        this.setData({
            title: options.title
        })
    }
})
```

## audio

音频。

属性名	类型	默认值	说明
action	Object		控制音频的播放、暂停，播放速率、播放进度的对象，有 <code>method</code> 和 <code>data</code> 两个参数
src	String		要播放音频的资源地址
loop	Boolean	false	是否循环播放
controls	Boolean	true	是否显示默认控件
poster	String		默认控件上的音频封面的图片资源地址，如果 <code>controls</code> 属性值为 <code>false</code> 则设置 <code>poster</code> 无效
name	String	未知音頻	默认控件上的音频名字，如果 <code>controls</code> 属性值为 <code>false</code> 则设置 <code>name</code> 无效
author	String	未知作者	默认控件上的作者名字，如果 <code>controls</code> 属性值为 <code>false</code> 则设置 <code>author</code> 无效
binderror	EventHandle		当发生错误时触发 <code>error</code> 事件， <code>detail = {errMsg: MediaError.code}</code>
bindplay	EventHandle		当开始/继续播放时触发 <code>play</code> 事件
bindpause	EventHandle		当暂停播放时触发 <code>pause</code> 事件
bindratechange	EventHandle		当播放速率改变时触发 <code>ratechange</code> 事件
bindtimeupdate	EventHandle		当播放进度改变时触发 <code>timeupdate</code> 事件， <code>detail = {currentTime, duration}</code>
bindended	EventHandle		当播放到末尾时触发 <code>ended</code> 事件

### MediaError.code

返回错误码	描述
MEDIA_ERR_ABORTED	获取资源被用户禁止
MEDIA_ERR_NETWORK	网络错误
MEDIA_ERR_DECODE	解码错误
MEDIA_ERR_SRC_NOT_SUPPORTED	不合适资源

### Action

method	描述	data
play	播放	
pause	暂停	
setPlaybackRate	调整速度	倍速
setCurrentTime	设置当前时间	播放时间

示例代码：

action的method属性只能是 play 、 pause 、 setPlaybackRate 、 setCurrentTime ，用法如下：

```
<!-- 循环播放 -->
<audio poster="{{poster}}" name="{{name}}" author="{{author}}" src="{{src}}" action="{{action}}" controls loop></audio>

<button type="primary" bindtap="audioPlay">播放</button>
<button type="primary" bindtap="audioPause">暂停</button>
<button type="primary" bindtap="audioPlaybackRateSpeedUp">调为2倍速</button>
<button type="primary" bindtap="audioPlaybackRateNormal">调为1倍速</button>
<button type="primary" bindtap="audioPlaybackRateSlowDown">调为0.5倍速</button>
<button type="primary" bindtap="audio14">设置当前播放时间为14秒</button>
<button type="primary" bindtap="audioStart">回到开头</button>
```

```

// app-service
Page({
  data: {
    poster: 'http://y.gtimg.cn/music/photo_new/T002R300x300M000003rsKF44GyaSk.jpg?max_age=2592000',
    name: '此时此刻',
    author: '许巍',
    src: 'http://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?guid=ffffffff82def4af4b12b3cd9337d5e7&uin=346897220&vke
y=6292F51E1E384E06DCBDC9AB7C49FD713D632D313AC4858BACB8DDD29067D3C601481D36E62053BF8DFEA74C0A5CCFADD6471160CAF3E6A&fromt
ag=46',
  },
  audioPlay: function () {
    this.setData({
      action: {
        method: 'play'
      }
    })
  },
  audioPause: function () {
    this.setData({
      action: {
        method: 'pause'
      }
    })
  },
  audioPlaybackRateSpeedUp: function () {
    this.setData({
      action: {
        method: 'setPlaybackRate',
        data: 2
      }
    })
  },
  audioPlaybackRateNormal: function () {
    this.setData({
      action: {
        method: 'setPlaybackRate',
        data: 1
      }
    })
  },
  audioPlaybackRateSlowDown: function () {
    this.setData({
      action: {
        method: 'setPlaybackRate',
        data: 0.5
      }
    })
  },
  audio14: function () {
    this.setData({
      action: {
        method: 'setCurrentTime',
        data: 14
      }
    })
  },
  audioStart: function () {
    this.setData({
      action: {
        method: 'setCurrentTime',
        data: 0
      }
    })
  }
})

```

## audio

---



## image

图片。

属性名	类型	默认值	说明
src	String		图片资源地址
mode	String	'scaleToFill'	图片裁剪、缩放的模式
binderror	HandleEvent		当错误发生时，发布到 AppService 的事件名，事件对象 event.detail = {errMsg: 'something wrong'}
bindload	HandleEvent		当图片载入完毕时，发布到 AppService 的事件名，事件对象 event.detail = {}

注： **image** 组件默认宽度 **300px**、高度 **225px**

**mode** 有 12 种模式，其中 3 种是缩放模式，9 种是裁剪模式。

模式	说明
scaleToFill	不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 元素
aspectFit	保持纵横比缩放图片，使图片的长边能完全显示出来。也就是说，可以完整地将图片显示出来。
aspectFill	保持纵横比缩放图片，只保证图片的短边能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取。

裁剪模式

模式	说明
top	不缩放图片，只显示图片的顶部区域
bottom	不缩放图片，只显示图片的底部区域
center	不缩放图片，只显示图片的中间区域
left	不缩放图片，只显示图片的左边区域
right	不缩放图片，只显示图片的右边区域
top left	不缩放图片，只显示图片的左上边区域
top right	不缩放图片，只显示图片的右上边区域
bottom left	不缩放图片，只显示图片的左下边区域
bottom right	不缩放图片，只显示图片的右下边区域

示例：

```

<view class="page">
  <view class="page_hd">
    <text class="page_title">image</text>
    <text class="page_desc">图片</text>
  </view>
  <view class="page_bd">
    <view class="section section_gap" wx:for="{{array}}" wx:for-item="item">
      <view class="section_title">{{item.text}}</view>
      <view class="section_ctn">
        <image style="width: 200px; height: 200px; background-color: #eeeeee;" mode="{{item.mode}}" src="{{src}}"/>
      </view>
    </view>
  </view>
</view>

```

```

Page({
  data: {
    array: [
      {
        mode: 'scaleToFill',
        text: 'scaleToFill: 不保持纵横比缩放图片，使图片完全适应'
      },
      {
        mode: 'aspectFit',
        text: 'aspectFit: 保持纵横比缩放图片，使图片的长边能完全显示出来'
      },
      {
        mode: 'aspectFill',
        text: 'aspectFill: 保持纵横比缩放图片，只保证图片的短边能完全显示出来'
      },
      {
        mode: 'top',
        text: 'top: 不缩放图片，只显示图片的顶部区域'
      },
      {
        mode: 'bottom',
        text: 'bottom: 不缩放图片，只显示图片的底部区域'
      },
      {
        mode: 'center',
        text: 'center: 不缩放图片，只显示图片的中间区域'
      },
      {
        mode: 'left',
        text: 'left: 不缩放图片，只显示图片的左边区域'
      },
      {
        mode: 'right',
        text: 'right: 不缩放图片，只显示图片的右边区域'
      },
      {
        mode: 'top left',
        text: 'top left: 不缩放图片，只显示图片的左上边区域'
      },
      {
        mode: 'top right',
        text: 'top right: 不缩放图片，只显示图片的右上边区域'
      },
      {
        mode: 'bottom left',
        text: 'bottom left: 不缩放图片，只显示图片的左下边区域'
      },
      {
        mode: 'bottom right',
        text: 'bottom right: 不缩放图片，只显示图片的右下边区域'
      }
    ],
    src: '../../resources/cat.jpg'
  },
  imageError: function(e) {
    console.log('image3发生error事件，携带值为', e.detail errMsg)
  }
})

```

原图



**scaleToFill**

不保持纵横比缩放图片，使图片完全适应



**aspectFit**

保持纵横比缩放图片，使图片的长边能完全显示出来



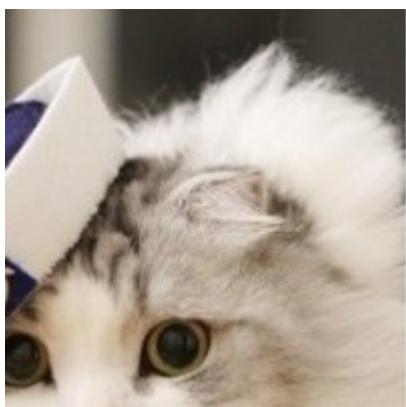
**aspectFill**

保持纵横比缩放图片，只保证图片的短边能完全显示出来



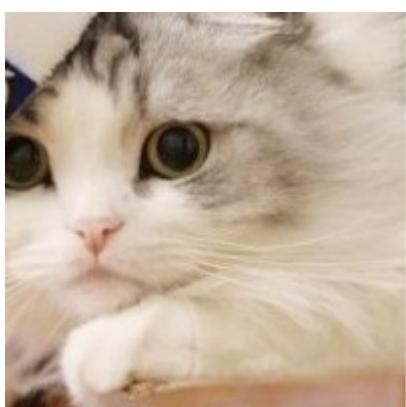
**top**

不缩放图片，只显示图片的顶部区域



**bottom**

不缩放图片，只显示图片的底部区域



**center**

不缩放图片，只显示图片的中间区域



**left**

不缩放图片，只显示图片的左边区域



**right**

不缩放图片，只显示图片的右边边区域



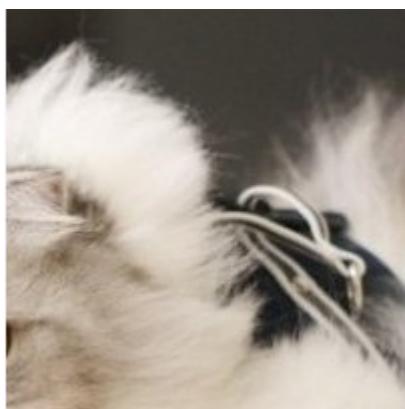
**top left**

不缩放图片，只显示图片的左上边区域



**top right**

不缩放图片，只显示图片的右上边区域



**bottom left**

不缩放图片，只显示图片的左下边区域



**bottom right**

不缩放图片，只显示图片的右下边区域



## video

视频。

属性名	类型	默认值	说明
src	String		要播放视频的资源地址
binderror	EventHandle		当发生错误时触发error事件, event.detail = {errMsg: 'something wrong'}

video标签认宽度300px、高度225px，设置宽高需要通过wxss设置width和height。

示例代码：

```
<view class="section tc">
  <video src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?filekey=30280201010421301f0201690402534804102ca905ce
620b1241b726bc41dcff44e00204012882540400&bizid=1023&hy=SH&fileparam=302c02010104253023020413ffd93020457e3c4ff02024ef202
031e8d7f02030f42400204045a320a0201000400" binderror="videoErrorCallback"></video>
</view>

<view class="section tc">
  <video src="{{src}}></video>
  <view class="btn-area">
    <button bindtap="bindButtonTap">获取视频</button>
  </view>
</view>
```

```
Page({
  data: {
    src: ''
  },
  bindButtonTap: function() {
    var that = this
    wx.chooseVideo({
      sourceType: ['album', 'camera'],
      maxDuration: 60,
      camera: ['front','back'],
      success: function(res) {
        that.setData({
          src: res.tempFilePath
        })
      }
    })
  },
  videoErrorCallback: function(e) {
    console.log('视频错误信息：')
    console.log(e.detail.errMsg)
  }
})
```

video

---



## map

地图。

属性名	类型	默认值	说明
longitude	Number		中心经度
latitude	Number		中心纬度
scale	Number	16	缩放级别
markers	Array		标记点
covers	Array		覆盖物

标记点

标记点用于在地图上显示标记的位置，不能自定义图标和样式

属性	说明	类型	必填	备注
latitude	纬度	Number	是	浮点数，范围 -90 ~ 90
longitude	经度	Number	是	浮点数，范围 -180 ~ 180
name	标注点名	String	是	
desc	标注点详细描述	String	否	

覆盖物

覆盖物用于在地图上显示自定义图标，可自定义图标和样式

属性	说明	类型	必填	备注
latitude	纬度	Number	是	浮点数，范围 -90 ~ 90
longitude	经度	Number	是	浮点数，范围 -180 ~ 180
iconPath	显示的图标	String	是	项目目录下的图片路径，支持相对路径写法
rotate	旋转角度	Number	否	顺时针旋转的角度，范围 0 ~ 360， 默认为 0

地图组件的经纬度必填，如果不填经纬度则默认值是北京的经纬度。标记点**markers**只能在初始化的时候设置，不支持动态更新。

注意：开发工具暂时不支持**map**组件。

示例：

```
<!-- map.wxml -->
<map longitude="113.324520" latitude="23.099994" markers="{{markers}}" covers="{{covers}}" style="width: 375px; height: 200px;"></map>
```

```
// map.js
Page({
  data: {
    markers: [
      {
        latitude: 23.099994,
        longitude: 113.324520,
        name: 'T.I.T 创意园',
        desc: '我现在的位置'
      },
      {
        latitude: 23.099794,
        longitude: 113.324520,
        iconPath: '../images/car.png',
        rotate: 10
      },
      {
        latitude: 23.099298,
        longitude: 113.324129,
        iconPath: '../images/car.png',
        rotate: 90
      }
    ]
  }
})
```

# canvas

画布。

属性名	类型	默认值	说明
canvas-id	String		canvas 组件的唯一标识符
binderror	EventHandle		当发生错误时触发 error 事件, detail = {errMsg: 'something wrong'}

注：

1. **canvas**标签默认宽度**300px**、高度**225px**
2. 同一页面中的**canvas-id**不可重复，如果使用一个已经出现过的**canvas-id**，该**canvas** 标签对应的画布将被隐藏并不会再正常工作

示例代码：[下载](#)

```
<!-- canvas.wxml -->
<canvas style="width: 300px; height: 200px;" canvas-id="firstCanvas"></canvas>
<!- 当使用绝对定位时，文档流后边的canvas的显示层级高于前边的canvas-->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas"></canvas>
<!- 因为canvas-id与前一个canvas重复，该canvas不会显示，并会发送一个错误事件到AppService -->
<canvas style="width: 400px; height: 500px;" canvas-id="secondCanvas" binderror="canvasIdErrorCallback"></canvas>
```

```
// canvas.js
Page({
  canvasIdErrorCallback: function (e) {
    console.error(e.detail.errMsg)
  },
  onReady: function (e) {

    //使用wx.createContext获取绘图上下文context
    var context = wx.createContext()

    context.setStrokeStyle("#00ff00")
    context.setLineWidth(5)
    context.rect(0, 0, 200, 200)
    context.stroke()
    context.setStrokeStyle("#ff0000")
    context.setLineWidth(2)
    context.moveTo(160, 100)
    context.arc(100, 100, 60, 0, 2 * Math.PI, true)
    context.moveTo(140, 100)
    context.arc(100, 100, 40, 0, Math.PI, false)
    context.moveTo(85, 80)
    context.arc(80, 80, 5, 0, 2 * Math.PI, true)
    context.moveTo(125, 80)
    context.arc(120, 80, 5, 0, 2 * Math.PI, true)
    context.stroke()

    //调用wx.drawCanvas，通过canvasId指定在哪张画布上绘制，通过actions指定绘制行为
    wx.drawCanvas({
      canvasId: 'firstCanvas',
      actions: context.getActions() //获取绘图动作数组
    })
  }
})
```

相关api: [wx.createContext](#)、[wx.drawCanvas](#)

# API

框架提供丰富的微信原生API，可以方便的调起微信提供的能力，如获取用户信息，本地存储，支付功能等。

说明：

- `wx.on` 开头的 API 是监听某个事件发生的API接口，接受一个 `CALLBACK` 函数作为参数。当该事件触发时，会调用 `CALLBACK` 函数。
- 如未特殊约定，其他 API 接口都接受一个`OBJECT`作为参数。
- `OBJECT`中可以指定 `success` , `fail` , `complete` 来接收接口调用结果。

参数名	类型	必填	说明
<code>success</code>	<code>Function</code>	否	接口调用成功的回调函数
<code>fail</code>	<code>Function</code>	否	接口调用失败的回调函数
<code>complete</code>	<code>Function</code>	否	接口调用结束的回调函数（调用成功、失败都会执行）

API列表： 网络 API 列表：

API说明	
<code>wx.request</code>	发起网络请求
<code>wx.uploadFile</code>	上传文件
<code>wx.downloadFile</code>	下载文件
<code>wx.connectSocket</code>	创建 WebSocket 连接
<code>wx.onSocketOpen</code>	监听 WebSocket 打开
<code>wx.onSocketError</code>	监听 WebSocket 错误
<code>wx.sendSocketMessage</code>	发送 WebSocket 消息
<code>wx.onSocketMessage</code>	接受 WebSocket 消息
<code>wx.closeSocket</code>	关闭 WebSocket 连接
<code>wx.onSocketClose</code>	监听 WebSocket 关闭

媒体 API 列表： | API说明 | | --- | --- | | `wx.chooseImage` | 从相册选择图片，或者拍照 | | `wx.previewImage` | 预览图片 | | `wx.startRecord` | 开始录音 | | `wx.stopRecord` | 结束录音 | | `wx.playVoice` | 播放语音 | | `wx.pauseVoice` | 暂停播放语音 | | `wx.stopVoice` | 结束播放语音 | | `wx.getBackgroundAudioPlayerState` | 获取音乐播放状态 | | `wx.playBackgroundAudio` | 播放音乐 | | `wx.pauseBackgroundAudio` | 暂停播放音乐 | | `wx.seekBackgroundAudio` | 控制音乐播放进度 | | `wx.stopBackgroundAudio` | 停止播放音乐 | | `wx.onBackgroundAudioPlay` | 监听音乐开始播放 | | `wx.onBackgroundAudioPause` | 监听音乐暂停 | | `wx.onBackgroundAudioStop` | 监听音乐结束 | | `wx.chooseVideo` | 从相册选择视频，或者拍摄 | | `wx.saveFile` | 保存文件 |

数据 API 列表：

API说明	
wx.getStorage	获取本地数据缓存
wx.setStorage	设置本地数据缓存
wx.clearStorage	清理本地数据缓存

位置 API 列表：

API说明	
wx.getLocation	获取当前位置
wx.openLocation	打开内置地图

设备 API 列表：

API说明	
wx.getNetworkType	获取网络类型
wx.getSystemInfo	获取系统信息
wx.onAccelerometerChange	监听重力感应数据
wx.onCompassChange	监听罗盘数据

界面 API 列表：

API说明	
wx.setNavigationBarTitle	设置当前页面标题
wx.showNavigationBarLoading	显示导航条加载动画
wx.hideNavigationBarLoading	隐藏导航条加载动画
wx.navigateTo	新窗口打开页面
wx.redirectTo	原窗口打开页面
wx.navigateBack	退回上一个页面
wx.createAnimation	动画
wx.createContext	创建绘图上下文
wx.drawCanvas	绘图
wx.hideKeyboard	隐藏键盘
wx.stopPullDownRefresh	停止下拉刷新动画

开放接口：

API说明	
wx.login	登录
wx.getUserInfo	获取用户信息
wx.requestPayment	发起微信支付

每个微信小程序需要事先设置一个通讯域名，小程序可以跟指定的域名与进行网络通信。包括普通 HTTPS 请求（`wx.request`）、WebSocket 通信（`wx.connectSocket`）、上传文件（`wx.uploadFile`）和下载文件（`wx.downloadFile`）。

网络**API**列表：

API说明	
<code>wx.request</code>	发起网络请求
<code>wx.uploadFile</code>	上传文件
<code>wx.request</code>	发起网络请求
<code>wx.uploadFile</code>	上传文件
<code>wx.downloadFile</code>	下载文件
<code>wx.connectSocket</code>	创建 WebSocket 连接
<code>wx.onSocketOpen</code>	监听 WebSocket 打开
<code>wx.onSocketError</code>	监听 WebSocket 错误
<code>wx.sendSocketMessage</code>	发送 WebSocket 消息
<code>wx.onSocketMessage</code>	接受 WebSocket 消息
<code>wx.closeSocket</code>	关闭 WebSocket 连接
<code>wx.onSocketClose</code>	监听 WebSocket 关闭

## wx.request(OBJECT)

`wx.request` 发起的是https请求。一个微信小程序，同时只能有5个网络请求连接。

**OBJECT**参数说明：

参数名	类型	必填	说明
url	String	是	开发者服务器接口地址
data	Object、String	否	请求的参数
header	Object	否	设置请求的 header , header 中不能设置 Referer
method	String	否	默认为 GET，有效值：OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
success	Function	否	收到开发者服务成功返回的回调函数， <code>res = {data: '开发者服务器返回的内容'}</code>
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.request({
  url: 'test.php',
  data: {
    x: '',
    y: ''
  },
  header: {
    'Content-Type': 'application/json'
  },
  success: function(res) {
    console.log(res.data)
  }
})
```

## wx.uploadFile(OBJECT)

将本地资源上传到开发者服务器。如页面通过 `wx.chooseImage` 等接口获取到一个本地资源的临时文件路径后，可通过此接口将本地资源上传到指定服务器。客户端发起一个 **HTTPS POST** 请求，其中 `Content-Type` 为 `multipart/form-data`。

**OBJECT**参数说明：

参数	类型	必填	说明
url	String	是	开发者服务器 url
filePath	String	是	要上传文件资源的路径
name	String	是	文件对应的 key，开发者在服务器端通过这个 key 可以获取到文件二进制内容
header	Object	否	HTTP 请求 Header，header 中不能设置 Referer
formData	Object	否	HTTP 请求中其他额外的 form data
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.chooseImage({
  success: function(res) {
    var tempFilePaths = res.tempFilePaths
    wx.uploadFile({
      url: 'http://example.com/upload',
      filePath: tempFilePaths[0],
      name: 'file',
      formData:{
        'user': 'test'
      }
    })
  }
})
```

## wx.downloadFile(OBJECT)

下载文件资源到本地。客户端直接发起一个 **HTTP GET** 请求，把下载到的资源根据 `type` 进行处理，并返回文件的本地临时路径。

**OBJECT**参数说明：

参数	类型	必填	说明
url	String	是	下载资源的 url
type	String	否	下载资源的类型，用于客户端识别处理，有效值：image/audio/video
header	Object	否	HTTP 请求 Header
success	Function	否	下载成功后以 tempFilePath 的形式传给页面，res = {tempFilePath: '文件的临时路径'}
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

注：文件的临时路径，在小程序本次启动期间可以正常使用，如需持久保存，需在主动调用 `**wx.saveFile**`，在小程序下次启动时才能访问得到。

示例代码：

```
wx.downloadFile({
  url: 'http://example.com/audio/123',
  type: 'audio',
  success: function(res) {
    wx.playVoice({
      filePath: res.tempFilePath
    })
  }
})
```

## wx.connectSocket(OBJECT)

创建一个 [WebSocket](#) 连接；一个微信小程序同时只能有一个 **WebSocket** 连接，如果当前已存在一个 **WebSocket** 连接，会自动关闭该连接，并重新创建一个 **WebSocket** 连接。

**OBJECT**参数说明：

参数	类型	必填	说明
url	String	是	开发者服务器接口地址，必须是 HTTPS 协议，且域名必须是后台配置的合法域名
data	Object	否	请求的数据
header	Object	否	HTTP Header , header 中不能设置 Referer
method	String	否	默认是GET，有效值为： OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.connectSocket({
  url: 'test.php',
  data:{
    x: '',
    y: ''
  },
  header:{
    'content-type': 'application/json'
  },
  method:"GET"
})
```

## wx.onSocketOpen(CALLBACK)

监听WebSocket连接打开事件。

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})
wx.onSocketOpen(function(res) {
  console.log('WebSocket连接已打开！')
})
```

## wx.onSocketError(CALLBACK)

监听WebSocket错误。

示例代码：

```

wx.connectSocket({
  url: 'test.php'
})
wx.onSocketOpen(function(res){
  console.log('WebSocket连接已打开! ')
})
wx.onSocketError(function(res){
  console.log('WebSocket连接打开失败, 请检查! ')
})

```

## wx.sendSocketMessage(OBJECT)

通过 WebSocket 连接发送数据，需要先 `wx.connectSocket`，并在 `wx.onSocketOpen` 回调之后才能发送。

**OBJECT**参数说明：

参数	类型	必填	说明
<code>data</code>	<code>String</code>	是	需要发送的内容
<code>success</code>	<code>Function</code>	否	接口调用成功的回调函数
<code>fail</code>	<code>Function</code>	否	接口调用失败的回调函数
<code>complete</code>	<code>Function</code>	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```

var socketOpen = false
var socketMsgQueue = []
wx.connectSocket({
  url: 'test.php'
})

wx.onSocketOpen(function(res) {
  socketOpen = true
  for (var i = 0; i < socketMsgQueue.length; i++){
    sendSocketMessage(socketMsgQueue[i])
  }
  socketMsgQueue = []
})

function sendSocketMessage(msg) {
  if (socketOpen) {
    wx.sendSocketMessage({
      data:msg
    })
  } else {
    socketMsgQueue.push(msg)
  }
}

```

## wx.onSocketMessage(CALLBACK)

监听WebSocket接受到服务器的消息事件。

**CALLBACK**返回参数：

参数	类型	说明
<code>data</code>	<code>String</code>	服务器返回的消息

示例代码：

```
wx.connectSocket({
  url: 'test.php'
})

wx.onSocketMessage(function(res) {
  console.log('收到服务器内容: ' + res.data)
})
```

## wx.closeSocket()

关闭WebSocket连接。

## wx.onSocketClose(CALLBACK)

监听WebSocket关闭。

```
wx.connectSocket({
  url: 'test.php'
})

//注意这里有时序问题,
//如果 wx.connectSocket 还没回调 wx.onSocketOpen, 而先调用 wx.closeSocket, 那么就做不到关闭 WebSocket 的目的。
//必须在 WebSocket 打开期间调用 wx.closeSocket 才能关闭。
wx.onSocketOpen(function() {
  wx.closeSocket()
})

wx.onSocketClose(function(res) {
  console.log('WebSocket 已关闭!')
})
```

## wx.chooseImage(OBJECT)

从本地相册选择图片或使用相机拍照。

**OBJECT**参数说明：

参数	类型	必填	说明
count	Number	否	最多可以选择的图片张数，默认9
sizeType	StringArray	否	original 原图, compressed 压缩图, 默认二者都有
sourceType	StringArray	否	album 从相册选图, camera 使用相机, 默认二者都有
success	Function	是	成功则返回图片的本地文件路径列表 tempFilePaths
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

注：文件的临时路径，在小程序本次启动期间可以正常使用，如需持久保存，需在主动调用 [wx.saveFile](#)，在小程序下次启动时才能访问得到。

示例代码：

```
wx.chooseImage({
  count: 1, // 默认9
  sizeType: ['original', 'compressed'], // 可以指定是原图还是压缩图, 默认二者都有
  sourceType: ['album', 'camera'], // 可以指定来源是相册还是相机, 默认二者都有
  success: function (res) {
    // 返回选定照片的本地文件路径列表, tempFilePath可以作为img标签的src属性显示图片
    var tempFilePaths = res.tempFilePaths
  }
})
```

## wx.previewImage(OBJECT)

预览图片。

**OBJECT**参数说明：

参数	类型	必填	说明
current	String	否	当前显示图片的链接, 不填则默认为 urls 的第一张
urls	StringArray	是	需要预览的图片链接列表
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

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

## wx.startRecord(OBJECT)

开始录音。当主动调用 `wx.stopRecord`，或者录音超过1分钟时自动结束录音，返回录音文件的临时文件路径。

**OBJECT**参数说明：

参数	类型	必填	说明
success	Function	否	录音成功后调用，返回录音文件的临时文件路径， <code>res = {tempFilePath:'录音文件的临时路径'}</code>
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

注：文件的临时路径，在小程序本次启动期间可以正常使用，如需持久保存，需在主动调用 `wx.saveFile`，在小程序下次启动时才能访问得到。

## wx.stopRecord()

主动调用停止录音。

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
  },
  fail: function(res) {
    //录音失败
  }
})
setTimeout(function() {
  //结束录音
  wx.stopRecord()
}, 10000)
```

## wx.playVoice(OBJECT)

开始播放语音，同时只允许一个语音文件正在播放，如果前一个语音文件还没播放完，将中断前一个语音播放。

**OBJECT**参数说明：

参数	类型	必填	说明
filePath	String	是	需要播放的语音文件的文件路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath,
      complete: function(){
        }
    })
  }
})
```

## wx.pauseVoice()

暂停正在播放的语音。再次调用wx.playVoice播放同一个文件时，会从暂停处开始播放。如果想从头开始播放，需要先调用wx.stopVoice。

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath: tempFilePath
    })

    setTimeout(function() {
      //暂停播放
      wx.pauseVoice()
    }, 5000)
  }
})
```

## wx.stopVoice()

结束播放语音。

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.playVoice({
      filePath:tempFilePath
    })

    setTimeout(function(){
      wx.stopVoice()
    }, 5000)
  }
})
```

## wx.getBackgroundAudioPlayerState(OBJECT)

获取音乐播放状态。

**OBJECT**参数说明：

参数	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

**success**返回参数说明：

参数	说明
duration	选定音频的长度（单位：s），只有在当前有音乐播放时返回
currentPosition	选定音频的播放位置（单位：s），只有在当前有音乐播放时返回
status	播放状态（2：没有音乐在播放，1：播放中，0：暂停中）
downloadPercent	音频的下载进度（整数，80 代表 80%），只有在当前有音乐播放时返回
dataUrl	歌曲数据链接，只有在当前有音乐播放时返回

示例代码：

```
wx.getBackgroundAudioPlayerState({
  success: function(res) {
    var status = res.status
    var dataUrl = res.dataUrl
    var currentPosition = res.currentPosition
    var duration = res.duration
    var downloadPercent = res.downloadPercent
  }
})
```

## wx.playBackgroundAudio(OBJECT)

播放音乐，同时只能有一首音乐正在播放。

**OBJECT**参数说明

参数	类型	必填	说明
dataUrl	String	是	音乐链接
title	String	否	音乐标题
coverImgUrl	String	否	封面URL
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码

```
wx.playBackgroundAudio({
  dataUrl: '',
  title: '',
  coverImgUrl: ''
})
```

## wx.pauseBackgroundAudio()

暂停播放音乐。

示例代码

```
wx.pauseBackgroundAudio()
```

## wx.seekBackgroundAudio(OBJECT)

控制音乐播放进度。

OBJECT参数说明

参数	类型	必填	说明
position	Number	是	音乐位置，单位：秒
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码

```
wx.seekBackgroundAudio({
  position: 30
})
```

## wx.stopBackgroundAudio()

停止播放音乐。

示例代码

```
wx.stopBackgroundAudio()
```

## wx.onBackgroundAudioPlay(CALLBACK)

监听音乐播放。

## wx.onBackgroundAudioPause(CALLBACK)

监听音乐暂停。

## wx.onBackgroundAudioStop(CALLBACK)

监听音乐停止。

## wx.saveFile(OBJECT)

保存文件到本地。

**OBJECT**参数说明：

参数	类型	必填	说明
tempFilePath	String	是	需要保存的文件的临时路径
success	Function	否	返回文件的保存路径, <code>res = {savedFilePath: '文件的保存路径'}</code>
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.startRecord({
  success: function(res) {
    var tempFilePath = res.tempFilePath
    wx.saveFile({
      tempFilePath: tempFilePath,
      success: function(res) {
        var savedFilePath = res.savedFilePath
      }
    })
  }
})
```

## wx.chooseVideo(OBJECT)

拍摄视频或从手机相册中选视频，返回视频的临时文件路径。

**OBJECT**参数说明：

参数	类型	必填	说明
sourceType	StringArray	否	album 从相册选视频，camera 使用相机拍摄，默认为：['album', 'camera']
maxDuration	Number	否	拍摄视频最长拍摄时间，单位秒。最长支持60秒
camera	StringArray	否	前置或者后置摄像头，默认为前后都有，即：['front', 'back']
success	Function	否	接口调用成功，返回视频文件的临时文件路径，详见返回参数说明
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

返回参数说明：

参数	说明
tempFilePath	选定视频的临时文件路径
duration	选定视频的时间长度
size	选定视频的数据量大小
height	返回选定视频的长
width	返回选定视频的宽

注：文件的临时路径，在小程序本次启动期间可以正常使用，如需持久保存，需在主动调用 **\*\*wx.saveFile\*\***，在小程序下次启动时才能访问得到。

示例代码：

```
<view class="container">
  <video src="{{src}}></video>
  <button bindtap="bindButtonTap">获取视频</button>
</view>
```

```
Page({
  bindButtonTap: function() {
    var that = this
    wx.chooseVideo({
      sourceType: ['album','camera'],
      maxDuration: 60,
      camera: ['front','back'],
      success: function(res) {
        that.setData({
          src: res.tempFilePath
        })
      }
    })
  }
})
```

每个微信小程序都可以有自己的本地缓存，可以通过 `wx.setStorage` (`wx.setStorageSync`)、`wx.getStorage` (`wx.getStorageSync`)、`wx.clearStorage` (`wx.clearStorageSync`) 可以对本地缓存进行设置、获取和清理。本地缓存最大为10MB。

注意： `localStorage` 是永久存储的，但是我们不建议将关键信息全部存在 `localStorage`，以防用户换设备的情况。

## wx.setStorage(OBJECT)

将数据存储在本地缓存中指定的 `key` 中，会覆盖掉原来该 `key` 对应的内容，这是一个异步接口。

**OBJECT**参数说明：

参数	类型	必填	说明
<code>key</code>	<code>String</code>	是	本地缓存中的指定的 <code>key</code>
<code>data</code>	<code>Object/String</code>	是	需要存储的内容
<code>success</code>	<code>Function</code>	否	接口调用成功的回调函数
<code>fail</code>	<code>Function</code>	否	接口调用失败的回调函数
<code>complete</code>	<code>Function</code>	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码

```
wx.setStorage({  
  key:"key"  
  data:"value"  
})
```

## wx.setStorageSync(KEY,DATA)

将 `data` 存储在本地缓存中指定的 `key` 中，会覆盖掉原来该 `key` 对应的内容，这是一个同步接口。

参数说明：

参数	类型	必填	说明
<code>key</code>	<code>String</code>	是	本地缓存中的指定的 <code>key</code>
<code>data</code>	<code>Object/String</code>	是	需要存储的内容

示例代码

```
try {  
  wx.setStorageSync('key', 'value')  
} catch (e) {  
}
```

## wx.getStorage(OBJECT)

从本地缓存中异步获取指定 `key` 对应的内容。

**OBJECT**参数说明：

参数	类型	必填	说明
<code>key</code>	<code>String</code>	是	本地缓存中的指定的 <code>key</code>
<code>success</code>	<code>Function</code>	是	接口调用的回调函数, <code>res = {data: key对应的内容}</code>
<code>fail</code>	<code>Function</code>	否	接口调用失败的回调函数
<code>complete</code>	<code>Function</code>	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.getStorage({
  key: 'key',
  success: function(res) {
    console.log(res.data)
  }
})
```

## wx.getStorageSync(KEY)

从本地缓存中同步获取指定 `key` 对应的内容。

参数说明：

参数	类型	必填	说明
<code>key</code>	<code>String</code>	是	本地缓存中的指定的 <code>key</code>

示例代码：

```
var value = wx.getStorageSync('key')

if (value) {
  // Do something with return value
}
```

## wx.clearStorage()

清理本地数据缓存。

示例代码：

```
wx.clearStorage()
```

## wx.clearStorageSync()

## 同步清理本地数据缓存

示例代码：

```
try {
    wx.clearStorageSync()
} catch(e) {
}
```

## wx.getLocation(OBJECT)

获取当前的地理位置、速度。

**OBJECT**参数说明：

参数	类型	必填	说明
type	String	否	默认为 wgs84 返回 gps 坐标, gcj02 返回可用于 wx.openLocation 的坐标
success	Function	是	接口调用成功的回调函数, 返回内容详见返回参数说明。
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数(调用成功、失败都会执行)

**success**返回参数说明：

参数	说明
latitude	纬度, 浮点数, 范围为-90~90, 负数表示南纬
longitude	经度, 浮点数, 范围为-180~180, 负数表示西经
speed	速度, 浮点数, 单位m/s
accuracy	位置的精确度

示例代码：

```
wx.getLocation({
  type: 'wgs84',
  success: function(res) {
    var latitude = res.latitude
    var longitude = res.longitude
    var speed = res.speed
    var accuracy = res.accuracy
  }
})
```

## wx.openLocation(OBJECT)

使用微信内置地图查看位置

**OBJECT**参数说明：

参数	类型	必填	说明
latitude	Float	是	纬度，范围为-90~90，负数表示南纬
longitude	Float	是	经度，范围为-180~180，负数表示西经
scale	INT	否	缩放比例，范围1~28，默认为28
name	String	否	位置名
address	String	否	地址的详细说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.getLocation({
  type: 'gcj02', //返回可以用于wx.openLocation的经纬度
  success: function(res) {
    var latitude = res.latitude
    var longitude = res.longitude
    wx.openLocation({
      latitude: latitude,
      longitude: longitude,
      scale: 28
    })
  }
})
```

## wx.getNetworkType(OBJECT)

获取网络类型。

**OBJECT**参数说明：

参数	类型	必填	说明
success	Function	是	接口调用成功，返回网络类型 networkType
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

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

## wx.getSystemInfo(OBJECT)

获取系统信息。

**OBJECT**参数说明：

参数	类型	必填	说明
success	Function	是	接口调用成功的回调
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

**success**回调参数说明：

属性	说明
model	手机型号
pixelRatio	设备像素比
windowWidth	窗口宽度
windowHeight	窗口高度
language	微信设置的语言
version	微信版本号

示例代码：

```
wx.getSystemInfo({
  success: function(res) {
    console.log(res.model)
    console.log(res.pixelRatio)
    console.log(res.windowWidth)
    console.log(res.windowHeight)
    console.log(res.language)
    console.log(res.version)
  }
})
```

## wx.onAccelerometerChange(CALLBACK)

监听重力感应数据，频率：5次/秒

**CALLBACK**返回参数：

参数	类型	说明
x	Number	X轴
y	Number	Y轴
z	Number	Z轴

示例代码：

```
wx.onAccelerometerChange(function(res) {
  console.log(res.x)
  console.log(res.y)
  console.log(res.z)
})
```

## wx.onCompassChange(CALLBACK)

监听罗盘数据，频率：5次/秒

**CALLBACK**返回参数：

参数	类型	说明
direction	Number	面对的方向度数

示例代码：

```
wx.onCompassChange(function (res) {
  console.log(res.direction)
})
```

## **wx.setNavigationBarTitle(OBJECT)**

动态设置当前页面的标题。

**OBJECT**参数说明：

参数	类型	必填	说明
title	String	否	页面标题
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.setNavigationBarTitle({  
  title: '当前页面'  
})
```

## **wx.showNavigationBarLoading()**

在当前页面显示导航条加载动画。

## **wx.hideNavigationBarLoading()**

隐藏导航条加载动画。

## wx.navigateTo(OBJECT)

保留当前页面，跳转到应用内的某个页面，使用 `wx.navigateBack` 可以返回到原页面。

**OBJECT**参数说明：

参数	类型	必填	说明
url	String	是	需要跳转的应用内页面的路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.navigateTo({  
  url: 'test?id=1'  
})
```

注意：为了不让用户在使用小程序时造成困扰，我们规定页面路径只能是五层，请尽量避免多层级的交互方式。

## wx.redirectTo(OBJECT)

关闭当前页面，跳转到应用内的某个页面。

**OBJECT**参数说明：

参数	类型	必填	说明
url	String	是	需要跳转的应用内页面的路径
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.redirectTo({  
  url: 'test?id=1'  
})
```

## wx.navigateBack()

关闭当前页面，回退前一页面。

## wx.createAnimation(OBJECT)

创建一个动画实例`animation`。调用实例的方法来描述动画。最后通过动画实例的 `export` 方法导出动画数据传递给组件的 `animation` 属性。

注意：`export` 方法每次调用后会清掉之前的动画操作

**OBJECT**参数说明：

参数	类型	必填	说明
duration	Integer	否	动画持续时间，单位ms，默认值 400
timingFunction	String	否	定义动画的效果，默认值"linear"，有效值："linear","ease","ease-in","ease-in-out","ease-out","step-start","step-end"
delay	Integer	否	动画延迟时间，单位 ms， 默认值 0
transformOrigin	String	否	设置transform-origin，默认为"50% 50% 0"

```
var animation = wx.createAnimation({
  transformOrigin: "50% 50%",
  duration: 1000,
  timingFunction: "ease",
  delay: 0
})
```

## animation

动画实例可以调用以下方法来描述动画，调用结束后会返回自身，支持链式调用的写法。

样式：

方法	参数	说明
opacity	value	透明度，参数范围 0~1
backgroundColor	color	颜色值
width	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
height	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
top	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
left	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
bottom	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值
right	length	长度值，如果传入 Number 则默认使用 px，可传入其他自定义单位的长度值

旋转：

方法	参数	说明
rotate	deg	deg的范围-180~180, 从原点顺时针旋转一个deg角度
rotateX	deg	deg的范围-180~180, 在X轴旋转一个deg角度
rotateY	deg	deg的范围-180~180, 在Y轴旋转一个deg角度
rotateZ	deg	deg的范围-180~180, 在Z轴旋转一个deg角度
rotate3d	(x,y,z,deg)	同 <a href="#">transform-function rotate3d</a>

缩放:

方法	参数	说明
scale	sx,[sy]	一个参数时, 表示在X轴、Y轴同时缩放sx倍数; 两个参数时表示在X轴缩放sx倍数, 在Y轴缩放sy倍数
scaleX	sx	在X轴缩放sx倍数
scaleY	sy	在Y轴缩放sy倍数
scaleZ	sz	在Z轴缩放sz倍数
scale3d	(sx,sy,sz)	在X轴缩放sx倍数, 在Y轴缩放sy倍数, 在Z轴缩放sz倍数

偏移:

方法	参数	说明
translate	tx,[ty]	一个参数时, 表示在X轴偏移tx, 单位px; 两个参数时, 表示在X轴偏移tx, 在Y轴偏移ty, 单位px。
translateX	tx	在X轴偏移tx, 单位px
translateY	ty	在Y轴偏移tx, 单位px
translateZ	tz	在Z轴偏移tx, 单位px
translate3d	(tx,ty,tz)	在X轴偏移tx, 在Y轴偏移ty, 在Z轴偏移tz, 单位px

倾斜:

方法	参数	说明
skew	ax,[ay]	参数范围-180~180; 一个参数时, Y轴坐标不变, X轴坐标延顺时针倾斜ax度; 两个参数时, 分别在X轴倾斜ax度, 在Y轴倾斜ay度
skewX	ax	参数范围-180~180; Y轴坐标不变, X轴坐标延顺时针倾斜ax度
skewY	ay	参数范围-180~180; X轴坐标不变, Y轴坐标延顺时针倾斜ay度

矩阵变形:

方法	参数	说明
matrix	(a,b,c,d,tx,ty)	同 <a href="#">transform-function matrix</a>
matrix3d		同 <a href="#">transform-function matrix3d</a>

动画队列

调用动画操作方法后要调用 `step()` 来表示一组动画完成，可以在一组动画中调用任意多个动画方法，一组动画中的所有动画会同时开始，一组动画完成后才会进行下一组动画。`step` 可以传入一个跟 `wx.createAnimation()` 一样的配置参数用于指定当前组动画的配置。

示例：

```
<view animation="{{animationData}}" style="background:red;height:100rpx;width:100rpx"></view>
```

```
Page({
  data: {
    animationData: {}
  },
  onShow: function(){
    var animation = wx.createAnimation({
      duration: 1000,
      timingFunction: 'ease',
    })

    this.animation = animation

    animation.scale(2,2).rotate(45).step()

    this.setData({
      animationData:animation.export()
    })

    setTimeout(function() {
      animation.translate(30).step()
      this.setData({
        animationData:animation.export()
      })
    }.bind(this), 1000)
  },
  rotateAndScale: function () {
    // 旋转同时放大
    this.animation.rotate(45).scale(2, 2).step()
    this.setData({
      animationData: this.animation.export()
    })
  },
  rotateThenScale: function () {
    // 先旋转后放大
    this.animation.rotate(45).step()
    this.animation.scale(2, 2).step()
    this.setData({
      animationData: this.animation.export()
    })
  },
  rotateAndScaleThenTranslate: function () {
    // 先旋转同时放大，然后平移
    this.animation.rotate(45).scale(2, 2).step()
    this.animation.translate(100, 100).step({ duration: 1000 })
    this.setData({
      animationData: this.animation.export()
    })
  }
})
```

## wx.createContext()

创建并返回绘图上下文 `context` 对象。

### context

`context` 只是一个记录方法调用的容器，用于生成记录绘制行为的 `actions` 数组。`context` 跟 `<canvas/>` 不存在对应关系，一个`context`生成画布的绘制动作数组可以应用于多个 `<canvas/>`。

```
// 假设页面上有3个画布
var canvas1Id = 3001
var canvas2Id = 3002
var canvas3Id = 3003

var context = wx.createContext();

[canvas1Id, canvas2Id, canvas3Id].forEach(function (id) {
  context.clearActions()
  // 在context上调用方法
  wx.drawCanvas({
    canvasId: id,
    actions: context.getActions()
  })
})
```

`context`对象的方法列表：

方法	参数	说明
<code>getActions</code>	无	获取当前 <code>context</code> 上存储的绘图动作
<code>clearActions</code>	无	清空当前的存储绘图动作
变形		
<code>scale</code>		对纵横坐标进行缩放
<code>rotate</code>		对坐标轴进行顺时针旋转
<code>translate</code>		对坐标原点进行缩放
<code>save</code>	无	保存当前坐标轴的缩放、旋转、平移信息
<code>restore</code>	无	恢复之前保存过的坐标轴的缩放、旋转、平移信息
绘制		
<code>clearRect</code>		在给定的矩形区域内，清除画布上的像素
<code>fillText</code>		在画布上绘制被填充的文本
<code>drawImage</code>		在画布上绘制图像
<code>fill</code>	无	对当前路径进行填充
<code>stroke</code>	无	对当前路径进行描边
路径		
<code>beginPath</code>	无	开始一个路径
<code>closePath</code>	无	关闭一个路径
<code>moveTo</code>		把路径移动到画布中的指定点，但不创建线条。
<code>lineTo</code>		添加一个新点，然后在画布中创建从该点到最后指定点的线条。
<code>rect</code>		添加一个矩形路径到当前路径。
<code>arc</code>		添加一个弧形路径到当前路径，顺时针绘制。
<code>quadraticCurveTo</code>		创建二次方贝塞尔曲线
<code>bezierCurveTo</code>		创建三次方贝塞尔曲线
样式		
<code>setFillStyle</code>		设置填充样式
<code>setStrokeStyle</code>		设置线条样式
<code>setShadow</code>		设置阴影
<code>setFont</code>		设置字体大小
<code>setLineCap</code>		设置线条端点的样式
<code>setLineJoin</code>		设置两线相交处的样式
<code>setLineWidth</code>		设置线条宽度
<code>setMiterLimit</code>		设置最大倾斜

方法详解：

## scale

在调用 `scale` 方法后，之后创建的路径其横纵坐标会被缩放。多次调用 `scale`，倍数会相乘。

参数	类型	范围	说明
<code>scaleWidth</code>	Number	<code>1 = 100%</code> , <code>0.5 = 50%</code> , <code>2 = 200%</code> , 依次类推	横坐标缩放的倍数
<code>scaleHeight</code>	Number	<code>1 = 100%</code> , <code>0.5 = 50%</code> , <code>2 = 200%</code> , 依次类推	纵坐标轴缩放的倍数

示例代码：\*\*[下载](#)\*\*

```
<!--scale.wxml -->
<canvas canvas-id="1"/>
```

```
//scale.js
Page({
  onReady: function(e) {
    var context = wx.createContext()
    context.rect(5, 5, 25, 15)
    context.stroke()
    context.scale(2, 2) //再放大2倍
    context.rect(5, 5, 25, 15)
    context.stroke()
    context.scale(2, 2) //再放大2倍
    context.rect(5, 5, 25, 15)
    context.stroke()
    wx.drawCanvas({
      canvasId: 1
      actions: context.getActions()
    })
  }
})
```

●●○○○ 中国移动

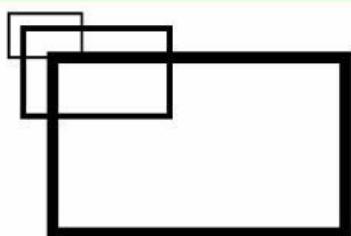
18:54

33%



canvas

•••



## rotate

以原点为中心，原点可以用 [translate](#)方法修改。顺时针旋转当前坐标轴。多次调用 `rotate`，旋转的角度会叠加。

参数	类型	范围	说明
<code>rotate</code>	Number	<code>degrees * Math.PI/180;</code> <code>degrees</code> 范围为0~360	旋转角度，以弧度计

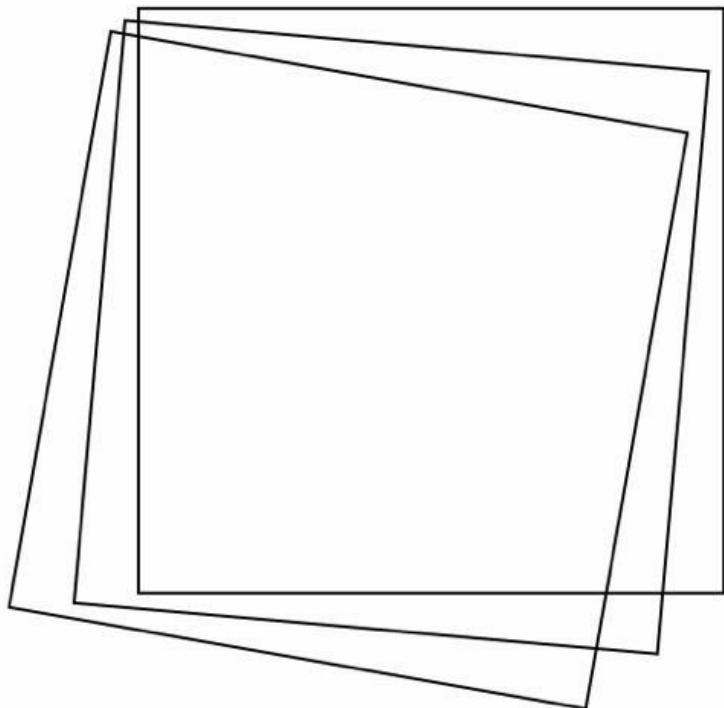
示例代码：[下载](#)

```
//rotate.js
Page({
  onReady: function(e) {
    var context = wx.createContext()
    context.rect(50, 50, 200, 200)
    context.stroke()
    context.rotate(5 * Math.PI / 180)
    context.rect(50, 50, 200, 200)
    context.stroke()
    context.rotate(5 * Math.PI / 180)
    context.rect(50, 50, 200, 200)
    context.stroke()

    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    })
  }
})
```



## canvas



## translate

对当前坐标系的原点(0, 0)进行变换， 默认的坐标系原点为页面左上角。

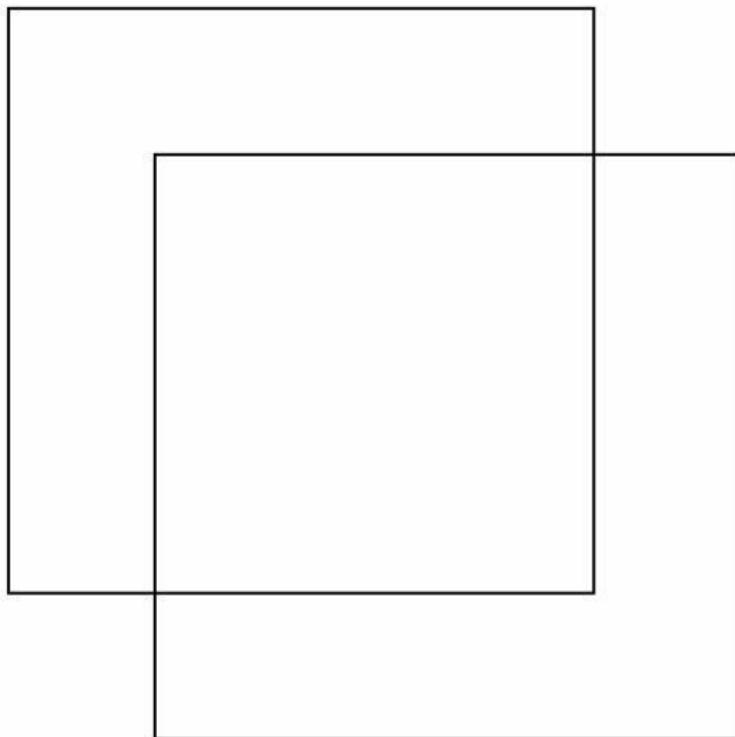
参数	类型	范围	说明
x	Number		水平坐标平移量
y	Number		竖直坐标平移量

示例代码： [下载](#)

```
//translate.js
Page({
  onReady: function() {
    var context = wx.createContext()

    context.rect(50, 50, 200, 200)
    context.stroke()
    context.translate(50, 50)
    context.rect(50, 50, 200, 200)
    context.stroke()

    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    })
  }
})
```



## clearRect

清除画布上在该矩形区域内的内容。

参数	类型	范围	说明
x	Number		矩形区域左上角的x坐标
y	Number		矩形区域左上角的y坐标
width	Number		矩形区域的宽度
height	Number		矩形区域的高度

示例代码：[下载](#)

```
//clearrect.js
Page({
  onReady: function() {
    var context = wx.createContext()

    context.rect(50, 50, 200, 200)
    context.fill()
    context.clearRect(100, 100, 50, 50)

    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    })
  }
})
```

●●○○○ 中国移动

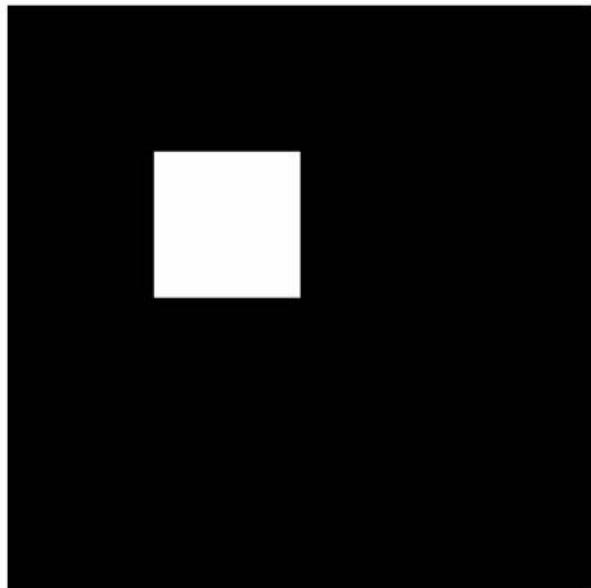
18:54

32%



canvas

•••



## drawImage

绘制图像，图像保持原始尺寸。

参数	类型	范围	说明
imageResource	String	通过 <code>chooseImage</code> 得到一个文件路径或者一个项目目录内的图片	所要绘制的图片资源
x	Number		图像左上角的x坐标
y	Number		图像左上角的y坐标
width	Number		图像宽度
height	Number		图像高度

示例： [下载](#)

```
//drawimage.js
Page({
  onReady: function(e) {
    var context = wx.createContext()
    wx.chooseImage({
      success: function(res) {
        context.drawImage(res.tempFilePaths[0], 0, 0)
        wx.drawCanvas({
          canvasId: 1,
          actions: context.getActions()
        })
      }
    })
  }
})
```

●○○○○ 中国移动

18:57

32%



canvas

...

●●○○○ 中国移动

15:

9月11日

## fillText

在画布上绘制被填充的文本。

参数	类型	范围	说明
text	String		在画布上输出的文本
x	Number		绘制文本的左上角x坐标位置
y	Number		绘制文本的左上角y坐标位置

示例代码：[下载](#)

```
//filltext.js
Page({
  onReady:function(){
    var context = wx.createContext()

    context.setFontSize(14)
    context.fillText("MINA", 50, 50)
    context.moveTo(0, 50)
    context.lineTo(100, 50)
    context.stroke()

    context.setFontSize(20)
    context.fillText("MINA", 100, 100)
    context.moveTo(0, 100)
    context.lineTo(200, 100)
    context.stroke()
    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    });
  }
})
```

●○○○○ 中国移动

18:57

32%



canvas

•••

MINA

MINA

## **beginPath**

开始创建一个路径，需要调用 `fill` 或者 `stroke` 才会使用路径进行填充或描边。同一个路径内的多次 `setFillStyle`、`setStrokeStyle`、`setLineWidth` 等设置，以最后一次设置为准。

## **closePage**

关闭一个路径。

## **moveTo**

把路径移动到画布中的指定点，不创建线条。

参数	类型	范围	说明
x	Number		目标位置的x坐标
y	Number		目标位置的y坐标

## **lineTo**

在当前位置添加一个新点，然后在画布中创建从该点到最后指定点的路径。

参数	类型	范围	说明
x	Number		目标位置的x坐标
y	Number		目标位置的y坐标

## **rect**

添加一个矩形路径到当前路径。

参数	类型	范围	说明
x	Number		矩形路径左上角的x坐标
y	Number		矩形路径左上角的y坐标
width	Number		矩形路径的宽度
height	Number		矩形路径的高度

## **arc**

添加一个弧形路径到当前路径，顺时针绘制。

参数	类型	范围	说明
x	Number		矩形路径左上角的x坐标
y	Number		矩形路径左上角的y坐标
radius	Number		矩形路径的宽度
startAngle	Number	弧度, 0到2pi	起始弧度
sweepAngle	Number	弧度, 0到2pi	从起始弧度开始，扫过的弧度

## **quadraticCurveTo**

创建二次贝塞尔曲线路径。

参数	类型	范围	说明
cpx	Number		贝塞尔控制点的x坐标
cpy	Number		贝塞尔控制点的y坐标
x	Number		结束点的x坐标
y	Number		结束点的y坐标

### bezierCurveTo

创建三次方贝塞尔曲线路径。

参数	类型	范围	说明
cp1x	Number		第一个贝塞尔控制点的 x 坐标
cp1y	Number		第一个贝塞尔控制点的 y 坐标
cp2x	Number		第二个贝塞尔控制点的 x 坐标
cp2y	Number		第二个贝塞尔控制点的 y 坐标
x	Number		结束点的 x 坐标
y	Number		结束点的 y 坐标

### setFillStyle

设置纯色填充。

设置颜色为填充样式

参数	类型	范围	说明
color	String	'rgb(255, 0, 0)'或'rgba(255, 0, 0, 0.6)'或'#ff0000'格式的颜色字符串	设置为填充样式的颜色

### setStrokeStyle

设置纯色描边，参数同 [setFillStyle](#)。

示例代码：[下载](#)

```
//setfillstyle.js
Page({
  onReady: function(e) {
    var context = wx.createContext()

    context.setFillStyle("#ff00ff")
    context.setStrokeStyle("#00ffff")

    context.rect(50, 50, 100, 100)
    context.fill()
    context.stroke()
    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    });
  }
})
```

●●○○○ 中国移动

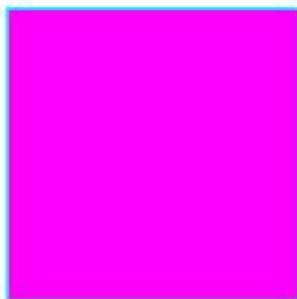
18:57

32%



canvas

•••



### **setShadow**

设置阴影样式。

参数	类型	范围	说明
offsetX	Number		阴影相对于形状在水平方向的偏移
offsetY	Number		阴影相对于形状在竖直方向的偏移
blur	Number	0~100	阴影的模糊级别，数值越大越模糊
color	Color	'rgb(255, 0, 0)'或'rgba(255, 0, 0, 0.6)'或'#ff0000'格式的颜色字符串	阴影的颜色

### **setFontsize**

设置字体的字号。

参数	类型	范围	说明
fontSize	Number		字体的字号

### **setLineWidth**

设置线条的宽度。

参数	类型	范围	说明
lineWidth	Number		线条的宽度

### **setLineCap**

设置线条的结束端点样式。

参数	类型	范围	说明
lineCap	String	'butt'、'round'、'square'	线条的结束端点样式

### **setLineJoin**

设置两条线相交时，所创建的拐角类型。

参数	类型	范围	说明
lineJoin	String	'bevel'、'round'、'miter'	两条线相交时，所创建的拐角类型

### **setMiterLimit**

设置最大斜接长度，斜接长度指的是在两条线交汇处内角和外角之间的距离。当 `setLineJoin` 为 `miter` 时才有效。超过最大倾斜长度的，连接处将以 `lineJoin` 为 `bevel` 来显示

参数	类型	范围	说明
miterLimit	Number		最大斜接长度

示例代码：[下载](#)

```
//line.js
Page({
  onReady: function(e) {
    var context = wx.createContext()

    context.setLineWidth(10)
    context.setLineCap("round")
    context.setLineJoin("miter")
    context.setMiterLimit(10)
    context.moveTo(20, 20)
    context.lineTo(150, 27)
    context.lineTo(20, 54)
    context.stroke()

    context.beginPath()

    context.setMiterLimit(3)
    context.moveTo(20, 70)
    context.lineTo(150, 77)
    context.lineTo(20, 104)
    context.stroke()

    wx.drawCanvas({
      canvasId: 1,
      actions: context.getActions()
    });
  }
})
```

●●○○○ 中国移动

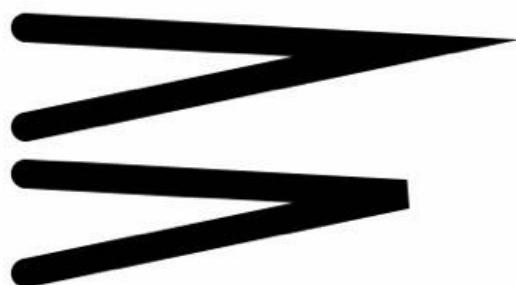
18:57

32%



canvas

•••



## wx.drawCanvas(OBJECT)

OBJECT参数说明：

参数	类型	必填	说明
canvasId	String	是	画布标识，传入 <code>&lt;canvas/&gt;</code> 的 canvas-id
actions	Array	是	绘图动作数组，由 <code>wx.createContext</code> 创建的 context，调用 <code>getActions</code> 方法导出绘图动作数组。

示例： [下载](#)

```
<!--canvas.wxml-->
<canvas cavas-id="firstCanvas"/>

// index.js
Page({
  canvasIdErrorCallback: function (e) {
    console.error(e.detailerrMsg)
  },
  onReady: function(e) {
    //使用wx.createContext获取绘图上下文context
    var context = wx.createContext()

    context.setStrokeStyle("#00ff00")
    context.setLineWidth(5)
    context.rect(0, 0, 200, 200)
    context.stroke()
    context.setStrokeStyle("#ff0000")
    context.setLineWidth(2)
    context.moveTo(160, 100)
    context.arc(100, 100, 60, 0, 2 * Math.PI, true)
    context.moveTo(140, 100)
    context.arc(100, 100, 40, 0, Math.PI, false)
    context.moveTo(85, 80)
    context.arc(80, 80, 5, 0, 2 * Math.PI, true)
    context.moveTo(125, 80)
    context.arc(120, 80, 5, 0, 2 * Math.PI, true)
    context.stroke()

    // 调用 wx.drawCanvas，通过 canvasId 指定在哪张画布上绘制，通过 actions 指定绘制行为
    wx.drawCanvas({
      canvasId: 'firstCanvas',
      actions: context.getActions() // 获取绘图动作数组
    })
  }
})
```

## **wx.hideKeyboard()**

收起键盘。

## **wx.stopPullDownRefresh()**

停止当前页面下拉刷新。详见[页面相关事件处理函数](#)。

## wx.login(OBJECT)

调用接口获取登录凭证（**code**）进而换取用户登录态信息，包括用户的唯一标识（**openid**）及本次登录的会话密钥（**session\_key**）。用户数据的加解密通讯需要依赖会话密钥完成。

**OBJECT**参数说明：

参数名	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

**success**返回参数说明：

参数名	类型	说明
errMsg	String	调用结果
code	String	用户允许登录后，回调内容会带上 <b>code</b> （有效期五分钟），开发者需要将 <b>code</b> 发送到开发者服务器后台，使用 <b>code</b> 换取 <b>session_key</b> api，将 <b>code</b> 换成 <b>openid</b> 和 <b>session_key</b>

示例代码：

```
//app.js
App({
  onLaunch: function() {
    wx.login({
      success: function(res) {
        if (res.code) {
          //发起网络请求
          wx.request({
            url: 'https://test.com/onLogin',
            data: {
              code: res.code
            }
          })
        } else {
          console.log('获取用户登录态失败！' + res.errMsg)
        }
      }
    });
  }
})
```

## code 换取 session\_key

这是一个 HTTP 接口，开发者服务器使用登录凭证 **code** 获取 **session\_key** 和 **openid**。其中 **session\_key** 是对用户数据进行[加密签名](#)的密钥。为了自身应用安全，**session\_key** 不应该在网络上传输。

接口地址：

```
https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JS CODE&grant_type=authorization_code
```

请求参数：

参数	必填	说明
appid	是	小程序唯一标识
secret	是	小程序的 app secret
js_code	是	登录时获取的 code
grant_type	是	填写为 authorization_code

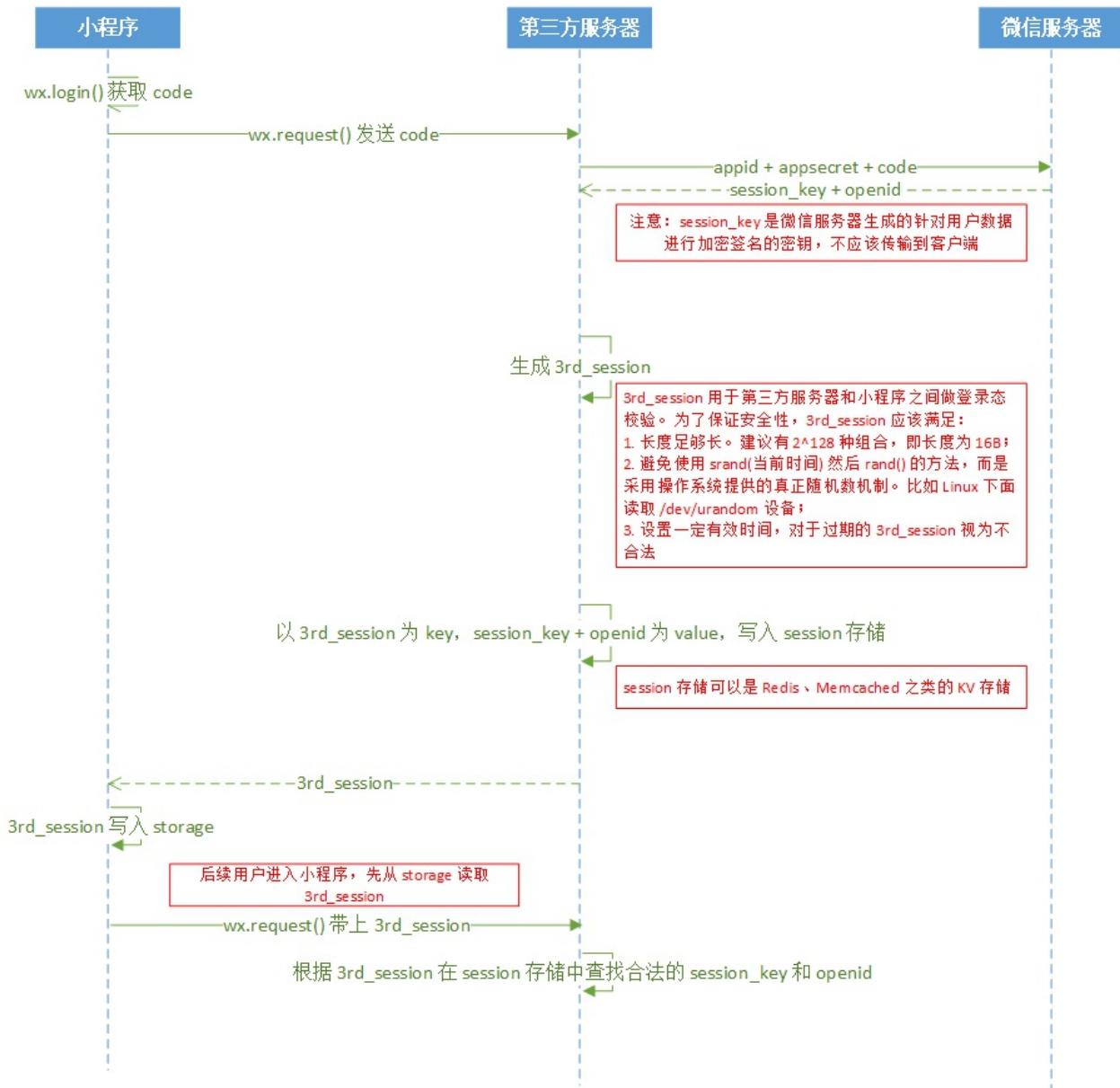
返回参数：

参数	说明
openid	用户唯一标识
session_key	会话密钥
expires_in	会话有效期, 以秒为单位, 例如2592000代表会话有效期为30天

返回说明：

```
//正常返回的JSON数据包
{
    "openid": "OPENID",
    "session_key": "SESSIONKEY"
    "expires_in": 2592000
}
//错误时返回JSON数据包(示例为Code无效)
{
    "errcode": 40029,
    "errmsg": "invalid code"
}
```

## 登录时序图



## 签名校验

1. 使用 `wx.login` 获取 code
2. 通过 code 换取用户的 session-key 和 openid
3. 需要页面自己去做登录态管理（如在 cookie 设置 openid）
4. 通过调用接口（如 `wx.getUserInfo`）获取敏感数据时，接口会同时返回 rawData、signature，其中 `signature = sha1(rawData + session_key)`
5. 将 signature、rawData、以及用户登录态发送给开发者服务器，开发者在数据库中找到该用户对应的 session-key，使用相同的算法计算出签名 signature2，比对 signature 与 signature2 即可校验数据的可信度。

如`wx.getUserInfo`的数据校验：

接口返回的 rawData:

```
{  
  "nickName": "Band",  
  "gender": 1,  
  "language": "zh_CN",  
  "city": "Guangzhou",  
  "province": "Guangdong",  
  "country": "CN",  
  "avatarUrl": "http://wx.qlogo.cn/mmopen/vi_32/1vZvI39NWFQ9XM4LtQpFrQJ1xlgZxx3w7bQxKARo16503Iuswjjn6nIGBiaycAjAtpujxyzY  
srztuuICqIM5ibXQ/0"  
}
```

用户的 session-key:

```
HyVfkG15F50QWJZZaNzBBg==
```

所以，用于签名的字符串为：

```
{  
  "nickName": "Band",  
  "gender": 1,  
  "language": "zh_CN",  
  "city": "Guangzhou",  
  "province": "Guangdong",  
  "country": "CN",  
  "avatarUrl": "http://wx.qlogo.cn/mmopen/vi_32/1vZvI39NWFQ9XM4LtQpFrQJ1xlgZxx3w7bQxKARo16503Iuswjjn6nIGBiaycAjAtpujxyzY  
srztuuICqIM5ibXQ/0"}HyVfkG15F50QWJZZaNzBBg=="
```

使用sha1得到的结果为

```
75e81ceda165f4ffa64f4068af58c64b8f54b88c
```

## 加密数据解密算法

接口如果涉及敏感数据（如 `wx.getUserInfo` 当中的 `openid`），接口的明文内容将不包含敏感数据。开发者如需要获取敏感数据，需要对接口返回的加密数据(`encryptData`)进行对称解密。解密算法如下：

1. 对称解密使用的算法为 AES-128-CBC，数据采用PKCS#7填充。

2. 对称解密的目标密文为 `Base64_Decode(encryptData)`,
3. 对称解密秘钥 `aeskey = Base64_Decode(session_key)`, `aeskey` 是16字节
4. 对称解密算法初始向量 `iv = aeskey`, 同样是16字节

## wx.getUserInfo(OBJECT)

获取用户信息，需要先调用 [wx.login](#) 接口。

**OBJECT**参数说明：

参数名	类型	必填	说明
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

**success**返回参数说明：

参数	类型	说明
userInfo	OBJECT	用户信息对象，不包含 openid 等敏感信息
rawData	String	不包括敏感信息的原始数据字符串，用于计算签名。
signature	String	使用 <code>sha1(rawData + sessionkey)</code> 得到字符串，用于校验用户信息。
encryptData	String	包括敏感数据在内的完整用户信息的加密数据，详见 <a href="#">加密数据解密算法</a>

示例代码：

```
wx.getUserInfo({
  success: function(res) {
    var userInfo = res.userInfo
    var nickName = userInfo.nickName
    var avatarUrl = userInfo.avatarUrl
    var gender = userInfo.gender //性别 0: 未知、1: 男、2: 女
    var province = userInfo.province
    var city = userInfo.city
    var country = userInfo.country
  }
})
```

**encryptData** 解密后为以下 json 结构，详见[加密数据解密算法](#)

```
{
  "openId": "OPENID",
  "nickName": "NICKNAME",
  "gender": 1,
  "city": "CITY",
  "province": "PROVINCE",
  "country": "COUNTRY",
  "avatarUrl": "AVATARURL",
  "unionId": "UNIONID"
}
```

## UnionID机制说明：

如果开发者拥有多个移动应用、网站应用、和公众帐号（包括小程序），可通过unionid来区分用户的唯一性，因为只要是同一个微信开放平台帐号下的移动应用、网站应用和公众帐号（包括小程序），用户的unionid是唯一的。换句话说，同一用户，对同一个微信开放平台下的不同应用，unionid是相同的。

微信开放平台绑定小程序流程

前提：微信开放平台帐号必须已完成开发者资质认证

开发者资质认证流程：

登录微信开放平台([open.weixin.qq.com](http://open.weixin.qq.com)) – 帐号中心 – 开发者资质认证

The screenshot shows the 'Developer Certification' section of the WeChat Open Platform. On the left sidebar, 'Developer Certification' is highlighted in green. The main area displays the status as 'Not Certified' with a red exclamation mark. A summary box contains the following information:

**Introduction**

- WeChat Open Platform developer certification provides more secure and strict authenticity verification, better protecting enterprises and users' legal rights.
- After developer certification, applications under the WeChat Open Platform account will obtain WeChat login, intelligent interface, and official account third-party platform development capabilities.
- Certification validity period: one year, valid for two months before expiration to apply for annual review to renew.
- Review fee: 300 RMB.

At the top right, there are 'View Order' and 'Apply Now' buttons.

绑定流程：

登录微信开放平台 ([open.weixin.qq.com](http://open.weixin.qq.com)) — 管理中心 — 公众帐号 — 绑定公众帐号

The screenshot shows the 'Bind Public Account' section of the WeChat Open Platform management center. The 'Public Account' tab is selected. At the top, there are buttons for 'Bind Public Account' (green), 'Bind Test Account' (grey), and a note indicating 9 more accounts can be bound. Below this, a table lists public accounts with columns for 'Account Type' and 'Operation'.

## wx.requestPayment(OBJECT)

发起微信支付。

**Object**参数说明：

参数	类型	必填	说明
timeStamp	DateInt	是	时间戳从1970年1月1日00:00:00至今的秒数,即当前的时间
nonceStr	String	是	随机字符串，长度为32个字符以下。
package	String	是	统一下单接口返回的 prepay_id 参数值，提交格式如： prepay_id=*
signType	String	是	签名算法，暂支持 MD5
paySign	String	是	签名,具体签名方案参见 <a href="#">微信公众号支付帮助文档</a> ;
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

示例代码：

```
wx.requestPayment({
  'timeStamp': '',
  'nonceStr': '',
  'package': '',
  'signType': 'MD5',
  'paySign': '',
  'success':function(res){
  },
  'fail':function(res){
  }
})
```

基于微信的通知渠道，我们为开发者提供了可以高效触达用户的模板消息能力，以便实现服务的闭环并提供更佳的体验。

模板推送位置：服务通知

模板下发条件：用户本人在微信体系内与页面有交互行为后触发，详见[下发条件说明](#)

模板跳转能力：点击查看详情仅能跳转下发模板的该帐号的各个页面

## 使用说明

### 1. 获取模板 id

登录<https://mp.weixin.qq.com>获取模板，如果没有合适的模板，可以申请添加新模板，审核通过后可使用，详见[模板审核说明](#)

The screenshot shows the 'Template Message' management interface in the WeChat MP Platform. On the left sidebar, under 'Template Message', the 'My Templates' tab is selected. The main area displays a table titled 'Personal Template Library' with four rows of template entries:

标题	关键词	模板ID	操作
购买成功通知	购买地点、价格	nqLSWNP-W64gkqRN...	复制 详情 删除
购买成功通知	购买地点、购买时间、...	nqLSWNP-W64gkqRN...	复制 详情 删除
续费成功通知	金额、续费类型、到期...	54jkbpeoZ9ngv1CohN...	复制 详情 删除
酒店预订成功通知	酒店名称、地点、金额...	khOICmNE3Bc2eVBeg...	复制 详情 删除

1. 页面的 `<form/>` 组件，属性 `report-submit` 为 `true` 时，可以声明为需发模板消息，此时点击按钮提交表单可以获取 `formId`，用于发送模板消息。或者当用户完成[支付行为](#)，可以获取 `prepay_id` 用于发送模板消息。
2. 调用接口下发模板消息（详见[接口说明](#)）

## 接口说明

### 1. 获取 `access_token`

`access_token` 是全局唯一接口调用凭据，开发者调用各接口时都需使用 `access_token`，请妥善保存。`access_token` 的存储至少要保留512个字符空间。`access_token` 的有效期目前为2个小时，需定时刷新，重复获取将导致上次获取的 `access_token` 失效。

公众平台的 API 调用所需的 `access_token` 的使用及生成方式说明：

1. 为了保密 appsecret, 第三方需要一个 `access_token` 获取和刷新的中控服务器。而其他业务逻辑服务器所使用的 `access_token` 均来自于该中控服务器, 不应该各自去刷新, 否则会造成 `access_token` 覆盖而影响业务;
2. 目前 `access_token` 的有效期通过返回的 `expires_in` 来传达, 目前是7200秒之内的值。中控服务器需要根据这个有效时间提前去刷新新 `access_token`。在刷新过程中, 中控服务器对外输出的依然是老 `access_token`, 此时公众平台后台会保证在刷新短时间内, 新老 `access_token` 都可用, 这保证了第三方业务的平滑过渡;
3. `access_token` 的有效时间可能会在未来有调整, 所以中控服务器不仅需要内部定时主动刷新, 还需要提供被动刷新 `access_token` 的接口, 这样便于业务服务器在 API 调用获知 `access_token` 已超时的情况下, 可以触发 `access_token` 的刷新流程。

开发者可以使用 `AppID` 和 `AppSecret` 调用本接口来获取 `access_token`。`AppID` 和 `AppSecret` 可登录微信公众平台官网-设置-开发设置中获得（需要已经绑定成为开发者，且帐号没有异常状态）。`AppSecret` 生成后请自行保存，因为在公众平台每次生成查看都会导致 `AppSecret` 被重置。注意调用所有微信接口时均需使用 `https` 协议。如果第三方不使用中控服务器，而是选择各个业务逻辑点各自去刷新 `access_token`，那么就可能会产生冲突，导致服务不稳定。

接口地址：

```
https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&appid=APPID&secret=APPSECRET
```

**HTTP请求方式：**

GET

参数说明：

参数	必填	说明
<code>grant_type</code>	是	获取 <code>access_token</code> 填写 <code>client_credential</code>
<code>appid</code>	是	第三方用户唯一凭证
<code>secret</code>	是	第三方用户唯一凭证密钥, 即 <code>appsecret</code>

返回参数说明：

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
<code>access_token</code>	获取到的凭证
<code>expires_in</code>	凭证有效时间, 单位: 秒

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"}
```

## 2. 发送模板消息

接口地址：(**ACCESS\_TOKEN** 需换成上文获取到的 `access_token`)

```
https://api.weixin.qq.com/cgi-bin/message/wxopen/template/send?access_token=ACCESS_TOKEN
```

**HTTP请求方式：**

POST

#### POST参数说明：

参数	必填	说明
touser	是	接收者（用户）的 openid
template_id	是	所需下发的模板消息的id
page	否	点击模板查看详情跳转页面，不填则模板无跳转
form_id	是	表单提交场景下，为 submit 事件带上的 formId；支付场景下，为本次支付的 prepay_id
value	是	模板内容，不填则下发空模板
color	否	模板内容字体的颜色，不填默认黑色
emphasis_keyword	否	模板需要放大的关键词，不填则默认无放大

示例：

```
{  
    "touser": "OPENID",  
    "template_id": "TEMPLATE_ID",  
    "page": "index",  
    "form_id": "FORMID",  
    "data": {  
        "keyword1": {  
            "value": "339208499",  
            "color": "#173177"  
        },  
        "keyword2": {  
            "value": "2015年01月05日 12:30",  
            "color": "#173177"  
        },  
        "keyword3": {  
            "value": "粤海喜来登酒店",  
            "color": "#173177"  
        },  
        "keyword4": {  
            "value": "广州市天河区天河路208号",  
            "color": "#173177"  
        }  
    },  
    "emphasis_keyword": "keyword1.DATA"  
}
```

返回码说明：

在调用模板消息接口后，会返回JSON数据包。

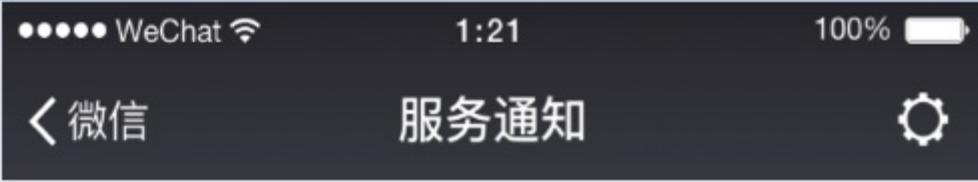
正常时的返回JSON数据包示例：

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

错误时会返回错误码信息，说明如下：

返回码	说明
40037	template_id不正确
41028	form_id不正确，或者过期
41029	form_id已被使用
41030	page不正确

使用效果：



下午 4:16



酒店预订助手

...

## 酒店预订成功通知

4月27日 16:16

订单号

339208499

时间 2015年01月05日 12:30

酒店 粤海喜来登酒店

地点 广州市天河区天河路208号

查看详情 >

注意：内部测试阶段，模板消息下发后，在客户端仅能看到由“公众号安全助手”下发的简单通知。能收到该提示，即表明模板消息功能已经调试成功。待该功能正式上线后，将可以展示成上图效果。

下发条件说明

## 1. 支付

当用户在小程序内完成过支付行为，可允许开发者向用户在7天内推送有限条数的模板消息（1次支付可下发1条，多次支付下发条数独立，互相不影响）

## 2. 提交表单

当用户在小程序内发生过提交表单行为且该表单声明为要发模板消息的，开发者需要向用户提供服务时，可允许开发者向用户在7天内推送有限条数的模板消息（1次提交表单可下发1条，多次提交下发条数独立，相互不影响）

# 审核说明

## 1. 标题

1.1 标题不能存在相同

1.2 标题意思不能存在过度相似

1.3 标题必须以“提醒”或“通知”结尾

1.4 标题不能带特殊符号、个性化字词等没有行业通用性的内容

1.5 标题必须能体现具体服务场景

1.6 标题不能涉及营销相关内容，包括但不限于：

消费优惠类、购物返利类、商品更新类、优惠券类、代金券类、红包类、会员卡类、积分类、活动类等营销倾向通知

## 2. 关键词

2.1 同一标题下，关键词不能存在相同

2.2 同一标题下，关键词不能存在过度相似

2.3 关键词不能带特殊符号、个性化字词等没有行业通用性的内容

2.4 关键词内容示例必须与关键词对应匹配

2.5 关键词不能太过宽泛，需要具有限制性，例如：“内容”这个就太宽泛，不能审核通过

# 违规说明

除不能违反运营规范外，还不能违反以下规则，包括但不限于：

1. 不允许恶意诱导用户进行触发操作，以达到可向用户下发模板目的
2. 不允许恶意骚扰，下发对用户造成骚扰的模板
3. 不允许恶意营销，下发营销目的模板
4. 不允许通过服务号下发模板来告知用户在小程序内触发的服务相关内容

# 处罚说明

根据违规情况给予相应梯度的处罚，一般处罚规则如下：

第一次违规，删除违规模板以示警告，

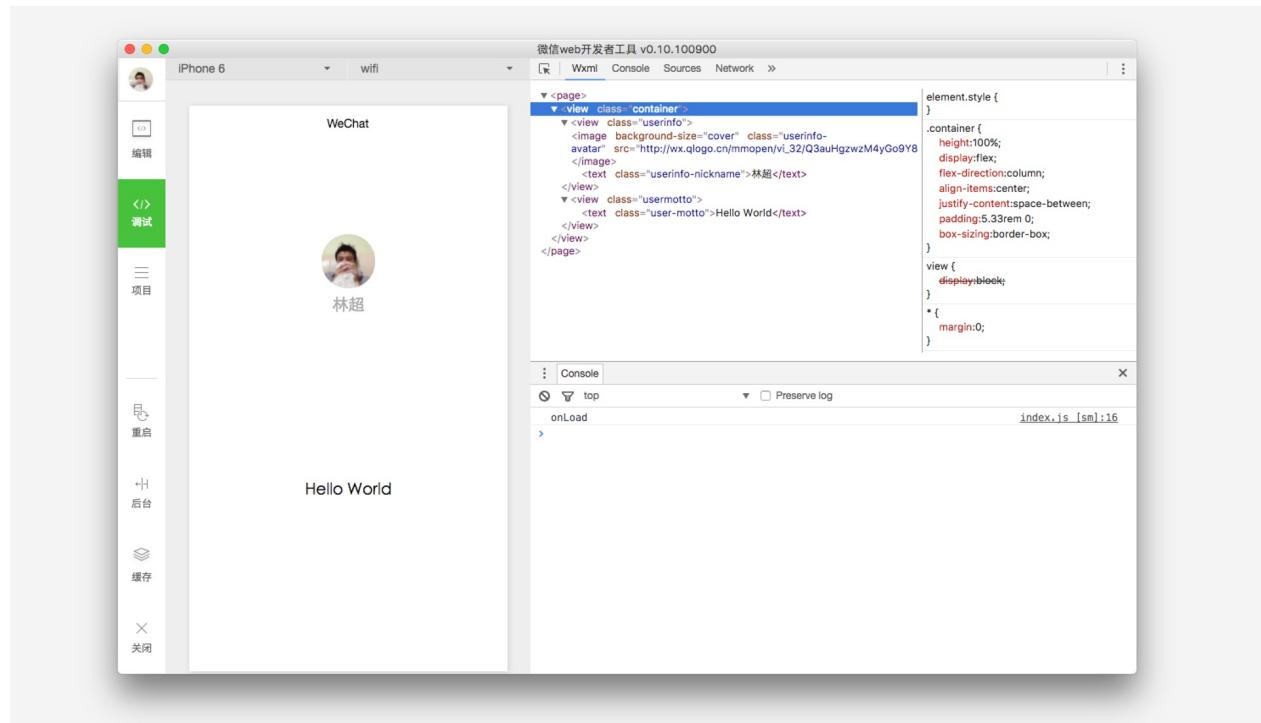
第二次违规，封禁接口7天，

第三次违规，封禁接口30天，

第四次违规，永久封禁接口

处罚结果及原因以站内信形式告知

为了帮助开发者简单和高效地开发微信小程序，我们推出了全新的开发者工具，集成了开发调试、代码编辑及程序发布等功能。



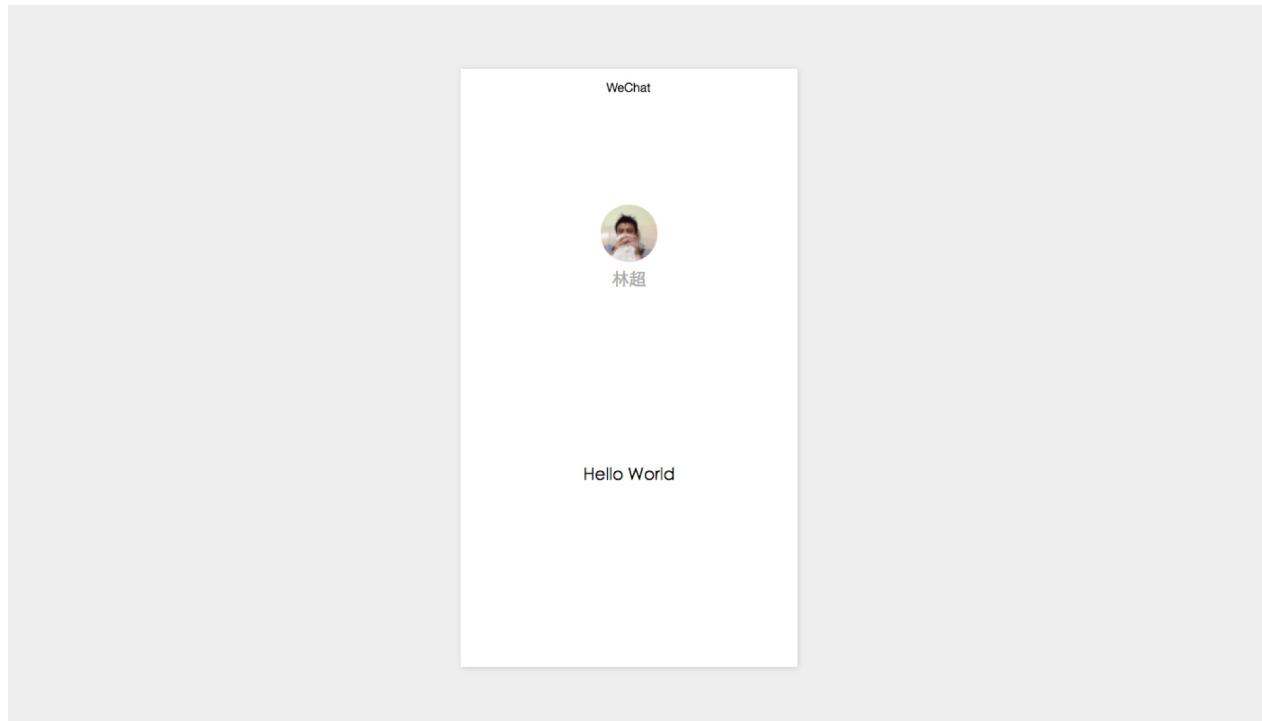
## 扫码登录

启动工具时，开发者需要使用已在后台绑定成功的微信号扫描二维码登录，后续所有的操作都会基于这个微信帐号

程序调试主要有三大功能区：模拟器、调试工具和小程序操作区

## 模拟器

模拟器模拟微信小程序在客户端真实的逻辑表现，对于绝大部分的 API 均能够在模拟器上呈现出正确的状态。

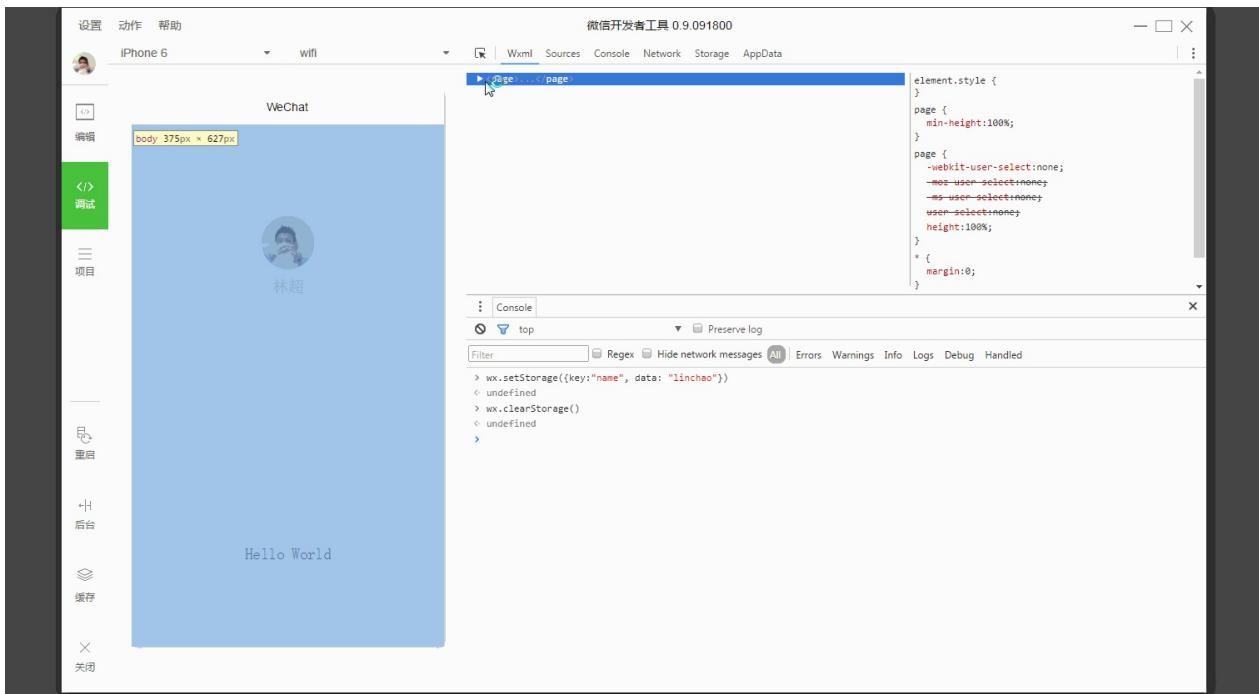


## 调试工具

调试工具分为 6 大功能模块：Wxml、Console、Sources、Network、Appdata、Storage

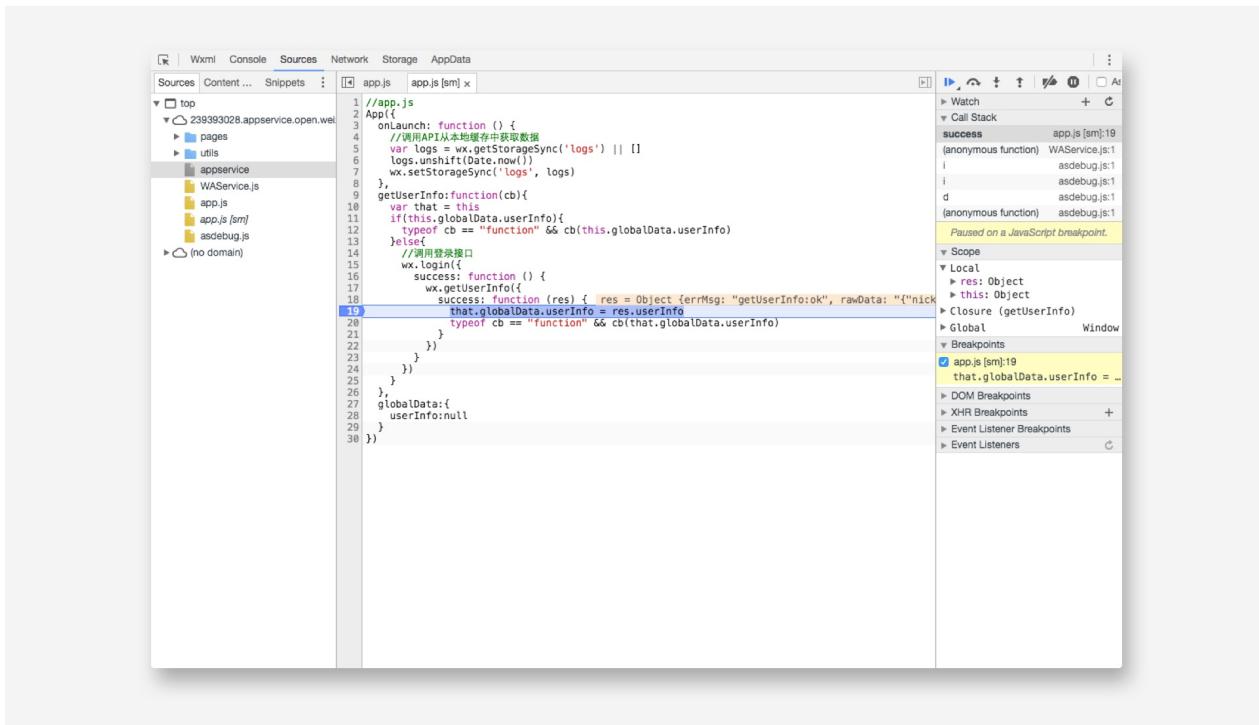
### **Wxml panel**

**Wxml panel** 用于帮助开发者开发 Wxml 转化后的界面。在这里可以看到真实的页面结构以及结构对应的 wxss 属性，同时可以通过修改对应 wxss 属性，在模拟器中实时看到修改的情况。通过调试模块左上角的选择器，还可以快速找到页面中组件对应的 wxml 代码。



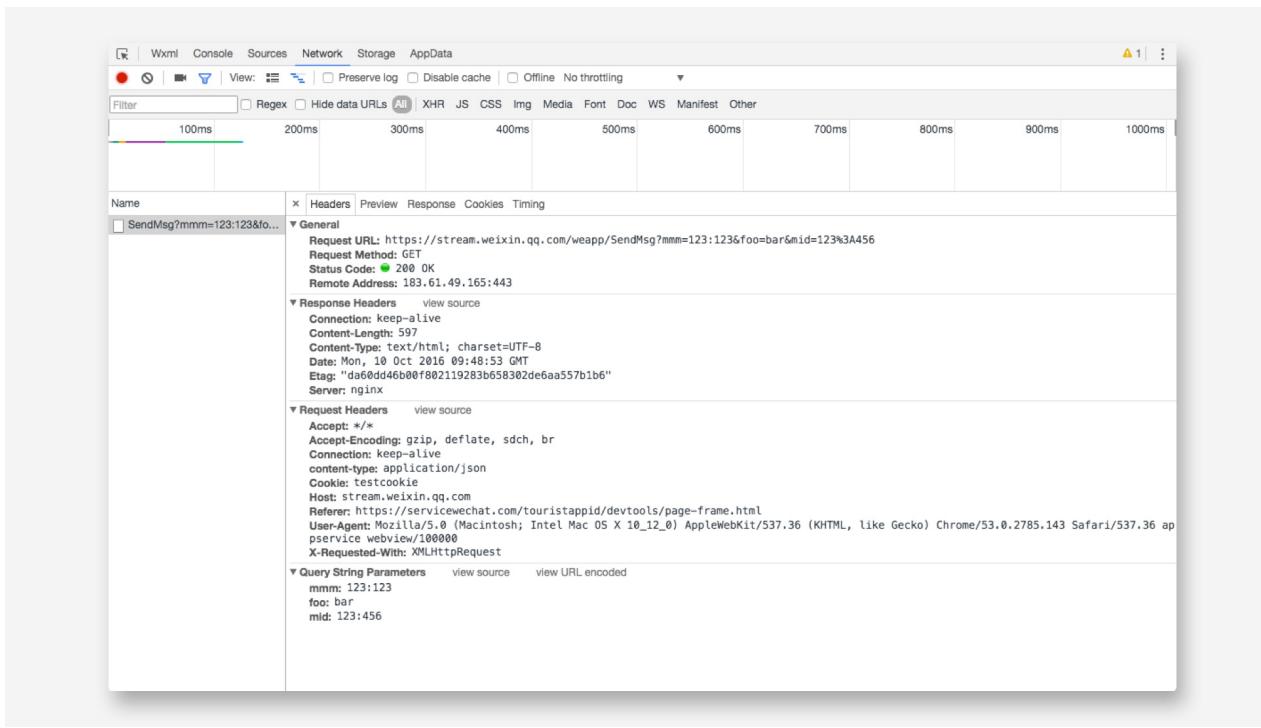
## Sources panel

Sources panel 用于显示当前项目的脚本文件，同浏览器开发不同，微信小程序框架会对脚本文件进行编译的工作，所以在 Sources panel 中开发者看到的文件是经过处理之后的脚本文件，开发者的代码都会被包裹在 `define` 函数中，并且对于 Page 代码，在尾部会有 `require` 的主动调用。



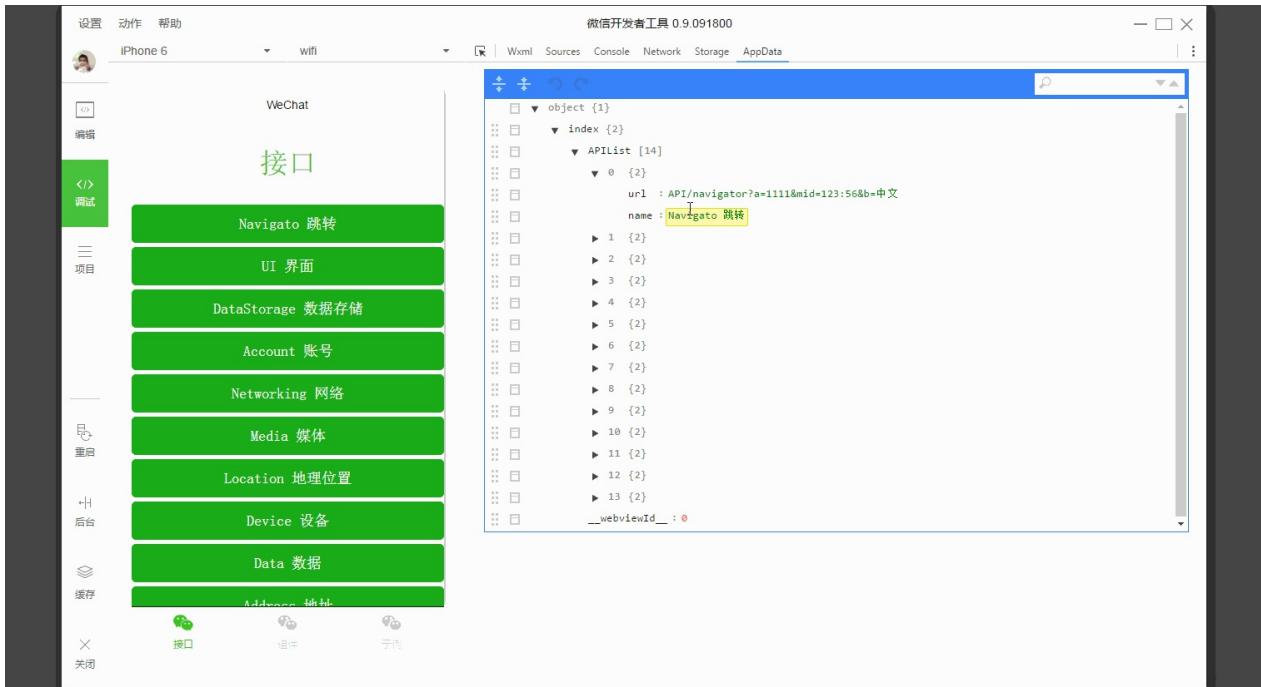
## Network panel

Netwrok Pannel 用于观察和显示 request 和 socket 的请求情况



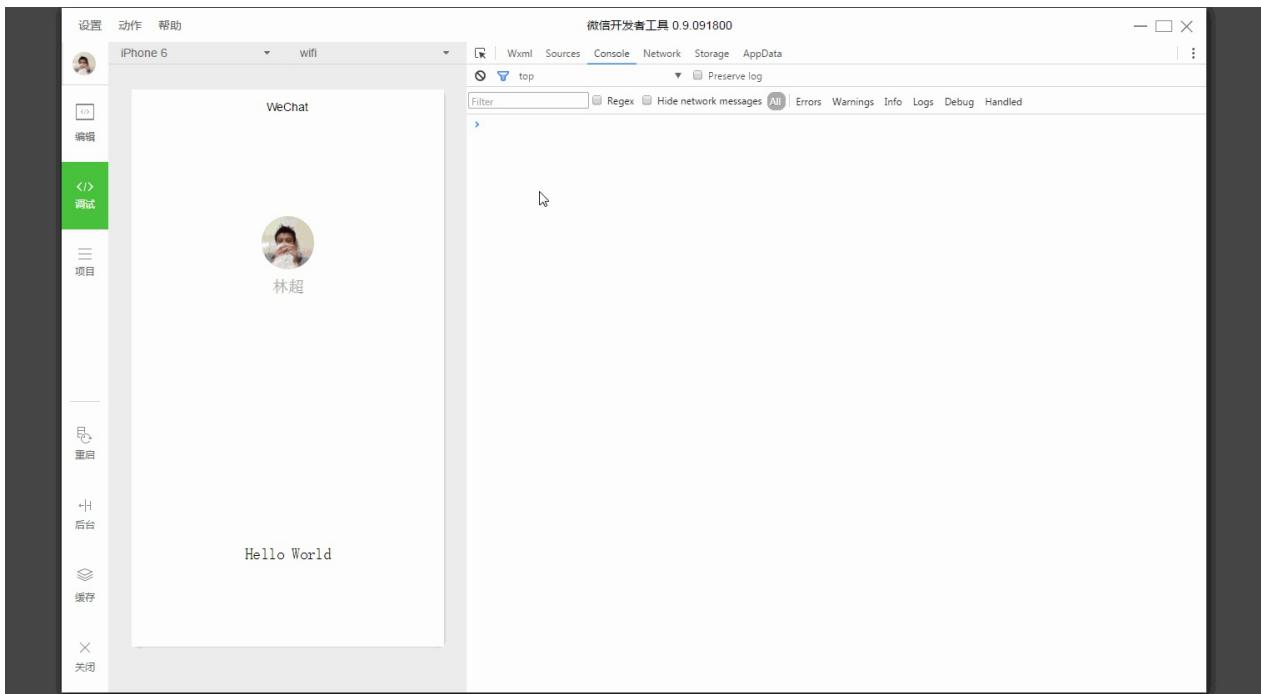
## Appdata panel

Appdata panel 用于显示当前项目当前时刻 appdata 具体数据，实时地反馈项目数据情况，可以在此处编辑数据，并及时地反馈到界面上。



## Storage panel

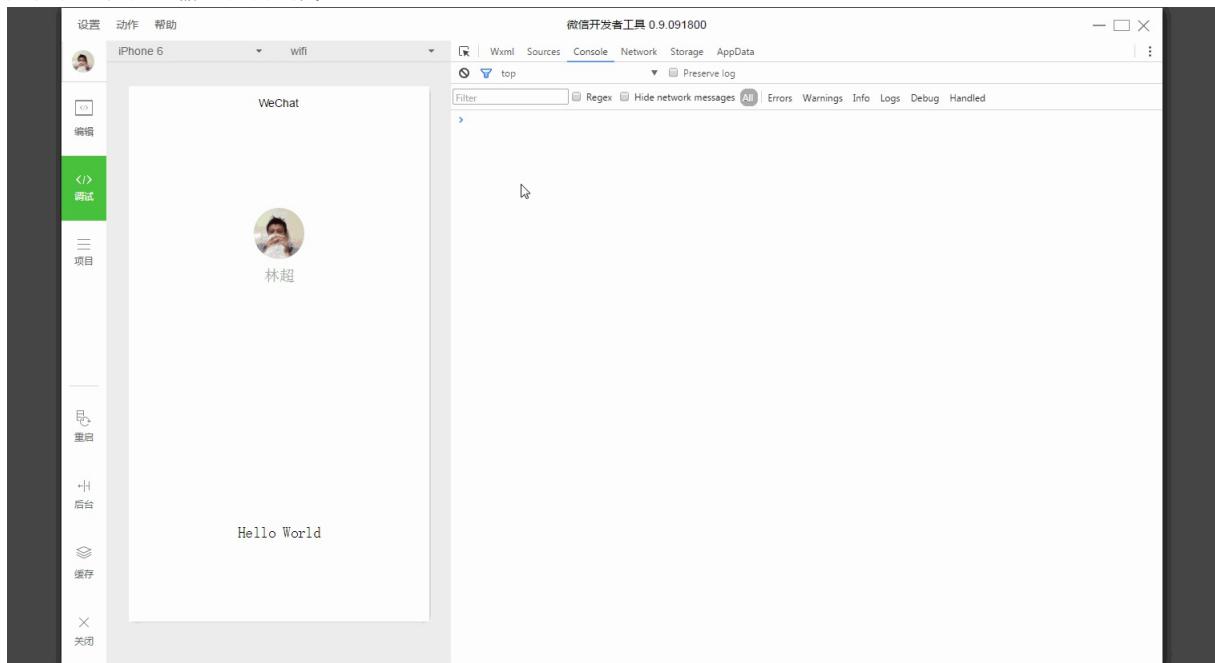
Storage panel 用于显示当前项目的使用 `wx.setStorage` 或者 `wx.setStorageSync` 后的数据存储情况。



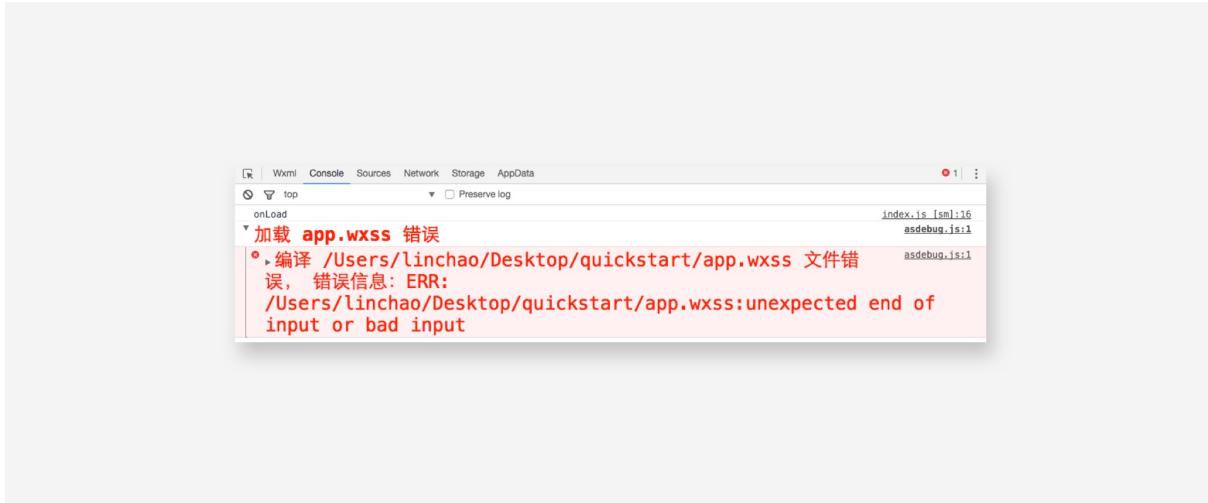
## Console panel

Console panel 有两大功能：

- 开发者可以在此输入和调试代码

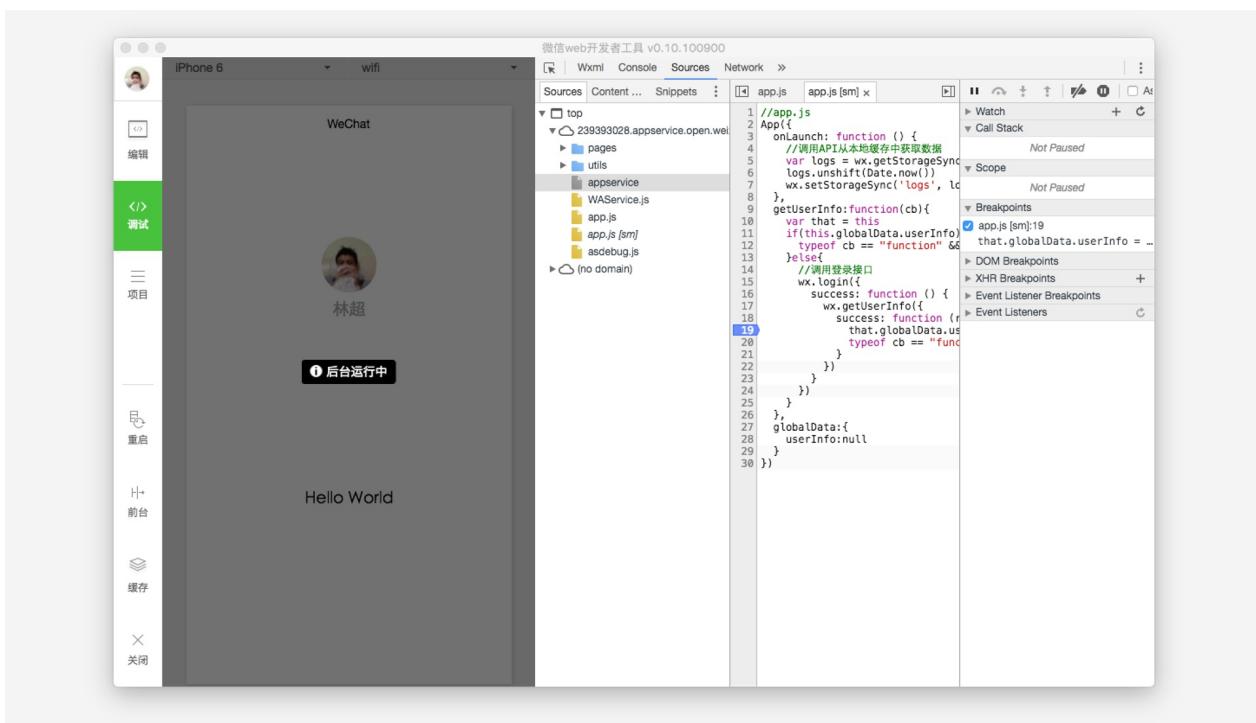


- 小程序的错误输出，会显示在此处

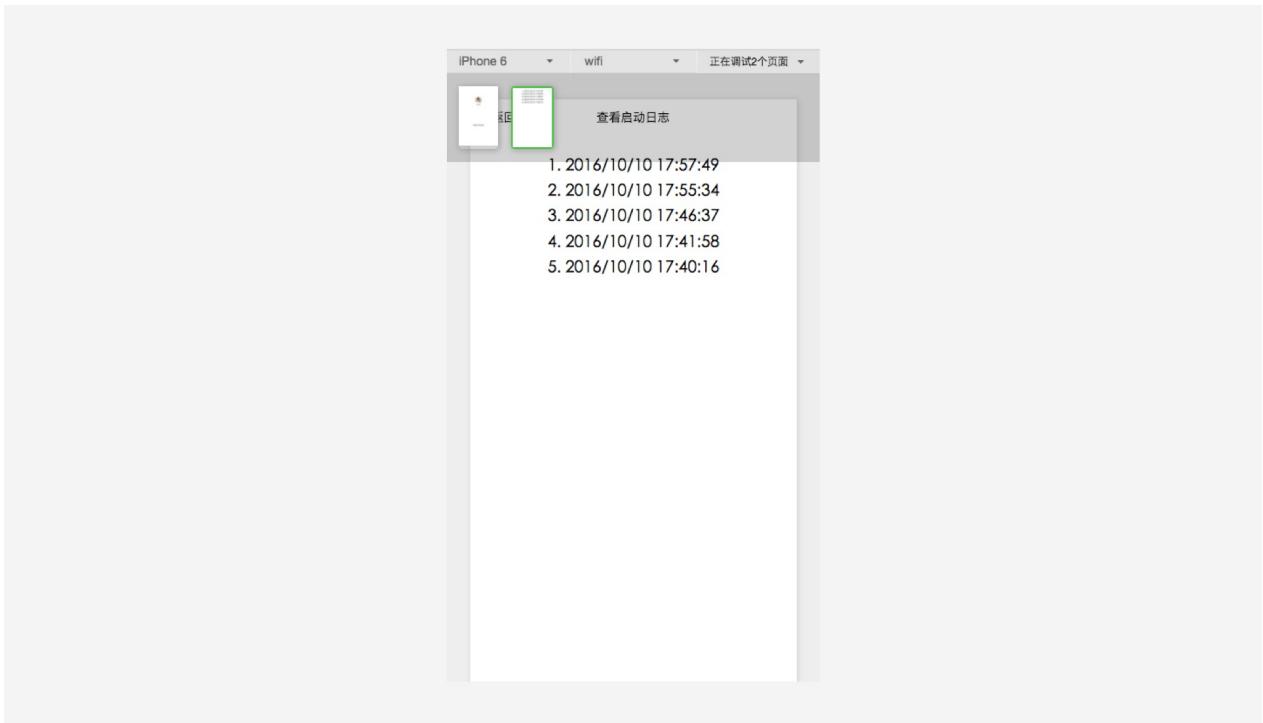


## 小程序操作区

小程序操作区帮助开发者模拟一些客户端的环境操作。例如当用户从小程序中回到聊天窗口，会触发一个小程序被设置为后台的api。



当小程序使用到多窗口的时候，可以在顶部操作区进行页面切换，需要注意的是这个操作只是为了方便开发者才存在的，在真实的微信客户端中是不会有的。

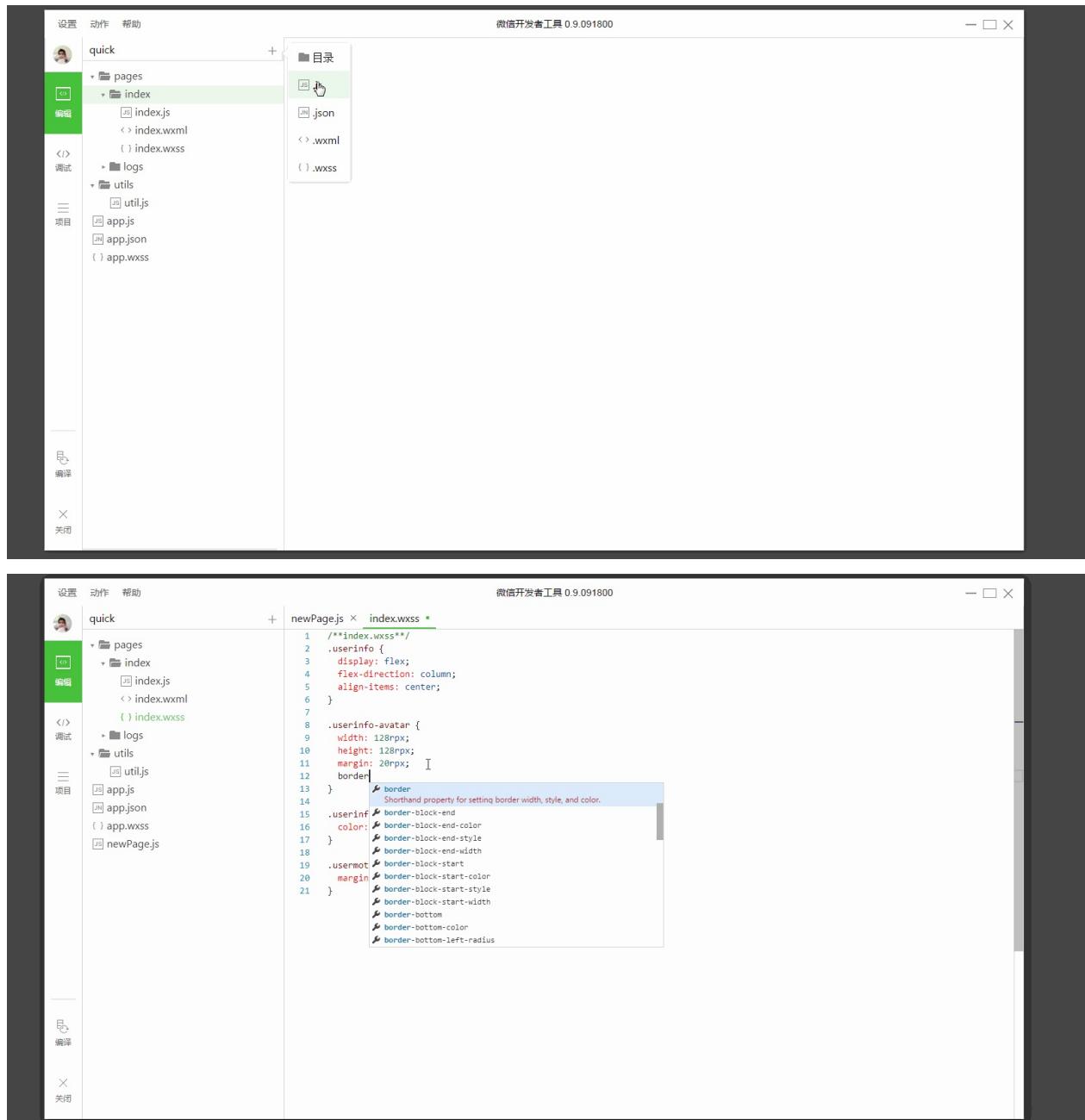


编辑区可以对当前项目进行代码书写工作，同时可以对文件进行基本的添加、删除以及重命名。

工具目前提供了4种文件的编辑：wxml wxss js json

## 自动补全

同大部分编辑器一样，我们提供了完善的自动补全



## 常用快捷键

### 格式调整

- **Ctrl+S:** 保存文件
- **Ctrl+[, Ctrl+]:** 代码行缩进
- **Ctrl+Shift+[, Ctrl+Shift+]:** 折叠打开代码块

- **Ctrl+C Ctrl+V:** 复制粘贴，如果没有选中任何文字则复制粘贴一行
- **Shift+Alt+F:** 代码格式化
- **Alt+Up, Alt+Down:** 上下移动一行
- **Shift+Alt+Up, Shift+Alt+Down:** 向上向下复制一行
- **Ctrl+Shift+Enter:** 在当前行上方插入一行

## 光标相关

- **Ctrl+End:** 移动到文件结尾
- **Ctrl+Home:** 移动到文件开头
- **Ctrl+i:** 选中当前行
- **Shift+End:** 选择从光标到行尾
- **Shift+Home:** 选择从行首到光标处
- **Ctrl+Shift+L:** 选中所有匹配
- **Ctrl+D:** 选中匹配
- **Ctrl+U:** 光标回退

## 界面相关

- **Ctrl + \:** 隐藏侧边栏

项目页卡主要有三个功能

## 显示当前项目细节

包括图标、AppID、目录信息，以及上次提交代码的时间以及代码包大小。

## 提交预览和提交上传

- 点击预览功能，工具会上传源代码到微信服务器，成功后将会显示一个二维码，开发者用新版微信扫描二维码即可在手机上看到相应项目的真实表现。
- 点击上传，工具会上传源代码到微信服务器，开发者可以在 mp 管理后台看到本次提交的情况。需要注意的是，内测阶段，代码上传 功能仅管理员微信号可操作。

## ES6 转 ES5

微信小程序运行在三端：iOS、Android 和用于调试的开发者工具

- 在 iOS 上，小程序的 javascript 代码是运行在 JavaScriptCore 中
- 在 Android 上，小程序的 javascript 代码是运行在 X5 内核中
- 在开发工具上，小程序的 javascript 代码是运行在 nwjs（chrome）中

虽然三个运行环境在大部分情况下是相似的，但是还有一些细微的区别，为了帮助开发者解决这种区别带来的困扰，开发工具会自动帮助开发者将 ES6 的代码转为 ES5 的代码。

对于使用其他构建工具的开发者，可以在项目中开关闭掉这个功能，使用自己的构建和转码工具。

## 下载地址

最新版本 **0.10.101100**

[windows 64](#)、[windows 32](#)、[mac](#)

## 发布计划

内测阶段，开发工具将会稳定的按照 2 周一次的节奏发布版本更新，但如果遇到影响的使用的问题，小程序团队会在 2 个版本之间发布小版本来解决这些问题。

更新日志（**0.10.101400**）小版本更新

### 开发者工具基础功能

1. F 修复 下拉刷新无法使用的问题
2. F 修复 app.json 文件修改后开发工具没有及时更新的问题

更新日志（**0.10.101100**）大版本更新

### 基础功能

1. A 增加 <video/> Android 添加了默认的控件
2. A 增加 模块化中可使用 exports 对外暴露接口
3. A 增加 模块化中 require 可不写 .js 后缀
4. F 修复 <swiper/> 滑动灵敏度
5. F 修复 <toast/> 中图标位置偏上，没有居中的问题
6. F 修复 <view/> 标签 hidden 属性失效的问题
7. F 修复 <input/> iOS10 下首次输入不显示的问题
8. F 修复 <button/> type="mini" 的问题
9. F 修复 <button/> 出现 loading 时，loading 和文字对齐的问题
10. F 修复 <canvas/> drawImage 图片路径不正确的问题
11. F 修复 Page 中 data 之外的数据无法被重置的问题
12. F 修复 大小写导致的 wx.request 的 header 参数属性被重复设置的问题
13. F 修复 app.js 中无法使用 require 的问题
14. R 移除 <switch/> 组件多余点击态
15. R 移除 <view/> 标签 inline 属性
16. R 移除 <page/> 标签的 height 100% 的默认样式

### 开发者工具基础功能

1. A 增加 ES6 到 ES5 的转换，默认开启，开发者可以在项目中主动关闭
2. A 增加 提交代码时候可选压缩代码，默认关闭，开发者可以在项目中主动开启
3. A 增加 wx.uploadFile 和 wx.downloadFile 调试支持
4. A 增加 下拉刷新的调试支持
5. A 增加 <form/> reportSubmit 模式模拟返回 formId 调试支持
6. A 增加 <video/> 添加了滑动进度条的功能
7. A 增加 <picker/> mode=time mode=date
8. F 修复 打开地图导致错误的问题
9. F 修复 <map/> 组件不显示的问题

10. F 修复 `<canvas/>` 中 `drawImage` 闪烁的问题
11. F 修复 `json` 中 `navigationBarTextStyle` 缺省值设置出错的问题
12. F 修复 `<picker/>` 在表单提交事件中 `value` 为空的问题
13. F 修复 背景音乐停止时会触发一次 `wx.onBackgroundAudioPause` 的问题
14. F 修复 `wx.request` 超时会触发两次 `fail` 和 `complete` 回调的问题
15. F 修复 小屏幕下开发者工具无法拖动到底部的问题
16. F 修复 `wx.setStorage` 没有限制大小的问题
17. F 修复 某些情况下修改了代码文件但工具没有更新的问题

## 编辑模块

1. A 增加侧边栏可以拖拽位置保存
2. A 增加快捷键 `ctrl + \` 或 `command + \` 隐藏侧边栏
3. A 增加文件页卡可以拖动排序的功能
4. F 修复 `wx.setNavigationBarTitle` 提示错误的问题
5. F 修复 某些情况下文件修改不生效的问题

## 怎么获取用户输入

能够获取用户输入的组件，需要使用组件的属性 `bindchange` 将用户的输入内容同步到 AppService。

```
<input id="myInput" bindchange="bindChange" />
<checkbox id="myCheckbox" bindchange="bindChange" />

var inputContent = {}

Page({
  data: {
    inputContent: {}
  },
  bindChange: function(e) {
    inputContent[e.currentTarget.id] = e.detail.value
  }
})
```

## 为什么脚本内不能使用 `window` 等对象

页面的脚本逻辑是在 `JsCore` 中运行，`JsCore` 是一个没有窗口对象的环境，所以不能在脚本中使用 `window`，也无法在脚本中操作组件

## 为什么 `zepto/jquery` 无法使用

`zepto/jquery` 会使用到 `window` 对象和 `document` 对象，所以无法使用。

## `wx.navigateTo` 无法打开页面

一个应用同时只能打开5个页面，当已经打开了5个页面之后，`wx.navigateTo` 不能正常打开新页面。请避免多层级的交互方式，或者使用 `wx.redirectTo`

## 样式表不支持级联选择器

WXSS 支持以 `.` 开始的类选择器。如：

```
.normal_view {
  color: #000000;
  padding: 10px;
}
```

可以使用标签选择器，控制同一类组件的样式。如：使用 `input` 标签选择器控制 `<input/>` 的默认样式。

```
input {
  width: 100px;
}
```

## 本地资源无法通过 `css` 获取

`background-image` : 可以使用网络图片, 或者 `base64`, 或者使用 `<image/>` 标签

## 如何修改窗口的背景色

使用 `page` 标签选择器, 可以修改顶层节点的样式

```
page {  
  display: block;  
  min-height: 100%;  
  background-color: red;  
}
```

我们建议你先完整阅读该开发文档，这将有助于更快地完成开发。如果发现我们的文档有任何错漏，或者开发过程中有任何疑问，欢迎通过下列邮箱联系我们。

weixin\_developer@qq.com

为方便定位原因，如果是开发过程中的问题，我们建议你提供更多信息，包括但不限于：

公司名称

mp账户

开发者微信号

机型

操作系统

是否必现

出现时间

操作路径

问题描述

问题截图

代码片段截图