简易教程

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（类似css）。微信小程序会读取这些文件，并生成[小程序实例](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/app.html?t=20161230)。

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](https://mp.weixin.qq.com/debug/wxadoc/dev/api/?t=20161230)

//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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230)

{

"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;

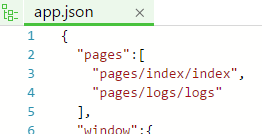
padding: 200rpx 0;

box-sizing: border-box;

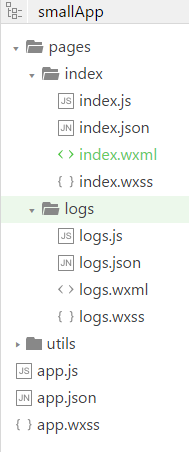
}

创建页面

在这个教程里，我们有两个页面，index 页面和 logs 页面(app.json文件中有体现)，



即欢迎页和小程序启动日志的展示页，他们都在 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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230)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/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/view.html?t=20161230)、[<image/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/image.html?t=20161230)、[<text/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/text.html?t=20161230)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（即越往里优先级越高）。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/>](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/list.html?t=20161230#block-wxfor) 控制标签来组织代码，在 <block/> 上使用 [wx:for](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/list.html?t=20161230#block-wxfor) 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))

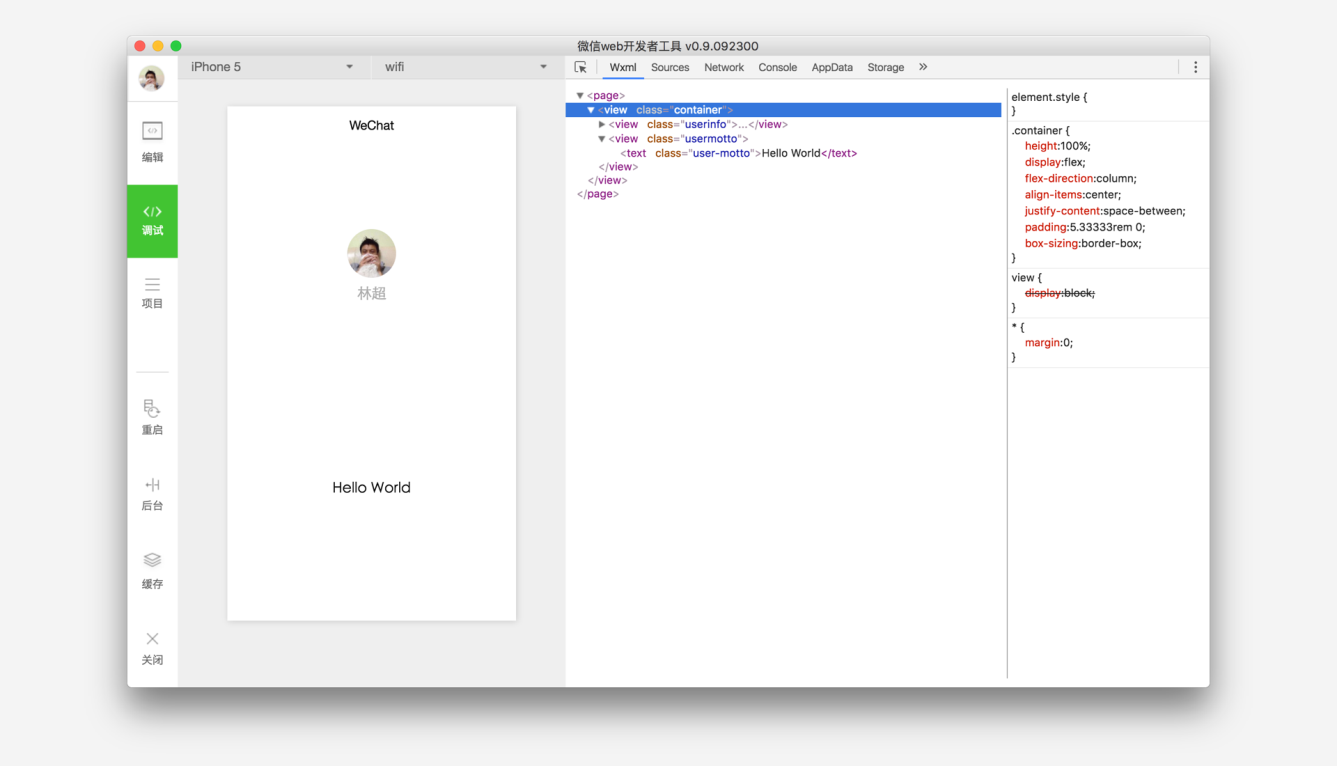
})

})

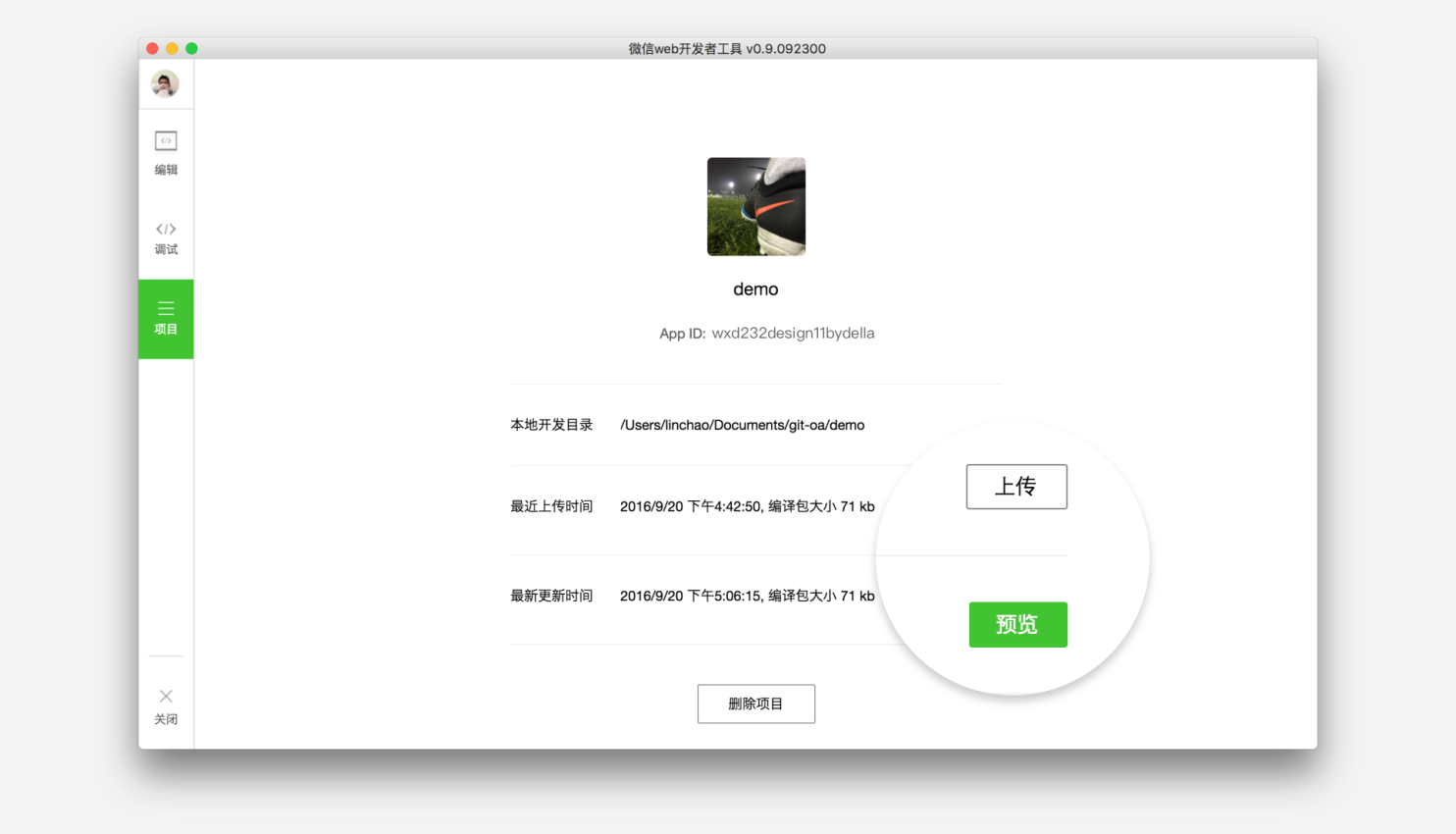
}

})

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。

框架:

目录结构：

# 文件结构

小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。

一个小程序主体部分由三个文件组成，必须放在项目的根目录（相比每个page，缺少page页wxml），如下：

| 文件 | 必填 | 作用 |
| --- | --- | --- |
| [app.js](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/app.html?t=20161230) | 是 | 小程序逻辑 |
| [app.json](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230) | 是 | 小程序公共设置 |
| [app.wxss](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxss.html?t=20161230) | 否 | 小程序公共样式表 |

一个小程序页面由四个文件组成(在以上的基础上增加了wxml页面)，分别是：

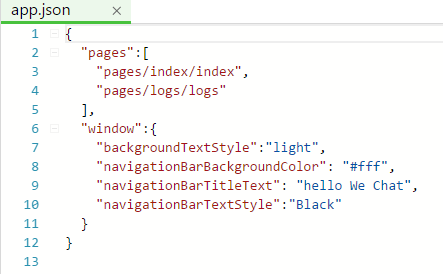
| 文件类型 | 必填 | 作用 |
| --- | --- | --- |
| [js](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230) | 是 | 页面逻辑 |
| [wxml](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/?t=20161230) | 是 | 页面结构 |
| [wxss](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxss.html?t=20161230) | 否 | 页面样式表 |
| [json](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#pagejson) | 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：

# 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 ：

{

页面配置，必须写在此

"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 |
| --- | --- | --- | --- |
| [pages](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#pages) | String Array | 是 | 设置页面路径 |
| [window](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#window) | Object | 否 | 设置默认页面的窗口表现 |
| [tabBar](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#tabbar) | Object | 否 | 设置底部 tab 的表现 |
| [networkTimeout](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#networktimeout) | Object | 否 | 设置网络超时时间 |
| [debug](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#debug) | Boolean | 否 | 设置是否开启 debug 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（pop和push）。**

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 |
| --- | --- | --- | --- |
| navigationBarBackgroundColor | HexColor | #000000 | 导航栏背景颜色，如"#000000" |
| navigationBarTextStyle | String | white | 导航栏标题颜色 |
| navigationBarTitleText | String |  | 导航栏标题文字内容 |
| backgroundColor | HexColor | #ffffff | 窗口的背景色 |
| backgroundTextStyle | String | dark | 下拉背景字体、loading 图的样式，仅支持 dark/light |
| enablePullDownRefresh | Boolean | false | 是否开启下拉刷新，详见[页面相关事件处理函数](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230#页面相关事件处理函数)。 |

**注：HexColor（十六进制颜色值），如"#ff00ff"**

如 app.json ：

{

"window":{

"navigationBarBackgroundColor": "#ffffff",

"navigationBarTextStyle": "black",

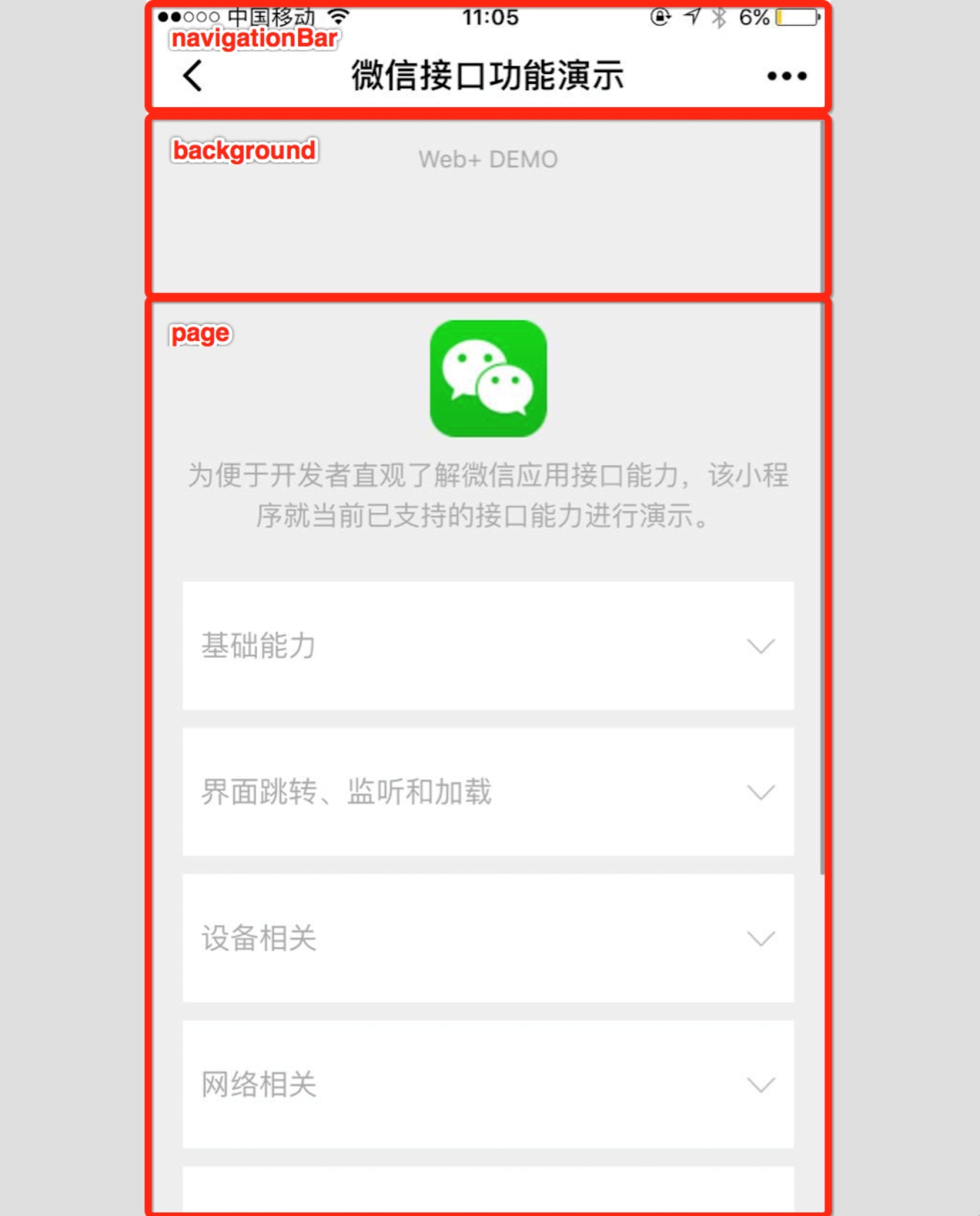
"navigationBarTitleText": "微信接口功能演示",

"backgroundColor": "#eeeeee",

"backgroundTextStyle": "light"

}

}



### tabBar

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

**Tip：** 通过页面跳转（wx.navigateTo）或者页面重定向（wx.redirectTo）所到达的页面，即使它是定义在 tabBar 配置中的页面，也不会显示底部的 tab 栏。

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 |
| position | String | 否 | bottom | 可选值 bottom、top |

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](https://mp.weixin.qq.com/debug/wxadoc/dev/api/network-request.html?t=20161230)的超时时间，单位毫秒，默认为：60000 |
| connectSocket | Number | 否 | [wx.connectSocket](https://mp.weixin.qq.com/debug/wxadoc/dev/api/network-socket.html?t=20161230)的超时时间，单位毫秒，默认为：60000 |
| uploadFile | Number | 否 | [wx.uploadFile](https://mp.weixin.qq.com/debug/wxadoc/dev/api/network-file.html?t=20161230#wxuploadfileobject)的超时时间，单位毫秒，默认为：60000 |
| downloadFile | Number | 否 | [wx.downloadFile](https://mp.weixin.qq.com/debug/wxadoc/dev/api/network-file.html?t=20161230#wxdownloadfileobject)的超时时间，单位毫秒，默认为：60000 |

### 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：

| 属性 | 类型 | 默认值 | 描述 |
| --- | --- | --- | --- |
| navigationBarBackgroundColor | HexColor | #000000 | 导航栏背景颜色，如"#000000" |
| navigationBarTextStyle | String | white | 导航栏标题颜色，仅支持 black/white |
| navigationBarTitleText | String |  | 导航栏标题文字内容 |
| backgroundColor | HexColor | #ffffff | 窗口的背景色 |
| backgroundTextStyle | String | dark | 下拉背景字体、loading 图的样式，仅支持 dark/light |
| enablePullDownRefresh | Boolean | false | 是否开启下拉刷新，详见[页面相关事件处理函数](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230#页面相关事件处理函数)。 |
| disableScroll | Boolean | false | 设置为 true 则页面整体不能上下滚动；只在 page.json 中有效，无法在 app.json 中设置该项 |

{

"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。

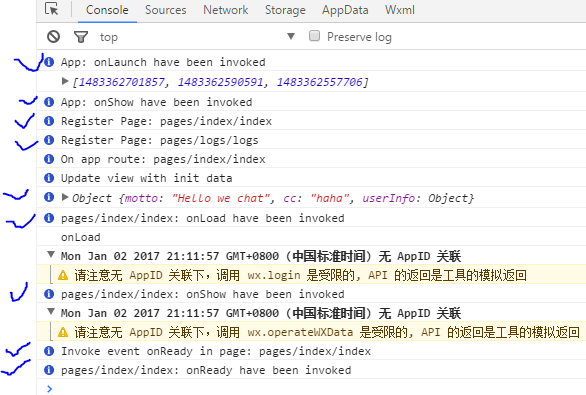
* 增加 [App](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/app.html?t=20161230) 和 [Page](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230) 方法，进行程序和页面的注册。
* 增加 getApp 和 getCurrentPages 方法，分别用来获取 App 实例和当前页面栈。
* 提供丰富的 [API](https://mp.weixin.qq.com/debug/wxadoc/dev/api/?t=20161230)，如微信用户数据，扫一扫，支付等微信特有能力。
* 每个页面有独立的[作用域](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/module.html?t=20161230#文件作用域)，并提供[模块化](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/module.html?t=20161230#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。

注册程序：

# App

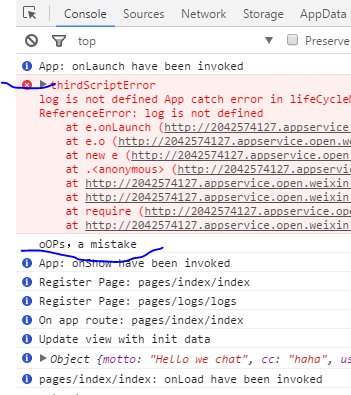
### App()

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 |
| --- | --- | --- | --- |
| onLaunch | Function | 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 onLaunch（全局只触发一次） |
| onShow | Function | 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（可以是从其他app转到小程序app），会触发 onShow |
| onHide | Function | 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 onHide |
| onError | Function | 错误监听函数 | 当小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息 |
| 其他 | Any |  | 开发者可以添加任意的函数或数据到 Object 参数中，用 this 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.（后台到前台或者开始启动显示）

},

onHide: function() {

// Do something when hide.（前天到后台或者销毁）

},

onError: function(msg) {

console.log(msg)

},

globalData: 'I am global data'

})

### App.prototype.getCurrentPage()

将被废弃，请用[getCurrentPages()](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230#getCurrentPages())替代

### 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() //这里的appInstance是局部变量

console.log(appInstance.globalData) // I am global data

**注意：**

App() 必须在 app.js 中注册（即定义），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（onShow,onHide,onLaunch,onUnlaunch）。

注册页面：

# 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（data,函数，与vue不同的是将方法和data并列，只是在使用的时候才能判断时调用方法还是引用参数）。

**object 参数说明：**

| 属性 | 类型 | 描述 |
| --- | --- | --- |
| [data](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230#初始化数据) | Object | 页面的初始数据 |
| onLoad | Function | 生命周期函数--监听页面加载 |
| onReady | Function | 生命周期函数--监听页面初次渲染完成 |
| onShow | Function | 生命周期函数--监听页面显示（App中是小程序显示） |
| onHide | Function | 生命周期函数--监听页面隐藏（App中是小程序隐藏） |
| onUnload | Function | 生命周期函数--监听页面卸载 |
| onPullDownRefresh | Function | 页面相关事件处理函数--监听用户下拉动作 |
| onReachBottom | Function | 页面上拉触底事件的处理函数 |
| onShareAppMessage | Function | 用户点击右上角分享 |
| 其他 | Any | 开发者可以添加任意的函数或数据到 object 参数中，在页面的函数中用 this 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.

},

onReachBottom: function() {

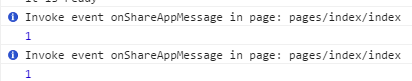
// Do something when page reach bottom.

},

onShareAppMessage: function () {

// return custom share data when user share.

},



// Event handler.

viewTap: function() {

this.setData({

text: 'Set some data for updating view.'

})

},

customData: {

hi: 'MINA'

}

})

### 

### 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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/?t=20161230) 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。
  + 接收页面参数可以获取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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/page.html?t=20161230#生命周期)
* onHide: 页面隐藏
  + 当navigateTo或底部tab切换时调用（隐藏不代表销毁）。
* 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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/config.html?t=20161230#window)选项中开启enablePullDownRefresh。
  + 当处理完数据刷新后，[wx.stopPullDownRefresh](https://mp.weixin.qq.com/debug/wxadoc/dev/api/ui-other.md?t=20161230)可以停止当前页面的下拉刷新。
* onShareAppMessage: 用户分享
  + 只有定义了此事件处理函数，右上角菜单才会显示“分享”按钮
  + 用户点击分享按钮的时候会调用
  + 此事件需要 return 一个 Object，用于自定义分享内容

**自定义分享字段**

| 字段 | 说明 | 默认值 |
| --- | --- | --- |
| title | 分享标题 | 当前小程序名称 |
| desc | 分享描述 | 当前小程序名称 |
| path | 分享路径 | 当前页面 path ，必须是以 / 开头的完整路径 |

**示例代码**

Page({

onShareAppMessage: function () {

return {

title: '自定义分享标题',

desc: '自定义分享描述',

path: '/page/user?id=123'

}

}

})

### 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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/event.html?t=20161230)，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>{{object.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 danamic 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'

})

}

})

### getCurrentPages()

getCurrentPages() 函数用于获取当前页面栈的实例，以数组形式按栈的顺序给出，第一个元素为首页，最后一个元素为当前页面。

**Tip：不要尝试修改页面栈，会导致路由以及页面状态错误。**

### 页面栈

框架以栈的形式维护了当前的所有页面。 当发生路由切换的时候，页面栈的表现如下：

| 路由方式 | 页面栈表现 |
| --- | --- |
| 初始化 | 新页面入栈 |
| 打开新页面 | 新页面入栈 |
| 页面重定向 | 当前页面出栈，新页面入栈 |
| 页面返回 | 页面不断出栈，直到目标返回页，新页面入栈 |
| Tab 切换 | 页面全部出栈，只留下新的 Tab 页面 |

**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 [wx.navigateTo](https://mp.weixin.qq.com/debug/wxadoc/dev/api/ui-navigate.html?t=20161230#wxnavigatetoobject) 或使用组件 [<navigator open-type="navigate"/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/navigator.html?t=20161230) | onLoad，onShow | onHide |
| 页面重定向 | 调用 API [wx.redirectTo](https://mp.weixin.qq.com/debug/wxadoc/dev/api/ui-navigate.html?t=20161230#wxredirecttoobject) 或使用组件 [<navigator open-type="redirect"/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/navigator.html?t=20161230) | onLoad，onShow | onUnload |
| 页面返回 | 调用 API [wx.navigateBack](https://mp.weixin.qq.com/debug/wxadoc/dev/api/ui-navigate.html?t=20161230#wxnavigateback) 或用户按左上角返回按钮 | onShow | onUnload（多层页面返回每个页面都会按顺序触发onUnload） |
| Tab 切换 | 调用 API [wx.switchTab](https://mp.weixin.qq.com/debug/wxadoc/dev/api/ui-navigate.html?t=20161230#wxswitchtab) 或使用组件 [<navigator open-type="switchTab"/>](https://mp.weixin.qq.com/debug/wxadoc/dev/component/navigator.html?t=20161230) 或用户切换 Tab |  | 各种情况请参考下表 |

Tab 切换对应的生命周期（以 A、B 页面为 Tabbar 页面，C 是从 A 页面打开的页面，D 页面是从 C 页面打开的页面为例）：

| 当前页面 | 路由后页面 | 触发的生命周期（按顺序） |
| --- | --- | --- |
| A | A | Nothing happened |
| A | B | A.onHide(), B.onLoad(), B.onShow() |
| A | B（再次打开） | A.onHide(), B.onShow() |
| C | A | C.onUnload(), A.onShow() |
| C | B | C.onUnload(), B.onLoad(), B.onShow() |
| D | B | D.onUnload(), C.onUnload(), B.onLoad(), B.onShow() |
| D（从分享进入） | A | D.onUnload(), A.onLoad(), A.onShow() |
| D（从分享进入） | B | D.onUnload(), B.onLoad(), B.onShow() |

#### Bug & Tip

1. bug: iOS/Android 6.3.30, 首次进入页面，如果页面不满一屏时会触发 onReachBottom ，应为只有用户主动上拉才触发；
2. bug: iOS/Android 6.3.30, 手指上拉，会触发多次 onReachBottom ，应为一次上拉，只触发一次；

## 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()](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/app-service/app.html?t=20161230#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 it run before b.js, now the globalData shoule 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')

}

})

# 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](https://mp.weixin.qq.com/debug/wxadoc/dev/api/?t=20161230)

# 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](https://mp.weixin.qq.com/debug/wxadoc/dev/component/?t=20161230)、[事件系统](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/event.html?t=20161230)，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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/data.html?t=20161230)

<!--wxml-->

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

// page.js

Page({

data: {

message: 'Hello MINA!'

}

})

### [列表渲染](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/list.html?t=20161230)

<!--wxml-->

<view wx:for="{{array}}"> {{item}} </view>

// page.js

Page({

data: {

array: [1, 2, 3, 4, 5]

}

})

### [条件渲染](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/conditional.html?t=20161230)

<!--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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/template.html?t=20161230)

<!--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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/event.html?t=20161230)

<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](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/data.html?t=20161230)、[列表渲染](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/list.html?t=20161230)、[条件渲染](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/conditional.html?t=20161230)、[模板](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/template.html?t=20161230)、[事件](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/event.html?t=20161230)、[引用](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/import.html?t=20161230)

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

}

})

### 关键字(需要在双引号之内)

true：boolean 类型的 true，代表真值。

false： boolean 类型的 false，代表假值。

<checkbox checked="{{false}}"> </checkbox>

**特别注意：不要直接写*checked="false"*，其计算结果是一个字符串，转成 boolean 类型后代表真值。**

## 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'

}

})

### 数据路径运算

<view>{{object.key}} {{array[0]}}</view>

Page({

data: {

object: {

key: 'Hello '

},

array: ['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}。