# 注意

菜单图片尺寸为81

小程序中的更改值必须在this.setData（{}）中进行操作。

若数据无变化则不更新视图层，但是会继续往下执行。

无法使用本地图片背景图在小程序中显示，只能通过图片的方式，背景图只能使用网络图片的形式。

只用使用test包围的文字才可以查案选中。

小程序不认识“true”,布尔值必须加“{{true}}”才可识别。

可使用data-后接自定义属性名，来定义自定义属性，确定在for遍历得到的是哪一数据。该属性可在事件中的到。

因为获取是异步的，当渲染的时候会报错，加上wx:if就可以实现避免报错。在图片渲染时候报错。

非需要数据重新渲染的不用使用setData会增大渲染开销。

原生的类似video会置顶显示，所以在需要出蒙版的时候可以写点击在弹出蒙版的同时删除组件。

切图，任何图标都要是正方形，相同的要使用统一的大小正方形。

在小程序中，即时不需要写js的代码，那js文件中也必须要加上Page({})。

Img标签在更改前后两个数据相同时不可重复请求同一个路径，先置空就可正常显示。

**页面必须写在一个单独的文件夹内。**

**低于一个像素不可写边框。**

**在组件wxss中不应使用ID选择器、属性选择器和标签名选择器。**

**this.setData({**

**[key]:value**

**})**

**[button](https://mp.weixin.qq.com/debug/wxadoc/dev/component/button.html) 组件通过设置 open-type="contact" 亦可进入客服会话**

**App注册完成之后加载页面，但是登录事件为异步。**

**Scroll-view里面不能直接放东西，把里面的放在一个view里，相邻的也要放在一个view里面，不能散开放。**

**上拉必须有一块固定不动。**

**Image的mode属性不可多用，在初始加载时图片会变形，最好使用文字图标或者写死宽高。可设置mode截取长边留下短边，然后宽高写死。**

# **主要文件**

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。

## 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。

## 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。

## window

用于设置小程序的状态栏、导航条、标题、窗口背景色。

## tabBar

如果小程序是一个多 tab 应用（客户端窗口的底部或顶部有 tab 栏可以切换页面），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。

## networkTimeout

可以设置各种网络请求的超时时间。

## 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。

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。

# **结构**

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)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.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.wxss 是页面的样式表：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。

Utils是公用的文件，可以写各种方法。

## 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。

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

## getApp()

全局的 getApp() 函数可以用来获取到小程序实例。

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 getCurrentPages()，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。

# **标签**

view（类似div）、text（类似span）、icon等等，class同html指定样式，bindtap绑定事件（类似onclick）

block配合wx:if、wx:for使用，标明if、for的作用范围，block 只作为列表渲染用，自身不显示在页面中，不起布局作用。

1rpx = 0.5px = 1物理像素。http://blog.csdn.net/qq\_31383345/article/details/52817807

## scroll-view可滚动视图区域。

通过scroll-x与Y来控制那个方向的滚动，但是要对scroll-view设置好大小。

scroll-top与left用来设置初始化的时候滚动的位置在哪里。

Scroll中的子元素必须为行内块元素，其本身必须设置强制不换行：white-space: nowrap;

#### **image**

图片。

| 属性名 | 类型 | 默认值 | 说明 | 最低版本 |
| --- | --- | --- | --- | --- |
| src | String |  | 图片资源地址 |  |
| mode | String | 'scaleToFill' | 图片裁剪、缩放的模式 |  |
| lazy-load | Boolean | false | 图片懒加载。只针对page与scroll-view下的image有效 | [1.5.0](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/compatibility.html" \o "基础库 1.5.0 开始支持，低版本需做兼容处理。) |
| 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 = {height:'图片高度px', width:'图片宽度px'} |  |

注：宽百分百高度自动的必须这是mode类型才可

## Vedio

注：video 封面poster图片在真机IOS下会被撑大，不自适应。必须按照比例返回固定比例图片大小的图片，这个在不同机型上很难办到

seek(0)以后，再pause()，但是这样做会出现另外的bug：

第一次播放时，播放当前视频，其他视频会一起加载

除了第一次播放，虽然实现了伪停止效果，但是封面图会不见了

cover-image 里的图片不支持放大吗？

mp4视频用video组件播放会卡顿、绿屏,尤其是在快进、拖动时间条时.在video组件上，加 试试加个custom-cache=”{{false}}”属性，可以完美解决。

video封面: poster

使用的poster属性，开发工具上图片闪一下就没了。在video里用

cover-view和cover-image，开发工具里正常显示，但是图片不能自适应

video组件在填写poster属性后..转发可能引起小程序崩溃.

在一个页面内,放置2个以上的填写了poster属性的video组件,

之后在android端转发会引起小程序崩溃…不知道为什么…

# **指令**

## 循环：

使用 [<block/>](https://mp.weixin.qq.com/debug/wxadoc/dev/framework/view/wxml/list.html" \l "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" \l "block-wxfor) 绑定 logs 数据，并将 logs 数据循环展开节点

例： <block wx:for="{{logs}}" wx:for-item="log">

<text class="log-item">{{index + 1}}. {{log}}</text>

</block>

# **脚本层**

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。

Page为定义页面数据区域。

Page({

})

# **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。

触发事件更改数据

****注意：****

1. ****直接修改 this.data 而不调用 this.setData 是无法改变页面的状态的，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****。

# **Map**

注意: covers 属性即将移除，请使用 markers 替代。

display 'BYCLICK':点击显示; 'ALWAYS':常显。 只有设置后才会显示气泡。

1. tip: map 组件是由客户端创建的原生组件，它的层级是最高的。
2. tip: 请勿在 scroll-view 中使用 map 组件。
3. tip: css 动画对 map 组件无效。
4. tip: map 组件使用的经纬度是火星坐标系，调用 wx.getLocation 接口需要指定 type 为 gcj02

注：不可直接选择其他地图，但可通过微信内置地图跳转到其他地图App。

# **表单**

Form中需要加入catchsubmit="formSubmit" 提交数据 catchreset="formReset" 清空表单。

表单元素有设置好的样式可以直接套用，button有多种内置样式。

例：<button type="primary" formType="submit" loading="true">Submit</button>

# **动画**

在周期函数中以实例的方法创建createAnimation对象，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。有多种动画效果。

初始化动画名

alertBox: {},

在加载完的钩子函数中设置动画，

let animation = wx.createAnimation({

duration: 500,

timingFunction: 'ease',

})

this.animation = animation;

在定义事件中执行后跟链式动画效果，最后以step结尾，表明动画完成。最后在setData中进行更新。每执行一次export则执行一次动画。

close() {

this.animation.top("100%").opacity(0).step();

this.setData({

alertBox: this.animation.export()

})

},

页面引用

<view class='b-box dis\_f fle\_f' animation="{{alertBox}}" style='top:100%;opacity:0'>

# **页面跳转**

页面内跳转靠open-type设置跳转的方式：

例：<navigator open-type="navigate" url="/pages/pay/index/view/map/map" hover-class="navigator-hover"></navigator >

可以设置点击时的效果。

Js跳转至tobar页面需要使用

wx.switchTab({

url: '/pages/success/index',

success: function(res) {},

fail: function(res) {},

complete: function(res) {},

})

跳转至普通页面可用

redirectTo或者navigateTo

navigateTo只能跳转5次，为了避免多次跳转对用户的困惑。

而redirectTo是重定向页面，无法发返回，navigateTo则可以返回。

# **获取页面路径**

var pages = getCurrentPages() //获取加载的页面

var currentPage = pages[pages.length-1] //获取当前页面的对象

var url = currentPage.route //当前页面url

var options = currentPage.options //如果要获取url中所带的参数可以查看options

# **数据渲染**

## 组件属性(需要在双引号之内)

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

## 控制属性(需要在双引号之内)判断语句

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

## 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>

## 逻辑判断

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

## 字符串运算

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

## 数据路径运算

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

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}。

注意： 花括号和引号之间如果有空格，将最终被解析成为字符串

<view wx:for="{{[1,2,3]}} ">

{{item}}</view>

等同于

<view wx:for="{{[1,2,3] + ' '}}">

{{item}}</view>

## 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

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：

****注意：**** 花括号和引号之间如果有空格，将最终被解析成为字符串

## wx:if

在框架中，使用 wx:if="{{condition}}" 来判断是否需要渲染该代码块：也可以用 wx:elif 和 wx:else 来添加一个 else 块：

例： <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。如果要一次性判断多个组件标签，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。

## wx:key

如果列表中项目的位置会动态改变或者有新的项目添加到列表中，并且希望列表中的项目保持自己的特征和状态（如 <input/> 中的输入内容，<switch/> 的选中状态），需要使用 wx:key 来指定列表中项目的唯一的标识符。

wx:key 的值以两种形式提供

字符串，代表在 for 循环的 array 中 item 的某个 property，该 property 的值需要是列表中唯一的字符串或数字，且不能动态改变。

保留关键字 \*this 代表在 for 循环中的 item 本身，这种表示需要 item 本身是一个唯一的字符串或者数字，如：

当数据改变触发渲染层重新渲染的时候，会校正带有 key 的组件，框架会确保他们被重新排序，而不是重新创建，以确保使组件保持自身的状态，并且提高列表渲染时的效率。

如不提供 wx:key，会报一个 warning， 如果明确知道该列表是静态，或者不必关注其顺序，可以选择忽略。

# **组件**

**Wxml正常书写**

<navigator open-type='navigate' url='/pages/logs/logs'>

<view class="inner" bindtap='\_click'>

{{innerText}}

</view>

<slot name="after"></slot>

</navigator>

**组件wxml的slot**

在组件的wxml中可以包含 slot 节点，用于承载组件使用者提供的wxml结构。

默认情况下，一个组件的wxml中只能有一个slot。需要使用多slot时，可以在组件js中声明启用。**写在page外边。**

Component({

options: {

multipleSlots: true // 在组件定义时的选项中启用多slot支持

},

properties: { /\* ... \*/ },

methods: { /\* ... \*/ }

})

教程链接：<https://mp.weixin.qq.com/debug/wxadoc/dev/framework/custom-component/wxml-wxss.html>

**js部分写入组件内容**

Component({

behaviors: [],

properties: {

myProperty: { // 属性名

type: String, // 类型（必填），目前接受的类型包括：String, Number, Boolean, Object, Array, null（表示任意类型）

value: '' ,// 属性初始值（可选），如果未指定则会根据类型选择一个

observer: function(newVal, oldVal){} // 属性被改变时执行的函数（可选），也可以写成在methods段中定义的方法名字符串

},

myProperty2: String // 简化的定义方式

},

data: {}, // 私有数据，可用于模版渲染

// 生命周期函数，可以为函数，或一个在methods段中定义的方法名

attached: function(){},

moved: function(){},

detached: function(){},

methods: {

onMyButtonTap: function(){

this.setData({

// 更新属性和数据的方法与更新页面数据的方法类似

})

},

\_myPrivateMethod: function(){

// 内部方法建议以下划线开头

this.replaceDataOnPath(['A', 0, 'B'], 'myPrivateData') // 这里将 data.A[0].B 设为 'myPrivateData'

this.applyDataUpdates()

}

}

})

**组件样式**

组件对应 wxss 文件的样式，只对组件wxml内的节点生效。编写组件样式时，需要注意以下几点：

组件和引用组件的页面不能使用id选择器（#a）、属性选择器（[a]）和标签名选择器，请改用class选择器。

组件和引用组件的页面应尽量避免在样式中使用后代选择器（.a .b），否则可能导致不可预料的结果。

子元素选择器（.a>.b）只能用于 view 组件与其子节点之间，不可用于其他组件，否则可能导致不可预料的结果。

继承样式，如 font 、 color ，会从组件外继承到组件内。

除继承样式外， app.wxss 中的样式、组件所在页面的的样式对自定义组件无效。

#a { } /\* 在组件中不能使用 \*/

[a] { } /\* 在组件中不能使用 \*/

button { } /\* 在组件中不能使用 \*/

.a > .b { } /\* 除非 .a 是 view 组件节点，否则不一定会生效 \*/

除此以外，组件可以指定它所在节点的默认样式，使用 :host 选择器。

代码示例：

/\* 组件 custom-component.wxss \*/

:host {

color: yellow;

}

:host(.dark) {

color: black;

}

<!-- 页面的 WXML -->

<custom-component>这段文本是黄色的</custom-component>

<custom-component class="dark">这段文本是黑色的</custom-component>

Json中定义该文件是

{

"component": true

}

**引用：**

**wxml中**

<!-- 引用组件的页面模版 --><view>

<component-tag-name>

<!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->

<view slot="before">这里是插入到组件slot name="before"中的内容</view>

<!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->

<view slot="after">这里是插入到组件slot name="after"中的内容</view>

</component-tag-name></view>

**Json中**

{

"usingComponents": {

"component-tag-name": "path/to/the/custom/component"

}

}

**组件之间的事件**

引用页面中

<!-- 当自定义组件触发“myevent”事件时，调用“onMyEvent”方法 --><component-tag-name bindmyevent="onMyEvent" /><!-- 或者可以写成 --><component-tag-name bind:myevent="onMyEvent" />

Page({

onMyEvent: function(e){

e.detail // 自定义组件触发事件时提供的detail对象

}

})

**组件中**

<!-- 在自定义组件中 --><button bindtap="onTap">点击这个按钮将触发“myevent”事件</button>

Component({

properties: {}

methods: {

onTap: function(){

var myEventDetail = {} // detail对象，提供给事件监听函数

var myEventOption = {} // 触发事件的选项

this.triggerEvent('myevent', myEventDetail, myEventOption)

}

}

})

**触发事件的选项包括：**

| **选项名** | **类型** | **是否必填** | **默认值** | **描述** |
| --- | --- | --- | --- | --- |
| **bubbles** | **Boolean** | **否** | **false** | **事件是否冒泡** |
| **composed** | **Boolean** | **否** | **false** | **事件是否可以穿越组件边界，为false时，事件将只能在引用组件的节点树上触发，不进入其他任何组件内部** |
| **capturePhase** | **Boolean** | **否** | **false** | **事件是否拥有捕获阶段** |

**behaviors**

定义和使用 behaviors

behaviors 是用于组件间代码共享的特性，类似于一些编程语言中的“mixins”或“traits”。

每个 behavior 可以包含一组属性、数据、生命周期函数和方法，组件引用它时，它的属性、数据和方法会被合并到组件中，生命周期函数也会在对应时机被调用。每个组件可以引用多个 behavior 。 behavior 也可以引用其他 behavior 。

behavior 需要使用 Behavior() 构造器定义。

****代码示例：****

// my-behavior.jsmodule.exports = Behavior({

behaviors: [],

properties: {

myBehaviorProperty: {

type: String

}

},

data: {

myBehaviorData: {}

},

attached: function(){},

methods: {

myBehaviorMethod: function(){}

}

})

组件引用时，在 behaviors 定义段中将它们逐个列出即可。

****代码示例：****

// my-component.jsvar myBehavior = require('my-behavior')

Component({

behaviors: [myBehavior],

properties: {

myProperty: {

type: String

}

},

data: {

myData: {}

},

attached: function(){},

methods: {

myMethod: function(){}

}

})

在上例中， my-component 组件定义中加入了 my-behavior ，而 my-behavior 中包含有 myBehaviorProperty 属性、 myBehaviorData 数据字段、 myBehaviorMethod 方法和一个 attached 生命周期函数。这将使得 my-component 中最终包含 myBehaviorProperty 、 myProperty 两个属性， myBehaviorData 、 myData 两个数据字段，和 myBehaviorMethod 、 myMethod 两个方法。当组件触发 attached 生命周期时，会依次触发 my-behavior 中的 attached 生命周期函数和 my-component 中的 attached 生命周期函数。

# 组件间关系

组件之间的通信可以通过Component来实现，定义相关组件的联系。

关联一类组件，若他们有一个共同的behavior，则则在 relations 关系定义中，可使用这个behavior来代替组件路径作为关联的目标节点：

# **模版**

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

例：<template name="msgItem">

<view>

<text> {{index}}: {{msg}} </text>

<text> Time: {{time}} </text>

</view>

</template>

**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>

模板拥有自己的作用域，只能使用data传入的数据。

引入wxss使用方法为在目标wxss文件中：@import "common.wxss";

# **WXS 模块**

类似于javaScript标签，不可写在wxss中，只能在wxs模块中进行写入。WXS 代码可以编写在 wxml 文件中的 <wxs> 标签内，或以 .wxs 为后缀名的文件内。每个模块都有自己独立的作用域。即在一个模块里面定义的变量与函数，默认为私有的，对其他模块不可见。

1. wxs 不依赖于运行时的基础库版本，可以在所有版本的[小程序](http://www.wxapp-union.com/" \t "http://www.wxapp-union.com/_blank)中运行。
2. wxs 与 javascript 是不同的语言，有自己的语法，并不和 javascript 一致。
3. wxs 的运行环境和其他 javascript 代码是隔离的，wxs 中不能调用其他 javascript 文件中定义的函数，也不能调用小程序提供的API。
4. wxs 函数不能作为组件的事件回调。
5. 由于运行环境的差异，在 iOS 设备上小程序内的 wxs 会比 javascript 代码快 2 ~ 20 倍。在 android 设备上二者运行效率无差异。

只可做对数据的更改操作后返回，无法定义事件和行为。只能对数据操作后返回数据。

一个模块要想对外暴露其内部的私有变量与函数，只能通过 module.exports 实现。

例：module.exports = {

foo: foo,

bar: bar

};

在接受页面的page外引入，但是只能用相对路径，不能用绝对路径。

例：var data=require(“../../data/post-data.js”)

## require函数

在.wxs模块中引用其他 wxs 文件模块，可以使用 require 函数。

例：  
var foo = "'hello world' from tools.wxs";var bar = function (d) {

return d;

}module.exports = {

FOO: foo

};module.exports.msg = "some msg";

// /pages/logic.wxs

var tools = require("./tools.wxs");

console.log(tools.FOO);

console.log(tools.bar("logic.wxs"));

<!-- /page/index/index.wxml -->

<wxs src="./../logic.wxs" module="logic" />

## module 属性

module 属性是当前 <wxs> 标签的模块名。在单个 wxml 文件内，建议其值唯一。有重复模块名则按照先后顺序覆盖（后者覆盖前者）。不同文件之间的 wxs 模块名不会相互覆盖。

module 属性值的命名必须符合下面两个规则：

首字符必须是：字母（a-zA-Z），下划线（\_）

剩余字符可以是：字母（a-zA-Z），下划线（\_）， 数字（0-9）

****示例代码：****

<!--wxml-->

<wxs module="foo">

var some\_msg = "hello world";

module.exports = {

msg : some\_msg,

}</wxs><view> {{foo.msg}} </view>

页面输出：

hello world

上面例子声明了一个名字为 foo 的模块，将 some\_msg 变量暴露出来，供当前页面使用。

## src 属性

src 属性可以用来引用其他的 wxs 文件模块。

引用的时候，要注意如下几点：

只能引用 .wxs 文件模块，且必须使用相对路径。

wxs 模块均为单例，wxs 模块在第一次被引用时，会自动初始化为单例对象。多个页面，多个地方，多次引用，使用的都是同一个 wxs 模块对象。

如果一个 wxs 模块在定义之后，一直没有被引用，则该模块不会被解析与运行。

<wxs src="./../comm.wxs" module="some\_comms"></wxs>

### 注意

<wxs> 模块只能在定义模块的 WXML 文件中被访问到。使用 <include> 或 <import> 时，<wxs> 模块不会被引入到对应的 WXML 文件中。

<template> 标签中，只能使用定义该 <template> 的 WXML 文件中定义的 <wxs> 模块。

# **作用域与数据传递**

# **事件**

注意：通过事件改变的内容也必须经过setdata的更改才可实现。更改列表数据需要更新一下最新的数据。

例：this.setData({

startList: this.data.startList

})

后台获取的数据可以直接用this.setData()来存储到data中，不用传对象的方式。该方法的作用相当于刷新绑定数据的作用。

例：this.setData(num) num是对象类型，可以调取对象属性。

Tap事件为点击，默认加上bind即为事件全名，

前面加上catch的为非冒泡事件。

# **数据-传输-存储-获取**

顶级作用域是app。Js，其他的子页面中的js为局部作用域，app。Js中的数据都可以访问。

## 设置全局数据为globalData

访问方式为： var app = getApp()

var getFoo = app.globalData.foo

# **页面间传输数据**

1，url拼接的方式

<navigator url="/pages/mode/mode?name='{{user[0].name}}'&age={{user[0].age}}">点击跳转</navigator>

或

wx: wx.navigateTo({

url: '/pages/mode/mode?name=ni'

})

只可以传输字符串，或者转换成字符串。url后拼接?id(参数名字)=要传递的值 (如果多个参数用&分开 &name=value&…….)

获取的时候在跳转后的页面中的钩子函数中进行获取。

onLoad: function (options) {

console.log(options) 键值对的形式。

},

2，Js获取页面结构树，获取上一个页面实例，更改值并返回页面。

例：const pages = getCurrentPages();

const prePage=pages[pages.length-2];

prePage.setData({

taps: e.target.id

})

wx.navigateBack({

delta: 1

})

# **模版**

在要使用组件的页面的wxml文件中引用组件，同时使用模版，传入担任“命名空间”的对象

<import src="../../component/login-pannel/login-pannel.wxml"/> 使用include会直接把模版代码复制过来，适用于模版嵌套模版的情况。使用import引入的模版无法使用模版内引用的模版。

<template is="loginPannel" data="{{ ...\_\_lgpanel\_\_ }}"/> 传参给组件的方式是使用data传入

如果你的组件需要在多个页面中使用，那么在app.wxss中引入一次组件样式即可；如果只需要在摸个特定的页面使用，只需在该页面的样式文件中引入组件样式。

@import "./component/login-pannel/login-pannel.wxss";

如果你的组件需要在多个页面中使用，你需要在app.js中引用，同时传给App()：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | import { LoginPannel } from './component/login-pannel/login-pannel'  App({  LoginPannel,  \_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)  }  })  }  })  }  }  }) |

然后，在使用的页面的生命周期函数onLoad中“注册”该组件：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | onLoad: function () {  // 调用应用实例的方法获取全局数据  let app = getApp()    // “注册”组件  new app.LoginPannel()    app.\_getUserInfo((userInfo) => {  //更新数据  this.setData({  userInfo:userInfo  })  })  } |

然后，在这个页面中就可以愉快的使用组件啦：

# **小程序使用矢量图**

将其下载到本地，拉出来后缀是ttf的文件，在<https://transfonter.org>转成base64的格式转换后下载下来并且将原先文件中的css文件中的内容除了引入那一段其他的都复制到转化后的文件内后缀为wxss，在需要的cwxss中引入即可了。

# **富文本**

使用wxParse插件，

仓库地址：<http://blog.csdn.net/zhuming3834/article/details/74280038>

# **图片预览功能**

地址：[HTTP://mp.weixin.qq.com/debug/wxadoc/dev/api/media-picture.html#wxpreviewimageobject](HTTP://mp.weixin.qq.com/debug/wxadoc/dev/api/media-picture.html" \l "wxpreviewimageobject)

//图片预览

wx.previewImage({

current: src, // 当前显示图片的http链接

urls: imgList // 需要预览的图片http链接列表

})

# **交互反馈**

各种弹窗信息。

地址：<HTTP://mp.weixin.qq.com/debug/wxadoc/dev/api/api-react.html>

注：其中的颜色不可使用英文单词，ios上不显示。

# **下拉加载上拉刷新**

在app.json中定义enablePullDownRefresh，则全都开启，在单独页面定义则单独开启。

例：enablePullDownRefresh:true

传送门：<https://mp.weixin.qq.com/debug/wxadoc/dev/api/pulldown.html#onpulldownrefresh>

# **转发**

## onShareAppMessage(options)可在page内定义转发，也可在页面内点击转发，通过给 button 组件设置属性 open-type="share"

# **小程序之间的跳转**

wx.navigateToMiniProgram(OBJECT)，在小程序之间跳转的前提是两个小程序都是在同一个公众号下边。

https://mp.weixin.qq.com/debug/wxadoc/dev/api/navigateToMiniProgram.html