小程序的结构

App（程序）->Page（页面）->Component（组件）

App

app.js：创建App实例的代码以及一些全局相关的内容、

app.json：全局的一些配置、

app.wxss：全局的样式

Page

page.js：创建page实例的代码，以及页面相关的内容、

page.json：业务单独的配置，比如页面对应window配置、

page.wxml：页面的wxml布局代码、

page.wxss：页面的样式配置

Component

component.js：创建Component实例的代码，以及组件内部的内容

component.json：组件内部的配置，比如当前组件使用了别的组件

component.wxml：组件的wxml布局代码

component.wxss：组件的样式配置

快速生成结构

1. 通过微信开发者工具开发，保留project.config.json、sitemap.json其他可以全部删除
2. 创建app相关的三个文件，根据错误提示编写app.json的结构

```
{
"pages": [
"pages/home/home",
"pages/about/about"
]
}
```

3. 新版的工具需要在Pages目录中才能“新建page”，通过这个方式会自动生成Page默认的结构（4个文件），而且会自动在app.json中注册路径，虽然说app.js和app.wxss不是必须的，但我们会在后面使用到它们。
4. 保存后，自动更新显示界面，界面上显示的内容，就是home.wxml中的结构，修改标签中的内容，会时是更新显示，可以尝试用其他标签（也称为内置组件）生成元素，如果希望添加样式，则可以直接在home.wxss中设置，样式会自动渲染home.wxml文件中的标签。

项目托管

git标签管理项目版本

知识点保存在标签中，生成tag1、tag2

知识点结束回退到初始化状态，git reset --hard 初始化

查看某个知识点，git checkout tag1

操作步骤：

1. 在MiniProgram文件夹下，通过git bash 初始化仓库，输入以下命令初始化仓库

```
git init
```

2. 将所有文件添加到暂缓区，"."表示所有文件

```
git add .
```

3. 提交并打上初始化标记

```
git commit -m '初始化项目'
```

4. 打开github.com，创建远程仓库，"MiniProgram"，通过生成后的提示继续将远程仓库与本地仓库关联起来

> ```
> git init
> git add README.md
> git commit -m "first commit"
>
> git remote add origin https://github.com/5631723/MiniProgram.git
> git push -u origin master
> ```

5. 第一次提交可能需要登录github账户，输入当前仓库的账户即可。

> MI@SuperRao MINGW64 ~/Desktop/小程序/代码/MiniProgram (master)
> $ git push -u origin master
> git: 'credential-cache' is not a git command. See 'git --help'.
> Enumerating objects: 16, done.
> Counting objects: 100% (16/16), done.
> Delta compression using up to 8 threads
> Compressing objects: 100% (13/13), done.
> Writing objects: 100% (16/16), 1.98 KiB | 225.00 KiB/s, done.
> Total 16 (delta 1), reused 0 (delta 0)
> remote: Resolving deltas: 100% (1/1), done.
> To https://github.com/5631723/MiniProgram.git
>
> - [new branch] master -> master
>   Branch 'master' set up to track remote branch 'master' from 'origin'.

6. 学习添加标签，随意修改小程序中的代码，将home.wxml修改如下：

```
<view>1.学习添加标签，用于保存当前学习进度</view>
```

7. 将修改的文件添加到暂存区，可以将全部内容添加到暂存区

```
git add .
```

```
$ git add .
warning: LF will be replaced by CRLF in pages/home/home.wxml.
The file will have its original line endings in your working directory
```

8. 提交并打上标记

```
git commit -m '学习添加标签'
```

```
$ git commit -m '学习添加标签'
[master 321bbf8] 学习添加标签
1 file changed, 1 insertion(+)
```

9. 创建标签“01_添加标签”

```
git tag 01_添加标签
```

10. 查看已有标签

```
git tag
```

```
$ git tag
01_添加标签
```

11. 查看提交历史

```
git log
```

```
$ git log
commit 321bbf8170ebd9860aaf5304be04f3a22d9a779c (HEAD -> master, tag: 01_添加标
签)
Author: 5631723 5631723@qq.com
Date:   Sat May 9 12:23:20 2020 +0800

    学习添加标签

commit 2fedfb8dceed8e831cbf0fab709c35902b176b22 (origin/master)
Author: 5631723 5631723@qq.com
Date:   Sat May 9 11:53:14 2020 +0800

    初始化项目
```

12. 回退提交版本，不需要输入全部版本号，输入前六位即可

```
git reset 2fedfb
```

```
$ git reset 2fedfb
Unstaged changes after reset:
M       pages/home/home.wxml
```

如果修改了文件，但没有新的提交，但又希望回退到某个版本，则可以添加"--hard"参数强制回退。

```
git reset --hard 2fedfb
```

```
$ git reset --hard 2fedfb
HEAD is now at 2fedfb8 初始化项目
```

13. git status命令可以列出当前目录所有还没有被git管理的文件和被git管理且被修改但还未提交(git commit)的文件，此命令可以更新微信开发者工具一直提示文件被修改的bug，清除“m”图标提示。

```
git status
```

14. 将本地的tag推送到远程仓库，这样就可以在远程仓库中，查看到已有的Tag标签

```
git push --tags
```

15. 此时已经可以在远程仓库中查看到同步后的标签。如果需要将某个标签获取到本地编辑或查看，可以在本地重新创建一个新目录，打开git bash，输入克隆命令。克隆地址可以直接从远程仓库网站上“clone or download”处获取。

```
git clone https://github.com/5631723/MiniProgram.git
```

```
$ git clone https://github.com/5631723/MiniProgram.git
Cloning into 'MiniProgram'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 21 (delta 4), reused 20 (delta 3), pack-reused 0
Unpacking objects: 100% (21/21), done.
```

16. 在微信开发者工具中直接导入小程序，但是并没有看到标签中“01_添加标签”版本中的内容，此时还需要将标签CHECKOUT出来。

```
git checkout 01_添加标签
```

```
$ git checkout 01添加标签
Note: switching to '01添加标签'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 321bbf8 学习添加标签
M       app.json
M       project.config.json
```

17. 查看微信开发者工具中的页面，此时home.wxml中的标签1.学习添加标签，用于保存当前学习进度又恢复出来了。

小程序体验

数据绑定

是一个显示组件，就是一个容器。通过mustache语法来调用变量，在home.js中声明data属性，保存定义的变量。

home.js:

```
// pages/home/home.js
Page({
  data:{
    name:'raoqi',
    age:33
  }
})
```

home.wxml:

```
<!--pages/home/home.wxml-->
<view>Hello {{name}}</view>
<view>age:{{age}}</view>
```

列表渲染

通过wx:for来实现组件的遍历，默认通过item来遍历数组中的元素

```
// pages/home/home.js
Page({
  data: {
    name: 'raoqi',
    age: 33,
    students: [
        {id: 1,name: '小张',age: 18},
        {id: 2,name: '小王',age: 17},
        {id: 3,name: '小李',age: 16},
        {id: 4,name: '小陈',age: 19}
    ]
})
```

```
<!--pages/home/home.wxml-->
<!-- 数据绑定 -->
<view>Hello {{name}}</view>
<view>age:{{age}}</view>
<!-- 列表展示 -->
<view wx:for='{{students}}'>{{item.name}}的年龄是{{item.age}}岁</view>
```

事件监听

再小程序开发中，将定义的方法直接复制给事件，但并不能通过方法直接修改data中的数据，因为修改的数据并不能及时渲染到页面，所以需要通过this.setData方法将需要渲染的数据更新。

```
// pages/home/home.js
Page({
```

```
  data: {
    name: 'raoqi',
    age: 33,
    students: [
        {id: 1,name: '小张',age: 18},
        {id: 2,name: '小王',age: 17},
        {id: 3,name: '小李',age: 16},
        {id: 4,name: '小陈',age: 19}
    ],
    counter: 0
  },
  handleBtnClick: function () {
    console.log('点击测试')
    //不能直接更新数据
    // this.data.counter++;
    //需要通过this.setData来渲染页面上的数据更新
    this.setData({
      counter: this.data.counter + 1
    })
  },
  handleSubtraction() {
    this.setData({
      counter: this.data.counter - 1
    })
  }
})
```

```
<!--pages/home/home.wxml-->
<!-- 数据绑定 -->
<view>Hello {{name}}</view>
<view>age:{{age}}</view>
<!-- 列表展示 -->
<view wx:for='{{students}}'>{{item.name}}的年龄是{{item.age}}岁</view>
<!-- 事件监听 -->
<view>当前计数：{{counter}}</view>
<button size="mini" bindtap="handleSubtraction">-</button>
<button size="mini" bindtap="handleBtnClick">+</button>
```

MVVM

小程序也是使用MVVM架构。但与Vue（viewModel）不同的是，它是使用MINA框架（ViewModel）来连接视图层（VIew）和模型/逻辑层（Model）。

MVVM的两个作用是：

- DOM Listeners：ViewModel层可以将DOM的监听绑定到Model层
- Data Bindings：ViewModel层可以将数据的变化，响应式的渲染到View层

MVVM架构将我们从“命令式编程”（js/Jquery等）转移到“声明式编程”（React/Angular/vue等）。

配置小程序

小程序很多开发需求被规定在配置文件中。

- 这样做可以更有利于提高开发效率
- 可以保障开发出来的小程序的某些风格是比较统一的，例如导航栏、TapBar、页面路由等等。

常见配置文件包含以下文件。

- project.config.json：项目配置文件，比如项目名称、appid等，通常在微信开发者工具的"详情"中配置一次，保证团队开发的配置一致性。https://developers.weixin.qq.com/miniprogram/dev/devtools/projectconfig.html
- sitemap.json：小程序搜索相关。https://developers.weixin.qq.com/miniprogram/dev/framework/sitemap.html
- app.json：全局配置
- page.json：页面配置

全局配置app

文档地址：https://developers.weixin.qq.com/miniprogram/dev/reference/configuration/app.html

常用配置如下：

属性	类型	必填	描述	最低版本
pages	string[]	是	页面路径列表	
window	Object	否	全局的默认窗口表现	
tabBar	Object	否	底部 `tab` 栏的表现	

pages

用于指定小程序由哪些页面组成，每一项都对应一个页面的 路径（含文件名） 信息。文件名不需要写文件后缀，框架会自动去寻找对应位置的 `.json`, `.js`, `.wxml`, `.wxss` 四个文件进行处理。

数组的第一项代表小程序的初始页面（首页）。小程序中新增/减少页面，都需要对 pages 数组进行修改。

如开发目录为：

```
├── app.js
├── app.json
├── app.wxss
├── pages
|   |── index
|   |   ├── index.wxml
|   |   ├── index.js
|   |   ├── index.json
|   |   └── index.wxss
|   └── logs
|       ├── logs.wxml
|       └── logs.js
└── utils
```

则需要在 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	最低版本
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	
navigationStyle	string	default	导航栏样式，仅支持以下值：`default` 默认样式 `custom` 自定义导航栏，只保留右上角胶囊按钮。参见注 2。	微信客户端 6.6.0
backgroundColor	HexColor	#ffffff	窗口的背景色	
backgroundTextStyle	string	dark	下拉 loading 的样式，仅支持 `dark` / `light`	
backgroundColorTop	string	#ffffff	顶部窗口的背景色，仅 iOS 支持	微信客户端 6.5.16
backgroundColorBottom	string	#ffffff	底部窗口的背景色，仅 iOS 支持	微信客户端 6.5.16
enablePullDownRefresh	boolean	false	是否开启全局的下拉刷新。 详见 Page.onPullDownRefresh	
onReachBottomDistance	number	50	页面上拉触底事件触发时距页面底部距离，单位为 px。 详见 Page.onReachBottom	
pageOrientation	string	portrait	屏幕旋转设置，支持 `auto` / `portrait` / `landscape` 详见 响应显示区域变化	2.4.0 (auto) / 2.5.0 (landscape)

注 1：HexColor（十六进制颜色值），如"#ff00ff"

注 2：关于 `navigationStyle`

1. 客户端 7.0.0 以下版本，`navigationStyle` 只在 `app.json` 中生效。
2. 客户端 6.7.2 版本开始，`navigationStyle: custom` 对 web-view 组件无效
3. 开启 custom 后，低版本客户端需要做好兼容。开发者工具基础库版本切到 1.7.0（不代表最低版本，只供调试用）可方便切到旧视觉

如：

```
{
  "window": {
    "navigationBarBackgroundColor": "#0094ff",
    "navigationBarTextStyle": "white",
    "navigationBarTitleText": "小程序",
    "backgroundColor": "#eeeeee",
    "backgroundTextStyle": "light",
    "enablePullDownRefresh": false
  }
}
```

注3：`backgroundColor` 属性只有在IOS系统中自带下拉弹性效果时看到背景，其他系统需要通过设置 `enablePullDownRefresh:true` 来开启下拉加载更多功能查看背景颜色。

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。

属性	类型	必填	默认值	描述	最低版本
color	HexColor	是		tab 上的文字默认颜色，仅支持十六进制颜色	
selectedColor	HexColor	是		tab 上的文字选中时的颜色，仅支持十六进制颜色	
backgroundColor	HexColor	是		tab 的背景色，仅支持十六进制颜色	
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	tabBar 的位置，仅支持 `bottom` / `top`	
custom	boolean	否	false	自定义 tabBar，见详情	2.5.0

其中 list 接受一个数组，**只能配置最少 2 个、最多 5 个 tab。**tab 按数组的顺序排序，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	否	图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，不支持网络图片。 **当 `position` 为 `top` 时，不显示 icon。**
selectedIconPath	string	否	选中时的图片路径，icon 大小限制为 40kb，建议尺寸为 81px * 81px，不支持网络图片。 **当 `position` 为 `top` 时，不显示 icon。**

```
{
    "tabBar": {
      "list": [{
          "pagePath": "pagePath",
          "text": "text",
          "iconPath": "iconPath",
          "selectedIconPath": "selectedIconPath"
        }]
    }
}
```

> Sat May 09 2020 16:18:10 GMT+0800 (中国标准时间) app.json 文件内容错误
> VM824:1 app.json: ["tabBar"]["list"] 需至少包含 2 项

通过在app.json中键入tabBar来生成tabBar的结构，但直接保存时会报错，提示字少要包含两项。在根目录下创建assets文件夹，拷贝任意图标文件到该文件夹下，重新修改tabBar属性内容如下：

```
  "tabBar": {
    "selectedColor": "#0094ff",
    "list": [{
      "pagePath": "pages/home/home",
      "text": "首页",
      "iconPath": "/assets/1.jpg",
      "selectedIconPath": "/assets/1.jpg"
    },{
      "pagePath": "pages/about/about",
      "text": "关于",
      "iconPath": "/assets/2.jpg",
      "selectedIconPath": "/assets/2.jpg"
    }]
  }
```

`selectedColor`选中时文字的颜色，`pagePath`表示跳转的页面名称（完整路径），`text`显示文字，`iconPath`表示显示的图标（完整路径），`selectedIconPath`表示选中时显示的图标（完整路径）。

页面配置

- 单独配置about页面

在设置全局配置后，我们可以为每个页面单独配置某一个页面，，打开about目录下的对应文件设置如下代码，about.json:

```
{
  "usingComponents": {},
  "navigationBarBackgroundColor": "#0094ff",
  "navigationBarTextStyle": "white",
  "navigationBarTitleText": "关于我们"
}
```

```
/* pages/about/about.wxss */
text{
  font-size: 32px;
  color: cornflowerblue;
}
```

```
<!--pages/about/about.wxml-->
<text>关于页面独立配置</text>
```

运行后会发现页面配置的内容，覆盖掉了全局配置的内容，所以页面配置的优先级要高于全局配置。`usingComponents`用来设置自定义组件功能，后面讲组件功能时再详细描述。

- 添加编译模式

如果不是制作首页，每次保存编译后都需要点击调试的页面，这样比较麻烦，我们可以通过添加编译模式，自定义编译条件。选中“普通编译”旁边的小三角，在弹出的菜单中“点击添加编译模式”。这是程序会自动帮我们设置正在编辑的页面作为下次直接打开的地址。

小程序的双线程模型

“微信客户端”是小程序的宿主环境，宿主环境为了执行小程序的各种文件（.wxml/.wxss/.js），提供了小程序的双线程模型。

- 渲染层：通过WebView执行wxml布局文件，wxss样式文件。
- 逻辑层：通过JsCore执行JS文件（逻辑层）。

这两个线程都会经由微信客户端（Native）进行中转交互。

关于wxml布局文件，我们可以想象到，每个wxml等价于一颗DOM数，所以可以使用一个JS对象来模拟（虚拟DOM）。例如：

```
<!-- wxml布局 -->
<view>
  <view>a</view>
  <view>b</view>
  <view>c</view>
</view>
```

可以用下面的JS对象来表示。

```
//js对象
{
  name:"view",
  children:[
    {name:"view",children:[{text:"a"}]},
    {name:"view",children:[{text:"b"}]},
    {name:"view",children:[{text:"c"}]},
  ]
}
```

那么，WXML可以先转成JS对象，再渲染出真正的DOM树。通过setData把页面上的变量渲染成新的数据，数据发生变化的过程如下：

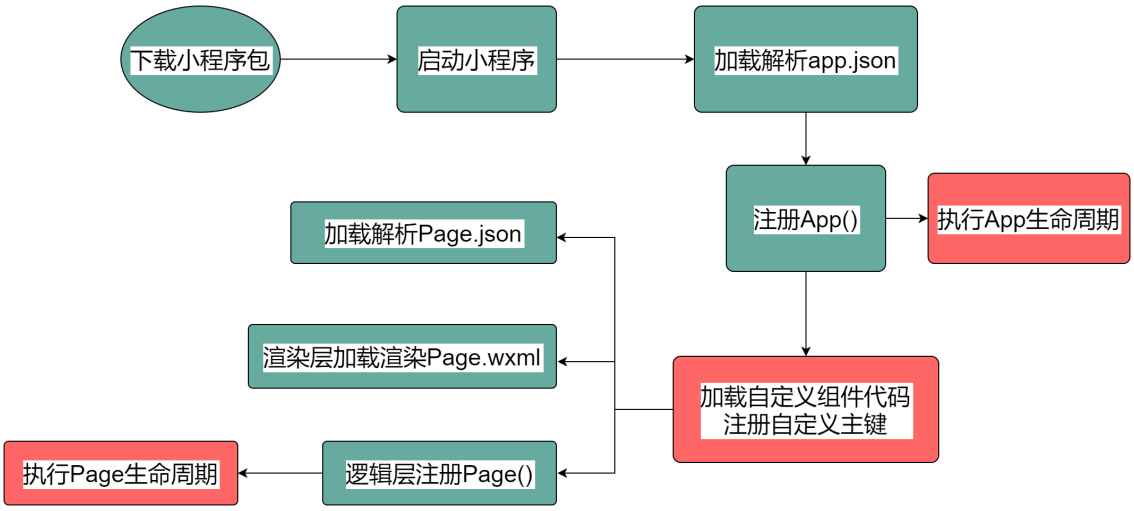
1. 产生的JS对象对应的节点就会发生变化
2. 此时可以对比前后两个JS对象得到变化的部分
3. 然后把这个差异应用到原来的DOM树上
4. 从而达到更新UI的目的，这就是“数据驱动”的原理

界面渲染整理流程：

1. 在渲染层，宿主环境会把WXML转化成对应的JS对象
2. 将JS对象再次转成真实DOM树，交由渲染层线程渲染
3. 数据变化时，逻辑层提供最新的变化数据，JS对象发生变化比较进行diff算法对比
4. 将最新变化的内容反映到真实的DOM树中，更新UI。

小程序启动流程

我们一起来看一下一个小程序的启动流程，通过了解小程序的启动流程，我们就能知道代码的执行顺序：



App(Object object)

注册小程序。接受一个 `Object` 参数，其指定小程序的生命周期回调等。

App() 必须在 `app.js` 中调用，必须调用且只能调用一次。不然会出现无法预期的后果。

app.js中的app()包含以下常用生命周期函数。

```
App({

  /**
   * 当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
   */
  onLaunch: function () {

  },

  /**
   * 当小程序启动，或从后台进入前台显示，会触发 onShow
   */
  onShow: function (options) {

  },

  /**
   * 当小程序从前台进入后台，会触发 onHide
   */
  onHide: function () {

  },

  /**
   * 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
   */
  onError: function (msg) {

  },
  //自定义的一个全局数据对象
  globalData: 'I am global data'
})
```

`globalData`变量可以修改成任意名称，它是一个自定义的全局对象，在早期时被创建在项目中使用。

除了自动生成的函数外，还包括其他一些函数，参考文档：https://developers.weixin.qq.com/miniprogram/dev/reference/api/App.html

App()参数

App(Object object)，App函数的参数是对象类型，包含以下内容：

属性	类型	默认值	必填	说明	最低版本
onLaunch	function		否	生命周期回调——监听小程序初始化。	
onShow	function		否	生命周期回调——监听小程序启动或切前台。	
onHide	function		否	生命周期回调——监听小程序切后台。	
onError	function		否	错误监听函数。	
onPageNotFound	function		否	页面不存在监听函数。	1.9.90
onUnhandledRejection	function		否	未处理的 Promise 拒绝事件监听函数。	2.10.0
onThemeChange	function		否	监听系统主题变化	2.11.0
其他	any		否	开发者可以添加任意的函数或数据变量到 `Object` 参数中，用 `this` 可以访问	

我们将在onLaunch函数中尝试获取用户信息，下面用一个例子来说明这些生命周期函数的作用。

- 例子，修改app.js中App(object)中的`onLaunch`函数：

```
onLaunch: function () {
  wx.getUserInfo({
    complete: (res) => {
      console.log(res);
    },
  })
}
```

当再次启动程序时，将会异步请求当前登录小程序的用户信息，控制台将显示当前登录的用户信息。

> {errMsg: "getUserInfo:ok", rawData: "{"nickName":"饶麒","gender":1,"language":"zh_CN","ci...yHPV9p7sibs7PPVADPbl4xV7adVIAF7oB91V27AskBw/132"}", userInfo: {...}, signature: "f166f68136bb35e5e757dc3babfe6b424c8ecf54", encryptedData: "yEiTxJ32iEZBxnxzlxs+fY/usFOfMO4lClUBNq/U3oHzOBKuRW...Zht9jfQKgmUAcGzz87x6zB2oRBDLNvtL/XS+0WVUpmcJ86g==", ...}

- 思考？在注册App()时，我们通常会做以下事情呢?
    1. 判断小程序的进入场景

2. 监听生性周期函数，在生命周期中执行对应的业务逻辑，比如在某个生命周期函数中获取微信用户的信息。
3. 因为App()实例只有一个，并且是全局共享的（单例对象），所以我们可以将一些共享数据存放其中。

- 前台/后台状态

1. 小程序启动后，界面被展示给用户，此时小程序处于**前台**状态。
2. 当用户点击右上角胶囊按钮关闭小程序，或者按了设备 Home 键离开微信时，小程序并没有完全终止运行，而是进入了**后台**状态，小程序还可以运行一小段时间。
3. 当用户再次进入微信或再次打开小程序，小程序又会从后台进入**前台**。但如果用户很久没有再进入小程序，或者系统资源紧张，小程序可能被**销毁**，即完全终止运行。

- 小程序启动

  小程序启动可以分为两种情况，一种是**冷启动**，一种是**热启动**。

  - 冷启动：如果用户首次打开，或小程序销毁后被用户再次打开，此时小程序需要重新加载启动，即冷启动。
  - 热启动：如果用户已经打开过某小程序，然后在一定时间内再次打开该小程序，此时小程序并未被销毁，只是从后台状态进入前台状态，这个过程就是热启动。

- 小程序销毁时机

  只有当小程序进入后台一定时间，或者系统资源占用过高，才会被销毁。具体而言包括以下几种情形：

  - 当小程序进入后台，可以维持一小段时间的运行状态，如果这段时间内都未进入前台，小程序会被销毁。
  - 当小程序占用系统资源过高，可能会被系统销毁或被微信客户端主动回收。
    - 在 iOS 上，当微信客户端在一定时间间隔内连续收到系统内存告警时，会根据一定的策略，主动销毁小程序，并提示用户 「运行内存不足，请重新打开该小程序」。具体策略会持续进行调整优化。
    - 建议小程序在必要时使用 wx.onMemoryWarning 监听内存告警事件，进行必要的内存清理。

小程序的打开场景

- 用户打开小程序时，场景可分为以下场景，我们通过在App()中的`onLaunch`或`onShow`方法中获取`options`参数查看`scene`属性即可获得对应的场景值。

```
/**
 * 当小程序启动，或从后台进入前台显示，会触发 onShow
 */
onShow: function (options) {
  console.log(options)
}
```

  重新运行小程序，控制台上会打印以下内容， **1001**代表场景值ID，可以参考对应的说明，了解小程序打开的场景。

> {path: "pages/about/about", query: {...}, scene: 1001, shareTicket: undefined, referrerInfo: {...}}

  文档地址：https://developers.weixin.qq.com/miniprogram/dev/reference/scene-list.html

场景值ID	说明	图例
1000	其他	/
1001	发现栏小程序主入口，「最近使用」列表（基础库2.2.4版本起包含「我的小程序」列表）	/
1005	微信首页顶部搜索框的搜索结果页	查看
1006	发现栏小程序主入口搜索框的搜索结果页	查看
1007	单人聊天会话中的小程序消息卡片	查看
1008	群聊会话中的小程序消息卡片	查看
1011	扫描二维码	查看
1012	长按图片识别二维码	查看
1013	扫描手机相册中选取的二维码	查看
1014	小程序模板消息	查看
1017	前往小程序体验版的入口页	查看
1019	微信钱包（微信客户端7.0.0版本改为支付入口）	查看
1020	公众号 profile 页相关小程序列表（已废弃）	查看
1022	聊天顶部置顶小程序入口（微信客户端6.6.1版本起废弃）	/
1023	安卓系统桌面图标	查看
1024	小程序 profile 页	查看
1025	扫描一维码	查看
1026	发现栏小程序主入口，「附近的小程序」列表	查看
1027	微信首页顶部搜索框搜索结果页「使用过的小程序」列表	查看

场景值 ID	说明	图例
1028	我的卡包	查看
1029	小程序中的卡券详情页	查看
1030	自动化测试下打开小程序	/
1031	长按图片识别一维码	查看
1032	扫描手机相册中选取的一维码	查看
1034	微信支付完成页	查看
1035	公众号自定义菜单	查看
1036	App 分享消息卡片	查看
1037	小程序打开小程序	查看
1038	从另一个小程序返回	查看
1039	摇电视	查看
1042	添加好友搜索框的搜索结果页	查看
1043	公众号模板消息	查看
1044	带 shareTicket 的小程序消息卡片 详情	查看
1045	朋友圈广告	查看
1046	朋友圈广告详情页	查看
1047	扫描小程序码	查看
1048	长按图片识别小程序码	查看
1049	扫描手机相册中选取的小程序码	查看

场景值ID	说明	图例
1052	卡券的适用门店列表	查看
1053	搜一搜的结果页	查看
1054	顶部搜索框小程序快捷入口（微信客户端版本6.7.4起废弃）	/
1056	聊天顶部音乐播放器右上角菜单	查看
1057	钱包中的银行卡详情页	查看
1058	公众号文章	查看
1059	体验版小程序绑定邀请页	/
1064	微信首页连Wi-Fi状态栏	查看
1067	公众号文章广告	查看
1068	附近小程序列表广告（已废弃）	/
1069	移动应用	查看
1071	钱包中的银行卡列表页	查看
1072	二维码收款页面	查看
1073	客服消息列表下发的小程序消息卡片	查看
1074	公众号会话下发的小程序消息卡片	查看
1077	摇周边	查看
1078	微信连Wi-Fi成功提示页	查看
1079	微信游戏中心	查看
1081	客服消息下发的文字链	查看

场景值ID	说明	图例
1082	公众号会话下发的文字链	查看
1084	朋友圈广告原生页	查看
1088	会话中查看系统消息，打开小程序	/
1089	微信聊天主界面下拉，「最近使用」栏（基础库2.2.4版本起包含「我的小程序」栏）	查看
1090	长按小程序右上角菜单唤出最近使用历史	查看
1091	公众号文章商品卡片	查看
1092	城市服务入口	查看
1095	小程序广告组件	查看
1096	聊天记录，打开小程序	查看
1097	微信支付签约原生页，打开小程序	查看
1099	页面内嵌插件	/
1102	公众号 profile 页服务预览	查看
1103	发现栏小程序主入口，「我的小程序」列表（基础库2.2.4版本起废弃）	/
1104	微信聊天主界面下拉，「我的小程序」栏（基础库2.2.4版本起废弃）	/
1106	聊天主界面下拉，从顶部搜索结果页，打开小程序	/
1107	订阅消息，打开小程序	/
1113	安卓手机负一屏，打开小程序（三星）	/
1114	安卓手机侧边栏，打开小程序（三星）	/
1124	扫“一物一码”打开小程序	/
1125	长按图片识别“一物一码”	/
1126	扫描手机相册中选取的“一物一码”	/
1129	微信爬虫访问 详情	/
1131	浮窗打开小程序	/

场景值 ID	说明	图例
1135	小程序资料页打开小程序	查看
1146	地理位置信息打开出行类小程序	查看
1148	卡包-交通卡，打开小程序	/
1150	扫一扫商品条码结果页打开小程序	查看
1153	“识物”结果页打开小程序	查看

对应的场景值非常的多，如果需要切换不同场景开发，可以点击微信开发者工具模拟器上的关闭圆圈按钮，此时会弹出以下界面供开发者选择测试场景。

在实际开发过程中，我们可以用代码来处理不同入口进入小程序的业务；

```
onShow: function (options) {
  switch (options.scene) {
```

```
      case 1001:
        //发现栏小程序入口业务
        break;
      case 1011:
        //扫描二维码业务
        break;
      case 1092:
        //城市服务入口业务
        break;
      ...
    }
  }
```

获取用户信息

1. wx.getUserInfo(Object object)调用前需要 用户授权 scope.userInfo。获取用户信息。

```
// 必须是在用户已经授权的情况下调用
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
  }
})
```

2. 通过按钮获取用户信息。通过按钮的两个特殊属性（事件），获取用户信息。（注意此处点击不能绑定**bindtap**事件）

```
  <!--pages/about/about.wxml-->
  <text>关于页面通过按钮获取用户信息</text>
  <button size='mini' open-type="getUserInfo"
bindgetuserinfo="handleGetUserInfo">获取用户信息</button>
```

```
// pages/about/about.js
Page({
  handleGetUserInfo(event) {
    console.log(event);

  }
})
```

点击按钮，控制台打印以下内容，其中**detail**属性中包含了用户信息。

> {type: "getuserinfo", timeStamp: 1944, target: {...}, currentTarget: {...}, mark: {...}, ...}
> type: "getuserinfo"
> timeStamp: 1944
> target: {id: "", offsetLeft: 128, offsetTop: 46, dataset: {...}}
> currentTarget: {id: "", offsetLeft: 128, offsetTop: 46, dataset: {...}}

> mark: {}
> detail: {errMsg: "getUserInfo:ok", rawData: "{"nickName":"饶
> 麒","gender":1,"language":"zh_CN","ci...
> yHPV9p7sibs7PPVADPbl4xV7adVIAF7oB91V27AskBw/132"}", signature:
> "f166f68136bb35e5e757dc3babfe6b424c8ecf54", encryptedData:
> "h3rljd15/DiyyVTn3LaTywHe+HPzG3HiBqwOAR98mDZKAJuixU...
> ObhNKKtqEPK9ctg9RnBa+vQUyteJEL2np6CPmhGcAANNRjQ==", iv:
> "KO6GoMk139I2Vk7jlHYtMA==", ...}
> mut: false
> _userTap: false
> **proto**: Object

3. open-data直接展示用户信息

文档位置：https://developers.weixin.qq.com/miniprogram/dev/component/open-data.html

通过在页面中直接声明，用于展示微信开放的数据。

```
<open-data type="userNickName"></open-data>
<open-data type="userAvatarUrl"></open-data>
<open-data type="userGender" lang="zh_CN"></open-data>
```

运行测试，会看到直接在页面上展示了当前用户的三个信息（昵称、头像、性别）。

4. AppObject **getApp(Object object)**

获取到小程序全局唯一的 `App` 实例。此处的`globalData`在**app.js**中声明了数据。

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

扩展：我们可以在监听小程序初始化`onLaunch`事件中，获取用户信息，并将值存入`globalData`中。这样就可以在全局通过`getApp()`来获取当前用户信息了。

注册页面

小程序中的每个页面，都有一个对应的js文件，其中调用Page方法注册页面示例。

- 在注册时，可以绑定初始化数据、生命周期回调、事件处理函数等。
- 文档位置：https://developers.weixin.qq.com/miniprogram/dev/reference/api/Page.html

```
// pages/other/other.js
Page({

  /**
   * 页面的初始数据
   */
  data: {
    message:'this a other page'
  },

  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {
```

```
    console.log("onLoad");
  },

  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function () {
    console.log("onReady");
  },

  /**
   * 生命周期函数--监听页面显示
   */
  onShow: function () {
    console.log("onShow");
  },

  /**
   * 生命周期函数--监听页面隐藏
   */
  onHide: function () {
    console.log("onHide");
  },

  /**
   * 生命周期函数--监听页面卸载
   */
  onUnload: function () {
    console.log("onUnload");
  },

  /**
   * 页面相关事件处理函数--监听用户下拉动作
   */
  onPullDownRefresh: function () {
    console.log("onPullDownRefresh");
  },

  /**
   * 页面上拉触底事件的处理函数
   */
  onReachBottom: function () {
    console.log("onReachBottom");
  },

  /**
   * 用户点击右上角分享
   */
  onShareAppMessage: function () {
    console.log("onShareAppMessage");
  }
})
```

尝试触发上面的监听函数。

发送网络请求、初始化页面

测试网络请求，在页面加载时，获取服务器端数据。在小程序中请求第三方网站，需要在小程序中配置安全域名，也可以在详情->本地设置中，勾选**不校验合法域名。**

例子：

```
Page({
  /**
   * 页面的初始数据
   */
  data: {
    message:'this a other page',
    list:[]
  },
  /**
   * 生命周期函数--监听页面加载
   */
  onLoad: function (options) {
    console.log("onLoad");
    wx.request({
      url: 'http://film.glkjjt.com/api/Movie/New',
      success: (res) => {
        console.log(res);
        this.setData({
          list:res.data
        })
      }
    })
  },
  ...
})
```

> Sat May 09 2020 23:58:06 GMT+0800 (中国标准时间) request 合法域名校验出错
> VM34 asdebug.js:1 如若已在管理后台更新域名配置，请刷新项目配置后重新编译项目，操作路径：“详情-域名信息”
> VM30:1 http://film.glkjjt.com 不在以下 request 合法域名列表中，请参考文档：https://developers.weixin.qq.com/miniprogram/dev/framework/ability/network.html

勾选**不校验合法域名**后显示：

> {data: Array(11), header: {…}, statusCode: 200, cookies: Array(1), errMsg: "request:ok"}
> data: Array(11)
> nv_length: (...)
> 0: {MovieID: 11, Name: "飞屋环游记", Actors: "爱德华·阿斯纳,乔丹·长井,鲍勃·彼德森", Type: "喜剧,动画,冒险", Rate: 8.9, ...}
> 1: {MovieID: 9, Name: "三傻大闹宝莱坞", Actors: "阿米尔·汗,黄渤,卡琳娜·卡普", Type: "剧情,喜剧,爱情,歌舞", Rate: 9.1, ...}
> 2: {MovieID: 7, Name: "泰坦尼克号", Actors: "莱昂纳多·迪卡普里奥,凯特·温丝莱特,比利·赞恩", Type: "历史,爱情,灾难", Rate: 9.5, ...}
> 3: {MovieID: 3, Name: "奇门遁甲", Actors: "大鹏,倪妮,李治廷", Type: "喜剧,动作,奇幻", Rate: 7.7, ...}
> 4: {MovieID: 4, Name: "帕丁顿熊2", Actors: "本·威士肖,杜江,休·格兰特", Type: "喜剧,动画,冒险", Rate: 9.6, ...}
> 5: {MovieID: 5, Name: "霸王别姬", Actors: "张国荣,张丰毅,巩俐", Type: "爱情,剧情", Rate: 9.6, ...}
> 6: {MovieID: 1, Name: "机器之血", Actors: "成龙,罗志祥,欧阳娜娜", Type: "动作,剧情", Rate: 8.8,

> ...}
> 7: {MovieID: 2, Name: "寻梦环游记", Actors: "安东尼·冈萨雷斯,本杰明·布拉特,盖尔·加西亚·贝纳尔", Type: "动画,冒险,家庭", Rate: 9.6, ...}
> 8: {MovieID: 6, Name: "肖申克的救赎", Actors: "蒂姆·罗宾斯,摩根·弗里曼,鲍勃·冈顿", Type: "犯罪,剧情", Rate: 9.5, ...}
> 9: {MovieID: 8, Name: "阿凡达", Actors: "萨姆·沃辛顿,佐伊·索尔达娜,米歇尔·罗德里格兹", Type: "动作,冒险,奇幻", Rate: 9, ...}
> 10: {MovieID: 10, Name: "神偷奶爸", Actors: "史蒂夫·卡瑞尔,杰森·席格尔,拉塞尔·布兰德", Type: "动画,家庭,喜剧", Rate: 9, ...}
> length: 11
> **proto**: Array(0)
> header: {Cache-Control: "no-cache", Pragma: "no-cache", Content-Type: "application/json; charset=utf-8", Expires: "-1", Server: "Microsoft-IIS/7.5", ...}
> statusCode: 200
> cookies: ["security_session_verify=b87ab05a8338540f69e5a584a2...ires=Tue 12-May-20 23:59:29 GMT; path=/; HttpOnly"]
> errMsg: "request:ok"
> **proto**: Object

```
<!--pages/other/other.wxml-->
<text>pages/other/other.wxml</text>
<view wx:for='{{list}}'>{{item.Name}}</view>
```

运行小程序，在other.wxml上查看获取到的服务器数据。

补充：网络请求时使用**箭头函数**方法实现**success**的调用，其中**this**关键字会向上一层一层的寻找data数据，但如果将**success**用`function(res){}`来声明，在function函数中使用**this**，**this**就会变成`undefined`，必须得在网络请求之前，将**this**用**_that**保存起来。

监听事件函数

```
// pages/other/other.js
...
  handleTextClick() {
    this.setData({
      message: '点击文字修改Message'
    })
  }, handleBtnClick() {
    this.setData({
      message: '点击按钮修改Message'
    })
  }
...
```

```
<!--pages/other/other.wxml-->
<text bindtap="handleTextClick">{{message}}</text>
<button bindtap="handleBtnClick">监听点击事件</button>
<view wx:for='{{list}}'>{{item.Name}}</view>
```

运行小程序，测试点击效果。

其他监听

常用的监听还有比如**页面滚动、上拉刷新、下拉加载更多。**

```
// pages/other/other.js
...
  /**
   * 页面相关事件处理函数--监听用户下拉动作
   */
  onPullDownRefresh: function () {
    console.log("onPullDownRefresh");
  },
  /**
   * 页面上拉触底事件的处理函数
   */
  onReachBottom: function () {
    console.log("onReachBottom");
  },
 /**
  * 监听用户滑动页面事件的处理函数
  */
  onPageScroll(obj) {
    console.log(obj);
  },
...
```

模拟对应动作，完成事件的触发。

组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/

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。

1. 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一致的样式。
- 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

2. 属性类型

类型	描述	注解
Boolean	布尔值	组件写上该属性，不管是什么值都被当作 `true`；只有组件上没有该属性时，属性值才为 `false`。 如果属性值为变量，变量的值会被转换为Boolean类型
Number	数字	`1`, `2.5`
String	字符串	`"string"`
Array	数组	`[ 1, "string" ]`
Object	对象	`{ key: value }`
EventHandler	事件处理函数名	`"handlerName"` 是 Page 中定义的事件处理函数名
Any	任意属性	

3. 公共属性

所有组件都有以下属性：

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

4. 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。

text组件

- text

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/text.html

1. 基本使用

```
<!-- 1. 基本使用 -->
<text class='title'>Hello Wrold!!\n</text>
<text>你好,小程序</text>
<text>text是行内元素\n</text>
```

2. selectable长按可选文本

```
<!-- 2. selectable:true -->
<!-- 默认情况下text中的文本长按是不能选中的 -->
<text selectable="{{true}}">Hi ,长按选择当前文本\n</text>
<text selectable>Hi ,selectable属性简写\n</text>
```

3. space决定文本空格大小

```
<!-- 3.space：决定文本空格的大小 -->
<text>Hello World\n</text>
<text space="nbsp">Hello World\n</text>
<text space="ensp">Hello World\n</text>
<text space="emsp">Hello World\n</text>
```

4.decode：是否解码文本

```
<!-- 4.decode：是否解码文本 -->
<text decode="{{true}}">&lt;p&gt;\n</text>
<text decode="{{false}}">&lt;p&gt;\n</text>
<text decode>&lt;p&gt;\n</text>
<text>&lt;p&gt;\n</text>
```

button组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/button.html

按钮。

1. button的基本使用

```
<!-- 1.button的基本使用 -->
<button>按钮</button>
```

2. size按钮的大小

```
<!-- 2.size属性：mini -->
<button size="mini">按钮</button>
<button size="mini">按钮</button>
```

3. type按钮的样式

```
<!-- 3.type属性: 颜色-->
<button size="mini" type="default">按钮</button>
<button size="mini" type="primary">按钮</button>
<button size="mini" type="warn">按钮</button>
```

4. plain 按钮是否镂空，背景色透明

```
<!-- 4.plain:按钮是否镂空，背景色透明 -->
<button size="mini" plain>按钮</button>
```

5. 是否禁用

```
<!-- 5.disabled:是否禁用 -->
<button size="mini" disabled>按钮</button>
```

6. loading名称前是否带 loading 图标

```
<!-- 6.loading:名称前是否带 loading 图标 -->
<button size="mini" loading="{{true}}">按钮</button>
```

7. hover-class:指定按钮按下去的样式类

```
<!-- 7.hover-class:指定按钮按下去的样式类 -->
<button size="mini" hover-class="hover">按钮</button>
```

view组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/view.html

视图容器

1. view的基本使用

```
/* pages/view/view.wxss */
.box{
  background-color: black;
  color: aliceblue;
}
view{
  margin-bottom: 5px;
}
.big{
  border: 1px solid black;
  width: 100px;
  height: 100px;
}
.small{
  border: 1px solid red;
  width: 50px;
  height: 50px;
}
```

```
<!-- 1.view的基本使用 -->
<view>
    <text>文本</text>
</view>
<view>块级元素</view>
<view class='box'>带样式view</view>
```

2. hover-class

hover-class：指定按下去的样式类

hover-stay-time：手指松开后点击态保留时间，单位毫秒

hover-start-time：按住后多久出现点击态，单位毫秒

```
<!-- 2.hover-class -->
<view hover-class="box" hover-stay-time="100" hover-start-time="0">带有hover-class属性的view</view>
```

3. hover-stop-propagation指定是否阻止本节点的祖先节点出现点击态（拒绝冒泡）

```
<!-- 3.hover-stop-propagation -->
<view class="big" hover-class="box">
    <view class="small" hover-stop-propagation>small</view>
</view>
```

image组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/image.html

图片。支持 JPG、PNG、SVG、WEBP、GIF 等格式。

- image组件默认宽度320px、高度240px
- image组件中二维码/小程序码图片不支持长按识别。仅在wx.previewImage中支持长按识别
- image组件是一个行内块级元素（inline-block）

1. image组件可以写成单标签或双标签

```
<!-- 1.image组件可以写成单标签或双标签 -->
<image src="/assets/1.jpg"></image>
<image src="/assets/1.jpg" />
```

2. 远程图片

```
<!-- 2.远程图片 -->
<image src="https://res.wx.qq.com/wxdoc/dist/assets/img/0.4cb08bb4.jpg"></image>
```

3. 相册图片或拍照（拍照需要真机调试）

```
<!-- 3.相册图片 -->
<image src="{{imagePath}}"></image>
<button bindtap="handleChooseAlbum">选择图片</button>
```

```
// pages/image/image.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    imagePath:''
  },
  // 选择图片
  handleChooseAlbum()
  {
```

```
    // 系统api，选择相册或者拍照
    wx.chooseImage({
      complete: (res) => {
        //测试获取的图片对象
        console.log(res);
        //设置图片路径
        this.setData({
          imagePath:res.tempFilePaths[0]
        })
      },
    })
  }
})
```

4. lazy-load懒加载图片

图片懒加载，在即将进入一定范围（上下三屏）时才开始加载，通过监听图片加载`bindload`事件测试懒加载效果。

```
<!-- 4.lazy-load -->
<image src="/assets/1.jpg" />
<image src="/assets/1.jpg" />
<image src="/assets/1.jpg" />
<image src="https://res.wx.qq.com/wxdoc/dist/assets/img/0.4cb08bb4.jpg"
bindload="handleImageLoad" lazy-load></image>
```

```
  handleImageLoad()
  {
    console.log('图片加载完成');
  }
```

5. show-menu-by-longpress开启长按图片显示识别小程序码菜单

这里提供一张测试图片，图片地址：https://s1.ax1x.com/2020/05/10/Y8Zx2Q.jpg，测试效果也需要在**预览**中生成真机调试二维码测试，

```
<!-- 5.长按图片出现识别小程序码 -->
<image src="https://s1.ax1x.com/2020/05/10/Y8Zx2Q.jpg" show-menu-by-longpress>
</image>
```

6. mode图片裁剪、缩放的模式

mode 的合法值

值	说明	最低版本
scaleToFill	缩放模式，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	缩放模式，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	缩放模式，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。	
widthFix	缩放模式，宽度不变，高度自动变化，保持原图宽高比不变	
heightFix	缩放模式，高度不变，宽度自动变化，保持原图宽高比不变	2.10.3
top	裁剪模式，不缩放图片，只显示图片的顶部区域	
bottom	裁剪模式，不缩放图片，只显示图片的底部区域	
center	裁剪模式，不缩放图片，只显示图片的中间区域	
left	裁剪模式，不缩放图片，只显示图片的左边区域	
right	裁剪模式，不缩放图片，只显示图片的右边区域	
top left	裁剪模式，不缩放图片，只显示图片的左上边区域	
top right	裁剪模式，不缩放图片，只显示图片的右上边区域	
bottom left	裁剪模式，不缩放图片，只显示图片的左下边区域	
bottom right	裁剪模式，不缩放图片，只显示图片的右下边区域	

例子：缩放模式，宽度不变，高度自动变化，保持原图宽高比不变（其他效果自行测试，或查阅文档）

```
<!-- 6.图片缩放裁剪 -->
<image src="https://s1.ax1x.com/2020/05/10/Y8Zx2Q.jpg" mode='widthFix'></image>
```

input组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/input.html

输入框。该组件是原生组件，使用时请注意相关限制

1. input基本使用

```
/* pages/input/input.wxss */
input{
  border: 1px solid black;
  margin:15px 25px;
}
```

```
<!-- 1.input基本使用 -->
<input></input>
<input/>
```

2. value输入框的初始内容

```
<!-- 2.value -->
<input value="hello"></input>
```

3. type 弹出键盘的类型（真机测试）

```
<!-- 3.typeinput 的类型 -->
<input type="number"></input>
```

4. password是否为密码类型

```
<!-- 4.password: 是否为密码类型 -->
<input password></input>
```

5. placeholder输入框为空时占位符

```
<!-- 5.placeholder:输入框为空时占位符 -->
<input placeholder="在此输入文字"></input>
```

6. disabled是否禁用

```
<!-- 6.disabled:是否禁用 -->
<input disabled></input>
```

7. confirm-type：设置键盘右下角按钮的文字，仅在type='text'时生效。

```
<!-- 7.confirm-type 设置键盘右下角按钮的文字，仅在type='text'时生效 -->
<input confirm-type='search' type="text"></input>
```

confirm-type 的合法值

值	说明	最低版本
send	右下角按钮为“发送”	
search	右下角按钮为“搜索”	
next	右下角按钮为“下一个”	
go	右下角按钮为“前往”	
done	右下角按钮为“完成”	

8. input事件绑定

```
// pages/input/input.js
Page({
  data: {
```

```
  },
  handleInput(event){
    console.log(event);
  },
  handleFocus(event){
    console.log(event);
  },
  handleBlur(event){
    console.log(event);
  }
})
```

```
<!-- 8.input绑定事件 -->
<input bindfocus="handleFocus" bindblur="handleBlur" bindinput="handleInput">
</input>
```

出发后，控制台显示如下：

> {type: "focus", timeStamp: 1371, target: {...}, currentTarget: {...}, detail: {...}}
> {type: "input", timeStamp: 1589116223324, detail: {...}, target: {...}, currentTarget: {...}, ...}
> {type: "blur", timeStamp: 3701, target: {...}, currentTarget: {...}, detail: {...}}

scroll-view组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/component/scroll-view.html

可滚动视图区域。使用竖向滚动时，需要给scroll-view一个固定高度，通过 WXSS 设置 height。组件属性的长度单位默认为px，2.4.0起支持传入单位(rpx/px)。

1. scroll-x 水平滚动

```
<!--pages/scroll/scroll.wxml-->
<!-- 1.水平滚动:scroll-x -->
<scroll-view class='container1' scroll-x>
<view wx:for="{{10}}" class='item1'>{{item}}</view>
</scroll-view>
```

```
/* pages/scroll/scroll.wxss */
.container1{
  background-color: blanchedalmond;
  /* 不要换行 */
  white-space: nowrap;
}
.item1{
  width: 100px;
  height: 100px;
  background-color: chocolate;
  margin:10px;
  /* 行内块级元素，横向排列，宽度不够换行显示 */
  display: inline-block;
}
```

2. scroll-y 垂直滚动

```
<!-- 2.垂直滚动:scroll-y -->
<scroll-view class='container2' scroll-y>
<view wx:for="{{10}}" class='item2'>{{item}}</view>
</scroll-view>
```

```
.container2{
  margin-top: 20px;
  background-color: lavender;
  height: 230px;
}
.item2{

  height: 100px;
  background-color: lawngreen;
  margin:10px;
}
```

3. 事件补充

```
<!-- 3.其他补充 -->
<scroll-view class='container2' scroll-y bindscroll='handleScroll'>
<view wx:for="{{10}}" class='item2'>{{item}}</view>
</scroll-view>
```

```
// pages/scroll/scroll.js
Page({

  /**
   * 页面的初始数据
   */
  data: {

  },
  handleScroll(event) {
    console.log(event);
  }

})
```

垂直滚动的基础上，获取滚动的数据参数。

> {type: "scroll", timeStamp: 2300, target: {...}, currentTarget: {...}, detail: {...}}*
>
> type: "scroll"
>
> timeStamp: 2300
>
> target: {id: "", offsetLeft: 0, offsetTop: 390, dataset: {...}}
>
> currentTarget: {id: "", offsetLeft: 0, offsetTop: 390, dataset: {...}}
>
> detail: {scrollLeft: 0, scrollTop: 4.800000190734863, scrollHeight: 1110, scrollWidth: 320, deltaX: 0, ...}
>
> **proto**: Object

其他事件可以参考官方文档。

wxss&wxml&wxs

wxss

文档地址：https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxss.html

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，WXSS 具有 CSS 大部分特性。同时为了更适合开发微信小程序，WXSS 对 CSS 进行了扩充以及修改。

与 CSS 相比，WXSS 扩展的特性有：

- 尺寸单位
- 样式导入

1. 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750rpx = 375px = 750物理像素，1rpx = 0.5px = 1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	1rpx = 0.42px	1px = 2.34rpx
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6 Plus	1rpx = 0.552px	1px = 1.81rpx

```
<!-- 前端也需要进行配置尺寸的适配：em/rem/vw/vh -->
<view class='box1'></view>
<view class='box2'></view>
<view class='content1'>文字大小测试</view>
<view class='content2'>文字大小测试</view>
```

```
/* iphone6的设备上，rpx的值是px的一半 */
/* 随着设备宽高的调整，rpx会自动调整大小 */
.box2{
  width: 200rpx;
  height: 200rpx;
  background-color: rgb(154, 91, 179);
}
/* iphone6的设备上，文字值rpx比px大一倍 */
.content1{
  font-size: 24px;
}
.content2{
  font-size: 48rpx;
}
```

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。

2. 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;
}
```

页面样式

- 页面样式三种写法

```
<!--pages/wxss/wxss.wxml-->
<!-- 1.行内（内联）样式 -->
<view style="color:red;font-size:24px;">行内（内联）样式</view>

<!-- 2.页内样式 -->
<view class='box'>页内样式</view>

<!-- 3.全局样式 -->
<view class='container'>全局样式</view>
```

```
/* pages/wxss/wxss.wxss */
.box{
  color: aqua;
  font-size: 24px;
}
```

```
/* app.wxss */
.container{
  color: brown;
  font-size: 24px;
}
```

- 如果有相同的样式

优先级依次是：**行内样式 > 页面样式 > 全局样式**

选择器

文档地址：https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxss.html#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 class="intro" 的组件
#id	`#firstname`	选择拥有 id="firstname" 的组件
element	`view`	选择所有 view 组件
element, element	`view, checkbox`	选择所有文档的 view 组件和所有的 checkbox 组件
::after	`view::after`	在 view 组件后边插入内容
::before	`view::before`	在 view 组件前边插入内容

样式权重：例如在全局样式中，对某个样式属性增加`!important`就能让此样式生效。

权重设置	权重值
!important	无限大
style=""	1000
#id	100
.class	10
element	1

样式库

Github地址：https://github.com/tencent/weui-wxss

- 使用官方样式库，来实现获取一个效果

1. 通过微信开发者工具导入下载解压后的**dist**目录。
2. 查阅需要的某个效果（例如反馈**toast**）
3. 拷贝根目录下的三个目录`libs`、`mixin`、`style`，以及`app.wxss`全局样式到本地项目
4. 找到**toast**目录，分别将**js、wxss、wxml**中的代码拷贝到当前页面上，即可使用。

wxml

文档地址：https://developers.weixin.qq.com/miniprogram/dev/reference/wxml/data.html

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：

1. 内容

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

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

2. 组件属性(需要在双引号之内)

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

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

3. 控制属性(需要在双引号之内)

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

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

4. 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：

1. 三元运算

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

2. 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`。

3. 逻辑判断

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

4. 字符串运算

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

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

5. 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。

1. 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]`。

2. 对象

```
<!--pages/wxml/wxml.wxml-->
<!-- 定义模板 -->
<template name='tmp'>
<view>{{for}}</view>
<view>{{bar}}</view>
</template>

<!-- 使用模板，并通过data传参 -->
<template is='tmp' data="{{for: a, bar: b}}"></template>
```

```
// pages/wxml/wxml.js
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

```
<!--pages/wxml/wxml.wxml-->
<!-- 定义模板 -->
<template  name='tmp'>
    <view>{{a}}</view>
    <view>{{b}}</view>
    <view>{{c}}</view>
    <view>{{d}}</view>
    <view>{{e}}</view>
</template>

<!-- 使用模板，并通过data传参 -->
<template is='tmp'  data="{{...obj1, ...obj2, e: 5}}"></template>
```

```
// pages/wxml/wxml.js
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。

```
<!--pages/wxml/wxml.wxml-->
<!-- 定义模板 -->
<template name='tmp'>
<view>{{foo}}</view>
<view>{{bar}}</view>
</template>

<!-- 使用模板，并通过data传参 -->
<template is="tmp" data="{{foo, bar}}"></template>
```

```
// pages/wxml/wxml.js
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：

```
<!--pages/wxml/wxml.wxml-->
<!-- 定义模板 -->
<template name='tmp'>
<view>{{a}}</view>
<view>{{b}}</view>
<view>{{c}}</view>
</template>

<!-- 使用模板，并通过data传参 -->
<template is="tmp" data="{{...obj1, ...obj2, a, c: 6}}"></template>
```

```
// pages/wxml/wxml.js
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>
```

条件渲染

1. wx:if

在框架中，使用 `wx:if=""` 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>
```

2. 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。

3. `wx:if` vs `hidden`

因为 `wx:if` 之中的模板也可能包含数据绑定，所以当 `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

1. 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="{{array}}">
  {{index}}: {{item.message}}
</view>
```

```
Page({
  data: {
    array: [{
      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>
```

2. 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>
```

3. 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

1. 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。
2. 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。

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。

示例代码：

```
<switch wx:for="{{objectArray}}" wx:key="unique" style="display: block;">
{{item.id}} </switch>
<button bindtap="switch"> Switch </button>
<button bindtap="addToFront"> Add to the front </button>

<switch wx:for="{{numberArray}}" wx:key="*this" style="display: block;">
{{item}} </switch>
<button bindtap="addNumberToFront"> Add to the front </button>
```

```
Page({
  data: {
    objectArray: [
      {id: 5, unique: 'unique_5'},
      {id: 4, unique: 'unique_4'},
      {id: 3, unique: 'unique_3'},
      {id: 2, unique: 'unique_2'},
      {id: 1, unique: 'unique_1'},
      {id: 0, unique: 'unique_0'},
    ],
    numberArray: [1, 2, 3, 4]
  },
  switch: function(e) {
    const length = this.data.objectArray.length
    for (let i = 0; i < length; ++i) {
      const x = Math.floor(Math.random() * length)
      const y = Math.floor(Math.random() * length)
      const temp = this.data.objectArray[x]
      this.data.objectArray[x] = this.data.objectArray[y]
      this.data.objectArray[y] = temp
    }
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addToFront: function(e) {
    const length = this.data.objectArray.length
    this.data.objectArray = [{id: length, unique: 'unique_' +
length}].concat(this.data.objectArray)
    this.setData({
      objectArray: this.data.objectArray
    })
  },
  addNumberToFront: function(e){
    this.data.numberArray = [ this.data.numberArray.length + 1
].concat(this.data.numberArray)
    this.setData({
      numberArray: this.data.numberArray
    })
```

```
  }
})
```

注意：**

当 `wx:for` 的值为字符串时，会将字符串解析成字符串数组

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

等同于

```
<view wx:for="{{['a','r','r','a','y']}}">
  {{item}}
</view>
```

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>
```

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。

1. 定义模板

使用 **name** 属性，作为模板的名字。然后在``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>
```

2. 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: '2020-05-11'
    }
  }
})
```

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>
```

3. 模板的作用域

模板拥有自己的作用域，只能使用 **data** 传入的数据以及模板定义文件中定义的 `wxs` 模块。

引用

WXML 提供两种文件引用方式`import`和`include`。

1. 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：

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

2. import 的作用域

`import` 有作用域的概念，即只会 import（引用） 目标文件中定义的 template，而不会 import （引用）目标文件 import（引用） 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"/> <!-- 引入模板A，可以在当前wxml文件中使用模板A，但不能将模板A传递到其他wxml文件中 -->
<template name="B">
  <text> B template </text>
</template>
```

```
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/>  <!-- 错误！ 这里不能使用模板A，如需使用，需要手动导入模板A -->
<template is="B"/>
```

3. include

`include` 可以将目标文件**除了** `template`、`wxs` **外**的整个代码引入，相当于是拷贝到 `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>
```

WXS

文档地址：https://developers.weixin.qq.com/miniprogram/dev/reference/wxs/

WXS（WeiXin Script）是小程序的一套脚本语言，结合 `WXML`，可以构建出页面的结构。

注意

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

- 页面渲染

```
<!--pages/wxs/wxs.wxml-->
<wxs module="m1">
var msg = "hello world";
<!-- 模块化导出 -->
module.exports.message = msg;
</wxs>

<view> {{m1.message}} </view>
```

页面输出：

> hello world

- 数据处理

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

```
<!--pages/wxs/wxs.wxml-->
<!-- 下面的 getMax 函数，接受一个数组，且返回数组中最大的元素的值 -->
<wxs module="m1">
var getMax = function(array) {
  var max = undefined;
  for (var i = 0; i < array.length; ++i) {
    max = max === undefined ?
      array[i] :
      (max >= array[i] ? max : array[i]);
  }
  return max;
}
<!-- 模块化导出 -->
module.exports.getMax = getMax;
</wxs>

<!-- 调用 wxs 里面的 getMax 函数，参数为 page.js 里面的 array -->
<view> {{m1.getMax(array)}} </view>
```

页面输出：

> 5

应用

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。

- 模块

每一个 `.wxs` 文件和 `wxs` 标签都是一个单独的模块。

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。

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。

- .wxs 文件

在**微信开发者工具**里面，右键可以直接创建 `.wxs` 文件，在其中直接编写 WXS 脚本。

1. **例子**

格式化价格。

在根目录下创建**wxs目录**，新建**format.wxs文件**用于封装`wxs`功能。

```
// /wxs/format.wxs
function priceFormat(price, len) {
  // 如果没有传入，默认为2
  len = len || 2;
  //确保类型是小数
  var f_price = parseFloat(price);
  //保留指定小数位
  return f_price.toFixed(len);
}

//导出功能
module.exports = {
  priceFormat: priceFormat
}
```

在**wxs.js**中定义初始化价格

```
// pages/wxs/wxs.js
Page({
  data: {
    price:33.3333333333
  }
})
```

在wxs.wxml页面中，首先引入`wxs`文件，然后才能使用`priceFormat`的功能，如有必要可以添加第二个参数用于控制小数位数。

注意：此处不能使用绝对路径，必须使用相对路径引用wxs文件。

```
<!--pages/wxs/wxs.wxml-->

<!-- 引入wxs -->
<wxs src='../../wxs/format.wxs' module='format' />

<!-- 使用wxs中定义的方法 -->
<view>您需要支付的价格：{{format.priceFormat(price)}}</view>
```

- `wxs`标签

属性名	类型	默认值	说明
module	String		当前 `wxs` 标签的模块名。必填字段。
src	String		引用 .wxs 文件的相对路径。仅当本标签为**单闭合标签**或**标签的内容为空**时有效。

- **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：

1. 首字符必须是：字母（a-zA-Z），下划线（_）
2. 剩余字符可以是：字母（a-zA-Z），下划线（_），数字（0-9）

- **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：

1. 只能引用 `.wxs` 文件模块，且必须使用相对路径。
2. `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。
3. 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。

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。

事件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxml/event.html

常用事件

类型	触发条件	最低版本
touchstart	手指触摸动作开始	
touchmove	手指触摸后移动	
touchend	手指触摸动作结束	
tap	手指触摸后马上离开	
longpress	手指触摸后，超过350ms再离开，如果指定了事件回调函数并触发了这个事件，tap事件将不被触发	1.5.0

1. 例子

```
<!--pages/event/event.wxml-->
<view style="width:200rpx;height:200rpx;background-color:#534595"
bindtouchstart="handletouchstart"
bindtouchmove="handletouchmove"
bindtouchend="handletouchend"
bindtap="handletap"
bindlongpress="handlelongpress"
></view>
```

```
// pages/event/event.js
Page({
  handletouchstart() { console.log("handletouchstart"); },
  handletouchmove() { console.log("handletouchmove"); },
  handletouchend() { console.log("handletouchend"); },
  handletap() { console.log("handletap"); },
  handlelongpress() { console.log("handlelongpress"); }
})
```

短触：

> handletouchstart
> handletouchend
> handletap

长触：

> handletouchstart
> handlelongpress
> handletouchend

触摸移动：

> handletouchstart
> 33 handletouchmove
> handletouchend

注意：longpress与tap两个事件互斥。

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。

```
<button bindtap="handleEventClick">事件对象</button>
```

```
// pages/event/event.js
Page({
handleEventClick(event){
    console.log(event);
  }
})
```

控制台输出：

> {type: "tap", timeStamp: 976, target: {...}, currentTarget: {...}, detail: {...}, ...}
> type: "tap"
> timeStamp: 976
> target: {id: "", offsetLeft: 0, offsetTop: 85, dataset: {...}}
> currentTarget: {id: "", offsetLeft: 0, offsetTop: 85, dataset: {...}}
> detail: {x: 166.40000915527344, y: 108.00001525878906}
> touches: [{...}]
> changedTouches: [{...}]
> _requireActive: true
> **proto**: Object

事件对象属性列表：

属性	类型	说明	基础库版本
type	String	事件类型	
timeStamp	Integer	事件生成时的时间戳	
target	Object	触发事件的组件的一些属性值集合	
currentTarget	Object	当前组件的一些属性值集合	
detail	Object	额外的信息	
touches	Array	触摸事件，当前停留在屏幕中的触摸点信息的数组	
changedTouches	Array	触摸事件，当前变化的触摸点信息的数组	

touches

touches 是一个数组，每个元素为一个 Touch 对象（canvas 触摸事件中携带的 touches 是 CanvasTouch 数组）。 表示当前停留在屏幕上的触摸点。

changedTouches

changedTouches 数据格式同 touches。 表示有变化的触摸点，如从无变有（touchstart），位置变化（touchmove），从有变无（touchend、touchcancel）。

target

触发事件的源组件。

currentTarget

事件绑定的当前组件。

事件参数

所有组件都有以下属性：

属性名	类型	描述	注解
data-*	Any	自定义属性	组件上触发的事件时，会发送给事件处理函数

通过**data-***来传递事件参数

```
/* pages/event/event.wxss */
.item
{
  border: 1px solid #808000;
  margin: 50rpx 50rpx;
  text-align: center;
}
```

```
// pages/event/event.js
Page({
  data:{
    list:["A","B","C"]
  },
  handleItemClick(event){
    console.log(event)
    // 获取data-传递过来的参数
    console.log(event.currentTarget.dataset.index);
    console.log(event.currentTarget.dataset.item);
  }
}
```

```
<!--pages/event/event.wxml-->
<view>
    <block wx:for="{{list}}" wx:key="*this">
        <view class="item"
              bindtap="handleItemClick"
              data-index="{{index}}"
              data-item="{{item}}"
              >{{item}}</view>
    </block>
</view>
```

通过点击不同的按钮，控制台打印如下：

> {type: "tap", timeStamp: 790091, target: {...}, currentTarget: {...}, detail: {...}, ...}type: "tap"timeStamp: 790091target: {id: "", offsetLeft: 21, offsetTop: 152, dataset: {...}}currentTarget: id: ""offsetLeft: 21offsetTop: 152dataset: index: 0item: "A"**proto**: Object**proto**: Objectdetail: {x: 180, y: 164.8000030517578}touches: [{...}]changedTouches: [{...}]_requireActive: true**proto**: Object
> 0
> A
> {type: "tap", timeStamp: 791565, target: {...}, currentTarget: {...}, detail: {...}, ...}
> 1
> B
> {type: "tap", timeStamp: 792860, target: {...}, currentTarget: {...}, detail: {...}, ...}
> 2
> C

冒泡和捕获

除 `bind` 外，也可以用 `catch` 来绑定事件。与 `bind` 不同， `catch` 会阻止事件向上冒泡。

```
/* pages/event/event.wxss */
#outer {
  width: 500rpx;
  height: 500rpx;
  background-color: cornflowerblue;
  margin: auto;
}

#middle {
  width: 300rpx;
  height: 300rpx;
```

```
  background-color: rgb(200, 95, 206);
  margin: auto;
}

#inner {
  width: 150rpx;
  height: 150rpx;
  background-color: rgb(163, 230, 109);
  margin: auto;
}
```

```
<!--pages/event/event.wxml-->
<!-- bind:会逐级触发 -->
<!-- catch:阻止下一级触发 -->
<view id="outer" bindtap="handleTap1"  capture-bind:tap='captureTap1'>
  outer view
  <view id="middle" bindtap="handleTap2"  capture-bind:tap='captureTap2'>
    middle view
    <view id="inner" bindtap="handleTap3"  capture-bind:tap='captureTap3'>
      inner view
    </view>
  </view>
</view>
```

```
// pages/event/event.js
// 冒泡方法
  handleTap1() { console.log("outerbindTap1"); },
  handleTap2() { console.log("middlebindTap2"); },
  handleTap3() { console.log("innerbindTap3"); },
  // 捕获方法
  captureTap1() { console.log("outercaptureTap1"); },
  captureTap2() { console.log("middlecaptureTap2"); },
  captureTap3() { console.log("innercaptureTap3"); }
```

点击`#inner`标签，依次触发下列6个事件。

> outercaptureTap1
> middlecaptureTap2
> innercaptureTap3
> innerbindTap3
> middlebindTap2
> outerbindTap1

观察事件触发的顺序，依次由外至内实现`capture-bind`（捕获）然后由内至外执行`bind`（冒泡）,那么修改`bind`为`catch`，可以阻止后一级事件的触发。

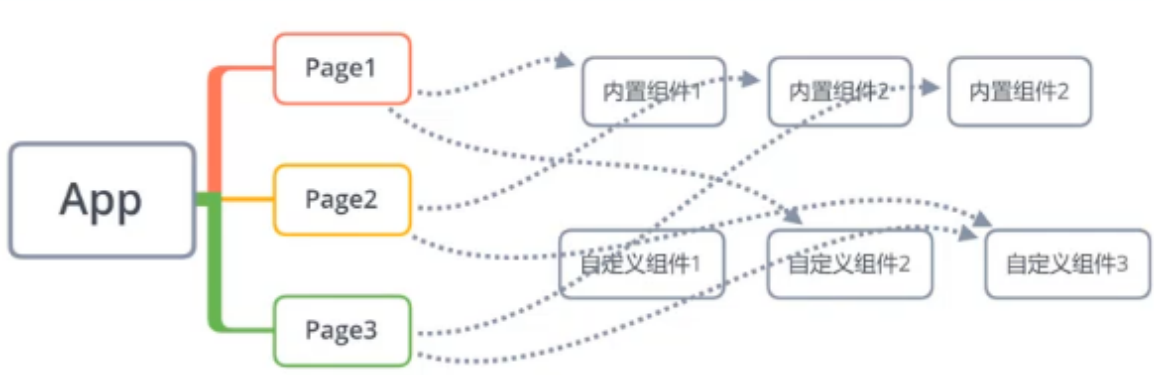
```
<!--pages/event/event.wxml-->
<!-- bind:会逐级触发 -->
<!-- catch:阻止下一级触发 -->
<view id="outer" bindtap="handleTap1"  capture-bind:tap='captureTap1'>
  outer view
  <view id="middle" bindtap="handleTap2"  capture-bind:tap='captureTap2'>
    middle view
    <view id="inner" bindtap="handleTap3"  capture-catch:tap='captureTap3'>
      inner view
    </view>
  </view>
</view>
```

修改上面代码，在`#inner`上将`capture-bind:tap`修改为`capture-catch:tap`，则会在第三个事件触发，拒绝后面的事件执行。

> outercaptureTap1
> middlecaptureTap2
> innercaptureTap3

组件化开发

- 人面对复杂问题的处理方式?
  - 任何认处理信息的逻辑能力都是有限的
  - 所以，当面对一个非常复杂的问题时，我们不太可能一次性搞定一大堆问题
  - 但是，我们有一种天生的能力，就是将问题进行拆解
  - 如果将一个复杂的问题，拆分成很多个可以处理的小问题，再将其放在整体当中，你会发现大的问题也会迎刃而解。
- 组件化也是类似的思想。
  - 如果我们将一个页面中所有的处理逻辑全部放在一起，处理起来就会变得非常复杂，而且不利于后续的管理以及扩展
  - 但是，我们将一个页面拆分成一个个小的功能模块，每个功能模块完成属于自己这部分独立的逻辑，那么之后整个页面的管理和维护就变得非常容易了。



- 组件化思想的应用
  - 有了组件化的思想，我们在之后的开发中就要充分的利用它
  - 尽可能的将页面拆分成一个个小的、可复用的组件
  - 这样让我们的代码更加方便组织和挂你，并且扩展性也更强。

自定义组件

文档地址：https://developers.weixin.qq.com/miniprogram/dev/framework/custom-component/

自定义组件类似于页面，由**json、wxml、wxss、js**四个文件组成。

- 创建一个自定义组件
    1. 在根目录下创建一个components目录用来存放公共组件
    2. 创建自定义组件包含四个文件，但同时建议将这四个文件放在与组件同名目录下。
    3. 编辑自定义组件
    4. 使用自定义组件

在**js**文件中封装两个变量

```
// components/my-cpn/my-cpn.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {

  },

  /**
   * 组件的初始数据
   */
  data: {
    title:'我是标题',
    content:'我是自定义组件的内容'
  },

  /**
   * 组件的方法列表
   */
  methods: {

  }
})
```

components/my-cpn/my-cpn.json文件中**"component": true**这个属性时自动生成的，因为有这个属性，所以它才是个组件，不能随便删除。

```
{
  "component": true,
  "usingComponents": {}
}
```

数据定义在**js**文件中

```
<!--components/my-cpn/my-cpn.wxml-->
<view class='title'>{{title}}</view>
<view class='content'>{{content}}</view>
```

设置简单的样式

```
/* components/my-cpn/my-cpn.wxss */
.title{
  font-size: 40rpx;
  font-weight: 700;
}
.content{
  font-size: 30rpx;
}
```

使用自定义组件有2个步骤，首先要在**json**文件中声明组件，以键值对的`key`表示组件被调用时**组件**标签的名称，`value`则是被调用组件的路径地址。**注意，这个地址是四个文件的名称地址，不是目录的地址。**

```
{
  "usingComponents": {
    "my-cpn":"/components/my-cpn/my-cpn"
  }
}
```

然后，可以在页面中以标签形式调用组件，可多次调用同一组件。

```
<!--pages/mycomponent/mycomponent.wxml-->
<text>在当前页面中使用自定义组件</text>

<!-- 1.使用自定义的组件 -->
<my-cpn></my-cpn>
<my-cpn/>
```

自定义组件注意事项

- 因为wxml节点标签名只能是**小写字母、中划线和下划线**的组合，所以自定义组件的标签名也只能包含这些字符。
- 自定义组件也可以引用其他自定义组件，引用方法类似于页面引用自定义组件的方式。
- 自定义组件和页面所在的项目根目录名**不能以“wx-”为前缀**，否则会报错。
- 如果在app.json的usingComponents声明某个组件，那么所有页面和组件都可以直接使用该组件，但不推荐这样使用。

组件的样式

- 组件内的样式对外部样式的影响包括：
  - 组件内的**class样式**，只对组件wxml内的节点生效，对于引用组件的Page页面不生效。
  - 组件内不能使用id选择器、属性选择器、标签选择器。

如果在组件**wxss**文件中使用了以上选择器，会提示以下错误。

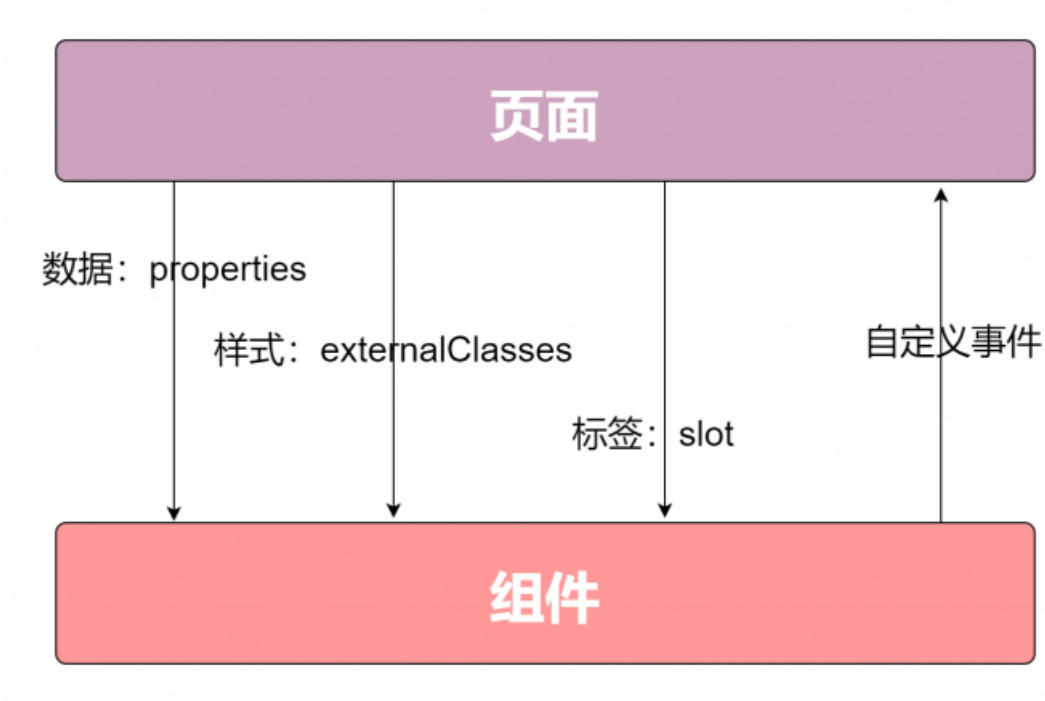
> Some selectors are not allowed in component wxss, including tag name selectors, ID selectors, and attribute selectors.(./components/my-style/my-style.wxss:9:1)

- 外部样式对组件内的影响
  - 外部使用了class的样式，只对外部wxml的class生效，对组件内是不生效的
  - 外部使用了id选择器、属性选择器不会对组件内产生影响
  - 外部使用了标签选择器，会对组件内产生影响
- 总结
  - 组件内的class样式和组件外的class样式，默认是有一个隔离效果的；

- 为了防止样式的错乱，官方不推荐使用id、属性、标签选择器；

组件之间的通信

很多情况下，组件内展示的内容（数据、样式、标签），并不是在组件内定义死的，而是由使用者来决定的。



- 向组件传递数据——properties

1. 在**components**目录下创建**my-prop**文件夹，添加名为**my-prop**的组件。
2. 打开**my-prop.js**文件，在**properties**中添加属性名称，属性名称有两种定义方法，推荐使用标准方法定义，其中observer可以用来监听新传入的值和原来的值。
3. 在页面的测试页面**mycomponent.json**文件中定义**my-prop**组件。
4. 最后在**mycomponent.wxml**文件中调用组件即可，直接在组件标签中传入希望的数据即可。

属性支持的类型包括：String、Number、Boolean、Object、Array、Null（不限制类型）

定义可传入数据的组件

```
// components/my-prop/my-prop.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {
    //简易属性定义
    title: String,
    //标准属性定义
    content: {
      type: String,
      value: '我是默认的内容',
      observer: function (newVal, oldVal) {
        // 监听新值和旧值
        console.log(newVal, oldVal);
      }
    }
  }
})
```

```
/* components/my-prop/my-prop.wxss */
.title{
  font-size: 50rpx;
  font-weight: 700;
}
.content{
  font-size: 40rpx;
}
```

```
<!--components/my-prop/my-prop.wxml-->
<view class='title'>{{title}}</view>
<view class='content'>{{content}}</view>
```

使用组件时，并传入数据；先定义需要使用的组件名称和路径。

```
{
  "usingComponents": {
    "my-prop":"/components/my-prop/my-prop"
  }
}
```

```
<!--pages/mycomponent/mycomponent.wxml-->
<text>在当前页面中使用自定义组件</text>

<!-- 3.组建通信 -->
<my-prop title='我是传递到组件的标题' content='我是传入的内容'/>
```

当属性没有默认值，且又没有传入值时，属性值为该属性类型的默认值。运行后，**observer**监视的数据如下：

> 我是传入的内容 我是默认的内容

- 向组件内传入样式——externalClasses
    1. 在**my-prop.js**文件中设置**externalClasses**的值，**externalClasses**是一个数组内心个，每个值都表示一个可以用于作为**class**属性**样式名称**。注意样式名称**不能大写**。
    2. 在**my-prop.wxml**的标签中的**class**属性上设置刚才设置的**样式名称**。
    3. 在页面调用时，把样式名称当做属性来传入新样式，注意，新样式需要在页面中存在。

设置组件中可传入的样式

```
// components/my-prop/my-prop.js
Component({
  /**
   * 组件的属性列表
   */
  properties: {
    //简易属性定义
    title: String,
    //标准属性定义
    content: {
      type: String,
      value: '我是默认的内容',
      observer: function (newVal, oldVal) {
        // 监听新值和旧值
```

```
        console.log(newVal, oldVal);
      }
    }
  },
  //组件样式列表，可以通过此样式属性将页面的样式传递到组件内
  externalClasses:["titleclass"]
})
```

```
<!--components/my-prop/my-prop.wxml-->
<view class='title titleclass'>{{title}}</view>
<view class='content'>{{content}}</view>
```

使用组件，并传入样式；先定义需要使用的组件名称和路径。

```
{
  "usingComponents": {
    "my-prop":"/components/my-prop/my-prop"
  }
}
```

```
<!--pages/mycomponent/mycomponent.wxml-->
<text>在当前页面中使用自定义组件</text>

<!-- 3.组建通信 -->
<my-prop title='我是传递到组件的标题' content='我是传入的内容' titleclass='mycss'/>
```

```
/* pages/mycomponent/mycomponent.wxss */
.mycss{
  color: darkgrey;
}
```

- 组件向外传递事件——自定义事件
    1. 创建自定义组件**my-event**，并在其中定义需触发的事件
    2. 在事件中通过**triggerEvent**函数将功能传递到外部，该函数两个参数：**1：事件名称；2：提供的参数。**
    3. 在页面调用组件时，在组件标签上绑定刚才设置的**事件名称。**
    4. 执行测试，注意在触发事件时，事件附带参数**event.detail**属性包含组件所提供的参数

定义组件代码如下：

```
// components/my-event/my-event.js
Component({
  /**
   * 组件的方法列表
   */
  methods: {
    handleIncrement(){
      //发送事件（事件名称，参数）
      this.triggerEvent("increment",{name:'test'})
    }
  }
})
```

```
<!--components/my-event/my-event.wxml-->
<!-- 触发事件，引发页面内容改变 -->
<button size='mini' bindtap="handleIncrement">+1</button>
```

页面调用组件代码如下：

```
{
  "usingComponents": {
    "my-event":"/components/my-event/my-event"
  }
}
```

```
<!--pages/mycomponent/mycomponent.wxml-->
<text>在当前页面中使用自定义组件</text>

<!-- 4.组件内部发送自定义事件 -->
<view>当前计数：{{counter}}</view>
<my-event bind:increment='handleIncrement'/>
```

```
// pages/mycomponent/mycomponent.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    counter: 0
  },
  //+1方法
  handleIncrement(event) {
    //获取组件传过来的参数
    console.log(event.detail);
    this.setData({
      counter: this.data.counter + 1
    })
  }
})
```

测试有个打印的结果是组件传递个页面的参数：

> {name: "test"}