

互联网应用开发技术

Web Application Development

第15课 微信小程序

Episode Fifteen
WeChat App

陈昊鹏
chen-hp@sjtu.edu.cn



什么是微信小程序

- 可以通过微信直接进入
- 轻量级



微信小程序的特点



REliable, INtelligent & Scalable Systems

- 使用方便
- 开发方便（开发者可以直接调用微信提供的API）
- 推广方便（可以借助微信的巨大流量）

- 登录微信公众平台 <https://mp.weixin.qq.com>
- 注册一个小程序账号，并获取开发者AppId

小程序注册



每个邮箱仅能申请一个小程序

邮箱 作为登录帐号。请填写未被微信公众平台注册，未被微信开放平台注册，未被个人微信号绑定的邮箱

密码 字母、数字或者英文字符，最短8位，区分大小写

确认密码 请再次输入密码

验证码  换一张

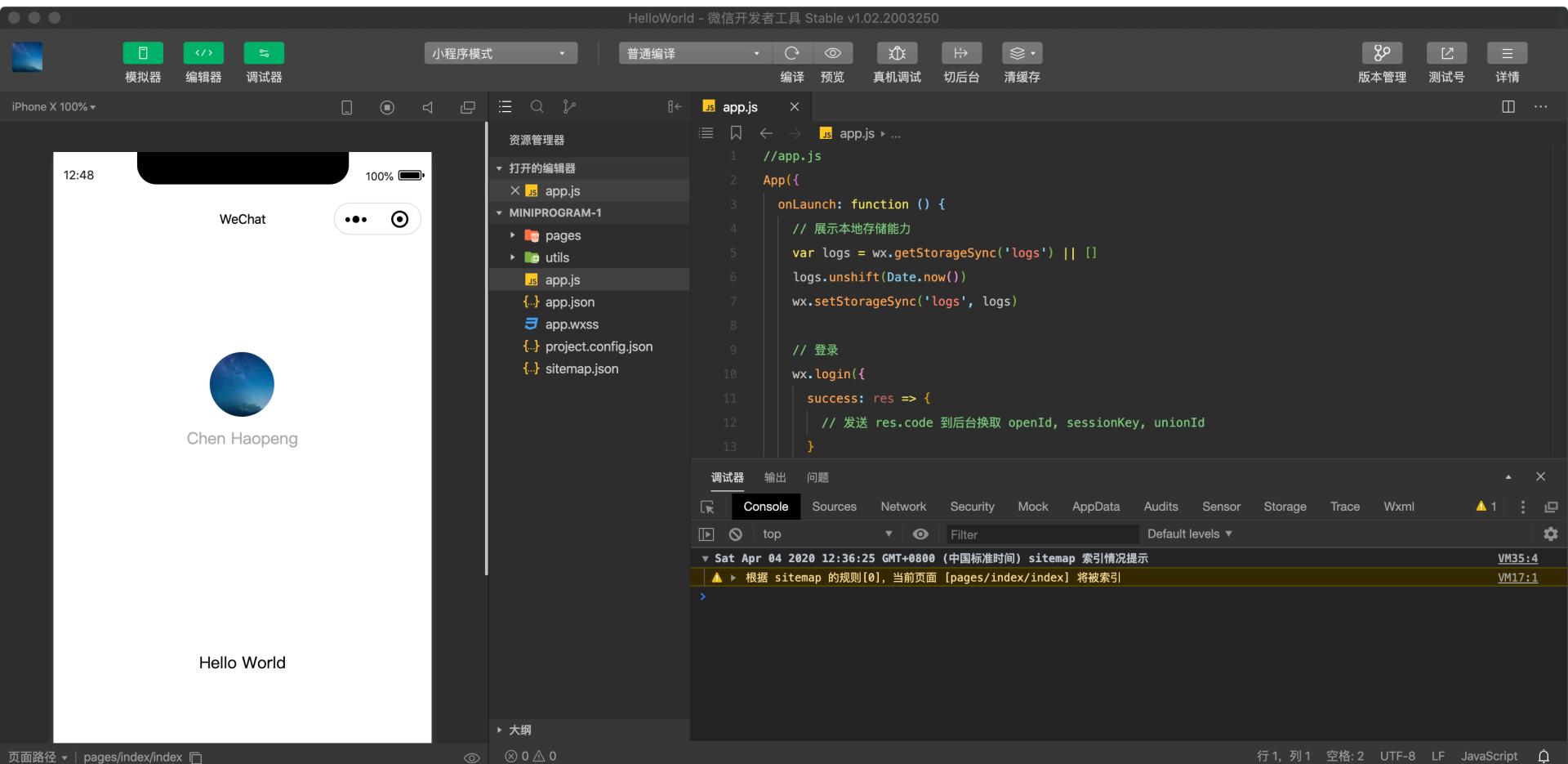
你已阅读并同意《微信公众平台服务协议》及《微信小程序平台服务条款》

帐号信息

AppID(小程序ID)  9abcd6c

前期准备

- 安装小程序开发专用IDE：微信开发者工具
- 左侧是预览界面，右侧是代码编辑界面

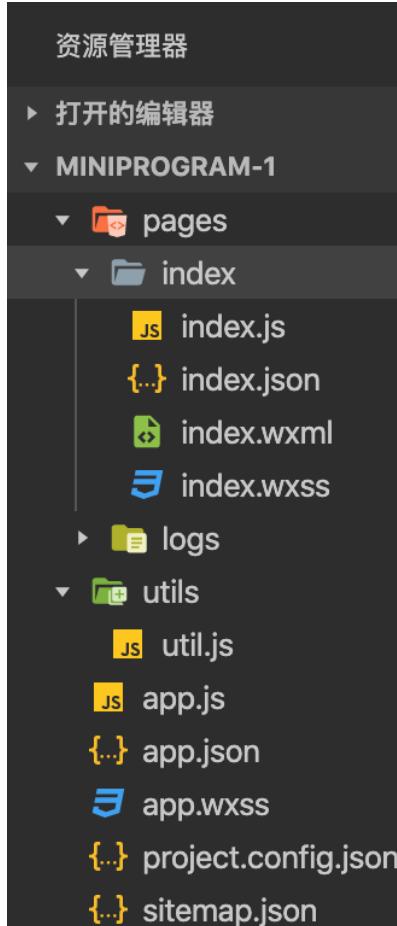


小程序代码构成



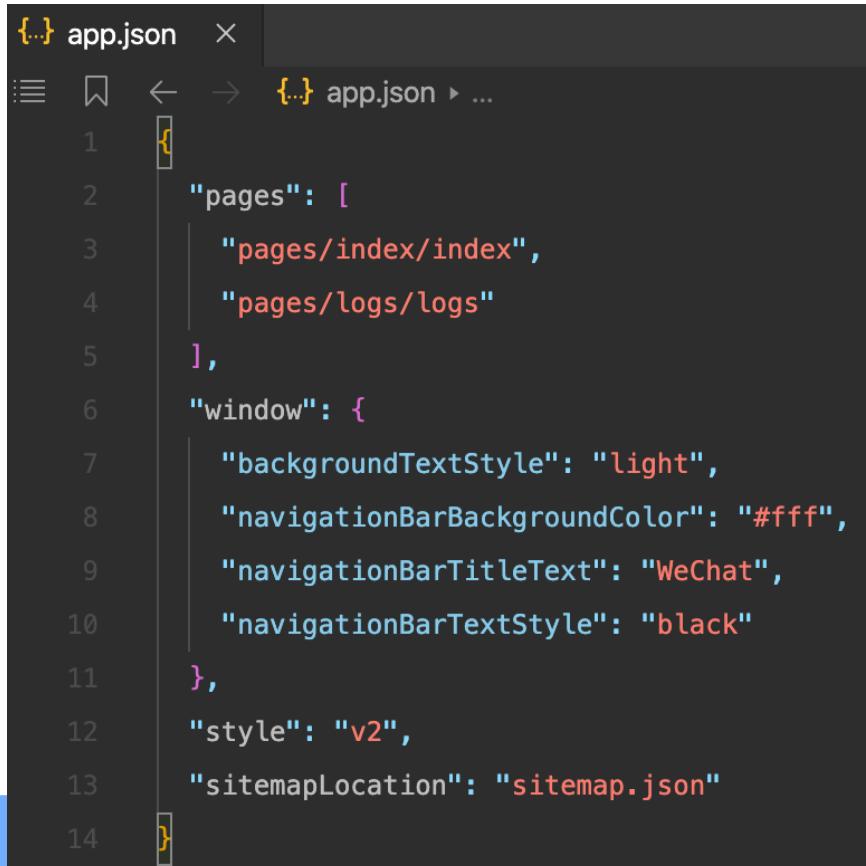
REliable, INtelligent & Scalable Systems

- 我们通过开发者工具快速创建了一个 QuickStart 项目
- 你可以留意到这个项目里边生成了不同类型的文件：
 - json 后缀的 JSON 配置文件
 - wxml 后缀的 WXML 模板文件
 - wxss 后缀的 WXSS 样式文件
 - js 后缀的 JS 脚本逻辑文件



小程序代码构成 – 小程序配置

- **app.json** – 当前小程序的全局配置
 - 配置所有页面路径
 - 配置界面表现、网络超时时间、底部tab等
 - 配置所用到的组件（也可以在页面单独的json文件中配置）



```
{...} app.json x
...
1 {
2   "pages": [
3     "pages/index/index",
4     "pages/logs/logs"
5   ],
6   "window": {
7     "backgroundTextStyle": "light",
8     "navigationBarBackgroundColor": "#fff",
9     "navigationBarTitleText": "WeChat",
10    "navigationBarTextStyle": "black"
11  },
12  "style": "v2",
13  "sitemapLocation": "sitemap.json"
14 }
```

小程序代码构成 – 小程序配置

- project.config.json – 工具配置

```
{  
    "description": "项目配置文件",  
    "packOptions": {  
        "ignore": []  
    },  
    "setting": {  
        "urlCheck": true,  
        "es6": true,  
        "postcss": true,  
        "preloadBackgroundData": false,  
        "minified": true,  
        "newFeature": true,  
        "autoAudits": false,  
        "coverView": true,  
        "showShadowRootInWxmlPanel": true,  
        "scopeDataCheck": false  
    },  
    "compileType": "miniprogram",  
    "libVersion": "2.10.4",  
    "appid": "wx0eff7e14a53f6909",  
    "projectname": "HelloWorld",  
    "debugOptions": {  
        "hidedInDevtools": []  
    },  
    "isGameTourist": false,  
    "simulatorType": "wechat",  
    "simulatorPluginLibVersion": {},  
    "condition": {  
        "search": {  
            "current": -1,  
            "list": []  
        },  
        "conversation": {  
            "current": -1,  
            "list": []  
        },  
        "game": {  
            "currentL": -1,  
            "list": []  
        },  
        "miniprogram": {  
            "current": -1,  
            "list": []  
        }  
    }  
}
```

小程序代码构成 – 小程序配置

- `page.json` – 页面配置
 - `pages/logs` 目录下的 `logs.json` 这类和小程序页面相关的配置



The screenshot shows a code editor window with a dark theme. The file is named `logs.json`. The code content is as follows:

```
{...} logs.json X
  pages > logs > {...} logs.json > ...
1  {
2      "navigationBarTitleText": "查看启动日志",
3      "usingComponents": {}
4 }
```

小程序代码构成 – WXML模板



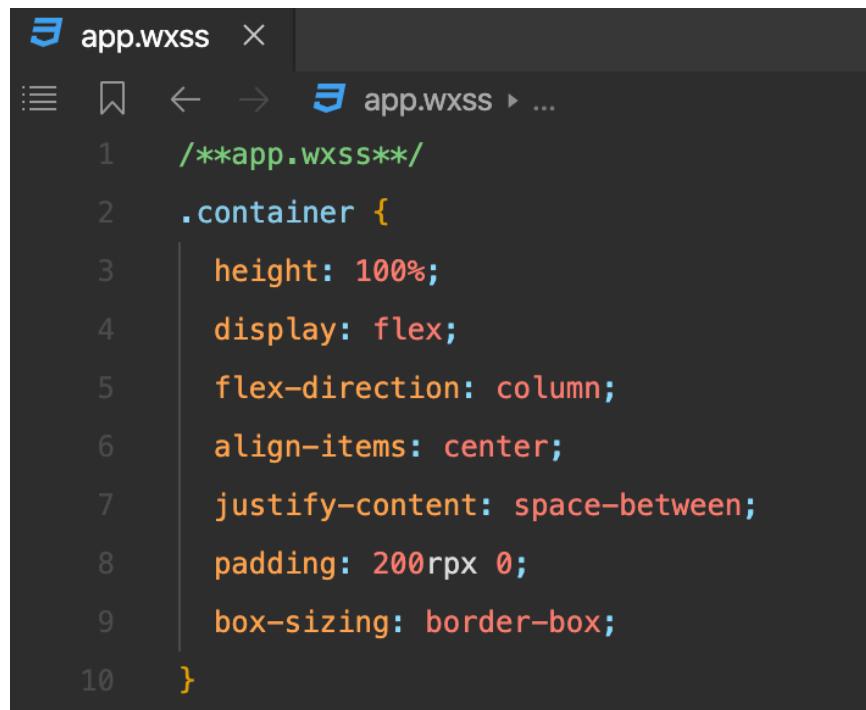
REliable, INtelligent & Scalable Systems

- 类似于HTML
 - 标签名字有点不一样
 - 多了一些 wx:if 这样的属性以及 {{ }} 这样的表达式
 - <https://developers.weixin.qq.com/miniprogram/dev/reference/wxml/list.html>

```
index.wxml ×
pages › index › index.wxml
1  <!--index.wxml-->
2  <view class="container">
3    <view class="userinfo">
4      <button wx:if="{{!hasUserInfo && canIUse}}" open-type="getUserInfo" bindgetuserinfo="getUserInfo"> 获取头像昵称 </button>
5      <block wx:else>
6        <image bindtap="bindViewTap" class="userinfo-avatar" src="{{userInfo.avatarUrl}}" mode="cover"></image>
7        <text class="userinfo-nickname">{{userInfo.nickName}}</text>
8      </block>
9    </view>
10   <view class="usermotto">
11     <text class="user-motto">{{motto}}</text>
12   </view>
13 </view>
```

小程序代码构成 – WXSS样式

- WXSS 具有 CSS 大部分的特性，另外做了一些扩充和修改



```
app.wxss
1  /**app.wxss*/
2  .container {
3    height: 100%;
4    display: flex;
5    flex-direction: column;
6    align-items: center;
7    justify-content: space-between;
8    padding: 200rpx 0;
9    box-sizing: border-box;
10 }
```



```
index.wxss
1  /**index.wxss*/
2  .userinfo {
3    display: flex;
4    flex-direction: column;
5    align-items: center;
6  }
7
8  .userinfo-avatar {
9    width: 128rpx;
10   height: 128rpx;
11   margin: 20rpx;
12   border-radius: 50%;
13 }
14
15 .userinfo-nickname {
16   color: #aaa;
17 }
18
19 .usermotto {
20   margin-top: 200px;
21 }
```

小程序代码构成 – JS逻辑交互



REliable, INtelligent & Scalable Systems

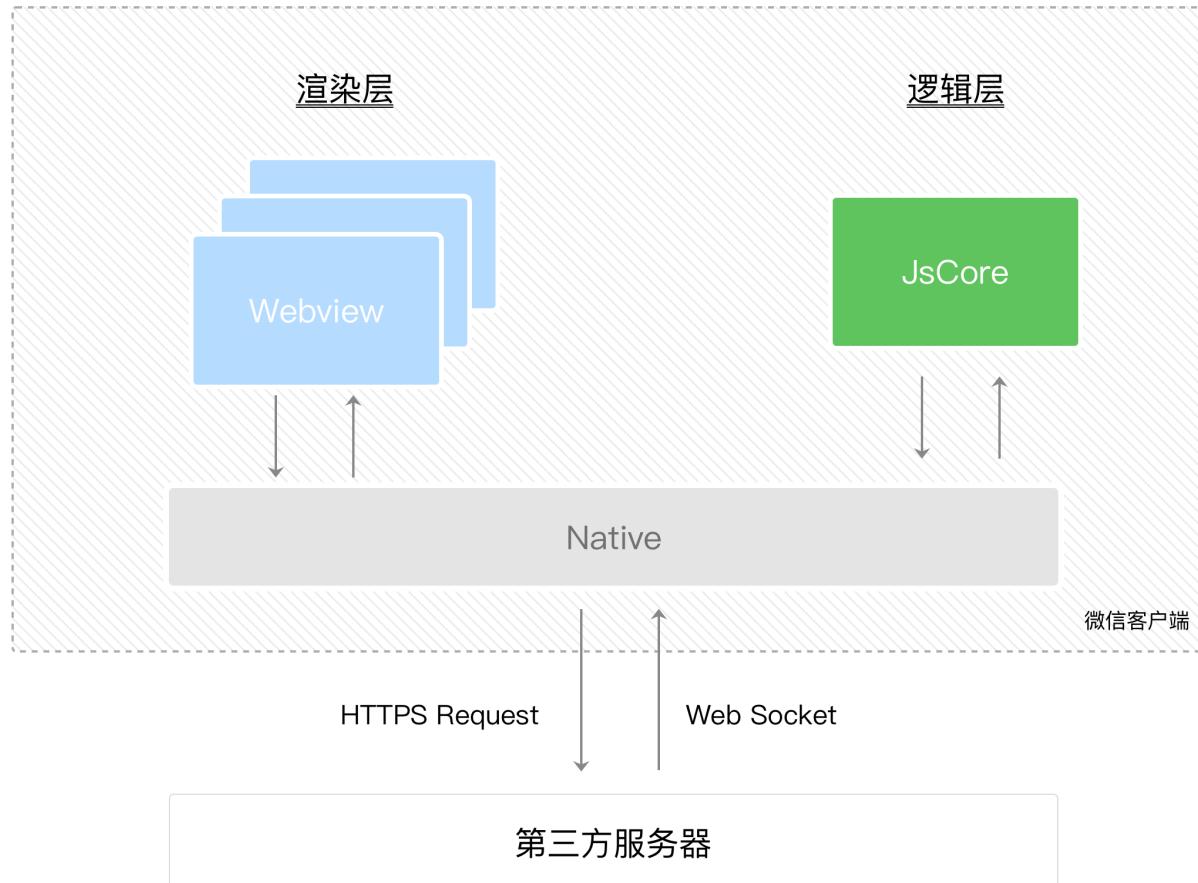
- 在小程序里边，通过编写 JS 脚本文件来处理用户的操作

```
<view>{{ msg }}</view>
<button bindtap="clickMe">点击我</button>
```

```
Page({
  clickMe: function() {
    this.setData({ msg: "Hello World" })
  }
})
```

小程序宿主环境 – 渲染层与逻辑层

- 微信客户端给小程序所提供的环境为宿主环境



小程序宿主环境 – 程序与页面

- 微信客户端在打开小程序之前，会把整个小程序的代码包下载到本地
 - 紧接着通过 app.json 的 pages 字段就可以知道当前小程序的所有页面路径
 - 整个小程序只有一个 App 实例，是全部页面共享的

```
{  
  "pages": [  
    "pages/index/index",  
    "pages/logs/logs"  
  ]  
}  
  
App({  
  onLaunch: function () {  
    // 小程序启动之后 触发  
  }  
})
```

- pages/logs/logs 下包括4种文件
 - 微信客户端会先根据 logs.json 配置生成一个界面
 - 顶部的颜色和文字你都可以在这个 json 文件里边定义好
 - 紧接着客户端就会装载这个页面的 WXML 结构和 WXSS 样式
 - 最后客户端会装载 logs.js，可以看到 logs.js 的大体内容就是：

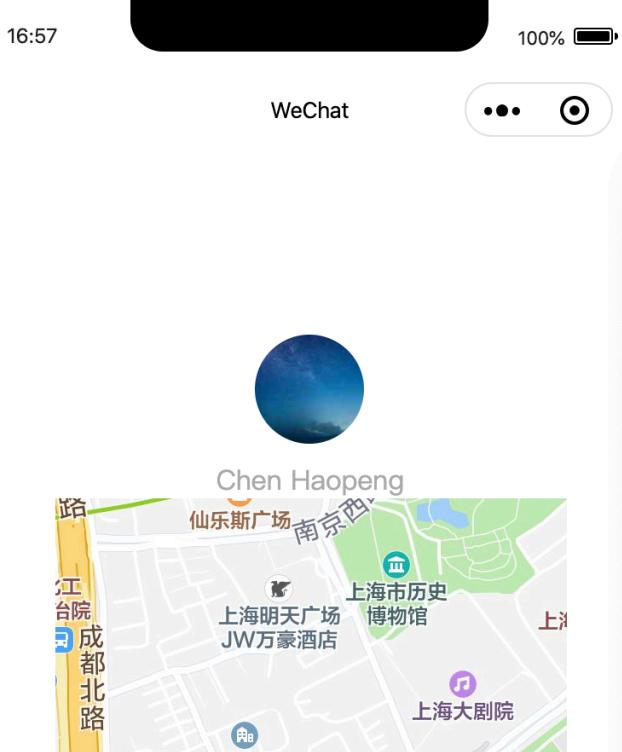
```
Page({  
  data: { // 参与页面渲染的数据  
    logs: []  
  },  
  onLoad: function () {  
    // 页面渲染后 执行  
  }  
})
```

- Page 是一个页面构造器，这个构造器就生成了一个页面

小程序宿主环境 – 组件

- 在小程序中，只需要在 WXML 写上对应的组件标签名字就可以把该组件显示在界面上

```
<map bindmarkertap="markertap" longitude="121.47" latitude="31.23"></map>
```



- 多数 API 的回调都是异步，需要处理好代码逻辑的异步问题
 - 获取用户的地理位置

```
wx.getLocation({  
  type: 'wgs84',  
  success: (res) => {  
    var latitude = res.latitude // 纬度  
    var longitude = res.longitude // 经度  
  }  
})
```

- 调用扫一扫

```
wx.scanCode({  
  success: (res) => {  
    console.log(res)  
  }  
})
```

- 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，如下：

文件	必需	作用
app.js	是	小程序逻辑
app.json	是	小程序公共配置
app.wxss	否	小程序公共样式表

小程序目录结构



REliable, INtelligent & Scalable Systems

- 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
- 一个小程序页面由四个文件组成，分别是：

文件类型	必需	作用
js	是	页面逻辑
wxml	是	页面结构
json	否	页面配置
wxss	否	页面样式表

- WXML 全称是 WeiXin Markup Language

<!-- 在此处写注释 -->

<标签名 属性名1="属性值1" 属性名2="属性值2" ...> ...</标签名>

- WXML 要求标签必须是严格闭合的，没有闭合将会导致编译错误

<!--一个简单的文本标签 -->

```
<text>hello world</text>
```

<!-- view 中包含了 text 标签 -->

```
<view>
```

```
  <text>hello world</text>
```

```
</view>
```

```
<image class="userinfo-avatar" src=".//image/a.png" ></image>
```

- 使用 WXML 语言所提供的数据绑定功能，实现动态改变渲染界面

```
<!--pages/wxml/index.wxml-->
<text>当前时间: {{time}}</text>
```

```
// pages/wxml/index.js
Page({
  /**
   * 页面的初始数据
   */
  data: {
    time: (new Date()).toString()
  }
})
```

- 属性值也可以动态的去改变，有所不同的是，属性值必须被包裹在双引号中

<!-- 正确的写法 -->

```
<text data-test="{{test}}> hello world</text>
```

<!-- 错误的写法 -->

```
<text data-test={{test}}> hello world </text >
```

- WXML 中，使用 `wx:if="{{condition}}"` 来判断是否需要渲染该代码块：

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

- 使用 `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>
```

- 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>
```

- 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：

```
<!-- array 是一个数组 -->
<view wx:for="{{array}}>
    {{index}}: {{item.message}}
</view>
```

```
<!-- 对应的脚本文件
Page({
  data: {
    array: [
      {
        message: 'foo',
      },
      {
        message: 'bar'
      }
    ]
  }
})
-->
```

- 使用 `wx:for-item` 指定数组当前元素的变量名，使用 `wx:for-index` 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>
```

- 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

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

- 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：

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

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

WXML模板 – 列表渲染



REliable, INtelligent & Scalable Systems

```
Page({
    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
        })
    }
},
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
    })
},
```

- 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

```
<!--  
item: {  
    index: 0,  
    msg: 'this is a template',  
    time: '2016-06-18'  
}  
-->  
  
<template name="msgItem">  
    <view>  
        <text> {{index}}: {{msg}} </text>  
        <text> Time: {{time}} </text>  
    </view>  
</template>  
  
<template is="msgItem" data="{{...item}}"/>  
  
<!-- 输出  
0: this is a template Time: 2016-06-18  
-->
```

- `is`可以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>

<!-- 输出
odd
even
odd
even
odd
-->
```

- WXML 提供两种文件引用方式import和include

- import 可以在该文件中使用目标文件定义的 template，如：
 - 在 item.wxml 中定义了一个叫 item 的 template :

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

- 在 index.wxml 中引用了 item.wxml，就可以使用 item 模板：
- ```
<import src="item.wxml"/>
<template is="item" data="{{text: 'forbar'}}"/>
```
- 需要注意的是 import 有作用域的概念，即只会 import 目标文件中定义的 template，而不会 import 目标文件中 import 的 template，简言之就是 import 不具有递归的特性

- WXML 提供两种文件引用方式import和include
  - include 可以将目标文件中除了 <template/> <wxs/> 外的整个代码引入，相当于拷贝到 include 位置，如代码2-22、代码2-23、代码2-24所示
  - 代码清单2-22 index.wxml

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

- 代码清单2-23 header.wxml

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

- 代码清单2-24 footer.wxml

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

- WXML 提供两种文件引用方式import和include

- 代码清单2-19 模板 A

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

- 代码清单2-20 模板 B

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

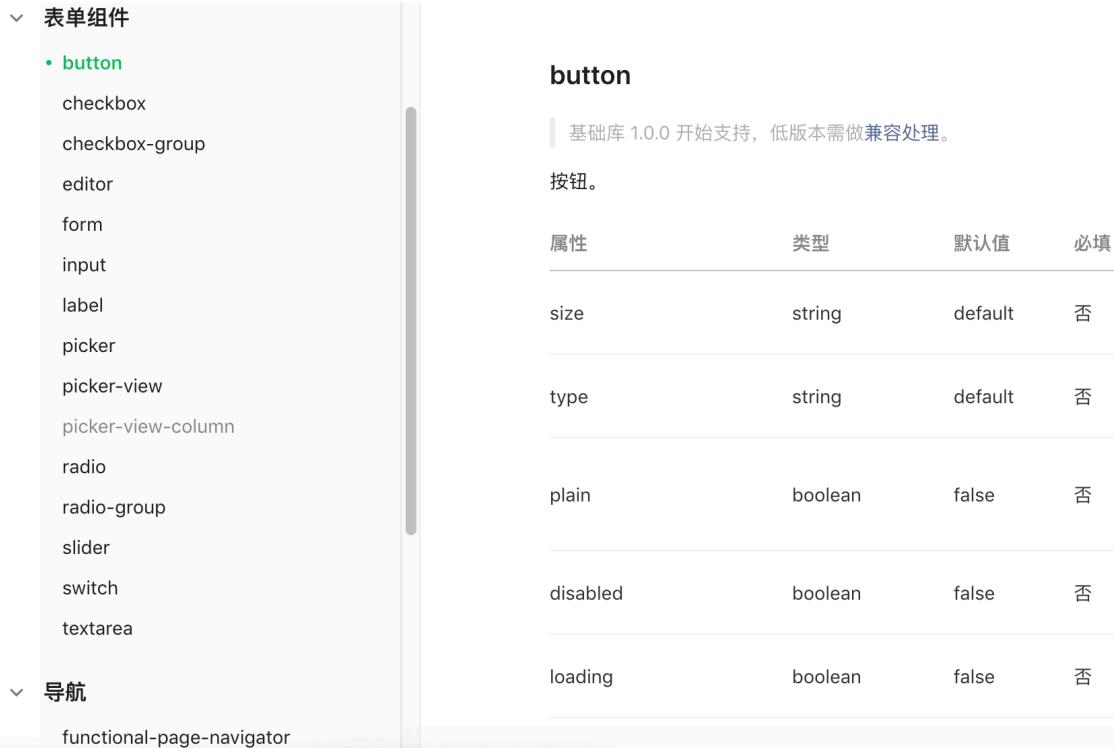
- 代码清单2-21 模板 C

```
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/> <!-- 这里将会触发一个警告，因为 b 中并没有定义模板 A -->
<template is="B"/>
```

- 所有wxml 标签都支持的属性称之为共同属性

属性名	类型	描述	注解
id	String	组件的唯一标识	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	

- 微信提供了非常多的页面组件供我们使用
- 参考文档：  
<https://developers.weixin.qq.com/miniprogram/dev/component/button.html>
- 也可以按照个人喜好选择支持小程序的第三方组件库



The screenshot shows the WeChat Developers Documentation website. On the left, there is a sidebar with a tree view of components under '表单组件' (Form Components), including 'button' (highlighted in green), 'checkbox', 'checkbox-group', 'editor', 'form', 'input', 'label', 'picker', 'picker-view', 'picker-view-column', 'radio', 'radio-group', 'slider', 'switch', and 'textarea'. Below this is another section titled '导航' (Navigation) with 'functional-page-navigator'. On the right, the main content area is titled 'button'. It contains a brief description: '基础库 1.0.0 开始支持，低版本需做兼容处理。' (Supported from version 1.0.0, compatibility handling is required for lower versions.) followed by the text '按钮。'. Below this is a table showing properties for the 'button' component:

属性	类型	默认值	必填
size	string	default	否
type	string	default	否
plain	boolean	false	否
disabled	boolean	false	否
loading	boolean	false	否

# WXSS样式 – 文件组成



REliable, INtelligent & Scalable Systems

The screenshot shows a file explorer interface with the following structure:

- + 普通编译
- ... ⌂
- 编辑 预览 切后台 清缓存
- + 搜索
- 其他
- 项目公共样式
- 页面样式
- demo
- images
- mod
  - common.wxss
  - tabBar.wxss
- pages
  - current
  - history
  - index
    - index.js
    - index.wxml
    - index.wxss
- utils
- widgets
- app.js
- app.json
- app.wxss
- gh\_050420932e31\_344.jpg

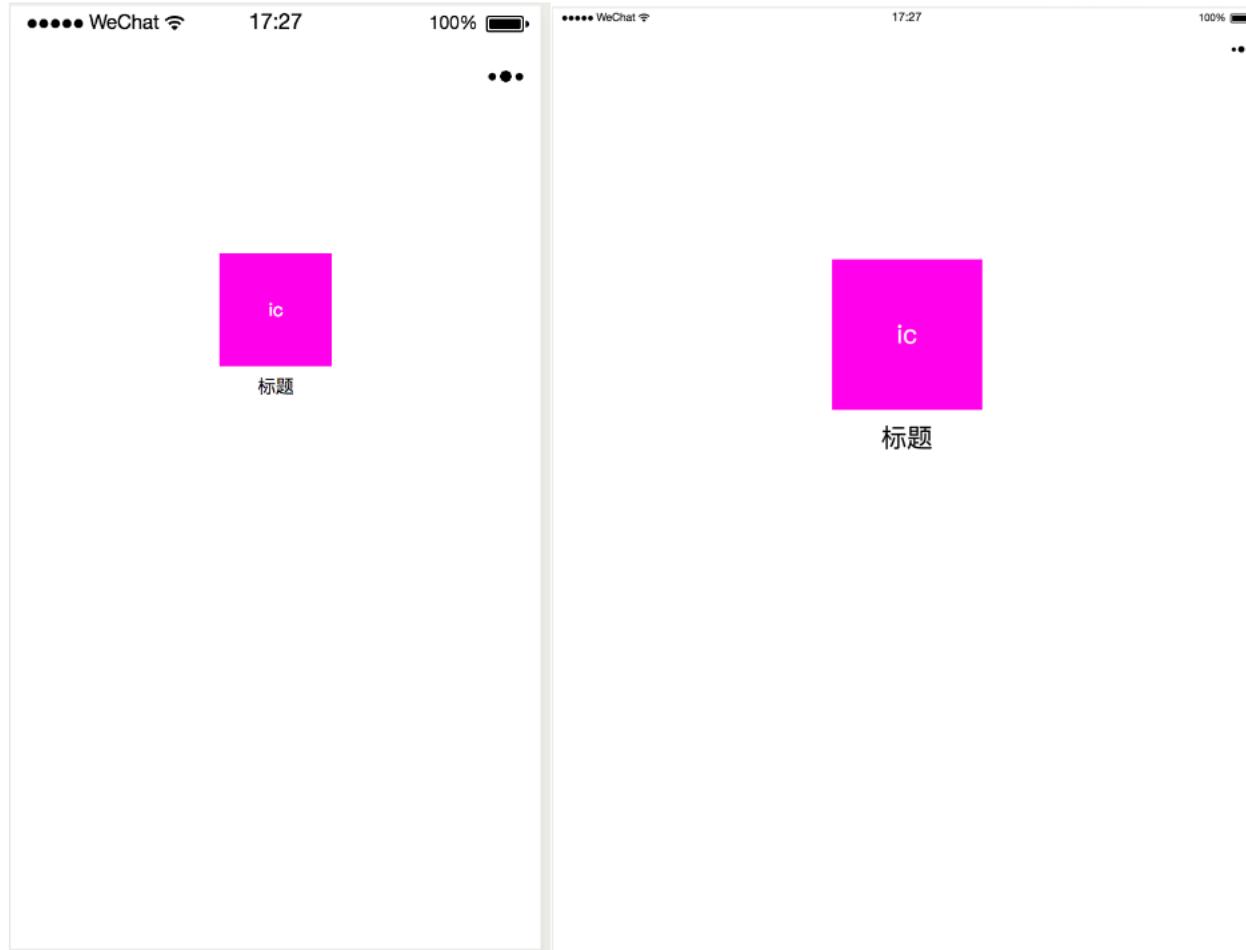
The files under 'common.wxss', 'index.wxss', and 'app.wxss' are highlighted in green, indicating they are WXSS files.

On the right, the content of the app.json file is displayed:

```
1 {
2 "pages": [
3 "pages/rpx/index",
4 "pages/box/index",
5 "pages/inlinestyle/index",
6 "pages/flex/index"
7],
8 "window": {
9 "navigationBarTextStyle": "light",
10 "navigationBarBackgroundColor": "#fff",
11 "navigationBarTitleText": " ",
12 "navigationBarTextStyle": "black"
13 }
14 }
15 }
```

# WXSS样式 – 尺寸单位

- 在WXSS中，引入了rpx (responsive pixel) 尺寸单位
  - 引用新尺寸单位的目的是，适配不同宽度的屏幕，开发起来更简单



- 在CSS中，开发者可以这样引用另一个样式文件：  
`@import url('./test_0.css')`
- 在小程序中，依然可以实现样式的引用，样式引用是这样写：  
`@import './test_0.wxss'`
- 由于WXSS最终会被编译打包到目标文件中
  - 用户只需要下载一次，在使用过程中不会因为样式的引用而产生多余的文件请求

- WXSS内联样式与Web开发一致:

```
<!--index.wxml-->
<!--内联样式-->
<view style="color: red; font-size: 48rpx"></view>
```

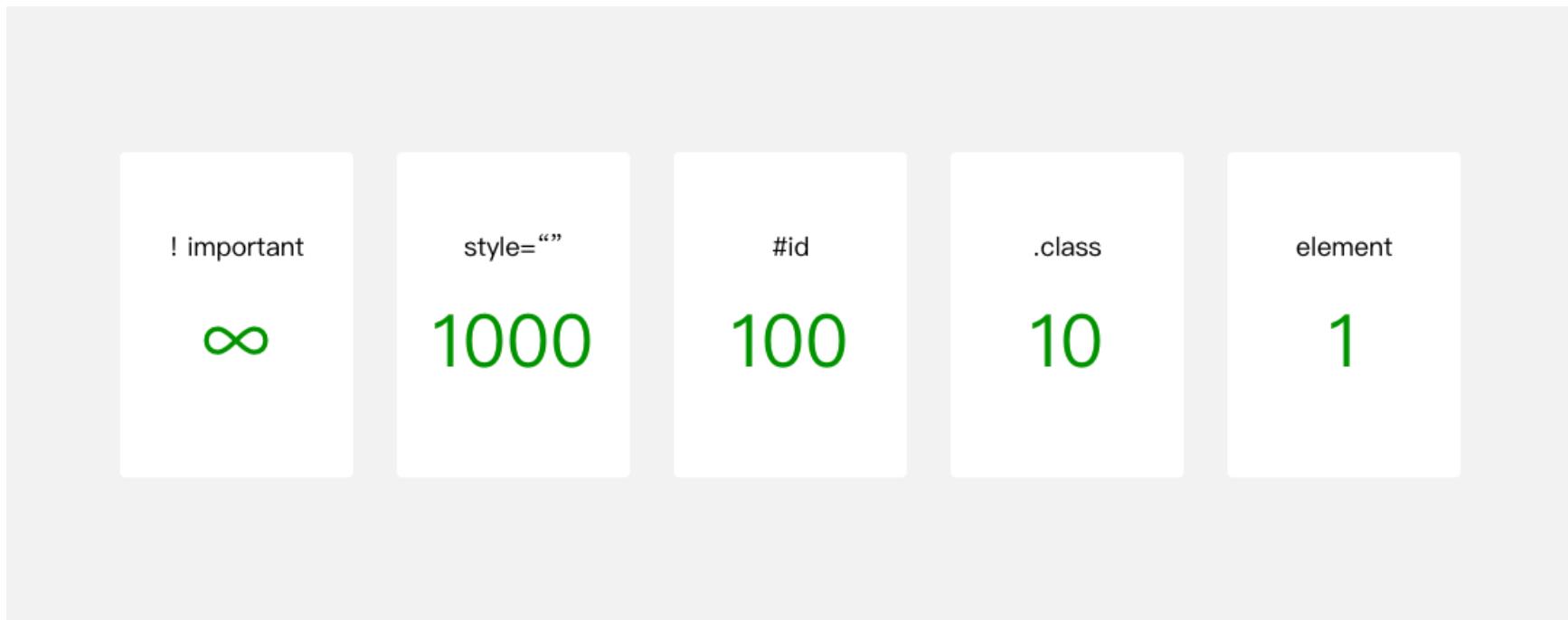
- 小程序支持动态更新内联样式:

```
<!--index.wxml-->
<!--可动态变化的内联样式-->
<!--
{
 eleColor: 'red',
 eleFontSize: '48rpx'
}
-->
<view style="color: {{eleColor}}; font-size: {{eleFontSize}}"></view>
```

- 目前支持的选择器

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

- WXSS优先级与CSS类似
  - 权重越高越优先
  - 在优先级相同的情况下，后设置的样式优先级高于先设置的样式



- WXSS 选择器优先级权重

```
view{ // 权重为 1
 color: blue
}

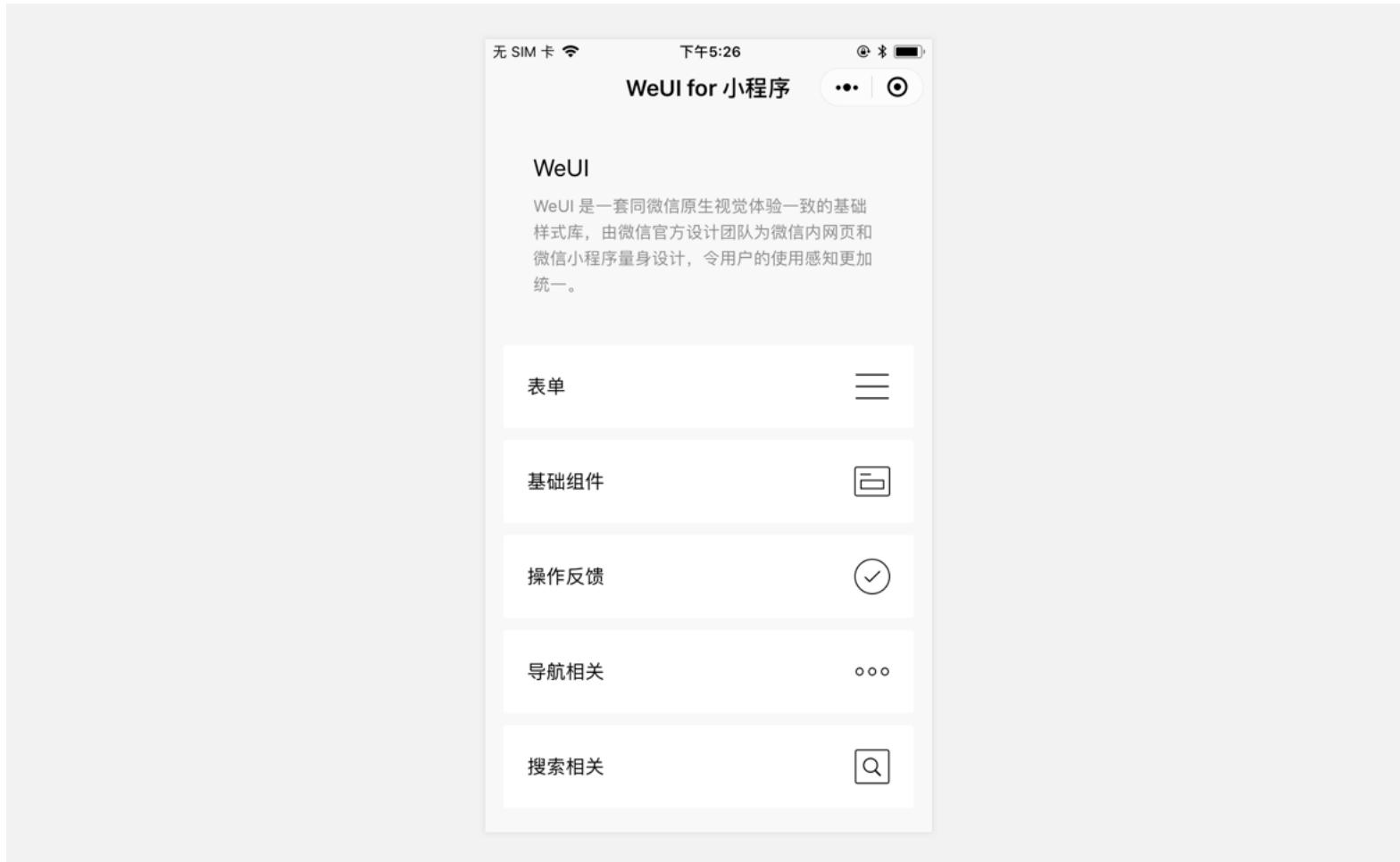
.ele{ // 权重为 10
 color: red
}

#ele{ // 权重为 100
 color: pink
}

view#ele{ // 权重为 1 + 100 = 101, 优先级最高, 元素颜色为orange
 color: orange
}

view.ele{ // 权重为 1 + 10 = 11
 color: green
}
```

- 微信提供了WeUI.wxss基础样式库



- 渲染层与逻辑层
- 小程序的运行环境分成渲染层和逻辑层
  - WXML 模板和 WXSS 样式工作在渲染层
  - JS 脚本工作在逻辑层
- 小程序的渲染层和逻辑层分离是经过很多考虑得出来的模型

# 渲染层和逻辑层 – 渲染“Hello World”

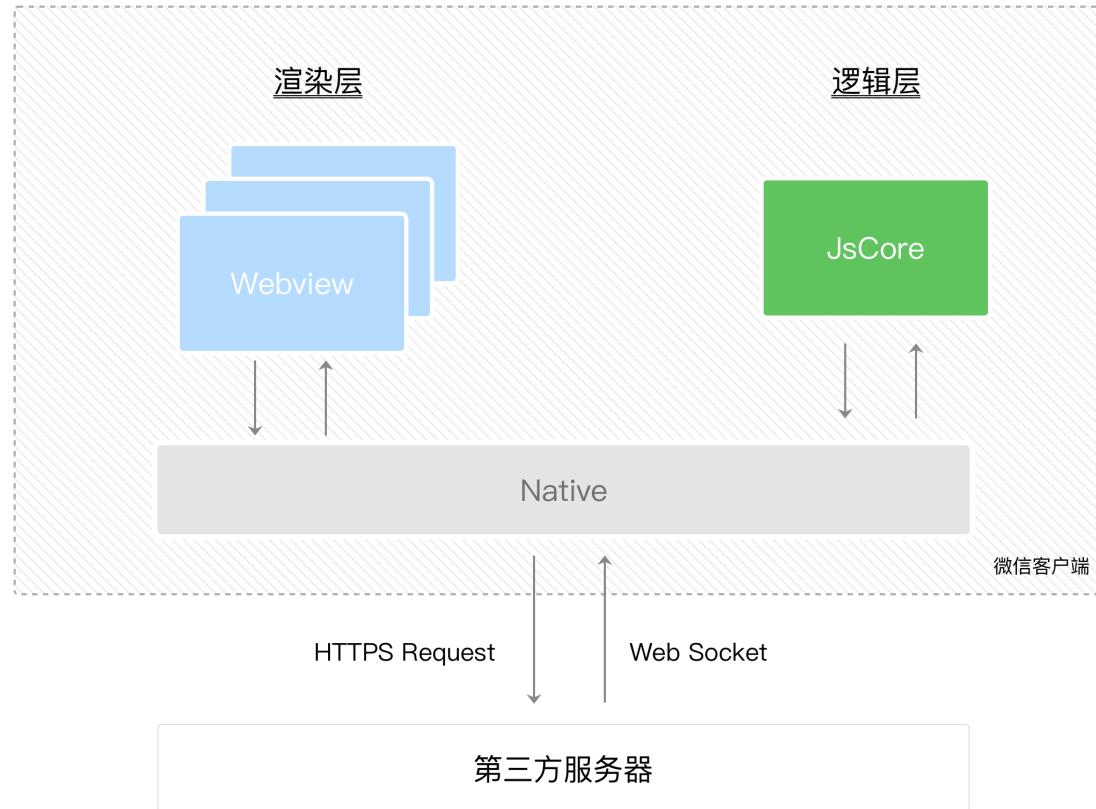
- WXML模板使用 view 标签，其子节点用 {{ }} 的语法绑定一个 msg 的变量
  - 渲染“Hello World”WXML代码

```
<view>{{ msg }}</view>
```
- 在 JS 脚本使用 this.setData 方法把 msg 字段设置成“Hello World”
  - 渲染“Hello World”JS脚本

```
Page({
 onLoad: function () {
 this.setData({ msg: 'Hello World' })
 }
})
```
- 从这个例子我们可以看到3个点：
  1. 渲染层和数据相关
  2. 逻辑层负责产生、处理数据
  3. 逻辑层通过 Page 实例的 setData 方法传递数据到渲染层

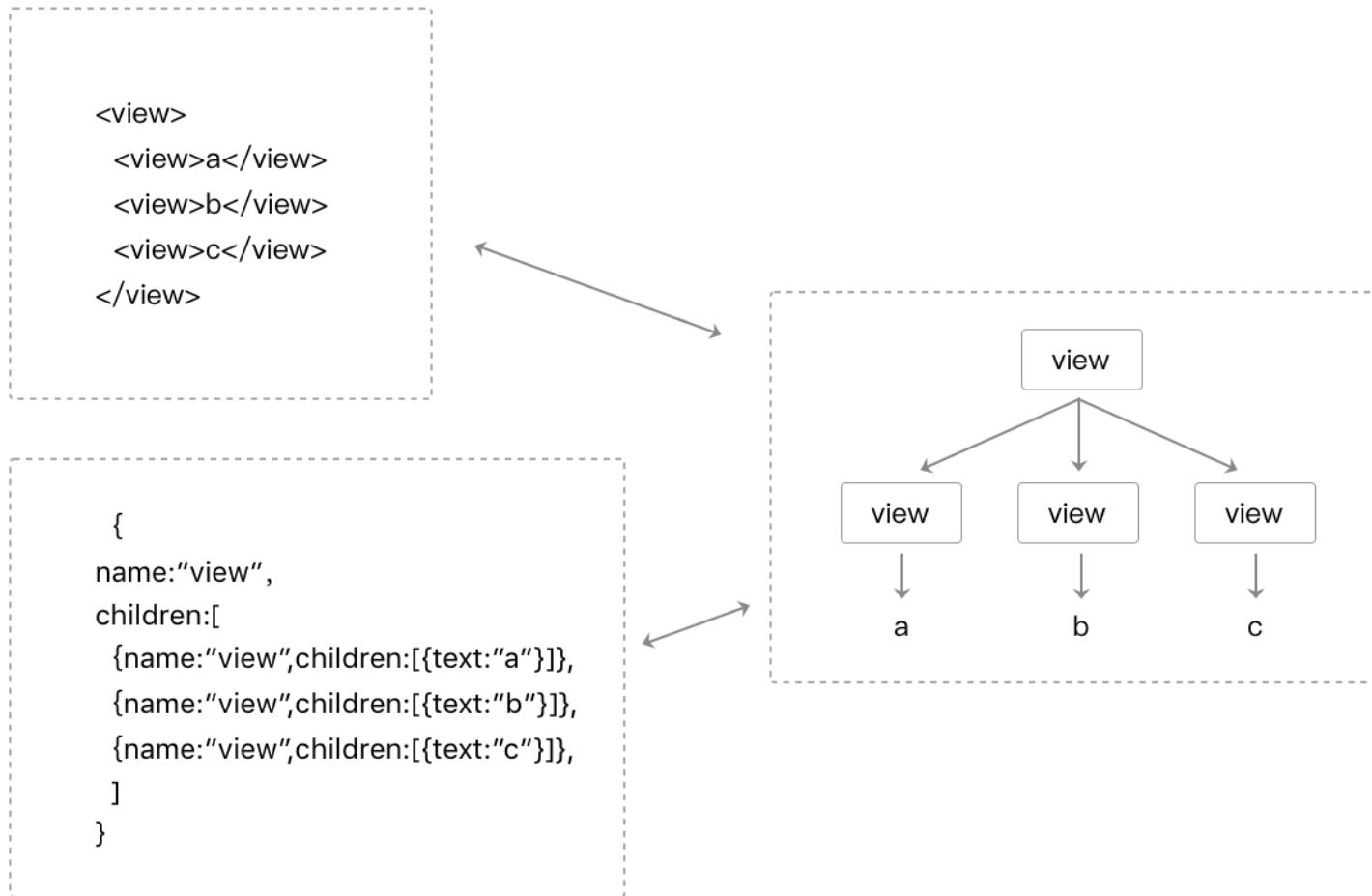
# 渲染层和逻辑层 – 通信模型

- 小程序的渲染层和逻辑层分别由2个线程管理：
  - 渲染层的界面使用了WebView 进行渲染
  - 逻辑层采用JsCore线程运行JS脚本



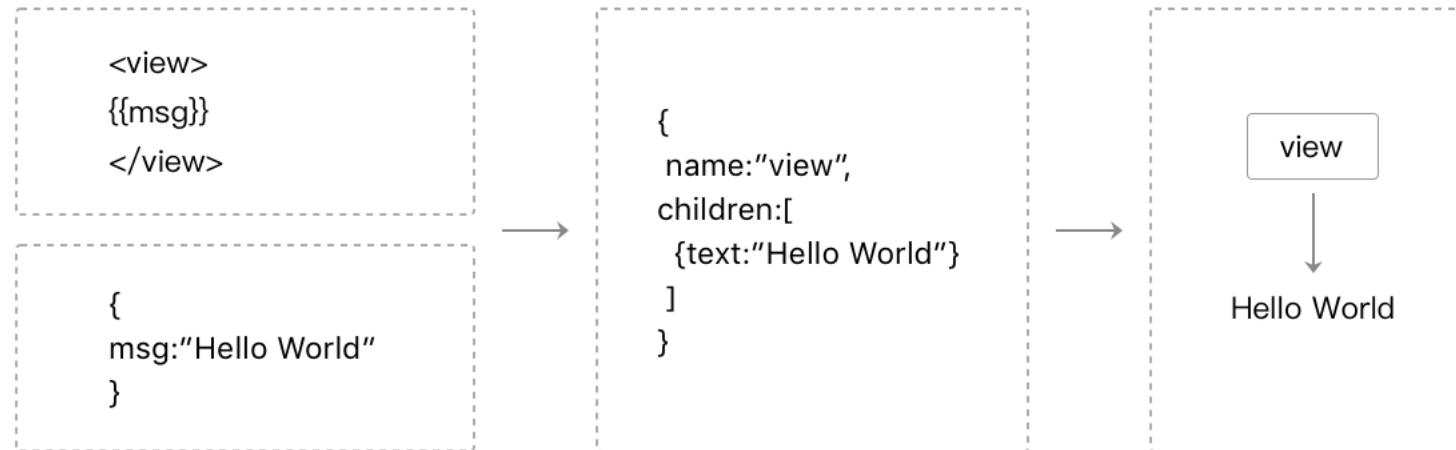
# 渲染层和逻辑层 – 数据驱动

- 让状态和视图绑定在一起（状态变更时，视图也能自动变更）



# 渲染层和逻辑层 – 数据驱动

- 让状态和视图绑定在一起（状态变更时，视图也能自动变更）

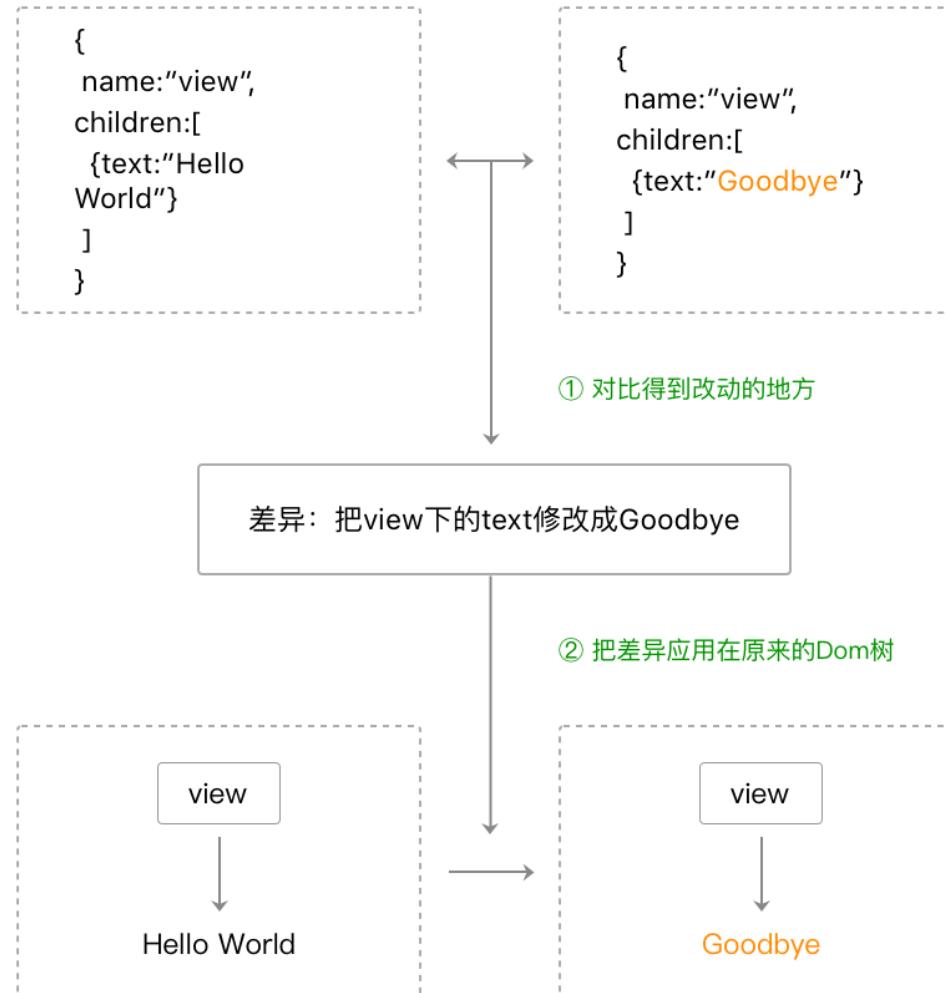


# 渲染层和逻辑层 – 数据驱动



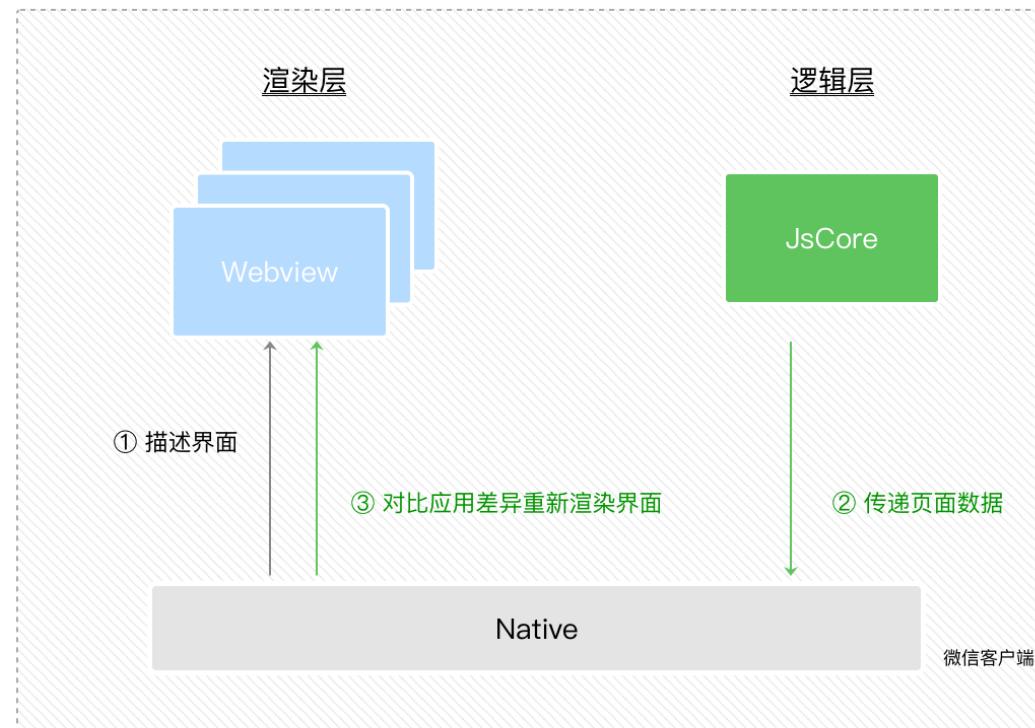
REliable, INtelligent & Scalable Systems

- 通过setData把msg数据从“Hello World”变成“Goodbye”



# 渲染层和逻辑层 - 双线程下的界面渲染

- 小程序的逻辑层和渲染层是分开的两个线程
  - 在渲染层，宿主环境会把WXML转化成对应的JS对象
  - 在逻辑层发生数据变更的时候，我们需要通过宿主环境提供的setData方法把数据从逻辑层传递到渲染层，再经过对比前后差异，把差异应用在原来的Dom树上，渲染出正确的UI界面



- 从逻辑组成来说，一个小程序是由多个“页面”组成的“程序”
  - “小程序”指的是产品层面的程序
  - 而“程序”指的是代码层面的程序实例

- 宿主环境提供了 App() 构造器用来注册一个程序App
- 需要留意的是App() 构造器必须写在项目根目录的app.js里
- App实例是单例对象，在其他JS脚本中可以使用宿主环境提供的 getApp() 来获取程序实例

```
// other.js
var appInstance = getApp()
```

```
App({
 onLaunch: function(options) {},
 onShow: function(options) {},
 onHide: function() {},
 onError: function(msg) {},
 globalData: 'I am global data'
})
```

- 1. 程序构造器App()

表3-1 App构造器的参数

参数属性	类型	描述
onLaunch	Function	当小程序初始化完成时，会触发 onLaunch（全局只触发一次）
onShow	Function	当小程序启动，或从后台进入前台显示，会触发 onShow
onHide	Function	当小程序从前台进入后台，会触发 onHide
onError	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
其他字段	任意	可以添加任意的函数或数据到 Object 参数中，在App实例回调用 this 可以访问

- 一个小程序可以有很多页面，每个页面承载不同的功能，页面之间可以互相跳转
  - 一个页面是分三部分组成：界面、配置和逻辑
  - 界面由WXML文件和WXSS文件来负责描述
  - 配置由JSON文件进行描述
  - 页面逻辑则是由JS脚本文件负责
  - 一个页面的文件需要放置在同一个目录下
    - 其中WXML文件和JS文件是必须存在的，JSON和WXSS文件是可选的
  - app.json声明页面路径

```
{
 "pages": [
 "pages/index/page", // 第一项默认为首页
 "pages/other/other"
]
}
```

- Page构造器的参数

参数属性	类型	描述
data	Object	页面的初始数据
onLoad	Function	生命周期函数--监听页面加载，触发时机早于onShow和onReady
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，触发事件早于onReady
onHide	Function	生命周期函数--监听页面隐藏
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	用户点击右上角转发
onPageScroll	Function	页面滚动触发事件的处理函数
其他	Any	可以添加任意的函数或数据，在Page实例的其他函数中用 this 可以访问

- 页面的数据

```
// page.js
Page({
 data: {
 a: 1, b: 2, c: 3, d: [1, {text: 'Hello'}, 3, 4]
 }
 onLoad: function(){
 // a需要变化时，只需要setData设置a字段即可
 this.setData({a: 2})
 }
})
```

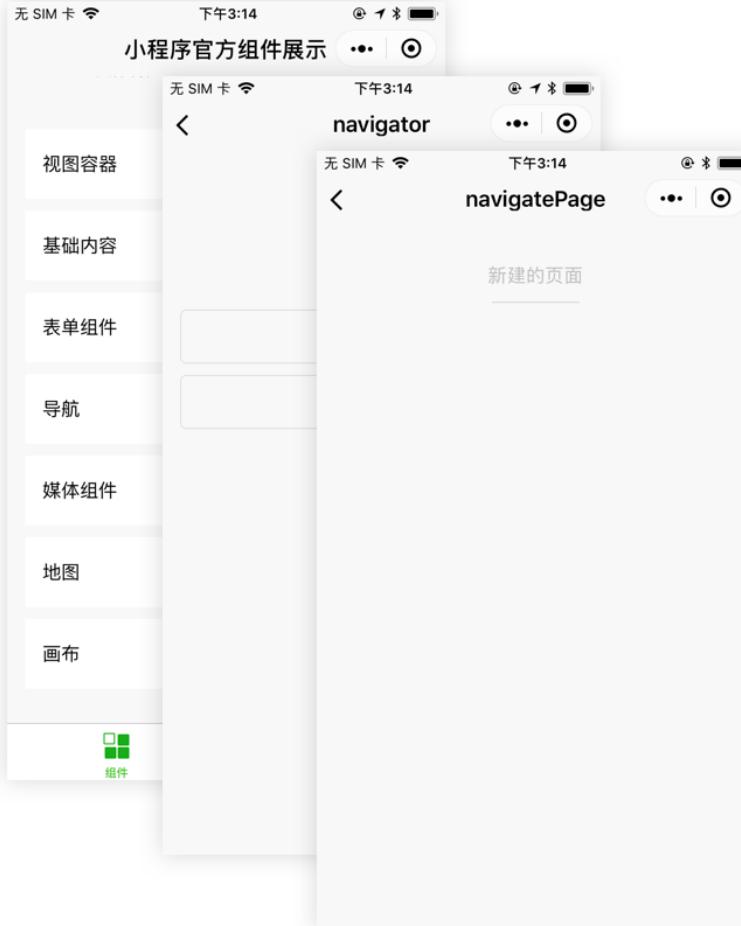
- 此外需要注意以下3点：

- 直接修改 Page实例的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。
- 由于setData是需要两个线程的一些通信消耗，为了提高性能，每次设置的数据不应超过1024kB。
- 不要把data中的任意一项的value设为undefined，否则可能会有引起一些不可预料的bug。

- 页面的用户行为：小程序宿主环境提供了四个和页面相关的用户行为回调
  - 下拉刷新 `onPullDownRefresh` 监听用户下拉刷新事件
  - 上拉触底 `onReachBottom` 监听用户上拉触底事件
  - 页面滚动 `onPageScroll` 监听用户滑动页面事件
  - 用户转发 `onShareAppMessage` 只有定义了此事件处理函数，右上角菜单才会显示“转发”按钮
  - 代码清单3-13 使用`onShareAppMessage`自定义转发字段

```
// page.js
Page({
 onShareAppMessage: function () {
 return {
 title: '自定义转发标题',
 path: '/page/user?id=123'
 }
 }
})
```

- 页面跳转和路由 - 页面栈
  - 一个小程序拥有多个页面，可以通过wx.navigateTo推入一个新的页面



- 页面路由：
  - 假设页面栈： [ pageA, pageB, pageC ]
  - 其中pageA在最底下， pageC在最顶上
  - 使用 `wx.navigateTo({ url: 'pageD' })` 可以往当前页面栈多推入一个 pageD
    - 此时页面栈变成 [ pageA, pageB, pageC, pageD ]
  - 使用 `wx.navigateBack()` 可以退出当前页面栈的最顶上页面
    - 此时页面栈变成 [ pageA, pageB, pageC ]
  - 使用 `wx.redirectTo({ url: 'pageE' })` 是替换当前页变成pageE
    - 此时页面栈变成 [ pageA, pageB, pageE ]，当页面栈到达10层没法再新增的时候，往往就是使用redirectTo这个API进行页面跳转

- 页面路由：

表3-5 页面路由触发方式及页面生命周期函数的对应关系

路由方式	触发时机	路由前页面生命周期	路由后页面生命周期
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	onHide	onLoad, onShow
页面重定向 调用	API wx.redirectTo	onUnload	onLoad, onShow
页面返回 调用	API wx.navigateBack	onUnload	onShow
Tab	切换 调用 API wx.switchTab	请参考表3-6	请参考表3-6
重启动	调用 API wx.reLaunch	onUnload	onLoad, onShow

- 页面路由：

表3-6 页面路由触发方式及页面生命周期函数的对应关系

当前页面	路由后页面	触发的生命周期（按顺序）
A	A	无
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(从转发进入)	A	D.onUnload(), A.onload(), A.onShow()
D(从转发进入)	B	D.onUnload(), B.onload(), B.onShow()

- 一个小程序页面可以分解成多个部分组成
  - 组件就是小程序页面的基本组成单元

表3-7 组件共有属性

属性名	类型	描述	其他说明
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	可以通过数据绑定进行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	事件	详情见3.5节

- 一个小程序页面可以分解成多个部分组成
  - 组件就是小程序页面的基本组成单元

表3-8 Image图片组件属性

属性名	类型	默认值	描述
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
binderror	HandleEvent		当错误发生时触发事件，事件对象event.detail = {errMsg: 'something wrong'}
bindload	HandleEvent		当图片载入完毕时触发事件，事件对象event.detail = {height:'图片高度px', width:'图片宽度px'}

- <https://mp.weixin.qq.com/debug/wxadoc/dev/component/>

- 宿主环境提供了丰富的API，可以很方便调起微信提供的能力
  - <https://mp.weixin.qq.com/debug/wxadoc/dev/api/>
- 小程序提供的API按照功能主要分为几大类：
  - 网络、媒体、文件、数据缓存、位置、设备、界面、界面节点信息还有一些特殊的开放接口
- API一般调用的约定：
  - wx.on\* 开头的 API 是监听某个事件发生的API接口，接受一个 Callback 函数作为参数。当该事件触发时，会调用 Callback 函数
  - 如未特殊约定，多数 API 接口为异步接口，都接受一个Object作为参数
  - API的Object参数一般由success、fail、complete三个回调来接收接口调用结果
  - wx.get\* 开头的API是获取宿主环境数据的接口
  - wx.set\* 开头的API是写入数据到宿主环境的接口

代码清单3-17 通过wx.request发起网络请求

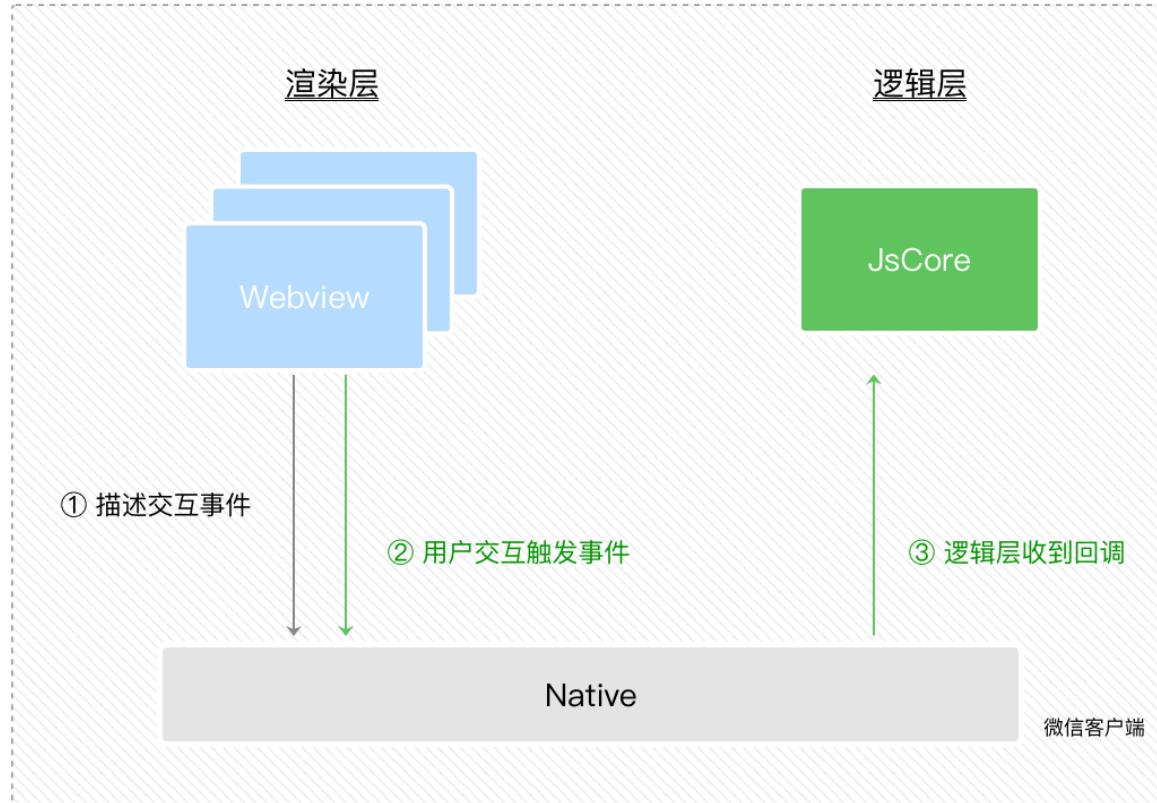
```
wx.request({
 url: 'test.php',
 data: {},
 header: { 'content-type': 'application/json' },
 success: function(res) {
 // 收到https服务成功后返回
 console.log(res.data)
 },
 fail: function() {
 // 发生网络错误等情况触发
 },
 complete: function() {
 // 成功或者失败后触发
 }
})
```

## • API调用

代表3-9 API接口回调说明

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

- UI界面的程序与用户的互动
  - 用户在渲染层的行为反馈”以及“组件的部分状态反馈”抽象为渲染层传递给逻辑层的“事件”



```
<!-- page.wxml -->
<view id="tapTest" data-hi="WeChat" bindtap="tapName"> Click me! </view>

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

- 常见的事件类型

类型	触发条件
touchstart	手指触摸动作开始
touchmove	手指触摸后移动
touchcancel	手指触摸动作被打断, 如来电提醒, 弹窗
touchend	手指触摸动作结束
tap	手指触摸后马上离开
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
longtap	手指触摸后, 超过350ms再离开 (推荐使用longpress事件代替)
transitionend	会在 WXSS transition 或 wx.createAnimation 动画结束后触发
animationstart	会在一个 WXSS animation 动画开始时触发
animationiteration	会在一个 WXSS animation 一次迭代结束时触发
animationend	会在一个 WXSS animation 动画完成时触发

- 事件对象属性

属性	类型	说明
type	String	事件类型
timeStamp	Integer	页面打开到触发事件所经过的毫秒数
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

- 事件对象示例

```
<!-- page.wxml -->
<view id="outer" catchtap="handleTap">
 <view id="inner">点击我</view>
</view>
```

```
// page.js
Page({
 handleTap: function(evt) {
 // 当点击inner节点时
 // evt.target 是inner view组件
 // evt.currentTarget 是绑定了handleTap的outer view组件
 // evt.type == "tap"
 // evt.timeStamp == 1542
 // evt.detail == {x: 270, y: 63}
 // evt.touches == [{identifier: 0, pageX: 270, pageY: 63, clientX: 270, clientY: 63}]
 // evt.changedTouches == [{identifier: 0, pageX: 270, pageY: 63, clientX: 270, clientY: 63}]
 }
})
```

- target和currentTarget事件对象属性

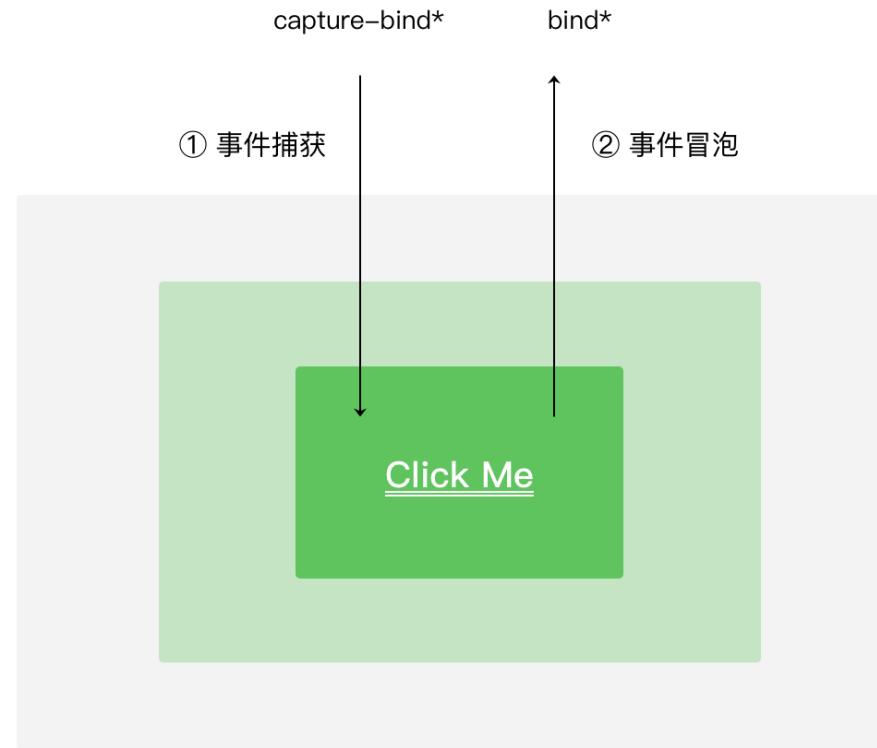
属性	类型	说明
id	String	当前组件的id
tagName	String	当前组件的类型
dataset	Object	当前组件上由data-开头的自定义属性组成的集合

- touch和changedTouches对象属性

属性	类型	说明
identifier	Number	触摸点的标识符
pageX, pageY	Number	距离文档左上角的距离， 文档的左上角为原点， 横向为X轴， 纵向为Y轴
clientX, clientY	Number	距离页面可显示区域（屏幕除去导航条）左上角距离， 横向为X轴， 纵向为Y轴

- 事件绑定与冒泡捕获

- 事件绑定的写法和组件属性一致，以key="value"的形式，其中：
    - key以bind或者catch开头，然后跟上事件的类型。
    - value是一个字符串，需要在对应的页面Page构造器中定义同名的函数



- 以下示例中
  - 点击 inner view 会先后调用handleTap2、handleTap4、handleTap3、handleTap1

```
<view
 id="outer"
 bind:touchstart="handleTap1"
 capture-bind:touchstart="handleTap2"
>
 outer view
<view
 id="inner"
 bind:touchstart="handleTap3"
 capture-bind:touchstart="handleTap4"
>
 inner view
</view>
</view>
```

- 事件绑定与冒泡

- bind事件绑定不会阻止冒泡事件向上冒泡
- catch事件绑定可以阻止冒泡事件向上冒泡
- 如果将上面代码中的第一个capture-bind改为capture-catch，将只触发handleTap2(capture-catch将中断捕获阶段和取消冒泡阶段)

```
<view
 id="outer"
 bind:touchstart="handleTap1"
 capture-catch:touchstart="handleTap2"
>
 outer view
<view
 id="inner"
 bind:touchstart="handleTap3"
 capture-bind:touchstart="handleTap4"
>
 inner view
</view>
</view>
```

- 通过wx.getSystemInfoSync获取宿主环境信息

```
wx.getSystemInfoSync()
/*
{
 brand: "iPhone", // 手机品牌
 model: "iPhone 6", // 手机型号
 platform: "ios", // 客户端平台
 system: "iOS 9.3.4", // 操作系统版本
 version: "6.5.23", // 微信版本号
 SDKVersion: "1.7.0", // 小程序基础库版本
 language: "zh_CN", // 微信设置的语言
 pixelRatio: 2, // 设备像素比
 screenWidth: 667, // 屏幕宽度
 screenHeight: 375, // 屏幕高度
 windowHeight: 375, // 可使用窗口高度
 windowWidth: 667, // 可使用窗口宽度
 fontSizeSetting: 16 // 用户字体大小设置
}
*/
```

```
if (wx.openBluetoothAdapter) {
 wx.openBluetoothAdapter()
} else {
 // 如果希望用户在最新版本的客户端上体验您的小程序，可以这样子提示
 wx.showModal({
 title: '提示',
 content: '当前微信版本过低，无法使用该功能，请升级到最新微信版本后重试。'
 })
}

// 判断接口及其参数在宿主环境是否可用
wx.canIUse('openBluetoothAdapter')
wx.canIUse('getSystemInfoSync.return.screenWidth')
wx.canIUse('getSystemInfo.success.screenWidth')
wx.canIUse('showToast.object.image')
wx.canIUse('onCompassChange.callback.direction')
wx.canIUse('request.object.method.GET')

// 判断组件及其属性在宿主环境是否可用
wx.canIUse('contact-button')
wx.canIUse('text.selectable')
wx.canIUse('button.open-type.contact')
```

- 如果我们需要从 <https://test.com/getinfo> 接口拉取用户信息，其代码示例如下所示

```
wx.request({

 url: 'https://test.com/getinfo',

 success: function(res) {

 console.log(res)// 服务器回包信息

 }

})
```

## • wx.request详细参数

参数名	类型	必填	默认值	描述
url	String	是		开发者服务器接口地址
data	Object/String	否		请求的参数
header	Object	否		设置请求的 header, header 中不能设置 Referer, 默认 header['content-type'] = 'application/json'
method	String	否	GET	(需大写) 有效值: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
dataType	String	否	json	回包的内容格式, 如果设为json, 会尝试对返回的数据做一次 JSON解析
success	Function	否		收到开发者服务成功返回的回调函数, 其参数是一个 Object, 见表4-2。
fail	Function	否		接口调用失败的回调函数
complete	Function	否		接口调用结束的回调函数 (调用成功、失败都会执行)

- 请求参数
  - 通过wx.request这个API，有两种方法把数据传递到服务器：通过url上的参数以及通过data参数

```
// 通过url参数传递数据
wx.request({
 url:'https://test.com/getinfo?id=1&version=1.0.0',
 success: function(res) {
 console.log(res)// 服务器回包信息
 }
})
```

```
// 通过data参数传递数据
wx.request({
 url: 'https://test.com/getinfo',
 data: { id:1, version:'1.0.0' },
 success: function(res) {
 console.log(res)// 服务器回包信息
 }
})
```

- 请求参数
  - wx.request发起POST请求包体使用json格式

```
// 请求的包体为 {"a":{"b":[1,2,3],"c":{"d":"test"}}}
wx.request({
 url: 'https://test.com/postdata',
 method: 'POST',
 header: { 'content-type': 'application/json' },
 data: {
 a: {
 b: [1, 2, 3],
 c: { d: "test" }
 }
 },
 success: function(res) {
 console.log(res)// 服务器回包信息
 }
})
```

- 收到回包
  - wx.request的success返回参数

参数名	类型	描述
data	Object/String	开发者服务器返回的数据
statusCode	Number	开发者服务器返回的 HTTP 状态码
header	Object	开发者服务器返回的 HTTP Response Header

# 设置超时时间



REliable, INtelligent & Scalable Systems

- app.json指定wx.request超时时间为3000毫秒

```
{
 "networkTimeout": {
 "request": 3000
 }
}
```

# 请求前后的状态处理



REliable, INtelligent & Scalable Systems

```
var hasClick = false;
Page({
 tap: function() {
 if (hasClick) {
 return
 }
 hasClick = true
 wx.showLoading()
 wx.request({
 url: 'https://test.com/getinfo',
 method: 'POST',
 header: { 'content-type': 'application/json' },
 data: {},
 success: function (res) {
 if (res.statusCode === 200) {
 console.log(res.data)// 服务器回包内容
 }
 },
 fail: function (res) {
 wx.showToast({ title: '系统错误' })
 },
 complete: function (res) {
 wx.hideLoading()
 hasClick = false
 }
 })
 }
})
```

- 微信小程序开发文档
  - <https://developers.weixin.qq.com/miniprogram/dev/framework/>
- 微信小程序开发者社区
  - <https://developers.weixin.qq.com/>



- *Web*开发技术
- *Web Application Development*

Thank You!