# 小程序

# WXML模板（HTML）

## 逻辑与法

三元运算：

<!-- 根据 a 的值是否等于 10 在页面输出不同的内容 -->

<text>{{ a === 10? "变量 a 等于10": "变量 a 不等于10"}}</text>

算数运算：

<!-- { a: 1, b: 2, c: 3 } -->

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

<!-- 输出 3 + 3 + d -->

字符串拼接：

<!-- { name: 'world' } -->

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

<!-- 输出 hello world -->

## 条件逻辑

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。

<view wx:if="{{length > 5}}"> 1 </view>

<view wx:elif="{{length > 2}}"> 2 </view>

<view wx:else> 3 </view>

## 列表渲染

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

<!-- 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。例如：

<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>

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'},

]

}

})

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

<!--

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>

## 引用

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，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。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'}}"/>

## 共同属性

所有wxml 标签都支持的属性称之为共同属性。

Id 组件的唯一标识 整个页面唯一

Class 组件的样式类 在对应的 WXSS 中定义的样式类

Style 组件的内联样式 可以动态设置的内联样式

Hidden 组件是否显示 所有组件默认显示

data-\* 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\* 组件的事件

# WXSS样式（CSS）

项目公共样式：根目录中的app.wxss为项目公共样式，它会被注入到小程序的每个页面。

页面样式：与app.json注册过的页面同名且位置同级的WXSS文件。app.json注册了pages/rpx/index页面，那pages/rpx/index.wxss为页面pages/rpx/index.wxml的样式。

其它样式：其它样式可以被项目公共样式和页面样式引用。

## 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。

小程序编译后，rpx会做一次px换算。换算是以375个物理像素为基准，也就是在一个宽度为375物理像素的屏幕下，1rpx = 1px。举个例子：iPhone6屏幕宽度为375px，共750个物理像素，那么1rpx = 375 / 750 px = 0.5px。

## WXSS引用

在CSS中：@import url('./test\_0.css')

在小程序中：@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。

## 内联样式

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>

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。

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

}

# JavaScript脚本

## 执行环境

小程序目前可以运行在三大平台：

iOS平台，包括iOS9、iOS10、iOS11

Android平台

小程序ID

## 模块化

在需要使用这些模块的文件中，使用 require(path) 将公共代码引入，被调用的方法在js中必须是exports状态。

<!—common.js-->

module.exports = {

sayHello: function(text},

sayGoodbye: function(text){}

}

<!—test.js-->

var common = require('common.js')

Page({

helloMINA: function() {

common.sayHello('MINA')

},

goodbyeMINA: function() {

common.sayGoodbye('MINA')

}

})

## 全局变量

当需要使用全局变量的时，通过使用全局函数 getApp() 获取全局的实例，并设置相关属性值。

// a.js

// 获取全局变量

var global = getApp()

global.globalValue = 'globalValue'

// b.js

// 访问全局变量

var global = getApp()

console.log(global.globalValue) // 输出 globalValue

当需要保证全局的数据可以在任何文件中安全的被使用到，那么可以在 App() 中进行设置

// app.js

App({

globalData: 1

})

# 生命周期

## 程序构造器App

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。

获取App实例

// other.js

var appInstance = getApp()

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(options) {},

//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) {},

globalData: 'I am global data'

})

## 程序构造器Page

宿主环境提供了 Page() 构造器用来注册一个小程序页面，Page()在页面脚本page.js中调用。

Page({

//页面的初始数据

data: { text: "This is page data." },

//生命周期函数--监听页面加载，触发时机早于onShow和onReady

onLoad: function(options) { },

//生命周期函数--监听页面初次渲染完成

onReady: function() { },

//生命周期函数--监听页面显示，触发事件早于onReady

onShow: function() { },

//生命周期函数--监听页面隐藏

onHide: function() { },

//生命周期函数--监听页面卸载

onUnload: function() { },

//页面相关事件处理函数--监听用户下拉动作

onPullDownRefresh: function() { },

//页面上拉触底事件的处理函数

onReachBottom: function() { },

//用户点击右上角转发

onShareAppMessage: function () { },

//页面滚动触发事件的处理函数

onPageScroll: function() { }

})

页面数据

// page.js

Page({

onLoad: function(){

this.setData({

text: 'change data'

}, function(){

// 在这次setData对界面渲染完毕后触发

})

}

})