

# WXSS&WXML&WXS

讲师: 王红元  
微博: coderwhy



小程序交流群



加小撩拿资料



和老师交流



# 页面样式写法

## ■ 页面样式的三种写法：

- 行内样式、页面样式、全局样式
- 三种样式都可以作用于页面的组件

## ■ 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

```
home.wxml  ×  app.wxss  home.wxss
1  <!-- 1.行内样式 -->
2  <view style='color: red; font-size:20px;'>行内样式</view>
3
4  <!-- 2.页面样式 -->
5  <view class='box'>页面样式</view>
6
7  <!-- 3.全局样式 -->
8  <view class='container'>全局样式</view>
9
10 <!-- 4.三种样式同时作用 -->
11 <view style='color: orange; background: blue;' class='page app'>三种样式的作用</view>
12
```

```
home.wxss  ×
1  .box {
2    color: blue;
3    font-size: 25px;
4  }
5
6  .page {
7    font-size: 35px;
8    background: green;
9  }
10
```

```
app.wxss  ×
1  .container {
2    color: green;
3    font-size: 30px;
4  }
5
6  .app {
7    text-decoration: underline;
8    background: purple;
9  }
10
```



# 支持的选择器

| 选择器      | 样例           | 样例描述            |
|----------|--------------|-----------------|
| .class   |              |                 |
| #id      |              |                 |
| element  | ! important  | style=""        |
| element  | ∞            | 1000            |
|          |              | #id             |
|          |              | 100             |
|          |              | .class          |
|          |              | 10              |
|          |              | element         |
|          |              | 1               |
| ::after  |              |                 |
| ::before | view::before | 在 view 组件前边插入内容 |





# wxss的扩展 – 尺寸单位

## ■ 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 $750\text{rpx} = 375\text{px} = 750\text{物理像素}$ ,  $1\text{rpx} = 0.5\text{px} = 1\text{物理像素}$ 。

| 设备           | rpx换算px (屏幕宽度/750)             | px换算rpx (750/屏幕宽度)            |
|--------------|--------------------------------|-------------------------------|
| iPhone5      | $1\text{rpx} = 0.42\text{px}$  | $1\text{px} = 2.34\text{rpx}$ |
| iPhone6      | $1\text{rpx} = 0.5\text{px}$   | $1\text{px} = 2\text{rpx}$    |
| iPhone6 Plus | $1\text{rpx} = 0.552\text{px}$ | $1\text{px} = 1.81\text{rpx}$ |



- **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。



# wxss的扩展 – 样式导入

## ■ 为什么使用样式导入？

- 在某些情况下，我们可能会将样式分在多个wxss文件中，方便对样式的管理。
- 这个时候，我们就可以使用样式导入，来让单独的wxss生效

## ■ 我们可以在一个wxss中导入另一个wxss文件：

- 1.使用@import进行导入
- 2. @import后跟需要导入的外联样式表的相对路径（或者绝对路径也可以），用;表示语句结束。

## ■ 导入的位置在哪里？

- 可以在app.wxss中导入这个样式
- 也可以在page.wxss导入这个样式

import.wxml ×

```
1 <!--pages/import/import.wxml-->
2 <text>pages/import/import.wxml</text>
3
4 <view class='box'>我是view组件</view>
5 |
```

app.wxss ●

```
1 @import "/css/normal.wxss";
2
```

import.wxss ×

```
1 /* pages/import/import.wxss */
2 @import "/css/normal.wxss";
3
```





# 官方样式库

- 为了减少开发者样式开发的工作量，小程序官方提供了WeUI.wxss基本样式库
- <https://github.com/Tencent/weui-wxss>





# Mustache语法(一)

## ■ WXML基本格式:

- 类似于HTML代码: 比如可以写成单标签, 也可以写成双标签
- 必须有严格的闭合: 没有闭合会导致编译错误
- 大小写敏感: class和Class是不同的属性

## ■ 开发中, 界面上展示的数据并不是写死的, 而是会根据服务器返回的数据, 或者用户的操作来进行改变.

- 如果使用原生JS或者jQuery的话, 我们需要通过操作DOM来进行界面的更新.
- 小程序和Vue/React一样, 提供了插值语法: **Mustache语法(双大括号)**

```
3 <!-- 1.Mustache语法基本使用 -->
4 <view>{{message}}</view>
5 <view>{{firstname}} {{lastname}}</view>
6 <view>当前时间: {{time}}</view>
```

```
Hello World
你好啊,李银河
Kobe Bryant
当前时间: 2019/5/7 下午9:47:57
```

```
home.js  x
1 Page({
2   data: {
3     message: "你好啊,李银河",
4     firstname: 'Kobe',
5     lastname: 'Bryant',
6     time: new Date().toLocaleString()
7   },
8   onLoad() {
9     setInterval(() => {
10      this.setData({
11        time: new Date().toLocaleString()
12      })
13    }, 1000)
14  }
15 })
16
```





# Mustache语法(二)

- Mustache语法不仅仅可以直接显示数据, 也可以使用表达式:

```
<!-- 2.Mustache语法表达式 -->
<view>{{ age >= 18 ? "成年人": "未成年人"}}</view>
<view>{{age + 20}}</view>
<view>{{age + "岁"}}</view>
```

- 并且可以绑定到属性:

```
<!-- 3.绑定属性 -->
<view class='content {{active ? "active": ""}}'>我是view</view>
```

```
/* home.wxss */
.content {
  font-size: 50rpx;
}

.active {
  color: red;
}
```

```
data: {
  message: "你好啊, 李银河",
  time: (new Date()).toLocaleString(),
  age: 20,
  active: true
},
```





# 逻辑判断 wx:if – wx:elif – wx:else

- 某些时候, 我们需要根据条件来决定一些内容是否渲染:

- 当条件为true时, view组件会渲染出来
- 当条件为false时, view组件不会渲染出来

```
<!-- 1.直接传入true/false -->  
<view wx:if="{{true}}">是否渲染的内容</view>
```

- 根据按钮点击, 决定是否渲染:

```
<!-- 2.控制是否渲染 -->  
<button size='mini' bind:tap="onToggle">切换</button>  
<view wx:if="{{isShow}}">我是内容,哈哈</view>
```

- 也可以有多个条件:

```
<!-- 3.多个条件判断 -->  
<button size='mini' bindtap='onIncrement'+10</button>  
<view wx:if="{{score > 90}}">优秀</view>  
<view wx:elif="{{score > 80}}">良好</view>  
<view wx:elif="{{score > 60}}">及格</view>  
<view wx:else>不及格</view>
```



# 逻辑判断补充二：

## ■ hidden属性:

- hidden是所有的组件都默认拥有的属性,
- 当hidden属性为true时, 组件会被隐藏
- 当hidden属性为false时, 组件会显示出来

```
<!-- hidden属性 -->  
<button size='mini' bindtap="onShowToggle">hidden属性</button>  
<view hidden="{{isShow}}">hidden控制内容是否显示</view>
```

## ■ hidden和wx:if的区别

- hidden控制隐藏和显示是控制是否添加hidden属性
- wx:if是控制组件是否渲染的

```
<view><button size='mini' bindtap='onToggle'>切换</button></view>  
<view wx:if="{{!isShow}}">我是内容, 呵呵呵</view>  
<view hidden='{{isShow}}'>我是内容, 哈哈</view>
```





# 列表渲染 – wx:for基础

## ■ 为什么使用wx:for?

- 我们知道，在实际开发中，服务器经常返回各种列表数据，我们不可能一一从列表中取出数据进行展示；
- 需要通过for循环的方式，遍历所有的数据，一次性进行展示；

## ■ 在组件中，我们可以使用wx:for来遍历一个数组（字符串 - 数字）

- 默认情况下，遍历后在wxml中可以使用一个变量index，保存的是当前遍历数据的下标值。
- 数组中对应某项的数据，使用变量名item获取。

```
<!-- 1.遍历数据 -->
<!-- 1.1.遍历一个数组 -->
<view wx:for="{{['abc', 'cba', 'nba']}}">{{index}}.{{item}}</view>
<!-- 1.2.遍历一个字符串 -->
<view wx:for="abcd">{{index}}.{{item}}</view>
<!-- 1.3.遍历一个数字 -->
<view wx:for="{{5}}">{{index}}.{{item}}</view>
```





# block标签

## ■ 什么是block标签？

- 某些情况下，我们需要使用 wx:if 或 wx:for时，可能需要包裹一组组件标签
- 我们希望对这一组组件标签进行整体的操作，这个时候怎么办呢？

## ■ 方式一：使用一个view组件包裹：

```
<view wx:if="{{isShow}}">
  <view>哈哈</view>
  <view>呵呵</view>
</view>
```

```
<view wx:for="{{movies}}">
  <view>电影序号:{{index}}</view>
  <view>电影名称:{{item}}</view>
</view>
```

## ■ 方式二：使用block标签

```
<block wx:if="{{isShow}}">
  <view>哈哈</view>
  <view>呵呵</view>
</block>
```

```
<block wx:for="{{movies}}">
  <view>电影序号:{{index}}</view>
  <view>电影名称:{{item}}</view>
</block>
```



# block标签的意义

- 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。
- 使用block有两个好处：
  - 1) 将需要进行遍历或者判断的内容进行包裹。
  - 2) 将遍历和判断的属性放在block便签中，不影响普通属性的阅读，提高代码的可读性。





# 列表渲染 – item/index名称

- 默认情况下，item – index的名字是固定的
  - 但是某些情况下，我们可能想使用其他名称
  - 或者当出现多层遍历时，名字会重复
- 这个时候，我们可以指定item和index的名称：

```
<!-- 2.item/index命名 -->
<block wx:for="{{movies}}"
      wx:for-item="movie"
      wx:for-index="i">
  <view>{{i}}.{{movie}}</view>
</block>
```





# 列表渲染 – key作用

## ■ 我们看到，使用wx:for时，会报一个警告：

□ 这个提示告诉我们，可以添加一个key来**提供性能**。

## ■ 为什么需要这个key属性呢（了解）？

□ 这个其实和小程序内部也使用了虚拟DOM有关系（和Vue、React很相似）。

## ■ 当某一层有很多相同的节点时，也就是列表节点时，我们希望**插入一个新的节点**

□ 我们希望可以在B和C之间加一个F，Diff算法默认执行起来是这样的。

□ 即把C更新成F，D更新成C，E更新成D，最后再插入E，是不是很没有效率？

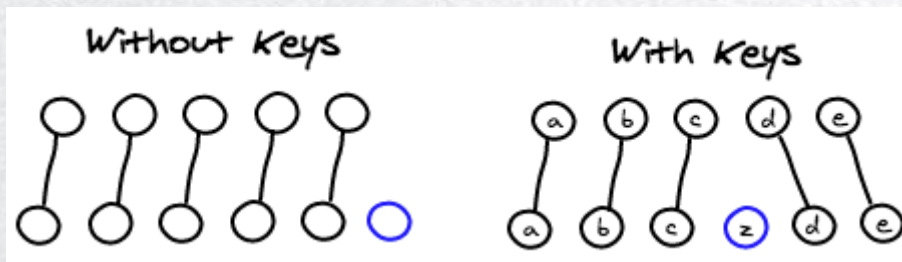
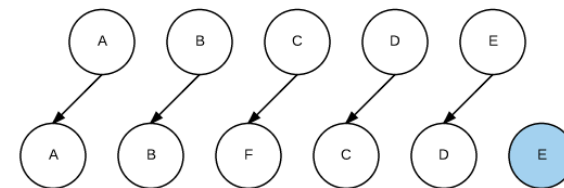
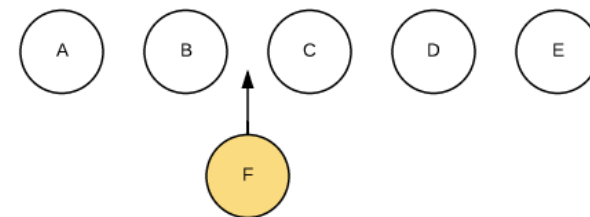
## ■ 所以我们需要使用key来给每个节点**做一个唯一标识**

□ Diff算法就可以正确的识别此节点

□ 找到正确的位置去插入新的节点。

## ■ 所以一句话，**key的作用主要是为了高效的更新虚拟DOM。**

```
./pages/loop/loop.wxml
Now you can provide attr `wx:key` for a `wx:for` to improve performance.
11 |
12 | <!-- 2.item/index命名 -->
> 13 | <block wx:for="{{movies}}"
    | ^
14 |     wx:for-item="movie"
15 |     wx:for-index="i">
16 |     <view>{{i}}.{{movie}}</view>
```

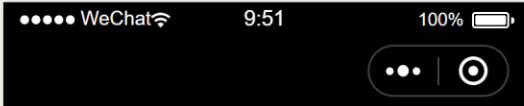




# 模板用法

■ WXML提供**模板 (template)**，可以在模板中定义代码片段，在不同的地方调用。(是一种wxml代码的复用机制)

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



pages/template/template.wxml

0.哈哈-2018.08.08

1.呵呵-2019.09.09

2.嘻嘻-2020.10.10

3.嘿嘿-2021.11.11

奇数: 11

偶数: 22

奇数: 7

奇数: 13

偶数: 100

- pages
  - block
  - condition
  - hidden
  - home
  - key
  - loop
  - template
- JS app.js
- app.json
- wxss app.wxss
- project.config.json
- sitemap.json

```
1 <!--pages/template/template.wxml-->
2 <text>pages/template/template.wxml</text>
3
4 <!-- 1.定义模板 -->
5 <template name="msgItem">
6   <view>{{index}}.{{content}}-{{time}}</view>
7 </template>
8
9 <!-- 2.使用模板 -->
10 <template is="msgItem" data="{index: 0, content: '哈哈', time: '2018.08.08'}"/>
11 <template is="msgItem" data="{index: 1, content: '呵呵', time: '2019.09.09'}"/>
12 <template is="msgItem" data="{index: 2, content: '嘻嘻', time: '2020.10.10'}"/>
13 <!-- 使用ES6的扩展运算符 -->
14 <template is="msgItem" data="{...item}"/>
15
16 <!-- 3.根据条件使用不同的模板 -->
17 <template name="odd">
18   <view>奇数: {{num}}</view>
19 </template>
20
21 <template name="even">
22   <view>偶数: {{num}}</view>
23 </template>
24
25 <block wx:for="{[11, 22, 7, 13, 100]}">
26   <template is="{{item % 2 == 0 ? 'even': 'odd'}}" data="{num: item}"/>
27 </block>
```





# wxml的引入

■ 小程序wxml中提供了两种引入文件的方式：import和include

■ **Import引入**：import 可以在该文件中使用目标文件定义的 template

■ 比如下面的演练：

□ 在item.wxml中定义一个item的template

□ 在home.wxml中引入，并且使用template

```
1 <template name="item">
2   <view>我是一个item的template: {{text}}</view>
3 </template>
```

```
1 <import src="./wxml/item.wxml"/>
2 <template is="item" data="{{text: '哈哈'}}"/>
```

■ 注意：wxml中不能递归引入（也就是A引入了B的template，不会引入B中引入C的template）

```
1 <template name="b">
2   <view>我是B的template</view>
3 </template>
```

```
1 <import src="./B.wxml"/>
2
3 <template name="a">
4   <view>我是A的template</view>
5 </template>
```

```
1 <import src="./wxml/A.wxml"/>
2 <template is="a"/>
3 <template is="b"/> // 这里会报警告，
```





# include引入

- include 可以将目标文件中除了 `<template/>` `<wxs/>` 外的整个代码引入，相当于是拷贝到 include 位置：

nav.wxml

```
1 <view>我是导航</view>
```

header.wxml

```
1 <include src="nav.wxml"/>
2 <view>我是头部</view>
```

footer.wxml

```
1 <view>我是尾部</view>
```

home.wxml

```
1 <include src="/wxml/header.wxml"/>
2
3 <text>Hello World</text>
4 <view class='title'>你好,小程序</view>
5
6 <include src="/wxml/footer.wxml"/>
```



# WXS模块

■ **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。

□ 官方：WXS 与 JavaScript 是不同的语言，有自己的语法，并不和 JavaScript 一致。（不过基本一致）

■ **为什么要设计WXS语言呢？**

□ 在WXML中是不能直接调用Page/Component中定义的函数的。

□ 但是某些情况，我们可以希望使用函数来处理WXML中的数据(类似于Vue中的过滤器)，这个时候就使用WXS了

■ **WXS使用的限制和特点：**

□ 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。

□ WXS 函数不能作为组件的事件回调。

□ 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。





# WXS的写法

## ■ WXS有两种写法：

- - 写在<wxs>标签中
- - 写在以.wxs结尾的文件中

## ■ <wxs>标签的属性：

| 属性名    | 类型     | 默认值 | 说明  |
|--------|--------|-----|---|
| module | String |     | 当前 <code>&lt;wxs&gt;</code> 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 <b>单闭合标签</b> 或 <b>标签的内容为空</b> 时有效。 |





# wxs的简单演练

写在wxs标签中:

```
1 <wxs module="info_m">
2   var foo = "Hello World";
3   var bar = function() {
4     return "你好, 小程序"
5   }
6
7   module.exports = {
8     foo: foo,
9     bar: bar
10  }
11 </wxs>
12
13 <text>pages/home/home.wxml</text>
14 <view>{{info_m.foo}}</view>
15 <view>{{info_m.bar()}}</view>
```

写在.wxs文件中:

- 创建一个.wxs文件:

```
1 var name = 'why'
2 var age = 18
3
4 function sayHello() {
5   return 'Hello Flutter'
6 }
7
8 module.exports = {
9   name: name,
10  age: age,
11  sayHello: sayHello
12 }
```

- 在wxml中使用

```
1 <wxs src="../../tools/info_n.wxs" module="info_n"/>
2 <view>{{info_n.name}}</view>
3 <view>{{info_n.age}}</view>
4 <view>{{info_n.sayHello()}}</view>
```



# WXS练习

## ■ 练习一: 价格保留小数

## ■ 练习二: 日期格式化

```
1 function priceFormat(price, number) {
2   var number = number || 2;
3
4   // 1.转成浮点型
5   var f_price = parseFloat(price);
6
7   // 2.保留几位小数
8   return f_price.toFixed(number)
9 }
10
11
12 module.exports = {
13   priceFormat: priceFormat
14 }
```

```
1 var dateFormat = function (timestamp, format) {
2   if (!format) {
3     format = "yyyy-MM-dd hh:mm:ss";
4   }
5   timestamp = parseInt(timestamp * 1000);
6   var realDate = getDate(timestamp);
7   function timeFormat(num) {
8     return num < 10 ? '0' + num : num;
9   }
10  var date = [
11    ["M+", timeFormat(realDate.getMonth() + 1)],
12    ["d+", timeFormat(realDate.getDate())],
13    ["h+", timeFormat(realDate.getHours())],
14    ["m+", timeFormat(realDate.getMinutes())],
15    ["s+", timeFormat(realDate.getSeconds())],
16    ["q+", Math.floor((realDate.getMonth() + 3) / 3)],
17    ["S+", realDate.getMilliseconds()],
18  ];
19
20  var regYear = getRegExp("(y+)", "i");
21  var reg1 = regYear.exec(format);
22  // console.log(reg1[0]);
23  if (reg1) {
24    format = format.replace(reg1[1], (realDate.getFullYear() + '').substring(4 -
25      reg1[1].length));
26  }
27  for (var i = 0; i < date.length; i++) {
28    var k = date[i][0];
29    var v = date[i][1];
30
31    var reg2 = getRegExp("(" + k + ")").exec(format);
32    if (reg2) {
33      format = format.replace(reg2[1], reg2[1].length == 1
34        ? v : ("00" + v).substring(("" + v).length));
35    }
36  }
37  return format;
38 }
```