微信小程序学习摘要系列《三》视图层

MINA 的视图层由 WXML 与 WXSS 编写。

将逻辑层的数据反应成视图，同时将视图层的事件发送给逻辑层。

WXML(WeiXin Markup language)用于描述页面的结构。

WXSS(WeiXin Style Sheet)用于描述页面的样式。

组件(Component)是视图的基本组成单元。

WXML-数据绑定

WXML 中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：

内容

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

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

组件属性(需要在双引号之内)

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

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

```
})
```

控制属性(需要在双引号之内)

```
<view wx:if="{{condition}}"> </view>
Page({
  data: {
    condition: true
  }
})
```

运算

可以在{{}}内进行简单的运算，支持的有如下几种方式：

三元运算

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

算数运算

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

Page({
  data: {
    a: 1,
    b: 2,
    c: 3
  }
})
```

view 中的内容为 3 + 3 + d

逻辑判断

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

字符串运算

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

Page({
  data:{
    name:"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

数组

```
<view wx:for-items="{{[zero, 1, 2, 3, 4]}}"> {{item}} </view>

Page({
  data: {
```

```
    zero: 0
  }
})
```

最终组合成数组[0, 1, 2, 3, 4]

对象

```
<template is="objectCombine" data="{{for: a, bar: b}}"></template>
Page({
  data: {
    a: 1,
    b: 2
  }
})
```

最终组合成的对象是{for: 1, bar: 2}

WXML-条件渲染

wx:if

在 MINA 中，我们用 wx:if="{{condition}}"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>
```

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。但是如果我们想一次性判断多个组件标签，我们可以使用一个 <block/>标签将多个组件包装起来，并在上边使用 wx:if 控制属性。

```
<block wx:if="{{true}}">
  <view> view1 </view>
  <view> view2 </view>
</block>
```

注意： <block/>并不是一个组件，它仅仅是一个包装元素，不会在页面中做任何渲染，只接受控制属性。

wx:if vs hidden

因为 wx:if 之中的模板也可能包含数据绑定，所有当 wx:if 的条件值切换时，MINA 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，MINA 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。

WXML-列表渲染

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="{{items}}">
  {{index}}: {{item.message}}
</view>
Page({
  items: [{
    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>

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>
```

WXML-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。



定义模板

使用 name 属性，作为模板的名字。然后在<template/>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>
```

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: '2016-09-15'
    }
  }
})
```

is 属性可以使用 Mustache 语法，在运行时来决定具体需要渲染哪个模板：

```
<template name="odd">
  <view> odd </view>
</template>
<template name="even">
  <view> even </view>
</template>
```

```
<block wx:for="{{[1, 2, 3, 4, 5]}}">
    <template is="{{item % 2 == 0 ? 'even' : 'odd'}}"/>
</block>
```

模板的作用域

模板拥有自己的作用域，只能使用 data 传入的数据。

WXML-事件

什么是事件

事件是视图层到逻辑层的通讯方式。

事件可以将用户的行为反馈到逻辑层进行处理。

事件可以绑定在组件上，当达到触发事件，就会执行逻辑层中对应的事件处理函数。

事件对象可以携带额外信息，如 id, dataset, touches。

事件的使用方式

在组件中绑定一个事件处理函数。

如 bindtap，当用户点击该组件的时候会在该页面对应的 Page 中找到相应的事件处理函数。

```
<view id="tapTest" data-hi="MINA" bindtap="tapName"> Click me! </view>
```

在相应的 Page 定义中写上相应的事件处理函数，参数是 event。

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

事件详解

事件分类

事件分为冒泡事件和非冒泡事件

冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。

非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

WXML 的冒泡事件列表：

类型　　触发条件

touchstart　手指触摸

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 手指触摸后离开

longtap 手指触摸后，超过 350ms 再离开

注：除上表之外的其他组件自定义事件都是非冒泡事件，如<form/>的 submit 事件，<input/>的 input 事件，<scroll-view/>的 scroll 事件，(详见各个组件)

事件绑定

事件绑定的写法同组件的属性，以 key、value 的形式。

key 以 bind 或 catch 开头，然后跟上事件的类型，如 bindtap, catchtouchstart

value 是一个字符串，需要在对应的 Page 中定义同名的函数。不然当触发事件的时候会报错。

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。

如在下边这个例子中，点击 inner view 会先后触发 handleTap1 和 handleTap2(因为 tap 事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，不再向父节点传递)，点击 middle view 会触发 handleTap2，点击 outter view 会触发 handleTap1。

```
<view id="outter" bindtap="handleTap1">
  outer view
  <view id="middle" catchtap="handleTap2">
    middle view
    <view id="inner" bindtap="handleTap3">
      inner view
    </view>
  </view>
</view>
```

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。

事件对象的属性列表：

属性	类型	说明
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
touches	Array	触摸事件，触摸点信息的数组
detail	Object	额外的信息

WXML-引用

WXML 提供两种文件引用方式 import 和 include。

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：

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

import 的作用域

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 import 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"/>
<template name="B">
  <text> B template </text>
</template>
<!-- C.wxml -->
<import src="b.wxml"/>
<template is="A"/>    <!-- Error! Can not use tempalte when not import A. -->
<template is="B"/>
```

include

include 可以将目标文件出了<template/>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>
```

WXSS

WXSS(WeiXin Style Sheets)是 MINA 设计的一套样式语言，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，我们的 WXSS 具有 CSS 大部分特性。 同时为了更适合开发微信小程序，我们对 CSS 进行了扩充以及修改。

与 css 相比我们扩展的特性有：

尺寸单位

样式导入

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 = 2.34px
iPhone6	1rpx = 0.5px	1px = 2rpx
iPhone6s	rpx = 0.552px	1px = 1.81rpx

rem（root em）：规定屏幕宽度为 20rem；1rem = (750/20)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。

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;
}
```

内联样式

MINA 组件上支持使用 style、class 属性来控制组件的样式。

style：静态的样式统一写到 class 中。style 接收动态的样式，在运行时会进行解析，所以不要将静态的样式写进 style 中，以免影响渲染速度。

```
<view style="color:{{color}};" />
```

class：用于指定样式规则，其属性值是样式规则中类选择器名(样式类名)的集合，样式类名不需要带上.，样式类名之间用空格分隔。

```
<view class="normal_view" />
```

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

全局样式与局部样式

定义在 app.wxss 中的样式为全局样式，作用于每一个页面。在 page 的 wxss 文件中定义的样式为局部样式，只作用在对应的页面，并会覆盖 app.wxss 中相同的选择器。