
黑马程序员™
www.itheima.com
传智播客旗下
高端IT教育品牌
小程序的视图与逻辑

目录 Contents

- 数据绑定与事件绑定
- wxs脚本
- 页面渲染
- 页面事件

1. 数据绑定与事件绑定

1.1 数据绑定

1. 如何定义页面的数据

小程序中每个页面，由4部分组成，其中 .js 文件内可以定义页面的**数据**、**生命周期函数**、**其它业务逻辑**；

如果要在.js文件内定义页面的数据，只需**把数据定义到 data 节点下**即可；

示例代码如下：

```
Page({
  data: {
    info: 'init data', // 字符串类型的数据
    array: [{msg: '1'}, {msg: '2'}] // 数组类型的数据
  }
})
```

1. 数据绑定与事件绑定

1.1 数据绑定

2. Mustache语法格式

把data中的数据绑定到页面中渲染，使用 **Mustache 语法**（双大括号）将变量包起来即可；

语法格式为：

```
<view> {{ 要绑定的数据名称 }} </view>
```

Mustache 语法的主要应用场景：

- 绑定内容
- 绑定属性
- 运算（三元表达式、算术运算、逻辑判断、字符串运算、数据路径运算）

1. 数据绑定与事件绑定

1.1 数据绑定

3. 绑定内容

页面结构：

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

页面数据：

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

1. 数据绑定与事件绑定

1.1 数据绑定

4. 绑定属性（需要在双引号之内）

页面结构：

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

页面数据：

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

1. 数据绑定与事件绑定

1.1 数据绑定

5. 运算（三元运算）

页面结构：

```
<view> {{ flag ? '条件为真' : '条件为假' }} </view>
```

页面数据：

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

1. 数据绑定与事件绑定

1.2 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，当组件触发事件，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。

1. 数据绑定与事件绑定

1.3 bindtap绑定触摸事件

在小程序中，不存在网页中的 onclick 鼠标点击事件，而是通过 **tap 事件**来响应触摸行为；

1. 通过 bindtap，可以为组件绑定触摸事件，语法如下：

```
<view bindtap="tapName"> Click me! <view>
```

2.在相应的Page定义中写上相应的事件处理函数，事件参数是event：

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

1. 数据绑定与事件绑定

1.4 bindinput绑定文本框输入事件

在小程序中，通过 **input 事件**来响应文本框的输入事件；

1. 通过 bindinput，可以为文本框绑定输入事件，语法如下：

```
<input bindinput="inputName"><input>
```

2.在相应的Page定义中写上相应的事件处理函数，事件参数是event：

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

1.5 data和文本框之间的数据同步

1. 监听文本框的数据变化

在文本框的 input 事件处理函数中，通过事件参数 event，能够访问到文本框的最新值：

语法：event.detail.value

示例代码如下：

```
inputName: function (event) {
  // 获取到文本框中最新的内容
  console.log(event.detail.value)
}
```

1.5 data和文本框之间的数据同步

2. 修改data中的数据

通过 **this.setData(dataObject)** 方法，可以给页面中的 data 数据重新赋值。

例如：监听文本框的数据变化，并把最新的值赋值给 data 中的 msg

示例代码如下：

```
inputName: function (event) {
  this.setData({
    msg: event.detail.value // 为 data 中的 msg 重新赋值
  })
}
```

1.6 事件传参

1. 不能在绑定事件的同时传递参数

小程序中的事件传参比较特殊，不能在为组件绑定事件的同时，为事件处理函数传递参数。

例如，下面的代码将不能正常工作：

```
<button type="primary" bindtap='btnHandler(123)'>事件传参</button>
```

因为小程序会把 bindtap 后指定的值，统一当作事件名称来处理；

1.6 事件传参

2. 通过 data-* 自定义属性传参

如果要在组件触发事件处理函数的时候，传递参数，可以为组件提供 data-* 自定义属性传参。

示例代码如下：

```
<button bindtap='btnHandler' data-info="{{123}}">事件传参</button>
```

其中，info 会被当作参数名，数值 123 会被当作参数值。

1.6 事件传参

3. 获取 data-* 自定义属性中传递的参数

通过事件参数 **event.target.dataset.参数名**，能够获取 data-* 自定义属性传递到事件处理函数中的参数。

示例代码如下：

```
btnHandler: function(event){
  console.log(event.target.dataset.info)
}
```

目录 Contents

- 数据绑定与事件绑定
- wxs 脚本
- 页面渲染
- 页面事件

2. wxs 脚本

2.1 wxs 概述

1. 什么是 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。

2. wxs 脚本

2.1 wxs 概述

2. wxs 的注意点

- **没有兼容性：**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
- **与 javascript 不同：**wxs 与 javascript 是不同的语言，有自己的语法，并不和 javascript 一致
- **隔离性：**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
- **不能作为事件回调：**wxs 函数不能作为组件的事件回调
- **iOS设备上比 javascript 运行快：**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

2. wxs 脚本

2.1 wxs 概述

3. wxs 遵循 CommonJS 模块化规范

CommonJS 是 javascript 的模块化规范之一，小程序的脚本语言 wxs 遵循了 CommonJS 规范，因此，使用 wxs 时的体验和使用 node.js 的体验比较相似。

在 wxs 中，可以使用 CommonJS 中规定的如下成员：

- **module 对象**：每个 wxs 都是独立的模块，每个模块均有一个内置的 module 对象，每个模块都有自己独立的作用域。
- **module.exports**：由于 wxs 拥有独立作用域，所以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，通过 module.exports 属性，可以对外共享本模块的**私有变量与函数**。
- **require函数**：在.wxs模块中引用其他 wxs 文件模块，可以使用 require 函数。

2. wxs 脚本

2.2 wxs 基础语法

1. 使用 module.exports 向外共享成员

通过 module.exports 属性，可以对外共享本模块的**私有变量**与**函数**。示例代码如下：

```
var foo = "'hello world' from wxs"; // 定义私有变量 foo
var bar = function (d) {            // 定义私有函数 bar
  return d;
}
module.exports = { // 通过 modules.exports 向外共享私有成员
  FOO: foo, // 向外共享私有变量 foo
  bar: bar, // 向外共享私有函数 bar
};
module.exports.msg = "some msg"; // 额外向 module.exports 中挂载 msg 变量
```

2.2 wxs 基础语法

2. 使用 require 引入其它 wxs 模块

假设有两个 wxs 模块，路径分别为 **/pages/tools.wxs** 和 **/pages/logic.wxs**，如果要在 logic.wxs 中引入 tools.wxs 脚本，则示例代码如下：

```
// 使用 require 导入 tools.wxs 脚本
var tools = require("./tools.wxs");
// 得到的 tools 对象，可以直接访问到 tools.wxs 中向外暴露的变量和方法
```

2.2 wxs 基础语法

3. 使用 require 的注意点

在.wxs模块中引用其他 wxs 文件模块，可以使用 **require** 函数。

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。

2.2 wxs 基础语法

4. 支持的数据类型

WXS 语言目前共有以下8种数据类型：

number 数值类型、**string** 字符串类型、**boolean** 布尔类型、**object** 对象类型、

function 函数类型、**array** 数组类型、 **date** 日期类型、 **regexp** 正则

详细的类型说明文档，请翻阅如下网址：

https://developers.weixin.qq.com/miniprogram/dev/framework/view/wxs/06datatype.html

注意：由于wxs 与 javascript 是不同的语言，有自己的语法，并不和 javascript 一致，所以在使用以上8种数据类型的时候，一定要先翻阅官方文档，再进行使用！

2.3 内嵌 wxs 脚本

wxs 代码可以编写在 wxml 文件中的 **<wxs></wxs>** 标签内，就像 javascript 代码可以编写在 html 文件中的 <script></script> 标签内一样。

wxml 文件中的每个 <wxs></wxs> 标签，**必须提供一个 module 属性**，用来指定当前 <wxs></wxs> 标签的**模块名**。在单个 wxml 文件内，建议其值唯一。

module 属性值的命名必须符合下面两个规则：

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

2. wxs 脚本

2.4 内嵌 wxs 脚本的示例代码

```
<!--wxml-->
<wxs module="foo">
var some_msg = "hello world";
module.exports = {
        msg : some_msg,
}
</wxs>
<view> {{foo.msg}} </view>
```

页面输出：

hello world

上面例子声明了一个名字为 **foo** 的模块，将 some_msg 变量暴露出来，供当前页面使用。

2. wxs 脚本

2.5 外联 wxs 脚本

wxs 代码还可以编写在**以 .wxs 为后缀名的文件内**，就像 javascript 代码可以编写在以 .js 为后缀名的文件中一样。

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。

示例代码如下：

```
// /pages/tools.wxs
var foo = "'hello world' from tools.wxs";
var bar = function (d) {
  return d;
}
module.exports = {
  FOO: foo,
  bar: bar,
};
module.exports.msg = "some msg";
```

2.6 wxml 内引用外联的 wxs 脚本

在 wxml 中如果要引入外联的 wxs 脚本，**必须**为 <wxs></wxs> 标签添加 **module** 和 **src** 属性。

- **module** 用来为 <wxs></wxs> 标签指定模块名，作为当前页面访问这个模块的标识名称；
- **src** 用来指定当前 <wxs></wxs> 标签要引入的脚本路径，**必须是相对路径**；

示例代码如下：

```
<!-- page/index/index.wxml -->
<wxs src="./../tools.wxs" module="tools" />
<view> {{tools.msg}} </view>
<view> {{tools.bar(tools.FOO)}} </view>
```

目录 Contents

- 数据绑定与事件绑定
- wxs 脚本
- 页面渲染
- 页面事件

3. 页面渲染

3.1 条件渲染

1. wx:if

在小程序中，使用 **wx:if="{{condition}}"** 来判断是否需要渲染该代码块：

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

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>
```

3. 页面渲染

3.1 条件渲染

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></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. 页面渲染

3.1 条件渲染

3. hidden

在小程序中，直接使用 **hidden="{{condition}}"** 也能控制元素的显示与隐藏：

```
<view hidden="{{condition}}"> 条件为 true 隐藏，条件为 false 显示 </view>
```

3. 页面渲染

3.1 条件渲染

4. wx:if 与 hidden 的对比

- 被 wx:if 控制的区域，框架有一个**局部渲染**的过程，会根据控制条件的改变，动态创建或销毁对应的UI结构。
- 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**。
- **总结：**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。

3. 页面渲染

3.2 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**。

语法示例如下：

```
<view wx:for="{{array}}">
  索引是：{{index}} 当前项是：{{item}}
</view>
```

3.2 列表渲染

2. 手动指定索引和当前项的变量名

- 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，示例代码如下：

```
<view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
  索引是: {{idx}} 当前项是: {{itemName}}
</view>
```

3.2 列表渲染

3. block wx:for

类似 block wx:if，也可以将 wx:for 用在<block></block>标签上，以渲染一个包含多节点的结构块。

示例代码如下：

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

3.3 列表渲染中的 key

1. key 在列表循环中的作用

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，<checkbox/> 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**。

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。

3.3 列表渲染中的 key

2. key 值的注意点

① key 值必须具有唯一性，且不能动态改变

② key 的值必须是数字或字符串

③ 保留关键字 ***this** 代表在 for 循环中的 **item 本身**，它也可以充当 key 值，但是有以下限制：需要 **item 本身**是一个**唯一的字符串**或者**数字**。

④ 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。

3.3 列表渲染中的 key

2. key 值的两种形式

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**。

注意：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。

目录 Contents

- 数据绑定与事件绑定
- wxs 脚本
- 页面渲染
- 页面事件

4. 页面事件

4.1 下拉刷新

1. 下拉刷新的概念及应用场景

概念：下拉刷新是移动端更新列表数据的交互行为，用户通过手指在屏幕上自上而下的滑动，可以触发页面的下拉刷新，更新列表数据。

应用场景：在移动端，数据列表是常见的页面效果，更新列表数据是最基本的页面需求，相比于按钮刷新、定时刷新来说，**下拉刷新**的用户体验方便友好，已经成为移动端刷新列表数据的**最佳解决方案**。

4. 页面事件

4.1 下拉刷新

2. 启用下拉刷新

两种方式：

① 需要在 app.json 的 window 选项中或页面配置中开启 **enablePullDownRefresh**。但是，一般情况下，推荐在页面配置中为需要的页面单独开启下拉刷新行为。

② 可以通过 **wx.startPullDownRefresh()** 触发下拉刷新，调用后触发下拉刷新动画，效果与用户手动下拉刷新一致。

4. 页面事件

4.1 下拉刷新

3. 配置下拉刷新窗口的样式

需要在 app.json 的 window 选项中或页面配置中修改 backgroundColor 和 backgroundTextStyle 选项。

- backgroundColor 用来配置下拉刷新窗口的背景颜色，仅支持16进制颜色值
- backgroundTextStyle 用来配置下拉刷新 loading 的样式，仅支持 dark 和 light

4. 页面事件

4.1 下拉刷新

4. 监听页面的下拉刷新事件

为页面添加 **onPullDownRefresh()** 函数，可以监听用户在当前页面的下拉刷新事件。

4.1 下拉刷新

5. 停止下拉刷新效果

当处理完下拉刷新后，下拉刷新的 loading 效果会一直显示，**不会主动消失**，所以需要手动隐藏下拉刷新的 loading 效果。此时，调用 **wx.stopPullDownRefresh()** 可以停止当前页面的下拉刷新。

4.2 上拉加载更多

1. 上拉加载更多的概念及应用场景

概念：在移动端，随着手指不断向上滑动，当内容将要到达屏幕底部的时候，页面会随之不断的加载后续内容，直到没有新内容为止，我们称之为**上拉加载更多**。上拉加载更多的**本质**就是**数据的分页加载**。

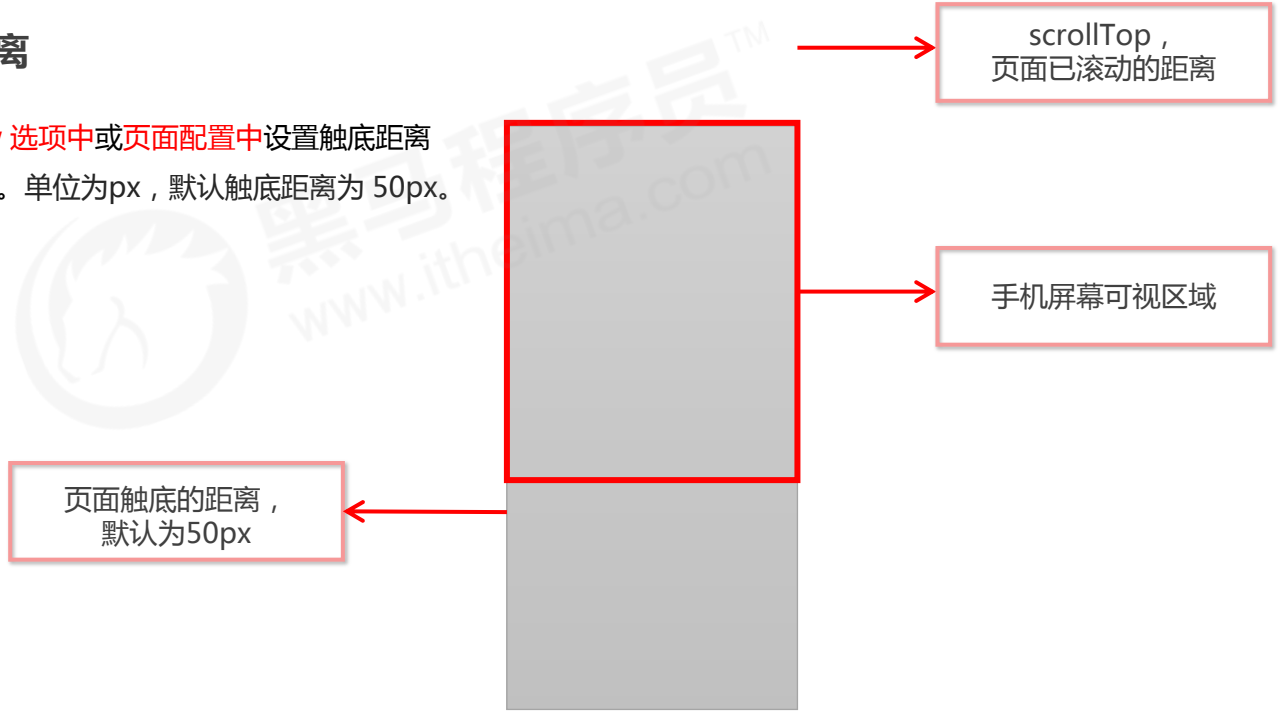
应用场景：在移动端，列表数据的分页加载，首选的实现方式就是上拉加载更多。

4. 页面事件

4.2 上拉加载更多

2. 配置上拉触底的距离

可以在 app.json 的 window 选项中或页面配置中设置触底距离 **onReachBottomDistance**。单位为px，默认触底距离为 50px。



4.2 上拉加载更多

3. 监听页面的上拉触底事件

为页面添加 **onReachBottom()** 函数，可以监听用户在当前页面的上拉触底事件，从而实现上拉加载更多列表数据的效果。

4.3 其它页面事件

1. onPageScroll(Object)

监听用户滑动页面事件。其中 Object 参数说明如下：

属性	类型	说明
scrollTop	Number	页面在垂直方向已滚动的距离（单位px）

4.3 其它页面事件

2. onShareAppMessage(Object)

监听用户点击页面内转发按钮（<button> 组件 open-type="share"）或右上角菜单“转发”按钮的行为，并自定义转发内容。其中 Object 参数说明如下：

参数	类型	说明
from	String	转发事件来源。button：页面内转发按钮；menu：右上角转发菜单
target	Object	如果 from 值是 button，则 target 是触发这次转发事件的 button，否则为 undefined
webViewUrl	String	页面中包含<web-view>组件时，返回当前<web-view>的url

4.3 其它页面事件

2. onShareAppMessage(Object)

同时，此转发事件需要 return 一个 Object，用于**自定义转发内容**，返回内容如下：

字段	说明	默认值
title	转发标题	当前小程序名称
path	转发路径	当前页面 path ，必须是以 / 开头的完整路径
imageUrl	自定义图片路径，可以是本地文件路径、代码包文件路径或者网络图片路径。支持PNG及JPG。显示图片长宽比是 5:4。	使用默认截图

4.3 其它页面事件

2. onShareAppMessage(Object)

转发功能的示例代码如下：

```
Page({
    onShareAppMessage: function (res) {
        if (res.from === 'button') {
            // 来自页面内转发按钮
            console.log(res.target)
        }
        return {
            title: '自定义转发标题',
            path: '/page/user?id=123'
        }
    }
})
```

4.3 其它页面事件

3. onTabItemTap(Object)

点击 tab 时触发，其中 Object 参数说明如下：

参数	类型	说明
index	String	被点击 tabItem 的序号，从0开始
pagePath	String	被点击 tabItem 的页面路径
text	String	被点击 tabItem 的按钮文字

4. 页面事件

4.3 其它页面事件

3. onTabItemTap(Object)

示例代码如下：

```
onTabItemTap(item) {
    console.log(item.index)
    console.log(item.pagePath)
    console.log(item.text)
}
```

传智播客旗下高端IT教育品牌