

# 小程序 - 视图与逻辑



黑马程序员™  
www.itheima.com

传智播客旗下  
高端IT教育品牌



# 目录 Contents

◆ 页面导航

◆ 页面事件

◆ 生命周期

◆ WXS 脚本

◆ 案例 - 本地生活（列表页面）

## 1. 什么是页面导航

页面导航指的是页面之间的相互跳转。例如，浏览器中实现页面导航的方式有如下两种：

- ① <a> 链接
- ② location.href



## 2. 小程序中实现页面导航的两种方式

### ① 声明式导航

- 在页面上声明一个 `<navigator>` 导航组件
- 通过点击 `<navigator>` 组件实现页面跳转

### ② 程式化导航

- 调用小程序的导航 API，实现页面的跳转

## 1. 导航到 tabBar 页面

tabBar 页面指的是被配置为 tabBar 的页面。

在使用 <navigator> 组件跳转到指定的 tabBar 页面时，需要指定 url 属性和 open-type 属性，其中：

- url 表示要跳转的页面的地址，必须以 / 开头
- open-type 表示跳转的方式，必须为 switchTab

示例代码如下：

```
1 <navigator url="/pages/message/message" open-type="switchTab">导航到消息页面</navigator>
```



### 2. 导航到非 tabBar 页面

非 tabBar 页面指的是没有被配置为 tabBar 的页面。

在使用 <navigator> 组件跳转到普通的非 tabBar 页面时，则需要指定 url 属性和 open-type 属性，其中：

- url 表示要跳转的页面的地址，必须以 / 开头
- open-type 表示跳转的方式，必须为 navigate

示例代码如下：

```
1 <navigator url="/pages/info/info" open-type="navigate">导航到info页面</navigator>
```

注意：为了简便，在导航到非 tabBar 页面时，open-type="navigate" 属性可以省略。



### 3. 后退导航

如果要后退到上一页面或多级页面，则需要指定 `open-type` 属性和 `delta` 属性，其中：

- `open-type` 的值必须是 `navigateBack`，表示要进行后退导航
- `delta` 的值必须是数字，表示要后退的层级

示例代码如下：

```
1 <navigator open-type='navigateBack' delta='1'>返回上一页</navigator>
```

注意：为了简便，如果只是后退到上一页面，则可以省略 `delta` 属性，因为其默认值就是 1。

## 1. 导航到 tabBar 页面

调用 `wx.switchTab(Object object)` 方法，可以跳转到 tabBar 页面。其中 Object 参数对象的属性列表如下：

属性	类型	是否必选	说明
url	string	是	需要跳转的 tabBar 页面的路径，路径后不能带参数
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）





## 1. 导航到 tabBar 页面

示例代码如下：

```
1 // 页面结构
2 <button bindtap="gotoMessage">跳转到消息页面</button>
3
4 // 通过编程式导航，跳转到 message 页面
5 gotoMessage() {
6   wx.switchTab({
7     url: '/pages/message/message'
8   })
9 }
```

## 2. 导航到非 tabBar 页面

调用 `wx.navigateTo(Object object)` 方法，可以跳转到非 tabBar 的页面。其中 Object 参数对象的属性列表如下：

属性	类型	是否必选	说明
url	string	是	需要跳转到的非 tabBar 页面的路径，路径后可以带参数
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）



### 2. 导航到非 tabBar 页面

示例代码如下：

```
1 // 页面结构
2 <button bindtap="gotoInfo">跳转到info页面</button>
3
4 // 通过编程式导航，跳转到 info 页面
5 gotoInfo() {
6   wx.navigateTo({
7     url: '/pages/info/info'
8   })
9 }
```

## 3. 后退导航

调用 `wx.navigateBack(Object object)` 方法，可以返回上一页面或多级页面。其中 Object 参数对象可选的属性列表如下：

属性	类型	默认值	是否必选	说明
delta	number	1	否	返回的页面数，如果 delta 大于现有页面数，则返回到首页
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）



### 3. 后退导航

示例代码如下：

```
1 // 页面结构
2 <button bindtap="gotoBack">后退</button>
3
4 // 编程式导航，后退到上一页面
5 gotoBack() {
6   wx.navigateBack()
7 }
```



## 1. 声明式导航传参

navigator 组件的 url 属性用来指定将要跳转到的页面的路径。同时，路径的后面还可以携带参数：

- 参数与路径之间使用 ? 分隔
- 参数键与参数值用 = 相连
- 不同参数用 & 分隔

代码示例如下：

```
1 <navigator url="/pages/info/info?name=zs&age=20">跳转到info页面</navigator>
```



### 2. 编程式导航传参

调用 `wx.navigateTo(Object object)` 方法跳转页面时，也可以携带参数，代码示例如下：

```
1 // 页面结构
2 <button bindtap="gotoInfo2">跳转到info页面</button>
3
4 // 通过编程式导航，跳转到 info 页面，并携带参数
5 gotoInfo2() {
6   wx.navigateTo({
7     url: '/pages/info/info?name=ls&gender=男'
8   })
9 }
```



### 3. 在 onLoad 中接收导航参数

通过声明式导航传参或程式化导航传参所携带的参数，可以直接在 **onLoad 事件** 中直接获取到，示例代码如下：

```
1 /**
2  * 生命周期函数--监听页面加载
3  */
4 onLoad: function(options) {
5     // options 就是导航传递过来的参数对象
6     console.log(options)
7 }
```



# 目录 Contents

◆ 页面导航

◆ 页面事件

◆ 生命周期

◆ WXS 脚本

◆ 案例 - 本地生活（列表页面）

## 1. 什么是下拉刷新

**下拉刷新**是移动端的专有名词，指的是通过手指在屏幕上的下拉滑动操作，从而**重新加载页面数据**的行为。



## 2. 启用下拉刷新

启用下拉刷新有两种方式：

### ① 全局开启下拉刷新

- 在 app.json 的 window 节点中，将 enablePullDownRefresh 设置为 true

### ② 局部开启下拉刷新

- 在页面的 .json 配置文件中，将 enablePullDownRefresh 设置为 true

在实际开发中，推荐使用第 2 种方式，为需要的页面单独开启下拉刷新的效果。

### 3. 配置下拉刷新窗口的样式

在全局或页面的 .json 配置文件中，通过 `backgroundColor` 和 `backgroundTextStyle` 来配置下拉刷新窗口的样式，其中：

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



### 4. 监听页面的下拉刷新事件

在页面的 .js 文件中，通过 `onPullDownRefresh()` 函数即可监听当前页面的下拉刷新事件。

例如，在页面的 wxml 中有如下的 UI 结构，点击按钮可以让 count 值自增 +1：

```
1 // 页面结构
2 <view>count值为: {{count}}</view>
3 <button bindtap="countAdd">+1</button>
4
5 // +1 按钮的点击事件处理函数
6 countAdd() {
7   this.setData({
8     count: this.data.count + 1
9   })
10 }
```



### 4. 监听页面的下拉刷新事件

在触发页面的下拉刷新事件的时候，如果要把 count 的值重置为 0，示例代码如下：

```
1 /**
2  * 页面相关事件处理函数--监听用户下拉动作
3  */
4 onPullDownRefresh: function () {
5   this.setData({
6     count: 0
7   })
8 }
```



### 5. 停止下拉刷新的效果

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。示例代码如下：

```
1 /**
2  * 页面相关事件处理函数--监听用户下拉动作
3  */
4 onPullDownRefresh: function () {
5   this.setData({
6     count: 0
7   })
8   // 当数据重置成功之后，调用此函数，关闭下拉刷新的效果
9   wx.stopPullDownRefresh()
10 }
```

## 1. 什么是上拉触底

**上拉触底**是移动端的专有名词，通过手指在屏幕上的上拉滑动操作，从而**加载更多数据**的行为。







## 2. 监听页面的上拉触底事件

在页面的 .js 文件中，通过 `onReachBottom()` 函数即可监听当前页面的上拉触底事件。示例代码如下：

```
1 /**
2  * 页面上拉触底事件的处理函数
3  */
4 onReachBottom: function () {
5   console.log('触发了上拉触底的事件')
6 }
```



### 3. 配置上拉触底距离

上拉触底距离指的是触发上拉触底事件时，滚动条距离页面底部的距离。

可以在全局或页面的 .json 配置文件中，通过 `onReachBottomDistance` 属性来配置上拉触底的距离。

小程序默认的触底距离是 50px，在实际开发中，可以根据自己的需求修改这个默认值。



黑马程序员  
www.itheima.com

## 1. 案例效果展示





## 2. 案例的实现步骤

- ① 定义获取随机颜色的方法
- ② 在页面加载时获取初始数据
- ③ 渲染 UI 结构并美化页面效果
- ④ 在上拉触底时调用获取随机颜色的方法
- ⑤ 添加 loading 提示效果
- ⑥ 对上拉触底进行节流处理



黑马程序员™  
www.itheima.com



### 3. 步骤1 - 定义获取随机颜色的方法

```
1 data: {  
2     colorList: [] // 随机颜色的列表  
3 },  
4  
5 getColors() { // 获取随机颜色的方法  
6     wx.request({ // 发起请求, 获取随机颜色值的数组  
7         url: 'https://www.escook.cn/api/color',  
8         method: 'GET',  
9         success: ({ data: res }) => {  
10             this.setData({  
11                 colorList: [...this.data.colorList, ...res.data]  
12             })  
13         }  
14     })  
15 }
```

### 3. 步骤2 - 在页面加载时获取初始数据

```
1  /**
2   * 生命周期函数--监听页面加载
3   */
4  onLoad: function (options) {
5      this.getColors()
6  },
```



### 3. 步骤3 - 渲染 UI 结构并美化页面效果

```
1 // wxml 的结构
2 <view wx:for="{{colorList}}" wx:key="index" class="num-item" style="background-color:
  rgba({{item}});">{{item}}</view>
3
4 // wxss 样式
5 .num-item {
6   border: 1rpx solid #efefef;
7   border-radius: 8rpx;
8   line-height: 200rpx;
9   margin: 15rpx;
10  text-align: center;
11  text-shadow: 0rpx 0rpx 5rpx #fff;
12  box-shadow: 1rpx 1rpx 6rpx #aaa;
13 }
```

### 3. 步骤4 - 上拉触底时获取随机颜色

```
1  /**
2   * 页面上拉触底事件的处理函数
3   */
4  onReachBottom: function () {
5      // 调用获取随机颜色的方法
6      this.getColors()
7  },
```





### 3. 步骤5 - 添加 loading 提示效果

```
1 getColors() {  
2   wx.showLoading({ title: '数据加载中...' }) // 1. 展示 loading 效果  
3  
4   // 发起请求，获取随机颜色值的数组  
5   wx.request({  
6     // 省略其它代码...  
7     complete: () => {  
8       wx.hideLoading() // 2. 隐藏 loading 效果  
9     }  
10  })  
11 },
```

### 3. 步骤6 - 对上拉触底进行节流处理

#### ① 在 data 中**定义** isloading 节流阀

- false 表示当前没有进行任何数据请求
- true 表示当前正在进行数据请求

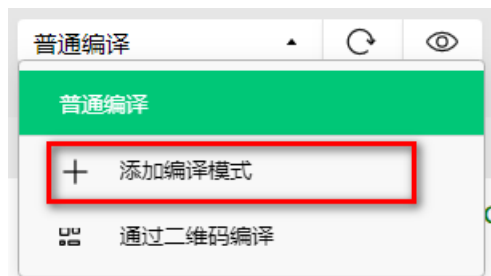
#### ② 在 getColors() 方法中**修改** isloading 节流阀的值

- 在刚调用 getColors 时将节流阀设置 true
- 在网络请求的 complete 回调函数中，将节流阀重置为 false

#### ③ 在 onReachBottom 中**判断**节流阀的值，从而对数据请求进行节流控制

- 如果节流阀的值为 true，则阻止当前请求
- 如果节流阀的值为 false，则发起数据请求

## 自定义编译模式



### 自定义编译条件

解析二维码

仅支持上传png、jpg文件，通过解析二维码可自动生成启动页面和启动参数。

模式名称

启动页面  ▼

启动参数

进入场景  ▼

编译设置 ☐ 下次编译时模拟更新 (需 1.9.90 及以上基础库版本)

# 目录 Contents

- ◆ 页面导航
- ◆ 页面事件
- ◆ 生命周期
- ◆ WXS 脚本
- ◆ 案例 - 本地生活（列表页面）

## 1. 什么是生命周期

生命周期（Life Cycle）是指一个对象从创建 -> 运行 -> 销毁的整个阶段，强调的是时间段。例如：

- 张三出生，表示这个人生命周期的开始
- 张三离世，表示这个人生命周期的结束
- 中间张三的一生，就是张三的生命周期

我们可以把每个小程序运行的过程，也概括为生命周期：

- 小程序的启动，表示生命周期的开始
- 小程序的关闭，表示生命周期的结束
- 中间小程序运行的过程，就是小程序的生命周期

## 2. 生命周期的分类

在小程序中，生命周期分为两类，分别是：

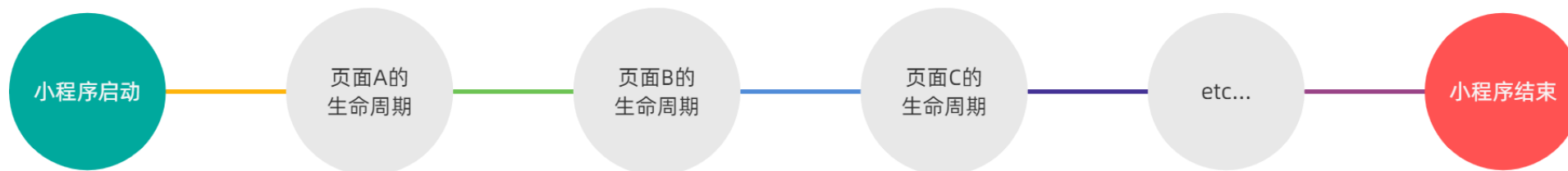
### ① 应用生命周期

- 特指小程序从启动 -> 运行 -> 销毁的过程

### ② 页面生命周期

- 特指小程序中，每个页面的加载 -> 渲染 -> 销毁的过程

其中，页面的生命周期范围较小，应用程序的生命周期范围较大，如图所示：



## 3. 什么是生命周期函数

**生命周期函数**：是由小程序框架提供的**内置函数**，会伴随着生命周期，**自动按次序执行**。

**生命周期函数的作用**：允许程序员在**特定的时间点**，**执行某些特定的操作**。例如，页面刚加载的时候，可以在onLoad 生命周期函数中初始化页面的数据。

注意：**生命周期**强调的是**时间段**，**生命周期函数**强调的是**时间点**。

## 4. 生命周期函数的分类

小程序中的生命周期函数分为两类，分别是：

### ① 应用的生命周期函数

- 特指小程序从启动 -> 运行 -> 销毁期间依次调用的那些函数

### ② 页面的生命周期函数

- 特指小程序中，每个页面从加载 -> 渲染 -> 销毁期间依次调用的那些函数



## 5. 应用的生命周期函数

小程序的**应用生命周期函数**需要在 `app.js` 中进行声明，示例代码如下：

```
1 // app.js 文件
2 App({
3   // 小程序初始化完成时，执行此函数，全局只触发一次。可以做一些初始化的工作。
4   onLaunch: function(options) { },
5   // 小程序启动，或从后台进入前台显示时触发。
6   onShow : function(options) { },
7   // 小程序从前台进入后台时触发。
8   onHide : function() { }
9 })
```

## 6. 页面的生命周期函数

小程序的页面生命周期函数需要在页面的 `.js` 文件中进行声明，示例代码如下：

```
1 // 页面的 .js 文件
2 Page({
3   onLoad : function(options) { }, // 监听页面加载，一个页面只调用1次
4   onShow : function() { },       // 监听页面显示
5   onReady : function() { },      // 监听页面初次渲染完成，一个页面只调用1次
6   onHide : function() { },       // 监听页面隐藏
7   onUnload: function() { }       // 监听页面卸载，一个页面只调用1次
8 })
```

# 目录 Contents

- ◆ 页面导航
- ◆ 页面事件
- ◆ 生命周期
- ◆ WXS 脚本
- ◆ 案例 - 本地生活（列表页面）

## 1. 什么是 wxs

WXS (WeiXin Script) 是小程序独有的一套脚本语言，结合 WXML，可以构建出页面的结构。



## 2. wxs 的应用场景

wxml 中无法调用在页面的 .js 中定义的函数，但是，wxml 中可以调用 wxs 中定义的函数。因此，小程序中 wxs 的典型应用场景就是“过滤器”。



## 3. wxs 和 JavaScript 的关系\*

虽然 wxs 的语法类似于 JavaScript，但是 wxs 和 JavaScript 是完全不同的两种语言：

### ① wxs 有自己的数据类型

- `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

### ② wxs 不支持类似于 ES6 及以上的语法形式

- **不支持**：let、const、解构赋值、展开运算符、箭头函数、对象属性简写、etc...
- **支持**：var 定义变量、普通 function 函数等类似于 ES5 的语法

### ③ wxs 遵循 CommonJS 规范

- `module` 对象
- `require()` 函数
- `module.exports` 对象

## 1. 内嵌 wxs 脚本

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

wxml 文件中的每个 `<wxs></wxs>` 标签，**必须提供 module 属性**，用来指定当前 wxs 的模块名称，方便在 wxml 中访问模块中的成员：

```
1 <view>{{m1.toUpper(username)}}</view>
2
3 <wxs module="m1">
4   // 将文本转为大写形式 zs -> ZS
5   module.exports.toUpper = function(str) {
6     return str.toUpperCase()
7   }
8 </wxs>
```

## 2. 定义外联的 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。示例代码如下：

```
1 // tools.wxs 文件
2 function toLower(str) {
3     return str.toLowerCase()
4 }
5
6 module.exports = {
7     toLower: toLower
8 }
```





## 3. 使用外联的 wxs 脚本

在 wxml 中引入外联的 wxs 脚本时，**必须**为 <wxs> 标签添加 **module** 和 **src** 属性，其中：

- **module** 用来指定模块的名称
- **src** 用来指定要引入的脚本的路径，且**必须是相对路径**

示例代码如下：

```
1 <!-- 调用 m2 模块中的方法 -->
2 <view>{{m2.toLower(country)}}</view>
3
4 <!-- 引用外联的 tools.wxs 脚本，并命名为 m2 -->
5 <wxs src="../../utils/tools.wxs" module="m2"></wxs>
```

## 1. 与 JavaScript 不同

为了降低 wxs (WeiXin Script) 的学习成本, wxs 语言在设计时借大量鉴了 JavaScript 的语法。但是本质上, wxs 和 JavaScript 是完全不同的两种语言!





# WXS 脚本 - WXS 的特点

## 2. 不能作为组件的事件回调

wxs 典型的应用场景就是“过滤器”，经常配合 Mustache 语法进行使用，例如：

```
1 <view>{{m2.toLower(country)}}</view>
```

但是，在 wxs 中定义的函数不能作为组件的事件回调函数。例如，下面的用法是错误的：

```
1 <button bindtap="m2.toLower">按钮</button>
```

## 3. 隔离性

**隔离性**指的是 wxs 的运行环境和其他 JavaScript 代码是隔离的。体现在如下两方面：

- ① wxs 不能调用 js 中定义的函数
- ② wxs 不能调用小程序提供的 API



## 4. 性能好

- 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，二者的运行效率无差异



黑马程序员™  
www.itheima.com

# 目录 Contents

- ◆ 页面导航
- ◆ 页面事件
- ◆ 生命周期
- ◆ WXS 脚本
- ◆ 案例 - 本地生活（列表页面）



# 案例 - 本地生活（列表页面）

## 1. 演示页面效果以及主要功能



- 页面导航并传参
- 上拉触底时加载下一页数据
- 下拉刷新列表数据



黑马程序员  
www.itheima.com



## 案例 - 本地生活（列表页面）

### 2. 列表页面的 API 接口

以**分页**的形式，加载**指定分类下**商铺列表的数据：

#### ① 接口地址

- [https://www.escook.cn/categories/:cate\\_id/shops](https://www.escook.cn/categories/:cate_id/shops)
- URL 地址中的 :cate\_id 是动态参数，表示分类的Id

#### ② 请求方式

- GET 请求

#### ③ 请求参数

- **\_page** 表示请求第几页的数据
- **\_limit** 表示每页请求几条数据





## 案例 - 本地生活（列表页面）

### 3. 判断是否还有下一页数据

如果下面的公式成立，则证明没有下一页数据了：

页码值 \* 每页显示多少条数据  $\geq$  总数据条数

$page * pageSize \geq total$

案例1：总共有 77 条数据，如果每页显示 10 条数据，则总共分为 8 页，其中第 8 页只有 7 条数据

$page(7) * pageSize(10) \geq total(77)$

$page(8) * pageSize(10) \geq total(77)$

案例2：总共有 80 条数据，如果每页显示 10 条数据，则总共分为 8 页，其中第 8 页面有 10 条数据

$page(7) * pageSize(10) \geq total(80)$

$page(8) * pageSize(10) \geq total(80)$

# 总结

- ① 能够知道如何实现页面之间的导航跳转
  - 声明式导航、编程式导航
- ② 能够知道如何实现下拉刷新效果
  - enablePullDownRefresh、onPullDownRefresh
- ③ 能够知道如何实现上拉加载更多效果
  - onReachBottomDistance、onReachBottom
- ④ 能够知道小程序中常用的生命周期函数
  - 应用生命周期函数：onLaunch, onShow, onHide
  - 页面生命周期函数：onLoad, onShow, onReady, onHide, onUnload



黑马程序员

[www.itheima.com](http://www.itheima.com)

传智播客旗下高端IT教育品牌