# 小程序

一、小程序简介

1、小程序与普通网页开发的区别

（1）运行环境不同

**网页**运行在浏览器环境中

**小程序**运行在微信环境中

（2）API 不同

由于运行环境的不同，所以小程序中，无法调用 DOM 和 BOM 的 API。

但是，小程序中可以调用微信环境提供的各种 API，例如：

⚫ 地理定位

⚫ 扫码

⚫ 支付

（3）开发模式不同

网页的开发模式：浏览器 + 代码编辑器

小程序有自己的一套标准开发模式：

⚫ 申请小程序开发账号

⚫ 安装小程序开发者工具

⚫ 创建和配置小程序项目

二、注册小程序开发帐号

https://mp.weixin.qq.com/

注意：个人类型包括：由自然人注册和运营的公众帐号。

帐号能力：个人类型暂不支持微信认证、微信支付及高级接口能力。

1、获取小程序的 AppID



2、安装开发者工具

微信开发者工具是官方推荐使用的小程序开发工具，它提供的主要功能如下：

① 快速创建小程序项目

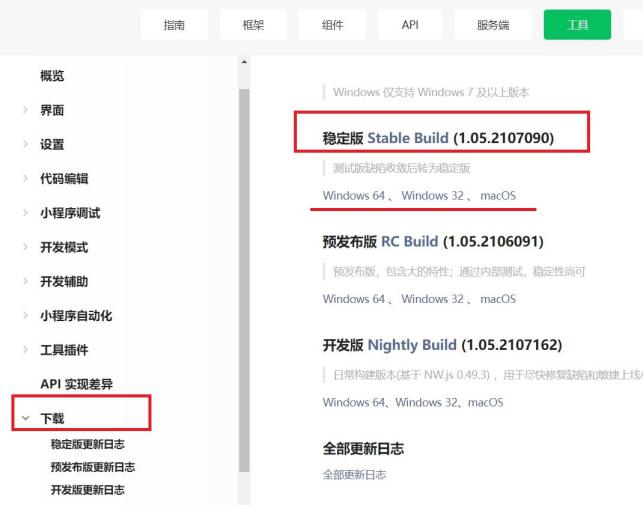
② 代码的查看和编辑

③ 对小程序功能进行调试

④ 小程序的预览和发布

（1）推荐下载和安装最新的稳定版（ Stable Build）的微信开发者工具，下载页面的链接如下：

https://developers.weixin.qq.com/miniprogram/dev/devtools/devtools.html



（2）外观和代理

一般将代理关闭了，本机上的一些代理软件可能会影响它正常使用。

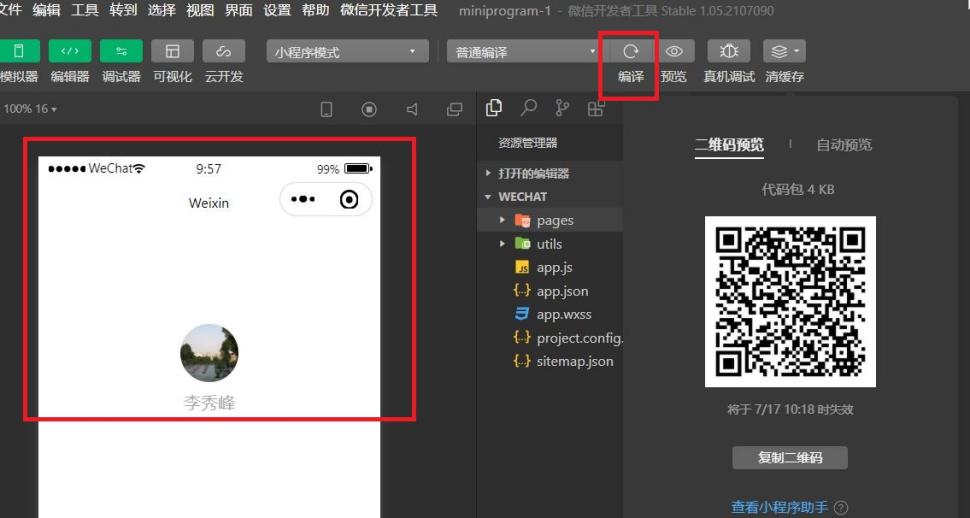


（3）创建小程序

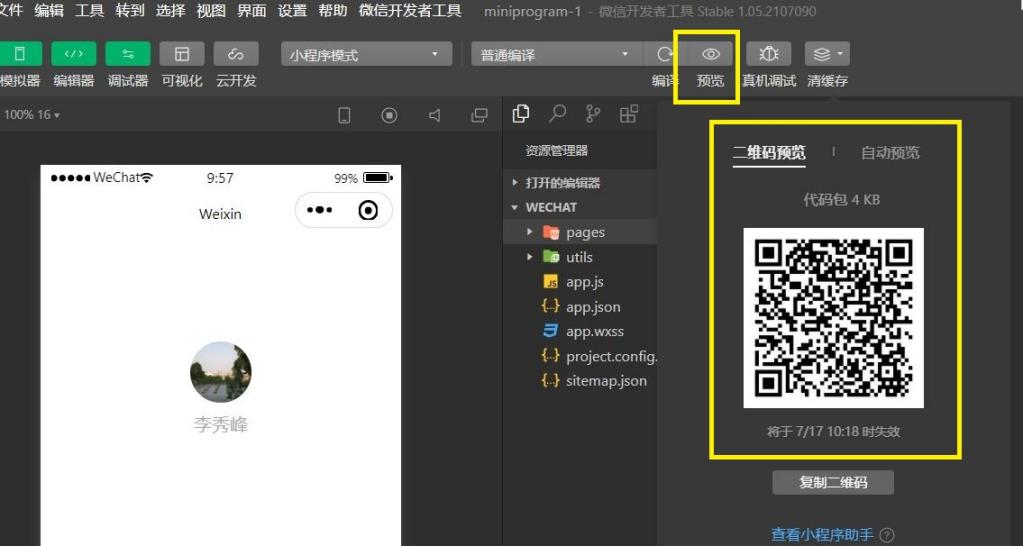


后端服务：这两种开发模式是完全不同的

（4）在模拟器上查看项目效果



1. 在真机上预览项目效果



模拟器的机型建议选择iPhone6/7/8，比例修改成 85%，能够看到全部

三、项目的基本代码组成结构

1、了解项目的基本组成结构

① pages 用来存放所有小程序的页面

② utils 用来存放工具性质的模块（例如：格式化时间的自定义模块）

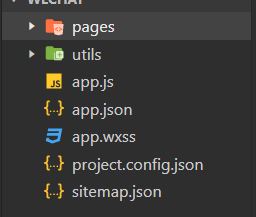
③ app.js 小程序项目的入口文件

④ app.json 小程序项目的全局配置文件

⑤ app.wxss 小程序项目的全局样式文件

⑥ project.config.json 项目的配置文件

⑦ sitemap.json 用来配置小程序及其页面是否允许被微信索引



2、小程序页面的组成部分

小程序官方建议把所有小程序的页面，都存放在 pages 目录中，以单独的文件夹存在，如图所示：

其中，每个页面由 4 个基本文件组成，它们分别是：

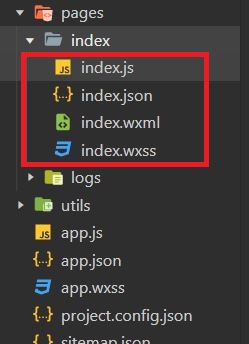
① .js 文件（页面的脚本文件，存放页面的数据、事件处理函数等）

② .json 文件（当前页面的配置文件，配置窗口的外观、表现等）

③ .wxml 文件（页面的模板结构文件）

④ .wxss 文件（当前页面的样式表文件）

最后这四个文件会被编译成为一个文件。



3、JSON 配置文件的作用

JSON 是一种数据格式，在实际开发中， JSON 总是以配置文件的形式出现。小程序项目中也不例外：通过不同的 .json 配置文件，可以对小程序项目进行不同级别的配置。

小程序项目中有 4 种 json 配置文件，分别是：

① 项目根目录中的 app.json 配置文件

② 项目根目录中的 project.config.json 配置文件

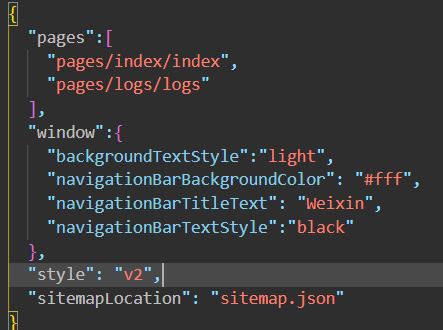
③ 项目根目录中的 sitemap.json 配置文件

④ 每个页面文件夹中的 .json 配置文件

（1）app.json 文件

app.json 是当前小程序的**全局配置**，包括了小程序的所有页面路径、 窗口外观、界面表现、 底部 tab 等。

Demo 项目里边的 app.json 配置内容如下：



简单了解下这 4 个配置项的作用：

① pages：用来记录当前小程序所有页面的路径

② window：全局定义小程序所有页面的背景色、文字颜色等

③ style：全局定义小程序组件所使用的样式版本

④ sitemapLocation：用来指明 sitemap.json 的位置

“style”: ”v2”代表的是使用最新版的样式表文件。如果想要使用旧版本，把这语句删除就好

（2）project.config.json 文件

project.config.json 是项目配置文件，用来记录我们对小程序开发工具所做的个性化配置（在详情中 --- 本地设置 --- 的配置，也会同步更新进去），例如：

⚫ setting 中保存了编译相关的配置

⚫ projectname 中保存的是项目名称

⚫ appid 中保存的是小程序的账号 ID

拿到别人的程序时候要修改其中的appid。

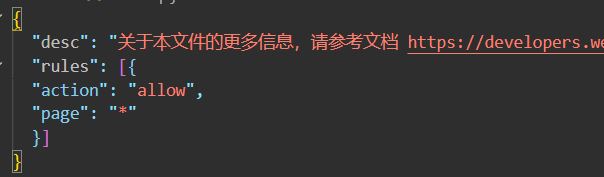
Projectname不等于小程序的名称。

Setting节点中的值，会随着“详情”--- 本地设置的修改，而更改。

（3）sitemap.json 文件

微信现已开放小程序内搜索，效果类似于 PC 网页的 SEO。 sitemap.json 文件用来配置小程序页面是否允许微信索引 。

当开发者允许微信索引时，微信会通过爬虫的形式，为小程序的页面内容建立索引。当用户的搜索关键字和页面的索引匹配成功的时候，小程序的页面将可能展示在搜索结果中。



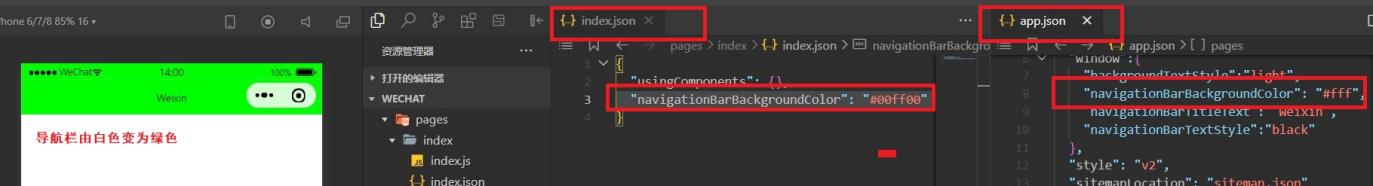
rules索引规则，\*表示所有页面，allow，disallow（不允许被索引）

1

注意： sitemap 的索引提示是默认开启的，如需要关闭 sitemap 的索引提示，可在小程序项目配置文件project.config.json 的 setting 中配置字段 checkSiteMap 为 false。

（4）页面的 .json 配置文件

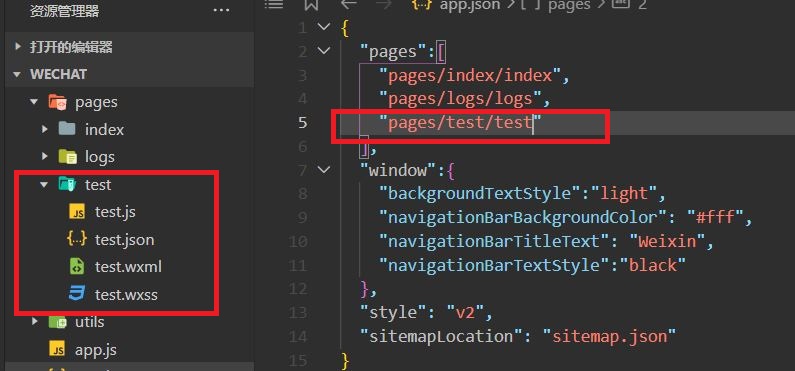
小程序中的每一个页面，可以使用 .json 文件来对本页面的窗口外观进行配置，页面中的配置项会覆盖app.json 的 window 中相同的配置项。例如：



4、新建小程序页面

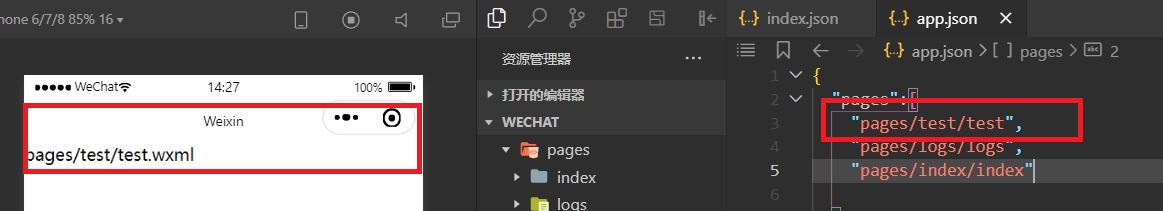
只需要在 app.json -> pages 中新增页面的存放路径，小程序开发者工具即可帮我们自动创建对应的页面文件，要保存后才有。

如图所示：



5、修改项目首页

只需要调整 app.json -> pages 数组中页面路径的前后顺序，即可修改项目的首页。小程序会把排在第一位的页面，当作项目首页进行渲染，如图所示：



6、WXML 模板

（1）什么是 WXML

WXML（ **WeiXin Markup Language**）是小程序框架设计的一套**标签语言**， 专门用来构建小程序页面的结构，其作用类似于网页开发中的 HTML。

（2） WXML 和 HTML 的区别

① 标签名称不同

⚫ HTML （ div, span, img, a）

⚫ WXML（ view, text, image, navigator）

② 属性节点不同

⚫ <a href="#">超链接</a>

⚫ <navigator url="/pages/home/home"></navigator>

③ 提供了类似于 Vue 中的模板语法

⚫ 数据绑定

⚫ 列表渲染

⚫ 条件渲染

7、WXSS 样式

（1）什么是 WXSS

WXSS (WeiXin **Style Sheets**)是一套样式语言，用于描述 WXML 的组件样式，类似于网页开发中的 CSS。

（2）WXSS 和 CSS 的区别

① 新增了 rpx 尺寸单位

⚫ CSS 中需要手动进行像素单位换算，例如 rem

⚫ WXSS 在底层支持新的尺寸单位 rpx，在不同大小的屏幕上小程序会自动进行换算

② 提供了全局的样式和局部样式

⚫ 项目根目录中的 app.wxss 会作用于所有小程序页面

⚫ 局部页面的 .wxss 样式仅对当前页面生效

③ WXSS 仅支持部分 CSS 选择器

⚫ .class 和 #id

⚫ element

⚫ 并集选择器、后代选择器

⚫ ::after 和 ::before 等伪类选择器

不要使用生僻的选择器，可能不支持，使用常见的即可

8、JS 逻辑交互

（1）小程序中的 .js 文件

一个项目仅仅提供界面展示是不够的，在小程序中，我们通过 .js 文件来处理用户的操作。例如：响应用户的点击、获取用户的位置等等。

（2）小程序中 .js 文件的分类

小程序中的 JS 文件分为三大类，分别是：

① app.js

⚫ 是整个小程序项目的入口文件，通过调用 App() 函数来启动整个小程序

② 页面的 .js 文件

⚫ 是页面的入口文件，通过调用 Page() 函数来创建并运行页面

③ 普通的 .js 文件（如utils.js中封装时间格式化的函数，如果需要使用导进去就行）

⚫ 是普通的功能模块文件，用来封装公共的函数或属性供页面使用

四、小程序的宿主环境（组件、API）

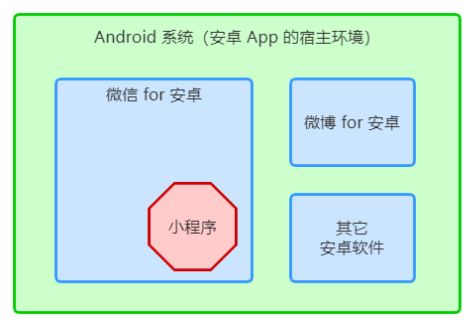
1、什么是宿主环境

宿主环境（ host environment ）指的是**程序运行所必须的依赖环境**。例如：

Android 系统和 iOS 系统是两个不同的宿主环境。安卓版的微信 App 是不能在 iOS 环境下运行的，所以，Android 是安卓软件的宿主环境， 脱离了宿主环境的软件是没有任何意义的！

2、小程序的宿主环境

**手机微信**是小程序的宿主环境，如图所示：



小程序借助**宿主环境提供的能力** ，可以完成许多普通网页无法完成的功能，例如：

微信扫码、微信支付、微信登录、地理定位、 etc…

（1）小程序宿主环境包含的内容

① 通信模型

② 运行机制

③ 组件

④ API

3、通信模型

（1）通信的主体

小程序中通信的主体是**渲染层**和**逻辑层**，其中：

① WXML 模板和 WXSS 样式工作在渲染层

② JS 脚本工作在逻辑层

（2）小程序的通信模型

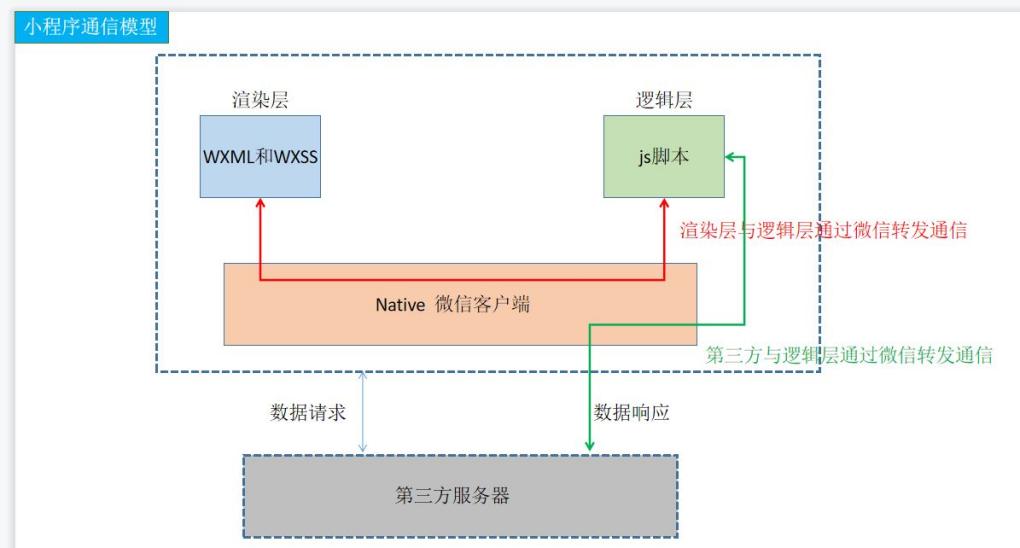
小程序中的通信模型分为两部分：

① **渲染层**和**逻辑层**之间的通信

⚫ 由微信客户端进行转发

② **逻辑层**和**第三方服务器**之间的通信

⚫ 由微信客户端进行转发



4、运行机制

（1）小程序启动的过程

① 把小程序的代码包下载到本地

② 解析 app.json 全局配置文件

③ 执行 app.js 小程序入口文件， 调用 **App() 创建小程序实例**

④ 渲染小程序首页

⑤ 小程序启动完成

（2）页面渲染的过程

① 加载解析页面的 .json 配置文件

② 加载页面的 .wxml 模板和 .wxss 样式

③ 执行页面的 .js 文件， 调用 **Page() 创建页面实例**

④ 页面渲染完成

5、组件

（1）小程序中组件的分类

小程序中的组件也是由宿主环境提供的，开发者可以基于组件快速搭建出漂亮的页面结构。官方把小程序的组件分为了 9 大类，分别是：

① 视图容器

② 基础内容

③ 表单组件

④ 导航组件

⑤ 媒体组件

⑥ map 地图组件

⑦ canvas 画布组件

⑧ 开放能力

⑨ 无障碍访问

（2）常用的视图容器类组件

① view

⚫ 普通视图区域

⚫ 类似于 HTML 中的 div，是一个块级元素

⚫ 常用来实现页面的布局效果

② scroll-view

⚫ 可滚动的视图区域

⚫ 常用来实现滚动列表效果

scroll-y属性：允许纵向滚动。

scroll-x属性：允许横向滚动。

注意：必须给滚动的方向，scroll-view一个固定的宽高度。

|  |
| --- |
| 1. 首先用srcoll-view包裹元素 2. 如果是纵向滚动，要加上scroll-y（scroll-x） 3. 必须要给scroll-view加上固定高（宽）度   <scroll-view class="container1" scroll-y>  <view>A</view>  <view>B</view>  <view>C</view>  </scroll-view>  /\* pages/test/test.wxss \*/  .container1 {  width: 100px;  border: 1px solid red;  height: 200px;  }  .container1 view {  width: 100px;  height: 100px;  text-align: center;  line-height: 100px;  }  .container1 view:nth-of-type(1) {  background-color: aqua;  }  .container1 view:nth-of-type(2) {  background-color: purple;  }  .container1 view:nth-of-type(3) {  background-color: pink;  }  1 |

③ swiper 和 swiper-item

⚫ 轮播图容器组件 和 轮播图 item 组件

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 类型 | 默认值 | 说明 |
| indicator-dots | boolean | false | 是否显示面板指示点 |
| indicator-color | color | color rgba(0, 0, 0, .3) | 指示点颜色 |
| indicator-active-color | color | #000000 | 当前选中的指示点颜色 |
| autoplay | boolean | false | 是否自动切换 |
| interval | number | 5000 | 自动切换时间间隔 |
| circular | boolean | false | 是否采用衔接滑动 |

|  |
| --- |
| 需要给swiper和swipe-item加上一定的样式，才能看到效果。  <swiper class="swiper">  <swiper-item>  <view class="item">A</view>  </swiper-item>  <swiper-item>  <view class="item">B</view>  </swiper-item>  <swiper-item>  <view class="item">C</view>  </swiper-item>  </swiper>  /\* pages/test/test.wxss \*/  .swiper {  height: 200px;  }  .swiper .item {  height: 100%;  text-align: center;  line-height: 200px;  background-color: red;  }  swiper-item:nth-of-type(2) .item {  background-color: blueviolet;  }  swiper-item:nth-of-type(3) .item {  background-color: skyblue;  }  /\* 注意：nth-of-type一般是属于兄弟关系才能选得到的 \*/ |

（3）常用的基础内容组件

① text

⚫ 文本组件

⚫ 类似于 HTML 中的 span 标签，是一个行内元素

在小程序里边，只有text组件中的内容支持长按选中操作，必须加上selectable。在模拟器中不能实现百分百操作，用手机微信（真机）可以。

注意：[pages/test/test] [text] selectable is deprecated, use user-select instead

|  |
| --- |
| <!--pages/test/test.wxml-->  <view>  手机号码：<text selectable="true">13927043839</text>  手机号码：<text user-select="true">13927043839</text>  </view> |

② rich-text

⚫ 富文本组件

⚫ 支持把 HTML 字符串渲染为 WXML 结构

通过 rich-text 组件的 nodes 属性节点， 把 HTML 字符串渲染为对应的 UI 结构：

|  |
| --- |
| <rich-text nodes="<div style='border: 1px solid red; width: 100px; height: 100px'>hello</div>"></rich-text>  1 |

富文本，使用场景，把服务器返回的html代码，直接渲染成wxml

（4）其它常用组件

① button

⚫ 按钮组件

⚫ 功能比 HTML 中的 button 按钮丰富

⚫ 通过 open-type 属性可以调用微信提供的各种功能（客服、转发、获取用户授权、获取用户信息等）

通过 type 指定按钮的类型

size=mini小尺寸按钮

plain镂空按钮

|  |
| --- |
| <!--pages/test/test.wxml-->  <view>  <button>普通按钮</button>  <button type="primary">主色调按钮</button>  <button type="warn">警告按钮</button>  <!-- 小尺寸按钮 -->  <button size="mini">普通按钮</button>  <button type="primary" size="mini">主色调按钮</button>  <button type="warn" size="mini">警告按钮</button>  <!-- 镂空按钮 -->  <button type="primary" plain>镂空按钮</button>  </view>  1 |

② image

⚫ 图片组件

⚫ image 组件默认宽度约 300px、高度约 240px（**空的话，也会占这么多的位置**）

⚫ image 组件的 mode 属性

image 组件的 mode 属性用来指定图片的**裁剪**和**缩放**模式，常用的 mode 属性值如下：（用一些宽高相差大点，比较容易理解）

|  |  |
| --- | --- |
| mode 值 | 说明 |
| scaleToFill | （默认值）缩放模式，**不保持纵横比缩放**图片，使图片的宽高完全拉伸至填满 image 元素。 |
| aspectFit | 缩放模式，保持纵横比缩放图片，使图片的**长边**能完全显示出来。也就是说，可以完整地将图片显示出来。会有空白 |
| aspectFill | 缩放模式，保持纵横比缩放图片，只保证图片的**短边**能完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取。 |
| widthFix | 缩放模式，**宽度不变，image高度**自动变化，保持原图宽高比不变。 |
| heightFix | 缩放模式，**高度不变，image宽度**自动变化，保持原图宽高比不变。 |

|  |
| --- |
| <view>  <image></image>  <image src="/images/kmr.jpeg" mode="heightFix"></image>  </view> |

③ navigator（后面课程会专门讲解）

⚫ 页面导航组件

⚫ 类似于 HTML 中的 a 链接

6、小程序 API 的 3 大分类

（1）小程序官方把 API 分为了如下 3 大类：

① 事件监听 API

⚫ 特点：以 **on** 开头，用来监听某些**事件的触发**

⚫ 举例： wx.**onWindowResize**(function callback) 监听窗口尺寸变化的事件

wx对象相当于是window顶级对象

② 同步 API

⚫ 特点1 ：以 **Sync** 结尾的 API 都是同步 API

⚫ 特点2：同步 API 的执行结果，可以通过函数返回值直接获取，如果执行出错会抛出异常

⚫ 举例： wx.**setStorageSync**('key', 'value') 向本地存储中写入内容

③ 异步 API

⚫ 特点：类似于 jQuery 中的 $.ajax(options) 函数，需要通过 success、 fail、 complete 接收调用的结果

⚫ 举例： wx.**request**() 发起网络数据请求，通过 success 回调函数接收数据

五、WXML 模板语法 - 数据绑定

1、数据绑定的基本原则

（1）在 data 中定义数据

（2）在 WXML 中使用数据

2、在 data 中定义页面的数据

在页面对应的 .js 文件中，把数据定义到 data 对象中即可：

data: {

name: '凯美瑞',

},

3、Mustache 语法的格式

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>{{ name }}</view>

（1）Mustache 语法的主要应用场景如下：

⚫ 绑定内容---跟name为例子

⚫ 绑定属性

⚫ 运算（三元运算、算术运算等）

4、动态绑定属性

页面的结构如下：

<image src=**"{{ imgSrc }}"** mode="widthFix"></image>

页面的数据如下：

data: {

name: '凯美瑞',

**imgSrc: '/images/kmr.jpeg',**

},

5、三元运算

<view>{{ randomNum >= 5? '数字 >= 5':'数字 < 5' }}</view>

data: {

......

randomNum: Math.random() \* 10,

},

6、算数运算

<view>{{ randomNum1 \* 100 }}</view>

data: {

......

randomNum1: Math.random().toFixed(2),// 取两位小数

},

六、WXML 模板语法 - 事件绑定

1、 什么是事件

事件是**渲染层到逻辑层的通讯方式**。通过事件可以将用户在渲染层产生的行为，反馈到逻辑层进行业务的处理。



2、小程序中常用的事件

|  |  |  |
| --- | --- | --- |
| 类型 | 绑定方式 | 事件描述 |
| tap | bindtap 或 bind:tap | 手指触摸后马上离开，类似 HTML 中的 click 事件 |
| input | bindinput 或 bind:input | 文本框的输入事件 |
| change | bindchange 或 bind:change | 状态改变时触发（如果checkbox状态改变） |

3、事件对象的属性列表

当事件回调触发的时候，会收到一个事件对象 event，它的详细属性如下表所示：

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| type | String | 事件类型 |
| timeStamp | Integer | 页面打开到触发事件所经过的毫秒数 |
| 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 |
| detail | Object | 额外的信息 |
| touches | Array | 触摸事件，当前停留在屏幕中的触摸点信息的数组 |
| changedTouches | Array | 触摸事件，当前变化的触摸点信息的数组 |

4、target 和 currentTarget 的区别

**target** 是**触发该事件的源头组件**；

currentTarget 则是当前事件所绑定的组件。举例如下：

|  |
| --- |
|  |

？？？暂时看到的数据都是一样，id会不一样。待测试

点击内部的按钮时，点击事件以冒泡的方式向外扩散，也会触发外层 view 的 tap 事件处理函数。

此时，对于外层的 view 来说：

⚫ e.target 指向的是触发事件的源头组件，因此， e.target 是内部的按钮组件

⚫ e.currentTarget 指向的是当前正在触发事件的那个组件，因此， e.currentTarget 是当前的 view 组件

5、bindtap 的语法格式

在小程序中，不存在 HTML 中的 onclick 鼠标点击事件，而是通过 tap 事件来响应用户的触摸行为。

1. 通过 bindtap，可以为组件绑定 tap 触摸事件。

（2）在页面的 .js 文件中定义对应的事件处理函数，事件参数通过形参 event（一般简写成 e） 来接收。

|  |
| --- |
| <view class="box">  <button type="primary" bindtap="btnTapHandle" plain="" class="inbox">内部按钮</button>  </view>  与data同级  btnTapHandle(e) {  console.log(e);  }, |

6、在事件处理函数中为 data 中的数据赋值

（1）通过调用 **this.setData(dataObject)** 方法，可以给页面 data 中的数据重新赋值，示例如下

changeDataHandle(e) {

**this.setData({**

count: this.data.count + 1,

**});**

console.log(this.data.count);

},

7、事件传参

小程序中的事件传参比较特殊， 不能在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="changeDataHandle(123)" data-step="3" class="inbox">加 1 按钮</button> |

因为小程序会把 bindtap 的属性值，统一当作**事件名称**来处理，相当于要调用一个名称为 **btnHandler(123)**的事件处理函数。

（1）可以为组件提供 **data-\*** 自定义属性传参，其中 **\*** 代表的是参数的名字，值使用“{{ }}”，如果直接写数字，会被解析为文本字符串。

⚫ **info** 会被解析为**参数的名字**

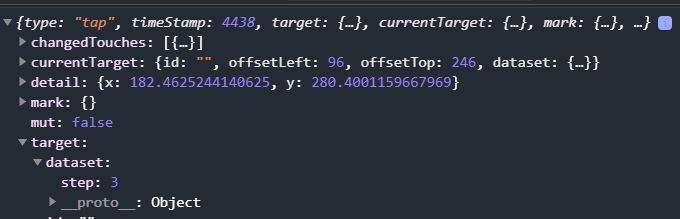
⚫ 数值 **2** 会被解析为**参数的值**

（2）在事件处理函数中，通过 **event.target.dataset.参数名，**即可获取到具体参数的值

|  |
| --- |
| <button type="primary" bindtap="changeDataHandle" data-step="3" class="inbox">加 1 按钮</button>  changeDataHandle(e) {  this.setData({  count: this.data.count + e.target.dataset.step,  });  // console.log(e);  },  1  2 |

使用 {{ }}

<button type="primary" bindtap="changeDataHandle" data-step="{{ 3 }}" class="inbox">加 1 按钮</button>



8、bindinput 的语法格式

在小程序中，通过 input 事件来响应文本框的输入事件，语法格式如下：

（1）通过 bindinput，可以为文本框绑定输入事件：

（2）在页面的 .js 文件中定义事件处理函数：

（3）e.detail.value是变化过后，文本框最新的值（是字符串类型的）

|  |
| --- |
| <input bindinput="inputHandle"></input>  inputHandle(e) {  console.log(e.detail.value);  }, |

（4）实现文本框和 data 之间的数据同步

实现步骤：

① 定义数据

② 渲染结构

③ 美化样式

④ 绑定 input 事件处理函数

|  |
| --- |
| <input bindinput="inputHandle" **value**="{{ text }}"></input>  美化样式  input {  border: 1px solid #000;  margin: 5px;  padding: 5px;  border-radius: 3px;  }  同步数据  inputHandle(e) {  // console.log(e.detail.value);  this.setData({  text: e.detail.value,  });  }, |

七、WXML 模板语法 - 条件渲染

1、wx:if、wx:elif、wx:else（跟if类似）

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

也可以用 **wx:elif** 和 **wx:else** 来添加 else 判断：

wx:else不需要加任何的条件，它是最后一种情况。

|  |
| --- |
| <view wx:if="**{{** text === '1' **}}**">男</view>  <view wx:elif="**{{** text === '2' **}}**">女</view>  <view wx:else>保密</view> |

2. 结合 <block> 使用 wx:if

如果要**一次性控制多个组件的展示与隐藏**，可以使用一个 **<block></block>** 标签将多个组件包装起来，并在<block> 标签上使用 wx:if 控制属性。

注意： <block> 并**不是一个组件**，它只是一个包裹性质的容器，**不会在页面中做任何渲染。可以避免不必要的渲染**

|  |
| --- |
| 结构：  2  渲染结果：  1  <block wx:if="{{ text === '1' }}">  <view>我是block包裹的view</view>  <view>我是block包裹的view</view>  </block>  2 |

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。**条件为true的时候，隐藏**。false的时候，显示。

|  |
| --- |
| <view hidden="{{ flag }}">测试hidden，true隐藏，false显示</view> |

4.、wx:if 与 hidden 的对比

（1）运行方式不同

⚫ wx:if 以动态创建和移除元素的方式，控制元素的展示与隐藏

⚫ hidden 以切换样式的方式（ display: none/block;），控制元素的显示与隐藏

（2）使用建议

⚫ **频繁切换时**，建议使用 **hidden**。

⚫ 控制条件复杂时，建议使用 wx:if 搭配 wx:elif、 wx:else 进行展示与隐藏的切换。

1



八、WXML 模板语法 - 列表渲染

1、wx:for

通过 wx:for 可以根据指定的数组，循环渲染重复的组件结构，语法示例如下：

默认情况下，当前循环项的索引用 index 表示；当前循环项用 item 表示。

2、指定索引和当前项的变量名\*（作为了解，实际用的不多，比较麻烦）

⚫ 使用 **wx:for-index** 可以指定当前循环项的索引的变量名

⚫ 使用 **wx:for-item** 可以指定当前项的变量名

|  |
| --- |
| <view wx:for="{{ fruit }}">索引：{{ index }} 名称：{{ item }}</view>  <view wx:for="{{ fruit }}" **wx:for-index**="i" **wx:for-item**="itemname">索引：{{ **i** }} 名称：{{ **itemname** }}</view>  1 |

（1）会有警告Now you can provide attr `wx:key` for a `wx:for` to improve performance.

3、wx:key 的使用

类似于 Vue 列表渲染中的 **:key**，小程序在实现列表渲染时，也建议为渲染出来的列表项指定唯一的 key 值，从而**提高渲染的效率**。

（1）注意：它不用大括号括起来。

<view wx:for="{{ userList }}" **wx:key="id"**>{{ item.name }}</view>

九、WXSS 模板样式

1、什么是 WXSS

WXSS (WeiXin Style Sheets)是一套**样式语言**，用于美化 WXML 的组件样式，类似于网页开发中的 CSS。

2、WXSS 和 CSS 的关系

WXSS 具有 CSS 大部分特性，同时， WXSS 还对 CSS 进行了扩充以及修改，以适应微信小程序的开发。

（1）与 CSS 相比， WXSS 扩展的特性有：

⚫ **rpx** 尺寸单位

⚫ **@import** 样式导入



3、rpx 的实现原理

（1）什么是 rpx 尺寸单位

**rpx**（ responsive pixel）是微信小程序独有的，用来**解决屏适配的尺寸单位**。

（2）rpx 的实现原理非常简单

鉴于不同设备屏幕的大小不同，为了实现屏幕的自动适配， rpx 把所有设备的屏幕，在宽度上**等分为 750 份**（即：**当前屏幕的总宽度为 750rpx**）。

⚫ 在**较小**的设备上， 1rpx 所代表的**宽度较小**。

⚫ 在**较大**的设备上， 1rpx 所代表的**宽度较大**。

小程序在不同设备上运行的时候，会自动把 rpx 的样式单位换算成对应的像素单位来渲染，从而实现屏幕适配。这样我们就不用去做处理了。

4、rpx 与 px 之间的单位换算\*（了解）

在 iPhone6 上，屏幕宽度为375px，共有 750 个物理像素，等分为 750rpx。则：

750rpx = 375px = 750 物理像素

1rpx = 0.5px = 1 物理像素

2rpx = 1px = 2 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 = 2.34rpx |
| iPhone6 | 1rpx = 0.5px | 1px = 2rpx |
| hone6 Plus | 1rpx = 0.552px | 1px = 1.81rpx |

官方建议：开发微信小程序时，设计师可以用 **iPhone6** 作为视觉稿的标准。

开发举例：在 iPhone6 上如果要绘制宽**100px**， 高**20px**的盒子，换算成rpx单位，宽高分别为 **200rpx** 和 **40rpx**。

5、@import样式导入

（1）使用 WXSS 提供的 **@import** 语法，可以导入外联的样式表。

（2）@import 的语法格式

@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。

**@import** '/common/commonWXSS.wxss'**;**

6、全局样式和局部样式

（1）app.wxss全局样式

定义在 app.wxss 中的样式为全局样式，作用于每一个页面。

（2）局部样式

在页面的 .wxss 文件中定义的样式为局部样式，只作用于当前页面。

注意：

① 当局部样式和全局样式冲突时，根据**就近原则**，局部样式会**覆盖**全局样式

② 当局部样式的**权重大于或等于**全局样式的权重时，才会覆盖全局的样式

当鼠标移动到类名上边的时候，会显示权重

十、全局配置

1、 **app.json** 全局配置文件及常用的配置项

小程序根目录下的 **app.json** 文件是小程序的**全局配置文件**。常用的配置项如下：

（1）pages

⚫ 记录当前小程序所有页面的存放路径

**（2）window**

全局设置小程序窗口的外观，是导航栏区域、背景区域（默认是不可见的，下拉才显示），下边一整块，是属于页面的整体区域。

①了解 window 节点常用的配置项

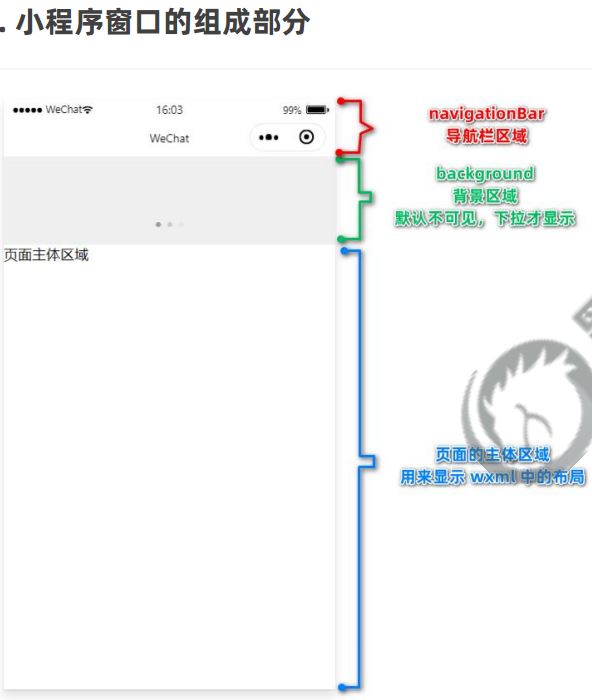
|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 类型 | 默认值 | 说明 |
| navigationBarTitleText | String | 字符串 | 导航栏标题文字内容 |
| navigationBarBackgroundColor | HexColor | #000000 | 导航栏背景颜色，如 #000000 |
| navigationBarTextStyle | String | white | 导航栏标题颜色，仅支持 black / white  注意： navigationBarTextStyle 的可选值只有 black 和 white |
| backgroundColor | HexColor | #ffffff | 窗口的背景色，下拉时候出现的背景。 |
| backgroundTextStyle | String | dark | 下拉 loading 的样式，仅支持 dark / light |
| enablePullDownRefresh | Boolean | false | 是否全局开启下拉刷新 |
| onReachBottomDistance | Number | 50 | 页面上拉触底事件触发时距页面底部距离，单位为px |

（3）tabBar

⚫ 设置小程序底部的 tabBar 效果

（4）style

⚫ 是否启用新版的组件样式，如果想使用旧版本，那么把这语句删除就可以



（5）全局开启下拉刷新功能

概念： 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。

设置步骤： app.json -> window -> 把 **enablePullDownRefresh** 的值设置为 true。

注意：在 app.json 中启用下拉刷新功能，会作用于每个小程序页面！

真机上下拉之后，他不会合上去，需要另外做处理，确实是的

如果模拟器中出现一些问题，以真机为主。

（6）设置下拉刷新时 loading 的样式（三个小圆点）

当**全局开启下拉刷新功能之后**，默认窗口的 loading 样式为白色。

如果要更改 loading 样式的效果，设置步骤为 app.json -> window -> 为 backgroundTextStyle 指定 dark 值。效果如下：

注意： backgroundTextStyle 的可选值只有 light 和 dark

（7）设置上拉触底的距离（距离底部多少时候，加载更多数据）

概念：**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。

设置步骤： app.json -> window -> 为 onReachBottomDistance 设置新的数值，数值就ok。

注意： **默认距离为50px**，如果没有特殊需求，建议使用默认值即可。

2、全局配置 - tabBar（实现页面的快速切换）

tabBar 是移动端应用常见的页面效果， 用于实现多页面的快速切换。小程序中通常将其分为：**底部 tabBar**，**顶部 tabBar。**

注意：

⚫ tabBar中只能配置**最少 2 个**、 **最多 5 个** tab 页签

⚫ 当渲染**顶部** tabBar 时， 不显示 icon，只显示文本



3、tabBar 的 6 个组成部分

① backgroundColor： tabBar 的背景色

② selectedIconPath：选中时的图片路径

③ borderStyle： tabBar 上边框的颜色

④ iconPath：未选中时的图片路径

⑤ selectedColor： tab 上的文字选中时的颜色

⑥ color： tab 上文字的默认（未选中）颜色



4、tabBar 节点的配置项

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性 | 类型 | 必填 | 默认值 | 描述 |
| position | String | 否 | bottom | tabBar 的位置， 仅支持 bottom/top |
| borderStyle | String | 否 | black | tabBar 上边框的颜色，仅支持black/white |
| color | HexColor | 否 |  | tab 上文字的默认（未选中）颜色 |
| selectedColor | HexColor | 否 |  | tab 上的文字选中时的颜色 |
| backgroundColor | HexColor | 否 |  | tabBar 的背景色 |
| list | Array | **是** |  | 是 tab 页签的列表；  最少 **2** 个、最多 **5** 个 tab |

5、每个 tab 项的配置选项（list---Array数组对象中---配置）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 类型 | 必填 | 描述 |
| pagePath | String | 是 | 页面路径，页面必须在 pages 中预先定义，不能用 / 根路径开头 |
| text | String | 是 | tab上显示的文字 |
| iconPath | String | 否 | **未选中**时的图标路径；当 postion 为 top 时，不显示 icon |
| selectedIconPath | String | 否 | **选中时**的图标路径；当 postion 为 top 时，不显示 icon |

注意：tabBar数字对应pages的页面要放在，pages最前边，tabBar页面必须放在头部位置

6、页面.json配置

（1）页面配置文件.json的作用

小程序中，每个页面都有自己的 .json 配置文件，用来对**当前页面**的窗口外观、页面效果等进行配置。

（2）页面配置和全局配置的关系

小程序中， app.json 中的 window 节点，可以全局配置小程序中每个页面的窗口表现。

如果某些小程序页面想要拥有**特殊的窗口表现**，此时，“页面级别的 .json 配置文件”就可以实现这种需求。

注意：当页面配置与全局配置**冲突**时，根据**就近原则**，最终的效果以**页面配置为准**。

（3）页面配置中常用的配置项（直接写键值对就行，不用写window节点）

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 类型 | 默认值 | 说明 |
| navigationBarTitleText | String | 字符串 | 当前页面导航栏标题文字内容 |
| navigationBarBackgroundColor | HexColor | #000000 | 当前页面导航栏背景颜色，如 #000000 |
| navigationBarTextStyle | String | white | 当前页面导航栏标题颜色，仅支持 black / white  注意： navigationBarTextStyle 的可选值只有 black 和 white |
| backgroundColor | HexColor | #ffffff | 当前页面窗口的背景色，下拉时候出现的背景。 |
| backgroundTextStyle | String | dark | 当前页面下拉 loading 的样式，仅支持 dark / light |
| enablePullDownRefresh | Boolean | false | 是否为当前页面全局开启下拉刷新 |
| onReachBottomDistance | Number | 50 | 页面上拉触底事件触发时距页面底部距离，单位为px |

在全局不推荐开启下拉，一般是列表才开，在当前页面中再去开

十一、网络数据请求（需要补充它完成时的方法）

1、小程序中网络数据请求的限制

出于安全性方面的考虑，小程序官方对数据接口的请求做出了如下两个限制：

（1）只能请求 **HTTPS** 类型的接口

（2）必须将**接口的域名**添加到**信任列表**中



2、配置 request 合法域名

（1）需求描述：假设在自己的微信小程序中，希望请求 https://www.escook.cn/ 域名下的接口。

（2）配置步骤：登录微信小程序管理后台 -> 开发 -> 开发设置 -> 服务器域名 -> 修改 request 合法域名

（3）注意事项：

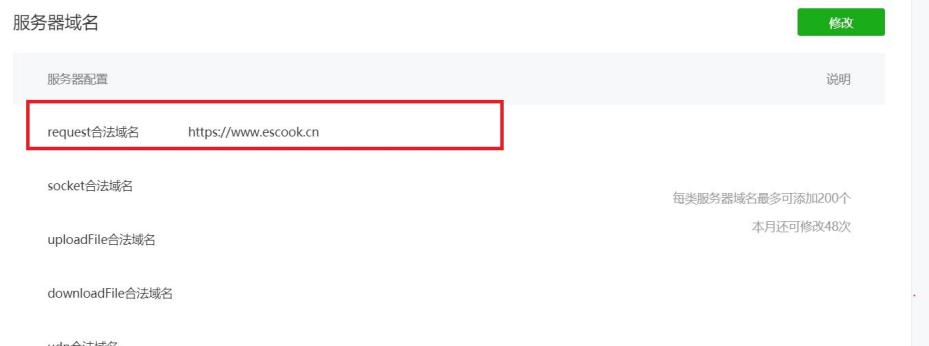
① 域名只支持 https 协议

② 域名不能使用 IP 地址或 localhost

③ 域名必须经过 ICP 备案

④ 服务器域名一个月内最多可申请 5 次修改（微信官网的显示是有问题的）





3、wx.request()发起 GET 请求

注意：success要写成箭头函数，写成普通函数，会造成this的指向改变，不知道是指向谁。无法设置this.setData()。

调用微信小程序提供的 **wx.request()** 方法，可以发起 **GET** 数据请求。

|  |
| --- |
| // 发送get请求  getInfo() {  wx.request({  url: 'https://www.escook.cn/api/get',  data: {  name: 'Tom',  age: 23,  },  method: **'GET'**,//小写也行  success: (result) => {  console.log(result.data);  }  })  }, |

4、wx.request()发起 POST 请求

调用微信小程序提供的 **wx.request()** 方法，可以发起 **POST** 数据请求。

|  |
| --- |
| // 发送post请求  postInfo() {  wx.request({  url: 'https://www.escook.cn/api/**post'**,  data: {  name: 'Jerry',  age: 23,  },  **method: 'post',**  success: (result) => {  console.log(result.data);  }  })  }, |

5、在页面刚加载时onLoad，请求数据

在很多情况下，我们需要在**页面刚加载的时候， 自动请求一些初始化**的数据。此时需要在页面的 onLoad 事件中调用获取数据的函数。

|  |
| --- |
| /\*\*  \* 生命周期函数--监听页面加载  \*/  onLoad: function (options) {  this.getInfo();  this.postInfo();  }, |

6、跳过 request 合法域名校验

如果后端程序员**仅仅提供了 http 协议的接口**、暂时**没有提供 https协议**的接口。

此时为了不耽误开发的进度，我们可以在微信开发者工具中，临时开启「 开发环境不校验请求域名、 TLS 版本及 HTTPS 证书」选项，跳过 request 合法域名的校验。

注意：跳过 request 合法域名校验的选项， 仅限在**开发**与**调试阶段**使用！

没有把域名添加到那个服务器域名列表中，也能请求到。

例如请求url: 'http://www.liulongbin.top:3005/api/getprodlist'，跳过之后能获取数据，会有警告





7、关于跨域和 Ajax 的说明

跨域问题**只存在于**基于浏览器的 Web 开发中。由于**小程序的宿主环境**不是浏览器，而是**微信客户端**，所以**小程序中不存在跨域的问题**。

Ajax 技术的核心是依赖于浏览器中的 XMLHttpRequest 这个对象，由于小程序的宿主环境是微信客户端，所以小程序中不能叫做“发起 Ajax 请求”，而是叫做“**发起网络数据请求”**。

十二、本地生活小案例

1、首页效果以及实现步骤

① 新建项目并梳理项目结构

② 配置导航栏效果

③ 配置 tabBar 效果

④ 实现轮播图效果

⑤ 实现九宫格效果

⑥ 实现图片布局

2、接口地址

（1）获取轮播图数据列表的接口

⚫ 【 GET】 https://www.escook.cn/slides

（2）获取九宫格数据列表的接口

⚫ 【 GET】 https://www.escook.cn/categories

（3）获取随机颜色数据接口

⚫ 【 GET】 https://www.escook.cn/api/color

（4）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

⚫ 类https://www.escook.cn/categories/:cate\_id/shops

⚫ URL 地址中的 :cate\_id 是动态参数，表示分的 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

如果一个属性中带了其他符号，就不能直接点出来，要通过x[‘xxx-xx’]这样的方式

下拉刷新，要把所有的数据都重置成最初的样子，下拉结束的话，如果放在数据请求完毕，取消下拉的处理，那么它会重复被执行，这样就给他传一个回调函数，设置一下 &&，如果传递了回调，就帮我们去执行一下这个回调函数，

// 处理下拉刷新

cb && cb();

它返回header中的total是什么意思

要多去看官方文档的说明

style="background-color: rgba({{ item }})

跟html的有区别

十三、页面导航

1、什么是页面导航（a，localtion.href）

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）编程式导航

⚫ 调用小程序的导航 API，实现页面的跳转

十四、页面导航 - 声明式导航

1、导航到 tabBar 页面（**switchTab**）

**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**（如果没有写就跳转不成功）。

|  |
| --- |
| <navigator url="/pages/message/message" open-type="**switchTab**">去往信息页面</navigator> |

2、导航到非 tabBar 页面（navigate）

**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**

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

|  |
| --- |
| <navigator url="/pages/index/index" open-type="**navigate**">导航去往非tabbar页面</navigator> |

3、后退导航（**navigateBack**）

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

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** 。

|  |
| --- |
| <navigator open-type="**navigateBack**" delta="1">后退</navigator> |

十五、页面导航 - 编程式导航

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） |

// 跳转message页面

goToMessage() {

wx.**switchTab**({

url: '/pages/message/message',

})

},

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，跟上边一样。

goToIndex() {

wx.**navigateTo**({

url: '/pages/index/index',

})

},

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 | 是否必选 | 说明 |
| delta | number，默认值是 **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） |

// 后退

goBack() {

wx.**navigateBack**();

},

十六、页面导航 - 导航传参

1、声明式导航传参（tabbar页面不能传递？？？）

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：

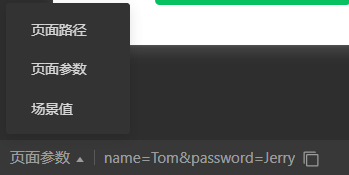
⚫ **参数**与**路径**之间使用 ? 分隔

⚫ **参数键**与**参数值**用 **=** 相连

⚫ 不同参数用 **&** 分隔

这里跳转到非tabber页面

<navigator url="/pages/index/index**?name=Tom&password=Jerry**" open-type="navigate">导航去往非tabbar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。

goToIndex() {

wx.navigateTo({

url: '/pages/index/index**?name=Tom&password=Jerry',**

})

},

3、在 onLoad 中接收导航参数

通过声明式导航传参或编程式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：

一般会把options值挂载到data，因为options别人访问不到。

onLoad(options) {

this.setData({

query: options

}),

console.log(this.data.query);

},

1

4、注意，tabbar页面无法传参

十七、页面事件 - 下拉刷新事件

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（跟data同级）即可监听当前页面的下拉刷新事件。松手之后会触发

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，下拉刷新后置为 0。

|  |
| --- |
| <view>{{ count }}</view>  <button bindtap="addCount" type="primary">+1</button>  /\*\*  \* 页面的初始数据  \*/  data: {  count: 0,  },  // 加一操作  addCount() {  this.setData({  count: this.data.count + 1,  });  },  /\*\*  \* 页面相关事件处理函数--监听用户下拉动作  \*/  onPullDownRefresh: function () {  this.setData({  count: 0,  });  }, |

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。（在真机上才出现效果，模拟器不能看到）

onPullDownRefresh: function () {

this.setData({

count: 0,

});

// 数据的处理已经完毕了，停止下拉刷新

wx.stopPullDownRefresh();

},

十八、页面事件 - 上拉触底事件

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。反复的在边界处上下滑动，会不断的触发上拉触底的事件，这个是在实际中是需要进行处理的。**需要做节流的处理**。

/\*\*

\* 页面上拉触底事件的处理函数

\*/

**onReachBottom**: function () {

console.log('上拉触底');

},

3、配置上拉触底距离

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（滚动条距离底部），在实际开发中，可以根据自己的需求修改这个默认值。

4、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

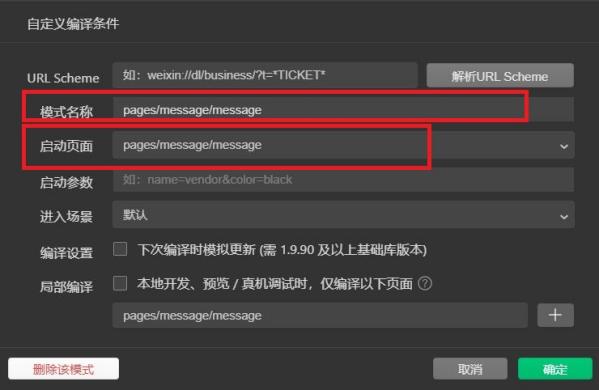
⚫ 如果节流阀的值为 true，则阻止当前请求

⚫ 如果节流阀的值为 false，则发起数据请求

十九、扩展 - 自定义编译

1、自定义编译模式

当我们修改样式编译之后，它又显示首页，我们要去手动去点击到当前的页面，这样就比较麻烦。可以自定义编译，编译后定位我们启动的页面。



2、添加切后台功能，在工具中---工具管理，将后台功能选择

二十、生命周期

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， 强调的是一个时间段。例如：

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

（1）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**。

（2）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。

（3）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

小程序的应用生命周期函数需要在 **app.js** 中进行声明（有四个）

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

onLaunch: function () {

console.log('onlaunch');

},

console.log('onshow');小程序启动，或从后台进入前台时触发

onShow: function (options) {

},

小程序从前台进入后台时触发

onHide: function () {

console.log('onhide');

},

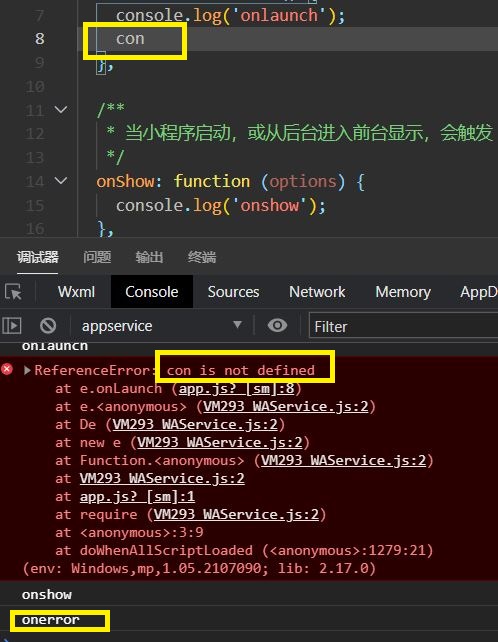
当小程序发生脚本错误，或者 api 调用失败时，会触发 onError 并带上错误信息

onError: function (msg) {

console.log('onerror');

}





（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

生命周期函数--监听页面加载，一个页面只调用一次，切换并不会触发

onLoad: function (options) {

console.log(options);

},

生命周期函数--监听页面初次渲染完成，一个页面只调用一次，一些动态渲染的操作要放在页面渲染完成后，才行，页面都没有就渲染不了

onReady: function () {

},

生命周期函数--监听页面显示，执行多次，切换会触发，它会比onReady先执行

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，一个页面只调用一次

onUnload: function () {

},

5、Tabbar页面初始化之后不会被销毁，也就是只会执行一次 onLoad 函数

二十一、WXS 脚本

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”。对数据进行包装处理，如格式化时间。

3、（了解）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

4、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：

<view>{{ m1.toUpper(username) }}</view>

<wxs module="m1">

module.exports.toUpper = function (str) {

return str.toUpperCase();

}

</wxs>

5、使用外联的 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。

function toUpper(str) {

return str.toUpperCase();

}

**module.exports** = {

toUpper: toUpper,

}

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**

<view>{{ m2.toUpper(country) }}</view>

<wxs src="../../utils/tools.wxs" module="**m2**"></wxs>

6、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！

（1）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（{{xxxxxxxx}}）进行使用，例如：

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：

给按钮绑定了，wxs中的函数是不可以的。

（2）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

（3）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。

二十二、自定义组件 - 组件的创建与引用、样式

1、创建组件

（1）在项目的根目录中，鼠标右键，创建名字为 components -> com 文件夹

（2）在新建的 components -> com 文件夹上，鼠标右键，点击“**新建 Component**”

（3）键入**组件的名称之后回车**，会自动生成组件对应的 4 个文件，后缀名分别为 .js， .json， .wxml 和 .wxss，这四个文件最终编译为一个文件。在引入的时候，只需要引入名称就可以。

注意：为了保证目录结构的清晰，建议把不同的组件，存放到单独目录中，例如：



2、引用组件

组件的引用方式分为“局部引用 ”和“全局引用”，顾名思义：

（1）局部引用：组件只能在当前被引用的页面内使用

在页面的 .json 配置文件中引用组件的方式，叫做“局部引用”。

{

"usingComponents": {

"my-com": "/components/com/com"

},

}

在home.wxml文件中

<my-com></my-com>

（2）全局引用：组件可以在**每个小程序页面**中使用（用的比较多）

在 app.json 全局配置文件中引用组件的方式，跟window平级，叫做“全局引用”。

3、全局引用 VS 局部引用

根据组件的使用频率和范围，来选择合适的引用方式：

⚫ 如果某组件在多个页面中**经常被用到**，建议进行“全局引用”

⚫ 如果某组件只在**特定的页面**中被用到，建议进行“局部引用”

4、组件和页面的区别

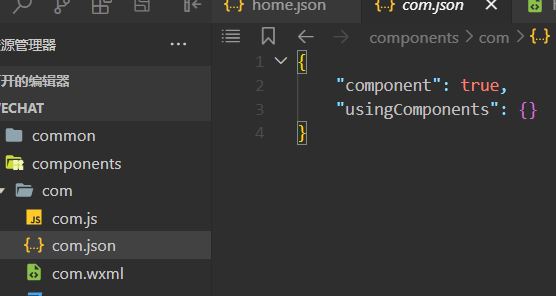
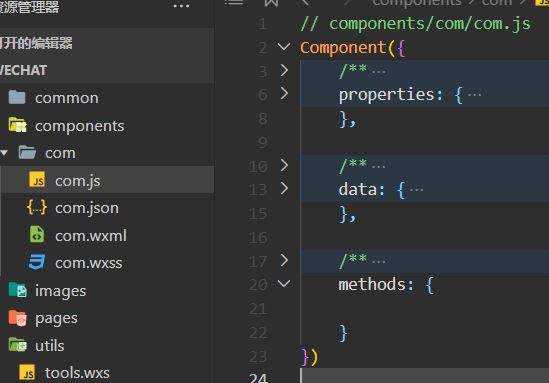
从表面来看，组件和页面都是由 .js、 .json、 .wxml 和 .wxss 这四个文件组成的。

但是，组件和页面的 **.js** 与**.json** 文件有明显的不同：

（1）组件的 .json 文件中需要声明 **"component": true** 属性

（2）组件的 .js 文件中调用的是 **Component()** 函数

（3）组件的事件处理函数需要定义到 **methods** 节点中



5、自定义组件样式

（1）组件样式隔离

默认情况下，自定义组件的样式只对当前组件生效，不会影响到组件之外的UI 结构，如图所示：

⚫ 组件 A 的样式**不会**影响组件 C 的样式

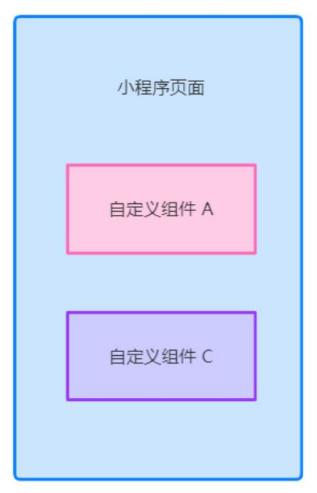
⚫ 组件 A 的样式**不会**影响小程序页面的样式

⚫ 小程序页面的样式**不会**影响组件 A 和 C 的样式

好处：

① 防止外界的样式影响组件内部的样式

② 防止组件的样式破坏外界的样式



（2）组件样式隔离的注意点

⚫ app.wxss 中的全局样式对组件无效（针队于class选择器）

⚫ 只有 class 选择器会有样式隔离效果， id 选择器、属性选择器、标签选择器不受样式隔离的影响（全局app.wxss）

建议：在**组件**和**引用组件的页面**中建议使用 class 选择器， **不要使用** id、属性、标签选择器！

（3）修改组件的样式隔离选项

默认情况下，自定义组件的样式隔离特性能够防止组件内外样式互相干扰的问题。但有时，我们希望在外界能够控制组件内部的样式，此时，可以通过 styleIsolation 修改组件的样式隔离选项，用法如下：

Component({

    options: {

        styleIsolation: 'shared',

    },

}

（4）styleIsolation 的可选值（针对于类样式？待进一步官网确认）

注意：可选值在双引号中，不要添加空格，或者其他符号，会导致失效

|  |  |  |
| --- | --- | --- |
| 可选值 | 默认值 | 描述 |
| isolated | 是 | 表示启用样式隔离，在自定义组件内外，使用 class 指定的样式将不会相互影响。 |
| apply-shared | 否 | 表示页面 wxss 样式将影响到自定义组件，但自定义组件 wxss 中指定的样式不会影响页面。（局限于类样式） |
| shared | 否 | 表示页面 wxss 样式将影响到自定义组件，自定义组件 wxss 中指定的样式也会影响页面和其他设置了 apply-shared 或 shared 的自定义组件。 |

二十三、自定义组件 - 数据、方法和属性

1、data 数据

在小程序组件中，用于组件模板渲染的**私有数据**，需要定义到 **data 节点**中。

2、methods 方法

在小程序组件中，**事件处理函数（给tap、input等事件处理）**和**自定义方法（输出信息什么）**需要定义到 **methods 节点**中，示例代码如下：

自定义方法建议以 \_ 下划线开头，便于区别。

|  |
| --- |
| methods: {  addCount() {  this.setData({  count: this.data.count + 1,  });  this.\_showCount();  },  **\_showCount()** {  wx.showToast({  title: this.data.count + '',  icon: 'none',  })  }  }  <button type="primary" plain bindtap="addCount">+1</button> |

3、properties 属性

在小程序组件中，properties 是组件的对外属性， 用来接收外界传递到组件中的数据，通过组件上的属性来传递数据。

（1）完整定义属性的方式（当需要指定属性的默认值时，建议使用此方式）

需要配置属性值的类型，属性默认值

（2）简化定义属性的方式（不需要指定属性默认值时，可以使用简化方式）

|  |
| --- |
| home.wxml  <my-com max="7"></my-com>  com.js  properties: {  max: {  type: Number,  value: 10,  }  // 简写形式  // max: Number  },  methods: {  addCount() {  if (this.data.count >= this.properties.max) return;  this.setData({  count: this.data.count + 1,  });  this.\_showCount();  }, |

4、data 和 properties 的区别

在小程序的组件中，properties 属性和 data 数据**的用法相同**，它们都是**可读可写**的，只不过：

⚫ data 更倾向于存储组件的私有数据

⚫ properties 更倾向于存储**外界**传递到组件中的数据

它们都是指向了同一个对象（同一个地址）。它们的倾向是人为划分

|  |
| --- |
| showDataAndProperties() {  console.log(this.data);  console.log(this.properties);  console.log(this.data === this.properties);  }  1 |

5、使用 setData 修改 properties 的值

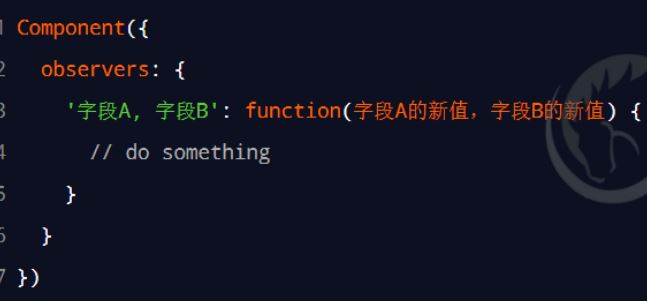
由于 data 数据和 properties 属性在**本质上没有任何区别**，因此 properties 属性的值**也可以用于页面渲染**，或使用 setData 为 properties 中的属性重新赋值，示例代码如下：

|  |
| --- |
| <view>max的值：{{ max }}</view>  addCount() {  if (this.data.count >= this.properties.max) return;  this.setData({  count: this.data.count + 1,  max: this.properties.max + 1,  });  this.\_showCount();  }, |

二十四、自定义组件 - 数据监听器

1、什么是数据监听器

数据监听器用于监听和响应任何属性和数据字段的变化，从而执行特定的操作。它的作用类似于 vue 中的watch 侦听器。在小程序组件中，数据监听器的基本语法格式如下：



字段顺序，跟字段新值顺序对应

2、数据监听器的基本用法

组件的 UI 结构如下：

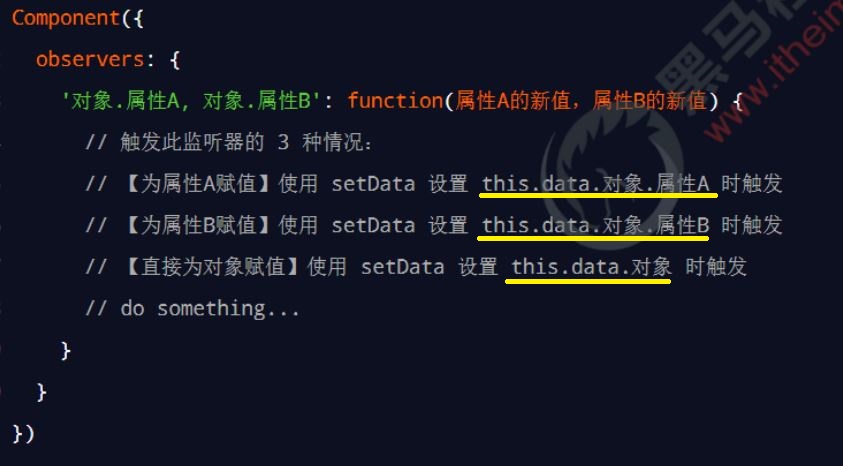
|  |
| --- |
| <view>{{ **n1** }} + {{ **n2** }} = {{ **sum** }}</view>  <button type="primary" plain bindtap="addN1">n1自增</button>  <button type="warn" plain bindtap="addN2">n2自增</button> |

组件的 .js 文件代码如下：

|  |
| --- |
| observers: {  **'**n1, n2**'**: function (newN1, newN2) {  this.setData({  sum: newN1 + newN2,  });  }  }, |

3、监听对象属性的变化

数据监听器支持监听对象中**单个或多个属性**的变化，示例语法如下：



4、监听对象中所有属性的变化（代码测试，直接卡死电脑）

如果某个对象中需要被监听的属性太多，为了方便，可以使用 **通配符 \*\*** 来监听对象中所有属性的变化。

二十五、自定义组件 - 纯数据字段

1、什么是纯数据字段

概念： 纯数据字段指的是那些不用于界面渲染的 **data** 字段。

应用场景：例如有些情况下，某些 data 中的字段既不会展示在界面上， 也不会传递给其他组件，仅仅在当前组件内部使用。带有这种特性的 data 字段适合被设置为纯数据字段。

好处：纯数据字段有助于提升页面更新的性能。

2、使用规则

在 Component 构造器的 options 节点中，指定 **pureDataPattern** 为一个**正则表达式**，字段名符合这个正则表达式的字段将成为纯数据字段，示例代码如下：

跟methods同级

options: {

pureDataPattern: **/^\_/**,

},

3、使用纯数据字段改造数据监听器案例

纯数据是无法选渲染到页面上的，\_rgb的数据是无法渲染到页面上的。

data: {

\_rgb: {

r: 0,

g: 0,

b: 0,

},

step: 1,

fullColor: '0, 0, 0',

},

二十六、自定义组件 - 组件的生命周期

1、组件**全部**的生命周期函数

小程序组件可用的全部生命周期如下表所示：

|  |  |  |
| --- | --- | --- |
| 生命周期函数 | 参数 | 描述说明 |
| **created** | 无 | 在组件实例**刚刚被创建时**执行 |
| **attached** | 无 | 在组件实例**进入页面节点树后**执行 |
| ready | 无 | 在组件在视图层布局完成后执行 |
| moved | 无 | 在组件实例被移动到节点树另一个位置时执行 |
| **detached** | 无 | 在组件实例被**从页面节点树移除时**执行 |
| error | Object | Error 每当组件方法抛出错误时执行 |

2、组件主要的生命周期函数

在小程序组件中，最重要的生命周期函数有 3 个，分别是 created、 **attached**、 detached。它们各自的特点

如下：

（1）created 组件实例刚被创建好的时候，created 生命周期函数会被触发

⚫ 此时还不能调用 setData

⚫ 通常在这个生命周期函数中，只应该用于给组件的 this 添加一些自定义的属性字段

（2）attached 在组件**完全初始化完毕、进入页面节点树后**，attached 生命周期函数会被触发。

⚫ 此时， this.data 已被初始化完毕

⚫ 这个生命周期很有用，绝大多数初始化的工作可以在这个时机进行（例如发请求获取初始数据）

（3）detached 在组件离开页面节点树后，detached 生命周期函数会被触发

⚫ 退出一个页面时，会触发页面内每个自定义组件的 detached 生命周期函数

⚫ 此时适合做一些清理性质的工作

3、lifetimes 节点

在小程序组件中，生命周期函数可以直接定义在 Component 构造器的第一级参数中，可以在 lifetimes 字段内进行声明（ 这是推荐的方式，其优先级最高）。示例代码如下：



如果同时在Component中声明的话，那么在lifetimes对象中函数生效。

4、组件所在页面的生命周期

（1）自定义组件的行为**有时依赖于页面状态的变化**，此时就需要用到**组件所在页面的生命周期**。组件可以在自己的 .js 中监听页面的生命周期。

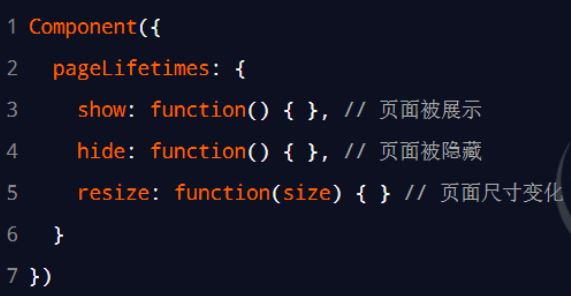
例如：每当触发页面的 show 生命周期函数的时候，我们希望能够重新生成一个随机的 RGB 颜色值。

在自定义组件中，组件所在页面的生命周期函数有如下 3 个，分别是：

|  |  |  |
| --- | --- | --- |
| 生命周期函数 | 参数 | 描述 |
| show | 无 | 组件所在的页面被展示时执行 |
| hide | 无 | 组件所在的页面被隐藏时执行 |
| resize | Object | Size 组件所在的页面尺寸变化时执行 |

（2）pageLifetimes 节点

组件所在页面的生命周期函数，需要定义在 **pageLifetimes** 节点中，示例代码如下：

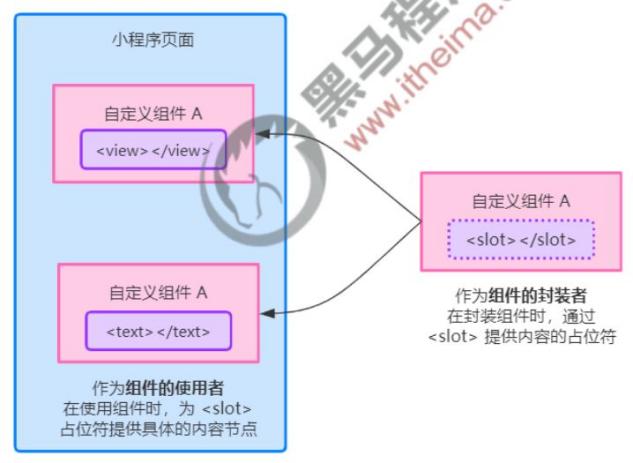


|  |
| --- |
| // 监听所在页面的生命周期，注意不能写成箭头函数，它是在构造函数中，要指向所属对象。  pageLifetimes: {  show: function () {  this.\_getRandomColor();  },  },  \_getRandomColor() {  this.setData({  '\_rgb.r': Math.floor(Math.random() \* 256),  '\_rgb.g': Math.floor(Math.random() \* 256),  '\_rgb.b': Math.floor(Math.random() \* 256),  });  }, |

二十七、自定义组件 - 插槽

1、什么是插槽

在自定义组件的 wxml 结构中，可以提供一个 <slot> 节点（插槽）， 用于**承载组件使用者提供的 wxml 结构**。



2、单个插槽

在小程序中，默认每个自定义组件中只允许使用一个 <slot> 进行占位，这种**个数上的限制**叫做单个插槽。

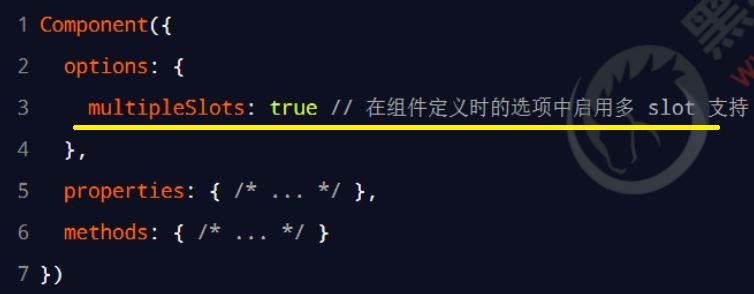
记得去app.json、或者 .js 注册组件后才能使用。



3、启用并使用多个插槽

（1）启用插槽

在小程序的自定义组件中，需要使用多 <slot> 插槽时，可以在组件的 .js 文件中，通过如下方式进行启用。



（2）定义多个插槽

可以在组件的 .wxml 中使用多个 <slot> 标签，以不同的 **name** 来区分不同的插槽。

（3）使用多个插槽

在使用 带有多个插槽的自定义组件时，需要用 **slot 属性**来将节点插入到不同的 <slot> 中。

|  |
| --- |
| 组件使用者  <my-com-slot>  <view slot="before">前面的插槽</view>  <view slot="after">后面的插槽</view>  </my-com-slot>  组件  <slot name="before"></slot>  <view>我是组件的内部</view>  <slot name="after"></slot>  1 |

二十八、自定义组件 - 父子组件之间的通信

1、父子组件之间通信的 3 种方式

（1）属性绑定

⚫ 用于父组件向子组件的指定属性设置数据，仅能设置 JSON 兼容的数据。

（2）事件绑定

⚫ 用于子组件向父组件传递数据，可以传递任意数据（简单数据、方法）。

（3）获取组件实例

⚫ 父组件还可以通过 this.selectComponent() 获取子组件实例对象。

⚫ 这样就可以直接访问子组件的任意数据和方法。

2、属性绑定，父 传 子（单向传递的，子组件值的改变，不会影响父组件的值）

（1）属性绑定用于实现**父向子**传值，而且只能传递**普通类型的数据**，无法将方法传递给子组件。

|  |
| --- |
| data: {  count: 100,  },  <my-com-slot countFromFather="{{ count }}"></my-com-slot> |

（2）子组件在 properties 节点中声明对应的属性并使用。

|  |
| --- |
| properties: {  countFromFather: Number,  },  <view>父组件传递的值 {{ countFromFather }}</view> |

3、事件绑定，子 传 父

**bind:**xxx，自定义事件命名（需要看官网的是不是正确）

事件绑定用于实现子向父传值，可以传递任何类型的数据。使用步骤如下：

（1）在**父组件**的 js 中，定义一个函数，这个函数即将通过自定义事件的形式，传递给子组件。

（2）在**父组件**的 wxml 中，通过自定义事件的形式，将步骤 1 中定义的函数引用，传递给子组件。

（3）在**子组件**的 js 中，通过调用 **this.triggerEvent('自定义事件名称', { /\* 参数对象 \*/ }**) ，将数据发送到父组件。

注意：在传递参数的时候，你的值是给什么属性，在父组件e.detail.属性的名字要一样

（4）在**父组件**的 js 中，通过 e.detail 获取到子组件传递过来的数据。

|  |
| --- |
| 父组件 .js  // 给子组件绑定事件  syncCount(e) {  // console.log('syncCount');  // console.log(e);  this.setData({  count: e.detail.value,  });  },  1 |
| 父组件wxml  <my-com-slot countFromFather="{{ count }}" **bind:**sync="syncCount"></my-com-slot> |
| 子组件 js  addCount() {  this.setData({  countFromFather: this.data.countFromFather + 1,  });  **this.triggerEvent**('sync', {value: this.data.countFromFather});  }, |

4、获取组件实例

可在父组件里调用 this.selectComponent("**id或class选择器**") ，获取子组件的实例对象，从而直接访问子组件的任意数据和方法。调用时需要传入**一个选择器**，例如 this.selectComponent(".my-component")。

（1）先提供class选择器，或者id选择器

|  |
| --- |
| <my-com-slot countFromFather="{{ count }}" bind:sync="syncCount" class="my-slot"></my-com-slot>  // 获取子组件中的对象  getChildCom() {  const child = this.selectComponent('.my-slot');  // console.log(child);  console.log(child.properties.countFromFather);  child.addCount();  },  注意：跟调用子组件方法的时候，注意不要使用this，因为此时的this指向的是父组件实例 |

二十九、自定义组件 - behaviors，代码共享

1、什么是 behaviors

behaviors 是小程序中，用于实现**组件间代码共享**的特性，类似于 Vue.js 中的 “mixins”。引用之后代码是独立的，它是不同的实例。

2、behaviors 的工作方式

每个 behavior 可以包含一组（共享的属性才有必要，私有的就没有必要了）**属性、 数据、 生命周期函数和方法**。组件引用它时，它的属性、数据和方法会**被合并到组件**中。

每个组件可以引用多个 behavior， behavior 也可以引用其它 behavior。

3、创建 behavior

调用 Behavior(Object object) 方法即可创建一个共享的 behavior 实例对象，供所有的组件使用：

|  |
| --- |
| module.exports = **Behavior**({  data: {  username: 'lxf',  },  properties: {  },  methods: {  },  }); |

4、导入并使用 behavior

在组件中，使用 **require()** 方法导入需要的 behavior， **挂载后即可访问 behavior 中的数据或方法。**

（1）使用require() 导入需要自定义使用的 behavior 模块

（2）将导入的 behavior 实例对象，挂载到 behaviors（他是一个数组） 节点中就能生效。

|  |
| --- |
| const behavior = require('../../myBehavior/myBehavior.js');  与data平级  behaviors: [behavior], |

5、behavior 中所有可用的节点

|  |  |  |  |
| --- | --- | --- | --- |
| 可用的节点 | 类型 | 是否必填 | 描述 |
| **properties** | Object Map | 否 | 同组件的属性 |
| **data** | Object | 否 | 同组件的数据 |
| **methods** | Object | 否 | 同自定义组件的方法 |
| **behaviors** | String Array | 否 | 引入其它的 behavior |
| created | Function | 否 | 生命周期函数 |
| attached | Function | 否 | 生命周期函数 |
| ready | Function | 否 | 生命周期函数 |
| moved | Function | 否 | 生命周期函数 |
| detached | Function | 否 | 生命周期函数 |

6、同名字段的覆盖和组合规则\*

组件和它引用的 behavior 中可以**包含同名的字段**，此时可以参考如下 3 种同名时的处理规则：

（1）同名的数据字段 (**data**)

（2）同名的属性 (**properties**) 或方法 (**methods**)

（3）同名的**生命周期函数**

关于详细的覆盖和组合规则（会根据优先级进行合并），大家可以参考微信小程序官方文档给出的说明：

https://developers.weixin.qq.com/miniprogram/dev/framework/custom-component/behaviors.html

三十、使用 npm 包

1、小程序对 npm 的支持与限制

目前，小程序中已经支持使用 npm 安装第三方包，从而来提高小程序的开发效率。但是，在小程序中使用npm 包有如下 3 个限制：

（1）不支持依赖于 **Node.js 内置库**的包（如需要path、fs的包）

（2）不支持依赖于**浏览器内置对象**的包（jQuery，它依赖window）

（3）不支持依赖于 **C++ 插件**的包（加密的包）

总结：虽然 npm 上的包有千千万，但是能供小程序使用的包却“为数不多”。

三十一、使用 npm 包 - Vant Weapp

1、什么是 Vant Weapp

Vant Weapp 是有赞前端团队开源的一套**小程序 UI 组件库**，助力开发者快速搭建小程序应用。它所使用的是**MIT 开源许可协议**，对商业使用比较友好。

官方文档地址 https://youzan.github.io/vant-weapp

2、安装 Vant 组件库

在小程序项目中，安装 Vant 组件库主要分为如下步骤：

（1）在项目使用npm init -y先初始化package.json，然后才能装包

（2）通过 npm 安装

npm i @vant/weapp -S --production

（3）修改 project.config.json（这一步没有搞也行）

开发者工具创建的项目，miniprogramRoot 默认为 miniprogram，package.json 在其外部，npm 构建无法正常工作。

需要手动在 project.config.json 内添加如下配置，使开发者工具可以正确索引到 npm 依赖的位置。

{

...

"setting": {

...

"packNpmManually": true,

"packNpmRelationList": [

{

"packageJsonPath": "./package.json",

"miniprogramNpmDistDir": "./miniprogram/"

}

]

}

}

注意： 由于目前新版开发者工具创建的小程序目录文件结构问题，npm构建的文件目录为miniprogram\_npm，并且开发工具会默认在当前目录下创建miniprogram\_npm的文件名，所以新版本的miniprogramNpmDistDir配置为'./'即可

（4）修改 app.json

将 app.json 中的 "style": "v2" 去除，小程序的新版基础组件强行加上了许多样式，难以覆盖，不关闭将造成部分组件样式混乱。

（5）构建npm包

打开微信开发者工具，点击 工具 -> 构建 npm，并勾选 使用 npm 模块 选项，构建完成后，即可引入组件。



详细的操作步骤，大家可以参考 Vant 官方提供的快速上手教程：

https://youzan.github.io/vant-weapp/#/quickstart#an-zhuang

3、使用 Vant 组件

安装完 Vant 组件库之后，可以在 app.json 的 usingComponents 节点中引入需要的组件，即可在 wxml 中直接使用组件。

在app.json或index.json中引入组件，详细介绍见快速上手。

"usingComponents": {

"van-button": "@vant/weapp/button/index"

}

<van-button type="primary">按钮</van-button>

4、定制全局主题样式

Vant Weapp 使用 **CSS 变量**来实现定制主题。 关于 CSS 变量的基本用法，请参考 MDN 文档：

https://developer.mozilla.org/zh-CN/docs/Web/CSS/Using\_CSS\_custom\_properties

举例子：同样一个颜色值在成千上百个地方被使用到，如果这个值发生了变化，需要全局搜索并且一个一个替换。这样很容易出错。

Css自定义属性在某个地方存储一个值，然后在其他许多地方去引用它。

（1）基本用法

声明一个自定义的属性，属性名需要以**两个减号（--）**开始，属性值则可以是任何有效的css值。 名字大小写是敏感的。它也有作用域的，在某个类中或者标签中，最好定义在根节点中，这样大家都能访问到。

使用一个局部变量时用**var()**，函数包裹以示一个合法的属性值

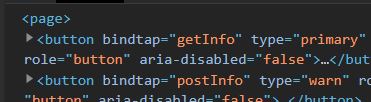
color: var(--xxxxx，red);

后边当变量的值不存在的时候，用red来给他进行代替

在 app.wxss 中，写入对应的 CSS 变量，即可对全局生效：所有可用的颜色变量。

（2）page节点

在调试器的wsml中，page是根节点，每个页面都是



（3）怎么知道vant对应的css变量名称，可以去官方文档中，定制主题，看配置文件

|  |
| --- |
| page {  --button-danger-background-color: #ccc;  } |

请参考 Vant 官方提供的配置文件：

https://github.com/youzan/vant-weapp/blob/dev/packages/common/style/var.less

三十二、使用 npm 包 - API Promise化

1、基于回调函数的异步 API 的缺点

默认情况下，小程序官方提供的异步 API 都是基于**回调函数**实现的，例如，网络请求的 API 需要按照如下的方式调用：



缺点：容易造成**回调地狱**的问题，代码的可读性、 **维护性差**！

2、什么是 API Promise 化

API Promise化，指的是通过**额外的配置**，将官方提供的、基于回调函数的异步 API， 升级**改造为基于Promise 的异步 API**，从而提高代码的可读性、维护性，避免回调地狱的问题。

3、实现 API Promise 化

在小程序中，实现 API Promise 化主要依赖于 **miniprogram-api-promise** 这个第三方的 npm 包。它的安装和使用步骤如下：

npm install miniprogram-api-promise --save

（1）重新构建npm

每次装完包后，先去PC文件夹删除原来的miniprogram\_npm，要**重新构建npm**，小程序读取不到node\_modules，通过构建把包迁移到，miniprogram\_npm中。

4、调用 Promise 化之后的异步 API

（1）先导入 promisifyAll，用es6语法

（2）在小程序的入口文件app.js中，调用一次 promisifyAll()，即可实现异步 API的 Promise化。

参数一：wx

参数二：把promise化的方法挂载到这个空对象上。

（3）在页面的 .js中 就能使用async、await了

（4）注意：不能在写上回调函数 success了。

|  |
| --- |
| app.js头部  // 导入promise化  import { promisifyAll } from 'miniprogram-api-promise';  // 直接把promise化的API挂载到wx对象的一个属性上  const wxp = wx.p = {}  页面.js  async testPromise() {  const {data: res} = await wx.p.request({  url: 'https://www.escook.cn/api/get',  data: {  name: 'Tom',  age: 23,  },  method: 'GET',  });  console.log(res);  }, |

三十三、全局数据共享

1、什么是全局数据共享

**全局数据共享**（又叫做：状态管理）是为了解决**组件之间数据共享**的问题。开发中常用的全局数据共享方案有： Vuex、 Redux、 MobX 等。

2、小程序中的全局数据共享方案

在小程序中，可使用 **mobx-miniprogram** 配合 **mobx-miniprogram-bindings** 这两个包实现全局数据共享。其中：

（1）mobx-miniprogram 用来**创建 Store** 实例对象。

（2）mobx-miniprogram-bindings 用来把 **Store 中的共享数据或方法，绑定到组件或页面中**使用。

3、安装 MobX 相关的包

在项目中运行如下的命令，安装 MobX 相关的包：

npm install mobx-miniprogram --save

npm install mobx-miniprogram-bindings --save

注意： MobX 相关的包安装完毕之后，**记得删除 miniprogram\_npm 目录**后，**重新构建 npm**。

4、创建 MobX 的 Store 实例

（1）先导入包

observable, action

（2）设置计算属性，get修饰符

代表当前的这个计算属性值是只读的，无法为它重新赋值，它会自动计算

（3）action方法，用来修改store中的数据，只允许外界调用这些方法来修改数据，不能直接修改

|  |
| --- |
| import {observale, action} from 'mobx-miniprogram';  export const store = observable({  num1: 1,  num2: 2,  get sum() {  return this.num1 + this.num2;//不要忘了this  },  updateNum1: action(function (step) {  this.num1 += step;  }),  updateNum2: action(function (step) {  this.num2 += step;  }),  }); |

5、将 Store 中的成员绑定到页面中，并使用

（1）需要导入 mobx-miniprogram-bindings 和store.js

1. 在加载页面onload，进行绑定工作，把数据和方法挂载到当前页面对象上，还需要把createStoreBindings函数的返回值，用对象的一个属性来接收，方便卸载的时候，进行解绑操作

store数据源、fields []哪些字段绑定到当前页面？？？待了解清楚、actions []

（3）在卸载页面的时候unload，进行解绑工作

|  |
| --- |
| import { createStoreBindings } from 'mobx-miniprogram-bindings';  import { store } from '../../store/store.js';  onLoad: function (options) {  // this.getInfo();  // this.postInfo();    // 进行的是跟store方法进行绑定到当前页面对象上，返回值是在解绑有用到，  this.stroeBindings = createStoreBindings(this, {  store,  fields: ['num1', 'num2', 'sum'],  actions: ['updateNum1','updateNum2'],  });  console.log(this.stroeBindings);  },  onUnload: function () {  this.stroeBindings.destroyStoreBindings();  },  changeNum1(e) {  // console.log(e);  this.updateNum1(e.target.dataset.step - 0);  },  <view> {{num1}} + {{num2}} = {{sum}} </view>  <van-button bindtap="changeNum1" data-step="{{ 1 }}">num1 + 1</van-button>  <van-button bindtap="changeNum2" data-step="-1">num2 - 1</van-button>  注意：data-step如果没有使用{{}}，它传递的是一个字符串数字，需要进行转换，如果使用{{}}，那么就不用进行转换。 |

6、将 Store 中的成员绑定到组件中，并使用

（1）需要导入 storeBindingsBehavior 和store.js

（2）提供behaviors数组，通过storeBindingsBehavior 实现自动绑定。

1. 声明一个配置对象，storeBindings，在这个对象里边接收三个数据属性，分别是store、fileds（要绑定的字段）、actions。

为什么要搞一个对象？？？这是组件的属性？？

fields: {

num1: () => store.num1 // 绑定字段的第一种方式

num2: (store) => store.num2 //绑定字段的第二种方式

sum: ‘sum’ //绑定字段的第三种方式

}

|  |
| --- |
| import { storeBindingsBehavior } from 'mobx-miniprogram-bindings'  import { store } from '../../store/store.js';  Component({  behaviors: [storeBindingsBehavior],    storeBindings: {  store,  fields: {  // 采用简单的方式获取  num1: 'num1',  num2: 'num2',  sum: 'sum',  },  actions: {  updateNum1: 'updateNum1',  }  },  \* 组件的方法列表  \*/  methods: {  changeNum1(e) {  this.updateNum1(e.target.dataset.step);  }  }  })  组件wxml  <view> {{num1}} + {{num2}} = {{sum}} </view>  <van-button bindtap="changeNum1" data-step="{{ 1 }}" type="primary">num1 + 1</van-button>  <van-button bindtap="changeNum2" data-step="-1" type="danger">num2 - 1</van-button> |

7、延迟更新与立刻更新

为了提升性能，在 store 中的字段被更新后，并不会立刻同步更新到 this.data 上，而是等到下个 wx.nextTick 调用时才更新。（这样可以显著减少 setData 的调用次数。）

如果需要立刻更新，可以调用：

this.updateStoreBindings() （在 behavior 绑定 中）

this.storeBindings.updateStoreBindings() （在 手工绑定 中）

三十四、分包 - 基础概念

1、什么是分包

分包指的是把一个**完整的小程序项目** ，按照需求**划分为不同的子包**，在构建时打包成不同的分包，用户在使用时**按需进行加载**。

2、分包的好处

对小程序进行分包的好处主要有以下两点：

⚫ 可以优化小程序首次启动的下载时间

⚫ 在多团队共同开发时可以更好的解耦协作

3、分包前项目的构成

分包前，小程序项目中**所有的页面**和**资源**都被打包到了一起，导致整个**项目体积过大**， 影响小程序首次启动的下载时间。

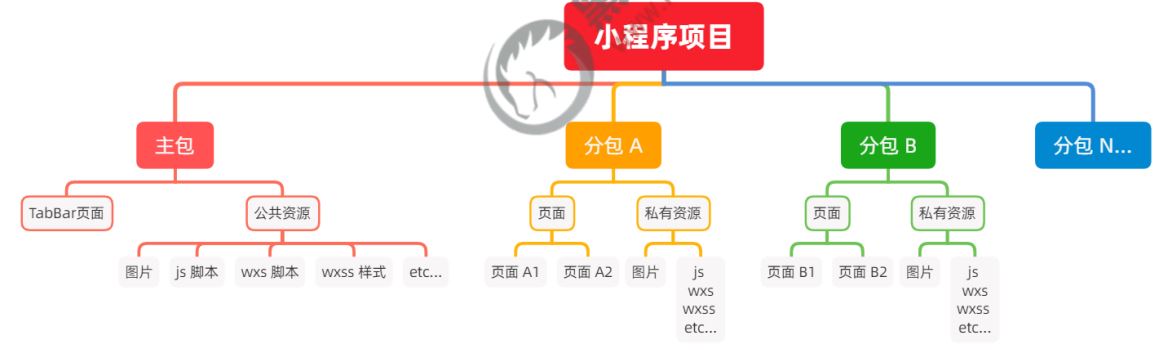


4、分包后项目的构成

分包后，小程序项目由 **1 个主包** + **多个分包**组成：

（1）主包：一般只包含项目的**启动页面**或 **TabBar 页面**、以及所有分包都需要用到的一些**公共资源（分包可以访问公共资源）**

（2）分包：只包含和当前分包有关的页面和私有资源（自己顾自己的）



5、分包的加载规则

（1） 在小程序启动时，默认会下载**主包**并**启动主包内页面**

⚫ tabBar 页面需要放到主包中，只能放到主包

（2）当用户进入分包内某个页面时，**客户端会把对应分包下载下来**，下载完成后再进行展示

⚫ 非 tabBar 页面可以按照功能的不同，划分为不同的分包之后，进行按需下载

6、分包的体积限制

目前，小程序分包的大小有以下两个限制：超过了，会发布失败

⚫ 整个小程序所有分包大小不超过 **16M**（主包 + 所有分包）。

⚫ 单个**分包/主包**大小不能超过 **2M**。

三十五、分包 - 使用分包

1、配置方法

在app.json中添加分包的配置项 subpackages ，保存之后，微信开发工具就会自动帮我们创建分包的目录结构。

（1）root属性

小程序会按 subpackages 的配置进行分包，在数组中有几个分包就有几个对象，在对象里边必须要有一个root属性，来指定分包的根目录。

（2）pages属性--数组

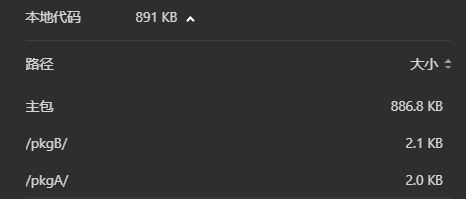
在pages数组中，你有几个页面就放几个页面的路径就可以，注意页面的路径都是相对root根目录来进行存放的。

（3）name属性

添加name，是分包的别名。

（4）查看分包的体积

详情---基本信息---本地代码---就能看到所有的分包



2. 打包原则

（1）小程序会按 **subpackages** 的配置进行分包，subpackages **之外的目录**将被打包到主包中。

（2）主包也可以有自己的 pages（即最外层的 pages 字段）

（3）**tabBar 页面必须在主包内**

（4）分包之间**不能互相嵌套**

3、引用原则

（1）主包无法引用分包内的私有资源

（2）分包之间不能相互引用私有资源

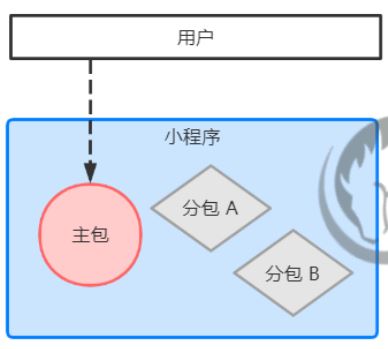
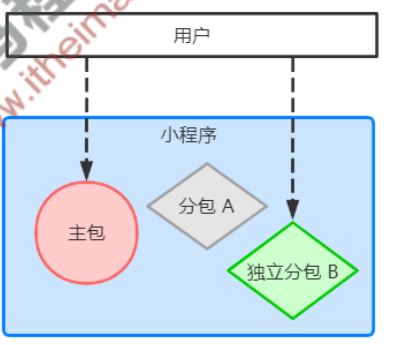
（3）**分包可以引用主包内的公共资源**

三十六、分包 - 独立分包，咋用？？？

1、什么是独立分包

独立分包**本质上也是分包**，只不过它比较特殊， **可以**独立于主包和其他分包而**单独运行。**

默认从主包打开其他的分包。有时候也能通过分包直接打开小程序，此时就需要独立分包。

2、独立分包和普通分包的区别

最主要的区别：**是否依赖于主包才能运行**

普通分包**必须依赖**于主包才能运行

独立分包可以在不下载主包的情况下，**独立运行**

3、独立分包的应用场景

开发者可以按需，将某些**具有一定功能独立性**的页面配置到独立分包中。原因如下：

（1）当小程序从普通的分包页面启动时，需要首先下载主包

（2）而独立分包**不依赖主包**即可运行， 可以很大程度上提升分包页面的启动速度

注意：一个小程序中可以有多个独立分包。

4、独立分包的配置方法independent

普通分包跟独立分包的区别，就在于它有没有independent这个节点，跟root同级，值为true。



5、引用原则

独立分包和普通分包以及主包之间，是**相互隔绝的**，不能相互引用彼此的资源！例如：

（1）主包无法引用独立分包内的私有资源

（2）独立分包之间， 不能相互引用私有资源

（3）独立分包和普通分包之间， 不能相互引用私有资源

（4）**特别注意：**独立分包中不能引用主包内的公共资源

三十七、分包 - 分包预下载

1、什么是分包预下载

分包预下载指的是：在进入小程序的某个页面时， 由**框架自动预下载可能需要的分包**，从而提升进入后续分包页面时的启动速度。

2、配置分包的预下载 -- preloadRule

预下载分包的行为，会在**进入指定的页面时触发**。在 app.json 中，使用 **preloadRule** 节点定义分包的预下载规则，示例代码如下：



|  |
| --- |
| "subpackages": [  {  "root": "pkgA",  "name": "A",  "pages": [  "pages/cat",  "pages/dog"  ]  },  {  "root": "pkgB",  "name": "B",  "pages": [  "pages/apple",  "pages/orange"  ]  }  ],  "preloadRule": {  "pages/concat/concat": {进入这个页面的时候，触发预下载  "packages": ["pkgA"],  "network": "all"  }  }, |

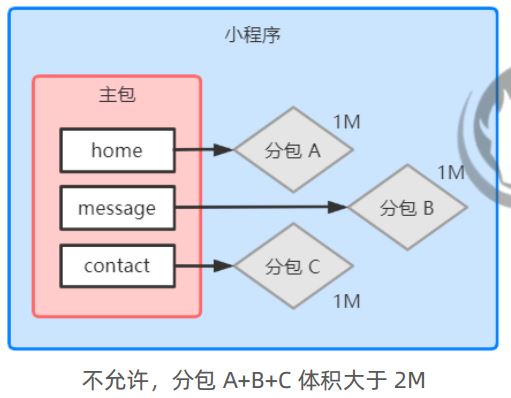
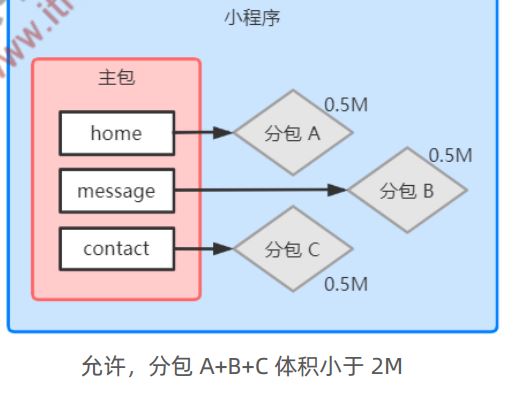
1

设置网络为WiFi，手机为3g模式下



3、分包预下载的限制

**同一个分包中的页面**享有共同的预**下载大小限额 2M**，例如：

主包中有三个页面home，message，contact，它们各自预下载分包1M，这样的解释有点问题，分包之间不是不能嵌套吗？？?

理解成主包/分包中有多个页面，它们的大小限额为2M。

三十八、案例 - 自定义tabbar（是组件）

在自定义 tabBar 模式下

为了保证低版本兼容以及区分哪些页面是 tab 页，tabBar 的相关配置项需完整声明，但这些字段不会作用于自定义 tabBar 的渲染。

与 tabBar 样式相关的接口，如 wx.setTabBarItem 等将失效。

**每个 tab 页下的自定义 tabBar 组件实例是不同的，**可通过自定义组件下的 getTabBar 接口，获取当前页面的自定义 tabBar 组件实例。

1、**自定义tabbar**分为3大步骤，分别是

（1）配置信息

在 app.json 中的 tabBar 项指定 custom true 字段，同时其余 tabBar 相关配置也补充完整。

所有 tab 页的 json 里需声明 usingComponents 项，也可以在 app.json 全局开启。

（2）添加tabbar代码文件

它的目录名字是custom-tab-bar是固定的，里边的名字叫index组件

在代码根目录下添加入口文件:

custom-tab-bar/index.js

custom-tab-bar/index.json

custom-tab-bar/index.wxml

custom-tab-bar/index.wxss

（3）编写tabbar代码

List节点要保存下来，它兼容低版本的，低版本不支持自定义tabbar时候，它可以用来显示。

只要在 tabbar页面写了custom:true，以及添加custom-tab-bar代码文件夹，小程序会自动识别，把自定义tabbr组件渲染上去。

2、vant相关知识点

Vant一些组件需要联网使用的，下载字体图标。Vant支持自定义图标。

修改vant组件样式值的时候，要注意styleIsolation，的配置，在js中options，因为它是组件，类样式是隔离的。

徽标超出灰色区域，通过F12看间距都是谁的，看能不能去除。

三元运算符的使用。0是没有进行渲染的必要，使用空字符，这里使用标志位，true的话，会一直渲染。

tabbar页面的切换使用switchTab API，不是页面的切换。

3、存在的疑问

（1）点击tabbar页面会创建一个tabbar的实例出来，存在内存中，这样它每个页面都有自己的active数据，所以会导致切换的时候，效果有问题，点击时能看到正常的现象。

**自定义tabbar，实例会被创建两次，暂时未解**，微信讨论社区有相关问题，怀疑是栈错乱，这样可能跟最后那个图标切换问题跟这个有关系。

目前问题是出在active，可以确定的每次会创建一个新的tabbar实例，data中的active也是全新的。

解决就把active扔到mobx中去，这样它跟tabbar实例就没有关系了

（2）延迟更新，在mobx部分有说明，可以手动更新，不然就要等

注意，在修改mobx中的数据时候，它是需要消耗一定的时间，如果你修改后，马上要使用该值，需要一点延迟时间。下边这种处理方式是不靠谱的，要用官方提供的API会好点。

storeBindings: {

store,

fields: {

sum: 'sum',

active: 'avtiveIndex',

},

actions: {

updateActive: "updateActive",

}

},

onChange(event) {

this.updateActive(event.detail);

// 如果要使用this.data.active来跳转，就需要加延时，否则还没来得及更新

// 必须使用箭头函数，因为使用了this。不能改变它的指向

setTimeout( () => {

wx.switchTab({

url: this.data.list[this.data.active].pagePath,

})

}, 10);

// 使用event.detail就不用加延时

}

三十九、补充的知识点

1、逻辑层 App Service

小程序开发框架的逻辑层使用 **JavaScript 引擎**为小程序提供开发者 **JavaScript 代码**的运行环境以及微信小程序的特有功能。

逻辑层将数据进行处理后发送给视图层，同时接受视图层的事件反馈。

开发者写的所有代码最终将会打包成一份 JavaScript 文件，并在小程序启动的时候运行，直到小程序销毁。这一行为类似 ServiceWorker，所以逻辑层也称之为 App Service。

2、App实例，getApp

整个小程序只有一个 App 实例，是全部页面共享的。开发者可以通过 **getApp** 方法获取到全局唯一的 App 实例，获取App上的数据或调用开发者注册在 App 上的函数。

// xxx.js

const appInstance = getApp()

console.log(appInstance.globalData) // I am global data

3、在自定义组件中触发双向绑定更新

自定义组件还可以自己触发双向绑定更新，做法就是：使用 setData 设置自身的属性。例如：

// custom-component.js

Component({

properties: {

myValue: String

},

methods: {

update: function() {

// 更新 myValue

this.setData({

myValue: 'leaf'

})

}

}

})

后退为什么 0 也能退，负数也行？？

答：返回的页面数，如果 delta 大于现有页面数，则返回到首页。

container类名怎么居中了。

答：在app.wxss全局配置文件有初始化了

data中的数据不能自增吗 a++？

[...,...]这样能进行拼接？？可以

小程序中的页面也一直是存在内存中吗，单独的页面，和tabbar页面似乎不一样。待确认

Tabbar的生命周期是怎么样的，

Flex image跟文字排一起为什么image被挤了，加上盒子就不会，答：在css中可以用别的flex-shrink解决

为什么behavior用的是require，而stroe使用import。

答：这两者官方都是支持的