微信小程序

1、小程序 -- 起步

1.1 小程序简介

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**中，**小程序**运行在**微信环境**中
2. API不同
    - 小程序无法调用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
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

体验小程序

注册小程序开发账号

1. 打开网站： https://mp.weixin.qq.com/ ，点击 立即注册
2. 选择注册账号的类型为 小程序
3. 填写账号信息
4. 提示邮箱激活
5. 点击链接激活账号
6. 选择主体类型 个人，并完成主体信息登记
7. 获取小程序的AppID，点击 开发管理 - 开发设置 - 获取开发者ID

安装开发者工具

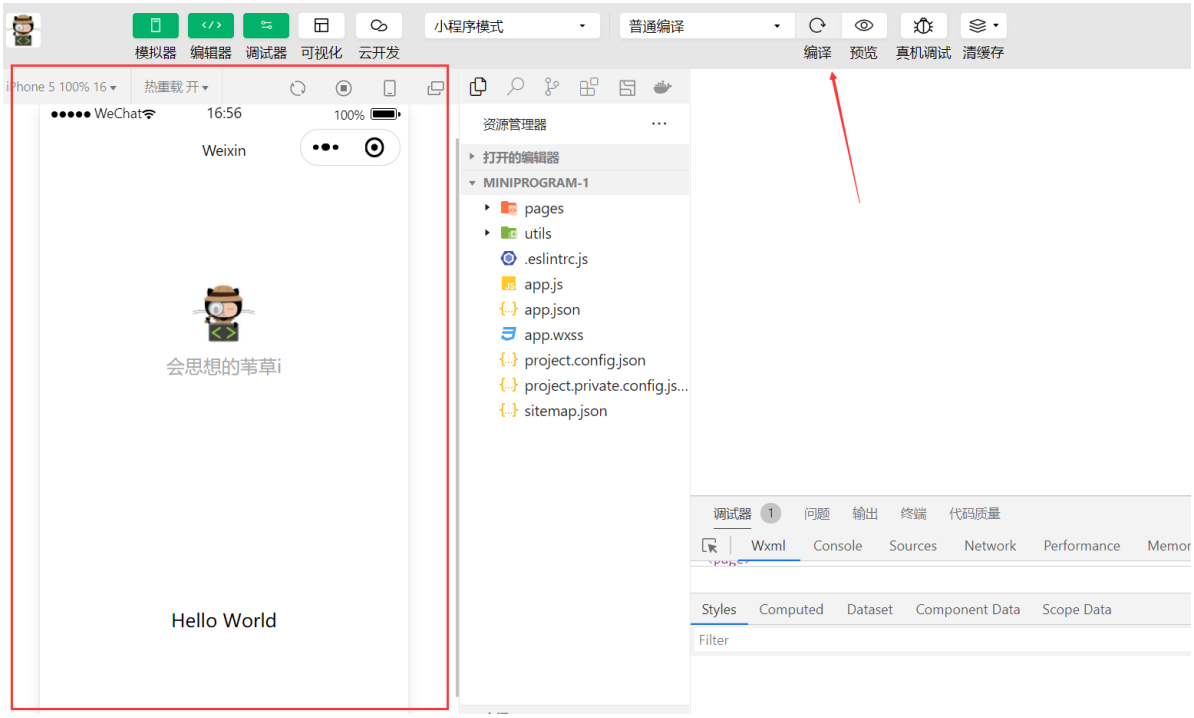
- 微信开发者工具是官方推荐使用的小程序开发工具，它提供的主要功能如下:
    - 快速创建小程序项目
    - 代码的查看和编辑
    - 对小程序功能进行调试
    - 小程序的预览和发布
- 步骤:

1. 打开网站 微信官方文档--下载 下载即可
2. 下载好后安装
3. 手机微信扫码登录
4. 设置外观和代理(不使用代理)

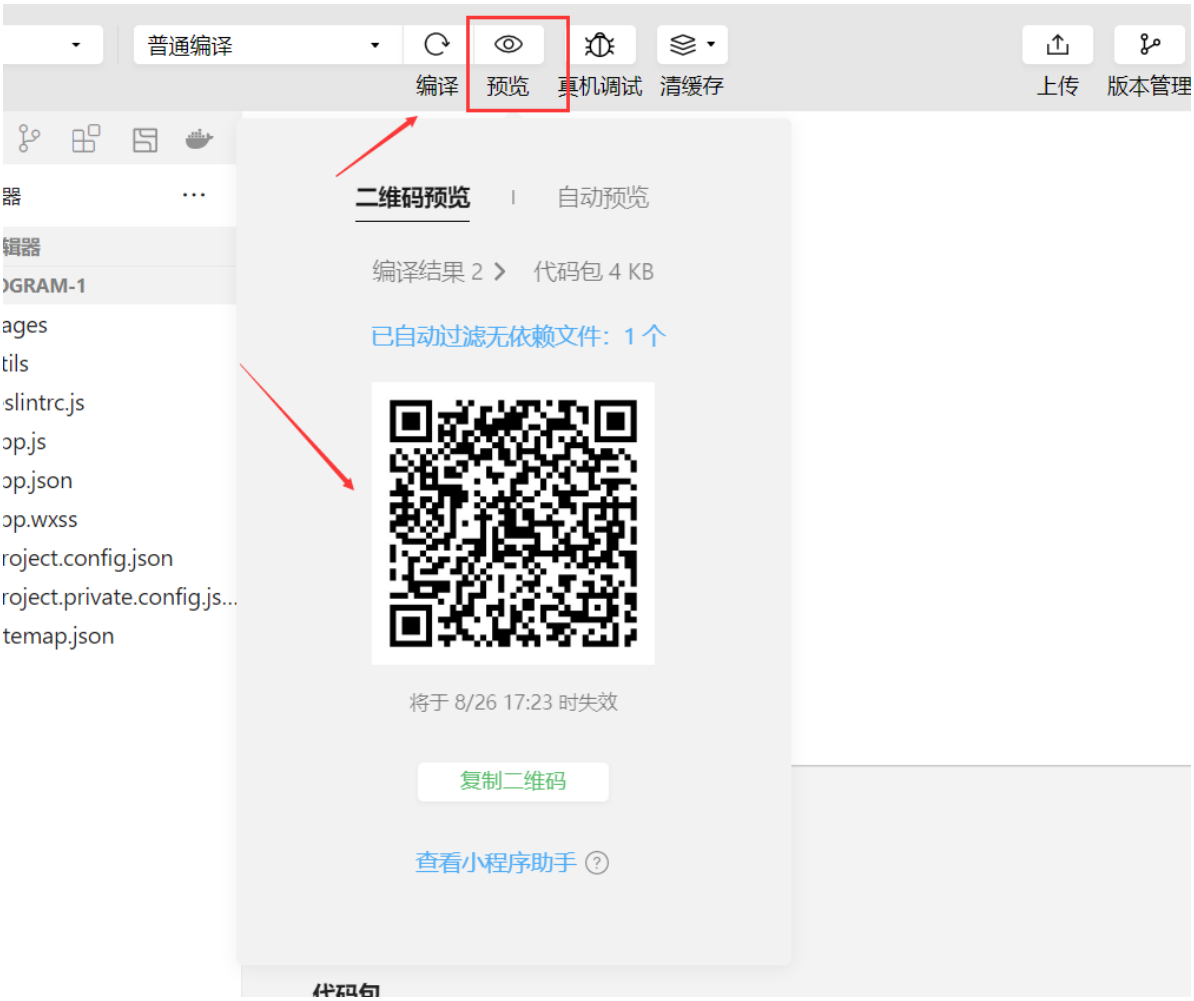
创建小程序项目

1. 点击"加号"按钮
2. 填写项目信息

3. 项目创建完成
4. 在模拟器上查看项目效果

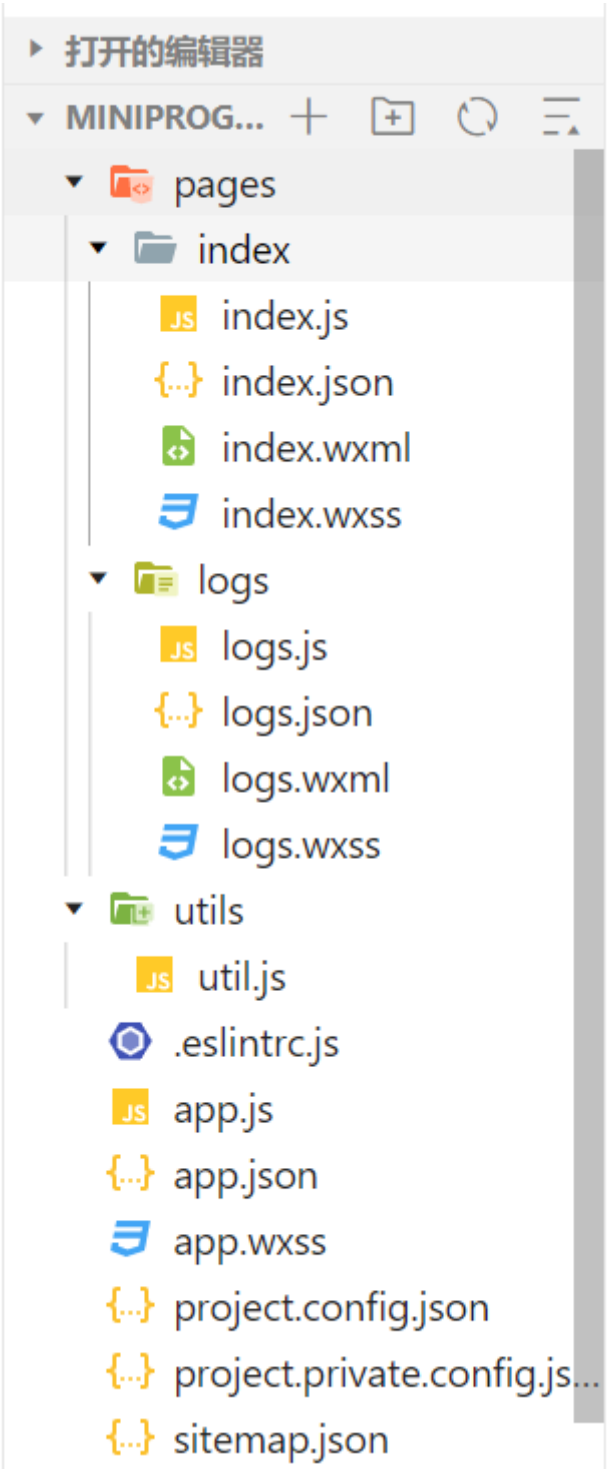


5. 在真机上查看项目效果



1.2 小程序构成

基本组成结构



- pages用来存放所有小程序的页面
  - .js文件(页面的脚本文件，存放页面的数据、事件处理函数等)
  - .json文件(当前页面的配置文件，配置窗口的外观、表现等)，会覆盖其他配置项
  - .wxml文件(页面的模板结构文件)
  - .wxss文件(当前页面的样式表文件)
- 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)
- app.js小程序项目的入口文件
- app.json小程序项目的全局配置文件，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
  - pages:用来记录当前小程序所有页面的路径
  - window:全局定义小程序所有页面的背景色、文字颜色等
  - style:全局定义小程序组件所使用的样式版本
  - sitemapLocation:用来指明sitemap.json的位置

- app.wxss小程序项目的全局样式文件
- projectconfig.json项目的配置文件
    - 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
- sitemap.json用来配置小程序及其页面是否允许被微信索引(类似网页的SEO)
- 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
- 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

WXML模板

- WXML (Weixin Markup Language)是小程序框架设计的一套标签语言，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
- WXSS和HTML的区别：
- **标签名称不同**
- HTML (div,span, img, a)
- WXML (view, text, image, navigator)
- **属性节点不同**
    - < a href="#">超链接< /a>
    - < navigator url="/pages/home/home" > < /navigator>
- **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

WXSS样式

- 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
- WXSS和CSS的区别：
- **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
- **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
- **WSSS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

JS逻辑交互

- 通过.js文件来处理用户的操作
- 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
    - 是普通的功能模块文件,用来封装公共的函教或属性供页面使用

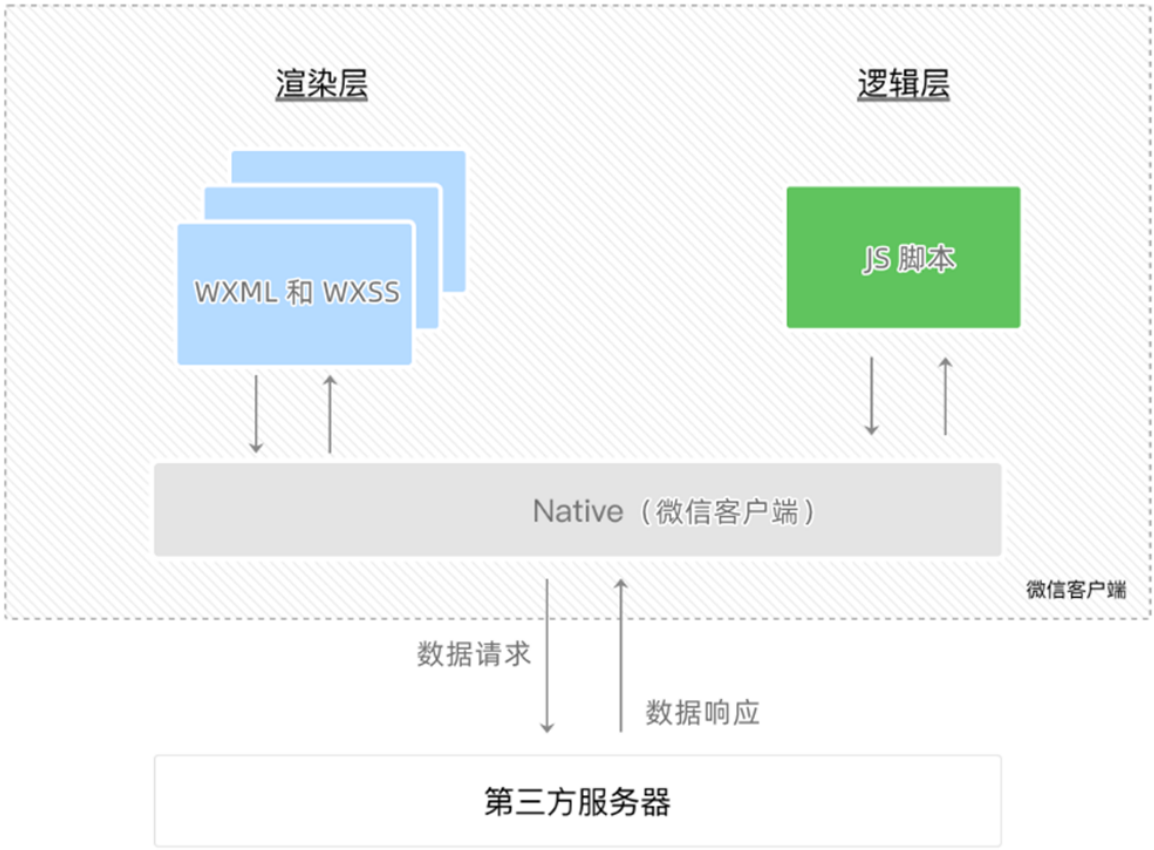
1.3 小程序的宿主环境

宿主环境简介

- 宿主环境(host environment)指的是程序运行所必须的依赖环境，如Android和iOS是两个不同的宿主环境
- 手机微信是小程序的宿主环境
- 小程序宿主环境包含的内容：
    - 通信模型
    - 运行机制
    - 组件
    - API

通信模型

- 小程序中通信的主体是渲染层和逻辑层，其中:
    - WXML模板和WXSS 样式工作在渲染层
    - JS脚本工作在逻辑层
    - 渲染层和逻辑层之间的通信 -- 由微信客户端进行转发
    - 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



运行机制

- 小程序启动过程：

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

- 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

组件

- 组件分类：

1. 视图容器
    - 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

- scroll-view
  - 可滚动的视图区域
  - 常用来实现滚动列表效果
- swiper和 swiper-item
  - 轮播囤容器组件和轮播图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	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

2. 基础内容
   - text
     - 文本组件
     - 类似于**HTML中的span标签**，是一个行内元素
   - rich-text
     - 富文本组件
     - 支持把HTML字符串渲染为WXML结构
   - 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)
   - image
     - 图片组件
     - image 组件默认宽度约300px、高度约240px

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。
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，高度自动变化**，保持原图宽高比不变
heightFix	缩放模式，**高度不变，宽度自动变化**，保持原图宽高比不变

   - navigator

    - 页面导航组件
    - 类似于**HTML中的a链接**
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

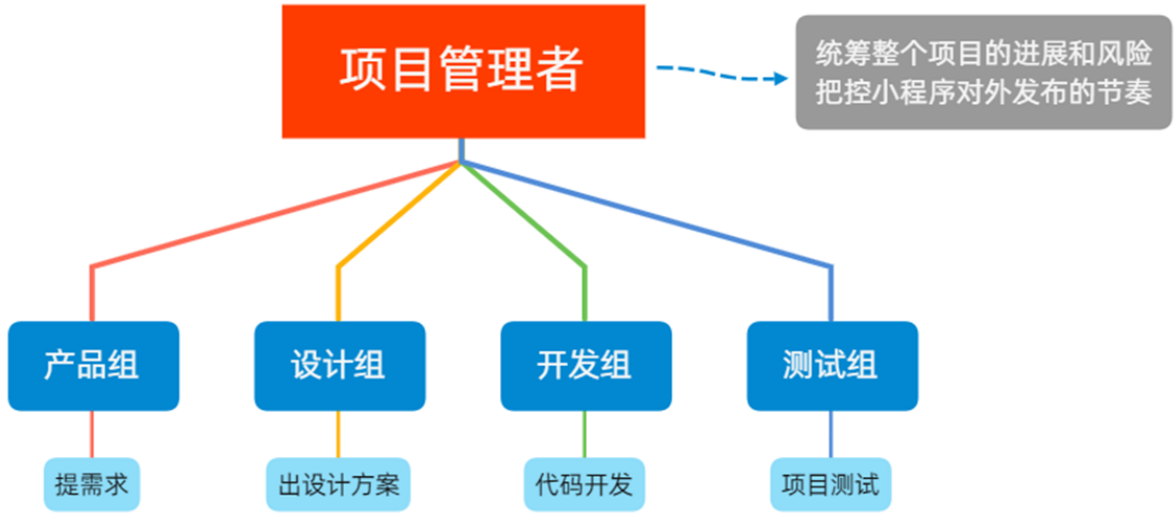
API

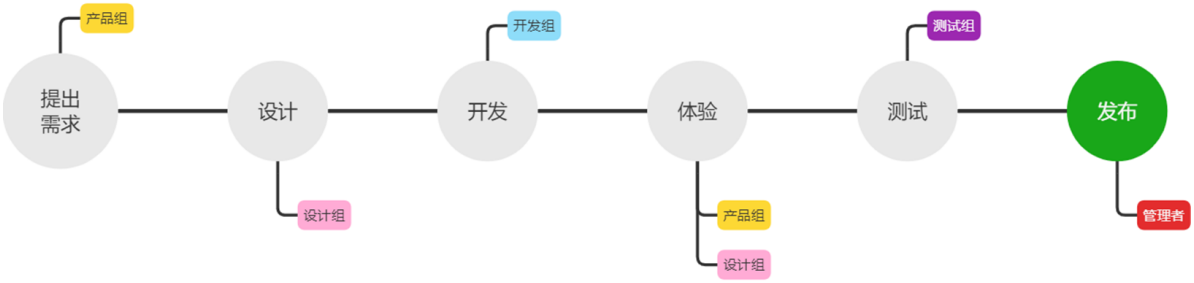
- 事件监听API
    - 特点:以**on开头**，用来监听某些事件的触发
    - 举例:wx.onWindowResize(function callback)监听窗口尺寸变化的事件
- 同步API
    - 特点1:以 **Sync结尾**的API都是同步API
    - 特点2:同步API的执行结果，可以通过函数返回值直接获取，如果执行出错会抛出异常
    - 举例: wx.setStorageSync("key' , 'value')向本地存储中写入内容
- 异步API
    - 特点:类似于jQuery中的$.ajax(options)函数，需要通过success、fail、complete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

1.4 协同工作和发布

协同工作

- 出于管理需要，我们需要对不同岗位、不同角色的员工的权限进行边界的划分，使他们能够高效的进行协同工作





- 小程序成员管理体现在**管理员**对小**程序项目成员**及**体验成员**的管理:
  - 项目成员:
    - 表示参与小程序开发、运营的成员可登录小程序管理后台
    - 管理员可以添加、删除项目成员，并设置项目成员的角色
  - 体验成员:
    - 表示参与小程序内测体验的成员
    - 可使用体验版小程序，但不属于项目成员管理员及项目成员均可添加、删除体验成员
- 不同项目成员对应的权限:

权限	运营者	开发者	数据分析者
开发者权限		√	
体验者权限	√	√	√
登录	√	√	√
数据分析			√
微信支付	√		
推广	√		
开发管理	√		
开发设置		√	
暂停服务	√		
解除关联公众号	√		
腾讯云管理		√	
小程序插件	√		
游戏运营管理	√		

- 可以在 微信公众平台-小程序，成员管理中添加项目成员和体验成员

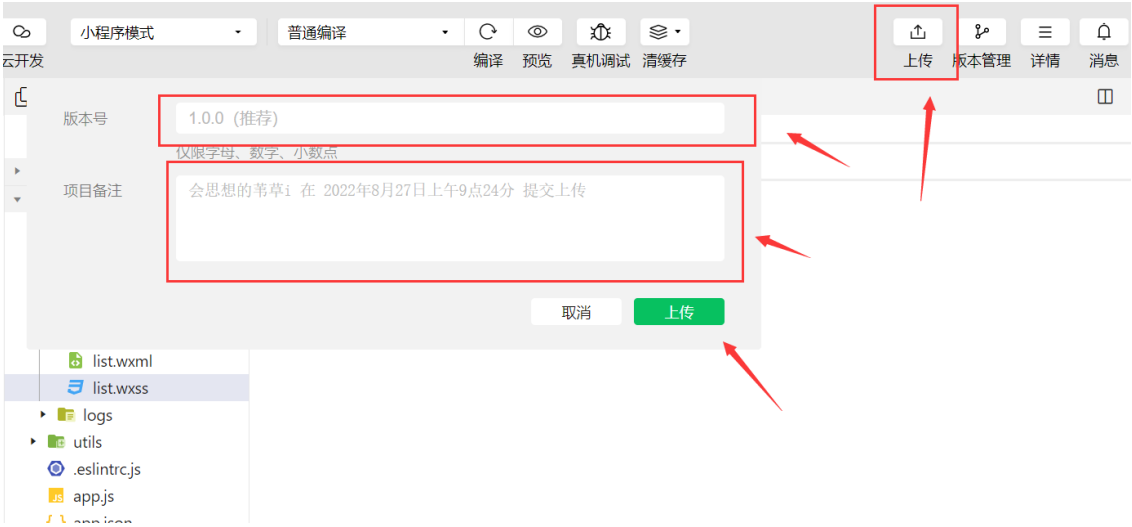
小程序版本

版本阶段	说明
开发版本	使用开发者工具，可将代码上传到开发版本中。 开发版本只保留每人最新的一份上传的代码。 点击提交审核，可将代码提交审核。开发版本可删除，不影响线上版本和审核中版本的代码。
体验版本	可以选择某个开发版本作为体验版，并且选取一份体验版。
审核版本	只能有一份代码处于审核中。有审核结果后可以发布到线上，也可直接重新提交审核，覆盖原审核版本。
线上版本	线上所有用户使用的代码版本，该版本代码在新版本代码发布后被覆盖更新。

发布上线

- 步骤：

1. 上传代码：点击开发者工具的 **上传** 按钮，**填写版本号**及**备注**(查看版本：登录小程序管理后台 -- 管理 -- 版本管理)



2. 提交审核：小程序管理后台版本管理处点击 **提交审核** 即可
3. 发布：审核通过后会收到通知，在审核列表处点击 **发布** 即可

- 获取小程序码 ： 登录小程序管理后台 -> 设置 -> 基本设置 -> 基本信息 -> 小程序码及线下物料下载

运营数据

- 在“小程序后台”查看
    1. 登录小程序管理后台
    2. 点击侧边栏的“统计"
    3. 点击相应的tab可以看到相关的数据
- 使用“小程序数据助手”查看
    1. 打开微信
    2. 搜索“小程序数据助手”
    3. 查看已发布的小程序相关的数据

2、小程序 -- 模板与配置

2.1 WXML模板语法

数据绑定

- 在data中定义数据

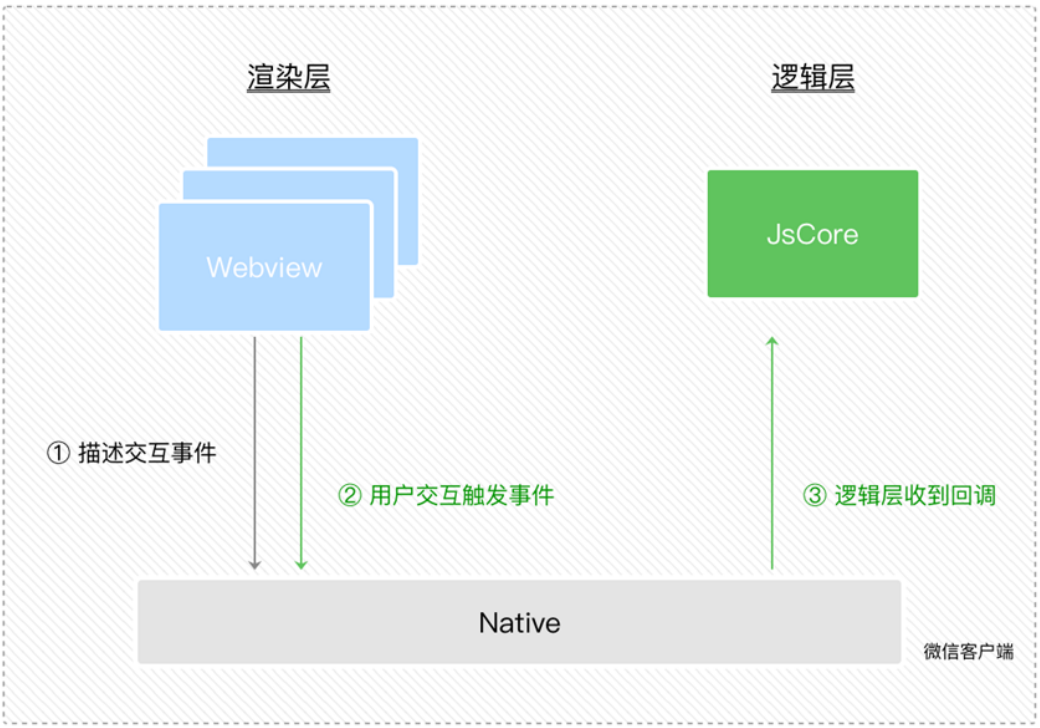
```
Page({
    data:{
        //字符串类型数据
        info:'init data',
        //数组类型数据
        msgList:[{msg:'hello'},{nsf:'world'}]
    }
})
```

- 在WXML中使用数据

```
<view>{{ 要绑定的数据名称 }}</view>
//Mustache语法(双大括号)
//应用场景：绑定内容,绑定属性,运算（三元运算、算术运算等)
```

事件绑定

- 事件是渲染层到逻辑层的通讯方式



- 小程序中常用事件

类型	绑定方式	事件描述
tap	bindtap 或 bind:tap	手指触摸后马上离开，类似于 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

- 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(源头)组件**的一些属性值集合
currentTarget	Object	**当前(正在触发事件)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

- 事件绑定语法格式:

```
//wxml
<button type="primary" bindtap="btnTapHandler">按钮</button>

//js
Page({
  btnTapHandler(e){
    console.log(e)
  },
})
```

- 在事件处理函数中为data中的数据赋值

```
Page({
  data:{
    count:0
  },
  changeCount(){
    this.setData({
      count:this.data.count + 1//修改
    })
  }
})
```

- 事件传参

```
//wxml
<button bindtap = "btnHandler" data-info="{{2}}">事件传参</button>
//通过data-*自定义属性传参

//js
btnHandler(event){
    console.log(event.target.dataset.info)
}
//通过event.target.dataset.参数名获取到具体参数的值
```

- bindinput语法格式
    - 通过input事件来响应文本框的输入事件

```
//wxml
<input bindinput="inputHandler"></input>

//js
Page({
  inputHandler(e){
    console.log(e.detail.value)
  }
})
```

数据同步

- 步骤

1. 定义数据(data数据)
2. 渲染结构(wxml)
3. 美化样式(wxss)
4. 绑定input事件处理函数(js函数)

条件渲染

```
<view wx:if="{{type === 1}}">男</view>
<view wx:elif="{{type === 2}}">女</view>
<view wx:else>保密</view>

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

//hidden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>
```

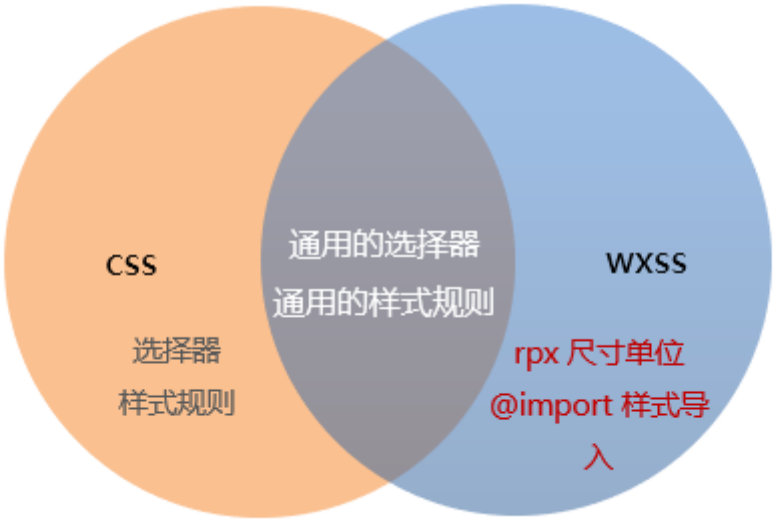
- wx:if 与 hidden 的对比
    1. 运行方式不同
        - wx:if 以动态创建和移除元素的方式，控制元素的展示与隐藏
        - hidden 以切换样式的方式(display: none/block;)，控制元素的显示与隐藏
    2. 使用建议
        - **频繁切换时,建议使用hidden**
        - **控制条件复杂时**，建议使用wx:if 搭配wx:elif、wx:else进行展示与隐藏的切换

列表渲染

```
<view wx:for="{{arr}}">
  索引是：{{index}}, item项：{{item}}
</view>
//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
//也可以手动指定索引和当前项的变量名 wx:for-index="" wx:for-item=""

//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
//wxml
<view wx:for="{{userList}}" wx:key="id">{{item.name}}</view>
//js
Page({
  data:{
    userList:[
      {id:1,name:'小红'},
      {id:2,name:'小黄'},
      {id:3,name:'小白'}
    ]
  },
})
```

2.2 WXSS模板样式



rpx

- rpx (responsive pixel）是微信小程序独有的，用来解决屏适配的尺寸单位
- rpx的实现原理：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

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
iPhone6 Plus	1rpx = 0.552px	1px = 1.81rpx

- 开发中建议采用iPhone 6作为视觉设计稿

样式导入

- 使用WXSS提供的import语法。可以导入外联的样式表

```
@import "demo.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

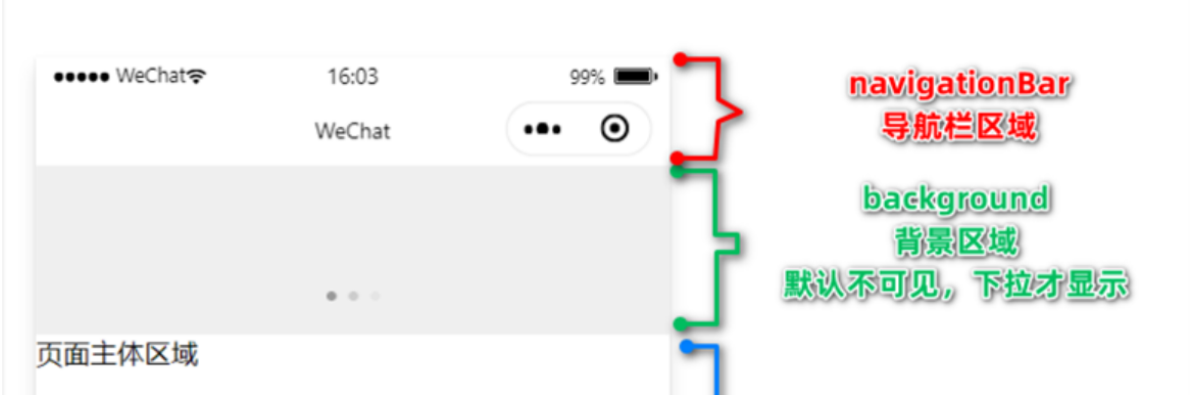
局部样式

- 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:
  - 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**
  - 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

2.3 全局配置

- 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:
  - pages：记录当前小程序所有页面的存放路径
  - window：全局设置小程序窗口的外观
  - tabBar：设置小程序底部的tabBar效果
  - style：是否启用新版的组件样式

window

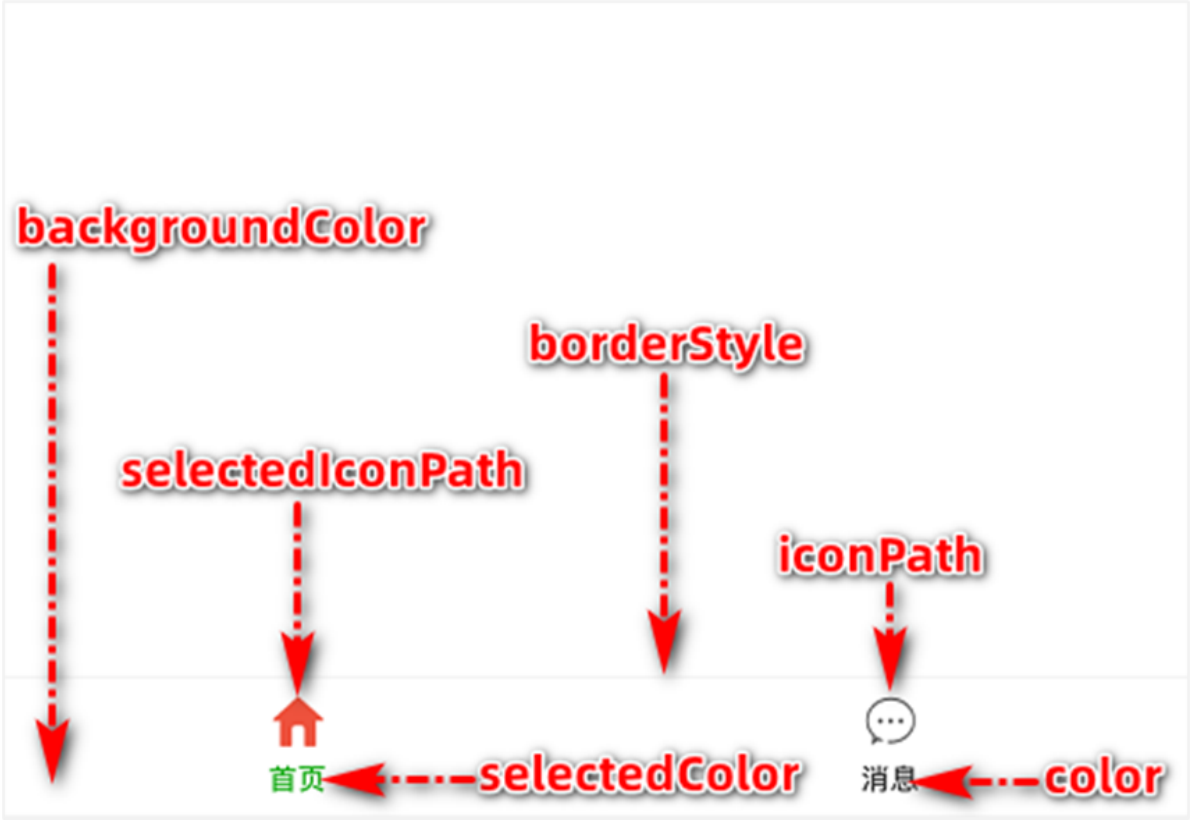


- 常用配置项

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，**仅支持16进制颜色**
navigationBarTextStyle	String	white	导航栏标题颜色，**仅支持 black / white**
backgroundColor	HexColor	#ffffff	窗口的背景色，**仅支持16进制颜色**
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

tabBar

- 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，分为底部和顶部两类
- 只能配置**最少2个，最多5个**tab页签；渲染顶部tabBar时并不显示icon



- tabBar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**		tab 页签的列表， 最少 2 个、最多 5 个 tab

- 每个tab项的配置选项

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
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

2.4 页面配置

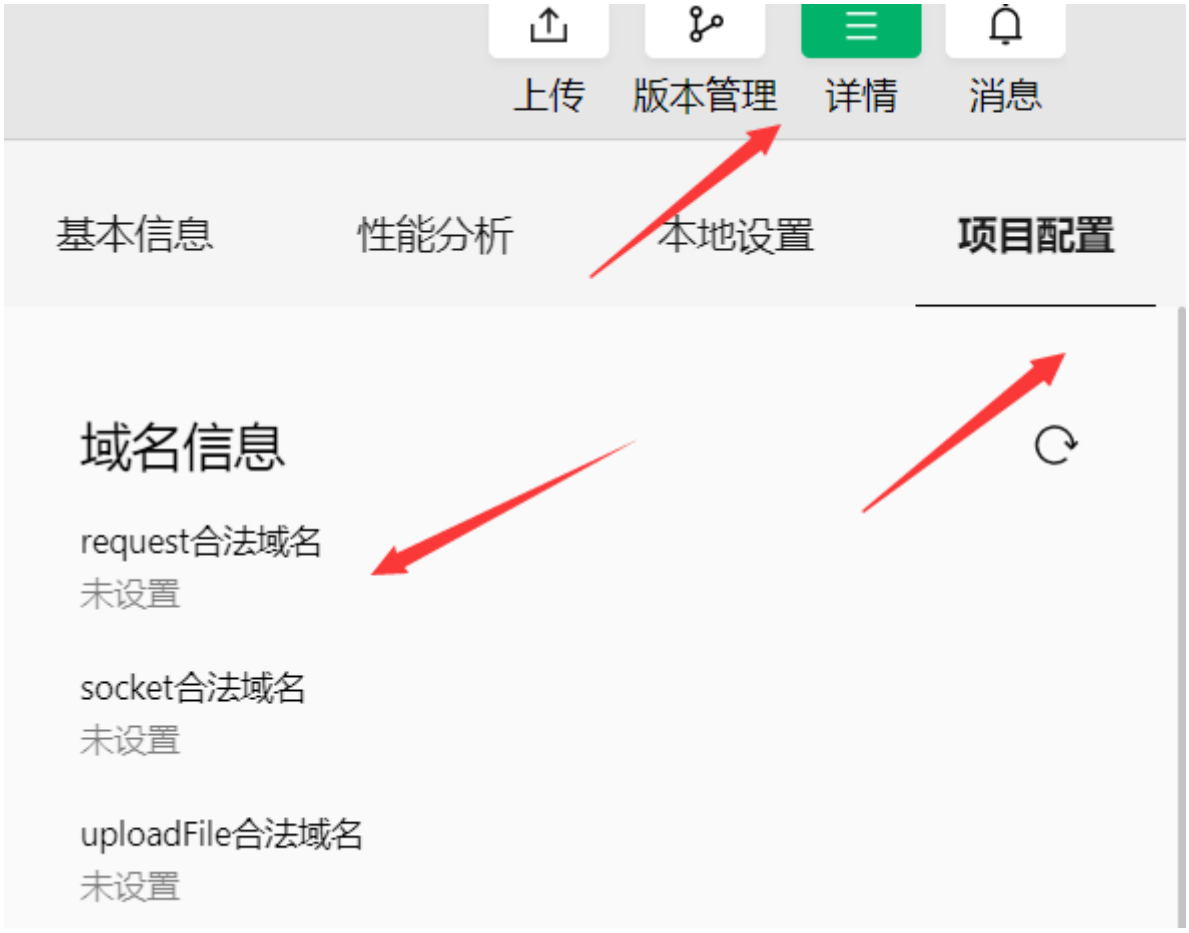
- 小程序中，每个页面都有自己的.json配置文件，用来对当前页面的窗口外观、页面效果等进行配置
- 当页面配置与全局配置冲突时，根据就近原则，最终的效果以页面配置为准
- 常用配置项：(同全局配置项)

属性	类型	默认值	描述
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
navigationBarTitleText	String		当前页面导航栏标题文字内容
backgroundColor	HexColor	#ffffff	当前页面窗口的背景色
backgroundTextStyle	String	dark	当前页面下拉 loading 的样式，仅支持 dark / light
enablePullDownRefresh	Boolean	false	是否为当前页面开启下拉刷新的效果
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

2.5 网络数据请求

- 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:

- 只能请求HTTPS类型的接口
- 必须将接口的域名添加到信任列表中



- 配置域名步骤：**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**
- 注意：
  - 域名只支持https协议
  - 域名不能使用IP地址或localhost
  - 域名必须经过ICP备案
  - 服务器域名一个月内最多可申请5次修改
- 发起GET请求

```
wx.request({
      url: 'https://www.escook.cn/api/get', //请求的接口地址，必须是基于https协议
      method:'GET',  //请求的方式
      data:{         //发送到服务器的数据
        name:'xyk',
        age:21
      },
      success:(res=>{  //请求成功之后的回调函数
        console.log(res)
      })
    })
```

- 发起POST请求

```
wx.request({
      url: 'https://www.escook.cn/api/post', //请求的接口地址，必须是基于https协议
      method:'POST',  //请求的方式
      data:{          //发送到服务器的数据
        name:'xyk',
        age:21
      },
      success:(res)=>{  //请求成功之后的回调函数
        console.log(res)
      }
    })
```

- 页面加载时请求数据

```
//生命周期函数
onLoad(options) {
  this.getInfo()
  this.postInfo()
},
```

- 跳过合法域名检验
    - 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
    - **仅限在开发与调试阶段使用**

- 关于跨域和Ajax
  - 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”

3、小程序 -- 视图与逻辑

3.1 页面导航

- 页面导航指页面之间的相互跳转
- 小程序实现页面导航的两种方式：
  - 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
    - 导航到tabBar页面

```
//页面地址必须以/开头
//必须指定跳转方式，为switchTab
<navigator url="/pages/message/message" open-type="switchTab">导航消息页面
</navigator>
```

- 导航到非tabBar页面

```
//页面地址必须以/开头
//指定跳转方式，为navigate，可省略
<navigator url="/pages/info/info" open-type="navigate">导航到info页面
</navigator>
```

- 后退导航

```
//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,可省略，默认值为1
<navigator open-type="navigateBack" delta="1">后退</navigator>
```

- 编程式导航
  - 调用小程序的导航API，实现页面的跳转

属性	类型	是否必选	说明
url	string	**是**	需要跳转的页面的路径，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）

- 导航到tabBar页面

```
//wxml
<button bindtap="gotoMessage">跳转到消息页面</button>

//js
gotoMessage(){
    wx.switchTab({
        url:'/pages/message/message'
    })
}
```

- 导航到非tabBar页面

```
//wxml
<button bindtap="gotoInfo">跳转到信息页面</button>

//js
gotoInfo(){
  wx.navigateTo({
    url: '/pages/info/info'
  })
},
```

- 后退导航

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）

```
//wxml
<button bindtap="gotoBack">后退</button>

//js
gotoBack(){
    wx.navigateBack({
  })
}
```

- 导航传参
  - 声明式导航传参
    - 路径后面携带参数：
      - 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

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

  - 编程式导航传参

```
//wxml
<button bindtap=gotoInfo2">跳转到info页面</button>

//js
gotoInfo2(){
    wx.navigateTo({
    url: " /pages/infolinfo?name=ls&gender=男"
  })
}
```

- 在onLoad中接收导航参数

```
onLoad:function(options){
    //options即导航传递过来的参数对象
    console.log(options)
}
```

3.2 页面事件

下拉刷新事件

- 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
- 启用下拉刷新的方式：
  - **全局**开启下拉刷新(app.json的windows结点设置)
  - **局部**开启下拉刷新(页面的.json中配置)
  - **实际开发中为需要的页面单独开启下拉刷新的效果**
- 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

```
onPullDownRefresh() {
    console.log('触发下拉刷新事件')
    wx.stopPullDownRefresh()
  },
```

- 停止下拉刷新的效果
  - 调用**wx.stopPullDownRefresh()**可以停止当前页面的下拉刷新

上拉触底事件

- 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
- 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**(节流 --在data 中定义isloading节流阀；在方法中修改isloading节流阀的值；在onReachBottom中判断节流阀的值，从而对数据请求进行节流控制)**
- 配置上拉触底距离
  - 触发上拉触底事件时，滚动条距离页面底部的距离，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

3.3 生命周期

- 在小程序中，生命周期分为两类,分别是:
  - 应用生命周期(范围较大)特指小程序从启动->运行->销毁的过程
  - 页面生命周期(范围较小)特指小程序中，每个页面的加载->渲染->销毁的过程
- 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，允许程序员在特定时间点执行某些特定的操作
- **生命周期**强调的是**时间段**，**生命周期函数**强调的是**时间点**
- 应用生命周期函数

```
//app.js 文件
App({
    //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(options) { },
    //小程序启动，或从后台进入前台显示时触发。
    onShow : function(options) { },
    //小程序从前台进入后台时触发。
    onHide : function() { }
})
```

- 页面生命周期函数

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

3.4 WXS脚本

概述

- 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
- 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
- WXS与JavaScript的关系
  - **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
  - **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
  - **wxs遵循CommonJS规范**
    - module对象、require()函数、module.exports对象

基础语法

- 内嵌wxs脚本
- 外联wxs脚本

wxs的特点

1. 大量借鉴JavaScript的语法(但是是两种不同的语言)
2. 不能作为组件的事件回调(配合Mustache语法使用)
3. 隔离性(不能调用js定义的函数及小程序提供的API)
4. 性能好(iOS上比js快2-20倍)

4、小程序 -- 基础加强

4.1 自定义组件

组件的创建

- 在项目的根目录中，鼠标右键。创建components -> test文件夹
- 在新建的components -> test文件夹上，鼠标右键，点击“新建Component"
- 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

组件的引用

- 局部引用:组件只能在当前被引用的页面内使用**（在pages中配置）**

```
//json
{
  "usingComponents": {
    "my-test1":"/components/test/test"
  }
}
//wxml
<my-test1></my-test1>
```

- 全局引用:组件可以在每个小程序页面中使用**（在app.json中配置）**

组件与页面的区别

- 组件的.json文件中需要声明"component": true属性
- 组件的.js文件中调用的是Component()函数
- 组件的事件处理函数需要定义到 methods节点中

组件样式

- 组件样式隔离：组件不会影响到小程序页面和其它组件的样式
- **app.wxss的全局样式对组件无效，只有class选择器会有样式隔离效果**
- 可以通过**styleIsolation**修改组件的样式隔离选项

```
//.js
options:{
    styleIsolation:'isolated'
  },
//或者
//.json
{
    "styleIsolation":"isolated"
}
```

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
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

数据、方法和属性

- 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
- 在小程序组件中，**事件处理函数和自定义方法**需要定义到methods节点中
- 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**
    - 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

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

```
observers:{
    'n1,n2':function name(newN1,newN2) {
      this.setData({
        sum:newN1+newN2
      })
    }
  }
```

- 数据监听器支持监听对象中单个或多个属性的变化**('对象.属性A')**
- 如果某个对象中需要被监听的属性太多，可以使用通配符**来监听对象中所有属性的变化

纯数据字段

- 纯数据字段指的是那些不用于界面渲染的data字段,**既不会展示在界面上。也不会传递给其他组件**
- 纯数据字段有助于**提升页面更新的性能**

```
options:{
    //指定纯数据字段格式
    pureDataPattern:/^_/
  },
  data:{
    a:true,//普通数据字段
    _b:true//纯数据字段
  }
```

组件的生命周期

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**时执行
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

- 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
- 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**
- 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

- 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)**。

```
Component({
    lifetimes:{     //推荐用法
        attached(){ }
    },
    //attached(){ }, 旧式定义方式
})
```

组件所在页面的生命周期

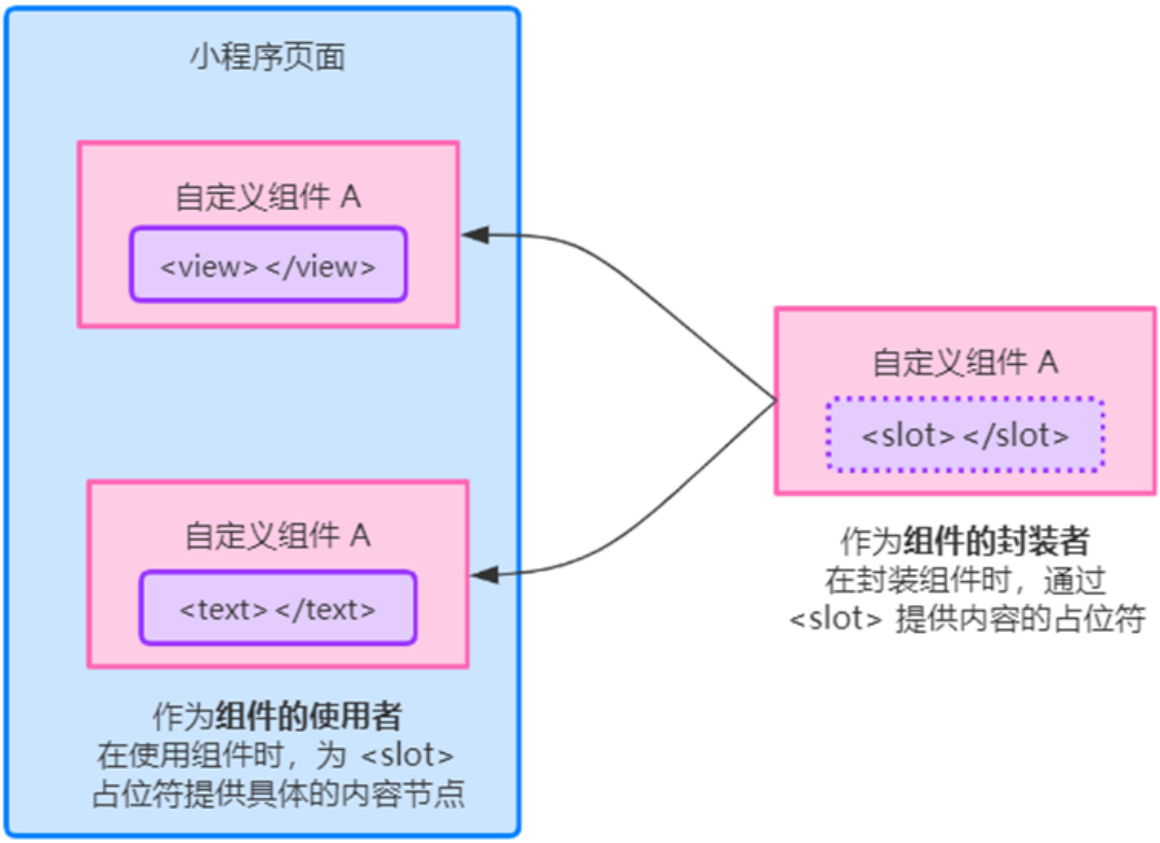
- 自定义组件的行为依赖于页面状态的变化，此时就需要用到组件所在页面的生命周期。

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

```
Component({
    pageLifetimes:{
        show:function(){ },//页面被展示
        hide:function(){ },//页面被隐藏
        resize:function(size){ }//页面尺寸变化
    }
})
```

插槽

- 提供一个< slot>节点(插槽)，用于承载组件使用这提供的wxml结构，默认只允许使用单个插槽



- 多个插槽，以不同的name来区分不同的插槽

```
//component.js
options:{
    multipleSlots:true
  },

//component.wxml
<view>
  <slot name="before"></slot>
  <view>这里是组件的内部结构</view>
  <slot name="after"></slot>
</view>

//pages.wxml
<my-test>
  <view slot="before">这里通过before插槽填充的内容</view>
  <view slot="after">这里通过after插槽填充的内容</view>
</my-test>
```

父子组件之间的通信

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

```
//父组件data节点
data: {
    count:0
  },
```

```
// 父组件wxml
<my-test5 count="{{count}}"></my-test5>
<view>~~~~~~~~~~</view>
<view> 父组件中，count值是：{{count}}</view>
//子组件properties节点
properties: {
    count:Number
  },
// 子组件wxml
<view>子组件中，count值是：{{count}}</view>
```

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

```
//父组件定义方法
syncCount(e){
    this.setData({
      count:e.detail.value
    })
  },
//父组件wxml，使用 bind:自定义事件名称
<my-test5 count="{{count}}" bind:sync="syncCount"></my-test5>
//子组件wxml
<view>子组件中，count值是：{{count}}</view>
<button bindtap="addCount">+1</button>
//子组件方法
methods: {
    addCount(){
      this.setData({
        count:this.properties.count+1
      })
      this.triggerEvent('sync',{value:this.properties.count})
    }
  }
```

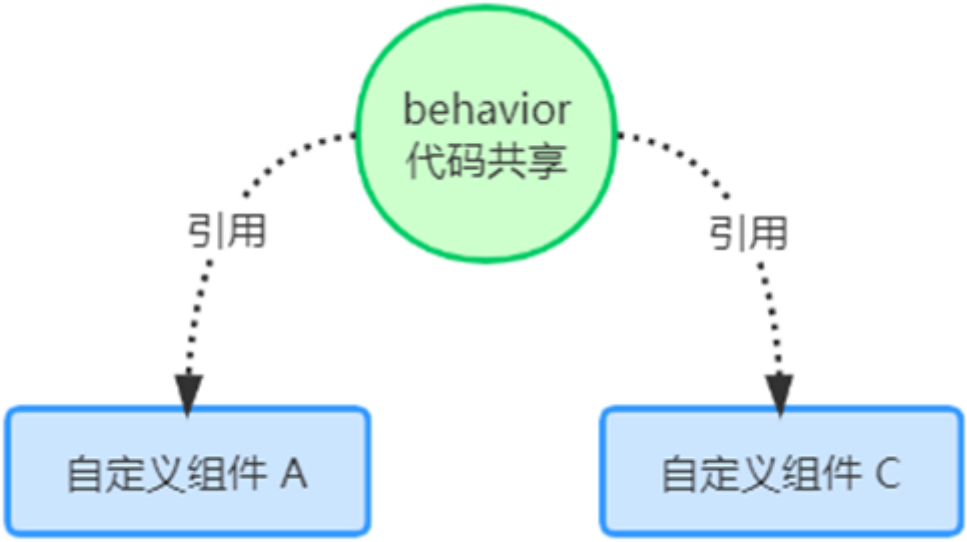
3. 获取组件实例

   - 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**

```
//父组件wxml结构
<my-test5 count="{{count}}" bind:sync="syncCount" class="customA"></my-
test5>
<view> 父组件中，count值是：{{count}}</view>
<button bindtap="getChild">获取子组件的实例对象</button>
//父组件js方法 id选择器或者class选择器
getChild(){
    const child = this.selectComponent('.customA')
    child.setData({count:child.properties.count + 1})
    child.addCount()
  },
```

behaviors

- 用于**实现组件代码共享**的特性



- 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

```
//创建文件夹，创建文件
module.exports = Behavior({
  data:{username:'zs'},
  properties:{},
  methods:{}
})
//js文件中使用
const myBehavior =  require('../../behaviors/my-behaviors')
// components/test5/test5.js
Component({
  behaviors:[myBehavior],
})
```

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

- 覆盖和组合规则：https://developers.weixin.qq.com/miniprogram/dev/framework/custom-component/behaviors.html

4.2 使用npm包

npm包的限制

- 不支持依赖于Node.js内置库的包
- 不支持依赖于浏览器内置对象的包
- 不支持依赖于C++插件的包

Vant Weapp

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

- 安装Vant组件库(详细参考https://youzan.github.io/vant-weapp/#/quickstart#an-zhuang)
    1. 通过npm安装(建议指定版本为@1.3.3)
    2. 构建npm包
    3. 修改app.json
- 使用Vant组件：

```
//app.json
"usingComponents": {
    "van-button": "@vant/weapp/button/index"
  }
//pages的wxml结构
<van-button type="primary">按钮</van-button>
```

- 定制全局主题样式

- CSS变量的基本用法：https://developer.mozilla.org/zh-CN/docs/Web/CSS/Using_CSS_custom_properties
- 颜色变量：https://github.com/youzan/vant-weapp/blob/dev/packages/common/style/var.less

```
page {
//定制警告按钮的背景颜色和边框颜色
  --button-danger-background-color:#C00000;
  --button-danger-border-color:#D60000;
}
```

API Promise化

- 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，**存在回调地狱**，代码的可读性、维护性差
- 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

```
//装包，需重新构建
npm i --save miniprogram-api-promise@1.0.4
//app.js
import {promisifyAll} from 'miniprogram-api-promise'
const wxp = wx.p = {}
promisifyAll(wx,wxp)
```

4.3 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
- 在小程序中，可使用mobx-miniprogram配合mobx-miniprogram-bindings实现全局数据共享。
  - mobx-miniprogram 用来**创建Store实例对象**
  - 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**

```
//装包，重新构建
npm i --save mobx-miniprogram@4.13.2 mobx-miniprogram-bindings@1.2.1
```

```
//创建MobX的Store实例 创建store文件夹、store.js文件
import {observable,action} from 'mobx-miniprogram'

export const store = observable({
  //数据字段
  numA:1,
  numB:2,
  //计算属性
  get sum(){
    return this.numA + this.numB
  },
  //actions方法，用来修改store中的数据
  updateNum1:action(function(step){
```

```
    this.numA += step
  }),
  updateNum2:action(function(step){
    this.numB += step
  }),
})


//将Store中的成员绑定到页面中 页面的.js文件
import {createStoreBindings} from 'mobx-miniprogram-bindings'
import{store} from '../../store/store'

Page({
  onLoad:function(){ //监听页面加载
    this.storeBindings = createStoreBindings(this,{
      store,    //指定要绑定的Store
      fields:['numA','numB','sum'],//指定要绑定的字段数据
      actions:['updateNum1']//指定要绑定的方法
    })
  },
  onUnload:function(){//监听页面卸载
    this.storeBindings.destroyStoreBindings()
  }
})


//页面wxml
<view> {{numA}} + {{numB}} = {{sum}} </view>
<van-button type="primary" bindtap="btnHandler1" data-step="{{1}}"> numA+1
</van-button>
<van-button type="danger" bindtap="btnHandler1" data-step="{{-1}}"> numA-1
</van-button>


//组件js
import { storeBindingsBehavior } from 'mobx-miniprogram-bindings'
import {store} from '../..store/store'

Component({
  behaviors:[storeBindingsBehavior],
  storeBindings:{
    store,
    fields:{
      numA:'numA',
      numB:'numB',
      sum:'sum'
    },
    actions:{
      updateNum2:'updateNum2'
    }
  },
})


//组件wxml
```

```
<my-numbers></my-numbers>


//组件js
methods: {
    btnHandler2(e){
      this.updateNum2(e.target.dataset.step)
    }
  }
```

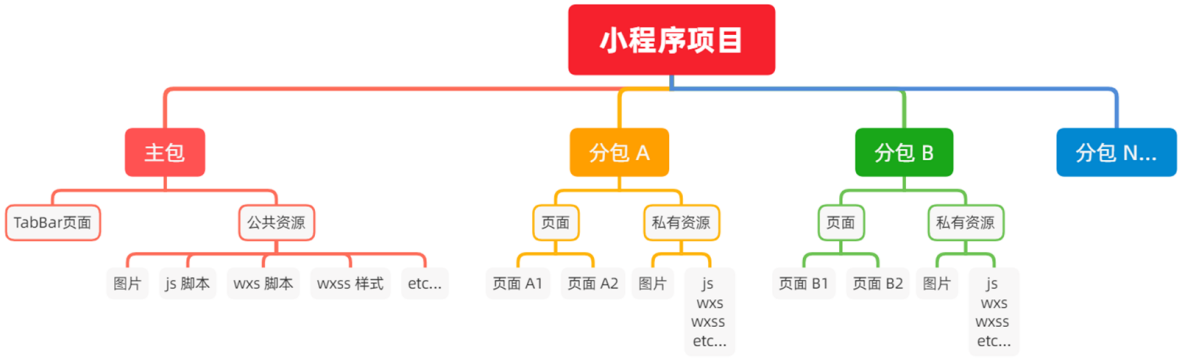
4.4 分包

概念

- 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
- 优点：
  - 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
- 分包项目由1个主包 + 多个分包组成
  - 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**(启动是默认下载并启动)**
  - 分包:只包含和当前分包有关的页面和私有资源**(进入时下载)**
  - 总包不超过**16M**，单包不超过**2M**

使用

- 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
- 主包也可以有自己的pages(即最外层的 pages字段)
- tabBar页面必须在主包内
- 分包之间不能互相嵌套



- **主包无法引用分包**内的私有资源
- **分包之间不能相互引用**私有资源
- **分包可以引用主包**内的公共资源

独立分包

- **独立分包本质也是分包，可以独立于主包和其他分包而单独运行**`"independent": true`

- 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
  - 小程序从普通分包页面启动时，首先需要下载主包
  - 独立分包不依赖主包即可运行，**提升分包页面的启动速度**
- **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！！**

分包预下载

- 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
- 同一个分包中的页面享有共同的预下载大小限额**2M**

```
//app.json
"preloadRule": {
    "pages/contact/contact":{
      //指定的网络模式下进行预下载
      "network": "all",
      //进入页面后预下载哪些分包
      "packages":["pkgA"]
    }
  },
```