# 基础

## 事件



### bindtap 的语法格式



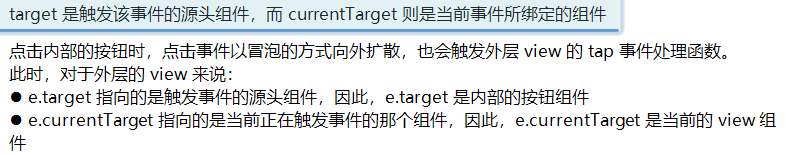
### 事件传参



### 事件对象



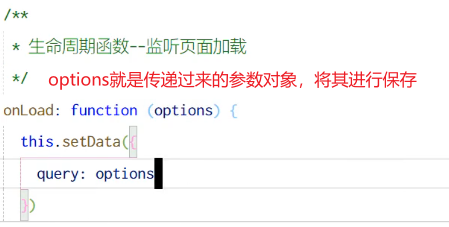
#### target 和 currentTarget 的区别



## 发送网络请求



## 接收传递的参数

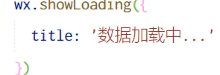


## 设置页面标题



## loading效果

### 设置loading



### 隐藏loading



## \* 页面相关事件处理函数--监听用户下拉动作

   \*/

  onPullDownRefresh: function () {

  },

## \* 页面上拉触底事件的处理函数

   \*/

  onReachBottom: function () {

    if(this.data.flag) return

    this.setData({page:this.data.page+1})

    this.getGoodsList()

  },

## 加载弹窗



## 页面配置(开启下拉刷新等)



## 关闭下拉刷新



## 好的写法

getGoodsList(cb){

    …

      complete:()=>{

        wx.hideLoading()

        this.setData({flag:false})

        cb && cb() // 用传入的参数决定是否执行某些方法

      }

    })

  },

    this.getGoodsList(()=>{

      wx.stopPullDownRefresh()

    })

## 导航

### 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

#### 导航到 tabBar 页面



#### 导航到非 tabBar 页面



#### 后退导航



### ② 编程式导航

⚫ 调用小程序的导航 API，实现页面的跳转

#### 导航到 tabBar 页

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：





#### 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：





#### 后退导航





### 导航传参

#### 声明式导航传参

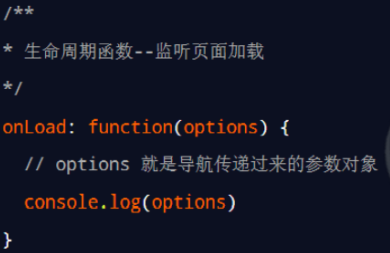


#### 编程式导航传参



#### 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



## 组件

### 组件的注册

#### 局部注册



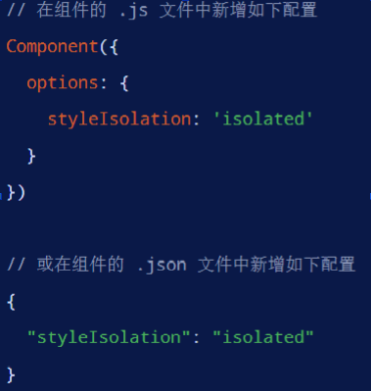
#### 全局注册

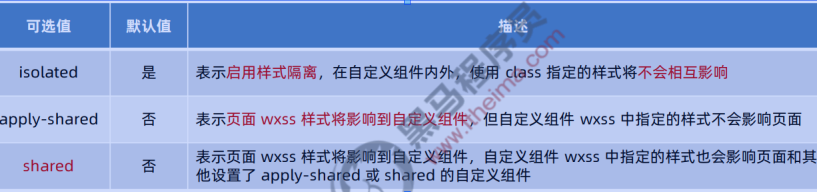


### 样式隔离

页面的类样式不会影响组件的样式，但其他选择器会

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





### 接收父组件/页面传递的属性值

this.properties.xxx



### data和properties所定义的数据是放置在同一个对象中的

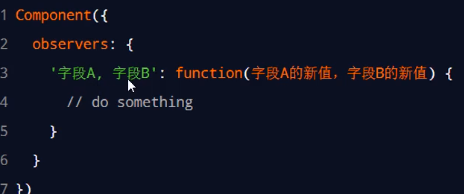
this.data === this.properties // true

只是人为的对其进行了功能的划分

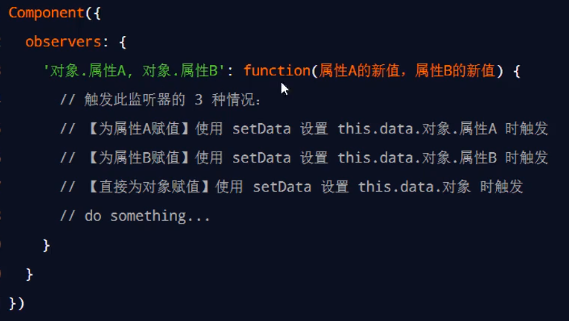
⚫ data 更倾向于存储组件的私有数据

⚫ properties 更倾向于存储外界传递到组件中的数据

### 数据监听器Observers



#### 监听对象单个属性的变化



#### 监听对象上所有属性的变化



## 纯数据字段pureDataPattern



### 使用方式





## 生命周期

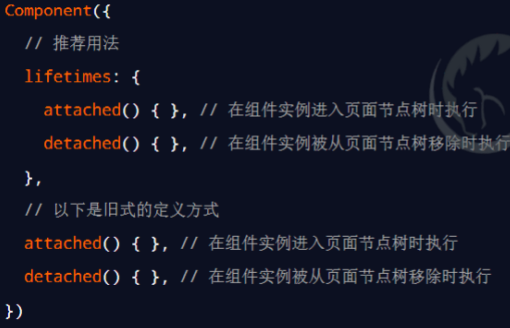
### 组件的生命周期





#### 组件生命周期的定义方式

两种同时存在，以新的为主



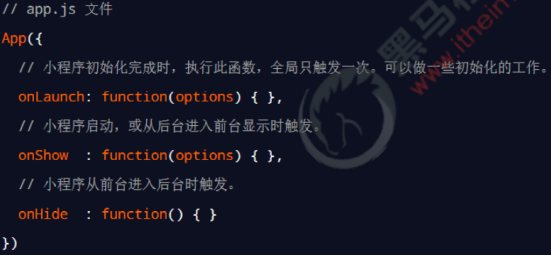
#### 组件所在页面的生命周期函数



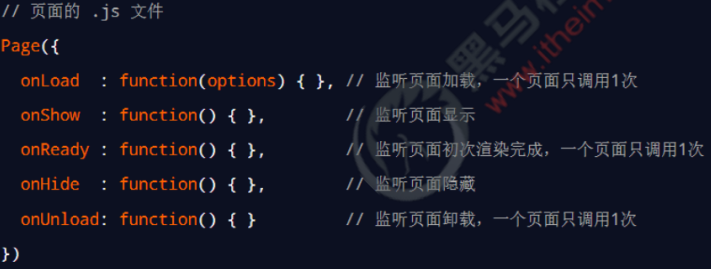
##### 组件所在页面的生命周期函数，需要定义在 pageLifetimes 节点中，示例代码如下



### 应用的生命周期



### 页面的生命周期



## wxs

### 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

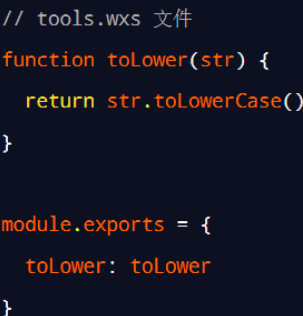
### wxs 的应用场景

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”】

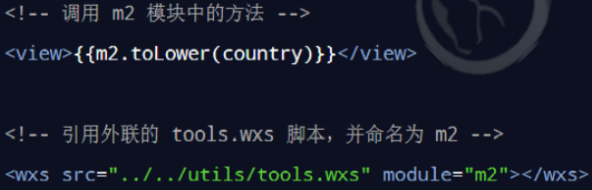
### 内嵌 wxs 脚本



### 定义外联的 wxs 脚本



### 使用外联的 wxs 脚本



### WXS 的特点

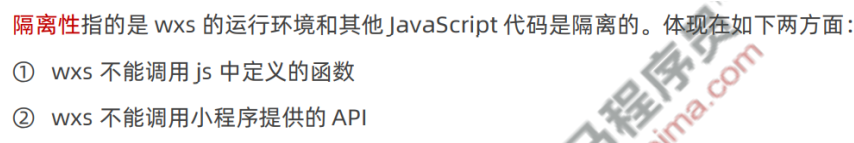
#### 与 JavaScript 不同

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

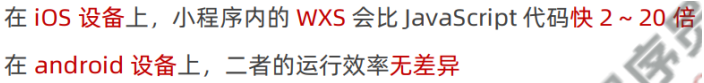
#### 不能作为组件的事件回调



#### 隔离性



#### 性能好



## 插槽

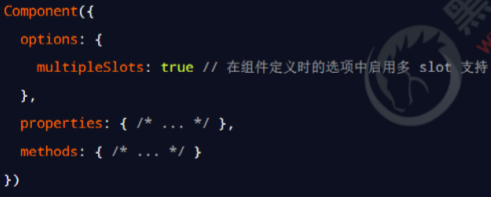
### 单个插槽

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

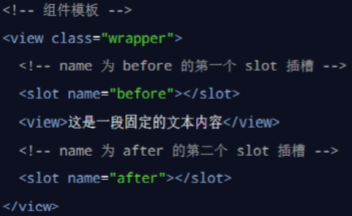


### 启用多个插槽

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，通过如下方式启用



### 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

### 使用插槽



## 父子组件之间的通信

### 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

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

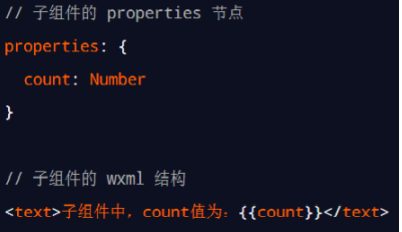
⚫ 这样就可以直接访问子组件的任意数据和方法

### 属性绑定

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。

【父组件的示例代码】 如下：





### 事件绑定

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：

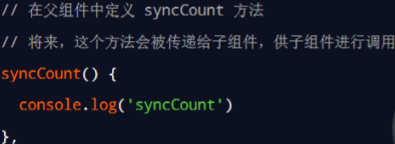
① 在父组件的 js 中，定义一个函数，这个函数即将通过自定义事件的形式，传递给子组件

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

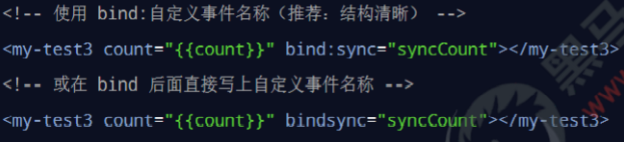
③ 在子组件的 js 中，通过调用 this.triggerEvent('自定义事件名称', { /\* 参数对象 \*/ }) ，将数据发送到父组件

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

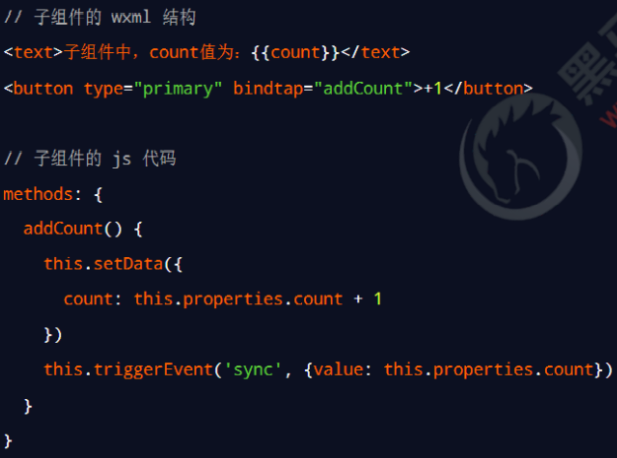
步骤1：在父组件的 js 中，定义一个函数，这个函数即将通过自定义事件的形式，传递给子组件



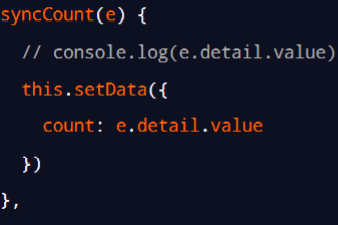
步骤2：在父组件的 wxml 中，通过自定义事件的形式，将步骤 1 中定义的函数引用，传递给子组件



步骤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



步骤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



### 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")



注意红框位置：使用child子组件实例，不能使用this。this指向父组件，无法获取子组件的属性和方法

## behavior

### 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”

每个 behavior 可以包含一组属性、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

### 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：



### 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，示例代码如下



### behavior 中所有可用的节点



### 同名字段的覆盖和组合规则\*



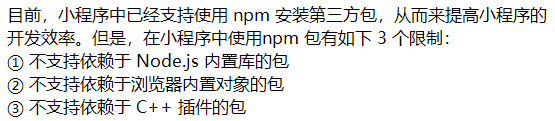
## 使用 npm 包

### 注意点

每次下载安装新的npm包，都需要重新删除miniprogram\_npm文件夹

然后重新 工具--构建npm 否则有可能出问题

### 小程序对 npm 的支持与限制



### 使用vant组件/第三方npm组件库的步骤

具体看官网

[快速上手 - Vant Weapp (youzan.github.io)](https://youzan.github.io/vant-weapp/#/quickstart)

1、npm init -y

2、npm i @vant/weapp@1.3.3 -S –production

3、

4 app.json 中的 **"style": "v2"** 去除，小程序的[新版基础组件](https://developers.weixin.qq.com/miniprogram/dev/reference/configuration/app.html#style)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。

### 使用css自定义属性

<https://developer.mozilla.org/zh-CN/docs/Web/CSS/Using_CSS_custom_properties#%E4%BD%BF%E7%94%A8%E8%87%AA%E5%AE%9A%E4%B9%89%E5%B1%9E%E6%80%A7%E7%9A%84%E7%AC%AC%E4%B8%80%E6%AD%A5>

### 修改导入的vant组件的样式

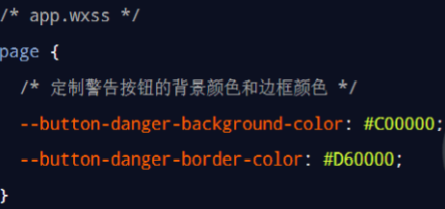
1 查看对应的样式变量名称

[vant-weapp/var.less at dev · youzan/vant-weapp · GitHub](https://github.com/youzan/vant-weapp/blob/dev/packages/common/style/var.less)

2 修改

vant上的变量使用的是less写的，所以变量名都是@开头，我们在微信小程序上使用

-- 开头



## API的promise化



### 实现 API Promise 化



const wxp=wx.p ={}作用：

为了确保每一个页面都能访问到promise化之后的api，所以还在wx这个顶级对象上添加了一个自定义的属性，接收promise化之后的api

理解：

1 使用promisifyAll方法，对wx里面的所有api promise化

2 把wx这个顶级对象和wxp这个空对象传递进去

3 最终会把wx上的所有基于异步回调的api进行promise化

4 promise化完毕之后挂载到wxp这个空对象上

5 之后就可以在每一个页面上通过wx.p这个顶级对象下的自定义属性，来调用那些promise化的api了

### 调用 Promise 化之后的异步 API



## 全局数据共享

### 小程序中的全局数据共享方案



### 安装 MobX 相关的包



npm i -s mobx-miniprogram@4.13.2 mobx-miniprogram-bindings@1.2.1

### 创建 MobX 的 Store 实例



### 将 Store 中的成员绑定到页面中



### 在页面上使用 Store 中的成员



### 将 Store 中的成员绑定到组件中



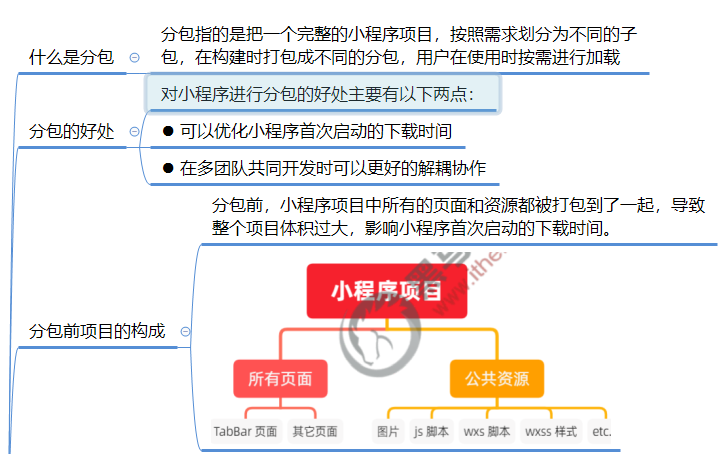
### 在组件中使用 Store 中的成员

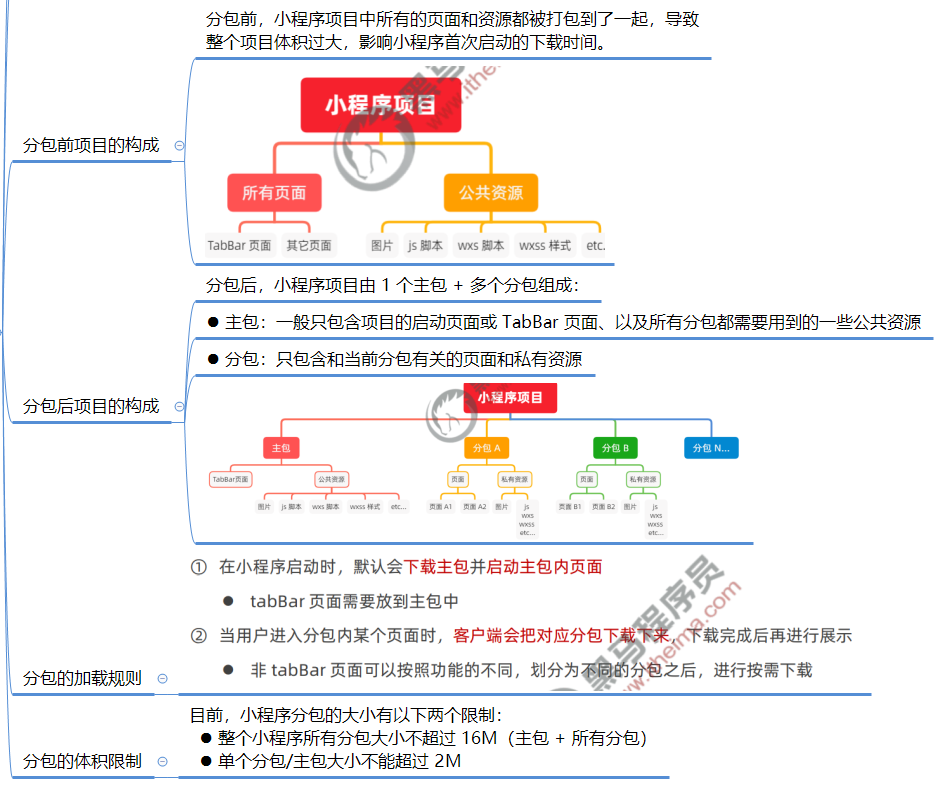


## 分包

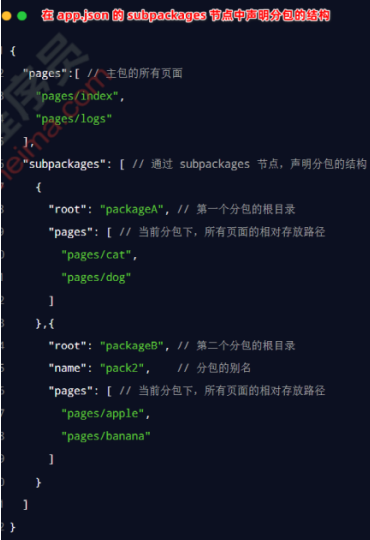
[(4条消息) 微信小程序分包\_黑面饺子皮的博客-CSDN博客\_微信小程序分包](https://blog.csdn.net/m0_47298312/article/details/126753782)

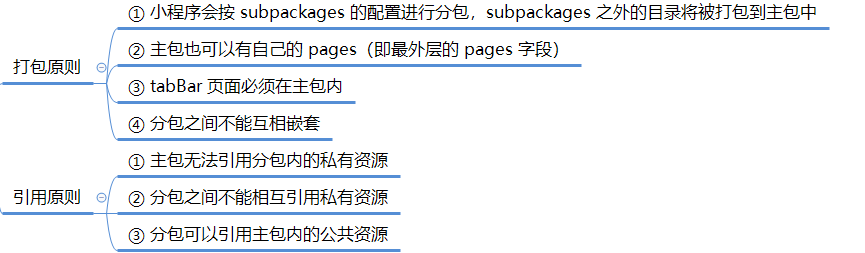
### 分包的基础概念



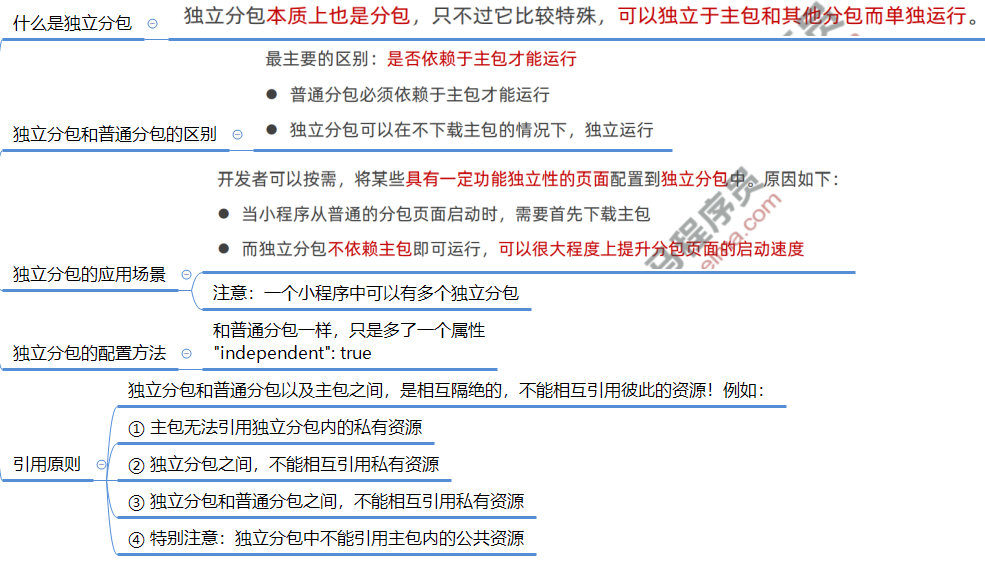


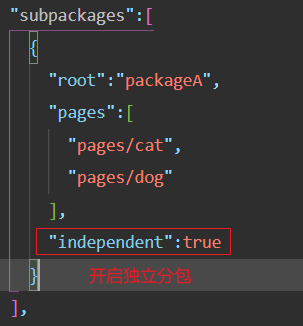
### 使用分包





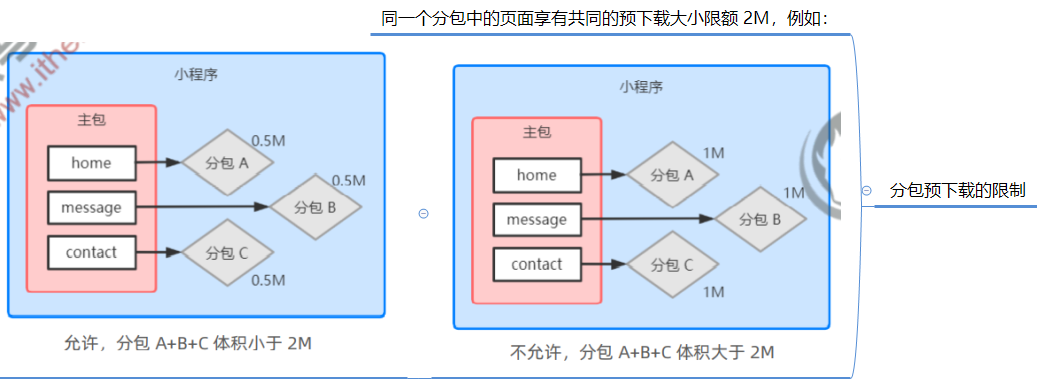
### 独立分包





### 分包预下载





## 自定义tabbar

### 实现步骤

自定义 tabBar 分为 3 大步骤，分别是：

① 配置信息

② 添加 tabBar 代码文件

③ 编写 tabBar 代码

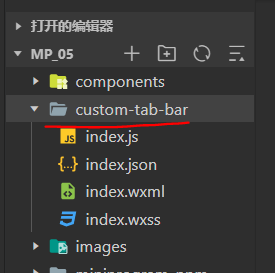
详细步骤，可以参考小程序官方给出的文档：

<https://developers.weixin.qq.com/miniprogram/dev/framework/ability/custom-tabbar.html>

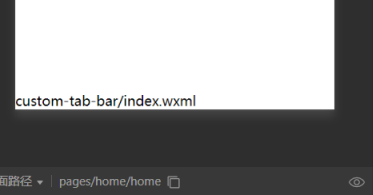
### 配置信息



新建文件夹custom-tab-bar，文件夹里面必须是index命名的文件



会在页面底部自动识别到自定义tabbar组件信息

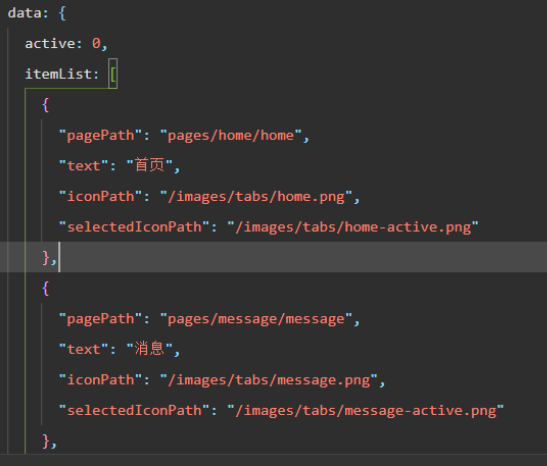


### 渲染自定义的tabbar样式

使用vant组件库的tabbar组件 <https://youzan.github.io/vant-weapp/#/tabbar>



index.js文件

渲染tabbar上的数字徽标样式

custom-tab-bar\index.wxss

.van-tabbar-item{

  --tabbar-item-margin-bottom:0

}

custom-tab-bar\index.js

Component({

  options:{

    styleIsolation:"shared" // 取消样式隔离

  },

….

})

以上操作是用来减小图标和文字之间的间隔

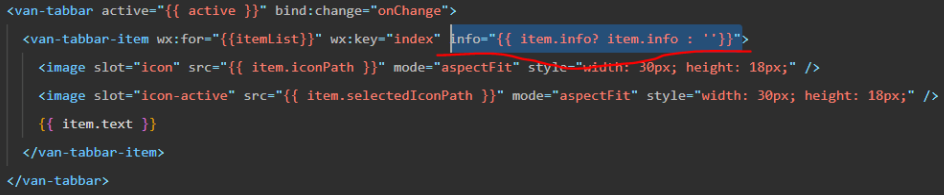


### 渲染tabbar数字徽标

3.1、在index.js中给某个tabbar中添加info属性

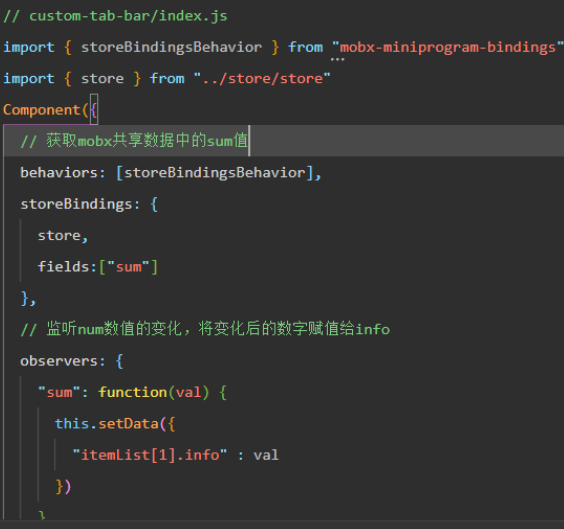


3.2、在wxml中进行判断

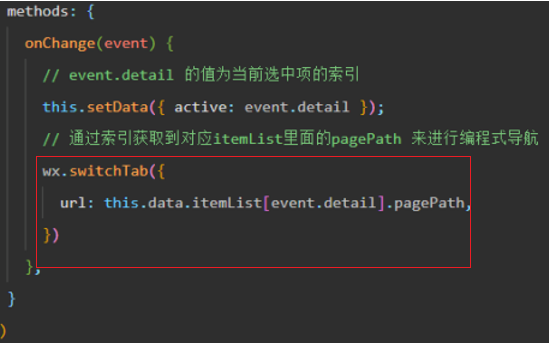


### 动态修改数字徽标(需要配合全局数据共享一起完成)

获取mobx共享数据，监听数据变化，将变化后的数值赋值给info



### 5、 tabbar页面跳转



#### 页面跳转的bug1

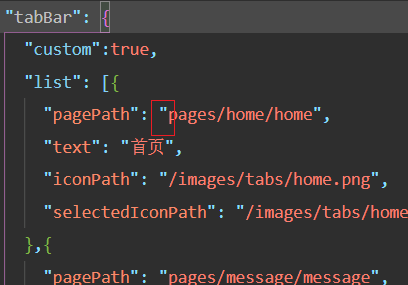
下图中，如果不加 / 就会报错



会默认多一层路径，导致路径匹配失败

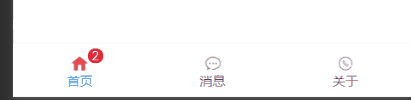


在app.json中的tabbar配置上则不需要添加 /



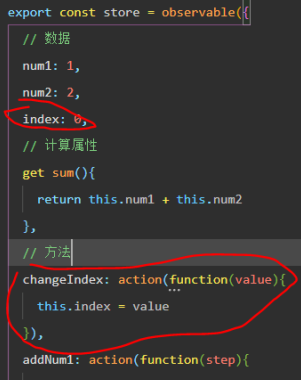
#### 页面跳转bug2

选中项索引会出问题

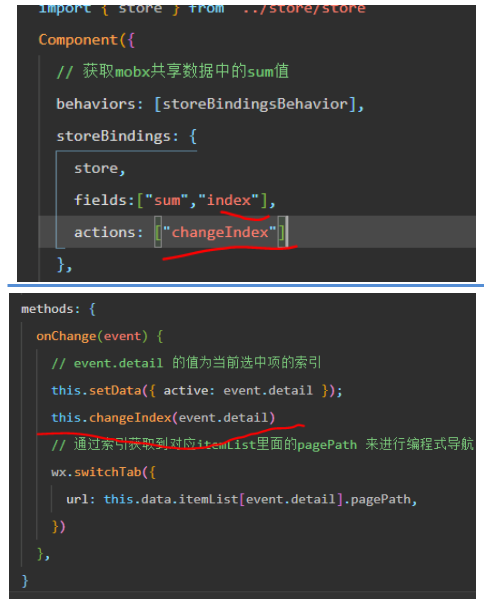


解决方案

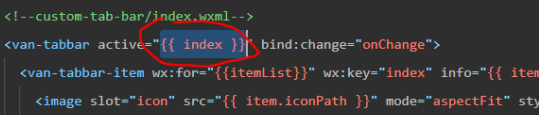
1. 将放置于局部的index索引，存放到mobx的store中，并在store中定义一个方法来改变索引



1. 在index.js中使用store中的索引，在点击tabbar时触发change事件，event.detail是当前选中的tabbar索引值，将该值当作参数传递给store中的修改索引值的方法



1. 页面直接调用该索引值



# websocket

[wx.sendSocketMessage(Object object) | 微信开放文档 (qq.com)](https://developers.weixin.qq.com/miniprogram/dev/api/network/websocket/wx.sendSocketMessage.html)

## 创建方式

下方两种方式创建webscoket的区别

### 第一种：



仅支持创建一个websocket

#### api使用

// 发送数据

  send: function(){

    // 拼接数据

    const msg=JSON.stringify({

      msg:this.data.msg

    })

    // 发送数据给后台

    wx.sendSocketMessage({

      data: msg

    })

  },

onLoad: function (options) {

    // 监听websocket连接打开

    wx.onSocketOpen((result) => {

      console.log("onSocketOpen:", result)

    })

    // 创建websocket连接

    wx.connectSocket({

      url: 'ws://127.0.0.1:8282',

      success:res=>{

        console.log("connectSocket", res);

      }

    })

    // 接收服务端发来的数据

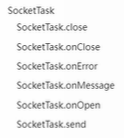
    wx.onSocketMessage((result) => {

      console.log(result)

    })

  },

### 第二种：



支持多个websocket连接，最多5个

#### api使用

let app = getApp() // 获取小程序全局唯一的 App 实例

// 发送数据给后台

    app.globalData.myws.send({

      data: msg

    })

onLoad: function (options) {

// app.globalData.myws是自己在app实例对象上定义的对象

    // 创建websocket连接

    app.globalData.myws = wx.connectSocket({

      url: 'ws://127.0.0.1:8282',

      success: res => {

        console.log("connectSocket",res);

      }

    })

    // 监听websocket连接打开

    app.globalData.myws.onOpen((result) => {

      console.log("onOpen:", result)

    })

    // 接收服务端发来的数据

    app.globalData.myws.onMessage((result) => {

        console.log(result)

    })

  },