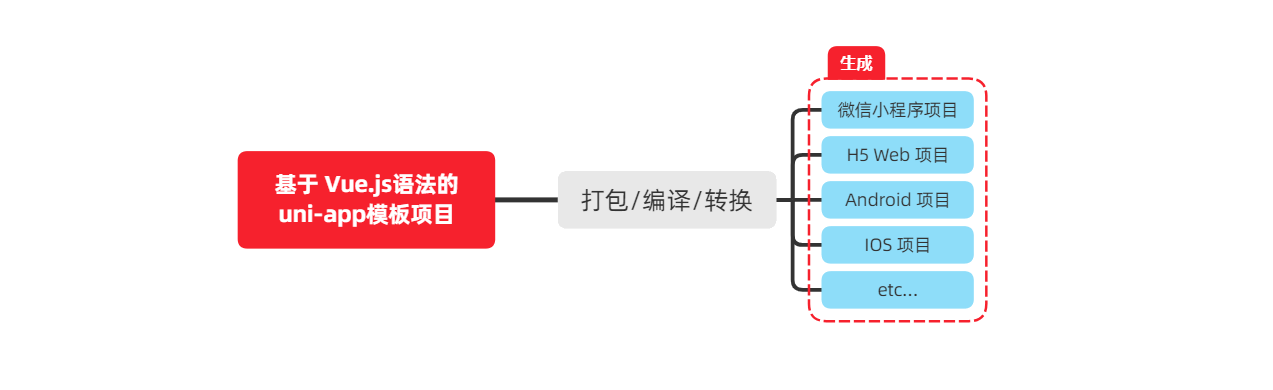
# 起步

## uni-app 简介

**uni-app 是一个使用 Vue.js 开发所有前端应用的框架**。开发者编写一套代码，可发布到 iOS、Android、H5、以及各种小程序（微信/支付宝/百度/头条/QQ/钉钉/淘宝）、快应用等多个平台。



详细的 uni-app 官方文档，请翻阅 <https://uniapp.dcloud.net.cn/>

参考

https://www.escook.cn/docs-uni-shop/mds/1.start.html#\_1-2-4-快捷键方案切换

https://applet-base-api-t.itheima.net/docs-uni-shop/index.htm

## 开发工具

uni-app 官方推荐使用 HBuilderX 来开发 uni-app 类型的项目。主要好处：

* 模板丰富
* 完善的智能提示
* 一键运行

当然，你依然可以根据自己的喜好，选择使用 VS Code、Sublime、记事本... 等自己喜欢的编辑器！

### 下载 HBuilderX并安装

进入<https://www.dcloud.io/hbuilderx.html>，



将下载的 zip包 进行解压缩

将解压之后的文件夹，存放到纯英文的目录中（且不能包含括号等特殊字符）

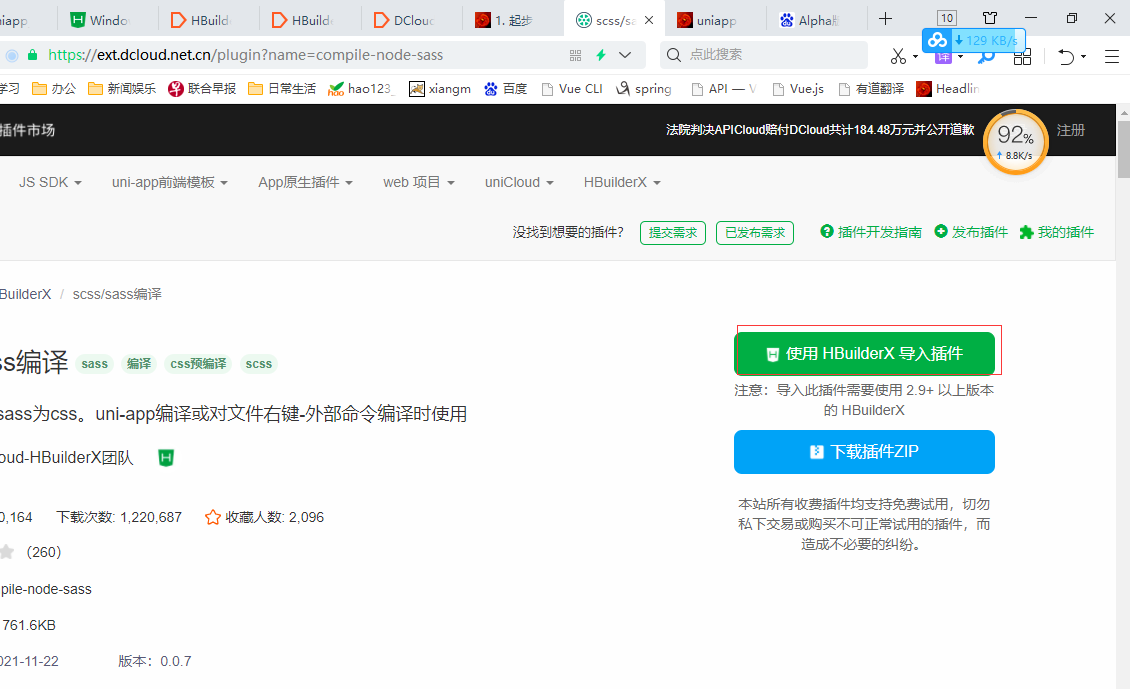
双击 HBuilderX.exe 即可启动 HBuilderX

### 安装 scss/sass 编译

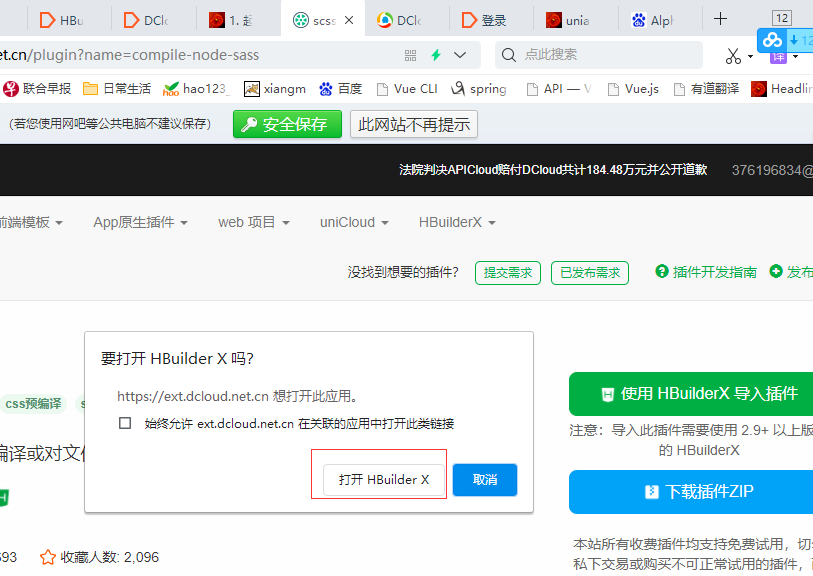
为了方便编写样式（例如：<style lang="scss"></style>），建议安装 scss/sass 编译 插件。插件下载地址：

https://ext.dcloud.net.cn/plugin?name=compile-node-sass

进入插件下载页面之后，点击右上角的 使用 HBuilderX 导入插件 按钮进行自动安装，截图如下：





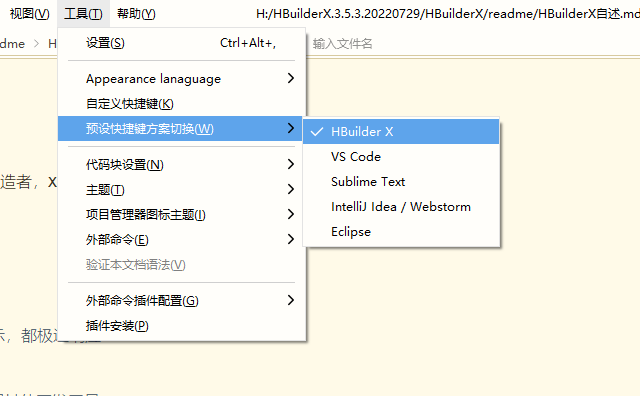


安装完成如下：



### 安装 scss/sass 编译

操作步骤：工具 -> 预设快捷键方案切换 -> VS Code



### 修改编辑器的基本设置

操作步骤：工具 -> 设置 -> 打开 Settings.json 按需进行配置

**源码视图**下可用的参考配置：

{

"editor.colorScheme": "Default",

"editor.fontSize": 12,

"editor.fontFamily": "Consolas",

"editor.fontFmyCHS": "微软雅黑 Light",

"editor.insertSpaces": true,

"editor.lineHeight": "1.5",

"editor.minimap.enabled": false,

"editor.mouseWheelZoom": true,

"editor.onlyHighlightWord": false,

"editor.tabSize": 2,

"editor.wordWrap": true,

"explorer.iconTheme": "vs-seti",

"editor.codeassist.px2rem.enabel": false,

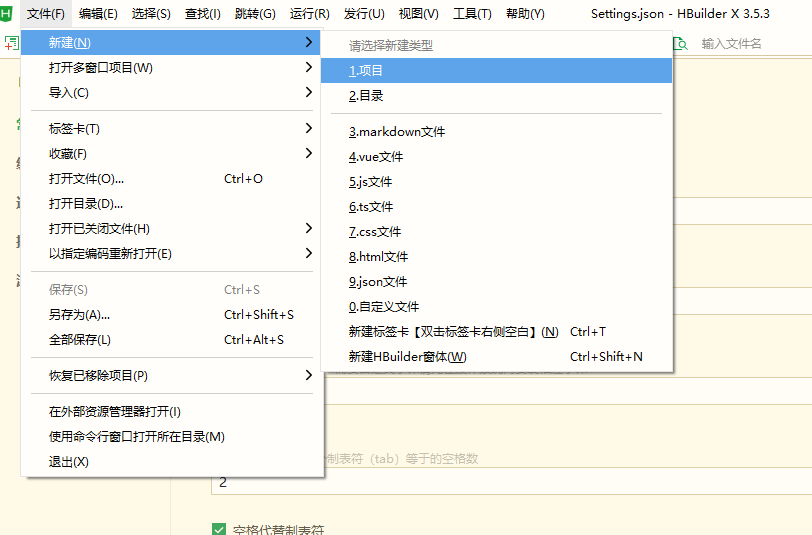
"editor.codeassist.px2upx.enabel": false

}

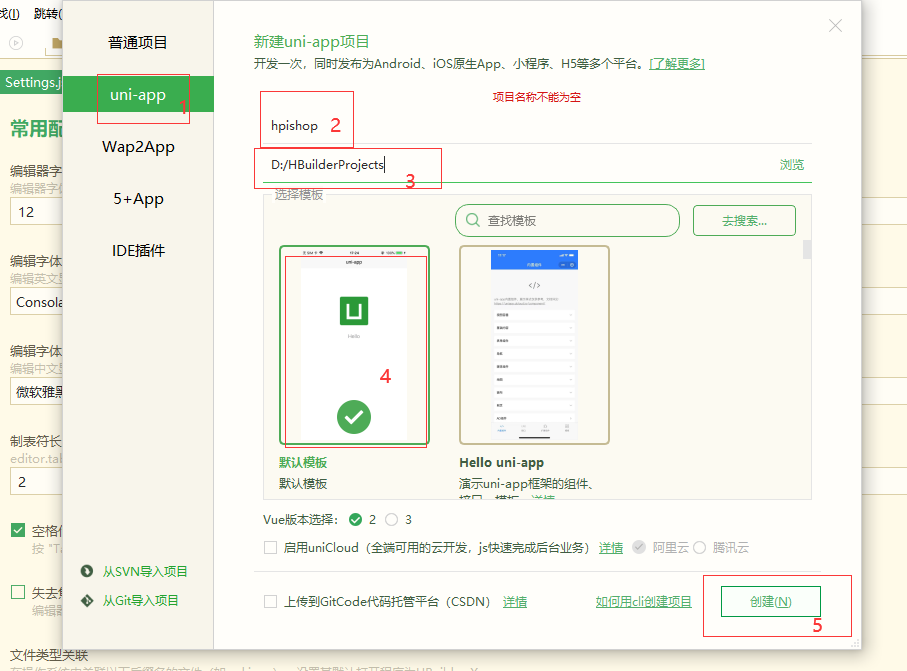


## 新建 uni-app 项目

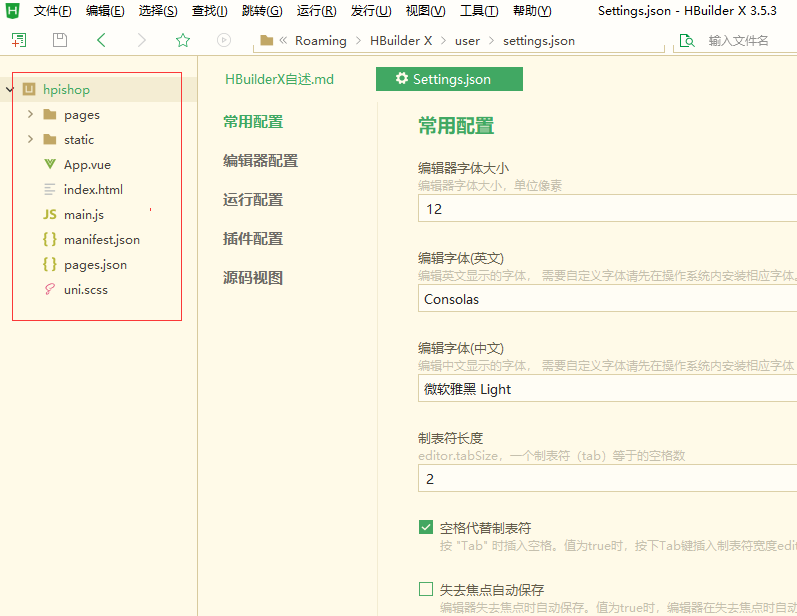
1. 文件 -> 新建 -> 项目



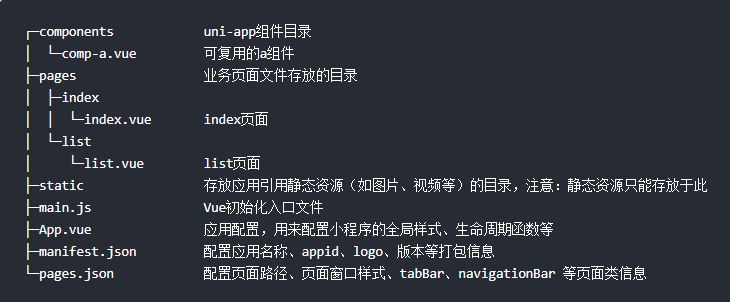
1. 填写项目基本信息



1. 项目创建成功

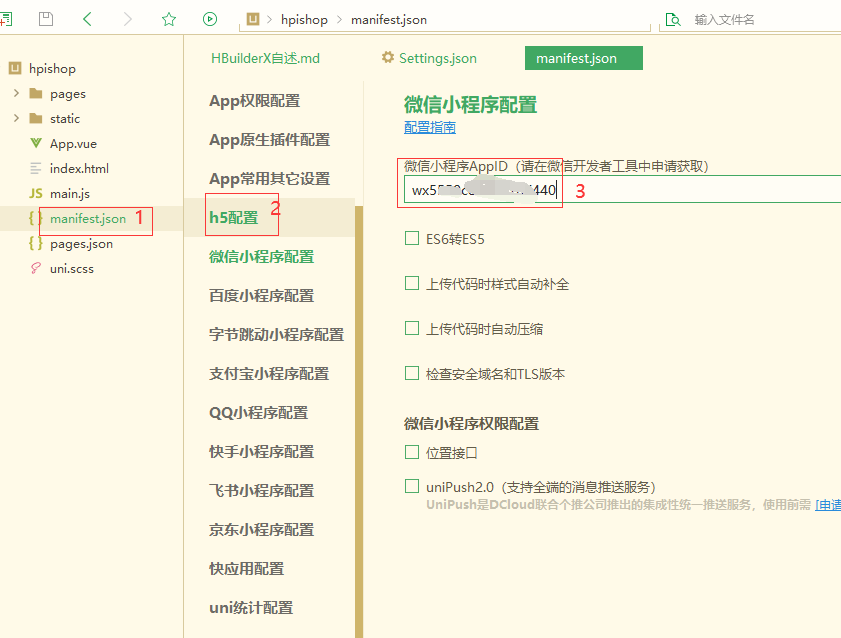


1. 目录结构



## 把项目运行到微信开发者工具

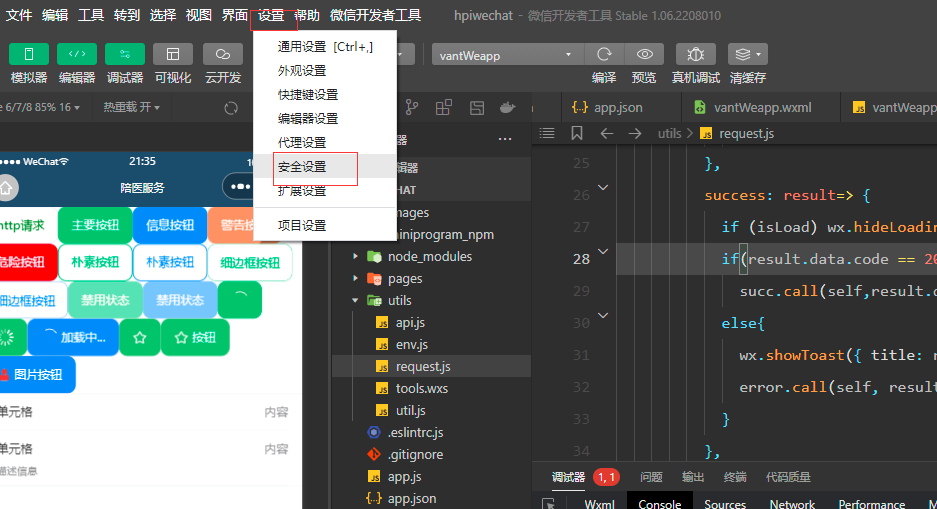
1. 填写自己的微信小程序的 AppID：

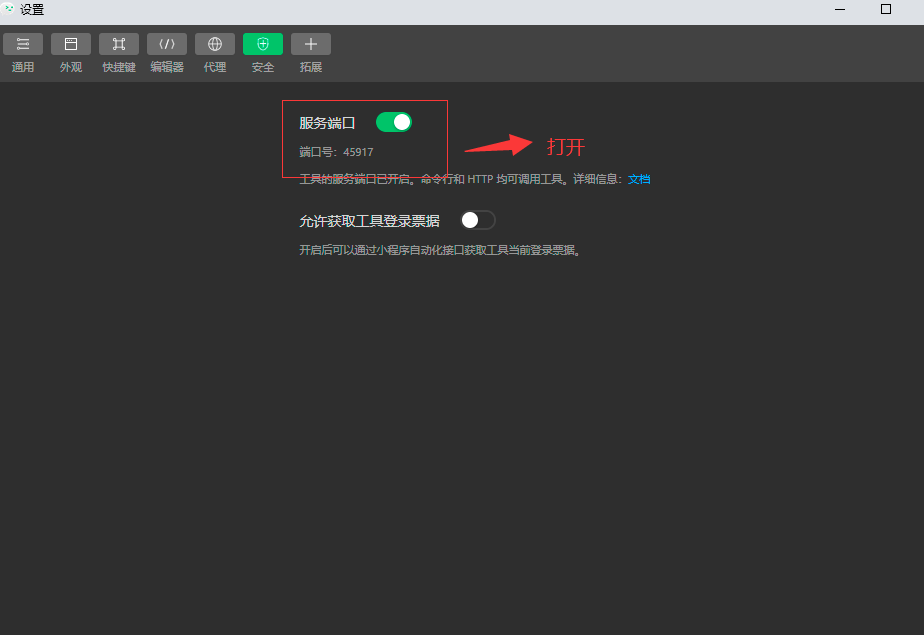


1. 在 HBuilderX 中，配置“微信开发者工具”的**安装路径**：

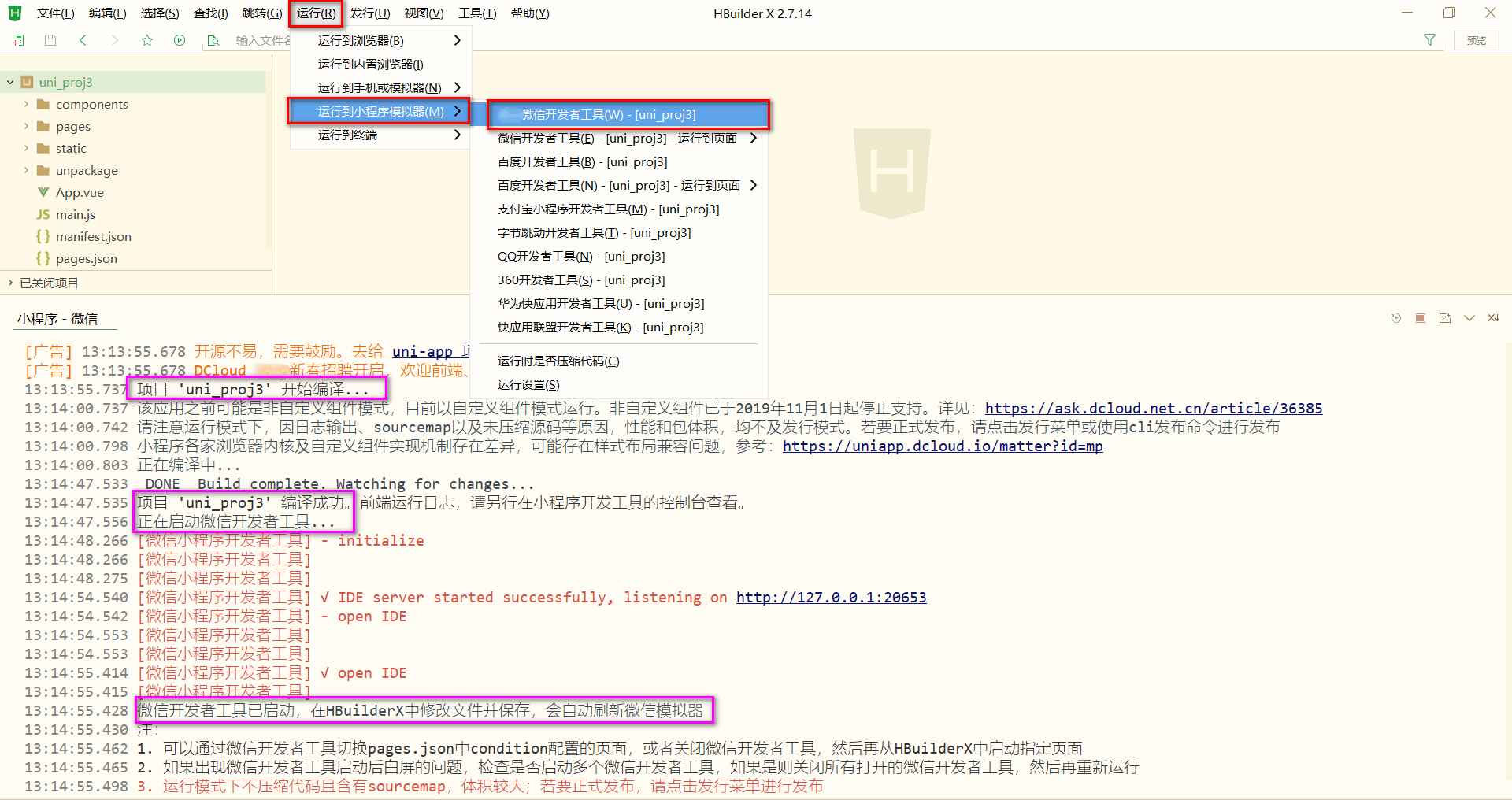


1. 在微信开发者工具中，通过 设置 -> 安全设置 面板，开启“微信开发者工具”的**服务端口**：

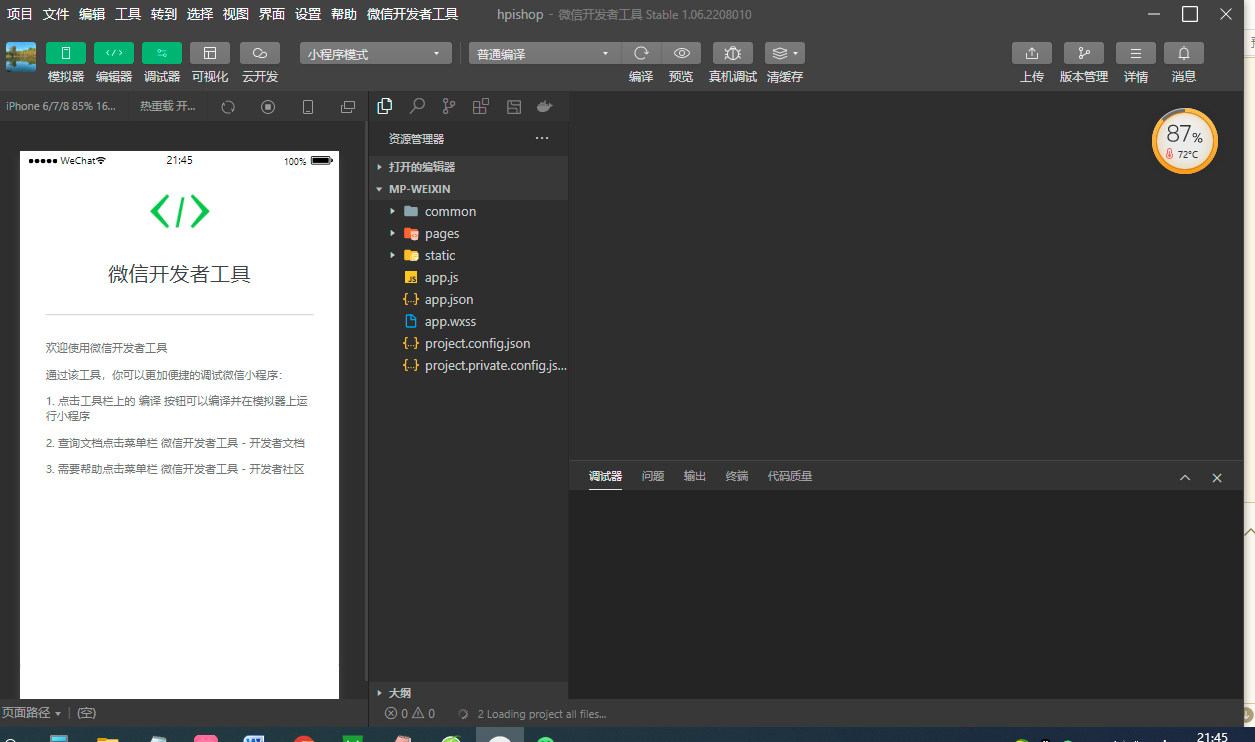




1. 在 HBuilderX 中，点击菜单栏中的 运行 -> 运行到小程序模拟器 -> 微信开发者工具，将当前 uni-app 项目编译之后，自动运行到微信开发者工具中，从而方便查看项目效果与调试：



中间可能要安装vue插件，比较慢，稍等片刻，安装成功后，重新运行，运行效果如下：



## 使用 Git 管理项目

### 本地管理

1. 在项目根目录中新建 .gitignore 忽略文件，并配置如下：

# 忽略 node\_modules 目录

/node\_modules

/unpackage/dist

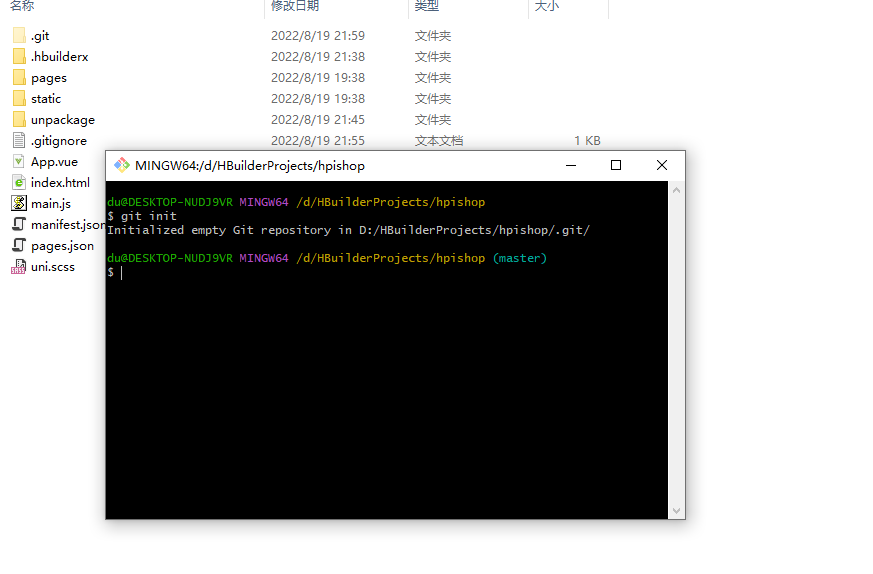


**注意：由于我们忽略了 unpackage 目录中仅有的 dist 目录，因此默认情况下， unpackage 目录不会被 Git 追踪**

**此时，为了让 Git 能够正常追踪 unpackage 目录，按照惯例，我们可以在 unpackage 目录下创建一个叫做 .gitkeep 的文件进行占位**

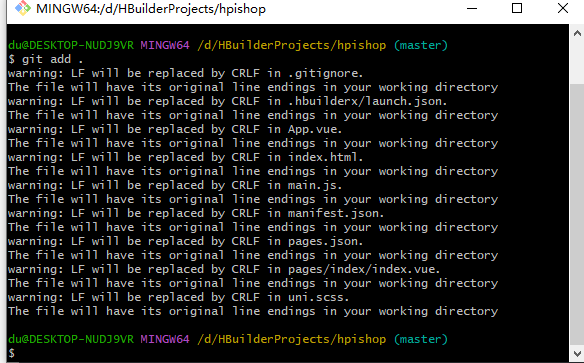
### 初始化版本库并提交到本地

1.打开终端，切换到项目根目录中，运行如下的命令，初始化本地 Git 仓库：



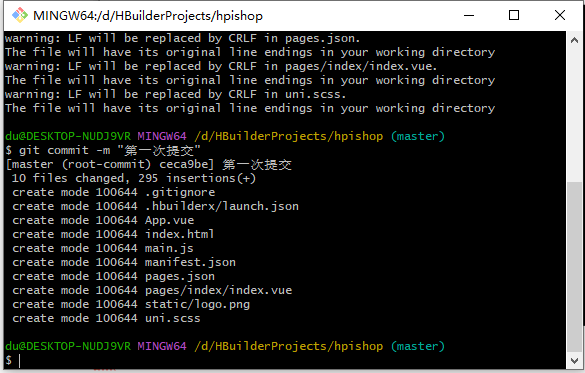
2.将所有文件都加入到暂存区：

git add .



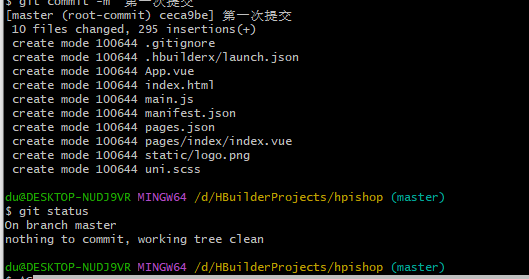
3.本地提交更新：

git commit -m "第一次提交"



查看状态

$ git status

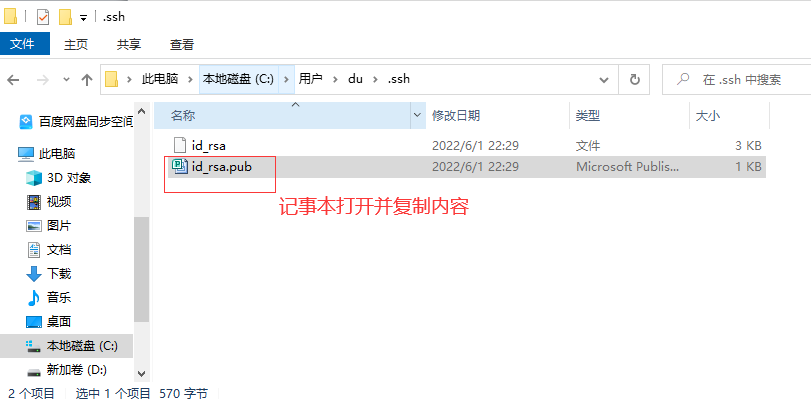


### 托管到码云

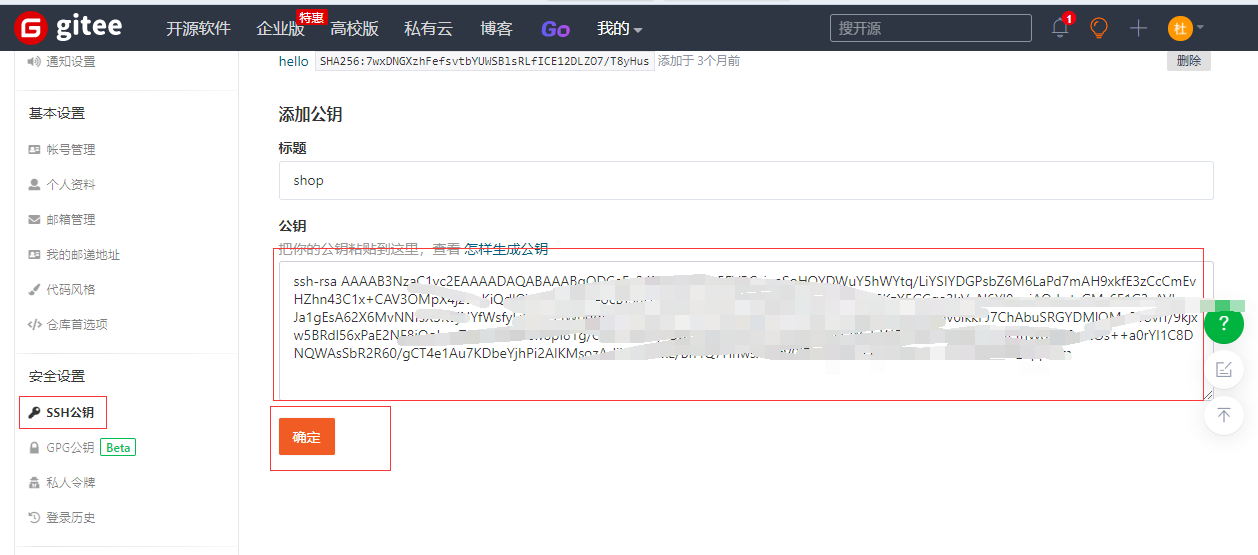
1. 注册并激活码云账号（ 注册页面地址：https://gitee.com/signup ）
2. 生成并配置 SSH 公钥（运行 ssh –t [git@gitee.com](mailto:git@gitee.com) 检查ssh公匙是否配置成功）



打开本地ssh公匙



把复制的粘贴到如下位置



在cmd检查

ssh -t git@gitee.com



1. 创建空白的码云仓库





点击创建即可

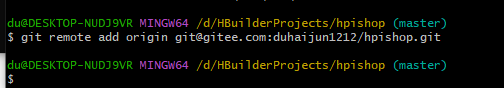
1. 把本地项目上传到码云对应的空白仓库中

创建成功后



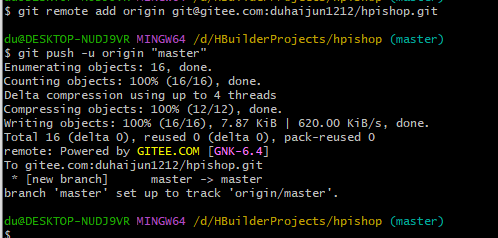
关联远程仓库

git remote add origin git@gitee.com:duhaijun1212/hpishop.git



推送

git push -u origin "master"



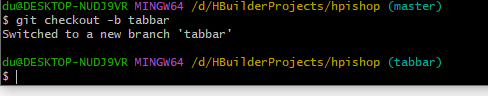
# 创建项目tabBar

## 2.1创建 tabBar 分支

运行如下的命令，基于 master 分支在本地创建 tabBar 子分支，用来开发和 tabBar 相关的功能：

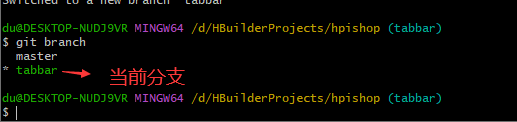
创建并切换分支

git checkout -b tabbar



查看分支

git branch



## 2.2创建 tabBar 页面

在 pages 目录中，创建首页(home)、分类(cate)、购物车(cart)、我的(my) 这 4 个 tabBar 页面。在 HBuilderX 中，可以通过如下的两个步骤，快速新建页面：

1. 在 pages 目录上鼠标右键，选择**新建页面**
2. 在弹出的窗口中，填写**页面的名称**、**勾选 scss 模板**之后，点击创建按钮。截图如下：



## 2.3配置 tabBar 效果

1.资源文件放在static目录中

2. 修改项目根目录中的 pages.json 配置文件，新增 tabBar 的配置节点如下：

{

"tabBar": {

"selectedColor": "#C00000",

"list": [

{

"pagePath": "pages/home/home",

"text": "首页",

"iconPath": "static/tab\_icons/home.png",

"selectedIconPath": "static/tab\_icons/home-active.png"

},

{

"pagePath": "pages/cate/cate",

"text": "分类",

"iconPath": "static/tab\_icons/cate.png",

"selectedIconPath": "static/tab\_icons/cate-active.png"

},

{

"pagePath": "pages/cart/cart",

"text": "购物车",

"iconPath": "static/tab\_icons/cart.png",

"selectedIconPath": "static/tab\_icons/cart-active.png"

},

{

"pagePath": "pages/my/my",

"text": "我的",

"iconPath": "static/tab\_icons/my.png",

"selectedIconPath": "static/tab\_icons/my-active.png"

}

]

}

}

创建完成后，在微信开发者工具中就可以看到效果，如果没有看到，可能是pages.json中home页面不是第一个



## 2.4删除默认的 index 首页

1. 在 HBuilderX 中，把 pages 目录下的 index首页文件夹 删除掉
2. 同时，把 page.json 中记录的 index 首页 路径删除掉
3. 为了防止小程序运行失败，在微信开发者工具中，手动删除 pages 目录下的 index 首页文件夹
4. 同时，把 components 目录下的 uni-link 组件文件夹 删除掉

## 2.5修改导航条的样式效果

1. 打开 pages.json 这个全局的配置文件
2. 修改 globalStyle 节点如下：
3. {
4. "globalStyle": {
5. "navigationBarTextStyle": "white",
6. "navigationBarTitleText": "黑马优购",
7. "navigationBarBackgroundColor": "#C00000",
8. "backgroundColor": "#FFFFFF"
9. }

}

如果navigationBarTitleText没有显示，一般情况下是因为全局覆盖的问题,可以查看home页面是否像我这样设置了navigationBarTitleText，如果在首页也设置了这一属性,那么把这一属性去掉就好了,页面就能显示了

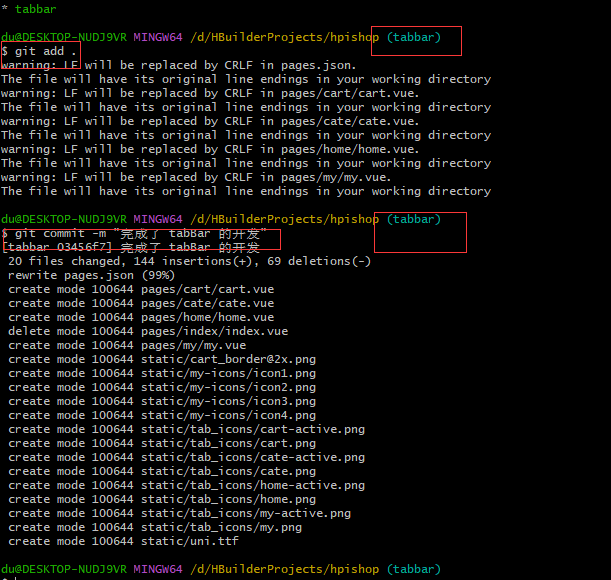


## 2.6分支的提交与合并

1.将本地的 tabbar 分支进行本地的 commit 提交：

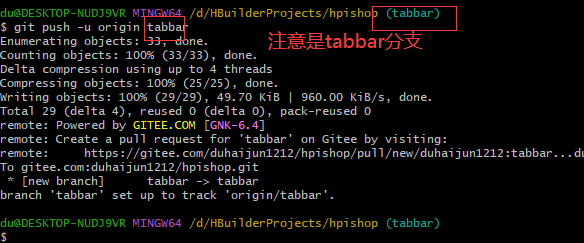
git add .

git commit -m "完成了 tabBar 的开发"



2. 本地的 tabbar 分支推送到远程仓库进行保存：

git push -u origin tabbar



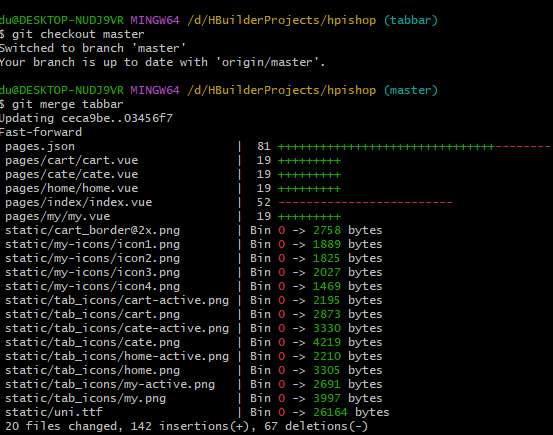
1. 将本地的 tabbar 分支合并到本地的 master 分支

先切换到 master分支，必须在主分支合并

git checkout master

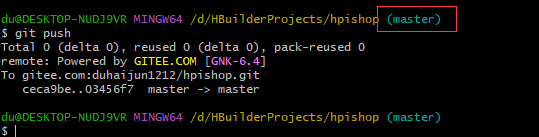
合并

git merge tabbar



1. 将合并后的master分支推送到远程仓库

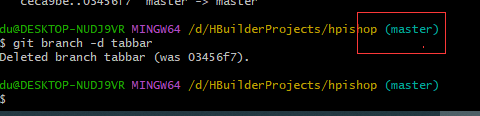
git push



1. 删除本地的 tabbar 分支，因为tabbar分支已经完成使命，想看tabbar分支在远程仓库看

必须在master分支删除，不能再本分支删除 –d代表本地

git branch -d tabbar



# 首页

## 3.1创建 home分支

运行如下的命令，基于 master 分支在本地创建 home 子分支，用来开发和 home 首页相关的功能：

git checkout -b home



## 3.2配置网络请求

由于平台的限制，小程序项目中**不支持 axios**，而且原生的 wx.request() API 功能较为简单，**不支持拦截器**等全局定制的功能。因此，建议在 uni-app 项目中使用 @escook/request-miniprogram 第三方包发起网络数据请求。

**请参考 @escook/request-miniprogram 的官方文档进行安装、配置、使用**

**官方文档：https://www.npmjs.com/package/@escook/request-miniprogram**

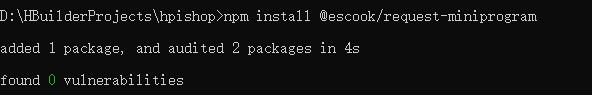
最终，在项目的 main.js 入口文件中，通过如下的方式进行配置：

1. 安装

在根目录下打开cmd

npm init

npm install @escook/request-miniprogram



1. 导入并配置拦截器

在根目录man.js中

// 按需导入 $http 对象

import { $http } from '@escook/request-miniprogram'

// 将按需导入的 $http 挂载到 wx 顶级对象之上，方便全局调用

wx.$http = $http

// 在 uni-app 项目中，可以把 $http 挂载到 uni 顶级对象之上，方便全局调用

uni.$http = $http



配置拦截器

// 请求拦截器,请求开始之前做一些事情

$http.beforeRequest = function(options) {

uni.showLoading({

title: '数据加载中...'

})

}

// 响应拦截器

$http.afterRequest = function() {

uni.hideLoading()

}

## 3.3轮播图区域

### 3.3.1轮播图区域

实现步骤：

1. 在 data 中定义轮播图的数组

data() {

return {

// 这是轮播图的数据列表

swiperList: [],

baseurl:uni.$http.baseUrl,

}

1. 在 onLoad 生命周期函数中调用获取轮播图数据的方法

onLoad() {

// 调用方法，获取轮播图的数据

this.getSwiperList()

}

1. 在 methods 中定义获取轮播图数据的方法

methods: {

async getSwiperList() {

const { data: res } = await uni.$http.get('/ydpub/swiper/my')

console.log(res);

console.log(uni.$http.baseUrl);

// 请求失败

if (res.code !== 200) return uni.$showMsg()

this.swiperList = res.result

},

}

1. 渲染 UI 结构

<swiper :indicator-dots="true" :autoplay="true" :interval="3000" :duration="1000" :circular="true">

<swiper-item v-for="(item,i) in swiperList" :key='i'>

<view class="swiper-item">

<image :src="baseurl+item.path" mode=""></image>

</view>

</swiper-item>

</swiper>

1. 美化 UI 结构：

<style lang="scss">

swiper{

height: 330rpx;

.swiper-item,

image {

width: 100%;

height: 100%;

}

}

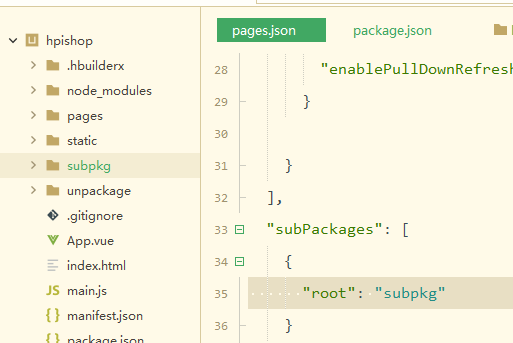
</style>

### 3.3.2配置小程序分包

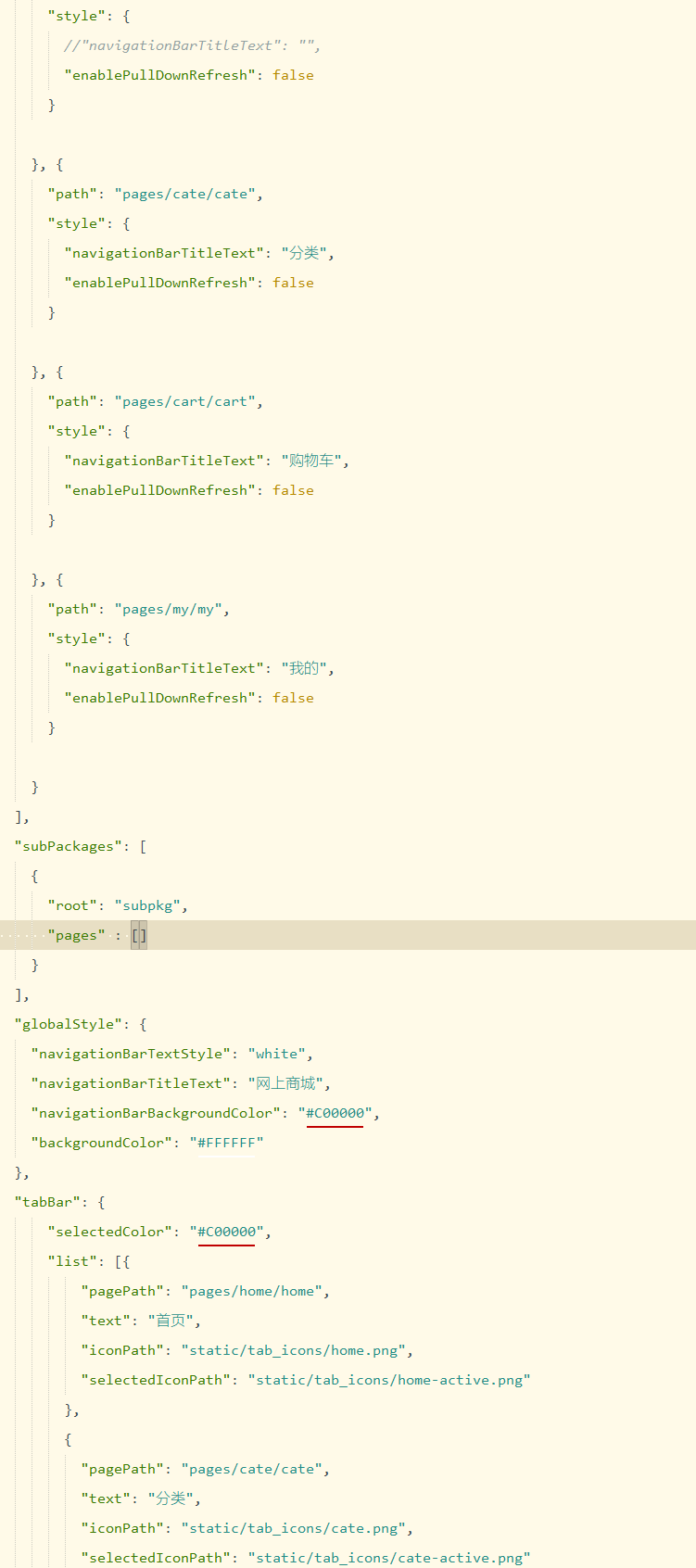
**分包可以减少小程序首次启动时的加载时间**

为此，我们在项目中，把 tabBar 相关的 4 个页面放到主包中，其它页面（例如：商品详情页、商品列表页）放到分包中。在 uni-app 项目中，配置分包的步骤如下：

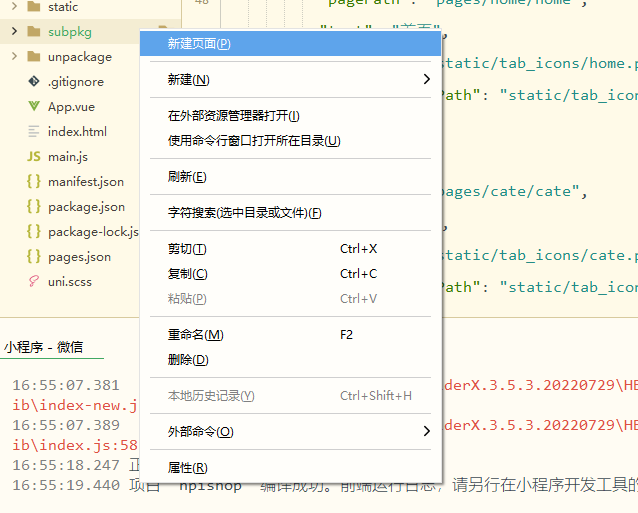
1. 在 在项目根目录中，创建分包的根目录，命名为 subpkg



1. 在pages.json 中，和 pages 节点平级的位置声明 subPackages 节点，用来定义分包相关的结构，在subPackages 中，定义root 节点为分包的根节点subpkg，pages为页面数组



1. 在 subpkg 目录上鼠标右键，点击 新建页面 选项，并填写页面的相关信息：

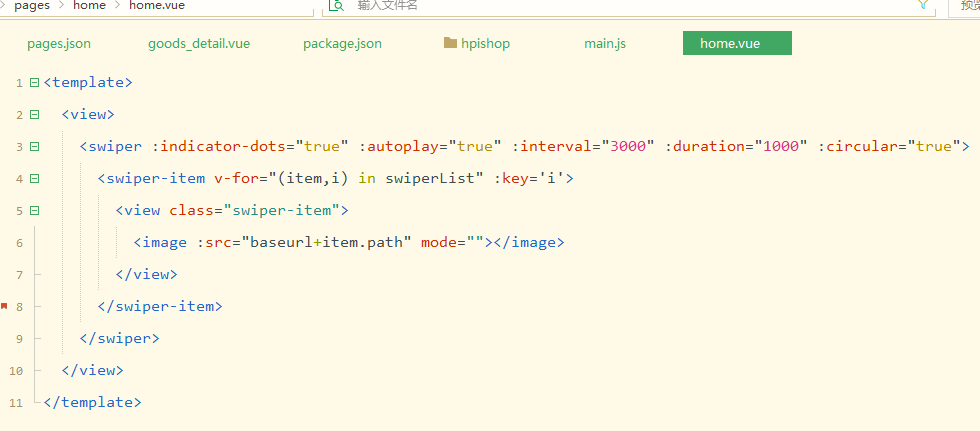




### 3.3.3点击轮播图跳转到商品详情页面

将home  <swiper-item></swiper-item> 节点内的 view 组件，改造为 navigator 导航组件，并动态绑定 url 属性 的值。

1. 改之前



1. 改之后



## 3.3分类导航区域

### 3.3.1获取分类导航的数据

实现思路：

1. 定义 data 数据
2. 在 onLoad 中调用获取数据的方法
3. 在 methods 中定义获取数据的方法

示例代码如下：



### 3.3.2切换到分类导航

1.为 nav-item 绑定点击事件处理函数：



2.定义 navClickHandler 事件处理函数：



## 3.4楼层区域

见代码

## 3.5 home分支的合并与提交

1.将本地的 home 分支进行本地的 commit 提交：

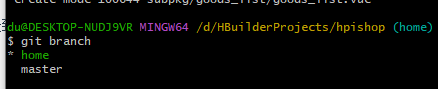
git add .

git commit -m "完成了 home 首页的开发"

2.将本地的 home 分支推送到远程仓库进行保存：

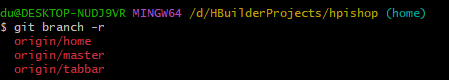
查看分支

git branch



查看远程仓库分支

git branch -r



推送home分支到远程仓库

git push -u origin homeigin home

3. 将本地的 home 分支合并到本地的 master 分支：

git checkout master

git merge home

1. 推送master分支到远程仓库：

$ git push

1. 删除本地的 home 分支

git branch -d home

# 分类

## 4.1创建 cate分支

运行如下的命令，基于 master 分支在本地创建 cate 子分支，用来开发分类页面相关的功能：

git checkout -b cate

## 4.2渲染分类页面的基本结构

见程序

# 搜索组件

## 5.1创建 search分支

git checkout -b search

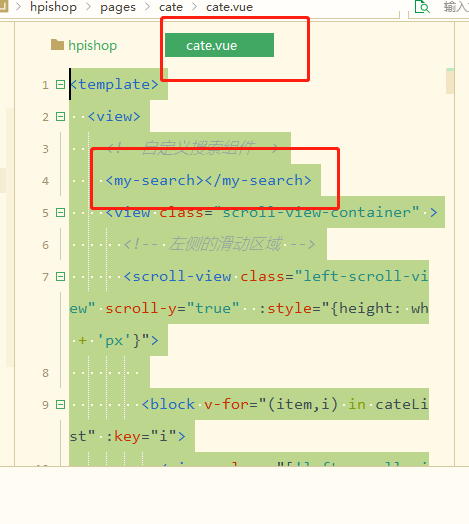
## 5.2自定义搜索组件

### 5.2.1自定义 my-search 组件

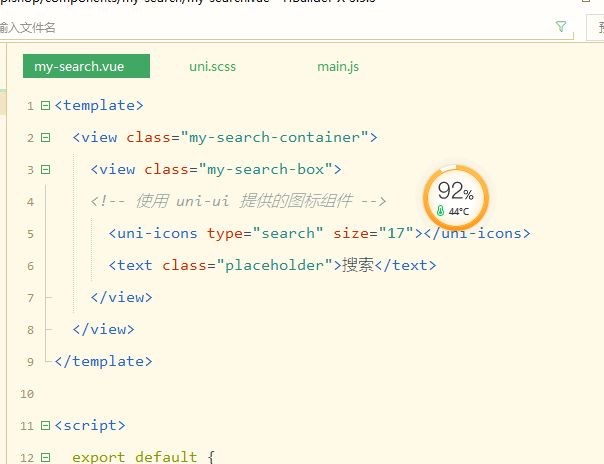
1.在项目中，规定组件必须在components 目录中，我们在项目根目录的 components 目录上，鼠标右键，选择 新建组件，填写组件信息后，最后点击 创建 按钮：



1. 在分类页面的 UI 结构中，直接以标签的形式使用 my-search 自定义组件：



1. 定义 my-search 组件的 UI 结构如下：



**注意：在当前组件中，我们使用 view 组件模拟 input 输入框的效果；并不会在页面上渲染真正的 input 输入框**

1. 美化自定义 search 组件的样式：

参照组件样式

1. 由于自定义的 my-search 组件高度为 50px，因此，需要重新计算分类页面窗口的可用高度：

### 5.2.2通过自定义属性增强组件的通用性

为了增强组件的通用性，我们允许使用者自定义搜索组件的 背景颜色 和 圆角尺寸。

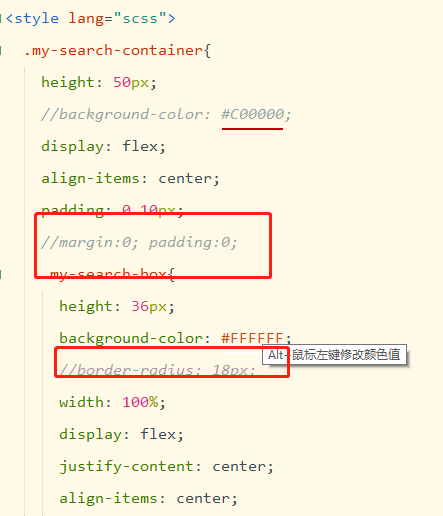
1. 通过 props 定义 bgcolor 和 radius 两个属性，并指定值类型和属性默认值：



1. 通过属性绑定的形式，为 .my-search-container 盒子和 .my-search-box 盒子动态绑定 style 属性：



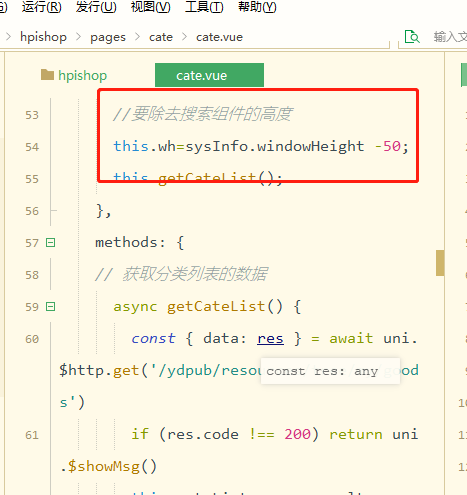
1. 移除对应 scss 样式中的 背景颜色 和 圆角尺寸：



1. 在使用组件的页面动态绑定属性颜色和圆角



1. 由于自定义的 my-search 组件高度为 50px，因此，需要重新计算分类页面窗口的可用高度：



### 5.2.3为自定义组件封装 click 事件

在应用自定义组件的组件上直接使用clik是无效的，因为自定义的组件没有封装clik事件，要使用必须在自定义的组件封装一个clik事件



按照下列步骤执行

* 1. 在 my-search 自定义组件内部，给类名为 .my-search-box 的 view 绑定 click 事件处理函数：



* 1. 在 my-search 自定义组件的 methods 节点中，声明事件处理函数如下



注意：emit（“click”）中为事件名称，看下图，如果<my-search @myclick="gotoSearch" ></my-search>，那没也必须$emit(“myclick”)

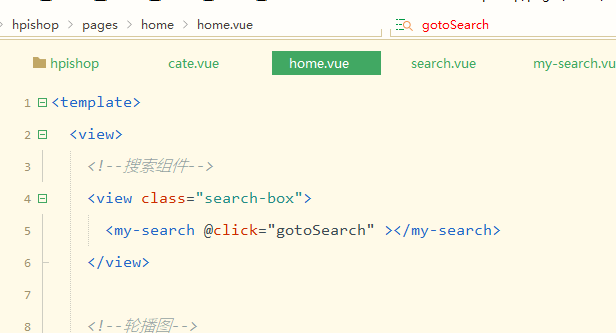


* 1. 在分类页面中使用 my-search 自定义组件时，即可通过 @click 为其绑定点击事件处理函数：



### 5.2.4实现首页(home)搜索组件的吸顶效果

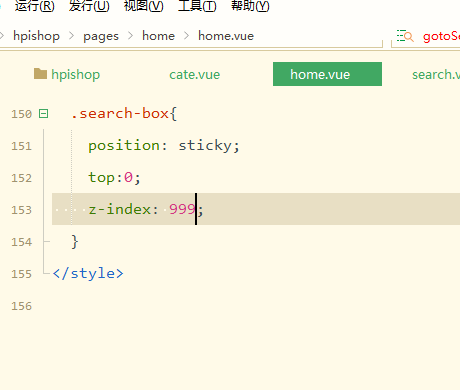
1.在 home 首页定义如下的 UI 结构：



1. 在home 首页定义如下的事件处理函数：



1. 通过如下的样式实现吸顶的效果：



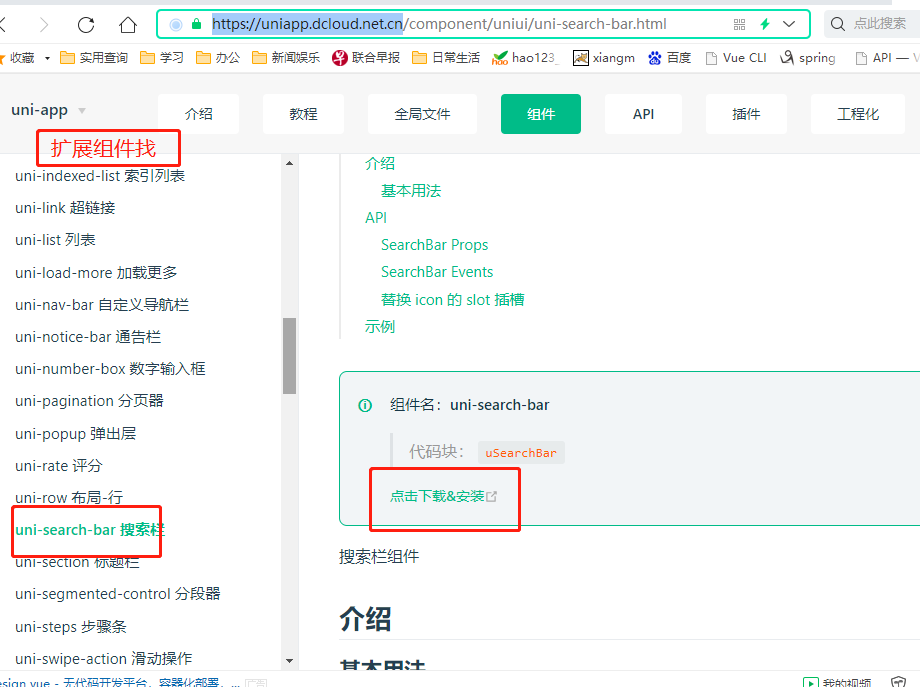
## 5.3搜索建议

### 5.3.1渲染搜索页面的基本结构

1.下载安装uni-search-bar 组件

<uni-search-bar @confirm="search" @input="input" ></uni-search-bar>

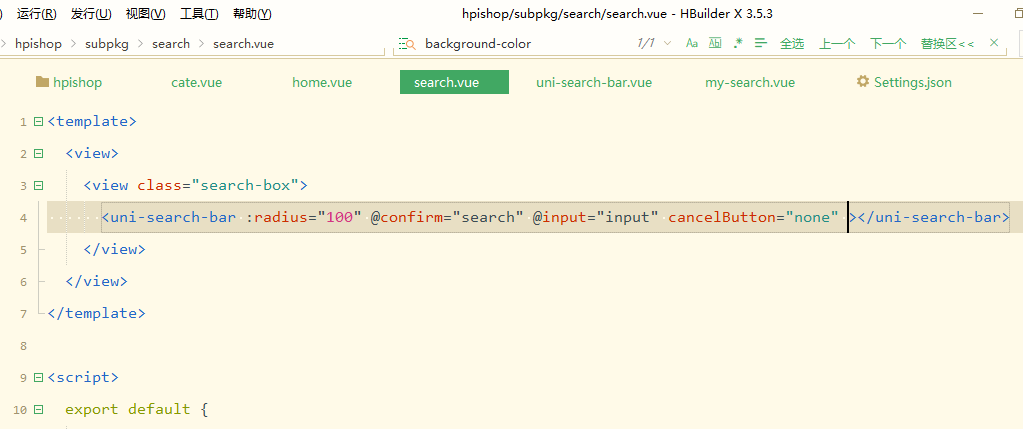
打开uniapp官网，<https://uniapp.dcloud.net.cn/>，在组件栏目找到uni-search-bar组件



如果没有安装需要先安装

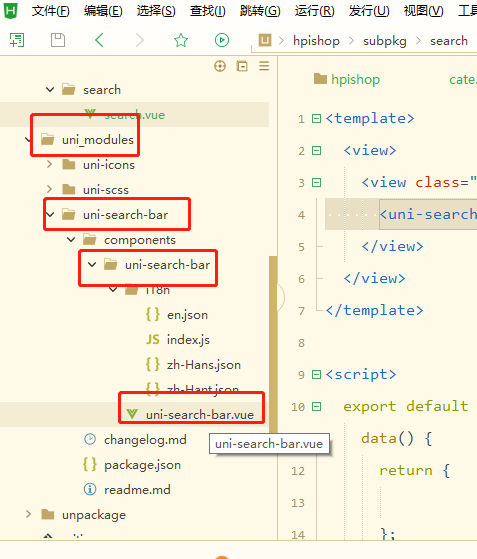
然后重新运行小程序

2.定义如下的 UI 结构：

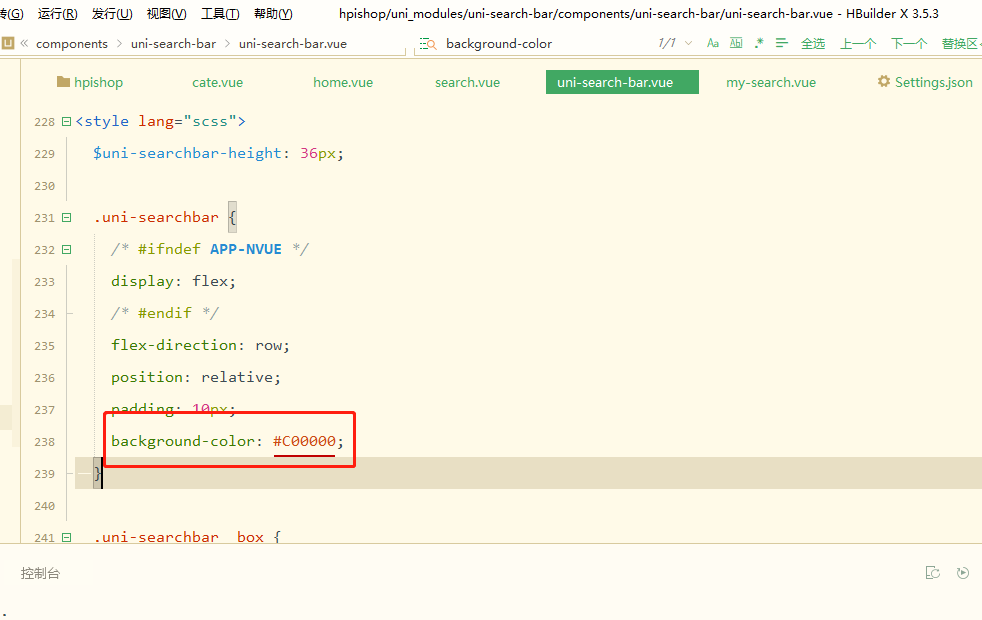


3. 修改 components -> uni-search-bar -> uni-search-bar.vue 组件，将默认的白色搜索背景改为 #C00000 的红色背景：

打开uni-search-bar组件源代码，



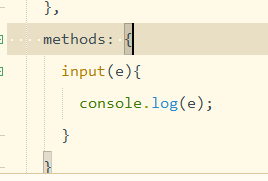
修改如下内容：



1. 实现搜索框的吸顶效果：



1. 定义如下的 input 事件处理函数：

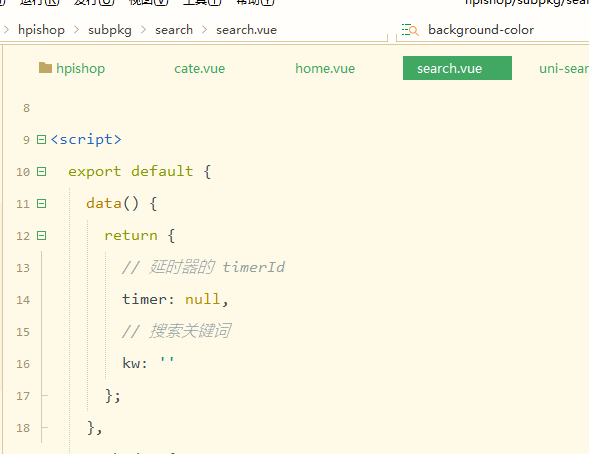


### 5.3.2实现搜索框自动获取焦点

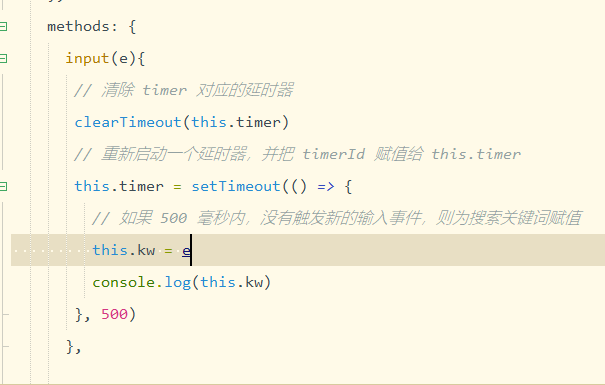


### 5.3.3实现搜索框的防抖处理

1.在 data 中定义防抖的延时器 timerId 如下：



2.修改 input 事件处理函数如下：



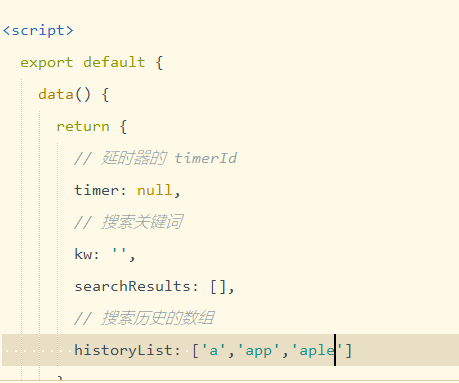
### 5.3.4渲染搜索建议列表

见代码

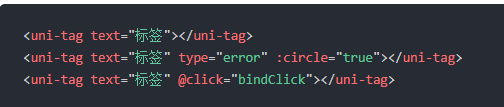
## 5.4搜索历史

### 5.4.1渲染搜索历史记录的基本结构

1.在 data 中定义搜索历史的假数据：



1. 渲染搜索历史区域的 UI 结构：
2. 历史列表组件uni-tag

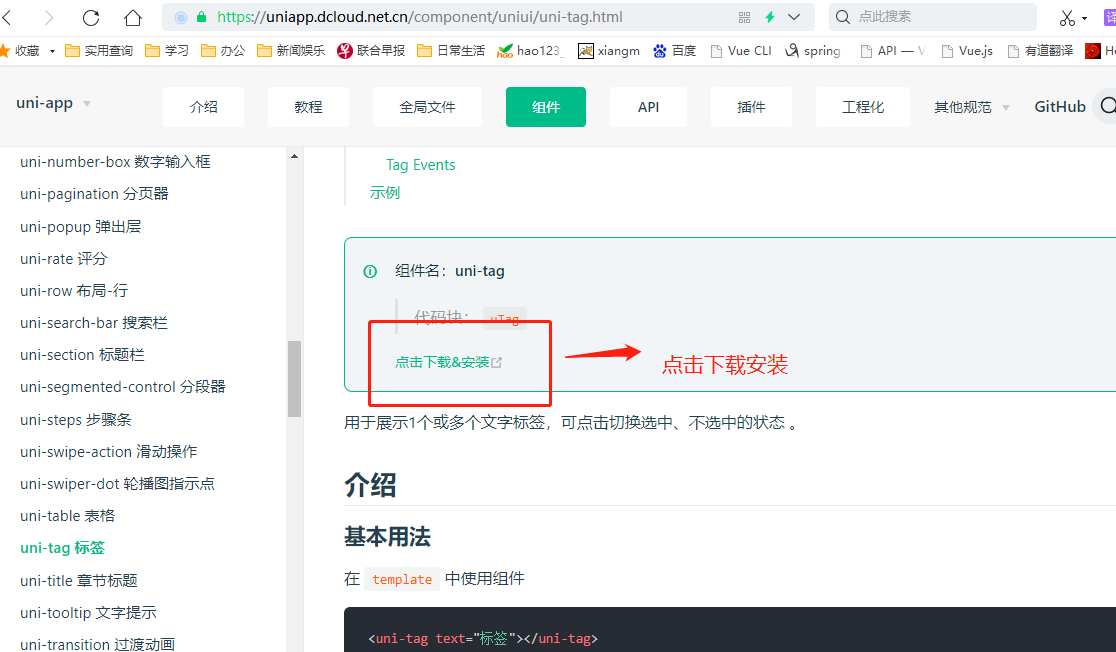


如果没有安装，需要在uni官方下载安装

地址为

<https://uniapp.dcloud.net.cn/component/uniui/uni-tag.html>

打开以后，



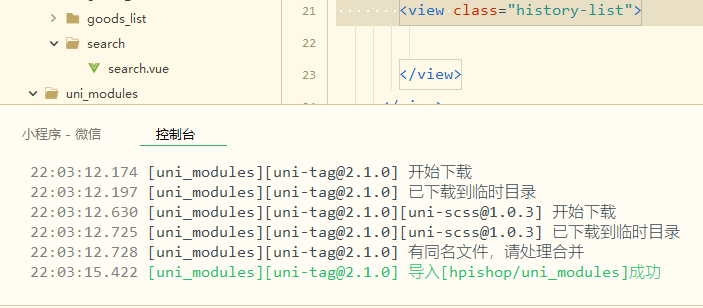
跳转到安装页面







最后点击合并



1. 渲染搜索历史区域的 UI 结构：



1. 美化结构

### 5.4.2解决关键字前后顺序的问题

放在历史记录中的数据（historyList中的数据）最后放在应该放在第一个，要解决这个问题，在定义一个计算方法histories，在计算方法中把historyList数据颠倒，



### 5.4.3解决关键字重复的问题



### 5.4.4将搜索历史记录持久化存储到本地

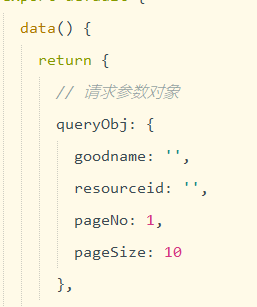


# 商品列表

## 6.1创建 goodslist 分支



## 6.2定义请求参数对象



## 6.3渲染商品列表结构

在页面中，通过 v-for 指令，循环渲染出商品的 UI 结构：



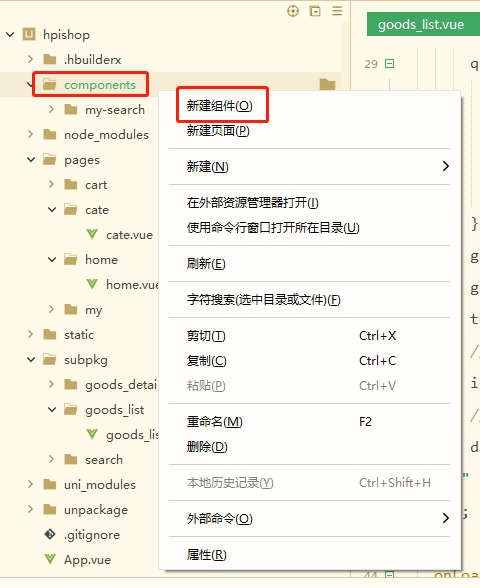
了防止某些商品的图片不存在，需要在 data 中定义一个默认的图片：



美化商品列表的 UI 结构

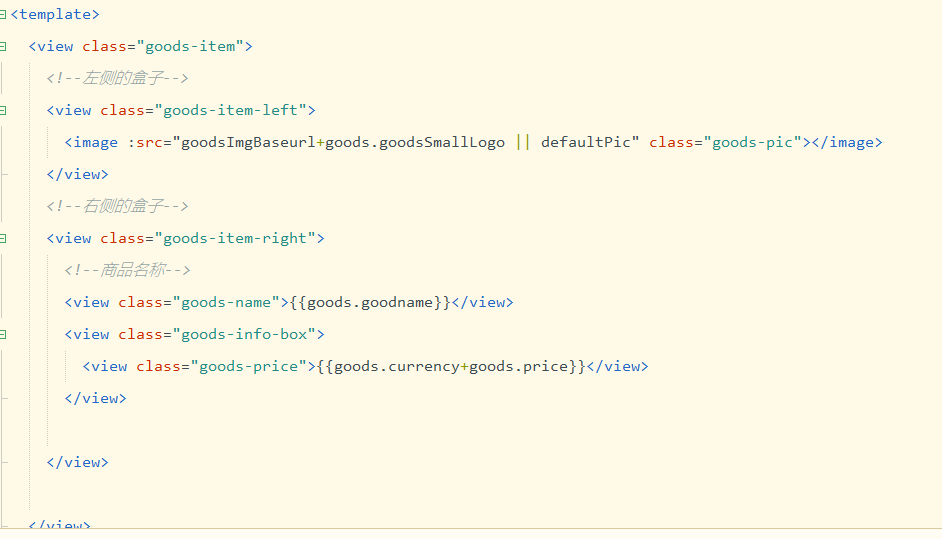
## 6.3把商品 item 项封装为自定义组件

1.在 components 目录上鼠标右键，选择 新建组件：





2. 将 goods\_list 页面中，关于商品 item 项相关的 UI 结构、样式、data 数据，封装到 my-goods 组件中：





<style lang="scss">

.goods-item{

display: flex;

padding: 10px 5px;

border-bottom: 1px solid #f0f0f0;

.goods-item-left{

margin-right: 5px;

.goods-pic{

width: 100px;

height: 100px;

display: block;

}

}

.goods-item-right{

display: flex;

flex-direction: column;

justify-content: space-between;

.goods-name{

font-size: 13px;

}

.goods-info-box{

.goods-price{

color: #C00000;

font-size: 16px;

}

}

}

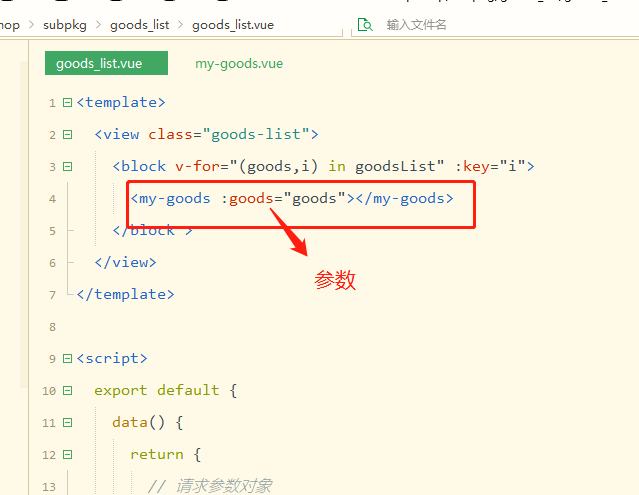
}

</style>

2. 传递参数，在my-goods组件中创建props节点，传递goods



引用组件



## 6.4使用过滤器处理价格

1．在 my-goods 组件中，和 data 节点平级，声明 filters 过滤器节点如下：



2．在渲染商品价格的时候，通过管道符 | 调用过滤器：

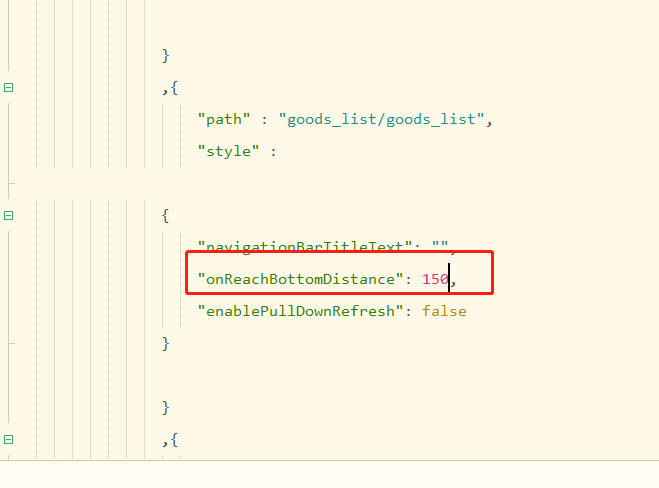


## 6.5上拉加载更多

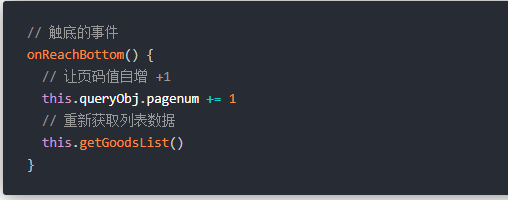
### 6.5.1初步实现上拉加载更多：

1.打开项目根目录中的 pages.json 配置文件，为 subPackages 分包中的 goods\_list 页面配置上拉触底的距离：

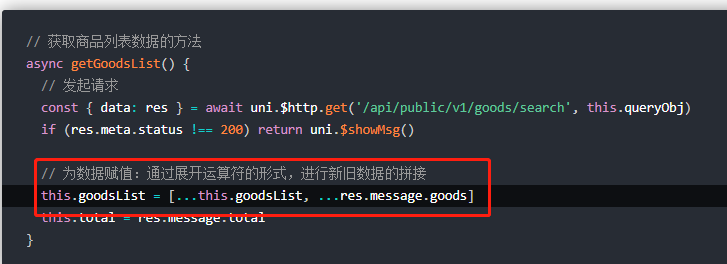




1. 在 goods\_list 页面中，和 methods 节点平级，声明 onReachBottom 事件处理函数，用来监听页面的上拉触底行为：



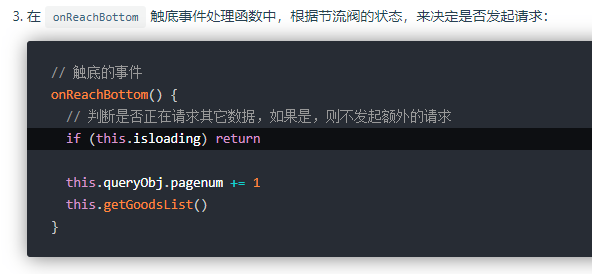
1. 改造 methods 中的 getGoodsList 函数，当列表数据请求成功之后，进行新旧数据的拼接处理：



### 6.5.2通过节流阀防止发起额外的请求：

在上次数据还没有请求完成前，又一次下拉请求数据，这样影响效率，为了处理这个问题，定义一个变量来处理





### 6.5.3判断数据是否加载完毕：



## 6.6下拉涮新

1. 在 pages.json 配置文件中，为当前的 goods\_list 页面单独开启下拉刷新效果：

"enablePullDownRefresh": true,

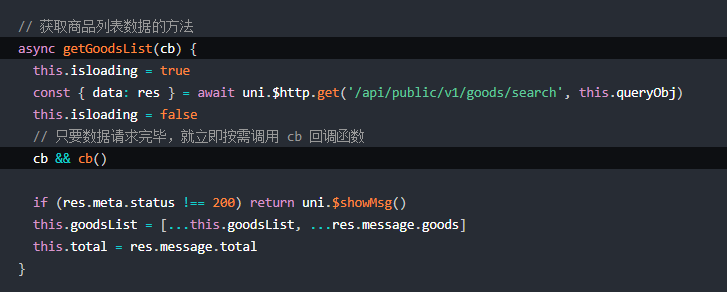
"backgroundColor": "#F8F8F8"



1. 监听页面的 onPullDownRefresh 事件处理函数



1. 修改 getGoodsList 函数，接收 cb 回调函数并按需进行调用：



## 6.7点击商品 item 项跳转到详情页面



## 6.8提交goodslist分支



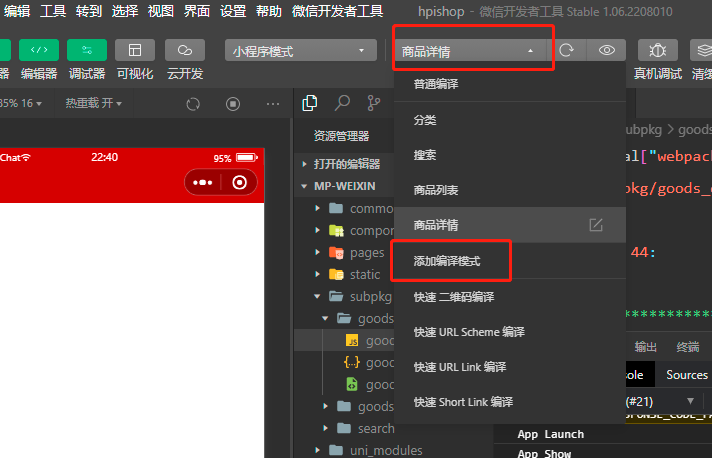
# 商品详情

## 7.1创建 goodsdetail 分支



## 7.2添加商品详情页的编译模式

1.在微信开发者工具中，点击工具栏上的编译模式下拉菜单，选择 添加编译模式 选项

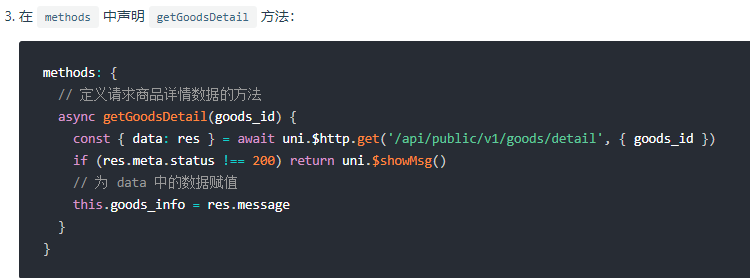


1. 选择 启动页面 的路径，并填写了 启动参数 之后，点击 确定 按钮，添加详情页面的编译模式：



## 7.3获取商品详情数据

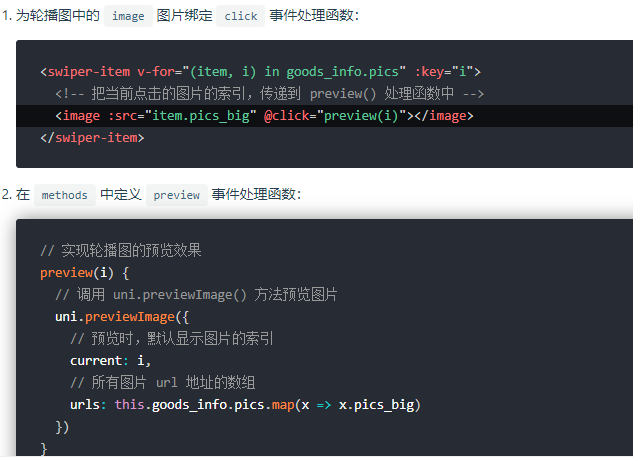




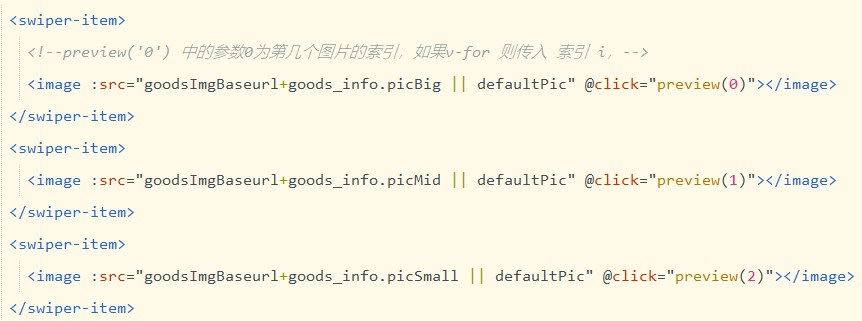
## 7.4渲染商品详情页的 UI 结构



## 7.5实现轮播图预览效果



我的轮播图不是数组，是加了三张图片





## 7.6渲染商品信息区域

### 7.6.1渲染商品基本信息：



2.美化商品信息区域的样式：

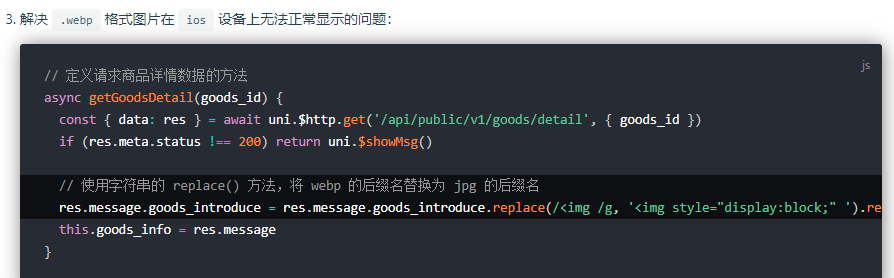
### 7.6.2渲染商品介绍详情信息：

1.在页面结构中，使用 rich-text 组件，将带有 HTML 标签的内容，渲染为小程序的页面结构：



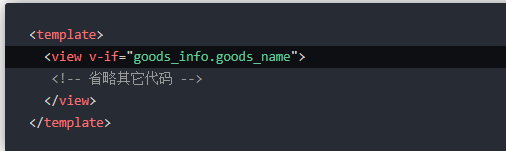
下列两项非必要





### 7.6.3解决商品价格闪烁的问题：

1. 导致问题的原因：在商品详情数据请求回来之前，data 中 goods\_info 的值为 {}，因此初次渲染页面时，会导致 商品价格、商品名称 等闪烁的问题。
2. 解决方案：判断 goods\_info.goods\_name 属性的值是否存在，从而使用 v-if 指令控制页面的显示与隐藏：





## 7.7渲染详情页底部的商品导航区域

### 7.7.1渲染商品导航区域的 UI 结构：

**基于 uni-ui 提供的**[GoodsNav](https://ext.dcloud.net.cn/plugin?id=865)**组件来实现商品导航区域的效果，地址为：<https://ext.dcloud.net.cn/plugin?id=865>**



1. 在 data 中，通过 options 和 buttonGroup 两个数组，来声明商品导航组件的按钮配置对象：

options: [{

icon: 'shop',

text: '店铺',

infoBackgroundColor: '#007aff',

infoColor: "red"

}, {

icon: 'cart',

text: '购物车',

info: 0

}],

buttonGroup: [{

text: '加入购物车',

backgroundColor: '#ff0000',

color: '#fff'

},

{

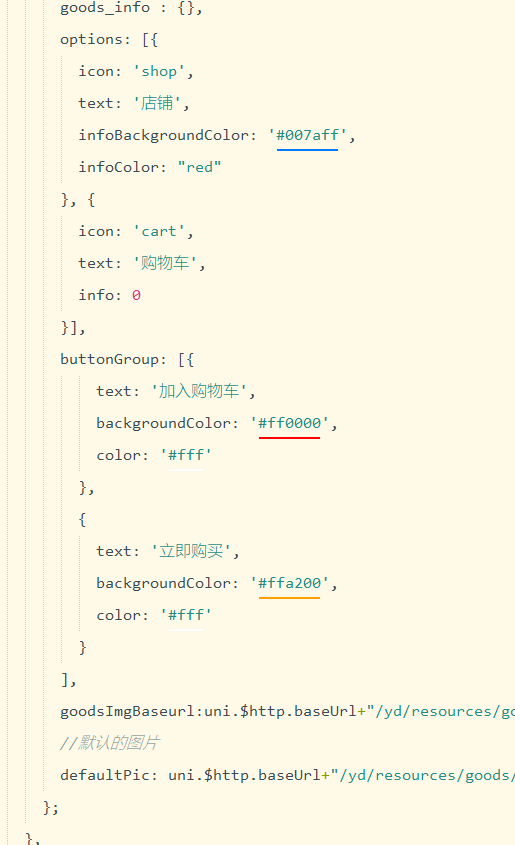
text: '立即购买',

backgroundColor: '#ffa200',

color: '#fff'

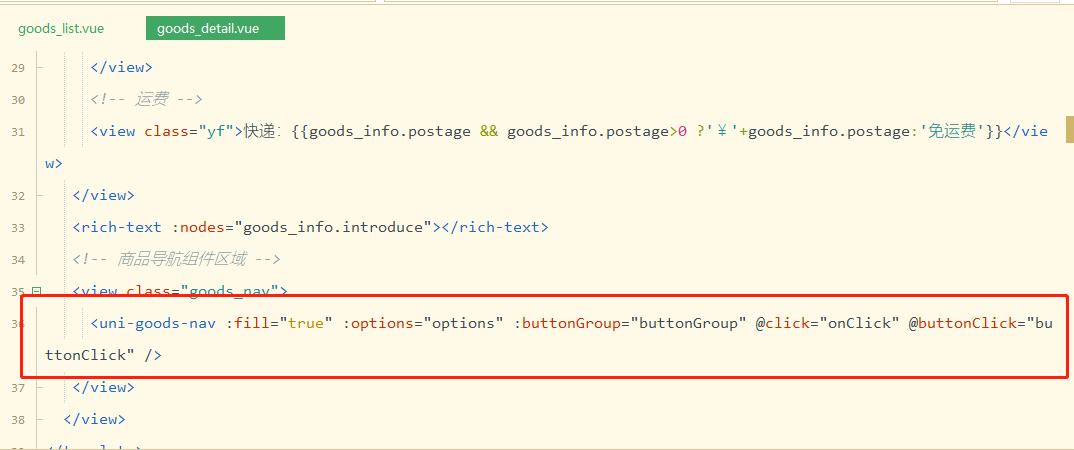
}

],

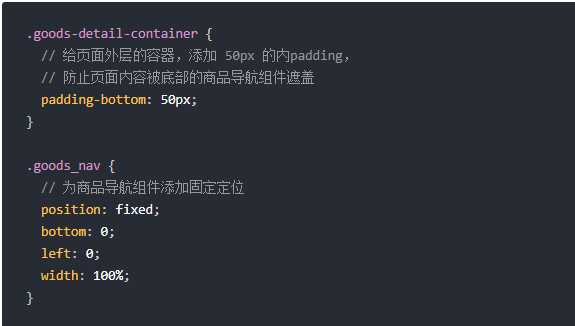


1. 在页面中使用 uni-goods-nav 商品导航组件：

<uni-goods-nav :fill="true" :options="options" :buttonGroup="buttonGroup" @click="onClick" @buttonClick="buttonClick" />

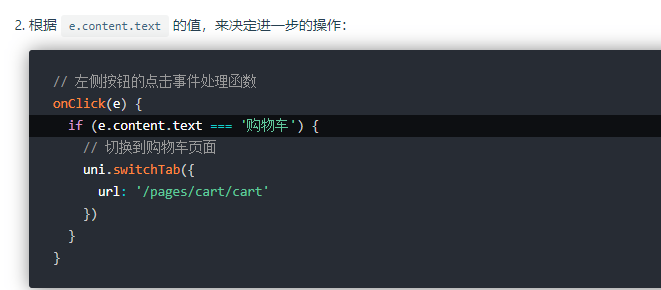


3. 美化商品导航组件，使之固定在页面最底部：



### 7.7.2点击跳转到购物车页面：





## 7.8提交goodsdetail分支





# 加入购物车

## 8.1创建 cart 分支

运行如下的命令，基于 master 分支在本地创建 cart 子分支，用来开发购物车相关的功能：

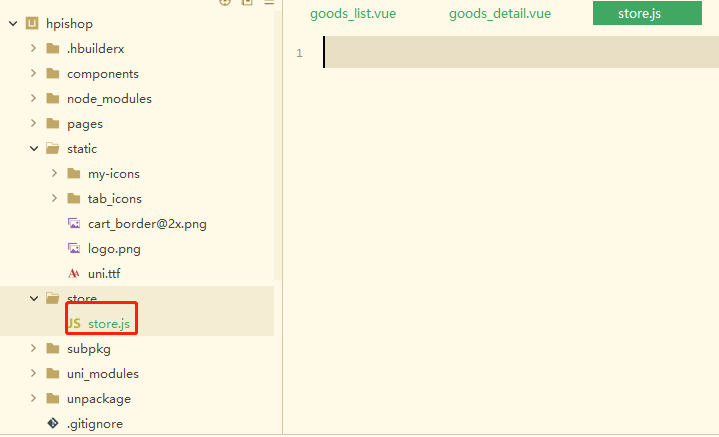
git checkout -b cart

## 8.2配置 vuex

1. 在项目根目录中创建 store 文件夹，专门用来存放 vuex 相关的模块

2. 在 store 目录上鼠标右键，选择 新建 -> js文件，新建 store.js 文件：





1. 在 store.js 中按照如下 4 个步骤**初始化 Store 的实例对象**：



1. 在 main.js 中导入 store 实例对象并挂载到 Vue 的实例上



## 8.3创建购物车的 store 模块

1. 在 store 目录上鼠标右键，选择 新建 -> js文件，创建购物车的 store 模块，命名为 cart.js：



1. 在 cart.js 中，初始化如下的 vuex 模块：



1. 在 store/store.js 模块中，导入并挂载购物车的 vuex 模块，示例代码如下：



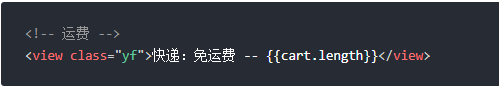
## 8.4在商品详情页中使用 Store 中的数据

1. 在 goods\_detail.vue 页面中，修改 <script></script> 标签中的代码如下：



**注意：今后无论映射 mutations 方法，还是 getters 属性，还是 state 中的数据，都需要指定模块的名称，才能进行映射。**

2.在页面渲染时，可以直接使用映射过来的数据，例如：

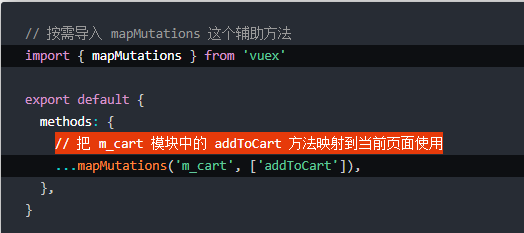


## 8.5实现加入购物车的功能

1. 在 store 目录下的 cart.js 模块中，封装一个将商品信息加入购物车的 mutations 方法，命名为 addToCart。示例代码如下：



2. 在商品详情页面中，通过 mapMutations 这个辅助方法，把 vuex 中 m\_cart 模块下的 addToCart 方法映射到当前页面：

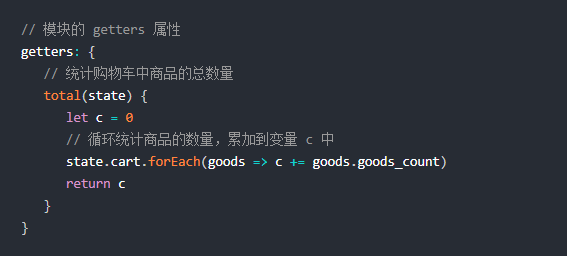


3. 为商品导航组件 uni-goods-nav 绑定 @buttonClick="buttonClick" 事件处理函数：

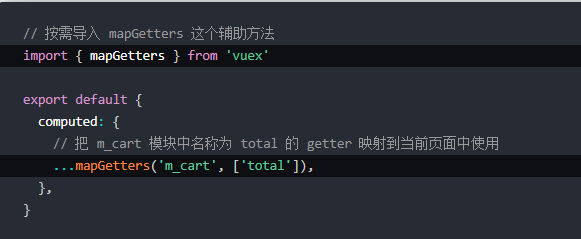


## 8.6动态统计购物车中商品的总数量

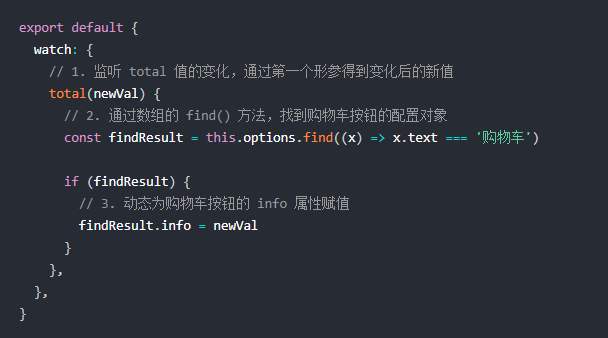
1. 在 cart.js 模块中，在 getters 节点下定义一个 total 方法，用来统计购物车中商品的总数量



2. 在商品详情页面的 script 标签中，按需导入 mapGetters 方法并进行使用：



3. 通过 watch 侦听器，监听计算属性 total 值的变化，从而**动态为购物车按钮的徽标赋值**：



## 8.7持久化存储购物车中的商品

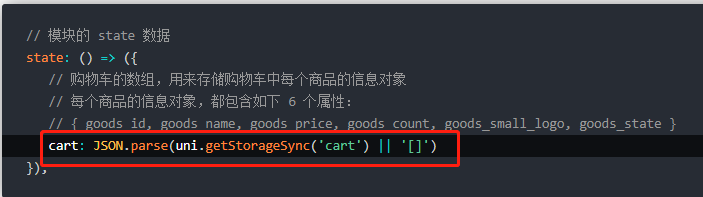
1. 在 cart.js 模块中，声明一个叫做 saveToStorage 的 mutations 方法，此方法负责将购物车中的数据持久化存储到本地



2. 修改 mutations 节点中的 addToCart 方法，在处理完商品信息后，调用步骤 1 中定义的 saveToStorage 方法：



3. 修改 cart.js 模块中的 state 函数，读取本地存储的购物车数据，对 cart 数组进行初始化：

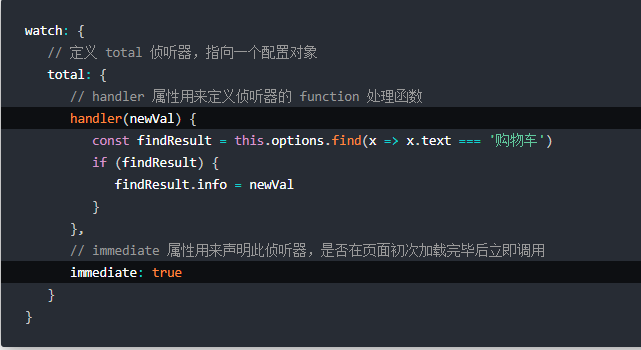


## 8.8 侦听器立即调用（优化商品详情页的 total 侦听器）

1. 使用**普通函数的形式**定义的 watch 侦听器，**在页面首次加载后不会被调用**。因此导致了商品详情页在首次加载完毕之后，不会将商品的总数量显示到商品导航区域：



2. 为了防止这个上述问题，可以使用**对象的形式**来定义 watch 侦听器（详细文档请参考 Vue 官方的 [watch 侦听器](https://cn.vuejs.org/v2/api/#watch)教程），示例代码如下：



## 8.9动态为 tabBar 页面设置数字徽标

**需求描述：从商品详情页面导航到购物车页面之后，需要为 tabBar 中的购物车动态设置数字徽标。**

**1.** 把 Store 中的 total 映射到 cart.vue 中使用：



**2.** 在页面刚显示出来的时候，立即调用 setBadge 方法，为 tabBar 设置数字徽标：



**3.** 在 methods 节点中，声明 setBadge 方法如下，通过 uni.setTabBarBadge() 为 tabBar 设置数字徽标：



## 8.10将设置 tabBar 徽标的代码抽离为 mixins

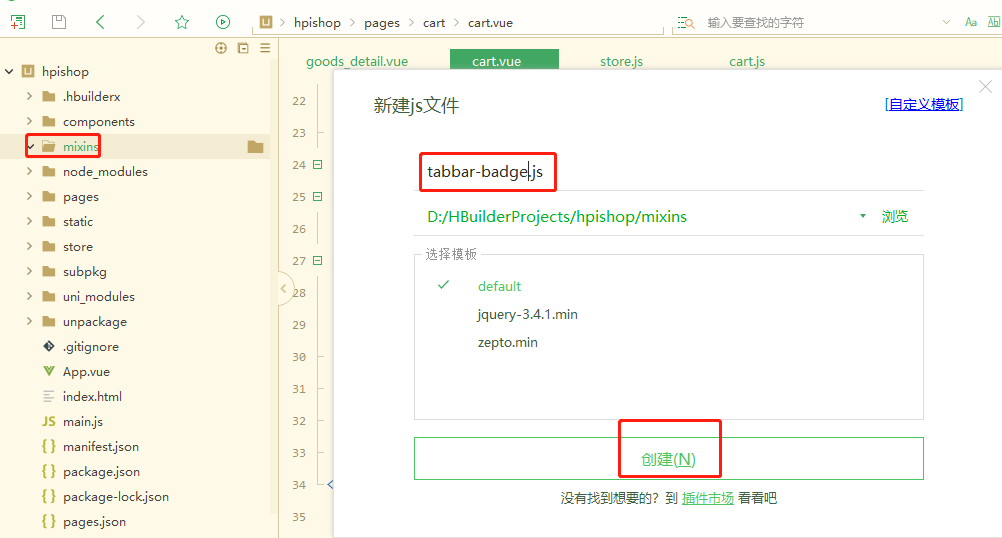
**注意：除了要在 cart.vue 页面中设置购物车的数字徽标，还需要在其它 3 个 tabBar 页面中，为购物车设置数字徽标。**

**此时可以使用 Vue 提供的**[mixins](https://cn.vuejs.org/v2/guide/mixins.html" \t "_blank)**特性，提高代码的可维护性。**

**Mixins场景描述：**

当我们存在多个组件中的数据或者功能很相近时，我们就可以利用mixins将公共部分提取出来，通过 mixins封装的函数，组件调用他们是不会改变函数作用域外部的。

**1.** 在项目根目录中新建 mixins 文件夹，并在 mixins 文件夹之下新建 tabbar-badge.js 文件，用来把设置 tabBar 徽标的代码封装为一个 mixin 文件：



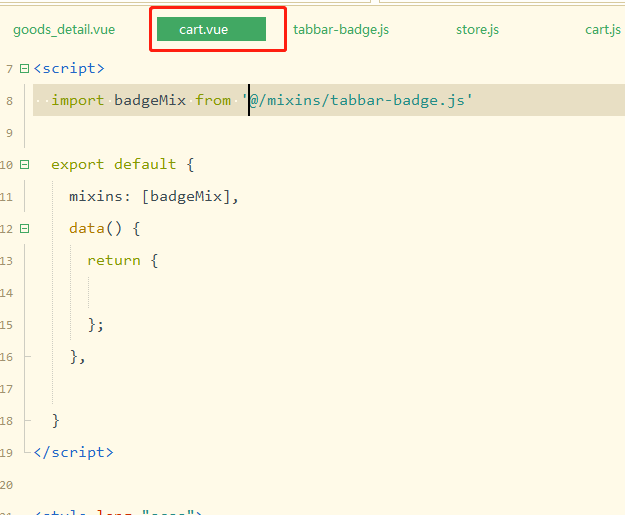
内容为：把原cart.vue （即本章8.9动态为 tabBar 页面设置数字徽标）写的放在这里，如下图：

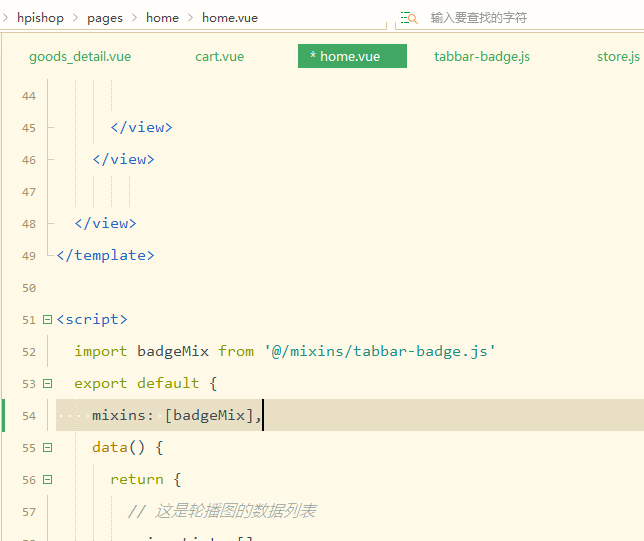


**2.** 修改 home.vue，cate.vue，cart.vue，my.vue 这 4 个 tabBar 页面的源代码，分别导入 @/mixins/tabbar-badge.js 模块并进行使用：



**例如：**





# 购物车页面

**创建购物车页面的编译模式略，参看前面的介绍**

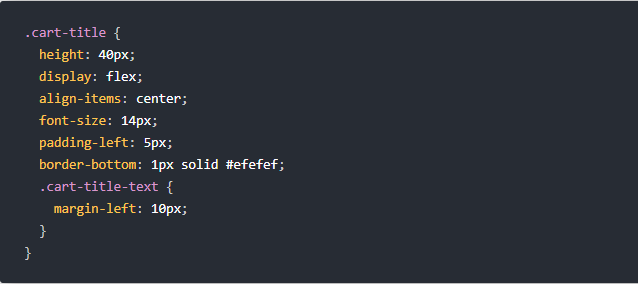
## 9.1商品列表区域

### 9.1.1渲染购物车商品列表的标题区域：

**1.** 定义如下的 UI 结构：



**2.** 美化样式：

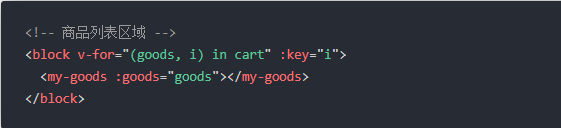


### 9.1.2渲染购物车商品列表的标题区域：

**1.** 通过 mapState 辅助函数，将 Store 中的 cart 数组映射到当前页面中使用

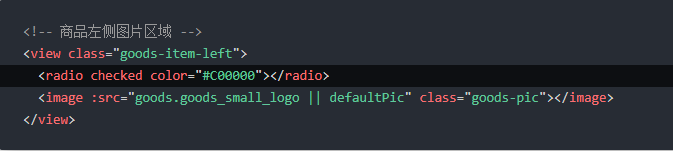


**2.** 在 UI 结构中，通过 v-for 指令循环渲染自定义的 my-goods 组件：

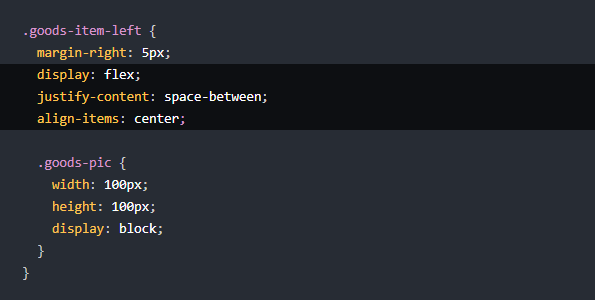


### 9.1.3为 my-goods 组件封装 radio 勾选状态：

**1.** 打开 my-goods.vue 组件的源代码，为商品的左侧图片区域添加 radio 组件：



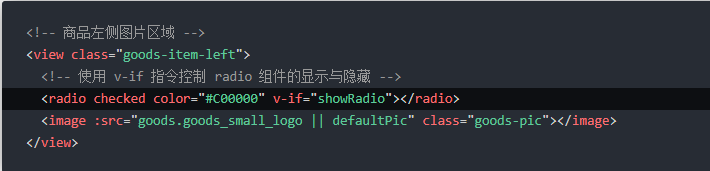
**2.** 给类名为 goods-item-left 的 view 组件添加样式，实现 radio 组件和 image 组件的左右布局：



**3.** 封装名称为 showRadio 的 props 属性，来控制当前组件中是否显示 radio 组件：



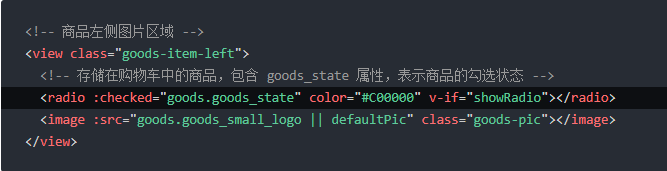
**4.** 使用 v-if 指令控制 radio 组件的按需展示：



**5.** 在 cart.vue 页面中的商品列表区域，指定 :show-radio="true" 属性，从而显示 radio 组件：

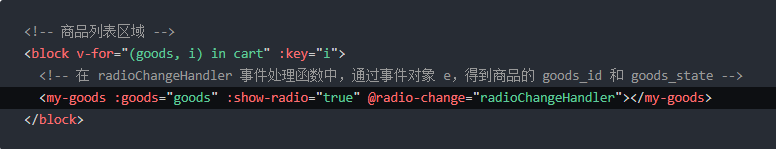


6. 修改 my-goods.vue 组件，动态为 radio 绑定选中状态：

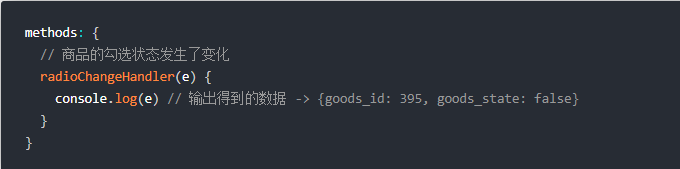


### 9.1.4为 my-goods 组件封装 radio-change 事件：

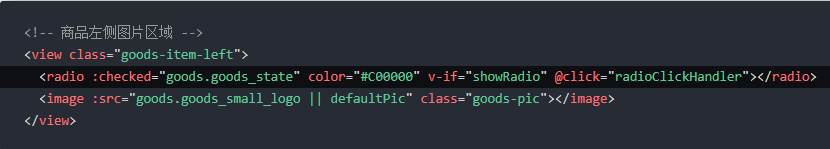
**1.** 当用户点击 radio 组件，**希望修改当前商品的勾选状态**，此时用户可以为 my-goods 组件绑定 @radio-change 事件，从而获取当前商品的 goods\_id 和 goods\_state：



定义 radioChangeHandler 事件处理函数如下：



**2.** 在 my-goods.vue 组件中，为 radio 组件绑定 @click 事件处理函数如下：

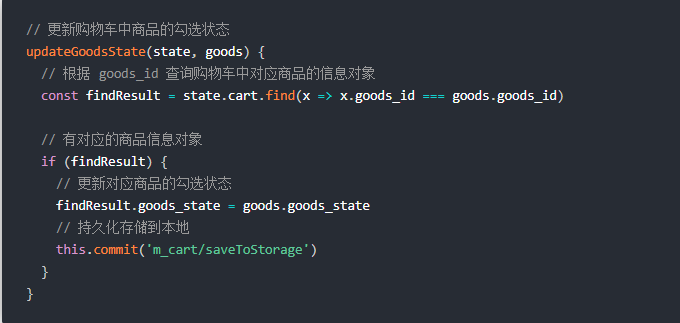


**3.** 在 my-goods.vue 组件的 methods 节点中，定义 radioClickHandler 事件处理函数：

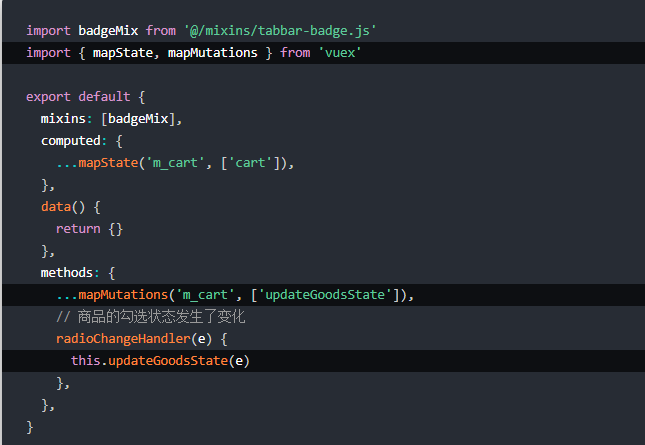


### 9.1.5修改购物车中商品的勾选状态：

**1.** 在 store/cart.js 模块中，声明如下的 mutations 方法，用来修改对应商品的勾选状态：



**2.** 在 cart.vue 页面中，导入 mapMutations 这个辅助函数，从而将需要的 mutations 方法映射到当前页面中使用：



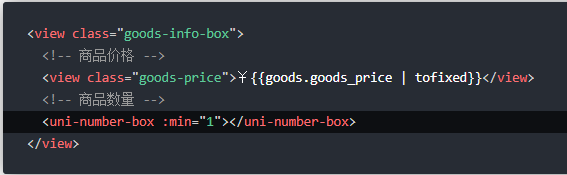
### 9.1.6为 my-goods 组件封装 NumberBox：

**注意：**[NumberBox](https://ext.dcloud.net.cn/plugin?id=31)**组件是 uni-ui 提供的**

**地址为：**[**https://ext.dcloud.net.cn/plugin?id=31**](https://ext.dcloud.net.cn/plugin?id=31)**，如果没有安装则需要安装**



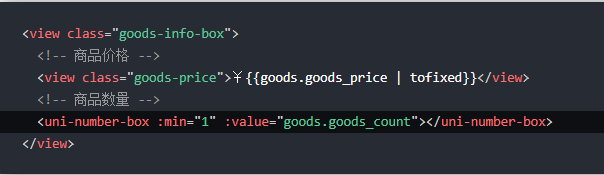
**1.** 修改 my-goods.vue 组件的源代码，在类名为 goods-info-box 的 view 组件内部渲染 NumberBox 组件的基本结构：



**2.** 美化页面的结构：



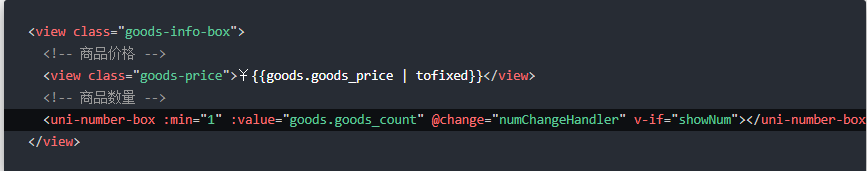
**3.** 在 my-goods.vue 组件中，动态为 NumberBox 组件绑定商品的数量值：



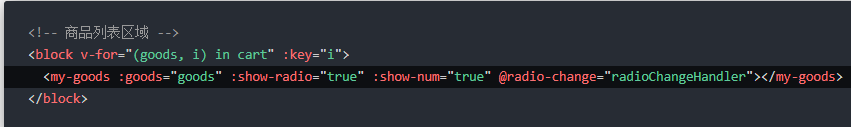
**4.** 在 my-goods.vue 组件中，封装名称为 showNum 的 props 属性，来控制当前组件中是否显示 NumberBox 组件：



**5.** 在 my-goods.vue 组件中，使用 v-if 指令控制 NumberBox 组件的按需展示：

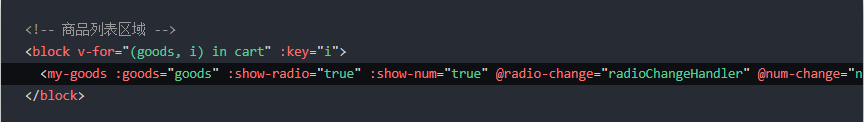


**6.** 在 cart.vue 页面中的商品列表区域，指定 :show-num="true" 属性，从而显示 NumberBox 组件：



### 9.1.7为 my-goods 组件封装 num-change 事件：

**1.** 当用户修改了 NumberBox 的值以后，希望将最新的商品数量更新到购物车中，此时用户可以为 my-goods 组件绑定 @num-change 事件，从而获取当前商品的 goods\_id 和 goods\_count：



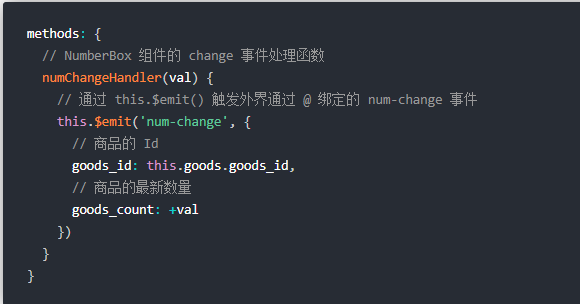
定义 numberChangeHandler 事件处理函数如下：



**2.** 在 my-goods.vue 组件中，为 uni-number-box 组件绑定 @change 事件处理函数如下：



**3.** 在 my-goods.vue 组件的 methods 节点中，定义 numChangeHandler 事件处理函数：



### 9.1.8解决 NumberBox 数据不合法的问题：

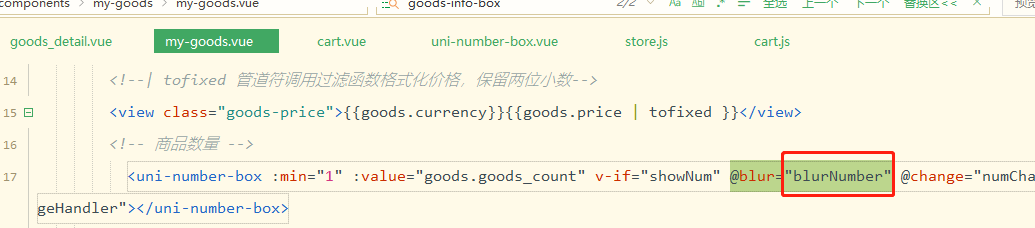
**问题说明：当用户在 NumberBox 中输入字母等非法字符之后，会导致 NumberBox 数据紊乱的问题**

**1.** 打开项目根目录中 components/uni-number-box/uni-number-box.vue 组件，修改 methods 节点中的 \_onBlur 函数如下（不建议）



**还有一种方法，不该源代码**

**给uni-number-box组件绑定@blur="blurNumber"，**



**然后创建一个方法，处理输入是字符的问题，建议这种**



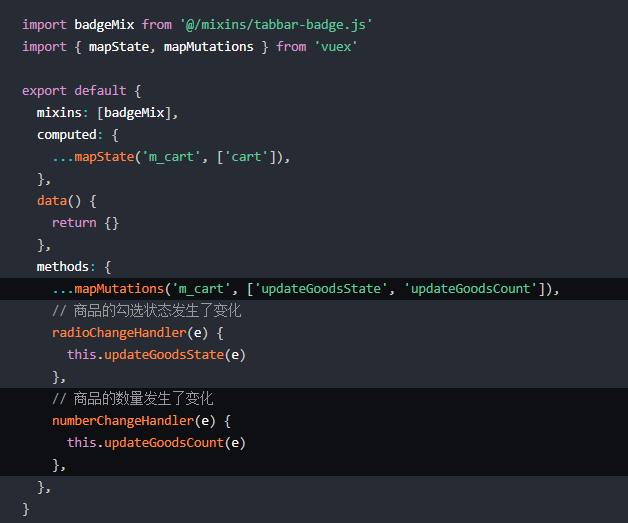
**2.** 修改完毕之后，用户输入小数会被转化为整数，用户输入非法字符会被替换为默认值 1

### 9.1.9修改购物车中商品的数量：

**1.** 在 store/cart.js 模块中，声明如下的 mutations 方法，用来修改对应商品的数量：



**2.** 在 cart.vue 页面中，通过 mapMutations 这个辅助函数，将需要的 mutations 方法映射到当前页面中使用：



### 9.1.10渲染滑动删除的 UI 效果：

**滑动删除需要用到 uni-ui 的 uni-swipe-action 组件和 uni-swipe-action-item。详细的官方文档请参考**[SwipeAction 滑动操作](https://ext.dcloud.net.cn/plugin?id=181)**。**

**地址为：**[**https://ext.dcloud.net.cn/plugin?id=181**](https://ext.dcloud.net.cn/plugin?id=181)



**如果没有需要安装**

**1.** 改造 cart.vue 页面的 UI 结构，将商品列表区域的结构修改如下（可以使用 **uSwipeAction** 代码块快速生成基本的 UI 结构）：



**<!-- uni-swipe-action 是最外层包裹性质的容器 -->**

**<uni-swipe-action>**

**<block v-for="(goods, i) in cart" :key="i">**

**<!-- uni-swipe-action-item 可以为其子节点提供滑动操作的效果。需要通过 options 属性来指定操作按钮的配置信息 -->**

**<uni-swipe-action-item :right-options="options" @click="swipeActionClickHandler(goods)">**

**<my-goods :goods="goods" :showRadio="true" :showNum="true" @radio-change="radioChangeHandler"**

**@num-change="numberChangeHandler"></my-goods>**

**</uni-swipe-action-item>**

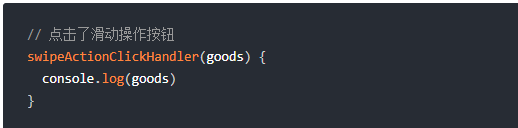
**</block>**

**</uni-swipe-action>**

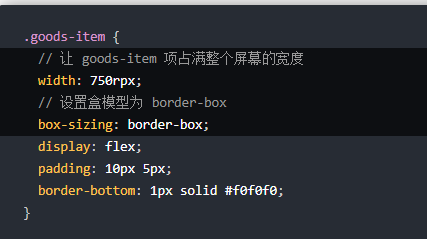
**2.** 在 data 节点中声明 options 数组，用来定义操作按钮的配置信息：



**3.** 在 methods 中声明 uni-swipe-action-item 组件的 @click 事件处理函数：

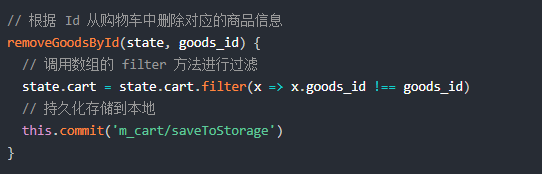


**4.** 美化 my-goods.vue 组件的样式：

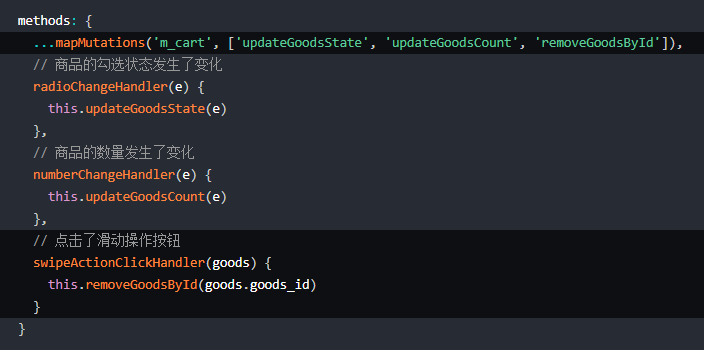


### 9.1.11实现滑动删除的功能：

**1.** 在 store/cart.js 模块的 mutations 节点中声明如下的方法，从而根据商品的 Id 从购物车中移除对应的商品



**2.** 在 cart.vue 页面中，使用 mapMutations 辅助函数，把需要的方法映射到当前页面中使用：



## 9.2收货地址区域