30 现在流行瀑布流：开发实现商城首页面

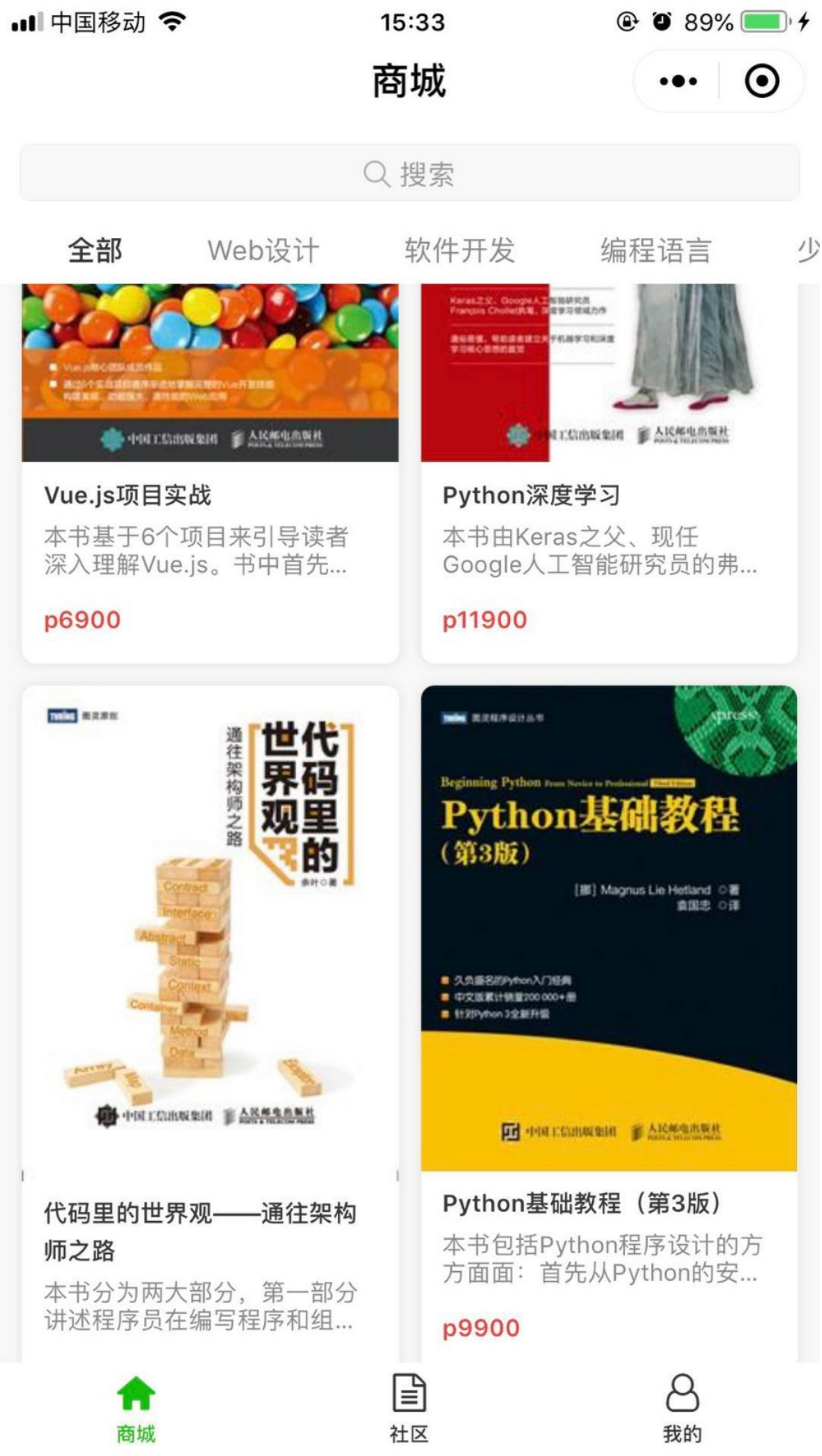
更新时间：2019-08-28 15:27:02

> 世界上最快乐的事，莫过于为理想而奋斗。
>
> ——苏格拉底

在前两节中我们已经完成了商城的业务设计、功能设计和数据库设计，本节我们将一起来实现商城首页。

在第二节中，我们已经设计了商城首页的页面效果，如图 9 所示。

图 9 商城首页页面



实现页面的第一步，是根据业务设计、功能设计和页面效果图，使用“分类拆解法”的拆解步骤，将页面进行详细拆解。

> Tips：
>
> 在进行产品需求设计、项目需求分析时，产品经理（PM）、业务分析师（BA）或售前经理（Pre-Sales）通常会使用原型设计工具，将业务设计和功能设计的内容变为更直观易懂的原型（通常是含交互效果的原型）。在需求确定后，再由 UI 设计师根据设计文档和原型制作高保真的页面效果图（本专栏的页面图都是高保真效果图）。
>
> 原型设计工具有很多种，比较流行的原型工具有墨刀、Axure 等，有兴趣的同学可以进行延伸学习。

1. 拆解步骤

在本节仅给出拆解结果，拆解过程请回顾第二章第三节对“分类拆解法”的详细讲解内容。

请各位同学自己动手实践，按照拆解步骤拆解，结合第一节的业务设计、第二节的功能设计，拆解图 9 的商城首页页面，然后将自己的拆解结果与本节列出的拆解结果进行对照总结。这是本节内容的实践环节之一。

商城首页页面可以由上到下拆解为 3 个子部件，具体拆解结果如下：

子部件 1 ：顶部搜索栏

子部件 1 的显示元素包括：

搜索框

静态界面，事件为用户点击屏幕事件，无数据

搜索框固定在页面顶部，不随屏幕滑动移动位置

用户点击事件的事件响应为：在用户点击搜索框后，页面跳转到商品搜索页面

子部件 2 ：分类菜单栏

子部件 2 的显示元素包括：

分类菜单

多条数据的交互界面，事件为用户下滑屏幕到页面底部与用户点击屏幕事件，数据为分类信息列表

分类菜单固定在页面顶部，不随屏幕滑动移动位置

用户屏幕滑动事件的事件响应为：根据用户向左或向右滑动屏幕的方向，显示分类信息列表中靠前或靠后的分类名称

用户点击屏幕事件的事件响应为：将用户点击选中的分类名称变为黑色，其余分类名称变为灰色，同时在子部件 3 中显示用户选中的分类的商品列表

分类信息列表在商品分类库中，该数据需要从 `membership-miniprogram/data/json_data.js` 中获取

在分类菜单中增加一个分类名称“全部”，第一次打开商城首页时默认选中“全部”分类，显示全部分类的商品列表，商品列表信息从数据库获取

子部件 3 ：商品列表栏

子部件 3 的显示元素包括：

商品列表

多条数据的交互界面，事件为用户屏幕滑动事件与用户点击屏幕事件，数据为：属于子部件 2 选中分类的商品信息数组

第一次打开商城首页时默认选中 “全部” 分类，显示全部分类的商品列表

用户下滑屏幕到页面底部事件的事件响应为：从商品信息表中分页获取当前选中分类的下一页商品信息数据，并追加到商品信息数组末尾。如果商品信息表中当前选中分类的所有数据都获取完毕，用户下滑屏幕将不再获取分页数据

商品信息数组中的商品数据以左右两列卡片式瀑布流的方式显示，一个卡片显示一个商品信息，每个卡片显示商品的缩略图、商品名称（黑色）、商品简介（灰色）、商品原价（红色）

用户点击屏幕事件的事件响应为：在用户点击商品卡片后，页面跳转到商品详情页面，需要向商品详情页面传递商品 ID

商品信息表的数据需要从云数据库中获取

2. 数据服务

请先按照本章第二节数据库设计的内容，在云数据库中新建商品信息表 `product` ，并导入商品数据。

在商城首页中，需要获取商品信息表 `product` 的信息，因此我们首先要建立数据服务 `ProductService` 。

`ProductService` 类在 ProductService.js 中实现，在该类中提供了一个方法 `getProductList` 用于对云数据库的 `product` 表进行条件查询，并**分页获取**商品记录，该方法会调用入参异步回调函数 `successCallback(productArray)` ，将商品记录交给异步回调函数进行后续业务处理。

`getProductList` 的整个实现逻辑与第六章第四节的 `getPointChangeList` 基本一致，主要区别是查询条件。商品查询的条件有两类：

- 第一类是在商城首页根据分类查询
- 第二类是在商品搜索页面根据商品名称查询

因此， `getProductList` 的查询参数有两个，一个是分类名称 `category` ，一个是商品名称搜索关键词 `keyword` 。如果某个查询参数不为空字符串，则构造该查询参数的查询条件。

在云数据的数据查询中，字符串匹配使用正则表达式实现。在微信官方的"小程序开发文档"中有云数据库正则表达式查询的详细说明，文档位置为：“云开发” -> “小程序端 API 文档” -> “数据库” -> " API 列表" -> “db.regexp”。

`getProductList` 中构造查询条件的代码如下：

```
/**
 * 从数据库获取用于显示的产品列表信息
 * @param {string} category 需要获取的数据的分类名称，值为“全部”获取全部分类
 * @param {string} keyword 需要获取的数据的书名关键词
 * @param {bool} isReset 是否清空分页缓存
 * @param {function} successCallback(productArray) 处理数据查询结果的回调函数
 * productArray数据结构：
 * [{
 *    index,  //商品ID
 *    name,   //书名（即商品名称）
 *    img,    //商品缩略图地址
 *    height, //商品缩略图高度（px）
 *    width,  //商品缩略图宽度（px）
 *    desc,   //本书简介（即商品描述）
 *    price,  //价格
 *    type    //数组类型，商品为'product'
 * },]
 */
getProductList(category, keyword, isReset, successCallback) {
  //... 略，详见专栏源代码

  //构造查询条件
  var query = db.collection('product')
  //如果分类为“全部”，则获取所有商品数据
  if (category != "全部" && category != "") {
    //如果分类不是“全部”，构造分类查询条件
    query = query.where({
      //查询条件使用正则表达式进行字符串匹配，云数据库的正则表达式查询详见说明文档：
      //https://developers.weixin.qq.com/miniprogram/dev/wxcloud/reference-client-api/database/db.regexp.html
      secondcategory: db.RegExp({
        regexp: category,
        options: 'i',
      })
    })
  } else if (keyword != "") {
    //如果书名关键词不为空，构造书名关键词查询条件
    query = query.where({
      //查询条件使用正则表达式进行字符串匹配
      bookname: db.RegExp({
        regexp: '.*' + keyword + '.*',
        options: 'i',
      })
    })
  }

  //... 略，详见专栏源代码
}
```

`getProductList` 的完整实现请回顾第六章第四节 `getPointChangeList` 方法的代码，或参考专栏源代码 ProductService.js，具体代码位置见本节末尾图 10。

3. 编程步骤

本节讲解的商城首页实现方式是直接在页面中实现所有功能，在源代码中使用了自定义组件的方式实现子部件 2 与 子部件 3 。

分类菜单的自定义组件源代码位于 `\membership-miniprogram\component\ScrollCategory\` 目录中。

瀑布流商品列表的自定义组件源代码位于 `\membership-miniprogram\component\WaterFallView\` 目录中。

学有余力的同学在按照本节讲解的方式实现商城首页后，可以继续学习微信官方“小程序开发文档”中自定义组件的内容（文档位置：“指南”->“自定义组件”），结合专栏源代码理解自定义组件实现方式，然后使用自定义组件的方式重构商城首页代码。

3.1 定义页面子部件及其排列顺序

首先在 WXML 页面模板中定义 3 个子部件的容器：

```
<!-- 1：顶部搜索栏 和 子部件2：分类菜单栏 固定在页面顶部 -->
<view class="page_top">
  <!-- 子部件1：顶部搜索栏 使用 WeUI 的 SearchBar 组件样式-->
  <view class="weui-search-bar">
  </view>
  <!-- 子部件2：分类菜单栏 使用微信官方的scroll-view控件实现横向滚动 -->
  <scroll-view scroll-x="true" class="category_tab_container">
  </scroll-view>
</view>
<!-- 子部件3：商品列表栏 -->
<view class="page_body">
</view>
```

子部件 1 ：顶部搜索栏 和 子部件 2 ：分类菜单栏固定在页面顶部，可使用 position 与 z-index 样式属性实现：

```
/* 搜索栏和分类菜单栏顶部固定，高度160rpx */
.page_top {
  position: fixed;
  z-index: 9999;
  top: 0px;
  width: 100%;
  height: 160rpx;
}
/* 商品列表栏，距离顶部160rpx*/
.page_body {
  position: absolute;
  top: 160rpx;
  width: 100%;
}
```

3.2 实现子部件 1 ：顶部搜索栏

顶部搜索栏的样式可以借用 WeUI 的 SearchBar 组件实现。

首先去除 WeUI 的 SearchBar 组件示例代码中的数据与事件绑定代码，然后设置搜索框为不可输入状态 disabled="true" ，再添加 Navigator 实现点击页面跳转：

```
<!-- 子部件1：顶部搜索栏 使用 WeUI 的 SearchBar 组件样式-->
<view class="weui-search-bar">
  <view class="weui-search-bar__form">
    <!-- 点击搜索框，页面跳转到商品搜索页面-->
    <navigator url="../search/search">
      <view class="weui-search-bar__box">
        <input type="text" class="weui-search-bar__input" placeholder="搜索" disabled="true" />
      </view>
      <label class="weui-search-bar__label">
        <icon class="weui-icon-search" type="search" size="14"></icon>
        <view class="weui-search-bar__text">搜索</view>
      </label>
    </navigator>
  </view>
</view>
```

WeUI 的 SearchBar 组件是灰色背景白色搜索框，要实现图 9 中的白色背景灰色搜索框效果，还需要对样式进行调整，调整后的样式请参考专栏源代码 index.wxss 中注释为 “/* 修改weui搜索栏样式 */” 的部分。

3.3 实现子部件 2 ：分类菜单栏

在第五章第四节与第七章第五节中，我们都使用了小程序官方组件 scroll-view 来实现横向滚动，分类菜单栏同样使用该组件实现，实现代码也与前面章节类似。

首先设置分类数据，定义用户点击分类的事件响应函数：

```
/**
 * 商品分类数据
 */
var productData = require("../../../../../data/json_data.js")

Page({

  /**
   * 页面的初始数据
   */
  data: {
    categories: [],    //分类数组
    selectedId: "0",    //当前选中的分类id，页面加载时默认选中全部分类
  },

  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function() {
    //读取分类数据
    this.setData({
      categories: productData.categories
    })
  },

  /**
   * 响应分类导航栏组件的分类点击事件
   */
  oncategoryChangeEvent(event) {
    //获取用户点击的分类
    var id = event.currentTarget.dataset.item.id;
    //如果用户点击的分类与当前选中的分类不一致，则切换当前选中的分类
    if (this.data.selectedId != id) {
      this.setData({
        selectedId: id,
      });
      //切换分类，子部件 3 需要重新初始化显示新分类的商品列表
      this.refreshProductList(event.currentTarget.dataset.item.title)
    }
  },
})
```

页面加载时默认选中全部分类，分类数据从 `membership-miniprogram/data/json_data.js` 文件中读取。

需要注意的是，在切换分类后，子部件 3 中要显示新选中分类的商品数据，因此需要先定义一个刷新商品列表显示的函数 `refreshProductList` ，函数的具体逻辑在实现子部件 3 时实现。

然后在 WXML 页面模板中实现横向分类滚动条：

```
<!-- 子部件2：分类菜单栏 使用微信官方的 scroll-view 控件实现横向滚动 -->
<scroll-view scroll-x="true" class="category_tab_container">
  <view class="category_tab_container_padd"></view>
  <block wx:for="{{categories}}" wx:key="{{item.id}}">
    <view class="category_tab_item {{item.id == selectedId ? 'category_tab_item_selected' : ''}}" bindtap="oncategoryChangeEvent" data-item="{{item}
}">
      {{item.title}}
    </view>
  </block>
  <view class="category_tab_container_padd"></view>
</scroll-view>
```

实现图 9 效果的自定义样式见专栏源代码 `\membership-miniprogram\component\ScrollCategory\ScrollCategory.wxss` 。

3.4 实现子部件 3 ：商品列表栏

商品列表的实现主要包含两部分，一是根据用户选择的分类分页获取商品数据，二是将获取到的数据以瀑布流的方式展示在页面中。

3.4.1 获取商品数据

在第六章第四节，我们已经实现了积分记录的分页读取，商品列表的分页读取实现与之类似。

首先引用 DataService：

```
//引用商品信息的数据库访问类
import ProductService from '../../../../../dataservice/ProductService.js'
var productService = new ProductService()
```

定义数据：

```
data: {
  selectedCategory: "全部", //当前选中的分类名称，页面加载时默认选中全部分类，用于数据库查询
  isNoMoreData: false, //记录是否已加载完所有分页数据
},
```

实现分页加载数据的逻辑：

```
/**
 * 调用 ProductService 的 getProductList 方法获取用户选中分类的商品列表
 */
getProductList(isCategoryChanged) {
  var that = this
  //从数据库获取用户选中分类的商品列表信息
  productService.getProductList(
    this.data.selectedCategory, //用户选中分类的分类名称
    "",
    isCategoryChanged,
    function (productArray) {
      //填充数据到瀑布流中显示
      that.fillData(isCategoryChanged, productArray)
      if (productArray.length <= 0) {
        that.setData({
          isNoMoreData: true //设置数据全部加载完毕的标志
        })
      }
    })
},
```

`fillData` 函数负责填充新获取到的分页商品数据到瀑布流中显示，在本节 3.4.2 中具体讲解。

实现第一次打开商城首页时默认选中 “全部” 分类，显示全部分类的商品列表：

```
/**
 * 生命周期函数--监听页面初次渲染完成
 */
onReady: function() {
  //读取分类数据
  this.setData({
    categories: productData.categories
  })
  //页面加载时获取 “全部” 分类的第一页数据
  this.getProductList(true)
},
```

实现用户下滑屏幕到页面底部事件的事件响应：

```
/**
* 下滑屏幕到页面底部事件的处理函数
*/
onReachBottom: function () {
  if (!this.data.isNoMoreData) { //如果还有数据未加载完，则获取更多数据
    this.getProductList(false)
  }
},
```

此外，我们还要实现用户点击不同分类后刷新商品列表显示的函数 `refreshProductList` ：

```
/**
 * 切换分类后重新加载商品列表
 */
refreshProductList(title){
  //重新初始化显示新分类的商品列表
  this.setData({
    selectedCategory: title, //设置用户当前选中的分类名称
    isNoMoreData: false, //重置分页数据全部加载完毕的标志
  });
  //页面回到顶部
  wx.pageScrollTo({
    scrollTop: 0,
    duration: 0
  })
  //显示新分类商品列表的第一页数据
  this.getProductList(true)
},
```

3.4.2 瀑布流显示

几乎所有 APP 、小程序的商城模块都使用瀑布流的方式来展示商品列表。

在网上能搜索到很多瀑布流的实现方式，但大都是针对 HTML5 而不是针对微信小程序的。

在简书中我发现了一个比较好的微信小程序瀑布流实现方式：微信小程序瀑布流最好最简单的解决方案，使用这一方案可以很简单的实现小程序瀑布流。

请各位同学详细阅读这篇文章。

在详细阅读该方案后，我们可以移植该方案代码来实现商品列表的瀑布流显示：

首先参考该方案的 “5. 所有JS代码” 在 JS 逻辑中完成瀑布流的核心逻辑“如何把要展示的数据 item 放入 leftList、rightList 这两个数组中”：

```
/**
 * 瀑布流全局变量
 */
var leftList = new Array(); //瀑布流左侧集合
var rightList = new Array(); //瀑布流右侧集合
var leftHight = 0, //瀑布流左边一列的高度
  rightHight = 0, //瀑布流右边一列的高度
  itemWidth = 0, //根据手机的屏幕分辨率计算出每张图片的固定宽度
  maxHeight = 0; //每张图片的最大高度

Page({

  /**
   * 页面的初始数据
   */
  data: {
//... 前面内容中已定义的数据 略
    leftList: [], //瀑布流左侧集合
    rightList: [], //瀑布流右侧集合
    itemWidth: 0 //根据手机的屏幕分辨率计算出每张图片的固定宽度
```

```
  },

  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function() {
    //根据手机的屏幕分辨率设置图片宽度和每张图片的最大高度
    wx.getSystemInfo({
      success: (res) => {
        var percentage = 750 / res.windowWidth;
        var margin = 60 / percentage;
        itemWidth = (res.windowWidth - margin) / 2;
        maxHeight = itemWidth / 0.5
      }
    });
  //... 前面内容中已定义的逻辑 略
  },

  //... 前面内容中已定义的函数与事件 略

  /**
   * 填充数据到瀑布流中显示
   * @param {bool} isPull 是否清空数据
   * @param {array} listData 在瀑布流末尾追加显示的多条数据
   * [{
   *     index,  //记录id
   *     width,  //图片宽度
   *     height, //图片高度
   *     img,    //图片URL
   *     ...     //其他需要在模板中显示的内容
   *},]
   */
  fillData: function (isPull, listData) {
    this.setData({
      itemWidth: itemWidth
    })
    if (isPull) {
      //清空已加载的数据
      leftList.length = 0;
      rightList.length = 0;
      leftHight = 0;
      rightHight = 0;
    }

    for (var i = 0, len = listData.length; i < len; i++) {
      var tmp = listData[i];
      //计算每张图片的高度和宽度
      tmp.width = parseInt(tmp.width);
      tmp.height = parseInt(tmp.height);
      tmp.itemWidth = itemWidth
      var per = tmp.width / tmp.itemWidth;
      tmp.itemHeight = tmp.height / per;
      if (tmp.itemHeight > maxHeight) {
        tmp.itemHeight = maxHeight;
      }
      //计算并将每条数据放入瀑布流的左列或右列显示
      if (leftHight == rightHight) {
        leftList.push(tmp);
        leftHight = leftHight + tmp.itemHeight;
      } else if (leftHight < rightHight) {
        leftList.push(tmp);
        leftHight = leftHight + tmp.itemHeight;
      } else {
        rightList.push(tmp);
        rightHight = rightHight + tmp.itemHeight;
      }
    }
    //设置更新后的瀑布流数据，刷新瀑布流显示
    this.setData({
      leftList: leftList,
      rightList: rightList,
    });
  },
```

```
})
```

然后在 WXML 页面模板中显示瀑布流左侧数据 leftList 与 右侧数据 rightList ：

```
<!-- 子部件3：商品列表栏 -->
<view class="page_body">
 <!-- 瀑布流显示内容 -->
 <view class='fall-container'>
  <!-- 左边一列 -->
  <view class='fall-left cards'>
   <block wx:for="{{leftList}}" wx:key="{{item.index}}">
    <!--瀑布流内容卡片-->
    <template is='productsCard' data="{{data:item}}" />
   </block>
  </view>
  <!--右边一列 -->
  <view class='fall-right cards'>
   <block wx:for="{{rightList}}" wx:key="{{item.index}}">
    <!--瀑布流内容卡片-->
    <template is='productsCard' data="{{data:item}}" />
   </block>
  </view>
 </view>
</view>
```

由于左右两列的商品卡片显示样式与内容都相同，可以使用 WXML 的模板语法（template）来简化代码。WXML 的模板语法介绍请阅读官方文档：“框架” -> “WXML 语法参考” -> “模板”。

在商品列表的内容卡片模板中，显示商品的缩略图、商品名称（黑色）、商品简介（灰色）、商品原价（红色），并添加 Navigator 实现点击卡片跳转到商品详情页：

```
<!--商品列表的内容卡片模板-->
<template name='productsCard'>
 <view class="card">
  <navigator url="../../shop/product/product?index={{data.index}}">
   <image class='card-img' mode='aspectFill' style='width:{{data.itemWidth}}px;height:{{data.itemHeight}}px;' src='{{data.img}}' lazy-load>
   </image>
   <view class='text'>
    <view class='title'>{{data.name}}</view>
    <view class='desc'>
     <rich-text nodes="{{data.desc}}"></rich-text>
    </view>
    <view class='price'>p{{data.price}}</view>
   </view>
  </navigator>
 </view>
</template>
```

实现图 9 中卡片显示效果的自定义样式见专栏源代码 \membership-miniprogram\component\WaterFallView\WaterFallView.wxss 中注释为 “/* 卡片样式 */” 的部分。

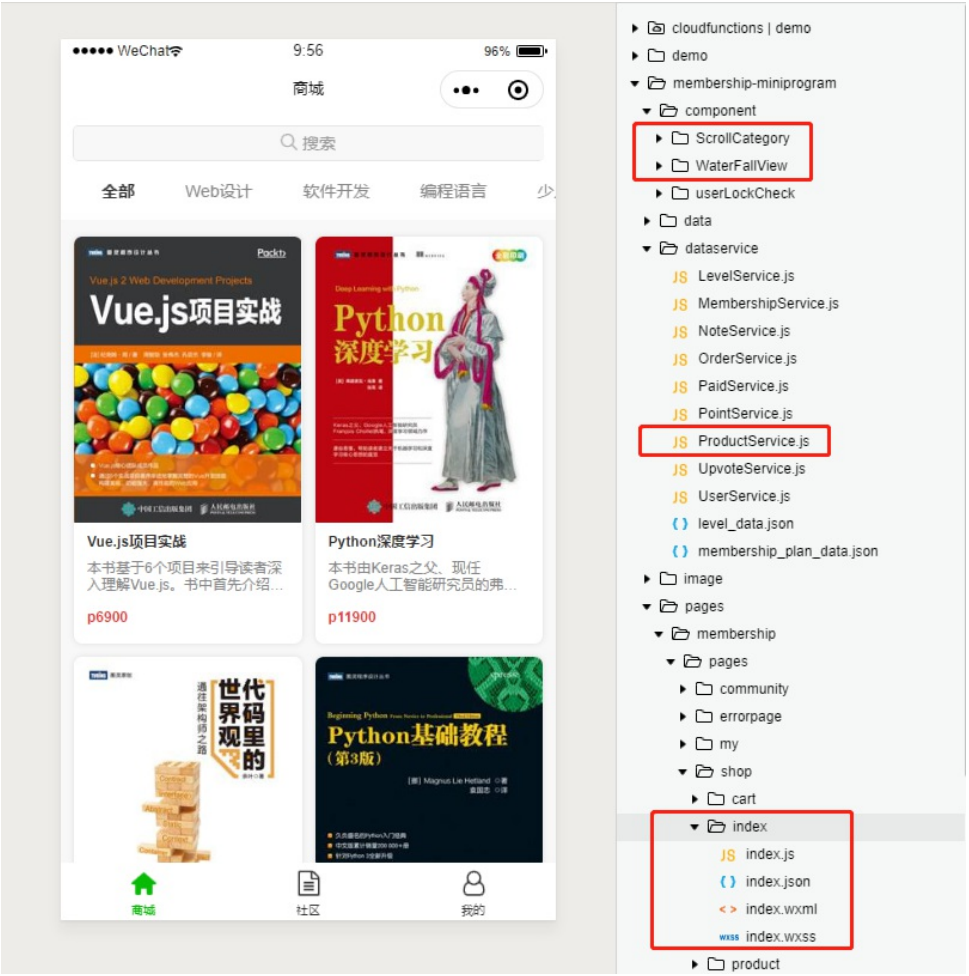
4. 专栏源代码

本专栏源代码已上传到 GitHub ， 请访问以下地址获取：

https://github.com/liujiec/Membership-ECommerce-Miniprogram

本节源代码内容在图 10 红框标出的位置。

图 10 本节源代码位置



下节预告

下一节，我们将开发商品搜索页面，完成搜索热词、搜索历史和联想搜索等功能。

实践环节

实践是通往大神之路的唯一捷径。

本节实操内容：

- 请结合第二章第三节对“分类拆解法”的详细讲解内容，拆解图 9 的商城首页，然后将自己的拆解结果与本节列出的拆解结果进行对照总结。
- 编写代码完成图 9 所示的页面，如碰到问题，请阅读本专栏源代码学习如何实现。
- 学有余力的同学在按照本节讲解的方式实现商城首页后，可以继续学习微信官方“小程序开发文档”中自定义组件的内容（文档位置：“指南” -> “自定义组件”），结合专栏源代码理解自定义组件实现方式，然后使用自定义组件的方式重构商城首页代码。

}

29 商城功能与数据库设计

31 搜索热词、搜索历史和联想搜索