电商类小程序实战教程 Vol.2：列表加载

在上一期文章中，我们以爱范儿旗下的玩物志电商小程序作为 Demo，介绍了首页 banner 部分的开发，以及微信小程序内置组件、数据绑定和发送请求 API 的用法。

本期，知晓程序依然以玩物志电商小程序为基础，为大家带来有关于列表加载的实现教程。为布局而生的 flex 传统的布局方式一般都是基于 div 盒子模型，利用 float，position，display 来进行布局。每个前端开发者对这些布局方式都非常熟悉。对一些特殊的布局来说，使用这些属性并不方便，由此还衍生出各种 hack 方案。直到 CSS3 提出了专为布局而生的解决方案，就是 flex 布局。但是，由于浏览器的兼容问题，大多数开发者都不敢将 flex 布局直接应用到实际项目之中。在常规项目中，大量应用 flex 布局还需要一个过程。而对于 WeChat Only 的小程序来说，则**完全不用担心有浏览器兼容的问题，它只要适配微信客户端即可**。同时，由于小程序的布局不会非常复杂，所以，也不用过多地担心大量使用 flex 引起的性能问题。**我们可以放心地在小程序中使用 flex 布局。**如何更方便地使用 rpxrpx 的全称是 responsive pixel，它是小程序自己定义的一个尺寸单位，可以根据当前设备的屏幕宽度进行自适应。小程序中规定，所有的设备屏幕宽度都为 750rpx，**根据设备屏幕实际宽度的不同，1rpx 所代表的实际像素值也不一样**。例如，在 iPhone 6 上，屏幕实际宽度为 375px，则 750rpx = 375px，1rpx = 0.5px；而在 iPhone 5 上，屏幕实际宽度为 320px，则 750rpx = 320px，1rpx = 0.42px。其实，我们并不必关心每种设备屏幕下 1rpx 到底代表多少个像素，**只要抓住「所有的设备屏幕宽度都为 750rpx」这个原则**，就能很好地实现对任意设备屏幕大小的自适应布局。知晓程序（微信号 zxcx0101）**强烈建议设计人员用 iPhone 6 作为视觉稿的标准，即将视觉稿总宽度设成 750px**。这样，开发者就能很方便地对相关的尺寸进行量取。比如，在总宽度为 750px 的 iPhone 6 视觉稿中，量取一个图片的宽度为 200px，那么，这个图片的宽度即可设置为 200rpx。**简单一句话解释：量取多少就设置多少。**flex 配合 rpx 的使用案例现在，我们就在小程序中使用 flex 和 rpx 进行布局，体验一下这种解决方案的便利。**案例一：货架列表**来看看货架列表的效果图：首先还是贴两段代码。以下是 WXML 的实现：

```
/*** index.wxml ***/
<view class="shelf-nav">
 <view class="shelf-nav-item" wx:for="{{ shelfNavList }}">
 <navigator url="../list/list?id={{ item.id }}">
 <image src="{{ item.cover_image }}"
  class="shelf-nav-item__image">
 </image>
 <text>{{ item.name }}</text>
 </navigator>
 </view>
</view>
```

然后 WXSS 的代码如下，我们在关键代码处已经给出简要注释：

```
/*** index.wxss ***/
.shelf-nav {
 display: flex; //设置 display: flex; 将它变为 flex 布局的元素
 flex-wrap: wrap; //当子元素总宽度超过父元素宽度时换行显示
 padding: 30rpx;
}

.shelf-nav-item {
 width: 25%; //因为每行显示 4 个货架，所以宽度设置为 25%;
 margin-bottom: 20rpx;
 text-align: center; //让它中间的图片和标题居中显示
}

.shelf-nav-item__image {
 width: 130rpx; //在视觉稿中量取图片宽高为 130px
 height: 130rpx; //于是设置 width 和 height 都设置为130rpx;
 border-radius: 50%; //把图片设置成圆形
 border: 1px solid #d9d9d9; //加上外边框
}
```

简单几步就完成了货架列表的布局，**并且完美兼容各种大小的设备屏幕**。**案例二：货架分类标题**再来一个例子，看一下商品列表货架分类的标题：

如上图所示，货架标题居左，「查看更多」的图标居右并且垂直居中。代码结构如下：

```
/*** index.wxml ***/
<view class="shelf-header">
 <view class="shelf-title">
 <text class="shelf-title-en">{{ shelf.english_name }}</text>
 <text class="shelf-title-cn">{{ shelf.name }}</text>
 </view>
 <view class="shelf-more">
 <navigator url="../list/list?id={{ shelf.id }}"
  class="ifanrin-more-icon ifanrin">
 </navigator>
 </view>
</view>
```

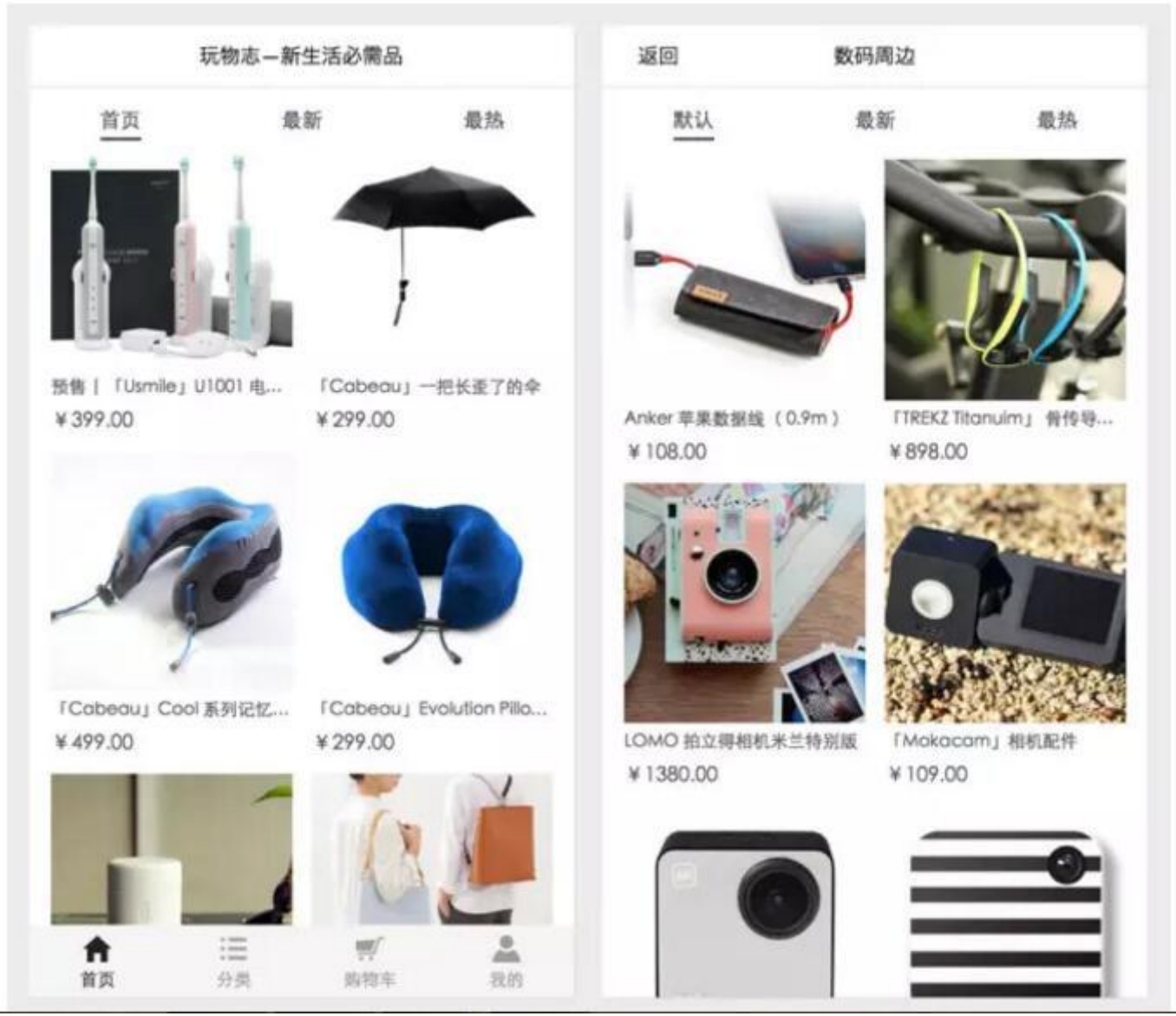
有经验的同学一眼就可以看出，要实现需求需要：
让 .shelf-title 向左浮动
让 .shelf-more 水平方向右对齐、垂直方向居中
最后还要给 .shelf-header clearfix 一下
那现在来看看 flex 是怎么做的：

```
/*** index.wxss ***/
.shelf-header {
 display: flex; //设置为 flex 布局的元素
 justify-content: space-between; //均匀排列每个元素，
                                 //首、末两元素分别位于起点和终点
}

.shelf-more {
 align-self: center; //设置居中
}
```

只要区区几行代码，就能轻松使用 `flex` **调整排版。**接下来，结合上一期的内容，发起一个请求获取数据，然后 `setData()` 绑定数据：

```
/*** index.js ***/
Page(){
 ...,
 onLoad: function () {
  var that = this;
  wx.request({
   url: CONFIG.API_URL.SHELF_QUERY,
   method: 'GET',
   data: {
    img_size: 'small'
   },

   success: function (res) {
    that.setData({
     shelfNavList: res.data.objects
    });
   }
  });
 },
 ...
}
```

货架列表就完成了。template 模板的使用我们再来看看，首页的分类列表和货架分类列表页两个页面的对比：



不难发现，这两个页面的列表的样式是一模一样的。那么，我们就可以把每个商品视为一个模块，暂且把这个商品模块称为 `product-card`，我们可以将它用 `template` 封装起来，在需要的地方就将它引入并进行调用。

这里先要弄清楚一个概念，template 的主要功能更多的是在于定义一个 wxml 的代码片段，然后在不同的地方调用。

template 拥有自己的作用域，只能使用 data 传入的数据。我们来看下 template 到底如何使用。1. 定义 template 模板为了方便代码组织，我们在 templates 目录下，新建一个 productCard 文件夹，并在 product-card 文件夹下新建 productCard.wxml 和 productCard.wxss 文件。代码如下：

```
/*** productCard.wxml ***/
<template name="productCard">
 <view class="product-card" >
 <navigator url="../detail/detail?id={{ id }}">
 <view style="background-image: url({{ cover_image }})"
  class="product-cover">
 </view>
 <view>
 <text class="product-title">{{ title }}</text>
 <text class="product-price">￥{{ price }}</text>
 </view>
 </navigator>
 </view>
</template>
```

使用 name 属性定义模板的名字，然后将代码片断保存在 template 中。2. 引入 template 模板以首页为例，当要使用到 productCard 模板时，我们只需要使用 import 引入模板。在需要显示的

位置，外层用 wx:for 循环渲染列表，template 为子项，使用 is 声明需要的使用的模板，用 data 传入数据：

```
/*** index.wxml ***/
<import src="../../templates/productCard/productCard.wxml"/>
<view class="product-list">
 <block wx:for="{{ productNewList }}"
 <template is="productCard" data="{{ ...item }}"/>
 </block>
</view>
```

留意一下 data="{{ ...item }} 的写法，item 是 wx:for 中代表数组当前项的默认变量名，item 前面的 ... 操作符相当于 ES6 中的展开运算符，可用于需要解构赋值的地方，没有了解过展开运算符和解构（Destructuring）的同学可以先了解一下它们的基本概念。通过解构，template 中就可以直接写成 {{ id }}, {{ cover_image }}，而不用写 {{ item.id }}, {{ item.cover_image }}。它的意义在于实现了 template 与 wx:for-item 之间的解耦，比如这里设置了 wx:for-item="product"，我们只要设改变 data="{{ ...product }} 就可以了。如果数据没有通过解构，就要将 template 的 {{ item.id }} 修改成 {{ product.id }}，很不方便。接下来是 productCard.wxss 的引入。先在 productCard 写好样式，这里就不贴代码了。模板的 WXSS 文件如何引入首先请思考一下，在哪里可以引入 WXSS 文件？一种方法是在用到 productCard 模板的页面里引入，在这里是在 list.wxss 中 import 进来。另一种方法是，直接在 app.wxss 中引入。相比较于上一种方法，这个方法**只需要一次引入，而所有用到 productCard 模板的页面都不用再去引入 productCard.wxss 了。**

```
/*** app.wxss ***/
@import "./templates/productCard/productCard.wxss";
```

获取商品列表数据，渲染视图先通过 onLoad 的 options 取得货架 id：


```
/*** app.wxss ***/
@import "./templates/productCard/productCard.wxss";
```


最后，调用 wx:request() 获取商品列表数据，通过 setData() 设置，即可在视图层渲染出完整的列表。

```
/*** list.js ***/
onLoad: function (options) {
 ...
 wx.request({
  url: CONFIG.API_URL.PRODUCT_LIST,
  method: 'GET',
  data: {
   shelf__id: this.data.shelfId
  },

  success: function (res) {
   if (res.status == 200) {
    that.setData({
     productNewList: res.data.objects
    });
   }
  }
 });
 ...
}
```

到这里，一个像样的列表页面就做好了。但如上面的代码所示，现在小程序会一次性将所有商品列表查询并渲染至页面中。现实中，这种做法显然是不科学的。我们还需要一个「加载更多」的功能。这个功能要求我们做到：用户访问时，页面加载 20 个商品，点击列表底部的「查看更多」按钮可再多加载 10 个商品。

具体如何实现呢？请继续关注下一期的内容。