35 给用户看账单：开发实现订单列表页面

更新时间：2019-09-09 14:25:16

> 立志是事业的大门，工作是登门入室的的旅途。
>
> ——巴斯德

通过前面七节内容，我们已经开发完成了商城的大部分功能，本节是商城开发的最后一小节，将实现订单列表页面。

在第二节中，我们已经设计了订单列表页面效果，如图 18 所示。

图 18 订单列表页面

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，拆解本节图 18 的订单列表页面，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。

订单列表页面主要包括 1 个子部件，具体拆解结果如下：

子部件 1 ：订单列表栏

子部件 1 的显示元素包括：

订单列表

多条内容交互界面，事件为用户点击事件与用户下滑屏幕到页面底部事件，数据为订单记录数组

订单记录数组的第一个分页数据在页面加载时获取，订单的显示按照购买时间倒序排列

每个订单记录需要在第一行左对齐显示购买时间，中间显示订单中每个商品的信息，最后一行右对齐显示订单中的商品数量与订单实际支付价格；每个商品的信息用一行显示，左侧显示商品缩略图，右侧分两行显示商品名称和商品描述

订单记录数组包含订单记录与订单中每个商品的商品信息，从订单记录表中获取订单记录列表后，还需要根据商品 ID 从商品信息表中查找每个订单中每个商品的商品信息

用户下滑屏幕到页面底部事件的事件响应为：从订单记录表中分页获取下一页数据，并追加到订单记录数组末尾。如果订单记录表中所有数据都获取完毕，用户下滑屏幕将不再获取分页数据

用户点击事件的事件响应函数为：用户点击订单列表中某一商品信息，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

在页面加载时，从订单记录表中读取用户已支付成功的订单记录，并从数据库中读取每个订单中包含的每个商品的详细信息。

2. 数据服务

订单列表页面需要分页显示订单记录表的数据，因此需要新建一个 Data Service `OrderService` 供页面调用。`OrderService` 在 OrderService.js 中实现，提供一个方法 `getOrderList(isReset, successCallback)` 供页面分页获取订单记录表中的数据。

** 订单列表页面的功能类似第六章第四节的积分体系页面，主要功能是分页列表显示数据库记录。** 在第六章第四节已经详细讲解了分页获取积分记录表数据方法 `getPointChangeList` 的具体实现代码，`getOrderList` 的实现代码与之基本一致。

`getOrderList` 的实现代码请参考第六章第四节，也可参考专栏源代码的 `OrderService.js` 文件，具体代码位置见本节末尾图 19。

从订单记录表中获取的数据并不能直接显示到页面中，因为订单记录表中只包含商品 ID 数组，不包含具体的商品信息。

在获取到订单记录表的数据后，还需要根据商品 ID 从商品信息表中查找每个订单中每个商品的商品信息。如果每次查找商品 ID 对应的商品信息都调用本章第六节中实现的 `getProductByIndex` 方法，显示一页订单列表数据可能要访问几十次商品信息表，这样的性能开销非常大的。

为解决性能开销问题，我们需要构建一个减少访问数据库次数的算法。

在这里给出专栏源代码中使用的算法思路：

- 第一步，调用数据服务获取一页订单列表
- 第二步，将订单列表中每个订单的所有商品 ID 都存放到一个全量商品 ID 数组中，存放时去重
- 第三步，在商品信息的数据服务中增加一个方法 `getProductsByIndex`，输入参数为全量商品 ID 数组，返回结果为全量商品 ID 数组对应的全量商品信息数组，该方法仅请求一次数据库
- 第四步，订单列表中每个订单增加商品信息数组字段，存放每个商品 ID 对应的商品信息，商品 ID 对应的商品信息可以从第三步得到的全量商品信息数组中查询获取

该算法需要在 `ProductService` 类中增加一个方法 `getProductsByIndex` 。 `getProductsByIndex` 的具体实现代码请阅读专栏源代码的 `ProductService.js` 文件。

解决性能开销的算法远不止一种，有兴趣的同学可以思考还有哪些方法可以降低性能开销。

3. 编程步骤

订单列表页面只有一个显示元素，就是订单列表。

订单列表的数据为分页获取的订单记录数组，**分页获取列表数据的具体实现方式在第六章第四节和第三章第二节的“4. 列表数据显示”中均有有详细讲解，具体算法思路请回顾该节内容**，本节仅给出代码实现结果。

订单列表页面的数据包括：

```
data: {
  orders: [], //订单记录数组
  isNoMoreData: false, //记录是否已加载完所有分页数据
},
```

按照前面介绍的算法思路，订单记录数组的分页加载数据的逻辑实现如下：

```
/**
 * 从数据库获取订单列表
 */
getOrderList(isReset) {
  var orders = this.data.orders
  var that = this
  //第一步，调用数据服务获取一页订单列表：调用 OrderService 的 getOrderList 方法
  orderService.getOrderList(
    isReset, //是否重新从第一页开始返回分页数据
    //处理数据库查询结果的回调函数
    function(orderArray) {
      if (orderArray.length > 0) {
        //第二步，将订单列表中每个订单的所有商品 ID 都存放到一个全量商品 ID 数组中，存放时去重
        var productsIndex = [] //全量商品 ID 数组
        //循环将每个订单的每个商品 ID存放到全量商品 ID 数组中
        for (var i in orderArray) {
          for (var j in orderArray[i].productsIndex) {
            //进行去重判断
            if (productsIndex.indexOf(orderArray[i].productsIndex[j]) < 0) {
              productsIndex.push(orderArray[i].productsIndex[j])
            }
          }
        }
        //第三步，调用 ProductService 的 getProductsByIndex 方法，获得商品 ID 数组对应的商品信息数组，该方法仅请求一次数据库
        productService.getProductsByIndex(
          productsIndex, //全量商品 ID 数组
          //处理数据库查询结果的回调函数
          function(productArray) {
            if (productArray.length > 0) {
              //第四步，订单列表中每个订单增加商品信息数组字段，存放每个商品 ID 对应的商品信息
              for (var i in orderArray) {
                //在每个订单中增加商品信息数组字段
                orderArray[i].products = []
                for (var j in orderArray[i].productsIndex) {
                  //在全量商品信息数组中查询每一个商品 ID 对应的商品信息，并存放到商品信息数组字段
                  var filter = productArray.filter(e => e.index == orderArray[i].productsIndex[j])
                  if (filter.length > 0){
                    orderArray[i].products.push(filter[0])
                  }
                }
              }
            }
            //使用数组的 concat 方法将新获取到的数据添加到数据集合末尾
            orders = orders.concat(orderArray)
            //更新数据集合，使页面显示新获取到的数据
            that.setData({
              orders: orders
            })
          })
      } else {
        //如果查询没有获取到数据，说明云开发数据库中所有数据都已经获取完毕
        //设置数据全部加载完毕的标志
        that.setData({
          isNoMoreData: true
        })
      }
    })
},
```

请参照前面章节的内容自行添加数据服务引用。

在页面加载时显示第一页订单列表，用户滑动屏幕到页面底部时加载后一页的订单列表：

```
  /**
   * 生命周期函数--监听页面初次渲染完成
   */
  onReady: function() {
    this.getOrderList(true) //页面加载完毕后获取订单列表的第一页分页数据
  },

  /**
   * 用户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.getOrderList(false)
    }
  },
```

订单记录数组 `orders` 的显示使用 WXML 的列表渲染语法 `wx:for` ，每一条订单记录的显示样式可直接套用 WeUI 的 Panel 组件中的“图文组合列表”：

```
<!-- 子部件 1 ：订单列表栏 -->
<view class="order_list">
  <block wx:for="{{orders}}" wx:key="{{item._id}}">
    <!-- 直接套用 WeUI Panel 组件中的“图文组合列表”样式显示一条订单记录 -->
    <view class="weui-panel weui-panel_access">
      <!-- 第一行左对齐显示购买时间 -->
      <view class="weui-panel__hd">{{item.date}}</view>
      <!-- 中间显示订单中每个商品的信息 -->
      <view class="weui-panel__bd">
        <block wx:for="{{item.products}}" wx:for-item="product" wx:key="{{product.index}}">
          <!-- 使用 WeUI Panel 组件中的“图文组合列表”样式显示每个商品信息 -->
          <!-- 点击商品信息后页面跳转商品详情页面 -->
          <navigator url="../../shop/product/product?index={{product.index}}" class="weui-media-box weui-media-box_appmsg" hover-class="weui-cell_acti
ve">
            <view class="weui-media-box__hd weui-media-box__hd_in-appmsg">
              <image class="weui-media-box__thumb" src="{{product.smallcoverimg}}" />
            </view>
            <view class="weui-media-box__bd weui-media-box__bd_in-appmsg">
              <view class="weui-media-box__title">{{product.bookname}}</view>
              <view class="weui-media-box__desc">{{product.desc}}</view>
            </view>
          </navigator>
        </block>
      </view>
      <!-- 最后一行右对齐显示订单中的商品数量与订单实际支付价格 -->
      <view class="weui-panel__ft">
        <view class="weui-cell weui-cell_access weui-cell_link">
          <view class="weui-cell__bd text-right">共{{item.products.length }}件商品 实际支付：p{{item.paymentFee}}</view>
        </view>
      </view>
    </view>
  </block>
</view>
```

由于篇幅所限，包含完整样式的 **WXML** 页面模板代码请查阅专栏源代码 **order.wxml** 与 **order.wxss** ，完整的 **JS** 逻辑代码见 **order.js** ，具体代码位置见本节末尾图 **19** 。

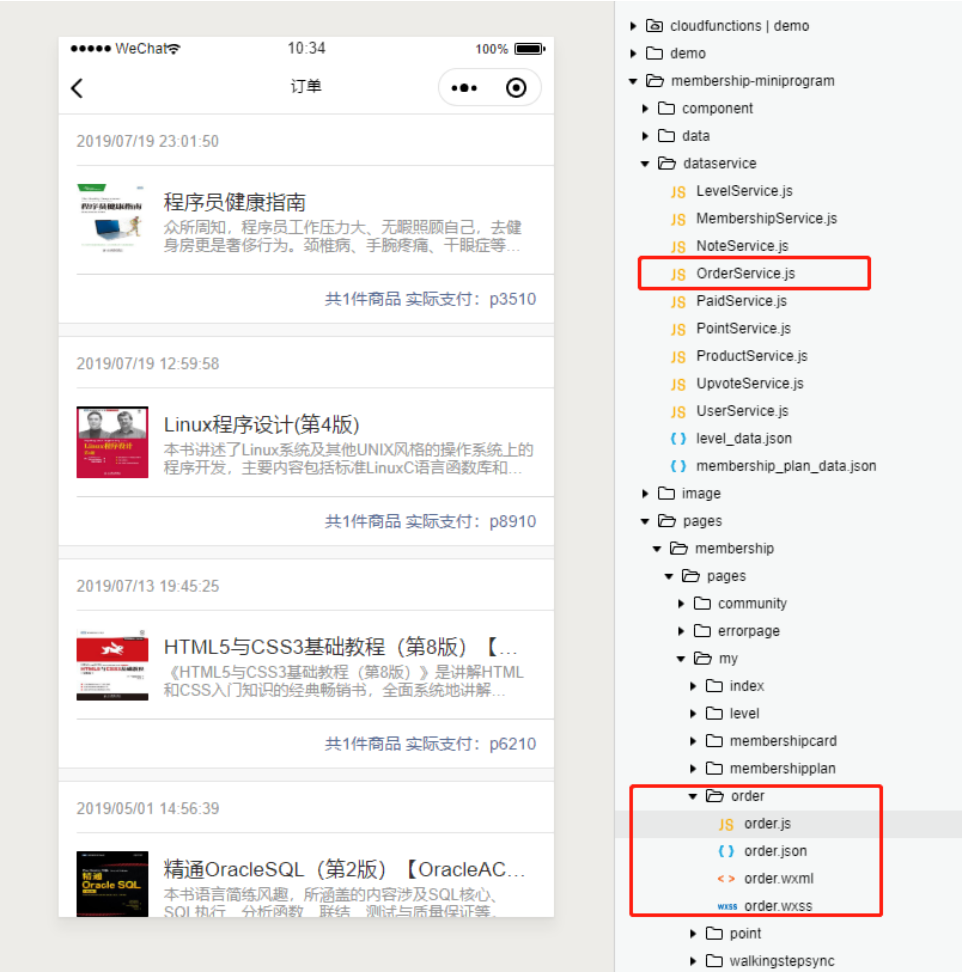
4. 专栏源代码

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 19 红框标出的位置。

图 19 本节源代码位置



下节预告

从下一节开始，我们将设计并实现 UGC 社区模块。

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，拆解本节图 18 的订单列表页面，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。
- 编写代码完成订单列表页面，如碰到问题，请阅读本专栏源代码学习如何实现。

}

34 让用户买买买：开发实现商城购物车页面

36 UGC 社区业务设计