阶段一：小程序高级电商前端

直接对标真实商业级小程序的精致电商小程序，覆盖商品专题、商品Spu、商品Sku、优惠券、二级商品分类、折扣系统、支付系统等主流电商功能。

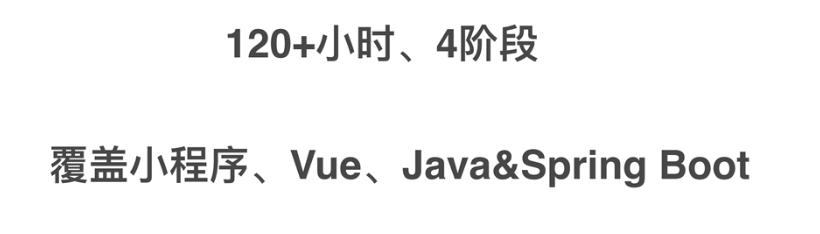
第一周：走进 web 全栈工程师

开课第一周，我们会公布完整的课程体系计划，展示完整的项目效果，然后开始进行项目分析，项目技术选型，和小程序组件化开发方式和技巧。

## 序章：三十周的试炼，从青铜到王者

### Java 全栈序章

这里进行一些介绍。



小程序和 CMS 演示，可以通过下面的截图以及地址进行访问，看有哪些功能。

这里的 CMS 不是一个玩具系统，有着非常全的功能，有权限、日志各种必要全的功能。

并且所有的权限都是具体到按钮级别，并且是配置级别的。

并且 CMS 开发是比 C 端的功能还是复杂的，交互难度是比较复杂的

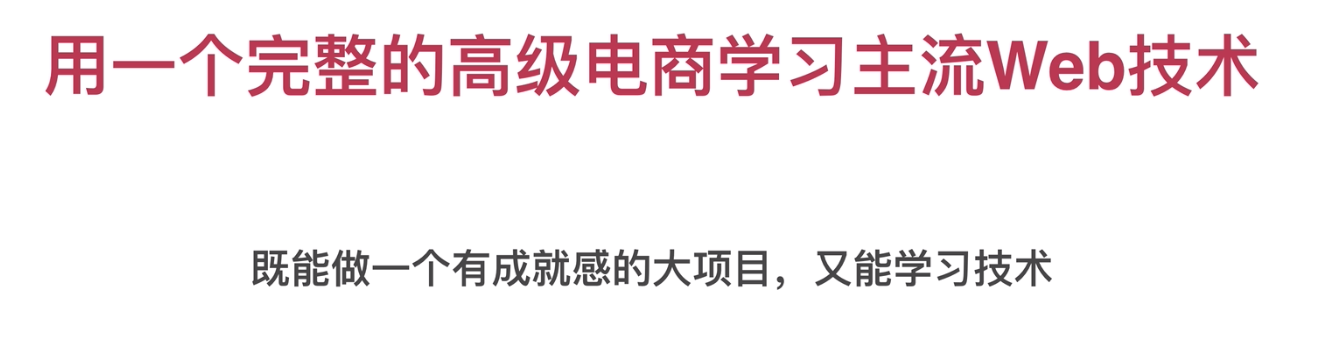


特点如下：

1、能够在学习的时候有着一个成就感，如果当成一个职业来的话，兴趣很可能会减弱。

2、技术的学习必须要有复杂的业务的支撑。

3、项目要有足够的复杂，这样才会想尽各种办法来解决，这样才能提高技术



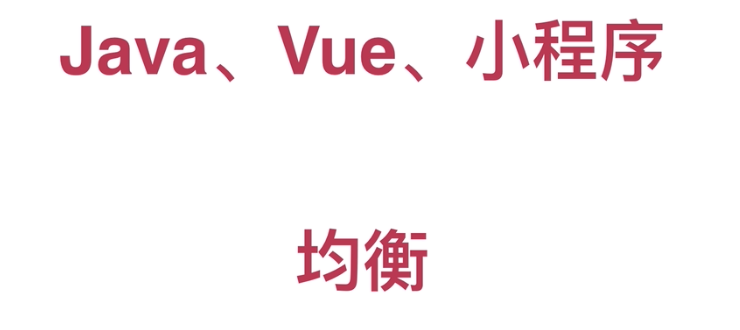
四个阶段：

如果项目比较大的话，比如在公司开发，C 端 API 和 CMSAPI 是分开的。

如果是小项目的话，以 CMSAPI 项目为基础来进行开发 C 端 API。



在 Java、Vue、小程序中间进行均衡，做一个很好的产品



具体看一下技术细节：

1、小程序

2、Java SpringBoot

3、Vue3.0

4、Java CMS

用到的两个开源技术组件：

Lin UI & Lin CMS

地址：https://github.com/TaleLin

### 长期课程规划与前置要求

1、非快餐课程、长期维护

后续可能会增加第二期，会基于第一期的内容进行扩展

比如：SpringCloud 微服务

微服务必须是演进出来的

2、前置基础要求

**第一阶段：**小程序要求

JS、CSS、小程序基础、自定义组件、ES6、Promise

>> 这里会对小程序进行改造，改成都是 Promise 的风格

不管基础跟着项目走、自学参考、参考《纯正商业应用微信小程序实战》

**第二阶段：**Java & Java SpringBoot

良好的 Java 语法基础(注解、接口、类、IoC)

MySQL 的基础知识

**第三阶段：**Vue JavaSpringBoot

不做过多要求，有良好的小城基础 vue 其实就差不多了

初中高级 vue 技术点都会讲到

**第四阶段：**Java SpringBoot

这一阶段不是问题，主要是一些方法、思想，解决问题的思路

### appkey 申请、课程免费补充资料

我的 key，大家用不了：oyzTMfOJijBhOIBW

1、appkey 申请：

在http://www.7yue.pro申请。点击课程服务（没有注册的话先进行注册），选择对应课程进行申请 appkey，在个人设置里面的头像的下面就会有用户的 ID。这个 appkey 只能自己使用

2、 API 文档

点击右上角的专栏，点击 7 月的新全栈课程，就可以进行全栈课的 API 文档说明。

这里还涉及到错误码、api 清单。

后面也会涉及到一些常用软件的安装都会贴上来。

## 第一阶段程序预习与<风袖>架构

### 第一阶段第一次作业(风袖细节分析)

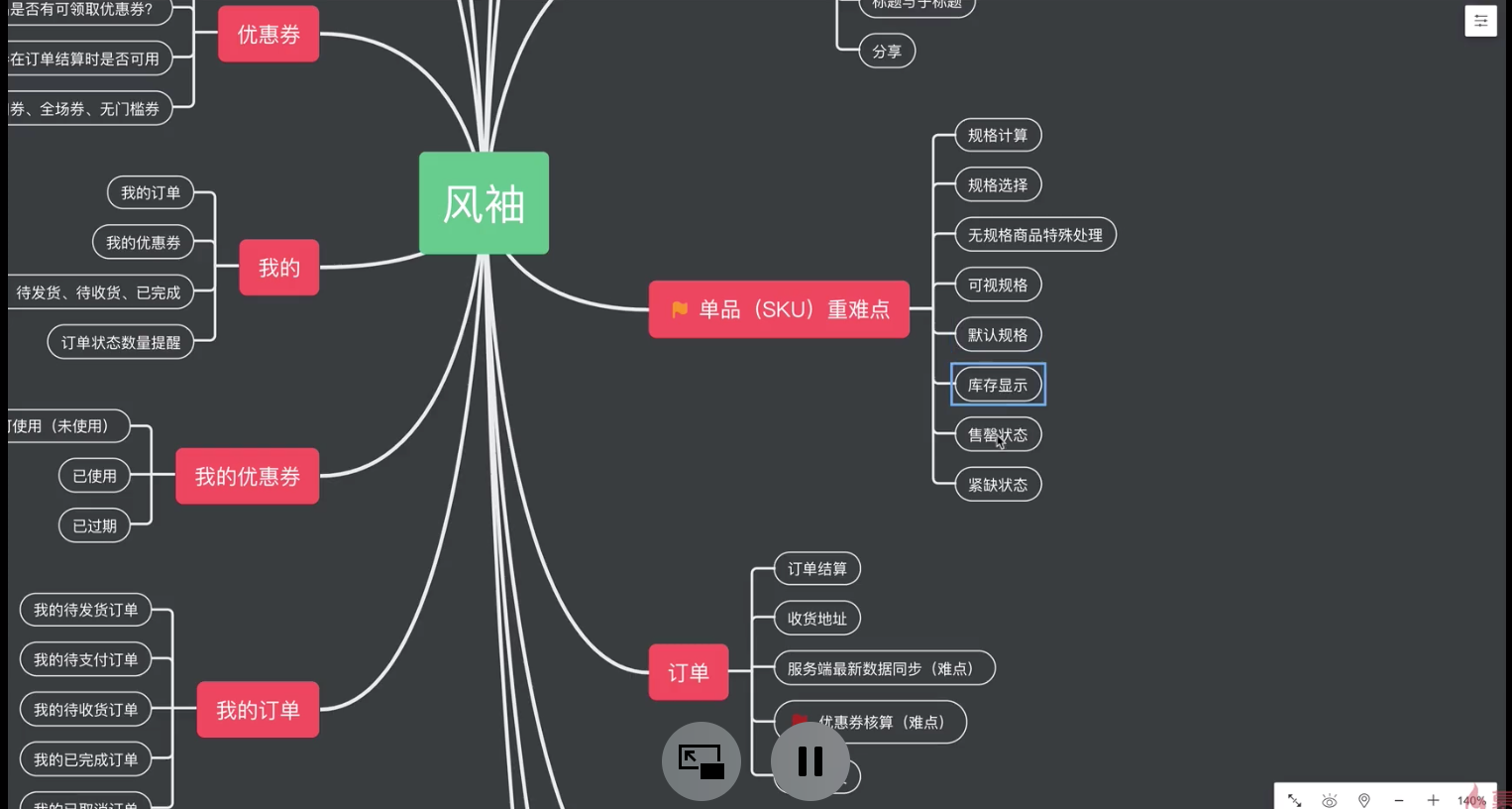
作业：自己研究《风袖》小程序，进行功能的梳理

目的 1：从全局理清风袖功能

目的 2：提升综合分析能力

成果：一份思维导图，思维导图贴到慕课网课程提问区。

要么就不做，要做就做到极致。还有就是讲课的时候不要是我觉得怎么样就怎么样，要你觉得怎么样就是怎么样。



这里 sku 的例子演示：sku 的各种情况。什么时候设置默认的 sku，什么时候不设置默认的额 sku，这个就涉及到产品、运营一些知识。

这里的举例子就是要对产品进行更深度的思考。

加入购物车的时候，sku 的选择计算是重点，并且还会涉及到反选的时候，有的没有，有的就有。如规格的不同，库存的不同，打折的不同，都会影响到最终的价格；并且请选择提示的文案是随着你选择的规格是变的，加入购物车警告的文案也是变的；并且如果库存不足的话提示的文案也会变，并且加入购物车也会变成暂时缺货；还有就是颜色规格是有图片的，就是可视规格

下面会对 CMS 进行一些功能的演示。

### 小程序注册与新建项目

1、搜索**微信公众平台**

2、点击小程序的详情，点击前往注册。然后填入邮箱、密码，尽量是企业的性质，可以使用微信支付

3、回到公众平台首页后，进行登录。

4、首先是获取到小程序的 AppiD，点击左侧开发，然后开发设置里面会有 AppID 和 AppSecret。

5、点击文档 -> 工具，进行开发工具的选择，目前有三个版本：稳定版、预发布、开发板。

稳定性从上而下依次减弱。预发布为候选版本。最终还是选择稳定版。

6、打开小程序开发工具后，点击+号，选择目录，以及项目名称，AppID，开发模式，后端服务（不使用云服务，小程序云开发-简单小项目），语言(JavaScritp和 TypeScript)

### 开发工具推荐

微信开发工具版本：v1.02.1907300版本

VsCode 和 WebStorm

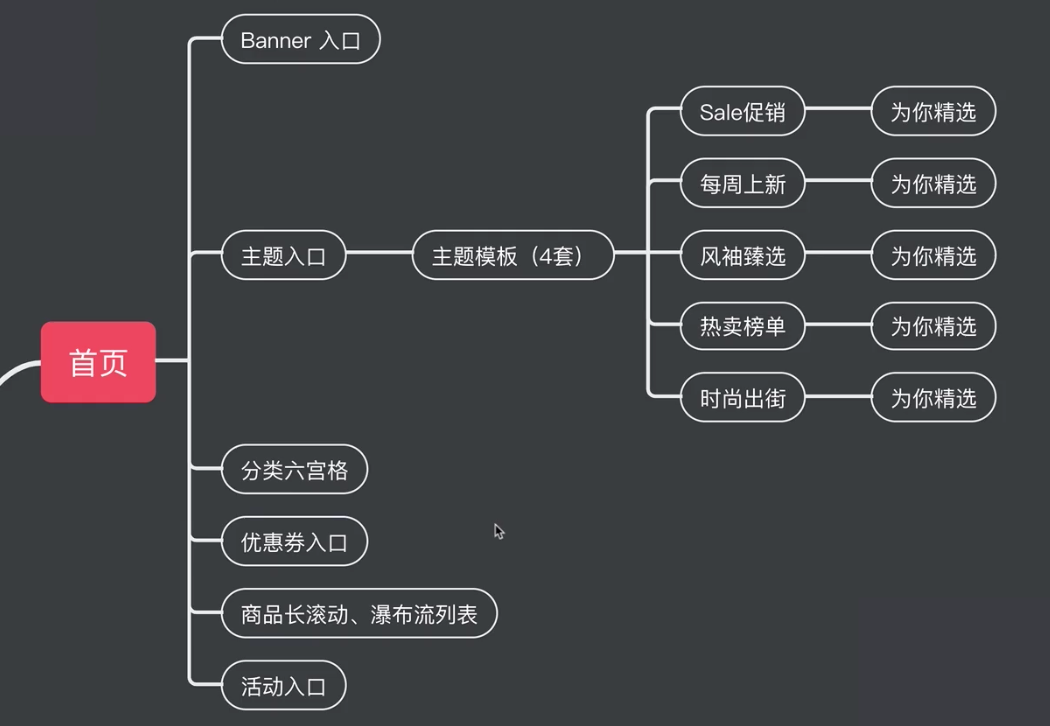
所有的小程序都是从示例开始，有个 HelloWorld 文字。

这里综合选择 WebStorm。使用 VSCode 也可以。

但是Java来的话目前 IDEA 是绝佳选择，Eclipse可能现在基本不用了。

如果有开源项目可以申请 License，并且是需要持续维护的，可以赠送 10 个账号。

### 《风袖》首页布局详尽分析



这里我们进行首页的布局讲解：（对首页进行提取概念，有了概念就会有逻辑的思考）

**主题对应的是商品的入口，活动对应的商劵的入口**

Banner：

分类六宫格：主要的一个分类入口

优惠券入口：

每周上新：Banner 和每周上新都是左右滑动的，这个为什么一个就是 Banner，一个是主题。主题有查看更多，主题一般都是统一入口。并且 Banner 是比较复杂的，有的是跳商品，有的是跳主题，有的是跳优惠券，甚至可以跳转到分类。

热卖榜单：是一个 Banner 和主题的混合，从前端解释不清楚，从后台看的话归为 banner 和归为主题都不好设计，唯独就是进行混合设计。

时尚出街：是一个主题

为你推荐：长滚动列表，底部下拉列表，其实就是一个瀑布流，并且尺寸不一样的，都能很好地显示，图片和文字都能撑开。小程序如果不设置图片的宽高，就按默认的宽高显示。

### 第一阶段第二次作业(LinUI瀑布流练习)

作业目的：培养查看文档、分析问题的一个能力。

看视频跟着敲代码对思考能力是没有什么用的，最好都是先动手做，全力以赴的，然后对比视频里面的实现思路，

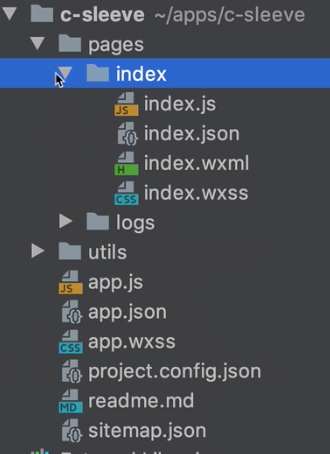
下面介绍 Lin-ui 组见，地址：https://github.com/TaleLin/lin-ui

这次是瀑布流作业，布局里面有个瀑布流

### 真实的项目必须是动态的、CMS 可控的

开始完成首页的工作。

默认的工程结构如下：



1、在小程序中创建我们自己的首页：

pages->home->新建 page（home）

2、设置首页：

普通编译模式：app.json->pages里面的第一个选项是默认首页

添加编译模式：在添加编译模式下，可以自定义设置主页。

3、删除 logs 目录

小程序就报错了，进行 app.json的一些内容清理

4、编写顶部的 sale 图片

这里从 cms 后台配置的图片显示，都是动态获取的。这个项目才真实。

下个小节开始调用后台 api 的进行开发。

并且不是对着已经做好的小程序开发了，不利于培养从静态的效果图到真实的产品的一个思考过程。通过自己对业务的理解，猜测，思考出来一个很好的结果。

### webstorm 开发小程序必备

高度是 130px=260rpx。

home.wxss:

|  |
| --- |
| .top-theme {  width: 100%;  height: 260rpx;  } |

webstorm不支持 rpc 单位，可以关闭这个检测

home.wxml:

|  |
| --- |
| <view>  <image class="top-theme" src=""/>  </view> |

下面说一下 webstorm 的对小程序的配置；

1、取消对rpx 的检查

取消css语法检查: preference->editor->inspection->invalid css property value 取消掉钩钩

2、格式化代码：option+cmd+L对 rpx 的支持

需要利用文件监听： preference->Tools->File Watchers

点击下方 + 号，新增一个文件监听

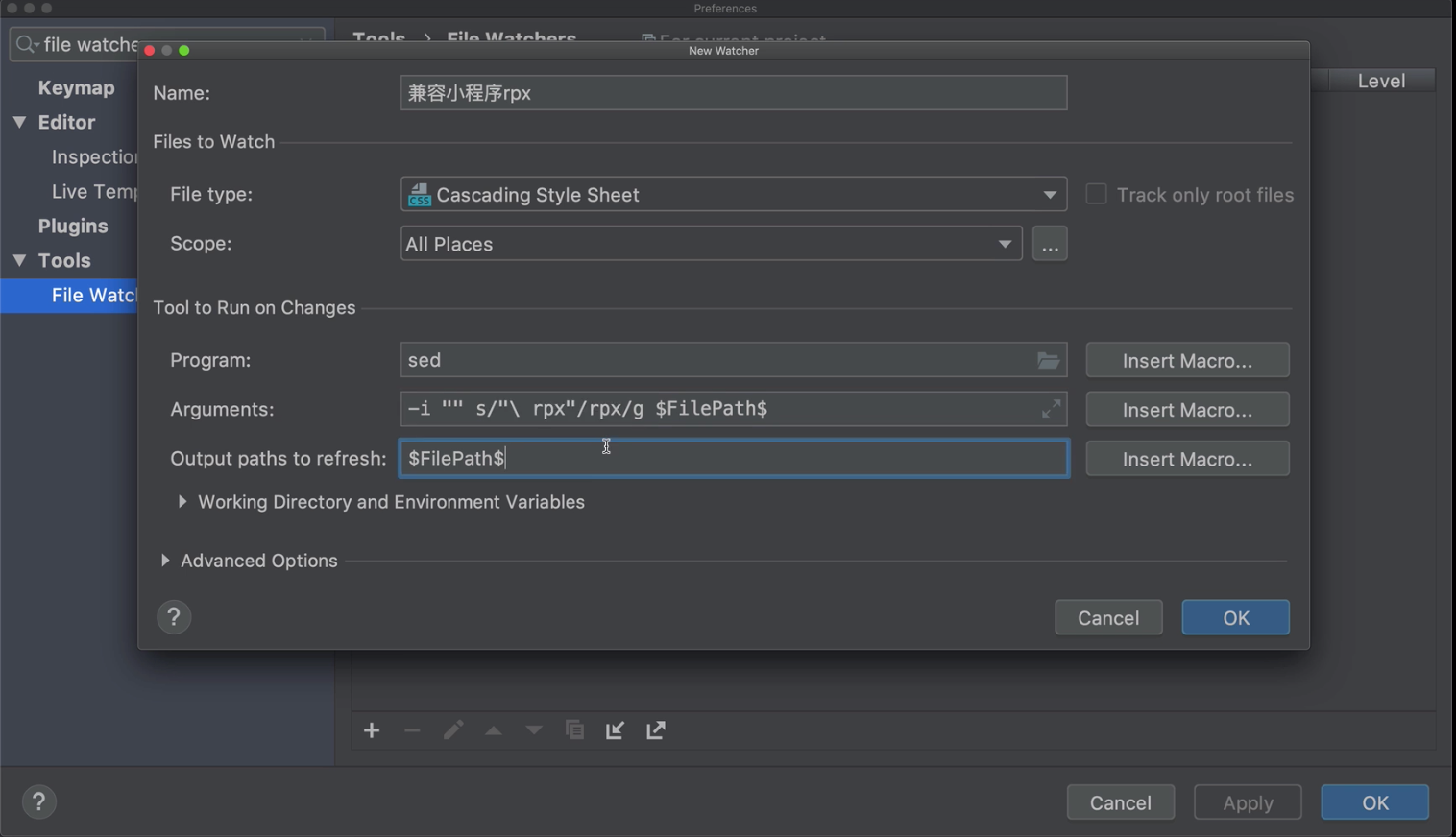
name: 自定义即可

File type: 选择 Cascading Style Sheet

Program: 写上sed

Arguments: -i "" s/"\ rpx"/rpx/g $FilePath$

Output paths to refresh: $FilePath$

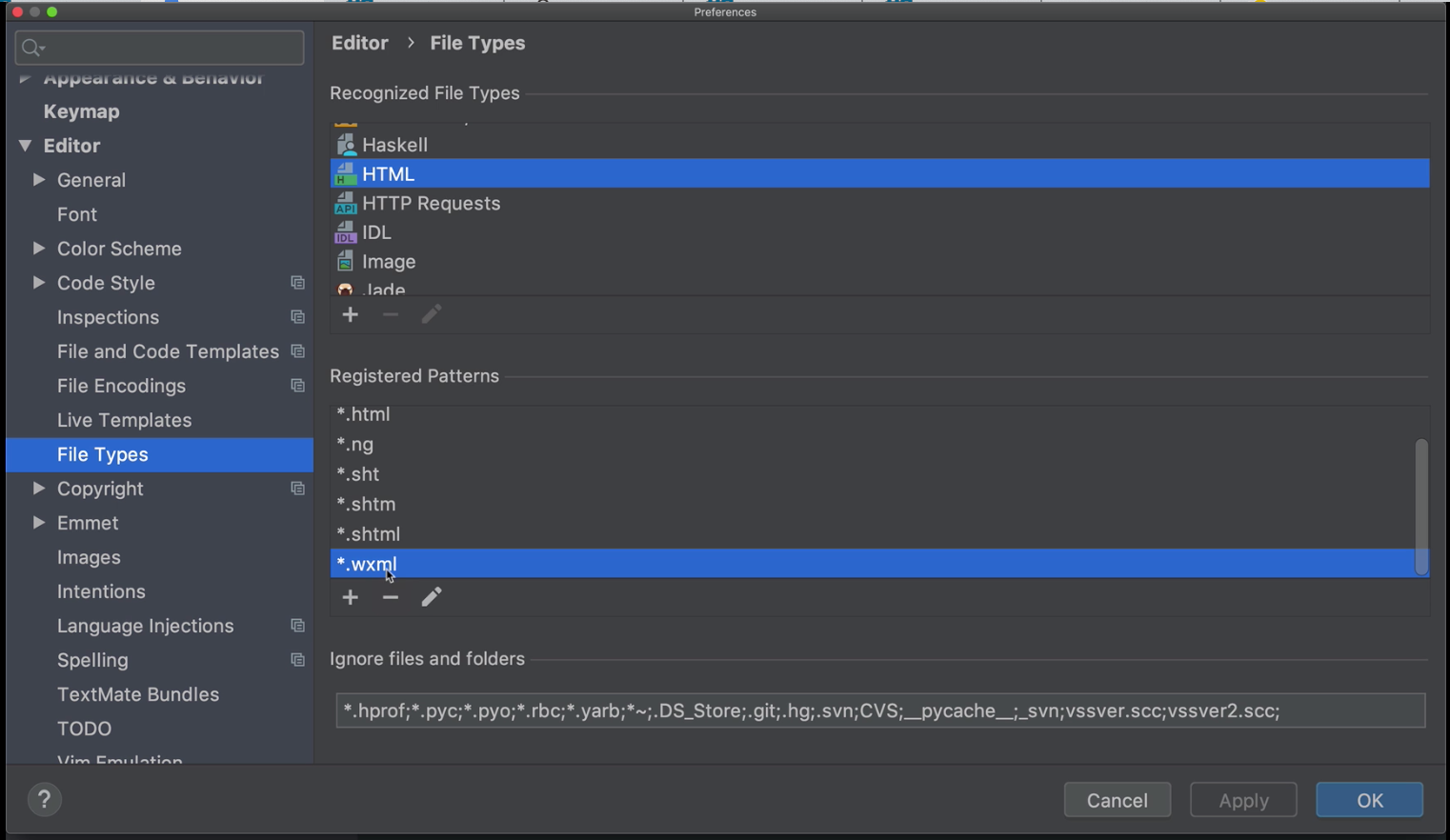


3、支持wxml,wxss 等标签

将wxss和css文件类型进行关联，同样，将wxml和html文件类型进行关联

preference->editor->file types

选中Html，在下面添加\*.wxml  
选中Cascading Style Sheet ，在下面添加\*.wxss



### 调用服务端 API 获取数据

1、在 home.js里面的onload 方法里。

使用 wx.request方法

|  |
| --- |
| wx.request({  url: 'http://http://se.7yue.pro',  data: {  names: 't-1'  },  header: {  'appkey': 'zCr1cdeqweq'  },  success(res) {  console.log(res)  }  }) |

2、新建 config 目录，新建 config.js文件。

wxs 是不支持 es6 语法格式

|  |
| --- |
| const config = {  appKey: '',  apiBaseUrl: 'http://se.7yue.pro'  }  export {  config  } |

### 小程序开发前必备配置

在小程序开发工具里面进行设置。设置 -> 项目设置

>> 调试基础库不用改2.8.3(视频里面)

>> ES6转 ES5

>> 增强编译（支持 async 语法）

>> 不校验合法域名（开发阶段是 http 的，上线一定要是 https 的）

## http 请求的完美封装与前端架构探讨

### 配置文件与第一次调用服务端 API

1、home.js导入 config.js文件

|  |
| --- |
| import { config } from '../../config/config'  data: {  topTheme: null  }  onLoad: function(option) {  wx.request({  url: `{config.apiBaseUrl}/theme/by/names`,  method: 'GET',  data: {  names: 't-1'  },  header: {  appKey: ''  },  // success(res) { // 箭头函数避免回调函数里面 this 指代的问题  success: res => {  console.log(res)  this.setData({topTheme : res.data[0])  }  })  } |

2、home.wxml

|  |
| --- |
| <view>  <image class="top-theme" src = "{{topTheme. entrance\_img}}}"/>  </view> |

### 业务对象的重要性

现在说明wx.request的封装和重构。

现在分享一下前端的层次结构问题，就是 js 问题里面写什么。毫无疑问wxss里面是 css 样式，wxml是 html 标签内容。

页面的 js 进行数据的绑定，也就是 view 视图层，不应该有业务逻辑，作为一个桥梁，是一个中间层。

MVC：Controller、Model、View

阿里规范：service,manager

虽然前段不像后台那样分层复杂，那前端至少有一层，我们这里叫做 model 层

新建model目录，然后创建 model的各个业务类，也就是业务对象，这里创建home.js就不好了，这里是 theme 主题，创建了theme.js文件

业务对象：theme、banner、spu、sku、address、user

theme.js

|  |
| --- |
| class Theme{  // getTopTheme 名字不合适，首页那么多 theme，其他怎么取名  // getSaleTheme 如果是业务了，但是如果他变了，也不太好  // 最后的就是用位置等一些关键词，摆脱业务的意义  static getHomeLocationA() { // 这里定义为 static，直接调用，为什么使用 static 了？后面会讲  wx.request({}) // 把上面小节的请求的代码移动到这里  }  } |

### 封装http请求

theme.js

|  |
| --- |
| // 导出  class Thema{}  export {  Theme  }  // 在success回调  getHomeLocationA(callback)  success: res => {  callback(res.data)  }  // 在此文件中wx.request太多，这里进行重构 |

home.js

|  |
| --- |
| onLoad: function(option) {  Theme.getHomeLocationA(data => {  this.setData({topTheme: data[0]})  })  } |

utils目录新建http.js文件

http.js:

|  |
| --- |
| class Http {  // {} 加上这个符号就可以对象传参数了  static request({url, data, method='GET', callback}) {  wx.request({  url: `${config.apiBaseUrl}${url}`,  method: method,  data: data,  header: {  appKey: config.appKey  },  success(res) {  calllbakc(res.data)  }  })  }  }  export { Http } |

theme.js

|  |
| --- |
| static getHomeLocationA(callback) {  Http.request({  url: 'theme/by/names',  data: { 'names': 't-1' },  success: data => { callback(data) }  })  } |

这是第一次重构代码，不够也有缺点，就是回调函数里面有回调函数，不太好，后面肯定会再进行重构的。

### 小程序中使用async和await难点分析

上面对 request 请求稍微简介了，但是还是有回调函数的调用的

异步调用的方式：callback -> promise -> async await

函数内部使用 await 的时候，必须使用 async 修饰该函数

**难点：**但是要 await 使用的时候，需要返回的是 Promise 对象，才能使用 await。但是 wx.request 这个版本还没有返回 Promise

### 使用 LinUI Promisic 让小程序内置 API 支持Promise

先将代码拷贝到工程里面

新建 utils -> utils.js:

|  |
| --- |
| const promisic = function (func) {  return function (params = {}) {  return new Promise((resolve, reject) => {  const args = Object.assign(params, {  success: (res) => {  resolve(res);  },  fail: (error) => {  reject(error);  }  });  func(args);  });  };  }; |

http.js:

这里将一个函数传入了一个函数的参数里面，在动态类型的语言里面很常见的，如 python

|  |
| --- |
| promisic(wx.request)({  url: ,  data: {}  })  static async request({}){  await promisic(wx.request)({    })  } |

promisic使用的是代理设计模式：

### 将回调函数全部替换为async和 await

async 和 await 是 ES7 的语法，需要勾选**增强编译**的选项

一个函数的前面加上 async 后，函数最终返回的结果一定是 Promise 类型的。

theme.js:

|  |
| --- |
| static async getHomeLocationA(){  return await Http.request({  url: '',  data: { names: 't-1' }  })  } |

home.js:

|  |
| --- |
| onLoad: async function(){  const data = await Theme.getHomeLocationA()  this.setData({ topTheme: data[0] })  } |

http.js:

|  |
| --- |
| const data =  return data.data |

### 获取 Banner 数据

参数命名都没有使用有关于业务的，这里使用b-1,b-2，但是看不见是哪里的 banner。

实际情况，哪里都有 banner，哪里都有主题。有很多的，所以需要使用这种跟业务没有关系的名字来命名。

使用文档说明里面的 API 来获取数据

新建：model -> banner.js

|  |
| --- |
| class Banner {  static locationB = 'b-1'  static async getHomeLocationB() {  return await Http.request({  url: `/banner/name/${Banner.locationB}`  })  }  } |

home.js

|  |
| --- |
| data: {  themeA: null,  bannerB: null  }  onLoad: async functiiion() {  initAllData()  }  async initAllData() {  const bannerB = await Banner.getHomeLocationB()  const themeA = await Theme.getHomeLocationA()  this.setData({ themeA: themeA[0], bannerB })  } |

### banner 轮播图实现与插槽的基本概念

数据解释：items 数组是 banner 集合

type: 跳转类型

img：图片的url

这里使用 slot 插槽，image 是自定义标签

home.wxml

|  |
| --- |
| <view>  <img/>  <swiper class="siwper-img" indicator-dots incicator-active-color="#157658" autoplay circular>  <block wx:for="{{bannerB.items}}">  <swipter-item>  <image class="siwper-img" src="{{item.img}}"></iamge>  </swipter-item>  </block>  </swiper>  </view> |

home.wxss

|  |
| --- |
| .siwper-img {  width: 100%;  height: 360rpx;  } |

### npm的 semver 语法规则

这里进行分类六宫格的编写。

服务器 API：category 商品分类 ->> 获取六宫格数据

如果手动写起来六宫格还是比较麻烦的，写的比较灵活的

这里使用 linui 组件的宫格组件，linui 使用的就是小程序自定义组件编写的，非常灵活，不仅可以用在原生小程序中，还可以用在 mpvue 等三方框架中。

首先cd 到小程序的目录中，执行 npm init 中，进行一些列目录、描述的设定，完成以后就会在目录中看见 package.json 的文件。

**安装 linui：**

**第一种方式：**在 package.json 文件中编写如下的代码，然后执行 npm i 命令，等价于npm install

|  |
| --- |
| "dependencies": {  "lin-ui": "**~**0.6.0"  } |

波浪线的意思：去选择库里最后一个版本号为最高版本的

**semver 语法规则**：定义了一些语法规则，如版本号，第三位为修复版本号，第二为次要版本，第一版本为主要版本，~ ^ 这些都在这个语法里面有讲。

对于开源版本的版本号是有必要知道这些规则的，如我修复了一个 bug，现在是0.6.1，这时候我就需要升级到0.6.2。这个时候使用者不需要修改版本号，只需要升级就可以。

但是对于包嵌套的问题，也就是依赖的问题，你没有办法修改里面的版本，这个时候波浪线的写法的好处就体现出来了：安装的时候就会安装最新的第三位版本。

**^** ：这个符号为次要版本为主，如果次要版本有最新的，那么就更新次要版本为最新的

### LinUI 安装、主题色配置与按需加载

小程序中：工具 -> 构建 npm

前提是需要有 package.json 文件

**第二种方式：**

直接 clone 的 Lin-ui 的代码，拷贝 dist 目录下的代码到项目的代码。

**最后建议使用 npm 方式的安装，更新组件会方便点，也就是第一种方式。**

**特性：**

1、自定义主题色

默认的主题色是蓝色，有时候需要修改颜色，可以看 Lin-UI 的文档，修改完颜色后需要重新编译

2、按需加载

可以修改 components 的内容，来选择自己需要的一些组件，然后编译就可以缩小整个小程序打包的体积。

**升级版本：**

1、删除 node\_modules，miniprogram\_npm/lin-ui

2、执行 npm i 命令，重新安装就可以

## 自定义组件与 LinUI 的小试牛刀

### LinUIGrid 组件的构建分类六宫格(上)

home.wxml 的元素比较多，写在一个文件中也可以，但是代码过多不利于维护，同样也会导致 wxss 和 js 文件都非常多的代码。

目录结构：{project\_home} -> components -> category-grid -> new Component

>> 工程根目录新建 components 目录：统一方所有的自定义组件的代码

>> 新建一个 category-grid 目录方分类六宫格，在小程序中新建一个组件，就会自动生成四个文件。这里自定义组件命名就写为 index。

app.json

|  |
| --- |
| {  "usingComponents": {  "l-grid": "/miniprogram\_npm/lin-ui/grid/index",  "l-grid-item": "/miniprogram\_npm/lin-ui/grid-item/index ",  }  } |

index.js

|  |
| --- |
| Componet({  // 定义外部属性  properties: {  // grid: Array  grid: {  type: Array,  value: []  }  },  // 内部的数据变量  data: {    },  methods: {    }  }) |

index.wxml

|  |
| --- |
| <view class="container">  <l-grid l-class="inner-container">  <block wx:for="{{grid}}">  <l-grid-item key="{{index}}" slot="{{index}}">  <view class="grid-item">  <image class="img" src="{{item.img}}">  <text class="text">{{item.title}}</text>  </view>  </l-grid-item>  </blokc>  </l-grid>  </view> |

index.wxss

|  |
| --- |
| .container {  heitht: 320rpx;  width: 100%;  display: flex;  flex-direction: row;  align-items: center;  justify-content: center;  }  .inner-container {  height: 300rpx;  width: 730rpx;  border: 1px dashed #dcebe6  }  . grid-item {  // height: 200rpx; 这是没有必要设置高的，grid 组件会自动计算高的  width: 200rpx; // 宽设置其实不影响布局，只是会影响触摸区域的大小  display: flex;  flex-direction: column;  align-items: center;  justify-content: center;  }  .img {  height: 60rpx;  width: 60rpx;  }  .text {  margin-top: 6rpx;  font-size: 24rpx;  color: #333333;  } |

home.js

|  |
| --- |
| data: {  bannerB: null,  grid: []  }  async initAllData() {  const grid = await Category. getGridCategory()  this.setData({  themeA: themeA[0],  bannerB,  grid  })  } |

model-> category.js

|  |
| --- |
| class Category {  static async getGridCategory() {  return await Http.request({  url: `category/all`  })  }  } |

home.wxml

|  |
| --- |
| <view>  <image>.....</image>  <swiper>......</swiper>  <s-category-grid grid="{{grid}}"></s-category-grid>  </view> |

home.js

|  |
| --- |
| {  "usingCompoents": {  "s-category-grid": "/components/category-grid/index"  }  } |

### LinUIGrid 组件的构建分类六宫格(中)

继续丰富上面的代码，主要是 html 和 js 的书写

发送 http 请求应该写在哪里？

1、组件里面的 js 方法 -> 尽量避免这种

2、在组件调用的地方调用，数据传到组件里面去 -> 比较规范的方法

### LinUIGrid 组件的构建分类六宫格(下)

这一节主要是编写样式(css)，代码也是参考(上)里面的代码

inner-container 设置样式以后，发现没有生效，对自定义组件使用样式时，不能使用内部样式类(class)，而是用外部样式类(l-class)

进行水平居中，以及 item 的垂直居中。

item 的 width 和 height 的大小设置技巧：height 不用设置，会被撑开，width 其实不用设置也可以保持六宫格的布局，grid 组件会自动计算好，但是设置的大小会影响触摸区域的大小

另外也可以使用 row-num参数来设置每行的个数，不过该数过多的时候，需要调整 item 的 width 大小。

细节：引入组件名和使用的时候，名称不对，如果使用不对的话，小程序是不会报错的

### 组件设计与 LinUI 使用的几个非常重要的原则

微调一下文本的样式，代码见上述的(上)章节

1、组件灵活性、易用性、稳定性之间做出一个选择，或者找到一个平衡点。

2、组件的意义？

vue 里面都是组件来进行开发，对于小程序中，Page其实都可以替换为 Component 关键字，这个就是比Page灵活、可以继承

组件的三大特性：样式、骨架、业务逻辑(行为)

组件在业务比较单一的时候就不会怎么体现出复杂性，如果业务情况比较多的时候，就会出现很多逻辑，这个时候一个很好的组件就显得很重要了。

既然是组件，必然需要考虑到它的灵活性，必须提供一些机制和手段来让开发者进行一些定制化开发：

1、外部样式类

2、slot 插槽

3、业务逻辑改造 -- 没有一个很好的支撑，另外的说就是 behavior，可以提供一些行为来让开发者进行选择，但是 behavior 的问题反馈的比较少。

一个很好的组件库必须在样式和骨架方面比较优秀。

对于业务逻辑来说：如果改动比较大的话，那么组件就不属于该组件的特征了。

经验性的思考比技术有时候会更重要

LinUI 组件：更加灵活点，提高生产力

默认值：来对样式和骨架进行设置默认值，来均衡这两个点。

### 如果没有明确理由，不要固定高和宽

根据 grid 组件来引入这个问题。

当你看到某个元素的时候，想一下有没有必要设置它的高和宽，也就是如果我不设置它的宽和高，会有什么影响？

如果不设置宽和高，么有很大的影响，那么就不要设置宽和高。

如果有理由就设置宽和高，如 grid-item 来说，设置宽度就是为了扩大点击的区域，而高度可以由里面的图片和文字撑开。

让很多东西成为自适应，尽量不要写成固定的值。

第二周：理解 RESTAPI开发开启三端分离编程

本周将正式进入电商项目的前端开发环节，在开发前会先讲解REST API开发规范，前端是怎么配合服务端同学工作的，再开始电商首页开发，更深入掌握小程序组件化编程。

## 渐入佳境，组件化编程

### 优惠券的一些基本概念

上面的六宫格还有点击事件等一些没有做，后面我们会继续去做

优惠券的复杂度（前端）：核算

优惠券是分类型的：满减、折扣

优惠券使用条件：有的并不是全场通用的，只是适用于某个品类的

优惠券过期：前端判断是否过期，或者后台判断后直接返回是否过期字段

不过优惠券的最大的难度还是在服务端，会经过很多的复杂计算，后面讲后台的时候应该会讲到。

优惠券的**领取**和**使用**是分开的，需要区分这两个概念

优惠券活动：

>> 有 start\_time,end\_time;

>> 也会有online的概念;

>> 并且会有分类的概念（放弃），最终采用 coupon 本身的做法(优惠券类型)来达到这个目的

用户领取、用户使用、过期、优惠券类型

### 优惠券入口

优惠券是以活动形式存在

接口文档：Activity 优惠券活动

插曲：category.js -> getGridCateogry 重名为： getHomeLocationC

model -> activity.js

|  |
| --- |
| class Activity {  static locationD = 'a-2'  static async getHomeLocationD() {  return await Http.request({  url: `activity/name//${Activity.locationD}`  })  }  }  export { Activity } |

pages -> home -> home.js

|  |
| --- |
| data: {  activityD: null  }  async initAllData(){  const activityD = await Activity. getHomeLocationD()  this.setData({  activityD  })  } |

pages -> home -> home.wxml

|  |
| --- |
| <view>  ... 前面的代码  <image class="activity" src = "{{activityD.entrance\_img}}"/>  </view> |

pages -> home -> home.wxss

|  |
| --- |
| .activity {  margin-top: 20rpx;  width: 100%;  height: 310rpx;  } |

图片和上面的六宫格是有间距的，并且还有背景色，怎么设置了？

方法一：创建一个 view，设置这个 view 的的颜色是这个灰色。不太好（页面这种的太多了）

### 背景颜色到底怎么设置

小程序：框架 -> 页面配置 -> backgroudColor

下拉的时候显示出来的区域就是这个 backgroudColor，所以是不适合的

**方法一：**修改 home 中的页面的 view 的背景色为灰色。这里修改为黑色进行测试，会发现有下面的问题

home.wxss

.container {

backgroud-color: #ffffff;

}

**方法二：**app.wxss

page {

backgroud-color: #f5f5f5;

}

问题：

1、顶部有黑色：container样式和 app.wxss中的.container的样式冲突了

2、sale 图片下面有个间距，也是黑色的，小程序自带的边距

解决方法添加图片样式：display: flex

3、六宫格也是黑色的：components -> category-grid -> index.wxss

.container {

backgroud-color: #ffffff;

}

### 页面到底是否应该合并 http 请求

每周上新模块 <--- 也是作为主题

model -> theme.js

|  |
| --- |
| class Theme {  static locationE = 't-2'    // 这里还有 A 主题的，可能还有 B 主题、C 主题。  // 所以这里根据 names 参数来说，一次性可以获取所有的主题  static async getHomeLocationE() {  return Http.reqeust({  url: `theme/by/names`,  data: { names: Theme. locationE }  })  }  } |

选择方案：

a、每一个数据发送 http ☆

b、Home 只发送一个 http ☆

c、有选择的把部分 http 请求合并成一个 ☆☆☆

评判准则：

1、http请求数量

2、http 多少次数据库查询，用O来衡量复杂度，这是主要指标

3、接口的灵活性，接口的维护性。

对于 web 应用来说，是 IO 密集型的应用，主要是对于数据库查询

对应有 CPU 密集型的应用：视频转码解码

### 函数式编程概述

对上面的代码进行重新书写

|  |
| --- |
| static locationA = 't-1'  static locationE = 't-2'  static locationF = 't-3'  static locationH = 't-4'  static async getThemes(){  constt names = `${Theme.locationA},${Theme.locationE},${ Theme.locationF }`  Http.request({  url: `theme/by/names`,  data: { names }  })  } |

pages -> home -> home.js

|  |
| --- |
| async initAllData() {  const themes = await Theme.getThemes()  // names 里面查找name='t-1'的值  // 这里使用函数式编程，find,filter,map,some,reduce  const themeA = themes.find(t => t.name === 't-1')  const themeE = themes.find(t => t.name === 't-2')  this.setData({  themeA,  themeE  })  } |

### 第一阶段 第三次作业

对于 sale 的图片来说，可能后续会改变高度，对于运营来说，最好是动态的。

组件 -> image

aspectFill

最好的效果就是动态计算宽和高

作业：

1、想办法实现动态计算宽和高

2、瀑布流的图片的自动高度

想法：应该就是等比缩放的问题

### 类可以保存数据，但对象可以保存状态

问题一：名字 t-1,t-2,t-3在代码里都是硬编码

问题二：比如 themeA 来说的话，如果别处也需要这些数据，那么数据要请求两次 http 吗？

总结两种需求：

1、尽量减少 http 请求，也就是一次请求获取 themes

2、在页面任何位置都很方便的拿到 themes 数据。

数据可以存缓存、全局变量里，但是这些方法都不太好

保存数据：类的对象本身就有保存数据的功能

类是可以保存数据，但是类不能保存状态

但是类的对象是可以保存数据也可以保存状态的

### 重构 Theme 获取

按照上面讲的重点，来对 theme 的代码进行重构

theme.js

|  |
| --- |
| class Theme {  themes = []  async getThemes() {  const names = `....`  this.themes = await Http.request({  .....  })  }  async getHomeLocationA() {  const themeA = this.thems.find(t => t.name === Theme.locationA)  return themeA  }  async initAllData() {  const theme = new Theme()  cost themeA = wait theme.getHomeLocationA()  this.setData({  themeA  })  }  } |

和类相关的静态方，全部改为实例方法

调用方的调用过程是简单的

### 第一阶段第四次作业

技术输出能力和资料整理能力

作业：SPU 和 SKU 的概念，以一种的小教程(手记)的形式产出

SPU 商品

SKU 单品

SPU：Standard Product Unit 标准化产品单品

SKU：Stock keeping unit 库存量单位

学习的方式只是输入，不输出是不对的。输入和输出是非常紧密的

输出已有知识的体系，并且锻炼你的表达能力和逻辑思维能力，写作|读书|画画这都是可以提高能力

## Price、Scroll-View 组件的灵活使用

### scroll-view 组件的灵活运用

每周上新-》使用自定义组件

components -> **spu-scroll** -> index.js index.wxml index.json index.wxss

index.js

|  |
| --- |
| Component({  externalClasses: ['l-class'],  properties: {  theme: Object，  spuList：Array  }  }) |

index.wxml 具体代码见 6.4 小节

|  |
| --- |
|  |

index.wxss

|  |
| --- |
|  |

index.json

|  |
| --- |
|  |

theme.js

|  |
| --- |
| async initAllData() {  const themeE = await theme.getHomeLocationE()  let themeESpu = []  if(themeE.online) {  const data = await Thme.getHomeLocationESpu()  if(data) {  themeESpu = data.spu\_list.slice(0, 8)  }  }  this.setData({  themeESpu,  themeE  })  }  static async getHomeLocationESpu(){  return **Theme.**getThemeSpuByName(Theme.locationE)  }  static async getThemeSpuByName(name){  const theme = await Http.request({  url: `theme/name/${name}/with\_spu`  })  return theme  } |

这里又讨论下 static 什么时候用，什么时候不用的问题？

答案：是否保存状态，不需要的话就用 static，如果是保存状态，就做成实例的方式

### 是不是方法都需要加 async 和 await

第二个问题：是不是每个方法都需要加 async 和 await 了？

上面的两个方法其实都不用 await。

await 其实就是等待此次的调用结果，如果下面没有其他执行的代码，不需要等待，也就是不需要加的。

async 是否需要强制返回 Promise，如果需要的话就加上呗

接下来还是完善上面的代码，进行判断然后获取主题数据

开始使用spu-scroll 组件

home.wxml

|  |
| --- |
| <view>  <s-spu-scroll theme="{{themeE}}"  spu-list="{{themeESpu}}"  wx:if="{{themeE.online}}">    </s-spu-scroll>  </view> |

home.json

|  |
| --- |
| {  "usingComponents": {  "s-spu-grid": "/components/spu-scroll/index"  }  } |

### 第一阶段第五次作业

分页加载的问题

瀑布流的数据应该是动态加载出来的

考虑到的问题：底部需要显示正在加载的状态，如加载完成的状态，如果是到底了，我是有底线的状态。并且考虑到重复加载的问题

考虑到组件的封装

提交形式：问答区，描述思路，并且截图看见各种状态

### spu-scroll 自定义组件 1

编写 spu-scroll 的组件

index.wxml

|  |
| --- |
| <wxs src="../../miniprogram\_npm/lin-ui/filter/string.wxs" module="s"></wxs>  <view class="container l-class">  <image class="title" src="{{theme.title\_img}}"> </image>  <scroll-view scroll-x class="scroll-view">  <view class="innter">  <block wx:for="{{spuList}}">  <view class="spu-container">  <image class="spu-img" src="{{item.img}}"></image>  <l-price autofix  l-unit-class="price-unit"  l-price-class="price-value"  value="{{item.price}}"></price>  <text class="spu-text">  {{item.title.length >= 8 ? s.substring(item.title 0,7)+'...': item.title }}  </text>  </view>  </block>  </view>  </scroll-view>  </view> |

index.wxss

|  |
| --- |
| .container {  padding: 42rpx 0rpx 34rpx 28rpx;  }  .title {  width: 694rpx;  height: 90rpx;  }  . price-unit {  font-size: 24rpx !important;  color: #157658 !important;  }  .price-value {  font-size: 26rpx !important;  color: #157658 !important;  }  . spu-img {  width: 160rpx;  height: 160rpx;  }  // 图片和价格以及文字在列方向上居中，使用 flex 布局  . spu-container {  margin-right: 60rpx;  display: flex;  flex-direction: column;  align-items: center;  }  . scroll-view {  margin-top: 28rpx;  display: flex;  flex-direction: row;  }  .inner {  display: flex;  flex-direction: row;  }  .spu-text {  font-size: 20rpx;  color: #666666;  margin-top: 20rpx;  /\*width: 150rpx;  overflow:hidden:  text-overflow:ellipsis;  white-space: nowrap;\*/  } |

分类、购物车都会使用 scroll-view 这个组件。但是高度有时候是动态的

### LinUI Price价格组件使用

使用 LinUI Price 组件

app.json <<-- 引入 lin-ui，price 组件

|  |
| --- |
|  |

功能强大的还是使用外部样式类

然后进行页面的编写以及样式的书写，代码见上面的小节

l-unit-class和l-price-class可以设置不一样的样式，如果是一样的，就可以用 l-class样式

autofix可以设置价格显示的格式，如 399.00 样式的

!important 需要写的原因：小程序内部的样式有着不确定性的优先级，就算放在后面，有时候也不会覆盖原先的样式。这里又涉及到 css 样式的权重，最简单的就是!important的使用

第三周：热身结束，开启试炼

前两周我们节奏稍缓，主要是给大家留出做作业的时间，本周开始我们逐步结束热身环节，开启进阶的内容，所以从本周开始我们节奏会加快。本周主要完成首页的热卖、上新、底部长滚动列表和瀑布流。

## WXS、LinUIFIlter 过滤器与热卖榜单的实现

### spu-scroll自定义组件 2

继续在 6.4 上修改代码

这里主要是编写 flex 布局。

第一点：对 spu-container 进行 flex 布局

第二点：对 scroll-view 进行 flex 布局，有问题的？enable-flex：开启 flex 布局

第三点：scroll-x，开启横向的滚动

第四点：设置整体的 container样式，内边距这些。注意：只要左内边距

第五点：到这里的时候，下面就会很多的空白，为什么会撑开了这么多的空间了？，下节进行解决问题

### 通过 scroll-view 抽象出的几个 css 编写规则

这里解决7.1 中的第五点问题，猜测 scroll-view 的问题，可以看成是功能更丰富的 view 组件，并不能完全与 div 对等。

那么为什么下面撑开了那么多的空间，推测：没有 flex 布局的时候 8 个依次的空间，好像是这么大，

将 scroll-view 换成 view 的话就没有了底部的空间。

解决办法：

1、设置 scroll-view 高度；

2、但是这里的情况又不好计算高度，那么怎么做了？现在里面写个 view，设置这个 view 为 flex 布局，但是还是有高度。这里再次猜测是enable-flex的问题，去掉以后就解决掉了

两个原则：1、多去尝试，不一定对着文档就能找到正确答案，从现象反推出现的原因

2、前端 css 样式的时候高度不一定要写死了

3、关键需要抽象出一些原则出来，对以后的编程是有很大的提高的。

### wxs 与小程序的内核简介

1、对上面代码进行小调整工作。详细见6.4 代码

2、对字体长度比较长的时候，需要截取字符串显示。三种办法：①：js 处理，②：wxs 处理；③css 处理

所有的 wxs 代码都可以写在 js 里面。但是大量的代码是不好的。因为 js 是逻辑处理的，对于样式这样的处理是不太好的。

wxs 本身是 JavaScript，那么与页面和组件写的 js 代码有什么不一样了？

答案：逻辑层/视图层，逻辑层都是 js 代码，处理逻辑专用的；那么在 wxml 里面又不能写 js 代码，那么 wxs 就是干这个用的，也就是在页面上的一段 js 代码。

wxs 只支持 ES5 语法，js 文件里面的最高支持到 ES7语法了

**小程序运行原理**：IOS，Android，开发工具。主要是 web 技术作为驱动

IOS：逻辑层：JavaScriptCore 视图层：WKWebView

Android：逻辑层：V8 视图层：XWeb

开发工具：逻辑层：Nw.js 视图层：Chromium

### wxs 与 lin ui filter 的应用

代码见 6.4 小节。

**css方式：**没有出现省略号的效果(要出现省略号就需要固定宽度)

但是也会有问题，文本过短的时候，150rpx 的宽度是居中的，但是文字是不居中的，所以也不太符合要求

|  |
| --- |
| width: 150rpx;  overflow:hidden:  text-overflow:ellipsis;  white-space: nowrap; |

**wxs 方式：**直接使用 lin-ui里面的过滤器(在 wxml 里面引入 wxs 文件，然后使用里面的函数)

|  |
| --- |
| <wxs src="../../miniprogram\_npm/lin-ui/filter/string.wxs" module="s"></wxs> |

这里只是初步介绍 wxs 的功能

### webstorm 的骚操作自定义代码片段

Editor -> Live Templates

添加分组： Template Group

再点击+号：Live Template

Abbraviation：填写快捷键命令

Template text: 需要生成的代码

选择上下文

### 自定义组件的边距问题与外部样式类应用

小小的细节需要完成，商品和图片需要有个间距，代码见 6.4 小节

pages->home->home.wxml

|  |
| --- |
| <view>  ...  <iamge class="activity" src=""> </image>  <s-spu-scroll  l-class="spu-scroll"  theme="{{themeE}}"  wx:if="{{themeE.online}}" >  </s-spu-scroll>  </view> |

第一种方式：image 加上下边距

pages->home->home.wxss

|  |
| --- |
| .activity{  // margin-bottom: 20rpx;  } |

完成以后，看起来没有生效，其实是生效的，

第二种方式：spu-scroll 添加上边距

|  |
| --- |
| .spu-scroll {  margin-top: 20rpx;  } |

自定义组件的时候，设置边距是失效的

原则：优先设置原生组件的样式类，如果是自定义组件必须要用了，那么建议用外部样式类，不用内部样式类

在组件的 js 中externalClasses: ['l-class'],定义此内容，那么就定义好了外部样式类,然后在组件的地方使用l-class="spu-scroll"调用样式。

第三种方式：在自定义组件外面加一层 view

|  |
| --- |
| <view>  ...  <iamge class="activity" src=""> </image>  **<view class=" spu-scroll "></view>**  <s-spu-scroll  theme="{{themeE}}"  wx:if="{{themeE.online}}" >  </s-spu-scroll>  </view> |

总结以上：这里推荐使用第一种方式，在 image 标签设置下边距来达到目的

### 学会抽象来简化数据库表设计

### 新监听器 Observers 与 hot-list 组件

### flex 布局的灵活应用

### hover 点击态的应用于@import公共样式表

## 瀑布流、抽象组件与分页插件封装

### 组件自定义性的终极方案：抽象节点

### 防抖、截流与分页细节全分析

### Paging 对象(1)

### 编程规则：多用 return 提前结束函数，少用 else

### Paging 对象(2)

### Paging 对象(3)

### 第一阶段第六次作业 Sku 预研作业

### Paging 对象(4)

第四周 攻克 SKU 和商品购物车

第四周很重要！我们会重点讲解SKU，SPU和购物车，完善购物车的复杂逻辑是每一个前端开发头疼的问题，更是电商项目的精髓所在，助力同学攻克难关！

## 瀑布流的静态样式

### l-water-flow 抽象组件的深入剖析

### LinUITag 组件快速构建标签

### 折扣价和原价常见错误逻辑

### 使用 wxs 解决负责的逻辑问题

### 瀑布流的初步成型

### 动态计算图片宽和高

### Loading 的思维逻辑：正在加载其实是一种常驻状态

### 实现页底提示

## 推倒 SKU 这个大怪物

### 页面跳转与新机制 EventChannel

### Spu,Sku,规格名和规格值

### 从一个提问谈思考的精神

### 调整思维，才能解决复杂的逻辑问题

### 借助符号化的命名方式思考问题

### 控制中心 Realm 与栅栏 Fence

### fence 与 fence-group 对象

## SKU 总体思路流程分析与矩阵思维简化规格值提取

### SKU 方案整理一：规格值的提取

### SKU 方案整理二：核心问题是规格值状态的确定

### 用矩阵的思想提取规格值

### 数学小课堂：矩阵的转置与旋转

### 矩阵的遍历

### Fence 的实例化

### 测试 Fences

### 自己实现矩阵的转置

### 谈数学、面向对象与思维编程

### 构建 Cell 对象

第五周 SKU 核心：规格状态、编码、无货逻辑

上周我们为SKU打下了坚实的理论基础，本章我们逐步进入SKU的核心算法

## Cell 对象

### SKU 串码编号

### Fence 与规格名的对应关系

### Cell 去重

### Cell 自定义组件

### 初步完成规格矩阵渲染

### SKU 算法的难点是确定规格的禁用状态

### SKU 核心算法剖析(1)

### SKU 核心算法其实就是一句话，我之前已经讲过了

### 矩阵列不同规格的问题解答

## SKU 核心算法与具体实现

### 数学小课堂：排列组合在 SKU 中的应用

### SkuCode 类拆分 Sku 可能的路径

### Combination 组合函数