[初始化在线教育APP项目 1](#_Toc487828495)

[创建项目 1](#_Toc1988434866)

[分类区功能实现 23](#_Toc484278248)

[近期上新功能实现 43](#_Toc301492006)

[热门/上新/免费/付费课程数据接口 60](#_Toc1262221569)

[分类页面功能 74](#_Toc1314445117)

[需求 74](#_Toc714804730)

[动态数据渲染分类页 79](#_Toc699575792)

[搜索关键字保存到Storage中 98](#_Toc297368819)

[课程列表数据渲染 128](#_Toc683294364)

[课程详情页数据接口 180](#_Toc1533315239)

[EasyMock 创建课程数据接口 180](#_Toc625457873)

[封装API请求接口方法 183](#_Toc138019446)

[创建与配置视频播放页面 210](#_Toc410490162)

[渲染商品信息 258](#_Toc1390678570)

[我的余额页面 263](#_Toc2070195689)

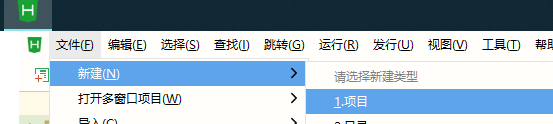
[服务端生成支付订单接口 280](#_Toc248896329)

[微信小程序支付实现 289](#_Toc2049940794)

# 初始化在线教育APP项目

mxg-education-app

## 创建项目

1. 文件-新建-项目
2. 打开新建项目窗口后，项目名 , 具体如下操作，



**安装scss/sass编译**

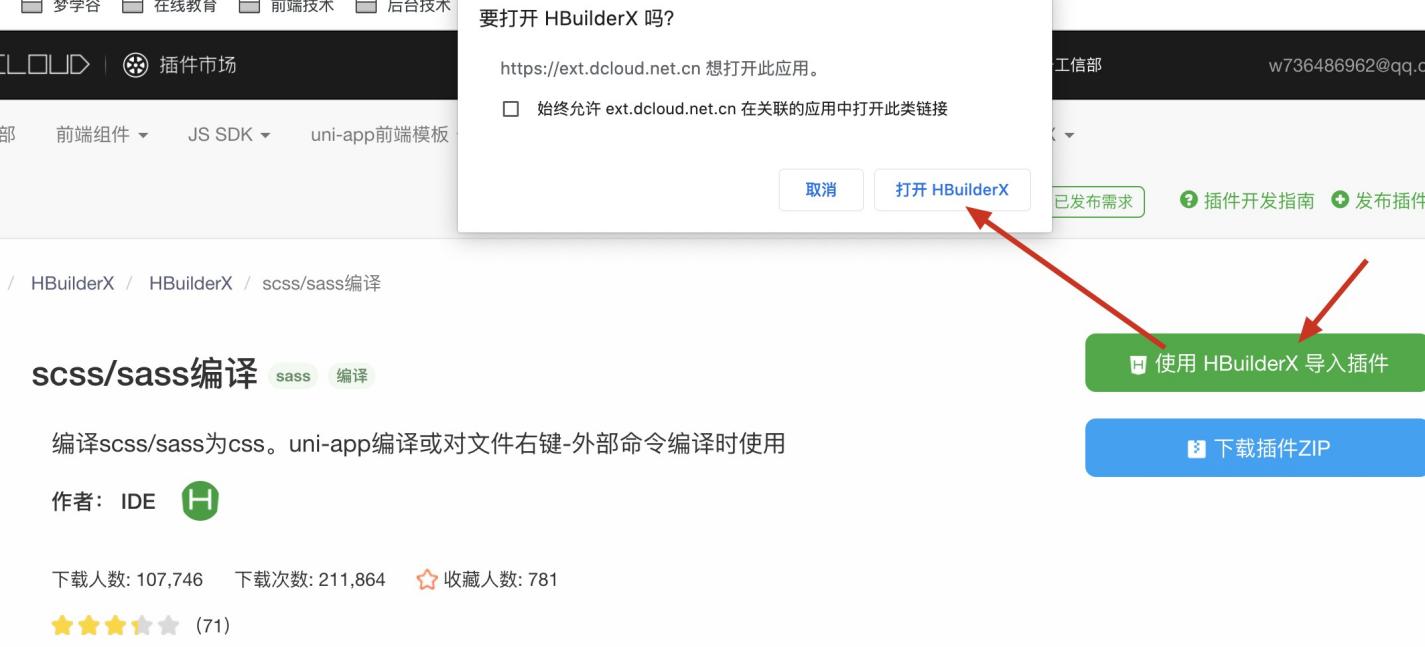
使用了scss/sass语法后需要编译为css。uni-app会自动编译或对文件右键-外部命令编译时使用。

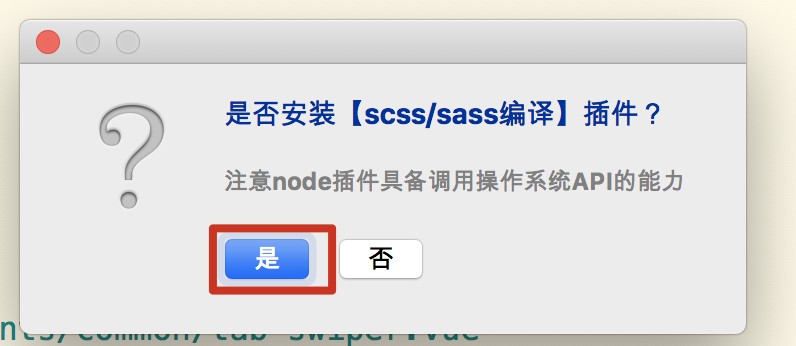
1. 浏览器访问下面链接，点击右侧 > 弹出窗口点击 ：

使用HBuilderX 入插件

打开HBuilderX

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



1. 在 HuilderX 会弹出是否安装，点击



是

1. 安装成功后，在页面的style标签上使用 lang="scss" 后即可编写scss代码

<style lang="scss">

</style>

**完善项目目录及文件**

提前创建好项目所需要的目录，在项目名右击》新建》目录，创建好后如下图：





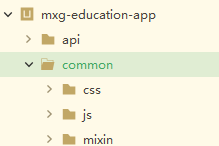
**等资源，components 目录放**

**less/scss**

**static 目录放图片、图标资源。common目录放 js 、 css 、**

**可复用vue组件**

**拷贝静态资源到项目中**

1. 将 下的目录文件拷贝到项目中

03-配套资料\common

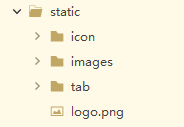
mxg-education-app\common

1. 完善 static 静态图片，将

03-配套资料\static

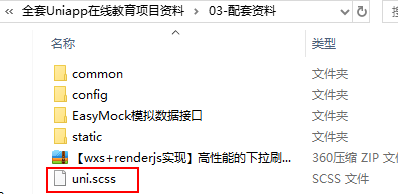
1. 完善 uni.css 统一样式，将

目录覆盖项目的 static 目录



文件覆盖项目根据目录的 uni.css 文件

03-配套资料\uni.css



1. 在 App.vue 文件中引入 common.css 公共样式文件

<style >

/\* 公共样式 \*/

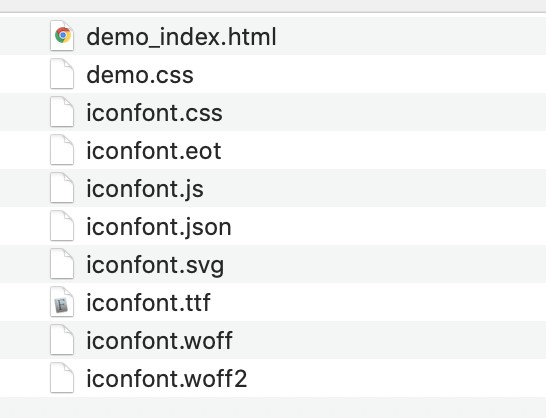
@import url("@/common/css/common.css");

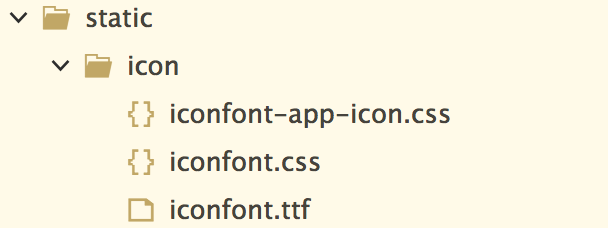
</style>

**iconfont 字体图标**

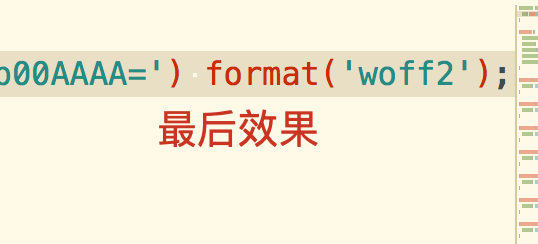
**下载图标库**

1. 进入阿里巴巴矢量图标库 <https://www.iconfont.cn/>，添加图标到项目，然后下载至本地
2. 下载解压后的本地文件如下：



1. 把 03-配套资料\static\icon 文件拷贝到 /static/icon/ 目录下
2. 修改 iconfont.css （Hbuilder 工具取消自动换行（**视图 > 自动换行**），不管多长都在一行显示） **修改后如下图，注意 src 最后加上个分号**





**vue 文件引用字体图标**

#### Font-class 类名引用步骤(推荐方式)

<style>

iconfont.css

1. 在 App.vue 文件的

标签下使用 import 导入

字体文件

**注意：font-class类名方式，微信小程序开发工具上不支持网络链接，只支持本地方式；打包上线时，只支持 在线方式**

<style>

/\* 公共样式 \*/

@import url("@/common/css/common.css");

/\* 本地方式 \*/

@import url("~@/static/icon/iconfont.css");

/\*

微信小程序开发工具上，不支持网络链接，只支持本地方式打包上线时，只支持在线方式

\*/

/\* @import url("https://at.alicdn.com/t/font\_2116121\_4hwc792r6yl.css");\*/

</style>

阿里 在线图标链接复制如下（一定不要忘记加 https://）

Font class

1. 挑选相应图标并获取类名，应用于 **.vue** 页面

iconfont.css

icon-haoping

注意： 和 是

iconfont

icon-haoping

文件声明的，

是具体的图标类名

<text class="iconfont icon-haoping"></text>

当前使用 font-class 方式引入字体图标后，也可以采用Unicode字符编码方式使用图标。

<text class="iconfont">&#xe64e;</text>

**实战项目配置项**

参考文档：<https://uniapp.dcloud.net.cn/collocation/pages>

**在项目 globalStyle 全局配置**

用于设置应用的状态栏、导航条、标题、窗口背景色等。下面配置项默认情况会应用于每个页面。

**注意：更改 pages.json 里面的配置信息后，最好是重新运行项目，并且手机上的app也要退出重新打开；不然有时候不生效。**

"globalStyle": {

// 状态栏与导航栏背景色

"navigationBarBackgroundColor": "#FFFFFF",

// 状态栏与导航栏字体颜色，仅支持 black/white(默认) "navigationBarTextStyle": "black",

// 上拉下拉回弹背景色（ios和真机微小程序）

"backgroundColor":"#F8F9FB"

}

**tabBar 底部标签导航配置**

**需求**

如果应用是一个多 tab 应用，可以通过 tabBar 配置项指定 tab 栏的表现，以及 tab 切换时显示的对应页。如下图：



**说明**

在 pages.json 中配置 tabBar , 参考：<https://uniapp.dcloud.net.cn/collocation/pages?id=tabbar>

tabBar 中的 list 是一个数组，只能配置最少2个、最多5个 tab，tab 按数组的顺序排序。

tabbar 切换第一次加载时可能渲染不及时，可以在每个tabbar页面的onLoad生命周期里先弹出一个等待雪花（hello uni-app使用了此方式）

tabbar 的页面展现过一次后就保留在内存中，再次切换 tabbar 页面，只会触发每个页面的onShow，不会再触发onLoad。

当设置 position 为 top 时，标签导航栏放到顶部，则不可以显示图标，只有文字。

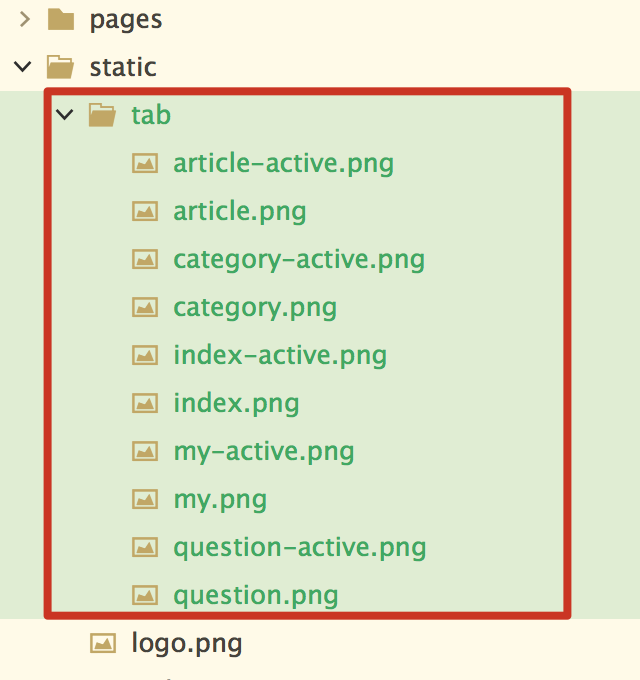
顶部的 tabbar 目前仅微信小程序上支持。需要用到顶部选项卡的话，建议不使用 tabbar 的顶部设置，而是自己做顶部选项卡，可参考 hello uni-app->模板->顶部选项卡。

**代码实现**

1. 复制 tabBar 图片目录点击后，1张未点击。

03-配套资料\静态资源文件\tab

到项目的 **static** 目录下。其中每个标签2张图片，1张



1. 在 pages.json 文件中配置 tabBar ，有5个标签页。tabBar 选项中的pagePath 必须在pages数组中存在注意：任何地方都不能多有一个逗号，

{

"pages": [

//pages数组中第一项表示应用启动页，参考：https://uniapp.dcloud.io/collocation/pages

{

"path" : "pages/index/index" // 文件名不需要写后缀

},

{

"path" : "pages/category/category"

},

{

"path" : "pages/article/article"

},

{

"path" : "pages/question/question"

},

{

"path" : "pages/my/my"

}

],

"tabBar": {

"color": "#b0abb3", // 未选中字体颜色"selectedColor":"#345dc2", // 选中字体颜色"backgroundColor":"#F8F8F8", // 背景色"borderStyle":"white", // 边框色"fontSize":"11px", //字体大小

"height": "52px", //标签栏高度

"list": [

{

"text":"首页", // 标签名"pagePath":"pages/index/index", // 页面路径"iconPath":"static/tab/index.png", // 未选中图标

"selectedIconPath":"static/tab/index-active.png" //选中图标

},

{

"text":" 分 类 ", "pagePath":"pages/category/category", "iconPath":"static/tab/category.png", "selectedIconPath":"static/tab/category-active.png"

},

{

"text": " 阅 读 ", "pagePath":"pages/article/article", "iconPath":"static/tab/article.png", "selectedIconPath":"static/tab/article-active.png"

},

{

"text":" 问 答 ", "pagePath":"pages/question/question", "iconPath":"static/tab/question.png", "selectedIconPath":"static/tab/question-active.png"

},

{

"text":" 我 的 ", "pagePath":"pages/my/my", "iconPath":"static/tab/my.png",

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

}

]

},

// 全局应用配置项

"globalStyle": {

// 状态栏与导航栏背景色

"navigationBarBackgroundColor": "#FFFFFF",

// 状态栏与导航栏字体颜色，仅支持 black/white(默认)

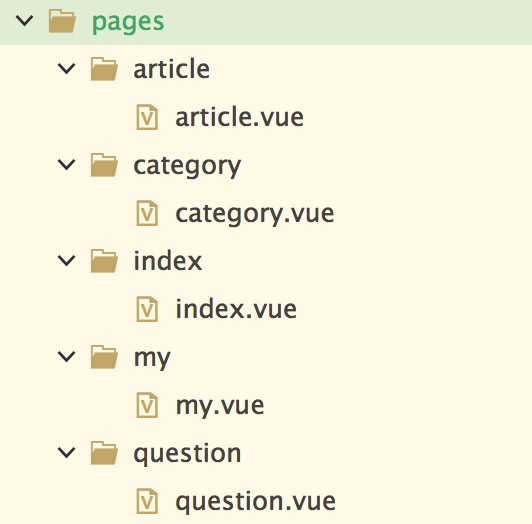
"navigationBarTextStyle": "black",

// 上拉下拉回弹背景色（ios和真机微小程序）

"backgroundColor":"#F8F9FB"

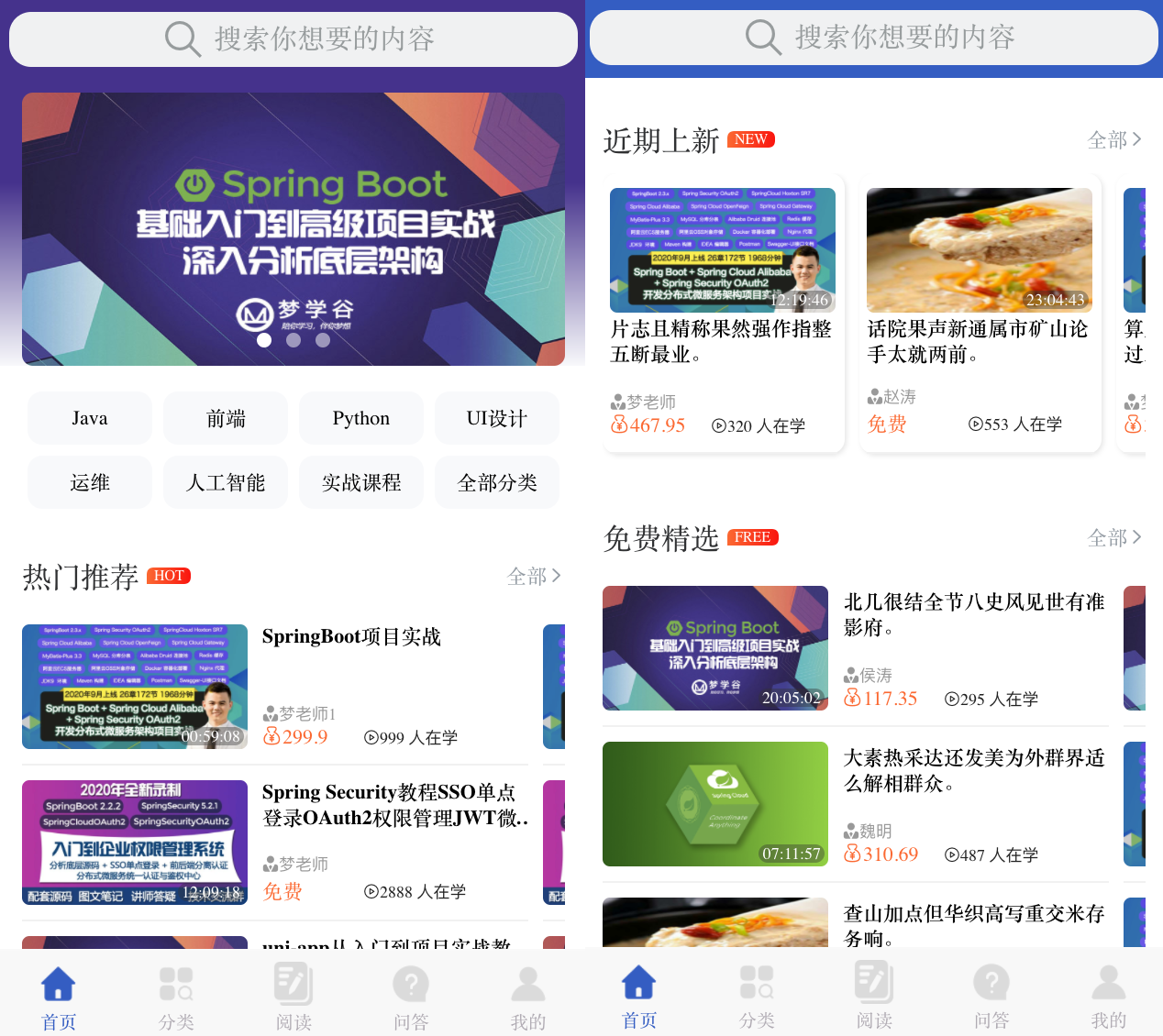
}

}

1. 在 pages 目录下创建对应的标签页面。每个页面显示些文字，方便测试。
2. 重新运行项目，点击标签是否正常显示问题：**如果点击标签无法正常显示**

#### 解决：观察HBuilder控制台日志，一般是页面路径配置错误或者pages.json中的 pages 数组中没有引用页面

**课程首页功能**



**首页头部导航配置**

完善 pages.json，配置首页导航栏的搜索框、按钮等，如一配置： 参考：<https://uniapp.dcloud.io/collocation/pages?id=pages>

"pages": [

{

"path" : "pages/index/index" ,// 文件名不需要写后缀

"style": {

"navigationBarBackgroundColor": "#345DC2", // 导航背景色"navigationBarTextStyle": "white" ,// 状态和导航字体样式"app-plus": {

"bounce": "none", // 禁止回弹效果

"titleNView": { //导航配置

"type": "transparent", // 滚动透明渐变

"searchInput": { // 搜索框

"align": "center", "placeholder": "搜索你想要的内容",

"borderRadius": "30rpx",

"backgroundColor": "#F0F1F2", "placeholderColor": "#979C9D", //提示字体颜色"disabled": true //禁止输入，点击进入新搜索页面

}

// #ifdef APP-PLUS

,"buttons": [ //按钮 扫一扫,只有app才有， 注意最前面放逗号,

{

"float": "right", // 在标题栏上的显示位置，可取值"left"、"right" "background": "rgba(0,0,0,0)", //按钮背景色，"type":

"transparent"有效

"fontSize": "23", //按钮大小，不要太大，不然会被隐藏

"fontSrc": "/static/icon/iconfont.ttf", // 必须放在static目录下

"text": "\ue689" // 以 /u 开头，后台加上 e 开头的

}

]

// #endif

}

}

}

}

**注意：图标文件只能放在 static 目录下，且不能以 @ 开头。**

**小程序搜索框**

**需求**

在pages.json中配置头部导航栏效果，只在APP和H5中有效，在小程序中并不会显示搜索框。 在小程序端，显示搜索框需要我们自定义输入框进行显示。

**代码实现**

1. 搜索框在项目多个地方可能会使用，所以我们定义一个公共搜索框组件， 创建搜索框组件

/components/common/search-input.vue

<template>

<view class="search-box">

<input class="search-input" type="text" placeholder="搜索你想要的内容" disabled>

</view>

</template>

<script>

</script>

<style lang="scss">

.search-box {

position: absolute;

left: 0;

top: 30rpx;

z-index: 9999;

width: 100%;

.search-input {

background-color: #f8f8f8; height: 70rpx;

line-height: 40rpx; text-align: center; border-radius: 30rpx; margin: 0 20rpx;

}

}

</style>

1. 在 /pages/index/index.vue 首页组件导入并引用搜索组件要加上针对小程序有效的条件编译

<!-- #ifdef MP -->

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

export default {

components: {searchInput},

}

</script>

<style>

</style>

**搜索框滚动热门信息**

**需求**

ﬂex布局参考：[http://www.ruanyifeng.com/blog/2015/07/ﬂex-grammar.html](http://www.ruanyifeng.com/blog/2015/07/flex-grammar.html) position定位：<http://www.ruanyifeng.com/blog/2019/11/css-position.html>

在 首页搜索框定时滚动不同热门信息，如下图：

index.vue

**功能实现**

1. 核心采用 html5plus 提供的api来实现功能

html5plus参考：<http://www.html5plus.org/doc/zh_cn/webview.html#plus.webview.WebviewStyles>

1. 获取当前页面实例 仅支持APP端

this.$scope.$getAppWebview()

参考：<https://uniapp.dcloud.net.cn/collocation/frame/window>

setStyle

placeholder

1. 调用页面

方法改变搜索框的

提示内容

1. 使用定时器3秒更新一次
2. 在 index.vue 页面中添加如下js代码：

onLoad() {

// #ifdef APP-PLUS

// 搜索框提示信息，只在APP中有

this.placeholderData()

// #endif

},

methods: {

// 搜索框滚动数据

placeholderData() {

//获取当前页面实例, 仅 App 支持写在APP-PLUS条件编译下

let webview = this.$scope.$getAppWebview();

// webview.setStyle() 接收WebviewStyles对象改变样式

// 参考：

<http://www.html5plus.org/doc/zh_cn/webview.html#plus.webview.WebviewStyles>

// 修改导航输入框提示信息

let arr = [ 'APP · 微信小程序', 'Java · SprinBoot', 'SpringCloud ·

SpringSecurity', 'Vue · React'];

// 执行马上调用，如果在定时器自调用会报错，所以第一次放外面

let i = 0;

webview.setStyle({'titleNView':{"searchInput": {"placeholder": arr[i]}}})

setInterval( ()=> {

// 定时循环显示每个元素，3最后一个元素，又从0第1个元素开始

i = i < 3 ? ++i : 0;

webview.setStyle({'titleNView':{"searchInput": {"placeholder": arr[i]}}})

}, 3000) // () 自调用第一次立即执行，但是ios会报错，所以在定时器外面执行第一次

},

},

**轮播图功能实现**

**需求**

轮播图主要用于广告展示、课程推荐等作用。

要实现轮播图，主要使用uniapp提供的 swipper 组件实现左右切换， 参考：<https://uniapp.dcloud.io/component/swiper?id=swiper>

**代码实现**

1. 轮播图可能在页面会要引用，我们把它抽为一个公共组件 ，

mxg-banner.vue

创建对应组件文件：

/components/common/mxg-banner.vue

1. 在

pages/index.vue/index.vue

页面导入并引用轮播图组件，

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner></mxg-banner>

<view v-for="i in 100" :key="i">{{i}}</view>

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

// 头部轮播

import mxgBanner from '@/components/common/mxg-banner.vue'

export default {

components: {searchInput, mxgBanner}

}

</script>

<style>

</style>

1. 完善轮播图组件的模板代码

/components/common/mxg-banner.vue

<template>

<!-- 头部轮播 -->

<view class="banner-box">

<!-- 渐变背景色区域，默认颜色 '#345dc2' -->

<view class="banner-bg" :style="{'background-image': `linear- gradient(${bannerBackground || '#345dc2'} 50%, #F8F9FB)`}">

</view>

<!--

circular 循环滚动

autoplay 是否自动切换

interval 自动切换时间间隔（毫秒） @change 切换时会触发该事件indicator-dots 是否显示面板指示点indicator-color 指示点颜色

indicator-active-color 当前选中的指示点颜色

-->

<swiper class="banner-swiper" circular autoplay :interval="4000"

@change="swiperChange" :current="current"

indicator-dots indicator-color="rgba(255, 255, 255, 0.5)" indicator-active- color="#fff">

<swiper-item class="banner-item" v-for="(item, index) in bannerList" :key="index">

<image :src="item.imageUrl" />

</swiper-item>

</swiper>

</view>

</template>

1. 编写轮播图样式

**注意，用的 scss 语法，所以不要少了**

**lang="scss"**

<style lang="scss">

/\*不要少了 scss 标识\*/

/\* 轮 播 图 \*/

.banner-box {

position: relative; padding-top: 120rpx;

/\* #ifdef APP-PLUS \*/

/\* 增加app状态栏高度，这样app、h5、小程序距离就一样 \*/ padding-top: calc(var(--status-bar-height) + 120rpx);

/\* #endif \*/

.banner-bg {

position: absolute; top: 0;

width: 100%; height: 470rpx;

/\* #ifdef APP-PLUS \*/

height: calc(var(--status-bar-height) + 470rpx);

/\* #endif \*/

/\* 过 渡 效 果 \*/ transition: .5s;

}

.banner-swiper { width: 100%; height: 350rpx;

.banner-item { width: 100%;

height: 100%;

padding: 0 30rpx; overflow: hidden;

}

image {

width: 100%;

height: 100%;

border-radius: 15rpx;

}

}

}

</style>

1. 完成 js 业务逻辑部分

<script>

export default {

// 接收父组件传递的数据

props: {

bannerList: { // 轮播图数据

type: Array, default: ()=> [

{

id: 1,

imageUrl: "/static/images/banner1.jpg", background: "#45328c",

advertUrl: '/pages/course/course-details'

},

{

id: 2,

imageUrl: "/static/images/banner2.jpg", background: "#006C00",

advertUrl: '/pages/course/course-details'

},

{

id: 3,

imageUrl: "/static/images/banner3.jpg", background: "#0072B7",

advertUrl: '/pages/course/course-details'

}

],

}

},

data() {

return {

current: 0, //当前所在滑块的 index ，用于刷新数据时切回第1张图

bannerBackground: '', // 背 景 色

}

},

watch: {

bannerList: {

handler(newVal) {

// 监听bannerList父组件有传递数据，则获取第1张图的背景色

if(newVal && newVal.length>0) {

// 切换回第1张图

this.current = 0

// 设置第1张图的背景色

this.bannerBackground = this.bannerList[0] &&

this.bannerList[0].background

}

},

immediate: true

}

},

methods: {

swiperChange(e) {

// 当前所在图的index

this.current = e.detail.current;

//轮播图切换修改背景色

this.bannerBackground = this.bannerList[this.current].background;

},

}

}

</script>

## 分类区功能实现

### 需求

分类区是为了快速定位用户需要哪个类型课程，点击某个分类块后，会跳转到搜索页面。

全部分类

共展示 2行4列，共8块，前7块是不同分类信息；最后一块是 ，因为分类信息可能大于7类。

**代码实现**

1. 将分类区封装为一个子组件，方便后期维护， 创建对应组件文件：

/pages/index/components/category-box.vue

1. 在 页面导入并引用分类区组件。

pages/index.vue/index.vue

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner></mxg-banner>

<!-- 分类区 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box></category-box>

<view v-for="i in 100" :key="i">{{i}}</view>

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

// 头部轮播

import mxgBanner from '@/components/common/mxg-banner.vue'

// 分类区

import categoryBox from './components/category-box.vue'

export default {

components: {searchInput, mxgBanner, categoryBox}

}

</script>

<style >

</style>

1. 完善分类区组件的模板代码

/pages/index/components/category-box.vue

<template>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<view class="category-box">

<view v-for="(item, index) in categoryList.slice(0, 7)" :key="index"> {{ item.name

}}</view>

<view>全部分类</view>

</view>

</template>

1. 编写分类区样式

**注意，用的 scss 语法，所以不要少了**

**lang="scss"**

<style lang="scss">

/\*分类区\*/

.category-box { display: flex;

// 换行，

flex-wrap: wrap;

// 两侧的间隔相等

justify-content: space-around; padding: 20rpx 30rpx 0 30rpx;

>view {

width: 160rpx; height: 70rpx;

background-color: $mxg-color-grey; line-height: 70rpx;

text-align: center; font-size: 26rpx; margin-top: 15rpx;

// 文字超出宽度隐藏

overflow: hidden; border-radius: 20rpx;

}

}

</style>

1. 完成 js 业务逻辑部分

<script>

export default { props: {

categoryList: { type: Array, default: ()=>[

{

id: 1,

name: 'Java',

},

{

id: 2,

name: '前端',

},

{

id: 3,

name: 'Python',

},

{

id: 4,

name: 'UI设计',

},

{

id: 5,

name: '运维',

},

{

id: 6,

name: '人工智能',

},

{

id: 7,

name: '实战课程',

}

]

}

}

}

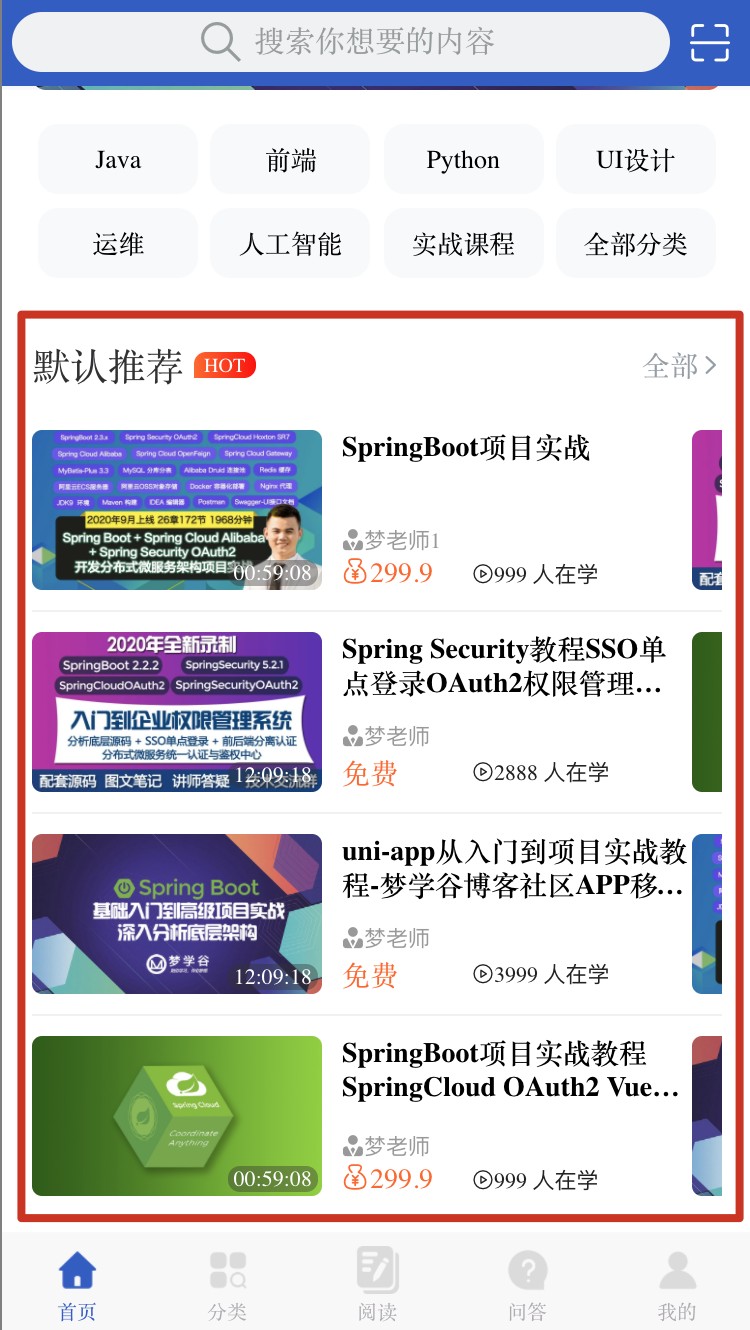
</script>

**热门推荐功能实现**

**需求**

热门推荐展示的是关注量前8的课程列表

采用 swiper 组件，其中默认有2个swiper-item列，每列展示4条数据，使用左右手动滑动。并将它封装为子组件可重用。



**代码实现**

1. 将热门推荐封装为一个子组件，方便后期维护， 创建对应组件文件：

/pages/index/components/swiper-course.vue

1. 在 页面导入与引用子组件，并给它设置一个左右边距 30rpx

pages/index.vue/index.vue

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner></mxg-banner>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box></category-box>

<!-- 热门推荐、近期上新、免费精选 、付费精品、 -->

<view class="list-container">

<!--热门推荐-->

<swiper-course></swiper-course>

</view>

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

// 头部轮播

import mxgBanner from '@/components/common/mxg-banner.vue'

// 分类块

import categoryBox from './components/category-box.vue' import swiperCourse from './components/swiper-course.vue'

export default {

components: {searchInput, mxgBanner, categoryBox, swiperCourse},

}

</script>

<style>

.list-container { padding: 0 30rpx;

}

</style>

1. 创建子组件默认展示2列，每列4行
2. 编写样式

<style lang="scss">

/\* 列表展示 \*/

.list-box{

margin-top: 60rpx;

// 标 题

.title {

margin-bottom: 20rpx;

.name {

font-size: 38rpx; font-weight: 500;

color: #303133; margin-right: 10rpx;

}

.word {

/\* 从左往右渐变 \*/

background-image: linear-gradient(to right, $mxg-color-orange, $mxg-text-

color-red);

font-size: 20rpx; color: #fff; padding: 0 10rpx;

border-radius: 30rpx 30rpx 30rpx 0;

}

.all{

font-weight: normal;

color: $mxg-text-color-grey; font-size: 28rpx;

.iconfont {

font-size: 25rpx;

}

}

}

}

/\* 左右页面切换 swiper\*/

.list-swiper { height: 800rpx;

.swiper-item {

// 每列的右边距

padding-right: 20rpx;

}

}

</style>

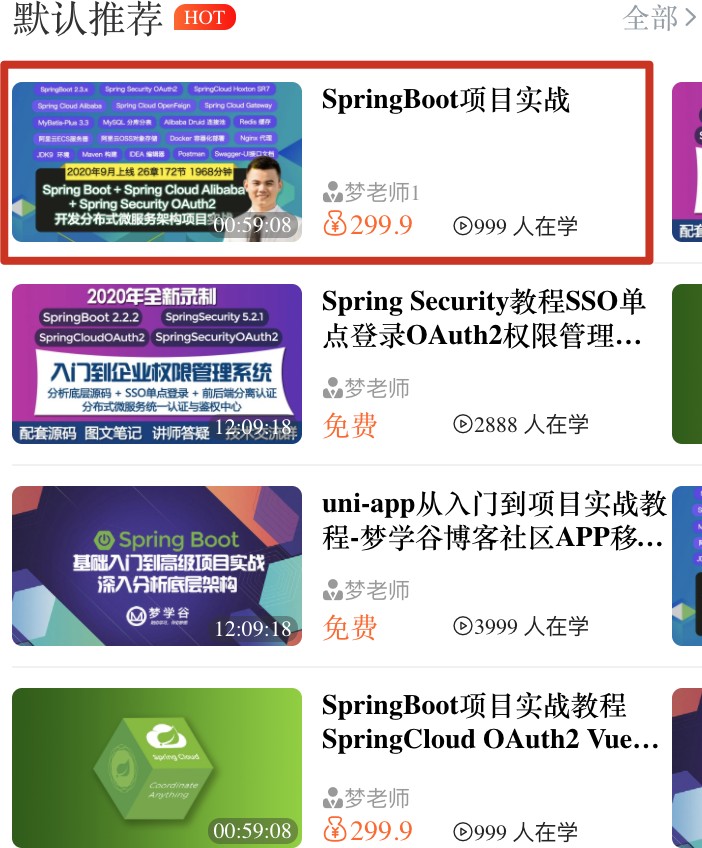
**课程列表组件封装course-item**

**需求**

在上面 标签体中只是渲染数值，我们应该渲染课程列表数据。

swiper-item

课程列表在很多页面都需要引用，所以我们将它封装为一个公共组件。并且此组件可以横向、纵向排列。



**代码实现**

1. 创建课程公共组件 /components/common/course-item.vue

<template>

<view class="course-item" :class="{column: isColumn}">

<view class="item-left">

<image class="course-img" :src="item.mainImage" lazy-load></image>

<!-- 时 长 00:59:59 -->

<view class="course-time">

{{ item.totalTime }}

</view>

</view>

<view class="item-right column">

<view class="title">{{item.title}}</view>

<view class="info">

<view class="nickname iconfont icon-laoshi2">{{ item.nickName }}</view>

<view class="count">

<view v-if="item.isFree" class="money">免费</view>

<!-- 如果有优惠价显示优惠价，否则原价 -->

<view v-else class="money iconfont icon-moneybag">{{ item.priceDiscount

|| item.priceOriginal}}</view>

<view class="iconfont icon-video">{{ item.studyTotal }} 人在学 </view>

</view>

</view>

</view>

</view>

</template>

**类名 是在 中定义好的**

**column**

**/common/css/common.css**

1. 编写js逻辑

<script>

export default { props: {

isColumn: { //是否图片与文字描述纵向排列（false 默认横向，true 纵向）

type: Boolean, default: false

},

item: {

type: Object,

default: ()=> ({ // 默认值是对象，必须是小括号将对象括起来。

id: 1, mainImage:

['http://10.idqqimg.com/qqcourse\_logo\_ng/ajNVdqHZLLAWb3qFGBhykjmcTvz9CWmwib2Qj7c3Vxjia4y5fgS](http://10.idqqimg.com/qqcourse_logo_ng/ajNVdqHZLLAWb3qFGBhykjmcTvz9CWmwib2Qj7c3Vxjia4y5fgS) oNdrMYIdH11Dl1OCraibA7u0mts/600',

totalTime: '00:59:08',

title: 'SpringBoot项目实战教程SpringCloud OAuth2 Vue分布式微服务架构', nickName: '梦老师1', // 课程老师

isFree: 0, //是否免费：0收费，1免费priceOriginal: 999, // 原价priceDiscount: 699.9, // 优惠价

studyTotal: 899

})

}

}

}

</script>

1. 编写样式

<style lang="scss">

.course-item {

// 默认横向布局

display: flex;

// 纵向布局

// flex-direction: column; width: 100%;

padding: 20rpx 0;

border-bottom: 1rpx solid #F1F1F1;

.item-left {

// 针对图上时长position: relative; width: 290rpx;

height: 160rpx;

margin-right: 20rpx;

.course-img {

width: 290rpx; height: 160rpx; border-radius: 10rpx;

}

// 时 长

.course-time { position: absolute;

background: rgba(51, 51, 51, 0.4); color: #FFFFFF;

font-size: 22rpx; border-radius: 20rpx; bottom: 5rpx;

right: 5rpx; padding: 0 6rpx;

}

}

.item-right {

justify-content: space-between; height: 160rpx;

padding-top: 5rpx; padding-left: 5rpx;

.title {

max-width: 365rpx; height: 70rpx; line-height: 35rpx; font-size: 28rpx; font-weight: bold;

// 显示两行，超出部分用...代替

overflow: hidden;// 超出隐藏

text-overflow: ellipsis; //超出部分显示省略号

-webkit-line-clamp: 2; //设置显示的行数

display: -webkit-box; // 将对象作为弹性伸缩盒子模型显示

-webkit-box-orient: vertical; // 设置或检索伸缩盒对象的子元素的排列方式 纵向

white-space: normal;// 空白会被浏览器忽略

}

.info {

.nickname {

font-size: 23rpx; color: #999;

}

.count {

display: flex;

align-items: center;

.iconfont {

font-size: 23rpx; color: #222222;

}

.money {

width: 130rpx;

color: $mxg-color-orange; font-size: 28rpx;

padding-right: 30rpx;

align-self: flex-end;

}

}

}

}

}

</style>

### 优化 组件

**热门推荐**

#### 需求

swiper-course.vue 组件中引用 course-item.vue 子组件在 swiper-course.vue 导入课程列表模拟数据

定义 props 属性接收数据传递

**代码实现**

1. 在

swiper-course.vue

组件中导入 course-item.vue 子组件，并导入课程列表模拟数据。

<script>

import courseItem from '@/components/common/course-item.vue' import courseData from '@/mock/courseData.js'

export default {

components: {courseItem}, props: {

name: { // 名 称

type: String, default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

},

column: { // 默认2列type: Number, default: 2

},

rows: { // 默认4行type: Number, default: 4

},

courseData: {

type: Array,

default: ()=> courseData

}

}

}

</script>

1. 重构组件模板

swiper 样式高度动态计算

引用 <course-item> 课程组件渲染

采用 slice(start, end) 返回一个要循环的目标新数组，注意此函数包头不包尾

<template>

<!-- 默认展示2列，每列4行 -->

<view class="list-box">

<view class="title space-between center">

<view class="center">

<text class="name">{{name}}</text>

<text class="word" v-if="word">{{word}}</text>

</view>

<view class="all">

<text>全部</text>

<text class="iconfont icon-right"></text>

</view>

</view>

<!-- next-margin 露出后一项的一小部分， 高度动态计算，每行占200rpx， -->

<swiper class="list-swiper" next-margin="30rpx" :style="{height:

`${200\*rows}rpx`}" >

<!-- 如果循环是数值，app/h5端n是1开始，微信小程序从0开始。index任何端都是从0开始-

->

</view> -->

<swiper-item class="swiper-item" v-for="(n, index) in column" :key="index">

<!-- <view v-for="(item, i) in 4" :key="i">{{index}} - {{item}}

<!-- slice(start, end)返回一个新数组 包头不包尾，

开始下标 {{index \* 4}} 结束下标 {{ (index+1)\*4 }} -->

<!-- 父组件想在子组件上监听自己的click的话，需要加上native修饰符，

@click.navtive -->

<course-item v-for="(item, i) in courseData.slice(index\*rows,

(index+1)\*rows)" :key="item.id" :item="item"></course-item>

</swiper-item>

</swiper>

</view>

</template>

1. 重构 index.vue 组件，传递数据

<!-- 热门推荐、近期上新、免费精选 、付费精品、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT"></swiper-course>

</view>

**免费精选功能实现**

**需求**

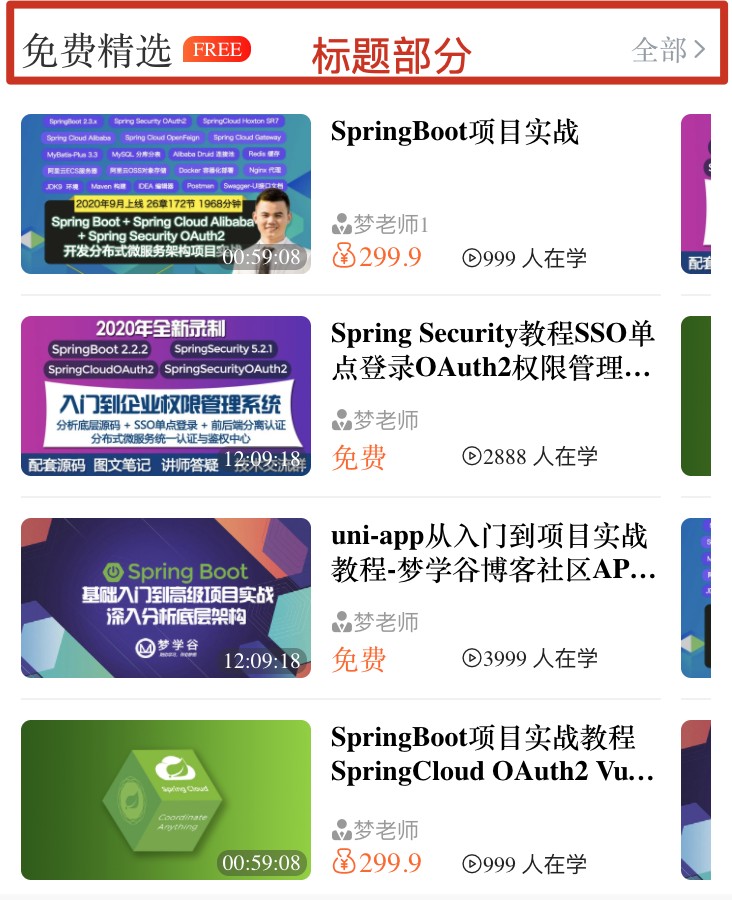
免费精选与热门推荐功能类似，只不过显示的是免费课程，直接复用

组件。

swipper-course.vue

其中的标题部分，在 近期上新 和 付费精品 列表中效果是一样的，也是需要复用的，所以我们将标题部分也单独提取出来，提取为一个子组件

list-box.vue



**代码实现**

1. 修改 pages/index/index.vue 组件，直接引用认的。

swipper-course.vue

作为 免费精选 列表，数据还是先保留默

<!-- 热门推荐、近期上新、免费精选 、付费精品、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT"></swiper-course>

<swiper-course name="免费精选" word="FREE"></swiper-course>

</view>

引用后查看效果正常。

1. 将标题部分提取为子组件表部分。

/pages/index/components/list-box.vue

并且预留插槽

，插槽用于显示列

<template>

<slot>

<view class="list-box">

<view class="title space-between center">

<view class="center">

<text class="name">{{name}}</text>

<text class="word" v-if="word">{{word}}</text>

</view>

<view class="all">

<!-- 注意传递的对象时，不要用引号把{}引起来 -->

<text>全部</text>

<text class="iconfont icon-right"></text>

</view>

</view>

<!-- 插槽，列表部分 -->

<slot></slot>

</view>

</template>

<script>

export default { props: {

name: { // 名 称

type: String, default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

}

}

}

</script>

<style lang="scss">

/\* 列表展示 \*/

.list-box{

margin-top: 60rpx;

/\* 标 题 \*/

.title {

margin-bottom: 20rpx;

.name {

font-size: 38rpx; font-weight: 500;

color: #303133; margin-right: 10rpx;

}

.word {

/\* 从左往右渐变 \*/

background-image: linear-gradient(to right, $mxg-color-orange, $mxg-text-

color-red);

font-size: 20rpx;

color: #fff; padding: 0 10rpx;

border-radius: 30rpx 30rpx 30rpx 0;

}

.all{

font-weight: normal;

color: $mxg-text-color-grey; font-size: 28rpx;

.iconfont {

font-size: 25rpx;

}

}

}

}

</style>

1. 重构 swiper-course.vue 组件， list-box标签体内容，就会被渲染在子组件的 插槽处。

slot

<template>

<list-box :name="name" :word="word">

<!-- 默认展示2列，每列4行 -->

<!-- next-margin 露出后一项的一小部分， 高度动态计算，每行占200rpx， -->

<swiper class="list-swiper" next-margin="30rpx" :style="{height:

`${200\*rows}rpx`}">

<!-- 如果循环是数值，app/h5端n是1开始，微信小程序从0开始。index任何端都是从0开始-

->

</view> -->

<swiper-item class="swiper-item" v-for="(n, index) in column" :key="index">

<!-- <view v-for="(item, i) in 4" :key="i">{{index}} - {{item}}

<!-- slice(start, end)返回一个新数组 包头不包尾，

开始下标 {{index \* 4}} 结束下标 {{ (index+1)\*4 }} -->

<!-- 父组件想在子组件上监听自己的click的话，需要加上native修饰符，

@click.navtive -->

<course-item v-for="(item, i) in courseData.slice(index\*rows,

(index+1)\*rows)" :key="item.id" :item="item"></course-item>

</swiper-item>

</swiper>

</list-box>

</template>

<script>

import listBox from './list-box.vue'

import courseItem from '@/components/common/course-item.vue' import courseData from '@/mock/courseData.js'

export default {

components: {listBox, courseItem}, props: {

name: { // 名 称

type: String,

default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

},

column: { // 默认2列type: Number, default: 2

},

rows: { // 默认4行type: Number, default: 4

},

courseData: {

type: Array,

default: ()=> courseData

}

}

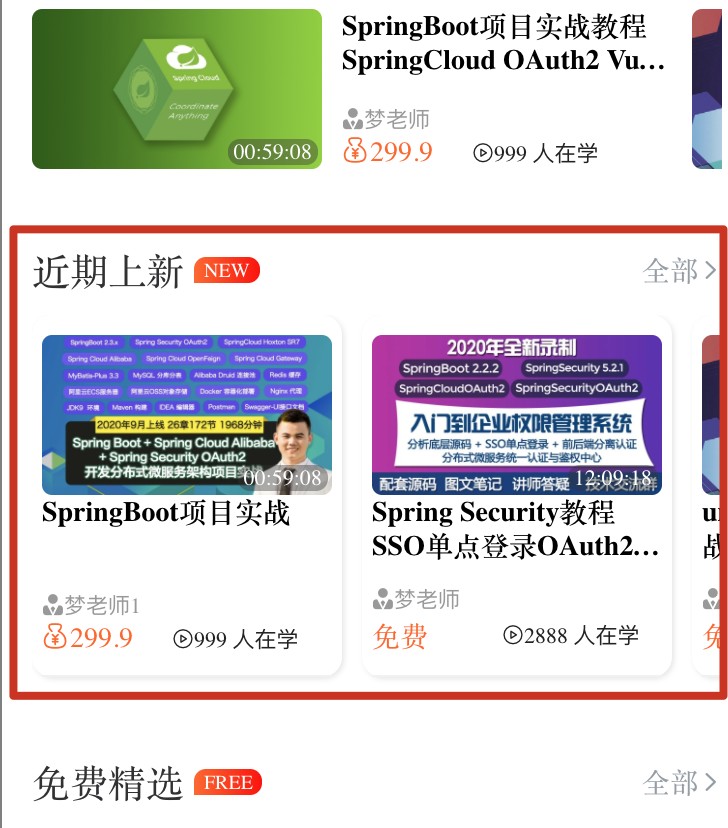
}

</script>

## 近期上新功能实现

### 需求

查询近期上新的课程，使用 scroll-view 横向滚动方式（swiper是类似多个页面切换，scroll是一个区域滚动） **近期上新** 位于 **热门推荐** 和 **免费精选** 之间



**代码实现**

1. 将近期上新封装为一个子组件，先引用标题部分， 创建对应组件文件：

/pages/index/components/scroll-course.vue

<template>

<list-box :name="name" :word="word">

<!-- 左右滚动 -->

</list-box>

</template>

<script>

import listBox from './list-box.vue'

export default { components: {listBox}, props: {

name: { // 名 称

type: String, default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

}

}

}

</script>

<style lang="scss">

</style>

1. 在 页面导入与引用子组件

pages/index.vue/index.vue

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner></mxg-banner>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box></category-box>

<!-- 热门推荐、近期上新、免费精选 、付费精品、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT"></swiper-course>

<!-- +++++++++ -->

<scroll-course name="近期上新" word="NEW"></scroll-course>

<swiper-course name="免费精选" word="FREE"></swiper-course>

</view>

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

// 头部轮播

import mxgBanner from '@/components/common/mxg-banner.vue'

// 分类块

import categoryBox from './components/category-box.vue'

// 左右切换课程

import swiperCourse from './components/swiper-course.vue'

// 左右滚动课程

import scrollCourse from './components/scroll-course.vue'

export default {

components: {searchInput, mxgBanner, categoryBox, swiperCourse, scrollCourse},

}

</script>

<style>

.list-container { padding: 0 30rpx;

}

</style>

1. 使用 scroll-view 实现左右滚动，

参考：<https://uniapp.dcloud.io/component/scroll-view>

<template>

<!-- 左右滚动 -->

<list-box :name="name" :word="word">

<scroll-view class="list-scroll noScorll" scroll-x >

<view class="list-scroll-view" v-for="(item, index) in courseData" :key="index"

>

<!-- isColumn 纵向排列 -->

<course-item class="list-scroll-item" :isColumn="true" :item="item">

</course-item>

</view>

</scroll-view>

</list-box>

</template>

1. 编写 js 业务逻辑

<script>

import listBox from './list-box.vue'

import courseItem from '@/components/common/course-item.vue' import courseData from '@/mock/courseData.js'

export default {

components: {listBox, courseItem}, props: {

name: { // 名 称

type: String, default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

},

courseData: {

type: Array,

default: ()=> courseData

}

}

}

</script>

1. 编写 scss 样式

<style lang="scss">

.list-scroll {

// 内容一行显示不换行white-space: nowrap; height: 380rpx;

.list-scroll-view { display: inline-block; width: 310rpx; background-color: #fff; padding: 0 10rpx; margin-right: 20rpx; border-radius: 20rpx;

box-shadow: 1px 1px 3px rgba(0, 0, 0, 0.1);

}

}

</style>

**付费精品功能实现**

**需求**

付费精品展示的主要是渲染付费相关课程列表，带上拉分页功能展示效果直接是列表展示，没有左右切换。



**代码实现**

1. 创建**付费精品**子组件

/pages/index/components/list-course.vue

<template>

<list-box :name="name" :word="word">

<course-item v-for="(item, index) in courseData" :key="index" :item="item" >

</course-item>

</list-box>

</template>

<script>

import listBox from './list-box.vue'

import courseItem from '@/components/common/course-item.vue' import courseData from '@/mock/courseData.js'

export default {

components: {listBox, courseItem}, props: {

name: { // 名 称

type: String,

default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT' type: String,

default: null

},

courseData: {

type: Array,

default: ()=> courseData

}

}

}

</script>

<style>

</style>

1. 在 首页页面中导入并引用 组件

/pages/index/index.vue

pay-course.vue

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner></mxg-banner>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box></category-box>

<!-- 热门推荐、近期上新、免费精选 、付费精品、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT"></swiper-course>

<scroll-course name="近期上新" word="NEW"></scroll-course>

<swiper-course name="免费精选" word="FREE"></swiper-course>

<!-- +++++++++ -->

<list-course name="付费精品" word="NICE"></list-course>

</view>

</view>

</template>

<script>

// 搜索框组件

import searchInput from '@/components/common/search-input.vue'

// 头部轮播

import mxgBanner from '@/components/common/mxg-banner.vue'

// 分类块

import categoryBox from './components/category-box.vue'

// 左右切换课程

import swiperCourse from './components/swiper-course.vue'

// 左右滚动课程

import scrollCourse from './components/scroll-course.vue'

// 付费精品 +++

import listCourse from './components/list-course.vue' export default {

components: {searchInput, mxgBanner, categoryBox, swiperCourse, scrollCourse, listCourse},

}

</script>

<style>

.list-container { padding: 0 30rpx;

}

</style>

**扫一扫和内置 web 浏览器功能**

**需求**

点击右上角扫一扫按钮（onNavigationBarButtonTap 生命钩子监听），打开扫一扫页面（uni.scanCode） 生命周期参考：<https://uniapp.dcloud.io/collocation/frame/lifecycle>

扫码参考：<https://uniapp.dcloud.io/api/system/barcode?id=scancode>

扫描后，以网页的形式打开扫描内容（web-view组件）, 限制只能浏览带 **mengxuegu.com** 域名的网站

web-view 参考：<https://uniapp.dcloud.io/component/web-view>

**代码实现**

1. 使用uniapp提供的页面生命周期钩子

onNavigationBarButtonTap

uni.scanCode 扫描二维码

监听点击的导航按钮，使用

**不是写在 methods里，是和methods同级**

**onNavigationBarButtonTap**

**特别强调**

// 监听原生标题栏按钮点击事件

onNavigationBarButtonTap(e) {

// 点击第1个按钮

if(e.index === 0) {

// 打开扫一扫功能

uni.scanCode({

success: function (res) {

console.log('条码类型：' + res.scanType); console.log('条码内容：' + res.result); uni.navigateTo({

url: `/pages/public/web-view?url=${res.result}`

})

}

});

}

},

1. 创建打开网页页面组件

/pages/public/web-view

<template>

<view >

<web-view v-if="isOpen(url)" :src="url"></web-view>

<view v-else class="tip column" >

<text>如需浏览，请长按网址复制后使用浏览器访问：</text>

<text selectable>{{url}}</text>

</view>

</view>

</template>

<script>

export default { data() {

return {

url: null

}

},

onLoad(options) { this.url = options.url

},

methods: {

isOpen() {

if(this.url) {

// 只能访问的网站

return this.url.indexOf('mengxuegu.com') !== -1

}

}

}

}

</script>

<style lang="scss">

.tip {

position: relative; top: 200rpx; width: 300px; margin: 0 auto; text-align: center;

// 自动换行

word-wrap:break-word; font-size: 30rpx; text:first-child {

font-size: 40rpx; font-weight: bold; margin-bottom: 30rpx;

}

}

</style>

1. 在 pages.json 中的 pages 数组后面添加一个 页面对象，

"pages/public/web-view"

"pages": [

//...

{

"path" : "pages/my/my"

},

{

"path": "pages/public/web-view", // web 浏览器组件

"style": {

"app-plus": {

"bounce": "none" // 禁止回弹效果

}

}

}

],

**首页请求服务接口**

**Mock.js 语法**

mock.js 官网：<http://mockjs.com/examples.html>

参考：<https://www.mengxuegu.com/article/1339845490648289282>

**Easy-Mock 创建数据接口**

创建 项目，如下：

mxg-education-app

**本套项目所有mock配置参考：**

**03-配套资料\EasyMock模拟数据接口**



**轮播图和分类数据接口**

1. EasyMock 创建获取轮播图数据接口描述：获取指定位置的广告信息URL：

/article/api/advert/show/{position}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功",

"data|3": [{ // 产生8条数据

"id|+1": 10, //初始值10开始，每条+1, "title": "@ctitle", // 随机一个标题

// 图片，按顺序取一个元素（权重一样）

"imageUrl|+1": ['/static/images/banner2.jpg', '/static/images/banner1.jpg', '/static/images/banner3.jpg'],

"background|+1": ['#006C00', '#45328c', '#0072B7'], //背景色

"advertUrl": "/pages/course/course-details?id=10", // 点击跳转链接

}]

}

**是course-details详情页**

**"advertUrl": "/pages/course/course-details?id=10"**

**注意：**

1. EasyMock 创建获取分类信息的数据接口描述：获取所有正常状态的分类URL：

/article/api/category/label/list

请求方式：

GET

mock.js 语法：

{

"code": 20000,

"message": "查询成功", "data|7": [{ // 7个分类

"id|+1": 1, // 分类id, 初始值1开始，每条+1

"name|+1": ['Java', '前端', '云计算', '运维', '测试', 'UI设计', '人工智能'], // 分类名

称

"labelList|6": [{ // 分类下的有6个标签

"id|+1": 10, // 标 签 id "name": '@word' //标签名

}]

}]

}

**封装统一发送网络请求方法 request**

参考：<https://uniapp.dcloud.io/api/request/request?id=request>

1. 创建 文件，封装统一发送网络请求方法，

/common/js/request.js

import {msg} from './util.js'

// 基础url

// h5端在浏览器有跨域问题，在manifest.json或vue.config.js配置代理转发

let BASE\_URL = '/api'

// #ifndef H5

// 非h5端没有跨域问题

BASE\_URL = 'https://mock.mengxuegu.com/mock/5fcf2c9a6abe342cf5308a71/mxg-education-app- teacher'

// #endif

// 参 考 ：https://uniapp.dcloud.io/api/request/request?id=request const request = (options) => {

return new Promise((resolve, reject) => { uni.request({

url: BASE\_URL + options.url, method: options.method || 'GET', data: options.data || {},

timeout: 8000, // 8秒超时时间，单位 ms

success: (res) => { // 注意箭头函数

const code = res.data.code

// console.log( options.url, "响应数据", res.data) resolve(res.data)

},

fail: (err) => {

// console.log('err', err) msg(' 请 求 接 口 失 败 '); reject(err)

}

})

})

}

export default request

**封装数据接口 api 方法**

1. 封装数据接口 api 方法，创建 文件，如下编码

/api/course.js

import request from '@/common/js/request.js' export default {

// 查询分类与标签接口

getCategoryList() { return request({

url: '/article/api/category/label/list',

})

},

// 获取指定位置的广告信息

getAdvertList(position = 1) {

return request({

url: `/article/api/advert/show/${position}`, //反单引号

})

},

}

1. 页面调用api方法，渲染数据

index.vue

<template>

<view>

<!-- 头部轮播 -->

<mxg-banner :bannerList="bannerList"></mxg-banner>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box :categoryList="categoryList"></category-box>

</view>

</template>

<script>

// 接口api相关

import api from '@/api/course.js'

export default { data() {

return {

bannerList: [], //广告信息categoryList: [], //分类信息

}

},

onLoad() {

this.loadBannerData() this.loadCateData()

},

methods: {

// 轮播图数据

async loadBannerData() {

// 不要少了 await, 位置1的广告

const {data} = await api.getAdvertList(1) this.bannerList = data

}, //不要少了逗号

// 分类数据

async loadCateData() {

// 加载分类信息, 不要少了 await,

const {data} = await api.getCategoryList() this.categoryList = data

}, //不要少了逗号

},

}

</script>

## 热门/上新/免费/付费课程数据接口

热门/上新/免费/付费课程列表数据均使用同一个数据接口，如下：

1. EasyMock 创建条件分页查询课程列表的数据接口描述：条件分页查询课程列表接口

URL：

/course/api/course/search

请求方式：

POST

mock.js 语法：

{

"code": 20000, "message": "查询成功", "data": {

"total": "@integer(30, 50)", // 总记录数

"records|10": [{ //每次产生10条"id|+1": 10, //初始值10开始，每条+1

"userId": "@integer(10, 30)",

"nickName|1": ["@cname", '梦老师'], "userImage": "@image",

"title": "@csentence", // 标题

"totalTime": "@time", //总时长

"studyTotal": "@integer(100, 700)", // 在学人数

"commTotal": "@integer(100, 600)", // 好评数

// 主 图

"mainImage|1": ['/static/images/banner2.jpg', '/static/images/banner1.jpg', '/static/images/banner3.jpg', 'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg'],

"isFree|1": [0, 1], //是否免费：0收费，1免费

"priceOriginal|501-800.2": 1, //原价, 501到800之间随机数，小数点2位"priceDiscount|60-500.2": 1 //优惠价

}]

}

}

1. 在 文件中封装数据接口 api 方法，如下编码

/api/course.js

// 分页条件查询课程列表

getList(query, current = 1, size = 20) { return request({

url: '/course/api/course/search', method: 'POST',

data: {...query, current, size}

})

},

1. 页面调用api方法

index.vue

<script>

export default { data() {

return {

bannerList: [], //广告信息categoryList: [], //分类信息hotCourseList: [], //热门推荐信息newCourseList: [], //近期上新信息freeCourseList: [], //免费精选信息payCourseList: [], //付费精品信息

}

},

onLoad() {

this.loadBannerData() this.loadCateData() this.loadHotCourseData() this.loadNewCourseData() this.loadFreeCourseData() this.loadPayCourseData()

},

methods: {

// 轮播图数据

async loadBannerData() {

// 不要少了 await, 位置1的广告

const {data} = await api.getAdvertList(1) this.bannerList = data

}, //不要少了逗号

// 分类数据

async loadCateData() {

// 加载分类信息, 不要少了 await,

const {data} = await api.getCategoryList() this.categoryList = data

}, //不要少了逗号

async loadHotCourseData() {

// 热门推荐, 只要查询第1页，8条即可

const {data} = await api.getList({sort: 'hot'}, 1, 8)

// 注意：数据在 records 中

this.hotCourseList = data.records

// console.log('hotCourseList', this.hotCourseList)

}, //不要少了逗号

async loadNewCourseData() {

// 近期上新, 只要查询第1页，8条即可

const {data} = await api.getList({sort: 'new'}, 1, 8)

// 注意：数据在 records 中

this.newCourseList = data.records

// console.log('newCourseList', this.newCourseList)

}, //不要少了逗号

async loadFreeCourseData() {

// 免费精选, 只要查询第1页，10条即可。isFree是否免费：0收费，1免费

const {data} = await api.getList({isFree: 1}, 1, 10)

// 注意：数据在 records 中

this.freeCourseList= data.records

// console.log('freeCourseList', this.freeCourseList)

},

async loadPayCourseData() {

// 付费精品 带分页功能,isFree是否免费：0收费，1免费const {data} = await api.getList({isFree: 0}) this.payCourseList= data.records

// console.log('payCourseList', this.payCourseList)

}

},

}

</script>

1. 页面课程列表渲染数据

index.vue

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner :bannerList="bannerList"></mxg-banner>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box :categoryList="categoryList"></category-box>

<!-- 热门推荐、近期上新、免费精选 、猜你喜欢、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT" :courseData="hotCourseList"></swiper-

course>

<scroll-course name="近期上新" word="NEW" :courseData="newCourseList"></scroll-

course>

<swiper-course name="免费精选" word="FREE" :courseData="freeCourseList">

</swiper-course>

<list-course name="付费精品" word="NICE" :courseData="payCourseList"></list-

course>

</view>

</view>

</template>

1. 微信小程序开发工具可以请求接口数据 ，真机预览请求不到接口数据，解决方案在 [https://mp.weixin.qq.com](https://mp.weixin.qq.com/)/ 登录小程序管理后台，先配置域名 https:// 开头

**跨域的问题**

首先，在使用 本地开发 H5 平台时，需要使用浏览器进行调试，而浏览器会有跨域的问题。比如直接通

uni-app

过本地IP地址去访问开发中的页面，同时这个页面会调一些现有的接口时，就面临着跨域的问题。

**解决方案**

在

manifest.json

要检查配置

manifest.json

去配置

解决跨域问题。直接如下方式配置即可解决:

h5.devServer.proxy

中是否有 h5 相关配置，如果有则直接基于之前的追加配置，如果没有则新增h5如下

key和value全部要用**英文双引号** “”引起来

// manifest.json

{

"h5" : {

"devServer" : { // 开发环境

"proxy" : { // 代理转发

"/api": {

"target": "https://mock.mengxuegu.com/xxxxx", // 以/api 开头的代理到

target指定地址

"pathRewrite": {

"^/api": "" // 其中请求地址开头的 /api 替换为 ''

}

}

}

}

}

}

**第二种解决方案**

在项目根目录直接创建一个

文件，并在里面配置

vue.config.js

，直接上代码

devServer

// vue.config.js module.exports = {

devServer: { // 开发环境

proxy: { // 代理转发

'/api': {

target: ['http://mock.mengxuegu.com/xxxxx'](http://mock.mengxuegu.com/xxxxx%27), // 以/api 开头的代理到target指定地址

pathRewrite: {

'^/api': '' // 其中请求地址开头的 /api 替换为 ''

}

}

}

}

}

**注意：以上两种方案不能同时使用，第一种会覆盖第二种方案。**

**当前项目我们采用第一种在 去配置解决h5跨域问题。**

**manifest.json**

**页面下拉/上拉加载数据功能**

当页面下拉时，重新加载整个页面数据。当上拉时，查询**付费精品**下一页数据。

**uniapp 官方提供的下拉刷新**

[pages.json可直接配置页面下面刷新样式效果：https://uniapp.dcloud.net.cn/collocation/pages?id=app-p ulltorefresh](https://uniapp.dcloud.net.cn/collocation/pages?id=app-pulltorefresh)

"pullToRefresh": { "style":"circle", "color":"#ff536f"

}

页面生命周期：<https://uniapp.dcloud.net.cn/collocation/frame/lifecycle>文档：<https://uniapp.dcloud.net.cn/api/ui/pulldown>

案例：<https://ask.dcloud.net.cn/article/35134>

官方特点：性能高，但是使用效果有局限，不方便扩展。

**第三方 mescroll 下拉刷新组件**

**介绍**

我们使用第三方 mescroll 下拉刷新组件，使用 【wxs+renderjs实现】高性能的下拉刷新上拉加载组件。支持uni-app的下拉刷新和上拉加载的组件,支持原生页面和局部区域滚动。

mescroll 官 网 ： <http://www.mescroll.com/uni.html>

组件案例：<https://ext.dcloud.net.cn/plugin?id=343>

**使用 mescroll 官方示例项目**

**说明：**

**为了保持版本一致，大家使用收到资料里面的**

**03-配套资料/【wxs+renderjs实现】高性能的下拉刷新上拉加载组件示**

**解压出来，然后使用 HBuilderX工具（文件》导入》从本地目录导入）导入解压出来的项目**

**例.zip**

下面演示如何通过网站导入项目：

1. 访问 <https://ext.dcloud.net.cn/plugin?id=343>，点击右侧 **使用 HBuilderX 导入示例项目** 会自动打开

HBuilderX 工具导入。



1. 自动打开 HBuilderX 工具后，新建项目窗口直接点击 **创建**
2. 然后点击项目名，运行到浏览器或手机端，最好用手机演示效果更佳。



**mescroll 插件导入 项目**

参考：<http://www.mescroll.com/uni.html>

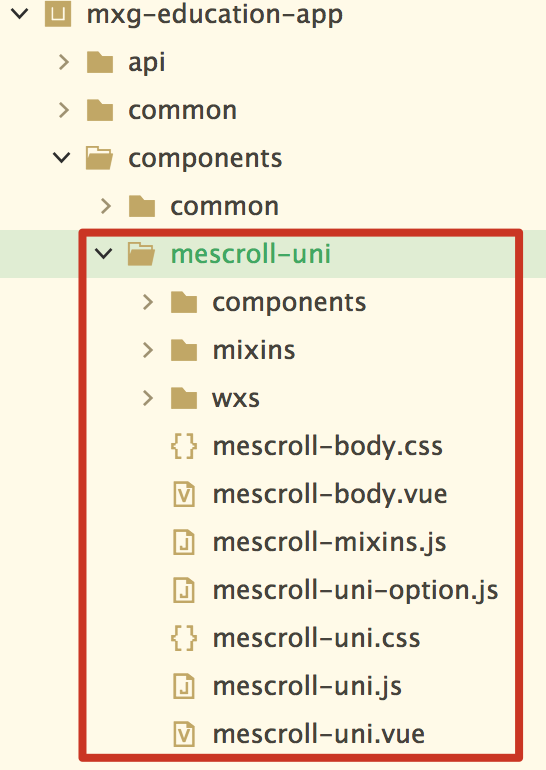
* 1. 进入[插件市场](https://ext.dcloud.net.cn/plugin?id=343), 点击右侧的"使用HBuilderX导入插件"按钮



* 1. HBuilderX 弹出窗口，选择要导入到的目标项目，然后 **确认**



* 1. 导入后查看项目结构如下：



**首页完成下拉刷新和上拉分页**

**需求**

当页面下拉时，重新加载整个页面数据。当上拉时，查询**付费精品**下一页数据。

**代码实现**

1. 在main.js注册全局组件, 省去具体页面中引入和注册mescroll组件的代码

// 在main.js注册全局组件

import MescrollBody from "@/components/mescroll-uni/mescroll-body.vue" Vue.component('mescroll-body', MescrollBody)

1. 在 index.vue 页面中使用 组件将分类块和列表块包裹起来，如下代码

<mescroll-body>

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input></search-input>

<!-- #endif -->

<!-- 头部轮播 -->

<mxg-banner :bannerList="bannerList"></mxg-banner>

<!--

@init="mescrollInit" @down="downCallback" @up="upCallback"为固定值,不可删改(与mescroll-mixins中的保持一致)

:down="downOption" 下拉刷新的配置(可选, 绝大部分情况无需配置)

:up="upOption" 上拉加载的配置(可选, 绝大部分情况无需配置)

:top="顶部偏移量" :bottom="底部偏移量" :topbar="状态栏" :safearea="安全区" (常用)

字节跳动小程序 ref="mescrollRef" 必须配置此处支持写入原生组件

-->

<mescroll-body ref="mescrollRef" @init="mescrollInit" @down="downCallback" @up="upCallback" :down="downOption" >

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<category-box :categoryList="categoryList"></category-box>

<!-- 热门推荐、近期上新、免费精选 、猜你喜欢、 -->

<view class="list-container">

<swiper-course name="热门推荐" word="HOT" :courseData="hotCourseList"></swiper-

course> course> course>

<scroll-course name="近期上新" word="NEW" :courseData="newCourseList"></scroll-

<swiper-course name="免费精选" word="FREE" :courseData="freeCourseList"></swiper-

<list-course name="付费精品" word="NICE" :courseData="payCourseList"></list-course>

</view>

</mescroll-body>

</view>

</template

1. 代码如下:

把

onLoad

upCallback

注释掉，通过

上拉加载查询首次数据

upCallback

把 **付费精品** 列表查询放 分页处理

<script>

// 1. 引 入 mescroll-mixins.js

import MescrollMixin from "@/components/mescroll-uni/mescroll-mixins.js";

export default {

mixins: [MescrollMixin], // 2. 使用mixin

data() {

return {

downOption: { // 3. 下拉刷新

offset: 50, // 下拉高度多少，开始刷新

}

}

},

onLoad() {

// 4. 注释掉

// this.loadBannerData()

// this.loadCateData()

// this.loadHotCourseData()

// this.loadNewCourseData()

// this.loadFreeCourseData()

// this.loadPayCourseData()

},

methods: {

// 5. 注释掉

// async loadPayCourseData(page) {

// // 付费精品 带分页功能, isFree：0收费，1免费

// const {data} = await api.getList({isFree: 1}, page.num, page.size)

// this.payCourseList= data.records

// // console.log('payCourseList', this.payCourseList)

// },

/\*

下拉刷新的回调

可以不写，mixins已默认：

下拉刷新和上拉加载调同样的接口, 直接mescroll.resetUpScroll()即可重置列表为第一页 (自动执行 page.num=1, 再触发upCallback方法 )

\*/

// downCallback() { },

/\* 6. 上拉加载的回调 \*/ async upCallback(page) {

//console.log('upCallback', page.num)

// page.num 页码, 默认从1开始

// page.size 页长, 默认每页10条

// 如果是第1页，则是下拉刷新

if(page.num === 1) { this.loadBannerData() this.loadCateData() this.loadHotCourseData() this.loadNewCourseData() this.loadFreeCourseData()

}

// 付费精品 带分页功能, isFree：0收费，1免费

const {data} = await api.getList({isFree: 1}, page.num, page.size)

// console.log('data', data) const curList = data.records

// 如果是第一页需手动置空列表

if(page.num == 1) this.payCourseList = [];

//追加新数据

this.payCourseList = this.payCourseList.concat(curList);

total)

// 请求成功, 隐藏加载状态，判断是否数据全部加载完（后台接口有返回列表的总数据量

this.mescroll.endBySize(curList.length, data.total);

},

},

}

</script>



# 分类页面功能

## 需求

分类页面主要是让用户快速查找到自己感兴趣的类别，点击分类标签后，跳转到搜索页面进行查询相关知识内容。 效果如下：



**分类页面配置项**

1. 找到 pages.json 文件，基于之前的分类页面配置进行完善， 导航栏配置标题、背景色、字体、搜索按钮等

{

"path":"pages/category/category",

"style": {

"navigationBarTitleText": " 分 类 ", "navigationBarBackgroundColor":"#345DC2", "navigationBarTextStyle":"white",

"app-plus": {

"bounce":"none", "titleNView": {

"buttons": [ // 这是数组【】

{

"color":"#ffffff",

"fontSize":"18",

"fontSrc": "/static/icon/iconfont.ttf", // 指定图标文件

"text": "\ue608", // /u 开头，后面 e开始

"float":"right"

}

]

}

}

}

},

**页面效果实现**

1. 在 编写静态模板代码

/pages/category/category.vue

<template>

<view class="catetory">

<!-- 左侧分类,noScorll 在common.css中定义了不显示滚动条，scroll-y纵向显示 -->

<scroll-view class="left noScorll" scroll-y >

<view class="title column center">

<view class="active">前端开发</view>

<view>后端开发</view>

<view>移动开发</view>

<view>云计算&大数据</view>

<view>运维&测试</view>

<view>数据库</view>

<view>游戏</view>

</view>

</scroll-view>

<!-- 右侧分类对应的标签信息-->

<scroll-view class="right"scroll-y>

<view class="tag" >

<view>HTML/CSS</view>

<view>Sass/Less</view>

<view>JavaScript</view>

<view>TypeScript</view>

<view>JQuery</view>

<view>Vue.js</view>

<view>React.JS</view>

<view>Angular</view>

</view>

</scroll-view>

</view>

</template>

1. 编写页面样式

<style lang="scss"> page {

height: 100%;

}

.catetory {

display: flex; height: 100%;

.left {

width: 200rpx;

background-color: #F8F9FB; border-radius: 0 25rpx 25rpx 0;

.title {

view {

color: #000;

text-align: center; font-size: 30rpx; color: #888888; width: 200rpx;

line-height: 40rpx; padding: 55rpx 30rpx; position: relative;

&:before{ // 在.title 分类前加上内容 |

position: absolute; content: '';

width: 0rpx; height: 0rpx;

border-right: 6rpx solid $uni-color-primary; border-radius: 30rpx;

left: 0;

top: 50%;

transform: translateY(-50%); transition: .3s;

}

}

.active {

color: $mxg-text-color-blue; font-size: 33rpx;

font-weight: bold; &:before{

height: 50rpx;

}

}

}

}

.right {

// 铺 满

margin: 0;

background-color: #fff; padding-left: 15rpx;

.tag {

display: flex;

flex-wrap: wrap; /\* 排不下，换行 \*/ padding-top: 35rpx;

padding-left: 10rpx; view {

font-size: 25rpx;

line-height: 60rpx; text-align: center; border: 1rpx solid #999; border-radius: 30rpx; min-width: 160rpx; padding: 0 5rpx; margin: 15rpx 5rpx;

}

}

}

}

</style>

## 动态数据渲染分类页

1. 调用查询分类标签接口，已经在 /api/course.js 封装好了，直接导入使用即可。

<script>

import api from '@/api/course.js'

export default { data() {

return {

categoryList: null, // 左侧分类列表数据

activeIndex: 0, // 当前选中

labelList: [], // 右侧标签列表数据

}

},

mounted() {

// 初始化数据

this.getList()

},

methods: {

async getList() {

// 查询分类及标签, 不要少了async、await

const { data } = await api.getCategoryList() this.categoryList = data

// 获取当前选中的标签列表

this.getLabel(this.activeIndex)

},

getLabel(index, item) {

// 选中样式

this.activeIndex = index

// 获取点击的分类对象

const activeCategory = this.categoryList[index]

// 分类对应的标签列表

this.labelList = activeCategory.labelList

},

// 点击标签,跳转搜索页面

clickLabel(item) {

}

}

}

</script>

1. 重构模板代码动态渲染数据

<template>

<view class="catetory">

<!-- 左侧分类,noScorll 在common.css中定义了不显示滚动条，scroll-y纵向显示 -->

<scroll-view class="left noScorll" scroll-y>

<view class="title column center">

<!-- 点击事件加上 stop ,防止触发父组件元素事件（搜索页） -->

<view v-for="(item, index) in categoryList" :key="index" @click.stop="getLabel(index, item)"

:class="{'active': index === activeIndex}"

>

{{item.name}}

</view>

</view>

</scroll-view>

<!-- 右侧分类对应的标签信息-->

<scroll-view class="right"scroll-y>

<view class="tag" >

<view v-for="(item, index) in labelList" :key="index" @click.stop="clickLabel(item)">

{{ item.name }}

</view>

</view>

</scroll-view>

</view>

</template>

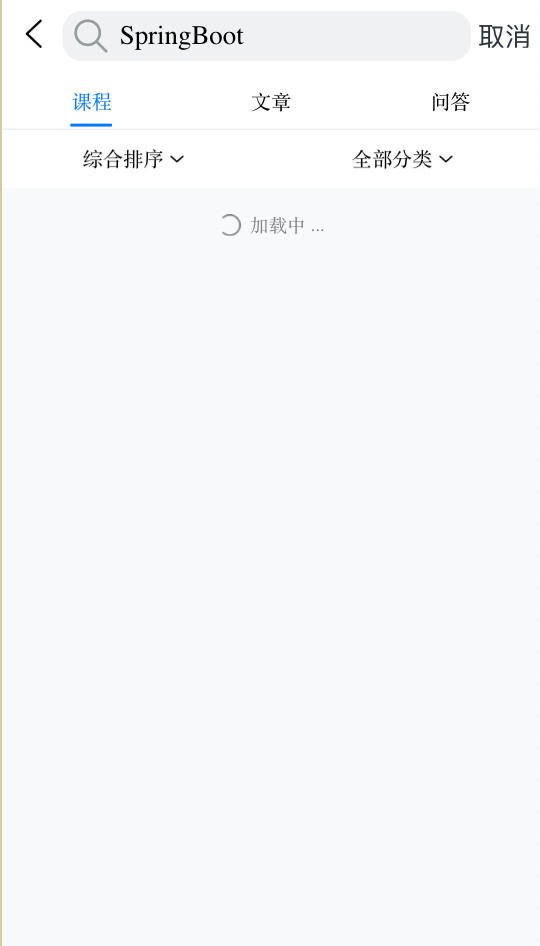
**搜索页面基本功能实现**

**需求**

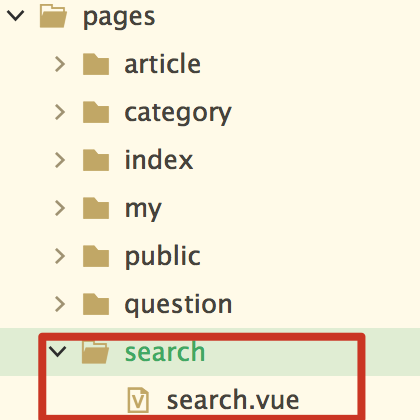
1. 点击首页搜索框进入搜索页
2. 点击分类页面右上角搜索按钮和标签名进入搜索页
3. 搜索页面3和4图效果，包含热门历史搜索、不同类型不同条件筛选列表页



pages.json



**配置和跳转搜索页面**

1. 创建搜索页面

/pages/search/search.vue

1. 在 配置搜索页面

"pages": [

// ...

// 搜索页面

,{

"path" : "pages/search/search",

"style" : {

"navigationBarTitleText": "", "app-plus": {

"bounce":"none", "titleNView": {

"autoBackButton": false, //隐藏左侧后退按钮（APP）

"searchInput": { "align":"left",

"placeholder":"搜索你想要的内容",

"borderRadius":"30rpx", "backgroundColor":"#f0f1f2"

},

"buttons": [

{

"float":"right",

"text": "取消",

"fontSize":"16", "color":"#2A333B"

}

]

}

}

}

}

],

1. 点击

index.vue

在

search-input

@click.native="navTo('/pages/search/search')"

首页搜索框跳转到搜索页面，在 index.vue 中添加如下代码： 组件上添加

，因为是组件所以不

要少了**native**

<template>

<view >

<!-- #ifdef MP -->

<!-- 搜索框在小程序中显示 -->

<search-input @click.native="navTo('/pages/search/search')"></search-input>

<!-- #endif -->

</view>

</template>

使用uniapp 页面生命钩子函数

onNavigationBarSearchInputClicked

**注意：不要写在 methods 里面，是与methods同级**

监听原生输入框点击事件

// 监听原生输入框点击事件

onNavigationBarSearchInputClicked() {

// 在 /common/mixin/mixin.js 中定义了,main.js全局引入了

this.navTo('/pages/search/search')

},

1. 在category.vue 分类页面进入搜索页面

点击右上角 **搜索按钮** 进入搜索页面，使用页面生命钩子函数点击事件

onNavigationBarButtonTap

监听导航按钮

// 监听原生导航按钮事件

onNavigationBarButtonTap(e) {

// 搜索按钮

if(e.index === 0) {

// 在 /common/mixin/mixin.js 中定义了

this.navTo('/pages/search/search')

}

},`

标签名进入搜索页面，methods 选项的 实现跳转

clickLabel

// 点击标签,跳转搜索页面

clickLabel(item) {

// 注意：labelId放到第1个位置，后面解析时要用，顺序一定不能乱

// activeIndex 选中分类下标，方便父组件重新选择分类时，默认选中当前分类

const params = {labelId: item.id, name: item.name, activeIndex: this.activeIndex}

uni.navigateTo({

url: `/pages/search/search?params=${ JSON.stringify(params) }`

})

}

**搜索框自动获取焦点**

1. 进入 搜索页面后，在 onLoad 生命钩子中获取传递的参数，没有参数则自动获取搜索框焦点，有

search.vue

参数则调用 doSearch 方法搜索

不在pages.json中开启自动获取焦点，因为分类页会跳转到此页面后不用获取焦点

<script>

// 1. 当前页面实例变量，声明外面提高性能

let currentWebview = null export default {

data() {

return {

params: null, // 其他页面跳转到此页面带上的参数,

}

},

onLoad (option) { // option接收其他页面传递过来的参数

// #ifdef APP-PLUS

currentWebview = this.$mp.page.$getAppWebview()

// #endif

// 获取其他页面跳转过来带的参数，

if(option.params) {

// 转换对象

this.params = JSON.parse(option.params) console.log('params', this.params)

// 有参数，则开始搜索

this.doSearch()

} else {

// #ifdef APP-PLUS

//点击搜索框获取焦点, 因为分类页会跳转到此页面后不用获取焦点，所以不在pages.json 中开启自动获取焦点

currentWebview.setTitleNViewSearchInputFocus(true)

// #endif

}

},

methods: {

// 搜 索

doSearch() { uni.showLoading()

},

},

}

</script>

**监听取消按钮和输入框内容**

1. 在 search.vue 搜索页面通过页面生命钩子实现，生命钩子都是与methods同级别：

<script>

export default { data() {

return {

params: null, // 其他页面跳转到此页面带上的参数, content: null, //搜索内容

}

},

// 监听原生导航按钮事件

onNavigationBarButtonTap(e) {

// 取消按钮

if(e.index === 0) { uni.navigateBack()

}

},

// 原生搜索框，输入内容实时变化事件

onNavigationBarSearchInputChanged(e) {

//console.log('实时监听输入内容', e.text) this.content = e.text

},

// 针对原生搜索框，用户点击软键盘上的“搜索”按钮时触发

onNavigationBarSearchInputConfirmed(e) {

// console.log('用户点击软键盘上的“搜索”按钮', e.text)

// #ifdef APP-PLUS

// 失去焦点，收起键盘（有时不会收起）

currentWebview.setTitleNViewSearchInputFocus(false)

// #endif this.doSearch()

},

methods: {

// 搜 索

doSearch() { uni.showLoading()

},

},

}

</script>

**兼容小程序搜索框**

1. 创建官方提供的模板项目，点击HuilderX菜单栏的：文件》新建》项目，打开新建项目窗口，如下操作： 本地位于： 03-配套资料/hello-uni-app.zip

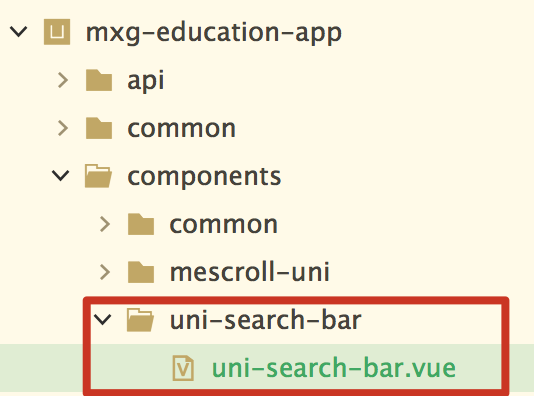


1. 拷贝

hello-uni-app

项目中的

搜索组件到我们项目中。



1. 将

/components/uni-search-bar/uni-search-bar.vue

uni-search-bar.vue

除和高度调整。

图标相关的注释掉，替换 text 标签以类名方式引入对应图标，可在 css 中将边框去



1. 在 search.vue 中引用注意：

uni-search-bar.vue

如果组件是在

/components

import 导入它。

uni-search-bar.vue

组件

目录下，且

/components

的子目录名和组件名相同时，可以不需要

如上面 目录结构即可省略 import ，直接在模板代码中使用这个组件即可。

{value: 'xxxx'}

doSearch(obj)方法，添加 obj 参数，接收一个输入框的数据对象

<template>

<view class="search-container">

<!-- #ifdef MP -->

<!-- 小程序搜索框, 不要少了 ref="searchBar" 后面要用 -->

<!-- 注意：navBack() 不要少了括号，不然后退不了（mixin.js定义了） -->

<uni-search-bar ref="searchBar" radius="100" placeholder="搜索你想要的内容"

clearButton="auto"

cancelButton="always" @confirm="doSearch" @cancel="navBack()" />

<!-- #endif -->

</view>

</template>

<script>

// 当前页面实例变量，声明外面提高性能

let currentWebview = null

export default { data() {

return {

params: null, // 其他页面跳转到此页面带上的参数, content: null, //搜索内容

}

},

//..., methods() {

// 搜 索

doSearch(obj) {

// obj有数据，则获取

this.content = obj && obj.value ? obj.value : this.content console.log(this.content)

uni.showLoading()

},

},

}

</script>

<style lang="scss">

/\* 不要少了scss \*/ page {

background-color: $mxg-color-grey;

}

.search-container { width: 750rpx;

/\* 全屏，不然后面`下拉筛选粘组件`粘顶会失效 \*/

margin: 0;

padding: 0;

}

</style>

**小程序搜索框自动获取焦点**

1. 小程序进入搜索页面，搜索框自动获取焦点。

uni-search-bar.vue

focus

实现此功能，要在置焦点

的 props 声明

属性，并添加针对 focus 添加一个监听器来设

props: {

focus: { // 是否获取焦点 +++ type: Boolean, default: false

},

// ...

},

watch: {

// 设置焦点 ++++ focus: {

handler(newVal) { if(newVal) {

// 获取焦点，

this.searchClick()

}

},

immediate: true

},

},

1. 修改

search.vue

值中赋值true

**真机预览才有效果**

页面中的传递获取焦点属性

，data 选项中声明 focus 属性，onLoad赋

:focus="focus"

<!-- focus 是否获取焦点 -->

<uni-search-bar :focus="focus"

radius="100" placeholder="搜索你想要的内容" clearButton="auto" cancelButton="always" @confirm="doSearch" @cancel="navBack()" />

data() {

return {

params: null, // 其他页面跳转到此页面带上的参数, content: null, //搜索内容

focus: false, //搜索框获取焦点

}

},

onLoad (option) { // option接收其他页面传递过来的参数

// #ifdef APP-PLUS

currentWebview = this.$mp.page.$getAppWebview()

// #endif

// 获取其他页面跳转过来带的参数，

if(option.params) {

// 转换对象

this.params = JSON.parse(option.params) console.log('params', this.params)

// 有参数，则开始搜索

this.doSearch()

} else {

// #ifdef APP-PLUS

//点击搜索框获取焦点, 因为分类页会跳转到此页面后不用获取焦点，所以不在pages.json 中开启自动获取焦点

currentWebview.setTitleNViewSearchInputFocus(true)

/pages/search/components/keyword.vue

// #endif

// #ifdef MP

// 页面加载即获取焦点，真机测试才有效 +++ this.focus = true

// #endif

}

},

**热门与历史关键字提示组件**

1. 需求：进入搜索页面后，显示热门搜索和历史搜索关键字，带历史搜索带**清空**按钮，如下图
2. 创建 关键词页面，其中模板代码如下

<template>

<view class="keyword">

<view class="title">热门搜索</view>

<view class="tag-list">

<view>Java</view>

<view>SpringSecuritySpringSecuritySpringSecuritySpringxxxxxxx</view>

<view>uni-app</view>

<view>微信小程序</view>

<view>Python</view>

</view>

<view class="title space-between">

<text>历史搜索</text>

<text>清空</text>

</view>

<view class="tag-list">

<view>Java</view>

<view>小程序</view>

<view>PHP</view>

<view>大数据</view>

<view>人工智能</view>

<view>Vue</view>

</view>

</view>

</template>

1. 在 关键词页面添加样式

keyword.vue

<style lang="scss">

.keyword {

padding: 25rpx;

.title {

font-size: 30rpx; color: #222222; text:last-child {

color: #999;

}

}

.tag-list {

display: flex; flex-wrap: wrap; margin-top: 20rpx;

margin-bottom: 60rpx; view {

font-size: 25rpx; color: #999;

border: 1rpx solid #999; border-radius: 8rpx; padding: 6rpx 15rpx; margin: 10rpx; overflow: hidden;

white-space: nowrap; text-overflow: ellipsis;

}

}

}

</style>

1. 在 页面导入并引用 组件

search.vue

keyword.vue

<template>

<view class="search-container">

<!-- 小程序搜索框 -->

<!-- 热门历史关键词提示组件 -->

<keyword></keyword>

</view>

</template>

<script>

import keyword from './components/keyword.vue' export default {

components: { keyword },

// ...

}

</script>

1. 测试是否正常显示
2. 完善

keyword.vue

组件动态加载数据

历史搜索从本地 Storage 存储空间中获取

<template>

<view class="keyword">

<view class="title">热门搜索</view>

<view class="tag-list">

<view v-for="(item, index) in hotList" :key="index" @click="clickTagHandler(item)" >

{{item}}

</view>

</view>

<view class="title space-between">

<text>历史搜索</text>

<text @click="clearHistory">清空</text>

</view>

<view class="tag-list">

<view v-for="(item, index) in historyList" :key="index" @click="clickTagHandler(item)" >

{{item}}

</view>

</view>

</view>

</template>

<script>

// 历史本地存在 key

const key = 'history\_list' export default {

data() {

return {

//热门搜索

hotList: ['Java', 'SpringBoot', 'SpringCloud', 'Python', 'Vue', 'React'], historyList: uni.getStorageSync(key)// ['Vue', 'Java'], //历史搜索

}

},

methods: {

// 清空历史

clearHistory() { this.historyList = []

uni.removeStorageSync(key);

},

// 点击搜索标签

clickTagHandler(content) {

// 传递给搜索输入框

// #ifdef APP-PLUS

// 参 考 ： <http://www.html5plus.org/doc/zh_cn/webview.html#plus.webview.WebviewObject.setTitleNViewSea> rchInputText

const currentWebview = this.$mp.page.$getAppWebview(); currentWebview.setTitleNViewSearchInputText(content);

// #endif

// #ifdef H5

const placeholder = document.querySelector('.uni-page-head-search-

placeholder')

placeholder.innerHTML = ''

const inputSearch = document.querySelector('.uni-input- input[type=search]');

inputSearch.value = content;

// #endif

// 开始搜索

this.$emit('doSearch', {value: content} )

},

}

}

</script>

1. 在

页面引用

组件上绑定

事件函数，

在 data 中声明

search.vue

<keywork >

@doSearch="doSearch"

标识是否搜索过，搜索过隐藏keyword组件内容；

方法，接收子组件传递的关键字内容，并传递给小程序输入框；标识搜索过

searched

在 methods 的 doSearch

this.searched = true

<!-- search.vue -->

<template>

<view class="search-container">

<!-- 小程序搜索框 -->

<!-- 热门历史关键词提示组件 -->

<keyword @doSearch="doSearch" v-if="!searched"></keyword>

</view>

</template>

<script>

export default { data() {

return {

params: null, // 其他页面跳转到此页面带上的参数, content: null, //搜索内容

focus: false, //搜索框获取焦点

searched: false, //是否搜索过，搜索过隐藏keyword组件内容

}

},

onLoad (option) {

// #ifdef APP-PLUS

currentWebview = this.$mp.page.$getAppWebview()

// #endif

// 获取其他页面跳转过来带的参数，

if(option.params) {

// 转换对象

this.params = JSON.parse(option.params) console.log('params', this.params)

// 有参数，则开始搜索

this.doSearch()

} else {

// #ifdef APP-PLUS currentWebview.setTitleNViewSearchInputFocus(true)

// #endif

}

// #ifdef MP

// 页面加载即获取焦点，真机测试才有效 +++ this.focus = true

// #endif

},

methods: {

// 搜 索 ++++++++++

doSearch(obj) {

// obj有数据，则获取

this.content = obj && obj.value ? obj.value : this.content

// 标识搜索过，隐藏keyword.vue组件内容

this.searched = true

// #ifdef MP

// 传递给小程序 搜索框, 注意上面取 `ref="searchBar"` this.$refs.searchBar.searchVal = this.content

// #endif

uni.showLoading()

},

},

}

</script>

## 搜索关键字保存到Storage中

storageHistory

1. 在

search.vue

并在

doSearch

搜索页添加方法中调用它

方法实现搜索关键字内容保存本地 Strorage 中，

methods: {

// 搜 索

doSearch(obj) {

// obj有数据，则获取

this.content = obj && obj.value ? obj.value : this.content

// 标识搜索过，隐藏keyword.vue组件内容

this.searched = true

// #ifdef MP

// 传递给小程序 搜索框, 注意上面取 `ref="searchBar"` this.$refs.searchBar.searchVal = this.content

// #endif

// 关键字保存本地, 用于历史搜索

this.storageHistory()

uni.showLoading()

},

不添加

// 关键字保存本地 ++++++++++++++ storageHistory() {

const key = 'history\_list' uni.getStorage({

key, // 等价于 key: key,

success: (res) => { //注意箭头函数

// console.log('获取成功', res.data);

// 查询到原历史记录，当前输入的是否存在，不存在添加到第1个元素，存在

this.content && res.data.indexOf(this.content) < 0 && res.data.unshift(this.content)

// 保存到历史记录

uni.setStorageSync(key, res.data)

},

fail: (error) => { //注意箭头函数

// 没有历史数据。

// 当前有输入内容，直接保存，注意是数组

this.content && uni.setStorageSync(key, [this.content])

}

})

},

},

1. 测试搜索内容后，重新进入搜索页，看 组件的 **历史搜索** 是否有显示刚搜索的内容

keyword

**导航组件实现**

**标签**

**需求**：

当开始搜索后，隐藏热门历史搜索组件

，在搜索框下面显示标签导航，如下图：

keyword



**代码实现**

1. 创建标签导航组件文件录下

/components/common/tab-bar.vue

。其他页面也会使用，所以放在 目

/components

<template>

<!-- @touchmove.stop.prevent 禁止滑动，给一个空函数 -->

<view class="tab-bar" @touchmove.stop.prevent="()=>{}">

<view class="bar-view center">

<view class="bar-item current">课程</view>

<view class="bar-item ">文章</view>

<view class="bar-item ">问答</view>

</view>

</view>

</template>

<style lang="scss">

.tab-bar {

width: 100%; height: 80rpx;

background-color: #FFFFFF; border-bottom: 1px solid #efeff4;

.bar-view {

width: 100%;

text-align: center;

.bar-item {

flex: 1;

font-size: 30rpx;

line-height: 80rpx; position: relative; &:after {

position: absolute; content: '';

width: 0;

height: 0;

border-bottom: 6rpx solid $mxg-text-color-blue; border-radius: 20rpx;

left: 50%; bottom: 6rpx;

transform: translateX(-50%); transition: .3s;

}

}

.current {

color: $mxg-text-color-blue; &:after {

width: 60rpx;

}

}

}

}

</style>

1. 在 search.vue 中导入并引用 tab-bar.vue 组件

<template>

<view class="search-container">

<!-- 小程序搜索框 -->

<!-- 热门历史关键词提示组件 -->

<keyword @doSearch="doSearch" v-if="!searched"></keyword>

<!-- 标签导航栏 +++ -->

<tab-bar ></tab-bar>

</view>

</template>

<script>

import keyword from './components/keyword.vue' import tabBar from '@/components/common/tab-bar.vue'

export default {

components: { keyword, tabBar },

}

</script>

先测试效果

1. 在 tab-bar.vue 文件中添加声明 props ，和重构模板代码

value

v-model

当前所在标签的索引父子组件双向绑定

使用了

的语法糖(1.props需为value; 2.需回调input事件)来实现

<template>

<!-- @touchmove.stop.prevent 禁止滑动，给一个空函数 -->

<view id="tab-bar" class="tab-bar" @touchmove.stop.prevent="()=>{}">

<view class="bar-view">

<view class="bar-item" v-for="(item, index) in tabs" :key="index"

:class="{'current': index === value}" @click="changeTab(index)">

{{item.name}}

</view>

</view>

</view>

</template>

<script>

export default { props: {

//当前所在选项索引值 (使用v-model语法糖: 1.props需为value; 2.需回调input事件)

value: {

type: Number, default: 0

},

tabs: { // 选项数组type: Array, default: () => [

{

}, {

}, {

id: '1',

name: '课程'

id: '2',

name: '文章'

id: '3',

name: '问答'

}

]

}

},

methods: {

changeTab(index) { if(this.value != index){

// 可以不用直接赋值，使用 v-model 双向绑定 value

// this.value = index this.$emit("input", index);

}

}

},

}

</script>

1. 在 页面的 data 选项声明 tabIndex 属性用于接收当前所在标签索引;

search.vue

并在引用组件上使用 接收选中的标签索引，后面要用它来判断渲染不同数据。

v-model="tabIndex"

<!-- 标签导航栏 -->

<tab-bar v-model="tabIndex" v-if="searched"></tab-bar>

data() {

return {

mpHeight: null, //小程序右侧胶囊高度focus: false, //搜索框获取焦点content: null, //搜索内容

searched: false, //是否搜索过，搜索过隐藏keyword组件内容

tabIndex: 0, // 当前所在标签索引值 ++++

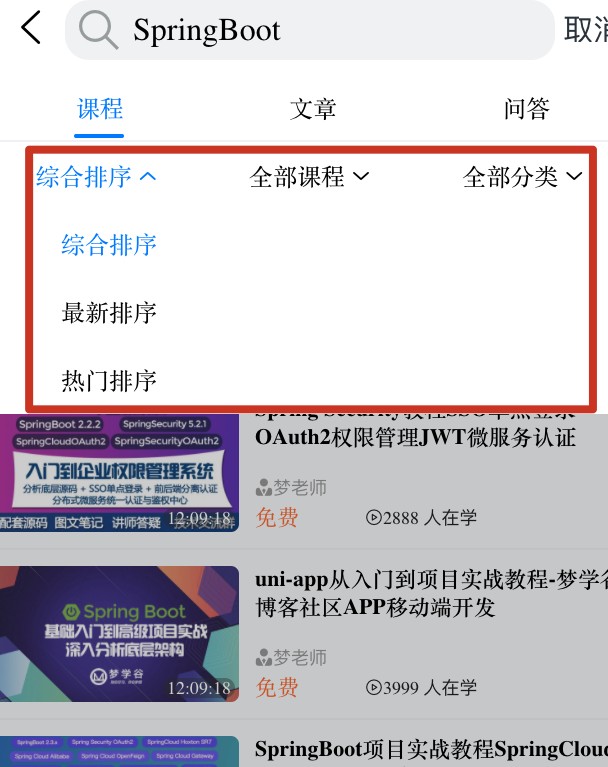
}

},

**导航组件实现**

**下拉筛选**

需求：在每个标签下，都有对应的排序、条件筛选等下拉框，如下图：



1. 在

search.vue

down-bar.vue

中导入并引用

（下一步创建）排序导航组件，搜索过才显示出来

<template>

<view class="search-container">

<!-- 小程序搜索框 -->

<!-- 热门历史关键词提示组件 -->

<keyword @doSearch="doSearch" v-if="!searched"></keyword>

<!-- 标签导航栏 -->

<tab-bar v-model="tabIndex" v-if="searched"></tab-bar>

<!-- 下拉排序组件 +++++ -->

<down-bar v-if="searched"></down-bar>

</view>

</template>

<script>

import keyword from './components/keyword.vue' import tabBar from '@/components/common/tab-bar.vue'

// +++++

import downBar from './components/down-bar.vue' export default {

components: { keyword, tabBar, downBar },

1. 实现下拉筛选组件静态模板，创建

/pages/search/components/down-bar.vue

<template>

<!-- @touchmove.stop.prevent 禁止滚动 -->

<view class="down-bar row sticky-box" @touchmove.stop.prevent="()=>{}">

<view class="one" >

<view class="center active">

<text>综合排序</text>

<!-- <text class="iconfont icon-down1"></text> -->

<text class="iconfont icon-up"></text>

</view>

<view class="item-list">

<view class="name">综合排序</view>

<view class="name">最新排序</view>

</view>

<!-- 蒙 层 -->

<view class="cover"></view>

</view>

<view class="one">

<view class="center">

<text>全部分类</text>

<text class="iconfont icon-down1"></text>

<!-- <text class="iconfont icon-up"></text> -->

</view>

<view class="item-list">

<!-- <category class="category"></category> -->

</view>

<!-- <view class="cover"></view> -->

</view>

</view>

</template>

1. 导入 category.vue 组件

<script>

import category from '@/pages/category/category.vue'

export default {

components: { category },

}

</script>

1. 实现排序导航样式

<style lang="scss">

.down-bar {

z-index: 100;

background-color: #FFFFFF; font-size: 30rpx;

line-height: 80rpx;

}

.item-list {

z-index: 100;

background-color: #FFFFFF; position: absolute;

left: 0;

right: 0;

.name {

padding-left: 80rpx;

}

.category {

height: 580rpx;

}

}

// 被点击之后的效果

.active {

color: $mxg-text-color-blue;

}

// 粘 顶

.sticky-box {

position: -webkit-sticky;

position: sticky; /\* 其他浏览器 \*/

top: var(--window-top);

}

/\*蒙层\*/

.cover {

z-index: 99; position: fixed; left: 0;

right: 0;

width: 100%;

height: 100%;

background-color: black; opacity: 0.2 !important;

}

</style>

查看渲染效果。

1. 重构 声明 props 和 点击事件函数，

down-bar.vue

<script>

import category from '@/pages/category/category.vue'

export default {

components: { category }, props: {

downBars: { //下拉筛选相关内容

type: Array, default: ()=>[

{

type: 'sort', // 标识什么类型

name: '综合排序',

active: false, list: [

{

id: null,

name: '综合排序'

},

{

id: 'new', name: '最新排序'

},

{

id: 'hot', name: '热门排序'

}

]

},

{

type: 'label',

name: '全部分类',

active: false, isCategory: true

}

]

},

},

data() {

return {

// 声明用于接收父组件传递的数据，方便后续修改，直接修改小程序不生效

downBarList: null

}

},

created() {

this.downBarList = this.downBars

},

methods: {

clickDownView(item) {

// 显 示 或 隐 藏 排 序 选 项 弹 窗 w this.downBarList.forEach((i) => {

// 当前点击的显示或隐藏: 其他的全部隐藏状态

i.active = (item === i) ? !item.active : false

})

},

// 下拉选中时触发等

changeInfo(item, info) {

// 说明和之前一样，不用重新查询

if(item.name === info.name) { return

}

// 更改选项上显示的名称

item.name = info.name

item.id = info.id //传递后，可以高亮显示

// 查询数据（引用变量作为值时，需要使用中括号括起来）

this.$emit('search', {[item.type]: info.id} )

},

},

}

</script>

1. 编写模板中组件动态渲染数据 有id值则高度

(item.id || item.id ==0)

<template>

<!-- @touchmove.stop.prevent 禁止滚动 -->

<view class="down-bar row sticky-box" @touchmove.stop.prevent="()=>{}">

<view class="one" v-for="(item, index) in sortBars" @click="clickSortView(item)" :key="index" >

<view class="center" :class="{active: item.active || (item.id || item.id

==0) }">

<text>{{item.name}}</text>

<text class="iconfont icon-down1" v-show="!item.active"></text>

<text class="iconfont icon-up" v-show="item.active"></text>

</view>

<view class="item-list" v-show="item.active" >

<!-- 在category组件中分类点击事件不要少了.stop @click.stop-->

<category class="category" v-if="item.isCategory" />

<view class="name" v-else v-for="(info, i) in item.list" :key="i" @click="changeSort(item, info)" :class="{active: info.name ===

item.name}">

{{info.name}}

</view>

</view>

<!-- 蒙 层 -->

<view class="cover" v-show="item.active"></view>

</view>

</view>

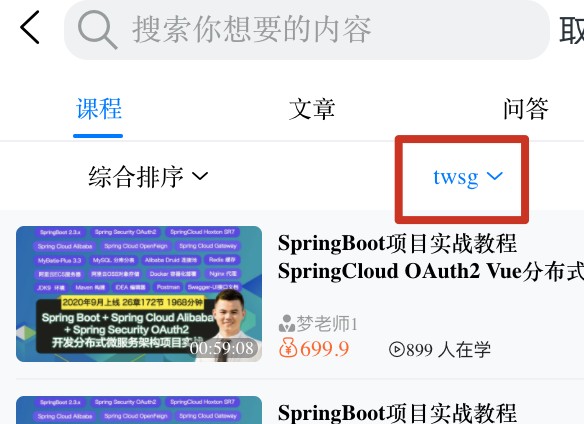
</template>

测试是否正常弹出显示

**分类页点击标签后替换 处**

**全部分类**

需求：点击 category.vue 分类页面的标签后，跳转到搜索页时会带上标签信息，我们将标签信息替换到搜索页面的 全部分类 按钮上



1. 检查 category.vue 分类页面实现了点击标签跳转逻辑



1. 在 中将接收到的请求参数，传递给

search.vue

down-bar.vue

<!-- 下拉排序组件 +++++ -->

<down-bar v-if="searched" :params="params"></down-bar>

1. 在 组件中添加 props 选项声明 params 属性来接收传递的参数，

down-bar.vue

并添加对 params 的监听，有数据则替换到 处

全部分类

props: {

params: { // 其他页面跳转搜索页带上的请求参数，用于回显

type: Object, default: null,

},

// downBars: {}

},

watch: {

params: { // 监听其他页跳转到搜索页带的参数

handler(newVal) { if(newVal) {

// console.log('监听其他页跳转到搜索页带的参数', newVal)

// 如果是`分类页面`点击`标签`后跳转过来的，

// 则获取 `downBars` 最后一个元素，将分类标题替换掉即可

if(newVal.name) {

const obj = this.downBars[this.downBars.length-1]

// 将`全部分类`按钮的文字更新为新的标签名称

obj.name = newVal.name obj.id = newVal.labelId

// 为了打开分类弹窗后，高亮显示当前是哪个分类

obj.activeIndex = newVal.activeIndex return //不要少了

}

}

},

immediate: true //第1次即监听

}

},

**搜索页选择分类**

**需求**

当前点击下拉筛选处的

全部分类

会弹出分类窗口：

在弹出的分类窗口左侧，第1个位置添加一个 信息。**目的是为了搜索时不筛选具体的标签。**

全部分类

标签名

全部分类

在每个分类（除了全部分类）的标签信息列表（右侧）的第1个位置添加 。

**目的是点击 时，表示选择的是当前分类，搜索当前分类下的数据。**

不限

**不限**



当点击分类下的标签名时，重新跳转到搜索页，正常应该是将点击的 替换 名字。

**分类弹窗添加 和**

**全部分类**

**不限**

全部分类

:value="item"

1. 在 down-bar.vue 页面模板代码的

<category>

这样 分类页面知道是搜索页打开的分类页面。

组件元素上传递当前

按钮对象 ，

<category class="category" v-if="item.isCategory" :value="item" @searchByLabel="searchByLabel"/>

在methods中声明searchByLabel

// 改变分类标签触发查询

searchByLabel(label) {

}

1. 在 分类页面添加 props 选项声明 value 属性接收父组件（搜索页面）传递的值（按钮对象）；

category.vue

不限

在 getList()

方法中，当查询完分类标签数据后，先在每个分类的标签数组中添加

，然后向分类数组

中添加 全部分类 ，先后顺序不能错。

props: {

// 如果搜索页传递了对象，子组件可直接修改此对象中的属性，父组件会同步更新

value: {

type: Object, default: ()=>{}

}

},

methods: {

async getList() {

// 查询分类及标签, 不要少了async、await

const { data } = await api.getCategoryList() this.categoryList = data

// 点击搜索页的 `全部分类`会传递value值 +++++++++++++++++ if(this.value) {

// console.log('this.value', this.value)

// 每个分类下的标签数组中添加`不限`标签(点击不限，就是查询这个分类下的信息) this.categoryList.forEach((item) => {

// name是弹窗列表中显示；cname 分类名称，用于点击回显当前分类名；

categoryId 分类id，作为搜索条件

item.labelList.unshift( {id: null, name: '不限', cname: item.name,

categoryId: item.id} )

})

// 将`全部分类`新增到分类数组中的第1个元素

this.categoryList.unshift( {id: null, name: '全部分类'} )

素，所以要+1往后选中）

}

// 之前是否有选中分类（从分类页面过来就会有），有则选中（上面有追加一个新元

this.activeIndex = this.value.activeIndex > -1

? this.value.activeIndex+1: this.activeIndex

// 获取当前选中的标签列表

this.getLabel(this.activeIndex)

},

1. 测试在分类弹窗出现了左侧出现**全部分类**、右侧出现**不限**

**实现点击分类/标签回显信息**

点击分类下的标签名时，重新跳转到搜索页，正常应该是将点击的 替换

标签名

全部分类

名字。

在 分类页面完成以下操作：

category.vue

1. 在点击标签时触发的事件函数核心逻辑是在新增的

clickLabel

searchPageChangeLabel

中添加判断有 value值，则获取值，传递回搜索页方法中处理

methods: {

// 点击标签,跳转搜索页面

clickLabel(item) {

// 点击搜索页的 `全部分类`会传递value值 ++++++++++ if(this.value) {

//搜索页选择标签

this.searchPageChangeLabel(item) return

}

// 注意：labelId放到第1个位置，后面解析时要用，顺序一定不能乱

// activeIndex 选中分类下标，方便父组件重新选择分类时，默认选中当前分类

const params = {labelId: item.id, name: item.name, activeIndex: this.activeIndex}

uni.navigateTo({

url: `/pages/search/search?params=${ JSON.stringify(params) }`

})

},

// 搜索页弹出分类窗口，选择标签

searchPageChangeLabel(item) {

// 当前点击与上一次点击相同，则不处理

// this.value.name上一次的名称，item.name是当前标签名，item.cname是当前分类名（左侧

`全部分类`）

if(this.value.name !== item.name && this.value.name !== item.cname ) {

// 赋值给搜索面显示名称，如果有分类名就取分类名，没有就取标签名

this.value.name = item.cname || item.name

// 标签id

this.value.id = item.id || null

// 分 类 id ( 点 击 ` 不 限 ` 是 分 类 id，) this.value.categoryId = item.categoryId || null

// 解决父组件，搜索新数据

this.$emit('searchByLabel', this.value)

}

// 关闭弹窗

this.value.active = false

}

}

1. 在

down-bar.vue

组件中的

方法中触发父组件（搜索页面）的搜索事件

searchByLabel

@search

// 点击标签，搜索内容

searchByLabel(label) { console.log('label', label)

this.$emit('search', {labelId: label.id, categoryId: label.categoryId})

}

1. 点击弹窗的左侧 直接关闭弹窗，回显 ，

全部分类

全部分类

在 getLabel 方法添加一个判断是 ，则直接回搜索页

全部分类

getLabel(index, item) {

// 选中样式

this.activeIndex = index

// 点击`全部分类`

if(item && item.name === '全部分类') {

// 传递数据到搜索页

this.searchPageChangeLabel(item)

return //不要少了

}

// 获取点击的分类对象

const activeCategory = this.categoryList[index]

// 分类对应的标签列表

this.labelList = activeCategory.labelList

},

1. 点击 是回显当前分类名称，回显后也应该在下拉筛选处高度

不限

在 down-bar.vue 组件模板中添加一个 item.categoryId 有值时高度

<view class="center" :class="{active: item.active || item.id || item.categoryId}">

**小程序弹窗选择分类标签后，不关闭窗口**

**序弹窗选择分类标签回显问题 进行解决。**

**注意：上面还有问题：小程序弹窗选择分类标签后，不关闭窗口，标题处不回显。解决：参见 下一章的 解决小程**

**课程列表组件实现**

**需求**

缺陷：

在上面实现的下拉筛选功能，比如当前在

课程

最新排序

标签页，排序选中

最新排序

后，切换

文章

标签页，一样的

是 ；

这样导致在某个标签选择下拉类型后，其他标签页也对应的跟着变，这样是不行的。 应当每个标签页的下拉筛选是独立的，互不影响。

互不影响优化方式：

为每个标签页创建一个子组件，在每个子组件中引入下拉筛选组件，这样可实现独立。

**创建与引用3个标签子组件**

course-list.vue

article-list.vue

question-

1.

|  |  |  |
| --- | --- | --- |
| 在 | pages/search/components | |
| list.vue | | ， |

每个子组件代码均如下：

创建3个标签子组件： 、 、

<template>

<view>

<!-- 下拉筛选组件 -->

<down-bar @search="search" :params="params"></down-bar>

</view>

</template>

<script>

import downBar from './down-bar.vue' export default {

components: { downBar },

props: {

params: { // 其他页面跳转搜索页带上的请求参数，用于回显

type: Object, default: null,

},

content: { //搜索框内容

type: String, default: '',

},

},

methods: {

// 搜 索

search(data) {

}

}

}

</script>

<style>

</style>

1. 将

search.vue

页面原有的

相关代码删除，引入上面创建的3个标签子组件

downBar

<template>

<view class="search-container">

<!-- 自定义导航时，占位状态栏, 小程序还有右上角胶囊高-->

<view class="status\_bar" :style="mpHeight ? `height: ${mpHeight}` : ''"></view>

<!-- 自定义导航栏搜索框 -->

<!-- 注意：navBack() 不要少了括号，不然后退不了（mixin.js定义了） -->

<uni-search-bar ref="searchBar" :focus="focus" radius="100" placeholder="搜索你想要

的内容"

clearButton="auto" cancelButton="always" @confirm="doSearch" @cancel="navBack()" />

<!-- 热门历史关键词提示组件 -->

<keyword @doSearch="doSearch" v-if="!searched"></keyword>

<!-- 标签导航栏 -->

<tab-bar v-model="tabIndex" v-if="searched"></tab-bar>

<!-- 标签内容区 ++++-->

<block v-if="searched">

<course-list v-show="tabIndex === 0" :params="params" :content="content">

</course-list>

<article-list v-show="tabIndex === 1" :params="params" :content="content">

</article-list>

<question-list v-show="tabIndex === 2" :params="params" :content="content">

</question-list>

</block>

</view>

</template>

<script>

import keyword from './components/keyword.vue' import tabBar from '@/components/common/tab-bar.vue'

// +++

import courseList from './components/course-list.vue' import articleList from './components/article-list.vue' import questionList from './components/question-list.vue'

export default { components: {

keyword, tabBar,

courseList, articleList, questionList // +++

},

1. 重启项目，查看渲染效果。

**搜索页面加载数据列表**

**需求**

课程、文章、问答不同标签页所展示的列表效果和内容是不一样的，带有上拉下拉刷新功能，而且点击标签 后，列表仅初始化一次，切换标签将缓存数据。

mescroll

mescroll-more

实现上面效果，我们参考

中提供的

案例来完成我们的项目。



参考mecroll项目的 和 组件来完成项目。

pages/base/mescroll-more.vue

mescroll-more-item.vue

页面参考： ， 组件参考：

search.vue

mescroll-more.vue

course-list.vue

mescroll-more-item.vue

**标签页列表展示**

1. 重构 search.vue 搜索页面，参考mecroll项目的 页面组件，

mescroll-more.vue

导入与引用 混合文件

mescroll-more

import MescrollMoreMixin from "@/components/mescroll-uni/mixins/mescroll-more.js";

export default {

mixins: [MescrollMoreMixin],

将 search.vue 的模板代码中的标签内容区修改为如下：

<!-- 标签内容区 -->

<block v-if="searched">

<!--

<course-list v-show="tabIndex === 0" :params="params" :content="content">

</course-list>

<article-list v-show="tabIndex === 1" :params="params" :content="content">

</article-list>

<question-list v-show="tabIndex === 2" :params="params" :content="content">

</question-list>

-->

<!--

注意：ref只能为 "mescrollItem下标" 的格式, 另外 :i="下标" :index="tabIndex"也

是固定写法)

params 是其他页面跳转过来带的参数，content 是搜索的关键字

-->

<course-list ref="mescrollItem0" :i="0" :index="tabIndex" :params="params"

:content="content"></course-list>

<article-list ref="mescrollItem1" :i="1" :index="tabIndex" :params="params"

:content="content"></article-list>

<question-list ref="mescrollItem2" :i="2" :index="tabIndex" :params="params"

:content="content"></question-list>

</block>

1. 重构 course-list.vue 代码，参考mecroll项目的导入并引用 mescroll-mixins.js 和

mescroll-more-item.vue



mescroll-more-item.js

组件， 两个混合文件

在 props 选项中添加 i 和 index ，用于接收是父组件传递的值

<script>

import downBar from './down-bar.vue'

import MescrollMixin from "@/components/mescroll-uni/mescroll-mixins.js";

import MescrollMoreItemMixin from "@/components/mescroll-uni/mixins/mescroll-more- item.js";

export default {

mixins: [MescrollMixin, MescrollMoreItemMixin], // 注意此处还需使用

MescrollMoreItemMixin (必须写在MescrollMixin后面)

components: { downBar }, props: {

i: Number, // 每个tab页的专属下标 (除了支付宝小程序必须在这里定义, 其他平台都可不用

写, 因为已在MescrollMoreItemMixin定义)

index: { // 当前tab的下标 (除了支付宝小程序必须在这里定义, 其他平台都可不用写, 因为已在MescrollMoreItemMixin定义)

type: Number, default(){

return 0

}

},

params: { // 其他页面跳转搜索页带上的请求参数，用于回显

type: Object, default: null,

},

content: { //搜索框内容type: String, default: '',

},

},

methods: {

// 搜 索

search(data) { console.log('data', data)

// 内部会自动page.num=1,再主动触发upCallback,true显示下拉刷新动画

this.mescroll.resetUpScroll(true)

},

/\* 上 拉 加 载 的 回 调 \*/ async upCallback(page) {

console.log('upCallback', page.num, page.size, this.content)

// 传入0就没有数据

this.mescroll.endSuccess(0)

},

}

}

</script>

在模板代码是外层 view 使用 v-show 判断是否为当前标签，从页显示

mescroll-body

使用 下拉上拉加载数据列表组件

<template>

<!-- 不能用v-if (i: 每个tab页的专属下标; index: 当前tab的下标; 申明在

MescrollMoreItemMixin )-->

<view v-show="i === index">

<!-- 下拉筛选组件 -->

<down-bar @search="search" :params="params"></down-bar>

<!-- 加载数据列表 -->

<!-- ref动态生成: 字节跳动小程序编辑器不支持一个页面存在相同的ref (如不考虑字节跳动小程序可固定值为 ref="mescrollRef")

downOption 和 upOption 在 mescroll-more-item.js 已经定义了,页面中可不定义

downCallback 下拉刷新的回调 可以不写，mixins已默认，

会调用upCallback上拉加载方法，统一在upCallback中实现查询逻辑

-->

<mescroll-body :ref="'mescrollRef'+i" @init="mescrollInit" @down="downCallback" @up="upCallback" :down="downOption" :up="upOption">

</mescroll-body>

</view>

</template>

1. 将 组件内容复制到 和 中

course-list.vue

question-list.vue

article-list.vue

1. 测试点击标签时，首次会加载，后面再次点击它，数据会缓存（不会显示 ）

加载中

5.

**显示的图片不居中：**

**~空空如也~**

**如果数据为空时，**

找 到 ：/mxg-education-app/components/mescroll-uni/components/mescroll-empty.vue 然后针对 image 元素添加样式

image {

display:inline-block;

}

**取消自动触发下拉刷新**

在搜索框输入关键字后，点击键盘上的搜索按钮，触发当前标签页的列表子组件（ 、

course-list.vue

question-

list.vue

article-list.vue

search

和 ）的 方法，来进行查询数据。

1. 在 search.vue 组件中 doSearch 方法触发子组件搜索方法 search，并使用节流方式this.$util.throttle在/common/js/util.js中声明，在 main.js 挂载到了vue实例上

doSearch(obj) {

// obj有数据，则获取

this.content = obj && obj.value ? obj.value : this.content

// 标识搜索过，隐藏keyword.vue组件内容

this.searched = true

// #ifdef MP

// 传递给小程序 搜索框, 注意上面取 `ref="searchBar"` this.$refs.searchBar.searchVal = this.content

// #endif

// 关键字保存本地, 用于历史搜索

this.storageHistory()

// 调用对应子组件方法搜索++++++++++++++++ this.$nextTick(() => {

// 节流, 避免过快多次请求

this.$util.throttle(()=> {

// 调用对应子组件方法搜索

this.$refs[`mescrollItem${this.tabIndex}`].search()

})

})

// uni.showLoading()

},

1. **将 util.js 全局挂载到 Vue 实例上, 在 main.js 添加如下代码 :**

// 工具方法 {msg, isLogin, throttle, checkStr, dateFormat, format} import \* as util from './common/js/util.js'

// 挂载到vue原型上 this.$util.msg

Vue.prototype.$util = util

1. 在列表子组件中的可以看出）

upCallback

打印日志，会发现输入关键字搜索后，

会调用两次（打印日志

async upCallback(page) {

console.log('upCallback', page.num, page.size, this.content)

// 传入0就没有数据

this.mescroll.endSuccess(0)

},

原因：当加载页面时，mescroll 会自动触发解决：取消加载页面时 mescroll 自动触发

upCallback

upCallback

方法，而search.vue又触发了一次，所以2次。方法，

无效方法：~~官方说明取消自动触发配置如下，但是没有效果，所以下面不可行（其实在mescroll-more-~~ ~~item.js文件已经配置了~~

upCallback

data() {

return {

downOption:{

auto:false // 不自动加载

},

upOption:{

auto:false // 不自动加载

}

}

},

**有效方法：**

针对多标签页面，把mescroll-more-item.js文件中的 mescrollInit() 方法里的

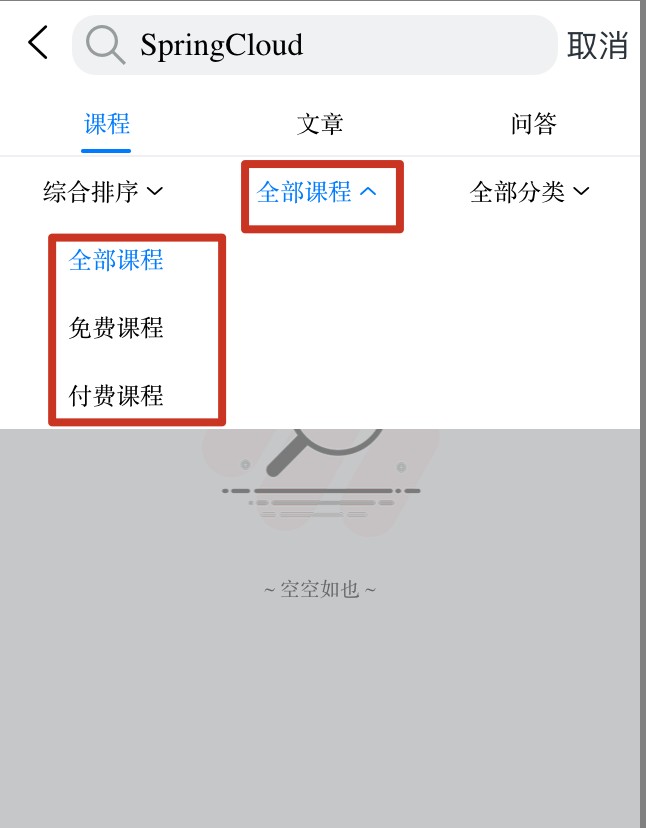
注释掉,就不会第一次自动加载，

this.mescroll.triggerDownScroll();

**课程下拉筛选处添加免费/付费下拉框**

需求：在 课程 标签下的下拉筛选处添加 ，包含免费和付费课程，如下图：

全部课程



1. 会接收父组件传递的可。

down-bar.vue

downBars

属性值，所以我们在

组件传递对应数据即

创建 下拉筛选的渲染数据文件，内容如下：

course-list.vue

/config/course-down-bar.js

// export default [] 直接导出默认数组，页面导入后数据会被页面缓存, 重新进入页面不会重新初始化这个数据，

// 需要被强制性不缓存，则使用导出默认方法形式, 引用的时候也要以方法的形式引用

export default () => { return [

{

type: 'sort', // 标识什么类型

name: '综合排序',

active: false, list: [

{

id: null,

name: '综合排序'

},

{

id: 'new', name: '最新排序'

},

{

id: 'hot', name: '热门排序'

}

]

},

{

type: 'isFree',

name: '全部课程', active: false, list: [

{

id: null,

name: '全部课程'

},

{

id: 0,

name: '付费课程'

},

{

id: 1,

name: '免费课程'

}

]

},

{

type: 'label',

name: '全部分类', active: false, isCategory: true

}

]

}

1. 在

course-list.vue

course-down-bar.js

组件中导入

课程下拉筛选数据文件，data 选项中声明 downBars

属性并调用方法获取导入的数据

**调用方法不要少了小括号()**

<script>

import downBar from './down-bar.vue'

import downBars from '@/config/course-down-bar.js' // 下拉筛选数据文件

export default {

data() {

return {

downBars: downBars(), // 下拉筛选展示数据，调用方法不要少了小括号()

}

},

}

1. 在 组件模板中将 downBars 数据传递给 组件

course-list.vue

down-bar

<down-bar @search="search" :params="params" :downBars="downBars"></down-bar>

1. 测试点击**全部课程**下拉框是否有效果

## 课程列表数据渲染

course-item.vue

1. 在

course-list.vue

组件中导入并引用

课程列表组件：

<template>

<!-- 不能用v-if (i: 每个tab页的专属下标; index: 当前tab的下标; 申明在

MescrollMoreItemMixin )-->

<view v-show="i === index">

<!-- 下拉筛选组件 -->

<down-bar @search="search" :params="params" :downBars="downBars"></down-bar>

<!-- 加载数据列表 -->

<mescroll-body :ref="'mescrollRef'+i" @init="mescrollInit" @down="downCallback" @up="upCallback" :down="downOption" :up="upOption">

<view style="padding: 0 30rpx;" >

<course-item></course-item>

</view>

</mescroll-body>

</view>

</template>

<script>

import courseItem from '@/components/common/course-item.vue' export default {

components: { downBar, courseItem },

}

1. 选项添加两个属性：

data

list

searchData

列表展示数据、

封装搜索条件数据

data() {

return {

downBars, // 下拉筛选展示数据list: [] ,// 列表数据searchData: { // 封装搜索条件数据

content: null, // 关键字内容

sort: null, // 排序类型（new最新，hot最热) isFree: null, // 0免费、1付费

labelId: null ,// 标 签 id categoryId: null, // 分类id

},

}

},

1. 在 mouted 组件生命钩子中，合并其他页面跳转传递的 params 参数到 searhData 属性中。

mounted() {

// 其他页面有传递params参数时，将需要的属性合并到搜索条件对象this.searchData

// keys 返回 searchData 对象中所有的key值的数组，为了不要冗余的属性，

this.params && Object.keys(this.searchData).forEach((key)=>{ this.searchData[key] = this.params[key] || null

})

// console.log('mounted', this.searchData)

},

1. 在 search 方法中，合并搜索框关键字内容content、下拉筛选的选中数据data，到 searchData 属性中。

// 搜索动作

search(data) {

// console.log('data', data)

// 搜索框内容, 去除前后空格

this.searchData.content = this.content && this.content.trim()

// 合并数据，data中的属性会合并到searchData中对应属性值

Object.assign(this.searchData, data)

// console.log('search', this.searchData)

// 内部会自动page.num=1,再主动触发upCallback,true显示下拉刷新动画

this.mescroll.resetUpScroll(true)

},

1. 导入

/api/couse.js

api文件，在

方法中实现查询条件分页查询逻辑，

upCallback

import api from '@/api/course.js' export default {

methods: {

/\* 下拉、上拉加载的回调 \*/ async upCallback(page) {

console.log('upCallback', page.num, page.size, this.searchData)

// 根据条件分页查询列表数据

const {data} = await api.getList(this.searchData, page.num, page.size)

// 注意：数据在 records 中, 声明一个 const 常量接收

const list = data.records

// 如果是第一页需手动置空列表,回到顶部

if(page.num == 1) { this.list = [];

this.mescroll.scrollTo(0,0)

}

//追加新数据

this.list = this.list.concat(list);

// 请求成功, 隐藏加载状态，判断是否数据全部加载完（后台接口有返回列表的总数据量 total) this.mescroll.endBySize(list.length, data.total);

},

}

}

1. 重构模板代码 ，课程列表处渲染 list 数组中的数据。

<view style="padding: 0 30rpx;" >

<course-item v-for="(item, index) in list" :key="index" :item="item"></course-item>

</view>

**文章列表数据渲染**

1. EasyMock 创建条件分页查询文章列表的数据接口描述：条件分页查询文章列表接口

URL：

/article/api/article/search

请求方式：

POST

mock.js 语法：

{

"code": 20000,

"message": "查询成功", "data": {

"total": "@integer(50, 80)", // 总记录数

"records|10": [{ //生成10条数据

"id|+1": 10, //初始值10开始，每条+1

"userId|+1": 1, // 发 布 者 id "nickName": "@cname", // 发布者昵称"title": "@csentence", // 标 题

"summary": "@csentence(50, 100)", "imageUrl|+1": [null,

"https://img.alicdn.com/bao/uploaded/i2/3603079088/O1CN01rGCkfb2H0M1O7Lj45\_!!0- item\_pic.jpg"],

"viewCount": "@integer(0, 100000)", // 浏览次数

"thumhup": "@integer(0, 100000)", // 点赞数

"updateDate": "@datetime"

}]

}

}

1. 创建关于文章的api文件 ，添加 getList 分页条件查询文件方法

/api/article.js

import request from '@/common/js/request.js' export default {

// 分页条件查询文章列表

getList(query, current = 1, size = 20) { return request({

url: '/article/api/article/search', method: 'POST',

data: {...query, current, size}

})

},

}

1. 在

article-list.vue

data

的 选项添加两个属性：

列表展示数据、

封装搜索条件数据

list

searchData

data() {

return {

list: [] ,// 列表数据

searchData: { // 封装搜索条件数据

content: null, // 关键字内容

sort: null, // 排序类型（new最新，hot最热) labelId: null ,// 标签id

categoryId: null, // 分类id

},

}

},

1. 在 mouted 组件生命钩子中，合并其他页面跳转传递的 params 参数到 searhData 属性中。

mounted() {

// 其他页面有传递params参数时，将需要的属性合并到搜索条件对象this.searchData

// keys 返回 searchData 对象中所有的key值的数组，为了不要冗余的属性，

this.params && Object.keys(this.searchData).forEach((key)=>{ this.searchData[key] = this.params[key] || null

})

// console.log('mounted', this.searchData)

},

1. 在 search 方法中，合并搜索框关键字内容content、下拉筛选的选中数据data，到 searchData 属性中。

// 搜索动作

search(data) {

// console.log('data', data)

// 搜索框内容, 去除前后空格

this.searchData.content = this.content && this.content.trim()

// 合并数据，data中的属性会合并到searchData中对应属性值

Object.assign(this.searchData, data)

// console.log('search', this.searchData)

// 内部会自动page.num=1,再主动触发upCallback,true显示下拉刷新动画

this.mescroll.resetUpScroll(true)

},

1. 在 导入

article-list.vue

/api/article.js

辑，

api文件，在

方法中实现查询条件分页查询逻

upCallback

import api from '@/api/article.js' export default {

methods: {

/\* 下拉、上拉加载的回调 \*/ async upCallback(page) {

console.log('upCallback', page.num, page.size, this.searchData)

// 根据条件分页查询列表数据

const {data} = await api.getList(this.searchData, page.num, page.size)

// 注意：数据在 records 中, 声明一个 const 常量接收

const list = data.records

// 如果是第一页需手动置空列表,回到顶部

if(page.num == 1) { this.list = [];

this.mescroll.scrollTo(0,0)

}

//追加新数据

this.list = this.list.concat(list);

// 请求成功, 隐藏加载状态，判断是否数据全部加载完（后台接口有返回列表的总数据量 total)

this.mescroll.endBySize(list.length, data.total);

},

}

}

1. 创建渲染文章列表的子组件 组件，代码如下：

/components/common/article-item.vue

<template>

<view class="article-item">

<view class="article-content row">

<view class="left-text column">

<text class="title text-ellipsis" >

{{ item.title }}

</text>

<text class="summary text-ellipsis">

{{ item.summary }}

</text>

</view>

<view v-if="item.imageUrl" class="right-image">

<image :src="item.imageUrl"></image>

</view>

</view>

<view class="article-info">

<text >

{{ item.nickName }}

</text>

<text >

* + {{ $util.dateFormat(item.updateDate) }}

</text>

<text >

* + {{ item.thumhup }} 赞

</text>

</view>

</view>

</template>

<script>

export default { props: {

item: {

type: Object, default: ()=>({

id: 1,

title: 'SpringBoot项目实战教程分布式微服务架构项目实战教程分布式微服务架构项目实战教程分布式微服务架构',

summary: 'SpringCloud OAuth2 Vue分布式微服务架构项目实战教程分布式微服务架构分布式微服务架构项目实战教程分布式微服务架构分布式微服务架构项目实战教程分布式微服务架构分布式 微服务架构项目实战教程分布式微服务架构',

imageUrl:

['http://10.idqqimg.com/qqcourse\_logo\_ng/ajNVdqHZLLAWb3qFGBhykjmcTvz9CWmwib2Qj7c3Vxjia4y5fgS](http://10.idqqimg.com/qqcourse_logo_ng/ajNVdqHZLLAWb3qFGBhykjmcTvz9CWmwib2Qj7c3Vxjia4y5fgS) oNdrMYIdH11Dl1OCraibA7u0mts/600',

nickName: '梦老师',

thumhup: 100, updateDate: new Date(),

}),

}

},

}

</script>

<style lang="scss">

.text-ellipsis { overflow: hidden;

text-overflow: ellipsis; //显示省略号

-webkit-line-clamp: 2; //最多2行

display: -webkit-box;

-webkit-box-orient: vertical; white-space: normal;

}

.article-item { padding: 30rpx;

border-bottom: 1rpx solid #f1f1f1;

.article-content {

.left-text {

.title {

font-size: 35rpx; font-weight: bold;

}

.summary {

padding-top: 5rpx; font-size: 26rpx; color: #999;

-webkit-line-clamp: 1; //最多1行

}

}

.right-image {

min-width: 220rpx; margin-left: 5rpx; image {

width: 220rpx; height: 140rpx; border-radius: 10rpx;

}

}

}

.article-info { text {

font-size: 26rpx; color: #999;

padding-right: 10rpx; &:first-child {

color: #222222;

}

}

}

}

</style>

1. 在

article-list.vue

组件中导入并引用

文章列表组件：

<template>

<!-- 不能用v-if (i: 每个tab页的专属下标; index: 当前tab的下标; 申明在

MescrollMoreItemMixin )-->

<view v-show="i === index">

<!-- 下拉筛选组件 -->

<down-bar @search="search" :params="params"></down-bar>

<!-- 加载数据列表 -->

<mescroll-body :ref="'mescrollRef'+i" @init="mescrollInit" @down="downCallback" @up="upCallback" :down="downOption" :up="upOption">

<article-item v-for="(item, index) in list" :key="index" :item="item">

</article-item>

</mescroll-body>

</view>

</template>

<script>

import articleItem from '@/components/common/article-item.vue' import api from '@/api/article.js'

export default {

components: { downBar, articleItem },

}

**解决小程序弹窗选择分类标签回显问题**

1. 问题：在微信小程序的搜索页点击和关闭弹窗

全部分类

弹出分类窗口，选择分类或标签后，在下拉筛选标题处不会回显

1. 解决：重构 down-bar.vue 组件的searchByLabel方法 ，对小程序端特殊处理下：

article-item.vue

// 分类子组件触发此方法，来查询对应分类或标签的数据

searchByLabel(label) {

// console.log('选择的标签信息', label)

// #ifdef MP

//关闭弹窗this.clickDownView(label)

// 获取最后一个全部分类对象，将标题替换

const obj = this.downBarList[this.downBarList.length-1] obj.name = label.name

obj.id = label.id

obj.categoryId = label.categoryId

// #endif

this.$emit('search', {labelId: label.id, categoryId: label.categoryId})

}

**问答列表数据渲染**

1. EasyMock 创建条件分页查询问答列表的数据接口描述：条件分页查询问答列表接口

URL：

/question/api/question/search

请求方式：

POST

mock.js 语法：

{

"code": 20000,

"message": "查询成功", "data": {

"total": "@integer(50, 60)", // 总记录数

"records|10": [{

"id|+1": 10, //初始值10开始，每条+1

"userId": "@integer(10, 30)", "nickName": "@cname", "userImage": "@image",

"title": "@csentence", // 标 题

"viewCount": "@integer(5, 300)", // 浏览数"thumhup": "@integer(2, 20)", // 点赞数"reply": "@integer(2, 20)", // 回复数"status": 1, // 1：未解决，2：已解决

"createDate": "@datetime", "updateDate": "@datetime"

}]

}

}

1. 创建关于问答的api文件 ，添加 getList 分页条件查询文件方法

/api/question.js

import request from '@/common/js/request.js' export default {

// 分页条件查询问答列表

getList(query, current = 1, size = 20) { return request({

url: '/question/api/question/search', method: 'POST',

data: {...query, current, size}

})

},

}

1. 在

question-list.vue

的 选项添加两个属性：

列表展示数据、

封装搜索条件数据

data

list

searchData

data() {

return {

list: [] ,// 列表数据

searchData: { // 封装搜索条件数据

content: null, // 关键字内容

sort: null, // 排序类型（new最新，hot最热) labelId: null ,// 标签id

categoryId: null, // 分类id

},

}

},

1. 在 mouted 组件生命钩子中，合并其他页面跳转传递的 params 参数到 searhData 属性中。

mounted() {

// 其他页面有传递params参数时，将需要的属性合并到搜索条件对象this.searchData

// keys 返回 searchData 对象中所有的key值的数组，为了不要冗余的属性，

this.params && Object.keys(this.searchData).forEach((key)=>{ this.searchData[key] = this.params[key] || null

})

// console.log('mounted', this.searchData)

},

1. 在 search 方法中，合并搜索框关键字内容content、下拉筛选的选中数据data，到 searchData 属性中。

// 搜索动作

search(data) {

// console.log('data', data)

// 搜索框内容, 去除前后空格

this.searchData.content = this.content && this.content.trim()

// 合并数据，data中的属性会合并到searchData中对应属性值

Object.assign(this.searchData, data)

// console.log('search', this.searchData)

// 内部会自动page.num=1,再主动触发upCallback,true显示下拉刷新动画

this.mescroll.resetUpScroll(true)

},

1. 在 导入

question-list.vue

/api/question.js

辑，

api文件，在

方法中实现查询条件分页查询逻

upCallback

import api from '@/api/question.js'

export default { methods: {

/\* 下拉、上拉加载的回调 \*/ async upCallback(page) {

console.log('upCallback', page.num, page.size, this.searchData)

// 根据条件分页查询列表数据

const {data} = await api.getList(this.searchData, page.num, page.size)

// 注意：数据在 records 中, 声明一个 const 常量接收

const list = data.records

// 如果是第一页需手动置空列表,回到顶部

if(page.num == 1) { this.list = [];

this.mescroll.scrollTo(0,0)

}

//追加新数据

this.list = this.list.concat(list);

// 请求成功, 隐藏加载状态，判断是否数据全部加载完（后台接口有返回列表的总数据量 total) this.mescroll.endBySize(list.length, data.total);

},

}

}

1. 创建渲染问答列表的子组件 组件，代码如下：

/components/common/question-item.vue

<template>

<view class="question-item">

<text class="title text-ellipsis" >

{{ item.title }}

</text>

<view class="foot">

<view class="info-left">

<text class="info-text">

{{item.reply}} 回 答 ·

</text>

<text class="info-text">

{{ item.viewCount }} 浏 览

</text>

</view>

<view class="info-right">

<text class="info-text">

{{ item.nickName }} ·

</text>

<text class="info-text">

{{ $util.dateFormat(item.updateDate) }}

</text>

</view>

</view>

</view>

</template>

<script>

export default {

props: {

item: {

type: Object, default: ()=>({

id: 1,

title: 'Uniapp如何快速掌握？', reply: 18,

viewCount: 999, nickName: ' 梦 老 师 ', updateDate: new Date(),

})

}

},

}

</script>

<style lang="scss">

.text-ellipsis {

// 显示两行，超出部分用...代替

overflow: hidden;// 超出隐藏

text-overflow: ellipsis; //超出部分显示省略号display: -webkit-box; // 将对象作为弹性伸缩盒子模型显示

-webkit-line-clamp: 2; //设置显示的行数

-webkit-box-orient: vertical; // 设置或检索伸缩盒对象的子元素的排列方式 纵向

}

.question-item { padding: 25rpx;

flex-direction: column;

border-bottom: 1rpx solid #F1F1F1; min-height: 80rpx;

.title {

font-size: 33rpx; font-weight: 410;

}

.foot {

display: flex;

justify-content: space-between;

.info-text {

font-size: 26rpx; color: #999;

padding-right: 10rpx;

}

}

}

</style>

1. 在

question-list.vue

组件中导入并引用

问答列表组件：

question-item.vue

<template>

<!-- 不能用v-if (i: 每个tab页的专属下标; index: 当前tab的下标; 申明在

MescrollMoreItemMixin )-->

<view v-show="i === index">

<!-- 下拉筛选组件 -->

<down-bar @search="search" :params="params"></down-bar>

<!-- 加载数据列表 -->

<mescroll-body :ref="'mescrollRef'+i" @init="mescrollInit" @down="downCallback" @up="upCallback" :down="downOption" :up="upOption">

<question-item v-for="(item, index) in list" :key="index" :item="item">

</question-item>

</mescroll-body>

</view>

</template>

<script>

import questionItem from '@/components/common/question-item.vue' import api from '@/api/question.js'

export default {

components: { downBar, questionItem },

}

**首页跳转搜索页**

**需求**

点击首页

点击首页

的名称，带上分类ID和名称，作为参数传递给搜索组件



按钮，带上对应搜索条件，作为参数传递给搜索组件

分类区

search.vue

全部

search.vue



**点击首页分类跳转搜索页**

1. 在首页分类区组件

category-box.vue

添加分类名点击事件，跳转搜索页

**注意：小程序从item中获取值无法传递成功(item.id, item.name)，需要通过categoryList[index]单独获取元素再传递**

<template>

<!-- 分类块 ( 展示排名前7的)，2行4列，8块，最后一块`全部分类` -->

<view class="category-box center">

<!-- 注意：小程序从item中获取值无法传递成功(item.id, item.name)，需要通过categoryList[index]单独获取元素再传递 -->

<view @click="toPageSearch(categoryList[index].id, categoryList[index].name,

index)"

v-for="(item, index) in categoryList.slice(0, 7)" :key="index">

{{ item.name }}

</view>

<!-- index传递-1是因为category.vue弹窗高亮时，会+1 -->

<view @click="toPageSearch(null, '全部分类', -1)">全部分类</view>

</view>

</template>

1. 在

category-box.vue

methods

的 选项中添加

方法实现跳转动作

methods: {

toPageSearch(id, name, index) {

const params = {categoryId: id, name: name, activeIndex: index} this.navTo(`/pages/search/search?params=${ JSON.stringify(params) }`)

}

}

1. 重构

/pages/search/components/down-bar.vue

高亮显示

下拉筛选组件中的 params 监听器，传递一个分类id，用于

toPageSearch

watch: {

params: { // 监听其他页跳转到搜索页带的参数

handler(newVal) { if(newVal) {

// console.log('监听其他页跳转到搜索页带的参数', newVal)

// 如果是`分类页面`点击`标签`后跳转过来的，

// 则获取 `downBars` 最后一个元素，将分类标题替换掉即可

if(newVal.name) {

const obj = this.downBars[this.downBars.length-1]

// 将`全部分类`按钮的文字更新为新的标签名称

obj.id = newVal.labelId obj.name = newVal.name

// 为了打开分类弹窗后，高亮显示当前是哪个分类

obj.activeIndex = newVal.activeIndex

// 首 页 传 递 分 类 id +++++++ obj.categoryId = newVal.categoryId return //不要少了

}

}

},

immediate: true //第1次即监听

}

}

**点击首页**

**全部**

1. 在首页

index.vue

**跳转搜索页**

模板中传对应子组件传递条件参数 :params

<view class="list-container">

<swiper-course name="热门推荐" word="HOT" :courseData="hotCourseList" :params="{sort: 'hot'}"></swiper-course>

<scroll-course name="近期上新" word="NEW" :courseData="newCourseList" :params="{sort:

'new'}"></scroll-course>

<swiper-course name="免费精选" word="FREE" :courseData="freeCourseList" :params="

{isFree: 1}"></swiper-course>

<list-course name="付费精品" word="NICE" :courseData="payCourseList" :params="{isFree: 0}"></list-course>

</view>

1. 在

swiper-course.vue

scroll-course.vue

category-box.vue

list-course.vue

、 、 、

每个组件中的

props声明 params ，接收父组件传递的 params。

并在模板传递给

list-box.vue

export default {

components: {listBox, courseItem}, props: {

params: Object, // 点击`全部`跳转传递的参数 +++++

<list-box :name="name" :word="word" :params="params">

1. 在

list-box.vue

全部

@click="toSearchPage"

的props声明 params接收参数元素上绑定点击事件函数

methods 声明 方法，实现跳转搜索页，**注意：引用 this.params 不要少了 this.**

toSearchPage

<template>

<text @click="toSearchPage">全部</text>

</template>

<script>

export default { props: {

params: Object, // 点击`全部`跳转传递的参数

name: { // 名 称

type: String, default: '热门推荐'

},

word: { // 高亮单词，如： 'HOT'

type: String, default: null

}

},

methods: {

toSearchPage() { console.log('this.params', this.params)

this.navTo(`/pages/search/search?params=${ JSON.stringify(this.params) }`)

}

}

}

</script>

1. 在 down-bar.vue 的 params 监听器中完成回显下拉筛选处信息，

watch: {

params: { // 监听其他页跳转到搜索页带的参数

handler(newVal) {

if(newVal) {

// console.log('监听其他页跳转到搜索页带的参数', newVal)

// 如果是`分类页面`点击`标签`后跳转过来的，

// 则获取 `downBars` 最后一个元素，将分类标题替换掉即可

if(newVal.name) {

const obj = this.downBars[this.downBars.length-1]

// 将`全部分类`按钮的文字更新为新的标签名称

obj.id = newVal.labelId obj.name = newVal.name

// 为了打开分类弹窗后，高亮显示当前是哪个分类

obj.activeIndex = newVal.activeIndex

// 首页传递分类id

obj.categoryId = newVal.categoryId return //不要少了

}

// 其他页面(非分类页)跳转过来参数 +++++++++++++++

// 1. 从数组中找到对应类型的目标元素

// 取 params 对象中第1个key名，与item.type进行匹配

const key = Object.keys(newVal)[0]

// item.type 与 key 匹配，找到目标元素

const target = this.downBars.find( item => { return item.type === key

})

// 2. 从目标元素的 list 数组元素id匹配value,

// 匹配到了，将对应名称name,显示到标题上

const value = newVal[key] target.list.find(item => {

if(item.id === value) {

// 赋值到目标下拉对象上，显示对应的标题信息

target.name = item.name

target.id = item.id

//匹配到了return true,不然会执行下去

return true

}

})

}

},

immediate: true //第1次即监听

}

}

**列表子组件mounted合并参数0问题**

1. 问题：在首页点击 **付费精品** 右侧 **全部** 按钮带上的请求参数是

{isFree: 0}

，在子组件

course-list.vue

的 mounted 中将请求参数 params 合并到查询对象 searchData 时，isFree是 null,而不传递的0，正常应该是0

1. 解决：

因为 0 在js中认为就是false, 所以 此时认为是false, 所以取的是 .

this.params[key] || null

null

所以在 mounted 中对参数值为0的情况进行处理，为0也是有效值。

在 、 、 组件中均重构为以下代码：

course-list.vue

article-list.vue

question-list.vue

mounted() {

// 如果有其他页面带的请求参数，

//则获取对应的请求参数值，将参数值与 this.searchData合并

// Object.keys 返回对象中所有的key名称，返回值数组

this.params && Object.keys(this.searchData).forEach((key)=>{

// 防止参数值为0也是有效的，所以要将0也正常处理，因为默认0会被false this.searchData[key] =

( this.params[key] || this.params[key] === 0 ) ? this.params[key] :

null

})

// console.log('问答mounted合并的searchData', this.searchData)

},

**课程详情页实现**



**创建与配置页面**

1. 创建课程详情页面文件 ，设置当前页面背景色为灰色

<template>

<view>

课程详情页

</view>

</template>

course/course-details.vue

,{ // 课程详情页

"path" : "pages/course/course-details",

"style" :

{

"navigationBarBackgroundColor": "#345DC2", "navigationBarTextStyle":"white",

"app-plus": {

"bounce": "none", // 禁止app回弹效果"scrollIndicator": "none", // 隐藏页面滚动条（APP） "titleNView": { //导航配置

"type": "transparent" // 滚动透明渐变, 后面不要逗号

// #ifdef APP-PLUS

,"buttons": [ // 只有app才有，注意最前面放逗号,

{

"type":"share" // 分享按钮

}

]

// #endif

}

}

}

}

1. 轮播图跳转课程详情页面检查EasyMock上的

获取指定位置的广告信息

数据接口的

是否正确

advertUrl



找到轮播图组件 /components/common/mxg-banner.vue 的 image 元素绑定点击事件，跳转到详情页

<image :src="item.imageUrl" @click="navTo(`${item.advertUrl}`)"/>

1. 在课程列表公共组件 components/common/course-item.vue 的最外层view添加点击事件，带上课程ID跳转到详情页

<template>

<view @click="navTo(`/pages/course/course-details?id=${item.id}`)" class="course-item" :class="{column: isColumn}" >

**主图和简介组件**



1. 创建主图和简介子组件 /pages/course/components/course-header.vue

<template>

<view class="course-header">

<image class="img" src="/static/images/banner1.jpg" lazy-load></image>

<view class="header-info">

<view class="price-info">

<!-- 优 惠 价 -->

<text v-if="true">￥7480.00</text>

<!-- 原价一直会显示（有优惠价时，显示price类名） -->

<text class="price">￥8800.00</text>

<text v-if="true" class="youhui">优惠价</text>

</view>

<view class="title">

Uniapp全家桶在线教育视频项目

</view>

<view class="count">

<text class="iconfont icon-haoping2">100%好评</text>

<text class="iconfont icon-touxiang2">1999人在学</text>

</view>

</view>

</view>

</template>

<script>

</script>

<style lang="scss">

.course-header{

.img {

width: 750rpx; height: 420rpx;

// 去掉底部白色背景

display: block;

}

.header-info { padding: 25rpx;

background-color: #FFFFFF;

border: 25rpx solid $mxg-color-grey; border-left: 0; //左边不无框

border-right: 0; //右边不无框

}

.price-info { text {

margin-right: 15rpx; &:first-child{

color: $mxg-text-color-red; font-size: 46rpx;

font-weight: bold;

}

}

.price {

color: #B6BBBF; font-size: 30rpx;

text-decoration: line-through; // 删 除 线

}

.youhui {

color: $mxg-text-color-red; font-size: 20rpx;

border: 1px solid $mxg-text-color-red; border-radius: 10rpx;

padding: 0 3rpx;

}

}

.title {

font-size: 35rpx; font-weight: bold; color: #1d1d1f; padding-left: 8rpx;

}

.count {

padding: 15rpx 0; text {

margin-right: 15rpx; font-size: 20rpx; color: #7d828f; border-radius: 30rpx; padding: 15rpx;

background-color: $mxg-color-grey;

}

}

}

</style>

1. 声明props 属性接收父组件传递的值动态渲染

<template>

<view class="course-header">

<image :src="course.mainImage" lazy-load></image>

<view class="header-info">

<view class="price-info" v-if="!course.isFree">

<!-- 优 惠 价 -->

<text v-if="course.priceDiscount">￥{{course.priceDiscount}}</text>

<!-- 原价一直会显示（有优惠价时，显示price类名） -->

<text :class="{price: course.priceDiscount}">￥{{course.priceOriginal}}

</text>

<text v-if="course.priceDiscount" class="youhui">优惠价</text>

</view>

<view class="title">

{{course.title}}

</view>

<view class="count">

<text class="iconfont icon-haoping2">{{course.goodRate}}好评</text>

<text class="iconfont icon-touxiang2">{{course.studyTotal}}人在学</text>

</view>

</view>

</view>

</template>

<script>

export default { props: {

course: {

type: Object,

default: ()=>({

"id": 10,

"title": "Uniapp兼容多端在线教育项目实战", "studyTotal": 590, //在学人数

"goodRate": '100%', //好评率

"mainImage": "https://gd4.alicdn.com/imgextra/i4/3603079088/O1CN01dczOSM2H0LvTowhkl\_!!3603079088.png",

"isFree": 0, // 0 付费，1免费

"priceOriginal": 614.51, //原价"priceDiscount": 281.91 // 优惠价

})

}

}

}

</script>

1. 导入和引用

<template>

<view>

<!-- 主图和简介 -->

<course-header ></course-header>

</view>

</template>

<script>

import courseHeader from './components/course-header.vue' export default {

components: {courseHeader},

}

</script>

**标签选项卡与详情内容静态模板**

1. 选项卡有4个标签，所以针对课程详情页创建一个配置文件

/config/course-details-tabs.js

export default [

{

id: 1, name: '详情'

},

{

id: 2, name: '章节'

},

{

id: 3, name: '评论'

},

{

id: 4,

name: '套餐'

}

]

1. 编写模板代码，引用 组件和swiper左右切换、scroll-view 上下滚动

tab-bar.vue

<template>

<view>

<!-- 主图和简介 -->

<course-header></course-header>

<!--

所以：整个页面高度 = 上面course-header高度 + 下面view高度下面区域的高度为占用的是空页面时的视口高度，

-->

<view class="course-details" :style="'height:' + pageHeight + 'px'">

<!-- 标签选项卡 -->

<tab-bar :tabs="tabs" v-model="tabIndex"></tab-bar>

<!--

duration 滑动动画时长

current 当前所在滑块的 index @change 当 current 改变时会触发circular 到末尾后重新回到开头

-->

<swiper class="swiper-box" circular :duration="300" :current="tabIndex" @change="changeTab" >

<swiper-item v-for="(item, index) in tabs" :key="index" >

<!--

scroll-y 是否允许纵向滚动

upper-threshold 距顶部多远时(单位px) @scrolltoupper 滚动到顶部触发

-->

<scroll-view id="scroll-box" class="scroll-box"

:scroll-y="enableScroll" upper-threshold="0" @scrolltoupper="scrolltoupper"

>

<view class="details-info">

<view v-for="i in 200" :key="i">

{{tabIndex}} —— {{i}}

</view>

</view>

</scroll-view>

</swiper-item>

</swiper>

</view>

</view>

</template>

1. 导入 组件和导入 数据

tab-bar.vue

/config/course-details-tabs.js

<script>

import courseHeader from './components/course-header.vue'

import tabBar from '@/components/common/tab-bar.vue' import tabs from '@/config/course-details-tabs.js'

export default {

components: {courseHeader, tabBar},

data() {

return {

tabs, // 标签导航

tabIndex: 0 ,// 当前所在标签enableScroll: true, // 是否允许滚动statusNavHeight: 0, // 状态栏+导航高度pageHeight: 300, // 详情内容区域高度

}

},

onLoad() {

//获取当前页面视口高度this.getPageHeight()

},

methods: {

// 左右切换时触发

changeTab(e) {

this.tabIndex = e.detail.current

},

// scroll-view 滚动到顶部时触发

scrolltoupper() {

},

//获取当前页面视口高度getPageHeight() {

const res = uni.getSystemInfoSync()

// 平 台

const system = res.platform

// 状态栏高度

const statusBarHeight = res.statusBarHeight

// 状态栏+导航高度

if (system === 'android') {

this.statusNavHeight = (48 + statusBarHeight)

} else if (system === 'ios') { this.statusNavHeight = (44 + statusBarHeight)

}

// console.log('statusNavHeight', this.statusNavHeight)

// screenHeight 是整合手机屏幕的高度，

// windowHeight 是webview（不包括手机通知栏、小程序标题栏和tabBar）的页面高度；

// 如果导航是渐变效果, 上面两个值是一样的，需要自己计算，减去后面显示出来的状态导

航高

// 页面视口高度

this.pageHeight = res.windowHeight - this.statusNavHeight

},

}

}

</script>

1. 编写课程详情页面样式

<style lang="scss">

.course-details{

// 内容超出空页面高度时，启动 scroll-view 滚动

overflow: hidden;

.swiper-box, .scroll-box{ height: 100%;

}

.details-info {

// 解决底部被遮挡隐藏（底部有：立即购买）

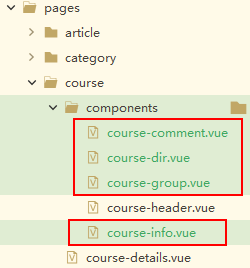
padding-bottom: 180rpx;

}

}

</style>

**左右切换详情内容组件**

1. 创建选项对应子组件文件 course/components/course-comment.vue 、course-dir.vue、course- info.vue、course-group.vue
2. js区域导入组件、在详情区域导入引用各子组件

<script>

import courseHeader from './components/course-header.vue'

import tabBar from '@/components/common/tab-bar.vue' import tabs from '@/config/course-details-tabs.js'

import courseInfo from './components/course-info.vue' import courseDir from './components/course-dir.vue'

import courseComment from './components/course-comment.vue' import courseGroup from './components/course-group.vue' export default {

components: {

courseHeader, tabBar,

courseInfo, courseDir, courseComment, courseGroup

},

<swiper class="swiper-box" circular :duration="300"

:current="tabIndex" @change="changeTab" >

<swiper-item v-for="(item, index) in tabs" :key="index" >

<scroll-view id="scroll-box" class="scroll-box"

:scroll-y="enableScroll" upper-threshold="0" @scrolltoupper="scrolltoupper"

>

<view class="details-info">

<course-info v-if="index==0"></course-info>

<course-dir v-if="index==1"></course-dir>

<course-comment v-if="index==2"></course-comment>

<course-group v-if="index==3"></course-group>

</view>

</scroll-view>

</swiper-item>

</swiper>

**优化上下滚动效果**

目前有两条滚动条：一条页面滚动条，一条

scroll-view

1. 在 pages.json中配置隐藏页面滚动条（ 启项目（只有APP有效）
2. 禁用详情区域滚动操作

详情区域滚动条

）,如果禁用还是有则重

style.app-plus.scrollIndicator: "none"

data中修改 为 ， 组件上绑定

enableScroll

false

scroll-view

:scroll-y="enableScroll"

data() {

return {

tabs, // 标签导航

tabIndex: 0 ,// 当前所在标签

enableScroll: false, // 是否允许滚动 ++++++ 改false statusNavHeight: 0, // 状态栏+导航高度

pageHeight: 300, // 详情内容区域高度

}

},

1. 监听

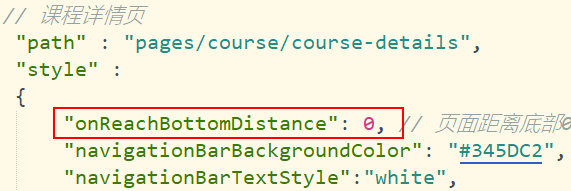
页面滚动条

style.onReachBottomDistance: 0

到达底部事件要在page.json中配置

，默认是50。

onReachBottom



onReachBottom 监听到达页面底部后，开启

scroll-view

true ， 这样详情区域就可滚动。

详情区域纵向滚动

this.enableScroll =

// 到达页面底部触发（与methods同级）

onReachBottom(){

// 详情区域可滚动

this.enableScroll = true

},

1. 监听滚动到详情区域顶部时

scroll-view 添 加 upper-threshold="0" 和 @scrolltoupper="scrolltoupper"

在 scrolltoupper 方法中，将页面区域也回到顶部 uni.pageScrollTo

滚动到页面顶部，禁用详情滚动 this.enableScroll = false

// scroll-view 滚动到顶部时触发（methods中定义）

scrolltoupper() {

// 页面区域也回到顶部

uni.pageScrollTo({ scrollTop: 0,

duration: 150

})

// 滚动到页面顶部，禁用详情滚动

this.enableScroll = false

},

1. 当页面回到顶部或处于页面滚动时，那就禁止滚动详情区域

在 data 选项声明头部简介的高度）

detailTop

，onReady 页面初次渲染完成钩子中，获取详情标签到顶部高度（就是

data() {

return {

detailTop: 0, //详情区域距离顶部高度（头部高度）

}

},

// 页面初次渲染完成

onReady() {

//获取详情区域距离顶部高度

let view = uni.createSelectorQuery().in(this).select(".course-details"); view.fields({

rect: true

}, data => {

// console.log("节点的顶部高度为" + data.top); this.detailTop = data.top

}).exec();

},

添加实时监听页面滚动的 事件，判断当前滚动条距离顶部高度 <

onPageScroll(e)

e.scrollTop

this.detailTop 详情标签到顶部高度，且 this.enableScroll为true时， 使用

this.enableScroll=false 禁止详情滚动

// 实时监听页面滚动 （与methods同级）

onPageScroll(e) {

// 当前滚动条距离顶部高度 小于 详情标签到顶部高度

if(this.enableScroll && e.scrollTop < this.detailTop){

//禁止详情滚动this.enableScroll = false

}

},

1. 在 onLoad 钩子里，禁止点击状态栏回到顶部（因为 scroll-view 无法回到顶部，所以禁用它）

onLoad() {

//获取当前页面视口高度this.getPageHeight()

// #ifdef APP-PLUS

// 禁用点击状态栏回到顶部

this.noStatusScrollTop()

// #endif

},

methods: {

// 禁用点击状态栏回到顶部

noStatusScrollTop() {

const webview = this.$scope.$getAppWebview(); webview.setStyle({'scrollsToTop': false})

},

}

**切换选项-将页面滚动条到达底部**

需求1：

如果上下滑动了详情区域的内容，而切换到章节选项后，在章节选项中进行了页面滚动，也就是头部显示出 来了，此时切换回详情选项，发现头部显示了内容，详情内容又被滑动了，

正常是：内容被滑动了，头部应该就是被挡住的，也就是页面滚动条应该在底部的。 解决：

只要切换选项，就将页面滚动条到达底部。在 changeTab 中添加 uni.pageScrollTo 切换到页面底部。

// 左右切换时触发

changeTab(e) {

this.tabIndex = e.detail.current

// 切换选项，将页面滚动条切换到底部 ++ uni.pageScrollTo({

scrollTop: this.detailTop, duration: 300

})

},

需求2：

切换选项后，详情区域不可以滚动，页面到达底部 onReachBottom 事件可能不会被触发，或触发后还会调用

onPageScroll

onPageScroll

； 后面被触发说明 enableScroll 就是 false，详情区域不可以滚动；

**在 和 打印日志可查看执行先后顺序**

**onReachBottom**

**onPageScroll**

解决：在 onPageScroll 中通过 e.scrollTop 判断页面是否已经到了底部,

说明到达页面底部，

this.detailTop-this.statusNavHeight <= e.scrollTop

就允许详情区域滚动

this.enableScroll=true

// 实时监听页面滚动

onPageScroll(e) {

// console.log('实时监听页面滚动', e.scrollTop)

// 到达页面底部问题，触发了onReachBottom底部事件，后面还会触发onPageScroll，

// 这样会导致enableScroll会变成false，导致页面到底部了，详情区域也不可以

// 当页面滚动条到达底部时，就是滚动条距离 <= 选项卡到顶部距离，开启详情区域纵向滚动

if(this.detailTop-this.statusNavHeight <= e.scrollTop){

// 可以滚动详情区域this.enableScroll = true return

}

// 当前滚动条距离顶部高度 小于 详情标签到顶部高度

if(this.enableScroll && e.scrollTop < this.detailTop){

//禁止详情滚动this.enableScroll = false

}

},

**详情页组件实现**

详情页子组件主要以图片形式展示课程介绍

1. 详情页子组件模板代码

<template>

<view class="center column">

<!-- mode="widthFix" 宽度比例不变 -->

<image v-for="(url, index) in detailUrls" :key="index" :src="url" @load="load" lazy-load mode="widthFix"></image>

<image v-if="loading" class="loading" src="/static/images/loading.gif" ></image>

<!-- 写相对路径，真机上不显示 -->

<text v-else class="footer-text">已经到达底部，没有更多内容了</text>

</view>

</template>

1. 接收父组件传递的参数

<script>

export default { props: {

detailUrls: {

type: Array, default: ()=>[

'https://img.alicdn.com/imgextra/i2/3603079088/O1CN01V8VO2y2H0M5Bx33L3\_!!3603079088.jpg'

]

}

},

data() {

return {

loading: true //图片是否加载中

}

},

methods: {

load() {

//加载完成this.loading = false

}

}

}

</script>

1. 样式代码

<style >

image {

width: 100%;

}

.loading {

height: 90rpx; width: 90rpx; margin-top: 50rpx;

}

.footer-text {

font-size: 30rpx; color: #777777;

margin: 20rpx 0;

}

</style>

**章节列表组件实现**

章节子组件主要展示课程章节信息，如下图



1. 章节子组件模板代码

<template>

<view class="course-dir">

<view v-for="(chapter, index) in chapterList" :key="index">

<!-- 第 几 章 -->

<text class="chapters text-ellipsis">第{{index+1}}章 {{chapter.name}}</text>

<!-- 第 几 节 -->

<view class="sections row" v-for="(section, index2) in chapter.sectionList"

:key="index2">

<text class="iconfont icon-roundrightfill"></text>

<view class="row">

<text>{{index+1}}-{{index2+1}}</text>

<text class="title text-ellipsis">{{section.name}}</text>

</view>

<text v-if="section.isFree" class="see">试看</text>

</view>

</view>

</view>

</template>

1. 接收父组件传递的参数

<script>

export default { props: {

chapterList: { //章节信息

type: Array, default: ()=>[

{

id: 1,

name: ' 课 程 介 绍 ', sectionList:[ //课节信息

{

id: 1,

name: '什么是uni-app框架',

isFree: 1, // 是否付费（0付费，1免费）

},

{

id: 2,

name: '安装HBuilderX和插件',

isFree: 0, // 是否付费（0付费，1免费）

},

{

id: 3,

name: '创建uni-app项目和运行Web浏览器', isFree: 0, // 是否付费（0付费，1免费）

}

]

},

{

id: 2,

name: '初始化 在线教育APP项目', sectionList:[ //课节信息

{

id: 1,

name: '创建运行项目和完善项目目录', isFree: 0, // 是否付费（0付费，1免费）

},

{

id: 2,

name: '拷贝静态资源到项目和mixin混合讲解', isFree: 0, // 是否付费（0付费，1免费）

}

]

}

]

}

}

}

</script>

1. 样式代码

<style lang="scss">

.course-dir {

padding: 0 36rpx;

.chapters {

margin-top: 30rpx; margin-bottom: 10rpx;

color: $mxg-text-color-black; font-size: 35rpx;

font-weight: bold;

}

.sections {

padding: 20rpx 0; font-size: 30rpx;

border-bottom: $mxg-underline; line-height: 40rpx;

color: #3d3c40; text {

margin-right: 10rpx;

}

.title {

width: 470rpx;

}

.see {

color: $mxg-text-color-blue; font-size: 25rpx;

// 靠 右

flex: 1;

text-align: right;

}

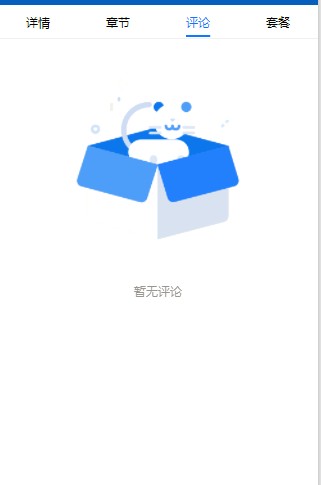
}

}

</style>

**评论组件实现**

评论子组件主要展示评论信息，如下图



1. 创建没有数据时展示内容组件

/components/no-data/no-data.vue

**在 /components 下的目录名和组件名一致，其他引入此组件时，不需要 import 导入，直接模板中引用**

<template>

<view>

<view class="center column">

<image src="/static/images/no.png" lazy-load mode="aspectFill"></image>

<text class="title">{{desc}}</text>

</view>

</view>

</template>

<script>

export default { props: {

desc: {

type: String, default: '暂无数据'

}

}

}

</script>

<style>

.title {

color: #999;

font-size: 30rpx; margin-top: 30rpx;

}

</style>

1. 评论子组件模板代码

<template>

<view>

<no-data v-if="!commentList || commentList.length<=0" desc="暂无评论"></no-data>

<!-- 评 论 -->

<view v-else class="comment">

<view class="comment-item" v-for="(item, index) in commentList" :key="index">

<view class="info">

<image :src="item.userImage || '/static/tab/my.png'"

:style="{'padding': !item.userImage && '10rpx'}" mode="aspectFill">

</image>

<view class="user">

<view>{{item.nickName}}</view>

<view>{{ $util.dateFormat(item.createDate) }}</view>

</view>

<text class="iconfont icon-haoping2" :class="{grey: !item.isGood}">

</text>

</view>

<view class="content">

{{item.content}}

</view>

<view class="replay" v-if="item.children && item.children.content">

<text>讲师回复：</text>

<text>{{item.children.content}}</text>

</view>

</view>

</view>

</view>

</template>

1. 接收父组件传递的参数

<script>

export default { props: {

commentList: { type: Array, default: () => [

{

"id": "1",

"parentId": "-1",

"nickName": " 萧 强 ", "userImage": null, "isGood": 1, // 1好评，0差评

"content": "很认可 ，这套课程项目与企业开发流程功能一样",

"createDate": Date.now(), //wx小程序不支持new Date()

"children": null

},

{

"id": "2",

"parentId": "-1",

"nickName": "江秀英", "userImage":

"https://fuss10.elemecdn.com/e/5d/4a731a90594a4af544c0c25941171jpeg.jpeg", "isGood": 0, // 1好评，0差评

"content": "但非约事京或改各史习反候至百。",

"createDate": "2008-03-12 09:08:06",

"children": {

"id": "3",

"parentId": "2",

"nickName": "李涛", "userImage": null,

"content": "感谢你一直以来的支持！ -陪你学习，伴你梦想！",

"createDate": "2008-03-12 09:08:06",

}

}

]

}

}

}

</script>

1. 样式代码

<style lang="scss">

/\* 评 论 \*/

.comment {

background-color: #FFFFFF; padding: 0 36rpx;

font-size: 30rpx;

.comment-item { margin-top: 36rpx;

border-bottom: $mxg-underline;

.info {

display: flex;

align-items: center; image{

width: 70rpx; height: 70rpx;

border-radius: 50rpx;

margin-right: 20rpx;

}

.user {

font-weight: bold; line-height: 33rpx;

:last-child {

color: #999;

font-size: 25rpx; font-weight:normal;

}

}

:last-child {

// 最右边

margin-left: auto; font-size: 35rpx; color: #ff001b;

}

.grey {

color: $mxg-text-color-grey;

}

}

.content {

margin: 25rpx 0;

}

.replay {

margin-bottom: 30rpx; color: #7d828f;

background-color: #F8F9FB; padding: 15rpx;

border-radius: 10rpx;

}

}

}

</style>

**套餐组件实现**

套餐评论子组件主要展示套餐信息，如下图



1. 套餐组件模板代码

<template>

<view>

<no-data v-if="!groupList || groupList.length<=0" desc="暂无套餐"></no-data>

<view v-else class="group-list">

<!-- Android平台weex对阴影样式(box-shadow)支持不完善,

新增elevation属性（组件的属性，不是css样式）设置组件的层级， Number类型，层级值越大阴影越明显，阴影效果也与组件位置有关， 越靠近页面底部 阴影效果越明显 -->

<view class="group-item" elevation="12px"

v-for="(item, index) in groupList" :key="index"

>

<text class="title">{{item.title}}</text>

<course-item v-for="(info, i) in item.list" :key="i" :item="info"></course-

item>

<view class="price-box space-between center">

<view class="price">

<text class="group-price">￥{{item.groupPrice}}</text>

<text class="total-price">￥{{item.totalPrice}}</text>

</view>

<text class="buy" @click="buyGroupHandler(item)">购买套餐</text>

</view>

</view>

</view>

</view>

</template>

1. 将

03-配套资料\mock\courseGroupData.js

1. 导入

course-item.vue

套餐模拟数据导入项目的 mock 目录下

，和接收父组件传递的参数

<script>

import courseItem from '@/components/common/course-item.vue' import groupData from '@/mock/courseGroupData.js'

export default {

components: { courseItem }, props: {

groupList: {

type: Array,

default: ()=> groupData

}

},

}

</script>

1. 样式代码

<style lang="scss">

.group-item {

margin: 36rpx 30rpx; background-color: #fff; border-radius: 20rpx;

box-shadow: 2px 0 10px 0 rgba(0,0,0, 0.1);

padding: 0 20rpx;

.title {

font-size: 35rpx; font-weight: 500; padding-top: 20rpx;

}

.price-box {

color: $mxg-text-color-red; font-size: 30rpx;

line-height: 80rpx; font-weight: bold;

.group-price {

font-size: 37rpx;

margin-right: 10rpx;

}

.total-price { color: #999;

font-weight: normal;

text-decoration: line-through;

}

}

}

</style>

**立即购买按钮**

需求：

课程详情页面底部添加一个立即购买按钮，我们将它封装为子组件

/pages/course/components/bottom-btn.vue

<template>

<view class="bottom center" @click="clickHandler">

<text class="bottom-btn">{{btnText}}</text>

</view>

</template>

<script>

export default { props: {

btnText: {

type: String, default: '立即购买'

},

courseId: [String, Number]

},

methods: {

clickHandler() {

// 点击按钮触发事件

}

}

}

</script>

<style lang="scss">

.bottom {

position: fixed; left: 0;

right: 0;

bottom: 0;

background-color: #FFFFFF; height: 100rpx;

border-top: $mxg-underline;

}

.bottom-btn {

width: 700rpx;

background-color: #345DC2; text-align: center;

font-size: 35rpx; color: #FFFFFF; border-radius: 50rpx; line-height: 80rpx;

}

</style>

在 course-details.vue 导入并引用 ，注意：**padding-bottom: 180rpx;**

bottom-btn.vue

<!-- 底部按钮：立即购买 -->

<bottom-btn></bottom-btn>

</view>

</template>

<script>

import bottomBtn from './components/bottom-btn.vue' export default {

components: {

courseHeader, tabBar, courseInfo, courseDir, courseComment, courseGroup, bottomBtn

},

}

</script>

<style lang="scss">

.course-details { overflow: hidden;

.swiper-box, .scroll-box { height: 100%;

}

.details-info {

// 被隐藏的80rpx（标签选项卡高度）

padding-bottom: 180rpx;

}

}

</style>

# 课程详情页数据接口

## EasyMock 创建课程数据接口

1. EasyMock 创建查询课程基本信息的数据接口描述：查询课程基本信息接口

URL：

/course/api/course/{id}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功", "data": {

"id": 10,

"userId": "@integer(10, 30)", "nickName|1": "@cname", "userImage": "@image", "title": "@csentence", // 标题"totalTime": "@time", //总时长

"studyTotal": "@integer(100, 700)", // 在学人数"goodRate|1": ["100%", "99%"], //好评率"mainImage":

"https://gd4.alicdn.com/imgextra/i4/3603079088/O1CN01dczOSM2H0LvTowhkl\_!!3603079088.png ",

"isFree": 0, // 0 付费，1免费

"priceOriginal|501-800.2": 1, //原价, 501到800之间随机数，小数点2位"priceDiscount|60-500.2": 1, //优惠价

"detailUrls": [ //课程详情图

'https://img.alicdn.com/imgextra/i2/3603079088/O1CN01V8VO2y2H0M5Bx33L3\_!!3603079088.jp g'

]

}

}

1. EasyMock 创建通过课程id查询章节列表的数据接口描述：通过课程id查询章节列表数据

URL：

/course/api/chapter/section/list/{id}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功",

"data|8-30": [{ // 章

"id": 10,

"name": '@ctitle', // 7到15个字"sectionList|3-18": [ //课节信息

{

"id|+1": 1,

"name": '@ctitle(7, 15)', // 7到15个字"isFree|1": [0, 1], // 是否付费（0付费，1免费） "videoUrl|+1": [

['http://admin.mengxuegu.com/video/hls/git.m3u8'](http://admin.mengxuegu.com/video/hls/git.m3u8%27), ['http://baobab.kaiyanapp.com/api/v1/playUrl?](http://baobab.kaiyanapp.com/api/v1/playUrl)

vid=164016&resourceType=video&editionType=low&source=aliyun&playUrlType=url\_oss'

]

}

]

}]

}

1. EasyMock 创建通过课程id查询评论列表的数据接口描述：通过课程id查询评论列表数据

URL：

/course/api/comment/list/{id}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功",

"data|3-10": [{ "id|+1": 1,

"parentId": "-1", "userId|+1": 1, // 评论者id "nickName": "@cname",

"userImage|1": [null, "https://fuss10.elemecdn.com/e/5d/4a731a90594a4af544c0c25941171jpeg.jpeg"],

"isGood|1": [1, 0], // 1好评，0差评

"content": "@csentence", // 评论内容

"createDate": "@datetime", "children|1": [

null,

{ // 老师回复

"id": 19,

"userId": 1, // 回复者id

"nickName": "@cname", // 回复者昵称

"userImage": null,

"content|1": ["@csentence", "感谢你一直以来的支持！ -陪你学习，伴你梦想！"], "createDate": "@datetime",

}

]

}]

}

1. EasyMock 创建通过课程id查询套餐搭配的数据接口描述：通过课程id查询套餐搭配数据

URL：

/course/api/group/list/{id}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功",

"data|1-3": [{ // 1到3个套餐

"id|+1": 1, //初始值1开始，每条+1 "title": '@ctitle(7,10)——组合套餐', "groupPrice|301-800.2": 1, //套餐价,

"totalPrice|401-1200.2": 1, //总价, "list|2-4": [{ //每个套餐2到4个课程

"id|+1": 10, //初始值10开始，每条+1

"userId": "@integer(10, 30)",

"nickName|1": ["@cname", '梦老师'], "userImage": "@image",

"title": "@csentence", // 标题

"totalTime": "@time", //总时长

"studyTotal": "@integer(100, 700)", // 在学人数

"commTotal": "@integer(100, 600)", // 好评数

// 主 图

"mainImage|1": ['/static/images/banner2.jpg', '/static/images/banner1.jpg',

'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg'], "isFree": 0, //是否免费：0免费，1收费，

"priceOriginal|501-800.2": 1, //原价, 501到800之间随机数，小数点2位

"priceDiscount|60-500.2": 1 //优惠价

}]

}]

}

## 封装API请求接口方法

1. 在课程的api文件 封装请求上面接口的4个方法

/api/course.js

,

// 通过课程id查询基本信息

getCourseById(id){ return request({

url: `/course/api/course/${id}`, method: 'GET'

})

},

// 通过课程id查询章节列表数据

getChapterSectionList(id) { return request({

url: `/course/api/chapter/section/list/${id}`, method: 'GET'

})

},

// 通过课程id查询评论列表数据

getCourseCommentList(id) { return request({

url: `/course/api/comment/list/${id}`, method: 'GET'

})

},

// 通过课程id查询套餐搭配数据

getCourseGroupList(id) { return request({

url: `/course/api/group/list/${id}`, method: 'GET'

})

},

**页面调用数据接口渲染**

1. 导入 course.js ，data 声明渲染数据属性，onLoad 接收课程ID和调用数据接口

<script>

import api from '@/api/course.js' export default {

data() {

return {

id: null, // 课 程 id course: {}, //课程基本信息

chapterList: [], // 章节信息

commentList: [], // 评论信息

groupList: [] // 套餐信息

}

},

onLoad(option) {

// 查询数据

this.id = option.id this.loadData()

},

methods: {

// 加载数据

loadData() {

this.getCourseById() this.getChapterSectionList() this.getCourseCommentList() this.getCourseGroupList()

},

// 查询课程基本信息

async getCourseById() {

const {data} = await api.getCourseById(this.id) this.course = data

// 将课程名称赋值导航标题

uni.setNavigationBarTitle({ title: data.title

})

},

// 查询章节列表数据

async getChapterSectionList() {

const {data} = await api.getChapterSectionList(this.id) this.chapterList = data

},

// 查询评论列表数据

async getCourseCommentList() {

const {data} = await api.getCourseCommentList(this.id) this.commentList = data

},

// 查询套餐搭配数据

async getCourseGroupList() {

const {data} = await api.getCourseGroupList(this.id) this.groupList = data

},

}

}

</script>

1. 课程详情页面模板代码

<template>

<view >

<!-- 主图和简介 +++++++++++++++++++++ -->

<course-header :course="course"></course-header>

<view class="course-details" :style="'height:' + pageHeight + 'px'">

<!-- 标签选项卡 -->

<tab-bar :tabs="tabs" v-model="tabIndex"></tab-bar>

<swiper class="swiper-box" circular :duration="300" :current="tabIndex" @change="changeTab" >

<swiper-item v-for="(item, index) in tabs" :key="index" >

<scroll-view id="scroll-box" class="scroll-box"

:scroll-y="enableScroll" upper-threshold="0" @scrolltoupper="scrolltoupper"

>

<view class="details-info">

<!-- +++++++++++++++++++++ -->

<course-info v-if="index==0"

:detailUrls="course.detailUrls"></course-info>

<course-dir v-if="index==1"

:chapterList="chapterList"></course-dir>

<course-comment v-if="index==2"

:commentList="commentList"></course-

comment>

<course-group v-if="index==3"

:groupList="groupList"></course-group>

</view>

</scroll-view>

</swiper-item>

</swiper>

<!-- 立即购买 -->

<buy-btn :courseId="id"></buy-btn>

</view>

</view>

</template>

**APP&小程序分享功能**

**APP端分享方式一**

1. APP端点击右上角分享按钮，分享到微信、QQ、微博等

分享参考：<https://uniapp.dcloud.io/api/plugins/share?id=sharewithsystem>



/pages/course/course-details.vue 中监听导航按钮点击事件如下：

// app端分享onNavigationBarButtonTap(e) {

if(e.type === 'share'){

// app端 分享, 先保存本地，再分享

uni.downloadFile({

url: this.course.mainImage, success: (res) => {

// 网络图片临时存储本地,本次启动期间可以正常使用

// console.log(res.tempFilePath, this.course.title) if (res.statusCode === 200) {

uni.shareWithSystem({

summary: this.course.title,

href: 'https://www.mengxuegu.com',

imageUrl: res.tempFilePath, // android使用本地图片type: 'image',

success(e) {

uni.showToast({

title: '分享成功'

})

}

})

}

}

});

}

**注意事项**

Android端当msg参数中设置图片（ imageUrl 属性）时，分享类型自动变为为

text

，在分享时可能只

image

会发送图片（如微信）；没有设置图片时分享类型则认为是文本 。

href

iOS端不同的分享程序对分享内容有要求，如微信分享时必需添加链接地址注：iOS8.0及以上系统触发成功回调则表示发送消息成功。

1. 完善 href 属性，它用于点击分享内容打开的链接，我们将它设置为 h5 端的访问地址在 utils.js 中有一个routePath方法：当前路由地址和参数

，否则微信分享失败。

/\*\*

\* 获取当前页面的路由地址+参数

\*/

export const routePath = () => {

// 获取所有页面对象

const pages = getCurrentPages();

// 获取当前页面实例

const page = pages[pages.length - 1];

// 当前路由参数对象

// 在小程序或app中，使用 page.options ; H5端使用 page.$route.query

const query = page.options || page.$route.query;

// 将参数对象转换为 key1=value1&key2=value2形式let queryStr = '';

query && Object.keys(query).forEach((key)=>{ queryStr += `${key}=${query[key]}`

});

// 当前路由地址+参数

return queryStr ? `${page.route}?${queryStr}`: page.route;

}

创建 /conﬁg/env.js 环境配置文件，配置H5端部署域名。

// 环境配置

// h端访问主机名，注意最后必须加上 /#/

let HOST\_H5 = ['https://www.mengxuegu.com/#/'](http://www.mengxuegu.com/%23/%27) if(process.env.NODE\_ENV === 'development'){

console.log('开发环境')

HOST\_H5 = 'https://static-3f6cc99f-e041-4662-9069-5c1175816bf6.bspapp.com/#/'

}else{

console.log('生产环境')

}

export {

HOST\_H5

}

在 main.js 中将 导出的属性，全部挂载到Vue实例上，方便后续访问

/config/env.js

import \* as env from './config/env.js'

// 挂载到vue原型上,访问 this.$env.HOST\_H5 Vue.prototype.$env = env

完善app端分享的href:

this.$env.HOST\_H5 + this.$util.routePath()

uni.shareWithSystem({

summary: this.course.title,

href: this.$env.HOST\_H5 + this.$util.routePath(), imageUrl: res.tempFilePath, // android使用本地图片type: 'image',

success(e) {

uni.showToast({ title: '分享成功'

})

}

})

**如果打开不了，检查 href 的地址是否有效，**

**APP端分享方式二**

因为 上面有兼容性问题，所以采用下面自定义方式 uni.share 方法

1. 创建分享组件 /components/mxg-share/mxg-share.vue，编写分享模板代码

<template>

<view>

<!-- 遮罩蒙层-->

<view class="mask" v-show="isShow" @click="showHandler" @touchmove.stop.prevent=" ()=>{}"></view>

<view class="share-body" v-show="isShow">

<scroll-view class="share-scroll noScorll" scroll-x>

<view class="share-item" >

<image src="/static/share/weixin.png"></image>

<view>微信好友</view>

</view>

<view class="share-item" >

<image src="/static/share/pengyouquan.png"></image>

<view>微信朋友圈</view>

</view>

<view class="share-item" >

<image src="/static/share/weibo.png"></image>

<view>新浪微博</view>

</view>

<view class="share-item" >

<image src="/static/share/qq.png"></image>

<view>QQ好友</view>

</view>

<view class="share-item" >

<image src="/static/share/link.png"></image>

<view>复制链接</view>

</view>

</scroll-view>

<view class="share-cancel" @click="showHandler">取消</view>

</view>

</view>

</template>

1. 编写样式代码

<style lang="scss">

/\*遮罩蒙层\*/

.mask {

z-index: 99; position: fixed; top: 0;

left: 0;

right: 0;

bottom: 0;

background-color: rgba(0,0,0,0.6);

}

.share-body {

position: fixed; z-index: 100;

left: 0;

right: 0;

bottom: 0;

.share-scroll{

background-color: #F7F7F7; width: 100%;

height: 200rpx;

display: flex;

// 一行显示不换行white-space: nowrap; padding-top: 45rpx;

.share-item {

width: 25%;

display: inline-flex; flex-direction: column; justify-content: center; align-items: center; image {

width: 60rpx; height: 60rpx; border-radius: 100%;

}

view{

color: $mxg-text-color-grey; font-size: 25rpx;

padding-top: 10rpx;

}

}

}

.share-cancel { padding: 25rpx 0;

width: 100%;

text-align: center; background-color: #FFFFFF;

}

}

</style>

1. 编写js业务逻辑代码,显示和隐藏分享分享组件

<script>

export default { data() {

return {

isShow: false, //是否显示

}

},

methods: {

// 显 示

showHandler() {

this.isShow = !this.isShow

},

}

}

</script>

1. 在 course-details.vue 页面模板直接使用 ，注意要加上 ref="share"。

mxg-share

点击导航按钮页面钩子中调用mxg-share组件中的显示隐藏方法mxg-share，

<mxg-share ref="share"></mxg-share>

// app端分享onNavigationBarButtonTap(e) {

if(e.type === 'share'){ this.$refs.share.showHandler()

}

},

1. 参考 hello-uni-app 项目中的分享组件 /hello-uni-app/pages/API/share/share.vue 的 data/onLoad/method

选项中的内容

完善 组件分享逻辑

mxg-share.vue

完善 data 选项, **shareType: 0 分享图文**

data() {

return {

isShow: false, //是否显示

title: '在线教育APP实战课程', // 分享标题shareText: ' -陪你学习，伴你梦想', // 分享内容href: 'https://www.mengxuegu.com', // 用访问链接

image: 'https://gd4.alicdn.com/imgextra/i4/3603079088/O1CN01dczOSM2H0LvTowhkl\_!!3603079088.png

', // 分享图片

shareType: 0, // 分享的类型 0 图文，1 文字，2 图片

providerList: [] // 提供商（上面模板中展示出来的第3方）

}

},

添加组件生命钩子 created , 不能声明 onLoad，因为钩子；

mxg-share.vue

将 hello-uni-app 项目中的onLoad代码，拷贝到 created 中，

是子组件，不是页面，没有页面

拷贝后，在每个data.push对象中 修改name名称，和添加对应提供商图标icon，用于模板中渲染在for循环外面，data.push 分享链接信息

created: function () {

// 获取提供商

uni.getProvider({

service: 'share', // 分享服务

success: (e) => { console.log('success', e);

let data = [] // 封装提供商

for (let i = 0; i < e.provider.length; i++) { switch (e.provider[i]) {

case 'weixin':

data.push({

name: '分享到微信好友', id: 'weixin',

sort: 0,

icon: '/static/share/weixin.png'

})

data.push({

name: '分享到微信朋友圈', id: 'weixin', type:'WXSenceTimeline', sort:1,

icon: '/static/share/pengyouquan.png'

})

break;

case 'sinaweibo': data.push({

name: '分享到新浪微博',

id: 'sinaweibo', sort:2,

icon: '/static/share/weibo.png'

})

break; case 'qq':

data.push({

name: '分享到QQ',

id: 'qq', sort:3,

icon: '/static/share/qq.png'

})

break; default:

break;

}

}

// 追加复制链接

data.push({

name: '复制链接',

id: 'copy', sort: 4,

icon: '/static/share/link.png'

})

this.providerList = data.sort((x,y) => { return x.sort - y.sort

});

},

fail: (e) => {

console.log('获取分享通道失败', e); uni.showModal({

content:'获取分享通道失败',

showCancel:false

})

}

});

},

1. 完善 mxg-share.vue 模板动信息态渲染提供商，和绑定分享点击事件@click="share(item)"

<template>

<view>

<!-- 遮罩蒙层-->

<view class="mask" v-show="isShow" @click="showHandler" @touchmove.stop.prevent=" ()=>{}"></view>

<view class="share-body" v-show="isShow">

<scroll-view class="share-scroll noScorll" scroll-x>

<view class="share-item"

v-for="(item, index) in providerList" :key="index" @click="share(item)"

>

<image :src="item.icon"></image>

<view>{{ item.name }}</view>

</view>

</scroll-view>

<view class="share-cancel">取消</view>

</view>

</view>

</template>

1. 将 hello-uni-app 项目中的分享动作 share 方法和IOS需要压缩图片使用的 compress 方法复制到 mxg-share

的methods 下：

复制后，share方法体前后分别加上加载中，隐藏加载中效果

async share(e) {

console.log('分享通道:'+ e.id +'； 分享类型:' + this.shareType);

// 分享中

uni.showLoading()

// ，，，

// 隐藏加载中

uni.hideLoading()

// 开始分享

uni.share(shareOPtions);

},

修改 share 方法 中的 switch 方法 中静态文字改为动态属性传递

case 0:

case 0:

shareOPtions.summary = this.shareText; shareOPtions.imageUrl = this.image; shareOPtions.title = this.title; shareOPtions.href = this.href;

break; case 1:

关注 share 方法中

if(shareOPtions.type === 0 && plus.os.name === 'iOS'){

调用 compress 方法压缩图片。

IOS端图文分享需要

但是 compress 方法是用于压缩本地图片，而不是网络图片 http，我们需要将网络图片下载到本地后再压缩，所以下面封装一个下载图片文件方法

// 下载图片文件

downFile(url) {

return new Promise((resolve) => { uni.downloadFile({

url,

success: (res) => {

// 网络图片临时存储本地,本次启动期间可以正常使用

//console.log('tempFilePath', res.tempFilePath) resolve(res.tempFilePath)

}

})

})

}

在 compress 方法 中调用 downFile，细节如下：

compress(){//压缩图片 图文分享要求分享图片大小不能超过20Kb console.log('开始压缩');

let img = this.image;

return new Promise( async (res) => {

// var localPath = plus.io.convertAbsoluteFileSystem(img.replace('file://', ''));

// console.log('after' + localPath); if(img.startsWith('http')) {

// 下载网络图片

img = await this.downFile(img)

}

// 压缩size

plus.io.resolveLocalFileSystemURL(img, (entry) => {

// plus.io.resolveLocalFileSystemURL(localPath, (entry) => {

测试分享效果， 模板上绑定@click="share(item)"

<view class="share-item" @click="share(item)"

1. 在 course-details.vue 页面向 mxg-share.vue 子组件传递数据course

<mxg-share ref="share" :shareData="course"></mxg-share>

1. mxg-share.vue 子组件接收父组件传递的分享数据

<script>

export default { props: {

shareData: Object,

},

watch: {

shareData(newVal) {

this.image = newVal.mainImage this.title = newVal.title

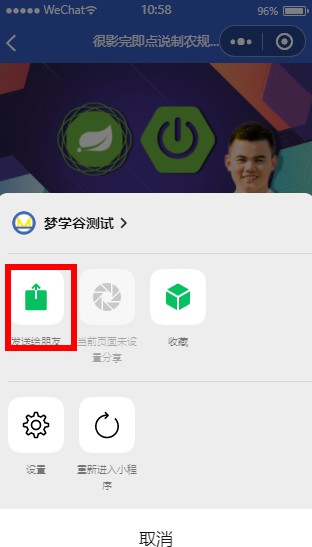
this.href = this.$env.HOST\_H5 + this.$util.routePath()

}

},

**小程序端分享**

1. 小程序端点击右上角胶囊按钮，弹出窗口点击分享按钮可分享给好友、群

参考：<https://uniapp.dcloud.io/api/plugins/share?id=onshareappmessage>

/pages/course/course-details.vue 中监听小程序端分享事件

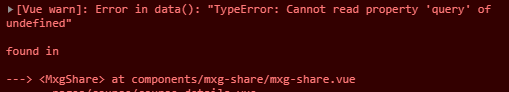
// 小程序端分享给好友（与onLoad同级）

onShareAppMessage(res) { return {

title: this.course.title, path: this.$util.routePath()

}

},

不能在data选项中直接获取当前页面路由，因为在小程序端页面获取路由参数会有问题

watch: {

shareData(newVal) {

this.image = newVal.mainImage this.title = newVal.title

// 监听器赋值

this.href = this.$env.HOST\_H5 + this.$util.routePath()

}

},

**复制链接**

1. 在 /components/mxg-share/mxg-share.vue 的share方法中添加复制逻辑

参考：<https://uniapp.dcloud.io/api/system/clipboard?id=setclipboarddata>

uni.setClipboardData

async share(e) {

console.log('分享通道:'+ e.id +'； 分享类型:' + this.shareType);

// 复制链接

if(e.id === 'copy') { uni.setClipboardData({

data: this.href, success:()=>{

// 隐藏默认提示

uni.hideToast() this.$util.msg('已复制到剪贴板')

}

})

// 隐 藏this.showHandler() return;

}

//...

}

**实现课程是否已购买**

需求：课程详情页底部按钮动态显示： 立即购买 或 立即观看。当已购买或免费课程时显示：**立即观看**，否则显示：立即购买。下面查看当前登录用户是否已经购买该课程。

**创建数据接口**

1. EasyMock 创建查询课程是否已购买接口描述：查询课程是否已购买

URL：

/course/course/is-buy/{courseId}

请求方式：

GET

mock.js 语法：

{

"code": 20000,

"message": "查询成功", "data": {

"isBuy|+1": true //是否购买, 1次true,一次false

}

}

1. 在课程的api文件 封装请求方法

/api/course.js

// 查询课程是否已购买

getCourseIsBuy(courseId) { return request({

url: `/course/course/is-buy/${courseId}`, method: 'GET'

})

},

**实现业务逻辑**

1. 详情页course-details.vue调用接口

data() {

return {

isBuy: false, // 是否购买+++

}

},

methods: {

// 加载数据

loadData() {

//查询课程是否已购买 +++ this.getCourseIsBuy()

this.getCourseById() this.getChapterSectionList() this.getCourseCommontList() this.getCourseGroupList()

},

//查询课程是否已购买

async getCourseIsBuy() {

// 如果已登录，则判断是否已购买

const token = uni.getStorageSync('mxgEducationToken')

if(token) {

const {data} = await api.getCourseIsBuy(this.id) this.isBuy = data.isBuy

}

},

},

1. 页面底部按钮判断显示按钮文字，绑定点击按钮触发的事件，和声明clickBottom方法

<bottom-btn :btnText="isBuy||course.isFree ? '立即观看': '立即购买'" @clickBottom="clickBottom" ></bottom-btn>

// 点击底部按钮触发

clickBottom(){

if(this.isBuy || this.course.isFree) {

// 跳 转 已 购 买 视 频 播 放 页 ( 后 面 创 建 页 面 ) this.navTo(`/pages/course/course-play?id=${this.id}`)

}else {

// 跳转确认购买页

}

}

1. 在bottom-btn组件中触发父组件的事件

clickHandler() {

// 点击按钮触发事件

this.$emit('clickBottom')

}

1. course-header.vue 课程基本信息价格显示处，如果是免费课程则显示**免费**

<view class="price-info" >

<text v-if="course.isFree">免费</text>

<block v-else>

<!-- 优 惠 价 -->

<text v-if="course.priceDiscount">￥{{course.priceDiscount}}</text>

<!--原价一直显示（有优惠价时显示price类名）-->

<text :class="{price: course.priceDiscount}">￥{{course.priceOriginal}}</text>

<text v-if="course.priceDiscount" class="youhui">优惠价</text>

</block>

</view>

**实现试看视频功能**

点击课程的章节列表中**试看**按钮，弹出视频播放窗口，如下图：



视频播放组件参考：<https://uniapp.dcloud.io/component/video>

1. 在 course-details.vue 添加播放模板代码：

<!-- 试 看 -->

<view v-if="videoUrl" @click="closePlay" class="mask video-box" @touchmove.stop.prevent="()=>{}">

<view class="name">

<text>免费试看</text>

<text class="iconfont icon-close"></text>

</view>

<video id="playVideo" class="video" :src="videoUrl">

</video>

</view>

mask 类名对应蒙层样式声明在 common.css 中，如下：

/\*遮罩蒙层\*/

.mask {

z-index: 99; position: fixed; top: 0;

left: 0;

right: 0;

bottom: 0;

background-color: rgba(0,0,0,0.7);

}

1. 在 course-details.vue 添加如下样式：

.video-box {

z-index: 100;

text-align: center;

.video {

width: 750rpx; height: 420rpx; position: absolute; top: 50%;

left: 50%;

transform: translate(-50%,-50%);

}

.name {

position: relative; top: 380rpx; color: #FFFFFF; font-size: 38rpx; font-weight: bold;

}

.icon-close {

margin-left: 20rpx;

}

}

1. 在 course-details.vue 编写试看逻辑

data() {

return {

videoContext: null,//播放器实例videoUrl: null // 播放视频

}

},

onReady() {

// 获取video实例, 注意 video 组件上声明 id="playVideo"

this.videoContext = uni.createVideoContext('playVideo', this)

},

methods: {

// 试看视频

playVideo(obj) { if(this.isBuy) {

// 已购买视频播放页

this.navTo(`/pages/course/course-play?id=${this.courseId}`) return

}

this.videoUrl = obj.section.videoUrl this.$nextTick(() => { // 箭头函数

this.videoContext.play()

})

},

// 关闭播放弹窗

closePlay() { this.videoContext.stop() this.videoUrl = null

}

}

1. 在 course-details.vue 模板的 组件元素上，绑定是否购买，试看视频事件

course-dir

@playVideo="playVideo"

<course-dir v-if="index==1" :chapterList="chapterList" :isBuy="isBuy" @playVideo="playVideo"></course-dir>

1. 在 章节列表组件的模板中，针对节行绑定点击事件，当**免费且未购买**时显示**试看**

course-dir.vue

<view class="sections row"

v-for="(section, index2) in chapter.sectionList"

:key="index2"

@click="playVideo(index, index2, section)"

>

<text class="see" v-if="section.isFree && !isBuy">试看</text>

</view>

props 接收isBuy属性，methods选项中添加 playVideo 方法，触发父组件的播放事件

props: {

isBuy: Boolean, // 是否购买

},

methods: {

playVideo(chapterIndex, sectionIndex, section) {

// console.log('section.isFree',section)

// 已经购买会跳转到视频播放页

if(section.isFree || this.isBuy) {

//传递

this.$emit('playVideo', {section: section} )

} else {

this.$util.msg('请先购买')

}

}

}

检查EasyMock上查询章节的数据接口否有 videoUrl 属性，如果没有则加上它

/course/api/chapter/section/list/{id}

中，sectionList数组元素是

{

"code": 20000, "message": "查询成功",

"data|2": [{ // 章

"id": 10,

"name": '@ctitle', // 7到15个字"sectionList|3": [ //课节信息

{

"id|+1": 1,

"name": '@ctitle(7, 15)', // 7到15个字"isFree|1": [0, 1], // 是否付费（0付费，1免费） "videoUrl|+1": [

['http://admin.mengxuegu.com/video/hls/git.m3u8'](http://admin.mengxuegu.com/video/hls/git.m3u8%27), ['http://baobab.kaiyanapp.com/api/v1/playUrl?](http://baobab.kaiyanapp.com/api/v1/playUrl)

vid=164016&resourceType=video&editionType=low&source=aliyun&playUrlType=url\_oss'

]

}

]

}]

}

试看按钮在购买后会隐藏掉，让整套都可以看，后面会完成

**高亮显示正在播放章节**

在 /pages/course/components/course-dir.vue 组件完成以下步骤

1. 添加一个亮度样式

.active {

color: $mxg-text-color-blue;

}

1. 在模板中渲染第几节元素上引用样式

<!-- 第 几 节 -->

<view class="sections row"

v-for="(section, index2) in chapter.sectionList"

:key="index2"

@click="playVideo(index, index2, section)"

:class="{active: activeObj.chapterIndex === index && activeObj.sectionIndex === index2}"

>

1. js声明activeObj当前播放章节，使用字符串拼接： **后面播放列表页面会切换章节播放时需要传递**

第几章下标，第几节下标

props: {

isBuy: Boolean, // 是否购买

activeObj: { // 当前所在章节

type: Object, default: ()=>({

chapterIndex: -1, //章下标

sectionIndex: -1 //节下标

})

},

},

methods: {

playVideo(chapterIndex, sectionIndex, section) {

// console.log('section.isFree',section) if(section.isFree || this.isBuy) {

this.activeObj.chapterIndex = chapterIndex this.activeObj.sectionIndex = sectionIndex

// 传 递

this.$emit('playVideo', {section, activeObj: this.activeObj} )

} else {

this.$util.msg('请先购买')

}

}

}

1. 上面微信小程序中无法高亮，原因是微信小程序中能在子组件修改父组件传递的对象值，页面不会更新。 要将下标对象更新的值传递给父组件，然后再通过父组件传递回子组件.

下面在/pages/course/course-details.vue组件中完成

data 声明 activeObj，playVideo 方法中赋值给activeObj

data() {

return {

activeObj: {}, //当前所选视频下标对象 ++++

}

},

methods: {

// 试看视频

playVideo(obj) {

if(this.isBuy) {

// 已购买视频播放页

this.navTo(`/pages/course/course-play?id=${this.courseId}`) return

}

this.activeObj = obj.activeObj // ++++ this.videoUrl = obj.section.videoUrl this.$nextTick(() => {

this.videoContext.play()

})

},

}

course-dir 组件元素上绑定:activeObj="activeObj"

<course-dir v-if="index==1" :chapterList="chapterList" @playVideo="playVideo"

:activeObj="activeObj"></course-dir>

这样就可以解决微信小程序不高亮问题

**subNVue原生子窗体实现视频播放**

**视频播放一定要使用 HBuilderX2.9.3版本，不然无法正常退出全屏**



nvue 页面参考：

<https://uniapp.dcloud.io/nvue-outline> subNVue参考：

<https://uniapp.dcloud.io/collocation/pages?id=app-subnvues>

<https://ask.dcloud.net.cn/article/35948>

自定义视频播放参考：

<http://www.html5plus.org/doc/zh_cn/video.html>

**[样式注意事项](https://uniapp.dcloud.io/nvue-css?id=%e6%b3%a8%e6%84%8f%e4%ba%8b%e9%a1%b9)**

nvue的css**仅支持ﬂex布局**，是webview的css语法的子集。这是因为操作系统原生排版不支持非ﬂex之 外的web布局。当然ﬂex足以排布出各种页面，只是写法需要适应。

在选择器方面支持的较少，只支持简单的 。

class="classA"

class 进行绑定时只支持数组语法。不支持媒体查询

不支持复合样式，不支持简写不能在 style 中引入字体文件

布局不能使用百分比，如 ；

width：100%

有些web的css属性在nvue里无法支持，比如背景图。但可以使用image组件和层级来实现类似web中的 背景效果。因为原生开发本身也没有web这种背景图概念

nvue 的各组件在安卓端默认是透明的，如果不设置 ，可能会导致出现重影的问题

background-color

text

text

文字内容，必须只能在

组件下，

组件不能换行写内容，否则会出现无法去除的周边空白

只有 text 标签可以设置字体大小，字体颜色

## 创建与配置视频播放页面

1. 创建视频播放页面

/pages/course/course-play.vue

2.

/pages/course/course-details.vue

组件中的进行跳转到视频播放页面

// 点击底部按钮触发

clickBottom() {

if(this.isBuy || this.course.isFree) {

// 已购买过或免费，跳转视频播放页面

this.navTo(`/pages/course/course-play?id=${this.id}`)

}else {

// 未购买过，跳转确认购买页

}

}

1. 在 pages.json 配置course-play.vue 和对应的视频播放原生子窗体

,{

"path" : "pages/course/course-play",

"style" : {

"app-plus": {

"titleNView": false, //禁用原生导航栏"subNVues": [{ // 原生子窗体

"id": "top", // 唯一标识

"path": "pages/course/subNVue/video", // nvue文件路径"style":{ // 样 式

//在父容器中的定位方式，dock页面中停靠在dock属性值位置

"position": "dock",

"dock": "top", // top 停靠顶部，

"width": "100%",

"height": "245px"

}

}]

}

}

}

1. 创建视频播放原生子窗体文件 /pages/course/subNVue/video**.nvue** , 基础模板如下

<template>

<view class="video-box">

<view :style="{ height: statusBarHeight + 'rpx'}"></view>

<!--

poster 视频封面的图片网络资源地址

src 要播放视频的资源地址

show-center-play-btn 是否显示视频中间的播放按钮

show-fullscreen-btn 是否显示全屏按钮

show-progress 是否显示进度条

:controls="false" 取消默认控制按钮@timeupdate 播放进度变化时触发@ended 当播放到末尾时触发

-->

<video

ref="video" class="video"

:poster="videoMedia.mainImage"

:src="videoMedia.videoUrl"

:controls="false"

:show-center-play-btn="false"

:show-fullscreen-btn="false"

:show-progress="false" @timeupdate="timeupdate" @ended="nextPlay"

>

</video>

</view>

</template>

1. 声明 data 属性和事件方法、获取视频实例

<script>

// 当前video实例

let videoContext = null

export default { data() {

return {

statusBarHeight: 0, // 状态栏高度

videoMedia: { id: 1,

title: '01\_项目需求分析项目需求分析',

videoUrl: ["http://baobab.kaiyanapp.com/api/v1/playUrl?](http://baobab.kaiyanapp.com/api/v1/playUrl) vid=164016&resourceType=video&editionType=low&source=aliyun&playUrlType=url\_oss",

mainImage: "https://gd4.alicdn.com/imgextra/i4/3603079088/O1CN01dczOSM2H0LvTowhkl\_!!3603079088.png",

},

}

},

onLoad() {

// 获取状态栏高度

this.statusBarHeight = uni.getSystemInfoSync().statusBarHeight

},

mounted() {

// 注意，在 subNvue中要使用这种方式获取video实例videoContext = this.$refs.video

},

methods: {

// 播放进度变化时触发，获取视频时长、当前播放时长

timeupdate(event){

},

// 播放到末尾，切换下一节

nextPlay() {

}

}

}

</script>

1. 添加样式, 注意不支持scss，只支持ﬂex布局且默认ﬂex布局（不要写display:ﬂex)

<style>

.video {

width: 750rpx; height: 225px;

}

</style>

**加载自定义字体图标**

参 考 ：<https://uniapp.dcloud.io/nvue-api?id=addrule>注意：图标文件在发布时需要变成https://开头的

1. beforeCreate 生命钩子加载图标文件

beforeCreate() {

// nvue文件 加载图标，在style中声明 .ionfont样式const dom = weex.requireModule('dom'); dom.addRule('fontFace', {

fontFamily: 'iconfont',

// 推荐使用 ~@ 开头，发布时需要变成https://开头的

src: "url('~@/static/iconfont/iconfont.ttf')"

});

},

1. 在style中声明 .ionfont样式

<style>

.iconfont {

font-family: iconfont; /\* iconfont 对应上面 fontFamily 的值, \*/ font-size: 50rpx;

color: #FFFFFF;

}

.video {

width: 750rpx; height: 225px;

}

</style>

1. 采用 Unicode 方式使用图标

<text class="iconfont" >&#xe679;</text>

**自定义视频播放器视图**

要覆盖在vedio原生组件上的视图，推荐使用 cover-view 组件，

**头部后退&标题**

1. 编写模板

<video

ref="video" class="video"

:poster="videoMedia.mainImage"

:src="videoMedia.videoUrl"

:controls="false"

:show-center-play-btn="false"

:show-fullscreen-btn="false"

:show-progress="false" @timeupdate="timeupdate" @ended="nextPlay"

>

<!-- 覆盖在vedio原生组件上的文本视图。作用与普通view一样 -->

<cover-view class="controls-top row" ref="controlsTop" >

<!-- 如果图标没有效果，则删除了重新添加 -->

<text class="iconfont" @click.stop="backHandler">&#xe679;</text>

<text class="title">{{videoMedia.title}}</text>

</cover-view>

</video>

1. 编写样式

/\* 头部控制层 \*/

.controls-top { height: 90rpx; padding: 0 15rpx;

align-items: center;

background-image: linear-gradient(to top, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.8));

}

.title {

font-size: 30rpx; color: #FFFFFF; width: 500rpx;

/\*nvue 文件中，文本溢出显示省略号\*/

lines:1;

text-overflow: ellipsis;

}

1. 在 method 选项声明 @click="backHandler" 后退方法

methods: {

// 后 退

backHandler() {

},

}

**中间播放按钮**

1. 编写模板，在 video 元素体中

<!-- 中间区域：控制显示隐藏 -->

<cover-view class="controls-center center">

<text class="stop iconfont" @click.stop="playVideo">&#xe625;</text>

</cover-view>

1. 编写样式

/\* 中间区域 \*/

.controls-center {

/\* 占中间区域，后面需要点击控制显示隐藏\*/ position: absolute;

top: 90rpx; left: 0;

right: 0;

bottom: 90rpx;

/\* 透明背景占位，防止ios渐变有问题 \*/ background-color: rgba(0, 0, 0, 0);

/\*

border-width: 1px; border-style: solid; border-color: red;

\*/

}

.stop {

padding: 20rpx 35rpx;

background-color: rgba(0, 0, 0, 0.6); border-radius: 10px;

}

1. 在 method 选项声明 @click="playVideo" 播放暂停视频

methods: {

// 播放视频

playVideo() {

},

}

**底部播放&暂停&时长&倍速&全屏等按钮**

1. 编写模板，在 video 元素体中

<!-- 底 部 -->

<cover-view class="controls-bottom row center" >

<view class="row">

<!-- 播 放 -->

<text @click="playVideo" class="video-btn iconfont">&#xe625;</text>

<!-- 暂 停 -->

<!-- <text @click="playVideo" class="video-btn iconfont">&#xe62c;

</text> -->

<!-- 下 一 集 -->

<text class="video-btn iconfont">&#xe61b;</text>

</view>

<!-- 当前播放时长 -->

<text class="time">00:00</text>

<view class="bottom-slider">

<!-- 当前观看所在时长 -->

<!-- 进 度 条 -->

</view>

<!-- 总 时 长 -->

<text class="time">88:01</text>

<view class="row">

<!-- 倍 速 -->

<text class="video-btn">1.0x</text>

<!-- 全 屏 -->

<text class="video-btn iconfont">&#xe623;</text>

</view>

</cover-view>

1. 编写样式

/\* 底部控制 \*/

.controls-bottom { position: absolute; bottom: 0;

width: 750rpx; height: 90rpx;

background-image: linear-gradient(to bottom, rgba(0, 0, 0, 0), rgba(0, 0, 0, 0.8));

}

.video-btn {

width: 80rpx;

line-height: 90rpx; text-align: center; color: #fff;

font-size: 30rpx;

}

.bottom-slider {

flex-direction: row; align-items: center; height: 90rpx;

/\* 左右固定了宽度，中间占满剩余宽度 \*/

flex: 1;

text-align: center;

}

.time {

font-size: 28rpx; color: #FFFFFF; width: 110rpx; text-align: center;

}

**播放进度条**

1. 创建播放进度条子组件：

pages/course/subNVue/components/slider-bar.nvue

<template>

<view>

<view class="slider row center" :style="{ width: `${sliderWidth}px`}" @touchstart="blockTouchStart" @touchend="blockTouchEnd"

@touchmove.stop="blockTouchMove"

>

</view>

<!-- 圆点左侧已播放进度条 -->

<view class="slider-left" :style="{ width: `${leftWidth}px` }" ></view>

<!-- 圆点右侧未播放进度条 -->

<view class="slider-right" :style="{ width: `${sliderWidth-leftWidth}px` }">

<!-- 圆 点 -->

<view class="block-inner" :style="{left: `${leftWidth-5}px`}"></view>

</view>

</view>

</template>

<script>

export default{ props:{

totalTime:{ //总时长

type: Number, default: 60

},

currentTime: { // 当前播放时长

type:Number, default: 0

},

sliderWidth: { //进度条宽度type:Number,

default: 100

},

ratio: { //倍速type: Number, default: 1

},

direction:{ //判断屏幕方向，非全屏screenX，全屏screenY，用于计算滑动长度type:String,

default: 'screenX'

},

},

data(){

return{

oldToucesX: 0,

leftWidth: 0,

oldLeftWidth: 0, touch: false

}

},

methods:{

// 触摸开始，记录原坐标

blockTouchStart(e) { this.touch = true

this.oldLeftWidth = this.leftWidth

// screenX 点击点相对于屏幕左侧边缘的 X 轴坐标, screenY 点击点相对于屏幕顶部边缘

的 Y 轴 坐 标

},

this.oldToucesX = e.changedTouches[0][this.direction];

// 计算方向

blockTouchMove(e) {

let newToucesX = e.changedTouches[0][this.direction]; this.leftWidth = (newToucesX - this.oldToucesX) \* this.ratio +

this.oldLeftWidth

// 起出宽度，则移动进度条到右边缘，

this.leftWidth = this.leftWidth > this.sliderWidth ? this.sliderWidth :

this.leftWidth

},

// 小于0，则移动进度条到左边缘，

this.leftWidth = this.leftWidth < 0 ? 0 : this.leftWidth

// 结束触摸

blockTouchEnd(e) {

// 计算结束点，触发父组件，进行结束点位置播放

let current = this.leftWidth / this.sliderWidth \* this.totalTime const event = {detail:{value: current}}

this.$emit('change', event) this.touch = false

},

},

watch:{

// 监听播放时间变化，移动滚动点

currentTime:{

immediate:true, handler(newVal, oldVal){

if(this.touch) return

this.leftWidth = newVal/this.totalTime \* this.sliderWidth this.leftWidth = this.leftWidth > this.sliderWidth? this.sliderWidth :

this.leftWidth

}

}

}

}

</script>

<style >

.slider {

height: 90rpx;

}

.slider-left{

/\* flex: 1; \*/ height: 3px;

border-radius: 10rpx; background-color: #345dc2;

}

.slider-right{ height: 3px;

background-color: rgba(175, 175, 175, 1); border-radius: 10rpx;

}

.block-inner{

position: absolute; background-color: #fff; height: 20rpx;

width: 20rpx;

border-radius: 100%;

z-index: 10;

}

</style>

1. 在 video.nvue 组件中导入并引用 slider-bar 组件

<view class="bottom-slider">

<!-- 当前观看所在时长 -->

<!-- 进 度 条 -->

<slider-bar />

</view>

<script>

// 当前video实例

let videoContext = null

// 进度条

import sliderBar from './components/slider-bar.nvue' export default {

components: {

sliderBar,

},

**播放器业务逻辑**

**播放暂停功能**

点击视频窗口中间按钮和左下角第1个按钮，触发视频播放暂停功能 @click="playVideo" data 中声明

data() {

return {

play: false, //播放状态showControls: true, // 控制层显示隐藏

}

},

在 playVideo 方法实现播放暂停, **timer 定时器实例声明在外面**

<script>

// 当前video实例

let videoContext = null

// 定时器关闭控制层

let timer = null export default {

methods: {

// 播放视频

playVideo() {

if(this.play){

// 暂 停

videoContext.pause()

}else{

// 播放视频

videoContext.play()

// 隐藏控制层

this.showHideControls()

}

this.play = !this.play

},

// 显示隐藏控制层

showHideControls() {

this.showControls = !this.showControls

// 定时隐藏控制层

if(this.showControls) {

//清除原有定时器

timer = timer ? clearTimeout(timer) : null

// 3秒后隐藏控制层

timer = setTimeout(()=> { this.showControls = false

}, 3000)

}

},

}

模板中判断显示隐藏和中间区域调用showHideControls。 注意：不能用v-show指令,要用v-if

<!-- 头部区域 不能用v-show -->

<cover-view class="controls-top row" ref="controlsTop" v-if="showControls" >

<!-- 中间区域 -->

<cover-view class="controls-center center" @click="showHideControls">

<text class="stop iconfont" @click.stop="playVideo" v-if="!play && showControls">&#xe625;</text>

</cover-view>

<!-- 底 部 -->

<cover-view class="controls-bottom row center" v-if="showControls">

**获取视频时长功能**

1. 获取当前时长和总时长（单位：秒），通过 video 组件的 @timeupdate="timeupdate" 事件方法实现

data 中声明duration、current

data() {

return {

duration: 1, // 视频总时长（秒），默认1秒显示进度条

current: 0, //当前播放时长（秒）

}

},

获取时长,开始播放后会触发获取

methods: {

// 播放进度变化时触发，获取视频时长、当前播放时长

timeupdate(event){

//console.log('视频总时长', event.detail.duration)

//console.log('当前播放时长', event.detail.currentTime)

this.duration = event.detail.duration this.current = event.detail.currentTime

},

}

定义计算属性，格式化秒转为 分钟:秒，用于模板渲染

computed:{

// 总时长 分钟:秒

durationStr(){

return this.formatSeconds(this.duration)

},

// 当前时长 分钟:秒

currentStr(){

return this.formatSeconds(this.current)

},

},

methods: {

/\*\*

* 格式化时间 分钟:秒
* @param {Object} totalSecond 总秒数

\*/

formatSeconds (totalSecond) {

let minute = parseInt(totalSecond/60); if(minute<10) minute = "0" + minute; let second = parseInt(totalSecond%60); if(second<10) second = "0" + second; let time = minute + ":" + second; if(totalSecond>=0){

return time;

}else{

return "00:00";

}

},

}

模板中渲染格式化后的值

<!-- 当前播放时长 -->

<text class="time">{{currentStr}}</text>

<view class="bottom-slider">

<!-- 当前观看所在时长 -->

<!-- 进 度 条 -->

<slider-bar />

</view>

<!-- 总 时 长 -->

<text class="time">{{durationStr}}</text>

**全屏功能**

点击视频窗口右下角按钮，触发全屏功能 @click="fullScreenHandler" data 中声明

data() {

return {

fullScreen: false, // 全屏状态controlsWidth: 0,//全屏宽controlsHeight: 0,//全屏高

}

},

在 fullScreenHandler 方法实现全屏功能

methods: {

// 点击全屏

fullScreenHandler() { if(this.fullScreen){

//退出全屏

videoContext.exitFullScreen()

}else{

// 进入全屏

videoContext.requestFullScreen()

}

// 改变状态

this.fullScreen = !this.fullScreen

},

}

绑定全屏 图标绑定点击事件 @click="fullScreenHandler"

<!-- 全 屏 -->

<text class="video-btn iconfont" @click.stop="fullScreenHandler">&#xe623;</text>

演示效果，宽度无法自动填满整屏，下面进行优化

添加一个针对 fullScreen 全屏状态的监听器，动态获取屏幕宽高

onLoad() {

const res = uni.getSystemInfoSync()

// 获取状态栏高度

this.statusBarHeight = res.statusBarHeight

// 屏幕宽高

this.controlsHeight = res.screenHeight this.controlsWidth = res.screenWidth

},

watch: {

fullScreen(newVal, oldVal) {

// 获取当前的宽度。用于交换宽高，

const width = this.controlsWidth

this.controlsWidth = this.controlsHeight this.controlsHeight = width

},

},

在底部控制层的父元素 cover-view 上动态获取宽度

<!-- 底 部 -->

<cover-view class="controls-bottom row center"

v-if="showControls" :style="{width: `${controlsWidth}px`}">

**后退按钮功能**

点击头部后退按钮，非全屏状态：返回上一页，全屏状态：退出全屏

1. 在 backHandler 方法实现后退功能

methods: {

// 点击头部后退图标

backHandler() { if(this.fullScreen) {

// 全屏时，回到非全屏状态，

this.fullScreenHandler()

}else {

// 非全屏时，返回上一页面

uni.navigateBack()

}

},

}

**进度条动态宽度和拖动播放功能**

1. 使用 sliderWidth 计算属性，当屏幕宽度改变（旋转）动态计算进度条长度

computed:{

//进度条长度

sliderWidth(){

// rpx转px，320是底部4个按钮\*每个80rpx，220是时长各110rpx， return this.controlsWidth - uni.upx2px(320+220)

}

},

1. 向slider-bar组件传递属性值和绑定事件 @change="change" 拖动进度条触发事件

<view class="bottom-slider">

<!-- 进 度 条 -->

<slider-bar :sliderWidth="sliderWidth" :duration="duration" :current="current"

:direction="fullScreen ? 'screenY': 'screenX'" @change="change"

/>

</view>

1. 在 methods 中定义拖动进度条触发事件方法 change ，方法中实现切换指定时间点播放

// 拖动滑块

change(currentSecond) { this.current = currentSecond

//切换指定时间点播放

videoContext.seek(currentSecond)

},

**倍速播放功能**

点击 1.0x倍速按钮弹出选择倍速窗口，进行选择速率后，进行对应倍速播放注意：ios不支持1.25和1.5倍速

data 中声明

data() {

return {

rates: ['0.8', '1.0', '1.25', '1.5', '2.0'], // ios不支持1.25和1.5

currentRate: '1.0',// 当前倍速率

rateView: false, // 是否显示倍速视图

}

},

在模板中video组件元素体内添加倍速选择窗口模板

<!-- 加速列表 -->

<cover-view v-if="rateView" class="center"

:class="fullScreen ? 'rate-list-full': 'rate-list'"

:style="{height: fullScreen ? controlsHeight: null}" >

<!-- 注意active样式放最后，不然不生效，会被覆盖 -->

<text :class="{'rate-item-full' : fullScreen, 'rate-item': !fullScreen, 'active': rate

=== currentRate}"

v-for="(rate, index) in rates" :key="index" @click.stop="clickRate(rate)"

>{{rate}}x</text>

</cover-view>

编写样式

/\* 非全屏倍速样式 \*/

.rate-list{

position: absolute; bottom: 70rpx; right: 65rpx;

background-color: rgba(0,0,0, 0.7); width: 110rpx;

border-radius: 10rpx;

}

.rate-item{

color: #FFFFFF; font-size: 35rpx; line-height: 60rpx;

}

.active {

color: #345dc2;

}

/\* 全屏倍速样式 \*/

.rate-list-full { width: 400rpx;

background-color: rgba(0,0,0, 0.7); position: absolute;

right: 0;

top: 0;

bottom: 0;

}

.rate-item-full { color: #FFFFFF; font-size: 50rpx;

line-height: 130rpx;

}

在 methods 中定义点击倍速按钮事件方法播放事件方法 clickRate

showHideRateView

弹出倍速窗口，和定义点击某个倍速进行倍速

// 显示隐藏倍速窗口

showHideRateView() { this.rateView = !this.rateView

},

// 倍速播放

clickRate(rate) { this.currentRate = rate

// 设置倍速，注意转浮点数

videoContext.playbackRate( parseFloat(rate) )

// 关闭倍速选项弹层

this.showHideRateView()

},

模板中的倍速按钮绑定点击事件,动态显示当前速度

{{currentRate}}

<view class="row">

<!-- 倍 速 -->

<text class="video-btn" @click.stop="showHideRateView">{{currentRate}}x</text>

<!-- 全 屏 -->

<text class="video-btn iconfont" @click.stop="fullScreenHandler">&#xe623;</text>

</view>

显示隐藏控制层时，将倍速视频隐藏

// 显示隐藏控制层

showHideControls() {

this.showControls = !this.showControls

// 倍速视图

this.rateView = false

// 定时隐藏控制层

if(this.showControls) {

//清除原有定时器

timer = timer ? clearTimeout(timer) : null

// 3秒后隐藏控制层

timer = setTimeout(()=> { this.showControls = false

}, 3000)

}

},

向进度条传递当前倍速率

:rate="parseFloat(currentRate)

<slider-bar :duration="duration" :current="current"

:sliderWidth="sliderWidth" :direction="fullScreen ? 'screenY': 'screenX'" @change="change"

:rate="parseFloat(currentRate)"

>

</slider-bar>

**视频播放列表页面**



**静态页面效果实现**

在 /pages/course/course-play.vue 页面中完成以下操作

1. 编写模板

<template>

<view class="course-play">

<!-- #ifndef APP-PLUS -->

<video id="myVideo" style="width: 750rpx; height: 423rpx;"

:poster="poster"

:src="src"

>

</video>

<!-- #endif -->

<!-- 课程信息 -->

<view class="course-info space-between">

<text class="title">Uniapp兼容多端在线教育项目实战兼容多端在线教育项目实战</text>

<view class="right" @click="navTo(`/pages/course/course-details?id=${id}`)">

<text>详情</text>

<text class="iconfont icon-right"></text>

</view>

</view>

<course-dir :isBuy="true" ></course-dir>

<!-- 底部按钮 -->

<view class="bottom center">

<!-- #ifdef APP-PLUS -->

<view class="btn-item column one">

<text class="iconfont icon-Share-Outline"></text>

<text >分享</text>

</view>

<!-- #endif -->

<view class="btn-item column one">

<text class="iconfont icon-edit"></text>

<text>评价</text>

</view>

</view>

</view>

</template>

1. 编写样式

<style lang="scss">

.course-play {

padding-bottom: 100rpx;

}

.course-info { padding: 30rpx; font-weight: bold;

border-bottom: $mxg-underline;

.title {

width: 500rpx;

font-size: 35rpx;

line-height: 60rpx;

color: $mxg-text-color-black;

}

.right>text{

font-size: 28rpx; line-height: 60rpx;

color: $mxg-text-color-black;

}

}

.bottom {

position: fixed; left: 0;

right: 0;

bottom: 0; height: 100rpx;

background-color: #FFFFFF; border-top: $mxg-underline;

}

.btn-item {

text {

text-align: center;

color: $mxg-text-color-grey; font-size: 25rpx;

}

.iconfont {

font-size: 38rpx;

}

}

</style>

1. 编写js逻辑

<script>

import courseDir from './components/course-dir.vue' let videoContext = null

export default {

components: {courseDir},

data() {

return {

id: null, //课程id poster:

'https://gd4.alicdn.com/imgextra/i4/3603079088/O1CN01dczOSM2H0LvTowhkl\_!!3603079088.png', src: ['http://baobab.kaiyanapp.com/api/v1/playUrl?](http://baobab.kaiyanapp.com/api/v1/playUrl)

vid=164016&resourceType=video&editionType=low&source=aliyun&playUrlType=url\_oss',

}

},

onReady() {

// #ifndef APP-PLUS

videoContext = uni.createVideoContext('myVideo', this)

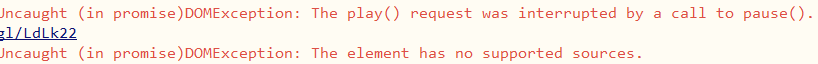
// console.log('videoContext', videoContext)

// #endif

},

}

</script>

1. 注意：当前 h5 端在浏览器中播放视频报如下错，无法播放, uniapp官方还在修复中

使用微信小程序开发工具测试。

**调用接口获取课程数据**

1. EasyMock 创建通过课程id查询已购买的视频列表接口描述：通过课程id查询已购买的视频列表

URL：

/course/course/buy/list/{id}

请求方式：

GET

mock.js 语法：

{

"code": 20000, "message": "查询成功",

"data|8-30": [{ // 章

"id": 10,

"name": '@ctitle', // 7到15个字"sectionList|3-18": [ //课节信息

{

"id|+1": 1,

"name": '@ctitle(7, 15)', // 7到15个字"isFree|1": [0, 1], // 是否付费（0付费，1免费） "videoUrl|+1": [

['http://admin.mengxuegu.com/video/hls/git.m3u8'](http://admin.mengxuegu.com/video/hls/git.m3u8%27), ['http://baobab.kaiyanapp.com/api/v1/playUrl?](http://baobab.kaiyanapp.com/api/v1/playUrl)

vid=164016&resourceType=video&editionType=low&source=aliyun&playUrlType=url\_oss'

]

}

]

}]

}

1. course.js 文件中封装API查询接口方法

// 通过课程id获取已经购买课程视频列表

getCourseBuyList(id) { return request({

url: `/course/course/buy/list/${id}`, method: 'GET'

})

},

1. 导入course.js接口API文件, 声明 data 属性，调用接口获取数据

<script>

import api from '@/api/course.js' export default {

data() {

return {

course: {}, chapterList: [],

activeObj: { //列表选中下标

chapterIndex: 0, // 章下标

sectionIndex: 0 // 节下标

}

}

},

onLoad(option) { this.id = option.id this.loadData()

},

methods: {

async loadData() {

// 查询课程基本信息

const {data: course} = await api.getCourseById(this.id) this.course = course

// 查询已购买课程视频列表信息

const {data: chapterList} = await api.getCourseBuyList(this.id) this.chapterList = chapterList

// #ifndef APP-PLUS

this.poster = this.course.mainImage

const chapter = this.chapterList[this.activeObj.chapterIndex]

const section = chapter && chapter.sectionList[this.activeObj.sectionIndex] this.src = section && section.videoUrl || ''

// #endif

},

// 切换视频播放

changeVideo(obj) {

}

1. 模板接收数据动态渲染

<!-- 课程信息 -->

<view class="course-info space-between">

<text class="title">{{course.title}}</text>

<view class="right" @click="navTo(`/pages/course/course-details?id=${id}`)">

<text>详情</text>

<text class="iconfont icon-right"></text>

</view>

</view>

<course-dir :activeObj="activeObj" :chapterList="chapterList" @playVideo="changeVideo" >

</course-dir>

**非APP端切换视频播放功能**

需求：点击课程列表中某一节课，切换到这节课进行播放视频

1. 在 changeVideo 方法中实现：

// 切换视频播放

changeVideo(obj) {

// 选中课程下标

this.activeObj = obj.activeObj

// #ifndef APP-PLUS videoContext.pause()

this.src = obj.section.videoUrl setTimeout(()=>{

videoContext.play()

}, 300)

// #endif

},

**APP 端动态播放视频**

需求：

在 course-play.vue 页面中初始化课程数据后，将数据传递给 video.nvue 原生子窗体页面。页面间通讯参考：<https://ask.dcloud.net.cn/article/36010>

uni.$on

uni.$off

uni.$once

使用 触发事件、

uni.$emit

监听事件 、

移除监听事件 ，

一次性的监听事

件 ，不需要移除 。可以方便的进行页面的通讯 ，触发的事件都是 App 全局级别的，跨任意组件，页面，

nvue，vue 等

1. 在 course-play.vue 页面的 loadData 方法中，当加载完数据，使用 触发 事件传递参数

uni.$emit

video

async loadData() {

// #ifdef APP-PLUS

// 传递给 video.nvue 组件

uni.$emit('video', { type: 'init', params: {

course: this.course, //基本信息

chapterList: this.chapterList, //课程列表activeObj: this.activeObj // 当前播放视频下标对象

}

})

// #endif

},

1. 在 video.nvue 页面的 onLoad 钩子中使用 监听 事件；在onUnload钩子中使

uni.$on

video

uni.$off

video

用 移除 监听事件。

onLoad() {

// 监听video事件， couse-play.vue 组件传递数据

uni.$on('video', this.videoHandlerEvent)

},

onUnload() {

// 移除监听事件

uni.$off('video')

},

1. 在 video.nvue 页面的 methods 选项中定义监听事件方法值。

videoHandlerEvent

，方法中进行视频播放动态赋

// 获取course-play.vue组件传递过来的数据

videoHandlerEvent(e) { switch(e.type) {

// 初始化数据

case 'init':

// 课程基本信息

const course = e.params.course this.videoMedia.id = course.id this.videoMedia.mainImage = course.mainImage

// 当前播放下标对象

this.activeObj = e.params.activeObj

// 课程列表

this.chapterList = e.params.chapterList

// 播放当前视频

const chapter = this.chapterList[this.activeObj.chapterIndex]

const section = chapter && chapter.sectionList[this.activeObj.sectionIndex] if(section) {

this.videoMedia.videoUrl = section.videoUrl this.videoMedia.title = section.name

}

break;

case 'change':

break;

}

},

**APP 端切换视频播放功能**

需求：APP 端点击课程列表中某一节课，切换到这节课进行播放视频

1. 在 course-play.vue 页面的changeVideo方法中，使用 触发

uni.$emit

事件传递当前选中课程参数

video

// 切换视频播放

changeVideo(obj) {

// #ifdef APP-PLUS

uni.$emit('video', { type: 'change', params: {

activeObj: obj.activeObj, section: obj.section

}

})

// #endif

},

1. 在 video.nvue 页面的

videoHandlerEvent

方法中的

下进行视频播放动态赋值。

case 'change':

// 获取course-play.vue组件传递过来的数据

videoHandlerEvent(e) { switch(e.type) {

// 初始化数据

case 'init':

break; case 'change':

this.activeObj = e.params.activeObj

// console.log('activeObj', this.activeObj, e.params.section)

// 播放视频this.playSection(e.params.section) break;

}

},

// 播放指定视频

playSection(section) {

if(section && section.videoUrl) {

//暂停videoContext.pause()

this.videoMedia.videoUrl = section.videoUrl this.videoMedia.title = section.name

// 播 放

setTimeout(()=> { this.play = true videoContext.play()

}, 300)

}else {

uni.showToast({

title: '视频资源不存在', icon: 'none'

})

}

},

**播放到末尾,切换下一集**

需求：当前视频播放结束后，或点击视频窗口 图标时，切换到下一集视频进行播放，



>|

获取当前章节的下一集，有下集，则播放下一集

当前章节没有下一集，查看是否有下一章， 有下章, 则播放下章第一节课没有下章节，则暂停视频，提示 **已观看完最后一节课**

1. 在 video.nvue 的 nextPlay 方法实现如下：

// 播放到末尾,切换下一节

nextPlay() {

// 获取当前章

let chapter = this.chapterList[this.activeObj.chapterIndex]

// 获取当前章节的下一集

let section = chapter && chapter.sectionList[this.activeObj.sectionIndex+1] if(section && section.videoUrl) {

// 1. 有下集，则播放下一集

console.log('播放下一集', this.activeObj.sectionIndex)

this.activeObj.sectionIndex++ this.playSection(section)

} else {

// 2. 没有下集，查看是否有下一章

chapter = this.chapterList[this.activeObj.chapterIndex+1] console.log(' 查 看 是 否 有 下 一 章 ', this.activeObj.chapterIndex) if(chapter && chapter.sectionList && chapter.sectionList.length>0) {

// 有下章,则获取下章第一节课this.activeObj.chapterIndex++ this.activeObj.sectionIndex = 0

section = chapter.sectionList[this.activeObj.sectionIndex]

// 2. 播放下章第一节课

this.playSection(section)

}else {

// 3. 没有下章节，则暂停videoContext.pause() this.play = false uni.showToast({

title: '已观看完最后一节课', icon: 'none'

})

}

}

},

1. 在 video.nvue 页面的 video 元素上绑定监听播放末尾事件，视频窗口 图标绑定点击事件



>|

<video @ended="nextPlay"

<!-- 下 一 集 -->

<text @click="nextPlay" class="video-btn iconfont">&#xe61b;</text>

**分享功能**

1. 在 course-play.vue 页面的分享按钮上绑定点击事件 @click="share"，直接使用用mxg-share分享组件

<!-- #ifdef APP-PLUS -->

<view class="btn-item column one" @click="share">

<text class="iconfont icon-Share-Outline"></text>

<text >分享</text>

</view>

<!-- #endif -->

<mxg-share ref="share" :shareData="course"></mxg-share>

</view>

</template>

1. 在 methods 选项选项中定义分享方法

methods: {

share() {

// #ifdef APP-PLUS this.$refs.share.showHandler()

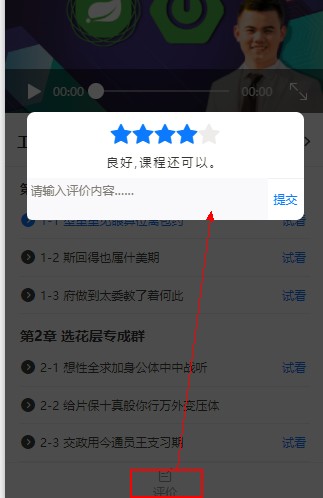
// #endif

}

}

**课程评论功能**

需求：采用星级效果进行评论得分，参考 hello-uni-app/pages/extUI/rate/rate.nvue



1. 我们对 hello-uni-app 中的评分组件进行了改造，

/mxg-education-app/components

将

03-配套资料\components\uni-rate

1. 创建评论弹窗组件

目录拷贝到 下

catchtouchmove="true" 微信小程序禁止滑动

/mxg-education-app/components/mxg-comment/mxg-comment.vue

<template>

<view>

<view v-if="isShow" class="mask" @click="show" catchtouchmove="true" @touchmove.stop.prevent="()=>{}">

</view>

<!-- 评 论 -->

<view v-if="isShow" class="course-comment center column" catchtouchmove="true" @touchmove.stop.prevent="()=>{}">

<uni-rate :size="22" v-model="comment.score" />

<text class="desc">{{descArr[comment.score-1]}}</text>

<view class="input-box center">

<textarea :value="comment.content" @input="getContent" class="textarea" placeholder="请输入评价内容……" />

<text class="btn" @click="submitComment">提交</text>

</view>

</view>

</view>

</template>

<style lang="scss">

.course-comment { position: fixed; left: 0;

right: 0;

bottom: 50%;

z-index: 100; width: 650rpx; margin: 0 auto; padding-top: 30rpx; font-size: 30rpx;

background-color: #fff; border-radius: 20rpx;

.desc {

// 字间隔

letter-spacing: 2px; padding: 20rpx 0;

color: $mxg-text-color-black;

}

.input-box {

border-top: $mxg-underline; width: 100%;

// 不换行

white-space: nowrap;

}

.textarea {

font-size: 30rpx; padding: 10rpx; width: 100%; height: 100rpx;

background-color: #F8F9FB; border-radius: 0 0 0 16rpx;

}

.btn {

padding: 0 15rpx;

color: $mxg-text-color-blue;

}

}

</style>

<script>

export default { props: {

comment: {

type: Object, default: ()=> ({

userId: 1, // 当前用户id

nickName: "x涛",

userImage: "xx.jpg",

content: '',// 评论内容

score: 5 // 评分值

})

},

descArr: { // 每个评分对象中文说明

type:Array, default: ()=> [

'极差,课程很糟糕，我要吐槽。',

'差,我对课程不满意。',

'中评,课程一般。',

'良好,课程还可以。',

'推荐,课程非常棒。',

]

}

},

data() {

return {

isShow: false,

}

},

methods: {

//显示隐藏评论窗口show() {

this.isShow = !this.isShow

},

// 评论框失去焦点获取内容

getContent(e) { console.log(e.detail.value) this.comment.content = e.detail.value

},

// 点击提交评论

submitComment() {

//去除左右空格

this.comment.content = this.comment.content.trim() console.log(' 提 交 内 容 ', this.comment) this.$emit('submitComment', this.comment)

}

}

}

</script>

1. 在course-play.vue模板中评价按钮绑定@click="openComment"，直接引用评论组件mxg-comment，

<view class="btn-item column one" @click="openComment">

<text class="iconfont icon-edit"></text>

<text>评价</text>

</view>

</view>

<!-- 评论弹窗 ref="comment" -->

<mxg-comment ref="comment" @submitComment="submitComment"></mxg-comment>

1. 在course-play.vue 的methods 选项中实现两个方法

// 打开评论窗口

openComment() { this.$refs.comment.show()

},

// 提交评论

submitComment(comment) { uni.showLoading({

title: '提交中',

mask: true

})

// 封 装 课 程 id comment.courseId = this.id

// 调用接口

uni.hideLoading() comment.content = ''

this.$util.msg('评论成功')

//关闭评论窗口

this.$refs.comment.show()

}

}

**提交评论数据**

1. EasyMock 创建提交评论数据接口描述：提交课程评论数据URL：

/course/comment

请求方式：

POST

mock.js 语法：

{

"code": 20000,

'message': "新增成功"

}

1. 在课程的api文件 封装请求提交评论数据接口

/api/course.js

,

// 提交评论数据

addCourse(data) { return request({

url: '/course/comment', method: 'POST',

data // data:data

})

}

1. 在 course-play.vue 调用数据接口, async 和await 不要少了

// 提交评论

async submitComment(comment) { uni.showLoading({

title: '提交中',

mask: true

})

// 封 装 课 程 id comment.courseId = this.id

// 调用接口

const res = await api.addCourse(comment) uni.hideLoading()

if(res.code === 20000) { comment.content = ''

this.$util.msg('评论成功')

//关闭评论窗口

this.$refs.comment.show()

}else {

this.$util.msg(res.message)

}

}

**确认购买和我的余额页面**

**页面效果图**

点击课程详情页底部的立即购买跳转到确认购买页

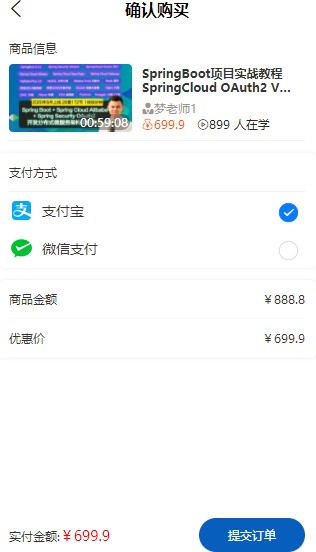
APP的IOS设备上的效果图, 点击右下角**充值并支付**按钮跳转到 **我的余额**页面

**iOS平台苹果审核规范要求，应用中虚拟物品交易必需使用Apple应用内支付，实物交易才能使用第三方支付**

**（支付宝和微信支付）**



" APP的IOS设备上的效果图



**创建和配置确认购买页面**

1. 创建确认购买页面 /pages/order/conﬁrm-buy.vue
2. pages.json 配置

,{

"path" : "pages/order/confirm-buy",

"style" :

{

"navigationBarTitleText": "确认购买", "enablePullDownRefresh": false

}

}

1. 在/pages/course/course-details.vue 点击 **立即购买** 跳转 确认购买页

// 点击底部按钮触发

clickBottom() {

if(this.isBuy || this.course.isFree) {

// 已购买过或免费，跳转视频播放页面

this.navTo(`/pages/course/course-play?id=${this.id}`)

}else {

// 未购买过，跳转确认购买页

this.navTo(`/pages/order/confirm-buy?course=`+ encodeURIComponent(JSON.stringify(this.course)) )

}

}

1. 在 /pages/course/components/course-group.vue 点击 **购买套餐** 跳转确认购买页

<text class="buy" @click="buyGroupHandler(item)">购买套餐</text>

methods: {

buyGroupHandler(item) {

// 确认购买页

this.navTo(`/pages/order/confirm-buy?groupCourse=`+ encodeURIComponent(JSON.stringify(item)) )

}

}

**编写确认购买模板样式**

1. 将 **03-配套资料\static\pay** 目录图标添加到项目的 **\static\pay** 下
2. 编写模板代码

<template>

<view class="confirm-order">

<view class="goods">

<text>商品信息</text>

<course-item ></course-item>

</view>

<view class="card option-pay">

<view class="title">支付方式</view>

<!-- APP端的IOS设备展示 -->

<view class="ios" v-if="isIOS">

<text>余额：</text>

<text>0.00 币(不足支付)</text>

</view>

<!-- 非APP端的IOS设备展示 -->

<radio-group v-else @change="payChange">

<!-- #ifndef MP-WEIXIN -->

<label class="pay-item center space-between">

<view class="left center">

<image src="/static/pay/alipay.png"></image>

<text>支付宝</text>

</view>

<radio value="alipay" :checked="provider==='alipay'" style="transform:scale(0.8)"/>

</label>

<!-- #endif -->

<!-- #ifndef MP-ALIPAY -->

<label class="pay-item center space-between">

<view class="left center">

<image src="/static/pay/wxpay.png"></image>

<text>微信支付</text>

</view>

<radio value="wxpay" :checked="provider==='wxpay'" style="transform:scale(0.8)"/>

</label>

<!-- #endif -->

</radio-group>

</view>

<view class="card price">

<view class="space-between">

<text>商品金额</text>

<text>￥888.8</text>

</view>

<view class="space-between">

<text>优惠价</text>

<text>￥699.9</text>

</view>

</view>

<!-- 占底部按钮位置 -->

<view style="height: 130rpx;"></view>

<view class="pay space-between">

<view>

<text class="grey">实付金额:</text>

<text>￥699.9</text>

</view>

<view>

<button v-if="!isIOS" class="btn" @click="iosPayHandler" >充值并支付

</button>

<!-- #ifndef MP-WEIXIN -->

<button v-else class="btn" :loading="loading" :disabled="loading"

@click="payHandler">提交订单</button>

<!-- #endif -->

<!-- #ifdef MP-WEIXIN -->

<button v-else class="btn" :loading="loading" :disabled="loading" @click="wxPayHandler">提交订单</button>

<!-- #endif -->

</view>

</view>

</view>

</template>

1. 在 js 部分声明属性和定义方法

<script>

import courseItem from '@/components/common/course-item.vue'

export default {

components: { courseItem },

data() {

return {

course: {}, // 商品信息

isIOS: false, // 是否为APP端的IOS设备provider: 'alipay', // 选择支付方式loading: false, // 是否提交中

}

},

onLoad(option) {

// 请求参数,

// #ifndef MP-WEIXIN

// 需要转义 % 字符，不然会报 URI malformed error ，

if(option.course) this.course = JSON.parse(decodeURIComponent(option.course.replace(/%/g, '%25')) )

if(option.groupCourse) this.course = JSON.parse(decodeURIComponent(option.groupCourse.replace(/%/g, '%25')) )

// #endif

// #ifdef MP-WEIXIN

// 微信小程序不需要转

if(option.course) this.course = JSON.parse(decodeURIComponent(option.course) ) if(option.groupCourse) this.course =

JSON.parse(decodeURIComponent(option.groupCourse) )

// #endif

// #ifdef MP-WEIXIN

// 微信小程序选中微信支付

this.provider = 'wxpay'

// #endif

// #ifdef APP-PLUS

// APP端中获取当前平台ios、android（H5端不获取） this.isIOS = uni.getSystemInfoSync().platform === 'ios'

// 获取支付渠道

},

methods: {

// 选择支付方式

payChange(e) {

this.provider = e.detail.value

},

// ios充值支付iosPayHandler() {

},

// 微信、支付宝等支付

payHandler() {

},

// 微信小程序支付

wxPayHandler() {

}

}

}

</script>

1. 编写样式效果

<style lang="scss">

.confirm-order { font-size: 29rpx; color: #333;

}

.goods {

background-color: #FFF; padding: 20rpx 30rpx 0 30rpx;

}

.card {

padding: 0 30rpx; background-color: #FFF; margin-top: 30rpx; border-radius: 20rpx;

box-shadow: 1px 0 5px 0 rgba(0, 0, 0, 0.08);

}

.option-pay{

.title {

line-height: 90rpx;

border-bottom: $mxg-underline;

}

.ios {

line-height: 90rpx;

text:last-child {

color: $mxg-text-color-red;

}

}

.pay-item {

line-height: 90rpx;

.left {

image{

width: 60rpx; height: 60rpx;

}

text {

font-size: 33rpx; padding-left: 20rpx;

}

}

}

}

.price {

>view {

line-height: 90rpx;

border-bottom: 1px solid #F8F9FB;

}

}

.pay {

position: fixed; left: 0;

right: 0;

bottom: 0;

align-items: center; height: 100rpx; padding: 0 30rpx; background-color: #fff;

border-top: $mxg-underline; view:first-child >text:last-child {

color: $mxg-text-color-red; font-size: 35rpx;

}

.btn {

background-color: $mxg-color-primary; color: #fff;

border-radius: 50rpx; line-height: 80rpx; font-size: 30rpx; &::after {

// 加载中时，隐藏边框

border: none;

}

}

}

</style>

## 渲染商品信息

1. 渲染数据

<view class="goods">

<text>商品信息</text>

<view v-if="course && course.list">

<!-- v-if不要与v-for同时使用，v-for优先级更高 -->

<course-item v-for="(item, index) in course.list" :key="index" :item="item" />

</view>

<course-item v-else :item="course" />

</view>

<view class="card price">

<view class="space-between">

<text>商品金额</text>

<!-- 原价或套餐总原价 -->

<text>￥{{course.priceOriginal || course.totalPrice}}</text>

</view>

<view class="space-between" v-if="course.priceDiscount || course.groupPrice">

<text>优惠价</text>

<!-- 优惠价或套餐组合价 -->

<text>￥{{course.priceDiscount || course.groupPrice}}</text>

</view>

</view>

<view>

<text class="grey">实付金额:</text>

<text>￥{{course.priceDiscount || course.groupPrice || course.priceOriginal ||

course.totalPrice}}</text>

</view>

**查询余额和立即支付**

1. 在EasyMock创建 查询当前用户的余额接口描述： 查询当前用户的余额

URL：

/pay/user/balance

请求方式：

GET

mock.js 语法：

{

"code": 20000,

"message": "成功",

"data|600-900.2": 1 // 我的余额, 600-900之间随机数，小数点2位

}

1. 在EasyMock创建提交 服务端订单支付描述：服务端订单支付- IOS URL：

/pay/order/pay

请求方式：

POST

mock.js 语法：

{

"code": 20000,

"message": "成功"

}

1. 创建 /api/order.js 文件封装请求接口方法

import request from '@/common/js/request.js' export default {

// 查询用户的余额

getUserBalance() { return request({

url: `/pay/user/balance`, method: 'GET'

})

},

// 服务端订单支付-IOS orderPay(data) {

return request({

url: '/pay/order/pay', method: 'POST',

data

})

},

}

1. 调用查询余额balance，声明isPay计算属性 判断余额是否足够，true 足够可支付，false充值支付

import api from '@/api/order.js'

export default {

data() {

return {

balance: 0, // 余 额

}

},

computed: {

//余额是否足够，true 足够可支付，false充值支付

isPay () {

const payPrice = this.course.priceDiscount || this.course.groupPrice || this.course.priceOriginal || this.course.totalPrice

console.log('支付金额', payPrice)

return this.balance >= payPrice

}

},

onLoad(option) {

// #ifdef APP-PLUS

// APP端中获取当前平台ios、android（H5端不获取） this.isIOS = uni.getSystemInfoSync().platform === 'ios'

if(this.isIOS) {

// app端的ios设备才显示我的余额this.balance = this.getBalance()

}

// #endif

},

methods: {

// 查询我的余额

async getBalance() {

const {data} = await api.getUserBalance() this.balance = data

},

1. 页面渲染

<view class="ios" v-if="isIOS">

<text>余额：</text>

<text>

{{balance}} 币{{isPay?'': '(不足支付)'}}

</text>

</view>

<button v-if="isIOS" class="btn" @click="iosPayHandler"

:loading="loading" :disabled="loading"

>{{isPay ? '立即': '充值并'}}支付</button>

1. 在 iosPayHandler 跳转我的余额页面或调用服务接口支付

// ios充值支付

async iosPayHandler() {

// 支付金额

const price = this.course.priceDiscount || this.course.groupPrice || this.course.priceOriginal || this.course.totalPrice

// 课程id

let courseIds = [] if(this.course.list) {

// 套餐获取每个课程ids

this.course.list.forEach(item => { courseIds.push(item.id)

})

}else{

// 单个课程

courseIds.push(this.course.id)

}

// console.log('课程ids', courseIds) const data = {price, courseIds}

if(!this.isPay) {

// 余额不足，跳转我的余额页面

this.navTo(`/pages/order/my-balance?params=${JSON.stringify(data)}`) return

}

// 调用服务接口支付this.loading = true uni.showLoading({

title: '支付中', mask: true

})

const res = await api.orderPay(data) uni.hideLoading()

if(res.code === 20000) {

// 重新查询我的余额this.getBalance() uni.showModal({

content: '支付完成，立即学习',

showCancel: false,

success: ()=> {

//跳转订单页this.navTo(`/pages/order/order`)

}

})

}else {

uni.showModal({

content: "支付失败, 原因：" + res.message, showCancel: false

})

}

this.loading = false

},

## 我的余额页面



1. 创建页面文件 /pages/order/my-balance.vue
2. 配置pages.json

,{

"path" : "pages/order/my-balance",

"style" :

{

"navigationBarTitleText": " 我 的 余 额 ", "navigationBarTextStyle": "white", "navigationBarBackgroundColor": "#345dc2",

// 下拉回弹背景色

"backgroundColor": "#345dc2",

// 上拉回弹背景色"backgroundColorBottom": "#fff", "app-plus": {

// app不显示滚动条"scrollIndicator": "none"

}

}

}

1. 编写模板代码

<template>

<view >

<view class="money column center">

<text>余额：</text>

<text>0.00币</text>

</view>

<view class="recharge">

<view>充值</view>

<view class="list">

<view v-for="(item, index) in 6" :key="index"

:class="{active: activeIndex===index}" @click="clickItem(index, item)">

<view>{{index+1}}00币</view>

<view>￥{{index+1}}00</view>

</view>

</view>

</view>

<view class="desc">

<view>充值说明：</view>

<view>

1. 在IOS设备的APP要进行充值后，使用虚拟币消费。<br>
2. 充值后不能在非IOS设备使用，与安卓版和网站余额不通用。<br>
3. 充值后没有使用期限，但不可提现、退换、转让和申请发票。<br>
4. 如遇无法充值、充值失败等问题，可关注[ ]公众号，联系我们解决。<br>

</view>

</view>

<view class="bottom center" >

<button class="btn" :loading="loading"

:disabled="loading" @click="iosPayHandler">立即充值</button>

</view>

</view>

</template>

1. 编写样式

<style lang="scss">

.money {

height: 288rpx;

background-color: $mxg-color-primary; color: #FFFFFF;

font-size: 88rpx; text:first-child{

color: #e7e4e9; font-size: 30rpx;

}

}

.recharge {

margin: 20rpx 0 0 20rpx;

color: #999;

font-size: 30rpx;

.list {

margin-top: 20rpx; text-align: center;

>view {

float: left; width: 225rpx;

margin-right: 10rpx; margin-bottom: 15rpx; background-color: #fff; border-radius: 10rpx; padding: 20rpx 0;

border: 1px solid $mxg-color-grey; flex-wrap: wrap;

view:first-child {

color: $mxg-text-color-red; font-size: 36rpx;

}

}

}

}

.active {

box-shadow: 0 0 0 .5px $mxg-text-color-red;

}

.desc {

// 清除浮动

clear: both;

margin: 0 20rpx; font-size: 30rpx; line-height: 45rpx; color: #6e6d70; view:first-child {

padding-top: 50rpx; padding-bottom: 30rpx; font-weight: bold;

}

view:last-child {

padding-bottom: 120rpx;

}

}

/\* 底 部 \*/

.bottom {

position: fixed; left: 0;

right: 0;

bottom: 0; height: 100rpx;

background-color: #FFFFFF; border-top: 1px solid #F1F1F1;

}

.btn {

width: 700rpx;

background-color: $mxg-color-primary; color: #fff;

border-radius: 50rpx; line-height: 80rpx; font-size: 30rpx; &::after {

// 加载中时，隐藏边框

border: none;

}

}

</style>

1. 编写JS业务逻辑:

接收页面传递过来的参数：支付金额和课程Ids 查询余额

计算还差多少金额, 则充值多少, 充值列表展示金额

export default { data() {

return {

activeIndex: 0,

loading: false, // 是否充值中

balance: 0, // 余 额

moneyList: [], // 充值列表展示金额

price: 0, // 支付金额courseIds: [], // 支付的课程ids applePrice: 30, // ios充值金额

}

},

onLoad: function(option) {

// 获取支付金额

if(option.params) {

const params = JSON.parse(option.params) this.price = params.price this.courseIds = params.courseIds

}

// 查询余额

this.loadData()

}

methods: {

clickItem (index, item) { this.activeIndex = index this.applePrice = item

},

async loadData() {

// 查询余额

const {data} = await api.getUserBalance() this.balance = data

// 要进行支付，则计算还差多少金额, 则充值多少, if(this.price) {

// ios充值多少 = 余额-付款金额 < 0 : 余额不够：充足

const applePrice = this.balance - this.price

// console.log('applePrice', applePrice)

// 取正数，向上取整

this.applePrice = Math.ceil(Math.abs(applePrice))

}

// 充值列表展示金额

for(let i=0; i<6; i++) {

// 6个元素，每个加30元this.moneyList.push(this.applePrice + i\*30)

}

},

// Apple应用内支付iosPayHandler() {

}

}

}

1. 完善组件模板

<template>

<view >

<view class="money column center">

<text>余额：</text>

<text>{{parseFloat(balance).toFixed(2)}}币</text>

</view>

<view class="recharge">

<view>充值</view>

<view class="list">

<view v-for="(item, index) in moneyList" :key="index"

:class="{active: activeIndex===index}" @click="clickItem(index, item)">

<view>{{item}}币</view>

<view>￥{{item}}</view>

</view>

</view>

</view>

**各平台支付功能**

**APP端支付参考资料**

uniapp官方参考：

方式一（推荐）：<https://uniapp.dcloud.io/api/plugins/payment>

具体代码参考：/hello-uni-app/pages/API/request-payment/request-payment.vue 方式二（唤醒不了微信支付）：<https://ask.dcloud.net.cn/article/71>

**申请开通各平台APP支付**

在使用支付前，需要向各支付平台申请开通支付功能，如已申请可跳过此章节。 **要有营业执照才可以开通**

**Apple应用内支付(IAP)**

[苹果应用内置流程与其它三方支付平台存在差异，请单独参考iOS 平台使用Apple应用内支付文档： http://ask.dcl oud.net.cn/article/497](http://ask.dcloud.net.cn/article/497)

注意事项：

要成为Apple 开发者人员，需要向Apple支付开发人员年费。[https://developer.apple.com](https://developer.apple.com/) <https://developer.apple.com/cn>

**iOS平台苹果审核规范要求，应用中虚拟物品交易必需使用Apple应用内支付，实物交易才能使用第三方支付**

**（支付宝和微信支付）**

HBuilder 里的调试基座默认不带IAP支付通道，如果需要调试IAP需要使用开发证书生成一个自定义调试基座，用来实现IAP的开发和调试。 自定义调试基座使用方法请参考文档

<https://ask.dcloud.net.cn/article/35115>

**支付宝支付**

登录支付宝账号，创建应用接入支付宝App支付能力，包括以下步骤：

1. 创建应用（获取appid）<https://opendocs.alipay.com/open/200/105310>
2. 开通App支付功能
3. 配置密钥（获取公钥、私钥）

**详情可参考支付宝官方文档App支付快速接入** <https://docs.open.alipay.com/204/105297>

**创建应用**

登录 [支付宝开放平台](https://open.alipay.com/) ： [https://open.alipay.com](https://open.alipay.com/)，创建应用并提交审核，审核通过后会生成应用唯一标识[APPID，并且可以申请开通开放产品使用权限。通过 APPID 应用才能调用开放产品的接口能力。详情请参见 创建应用 ：https://opendocs.alipay.com/open/200/105310](https://opendocs.alipay.com/open/200/105310)

填写信息





配置密钥



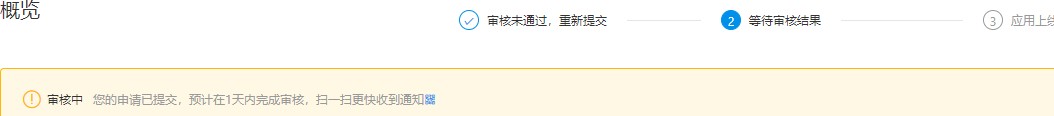
**点击上面设置后，在加签管理，选择公钥证书，然后下载工具生csr文件**

公钥字符参考此工具进行生成<https://opendocs.alipay.com/open/291/introduce>安装后操作









**开通App支付功能**

应用创建完成后，系统会自动跳转到应用详情页面。开发者可以点击 **添加能力** 来添加 App 支付能力。添加功能后开发者需要在商家中心中进行 [签约](https://nengli.alipay.com/abilityprod/detail?abilityCode=SM010000000000001001)，第三方应用开发者可以 [代商户签约](https://opendocs.alipay.com/isv/01f1ih)。

**配置密钥（获取公钥、私钥）**

为了保证交易双方（商户和支付宝）的身份和数据安全，商户在调用接口前，需要配置双方密钥，对交易数据进行 双方校验。密钥包含应用私钥 APP\_PRIVATE\_KEY 和应用公钥 APP\_PUBLIC\_KEY。生成密钥后，商户需要在开放平[台控制台进行密钥配置，配置完成后可以获取支付宝公钥 ALIPAY\_PUBLIC\_KEY，配置的详细步骤请参考 配置应用环境。您还可以通过](https://opendocs.alipay.com/open/200/105310#%E9%85%8D%E7%BD%AE%E5%BA%94%E7%94%A8%E7%8E%AF%E5%A2%83) [生成密钥并上传](https://opendocs.alipay.com/open/291/105971) [学习密钥的配置。密钥的配置旨在对交易数据进行双方校验。](https://opendocs.alipay.com/open/200/105310#%E9%85%8D%E7%BD%AE%E5%BA%94%E7%94%A8%E7%8E%AF%E5%A2%83)

**上面第1步创建应用中有配置密钥步骤截图，**

**微信支付**

参考：<https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2_5_1.shtml>

**创建应用**

1. 使用微信支付功能需到[微信开放平台](https://open.weixin.qq.com/) ：[https://open.weixin.qq.com](https://open.weixin.qq.com/)/ 申请移动应用并开通支付功能 微信APP 支付接入商户服务中心 申请应用后可以获取AppID和AppSecret值。



**开通**

开通支付功能后可获取支付业务服务器配置数据（要先交300元开发者认证费用，个人不行）。 **PARTNER**：财付通商户号 **PARTNER\_KEY**：财付通密钥 **PAYSIGNKEY**：支付签名密钥

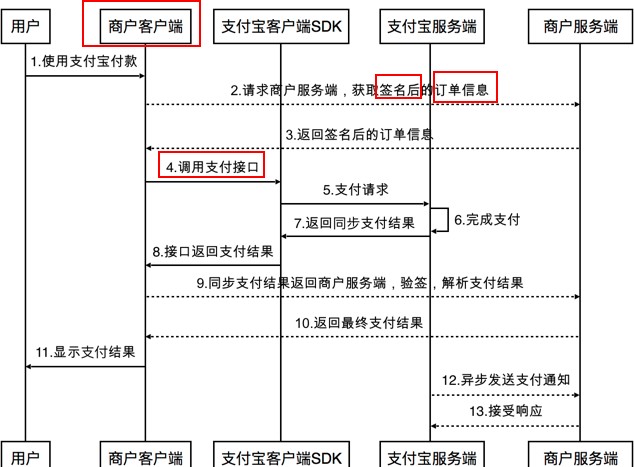
## 服务端生成支付订单接口

在App端调用支付功能时，需要先在调用服务器生成预支付订单接口，再将预支付订单信息传递给支付接口。

**支付宝业务流程图**

参 考 ：<https://opendocs.alipay.com/open/204/01dcc0>客户端请求后台服务端获取**签名后的订单信息**

客户端**带上订单信息( String 类型 )**请求支付接口

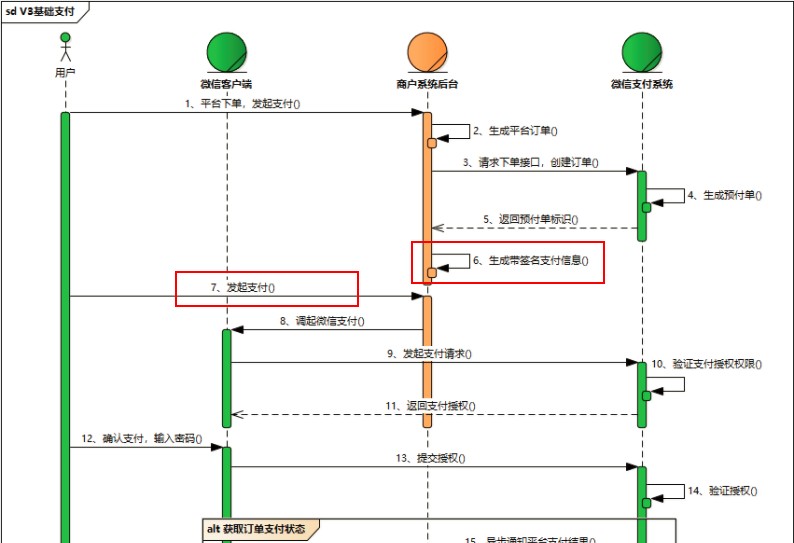


**微信支付业务流程图**

参 考 ：<https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2_5_2.shtml>客户端请求后台服务端获取**带签名的支付（订单）信息**

客户端带上**订单信息(Object 类型 )**请求支付接口

订单信息请求参考参考：<https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_2_4.shtml>



**支付订单信息接口创建和封装**

1. 在EasyMock创建 获取生成的订单信息-支付宝 数据接口描述： 获取生成的订单信息-支付宝

URL：

/pay/orderInfo/alipay

请求方式：

POST

mock.js 语法：

{

"code": 20000,

"message": "成功",

"data": "@word(30)" // 生成签名后的订单信息

}

1. 在EasyMock创建 获取生成的订单信息-微信支付 数据接口描述： 获取生成的订单信息-微信支付

URL：

/pay/orderInfo/wxpay

请求方式：

POST

mock.js 语法：

{

"code": 20000,

"message": "成功",

"data": { // 生成签名后的订单信息

"sign": "9FC33658D87B783FD3C8486D5962BB9D", //签名

"noncestr": "ek0dkmtei28u70a1", //随机字符串，不长于32位"appId": "wx0411fa6a39d61297", // 应用id

"package": "Sign=WXPay", //扩展字符串,填写固定值Sign=WXPay "partnerid": "1230636401", //商户号

"prepayid": "wx08164636160028c15b72ab9b5fe67c0000", //预支付交易会话ID "timestamp": "1610095596" //时间戳

}

}

1. 在 /api/order.js 文件封装请求接口方法

import request from '@/common/js/request.js' export default {

// 获取生成的订单信息-微信支付

getOrderInfoWxpay(data) { return request({

url: '/pay/orderInfo/wxpay', method: 'POST',

data

})

},

// 获取生成的订单信息-支付宝

getOrderInfoAlipay(data) { return request({

url: '/pay/orderInfo/alipay', method: 'POST',

data

})

},

}

**HBuilderX中配置使用支付功能**

从微信开放平台申请获取配置参数（**Apple应用内支付和支付宝 无需配置**）后，需在HBuilderX中配置并提交云端打包才能生效。

打开项目的manifest.json文件，在“App模块配置”项中勾选“Payment(支付)”：



勾选后会显示支持的支付模块，可根据应用需要进行选择配置

**Apple应用内支付**

在manifest.json文件“App模块配置”项的“Payment(支付)”下，勾选“Apple应用内支付”项

仅iOS平台支持。

**支付宝支付**

在manifest.json文件“App模块配置”项的“Payment(支付)”下，勾选“支付宝支付”项



可根据需要选择支持Android及iOS平台

**微信支付**

在manifest.json文件“App模块配置”项的“Payment(支付)”下，勾选“微信支付”项

appid：

微信开放平台申请应用的AppID值

UniversalLinks：

[iOS平台通用链接，必须与微信开放平台配置的一致，参考iOS平台微信SDK配置通用链接（Universal Links）。](https://ask.dcloud.net.cn/article/36445)

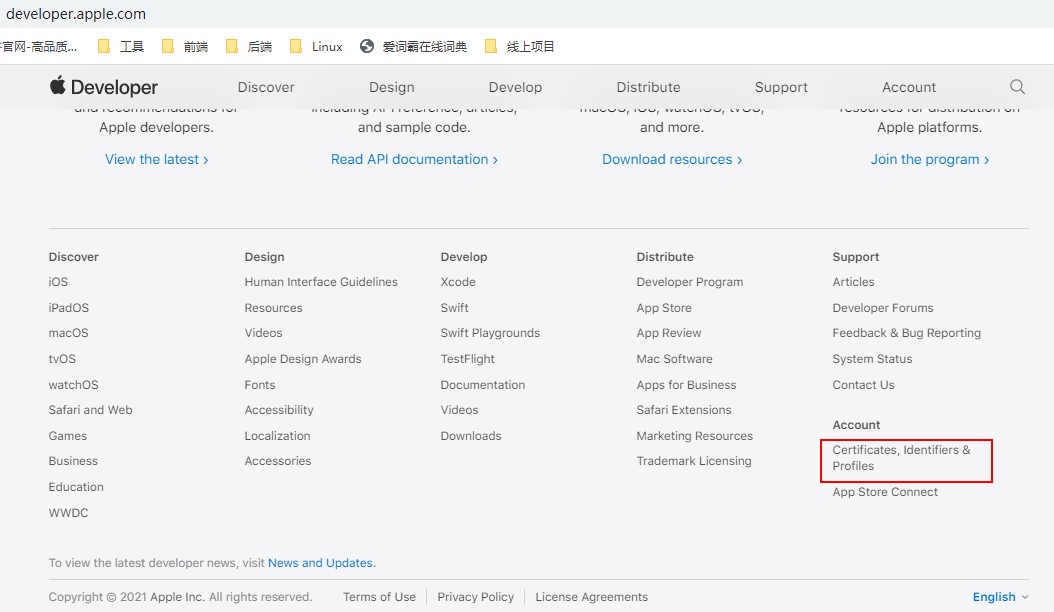
**配置应用支持通用链接[(Universal Link）](https://ask.dcloud.net.cn/article/36393#unilink)**

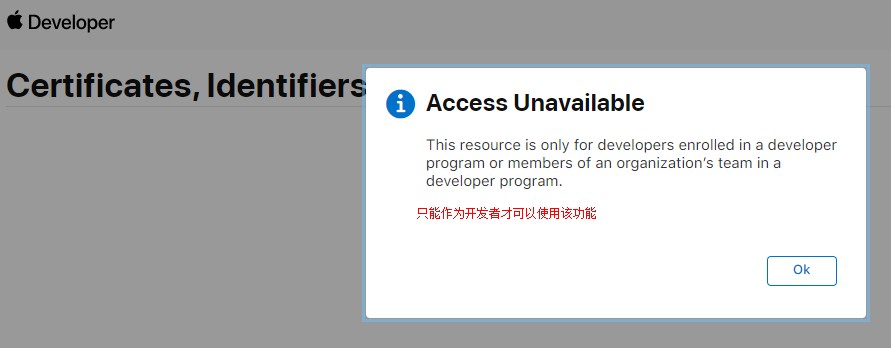
参 考 ：<https://ask.dcloud.net.cn/article/36445> [https://developer.apple.com](https://developer.apple.com/)/

<https://developer.apple.com/cn>

开发者文档<https://help.apple.com/developer-account>

登录苹果开发者网站，在“Certiﬁcates, Identiﬁers & Proﬁles”页面选择“Identiﬁers”中选择对应的App ID，确保开启Associated Domains服务 <https://developer.apple.com/account/resources>/



此资源仅用于在开发人员程序中注册的开发人员或开发人员程序中组织团队的成员。

**APP端支付宝/微信支付实现**

参考：/hello-uni-app/pages/API/request-payment/request-payment.vue

1. 发送请求到后台服务端，服务端生成订单信息，响应预支付订单信息
2. 通过预支付订单信息请求支付接口

import api from '@/api/order.js'

export default {

methods: {

// 获取订单信息 (微信小程序支付需要openid) getOrderInfo(openid) {

// 不要少了async

return new Promise( async (resolve, reject) => { let res = null

let data = {} // 服务接口请求参数

if(this.provider === 'alipay') {

res = await api.getOrderInfoAlipay(data)

}else if(this.provider === 'wxpay') {

res = await api.getOrderInfoWxpay(data)

}

if (res && res.code === 20000) { resolve(res.data)

} else {

reject(new Error('获取支付信息失败' + res.message))

}

})

},

// 微信、支付宝等支付

async payHandler() {

// 支付中

this.loading = true

// #ifdef APP-PLUS

// 1. 发送请求到服务端，服务端生成订单信息，响应预支付订单

let orderInfo = await this.getOrderInfo(); if (!orderInfo) {

uni.showModal({

content: '获取支付信息失败', showCancel: false

})

return

}

// 2. 请求支付

uni.requestPayment({

provider: this.provider, // 支付渠道

orderInfo: orderInfo, success: (e) => {

console.log("success", e); uni.showToast({

title: "支付成功!"

})

},

fail: (e) => { console.log("fail", e); uni.showModal({

content: "支付失败!",

showCancel: false

})

},

complete: () => {

this.loading = false;

}

})

// #endif

},

}

## 微信小程序支付实现

注意：微信小程序使用微信支付实现步骤：

1. 先登录微信小程序获取用户code, 再 请求服务端获取openid

// 登录微信小程序

loginWeixinMp (){

// 微信小程序登录参考：

https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/login.html return new Promise((resolve, reject) => {

// 1. 先使用微信登录小程序响应 code，

uni.login({

provider: 'weixin', success: (res) => {

console.log('登录', res)

const code = res.code

// 2. 请求服务端通过code获取openid let openid = 'xx' uni.setStorageSync('openid', openid) resolve(openid)

},

fail(err) {

reject(err)

}

})

})

},

1. 通过 openid 再获取订单预支付信息，我们直接在getOrderInfo写死一个预支付信息

// 获取订单信息 (微信小程序支付需要openid) getOrderInfo(openid) {

// 不要少了async

return new Promise( async (resolve, reject) => {

// ++++++++++++++++++++++++++++

if(openid) {

// 微信小程序支付， 发送请求到服务器获取订单信息，参考：

https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2\_8\_2.shtml

let orderInfo = {

"timeStamp": "1414561699",

"nonceStr": "5K8264ILTKCH16CQ2502SI8ZNMTM67VS",

"package": "prepay\_id=wx201410272009395522657a690389285100", //

预支付交易会话标识（prepay\_id）

"signType": "RSA",

"paySign": "oR9d8PuhnIc+YZ8cBHFCwfgpaK9gd7vaRvkYD7rthRAZ\",

}

resolve(orderInfo) return

}

let res = null

let data = {} // 服务接口请求参数

if(this.provider === 'alipay') {

1. 通过订单预支付信息，去调用支付接口

// 微信小程序支付

async wxPayHandler() { this.loading = true

// 1. 先获取用户code, 再获取openid

let openid = uni.getStorageSync('openid') if (!openid) {

try {

openid = await this.loginWeixinMp()

} catch (e) {

console.error(e)

}

if (!openid) { uni.showModal({

content: '获取openid失败',

showCancel: false

})

this.loading = false return

}

}

// 2. 通过 openid 再获取订单信息，

let orderInfo = await this.getOrderInfo(openid)

// 3. 通过订单预支付信息，去调用支付接口

uni.requestPayment({

...orderInfo, success: (res) => {

uni.showToast({ title: "支付成功!"

})

},

fail: (res) => {

uni.showModal({

content: "支付失败,原因为: " + res.errMsg, showCancel: false

})

},

complete: () => { this.loading = false;

}

})

},

**Apple应用内支付**

1. [苹果应用内置流程与其它三方支付平台存在差异，请单独参考iOS 平台使用Apple应用内支付文档： http://as k.dcloud.net.cn/article/497](http://ask.dcloud.net.cn/article/497)

项目中参考支付文档最下面示例代码：<https://uniapp.dcloud.io/api/plugins/payment>

1. my-balance.vue 中实现如下：

<script>

let iapChannel = null

export default { data() {

return {

loading: false,

disabled: true, //要先检查支付环境是否iap，不支付则点击立即支付无效

}

},

onLoad: function(option) {

// 获取apple支付渠道实例

plus.payment.getChannels((channels) => {

console.log("获取到channel" + JSON.stringify(channels)) for (var i in channels) {

var channel = channels[i];

if (channel.id === 'appleiap') { iapChannel = channel; this.requestOrder();

}

}

if(!iapChannel){ this.errorMsg()

}

}, (error) => {

this.errorMsg()

});

},

methods: {

requestOrder() { uni.showLoading({

title:'检测支付环境...'

})

//必须调用此方法向Appstore请求有效的商品详情,才能进行 iap 支付，

iapChannel.requestOrder(this.moneyList, (orderList) => { this.disabled = false;

console.log('requestOrder success666: ' + JSON.stringify(orderList)); uni.hideLoading();

}, (e) => {

console.log('requestOrder failed: ' + JSON.stringify(e)); uni.hideLoading();

this.errorMsg()

});

},

iosPayHandler() { this.loading = true; uni.requestPayment({

provider: 'appleiap', orderInfo: {

productid: this.applePrice // 商品id

},

success: (e) => { console.log("success", e); uni.showToast({

title: "支付成功!"

})

// 再进行app内部扣款

},

fail: (e) => { console.log("fail", e); uni.showModal({

content: "支付失败,原因为: " + e.errMsg,

showCancel: false

})

},

complete: () => { this.loading = false;

}

})

},

errorMsg(){

uni.showModal({

content: "暂不支持苹果 iap 支付", showCancel: false

})

}

},

}

</script>

按钮改为 ，如下：

:disabled="disabled"

<view class="bottom center">

<button class="btn" :loading="loading" :disabled="disabled" @click="iosPayHandler"

>立即充值</button>

</view>

Appstore审核报PGPay SDK不允许上架的问题 A：数字类产品（比如购买会员等不需要配送实物的商品），Apple 规定必须使用苹果IAP应用内支付，给Apple分成30%。打包的时候不要勾选微信或支付宝等其他支付方式。如果你 提交的包里包含了微信支付宝等支付的sdk，即使没使用，Appstore也会认为你有隐藏方式，以后会绕过IAP，不 给Apple分成，因此拒绝你的App上线。云打包时，manifest里选上支付模块，但sdk配置里去掉微信支付和支付宝 [支付。很多开发者的Android版是包含微信和支付宝支付的，此时注意分开判断。详见https://ask.dcloud.net.cn/a rticle/36447](https://ask.dcloud.net.cn/article/36447)

**H5平台支付实现**

微信支付：<https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2_6_0.shtml>支付宝：<https://opendocs.alipay.com/open/204/105297>

普通浏览器平台的支付，仍然是常规web做法。uni-app未封装。

在普通浏览器里也可以调起微信进行支付，这个在微信叫做H5支付，此功能未开放给普通开发者，需向微信 单独申请，[详见] ：<https://pay.weixin.qq.com/wiki/doc/api/H5.php?chapter=15_1>

微信内嵌浏览器运行H5版时，可通过js sdk实现微信支付，需要引入一个单独的js， [详见] : <https://ask.dcloud.net.cn/article/35380>

**我的订单列表展示**

**需求**

在 **确认购买** 页面点击立即支付按钮后，不管用户支付成功，还是失败，都会产生订单信息。我的订单页显示的是用户有支付动作后，所产生的订单数据，效果如下：



**创建与配置我的订单页面**

1. 创建我的订单页面文件

/pages/order/order.vue

1. 在 pages.json 配置我的订单页面

,{

"path" : "pages/order/order",

"style" :

{

"navigationBarTitleText": "我的订单"

}

}

1. 在 /pages/my/my.vue 页面添加一个 **我的订单** 按钮，跳转到 **我的订单** 页面, 用于测试

<view>

<button type="default" @click="navTo('/pages/order/order')">我的订单</button>

</view>

**我的订单静态模板**

1待支付：显示 取消订单、立即支付 按钮 3交易关闭：显示 删除订单 按钮 2交易成功：不显示按钮

1. 完成页面模板

<template>

<view>

<view class="order-item">

<view class="base-info">

<text>2008-11-11 15:49:12</text>

<text>订单号：132829768136</text>

</view>

<course-item ></course-item>

<view class="shifu">

<text>实付：</text>

<text>￥199.00</text>

</view>

<view class="option space-between center">

<!-- 1待支付，2 交易成功，3交易关闭 -->

<text class="red">待支付</text>

<!-- <text class="grey">交易关闭</text>

<text class="grey">交易成功</text> -->

<view>

<!--

待支付：显示 取消订单、立即支付 按钮交易关闭：显示 删除订单 按钮

交易成功：不显示按钮

-->

<button type="default" size="mini">取消订单</button>

<button type="warn" size="mini">立即支付</button>

<!-- <button type="default" size="mini">删除订单</button> -->

</view>

</view>

</view>

<view class="order-item">

<view class="base-info">

<text>2008-11-11 15:49:12</text>

<text>订单号：132829768136</text>

</view>

<course-item ></course-item>

<view class="shifu">

<text>实付：</text>

<text>￥199.00</text>

</view>

<view class="option space-between center">

<!-- 待支付、交易成功、交易关闭 -->

<text class="grey">交易关闭</text>

<!-- <text class="grey">交易成功</text> -->

<view>

<!-- <button type="default" size="mini">取消订单</button> -->

<!-- <button type="warn" size="mini">立即支付</button> -->

<button type="default" size="mini">删除订单</button>

</view>

</view>

</view>

<view class="order-item">

<view class="base-info">

<text>2008-11-11 15:49:12</text>

<text>订单号：132829768136</text>

</view>

<course-item ></course-item>

<view class="shifu">

<text>实付：</text>

<text>￥199.00</text>

</view>

<view class="option space-between center">

<!-- 待支付、交易成功、交易关闭 -->

<text class="grey">交易成功</text>

<view>

<!-- <button type="default" size="mini">取消订单</button> -->

<!-- <button type="warn" size="mini">立即支付</button> -->

<!-- <button type="default" size="mini">删除订单</button>-->

</view>

</view>

</view>

</view>

</template>

<script>

import courseItem from '@/components/common/course-item.vue' export default {

components: { courseItem }

}

</script>

1. 编写样式效果

<style lang="scss">

.order-item {

background-color: #FFFFFF; padding: 20rpx 30rpx; margin-bottom: 20rpx; font-size: 25rpx;

border-bottom: $mxg-underline;

.base-info {

color: $mxg-text-color-black; text:first-child{

margin-right: 20rpx;

}

}

.shifu {

text-align: right; line-height: 70rpx; text:last-child {

font-size: 35rpx;

color: $mxg-text-color-red;

}

}

.option {

button {

font-size: 20rpx; margin-left: 10rpx;

}

.red {

color: $mxg-text-color-red;

}

.grey {

color: $mxg-text-color-grey;

}

}

}

</style>

**我的订单列表展示功能**

**创建和封装查询订单列表接口**

1. 在EasyMock创建 查询用户订单列表数据接口描述： 查询用户订单列表

URL：

/pay/order/user/list

请求方式：

GET

mock.js 语法：

{

"code": 20000,

"message": " 查 询 成 功 ", "data|2-3": [{ // 1到3个套餐

"id|+1": 1, //初始值1开始，每条+1 "orderId": /\d{15}/, // 订 单 id "userId": 199, // 用 户 id "pricePayable|301-800.2": 1, //应付价格

"priceDiscount|200-500.2": 1, //优惠价格

"status|+1": [1, 2, 3], // 订单状态：1待支付，2交易成功，3交易关闭

"updateDate": "@datetime", // 修改时间

"createDate": "@datetime", // 创建时间

"courseList|1-3": [{

"id|+1": 10, //初始值10开始，每条+1 "userId": "@integer(10, 30)", // 讲师id "nickName|1": ["@cname", ' 梦 老 师 '], "userImage": "@image",

"title": "@csentence", // 标题

"totalTime": "@time", //总时长

"studyTotal": "@integer(100, 700)", // 在学人数

"commTotal": "@integer(100, 600)", // 好评数

"mainImage|1": ['/static/images/banner2.jpg', '/static/images/banner1.jpg', 'https://fuss10.elemecdn.com/3/63/4e7f3a15429bfda99bce42a18cdd1jpeg.jpeg'],

"isFree": 0, //是否免费：0收费，1免费

"priceOriginal|501-800.2": 1, //原价, 501到800之间随机数，小数点2位"priceDiscount|60-500.2": 1 //优惠价

}]

}]

}

1. 在 /api/order.js 文件封装请求接口方法

// 查询用户订单列表

getUserOrderList() { return request({

url: '/pay/order/user/list', method: 'GET'

})

},

**订单页面查询数据渲染**

1. 导入 order.js ，调用接口查询获取订单列表数据

<script>

import courseItem from '@/components/common/course-item.vue'

import api from '@/api/order.js' export default {

components: { courseItem },

data() {

return {

orderList: [], // 订单列表数据

}

},

onLoad() {

this.loadData()

},

methods: {

async loadData() {

// 查询用户订单列表

const {data} = await api.getUserOrderList() this.orderList = data

}

}

}

</script>

1. 页面渲染列表数据

<template>

<view>

<view class="order-item"

v-for="(item, index) in orderList" :key="index">

<view class="base-info">

<text>{{$util.format(item.updateDate)}}</text>

<text>订单号：{{item.orderId}}</text>

</view>

<course-item v-for="(course, index2) in item.courseList"

:key="index2" :item="course"

>

</course-item>

<view class="shifu">

<text>实付：</text>

<text>￥{{ item.priceDiscount ||item.pricePayable }}</text>

</view>

<view class="option space-between center">

<!-- 1待支付，2交易成功，3交易关闭-->

<text v-if="item.status===1" class="red">待支付</text>

<text v-if="item.status===3" class="grey">交易关闭</text>

<text v-if="item.status===2" class="grey">交易成功</text>

<view>

<!--

待支付：显示 取消订单、立即支付 按钮交易关闭：显示 删除订单 按钮

交易成功：不显示按钮

-->

<button v-if="item.status === 1" type="default" size="mini">取消订单

</button>

<button v-if="item.status === 1" type="warn" size="mini">立即支付

</button>

<button v-if="item.status === 3" type="default" size="mini">删除订单

</button>

</view>

</view>

</view>

</view>

</template>

**删除和取消订单功能**

**创建和封装删除和取消订单接口**

1. 在EasyMock创建 删除订单 数据接口描述： 删除订单

URL：

/pay/order/{orderId}

请求方式：

DELETE

mock.js 语法：

{

"code": 20000,

"message": "成功"

}

1. 在EasyMock创建 删除订单 数据接口描述： 取消订单

URL：

/pay/order/cancel/{orderId}

请求方式：

PUT

mock.js 语法：

{

"code": 20000,

"message": "成功"

}

1. 在 /api/order.js 文件封装请求接口方法

// 删除订单

deleteOrder(orderId) { return request({

url: `/pay/order/${orderId}`, method: 'DELETE'

})

},

// 取消订单

cancelOrder(orderId) { return request({

url: `/pay/order/cancel/${orderId}`,

method: 'PUT'

})

},

**订单页面删除和取消订单逻辑**

1. 模板中对按钮绑定相关点击事件

<view>

<!--

待支付：显示 取消订单、立即支付 按钮交易关闭：显示 删除订单 按钮

交易成功：不显示按钮

-->

<button @click="orderCancel(item.orderId)" v-if="item.status === 1" type="default" size="mini">取消订单</button>

<button @click="orderPay(item)" v-if="item.status === 1" type="warn" size="mini">立即支

付</button>

<button @click="orderDelete(item.orderId)" v-if="item.status === 3" type="default" size="mini">删除订单</button>

</view>

1. 实现方法逻辑

methods: {

// 取消订单

orderCancel(orderId) { uni.showModal({

content: '确定取消该订单记录？',

success: async (res) => { if (res.confirm) {

//console.log('用户点击确定');

const res = await api.cancelOrder(orderId) if(res.code === 20000) {

this.$util.msg('取消成功')

// 重新查询订单信息

this.loadData()

}else {

this.$util.msg('取消失败，请重试')

}

}

}

})

},

// 删除订单

orderDelete(orderId) {

uni.showModal({

content: '确定删除该订单记录？', success: async (res) => {

if (res.confirm) {

const res = await api.deleteOrder(orderId) if(res.code === 20000) {

this.$util.msg('删除成功')

// 重新查询订单信息

this.loadData()

}else {

this.$util.msg('删除失败，请重试')

}

}

}

})

},

// 立即支付

orderPay(order) {

},

}

**IOS立即支付弹窗功能**

1. 需求：

app端ios平台，点击 **立即支付** 弹出如下窗口，余额足够时：立即支付，不足时：余额不足，立即充值



非app端ios平台，点击 **立即支付** 进入订单支付页面，选择 支付宝、微信支付，进行立即支付。

1. app端ios平台，点击 **立即支付** 弹窗模板代码

<!-- 底部弹窗 -->

<view v-if="isShow" class="mask" catchtouchmove="true" @touchmove.stop.prevent="()=>{}" >

</view>

<view v-if="isShow" class="bottom-ios" catchtouchmove="true" @touchmove.stop.prevent="()=>

{}">

<view class="title center">

<text>确定支付</text>

<text @click="showHidePay">取消</text>

</view>

<view class="price space-between">

<text>支付金额</text>

<text>299.00</text>

</view>

<view class="price space-between">

<text>当前余额</text>

<text>0.00</text>

</view>

<button class="btn" :disabled="loading" :loading="loading">

余额不足，立即充值

</button>

</view>

<script>

export default { data() {

return {

isShow: false, // 是否显示ios支付窗口

loading: false, // 支付中

}

},

methods: {

// 立即支付

orderPay(order) { this.showHidePay()

},

// 显示和隐藏ios支付窗口

showHidePay() {

this.isShow = !this.isShow

},

}

}

</script>

<style lang="scss">

.order-item {

}

.bottom-ios {

z-index: 100; position: fixed; left: 0;

right: 0;

bottom: 0;

background-color: #FFFFFF; padding: 30rpx 40rpx; border-top: $mxg-underline;

.title {

font-size: 38rpx; text-align: center; margin-bottom: 20rpx; text:first-child {

flex: 1;

// 左侧占用取消按钮的宽度，居中

margin-left: 70rpx;

}

text:last-child {

color: $mxg-text-color-grey; font-size: 35rpx;

}

}

.price {

font-size: 30rpx; line-height: 90rpx; text:last-child {

color: $mxg-text-color-red;

}

}

.btn {

background-color: $mxg-color-primary; color: #fff;

border-radius: 50rpx; line-height: 80rpx; font-size: 30rpx; margin-top: 30rpx; &::after {

// 加载中时，隐藏边框

border: none;

}

}

}

</style>

1. 业务逻辑实现

判断是否为APP端的IOS设备，标识 isIOS

是，弹出窗口，查询用户余额渲染， 余额充足：立即扣款支付

余额不足：进入我的余额页面，进行充值

否，进入订单支付页面（/pages/order/order-pay.vue 后面创建）

<script>

import api from '@/api/order.js'

export default { data() {

return {

isIOS: false, // 是否为APP端的IOS设备

balance: 0, // 余 额

order: {} , // 立即支付订单

}

},

onLoad() {

this.loadData()

// #ifdef APP-PLUS

// APP端中获取当前平台ios、android（H5端不获取） this.isIOS = uni.getSystemInfoSync().platform === 'ios'

// #endif

},

computed: {

isPay() { //app端ios平台余额是否足够支付

return this.balance >= (this.order.priceDiscount || this.order.pricePayable)

}

},

methods: {

// 点击订单列表中的 立即支付 按钮

async orderPay(order) { this.order = order

// 1. 判断平台

if(this.isIOS) {

// app端ios平台，弹窗，查询余额显示this.showHidePay()

// 查询余额

this.loading = true

const {data} = await api.getUserBalance() this.balance = data

this.loading = false

}else {

// 非app端ios平台跳转 微信、支付宝选择页面

const price = this.order.priceDiscount || this.order.pricePayable const data = {price, orderId: this.order.orderId} this.navTo(`/pages/order/order-pay?params=${ JSON.stringify(data) }`)

}

},

// app端ios平台：判断余额是否充足，不足提示跳转充值页面async iosPay() {

const price = this.order.priceDiscount || this.order.pricePayable

const data = {price, orderId: this.order.orderId}

if(this.isPay) {

// 余额足够，立即支付this.loading = true uni.showLoading({

title: '支付中，请勿离开此页面', mask: true

})

const res = await api.orderPay(data) uni.hideLoading()

if(res.code === 20000) { this.showHidePay() uni.showToast({

title: '支付成功'

})

// 刷新订单列表

this.loadData()

}else {

uni.showModal({

content: "支付失败,请重试！原因：" + res.message, showCancel: false

})

}

}else {

// 余额不足，跳转充值页

this.navTo(`/pages/order/my-balance?params=${JSON.stringify(data)}`)

}

},

// 显示和隐藏ios支付窗口

showHidePay() {

this.isShow = !this.isShow

},

}

}

</script>

1. 页面模板动态渲染数据: 支付金额，当前余额，按钮文字

<!-- 底部弹窗 -->

<view v-if="isShow" class="mask" catchtouchmove="true" @touchmove.stop.prevent="()=>{}" >

</view>

<view v-if="isShow" class="bottom-ios" catchtouchmove="true" @touchmove.stop.prevent="()=>

{}">

<view class="title center">

<text>确定支付</text>

<text @click="showHidePay">取消</text>

</view>

<view class="price space-between">

<text>支付金额</text>

<text>{{order.priceDiscount || order.pricePayable}}</text>

</view>

<view class="price space-between">

<text>当前余额</text>

<text>{{parseFloat(balance).toFixed(2)}}</text>

</view>

<button class="btn"

@click="iosPay" :disabled="loading" :loading="loading">

{{isPay ? '立即支付': '余额不足，立即充值'}}

</button>

</view>

**订单支付页面功能**

1. 需求：

非app端ios平台，点击 **立即支付** 进入支付页面，选择 支付宝、微信支付，进行立即支付。

1. 创建 和配置订单支付页面

pages/order/order-pay.vue

,{

"path" : "pages/order/order-pay",

"style" :

{

"navigationBarTitleText": "订单支付", "app-plus": {

"bounce":"none"

}

}

}

1. 订单页面模板

<template>

<view >

<view class="card option-pay">

<view class="title">支付方式</view>

<radio-group @change="payChange">

<!-- #ifndef MP-WEIXIN -->

<label class="pay-item center space-between">

<view class="left center">

<image src="/static/pay/alipay.png"></image>

<text>支付宝</text>

</view>

<radio value="alipay" :checked="provider==='alipay'" style="transform:scale(0.8)"/>

</label>

<!-- #endif -->

<!-- #ifndef MP-ALIPAY -->

<label class="pay-item center space-between">

<view class="left center">

<image src="/static/pay/wxpay.png"></image>

<text>微信支付</text>

</view>

<radio value="wxpay" :checked="provider==='wxpay'" style="transform:scale(0.8)"/>

</label>

<!-- #endif -->

</radio-group>

</view>

<view class="card price space-between">

<text>实付金额</text>

<text>￥888.8</text>

</view>

<button class="btn" >立即支付</button>

</view>

</template>

1. 样式编写

<style lang="scss">

.card {

margin: 0 30rpx;

padding: 0 20rpx; background-color: #FFF; margin-top: 30rpx; border-radius: 20rpx;

box-shadow: 1px 0 5px 0 rgba(0, 0, 0, 0.08);

}

.option-pay{

.title {

font-size: 35rpx; line-height: 90rpx;

border-bottom: $mxg-underline;

}

.pay-item {

line-height: 90rpx;

.left {

image{

width: 60rpx; height: 60rpx;

}

text {

font-size: 33rpx; padding-left: 20rpx;

}

}

}

}

.price {

font-size: 30rpx; line-height: 90rpx;

border-bottom: 1px solid #F8F9FB; text:last-child {

color: $mxg-text-color-red;

}

}

.btn {

margin: 60rpx 30rpx;

background-color: $mxg-color-primary; color: #fff;

line-height: 80rpx; font-size: 30rpx; &::after {

// 加载中时，隐藏边框

border: none;

}

}

</style>

1. 获取页面参数，用于提交支付时，作为请求参数获取预支付订单信息，

<view class="card price space-between">

<text>实付金额</text>

<text>￥{{data.price}}</text>

</view>

<script>

export default { data() {

return {

data: { //支付数据price: 0, orderId: null

},

}

},

onLoad(option) {

//获取支付数据

if(option.params) this.data = JSON.parse(option.params)

},

}

</script>

1. 创建关于支付的混合文件 ，

/pages/order/mixins/pay.js

封装支付相关的业务代码

相关支付代码参考 conﬁrm-buy.vue 页面，**其中获取预支付订单信息的请求参数data，在data选项中进 行声明 data属性，这样引入了此混合文件，可动态传递不同值。**

资料参考：\03-配套资料\mixins\pay.js

import api from '@/api/order.js'

export default { data() {

return {

provider: 'alipay', // 支付方式,即提供商

loading: false, // 是否提交中

data: {} , //订单信息服务接口请求参数 ++++

}

},

onLoad(option) {

// #ifdef MP-WEIXIN

// 微信小程序选中微信支付

this.provider = 'wxpay'

// #endif

},

methods: {

// 选择支付方式

payChange(e) {

this.provider = e.detail.value

},

// 微信支付、支付宝操作

async payHandler() {

// 支付中

this.loading = true

// #ifdef APP-PLUS

// 1. 发送请求到服务端，服务端生成订单信息，响应预支付订单号

let orderInfo = await this.getOrderInfo(); if (!orderInfo) {

uni.showModal({

content: '获取支付信息失败', showCancel: false

})

return

}

// 2. 请求支付

uni.requestPayment({

provider: this.provider, // 支付渠道

orderInfo: orderInfo, success: (e) => {

console.log("success", e); uni.showToast({

title: "支付成功!"

})

// 回到订单页面

this.navTo('/pages/order/order')

},

fail: (e) => { console.log("fail", e); uni.showModal({

content: "支付失败，请重试。",

showCancel: false

})

},

complete: () => { this.loading = false;

}

})

// #endif

},

// 获取订单信息 (微信小程序支付需要openid) getOrderInfo(openid) {

// 不要少了async

return new Promise( async (resolve, reject) => { if(openid) {

// 微信小程序支付， 发送请求到服务器获取订单信息，参考：

https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2\_8\_2.shtml let orderInfo = {

"timeStamp": "1414561699",

"nonceStr": "5K8264ILTKCH16CQ2502SI8ZNMTM67VS",

"package": "prepay\_id=wx201410272009395522657a690389285100", //预支付交易会话标识（prepay\_id）

"signType": "RSA",

"paySign":

"oR9d8PuhnIc+YZ8cBHFCwfgpaK9gd7vaRvkYD7rthRAZ\/X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+lwSN22f5YZvI45M Lko8Pfso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GfE1Wfl2gHxIIY5lLdUgWFts17D4WuolLLkiFZV+ JSHMvH7eaLdT9N5GBovBwu5yYKUR7skR8Fu+LozcSqQixnlEZUfyE55feLOQTUYzLmR9pNtPbPsu6WVhbNHMS3Ss2+A ehHvz+n64GDmXxbX++IOBvm2olHu3PsOUGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\/xDg==",

}

resolve(orderInfo) return

}

let res = null

console.log('订单信息请求参数：', this.data) if(this.provider === 'alipay') {

res = await api.getOrderInfoAlipay(this.data)

}else if(this.provider === 'wxpay') {

res = await api.getOrderInfoWxpay(this.data)

}

if (res && res.code === 20000) { resolve(res.data)

} else {

reject(new Error('获取支付信息失败' + res.message))

}

})

},

// 微信小程序支付

async wxPayHandler() { this.loading = true

// 1. 先获取用户code, 再获取openid

let openid = uni.getStorageSync('openid') if (!openid) {

try {

openid = await this.loginWeixinMp()

} catch (e) {

console.error(e)

}

if (!openid) { uni.showModal({

content: '获取openid失败',

showCancel: false

})

this.loading = false return

}

}

// 2. 通过 openid 再获取订单信息，

let orderInfo = await this.getOrderInfo(openid)

// 3. 通过订单预支付信息，去调用支付接口

uni.requestPayment({

...orderInfo, success: (res) => {

uni.showToast({

title: "支付成功!"

})

},

fail: (res) => { uni.showModal({

content: "支付失败,原因为: " + res.errMsg,

showCancel: false

})

},

complete: () => { this.loading = false;

}

})

},

// 登录微信小程序

loginWeixinMp (){

// 微信小程序登录参考：

https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/login.html return new Promise((resolve, reject) => {

// 1. 先使用微信登录小程序响应 code，

uni.login({

provider: 'weixin', success: (res) => {

console.log('登录', res)

const code = res.code

// 2. 请求服务端通过code获取openid let openid = 'xx' uni.setStorageSync('openid', openid) resolve(openid)

},

fail(err) {

reject(err)

}

})

})

},

}

}

1. 在 order-pay.vue 页面导入与引用 pay.js 混合文件即可

<script>

import payMixin from './mixins/pay.js'

export default { mixins: [payMixin],

data() {

return {

data: { //支付数据

price: 0,

orderId: null

},

}

},

onLoad(option) {

//获取支付数据

if(option.params) this.data = JSON.parse(option.params)

},

methods: {

}

}

</script>

1. APP端IOS平台测试支付，将 order.vue 中 orderPay 方法判断平台时，加上叹号去掉

!this.isIOS

，测试完后把感

// 立即支付

async orderPay(order) { this.order = order

// 1. 判断平台

if( !this.isIOS ) {