智能移动开发小程序说明文档

软件学院 1813008 刘宇鑫

1.概述

1.1开发目标与意义

在疫情防控常态化、实体行业不景气、电子商务持续发展的当下，结合本次2022微信小程序应用开发赛所选的“用科技创造社会价值与助力乡村振兴”的主题，我决定开发一款可以专注于解除乡村特产等商品所受到的时空局限性、打通城市乡村之间销售渠道的线上商城小程序，创造社会价值、助力经济发展。

1.2开发环境

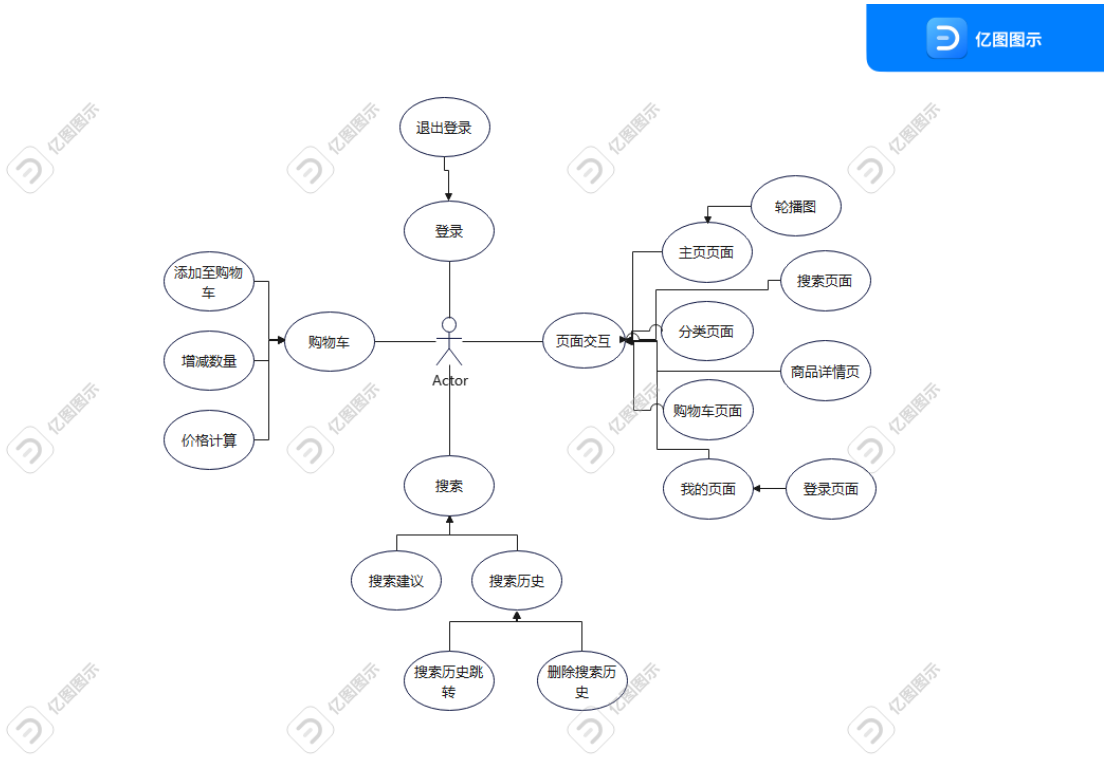
前端框架：uni-app

开发平台：HBuilderX、微信开发者工具

系统环境：Windows 10 专业版 19044.1706

部分商品素材来源接口：https://api-ugo-web.itheima.net/

1.3功能与需求（UML图展示）



2.迭代一

1.tabBar的实现

在 pages 目录中，创建了首页(home)、分类(cate)、购物车(cart)、我的(my) 4 个页面；

修改pages.json配置文件，新增配置节点并修改globalStyle节点以修改导航条的样式效果：

2.轮播图区域的实现

在 data 中定义轮播图的数组swiperList: []；

在 methods 中定义获取轮播图数据的方法getSwiperList()，发起请求并在请求成功后为 data 中的数据赋值；

在 onLoad 生命周期函数中调用getSwiperList()；

渲染轮播图的 UI 结构，使用v-for循环渲染轮播图的item 项，动态绑定图片的 src 属性；

将swiper-item节点内的view 组件，改造为 navigator 导航组件，并动态绑定 url 属性的值，为之后点击跳转至商品详情页做准备：

```
<navigator class="swiper-item" :url="'/subpkg/goods_detail/goods_detail?
goods_id=' + item.goods_id">
      <image :src="item.image_src"></image>
</navigator>
```

3.分类导航区域的实现

在 data 中定义分类导航的数据列表navList: []；

在 methods 中定义获取数据的方法getNavList()；

在 onLoad 生命周期函数中调用getSwiperList()；

定义 navClickHandler 事件处理函数，并为 nav-item 绑定，实现点击第一项，切换到分类页面

```
// nav-item 项被点击时候的事件处理函数
navClickHandler(item) {
  // 判断点击的是哪个 nav
  if (item.name === '分类') {
    uni.switchTab({
      url: '/pages/cate/cate'
    })
  }
```

3.迭代二

1.分类页面

1.1基本结构

定义左侧的滚动视图区域"left-scroll-view"和右侧的滚动视图区域"right-scroll-view"；

动态计算剩余高度以适应窗口

```
// 获取当前系统的信息
const sysInfo = uni.getSystemInfoSync()
// 为 wh 窗口可用高度动态赋值
this.wh = sysInfo.windowHeight
```

1.2获取分类数据

在 data 中定义分类数据列表cateList: []；

在 methods 中定义获取数据的方法getCateList()；

在 onLoad 生命周期函数中调用getCateList()；

1.3左侧一级分类

循环渲染结构时，为选中项动态添加 .active 类名i === active ? 'active' : ''，为一级分类的 Item 项绑定点击事件处理函数 activeChanged，实现动态修改选中项的索引

```
activeChanged(i) {
    this.active = i
  }
```

在 data 中定义 滚动条距离顶部的距离scrollTop: 0，动态为右侧的 组件绑定 scroll-top 属性的值，修改 activeChanged 方法以实现切换一级分类后重置滚动条的位置

```
this.scrollTop = this.scrollTop ? 0 : 1
```

1.4右侧二级分类

定义二级分类列表cateLevel2: [];

分别修改 getCateList 方法实现为二级分类列表数据赋值，activeChanged 方法实现在一级分类选中项改变之后，为二级分类列表数据重新赋值

```
// 为二级分类赋值
this.cateLevel2 = res.message[0].children
// 为二级分类列表重新赋值
this.cateLevel2 = this.cateList[i].children
```

2.搜索功能

1.自定义搜索组件

自定义uni-app组件 my-search；

实现input 输入框的ui

```
<view class="my-search-box">
      <uni-icons type="search" size="17"></uni-icons>
      <text class="placeholder">搜索</text>
</view>
```

为实现点击功能，在组件内部，给my-search-box 的 view 绑定 click 事件处理函数 @click="searchBoxHandler"

```
searchBoxHandler() {
    this.$emit('click')
  }
```

在分类页面中使用搜索组件时，为其绑定点击事件处理函数@click="gotoSearch"跳转到分包中的搜索页面

```
gotoSearch() {
     uni.navigateTo({
       url: '/subpkg/search/search'
     })
}
```

2.搜索建议

使用 uni-ui 提供的搜索组件uni-search-bar，定义输入事件处理函数input(e);

为准确记录搜索历史，需实现搜索框的防抖处理，定义防抖的延时器 timerId，修改input 事件处理函数

```
// 清除延时器
clearTimeout(this.timer)
// 重新启动一个延时器，并把 timerId 赋值给 this.timer
this.timer = setTimeout(() => {
// 如果 500 毫秒内，没有触发新的输入事件，则为搜索关键词赋值
this.kw = e
console.log(this.kw)
}, 500)
```

为根据关键词查询搜索建议列表，定义搜索关键词kw和搜索结果列表searchResults: []，定义getSearchList方法获取搜索建议列表

```
// 判断关键词是否为空
  if (this.kw === '') {
    this.searchResults = []
    return
  }
// 发起请求，获取搜索建议列表
  const { data: res } = await uni.$http.get('qsearch', { query: this.kw })
  if (res.meta.status !== 200) return uni.$showMsg()
  this.searchResults = res.message
```

为实现点击搜索建议的商品跳转到商品详情页面，定义事件处理函数gotoDetail()

```
gotoDetail(goods_id) {
  uni.navigateTo({
    url: '/subpkg/goods_detail/goods_detail?goods_id=' + goods_id
  })
}
```

3.搜索历史

定义搜索关键词的历史记录historyList: [];

当搜索结果列表的长度不为 0的时候，需要展示搜索建议区域，隐藏搜索历史区域;

当搜索结果列表的长度等于 0的时候，则需要隐藏搜索建议区域，展示搜索历史区域

使用 v-if 和 v-else实现搜索建议和搜索历史的按需展示

```
//搜索建议列表
<view class="sugg-list" v-if="searchResults.length !== 0"></view>
//搜索历史
<view class="history-box" v-else></view>
```

为将搜索关键词存入 historyList，定义保存搜索关键词的方法saveSearchHistory()，但直接push有两个问题，一是关键词前后顺序的问题，二是关键词重复的问题

```
saveSearchHistory() {
    this.historyList.push(this.kw)
  }
```

定义一个计算属性 historys，其值为historyList 数组 reverse后以解决关键字前后顺序的问题

```
computed: {
  historys() {
    return [...this.historyList].reverse()
  }
}
```

使用 Set 对象解决关键词重复的问题，修改 saveSearchHistory 方法

```
//Array 数组转化为 Set 对象
const set = new Set(this.historyList)
//调用 Set 对象的 delete 方法，移除对应的元素
set.delete(this.kw)
//调用 Set 对象的 add 方法，向 Set 中添加元素
set.add(this.kw)
//将 Set 对象转化为 Array 数组
this.historyList = Array.from(set)
```

实现清空搜索历史记录功能，定义 cleanHistory 处理函数

```
cleanHistory() {
  // 清空 data 中保存的搜索历史
  this.historyList = []
  // 清空本地存储中记录的搜索历史
  uni.setStorageSync('kw', '[]')
}
```

实现点击搜索历史跳转到商品列表页面，定义 gotoGoodsList 处理函数

```
gotoGoodsList(kw) {
  uni.navigateTo({
    url: '/subpkg/goods_list/goods_list?query=' + kw
  })
}
```

4.迭代三

1.商品列表

根据接口的要求，事先定义一个请求参数对象queryObj，并将页面跳转时携带的参数，转存到queryObj 对象中

```
this.queryObj.query = options.query || ''
this.queryObj.cid = options.cid || ''
```

1. 获取商品列表数据

定义商品列表的数据goodsList: [];
定义总数量total，用来实现分页；

定义获取商品列表数据的方法getGoodsList();

为了防止某些商品的图片不存在，定义一个默认的图片defaultPic；

为将价格数字处理为带两位小数点的数字，定义filters 过滤器节点

```
filters: {
  tofixed(num) {
    return Number(num).toFixed(2)
  }
}
```

2.实现上拉加载更多

在 goods_list 页面中声明 onReachBottom 事件处理函数，用来监听页面的上拉触底行为

```
onReachBottom() {
  // 让页码值自增 +1
  this.queryObj.pagenum += 1
  // 重新获取列表数据
  this.getGoodsList()
}
```

修改getGoodsList 函数，当列表数据请求成功之后，进行新旧数据的拼接处理

```
// 为数据赋值：通过展开运算符的形式，进行新旧数据的拼接
this.goodsList = [...this.goodsList, ...res.message.goods]
```

定义 isloading 节流阀防止发起额外的请求，在getGoodsList 方法中，请求数据前后分别打开和关闭节流阀

```
//打开节流阀
this.isloading = true
// ** 关闭节流阀
this.isloading = false
```

修改onReachBottom函数，根据节流阀的状态，来决定是否发起请求，并判断数据是否加载完毕

```
// 判断是否正在请求其它数据
if (this.isloading) return
// 判断是否还有下一页数据
if (this.queryObj.pagenum * this.queryObj.pagesize >= this.total) return
uni.$showMsg('数据加载完毕! ')
```

3.实现下拉刷新

修改pages.json 配置文件，为goods_list 页面开启下拉刷新效果

```
"enablePullDownRefresh": true
```

定义监听页面的 onPullDownRefresh 事件处理函数

```
onPullDownRefresh() {
  // 1. 重置关键数据
  this.queryObj.pagenum = 1
  this.total = 0
  this.isloading = false
  this.goodsList = []
  // 2. 重新发起请求
  this.getGoodsList(() => uni.stopPullDownRefresh())
}
```

修改 getGoodsList 函数，接收 cb 回调函数并按需进行调用：cb && cb()

4.点击商品 item 项跳转到详情页面

定义 gotoDetail 事件处理函数

```
gotoDetail(item) {
  uni.navigateTo({
    url: '/subpkg/goods_detail/goods_detail?goods_id=' + item.goods_id
  })
}
```

将循环时的 block 组件修改为 view 组件，并绑定 click 点击事件处理函数@click="gotoDetail(item)

2.商品详情页

1.获取商品详情数据

定义商品详情的数据节点goods_info: {}

在 onLoad 中获取商品的 Id，并调用请求商品详情的方法getGoodsDetail()

```
// 获取商品 Id
const goods_id = options.goods_id
// 调用请求商品详情数据的方法
this.getGoodsDetail(goods_id)
```

声明 getGoodsDetail 方法

```
async getGoodsDetail(goods_id) {
    const { data: res } = await uni.$http.get('/api/public/v1/goods/detail', {
goods_id })
    if (res.meta.status !== 200) return uni.$showMsg()
    // 为 data 中的数据赋值
    this.goods_info = res.message
  }
```

2.渲染商品详情页的 UI 结构

实现轮播图预览效果,为轮播图中的 image 图片绑定 click 事件处理函数@click="preview(i)"

定义 preview 事件处理函数

```
preview(i) {
  // 调用 uni.previewImage() 方法预览图片
  uni.previewImage({
    // 预览时，默认显示图片的索引
    current: i,
    // 所有图片 url 地址的数组
    urls: this.goods_info.pics.map(x => x.pics_big)
  })
}
```

渲染商品信息区域、渲染商品详情信息，修改 getGoodsDetail 方法，解决图片底部存在空白间隙的问题

```
// 使用字符串的 replace() 方法，为 img 标签添加行内的 style 样式，从而解决图片底部空白间隙的问题
res.message.goods_introduce = res.message.goods_introduce.replace(/<img /g, '<img style="display:block;" ')
```

解决 .webp 格式图片在 ios 设备上无法正常显示的问题

```
// 使用字符串的 replace() 方法，将 webp 的后缀名替换为 jpg 的后缀名
res.message.goods_introduce = res.message.goods_introduce.replace(/<img /g, '<img style="display:block;" ').replace(/webp/g, 'jpg')
```

基于 uni-ui 提供的 GoodsNav 组件来实现底部商品导航区域的效果，定义options 和 buttonGroup 两个数组，来声明商品导航组件的按钮配置对象

```
<uni-goods-nav :fill="true" :options="options" :buttonGroup="buttonGroup" @click="onClick" @buttonClick="buttonClick" />
```

定义onClick处理函数实现点击跳转到购物车页面

```
onClick(e) {
  if (e.content.text === '购物车') {
    // 切换到购物车页面
    uni.switchTab({
      url: '/pages/cart/cart'
    })
  }
}
```

5.迭代四

1.加入购物车功能

导入 Vue 和 Vuex， 并将 Vuex 安装为 Vue 的插件Vue.use(Vuex)；

初始化 Store 的实例对象

```
//创建 Store 的实例对象
const store = new Vuex.Store({
  // TODO：挂载 store 模块
  modules: {},
})

//向外共享 Store 的实例对象
export default store
```

在购物车的 store 模块cart.js中，使用state存储购物车中每个商品的信息对象

```
// 每个商品的信息对象，都包含如下 6 个属性：
// { goods_id, goods_name, goods_price, goods_count, goods_small_logo,
goods_state }
```

为在商品详情页中使用 Store 中的数据，需要从 vuex 中按需导出 mapState 辅助方法，调用 mapState 方法，就可以把 m_cart 模块中的 cart 数组映射到当前页面中，作为计算属性来使用

1.实现加入购物车的功能

定义一个将商品信息加入购物车的 mutations 方法addToCart()

```
addToCart(state, goods) {
      // 根据提交的商品的Id，查询购物车中是否存在这件商品
      const findResult = state.cart.find((x) => x.goods_id === goods.goods_id)

      if (!findResult) {
        state.cart.push(goods)
      } else {
        findResult.goods_count++
      }
}
```

为商品导航组件 uni-goods-nav 绑定 @buttonClick="buttonClick" 事件处理函数：

```
buttonClick(e) {
   //判断是否点击按钮
   if (e.content.text === '加入购物车') {

      //商品的信息对象
      const goods = {
         goods_id: this.goods_info.goods_id,
         goods_name: this.goods_info.goods_name,
         goods_price: this.goods_info.goods_price,
         goods_count: 1,
         goods_small_logo: this.goods_info.goods_small_logo,
         goods_state: true
      }

      //通过 this 调用 addToCart 方法，把商品信息对象存储到购物车中
      this.addToCart(goods)

   }
}
```

2.动态统计购物车中商品的总数量

定义一个用来统计购物车中商品的总数量的方法total()

```
total(state) {
      let c = 0
      state.cart.forEach(goods => c += goods.goods_count)
      return c
}
```

通过 watch 侦听器，监听计算属性 total 值的变化，从而动态为购物车按钮的徽标赋值

```
// 定义 total 侦听器，指向一个配置对象
total: {
      handler(newVal) {
         const findResult = this.options.find(x => x.text === '购物车')
         if (findResult) {
            findResult.info = newVal
         }
      },
      // immediate 属性用来声明此侦听器，是否在页面初次加载完毕后立即调用
      immediate: true
   }
```

2.购物车页面

通过 mapState 辅助函数，将 Store 中的 cart 数组映射到当前页面中使用

1.商品勾选状态

为商品的左侧图片区域添加 radio 组件

封装控制当前组件中是否显示 radio 组件的props 属性showRadio

```
showRadio: {
      type: Boolean,
      // 如果外界没有指定 show-radio 属性的值，则默认不展示 radio 组件
      default: false,
}
```

使用 v-if 指令控制 radio 组件的按需展示

```
<radio :checked="goods.goods_state" color="#C00000" v-if="showRadio"></radio>
```

为实现修改当前商品的勾选状态功能，为 my-goods 组件绑定 @radio-change 事件，定义 radioChangeHandler 事件处理函数和radioClickHandler 事件处理函数，更新购物车中商品的勾选状态的mutations 方法updateGoodsState

```
radioChangeHandler(e) {
      this.updateGoodsState(e)
}
```

```
radioClickHandler() {
    // 通过 this.$emit() 触发外界通过 @ 绑定的 radio-change 事件，
    // 同时把商品的 Id 和 勾选状态 作为参数传递给 radio-change 事件处理函数
    this.$emit('radio-change', {
      goods_id: this.goods.goods_id,
      goods_state: !this.goods.goods_state
    })
  }
```

```
updateGoodsState(state, goods) {
  // 根据 goods_id 查询购物车中对应商品的信息对象
  const findResult = state.cart.find(x => x.goods_id === goods.goods_id)
  // 有对应的商品信息对象
  if (findResult) {
    // 更新对应商品的勾选状态
    findResult.goods_state = goods.goods_state
  }
```

2.商品数量

在goods-info-box 的 view 组件内部渲染 NumberBox 组件的基本结构，在 my-goods.vue 组件中，封装名称为 showNum 的 props 属性，来控制当前组件中是否显示 NumberBox 组件；

为实现用户修改了 NumberBox 的值后将最新的商品数量更新到购物车中，为 my-goods 组件绑定 @num-change 事件，定义 numChangeHandler 事件处理函数

```
numChangeHandler(val) {
    // 通过 this.$emit() 触发外界通过 @ 绑定的 num-change 事件
    this.$emit('num-change', {
      // 商品的 Id
      goods_id: this.goods.goods_id,
      // 商品的最新数量
      goods_count: +val
    })
  }
```

为防止用户输入 NumberBox 数据不合法，修改_onBlur 函数

```
// 将用户输入的内容转化为整数
  let value = parseInt(event.detail.value);
  if (!value) {
    //如果转化之后的结果为 NaN，则给定默认值为 1
    this.inputValue = 1;
    return;
  }
```

为修改购物车中商品的数量，声明mutations 方法updateGoodsCount

```
updateGoodsCount(state, goods) {
  // 根据 goods_id 查询购物车中对应商品的信息对象
  const findResult = state.cart.find(x => x.goods_id === goods.goods_id)
  if(findResult) {
    // 更新对应商品的数量
    findResult.goods_count = goods.goods_count
  }
}
```

3.已勾选商品的总数量

在 store/cart.js 模块中，定义一个名称为 checkedCount 的 getters

```
checkedCount(state) {
  // 先使用 filter 方法，从购物车中过滤器已勾选的商品
  // 再使用 reduce 方法，将已勾选的商品总数量进行累加
  // reduce() 的返回值就是已勾选的商品的总数量
  return state.cart.filter(x => x.goods_state).reduce((total, item) => total +=
item.goods_count, 0)
}
```

4.全选按钮的选中状态

定义一个叫做 isFullCheck 的计算属性

```
 isFullCheck() {
      return this.total === this.checkedCount
 }
```

5.商品的全选/反选功能

定义一个叫做 updateAllGoodsState 的 mutations 方法，用来修改所有商品的勾选状态

```
updateAllGoodsState(state, newState) {
  // 循环更新购物车中每件商品的勾选状态
  state.cart.forEach(x => x.goods_state = newState)
}
```

在 my-settle 组件的 methods 节点中，声明 changeAllState 事件处理函数

```
changeAllState() {
    // 修改购物车中所有商品的选中状态
    this.updateAllGoodsState(!this.isFullCheck)
  }
```

6.已勾选商品的总价格

定义一个叫做 checkedGoodsAmount 的 getters，用来统计已勾选商品的总价格

```
checkedGoodsAmount(state) {
  return state.cart.filter(x => x.goods_state)
                   .reduce((total, item) => total += item.goods_count *
item.goods_price, 0)
                   .toFixed(2)
}
```

7.购物车徽标的数值

存在问题：当修改购物车中商品的数量之后，tabBar 上的数字徽标不会自动更新；

使用 watch 侦听器，监听 total 总数量的变化

```
watch: {
   total() {
     // 调用 methods 中的 setBadge 方法，重新为 tabBar 的数字徽章赋值
     this.setBadge()
   },
 }
```

3.登录页面

使用v-if/v-else实现登录组件和用户信息组件的按需展示

```
<!-- 用户未登录时，显示登录组件 -->
<my-login v-if="!token"></my-login>
<!-- 用户登录后，显示用户信息组件 -->
<my-userinfo v-else></my-userinfo>
```

使用微信账号信息登录，为登录按钮绑定@click="getUserProfile"事件处理函数，定义获取微信用户的基本信息的函数getUserProfile()

```
getUserProfile() {
        uni.getUserProfile({
          desc: '你的授权信息',
          success: (res) => {
            // 将信息存到 vuex 中
            this.updateUserInfo(res.userInfo)
            this.getToken(res)
          },
          fail: (res) => {
            return uni.$showMsg('您取消了登录授权')
          }
        })
     }
```

在 store/user.js 模块的 mutations 节点中，声明更新用户的基本信息的方法updateUserInfo和将 userinfo 持久化存储到本地的方法saveUserInfoToStorage

```
updateUserInfo(state, userinfo) {
    state.userinfo = userinfo
    // 通过 this.commit() 方法，调用 m_user 模块下的 saveUserInfoToStorage 方法，将
userinfo 对象持久化存储到本地
    this.commit('m_user/saveUserInfoToStorage')
  },
将 userinfo 持久化存储到本地
  saveUserInfoToStorage(state) {
    uni.setStorageSync('userinfo', JSON.stringify(state.userinfo))
  }
```

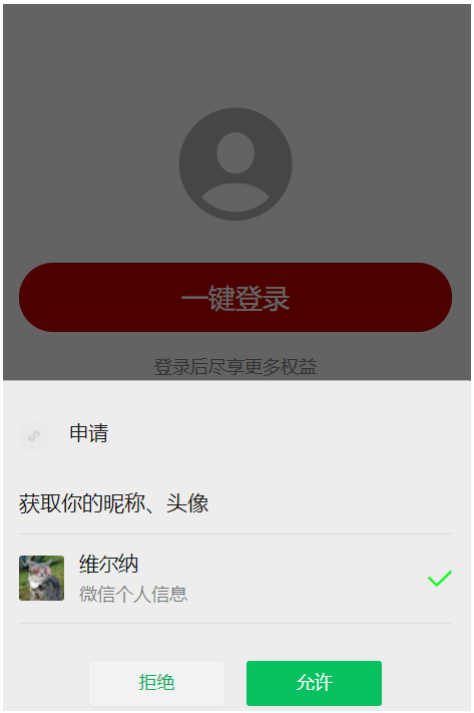
当获取到了微信用户的基本信息之后，还需要进一步调用登录相关的接口，从而换取登录成功之后的 Token 字符串；

定义 getToken 方法，调用登录相关的 API，实现登录的功能

```
async getToken(info) {
  // 调用微信登录接口
  const [err, res] = await uni.login().catch(err => err)
  // 判断是否 uni.login() 调用失败
  if (err || res.errMsg !== 'login:ok') return uni.$showError('登录失败！')

  // 准备参数对象
  const query = {
    code: res.code,
    encryptedData: info.encryptedData,
    iv: info.iv,
    rawData: info.rawData,
    signature: info.signature
  }

  // 换取 token
  const { data: loginResult } = await
uni.$http.post('/api/public/v1/users/wxlogin', query)
  if (loginResult.meta.status !== 200) return uni.$showMsg('登录失败！')
  uni.$showMsg('登录成功')
}
```



4.用户信息页面

1.渲染用户的头像和昵称

在 my-userinfo 组件中，通过 mapState 辅助函数，将需要的成员映射到当前组件中使用，将用户的头像和昵称渲染到页面中

```
<image :src="userinfo.avatarUrl" class="avatar"></image>
<view class="nickname">{{userinfo.nickName}}</view>
```

2.实现退出登录的功能

为退出登录项绑定 click 点击事件处理函数@click="logout"，定义 logout 事件处理函数

```
async logout() {
  // 询问用户是否退出登录
  const [err, succ] = await uni.showModal({
    title: '提示',
    content: '确认退出登录吗？'
  }).catch(err => err)

  if (succ && succ.confirm) {
     // 用户确认了退出登录的操作
     // 需要清空 vuex 中的 userinfo token
     this.updateUserInfo({})
     this.updateToken('')
  }
}
```

6.补充说明

AppID(小程序ID)　wx0365bdc5d39ef284

体验版二维码

小程序使用的部分商品价格、图片等素材来自互联网公开接口，但可能由于接口网站证书过期等原因，导致这些素材除在本地调试外，在真机与小程序线上可能无法显示。