

## 9. 购物车页面

### 9.3 结算区域

#### 9.3.1 把结算区域封装为组件

1. 在 `components` 目录中, 新建 `my-settle` 结算组件:



2. 初始化 `my-settle` 组件的基本结构和样式:

```
1 <template>
2   <!-- 最外层的容器 -->
3   <view class="my-settle-container">
4     结算组件
5   </view>
6 </template>
7
8 <script>
9   export default {
10     data() {
11       return {}
12     },
13   }
14 </script>
15
16 <style lang="scss">
17   .my-settle-container {
18     /* 底部固定定位 */
19     position: fixed;
20     bottom: 0;
21     left: 0;
```

```

22     /* 设置宽高和背景色 */
23     width: 100%;
24     height: 50px;
25     background-color: cyan;
26 }
27 </style>

```

3. 在 `cart.vue` 页面中使用自定义的 `my-settle` 组件，并美化页面样式，防止页面底部被覆盖：

```

1     <template>
2       <view class="cart-container">
3         <!-- 使用自定义的 address 组件 -->
4
5         <!-- 购物车商品列表的标题区域 -->
6
7         <!-- 商品列表区域 -->
8
9         <!-- 结算区域 -->
10        <my-settle></my-settle>
11      </view>
12    </template>
13
14    <style lang="scss">
15      .cart-container {
16        padding-bottom: 50px;
17      }
18    </style>

```

### 9.3.2 渲染结算区域的结构和样式

1. 定义如下的 UI 结构：

```

1     <!-- 最外层的容器 -->
2     <view class="my-settle-container">
3       <!-- 全选区域 -->
4       <label class="radio">
5         <radio color="#C00000" :checked="true" /><text>全选</text>
6       </label>
7
8       <!-- 合计区域 -->
9       <view class="amount-box">
10        合计:<text class="amount">¥1234.00</text>
11      </view>
12
13      <!-- 结算按钮 -->
14      <view class="btn-settle">结算(0)</view>
15    </view>

```

2. 美化样式：

```

1     .my-settle-container {
2       position: fixed;
3       bottom: 0;
4       left: 0;
5       width: 100%;
6       height: 50px;

```

```

7      // 将背景色从 cyan 改为 white
8      background-color: white;
9      display: flex;
10     justify-content: space-between;
11     align-items: center;
12     padding-left: 5px;
13     font-size: 14px;
14
15     .radio {
16         display: flex;
17         align-items: center;
18     }
19
20     .amount {
21         color: #c00000;
22     }
23
24     .btn-settle {
25         height: 50px;
26         min-width: 100px;
27         background-color: #c00000;
28         color: white;
29         line-height: 50px;
30         text-align: center;
31         padding: 0 10px;
32     }
33 }

```

### 9.3.3 动态渲染已勾选商品的总数量

1. 在 `store/cart.js` 模块中，定义一个名称为 `checkedCount` 的 getters，用来统计已勾选商品的总数量：

```

1      // 勾选的商品的总数量
2      checkedCount(state) {
3          // 先使用 filter 方法，从购物车中过滤器已勾选的商品
4          // 再使用 reduce 方法，将已勾选的商品总数量进行累加
5          // reduce() 的返回值就是已勾选的商品的总数量
6          return state.cart.filter(x => x.goods_state).reduce((total, item) =>
7              total += item.goods_count, 0)
8      }

```

2. 在 `my-settle` 组件中，通过 `mapGetters` 辅助函数，将需要的 getters 映射到当前组件中使用：

```

1      import { mapGetters } from 'vuex'
2
3      export default {
4          computed: {
5              ...mapGetters('m_cart', ['checkedCount']),
6          },
7          data() {
8              return {}
9          },
10     }

```

3. 将 `checkedCount` 的值渲染到页面中：

```
1 <!-- 结算按钮 -->
2 <view class="btn-settle">结算({{checkedCount}})</view>
```

### 9.3.4 动态渲染全选按钮的选中状态

1. 使用 `mapGetters` 辅助函数，将商品的总数量映射到当前组件中使用，并定义一个叫做 `isFullCheck` 的计算属性：

```
1 import { mapGetters } from 'vuex'
2
3 export default {
4   computed: {
5     // 1. 将 total 映射到当前组件中
6     ...mapGetters('m_cart', ['checkedCount', 'total']),
7     // 2. 是否全选
8     isFullCheck() {
9       return this.total === this.checkedCount
10    },
11  },
12  data() {
13    return {}
14  },
15 }
```

2. 为 radio 组件动态绑定 `checked` 属性的值：

```
1 <!-- 全选区域 -->
2 <label class="radio">
3   <radio color="#C00000" :checked="isFullCheck" /><text>全选</text>
4 </label>
```

### 9.3.5 实现商品的全选/反选功能

1. 在 `store/cart.js` 模块中，定义一个叫做 `updateAllGoodsState` 的 mutations 方法，用来修改所有商品的勾选状态：

```
1 // 更新所有商品的勾选状态
2 updateAllGoodsState(state, newState) {
3   // 循环更新购物车中每件商品的勾选状态
4   state.cart.forEach(x => x.goods_state = newState)
5   // 持久化存储到本地
6   this.commit('m_cart/saveToStorage')
7 }
```

2. 在 `my-settle` 组件中，通过 `mapMutations` 辅助函数，将需要的 mutations 方法映射到当前组件中使用：

```

1 // 1. 按需导入 mapMutations 辅助函数
2 import { mapGetters, mapMutations } from 'vuex'
3
4 export default {
5   // 省略其它代码
6   methods: {
7     // 2. 使用 mapMutations 辅助函数, 把 m_cart 模块提供的
    updateAllGoodsState 方法映射到当前组件中使用
8     ...mapMutations('m_cart', ['updateAllGoodsState']),
9   },
10 }

```

3. 为 UI 中的 `label` 组件绑定 `click` 事件处理函数:

```

1 <!-- 全选区域 -->
2 <label class="radio" @click="changeAllState">
3   <radio color="#C00000" :checked="isFullCheck" /><text>全选</text>
4 </label>

```

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

```

1 methods: {
2   ...mapMutations('m_cart', ['updateAllGoodsState']),
3   // label 的点击事件处理函数
4   changeAllState() {
5     // 修改购物车中所有商品的选中状态
6     // !this.isFullCheck 表示: 当前全选按钮的状态取反之后, 就是最新的勾选状态
7     this.updateAllGoodsState(!this.isFullCheck)
8   }
9 }

```

### 9.3.6 动态渲染已勾选商品的总价格

1. 在 `store/cart.js` 模块中, 定义一个叫做 `checkedGoodsAmount` 的 `getters`, 用来统计已勾选商品的总价格:

```

1 // 已勾选的商品的总价
2 checkedGoodsAmount(state) {
3   // 先使用 filter 方法, 从购物车中过滤器已勾选的商品
4   // 再使用 reduce 方法, 将已勾选的商品数量 * 单价之后, 进行累加
5   // reduce() 的返回值就是已勾选的商品的总价
6   // 最后调用 toFixed(2) 方法, 保留两位小数
7   return state.cart.filter(x => x.goods_state)
8     .reduce((total, item) => total += item.goods_count *
    item.goods_price, 0)
9     .toFixed(2)
10 }

```

2. 在 `my-settle` 组件中, 使用 `mapGetters` 辅助函数, 把需要的 `checkedGoodsAmount` 映射到当前组件中使用:

```

1 ...mapGetters('m_cart', ['total', 'checkedCount', 'checkedGoodsAmount'])

```

3. 在组件的 UI 结构中, 渲染已勾选的商品的总价:

```

1    <!-- 合计区域 -->
2    <view class="amount-box">
3      合计:<text class="amount">¥ {{checkedGoodsAmount}}</text>
4    </view>

```

### 9.3.7 动态计算购物车徽标的数值

1. **问题说明**：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。
2. **解决方案**：改造 `mixins/tabbar-badge.js` 中的代码，使用 `watch` 侦听器，监听 `total` 总数量的变化，从而动态为 tabBar 的徽标赋值：

```

1    import { mapGetters } from 'vuex'
2
3    // 导出一个 mixin 对象
4    export default {
5      computed: {
6        ...mapGetters('m_cart', ['total']),
7      },
8      watch: {
9        // 监听 total 值的变化
10       total() {
11         // 调用 methods 中的 setBadge 方法，重新为 tabBar 的数字徽章赋值
12         this.setBadge()
13       },
14     },
15     onShow() {
16       // 在页面刚展示的时候，设置数字徽标
17       this.setBadge()
18     },
19     methods: {
20       setBadge() {
21         // 调用 uni.setTabBarBadge() 方法，为购物车设置右上角的徽标
22         uni.setTabBarBadge({
23           index: 2,
24           text: this.total + '', // 注意: text 的值必须是字符串，不能是数字
25         })
26       },
27     },
28   }

```

### 9.3.8 渲染购物车为空时的页面结构

1. 将 `资料` 目录中的 `cart_empty@2x.png` 图片复制到项目的 `/static/` 目录中
2. 改造 `cart.vue` 页面的 UI 结构，使用 `v-if` 和 `v-else` 控制购物车区域和空白购物车区域的按需展示：

```

1    <template>
2      <view class="cart-container" v-if="cart.length !== 0">
3
4        <!-- 使用自定义的 address 组件 -->
5
6        <!-- 购物车商品列表的标题区域 -->
7
8        <!-- 商品列表区域 -->
9

```

```

10     <!-- 结算区域 -->
11
12     </view>
13
14     <!-- 空白购物车区域 -->
15     <view class="empty-cart" v-else>
16         <image src="/static/cart_empty@2x.png" class="empty-img"></image>
17         <text class="tip-text">空空如也~</text>
18     </view>
19 </template>

```

### 3. 美化空白购物车区域的样式:

```

1  .empty-cart {
2      display: flex;
3      flex-direction: column;
4      align-items: center;
5      padding-top: 150px;
6
7      .empty-img {
8          width: 90px;
9          height: 90px;
10     }
11
12     .tip-text {
13         font-size: 12px;
14         color: gray;
15         margin-top: 15px;
16     }
17 }

```

## 9.4 分支的合并与提交

### 1. 将 `cart` 分支进行本地提交:

```

1  git add .
2  git commit -m "完成了购物车的开发"

```

### 2. 将本地的 `cart` 分支推送到码云:

```

1  git push -u origin cart

```

### 3. 将本地 `cart` 分支中的代码合并到 `master` 分支:

```

1  git checkout master
2  git merge cart
3  git push

```

### 4. 删除本地的 `cart` 分支:

```

1  git branch -d cart

```

## 10. 登录与支付

## 10.0 创建 settle 分支

运行如下的命令，基于 `master` 分支在本地创建 `settle` 子分支，用来开发登录与支付相关的功能：

```
1 git checkout -b settle
```

## 10.1 点击结算按钮进行条件判断

说明：用户点击了结算按钮之后，需要先后判断是否勾选了要结算的商品、是否选择了收货地址、是否登录。

1. 在 `my-settle` 组件中，为结算按钮绑定点击事件处理函数：

```
1 <!-- 结算按钮 -->
2 <view class="btn-settle" @click="settlement">结算({{checkedCount}})</view>
```

2. 在 `my-settle` 组件的 `methods` 节点中声明 `settlement` 事件处理函数如下：

```
1 // 点击了结算按钮
2 settlement() {
3   // 1. 先判断是否勾选了要结算的商品
4   if (!this.checkedCount) return uni.$showMsg('请选择要结算的商品!')
5
6   // 2. 再判断用户是否选择了收货地址
7   if (!this.addstr) return uni.$showMsg('请选择收货地址!')
8
9   // 3. 最后判断用户是否登录了
10  if (!this.token) return uni.$showMsg('请先登录!')
11 }
```

3. 在 `my-settle` 组件中，使用 `mapGetters` 辅助函数，从 `m_user` 模块中将 `addstr` 映射到当前组件中使用：

```
1 export default {
2   computed: {
3     ...mapGetters('m_cart', ['total', 'checkedCount',
4       'checkedGoodsAmount']),
5     // addstr 是详细的收货地址
6     ...mapGetters('m_user', ['addstr']),
7     isFullCheck() {
8       return this.total === this.checkedCount
9     },
10  },
11 }
```

4. 在 `store/user.js` 模块的 `state` 节点中，声明 `token` 字符串：

```
1 export default {
2   // 开启命名空间
3   namespaced: true,
4
5   // state 数据
6   state: () => ({
```



```

7      // 收货地址
8      address: JSON.parse(uni.getStorageSync('address') || '{}'),
9      // 登录成功之后的 token 字符串
10     token: '',
11   }},
12
13   // 省略其它代码
14 }

```

5. 在 `my-settle` 组件中, 使用 `mapState` 辅助函数, 从 `m_user` 模块中将 `token` 映射到当前组件中使用:

```

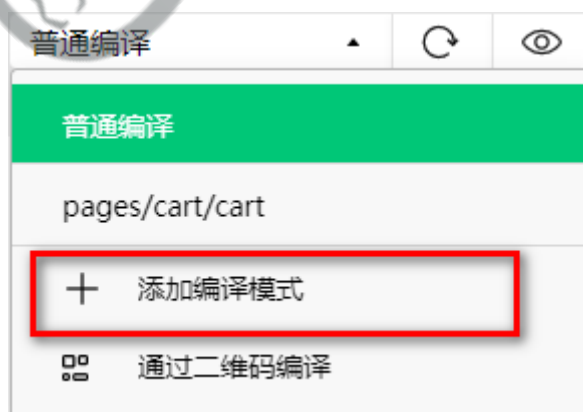
1  // 按需从 vuex 中导入 mapState 辅助函数
2  import { mapGetters, mapMutations, mapState } from 'vuex'
3
4  export default {
5    computed: {
6      ...mapGetters('m_cart', ['total', 'checkedCount',
7        'checkedGoodsAmount']),
8      ...mapGetters('m_user', ['addstr']),
9      // token 是用户登录成功之后的 token 字符串
10     ...mapState('m_user', ['token']),
11     isFullCheck() {
12       return this.total === this.checkedCount
13     },
14   },
15 }

```

## 10.2 登录

### 10.2.1 定义 my 页面的编译模式

1. 点击 `微信开发者工具` 工具栏上的编译模式下拉菜单, 选择 `添加编译模式`:



2. 勾选启动页面的路径之后, 点击确定按钮:

自定义编译条件

解析二维码

仅支持上传png、jpg文件。通过解析二维码可自动生成启动页面和启动参数。

1 模式名称

启动页面

启动参数

进入场景

编译设置 ☐ 下次编译时模拟更新 (需 1.9.90 及以上基础库版本)

2

## 10.2.2 实现登录和用户信息组件的按需展示

1. 在 `components` 目录中新建登录组件：

新建uni-app组件 [自定义模板]

1

'Users' /Desktop/小程序/uni\_proj3/components ▾ 浏览

选择模板

默认模板

使用less的组件

2 ☒ 使用scss的组件

3 ☒ 创建同名目录

4

没有找到想要的？到 [插件市场](#) 看看吧

2. 在 `components` 目录中新建用户信息组件：



3. 在 `my.vue` 页面中, 通过 `mapState` 辅助函数, 导入需要的 `token` 字符串:

```
1 import badgeMix from '@mixins/tabbar-badge.js'
2 // 1. 从 vuex 中按需导入 mapState 辅助函数
3 import { mapState } from 'vuex'
4
5 export default {
6   mixins: [badgeMix],
7   computed: {
8     // 2. 从 m_user 模块中导入需要的 token 字符串
9     ...mapState('m_user', ['token']),
10   },
11   data() {
12     return {}
13   },
14 }
```

4. 在 `my.vue` 页面中, 实现登录组件和用户信息组件的按需展示:

```
1 <template>
2   <view>
3
4     <!-- 用户未登录时, 显示登录组件 -->
5     <my-login v-if="!token"></my-login>
6
7     <!-- 用户登录后, 显示用户信息组件 -->
8     <my-userinfo v-else></my-userinfo>
9
10   </view>
11 </template>
```

### 10.2.3 实现登录组件的基本布局

1. 为 `my-login` 组件定义如下的 UI 结构:

```

1   <template>
2     <view class="login-container">
3       <!-- 提示登录的图标 -->
4       <uni-icons type="contact-filled" size="100" color="#AFAFAF"></uni-
icons>
5       <!-- 登录按钮 -->
6       <button type="primary" class="btn-login">一键登录</button>
7       <!-- 登录提示 -->
8       <view class="tips-text">登录后尽享更多权益</view>
9     </view>
10  </template>

```

## 2. 美化登录组件的样式:

```

1   .login-container {
2     // 登录盒子的样式
3     height: 750rpx;
4     display: flex;
5     flex-direction: column;
6     align-items: center;
7     justify-content: center;
8     background-color: #f8f8f8;
9     position: relative;
10    overflow: hidden;
11
12    // 绘制登录盒子底部的半椭圆造型
13    &::after {
14      content: ' ';
15      display: block;
16      position: absolute;
17      width: 100%;
18      height: 40px;
19      left: 0;
20      bottom: 0;
21      background-color: white;
22      border-radius: 100%;
23      transform: translateY(50%);
24    }
25
26    // 登录按钮的样式
27    .btn-login {
28      width: 90%;
29      border-radius: 100px;
30      margin: 15px 0;
31      background-color: #c00000;
32    }
33
34    // 按钮下方提示消息的样式
35    .tips-text {
36      font-size: 12px;
37      color: gray;
38    }
39  }

```

### 10.2.4 点击登录按钮获取微信用户的基本信息

需求描述：需要获取微信用户的头像、昵称等基本信息。

1. 为登录的 `button` 按钮绑定 `open-type="getUserInfo"` 属性，表示点击按钮时，希望获取用户的基本信息：

```
1 <!-- 登录按钮 -->
2 <!-- 可以从 @getUserInfo 事件处理函数的形参中，获取到用户的基本信息 -->
3 <button type="primary" class="btn-login" open-type="getUserInfo"
  @getUserInfo="getUserInfo">一键登录</button>
```

2. 在 `methods` 节点中声明 `getUserInfo` 事件处理函数如下：

```
1 methods: {
2   // 获取微信用户的基本信息
3   getUserInfo(e) {
4     // 判断是否获取用户信息成功
5     if (e.detail.errMsg === 'getUserInfo:fail auth deny') return
    uni.$showMsg('您取消了登录授权! ')
6
7     // 获取用户信息成功， e.detail.userInfo 就是用户的基本信息
8     console.log(e.detail.userInfo)
9   }
10 }
```

### 10.2.5 将用户的基本信息存储到 vuex

1. 在 `store/user.js` 模块的 `state` 节点中，声明 `userinfo` 的信息对象如下：

```
1 // state 数据
2 state: () => ({
3   // 收货地址
4   // address: {}
5   address: JSON.parse(uni.getStorageSync('address')) || '{}'),
6   // 登录成功之后的 token 字符串
7   token: '',
8   // 用户的基本信息
9   userinfo: JSON.parse(uni.getStorageSync('userinfo')) || '{}')
10 },
```

2. 在 `store/user.js` 模块的 `mutations` 节点中，声明如下的两个方法：

```
1 // 方法
2 mutations: {
3   // 省略其它代码...
4
5   // 更新用户的基本信息
6   updateUserInfo(state, userinfo) {
7     state.userinfo = userinfo
8     // 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
9     this.commit('m_user/saveUserInfoToStorage')
10   },
11
12   // 将 userinfo 持久化存储到本地
13   saveUserInfoToStorage(state) {
14     uni.setStorageSync('userinfo', JSON.stringify(state.userinfo))
15   }
16 }
```

```
16   }
```

3. 使用 `mapMutations` 辅助函数，将需要的方法映射到 `my-login` 组件中使用：

```
1  // 1. 按需导入 mapMutations 辅助函数
2  import { mapMutations } from 'vuex'
3
4  export default {
5    data() {
6      return {}
7    },
8    methods: {
9      // 2. 调用 mapMutations 辅助方法，把 m_user 模块中的 updateUserInfo 映射到
      当前组件中使用
10     ...mapMutations('m_user', ['updateUserInfo']),
11     // 获取微信用户的基本信息
12     getUserInfo(e) {
13       // 判断是否获取用户信息成功
14       if (e.detail.errMsg === 'getUserInfo:fail auth deny') return
      uni.$showMsg('您取消了登录授权! ')
15       // 获取用户信息成功， e.detail.userInfo 就是用户的基本信息
16       // console.log(e.detail.userInfo)
17
18       // 3. 将用户的基本信息存储到 vuex 中
19       this.updateUserInfo(e.detail.userInfo)
20     },
21   },
22 }
```

## 10.2.6 登录获取 Token 字符串

需求说明：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。

1. 在 `getUserInfo` 方法中，预调用 `this.getToken()` 方法，同时把获取到的用户信息传递进去：

```
1  // 获取微信用户的基本信息
2  getUserInfo(e) {
3    // 判断是否获取用户信息成功
4    if (e.detail.errMsg === 'getUserInfo:fail auth deny') return
      uni.$showMsg('您取消了登录授权! ')
5
6    // 将用户的基本信息存储到 vuex 中
7    this.updateUserInfo(e.detail.userInfo)
8
9    // 获取登录成功后的 Token 字符串
10    this.getToken(e.detail)
11  }
```

2. 在 `methods` 中定义 `getToken` 方法，调用登录相关的 API，实现登录的功能：

```
1  // 调用登录接口，换取永久的 token
2  async getToken(info) {
3    // 调用微信登录接口
4    const [err, res] = await uni.login().catch(err => err)
```

```

5    // 判断是否 wx.login() 调用失败
6    if (err || res.errMsg !== 'login:ok') return uni.$showError('登录失败! ')
7
8    // 准备参数对象
9    const query = {
10      code: res.code,
11      encryptedData: info.encryptedData,
12      iv: info.iv,
13      rawData: info.rawData,
14      signature: info.signature
15    }
16
17    // 换取 token
18    const { data: loginResult } = await
uni.$http.post('/api/public/v1/users/wxlogin', query)
19    if (loginResult.meta.status !== 200) return uni.$showMsg('登录失败! ')
20    uni.$showMsg('登录成功')
21  }

```

## 10.2.7 将 Token 存储到 vuex

1. 在 `store/user.js` 模块的 `mutations` 节点中, 声明如下的两个方法:

```

1    mutations: {
2      // 省略其它代码...
3
4      // 更新 token 字符串
5      updateToken(state, token) {
6        state.token = token
7        // 通过 this.commit() 方法, 调用 m_user 模块下的 saveTokenToStorage 方法,
      将 token 字符串持久化存储到本地
8        this.commit('m_user/saveTokenToStorage')
9      },
10
11      // 将 token 字符串持久化存储到本地
12      saveTokenToStorage(state) {
13        uni.setStorageSync('token', state.token)
14      }
15    }

```

2. 修改 `store/user.js` 模块的 `state` 节点如下:

```

1    // state 数据
2    state: () => ({
3      // 收货地址
4      address: JSON.parse(uni.getStorageSync('address')) || '{}',
5      // 登录成功之后的 token 字符串
6      token: uni.getStorageSync('token') || '',
7      // 用户的基本信息
8      userinfo: JSON.parse(uni.getStorageSync('userinfo')) || '{}')
9    },

```

3. 在 `my-login` 组件中, 把 vuex 中的 `updateToken` 方法映射到当前组件中使用:

```

1    methods: {
2      // 1. 使用 mapMutations 辅助方法, 把 m_user 模块中的 updateToken 方法映射到当前组件中使用

```

```

3      ...mapMutations('m_user', ['updateUserInfo', 'updateToken'])
4
5      // 省略其它代码...
6
7      // 调用登录接口, 换取永久的 token
8      async getToken(info) {
9          // 调用微信登录接口
10         const [err, res] = await uni.login().catch(err => err)
11         // 判断是否 wx.login() 调用失败
12         if (err || res.errMsg !== 'login:ok') return uni.$showError('登录失
败! ')
13
14         // 准备参数对象
15         const query = {
16             code: res.code,
17             encryptedData: info.encryptedData,
18             iv: info.iv,
19             rawData: info.rawData,
20             signature: info.signature
21         }
22
23         // 换取 token
24         const { data: loginResult } = await
uni.$http.post('/api/public/v1/users/wxlogin', query)
25         if (loginResult.meta.status !== 200) return uni.$showMsg('登录失败! ')
26
27         // 2. 更新 vuex 中的 token
28         this.updateToken(loginResult.message.token)
29     }
30 }

```

## 10.3 用户信息

### 10.3.1 实现用户头像昵称区域的基本布局

1. 在 `my-userinfo` 组件中, 定义如下的 UI 结构:

```

1      <template>
2          <view class="my-userinfo-container">
3
4              <!-- 头像昵称区域 -->
5              <view class="top-box">
6                  <image src="" class="avatar"></image>
7                  <view class="nickname">xxx</view>
8              </view>
9
10             </view>
11         </template>

```

2. 美化当前组件的样式:

```

1      .my-userinfo-container {
2          height: 100%;
3          // 为整个组件的结构添加浅灰色的背景
4          background-color: #f4f4f4;

```



```

5
6   .top-box {
7     height: 400rpx;
8     background-color: #c00000;
9     display: flex;
10    flex-direction: column;
11    align-items: center;
12    justify-content: center;
13
14    .avatar {
15      display: block;
16      width: 90px;
17      height: 90px;
18      border-radius: 45px;
19      border: 2px solid white;
20      box-shadow: 0 1px 5px black;
21    }
22
23    .nickname {
24      color: white;
25      font-weight: bold;
26      font-size: 16px;
27      margin-top: 10px;
28    }
29  }
30 }

```

3. 在 `my.vue` 页面中，为最外层包裹性质的 `view` 容器，添加 `class="my-container"` 的类名，并美化样式如下：

```

1  page,
2  .my-container {
3    height: 100%;
4  }

```

### 10.3.2 渲染用户的头像和昵称

1. 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：

```

1  // 按需导入 mapState 辅助函数
2  import { mapState } from 'vuex'
3
4  export default {
5    computed: {
6      // 将 m_user 模块中的 userinfo 映射到当前页面中使用
7      ...mapState('m_user', ['userinfo']),
8    },
9    data() {
10     return {}
11   },
12 }

```

2. 将用户的头像和昵称渲染到页面中：

```

1      <!-- 头像昵称区域 -->
2      <view class="top-box">
3          <image :src="userinfo.avatarUrl" class="avatar"></image>
4          <view class="nickname">{{userinfo.nickName}}</view>
5      </view>

```

### 10.3.3 渲染第一个面板区域

1. 在 `my-userinfo` 组件中，定义如下的 UI 结构：

```

1      <!-- 面板的列表区域 -->
2      <view class="panel-list">
3          <!-- 第一个面板 -->
4          <view class="panel">
5              <!-- panel 的主体区域 -->
6              <view class="panel-body">
7                  <!-- panel 的 item 项 -->
8                  <view class="panel-item">
9                      <text>8</text>
10                     <text>收藏的店铺</text>
11                 </view>
12                 <view class="panel-item">
13                     <text>14</text>
14                     <text>收藏的商品</text>
15                 </view>
16                 <view class="panel-item">
17                     <text>18</text>
18                     <text>关注的商品</text>
19                 </view>
20                 <view class="panel-item">
21                     <text>84</text>
22                     <text>足迹</text>
23                 </view>
24             </view>
25         </view>
26
27         <!-- 第二个面板 -->
28
29         <!-- 第三个面板 -->
30     </view>

```

2. 美化第一个面板的样式：

```

1      .panel-list {
2          padding: 0 10px;
3          position: relative;
4          top: -10px;
5
6          .panel {
7              background-color: white;
8              border-radius: 3px;
9              margin-bottom: 8px;
10
11              .panel-body {
12                  display: flex;
13                  justify-content: space-around;
14

```

```

15     .panel-item {
16         display: flex;
17         flex-direction: column;
18         align-items: center;
19         justify-content: space-around;
20         font-size: 13px;
21         padding: 10px 0;
22     }
23 }
24 }
25 }

```

### 10.3.4 渲染第二个面板区域

1. 定义第二个面板区域的 UI 结构：

```

1     <!-- 第二个面板 -->
2     <view class="panel">
3         <!-- 面板的标题 -->
4         <view class="panel-title">我的订单</view>
5         <!-- 面板的主体 -->
6         <view class="panel-body">
7             <!-- 面板主体中的 item 项 -->
8             <view class="panel-item">
9                 <image src="/static/my-icons/icon1.png" class="icon"></image>
10                <text>待付款</text>
11            </view>
12            <view class="panel-item">
13                <image src="/static/my-icons/icon2.png" class="icon"></image>
14                <text>待收货</text>
15            </view>
16            <view class="panel-item">
17                <image src="/static/my-icons/icon3.png" class="icon"></image>
18                <text>退款/退货</text>
19            </view>
20            <view class="panel-item">
21                <image src="/static/my-icons/icon4.png" class="icon"></image>
22                <text>全部订单</text>
23            </view>
24        </view>
25    </view>

```

2. 对之前的 SCSS 样式进行改造，从而美化第二个面板的样式：

```

1     .panel-list {
2         padding: 0 10px;
3         position: relative;
4         top: -10px;
5
6         .panel {
7             background-color: white;
8             border-radius: 3px;
9             margin-bottom: 8px;
10
11             .panel-title {
12                 line-height: 45px;
13                 padding-left: 10px;

```

```

14     font-size: 15px;
15     border-bottom: 1px solid #f4f4f4;
16 }
17
18 .panel-body {
19     display: flex;
20     justify-content: space-around;
21
22     .panel-item {
23         display: flex;
24         flex-direction: column;
25         align-items: center;
26         justify-content: space-around;
27         font-size: 13px;
28         padding: 10px 0;
29
30         .icon {
31             width: 35px;
32             height: 35px;
33         }
34     }
35 }
36 }
37 }

```

### 10.3.5 渲染第三个面板区域

1. 定义第三个面板区域的 UI 结构:

```

1  <!-- 第三个面板 -->
2  <view class="panel">
3      <view class="panel-list-item">
4          <text>收货地址</text>
5          <uni-icons type="arrowright" size="15"></uni-icons>
6      </view>
7      <view class="panel-list-item">
8          <text>联系客服</text>
9          <uni-icons type="arrowright" size="15"></uni-icons>
10     </view>
11     <view class="panel-list-item">
12         <text>退出登录</text>
13         <uni-icons type="arrowright" size="15"></uni-icons>
14     </view>
15 </view>

```

2. 美化第三个面板区域的样式:

```

1 .panel-list-item {
2     height: 45px;
3     display: flex;
4     justify-content: space-between;
5     align-items: center;
6     font-size: 15px;
7     padding: 0 10px;
8 }

```

### 10.3.6 实现退出登录的功能

1. 为第三个面板区域中的 `退出登录` 项绑定 `click` 点击事件处理函数：

```
1 <view class="panel-list-item" @click="logout">
2   <text>退出登录</text>
3   <uni-icons type="arrowright" size="15"></uni-icons>
4 </view>
```

2. 在 `my-userinfo` 组件的 `methods` 节点中定义 `logout` 事件处理函数：

```
1 // 退出登录
2 async logout() {
3   // 询问用户是否退出登录
4   const [err, succ] = await uni.showModal({
5     title: '提示',
6     content: '确认退出登录吗?'
7   }).catch(err => err)
8
9   if (succ && succ.confirm) {
10    // 用户确认了退出登录的操作
11    // 需要清空 vuex 中的 userinfo、token 和 address
12    this.updateUserInfo({})
13    this.updateToken('')
14    this.updateAddress({})
15  }
16 }
```

3. 使用 `mapMutations` 辅助方法，将需要用到的 mutations 方法映射到当前组件中：

```
1 // 按需导入辅助函数
2 import { mapState, mapMutations } from 'vuex'
3
4 export default {
5   methods: {
6     ...mapMutations('m_user', ['updateUserInfo', 'updateToken',
7     'updateAddress']),
8   },
9 }
```

## 10.4 三秒后自动跳转

### 10.4.1 三秒后自动跳转到登录页面

需求描述：在购物车页面，当用户点击“结算”按钮时，如果用户没有登录，则 3 秒后自动跳转到登录页面

1. 在 `my-settle` 组件的 `methods` 节点中，声明一个叫做 `showTips` 的方法，专门用来展示倒计时的提示消息：

```
1 // 展示倒计时的提示消息
2 showTips(n) {
3   // 调用 uni.showToast() 方法，展示提示消息
4   uni.showToast({
5     // 不展示任何图标
6     icon: 'none',
```

```

7      // 提示的消息
8      title: '请登录后再结算!' + n + ' 秒后自动跳转到登录页',
9      // 为页面添加透明遮罩, 防止点击穿透
10     mask: true,
11     // 1.5 秒后自动消失
12     duration: 1500
13   })
14 }

```

2. 在 `data` 节点中声明倒计时的秒数:

```

1  data() {
2    return {
3      // 倒计时的秒数
4      seconds: 3
5    }
6  }

```

3. 改造 `结算` 按钮的 `click` 事件处理函数, 如果用户没有登录, 则预调用一个叫做 `delayNavigate` 的方法, 进行倒计时的导航跳转:

```

1  // 点击了结算按钮
2  settlement() {
3    // 1. 先判断是否勾选了要结算的商品
4    if (!this.checkedCount) return uni.$showMsg('请选择要结算的商品!')
5
6    // 2. 再判断用户是否选择了收货地址
7    if (!this.addstr) return uni.$showMsg('请选择收货地址!')
8
9    // 3. 最后判断用户是否登录了, 如果没有登录, 则调用 delayNavigate() 进行倒计时的导航跳转
10   // if (!this.token) return uni.$showMsg('请先登录!')
11   if (!this.token) return this.delayNavigate()
12 },

```

4. 定义 `delayNavigate` 方法, 初步实现倒计时的提示功能:

```

1  // 延迟导航到 my 页面
2  delayNavigate() {
3    // 1. 展示提示消息, 此时 seconds 的值等于 3
4    this.showTips(this.seconds)
5
6    // 2. 创建定时器, 每隔 1 秒执行一次
7    setInterval(() => {
8      // 2.1 先让秒数自减 1
9      this.seconds--
10     // 2.2 再根据最新的秒数, 进行消息提示
11     this.showTips(this.seconds)
12   }, 1000)
13 },

```

上述代码的问题: **定时器不会自动停止**, 此时秒数会出现等于 0 或小于 0 的情况!

5. 在 `data` 节点中声明定时器的 Id 如下:

```

1  data() {
2    return {
3      // 倒计时的秒数
4      seconds: 3,
5      // 定时器的 Id
6      timer: null
7    }
8  }

```

6. 改造 `delayNavigate` 方法如下:

```

1  // 延迟导航到 my 页面
2  delayNavigate() {
3    this.showTips(this.seconds)
4
5    // 1. 将定时器的 Id 存储到 timer 中
6    this.timer = setInterval(() => {
7      this.seconds--
8
9      // 2. 判断秒数是否 <= 0
10     if (this.seconds <= 0) {
11       // 2.1 清除定时器
12       clearInterval(this.timer)
13
14       // 2.2 跳转到 my 页面
15       uni.switchTab({
16         url: '/pages/my/my'
17       })
18
19       // 2.3 终止后续代码的运行 (当秒数为 0 时, 不再展示 toast 提示消息)
20       return
21     }
22
23     this.showTips(this.seconds)
24   }, 1000)
25 },

```

上述代码的问题: `seconds` 秒数不会被重置, 导致第 2 次, 3 次, n 次的倒计时跳转功能无法正常工作

7. 进一步改造 `delayNavigate` 方法, 在执行此方法时, 立即将 `seconds` 秒数重置为 3 即可:

```

1  // 延迟导航到 my 页面
2  delayNavigate() {
3    // 把 data 中的秒数重置成 3 秒
4    this.seconds = 3
5    this.showTips(this.seconds)
6
7    this.timer = setInterval(() => {
8      this.seconds--
9
10     if (this.seconds <= 0) {
11       clearInterval(this.timer)
12       uni.switchTab({
13         url: '/pages/my/my'
14       })

```

```

15         return
16     }
17
18     this.showTips(this.seconds)
19 }, 1000)
20 }

```

## 10.4.2 登录成功之后再返回之前的页面

核心实现思路：在自动跳转到登录页面成功之后，把返回页面的信息存储到 vuex 中，从而方便登录成功之后，根据返回页面的信息重新跳转回去。

返回页面的信息对象，主要包含 { openType, from } 两个属性，其中 openType 表示以哪种方式导航回之前的页面；from 表示之前页面的 url 地址；

1. 在 store/user.js 模块的 state 节点中，声明一个叫做 redirectInfo 的对象如下：

```

1  // state 数据
2  state: () => ({
3      // 收货地址
4      address: JSON.parse(uni.getStorageSync('address')) || '{}',
5      // 登录成功之后的 token 字符串
6      token: uni.getStorageSync('token') || '',
7      // 用户的基本信息
8      userinfo: JSON.parse(uni.getStorageSync('userinfo')) || '{}',
9      // 重定向的 object 对象 { openType, from }
10     redirectInfo: null
11  }),

```

2. 在 store/user.js 模块的 mutations 节点中，声明一个叫做 updateRedirectInfo 的方法：

```

1  mutations: {
2      // 更新重定向的信息对象
3      updateRedirectInfo(state, info) {
4          state.redirectInfo = info
5      }
6  }

```

3. 在 my-settle 组件中，通过 mapMutations 辅助方法，把 m\_user 模块中的 updateRedirectInfo 方法映射到当前页面中使用：

```

1  methods: {
2      // 把 m_user 模块中的 updateRedirectInfo 方法映射到当前页面中使用
3      ...mapMutations('m_user', ['updateRedirectInfo']),
4  }

```

4. 改造 my-settle 组件 methods 节点中的 delayNavigate 方法，当成功跳转到 my 页面之后，将重定向的信息对象存储到 vuex 中：

```

1  // 延迟导航到 my 页面
2  delayNavigate() {
3      // 把 data 中的秒数重置成 3 秒
4      this.seconds = 3
5      this.showTips(this.seconds)

```



```

6
7     this.timer = setInterval(() => {
8         this.seconds--
9
10        if (this.seconds <= 0) {
11            // 清除定时器
12            clearInterval(this.timer)
13            // 跳转到 my 页面
14            uni.switchTab({
15                url: '/pages/my/my',
16                // 页面跳转成功之后的回调函数
17                success: () => {
18                    // 调用 vuex 的 updateRedirectInfo 方法，把跳转信息存储到 Store 中
19                    this.updateRedirectInfo({
20                        // 跳转的方式
21                        openType: 'switchTab',
22                        // 从哪个页面跳转过去的
23                        from: '/pages/cart/cart'
24                    })
25                }
26            })
27
28            return
29        }
30
31        this.showTips(this.seconds)
32    }, 1000)
33 }

```

5. 在 `my-login` 组件中，通过 `mapState` 和 `mapMutations` 辅助方法，将 vuex 中需要的数据和方法，映射到当前页面中使用：

```

1    // 按需导入辅助函数
2    import { mapMutations, mapState } from 'vuex'
3
4    export default {
5        computed: {
6            // 调用 mapState 辅助方法，把 m_user 模块中的数据映射到当前组件中使用
7            ...mapState('m_user', ['redirectInfo']),
8        },
9        methods: {
10            // 调用 mapMutations 辅助方法，把 m_user 模块中的方法映射到当前组件中使用
11            ...mapMutations('m_user', ['updateUserInfo', 'updateToken',
12                'updateRedirectInfo']),
13        },
14    }

```

6. 改造 `my-login` 组件中的 `getToken` 方法，当登录成功之后，预调用 `this.navigateBack()` 方法返回登录之前的页面：

```

1    // 调用登录接口，换取永久的 token
2    async getToken(info) {
3        // 省略其它代码...
4
5        // 判断 vuex 中的 redirectInfo 是否为 null
6        // 如果不为 null，则登录成功之后，需要重新导航到对应的页面
7        this.navigateBack()
8    }

```

7. 在 `my-login` 组件中，声明 `navigateBack` 方法如下：

```

1    // 返回登录之前的页面
2    navigateBack() {
3        // redirectInfo 不为 null，并且导航方式为 switchTab
4        if (this.redirectInfo && this.redirectInfo.openType === 'switchTab') {
5            // 调用小程序提供的 uni.switchTab() API 进行页面的导航
6            uni.switchTab({
7                // 要导航到的页面地址
8                url: this.redirectInfo.from,
9                // 导航成功之后，把 vuex 中的 redirectInfo 对象重置为 null
10               complete: () => {
11                   this.updateRedirectInfo(null)
12               }
13            })
14        }
15    }

```

## 10.5 微信支付

### 10.5.1 在请求头中添加 Token 身份认证的字段

- 原因说明：只有在登录之后才允许调用支付相关的接口，所以必须为有权限的接口添加身份认证的请求头字段
- 打开项目根目录下的 `main.js`，改造 `$http.beforeRequest` 请求拦截器中的代码如下：

```

1    // 请求开始之前做一些事情
2    $http.beforeRequest = function(options) {
3        uni.showLoading({
4            title: '数据加载中...',
5        })
6
7        // 判断请求的是否为有权限的 API 接口
8        if (options.url.indexOf('/my/') !== -1) {
9            // 为请求头添加身份认证字段
10           options.header = {
11               // 字段的值可以直接从 vuex 中进行获取
12               Authorization: store.state.m_user.token,
13           }
14       }
15   }

```

### 10.5.2 微信支付的流程

- 创建订单

- 请求创建订单的 API 接口：把（订单金额、收货地址、订单中包含的商品信息）发送到服务器
  - 服务器响应的结果：订单编号
2. 订单预支付
- 请求订单预支付的 API 接口：把（订单编号）发送到服务器
  - 服务器响应的结果：订单预支付的参数对象，里面包含了订单支付相关的必要参数
3. 发起微信支付
- 调用 `uni.requestPayment()` 这个 API，发起微信支付；把步骤 2 得到的“订单预支付对象”作为参数传递给 `uni.requestPayment()` 方法
  - 监听 `uni.requestPayment()` 这个 API 的 `success`，`fail`，`complete` 回调函数

### 10.5.3 创建订单

1. 改造 `my-settle` 组件中的 `settlement` 方法，当前三个判断条件通过之后，调用实现微信支付的方法：

```

1 // 点击了结算按钮
2 settlement() {
3   // 1. 先判断是否勾选了要结算的商品
4   if (!this.checkedCount) return uni.$showMsg('请选择要结算的商品!')
5
6   // 2. 再判断用户是否选择了收货地址
7   if (!this.addstr) return uni.$showMsg('请选择收货地址!')
8
9   // 3. 最后判断用户是否登录了
10  // if (!this.token) return uni.$showMsg('请先登录!')
11  if (!this.token) return this.delayNavigate()
12
13  // 4. 实现微信支付功能
14  this.payOrder()
15 },

```

2. 在 `my-settle` 组件中定义 `payOrder` 方法如下，先实现创建订单的功能：

```

1 // 微信支付
2 async payOrder() {
3   // 1. 创建订单
4   // 1.1 组织订单的信息对象
5   const orderInfo = {
6     // 开发期间，注释掉真实的订单价格，
7     // order_price: this.checkedGoodsAmount,
8     // 写死订单总价为 1 分钱
9     order_price: 0.01,
10    consignee_addr: this.addstr,
11    goods: this.cart.filter(x => x.goods_state).map(x => ({ goods_id:
x.goods_id, goods_number: x.goods_count, goods_price: x.goods_price }))
12  }
13  // 1.2 发起请求创建订单
14  const { data: res } = await
uni.$http.post('/api/public/v1/my/orders/create', orderInfo)
15  if (res.meta.status !== 200) return uni.$showMsg('创建订单失败!')
16  // 1.3 得到服务器响应的“订单编号”
17  const orderNumber = res.message.order_number
18
19  // 2. 订单预支付

```

```
20
21      // 3. 发起微信支付
22    }
```

## 10.5.4 订单预支付

1. 改造 `my-settle` 组件的 `payOrder` 方法，实现订单预支付功能：

```
1    // 微信支付
2    async payOrder() {
3      // 1. 创建订单
4      // 1.1 组织订单的信息对象
5      const orderInfo = {
6        // 开发期间，注释掉真实的订单价格，
7        // order_price: this.checkedGoodsAmount,
8        // 写死订单总价为 1 分钱
9        order_price: 0.01,
10       consignee_addr: this.addstr,
11       goods: this.cart.filter(x => x.goods_state).map(x => ({ goods_id:
x.goods_id, goods_number: x.goods_count, goods_price: x.goods_price }))
12     }
13     // 1.2 发起请求创建订单
14     const { data: res } = await
uni.$http.post('/api/public/v1/my/orders/create', orderInfo)
15     if (res.meta.status !== 200) return uni.$showMsg('创建订单失败! ')
16     // 1.3 得到服务器响应的“订单编号”
17     const orderNumber = res.message.order_number
18
19     // 2. 订单预支付
20     // 2.1 发起请求获取订单的支付信息
21     const { data: res2 } = await
uni.$http.post('/api/public/v1/my/orders/req_unifiedorder', {
order_number: orderNumber })
22     // 2.2 预付订单生成失败
23     if (res2.meta.status !== 200) return uni.$showError('预付订单生成失败! ')
24     // 2.3 得到订单支付相关的必要参数
25     const payInfo = res2.message.pay
26
27     // 3. 发起微信支付
28   }
```

## 10.5.5 发起微信支付

1. 改造 `my-settle` 组件的 `payOrder` 方法，实现微信支付的功能：

```
1    // 微信支付
2    async payOrder() {
3      // 1. 创建订单
4      // 1.1 组织订单的信息对象
5      const orderInfo = {
6        // 开发期间，注释掉真实的订单价格，
7        // order_price: this.checkedGoodsAmount,
8        // 写死订单总价为 1 分钱
9        order_price: 0.01,
10       consignee_addr: this.addstr,
11       goods: this.cart.filter(x => x.goods_state).map(x => ({ goods_id:
x.goods_id, goods_number: x.goods_count, goods_price: x.goods_price }))
```

```

12     }
13     // 1.2 发起请求创建订单
14     const { data: res } = await
uni.$http.post('/api/public/v1/my/orders/create', orderInfo)
15     if (res.meta.status !== 200) return uni.$showMsg('创建订单失败!')
16     // 1.3 得到服务器响应的“订单编号”
17     const orderNumber = res.message.order_number
18
19     // 2. 订单预支付
20     // 2.1 发起请求获取订单的支付信息
21     const { data: res2 } = await
uni.$http.post('/api/public/v1/my/orders/req_unifiedorder', {
order_number: orderNumber })
22     // 2.2 预付订单生成失败
23     if (res2.meta.status !== 200) return uni.$showError('预付订单生成失败!')
24     // 2.3 得到订单支付相关的必要参数
25     const payInfo = res2.message.pay
26
27     // 3. 发起微信支付
28     // 3.1 调用 uni.requestPayment() 发起微信支付
29     const [err, succ] = await uni.requestPayment(payInfo)
30     // 3.2 未完成支付
31     if (err) return uni.$showMsg('订单未支付!')
32     // 3.3 完成了支付, 进一步查询支付的结果
33     const { data: res3 } = await
uni.$http.post('/api/public/v1/my/orders/chkOrder', { order_number:
orderNumber })
34     // 3.4 检测到订单未支付
35     if (res3.meta.status !== 200) return uni.$showMsg('订单未支付!')
36     // 3.5 检测到订单支付完成
37     uni.showToast({
38         title: '支付完成!',
39         icon: 'success'
40     })
41 }

```

## 10.6 分支的合并与提交

1. 将 `settle` 分支进行本地提交:

```

1  git add .
2  git commit -m "完成了登录和支付功能的开发"

```

2. 将本地的 `settle` 分支推送到码云:

```

1  git push -u origin settle

```

3. 将本地 `settle` 分支中的代码合并到 `master` 分支:

```

1  git checkout master
2  git merge settle
3  git push

```

4. 删除本地的 `settle` 分支:

# 11. 发布小程序

## 11.1 为什么要发布

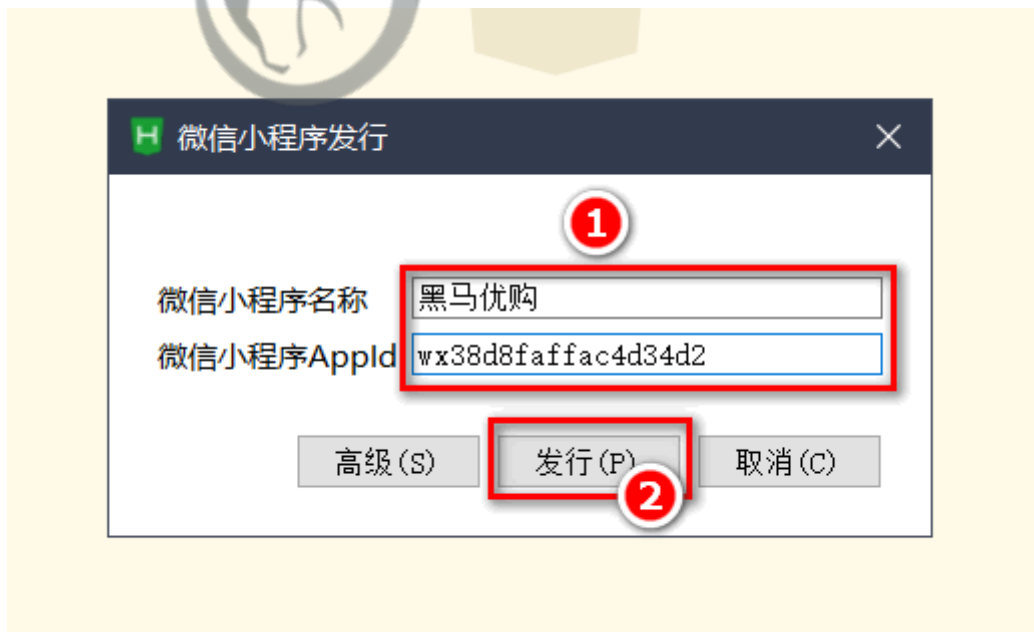
1. 小程序只有发布之后，才能被用户搜索并使用
2. 开发期间的小程序为了便于调试，含有 sourcemap 相关的文件，并且代码没有被压缩，因此体积较大，不适合直接当作线上版本进行发布
3. 通过执行“小程序发布”，能够优化小程序的体积，提高小程序的运行性能

## 11.2 发布小程序的流程

1. 点击 **HBuilderX** 菜单栏上的 **发行** -> **小程序-微信 (仅适用于uni-app)** :



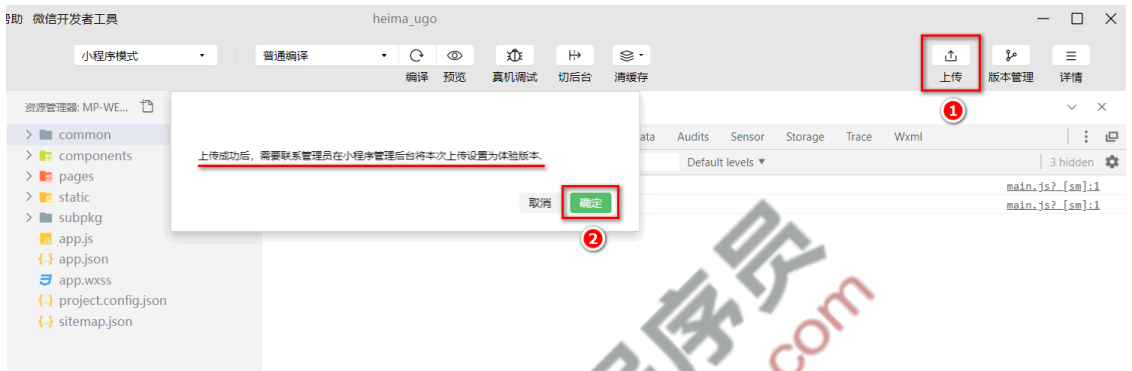
2. 在弹出框中填写要发布的小程序的名称和AppId之后，点击发行按钮：



3. 在 **HBuilderX** 的控制台中查看小程序发布编译的进度：

```
小程序 - 微信 控制台
16:05:33.859
[广告] 16:05:33.870 开源不易，需要鼓励。去给 uni-app 项目 点个 star 吧 [不再提示]
[广告] 16:05:33.874 DCloud 2020新春招聘开启，欢迎前端、Android、C++/QT来投简历！详情点击
[Builder] 16:05:33.875 项目 'uni_proj3' 开始发布微信小程序...
[Builder] 16:05:33.877 项目 'uni_proj3' 开始编译...
[Builder] 16:05:40.222 该应用之前可能是非自定义组件模式，目前以自定义组件模式运行。非自定义组件已于2019年11月1日起停止支持。详见: https://ask.dcloud
[Builder] 16:05:40.269 小程序各家浏览器内核及自定义组件实现机制存在差异，可能存在样式布局兼容问题，参考: https://uniapp.dcloud.io/matter?id=mp
[Builder] 16:05:40.308 正在编译中...
[Builder] 16:06:41.718 DONE Build complete.
[Builder] 16:06:41.718 项目 'uni_proj3' 编译成功。
[Builder] 16:06:42.188 项目 'uni_proj3' 导出微信小程序成功，路径为: C:/Users/liulongbin/Desktop/小程序/uni_proj3/unpackage/dist/build/mp-weixin
[Builder] 16:06:42.206 正在启动微信开发者工具...
[Builder] 16:06:43.447 [微信小程序开发者工具] - initialize
[Builder] 16:06:43.447 [微信小程序开发者工具]
[Builder] 16:06:43.458 [微信小程序开发者工具] ✓ IDE server has started, listening on http://127.0.0.1:31152
[Builder] 16:06:43.459 [微信小程序开发者工具] - open IDE
[Builder] 16:06:43.473 [微信小程序开发者工具]
[Builder] 16:06:43.473 [微信小程序开发者工具]
[Builder] 16:06:44.769 [微信小程序开发者工具] ✓ open IDE
[Builder] 16:06:44.769 [微信小程序开发者工具]
[Builder] 16:06:44.786 请在微信小程序开发者工具中点击上传
```

4. 发布编译完成之后，会自动打开一个新的微信开发者工具界面，此时，点击工具栏上的上传按钮：



5. 填写版本号和项目备注之后，点击上传按钮：



6. 上传完成之后，会出现如下的提示消息，直接点击确定按钮即可：



7. 通过微信开发者工具上传的代码，默认处于版本管理的开发版本列表中，如图所示：



8. 将 开发版本提交审核 -> 再将 审核通过的版本发布上线，即可实现小程序的发布和上线：



## 11.3 发布为 Android App 的流程

1. 点击 HBuilderX 状态栏左侧的未登录按钮，弹出登录的对话框：



2. 在弹出的登录对话框中，填写账号和密码之后，点击登录即可：

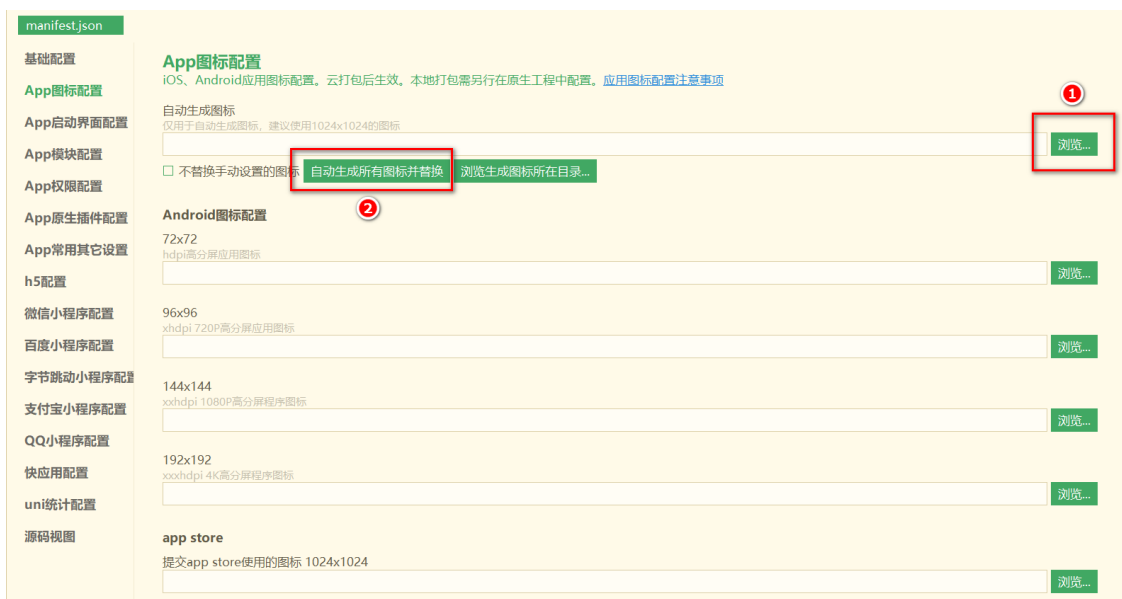




3. 打开项目根目录中的 `manifest.json` 配置文件, 在基础配置面板中, 获取uni-app 应用标识, 并填写应用名称:



4. 切换到 App 图标配置面板, 点击浏览按钮, 选择合适的图片之后, 再点击 自动生成所有图标并替换 即可:



5. 点击菜单栏上的 发行 -> 原生 App-云打包:



6. 勾选打包配置如下:

uni\_proj3 - App云端打包

应用名称: 黑马优购

应用版本号: 100

修改manifest配置

1

☒ Android (apk包)

☐ iOS (ipa包)

Android设置

iOS设置

Android包名

uni.UNI7F0B001

Android证书使用指南

☐ 使用自有证书

[如何生成证书](#)

☒ 使用公共测试证书

[详情](#)

☐ 使用DCloud老版证书

证书别名

证书私钥密码

证书文件

浏览...

渠道包

[渠道包制作指南](#)

☒ 打正式包

☐ 打自定义调试基座 (iOS的Safari调试需要用苹果开发证书打包)

[什么是自定义调试基座?](#)

原生混淆

☐ 对配置的js/nvue文件进行原生混淆

[\[配置指南\]](#)

广告联盟

加入uni-AD广告联盟, 帮助你的App变现。[\[了解详情\]](#)

☐ 基础开屏广告

☐ 悬浮红包广告

☐ push广告

集成三方广告SDK。[\[AD组件开发指南\]](#)

☐ 腾讯广点通

☐ 今日头条穿山甲

☐ 360广告联盟

换量联盟

☐ 加入换量联盟, 免费获取更多用户, 开通越早, 权重越高

[\[点此设置\]](#) [\[了解详情\]](#)

注: DCloud郑重承诺不保留开发者证书及代码

4

打包(F)

取消(C)

## 7. 在控制台中查看打包的进度信息:

```
[HBuilder] 15:42:42.205 项目 uni_proj3 [__UNI__7F0B001]的打包状态:
[HBuilder] 15:42:42.205 时间: 15:42:35 类型: Android 正在打包
[HBuilder] 15:42:42.205

[HBuilder] 15:42:42.205 打包成功后会返回下载链接。
[HBuilder] 15:42:42.205 打包过程查询请点击菜单发行-查看云打包状态。
[HBuilder] 15:42:42.205 周五傍晚等高峰期打包排队较长, 请耐心等待。
[HBuilder] 15:42:42.205 如果是为了三方SDK调试, 请使用自定义调试基座 (菜单运行-手机或模拟器-制作自定义调试基座), 不要反复打包。

[HBuilder] 15:43:33.480 项目 uni_proj3 [__UNI__7F0B001]打包成功:
类型: Android 公共测试证书 下载地址: https://service.dcloud.net.cn/build/download/b3d31620-ec26-99c6ec2c2b88 (注意该地址为临时下载地址, 只能下载5次)
```

## 8. 点击链接下载 apk 的安装包, 并安装到 Android 手机中查看打包的效果。

注意: 由于开发期间没有进行多端适配, 所以有些功能在 App 中无法正常运行, 例如: 选择收货地址、微信登录、微信支付