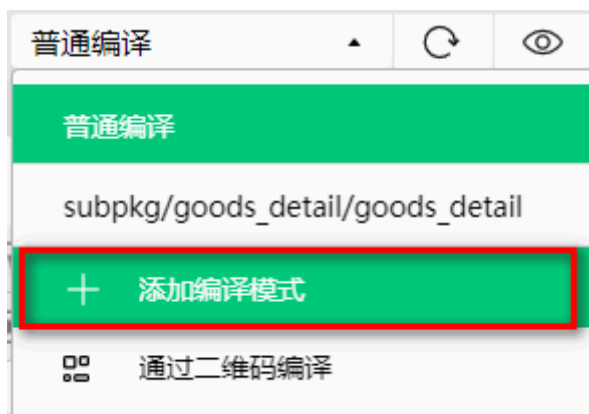


9. 购物车页面

9.0 创建购物车页面的编译模式

1. 打开微信开发者工具，点击工具栏上的“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，新增购物车页面的编译模式：

自定义编译条件

解析二维码

上传文件

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。

模式名称

pages/cart/cart

1

启动页面

pages/cart/cart

▼

启动参数

如：name=vendor&color=black

进入场景

默认

▼

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)

取消

确定

2

9.1 商品列表区域

9.1.1 渲染购物车商品列表的标题区域

1. 定义如下的 UI 结构：

```

1      <!-- 购物车商品列表的标题区域 -->
2      <view class="cart-title">
3          <!-- 左侧的图标 -->
4          <uni-icons type="shop" size="18"></uni-icons>
5          <!-- 描述文本 -->
6          <text class="cart-title-text">购物车</text>
7      </view>

```

2. 美化样式:

```

1      .cart-title {
2          height: 40px;
3          display: flex;
4          align-items: center;
5          font-size: 14px;
6          padding-left: 5px;
7          border-bottom: 1px solid #efefef;
8          .cart-title-text {
9              margin-left: 10px;
10         }
11     }

```

9.1.2 渲染商品列表区域的基本结构

1. 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      import badgeMix from '@mixins/tabbar-badge.js'
2      // 按需导入 mapState 这个辅助函数
3      import { mapState } from 'vuex'
4
5      export default {
6          mixins: [badgeMix],
7          computed: {
8              // 将 m_cart 模块中的 cart 数组映射到当前页面中使用
9              ...mapState('m_cart', ['cart']),
10         },
11         data() {
12             return {}
13         },
14     }

```

2. 在 UI 结构中, 通过 `v-for` 指令循环渲染自定义的 `my-goods` 组件:

```

1      <!-- 商品列表区域 -->
2      <block v-for="(goods, i) in cart" :key="i">
3          <my-goods :goods="goods"></my-goods>
4      </block>

```

9.1.3 为 my-goods 组件封装 radio 勾选状态

1. 打开 `my-goods.vue` 组件的源代码, 为商品的左侧图片区域添加 `radio` 组件:

```

1      <!-- 商品左侧图片区域 -->
2      <view class="goods-item-left">
3          <radio checked color="#C00000"></radio>
4          <image :src="goods.goods_small_logo || defaultPic" class="goods-pic">
5      </image>
6      </view>

```

2. 给类名为 `goods-item-left` 的 `view` 组件添加样式，实现 `radio` 组件和 `image` 组件的左右布局：

```

1      .goods-item-left {
2          margin-right: 5px;
3          display: flex;
4          justify-content: space-between;
5          align-items: center;
6
7      .goods-pic {
8          width: 100px;
9          height: 100px;
10         display: block;
11     }
12 }

```

3. 封装名称为 `showRadio` 的 `props` 属性，来控制当前组件中是否显示 `radio` 组件：

```

1      export default {
2          // 定义 props 属性，用来接收外界传递到当前组件的数据
3          props: {
4              // 商品的信息对象
5              goods: {
6                  type: Object,
7                  default: {},
8              },
9              // 是否展示图片左侧的 radio
10             showRadio: {
11                 type: Boolean,
12                 // 如果外界没有指定 show-radio 属性的值，则默认不展示 radio 组件
13                 default: false,
14             },
15         },
16     }

```

4. 使用 `v-if` 指令控制 `radio` 组件的按需展示：

```

1      <!-- 商品左侧图片区域 -->
2      <view class="goods-item-left">
3          <!-- 使用 v-if 指令控制 radio 组件的显示与隐藏 -->
4          <radio checked color="#C00000" v-if="showRadio"></radio>
5          <image :src="goods.goods_small_logo || defaultPic" class="goods-pic">
6      </image>
7      </view>

```

5. 在 `cart.vue` 页面中的商品列表区域，指定 `:show-radio="true"` 属性，从而显示 `radio` 组件：

```

1      <!-- 商品列表区域 -->
2      <block v-for="(goods, i) in cart" :key="i">
3          <my-goods :goods="goods" :show-radio="true"></my-goods>
4      </block>

```

6. 修改 `my-goods.vue` 组件，动态为 `radio` 绑定选中状态：

```

1      <!-- 商品左侧图片区域 -->
2      <view class="goods-item-left">
3          <!-- 存储在购物车中的商品，包含 goods_state 属性，表示商品的勾选状态 -->
4          <radio :checked="goods.goods_state" color="#C00000" v-if="showRadio">
5              <image :src="goods.goods_small_logo || defaultPic" class="goods-pic">
6              </image>
7          </radio>
8      </view>

```

9.1.4 为 my-goods 组件封装 radio-change 事件

1. 当用户点击 `radio` 组件，希望修改当前商品的勾选状态，此时用户可以为 `my-goods` 组件绑定 `@radio-change` 事件，从而获取当前商品的 `goods_id` 和 `goods_state`：

```

1      <!-- 商品列表区域 -->
2      <block v-for="(goods, i) in cart" :key="i">
3          <!-- 在 radioChangeHandler 事件处理函数中，通过事件对象 e，得到商品的 goods_id
4              和 goods_state -->
5          <my-goods :goods="goods" :show-radio="true" @radio-
6              change="radioChangeHandler"></my-goods>
7      </block>

```

定义 `radioChangeHandler` 事件处理函数如下：

```

1      methods: {
2          // 商品的勾选状态发生了变化
3          radioChangeHandler(e) {
4              console.log(e) // 输出得到的数据 -> {goods_id: 395, goods_state: false}
5          }
6      }

```

2. 在 `my-goods.vue` 组件中，为 `radio` 组件绑定 `@click` 事件处理函数如下：

```

1      <!-- 商品左侧图片区域 -->
2      <view class="goods-item-left">
3          <radio :checked="goods.goods_state" color="#C00000" v-if="showRadio"
4              @click="radioClickHandler"></radio>
5          <image :src="goods.goods_small_logo || defaultPic" class="goods-pic">
6          </image>
7      </view>

```

3. 在 `my-goods.vue` 组件的 `methods` 节点中，定义 `radioClickHandler` 事件处理函数：

```

1  methods: {
2    // radio 组件的点击事件处理函数
3    radioClickHandler() {
4      // 通过 this.$emit() 触发外界通过 @ 绑定的 radio-change 事件,
5      // 同时把商品的 Id 和 勾选状态 作为参数传递给 radio-change 事件处理函数
6      this.$emit('radio-change', {
7        // 商品的 Id
8        goods_id: this.goods.goods_id,
9        // 商品最新的勾选状态
10       goods_state: !this.goods.goods_state
11     })
12   }
13 }

```

9.1.5 修改购物车中商品的勾选状态

1. 在 `store/cart.js` 模块中, 声明如下的 `mutations` 方法, 用来修改对应商品的勾选状态:

```

1  // 更新购物车中商品的勾选状态
2  updateGoodsState(state, goods) {
3    // 根据 goods_id 查询购物车中对应商品的信息对象
4    const findResult = state.cart.find(x => x.goods_id === goods.goods_id)
5
6    // 有对应的商品信息对象
7    if (findResult) {
8      // 更新对应商品的勾选状态
9      findResult.goods_state = goods.goods_state
10     // 持久化存储到本地
11     this.commit('m_cart/saveToStorage')
12   }
13 }

```

2. 在 `cart.vue` 页面中, 导入 `mapMutations` 这个辅助函数, 从而将需要的 mutations 方法映射到当前页面中使用:

```

1  import badgeMix from '@mixins/tabbar-badge.js'
2  import { mapState, mapMutations } from 'vuex'
3
4  export default {
5    mixins: [badgeMix],
6    computed: {
7      ...mapState('m_cart', ['cart']),
8    },
9    data() {
10     return {}
11   },
12   methods: {
13     ...mapMutations('m_cart', ['updateGoodsState']),
14     // 商品的勾选状态发生了变化
15     radioChangeHandler(e) {
16       this.updateGoodsState(e)
17     },
18   },
19 }

```

9.1.6 为 my-goods 组件封装 NumberBox

注意：NumberBox 组件是 uni-ui 提供的

1. 修改 `my-goods.vue` 组件的源代码，在类名为 `goods-info-box` 的 view 组件内部渲染 NumberBox 组件的基本结构：

```
1 <view class="goods-info-box">
2   <!-- 商品价格 -->
3   <view class="goods-price">¥{{goods.goods_price | tofixed}}</view>
4   <!-- 商品数量 -->
5   <uni-number-box :min="1"></uni-number-box>
6 </view>
```

2. 美化页面的结构：

```
1 .goods-item-right {
2   display: flex;
3   flex: 1;
4   flex-direction: column;
5   justify-content: space-between;
6
7   .goods-name {
8     font-size: 13px;
9   }
10
11   .goods-info-box {
12     display: flex;
13     align-items: center;
14     justify-content: space-between;
15   }
16
17   .goods-price {
18     font-size: 16px;
19     color: #c00000;
20   }
21 }
```

3. 在 `my-goods.vue` 组件中，动态为 NumberBox 组件绑定商品的数值：

```
1 <view class="goods-info-box">
2   <!-- 商品价格 -->
3   <view class="goods-price">¥{{goods.goods_price | tofixed}}</view>
4   <!-- 商品数量 -->
5   <uni-number-box :min="1" :value="goods.goods_count"></uni-number-box>
6 </view>
```

4. 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：

```
1 export default {
2   // 定义 props 属性，用来接收外界传递到当前组件的数据
3   props: {
4     // 商品的信息对象
5     goods: {
6       type: Object,
7       default: {},
8     },
9     // 是否展示图片左侧的 radio
```

```

10     showRadio: {
11         type: Boolean,
12         // 如果外界没有指定 show-radio 属性的值，则默认不展示 radio 组件
13         default: false,
14     },
15     // 是否展示价格右侧的 NumberBox 组件
16     showNum: {
17         type: Boolean,
18         default: false,
19     },
20 },
21 }

```

- 在 `my-goods.vue` 组件中，使用 `v-if` 指令控制 `NumberBox` 组件的按需展示：

```

1  <view class="goods-info-box">
2      <!-- 商品价格 -->
3      <view class="goods-price">¥{{goods.goods_price | tofixed}}</view>
4      <!-- 商品数量 -->
5      <uni-number-box :min="1" :value="goods.goods_count"
        @change="numChangeHandler" v-if="showNum"></uni-number-box>
6  </view>

```

- 在 `cart.vue` 页面中的商品列表区域，指定 `:show-num="true"` 属性，从而显示 `NumberBox` 组件：

```

1  <!-- 商品列表区域 -->
2  <block v-for="(goods, i) in cart" :key="i">
3      <my-goods :goods="goods" :show-radio="true" :show-num="true" @radio-
        change="radioChangeHandler"></my-goods>
4  </block>

```

9.1.7 为 my-goods 组件封装 num-change 事件

- 当用户修改了 `NumberBox` 的值以后，希望将最新的商品数量更新到购物车中，此时用户可以为 `my-goods` 组件绑定 `@num-change` 事件，从而获取当前商品的 `goods_id` 和 `goods_count`：

```

1  <!-- 商品列表区域 -->
2  <block v-for="(goods, i) in cart" :key="i">
3      <my-goods :goods="goods" :show-radio="true" :show-num="true" @radio-
        change="radioChangeHandler" @num-change="numberChangeHandler"></my-goods>
4  </block>

```

定义 `numberChangeHandler` 事件处理函数如下：

```

1  // 商品的数量发生了变化
2  numberChangeHandler(e) {
3      console.log(e)
4  }

```

- 在 `my-goods.vue` 组件中，为 `uni-number-box` 组件绑定 `@change` 事件处理函数如下：

```

1   <view class="goods-info-box">
2     <!-- 商品价格 -->
3     <view class="goods-price">¥{{goods.goods_price | tofixed}}</view>
4     <!-- 商品数量 -->
5     <uni-number-box :min="1" :value="goods.goods_count"
      @change="numChangeHandler"></uni-number-box>
6   </view>

```

3. 在 `my-goods.vue` 组件的 `methods` 节点中, 定义 `numChangeHandler` 事件处理函数:

```

1   methods: {
2     // NumberBox 组件的 change 事件处理函数
3     numChangeHandler(val) {
4       // 通过 this.$emit() 触发外界通过 @ 绑定的 num-change 事件
5       this.$emit('num-change', {
6         // 商品的 Id
7         goods_id: this.goods.goods_id,
8         // 商品的最新数量
9         goods_count: +val
10      })
11    }
12  }

```

9.1.8 解决 NumberBox 数据不合法的问题

问题说明: 当用户在 NumberBox 中输入字母等非法字符之后, 会导致 NumberBox 数据紊乱的问题

1. 打开项目根目录中 `components/uni-number-box/uni-number-box.vue` 组件, 修改 `methods` 节点中的 `_onBlur` 函数如下:

```

1   _onBlur(event) {
2     // 官方的代码没有进行数值转换, 用户输入的 value 值可能是非法字符:
3     // let value = event.detail.value;
4
5     // 将用户输入的内容转化为整数
6     let value = parseInt(event.detail.value);
7
8     if (!value) {
9       // 如果转化之后的结果为 NaN, 则给定默认值为 1
10      this.inputValue = 1;
11      return;
12    }
13
14    // 省略其它代码...
15  }

```

2. 修改完毕之后, 用户输入**小数会被转化为整数**, 用户输入**非法字符会被替换为默认值 1**

9.1.9 完善 NumberBox 的 inputValue 侦听器

问题说明: 在用户每次输入内容之后, 都会触发 inputValue 侦听器, 从而调用 `this.$emit("change", newVal)` 方法。这种做法可能会把不合法的内容传递出去!

1. 打开项目根目录中 `components/uni-number-box/uni-number-box.vue` 组件, 修改 `watch` 节点中的 `inputValue` 侦听器如下:

```
1   inputValue(newVal, oldVal) {
2     // 官方提供的 if 判断条件, 在用户每次输入内容时, 都会调用 this.$emit("change",
newVal)
3     // if (+newVal !== +oldVal) {
4
5     // 新旧内容不同 && 新值内容合法 && 新值中不包含小数点
6     if (+newVal !== +oldVal && Number(newVal) && String(newVal).indexOf('.')
=== -1) {
7       this.$emit("change", newVal);
8     }
9   }
```

2. 修改完毕之后, NumberBox 组件只会把合法的、且不包含小数点的新值传递出去

9.1.10 修改购物车中商品的数量

1. 在 `store/cart.js` 模块中, 声明如下的 `mutations` 方法, 用来修改对应商品的数量:

```
1   // 更新购物车中商品的数量
2   updateGoodsCount(state, goods) {
3     // 根据 goods_id 查询购物车中对应商品的信息对象
4     const findResult = state.cart.find(x => x.goods_id === goods.goods_id)
5
6     if(findResult) {
7       // 更新对应商品的数量
8       findResult.goods_count = goods.goods_count
9       // 持久化存储到本地
10      this.commit('m_cart/saveToStorage')
11    }
12  }
```

2. 在 `cart.vue` 页面中, 通过 `mapMutations` 这个辅助函数, 将需要的 `mutations` 方法映射到当前页面中使用:

```
1   import badgeMix from '@mixins/tabbar-badge.js'
2   import { mapState, mapMutations } from 'vuex'
3
4   export default {
5     mixins: [badgeMix],
6     computed: {
7       ...mapState('m_cart', ['cart']),
8     },
9     data() {
10      return {}
11    },
12    methods: {
13      ...mapMutations('m_cart', ['updateGoodsState', 'updateGoodsCount']),
14      // 商品的勾选状态发生了变化
15      radioChangeHandler(e) {
16        this.updateGoodsState(e)
17      },
18      // 商品的数量发生了变化
19      numberChangeHandler(e) {
20        this.updateGoodsCount(e)
```

```

21     },
22     },
23 }

```

9.1.11 渲染滑动删除的 UI 效果

滑动删除需要用到 uni-ui 的 uni-swipe-action 组件和 uni-swipe-action-item。详细的官方文档请参考 [SwipeAction 滑动操作](#)。

1. 改造 `cart.vue` 页面的 UI 结构，将商品列表区域的结构修改如下（可以使用 `uSwipeAction` 代码块快速生成基本的 UI 结构）：

```

1  <!-- 商品列表区域 -->
2  <!-- uni-swipe-action 是最外层包裹性质的容器 -->
3  <uni-swipe-action>
4    <block v-for="(goods, i) in cart" :key="i">
5      <!-- uni-swipe-action-item 可以为子节点提供滑动操作的效果。需要通过
options 属性来指定操作按钮的配置信息 -->
6      <uni-swipe-action-item :options="options"
@click="swipeActionClickHandler(goods)">
7        <my-goods :goods="goods" :show-radio="true" :show-num="true"
@radio-change="radioChangeHandler" @num-change="numberChangeHandler"></my-
goods>
8      </uni-swipe-action-item>
9    </block>
10 </uni-swipe-action>

```

2. 在 `data` 节点中声明 `options` 数组，用来定义操作按钮的配置信息：

```

1  data() {
2    return {
3      options: [{
4        text: '删除', // 显示的文本内容
5        style: {
6          backgroundColor: '#C00000' // 按钮的背景颜色
7        }
8      }]
9    }
10 }

```

3. 在 `methods` 中声明 `uni-swipe-action-item` 组件的 `@click` 事件处理函数：

```

1  // 点击了滑动操作按钮
2  swipeActionClickHandler(goods) {
3    console.log(goods)
4  }

```

4. 美化 `my-goods.vue` 组件的样式：

```

1  .goods-item {
2    // 让 goods-item 项占满整个屏幕的宽度
3    width: 750rpx;
4    // 设置盒模型为 border-box
5    box-sizing: border-box;
6    display: flex;
7    padding: 10px 5px;
8    border-bottom: 1px solid #f0f0f0;
9  }

```

9.1.12 实现滑动删除的功能

1. 在 `store/cart.js` 模块的 `mutations` 节点中声明如下的方法，从而根据商品的 `id` 从购物车中移除对应的商品：

```

1  // 根据 Id 从购物车中删除对应的商品信息
2  removeGoodsById(state, goods_id) {
3    // 调用数组的 filter 方法进行过滤
4    state.cart = state.cart.filter(x => x.goods_id !== goods_id)
5    // 持久化存储到本地
6    this.commit('m_cart/saveToStorage')
7  }

```

2. 在 `cart.vue` 页面中，使用 `mapMutations` 辅助函数，把需要的方法映射到当前页面中使用：

```

1  methods: {
2    ...mapMutations('m_cart', ['updateGoodsState', 'updateGoodsCount',
3    'removeGoodsById']),
4    // 商品的勾选状态发生了变化
5    radioChangeHandler(e) {
6      this.updateGoodsState(e)
7    },
8    // 商品的数量发生了变化
9    numberChangeHandler(e) {
10     this.updateGoodsCount(e)
11   },
12   // 点击了滑动操作按钮
13   swipeActionClickHandler(goods) {
14     this.removeGoodsById(goods.goods_id)
15   }
16 }

```

9.2 收货地址区域

9.2.1 创建收货地址组件

1. 在 `components` 目录上鼠标右键，选择 `新建组件`，并填写组件相关的信息：



2. 渲染收货地址组件的基本结构:

```
1 <view>
2
3 <!-- 选择收货地址的盒子 -->
4 <view class="address-choose-box">
5   <button type="primary" size="mini" class="btnChooseAddress">请选择收货
   地址</button>
6 </view>
7
8 <!-- 渲染收货信息的盒子 -->
9 <view class="address-info-box">
10   <view class="row1">
11     <view class="row1-left">
12       <view class="username">收货人: <text>escook</text></view>
13     </view>
14     <view class="row1-right">
15       <view class="phone">电话: <text>138XXX5555</text></view>
16       <uni-icons type="arrowright" size="16"></uni-icons>
17     </view>
18   </view>
19   <view class="row2">
20     <view class="row2-left">收货地址: </view>
21     <view class="row2-right">河北省邯郸市肥乡区xxx 河北省邯郸市肥乡区xxx 河北
   省邯郸市肥乡区xxx 河北省邯郸市肥乡区xxx </view>
22   </view>
23 </view>
24
25 <!-- 底部的边框线 -->
26 <image src="/static/cart_border@2x.png" class="address-border"></image>
27 </view>
```

3. 美化收货地址组件的样式:

```
1 // 底部边框线的样式
2 .address-border {
3   display: block;
```

```

4     width: 100%;
5     height: 5px;
6 }
7
8 // 选择收货地址的盒子
9 .address-choose-box {
10    height: 90px;
11    display: flex;
12    align-items: center;
13    justify-content: center;
14 }
15
16 // 渲染收货信息的盒子
17 .address-info-box {
18    font-size: 12px;
19    height: 90px;
20    display: flex;
21    flex-direction: column;
22    justify-content: center;
23    padding: 0 5px;
24
25    // 第一行
26    .row1 {
27        display: flex;
28        justify-content: space-between;
29
30        .row1-right {
31            display: flex;
32            align-items: center;
33
34            .phone {
35                margin-right: 5px;
36            }
37        }
38    }
39
40    // 第二行
41    .row2 {
42        display: flex;
43        align-items: center;
44        margin-top: 10px;
45
46        .row2-left {
47            white-space: nowrap;
48        }
49    }
50 }

```

9.2.2 实现收货地址区域的按需展示

1. 在 data 中定义收货地址的信息对象：

```

1   export default {
2     data() {
3       return {
4         // 收货地址
5         address: {},
6       }
7     },
8   }

```

2. 使用 `v-if` 和 `v-else` 实现按需展示:

```

1   <!-- 选择收货地址的盒子 -->
2   <view class="address-choose-box" v-if="JSON.stringify(address) === '{}'">
3     <button type="primary" size="mini" class="btnChooseAddress">请选择收货地址+
  </button>
4   </view>
5
6   <!-- 渲染收货信息的盒子 -->
7   <view class="address-info-box" v-else>
8     <!-- 省略其它代码 -->
9   </view>

```

9.2.3 实现选择收货地址的功能

1. 为 请选择收货地址+ 的 `button` 按钮绑定点击事件处理函数:

```

1   <!-- 选择收货地址的盒子 -->
2   <view class="address-choose-box" v-if="JSON.stringify(address) === '{}'">
3     <button type="primary" size="mini" class="btnChooseAddress"
  @click="chooseAddress">请选择收货地址</button>
4   </view>

```

2. 定义 `chooseAddress` 事件处理函数, 调用小程序提供的 `chooseAddress()` API 实现选择收货地址的功能:

```

1   methods: {
2     // 选择收货地址
3     async chooseAddress() {
4       // 1. 调用小程序提供的 chooseAddress() 方法, 即可使用选择收货地址的功能
5       // 返回值是一个数组: 第 1 项为错误对象; 第 2 项为成功之后的收货地址对象
6       const [err, succ] = await uni.chooseAddress().catch(err => err)
7
8       // 2. 用户成功的选择了收货地址
9       if (err === null && succ.errMsg === 'chooseAddress:ok') {
10        // 为 data 里面的收货地址对象赋值
11        this.address = succ
12      }
13    }
14  }

```

3. 定义收货详细地址的计算属性:

```

1    computed: {
2      // 收货详细地址的计算属性
3      addstr() {
4        if (!this.address.provinceName) return ''
5
6        // 拼接 省，市，区，详细地址 的字符串并返回给用户
7        return this.address.provinceName + this.address.cityName +
8          this.address.countyName + this.address.detailInfo
9      }
10   }

```

4. 渲染收货地址区域的数据：

```

1    <!-- 渲染收货信息的盒子 -->
2    <view class="address-info-box" v-else>
3      <view class="row1">
4        <view class="row1-left">
5          <view class="username">收货人: <text>{{address.userName}}</text>
6        </view>
7        <view class="row1-right">
8          <view class="phone">电话: <text>{{address.telNumber}}</text></view>
9          <uni-icons type="arrowright" size="16"></uni-icons>
10         </view>
11       </view>
12       <view class="row2">
13         <view class="row2-left">收货地址: </view>
14         <view class="row2-right">{{addstr}}</view>
15       </view>
16     </view>

```

9.2.4 将 address 信息存储到 vuex 中

1. 在 `store` 目录中，创建用户相关的 `vuex` 模块，命名为 `user.js`：

```

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: {},
9      }),
10
11     // 方法
12     mutations: {
13       // 更新收货地址
14       updateAddress(state, address) {
15         state.address = address
16       },
17     },
18
19     // 数据包装器
20     getters: {},
21   }

```

2. 在 `store/store.js` 模块中, 导入并挂载 `user.js` 模块:

```
1 // 1. 导入 Vue 和 Vuex
2 import Vue from 'vue'
3 import Vuex from 'vuex'
4 // 导入购物车的 vuex 模块
5 import moduleCart from './cart.js'
6 // 导入用户的 vuex 模块
7 import moduleUser from './user.js'
8
9 // 2. 将 Vuex 安装为 Vue 的插件
10 Vue.use(Vuex)
11
12 // 3. 创建 Store 的实例对象
13 const store = new Vuex.Store({
14   // TODO: 挂载 store 模块
15   modules: {
16     // 挂载购物车的 vuex 模块, 模块内成员的访问路径被调整为 m_cart, 例如:
17     // 购物车模块中 cart 数组的访问路径是 m_cart/cart
18     m_cart: moduleCart,
19     // 挂载用户的 vuex 模块, 访问路径为 m_user
20     m_user: moduleUser,
21   },
22 })
23
24 // 4. 向外共享 Store 的实例对象
25 export default store
```

3. 改造 `address.vue` 组件中的代码, 使用 vuex 提供的 address 计算属性 替代 data 中定义的本地 address 对象:

```
1 // 1. 按需导入 mapState 和 mapMutations 这两个辅助函数
2 import { mapState, mapMutations } from 'vuex'
3
4 export default {
5   data() {
6     return {
7       // 2.1 注释掉下面的 address 对象, 使用 2.2 中的代码替代之
8       // address: {}
9     }
10  },
11  methods: {
12    // 3.1 把 m_user 模块中的 updateAddress 函数映射到当前组件
13    ...mapMutations('m_user', ['updateAddress']),
14    // 选择收货地址
15    async chooseAddress() {
16      const [err, succ] = await uni.chooseAddress().catch((err) => err)
17
18      // 用户成功的选择了收货地址
19      if (err === null && succ.errMsg === 'chooseAddress:ok') {
20        // 3.2 把下面这行代码注释掉, 使用 3.3 中的代码替代之
21        // this.address = succ
22
23        // 3.3 调用 Store 中提供的 updateAddress 方法, 将 address 保存到
24        Store 里面
25        this.updateAddress(succ)
26      }
27    },
28  },
29 }
```



```

27     },
28     computed: {
29       // 2.2 把 m_user 模块中的 address 对象映射当前组件中使用，代替 data 中
address 对象
30       ...mapState('m_user', ['address']),
31       // 收货详细地址的计算属性
32       addstr() {
33         if (!this.address.provinceName) return ''
34
35         // 拼接 省，市，区，详细地址 的字符串并返回给用户
36         return this.address.provinceName + this.address.cityName +
this.address.countyName + this.address.detailInfo
37       },
38     },
39   }

```

9.2.5 将 Store 中的 address 持久化存储到本地

1. 修改 `store/user.js` 模块中的代码如下：

```

1   export default {
2     // 开启命名空间
3     namespaced: true,
4
5     // state 数据
6     state: () => ({
7       // 3. 读取本地的收货地址数据，初始化 address 对象
8       address: JSON.parse(uni.getStorageSync('address') || '{}'),
9     }),
10
11    // 方法
12    mutations: {
13      // 更新收货地址
14      updateAddress(state, address) {
15        state.address = address
16
17        // 2. 通过 this.commit() 方法，调用 m_user 模块下的
saveAddressToStorage 方法将 address 对象持久化存储到本地
18        this.commit('m_user/saveAddressToStorage')
19      },
20      // 1. 定义将 address 持久化存储到本地 mutations 方法
21      saveAddressToStorage(state) {
22        uni.setStorageSync('address', JSON.stringify(state.address))
23      },
24    },
25
26    // 数据包装器
27    getters: {},
28  }

```

9.2.6 将 addstr 抽离为 getters

目的：为了提高代码的复用性，可以把收货的详细地址抽离为 getters，方便在多个页面和组件之间实现复用。

1. 剪切 `my-address.vue` 组件中的 `addstr` 计算属性的代码，粘贴到 `user.js` 模块中，作为一个 `getters` 节点：

```
1    // 数据包装器
2    getters: {
3        // 收货详细地址的计算属性
4        addstr(state) {
5            if (!state.address.provinceName) return ''
6
7            // 拼接 省，市，区，详细地址 的字符串并返回给用户
8            return state.address.provinceName + state.address.cityName +
9                state.address.countyName + state.address.detailInfo
10        }
11    }
```

2. 改造 `my-address.vue` 组件中的代码，通过 `mapGetters` 辅助函数，将 `m_user` 模块中的 `addstr` 映射到当前组件中使用：

```
1    // 按需导入 mapGetters 辅助函数
2    import { mapState, mapMutations, mapGetters } from 'vuex'
3
4    export default {
5        // 省略其它代码
6        computed: {
7            ...mapState('m_user', ['address']),
8            // 将 m_user 模块中的 addstr 映射到当前组件中使用
9            ...mapGetters('m_user', ['addstr']),
10        },
11    }
```

9.2.7 重新选择收货地址

1. 为 `class` 类名为 `address-info-box` 的盒子绑定 `click` 事件处理函数如下：

```
1    <!-- 渲染收货信息的盒子 -->
2    <view class="address-info-box" v-else @click="chooseAddress">
3        <!-- 省略其它代码 -->
4    </view>
```

9.2.8 解决收货地址授权失败的问题

如果在选择收货地址的时候，用户点击了**取消授权**，则需要进行**特殊的处理**，否则**用户将无法再次选择收货地址**！

1. 改造 `chooseAddress` 方法如下：

```
1    // 选择收货地址
2    async chooseAddress() {
3        // 1. 调用小程序提供的 chooseAddress() 方法，即可使用选择收货地址的功能
4        // 返回值是一个数组：第1项为错误对象；第2项为成功之后的收货地址对象
5        const [err, succ] = await uni.chooseAddress().catch(err => err)
6
7        // 2. 用户成功的选择了收货地址
8        if (succ && succ.errMsg === 'chooseAddress:ok') {
9            // 更新 vuex 中的收货地址
10            this.updateAddress(succ)
11        }
12    }
```

```

11     }
12
13     // 3. 用户没有授权
14     if (err && err.errMsg === 'chooseAddress:fail auth deny') {
15         this.reAuth() // 调用 this.reAuth() 方法，向用户重新发起授权申请
16     }
17 }

```

2. 在 `methods` 节点中声明 `reAuth` 方法如下：

```

1 // 调用此方法，重新发起收货地址的授权
2 async reAuth() {
3     // 3.1 提示用户对地址进行授权
4     const [err2, confirmResult] = await uni.showModal({
5         content: '检测到您没打开地址权限，是否去设置打开？',
6         confirmText: "确认",
7         cancelText: "取消",
8     })
9
10    // 3.2 如果弹框异常，则直接退出
11    if (err2) return
12
13    // 3.3 如果用户点击了“取消”按钮，则提示用户“您取消了地址授权！”
14    if (confirmResult.cancel) return uni.$showMsg('您取消了地址授权! ')
15
16    // 3.4 如果用户点击了“确认”按钮，则调用 uni.openSetting() 方法进入授权页面，
    让用户重新进行授权
17    if (confirmResult.confirm) return uni.openSetting({
18        // 3.4.1 授权结束，需要对授权的结果做进一步判断
19        success: (settingResult) => {
20            // 3.4.2 地址授权的值等于 true，提示用户“授权成功”
21            if (settingResult.authSetting['scope.address']) return
            uni.$showMsg('授权成功！请选择地址')
22            // 3.4.3 地址授权的值等于 false，提示用户“您取消了地址授权”
23            if (!settingResult.authSetting['scope.address']) return
            uni.$showMsg('您取消了地址授权! ')
24        }
25    })
26 }

```

9.2.9 解决 iPhone 真机上无法重新授权的问题

问题说明：在 iPhone 设备上，当用户取消授权之后，再次点击选择收货地址按钮的时候，无法弹出授权的提示框！

1. 导致问题的原因 - 用户取消授权后，再次点击“选择收货地址”按钮的时候：

- 在 **模拟器** 和 **安卓真机** 上，错误消息 `err.errMsg` 的值为 `chooseAddress:fail auth deny`
- 在 **iPhone 真机** 上，错误消息 `err.errMsg` 的值为 `chooseAddress:fail authorize no response`

2. 解决问题的方案 - 修改 `chooseAddress` 方法中的代码，进一步完善用户没有授权时的 `if` 判断条件即可：

```

1 async chooseAddress() {
2     // 1. 调用小程序提供的 chooseAddress() 方法，即可使用选择收货地址的功能

```

```
3      //      返回值是一个数组：第1项为错误对象；第2项为成功之后的收货地址对象
4      const [err, succ] = await uni.chooseAddress().catch(err => err)
5
6      // 2. 用户成功的选择了收货地址
7      if (succ && succ.errMsg === 'chooseAddress:ok') {
8          this.updateAddress(succ)
9      }
10
11     // 3. 用户没有授权
12     if (err && (err.errMsg === 'chooseAddress:fail auth deny' || err.errMsg
13     === 'chooseAddress:fail authorize no response')) {
14         this.reAuth()
15     }
```