## 前言

想走完一套完整的购物流程，地址的选择是必不可少的环节，本章节我们就来分析地址的选择以及地址栏管理。本章节代码会涉及到添加两个页面，一个是地址栏列表，另一个是地址栏添加和编辑页面。

## 地址栏列表页

首先我们在 src/views 下添加地址栏列表页面 Address.vue ，代码如下：

<template>

<div class="address-box">

<s-header :title="'地址管理'" :back="'/user'"></s-header>

<div class="address-item">

<van-address-list

v-if="from != 'mine'"

v-model="chosenAddressId"

:list="list"

default-tag-text="默认"

@add="onAdd"

@edit="onEdit"

@select="select"

/>

<van-address-list

v-else

v-model="chosenAddressId"

:list="list"

default-tag-text="默认"

@add="onAdd"

@edit="onEdit"

/>

</div>

</div></template>

<script>**import** { **Toast** } **from** 'vant';**import** sHeader **from** '@/components/SimpleHeader'**import** { getAddressList } **from** '../service/address'**export** **default** {

components: {

sHeader

},

**data**() {

**return** {

chosenAddressId: '1',

list: [],

from: this.$route.query.from,

}

},

**async** **mounted**() {

**const** { data } = **await** **getAddressList**()

this.list = data.**map**(item => {

**return** {

id: item.addressId,

name: item.userName,

tel: item.userPhone,

address: `${item.provinceName} ${item.cityName} ${item.regionName} ${item.detailAddress}`,

isDefault: !!item.defaultFlag

}

})

},

methods: {

**onAdd**() {

this.$router.**push**({ path: `address-edit?type=add&from=${this.**from**}` })

},

**onEdit**(item, index) {

this.$router.**push**({ path: `address-edit?type=edit&addressId=${item.id}&from=${this.**from**}` })

},

**select**(item, index) {

this.$router.**push**({ path: `create-order?addressId=${item.id}&from=${this.**from**}` })

}

}

}</script>

<style lang="less">

**@import** '../common/style/mixin';

.address-box {

.van-radio\_\_icon {

display: none;

}

.address-item {

margin-top: 44px;

.van-button {

background: @primary;

border-color: @primary;

}

}

}</style>

上述代码中 from 参数是浏览器地址上带的参数，如果我是从生产订单页面进入了选择地址的页面，那么是需要选择一个地址作为发货地址的，所以通过 v-if="from != 'mine'" 去展示有 select 方法的 van-address-list 组件。地址管理页面还有一个入口，那就是“我的”页面进入，只需要对地址栏进行增删改处理，所以就无须添加 select 方法去选择一个地址。

onAdd 和 onEdit 方法内对应的跳转路由 address-edit 都是同一个页面，通过浏览器地址传入 id 参数作为是编辑还是新增的判断依据，具体的页面逻辑会在下文进行分析描述。

Vant 组件库为我们提供了 van-address-list 组件，我们不必再从头设计列表样式，只需在在 main.js 脚本内注册相关组件：

**import** **Vue** **from** 'vue'**import** md5 **from** 'js-md5';**import** **App** **from** './App.vue'**import** router **from** './router'**import** store **from** './store'**import** {**Button**, **Form**, **Field**, **Toast**, **Icon**, **Swipe**, **SwipeItem**, **SwipeCell**, **PullRefresh**, **List**, **Tab**, **Tabs**, **GoodsAction**, **GoodsActionIcon**, **GoodsActionButton**, **CheckboxGroup**, **Checkbox**, **SubmitBar**, **Stepper**, **Popup**, **AddressList** } **from** 'vant'**import** 'lib-flexible/flexible'

**Vue**.**use**(**Form**).**use**(**Field**).**use**(**Toast**).**use**(**Button**).**use**(**Icon**)

.**use**(**Swipe**).**use**(**SwipeItem**).**use**(**SwipeCell**).**use**(**PullRefresh**)

.**use**(**List**).**use**(**Tab**).**use**(**Tabs**).**use**(**GoodsAction**).**use**(**GoodsActionButton**)

.**use**(**GoodsActionIcon**).**use**(**CheckboxGroup**).**use**(**Checkbox**)

.**use**(**SubmitBar**).**use**(**Stepper**).**use**(**Popup**).**use**(**AddressList**)

**Vue**.config.productionTip = false

**new** **Vue**({

router,

store,

render: h => **h**(**App**)

}).$mount('#app')

**Vue**.**prototype**.$md5 = md5;window.vRouter = router

别忘了，添加相应的路由：

*// router.js*

...

{

path: '/address',

name: 'address',

meta: {

index: 2

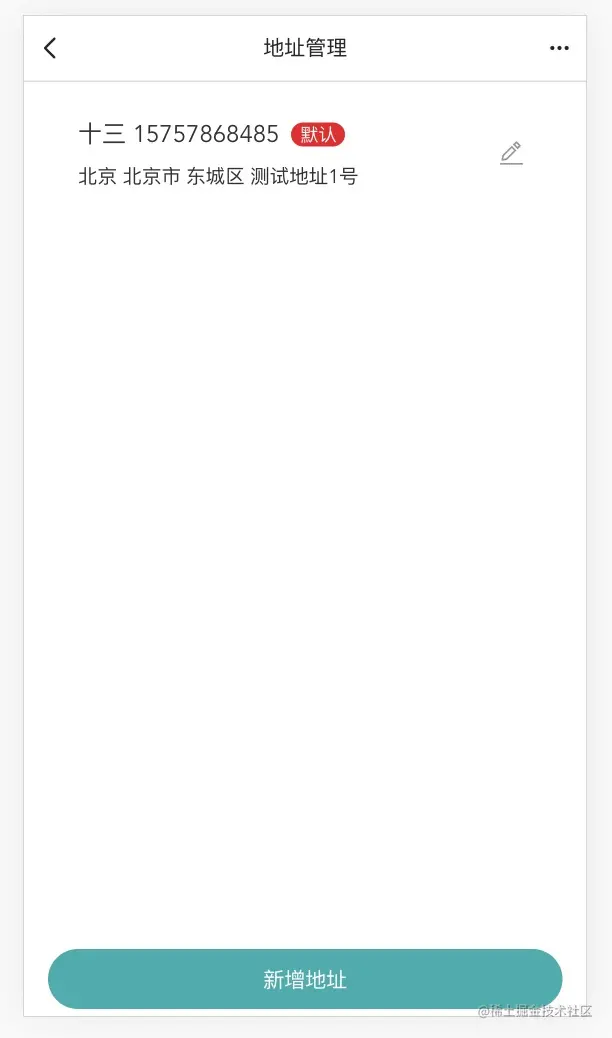
},

component: () => **import**(*/\* webpackChunkName: "address" \*/* '../views/Address.vue'),

},

...

浏览器打开相应地址展示如下：



由于我事先添加过一个默认地址，所以展示如上图所示。同学们应该是空的列表页面。点击下方「新增地址」按钮，我们将会跳转到添加地址页面，下面会介绍添加地址页面的逻辑。

## 添加、编辑地址详情页面

Vant 在提供地址列表组件的同时，也为我们提供了编辑地址详情组件，名字叫 AddressEdit ，同样需要在 main.js 下全局注册，这里就不赘述，同学们自行添加。

在 src/views 目录下添加 AddressEdit.vue 文件，用于新增和编辑地址，代码如下所示：

<template>

<div class="address-edit-box">

<s-header :title="`${type == 'add' ? '新增地址' : '编辑地址'}`"></s-header>

<van-address-edit

class="edit"

:area-list="areaList"

:address-info="addressInfo"

:show-delete="type == 'edit'"

show-set-default

show-search-result

:search-result="searchResult"

:area-columns-placeholder="['请选择', '请选择', '请选择']"

@save="onSave"

@delete="onDelete"

/>

</div>

</template>

<script>

import { Toast } from 'vant'

import sHeader from '@/components/SimpleHeader'

import { addAddress, EditAddress, DeleteAddress, getAddressDetail } from '../service/address'

import { tdist } from '@/common/js/utils'

export default {

components: {

sHeader

},

data() {

return {

areaList: {

province\_list: {},

city\_list: {},

county\_list: {}

},

searchResult: [],

type: 'add',

addressId: '',

addressInfo: {},

from: ''

}

},

async mounted() {

// 省市区列表构造

let \_province\_list = {}

let \_city\_list = {}

let \_county\_list = {}

tdist.getLev1().forEach(p => {

\_province\_list[p.id] = p.text

tdist.getLev2(p.id).forEach(c => {

\_city\_list[c.id] = c.text

tdist.getLev3(c.id).forEach(q => \_county\_list[q.id] = q.text)

})

})

this.areaList.province\_list = \_province\_list

this.areaList.city\_list = \_city\_list

this.areaList.county\_list = \_county\_list

const { addressId, type, from } = this.$route.query

this.addressId = addressId

this.type = type

this.from = from || ''

if (type == 'edit') {

const { data: addressDetail } = await getAddressDetail(addressId)

let \_areaCode = ''

const province = tdist.getLev1()

Object.entries(this.areaList.county\_list).forEach(([id, text]) => {

// 先找出当前对应的区

if (text == addressDetail.regionName) {

// 找到区对应的几个省份

const provinceIndex = province.findIndex(item => item.id.substr(0, 2) == id.substr(0, 2))

// 找到区对应的几个市区

const cityItem = Object.entries(this.areaList.city\_list).filter(([cityId, cityName]) => cityId.substr(0, 4) == id.substr(0, 4))[0]

// 对比找到的省份和接口返回的省份是否相等，因为有一些区会重名

if (province[provinceIndex].text == addressDetail.provinceName && cityItem[1] == addressDetail.cityName) {

\_areaCode = id

}

}

})

this.addressInfo = {

id: addressDetail.addressId,

name: addressDetail.userName,

tel: addressDetail.userPhone,

province: addressDetail.provinceName,

city: addressDetail.cityName,

county: addressDetail.regionName,

addressDetail: addressDetail.detailAddress,

areaCode: \_areaCode,

isDefault: !!addressDetail.defaultFlag

}

}

},

methods: {

async onSave(content) {

const params = {

userName: content.name,

userPhone: content.tel,

provinceName: content.province,

cityName: content.city,

regionName: content.county,

detailAddress: content.addressDetail,

defaultFlag: content.isDefault ? 1 : 0,

}

if (this.type == 'edit') {

params['addressId'] = this.addressId

}

const { message } = await this.type == 'add' ? addAddress(params) : EditAddress(params)

Toast('保存成功')

setTimeout(() => {

this.$router.push({ path: `address?from=${this.from}` })

}, 1000)

},

async onDelete() {

const { data } = await DeleteAddress(this.addressId)

Toast('删除成功')

setTimeout(() => {

this.$router.push({ path: 'address' })

}, 1000)

}

}

}

</script>

<style lang="less">

@import '../common/style/mixin';

.address-edit-box {

margin-top: 44px;

.van-address-edit {

.van-button--danger {

background: @primary;

border-color: @primary;

}

.van-switch--on {

background: @primary;

}

}

}

</style>

tdist 是全国省市区静态资源，内部抛出了几个方法。

* tdist.getLev1：获取地址栏第一级省级。
* tdist.getLev2：获取地址栏第二级市级。
* tdist.getLev1：获取地址栏第三级县、区等。

进入页面后 mounted 内的代码会初始化省市区，通过下面代码块：

**let** \_province\_list = {} *// 声明省级对象***let** \_city\_list = {} *// 声明市级对象***let** \_county\_list = {} *// 声明县、区级对象*

tdist.**getLev1**().**forEach**(p => {

\_province\_list[p.id] = p.text

tdist.**getLev2**(p.id).**forEach**(c => {

\_city\_list[c.id] = c.text

tdist.**getLev3**(c.id).**forEach**(q => \_county\_list[q.id] = q.text)

})

})

this.areaList.province\_list = \_province\_listthis.areaList.city\_list = \_city\_listthis.areaList.county\_list = \_county\_list

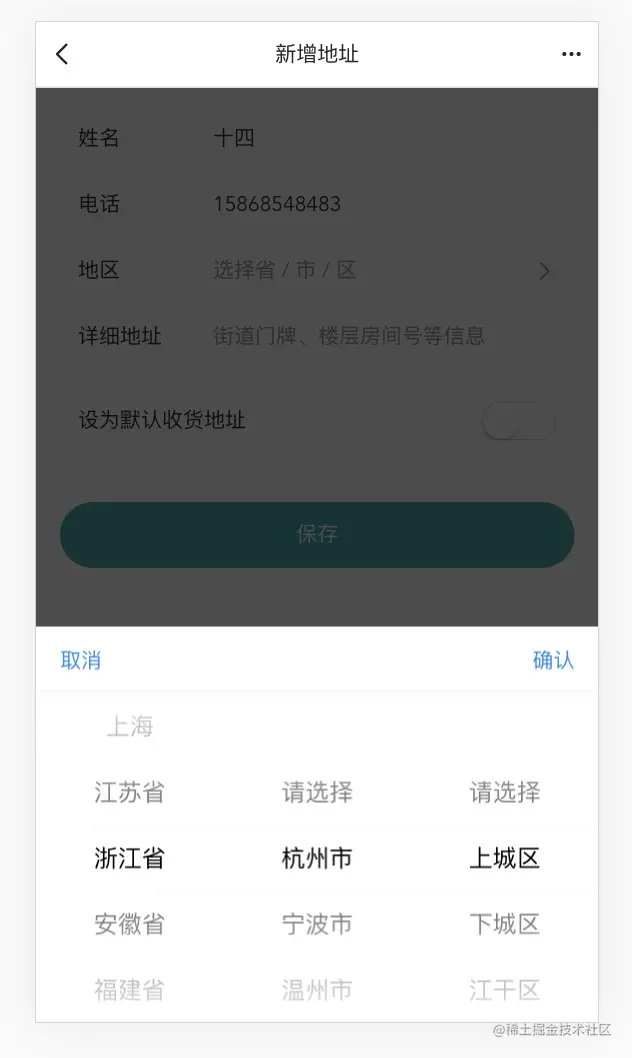
tdist.getLev1() 返回一个数组，通过 forEach 循环遍历，将省级数据的 id 作为键，省级数据的 text 作为值，赋值给 \_province\_list 变量。

在遍历省级的时候，通过省级 id 拿到当前省的市级数据 tdist.getLev2(p.id) 再次通过 forEach 遍历，同样通过 \_city\_list[c.id] = c.text 方式将 id 和对应的 text 赋值给 \_city\_list 变量。

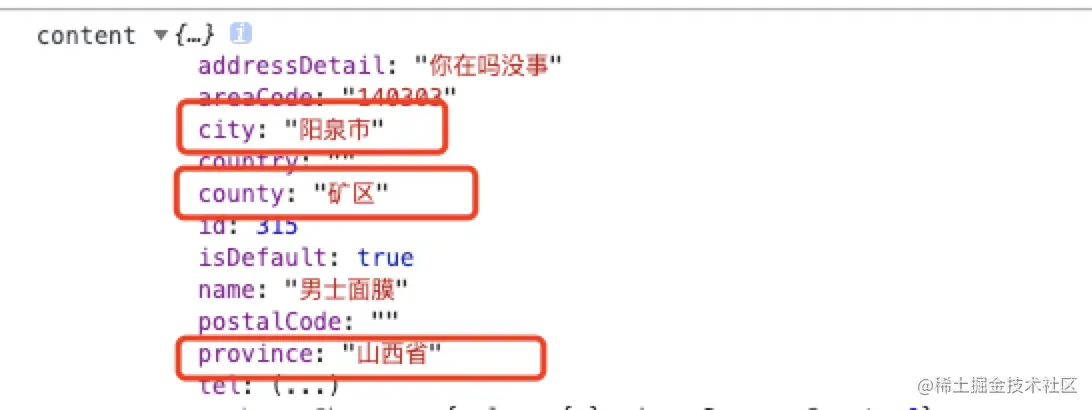
在遍历市级数据的时候，通过市级 id 拿到当前市级下的区、县级数据 tdist.getLev3(c.id) ，再次 forEach 遍历，通过 \_county\_list[q.id] = q.text 将县、区级数据赋值给 \_county\_list 变量。

经过上述过程之后，我们将 province\_list、\_city\_list、\_county\_list 三个变量赋值给 this.areaList 如上述代码所示。之所以要这样做，是因为 van-address-edit 组件接收一个变量 area-list，我们必须要将数据构造成这个变量所支持的对象格式，才能正确的渲染出地址栏。

初始化完毕之后，赋值给 areaList 变量，组件 van-address-edit 中的 :area-list 属性便会初始化三级联动，若是浏览器上的查询参数 type 传的是 edit。则需要通过地址 id 变量去获取相应的地址详情进行编辑，否则为新增地址，浏览器效果如下所示：



设置好是否为默认地址之后，点击保存按钮，接口回调成功则回到地址列表页。

点击保存之后 onSave 方法回掉一个参数 content 打印之后内容如下：  我们可以取上图红框内的三个数据作为省市区的参数，如下所示：

*// 上传参数***const** params = {

userName: content.name,

userPhone: content.tel,

provinceName: content.province,

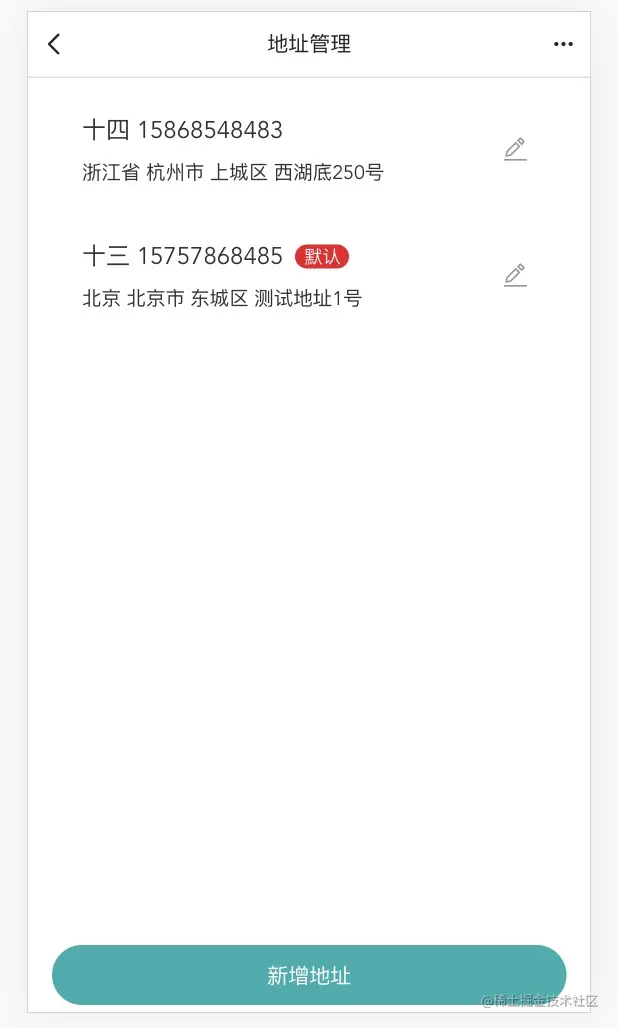
cityName: content.city,

regionName: content.county,

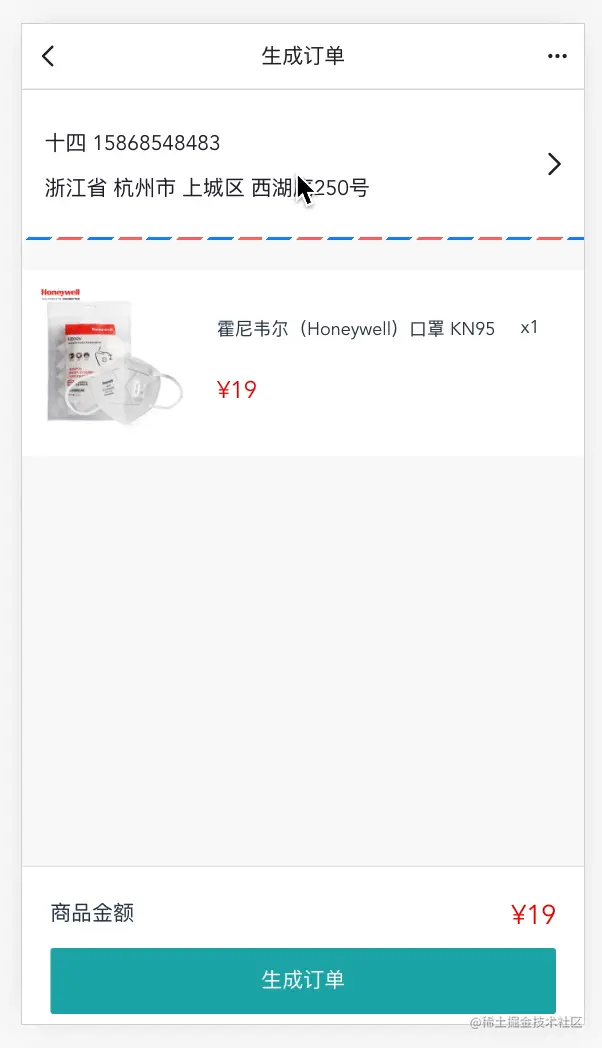
detailAddress: content.addressDetail,

defaultFlag: content.isDefault ? 1 : 0,

}



那么上节课的订单生成页面也就顺理成章，能点击顶部的地址信息栏，跳转到地址列表页面进行地址的筛选。如下所示：



还有一点要注意的是，点击编辑进来的时候，我们只能获取到省市区的名字，只有名字的话，省市区那一栏是没有默认数据的，就比如我设置的是浙江省杭州市西湖区，点击编辑进来之后，在地址那一栏是看不到默认数据的，我们通过如下形式，去解析出地址栏的 id 数据：

**if** (type == 'edit') {

**const** { data: addressDetail } = **await** **getAddressDetail**(addressId)

**let** \_areaCode = ''

**const** province = tdist.**getLev1**()

**Object**.**entries**(this.areaList.county\_list).**forEach**(([id, text]) => {

*// 先找出当前对应的区*

**if** (text == addressDetail.regionName) {

*// 找到区对应的几个省份*

**const** provinceIndex = province.**findIndex**(item => item.id.**substr**(0, 2) == id.**substr**(0, 2))

*// 找到区对应的几个市区*

**const** cityItem = **Object**.**entries**(this.areaList.city\_list).**filter**(([cityId, cityName]) => cityId.**substr**(0, 4) == id.**substr**(0, 4))[0]

*// 对比找到的省份和接口返回的省份是否相等，因为有一些区会重名*

**if** (province[provinceIndex].text == addressDetail.provinceName && cityItem[1] == addressDetail.cityName) {

\_areaCode = id

}

}

})

this.addressInfo = {

id: addressDetail.addressId,

name: addressDetail.userName,

tel: addressDetail.userPhone,

province: addressDetail.provinceName,

city: addressDetail.cityName,

county: addressDetail.regionName,

addressDetail: addressDetail.detailAddress,

areaCode: \_areaCode,

isDefault: !!addressDetail.defaultFlag

}

}

首先我们获取 addressDetail 地址详情，然后通过 regionName 先找出当前对应的区，因为某些区在不同的省份会重名，比如江苏省和福建省都有鼓楼区，所以这边我们可能会返回好几个省。省级数据 code 编码和它的区前面两个是一样的，如北京是 110000，下面的区都是 11 开头的，如东城区是 110101 ，所以我们对比前两个数据就能分析出当前区级是属于哪个省级的。最后我们再对比找到的省份和接口返回的省份是否相等，因为有一些区会重名。最终我们的目的是拿到县、区级的 code 编码，因为 van-address-edit 组件的 address-info 属性，接收到县、区级的 code 编码就能显示对应的省市了。

## 总结

地址的增删改查不是难点，难点在于三级列表如何建立，本章节的全国省市区静态资源是我找了很久才找到的资源，可以运用于真实项目开发，有需要的同学可以作为生产资源保存，待日后用到之时，不必满大街的找资源。