1.v-once:单独存在，可定义在dev或h中，只显示初始化一次值，再次响应值不变；

2.v-html:将带有html系列标签内容映射到命令中，自动解析出来；

3.v-text:将字符串映射到命令中，显示内容，相当于{{}}，展示数据；

4.v-pre:单独存在，将标签引用中的内容原样输出；

5.v-cloak:单独存在，当遇到延迟响应js时，执行完之后会自动去掉，不会展示错误内容，结合display样式实现；

6.v-bind:绑定到标签特有属性上如href或src上，回显data数据，使用冒号代替；

能绑定在:class上，用对象{}或数组[]表示，{}键值对形式，[]再data中值罗列；

还可以绑定在:style上，用[]表示，在data中用{}定义好样式，赋值到数组中；

注：:class和:style用方法实现，需要加方法小括号。

7.computed:{}作用:将data中的数据强化为属性，反显到html中，不加（），属性中存在set()和get()方法；

与methods方法区别：computed多次请求只执行一次，存在缓存，methods执行多次；

8.使用let或const代替var：let定义变量，const定义常量，var没有块级作用域，比如if语句块或for语句块；

9.const定义对象时，可以修改对象中的属性，不能修改常量声明值的引用；

10.v-on: 监听事件，对应语法糖@，如：v-on:click == @click

11.不带小括号调用单个参数方法打印的是浏览器event事件，带小括号打印undefined，多个参数的情况下不传参数，第一个参数打印的是浏览器event事件，其余参数打印的undefined，使用$event来调用浏览器event事件；

12.修饰符.stop:用于对事件的操作后设置@click.stop，功能取消冒泡操作上级事件；

13.修饰符.prevent:@click.prevent取消标签自带属性功能的作用；

14.@keyup.enter:@keyup键盘抬起事件，键盘监听enter键来触发事件；

15.修饰符.once:@click.once按钮只执行一次；

16.v-if,v-else-if和v-else:v-if和v-else-if后跟表达式，v-else用在末尾；

17.label标签for功能:可以点击文字触发编辑框获取焦点，for中的内容要和编辑框id名称保持一致；

18.操作输入框切换还带有文字问题，解决方法：每个输入框添加key属性输入不同的值，切换自动清除文字；

19.v-show:判断标签内容是否需要展示，与v-if的区别，当为false时，v-if所在标签直接删除，v-show所在标签添加样式display:none，应用场景，频繁切换标签时使用v-show，只有一次切换时使用v-if。

20.v-for遍历数组：<li v-for="(item, index) in nameList" :key="item">{{index + 1}}-{{item}}</li>，item数组值，index数组索引值；

21.v-for遍历对象：<li v-for="(value, key, index) in nameObj" :key="value">{{index+1}}-{{key}}:{{value}}</li>，value对象元素值，key对象元素key，对象元素的索引值；

22.v-for中:key的作用：在数组中间添加元素是，能够从虚拟dom到浏览器dom一样的元素能够复用，只会针对新的元素直接添加，提交效率；

23.v-for对数组响应式的一些方法：

    ①.unshift()：在数组首部添加元素，可一次性添加多条；

②.push()：在数组末尾添加元素，可一次性添加多条；

③.shift()：不含参数，删除数组首个元素；

④.pop()：不含参数，删除数组最后一个元素；

⑤.sort()：数组正序排序；

⑥.reverse()：数组倒叙排序；

⑦.splice()：删除元素/替换元素/插入元素，只有一个参数时，删除位置后所有元素；当有两个参数时，删除第一个参数位置指定第二个参数元素个数；当有三个参数时，将第一个参数位置后指定的第二个参数个数替换为第三个参数的内容，如果第二个参数时0，就会在第一个参数位置后新增第三个参数的内容；

24.:disabled:按钮置灰不可用，赋值true/false；

25.方法.toFixed(2):生成两位小数，默认时0；

26.filters:{}作用:将符号‘|’前数字传递进来，自定义加强并返回；

27.for遍历集合三种方式：

①for(let i = 0; i < this.books.length; i++) {console.log(this.books[i].name);}

②for(let i in this.books) {console.log(this.books[i].name);}

③for(let book of this.books) {console.log(book.name);}

28.高阶函数应用：

①this.dataList = this.dataList.filter(function(n){return n < 100;})，过滤返回ture/false，筛选返回到新的集合中；

②this.dataList = this.dataList.map(function(n) {return n \* 2;})，遍历集合，扩展集合内容，返回到新的集合当中；

③this.totalResult = this.dataList.reduce(function(previousValue, n){return previousValue + n;}, 0)，对整个集合求总值，返回到新的属性当中；

29.v-model：用于表单数据的双向绑定，既可以展示data属性数据，又可以将输入的数据映射到data属性中；

  ①绑定在text框；可使用方法.lazy / .number / .trim，.lazy当移除焦点才会将数据双向映射;.number先设置type=”number”，再加上.number输入的格式是数字，不加就是字符类型；.trim:消除字符串前后空格；

  ②绑定在radio框：name属性可以使单选框单个选中，使用v-model可以代替name属性可以实现单选；

  ③绑定在checkbox框：单个选中复选框使用，映射到data属性是true/false形式；多个选中复选框使用，映射data属性是数组形式；

  ④绑定在select框：v-model要写在select上，单个选中下拉框使用，映射到data属性是字符串形式；多个选中下拉框使用，映射到data属性是字符串数组形式；

30.值绑定：使用v-for遍历数据时，表单属性使用:value获取到数组列表值；

31.使用组件步骤：为了一次定义多次使用，1.创建组件构造器对象；2.注册组件；3.使用组件；

// 1.创建组件构造器

const cpnC = Vue.extend({

  template:

  `<div>

     <h2>vue模板学习</h2>

     <p>加油学习</p >

   </div>`

})

// 2.注册组件

Vue.component('cpn', cpnC);

// 3.就可以在html元素中，使用组件

<cpn></cpn>

32.注册组件有两种方式，1.全局注册Vue.component('cpn', cpnC);所有用el挂载的对象都可以使用；2.范围内单个对象使用，使用components:{cpn: cpnC}，只有范围内当前el挂载的对象可以使用；

33.父/子组件：定义子组件；再定义父组件，在父组件中使用components:{sonCpnInfo : sonCpn}挂载子组件并且在template当中设置<sonCpnInfo></sonCpnInfo>，获取子组件template；最后将父组件挂载到el所在的对象components: {cpn:parentCpn}中；HTML使用<cpn></cpn>就可以获取到父和子组件的模板信息。

34.组件语法糖形式及模板抽离：全局方式Vue.component("cpn", {template: '#comp'})，且#comp是HTML元素定义的<template/>内容<template id="comp"> <div><h1>aa</h1> <p>bb</p></div></template>；在父组件内部使用语法糖方式：components: {'cpn': {template: '#comp'}}，同理#comp也是HTML定义的<template/>定义的模板；注：组件存在多个元素用<div>包起来。

35.组件模板数据实现响应式：模板中的数据不能放到root中实现不了数据响应，将响应的数据放到定义模板的data() { return {title : '组件'}}函数中（与root中data:{}属性有区别，函数返回的数据存在多份且互不影响），可以实现响应式，模板中也存在methods:{};

36.父组件传递子组件：父组件在data:{}中定义好数据，子组件是用props: {sonmovies : {type: Array,default() {return []} },sonmessage : {type: String,default: "",required: true}}接收，并取个别名，root配置components: {'cpn': sonComp}来使用模板，配置绑定<cpn :sonmovies="movies" :sonmessage="message"></cpn>，子组件才能使用父组件的信息。注：子组件接收取得别名是驼峰状，配置绑定就需要配置为:son-movies。

37.子组件传递父组件：多数都会以事件函数的形式传递，子组件定义方法传递参数信息给父组件，sentSonItem(index, item) {this.$emit("itemclick", index, item);}，监听函数HTML<cpn @itemclick="cpnClick"></cpn>，父组件定义方法获取子组件的信息，cpnClick(index, item) {console.log(index, item);}，注：①子组件定义监听方法itemclick小写字母，不采用驼峰状；②监听函数父组件调用不传递参数，父组件定义方法和子组件定义方法参数顺序一致；

38.父组件获取子组件的属性及方法：this.$children[0].属性/方法；可以使用this.$refs.设置在组件上ref的名称.属性/方法。

39.子组件获取父组件的属性及方法：this.$parent.属性/方法；如果存在多层父子组件，可以使用this.$root.属性/方法获取根路径的数据。

40.插槽slot使用：子组件模板通过<slot>aa</slot>设置插槽，默认值aa，在父组件中来控制插槽位置的内容，具名插槽：子组件通过给插在设置name属性，父组件就可以根据slat=””具体对某个插槽设置内容；

41.插槽作用域：子组件提供数据，不同父组件拿到子组件的数据会有不同的展示，将子组件数据绑定在插槽上<slot :data=”sonData”></slot>，父组件需要先拿到子组件的数据，在对样式进行修改<template slot-scope=”slot”>{{slot.data.join(‘ - ‘)}}</template>，slot-scope=”slot”，引用着插槽的内容，名称随意起，slot.data，获取子组件的数据。

42.模块化导入导出：commentjs形式：module.exports= {add, mul}模块化导出；const {add, mul} = require('./mathUtils.js')模块化导入。

ES6形式：导出使用export const age = 50，导入使用import {age} from "./personInfo"，每个文件中只能导出一个默认的export default。

43.webpack：前端模块化打包工具，模块化：每个js文件都是模块化存在，互不影响，低耦合，必须依赖node环境，node要依赖各种包就需要npm工具来管理；

44.安装完node后就可以安装webpack，node -v安装查看版本，webpack -version是否已安装查看版本，npm install [webpack@3.6.0](mailto:webpack@3.6.0) -g，-g是全局安装webpack；

45.本地安装webpack: npm install webpack@3.6.0 --save-dev，全局安装和本地安装webpack的区别是当全局安装的webpack与项目的版本不一致时，就需要本地安装webpack相应的版本；本地安装完成后在package.json中就会生成devDependencies及版本号；

46.webpack ./src/main.js ./dist/bundle.js进行模块化打包：./src/main.js要进行打包的js源码路径，可以将关联引用文件一起进行打包（入口），./dist/bundle.js模块化打包后的路径；html模块打包后的路径，就能展示效果（出口）；

47.自动解析入口和出口，只使用webpack打包：创建名称webpack.config.js文件

const path = require('path') //调用node的path路径  
module.exports= {  
 entry: './src/main.js',//入口   
 output: { //出口  
 path : path.resolve(\_\_dirname, 'dist'),//出口路径，绝对路径，\_\_dirname当前文件路径  
 filename : 'bundle.js' //文件名  
 }  
}

如果用到了node，就需要生成package.json配置文件，使用命令npm init；

package.json生成后，文件中设置"scripts": {"build": "webpack"}，就可以使用npm run build来代替直接使用webpack；默认终端的webpack使用的全局安装，配置scripts后，找本地安装，本地没有再找全局安装；

1. 模块化添加css样式：require('./css/normal.css')将css导入到js中，再在webpack.config.js中配置module: {rules: [{test: /\.css$/,use: [ 'style-loader', 'css-loader' ]}]}，命令安装css-loader和style-loader（npm install --save-dev css-loader@2.0.2和npm install --save-dev style-loader@0.23.1），再npm run install运行项目。

49.利用less编写css样式：require('./css/special.less')将less编写的css导入到js中，npm install --save-dev less-loader@4.1.0 less利用命令加载less-loader，在webpack.config.js配置{  
test: /\.less$/,use: [{loader: "style-loader"}, {loader: "css-loader"}, {loader: "less-loader"}]}生成loader配置，（document.writeln("hello world")添加文字），npm run build运行项目。

50.模块化图片加载：将图片引入css样式中，再将样式引入js中，npm install --save-dev url-loader@1.1.2利用命令加载url-loader，在webpack.config.js中配置{test: /\.(png|jpg|gif)$/,use: [{loader: 'url-loader',options: {limit: 8192,name: 'img/[name].[hash:8].[ext]'},}]}，如果图片小于limit:8192，图片就会转成base64展示在浏览器上，如果图片大于limit:8192，就需要安装npm install --save-dev file-loader@0.11.2，并且在output{publicPath:’dist/’}配置访问图片的路径，文件名称不需要系统自动生成就需要配置name: 'img/[name].[hash:8].[ext]'。

51.模块化打包后的文件将ES6的写法改成ES5支撑浏览器：命令npm install --save-dev babel-loader@7 babel-core babel-preset-es2015安装babel-loader，在webpack.config.js中配置{test: /\.js$/,exclude: /(node\_modules|bower\_components)/,use: {loader:'babel-loader',options: {presets: ['es2015']}}}，运行项目会将ES6的语法改成ES5。

52.使用vue开发项目：安装npm install --save vue@2.5.21，js中引入并编写vue代码

import Vue from 'vue' const app = new Vue({el: '#app', data: {message: "hello webpack"}})

项目需要引入template就需要在webpack.config.js中配置resolve: {alias: {'vue$': 'vue/dist/vue.esm.js' }}。

1. 在vue开发的代码中添加template创建模板，与el的区别：会将template模板内容覆盖el的标识符的位置，理想情况多个js只使用一个HTML。
2. 使用Vue Component开发template：npm install --save-dev vue-loader@13.0.0 vue-template-compiler@2.5.21安装vue-loader和vue-template-compiler的loader，在webpack.config.js配置{test: /\.vue$/,use: ['vue-loader']}，将生成好的模板导入到js当中import App from './vue/App.vue'  
   const app = new Vue({  
    el: '#app',  
    template: '<App/>',  
    components: {  
    App  
    }  
   })
3. 设置代码版权：在webpack.config.js中导入const webpack = require('webpack')，在配置plugins: [new webpack.BannerPlugin("版权归小葱所有")]
4. 模块化生成html页面：首先命令安装npm install html-webpack-plugin@3.2.0 --save-dev插件，在webpack.config.js中引入const htmlWebpackPlugins = require('html-webpack-plugin')，配置plugins: [new htmlWebpackPlugins({template: 'index.html'})]，执行npm run build生成html页面。

57.压缩模块化生成的js文件：生成第三方插件npm install uglifyjs-webpack-plugin@1.1.1 --save-dev，在webpack.config.js中引入const uglifyjsWebpackPlugin = require('uglifyjs-webpack-plugin')，并使用plugins: [new uglifyjsWebpackPlugin()]，再打开就是压缩有的情况，取消注释等信息。

58.实时监听代码变动，生成地址浏览器访问打开页面：命令安装一个node的服务器npm install --save-dev webpack-dev-server@2.9.1，在webpack.config.js中配置devServer: {  
contentBase: './dist',inline: true}会实时监听代码变动，在package.json中配置"scripts": {"build": "webpack", "dev": "webpack-dev-server --open"}环境 --open会自动打开浏览器调用地址，运行npm run dev会运行项目生成地址实时监听。注：操作只是在内存中，只有运行npm run build才会生成到硬盘。

59.webpack.config.js拆分成开发配置和编译时配置，外加个公共的配置三个文件：首先安装npm install webpack-merge@4.1.5 --save-dev，再分离的开发和编译的js配置中引入：

const webpackMerge = require('webpack-merge') //引入webpack合并包  
const baseConfig = require('./base.config') //引入公共的配置  
  
module.exports = webpackMerge(baseConfig, {  
 devServer: {  
 contentBase: './dist',  
 inline: true  
 }  
})

在package.json中设置调用启动的配置路径：

"scripts": {  
 "build": "webpack --config ./build/prod.config.js",  
 "dev": "webpack-dev-server --open --config ./build/dev.config.js"  
}

60.vue cli：vue脚手架学习，自动构建webpack，减少手动构建webpack的复杂度；

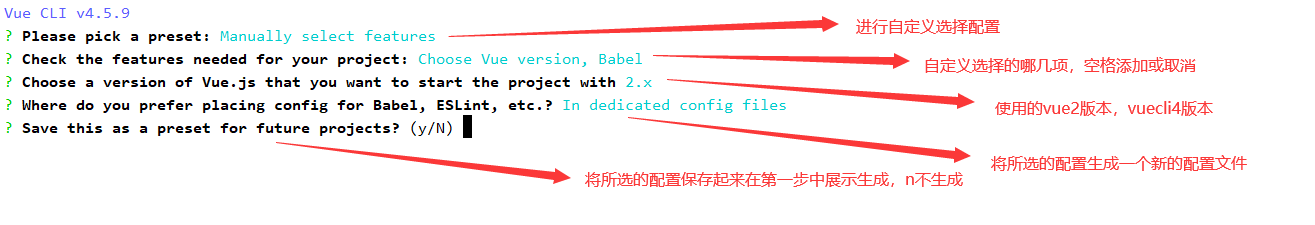
61.安装脚手架版本3或版本4：npm install -g @vue/cli全局只安装一次，在基础上拉取版本2：npm install @vue/cli-init -g。

62.idea终端使用脚手架2创建项目：vue init webpack vuecli2vue2，项目生成结构：



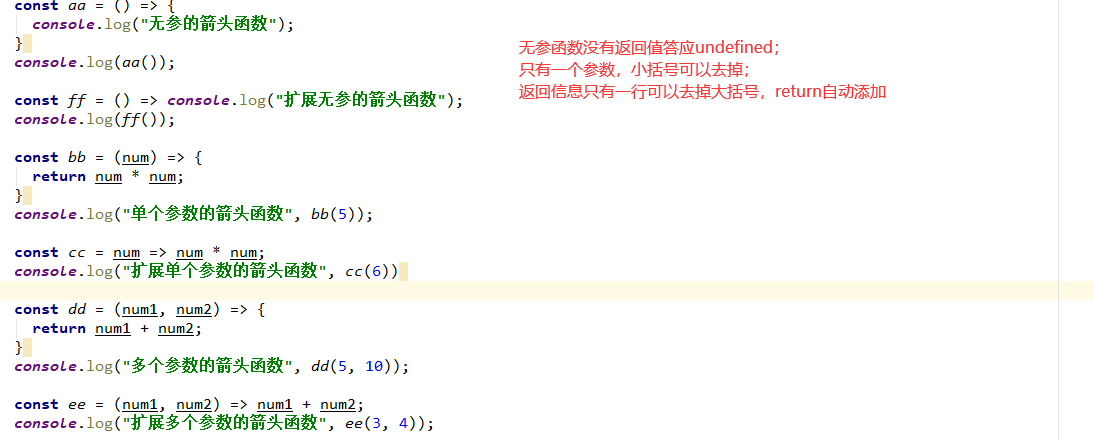
Runtime-compiler和runtime-only的区别：runtime-compiler生成的代码new Vue({el: '#app',  
components: { App }, template: '<App/>'})，内部还会将模板生成render函数；runtime-only生成的代码new Vue({el: '#app', render: function (createElement) { return createElement(App)  
}})，直接就会生成render函数，代码少，效率高。

63.idea终端使用脚手架3或4创建项目：vue create vuecli3vue2，项目生成结构：



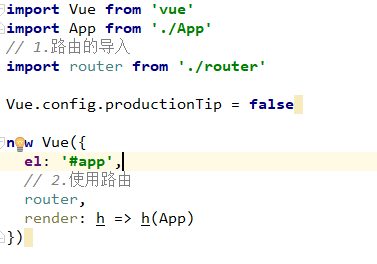
对比脚手架2隐藏掉了文件夹build，config，static配置文件，在node\_modules/@vue/ cli-server文件夹下可以看到，减少了package.json引用的一些内容，如果要新增一些配置需要在根目录下创建vue.config.js文件并导出module.exports = {}。

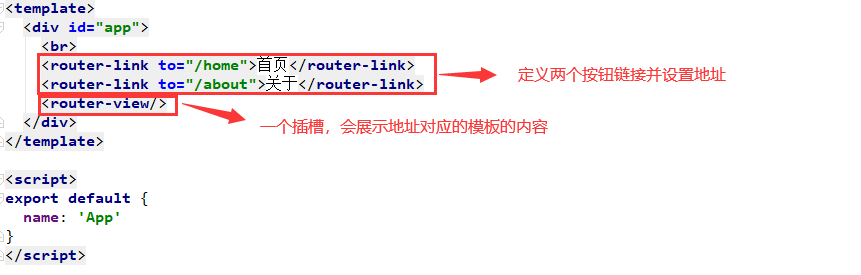
64.箭头函数实例：



65.this的使用：打印的是上级被调用的对象，如果在function(){}中打印this，结果是windows。

66.router路由的使用：首先在创建项目时，‘yes’安装路由插件；在src/router/index.js中配置并导出路由，在src/main.js中导入路由，App.vue展示效果。





67.router-link的一些属性设置：

①to:用户指定跳转的路径，可以使用:to动态挑战路径链接；

②tag:可以指定<router-link>之后渲染成什么组件；

③replace:不会留下history记录，后退键返回不能返回到上一个页面中；

④active-class:将自动给当前元素的router-link-active的class该名称；可以通过路由统一修改名称const router = new Router({linkActiveClass: 'active'});

1. 动态传递部分路径，在App.vue中动态传递地址:to="'/user/' + userName"，在router/index.js中动态配置地址const routeInfo = [ { path: '/user/:aa', component: User } ]，

跳转的模板user.vue中想要获取动态的参数使用{{$route.params.aa}};

69.模板使用query动态传递参数：在App.vue中动态跳转地址配置:to="{path: '/profile', query: {name: '张三', age: 16, height: 1.88}}"，路由地址跟随path不变，跳转模板获取参数的方式{{$route.query.name}}；

70.this.$router和this.$route的区别：this.$router获取的是路由的对象，两个常用方法this.$router.push(字符串或对象)/this.$router.replace()不能后退；this.$route获取活跃的路由地址路径；

71.懒加载使用:在/router/index.js中只需要将调用模板import Home from '@/components/Home'改为const Home = () => import("@/components/Home")方式，就能实现懒加载，懒加载在build打包时，每个template都会生成一个压缩js文件；

72.路由对模板的嵌套使用，关键字children[]:

{  
 path: '/profile',  
 component: Profile,  
 children: [  
 {  
 path: '/material',  
 component: Material  
 },  
 {  
 path: '/follow',  
 component: Follow  
 }  
 ]  
}

访问子模板的地址/profile/material

73.所有的组件都继承自Vue类的原型：使用Vue.prototype.test = '33'放入到原型当中，获取：console.log(this.test);

74.生命周期函数：created(){}组件被创建出来回调函数；mounted(){}template模板被挂载到dom上后回调的函数；updated(){}界面一旦被刷新回调的函数；destroyed() {}模板被销毁触发回调函数；

75.路由的前置钩子，在每个地址和模板关系中设置meta: {title: '关注'}，两个按钮切换前调用 router.beforeEach((to, from, next) => {  
 document.title = to.matched[0].meta.title;  
 next()；//前置钩子next()不能省略  
 })

后置钩子：router.afterEach((to, from) => {console.log("后置钩子")})

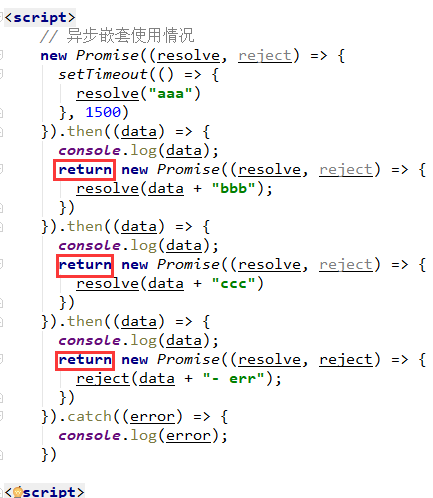
1. 保持页面状态，跳转返回还是当前页面：

<keep-alive exclude="home,user"><router-view/></keep-alive>

排除的模板可使用exclude对应模板的name属性名称，中间不能有空格；

可使用的两个方法activated() {}模板处于活跃状态触发事件和deactivated() {}模板处于不活跃状态触发事件；

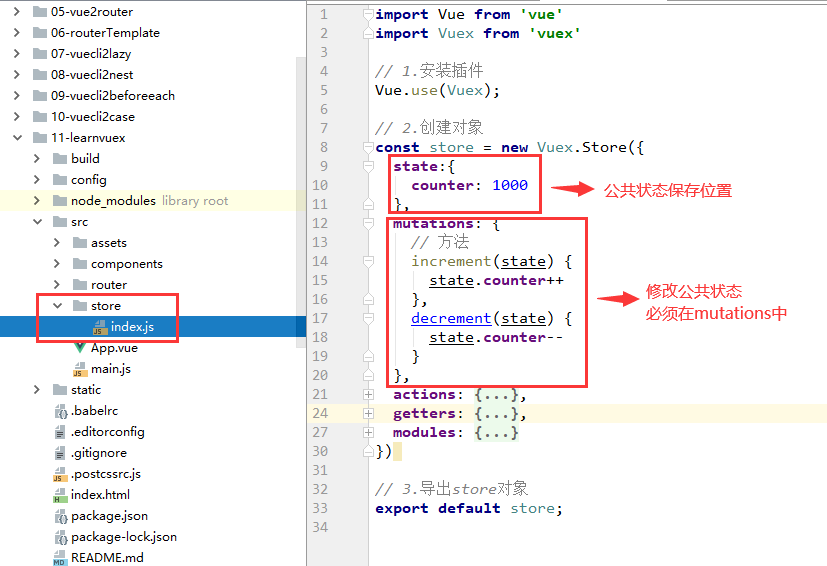
77.一般情况下有异步操作时，使用Promise对这个异步操作进行封装，简单使用：



78.vuex集中式存储管理应用的所有组件的状态，比如token，使用：

①安装vuex：npm install vuex --save

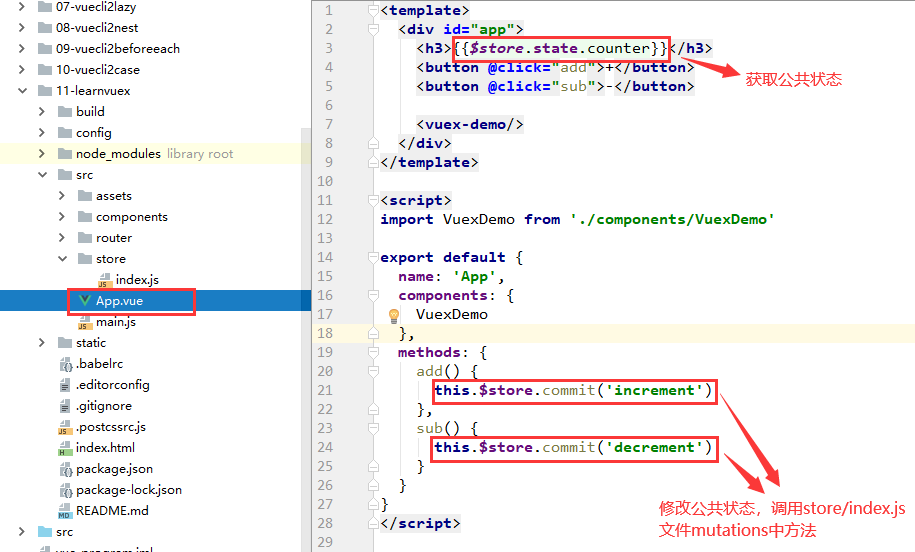
②创建文件夹及文件store/index.js，使用：



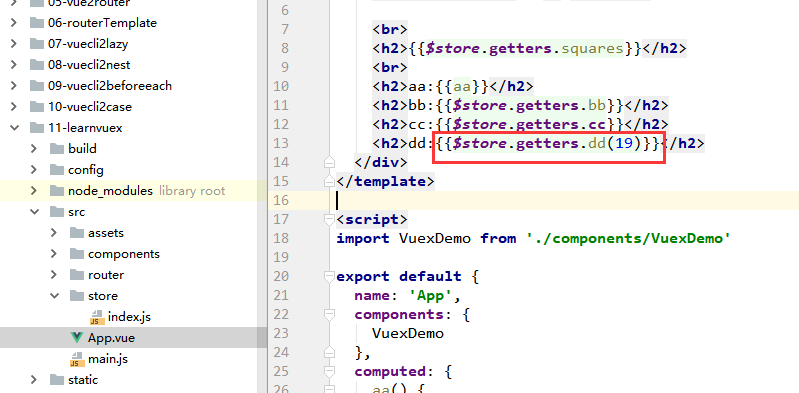
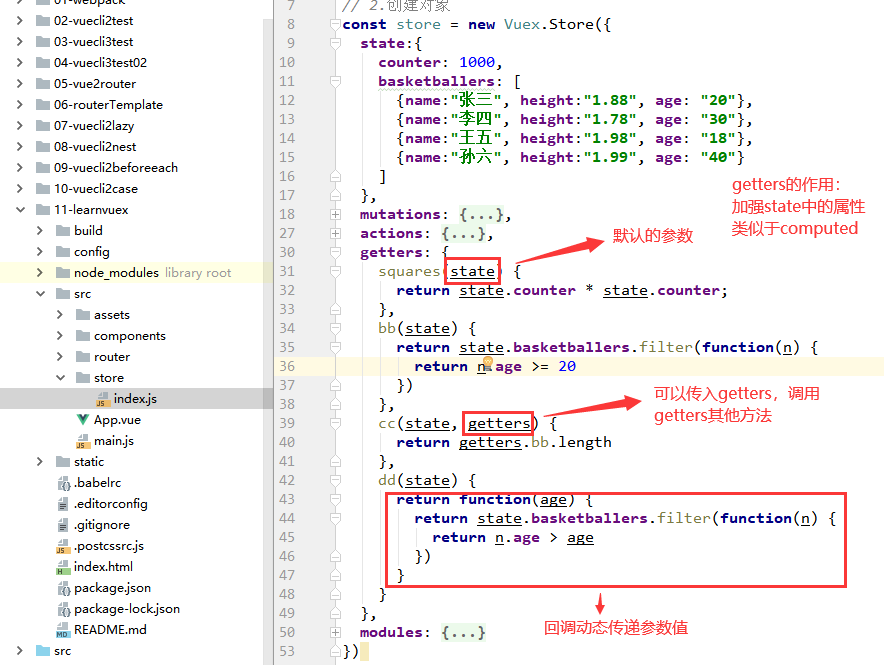
③main.js中导入store/index.js:



④模板组件调用公共状态及修改公共状态



79.vuex之getters使用及模板调用：加强state中的状态值属性并返回

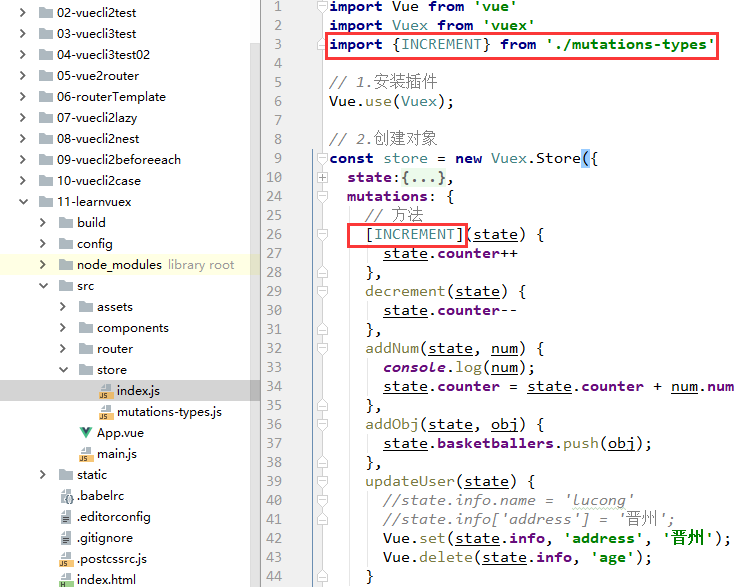
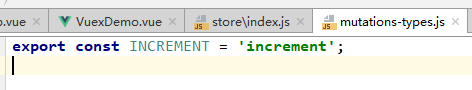


1. vuex之mutations使用及模板调用：修改state中的状态值



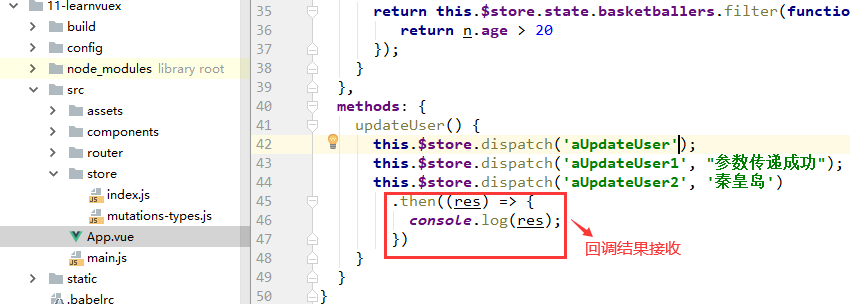
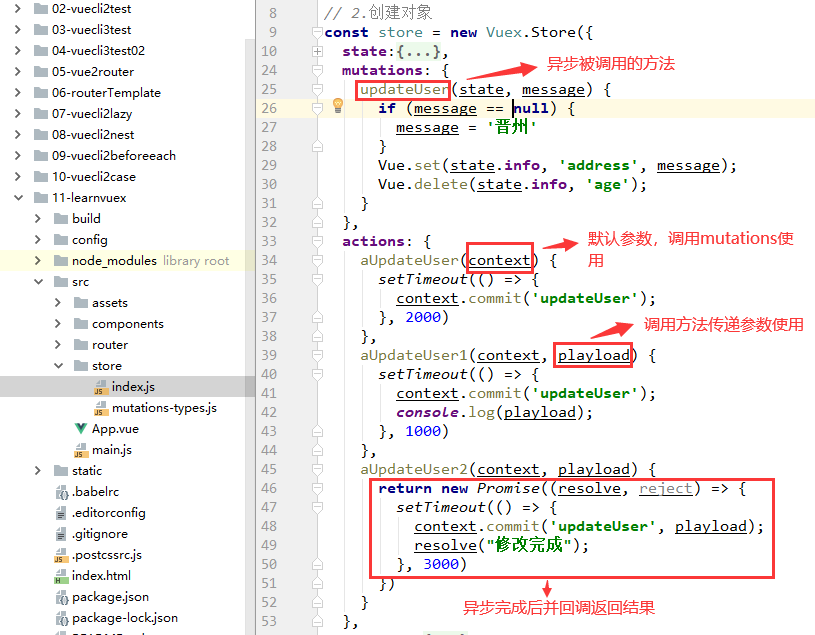
响应式新增和删除对象属性：Vue.set(state.info, ‘address’, ‘晋州’), Vue.delete(state.info, ‘age’);

将方法名抽成常量的方式：新建常量文件夹mutations-types.js

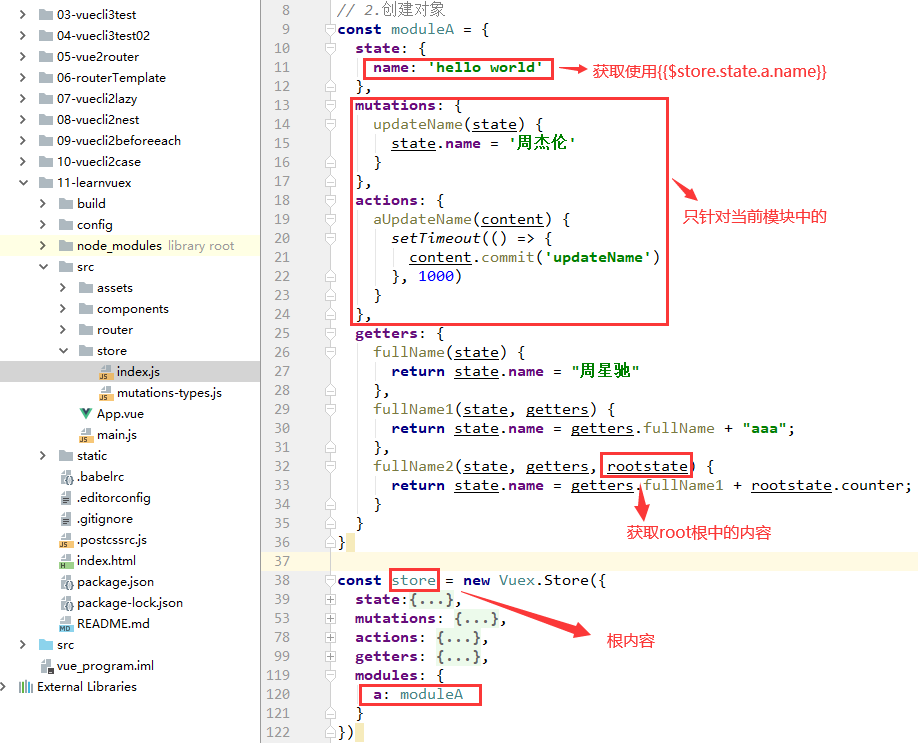


注：Vuex要求Mutation中的方法必须是同步方法；

81.vuex之actions使用及模板调用：actions代替mutations进行异步操作

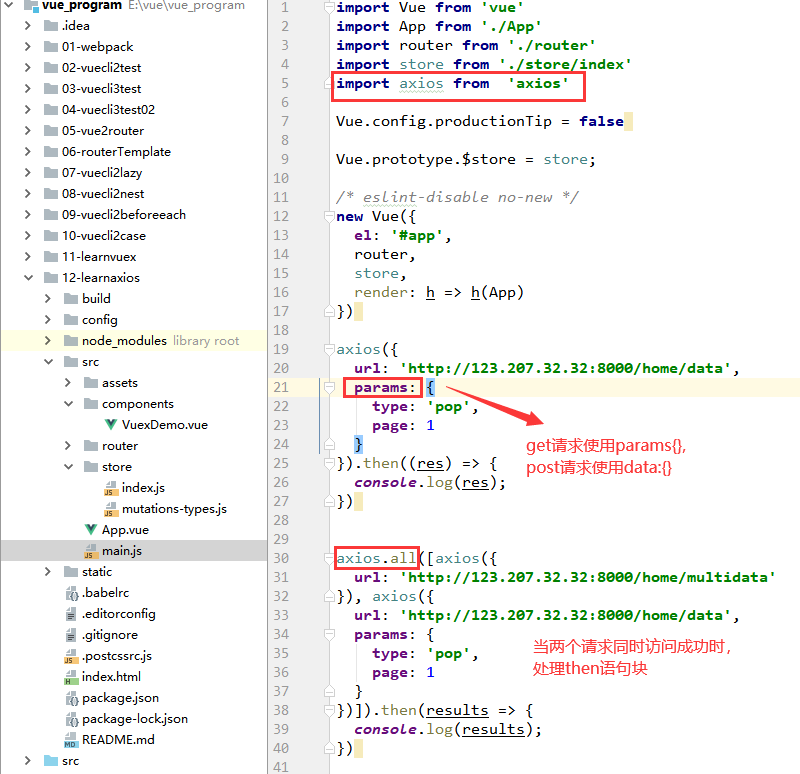


1. Vuex之modules使用及模板调用：模块抽离，减轻state中的复杂度

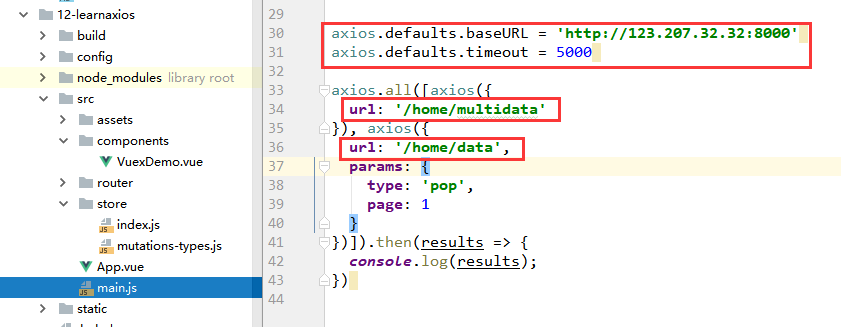


注：抽离的模块内容必须放在根内容的上方；

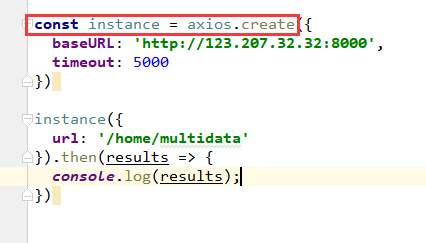
82.axios：发送http网络请求使用，安装axios：npm install axios --save，注入及简单使用：



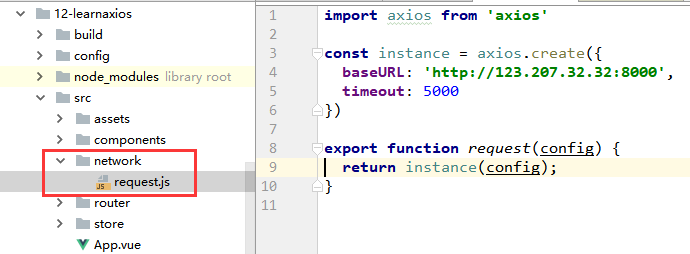
83.axios多次调用相同地址，抽离成公共地址，实现：



第二种方式创建实例保存地址，baseURL注意大小写：



84.抽离axios相关代码，main.js注入并调用：



1. 拦截器的使用：请求拦截和响应拦截

