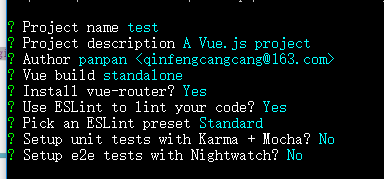
## 一 vue安装

我们使用vue-cli来安装vue项目，这样可以直接：  
 生成目录结构  
 本地开发调试  
 代码部署  
 热加载  
 单元测试  
  
安装步骤：  
 npm install -g vue-cli ----安装  
 vue -V ----查看版本  
 vue init webpack test ----安装完整vue模板，文件夹名为test  
 （vue init webpack#1.0 test ----安装webpack1.0版本vue）

安装提示如图：



进入test目录，执行依赖安装：npm install

然后启动：npm run dev

如果成功自动打开网页，即安装成功。

webstorme图形化启动：

右键点击package.json文件， show npm scripts 就可以选择脚本双击启动。

## 二 配置文件介绍

babel文件：

.babelrc modules: 代码规范风格，对应的值有：CMD,AMD

.eslintignore 代码检查时候忽略的文件

.eslintrc 使用vue-cli安装项目时，代码规则检查

eslintrc部分常用检查：

‘no-unused-vars’:0 忽略定义变量未使用检查

‘arrow-parens’:0 箭头函数无带括号，值为2必须带括号

在代码中我们添加一句话，可以跳过eslint检查

/\* eslint-disable no-new \*/

## 三 vue项目文件介绍

### 1 模板文件index.html

项目根目录的index.html是项目模板文件。webpack会自动将打包后生成的app.js插入在index.html中。

### 2 根组件App.vue

项目的根组件。

|  |
| --- |
| <**template**>  <**div id="app"**>  <**div**><**h1**>vue开始了</**h1**></**div**>  <**router-view**></**router-view**>  </**div**> </**template**>  <**script**> </**script**>  <**style**> </**style**> |

router-view标签会渲染默认的页面。

### 3 入口文件main.js

Vue.config.productionTip = false 可以关闭生产环境一些提示

|  |
| --- |
| **import** Vue **from 'vue' import** App **from './App' import** router **from './router'** Vue.config.**productionTip** = **false** */\* eslint-disable no-new \*/* **new** Vue({ //启动应用  **el**: **'#app'**,  router,  **template**: **'<App/>'**,  **components**: { App } }) |

## 四 路由vue-router

路由：根据url分配到对应的处理程序，在vue中，通过vue-router来管理url，实现url和组件的对应与切换。

### 1 安装

安装：npm install vue-router --save

引入：import VueRouter from ‘vue-router’

使用：Vue.use(VueRouter) //插件式

渲染：<router-view></router-view>

### 2 路由模式

### 2.1 hash模式

vue默认的路由模式是hash模式，即地址中包含#，如下所示：

<li><a href=”#/admin”></a></li>

<li><a href=”#/about”></a></li>

### 2.2 history模式

在router的index.js中，修改mode为history:

export default new Router({

mode:’history’,

routes:

})

### 3 HelloWorld

在router文件夹中新建index.js文件后，main.js会在配置router:’’中查找，前面书写格式为：

**new** Vue({  
 **el**: **'#app'**,  
 router,  
 **template**: **'<App/>'**,  
 **components**: { App }  
})

不写具体的路由文件，会默认进入router文件夹中查找index.js

### 4 单页面应用的链接

在单页面的导航中，点击导航，页面内容区域会自动刷新为需要的内容，而导航一般是由a链接控制，点击会执行页面跳转，这是a标签的默认行为，却不是单页面应用所期望的。

vue提供了全新的链接标签router-link来解决上述问题。

#### 4.1 router-link基本使用

在APP.vue中：

|  |
| --- |
| <**template**>  <**div id="app"**>  <**h1**>欢迎来到vue</**h1**>  <**div class="nav-box"**>  <**ul class="nav"**>  *<!-- 基本绑定形式 -->*  <**router-link to="/index"**>index</**router-link**>  *<!-- 动态绑定形式 -->*  <**li**><**router-link :to="**{**path**:**'/document'**}**"**>document</**router-link**></**li**>  *<!-- 对象绑定形式 -->*  <**li**><**router-link :to="aboutBind"**>about</**router-link**></**li**>  </**ul**>  </**div**>  <**router-view**></**router-view**>  </**div**> </**template**>  <**script**>  **export default** {  **name**: **'app'**,  data () {  **return** {  **aboutBind**: **'/about'** }  }  } </**script**> |

#### 4.2 router-link渲染

router-link还有更强大的功能，他可以不再渲染为a标签，可以根据用户需求设置：

<**li**><**router-link to="/mine" tag=’div’**>mine</**router-link**></**li**>

#### 4.3 router-link标签包含

很多导航中还会有图片，传统做法一般是使用span、i等包含新的内容

<li>

<i><img></i>

<span><a></a></span>

</li>

router-link也可以包含额外的标签：

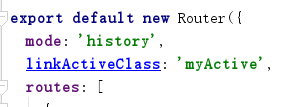
|  |
| --- |
| <**router-link to="/mine" tag="li"**>  <**i**></**i**>  <**span**>mine</**span**> </**router-link**> |

#### 4.4 router-link-active

router-link中默认有一个class类名：router-link-active，

意义为：被选中的标签将会触发该class样式。

当然，这个类名太长，我们可以在全局路由index.js中进行别名配置：



单独配置

|  |
| --- |
| <**router-link :to="aboutBind" active-class="myAbout"**>about</**router-link**> |

#### 4.5 router-link 事件控制

我们可以让router-link以事件形式触发控制css：

|  |
| --- |
| <**router-link :to="aboutBind" active-class="myabout" event="mouseover"**>about</**router-link**> |

如上所示：鼠标移入后触发 myabout的css效果。

### 5 全局样式

<**router-view class=’myclass’**></**router-view**>

如果在App.uve中设置了这样的样式，那么会设置全局的样式！

### 6 404与重定向

#### 1 404

404 在所有路由配置完毕后加入：

|  |
| --- |
| {  **path**: **'\*'**,  **component**: other } |

#### 2 重定向

|  |
| --- |
| {  **path**: **'/'**,  **redirect**: **'/index'**  } |

对象形式：redirect的值修改为{path: /index}，或者如下所示

|  |
| --- |
| redirect:  {  **path**: **'/index'**,  **name: indexRouter**  **component**: myIndex },  {  **path**: **'/'**,  **redirect**: {name: **'indexRouter’**} }, |

name就是路由的名字。

#### 3 动态设置重定向

redirect: (to) => {

return ‘/mine’

}

### 7 路由匹配

#### 1 alias

|  |
| --- |
| {  path:’/mine’,  component:mine,  alias:’/index’  } |

alias的意思是：当用户访问了/index，将会自动匹配到/mine这个路由

#### 2 exact精确匹配

路径的精确匹配属性：exact

<router-link to=’/’ exact tag=’li’></router-lin>

#### 3 多层路由与多重组件

多层路由：比如在路由/doc 下还有子路由 /doc/doc1 /doc/doc2等等

|  |
| --- |
| <**template**>  <**div**>  Docs  <**hr**>  <**ul class="nav"**>  <**router-link to="/doc/doc1" tag="li"**>  <**a**>doc1</**a**>  </**router-link**>  <**router-link to="/doc/doc2" tag="li"**>  <**a**>doc2</**a**>  </**router-link**>  <**router-view**></**router-view**>  </**ul**>  </**div**> </**template**> |

注意：

<a>doc1</a> 没有设置href，这里会自动载入父标签router-link的to地址。

红色部分是进入/doc路由后默认会将渲染的组件在此渲染

路由设计：

|  |
| --- |
| {  **path**: **'/doc'**,  **component**: mydoc,  **children**: [  {  **path**: **'doc1'**,  **component**: doc1  },  {  **path**: **'doc2'**,  **component**: doc2  }  ] }, |

注意：

我们也可以设置默认子路由，即打开页面即展示哪个子路由。

|  |
| --- |
| <**ul class="nav"**>  <**router-link to="/doc" exact tag="li"**>  <**a**>doc1</**a**>  </**router-link**>  <**router-link to="/doc/doc2" tag="li"**>  <**a**>doc2</**a**>  </**router-link**>  <**router-view**></**router-view**> </**ul**> |

这里默认展示第一个子路由，必须严格匹配，即添加 exact，且无需再写子路由地址

如果有了默认子路由，这里父组件的name属性应该设置在默认子路由上，如下所示：

|  |
| --- |
| {  **path**: **'/doc'**,  **component**: mydoc,  **children**: [  {  **path**: **''**,  **component**: doc1  },  {  **path**: **'doc2'**,  **component**: doc2  }  ] }, |

#### 4 动态绑定隐藏父路由

访问Doc组件下的doc1，doc2的路由理应是

/doc/doc1 和 /doc/doc2

在一些特殊环境下，我们希望路由这样设计：

/doc1 和 /doc2

直接就可以访问到子组件

解决方案：子路由的地址相对于跟根路径即可

为了保证访问正确，还需要设置 <router-link : to=”{name: ‘mydoc’}”></router-link >

name就是组件的name属性值。

|  |
| --- |
| {  **path**: **'/doc'**,  **component**: MineAbout,  **children**: [  {  **path**: **''**,  **component**: doc1  },  {  **path**: **'/doc2'**,  **component**: doc2  },  {  **path**: **'/doc3'**,  **component**: doc3  }  ] }, |

#### 5 命名视图

同级路由直接展示多个视图组件，而非父子嵌套的展示。

首先我们需要在主组件处额外添加显示信息，并取名

|  |
| --- |
| <**template**>  <**div id="app"**>  <**h1**>欢迎来到vue</**h1**>  <**div class="nav-box"**>  <**ul class="nav"**>  <**li**><**router-link to="/index"**>index</**router-link**></**li**>  <**li**><**router-link :to="**{**path**:**'/doc'**}**"**>document</**router-link**></**li**>  <**li**><**router-link :to="aboutBind"**>about</**router-link**></**li**>  </**ul**>  </**div**>  <**router-view**></**router-view**>  <**router-view name="myWork"**></**router-view**>  <**router-view name="myFamily"**></**router-view**>  </**div**> </**template**> |

然后配置路由:

|  |
| --- |
| {  **path**: **'/about'**,  **components**: {  **default**: myabout,  **myWork**: myWork,  **myFamily**: myFamily  } }, |

#### 6 浏览器前进后退

当我们在浏览网页时，回退到上一次的网页然后又再次回到现在的网页，会发现现在的网页默认从上往下显示，而不是回到刚才浏览的位置。我们可以更改滚动条的位置来解决：

在路由中添加如下配置：

|  |
| --- |
| scrollBehavior(to,from,savePosition){  **if**(savePosition){  **return** savePosition;  }**else** {  **return** {**x**: 0,**y**: 0}  } }, |

当然也有使用hash锚点的时候，比如路由地址是：/doc#abc

|  |
| --- |
| scrollBehavior(to,from,savePosition){  **if**(to.**hash**){  **selector**: to.**hash** } }, |

### 8动态路径

#### 1 动态路径匹配

当匹配到所有的路由，全部映射到同一个组件，这时候要使用动态路径

例如：/user/:userID

userID就是动态路径的参数，依据参数不同，给同一个组件提供不同的数据模型。

|  |
| --- |
| <**template**>  <**div**>  <**h1**>Users</**h1**>  <**router-link style="padding: 0 20px" :to="'/user/'**+ ***item***.**tip** + **'/'** + ***item***.**id" key="index" v-for="*item***,index in ***userList*"**>{{***item***.username}}</**router-link**>  </**div**> </**template**>  <**script**>  **let *userData*** = [  {**id**: 1, **username**: **'lisi1'**, **sex**: **'男'**, **tip**: **'vip'**},  {**id**: 2, **username**: **'lisi2'**, **sex**: **'男'**, **tip**: **'vip'**},  {**id**: 3, **username**: **'lisi3'**, **sex**: **'女'**, **tip**: **'common'**}  ]   **export default** {  data () {  **return** {  **userList**: ***userData*** }  }  }  </**script**> |

路由配置：

|  |
| --- |
| {  **path**: **'/user/:userTip?/:userID?'**,  **component**: myUsers }, |

userID? 中 ? 的作用： 不带问号，则不能匹配到/user，而带了问号，则参数userID可以有，可以没有，没有参数时，则会匹配到/user。

那么我们如何通过路由中的参数，获取具体对象的数据信息呢？

#### 2组件注入

我们如何从路由中获取参数？

$router 是router实例对象，$router是当前激活的路由信息信息对象，每个组件实例都会有。那么根据上述（知识点8）中的路由配置我们可以得到路由参数：

|  |
| --- |
| **export default** {  data () {  **return** {  **userList**: ***userData*** }  },  created () {  ***console***.log(**this**.$route)  } } |

在上述案例的myUser.vue中，添加如下钩子函数，便可以获取到路由中的参数，并取到数据模型。

|  |
| --- |
| created () {  ***console***.log(**this**.$route.params.userID)  **let** id = **this**.$route.params.userID  **if** (**this**.$route.params.userID) {  **for** (**var** i = 0; i < ***userData***.length; i++) {  **if** (***userData***[i].**id**.toString() === id) {  **this**.**userInfo** = ***userData***[i]  }  }  } **else** {  **this**.**userInfo** = {}  } } |

显示效果如图：



但是新的问题来了，我们点击 lisi1 lisi2 lisi3 路由变了，但是渲染内容没有变化，这是因为访问/users时，组件已经生成一次了，created钩子函数只执行一次，不会再次执行。我们要监视路由的变化，每次路由变化了created内部的代码都要执行，那么我们需要在creater前添加如下代码：

|  |
| --- |
| **watch**: {  $route () {  **let** id = **this**.$route.params.userID  **if** (**this**.$route.params.userID) {  **for** (**var** i = 0; i < ***userData***.length; i++) {  **if** (***userData***[i].**id**.toString() === id) {  **this**.**userInfo** = ***userData***[i]  }  }  } **else** {  **this**.**userInfo** = {}  }  } }, |

注意：

之前在案例myDoc myAbout中点击页面中的按钮可以渲染是因为点击按钮渲染的是新组件，而本案例中，点击按钮不再是渲染组件，而是现实当前组件中的html代码。

由于存在同样的代码逻辑，可以进行代码优化，下面是完整的myUsers.vue的代码：

|  |
| --- |
| <**template**>  <**div**>  <**h1**>Users</**h1**>  <**router-link style="padding: 0 20px" :to="'/user/'**+ ***item***.**tip** + **'/'** + ***item***.**id" key="index" v-for="*item***,index in ***userList*"**>{{***item***.username}}</**router-link**>  <**div**>  姓名：{{userInfo.username}},性别：{{userInfo.**sex**}}  </**div**>  </**div**> </**template**>  <**script**>  **let *userData*** = [  {**id**: 1, **username**: **'lisi1'**, **sex**: **'男'**, **tip**: **'vip'**},  {**id**: 2, **username**: **'lisi2'**, **sex**: **'男'**, **tip**: **'vip'**},  {**id**: 3, **username**: **'lisi3'**, **sex**: **'女'**, **tip**: **'common'**}  ]   **export default** {  data () {  **return** {  **userList**: ***userData***,  **userInfo**: {}  }  },  **watch**: {  $route () {  **this**.getData()  }  },  created () {  ***console***.log(**this**.$route.params.userID)  **this**.getData()  },  **methods**: {  getData () {  **let** id = **this**.$route.params.userID  **if** (**this**.$route.params.userID) {  **for** (**var** i = 0; i < ***userData***.length; i++) {  **if** (***userData***[i].**id**.toString() === id) {  **this**.**userInfo** = ***userData***[i]  }  }  } **else** {  **this**.**userInfo** = {}  }  }  }  } </**script**> |

#### 3 query字符串传参

两种写法

|  |
| --- |
| <**div**>  <**router-link exact to="?info=follow"**>查看他的关注</**router-link**>  <**router-link exact :to="**{**path**: **''**,**query**: {**info**: **'share'**,**a**: **'3'**}}**"**>查看分享</**router-link**>  <**div**>  发送的参数是：{{$route.**query**}}  </**div**> </**div**> |

这里没有书写具体的路由，只写了参数，会自动在当前路由后添加以上参数。

当然还有优化的地方，当我们在 查看他的关注 查看他的分享 这里切换的时候，自然是没有问题的，但是如果我们切换他们的上级导航如lisi1，lisi2，那么就不会再显示参数，而底部发送的参数是：{{$route.**query**}}的结果就是空，造成页面很难看。

我们可以对导航上的路由进行改造：

|  |
| --- |
| <**router-link :to="**{**path**: **'/users/'**+***item***.**tip**+**'/'**+***item***.**id**,**query**: {**info**: **'follow'**}}**" key="index" v-for="*item***,index in ***userList*"**>  {{***item***.**userName**}} </**router-link**> |

## 五 导航过度动画

### 1 基本用法

vue提供了transition封装组件，来添加过渡动画，通过添加删除css类名或者钩子函数来控制。

常见CSS类名：

v-enter 进入过渡的开始状态

v-enter-active 进入活动状态

v-enter-to 既然怒的结束状态

v-leave

v-leave-active

v-leave-to

基本用法：

|  |
| --- |
| template部分：  <**transition**>  <**router-view**></**router-view**> </**transition**>  style部分：  .**v-enter**{  **opacity**: 0; } .**v-enter-to**{  **opacity**: 1; } .**v-enter-active**{  **transition**: 1**s**; } |

当然还有很多其他的动画，非v-开头，使用方式如下：

|  |
| --- |
| <**transition name="left"**>  <**router-view**></**router-view**> </**transition**>  .**left-enter**{  **transform**: **translateX**(100%); } .**left-enter-to**{ //默认值可以不写  **transform**: **translateX**(0); } .**left-enter-active**{  **transition**: 1**s**; } |

### 2 过渡模式

当我们在切换导航时，可能切换动画还未完成，又点击了别的导航，这样会造成不同切换之间的动画互相重叠的问题。这时候需要使用过渡模式。

过渡模式：

in-out：新元素先进行过渡，完成之后当前元素过渡离开

out-in：当前元素先进行过渡，完成之后新元素过渡进入

|  |
| --- |
| <**transition mod="in-out"**>  <**router-view**></**router-view**> </**transition**> |

## 六 路由元信息

在路由配置中meta可以配置一些数据，用在路由信息对象中，访问meta中的数据方式：$route.meta

|  |
| --- |
| {  **path**: **'/index'**,  **component**: myindex,  **meta**: {  **subIndex**: 1  } }, |

使用：

|  |
| --- |
| **watch**: {  $route (to, **from**) {  ***console***.log(to.meta.**subIndex**)  ***console***.log(**from**.meta.**subIndex**)  } }, |

其中，to和from分别进入、离开的路由对象。

## 七 编程式导航

借助于router的实例方法，以代码方式实现导航的切换。

back：回退一步

forward：前进一步

go：制定前进回退步数

push：导航到不同url，向history栈添加一个新的记录

replace：导航到不同url，替换history中的当前记录

|  |
| --- |
| <**input type="button" value="后退一步" @click="**backHandle**"**/> <**input type="button" value="后退两步" @click="**DBbackHandle**"**/> <**input type="button" value="push到首页" @click="**pushHandle**"**/> |

|  |
| --- |
| **methods**: {  backHandle () {  **this**.$router.back()  },  DBbackHandle () {  **this**.$router.go(-2)  },  pushHandle () { *// this.$router.push('/index')* **this**.$router.push({**path**: **'/index'**})  }  } |

不能让给出的回退、前进数值超过历史记录值，因为是无效的。

## 八 钩子函数

### 1 基本使用

|  |
| --- |
| Vue.use(Router)  **let** router = **new** Router({  ......... }) router.beforeEach((to, from, next) => {  **if** (to.**meta**.login === **true**) {  next(**'./login'**)  } **else** {  next()  } }) **export default** router |

如果不写next() 或者next的参数写为false，将会不渲染组件内容。

常用的使用案例：当一个页面登陆后才会让用户访问，这时候用户访问路由，我们可以使用next()进行重定向到登录页。

其他案例，比如我们在进入一个导航后，网页的title需要改变为当前页面需要的标题，那么我们可以使用afterEach这个导航钩子函数。（注意：title不能直接获得，需要完整的写：window.document.title）

### 2 组件级钩子函数

以上的钩子函数写在了路由外部，作用在了全局，如果在单个路由中的书写钩子函数，这个钩子函数只作用于该留有，当然也有组件级的钩子函数，书写在具体的组件script中。

|  |
| --- |
| <**script**>  **export default** {  data () {  **return** {  **test**: **'改变前'** }  },  beforeCreate () {  ***console***.log(**'beforeCreate'**)  },  beforeRouteEnter (to, **from**, next) {  next((vm) => {  vm.**test** = **'改变了'** })  }  } </**script**> |

执行顺序是：全局》路由》组件

### 3 常见钩子函数

router实例：

beforeEach afterEach

单个路由：

beforeEnter

组件级钩子：

beforeRouteEnter beforeRouteUpdate beforeRouteLeave

钩子函数的参数：

to：要进入的目标路由对象，到哪里去

frome：正要离开的导航路由对象，从哪里来

next：决定是否跳转或者取消导航

## 九 Vuex

### 1 父子组件传值

案例：在项目中存在一个如图所示的选项卡组件



点击输入框后显示选项卡列表，点击对应的选项卡，我们将选项卡内容填充到输入框中。

这里使用一个父组件select包含两个子组件：输入框select-input，选项卡list

父组件：

|  |
| --- |
| <**template**>  <**section class="warp"**>  <**div class="searchIpt clearFix"**>  <**select-input :is-show.sync="listShow"**></**select-input**>  <**list v-show="listShow" :data="*listData*"**></**list**>  </**div**>  </**section**> </**template**>  <**script**>  **import** selectInput **from '@/components/selectInput' import** list **from '@/components/list'  let *listData*** = [  {  **title**: **'js'** },  {  **title**: **'html'** },  {  **title**: **'css'** } ]  **export default** {  data () {  **return** {  **listShow**: **false**,  **listData**: ***listData*** }  },  **components**: {  selectInput,  list  } } </**script**> |

子组件select-input

|  |
| --- |
| <**template**>  <**div class="clearFix"**>  <**input  class="keyWord"  @click="**showListHandle**"** />  <**input type="button" value="GO"**>  <**span**></**span**>  </**div**> </**template**>  <**script**>  **export default** {  **props**: [**'isShow'**],  **computed**: {  initShow(){  **return this**.isShow  }  },  **methods**:{  showListHandle(){  **this**.$emit(**"update:isShow"**,!**this**.initShow)  }  } } </**script**> |

父子传值：

在父组件的<**select-input :is-show.sync="listShow"**></**select-input**>中，我们定义了一个属性，:is-show.sync控制list组件的展示与否，sync 用来同步展示组件父子组件数据变动。

在子组件中，我们使用 **this**.$emit(**"update:isShow"**,!**this**.initShow)来将计算的结果值传递给父组件。

子组件list

|  |
| --- |
| <**template**>  <**ul class="list"**>  <**li v-for="*item*** in ***data*"**>{{***item***.title}}</**li**>  </**ul**> </**template**>  <**script**> **export default** {  **props**: [**'data'**] } </**script**> |

在父组件定义 <**list v-show="listShow" :data="*listData*"**></**list**>

在子组件中，使用 **props**: [**'data'**] 来接收该属性的值，并做数据循环处理

### 2 兄弟组件传值

在上述案例中，如果我们点击list列表，需要将列表中的文字显示在输入框中，即子组件list的内容传递给兄弟组件select-input中，该如何处理：

解决方案：在子组件list中点击选项时，给父组件定义一个状态来改变该状态，由父组件的状态控制另外一个子组件select-input的内容。

第一步：

给父组件定义一个title状态 **title**: **'',**

给子组件list添加自定义事件 **@changeTitle="**changeTitle**"**

第二步：list传递值

<**li v-for="*item*** in ***data*" @click="**showTitle(***item***.title)**"**>{{***item***.title}}</**li**>

**methods**: {  
 showTitle(title){  
 **this**.$emit(**'changeTitle'**,title)  
 }  
}

第三步：父组件控制显示title

<**select-input :title="**title**" :is-show.sync="listShow"**></**select-input**>

**methods**: {  
 changeTitle(title){  
 **this**.**title** = title  
 }  
},

完整代码

|  |
| --- |
| 父组件：  <**template**>  <**section class="warp"**>  <**div class="searchIpt clearFix"**>  <**select-input :title="**title**" :is-show.sync="listShow"**></**select-input**>  <**list v-show="listShow" :data="*listData*" @changeTitle="**changeTitle**"**></**list**>  </**div**>  </**section**> </**template**>  <**script**>  **import** selectInput **from '@/components/selectInput' import** list **from '@/components/list'  let *listData*** = [  {  **title**: **'js'** },  {  **title**: **'html'** },  {  **title**: **'css'** } ]  **export default** {  data () {  **return** {  **listShow**: **false**,  **listData**: ***listData***,  **title**: **''** }  },  **methods**: {  changeTitle(title){  **this**.**title** = title  }  },  **components**: {  selectInput,  list  } } </**script**>  子组件select-input  <**template**>  <**div class="clearFix"**>  <**input  class="keyWord"  @click="**showListHandle**"  :value="**title**"** />  <**input type="button" value="GO"**>  <**span**></**span**>  </**div**> </**template**>  <**script**>  **export default** {  **props**: [**'isShow'**,**'title'**],  **computed**: {  initShow(){  **return this**.isShow  }  },  **methods**:{  showListHandle(){  **this**.$emit(**"update:isShow"**,!**this**.initShow)  }  } } </**script**>  子组件list  <**template**>  <**ul class="list"**>  <**li v-for="*item*** in ***data*" @click="**showTitle(***item***.title)**"**>{{***item***.title}}</**li**>  </**ul**> </**template**>  <**script**> **export default** {  **props**: [**'data'**],  **methods**: {  showTitle(title){  **this**.$emit(**'changeTitle'**,title)  }  } } </**script**> |

### 3 Vuex简单使用

在前面的案例中，只嵌套了两层，如果嵌套了多层，那么值得传递将会是一层一层来传，很麻烦，这时候需要使用的是Vuex。

Vuex是专为Vue.js应用程序开发的状态管理模式。

Vuex采用集中式存储管理应用的所有组件状态，以相应的规则保证状态以一种可预测的方式发生变化。

状态包含：

组件内部状态：只在组件中使用的状态-data字段

应用级状态：多个组件公用的状态

使用Vuex的场景：

多个视图依赖于同一状态；

来自不同视图的行为需要变更同一个状态。

简单使用

安装模块：npm install vuex --save

作为插件使用：Vue.use(Vuex)

定义容器：new Vuex.Store()

注入根实例：{store}

代码实例：

在src下定义文件夹store，创建默认状态控制文件index.js，代码如下：

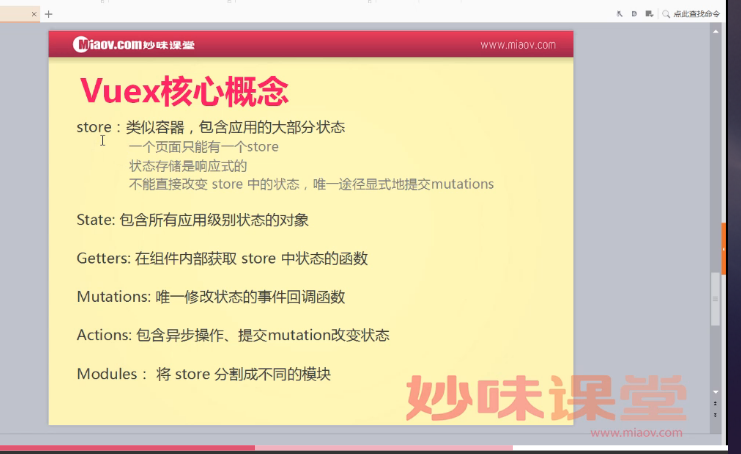
|  |
| --- |
| 组件代码：  <**template**>  <**div**>  <**h2**>简易加法计算器</**h2**>  <**div**>  <**input type="button" value="-" @click="**deHandle**"** />  <**span**>{{num}}</**span**>  <**input type="button" value="+" @click="**addHandle**"** />  <**p**></**p**>  </**div**>  </**div**> </**template**>  <**script**>    **export default** {  **computed**: {  num(){  **return this**.$store.**state**.count  }  },  **methods**: {  addHandle(){  *//点击按钮，提交mutation add* **this**.$store.**commit**(**'add'**)  },  deHandle(){  **this**.$store.**commit**(**'de'**,{  **n**: 10,  **time**: 1000  })  *//也可以以对象的形式进行传递 // this.$store.commit({ // type: 'de', // n: 10 // })* }  }  } </**script**>  vuex代码：  **import** Vue **from 'vue' import** Vuex **from 'vuex'** Vue.*use*(Vuex)  *//定义一个容器* **let** store = **new** Vuex.Store({  **state**: {  **count**: 100  },  **mutations**: {  add(state){  state.**count** += 1  },  de(state,params){  state.**count** -= params.**n** }  }  }) **export default** store |

流程解释：

在组件中，通过computed内的数据来展示当前count的值

在组件中，通过addHandle等方法来提交mutation给store中的index.js

在store中，相应的mudations进行修改。

、

### 4 异步

mutations的提交都是同步，但是实际项目中我们需要提交后做别的事情，得到了返回结果再执行任务，这时候需要异步提交，vuex提供了action来完成这个功能、

|  |
| --- |
| deHandle(){  **this**.$store.**dispatch**(**'deAction'**) } |

|  |
| --- |
| **actions**: {  deAction(context){  setTimeout(()=>{  *//改变状态，提交mutations* context.**commit**(**'de'**,{**n**: 5})  },1000)  } } |

### 5 getters

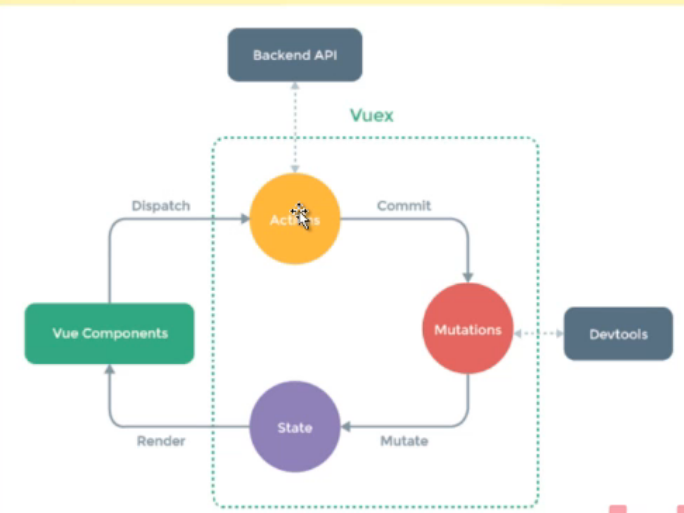
我们在得到值的时候可能需要对值进行处理，这时候可以使用getters

在store中：

|  |
| --- |
| **getters**: {  filterCount(state){  **return** state.**count** >=120 ? 120 : state.**count** } }, |

在组件中获取count

|  |
| --- |
| **computed**: {  num(){  **return this**.$store.**state**.count  },  getNum(){  **return this**.$store.**getters**.filterCount  } }, |



### 6 辅助函数

利用辅助函数可以简化vuex的书写。

三种写法：

|  |
| --- |
| **import** {mapState,mapGetters} **from 'vuex'  export default** {  **computed**: mapState({  *// num: state => state.count  // num: 'count'* num(state){  **return** state.**count** + 100  }  }), |

第四种写法： 此时需要将num替换成count

|  |
| --- |
| **computed**: mapState([**'count'**]) , |

第五种写法：

|  |
| --- |
| **computed**: {  ...mapState([**'count'**]) }, |

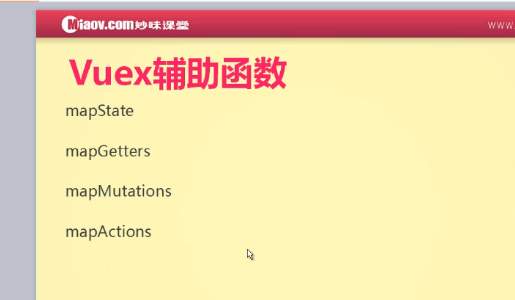
同理，引入mapGetters，mapActions后可以简写getters、action的数据获取，语法一致

### 7 划分模块module

我们无需把所有模块的状态都定义在vuex的根实例中

|  |
| --- |
| **let** inCrementModule = {  **state**: {  **count**: 100  },  **mutations**: {  add(state){  state.**count** += 1  },  de(state,params){  state.**count** -= params.**n** }  } } **let** store = **new** Vuex.Store({  **modules**: {  inCrementModule  } }) **export default** store |

获取子模块状态：this.$store.inCrementModule.count



## 十 Axios

### 1 Axios简介





### 2 简单使用

安装模块 axios与vueaxios后，在App.vue中使用axios：

|  |
| --- |
| <**script**> **import** axios **from 'axios' export default** {  **name**: **'app'**,  created(){  axios({  **method**: **'get'**,  **url**: **'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'** })  .then((res)=>{  ***console***.log(res.**data**)  })  .catch((err)=>{  ***console***.log(err)  })  } } </**script**> |

注意：axios也提供了直接使用axios.get(url)这样的方式发送不同类型的请求。

更简洁的方式：在main.js中全局使用：

|  |
| --- |
| **import** Axios **from 'axios' import** VueAxios **from 'vue-axios'** Vue.use(VueAxios,Axios) |

在任意的组件中都可以使用 this.$http.get(url)来发送请求。

### 3 参数传递

get请求参数

|  |
| --- |
| axios.get(**'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'**,{  **param**: {  **abc**: **'123'** } }) |

post请求参数

|  |
| --- |
| axios.post(**'http://easy-mock.com/mock/59b7f0dee0dc663341a77864/test-axios/test1'**,{  **abc**: **'123'** }) |

### 4 自定义请求





其他配置：  


需要return，没有return会是undefined

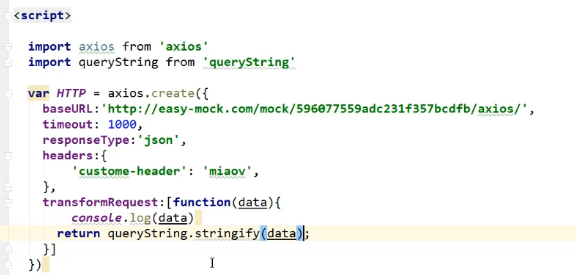


transform可以用来转换发送的数据格式

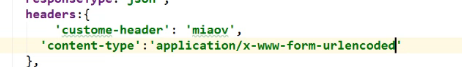
比如我们在请求时，请求数据是json形式：



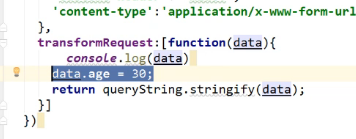
我们想使用字符串 miaov=ketang&username=leo这种格式，可以在transform中修改：



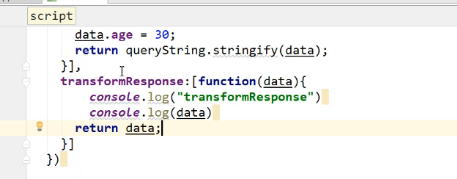
我们还需要设置头信息，让post请求支持这种字符串格式：



发送之前还可以对数据进行额外处理，如改变值：



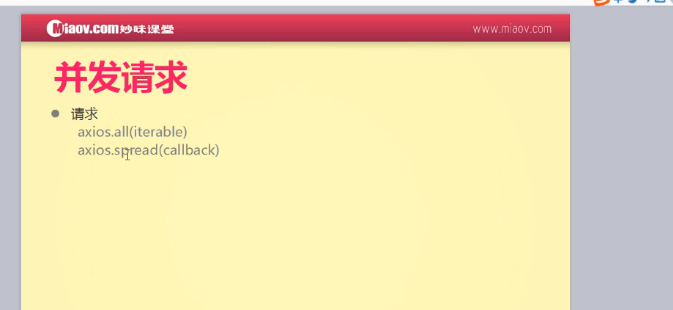
我们还可以请求结果进行加工：



### 5 取消请求配置



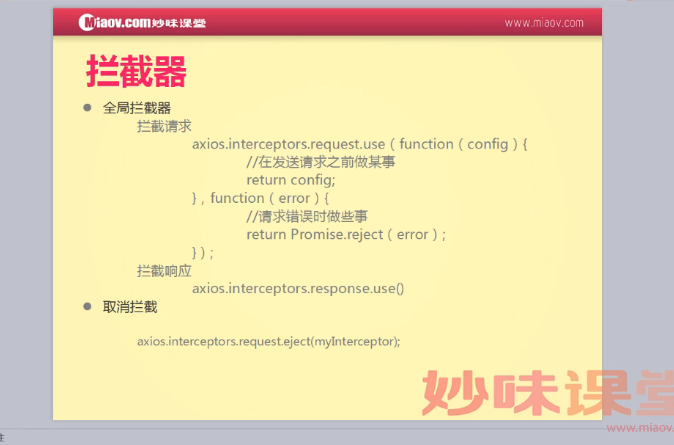
### 6并发请求



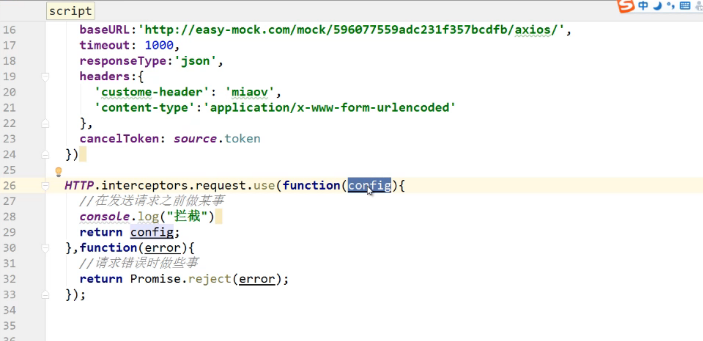
|  |
| --- |
| <script>  import axios from 'axios'  import queryString from 'queryString'  var HTTP = axios.create({  baseURL:'http://easy-mock.com/mock/596077559adc231f357bcdfb/axios/',  timeout: 1000,  responseType:'json',  headers:{  'custome-header': 'miaov',  'content-type':'application/x-www-form-urlencoded'  }  })  HTTP.interceptors.request.use(function(config){  //在发送请求之前做某事  console.log("拦截")  console.log(config)  return config;  },function(error){  //请求错误时做些事  return Promise.reject(error);  });  HTTP.interceptors.response.use(function(data){  console.log("response")  console.log(data)  return data;  })    export default {  name: 'hello',  created(){  function http1(){  return HTTP.get("test-axios")  }  function http2(){  return HTTP.post("test-post-axios")  }  axios.all([http1(),http2()]).then(axios.spread((res1,res2)=>{  console.log(res1)  console.log(res2)  }))  .catch((error) =>{  if (axios.isCancel(error)) {  console.log(error.message);  }else{  console.log(error)  }  })  }  }  </script> |

### 7 拦截器

我们可以对请求和相应进行拦截



请求拦截案例：



只有return config 后才会发哦送你个请求

响应拦截案例：

