爱创课堂前端培训

# VueJS

第4天课堂笔记（本课程共9天）

班级：北京前端训练营23期

讲师：张容铭

日期：2019年7月26日

张容铭老师

微博：@张容铭\_YYQH

QQ : 286031482

E-mail : yuye\_qinghe@qq.com

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

[VueJS 1](#_Toc10895)

[复习 4](#_Toc24317)

[一、vuex 5](#_Toc22213)

[1.1 flux 5](#_Toc6053)

[1.1.1组成部分 5](#_Toc26128)

[1.1.2通信流程 5](#_Toc27292)

[1.2 vuex 5](#_Toc25795)

[1.2.1组成部分 6](#_Toc6491)

[1.2.2通信流程 6](#_Toc8988)

[1.2.3 action分类 6](#_Toc7465)

[1.3 store组成 6](#_Toc32580)

[1.4安装vuex 6](#_Toc26017)

[1.5使用vuex 7](#_Toc21552)

[1.6同步消息 8](#_Toc3202)

[1.6.1 commit 8](#_Toc23740)

[1.7异步消息 9](#_Toc12523)

[1.7.1 dispatch 9](#_Toc5243)

[1.8计算属性数据 10](#_Toc32100)

[1.9 modules 11](#_Toc11271)

[1.10插件 13](#_Toc14197)

[1.10.1检测motations变化 13](#_Toc5338)

[1.10.2自定义插件 13](#_Toc27789)

[下午复习 14](#_Toc2301)

[二、路由 15](#_Toc12167)

[2.1前端路由实现 15](#_Toc21767)

[2.2 vue路由 15](#_Toc13085)

[2.2.1安装路由 16](#_Toc8007)

[2.2.2使用路由 16](#_Toc12536)

[2.2.3路由参数 17](#_Toc32014)

[2.2.4 props 19](#_Toc19010)

[2.2.5默认路由 19](#_Toc28529)

[2.2.6路由重定向 19](#_Toc13809)

[2.2.7子路由 20](#_Toc27392)

[2.3路由模式 20](#_Toc29204)

[2.4滚动条 20](#_Toc27199)

[2.5路由导航 21](#_Toc6387)

[2.6路由过渡 21](#_Toc18666)

[2.7路由守卫 22](#_Toc16318)

[三、异步请求 23](#_Toc22085)

[3.1使用axios 23](#_Toc13339)

[3.2安装 24](#_Toc16136)

# 复习

自定义指令 1 v-ickt-html 2 Vue.directive(name, {bind, update, unbind, inserted, componentUpdated} | fn)

dom, obj, vNode, oldVNode value !== oldValue

自定义组件 1 home

2 Vue.extend({ template, data, watch, computed, methods })

3 注册 Vue.component components

动态组件 component is

组件生命周期

创建期 beforeCreate， created， beforeMount， mounted

存在期 beforeUpdate， updated

销毁期 beforeDestroy， destroyed

keep-alive activated， deactivated

组件通信

父组件向子组件通信 1 属性 2 props: [] | { key: Number | [String] | { type, required, default, validator } }

自定义事件 $on(type, fn) $emit(type, ...arg) $off(type, fn) $once

模拟DOM事件 1 @demo=”fn” 2 定义方法 3 子组件发布消息 4 父组件接收数据，

传递方法 1 :deme=”fn” 2 props 3 子组件执行方法 4 父组件接收数据，

兄弟组件间通信 综合使用子组件向父组件通信以及父组件向子组件通信。

插槽 1 slot属性 2 slot组件 name

template v-slot

作用域 1 slot组件传递属性 2 v-slot:header=”scope” scope.key

v-slot语法糖 #

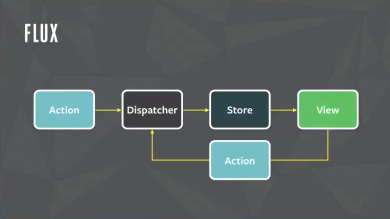
#header=”scope”

# 一、vuex

组件的通信成本很高，所以为了简化组件间通信（共享数据），vue提供了vuex插件。是基于flux思想实现的。

## 1.1 flux

是由react团队提供的数据通信架构思想。解决网状通信的问题。实现了数据单向流动。



### 1.1.1组成部分

由四部分组成

action 消息对象

dispatcher 用来捕获消息对象，并处理数据

store 存储数据

view 组件视图

### 1.1.2通信流程

一个组件或者用户发布一个action消息

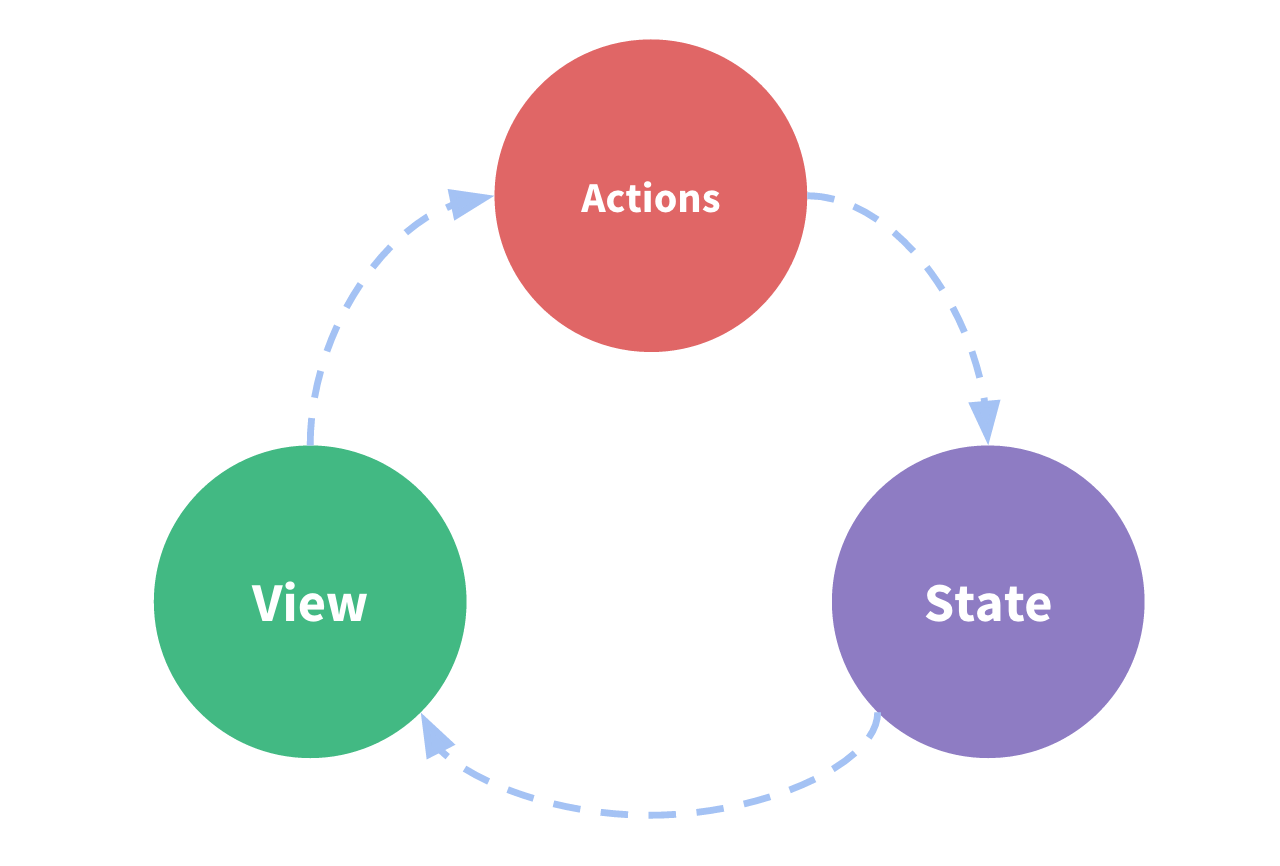
消息被dispatcher捕获，并根据消息类型，处理数据

并将新的数据传递给store去存储，

store中的数据发生改变，将数据传递给另一个组件。

## 1.2 vuex

vuex就是基于flux思想实现的，实现了单一数据源，数据单向流动等特征。



### 1.2.1组成部分

简化了flux，是由三个部分

action 消息模块

state 存储数据

view 组件视图

### 1.2.2通信流程

一个组件发布消息，

消息被action模块捕获，

并根据消息类型，处理数据

action将新的数据传递给state去存储。

state中的数据发生了改变，将新的数据传递给另一个组件。

### 1.2.3 action分类

在vuex中，将action分成两类

一类是同步消息：mutation

为了可测试

一类是异步消息：action

为了异步操作

## 1.3 store组成

下图中，绿色虚线部分就是vuex，在代码中用store表示。

由于vuex实现了单一数据源的特征，所以整个应用程序中，只能有且只有一个store对象

在store中，由五部分组成：

state 存储组件间共享的数据

getters 存储组件间共享的计算属性数据

getters与state的关系与data与computed关系是一样的。

mutations 订阅同步消息的模块（相当于观察者模式中的on方法）

通过commit方法来提交同步消息（相当于观察者模式中的trigger方法）

actions 订阅异步消息的模块（相当于观察者模式中的on方法）

通过dispatch方法来提交异步消息（相当于观察者模式中的trigger方法）

modules 用来切割模块的。

注意：如果没有异步消息，可以直接发布同步消息

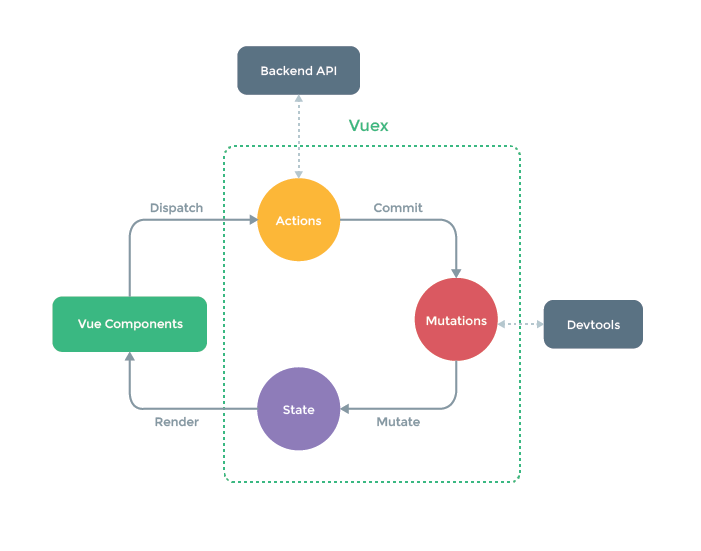
## 1.4安装vuex

在es5开发中，通过bower安装

bower install vuex

在es6开发中，通过npm安装

npm install vuex



## 1.5使用vuex

使用vuex分成五步

第一步 安装vuex

只有在模块化开发中，需要安装（seajs除外）

Vue.use(Vuex)

use方法是专门用来安装插件的。

第二步 实例化store，通过Vue.Store类创建

参数是对象，可以传递各个组成部分：stae，mutatins ......

第三步 注册store，

为vue实例化对象安装store，一旦安装之后，所有的组件都可以获取store对象

所以就可以通过store，获取state中的数据。

注册就是说让所有的组件能够获取$store对象

通过store属性注册。

第四步 一个组件使用store中的数据

第五步 一个组件发布消息

## 1.6同步消息

通过mutations定义同步消息

key表示消息名称

value是消息回调函数

第一个参数表示state数据对象

第二个参数表示接收的数据

### 1.6.1 commit

用来发布同步消息的方法。

第一个参数表示消息名称

第二个参数表示提交的数据

所有的数据只能在第二个参数中传递，

多个数据可以放在对象或者数组中，

|  |
| --- |
| 1. // 1 安装 2. Vue.use(Vuex); 3. // 2 实例化 4. let store = new Store({ 5. // state数据 6. state: { 7. num: 0 8. }, 9. // 同步消息 10. mutations: { 11. // 增加数字 12. addNum(state, num) { 13. // console.log(this, arguments) 14. // 更新数据 15. state.num += +num; 16. }, 17. // 减少数字 18. reduceNum(state, num) { 19. // console.log(this, arguments) 20. // 用num更新state中的数据 21. state.num -= num; 22. } 23. } 24. }) 25. // 增加数字 26. let AddNum = Vue.extend({ 27. template: ` 28. <div> 29. <button @click="$store.commit('addNum', 5)">增加5</button> 30. </div> 31. `, 32. // 组件创建完成 33. created() { 34. console.log('addNum', this) 35. } 36. }) 37. // 显示数字 38. let ShowNum = Vue.extend({ 39. // 4 一个组件使用数据 40. template: ` 41. <h1>num:{{$store.state.num}}</h1> 42. `, 43. // 组件创建完成 44. created() { 45. console.log('showNum', this) 46. } 47. }) 48. // 实例化 49. new Vue({ 50. // 3 注册store 51. store, 52. // 视图 53. el: '#app', 54. // 注册组件 55. components: { AddNum, ShowNum }, 56. // 数据 57. data: {}, 58. // 方法 59. methods: { 60. // 发布减少数字消息 61. sendReduceNumMessage() { 62. // 发布同步消息 63. // 多个数据，放在数组或者对象中 64. // this.$store.commit('reduceNum', [2, 5, 100, true]) 65. this.$store.commit('reduceNum', 2) 66. } 67. }, 68. // 组件创建完成 69. created() { 70. console.log('app', this) 71. } 72. }) |

## 1.7异步消息

通过actions订阅异步消息

key表示消息名称

value是消息回调函数

第一个参数表示store对象

第二个参数表示接收的数据

注意：虽然可以通过store访问state数据，但是不要直接修改数据，必须要通过commit方法发布同步消息，在同步的消息中再修改state数据（否则消息无法测试）

### 1.7.1 dispatch

用来发布异步消息的方法

第一个参数表示消息名称

第二个参数表示传递的数据

注意：数据只能在第二个参数中发布，所有有多个数据的时候，可以将数据放在对象或者数组中。

## 1.8计算属性数据

我们通过getters定义计算属性数据

key表示数据名称

value是计算方法。

第一个参数表示当前的state数据

第二个参数表示当前的getters数据

返回值就是获取的数据。

|  |
| --- |
| 1. let store = new Store({ 2. // state数据 3. state: { 4. num: 0 5. }, 6. // 计算属性数据 7. getters: { 8. doubleNum(state, getters) { 9. // console.log(arguments) 10. // num的二倍 11. return state.num \* 2; 12. } 13. }, 14. // 同步消息 15. mutations: { 16. // 增加数字 17. addNum(state, num) { 18. // console.log(this, arguments) 19. // 更新数据 20. state.num += +num; 21. }, 22. // 减少数字 23. reduceNum(state, num) { 24. // console.log(this, arguments) 25. // 用num更新state中的数据 26. state.num -= num; 27. }, 28. // 清空数字 29. clearNum(state, num) { 30. // 清空数字 31. state.num = num; 32. } 33. }, 34. // 异步消息 35. actions: { 36. // 清空数字 37. clear(store, num) { 38. // console.log(arguments, this) 39. // 异步操作 40. setTimeout(() => { 41. // 发布同步消息 42. store.commit('clearNum', num) 43. }, 2000) 44. } 45. } 46. }) 47. // 增加数字 48. let AddNum = Vue.extend({ 49. template: ` 50. <div> 51. <button @click="$store.commit('addNum', 5)">增加5</button> 52. <!-- <button @click="$store.dispatch('clear', [0, 100, true, 20])">2秒之后清空</button> --> 53. <button @click="$store.dispatch('clear', 0)">2秒之后清空</button> 54. </div> 55. `, 56. // 组件创建完成 57. created() { 58. console.log('addNum', this) 59. } 60. }) |

## 1.9 modules

我们通过modules来切割模块。

解决问题：如果应用程序很大，会有很多的共享数据，放在同一个state中，会有命名冲突问题。为了解决这类问题，我们可以通过modules切割store，来分类存储state数据。

切割出来的store，本质也是store，因此也可以传递store的各个模块：state，getters，mutations，actions，modules

store中的数据是作为state来存储的，因此切割store的本质，就是切割state，所以只有state需要添加命名空间，其它的数据可以正常使用。

modules属性值是对象

key 表示命名空间

value 是子store对象，可以定义store的各个部分。

此时就有了全局store与局部store对象了，它们内部可以定义同名的数据

data 使用的时候，要携带命名空间

mutations,action 正常使用，

局部的消息修改局部的数据

全局的消息修改全局的数据

但是发布同名的消息，局部与全局的都会执行

getters 不允许同名

|  |
| --- |
| 1. let store = new Store({ 2. // 全局数据 3. state: { 4. num: 10, 5. color: 'red' 6. }, 7. // 计算属性数据 8. getters: { 9. dealColor(state) { 10. return state.color.toUpperCase(); 11. }, 12. // 同名计算属性数据，不允许定义 13. // doubleNum() {} 14. }, 15. // 全局消息 16. mutations: { 17. addNum(state, num) { 18. state.num += num; 19. } 20. }, 21. // 切割store 22. modules: { 23. // 命名空间是ickt 24. ickt: { 25. // state数据 26. state: { 27. num: 0 28. }, 29. // 计算属性数据 30. getters: { 31. doubleNum(state, getters) { 32. // console.log(arguments) 33. // num的二倍 34. return state.num \* 2; 35. } 36. }, 37. // 同步消息 38. mutations: { 39. // 增加数字 40. addNum(state, num) { 41. // console.log(this, arguments) 42. // 更新数据 43. state.num += +num; 44. }, 45. // 减少数字 46. reduceNum(state, num) { 47. // console.log(this, arguments) 48. // 用num更新state中的数据 49. state.num -= num; 50. }, 51. // 清空数字 52. clearNum(state, num) { 53. // 清空数字 54. state.num = num; 55. } 56. }, 57. // 异步消息 58. actions: { 59. // 清空数字 60. clear(store, num) { 61. // console.log(arguments, this) 62. // 异步操作 63. setTimeout(() => { 64. // 发布同步消息 65. store.commit('clearNum', num) 66. }, 2000) 67. } 68. } 69. } 70. } 72. }) |

## 1.10插件

vuex允许我们通过插件的形式，为其做拓展。

通过plugins属性引入插件，是一个数组，可以引入多个插件。

vuex内置了一个插件：vuex/dist/logger.js

模块方法要执行，获取log方法

可以检测mutations消息。

不能检测actions消息

### 1.10.1检测motations变化

store对象提供了subscrube方法，可以检测mutations的变化

回调函数有两个参数

第一个参数表示mutation消息类型对象

第二个参数表示state对象

### 1.10.2自定义插件

vuex的插件是一个函数，参数是store对象

在函数中，我们可以对store对象做拓展。

检测mutaion消息。

|  |
| --- |
| 1. let store = new Store({ 2. // 使用插件 3. // plugins: [logger()], 4. // 使用自定义插件 5. plugins: [ myPlugin ] 6. // ...... 8. }) 9. // 检测store变化 10. // store.subscribe((...arg) => console.log(arg)) 11. // store.subscribe((mutation, state) => console.log(mutation.type, state.num)) 12. // 定义裁剪 13. function myPlugin(store) { 14. store.subscribe((mutation, state) => console.log(mutation.type, state.num)) 15. } |

### 下午复习

vuex

架构 state， action， view

store：state， gettters: { key: fn(state, getters){ return } }，

mutations: { key: fn(state, data){} }， commit(type, data)，

actions: { key: fn(store, data) }， dispatch(type, data)，

midules: state

1 use 2 new Store({}) 3 store 4 state 5 commit()

plugins: [logger]

fn(store) {}

subscribe(fn(m, state) {})

state代理

computed: { dealMsg: { set, get } }

# 二、路由

一个完整的URL由几部分组成？

https://www.icketang.com/static/img/banner\_news.jpg?num=10&color=red#title

协议 https://

域名 www.icketang.com

端口号 :443

路径 /static/img/

文件 banner\_news.jpg

搜索词 ?num=10&color=red

哈希 #title

前三个部分的改变会导致跨域的问题。

前六个部分的改变会导致浏览器端向服务器端发送新的请求。

只有hash的改变不会导致浏览器端向服务器端发送新的请求。

所以前端路由就是基于hash实现的，

例如：单页面应用程序SPA（Single Page Application）就是基于hash路由实现的

特点：快

## 2.1前端路由实现

前端的路由是基于hash实现的。

我们可以通过hashchange事件监听hash的变化

我们可以通过location获取hash数据。

location可以获取url上的所有数据

所以我们可以基于动态组件与hash事件实现前端路由。

|  |
| --- |
| // 监听路由变化  window.addEventListener('hashchange', router)  // 路由方法  function router() {  // console.log(location.hash)  // 截取组件的名称  let page = location.hash.slice(2);  // 更新页面  app.page = page;  }  // 页面加载完成执行  router(); |

## 2.2 vue路由

我们实现的路由太脆弱了，所以vue提供了路由插件，方便我们使用路由。

### 2.2.1安装路由

在es5开发中

bower install router

在es6开发中

npm install router

### 2.2.2使用路由

使用路由共分六步：

第一步 在模块化开发中，通过use方法安装路由

第二步 定义路由渲染位置。

通过router-view组件

第三步 定义页面组件。

路由简化了组件的定义

只需要定义extend参数对象集合

第四步 定义路由规则：

路由规则是一个数组，每一个成员代表一条规则，是一个对象：

path 定义规则，类似express中的路由规则

用/切割路径，默认是静态路由

静态路由：是一个一对一的路由规则

例如 /home/ickt 只对应 /home/ickt

在名称的前面，添加:就是一个动态路由规则了

动态路由：是一个一对多的路由规则，

例如 /list/:page

匹配 /list/10 /list/100 /list/abc

不匹配 /list /list/10/20 /detail/10

component 渲染的组件对象

name 组件名称

...

第五步 实例化路由，并传递路由规则：routes

第六步 在vue实例化对象中，注册路由，通过router属性。

|  |
| --- |
| 1. // 1 安装路由 2. Vue.use(Router); 3. // 3 定义组件 4. let Home = { template: '<h1>home page</h1>' } 5. let List = { template: '<h1>list page</h1>' } 6. let Detail = { template: '<h1>detail page</h1>' } 7. // 4 定义规则 8. let routes = [ 9. // 每一个成员代表一条规则 10. { path: '/home', component: Home, name: 'home' }, 11. // 列表页 12. { path: '/list/:page', component: List, name: 'list' }, 13. // 详情页 14. { path: '/detail/:id', component: Detail, name: 'detail' } 15. ]; 16. // 5 实例化路由 17. let router = new Router({ routes }) 18. // 实例化 19. let app = new Vue({ 20. // 6 注册路由 21. router, 22. // 视图 23. el: '#app', 24. // 数据 25. data: {} 26. }) |

### 2.2.3路由参数

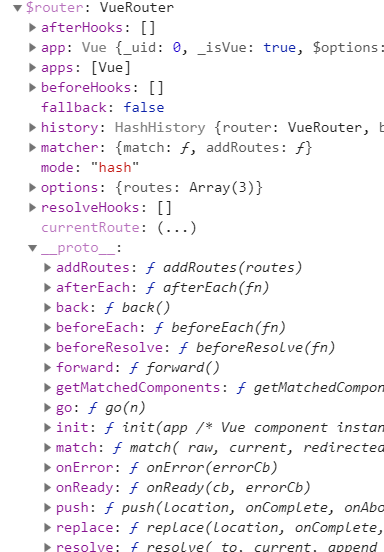
我们在vue实例化对象中，注册了router，此时所有的组件都会继承两个属性：$router, $route

添加给组件实例化对象自身，并设置了特性，所以即可在模板中使用，也可以在组件中使用。

$router 代表路由实例化对象，管理路由，提供了一些操作路由的方法：

例如：push，replace， go

类似js中的history



$route 存路由信息

fullPath 完整路径

hash 路由上的的hash（hash路由中url上第二个#后面的）

path 路径

query search数据（被解析成了对象）

params 动态路由数据（被解析成了对象）



### 2.2.4 props

我们在定义路由规则的时候，

可以设置props属性，可以为渲染的组件，传递属性数据

此时组件要通过props属性接收数据

props: true 将动态路由数据直接传递给组件，

props: fn

函数参数是route对象

返回值对象中的属性会传递给组件。

### 2.2.5默认路由

当没有匹配到路径，希望渲染默认页面的时候，可以使用默认路由。

直接将path匹配\*即可。

由于默认路由匹配的地址很广，因此常常将其配置在最后面。

### 2.2.6路由重定向

当我们输入一个地址，想渲染另外一个页面。此时可以使用路由重定向技术。

通过path匹配地址

通过redirect属性配置重定向的地址。

|  |
| --- |
| 1. let List = { 2. // 接收属性数据 3. props: [ 'page', 'color', 'num' ], 4. template: ` 5. <div> 6. <h1>list page</h1> 7. <h1>page: {{$route.params.page}}, color: {{$route.query.color}}, num: {{$route.query.num}}, hash: {{$route.hash}}</h1> 8. <!-- 使用传递的属性数据 --> 9. <h1>page: {{page}}, color: {{color}}, num: {{num}}</h1> 10. </div> 11. `, 12. // 组件创建完成 13. created() { 14. console.log(this) 15. } 16. } 17. let Detail = { template: '<h1>detail page</h1>' } 18. // 4 定义规则 19. let routes = [ 20. // 每一个成员代表一条规则 21. { path: '/home', component: Home, name: 'home' }, 22. // 列表页 23. { 24. path: '/list/:page', 25. component: List, 26. name: 'list', 27. // 默认传递动态路由数据 28. // props: true 29. // 函数传递多个属性 30. props(route) { 31. // console.log(arguments) 32. return { 33. // page: 'hello' 34. page: route.params.page, 35. color: route.query.color, 36. num: route.query.num 37. } 38. } 39. }, 40. // 详情页 41. { path: '/detail/:id', component: Detail, name: 'detail' }, 42. // 输入ickt 进入/detail/ickt 43. { path: '/ickt', redirect: '/detail/ickt' }, 44. // 默认路由 45. { path: '\*', component: Home } 46. ]; |

### 2.2.7子路由

在页面的某个区域切换子页面，我们可以使用子路由。

使用子路由分成两步

第一步 定义子路由的渲染位置

在父路由模板中，通过router-view来定义

第二步 在路由规则中，我们通过children属性定义子路由配置

是一个数组，每一个成员代表是一个子路由配置对象，

可以配置：path, component, name, redirect, props, children ...

path可以是绝对路径，也可以是相对路径

如果是绝对路径：以/开头的

完整路径：子路由路径

如果是相对路径：不是以/开头的

完整路径：父路由路径+子路由路径

## 2.3路由模式

我们通过mode属性定义路由模块

默认是hash路由，

我们可以设置成history路由（基于path切换页面的）

这种模式需要服务器端的配置

我们在实例化路由的时候，配置路由模式

## 2.4滚动条

我们在切换页面的时候，可以设置滚动条位置

通过scrollbehavior方法设置滚动条位置

第一个参数表示当前路数据由对象

第二个按时表示上一个路由数据对象

第三个参数表示当前滚动条位置

返回值就是设置的滚动条位置

x 代表水平方向滚动条位置

y 代表垂直方向滚动条位置

## 2.5路由导航

路由导航是用来切换路由的。

html中切换页面通过a标签的href属性

js中切换页面通过location对象

为了简化页面的切换，vue提供了路由导航组件：router-link。

to 必须的，用来设置地址，

即使是hash路由，不要加#

tag 用来设置渲染的元素

默认渲染成a标签

v-text 设置内容

是指令，属性值是js环境

内容还可以直接写在组件内。

|  |
| --- |
| 1. let Detail = { 2. template: ` 3. <div> 4. <h1>detail page</h1> 5. <!-- 1 子路由渲染位置 --> 6. <router-view></router-view> 7. </div> 8. ` 9. } 10. // 4 定义规则 11. let routes = [ 12. // 详情页 13. { 14. path: '/detail/:id', 15. component: Detail, 16. name: 'detail', 17. // 2 定义子路由规则 18. children: [ 19. // 绝对路径 20. { path: '/demo', component: { template: '<h1>demo page</h1>' } }, 21. // 相对路径 22. { path: 'search/:word', component: { template: '<h1>search page</h1>' } } 23. ] 24. } 25. ]; |

## 2.6路由过渡

路由过渡允许我们在切换页面的时候，添加过渡动画，

使用方式跟以前使用过渡的方式是一样的。

通过transition组件使用，通过name属性定义过渡名称。

我们可以使用mode属性设置过渡模式

例如：out-in模式

|  |
| --- |
| 1. <transition name="ickt" mode="out-in" appear> 2. <router-view></router-view> 3. </transition> |

## 2.7路由守卫

路由守卫就是监听路由的变化，有三种方式

第一种 全局路由守卫

通过路由实例化对象，订阅事件方法（beforeEach, afterEach等）监听路由变化

参数是回调函数

第一个参数表示当前路由对象

第二个参数表示上一个路由对象

如果是beforeEach有第三个参数，表示next方法，必须执行，类似express中的中间件的next方法。

全局路由守卫对所有组件都生效

第二种 局部路由守卫

在组件中，通过beforeRouteEnter，beforeRouteLeave, beforeRouteUpdate等方法，监听路由变化。

第一个参数表示当前路由对象

第二个参数表示上一个路由对象

第三个参数表示next方法，必须执行

针对当前组件生效

第三种 通过watch监听$route的变化。

第一个参数表示当前路由数据

第二个参数表示上一个路由数据

这种方式第一次不会执行，要等到路由变化的时候，才能执行，

为了弥补创建时候没有执行的问题，我们可以在创建期的方法中执行逻辑。

为了保证watch长期有效，我们可以使用keep-alive组件。

|  |
| --- |
| 1. // 局部路由守卫 2. beforeRouteEnter(route, oldRoute, next) { 3. console.log(arguments); 4. // next方法必须执行 5. next(); 6. }, 7. // watch监听器监听路由变化 8. watch: { 9. $route() { 10. console.log(arguments) 11. } 12. }, 13. // 全局路由守卫 14. router.afterEach((...arg) => console.log(arg)) 15. router.beforeEach((route, oldRoute, next) => { 16. console.log(route, oldRoute, next) 17. // next方法必须执行，否则看不到新的页面 18. next(); 19. }) |

# 三、异步请求

早期vue为了方便开发，提供了一个vue全家桶，包括（vue, vuex, vue-router, vue-resorce），其中vue-resource就是用来发送异步请求的插件，是基于ES5开发，如今我们使用ES6开，因此vue-resource就不适用了，vue作者也不再维护了，建议我们使用异步请求通用框架axios。

axios不是专门为vue设计的，可以应用在各个领域中，

所以axios不是vue家族的插件，不能用Vue.use方法来安装

## 3.1使用axios

axios实现了promise规范：将异步变成同步

是指将异步操作的写法，变成同步的写法。

使用方式跟jquery类似

.get(url, config) 发送get请求的方法

url 请求地址

config 配置对象

.post(url, data, config) 发送post请求的方法

url 请求地址

data 携带的数据

config 配置对象

注意：

1 query数据我们既可以写在url上，也可以在config的params中定义。

2 无论是get请求和post请求都可以添加query数据。

我们通过then方法监听数据返回

参数都是回调函数

第一个参数表示成功时候的回调函数

参数是请求对象，其中的data属性表示返回的数据，我们可以解构使用。

第二个参数表示失败时候的回调函数

失败时候的回调函数还可以定义在catch方法中。

当多次使用then方法的时候，前一个then方法的返回值将作为后一个then方法的参数。

axios发送的请求，使用的是最新的规范：json数据规范

而query是模拟的表单数据：application/x-www-form-urlendcoded.

我们可以在config中，配置请求头的Content-Type属性来修改请求规范。

发送异步请求要搭建服务器。

## 3.2安装

所谓安装就是让所有的组件可以直接使用该数据。

所有的组件都继承了vue类，所以让vue的原型具有axios，所有的组件就可以访问axios。

所以安装axios，可以对vue类的原型拓展axios，

这样所有的组件就可以直接使用axios。

注意：理论上axios可以起任何名字，但是为了语义化，我们常常定义成$http.

|  |
| --- |
| // 引入vue  import Vue from 'vue';  // 引入axios  import axios from 'axios';  // 安装axiso  // Vue.prototype.axios = axios;  // Vue.prototype.ickt = axios;  // 为了语义化，定义成$http  Vue.prototype.$http = axios;  // 定义组件  let Home = Vue.extend({  template: '<h1>home page</h1>',  created() {  console.log('home', this)  }  })  // 实例化  let app = new Vue({  // 注册组件  components: { Home },  // 视图  el: '#app',  // 数据  data: {  msg: ''  },  // 组件创建完成  created() {  // 发送请求  // axios  this.$http  // 1 url上添加query  // .get('data/demo.json?color=red', {  // // 2 配置中添加query  // params: {  // num: 100  // }  // })  // post请求  .post('data/demo.json?color=red', { msg: 'hello' }, {  // 2 配置中添加query  params: {  num: 100  },  // 修改请求格式  headers: {  'Content-Type': 'application/x-www-form-urlencoded'  }  })  // 解构并使用数据  .then(({ data }) => {  // 存储数据  this.msg = data.title;  })  // 监听数据返回  // .then(res => res.data)  // 前一个then方法的返回值，将是下一个then方法的参数  // .then(data => console.log(data))  // .then(res => console.log(res), err => console.log(111, err))  // .catch(err => console.log(222, err))  console.log(this)  }  }) |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |