Vue入门和企业项目开发

第5天课堂笔记

讲师：邵山欢

日期：2018年8月26日

# 路由

## 基本使用

Vue-router是vue的官方路由。

Vue-router是“嵌入式”路由，我们昵称“小电视”风格。

|  |
| --- |
| react-router-redux是“反包式”路由，子组件要自己主动被大组件包。  News.js：  render(){  return <App>  自己的代码  </App>  }  App.js:  render(){  return <div>  {this.props.children}  </div>  } |

官网：https://router.vuejs.org/

npm install vue-router

**<router-view>就是“小电视”。是一个占位符，这个地方到底会显示什么组件，由路由决定。**

入口文件：

import Vue from "vue";

import Vuex from "vuex";

import iview from "iview";

**import VueRouter from 'vue-router';**

import store from "./store/store.js";

**import CarlistLayout from "./layout/CarlistLayout.vue";**

//使用插件

Vue.use(Vuex);

Vue.use(iview);

**Vue.use(VueRouter);**

//路由

**const router = new VueRouter({**

**routes : [**

**{**

**path : "/haha",**

**component : CarlistLayout**

**}**

**]**

**})**

new Vue({

el : "#app",

render(h){

return h({

template : **"<router-view></router-view>"**

});

},

store : new Vuex.Store(store),

**router**

});

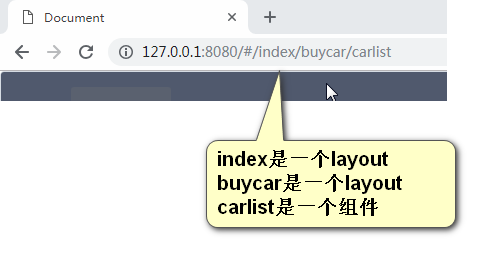
## 什么是layout什么是组件？

* layout文件夹中放置“布局架构”，这些.vue文件中一定会有<router-view></router-view>小电视标签。
* component或者view文件夹中放置 “组件”，这些.vue文件中没有router-view></router-view>小电视标签。

**很多页面都是一样的布局架构，它们的上header和下footer是一样的。**实际上都是layout

地址栏中的一个斜杠，表示一层layout。

换句话说，只要组件中有了router-view，一定伴随着新的一个URL斜杠。



import Index from "./layout/Index.vue";

import BuyCar from "./layout/BuyCar.vue";

import CarList from "./views/carlist/CarList.vue";

export default {

routes : [

{

path : "/index",

component : Index,

children : [

{

path : "buycar" ,

component : BuyCar,

children : [

{

path : "carlist" ,

component : CarList

}

]

}

]

}

]

}

## 首页和默认跳转

import Index from "./layout/Index.vue";

import BuyCar from "./layout/BuyCar.vue";

import CarList from "./views/carlist/CarList.vue";

import IndexComponent from "./views/index/IndexComponent.vue";

export default {

routes : [

{

path : "/index",

component : Index,

children : [

**{**

**path : "index",**

**component : IndexComponent**

**},**

{

path : "buycar" ,

component : BuyCar,

children : [

{

path : "carlist" ,

component : CarList

}

]

}

]

},

**{**

**path : "\*", //输入任意没有没捕获的网址时进行跳转**

**redirect : "/index/index" //跳转到这里**

**}**

]

}

## 菜单和跳转

当网页第一次打开的时候，不能够触发updated声明周期。

而mounted这些生命周期只有第一次打开的时候触发。

也就是说，没有任何一个组件的生命周期可以在打开网页的时候和每次路由跳转的时候都触发。

所以，当用户打开网页或者用户跳转页面的时候，我们需要一个事件钩子，就是说，要有地方写程序。

这个地方就是：

**Navigation Guards，导航守卫。**

import Vue from "vue";

import Vuex from "vuex";

import iview from "iview";

import VueRouter from 'vue-router';

import routes from "./routes.js";

import store from "./store/store.js";

//使用插件

Vue.use(Vuex);

Vue.use(iview);

Vue.use(VueRouter);

**const router = new VueRouter(routes);**

**//路由守卫，今天下午学习session做登陆的时候也要使用这个守卫**

**router.beforeEach((to, from, next) => {**

**next();**

**});**

new Vue({

el : "#app",

render(h){

return h({

template : "<router-view></router-view>"

});

},

store : new Vuex.Store(store),

router

});

为了标记每个子路由在这里是个什么栏目，我们就要使用meta：

import Index from "./layout/Index.vue";

import BuyCar from "./layout/BuyCar.vue";

import CarList from "./views/carlist/CarList.vue";

import IndexComponent from "./views/index/IndexComponent.vue";

export default {

routes : [

{

path : "/index",

component : Index,

children : [

{

path : "index",

component : IndexComponent,

**meta : {**

**"column" : "index"**

**}**

},

{

path : "buycar" ,

component : BuyCar,

**meta : {**

**"column" : "buycar"**

**},**

children : [

{

path : "carlist" ,

component : CarList,

**meta : {**

**"column" : "buycar"**

**}**

}

]

}

]

},

{

path : "\*", //输入任意没有没捕获的网址时进行跳转

redirect : "/index/index" //跳转到这里

}

]

}

跳转：

this.$router.push()

# 分步表单

<template>

<content style="padding:10px 50px;">

<Card>

<Steps :current="$store.state.salecar.step - 1">

<Step :title="showTitle(1)" content="车辆和车主基本信息"></Step>

<Step :title="showTitle(2)" content="车辆图片上传"></Step>

<Step :title="showTitle(3)" content="车辆附属文件资料上传"></Step>

<Step :title="showTitle(4)" content="等待管局审核"></Step>

</Steps>

<div class="h"></div>

<div :is="`Step${$store.state.salecar.step}`"></div>

</Card>

</content>

</template>

# 表单验证

https://github.com/yiminghe/async-validator

<Form ref="formInline" :model="form" :rules="rule" :label-width="120">

<FormItem prop="username" label="车主姓名">

<Input type="text" v-model="form.username"></Input>

</FormItem>

<FormItem prop="idcardnumber" label="车主身份证号">

<Input type="text" v-model="form.idcardnumber"></Input>

</FormItem>

<FormItem prop="brandandseries" label="车辆品牌和型号">

<Cascader :data="brandandseriesoptions" v-model="form.brandandseries"></Cascader>

</FormItem>

<FormItem prop="framenumber" label="车架号">

<Input type="text" v-model="form.framenumber"></Input>

</FormItem>

<FormItem prop="price" label="售价（万元）">

<Input type="text" v-model="form.price"></Input>

</FormItem>

<FormItem prop="km" label="里程表读数（公里）">

<Input type="text" v-model="form.km"></Input>

<Alert type="error" style="margin-top:10px;">私自篡改里程表读数是违法刑法，需要承担刑事责任</Alert>

</FormItem>

//验证规则

rule : {

username: [

{

required: true,

message: '必须输入姓名',

trigger: 'blur'

}

],

idcardnumber : [

{

required: true,

message: '必须输入身份证号',

trigger: 'blur'

},

{

type : "string",

pattern : /^[1-9]\d{5}(18|19|([23]\d))\d{2}((0[1-9])|(10|11|12))(([0-2][1-9])|10|20|30|31)\d{3}[0-9Xx]$/ ,

message: '请输入有效的身份证号码',

trigger: 'blur'

}

],

# HTML5新增的FileReader、FormData、XmlHttpRequest2.0、File

## 基本使用

这些东西用来做用户体验非常好的文件上传功能。

**异步的文件上传使用XHR1.0功能是不行的，IE10以上（不含IE10）开始支持XHR2.0。**

不管是1.0还是2.0，一律都是XMLHttpRequest()对象。只不过高级浏览器支持更多功能。

黄老师教给大家的是传统的IE6都能够兼容的iframe伪装法，我们今天是真正异步上传文件。

http://www.caniuse.com/可以查询兼容性

内置的文件选择框：

<input type="file" ref="file">

内置文件选择框的files属性是一个数组，里面存储的都是File对象

var file = this.$refs.file.files;

console.log(file);



我们现在要把这个File对象上传到服务器上。

必须借助FormData对象。

首先必须创建一个FormData对象。

var fd = new FormData();

在这个FormData对象上“追加”（append）刚刚的file对象：

fd.append("tupian" , fd);

然后我们就可以使用XHR对象，用以前一模一样的Ajax语句进行上传：

var file = this.$refs.file.files[0];

//比起普通的ajax,使用FormData的最大优点就是我们可以异步上传一个二进制文件.

var fd = new FormData();

**fd.append("tupian" , file);**

var self = this;

//ajax即可完成上传

var xhr = new XMLHttpRequest();

//上传进度

xhr.upload.onprogress = function(e){

self.percent = e.loaded / e.total \* 100;

}

xhr.open("post" , "/api/upload" , true);

xhr.send(**fd**);

FormData的API：https://developer.mozilla.org/zh-CN/docs/Web/API/FormData

## 上传前的预览图

HTML5新增了FileReader()内置构造器。可以读取一个本地文件。

它的实例最常用的方法是readAsDataURL方法，可以将图片转为base64文本，从而可以上树显示。

实例代码：

当file控件被改变的时候 = function(){

var file = this.$refs.file.files[0];

**var fr = new FileReader();**

//转为base64

**fr.readAsDataURL(file);**

//备份

var self = this;

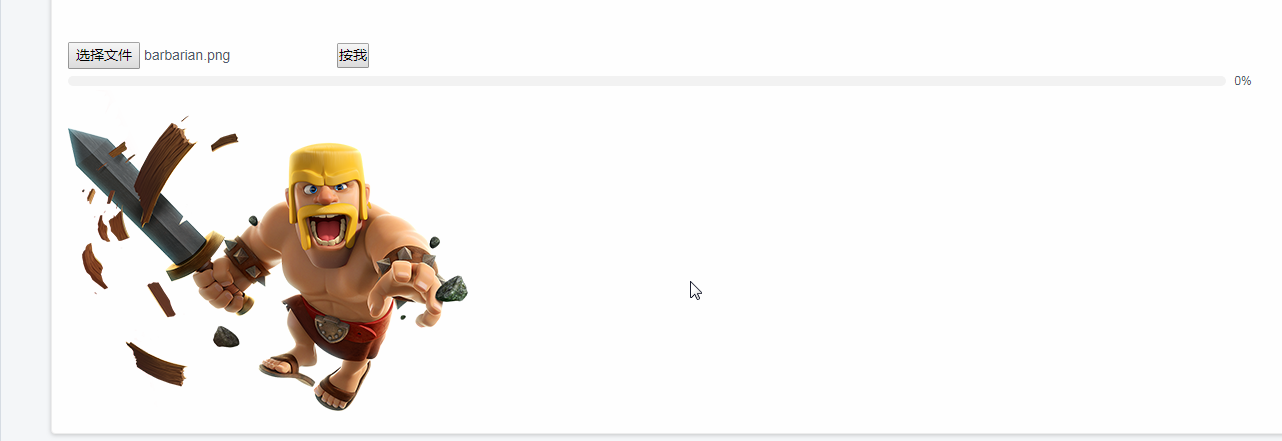
//必须写的一个事件，当图片加载完毕（不是上传）

fr.onload = function(e){

$("<img src=" + **e.target.result** + " />").appendTo($(self.$refs.yulanbox));

}

}



## 批量上传

没啥好说的，就是for循环

## 拖拽

HTML新增drag系列事件，共8个。

* dragstart
* drag
* dragend
* dragenter
* dragover
* dragleave
* drop
* dragexit

$(this.$refs.yulanbox).bind("dragenter" , function(e){

e.preventDefault();

$(this).addClass("blingbling")

});

//这是HTML5新增的事件，拽东西进来扔掉！！

$(this.$refs.yulanbox).bind("drop" , function(e){

e.preventDefault(); //很重要，必须阻止浏览器的默认行为

var files = e.originalEvent.dataTransfer.files;

self.files = self.files.concat(e.originalEvent.dataTransfer.files);

$(this).removeClass("blingbling")

for(let i = 0 ; i < files.length ; i++){

var fr = new FileReader();

//转为base64

fr.readAsDataURL(files[i]);

//必须写的一个事件，当图片加载完毕（不是上传）

fr.onload = function(e){

$("<div data-idx=" + i + " style='background:url(" + e.target.result + ");background-size:cover;background-position:center center;'><p></p></div>").appendTo($(self.$refs.yulanbox));

}

}

});

## 插件

npm install vue-simple-uploader --save