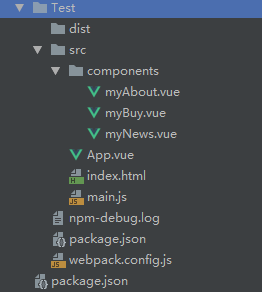
# 一 手动构建vue-router演示

## 1 目录简介

使用vue-router的演示Demo目录如下：



## 2 配置文件 webpack.config.js

const htmlWebpackPlugin = require('html-webpack-plugin');  
const path = require('path')  
module.exports = {  
 entry:{ *//main是默认入口,也可以是多入口* main:'./src/main.js'  
 },  
 *//出口* output:{  
 filename:'./build.js', *//指定js文件* path: path.join(\_\_dirname,'dist') *//最好是绝对路径  
 //代表当前目录的上一级的dist* },  
 module:{  
 *//一样的功能rules: webpack2.x之后新加的* loaders:[ *//require('./a.css||./a.js')* {test:/\.css$/,  
 loader:'style-loader!css-loader',  
 *//顺序是反过来的2!1* },  
 {  
 test:/\.(jpg|svg|png|gif)$/,  
 loader:'url-loader?limit=4096&name=[name].[ext]',  
 *//顺序是反过来的2!1   
 //[name].[ext]内置提供的，因为本身是先读这个文件  
 // options:{  
 // limit:4096,  
 // name:'[name].[ext]'  
 // }* },{*//处理ES6的js* test:/\.js$/,  
 loader:'babel-loader',  
 *//排除 node\_modules下的所有* exclude:/node\_modules/,  
 options:{  
 presets:['es2015'],*//关键字* plugins:['transform-runtime'],*//函数* }  
 },{*//解析vue* test:/\.vue$/,  
 loader:'vue-loader',*//vue-template-compiler是代码上的依赖* }  
 ]  
 },  
  
 plugins:[  
 *//插件的执行顺序是依次执行的* new htmlWebpackPlugin({  
 template:'./src/index.html',  
 })  
 *//将src下的template属性描述的文件根据当前配置的output.path，将文件移动到该目录* ]  
};

package.json配置：

{  
 **"scripts"**: {  
 **"dev"**: **"..\\node\_modules\\.bin\\webpack-dev-server --inline --hot --open"**,  
 **"build"**: **"webpack"** }  
}

## 3 模板文件index.html

<body>  
<div id="App"></div>  
</body>

## 4 入口文件main.js

import Vue from 'vue';  
import App from './App.vue';  
  
import VueRouter from 'vue-router'; *//引入路由插件*Vue.use(VueRouter); *//挂载插件*import myAbout from './components/myAbout.vue';  
import myNews from './components/myNews.vue';  
import myBuy from './components/myBuy.vue';  
  
*//创建路由规则方式*let router = new VueRouter({  
 routes:[   
 {path:'/news',component:myNews},  
 {path:'/buy',component:myBuy},  
 {name:'about',path:'/about',component:myAbout}*//推荐*

]  
});  
  
new Vue({  
 el: '#App',  
 router:router, *//可以直接简写为router* render:create=>create(App)  
});

## 5 根组件App.vue

<template>  
 <div>  
 <h2>一个神奇的网站</h2>  
 <!--最原始的做法-->  
 <a href="#/news">新闻</a>

<!--vue默认做法-->  
 <router-link to="/buy">商城</router-link>   
 <!--推荐做法-->

<router-link :to="{name:'about'}">关于我</router-link>   
 <router-view></router-view>  
 <h4>联系我@weibo.com</h4>  
 </div>  
</template>

## 6 package.json

{  
 **"name"**: **"webpack\_class"**,  
 **"version"**: **"1.0.0"**,  
 **"description"**: **""**,  
 **"main"**: **"index.js"**,  
 **"scripts"**: {  
 **"test"**: **"echo \"Error: no test specified\" && exit 1"**,  
 **"dev"**: **".\\node\_modules\\.bin\\webpack-dev-server --inline --hot --open --port 3000"**,  
 **"build"**: **"webpack"** },  
 **"keywords"**: [],  
 **"author"**: **""**,  
 **"license"**: **"ISC"**,  
 **"devDependencies"**: {  
 **"autoprefixer-loader"**: **"^3.2.0"**,  
 **"babel-core"**: **"^6.25.0"**,  
 **"babel-loader"**: **"^7.1.1"**,  
 **"babel-plugin-transform-runtime"**: **"^6.23.0"**,  
 **"babel-preset-es2015"**: **"^6.24.1"**,  
 **"css-loader"**: **"^0.28.4"**,  
 **"file-loader"**: **"^0.11.2"**,  
 **"html-webpack-plugin"**: **"^2.30.1"**,  
 **"less"**: **"^2.7.2"**,  
 **"less-loader"**: **"^4.0.5"**,  
 **"style-loader"**: **"^0.18.2"**,  
 **"url-loader"**: **"^0.5.9"**,  
 **"vue-loader"**: **"^13.0.4"**,  
 **"vue-template-compiler"**: **"^2.4.2"**,  
 **"webpack"**: **"^3.8.1"**,  
 **"webpack-dev-server"**: **"^2.6.1"** },  
 **"dependencies"**: {  
 **"axios"**: **"^0.16.2"**,  
 **"mint-ui"**: **"^2.2.9"**,  
 **"vue"**: **"^2.4.2"**,  
 **"vue-resource"**: **"^1.3.4"**,  
 **"vue-router"**: **"^2.7.0"** }  
}

# 二 vue-cli构建的结构演示

## 1 安装与配置

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）

配置文件：

.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

2 index模板文件与App.vue

index模板文件中什么都不写，即不标识 id="APP"的div标签，而是在App.vue中书写：

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

## 2 入口文件

|  |
| --- |
| **import** Vue **from 'vue' import** App **from './App' import** router **from './router'** Vue.config.**productionTip** = **false** *le no-//关闭生产环境提示**/\* eslint-disable no-new \*/* **new** Vue({   **el**: **'#app'**,  router,  **template**: **'<App/>'**,  **components**: { App } }) |

# 三 前端路由vue-router

## 1 前端路由与安装

路由： 根据url分配到对应的处理程序

前端路由： 在以前，我们发送请求都是通过ajax或者form表单；

前端路由通过锚点值的改变，根据不同的锚点值，渲染DOM的不同数据。

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> //路由中配置的组件会渲染到该处

## 2 路由模式

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>

还可以修改为历史模式：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 router-link

#### 4.1 router-link基本使用

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。

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

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**>  *<!-- 动态绑定形式-使用名称 推荐使用-->*  <**li**><**router-link :to="**{name:'doc'}**"**>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：

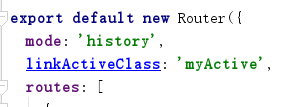
|  |
| --- |
| <**router-link to="/mine" tag="li"**>  <**i**></**i**>  <**span**>mine</**span**> </**router-link**> |

#### 4.3 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**> |

#### 4.4 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

404：在所有路由匹配末尾加入：

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

重定向：

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

|  |
| --- |
| 重定向对象形式：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。 },  动态设置重定向：  redirect: (to) => {  return ‘/mine’  } |

当然也都可以在addRoutes方法内设置：



## 7 get参数传输与获取

在根组件中生成数据，点击链接时候，将数据传递给组件about

data(){  
 return {id:1,name:'zs'}  
}

方式一：配置成 /about?id=1

发送：

<router-link :to="{name:'about',query:{id:id}}">关于我</router-link>

获取：

*//访问进到该组件的路由中挂载着两个对象是属性：  
//$route 信息数据 $router 功能函数*created(){  
 console.log(this.$route.query);  
}

方式二：配置成 /about/1

发送：

<router-link :to="{name:'about',params:{id:id}}">关于我</router-link>

注意这种配置方式需要去路由中额外声明：path:'/about/:id'

获取：

*//访问进到该组件的路由中挂载着两个对象是属性：  
//$route 信息数据 $router 功能函数*created(){  
 console.log(this.$route.params);  
}

方式三：编程导航传参

编程导航可以记录用户的浏览器记录，使用go方法进行前进、后退：

this.$router.go(-1)

也可以使用push方法进入指定的路由：

this.$router.push({name:'news'}) //进入路由mynews，参数也可以是``字符串模板

使用编程导航传参：

this.$router.push({

name:'news',query:{id:1} //即进入 /news?id=1

});

## 8 前端路由原理



# 四 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

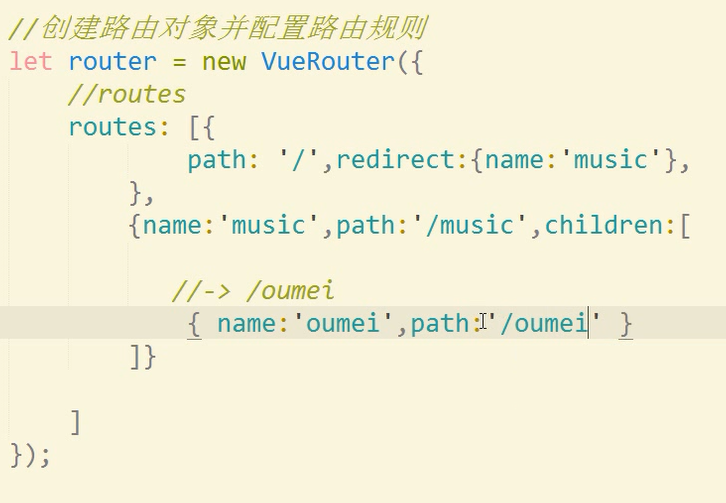


使用：



不写name将渲染default。

多视图：



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  } }, |