

Vue Router

Vue Router 基础回顾

代码演示

使用步骤

1. 创建 router 对象, router/index.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
// 路由组件
import index from '@/views/index'
// 组成插件
Vue.use(VueRouter)
// 路由规则
const routes = [
  {
    name: 'index',
    path: '/',
    component: index
  }
]

// 路由对象
const router = new VueRouter({
  routes
})
export default router
```

2. 注册 router 对象, main.js

```
import router from './router'

new Vue({
  render: h => h(App),
  router
}).$mount('#app')
```

3. 创建路由组建的占位, App.vue

```
<router-view></router-view>
```

4. 创建链接

```
<router-link to="/">首页</router-link>
<router-link :to="{name: 'index'}">首页</router-link>
```

动态路由传参

```
const routes = [
  {
    name: 'detail',
    // 路径中携带参数
    path: '/detail/:id',
    component: detail
  }
]

// detail 组件中接收路由参数
$route.params.id
```


```
const routes = [
  {
    name: 'detail',
    // 路径中携带参数
    path: '/detail/:id',
    component: detail,
    props: true
  }
]

// detail 组件中接收路由参数
const detail = {
  props: ['id'],
  template: '<div>Detail ID: {{ id }}</div>'
}
```

嵌套路由

[代码演示](#)

index 组件和 details 组件 嵌套 layout 组件

 路由嵌套

```
{
  path: '/',
  component: layout,
  children: [
    {
      name: 'index',
```

```

    path: '/',
    component: index
  },
  {
    name: 'details',
    path: '/details/:id',
    component: details
  }
]
}

```

编程式导航

```

// 跳转到指定路径
router.push('/login')
// 命名的路由
router.push({ name: 'user', params: { id: '5' } })

router.replace()
router.go()

```

Hash 模式和 History 模式的区别

- 表现形式的区别
 - Hash 模式

<https://music.163.com/#/playlist?id=3102961863>
 - History 模式

<https://music.163.com/playlist/3102961863>
- 原理的区别
 - hash 模式

Vue Router **默认**使用的是 hash 模式，使用 hash 来模拟一个完整的 URL，通过 onhashchange 监听路径的变化
 - History 模式

基于 [History API](#)

```

history.pushState()
history.replaceState()
history.go()

```

- 开启 History 模式

```
const router = new VueRouter({
  // mode: 'hash',
  mode: 'history',
  routes
})
```

HTML5 History 模式的使用

- History 需要服务器的支持
- 单页应用中，服务端不存在 <http://www.testurl.com/login> 这样的地址会返回找不到该页面
- 在服务端应该除了静态资源外都返回单页应用的 index.html

Node.js 环境

- 示例演示

nginx 环境配置

- 从官网下载 [nginx](#) 的压缩包
- 把压缩包解压到 c 盘根目录，c:\nginx-1.18.0 文件夹
- 修改 conf\nginx.conf 文件

```
location / {
  root    html;
  index   index.html index.htm;
  #新添加内容
  #尝试读取$uri(当前请求的路径)，如果读取不到读取$uri/这个文件夹下的首页
  #如果都获取不到返回根目录中的 index.html
  try_files $uri $uri/ /index.html;
}
```

- 打开命令行，切换到目录 c:\nginx-1.18.0
- nginx 启动、重启和停止

```
# 启动
start nginx
# 重启
nginx -s reload
# 停止
nginx -s stop
```

Vue Router 模拟实现

复习完了 Vue-Router 的基本使用接下来我们自己模拟一个 Vue Router，通过模拟 Vue Router 了解它的实现原理。

前置的知识：插件、slot 插槽、混入、render 函数、运行时和完整版的 Vue

- 回顾 Vue Router 的核心代码

```
// 注册插件
// vue.use() 内部调用传入对象的 install 方法
```

```

Vue.use(VueRouter)
// 创建路由对象
const router = new VueRouter({
  routes: [
    { name: 'home', path: '/', component: homeComponent }
  ]
})
// 创建 Vue 实例，注册 router 对象
new Vue({
  router,
  render: h => h(App)
}).$mount('#app')

```

实现思路

- 创建 LVueRouter 插件，静态方法 install
 - 判断插件是否已经被加载
 - 当 Vue 加载的时候把传入的 router 对象挂载到 Vue 实例上（注意：只执行一次）
- 创建 LVueRouter 类
 - 初始化，options、routeMap、app(简化操作，创建 Vue 实例作为响应式数据记录当前路径)
 - initRouteMap() 遍历所有路由信息，把组件和路由的映射记录到 routeMap 对象中
 - 注册 popstate 事件，当路由地址发生变化，重新记录当前的路径
 - 创建 router-link 和 router-view 组件
 - 当路径改变的时候通过当前路径在 routerMap 对象中找到对应的组件，渲染 router-view

代码实现

- 创建 LVueRouter 插件

```

export default class VueRouter {
  static install (Vue) {
    // 如果插件已经安装直接返回
    if (VueRouter.install.installed && _Vue === Vue) return

    VueRouter.install.installed = true
    _Vue = Vue

    Vue.mixin({
      beforeCreate () {
        // 判断 router 对象是否已经挂载了 Vue 实例上
        if (this.$options.router) {
          // 把 router 对象注入到 Vue 实例上
          _Vue.prototype.$router = this.$options.router
        }
      }
    })
  }
}

```

- 实现 LVueRouter 类 - 构造函数

```
constructor (options) {  
  this.options = options  
  // 记录路径和对应的组件  
  this.routeMap = {}  
  this.app = new _Vue({  
    data: {  
      // 当前的默认路径  
      current: '/'  
    }  
  })  
}
```

- 实现 LVueRouter 类 - initRouteMap()

```
initRouteMap () {  
  // routes => [{ name: '', path: '', component: }]  
  // 遍历所有的路由信息，记录路径和组件的映射  
  this.options.routes.forEach(route => {  
    // 记录路径和组件的映射关系  
    this.routeMap[route.path] = route.component  
  })  
}
```

- 实现 LVueRouter 类 - 注册事件

```
initEvents () {  
  // 当路径变化之后，重新获取当前路径并记录到 current  
  window.addEventListener('hashchange', this.onHashChange.bind(this))  
  window.addEventListener('load', this.onHashChange.bind(this))  
}  
  
onHashChange () {  
  this.app.current = window.location.hash.substr(1) || '/'  
}
```

- 实现 LVueRouter 类 - router-link 和 router-view 组件

```
initComponents () {  
  _Vue.component('RouterLink', {  
    props: {  
      to: String  
    },  
  },
```

```

// 需要带编译器版本的 vue.js
// template: "<a :href='\"#\"' + to><slot></slot></a>"
// 使用运行时版本的 vue.js
render (h) {
  return h('a', {
    attrs: {
      href: '#' + this.to
    }
  }, [this.$slots.default])
}
})

const self = this

_vue.component('RouterView', {
  render (h) {
    // 根据当前路径找到对应的组件, 注意 this 的问题
    const component = self.routeMap[self.app.current]
    return h(component)
  }
})
}

```

- 注意:

- vue-cli 创建的项目默认使用的是[运行时版本的 Vue.js](#)
- 如果想切换成带[编译器版本的 Vue.js](#), 需要修改 [vue-cli 配置](#)
 - 项目根目录创建 vue.config.js 文件, 添加 [runtimeCompiler](#)

```

module.exports = {
  runtimeCompiler: true
}

```

- 实现 LVueRouter 类 - init()

```

init () {
  this.initRouteMap()
  this.initEvents()
  this.initComponents()
}

// 插件的 install() 方法中调用 init() 初始化
if (this.$options.router) {
  _Vue.prototype.$router = this.$options.router
  // 初始化插件的时候, 调用 init
  this.$options.router.init()
}

```

<https://www.jianshu.com/p/4295aec31302>

<https://zhuanlan.zhihu.com/p/27588422>