# 前端路由

在web开发中，路由是指根据url分配到对应的处理程序；

# vue-router

作用：通过管理url，实现url和组件的对应，通过url进行组件之间的切换；

更加方便的管理，实现url和组件的对应，通过url进行组件之间的切换

# 单页应用（spa）

加载单个HTML页面，并在用户与应用程序交互时动态更新页面；

一次性加载，切换时不用再次加载，节约资源

# 使用vue-router

安装模块；

引入模板：

import Vue from ‘vue’;

import VueRouter from “vue-router”;

作为Vue的插件：

Vue.use(VueRouter);

创建路由实例对象：

import ‘home’ form ‘./component/home’

let router = new VueRouter({

routers:[{

path:’/’,

name:’Home’,

alias:’/index’, //和访问home一样，注意访问时候path的class匹配；

component:home

},{

path: '/(coming\_soon|in\_theaters|top250)',//合法的正则都可以；

component: Channel

}]

})

export default router;

注入vue选项参数：

new Vue({

el:’#app’,//渲染的挂载点；

router//路由配置；

template:’<App />’//使用组件；

component:{//注入组件

App

]

})

告知路由渲染位置：

<router-view></router-view>

# hash或者history模式路由

默认是hash模式

hash： 使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。

mode:’history’

history： 这种模式充分利用 history.pushState API 来完成 URL 跳转而无须重新加载页面。

还需要后台配置支持。因为我们的应用是个单页客户端应用，如果后台没有正确的配置，当用户在浏览器直接访问 http://oursite.com/user/id 就会返回 404，这就不好看了。

所以呢，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面，这个页面就是你 app 依赖的页面。

需要支持H5的history api；

# router-link组件

作用：

直接写跳转地址，不用再切换mode模式时候手动修改#进行模式匹配；

前进后退；

点击时默认添加class，可设置；

配置：

<router-link></router-link>

在router-link组件上设置的class回映射在响应组件的根节点上；

to: 跳转地址；

to=”/home”

:to=”/home” //动态绑定

:to=”{path:’/home’}” //动态绑定

tag: 默认a标签，修改生成标签；

html标签

生成a以外的标签不会添加href，默认会添加事件监听来进行url改变；

如果需要查看路径可以在router-link内部再签到a标签，此时router-link的to属性会自动渲染到a标签的href属性上；

class：

默认router-link-active；

路由构造函数选项linkActiveClass: ”active”进行全局配置

单独设置active=”pink”优先级高于路由设置；

event：

改变默认事件click

写入一个事件名

exact：

布尔值（出现即可，可省略），路径精确匹配自动触发点击状态，默认是贪婪匹配

append:

布尔值（出现即可，可省略），在当前路径后添加此路径

exact-active-class:2.5.0+

默认值: "router-link-exact-active"

路由构造函数选项 linkExactActiveClass 进行全局配置

单独设置router-link-exact-active=”pink”优先级高于路由设置；

# 重定向

let router = new VueRouter({

routers:[{

path:’\*’,

//component:home,

//redirect:’/home’//重定向

//redirect:{path:’/home’},

//redirect:{name:’Home’},

redirect: (to) =>{ //动态设置重定向的目标

//to:目标路由对象

//return ‘/home’ //返回跳转路径；

//根据不同的path、hash...可以做不同的重定向目标；

if(to.path === ‘/123’){

return ‘/home’;

}else if(to.path === ‘/456’){

return {path:’/home’}

}else{

return {name:’Home’}

}

}

}]

})

# 路由嵌套

let router = new VueRouter({

routers:[{

path:’/about’

alias:’/index’, //和访问about一样，注意访问时候path的class匹配；

component:about,

children: [{

path:’’,//默认子路由，设置了默认子路由父路由就不要设置name属性了，否则vue会发出警告，name设置在默认子路由即可；,

name:’About’,

component:’work’

},{

path:’study’,//不用加/,加了就是根路径了

name:’study’,

component:study

}]

}]

})

渲染的位置也是在父级组件里，只需添加<router-view></router-view>；

在渲染子路由的时候父组件也会被渲染；

此时渲染路径是 /about/study’;

如果想要’/study’可以这样处理；

在router-link中to属性使用绑定对象表示:to={name:’Study’}；

子路由的path中加上/，形成’/study’； 此时是组件嵌套，路由不嵌套

在渲染时候还是同样的渲染，只是路径上就没有了父级路由；

# 命名视图

有名字的router-view

<router-view></router-view>//渲染默认路由

<router-view name=”slider”></router-view>//命名视图渲染

let router = new VueRouter({

routers:[{

path:’/document’,

alias:’docu’, //和访问about一样，注意访问时候path的class匹配；

name:’Document’,

components:{

default: document,

slider: slider

}

}]

})

# 滚动行为

刷新的时候浏览器会记录上次页面滚动的位置，这是浏览器的默认行为；

当使用浏览器的前进后退时，就是初始化位置；

此时需要使用滚动行为：

let router = new VueRouter({

scrollBehavior(to,from,savePosition){

//在点击浏览器的前进、后退、切换导航时候触发

//to:目标路由对象 去哪里

//from:离开的路由对象 哪里来

//savePosition:记录滚动条的坐标 只有在点击前进后退的时候记录值

if(to.hash){//有hash使用hash

return {

selector:to.hash

}

}eles{//没有hash，检测是否有保存的滚动条坐标；

if(saverPosition){

return saverPosition;

}else{

return {x:0,y:0};

}

}

}

})

# 动态路径

匹配到的所有理由，都映射的同一个组件，根据不用的params进行不同的渲染；

let router = new VueRouter({

routers:[{

path:’/(user/:userId?’, //正则userId? 表示userId可有1次也可以没有；

alias:’/us’, //和访问about一样，注意访问时候path的class匹配；

name:’User’,

component:user,

}]

})

通过this.$route.params.userId访问

路径改变了，页面信息需要重新渲染怎么办？

new Vue({

created(){

//渲染组件时候只会调用一次此声明周期函数；

//复用这个组件，此声明周期函数不会再次调用；

//地址一旦发生变化，$route对象就会重新生成；

//所以可以通过监听$route对象来判断路由是否改变；

//再根据$route对象来重新渲染页面信息；

},

watch:{

$route(){

console.log(this.$route);

}

}  
})

# query字符串传参

相对与当前路径的查询字符串

写入

<router-link to=”?info=share”>分享</router-link>

绑定

<router-link :to=”{path:’’,query:{info:’follow’,num:’1000’}}”>分享</router-link>

# 过渡动效

提供了transition的封装组件，添加过渡动画

添加删除css类名



<transition name=”go” mode=”out-in”>

<router-view></router-view>

</transition>

name:过渡动画前缀：

默认是v

mode:过渡模式：默认同时进行；

in-out：进入的先过渡；

out-in：离开的先过渡；

# 路由元信息

在路由配置中mete可以配置一些数据，用在路由信息对象中访问meta中数据：$route.meta；

let router = new VueRouter({

routers:[{

path:’/’,

component:home,

meta:{

info:’来自index的元信息’

}

}]

})

# 编程式导航

<https://router.vuejs.org/zh/guide/essentials/navigation.html>

借助 router 的实例方法，通过编写代码来实现

<router-link :to="..."> this.$router.push(...)

const userId = 123

this.$router.push({ name: 'user', params: { userId }}) // -> /user/123

this.$router.push({ path: `/user/${userId}` }) // -> /user/123

// 这里的 params 不生效

this.$router.push({ path: '/user', params: { userId }}) // -> /user

<router-link :to="..." replace> this.$router.replace(...)

this.$router.go(n) 意思是在 history 记录中向前或者后退多少步，正数先前，负数后退

this.$router.back(); 后退

this.$router.forward(); 前进

router.push、router.replace 和 router.go 跟 window.history.pushState、 window.history.replaceState 和 window.history.go

# 导航钩子函数

导航发生变化时，导航钩子只要用来拦截导航，让它完成跳转或取消

执行钩子函数位置

router实例全局

单个路由

组件中

钩子函数

router实例上：beforeEach、afterEach、beforeResolve(2.5+)

beforeResolve和 router.beforeEach 类似，区别是在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后

单个路由中：beforeEnter

组件内的钩子：beforeRouteEnter、beforeRouteUpdate、beforeRouteLeave

beforeRouteEnter时组件的实例还没有创建this是undefined，此时next可以传入一个函数，传入一个参数，此参数就是组件的实例；

next( (vm)=>{ vm.age=’23’ } )

钩子函数接收的参数

to： 要进入导航的路由对象 到哪里去

from：要离开导航的路由对象 从哪里来

next：用来决定跳转或取消导航

next(‘/login’) 跳转到/login

next() 正常跳转

解析流程：[https://router.vuejs.org/zh/guide/advanced/navigation-guards.html#%E5%AE%8C%E6%95%B4%E7%9A%84%E5%AF%BC%E8%88%AA%E8%A7%A3%E6%9E%90%E6%B5%81%E7%A8%8B](https://router.vuejs.org/zh/guide/advanced/navigation-guards.html" \l "%E5%AE%8C%E6%95%B4%E7%9A%84%E5%AF%BC%E8%88%AA%E8%A7%A3%E6%9E%90%E6%B5%81%E7%A8%8B)

# 对组件的注入（组件内使用）

通过vue根实例的router配置传入router实例

this.$router router实例对象，每个组件都可以拿到；vue实例注入的router

this.$route 当前激活的路由信息对象，每个组件实例都会有

# 路由信息对象

表示当前激活的路由的状态信息，每次成功的导航后都会产生一个新的对象

$route.path

类型: string

字符串，对应当前路由的路径，总是解析为绝对路径，如 "/foo/bar"。

$route.params

类型: Object

一个 key/value 对象，包含了 动态片段 和 全匹配片段，如果没有路由参数，就是一个空对象。

$route.query

类型: Object

一个 key/value 对象，表示 URL 查询参数。例如，对于路径 /foo?user=1，则有 $route.query.user == 1，如果没有查询参数，则是个空对象。

$route.hash

类型: string

当前路由的 hash 值 (带 #) ，如果没有 hash 值，则为空字符串。

$route.fullPath

类型: string

完成解析后的 URL，包含查询参数和 hash 的完整路径。

$route.matched

类型: Array<RouteRecord>

一个数组，包含当前路由的所有嵌套路径片段的 路由记录 。路由记录就是 routes 配置数组中的对象副本（还有在 children 数组）。

const router = new VueRouter({

routes: [

// 下面的对象就是 route record

{ path: '/foo', component: Foo,

children: [

// 这也是个 route record

{ path: 'bar', component: Bar }

]

}

]

})

当 URL 为 /foo/bar，$route.matched 将会是一个包含从上到下的所有对象（副本）。

$route.name

当前路由的名称，如果有的话。（查看 命名路由）

# vue异步组件

( resolve ) =>{

setTimeout( ()=>{

resolve( require(‘componentUrl’) )

}, 2000 )

}

