# JS单页面应用实现前端路由（hash）

2016年11月21日 23:36:29

阅读数：9261

#### **背景介绍**

用了许多前端框架来做spa应用，比如说backbone，angular，vue他们都有各自的路由系统，管理着前端的每一个页面切换，想要理解其中路由的实现，最好的方法就是手动实现一个。   
前端路由有2种实现方式，一种是html5推出的historyapi，我们这里说的是另一种hash路由，就是常见的 # 号，这种方式兼容性更好。

#### **需求分析**

我们这里只是简单的实现一个路由轮子，基本的功能包含以下：   
切换页面   
异步加载js   
异步传参   
实现步骤   
1.切换页面：路由的最大作用就是切换页面，以往后台的路由是直接改变了页面的url方式促使页面刷新。但是前端路由通过 # 号不能刷新页面，只能通过 window 的监听事件 hashchange 来监听hash的变化，然后捕获到具体的hash值进行操作

//路由切换

window.addEventListener('hashchange',function(){

//do something

this.hashChange()

})

* 1
* 2
* 3
* 4
* 5

2.注册路由：我们需要把路由规则注册到页面，这样页面在切换的时候才会有不同的效果。

//注册函数

map:function(path,callback){

path = path.replace(/\s\*/g,"");//过滤空格

//在有回调，且回调是一个正确的函数的情况下进行存储 以 /name 为key的对象 {callback:xx}

if(callback && Object.prototype.toString.call(callback) === '[object Function]' ){

this.routers[path] ={

callback:callback,//回调

fn:null //存储异步文件状态，用来记录异步的js文件是否下载，下文有提及

}

}else{

//打印出错的堆栈信息

console.trace('注册'+path+'地址需要提供正确的的注册回调')

}

}

//调用方式

map('/detail',function(transition){

...

})

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

3.异步加载js：一般单页面应用为了性能优化，都会把各个页面的文件拆分开，按需加载，所以路由里面要加入异步加载js文件的功能。异步加载我们就采用最简单的原生方法，创建script标签，动态引入js。

var \_body= document.getElementsByTagName('body')[0],

scriptEle= document.createElement('script');

scriptEle.type= 'text/javascript';

scriptEle.src= xxx.js;

scriptEle.async = true;

scriptEle.onload= function(callback){

//为了避免重复引入js，我们需要在这里记录一下已经加载过的文件，对应的 fn需要赋值处理

callback()

}

\_body.appendChild(scriptEle);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10

4.参数传递：在我们动态引入单独模块的js之后，我们可能需要给这个模块传递一些单独的参数。这里借鉴了一下jsonp的处理方式，我们把单独模块的js包装成一个函数，提供一个全局的回调方法，加载完成时候再调用回调函数。

SPA\_RESOLVE\_INIT = function(transition) {

document.getElementById("content").innerHTML = '<p style="color:#F8C545;">当前异步渲染列表页'+ JSON.stringify(transition) +'</p>'

console.log("首页回调" + JSON.stringify(transition))

}

* 1
* 2
* 3
* 4

扩展：以上我们已经完成了基本功能，我们再对齐进行扩展，在页面切换之前beforeEach和切换完成afterEach的时候增加2个方法进行处理。思路是，注册了这2个方法之后，在切换之前就调用beforeEach，切换之后，需要等待下载js完成，在onload里面进行调用 afterEach

//切换之前一些处理

beforeEach:function(callback){

if(Object.prototype.toString.call(callback) === '[object Function]'){

this.beforeFun = callback;

}else{

console.trace('路由切换前钩子函数不正确')

}

},

//切换成功之后

afterEach:function(callback){

if(Object.prototype.toString.call(callback) === '[object Function]'){

this.afterFun = callback;

}else{

console.trace('路由切换后回调函数不正确')

}

},

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16

通过以上的思路分析，再加以整合，我们就完成了一个简单的前端路由，并且可以加到页面进行实际的SPA开发，不过还是非常简陋。

/\*

\*author:https://github.com/kliuj

\*\*使用方法

\* 1：注册路由 : spaRouters.map('/name',function(transition){

//异步加载js

spaRouters.asyncFun('name.js',transition)

//或者同步执行回调

spaRouters.syncFun(function(transition){},transition)

})

2：初始化 spaRouters.init()

3：跳转 href = '#/name'

\*/

(function() {

var util = {

//获取路由的路径和详细参数

getParamsUrl:function(){

var hashDeatail = location.hash.split("?"),

hashName = hashDeatail[0].split("#")[1],//路由地址

params = hashDeatail[1] ? hashDeatail[1].split("&") : [],//参数内容

query = {};

for(var i = 0;i<params.length ; i++){

var item = params[i].split("=");

query[item[0]] = item[1]

}

return {

path:hashName,

query:query

}

}

}

function spaRouters(){

this.routers = {};//保存注册的所有路由

this.beforeFun = null;//切换前

this.afterFun = null;

}

spaRouters.prototype={

init:function(){

var self = this;

//页面加载匹配路由

window.addEventListener('load',function(){

self.urlChange()

})

//路由切换

window.addEventListener('hashchange',function(){

self.urlChange()

})

//异步引入js通过回调传递参数

window.SPA\_RESOLVE\_INIT = null;

},

refresh:function(currentHash){

var self = this;

if(self.beforeFun){

self.beforeFun({

to:{

path:currentHash.path,

query:currentHash.query

},

next:function(){

self.routers[currentHash.path].callback.call(self,currentHash)

}

})

}else{

self.routers[currentHash.path].callback.call(self,currentHash)

}

},

//路由处理

urlChange:function(){

var currentHash = util.getParamsUrl();

if(this.routers[currentHash.path]){

this.refresh(currentHash)

}else{

//不存在的地址重定向到首页

location.hash = '/index'

}

},

//单层路由注册

map:function(path,callback){

path = path.replace(/\s\*/g,"");//过滤空格

if(callback && Object.prototype.toString.call(callback) === '[object Function]' ){

this.routers[path] ={

callback:callback,//回调

fn:null //存储异步文件状态

}

}else{

console.trace('注册'+path+'地址需要提供正确的的注册回调')

}

},

//切换之前一些处理

beforeEach:function(callback){

if(Object.prototype.toString.call(callback) === '[object Function]'){

this.beforeFun = callback;

}else{

console.trace('路由切换前钩子函数不正确')

}

},

//切换成功之后

afterEach:function(callback){

if(Object.prototype.toString.call(callback) === '[object Function]'){

this.afterFun = callback;

}else{

console.trace('路由切换后回调函数不正确')

}

},

//路由异步懒加载js文件

asyncFun:function(file,transition){

var self = this;

if(self.routers[transition.path].fn){

self.afterFun && self.afterFun(transition)

self.routers[transition.path].fn(transition)

}else{

console.log("开始异步下载js文件"+file)

var \_body= document.getElementsByTagName('body')[0];

var scriptEle= document.createElement('script');

scriptEle.type= 'text/javascript';

scriptEle.src= file;

scriptEle.async = true;

SPA\_RESOLVE\_INIT = null;

scriptEle.onload= function(){

console.log('下载'+file+'完成')

self.afterFun && self.afterFun(transition)

self.routers[transition.path].fn = SPA\_RESOLVE\_INIT;

self.routers[transition.path].fn(transition)

}

\_body.appendChild(scriptEle);

}

},

//同步操作

syncFun:function(callback,transition){

this.afterFun && this.afterFun(transition)

callback &&　callback(transition)

}

}

//注册到window全局

window.spaRouters = new spaRouters();

})()