前端单页路由的原理分析

原创 2019-09-25 09:48 · 沪漂码农小邹

一般来说，这些路由插件总是提供两种不同方式的路由方式： Hash 和 History，有时也会提供非浏览器环境下的路由方式Abstract，今天我们主要讲讲Hash 和 History的原理和区别。

1. Hash

1.1 相关 Api

Hash 方法是在路由中带有一个 #，主要原理是通过监听 # 后的 URL 路径标识符的更改而触发的浏览器 hashchange 事件，然后通过获取 location.hash 得到当前的路径标识符，再进行一些路由跳转的操作。

1. location.href：返回完整的 URL
2. location.hash：返回 URL 的锚部分
3. location.pathname：返回 URL 路径名
4. hashchange 事件：当 location.hash 发生改变时，将触发这个事件

比如访问一个路径
http://sherlocked93.club/base/#/page1，那么上面几个值分别为：

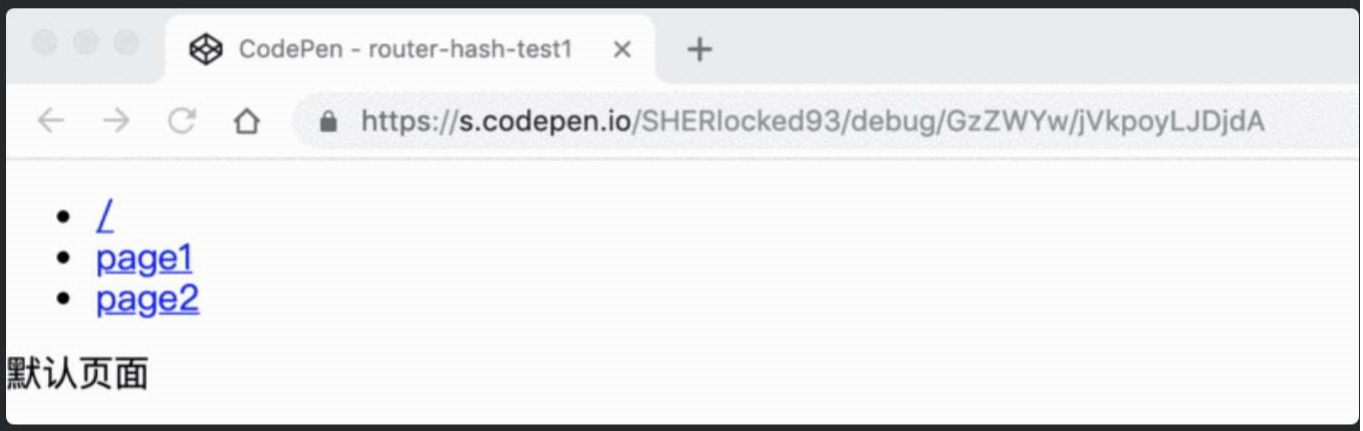
```
# http://sherlocked93.club/base/#/page1
{
 "href": "http://sherlocked93.club/base/#/page1",
```

```
"pathname": "/base/",
"hash": "#/page1"
}
```

> **注意：** Hash 方法是利用了相当于页面锚点的功能，所以与原来的通过锚点定位来进行页面滚动定位的方式冲突，导致定位到错误的路由路径，因此需要采用别的办法。

1.2 实例

这里简单做一个实现，原理是把目标路由和对应的回调记录下来，点击跳转触发 hashchange 的时候获取当前路径并执行对应回调，效果：



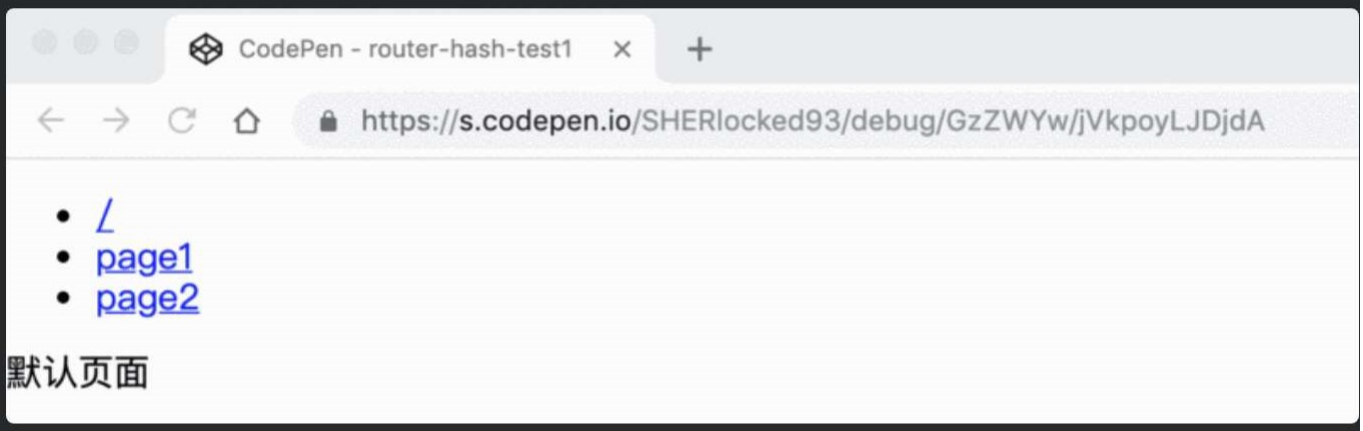

```
class RouterClass {
 constructor() {
 this.routes = {} // 记录路径标识符对应的cb
 this.currentUrl = '' // 记录hash只为方便执行cb
 window.addEventListener('load', () => this.render())
 window.addEventListener('hashchange', () => this.render())
 }

 /* 初始化 */
 static init() {
 window.Router = new RouterClass()
 }

 /* 注册路由和回调 */
 route(path, cb) {
 this.routes[path] = cb || function() {}
 }
```

```
  /* 记录当前hash，执行cb */
  render() {
  this.currentUrl = location.hash.slice(1) || '/'
  this.routes[this.currentUrl]()
  }
}
```

如果希望使用脚本来控制 Hash 路由的后退，可以将经历的路由记录下来，路由后退跳转的实现是对 location.hash 进行赋值。但是这样会重新引发 hashchange 事件，第二次进入 render 。所以我们需要增加一个标志位，来标明进入 render 方法是因为回退进入的还是用户跳转




```
class RouterClass {
  constructor() {
  this.isBack = false
  this.routes = {} // 记录路径标识符对应的cb
  this.currentUrl = '' // 记录hash只为方便执行cb
  this.historyStack = [] // hash栈
  window.addEventListener('load', () => this.render())
  window.addEventListener('hashchange', () => this.render())
  }

  /* 初始化 */
  static init() {
  window.Router = new RouterClass()
  }

  /* 记录path对应cb */
  route(path, cb) {
```

```
  this.routes[path] = cb || function() {}
  }

  /* 入栈当前hash，执行cb */
  render() {
  if (this.isBack) { // 如果是由backoff进入，则置false之后return
  this.isBack = false // 其他操作在backoff方法中已经做了
  return
  }
  this.currentUrl = location.hash.slice(1) || '/'
  this.historyStack.push(this.currentUrl)
  this.routes[this.currentUrl]()
  }

  /* 路由后退 */
  back() {
  this.isBack = true
  this.historyStack.pop() // 移除当前hash，回退到上一个
  const { length } = this.historyStack
  if (!length) return
  let prev = this.historyStack[length - 1] // 拿到要回退到的目标hash
  location.hash = `#${ prev }`
  this.currentUrl = prev
  this.routes[prev]() // 执行对应cb
  }
}
```

2. HTML5 History Api

2.1 相关 Api

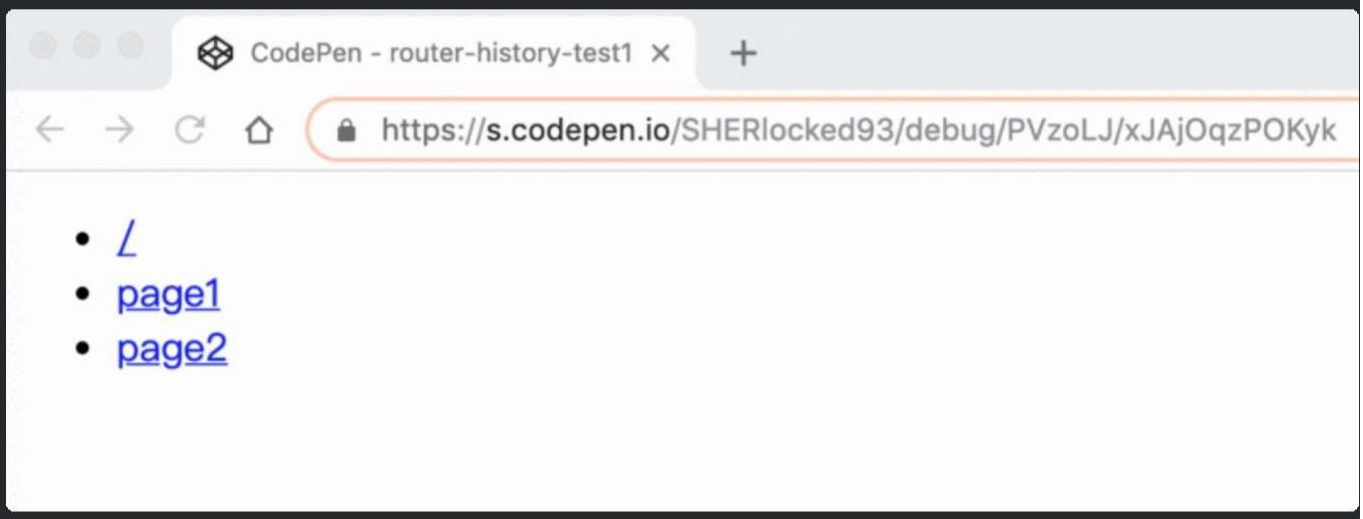
HTML5 提供了一些路由操作的 Api，关于使用可以参看 <Manipulating the browser history> 这篇 MDN 上的文章，这里就列举一下常用 Api 和他们的作用，具体参数什么的就不介绍了，MDN 上都有

1. history.go(n)：路由跳转，比如n为 2 是往前移动2个页面，n为 -2 是向后移动2个页面，n为0是刷新页面
2. history.back()：路由后退，相当于 history.go(-1)

3. history.forward()：路由前进，相当于 history.go(1)
4. history.pushState()：添加一条路由历史记录，如果设置跨域网址则报错
5. history.replaceState()：替换当前页在路由历史记录的信息
6. popstate 事件：当活动的历史记录发生变化，就会触发 popstate 事件，在点击浏览器的前进后退按钮或者调用上面前三个方法的时候也会触发，参见 MDN

2.2 实例

将之前的例子改造一下，在需要路由跳转的地方使用 history.pushState 来入栈并记录 cb，前进后退的时候监听 popstate 事件拿到之前传给 pushState 的参数并执行对应 cb，因为借用了浏览器自己的 Api，因此代码看起来整洁不少




```
class RouterClass {
 constructor(path) {
 this.routes = {} // 记录路径标识符对应的cb
 history.replaceState({ path }, null, path) // 进入状态
 this.routes[path] && this.routes[path]()
 window.addEventListener('popstate', e => {
 const path = e.state && e.state.path
 this.routes[path] && this.routes[path]()
 })
 }

 /* 初始化 */
 static init() {
 window.Router = new RouterClass(location.pathname)
 }
```

```
  /* 注册路由和回调 */
  route(path, cb) {
  this.routes[path] = cb || function() {}
  }

  /* 跳转路由，并触发路由对应回调 */
  go(path) {
  history.pushState({ path }, null, path)
  this.routes[path] && this.routes[path]()
  }
}
```

Hash 模式是使用 URL 的 Hash 来模拟一个完整的 URL，因此当 URL 改变的时候页面并不会重载。History 模式则会直接改变 URL，所以在路由跳转的时候会丢失一些地址信息，在刷新或直接访问路由地址的时候会匹配不到静态资源。因此需要在服务器上配置一些信息，让服务器增加一个覆盖所有情况的候选资源，比如跳转 index.html 什么的，一般来说是你的 app 依赖的页面，事实上 vue-router 等库也是这么推介的。