链滴

前端路由浅析

作者：zjhch123
原文链接：https://ld246.com/article/1538999751386
来源网站：链滴

前端路由浅析

很多传播活动开发过程中经常需要制作一些单页应用。基于种种不可抗原因不能使用Vue、React等架进行开发，所以每次开发时都会当场制作/copy前人的路由代码。

10月传播活动中因为时间充裕，便自己实现了简单的路由，以后活动如果需要使用可以继续基于此进开发。

何为路由

狭义上的路由指根据url变化页面也进行相应的跳变；广义上的路由指监听url的变化，如果url的值与设值相匹配则会触发预设的回调函数。

Hash or History

Hash

hash的原始意义指的是页面上的一个位置，比如https://act.you.163.com/act/static/Dr5VgrU6UG.html#rfMXMkaT这个页面的hash是#rfMXMkaT，其具体位置为"办公用品"，访问这个地址，浏览器会直接定位到"办公用品"这个位置。

hash有一个特性为 - 改变hash不会触发网页重载。因此可以利用这个特性对hash进行改变，利用 onashchange 事件进行监听。

使用 hash 的路由一般会让页面 url 变成这样：

http://test.hduzplus.xyz/#page1
http://test.hduzplus.xyz/#page2
http://test.hduzplus.xyz/#page3

History

vue-router、React-Router 等路由框架可以使用h5新增的 history api 进行路由变化，监听 window 的 onpopstate 事件进行路由监听。

使用 history 的路由页面 url 会清晰的多。

http://test.hduzplus.xyz/page1
http://test.hduzplus.xyz/page2
http://test.hduzplus.xyz/page3

有一定开发基础的小伙伴可能不能理解这种路由监听方式。比如我的主页是http://test.hduzplus.xyz/为什么访问http://test.hduzplus.xyz/page1不是404而是进入到路由逻辑内呢?

这实际上是 history 路由方式的一个缺点。在正常情况下服务端的确会返回404，因此我们得在服务做处理使其返回我们想要的数据才行。具体配置方式在此不表，其配置的主要思想是将该host下404面全部转发到index.html中，并由 index.html 内的 js 判断路由条件并触发相应的回调。

```
<!-- apache 配置demo -->
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
```

```
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

基于hash的路由实现

实现非常简单，无非就是注册对象，监听hashchange，执行相应的函数。

选择使用hash方式的路由的原因是因为就活动开发而言，开发者无法直接对服务端进行配置。

实现代码如下

```
class Router {
  constructor(config) {
    this.config = config
    this.router = {}
    this.from = this.to = ''
    this.view()
  }

  get now() {
    return window.location.hash.substring(1)
  }

  register({
    path, beforeChange = () => {}, change = () => {}, afterChange = () => {}
  }) {
    this.router[path] = {
      beforeChange, afterChange, change
    }
  }

  go(path) {
    if (this.now === path) {
      this.render(this.router[this.now])
      return
    }
    window.location.hash = path
  }

  view() {
    this.config.forEach(item => {
      this.register(item)
    })

    window.addEventListener('hashchange', () => {
      this.render(this.router[this.now])
    })
  }

  render(obj) {
    if (typeof obj === 'undefined' || obj === null) {
      return
    }
```

```
    [this.from ,this.to] = [this.to, this.now];

    const ret = obj.beforeChange(this.from, this.to)
    if (ret === false) {
      return
    }
    obj.change(this.from, this.to)
    obj.afterChange(this.from, this.to)
  }
}

export default Router
```

使用方式：

```
import Router from './Router'

const router = new Router([
  {
    path: 'page1',
    beforeChange: () => {},
    change: () => {},
    afterChange: () => {}
  },
  {
    path: 'page2',
    beforeChange: () => {},
    change: () => {},
    afterChange: () => {}
  },
])
```