曾几何时，网页的路由控制都是后端来控制的。

不同的路由指向哪个页面，返回什么数据都是通过后端的route、controller来实现的。打从前后端分离起，前端也开始自己来操作页面的路由，主要是在SPA和较大的tab页面切换中用的比较多。而实现前端路由功能的工具也很多了，包括jsRouter、backbone-router、vue-router、react-router都实现了前端路由的控制。

那么前端路由的控制是怎么实现的呢？我们先来回忆下和路由相关的两大window对象的属性、方法，分别是location和history。前端路由控制的实现主要就是靠这两个对象的方法来实现的。

一、window对象之Location

Location 对象属性

属性	描述
hash	设置或返回从井号 (#) 开始的 URL（锚）。
host	设置或返回主机名和当前 URL 的端口号。
hostname	设置或返回当前 URL 的主机名。
href	设置或返回完整的 URL。
pathname	设置或返回当前 URL 的路径部分。
port	设置或返回当前 URL 的端口号。
protocol	设置或返回当前 URL 的协议。
search	设置或返回从问号 (?) 开始的 URL（查询部分）。

Location 对象方法

属性	描述
assign()	加载新的文档。
reload()	重新加载当前文档。
replace()	用新的文档替换当前文档。

二、window对象之history

History 对象属性

属性	描述
length	返回浏览器历史列表中的 URL 数量。

History 对象方法

方法	描述
back()	加载 history 列表中的前一个 URL。
forward()	加载 history 列表中的下一个 URL。
go()	加载 history 列表中的某个具体页面。
pushState()	更改当前document的URL
replaceState()	替换当前document的URL

三、前端路由控制方法及兼容性

上面表中标红色的方法就是前端路由控制的关键所在。其实很简单，看mdn文档示例即可：
https://developer.mozilla.org/zh-CN/docs/Web/API/History/pushState

```
history.pushState(state, title, url);
```

很easy，当你在切换到新的组件的时候，就可以执行这个，把当前的路由更

改。反过来，也可以监听路由的变化去加载对应的组件，这些在不同框架的router部分中已经非常丰富且完善，我就不多说了，我更愿意多说一下兼容性。

很遗憾，pushState对IE9不支持。那么对于IE9及以下的浏览器怎么去兼容呢?

浏览器兼容性（IE9不支持）

Feature	Chrome	Edge	Firefox (Gecko)	Internet Explorer	Opera	Saf
replaceState, pushState	5	(Yes)	4.0 (2.0)	10	11.50	5.
history.state	18	(Yes)	4.0 (2.0)	10	11.50	6.

来看一下vue-router是如何兼容IE9的前端路由更新的，查看vue-router的源码push-state.js可以看到（如下截图），supportsPushState方法对pushState方法在不同环境的支持进行了判断。

是否支持pushState的方法判断--引自vue-router:push-state.js

```
export const supportsPushState = inBrowser && (function () {
  const ua = window.navigator.userAgent

  if (
    (ua.indexOf('Android 2.') !== -1 || ua.indexOf('Android 4.0') !== -1) &&
    ua.indexOf('Mobile Safari') !== -1 &&
    ua.indexOf('Chrome') === -1 &&
    ua.indexOf('Windows Phone') === -1
  ) {
    return false
  }

  return window.history && 'pushState' in window.history
})()
```

对于不支持pushState的IE9浏览器，vue-router进一步处理如下，

```
export function pushState (url?: string, replace?: boolean) {
  saveScrollPosition()
  // try...catch the pushState call to get around Safari
  // DOM Exception 18 where it limits to 100 pushState calls
  const history = window.history
  try {
    if (replace) {
      history.replaceState({ key: _key }, '', url)
    } else {
      _key = genKey()
      history.pushState({ key: _key }, '', url)
    }
  } catch (e) {
    window.location[replace ? 'replace' : 'assign'](url)
  }
}
```

意思就是如果history.pushState用不了的话，那就用window.location的replace和assign方法吧，虽然会重新加载文档，没有pushState那么好用而已，但好歹也能搞定不是吗。

四、popstate 事件

顺便提一下popstate事件，对路由变化的监测也是很有用的。

window.onpopstate是popstate事件在window对象上的事件处理程序。

每当处于激活状态的历史记录条目发生变化时,popstate事件就会在对应window对象上触发. 如果当前处于激活状态的历史记录条目是由history.pushState()方法创建,或者由history.replaceState()方法修改过的,则popstate事件对象的state属性包含了这个历史记录条目的state对象的一个拷贝。

调用history.pushState()或者history.replaceState()不会触发popstate事件。popstate事件只会在浏览器某些行为下触发, 比如点击后退、前进按钮(或者在JavaScript中调用history.back()、history.forward()、history.go()方法).

当网页加载时,各浏览器对popstate事件是否触发有不同的表现,Chrome和 Safari会触发popstate事件, 而Firefox不会。