vue-router

vue-router是vue的一个插件，用来创建单页应用。

在使用vue-router，有必要了解什么是单页应用及HTML5 History API。

一、单页应用

单页应用顾名思义，即web前端只有一个页面，页面切换时，并没有刷新页面而是通过js代码渲染新页面。

控制浏览器刷新

1.正常情况下，当页面发生变化时，浏览器会根据新的URL发送请求，页面随之刷新。
2.通过URL.HASH可以解决页面不刷新的问题，即URL中包含#号，#号后面就是URL的HASH。
3.由于HTTP请求中不包括#,当#号前面的部分不变，后面的部分发生变化时页面既不会发送请求也不会刷新页面。

URl.hash

```
http://www.example.com/index.html#print
```

右面的字符代表网页中的一个位置。就是该位置的标识符，浏览器读取这个URL后，会自动将位置滚动至可视区域
网页位置指定标识符，有两个方法。

```
// 使用锚点
<a name="print"></a>

// id属性
<div id="print"></div>
```

页面无刷新切换

修改URL
修改方式:1、直接修改浏览器URL；2、通过链接的点击形式 3.非链接的DOM形式

页面与URL对应

页面不刷新，就要求每个页面都有对应的URL.hash，javascript通过判断不同的URL.hash，渲染不同的页面，如果该页面的展示依赖服务器中的数据源，在切换时会发出ajax请求将服务端数据取到，然后会根据此数据源进行页面的渲染。

hashchange

当通过修改 URL.HASH 的方式来达到URL变化目的时，URL也会变化，但是因为 # 号的缘故，浏览器会不自动刷新到对应的页面。此时需要通过 javascript 来变通实现，定义window的hashchange事件，当URL变化时，会触发这个事件，然后在这个事件中执行新页面渲染。

```
window.addEventListener("hashchange",locationHashChanged,false);
function locationHashChanged(){
    if(loaction.hash  === '#some'){
        //  do ajax
    }
}
```

2、HTML5 History API

操作浏览器的历史记录

history.pushState()

```
window.history.pushState(stateObject, title, url)
```

stateObject： 状态对象，可以理解为拿来存储自定义数据的元素，与url关联在一起;
title：标题，目前各类浏览器都会忽略它，可以忽略，目前传空字符串；
url：URL地址， 新的历史记录的地址。新的URL不一定是绝对路径；如果是相对路径，它将以当前URL为基准；传入的URL与当前URL应该是同源的，否则，pushState()会抛出异常。该参数是可选的；不指定的话则为文档当前URL。
调用pushState()方法将新生成一条历史记录，方便用浏览器的“后退”和“前进”来导航

history.replaceState()

```
window.history.replaceState(stateObject, title, url)
```

与history.pushState()方法基本相同，区别只有一点，history.replaceState()不会新生成历史记录，而是将当前历史记录替换掉。

window.onpopstate

点击浏览器的“前进”、“后退”，或者是由JavaScript调用的history.back()等方法，同时切换前后的两条历史记录都属于同一个网页文档，才会触onpopstate。

三、基本使用方法

通过使用vue及vue-router可以创建单页应用，只需要将整个单页应用涉及的页面拆解成一个个独立的组件。通过将路由（可以理解为每个组件对应的url.hash）映射到各个组件中，然后根据不同的url.hash将对应组件渲染出来。

参见 1.html

四、API介绍

router-view

用来渲染匹配的组件,它基于vue组件。根据当前url,router-view会被替换为url对应的组件的内容
1.可以传递props；
2.支持v-transition及transition-mode（指定两个动态组件之间如何过渡） http://cn.vuejs.org/guide/transitions.html
3.keep-alive(如果把切换出去的组件保留在内存中，可以保留它的状态或可以避免重新渲染。)

v-link、路由对象及路由匹配

v-link

v-link 是一个用来让用户在 vue-router 应用的不同路径间跳转的指令。该指令接受一个 JavaScript 表达式。参考2.html
v-link 会监听点击事件，防止浏览器尝试重新加载页面。
replace。一个带有 replace: true 的链接被点击时产生的跳转不会留下历史记录
参见2.html

路由器实例属性

通过 new VueRouter()生成的对象。

```
var router = new VueRouter();

// 1.router.start(App,el)
// 启动一个启用了路由的应用。创建一个 App 的实例并且挂载到元素 el 。
// 2.router.stop()  ??
// 停止监听 popstate 和 hashchange 事件
// 3.router.map
// 4.router.on
// 5.router.go
// 导航到一个新的路由 配置方式与 v-link 几乎相同
// 6.router.replace
// 不会在浏览器历史创建一条新的纪录
// 7.router.redirect(redirectMap)
    router.redirect({
        // 重定向 /a 到 /b
```

```
        '/a': '/b',
        // 重定向可以包含动态片段
        // 而且重定向片段必须匹配
        '/user/:userId': '/profile/:userId',

        // 重定向任意未匹配路径到 /home
        '*': '/home'
    })
// 8.router.alias(aliasMap)
    router.alias({
        // 匹配 /a 时就像是匹配 /a/b/c
        '/a': '/a/b/c',
        // 别名可以包含动态片段
        // 而且重定向片段必须匹配
        '/user/:userId': '/user/profile/:userId'
    })
// 9.router.beforeEach(hook)
// 路由切换开始时调用,调用发生在整个切换流水线之前
    router.beforeEach(function (transition) {
        if (transition.to.path === '/forbidden') {
            transition.abort()
        } else {
            transition.next()
        }
    })
// 10. router.afterEach(hook)
// 路由切换成功进入激活阶段时被调用
    router.afterEach(function (transition) {
        console.log('成功浏览到: ' + transition.to.path)
    })
```

路由对象

在使用了 vue-router 的应用中，路由对象会被注入每个组件中，赋值为 this.$route ，并且当路由切换时，路由对象会被更新。
参见2.html clickFn方法的实现

路由匹配

动态片段

模式	匹配的路径	$route.params
/user/:username	/user/evan	`{ username: 'evan' }`
/user/:username/post/:post_id	/user/evan/post/123	`{ username: 'evan', post_id: 123 }`

全匹配片段

动态匹配只能匹配路径中的一部分，而全匹配字段类似于动态匹配的贪心版。

模式	匹配的路径	$route.params
/user/*any	/user/a/b/c	`{ any: 'a/b/c' }`
/foo/*any/bar	/foo/a/b/bar	`{ any: 'a/b' }`

路由嵌套

使用场景：

参见 3.html

路由切换

rouer-view在切换过程中，<router-view> 组件可以通过实现一些钩子函数来控制切换过程。 canReuse、canDeactivate、canActivate、deactivate、activate、data(参见 4.html) 每一个回调函数中都有一个参数 transition

```
transition.to

一个代表将要切换到的路径的路由对象。

transition.from

一个代表当前路径的路由对象。

transition.next()

调用此函数处理切换过程的下一步。

transition.abort([reason])

调用此函数来终止或者拒绝此次切换。

transition.redirect(path)

取消当前切换并重定向到另一个路由。
```