转 vue-router 快速入门

vue-router是Vue.js官方的路由插件，它和vue.js是深度集成的，适合用于构建单页面应用。vue的单页面应用是基于路由和组件的，路由用于设定访问路径，并将路径和组件映射起来。传统的页面应用，是用一些超链接来实现页面切换和跳转的。在vue-router单页面应用中，则是路径之间的切换，也就是组件的切换。

本文将以示例的形式来介绍vue-router的各个特性，一共包含6个示例，每个示例都有乞丐版，前5个示例有皇帝版。
乞丐版是将所有代码混杂在一起的HTML页面，皇帝版是基于vue-webpack-simple模板构建的。

第一个单页面应用(01)

现在我们以一个简单的单页面应用开启vue-router之旅，这个单页面应用有两个路径：`/home`和`/about`，与这两个路径对应的是两个组件Home和About。

1. 创建组件

首先引入vue.js和vue-router.js：

```
<script src="js/vue.js"></script>
<script src="js/vue-router.js"></script>
```

然后创建两个组件构造器Home和About：

```
var Home = Vue.extend({
    template: '<div><h1>Home</h1><p>{{msg}}</p></div>',
    data: function() {
        return {
            msg: 'Hello, vue router!'
        }
    }
})
```

```
var About = Vue.extend({
    template: '<div><h1>About</h1><p>This is the tutorial about vue-
router.</p></div>'
})
```

2. 创建Router

```
var router = new VueRouter()
```

调用构造器VueRouter，创建一个路由器实例router。

3. 映射路由

```
router.map({
    '/home': { component: Home },
    '/about': { component: About }
})
```

调用router的map方法映射路由，每条路由以key-value的形式存在，key是路径，value是组件。
例如：`'/home'`是一条路由的key，它表示路径；`{component: Home}`则表示该条路由映射的组件。

4. 使用v-link指令

```
<div class="list-group">
    <a class="list-group-item" v-link="{ path: '/home'}">Home</a>
    <a class="list-group-item" v-link="{ path: '/about'}">About</a>
</div>
```

在a元素上使用`v-link`指令跳转到指定路径。

5. 使用<router-view>标签

```
<router-view></router-view>
```

在页面上使用<router-view></router-view>标签，它用于渲染匹配的组件。

6. 启动路由

```
var App = Vue.extend({})
router.start(App, '#app')
```

路由器的运行需要一个根组件，router.start(App, '#app') 表示router会创建一个App实例，并且挂载到#app元素。
注意：使用vue-router的应用，不需要显式地创建Vue实例，而是调用start方法将根组件挂载到某个元素。

View Demo

当你从GitHub上获取到最新的源代码后，如果想运行皇帝版，以demo01为例，在Git Bash下执行以下命令：

```
npm run demo01-dev
```

然后在浏览器中访问地址http://127.0.0.1:8080

如果要编译和发布，请在Git Bash下执行以下命令：

```
npm run demo01-build
```

编写单页面的步骤

上面的6个步骤，可以说是创建一个单页面应用的基本步骤：

JavaScript

1. 创建组件：创建单页面应用需要渲染的组件

2. 创建路由：创建VueRouter实例
3. 映射路由：调用VueRouter实例的map方法
4. 启动路由：调用VueRouter实例的start方法

HTML

1. 使用v-link指令
2. 使用<router-view>标签

router.redirect

应用在首次运行时右侧是一片空白，应用通常都会有一个首页，例如：Home页。
使用router.redirect方法将根路径重定向到/home路径：

```
router.redirect({
    '/': '/home'
})
```

router.redirect方法用于为路由器定义全局的重定向规则，全局的重定向会在匹配当前路径之前执行。

执行过程

当用户点击v-link指令元素时，我们可以大致猜想一下这中间发生了什么事情：

- vue-router首先会去查找v-link指令的路由映射
- 然后根据路由映射找到匹配的组件
- 最后将组件渲染到<router-view>标签

嵌套路由(02)

嵌套路由是个常见的需求，假设用户能够通过路径/home/news
和/home/message访问一些内容，一个路径映射一个组件，访问这两个路径

也会分别渲染两个组件。

实现嵌套路由有两个要点：

- 在组件内部使用<router-view>标签
- 在路由器对象中给组件定义子路由

现在我们就动手实现这个需求。

组件模板：

```
<template id="home">
    <div>
        <h1>Home</h1>
        <p>{{msg}}</p>
    </div>
    <div>
        <ul class="nav nav-tabs">
            <li>
                <a v-link="{ path: '/home/news'}">News</a>
            </li>
            <li>
                <a v-link="{ path: '/home/message'}">Messages</a>
            </li>
        </ul>
        <router-view></router-view>
    </div>
</template>

<template id="news">
    <ul>
        <li>News 01</li>
        <li>News 02</li>
        <li>News 03</li>
    </ul>
</template>
<template id="message">
    <ul>
        <li>Message 01</li>
        <li>Message 02</li>
        <li>Message 03</li>
    </ul>
</template>
```

组件构造器:

```
var Home = Vue.extend({
    template: '#home',
    data: function() {
        return {
            msg: 'Hello, vue router!'
        }
    }
})

var News = Vue.extend({
    template: '#news'
})

var Message = Vue.extend({
    template: '#message'
})
```

路由映射:

```
router.map({
    '/home': {
        component: Home,
        // 定义子路由
        subRoutes: {
            '/news': {
                component: News
            },
            '/message': {
                component: Message
            }
        }
    },
    '/about': {
        component: About
    }
})
```

在/home路由下定义了一个subRoutes选项，/news和/message是两条子路由，它们分别表示路径/home/news和/home/message，这两条路由分别映射组件News和Message。

该示例运行如下：

View Demo

注意：这里有一个概念要区分一下，/home/news和/home/message是/home路由的子路由，与之对应的News和Message组件并不是Home的子组件。

具名路径(03)

在有些情况下，给一条路径加上一个名字能够让我们更方便地进行路径的跳转，尤其是在路径较长的时候。

我们再追加一个组件NewsDetail，该组件在访问/home/news/detail路径时被渲染，组件模板：

```
<template id="newsDetail">
    <div>
        News Detail - {{$route.params.id}} ......
    </div>
</template>
```

组件构造器：

```
var NewsDetail = Vue.extend({
    template: '#newsDetail'
})
```

具名路由映射

```
router.map({
    '/home': {
        component: Home,
        subRoutes: {
            '/news': {
                name: 'news',
                component: News,
                subRoutes: {
                    'detail/:id': {
                        name: 'detail',
```

```
                        component: NewsDetail
                    }
                }
            },
            '/message': {
                component: Message
            }
        }
    },
    '/about': {
        component: About
    }
})
```

注意：我们在定义/homes/news/和home/news/detail/:id路由时，给该路由指定了name属性。
/:id是路由参数，例如：如果要查看id = '01'的News详情，那么访问路径是/home/news/detail/01。

Home组件和News组件模板：

```
<template id="home">
    <div>
        <h1>Home</h1>
        <p>{{msg}}</p>
    </div>
    <div>
        <ul class="nav nav-tabs">
            <li>
                <a v-link="{ name: 'news'}">News</a>
            </li>
            <li>
                <a v-link="{ path: '/home/message'}">Messages</a>
            </li>
        </ul>
        <router-view></router-view>
    </div>
</template>

<template id="news">
    <div>
        <ul>
            <li>
                <a v-link="{ name: 'detail', params: {id: '01'} }">News
01</a>
```

```
            </li>
            <li>
                <a v-link="{ path: '/home/news/detail/02'}">News 02</a>
            </li>
            <li>
                <a v-link="{ path: '/home/news/detail/03'}">News 03</a>
            </li>
        </ul>
        <div>
            <router-view></router-view>
        </div>
    </div>
</template>
```

`<a v-link="{ name: 'news'}">News</a>`和`<a v-link="{ name: 'detail', params: {id: '01'} }">News 01</a>`这两行HTML代码，使用了用了具名路径。

该示例运行如下：

View Demo

v-link指令

用了这么久的v-link指令，是该介绍一下它了。

> `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，并会在用户点击元素时用该表达式的值去调用 `router.go`。

具体来讲，v-link有三种用法：

```
<!-- 字面量路径 -->
<a v-link="'home'">Home</a>

<!-- 效果同上 -->
<a v-link="{ path: 'home' }">Home</a>

<!-- 具名路径 -->
<a v-link="{ name: 'detail', params: {id: '01'} }">Home</a>
```

`v-link` 会自动设置 `<a>` 的 `href` 属性，你无需使用`href`来处理浏览器的调整，原因如下：

- 它在 HTML5 history 模式和 hash 模式下的工作方式相同，所以如果你决定改变模式，或者 IE9 浏览器退化为 hash 模式时，都不需要做任何改变。
- 在 HTML5 history 模式下，`v-link` 会监听点击事件，防止浏览器尝试重新加载页面。
- 在 HTML5 history 模式下使用 `root` 选项时，不需要在 `v-link` 的 URL 中包含 root 路径。

路由对象(04)

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。

路由对象暴露了以下属性：

- $route.path
  字符串，等于当前路由对象的路径，会被解析为绝对路径，如 `"/home/news"` 。
- $route.params
  对象，包含路由中的动态片段和全匹配片段的键值对
- $route.query
  对象，包含路由中查询参数的键值对。例如，对于 `/home/news/detail/01?favorite=yes` ，会得到`$route.query.favorite == 'yes'` 。
- $route.router
  路由规则所属的路由器（以及其所属的组件）。
- $route.matched
  数组，包含当前匹配的路径中所包含的所有片段所对应的配置参数对象。
- $route.name
  当前路径的名字，如果没有使用具名路径，则名字为空。

在页面上添加以下代码，可以显示这些路由对象的属性：

```
<div>
    <p>当前路径：{{$route.path}}</p>
    <p>当前参数：{{$route.params | json}}</p>
    <p>路由名称：{{$route.name}}</p>
    <p>路由查询参数：{{$route.query | json}}</p>
    <p>路由匹配项：{{$route.matched | json}}</p>
</div>
```

`$route.path`, `$route.params`, `$route.name`, `$route.query`这几个属性很容易理解，看示例就能知道它们代表的含义。

（由于$route.matched内容较长，所以没有将其显示在画面上）

这里我要稍微说一下`$router.matched`属性，它是一个包含性的匹配，它会将嵌套它的父路由都匹配出来。

例如，`/home/news/detail/:id`这条路径，它包含3条匹配的路由：

1. /home/news/detail/:id
2. /home/news
3. /home

另外，带有 `v-link` 指令的元素，如果 `v-link` 对应的 URL 匹配当前的路径，该元素会被添加特定的class，该class的默认名称为`v-link-active`。例如，当我们访问`/home/news/detail/03`这个URL时，根据匹配规则，会有3个链接被添加`v-link-active`。

View Demo

让链接处于活跃状态(05)

以上画面存在两个问题：

1. 当用户点击Home链接或About链接后，链接没有显示为选中
2. 当用户点击News或Message链接后，链接没有显示为选中

设置activeClass

第1个问题，可以通过设定v-link指令的activeClass解决。

```
<a class="list-group-item" v-link="{ path: '/home', activeClass:
'active'}">Home</a>
<a class="list-group-item" v-link="{ path: '/about', activeClass:
'active'}">About</a>
```

设定了v-link指令的activeClass属性后，默认的v-link-active被新的class取代。

第2个问题，为v-link指令设定activeClass是不起作用的，因为我们使用的是bootstrap的样式，需要设置a标签的父元素<li>才能让链接看起来处于选中状态，就像下面的代码所展现的：

```
<ul class="nav nav-tabs">
    <li class="active">
        <a v-link="{ path: '/home/news'}">News</a>
    </li>
    <li>
        <a v-link="{ path: '/home/message'}">Messages</a>
    </li>
</ul>
```

如何实现这个效果呢？你可能会想到，为Home组件的data选项追加一个currentPath属性，然后使用以下方式绑定class。

```
<ul class="nav nav-tabs">
    <li :class="currentPath == '/home/news' ? 'active': ''">
        <a v-link="{ path: '/home/news'}">News</a>
    </li>
    <li :class="currentPath == '/home/message' ? 'active': ''">
        <a v-link="{ path: '/home/message'}">Messages</a>
    </li>
</ul>
```

现在又出现了另一个问题，在什么情况下给currentPath赋值呢？

用户点击v-link的元素时，是路由的切换。
每个组件都有一个route选项，route选项有一系列钩子函数，在切换路由时会执行这些钩子函数。
其中一个钩子函数是data钩子函数，它用于加载和设置组件的数据。

```
var Home = Vue.extend({
    template: '#home',
    data: function() {
        return {
            msg: 'Hello, vue router!',
            currentPath: ''
        }
    },
    route: {
        data: function(transition){
            transition.next({
                currentPath: transition.to.path
            })
        }
    }
})
```

该示例运行效果如下：

View Demo

钩子函数(06)

路由的切换过程，本质上是执行一系列路由钩子函数，钩子函数总体上分为两大类：

- 全局的钩子函数
- 组件的钩子函数

全局的钩子函数定义在全局的路由对象中，组件的钩子函数则定义在组件的route选项中。

全局钩子函数

全局钩子函数有2个：

- `beforeEach`：在路由切换开始时调用
- `afterEach`：在每次路由切换成功进入激活阶段时被调用

组件的钩子函数

组件的钩子函数一共6个：

- data：可以设置组件的data
- activate：激活组件
- deactivate：禁用组件
- canActivate：组件是否可以被激活
- canDeactivate：组件是否可以被禁用
- canReuse：组件是否可以被重用

切换对象

每个切换钩子函数都会接受一个 `transition` 对象作为参数。这个切换对象包含以下函数和方法：

- transition.to
  表示将要切换到的路径的路由对象。
- transition.from
  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。

钩子函数的执行顺序

全局钩子函数和组件钩子函数加起来一共8个，为了熟练vue router的使用，有必要了解这些钩子函数的执行顺序。

为了直观地了解这些钩子函数的执行顺序，在画面上追加一个Vue实例：

```
var well = new Vue({
    el: '.well',
    data: {
        msg: '',
        color: '#ff0000'
    },
    methods: {
        setColor: function(){
            this.color = '#' + parseInt(Math.random()*256).toString(16)
                       + parseInt(Math.random()*256).toString(16)
                       + parseInt(Math.random()*256).toString(16)
        },
        setColoredMessage: function(msg){
            this.msg +=  '<p style="color: ' + this.color + '">' + msg +
'</p>'
        },
        setTitle: function(title){
            this.msg =  '<h2 style="color: #333">' + title + '</h2>'
        }
    }
})
```

well实例的HTML：

```
<div class="well">
    {{{ msg }}}
</div>
```

然后，添加一个RouteHelper函数，用于记录各个钩子函数的执行日志：

```
function RouteHelper(name) {
    var route = {
        canReuse: function(transition) {
            well.setColoredMessage('执行组件' + name + '的钩子函数:canReuse')
            return true
        },
        canActivate: function(transition) {
            well.setColoredMessage('执行组件' + name + '的钩子函
数:canActivate')
            transition.next()
        },
        activate: function(transition) {
```

```
            well.setColoredMessage('执行组件' + name + '的钩子函数:activate')
            transition.next()
        },
        canDeactivate: function(transition) {
            well.setColoredMessage('执行组件' + name + '的钩子函
数:canDeactivate')
            transition.next()
        },
        deactivate: function(transition) {
            well.setColoredMessage('执行组件' + name + '的钩子函
数:deactivate')
            transition.next()
        },
        data: function(transition) {
            well.setColoredMessage('执行组件' + name + '的钩子函数:data')
            transition.next()
        }
    }
    return route;
}
```

最后，将这些钩子函数应用于各个组件：

```
var Home = Vue.extend({
    template: '#home',
    data: function() {
        return {
            msg: 'Hello, vue router!',
            path: ''
        }
    },
    route: RouteHelper('Home')
})

var News = Vue.extend({
    template: '#news',
    route: RouteHelper('News')
})

var Message = Vue.extend({
    template: '#message',
    route: RouteHelper('Message')
})

var About = Vue.extend({
    template: '#about',
    route: RouteHelper('About')
```

```
})
```

我们按照以下步骤做个小实验：

1. 运行应用（访问/home路径）
2. 访问/home/news路径
3. 访问/home/message路径
4. 访问/about路径

View Demo

切换控制流水线

当用户点击了/home/news链接，然后再点击/home/message链接后，vue-router做了什么事情呢？它执行了一个切换管道

如何做到这些呢？这个过程包含一些我们必须要做的工作：

1. 可以重用组件Home，因为重新渲染后，组件Home依然保持不变。
2. 需要停用并移除组件News。
3. 启用并激活组件Message。
4. 在执行步骤2和3之前，需要确保切换效果有效——也就是说，为保证切换中涉及的所有组件都能按照期望的那样被停用/激活。

切换的各个阶段

我们可以把路由的切换分为三个阶段：可重用阶段，验证阶段和激活阶段。

我们以home/news切换到home/message为例来描述各个阶段。
（以下文字描述参考：http://router.vuejs.org/zh-cn/pipeline/index.html）

1. 可重用阶段

检查当前的视图结构中是否存在可以重用的组件。这是通过对比两个新的组件树，找出共用的组件，然后检查它们的可重用性（通过 canReuse 选项）。默认情况下， 所有组件都是可重用的，除非是定制过。

2. 验证阶段

检查当前的组件是否能够停用以及新组件是否可以被激活。这是通过调用路由配置阶段的canDeactivate 和canActivate 钩子函数来判断的。

3.激活阶段

一旦所有的验证钩子函数都被调用而且没有终止切换，切换就可以认定是合法的。路由器则开始禁用当前组件并启用新组件。

此阶段对应钩子函数的调用顺序和验证阶段相同，其目的是在组件切换真正执行之前提供一个进行清理和准备的机会。界面的更新会等到所有受影响组件的 deactivate 和 activate 钩子函数执行之后才进行。

data 这个钩子函数会在 activate 之后被调用，或者当前组件组件可以重用时也会被调用。