

## 小程序自定义组件+vue 路由

### 小程序自定义组件

#### 1. 创建自定义组件

每个自定义组件由四个代码文件组成：

json文件	用于放一些最基本的组件配置
wxml文件	组件模版
wxss文件	组件的样式，只在组件内部节点上生效
js组件	组件的js代码，负责逻辑部分

#### 2. 组件构成


配置文件：

要编写一个自定义组件，首先需要在json文件中进行自定义组件声明（将component字段设置为true）。下面是自定义组件 dialog.json



```
1 {  
2   "component": true,  
3   "usingComponents": {}  
4 }
```

使用自定义组件之前，首先需要在引用页面的json文件中进行引用声明，同时需要提供每个自定义组件的‘标签名’和对应的自定义组件的‘文件路径’（标签名称只能是小写字母，中划线和下划线的组合，组件根目录名不能以‘wx-’为前缀）。下面是引用自定义组件的页面 index.json



```
{  
  "usingComponents": {  
    "dialog": "../../components/Dialog/dialog"  
  }  
}
```

组件模版：

在wxml文件中编写组件模版，与之前的写法相同

```

<!--components/Dialog/dialog.wxml-->
<!--自定义组件的外观结构，比如定义弹框，就主要有三部分，如下-->
<view class="dialogContainer" wx:if="{{modalData.isShow}}">
  <view class="dialog">
    <view wx:if="{{modalData.title}}" class="dialogTitle">{
{modalData.title}}</view>
    <view class="dialogContent">{{modalData.content}}</view>
    <!-- <view class="dialogContent" wx:for="{{
{modalData.contentList}}">
      {{item.Content}}
    </view> -->
    <view class="dialogFooter">
      <view wx:if="{{modalData.cancelText}}" class="dialogBtn"
catchtap='cancelEvent'>{{modalData.cancelText}}</view>
      <view class="dialogBtn submitBtn" catchtap='confirmEvent'>{
{modalData.confirmText}}</view>
    </view>
  </view>
</view>

```

组件样式：

同样类似于页面，wxss 文件中可以指定组件节点的样式。其中的样式仅在组件内部生效。需要注意的是，样式只能通过类选择器，在组件选择器中不应使用id选择器，属性选择器和标签名选择器

组件js：

在自定义组件js中，需要使用component构造器来注册组件，并提供组件的属性，内部数据和自定义方法。组件的属性值和内部数据将被作用于组件wxml的渲染，其中属性值是可以由组件外部传入的

### 3. 组件定义

```

// components/Dialog/dialog.js
Component({
  /**
   * 组件的属性列表*/
  properties: {
    modalData: {
      type: Object,
      value: {},
      observer: function() {
        console.log(1)
        console.log(this.data.modalData)
      }
    }
  },
  /**
   * 私有数据，组件的初始数据可用于模版渲染
   */
  data: {},
  /**
   * 组件的方法列表*/
  methods: {
    // 内部的私有属性，建议以下划线开头，triggerevent用于触发事件
    cancelEvent() {
      //触发取消回调
      this.triggerEvent('cancel'); //抛出组件上绑定的两个事件，在调用的界面用
    },
    confirmEvent() {
      //触发成功回调
      this.triggerEvent('confirm');
    }
  }
})

```

```

type: String, // 类型 (必填), 目前接受的类型包括: String, Number, Boolean, Object, Array, null (表示任意类型)
value: '', // 属性初始值 (可选), 如果未指定则会根据类型选择一个
observer: function(newVal, oldVal, changedPath) {
  // 属性被改变时执行的函数 (可选), 也可以写成在methods段中定义的方法名字符串, 如: '_propertyChange'
  // 通常 newVal 就是新设置的数据, oldVal 是旧数据
}

```

## 4. 组件数据交互 (page与component数据交互)

page表示引用组件页面, component表示自定义组件页面

```

<!--index.wxml-->
<!--该页面用到了自定义弹框组件 -->
<view class="container">

  <button type="primary" bindtap='showDialog'>点击我</button>
  <view class="timeContainer" bindtap='timeSelect'>
    <text>时间自定义控件</text>
  </view>
</view>

<!--接收组件抛出来的两个事件, 并在对应的js页面进行编写 -->
<dialog bindcancel="cancelEvent" bindconfirm="confirmEvent" modal-data="{{modalData}}"/>

```

```

data: {
  modalData: { //可以是其他命名, 对应此页面wxml里modal-data绑定的值, 且里面的内容都会传到组件属性对应的变量
    //只有在里面的才会被传递过去
    isShow: false,
    title: "健身秘诀",
    content: "每天动一动, 身体更健康, 欢迎大家来到瘦身俱乐部, 让我们一起动起来吧!",
    cancelText: "不健身",
    confirmText: "健身",
    contentList: [
      {
        name: "瑜伽",
        Content: "抬起腿来, 呼吸呼吸"
      },
      {
        name: "爵士",
        Content: "快来动起来, 动动动"
      },
      {
        name: "民族舞",
        Content: "我喜欢民族舞, 快来培养气质吧!"
      }
    ],
    color: ['red', 'grey']
  },
},

```

```

// showDialog(){
showDialog(){
  let self=this;
  this.setData({ //this.setData里面是一个对象, 注意写法
    ['modalData.isShow']: true //注: 显示弹框, 只是让isShow显示为true, 其他内容不变
  }, function(){
    console.log(self.data.modalData)
  })
},
hideDialog: function(){
  this.setData({
    ['modalData.isShow']: false //显示弹框, 只是让isShow显示为false, 其他内容不变
  })
},
// 点击取消按钮
cancelEvent(){
  this.hideDialog();
  console.log("取消");
},
// 点击确定按钮
confirmEvent(){
  this.hideDialog();
  console.log("确认");
},

```

## 5. 自定义组件中的 slot

### 1) 单个使用:

组件模版

```

<view class="wrapper">
  <view>这里是组件的内部节点</view>
  <slot></slot>
</view>

```

<!-- 引用组件的页面模版 -->

```

<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot> 的位置上 -->
    <view>这里是插入到组件slot中的内容</view>
  </component-tag-name>
</view>

```

## 2) 多个使用

slot 默认使用单个，若要使用多个，要先声明：

```

// component.js
Component({
  options: {
    multipleSlots: true // 表示在组件定义时的选项中启用多slot支持
  }
})

```

使用 name 属性来区分 slot：

```

<!-- 组件模板 -->
<view class="wrapper">
  <slot name="before"></slot>
  <view>这里是组件的内部细节</view>
  <slot name="after"></slot>
</view>

```

引用页面用 slot 属性来选择要注入的 slot：

```

<!-- 引用组件的页面模版 -->
<view>
  <component-tag-name>
    <!-- 这部分内容将被放置在组件 <slot name="before"> 的位置上 -->
    <view slot="before">这里是插入到组件slot name="before"中的内容</view>
    <!-- 这部分内容将被放置在组件 <slot name="after"> 的位置上 -->
    <view slot="after">这里是插入到组件slot name="after"中的内容</view>
  </component-tag-name>
</view>

```

页面模板 view 里面的内容会直接替换组件 slot 标签里的内容，而不是叠加。

一般是用于非组件逻辑的代码块的注入，不用于数据注入。

## vue 路由

通常 vue 单页面应用（SPA）中前端路由有 2 种实现方式：

第一种：location.hash 模式

第二种：window.history 模式

### 1. hash 模式

随着 ajax 的流行，异步数据请求交互运行在不刷新浏览器的情况下进行。而异步交互体验的更高级版本就是 SPA —— 单页应用。单页应用不仅仅是在页面交互是无刷新的，连页面跳转都是无刷新的，为了实现单页应用，所以就有了前端路由。类似于服务端路由，前端路由实现起来其实也很简单，就是匹配不同的 url 路径，进行解析，然后动态的渲染出区域 html 内容。但是这样存在一个问题，就是 url 每次变化的时候，都会造成页面的刷新。那解决问题的思路便是在改变 url 的情况下，保证页面的不刷新。在 2014 年之前，大家是通过 hash 来实现路由，url hash 就是类似于：

`http://www.xxx.com/#/login`

这种 #。后面 hash 值的变化，并不会导致浏览器向服务器发出请求，浏览器不发出请求，也就不会刷新页面。另外每次 hash 值的变化，还会触发 hashchange 这个事件，通过这个事件我们就可以知道 hash 值发生了哪些变化。然后我们便可以监听 hashchange 来实现更新页面部分内容的操作：

```
function matchAndUpdate () {  
  // todo 匹配 hash 做 dom 更新操作  
}
```

```
window.addEventListener('hashchange', matchAndUpdate)
```

window 对象中有一个事件是 onhashchange，以下几种情况都会触发这个事件

- 直接更改浏览器地址，在最后面增加或改变#hash;
- 通过改变 location.href 或 location.hash 的值;
- 通过触发点击带锚点的链接;
- 浏览器前进后退可能导致 hash 的变化，前提是两个网页地址中的 hash 值不同。

看个例子：

```

<template>
  <div rel="navPage">
    <el-row>
      <el-menu
        :default-active="this.$route.path"
        router
        mode="horizontal"
        background-color="#000"
        text-color="#fff"
        active-text-color="orange">
        <el-menu-item index="/">首页</el-menu-item>
        <el-menu-item index="/advantage">平台优势</el-menu-item>
        <el-menu-item index="/technical">技术特性</el-menu-item>
        <el-menu-item index="/scene">应用场景</el-menu-item>
      </el-menu>
    </el-row>
  </div>
</template>

```

```

import Vue from 'vue';
import Router from 'vue-router';
import advantagePage from '@/components/page/advantagePage';
import technicalPage from '@/components/page/technicalPage';
import scenePage from '@/components/page/scenePage';
// import mainPage from '@/components/page/mainPage';

Vue.use(Router);

export default new Router({
  routes: [
    // {
    //   path: '/',
    //   name: 'mainPage',
    //   component: mainPage
    // },
    {
      path: '/advantage',
      name: 'advantagePage',
      component: advantagePage
    },
    {
      path: '/technical',
      name: 'technicalPage',
      component: technicalPage
    },
    {
      path: '/scene',
      name: 'scenePage',
      component: scenePage
    }
  ]
})

```

## 2. history 模式

使用时只需要在配置路由规则时，加入"mode: 'history'"

window.history 对象包含浏览器的历史，window.history 对象在编写时可不使用 window 这个前缀。history 是实现 SPA 前端路由是一种主流方法，它有几个原始方法：

history.back() - 与在浏览器点击后退按钮相同

history.forward() - 与在浏览器中点击按钮向前相同

history.go(n) - 接受一个整数作为参数，移动到该整数指定的页面，比如 go(1) 相当于 forward()，go(-1) 相当于 back()，go(0) 相当于刷新当前页面

如果移动的位置超出了访问历史的边界，以上三个方法并不报错，而是静默失败

在 HTML5, history 对象提出了 `pushState()` 方法和 `replaceState()` 方法, 这两个方法可以用来向历史栈中添加数据, 就好像 url 变化了一样 (过去只有 url 变化历史栈才会变化), 这样就可以很好的模拟浏览历史和前进后退了, 现在的前端路由也是基于这个原理实现的。

通过 `pushState` 和 `replaceState` 两个 API 可以改变 url 地址且不会发送请求。同时还有 `popstate` 事件。通过这些就能用另一种方式来实现前端路由了, 但原理都是跟 hash 实现相同的。用了 HTML5 的实现, 单页路由的 url 就不会多出一个 #, 变得更加美观。但因为没有 # 号, 所以当用户刷新页面之类的操作时, 浏览器还是会给服务器发送请求。为了避免出现这种情况, 所以这个实现需要服务器的支持, 需要把所有路由都重定向到根页面。因为我们的应用是个单页客户端应用, 如果后台没有正确的配置, 当用户在浏览器直接访问 <http://oursite.com/user/id> 就会返回 404, 这就不好看了。

所以, 需要在服务端增加一个覆盖所有情况的候选资源: 如果 URL 匹配不到任何静态资源, 则应该返回同一个 `index.html` 页面, 这个页面就是你 app 依赖的页面。

比如:

```
export const routes = [
  {path: '/', name: 'homeLink', component: Home},
  {path: '/register', name: 'registerLink', component: Register},
  {path: '/login', name: 'loginLink', component: Login},
  {path: '*', redirect: '/'}
]
```

此处就设置如果 URL 输入错误或者是 URL 匹配不到任何静态资源, 就自动跳到 Home 页面

### history.pushState:

`pushState(stateObj, title, url)` 方法向历史栈中写入数据, 其第一个参数是要写入的数据对象 (不大于 640kB), 第二个参数是页面的 title, 第三个参数是 url (相对路径)

- `stateObj`: 一个与指定网址相关的状态对象, `popstate` 事件触发时, 该对象会传入回调函数。如果不需要这个对象, 此处可以填 `null`。
- `title`: 新页面的标题, 但是所有浏览器目前都忽略这个值, 因此这里可以填 `null`。
- `url`: 新的网址, 必须与当前页面处在同一个域。浏览器的地址栏将显示这个网址。

关于 `pushState`，有几个值得注意的地方：

- `pushState` 方法不会触发页面刷新，只是导致 `history` 对象发生变化，地址栏会有反应，只有当触发前进后退等事件（`back()` 和 `forward()` 等）时浏览器才会刷新
- 这里的 `url` 是受到同源策略限制的，防止恶意脚本模仿其他网站 `url` 用来欺骗用户，所以当违背同源策略时将会报错

### **history.replaceState:**

`replaceState(stateObj, title, url)` 和 `pushState` 的区别就在于它不是写入而是替换修改浏览历史中当前纪录，其余和 `pushState` 一模一样

### **popstate 事件:**

- 定义：每当同一个文档的浏览历史（即 `history` 对象）出现变化时，就会触发 `popstate` 事件。
- 注意：仅仅调用 `pushState` 方法或 `replaceState` 方法，并不会触发该事件，只有用户点击浏览器倒退按钮和前进按钮，或者使用 JavaScript 调用 `back`、`forward`、`go` 方法时才会触发。另外，该事件只针对同一个文档，如果浏览历史的切换，导致加载不同的文档，该事件也不会触发。
- 用法：使用的时候，可以为 `popstate` 事件指定回调函数。这个回调函数的参数是一个 `event` 事件对象，它的 `state` 属性指向 `pushState` 和 `replaceState` 方法为当前 URL 所提供的状态对象（即这两个方法的第一个参数）。

## **3. vue-Router**

这里的路由并不是指我们平时所说的硬件路由器，这里的路由就是 SPA（单页应用）的路径管理器。再通俗的说，`vue-router` 就是 WebApp 的链接路径管理系统。

`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。路由模块的本质 就是建立起 `url` 和页面之间的映射关系。

至于我们为啥不能用 `a` 标签，这是因为用 Vue 做的都是单页应用，就相当于只有一个主的 `index.html` 页面，所以写的 `<a></a>` 标签是不起作用的，必须使用 `vue-router` 来进行管理。



## 实现原理：

SPA(single page application):单一页面应用程序，只有一个完整的页面；它在加载页面时，不会加载整个页面，而是只更新某个指定的容器中内容。单页面应用(SPA)的核心之一是：更新视图而不重新请求页面；vue-router 在实现单页面前端路由时，提供了两种方式：Hash 模式和 History 模式；根据 mode 参数来决定采用哪一种方式。

vue-router 默认 hash 模式 —— 使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。hash (#) 是 URL 的锚点，代表的是网页中的一个位置，单单改变#后的部分，浏览器只会滚动到相应位置，不会重新加载网页，也就是说 #是用来指导浏览器动作的，对服务器端完全无用，HTTP 请求中也不会不包括#；同时每一次改变#后的部分，都会在浏览器的访问历史中增加一个记录，使用”后退”按钮，就可以回到上一个位置；所以说 Hash 模式通过锚点值的改变，根据不同的值，渲染指定 DOM 位置的不同数据使用路由模块来实现页面跳转的方式：

方式 1: 直接修改地址栏

方式 2: `this.$router.push('路由地址')`

方式 3: `<router-link to="路由地址"></router-link>`

## vue-Router 使用方式：

下载 `npm i vue-router -S`

在 main.js 中引入 `import VueRouter from 'vue-router';`

安装插件 `Vue.use(VueRouter);`

创建路由对象并配置路由规则

```
let router = new VueRouter({routes:[{path:'/home',component:Home}]});
```

将其路由对象传递给 Vue 的实例，options 中加入 `router:router`

在 app.vue 中 `<router-view></router-view>`

在用 vue 脚手架时，自动安装的 vue-router，会在 src 文件夹下有个一个 `router -> index.js` 文件 在 index.js 中创建 routers 对象，引入所需的组件并配置路径

```

import Vue from 'vue';
import Router from 'vue-router';
// import advantagePage from '@/components/page/advantagePage';
// import technicalPage from '@/components/page/technicalPage';
// import scenePage from '@/components/page/scenePage';
import mainPage from '@/components/page/mainPage';

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: '/',
      name: 'mainPage',
      component: mainPage
    }
    // {
    //   path: '/advantage',
    //   name: 'advantagePage',
    //   component: advantagePage
    // },
    // {
    //   path: '/technical',
    //   name: 'technicalPage',
    //   component: technicalPage
    // }
  ]
});

```

在创建的 routers 对象中， path 配置了路由的路径， component 配置了映射的组件

然后在 main.js 里面引入 router 文件

```

import Vue from 'vue';
import App from './App';
import router from './router';
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';
import './global/style/common.css';

Vue.use(ElementUI);
Vue.config.productionTip = false;
/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
}).$mount('#app');

```

vue-Router 映射路由：

如果只需要跳转页面，不需要添加验证方法的情况，可以使用<router-link>来实现，<router-link> 就是定义页面中点击的部分，<router-view> 定义显示

部分，就是点击后，匹配的内容显示在什么地方。所以 `<router-link>` 还有一个非常重要的属性 `to`，定义点击之后，要到哪里去；js配置路由时首先要定义route， 一条路由的实现。它是一个对象，由两个部分组成： `path`和 `component`。 `path` 指路径，`component` 指的是组件。接着创建router 对路由进行管理，它是由构造函数 `new vueRouter()` 创建，接受`routes` 参数，最后配置完成后，把router 实例注入到 vue 根实例中,就可以使用路由了，如下：

```
<title>vue-router</title>
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
</head>
<body>
  <div id="app1">
    <h1>hello vue!</h1>
    <p>
      <!-- 使用router-link组件来导航，通过传入'to'属性制定链接，<router-link>默认会被渲染成一个'a'标签 -->
      <router-link to="/foo">go to foo</router-link>
      <router-link to="/bar">go to bar</router-link>
    </p>
    <!-- 路由出口，路由匹配到的组件将渲染这里 -->
    <router-view></router-view>
  </div>
```

```
<script>
// 如果使用模块化机制编程，导入vue和vue router,要调用Vue.use(VueRouter)
// 1.定义路由组件
// 可以从其他文件import进来
const foo = {template: '<div>foo</div>'};
const bar = {template: '<div>bar</div>'};
// 2.定义路由
// 每个路由应该映射一个组件，其中component可以通过通过Vue.extend()创建的组件构造器或者只是一个组件构造器
const routes = [
  { path: '/foo',component: foo},
  { path: '/bar',component: bar}
]
// 3.创建vue实例，然后传'routes'配置，还可以传别的配置参数
const router = new VueRouter({
  routes // (缩写)相当于routes: routes
})
// 4.创建和挂载根实例
// 注意要通过router配置参数注入路由，从而让整个应用都有路由功能
const app = new Vue({
  router
}).$mount('#app1')
```

vue-Router 嵌套路由：

```

<div id="box">
  <p>
    <router-link to="/home">home</router-link>
    <router-link to="/news">news</router-link>
  </p>
  <router-view></router-view>
</div>
<!-- 模板抽离出来 -->
<template id="home">
  <!-- 注意：组件只能有一个根元素，所以我们包装到这个div中 -->
  <div>
    <h2>首页</h2>
    <router-link to="/home/login">登录</router-link>
    <router-link to="/home/reg">注册</router-link>
    <!-- 路由匹配到的组件将渲染在这里 -->
    <router-view></router-view>
  </div>
</template>
<template id="news">
  <div>新闻</div>
</template>
<template id="login">
  <div>登录界面</div>
</template>
<template id="reg">
  <div>注册界面</div>
</template>

```

```

<script type="text/javascript">
  // 1. 定义（路由）组件。
  const Home = { template: '#home' };
  const News = { template: '#news' };
  const Login = { template: '#login' };
  const Reg = { template: '#reg' };
  // 2. 定义路由
  const routes = [
    { path: '/', redirect: '/home' },
    {
      path: '/home',
      component: Home,
      children: [
        { path: '/home/login', component: Login },
        { path: '/home/reg', component: Reg }
      ]
    },
    { path: '/news', component: News }
  ]
  // 3. 创建 router 实例，然后传 `routes` 配置
  const router = new VueRouter({
    routes // (缩写) 相当于 routes: routes
  })
  // 4. 创建和挂载根实例。 记得要通过 router 配置参数注入路由，从而让整个应用都有路由功能
  const app = new Vue({
    router
  }).$mount('#box')

```

vue-Router 编程式导航：

主要利用 `<router-link>`来创建可跳转链接，还可以在方法里利用 `this.$router.push('xxx')` 来进行跳转。

```
<template>
  <div rel="headPage" :class="fixed? 'fixed':''">
    <div v-for="(item, index) in navList" :key="index"
      class="nav cursor"
      :class="matchTab(item.hash)"
      @click="handleSelect(item,index)"
    >{{item.text}}</div>
  </div>
</template>
```

```
handleSelect(item, index){
  // this.activeIndex = index;
  console.log(item);
  this.$router.push({
    path: item.hash,
    hash: item.hash
  }, () => {
    setTimeout(() => {
      document.documentElement.scrollTop = this.getDocumentTop(item.hash.slice(1));
    }, 100);
  });
}
```

vue-Router 路由对象属性：

vue 路由对象为 `this.$route`，下面详细列一下该对象属性的详细信息

属性名	类型	读写	说明
<code>\$route.path</code>	string	只读	当前路由的名字(一般为#后面的部分,但不包含query查询值) 如：http://example.com/#/login?name=aa <code>this.\$route.path</code> ; //输出“/login”
<code>\$route.query</code>	object	只读	可访问携带的查询参数 如：this.\$router.push({name: 'login', query:{name: 'you'}}) 此时路由为: http://example.com/#/login?name=you 可直接访问this.\$route.query.name; //you
<code>\$route.params</code>	object	只读	路由跳转携带参数 如：this.\$route.push({name: 'hello', params: {name: 'you'}}) 此时可访问this.\$route.params.name; //you
<code>\$route.hash</code>	string	只读	当前路径的哈希值，带#
<code>\$route.fullPath</code>	string	只读	完整的路径值 如：http://example.com/#/login?name=aa <code>this.\$route.fullPath</code> ; //输出“/login?name=aa”
<code>\$route.name</code>	string	只读	命名路由的
<code>\$route.matched</code>	array	只读	当前路由由路由声明的所有信息，从父路由(如果有)到当前路由为止
<code>\$route.redirectedFrom</code>	string	只读	重定向来源 如：{ path: '*', redirect: {name: 'hello'}} 此时访问不存在的路由http://example.com/#/a会重定向到hello 在hello访问this.\$route.redirectedFrom; //输出“/a”

vue-Router 的核心点：

## 1. vue-router 如何参数传递

### ① 用 name 传递参数

在路由文件 src/router/index.js 里配置 name 属性

然后通过 \$router.name 来获取

### ② 通过 <router-link> 标签中的 to 传参

这种传参方法的基本语法：

```
<router-link :to="{name:xxx,params:{key:value}}">value</router-link>
```

比如先在 src/App.vue 文件中

```
<router-link :to="{name:'hi1',params:{username:'jspang', id:'555'}}">hello  
</router-link>
```

然后把 src/router/index.js 文件里给 hi1 配置的路由起个 name, 就叫 hi1.

```
{path:'/hi1',name:'hi1',component:Hi1}
```

最后在模板里 (src/components/Hi1.vue) 用 \$route.params.username 进行接收. {{ \$route.params.username }}-{{ \$route.params.id }}

### ③ vue-router 利用 url 传递参数----在配置文件里以冒号的形式设置参数。

我们在 /src/router/index.js 文件里配置路由

```
{  
  
  path:'/params/:newsId/:newsTitle',  
  
  component:Params  
  
}
```

我们需要传递参数是新闻 ID (newsId) 和新闻标题 (newsTitle). 所以我们在路由配置文件里制定了这两个值。

在 src/components 目录下建立我们 params.vue 组件, 也可以说是页面。我们在页面里输出了 url 传递的新闻 ID 和新闻标题。

通过`$router.params.newsId`和`$.router.params.newsTitle` 获取

## 2. 单页面多路由区域操作

在一个页面里我们有 2 个以上`<router-view>`区域，我们通过配置路由的 js 文件，来操作这些区域的内容

## 3. vue-Router 配置子路由（二级路由）

## 4. vue-router 跳转方法

```
<button @click="goToMenu" class="btn btn-success">Let's order! </button>
.....
<script>
  export default{
    methods:{
      goToMenu(){
        this.$router.go(-1)//跳转到上一次浏览的页面
        this.$router.replace('/menu')//指定跳转的地址
        this.$router.replace({name: 'menuLink'})// 指定跳转路由的名字下
        this.$router.push('/menu')通过push进行跳转
        this.$router.push({name: 'menuLink'})通过push进行跳转路由的名字下
      }
    }
  }
</script>
```

## 5. 404 页面的设置

用户会经常输错页面，当用户输错页面时，我们希望给他一个友好的提示页面，这个页面就是我们常说的 404 页面。vue-router 也为我们提供了这样的机制。

①设置我们的路由配置文件（`/src/router/index.js`）

```
path: '*',
component: Error
```

③ 在`/src/components/`文件夹下新建一个 `Error.vue` 的文件。简单输入一些有关错误页面的内容。

```

1   <template>
2     <div>
3       <h2>{{ msg }}</h2>
4     </div>
5   </template>
6   <script>
7     export default {
8       data () {
9         return {
10           msg: 'Error:404'
11         }
12       }
13     }
14   </script>

```

vue 的\$route 和\$router 的区别:

## 1. \$route 是一个对象

```

▼ $route: Object
  fullPath: "/home"
  hash: ""
  ▶ matched: [{...}]
  ▶ meta: {}
  name: "Home"
  ▶ params: {}
  path: "/home"
  ▶ query: {}
  ▶ __proto__: Object

```

可以获取当前页面的路由的路径query、params、meta等参数;

## 2. \$router 是 vue-Router 的一个实例对象

```

▼ $router: VueRouter
  ▶ afterHooks: []
  ▶ app: Vue {__uid: 0, _isVue: true, $options: {...}, _renderProxy: Proxy, _self: Vue, ...}
  ▶ apps: [Vue]
  ▶ beforeHooks: []
  fallback: false
  ▶ history: HTML5History {router: VueRouter, base: "", current: {...}, pending: null, ready: true, ...}
  ▶ matcher: {match: f, addRoutes: f}
  mode: "history"
  ▶ options: {mode: "history", routes: Array(6)}
  ▶ resolveHooks: []
  currentRoute: (...)
  ▶ __proto__: Object

```

在 options 中可以获取路由的 routes 配置参数。想要导航到不同 URL，则使用 \$router.push 方法



