vue-router的理解

安装

初始化安装：

在使用脚手架的时候自动安装

手动安装：

>npm install vue-router / yarn add vue-router

手动安装vue-router之后使用步骤：

第一步：import引入安装文件

代码：import VueRouter from 'vue-router';

作用：把VueRouter作为插件注册到Vue实例中，只有做了这步操作以后，才能在Vue实例中使用 <router-view><router-link> 等一些 vue-router 中的内置的组件

第二步：use加到vue原型链上

代码：Vue.use(VueRouter);

第三步：创建VueRouter对象，配置路由——一个对象就是一个路由

代码：const router = new VueRouter({

routes: [

// 每一个对象就是一个路由，url=>component，配置url与组件之间的关系

{}

]

});

第四步：把创建的VueRouter对象作为配置赋值给Vue实例

代码：new Vue({

el: '',

//...

// router: new VueRouter()

// router: router

router

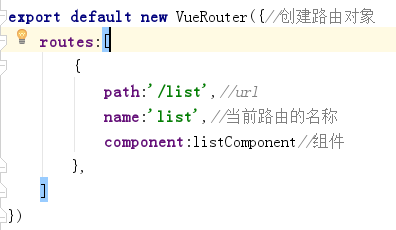
});

第五步：页面标签router-view

原理：根据url映射关系找到组件，将组件替换掉router-view标签。

clipboard.png

clipboard.png





clipboard.png

vue-router 内置组件

router-view

作用：显示组件（根据url显示组件）

router-link

作用：生成a标签

注意：跳转不要使用原生的a直接去写，而是要用router-link来生成，因为router-link生成的a标签是被处理过的，vue-router处理的a的点击，阻止了a的默认跳转，使用的是更改history或hash来实现url变化

vue实现选项卡的方法：

方法一：tab切换

思路：

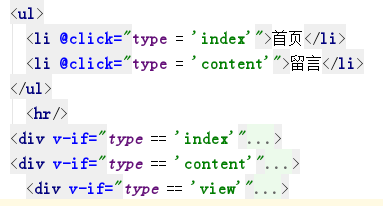
3个按钮，3个内容页

点击不同的按钮，将type改变，不同的type显示不同的内容页面

存在的问题：

页面复制，就需要比较多type。

如果选项卡套选项卡套选项卡，这样子操作就不方便





方法二：vue-router

思路：根据不同的url显示不同的内容,就是显示不同的组件

不同路由的形式

hash #。兼容低版本——/#/view/3

history /。高版本，SEO，美观——/view/1

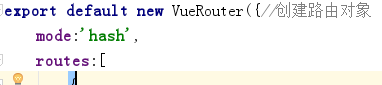
实现：默认情况下是hash，在路由配置参数里面加mode

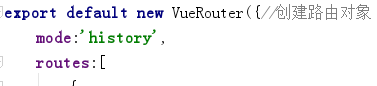
注意：path不能用params和query传输，要在紧跟path后面加数据参数

**:to="**{**path**:**'/view/'**+***item***.**id**+**''**}**"**

**:to="**{**name**:**'view'**,**params**:{**id**:***item***.**id**}}**"**

如果用route-link的话，就需要





**组件获取数据的方式：**

**组件通信**：

props：父组件传递，组件与组件需要有嵌套的包含关系

平级组件有相同的公用数据，可以把数据放到他们共同的父级上面

**url**：query或params

this.**$router**.push(name，query或params)

跳转：

router-link

this.$router.push(name:,query/params)

导航守卫参数next({path: '/login',query: { redirect: to.fullPath }})

**query或params实现动态路由**

理解：同一个路由，映射到同一个组件，通过params和query传入的数据不同，显示不同的内容。

**params**

路由

传输

处理接收的数据

clipboard.png

clipboard.png

clipboard.png

clipboard.png

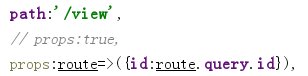
**query**

路由

传输

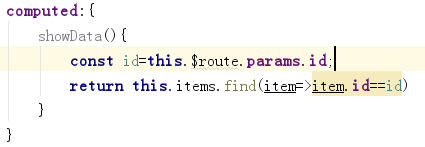
处理接收的数据

clipboard.png



clipboard.png

clipboard.png



高级匹配模式：正则的方式

比如：

clipboard.png

**组件通信props**

非路由组件——通过父组件的属性传输给props

路由组件——通过路由配置参数初始化props

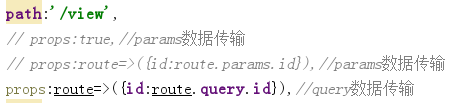
props:true..

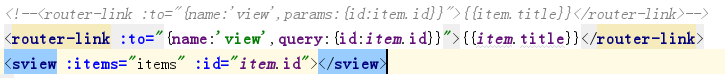
原理：将params里面的数据放到props同名的数据。params会自动处理

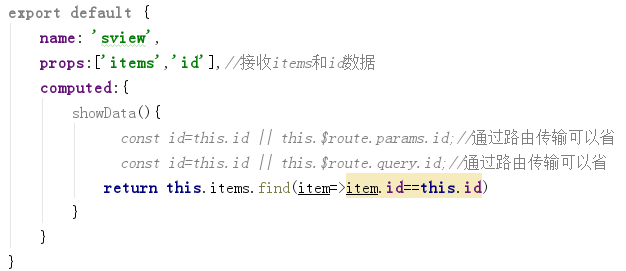
query要用函数式的——props:fn(return{}).和组件里面的props合并

props: route => ({id: route.query.id}),

原理：一个组件可以是路由组件和非路由组件，非路由可以通过接收的数据从父**标签属性发送**，路由可以通过设置路由配置参数里面的**路由props属性发送**。获取数据，在组件内容用props接收属性数据，并且在computed计算属性中可以省略两种传递过来的数据的合并。







**this.$route和this.$router的理解**

**this**.$route

当前路由对象，当前路由有关的信息

应用：获取传输的数据

例子：

**this**.$route.**params**.**id**;

*this.$route.query.id;*

**this**.$router

整个应用的路由routes信息

应用：实现跳转——编程式的导航

理解：页面跳转可以通过a标签，也可以通过js来实现。实现的原理是window的history来实现的

例子：

**this**.$router.push(**'/'**)，会生成历史记录

**this**.$router.push({ **name**: **'user'**, **params**: { **id**: 1 }})

**this**.$router.push({ **name**: **'user'**, **query**: { **id**: 1 }})

**this**.$router.go(n)

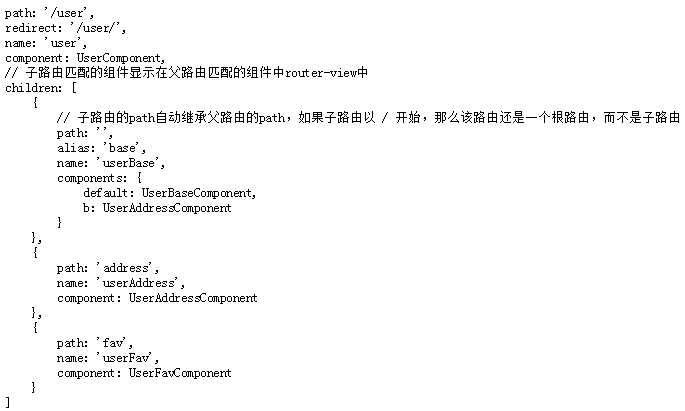
**this**.$router.replace({ **name**: **'user'**})，替换当前的历史记录

**嵌套路由的理解**

路由组件里面嵌套路由组件

子路由匹配的组件显示在父路由匹配的组件中router-view中

注意：子路由的path自动继承父路由的path，如果子路由以 / 开始，那么该路由还是一个根路由，而不是子路由



重定向和别名

可以写在路由对象配置参数中的每一个route中。

redirect——重定向，url改变。。冲a跳到b——应用，看有没有登入，没登入跳到登录

alias:——别名，url不改变.还是在a。。。。。不同名字访问同一个地址

命名路由

给不同的route去不同的name，可以通过访问name，访问相应的组件。

命名视图

视图即组件。就是给不同的组件起名字。

同一个路由可以匹配不同的组件，将不同的组件放到components下面，给不同的组件去不同的名字，在router-view标签的name属性指定相应的名字。

<router-view class="view three" name="b"></router-view>

<router-view class="view three"></router-view>//不给名字的就是默认组件

components: {

b: Baz，

default:adb

}

导航守卫

全局——给路由实例加

router.beforeEach（(to, from, next)）——当一个导航触发时

router.beforeResolve（(to, from, next)）——解析守卫

router.afterEach（(to, from)）

路由——给路由配置参数加

beforeEnter（(to, from, next)）

组件内部——组件内部加

beforeRouteEnter (to, from, next)

在渲染该组件的对应路由被 confirm 前调用

不！能！获取组件实例 `this`

因为当守卫执行前，组件实例还没被创建

beforeRouteUpdate (to, from, next)

在当前路由改变，但是该组件被复用时调用

举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

可以访问组件实例 `this`

beforeRouteLeave (to, from, next)

导航离开该组件的对应路由时调用

可以访问组件实例 `this`

注意：

next('/') 或者 next({ path: '/' })

确保要调用 next 方法，否则钩子就不会被 resolved。

完整的导航解析流程

导航被触发。

在失活的组件里调用离开守卫。 beforeRouteLeave

调用全局的 beforeEach 守卫，触发导航。

在重用的组件里调用 beforeRouteUpdate 守卫 (2.2+)。

在路由配置里调用 beforeEnter。

解析异步路由组件。

在被激活的组件里调用 beforeRouteEnter，不能读取this。

调用全局的 beforeResolve 守卫，解析导航。

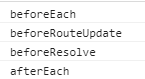
导航被确认。

调用全局的 afterEach 钩子。

触发 DOM 更新。

用创建好的实例调用 beforeRouteEnter 守卫中传给 next 的回调函数，可以读取this。

更新：



进入另一个：

