

vue-router前端路由及单页面开发

为什么需要前端路由？

在以前nodejs里面，我们学习MVC的框架 `express` 的时候，里面是有路由的，我们当时对路由的是理解是**根据 不同的路径处理不同的数据请求，然后展示不同的页面或返回不同的数据**。这个时候的路由是在后端处理的，所以如下

```
router.get("/login");
router.post("/addStu");
```

在上面，我们有两个路由地址，这两个路由地址在这个地址可以处理不同的请求（或者我们可以认为它是返回不同的页面）

现在的问题来了，我们现在是换了另外一咱开发模式（前后端分离开发），这个时候后端已经不再处理页面的问题了，那么前端又是如何实现页面的开发呢？

要解决上面的前后端分离式开发的问题，我们就必须要知道**前端路由**

什么是前端路由？

前端路由就是在浏览器端改变浏览器的地址以实现页面的切换，而现在的前后端分离式开发里面提供的是单页面开发，所以我们在一个页面上去完成页面切换

什么是SPA？

SPA全称是单页面开发，所有的界面展示与数据处理都在一个页面上完成，这样界面在切换的时候速度会非常快

手动模块前端路由的实现

前端路由是通过改变浏览器 `URL` 地址后面的 `#` 来实现的，所以我们现在尝试进行这样的操作

```
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>模拟SPA前端路由</title>
  <style>
    * {
      margin: 0;
      padding: 0;
    }

    #app {
      width: 100vw;
      height: 100vh;
    }

    #login {
      background-color: lightblue;
    }
  </style>
</head>
<body>
  <div id="app">
    <div id="login">
      <h1>登录</h1>
    </div>
  </div>
</body>
</html>
```

```

    #register {
      background-color: lightcoral;
    }

    #app>div {
      width: 100%;
      height: 100%;
      display: none;
    }
  </style>
</head>

<body>
  <div id="app">
    <div id="login">
      <h2>这是一个登陆页面</h2>
      <button type="button" onclick="toRegister()">我要去注册页面了</button>
    </div>
    <div id="register">
      <h2>这是一个注册页面-----哈哈</h2>
      <button type="button" onclick="toLogin()">我要去登陆页面</button>
    </div>
  </div>
</body>
<script>
  //改变url地址后面的东西无非就是改变hash值，所以我们只要监控这个hash的值改变就可以了
  window.onhashchange = function (event) {
    let hash = location.hash;
    document.querySelectorAll("#app>*").forEach(item => {
      item.style.display = "none";
    });
    document.querySelector(hash).style.display = "block";
  }

  function toRegister(){
    location.hash = "#register";
  }
  function toLogin(){
    location.hash = "#login";
  }
</script>
</html>

```

通过上面的手动实现方式，其实我们可以看一看到，前端路由页面的改变其实是通过改变了 URL 地址后面的 hash 值来实现的，所以后面的大多数前端框架的前端路由都是通过这种方式来实现的

同时上面这个案例也是模拟了在 1 个页面上进行 2 个页面的操作

vue-router 前端路由

在我们之前学习的 `vue` 的框架里面，它也可以实现前端路由的，只是它是一个单独的包，需要另外加载到 `vue` 当中去，我们可以通过 <https://router.vuejs.org/zh/> 去进行下载

Vue Router 是 [Vue.js](#) 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

- 嵌套的路由/视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- 基于 Vue.js 过渡系统的视图过渡效果
- 细粒度的导航控制
- 带有自动激活的 CSS class 的链接
- HTML5 历史模式或 hash 模式，在 IE9 中自动降级
- 自定义的滚动条行为

现在开始[起步](#)或尝试一下我们的[示例](#)吧 (查看仓库的 [README.md](#) 来运行它们)。

安装

```
$ npm install vue-router -S
```

使用与配置

vue是基于组件来进行单页面控制的框架，所以我们必须要先创建组件

```
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vue-router前端路由的使用</title>
  <style>
    .login {
      color: red;
    }
    .register {
      color: lightseagreen;
    }
  </style>
</head>

<body>
  <div id="app">
    <!-- 这个地方应该根据不同的前端路由地址去实现不同组件的加载 -->
    <router-view></router-view>
  </div>
  <template id="temp1">
    <div class="login">
      <h2>这是一个登陆的页面</h2>
      <router-link :to="{name: 'register'}">通过链接去【注册页面】</router-
link>
    </div>
  </template>
  <template id="temp2">
    <div class="register">
      <h2>这又是一个注册滴页面-----</h2>
      <router-link to="/login">通过链接去【登录页面】</router-link>
    </div>
  </template>
</body>
<script src="./js/vue.js"></script>
```

```

<script src="./js/vue-router.js"></script>
<script>
  // 定义两个局部组件
  let login = {
    template: "#temp1"
  }
  let register = {
    template: "#temp2"
  }
  //创建vue-router的管理对象
  let router = new VueRouter({
    mode: "hash",
    // 这一个属性包含了当前这个管理对象下面的所有路由
    routes: [{
      path: "/",
      redirect: "/login"
    }, {
      path: "/login",
      component: login,
      name: "login"
    }, {
      path: "/register",
      component: register,
      name: "register"
    }]
  });
  new Vue({
    el: "#app",
    //加载之前的路由管理对象
    router
  })
</script>
</html>

```

代码分析：

1. `vue-router` 是基于组件控制的，所以我们在上面的代码不如创建2个组件，分别是 `login` 与 `register` 组件
2. 要创建路由管理对象 `new VueRouter`，在这里面它接收参数，其中 `mode="hash"` 代表当前路由的模式，它还有一种模式叫 `mode="history"` 目前暂不支持
3. 路由管理对象下面有 `routes` 属性，它是一个数组，里面包含了每个路由对象
4. 在 `routes` 的每个路由对象下面，`path` 代表路由，`component` 代表要加载的组件，`name` 代表这个路由对象的名称
5. 在要显示组件的方添加 `<router-view>`，它的作用是相当于一个占位符，后期根据不同的前路由的路径去加载不同的组件
6. `<router-link>` 是进行路由跳转的时候使用的，它有一个 `to` 的属性用于设置要跳转的地址
7. 一定一定要记得，所创建的路由管理对象 `router` 一定要在 `Vue` 里面加载

注意区分： `router` 称之为路由管理对象，`route` 称之为路由对象

vue的路由管理对象

在上个案例当中，我们已经通过 `new VueRouter` 生成了一个路由管理对象 `router`，这个对象在每个 `vue` 的内部（或组件内部）都可以通过 `this.$router` 得到，得到这个对象以后，我们就可以对当前的路由进行管理。所以 `this.$router` 称之为路由管理对象

这个路由管理对象上面具备一些属性与方法，我现在截图如下

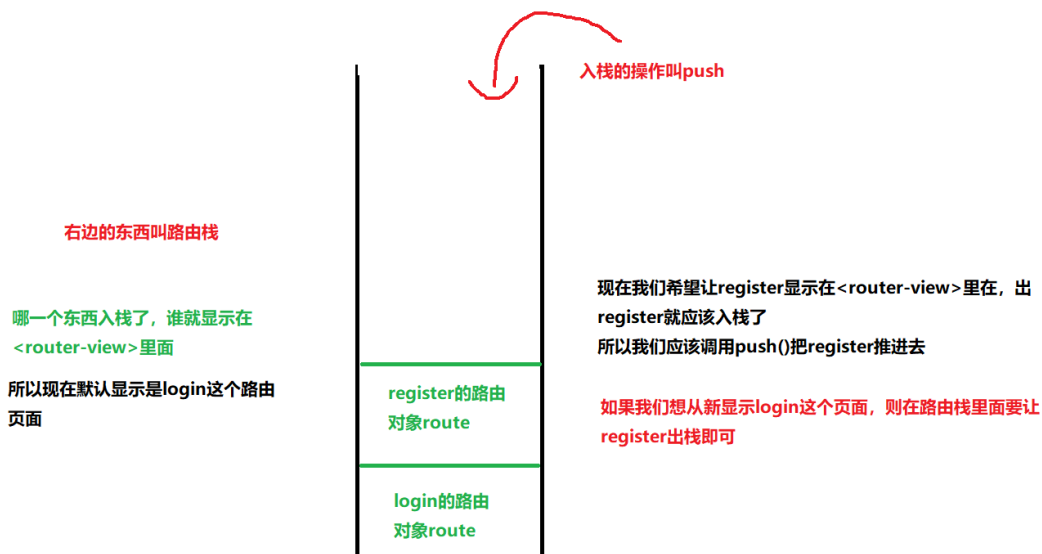
当前对象	原型链上面
<pre>▼ \$router: VueRouter ▶ afterHooks: [] ▶ app: Vue { _uid: 0, _isVue: true, \$options: {...}, _renderProxy: Proxy, _self: Vue, ...} ▶ apps: [Vue] ▶ beforeHooks: [] ▶ fallback: false ▶ history: HashHistory { router: VueRouter, base: "", current: {...}, pending: null, ready: true, ...} ▶ matcher: { match: f, addRoute: f, getRoutes: f, addRoutes: f } mode: "hash" ▶ options: { mode: "hash", routes: Array(3) } ▶ resolveHooks: [] ▶ currentRoute: Object</pre>	<pre>▼ __proto__: ▶ addRoute: f addRoute(parentOrRoute, route) ▶ addRoutes: f addRoutes(routes) ▶ afterEach: f afterEach(fn) ▶ back: f back() ▶ beforeEach: f beforeEach(fn) ▶ beforeResolve: f beforeResolve(fn) ▶ forward: f forward() ▶ getMatchedComponents: f getMatchedComponents(to) ▶ getRoutes: f getRoutes() ▶ go: f go(n) ▶ init: f init(app /* Vue component instance */) ▶ match: f match(raw, current, redirectedFrom) ▶ onError: f onError(errorCb) ▶ onReady: f onReady(cb, errorCb) ▶ push: f push(location, onComplete, onAbort) ▶ replace: f replace(location, onComplete, onAbort) ▶ resolve: f resolve(to, current, append) ▶ constructor: f VueRouter(options) ▶ currentRoute: (...) ▶ get currentRoute: f () ▶ __proto__: Object</pre>

在路由管理对象上面有几个特殊的属性与方法我们列举一下

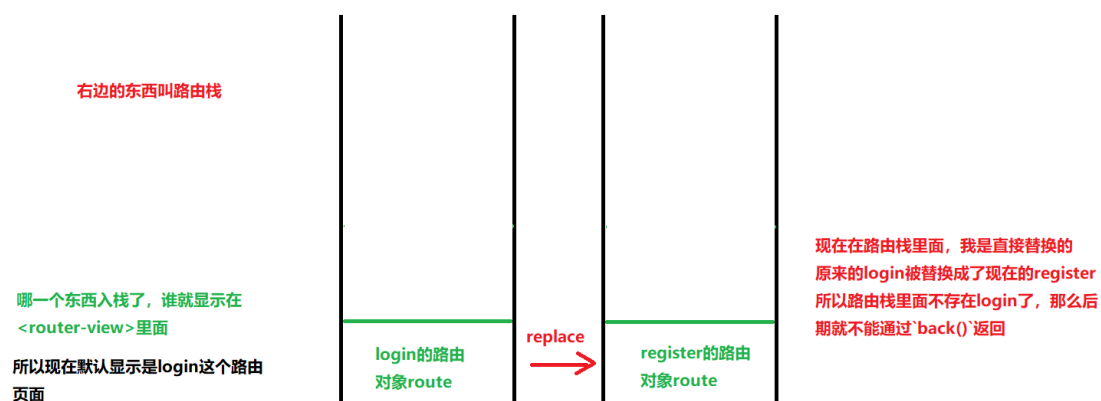
1. `back()` 使当前的前端路由后退，本质上面就是在当前路由栈里面让这个路由对象出栈
2. `forward()` 使当前的前端路由前进
3. `go(n)` 通过参数来前进或后退，如果是正数则前进，如果是负数则后退
4. `push()` 进入到一个新的路由，本质上面是在路由栈里面推入了一个新的路由对象
5. `replace()` 替换一个路由，可以理解为之前BOM里面的 `location.replace()`

vue的路由管理对象 `$router` 到底管理的是什么呢？本质上面它管理的是一个叫做路由栈的东西

🚩 路由对象的入栈与出栈操作



🚩 路由对象的 replace 操作



所以通过这个可以看到 `replace` 与 `push` 是有本质的区别的，同时也更进一步的说明路由管理对象是通过栈的原理来实现管理的

```
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vue-router前端路由的使用</title>
  <style>
    .login {
      color: red;
    }

    .register {
      color: lightseagreen;
    }
  </style>
</head>

<body>
  <div id="app">
    <!-- 这个地方应该根据不同的前端路由地址去实现不同组件的加载 -->
    <router-view></router-view>
  </div>
  <template id="temp1">
    <div class="login">
      <h2>这是一个登陆的页面</h2>
      <router-link :to="{name:'register'}">通过链接去【注册页面】</router-
link>
      <button type="button" @click="toRegister">通过按钮去【注册页面】
</button>
      <button type="button" @click="replaceToRegister">通过replace的方式去
【注册页面】</button>
    </div>
  </template>
  <template id="temp2">
    <div class="register">
      <h2>这又是一个注册滴页面-----</h2>
      <router-link to="/login">通过链接去【登录页面】</router-link>
      <button type="button" @click="toLogin">通过按钮返回【登录页面】</button>
    </div>
  </template>
</body>
<script src="./js/vue.js"></script>
<script src="./js/vue-router.js"></script>
<script>
  // 定义两个局部组件
  let login = {
    template: "#temp1",
    methods: {
      toRegister() {
        //让register这个路由对象入栈，关键是怎么找到这个register这个路由对象呢
        this.$router.push({
          name: "register"
        })
      }
    }
  }

  let register = {
    template: "#temp2",
    methods: {
      toLogin() {
        //让login这个路由对象入栈，关键是怎么找到这个login这个路由对象呢
        this.$router.push({
          name: "login"
        })
      }
    }
  }

  // 注册路由
  const routes = [
    {
      path: "/login",
      name: "login",
      component: login
    },
    {
      path: "/register",
      name: "register",
      component: register
    }
  ]

  const router = new VueRouter({
    routes
  })

  // 挂载到app上
  new Vue({
    router
  }).$mount("#app")
</script>
```

```

        });
    },
    replaceToRegister(){
        // 让register这个路由对象去替换之前的路由对象
        this.$router.replace({
            name: "register"
        });
    }
}
}
let register = {
    template: "#temp2",
    methods:{
        toLogin(){
            //返回到之前的login页面，这个时候必须让刚刚入栈的register出栈
            // back()本质上面就是退出当前的路由对象
            this.$router.back();
        }
    }
}
}
//创建vue-router的管理对象
let router = new VueRouter({
    mode: "hash",
    // 这一个属性包含了当前这个管理对象下面的所有路由
    routes: [{
        path: "/",
        redirect: "/login"
    }, {
        path: "/login",
        component: login,
        name: "login"
    }, {
        path: "/register",
        component: register,
        name: "register"
    }
    ]
});
new Vue({
    el: "#app",
    router
})
</script>
</html>

```

有了这个路由管理对象的方法以后，我们后期在开发过来当中跳转前端路由既可以使用 `<router-link>` 的链接试，还可以通过 `javascript` 的这种方式来跳转

前端路由传值

前端路由控制与模拟了我们之前的页面跳转，而在页面跳转的时候经常需要进行跨页面的传值，所以 `vue-router` 也提供了跨路由的传值，它的传值方式也有两种，现列举如下

1. 通过 `query` 的方式传值

这种传值方式模拟了 `get` 传值，它会在地址栏后面添加？

2. 通过 `params` 的方式传值

通过query方式传值

```
<body>
  <div id="app">
    <router-view></router-view>
  </div>
  <template id="temp1">
    <div>
      <h2>你正处于登陆页面</h2>
      <input type="text" placeholder="请输入登陆的账号" v-model="userName">
      <input type="text" placeholder="请输入年龄" v-model="age">
      <button type="button" @click="checkLogin">登录</button>
    </div>
  </template>
  <template id="temp2">
    <div>
      <h2>这是登陆以后的系统主页</h2>
      <h2>你的账号为: {{userName}},你的年龄为: {{age}}</h2>
    </div>
  </template>
</body>
<script src="./js/vue.js"></script>
<script src="./js/vue-router.js"></script>
<script>
  let login = {
    template: "#temp1",
    data() {
      return {
        userName: "",
        age: 18
      }
    },
    methods: {
      checkLogin() {
        this.$router.replace({
          name: "index",
          //在这个地方，我们写了一个query`
          query: {
            userName: this.userName,
            age: this.age
          }
        })
      }
    }
  }

  let index = {
    template: "#temp2",
    data() {
      return {
        userName: "",
        age: ""
      }
    },
    created() {
      //在这里我们要找路由对象，这个路由对象上面记录了你当前的路由信息

      //在路由对象上面我们拿到了传递过来的值
    }
  }

```



```

        this.userName = this.$route.query.userName;
        this.age = this.$route.query.age
    },

    }

    // 创建路由管理对象
    let router = new VueRouter({
        routes: [{
            path: "/",
            redirect: {
                name: "login"
            }
        }, {
            path: "/login",
            component: login,
            name: "login"
        }, {
            path: "/index",
            component: index,
            name: "index"
        }
    ]
    })

    new Vue({
        el: "#app",
        router
    })
</script>

```

代码说明：

1. 第一步：使用 query 进行传递

```

checkLogin() {
    this.$router.replace({
        name: "index",
        //在这个地方，我们写了一个query`
        query: {
            userName: this.userName,
            age: this.age
        }
    })
}

```

2. 第二步：在新的路由当中使用 query 进行接收

```

created() {
    //在这里我们要找路由对象，这个路由对象上面记录了你当前的路由信息
    this.userName = this.$route.query.userName;
    this.age = this.$route.query.age
},

```

3. 第三步：分析url地址

```
http://127.0.0.1:5500/04.html#/index?userName=biaogege&age=22
```

这个时候我们可以看到在新地地址栏上面是有 ? 存在的, ? 后面的则是代表了参数

通过params来进行传值

这是一传值方式相当于我们以前在 `express` 的框架中进行的 **路径变量** 传值

```
//这是后端路由
router.get("/checkLogin/:sid/:pwd");

//接收参数
req.params.sid
req.params.pwd
```

现在我们来完成这个操作过程

第一步：将原来的 `query` 变成 `params`

```
checkLogin() {
  this.$router.replace({
    name: "index",
    params: {
      userName: this.userName,
      age: this.age
    }
  })
}
```

第二步：修改路由信息

```
let router = new VueRouter({
  routes: [{
    path: "/",
    redirect: {
      name: "login"
    }
  }, {
    path: "/login",
    component: login,
    name: "login"
  }, {
    //关键应在这里，我们要传递的数据放在了路径里面
    path: "/index/:userName/:age",
    component: index,
    name: "index"
  }
]}
})
```

第三步：在新的路由界面的组件里面去接收值

```
created() {
  //在这里我们要找路由对象，这个路由对象上面记录了你当前的路由信息
  this.userName = this.$route.params.userName;
  this.age = this.$route.params.age;
},
```

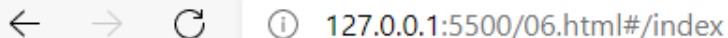
第四步：看生成的url地址

```
http://127.0.0.1:5500/05.html#/index/biaogege/123
```

它传递过来的的 `userName` 就是 `biaogege`，传递的 `age` 则是 `123`

通过上面的对比，我们其实发现一点无论使用哪一种方式进行传值，最终这个数据都会暴露在地址栏，所以还不安全的，但是 `vue` 在 `params` 的方式上面有一种特殊的使用方式，可以不用将信息暴露在地址栏

这个地方只用做一点，就是去掉上面的第二步操作，这个时候再去操作的时候，我们仍然可以取到值



这是登陆以后的系统主页

你的账号为：biaogege,你的年龄为：123123

通过上面的图片我们已经发现，浏览器的地址栏里面已经没有传递的值了

但是这么做有一个非常大的缺点，就是这种传值称之为状态传值，不记录当前传值的状态，网页不允许刷新，如果刷新这个值就会消失

这么做的好处还是有几点的

1. 数据不会显示在浏览器的地址栏，所以对于机密数据可以得到保护
2. 数据不用显示地址栏，所以不用受浏览器地址栏字符串多少的限制
3. 这不保存你之前传递的值的状态，刷新以后即时销毁这个值

嵌套路由

嵌套路由是在原有的组件里面再去嵌套一个路由，这样可以实现一个组件的某一个地方更新

```
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>嵌套路由</title>
</head>

<body>
  <div id="app">
    <!-- 在这里，我们要根据不同的路径去显示不同的组件 -->
    <router-view></router-view>
  </div>
  <template id="temp1">
    <div>
      <h2>这是第一个组件</h2>
      <router-link :to="{name: 'c2'}">去第二个组件</router-link>
    </div>
  </template>
</body>
</html>
```

```

    </div>
  </template>
  <template id="temp2">
    <div>
      <h2>这是第二个组件</h2>
      <router-link :to="{name: 'd1'}">去d1</router-link>
      <router-link :to="{name: 'd2'}">去d2</router-link>
      <hr>
      <!-- 在这里，我们要根据路径去显示不同的组件 -->
      <router-view></router-view>
    </div>
  </template>
  <template id="temp3">
    <div>
      <h3>哈哈，我是第1个子缓步件</h3>
    </div>
  </template>
  <template id="temp4">
    <div>
      <h3>哈哈，我又是一个小的子组件，我是第2个了</h3>
    </div>
  </template>
</body>
<script src="./js/vue.js"></script>
<script src="./js/vue-router.js"></script>
<script>
  let c1 = {
    template: "#temp1"
  };
  let c2 = {
    template: "#temp2"
  };
  let d1 = {
    template: "#temp3"
  };
  let d2 = {
    template: "#temp4"
  }
  let router = new VueRouter({
    mode: "hash",
    routes: [{
      path: "/",
      redirect: {
        name: "c1"
      }
    },
    {
      path: "/c1",
      component: c1,
      name: "c1"
    },
    {
      path: "/c2",
      component: c2,
      name: "c2",
      children: [
        {
          path: "d1",
          name: "d1",

```

```

        component:d1
      },{
        path:"d2",
        name:"d2",
        component:d2
      }
    ]
  }
})
new Vue({
  el: "#app",
  router
})
</script>

</html>

```

上面的案例当中，我们就实现了嵌套路由

代码分析：

1. 嵌套路由里面的路由对象与普通的路由对象保持一致，只是在 `path` 这个属性下面没有 `/` 了，它会自动的在你的父级路径下面形成一个子级路径
2. 嵌套路由的配置是在某一个路由对象下面的 `children` 下面去设置
3. 一定要在的嵌套的地方去添加 `<router-view>`

分析嵌套路由形成的路径：

- 一级路由所生成的路径

```
http://127.0.0.1:5500/01.html#/c1
```

```
http://127.0.0.1:5500/01.html#/c2
```

- 二级路由所生成的路径

```
http://127.0.0.1:5500/01.html#/c2/d1
```

```
http://127.0.0.1:5500/01.html#/c2/d2
```

嵌套路由案例

三栏式布局案例

```

<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vue嵌套路由典型案例</title>
  <style>
    * {
      margin: 0;
    }
  </style>

```

```

        padding: 0;
        list-style-type: none;
    }

    #app {
        width: 100vw;
        height: 100vh;
    }

    .login-box {
        width: 100%;
        height: 100%;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
    }

    .admin-box {
        width: 100%;
        height: 100%;
        position: fixed;
    }

    .top {
        height: 100px;
        border-bottom: 1px solid lightgray;
        box-sizing: border-box;
        display: flex;
        justify-content: center;
        align-items: center;
        font-size: 32px;
        font-weight: bold;
    }

    .left-menu {
        position: absolute;
        left: 0px;
        width: 270px;
        border-right: 1px solid lightgray;
        box-sizing: border-box;
        top: 100px;
        bottom: 0px;
    }

    .right-content {
        position: absolute;
        top: 100px;
        bottom: 0px;
        left: 270px;
        right: 0px;
    }
}

</style>
</head>

<body>
    <div id="app">
        <router-view></router-view>
    </div>
</body>

```

```

</div>
<template id="loginTemp">
  <div class="login-box">
    <p>
      用户名: <input type="text">
    </p>
    <p>
      密码: <input type="text">
    </p>
    <p>
      <button type="button" @click="checkLogin">登录</button>
    </p>
  </div>
</template>
<template id="adminIndexTemp">
  <div class="admin-box">
    <div class="top">欢迎使用学生管理系统</div>
    <div class="left-menu">
      <p>
        <router-link :to="{name:'addStu'}">新增学生</router-link>
      </p>
      <p>
        <router-link :to="{name:'stuInfoList'}">学生列表</router-
link>
      </p>
    </div>
    <div class="right-content">
      <router-view></router-view>
    </div>
  </div>
</template>
<template id="addStuTemp">
  <div>
    <h2>这是一个新增学生的界面</h2>
  </div>
</template>
<template id="stuInfoListTemp">
  <div>
    <h2>这是一个学生信息列表的展示界面-----</h2>
  </div>
</template>
</body>
<script src="./js/vue.js"></script>
<script src="./js/vue-router.js"></script>
<script>
  let login = {
    template: "#loginTemp",
    methods: {
      checkLogin() {
        this.$router.replace({
          name: "adminIndex"
        });
      }
    }
  }

  let adminIndex = {
    template: "#adminIndexTemp"
  }

```

```

let addStu = {
  template: "#addStuTemp"
};
let stuInfoList = {
  template: "#stuInfoListTemp"
};
let router = new VueRouter({
  mode: "hash",
  routes: [{
    path: "/",
    redirect: {
      name: "login"
    }
  }, {
    path: "/login",
    component: login,
    name: "login"
  }, {
    path: "/adminIndex",
    component: adminIndex,
    name: "adminIndex",
    children: [{
      path: "addStu",
      component: addStu,
      name: "addStu"
    }, {
      path: "stuInfoList",
      component: stuInfoList,
      name: "stuInfoList"
    }
  ]
}]
})
new Vue({
  el: "#app",
  router
})
</script>

</html>

```

欢迎使用学生管理系统

[新增学生](#)
[学生列表](#)


```
<!DOCTYPE html>
<html lang="zh">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>vue嵌套路由实现app常见布局</title>
  <style>
    * {
      margin: 0px;
      padding: 0px;
      list-style-type: none;
    }

    #app {
      width: 100vw;
      height: 100vh;
    }

    .home-box {
      width: 100%;
      height: 100%;
      display: flex;
      flex-direction: column;
    }

    .content-box{
      flex: 1;
    }

    .tab-bar{
      display: flex;
      flex-direction: row;
      justify-content: space-around;
      border-top: 1px solid lightgray;
    }

    .tab-bar>li{
      height: 55px;
      width: 55px;
      display: flex;
      justify-content: center;
      align-items: center;
    }

    .abc-active{
      color: tomato;
      font-weight: bold;
    }
  </style>
</head>

<body>
  <div id="app">
    <router-view></router-view>
  </div>
  <template id="homeTemp">
    <div class="home-box">
      <div class="content-box">
        <router-view></router-view>
      </div>
    </div>
  </template>
</body>
</html>
```

```

        </div>
        <ul class="tab-bar">
          <li>
            <router-link tag="span" :to="{name:'chooseFood'}" active-
class="abc-active">点餐</router-link>
          </li>
          <li><router-link tag="span" :to="{name:'order'}" active-
class="abc-active">订单</router-link></li>
          <li><router-link tag="span" :to="{name:'category'}" active-
class="abc-active">分类</router-link></li>
          <li><router-link tag="span" :to="{name:'mySelf'}" active-
class="abc-active">我的</router-link></li>
        </ul>
      </div>
    </template>
    <template id="chooseFoodTemp">
      <div>
        <h2>这是点餐的界面</h2>
        <p>
          <router-link :to="{name:'detail'}">我现在去点餐界面</router-link>
        </p>
      </div>
    </template>
    <template id="orderTemp">
      <div>
        <h2>这是订单的界面</h2>
      </div>
    </template>
    <template id="categoryTemp">
      <div>
        <h2>这是一个分类的界面</h2>
      </div>
    </template>
    <template id="mySelfTemp">
      <div>
        <h2>我的界面</h2>
      </div>
    </template>
    <template id="detailTemp">
      <div>
        <h2>这是菜品的详细信息</h2>
        <h2>请仔细的看一看我，我到是几级路由</h2>
      </div>
    </template>
  </body>
  <script src="./js/vue.js"></script>
  <script src="./js/vue-router.js"></script>
  <script>
    let home = {
      template: "#homeTemp"
    };
    let chooseFood = {
      template: "#chooseFoodTemp"
    };
    let order={
      template: "#orderTemp"
    };
    let category = {

```

```

        template: "#categoryTemp"
    };
    let myself = {
        template: "#mySelfTemp"
    };
    let detail = {
        template: "#detailTemp"
    };

    let router = new VueRouter({
        routes: [{
            path: "/",
            redirect: {
                name: "home"
            }
        }, {
            path: "/home",
            component: home,
            name: "home",
            children: [
                {
                    path: "chooseFood",
                    component: chooseFood,
                    name: "chooseFood"
                }, {
                    path: "order",
                    component: order,
                    name: "order"
                }, {
                    path: "category",
                    component: category,
                    name: "category"
                }, {
                    path: "mySelf",
                    component: myself,
                    name: "mySelf"
                }
            ]
        }, {
            path: "/detail",
            component: detail,
            name: "detail"
        }
    ]
    })
    new Vue({
        el: "#app",
        router
    })
</script>
</html>

```

这是点餐的界面

[我现在去点餐界面](#)

点餐

订单

分类

我的