

# Vue框架

## 一、Vue简介

---

### 1.1 使用Jquery的复杂性问题

- 使用jQuery进行前后端分离开发，即可以实现前后端交互（ajax），又可以完成数据渲染。
- 存在的问题：jQuery需要通过html标签拼接、DOM节点完成数据的操作。
- vue是继jQuery之后的一个优秀的前端框架，主要专注于前端数据的渲染---语法简单、渲染效率高。

### 1.2 Vue介绍

#### 1.2.1 前端框架

- 前端三要素：HTML CSS JavaScript
  - HTML决定网页结构
  - CSS决定显示效率
  - JavaScript决定网页功能（交互、数据显示）
- UI框架
  - Bootstrap
  - Layui
- Js框架
  - jQuery
  - React
  - angular
  - node.js-----后端开发
  - vue

#### 1.2.2 MVVM

项目结构经历的三个阶段：

后端MVC：单体架构，流程控制是由后端控制器来完成的

前端MVC：前后端分离，后端只负责接收响应请求

MVVM：前端请求后端接口，后端返回数据，前端接收数据，并将接收的数据设置"VM",HTML从 "vm"取值。

- M Model 数据模型 指的是后端接口返回的数据
- V view 视图
- VM viewModel 视图模型 数据模型与视图之间的桥梁，后端返回的model转换为前端所需的vm

## 二、VUE的入门使用

Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。

### 2.1 vue的引入

- 离线引入：下载vue.js文件，添加到前端项目中，在网页中通过script标签引入vue.js文件
- CDN引入

```
<script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js">
</script>
```

### 2.2 入门案例

```
<!DOCTYPE html>
<html lang="en">
  <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>Document</title>
  </head>
  <body>
```

```

<div id="container">
  从vm中获取的数据为<span style="color: red">{{str}}</span>
</div>
</body>
<script type="text/javascript" src="js/vue.js"></script>
<script>
  var vm = new Vue({
    el: "#container", // element 用来给vue实例定义一个作用域范围
    data: {
      // 用来给vue实例定义一些相关的数据
      str: "aa",
      username: "admin",
      pwd: "123456",
    },
  });
</script>
</html>

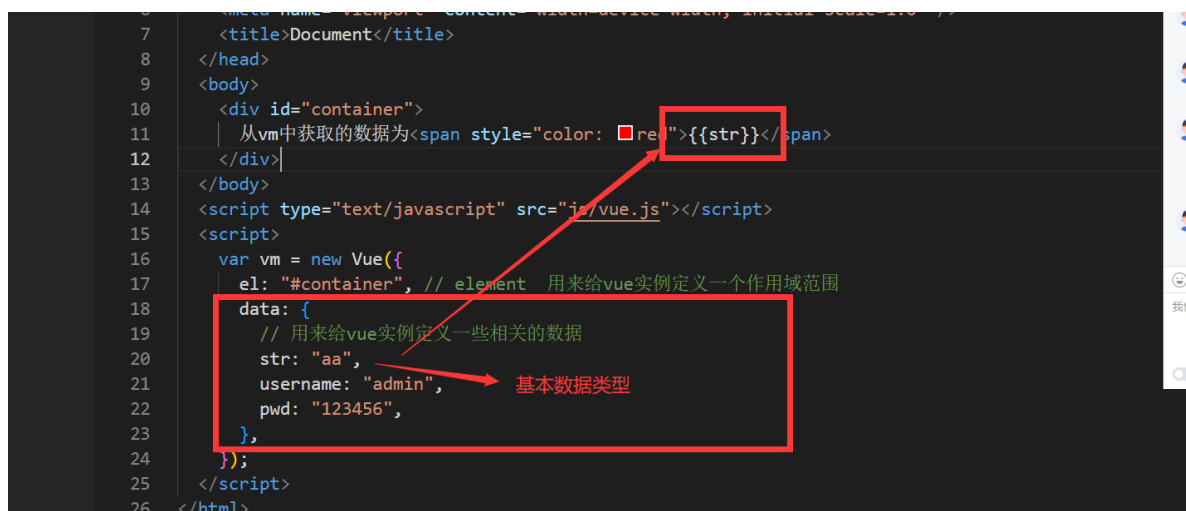
```

注意事项：1. 获取vuedata中的数据使用{{}}进行获取，一定是在vue的作用范围之内

2. el属性可以书写任意的css选择器，但是在使用vue开发时推荐使用id选择器

## 三、Vue的语法

### 3.1 基本类型数据和字符串



```

6  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7  <title>Document</title>
8  </head>
9  <body>
10   <div id="container">
11     从vm中获取的数据为<span style="color: red">{{str}}</span>
12   </div>
13 </body>
14 <script type="text/javascript" src="js/vue.js"></script>
15 <script>
16   var vm = new Vue({
17     el: "#container", // element 用来给vue实例定义一个作用域范围
18     data: {
19       // 用来给vue实例定义一些相关的数据
20       str: "aa",
21       username: "admin",
22       pwd: "123456",
23     },
24   });
25 </script>
26 </html>

```

基本数据类型

## 3.2 对象类型数据

- 支持ognli语法

```
8 </head>
9 <body>
10 <div id="container">
11   从vm中获取的数据:<br />
12   学号: <span style="color: red">{{student.stuNum}}</span><br />
13   姓名: <span style="color: red">{{student.stuName}}</span><br />
14   性别: <span style="color: red">{{student.stuGender}}</span><br />
15   年龄: <span style="color: red">{{student.stuAge}}</span>
16 </div>
17 </body>
18 <script type="text/javascript" src="js/vue.js"></script>
19 <script>
20   var vm = new Vue({
21     el: "#container", // element 用来给vue实例定义一个作用域范围
22     data: {
23       // 用来给vue实例定义一些相关的数据
24       student: {
25         stuNum: "1001",
26         stuName: "zhangsan",
27         stuGender: "男",
28         stuAge: 20,
29       },
30     },
31   });
32 </script>
33 </html>
```

对象类型数据

## 3.3 条件 v-if

```
7 <title>Document</title>
8 </head>
9 <body>
10 <div id="container">
11   从vm中获取的数据:<br />
12   学号: <span style="color: red">{{student.stuNum}}</span><br />
13   姓名: <span style="color: red">{{student.stuName}}</span><br />
14   性别: <span style="color: red">v-if="student.stuGender=='M'">男</span>
15   <br />
16   年龄: <span style="color: red">{{student.stuAge}}</span>
17 </div>
18 </body>
19 <script type="text/javascript" src="js/vue.js"></script>
20 <script>
21   var vm = new Vue({
22     el: "#container", // element 用来给vue实例定义一个作用域范围
23     data: {
24       // 用来给vue实例定义一些相关的数据
25       student: {
26         stuNum: "1001",
27         stuName: "zhangsan",
28         stuGender: "M",
29         stuAge: 20,
30       },
31     },
32   });
```

## 3.4 循环 v-for

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

```

<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>Document</title>
</head>
<body>
  <div id="container">
    <table border="1" cellspacing="0" width="400">
      <tr>
        <th>编号</th>
        <th>学号</th>
        <th>姓名</th>
        <th>性别</th>
        <th>年龄</th>
      </tr>
      <tr v-for="s,index in stus">
        <td>{{index+1}}</td>
        <td>{{s.stuNum}}</td>
        <td>{{s.stuName}}</td>
        <td v-if="s.stuGender=='M'">男</td>
        <td v-if="s.stuGender=='N'">女</td>
        <td>{{s.stuAge}}</td>
      </tr>
    </table>
  </div>
</body>
<script type="text/javascript" src="js/vue.js"></script>
<script>
  var vm = new Vue({
    el: "#container", // element 用来给vue实例定义一个作用域范
    data: {
      // 用来给vue实例定义一些相关的数据
      stus: [
        {
          stuNum: "10001",
          stuName: "tom",
          stuGender: "M",
          stuAge: 20,

```

```

    },
    {
      stuNum: "10002",
      stuName: "lucy",
      stuGender: "N",
      stuAge: 18,
    },
    {
      stuNum: "10003",
      stuName: "jim",
      stuGender: "M",
      stuAge: 22,
    },
  ],
},
));
</script>
</html>

```

### 3.5 v-bind绑定标签属性

- v-bind 可以简写为：

```

3  <head>
4    <meta charset="UTF-8" />
5    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7    <title>Document</title>
8  </head>
9  <body>
10   <div id="container">
11     
12     <a :href="msg">点击</a>
13   </div>
14 </body>
15 <script type="text/javascript" src="js/vue.js"></script>
16 <script>
17   var vm = new Vue({
18     el: "#container", // element 用来给vue实例定义一个作用域范围
19     data: {
20       stuImg: "images/a.jpg",
21       msg: "http://www.baidu.com.cn"
22     },
23   });
24 </script>
25 </html>
26

```

### 3.6 表单标签的双向绑定 v-model

- 只能使用在表单输入标签
- v-model:value 可以简写为: v-model

```

8   </head>
9   <body>
10    <div id="container">
11      <form>
12        <p>用户名:<input type="text" v-model="str" /></p>
13        <p>密码:<input type="text" v-model="str" /></p>
14        <p><input type="submit" value="登录" /></p>
15      </form>
16    </div>
17  </body>
18  <script type="text/javascript" src="js/vue.js"></script>
19  <script>
20    var vm = new Vue({
21      el: "#container", // element 用来给vue实例定义一个作用域范围
22      data: {
23        str: "aa",
24      },
25    });
26  </script>
27 </html>
28

```

### 3.7 v-on绑定事件

```

<!DOCTYPE html>
<html lang="en">
  <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>Document</title>
  </head>
  <body>
    <div id="container">
      <form>
        <h2>年龄: {{age}}</h2>
        <p>
          <!-- <input type="button" value="点我改变年龄"
@click="changeAge" />
          v-on 可以简写 为 @
          -->
          <input type="button" value="点我改变年龄" v-
on:click="changeAge" />
        </p>
      </form>
    </div>
  </body>
  <script type="text/javascript" src="js/vue.js"></script>
  <script>
    var vm = new Vue({

```

```
el: "#container", // element 用来给vue实例定义一个作用域范围

data: {
  age: 20,
},
methods: {
  //用来定义vue中事件
  changeAge: function () { //this代表获取当前vue实例中相关的数据
    this.age++;
  },
};
</script>
</html>
```

注意事项：1. 在vue中绑定事件通过v-on指令来完成，v-on:事件名如：v-on:click

2. 在vue定义的事件中可以通过this关键字来获取当前vue实例中的相关数据

3. v-on 可以简写为@

## 四、Axios的基本使用

### 4.1 axios介绍

Axios 是一个异步请求技术，核心作用就是用来在页面中发送异步请求，并获取对应数据在页面中渲染，页面局部更新技术 Ajax

### 4.2 Axios第一个程序

中文网站：<http://www.axios-js.com/>

安装：<https://unpkg.com/axios@0.27.2/dist/axios.min.js>

- 相比于原生ajax来讲，简洁、高效，对restful支持良好

#### 4.2.1 GET方式的请求



```

<!DOCTYPE html>
<html lang="en">
  <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>axios的使用</title>
  </head>
  <body>
    <div id="container">
      <button type="button" @click="change">测试</button>
    </div>
  </body>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript" src="js/axios.min.js">
</script>
  <script>
    var vm = new Vue({
      el: "#container",
      data: {},
      methods: {
        change: function () {
          axios
            .get(
              "http://localhost:8090/user/login?
username=zhangsan&password=88888888"
            )
            .then(function (res) {
              console.log(res);
            })
            .catch(function (err) {});
        },
      },
    });
  </script>
</html>

```

传参数方式：

```

<!DOCTYPE html>

```

```

<html lang="en">
  <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>axios的使用</title>
  </head>
  <body>
    <div id="container">
      <button type="button" @click="change">测试</button>
    </div>
  </body>
  <script type="text/javascript" src="js/vue.js"></script>
  <script type="text/javascript" src="js/axios.min.js">
</script>
  <script>
    var vm = new Vue({
      el: "#container",
      data: {},
      methods: {
        change: function () {
          //result返回的结果
          // axios.get('http://localhost:8090/user/login?
username=zhangsan&password=88888888').then((result) => {
            //      console.log(result)

            // }).catch((err) => {

            // });

          // 使用axios的get请求传递参数，需要将参数设置在params下
          (restful支持风格)
          axios
            .get(
              "http://localhost:8090/user/login",
              {
                params: {
                  username: "zhangsan",
                  password: "88888888",
                },
              },

```

```

        }
    )
    .then((result) => {
        console.log(result);
    })
    .catch((err) => {});
},
});
</script>
</html>

```

## 4.2.2 Post方式的请求

- 前端代码

```

<script>
    var vm = new Vue({
        el: "#container",
        data: {},
        methods: {
            change: function () {
                //后端都是以对象进行接收
                let data = { username: "tomaa", password: "88888888"
};

                //post请求
                axios
                    .post("http://localhost:8090/user/regist", data)
                    .then((result) => {
                        console.log(result);
                    })
                    .catch((err) => {});
            },
        },
    });
</script>

```

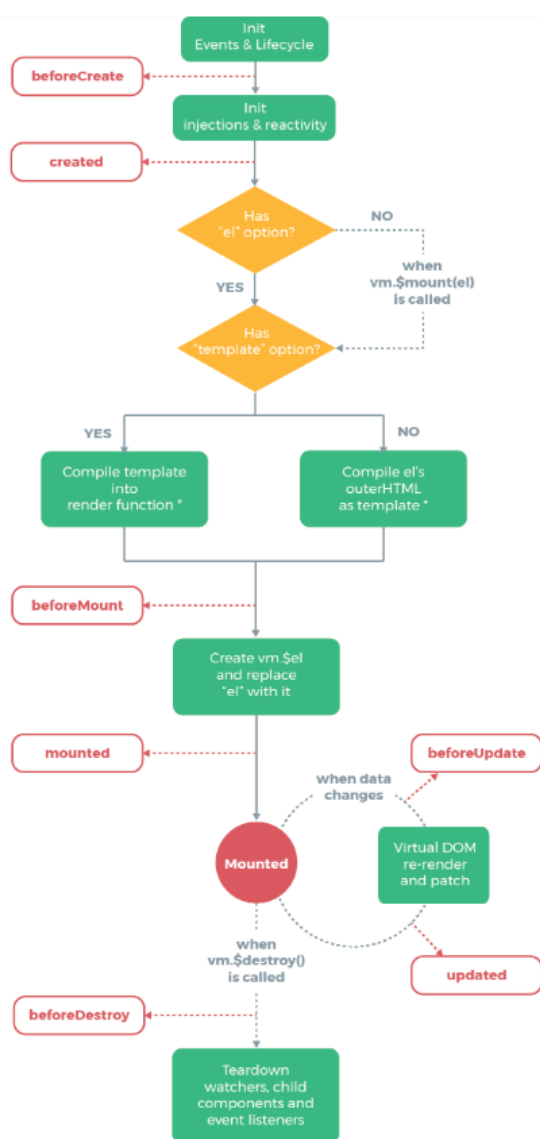
- 后端代码

```

@ApiOperation("用户注册接口")
@ApiImplicitParams({
    @ApiImplicitParam(dataType = "string",name = "username",value = "用户注册账号",required = true),
    @ApiImplicitParam(dataType = "string",name = "password",value = "用户注册密码",required = false,defaultVa
})
@PostMapping("/regist")
public ResultVO regist(@RequestBody Users users){
    System.out.println(users.toString()+"-----");
    ResultVO resultVO=usersService.userResgit(users.getUsername(),users.getPassword());
    return resultVO;
}
}

```

## 五、Vue生命周期



### # vue生命周期:

#### 1. 初始化阶段:

**beforeCreate():** 生命周期的第一个函数，该函数在执行时vue实例仅仅完成了自身事件的绑定和声明周期函数的初始化工作。

**created()**: 生命周期的第二个函数，该函数在执行时vue实例已经初始化了data属性和methods中相关的方法

**beforeMount()**: 生命周期中第三个函数，该函数在执行vue时将EL中指定作用范围作为模板编译

**mounted()**: 生命周期中第四个函数，该函数在执行过程中，已经将数据渲染到界面中 并且已经更新页面

## 2. 运行阶段

**beforeUpdate()**: 生命周期中第五个函数，该函数是data中数据发生变化时执行，这个事件执行时仅仅是vue实例中data数据变化页面显示的依然是原始数据

**updated()**: 生命周期第六个函数，该函数执行时data中数据发生变化，页面中数据也发生了变化，页面中数据已经和data中数据一致。

## 3. 销毁阶段

**beforeDestroy()**: 生命周期第七个函数，该函数执行时vue中所有数据methods 都没销毁

**destroyed()**: 生命周期第八个函数，该函数执行时，vue实例彻底销毁。

# 六、Vue中的组件

## 6.1 组件的作用

组件作用：用来减少vue实例对象中代码量，可以根据不同的业务功能将页面划分不同的组件，由多个组件去完成整个页面的布局，便于维护。

## 6.2 组件的使用

### 6.2.1 局部组件

#### 1. 第一种开发方式

```
<!-- 1.声明组件 template标签 注意：在vue实例作用范围外声明 -->
<template id="loginTemplate">
  <h1>用户登录</h1>
</template>
```

```
//2. 定义组件
let login = {
  template: "#loginTemplate", //使用自定义template标签选择器
即可。
};
```

```
var vm = new Vue({
  el: "#container",
  data: {},
  components: {
    // 3. 注册组件
    login: login,
  },
});
```

```
<!-- 4.使用组件 -->
<login></login>
```

## 2. 第二种开发方式

```
<!DOCTYPE html>
<html lang="en">
  <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>组件的使用</title>
  </head>
  <body>
    <div id="container">
      <login></login>
    </div>

  </body>
  <script type="text/javascript" src="js/vue.js"></script>
  <script>
    //1.自定义组件
    let login = { //组件的名称
      template: "<div><h1>用户登录</h1><h1>用户注册</h1></div>",
```

```

};

let vm=new Vue({
  el:"#container",
  data:{

  },
  components: {
    login:login //2.注册组件
  }
})

</script>
</html>

```

### 6.2.2 全局组件

说明：全局组件注册给 vue实例，日后可以在任意 vue实例的范围内使用该组件

```

<script>
  //1. 开发全局组件
  Vue.component("login", {
    template: "<div><h3>用户登录</h3></div>",
  });

  let vm = new Vue({
    el: "#container",
    data: {},
  });
</script>

<div id="container">
  <!-- 2.使用组件 在vue实例范围内 -->
  <login></login>
</div>

```

## 七、vue的路由

router是由vue官方提供的用于实现组件跳转的插件。

## 7.1 路由插件的引用

### 7.1.1 离线引入

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

### 7.1.2 在线cdn

```
<script type="text/javascript" src="https://unpkg.com/vue-router@4.0.15/dist/vue-router.global.js"></script>
```

## 7.2 路由的使用

### 1. 引入路由

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

### 2. 创建组件对象

```
let login = {
  template: "<h3>用户登录</h3>",
};

let register = {
  template: "<h3>用户注册</h3>",
};
```

### 3. 定义路由对象的规则



```
let router=new VueRouter({
  routes:[
    {path:"/login",component:login}, // path: 路由的路径
    {path:"/register",component:register}
  ]
})
```

#### 4. 将路由对象注册到vue实例中

```
let app=new Vue({
  el:"#container",
  data:{

  },
  router:router // 设置路由对象
})
```

#### 5. 在页面中显示路由的组件

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

#### 6. 根据连接切换路由

```
<a href="#/login">点我登录</a>
<a href="#/register">点我注册</a>
```

## 7.3 路由案例

```
<!DOCTYPE html>
<html lang="en">
  <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>路由案例</title>
    <style>
      * {
        margin: 0px;
        padding: 0px;
```

```

    }
    ul {
        list-style: none;
    }
    ul li {
        display: inline;
        float: left;
        margin-left: 20px;
        margin-bottom: 20px;
    }
    ul li a {
        text-decoration: none; /*去掉下划线*/
        color: white;
        font-size: 20px;
        font-weight: bold;
    }
    ul li a:hover {
        color: yellow;
    }
</style>
</head>
<body>
    <div id="container">
        <div style="width: 100%; height: 70px; background:
gray">
            <table>
                <tr>
                    <td>
                        <ul>
                            <li><a href="#/stu1">首页</a></li>
                            <li><a href="#/stu2">学生信息</a></li>
                            <li><a href="#/stu3">学生添加</a></li>
                            <li><a href="#/stu4">学生修改</a></li>
                            <li><a href="#/stu5">学生删除</a></li>
                        </ul>
                    </td>
                </tr>
            </table>
        </div>
        <div style="width: 100%; height: 400px">
            <!-- 显示路由 -->

```

```
        <router-view></router-view>
    </div>
</div>
</body>
<script type="text/javascript" src="js/vue.js"></script>
<script type="text/javascript" src="js/vue-router.js">
</script>
<script>
    //定义组件
    let stu1 = {
        template: "<h3>首页</h3>",
    };
    let stu2 = {
        template: "<h3>学生信息</h3>",
    };
    let stu3 = {
        template: "<h3>学生添加</h3>",
    };
    let stu4 = {
        template: "<h3>学生修改</h3>",
    };
    let stu5 = {
        template: "<h3>学生删除</h3>",
    };

    let routes = new VueRouter({
        routes: [
            { path: "/stu1", component: stu1 }, // path: 路由的路径
            // component: 路径对应的组件
            { path: "/stu2", component: stu2 },
            { path: "/stu3", component: stu3 },
            { path: "/stu4", component: stu4 },
            { path: "/stu5", component: stu5 },
        ],
    });

    let app = new Vue({
        el: "#container",
        data: {},
        router: routes, // 设置路由对象
    });
```

```
</script>
</html>
```

## 7.4 router-link的使用

- 作用：用来替换我们在切换路由时使用a标签切换路由
- 好处：就是可以自动给路由路径加入#不需要手动加入

```
<li><router-link to="/stu1" tag="a">首页</router-link></li>
<li><router-link to="/stu2" tag="a">学生信息</router-link></li>
<li><router-link to="/stu3" tag="a">学生添加</router-link></li>
<li><router-link to="/stu4" tag="a">学生修改</router-link></li>
<li><router-link to="/stu5" tag="a">学生删除</router-link></li>
```

### # 总结：

- 1.router-link 用来替换使用a标签实现路由切换 好处不需要书写#号路由路径
- 2.router-link to属性用来书写路由路径 tag属性：用来将router-link渲染成指定的标签

## 7.5 默认路由

- 作用：用来在第一次进入界面是显示一个默认组件

```
//2.定义路由规则
let routes = new VueRouter({
  routes: [
    { path: "/", redirect: "/stu1" }, //redirect: 用来当访问的是默认路由时，跳转到指定的路由展示
    { path: "/stu1", component: stu1 }, // path: 路由的路径
    component:路径对应的组件
    { path: "/stu2", component: stu2 },
    { path: "/stu3", component: stu3 },
    { path: "/stu4", component: stu4 },
    { path: "/stu5", component: stu5 },
  ],
});
```

## 7.6 路由中参数传递

- 第一种方式传递参数（传统方式）

## 1. 通过? 号形式拼接参数

```
<router-link to="/stu2?id=2&name=aa" tag="a">学生信息</router-link>
```

## 2. 组件中获取参数

```
//1. 定义组件
let stu1 = {
  template: "#login",
};
let stu2 = {
  template: "<h3>学生信息</h3>",
  data() {
    return {};
  },
  methods: {},
  created() {
    console.log(
      this.$route.query.id + "-----" +
      this.$route.query.name
    );
  },
};
```

## • 第二种方式传递参数 restful

### 1. 通过使用路径方式传递参数

```
<router-link to="/stu2/20/zhangsan" tag="a">学生信息</router-link>

let routes = new VueRouter({
  routes: [
    { path: "/stu2/:id/:name", component: stu2 }
  ],
});
```

### 2. 组件中获取参数

```
let stu1 = {
  template: "#login",
};
```

```

let stu2 = {
  template: "<h3>学生信息</h3>",
  data() {
    return {};
  },
  methods: {},
  created() {
    console.log(

this.$route.params.id+"===== "+this.$route.params.name

    );
  },
};

```

## 7.7 嵌套路由

### 1. 声明最外层和内层路由

```

<template id="product">
  <div>
    <h1>商品管理</h1>
    <router-link to="/add" tag="a">商品添加</router-link>
    <router-link to="/edit" tag="a">商品编辑</router-link>
    <router-view></router-view>
  </div>
</template>

```

### 2. 声明组件

### //1.声明组件

```
let product = {  
  template: "#product",  
};  
  
let add = {  
  template: "<h4>商品添加</h4>",  
};  
  
let edit = {  
  template: "<h4>商品编辑</h4>",  
};
```

## 3. 创建路由对象含有嵌套路由

### //2.定义路由的规则

//创建路由对象含有嵌套路由

```
let router = new VueRouter({  
  routes: [  
    {  
      path: "/product",  
      component: product,  
      children: [  
        {  
          path: "/add",  
          component: add,  
        },  
        {  
          path: "/edit",  
          component: edit,  
        },  
      ],  
    },  
  ],  
});
```

## 4. 注册路由对象

```
//3. 注册路由对象
const app = new Vue({
  el: "#container",
  data: {},
  router, //定义路由对象
});
```

## 5. 测试路由

```
<div id="container">
  <router-link to="/product">商品管理</router-link>
  <!-- 显示路由内容 -->
  <router-view></router-view>
</div>
```