

在工作之后重新捡起了Vue....现在都是全栈了嘛....

Vue2

第一个vue程序

快捷：html 5导入模板

```
<!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>Vue基础</title>
</head>
<body>
    <div id="app">
        {{ message }}
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        var app =new Vue({
            el:"#app",
            data:{
                message:"hello vue!"
            }
        })
    </script>
</body>
</html>
```

创建代码片段

文件 => 首选项 => 用户代码片段 => 新建全局代码片段/或文件夹代码片段：

名称为：vue-html.code-snippets

```
{
    "vue html": {
        "scope": "html",
        "prefix": "!v",
        "body": [
            "<!DOCTYPE html>",
            "<html lang=\"en\">",
            "",
            "<head>",
            "    <meta charset=\"UTF-8\">",
            "    <meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0\">",
            "    <meta http-equiv=\"X-UA-Compatible\" content=\"ie=edge\">",
            "    <title>Document</title>",
            "
```

```
"</head>",
"",
"<body>",
" <div id=\"app\">",
"",
" </div>",
" <script src=\"https://cdn.jsdelivr.net/npm/vue/dist/vue.js\">
</script>",
" <script>",
" new Vue({",
" el: '#app',
" data: {",
" $1",
" }",
" }",
" </script>",
"</body>",
"",
"</html>",
],
"description": "my vue template in html"
}
}
```

我设置的快捷键是: `!v` 如果有需要可以自己改

el挂载点

el是用来设置Vue实例挂载（管理）的元素

1.vue的作用范围

在el命中的元素内部可以被渲染

Vue会管理el选项 命中的元素及其内部的后代元素

```

<body>
    {{message}}
    <div id="app">
        {{message}}
        <span>{{ message }}</span>
    </div>
    
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
    var app=new Vue({
        el:"#app",
        data:{
            message:"Hello Vue! "
        }
    })
</script>
</body>

```

2.是否可以选用其他的选择器

可以，但是建议使用id选择器

3.是否可以设置其他的dom元素

可以使用其他的双标签，但是不能使用HTML和BODY标签

data 数据对象

1.Vue中用到的数据定义在data中

2.data中可以写复杂类型的数据

3.渲染复杂类型数据时，遵循js的语法即可 .语法，数组的索引语法

```

<!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="app">
        {{ message }}
        <h2>{{school.name}} {{school.mobile}}</h2>
        <ul>
            <li>{{campus[0]}}</li>
            <li>{{campus[1]}}</li>
    
```

```
</ul>
</div>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
var app = new Vue({
  el: "#app",
  data: {
    message: "hello,vue",
    school: {
      name: "小黑",
      mobile: "130302030302020"
    },
    campus: ["北京", "上海"]
  }
})
</script>
</body>
</html>
```

hello,vue

小黑 130302030302020

- 北京
- 上海

本地应用-介绍

- 1.通过Vue实现常见的网页效果
- 2.学习Vue指令，以案例巩固知识点
- 3.Vue指令指的是，以v-开头的一组特殊语法

本地应用-v-text

- 1.v-text指令的作用：设置标签的内容 (textContent)
- 2.默认写法会替换全部内容，使用差值表达式{{}}可以替换指定内容
- 3.内部支持写表达式（如字符串拼接）

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
7   <meta http-equiv="X-UA-Compatible" content="ie=edge">
8   <title>v-text指令</title>
9 </head>
10 <body>
11   <div id="app">
12     <h2 v-text="message">深圳</h2>
13     <h2 v-text="info">深圳</h2>
14     <h2>{{ message }}</h2>
15   </div>
16   <!-- 开发环境版本，包含了有帮助的命令行警告 -->
17   <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
18   <script>
19     var app = new Vue({
20       el:"#app",
21       data:{
22         message:"黑马程序员!!!",
23         info:"前端与移动教研部"
24       }
25     })
26   </script>
27 </body>
28
29 </html>

```



v-text

设置标签的文本值(textContent)

内容全部替换

```

<div id="app">
  <h2 v-text="message"></h2>
  <h2>深圳{{ message }}</h2>
</div>

```

局部替换

```

var app = new Vue({
  el:"#app",
  data:{
    message:"黑马程序员"
  }
})

```

字符的拼接

```

<div id="app">
  <h2 v-text="message+'!'"></h2>
  <h2>深圳{{ message + "!" }}</h2>
</div>

```

只能用单

能用单
也能用双

```

var app = new Vue({
  el:"#app",
  data:{
    message:"黑马程序员"
  }
})

```

The screenshot shows a code editor with a dark theme. The code is as follows:

```

1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8" />
6   <meta name="viewport" content="width=device-width,
7     initial-scale=1.0" />
8   <meta http-equiv="X-UA-Compatible" content="ie=edge" />
9   <title>v-text指令</title>
10 </head>
11
12 <body>
13   <div id="app">
14     <h2 v-text="message+'!'">深圳</h2>
15     <h2 v-text="info+'!'">深圳</h2>
16     <h2>{{ message + info }}深圳</h2>
17   </div>
18   <!-- 开发环境版本，包含了有帮助的命令行警告 -->
19   <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
20   <script>
21     var app = new Vue({
22       el:"#app",
23       data:{
24         message:"黑马程序员!!!",
25         info:"前端与移动教研部"
26       }
27     })
28   </script>
29 </body>
30 </html>

```

Annotations in red highlight the following parts of the code:

- A red arrow points from the text "黑马程序员!!!!" at the top right to the line `<h2 v-text="message+'!'">深圳</h2>`.
- A red arrow points from the text "前端与移动教研部!" to the line `<h2 v-text="info+'!'">深圳</h2>`.
- A red arrow points from the text "黑马程序员!!!!深圳" to the line `<h2>{{ message + info }}深圳</h2>`.
- A red arrow points from the text "拼接的使用" at the bottom right to the line `<h2>{{ message + info }}深圳</h2>`.

本地应用-v-html指令

1. v-html指令的作用是:设置元素的innerHTML
2. 内容中有html结构会被解析为标签
3. v-text指令无论内容是什么,只会解析为文本
4. 解析文本使用v-text

需要解析html结构使用v-html

The screenshot shows a code editor with a dark theme. The code is as follows:

```

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>v-html指令</title>
</head>
<body>
  <!-- 2.html结构 -->
  <div id="app">
    <p v-html="content"></p>
    <p v-text="content"></p>
  </div>
  <!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script>
    // 3.创建Vue实例
    var app = new Vue({
      el:"#app",
      data:{
        // content:"黑马程序员"
        content:<a href='http://www.itheima.com'>黑马程序员</a>
      }
    })
  </script>
</body>
</html>

```

Annotations in red highlight the following parts of the code:

- A red arrow points from the text "黑马程序员" at the top right to the line `content:"黑马程序员"`.
- A red arrow points from the text "黑马程序员" to the line `content:黑马程序员`.
- A red arrow points from the text "html" to the line `<p v-html="content"></p>`.
- A red arrow points from the text "text" to the line `<p v-text="content"></p>`.

本地应用-v-on指令基础

```

<div id="app">
  <input type="button" value="事件绑定" v-on:事件名="方法">
</div>

```

```

var app = new Vue({
  el:"#app",
})

```

举例子

```

<div id="app">
  <input type="button" value="事件绑定" v-on:click="doIt">
  <input type="button" value="事件绑定" v-on:mouseenter="doIt">
  <input type="button" value="事件绑定" v-on:dblclick="doIt">
  <input type="button" value="事件绑定" @dblclick="doIt">
</div>

```

```

var app = new Vue({
  el: "#app",
  methods: {
    doIt:function(){
      // 逻辑
    }
  }
})

```

v-on可以替换成@符号

1.v-on指令的作用是:为元素绑定事件(点击, 移入....)

2.事件名不需要写on

3.指令可以简写为@

4.绑定的方法定义在methods属性中

5.方法内部通过this关键字可以访问定义在data中数据

```

<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<meta http-equiv="X-UA-Compatible" content="ie=edge" />
<title>v-on指令基础</title>
</head>
<body>
  <!-- 2.html结构 -->
  <div id="app">
    <input type="button" value="v-on指令" v-on:click="doIt">
    <input type="button" value="v-on简写" @click="doIt">
    <input type="button" value="双击事件" @dblclick="doIt">
    <h2 @click="changeFood">{{ food }}</h2>
  </div>
  <!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  // 3.创建Vue示例
  var app = new Vue({
    el: "#app",
    ② data: {
      ③ food: "西兰花炒蛋"
    },
    ④ methods: {
      ⑤ doIt:function(){
        alert('做It');
      },
      changeFood:function(){
        // console.log(this.food);
        this.food+="好好吃!"
      }
    }
  })
</script>
</body>
</html>

```

修饰符

修饰符 (Modifiers) 是以半角句号 (.) 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。

例如 .prevent 修饰符告诉 v-on 指令对于触发的事件调用 event.preventDefault()：

即阻止事件原本的默认行为

```

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

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

```

```

<body>
    <div id="app">

        <form action="localhost:8080/save" v-on:submit.prevent="onSubmit">
            <input type="text" id="name" v-model="user.username" ></input>
            <button type="submit">保存</button>
        </form>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                user: {
                    username: "请输入你的密码"
                }
            },
            methods: {
                onSubmit() {
                    if(this.user.username){
                        console.log('提交表单')
                    }else{
                        alert("请输入用户名")
                    }
                }
            }
        })
    </script>
</body>

</html>

```

上面的代码是，提交表单，不提交到指定地址，相应的，触发onSubmit()方法

计数器

思路

1. data中定义数据:比如num
2. methods中添加两个方法:比如add(递增),sub(递减)
3. 使用v-text将num设置给span标签
4. 使用v-on将add,sub分别绑定给+,按钮
5. 累加的逻辑:小于10累加,否则提示
6. 递减的逻辑:大于0递减否则提示

总结

1. 创建Vue示例时:el(挂载点),data(数据),methods(方法);
2. v-on指令的作用是绑定事件,简写为@;
3. 方法中通过this关键字获取data中的数据;
4. v-text指令的作用是:设置元素的文本值简写为 {{ }};
5. v-html指令的作用是:设置元素的innerHTML;

```
<body>
```

```
<!-- 2.html结构 -->
<div id="app">
    <!-- 计数器功能区 -->
    <div class="input-num">
        <button @click="sub">-</button>
        <span>{{num}}</span>
        <button @click="add">+</button>
    </div>
</div>
<!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<!-- 编码 -->
<!-- 3.创建Vue实例 -->
<script>
    var app=new Vue({
        el:"#app",
        data:{
            num:1
        },
        methods:{
            add:function(){
                // console.log("add");
                if(this.num<10){
                    this.num++;
                }else{
                    alert("超过10啦");
                }
            },
            sub:function(){
                // console.log("sub");
                if(this.num>0){
                    this.num--;
                }else{
                    alert("到底啦");
                }
            }
        }
    })
</script>
</body>
```

```
<body>
  <!-- 2.html结构 -->
  <div id="app">
    <!-- 计数器功能区 -->
    <button @click="sub">-</button>
    <span>{{num}}</span>
    <button @click="add">+</button>
  </div>
  <!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <!-- 编码 -->
  <!-- 3.创建Vue实例 -->
  <script>
    var app = new Vue({
      el: "#app",
      data: {
        num: 1
      },
      methods: {
        add: function() {
          if(this.num < 10) {
            this.num++;
          } else {
            alert("超过10啦");
          }
        },
        sub: function() {
          if(this.num > 0) {
            this.num--;
          } else {
            alert("到底啦");
          }
        }
      }
    })
  </script>
```

本地应用-v-show指令

1.show指令的作用

根据真假 切换元素的显示

状态原理是修改元素的display,实现显示隐藏

2.指令后面的内容,最终都会解析为布尔值

3.true元素显示, 值为false元素隐藏

4.改变之后, 对应元素的显示状态会同步更新

```
<!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="app">
    <input type="button" value="切换显示状态" @click="changeIsShow">
    <input type="button" value="累加年龄" @click="addAge">
    
    <img v-show="age>=18" src="./1.jpg">
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script>
    var app = new Vue({
      el: "#app",
      data: {
        isShow: true,
        age: 10
      },
      methods: {
        changeIsShow() {
          this.isShow = !this.isShow;
        },
        addAge() {
          this.age++;
        }
      }
    })
  </script>
```

```

        data:{
            isShow:false,
            age:17
        },
        methods:{
            changeIsShow:function(){
                this.isShow=!this.isShow;
            },
            addAge:function(){
                this.age++;
            }
        }
    )
</script>

</body>
</html>

```

本地应用-v-if指令

- 1.v-if指令的作用是:根据表达式的真假切换元素的显示状态
- 2.本质是通过操纵dom元素来切换显示状态
- 3.表达式的值为true,元素存在于dom树中,为false,从dom树中移除
- 4.频繁的切换v-show, 反之使用v-if, 前者的切换消耗小

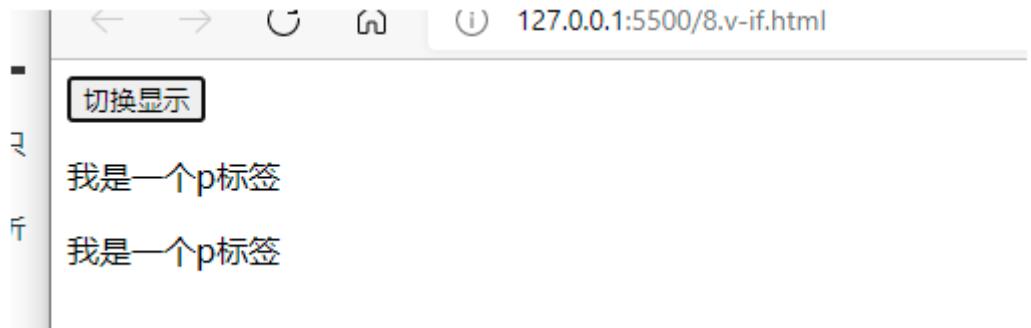
v-if和v-show的区别 v-show直接修改display 而v-if是直接抹除标签

```

<!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="app">
        <input type="button" value="切换显示" @click="change">
        <p v-if="true">我是一个p标签</p>
        <p v-if="isShow">我是一个p标签</p>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        var app=new Vue({
            el:"#app",
            data:{
                isShow:false
            },
            methods:{
                change:function(){
                    this.isShow=!this.isShow;
                }
            }
        })
    </script>

```

```
        }
    })
</script>
</body>
</html>
```



本地应用-v-bind指令

```
<div id="app">
  <img v-bind:src= "imgSrc" >
  <img v-bind:title="imgtitle+'!!!!'">
  <img v-bind:class="isActive?'active':''">
  <img v-bind:class="{active:isActive}">
</div>
```

两种写法

```
var app = new Vue({
  el:"#app",
  data:{
    imgSrc:"图片地址",
    imgTitle:"黑马程序员",
    isActive:false
  }
})
```

可以直接省略v-bind

```
<div id="app">
  <img :src= "imgSrc" >
  <img :title="imgtitle+'!!!!'">
  <img :class="isActive?'active':''">
  <img :class="{active:isActive}">
</div>
```

```
var app = new Vue({
  el:"#app",
  data:{
    imgSrc:"图片地址",
    imgTitle:"黑马程序员",
    isActive:false
  }
})
```

1.v-bind: 属性名=表达式

2.v-bind指令的作用是:为元素绑定属性

3.完整写法是v-bind:属性名

4.简写的话可以直接省略v-bind,只保留:属性名

5.需要动态的增删class建议使用对象的方式

v-bind指令.html

v-bind指令.html > body > div#app > img

```
6      <meta name="viewport" content="width=device-width,
7          initial-scale=1.0">
8      <meta http-equiv="X-UA-Compatible" content="ie=edge">
9      <title>v-bind指令</title>
10     <style>
11         .active{
12             border: 1px solid red;
13         }
14     </style>
15 
16 </head>
17 
18 <div id="app">
19     
20     <br>
21     
23     <br>
24     
26 </div>
27 
28 <!-- 开发环境版本，包含了有帮助的命令行警告 -->
29 <script src="https://cdn.jsdelivr.net/npm/vue@dist/vue.js"></script>
30 
31 <script>
32     var app = new Vue({
33         el:"#app",
34         data:{
35             imgSrc:"http://www.itheima.com/images/logo.png",
36             imgTitle:"黑马程序员",
37             isActive:false
38         },
39         methods: {
40             toggleActive:function(){}}
41     })
42 </script>
```

```
<!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>
        <style>
            .active{
                border: 2px solid red;
            }
        </style>
    </head>
    <body>
        <div id="app">
            
            <!-- <br>      ?      :      如果是      就怎么 -->
            
            <span :class="isActive?'active':'' " @click="toggleActive">
                
                    <!-- 点击变色
                        active取值依赖于isactive是否取值 -->
            </span>
        </div>
        <!-- 开发环境版本，包含了有帮助的命令行警告 -->
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
        <script>
            var app=new Vue({
                el:"#app",
                data:{
```

```

        isActive:false
    },
    methods:{
        toggleActive:function(){
            this.isActive=!this.isActive;
        }
    }
)
</script>
</body>
</html>

```

本地应用-v-for

The screenshot shows a browser window displaying a list of numbers from 1 to 10. The first five numbers (1-5) are highlighted in a red box, and the last five numbers (6-10) are highlighted in another red box. Arrows point from these highlighted sections to specific parts of the code editor on the right.

```

1  <!DOCTYPE html>
2  lang="en">
3
4
5  meta charset="UTF-8"
6  meta name="viewport" content="width=device-width, initial-scale=1.0"
7  meta http-equiv="X-UA-Compatible" content="ie=edge"
8  title>Document</title>
9
10
11
12  <div id="app">
13
14      <ul>
15          <li v-for="n in 10">{{n}}</li>
16      </ul>
17      <ol>
18          <li v-for="(n,index) in 10">{{n}}----{{index}}</li>
19      </ol>
20  </div>
21  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
22
23  new Vue({
24      el: '#app',
25      data: {
26
27      }
28  })
29  <script>
30
31

```

```

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

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

<body>
    <div id="app">

        <ul>
            <li v-for="n in 10">{{n}}</li>
        </ul>
        <ol>
            <li v-for="(n,index) in 10">{{n}}----{{index}}</li>
        </ol>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',

```

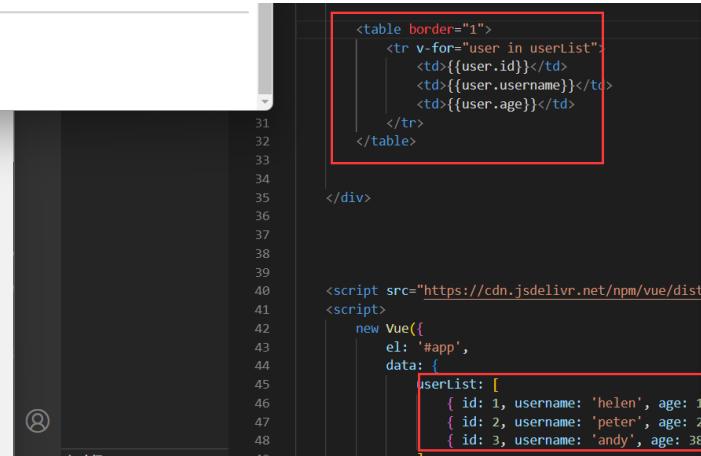
```

    data: {
        }
    })
</script>
</body>

</html>

```

遍历表格 (其中的user可以随便起名字。改成haohao都行)

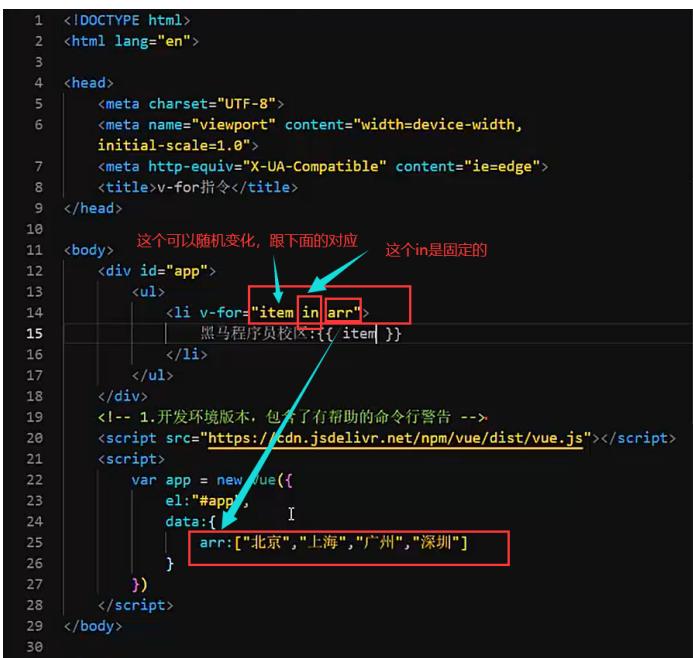
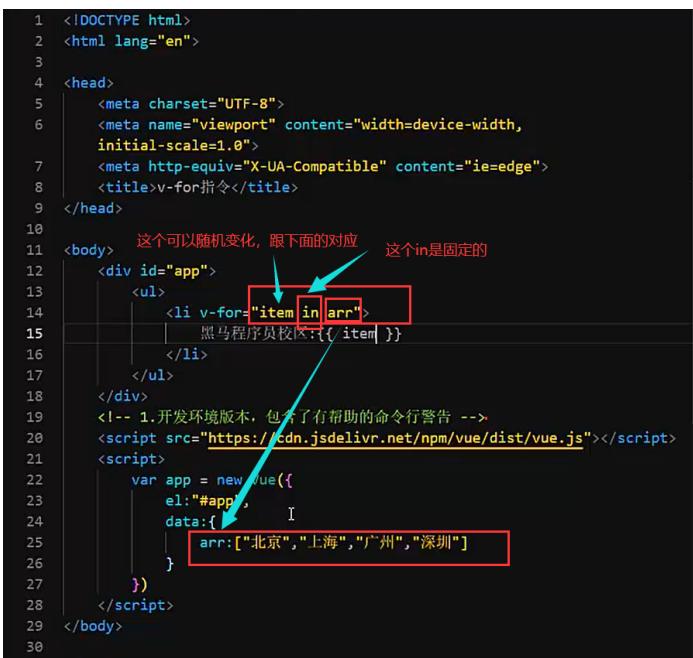
```


|             |                   |              |
|-------------|-------------------|--------------|
| {{user.id}} | {{user.username}} | {{user.age}} |
|-------------|-------------------|--------------|



<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
new Vue({
  el: '#app',
  data: {
    userList: [
      { id: 1, username: 'helen', age: 18 },
      { id: 2, username: 'peter', age: 28 },
      { id: 3, username: 'andy', age: 38 }
    ]
  }
})

```

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>v-for指令</title>
</head>
<body>这个可以随机变化，跟下面的对应          这个in是固定的
  <div id="app">
    <ul>
      <li v-for="item in arr">
        黑马程序员校区:{{ item }}
      </li>
    </ul>
  </div>
  <!-- 1. 开发环境版本，包含了有帮助的命令行警告 -->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  var app = new Vue({
    el:"#app",
    data:{
      arr:["北京","上海","广州","深圳"]
    }
  })
</script>
</body>

```

The screenshot shows a Vue.js application with the following code:

```
10 <body>
11   <div id="app">
12     <input type="button" value="添加数据" @click="add">
13     <input type="button" value="移除数据" @click="remove">
14
15     <ul>
16       <li v-for="(it,index) in arr">
17         {{ index+1 }}黑马程序员校区:{{ it }}
18       </li>
19     </ul>
20     <h2 v-for="item in vegetables" v-bind:title="item.name">
21       {{ item.name }}
22     </h2>
23   </div>
24   <!-- 1. 开发环境版本，包含了有帮助的命令行警告 --&gt;
25   &lt;script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"&gt;&lt;/script&gt;
26   &lt;script&gt;
27     var app = new Vue({
28       el:"#app",
29       data:{
30         arr:["北京","上海","广州","深圳"],
31         vegetables:[
32           {name:"西兰花炒蛋"},
33           {name:"蛋炒西蓝花"}
34         ]
35       },
36       methods: {
37         add:function(){
38           this.vegetables.push({ name:"花菜炒蛋" });
39         },
40         remove:function(){
41           this.vegetables.shift();
42         }
43       }
44     })
45   &lt;/script&gt;
46 </pre>

The application displays a list of cities (arr) and a list of vegetable dishes (vegetables). The vegetables list is styled with a title attribute for each item.


```

1.v-for指令的作用是:根据数据生成列表结构

2.数组经常和v-for结合使用

3.语法是(item,index) in 数据

4.item和index可以结合其他指令一起使用

5.数组长度的更新会同步到页面上，是响应式的

本地应用-v-on补充

1. 事件绑定的方法写成函数调用的形式，可以传入自定义参数
2. 定义方法时需要定义形参来接收传入的实参
3. 事件的后面跟上.修饰符可以对事件进行限制
4. .enter可以限制触发的按键为回车
5. 事件修饰符有多种: <https://cn.vuejs.org/v2/api/#v-on>

模板

```

3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width,
7     initial-scale=1.0">
8   <meta http-equiv="X-UA-Compatible" content="ie=edge">
9   <title>v-on补充</title>
10  </head>
11  <body>
12    <div id="app">
13      <input type="button" value="点击" @click="doIt(666,'老铁')">
14      <input type="text" @keyup.enter="sayHi">
15    </div>
16    <!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
17    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
18    <script>
19      var app = new Vue({
20        el:"#app",
21        methods:{
22          doIt:function(p1,p2){
23            console.log("做it");
24            console.log(p1);
25            console.log(p2);
26          },
27          sayHi:function(){
28            alert("吃了没");
29          }
30        }
31      })
32    </script>
33  </body>

```

```

<!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>

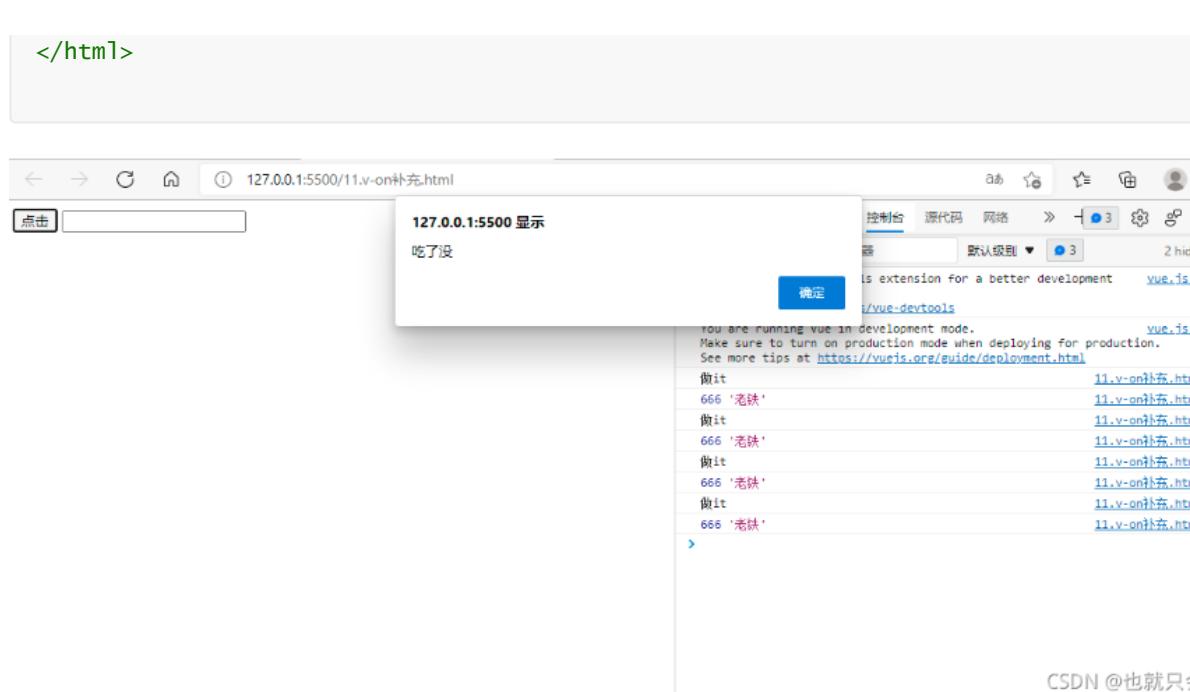
<!-- 2.html结构 -->
<div id="app">
  <input type="button" value="点击" @click="doIt(666,'老铁')">
  <input type="text" @keyup.enter="sayHi">

</div>
<!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<!-- 3.创建Vue实例 -->
<script>
  var app=new Vue({
    el:"#app",
    methods:{
      doIt:function(p1,p2){
        console.log("做it");
        console.log(p1,p2);
        alert("吃了没");
        alert(p1,p2);
      },
      sayHi:function(){
        alert("吃了没");
      }
    }
  })
</script>

</body>

```

```
</html>
```



CSDN @也就只

本地应用-v-model

简单来说双向绑定就是指修改文本框中的message，也会改变data中的message。

1.v-model：获取和设置表单元素的值(双向数据绑定)

The diagram illustrates the two-way data binding mechanism. On the left, an HTML code snippet shows a `<input type="text" v-model="message" />` element. A red arrow points from this element to a corresponding Vue.js script on the right. The script defines a new Vue instance with an `el: "#app"` selector and a `data` object containing a `message` property set to "黑马程序员". This visualizes how the `v-model` directive creates a bidirectional link between the DOM element and the Vue data model.

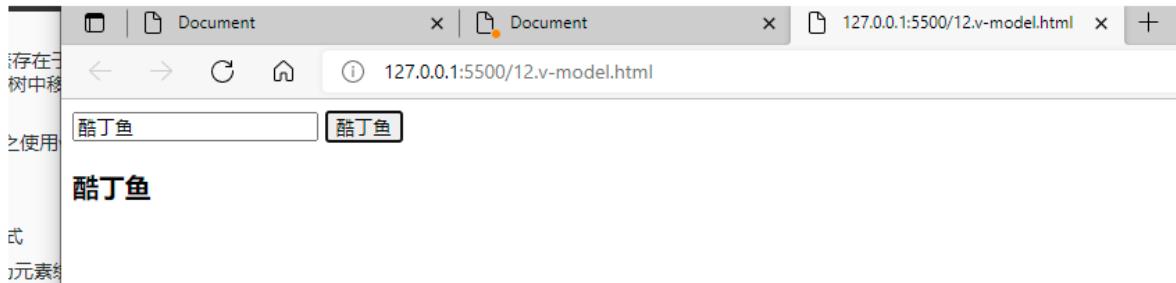
2.v-model指令的作用是便捷的设置和获取表单元素的值

3.绑定的数据会和表单元素值相关联

4.绑定的数据 \longleftrightarrow 表单元素的值

```
<!-- 2.html结构 -->
<div id="app">
  <input type="text" v-model="message" @keyup.enter="getMessage" />
  <input type="button" v-model="message" @click="setMessage" />
  <h3>{{message}}</h3>
</div>
<!-- 1.开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<!-- 3.创建Vue实例 -->
<script>
  var app=new Vue({
    el:"#app",
    data:{
      message:"沙丁鱼"
    },
  })
```

```
methods: {
    getMessage:function(){
        alert(this.message)
    },
    setMessage:function(){
        this.message="酷丁鱼";
    }
}
</script>
```

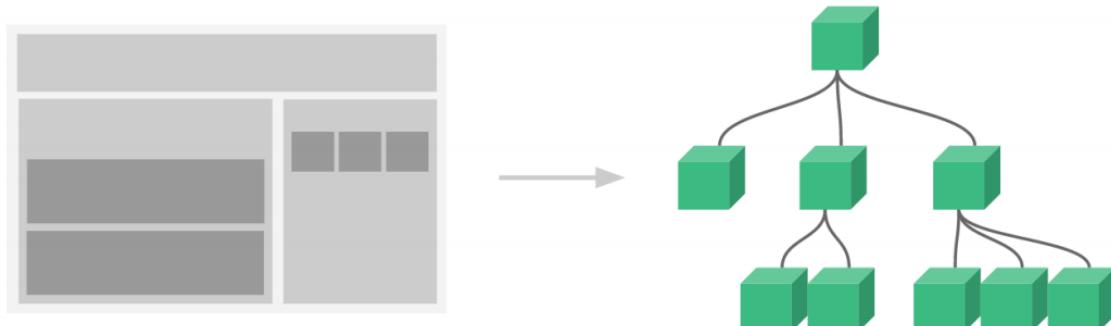


组件

组件 (Component) 是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



局部组件

The screenshot shows a code editor on the left and a browser preview on the right. The code editor displays a Vue.js file with the following content:

```
前端 > vue > ▶ 局部组件.html > ⏺ html > ⏺ body > ⏺ script > ⏺ components
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <meta http-equiv="X-UA-Compatible" content="ie=edge">
8      <title>Document</title>
9  </head>
10 <body>
11     <div id="app">
12         <haohao></haohao>
13     </div>
14     <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
15     </script>
16     new Vue({
17         el: '#app',
18         // 定义局部组件，这里可以定义多个局部组件
19         components: {
20             //组件的名字
21             'haohao': {
22                 //组件的内容
23                 template: '<ul><li>首页</li><li>学员管理</li></ul>'
24             }
25         }
26     })
27     </script>
28 </body>
29 </html>
```

The browser preview on the right shows a simple navigation menu with two items: "首页" (Home) and "学员管理" (Student Management). A red box highlights the "haohao" component in the code editor, and another red box highlights its template definition in the components object.

定义组件

```
new Vue({
    el: '#app',
    // 定义局部组件，这里可以定义多个局部组件
    components: {
        //组件的名字
        'haohao': {
            //组件的内容
            template: '<ul><li>首页</li><li>学员管理</li></ul>'
        }
    }
})
```

使用组件

```
<haohao></haohao>
```

全局组件

创建js文件

The screenshot shows a code editor interface. On the left, the 'Resource Manager' sidebar displays a tree structure of files and folders. A red box highlights the 'components' folder under 'vue', and another red box highlights the 'Navbar.js' file within it. The main editor area shows the following code:

```
// 定义全局组件
Vue.component('haohao', {
  template: '<ul><li>首页</li><li>学员管理</li><li>讲师管理</li></ul>'
})
```

```
// 定义全局组件
Vue.component('haohao', {
  template: '<ul><li>首页</li><li>学员管理</li><li>讲师管理</li></ul>'
})
```

The screenshot shows a code editor with the following template code:

```
body>
  <div id="app">
    <haohao></haohao> ②
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script src="components/Navbar.js"></script>
```

A red box highlights the 'haohao' component tag, and another red box highlights the script tag for 'components/Navbar.js'. A red circle with the number '2' is placed next to the component tag.

引入并使用

实例生命周期

我们主要了解的是在渲染页面之前会执行 `created()` 方法，在渲染完成后执行 `mounted()` 方法，那么如何查看呢，我们了解了这两个方法就可以了，因为主要用的就是这两个。

前端也有debug模式，通过 `debugger` 关键字来使用

```
<body>
  <div id="app">
    <h1>hahahahaha</h1>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <script>
    new Vue({
```

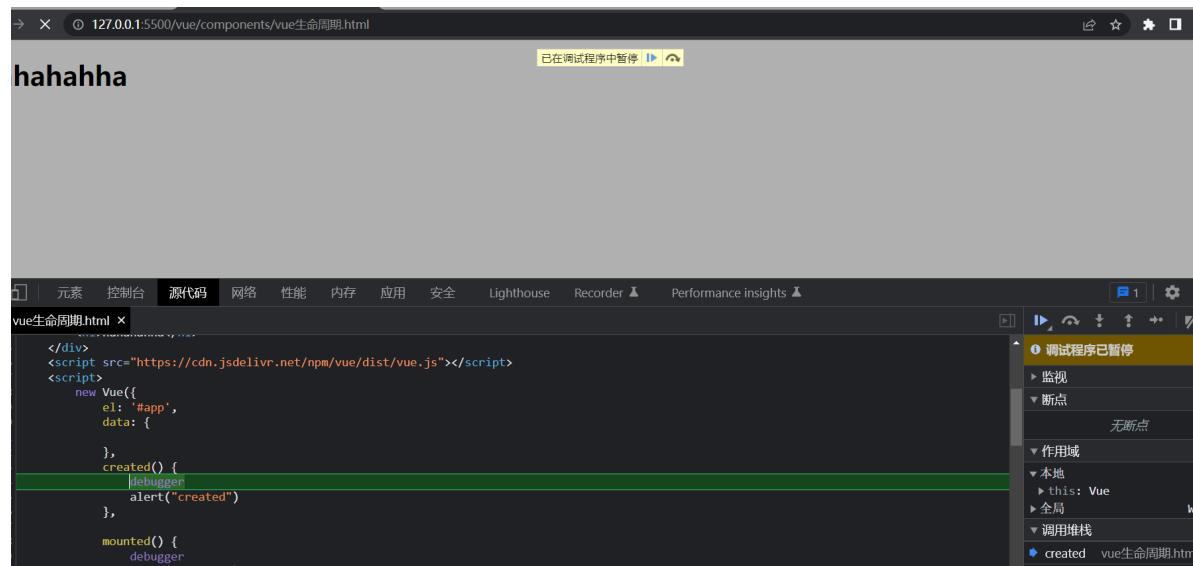
```

    el: '#app',
    data: {

    },
    created() {
        debugger
        alert("created")
    },
    mounted() {
        debugger
        alert("mounted")
    }
)
</script>
</body>

```

直接启动之后运行一下点刷新就可以进入debug模式了。



生命周期的八大钩子函数

什么是钩子函数，官方文档：

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会。

简单来说，钩子函数就算创建Vue在初始化、更新数据、销毁时会自动被调用的函数

[API — Vue.js (vuejs.org)](<https://v2.cn.vuejs.org/v2/api/#选项-生命周期钩子>)

八大钩子数分别是：

beforeCreate, created, beforeMount, mounted, beforeUpdate, updated, beforeDestory, destoryed

官网的还多了几个

API — Vue.js

https://v2.cn.vuejs.org/v2/api/#选项_生命周期钩子

您正在浏览的是 Vue 2 的文档。此外您还可以移步 [Vue 3](#) 的文档，或有关 [Vue 2.7 和 延长版 LTS](#) 的信息。

学习 生态系统 资源列表 延长版 LTS 新 多语言 参与翻译

选项 / 生命周期钩子

所有生命周期钩子的 `this` 上下文将自动绑定至实例中，因此你可以访问 `data`、`computed` 和 `methods`。这意味着你不应该使用箭头函数来定义一个生命周期方法（例如 `created: () => this.fetchTodos()`），因为箭头函数绑定了父级上下文，所以 `this` 不会指向预期的组件实例，并且 `this.fetchTodos` 将会是 `undefined`。

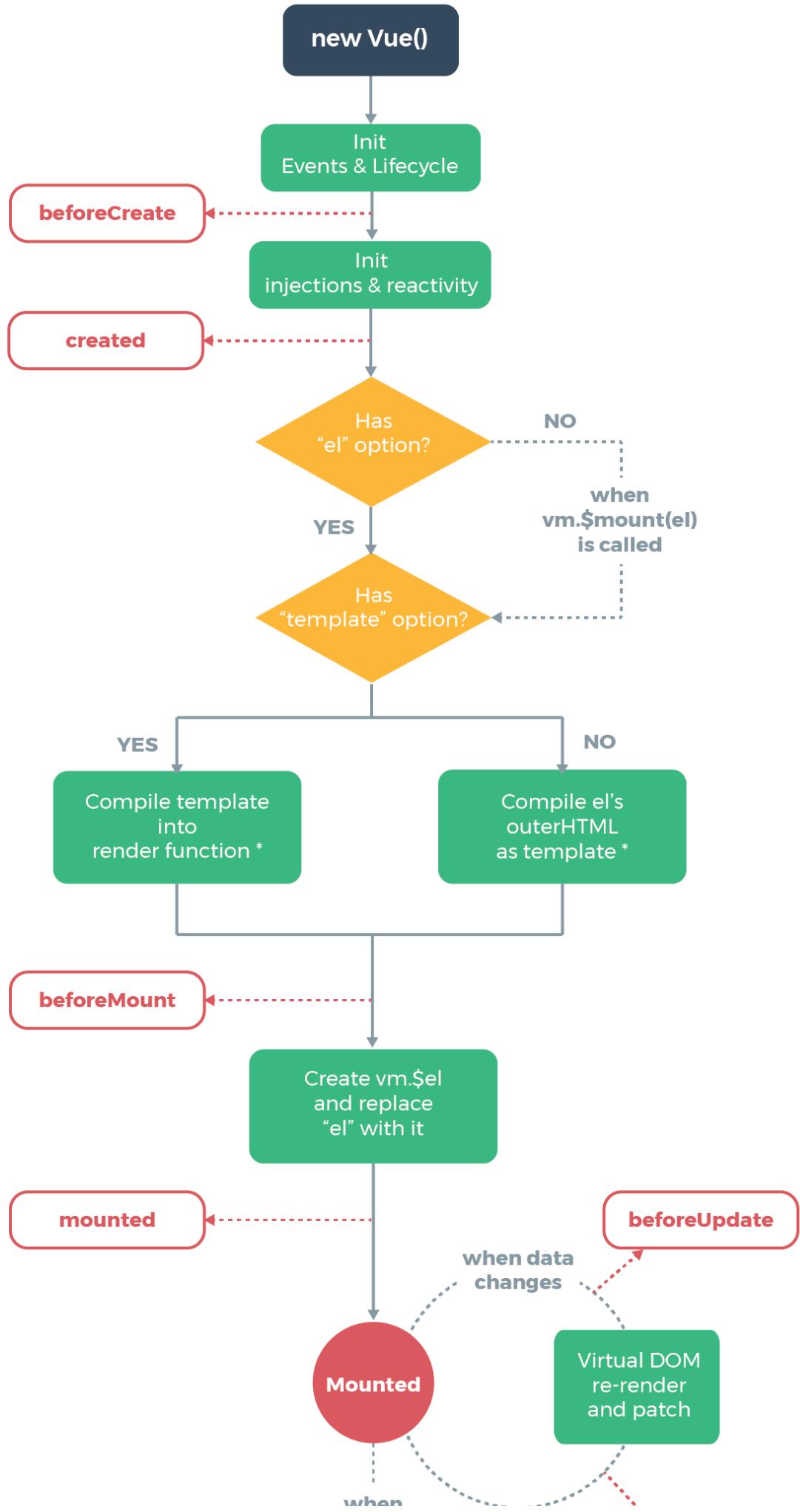
beforeCreate

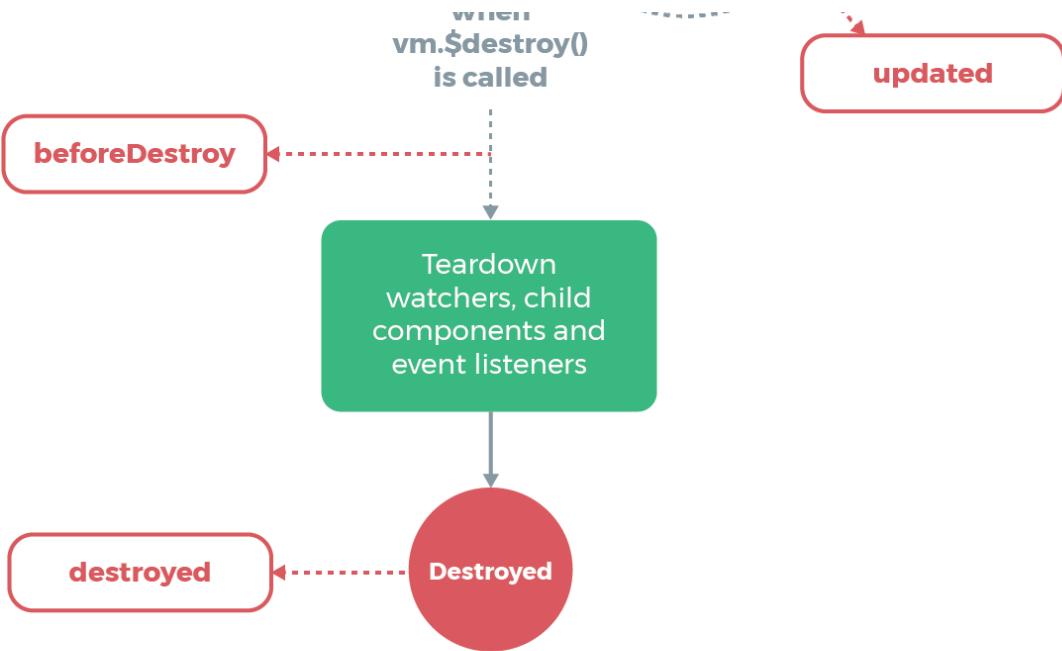
- 类型: `Function`
- 详细:
- 在实例初始化之后，进行数据侦听和事件/侦听器的配置之前同步调用。
- 参考: [生命周期图示](#)

created

Vue.js API 文档

用图片来表示





* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

一、beforeCreate,created

beforeCreate可以简单的理解为在数据初始化的之前被调用，**这时候data和methods尚未没有数据。**

created可以理解为在数据初始化之后被调用，**这时候data和methods已经被填充了相应的数据。**

```

<body>
  <div id="app"></div>
  <script>
    var vm=new Vue({
      el: "#app",
      data: {
        msg:"在这之间"          //添加msg数据
      },
      beforeCreate() {
        console.log("this= "+this)
        console.log("this.msg= "+this.msg)
        console.log("this.md= "+this.md)
        console.log(""))
      },
      created() {
        console.log("this= "+this)
        console.log("this.msg= "+this.msg)
        console.log("this.md= "+this.md)
        console.log("")
      },
      methods: {
        md: function(){},      //空方法
      }
    })
  </script>

```

```
        });
    </script>
</body>
```

结果：

```
this= [object Object]
this.msg= undefined      可见beforeCreate方法执行时：该Vue对象已创建，data中没有数据，methods中没有数据
this.md= undefined

this= [object Object]
this.msg= 在这之间      可见created方法执行时：该Vue对象已创建，data中有数据，methods中有数据
this.md= function () { [native code] }
```

在beforeCreate方法与create方法之间完成了资源的注入

二、beforeMount,mounted

上面实验已经证明Vue中数据已经注入完毕，beforeMount,mounted则是与页面渲染有关

beforeMount在页面尚未被渲染时使用，也就是Vue的数据没有传到页面。

mounted在页面渲染完成之后使用，也就是此时页面已完全取出Vue中的数据。

实验测试：

```
<body>
  <div id="app">
    <h1 id="ren">{{msg}}</h1>
  </div>
  <script>
    var vm=new Vue({
      el: "#app",
      data: {
        msg:"在这之间"          //添加msg数据
      },
      beforeMount() {
        let doc = document.querySelector("#ren"); //查询到id名为ren的节点
        console.log(doc)
        console.log("")
      },
      mounted() {
        let doc = document.querySelector("#ren");
        console.log(doc)
      },
    });
  </script>
</body>
```

结果如下：

```
<h1 id="ren">{{msg}}</h1>      msg的数据此时尚未取出
```

```
<h1 id="ren">在这之间</h1>      msg的值已经渲染到页面
```

此时，Vue对象中资源已注入完毕，页面也已经渲染完毕，上述四个方法在页面被加载时自动被执行

三、beforeUpdate,updated

- beforeUpdate在页面更新渲染完成后，DOM树发生改变前被调用
- updated在页面DOM树改变后被调用

需要注意的是如果只是改变了dom中的数据（data），未对页面造成任何影响，就不会触发beforeUpdate,updated方法。

```
<body>
  <div id="app">
    <h1 id="ren">
      <p v-if="msg"></p>
    </h1>
  </div>
  <script>
    var vm=new Vue({
      el: "#app",
      data: {
        msg:true          //添加msg数据
      },
      beforeupdate() {
        let a = document.getElementById("ren");
        console.log(a.childElementCount)
        console.log("")
      },
      updated() {
        let a = document.getElementById("ren");
        console.log(a.childElementCount)
      },
    });
  </script>
</body>
```

结果显示：

```
> vm.msg=false
  1      dom树还未删除节点
  0      dom树已经删除节点
```

四、beforeDestory,destoryed

beforeDestory是在Vue组件销毁之前被调用

destoryed在Vue组件销毁之后被调用

这里为了搭建环境，引入了组件的概念（注意由于解析时自上而下，所以组件写在Vue对象前）

```

<body>
  <div id="app">
    <mytest></mytest>          自定义组件就是模板，可以被多次引用
  </div>
  <script>
    let myname = Vue.component('mytest', {
      template: '<li>{{msg}}</li>',           在Vue中引入组件
      data :function(){
        return{
          msg : "这是我的测试"
        }
      },
      methods: {
        m1:function(){}
      },
    });
    var vm=new Vue({
      el: "#app",
      data: {},
      components:{             组件标签名，自定义
        "mytest" : myname,
      }
    });

  </script>
</body>

```

https://blog.csdn.net/a_killer_

```

<body>
  <div id="app">
    <mytest id="child" v-if="flag">
    </mytest>
  </div>
  <script>

    let myname = Vue.component('mytest', {
      template: '<p>yes</p>',

      beforeDestroy() {
        console.log("beforeDestroy被执行")
      },
      destroyed() {
        console.log("destroyed被执行")
      },
    });
    var vm=new Vue({
      el: "#app",
      data: {
        flag: true
      },
      components:{
        "mytest" : myname,
      },
    });

  </script>
</body>

```

SEE MORE TIPS AT <https://vuejs.org/guide/zh-CN/components/using-components.html>

```

> vm.flag=false
beforeDestroy被执行
destroyed被执行
< false

```

路由

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图的单页Web应用 (single page web application, SPA) 。

Vue.js 路由需要载入 vue-router 库

```
<!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>vue基础</title>
</head>
<body>

  <div id="app">
    <div id="app">
      <h1>Hello App!</h1>
      <p>
        <!-- 使用 router-link 组件来导航. -->
        <!-- 通过传入 `to` 属性指定链接. -->
        <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
        <router-link to="/">首页</router-link>
        <router-link to="/student">会员管理</router-link>
        <router-link to="/teacher">讲师管理</router-link>
      </p>
      <!-- 路由出口 -->
      <!-- 路由匹配到的组件将渲染在这里 -->
      <router-view></router-view>
    </div>
  </div>

  <script src="https://cdn.staticfile.org/vue/2.4.2/vue.min.js"></script>
  <!-- 低版本，也可以使用 -->
  <!-- <script src="https://cdn.staticfile.org/vue-router/2.7.0/vue-router.min.js"></script> -->
  <script src="https://cdn.jsdelivr.net/npm/vue-router@3.5.1/dist/vue-router.min.js"></script>

  <script>
    // 1. 定义（路由）组件。
    // 可以从其他文件 import 进来
    const Welcome = { template: '<div>欢迎</div>' }
    const Student = { template: '<div>student list</div>' }
    const Teacher = { template: '<div>teacher list</div>' }

    // 2. 定义路由
    // 每个路由应该映射一个组件。
    const routes = [
      { path: '/', redirect: '/welcome' }, //设置默认指向的路径
    ]
  </script>
```

```
{ path: '/welcome', component: Welcome },
{ path: '/student', component: Student },
{ path: '/teacher', component: Teacher }
]

// 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。
// 从而让整个应用都有路由功能
const app = new Vue({
  el: '#app',
  router
})
// 现在，应用已经启动了！
</script>
</body>
</html>
```

Hello App!

[首页](#) [会员管理](#) [讲师管理](#)

欢迎

点击超链接会改变下面文字的内容

基本结构

The screenshot shows a code editor window titled "VehicleCloudEye" with a dark theme. The file being edited is "hello.html". The code is a Vue.js application demonstrating routing. It includes imports for Vue and Vue Router, defines components for Welcome, Student, and Teacher, and sets up routes for '/', '/welcome', '/student', and '/teacher'. A callout box highlights the line "

5. 隐藏

没有数据时，隐藏元素（数组非空时v-if v-show）

6. 总结

1. 列表结构可以通过v-for指令结合数据生成
2. v-on结合事件修饰符可以对事件进行限制，比如.enter
3. v-on在绑定事件时可以传递自定义参数
4. 通过v-model可以快速的设置和获取表单元素的值
5. 基于数据的开发方式

```
<html>

<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <title>小黑记事本</title>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
  <meta name="robots" content="noindex,nofollow" />
  <meta name="googlebot" content="noindex,nofollow" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" type="text/css" href="./css/index.css" />
</head>

<body>
  <!-- 主体区域 -->
  <section id="todoapp">
    <!-- 输入框 -->
    <header class="header">
      <h1>小黑记事本</h1>
      <input v-model="inputValue" @keyup.enter="add" autofocus="autofocus"
        autocomplete="off" placeholder="请输入任务"
        class="new-todo" />
    </header>
    <!-- 列表区域 -->
    <section class="main">
      <ul class="todo-list">
        <li class="todo" v-for="(item, index) in list">
          <div class="view">
            <span class="index">{{ index+1 }}.</span>
            <label>{{ item }}</label>
            <button class="destroy" @click="remove(index)"></button>
          </div>
        </li>
      </ul>
    </section>
    <!-- 统计和清空 -->
    <footer class="footer" v-show="list.length!=0">
      <span class="todo-count" v-show="list.length!=0">
        <strong>{{list.length}}</strong> items left
      </span>
      <button v-show="list.length!=0" class="clear-completed" @click="clear">
        clear
      </button>
    </footer>
  </section>
</body>
```

```

        </footer>
    </section>
    <!-- 底部 -->
    <footer class="info">
        <p>
            <a href="http://www.itheima.com/"></a>
        </p>
    </footer>
    <!-- 开发环境版本，包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <script>
        var app = new Vue({
            el: "#todoapp",
            data: {
                list: ["写代码", "吃饭饭", "睡觉觉"],
                inputValue: "好好学习,天天向上"
            },
            methods: {
                add: function () {
                    this.list.push(this.inputValue);
                },
                remove: function (index) {
                    console.log("删除");
                    console.log(index);
                    this.list.splice(index, 1);
                },
                clear: function () {
                    this.list = [];
                }
            }
        })
    </script>
</body>

</html>

```



网络应用-介绍

1.Vue结合网络数据开发应用

2.axios-网络请求库

3.axios+vue-结合Vue一起

4.天气预报案例

网络应用- axios基本使用

基本应用

编写data.json

```
{  
    "success":true ,  
    "code" :20000,  
    "message":"成功",  
    "data":{  
        "items": [  
            {"name":"lucy", "age":20},  
            {"name":"mary", "age":100},  
            {"name":"jack", "age":200}  
        ]  
    }  
}
```

编写html和axios

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>  
<script>  
    new Vue({  
        el: '#app',  
        data: {  
  
            //定义变量。由于在json中写的是数组结构，我们也创建成数组结构  
            memberList: []  
        },  
        created(){  
            this.getList()  
        },  
        methods: {  
            getList(id) {  
                //axios.提交方式(请求的接口的路径或者地址).then().catch()  
                axios.get('data.json')  
                    .then(response => {//请求成功会执行then方法 (response是一个变  
                       量，里面封装着返回回来的数据)  
                        console.log(response)  
                        //把值赋值给memberList  
                        this.memberList = response.data.items  
                    })  
                    .catch(error => {//请求失败会执行catch方法 (error是一个变量  
                        名)  
                        console.log(error)  
                    })  
            }  
        }  
    })  
</script>
```

```
        }
    }
})
</script>
```

得到的结果

```
{data: {...}, status: 200, statusText: 'OK', headers: {...}, config: {...}, ...} ⓘ
▶ config: {transitional: {...}, transformRequest: Array(1), transformResponse: Array(1), timeout: 0, adapter: f, ...}
▶ data: {success: true, code: 20000, message: '成功', data: {...}}
▶ headers: {accept-ranges: 'bytes', access-control-allow-credentials: 'true', cache-control: 'public, max-age=0', conte
▶ request: XMLHttpRequest {onreadystatechange: null, readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttpRequest}
status: 200
statusText: "OK"
▶ [[Prototype]]: Object
```

是封装到data里面的

利用vue循环输出

```
<div id="app">
  <table border="1">
    <tr v-for="user in memberList">
      <td>{{user.name}}</td>
      <td>{{user.age}}</td>
    </tr>
  </table>
</div>
```

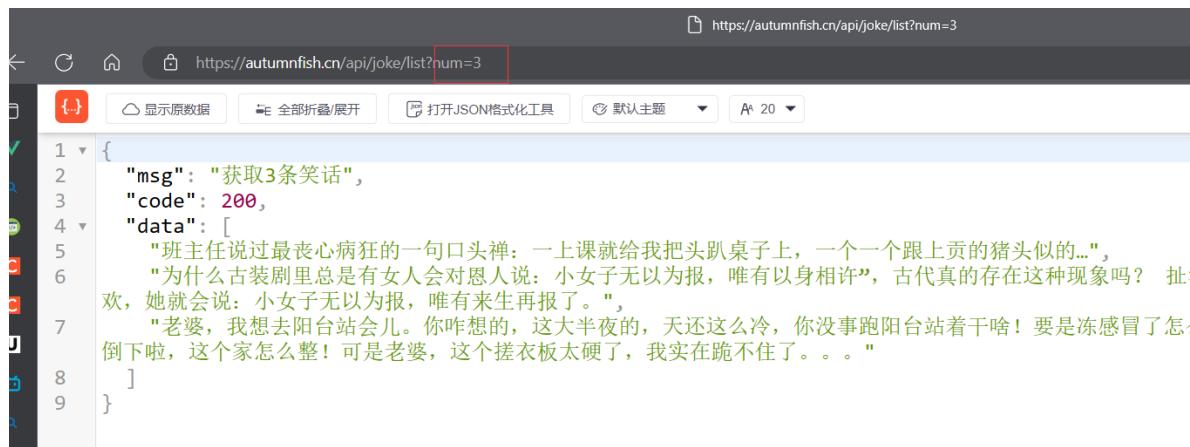
要点

1. axios: 功能强大的网络请求库
2. axios必须先导入才可以使用
3. 使用get或post方法即可发送对应的请求
4. then方法中的回调函数会在请求成功或失败时触发（请求成功是第一个函数，请求失败是执行的第二个函数）
5. 通过回调函数的形参可以获取响应内容,或错误信息
6. 文档传送门: <https://github.com/axios/axios>
7. axios官网文档: <http://www.axios-js.com/zh-cn/docs/>

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

模板:

```
axios.get(地址?key=value&key2=values).then(function(response){},function(err){})
axios.post(地址,{key:value,key2:value2}).then(function(response){},function(err){})
```



```
1 {  
2     "msg": "获取3条笑话",  
3     "code": 200,  
4     "data": [  
5         "班主任说过最丧心病狂的一句口头禅：一上课就给我把头趴桌子上，一个一个跟上贡的猪头似的...",  
6         "为什么古装剧里总是有女人会对恩人说：小女子无以为报，唯有以身相许”，古代真的存在这种现象吗？扯淡  
欢，她就会说：小女子无以为报，唯有来生再报了。",  
7         "老婆，我想去阳台站会儿。你咋想的，这大半夜的，天还这么冷，你没事跑阳台站着干啥！要是冻感冒了怎么  
倒下啦，这个家怎么整！可是老婆，这个搓衣板太硬了，我实在跪不住了。。。"  
8     ]  
9 }
```

第二个请求：



标头 载荷 预览 响应 启动器 时间

▼ 常规

请求网址: <https://autumnfish.cn/api/user/reg>

请求方法: POST

状态代码: 200 OK

远程地址: 127.0.0.1:7890

引荐来源网址政策: strict-origin-when-cross-origin

▼ 响应标头 查看源代码

Access-Control-Allow-Origin: *

Connection: keep-alive

Content-Length: 33

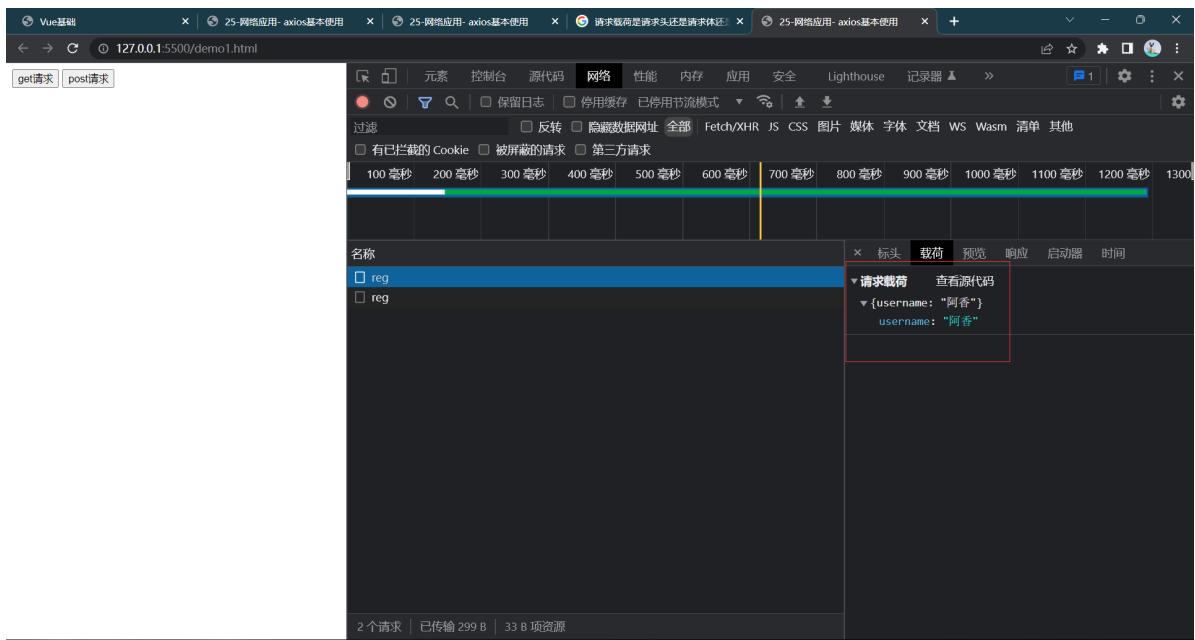
Content-Type: application/json; charset=utf-8

Date: Mon, 03 Apr 2023 03:12:25 GMT

ETag: W/\"21-RqM/Th93yZ8+pI8rDvL0w7zX0xQ"

Server: nginx/1.14.0

X-Powered-By: Express



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>25-网络应用- axios基本使用</title>
</head>
<body>
    <input type="button" value="get请求" class="get">
    <input type="button" value="post请求" class="post">
    <!-- 官网提供的 axios 在线地址 -->
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <script>
        /*
            接口1:随机笑话
            请求地址:https://autumnfish.cn/api/joke/list
            请求方法:get
            请求参数:num(笑话条数,数字)
            响应内容:随机笑话
        */
        document.querySelector(".get").onclick=function(){
            axios.get("https://autumnfish.cn/api/joke/list?num=3")
                .then(function(response){
                    console.log(response);
                })
                .catch(function(err){
                    console.log(err);
                })
        }
        /*
            接口2:用户注册
            请求地址:https://autumnfish.cn/api/user/reg
            请求方法:post
            请求参数:username(用户名,字符串)
            响应内容:注册成功或失败
        */
        document.querySelector(".post").onclick=function(){
            axios.post("https://autumnfish.cn/api/user/reg", {username: "阿香"})
        }
    </script>

```

```

        .then(function(response){
            console.log(response);
        },function(err){
            console.log(err);
        })
    }

</script>
</body>
</html>

```

The screenshot shows a browser window with the URL `127.0.0.1:5500/axios基本使用.html`. On the left, the page source code includes a script block with an `onClick` event handler for a 'get请求' button. This handler performs an `axios.get` request to `https://autumnfish.cn/api/joke/list?num=6`. The response is logged to the console, which is shown on the right. The response object contains a `jokes` array of 6 items, each a string joke.

```

axios基本使用.html > html > body > script > onclick
5   <meta charset="UTF-8" />
6     <meta name="viewport" content="width=device-width,
7       initial-scale=1.0" />
8     <meta http-equiv="X-UA-Compatible" content="ie=edge" />
9   <title>axios基本使用</title>
10
11 <body>
12   <input type="button" value="get请求" class="get">
13   <input type="button" value="post请求" class="post">
14   <!-- 官网提供的 axios 在线地址 -->
15   <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
16
17   /*
18     接口1:随机笑话
19     请求地址:https://autumnfish.cn/api/joke/list
20     请求方法: get
21     请求参数:num(笑话条数,数字)
22     响应内容:随机笑话
23   */
24   document.querySelector(".get").onclick = function () {
25     axios.get("https://autumnfish.cn/api/joke/list?num=6")
26       .then(function (response) {
27         console.log(response);
28       })
29   }
30
31   /*
32     接口2:笑话
33     请求地址:https://autumnfish.cn/api/joke/random
34     请求方法: post
35     请求参数:无
36   */

```

```

get请求 post请求
[{"date": {}, "status": 200, "statusText": "OK", "headers": {}, "config": {}, "data": {"jokes": [{"0": "今天看到老婆在睡午觉，看她汗出的很厉害，于是帮她开了空调，自己出去打麻将了。。。就刚打完牌，发现老婆今年50岁，既不太老，也不年轻，刚好是一个可以死去的年纪。", "1": "---摘自我弟弟的日记", "2": "那年我在服装厂，跟一个女孩子谈恋爱。一个小伙子明目张胆地接我，还给我下战书~接我，连输了两次。", "3": "3: \"哈，贫嘴从东土大闹而来，想向您化点东西！\"吧：\"哦，不知大师想化点什么？\"哈：\"请问女士，", "4": "4: \"下课了，老师叫住小明问她：\"你上课为什么不积极回答问题。\"小明说：\"我害怕怕什么？", "5": "5: \"刚才在网上玩穿越火线，见到一个人的昵称叫“其他昵称”，我问他你的昵称怎么这么怪呢？他说：“", "length": 6}, "msg": "获取6条笑话", "proto": {}]}

```

网络应用- axios与vue结合使用

- 1.axios回调函数中的this已经改变,无法访问到data中数据
- 2.把this保存起来,回调函数中直接使用保存的this即可
- 3.和本地应用的最大区别就是改变了数据来源

```

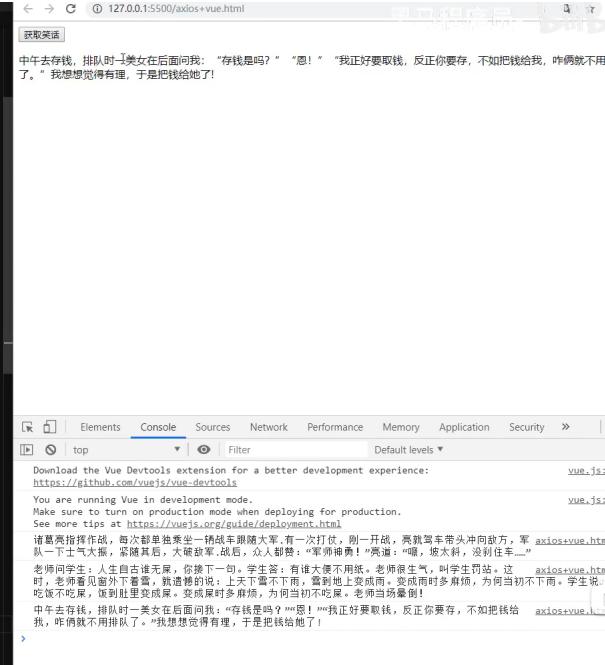
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>26-网络应用- axios与vue结合使用</title>
</head>
<body>
    <div id="app">
        <input type="button" value="获取笑话" @click="getJoke">
        <p>{{joke}}</p>
    </div>
    <!-- 开发环境版本, 包含了有帮助的命令行警告 -->
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    <!-- 官网提供的 axios 在线地址 -->
    <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
    <script>

```

```

/*
    接口:随机获取一条笑话
    请求地址:https://autumnfish.cn/api/joke
    请求方法:get
    请求参数:无
    响应内容:随机笑话
*/
var app=new Vue({
    el:"#app",
    data:{
        joke:"笑话显示区域",
    },
    methods:{
        getJoke:function(){
            console.log(this.joke);
            var that=this;
            axios.get("https://autumnfish.cn/api/joke")
            .then(function(response){
                console.log(response);
                console.log(response.data);
                console.log(this.joke);
                that.joke=response.data;
            }),function(err){
                console.log(err);
            })
        }
    }
})
</script>
</body>
</html>

```



The screenshot shows the browser's developer tools open at the address `127.0.0.1:5500/axios+vue.html`. The console tab displays the following output:

```

中牛去存钱，排队时一美女在后面问我：“存钱是吗？”“恩！”“我正好要取钱，反正你要存，不如把钱给我，咋俩就不用排了。”我想觉得有理，于是把钱给她了！

```

The code in the browser's developer tools matches the code in the previous snippet, with some parts highlighted in red and yellow to indicate specific sections of the code or its output.

网络应用-天知道

1. 按下回车(v-on .enter)
 2. 查询数据(axios 接口 v-model)
 3. 渲染数据(v-for 数组 that)
- 应用的逻辑代码建议和页面分离，使用单独的js文件编写
 axios回调函数中this指向改变了，需要额外的保存一份
 服务器返回的数据比较复杂时，获取的时候需要注意层级结构

```

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

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>天知道</title>
  <link rel="stylesheet" href="css/reset.css" />
  <link rel="stylesheet" href="css/index.css" />
</head>

<body>
  <div class="wrap" id="app">
    <div class="search_form">
      <div class="logo"></div>
      <div class="form_group">
        <input type="text" v-model="city" @keyup.enter="searchweather"
               class="input_txt" placeholder="请输入查询的天气"/>
        <button class="input_sub" @click="searchweather">
          搜 索
        </button>
      </div>
      <div class="hotkey">
        <a href="javascript:;" @click="changeCity('北京')">北京</a>
        <a href="javascript:;" @click="changeCity('上海')">上海</a>
        <a href="javascript:;" @click="changeCity('广州')">广州</a>
        <a href="javascript:;" @click="changeCity('深圳')">深圳</a>
      </div>
    </div>
    <ul class="weather_list">
      <li v-for="item in weatherList">
        <div class="info_type"><span class="iconfont">{{ item.type }}</span>
      </div>
        <div class="info_temp">
          <b>{{ item.low }}</b>
          ~
          <b>{{ item.high }}</b>
        </div>
        <div class="info_date"><span>{{ item.date }}</span></div>
      </li>
    </ul>
  </div>
  <!-- 开发环境版本，包含了有帮助的命令行警告 -->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
  <!-- 官网提供的 axios 在线地址 -->
  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
  <!-- 自己的js -->

```

```
<script src="./js/main.js"></script>
</body>

</html>
```

```
var app=new Vue({
  el:"#app",
  data:{
    city:'',
  },
  methods:{
    searchWeather:function(){
      var that =this;
      axios.get('http://wthrcdn.etouch.cn/weather_mini?city='+this.city)
      .then(function(response){
        that.weatherList=response.data.data.forecast;
      })
      .catch(function(err){})
    },
    changeCity:function(city){
      this.city=city;
      this.searchWeather();
    }
  }
})
```

综合应用（音乐播放器）

介绍：

- 1.歌曲搜索
- 2.歌曲播放（点击按钮播放）
- 3.歌曲封面
- 4.歌曲评论
- 5.播放动画
- 6.mv播放

综合应用-歌曲搜索

- 1.按下回车(v-on .enter)
- 2.查询数据(axios 接口 v-model)
- 3.渲染数据(v-for 数组 that)

综合应用-歌曲播放

- 1.点击播放(v-on 自定义参数)

点击播放按钮：播放歌曲的本质就是设置歌曲的src，切换歌曲就是更换不同的src，歌曲的地址从network查看，歌曲地址是通过接口获取到的，获取歌曲地址后找到歌曲播放地址，将播放地址存到data的musicUrl字段中，再传给给audio标签；

注：点击时需要传入参数，从接口获得的歌曲的点击事件才会才会被绑定。

- 2.歌曲地址获取：

根据接口确定一个传递的参数（歌曲id），搜索出的歌曲时服务器返回的结果数组中每一项都有歌

曲id，不同歌曲id不同，但查询逻辑是一样的；（总：接口调用，把所需的参数传过去）

3. 歌曲地址设置(v-bind)

data中增加歌曲属性，歌曲id依赖与歌曲的搜索结果，v-bind绑定到播放条中。

综合应用-歌曲封面

综合应用-播放动画

1. 监听音乐播放(v-on play)

核心：增删一个类

播放时碟片旋转，暂停时停时旋转，检测动画的播放状态，在对应的事件中增删类名，

2. 监听音乐暂停(v-on pause)

3. 操纵类名(v-bind 对象)

audio标签的play事件会在音频播放的时候触发

audio标签的pause事件会在音频暂停的时候触发

通过对对象的方式设置类名，类名生效与否取决于后面值的真假

综合应用-播放mv

1. mv图标显示(v-if)

2. mv地址获取

3. 遮罩层

4. mv地址设置

21-综合应用（音乐播放器）-介绍

注：

综合应用-音乐播放器

代码地址：

介绍

1.歌曲搜索

2.歌曲播放（点击按钮播放）

3.歌曲封面

4.歌曲评论

5.播放动画

6.mv播放

综合应用-歌曲搜索

1. 按下回车(v-on .enter)

2. 查询数据(axios 接口 v-model)

3. 渲染数据(v-for 数组 that)

23-综合应用-歌曲播放

4. 点击播放(v-on 自定义参数)

点击播放按钮：播放歌曲的本质就是设置歌曲的src，切换歌曲就是更换不同的src，歌曲的地址从network查看，歌曲地址是通过接口获取到的，获取歌曲地址后找到歌曲播放地址，将播放地址存到data的musicUrl字段中，再传给给audio标签；

注：点击时需要传入参数，从接口获得的歌曲的点击事件才会才会被绑定。

5. 歌曲地址获取：

根据接口确定一个传递的参数（歌曲id），搜索出的歌曲时服务器返回的结果数组中每一项都有歌曲id，不同歌曲id不同，但查询逻辑是一样的；（总：接口调用，把所需的参数传过去）

6. 歌曲地址设置(v-bind)

data中增加歌曲属性，歌曲id依赖与歌曲的搜索结果，v-bind绑定到播放条中。

综合应用-歌曲封面

综合应用-播放动画

1. 监听音乐播放(v-on play)

核心：增删一个类

播放时碟片旋转，暂停时停时旋转，检测动画的播放状态，在对应的事件中增删类名，

2. 监听音乐暂停(v-on pause)

3. 操纵类名(v-bind 对象)

audio标签的play事件会在音频播放的时候触发

audio标签的pause事件会在音频暂停的时候触发

通过对象的方式设置类名，类名生效与否取决于后面值的真假

综合应用-播放mv

1. mv图标显示(v-if)

2. mv地址获取

3. 遮罩层

4. mv地址设置

Vue3

创建项目

vue create vue-demo

启动时显示欢迎页

问题 输出 调试控制台 终端

Vue CLI v5.0.8
Failed to check for updates
? Please pick a preset:
Default ([Vue 3] babel, eslint)
Default ([Vue 2] babel, eslint)
> Manually select features

问题 1 输出 调试控制台 终端 +

Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
(*) Babel
() TypeScript
>(*) Progressive Web App (PWA) Support
() Router
() Vuex
() CSS Pre-processors
() Linter / Formatter
() Unit Testing
() E2E Testing

选项

然后安装

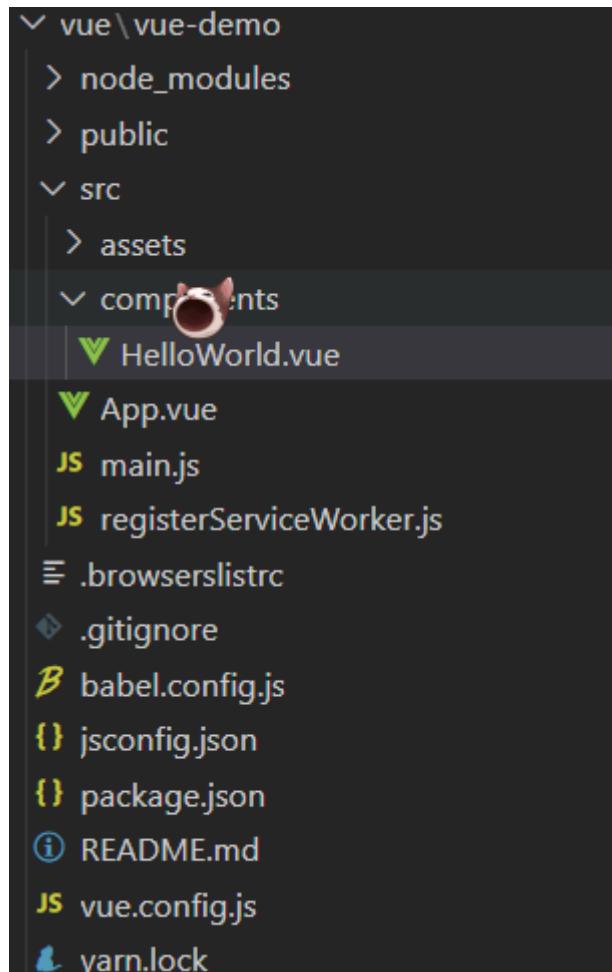
运行：

1. 先进入你要的目录 cd vue-demo
2. npm run serve

Get started with the following command
ds:
\$ cd vue-demo
\$ yarn serve

或者根据

然后生成这个



组件模板

components

The screenshot shows the Vetur IDE interface with the file `MyComponent.vue` open. The code editor displays the following template:

```
<template>
</template>
<script>
export default {
}
</script>
<style>
</style>
```

The code editor has three red boxes highlighting specific parts of the template:

- A red box surrounds the entire `<template>` block.
- A red box surrounds the `<script>` block, which contains the export statement.
- A red box surrounds the `<style>` block.

可以自定义组件

那么如何引入组件呢?

The screenshot shows the VS Code interface with the following file structure:

- Resource Manager: Shows a tree view of files and folders.
- Editor: Displays the `App.vue` file content. The code includes imports for `HelloWorld` and `MyComponent`, and a template section with both components.
- Bottom Bar: Shows tabs for '问题' (Issues), '输出' (Output), '调试控制台' (Debug Console), and '终端' (Terminal).

The code in `App.vue`:

```
<template>
  
  <HelloWorld msg="Welcome to Your Vue.js App"/>
  <MyComponent></MyComponent>
</template>

<script>
  import HelloWorld from './components/HelloWorld.vue'
  import MyComponent from './components/MyComponent'

  export default {
    name: 'App',
    components: {
      HelloWorld,
      MyComponent
    }
  }
</script>

<style>
  #app {
    font-family: Avenir, Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
    margin-top: 60px;
  }
</style>
```

这样就可以引入啦

还可以这样写

The screenshot shows the same `App.vue` file with a different component import and usage:

```
<template>
  <HelloWorld msg="Welcome to Your Vue.js App"/>
  <MyComponent></MyComponent>
  <my-component></my-component>
</template>

<script>
  import HelloWorld from './components/HelloWorld.vue'
  import MyComponent from './components/MyComponent.vue'

  export default {
    name: 'App',
    components: {
      HelloWorld,
      MyComponent
    }
  }
</script>
```

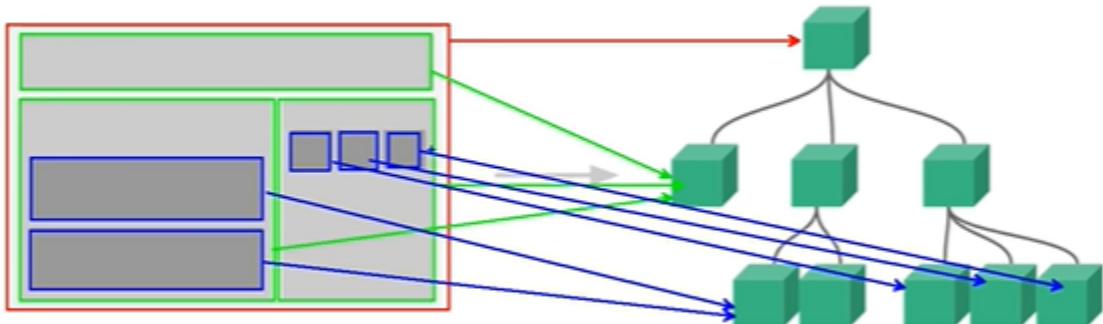
A red box highlights the line `<my-component></my-component>`, and the text "这两种都可以的" (Both ways are acceptable) is overlaid on the right side of the editor.

需要注意的是：

The screenshot shows a code block with a note about the `scoped` attribute:

```
<!-- scoped: 如果在style中添加此属性，就代表着，当前样式，只在当前组件中生效 -->
<style scoped>
  h3{
    color: red;
  }
</style>
```

组件树!



Props组件交互

组件交互其实就是组件之间可以进行数据传输，起到一个数据公共的情况

实现组件交互

文件结构:

components	•
└ HelloWorld.vue	M
└ MyComponent.vue	U
└ Prop.vue	1, U
└ App.vue	1, M

代码:

```
<template>
  <!-- title是传过去的参数的key, t是本页面的属性，不过最好写一样 -->
  <Prop :title="t" :age="a" :names="names"/>
</template>

<script>
import Prop from './components/Prop'
export default {
  name: 'App',
  data(){
    return{
      t:"t标题在这",
      a:20,
      names:["haohao", "yaoyao"]
    }
  },
  components:{
    Prop
  }
}
</script>
```

```
<template>
  <h1>{{ title }}</h1>
  <h1>{{ age }}</h1>
```

```
<h1>{{ names }}</h1>
</template>
<script>
export default{
  props:{
    title:{
      type:String,
      default:"hahh"
    },
    age:{
      type:Number,
      default:0
    },
    names:{
      type:Array,
      //数组和对象需要使用函数来进行返回
      default:function(){
        return []
      }
    }
  }
}
</script>
```

页面效果：



t标题在这

20

["haohao", "yaoyao"]

自定义事件的组件交互

自定义事件可以在组件中反向传递数据，`prop`可以将数据从父组件传递到子组件，那么反向如何操作呢？答案是：可以利用自定义事件实现`$emit`

现在是从工具人向老板传递数据

```
<template>
  <!-- 需要注意的是 getData是下面的方法，不能加括号-->
  <!--并且这个haohaoEvent要注意相对应-->
  <Prop @haohaoEvent="getData"></Prop>
  <h1>{{ mess }}</h1>
</template>

<script>
import Prop from './components/Prop'
export default {
  name: 'App',
  data() {
```

```
return{
  mess:""
}
},
components:{
  Prop
},
methods:{
  getData(data){
    this.mess=data;
  }
}
}
</script>
```

```
<template>
<!-- 需要注意的是 sendData是下面的方法，不能加括号--&gt;
&lt;button @click="sendData"&gt;必须要点击才能发送数据&lt;/button&gt;
&lt;/template&gt;
&lt;script&gt;
export default{
  data(){
    return{
      mess:"我是传输过来的数据"
    }
  },
  methods:{
    sendData(){
      //参数1：字符串，理论上是随便的，但是需要有对应关系
      //参数2，传递的数据
      //这里的$emit是固定的
      this.$emit("haohaoEvent",this.mess)
    }
  }
}
&lt;/script&gt;</pre>
```

效果：



点击按钮后



我是传输过来的数据

组件生命周期

每个组件在创建时都要经过一系列的初始化过程，在这个过程中会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会。

一共有八个生命周期钩子函数...之前用到过

为了方便记忆，我们可以将他们分类：

*
创建时：`beforeCreate`、`created`

渲染时：`beforeMount`、`mounted`

更新时：`beforeUpdate`、`updated`

卸载时：`beforeUnmount`、`unmounted`

网络请求就放到mounted就行了。

Vue引入第三方

awesome-vue, 一些vue的第三方组件.

[GitHub - vuejs/awesome-vue: 🎨 A curated list of awesome things related to Vue.js](https://github.com/vuejs/awesome-vue)

下面介绍swiper

安装swiper `cnpm install --save swiper`

使用

```
<template>
  <div>
    <Swiper>
      <Swiperslide>
        
      </Swiperslide>
      <Swiperslide>
        
      </Swiperslide>
      <Swiperslide>
        
      </Swiperslide>
    </Swiper>
  </div>
</template>
<script>
import { Swiper, Swiperslide } from 'swiper/vue';
import 'swiper/css';

export default{
  name:'SwiperDemo',
  components:{
    Swiper,
    Swiperslide
  }
}
</script>
```

在Vue中使用axios

首先切换到你的vue项目中安装axios `npm install --save axios`

如果报错试试 `npm install --save axios --location=global`

get请求

```
<template>
  
  <h1>{{ data.title }}</h1>
</template>

<script>
```

```

import axios from "axios"

export default {
  name: 'App',
  data(){
    return{
      data:[]
    }
  },
}

mounted(){
  axios({
    //get请求....
    method:"get",
    url:"http://iwenwiki.com/api/blueberrypai/getChengpinDetails.php"
  }).then(res=>{
    this.data=res.data.chengpinDetails[0]
  })
}
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

效果：



蓝莓派Impie.com 早餐牛奶杯创意咖啡杯陶瓷马克杯 蓝莓诚品

Post请求

先安装转换字符串格式的东西 `cnpm install --save querystring`

```
<template>
  
</template>

<script>

import axios from "axios"
import querystring from "querystring"

export default {
  name: 'App',
  data(){
    return{
      data:[]
    }
  },
  mounted(){
    axios({
      method:"post",
      url:"http://iwenwiki.com/api/blueberrypai/login.php",
      data:querystring.stringify({
        user_id:"iwen@qq.com",
        password:"iwen123",
        verification_code:"crfvw"
      })
    }).then(res=>{
      console.log(res.data)
    })
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

得到结果



标识项目的根文件夹以打开Visual Studio Code中的源文件并同步更改。 [设置根文件夹](#) [不再显示](#)

元素 控制台 源代码 网络 应用程序 欢迎 +

top 筛选器 默认级别 5

```
▼ {success: true, msg: '', msg_code: ''} ⓘ
  msg: ""
  msg_code: ""
  success: true
  ▼ [[Prototype]]: Object
    ► constructor: f Object()
    ► hasOwnProperty: f hasOwnProperty()
    ► isPrototypeOf: f isPrototypeOf()
    ► propertyIsEnumerable: f propertyIsEnumerable()
    ► toLocaleString: f toLocaleString()
    ► toString: f toString()
    ► valueOf: f valueOf()
    ► __defineGetter__: f __defineGetter__()
    ► __defineSetter__: f __defineSetter__()
    ► __lookupGetter__: f __LookupGetter__()
    ► __lookupSetter__: f __LookupSetter__()
```

简化方式

```
<template>
  
</template>

<script>
  import axios from "axios"
  import queryString from "querystring"

  export default {
    name: 'App',
    data(){
      return{
        data:[]
      }
    },
    mounted(){
      axios.get("http://iwenwiki.com/api/blueberrypai/getChengpinDetails.php")
        .then(res=>{
          console.log(res);
        })
    }
  }
}
```

```
axios.post("http://iwenwiki.com/api/blueberrypai/login.php", querystring.stringify({
    user_id:"iwen@qq.com",
    password:"iwen123",
    verification_code:"crfvw"
})).then(res=>{
    console.log(res)
})
}
}
</script>
```

```
<style>
#app {
    font-family: Avenir, Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
    margin-top: 60px;
}
</style>
```

全局引入axios

文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) ... ← → 🔍 qune

资源管理器 ...

> 打开的编辑器

QUNEE

- qunee
 - > data
 - > demo
 - > images
 - > js
- vue\vue-demo
 - > node_modules
 - > public
 - > src
 - > assets
 - logo.png
 - > components
 - HelloWorld.vue M
 - MyComponent.vue U
 - Prop.vue U
 - SwiperDemo.vue U
 - App.vue M
 - main.js M
 - registerServiceWorker.js

qunee > vue > vue-demo > src > main.js > ...

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import './registerServiceWorker'
4 import axios from "axios"
5
6
7 const app=createApp(App)
8 app.config.GlobalProperties.$axios=axios
9 app.mount('#app')
10
```

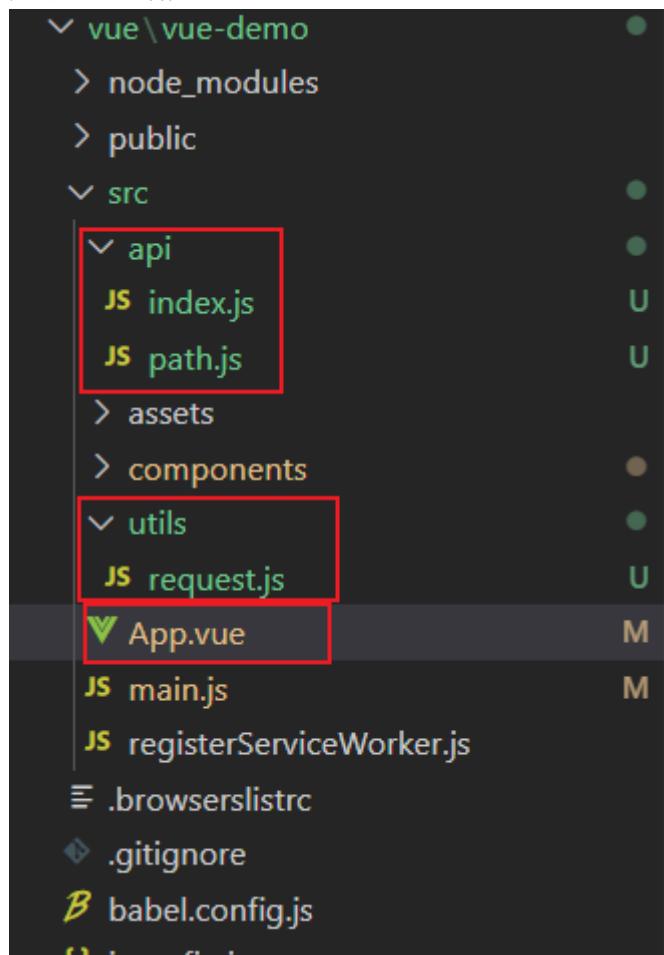
之后就可以这样调用了：

```
7
8 mounted(){
9     this.$axios.get("http://iwenwiki.com/api/blueberrypai/getChengpinDetails.php")
10    .then(res=>{
11        console.log(res);
12    })
13    this.$axios.post("http://iwenwiki.com/api/blueberrypai/login.php", queryString.stringify({
14        user_id:"iwen@qq.com",
15        password:"iwen123",
16        verification_code:"crfvw"
17    })).then(res=>{
18        console.log(res)
19    })
20}
```

具体是为啥老师也没讲

Axios网络请求封装

把网络请求进行封装，更好找，有利于维护
用到的目录结构



代码:

```
import axios from 'axios'
import queryString from "querystring"

//instance1 是一个 axios 实例化对象，通过调用 create() 方法创建，可以设置一些请求的配置项，比如超时时间。
const instance1=axios.create({
  //设置超时时间
  timeout:5000
})

//通过 interceptors 对象，  

//我们可以在数据发送前和数据接收后对数据进行一些拦截和处理操作，比如将 post 请求的数据格式转化为 querystring 格式，  

//或者在返回结果时判断返回状态码是否为 200。

//发送数据之前
instance1.interceptors.request.use(
  config=>{
    if(config.method==="post"){
      config.data=querystring.stringify(config.data)
    }
  }
)
```

```

    //config中包含着网络请求的所有信息
    return config;
},
error=>{
    return Promise.reject(error)
}
)
instance1.interceptors.response.use(
    response=>{
        return response.status==200?
Promise.resolve(response):Promise.reject(response);
},
error=>{
    console.log("error")
}
)

```

//通过 export default 将 instance1 对象导出供其他模块使用。

```

export default instance1;

```

```

const base={
    baseUrl:"http://iwenwiki.com",
    chengpin:"/api/blueberrypai/getChengpinDetails.php"
}
//导出base
export default base;

```

```

import axios from "../utils/request"
import path from "./path"
const api={
    getChengpin(){
        return axios.get(path.baseUrl+path.chengpin);
    }
}
export default api;

```

```

<template>
    
</template>

```

```

<script>

import api from "./api/index"
export default {
    name: 'App',
    mounted(){
        api.getChengpin().then(res=>{
            console.log(res.data)
        })
    }
}

```

```

        }
    }
</script>

<style>
#app {
    font-family: Avenir, Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
    margin-top: 60px;
}
</style>

```

结果：

```

App.vue?91a0:13
{
  success: true,
  msg: '',
  chengpinDetails: Array(1)
}
  0: {title: '蓝莓系简约风 早餐牛奶杯创意咖啡杯陶瓷马克杯 蓝莓诚品', content: '马克杯的意思是大柄杯子，因为马克杯的英文叫mug，所以翻译成马克杯。马克杯是家常杯子的一种，一般用...并且杯身的一侧带有把手。马克杯的...', length: 1}

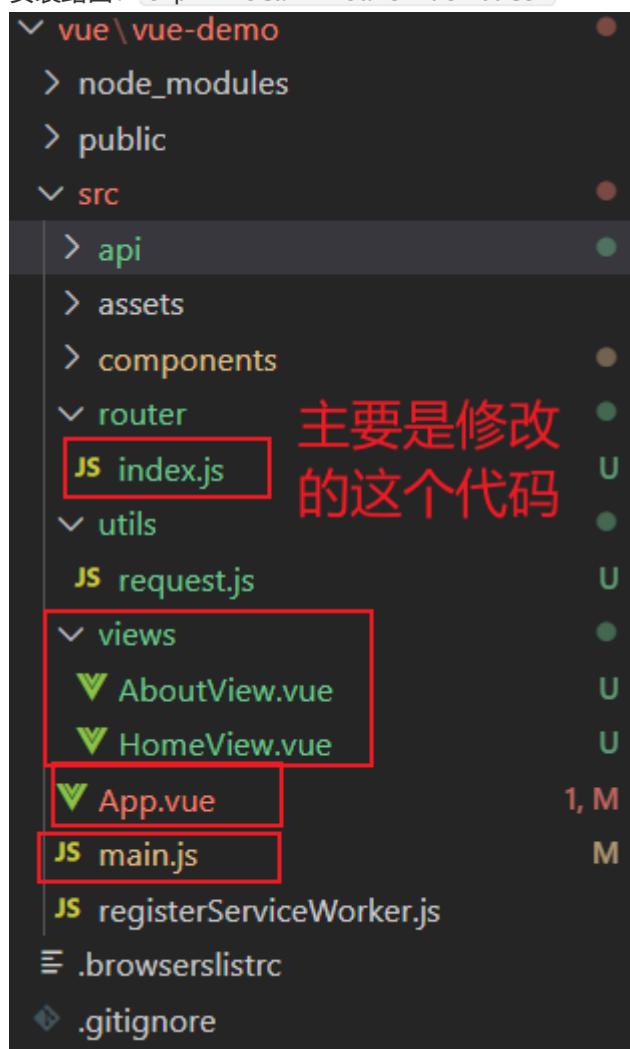
```

路由

通过路由的方式管理页面之间的关系

Vue Router 是 Vue 的官方路由！

安装路由！`cnpm install --save vue-router`



其中不用新创建的是 App.vue 和 main.js

```
<template>
    <h1>首页</h1>
</template>
```

```
<template>
    <h1>about页面</h1>
</template>
```

```
import {createRouter,createWebHashHistory} from 'vue-router'
import HomeView from "../views/HomeView"
import AboutView from "../views/AboutView"

const routes=[
{
    path:"/",
    component:HomeView
},
{
    path:"/about",
    // component:AboutView
    // 也可以下面这样写
    // 这样的是异步加载的方式，性能比较好
    component:()=>import('../views/AboutView.vue')
}
]
const router=createRouter({
    history:createWebHashHistory(),
    routes
})
//导出
export default router;
```

```
import { createApp } from 'vue'
import App from './App.vue'
import './registerServiceworker'
import axios from "axios"
import router from "./router"

const app=createApp(App)
//再主入口用路由，用.use的方式明确路由功能，
//这里添加的只有两句，一句是下面这个，一句是上面的import语句
app.use(router)
app.config.globalProperties.$axios=axios
app.mount('#app')
```

```
<template>
<router-link to="/">首页</router-link>
<br/>
```

```

<router-link to="/about">about</router-link>

<!-- 路由的样式就在这里显示 --&gt;
&lt;router-view&gt;&lt;/router-view&gt;
&lt;/template&gt;

&lt;script&gt;
// 这里啥都可以不写哈哈哈哈

export default {
}
&lt;/script&gt;
</pre>

```

需要注意的是：

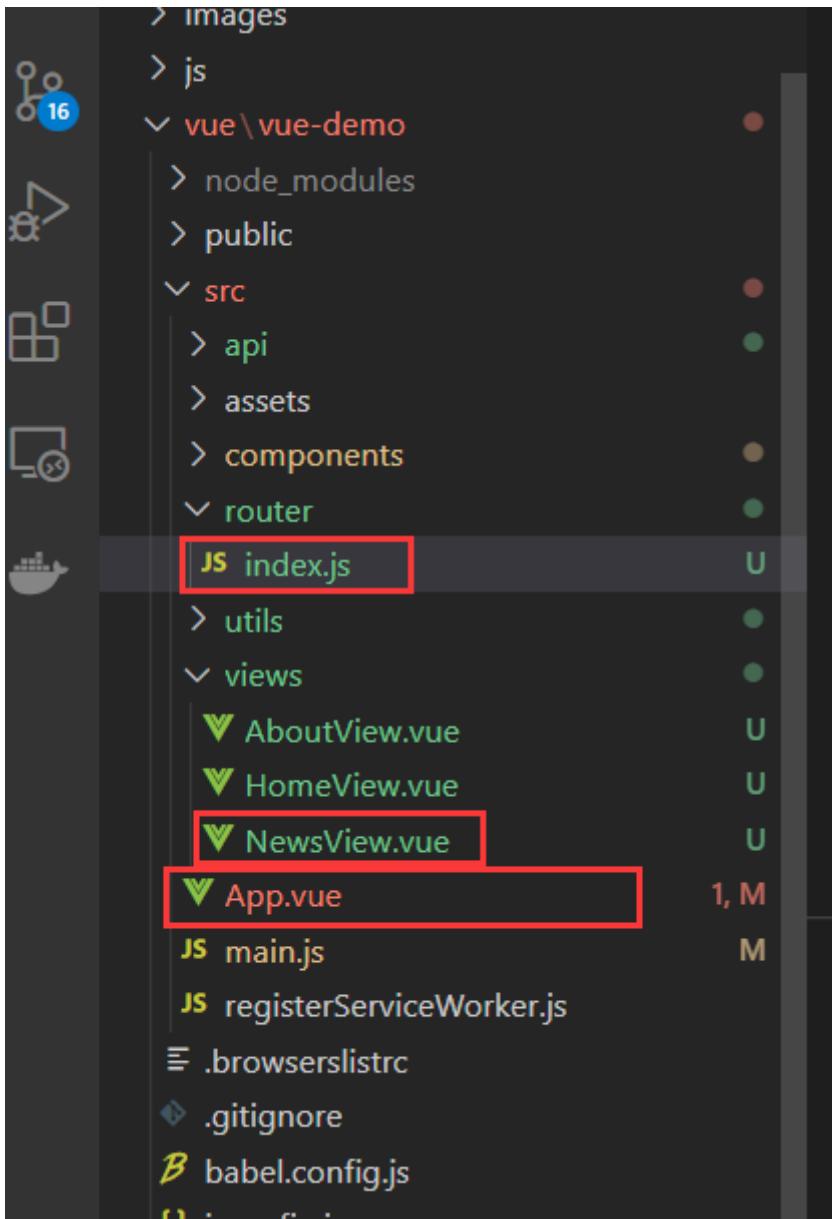
```

19 const router = createRouter({
20   /**
21    * createWebHashHistory
22    *   home:http://localhost:8080/#/
23    *   about:http://localhost:8080/#/about
24    *
25    * 原理：a标签锚点连接
26    */
27   /**
28    * createWebHistory
29    *   home:http://localhost:8080/
30    *   about:http://localhost:8080/about
31    * 此种方式，需要后台配合做重定向，否则会出现404问题
32    *
33    * 原理：H5 pushState()
34    */
35   history: createWebHashHistory(),
36   routes
37 })
38

```

路由传递参数

目录结构:



代码实现:

```
import {createRouter, createWebHashHistory} from 'vue-router'
const routes=[
    //第一步，在路由配置种指定参数的key，注意这个name和后面详情页的name相对应
    {
        path:"/list/:name",
        name:"list",
        component:()=>import("../views/NewsView.vue")
    }
]
const router=createRouter({
    history:createWebHashHistory(),
    routes
})
//导出
export default router;
```

```
<!-- 第二步：在跳转过程中携带参数 -->
<template>
    <li><router-link to="/list/网易新闻">网易新闻</router-link></li>
```

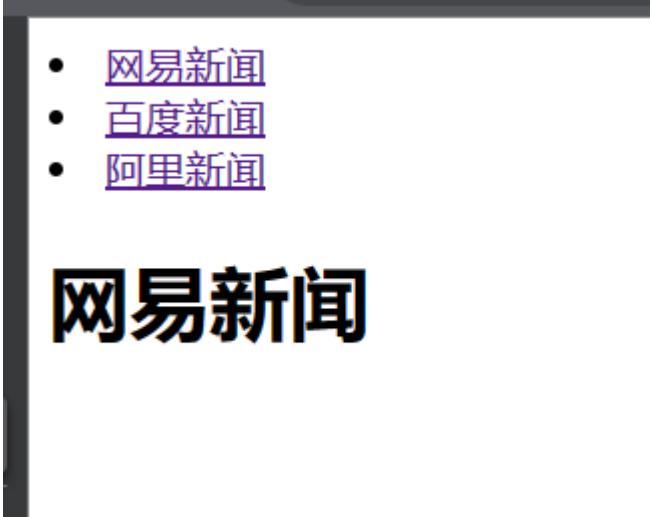
```
<li><router-link to="/list/百度新闻">百度新闻</router-link></li>
<li><router-link to="/list/阿里新闻">阿里新闻</router-link></li>
<!-- 路由的样式就在这里显示 --&gt;
&lt;router-view&gt;&lt;/router-view&gt;
&lt;/template&gt;</pre>
```

```
<script>
```

```
export default {
}
</script>
```

```
<!-- 第三步：在详情页中获取数据 -->
<template>
    <h1>{{ $route.params.name }}</h1>
</template>
```

效果：

- 
- [网易新闻](#)
 - [百度新闻](#)
 - [阿里新闻](#)

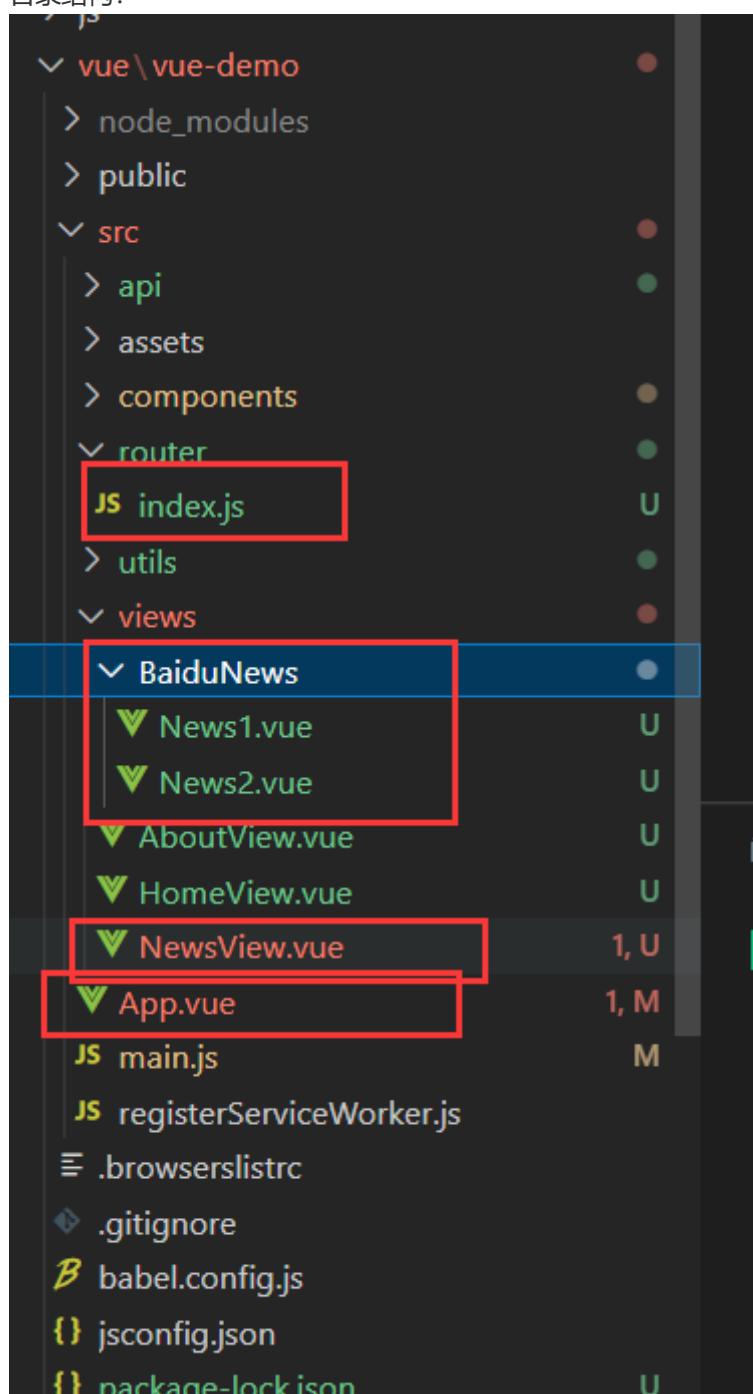
网易新闻

嵌套路由配置

嵌套路由就是这玩意：

The screenshot shows a web-based management interface for a pharmaceutical company. On the left, there's a sidebar with categories like '基本信息' (Basic Information), '平台财务管理' (Platform Financial Management), '小程序' (Microapp), and '资源'. Under '资源', there's a '优惠券' (Coupon) section. The main area is titled '发放列表' (Distribution List) and displays a table of coupon distributions. The table columns include: 操作 (Operation), 优惠券发放ID (Coupon Distribution ID), 优惠券名称 (Coupon Name), 发放时间 (Release Time), 发放场景 (Release Scenario), 领取数量 (Number of Acquisitions), 有效时间 (Validity Period), 状态 (Status), and 备注 (Remarks). There are 27 rows of data, with the last row being '共 27 条 20条/页 < 1 2 > 前往 1 页'.

目录结构：



```
... package.json
```

```
<template>
  <router-link to="/list">新闻</router-link>
  <!-- 路由的样式就在这里显示 -->
  <router-view></router-view>
</template>
<script>
export default {
}
</script>
```

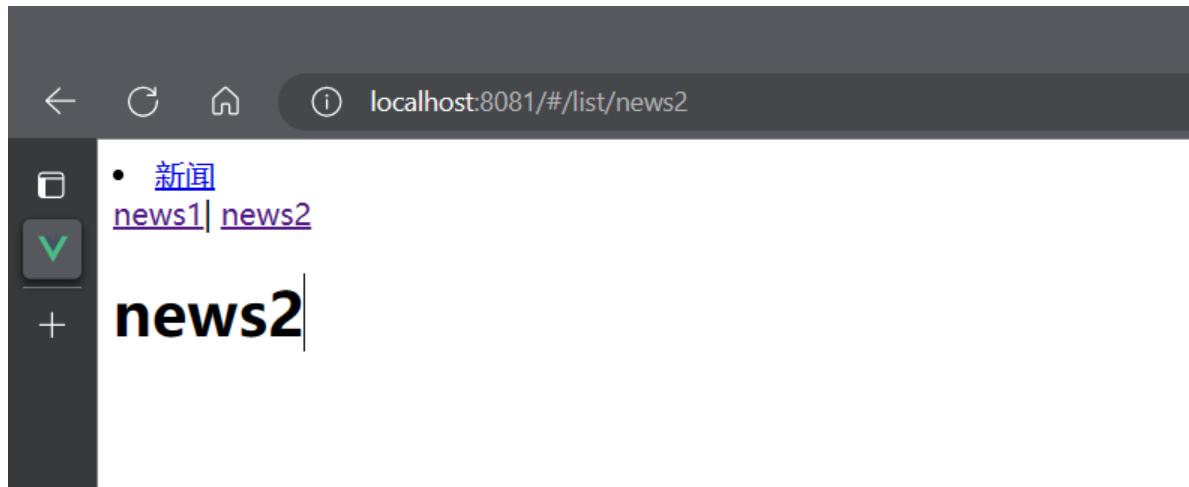
```
<template>
  <router-link to="/list/news1">news1</router-link>|
  <router-link to="/list/news2">news2</router-link>
  <router-view></router-view>
</template>
```

```
import {createRouter, createWebHashHistory} from 'vue-router'
const routes=[
  {
    path:"/list",
    name:"list",
    component:()=>import("../views/NewsView.vue"),
    //重定向， 默认访问子页面的那个路由
    redirect:"/list/news1",
    children:[
      {
        //注意这个不是文件名称， 不要加杠
        path:"news1",
        component:()=>import("../views/BaiduNews/News1.vue"),
      },
      {
        //注意这个不是文件名称
        path:"news2",
        component:()=>import("../views/BaiduNews/News2.vue"),
      }
    ]
  }
]
const router=createRouter({
  history:createWebHashHistory(),
  routes
})
//导出
export default router;
```

```
<template>
  <h1>news1</h1>
</template>
```

```
<template>
  <h1>news2</h1>
</template>
```

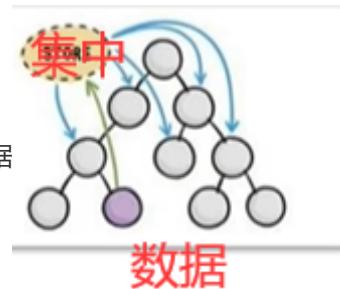
页面效果：



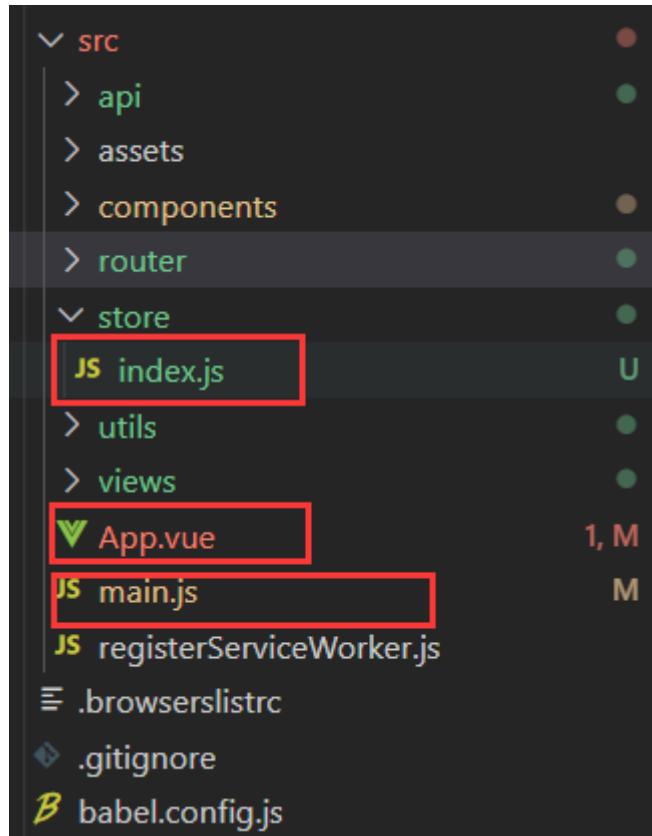
Vue状态管理 (Vuex)

状态管理可以理解成：为了更方便的管理组件之间的数据交互，提供了一个**集中式**的管理方案，任何组

件都可以按照指定的方式进行读取和改变数据



项目结构:



第一步:

安装vuex `npm install --save vuex`

第二步:

创建store目录，并且创建一个地方存放所有数据

```
import { createStore } from "vuex"
//vuex的核心作用就是帮我们管理组件之间的状态/数据
const store = createStore({
    //所有的数据都放在这里
    state: {
        counter: 0
    }
})
export default store;
```

第三步:

在main.js中引入vuex

```
import { createApp } from 'vue'
import App from './App.vue'
import './registerServiceWorker'
//添加vuex
import store from "./store"

const app=createApp(App)
//应用vuex
app.use(store)
app.config.globalProperties.$axios=axios
app.mount('#app')
```

第四步：

在主页面获取数据

方法一：

```
<template>
    <!-- 通过这种方式取到变量 -->
    <h1>{{ $store.state.counter }}</h1>
</template>

<script>
export default {
}
</script>
```

方法二：

```
<template>
    <!-- 通过这种方式取到变量 -->
    <h1>{{ counter }}</h1>
</template>

<script>
import {mapState} from "vuex"
export default {
    //这个方法专门用来读取vuex的数据
    computed:{
        ...mapState(["counter"])
    }
}
</script>
```

Vue状态管理核心 (Vuex)

getters的方法

```
import {createStore} from "vuex"
//vuex的核心作用就是帮我们管理组件之间的状态/数据
const store =createStore({
    //所有的数据都放在这里
    state:{
        counter:0
    },
    getters:{
        getCounter(state){
            return state.counter>0?state.counter:"小于等于0了的效果"
        }
    }
})
export default store;
```

他也有两种获取方法

```

<template>
<!-- 第一种方式 -->
<h1>{$store.getters.getCounter}</h1>
<!-- 第二种方式，需要导入mapGetters -->
<h1>{getCounter}</h1>
</template>
<script>
import {mapGetters} from "vuex"
export default {
  //这个方法专门用来读取vuex的数据
  computed: {
    ...mapGetters(["getCounter"])
  }
}
</script>

```

Mutation

更改Vuex的store中的状态的唯一方法是提交mutation,vuex中的mutation非常类似于事件:每个mutation都有一个字符串的事件类型type和一个回调函数(handler)。这个回调函数就是我们实际进行状态更改的地方，并且它会接受state作为第一个参数

```

import { createStore } from "vuex"

//vuex的核心作用就是帮我们管理组件之间的状态/数据
const store = createStore({
  //所有的数据都放在这里
  state: {
    counter: 10
  },
  mutations: {
    addCounter(state, num) {
      state.counter = state.counter + num;
    }
  }
})

export default store;

```

```

<template>
  <button @click="addClickHandler">+10</button>
</template>

<script>
import { mapState, mapGetters } from "vuex"

export default {
  methods: {
    addClickHandler() {

```

```
        this.$store.commit("addCounter",10)
    }
}
</script>
```

Action

```
import {createStore} from "vuex"
import {axios} from "axios"

//vuex的核心作用就是帮我们管理组件之间的状态/数据
const store =createStore({
    actions:{
        asyncAddCounter({commit}){
            axios.get("").then(res=>{
                commit("addCounter",res.data[0])
            })
        }
    }
})

export default store;
```

Vue3新特性

主要是组合API

ref和reactive

```
<template>
    <div>
        <h1>{{message}}</h1>
        <ul>
            <li v-for="(item,index) in names.list" :key="index">{{ item }}</li>
        </ul>
    </div>
</template>

<script>
import { ref,reactive } from 'vue';
export default {
    setup(){
        //用于基本数据
        const message=ref("消息在此")
        //用于复杂数据
        const names=reactive({
            list:["haohao","xth"]
        })
        //注意需要返回
    }
}
```

```
        return{
            message,
            names
        }
    }
}

</script>
```

在setup中定义函数

```
<template>
  <div>
    <h1>{{message}}</h1>
    <button @click="function1">函数</button>
  </div>
</template>

<script>
import { ref } from 'vue';
export default {
  setup(){
    //用于基本数据
    const message=ref("消息在此")
    function function1(){
      console.log("我是在setup里面定义的函数1")
      //需要注意的是这种方式需要利用value属性修改！！
      message.value="我是新的消息"
    }
    return{
      message,
      function1
    }
  }
}
</script>
```

引入Element-UI

第一步：安装依赖 `npm i element-ui -S` `npm install element-plus --save`

完整引用

这种方式的特点就是文件大小会比较大
在main.js中引入：

```
import vue from 'vue'
import App from './App.vue'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'

vue.config.productionTip = false
```

```
const app=new Vue({
  render: h => h(App)
})
app.use(ElementPlus)
app.$mount('#app')
```