

Vue

官网: <https://cn.vuejs.org>

测试vue官网的模板

data: 实例中的数据

{{}}: 插值表达式/模板的语法, 把数据和页面结构相关联

el: 管理指定的选择器

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <!--width: 可视区域的宽度, 值可为数字或关键词device-width(设备宽度)
  height: 同width
  maximum-scale=1.0, minimum-scale=1.0; 可视区域的缩放级别,
  maximum-scale 用户可将页面放大的程序, 1.0 将禁止用户放大到实际尺寸之上。
  user-scalable: 是否可对页面进行缩放, no 禁止缩放(也是放置再content中, 以逗号隔开)-->
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- 渲染网页用的引擎-->
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <title>vue_1</title>
<!-- 开发版本, 其包含了有帮助的命令行警告-->
  <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
</head>

<body>
  <div id="app">
    {{ message }}
  </div>

</body>

<script>
  var app = new Vue({
    el: '#app',
    data: {
      message: 'Hello vue!'
    }
  })
</script>

</html>
```

el: 挂载点

通过css选择器，设置el实例，管理元素

- el命中的元素内部，两个大括号修饰的部分，就会被data中同名的数据替换
- el范围：在**标签外部，不作用(无效)**；但是**指定的标签和其内部标签(后代元素)**
- el选择器：id, class, 标签；**建议使用id（具有唯一性）**
- el支持大部分标签；但html和body例外；**建议使用无样式的标签**，如：div, span

data: 数据对象

- vue中用到的数据**定义在data中**
- data中可以定义**复杂类型(对象/数组)**的数据
- 渲染复杂类型数据时，遵守**js语法**即可；**对象取其属性，数组取其下标**

本地应用

Vue指令，指的是；以v- 开头的一组特殊语法

v-text

v-text指令作用：设置标签的内容(textContent)

默认写法非替换全部内容，使用**差值表达式{{}}**可以替换指定内容

内部支持写表达式：拼接表达式

v-text="" 写在标签内部;其字符串后还可拼接表达式 如：

```
<div id = "app">
  <!--这种非把标签内部所有数据替换掉-->
  <div v-text="message + '！'"></div>
  <div>
    <!--这种是只替换大括号那么的message-->
    大学城{{ message + '！' }}
  </div>
</div>
```

```
var app = new Vue({
  el: "#app",
  data:{
    message:"hello v-text"
```

```
}  
})
```

v-html

v-html指令作用： 设置标签的innerHTML

内容中有html结构会被解析为标签

v-text指令无论内容是什么,只会解析为文本

解析文本使用**v-text**, 需要解析html结构使用**v-html**

```
<div id="app">  
  <p v-html="content"></p>  
  <p v-text="content"></p>  
</div>  
  
var app = new Vue({  
  el:"#app",  
  data:{  
    // content:"v-html"  
    content:"<a href='#>v-html</a>"  
  }  
})
```

内容绑定，事件绑定：v-on

v-on指令作用： 为元素绑定事件

事件名不需要写on(和jq类似), 如: click

指令v-on:可简写为@

绑定的方法定义在**methods**属性中

```
<div id="app">  
  <input type="button" value="v-on指令" v-on:click="doIt"><br/>  
  <input type="button" value="v-on简写" @click="doIt"><br/>  
  <input type="button" value="v-on双击" @dblclick="doIt">  
  <h2 @click="changeFood">  
    {{food}}  
  </h2>  
</div>  
  
var app = new Vue({  
  el:"#app",  
  data:{
```

```

        food:"番茄鸡蛋"
    },
    methods:{
        doIt:function () {
            alert("doIt方法");
        },
        changeFood:function () {
            alert(this.food + '还行')
        }
    }
}
})

```

计数器

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**值设置给**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

```

<div id="app">
  <div class="input-num">
    <button v-on:click="sub">-</button>
    <span>
      {{ num }}
    </span>
    <button @click="add">+</button>
  </div>
</div>

```

```

var app = new Vue({
  el:"#app",
  data:{
    num:1
  },
  methods:{
    add:function () {
      // alert("add")
      if (this.num < 10) {
        this.num++;
      }else {
        alert("我是有极限的┐ ┐")
      }
    },
    sub:function () {
      // alert("sub")
      if (this.num < 0) {
        this.num--;
      }
    }
  }
})

```

```

        }else {
            alert("我是有底线的ლ( '□' )ლ")
        }
    }
}
})

```

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

显示切换, 属性绑定: v-show

- **v-show指令作用**: 根据表达值的真假, 切换元素的显示和隐藏
- 原理: **修改元素的display**,实现显示隐藏
- 指令(v-show)后面的内容最终都会**解析为布尔值**
- 值为**true**元素**显示**,值为**false**元素**隐藏**
- **数据改变之后**,对应元素的显示状态会**同步更新**

```

<div id="app">
  <!-- 可取值: true可见/false不可见 或者自定义表达式-->
    
  <!-- 此处的方法名, 加不加(), 都可以-->
    <button v-on:click="changeIsShow">切换显示状态</button>

    <br /><br /><br /><br /><br />
    = 18">
    <button @click="addAge">累加年龄</button>
</div>

```

```

var app = new Vue({
  el:"#app",
  data:{
    isShow:true,
    age:16
  },
  methods:{
    changeIsShow:function () {
      this.isShow = !this.isShow;
    },
    addAge:function () {
      this.age++;
    }
  }
})

```

v-if

- **v-if指令作用**：根据表达值的真假,切换元素的显示和隐藏(操纵dom元素)
- 本质是通过**操纵dom**元素来切换显示状态
- 表达式的值为true,元素**存在于dom树中**, 为false,从dom树中移除
- 频繁的切换v-show,反之使用v-if前者的切换消耗小

```
<div id="app">
  <button @click="toggleIsShow">切换显示状态</button>
  <p v-if="isShow">
    v-if指令
  </p>
  <p v-show="isShow">
    v-show指令
  </p>
  <h2 v-if="temperature >= 37">热死了</h2>

</div>
```

```
var app = new Vue({
  el:"#app",
  data:{
    isshow:false,
    temperature:40
  },
  methods:{
    toggleIsShow:function () {
      this.isShow = !this.isShow;
    }
  }
})
```

v-bind

- **v-bind指令**的作用：设置元素的属性(比如:src,title,class)
- 完整写法是**v-bind:属性名**
- 简写的话可以直接**省略v-bind**,只保留**:属性名**
- 需要动态的增删class建议使用对象的方式

```
<div id="app">
<!--      使用三目运算符,如isActive方法成立,那么就添加active内,反之class为空串-->
  
  <br />
```

```

<!-- 使用对象，如果后面的方法生效，那么冒号前的值就生效；反之则不生效-->


</div>

var app = new Vue({
  el: "#app",
  data: {
    // 本地图片
    imgSrc: "../img/A.I.Channel.png",
    imgTitle: "爱酱",
    // 网络图片
    //
    imgSrc: "https://imgsa.baidu.com/forum/w%3D580/sign=099a653c6b09c93d07f20effaf3cf8bb/b40952dca3cc7cd94d9d1beb3501213fb90e91f3.jpg"

    imgSrc2: "https://imgsa.baidu.com/forum/w%3D580/sign=b2b5a3fa33f33a879e6d0012f65d1018/ced861773912b31bd7ddede38f18367adbb4e119.jpg",
    imgTitle2: "为美好的世界献上祝福",

    isActive: false
  },
  methods: {
    toggleActive: function () {
      this.isActive = !this.isActive;
    }
  }
})

```

图片切换

1. 定义图片数组
2. 添加图片索引
3. 绑定src属性
4. 图片切换逻辑
5. 显示状态切换

- 列表数据使用数组保存
- v-bind指令可以设置元素属性,比如src
- v-show和v-if都可以切换元素的显示状态,频繁切换用v-show

列表循环，表单元素绑定：v-for

- **v-for指令**的作用：根据数据生成列表结构
- **数组**经常和**v-for**结合使用
- 语法是(**item(每一个数据值)**,**index(下标)**) in 数据
- item和index可以结合其他指令一起使用
- 数组长度的更新会同步到页面上，是响应式的

```
<div id="app">
  <ul>
    <!--          arr: 需要循环的数据; in: 必须关键字; item: 遍历的每一项数据(可理解为别名);title为li标签的名称-->
    <!--<li v-for="item in arr" :title="item">
      {{ item }}
    </li>-->

    <!--          带下标的循环，若是需要加一，就直接在index后面+1({{ index + 1 }}),
    其中index也是下标的别名(可更改)-->
    <li v-for="(item, index) in arr" :title="item">
      {{ index }} , {{ item }}
    </li>
  </ul>

  <p v-for="item in vegetables">
    {{ item.name }}
  </p>

  <button value="添加数据" v-on:click="add">添加数据</button>
  <button value="移除数据" v-on:click="remove">移除数据</button>
</div>
```

```
var app = new Vue({
  el: "#app",
  data: {
    // 列表的生成依赖于数据，那么就需要先定义数据，数据包括：对象，数组(常用)，
    字符串，迭代器
    arr: ["美国", "中国", "日本", "德国", "英国"],

    //对象数组,name:对象的属性名
    vegetables: [
      {name: "北京"},
      {name: "上海"},
      {name: "广州"},
      {name: "深圳"},
      {name: "成都"},
      {name: "杭州"},
    ]
  }
})
```



```

    ]
  },
  methods:{
    add:function () {
      //添加方法, push: 在数组最后添加内容
      this.vegetbles.push({name:"重庆"});
    },
    remove:function () {
      //移除的方法, shift: 移除数组的第一个元素, 并返回被删元素的值
      this.vegetbles.shift()
    }
  }
})

```

v-on补充

v-on指令的作用: 传递自定义参数事件修饰符

常见修饰符, 查看官网: <https://cn.vuejs.org/v2/api/#v-on>

```

停止冒泡: @click.stop=""
阻止默认行为: @click.prevent=""
阻止默认行为, 没有表达式: @submit.prevent
串联修饰符: @click.stop.prevent=""
键修饰符, 键别名: @keyup.enter=""
键修饰符, 键代码: @keyup.13=""
点击回调只会触发一次: v-on:click.once=""

```

- 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
- 定义方法时需要**定义形参**来接收传入的实参
- 事件的后面跟上**.修饰符**可以对事件进行限制
- **.enter**可以限制触发的按键为回车
- 默认的@keyup, 不加任何修饰, 那么是键盘弹起/或值发生改变

```

<div id="app">
  <!-- 括号中为传递的值-->
  <button value="点击" @click="doIt('传递的值', '第二个值')">点击</button>
  <input type="text" placeholder="输入框" @keyup.f1="sayHi" />

</div>

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

```

```

    },
    methods:{
        // 括号中的value, 为方法的形参, 用来装调用方法时, 传递的值;可以传递多个参数, 再往后添加形参即可
        doIt:function (value, p2) {
            console.log("v-on补充" + "----->" + value);
            console.log(p2);
        },
        sayHi:function () {
            alert("Hi")
        }
    }
}

})

```

键盘别名:

```

.enter: 回车
.tab: tab键
.delete (捕获“删除”和“退格”键)
.esc: esc键
.space: 空格键
.up: 上键
.down: 下键
.left: 左键
.right: 右键

```

一些组合按键:

```

.ctrl
.alt
.shift
.meta(window系统是window键, mac下是command键)

```

Alt + C :

```
<input @keyup.alt.67="doSth">
```

v-model

v-model指令的作用: 便捷获取和设置表单元素的值(双向数据绑定)

无论更改标签内的值, 还是js里面控制的值, 只要发生改变, 那么两边同时改变

- 绑定的数据会和**表单元素值**相关联
- 绑定的数据 \longleftrightarrow 表单元素的值 (双向数据绑定)

```

<div id="app">
    <input type="text" placeholder="文本框" v-model="message" v-
on:keyup.enter="getMessage" />

```

```

<!-- 可以发现，标题内的文本，和文本框实时同步了-->
<h2>{{ message }}</h2>

<!-- 点击后，设置文本框的值-->
<input type="button" value="设置文本框的值" @click="setMessage" />
</div>

var app = new Vue({
  el: "#app",
  data: {
    message: "v-model指令"
  },
  methods: {
    getMessage: function () {
      // 输入文本后，按下回车，即弹出警告框
      alert(this.message);
    },
    setMessage: function () {
      this.message = "这是设置后的值";
    }
  }
})

```

小黑记事本

1. 新增

1. 生成列表结构(v-for 数组)
2. 获取用户输入(v-model)
3. 回车,新增数据(v-on.enter 添加数据)

掌握

1. v-for指令的作用
2. v-model指令的作用
3. v-on指令,事件修饰符

2. 删除

1. 点击删除指定内容(v-on splice索引)

掌握:

1. 数据改变,和数据绑定的元素同步改变
2. 事件的自定义参数
3. splice方法的作用

`splice`方法对数组进行操作时,会改变原有数组的结构,并生成一个新的数组

...

```
var list = [1, 2, 3, 4, 5];
```

```
list.splice(0, 1); // 删除 -> 从下标为0开始,长度为1
```

```
list.splice(1, 0, 5); // 表示在下标为1处添加一项5
```

```
list.splice(0, 1, 4); // 替换 -> 从下标为0开始,长度为1的数组元素替换成4
```

...

...

`arrayObject.splice(index,howmany,item1,.....,itemX)` 方法向/从数组中添加/删除项目,然后返回被删除的项目

index 必需。整数,规定添加/删除项目的位置,使用负数可从数组结尾处规定位置。

howmany 必需。要删除的项目数量。如果设置为 0,则不会删除项目。

item1, ..., itemX 可选。向数组添加的新项目

slice

定义: `slice()` 方法可从已有的数组中返回选定的元素。

使用方法: `arr.slice(start,end);` //`start`为初始位置,`end`为结尾位置,返回的结果是从`start`到`end`(不取)的新数组

```
arr.slice(start); //选取从start开始直至最后一个元素
```

...

3. 统计

1. 统计信息个数(`v-text length`)

掌握

1. **基于数据**的开发方式
2. `v-text`指令的作用

4. 清空

1. 点击清除所有信息(`v-on 清空数组 = []`)

掌握

1. **基于数据**的开发方式

5. 隐藏

1. 没有数据时,隐藏元素(`v-show` 或 `v-if` 数组非空)

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

- 基于数据的开发方式

网络应用

Vue结合网络数据开发应用

axios:网络请求库(需处于联网状态)

axions下载地址: <https://unpkg.com/axios@0.19.2/dist/axios.min.js>

长用axios+vue配合

请求完成触发第一个方法, 形参: 服务器响应内容; 请求失败时触发, 形参: 错误信息

传递参数: 地址后面以? 隔开; key为接口文档提供(某些文档中参数(key)名字, 可以随意取), value为具体数据

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

post请求: 和get方式类似, 只是传递参数处, 有变动

axios.get(地址,{key=value&key2=value2}).then(function(response){},function(err){})

- axios必须先导入才可以使用
- 使用get或post方法即可发送对应的请求
- then方法中的回调函数会在请求成功或失败时触发
- 通过回调函数的形参可以获取响应内容或错误信息

axions文档地址: <https://github.com/axios/axios>

通过get请求, 获取笑话的接口: <https://autumnfish.cn/api/joke/list>

image-20200524224615472

通过post请求, 注册用户: <https://autumnfish.cn/api/user/reg>

image-20200524224626303

```
<button value="get请求" class="get">get请求</button><br />
<button value="post请求" class="post">post请求</button>
```

/*

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 () {
    //表示, 获取三条笑话(参数名字, 可以随意取)
    axios.get("https://autumnfish.cn/api/joke/list?num=3")
    .then(function (response) {
        //输出获取的笑话
        console.log(response);
    },function (err) {
        //获取错误信息
        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 () {
//表示, 获取三条笑话
    axios.post("https://autumnfish.cn/api/user/reg", {username:"张三"})
    .then(function (response) {
        //输出响应内容
        console.log(response);
    },function (err) {
        //获取错误信息
        console.log(err);
    })
}

```

axions+vue

- axios回调函数中的this已经改变, 无法访问到data中数据
- 把this保存起来(存储到新的对象中),回调函数中直接使用保存的this即可
- 和本地应用的最大区别就是改变了数据来源

```

<div id="app">
    <button value="获取笑话" v-on:click="getJoke">获取笑话 </button>
    <p>{{ joke }}</p>
</div>

/*
接口:随机获取一条笑话

```

请求地址(直接访问, 得到一条随机笑话, 放在页面上): <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: "在网络中, 随机获取的一条笑话"
  },
  methods: {
    getJoke: function () {
      //console.log("this.joke:\t" + this.joke); //data里面的数据
      var that = this; //this对象, 赋值给一个新对象
      axios.get("https://autumnfish.cn/api/joke")
        .then(function (response) {
          //方法的数据
          // console.log(response);
          //console.log("this.joke:\t" + this.joke); //未定义

          //只获取笑话
          console.log(response.data);
          //网络返回的数据, 赋值给变量
          that.joke = response.data;
        }, function (err) {
          //错误信息
          console.log(err);
        })
    }
  }
})

```

天知道

1. 回车查询

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

掌握

1. 应用的逻辑代码建议和页面**分离**, 使用**单独的js文件** 编写
2. **axios回调函数**中**this**指向改变了, 需要额外的保存一份
3. 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

2. 点击查询

1. 点击城市(**v-on自定义参数**)
2. 查询数据(**this.方法()**)
3. 渲染数据

掌握

1. 自定义参数可以让代码的**复用性**更高


2. `methods`中定义的方法内部，可以通过`this`关键字点出其他的方法

综合应用

歌曲搜索接口: <https://autumnfish.cn/search>

 image-20200525152418053


歌曲url: <https://autumnfish.cn/song/url>

 image-20200525162311767

获取歌曲详细信息: <https://autumnfish.cn/song/detail>

 image-20200525171117834

获取歌曲评论: <https://autumnfish.cn/comment/hot?type=0>

 image-20200525173217689

mv的地址: <https://autumnfish.cn/mv/url>

 image-20200525183118335

1. 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`)

掌握

1. 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
2. 通过审查元素快速定位到需要操纵的元素

2. 歌曲播放

1. 点击播放(`v-on`自定义参数)
2. 歌曲地址获取(接口歌曲id)
3. 歌曲地址设置(`v-bind`)

掌握

1. 歌曲id依赖歌曲搜索的结果，对于不用的数据也需要关注

3. 歌曲封面

1. 点击播放(增加逻辑)
2. 歌曲封面获取(接口歌曲id)
3. 歌曲封面设置(v-bind)

掌握

1. 在vue中通过v-bind操纵属性
2. 本地无法获取的数据，基本都会有对应的接口

4. 歌曲评论

1. 点击播放(增加逻辑)
2. 歌曲评论获取(接口歌曲id)
3. 歌曲评论渲染(v-for)

掌握

1. 在vue中通过v-for生成列表

5. 播放动画

1. 监听音乐播放(v-on play)
2. 监听音乐暂停(v-on pause)
3. 操纵类名(v-bind对象)

掌握

1. audio标签的play事件会在音频播放的时候触发
2. audio标签的pause事件会在音频暂停的时候触发
3. 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

6. mv播放

1. mv图标显示(v-if)
2. mv地址获取(接口 mvid)
3. 遮罩层(v-show v-on)
4. mv地址设置(v-bind)

掌握

1. 不同的接口需要的数据是不同的，文档的阅读需要仔细
2. 页面结构复杂之后，通过审查元素的方式去快速定位相关元素
3. 响应式的数据都需要定义在data中定义

历史记录：

- 1、定义数组，存放输入过的记录
- 2、当数组大于或等于某个值时，移除数组最后一个

3、将数组放置html, 某标签内