Vue基础

一、MVVM开发概念及SPA概念

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式JavaScript框架。 与其它大型框架不同的是，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。另一方面，当与现代化的工具链以及各种支持类库结合使用时，**Vue 也完全能够为复杂的单页应用（SPA）提供驱动。**

在学习vue的框架之前先了解一个概念叫MVVM（model view and viewmodel），其核心理念就是数据驱动页面 。与之相对应的有一套技术叫 `MVC`

> 在以前我们很多后端的开发框架里面都会提到MVC的概念（Model View Controller），这个MVC的概念是前后端结合的一种开发方式【目前基本上已经被替代】。现在的主流的开发方式是前后端分离式开发（前后端在开发的时候在2个不同的项目里面，真正开发完了以后还可以再次合到一起）
> 如java里面会有 `servlet` 或 `spring MVC`，如 `php` 语言里面的 `thinkPHP`，如 `.net` 平台正同会有 `.net MVC`
> 现在这些技术在后端里面也慢慢的被淘汰了，正是因为后端改变了开发方式，如java里面的 `spring MVC` 现在换成了 `spring boot`，那么前端也要相应的做出一些改变，前端就从后端的代码里面分开了，成了一个单独的项目

在目前的前端里面，我们学习过很多与页面相关的知识，如布局我们使用CSS,如操作页面我们使用JS，而JS在操作页面的时候使用 `DOM` 技术，dom技术本质也是一种开发思路。但是今天我们就要完全放弃DOM操作了
MVVM不依赖于dom操作，它完全造数据，用数据去驱动页面
目前流行的MVVM框架或类似于MVVM核心理念的框架有很多，主流的在以下几个

1. `vue`【它是一个入门型的开发框架，很容易上手，但是提高很难】【尤雨溪，中国人】
2. `react`【上手难，提高更难】【facebook】
3. `angular`【难上加难，它需要借用于typescript去完成】【google开发的】
4. `svelte`【上手简单，但是很考验基本功，特别是原生JS和底层JS，还有ES6】

那什么是SPA的概念呢
SPA的全称叫单页面应用程序，指的是整个程序在经过处理以后只有一个页面
单页Web应用（single page web application，SPA），就是只有一张Web页面的应用，是加载单个HTML页面并在用户与应用程序交互时动态更新该页面的Web应用程序
这里就再次说明，`vue/react/angular` 都可以实现单页面的开发

二、Vue的安装与引入

目前vue的版本主要包含在 `vue2` 和 `vue3`,同时 `vue3` 里面包含了 `vue2` 基本上所有的功能核心，所以在这里，我们使用 `vue3` 的版本，同讲解目前的2种语法格式

1. 第一种就是基于 `vue2` 的语法格式叫 `options api`【选项式API】
2. 第二种就是基于 `vue3` 的语法格式叫 `composition api`【组合式的API】
   如果需要安装，我们可以通过 `npm` 去下载，也可以直接从官网去下载
   vue2的官网： https://v2.cn.vuejs.org/

vue3的官网： https://cn.vuejs.org/
vue的作用是尤雨溪，它是一个中国人，以一人之力对抗google与facebook

```
$ npm info vue
```

在vue里面，我们直接使用vue的话，我们要使用 `vue.global.js` 这个文件

```
<script src="./js/vue.global.js"></script>
```

三、Vue接管页面数据

vue是一个**数据驱动页面的框架**，数据驱动页面就是使用数据管理页面， 简而言之就是数据变化，页面变化；同时页面变化，数据也变化
数据与页面实现了双向的 绑定
在旧版本里面，我们使用的命令如下

```
new Vue();
```

这一种方式在新版本里面会报错，**新版本里面使用如下**

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

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>vue的第一课</title>
</head>

<body>
    <!-- Vue要去管理这个区域 -->
    <div id="app">
        <h2>大家好，这是一个标题</h2>
        <input type="text" v-model="userName">
        <h1>{{userName}}</h1>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        //vue是接管页面的数据的，那页面的数据应该来源于什么地方呢？
        data() {
            //返回的这个对象就是页面要接管的区域的数据
            return {
                userName:"标哥哥"
            }
        }
    }).mount("#app"); //创建的vue的对象要接管id为box的区域
</script>

</html>
```

在上面的案例里面，我们可以发现，如果要继续操作页面，已经可以不再依赖于DOM这种技术

四、Vue的数据绑定指令

1.插值语法

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <h2>{{age}}</h2>
        <h2>{{age>=18?"成年人":"未成年人"}}</h2>
        <h2>{{num1.toString(2)}}</h2>
        <h2>{{new Date().toLocaleDateString()}}</h2>
        <h2>{{1+1}}</h2>
        <div>
            {{str1}}
        </div>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三",
                age: 20,
                num1: 18,
                str1:"<input type='text' placeholder='请输入账号' />"
            }
        }
    }).mount("#app");
</script>
```

插值语法通过 {{}} 形式在页面上面显示数据，这个数据的来源是vue的 data 里在的东西，同时插件语法里面也可以写ES的普通代码 ，如条件运算符，如方法的调用
这一种插值语法就相当于设置了某一个元素的 innerText 的属性

2.v-text

这一种方式与上面的插值语法的试试是一模一样的，如下所示

```
<body>
    <div id="app">
       <h2>{{userName}}</h2>
       <h2 v-text="age"></h2>
       <h2 v-text="num1"></h2>
       <div v-text="str1"></div>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三",
                age: 20,
                num1: 18,
                str1:"<input type='text' placeholder='请输入账号' />"
```

```
            }
        }
    }).mount("#app");
</script>
```

这个 `v-text` 其实也相当于设置 `innerText` 的属性

3.v-html

```

<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <h2 v-text="userName"></h2>
        <h2 v-html="userName"></h2>
        <hr>
        <div>{{str1}}</div>
        <div v-text="str1"></div>
        <div v-html="str1"></div>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三",
                age: 20,
                num1: 18,
                str1:"<input type='text' placeholder='请输入账号' />"
            }
        }
    }).mount("#app");
</script>
```

> 这个与上面是不一样的，它相当于设置了元素的 `innerHTML` 的属性

张三

张三

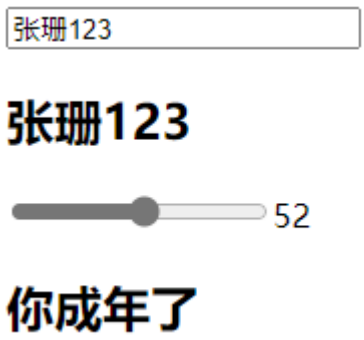
张三

<input type='text' placeholder='请输入账号' />
<input type='text' placeholder='请输入账号' />
请输入账号

4.v-model

在页面上面显示数据的时候， 我们还有一种特殊的元素就是表单如果， 如果要绑定表单元素的值，则要使用`v-model`

```
<body>
    <div id="app">
        <!-- 表单元素的绑定是一个双向绑定的过程 -->
        <input type="text" v-model="userName">
        <h2>{{userName}}</h2>
        <input type="range" v-model="age">{{age}}
        <h2>{{age>=18?"你成年了":"你未成年"}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张珊123",
                age: 18
            }
        }
    }).mount("#app");
</script>
```



下面还有单选框的表单绑定

```
<body>
    <div id="app">
        <input type="radio" value="男" v-model="a" name="sex">男
        <input type="radio" value="女" v-model="a" name="sex">女
        <hr>
        <h2>你选择的性别是：{{a}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                a:"女"
            }
        }
    }).mount("#app");
</script>
```

再来一个多选框的表单绑定

```
<body>
    <div id="app">
        <input type="checkbox" name="hobby" v-model="hobbyList" value="看书">看书
        <input type="checkbox" name="hobby" v-model="hobbyList" value="睡觉">睡觉
        <input type="checkbox" name="hobby" v-model="hobbyList" value="玩游戏">玩游戏
        <input type="checkbox" name="hobby" v-model="hobbyList" value="吃饭">吃饭
        <hr>
        <h2>{{hobbyList}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                hobbyList: ["看书","吃饭"]
            }
        }
    }).mount("#app");
</script>
```

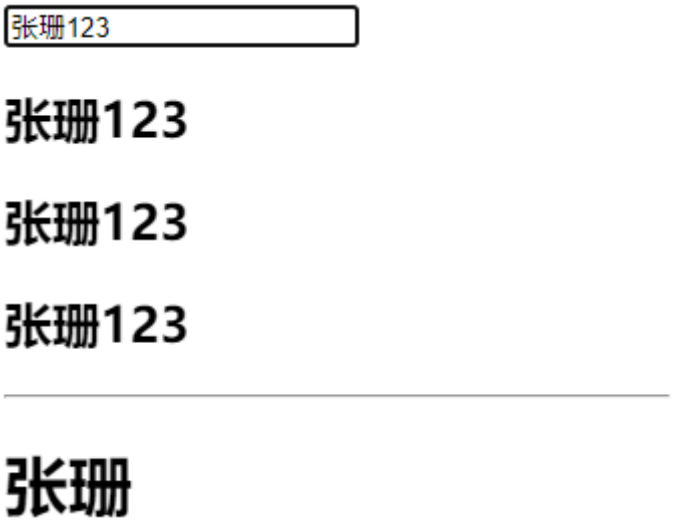
> 多选框的绑定需要使用数组的方式去完成

5.v-once

`v-once` 是单次绑定，它只会在第一次的时候做数据渲染，渲染完了以后就与vue断开了关联，后期如果数据发生了变化，这一个地方也不再变化

```
<body>
    <!-- 创建接管区域 -->
    <div id="app">
        <input type="text" v-model="userName">
        <h2>{{userName}}</h2>
        <h2 v-text="userName"></h2>
        <h2 v-html="userName"></h2>
        <hr>
        <h1 v-once>{{userName}}</h1>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    // const app = Vue.createApp()
    // app.mount("#app");
    Vue.createApp({
        data() {
            return {
                userName: "张珊"
            }
        }
    }).mount("#app");
</script>
```

如果后期我们要是把输入框的值改变，那么下面的值也会改变，但是`v-once`的地方就不会再变了



6.v-show

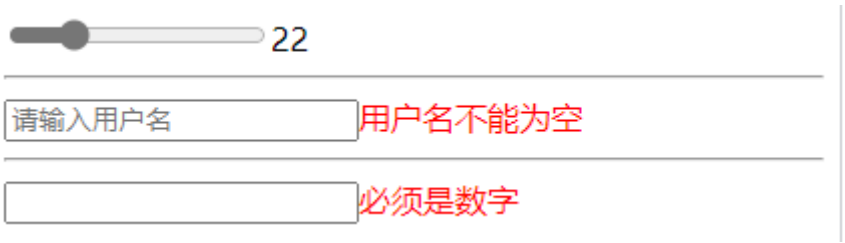
```
<body>
    <div id="app">
        <h2 v-show="true">今天是11月的第1天</h2>
        <h2 v-show="false">天气也还不错</h2>
        <h2 v-show="1>2">气温好像也不低,比较暖和</h2>
        <h2 v-show="123">大家都在教室里面认真的听课</h2>
        <h2 v-show="0">马淑圆很聪明</h2>
        <!-- 大家要知道 6 个false的值是什么 -->
        <h2 v-show="'0'">马淑圆很聪明,回答得非常好</h2>

    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({

    }).mount("#app")
</script>
```

通过上面的例子，我们可以看到，这里面显示或不显示已经变得非常简单的，只要`v-show`后面的值为真，就会显示这个元素。它在隐藏这个元素的时候， 是通过`display:none`来完成的

```
<div id="app" data-v-app>
  <h2>今天是11月的第1天</h2>
  <h2 style="display: none;">天气也还不错</h2> == $0
  <h2 style="display: none;">气温好像也不低,比较暖和</h2>
  <h2>大家都在教室里面认真的听课</h2>
  <h2 style="display: none;">马淑圆很聪明</h2>
  <!-- 大家要知道 6 个false的值是什么 -->
  <h2>马淑圆很聪明,回答得非常好</h2>
</div>
<script src="./js/vue.global.js"></script>
```

在工作当中， v-show 使用得非常频繁，如下例子所示

```
<body>
    <div id="app">
        <input type="range" v-model="age">{{age}}
        <h2 v-show="age<18">你的年龄没有达到18岁,所以你不能进网吧</h2>
        <hr>
        <input type="text" v-model="userName" placeholder="请输入用户名">
        <span v-show="userName.length==0" style="color: red;">用户名不能为空</span>
        <hr>
        <input type="text" v-model="money">
        <span v-show="!/^\d+(\.\d+)?$/.test(money)" style="color: red;">必须是数字
</span>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                age: 22,
                userName: "",
                money: 0
            }
        }
    }).mount("#app")
</script>
```



7.v-if，v-else，v-else-if

> 条件渲染，根据条件来决定是否渲染

这个东西与我们的v-show看起来非常像，但是本质上面是完全不一样的，如下所示

```
<body>
    <div id="app">
```

```
        <h2 v-if="age>=18">你是成年人了,你可以去领结婚证了</h2>
        <h2 v-else>你还是个小屁孩子 ,快去玩泥巴去吧</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                age: 10
            }
        }
    }).mount("#app")
</script>
```

`v-if` 只是根据一个条件渲染了一个元素上去，而如果使用 `v-show` 的话，它是有多个元素的，不符合条件的元素只是通过 `display:none` 隐藏掉了

同时要注意 `v-if` 还可以有多个条件去同时进行

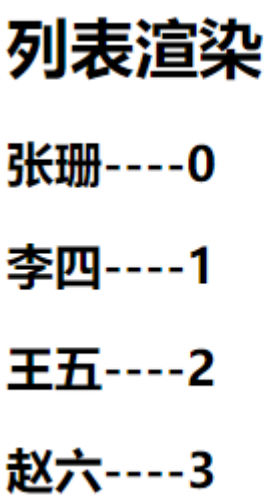
```
<body>
    <div id="app">
        <h2 v-if="score>=90">优秀</h2>
        <h2 v-else-if="score>=80">良好</h2>
        <h2 v-else-if="score>=70">中等</h2>
        <h2 v-else-if="score>=60">及格</h2>
        <h2 v-else>不及格</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                score: 85
            }
        }
    }).mount("#app")
</script>
```

> **在明天的时候，我们会讲到生命周期，讲到Vue的生命周期的时候，我们要重点再来看 `v-show` 与 `v-if` 的区别**

8.v-for

> 列表渲染，学完这个东西以后，同学们就把 `art-template` 丢了,不用它了

```
<body>
    <div id="app">
        <h1>列表渲染</h1>
        <!-- item代表的就是遍历的这一项,这个名子可以更改 -->
        <h2 v-for="(item,index) in stuList">{{item}}----{{index}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                stuList: ["张珊", "李四", "王五", "赵六"]
            }
        }
    }).mount("#app");
</script>
```



> 语法格式： `v-for="(当前遍历的项，当前遍历的索引) in 集合"`
>
> 这种语法格式 我更想让同学们理解为 `v-for=(value ,key) in 对象`

```
<body>
    <div id="app">
        <h1>列表渲染</h1>
        <!-- item代表的就是遍历的这一项,这个名子可以更改 -->
        <h2 v-for="(item,index) in stuList">{{item}}----{{index}}</h2>
        <hr>
        <h2 v-for="(item,index) in Array.from(teachers)">{{item}}---{{index}}
</h2>
        <hr>
        <h2 v-for="(item,index) in obj">{{item}}---{{index}}</h2>
        <hr>
        <h2 v-for="(item,index)  in obj2">{{item}}---{{index}}</h2>
    </div>
```

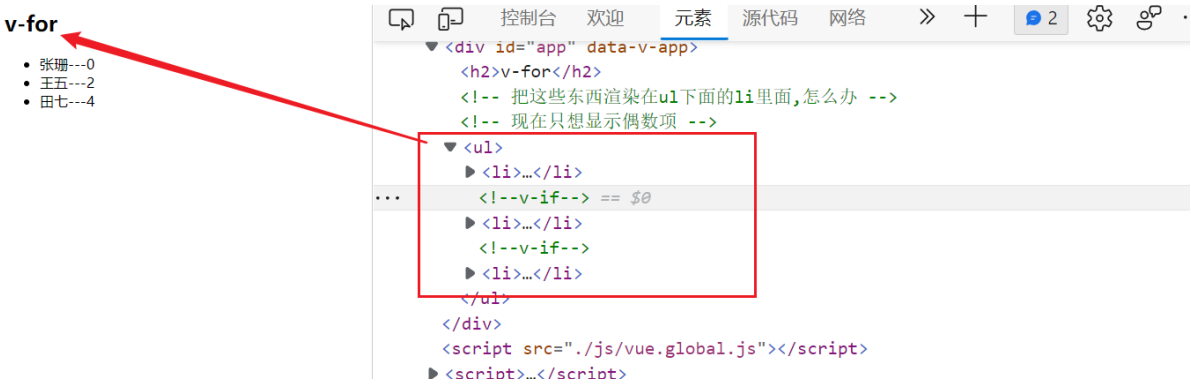
```
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                stuList: ["张珊", "李四", "王五", "赵六"],
                teachers: new Set(["标哥", "桃哥", "柴哥"]),
                obj: {
                    0: "陈润福",
                    1: "肖中",
                    2: "隗成勇",
                },
                obj2: {
                    userName: "小白",
                    sex: "女"
                }
            }
        }
    }).mount("#app");
</script>
```

当 `v-for` 与 `v-if` 同时进行的时候，这个时候在V3的版本下面是有一点区别改动的，如下所示

```
<body>
    <div id="app">
        <h2>v-for</h2>
        <!-- 把这些东西渲染在ul下面的li里面,怎么办 -->
        <!-- 现在只想显示偶数项 -->
        <ul>
            <template v-for="(item,index) in stuList">
                <li v-if="index%2==0">{{item}}---{{index}}</li>
            </template>
        </ul>
    </div>
</body>
```

```
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                stuList: ["张珊", "李四", "王五", "赵六", "田七"]
            }
        }
    }).mount("#app");
</script>
```

> 这里要注意，`v-for` 与 `v-if` 可以写在一个元素上面， 但是访问不到 `index` 的这个值，需要通过 `<template>` 再去包裹一层



9.v-if与v-for的扩展及注意事项

第一个注意事项

当我们同时去使用 `v-if` 与 `v-for` 的时候，看下面的代码

```
<body>
    <div id="app">
        <ul>
            <li v-for="(item,index) in stus">{{item}}----{{index}}</li>
        </ul>
        <hr>
        <ul>
            <li v-for="(item,index) in stus" v-if="index%2==0">{{item}}----
{{index}}</li>
        </ul>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                stus: ["张三", "李四", "王五", "赵六", "田七"]
```

```
            }
        }
    }).mount("#app")
</script>
```

当我们在执行第2个遍历的时候，就会报警告信息

```
You are running a development build of Vue.                    vue.global.js:10938
Make sure to use the production build (*.prod.js) when deploying for production.
▶ [Vue warn]: Property "index" was accessed during render vue.global.js:1622
but is not defined on instance.
  at <App>
[Five Server] connecting...                                    fiveserver.js:1
[Five Server] connected.                                       fiveserver.js:1
```

原理是因为 `v-if` 的优先级比 `v-for` 要高一些，当2这个指令在同一个元素上面的时候，它会先执行 `v-if` ，再去执行 `v-for`

在上面的代码里面，当我们先去执行 `v-if` 的时候，里面巾的条件 是 `index%2==0` ，这个时候 因为还没有执行 `v-for` ，所以 `index` 还不存在，这个时候就会报警告信息

> ⚠ 注意
>
> 同时使用 `v-if` 和 `v-for` 是**不推荐的**，因为这样二者的优先级不明显。请转阅风格指南查看更多细节。

当它们同时存在于一个节点上时， `v-if` 比 `v-for` 的优先级更高。这意味着 `v-if` 的条件将无法访问到 `v-for` 作用域内定义的变量别名：

在外新包装一层 `<template>` 再在其上使用 `v-for` 可以解决这个问题 (这也更加明显易读)：

```
<template v-for="todo in todos">
  <li v-if="!todo.isComplete">
    {{ todo.name }}
  </li>
</template>
```

第二个注意事项

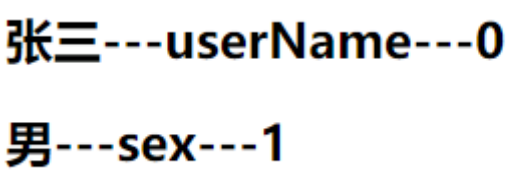
在 `V3` 里面执行 `v-for` 的遍历时候，不一定非要是数组，还可以是对象 ，对象在遍历的时候，它的语法如下

```
<元素 v-for="(value,key,index) in 对象"></元素>
```

所以看下面的案例

```
<body>
    <div id="app">
        <h2 v-for="(value,key,index) in obj">{{value}}---{{key}}---{{index}}</h2>
    </div>
</body>
```

```
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                obj: {
                    userName: "张三",
                    sex: "男"
                }
            }
        }
    }).mount("#app")
</script>
```



第三个注意事项

`v-for`还可以执行普通的`for`循环了，如下所示

```
<!-- item从1开始，index从0开始 -->
<h2 v-for="(item,index) in 10">{{item}}---{{index}}</h2>
```

第四个注意事项

`v-for`在执行遍历的时候可以直接解构了

```
<body>
    <div id="app">
        <ul>
            <li v-for="(item,index) in stuList">{{item.stuName}}</li>
        </ul>
        <hr>
        <ul>
            <li v-for="({stuName,sex},index) in stuList">{{stuName}}---{{sex}}
</li>
        </ul>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                stuList: [{
                    stuName: "张珊",
                    sex: "女"
                }, {
                    stuName: "李四",
                    sex: "男"
                }, {
                    stuName: "王五",
```

```
                    sex: "男"
                }]
            }
        }
    }).mount("#app")
</script>
```

- 张珊---女
- 李四---男
- 王五---男

10.v-once与v-show的惰性

```
<body>
    <div id="app">
        <input type="text" v-model="userName"> <button type="button" @click="flag=true">按钮</button>
        <h2>{{userName}}---{{flag}}</h2>
        <h2 v-once v-show="flag">{{userName}}</h2>
    </div>
    <!-- 当我们把输入的框的值改变 张珊123以后，再去点击按钮的时候 ，这个时候下面的h2的结果应该是A -->
    <!--
        把张珊改成张珊123以后，点击按钮，flag变为true下的h2标签 会出现什么结果
        A：不会做任何变化
        B：会显示，显示内容是第一次的 张珊
        C: 会显示，显示张珊123
        D：程序报错
    -->
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张珊",
                flag: false
            }
        }
    }).mount("#app")
</script>
```

11.v-bind

之前我们所学习的所有的指令都是让页面上面显示内容，但是如果想对一个HTML的属性进行绑定动态的值， 要使用 v-bind:属性 这一种方式来完成

```
<body>
    <div id="app">
        <!-- {{}}相当于innerText -->
        <!-- 如果要绑定一个属性 -->
        <a v-bind:href="y">{{x}}</a>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data(){
```

```
            return {
                x:"百度一下",
                y:"https://www.baidu.com"
            }
        }
    }).mount("#app")
</script>
```

```
<div id="app" data-v-app>
    <!-- {{}}相当于innerText -->
    <!-- 如果要绑定一个属性 -->
    <a href="https://www.baidu.com">百度一下</a>
</div>
<script src="./js/vue.global.js"></script>
```

因为属性绑这种操作在后期会经常使用，所以 vue 里面提供了一种简写的办法

```
<div id="app">
    <!-- {{}}相当于innerText -->
    <!-- 如果要绑定一个属性 -->
    <a v-bind:href="y">{{x}}</a>
    <hr>
    <a :href="y">{{x}}</a>
</div>
</body>
<script src="./js/vue.global.js"></script>
```

简写的属性绑定

Vue的事件及事件对象

vue是数据驱动页面的框架， 所以它不仅仅可以接管页面的数据， 还可以接管页面的事件

1. vue绑定事件的方式

```
<button type="button" v-on:click="">按钮1</button>
```

> 在上面的代码里面，我们可以看到 v-on:事件名 就是vue里面的事件绑定

因为事件的绑定也是一个非常常见的操作，所以在这里它也有简化的方法

```
<button type="button" @click="">按钮1</button>
```

> 它的简化方法就是 @事件名 即可

2. 以方法的形式接管事件

```
<body>
    <div id="app">
        <!-- 在接管区域里面，如果要做事件绑定，非常简单 -->
        <!-- 这里的事件方法是没有加括号，因为它不需要传递参数 -->
        <button type="button" v-on:click="sayHello">按钮1</button>
        <!-- 这里的事件方法加了括号，加了括号以后，我们就可以传递参数 -->
        <button type="button" v-on:click="abc(18)">按钮2</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        // 负责接管页面的数据
        data() {
            return {

            }
        },
        // 接管页面的事件方法
        methods: {
            sayHello() {
                alert("你好啊，我在弹窗");
            },
            abc(age){
                alert(`我今年${age}岁了`)
            }
        }
    }).mount("#app");
</script>
```

> **代码分析**
>
> 1. `data` 是负责接管页面的数据的，`methods` 就是负责接管页面上面的方法
> 2. `v-on:事件` 可以赋值一个函数名，这个函数名可以有括号，也可以没有括号【如果你需要传递参数，你就添加括号，如果你不需要传递参数，则这个括号就可以省略】

3. 以行内的方式接管事件

> 这一种写法非常简单，它直接把代码写在了事件的属性名里面

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <button type="button" @click="aaa">按钮1</button>
        <button type="button" @click="userName = '王五'">按钮2</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
```

```
                userName: "张三"
            }
        },
        methods: {
            aaa() {
                this.userName = "李四";
            }
        }

    }).mount("#app");
</script>
```

> **注意事项**
>
> 1. 如果行内的事件代码要操作vue自身的data里面的数据，不要加 `this` 【在vue的托管区域内部是严禁出现this关键字】
> 2. 如果是行内的事件代码，它内部没有 `window` ，所以也就没有 `window` 下面的那些对象

4. vue的方法访问vue的数据

> 掌握使用方法，了解一下原理就可以了

```
<body>
    <div id="app">
        <h2>{{girlName}}</h2>
        <button type="button" v-on:click="test1">按钮1</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        // 负责接管页面的数据
        data() {
            return {
                girlName: "小红"
            }
        },
        // 接管页面的事件方法
        methods: {
            test1() {
                //如果在事件方法里面,想拿到自己的data里面的数据 ，可以直接通过自己的对象来完成
                // 完整的写法
                console.log(this.$data.girlName);
                console.log(this.girlName);  //它是一个代理
                //当面试官问你，为什么this.girlName可以取到this.$data.girlName
                //因为this[vue]对象它是一个proxy代理对象 ，内部的handler代理方法代理了
                //$data的取值操作
                console.log(this);
            }
        }
    }).mount("#app");
</script>
```

> 这里可以会涉及到vue的源代码里面的代理模式这一块，先暂时不用了解
>
> 综合所述，如果要在方法里面访问data里面的数据则通过 `this.数据名` 就可以得到

vue的事件对象

在vue里面，我们可以在事件方法里面使用 `$event` 来获取事件对象，如下所示

```
<body>
    <div id="app">
        <button type="button" @click="aaa($event)">按钮</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        methods: {
            aaa(event) {
                console.log("你好");
                console.log(event);
            }
        }
    }).mount("#app")
</script>
```

Vue事件修饰符

> 修饰符：用来装饰某一个东西，所以事件修饰符应该也是用来装饰事件的

```
<body>
    <div id="app">
        <div class="box" @click="aaa">
            <button type="button" @click="bbb($event)">按钮</button>
        </div>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        methods: {
            aaa(){
                console.log("我是aaa的事件")

            },
            bbb(event){
                console.log("我是bbb的事件")
                event.stopPropagation();
            }
        }
    }).mount("#app")
</script>
```

在以前如果要阻止事件的冒泡，我们要获取事件对象以后再进行操作

但是vue的操作已经变得非常简单了

```
body>
  <div id="app">
      <div class="box" @click="aaa">
          <button type="button" @click.stop="bbb">按钮</button>
      </div>
  </div>                          如果想阻止事件的冒泡 ， 可以直接使用这个修饰符
body>
cript src="./js/vue.global.js"></script>
```

上面的的 `@click.stop` 就是使用事件修饰了， `stop` 的修饰符就是阻止事件冒泡， 与之相似的修饰符还有很多，我现在列举一下

事件修饰符

1. `.stop`
2. `.prevent`
3. `.self`
4. `.capture`
5. `.once`
6. `.passive`

按键修饰符

Vue 为一些常用的按键提供了别名：

- `.enter`
- `.tab`
- `.delete` (捕获“Delete”和“Backspace”两个按键)
- `.esc`
- `.space`
- `.up`
- `.down`
- `.left`
- `.right`

鼠标修饰符

- `.left`
- `.right`
- `.middle`

vue的小案例

```
<body>
    <div id="app">
        <!-- 在输入框里面输入内容以后,回车,添加在下面的列表里面 -->
        <input type="text" v-model.trim="txt" @keydown.enter="stuList.add(txt)">
        <hr>
        <ul>
            <!-- 当我们双击某一项的时候 ， 要把某一项删除 -->
            <li v-for="(item,index) in stuList" @dblclick="stuList.delete(item)">{{item}}---{{index}}</li>
        </ul>
    </div>
</body>
```

```
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                txt: "",
                stuList: new Set(["张三", "李四"])
            }
        }

    }).mount("#app");
</script>
```

this.$set与this.$delete使用

> 这个问题在vue3的版本里面已经做了处理，不需要再使用`this.$set()`和`this.$delete()`去完成了，因为`vue3`的内部使用的是代理全局拦截，即使是新增的属性也可以实现响应

以下的代码是vue2的代码

```
<body>
    <div id="app">
        <h2>{{userInfo.userName}}</h2>
        <h2>{{userInfo.sex}}</h2>
        <button type="button" @click="aaa">按钮1</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            userInfo: {
                userName: "张三"
            }
        },
        methods: {
            aaa() {
                // 第一种方式
                // this.userInfo.sex = "男";
                // 强制更新,强制重新渲染页面,这个非常消耗性能
                // this.$forceUpdate();

                //第二种方法
                this.$set(this.userInfo, "sex", "男");

                //可以实现新增的属性不能响应的问题
            }
        }
    })
</script>

```

> 在上面的代码上面，我们可以看到，最开始渲染的时候， `userInfo` 这个对象里面是没有 `sex` 这个属性的，但是后面我们主动追加了这个属性，结果，页面并没有正常显示这个属性，因为vue对新增的属性并没有实现响应
>
> 如果要解决这个问题，可以使用下面2个方法
>
> 1. `this.$forceUpdate()` 强制更新vue的状态，但是会非常消耗浏览器的性能
> 2. 使用 `this.$set()` 来扩展新属性，这样扩展出来的新属性 就是响应的

同理， `this.$delete` 也是一样的效果

```
<body>
    <div id="app">
        <h2 v-show="!userInfo.flag">{{userInfo.userName}}</h2>
        <button type="button" @click="aaa">按钮</button>
    </div>
</body>
<script src="./js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        data: {
            userInfo: {
                userName: "张三",
                flag: 123
            }
        },
        methods: {
            aaa() {
                //第一种情况
                // delete this.userInfo.flag;
                // this.$forceUpdate();

                this.$delete(this.userInfo, "flag");
            }
        }
    })
</script>
```

---

vue计算属性

> 之前我们讲过，vue是数据驱动页面的，它的数据主要来源于 `data` 这个区域，那么在这里我要扩展一下，接管区域里面的数据不一定仅仅只来源于data,还可以来源于其它的方

计算属性是需要经过计算以后才会得到的值，如下所示

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <hr>
        <h2>男生：3</h2>
        <h2>女生：2</h2>
    </div>
```

```
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        // 接管区域里面的数据来源于data
        data() {
            return {
                userName: "张三",
                stuList: [{
                    stuName: "岳圣哲",
                    sex: "男"
                }, {
                    stuName: "黄相立",
                    sex: "男"
                }, {
                    stuName: "王杜娉",
                    sex: "女"
                }, {
                    stuName: "马淑圆",
                    sex: "女"
                }, {
                    stuName: "兰宏宇",
                    sex: "男"
                }]
            }
        }
    }).mount("#app")
</script>
```

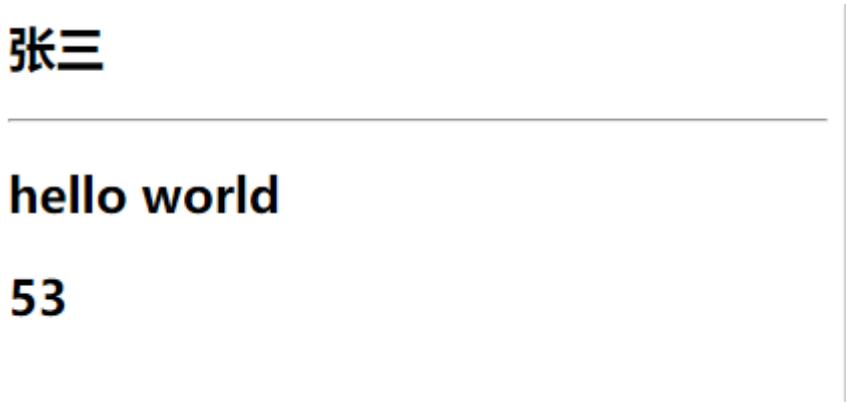
> 在上面的代码里面，如果我们想得到男或女的人数一共是多少，这样就非常困难，所以如果要实现这样的点，我们必须将数据经过计算以后才会得到结果，对于这一种需示，vue是可以实现的

现在我们先从一个简单的Demo来入手

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <hr>
        <h2>{{aaa}}</h2>
        <h2>{{bbb}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三"
            }
        },
        // 计算属性
        computed: {
            //每一个计算属性本质上面就是一个方法，只是这个方法有返回值了
            aaa() {
                return "hello world";
```

```
            },
            bbb() {
                return ~~(Math.random() * 100)
            }

        }
    }).mount("#app");
</script>
```

在上面的代码里面，我们可以的看到，页面的上面的数据不仅仅只来源于`data`，还可以来源于`computed`下面的计算属性



计算属笥的本质上面其实就是一个函数，这个函数需要有一个返回值页已

同时还要注意一点，计算属性也是可以访问`data`里面的数据的。如下所示

```
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三",
                age: 19
            }
        },
        // 计算属性
        computed: {
            //每一个计算属性本质上面就是一个方法，只是这个方法有返回值了
            aaa() {
                return "hello world";
            },
            bbb() {
                return ~~(Math.random() * 100)
            },
            ccc() {
                return this.age + 20;    //在灾里，我们使用了data里面的变量
            }

        }
    }).mount("#app");
</script>
```

经过了上面的demo以后，我们再来看一下刚刚的问题

```

<body>
    <div id="app">
        <h2>{{userName}}</h2>
```

```
        <hr>
        <h2>男生：{{boyCount}}</h2>
        <h2>女生：{{girlCount}}</h2>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        // 接管区域里面的数据来源于data
        data() {
            return {
                userName: "张三",
                stuList: [{
                    stuName: "岳圣哲",
                    sex: "男"
                }, {
                    stuName: "黄相立",
                    sex: "男"
                }, {
                    stuName: "王杜娉",
                    sex: "女"
                }, {
                    stuName: "马淑圆",
                    sex: "女"
                }, {
                    stuName: "兰宏宇",
                    sex: "男"
                }]
            }
        },
        computed: {
            // 男生数量
            boyCount() {
                let count = this.stuList.filter(item => item.sex === "男").length;
                return count;
            },
            //女生数量
            girlCount() {
                return this.stuList.filter(item => item.sex === "女").length;
            }
        }
    }).mount("#app")
</script>
```

> 上面的 `computed` 里面的东西就是计算属性，它是一个就去，对 `data` 里面的 `stuList` 做了二次计算，然后返回了一个结果，这个结果当前计算属性的值
> 在调用计算属性的时候，我们不用加上这个括号

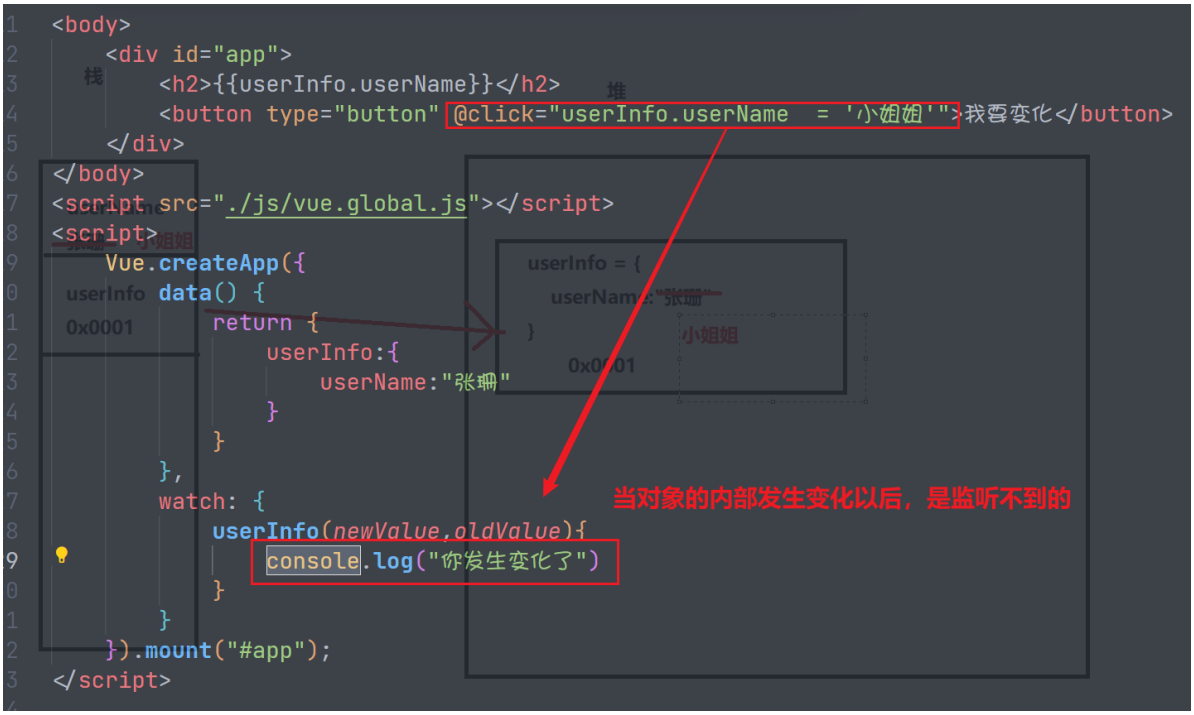
vue侦听属性（监听器）

```
<body>
    <div id="app">
        <h2>{{userName}}</h2>
        <button type="button" @click="userName='小姐姐'">改变userName的值</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userName: "张三",
            }
        },
        // 监听器
        watch: {
            //你要监听谁，你就哪个数据的同名函数
            userName(newValue, oldValue) {
                console.log("你发生了变化");
                // newValue代表更改之后的值  ，oldValue更改之前的值
                console.log(newValue, oldValue);
            }
        }
    }).mount("#app");
</script>
```

> 监听器就是写在`watch`下面的一个函数，如果要监听某一个变量，写写这个变量的同名函数，当这个变量发生变化的时候就会自动调用这个函数，同时这个函数会有2个参数，第一个参数代表变化以后的值，第二个参数代表变化之前的值

深度监听

在监听器里在，默认情况下，它只能监听基本数据类型 ，对于对象的变化它是监听不到的，如果想实现对象的监听，应该使用深度监听




```
<body>
    <div id="app">
        <h2>{{userInfo.userName}}</h2>
        <button type="button" @click="userInfo.userName  = '小姐姐'">我要变化
</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userInfo:{
                    userName:"张珊"
                }
            }
        },
        watch: {
            userInfo(newValue,oldValue){
                console.log("你发生变化了")
            }
        }
    }).mount("#app");
</script>
```

```
<body>
    <div id="app">
        <h2>{{userInfo.userName}}</h2>
        <button type="button" @click="userInfo.userName  = '小姐姐'">我要变化
</button>
    </div>
</body>
<script src="./js/vue.global.js"></script>
<script>
    Vue.createApp({
        data() {
            return {
                userInfo: {
                    userName: "张珊"
                }
            }
        },
        watch: {
            // userInfo(newValue,oldValue){
            //     console.log("你发生变化了")
            // }
            userInfo: {
                deep: true,
                handler(newValue,oldValue){
                    console.log(newValue)
                    console.log("我看你发生了变化了");
                }
            }
        }
    }).mount("#app");
</script>
```

上面的 `userInfo` 就是实现了深度监听，深度监听不仅仅只是一个单独的函数了，它是一个对象，这个对象上面有 `deep` 这个属性，用于设置是否处于深度监听状态，还有一个 `handler` 函数是变化以后的处理函数

vue样式class绑定

1. 对象语法

2.数组语法

vue样式style绑定

1.对象语法

2. 数组语法

vue的过滤器

> vue3已废弃

vue的DOM操作