Vue 第一天

VUE第一天

概念

是一套用于构建用户界面的　渐进式框架。

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。

完全能够为复杂的单页应用提供驱动。

整个生态系统的知识框架

- Vue.js 这是Vue的核心语法，用来构建web应用的视图。
- VueComponent 现代化的前端都采用组件化的开发思想，Vue同样提供了组件化开发的解决方案。
- Vue-Cli 官方提供的脚手架工具，用来构建大型的前端项目，基于webpack来构建。
- VueRouter 官方提供的路由器，实现单页应用的页面跳转。
- VueX 官方提供的状态管理工具，实现复杂应用的数据管理

模板语法

Vue初体验

页面上有一个计数器，初始值为0，放置一个按钮button，点击按钮，页面上的计算器自增1

原生JS代码

原生JS代码 Vue 复制代码

```
<body>
  <h2>0</h2>
    <button onclick="fn()">点我自增</button>
    <script>
    const h2 = document.querySelector('h2')
  const btn = document.querySelector('button')
  function fn(){
    h2.innerHTML = +h2.innerHTML + 1
  }
</script>
</body>
```

vue写法

在头标签中导入vue

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

设置挂载点

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

实例Vue对象

Vue 复制代码

```
new Vue({
  el:'#app',//理解为vue关联的视图区域
  data:{
    count:0
  },
  methods:{
    fn(){
      this.count++
    }
  }
})
```

DOM中使用data数据

```Vue
<div id="app">
<button @click="alert('这是提示内容')">按钮</button>
</div>
```

完整例子

vue 写法

```Vue
<body>
  <div id="app">
  <h2>{{count%2===0?'偶数':'奇数'}}</h2>
  <button @click="fn">点我自增</button>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
<script>
  new Vue({
    el:'#app',//理解为vue关联的视图区域
    data:{
      count:0
    },
    methods:{
      fn(){
        this.count++
      }
    }
  })
</script>
</body>
```

只需要关注数据本身的变化，遵守Vue语法控制试图即可

事件绑定指令

```vue
<body>
    <div id="app">
                                        <!-- {{}}叫做插值表达式 -->
      <h2 v-on:mouseover="fn1">{{count%2===0?'偶数':'奇数'}}</h2>
      <button v-on:click="fn">点我自增</button>
    //第一个指令 v-on:用来绑定事件的 它有一个简写的形式 @----即 <button @click="fn">
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
    <script>
      const vm = new Vue({//vue管理的函数不能写成箭头函数 必须写成普通函数
        el: "#app", //理解为vue关联的视图区域
        data: { //理解为存储所有视图区域的数据的对象
          count: 0,
        },
        methods: { //理解为存储所有视图区域事件回调的对象
          fn(){   //如果在methods配置选项中希望访问到data中的数据 必须通过this调用
           setTimeout(()=>{  //定时器的回调函数不属于vue管理的函数 如果希望内部访问this必须写成箭头函数
            this.count++     //让count自增
           },1000)
          },
          fn1(){
            console.log('鼠标移入了')
          }
        },
      })
      console.log(vm)
    </script>
  </body>
```

事件绑定细节

```vue
<body>
    <div id="app">
    <!-- <h1>
    {{count}}
  </h1> -->
    <button @click="alert('这是提示内容')">按钮</button>
      </div>
      <script src="./vue.js"></script>
    <script>
      new Vue({
        el: '#app',
        data: { //提取data的数据需要用this提取，如果是在模板是直接访问不用加 this
        },
        methods:{
          alert(msg){
            alert(msg)
          }
        }
      })
    </script>
</body>
```

事件绑定的传参

```vue
<body>
    <div id="app">
        <!-- 如果希望给事件传递参数 就必须加上小括号 -->
        <!-- 通过$event可以获取原生DOM -->
        <button @click="say('hi',$event)">say hi</button>
        <button @click="say('hello')">say hello</button>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {

            },
            methods:{
                //如果希望显示地接受事件对象的话 比如在事件触发时手动地传入$event的参数
                say(msg,e){
                    console.log(msg,e)
                }
            }
            })
    </script>
</body>
```

常见Vue指令

1.v-bind　动态地绑定标签内部的属性

```Vue
<div id="app">
        <!-- 点击a标签跳转到百度去 -->
        <!-- 如果希望动态地绑定标签内部的属性 必须通过v-bind:属性名 这个指令  可以简写成:-->
        <a :href="url">{{count}}</a>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                count:'百度',
                url:"http://www.baidu.com"
            }
            })
    </script>
```

2，v-if 用于条件渲染

```Vue
<body>
    <div id="app">
        <!-- v-if后面是通过布尔值来确认元素的显示与否 -->
        <!-- 第三个指令 v-if后面的布尔值如果为真 元素显示 否则元素隐藏 -->
        <h1 v-if="show">这是一个动态显示的标题</h1>
        <button @click="show = !show">点我切换标题的显示隐藏</button>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show:true
            }
         })
    </script>
</body>
```

3，v-else 也用于条件渲染 但是需要紧跟在v-if指令后面的 表示和v-if相反的显示

```vue
<body>
    <div id="app">
        <!-- v-else是需要紧跟在v-if指令后面的 表示和v-if相反的显示 -->
        <button @click="show = !show">点我切换标题的显示隐藏</button>
        <h1 v-if="show">标题1</h1>
        <h1 v-else>标题2</h1>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show:true
            }
         })
    </script>
</body>
```

4，template 渲染多个元素的条件渲染

```vue
<body>
    <div id="app">
       <!-- template是vue中给我们提供的一个空白标签 -->
       <button @click="show = !show">点我切换标题的显示隐藏</button>
       <template v-if="show">
        <h1>标题1</h1>
        <h1>标题2</h1>
        <h1>标题3</h1>
       </template>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show:true
            }
         })
    </script>
</body>
```

5，v-show 用于条件展示

Vue | 复制代码

```vue
<body>
    <div id="app">
        <!-- v-show是控制display的变化 v-if是控制元素是否渲染到页面之中 -->
        <h2 v-show="show">动态显示的标题</h2>
        <button @click="show = !show">按钮</button>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                show:true
            }
            })
    </script>
</body>
```

5.5， v-if 和 v-show的区别

v-if需要满足渲染条件，切换过程中组件会被删除。条件少用if

v-show 不管开始条件是什么，都会渲染，在css中进行切换占用内存少

6，v-for 用于循环渲染

Vue | 复制代码

```vue
<body>
    <div id="app">
        <ul>
            <!-- of 和 in 都可以实现循环效果 -->
            <li v-for="(item,index) in friends">
                序号:{{item.id}}--姓名:{{item.name}}--年龄：{{item.age}}--性别:{{item.gender}} --<button @click="del(index)">点我删除这条数据</button>
            </li>
        </ul>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                friends:[
                    {…},
                    {…},{
                        id:Math.random().toFixed(1),
                        name:'suxi',
                        age:3,
                        gender:'girl'
                    }
                ]
            },
            methods:{
                del(index){
                    //删除对应的那条数据
                    //在一个数组中删除某个索引值的数组元素
                   this.friends.splice(index,1)
                }
            }
        })
    </script>
</body>
```

6.1，v-for 渲染对象

```vue
<ul id="app" class="demo">
  <li v-for="(value,key,index) in object">
    {{ value }}
    {{ key }}
    {{ index }}
  </li>
</ul>
new Vue({
  el: '#app',
  data: {
    object: {
      title: 'How to do lists in Vue',
      author: 'Jane Doe',
      publishedAt: '2016-04-10'
    }
  }
})
```

6.2，v-for 渲染数字

```Vue
<body>
    <div id="app">
        <ul>
            <!-- 对象数据也可以渲染 普通数字也可以渲染 但实际渲染的数据都是数组 -->
            <li v-for="(value,index) in '10'">
               {{value}}--{{index}}
            </li>
        </ul>
    </div>
    <script src="./vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                person:{
                    name:'peiqi',
                    age:4,
                    gender:'girl'
                }
            }
            })
    </script>
</body>

```

6.5 v-for和v-if的使用

如果 v-for和v-if 放在同一个元素上，v-for的优先级更高，为了防止此事件发生，将v-if放到v-for外层元素， 这样v-if会根据条件选择性的执行循环，性能更好。

7，v-html　　v-text(用于插值)

v-html 可以添加文本样式 比如<h1>、等 Vue 复制代码

```vue
<div id="app">
        <div v-html="msg"></div>
    </div>
    <script src="/vue.js"></script>
    <script>
        new Vue({
            el:"#app",
            data:{
                msg:"<em>hello world </em>"
            }
        })
    </script>
```

v-text (用于插值)单纯的插入值 Vue 复制代码

```vue
<div id="app">
        <!-- 使用插值表达式和使用v-text指令的效果是一致的 -->
        <div>{{msg}}</div>
        <div v-text="msg"></div>
    </div>
    <script src="/vue.js"></script>
    <script>
        new Vue({
            el:"#app",
            data:{
                msg:"hello world"
            }
        })
    </script>
```

8，计算属性 computed

Vue | 复制代码

```vue
<div id="app">
        <!-- 如果模板中能直接使用一个变量 那么这个变量一定被会挂在到vue实例身上 -->
        <!-- data,methods,computed配置选项中的数据都会挂在到模板上 -->
        <h2>{{reverse}}</h2>
        <h3>{{count}}</h3>
        <button @click="msg+=1">按钮</button>
    </div>
    <script src="./vue.js"></script>
    <script>
        const vm = new Vue({
            el: '#app',
            data: {
                msg:'hello',
                count:0
            },
            computed:{
                //3.如果计算属性依赖的数据没有变化 它不会重新计算 同样能提高性能
                reverse(){
                    console.log('计算属性被调用')
                    //计算属性必须通过返回值的形式表达
                    return this.msg.split('').reverse().join('')
                }
            }
            })
            console.log(vm)
    </script>
```

1.计算属性在调用的时候不要加() 方法 mehdos需要加()

2.计算属性它有一个缓存功能 大大提高页面性能 多次调用只计算一次

3.如果计算属性依赖的数据没有变化 它不会重新计算 同样能提高性能