

第2章 VUE 基础语法

本章中，将会讲解生命周期函数，指令，模版，数据，侦听器，事件，循环渲染等基础语法知识点，帮助大家理解第一章重写过的代码，同时理解数据驱动的编程思想。

H2 Vue 中应用和组件的基础概念

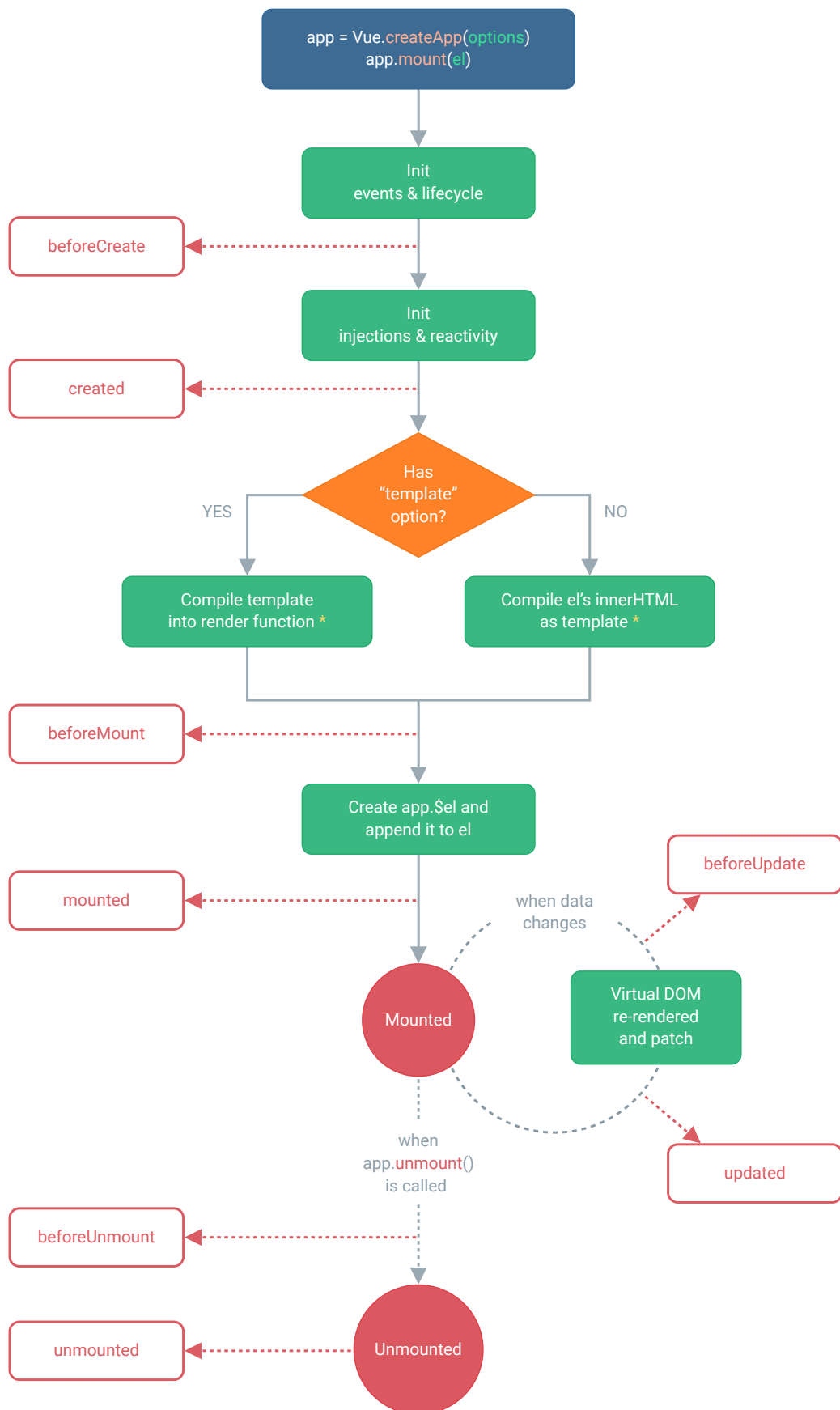
- createApp 表示创建一个 Vue 应用, 存储到 app 变量中
- 传入的参数表示, 这个应用最外层的组件, 应该如何展示
- MVVM 设计模式, M -> Model 数据, V -> View 视图, VM -> ViewModel 视图数据连接层
- vm 代表的就是 Vue 应用的根组件
- 获取根数据 用类似 `vm.$data.message`

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>lesson 5</title>
7   <script src="https://unpkg.com/vue@next"></script>
8 </head>
9
10 <body>
11   <div id="root"></div>
12 </body>
13 <script>
14   // createApp 表示创建一个 Vue 应用, 存储到 app 变量中
15   // 传入的参数表示, 这个应用最外层的组件, 应该如何展示
16   // MVVM 设计模式, M -> Model 数据, V -> View 视图, VM -> ViewModel 视图
    数据连接层
17   const app = Vue.createApp({
18     data() {
19       return {
20         message: 'hello world'
21       }
22     }
23   })
24   app.mount('#root')
```

```
22     },
23     template: "<div>{{message}}</div>"
24   });
25   // vm 代表的就是 Vue 应用的根组件
26   const vm = app.mount('#root');
27 </script>
28
29 </html>
```

H2 理解 **Vue** 中的生命周期函数

- `destroyed` 生命周期选项被重命名为 `unmounted`
- `beforeDestroy` 生命周期选项被重命名为 `beforeUnmount`



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



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta http-equiv="X-UA-Compatible" content="IE=edge">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>生命周期</title>
8   <script src="https://unpkg.com/vue@next"></script>
9 </head>
10 <body>
11   <div id='root'></div>
12 </body>
13 <script>
14   // 生命周期函数：在某一时刻会自动执行的函数
15   const app = Vue.createApp({
16     data(){
17       return{
18         message: 'hello'
19       }
20     },
21     // 在实例生成前会自动执行的函数
22     beforeCreate() {
23       console.log('beforeCreate')
24     },
25     // 在实例生成后会自动执行的函数
26     created() {
27       console.log('create')
28     },
29     // 在组件内容渲染到页面之前自动执行的函数
30     beforeMount() {
31       console.log('beforeMount')
32     },
33     // 在组件内容渲染到页面之后自动执行的函数
34     mounted() {
35       console.log('mounted')
36     },
37     // 在 数据发生变化时会立即执行的函数
38     beforeUpdate() {
39       console.log('beforeUpdate')
40     },
41     // 在 数据发生变化，页面重新渲染后，会自动执行的函数
42     updated() {
43       console.log('updated')
44     },
45     // 当 Vue 应用失效时，自动执行的函数
```

```

46     beforeUnmount() {
47         console.log('beforeUnmount')
48     },
49     // 当vue应用失效, 且 dom 完全销毁后, 自动执行的函数
50     Unmount() {
51         console.log('Unmount')
52     },
53     template: "<div>{{message}}<div>"
54 })
55 app.mount('#root')
56 </script>
57 </html>

```

H2 常用模版语法讲解

H3 插值表达式 {{}}

{{messgae}}

H3 一些指令

- v-html
- v-bind(:)
- v-on(@),
- 动态参数
 - :[name]='message'
 - @[event]='handleClick'
- 修饰符
 - @click.prevent
- v-if
- v-once 变量只使用一次

```

data() {
  return {
    message: "hello world"
  },
  template: `<div v-once>{{message}}</div>`
}

```

H2 数据，方法，计算属性和侦听器

- data & methods & computed & watcher
- computed 和 method 都能实现的一个功能，建议使用 computed，因为有缓存
- computed 和 watcher 都能实现的功能，建议使用 computed 因为更加简洁

H3 method:{}

- 有用this的话，不使用箭头函数
- handleClick () {console.log} //当前vue实例
- handleClick:()=>{console.log(this)} // undefined

H3 监听

可以异步操作 比如说setTimeout

```

1 watch: {
2   // price 发生变化时，函数会执行
3   price(current, prev) {
4     this.newTotal = current * this.count;
5   }
6 },
7

```

H3 计算属性

不能异步操作

```
1   computed: {  
2     // 当计算属性依赖的内容发生变更时，才会重新执行计算  
3     total() {  
4       return Date.now() + this.count;  
5       // return this.count * this.price  
6     }  
7   },
```

H2 样式绑定语法

H3 字符串

```
const app = Vue.createApp({  
  data() {  
    return {  
      classString: 'red',  
    },  
  },  
  template: `<div :class="classString">Hello World</div>`,  
});  
const vm = app.mount('#root');
```

The diagram illustrates the binding of the `classString` property to the `:class` attribute in the template. A red arrow points from the `classString` property in the `data()` object to the `classString` variable in the template string `<div :class="classString">Hello World</div>`. Another red arrow points from the `classString` variable to the `.red{color: red}` CSS class, indicating that the value of `classString` is used to determine the applied CSS class.

H3 对象

```

data() {
  return {
    classString: 'red',
    classObject: { red: false, green: true },
  },
  template: `
    <div :class="classObject">Hello World</div>
  `
}

```

对象的方式
true 展示
false 不展示

H3 数组

```

    classArray: ['red', 'green', {brown: false}],
  },
  template: `
    <div :class="classArray">Hello World</div>
  `

```

H3 子组件获取父组件上的颜色

- 单一元素的话 子组件不用做啥，父组件有class 就行
- 多个元素的话，父组件不知道将class 给谁

```

1 //父
2 <demo class='green' />
3
4 //子
5 <div :class="$attrs.class">one</div>
6 <div :class="$attrs.class">two</div>

```

H3 行内样式


```
1 styleString: 'color: yellow;background: orange',
2 styleObject: {
3   color: 'orange',
4   background: 'yellow'
5 }
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>lesson 9</title>
7   <style>
8     .red {
9       color: red;
10    }
11    .green {
12      color: green;
13    }
14  </style>
15  <script src="https://unpkg.com/vue@next"></script>
16 </head>
17 <body>
18   <div id="root"></div>
19 </body>
20 <script>
21   const app = Vue.createApp({
22     data() {
23       return {
24         classString: 'red',
25         classObject: { red: false, green: true },
26         classArray: ['red', 'green', {brown: false}],
27         styleString: 'color: yellow;background: orange',
28         styleObject: {
29           color: 'orange',
30           background: 'yellow'
31         }
32       }
33     },
```

```

34     template: `
35       <div :style="styleObject">
36         Hello World
37       </div>
38     `
39   });
40
41   app.component('demo', {
42     template: `
43       <div :class="$attrs.class">one</div>
44       <div :class="$attrs.class">two</div>
45     `
46   })
47
48   const vm = app.mount('#root');
49 </script>
50 </html>
51

```

H2 条件渲染

- v-if 直接控制DOM
 - v-if
 - v-else-if
 - v-else
- v-show display: none 频繁展示消失就用 v-show

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>lesson 10</title>
7   <script src="https://unpkg.com/vue@next"></script>
8 </head>
9 <body>
10   <div id="root"></div>
11 </body>
12 <script>
13   const app = Vue.createApp({
14     data() {
15       return {

```

```

16     show: false,
17     conditionOne: false,
18     conditionTwo: true
19   }
20 },
21 template: `
22   <div v-if="show">Hello World</div>
23
24   <div v-if="conditionOne">if</div>
25   <div v-else-if="conditionTwo">elseif</div>
26   <div v-else>else</div>
27
28   <div v-show="show">Bye World</div>
29 `
30 });
31
32 const vm = app.mount('#root');
33 </script>
34 </html>
35

```

H2 列表循环渲染

H3 数组 v-for = “ (item,index) in Array”

```

data() {
  return {
    list: ['dell', 'lee', 'teacher']
  }
},
template: `
  <div>
    <div v-for="(item, index) in list">
      {{item}} -- {{index}}
    </div>
  </div>
`

```

H3 对象 v-for="(value,key,index) in Obj"

```
listObject: {
  firstName: 'dell',
  lastName: 'lee',
  job: 'teacher'
},
template: `
<div>
  <div v-for="(value, key, index) in listObject">
    {{value}} -- {{key}} -- {{index}}
  </div>
</div>
`
```

H3 使用数组的变更函数 push, pop, shift, unshift, splice, sort, reverse

```
1  const app = Vue.createApp({
2    data() {
3      return {
4        listArray: ['dell', 'lee', 'teacher'],
5        listObject: {
6          firstName: 'dell',
7          lastName: 'lee',
8          job: 'teacher'
9        }
10     }
11   },
12   methods: {
13     handleAddBtnClick() {
14       // 1. 使用数组的变更函数 push, pop, shift, unshift, splice, sort,
reverse
15       this.listArray.push('hello');
16       this.listArray.pop();
17       this.listArray.shift();
18       this.listArray.unshift('hello');
19       this.listArray.reverse();
20     }
21   },
22   template: `
23     <div>
24       <div v-for="(item,index) in listArray " :key='index'>
```

```

25     {{item}}----{{index}}
26   </div>
27   <button @click="handleAddBtnClick"
28 </div>
29   `
30   });
31
32   const vm = app.mount('#root');

```

H3 直接替换数组

```

1  const app = Vue.createApp({
2    data() {
3      return {
4        listArray: ['dell', 'lee', 'teacher'],
5        listObject: {
6          firstName: 'dell',
7          lastName: 'lee',
8          job: 'teacher'
9        }
10     },
11   },
12   methods: {
13     handleAddBtnClick() {
14       // 2. 直接替换数组
15       this.listArray = ['bye', 'world']
16       this.listArray = ['bye', 'world'].filter(item => item !== 'bye');
17     }
18   },
19   template: `
20     <div>
21       <div v-for="(item,index) in listArray " :key='index'>
22         {{item}}----{{index}}
23       </div>
24       <button @click="handleAddBtnClick"
25     </div>
26     `
27   });
28
29   const vm = app.mount('#root');

```

H3 直接更新数组的内容

```
1  const app = Vue.createApp({
2    data() {
3      return {
4        listArray: ['dell', 'lee', 'teacher'],
5        listObject: {
6          firstName: 'dell',
7          lastName: 'lee',
8          job: 'teacher'
9        }
10     },
11   },
12   methods: {
13     handleAddBtnClick() {
14       // 3. 直接更新数组的内容
15       this.listArray[1] = 'hello'
16     }
17   },
18   template: `
19     <div>
20       <div v-for="(item,index) in listArray " :key='index'>
21         {{item}}----{{index}}
22       </div>
23       <button @click="handleAddBtnClick"
24     </div>
25   `
26 });
27
28 const vm = app.mount('#root');
```

H3 直接添加对象的内容，也可以自动的展示出来

```
1      this.listObject.age = 100;
2      this.listObject.sex = 'male';
```

H3 可以直接循环数字

```
1 <div v-for='item in 10'>
2   {{item}}
3 </div>
4 // 1 2 3 4 5 6 7 8 9 10
```

H3 切记 v-for 和 v-if 不能写在同层 v-for 优先级更高

```
1 <div v-for='item in 10' v-if='false'>
2   {{item}}
3 </div>
4 -----
5 <div v-for='item in 10' >
6   <div v-if='false'>
7     {{item}}
8   </div>
9 </div>
10 -----
11 //这里我们不渲染的时候还会有个div 空标签 优化下用template 相当于占位符
12 <template v-for='item in 10' >
13   <div v-if='false'>
14     {{item}}
15   </div>
16 </template>
```

H2 事件绑定

event, \$event

事件修饰符: stop, prevent, capture, self, once, passive

按键修饰符: enter, tab, delete, esc, up, down, left, right

鼠标修饰符: left, right, middle

精确修饰符: exact

H3 原生对象获取

```
methods: {  
  handleBtnClick(num, event) {  
    console.log(event);  
    this.counter += num;  
  },  
},  
template: `  
  <div>  
    {{counter}}  
    <button @click="handleBtnClick(2, $event)">button</button>  
  </div>  
`
```

如果不传参数可以直接
handleBtnClick (event) {
 console.log(event)
}

```
1 <button @click="handleBtnClick()">点击按钮</button>
```

```
1 <button @click="handleBtnClick(2,$event)">点击按钮</button>
```

H3 一个按钮绑定多个函数

```
1 <button @click="handleBtnClick(),handleBtnClick1()">点击按钮</button>
```

H3 事件修饰符

- stop
 - 阻止冒泡
- prevent
 - 阻止默认行为
- capture
 - 捕获 ---- 从外到内
- self

- 点击自己才实现
- once
 - 只执行一次
- passive
 -

H3 按键修饰符

- enter
- tab
- delete
- esc
- up
- down
- left
- right

```
1 <input @keydown.enter = "handleKeyDown" />
2
3 method:{
4   handleKeyDown(){
5     console.log('keydown')
6   }
7 }
```

H3 鼠标修饰符

left, right, middle

```
1 <input @click.left = "handleKeyDown" />
```

H3 精确修饰符

exact

```
1 <div @click.ctrl.exact="handleClick">123</div>
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>lesson 12</title>
7   <script src="https://unpkg.com/vue@next"></script>
8 </head>
9 <body>
10   <div id="root"></div>
11 </body>
12 <script>
13   // event, $event
14   // 事件修饰符: stop, prevent, capture, self, once, passive
15   // 按键修饰符: enter, tab, delete, esc, up, down, left, right
16   // 鼠标修饰符: left, right, middle
17   // 精确修饰符: exact
18   const app = Vue.createApp({
19     methods: {
20       handleClick() {
21         console.log('click')
22       },
23     },
24     template: `
25       <div>
26         <div @click.ctrl.exact="handleClick">123</div>
27       </div>
28     `
29   });
30
31   const vm = app.mount('#root');
32 </script>
33 </html>
34
```

H2 表单中双向绑定指令的使用

H3 input



```
1 <input v-model="message" />
```

H3 textarea



```
1 <textarea v-model="message" />
```

H3 checkbox



```
1 <input type="checkbox" v-model="message" />  
2 message 只能是 true false
```

高级使用 message:[] 在CheckBox里有value

```
<div>  
  {{message}}  
  jack <input type="checkbox" v-model="message" value="jack" />  
  dell <input type="checkbox" v-model="message" value="dell" />  
  lee <input type="checkbox" v-model="message" value="lee" />  
</div>
```

H3 radio

```
1 <input type="radio" v-model="message" value='jack' />
2 // radio 只能是一个 所以 message:""
```

H3 select

```
1 //单选
2 // message:""
3 <select v-model="message">
4   <option disabled value=''>请选择内容</option>
5   <option value='A'>A</option>
6   <option value='B'>B</option>
7   <option value='C'>C</option>
8 </select>
```

```
1 //多选
2 // message:[]
3 <select v-model="message" multiple>
4   <option disabled value=''>请选择内容</option>
5   <option value='A'>A</option>
6   <option value='B'>B</option>
7   <option value='C'>C</option>
8 </select>
```

H3 修饰符-lazy

```
<input v-model.lazy="message" />
```

H3 修饰符-number



```
1 <input v-model.number="message" type='number' />
2 //类型装换为number 不然本来输入的是string
```

H3 修饰符- trim



```
1 <input v-model.trim="message" />
2 //去除前后空格
```



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>lesson 13</title>
7   <script src="https://unpkg.com/vue@next"></script>
8 </head>
9 <body>
10   <div id="root"></div>
11 </body>
12 <script>
13   // input, textarea, checkbox, radio, select
14   // 修饰符 lazy, number, trim
15   const app = Vue.createApp({
16     data() {
17       return {
18         message: 'hello',
19       }
20     },
21     template: `
22       <div>
23         {{message}}
24         <input v-model.trim="message" />
25       </div>
26     `,
27   });
```

```
28
29     const vm = app.mount('#root');
30 </script>
31 </html>
32
```