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

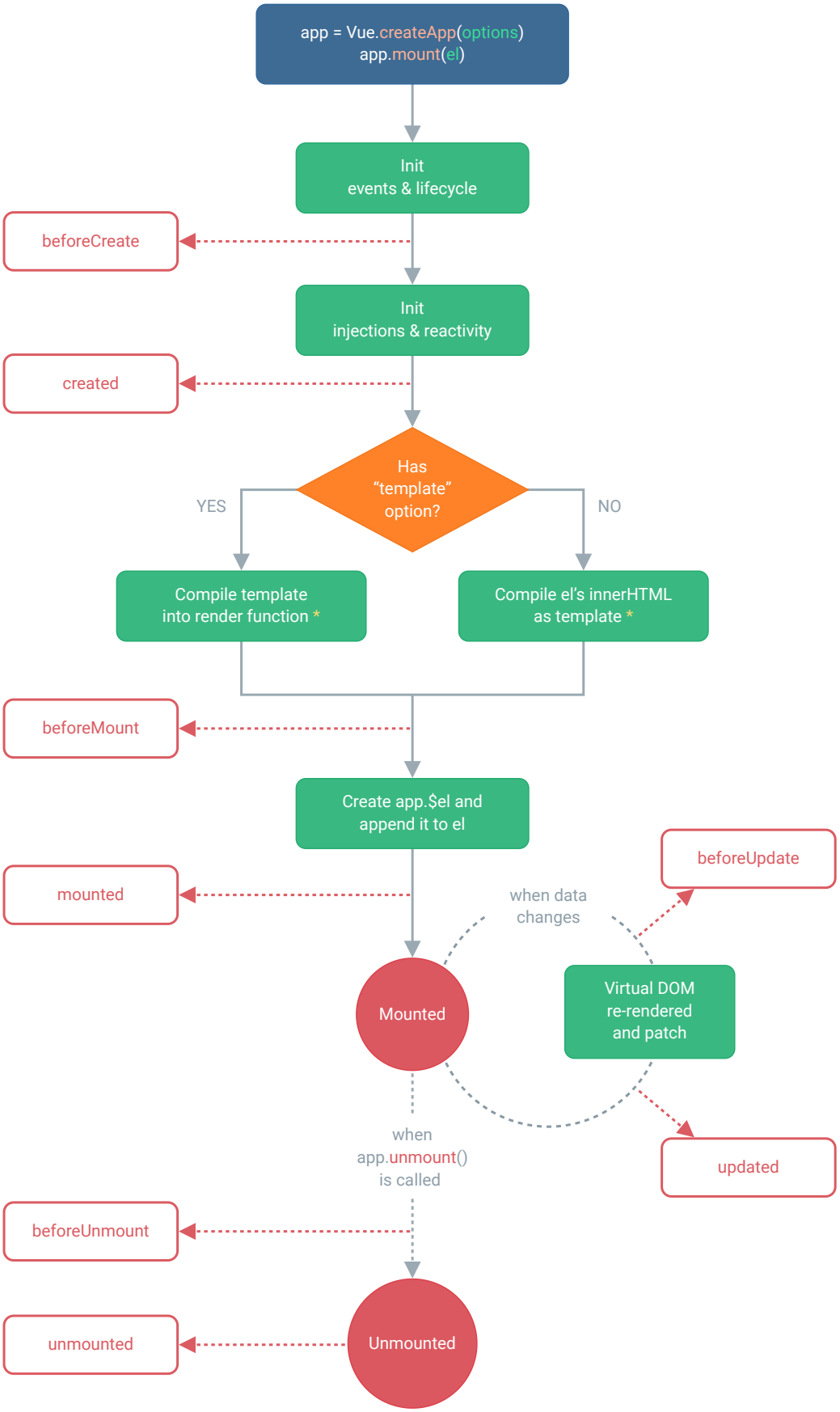
- createApp 表示创建一个 Vue 应用, 存储到 app 变量中
- 传入的参数表示，这个应用最外层的组件，应该如何展示
- MVVM 设计模式，M -> Model 数据， V -> View 视图， VM -> ViewModel 视图数据连接层
- vm 代表的就是 Vue 应用的根组件
- 获取根数据 用类似 `vm.$data.message`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>lesson 5</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>

<body>
  <div id="root"></div>
</body>
<script>
  // 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
  const app = Vue.createApp({
    data() {
      return {
        message: 'hello world'
      }
```

```
    },
    template: "<div>{{message}}</div>"
  });
  // vm 代表的就是 Vue 应用的根组件
  const vm = app.mount('#root');
</script>

</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.

```html
<!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>生命周期</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id='root'></div>
</body>
<script>
  // 生命周期函数：在某一时刻会自动执行的函数
  const app = Vue.createApp({
    data(){
      return{
        message:'hello'
      }
    },
    //  在实例生成前会自动执行的函数
    beforeCreate() {
      console.log('beforeCreate')
    },
    // 在实例生成后会自动执行的函数
    created() {
      console.log('create')
    },
    // 在组件内容渲染到页面之前自动执行的函数
    beforeMount() {
      console.log('beforeMount')
    },
    //   在组件内容渲染到页面之后自动执行的函数
    mounted() {
      console.log('mounted')
    },
    //  在 数据发生变化时会立即执行的函数
    beforeUpdate() {
      console.log('beforeUpdate')
    },
    //  在 数据发生变化，页面重新渲染后，会自动执行的函数
    updated() {
      console.log('updated')
    },
    // 当 Vue 应用失效时，自动执行的函数
```

```
    beforeUnmount() {
      console.log('beforeUnmount')
    },
    // 当vue应用失效，且 dom 完全销毁后，自动执行的函数
    Unmount() {
      console.log('Unmount')
    },
    template:"<div>{{message}}<div>"
  })
  app.mount('#root')
</script>
</html>
```

常用模版语法讲解

插值表达式 {{}}

{{messgae}}

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>`
```

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

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)} // undifined

监听

> 可以异步操作 比如说setTimeout

```
watch: {
  // price 发生变化时，函数会执行
  price(current, prev) {
    this.newTotal = current * this.count;
  }
},
```

计算属性

> 不能异步操作

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

样式绑定语法

字符串

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

数组

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

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

```
//父
<demo class='green' />

//子
<div :class="$attrs.class">one</div>
<div :class="$attrs.class">two</div>
```

行内样式

```
        styleString: 'color: yellow;background: orange',
        styleObject: {
          color: 'orange',
          background: 'yellow'
        }
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>lesson 9</title>
  <style>
    .red {
      color: red;
    }
    .green {
      color: green;
    }
  </style>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="root"></div>
</body>
<script>
  const app = Vue.createApp({
    data() {
      return {
        classString: 'red',
        classObject: { red: false, green: true },
        classArray: ['red', 'green', {brown: false}],
        styleString: 'color: yellow;background: orange',
        styleObject: {
          color: 'orange',
          background: 'yellow'
        }
      }
    },
```

```
    template: `
      <div :style="styleObject">
        Hello World
      </div>
    `
  });

  app.component('demo', {
    template: `
      <div :class="$attrs.class">one</div>
      <div :class="$attrs.class">two</div>
    `
  })

  const vm = app.mount('#root');
</script>
</html>

```

条件渲染

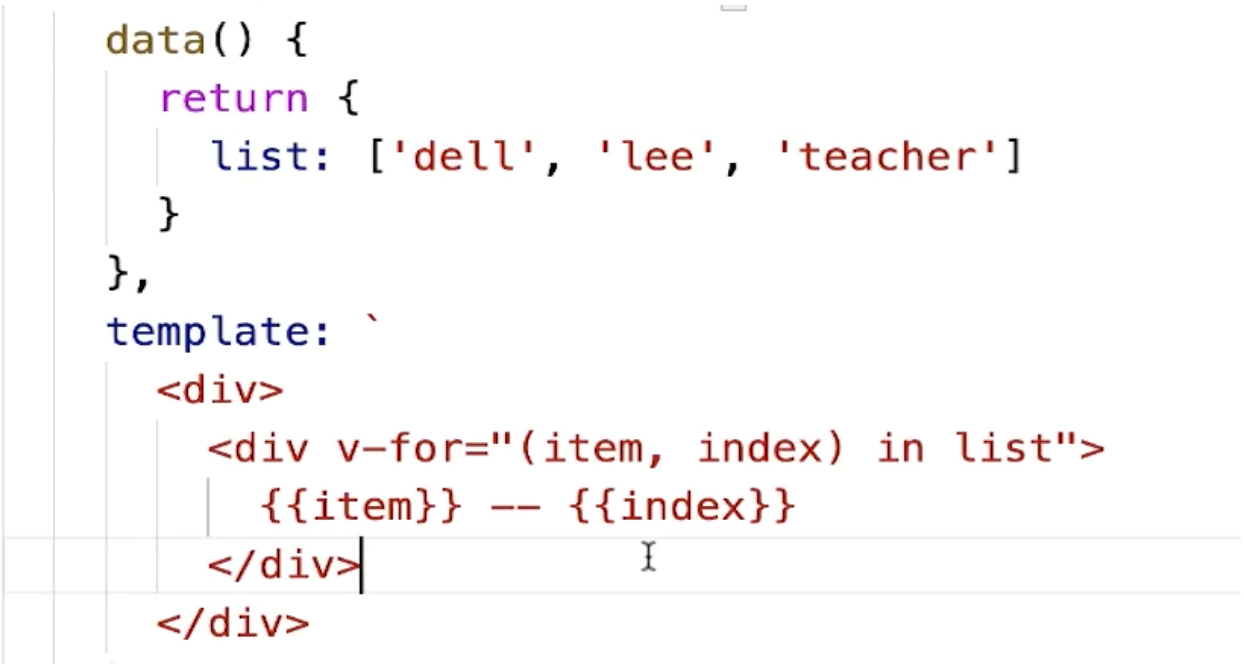
- v-if 直接控制DOM
    - v-if
    - v-else-if
    - v-else
- v-show display：none 频繁展示消失就用 v-show

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>lesson 10</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="root"></div>
</body>
<script>
  const app = Vue.createApp({
    data() {
      return {
```

```
        show: false,
        conditionOne: false,
        conditionTwo: true
      }
    },
    template: `
      <div v-if="show">Hello World</div>

      <div v-if="conditionOne">if</div>
      <div v-else-if="conditionTwo">elseif</div>
      <div v-else>else</div>

      <div v-show="show">Bye World</div>
    `
  });

  const vm = app.mount('#root');
</script>
</html>

```

列表循环渲染

数组 v-for =“（item,index）in Array”



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>
  `
```

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

```
const app = Vue.createApp({
    data() {
      return {
        listArray: ['dell', 'lee', 'teacher'],
        listObject: {
          firstName: 'dell',
          lastName: 'lee',
          job: 'teacher'
        }
      }
    },
    methods: {
      handleAddBtnClick() {
        // 1. 使用数组的变更函数 push, pop, shift, unshift, splice, sort, reverse
        this.listArray.push('hello');
        this.listArray.pop();
        this.listArray.shift();
        this.listArray.unshift('hello');
        this.listArray.reverse();
      }
    },
    template: `
    <div>
      <div v-for="(item,index) in listArray " :key='index'>
```

```
      {{item}}----{{index}}
    </div>
    <button @click="handleAddBtnClick"
  </div>
  `
});

const vm = app.mount('#root');
```

直接替换数组

```
const app = Vue.createApp({
    data() {
      return {
        listArray: ['dell', 'lee', 'teacher'],
        listObject: {
          firstName: 'dell',
          lastName: 'lee',
          job: 'teacher'
        }
      }
    },
    methods: {
      handleAddBtnClick() {
        // 2. 直接替换数组
        this.listArray = ['bye', 'world']
        this.listArray = ['bye', 'wolrd'].filter(item => item === 'bye');
      }
    },
    template: `
    <div>
      <div v-for="(item,index) in listArray " :key='index'>
       {{item}}----{{index}}
      </div>
      <button @click="handleAddBtnClick"
    </div>
    `
});

const vm = app.mount('#root');
```

直接更新数组的内容

```
const app = Vue.createApp({
    data() {
      return {
        listArray: ['dell', 'lee', 'teacher'],
        listObject: {
          firstName: 'dell',
          lastName: 'lee',
          job: 'teacher'
        }
      }
    },
    methods: {
      handleAddBtnClick() {
        // 3. 直接更新数组的内容
        this.listArray[1] = 'hello'
      }
    },
    template: `
    <div>
      <div v-for="(item,index) in listArray " :key='index'>
       {{item}}----{{index}}
      </div>
      <button @click="handleAddBtnClick"
    </div>
    `
  });

  const vm = app.mount('#root');
```

直接添加对象的内容，也可以自动的展示出来

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

可以直接循环数字

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

切记 v-for 和 v-if 不能写在同层 v-for优先级更高

```
<div v-for='item in 10' v-if='false'>
    {{item}}
</div>
------------------------------------------------------------------------
<div v-for='item in 10' >
    <div v-if='false'>
        {{item}}
    </div>
</div>
------------------------------------------------------------------------
//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
<template v-for='item in 10' >
    <div v-if='false'>
        {{item}}
    </div>
</template>
```

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

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)
}

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

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

一个按钮绑定多个函数

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

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
    -

按键修饰符

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

```
<input @keydown.enter = "handleKeyDown" />

method:{
handleKeyDown(){
console.log('keydown')
}
}
```

鼠标修饰符

left, right, middle

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

精确修饰符

exact

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

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>lesson 12</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="root"></div>
</body>
<script>
  // 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
  const app = Vue.createApp({
    methods: {
      handleClick() {
        console.log('click')
      },
    },
    template: `
      <div>
        <div @click.ctrl.exact="handleClick">123</div>
      </div>
    `
  });

  const vm = app.mount('#root');
</script>
</html>

```

表单中双向绑定指令的使用

input

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

textarea

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

checkbox

```
<input type="checkbox" v-model="message" />
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>
```

radio

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

select

```
//单选
// message:""
<select v-model="message">
     <option disabled value=''>请选择内容</option>
     <option value='A'>A</option>
     <option value='B'>B</option>
     <option value='C'>C</option>
</select>
```

```
//多选
// message:[]
<select v-model="message" multiple>
     <option disabled value=''>请选择内容</option>
     <option value='A'>A</option>
     <option value='B'>B</option>
     <option value='C'>C</option>
</select>
```

修饰符-lazy

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

修饰符-number

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

修饰符- trim

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

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>lesson 13</title>
  <script src="https://unpkg.com/vue@next"></script>
</head>
<body>
  <div id="root"></div>
</body>
<script>
  // input, textarea, checkbox, radio, select
  // 修饰符 lazy, number, trim
  const app = Vue.createApp({
    data() {
      return {
        message: 'hello',
      }
    },
    template: `
      <div>
        {{message}}
        <input v-model.trim="message"  />
      </div>
    `
  });
```

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