Vue 3.0从入门到精通

第一阶段 Vue 3.0零基础入门

1.1 核心API

1.1.1 快速起始

安装

```
<script src="https://unpkg.com/vue@next"></script>
```

起始

01-hello.html

```
<div id="app">
  {{title}}
</div>
<script>
  Vue.createApp({
    data() {
      return {
        title: 'hello, vue3!'
      }
    }
  }).mount('#app')
</script>
```

1.1.2 模板语法

插值 - interpolations

数据绑定最常见的形式就是使用“Mustache”语法 (双大括号) 的文本插值

范例：设置标题

```
<div id="app">
  <h2>
    <!-- 插值文本 -->
    {{title}}
  </h2>
</div>
```

特性 - attributes

HTML特性不能用Mustache语法，应该使用v-bind指令

范例：设置标题

```
<div id="app">
  <!-- 特性、属性值绑定使用v-bind指令 -->
  <h2 v-bind:title="title">...</h2>
</div>
```

HTML内容

HTML内容不能用Mustache语法，应该使用v-html指令

范例：显示HTML内容

```
<div id="app">
  <!-- 特性、属性值绑定使用v-bind指令 -->
  <h2 v-html="rawHtml"></h2>
</div>
```

1.1.3 计算属性和侦听器

计算属性

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在**模板中放入太多的逻辑会让模板过重且难以维护**。例如，想要把`title`反转一下：

```
<h2>{{ title.split('').reverse().join('') }}</h2>
```

所以，对于任何包含响应式数据的复杂逻辑，你都应该使用**计算属性**。

```
computed: {
  reverseTitle() {
    return this.title.split('').reverse().join('')
  }
}
```

侦听器

虽然计算属性在大多数情况下更合适，但有时也需要一个自定义的侦听器。这就是为什么 Vue 通过 `watch` 选项提供了一个更通用的方法，来响应数据的变化。**当需要在数据变化时执行异步或开销较大的操作时**，这个方式是最有用的。

```
<div id="watch-example">
  <p>
    Ask a yes/no question:
    <input v-model="question" />
  </p>
  <p>{{ answer }}</p>
</div>

<!-- 因为 AJAX 库和通用工具的生态已经相当丰富，Vue 核心代码没有重复 -->
<!-- 提供这些功能以保持精简。这也可以让你自由选择自己更熟悉的工具。 -->
<script src="https://cdn.jsdelivr.net/npm/axios@0.12.0/dist/axios.min.js">
</script>
<script>
  const watchExampleVM = Vue.createApp({
    data() {
      return {
        question: '',
        answer: 'Questions usually contain a question mark. ;-)'
      }
    },
    watch: {
      // whenever question changes, this function will run
      question(newQuestion, oldQuestion) {
        if (newQuestion.indexOf('?') > -1) {
          this.getAnswer()
        }
      }
    },
    methods: {
      getAnswer() {
        this.answer = 'Thinking...'
        axios
          .get('https://yesno.wtf/api')
          .then(response => {
          this.answer = response.data.answer
        })
          .catch(error => {
          this.answer = 'Error! Could not reach the API. ' + error
        })
      }
    }
  }).mount('#watch-example')
</script>
```

1.1.4 动态样式绑定

class与style绑定

操作元素的 class 列表和内联样式是数据绑定的一个常见需求。因为它们都是属性，所以我们可以用 `v-bind` 处理它们：只需要通过表达式计算出字符串结果即可。不过，字符串拼接麻烦且易错。因此，在将 `v-bind` 用于 `class` 和 `style` 时，Vue.js 做了专门的增强。**表达式结果的类型除了字符串之外，还可以是对象或数组。**

```
<!--动态地切换class-->
<div :class="{ active: isActive }"></div>

<!--传入更多字段来动态切换多个class-->
<div :class="{ active: isActive, 'text-danger': hasError }"></div>
```

```
data() {
  return {
    isActive: true,
    hasError: false
  }
}
```

绑定内联样式

`:style` 的对象语法十分直观——看着非常像 CSS，但其实是一个 JavaScript 对象。CSS property 名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用引号括起来) 来命名：

```
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

```
data() {
  return {
    activeColor: 'red',
    fontSize: 30
  }
}
```

1.1.5 条件和列表渲染

列表渲染

我们可以用 `v-for` 指令基于一个数组来渲染一个列表。`v-for` 指令需要使用 `item in items` 形式的特殊语法，其中 `items` 是源数据数组，而 `item` 则是被迭代的数组元素的**别名**。

范例：课程列表

```
<div id="app">
  <!-- 条件渲染 -->
  <p v-if="courses.length == 0">没有任何课程信息</p>
  <!-- 列表渲染 -->
  <ul>
    <li v-for="c in courses">{{c}}</li>
  </ul>
</div>

<script src="vue.js"></script>
<script>
  const app = Vue.createAppVue({
    el: '#app',
    data: {
      courses: ['web全栈', 'web高级']
    }
  })
</script>
```

条件渲染

`v-if` 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 truthy 值的时候被渲染。

范例：没有课程时的内容显示

```
<!-- 条件渲染 -->
<p v-if="courses.length == 0">没有任何课程信息</p>
<!-- v-else配合v-if使用 -->
<ul v-else>...</ul>
```

范例：结合前面动态样式绑定做点选样式

```
<ul>
  <!-- class绑定 -->
  <li v-for="c in courses"
```

```
      :class="{active: (selectedCourse === c)}"
      @click="selectedCourse = c">{{c}}</li>
  <!-- style绑定 -->
  <!-- <li v-for="c in courses"
      :style="{backgroundColor: (selectedCourse === c)?'#ddd':'transparent'}"
      @click="selectedCourse = c">{{c}}</li> -->
</ul>

<script>
  const app = Vue.createAppVue({
    data: {
      // 保存选中项
      selectedCourse: '',
    },
  })
</script>
<style>
.active {
  background-color: #ddd;
}
</style>
```

1.1.6 事件处理

可以用 `v-on` 指令监听 DOM 事件，并在触发时运行一些 JavaScript 代码。

范例：新增课程

```
<!-- 事件处理 -->
<button v-on:click="addCourse">新增课程</button>

<script>
  const app = Vue.createApp({
    methods: {
      addCourse() {
        this.courses.push(this.course);
      }
    },
  })
</script>
```

1.1.7 表单输入绑定

你可以用 `v-model` 指令在表单 `<input>`、`<textarea>` 及 `<select>` 元素上创建双向数据绑定。**它会根据控件类型自动选取正确的方法来更新元素。** `v-model` 本质上是**语法糖**。它将转换为**输入事件**以更新数据，并对一些极端场景进行一些特殊处理。

基本用法

text/textarea

```
<input v-model="message" placeholder="edit me" />
<p>Message is: {{ message }}</p>
```

checkbox/radio

```
<!--单个-->
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>

<!--多个-->
<div>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames" />
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames" />
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames" />
  <label for="mike">Mike</label>
  <br />
  <span>{{ checkedNames }}</span>
</div>
```

select

```
<div>
  <select v-model="selected">
    <option disabled value="">Please select one</option>
    <option>A</option>
    <option>B</option>
    <option>C</option>
  </select>
  <span>Selected: {{ selected }}</span>
</div>
```

范例：新增课程

```
<!-- 表单输入绑定 -->
<input v-model="course" type="text" v-on:keydown.enter="addCourse"/>
```

1.1.8 生命周期钩子

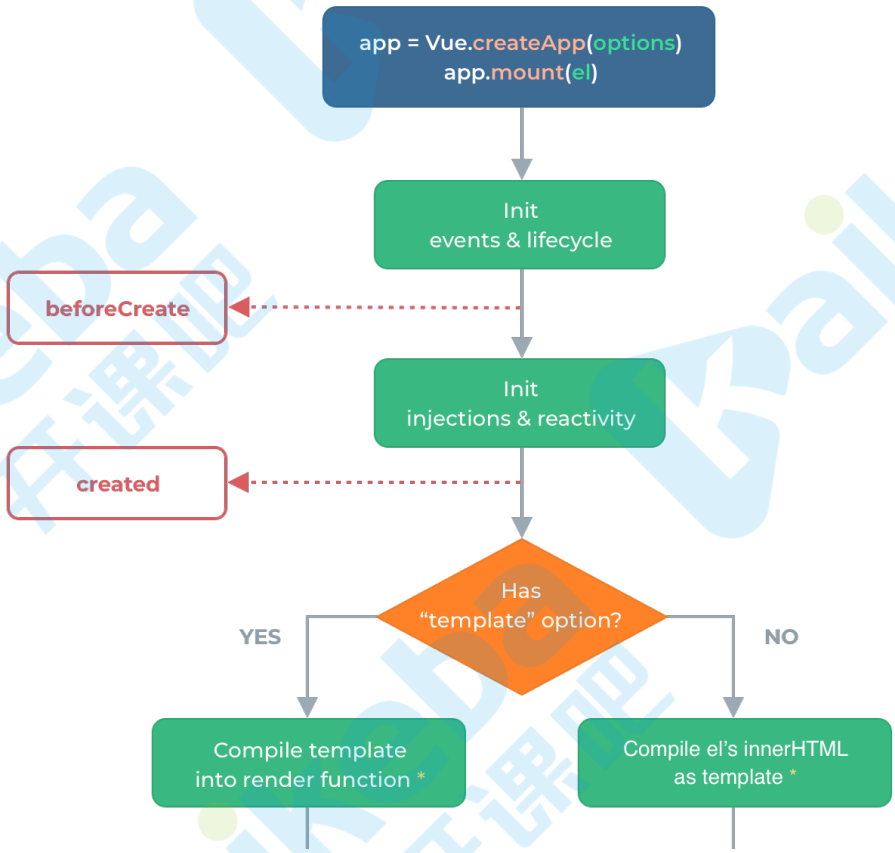
每个实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做**生命周期钩子**的函数，这给了用户在不同阶段添加自己的代码的机会。

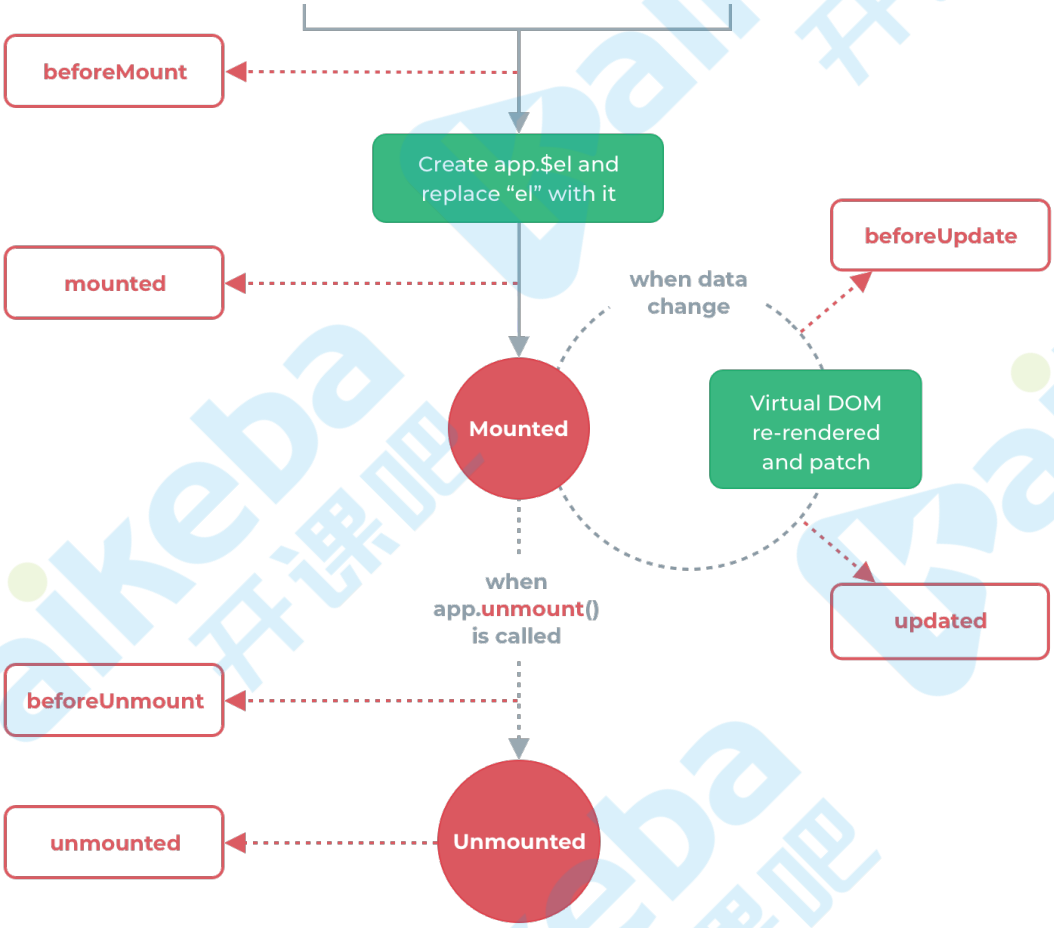
```
created() {
  setTimeout(() => {
    this.courses = ["web全栈架构师", "web高级工程师"];
  }, 1000);
},
```

生命周期图示

基本可划分为三个阶段：

- 初始化阶段：beforeCreate、created、beforeMount、mounted
- 更新阶段：beforeUpdate、updated
- 销毁阶段：beforeUnmount、unmounted





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

对比

V3	V2
`beforeCreate`	`beforeCreate`
`created`	`created`
`beforeMount`	`beforeMount`
`mounted`	`mounted`
`beforeUpdate`	`beforeUpdate`
`updated`	`updated`
`beforeUnmount`	`beforeDestroy`
`unmounted`	`destroyed`
`errorCaptured`	`errorCaptured`
`renderTracked`	-
`renderTriggered`	-