目录

[第1章 Vue.js介绍 4](#_Toc62326246)

[1.1什么是Vue.js 4](#_Toc62326247)

[1.2 Vue与其它框架 4](#_Toc62326248)

[1.2.1 Vue与React 4](#_Toc62326249)

[1.2.2 Vue与AngularJS 5](#_Toc62326250)

[1.2.3 Vue与Angular（Angular2） 5](#_Toc62326251)

[1.2.4 Vue与其他框架 6](#_Toc62326252)

[1.3 Vue 3的简述 6](#_Toc62326253)

[1.3.1 Vue 3的新特性 6](#_Toc62326254)

[1.3.2从Vue 2开始的重大改变 7](#_Toc62326255)

[1.3.3库与工具的支持 7](#_Toc62326256)

[1.4 本章小结 7](#_Toc62326257)

[1.5 练习题 8](#_Toc62326258)

[第2章 第一个Vue应用 9](#_Toc62326259)

[2.1准备Vue 3 9](#_Toc62326260)

[2.1.1引入Vue.js 9](#_Toc62326261)

[2.1.2安装Vue Devtools 10](#_Toc62326262)

[2.2 Vue实例和数据绑定 12](#_Toc62326263)

[2.2.1构建Vue 3项目 12](#_Toc62326264)

[2.2.2项目实例：Hello Vue3 13](#_Toc62326265)

[2.2.3 setup与生命周期 14](#_Toc62326266)

[2.2.4数据 16](#_Toc62326267)

[2.2.5数据实例：显示响应式对象 17](#_Toc62326268)

[2.2.6方法 18](#_Toc62326269)

[2.2.7方法实例：修改响应式对象的值 18](#_Toc62326270)

[2.3 本章小结 19](#_Toc62326271)

[2.4 练习题 20](#_Toc62326272)

[第3章 Vue的内置指令与语法 21](#_Toc62326273)

[3.1插值绑定 21](#_Toc62326274)

[3.1.1文本插值与表达式 21](#_Toc62326275)

[3.1.2过滤器 23](#_Toc62326276)

[3.1.3 HTML插值 23](#_Toc62326277)

[3.2计算属性 24](#_Toc62326278)

[3.2.1计算属性的概念 24](#_Toc62326279)

[3.2.2计算属性的setter 26](#_Toc62326280)

[3.2.3侦听属性watch 27](#_Toc62326281)

[3.3 v-bind属性绑定 29](#_Toc62326282)

[3.3.1 v-bind指令 29](#_Toc62326283)

[3.3.2绑定class、style与prop 30](#_Toc62326284)

[3.4 v-model双向绑定 31](#_Toc62326285)

[3.4.1 v-model指令 31](#_Toc62326286)

[3.4.2 v-model与修饰符 32](#_Toc62326287)

[3.4.3双向绑定实例：制作问卷 33](#_Toc62326288)

[3.5 v-if/v-show条件渲染 37](#_Toc62326289)

[3.5.1 v-if，v-else与v-else-if 37](#_Toc62326290)

[3.5.2 v-show 39](#_Toc62326291)

[3.5.3 v-if对比v-show 39](#_Toc62326292)

[3.5.4条件渲染实例：按钮权限控制 39](#_Toc62326293)

[3.6 v-for列表渲染 42](#_Toc62326294)

[3.6.1 v-for指令 42](#_Toc62326295)

[3.6.2在v-for里使用对象 43](#_Toc62326296)

[3.6.3列表的更新 44](#_Toc62326297)

[3.6.4列表渲染的key 45](#_Toc62326298)

[3.6.5过滤与排序 46](#_Toc62326299)

[3.6.6 v-for与v-if共用 47](#_Toc62326300)

[3.6.7列表渲染实例：帖子列表 48](#_Toc62326301)

[3.7 v-on事件绑定 50](#_Toc62326302)

[3.7.1 v-on指令 50](#_Toc62326303)

[3.7.2事件修饰符 51](#_Toc62326304)

[3.8指令在Vue 3.x中的变化 52](#_Toc62326305)

[3.8.1 v-if与v-for的key 52](#_Toc62326306)

[3.8.2 v-if与v-for的优先级 53](#_Toc62326307)

[3.8.3 v-bind合并行为 53](#_Toc62326308)

[3.8.4 v-for中的Ref数组 53](#_Toc62326309)

[3.8.5 v-model 54](#_Toc62326310)

[3.9本章小结 55](#_Toc62326311)

[3.10练习题 56](#_Toc62326312)

[第4章 Class与Style绑定 57](#_Toc62326313)

[4.1绑定HTML Class 57](#_Toc62326314)

[4.1.1对象语法 57](#_Toc62326315)

[4.1.2数组语法 59](#_Toc62326316)

[4.2绑定内联样式 60](#_Toc62326317)

[4.2.1对象语法 60](#_Toc62326318)

[4.2.2数组语法 62](#_Toc62326319)

[4.3本章小结 62](#_Toc62326320)

[4.4练习题 62](#_Toc62326321)

[第5章 过滤器 63](#_Toc62326322)

[5.1内置过滤器 63](#_Toc62326323)

[5.1.1字母操作 63](#_Toc62326324)

[5.1.2 json过滤器 64](#_Toc62326325)

[5.1.3限制过滤器 64](#_Toc62326326)

[5.1.4 currency过滤器 66](#_Toc62326327)

[5.1.5 debounce过滤器 66](#_Toc62326328)

[5.2本章小结 67](#_Toc62326329)

[5.3练习题 67](#_Toc62326330)

# 第1章 Vue.js介绍

本章将从Vue的概念出发，介绍Vue的特色以及与其他前端框架的不同。对于已经了解Vue 2的读者，本章也会介绍Vue 3的一些新特性。

## 1.1什么是Vue.js

Vue (读音类似view)是一种模型-视图-视图模型（Model-View-ViewModel，MVVM），View和Model是独立的，ViewModel是View 和Model交互的桥梁。当View的某个部分需要更新时，Vue会自动选择恰当的方法和时机进行更新。

Vue的开发者将其称为渐进式框架。与其他重量级框架不同的是，Vue的核心库只关注视图层，而且被设计为可以自底向上逐层应用。得益于Vue简单的API，以及可以阶段性使用的特点，使得Vue十分容易上手。同时，与第三方库或既有项目的整合也十分便捷。

## 1.2 Vue与其它框架

在 MVVM 框架中，Vue的表现十分优秀。原因在于Vue吸取了其他框架，例如React、Angular等的优势，在这一节将会介绍Vue与其他框架的对比。

### 1.2.1 Vue与React

React的特点是使用JavaScript一种语言就可以写前后端，但是因此HTML代码需要写在JavaScript文件中（现在的潮流也越来越多地将CSS也纳入到 JavaScript中来处理），同时前后端代码需要写在一起。这种编程方式也叫“多语言混合式编程” ，会导致代码难以理解、调试。

React与Vue有许多相似之处。首先它们都将注意力集中保持在核心库，使用虚拟DOM以降低页面开销，并且提供了响应式和组件化的视图组件。

在React应用中，当某个组件的状态发生变化时，它会以该组件为根，重新渲染整个组件子树。对于不必要子组件的重渲染则需要手动控制。而在Vue 应用中，组件的依赖是在渲染过程中自动追踪的，所以系统能精确知晓哪个组件确实需要被重渲染。Vue的这个特点使得开发者不再需要考虑此类优化。

在React中，所有的组件的渲染功能都依靠 JSX。但在Vue中也提供了渲染函数，也支持 JSX，同时HTML都是合法的Vue模板。对于CSS而言，在 React中是通过CSS-in-JS实现CSS作用域的，在Vue中则通过单文件组件里类似style的标签。

在应用开发方面，React Native支持使用相同的组件模型编写有本地渲染能力的iOS和 Android软件，并且能同时跨多平台开发。而Vue方面，Weex支持的原生应用的组件开发有待进一步成熟。

### 1.2.2 Vue与AngularJS

Vue的灵感来源于AngularJS，但是Vue解决了AngularJS中存在的许多问题。相比于AngularJS，Vue在速度与体积上都有很多优势。

Vue 与AngularJS相比最大的区别在于没有脏检测机制。在AngularJS中存在多个watcher，当watcher越来越多时，检测耗时会越来越长。因为作用域内的每一次变化，所有 watcher都要重新计算。并且，如果一些 watcher触发另一个更新，就会引发所有watcher重新计算，从而进入一种无限循环的脏检测。这种脏检测机制性能低下，而且有时没有简单的办法来优化有大量 watcher的作用域，并不适合大型Web应用。而Vue使用基于依赖追踪的观察系统并且异步队列更新，全局只设置一个watcher，用这一个watcher来记录和更新一组关联对象的值，因此所有的数据变化都是独立触发，除非它们之间有明确的依赖关系。

在数据流方面，Vue在不同组件间强制使用单向数据流，使得数据流更加清晰易懂。而AngularJS在单向数据流的视图渲染、事件绑定之外，还参与了View对Model层的数据更新，即双向数据绑定。

### 1.2.3 Vue与Angular（Angular2）

Angular具有优秀的组件系统，在AngularJS中的许多实现已经完全重写，API也完全改变了。重写过的Angular有着很快的速度，在性能测试数据上与Vue十分接近。

在体积方面，最近的 Angular版本中在使用了AOT和tree-shaking技术后使得最终的代码体积减小了许多。但Vue 项目还是要小得多。

在学习曲线上，学习Vue，只需要HTML和 JavaScript基础。而Angular 的学习曲线则比较陡峭，由于Angular的设计目标就是只针对大型的复杂应用，它的API更多，概念也更多。因此初学者上手时会有比较多的困难。

### 1.2.4 Vue与其他框架

Polymer也是Vue的一个灵感来源，两者具有相似的开发风格。最大的不同之处在于，Polymer需要重量级的polyfills来帮助工作，一方面导致了性能的下降，另一方面浏览器本身并不支持这些功能。相比而言，Vue不依赖polyfills，并且支持到了IE9。

Ember是一个全能框架。它提供了大量的约定，同时也就有学习曲线高、不灵活的问题。在性能上，Vue比 Ember好很多，Vue能够自动批量更新，而 Ember在性能敏感的场景时需要手动管理。Vue在普通JavaScript对象上建立响应，Ember中需要放在Ember对象内，并且手动为计算属性声明依赖。

## 1.3 Vue 3的简述

本章将会列举Vue 3的一些新特性以及与Vue 2不兼容的一些变更，以便有 Vue 2 经验的读者在开发过程中注意到这些改变。Vue 3所做改变的细节将会在每个对应章节内进行详述。

对比Vue 2，Vue 3通过tree-shaking减小约41%捆绑包体积、初始渲染速度提升约55%、更新速度提升约133%、内存使用率降低约54%等。

### 1.3.1 Vue 3的新特性

在基于对象的2.x API基本没有变化的情况下，Vue 3引入了Composition API。Composition API在代码组织模式上更灵活，类型推导也更稳定，使得Vue在大型应用中更有竞争力。目前已经有适用于Vue 2和Vue 3的Composition API实用程序库。

在Vue 3中提供了Teleport，允许开发者控制在DOM中哪个父节点下呈现 HTML，实现时将不必求助于全局状态或将其拆分为两个组件。

在Vue 3中，组件现在可以有多个根节点，也就是片段。前提是开发者明确定义属性应该分布在哪里。

Vue3中的<style scoped>现在可以包含全局规则或只针对插槽内容的规则

目前Vue 3有一些实验性特性，包括：单文件组件组合式 API 语法糖 (<script setup>) 、单文件组件状态驱动的 CSS 变量 (<style vars>) 。

### 1.3.2从Vue 2开始的重大改变

Global API方面，全局Vue API已更改为使用应用程序实例，全局和内部API已经被重构为可tree-shakable，通过tree-shakable可以减少多达一半的 runtime大小。

模板指令里，v-model的用法，<template v-for>和非v-for节点上key的用法，v-if和v-for的优先级都已经更改。v-bind="object"排序敏感，并且v-for中的ref不再注册ref数组。

Vue 3还移除了Vue 2中的一些API：keyCode支持作为v-on的修饰符、$on，$off和 $once实例方法、过滤、内联模板attribute、$destroy实例方法。

除了以上这些，Vue 3在组件、渲染函数等部分也做了一些改变，将会在对应的章节进行说明。

### 1.3.3库与工具的支持

Vue所有的官方库和工具都支持 Vue 3，并且计划在 2020 年底前稳定所有项目。

vue-cli现在提供了内置选项，可以在创建新项目时选择Vue 3预设。升级vue-cli并运行vue create来创建Vue 3项目。

Vue Router 4.0提供了Vue 3支持。新版本的Devtools目前只支持 Vue 3。但经过新的UI设计和内部结构重构，将会在未来的更新中支持多个Vue版本。

VSCode和Vetur都提供了对Vue 3的全面支持。

## 1.4 本章小结

本章从Vue的快速、轻量化特点作为切入，介绍了它作为渐进式的MVVM框架的特性。通过与其他框架，例如React、Angular的对比，读者可以更直观地选择适合实际开发的框架。而Vue作为渐进式框架，其平稳的学习曲线对于前端框架的新手而言是十分友好的，只需要HTML和 JavaScript基础知识便可以轻松上手。

新的Vue 3则在基于对象的Vue 2.x的基础上加入了Composition API，即组合式API。组合式API在代码组织模式上的灵活性，帮助Vue 3在大型应用中更有竞争力。其他的改变，例如组件、渲染、指令等，则会在本书后续的章节中进行详述。Vue所有的官方库和工具都支持 Vue 3。

在下一章中，将会介绍Vue 3的安装，并构建Vue 3的第一个项目。

## 1.5 练习题

* **问答题：**

1. 什么是MVVN框架？
2. 在学习Vue之前需要哪些基础知识？
3. Vue的灵感来源于哪个框架？Vue中组件的重渲染需要手动控制吗？
4. Vue 3对比Vue 2.x在那些性能方面有所提升？

# 第2章 第一个Vue应用

本章将会对Vue 3和工具的安装进行介绍，并在此基础上完成第一个Vue 3应用。Vue设计的初衷包括可以被渐进式地采用，这意味着它可以根据需求以多种方式集成到一个项目中。因此在最开始，我们可以很简单地引入Vue。

## 2.1准备Vue 3

### 2.1.1引入Vue.js

目前Vue的最新版本为3.0.2，将 Vue 3添加到项目中有三种主要方式。

第一种方式是在页面上以CDN package的形式导入，添加的位置如例**2-1**所示。

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="utf-8" />  <title>Vue</title>  <script src="https://unpkg.com/vue@next"></script>  </head>  <body>  </body>  </html> |

**例2-1 以CDN形式引入**

对于学习或制作原型，可以使用一下语句：

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

对于生产环境，则推荐链接到一个明确的版本号和构建文件，例如：

|  |
| --- |
| <script src="https://unpkg.com/vue@3.0.2"></script> |

第二种方式是通过npm安装Vue 3，推荐在构建大型应用时使用。在Node的官网下载Node，同时会得到npm工具。npm 能很好地和Webpack、Browserify等模块打包器配合使用。在命令行输入以下指令安装Vue 3最新稳定版：

|  |
| --- |
| npm install vue@next |

由于npm的仓库源布置在国外，传输速度较慢，作为替代可以使用淘宝镜像源的cnpm：

|  |
| --- |
| npm install -g cnpm --registry=https://registry.npm.taobao.org |

之后，就可以用 cnpm 指令替代 npm 指令进行安装。

第三种方式是使用命令行工具Vue CLI。Vue CLI是一个由官方提供的基于Vue.js进行快速开发的完整系统，为单页面应用(SPA)快速搭建繁杂的脚手架。它确保了各种构建工具能够基于智能的默认配置平稳衔接，这样开发者可以专注在开发应用上。Vue CLI可以快速运行并带有热重载、保存时lint校验，以及生产环境可用的构建版本。

对于Vue 3，应该使用npm上可用的 Vue CLI v4.5 作为 @vue/cli@next。可以通过全局重新安装最新版本的 @vue/cli 进行升级。使用下列任意一条指令进行安装。

|  |
| --- |
| yarn global add @vue/cli@next  npm install -g @vue/cli@next |

然后在Vue项目中运行：

|  |
| --- |
| vue upgrade --next |

### 2.1.2安装Vue Devtools

在开始开发Vue之前，推荐在浏览器中安装Vue Devtools，它可以方便开发者在一个更友好的界面中调试 Vue 应用。Vue Devtools提供了一个查看Vue组件和全局状态管理器Vuex中数据的界面。

对于Google Chrome和Firefox，都可以在浏览器的应用商店中搜索Vue Devtools直接安装。

访问商店有困难的情况下也可以根据以下步骤进行手动安装：

1. 在git bash中输入以下指令，克隆Vue Devtools仓库

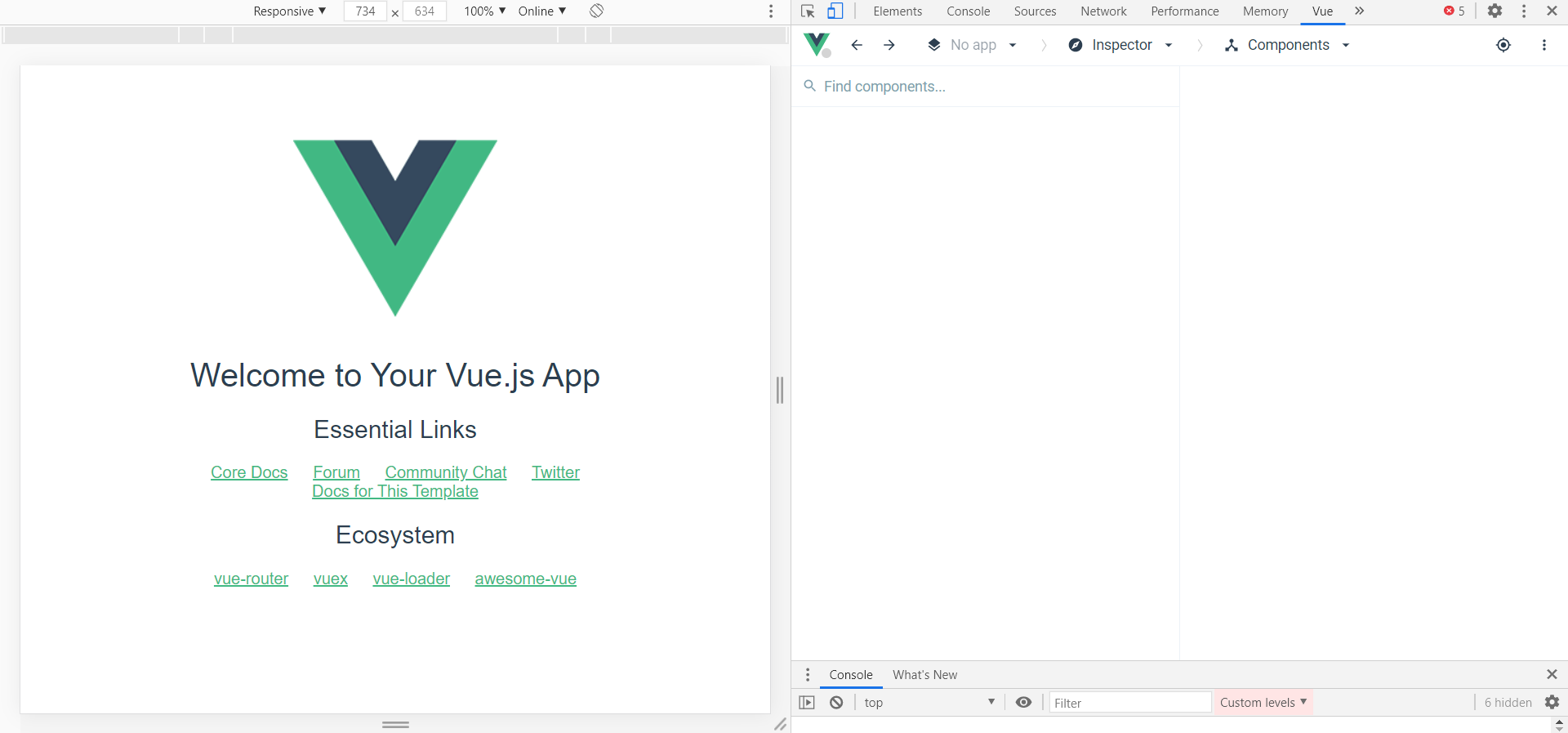
|  |
| --- |
| git clone https://github.com/vuejs/vue-devtools.git |

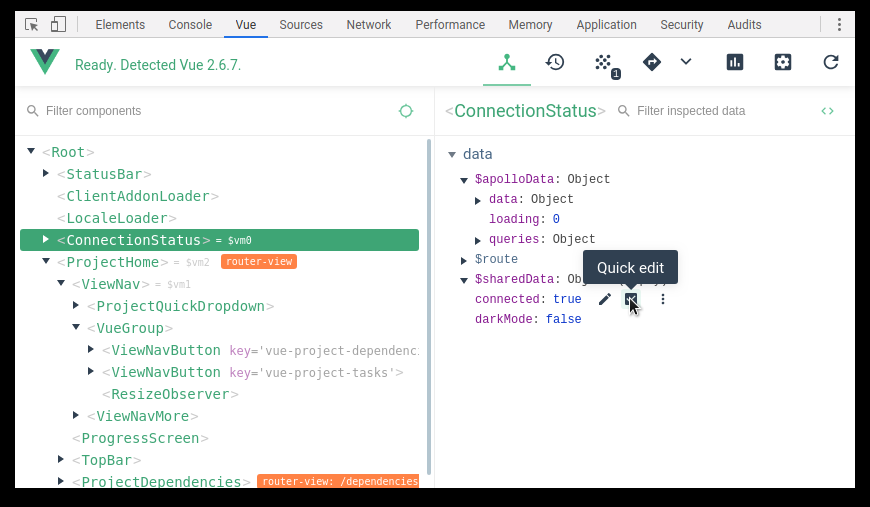
1. 进入新创建的vue-devtools文件夹
2. 在命令行中分别运行以下指令

|  |
| --- |
| yarn install  yarn run build |

1. 打开Chrome浏览器
2. 选择【菜单】→【更多工具】→【扩展程序】命令
3. 单击右上角的【开发者模式】按钮
4. 单击左侧的【加载已解压的扩展程序】按钮
5. 选择文件夹：vue-devtools/packages/shell-chrome/
6. 等待安装完成

安装完成后可以在网页点击F12呼出调试工具，查看Vue Devtools，其界面如图**2-1**所示。





**图2-1 Vue Devtools界面**

## 2.2 Vue实例和数据绑定

### 2.2.1构建Vue 3项目

Vue 3支持使用Vite构建一个项目。Vite是一个web开发构建工具，可以快速提供代码。通过在命令行中运行以下命令，可以使用Vite快速构建Vue项目。

使用npm：

|  |
| --- |
| npm init vite-app <project-name>  cd <project-name>  npm install  npm run dev |

或者使用yarn：

|  |
| --- |
| yarn create vite-app <project-name>  cd <project-name>  yarn  yarn dev |

### 2.2.2项目实例：Hello Vue3

在项目创建完成后，在***App.vue***与***main.js***中编写第一个Vue 3应用，代码分别如例**2-2**与例**2-3**所示。

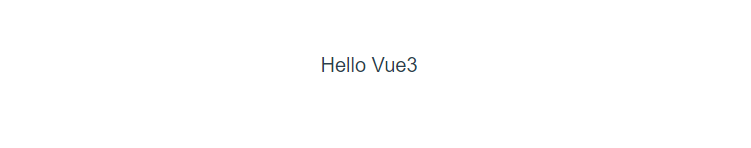
|  |
| --- |
| <template>  <div id="app">  <p>{{ hello }}</p>  </div>  </template>  <script>  import { ref } from "vue";  export default {  setup() {  const hello = ref("Hello Vue3");  return {  hello,  };  },  };  </script> |

**例2-2 App.vue中的代码**

|  |
| --- |
| import { createApp } from 'vue'  import App from './App.vue'  import './index.css'  createApp(App).mount('#app') |

**例2-3 main.js中的代码**

使用npm run dev指令正确运行后将会在网页上看到Hello Vue3的字样，如图**2-2**所**示**。至此，第一个Vue 3应用创建成功。



**图2-2 第一个Hello Vue3**

对于这个实例，在main.js中使用了Vue 3新增的createApp函数创建了一个新的应用实例，并挂载到名为app的DOM元素上。

在App.vue中定义了*hello*，并通过setup函数将其初始值设为'Hello Vue3'。其中，setup函数是组合API的入口点，而ref函数则帮助Vue监听*hello*，当它发生变化，Vue会自动更新视图。Vue会声明式地将数据渲染进DOM的系统，所有东西都是响应式的，也就是说当数据发生变化时，会同步更新其数据链和作用域中所有的相关状态。

### 2.2.3 setup与生命周期

setup函数是Vue 3中新增的一个组件选项，为使用Vue 3的组合式API 新特性提供了统一的入口。会在原有的beforeCreate函数之后、created函数之前执行。 setup的参数有两个：props和context。

在props中定义外界传递过来的参数名称和类型，并在setup函数的第一个形参接收。

setup函数的第二个形参context是一个上下文对象，包含了attrs、slots等属性。由于在setup函数中无法访问到this，因此使用形如context.attrs的方式访问这些属性。对于setup函数与props、context的使用，如例**2-4**所示。

|  |
| --- |
| <script>  export default {  setup(props, context) {  console.log(props.arg1)  console.log(context)  },  props: {  arg1: string,  arg2: int,  },  // context中的内容：  // context :{  // attrs: Data  // slots: Slots  // emit: (event: string, ...args: unknown[]) => void  // 以及parent, refs, root...  // }  };  </script> |

**例2-4 setup函数与props、context**

在Vue 3中，生命周期钩子注册函数也有所变化。在setup函数中可以使用直接导入的onX函数注册生命周期钩子。这些生命周期钩子注册函数只能setup期间同步使用，因为它们依赖于内部全局状态来定位当前活动实例，也就是正在调用其setup的组件实例。

在Vue 2.x的生命周期钩子注册函数中，beforeCreate与created函数删除，直接使用setup，其他生命周期钩子注册函数可以加上前缀on后在setup函数中使用，例如beforeMount变为onBeforeMount、mounted变为onMounted。如例2-5所示。

|  |
| --- |
| import { onMounted, onUpdated, onUnmounted } from ‘vue’;  export default {  setup(props, context) {  onMounted(() => {  console.log('mounted!')  })  onUpdated(() => {  console.log('updated!')  })  onUnmounted(() => {  console.log('unmounted!')  })  }  }; |

**例2-5 生命周期钩子注册函数**

### 2.2.4数据

在Vue 3中，通过ref函数与reactive函数在setup内创建响应式的对象。

ref函数接受一个内部值并返回一个响应式且可变的ref对象，这个对象上只包含一个.value属性。想要访问ref对象的内部值需要使用property .value。

reactive函数会返回原始对象的响应式副本，它影响所有嵌套property。等价于vue 2.x中的Vue.observable函数。

toRefs函数可以将reactive创建出来的响应式对象，转换为普通的对象，这个对象上的每个属性节点，都是ref类型的响应式数据。结果对象的每个property都是指向原始对象相应property的ref。

当把ref对象，挂载到reactive上时，会自动把响应式数据对象展开为原始的值，不需通过.value就可以直接被访问。

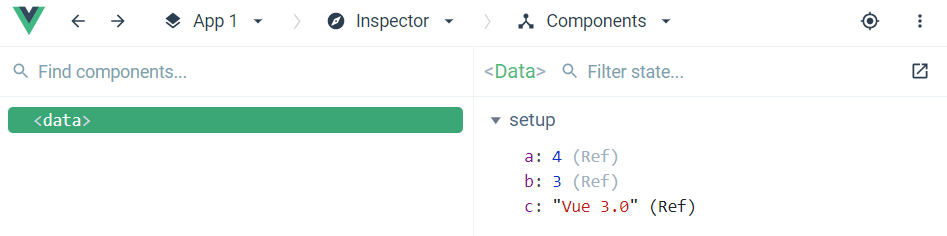
### 2.2.5数据实例：显示响应式对象

从vue中导入ref，reactive，toRefs函数后，在setup函数中创建ref对象*a*，以及reactive对象*state*，其中*state*包含*b*与*c*属性。在return中，将*state*转换为ref对象，使其具有响应式的属性。完整代码如例**2-6**所示。

|  |
| --- |
| <template>  <div id="app">  <p>{{ a }}</p>  <p>{{ b }}</p>  <p>{{ c }}</p>  </div>  </template>  <script>  import { ref, reactive, toRefs } from "vue";  export default {  setup() {  const a = ref(2); //一个响应式且可变的ref对象,此时a的值为2  //创建对象的响应式副本，并非直接响应式  const state = reactive({  b: 3,  c: "Vue 3.0",  });  a.value = 4;  return {  a,  ...toRefs(state), //toRef()函数将data转化为响应式  };  },  };  </script> |

**例2-6 main.js中的代码**

使用Vue Devtools可以查看*a*，*b*，*c*的值以及它们的类型，如图**2-3**所示。



**图2-3 使用Vue Devtools查看样例**

### 2.2.6方法

在Vue 2.x中，可以用methods选项向组件实例添加方法，在Vue 3中仍然支持。而Vue 3中，还可以将原来methods中的方法写在setup函数中，与Vue 2.x一样使用。需要注意的是，想要访问到方法，同样需要在setup的return中写入方法名。

### 2.2.7方法实例：修改响应式对象的值

在之前的数据实例中，再添加一个按钮，并赋予一个@click事件，在点击之后修改*a*，*b*，*c*的值。template中添加的代码如例**2-7**所示。

|  |
| --- |
| <button @click="changeBtn">改变</button> |

**例2-7 在template中添加按钮**

之后，在setup方法中编写修改值得方法，其中修改*a*的值需要通过*a*.value。编写完成后，在return中返回方法名称。setup中加入的代码如例**2-8**所示。

|  |
| --- |
| const changeBtn = () => {  console.log("---changing------");  state.b = 5;  state.c = "changed";  a.value = 8;  };  return {  a,  ...toRefs(state), //toRef()函数将data转化为响应式  changeBtn  }; |

**例2-8 在*setup*中编写修改方法**

在网页中点击按钮，可以看到*a*，*b*，*c*的值被修改。修改前后的对比如图**2-4**所示。

**图2-4 值修改前后对比**

## 2.3 本章小结

本章介绍了Vue 3的三种引入方式，分别为：CDN Package导入、npm安装以及命令行工具Vue CLI构建项目。此外，还介绍了浏览器调试工具Vue Devtools，它允许开发者在一个更友好的界面中审查和调试Vue应用。

在本章中，使用Vite快速构建了一个Vue 3项目。Vite是一个web开发构建工具，可以快速提供代码。在项目构建的实例中，使用了Vue 3新增的createApp函数创建了一个新的应用实例。

作为Vue 3的组合式AP的统一入口，本章介绍了setup函数的使用方法、其参数props和context，以及Vue 2.x中的生命周期钩子注册函数在setup函数中的使用方法。

对于数据与方法，Vue 3则在支持Vue 2.x中data与methods写法的同时，加入了通过ref函数与reactive函数在setup内创建响应式的对象，以及在setup函数内编写方法的语法。

最后，本章还提供了项目构建、数据、方法的实例，读者可以通过这些实例认识到Vue 3在实际项目中的使用。

在下一章中，将会介绍Vue的语法和指令，以及一些语法在Vue 3相对Vue 2.x的更新。

## 2.4 练习题

* **问答题：**

1. 引入Vue的方式有哪三种？分别尝试安装一下。
2. 简要说明ref、reactive与toRefs函数的作用。
3. *A*是一个ref对象，初始值为0，如何将*A*的值修改为2？

* **动手做：**

1. 尝试新建一个Vue 3项目，并在页面上打印“GOOD JOB!”。
2. 尝试在页面上实现一个计时器，点击按钮开始/暂停计时。

# 第3章 Vue的内置指令与语法

本章将会对Vue 3中的语法和内置指令进行介绍，包括差值绑定、计算属性、条件渲染指令、列表渲染指令等，对于在Vue 3中有变更的部分会在对应的小节中进行说明。

## 3.1插值绑定

### 3.1.1文本插值与表达式

文本插值最基本的方法是使用双大括号（Mustache语法）“{{ }}”，Vue将会获取计算后的值，将大括号里的内容替换为设定值，以文本的形式将其展示出来。通过任何方法修改数据设定值，大括号的内容都会被实时替换。例**2-2**中的*hello*，例**2-5**中的*a*，*b*，*c*都是通过这种方式在页面中显示数据的。

除了直接赋值，Mustache语法也接受表达式形式的值。表达式可由JavaScript表达式和过滤器构成。表达式可以有变量、数值、运算符等，表达式的值是它的运算结果。虽然不支持条件语句，但可以通过三元式实现简单的条件判断。

例**3-1**展示了通过文本插值与表达式计算变量、表达式、条件运算符的值，在页面中的效果如图**3-1**所示。

|  |
| --- |
| <template>  <div id="app">  <p><label>变量:</label> {{ num }}</p>  <p><label>表达式:</label> {{ 5 + 10 }}</p>  <p><label>条件运算符:</label> {{ true ? 5 : 10 }}</p>  </div>  </template>  <script>  import { ref } from "vue";  export default {  setup() {  const num = ref("2");  return {  num,  };  },  };  </script> </body>  </html> |

**例3-1 通过文本插值计算变量、表达式、条件运算符的值**



**图3-1 通过文本插值显示变量、表达式、条件运算符的值**

Vue的“{{ }}”内只支持单个表达式，不支持语句和流控制。并且在表达式中，不能使用用户自定义的全局变量。“{{ }}”可以放在HTML标签内，但Vue指令和自身特性内是不可以插值的。

如果想显示“{{ }}”标签，而不进行替换，可以使用v-pre跳过这个元素和它的子元素的编译过程。此外，HTML绑定Vue实例，在页面加载时可能会闪烁。原因是Vue来不及渲染，页面显示出了Vue源代码，可以使用v-cloak指令隐藏未编译的Mustache标签直到实例准备完毕。

### 3.1.2过滤器

在Vue 2.x中支持在“{{ }}”插值的尾部添加过滤器，用管道符“ |” 表示。经常用于格式化文本，比如字母全部大写、格式化日期等。过滤的规则是可以自定义的，通过给Vue实例添加filters来设置。在Vue 3.0中，过滤器已被移除，建议使用方法或计算属性来实现。

下面的例**3-2**实现了内置过滤器、过滤器串联与过滤器传参。uppercase是Vue的一个内置过滤器，可以将字符串转换为大写。通过使用多个管道符号可以将多个过滤器进行串联。

|  |
| --- |
| {{ string | uppercase }}  {{ string | filterA | filterB }}  {{ string | filter arg1 arg2 }} |

**例3-2 内置过滤器、过滤器串联与过滤器传参**

当有多个参数时，可以通过空格将参数分开，过滤器会将string作为第一个参数，arg1、arg2分别作为第二个与第三个参数传入。参数可以是表达式，也可以使用单引号传入字符串。

包括uppercase，Vue总共内置了10种过滤器，将会在过滤器的章节进行详细介绍。

### 3.1.3 HTML插值

HTML插值可以动态渲染DOM 节点，常用于处理开发者无可预知和难以控制的DOM 结构。与文本插值不同的是，文本插值中的代码被解释为节点的文本内容，而HTML插值中的代码则被渲染为视图节点。

对于值是HTML的片段，可以使用三个大括号“{{{ }}}”来绑定。也可以在标签内使用v-html=””的形式。所接受的字符串不会进行编译等操作，Vue会把被绑定的内容解析为DOM节点，按照普通HTML处理，从而实现动态渲染视图的效果。

需要注意的是，在网站上直接动态渲染任意HTML片段，容易导致XSS攻击。因此，开发者应尽量多地使用Vue 自身的模板机制，减少对HTML插值的使用，并且只对可信内容使用HTML插值。

## 3.2计算属性

在项目开发中，往往会在模板中使用表达式或过滤器来对数据进行处理。当表达式过长或者逻辑更复杂时，模板就会变得难以维护。为了避免这种问题，Vue提供了计算属性，对逻辑进行简化。

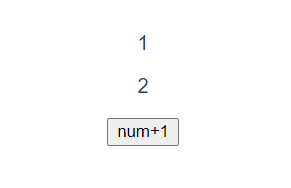
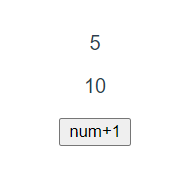
### 3.2.1计算属性的概念

计算属性会在其依赖属性的值发生变化时，对属性的值进行自动更新，同时更新相关的DOM 部分。通过从vue中导入computed来使用计算属性。

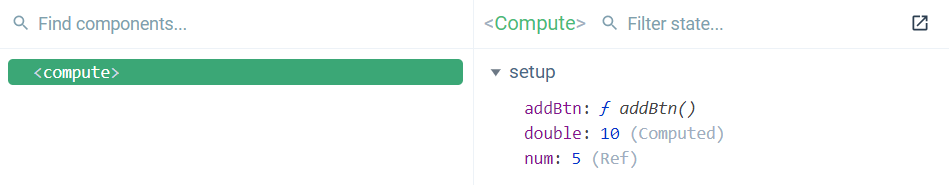
在例**3-3**中给出了计算属性的例子，*double*会始终保持为*num*的两倍，使用按钮增加*num*的值，*double*也会随之改变。显示结果如图**3-2**所示。在Vue Devtools中可以查看计算属性的值，如图**3-3**所示。

|  |
| --- |
| <template>  <div id="app">  <p>{{ num }}</p>  <p>{{ double }}</p>  <button @click="addBtn">num+1</button>  </div>  </template>  <script>  import { computed, toRefs, reactive, ref } from "vue";  export default {  setup() {  const data = reactive({  num: 1,  });  const double = computed(() => {  return data.num \* 2;  });  const addBtn = () => {  data.num++;  };  return {  ...toRefs(data),  double,  addBtn,  };  },  };  </script> |

**例3-3 计算属性**

**图3-2 计算属性的显示**



**图3-3 在Vue Devtools中查看属性**

除了以上这种写法，也可以在将computed写在*data*中，同样可以达到相同的效果。

|  |
| --- |
| const data = reactive({  num: 1,  doubleNum: computed(() => data.num \* 2),  }); |

**例3-4 计算属性的另一种写法**

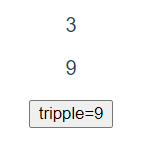
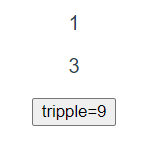
### 3.2.2计算属性的setter

在例3-3中，如果尝试修改*double*的值，会发现Vue会给出一个warning，原因是所创建的*double*是只读的计算属性。通过传入一个包含get和set函数的对象，可以得到一个可读可写的计算属性。

在例3-5中，创建了一个可读可写的计算属性*tripple*，它的值会保持为*num*的三倍。点击按钮将*tripple*的值修改为9，此时*num*的值也会被一同修改为3。修改前后的结果如图**3-4**所示。

|  |
| --- |
| const tripple = computed({  get: () => data.num \* 3,  set: (val) => {  data.num = val / 3;  },  }); |

**例3-5 可读可写的计算属性**



**图3-4 修改tripple的结果**

在computed中，get是取值函数，set是赋值函数。为计算属性赋值时，会触发set函数，触发set函数后，*num*的值会被更新。

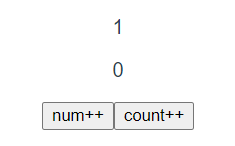
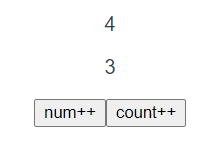
### 3.2.3侦听属性watch

watch函数用来监视指定数据项的变化，从而触发用户自定的操作。watch API完全等同于选项式API this.$watch。watch需要指定侦听的数据源，并在回调函数中执行副作用。默认情况下，回调仅在侦听的数据源发生改变时调用。通过从vue中导入watch来使用侦听属性。

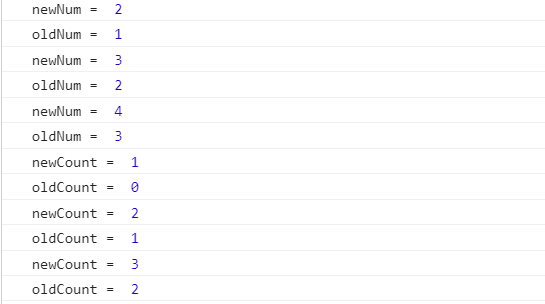
例**3-6**给出了使用watch侦听reactive类型的*num*与ref类型的*count*的例子。watch 可以获取到新值与更新前的值，当*num*与*count*改变时，会执行回调函数，在控制台打印更新前后的值。页面布局与控制台输出结果分别如图**3-5**与图**3-6**所示。

|  |
| --- |
| //侦听reactive类型的数据源  const data = reactive({  num: 1,  });  watch(  () => data.num,  (newNum, oldNum) => {  console.log("newNum = ", newNum);  console.log("oldNum = ", oldNum);  }  );  //侦听ref对象  const count = ref(0);  watch(count, (newCount, oldCount) => {  console.log("newCount = ", newCount);  console.log("oldCount = ", oldCount);  }); |

**例3-6 侦听num与count并打印变化**

**图3-5 通过按钮改变num与count**



**图3-6 控制台输出结果**

侦听器还可以使用数组同时侦听多个源，如例**3-7**所示。

|  |
| --- |
| //侦听多个reactive类型  watch(  [() => data.num1, () => data.num2],  ([newNum1, newNum2], [prevNum1, prevNum2]) => {  //Do something  }  );  //侦听多个ref类型  watch(  [count1, count2],  ([newCount1, newCount2], [prevCount1, prevCount2]) => {  //Do something  }  ); |

**例3-7 侦听多个数据源**

## 3.3 v-bind属性绑定

除了文本之外，DOM节点还有其他一些重要的属性，这些属性基本都可以用指令v-bind进行绑定。

### 3.3.1 v-bind指令

v-bind指令主要用于动态绑定DOM元素属性，可以将一个或多个attribute，或一个组件prop动态地绑定到表达式。如例**3-8**所示。

|  |
| --- |
| <!-- 绑定 attribute -->  <img v-bind:src="imageSrc" />  <!-- 缩写 -->  <img :src="imageSrc" />  <!-- 内联字符串拼接 -->  <img :src="'/path/to/images/' + fileName" />  <!-- 动态 attribute 名 -->  <button v-bind:[key]="value"></button>  <!-- 动态 attribute 名缩写 -->  <button :[key]="value"></button> |

**例3-8 v-bind示例**

### 3.3.2绑定class、style与prop

v-bind在绑定class或style的attribute时，支持其它类型的值，如数组或对象。虽然类名class和样式style可接收的类型都是字符串，但类名实际上是由数组拼接而成，而样式则是由对象键值对拼接而成的。

在绑定 prop时，prop必须在子组件中声明。可以用修饰符指定不同的绑定类型。如例3-9所示。

|  |
| --- |
| <!-- class 绑定 -->  <div :class="[classA, { classB: isB, classC: isC }]">  <!-- style 绑定 -->  <div :style="{ fontSize: size + 'px' }"></div>  <div :style="[styleObjectA, styleObjectB]"></div>  <!-- prop 绑定。"prop" 必须在 my-component 声明 -->  <my-component :prop="someThing"></my-component>  <!-- 通过 $props 将父组件的 props 一起传给子组件 -->  <child-component v-bind="$props"></child-component>  </div> |

**例3-9 绑定class、style、prop的attribute**

其中，类名*classB*与*classC*分别依赖于数据*isB*和*isC*，当*isB*和*isC*为true时，div会拥有类名*classB*与*classC*，为false时则没有。也可以使用三元表达式来根据条件切换类名。

使用:style时，CSS属性名称使用驼峰命名（camelCase）或短横分隔命名（kebab-case）。Vue会自动给特殊的CSS属性名称增加前缀，比如transform。

## 3.4 v-model双向绑定

表单控件在项目中十分常用，例如输入框、选择等，用它们可以完成数据的录入、提交等操作。通过使用指令v-model可以完成表单的数据双向绑定。

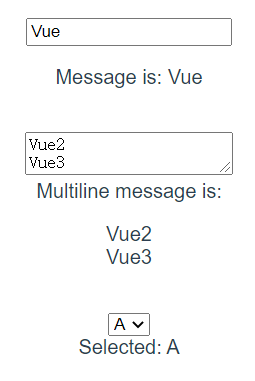
### 3.4.1 v-model指令

v-model指令用于在<input>、<textarea>及<select>等表单控件元素上创建双向绑定，它会根据控件类型自动选取正确的方法来更新元素。在修改表单元素值时，对应的实例中的属性值会被同时更新。反之，更改实例中的属性值，表单元素值也会被更新。

在例**3-10**中，给出了用v-model绑定<input>、<textarea>及<select>的简单例子，当表单中元素变化时，会更新属性值与视图。元素值更新前后如图**3-7**所示。

|  |
| --- |
| <!--输入-->  <input v-model="message" placeholder="请输入" />  <p>Message is: {{ message }}</p>  <br />    <!--多行文本-->  <textarea v-model="mulMessage" placeholder="请输入" ></textarea>  <br />  <span>Multiline message is:</span>  <p style="white-space: pre-line">{{ mulMessage }}</p>  <br />  <!--选择-->  <select v-model="selected">  <option>A</option>  <option>B</option>  <option>C</option>  </select>  <br />  <span>Selected: {{ selected }}</span> |

**例3-10 双向绑定message**

**图3-7 双向绑定message**

需要注意的是，在文本区域<textarea>中插值不起作用，应该使用v-model来代替。

另外，由于<select>的视图太差，而当前也不允许开发者自定义option的样式，所以一般都会使用其他元素来模拟下拉选择框。

在例**3-10**中，v-model绑定的值是一个字符串或布尔值。当需要绑定一个动态的数据，可以用v-bind来实现。

### 3.4.2 v-model与修饰符

v-model的修饰符可以用于控制数据同步的时机。各个修饰符的功能如表3-1所示。

|  |  |
| --- | --- |
| 修饰符 | 功能 |
| .lazy | 将用户输入的数据赋值于变量的时机延迟到数据改变时 |
| .number | 将用户输入转换为数值类型 |
| .tmn | 删除用户输入的首尾空白字符 |

**表3-1 v-mdoel的修饰符**

在输入框中，v-model默认是在每次input事件触发后将输入框的值与数据进行同步。使用修饰符.lazy会转变为在change事件中同步。数据不是实时改变的，而是在失焦或按回车时才更新。在例**3-10**的<input>中加入.lazy，在输入框内输入时，*message*不会立刻更新并显示，而是在输入回车，或是点击页面其他部分时更新*message*。

使用修饰符.number可以将输入转换为Number类型，默认情况下会将输入当做String类型。

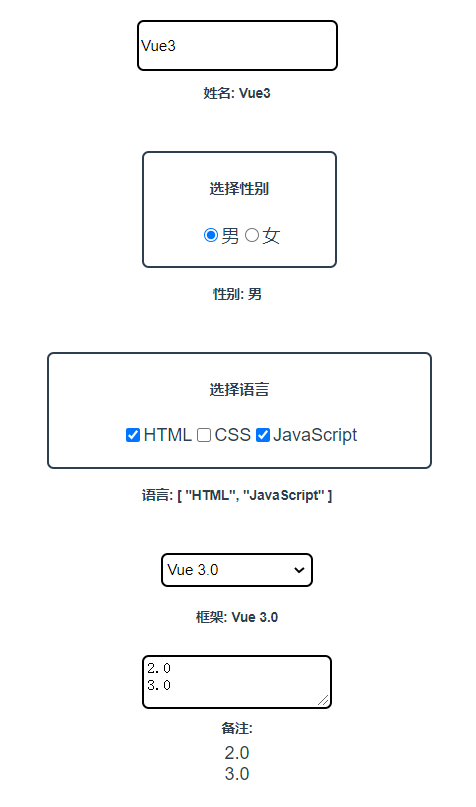
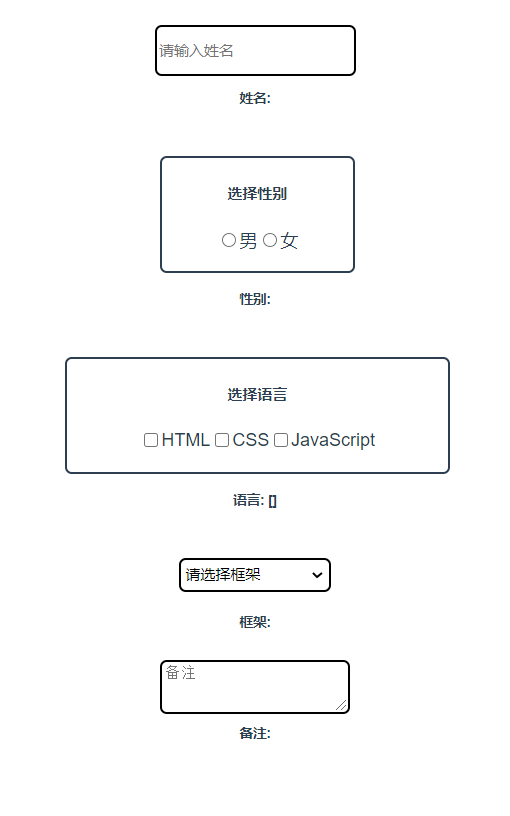
### 3.4.3双向绑定实例：制作问卷

学习了使用v-model进行双向绑定，现在可以制作一份实时显示各个表单控件元素中所填数据的问卷。通过指定<input>中的type，可以实现输入、单选、多选等。

通过问卷，采集了姓名、性别、编程语言、框架以及备注信息。代码如例3-11所示。代码运行结果如图3-8所示。

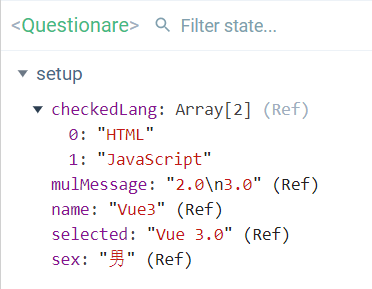
|  |
| --- |
| <template>  <div id="app">  <input v-model.trim="name" placeholder="请输入姓名" class="inText" />  <h6>姓名: {{ name }}</h6>  <br />  <div class="sex">  <h5>选择性别</h5>  <input type="radio" id="男" value="男" v-model="sex" />  <label for="男">男</label>  <input type="radio" id="女" value="女" v-model="sex" />  <label for="女">女</label>  </div>  <h6>性别: {{ sex }}</h6>  <br />  <div class="mulChoice">  <h5>选择语言</h5>  <input type="checkbox" id="HTML" value="HTML" v-model="checkedLang" />  <label for="HTML">HTML</label>  <input type="checkbox" id="CSS" value="CSS" v-model="checkedLang" />  <label for="CSS">CSS</label>  <input  type="checkbox"  id="JavaScript"  value="JavaScript"  v-model="checkedLang"  />  <label for="JavaScript">JavaScript</label>  </div>  <h6>语言: {{ checkedLang }}</h6>  <br />  <select v-model="selected" class="select">  <option disabled selected value style="display: none">请选择框架</option>  <option>Vue 2.x</option>  <option>Vue 3.0</option>  </select>  <h6>框架: {{ selected }}</h6>  <textarea  v-model="mulMessage"  placeholder="备注"  class="mulText"  ></textarea>  <br />  <h6>备注:</h6>  <p style="white-space: pre-line;margin-top: -20px ">{{ mulMessage }}</p>  <br />  </div>  </template>  <script>  import { reactive, ref, toRefs } from "vue";  export default {  setup() {  const data = reactive({  name: "",  mulMessage: "",  selected: "",  sex: "",  checkedLang: [],  });  return {  ...toRefs(data),  };  },  };  </script> |

**例3-11 问卷**



**图3-8 问卷前后效果**

可以看到，当每次在表单控件中输入信息时，数据会实时地被显示在页面中，而不需要对页面进行刷新操作。打开Vue Devtools，可以看到具体的数据内容以及对应的类型，每次对问卷中信息的修改也会直接反映在数据上。如图3-9所示。



**图3-9 在Vue Devtools中查看问卷数据**

## 3.5 v-if/v-show条件渲染

类似于其他程序中的条件语句if、else 、else if，Vue的条件指令同样可以根据表达式的值，在DOM中渲染或销毁元素与组件，称为条件渲染。

### 3.5.1 v-if，v-else与v-else-if

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真值的时候被渲染。

v-else-if用于充当v-if的“else-if块”，要紧跟v-if。当v-if中的表达式返回值为假时，开始判断v-else-if的表达式值，并根据返回值真假进行渲染。v-else-if可以连续使用。

v-else用来表示v-if的“else 块”，要紧跟v-else-if或v-if，当v-else-if与v-if的表达式值均为假时，渲染v-else中的内容。

在例3-12中使用了v-if、v-else-if与v-else进行条件渲染，当*type*的值为‘A’时，在页面中显示‘A’，为‘B’时显示‘B’，为‘C’时显示‘C’，其他情况下显示‘Not A/B/C’。

|  |
| --- |
| <template>  <div id="app">  <div v-if="type === 'A'">A</div>  <div v-else-if="type === 'B'">B</div>  <div v-else-if="type === 'C'">C</div>  <div v-else>Not A/B/C</div>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  type: 'A',  });  return {  ...toRefs(data),  };  },  };  </script> |

**例3-12 根据*type*进行条件渲染**

其中，在进行相等的条件判断时，应该使用===。===是Ecmascript的语言，表示严格等于判断，由于JS中的==有缺陷，因此在绝大多数情况下，都是使用三个等号的形式。

因为v-if是一个指令，所以必须将它添加到一个元素上。因此，如果想要一次判断多个元素，可以把一个Vue内置的<template>元素当做不可见的包裹元素，在上面使用v-if。最终的渲染结果将不包含<template>元素。如例3-13所示，Content A，B，C将会由同一个v-if指令决定是否渲染。

|  |
| --- |
| <template v-if="true">  <h1>Content A</h1>  <h1>Content B</h1>  <h1>Content C</h1>  </template> |

**例3-13 在 <template>上使用条件渲染**

### 3.5.2 v-show

v-show指令同样可以用于根据条件展示元素。用法与v-if大致一样。不同的是带有 v-show的元素始终会被渲染并保留在DOM中，因为v-show只是简单地切换元素CSS属性的display。当条件判定为假时，元素的display将被赋值为none；反之，元素的display将被设置为原有值。另外，v-show不支持<template>元素，也不支持v-else。

### 3.5.3 v-if对比v-show

v-if 和v-show 具有类似的功能，不过v-if 才是真正的条件渲染，它会根据表达式适当地销毁或重建元素及绑定的事件或子组件。若表达式初始值为false ，则一开始元素或组件并不会渲染，只有当条件第一次变为真时才开始编译。

而v-show只是简单的CSS属性切换，无论条件真与否，都会被编译。元素或组件保留在DOM中。

根据各自的特性，不难发现v-if更适合条件不经常改变的场景，因为它切换开销相对较大，而v-show 适用于频繁切换条件。

### 3.5.4条件渲染实例：按钮权限控制

在实际的网站开发中，往往会遇到需要对用户进行区分的情况，不同的用户也往往拥有不同的权限或功能。此时，通过条件渲染，可以实现对于不同的用户种类渲染出不同页面的效果。

在例3-14中，实现了根据用户作为学生、老师、助教的不同身份进行不同前端页面显示的效果。代码的运行结果如图3-10所示。

|  |
| --- |
| <template>  <div id="app">  <h1>User Type:</h1>  <template v-if="user === 'student'">  <h2>{{ user }}</h2>  <br />  <button @click="joinClass">加入课程</button>  </template>  <template v-if="user === 'teacher'">  <h2>{{ user }}</h2>  <br />  <button @click="startClass">开始上课</button>  </template>  <template v-if="user === 'teaching assistant'">  <h2>{{ user }}</h2>  <br />  <button @click="manageStudent">管理学生</button>  </template>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  user: "teacher",  });  const joinClass = () => {  console.log("Request already sent!");  };  const startClass = () => {  console.log("Class begin!");  };  const manageStudent = () => {  console.log("Checking requests!");  };  return {  ...toRefs(data),  joinClass,  startClass,  manageStudent,  };  },  };  </script> |

**例3-14 根据用户身份显示不同前端页面**

**图3-10 不同用户身份下的显示与输出**

通过修改按钮的方法，可以实现对不同用户进行功能区分的效果。

## 3.6 v-for列表渲染

### 3.6.1 v-for指令

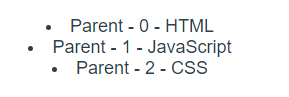
当需要将一个数组遍历或枚举一个对象循环显示时，就会用到列表渲染指令v-for来渲染一个列表。它的表达式需结合in来使用，类似item in items的形式。其中items是源数据数组，而item则是被迭代的数组元素的别名。也可以用of替代in作为分隔符。

v-for也可以像v-if一样用在内置标签＜template＞上，渲染多个元素。在自定义组件上同样可以使用v-for。但是由于组件有自己独立的作用域，数据不会被自动传递到组件中。此时，需要使用props把迭代数据传递到组件里。

在v-for块中，我们可以访问所有父作用域的属性。v-for还支持一个可选的第二个参数index，即当前项的索引。索引从0开始计数。具体代码如例3-15所示。

|  |
| --- |
| <template>  <div id="app">  <li v-for="(item,index) in items">  {{ parentMessage }} - {{ index }} - {{ item.message }}  </li>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  parentMessage: "Parent",  items: [  { message: "HTML" },  { message: "JavaScript" },  { message: "CSS" },  ],  });  return {  ...toRefs(data),  };  },  };  </script> |

**例3-15 用v-for显示数组中的message元素**



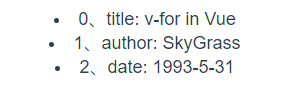
**图3-11 列表渲染**

### 3.6.2在v-for里使用对象

使用v-for可以遍历一个对象的属性。可以提供第二个的参数为属性的名称，即键名key。可以用第三个参数作为索引index。

|  |
| --- |
| <template>  <div id="app">  <li v-for="(value, name, index) in obj">  {{ index }}、{{ name }}: {{ value }}  </li>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  obj: {  title: "v-for in Vue",  author: "SkyGrass",  date: "1993-5-31",  },  });  return {  ...toRefs(data),  };  },  };  </script> |

**例3-16 用v-for遍历对象属性**



**图3-12 遍历对象属性结果**

### 3.6.3列表的更新

Vue的核心是数据与视图的双向绑定，当修改数组时，Vue会检测到数据变化，所以用v-for渲染的视图也会立即更新。Vue将被侦听的数组的变更方法进行了包裹，所以使用它们改变数组会触发视图更新。这些被包裹的方法与说明如表3-1所示。

|  |  |
| --- | --- |
| **名称** | **说明** |
| push() | 将一个或多个元素添加至数组末尾，返回新数组的长度 |
| pop() | 从数组中删除并返回最后一个元素 |
| shift() | 从数组中删除并返回第一个元素 |
| unshift() | 将一个或多个元素添加至数组开头，并返回新数组的长度 |
| splice() | 从数组中删除元素或向数组添加元素 |
| sort() | 对数组元素排序，默认按照Unicode编码排序，返回排序后的数组 |
| reverse() | 将数组中的元素位置颠倒，返回颠倒后的数组 |

**表3-1 数组的变更方法**

使用以上方法会改变被这些方法调用的原始数组，此外还有一些方法不会改变原数组，例如filter()、concat()、slice()，它们返回的是一个新数组。

需要注意的是，当直接使用下标或键名为数组或对象设置成员，以及修改数组长度时，Vue并不会将其加入数据响应式系统，如：arr[‘index’] = 'value' 、obj [' key '] = 'value'或arr.length=1。此时即使数据被修改，视图也不会进行更新。

### 3.6.4列表渲染的key

在使用v-for时，最好为每个迭代元素提供一个值不重复的key，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素。因为，当列表渲染被重新执行(数组内容发生改变)时，如果不使用key，Vue将不会移动DOM元素来匹配数据项的顺序，而是就地更新每个元素，并且确保它们在每个索引位置正确渲染。

通过为每项提供一个唯一的key，给 Vue进行提示，Vue就会根据key的变化重新排列节点顺序，并移除key不存在的节点。实质上，key的存在是为DOM节点标注了一个身份信息，让Vue能够有迹可循追踪到数据对应的节点。

在实战开发中，是否使用key都不会影响功能的实现。建议尽可能在使用v-for时提供key，如例3-17所示。

|  |
| --- |
| <div v-for="item in items" :key="item.id">  <!-- content -->  </div> |

**例3-17 指定v-for的key**

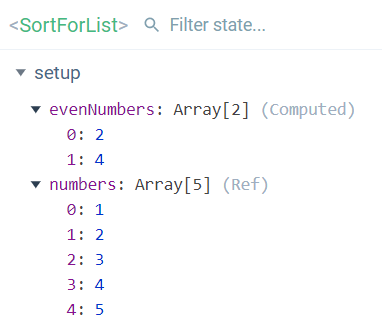
### 3.6.5过滤与排序

在展示数据时，往往需要在不变更或重置原始数据的情况下，显示经过过滤或排序的数组。在这种情况下，可以创建一个计算属性，来返回过滤或排序后的数组。

在例3-18中，对原有数组进行了过滤，筛选出了其中的偶数，Vue Devtools中的结果如图3-13所示。

|  |
| --- |
| <template>  <div id="app">  <li v-for="n in evenNumbers">  {{ n }}  </li>  </div>  </template>  <script>  import { computed, reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  numbers: [1, 2, 3, 4, 5],  });  const evenNumbers = computed(() => {  return data.numbers.filter((number) => number % 2 === 0);  });  return {  ...toRefs(data),  evenNumbers,  };  },  };  </script> |

**例3-18 过滤出数组中的偶数**



**图3-13 过滤数组的结果**

在计算属性不适用的情况下，例如，在嵌套列表渲染v-for的循环中，可以使用一个方法来达到相同的效果。

### 3.6.6 v-for与v-if共用

在多数情况下，不推荐在同一元素上同时使用v-if和v-for。当它们处于同一节点时，v-if的优先级比v-for更高，因此v-if将没有权限访问v-for里的变量。可以通过把v-for移动到外层的<template>标签中来实现相同的效果。

### 3.6.7列表渲染实例：帖子列表

学习完列表渲染v-for与条件渲染v-if，可以尝试制作一个帖子列表。帖子带有收藏功能，被收藏的帖子标题将会以红色显示，并带有收藏与取消收藏按钮。

具体代码如例3-19所示，代码的运行结果如图3-14所示。通过修改addStar(item)方法的内容，可以实现修改收藏状态的效果。在此基础上，也可以引申出置顶、删除帖子等许多方式。

|  |
| --- |
| <template>  <div id="app">  <div v-for="(item, index) in posts" :key="index" class="post">  <h1 v-if="item.isStar === true" style="color: red">  {{ item.title }}  </h1>  <h1 v-if="item.isStar === false">  {{ item.title }}  </h1>  <div class="content">  {{ item.detail }}  </div>  <button class="star" v-if="item.isStar === false" @click="addStar(item)">  收藏  </button>  <button class="star" v-if="item.isStar === true" @click="addStar(item)">  取消收藏  </button>  </div>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  posts: [  { title: "HTML", detail: "Something about HTML.", isStar: false },  {  title: "JavaScript",  detail: "Something about JavaScript.",  isStar: true,  },  { title: "CSS", detail: "Something about CSS.", isStar: false },  ],  });  const addStar = (item) => {  //do something  };  return {  ...toRefs(data),  addStar,  };  },  };  </script> |

**例3-19 帖子列表**



**图3-14 帖子列表的显示**

## 3.7 v-on事件绑定

### 3.7.1 v-on指令

v-on指令用于监听 DOM事件，并在触发事件时执行一些JavaScript，通常缩写为@符号。直接把JavaScript代码写在v-on指令中是一种方式。但是许多事件的处理过程十分复杂，因此v-on还可以接收一个需要调用的方法名称。用法为v-on:click="methodName" 或使用快捷方式@click="methodName"。

不难发现，事件绑定已经在之前的例子中出现过很多次。按钮的点击事件@click即等同于v-on:click。除了点击事件，v-on后面还可以接其他HTML的标准事件，如表3-2所示。

|  |  |
| --- | --- |
| **事件** | **说明** |
| click | 单击鼠标左键 |
| dblclick | 双击鼠标左键 |
| contextmenu | 单击鼠标右键 |
| mouseover | 指针移到有事件监听的元素或其子元素内 |
| mouseout | 指针移出元素，或者移到其子元素上 |
| keydown | 键盘动作：按下任意键 |
| keyup | 键盘动作：释放任意键 |

**表3-2 部分HTML的标准事件**

### 3.7.2事件修饰符

Vue为v-on提供了事件修饰符，修饰符是由点开头的指令后缀来表示的。使用多个修饰符时，顺序很重要。常见的修饰符见表3-3。

|  |  |  |
| --- | --- | --- |
| **名称** | **可用事件** | **说明** |
| .stop | 任意 | .当事件触发时，阻止事件冒泡 |
| .prevent | 任意 | 当事件触发时，阻止兀素默认行为 |
| .capture | 任意 | 当事件触发时，阻止事件捕获 |
| .self自身 | 任意 | 限制事件仅作用于节点自身 |
| .once | 任意 | 事件被触发一次后即解除监昕 |
| .passlve | 滚动 | 移动端，限制事件永不调用preventDefault ()方法 |

**表3-3 常见的事件修饰符**

Vue 允许为v-on或者@在监听键盘事件时添加按键修饰符，用于检查详细的按键。使用方法为：@keyup.enter=" methodName"。以下是全部的别名：

• .enter

• .tab

• .delete （“删除”和“退格”）

• .esc

• .space

• .up

• .down

• .left

• .right

除了键盘按键之外， Vue 也为鼠标按键配置了修饰符，包括.left、.right与.middle。

当需要按住多键或者鼠标与键盘共用以实现操作时, 可以使用组合修饰符，需要配合系统按键修饰使用，包括.ctrl、.alt、.shift与.meta。在Mac系统键盘上，meta对应command键。在Windows系统键盘上，meta对应Window键。

## 3.8指令在Vue 3.x中的变化

### 3.8.1 v-if与v-for的key

对于v-if/v-else/v-else-if的各分支项key属性不再是必须的，因为，即使没有为条件分支提供key，Vue 3.x也会自动生成唯一的key。因此，不建议在v-if/v-else/v-else-if的分支中继续使用key属性。如果主动提供key，那么每个分支必须使用唯一的key。故意使用相同的key也无法强制重用分支。

在Vue 3.x中，<template v-for>的key 应该设置在<template>标签上 ，而不是设置在它的子节点上。在Vue 2.x中，由于<template>标签不能拥有key，可以为其每个子节点分别设置key。同样，当使用<template v-for>时存在使用v-if的子节点，key应改为设置在<template>标签上。

### 3.8.2 v-if与v-for的优先级

在Vue 2.x版本中，在一个元素上同时使用v-if和v-for时，v-for会优先作用。在Vue 3.x版本中，v-if会拥有比v-for更高的优先级。

由于语法上存在歧义，建议避免在同一元素上同时使用两者。具体可以参考v-for章节中提到的方法。

### 3.8.3 v-bind合并行为

在元素上动态绑定attribute时，在一个元素中同时使用v-bind="object"语法和单独的property是一种常用的做法。这就涉及到了合并的优先级的问题。

在Vue 2.x，如果一个元素同时定义了v-bind="object" 和一个相同的单独的property，那么这个单独的property总是会覆盖object中的绑定。在Vue 3.x，声明绑定的顺序决定了它们如何合并。现在开发者对自己所希望的合并行为有了更好的控制。

如果希望依赖v-bind的覆盖功能，建议确保在单独的property之前定义v-bind属性。

### 3.8.4 v-for中的Ref数组

在Vue 2.x中，在v-for里使用的ref attribute会用ref数组填充相应的$refs property。但是，当存在嵌套的v-for时，会变得不明确且效率低下。

在 Vue 3 中，将不再在$ref中自动创建数组。要从单个绑定获取多个ref，可以将ref绑定到一个更灵活的函数上。如例3-20所示。

|  |
| --- |
| <template>  <div id="app">  <div v-for="item in list" :ref="setItemRef"></div>  </div>  </template>  <script>  import { ref, onBeforeUpdate, onUpdated } from 'vue'  export default {  setup() {  let itemRefs = []  const setItemRef = el => {  itemRefs.push(el)  }  onBeforeUpdate(() => {  itemRefs = []  })  onUpdated(() => {  console.log(itemRefs)  })  return {  itemRefs,  setItemRef  }  }  }  </script> |

**例3-20 ref绑定到函数**

值得注意的是，itemRefs不必是数组，它也可以是一个对象，其ref会通过迭代的 key被设置。itemRef也可以是响应式的且可以被监听。

### 3.8.5 v-model

在用于自定义组件时，v-model的prop和事件默认名称已更改。prop从value变更为modelValue，event从inpu变更为update:modelValue。v-bind的.sync修饰符和组件的model选项已移除，用v-model作为代替。现在可以在同一个组件上使用多个v-model进行双向绑定。

除了像.trim这样的Vue 2.x硬编码的v-model修饰符外，现在Vue 3.x还支持自定义v-model 修饰符。

迁移时，可以将所有使用.sync的部分替换为v-model。对于所有不带参数的v-model，确保分别将prop和event命名更改为modelValue和update:modelValue。

更多关于v-model与组件的更新将会在组件的章节进行说明。

## 3.9本章小结

本章介绍了插值绑定、计算属性、属性绑定、双向绑定、条件渲染、列表渲染、事件绑定以及Vue的内置指令从Vue 2.x到Vue 3.x的变化。

文本插值最基本的方法是使用双大括号（Mustache语法）“{{ }}”，Vue会将大括号里的内容替换为表达式值，以文本的形式将其展示出来。表达式可由JavaScript表达式和过滤器构成。通过任何方法修改数据设定值，大括号的内容都会被实时替换。Vue支持在“{{ }}”插值的尾部添加过滤器，用管道符“ |” 表示。经常用于格式化文本，

计算属性会在其依赖属性的值发生变化时，对属性的值进行自动更新，同时更新相关的DOM 部分。通过从vue中导入computed来使用计算属性。侦听属性watch用来监视指定数据项的变化，从而触发用户自定的操作。默认情况下，回调仅在侦听的数据源发生改变时调用。通过从vue中导入watch来使用侦听属性。

v-bind指令主要用于动态绑定DOM元素属性，可以将一个或多个attribute，或一个组件prop动态地绑定到表达式。

v-model指令用于在<input>、<textarea>及<select>等表单控件元素上创建双向绑定，它会根据控件类型自动选取正确的方法来更新元素。在修改表单元素值时，对应的实例中的属性值会被同时更新。

Vue根据表达式的值，在DOM中渲染或销毁元素与组件，称为条件渲染。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真值的时候被渲染。当v-if中的表达式返回值为假时，根据v-else-if的表达式值的真假进行渲染。当v-else-if与v-if的表达式值均为假时，渲染v-else中的内容。v-show用法与v-if大致一样。因为v-show只是切换元素CSS属性的display，所以v-show的元素始终会被渲染并保留在DOM中。

使用列表渲染指令v-for可以渲染一个列表，用于遍历一个数组或枚举一个对象循环显示。它的表达式需结合in来使用。

v-on指令用于监听 DOM事件，并在触发事件时执行一些JavaScript，通常缩写为@符号。

在指令方面，Vue 3.x的更新集中在v-if与v-for的key、v-if与v-for的优先级、v-bind合并行为、v-for中的Ref数组和v-model。

在下一章中，将会介绍Class与Style绑定的问题。

## 3.10练习题

* **问答题：**

1. v-if与v-for在同一元素上使用时优先级是怎样的？如何避免这种情况？
2. 在文本插值中，如何实现简单的条件判断？
3. 简要说明如何创建仅可读和可读可写的计算属性。
4. 分别说明几种修饰符如何控制v-model中数据同步的时机。
5. 列举v-if与v-show的异同。
6. 解释v-for中各参数以及key的作用。
7. 列举什么情况下对v-for数组与对象的更新不会导致视图的更新。

* **动手做：**

1. 尝试使用v-model、v-if与v-for的知识，制作一个简易贴吧。有普通用户和管理员两种角色，普通用户可以发帖、收藏帖子。管理员可以发帖、置顶帖子。

# 第4章 Class与Style绑定

在实际开发的过程中，DOM元素经常会动态地绑定一些class类名或style样式，本章将会对Vue 3中的绑定HTML Class与绑定内联样式进行介绍。在将v-bind用于class和style时，Vue做了专门的增强，本章也会展示使用v-bind指令来绑定class和style的多种方法。

## 4.1绑定HTML Class

### 4.1.1对象语法

通过传给v-bind:class一个对象，用于动态地切换class，可以将v-bind:class简写为:class。Vue也支持在对象中传入更多字段来动态切换多个class。:class指令也可以与普通的class attribute同时存在，如例4-1所示。

|  |
| --- |
| <template>  <div class="app">  <div  class="static"  :class="{ active: isActive, 'text-danger': hasError }"  ></div>  </div>  </template>  <script>  import { toRefs,reactive } from "vue";  export default {  setup() {  const data = reactive({  isActive: true,  hasError: false,  });  return {  ...toRefs(data),  };  },  };  </script> |

**例4-1 动态切换class**

上面的语法表示active这个class的存在与否取决于isActive的真假，text-danger的存在与否取决于hasError的真假。当isActive或者hasError变化时，class的值也将相应地更新。上面代码的渲染结果为：

<div class="static active"></div>

当变量值为undefined、null、值为0的数字、空字符串时，会被判定为假。[]、{}、-1、-0.1 也会被判定为真。

除了这种形式，也可以直接绑定一个Object数据或绑定一个返回对象的计算属性。一般当条件多于两个时，往往会使用计算属性，是一种很便利和常见的用法，如例4-2所示。

|  |
| --- |
| <div :class="classObject"></div>  const data = reactive({  isActive: true,  hasError: false,  });  const classObject = computed(() => {  return {  active: data.isActive && !data.error,  "text-danger": data.error && data.error.type === "fatal",  };  }); |

**例4-2 通过计算属性切换class**

### 4.1.2数组语法

与对象语法类似，可以把一个数组传给:class，用来应用一个class列表。想根据条件切换列表中的class，也可以使用三元表达式。当有多个条件 class 时这样写有些繁琐。所以在数组语法中也可以使用对象语法。在例4-3中分别使用了普通数组、三元表达式与对象语法。

|  |
| --- |
| <template>  <div class="app">  <div :class="[activeClass, errorClass]"></div>  <div :class="[{ active: isActive }, hasError ? errorClass : '']"></div>  </template>  <script>  import { toRefs,reactive } from "vue";  export default {  setup() {  const data = reactive({  activeClass: "active",  errorClass: "text-danger",  });  return {  ...toRefs(data),  };  },  };  </script> |

**例4-3 通过计算属性切换class**

上面代码的渲染结果为：

<div class="active text-danger"></div>

与对象语法一样，数组语法也可以使用data 、computed 和methods三种方法。在开发过程中，如果表达式较长或逻辑复杂，应该尽可能优先使用计算属性给元素动态设置类名。

## 4.2绑定内联样式

### 4.2.1对象语法

使用v-bind:style可以给元素绑定内联样式，方法与:class 类似，也有对象语法和数组语法，类似直接在元素上写CSS，同样可以简写为:style。

:style的语法十分直观，十分接近CSS，但其实是一个JavaScript对象。其中CSS property名可以用驼峰式(camelCase)或短横线分隔(kebab-case)来命名。

由于直接绑定到一个样式对象会让模板更清晰，因此通常使用对象的方式。同样的，对象语法也常常结合返回对象的计算属性使用。

在例4-4中分别使用了两种绑定方式，结果如图4-1所示。

|  |
| --- |
| <template>  <div class="app">  <div :style="{ color: activeColor, fontSize: fontSize + 'px' }">string style</div>  <div :style="styleObject">object style</div>  </div>  </template>  <script>  import { toRefs, reactive } from "vue";  export default {  setup() {  const data = reactive({  activeColor: "red",  fontSize: 30,  styleObject: {  color: 'red',  fontSize: '13px'  }  });  return {  ...toRefs(data),  };  },  };  </script> |

**例4-4 对象绑定样式**



**图4-1 对象绑定样式结果**

### 4.2.2数组语法

:style的数组语法可以将多个样式对象应用到同一个元素上，如下所示。

<div :style="[styleA , styleB]"></div>

在实际开发中，:style的数组语法并不常用，因为往往可以写在一个对象里面，这样对代码的理解更为直观。较为常用的是计算属性。

在:style中使用特殊的CSS property时，例如transform，Vue将自动添加相应的前缀。

## 4.3本章小结

本章介绍了Class与Style绑定，包括绑定HTML Class与绑定内联样式。分别可以通过对象语法和数组语法实现对class与style的动态切换。对象语法与数组语法一样，都可以使用data 、computed 和methods三种方法，其中最为常用与便捷的是计算属性。

通过为表达式结果的类型增加对象与数组，避免了将v-bind用于class和style时表达式字符串拼接麻烦且易错的问题。

在下一章中，将会进一步介绍过滤器。

## 4.4练习题

* **问答题：**

1. 列举:style中CSS property名的命名方法。
2. 分别说明对象语法中，条件判断结果为真或假的值有那些情况。
3. 简要说明绑定class与style使用计算属性的优势有哪些。

* **动手做：**

1. 设计一个按钮，让它可以根据两种用户身份切换激活与否。
2. 修改上一章的练习，通过绑定内联样式，让被收藏的帖子标题显示为黄色、置顶帖子的标题显示为红色。

# 第5章 过滤器

从Vue 3.0开始，过滤器已删除，不再支持。建议用计算属性或方法代替过滤器，而不是使用过滤器。因此，本章将简单介绍Vue原有的内置过滤器，而不对过滤器做过多的深入。

在第3章插值绑定中提到过，Vue 2.x允许在表达式后面添加可选的过滤器，以管道符|表示。事实上，过滤器的本质是一个函数，Vue 2.x支持在任何出现表达式的地方添加过滤器。过滤器接受管道符前面的值作为初始值，返回值为经过处理后的输出值。同时，过滤器也能接受额外的参数，始终以表达式的值作为第一个参数，其他参数跟在过滤器名称后面，参数之间以空格分隔。多个过滤器也可以进行串联，上一个过滤器的输出结果可以作为下一个过滤器的输入。在本章中将进一步介绍过滤器。

## 5.1内置过滤器

### 5.1.1字母操作

Vue 2.x内置了三个字母过滤器capitalize、uppercase、lowercase用于处理英文字符。

capitalize过滤器用于将表达式中的首字母转换为大写。

uppercase过滤器用于将表达式中的所有字母转换为大写。

lowercase过滤器用于将表达式中的所有字母转换为小写。

三种过滤器的用法如例5-1所示。

|  |
| --- |
| （｛ ’ abcd ’ | capitalize }}  ｛｛ ’ abcd ’ | uppercase }}  ｛｛ ’ ABCD ’ | lowercase }} |

**例5-1 字母过滤器的使用**

上面三段代码将分别输出‘Abcd’、‘ABCD’和‘abcd’。

### 5.1.2 json过滤器

Vue 2.x的json过滤器可以将表达式的值转换为JSON字符串，输出的结果等于表达式经过JSON.stringify()处理后的结果。json过滤器可接受一个类型为Number的参数，用于决定转换后的JSON字符串的缩进距离，默认的缩进距离为2。使用方式如例5-2：

|  |
| --- |
| {{ student | json 4 }} |

**例5-2 json过滤器的使用**

### 5.1.3限制过滤器

Vue 2.x内置了三个限制过滤器limitBy、filterBy、orderBy用于处理井返回过滤后的数组。这三个过滤器所处理的表达式的值必须是数组。

limitBy过滤器的作用是限制数组为前N个元素，通过传入的第一个参数指定N。第二个参数可选，用于指定开始的偏移量，默认为0不偏移。

filterBy过滤器的第一个参数可以是字符串或者函数。如果第一个参数是字符串，那么将在每个数组元素中搜索它，返回包含该字符串的元素组成的数组。如果filterBy的第一个参数是函数，则过滤器将根据函数的返回结果进行过滤。此时filterBy过滤器将调用JavaScript组中内置的函数filter()对数组进行处理，待过滤数组中的每个元素都将作为参数输入并执行传入filterBy中的函数。只有函数返回结果为true的数组元素才符合条件并将存入一个新的数组，最终filterBy的返回结果即为这个新的数组。

在对象中，过滤器将在所有属性中搜索。可以通过指定搜索宇段的方式缩小搜索范围。在多个字段中搜索时，字段与字段之间通过空格分隔，或者将搜索字段存放在一个数组中。如例5-3所示。

|  |
| --- |
| <div v-for＝”item in items | limitBy 10 5”  ＞＜/div>  <div v-for＝”student in students | filterBy ’sky’ in ’name’ ’nickname’” ＞＜／div> |

**例5-3 limitBy与filterBy过滤器的使用**

此外，还可以使用动态参数作为搜索目标或搜索字段。结合v-model实现输入提示效果。通过将输入值作为检索字段，实现输入提示。如例5-4所示。

|  |
| --- |
| <template>  <div class="app">  <input v-model="input" />  <li v-for="user in users | filterBy input in 'name'">  { { user. name } }  </li>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  input: "",  users: [{ name: "SKY" }, { name: "GRASS" }, { name: "NANA" }],  });  return {  ...toRefs(data),  };  },  };  </script> |

**例5-4 动态提示**

orderBy过滤器的作用是返回排序后的数组。第一个参数可以是字符串、数组或者函数。第二个参数order可选，order<0为降序，其他为升序，默认为1升序排列。

若输入参数为字符串，则可同时传入多个字符串作为排序键，字符串之间以空格分隔，或者存入一个数组中。此时将按照传入键名的顺序或数组的顺序进行排序。如例5-5所示。

|  |
| --- |
| <li v-for=" user in users | orderBy ' lastName '' firstName '' age ' "></li>  <li v-for=" user in users | orderBy sortList"></li> |

**例5-5 orderBy过滤器的使用**

### 5.1.4 currency过滤器

currency过滤器的作用是将数字转换为货币形式输出。第一个参数为String类型的货币符号，默认为美元符号$。第二个参数为Number类型，表示保留的小数位，默认为2。

|  |
| --- |
| <div>{{ amount | currency}}</div>  <div>{{ amount | currency '￥' }}</div>  <div>{{ amount | currency '$' 3}}</div> |

**例5-6 currency过滤器的使用**

当amount为12345时，上面得代码将分别显示$12,345.00、￥12,345.00和$12,345.000

### 5.1.5 debounce过滤器

debounce过滤器的作用是延迟执行时间。接受的表达式的值必须为函数，一般与v-on等指令结合使用。debounce接受一个可选的参数作为延迟时间，单位为毫秒，默认为300毫秒。经过debounce包装的处理器在调用之后将至少延迟设定的时间再执行。如果在延迟结束前再次调用，则延迟时长将重置为设定的时间。使用方法如例5-7所示,实现对了按键事件处理的延迟。

|  |
| --- |
| <input @keyup= "onKeyup | debounce 500"> |

**例5-6** debounce**过滤器的使用**

## 5.2本章小结

本章介绍了Vue原有的内置过滤器，字母过滤器、json过滤器、限制过滤器、currency过滤器和debounce过滤器。它们在处理表达式时有很便利的效果，但是这种方式打破了大括号内表达式“只是JavaScript”的原则。因此从Vue 3.0开始，建议用计算属性或方法替换过滤器。

在下一章中，将会进一步介绍过渡与动画的效果。

## 5.3练习题

* **问答题：**

1. 说明三种字母过滤器的作用。
2. 如何使用限制过滤器限制数组为第3个元素开始的10个元素？
3. currency过滤器的第一个参数采取默认形式，想要修改第二个参数（也就是小数位）时，该如何表示？

* **动手做：**

1. 使用计算属性，实现三种字母过滤器的效果。
2. 使用计算属性，在第三章的所有帖子标题与内容中搜索包含“sky”的部分，并返回包含“sky”的元素组成的数组。

# 第6章 过渡与动画

过波效果在在用户与网站交互的过程中提供了很好的体验。前面介绍的class和style声明就可以应用于动画和过渡，用于简单的用例。除此之外，Vue还内置了一套过渡系统用于帮助处理过渡和动画，可以在元素从DOM中插入或移除时触发CSS过渡或动画，自动应用过渡效果。Vue的过渡系统也支持JavaScript的过渡，使用钩子函数在过渡过程中执行自定义的DOM操作。

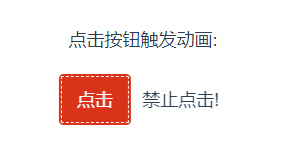
## 6.1过渡与动画概述

### 6.1.1基于class的动画和过渡

<transition> 组件对于组件的进入和离开非常有用，但也可以通过添加一个条件class 来激活动画，而无需挂载组件。通过class来激活动画的案例如例6-1所示。

|  |
| --- |
| <template>  <div id="app">  点击按钮触发动画:<br />  <div :class="{ shake: noActivated }">  <button @click="noActivated = true">点击</button>  <span v-if="noActivated">禁止点击!</span>  </div>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  noActivated: false,  });  return {  ...toRefs(data),  };  },  };  </script>  <style>  body {  margin: 30px;  }  button {  background: #d93419;  border-radius: 4px;  display: inline-block;  border: none;  padding: 0.75rem 1rem;  margin: 20px 10px 0 0;  text-decoration: none;  color: #ffffff;  font-family: sans-serif;  font-size: 1rem;  cursor: pointer;  text-align: center;  -webkit-appearance: none;  -moz-appearance: none;  }  button:focus {  outline: 1px dashed #fff;  outline-offset: -3px;  }  .shake {  animation: shake 0.82s cubic-bezier(0.36, 0.07, 0.19, 0.97) both;  transform: translate3d(0, 0, 0);  backface-visibility: hidden;  perspective: 1000px;  }  @keyframes shake {  10%,  90% {  transform: translate3d(-1px, 0, 0);  }  20%,  80% {  transform: translate3d(2px, 0, 0);  }  30%,  50%,  70% {  transform: translate3d(-4px, 0, 0);  }  40%,  60% {  transform: translate3d(4px, 0, 0);  }  }  </style> |

**例6-1 通过class激活动画**



**图6-1 动画前后效果**

点击按钮后的前后效果如图6-1所示。通过点击按钮会触发事件，将*noActived*属性的值修改为true，此时class的shake属性被激活，触发在<style>或CSS文件中的shake样式，展现出动画效果。同时v-if条件渲染的判断为真，在视图中渲染出“禁止点击”的字样。

### 6.1.2基于style的动画和过渡

一些过渡效果可以通过插值的方式来实现，这会导致在发生交互时将样式绑定到元素上。使用插值来创建动画的案例如例6-2所示。

|  |
| --- |
| <template>  <div id="app">  <div  @mousemove="xCoordinate"  :style="{ backgroundColor: `hsl(${x}, 80%, 50%)` }"  class="movearea"  >  <h3>在组件中左右移动鼠标</h3>  <p>x: {{ x }}</p>  </div>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  x: 0,  });  const xCoordinate = (e) => {  data.x = e.offsetX;  };  return {  ...toRefs(data),  xCoordinate,  };  },  };  </script>  <style>  .movearea {  transition: 0.2s background-color ease;  height: 100px;  width: 40%;  position: absolute;  left: 30%;  color: orange;  }  </style> |

**例6-2 将样式绑定到元素上**









**图6-2 不同鼠标位置时的效果**

在这个例子中使用了插值来创建动画，并且将动画的触发条件添加到了鼠标的移动过程上。同时将CSS过渡属性应用在元素上，让元素知道在更新时要使用什么过渡效果。在图6-2中展示了鼠标在块中左右移动时，<div>的背景颜色会随着*x*坐标不断变化。在JavaScript中，使用了event.offsetX来获取鼠标在<div>中的*x*坐标值，并将*x*值的变化通过hsl映射到背景颜色的变化上。

## 6.2单元素的过渡

### 6.2.1进入与离开过渡

Vue 提供了transition的封装组件，在使用包括条件渲染 (v-if)、条件展示 (v-show)、动态组件与组件根节点时，可以给任何元素和组件添加过渡的进入和离开。过渡动画的触发时机包括元素或组件初始渲染时、元素或组件显示或隐藏时(包括条件渲染时)、元素或组件切换时。在例6-3中，展示了在条件渲染情况下，元素渐进渐出效果的过渡。

|  |
| --- |
| <template>  <div id="app">  <button @click="show = !show">切换</button>  <transition name="fade">  <p v-if="show">Hello</p>  </transition>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  show: true,  });  return {  ...toRefs(data),  };  },  };  </script>  <style>  .fade-enter-active,  .fade-leave-active {  transition: opacity 0.5s ease;  }  .fade-enter-from,  .fade-leave-to {  opacity: 0;  }  </style> |

**例6-3 渐进渐出的过渡**

每当点击按钮时，会切换一次Hello的显示或隐藏。通过在0.5秒的持续时间内，对Hello的透明度进行改变，以实现渐进渐出的效果。

当插入或删除包含在transition组件中的元素时，Vue将会自动判断目标元素是否应用了CSS过渡动画，并在恰当的时机添加或删除带有过渡动画的CSS类名。另外，如果过渡组件提供了JavaScript钩子函数，钩子函数也将在恰当的时机被调用。如果没有找到JavaScript钩子函数，并且也没有检测到CSS过渡动画，则会在浏览器逐帧动画机制的下一帧中立即执行插入或删除的DOM操作。

在例6-3中，使用了fade-enter-from与fade-leave-to等类名，这些类名用于定义不同过渡阶段的样式。在进入或离开的过渡中，总共有6个class可以切换：

\*-enter-from定义进入过渡的开始状态，在元素被插入之前生效，在元素被插入之后的下一帧移除。

\*-enter-active定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡动画完成之后移除。这个类可以被用来定义进入过渡的过程时间、延迟和曲线函数。

\*-enter-to定义进入过渡的结束状态。在元素被插入之后下一帧生效，即\*-enter-from被移除的同时，在过渡动画完成之后移除。

\*-leave-from定义离开过渡的开始状态。在离开过渡被触发时立刻生效，下一帧被移除。

\*-leave-active定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡动画完成之后移除。这个类可以被用来定义离开过渡的过程时间、延迟和曲线函数。

\*-leave-to：离开过渡的结束状态。在离开过渡被触发之后下一帧生效，即\*-leave-from被删除的同时，在过渡动画完成之后移除。

如果使用一个没有名字的<transition>，则v-是这些类名的默认前缀。如果使用了 <transition name="my-transition">，那么\*-enter-from会替换为my-transition-enter-from。

### 6.2.2 CSS过渡与动画

常用的过渡都是使用CSS过渡。除了直接在元素上添加transition="name" 外，Vue也支持动态绑定CSS名称，便于元素根据场景使用多个过渡效果。如例6-4所示，在data中指定transitionName的值后就可以实现动态绑定。

|  |
| --- |
| <div v-if="show" :transition="transitionName"></div> |

**例6-4 动态绑定过渡CSS名称**

CSS动画用法类似CSS过渡，区别是在动画中，\*-enter-from类名在节点插入DOM后不会立即删除，而是在animationend事件触发时删除。在例6-5中展示了使用CSS动画制作弹跳显示效果。

|  |
| --- |
| <template>  <div id="app">  <button @click="show = !show">切换</button>  <transition name="bounce">  <p v-if="show">弹跳显示</p>  </transition>  </div>  </template>  <script>  import { reactive, toRefs } from "vue";  export default {  setup() {  const data = reactive({  show: true,  });  return {  ...toRefs(data),  };  },  };  </script>  <style>  .bounce-enter-active {  animation: bounce-in 0.5s;  }  .bounce-leave-active {  animation: bounce-in 0.5s reverse;  }  @keyframes bounce-in {  0% {  transform: scale(0);  }  50% {  transform: scale(1.25);  }  100% {  transform: scale(1);  }  }  </style> |

**例6-5 弹跳显示效果**

当点击切换按钮后，会首先放大再缩小“弹跳显示”的字样，直至消失。再次点击时，会反向完成之前的操作，实现弹跳显示的动画效果。

### 6.2.3自定义过渡class类名

在过渡的JavaScript中，可以通过enter-from-class、enter-active-class、enter-to-class、leave-from-class、leave-active-class、leave-to-class这几个attribute来自定义CSS过渡类名。他们的优先级高于普通的类名，会覆盖默认的类名。这样的方式有利于结合使用Vue的过渡系统和其他第三方CSS动画库。例如Animate.css是Vue官方推荐的CSS动画库，通过引入Animate.css的CSS文件就可以使用提供的预设动画。配合自定义过渡类名，可以达到非常不错的效果。

### 6.2.4 JavaScript过渡

类似CSS，同样可以在attribute中声明JavaScript钩子。只使用JavaScript过渡时，不需要定义任何CSS样式，但enter和leave钩子需要调用done函数，用来明确过渡结束时间，否则它们将被同步调用，导致过渡立即结束。推荐在只使用JavaScript钩子时，显式声明css: false, 此时Vue将跳过CSS检测，除了性能略高之外，还可以避免CSS规则干扰过渡。JavaScript钩子的使用如例6-6所示。

|  |
| --- |
| //HTML中  <transition  @before-enter="beforeEnter"  @enter="enter"  @after-enter="afterEnter"  @enter-cancelled="enterCancelled"  @before-leave="beforeLeave"  @leave="leave"  @after-leave="afterLeave"  @leave-cancelled="leaveCancelled"  :css="false"  >  </transition>  //JS中  beforeEnter(el) {  },  enter(el, done) {  done()  },  afterEnter(el) {  },  enterCancelled(el) {  },  beforeLeave(el) {  },  leave(el, done) {  done()  },  afterLeave(el) {  },  // leaveCancelled 只用于 v-show 中  leaveCancelled(el) {  } |

**例6-6 使用JavaScript钩子**