# vue/cli

npm install -g @vue/cli

或者

yarn global add @vue/cli

安装之后，就可以在命令行中访问 vue 命令

验证它是否安装成功。

vue --version

@vue/cli 4.5.4

升级

npm update -g @vue/cli

## 快速原型开发

npm install -g @vue/cli-service-global

安装这个

创建文件test-vue/app.vue或App.vue

<template>

<h1>Hello!</h1>

</template>

进入test-vue

vue serve

使用 vue serve对单个 \*.vue 文件进行快速原型开发Instant Prototyping

也可以显式地指定入口文件：

vue serve MyComponent.vue

vue build

在生产环境模式下零配置构建一个 .js 或 .vue 文件

## 创建一个项目

vue create project1

然后提示有两种选项，vue2或者vue3

Vue CLI 4.5.6

Please pick a preset:

>Default ([Vue 2] babel, eslint)

Default (Vue 3 Preview) ([Vue 3] babel, eslint)

还有一个自己手动创建预设的Manually select features选项

最好用Vue2这个

用vue2创建后main.js

import Vue from 'vue'

import App from './App.vue'

Vue.config.productionTip = false

new Vue({

render: h => h(App),

}).$mount('#app')

用Vue3创建后的mian.js

import { createApp } from 'vue'

import App from './App.vue'

createApp(App).mount('#app')

升级

npm update -g @vue/cli

## 插件

Vue CLI 使用了一套基于插件的架构。

# vue

## 引入

新建index.html

引入

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

## Vue实例

然后新建<script>标签在里边写东西就行了

<body>

<div id="aaa">

{{ message }}

</div>

<script>

var app = new Vue({

el: '#aaa',

data: {

message: 'Hello Vue!'

},

methods:{

log: function(){

return this.message

}

}

})

</script>

</body>

控制台修改app.message="nihao"，页面动态响应

在文档中经常使用 vm (ViewModel 的缩写) 这个变量名命名 Vue 实例。

Vue()声明方法传入的参数叫做选项对象option object

new Vue()创建的是根

Vue 实例还暴露了一些有用的实例 property 与方法，可以引用Vue实例来访问

它们都有前缀 $，以便与用户定义的 property 区分开来。

控制台输入

app.$data

{ message: 'Hello Vue!' }

app.$el

<div id=​"app">​

Hello Vue!

​</div>​

## 生命周期钩子

new Vue({

created: function () {

console.log('a is: ' + this.a)

}

})

this指向Vue实例

## 声明式渲染

<span>Message: {{ msg }}</span>

<span v-once>这个将不会改变: {{ msg }}</span>

使用 v-once 指令执行一次性地插值，当数据改变时，插值处的内容不会更新

<span v-bind:title="message">

鼠标悬停几秒钟查看此处动态绑定的提示信息！

</span>

v-bind被称为指令directive

指令是带有 v- 前缀的特殊 attribute

这里的意思是将这个元素节点的 title attribute 和 Vue 实例的 message保持一致

v-bind 缩写

<a v-bind:href="url">...</a>

<a :href="url">...</a>

v-on 缩写

<a v-on:click="doSomething">...</a>

<a @click="doSomething">...</a>

## 计算属性

<p>Computed reversed message: "{{ reversedMessage }}"</p>

data: {

message: 'Hello'

},

computed: {

reversedMessage: function () {

return this.message.split('').reverse().join('')

}

}

显示Computed reversed message: "olleH"

计算属性的属性是一个getter函数

计算属性是基于它们的响应式依赖进行缓存的。所以reversedMessages存储在缓存中，只要message不变，以后每次访问reversedMessage马上出结果，不执行函数

* 侦听属性Watched Property

data: {

firstName: 'Foo',

lastName: 'Bar',

fullName: 'Foo Bar'

},

watch: {

firstName: function (val) {

this.fullName = val + ' ' + this.lastName

},

}

有一些数据需要随着其它数据变动而变动

只要firstName发生变化，则watch中firstName函数触发

经测试有多个firstName属性只触发最后一个

## 条件与循环

<p v-if="seen">aaaaaaa</p>

也可以用 v-else 添加一个“else 块”：

<h1 v-if="awesome">Vue is awesome!</h1>

<h1 v-else>Oh no 😢</h1>

v-else 元素必须紧跟在带 v-if 或者 v-else-if 的元素的后面，否则它将不会被识别。

<li v-for="todo in todos">

{{ todo.text }}

</li>

<li v-for="(item, index) in items"></li>

可以遍历数组和对象

<div v-for="(value, name) in object"></div>

可以提供第二个参数为property 名称 (也就是键名)

最多三个参数

<div v-for="(value, name, index) in object"></div>

为了给 Vue 一个提示，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，你需要为每项提供一个唯一 key attribute：

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

请用字符串或数值类型的值作为key

v-for 也可以接受整数。在这种情况下，它会把模板重复对应次数。

<span v-for="n in 10">{{ n }} </span>

## 事件处理

<button v-on:click="reverseMessage">反转消息</button>

## 表单

Vue 还提供了 v-model 指令，它能轻松实现表单输入和应用状态之间的双向绑定。

<input v-model="message">

Vue.component('todo-item', {

template: '<li>这是个待办项</li>'

})

var app = new Vue(...)

## 组件

组件本质上是一个拥有预定义选项的一个 Vue 实例。

组件是可复用的 Vue 实例

* 注册组件

<div id="app">

<item></item>

<item></item>

</div>

Vue.component('item', {

template: '<li>这是个待办项</li>'

})

var app = new Vue({

el: '#app'

})

Vue.component注册组件

之后在Vue实例创建的DOM元素节点中可以创建任意数量的item组件。作为自定义元素使用

此例中渲染的组件都是重复的，渲染相同的文本

<div id="app">

<item v-for="a in todo\_list"

v-bind:todo="a"

v-bind:key="a.id"></item>

</div>

<script>

Vue.component('item', {

template: '<li>{{todo.message}}</li>',

props: ['todo']

})

var app = new Vue({

el: '#app',

data: {

todo\_list: [

{id: 0, message: 'nihao'},

{id: 1, message: '世界'},

]

}

})

</script>

在声明组件时加入props属性，使组件能够接受一个来自父作用域的变量todo

父作用域创建组件时，将组件的todo属性绑定了父作用域的变量a

这样，在父作用域中a改变，组件接收的todo属性也改变，形成数据流

v-for需要提供一个key

<div id="nihao">

<countbutton></countbutton>

</div>

Vue.component('countbutton', {

template:'<button v-on:click="click\_count">{{count}}</button>',

data: function(){

return {

count: 0

}

},

methods: {

click\_count: function(){

this.count++;

}

}

})

var vm = new Vue({

el: '#nihao'

})

组件中有template、data属性

组件一般有template、data、methods属性

组件名countbutton只能全小写或者count-button，不能写成countButton！坑了我一上午

组件的data并不是直接提供一个对象，而必须是一个函数

# BootstrapVue

BootstrapVue官方文档里有很多使用像ml-2, py-1之类的class，这些是帮助控制padding、margin等样式的Bootstrap v4.5 utility classes

全局box-sizing设置为了content-box

## 安装

vue create project1

用>Default ([Vue 2] babel, eslint)创建

进入项目目录

npm install bootstrap-vue

把main.js添加以下

import Vue from 'vue'

import { BootstrapVue, BootstrapVueIcons } from 'bootstrap-vue'

import 'bootstrap/dist/css/bootstrap.css'

import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.use(BootstrapVue)

Vue.use(BootstrapVueIcons)

至今不知道如何在选择Default (Vue 3 Preview) ([Vue 3] babel, eslint)创建项目后引入bootstrap。奥bootstrap3还没有正式发布

## card

<div>

<b-card title="Card title" sub-title="Card subtitle">

<b-card-text>A second paragraph of text in the card.</b-card-text>

<a href="#" class="card-link">Card link</a>

<b-link href="#" class="card-link">Another link</b-link>

</b-card>

</div>

可以添加title和sub-title属性

给a标签或b-link标签添加card-link类，成为卡片中的link

### b-card-body

<div>

<b-card no-body>

<b-card-body>

<b-card-title>Card Title</b-card-title>

</b-card-body>

<b-card-body>

<a href="#" class="card-link">Card link</a>

</b-card-body>

</b-card>

</div>

<b-card>的基本构造块是<b-card-body>，<b-card-body>有padding

默认情况下<b-card>的内容直接放置在<b-card-body>里，<b-card-body>不需要显示声明

激活<b-card>组件的no-body属性，则<b-card>内容不放在<b-card-body>里，padding也不存在

## Layout and Grid System

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Breakpoints | Extra small | Small | Medium | Large | Extra Large |
|  | <576px | >=576px | >=768px | >=992px | >=1200 |
| Container最大宽度 | auto | 540px | 720px | 960px | 1140px |
| 属性名 | cols="\*" | sm="\*" | md="\*" | lg="\*" | xl="\*" |
| Offset | offset="\*" | offset-sm="\*" | offset-md="\*" | offset-lg="\*" | offset-xl="\*" |

* 经测试，变换由视窗大小决定，和父容器宽度无关

Grid System在flexbox基础上实现

<b-container>

<b-row>

<b-col>1 of 3</b-col>

<b-col>2 of 3</b-col>

<b-col>3 of 3</b-col>

</b-row>

</b-container>

* 平均分成3列
* <b-container>有不同级别的响应式的像素宽度，<b-container fluid>总是占满一行

其实改的是CSS max-width属性，不是width

* 只有column可以是b-row的直接子元素

<b-row>

<b-col cols="2">1 of 3</b-col>

<b-col cols="4">2 of 3</b-col>

<b-col cols="6">ffff</b-col>

</b-row>

* cols规定列宽度，占12格中的几个。所有cols属性值加起来应该是12

<b-row>

<b-col sm="3" xl="9">1</b-col>

<b-col sm="9" xl="3">2</b-col>

</b-row>

* <b-col>有sm, md, lg, xl属性，可以表示在不同breakpoint情况下占多少格子。
* 上面这种写法，sm，md，lg时第一个<b-col>都占3格（虽然没有明确说明）
* 即sm="3"表示在视窗尺寸>=sm时占3格
* cols也可以和sm md lg xl配合使用。<b-col cols="3" xl="9">1</b-col>表示在sm md lg的屏幕占3格，在xl的屏幕占9格。

<b-row no-gutters>

* 移除<b-row>默认的margin和<b-col>默认padding
* <b-row>有默认的-15px的margin-right和margin-left，<b-col>有默认的15px的padding-right和padding-left。所以每个<b-col>之间有间隔，但是和容器的边和紧贴的
* columns的padding称为gutter沟壑

<b-container fluid="md">

100% wide until small breakpoint

</b-container>

在extra small, small区间是<b-container>，在medium, large, extra large区间是<b-container fluid>

<b-container class="bv-example-row">

<b-row>

<b-col>Column</b-col>

<b-col>Column</b-col>

<div class="w-100"></div>

<b-col>Column</b-col>

<b-col>Column</b-col>

</b-row>

</b-container>

* 创造两行两列大小相等的四个格子，中间插入一个<div class="w-100"></div>即可
* w-100代表width: 100%
* 当然写两个b-row也能实现，而且我觉得更好

<b-container class="bv-example-row">

<b-row>

<b-col>1 of 3</b-col>

<b-col cols="12" md="auto">Variable width content</b-col>

<b-col col lg="2">3 of 3</b-col>

</b-row>

</b-container>

* 设置成auto表示本身的宽度。在这里表示Variable width content这段话的宽度
* 像<b-col>什么都没有的自动延伸缩短
* 所以md="auto"就应该和<b-col>这种什么都不带的配合使用

### Stacked to horizontal

<b-container class="bv-example-row">

<b-row>

<b-col sm="8">col-sm-8</b-col>

<b-col sm="4">col-sm-4</b-col>

</b-row>

<b-row>

<b-col sm>col-sm</b-col>

<b-col sm>col-sm</b-col>

<b-col sm>col-sm</b-col>

</b-row>

</b-container>

* 只加sm="\*"或者sm属性，在极小的屏幕下变成一列好几行，在sm以上屏幕又变回好几列
* 所以在设置sm的同时相当于设置了cols="12"，让格子在超小屏幕下占满一整行
* 而如果不设置sm，格子即使在超小屏幕下都是平分屏幕
* 其实就是在设置sm, md, lg, xl任何一个时同时设置了cols="12"

### Ordering columns

<b-row class="mb-3">

<b-col>First in DOM, no order applied</b-col>

<b-col order="5">Second in DOM, with a larger order</b-col>

<b-col order="6">Third in DOM, with an order of 1</b-col>

</b-row>

可以用order属性调整一行中列的顺序

### Offsetting columns

<b-container fluid>

<b-row>

<b-col md="3" offset-md="3">md="3" offset-md="3"</b-col>

<b-col md="3" offset-md="3">md="3" offset-md="3"</b-col>

</b-row>

</b-container>

offset-md调整偏移

偏移量+每列占格子数=12

这样设置，在屏幕小于md时格子就没有偏移了

### Row columns

<b-row cols="2">

<b-col>Column</b-col>

<b-col>Column</b-col>

</b-row>

可以将cols属性赋给b-row，表示这一行不再有默认12列，而是只有两列，

<b-row cols="1" cols-sm="2" cols-md="4" cols-lg="6">

<b-col>Column</b-col>

<b-col>Column</b-col>

<b-col>Column</b-col>

<b-col>Column</b-col>

<b-col>Column</b-col>

<b-col>Column</b-col>

</b-row>

## Table

<b-table striped hover :items="items"></b-table>

<script>

export default {

data() {

return {

items: [

{ age: 40, first\_name: 'Dickerson', last\_name: 'Macdonald' },

{ age: 21, first\_name: 'Larsen', last\_name: 'Shaw' },

{ age: 89, first\_name: 'Geneva', last\_name: 'Wilson' },

]

}

}

}

</script>

列标题为Age、First Name、Last Name

可以写成firstName、first\_name、first-name，都自动转换成First Name

items是一个数组，每个数组由一行记录组成。记录是一个对象，key就是列头

{

age: 89,

first\_name: 'Geneva',

last\_name: 'Wilson',

\_rowVariant: 'danger'

},

整行变颜色

{

age: 40,

first\_name: 'Thor',

last\_name: 'MacDonald',

\_cellVariants: { age: 'info', first\_name: 'warning' }

},

单元格变颜色

### Fields

<b-table striped hover :items="items" :fields="fields"></b-table>

fields: ['first\_name', 'last\_name', 'age'],

items: [

{ isActive: true, age: 40, first\_name: 'Dickerson', last\_name: 'Macdonald' },

{ isActive: false, age: 21, first\_name: 'Larsen', last\_name: 'Shaw' },

{ isActive: false, age: 89, first\_name: 'Geneva', last\_name: 'Wilson' },

{ isActive: true, age: 38, first\_name: 'Jami', last\_name: 'Carney' }

]

fields决定要显示的字段，和字段顺序

fields: [

{

key: 'last\_name',

sortable: true

},

{

key: 'age',

label: 'Person age',

sortable: true,

variant: 'danger'

}

],

也可以是一个object组成的数组，决定每列是否可以排序，每列的颜色

variant就可以控制列标题样式啦

### Primary key

不知道是干啥的

### Table style options

striped 条纹

bordered 边框

borderless 没有上下边框线

outlined外边框

small 密集型

hover 鼠标移入变灰

dark 黑底白字

head-variant 标题行主题

table-variant 表格主题

### Row styling and attributes

设置单独一行的颜色

<b-table :items="items" :fields="fields" :tbody-tr-class="rowClass"></b-table>

<script>

export default {

data() {

return {

fields: ['first\_name', 'last\_name', 'age'],

items: [

{ age: 40, first\_name: 'Dickerson', last\_name: 'Macdonald', status: 'awesome' },

{ age: 21, first\_name: 'Larsen', last\_name: 'Shaw' },

{ age: 89, first\_name: 'Geneva', last\_name: 'Wilson' }

]

}

},

methods: {

rowClass(item, type) {

if (!item || type !== 'row') return

if (item.status === 'awesome') return 'table-success'

}

}

}

</script>

tbody-tr-class属性引用一个函数，如果函数返回一个字符串table-success，则这一行带有table-success这个class

函数有两个参数，item和type。item就是这一行的引用，可以用item.first\_name引用字段。type分为'row'、'row-details'、'row-top' 、'row-bottom' 、'table-busy'五种类型

### Responsive tables

<b-table responsive :items="items"></b-table>

加responsive属性，不知道有什么用

Using props responsive and fixed together will **not** work as expected. Fixed table layout uses the first row (table header in this case) to compute the width required by each column (and the overall table width) to fit within the width of the parent container — without taking cells in the <tbody> into consideration — resulting in table that may not be responsive. To get around this limitation, you would need to specify widths for the columns (or certain columns) via one of the following methods:

* Use <col> elements within the [table-colgroup slot](https://bootstrap-vue.org/docs/components/table#table-colgroup) that have widths set (e.g. <col style="width: 20rem">), or
* Wrap header cells in <div> elements, via the use of [custom header rendering](https://bootstrap-vue.org/docs/components/table#header-and-footer-custom-rendering-via-scoped-slots), which have a minimum width set on them, or
* Use the thStyle property of the [field definition object](https://bootstrap-vue.org/docs/components/table#field-definition-reference) to set a width for the column(s), or
* Use custom CSS to define classes to apply to the columns to set widths, via the thClass or class properties of the [field definition object](https://bootstrap-vue.org/docs/components/table#field-definition-reference).

### Stacked tables

变成只有两列，像栈一样的表格

### Table caption

说明性文字

<b-table :items="items" :fields="fields">

<template v-slot:table-caption>This is a table caption.</template>

</b-table>

<b-table :items="items" :fields="fields" caption-top>

<template v-slot:table-caption>Table caption at the top.</template>

</b-table>

放到上方

### Table busy state

<b-table :items="items" :busy="isBusy" class="mt-3" outlined>

<template v-slot:table-busy>

<div class="text-center text-danger my-2">

<b-spinner class="align-middle"></b-spinner>

<strong>Loading...</strong>

</div>

</template>

</b-table>

### Custom data rendering

* Scoped field slots作用域插槽

<b-table :fields="fields" :items="items">

<template v-slot:cell(index)="data1">

{{ data1.index + 1 }}

</template>

<template v-slot:cell(nameage)="data">

{{ data.item.name.first }} is {{ data.item.age }} years old

</template>

<template v-slot:cell()="data">

<i>{{ data.value }}</i>

</template>

</b-table>

<script>

export default {

data() {

return {

fields: [

'index',

{ key: 'name', label: 'Full Name' },

'age',

{ key: 'nameage', label: 'First name and age' }

],

items: [

{ name: { first: 'John', last: 'Doe' }, sex: 'Male', age: 42 },

{ name: { first: 'Jane', last: 'Doe' }, sex: 'Female', age: 36 }

]

}

}

}

</script>

自定义单元格内容

<template v-slot:cell(index)="data1">

{{ data1.index + 1 }}

</template>

* v-slot:cell(index)表示这是index这列的单元格的样式。index是一个列标题，在fields中定义了
* ="data1"这个是随便起的名字，在这个v-slot中data1是这一行数据的引用
* 可以用data1.index得到索引
* data1.item得到这一行的数据
* data1.value得到这个字段在记录里的值，比如得到这条记录index属性的数据。但是每条记录都没有index的属性，这个index列就是为了以后自定义使用的，所以data1.value取不到任何值。只有在列标题和item的属性一致时，data1.value才能取到值。比如v-slot:cell(age)="data2"这个data2.value可以取到值
* <template v-slot:cell()="data">匹配所有作用域的字段
* Formatter callback格式化回调函数

fields里：

{

key: 'name',

label: 'Full Name',

formatter: 'fullName'

},

methods里：

fullName(value) {

return `${value.first} ${value.last}`

}

所以显示的东西就经过格式化处理了

formatter: value => {

return value.charAt(0).toUpperCase()

}

也可以直接是一个箭头函数