# 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')

## 插件

Vue CLI 使用了一套基于插件的架构。

# 安装

创建一个HTML文件，通过script标签引入

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

vue的开发环境版本，体积较大，包含了有帮助的命令行警告

jsdelivr提供所有npm托管项目的cdn

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

vue的生产环境版本，体积较小

# Helloworld

<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
* Vue()声明方法传入的参数叫做选项对象option object
* 一个 Vue 应用会将其挂载到一个 DOM 元素上，然后对其进行完全控制。此例中app通过el: "#aaa"挂载到aaa div元素上
* Vue.js允许采用简洁的模板语法来声明式地将数据渲染进DOM，这种机制称为声明式渲染。即在Vue()中定义data.message，在DOM中使用{{message}}进行引用

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>​

# 指令

绑定元素的attribute

<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

<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

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>

# 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>

遍历数组

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

遍历对象，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>

# 生命周期钩子

new Vue({

created: function () {

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

}

})

this指向Vue实例

# 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

# 事件处理

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

# 表单

v-model 指令实现表单输入和应用状态之间的双向绑定。

<input v-model="message">

v-model本质是语法糖

* text 和 textarea 元素使用 value property 和 input 事件；
* checkbox 和 radio 使用 checked property 和 change 事件；
* select 字段将 value 作为 prop 并将 change 作为事件

事件发生时，调用事件处理函数将绑定的data修改为对应的property

## 复选框

单个复选框

<input type="checkbox" id="checkbox" v-model="checked">

checkbox没有value属性时，checked为true或false

<input type="checkbox" id="jack" value="Jack" v-model="checkedNames">

checkbox有value属性时，checkedNames应被初始化为一个数组，当勾选Jack, Bob时值为['Jack', 'Bob']

## 单选框

<input type="radio" id="one" value="One" v-model="picked">

<input type="radio" id="two" value="Two" v-model="picked">

picked应被初始化为一个空字符串。勾选one时picked值为'One'

## 下拉菜单

<select v-model="selected">

<option>C</option>

<option value="optionA">A</option>

<option>B</option>

</select>

selected应被初始化为一个空串。

当option没有value时，选中此option后selected为<option>标签里的字符串，比如B，C

当option有value时，选中此option后selected为value值，比如optionA

下拉菜单还可以多选。

<select v-model="selected" multiple>

按住ctrl多选，selected为一个数组

## 值绑定

value还可以绑定非字符串

<input id="input1" type="radio" v-bind:value="a1" v-model="m1" />

<input id="input2" type="radio" v-bind:value="a2" v-model="m1" />

a1，a2可以是对象

所以m1也有可能是对象

## 修饰符

* .number

自动将用户的输入值转为数值类型

<input v-model.number="age" type="number">

* .trim

自动过滤用户输入的首尾空白字符

<input v-model.trim="msg">

# 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

有些 HTML 元素，诸如 <ul>、<ol>、<table> 和 <select>，对于哪些元素可以出现在其内部是有严格限制的。而有些元素，诸如 <li>、<tr> 和 <option>，只能出现在其它某些特定的元素内部。使用这些有约束条件的元素时遇到一些问题，例如

<table >

<item></item>

</table>

这个自定义组件 <item> 会被作为无效的内容提升到外部，并导致最终渲染结果为

<li>giao</li>

<table></table>

特殊的 is attribute 给了我们一个变通的办法：

<table>

<tr is="item"></tr>

</table>

这样就渲染正常了

如果我们从以下来源使用模板的话，这条限制是不存在的：

* 字符串 (例如：template: '...')
* 单文件组件 (.vue)
* <script type="text/x-template">

没看懂这句话

## ­­­­­­­­注册组件

### 全局注册

<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
* 注册组件必需在创建Vue实例之前。即Vue.component()必须在new Vue()之前，否则无法正确渲染
* 组件必需在实例中使用，即<item>元素必需在<div id="app">中，放在外边无效
* 组件名countbutton只能全小写或者count-button，不能写成countButton！坑了我一上午
* 强烈推荐遵循 W3C 规范中的自定义组件名，字母全小写且必须包含一个连字符

Vue.component('countbutton', {

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

data: function(){

return {

count: 0

}

},

methods: {

click\_count: function(){

this.count++;

}

}

})

* 组件是可复用的Vue实例，所以和new Vue()接收相同的选项，有template、data、computed、生命周期钩子等属性。
* el属性时new Vue()特有的，而Vue.component的name参数也相当于el属性了
* 组件的data并不是对象，而必须是一个函数

template: `

<div class="blog-post">

<h3>{{ post.title }}</h3>

<div v-html="post.content"></div>

</div>

`

模板字符串语法，包含多行HTML

### 局部注册

var ComponentA = { /\* ... \*/ }

var ComponentB = { /\* ... \*/ }

然后在 components 选项中定义你想要使用的组件：

new Vue({

el: '#app',

components: {

'component-a': ComponentA,

'component-b': ComponentB

}

})

对于 components 对象中的每个 property 来说，其 property 名就是自定义元素的名字，其 property 值就是这个组件的选项对象。

局部注册的组件在其子组件中不可用。componentB中不能使用componentA

## 自定义attribute

<item v-bind:to-do="a"></item>

Vue.component('item', {

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

props: ['to-do']

})

var app = new Vue({

el: '#app',

data: {

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

}

})

* 在声明组件时加入props属性，使组件能够接受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。a可以是任何类型
* 这样，在父作用域中a改变，组件的todo属性也改变，形成数据流
* 不应该在一个子组件内部改变 prop，虽然这样做是可以的。会产生迷惑的数据流
* props里的属性也只能是全小写或者用-连接，坑了我一晚上
* 这是因为HTML中的attribute名是大小写不敏感的，所以浏览器会把所有大写字符解释为小写字符。
* 在DOM模板中，作为attribute需要使用to-do="hello!"，在JS中，需要使用toDo
* 传入一个对象的所有 property

将一个对象的所有property拆解后一一传入，而不是传入整个对象

对于一个给定的对象 post：

post: {

id: 1,

title: 'My Journey with Vue'

}

下面的模板：

<blog-post v-bind="post"></blog-post>

等价于：

<blog-post

v-bind:id="post.id"

v-bind:title="post.title"

></blog-post>

* 试图改变prop

上面说到不应该在组件内部改变prop的值。

如果这个 prop 用来传递一个初始值；这个子组件接下来希望将其作为一个本地的数据来使用。在这种情况下，最好定义一个本地的 data property 并将这个 prop 用作其初始值：

props: ['initialCounter'],

data: function () {

return {

counter: this.initialCounter

}

}

这样就可以改变counter而initialCounter不受影响

当然组件的data是一个函数需要注意

### prop验证

验证prop是否为预期的类型，如果不是，则产生一个控制台警告Invalid prop/Missing required prop

开发一个供别人使用的组件时很有用

Vue.component('my-component', {

props: {

propA: Number,

propB: [String, Number]

}

})

props 是一个带有验证需求的对象，而不是一个字符串数组。

`null` 和 `undefined` 会通过任何类型验证

[String, Number]表示propB是String或Number

可以指定的类型有String, Number, Boolean, Array, Object, Function, Promise, Date, Symbol和其他自定义构造器

必填的字符串

propC: {

type: String,

required: true

},

带有默认值的数字

propD: {

type: Number,

default: 100

},

带有默认值的对象

propE: {

type: Object,

default: function () {

return { message: 'hello' }

}

},

对象或数组默认值必须从一个工厂函数获取

自定义验证函数

propF: {

validator: function (value) {

return ['success', 'warning', 'danger'].indexOf(value) !== -1

}

}

### 非 Prop 的 Attribute

定义一个组件：

template: '<input type="text" placeholder="this is com1">'

在创建组件时，传入type属性。这个type属性没有在组件中通过Props声明

<com1 type="radio"></com1>

然后这个type attribute会自动添加到组件的根元素上

所以渲染出的input成了单选框

可以利用这个特性，传入一些没有通过props声明的attribute，让组件也能接收到

比较有用的是data-date-picker="activated"这类带data-前缀的attribute

传入class和style attribute不会直接替换根元素原本的class和style，而是会合并起来

## 监听子组件事件

父组件中声明enlargeAll方法，和fontSize属性

data: {

fontSize: 1,

},

methods: {

enlargeAll: function(){

this.fontSize+=1

}

}

给子组件todo-item的enlarge事件绑定事件处理函数enlargeAll方法，同时字体大小关联fontSize

<todo-item

@enlarge="enlargeAll"

:style="{fontSize: fontSize+'em'}"

></todo-item>

子组件中，由于访问不到enlargeAll方法，所以不能直接用子组件中的元素绑定enlargeAll方法

所以子组件中的元素想要触发enlarge事件，运行enlargeAll方法，就需要

<button @click="$emit('enlarge')">放大放大</button>

* 子组件的子元素触发click事件🡪子组件触发enlarge事件🡪调用父组件的enlargeAll方法
* @click="$emit('enlarge')"相当于在事件处理函数中写this. $emit('enlarge')，是组件层面的触发事件
* enlarge是一个自定义事件。自定义事件命名应遵循kebab-case，即全小写+连字符-
* 在组件上之间设置原生事件，如<todo-item @click="f1">是无效的。组件上只能设置自定义事件，绑定父组件中的事件处理函数，由组件的子元素通过emit触发
* 抛出参数

<button @click="$emit('enlarge', 1)">放大1倍</button>

enlargeAll: function(num){

this.fontSize=num

}

$emit第二个参数抛出一个值，作为事件处理函数的参数。时间处理函数第1个参数接收这个值

## v-model

一个组件上的 v-model 默认会利用名为 value 的 prop 和名为 input 的事件

回想一下，input元素上的v-model会将value属性绑定到一个变量m，给input事件绑定事件处理函数m=$event.target.value

如果在组件中使用v-model

Vue.component('custom-input', {

props: ['value'],

template: `

<input

v-bind:value="value"

v-on:input="$emit('input', $event.target.value)">`

})

组件声明时，需要将组件<input>元素的value attribute绑定到一个名叫 value 的 prop 上

在其 input 事件被触发时，将新的值通过自定义的input事件抛出

<custom-input v-model="searchText" @input="f1"></custom-input>

使用组件时像一个普通input元素一样使用

但是像单选框、复选框等类型的输入控件可能会将 value attribute 用于不同的目的。model 选项可以用来避免这样的冲突：

Vue.component('base-checkbox', {

model: {

prop: 'checked',

event: 'change'

},

props: {

checked: Boolean

},

template: `

<input

type="checkbox"

v-bind:checked="checked"

v-on:change="$emit('change', $event.target.checked)">`

})

可以使用不同于value的attribute和不同于input的自定义事件

可以看到组件的template只有一个input。这样做的目的可能仅仅为了给一个input取一个更加语义化的名字？

## 原生事件

如果组件只是一个input，则

<com1 v-model="nihao" v-on:focus.native="m1"></com1>

可以监听focus事件。相当于监听了组件内部input的focus事件

但如果组件形如

<label>

{{ label }}

<input v-bind:value="value" v-on:input="$emit('input', $event.target.value)">

</label>

就不能直接使用.native了

至于该怎么办，我也没看懂

## .sync修饰符

前面说到不建议在子组件内部修改props，推荐使用这样的方式

<text-document

v-bind:title="title1"

v-on:update:title="title1 = $event"

></text-document>

等价于

<text-document v-bind:title.sync="title1"></text-document>

子组件内部通过调用this.$emit('update:title', newTitle)触发update事件

<text-document v-bind.sync="doc"></text-document>

当doc是一个对象时，也可以这样用

这样会把 doc 对象中的每一个property都作为一个独立的 prop 传进去，然后各自添加用于更新的 v-on 监听器。

## 插槽

像这样合成组件

<navigation-link url="/profile">

Your Profile

</navigation-link>

<navigation-link>组件模板为

<a v-bind:href="url" class="nav-link">

<slot></slot>

</a>

这样，Your Profile就插入进了<navigation-link>组件中

<slot>元素叫承载分发内容的出口。

Your Profile可以是HTML，甚至是另外的组件

### 后备内容

Fallback Content

<button type="submit">

<slot>Submit</slot>

</button>

slot中有submit作为后备（默认）内容

在一个父级组件中使用 <submit-button> 并且不提供任何插槽内容时：

<submit-button></submit-button>

后备内容“Submit”将会被渲染：

<button type="submit">

Submit

</button>

### 具名插槽

有时需要多个插槽。

<slot> 元素有一个特殊的 attribute: name，可以用来定义额外的插槽

声明组件：

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

<footer>

<slot name="footer"></slot>

</footer>

</div>

一个不带 name 的 <slot> 出口会带有隐含的名字“default”。

在向具名插槽提供内容的时候，我们可以在一个 <template> 元素上使用 v-slot 指令，并以 v-slot 的参数的形式提供其名称

创建组件：

<base-layout>

<template v-slot:header>

<h1>Here might be a page title</h1>

</template>

<p>A paragraph for the main content.</p>

<template v-slot:footer>

<p>Here's some contact info</p>

</template>

</base-layout>

现在 <template> 元素中的所有内容都将会被传入相应的插槽。

任何没有被包裹在带有 v-slot 的 <template> 中的内容都会被视为默认插槽的内容，如上例中的<p>元素

等价于

<template v-slot:default>

<p>A paragraph for the main content.</p>

</template>

v-slot 只能添加在 <template> 上

* 具名插槽的缩写

跟 v-on 和 v-bind 一样，v-slot 也有缩写，即把参数之前的所有内容 (v-slot:) 替换为字符 #。例如 v-slot:header 可以被重写为 #header：

<base-layout>

<template #header>

<h1>Here might be a page title</h1>

</template>

</base-layout>

### 作用域插槽

<current-user> 组件模板

<slot>{{ data1 }}</slot>

{{ data1 }}是后备内容。父级组件不提供插槽内容时显示

父级组件创建current-user：

<current-user>

{{ data1 }}

</current-user>

然而上述代码不会正常工作，因为只有 <current-user> 组件可以访问到user，而我们提供的内容是在父级渲染的。

为了让data1在父级的插槽内容中可用，可以将data1作为<slot>元素的一个attribute 绑定上去：

<slot v-bind:data1="data1">可以在父级作用域使用{{ data1 }}</slot>

绑定在 <slot> 元素上的 attribute 被称为插槽 prop。

目前测试貌似data1只能绑定名叫data1的attribute，其他的名称不行

现在在父级作用域中，可以使用带值的 v-slot 定义插槽 prop 的名字：

<current-user>

<template v-slot:default="slotProps">

{{ slotProps.data1 }}

</template>

</current-user>

slotProps是包含所有插槽prop的对象

可以在<slot>元素中定义v-bind:data1="data1"，v-bind:data2="data2"多个插槽prop，但还是推荐只定义一个，以对象的形式，把data1，data2都视为对象中的属性

当只有默认插槽没有具名插槽时，可以把 v-slot 直接用在组件上：

<current-user v-slot:default="slotProps">

{{ slotProps.data1 }}

</current-user>

或者

<current-user v-slot="slotProps">

{{ slotProps.data1 }}

</current-user>

### 解构 插槽Prop

组件定义：

<slot name="header" v-bind:data1="data1">{{data1}}</slot>

data1: {

v1: 1122,

v2: 3322

}

父级作用域创建组件：

<com1>

<template v-slot:header="slotProps">

<h1>this is a named slot: Header {{slotProps}}</h1>

</template>

</com1>

显示{ "data1": { "v1": 1122, "v2": 3322 } }

利用ES2015的解构赋值特性，传入具体的插槽prop

<com1 v-model="nihao" @input="m1">

<template v-slot:header="{data1}">

<h1>this is a named slot: Header {{data1}}</h1>

</template>

</com1>

显示{ "v1": 1122, "v2": 3322 }

这样就不用slotProps.data1.v1，直接data1.v1就可以

相当于经过一步赋值

let { data1 }={ data1: { "v1": 1122, "v2": 3322 } }

### 动态插槽名

不知道在干啥

## 动态组件

## 异步组件

# 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