**VUEJS**

1. **Mục lục**

[**B.** **Khái niệm cơ bản** 4](#_Toc45873777)

[**I.** **Vuejs là gì? Cài đặt + Các tool hỗ trợ + chương trình đầu tiên** 4](#_Toc45873778)

[**1.** **Vue.js là gì?** 4](#_Toc45873779)

[**2.** **Kiến thức cần có để học vue.js** 5](#_Toc45873780)

[**3.** **Nhúng core vue** 5](#_Toc45873781)

[**4.** **Vue devtool (Tham khảo)** 5](#_Toc45873782)

[**5.** **Hello world!** 7](#_Toc45873783)

[**II.** **Instance trong Vue.js** 7](#_Toc45873784)

[**1.** **Khởi tạo** 7](#_Toc45873785)

[**2.** **Constructor** 8](#_Toc45873786)

[**3.** **Thuộc tính và phương thức** 8](#_Toc45873787)

[**4.** **Instance Lifecycle Hook.** 10](#_Toc45873788)

[**III.** **Template Syntax trong Vue.js** 12](#_Toc45873789)

[**1.** **Interpolation** 12](#_Toc45873790)

[**a.** **Text** 12](#_Toc45873791)

[**b.** **Raw HTML** 14](#_Toc45873792)

[**c.** **Attributes** 15](#_Toc45873793)

[**d.** **Using JavaScript Expressions** 17](#_Toc45873794)

[**2.** **Directives** 18](#_Toc45873795)

[**a.** **Arguments** 19](#_Toc45873796)

[**b.** **Modifiers** 21](#_Toc45873797)

[**c.** **Filters** 22](#_Toc45873798)

[**3.** **Shorthands** 25](#_Toc45873799)

[**a.** **V-bind** 25](#_Toc45873800)

[**b.** **V-on** 25](#_Toc45873801)

[**IV.** **Method, computed property và watcher trong vue.js** 25](#_Toc45873802)

[**1.** **Method** 25](#_Toc45873803)

[**2.** **Computed property** 26](#_Toc45873804)

[**a.** **Cú pháp** 27](#_Toc45873805)

[**b.** **Computed vs Methods** 29](#_Toc45873806)

[**c.** **Getter and Setter** 31](#_Toc45873807)

[**3.** **Watcher** 34](#_Toc45873808)

[**a.** **Cú pháp** 34](#_Toc45873809)

[**b.** **Computed vs Watcher** 36](#_Toc45873810)

[**V.** **Class và style bindings trong Vue.js** 38](#_Toc45873811)

[**1.** **Classes binding** 38](#_Toc45873812)

[**a.** **Cú pháp kiểu Object** 38](#_Toc45873813)

[**b.** **Cú pháp kiểu Array** 41](#_Toc45873814)

[**2.** **Binding inline styles** 45](#_Toc45873815)

[**a.** **Kiểu Object** 45](#_Toc45873816)

[**b.** **Kiểu Array** 46](#_Toc45873817)

[**VI.** **Conditional rendering trong Vue.js** 47](#_Toc45873818)

[**1.** **V-if** 47](#_Toc45873819)

[**2.** **V-show** 50](#_Toc45873820)

[**3.** **V-if vs v-show** 52](#_Toc45873821)

[**VII.** **List rendering trong Vue.js** 52](#_Toc45873822)

[**1.** **v-for.** 52](#_Toc45873823)

[**2.** **Template v-for.** 55](#_Toc45873824)

[**3.** **Object v-for.** 57](#_Toc45873825)

[**4.** **Range v-for.** 60](#_Toc45873826)

[**5.** **v-for with v-if.** 60](#_Toc45873827)

[**6.** **Array change detection.** 63](#_Toc45873828)

[**7.** **Displaying filtered or sorted results.** 64](#_Toc45873829)

[**VIII.** **Event handling trong Vue.js** 66](#_Toc45873830)

[**1.** **Listen to Events.** 66](#_Toc45873831)

[**2.** **Method Event Handlers.** 67](#_Toc45873832)

[**3.** **Event Modifiers.** 70](#_Toc45873833)

[**4.** **Key modifiers.** 71](#_Toc45873834)

[**5.** **Why listener in HTML.** 74](#_Toc45873835)

[**IX.** **Form input binding trong Vue.js** 74](#_Toc45873836)

[**1.** **Basic Usage.** 74](#_Toc45873837)

[**a.** **Text** 74](#_Toc45873838)

[**b.** **Textarea** 75](#_Toc45873839)

[**c.** **Checkbox** 76](#_Toc45873840)

[**d.** **Radio** 78](#_Toc45873841)

[**2.** **Value bindings.** 81](#_Toc45873842)

[**3.** **Modifiers.** 84](#_Toc45873843)

[**a.** **.lazy** 84](#_Toc45873844)

[**b.** **.number** 85](#_Toc45873845)

[**c.** **.trim** 85](#_Toc45873846)

[**4.** **v-model với component.** 86](#_Toc45873847)

[**X.** **Component trong Vue.js** 86](#_Toc45873848)

[**1.** **Component là gì?** 86](#_Toc45873849)

[**2.** **Using components** 87](#_Toc45873850)

[**a.** **Registration** 87](#_Toc45873851)

[**b.** **Local Registration** 88](#_Toc45873852)

[**c.** **DOM Template Parsing Caveats.** 89](#_Toc45873853)

[**d.** **Data must be a function** 91](#_Toc45873854)

[**e.** **Composing components** 94](#_Toc45873855)

[**3.** **Props.** 96](#_Toc45873856)

[**a.** **Passing Data with Props** 96](#_Toc45873857)

[**b.** **camelCase vs. kebab-case** 97](#_Toc45873858)

[**c.** **Dynamic Props** 98](#_Toc45873859)

[**d.** **Literal vs Dynamic** 99](#_Toc45873860)

[**e.** **One-Way Data Flow** 100](#_Toc45873861)

[**f.** **Prop Validation** 102](#_Toc45873862)

[**4.** **Non-prop Attributes.** 105](#_Toc45873863)

[**5.** **Custom Events.** 105](#_Toc45873864)

[**a.** **Using v-on** 105](#_Toc45873865)

[**b.** **Binding Native Events to Components.** 107](#_Toc45873866)

[**6.** **x-template.** 108](#_Toc45873867)

[**C.** **Transitions & Animation** 109](#_Toc45873868)

[**I.** **Enter/Leave & List Transitions** 109](#_Toc45873869)

[**1.** **Tổng quan** 109](#_Toc45873870)

[**2.** **Transitioning Single Elements/Components** 110](#_Toc45873871)

[**a.** **Transition Classes** 111](#_Toc45873872)

[**b.** **CSS Transitions** 112](#_Toc45873873)

[**c.** **CSS Animations** 113](#_Toc45873874)

[**d.** **Custom Transition Classes** 115](#_Toc45873875)

[**e.** **Using Transitions and Animations Together** 116](#_Toc45873876)

[**f.** **Explicit Transition Durations** 116](#_Toc45873877)

[**g.** **JavaScript Hooks** 117](#_Toc45873878)

[**3.** **Transitions on Initial Render** 120](#_Toc45873879)

[**4.** **Transitioning Between Elements** 121](#_Toc45873880)

[**5.** **Transitioning Between Components** 125](#_Toc45873881)

[**6.** **List Transitions** 126](#_Toc45873882)

[**a.** **List Entering/Leaving Transitions** 127](#_Toc45873883)

[**b.** **List Move Transitions** 129](#_Toc45873884)

[**c.** **Staggering List Transitions** 133](#_Toc45873885)

[**II.** **State Transitions** 141](#_Toc45873886)

[**1.** **Animating State with Watchers** 141](#_Toc45873887)

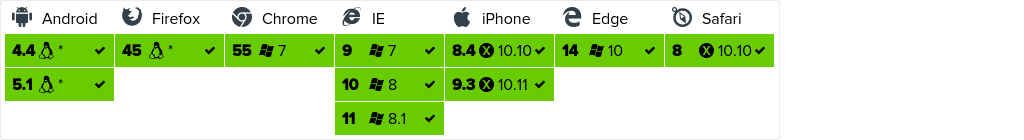
[**2.** **Dynamic State Transitions** 144](#_Toc45873888)

[**3.** **Organizing Transitions into Components** 144](#_Toc45873889)

[**4.** **Bringing Designs to Life** 147](#_Toc45873890)

1. **Khái niệm cơ bản**
2. **Vuejs là gì? Cài đặt + Các tool hỗ trợ + chương trình đầu tiên**
3. **Vue.js là gì?**

* Vue là một progressive framework dùng để xây dựng giao diện người dùng (UI).
* Core của Vue chỉ tập trung vào lớp view mà và nó rất dễ để làm quen hay tích hợp với các thư viện hoặc các project có sẵn. Hơn nữa, Vue lại chứa đựng một sức mạnh rất lớn trong việc xây dựng Single-Page Applications khi được kết hợp với công cụ build và các thư viện/component được xây dựng bởi cộng đồng.
* Vue.js cũng hỗ trợ hầu hết các trình duyệt hiện nay:



1. **Kiến thức cần có để học vue.js**

Vì vue.js là một Javascript Framework nên để học và làm việc với vue.js thì kiến thức về javascript là một điểu bắt buộc. Bên cạnh đó vue.js xử lý phần front-end nên cần có kiến thức về HTML và CSS.

Kết luận những kiến thức cần có để học vue.js:

* Javascript
* HTML
* CSS

1. **Nhúng core vue**

Đầu tiên để nhúng core của vue chúng ta sử dụng một trong 2 cách:

* Dẫn đến CDN

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

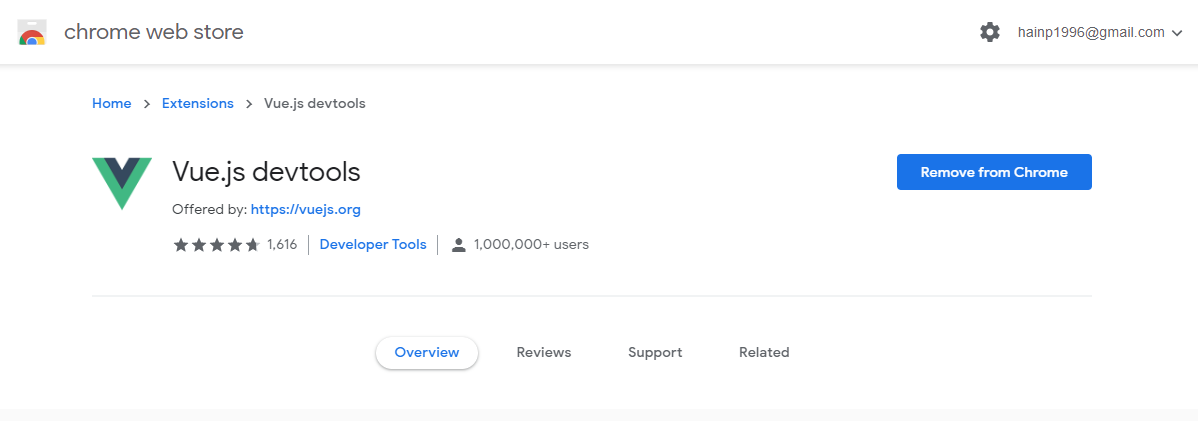
* Dẫn file local:

Ví dụ: <script src="/js/vue.js"></script>

Đối với Laravel Framework đã hỗ trợ tích hợp vue.js và chúng ta chỉ cần chạy câu lệnh: npm install

1. **Vue devtool (Tham khảo)**

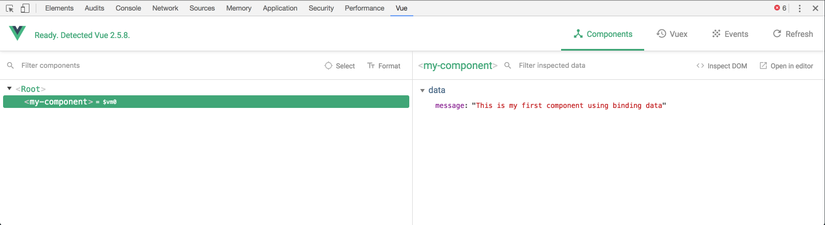
Vue devtool là một công cụ cực kì hữu hiệu trong quá trình phát triển ứng dụng sử dụng VueJS để có thể theo dõi quá trình thay đổi của dữ liệu trong toàn bộ ứng dụng VueJS.



Nếu chữ "V" (biểu tượng của Vue) sáng như kia là trang web hiện tại có thể sử dụng VueJS

https://images.viblo.asia/abedcdab-554a-4e53-b185-15f921aec6c3.png

Quay trở lại app của chúng ta, các bạn click chuột phải vào trình duyệt chọn Inspect, các bạn sẽ thấy 1 tab Vue, click vào tab này sẽ thấy xuất hiện như sau:



Ở đây các bạn có thể thế ứng dụng của chúng ta có một component cha là Root, đây cũng là component tổng cho bất kì ứng dụng nào khác, trong đó sẽ là các component do chúng ta tạo ra - ở đây là my-component.

Click vào my-component các bạn có thể thấy thuộc tính message trong data và giá trị của nó, còn khi click vào Root sẽ có các thông tin mà chúng ta thiết lập ở trong file app.js (data, methods, computed,...) nhưng hiện tại ở app.js chúng ta chưa có dữ liệu gì cả.

Chúng ta có thể chọn Select sau đó bấm vào bất kì phần tử nào trên màn hình để xem nó thuộc component nào.

1. **Hello world!**

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>test Vuejs</title>

    <link rel="stylesheet" href="">

</head>

<body>

    <div id="app">

        {{ message }}

    </div>

</body>

<script src="https://unpkg.com/vue/dist/vue.js"></script>

<script >

   var app= new Vue({

        el: '#app',

        data: {

                message:'Hello World!'

              }

  })

</script>

</html>

1. **Instance trong Vue.js**

Instance dịch ra tiếng việt thì nó có nghĩa là kể lại ví dụ, nhưng ở đây trong lập trình thì chúng ta có thể hiểu lại là nó mang một tính chất thể hiện lại đối tượng.

1. **Khởi tạo**

Thực hiện cú pháp để khởi tạo sau khi nhúng vue.js thành công:

var vm = new Vue({

// options

})

Trong đó options là nơi chứa các tùy chọn.

1. **Constructor**

Khi ta khởi tạo một instance Vue, ta cần phải vượt qua options có thể chứa các tùy chọn cho data, template, element, methods, callback...

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <p>Message = {{ message }}</p>

        <p>Phương thức getSite trả về: {{ getSite() }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: 'hello world'

            },

            methods: {

                getSite : function () {

                    return 'Hello World!'

                }

            }

        });

    </script>

</body>

</html>

1. **Thuộc tính và phương thức**

Trong vue.js ta có thể khai báo thuộc tính và phương thức vào trong data. Ví dụ như sau:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <p>Message = {{ message }}</p>

        <p>Phương thức getSite trả về: {{ getSite() }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                //thuộc tính

                message: 'hello world',

                // phương thức

                getSite : function () {

                    return 'Hello World!'

                }

            },

        });

    </script>

</body>

</html>

Nhưng bản thân phương thức trong khai báo trên thực ra cũng chỉ là một thuộc tính được viết dưới dạng 1 closure function. Và để khai báo một phương thức (được khuyên dùng) trong vue.js thì chúng ta sẽ khai báo trong scope methods với cú pháp tương tự như khai báo thuộc tính.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <p>Message = {{ message }}</p>

        <p>Phương thức initialVue trả về: <code>{{ initialVue() }}</code></p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: 'Hello world'

            },

            methods: {

                initialVue : function () {

                    return 'Function initial Vuew: ' + this.message;

                }

            }

        });

    </script>

</body>

</html>

1. **Instance Lifecycle Hook.**

*Instance lifecycle hook là một vòng đời hoạt động của vue.js, khi vue.js chạy đến một thời điểm nào đó trên vòng đời hoạt động thì sẽ có các scope event được gọi kèm theo đó (tương tự như magic methods trong PHP), các scope event này đã được định tên sẵn và bạn không thể thay đổi name của nó mà chỉ có thể override nó thôi.*



Trong vue.js có các scope event sau:

* beforeCreate - được gọi khi chúng ta khởi tạo vue.js và trước khi thực hiện tiến trình observe Data và init Events.
* created - được gọi khi tiến trình observe Data và init Events hoàn thành.
* beforeMount - được gọi ngay sau khi tiến trình render function hoàn tất.
* mounted - được gọi khi tiến trình replace el hoàn tất.
* beforeUpdate - được gọi khi data có sự thay đổi, và trước khi visualDOM re-rendered.
* updated - được gọi khi data đã được thay đổi.
* beforeDestroy - được gọi ngay trước khi vue instance được destroy.
* destroyed - được gọi khi vue instance đã được destroy.

1. **Template Syntax trong Vue.js**

Vue.js sử dụng các cú pháp mẫu trong HTML để bind và render DOM thành các tag HTML và tất cả các vue.js template này đều phải tuân thủ theo các tag HTML chuẩn.

1. **Interpolation**

Intercalation là một thuật ngữ trong Vue.js. Đây là quá trình thêm một văn bản, nội dung, attribute... Vào các thẻ HTML bằng Vue.js.

1. **Text**

Để bind một đoạn văn bản vào trong một tag HTML thì bạn sử dụng cú pháp sau:

{{ avariable }}

Trong đó, avariable là tên thuộc tính mà chúng ta đã khai báo ở vue instance.

Chú ý: Khi bind data dưới dạng này thì dữ liệu được bind sẽ luôn luôn ở dạng thô (tức là văn bản, giống hàm text trong jquery hay innerText trong javascript) mà không được append HTML

VD: sử dụng vue.js để bind data dưới dạng dữ liệu thô vào thẻ HTML.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 class="text-red" style="text-align: center">{{ text }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                text: 'Hello World!'

            },

        });

    </script>

</body>

</html>

Và trong cặp dấu {{}} thì có thể sử dụng các hàm của javascript.

VD: Bind dữ liệu ra tag HTML và đồng thời chuyển đổi data đó thành in hoa.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 class="text-red" style="text-align: center">{{ text.toUpperCase() }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                text: 'Hello World!'

            },

        });

    </script>

</body>

</html>

1. **Raw HTML**

Nếu như muốn hiển thị dữ liệu ra dưới dạng append cả HTML code (như hàm html trong jquery hoặc innerHTML trong javascript) thì có thể sử dụng cú pháp sau:

<tag v-html="data"></tag>

Trong đó:

* tag là các tag trong HTML.
* data là dữ liệu mà bạn muốn bind vào tag đó (dữ liệu này thường được khai báo trong data scope của vue.js).

VD: Bind data vào tag h1 đồng thời append các thẻ HTML vào tag h1.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 class="text-red" style="text-align: center" v-html="text"></h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                text: 'Hello <font color="orange">World!</font>'

            },

        });

    </script>

</body>

</html>

Tuy nhiên khi sử dụng cách này phải cẩn thận vì sẽ dẫn đến lỗi XSS.

1. **Attributes**

Để có thể thêm các attribute vào tag HTML bằng dữ liệu trong vue.js thì sử dụng cú pháp sau:

<tag v-bind:attributeName="data"></tag>

Trong đó:

* attributeName là tên của attribute mà bạn muốn thực hiện binding.
* data là data mà bạn đã thiết lập trong vue.js.

VD: Bind class name vào thẻ h1.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 v-bind:class="className" style="text-align: center" v-html="text"></h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                text: 'Hello <font color="orange">World!</font>',

                className: 'text-red',

            },

        });

    </script>

</body>

</html>

VD: Bind CSS style vào trong tag h1.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 v-bind:class="className" v-bind:style="styleData" v-html="text"></h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                text: 'Hello <font color="orange">World!</font>',

                className: 'text-red',

                styleData: 'text-align: center',

            },

        });

    </script>

</body>

</html>

1. **Using JavaScript Expressions**

Chúng ta có thể sử dụng code javascript trong cặp dấu {{}}, nhưng cần chú ý là chúng ta chỉ có thể sử dụng các scope đã được khai báo mặc định trong JavaScript thôi còn các biến hay object... mà người dùng tự định nghĩa thì sẽ không thể sử dụng được.

VD: Sử dụng các toán tử và câu lệnh điều kiện trong cặp dấu {{}}.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <p>Tuổi tiếp theo: {{ age + 1 }}</p>

        <p>Tuổi có là số chẵn: {{ age % 2 == 0 ? 'Chẵn' : 'Lẻ' }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                age : 22

            },

        });

    </script>

</body>

</html>

1. **Directives**

Directives trong vue.js là các attribute được thiết lập bằng các tiền tố v-. Giá trị của các attribute này thường là các biểu thức javascript duy nhất, chỉ trừ v-for (khái niệm này sẽ giải thích ở phần sau).

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body>

    <div id="app">

        <h1 v-if="publish" v-bind:class="className" v-bind:style="textCenter">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                className : "text-red",

                textCenter: "text-align: center",

                publish: true,

            },

        });

    </script>

</body>

</html>

1. **Arguments**

Trong một số các directives nó sẽ cho phép truyền tham số vào. Để truyền tham số vào các directive này thì chỉ cần ngăn giữa directives và tham số bằng dấu “:”

VD: Như v-bind attribute ở phần 1.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <img width="50%" v-bind:src="logoUrl">

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                logoUrl : "https://images6.alphacoders.com/102/thumb-1920-1020806.jpg"

            },

        });

    </script>

</body>

</html>

Hoặc cũng có thể thêm các event trên DOM bằng directives v-on trong Vue.js.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <p>Click vào button để xem kết quả</p>

        <button v-on:click="showAlert">Click</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            methods: {

                showAlert : function () {

                    alert('Bạn vừa click vào button');

                }

            }

        });

    </script>

</body>

</html>

1. **Modifiers**

Directives dạng này cho phép bạn định nghĩa một hành động đặc biệt khi thực hiện directives đó. Các modifiers này được ngăn cách với directives bởi dấu “.”

VD: Thêm sự kiên submit cho form và kèm theo đó là preventDefault.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <form action="https://facebook.com" v-on:submit.prevent="submitForm">

            <input type="text" name="name" placeholder="Nhập vào tên">

            <input type="submit" value="Gửi">

        </form>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            methods: {

                submitForm: function () {

                    alert('Bạn vừa submit form');

                }

            }

        });

    </script>

</body>

</html>

Ở trong ví dụ trên đã thêm sự kiện onsubmit cho form và đồng thời xác định modifier prevent cho form đó. Modifier này sau khi được render thì nó sẽ thành event.preventDefault()

1. **Filters**

Như đã nói ở trên thì trong cặp dấu {{}} thì chúng ta có thể sử dụng các biểu thức trong JavaScript, dựa vào điều này thì chúng ta cũng có thể sử dụng filter

VD:

{{ name | uppercase }}

Bên cạnh điều này thì vue.js còn hỗ trợ filter trong các directive v-bind và cũng sử dụng cú pháp tương tự.

VD:

<h1 v-bind:class="OldClass | newClass"></h1>

Filter trong v-bind chỉ đươc hỗ trợ từ phiên bản vue.js từ phiên bản 2.1.0. Nhưng với các biến đổi phức tạp hơn thì nên sử dụng computed

VD: Viết một filter để chuyển đổi chữ thường thành chữ hoa trong Vue.js.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ name | strUpperCase }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'vu thanh tai'

            },

            filters : {

                strUpperCase : function (data) {

                    return data.toUpperCase();

                }

            }

        });

    </script>

</body>

</html>

Nếu muốn sử dụng nhiều filter cùng 1 lúc thì ngăn cách nó với nhau bằng dấu “|”.

VD:

{{ name | strUpperCase | lastWord }}

Hoặc cũng có thể truyền thêm các tham số vào trong filter, nhưng tham số truyền vào tiếp theo sẽ được tính từ tham số thứ 2.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ name | strUpperCase |sliceString(9) }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'vu thanh tai'

            },

            filters : {

                strUpperCase : function (data) {

                    return data.toUpperCase();

                },

                sliceString : function (data, num) {

                    return data.slice(num);

                }

            }

        });

    </script>

</body>

</html>

1. **Shorthands**

Tiền tố v- phục vụ như một trực quan để đánh dấu sử dụng vue.js so với các framework khác, ví dụ như angular dùng ng- . Chính vì thế nên vue.js cũng có cung cấp cho chúng ta 2 cách viết ngắn gọn hơn với 2 dạng directive là v-bind và v-on như sau:

1. **V-bind**

<!-- cú pháp đầy đủ -->

<a v-bind:href="url"></a>

<!-- cú pháp ngắn gọn -->

<a :href="url"></a>

1. **V-on**

<!-- cú pháp đầy đủ -->

<a v-on:click="doSomething"></a>

<!-- cú pháp ngắn gọn -->

<a @click="doSomething"></a>

1. **Method, computed property và watcher trong vue.js**
2. **Method**

Methods là danh sách các phương thức của component, ở trong đây sẽ viết các phương thức mà từ đó có thể thực thi ở một nơi nào đó trong component.

Một phương thức có thể trả về một giá trị hoặc cũng có thể không, phương thức có thể nhận tham số đầu vào và xử lý. Phương thức sẽ được gọi mỗi khi component được render.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ message }}</</h1>

        <div><button @click="sayHello">Click me</button></div>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data() {

                return {

                    message: 'This is my first component using binding data'

                }

            },

            methods: {

                sayHello() {

                    alert('Hello World')

                }

            }

        });

    </script>

</body>

</html>

1. **Computed property**

Computed property trong Vue.js được thể hiện dưới dạng một phương thức hoặc một object chứa các phương thức setter và getter, nhưng khi bạn gọi một computed property thì bạn sẽ gọi nó như một thuộc tính bình thường. Và khi đó các hành động trong computed property sẽ được thực thi.

Chắc hẳn khi đọc xong dòng trên thì sẽ thắc mắc vậy tại sao không dùng luôn property đi bày đặt dùng computed property làm gì? Tuy nhiên thì trong một vài trường hợp sử dụng biểu thức đơn giản khi sử dụng trong property thì điều thắc mắc trên sẽ đúng, còn trường hợp các biểu thức phức tạp thì rất rối, không thể tái sử dụng và rất khó để bảo trì.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ name.split('').reverse().join('') }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai'

            }

        });

    </script>

</body>

</html>

1. **Cú pháp**

Để khai báo computed property trong Vue.js thì cần đặt nó ở trong scope object computed.

VD: Khai báo một computed chuyển đổi biến name trong data scope thành in hoa.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

<div id="app">

    <h1 class="text-red">{{ convertToUpper }}</h1>

</div>

<script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

<script type="text/javascript">

    var app = new Vue({

        el: '#app',

        data: {

            name: 'Vu Thanh Tai'

        },

        computed : {

            convertToUpper: function (){

                return this.name.toUpperCase();

            }

        }

    });

</script>

</body>

</html>

Nếu như trong một vài trường hợp nào đó muốn sử dụng computed property ở ngoài vue instance thì chỉ cần truy xuất theo cú pháp nameObject.property.

VD: In ra computed convertToUpper ở trên.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <p id="log"></p>

        <h1 class="text-red">{{ convertToUpper }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai'

            },

            computed : {

                convertToUpper: function (){

                    return this.name.toUpperCase();

                }

            }

        });

        document.getElementById("log").innerHTML = app.convertToUpper;

    </script>

</body>

</html>

1. **Computed vs Methods**

Chắc hẳn sau khi đọc hết phần trên các bạn cũng thắc mắc những thứ nêu ở trên thì method cũng có thể làm được? Điều đó hoàn toàn đúng, chúng ta có thể chuyển convertToUpper computed trên thành methods như sau:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js Toidicode.com</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ convertToUpper() }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai'

            },

            methods : {

                convertToUpper: function (){

                    return this.name.toUpperCase();

                }

            }

        });

    </script>

</body>

</html>

Nhưng computed sẽ khác với methods ở chỗ computed được lưu vào cache và computed sẽ chỉ được thay đổi khi dữ liệu bên trong nó thay đổi.

VD: Như với trường hợp computed convertToUpper ở trên thì khi mà biến name không thay đổi thì chương trình sẽ in ra luôn kết quả của lần trước đó mà không phải thực thi lại tiến trình bên trong nó.

Kết luận điểm khác biệt giữa computed và methods:

* Khi gọi computed ta không được thêm cặp dấu () đằng sau, điều đó tức là computed cũng không thể nhận tham số đầu vào như methods
* Vì không nhận tham số đầu vào nên computed chỉ nên dùng với các dữ liệu có sẵn trong data của component
* Computed sẽ chỉ tính toán lại mỗi khi các biến phụ thuộc trong nó thay đổi, còn methods sẽ được tính toán bất kì khi nào nó được gọi, nên nếu biết tận dụng computed để tính toán các dữ liệu có sẵn thì sẽ cải thiện được performance app của các bạn. Điều tuyệt vời của computed là nó sẽ được cached nên giả sử bạn có 1 computed với hàng loạt tính toán, nhiều vòng lặp trong đó, mà nếu các các biến phụ thuộc không thay đổi thì khi sử dụng nó sẽ chỉ mất thời gian tính 1 lần, những lần sau kết quả sẽ được sử dụng lại từ lần trước.

1. **Getter and Setter**

Ở mỗi computed property, mặc định thì nó sẽ chỉ chứa getter.

VD: Các viết computed convertToUpper ở trên tương đương với với cách sau:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ convertToUpper }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai'

            },

            computed : {

                convertToUpper: {

                    get : function () {

                        return this.name.toUpperCase();

                    }

                }

            }

        });

    </script>

</body>

</html>

Nhưng cũng có thể thêm setter cho các computed bằng cách tạo thêm một set function bên trong computer.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ convertToUpper }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai'

            },

            computed : {

                convertToUpper: {

                    get : function () {

                        return this.name.toUpperCase();

                    },

                    set: function (name) {

                        this.name = name;

                    }

                }

            }

        });

    </script>

</body>

</html>

Để xem cách hoạt động thì các chỉ cần gán giá trị cho computer là biết.

VD: Thực hiện set lại giá trị cho computed ở trên.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ convertToUpper }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'Vu Thanh Tai' //giá trị ban đầu.

            },

            computed : {

                convertToUpper: {

                    get : function () {

                        return this.name.toUpperCase();

                    },

                    set: function (name) {

                        this.name = name;

                    }

                }

            }

        });

        //thiết lập lại giá trị

        app.convertToUpper = "Hoc Vuejs";

    </script>

</body>

</html>

1. **Watcher**
2. **Cú pháp**

Watcher trong Vue.js đúng như nghĩa của nó có nghĩa là theo dõi, nó có nhiệm vụ theo dõi sự thay đổi của dữ liệu và thực thi hành động kèm theo.

Cú pháp: Để khai báo watcher trong vue.js thì các bạn cần phải tuân thủ các nguyên tắc sau:

* Tên của watcher phải trùng với tên của data cần theo dõi.
* Và các watcher phải được đặt trong watch scope.

VD: Khai báo watcher text có nhiệm vụ in ra nội dung của name mỗi khi input text.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js Toidicode.com</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ name }}</h1>

        <p>

          Name:

          <input v-model="text">

        </p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script src="https://cdn.jsdelivr.net/npm/lodash@4.13.1/lodash.min.js"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                name: 'I',

                text: ''

            },

            watch: {

                text: function () {

                  this.name = 'am'

                  this.getName()

                }

            },

            methods: {

              // \_.debounce là một hàm do Lodash cung cấp

              // Để tìm hiểu rõ hơn cách hoạt động của hàm này,

              // bạn có thể truy cập: https://lodash.com/docs#debounce

              getName: \_.debounce(

                function () {

                  this.name = this.text

                },

                // Đây là thời gian (đơn vị mili giây) chúng ta đợi người dùng dừng gõ.

                500

              )

            }

        });

    </script>

</body>

</html>

1. **Computed vs Watcher**

Nhìn chung thì cả computed và watcher đều được thực hiện khi chúng ta thay đổi data bên trong nó. Và cả hai đều có thể áp dụng để thực thi một hành động nào đó khi có 1 biến phụ thuộc thay đổi. Đối với trường hợp này chúng ta hay cùng xem cách xử lý của watcher và computed xem cái nay hay hơn nhé!

Watcher:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ fullName }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                firstName: 'Vu Thanh',

                lastName: 'Tai',

                fullName: 'Vu Thanh Tai',

            },

            watch: {

                firstName : function (val) {

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

             },

             lastName : function (val) {

                 this.fullName = this.firstName + ' ' + val;

             }

         }

     });

    </script>

</body>

</html>

Computed:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 class="text-red">{{ fullName }}</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                firstName: 'Vu Thanh',

                lastName: 'Tai',

            },

            computed: {

                fullName : function () {

                    return this.firstName + ' ' + this.lastName;

                }

            }

        });

    </script>

</body>

</html>

Nhìn vào hai cách trên thì computed có vẻ ngắn gọn và đơn giản hơn. Nhưng tùy vào từng trường hợp sẽ có các cách giải quyết khác nhau nên không khẳng định cái nào tốt hơn cái nào.

1. **Class và style bindings trong Vue.js**
2. **Classes binding**

Trong vue.js chúng ta có hai cách thực hiện binding HTML classes và hai cách này cũng sử dụng được trong component.

1. **Cú pháp kiểu Object**

Trong vue.js chúng ta có thể truyền 1 object vào v-bind:class để bật tắt class một cách tự động.

VD:

<h1 v-bind:class="{ active : isActive }"></h1>

Đoạn code trên thể hiện sự xuất hiện của class active là hoàn toàn phụ thuộc vào giá trị của isActive là true hay false. Và đương nhiên chúng ta cũng có thể tắt mở nhiều class bằng cách thêm các trường vào trong object.

VD:

<h1 v-bind:class="{ active : isActive, 'text-red': isRed }"></h1>

Lúc này chúng ta chỉ cần xác định giá trị cho các trường đó là true thì nó sẽ hiển thị vào false thì nó sẽ không hiển thị.

Để cho dễ hiểu hơn thì ta có thể tham khảo ví dụ sau:

VD: toggle class text-red của thẻ h1.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="{ 'text-red': isRed }">Hello World!/h1>

        <button type="button" v-on:click="toggleClass">Toggle Class</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                isRed: true

            },

            methods: {

                toggleClass : function () {

                    if (this.isRed == true) {

                        this.isRed = false;

                    } else {

                        this.isRed = true;

                    }

                }

            }

        });

    </script>

</body>

</html>

Tuy nhiên, thì object này cũng không nhất thiết phải viết inline trên tag HTML mà chúng ta có thể chuyển nó vào trong data scope như sau:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="classObject">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                classObject:

                { 'text-red': true }

            }

        });

    </script>

</body>

</html>

Hoặc cũng có thể cho object này vào trong một computed.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="classObject">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                isRed: true

            },

            computed: {

                classObject : function () {

                    return { 'text-red' : this.isRed}

                }

            }

        });

    </script>

</body>

</html>

1. **Cú pháp kiểu Array**

Ngoài cách bind bằng kiểu bject thì trong Vue.js cũng hỗ trợ cho chúng ta bind vào class dạng mảng.

<h1 v-bind:class="[active, textRed]"></h1>

Lúc này active, và textRed sẽ là 2 thuộc tính của vue instance chứa giá trị là tên của class cần thay thế.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

    .border-bottom{

        border-bottom: 1px solid orange;

        padding-bottom: 10px;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="[border, textRed]">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                border: 'border-bottom',

                textRed: 'text-red'

            }

        });

    </script>

</body>

</html>

Nếu như không muốn hiển thị class nào thì ta cũng chỉ cần thay đổi giá trị của trường đó thành false.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

    .border-bottom{

        border-bottom: 1px solid orange;

        padding-bottom: 10px;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="[border, textRed]">Hello World!</h1>

        <button type="button" v-on:click="toggleClass">Toggle Class</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                border: 'border-bottom',

                textRed: 'text-red'

            },

            methods: {

                toggleClass : function () {

                    if (this.textRed == 'text-red') {

                        this.textRed = false;

                    } else {

                        this.textRed = 'text-red';

                    }

                }

            }

        });

    </script>

</body>

</html>

Hoặc ta cũng có thể viết dưới dạng biểu thức điều kiện rút gọn như sau:

<h1 v-bind:class="[border ? 'border-bottom' : '', textRed]"></h1>

Và Vue.js cũng hỗ trợ chúng ta khai báo kiểu object trong mảng.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .text-red{

        color: red;

    }

    .border-bottom{

        border-bottom: 1px solid orange;

        padding-bottom: 10px;

    }

</style>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:class="[{'border-bottom' : border}, textRed]">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                border: true,

                textRed: 'text-red'

            }

        });

    </script>

</body>

</html>

1. **Binding inline styles**

Đối với binding inline styles chúng ta cũng có thể sử dụng được các cách bind nhưng binding class.

1. **Kiểu Object**

Đối với kiểu này thì chúng ta sử dụng như đối với binding class, nhưng chỉ khác là giá trị lúc này sẽ không phải true false nữa, mà giá trị sẽ là giá trị của thuộc tính chúng ta cần style cho nó. Nếu như không muốn hiển thị thuộc tính đó nữa thì chỉ việc đổi giá trị của nó thành false.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:style="{color: colorValue, borderBottom : borderValue, paddingBottom: paddingValue}">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                colorValue: 'red',

                borderValue: '1px solid orange',

                paddingValue: '10px',

            }

        });

    </script>

</body>

</html>

Đối với cách này thì các thuộc tính có từ hai từ trở lên, ví dụ như text-align, border-bottom,... thì vue.js cung cấp cho chúng ta 2 cách viết nó là có thể viết dưới dạng camelCase hoặc viết dưới dạng bình thường là snake\_case, nhưng đối với dạng này thì chúng ta phải bao ngoài nó là cặp dấu nháy “'”

VD:

<h1 v-bind:style="{'border-bottom' : borderValue, paddingBottom: paddingValue}"></h1>

1. **Kiểu Array**

Đối với kiểu array cũng tương tự như đối với binding class và vue.js cũng hỗ trợ chúng ta kiểu object trong kiểu array.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <h1 v-bind:style="[colorValue, borderValue, paddingValue]">Hello World!</h1>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                colorValue: {

                    color : 'red'

                },

                borderValue: {

                    //sanke\_case

                    'border-bottom' : '1px solid orange'

                },

                paddingValue: {

                    //camelCase

                    paddingBottom: '10px'

                }

            }

        });

    </script>

</body>

</html>

1. **Conditional rendering trong Vue.js**
2. **V-if**

Trong vue.js chúng ta sử dụng v-if directive để thực hiện các render có liên quan đến điều kiện.

<h1 v-if="show">Hello World!</h1>

Lúc này, nếu như giá trị bên trong v-if trả về là true thì tag h1 sẽ được hiển thị và ngược lại, là false thì tag h1 sẽ được ẩn. Điều này tương tự như cách hoạt động của câu lệnh if else bình thường chỉ có điều khác ở cách trình bày. Và đương nhiên ta cũng có thể kết hợp với v-else.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <h1 v-if="show">Hello World 1!</h1>

        <h1 v-else >Hello World 2!</h1>

        <button type="button" @click="toggleTag">Toggle Tag</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                show: true,

            },

            methods: {

                toggleTag : function () {

                    if (this.show == true ) {

                        this.show = false;

                    } else {

                        this.show = true;

                    }

                }

            }

        });

    </script>

</body>

</html>

Và nếu như ta có một nhóm các tag phụ thuộc vào nhau thì vue.js cũng cung cấp cho chúng ta gom nhóm nó trong tag <template>.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <template v-if="show">

            <h1>Hello World!</h1>

            <p>Genesis</p>

        </template>

        <button type="button" @click="toggleTag">Toggle Tag</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                show: true,

            },

            methods: {

                toggleTag : function () {

                    if (this.show == true ) {

                        this.show = false;

                    } else {

                        this.show = true;

                    }

                }

            }

        });

    </script>

</body>

</html>

Và nếu như điều kiện của bạn có nhiều nhánh thì kể từ phiên bản vue.js 2.1.0 đã hỗ trợ thêm cho chúng ta v-else-if để bạn có thể phân thêm nhánh cho condition.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <h1>Điểm bằng: {{ point }}</h1>

        <h2>

            =>

            <span v-if="point < 4"> F</span>

            <span v-else-if="point >= 4 && point < 5.5"> D</span>

            <span v-else-if="point >= 5.5 && point < 7"> C</span>

            <span v-else-if="point >= 7 && point < 8.5"> B</span>

            <span v-else> A</span>

        </h2>

        <button type="button" @click="toggleTag">Toggle Tag</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                point: 0,

            },

            methods: {

                toggleTag : function () {

                    if (this.point == 10 ) {

                        this.point = 0;

                    } else {

                        this.point++;

                    }

                }

            }

        });

    </script>

</body>

</html>

1. **V-show**

Ngoài cách sử dụng v-if ở trên thì ta cũng có thể sử dụng v-show với nguyên lý tương tự. Tức là giá trị bên trong v-show trả về true thì tag chứa nó sẽ hiển thị và ngược lại giá trị bên trong nó là false thì tag chứa nó sẽ ẩn.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body style="text-align: center;">

    <div id="app">

        <h1 v-show="show">Hello World!</h1>

        <button type="button" @click="toggleTag">Toggle Tag</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                show: true,

            },

            methods: {

                toggleTag : function () {

                    this.show = this.show == true ? false : true;

                }

            }

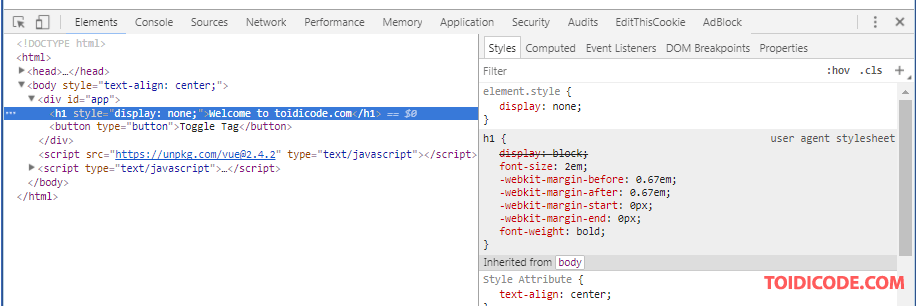
        });

    </script>

</body>

</html>

V-show luôn luôn được render nó chỉ ẩn đi bằng css còn lại thì trên DOM vẫn tồn tại tag chứa nó.



v-show không hỗ trợ thẻ <template> và cũng không hoạt động với v-else.

1. **V-if vs v-show**

* v-if không render ngay ra toàn bộ các element mà tùy vào giá trị kiểm tra, còn với v-show sẽ render ra ngay từ đầu và chỉ quyết định có display nó hay không
* Từ kết quả trên chúng ta thấy rằng: nếu nội dung trong block mà ít, và trong quá trình sử dụng mà user sẽ tác động đến block nhiều thì dùng v-show sẽ cho tốc độ tốt hơn, còn nếu nội dung trong block mà nhiều cùng với nhiều xử lý hoặc nội dung của block ít thay đổi trong suốt vòng đời của component thì khi đó không nên dùng v-show vì lúc đó trang của chúng ta sẽ nặng làm giảm performance và ta cần nghĩ đến dùng v-if

1. **List rendering trong Vue.js**

Hành động dùng vòng lặp để render dữ liệu

1. **v-for.**

Trong Vue.js chúng ta có thể sử dụng v-for directive để hiển thị ra các item có trong mảng với cú pháp:

v-for="item in list"

Trong đó:

* item là biến được gán cho các item có trong mảng.
* list là mảng dữ liệu các bạn cần duyệt.

VD: Render tất cả các name có trong mảng student của ví dụ sau.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <ul>

            <li v-for="list in student">{{ list }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                'Vũ Thanh Tài',

                'Nguyễn Thị Hồng Hạnh',

                'Nguyễn Văn A',

                'Lò Thị Vi Sóng',

                'Trần Như Nhộng',

                ]

            }

        });

    </script>

</body>

</html>

Nếu như ta muốn lấy ra index của phần tử nữa thì bạn sử dụng cú pháp sau:

v-for="(item, index) in list"

Khi đó tham số index sẽ chứa chỉ số index của phần tử trong mảng.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <ul>

            <li v-for="(list, index) in student">Name: {{ list }} - Vị Trí thứ: {{ index }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                'Vũ Thanh Tài',

                'Nguyễn Thị Hồng Hạnh',

                'Nguyễn Văn A',

                'Lò Thị Vi Sóng',

                'Trần Như Nhộng',

                ]

            }

        });

    </script>

</body>

</html>

Và nếu như đã quen sử dụng cú pháp của vòng lặp trong javascript thông thường thì ta cũng có thể thay thế in bằng of.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <ul>

            <li v-for="list of student">Name: {{ list }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                'Vũ Thanh Tài',

                'Nguyễn Thị Hồng Hạnh',

                'Nguyễn Văn A',

                'Lò Thị Vi Sóng',

                'Trần Như Nhộng',

                ]

            }

        });

    </script>

</body>

</html>

1. **Template v-for.**

Trong trường hợp ta muốn render ra 1 nhóm tag trong mỗi lần lặp thì Vue.js cũng hỗ trợ sử dụng tag template để gom nhóm các tag đó giống như v-if.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    ul{

        list-style: none;

        margin: 0;

        padding: 0;

    }

    .border{

        border-bottom: 1px solid orange;

        padding-bottom: 15px;

        margin-bottom: 15px;

    }

</style>

<body>

    <div id="app">

        <ul >

            <template v-for="list of student">

                <li class="border"></li>

                <li>{{ list }}</li>

            </template>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                'Vũ Thanh Tài',

                'Nguyễn Thị Hồng Hạnh',

                'Nguyễn Văn A',

                'Lò Thị Vi Sóng',

                'Trần Như Nhộng',

                ]

            }

        });

    </script>

</body>

</html>

1. **Object v-for.**

Và trong Vue.js ta cũng có thể dùng v-for để duyệt các phần tử bên trong object được.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul >

            <template v-for="list in student">

                <li>{{ list }}</li>

            </template>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: {

                    name : 'Vũ Thanh Tài',

                    age: 22,

                    address: 'Nam Định'

                }

            }

        });

    </script>

</body>

</html>

Và nếu như ta muốn lấy ra cả key và vị trí của phần tử trong object thì ta chỉ cần thêm tiếp các đối số theo cú pháp sau:

v-for="(list, key, index) in object"

Trong đó:

* list là giá trị của phần tử.
* key là key của phần tử đó.
* index là thứ tự của phần tử đó trong object (tính từ 0).

VD: in ra key và giá trị của key đó trong object.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul>

            <li v-for="(list, key) in student">{{ key }} = {{ list }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: {

                    name : 'Vũ Thanh Tài',

                    age: 22,

                    address: 'Nam Định'

                }

            }

        });

    </script>

</body>

</html>

VD: in ra giá trị, key và index của các phần tử có trong object.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul>

            <li v-for="(list, key, index) in student">{{ key }} = {{ list }}, index = {{ index }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: {

                    name : 'Vũ Thanh Tài',

                    age: 22,

                    address: 'Nam Định'

                }

            }

        });

    </script>

</body>

</html>

1. **Range v-for.**

Bên cạnh đó, khi ta muốn lặp dữ liệu với một số lần các định sẵn thì vue.js cũng hỗ trợ chúng ta luôn.

VD: Dùng v-for in ra các số từ 0 đến 10.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul>

            <li v-for="i in 10">{{ i }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **v-for with v-if.**

Trên cùng 1 node nếu như tồn tại cả v-for và v-if thì chương trình sẽ ưu tiên v-for trước, điều này rất có lợi nếu như ta muốn kiểm tra sự tồn tại của một nhánh nào đó trong vòng lặp.

VD: Chỉ in ra các object có chứa key name.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul >

            <li v-for="item in student" v-if="item.name">{{ item }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                { name : 'Vũ Thanh Tài'},

                { name : 'Trần Như Nhộng'},

                { age : 22},

                { name : 'Nguyễn Văn B'},

                ]

            }

        });

    </script>

</body>

</html>

Tương tự nếu như ta muốn xác định sự tồn tại của vòng lặp thì bạn chỉ cần đặt v-if bao ngoài v-for là được.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul v-if="allow">

            <li v-for="item in student">{{ item }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                allow: true,

                student: [

                { name : 'Vũ Thanh Tài'},

                { name : 'Trần Như Nhộng'},

                { name : 'Nguyễn Văn B'},

                ]

            }

        });

    </script>

</body>

</html>

Hoặc ta cũng có thể đặt v-if vào trong tag template.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <template v-if="allow">

            <ul>

                <li v-for="item in student">{{ item }}</li>

            </ul>

        </template>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                allow: true,

                student: [

                { name : 'Vũ Thanh Tài'},

                { name : 'Trần Như Nhộng'},

                { name : 'Nguyễn Văn B'},

                ]

            }

        });

    </script>

</body>

</html>

1. **Array change detection.**

Cũng giống như thường lệ, đối với mảng thì ta cũng có thể sự dụng các hàm liên quan để tác động vào mảng (xem danh sách các hàm xử lý mảng trong javascript) từ đó dữ liệu nếu như có sự thay đổi thì vue.js cũng cập nhật lại danh sách của v-for ngay lập tức.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul>

            <li v-for="item in student">{{ item }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                student: [

                { name : 'Vũ Thanh Tài'},

                { name : 'Trần Như Nhộng'},

                { name : 'Nguyễn Văn B'},

                ]

            }

        });

        app.student.push({name: 'Lo Vi Song'})

    </script>

</body>

</html>

1. **Displaying filtered or sorted results.**

Đôi lúc chúng ta cần thay sắp xếp hoặc lọc dữ liệu mà không thực sự muốn thay đổi dữ liệu thì ta cũng có thể áp dụng computed property vào đó.

VD: Dùng computed để lọc các dữ liệu là số chẵn.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

<div id="app">

    <ul>

        <li v-for="item in evenNumber">{{ item }}</li>

    </ul>

</div>

<script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

<script type="text/javascript">

    var app = new Vue({

        el: '#app',

        data: {

            numbers: [1,3,5,9,6,4,2,12,34]

        },

        computed : {

            evenNumber: function () {

                return this.numbers.filter(function (number) {

                  return number % 2 === 0

              });

            }

        }

    });

</script>

</body>

</html>

Tương tự như thế ta cũng có thể sử dụng phương thức để có thể sử dụng lại nhiều lần hơn.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <ul>

            <li v-for="item in evenNumber(numbers)">{{ item }}</li>

        </ul>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                numbers: [1,3,5,9,6,4,2,12,34]

            },

            methods : {

                evenNumber: function (numbers) {

                    return numbers.filter(function (number) {

                      return number % 2 === 0

                  });

                }

            }

        });

    </script>

</body>

</html>

1. **Event handling trong Vue.js**
2. **Listen to Events.**

Trong Vue.js chúng ta sử dụng v-on directive để lắng nghe các sự kiện từ phía DOM.

VD: Lắng nghe sự kiện click trên DOM.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <button v-on:click="counter += 1">Click</button>

        <p>Bạn vừa click vào button lần thứ {{ counter }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            }

        });

    </script>

</body>

</html>

1. **Method Event Handlers.**

Thông thường ở các bài toán thực tế thì code xử lý sự kiện sẽ không đơn giản là một dòng như trên, mà nó sẽ rất phức tạp. Chính vì thế nên Vue.js cũng hỗ trợ chúng ta tách phần code xử lý vào hàm và gọi hàm đó ở trên sự kiện như javascript thuần.

VD: Chuyển đoạn code xử lý sự kiện ở ví dụ trên vào hàm.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <button v-on:click="numberPlus">Click</button>

        <p>Bạn vừa click vào button lần thứ {{ counter }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                numberPlus: function () {

                    this.counter += 1;

                }

            }

        });

    </script>

</body>

</html>

Và ta cũng có thể chuyền thêm tham số vào hàm khi gọi chúng ở DOM (như javascript thuần).

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <button v-on:click="say('hello')">Hello</button>

        <button v-on:click="say('goodbye')">Goodbye</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                say: function (message) {

                    alert(message  + '!');

                }

            }

        });

    </script>

</body>

</html>

Đôi lúc nếu như ta muốn sử dụng các thông số mặc định của sự kiện thì trong Vue.js cũng hỗ trợ bạn luôn, bằng cách các ta truyền thêm vào hàm biến $event.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <button v-on:click="warn('Không thể submit form',$event)">Click</button>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                warn: function (message, event) {

                    if (event) event.preventDefault();

                    alert(message  + '!');

                }

            }

        });

    </script>

</body>

</html>

1. **Event Modifiers.**

Trong javascript thuần các ta thường sử dụng các event modifiles để tác động đến cách xử lý của sự kiện như: preventDefault(), stopPropagation(),... Và những cách này ta vẫn có thể sử dụng được đối với Vue.js, nhưng ngoài cách đó thì Vue.js còn hỗ trợ chúng ta khai báo nó ngay ở trên directive bằng cách thêm chúng vào đằng sau directive và ngăn cách giữa chúng bằng dấu “.”

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <form action="https://toidicode.com" v-on:submit.prevent="warn">

            <input type="text" placeholder="nhập vào tên...">

            <button type="submit">Click</button>

        </form>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                warn: function () {

                    alert('Bạn vừa thực hiện submit form, nhưng đã bị prevent');

                }

            }

        });

    </script>

</body>

</html>

Và dưới đây là danh sách các modifier được hỗ trợ:

* .stop
* .prevent
* .capture
* .self
* .once

1. **Key modifiers.**

Đối với javascript thông thường thì khi muốn bắt sự nhấn phím của mọt phím nào đó thì chúng ta cần phải biết được mã code của phím đó rồi đem đi so sánh. Và trong Vue.js chúng ta cũng có thể sử dụng như thế, nhưng ngoài ra thì nó còn hỗ trợ chúng ta khai báo luôn sự kiện nhấn phím luôn khi khai báo directive. Bằng cách thêm mã code của phím đó vào sau directive v-on:keyup và ngăn cách giữa chúng bời dấu .

VD: Bắt sự kiện nhấn phím enter.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <input type="text" v-on:keyup.13="warn">

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                warn: function () {

                    alert('Bạn vừa ấn phím enter');

                }

            }

        });

    </script>

</body>

</html>

Nhưng để nhớ được hết mã code của từng phím thì không phải là điều đơn giản, nên Vue.js đã thêm sẵn cho chúng ta các phím hay dùng thay thế cho key của các phím đó.

Và đây là danh sách các phím mà Vue.js đã thêm sẵn cho chúng ta:

* .enter
* .tab
* .delete (Phím này dùng được cho cả phím delete và backspace)
* .esc
* .space
* .up
* .down
* .left
* .right
* .ctrl
* .alt
* .shift
* .meta (Trên bàn phím window thì nó là phím window, còn trên bàn phím mac thì nó là phím command).

VD: Thay thế keycode của phím enter ở ví dụ trên bằng key mà Vue.js đã hỗ trợ sẵn.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <input type="text" v-on:keyup.enter="warn">

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                counter: 0,

            },

            methods: {

                warn: function () {

                    alert('Bạn vừa ấn phím enter');

                }

            }

        });

    </script>

</body>

</html>

Và Vue.js cũng cho phép các ta định nghĩa thêm các key Code mới thông qua config.keyCodes.

VD: Định nghĩa thêm key code cho phím f1.

Vue.config.keyCodes.f1 = 112;

1. **Why listener in HTML.**

Vue.js hoặt động theo mô hình ViewModel và nó ràng buộc rất chặt chẽ các hàm này nên ta hoàn toàn có thể yên tâm khi lắng nghe sự kiện ở trên HTML mà không hề ảnh hưởng đến nguyên lý "separation of concerns" và nó cũng không hề gây khó khăn gì cho việc bảo trì. Mà ngược lại thì chúng ta sẽ có được rất nhiều lợi ích khi sử dụng v-on trên HTML, như:

* Dễ dàng hơn trong việc xác định vị trí gọi hàm xử lý sự kiên bằng cách lướt qua các template HTML.
* Vì không phải attach các hàm xử lý sự kiện trong code JS, mà được vào mô hình ViewModel nên code của bạn sẽ logic, trong sáng và dễ test hơn.
* Và khi một ViewModel được destroy thì tất cả các hàm xử lý sự kiện cũng bị loại bỏ luôn, nên ta sẽ không phải lo nghĩ về việc bỏ sót 1 event nào.

1. **Form input binding trong Vue.js**
2. **Basic Usage.**

Trong vue.js ta có thể sử dụng v-model directive để tạo ra các ràng buộc dữ liệu hai chiều trên các input và textarea element. Và nó sẽ lựa chon đúng cách và cập nhật các element trên các input type. Mặc dù nó có một chút magic, nhưng về cơ bản thì thì v-model là một cú pháp đường dẫn để cập nhật dữ liệu trên các sự kiện của người dùng.

Đối với v-model nó sẽ bỏ qua tất cả các khởi tạo của value, checked hay selected trên các thành phần của form và nó luôn xử lý các dữ liệu có trong Vue instance, chính vì thế ta nên khai báo các giá trị khởi tạo đó ở trong data scope của Vue instance.

1. **Text**

VD: Sử dụng v-model trong Vue.js trên thẻ input.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="text" v-model="message" placeholder="Nhập cái gì đó vào đây đi ông bạn!">

        <p>Mesage = {{ message }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: null,

            }

        });

    </script>

</body>

</html>

1. **Textarea**

VD: Sử dụng v-model trên textarea.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <textarea v-model="message" placeholder="Nhập cái gì đó vào đây đi ông bạn!" rows="5" cols="70"></textarea>

        <p>Message = {{ message }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: null,

            }

        });

    </script>

</body>

</html>

Đối với thẻ tetarea thì ta không thể bind dữ liệu bằng cách <textarea>{{ message }}</textarea> được, mà ta phải sử dụng cú pháp v-model như ví dụ trên.

1. **Checkbox**

VD: checkbox đơn

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

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

        <label>{{ checked }}</label>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                checked: false,

            }

        });

    </script>

</body>

</html>

VD: multiple checkbox.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

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

        <label for="php">PHP</label>

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

        <label for="js">JS</label>

        <input type="checkbox" id="nodejs" value="Node.js" v-model="checked">

        <label for="nodejs">Node.js</label>

        <p>Các input đã check: {{ checked }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                checked: [],

            }

        });

    </script>

</body>

</html>

1. **Radio**

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="radio" id="php" name="language" value="PHP" v-model="checked">

        <label for="php">PHP</label>

        <input type="radio" id="js" name="language" value="JS" v-model="checked">

        <label for="js">JS</label>

        <input type="radio" id="nodejs" name="language" value="Node.js" v-model="checked">

        <label for="nodejs">Node.js</label>

        <p>Radio đã check: {{ checked }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                checked: null,

            }

        });

    </script>

</body>

</html>

Select

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <select name="language" v-model="selected">

            <option value="" disabled="true" selected>Chọn...</option>

            <option value="PHP">PHP</option>

            <option value="JS">JS</option>

            <option value="Node.js">Node.js</option>

            <option value="Vue.js">Vue.js</option>

        </select>

        <p>Select đã chọn: {{ selected }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                selected: '',

            }

        });

    </script>

</body>

</html>

*Nếu như giá trị khởi tạo của bạn không khớp với một option nào thì nó sẽ kích hoạt trạng thái unselected, nhưng điều này cũng đồng nghĩa với việc nó sẽ không hoạt động được trên IOS vì trên IOS không hỗ trợ trường hợp thay đổi này. Chính vì thế ta cần thêm một option ở trạng thái disable (như trong ví dụ trên) để khác phục hạn chế này.*

VD: Còn đối với multi select.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <select name="language" v-model="selected" multiple>

            <option value="" disabled="true" selected>Chọn...</option>

            <option value="PHP">PHP</option>

            <option value="JS">JS</option>

            <option value="Node.js">Node.js</option>

            <option value="Vue.js">Vue.js</option>

        </select>

        <p>Select đã chọn: {{ selected }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                selected: [],

            }

        });

    </script>

</body>

</html>

Và đương nhiên là ta cũng có thể kết hợp nó với v-for để render ra dữ liệu cho select element.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <select name="language" v-model="selected">

            <option v-for="item in option" v-bind:value="item.value">

                {{ item.text }}

            </option>

        </select>

        <p>Select đã chọn: {{ selected }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                selected: 'js',

                option: [

                { text: 'PHP', value: 'php' },

                { text: 'Node.js', value: 'node' },

                { text: 'Javascript', value: 'js' },

                { text: 'Vue.js', value: 'vuejs' },

                ],

            }

        });

    </script>

</body>

</html>

1. **Value bindings.**

Đối với các thành phần là radio hoặc select option thì v-model thường sẽ là chuỗi tĩnh. Còn nếu thành phần là checkbox thì thường sẽ là giá trị boolean true false. Nhưng đôi lúc nếu như ta muốn sử các thuộc tính có giá trị động trên vue instance thì bạn có thể sử dụng v-bind directive để thực hiện điều đó.

Ví dụ đối với checkbox element.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="checkbox" v-model="toggle" v-bind:true-value="yes" v-bind:false-value="no">

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                toggle: true,

                yes: true,

                no: false,

            }

        });

    </script>

</body>

</html>

Trong trường hợp trên nếu như giá trị của toggle mà bằng giá trị của yes thì checkbox sẽ checked và ngược lại giá trị của toggle mà bằng giá trị của no thì checkbox sẽ unchecked.

Và điều này cũng xảy ra tương tự đối với radio và selected.

VD: Đối với radio thì radio sẽ checked khi giá trị của pick bằng với giá trị của a.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="checkbox" v-model="pick" v-bind:value="a">

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                pick: 'b',

                a: 'b'

            }

        });

    </script>

</body>

</html>

VD: Đối với select thì Option sẽ được selected nếu giá trị của selected.value = với giá trị vủa value.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <select v-model="selected">

            <option v-bind:value="{value : 'php'}">{{ item.text }}</option>

        </select>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                selected: { value : 'php'}

            }

        });

    </script>

</body>

</html>

1. **Modifiers.**
2. **.lazy**

Thông thường thì v-model sẽ tự động đồng bộ trong quá trình soạn thảo (chỉ trừ các ngôn ngữ IME - xem ở phần lời kết nhé!), nhưng nếu ta muốn thay đổi trạng thái đó thì ta có thể sử dụng .lazy modifier để chuyển đổi nó sang trạng thái change event (có nghĩa là khi input thay đổi thì dữ liệu mới tiến hành đồng bộ).

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="text" v-model.lazy="message" placeholder="nhập dữ liệu...">

        <p>message = {{ message }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: null

            }

        });

    </script>

</body>

</html>

1. **.number**

Nếu như ta muốn ràng buộc dữ liệu đầu vào của input chỉ được phép là số thì hãy sử dụng modifier .number để ràng buộc điều này.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="text" v-model.number="message" placeholder="nhập dữ liệu...">

        <p>message = {{ message }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: null

            }

        });

    </script>

</body>

</html>

Điều này sẽ vô cùng có ích, bời ngay cả các input mà có type bằng number (type=number) nó cũng luôn trả về giá trị là 1 string.

1. **.trim**

Còn nếu như ta muốn loại bỏ những khoảng trắng ở hai đầu của dữ liệu thì .trim modifier là một lựa chọn cực kỳ tuyệt vời.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<body>

    <div id="app">

        <input type="text" v-model.trim="message" placeholder="nhập dữ liệu...">

        <p>message = {{ message }}</p>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            data: {

                message: null

            }

        });

    </script>

</body>

</html>

1. **v-model với component.**

Phần sau

1. **Component trong Vue.js**
2. **Component là gì?**

Component là một trong các tính năng mạnh mẽ nhất của Vue.js. Nó giúp cho chúng ta gom nhóm các mã HTML lại để tái sử dụng cho các module tương tự. Ở một mức độ cao thì component là một thành phần được Vue.js biên dịch để xử lý các hành vi. Và trong một vài trường hợp thì nó cũng có thể xuất hiện như một phần tử HTML với các attribute đặc biệt (cụ thể sẽ nói ở bên dưới).

1. **Using components**
2. **Registration**

Trong Vue.js để khai báo một component ở mức độ global thì chúng ta sử dụng cú pháp:

Vue.component('tag', options);

Trong đó:

* tag là tên mà bạn muốn gán cho component đó.
* options là một object chứa các tham số là template hay dữ liệu kèm theo.

*Lưu ý: Vue.js không thực thi các quy tắc của W3C cho tên của các component tag (tất cả đều là chữ thường, và phải có dấu nối giữa các từ) mặc dù theo như nhận xét thì chuẩn này tốt.*

VD: Đăng ký một component và gọi nó luôn ở trong ứng dụng.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <tdc-component></tdc-component>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        //đăng ký component

        Vue.component('tdc-component',{

            template: '<h1>Hello World!</h1>'

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **Local Registration**

Nếu như ta không muốn đăng ký một component ở trạng thái global nữa thì ta cũng có thể đăng ký nó ở một phạm vi sử dụng nào đó (Vue.js gọi đó là local) thì ta cũng có thể khai báo nó trong Vue instance.

VD: Chuyển component ở ví dụ trên về trạng thái local.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <tdc-component></tdc-component>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#app',

            components: {

                'tdc-component': {

                    template : '<h1>Hello World!</h1>'

                }

            }

        });

    </script>

</body>

</html>

Hoặc cũng có thể gán biến cho gọn

var temp = '<h1>Hello World!</h1>';

1. **DOM Template Parsing Caveats.**

Khi muốn sử dụng DOM như các template, thì ta sẽ phải tuân thủ theo một số các hạn chế lên quan đến cách hoạt động của HTML. Bởi Vue.js sẽ chỉ có thể truy xuất được nội dung khi mà trình duyệt đã xử lý và làm chuẩn hóa chúng (ở đây là các thẻ HTML). Điển hình như các tag ul, ol, table...

VD: Như khi chúng ta sử dụng các tr trong table.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <table>

            <tdc-component></tdc-component>

        </table>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var temp = {

            template : '<tr>Hello World!</tr>'

        };

        var app = new Vue({

            el: '#app',

            components: {

                'tdc-component': temp

            }

        });

    </script>

</body>

</html>

Lúc này trình duyệt nó sẽ render thành:

<tr>Hello World!</tr>

<table></table>

Nhưng chúng ta không cần phải lo, vì cái gì cũng có hướng giải quết của nó, hơn nữa với độ phổ biến của Vue.js hiện giờ thì... Để khắc phục vấn đề trên thì chúng ta sẽ sử dụng is như một attribute của HTML để bind component vào nó.

VD: Khắc phục ví dụ trên.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <table>

            <tr is="tdc-component"></tr>

        </table>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var temp = {

            template : '<tr>Hello World!</tr>'

        };

        var app = new Vue({

            el: '#app',

            components: {

                'tdc-component': temp

            }

        });

    </script>

</body>

</html>

Lúc này thì trình duyệt đã render ra cho chúng ta kết đúng:

<table>

    <tbody>

        <tr>Học Lập Trình Online Toidicode.com</tr>

    </tbody>

</table>

Nhưng Vue.js cũng đã tuyên bố rằng, nó sẽ không khắc phục được nếu như bạn sử dụng template ở các trường hợp sau:

* <script type="text/x-template">
* Javascript inline string.
* .vue component.

(Những cái này sẽ nói sau)

1. **Data must be a function**

Ở phần component thì có khác hơn so với bình thường một chút, là đối với data scope thì nó phải là một function, chứ không phải là một object như ở Vue instance nữa. Nếu như ta cố gắng sử dụng data là object thì nó sẽ có cảnh báo khuyên ta nên khai báo data là một hàm.

Đối với trường hợp data là hàm thì chúng ta cũng sử dụng tương tự như với object.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <tdc-component></tdc-component>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var data = {

            message: 'Hello World!'

        };

        Vue.component('tdc-component', {

            template: '<h1>{{ message }}</h1>',

            data: function() {

                return data;

            }

        });

        var app = new Vue({

            el: '#app'

        });

    </script>

</body>

</html>

Và các ta hãy tham khảo thêm một ví dụ sau nữa với trường hợp gọi một component nhiều lần.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <tdc-component></tdc-component>

        <tdc-component></tdc-component>

        <tdc-component></tdc-component>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var data = {

            count : 0

        };

        Vue.component('tdc-component', {

            template: '<button v-on:click="count += 1">{{ count }}</button>',

            data: function() {

                return data;

            }

        });

        var app = new Vue({

            el: '#app'

        });

    </script>

</body>

</html>

Sau khi chạy thử thì các bạn cũng thấy là khi click vào một button thì data của cả 3 nút đều thay đổi theo. Lý do là do chúng ta khai báo data ở trạng thái global nên khi tác động thì nó sẽ tác động đến cả 3 button. Và để khác phục điều đó thì chúng ta chỉ cần chuyển nó về dạng local là được.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <tdc-component></tdc-component>

        <tdc-component></tdc-component>

        <tdc-component></tdc-component>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('tdc-component', {

            template: '<button v-on:click="count += 1">{{ count }}</button>',

            data: function() {

                return { count : 0 };

            }

        });

        var app = new Vue({

            el: '#app'

        });

    </script>

</body>

</html>

1. **Composing components**

Trong Vue.js, các component được sử dụng cùng nhau, và thông thường nhất chúng ta có thể nhìn thấy là mối quan hệ parent-child (cha con): "Component A có thể sử dụng được componet B ở trong template của riêng nó, và nó không nhất thiết cần phải giao tiếp với nhau mà vẫn có thể gửi thông điệp cho nhau được".

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <component-b></component-b>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('component-a', {

            template: '<font v-bind:color="color">Hello World!</font>',

            data: function () {

                return { color: "red" }

            }

        });

        Vue.component('component-b', {

            template: '<h1><component-a></component-a></h1>',

        });

        var app = new Vue({

            el: '#app'

        });

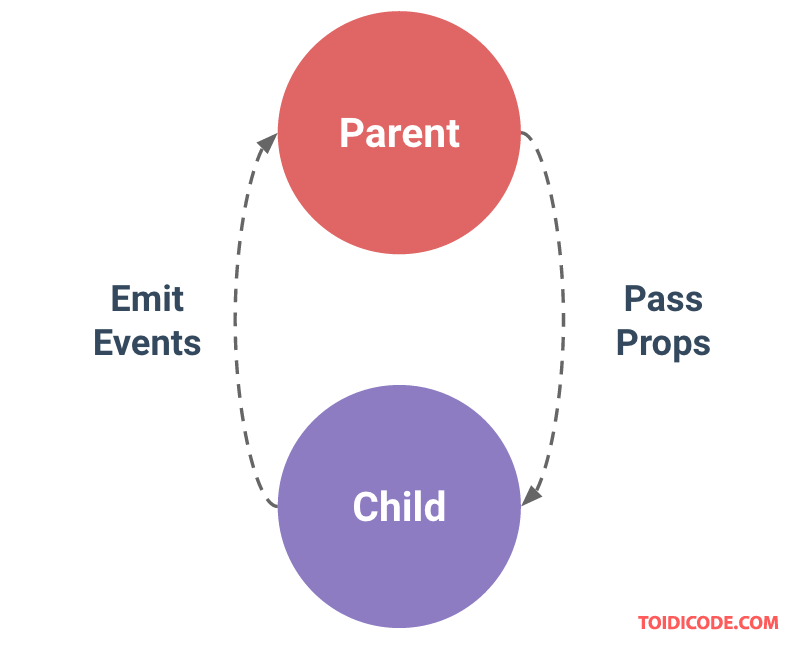
    </script>

</body>

</html>

Tuy nhiên, chúng ta nên tách nó ra các interface khác nhau để tiện cho việc bảo trì và có thể tái sử dụng lại nhiều lần. Và trong Vue.js nó có thể được tóm gọn lại như props down hay events up - parent truyền props cho child và child phản hồi lại cho parent bằng event.

Ta có thể tham khảo mô hình sau để hiểu hơn về nó.



Cụ thể về nó thì phần sau chúng ta sẽ tìm hiểu.

1. **Props.**
2. **Passing Data with Props**

Trong Vue.js thì tất cả các component đều có isolated scope (phạm vi cô lập) riêng. Điều này có nghĩa là ta không thể (và cũng không nên) tham chiếu trực tiếp các dữ liệu ở component cha (parent) vào trong component con (child). Mà thay vào đó chúng ta sẽ phải truyền chúng qua props.

Props ở đây là một attribute tùy chỉnh dùng để truyền data từ component parent đến các component child, và để có thể nhận được dữ liệu từ component parent thì component child phải khai báo prop mà nó mong muốn nhận được từ component parent, bằng cách đặt tên prop ở trong props scope.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child message="Hello World!"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['message'],

            template: '<h1>{{ message }}</h1>',

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **camelCase vs. kebab-case**

Ở trong HTML thì nó sẽ không phân biệt chữ hoa hay chữ thường (có nghĩa là onclick = OnCliCk), nên nếu như ta muốn sự dụng kiểu đặt tên kiểu camelCase cho prop thì ở lúc gọi componet ta phải chuyển đối nó về kebad-case.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child my-say="Hello World!"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['mySay'],

            template: '<h1>{{ mySay }}</h1>',

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **Dynamic Props**

Và cũng tương tự như đối với attribute, ta cũng có thể sử dụng v-bind để ràng buộc prop data trên cấp cha. Và khi dữ liệu ở cấp cha thay đổi thì cấp con cũng thay đổi theo.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <p>

            <input type="text" v-model="message">

        </p>

        <child v-bind:my-say="message"></child>

        <!-- hoặc ta cũng có thể sử dụng cú pháp thu gọn là

            :my-say="message"

         -->

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['mySay'],

            template: '<h1>{{ mySay }}</h1>',

        });

        var app = new Vue({

            el: '#app',

            data: {

                message : null

            }

        });

    </script>

</body>

</html>

1. **Literal vs Dynamic**

Nếu như ta muốn truyền dữ liệu dạng số (number) vào component thì ta sẽ phải sử dụng v-bind, còn nếu bạn dùng prop thông thường thì nó sẽ nhận là string. Hãy xem ví dụ trên để kiểm chứng:

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child my-say="1"></child>

        <child v-bind:my-say="1"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['mySay'],

            template: '<h1>{{ typeof(mySay) }}</h1>',

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **One-Way Data Flow**

Tất cả các props trong Vue.js đề là one-way-down nó dùng để binding data giữa cấp cha và cấp con. Khi dữ liệu ở cấp cha thay đổi thì nó sẽ được binding xuống cấp con. Nhưng khi cấp con thay đổi thì props sẽ không làm thay đổi được cấp cha.

Ngoài ra nếu như dữ liệu ở cấp cha (component parent) thay đổi thì cấp con (component child) sẽ cập nhật và lấy giá trị mới nhất của dữ liệu thay đổi đó. Điều đó tương đương với việc ta không thể nào thay đổi một prop bên trong component con. Nếu như bạn cố tình thực hiện điều đó, thì Vue.js sẽ đưa ra cảnh báo ở trong console.

Nhưng chúng ta hoàn toàn có thể thay đổi nó một cách gián tiếp bằng 2 cách sau:

* Khởi tạo một data scope chứa prop đó, từ đó a có thể thay đổi dữ liệu thông qua data scope.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child my-say="Hello!"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['mySay'],

            data : function () {

                return { dataSay : this.mySay + " This is dataSay!"};

            },

            template: '<h1>{{ dataSay }}</h1>',

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

* Đề xuất prop như một dữ liệu cần được thay đổi.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child my-say="hello!"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            props: ['mySay'],

            computed : {

                convertSay: function () {

                    return this.mySay.trim().toUpperCase();

                }

            },

            template: '<h1>{{ convertSay }}</h1>',

        });

        var app = new Vue({

            el: '#app'

        });

    </script>

</body>

</html>

1. **Prop Validation**

Trong một vài trường hợp mà ta muốn ràng buộc kiểu dữ liệu của một prop, hay xác định giá trị mặc định của nó thid Vue.js cũng hỗ trợ chúng ta điều đó.

Cú pháp xác định kiểu dữ liệu:

props: {

    propName: typeName

}

hoặc

props: {

    propName: [typeName1,typeName2,...typeNamen]

}

hoặc

props: {

    propName: {

      type: typeName

    }

}

Trong đó:

* propName là tên của prop mà bạn muốn ràng buộc.
* typeName là kiểu dữ liệu bạn muốn xác định. Các kiểu dữ liệu gồm có:
* String
* Number
* Boolean
* Function
* Object
* Array
* Symbol

VD:

Vue.component('ex',{

    props: {

        name: String,

        class: [Number, String],

        point: {

            type: Number

        }

    }

});

Hoặc ta cũng có thể tự custom validate cho riêng mình bằng cách sử dụng validator key trong object của prop ta cần ràng buộc.

VD:

Vue.component('ex',{

    props: {

        point: {

            validator: function (value) {

                return value > 10;

            }

        }

    }

});

Và ta cũng có thể thiết lập giá trị mặc định cho prop bằng default scope.

VD:

Vue.component('ex',{

    props: {

        point: {

            type: Number,

            default: 5,

        }

    }

});

Để xác định một prop bắt buộc phải là required thì ta chỉ cần thiết lập required scope thành true là được.

VD:

Vue.component('ex',{

    props: {

        point: {

            type: Number,

            required: true,

        }

    }

});

Nếu như dữ liệu validate không qua, mà bạn đang mode chương trình ở chế độ development thì Vue.js sẽ thông báo cho bạn ở cửa sổ console.

Và dữ liệu trong default scope hay trong validator scope sẽ phải là dữ liệu có sẵn (không dùng được dữ liệu của data, computed, methods) vì validate sẽ được thực hiện trước khi component instance.

1. **Non-prop Attributes.**

Trong Vue.js, Non-Prop attribute là một attribute được chuyển đến component, nhưng nó sẽ không có được tính chất prop đã định nghĩa.

Mặc dù các prop đã được xác định rõ ràng để chuyển thông tin đến child component, nhưng do chính tác giả của thư viện component trong Vue.js cũng không thể lường trước được bối cảnh của chúng ta sử dụng. Đó chính là lý do mà component cho phép chúng ta xác định bất kỳ một attribute nào cũng có thể được xác định trên component parent.

1. **Custom Events.**

Như đã nói ở cuối phần **Composing components** là component cha binding data xuống component con thông qua props và component con thông báo lại cho component cha thông qua các events. Vậy thì ở phần này chúng ta sẽ cùng nghiên cứu xem chúng ta có các cách gửi events nào.

1. **Using v-on**

Trong Vue.js, tất cả các Vue instance đều phải implements một event interface. Đều đó tương đương với việc chúng ta có thể:

* Lắng nghe tất cả các sự kiện thông qua $on(eventName).
* Kích hoạt các sự kiện thông qua $emit(eventName).

*Nhưng các event này không giống như chúng ta sử dụng các eventTarget trên trình duyệt. Có nghĩa là $on sẽ khác với addEventListener và $emit sẽ khác với dispatchEvent.*

Ngoài ra thì các component cha cũng có thể lắng nghe các sự kiện từ component con gửi lên thông qua v-on directive.

Nhưng chúng ta không thể sử dụng $on để lắng nghe các sự kiện được emmited ở component con. Mà phải sử dụng v-on directive như ví dụ sau:

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <p>{{ count }}</p>

        <child v-on:counter="countPlus"></child>

        <child v-on:counter="countPlus"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

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

            data : function () {

                return {count: 0};

            },

            methods: {

                numberPlus: function ()

                {

                    this.count += 1;

                    this.$emit('counter')

                }

            }

        });

        var app = new Vue({

            el: '#app',

            data: {

                count: 0

            },

            methods: {

                countPlus: function ()

                {

                    this.count += 1;

                }

            }

        });

    </script>

</body>

</html>

1. **Binding Native Events to Components.**

Nếu như ta không muốn sử dụng emit name sự kiện mà bạn muốn lắng nghe sự kiện gốc trên thành phần gốc thì ta có thể sử dụng .native modifier.

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <p>{{ count }}</p>

        <child v-on:click.native="countPlus"></child>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            template: '<button>Click</button>'

        });

        var app = new Vue({

            el: '#app',

            data: {

                count: 0

            },

            methods: {

                countPlus: function ()

                {

                    this.count += 1;

                }

            }

        });

    </script>

</body>

</html>

1. **x-template.**

Từ trước đến giờ chúng ta đã được học về khai báo template bằng cách khai báo nó ở trong template scope rồi, nhưng sẽ ra sao nếu template đó không chỉ đơn giản là một hai dòng nữa? Trong Vue.js có cung cấp cho chúng ta khai báo template ở trong cặp thẻ script có type là x-template. Với cú pháp như sau:

<script type="text/x-template" id="template-id">

    //content template

</script>

Trong đó: chúng ta cần xác định id cho template để có thể gọi nó ở trong component template.

Lúc này ở trong template scope ta chỉ cần xác định id của x-template. Với cú pháp:

Vue.component('component-tag', {

    template: '#template-id'

});

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <child></child>

    </div>

    <script type="text/x-template" id="h1-template">

        <h1>Template được khai báo trong x-template</h1>

    </script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        Vue.component('child', {

            template: '#h1-template'

        });

        var app = new Vue({

            el: '#app',

        });

    </script>

</body>

</html>

1. **Transitions & Animation**
2. **Enter/Leave & List Transitions**
3. **Tổng quan**

Vue cung cấp nhiều cách khác nhau để áp dụng các hiệu ứng transition khi các phần tử được thêm vào, thay đổi, hoặc gỡ bỏ khỏi DOM. Điều này bao gồm các công cụ để:

* tự động áp dụng các class CSS cho các transition và animation
* tích hợp các thư viện chuyển động CSS bên thứ ba, ví dụ như Animate.css
* sử dụng JavaScript để trực tiếp thay đổi DOM trong các hook transition
* tích hợp các thư viện chuyển động JavaScript bên thứ ba, ví dụ như Velocity.js

1. **Transitioning Single Elements/Components**

Vue cung cấp một component transition, cho phép ta áp dụng các hiệu ứng transition enter/leave lên các phần tử hoặc component trong các ngữ cảnh sau:

* Render theo điều kiện (sử dụng v-if)
* Hiển thị theo điều kiện (sử dụng v-show)
* Component động
* Root node của component

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="demo">

        <button v-on:click="show = !show">

            Kích hoạt

        </button>

        <transition name="fade">

            <p v-if="show">Xin chào</p>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: true

            }

        });

    </script>

</body>

</html>

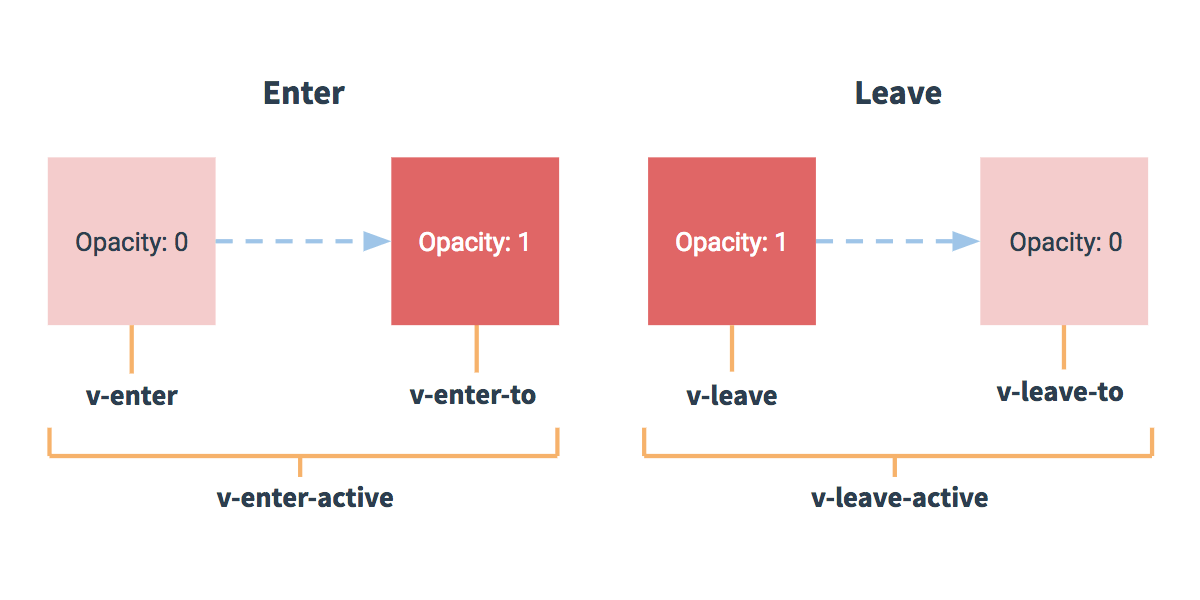
Khi một phần tử chứa trong component transition được thêm vào hoặc gỡ bỏ khỏi DOM, các bước sau đây sẽ diễn ra:

* Vue sẽ tự phát hiện ra nếu phần tử đang nhắc đến có CSS transition hoặc animation và thêm/bớt các class CSS transition vào đúng thời điểm.
* Nếu component cung cấp hook JavaScript, các hook này sẽ được gọi vào đúng thời điểm.
* Trong trường hợp không tìm thấy transition hoặc animation nào trong CSS và cũng không có hook JavaScript nào, việc thêm vào hoặc gỡ bỏ khỏi DOM sẽ được thực thi ngay trong frame tiếp theo (Lưu ý: đây là animation frame của trình duyệt, khác với khái niệm nextTick của Vue).

1. **Transition Classes**

Có tổng cộng 6 class được áp dụng cho enter/leave transition.

* v-enter: Trạng thái bắt đầu của enter. Được áp dụng trước khi phần tử được thêm vào DOM và gỡ bỏ đi một frame sau đó.
* v-enter-active: Trạng thái active của enter. Được áp dụng trong suốt quá trình enter, từ ngay sau khi phần tử được thêm vào DOM cho đến khi transition/animation kết thúc. Class này có thể được dùng để định nghĩa duration, delay, và hàm easing cho transition enter.
* v-enter-to: 2.1.8+. Trạng thái kết thúc của enter. Áp dụng một frame sau khi element được thêm vào DOM (cùng lúc với việc v-enter được gỡ bỏ), gỡ bỏ đi khi transition/animation kết thúc.
* v-leave: Trạng thái bắt đầu của leave. Được áp dụng ngay khi một leave transition được kích hoạt và gỡ bỏ đi một frame sau đó.
* v-leave-active: Trạng thái active của leave. Được áp dụng trong suốt quá trình leave, từ khi transition được kích hoạt cho đến khi transition/animation kết thúc. Class này có thể được dùng để định nghĩa duration, delay, và hàm easing cho leave transition.
* v-leave-to: 2.1.8+. Trạng thái kết thúc của leave. Áp dụng một frame sau khi leave transition được kích hoạt (cùng lúc với việc v-leave được gỡ bỏ), gỡ bỏ đi khi transition/animation kết thúc.



1. **CSS Transitions**

CSS transition là một trong những transition thông dụng nhất. Ví dụ:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    /\*

        Animation cho enter và leave có thể có giá trị

        duration và timing function khác nhau.

    \*/

    .slide-fade-enter-active {

    transition: all .3s ease;

    }

    .slide-fade-leave-active {

    transition: all .8s cubic-bezier(1.0, 0.5, 0.8, 1.0);

    }

    .slide-fade-enter, .slide-fade-leave-to

    /\* Trước 2.1.8 thì dùng .slide-fade-leave-active \*/ {

    transform: translateX(10px);

    opacity: 0;

    }

</style>

<body>

    <div id="demo">

        <button @click="show = !show">

            Kích hoạt

        </button>

        <transition name="slide-fade">

            <p v-if="show">Xin chào</p>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: true

            }

        });

    </script>

</body>

</html>

1. **CSS Animations**

CSS animation được áp dụng cùng một cách như CSS transition. Điểm khác nhau là v-enter không được gỡ bỏ ngay lập tức sau khi phần tử được thêm vào DOM mà là khi sự kiện animationend được phát ra.

Đây là một ví dụ (chúng ta sẽ bỏ đi các prefix CSS cho gọn):

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .bounce-enter-active {

    animation: bounce-in .5s;

    }

    .bounce-leave-active {

    animation: bounce-in .5s reverse;

    }

    @keyframes bounce-in {

        0% {

            transform: scale(0);

        }

        50% {

            transform: scale(1.5);

        }

        100% {

            transform: scale(1);

        }

    }

</style>

<body>

    <div id="demo">

        <button @click="show = !show">Kích hoạt</button>

        <transition name="bounce">

            <p v-if="show">Cân đẩu vân</p>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: true

            }

        });

    </script>

</body>

</html>

1. **Custom Transition Classes**

Bạn cũng có thể chỉ định các class tùy biến cho transition bằng cách cung cấp các thuộc tính sau đây:

* enter-class
* enter-active-class
* enter-to-class (2.1.8+)
* leave-class
* leave-active-class
* leave-to-class (2.1.8+)

Các thuộc tính này sẽ override những tên class theo thông lệ của Vue. Điều này đặc biệt có ích khi bạn muốn kết hợp giữa hệ thống transition của Vue và một thư viện CSS animation có sẵn như Animate.css.

Đây là một ví dụ:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">

<style type="text/css">

</style>

<body>

    <div id="demo">

        <button @click="show = !show">

            Kích hoạt

        </button>

        <transition

        name="custom-classes-transition"

        enter-active-class="animated tada"

        leave-active-class="animated bounceOutRight"

        >

        <p v-if="show">Thú nhún</p>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: true

            }

        });

    </script>

</body>

</html>

1. **Using Transitions and Animations Together**

Vue cần phải đính kèm một sự kiện để có thể biết khi nào thì một chuyển động kết thúc. Sự kiện này có thể là transitionend hoặc animationend, tùy thuộc vào các rule CSS được áp dụng. Nếu bạn chỉ sử dụng animation hoặc transition, Vue có thể tự phát hiện kiểu chuyển động (animation hay transition).

Tuy nhiên, trong một số trường hợp có thể bạn muốn dùng cả transition và animation trên cùng một phần tử, ví dụ sử dụng Vue để kích hoạt một CSS animation, đồng thời áp dụng một hiệu ứng CSS transition khi hover. Trong những trường hợp này, bạn sẽ phải khai báo rõ kiểu chuyển động bạn muốn Vue xử lí bằng cách dùng một thuộc tính type với giá trị là animation hoặc transition.

1. **Explicit Transition Durations**

Trong đa số các trường hợp, Vue có thể tự biết được khi nào một transition kết thúc. Mặc định, Vue đợi cho sự kiện transitionend hoặc animationend của phần tử transition gốc được phát ra. Tuy nhiên, không phải lúc nào đây cũng là điều bạn muốn – ví dụ, chúng ta có thể có một chuỗi các transition nối tiếp nhau, trong đó một số phần tử bên trong có transition được trì hoãn (delay) hoặc kéo dài lâu hơn transition của phần tử transition gốc.

Trong những trường hợp như vậy, bạn có thể chỉ định một cách tường minh thời lượng (duration) của transition (với đơn vị là mili giây) dùng thuộc tính duration trên component <transition>:

<transition :duration="1000">...</transition>

Bạn cũng có thể chỉ định hai giá trị tách biệt nhau cho thời lượng enter và leave:

<transition :duration="{ enter: 500, leave: 800 }">...</transition>

1. **JavaScript Hooks**

Ta cũng có thể định nghĩa các hook JavaScript trong các thuộc tính của component <transition>:

<transition

  v-on:before-enter="beforeEnter"

  v-on:enter="enter"

  v-on:after-enter="afterEnter"

  v-on:enter-cancelled="enterCancelled"

  v-on:before-leave="beforeLeave"

  v-on:leave="leave"

  v-on:after-leave="afterLeave"

  v-on:leave-cancelled="leaveCancelled"

>

  <!-- ... -->

</transition>

// ...

methods: {

    // -----

    // ENTER

    // -----

    beforeEnter: function (el) {

      // ...

    },

    // callback `done()` là không bắt buộc

    // khi sử dụng cùng với CSS

    enter: function (el, done) {

      // ...

      done()

    },

    afterEnter: function (el) {

      // ...

    },

    enterCancelled: function (el) {

      // ...

    },

    // -----

    // LEAVE

    // -----

    beforeLeave: function (el) {

      // ...

    },

    // callback `done()` là không bắt buộc

    // khi sử dụng cùng với CSS

    leave: function (el, done) {

      // ...

      done()

    },

    afterLeave: function (el) {

      // ...

    },

    // leaveCancelled chỉ hoạt động với v-show

    leaveCancelled: function (el) {

      // ...

    }

  }

Những hook này có thể được sử dụng độc lập hoặc dùng chung với CSS transition/animation.

* Khi sử dụng các transition JavaScript, hàm callback done là bắt buộc đối với các hook enter và leave. Trong các trường hợp khác, các hook này sẽ được gọi một cách đồng bộ và transition sẽ kết thúc ngay lập tức.
* Bạn cũng nên chỉ định rõ v-bind:css="false" cho các transition JavaScript để Vue có thể bỏ qua phần dò tìm CSS. Việc này cũng ngăn không cho các rule trong CSS can thiệp vào transition.

Ví dụ một transition JavaScript sử dụng Velocity.js:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="demo">

        <button @click="show = !show">

            Kích hoạt

        </button>

        <transition

            v-on:before-enter="beforeEnter"

            v-on:enter="enter"

            v-on:leave="leave"

            v-bind:css="false"

        >

        <p v-if="show">

            Demo

        </p>

        </transition>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: false

            },

            methods: {

                beforeEnter: function (el) {

                el.style.opacity = 0

                },

                enter: function (el, done) {

                    Velocity(el, { opacity: 1, fontSize: '1.4em' }, { duration: 300 })

                    Velocity(el, { fontSize: '1em' }, { complete: done })

                    },

                    leave: function (el, done) {

                    Velocity(el, { translateX: '15px', rotateZ: '50deg' }, { duration: 600 })

                    Velocity(el, { rotateZ: '100deg' }, { loop: 2 })

                    Velocity(el, {

                        rotateZ: '45deg',

                        translateY: '30px',

                        translateX: '30px',

                        opacity: 0

                    }, { complete: done })

                }

            }

        });

    </script>

</body>

</html>

1. **Transitions on Initial Render**

Nếu muốn áp dụng một transition ngay trong lần render đầu tiên của một node, bạn có thể dùng thuộc tính appear:

<transition appear>

    <!-- ... -->

</transition>

Mặc định, các transition được chỉ định khi enter và leave sẽ được sử dụng. Nếu muốn, bạn cũng có thể dùng các class CSS tùy biến:

<transition

  appear

  appear-class="custom-appear-class"

  appear-to-class="custom-appear-to-class" (2.1.8+)

  appear-active-class="custom-appear-active-class"

>

  <!-- ... -->

</transition>

và các hook JavaScript tùy biến:

<transition

  appear

  v-on:before-appear="customBeforeAppearHook"

  v-on:appear="customAppearHook"

  v-on:after-appear="customAfterAppearHook"

  v-on:appear-cancelled="customAppearCancelledHook"

>

  <!-- ... -->

</transition>

1. **Transitioning Between Elements**

Ta cũng có thể transition giữa các phần tử thô (raw element) sử dụng v-if/v-else

<transition>

    <table v-if="searchResults.length > 0">

      <!-- ... -->

    </table>

    <p v-else>Không tìm ra kết quả nào.</p>

</transition>

*Hãy luôn dùng key cho các item trong một component <transition>*

VD:

<transition>

    <button v-if="isEditing" key="save">

      Lưu

    </button>

    <button v-else key="edit">

      Sửa

    </button>

</transition>

Ta có thể dùng thuộc tính key cho việc chuyển tiếp giữa các trạng thái khác nhau trong cùng một phần tử. Thay vì dùng v-if và v-else, ví dụ trên có thể được viết lại như sau:

<transition>

    <button v-bind:key="isEditing">

      {{ isEditing ? 'Lưu' : 'Sửa' }}

    </button>

</transition>

**Các chế độ transition**

VD:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

.no-mode-fade-enter-active, .no-mode-fade-leave-active {

  transition: opacity .5s

}

.no-mode-fade-enter, .no-mode-fade-leave-active {

  opacity: 0

}

</style>

<body>

    <div id="no-mode-demo">

        <transition name="no-mode-fade">

            <button @click="on = !on" v-bind:key="on">

                {{ on ? 'On' : 'Off' }}

            </button>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#no-mode-demo',

            data: {

                on: false

            }

        });

    </script>

</body>

</html>

Tuy nhiên, không phải lúc nào chúng ta cũng muốn transition enter và leave xảy ra đồng thời. Vì thế, Vue cung cấp thêm một số chế độ transition thay thế:

* in-out: Transition đi vào (in) của phần tử mới xảy ra trước, và sau khi hoàn tất mới đến lượt transition đi ra (out) của phần tử hiện tại.
* out-in: Transition đi ra (out) của phần tử hiện tại xảy ra trước, và sau khi hoàn tất mới đến lượt transition đi vào (in) của phần tử mới.

Bây giờ chúng ta thử cập nhật transition cho các nút bật/tắt với out-in:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

.no-mode-fade-enter-active, .no-mode-fade-leave-active {

  transition: opacity .5s

}

.no-mode-fade-enter, .no-mode-fade-leave-active {

  opacity: 0

}

</style>

<body>

    <div id="no-mode-demo">

        <transition name="no-mode-fade" mode="out-in">

            <button @click="on = !on" v-bind:key="on">

                {{ on ? 'On' : 'Off' }}

            </button>

        </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#no-mode-demo',

            data: {

                on: false

            }

        });

    </script>

</body>

</html>

VD in-out:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

.in-out-translate-demo-wrapper {

  position: relative;

  height: 18px;

}

.in-out-translate-demo-wrapper button {

  position: absolute;

}

.in-out-translate-fade-enter-active, .in-out-translate-fade-leave-active {

  transition: all .5s;

}

.in-out-translate-fade-enter, .in-out-translate-fade-leave-active {

  opacity: 0;

}

.in-out-translate-fade-enter {

  transform: translateX(31px);

}

.in-out-translate-fade-leave-active {

  transform: translateX(-31px);

}

</style>

<body>

    <div id="in-out-translate-demo">

        <div class="in-out-translate-demo-wrapper">

            <transition name="in-out-translate-fade" mode="in-out">

                <button @click="on = !on" v-bind:key="on">

                    {{ on ? 'On' : 'Off' }}

                </button>

            </transition>

        </div>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#in-out-translate-demo',

            data: {

                on: false

            }

        });

    </script>

</body>

</html>

1. **Transitioning Between Components**

Transition giữa các component lại càng đơn giản hơn - chúng ta còn không cần dùng thuộc tính key. Thay vào đó, chúng ta wrap một component động:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .component-fade-enter-active, .component-fade-leave-active {

    transition: opacity .3s ease;

    }

    .component-fade-enter, .component-fade-leave-to {

    opacity: 0;

    }

</style>

<body>

    <div id="demo">

            <input type="radio" value="v-a" id="a" name="view" v-model="view">

            <label for="a">A</label>

            <input type="radio" value="v-b" id="b" name="view" v-model="view">

            <label for="b">B</label>

            <transition name="component-fade" mode="out-in">

                <component v-bind:is="view"></component>

            </transition>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                view: 'v-a'

            },

            components: {

                'v-a': {

                template: '<div>Component A</div>'

                },

                'v-b': {

                template: '<div>Component B</div>'

                }

            }

        });

    </script>

</body>

</html>

1. **List Transitions**

Chúng ta đã nói về transition cho:

* Các node đơn
* Nhiều node khác nhau nhưng chỉ có một node được render mỗi lúc

Vậy nếu chúng ta có một danh sách chứa các item mà ta muốn render đồng thời, ví dụ với v-for, thì sao? Trong trường hợp này, ta sẽ dùng component <transition-group>. Trước khi xem ví dụ, có một số điều quan trọng mà bạn cần biết về component này:

* Khác với <transition>, <transition-group> render một phần tử thật sự, mặc định là <span>. Bạn có thể thay đổi kiểu phần tử được render ra bằng thuộc tính tag.
* Các phần tử bên trong <transition-group> bắt buộc phải có thuộc tính key duy nhất

1. **List Entering/Leaving Transitions**

Ví dụ về transition cho enter/leave với cùng các class CSS mà ta đã dùng trên đây:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .list-item {

        display: inline-block;

        margin-right: 10px;

    }

    .list-enter-active, .list-leave-active {

        transition: all 1s;

    }

    .list-enter, .list-leave-to /\* .list-leave-active below version 2.1.8 \*/ {

        opacity: 0;

        transform: translateY(30px);

    }

</style>

<body>

    <div id="demo">

        <button v-on:click="add">Thêm</button>

        <button v-on:click="remove">Bớt</button>

        <transition-group name="list" tag="p">

          <span v-for="item in items" v-bind:key="item" class="list-item">

            {{ item }}

          </span>

        </transition-group>

    </div>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                items: [1,2,3,4,5,6,7,8,9],

                nextNum: 10

            },

            methods: {

                randomIndex: function () {

                    return Math.floor(Math.random() \* this.items.length)

                },

                add: function () {

                    this.items.splice(this.randomIndex(), 0, this.nextNum++)

                },

                remove: function () {

                    this.items.splice(this.randomIndex(), 1)

                },

            }

        });

    </script>

</body>

</html>

Có một vấn đề với ví dụ này: Khi bạn thêm hoặc bớt một item, những item xung quanh nó ngay lập tức nhảy đến vị trí mới thay vì có hiệu ứng như item vừa thêm. Chúng ta sẽ giải quyết vấn đề này sau.

1. **List Move Transitions**

Component <transition-group> có một tính năng nữa. Không chỉ có thể animate enter và leave, nó còn có thể animate vị trí của các item. Khái niệm mới duy nhất bạn cần biết đến để sử dụng tính năng này là class v-move, được thêm vào khi item thay đổi vị trí. Tương tự như các class khác, prefix của v-mode chính là thuộc tính name, và bạn có thể sử dụng một class khác với thuộc tính move-class.

Class này có ích nhất là để chỉ định thời lượng và hàm easing cho transition, như có thể thấy sau đây:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .flip-list-move {

        transition: transform 1s;

    }

</style>

<body>

    <div id="demo">

        <button v-on:click="shuffle">Xáo trộn</button>

        <transition-group name="flip-list" tag="ul">

            <li v-for="item in items" v-bind:key="item">

            {{ item }}

            </li>

        </transition-group>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.14.1/lodash.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                items: [1,2,3,4,5,6,7,8,9]

            },

            methods: {

                shuffle: function () {

                this.items = \_.shuffle(this.items)

                }

            }

        });

    </script>

</body>

</html>

Trở lại ví dụ đầu tiên:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .list-complete-item {

        transition: all 1s;

        display: inline-block;

        margin-right: 10px;

    }

    .list-complete-enter, .list-complete-leave-to

        /\* .list-complete-leave-active đối với các phiên bản trước 2.1.8 \*/ {

        opacity: 0;

        transform: translateY(30px);

    }

    .list-complete-leave-active {

        position: absolute;

    }

</style>

<body>

    <div id="demo">

        <button v-on:click="shuffle">Xáo trộn</button>

        <button v-on:click="add">Thêm</button>

        <button v-on:click="remove">Bớt</button>

        <transition-group name="list-complete" tag="p">

            <span

                v-for="item in items"

                v-bind:key="item"

                class="list-complete-item"

            >

            {{ item }}

            </span>

        </transition-group>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.14.1/lodash.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                items: [1,2,3,4,5,6,7,8,9],

                nextNum: 10

            },

            methods: {

                randomIndex: function () {

                return Math.floor(Math.random() \* this.items.length)

                },

                add: function () {

                this.items.splice(this.randomIndex(), 0, this.nextNum++)

                },

                remove: function () {

                this.items.splice(this.randomIndex(), 1)

                },

                shuffle: function () {

                this.items = \_.shuffle(this.items)

                }

            }

        });

    </script>

</body>

</html>

*Một điểm quan trọng cần lưu ý là các transition FLIP này không hoạt động đối với các phần tử có display: inline. Để thay thế, bạn có thể dùng display: inline-block hoặc đặt trong một flexbox.*

FLIP animation cũng không bị giới hạn chỉ trong một trục. Item trong một grid (lưới) đa chiều cũng có thể được animate:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

    .container {

        display: flex;

        flex-wrap: wrap;

        width: 238px;

        margin-top: 10px;

    }

    .cell {

        display: flex;

        justify-content: space-around;

        align-items: center;

        width: 25px;

        height: 25px;

        border: 1px solid #aaa;

        margin-right: -1px;

        margin-bottom: -1px;

    }

    .cell:nth-child(3n) {

        margin-right: 0;

    }

    .cell:nth-child(27n) {

        margin-bottom: 0;

    }

    .cell-move {

        transition: transform 1s;

    }

</style>

<body>

    <div id="demo">

        <h1>Lazy Sudoku</h1>

        <p>Keep hitting the shuffle button until you win.</p>

        <button @click="shuffle">

            Shuffle

        </button>

        <transition-group name="cell" tag="div" class="container">

            <div v-for="cell in cells" :key="cell.id" class="cell">

            {{ cell.number }}

            </div>

        </transition-group>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.14.1/lodash.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                cells: Array.apply(null, { length: 81 })

                    .map(function (\_, index) {

                    return {

                        id: index,

                        number: index % 9 + 1

                    }

                })

            },

            methods: {

                shuffle: function () {

                    this.cells = \_.shuffle(this.cells)

                }

            }

        });

    </script>

</body>

</html>

1. **Staggering List Transitions**

Bằng cách giao tiếp với các transition JavaScript thông qua các thuộc tính dữ liệu, ta có thể khiến cho các transition trong một danh sách diễn ra một cách tuần tự:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="demo">

        <input v-model="query">

        <transition-group

            name="staggered-fade"

            tag="ul"

            v-bind:css="false"

            v-on:before-enter="beforeEnter"

            v-on:enter="enter"

            v-on:leave="leave"

        >

            <li

                v-for="(item, index) in computedList"

                v-bind:key="item.msg"

                v-bind:data-index="index"

            >{{ item.msg }}</li>

        </transition-group>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                query: '',

                list: [

                { msg: 'Hai Bà Trưng' },

                { msg: 'Ngô Quyền' },

                { msg: 'Đinh Tiên Hoàng' },

                { msg: 'Lý Thường Kiệt' },

                { msg: 'Trần Hưng Đạo' }

                ]

            },

            computed: {

                computedList: function () {

                    var vm = this

                    return this.list.filter(function (item) {

                        return item.msg.toLowerCase().indexOf(vm.query.toLowerCase()) !== -1

                    })

                }

            },

            methods: {

                beforeEnter: function (el) {

                    el.style.opacity = 0

                    el.style.height = 0

                },

                enter: function (el, done) {

                    var delay = el.dataset.index \* 150

                    setTimeout(function () {

                        Velocity(

                            el,

                            { opacity: 1, height: '1.6em' },

                            { complete: done }

                        )

                    }, delay)

                },

                leave: function (el, done) {

                    var delay = el.dataset.index \* 150

                    setTimeout(function () {

                        Velocity(

                            el,

                            { opacity: 0, height: 0 },

                            { complete: done }

                        )

                    }, delay)

                }

            }

        });

    </script>

</body>

</html>

1. **Reusable Transitions**

Transition có thể được tái sử dụng trong hệ thống component của Vue. Để tạo ra một transition sử dụng lại được, bạn chỉ cần đặt một component <transition> hoặc <transition-group> ở phần tử root, sau đó truyền bất cứ component con nào vào.

Đây là một ví dụ sử dụng một component template:

Vue.component('my-special-transition', {

    template: '\

      <transition\

        name="very-special-transition"\

        mode="out-in"\

        v-on:before-enter="beforeEnter"\

        v-on:after-enter="afterEnter"\

      >\

        <slot></slot>\

      </transition>\

    ',

    methods: {

      beforeEnter: function (el) {

        // ...

      },

      afterEnter: function (el) {

        // ...

      }

    }

})

Và component chức năng (functional component) đặc biệt phù hợp cho nhiệm vụ này:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="demo">

        <input v-model="query">

        <transition-group

            name="staggered-fade"

            tag="ul"

            v-bind:css="false"

            v-on:before-enter="beforeEnter"

            v-on:enter="enter"

            v-on:leave="leave"

        >

            <li

                v-for="(item, index) in computedList"

                v-bind:key="item.msg"

                v-bind:data-index="index"

            >{{ item.msg }}</li>

        </transition-group>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                query: '',

                list: [

                { msg: 'Hai Bà Trưng' },

                { msg: 'Ngô Quyền' },

                { msg: 'Đinh Tiên Hoàng' },

                { msg: 'Lý Thường Kiệt' },

                { msg: 'Trần Hưng Đạo' }

                ]

            },

            computed: {

                computedList: function () {

                    var vm = this

                    return this.list.filter(function (item) {

                        return item.msg.toLowerCase().indexOf(vm.query.toLowerCase()) !== -1

                    })

                }

            },

            methods: {

                beforeEnter: function (el) {

                    el.style.opacity = 0

                    el.style.height = 0

                },

                enter: function (el, done) {

                    var delay = el.dataset.index \* 150

                    setTimeout(function () {

                        Velocity(

                            el,

                            { opacity: 1, height: '1.6em' },

                            { complete: done }

                        )

                    }, delay)

                },

                leave: function (el, done) {

                    var delay = el.dataset.index \* 150

                    setTimeout(function () {

                        Velocity(

                            el,

                            { opacity: 0, height: 0 },

                            { complete: done }

                        )

                    }, delay)

                }

            }

        });

    </script>

</body>

</html>

1. **Dynamic Transitions**

Đúng, ngay cả transition trong Vue cũng hướng dữ liệu! Ví dụ cơ bản nhất của một transition động bind thuộc tính name vào một property động.

<transition v-bind:name="transitionName">

    <!-- ... -->

</transition>

Điều này có thể có ích khi bạn đã định nghĩa transition/animation CSS với quy chuẩn đặt tên cho class transition của Vue và muốn hoán chuyển giữa các transition/animation này.

Tuy nhiên, sự thật thì bất kì thuộc tính transition nào cũng có thể được bind động. Và không chỉ có thuộc tính. Vì thật ra chỉ là các phương thức, các hook cho sự kiện có thể truy xuất đến bất kì dữ liệu nào trong ngữ cảnh hiện tại. Điều này có nghĩa là tùy vào trạng thái của component, các transition JavaScript của bạn có thể có các hành vi rất khác nhau.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="demo">

        Thời gian hiện ra: <input type="range" v-model="fadeInDuration" min="0" v-bind:max="maxFadeDuration">

        Thời gian mờ đi: <input type="range" v-model="fadeOutDuration" min="0" v-bind:max="maxFadeDuration">

        <transition

          v-bind:css="false"

          v-on:before-enter="beforeEnter"

          v-on:enter="enter"

          v-on:leave="leave"

        >

          <p v-if="show">Xin chào</p>

        </transition>

        <button

          v-if="stop"

          v-on:click="stop = false; show = false"

        >Bắt đầu</button>

        <button

          v-else

          v-on:click="stop = true"

        >Dừng lại</button>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#demo',

            data: {

                show: true,

                fadeInDuration: 1000,

                fadeOutDuration: 1000,

                maxFadeDuration: 1500,

                stop: true

            },

            mounted: function () {

                this.show = false

            },

            methods: {

                beforeEnter: function (el) {

                    el.style.opacity = 0

                },

                enter: function (el, done) {

                    var vm = this

                    Velocity(el,

                        { opacity: 1 },

                        {

                        duration: this.fadeInDuration,

                            complete: function () {

                                done()

                                if (!vm.stop) vm.show = false

                            }

                        }

                    )

                },

                leave: function (el, done) {

                    var vm = this

                    Velocity(el,

                        { opacity: 0 },

                        {

                            duration: this.fadeOutDuration,

                            complete: function () {

                                done()

                                vm.show = true

                            }

                        }

                    )

                }

            }

        });

    </script>

</body>

</html>

1. **State Transitions**
2. **Animating State with Watchers**

Watcher cho phép chúng ta animate các thay đổi từ bất kì thuộc tính dạng số nào sang một thuộc tính khác. Điều này nói một cách trừu tượng thì nghe có vẻ phức tạp, vì thế chúng ta hãy xem một ví dụ với Greensock:

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="animated-number-demo">

        <input v-model.number="number" type="number" step="20">

        <p>{{ animatedNumber }}</p>

    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/1.20.3/TweenMax.min.js"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var app = new Vue({

            el: '#animated-number-demo',

            data: {

                number: 0,

                tweenedNumber: 0

            },

            computed: {

                animatedNumber: function() {

                return this.tweenedNumber.toFixed(0);

                }

            },

            watch: {

                number: function(newValue) {

                TweenLite.to(this.$data, 0.5, { tweenedNumber: newValue });

                }

            }

        });

    </script>

</body>

</html>

Ví dụ CSS (tích hợp với Tween.js và Color.js):

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

.app-color-preview {

  display: inline-block;

  width: 50px;

  height: 50px;

}

</style>

<body>

    <div id="app">

        <input

            v-model="colorQuery"

            v-on:keyup.enter="updateColor"

            placeholder="Nhập vào một màu"

        >

        <p>

            <span

            v-bind:style="{ backgroundColor: tweenedCSSColor }"

            class="app-color-preview"

            ></span>

        </p>

        <p>{{ tweenedCSSColor }}</p>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/tween.js@16.3.4"></script>

    <script src="https://cdn.jsdelivr.net/npm/color-js@1.0.3"></script>

    <script src="https://unpkg.com/vue@2.4.2" type="text/javascript"></script>

    <script type="text/javascript">

        var Color = net.brehaut.Color

        var app = new Vue({

            el: '#app',

            data: {

                colorQuery: '',

                color: {

                red: 0,

                green: 0,

                blue: 0,

                alpha: 1

                },

                tweenedColor: {}

            },

            created: function () {

                this.tweenedColor = Object.assign({}, this.color)

            },

            watch: {

                color: function () {

                function animate () {

                    if (TWEEN.update()) {

                    requestAnimationFrame(animate)

                    }

                }

                new TWEEN.Tween(this.tweenedColor)

                    .to(this.color, 750)

                    .start()

                animate()

                }

            },

            computed: {

                tweenedCSSColor: function () {

                return new Color({

                    red: this.tweenedColor.red,

                    green: this.tweenedColor.green,

                    blue: this.tweenedColor.blue,

                    alpha: this.tweenedColor.alpha

                }).toCSS()

                }

            },

            methods: {

                updateColor: function () {

                this.color = new Color(this.colorQuery).toRGB()

                this.colorQuery = ''

                }

            }

        });

    </script>

</body>

</html>

1. **Dynamic State Transitions**

Cũng giống như component transition của Vue, transition cho dữ liệu cũng có thể được cập nhật trong thời gian thực (real time), và việc này đặc biệt hữu ích khi tạo các prototype (bản thử nghiệm, khuôn mẫu):

VD: <https://jsfiddle.net/chrisvfritz/65gLu2b6/>

1. **Organizing Transitions into Components**

Quản lí nhiều transition cho trạng thái có thể làm độ phức tạp của một đối tượng hoặc component Vue tăng lên một cách nhanh chóng. May thay, nhiều animation có thể được trích xuất ra thành các component con chuyên dụng.

<!DOCTYPE html>

<html>

<head>

    <meta charset="utf-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <title>Vue.js</title>

</head>

<style type="text/css">

</style>

<body>

    <div id="app">

        <input v-model.number="firstNumber" type="number" step="20"> +

        <input v-model.number="secondNumber" type="number" step="20"> =

        {{ result }}

        <p>

            <animated-integer v-bind:value="firstNumber"></animated-integer> +

            <animated-integer v-bind:value="secondNumber"></animated-integer> =

            <animated-integer v-bind:value="result"></animated-integer>

        </p>

    </div>

    <script src="https://cdn.jsdelivr.net/npm/tween.js@16.3.4"></script>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js" type="text/javascript"></script>

    <script type="text/javascript">

    Vue.component('animated-integer', {

        template: '<span>{{ tweeningValue }}</span>',

        props: {

            value: {

            type: Number,

            required: true

            }

        },

        data: function () {

            return {

            tweeningValue: 0

            }

        },

        watch: {

            value: function (newValue, oldValue) {

            this.tween(oldValue, newValue)

            }

        },

        mounted: function () {

            this.tween(0, this.value)

        },

        methods: {

            tween: function (startValue, endValue) {

            var vm = this

            function animate () {

                if (TWEEN.update()) {

                requestAnimationFrame(animate)

                }

            }

            new TWEEN.Tween({ tweeningValue: startValue })

                .to({ tweeningValue: endValue }, 500)

                .onUpdate(function () {

                vm.tweeningValue = this.tweeningValue.toFixed(0)

                })

                .start()

            animate()

            }

        }

    })

    new Vue({

        el: '#app',

        data: {

            firstNumber: 20,

            secondNumber: 40

        },

        computed: {

            result: function () {

            return this.firstNumber + this.secondNumber

            }

        }

    })

    </script>

</body>

</html>

1. **Bringing Designs to Life**

Theo một định nghĩa, animate nghĩa là “(làm gì đó) trở nên sinh động.” Tiếc thay, các icon, logo và mascot (linh vật) thường chỉ được tạo ra dưới dạng hình ảnh hoặc SVG tĩnh.

Vue có thể giúp bạn. Vì SVG thực chất chỉ là dữ liệu, chúng ta chỉ cần biết các sinh vật này trông như thế nào khi hứng khởi, khi đang suy nghĩ, hoặc khi sợ hãi. Sau đó Vue có thể giúp transition giữa các trạng thái này và làm cho ứng dụng của bạn trở nên sinh động hơn.

VD: <https://codepen.io/sdras/pen/YZBGNp>

1. **Tái sử dụng & kết hợp**
2. **Mixins**
3. **Basics**

Mixin là một cách linh hoạt để phân phối những tính năng tái sử dụng được cho component. Một object mixin có thể chứa bất kì những tùy chọn nào của component.

VD:

// định nghĩa một object mixin

var myMixin = {

    created: function () {

      this.hello()

    },

    methods: {

      hello: function () {

        console.log('Mixin xin chào!')

      }

    }

  }

  // định nghĩa một component sử dụng mixin này

  var Component = Vue.extend({

    mixins: [myMixin]

  })

  var component = new Component() // => "Mixin xin chào!"

1. **Option Merging**

Khi một mixin và component chứa những tùy chọn trùng nhau, những tùy chọn này sẽ được merge (hợp nhất) bằng cách sử dụng các chiến lược thích hợp.

Ví dụ, những object data sẽ được merge sâu một cấp (shallow merge), và khi có xung đột thì data của component sẽ được ưu tiên.

var mixin = {

    data: function () {

      return {

        message: 'chào anh',

        foo: 'abc'

      }

    }

  }

  new Vue({

    mixins: [mixin],

    data: function () {

      return {

        message: 'chào chị',

        bar: 'def'

      }

    },

    created: function () {

      console.log(this.$data)

      // => { message: "chào chị", foo: "abc", bar: "def" }

    }

})

Những hàm hook trùng tên với nhau được merge vào trong một mảng để tất cả những hàm hook đó đều được gọi. Thêm vào đó, những hook trong mixin sẽ được gọi trước những hook trong component:

var mixin = {

    created: function () {

      console.log('hook trong mixin sẽ được gọi trước')

    }

}

  new Vue({

    mixins: [mixin],

    created: function () {

      console.log('sau đó đến hook của component')

    }

})

// => "hook trong mixin sẽ được gọi trước"

// => "sau đó đến hook của component"

Những tùy chọn có giá trị là object như methods, components và directives sẽ được merge vào chung một object. Nếu khóa (key) của các tùy chọn trùng nhau thì tùy chọn của component sẽ được ưu tiên:

var mixin = {

    methods: {

      foo: function () {

        console.log('foo')

      },

      hàmTrùngTên: function () {

        console.log('từ mixin')

      }

    }

}

  var vm = new Vue({

    mixins: [mixin],

    methods: {

      bar: function () {

        console.log('bar')

      },

      hàmTrùngTên: function () {

        console.log('từ component')

      }

    }

})

vm.foo() // => "foo"

vm.bar() // => "bar"

vm.hàmTrùngTên() // => "từ component"

*Chú ý rằng cách merge này cũng được dùng trong Vue.extend()*

1. **Global Mixin**

Bạn cũng có thể áp dụng một mixin ở cấp toàn cục. Nhưng cần cẩn trọng khi sử dụng cách này, vì một mixin toàn cục sẽ tác động tới tất cả đối tượng Vue được khởi tạo sau đó. Khi được dùng một cách thích hợp, mixin có thể được sử dụng để chèn những xử lí logic cho các tùy chọn tùy biến:.

// chèn một handler cho option tùy biến `myOption`

Vue.mixin({

    created: function () {

      var myOption = this.$options.myOption

      if (myOption) {

        console.log(myOption)

      }

    }

})

new Vue({

    myOption: 'hello!'

})

// => "hello!"

*Nên hạn chế và cẩn trọng khi sử dụng mixin toàn cục, bởi vì nó sẽ tác động tới tất cả đối tượng Vue được khởi tạo, kể cả những component của bên thứ ba. Trong hầu hết các trường hợp, bạn chỉ nên sử mixin toàn cục cho việc xử lí tùy chọn tùy biến như ví dụ trên. Một ý tưởng hay nữa là chuyển chúng thành Plugins để tránh bị trùng lặp.*

1. **Laravel + Vuejs**
2. **Tại sao nên sử dụng Laravel + Vue.js**
3. **Mọi thứ xảy ra trên giao diện người dùng**

Các ứng dụng hiện nay được xây dựng để đảm bảo người dùng có trải nghiệm liền mạch.

Tất cả mọi thứ bây giờ xảy ra ngay trên page mà người dùng không phải tải lại trang một lần nữa.

1. **Axios**
2. **Ứng dụng**

Tạo form đăng nhập với Laravel + vue.js