# TÌM HIỂU VỀ VUE.JS

## Vue là gì

Là một framework linh động dùng để xây dựng giao diện người dùng (user interface)

## Cách cài đặt vuejs

Dùng CDN

|  |
| --- |
| <script src=”<https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js>”></script> |

## Khởi tạo vue

Một ứng dụng Vue luôn được bắt đầu bằng cách khởi tạo một đối tượng Vue (vue instance) sử dụng hàm vue

Xét đoạn văn bản html như sau

|  |
| --- |
| <div id="id1">  <p>{{title}}</p>  <p>{{greeting("world")}}</p>  </div> |
|  |
| var vm = new Vue({  el: "#id1",  data : {  title : "Chuong trinh dau tien su dung Vue js"  },  methods : {  greeting : function(text){  return "Hello "+ text;  }  }  }) |

Khởi tạo đối tương Vue ban đầu sẽ bao gồm 3 thành phần:

**el** : Khi khởi tạo một đối tượng Vue, muốn cho đối tượng Vue quản lý thành phần nào thì ta phải trỏ tới thành phần đó thông quan thuộc tính **el**.

**data**: Dùng để quan lý các thông tin của đối tượng được Vue quản lý. Trong ví dụ trên, tôi khai báo trong ***data*** một biến **“*title*”.** Khi muốn lấy giá trị của nó, ta sử dụng cú pháp **{{title}}** như ở bên văn bản html. Như vậy, tại thẻ <p> đó sẽ hiện thị ra giá trị của title mà ta khai báo trong Vue

**methods:** Các hàm hỗ trợ trong quá trình tính toán. Trong VD này, tôi khai báo một hàm có tên là greeting và có một tham số truyền vào. Tương tư data, khi muốn sử dụng methods tôi cũng sẽ có lời gọi **{{greeting(“world”)}}.**

## Data binding

Khi dùng dữ liệu muốn ràng buộc vào trong các thuộc tính của html thì ta không được dùng theo cách khai báo như trên mà phải dùng cú pháp chỉ thị. Ta xét VD sau:

|  |
| --- |
| <div id="id1">  <a v-bind:href = url >Chuyen sang trang chu google</a>  </div> |
|  |
| var vm = new Vue({  el: "#id1",  data : {  url: "Google.com"  }  }) |

Khi có chỉ thị v-bind thì chuong trình sẽ tự động liên kết với dữ liệu ta khai báo trong Vue nên không cần cặp **{{}}**

## Ràng buộc dữ liệu hai chiều

Chúngta có thể sử dụng **v-model** để tạo ràng buộc dữ liệu hai chiều lên các phần tử ***form, input, textaera***, v-model sẽ cập nhật dữ liệu dưa trên các sự kiện input từ người dùng.

|  |
| --- |
| <div id="id1">  <p>Ho ten : {{content}}</p>  <input v-model="content" type="text" placeholder="Nhap ho ten">  </div> |
|  |
| var vm = new Vue({  el: "#id1",  data : {  content: ""  }  }) |

Với cách khai báo trên, khi bạn nhập bất kỳ một ký tự nào thì nó sẽ được lưu ngay vào trong đối tượng Vue và đối tượng Vue sẽ cập nhật giá trị lập tức ra màn hình.

## Ràng buộc class

Class cũng là các thuộc tính trong html nên ta hoàn toàn có thể ràng buộc chúng bằng **v-bind**. Nhưng để linh động hơn thì chúng ta có thể sử dụng cấu trúc ràng buộc class do Vue cung cấp

|  |
| --- |
| <div id="id1">  <p class="change" v-bind:class = "objClass"></p>  </div> |
|  |
| var vm = new Vue({  el: "#id1",  data : {  isColor : true,  isFontSize : false  },  computed: {  objClass : function(){  return {  color : this.isColor,  fontSize : this.isFontSize  }  }  }  }) |

**computed** : sử dụng tương tự với methods nhưng nó sẽ sử dụng trục tiếp dữ liệu mà vue có nên khi gọi hàm thì không cần gọi tường minh như **methods**

Trong ví dụ trên, tên class sẽ được thay đổi phụ thuộc vào các thuộc tính mà vue giữ trong phần data và các thuộc tính này có thể thay đổi.

## Render template theo điều kiện if-else

Dùng **v-if** để render lại template

|  |
| --- |
| <div id="id1">  <button v-on:click = "change('cat')">Cat</button>  <button v-on:click = "change('input')">Input</button>  <img v-if ="kind === 'cat'" src="https://i.ytimg.com/vi/1.jpg">  <input v-else = "kind === 'input'" type="" name=""/>  </div> |
|  |
| var vm = new Vue({  el : '#id1',  data: {  kind :''  },  methods : {  change(text){  return this.kind = text;  }  }  }) |

Chúng ta đang điều khiển cấu trúc DOM bằng Vue.

Ở file html, ta tạo ra hai nút với mục đích hiện ra ảnh hoặc hiện ra ô input.

Ở mỗi một button sẽ có sự kiện click được Vue quản lý, và mỗi lần click vào thì biến kind sẽ được thay đổi.

Vue sẽ kiểm tra giá trị của kind để quyết định xem sẽ hiện phần tử nào, nếu kind có giá trị là ‘cat’ thì DOM sẽ hiển thị thẻ img bao gồm ảnh. Ngược lại, DOM sẽ hiển thị ra một ô input.

**Lưu ý:** *v-else phải xuất hiện ngay sau v-if.*

## Render template sử dụng vòng lặp for

|  |
| --- |
| <div id="id1">  <div v-for = "item in books">{{item}}</div>  </div> |
|  |
| var vm = new Vue({  el : '#id1',  data: {  books : ["Lão Hạc","Chí Phèo", "Số dỏ"]  },  methods : {  change(text){  return this.kind = text;  }  }  }) |

Ta dùng cú pháp **v-for** để hiện thị ra tất cả các thành phần trong danh sách cho trước mà dữ liệu của nó được lưu trên data.

## Component

Giúp kế thừa các thành phần HTML cơ bản

Dễ dàng đóng gói và tái sử dụng code

Là custom element mà trinh biên dịch của vue sẽ gắn các xử lý vào nó

## Passing dữ liệu giữa các component

Việc trao đổi dữ liệu giữa các component sẽ giúp chúng ta dễ dàng kiểm soát dữ liệu và tạo một luồn trao đổi dữ liệu an toàn trong ứng dụng

**Truyền dữ liệu từ cha xuống con thông qua props**

***Props*** là thuộc tính do chính chúng ta tạo ra . Sau đó, trong <template> chúng ta sẽ cung cấp giá trị thông qua thuộc tính tùy chỉnh đó

|  |
| --- |
| <template>  <div id=”app”>  <com-header v-bind:contentTitle=”title”></com-header>  </div>  </template>  <script>  import ComHeader from ‘./component/ComponentHeader’  export default {  name: ‘App’,  components: {ComHeader},  data () {  return {  title: ‘huyen’  }  }  }  </script> |

|  |
| --- |
| <template>  <header><h1>{{contentTitle}}</h1></header>  </template>  <script>  export default {  name: ‘com-header’,  props: {  contentTitle: String  }  }  </script> |

Attribute **contentTitle** do người dùng tự định nghĩa và giá trị của nó được vue quản lý thông qua data. Attribute **contentTiltle** đó là ở component cha. Vậy khi muốn sử dụng data của attribute đó ở component con thì ta phải sử dụng **props**. Trong props chỉ cần khai báo kiểu dữ liệu mà nó nhận về là đủ.

**Lưu ý:** *Props chỉ truyền dữ liệu một chiều từ component cha xuống component con.*

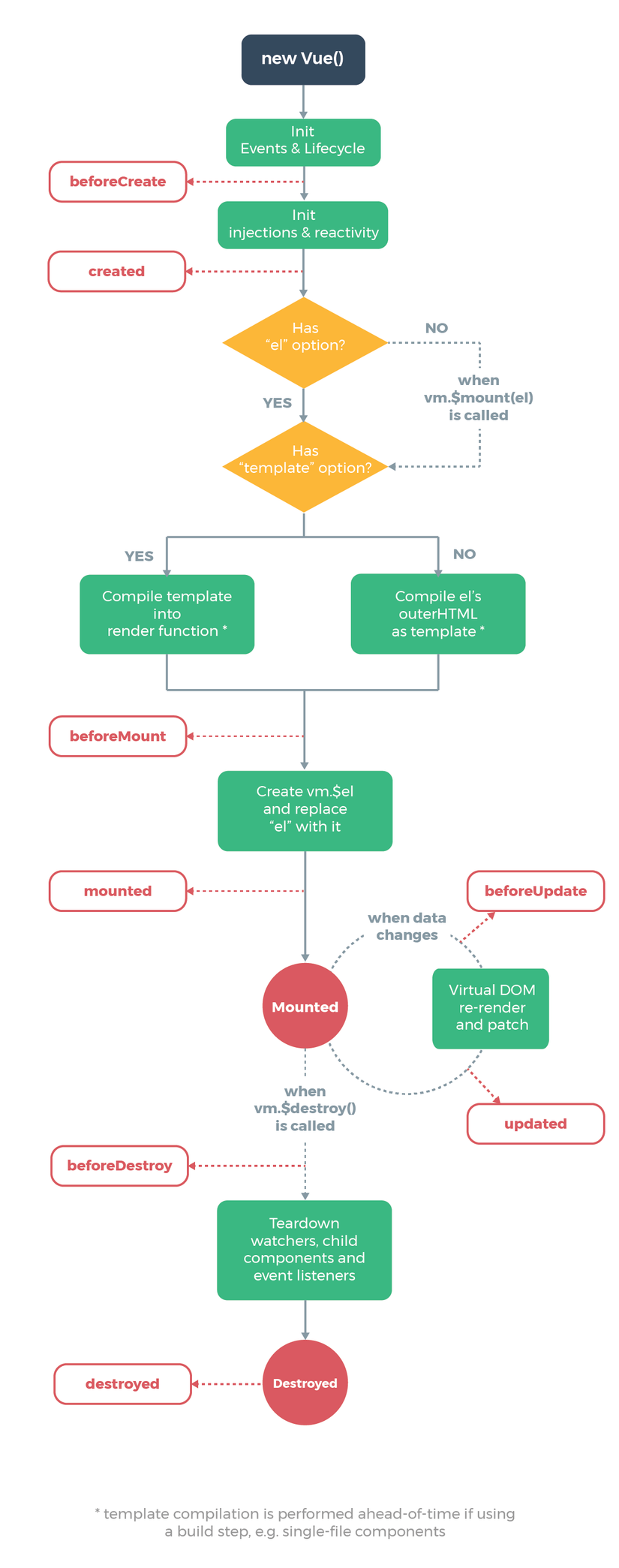
**Truyền dữ liệu từ con lên cha bằng event up**

Truyền thông báo từ component con tới component cha rằng nó muốn thay đổi dữ liệu

|  |
| --- |
| <template>  <div id=”app”>  <com-header v-bind:contentTitle=”title”  v-on:changeTitle= ”handleChange”></com-header>  </div>  </template>  <script>  import ComHeader from ‘./component/ComponentHeader’  export default {  name: ‘App’,  components: {ComHeader},  data () {  return {  Title: ‘huyen’  }  },  methods: {  handleChange () {  this.title = ‘xinh gai’  }  }  }  </script> |

Component cha nhận được thông báo thì sẽ thay đổi

|  |
| --- |
| <template>  <header><h1>{{contentTitle}}<br>  <button v-on:click= “changeTitle”></button>  </h1></header>  </template>  <script>  export default {  name: ‘com-header’,  props: {  contentTitle: String  },  methods: {  changeTitle() {  this.$emit(‘changeTitle’)  }  }  }  </script> |

Trong component cha, ta khai báo một sự kiện do ta tự định nghĩa là **changeTitle** và hàm truyền vào là hàm **handleChange** được vue quản lý trong methods. Tròn component con, ta cũng có một sự kiện click, và hàm truyền vào là **changeTitle**. Trong hàm này, ta dùng this.$emit để thông báo cho component cha thay đổi title. Tham số truyền vào this.$emit chính là sự kiện mà ta tự định nghĩa ở component cha.

## Lifecycle diagram vuejs

Vòng đời của đối tượng vue gồm 4 giai đoạn

* ***Khởi tạo một đối tượng Vue***

+ Trước khi hệ thống phản ứng xuất hiện thì hàm beforeCreate được gọi tới. Lúc này, ta không thể truy xuất bất cứ thành phần vào của đối tượng vue.

+ Tại hàm created thì các phần data và event được thiết lập. Vue có thể được truy xuất.

* ***Mounting (chèn phần tử DOM)***

Mounting cho phép truy xuất vào các component ngay lập tức trước và sau khi nó hiển thị lần đầu tiên.

+ beforeMount: nó sẽ thực hiện trước khi render chính thức phần tử DOM của lớp Vue. Tại đây thì ta chưa truy xuất được phần tử $el

+ mounted: Tại đây ta có quyền truy xuất tới các component , template, DOM thông qua this.$el

* ***Updating*** : Sẽ được gọi khi có sự thay đổi trong các component

+ beforeUpdate: được gọi tới ngay sau khi có sự thay đổi từ component và thực hiện trước khi DOM được render

+ Updated: Thực hiện ngay sau khi dữ liệu thay đổi trên component. Dữ liệu được truy xuất sẽ là dữ liệu mới nhất và được láy từ trên hàm beforeUpdate

* ***Destruction***: dùng để xóa bỏ DOM

+ beforeDestroy: Được gọi trước khi hủy bỏ component

+ destroyed: được gọi khi component đã bị xóa khỏi DOM.

## Lấy dữ liệu từ Vue bằng axios

Ta sử dụng axios để lấy và hiện thị dữ liệu từ api như sau:

|  |
| --- |
| <div id=”app”>{{infor}}</div> |

|  |
| --- |
| var vm = new Vue({  el: "#app",  data : {  infor: null  },  mounted () {  axios  .get('https://api.coindesk.com/v1/bpi/currentprice.json')  .then(response => (this.infor = response.data.bpi))  }) |