Cu Van Blue

Cuvankhoa.dev@gmail.com

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Vuejs documentation

Tổng hợp kiến thức tham khảo về Vuejs

Mục lục

[I. Mở đầu 1](#_Toc144996832)

[1. Giới thiệu 1](#_Toc144996833)

[2. Bắt đầu 1](#_Toc144996834)

[II. Cơ bản 1](#_Toc144996835)

[1. Create app 1](#_Toc144996836)

[2. Template syntax 2](#_Toc144996837)

[3. Reactivitiy fundamentals 4](#_Toc144996838)

[4. Computed props 4](#_Toc144996839)

[5. Class & style bindings 4](#_Toc144996840)

[6. Conditional redering 4](#_Toc144996841)

[7. List rendering 4](#_Toc144996842)

[8. Event 4](#_Toc144996843)

[9. Form input binding 4](#_Toc144996844)

[10. Life cycle hooks 4](#_Toc144996845)

[11. Watcher 4](#_Toc144996846)

[12. Template ref 4](#_Toc144996847)

[13. Component basics 4](#_Toc144996848)

[III. Component nâng cao 4](#_Toc144996849)

[IV. Tái sử dụng ( reuse) 4](#_Toc144996850)

[V. Component có sẵn ( built-in) 4](#_Toc144996851)

[VI. Mở rộng ( Scale up) 4](#_Toc144996852)

[VII. API 4](#_Toc144996853)

1. Mở đầu
2. Giới thiệu

* Vuejs là framework Js xây dựng giao diện người dùng
* Sử dụng html, css, js tiêu chuẩn
* Dựa trên mô hình code theo thành phần ( component-based) và khai báo
* 2 tính chất cốt lõi:
* Render giao diện dựa trên dữ liệu truyền vào
* React Render: tự động cập nhật giao diện khi dữ liệu thay đổi
* Điểm mạnh:
* File component đơn
* API style:
* Option api: khai báo script theo một khung có sẵn gồm: data ( dữ liệu dùng trong component), method ( các hàm xử lý), lifecycle hook
* Composition api: khai báo state với ref, các hàm xử lý, các lifecycle hook tự do không cần theo thứ tự

1. Bắt đầu

* Tạo dự án: > npm create vue@latest
* Cài đặt các gói cần thiết: >npm install
* Host local: >npm run dev
* Build product: >npm run build

1. Cơ bản
2. Create app

* Tạo một app với createApp()
* Truyền vào một component với vai trò là root (cha lớn nhất)
* Mount app vừa tạo vào div có id là ‘app’ trong file index.html => nội dung của component App sẽ hiển thị trong div có id là app
* Mount app luôn được gọi sau cùng

|  |
| --- |
| import { createApp } from ‘vue’  import App form ‘./App.vue’  const app = createApp({  /\* root component opts \*/  App  });  app.mount(‘#app’); |

* Cấu hình ứng dụng với app.config để xử lý tất cả lỗi của con cháu:

|  |
| --- |
| app.config.errorHandler = (err) => {  /\* handle error \*/  } |

* Đăng ký component có phạm vi mọi nơi trong ứng dụng

|  |
| --- |
| app.component('TodoDeleteButton', TodoDeleteButton) |

* Cho phép đăng ký nhiều phiên bản ứng dụng:

|  |
| --- |
| const app1 = createApp({  /\* ... \*/  })  app1.mount('#container-1')  const app2 = createApp({  /\* ... \*/  })  app2.mount('#container-2') |

1. Template syntax

Vue sử dụng cú pháp dựa trên html sau đó biên dịch thành js kết hợp với hệ thống render động ( react) có thể tìm ra số lượng thành phần tối thiểu, thao tác DOM tối thiểu để render lại

Có thể viết các hàm render như react nhưng không được hưởng tối ưu thời gian biên dịch

* 1. Nội suy văn bản ( chèn dữ liệu)
* Chèn dữ liệu vào html bằng dấu mustaches (ngoặc nhọn kép): {{ x }}
  1. Html thô
* Dấu mustaches chỉ chèn được plain text, nếu muốn chèn html phải sử dụng: v-html, nội dung thẻ span bên dưới sẽ bị thay bởi rawHtml:

|  |
| --- |
| <p>Using text interpolation: {{ rawHtml }}</p>  <p>Using v-html directive: <span v-html="rawHtml"></span></p> |

* 1. Thuộc tính động ( Attribute bindings)
* Dấu mustache cũng không thể dùng trong attribute html, thay vào đó dùng v-bind:tenthuoctinh hoặc viết tắt :tenthuoctinh :
* Nếu giá trị bind vào là null hoặc undefinded, thuộc tính sẽ bị bỏ qua.
* Với các thuộc tính chỉ chấp nhận giá trị true false, nếu truyền vào chuỗi rỗng, thuộc tính nhận giá trị chuỗi rỗng, nếu truyền các giá trị khác rỗng hoặc true, false, thuộc tính sẽ bị bỏ qua

|  |
| --- |
| <div v-bind:id="dynamicId"></div>  /\* short hand \*/  <div :id="dynamicId"></div> |

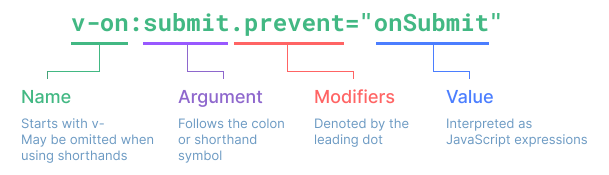
* Bind nhiều thuộc tính :

|  |
| --- |
| objectOfAttrs: {  id: 'container',  class: 'wrapper'  } |
| <div v-bind="objectOfAttrs"></div> |

* Các giá trị được biểu thị bởi phép nội suy hay binding sẽ được đồng bộ với biến truyền vào và được cập nhật mỗi khi biến đó thay đổi
  1. Biểu thức Javascript
* Trong Vue template, javascript có thể được viết tại:
* Bên trong dấu mustaches : {{ }}
* Bên trong thuộc tính v-bind: :id= “currentId++”
* Mỗi binding hoặc mustache chỉ chứa một biểu thức (toán tử 3 ngôi được tính là 1 biểu thức)

|  |
| --- |
| *<!-- this is a statement, not an expression: -->*  {{ var a = 1 }}  *<!-- flow control won't work either, use ternary expressions -->*  {{ if (ok) { return message } }} |

* Biểu thức trong template không thể truy cập các thuộc tính global như window, tuy nhiên có thể khai báo nó trong app.config.globalProperties
* Một số thư viện có thể dùng khi viết js trong template, nếu muốn dùng thêm thì phải khai báo: https://github.com/vuejs/core/blob/main/packages/shared/src/globalsAllowList.ts#L3
  1. Chỉ thị điều phối ( Directives)



Định nghĩa:

* Là một thuộc tính đặc biệt với tiền tố v- (v-bind, v-html,…)
* Nhận giá trị là một biểu thức javascript đơn ( single js expression) ngoại lệ: v-for, v-on, v-slot
* Nhiệm vụ: cập nhật cây DOM khi giá trị biểu thức nó nắm giữ thay đổi
* Một số chỉ thị có thể nhận đối số: v-bin:href hay :href dùng để update thuộc tính của thẻ html. Ví dụ với v-on dùng để lắng nghe sự kiện
* Truyền động thuộc tính : tên thuộc tính để trong ngoặc vuông, chấp nhận string. Nếu thuộc tính truyền vào bằng null || không hợp lệ => chỉ thị bị bỏ qua.

|  |
| --- |
| <a :[someAttr]="value"> ... </a> |

Modifiers:

* Là các hậu tố đặc biệt đằng sau dấu chấm, cho biết chỉ thị phải hoạt động theo một cách nào đó. Ở dưới đây, event.preventDefault() được gọi khi sự kiện submit xảy ra

|  |
| --- |
| <form @submit.prevent="onSubmit">...</form> |

1. Reactivitiy fundamentals
   1. Khai báo reactive state

* Dùng ref() để khai báo một reactive state

|  |
| --- |
| import { ref } from 'vue'  const count = ref(0)  const count = 0  // tạo một ref obj nhận vào giá trị ban đầu là 0 và thao tác với giá // trị đó thông qua count.value .Ex: count.value++;  // hoặc tạo một hàm increment ngay cạnh state:  function increment() {  // .value is needed in JavaScript  count.value++  } |

* Ref object là một mutable obj (obj khả biến) => có thể gán giá trị mới cho .value
* Ref object là một reactivity api => .value được theo dõi, khi thay đổi giá trị, nó trigger các hiệu ứng liên quan ( render lại)
* Vì sao dùng ref và ref.value thay vì biến đơn giản (plain variable)?
* Vue chỉ cần theo dõi sự thay đổi của biến thông qua set và get qua value. Trong khi dùng biến thuần thì không có cách nào để theo dõi sự thay đổi đột ngột của giá trị => ta cần một phương thức set và get duy nhất, khi đó mọi thay đổi của state bắt buộc phải đi qua .value
* Có thể truyền ref vào hàm trong khi vẫn giữ khả năng truy cập đến giá trị mới nhất, khả năng render lại và tái sử dụng code
* Deep Reactivity:
* Ref có thể chứa bất kì giá trị nào (kể cả obj, mảng đa chiều, cấu trúc dữ liệu built-in js (Map))
* Nếu gán cho ref object một giá trị object, và object này tham chiếu đến nhiều biến khác, thì tất cả các biến bên trong sẽ được lắng nghe và tính là một sự thay đổi nếu chúng thay đổi giá trị ( deeply unwrap)
* Những kiểu dữ liệu không nguyên thủy bị chuyển thành reactive proxies qua reactive(), cái mà sẽ được thảo luận bên dưới
* Để tránh trường hợp này, ta dùng shallowRef(). Khi đó chỉ những .value có quyền truy cập mới được theo dõi => tránh lãng phí tài nguyên để theo dõi lượng lớn biến tham chiếu // ref “nông” ??
* Thời gian cập nhật DOM:
* Khi thay đổi reactive state, DOM tự động cập nhật
* Tuy nhiên, DOM không cập nhật ngay lập tức mà đợi đến “next tick” trong chu trình cập nhật để đảm bảo mỗi thành phần chỉ cập nhật một lần dù state thay đổi bao nhiêu lần
* Để đợi DOM cập nhật hoàn tất, dùng nextTick()

|  |
| --- |
| import { nextTick } from 'vue'  async function increment() {  count.value++  await nextTick()  // Now the DOM is updated  } |

* 1. reactive()
* Là cách khác để dùng reactive state
* Không bao bọc giá trị state trong obj đặc biệt, reactive() chuyển bản thân obj thành reactive

|  |
| --- |
| import { reactive } from 'vue'  const state = reactive({ count: 0 }) |

* (1) Reactive object là [Javascript Proxy](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy) object và hoạt động như một object bình thường (object có thể chỉnh sửa cách thức nó đọc và ghi dữ liệu, định dạng đầu vào,…). Khác biệt là: vue có quyền chặn truy cập, thay đổi các thuộc tính của đối tượng để theo dõi và kích hoạt phản ứng
* (2) Giống ref, nếu truyền vào một object hay cấu trúc dữ liệu lồng nhau, reactive sẽ kiểm tra sự thay đổi của những phần tử cấp thấp nhất. Nếu ref được truyền vào một obj, reactive cũng sẽ được gọi bên trong.
* Tương tự ref, reactive kiểm soát theo dõi sâu shallowReactive()
* (3) Reactive là một proxy obj, thế nên nó chỉ phát hiện thay đổi khi proxy thay đổi.
* Từ (1), (2), (3) => Thuộc tính , hoặc đối tượng lồng trong proxy obj cũng là một proxy object

|  |
| --- |
| const raw = {}  const proxy = reactive(raw)  // proxy is NOT equal to the original.  // raw thay đổi không trigger đến reactive  console.log(proxy === raw) // false  // calling reactive() on the same object returns the same proxy  console.log(reactive(raw) === proxy) // true  // calling reactive() on a proxy returns itself  console.log(reactive(proxy) === proxy) // true  const proxy = reactive({})  const raw = {}  proxy.nested = raw  console.log(proxy.nested === raw) // false |

* Hạn chế của reactive():
* Reactive chỉ nhận vào kiểu object (obj, array, collection, map, set), không nhận vào kiểu nguyên thủy ( number, string, boolean)
* Không thể gán lại đối tượng reactive cho state mà chỉ có thể thay đổi state
* Khi destructure một thuộc tính nguyên thủy của reactive obj sang biến local hoặc truyền thuộc tính đó làm tham số của hàm, chúng ta mất khả năng theo dõi thuộc tính đó. => dùng ref() để khai báo reactive state là sự lựa chọn ưu tiên
  1. Chi tiết bổ sung về Ref Unwrapping
     1. Nếu ref là thuộc tính của Reactive Object
* Nếu đối tượng ref trong ngữ cảnh là một thuộc tính của reactive object, nó sẽ tự động mở (unwrap) và hoạt động như một thuộc tính bình thường

|  |
| --- |
| const count = ref(0)  // count là 1 proxy obj  // count.value = 0  const state = reactive({  count  }) // gán ref obj là một thuộc tính của reactive obj  // không cần state.count.value  console.log(state.count) // 0  state.count = 1  console.log(count.value) // 1 |

* Nếu gán một ref khác cho reactive obj, ref ban đầu bị thay thế và ngắt kết nối với reactive obj – nhưng vẫn hoạt động độc lập:

|  |
| --- |
| const otherCount = ref(2)  state.count = otherCount  console.log(state.count) // 2  // original ref is now disconnected from state.count  console.log(count.value) // 1 |

* Ref unwrapping chỉ hoạt động khi được “làm tổ” trong deep reactive obj. Nó sẽ không hoạt động trong shallow reactive obj
  + 1. Nếu ref là phần tử của Array hoặc Collection
* Nếu ref object là phần tử của một mảng reactive hoặc bộ sưu tập reactive, unwrap không được áp dụng. Thay vào đó cần dùng .value để thao tác với giá trị của ref
  + 1. Ref và Template
* Trong thẻ template, ref unwrapping chỉ áp dụng cho thuộc tính top-level:

|  |
| --- |
| const count = ref(0)  const object = { id: ref(1) }  // count và object là top level trong khi object.id thì không  {{ count + 1 }} // đúng  {{ object.id + 1 }} // sai  {{ object.id.value + 1 }} //đúng  // chuyển object.id về top-level bằng destructure:  Const { id } = object  {{ id }} // correct!! |

1. Computed props
   1. Basic Example

* Biểu thức in-template tiện dụng nhưng chỉ dành cho các logic đơn giản. Đưa quá nhiều logic phức tạp vào template sẽ khiến chúng cồng kềnh khó bảo trì

|  |
| --- |
| // có một reactive state như sau:  const author = reactive({  name: 'John Doe',  books: [  'Vue 2 - Advanced Guide',  'Vue 3 - Basic Guide',  'Vue 4 - The Mystery'  ]  })  // trong template muốn hiển thị xem tác giả đã publish sách nào //hay chưa?  <p>Has published books:</p>  <span>{{ author.books.length > 0 ? 'Yes' : 'No' }}</span> |

* Giả sử trong template có rất nhiều nơi cần hiển thị tương tự thì code sẽ trở nên rất dài và cồng kềnh => với những logic phức tạp bao gồm reactive state, chúng ta sử dụng Computed Properties. Ex:

|  |
| --- |
| <script setup>  import { reactive, computed } from 'vue'  const author = reactive({  name: 'John Doe',  books: [  'Vue 2 - Advanced Guide',  'Vue 3 - Basic Guide',  'Vue 4 - The Mystery'  ]  })  // a computed ref  const publishedBooksMessage = computed(() => {  return author.books.length > 0 ? 'Yes' : 'No'  })  </script>  <template>  <p>Has published books:</p>  <span>{{ publishedBooksMessage }}</span>  </template> |

* Giải thích:
* Hàm computed trả về một computed ref có thể được truy cập thông qua publishedBooksMessage.value
* Trong template, do là top-level nên publishedBookMessage trở nên unwrap và có thể lấy được giá trị mà không cần dùng .value
* Hàm Computed sẽ tự động theo dõi các reactive dependencies của nó ( trong TH này là author.book) và tự động cập nhật giá trị trả về nếu author.book thay đổi
* dfsf
  1. Computed Catching vs Methods

|  |
| --- |
| <p>{{ calculateBooksMessage() }}</p>  // in component  function calculateBooksMessage() {  return author.books.length > 0 ? 'Yes' : 'No'  } |

* Ở ví dụ phía trên, thay vì dùng computed ta sử dụng hàm bình thường để trả về kết quả cho nội dung thẻ <p>
* Vậy khác biệt ở đây là gì?
* Trong ví dụ trên, mỗi khi cần truy cập đến giá trị hoặc rerender, hàm calculateBooksMessage sẽ phải chạy lại
* Computed properties sẽ lưu giá trị tính toán trong bộ nhớ đệm ( cache). Mỗi khi cần dùng, nó sẽ trả về giá trị ngay trước đó ( lưu trong cache) mà không cần tính toán chừng nào reactive dependencies bên trong thay đổi . Giả sử computed properties không có react dependencies, giá trị trả về của nó sẽ không bao giờ thay đổi kể cả khi các giá trị bên trong của nó thay đổi:

|  |
| --- |
| const now = computed(() => Date.now()) |

* Tại sao cần bộ nhớ đệm? Giả sử reactive state của bạn là một list đòi hỏi khối lượng tính toán rất lớn mỗi khi lặp và bạn có những component phụ thuộc vào từng phần tử trong đó. Nếu không dùng cache, mỗi khi list thay đổi bạn sẽ phải thực hiện rất nhiều thao tác get dữ liệu không cần thiết.
* Nếu không muốn cache như ví dụ với Date.now(), bạn nên dùng method thông thường.
  1. Writable Computed
* Mặc định computed là getter-only ( chỉ đọc). Nếu cố ghi giá trị cho thuộc tính computed, sẽ báo lỗi runtime warning
* Trong những trường hợp hiếm hoi muốn ghi đè computed, bạn nên thêm 2 phương thức get và set. Ex:

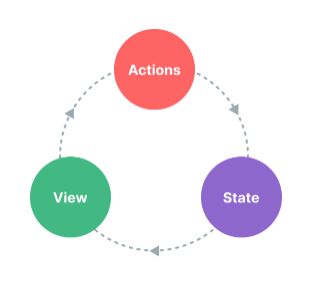
|  |
| --- |
| <script setup>  import { ref, computed } from 'vue'  const firstName = ref('John')  const lastName = ref('Doe')  const fullName = computed({  // getter  get() {  return firstName.value + ' ' + lastName.value  },  // setter  set(newValue) {  // Note: we are using destructuring assignment syntax here.  [firstName.value, lastName.value] = newValue.split(' ')  }  })  </script> |

* 1. Best practice
* Chỉ nên dùng computed để tính toán thuần túy mà không có tác dụng phụ ( side effect)
* Không viết bất đồng bộ trong computed
* Không viết DOM manipulate trong computed
* Tốt nhất các giá trị của computed nên để mặc định, tránh ghi đè ( read-only)

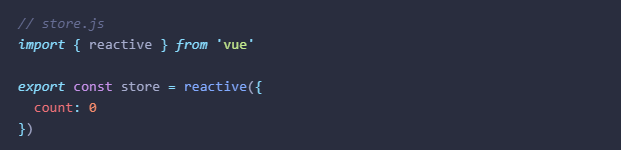
1. Class & style bindings
2. Conditional redering
3. List rendering
4. Event
5. Form input binding
6. Life cycle hooks
7. Watcher
8. Template ref
9. Component basics
10. Component nâng cao
11. Tái sử dụng ( reuse)
12. Component có sẵn ( built-in)
13. Mở rộng ( Scale up)
14. Single-File Components
15. Tooling
16. Routing
17. State Management
    1. Vấn đề

* Thông thường, mỗi component tự quản lý state của nó

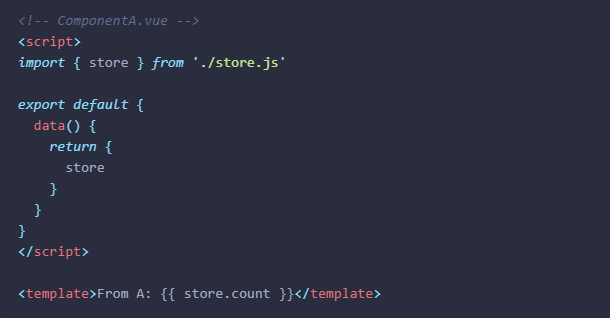
|  |
| --- |
| <script>  export default {  // state  data() {  return {  count: 0  }  },  // actions  methods: {  increment() {  this.count++  }  }  }  </script>  <!-- view -->  <template>{{ count }}</template> |



* Luồng xử lý: Views trigger vào các method của actions => Actions cập nhật states => States thay đổi kéo theo view render lại
* Vấn đề xảy ra: Nếu nhiều Views cần dùng dữ liệu từ một state, hoặc khi tác động vào một View để thay đổi state ta muốn các View khác phụ thuộc vào state này cũng phải thay đổi.=> Ta có thể phải truyền ‘sự thay đổi’ đó qua prop rất nhiều tầng lên lớp cha chung gần nhất rồi lại truyền xuống lớp con đích để component con có thể sử dụng (Prop Drilling)
* Giải pháp: tách riêng state để quản lý trong toàn cục, khi đó mọi component có quyền truy cập state hoặc trigger action
  1. Quản lý state với Reactivity API
* Tạo file store.js
* Import reactive
* Tạo obj store bằng hàm reactive và truyền vào một hoặc nhiều state dùng chung

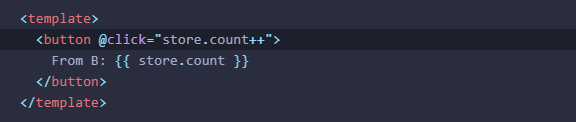


* Trong component, import file store và khai báo biến store trong data(). Truy cập state bằng store.tenState:

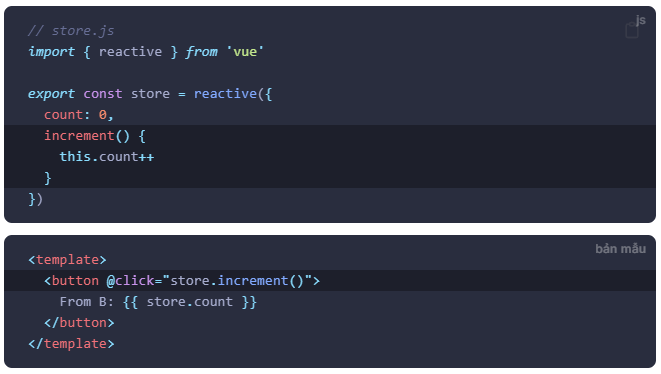




* Thay đổi state từ bất cứ component nào có import store:



* Hoặc tạo ra một hàm trong store chỉ dùng để thay đổi một state nhất định ( tương tự react hook setState ):



* Từ giờ, mỗi khi state thay đổi, cả component A và B sẽ tự động render lại theo state mới.
* Có thể sử dụng các Reactivity API khác như ref, computed
  1. SSR Considerations
  2. Pinia & Vuex
* Để quản lý state quy mô lớn, ta sử dụng thư viện hỗ trợ quản lý trạng thái có sẵn: pinia hoặc vuex
* Vuex - thư viện quản lý trạng thái chính thức trước đây của Vue – giờ đang ở trạng thái bảo trì và không được cập nhật thêm tính năng mới
* Pinia – phiên bản mới hơn của Vuex, cung cấp API đơn giản hơn, cung cấp các API kiểu Composition, suy luận kiểu chắc chắn và được hỗ trợ khi dùng với TypeScript

1. Testing
2. Server-Side Rendering (SSR)
3. API
4. Global API
5. Composition API
   1. Setup()
   2. Reactivity: Core

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

* 1. Reactivity: Utilities
  2. Reactivity: Advanced
  3. Lifecycle Hooks
  4. Dependency Injection

1. Options API
2. Built-ins
   1. Directives

|  |  |  |  |
| --- | --- | --- | --- |
| Chỉ thị | Đầu vào | Chi tiết | Ví dụ |
| v-text | String | * Thay thế, ghi đè nội dung trong thẻ html bằng text truyền vào. * Nếu cần update 1 phần nội dung thì dùng mustache | <span v-text="msg"></span> |
| v-html | String | * Nội dung của v-html chèn vào như html thuần * Vue template syntax không thực thi | <div v-html="html"></div> |
| v-show | Any | * Setting display inline css * Trigger transition khi thay đổi * Ứng dụng làm render có điều kiện | <h1 v-show="ok">Hello!</h1> |
| v-if | Any | * Khi điều kiện trong v-if thay đổi, thành phần và chỉ thị chứa nó sẽ bị phá hủy và xây dựng lại * Nếu điều kiện là sai thì không hiển thị * Trigger transition * V-if có độ ưu tiền > v-for * Không nên sử dụng v-if và v-for trên cùng 1 component | <h1 v-if="awesome">Vue is awesome!</h1> |
| v-else, v-else-if | Any | * Theo sau v-if * Có thể lặp lại nhiều lần | <button @click="awesome = !awesome">Toggle</button>  <h1 v-if="awesome">Vue is awesome!</h1>  <h1 v-else>Oh no 😢</h1> |
| v-for | Array | Object | number | string | Iterable | * Render lại thành phần chứa nó nhiều lần dựa trên data source * Thành phần render tiếp theo gắn vào sau thành phần trước đó * Để render theo thứ tự, cần thêm thuộc tính :key * V-for | <div v-for="(item, index) in items"></div>  <div v-for="(value, key) in object"></div>  <div v-for="(value, name, index) in object"></div>  <div v-for="item in items" :key="item.id">  {{ item.text }}  </div> |
| v-on ( viết tắt là @) | Function | Inline Statement | Object (without argument) | * Gắn event listener lên element * Trên thành phần DOM gốc, nó listen các sự kiện DOM gốc, trên thành phần tự định nghĩa, nó lắng nghe các sự kiện tự định nghĩa * Nhận tham số là event * Có các modifier là:  |  |  | | --- | --- | | .stop |  | | .prevent |  | | .capture |  | | .self |  | | .{keyAlias} |  | | .once |  | | .left |  | | .right |  | | .middle |  | | .passive |  | | <button v-on:[event]=" doThat('hello', $event)"></button> |
| v-bind ( viết tắt là : ) | Any || arr hoặc obj nếu thuộc tính là class hoặc style | * Liên kết động thuộc tính, prop với một biểu thức * Khi dùng với thuộc tính class hoặc style v-bind chấp nhận kiểu obj hoặc arr * Modifier:  |  |  | | --- | --- | | .camel | Chuyển sang camel format (ex: camelFormat) | | .prop | Bắt buộc binding trở thành thuộc tính | | .attr |  | | <button v-bind:[key]="value"></button>  <div :class="[classA, { classB: isB, classC: isC }]"></div>  <MyComponent :prop="someThing" /> |
| v-model |  |  |  |
| v-slot |  |  |  |
| v-pre |  |  |  |
| v-once |  |  |  |
| v-memo |  |  |  |
| v-cloak |  |  |  |

* 1. Component
  2. Special Element
  3. Special Attribute

1. Single-File Component
2. Advanced APIs
3. Vuex
4. Giới thiệu
   1. Vuex là gì
      1. State Management Pattern

* Lấy cảm hứng từ Redux, Flux,…
* Thiết kế riêng cho Vue để tận dụng hệ thống phản ứng chi tiết qua đó cập nhật state hiệu quả



* + 1. Khi nào sử dụng Vuex?
* Khi muốn quản lý state của SPA quy mô trung bình lớn
* Khi muốn xử lý state tốt hơn từ bên ngoài component
  1. Cài đặt
  2. Bắt đầu sử dụng
* Trung tâm của Vuex là store – một nơi chứa state, vì thế nó khác một global object đơn giản khác là:
* Vuex store là reactive. Nó sẽ tự động theo dõi và cập nhật state khi các thành phần thay đổi
* Không thể trực tiếp thay đổi state của store. Cách duy nhất để thay đổi chúng là thực hiện mutation. Điều này đảm bảo mỗi khi thay đổi, mutation để lại một bản ghi có thể theo dõi
* Store đơn giản:

|  |
| --- |
| import { createApp } from 'vue'  import { createStore } from 'vuex'  // Create a new store instance.  const store = createStore({  state () {  return {  count: 0  }  },  mutations: {  increment (state) {  state.count++  }  }  })  const app = createApp({ /\* your root component \*/ })  // Install the store instance as a plugin  app.use(store) |

1. Khái niệm cốt lõi
2. Nâng cao
3. Thay đổi từ 3.x sang 4.0
4. Pinia