Version develop:

src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js">

# sử dụng SASS / SCSS với dự án khởi động vue-cli mới nhất?

chạy 2 dòng lệnh

npm install -D sass-loader sass

-// use sever yarn

yarn add --dev sass-loader sass

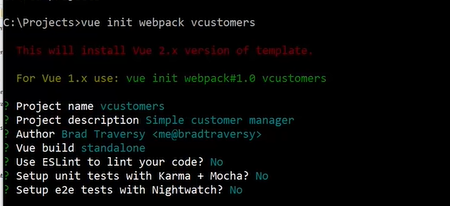
# sử dụng Bounce

tài liệ tham khảo:

<https://www.npmjs.com/package/vue-debounce>

Đợi sau 200 ms s nếu không có lời gọi hàm nào thì thực thi. Nếu có thì xóa lời gọi cũ lời gọi mới tiếp tục đợi 200ms s

# create project



# **Định nghĩa** (cung cấp tùy chọn cho component)

* Ta có thể thêm class cho thẻ con đầu tiên của <emplate> trong như sau

<Footer msg="Welcome to Your Vue.js App" class="h-container" />

## Is

## Text

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

{{ avariable }}

## Attributes

### [cú pháp object](https://vi.vuejs.org/v2/guide/class-and-style.html#Su-dung-cu-phap-object)

-Để có thể thêm các attribute vào tag HTML bằng dữ liệu trong vue.js thì bạn 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**.

<div id="app">

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

</div>

<script type="text/javascript"></script>

<script type="text/javascript">

var app = new Vue({

el: '#app',

data: {

slogan: 'Chào mừng bạn đã đến với website: <font color="orange">toidicode.com</font>',

className: 'text-red',

},

});

</script>

**VD**: Bind CSS style vào trong tag **h1.**

<div id="app">

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

</div>

<script type="text/javascript"></script>

<script type="text/javascript">

var app = new Vue({

el: '#app',

data: {

slogan: 'Chào mừng bạn đã đến với website: <font color="orange">toidicode.com</font>',

className: 'text-red',

styleData: 'text-align: center',

},

});

#### bật tắt thuộc tính

vd:

 <div v-bind:class="{success: attrColor}">text color</div>

cú pháp :

v-bind:<attribute name>={<attribute content>: <status(true or false)>}

viết tắt :<attribute name>={<attribute content>: <status(true or false)>}

[**cú pháp mảng**](https://vi.vuejs.org/v2/guide/class-and-style.html#Su-dung-cu-phap-mang)

Chúng ta có thể truyền một mảng vào v-bind:class để áp dụng một danh sách class:

|  |
| --- |
| <div v-bind:class="[activeClass, errorClass]"></div> |

|  |
| --- |
| data: {  activeClass: 'active',  errorClass: 'text-danger' } |

sẽ render ra kết quả sau:

|  |
| --- |
| <div class="active text-danger"></div> |

Nếu muốn bật tắt theo điều kiện một class trong danh sách, bạn có thể dùng một toán tử ba ngôi (ternary expression):

|  |
| --- |
| <div v-bind:class="[isActive ? activeClass : '', errorClass]"></div> |

Đoạn code trên sẽ luôn luôn áp dụng class errorClass, nhưng chỉ áp dụng activeClass khi isActive mang giá trị đúng.

Cách làm này có thể hơi dài dòng nếu bạn có nhiều class theo điều kiện. Do đó, bạn có thể dùng cú pháp object bên trong cú pháp mảng, như sau:

|  |
| --- |
| <div v-bind:class="[{ active: isActive }, errorClass]"></div> |

# **Render danh sách** (tạo nhiều dạng khác nhau của cùng một phần tử)

## v-for

vd:  <input type="text" v-model="newTask">

    <button v-on:click="tasks.push({content: newTask, status :false})">Thêm</button>

    <div v-for="(task, index) in tasks" :key="index">

      <input type="checkbox" v-model="task.status">

      <span v-bind:class="{success: task.status}">{{task.content}}</span>

      </div>

data(){

    return {

      isShow: true,

      attrColor: true,

      message: '',

      newTask: '',

      tasks: [

        { content: "spring" , status: false},

        { content: "authumn" , status: false},

        { content: "cover" , status: false},

        { content: "dance" , status: false},

      ]

    }

  },

# **Điều kiện** (nên hay không hiển thị / render phần tử)

## v-if

## v-else-if

## v-else

## v-show

## v-cloak

# **Modifier cho render** (thay đổi cách render một element)

## v-pre

## v-once

# **Nhận thức toàn cục** (cần các kiến thức vượt quá phạm vi của component)

## id

# **Thuộc tính không trùng lặp** (các thuộc tính cần giá trị không trùng lặp)

## ref

## key

**Tốt nhất là dùng key với v-if + v-else, nếu các phần tử là cùng loại (ví dụ cùng là <div>).**

Mặc định, Vue cố gắng cập nhật DOM càng hiệu quả càng tốt. Điều này có nghĩa là khi chuyển đổi giữa các phần tử cùng loại, Vue đơn giản chỉ chỉnh sửa phần tử hiện tại, thay vì gỡ bỏ rồi thêm một phần tử mới vào thay thế. Điều này có thể tạo ra [**các hiệu ứng phụ ngoài ý muốn**](https://jsfiddle.net/chrisvfritz/bh8fLeds/) nếu các phần tử này không nên được xem là như nhau.

Không nên

|  |
| --- |
| <div v-if="error">  Lỗi: {{ error }} </div> <div v-else>  {{ results }} </div> |

Nên

|  |
| --- |
| <div  v-if="error"  key="search-status" >  Lỗi: {{ error }} </div> <div  v-else  key="search-results" >  {{ results }} </div> |

|  |
| --- |
| <p v-if="error">  Lỗi: {{ error }} </p> <div v-else>  {{ results }} </div> |

## Slot

# **Ràng buộc một chiều** (kết hợp ràng buộc và sự kiện)

Trong component con

<input type="text" class="form-control" id="em-identifyPlace"

                v-bind:value="employee.identityPlace" />

prop:

props: {

    employee: Object

  },

Trong componet cha:

<EmployeeModalEdit

      ref="EmployeeModalEdit\_ref"

       :employee="employee"

      @edited="editedCofirm"

    />

Nhược điểm ; dữ liệu this.employee trong component con sẽ không bị thay đổi cho dù value trong input có thay đổi

# **Ràng buộc hai chiều** (kết hợp ràng buộc và sự kiện)

Như vd trê: Ta chỉ việc thay thế v-bind thành v-model

<input type="text" class="form-control" id="em-identifyPlace"

                v-model="employee.identityPlace" />

lưu ý trong trường hợp này employee của component cha cũng sẽ bị thay đổi

## v-model

# Ràng buộc form input

<https://vi.vuejs.org/v2/guide/forms.html>

# [Event modifier](https://vi.vuejs.org/v2/guide/events.html#Event-modifier)

## .stop

<!--  
 sự kiện click sẽ không được propagate (lan truyền)  
 điều này tương đương với event.stopPropagation()  
-->  
<a v-on:click.stop="doThis"></a>  
  
<!--

## .prevent

<!--  
 sự kiện submit sẽ không reload trang  
 điều này tương đương với event.preventDefault()  
-->  
<form v-on:submit.prevent="onSubmit"></form>

<!-- ta có thể nối modifier với nhau -->  
<a v-on:click.stop.prevent="doThat"></a>  
  
<!-- chỉ có modifier, không có phương thức xử lí -->  
<form v-on:submit.prevent></form>

.capture <!--  
 dùng "capture mode" khi thêm event listener  
 nghĩa là một sự kiện xảy ra với một phần tử con sẽ được xử lí với hàm doThis  
 trước khi được xử lí bởi phần tử đó  
 đọc thêm về event capturing: https://javascript.info/bubbling-and-capturing#capturing  
-->  
<div v-on:click.capture="doThis">...</div>  
  
<!--  
 chỉ kích hoạt phương thức xử lí nếu event.target là chính phần tử được click,  
 chứ không phải là một phần tử con  
-->  
<div v-on:click.self="doThat">...</div>

## .self

## .once

# Prop

## Truyền một giá trị tĩnh

sử dụng cú pháp

<ComponentName prop-name ="value"/>

## truyền dữ liệu động

thuộc tính data của vue, bạn có thể sử dụng

<template>

<ChildComponentName :color=color />

### Lưu ý :

Truyền dữ liệu phải trùng tên:

 :shopIdEdit="shopIdEdit"

Không nó sẽ được nhận giá trị sau khi chạy các hàm được gọi bởi phương thức $ của hàm cha :

Vì vậy chỉ sử dụng khác tên khi nó không đc sử dụng trong những hàm lên quan đến input và refs.show()

# Data

## Thuộc tính và phương thức

<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 'Toidicode.com'

}

},

});

</script>

! không nên tạo phương thức trong đáy

# Created

Hàm được chạy sau khi khởi tạo data

# Method

Đơn giản:

<p>Thông điệp: {{ message }}</p>

    <button v-on:click="message = 'lemon tree'">ok</button>

Trong giấu nháy kép là biểu thức

VD:

<template>

  <div>

    this is head

    <button @click="a += 1">A=A+1</button>

    <button @click="b += 1">B=B+1</button>

    <p>A + number = {{ addA() }}</p>

    <p>B + number = {{ addB() }}</p>

  </div>

</template>

<script>

export default {

  name: "Header",

  data() {

    return {

      a: 0,

      b: 0,

      number: 20,

    };

  },

  methods: {

    addA() {

      console.log("addA run");

      return this.a + this.number;

    },

    addB() {

      console.log("addB run");

      return this.b + this.number;

    },

  },

  props: {

    msg: String,

  },

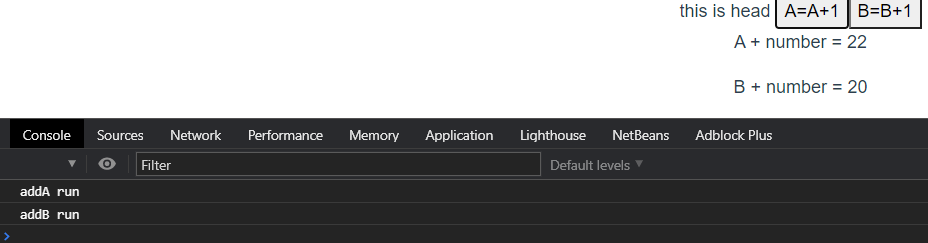
};

</script>

! Khi có bất kì sự thay đổi mào vuejs sẽ render lại toàn bộ template. Vì vậy cứ gặp lời gọi hàm là nó cạy bất kể dữ liệu bên trong hàm đấy có thay đổi hay không:

* điều này dẫn đến 1 việc đó là việc lãng phí tài nghuyên làm giảm hiệu suất:

Ta thấy khi nhấn 1 trong 2 button cả addA và addB được gọi



**v-on = @**

**có thể viết:**

var app = new Vue({

el: '#app',

data: {

message: 'hello world'

},

methods: {

initialVue : function () {

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

}

}

});

# Computed

Gần tương tự method -> điều khác ở đây là nó chỉ chạy khi data trong nó có sự thay đổi, cách gọi thì không cần ()

Khi sử dụng **computed property** thì giá trị của nó được lưu lại vào bộ nhớ đệm ([cache](https://vi.wikipedia.org/wiki/Cache_(tin_h%E1%BB%8Dc))), nếu chỉ render lại trang mà thành phần phụ thuộc không thay đổi thì sẽ lấy luôn giá trị trong bộ nhớ cache chứ không cần tính toán lại.

Còn sử dụng **methods** thì phải tính toán lại mỗi khi render lại trang (với bất kì thay đổi nào).

Nên sử dụng khi tao tác với dữ liệu trong data

* không thể gán giá trị của data trong component điều này dành cho watch
* không thể truyền parameter vào được

Vd:tương tự metod

<template>

  <div>

    this is head

    <button @click="a += 1">A=A+1</button>

    <button @click="b += 1">B=B+1</button>

    <p>A + number = {{ addA }}</p>

    <p>B + number = {{ addB }}</p>

  </div>

</template>

<script>

export default {

  name: "Header",

  data() {

    return {

      a: 0,

      b: 0,

      number: 20,

    };

  },

  computed:{

    addA() {

      console.log("addA run");

      return this.a + this.number;

    },

    addB() {

      console.log("addB run");

      return this.b + this.number;

    }

  },

Khi a thay đổi thì chỉ có hàm addA thay đổi  ,

Khi b thay đổi thì chỉ có hàm addB thay đổi

Khi number thay đổi cả hàm addA và addB thay đổi

# Watch

* hạn chế dùng watch

Trong framework VueJS, watcher được sử dụng để giám sát sự thay đổi của một đối tượng, và khi đối tượng thay đổi thì watcher sẽ đưa ra các hàm xử lí tương ứng. So với **computed\_property** thì **watcher** sẽ xử lí những tình huống phức tạp hơn, ví dụ như:

* Các hoạt động bất đồng bộ khi thay đổi dữ liệu
* Giới hạn việc tính toán và gán trạng thái trung gian cho tới khi có được kết quả cuối cùng.
* Các thiết lập giá trị ngay lập tức
* Nên sư dụng cho các tác vụ tìm kiếm
* Thường thì không nên return trong watch vì không thấy logic

Vd cú pháp:

Tên function == tên data

Có thể có 2 parameter là newValue, oldValue (thườn đặt tên như vậy cho dễ hiểu)

export default {

  name: "Header",

  data() {

    return {

      a: 0,

      b: 0,

      number: 20,

    };

  },

  watch: {

      a: function(){

        alert("run "+this.a);

      },

      b: function (newValue, oldValue) {

            console.log(' giá trị mới: ', newValue, ' giá trị cũ:', oldValue)

        },

  },

# So sánh computed, watch và methods:

|  | **Computed\_property** | **Methods** | **Watch** |
| --- | --- | --- | --- |
| Cho phép có tham số | Không | Có | Có |
| Thực hiện khi | Thành phần phụ thuộc thay đổi | Bất kì thay đổi nào (re-render) | Thuộc tính, đối tượng được giám sát bị thay đổi |
| Nên sử dụng khi | Khi muốn thao tác với dữ liệu trong instance, hạn chế tính toán lại | Cần đến một hàm thuần túy hoặc cần truyền tham số vào | Xử lí khi dữ liệu thay đổi nhiều, liên tục (VD dễ thấy nhất là thanh tìm kiếm) |

# [Filter](https://vi.vuejs.org/v2/guide/filters.html)

Vd:

 <div class="cell">{{ employee.DateOfBirth | getDate }}</div>

…

filters: {

    getDate(value){

      if(value!="" &&value!=null)

      var valuefomat = value.split("T")[0];

      return valuefomat;

    }

  },

Ta cũng có thể truyền param vào hàm **getDate** như sau

{{ employee.DateOfBirth | getDate("T") }}

…

filters: {

    getDate(value, arg1){

      if(value!="" &&value!=null)

      var valuefomat = value.split("T")[0] +arg1;

      return valuefomat;

    }

  },

## Map

## Fimcl

Slice

# Component

## Template

<template>

  <div id="app">

    <img alt="Vue logo" src="./assets/logo.png">

    <HelloWorld msg="Welcome to Your Vue.js App"/>

  </div>

</template>

<script>

import HelloWorld from './components/HelloWorld.vue'

export default {

  name: 'App',

  components: {

    HelloWorld

  }

}

### -root element

Mỗi template phải có 1 thẻ div bao bọc – điều này là bắt buộc

### - export default

Khi import 1 components phải phai báo component trong export default

## Script

## Style

# $

## $el

this.$el

trả về toàn bộ nội dung bên trong template

this.$el.querySelector("td div").innerHTML

lấy nội dung bên trong bộ trọn được tìm thấy đầu tiên

## $refs

Định nghĩa là referent của 1 thẻ được đánh dấu bao gồm cả 1 componet

Vd: <CreateModal ref="modal"

        id="createNV"

      />

* Cho biết thông tin về thẻ và các method của component đó

Vd: ta có thể gọi method mở form từ componet CreateModal như sau

methods: {

    openModal() {

     this.$refs.modal.show();

    },

  },

Cú pháp this.$refs.<tên refs>.<tên method>();

Vd: ta cũng có thể gọi computed trong component con như sau:

 methods: {

    openModal() {

      console.log(this.$refs.modal.chame);

     this.$refs.modal.chame;

  },

Vd: ta cũng có thể gọi data trong component con như sau:

methods: {

    openModal() {

      console.log(this.$refs.modal.visible);

    },

**Lưu ý: không thể truyền parameter vào được cho dù có khai báo**

Not openModal(parameter)

## $slots

## $props

## $emit

* Cách gửi dữ liệu từ component con đến component cha

Vd:

Lưu ý: không để this.$emit trong hàm bất đồng bộ

**ở component con**

methods: {

addToCart(item) {

this.$emit('update-cart', item)

}

}

Ta muốn chuyền 1 item (chỉ 1 mà thôi) với nhãn là 'update-cart' – cụ thể là 1 trình lắng nghe sự kiện

* Item có thể là mảng, chuỗi, object, biết,..

**ở component cha**

<shop-item v-for="item in this.items" :item="item" :key="item.id" @update-cart="updateCart">

</shop-item>

….

<script>

export default {

name: 'app',

data() {

return {

items: [

{ id: 205, name: 'Banana', price: 1, imageSrc: Banana },

{ id: 148, name: 'Orange', price: 2, imageSrc: Orange },

{ id: 248, name: 'Apple', price: 1, imageSrc: Apple }

],

cart: [],

total: 0

}

},

methods: {

updateCart(e) {

this.cart.push(e);

this.total = this.shoppingCartTotal;

},

emptyCart() {

this.cart = [];

this.total = 0;

}

},

}

</script>

Ta có: gán trình lắng nghe sự kiện @update-cart với method updateCart

<shop-item v-for="item in this.items" :item="item" :key="item.id" @update-cart="updateCart">

### Scoped

-giới hạn css trong 1 componen

<style coped>

# Axios

Quả lý các api

Cài đặt :

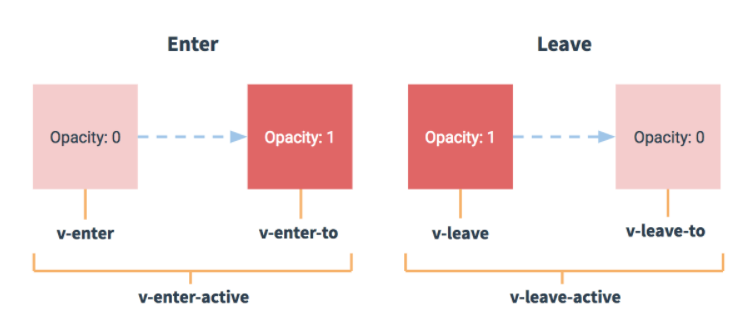
npm install --save axios

# trainsition

Cấu trúc tên class: .<transition name>**-**<[loại trạng thái](#_<loại_trạng_thái>)> {…}

## loại trạng thái

1. 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.
2. 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**](https://developer.mozilla.org/en-US/docs/Web/CSS/transition-duration), [**delay**](https://developer.mozilla.org/en-US/docs/Web/CSS/transition-delay), và [**hàm easing**](https://developer.mozilla.org/en-US/docs/Web/CSS/transition-timing-function) cho transition enter.
3. 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.
4. 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.
5. 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.
6. 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.



Vd:

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

.component-fade-enter-active {

  transition: all .3s ease-in-out;

}

.component-fade-leave-active {

  transition: all .3s ease-in-out;

}

.component-fade-enter, .component-fade-leave-to

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

  transform: translateY(-15px);

  opacity: 0;

}

## Sử dụng keyframe

* cho phép bạn tùy biến sâu hơn trong thời gian chạy
* không cần thiết lập các class trạng thái đầu cuối( các class không có từ active ở trên)
* mẹo nên để -leave-active là revesre khi keyframe của ẩn và hiện trùng nhau

vd:

<transition name="bounce">

.bounce-leave-active {

  animation: bounce-in 1s reverse;

}

@keyframes bounce-in {

  0% {

    transform:scale(0.6) translatex(-1000px) ;

    opacity: 0;

  }

  75% {

    transform:scale(1.2)  ;

    opacity: 1;

  }

  25% {

    transform:scale(1) translatex(0px) ;

    opacity: 1;

  }

Hoặc có thể tạo keyframe riêng cho leave

.bounce-enter-active {

  animation: bounce-in .5s;

}

.bounce-leave-active {

  animation: bounce-out .3s;

}

@keyframes bounce-in {

  0% {

    transform: translatey(-15px)  ;

    opacity: 0;

  }

  20% {

    transform:scale(1.05)  ;

    opacity: 1;

  }

  100% {

    transform:scale(1) translatey(0px) ;

    opacity: 1;

  }

}

@keyframes bounce-out {

  100% {

    transform: translatey(-15px) ;

    opacity: 0;

  }

}

# Slot

Mình khẳng định là nó khá mạnh

Ta có 1 component có phần dùng chung là các thẻ bao ngoài.

Các thẻ này có thể cho vào 1 component con. Phần của cha được đặt trong slot – đúng như tên gọi của nó

Vd tạo ra 1 component chung cho modal:

<template>

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

    <div class="h-confirm h-container-around-center" v-show="visible">

      <div class="h-icon-confirm">

        <i class="fas fa-check"></i>

      </div>

      <div class="h-confirm-content">

        <slot></slot>

      </div>

    </div>

  </transition>

</template>

Trong component cha:

<template>

<BaseModalConfirm ref="BaseModalConfirm\_ref">

  <p>{{message}}</p>

</BaseModalConfirm>

</template>

Slot sẽ thay thế <p>{{message}}</p>

\*Đôi khi slot được tạo ra chỉ để set hiệu ứng ^:^))

<template>

<transition name="fade" v-bind="$attrs" v-on="$listeners">

<slot></slot>

</transition>

</template>

## Slot được đặt tên

**slot trong vuejs**có một thuộc tính đặc biệt là **name**hỗ trợchúng ta có thể đặt tên cho **slot**đó. Trong một component có thể chứa nhiều **slot** với tên khác nhau và Vue.js cũng cho phép có một **slot**  không được đặt tên để chứa những nội dung không trùng khớp với bất kì slot name nào trong component.

<!-- Component card -->

<**div** class="card">

  <**header** class="card-header">

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

  </**header**>

  <**div** class="card-content">

    <**slot**></**slot**>

  </**div**>

  <**footer** class="card-footer">

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

  </**footer**>

</**div**>

Component cha sử dụng và truyền nội dung:

<div **class**="parent">

 <**card**>

   <**h2** slot="header">Vue.js thật tuyệt vời a mây zing goodjob</**h2**>

   <**p**>Framework JavaScript mạnh mẽ</**p**>

   <**p**>Một sản phẩm</**p**>

   <**p** slot="footer">Tạo ra bởi Evan You</**p**>

 </**card**>

</div>

Kết quả

<**div** class="parent">

 <**card**>

   <**h2** slot="header">Vue.js thật tuyệt vời a mây zing goodjob</**h2**>

   <**p**>Framework JavaScript mạnh mẽ</**p**>

   <**p**>Một sản phẩm</**p**>

   <**p** slot="footer">Tạo ra bởi Evan You</**p**>

 </**card**>

</**div**>

## Scoped slots

**Scoped slots** là một loại đặt biệt của **slot**giúp bạn có thể truyền dữ liệu từ component con lên component cha thông qua việc gán dữ liệu thông qua thuộc tính (nó cũng tương tự việc đưa dữ liệu vào props của component vậy). Chúng ta cùng xem qua ví dụ:

Mình có một child như sau:

<div **class**="children">

  <**slot** text="Hello from child!"></**slot**>

</div>

Như các bạn thấy, mình truyền vào slot một attribute là text kiểu như prop bạn thường thấy khi truyền vào component.

Tiếp theo đến component cha. Chúng ta sẽ lấy nội dung từ text thông qua thuộc tính slot-scope

<div **class**="parent">

  <**p**>Hello from parent</**p**>

  <**child**>

    <**p** slot-scope="props" v-text="props.text"></**p**>

  </**child**>

</div>

<!-- Hoặc có thể dùng tính năng destructuring syntax của ES2015 -->

<**p** slot-scope="{ text }" v-text="text"></**p**>

Những kiểu dữ liệu có thể sử dụng trong **scoped slots** có thể là **string**, **number**, **object**, **array**thậm chí bạn có thể truyền **method**.