Introduction to Vue 2

SE 331 Component Based Software Development

Class & Style Binding

- Simply using the inline css

```
" :style="{backgroundColor: variant.color}"
```

- Vue will bind the value of each CSS key
  - Replace with data()

```
<div v-for="variant in variants"
:key="variant.id"
@mouseover="updateImage(variant.image)"
class="color-circle"
:style="{backgroundColor: variant.color}"></div>
```

```
variants: [
    { id: 2234, color: 'green', image: './assets/images/socks_green.jpg' },
    { id: 2235, color: 'blue', image: './assets/images/socks_blue.jpg' }
],
```

We can bind many styles at once

- Bind to the styles variable

```
:style="styles"
```

- styles variable consists style information

```
data() {
    return {
        styles: {
            color: 'red',
            fontSize: '14px'
        },
```

Binding the attribute

- disabled
  - Html tag, make the button disable
- Use : to bind with data
  - If true, use disable
    - Otherwise remove the attribute

```
<button class=" button " :disabled='!inStock' @click="addToCart ">Add to Cart</button>
```

```
inStock: true,
```

Mapping with the class

- Same technique, select the class with the Boolean value

```
.disabledButton {
    background-color: #d8d8d8;
    cursor: not-allowed;
}
```

```
inStock: true,
```

```
<button class=" button "
        :class="{disabledButton: !inStock}"
        @click="addToCart ">Add to Cart</button>
```

For multiple classes

- Separate the object to selected

```
:class="{disabledButton: !inStock},{red:inStock}"
```

- Ternary Operator
  - Select classes in line

```
:class="inStock?'red':'disabledButton'"
```

  - The return type of the ternary operator should be string
  - The value is passed directly
    - everything in v-bind they will find as data except we specify the type

Camel Case Vs Kebab Case

- When bind to css use the Camel case

```
:style="{ backgroundColor: variant.color }">
```

- If you use the kebab case (background-color)
  - The JavaScript will design – as minus
- If you want to use the kebab case
  - Add the ‘ ‘ to make the name as string

```
:style="{ 'background-color': variant.color }
```

Computed Property

- Data is the static data
  - Provided at the beginning of the component creation
- Data need to be computed
  - If compute only, use the method
- Compute and want to show
  - Use computed property

Computed Property

- Under the computed object of the component input

```
computed: {
    title(){
        return this.brand + ' ' + this.product
    }
}
```

- Define as function
  - Need some computation
- But bind as simple property

```
<div class="product-info">
    <h1>{{ title}}</h1>
```

Usage

- Can get data from the data()
  - No need to load all data every time it call

```
variants: [
    { id: 2234, color: 'green', image: './assets/images/socks_green.jpg' }
    { id: 2235, color: 'blue', image: './assets/images/socks_blue.jpg' }
],
```

  - Instead of updating the image variable in data()

```
updateImage(variantImage) {
    this.image = variantImage
}
```

Update only the index

- We can get other related data

```
<div v-for="(variant,index) in variants" :key="variant.id"
@mouseover="updateVariant(index)"
class="color-circle"
```

```
updateVariant(index){
    this.selectedVariant = index;
}
```

- After we set the index

```
variants: [
    { id: 2234, color: 'green', image: './assets/images/socks_green.jpg' }
    { id: 2235, color: 'blue', image: './assets/images/socks_blue.jpg' }
],
```

  - Can refer to other data easily

How to refer to data?

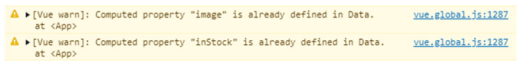
```
variants: [
    { id: 2234, color: 'green', image: './assets/images/socks_green.jpg' ,quantity: 50},
    { id: 2235, color: 'blue', image: './assets/images/socks_blue.jpg', quantity: 0 }
],
```

- Change only one variable
  - selectedVariant
  - Help us to find the other data which will be used to show

```
computed: {
    image(){
        return this.variants[this.selectedVariant].image;
    },
    inStock(){
        return this.variants[this.selectedVariant].quantity
    }
}
```

Binding Error

- {{ }} and v-bind try to find the value from computed, and data()
- Use the name only
  - So if we have functions or variable with the same name
    - Error will be occurred

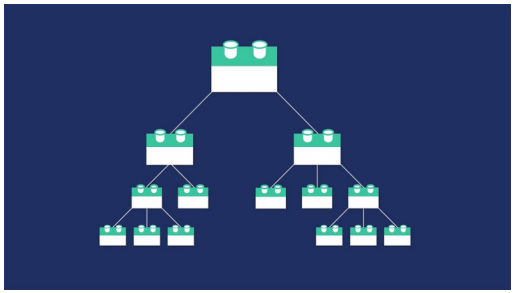


Components

- Creating the components
- A block of codes
  - One page can consist with many block
  - Previously we have one block

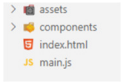
Components

- Each component(block)
  - May be composed with multiple components
- Can be “parent” component
  - Contains “child” components

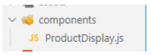


Naming Convention

- The components is stored in components folder



- The component should be the JS file



Creating the component

- Using the app.component

```
app.component('product-display',{})
```

```
const app = Vue.createApp({
```

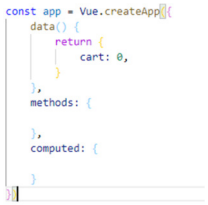
- The component can be used under the app mounted element only
- The html element name which the component will be placed is define as the first input parameter
- The second parameter provided the data(), method, computed as the app
  - Use the same logic
  - Easier to understand

The new value in the input parameter

- template
  - Replace the html code when the component is called
  - Separate the code in the parent to make the code cleaner
  - The literal template is used
  - /*html*/ keyword for
    es6-string-html extension
    just to highlight the html syntax

Then move all required data to the new component

No more these data in the main.js



To use the component

- Make a link from html file to the components file

```
<script src="./main.js "></script>
<!-- Import Components -->
<script src="./components/ProductDisplay.js"></script>
```

Place the html element in the specific location

```
<body>
    <div id="app">
        <div class="nav-bar"></div>

        <div class="cart">Cart({{ cart }})</div>

        <product-display></product-display>

    </div>
```

```
app.component('product-display',{
    template:
        /*html*/
        `<div class="product-display">
        <div class="product-container">
          <div class="product-image">
            <img v-bind:src="image">
          </div>
          <div class="product-info">
            <h1>{{ title }}</h1>

            <p v-if="inStock">In Stock</p>
            <p v-else>Out of Stock</p>

            <div
              v-for="(variant, index) in variants"
              :key="variant.id"
              @mouseover="updateVariant(index)"
              class="color-circle"
              :style="{ backgroundColor: variant.color }">
            </div>

            <button
              class="button"
              :class="{ disabledButton: !inStock }"
              :disabled="!inStock"
              v-on:click="addToCart">
              Add to Cart
            </button>
          </div>
        </div>
      </div>`
})
```

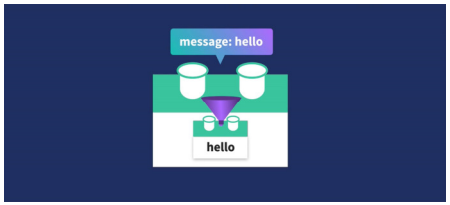
```
    data() {
        return {
            product: 'Shoes',
            brand: 'SE 331',

            inventory: 100,
            details: ['50% cotton', '30% wool', '20% polyester'],
            variants: [
                { id: 2234, color: 'green', image: './assets/images/socks_green.jpg' ,quantity: 50},
                { id: 2235, color: 'blue', image: './assets/images/socks_blue.jpg', quantity: 0 }
            ],
            activeClass: true,
            selectedVariant:0
        }
    },
    methods: {
        addToCart() {
            this.cart += 1
        },
        updateImage(variantImage) {
            this.image = variantImage
        },
        updateVariant(index){
            this.selectedVariant = index;
        }
    },
    computed: {
        title(){
            return this.brand + ' ' + this.product
        },
        image(){
            return this.variants[this.selectedVariant].image;
        },
        inStock(){
            return this.variants[this.selectedVariant].quantity
        }
    }
```

Which part should be created as Component?

- The components that can be reused
  - Use the same presentation
  - Different data
- How to set the different data?

props element

- Sending the data
  - From Parent to child
- Help for reuse data



Add props in the child component

```
app.component('product-display', {
  props: {
    premium: {
      type: Boolean,
      required: true
    }
  },
```

- Type is defined
  - To case the input data
- Required
  - State that we need to put it or not

To send the data

```
const app = Vue.createApp({
    data() {
        return {
            cart: 0,
            premium: true
        }
    },
```

- V-bind to the attribute

```
<product-display :premium="premium"></product-display>
```

```
!v>
```

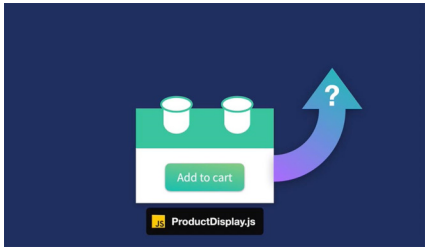
- The receive value is a variable
  - Can be used only inside the component

```
shipping() {
  if (this.premium){
    return 'Free'
  }
  return 30
}
```

```
<p v-if="inStock">In Stock</p>
<p v-else>Out of Stock</p>
<p>Shipping: {{shipping}}</p>
```

Sending the events

- When the child component receive the event
- We may want to send the event to the parent
  - To update some variable in the parent component



What to do?

- When some event occurred
  - Send the event to someone else
  - Using emit command

```
<button
  class="button"
  :class="{ disabledButton: !inStock }"
  :disabled="!inStock"
  v-on:click="addToCart">
  Add to Cart
</button>
```

```
methods: {
  addToCart() {
    this.$emit('add-to-cart')
  },
```

$emit

- Create the event
- Event name is the first input parameter
- Think like click
  - Event happen
  - Need some one to map handler for the event

Mapping the event handler

- Using the simple mapping

```
methods: {
  addToCart() {
    this.$emit('add-to-cart')
  },
```

```
<product-display :premium="premium" @add-to-cart="updateCart"></product-display>
```

```
methods: {
    updateCart() {
        this.cart += 1;
    }
},
```

We can send other information with event

- As the second parameters

```
addToCart() {
  this.$emit('add-to-cart',this.variants[this.selectedVariant].id)
},
```

- When handle the event, the input parameter are required
  - The Framework will inject the input parameter from the object sent by the event

```
<product-display :premium="premium" @add-to-cart="updateCart"></product-display>
```

```
methods: {
    updateCart() {
        this.cart += 1;
    }
},
```

```
,
methods: {
    updateCart(id) {
        this.cart.push(id);
    }
},
```

Two-ways binding

- Previously is a one-way binding
  - Link data from JavaScript to show in the html
  - Receive the event call from html to call JavaScript function
- 2-Way binding
  - Link the data from the JavaScript to show in the HTML
  - Receive the Data from html to put in the variable in the JavaScript
  - With in single command

Form

- Html to receive the data
- Contain the input values with it name
- Contain the button with type submit
  - When click, the data will be sent to the action attribute target
  - If action is blank, the data is sent to the same page

V-model

- Link the data via the variable name

```
data() {
  return {
    name: '',
    review: '',
    rating: null
  }
}
```

```
<input id="name" v-model="name">

<label for="review">Review:</label>
<textarea id="review" v-model="review"></textarea>
```

V-model

- Force type
  - Default
    - The data receive from the input is String
- Force the type
  - So the type in JavaScript can be used

```
<select id="rating" v-model.number="rating">
  <option>5</option>
  <option>4</option>
  <option>3</option>
  <option>2</option>
  <option>1</option>
</select>
```

V-model modifiers

- To force the bound data
- .number
  - typecasts the value as a number
- .trim
  - Remove the white space in front of the text, and after the text
- .lazy
  - Sync the data after change event instead of the input event

Mapping the data

- Data is changed all the time the user input data
- To submit the data
  - The event handler should be managed
- Normally when the submit button is clicked
  - The submit event is fired
  - The page will reload
- Block the default event

```
<form class="review-form" @submit.prevent="onSubmit">
 <h3>Leave a review</h3>
```

```
data() {
  return {
    name: '',
    review: '',
    rating: null
  }
},
methods:{
    onSubmit() {
        let productReview = {
          name: this.name,
          review: this.review,
          rating: this.rating,
        }
        this.$emit('review-submitted', productReview)

        this.name = ''
        this.review = ''
        this.rating = null
      }
}
```

The event is sent

- The event handler is required in the parent component

```
<review-form @review-submitted="addReview"></review-form>
```

```
addReview(review){
  this.reviews.push(review)
}
```

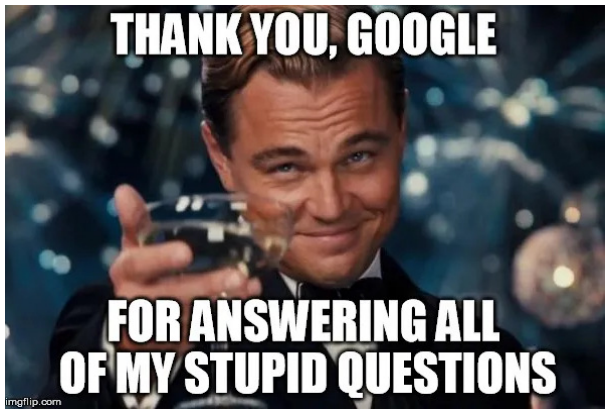
Then we can use the data to link with other form

- In the same parents

```
<review-list :reviews="reviews"></review-list>
<review-form @review-submitted="addReview"></review-form>
```

In summarize

- Web Framework
  - Data binding
    - Binding the value
    - Binding to attribute
    - Binding style and class
  - Creating component
    - Update components
    - Receive the data from parents
    - Send event to parents
  - Form
    - 2-way binding


THANK YOU, GOOGLE
FOR ANSWERING ALL
OF MY STUPID QUESTIONS
imgflip.com

Q&A