Sintaxe de modelo

Interpolação de texto

- Message: {{ msg }}

A marca bigode será substituída pelo valor da msg propriedade da instância do componente correspondente. Ele também será atualizado sempre que a msg propriedade for alterada.

Ligações de atributos

v-bind

-
- Sintaxe abreviada: <div :id="dynamicId">
- Se o valor vinculado for nullou undefined, o atributo será removido do elemento renderizado.

Atributos booleanos

- <button :disabled="isButtonDisabled">Button</button>

Vinculando vários atributos dinamicamente Importante

Se você tiver um objeto JavaScript representando vários atributos com esta aparência:

```
data() {
  return {
    objectOfAttrs: {
      id: 'container',
      class: 'wrapper'
    }
  }
}
```

```
<div v-bind="objectOfAttrs"></div>
```

Usando expressões JavaScript

Cada ligação pode conter apenas uma única expressão . Uma expressão é um pedaço de código que pode ser avaliado para um valor.

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div :id="`list-${id}`"></div>
```

Chamando funções

As funções chamadas dentro das expressões de ligação serão chamadas sempre que o componente for atualizado, portanto, não devem ter nenhum efeito colateral, como alteração de dados ou acionamento de operações assíncronas.

```
<span :title="toTitleDate(date)">
  {{ formatDate(date) }}
</span>
```

Diretivas

O trabalho de uma diretiva é aplicar atualizações reativamente ao DOM quando o valor de sua expressão muda**.** Aqui, a v-if diretiva removeria/inseriria o <p> elemento com base na veracidade do valor da expressão seen.

```
<p v-if="seen">Now you see me</p>
```

Argumentos Dinâmicos

```
<a v-bind:[attributeName]="url"> ... </a>
<!-- shorthand -->
<a :[attributeName]="url"> ... </a>
```

Aqui, attributeNameserá avaliado dinamicamente como uma expressão JavaScript e seu valor avaliado será usado como valor final para o argumento. Por exemplo, se a instância do seu componente tiver uma propriedade data, attributeName, cujo valor é "href", essa vinculação será equivalente a v-bind:href.

Da mesma forma, você pode usar argumentos dinâmicos para vincular um manipulador a um nome de evento dinâmico:

```
<a v-on:[eventName]="doSomething"> ... </a>
<!-- shorthand -->
<a @[eventName]="doSomething">
```

Reatividade

Com Options API, nós usamos o data para declarar o estado de um componente. Quaisquer propriedades de nível superior deste objeto são representadas na instância do componente (**this** em métodos e ganchos de ciclo de vida). Podemos passar **this** como um argumento para acessar o estado Vue em classes definidas com JavaScript.
Essas propriedades de instância são adicionadas apenas quando a instância é criada pela primeira vez. Onde necessário, use **null, undefined** ou algum outro valor de espaço reservado para propriedades onde o valor desejado ainda não está disponível. É possível adicionar uma nova propriedade diretamente this sem incluí-la em data. No entanto, as propriedades adicionadas dessa maneira não poderão acionar atualizações reativas.

Métodos

Você deve evitar usar funções de seta ao definir methods, pois isso impede que o Vue associe o this valor apropriado. Normalmente os métodos são mais usados como ouvintes de eventos.
@click=”método”

```
export default {
  methods: {
    increment: () => {
      // BAD: no `this` access here!
```

```
  }
```

Reatividade Profunda

No Vue, o estado é profundamente reativo por padrão. Isso significa que você pode esperar que as alterações sejam detectadas mesmo quando você modifica objetos ou matrizes aninhadas.

```
export default {
  data() {
    return {
      obj: {
        nested: { count: 0 },
        arr: ['foo', 'bar']
      }
    }
  },
  methods: {
    mutateDeeply() {
      // these will work as expected.
      this.obj.nested.count++
      this.obj.arr.push('baz')
    }
  }
}
```

Propriedades computadas

Eles são como métodos que usam dados reativos para calcular uma lógica mais complexa e evitar que o modelo fique poluído. Sempre que os dados forem alterados, a propriedade será atualizada.

```
computed: {
  publishedBooksMessage() {
    return this.author.books.length > 0 ? 'Yes' : 'No'
  }
}
```

Cache:

- A diferença é que as propriedades computadas são armazenadas em cache com base em suas dependências reativas. Uma propriedade computada só será reavaliada quando algumas de suas dependências reativas forem alteradas. Em comparação, uma invocação de método sempre executará a função sempre que ocorrer uma nova renderização. Por que precisamos de cache? Imagine que temos uma propriedade computada cara list, que requer um loop em uma matriz enorme e muitos cálculos. Então, podemos ter outras propriedades calculadas que, por sua vez, dependem de list.

Classes e estilos

Podemos passar um objeto para :class para mudar a classe dinamicamente.

```
<div :class="{ active: isActive }"></div>
```

No exemplo acima, se isActive tiver um valor verdadeiro a classe active é adicionada ao elemento. Podemos ter múltiplas classes:

```
data() {
  return { isActive: true, hasError: false }
}

<div class="static" :class="{ active: isActive, 'text-danger': hasError }" ></div>
```

Podemos passar um objeto com as classes relacionadas **Importante**

```
data() {
  return {
    classObject: {
      active: true,
     'text-danger': false
   }
 }
}

<div :class="classObject"></div>
```

Também podemos vincular a uma propriedade computada que retorna um objeto. **Importante**

Usando arrays para definir várias classes

```
const isActive = ref('active');
const isOutline = ref('outline');

<div :class="[activeClass, errorClass]"></div>

<style>
  …
</style>
```

Quando você usa o classatributo em um **componente** com um **único** elemento raiz, essas classes serão adicionadas ao elemento raiz do componente e mescladas com qualquer classe existente que já esteja nele.

```
<!-- child component template -->
<p class="foo bar">Hi!</p>

<!-- when using the component -->
<MyComponent class="baz boo" />

<p class="foo bar baz boo">Hi</p>
```

Se seu componente tiver vários elementos raiz, você precisará definir qual elemento receberá essa classe. Você pode fazer isso usando a $attrspropriedade do componente:

```
<p :class="$attrs.class">Hi!</p>
<span>This is a child component</span>
```

Renderização Condicional

Por ser uma diretiva, ela deve ser anexada a um **único elemento**. Mas e se quisermos alternar mais de um elemento? Neste caso podemos usar v-if com <template>, que serve como um invólucro invisível. O resultado renderizado final não incluirá o <template>.

v-if é a renderização condicional "real" porque garante que os ouvintes de evento e os componentes filho dentro do bloco condicional sejam devidamente destruídos e recriados durante as alternâncias.

v-if também é preguiçoso : se a condição for falsa na renderização inicial, não fará nada - o bloco condicional não será renderizado até que a condição se torne verdadeira pela primeira vez.

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

v-show

A propriedade v-show sempre será renderizado e permanecerá no DOM; v-show apenas alterna a display propriedade CSS do elemento.

v-show é muito mais simples - o elemento é sempre renderizado independentemente da condição inicial, com alternância baseada em CSS.

v-shown ão suporta o <template>elemento, nem funciona com v-else.

v-if x v-show
De um modo geral, v-if tem custos de alternância mais altos, enquanto v-show tem custos iniciais de renderização mais altos. Portanto, prefira v-show se precisar alterar algo com muita frequência e prefira v-if se for improvável que a condição mude no tempo de execução.