Fundamentales Vue I

Instructor: Abner Saavedra
Fecha: sept. de 2024
Email: ingenieroabnersaavedra@gmail.com
GracoSoft Centro Empresarial Plaza Madrid, piso 9, oficinas 9-7 a la 9-10

Agenda

- **Creación de una aplicación Vue**
  - La instancia de la aplicación
  - El componente raíz
  - Montar la aplicación
    - Plantilla de componente raíz en DOM
  - Configuraciones de la aplicación
    - Varias instancias de aplicación
- **Sintaxis de plantilla**
  - Interpolación de texto
  - HTML sin procesar
  - Vinculaciones de atributos
    - Taquigrafía
    - Taquigrafía del mismo nombre
    - Atributos booleanos
    - Vinculación dinámica de múltiples atributos
  - Uso de expresiones de JavaScript
    - Sólo expresiones
    - Funciones de llamada
    - Acceso global restringido
  - Directivas
    - v-if
    - v-else
    - v-show
    - v-for
    - v-for v-if

Creación de una aplicación Vue

La instancia de la aplicación

Cada aplicación Vue comienza creando una nueva instancia de aplicación con la función createApp:

```
import { createApp } from 'vue'

const app = createApp({
  /* root component options */
})
```

El componente raíz

El objeto al que estamos pasando es, de hecho, un componente. Cada aplicación requiere un "componente raíz" que puede contener otros componentes como sus elementos secundarios. createApp

Si está utilizando componentes de un solo archivo, normalmente importamos el componente raíz de otro archivo:

Si bien muchos ejemplos de este curso solo necesitan un solo componente, la mayoría de las aplicaciones reales se organizan en un árbol de componentes anidados y reutilizables. Por ejemplo, el árbol de componentes de una aplicación Todo podría tener el siguiente aspecto:

```
App (root component)
├─ TodoList
```

```
│   └─ TodoItem
│       ├─ TodoDeleteButton
│       └─ TodoEditButton
└─ TodoFooter
    ├─ TodoClearButton
    └─ TodoStatistics
```

En secciones posteriores del curso, discutiremos cómo definir y componer varios componentes juntos. Antes de eso, nos centraremos en lo que sucede dentro de un solo componente.

Montaje de la aplicación

Una instancia de aplicación no representará nada hasta que se llame a su método. Espera un argumento "contenedor", que puede ser un elemento DOM real o una cadena de selección: .mount()

Html:

```
<div id="app"></div>
```

Js:

```
app.mount('#app')
```

El contenido del componente raíz de la aplicación se representará dentro del elemento contenedor. El elemento contenedor en sí no se considera parte de la aplicación.

Siempre se debe llamar al método después de que se hayan realizado todas las configuraciones de la aplicación y los registros de recursos. Tenga en cuenta también que su valor devuelto, a diferencia de los métodos de registro de activos, es la instancia del componente raíz en lugar de la instancia de la aplicación. .mount()

Plantilla de componente raíz en DOM

La plantilla para el componente raíz suele formar parte del propio componente, pero también es posible proporcionar la plantilla por separado escribiéndola directamente dentro del contenedor de montaje:

```
<div id="app">
  <button @click="count++">{{ count }}</button>
</div>
```

```
import { createApp } from 'vue'
```

```
const app = createApp({
  data() {
    return {
      count: 0
    }
  }
})
```

```
app.mount('#app')
```

Vue usará automáticamente el contenedor como plantilla si el componente raíz aún no tiene una opción.innerHTMLtemplate

Las plantillas In-DOM se utilizan a menudo en aplicaciones que utilizan Vue sin un paso de compilación. También se pueden utilizar junto con marcos del lado del servidor, donde el servidor puede generar dinámicamente la plantilla raíz.

Configuraciones de la aplicación

La instancia de la aplicación expone un objeto que nos permite configurar algunas opciones a nivel de aplicación, por ejemplo, definir un controlador de errores a nivel de aplicación que capture errores de todos los componentes descendientes: .config

```
app.config.errorHandler = (err) => {
  /* handle error */
}
```

La instancia de la aplicación también proporciona algunos métodos para registrar recursos con ámbito de aplicación. Por ejemplo, registrar un componente:

```
app.component('TodoDeleteButton', TodoDeleteButton)
```

Esto hace que esté disponible para su uso en cualquier lugar de nuestra aplicación. Discutiremos el registro de componentes y otros tipos de activos en secciones posteriores de la guía. También puede examinar la lista completa de API de instancia de aplicación en su referencia de API. TodoDeleteButton

¡Asegúrese de aplicar todas las configuraciones de la aplicación antes de montar la aplicación!

Varias instancias de aplicación

No está limitado a una sola instancia de aplicación en la misma página. La API permite que varias aplicaciones de Vue coexistan en la misma página, cada una con su propio alcance para la configuración y los activos globales: createApp

```
const app1 = createApp({
  /* ... */
})
app1.mount('#container-1')
```

```
const app2 = createApp({
  /* ... */
})
app2.mount('#container-2')
```

Si está utilizando Vue para mejorar el HTML renderizado por el servidor y solo necesita Vue para controlar partes específicas de una página grande, evite montar una sola instancia de aplicación Vue en toda la página. En su lugar, cree varias instancias de aplicaciones pequeñas y móntelas en los elementos de los que son responsables.

Sintaxis de la plantilla

Vue utiliza una sintaxis de plantilla basada en HTML que le permite vincular declarativamente el DOM renderizado a los datos de la instancia del componente subyacente. Todas las plantillas de Vue son HTML sintácticamente válidos que pueden ser analizadas por navegadores que cumplan con las especificaciones y analizadores de HTML.

Bajo el capó, Vue compila las plantillas en código JavaScript altamente optimizado. Combinado con el sistema de reactividad, Vue puede determinar de manera inteligente el número mínimo de componentes para volver a renderizar y aplicar la cantidad mínima de manipulaciones del DOM cuando cambia el estado de la aplicación.

Si está familiarizado con los conceptos de Virtual DOM y prefiere la potencia bruta de JavaScript, también puede escribir directamente funciones de renderizado en lugar de plantillas, con soporte opcional para JSX. Sin embargo, tenga en cuenta que no disfrutan del mismo nivel de optimizaciones en tiempo de compilación que las plantillas.

Interpolación de texto

La forma más básica de enlace de datos es la interpolación de texto mediante la sintaxis "Bigote" (llaves dobles):

```
<span>Message: {{ msg }}</span>
```

La etiqueta bigote se reemplazará por el valor de la propiedad de la instancia del componente correspondiente. También se actualizará cada vez que cambie la propiedad. msg msg

HTML sin procesar

Los bigotes dobles interpretan los datos como texto sin formato, no HTML. Para generar HTML real, deberá utilizar la v-html directiva :

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

Usando interpolación de texto: Esto debería ser rojo.
Usando la directiva v-html: Esto debe ser rojo.

Aquí nos encontramos con algo nuevo. El **v-html** atributo que estás viendo se llama directiva. Las directivas tienen el prefijo para **v-** indicar que son atributos especiales proporcionados por Vue y, como habrás adivinado, aplican un comportamiento reactivo especial al DOM renderizado. Aquí, básicamente estamos diciendo "mantener el HTML interno de este elemento actualizado con la **rawHtml** propiedad en la instancia activa actual".

El contenido de la propiedad **span** se reemplazará con el valor de la **rawHtml** propiedad, interpretado como HTML simple; se ignoran los enlaces de datos. Tenga en cuenta que no puede usar **v-html** para componer partes de plantilla, porque Vue no es un motor de plantillas basado en cadenas. En cambio, se prefieren los componentes como la unidad fundamental para la reutilización y composición de la interfaz de usuario.

Advertencia de seguridad

La representación dinámica de código HTML arbitrario en su sitio web puede ser muy peligrosa, ya que puede generar vulnerabilidades XSS . Úselo solo **v-html** en contenido confiable y **nunca** en contenido proporcionado por el usuario.

Vinculaciones de atributos

Los bigotes no se pueden utilizar dentro de atributos HTML. En su lugar, utilice una v-binddirectiva :

```
<div v-bind:id="dynamicId"></div>
```

La **v-bind** directiva indica a Vue que mantenga el atributo del elemento **id** sincronizado con la **dynamicId** propiedad del componente. Si el valor enlazado es **null** o **undefined**, entonces el atributo se eliminará del elemento representado.

Taquigrafía

Debido a que v-bindse usa con tanta frecuencia, tiene una sintaxis abreviada dedicada:

```
<div :id="dynamicId"></div>
```

Los atributos que comienzan con **:** pueden verse un poco diferentes del HTML normal, pero de hecho es un carácter válido para los nombres de atributos y todos los navegadores compatibles con Vue pueden analizarlo correctamente. Además, no aparecen en el marcado final. La sintaxis abreviada es opcional, pero probablemente la apreciará cuando aprenda más sobre su uso más adelante.

Taquigrafía del mismo nombre

- Solo compatible con 3.4+

Si el atributo tiene el mismo nombre que el valor de JavaScript que se va a vincular, la sintaxis se puede acortar aún más para omitir el valor del atributo:

```
<!-- same as :id="id" -->
<div :id></div>

<!-- this also works -->
<div v-bind:id></div>
```

Esto es similar a la sintaxis abreviada de propiedades que se utiliza al declarar objetos en JavaScript. Tenga en cuenta que esta es una función que solo está disponible en Vue 3.4 y versiones posteriores.

Atributos booleanos

Los atributos booleanos son atributos que pueden indicar valores verdaderos o falsos por su presencia en un elemento. Por ejemplo, disabled es uno de los atributos booleanos más utilizados.

v-bind funciona un poco diferente en este caso:

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

El **disabled** atributo se incluirá si **isButtonDisabled** tiene un valor verdadero. También se incluirá si el valor es una cadena vacía, manteniendo la coherencia con **<button disabled="">.** Para otros valores falsos, se omitirá el atributo.

Vinculación dinámica de múltiples atributos

Si tiene un objeto JavaScript que representa múltiples atributos que se ve así:

```
const objectOfAttrs = {
  id: 'container',
  class: 'wrapper',
  style: 'background-color:green'
}
```

Puedes vincularlos a un solo elemento usando v-bindsin un argumento:

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

Uso de expresiones de JavaScript

Hasta ahora, solo hemos realizado enlaces a claves de propiedades simples en nuestras plantillas. Pero Vue realmente admite todo el poder de las expresiones de JavaScript dentro de todos los enlaces de datos:

```
{{ number + 1 }}
```

```
{{ ok ? 'YES' : 'NO' }}
```

```
{{ message.split('').reverse().join('') }}
```

```
<div :id="`list-${id}`"></div>
```

Estas expresiones se evaluarán como JavaScript en el ámbito de datos de la instancia del componente actual.

En las plantillas de Vue, las expresiones de JavaScript se pueden utilizar en las siguientes posiciones:

Interpolaciones de texto interno (bigotes)
En el valor del atributo de cualquier directiva de Vue (atributos especiales que comienzan con **v-**)

Sólo expresiones

Cada enlace solo puede contener **una única expresión**. Una expresión es un fragmento de código que se puede evaluar como un valor. Una comprobación sencilla es si se puede utilizar después de **return**.

Por lo tanto, lo siguiente **NO** funcionará:

```
<!-- this is a statement, not an expression: -->
{{ var a = 1 }}

<!-- flow control won't work either, use ternary expressions -->
{{ if (ok) { return message } }}
```

Funciones de llamada

Es posible llamar a un método expuesto por un componente dentro de una expresión de enlace:
Plantilla:

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

Funciones de llamada

Es posible llamar a un método expuesto por un componente dentro de una expresión de enlace:

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

CONSEJO

Las funciones llamadas dentro de expresiones de enlace se llamarán cada vez que se actualice el componente, por lo que **no** deberían tener efectos secundarios, como cambiar datos o activar operaciones asincrónicas.

Acceso global restringido

Las expresiones de plantilla están protegidas y solo tienen acceso a una lista restringida de variables globales. La lista expone variables globales integradas de uso común, como **Math** y **Date**.

Las variables globales que no se incluyen explícitamente en la lista, por ejemplo, las propiedades adjuntas por el usuario en **window**, no estarán accesibles en las expresiones de plantilla. Sin embargo, puede definir explícitamente variables globales adicionales para todas las expresiones de **Vue** agregándolas a app.config.globalProperties.

Directivas

Las directivas son atributos especiales con el **v-** prefijo. Vue ofrece una serie de directivas integradas , entre las que se incluyen v-**html** y **v-bind** que hemos presentado anteriormente.

Se espera que los valores de los atributos de directiva sean expresiones JavaScript individuales (con la excepción de **v-for**, **v-on** y **v-slot**, que se analizarán en sus respectivas secciones más adelante). **El trabajo de una directiva es aplicar actualizaciones reactivas al DOM cuando cambia el valor de su expresión**.

Tomemos **v-if** como ejemplo:

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

Aquí, la **v-if** directiva eliminaría/insertaría el <p>elemento en función de la veracidad del valor de la expresión **seen**.

v-if

- doc v-if
- La directiva **v-if** se usa para mostrar o ocultar un elemento de la plantilla.

```
<script setup>
    const name = "Vue 3";
    const active = true;
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <h2 v-if="active">v-if en acción</h2>
</template>
```

v-else

Un elemento **v-else** debe seguir inmediatamente a un **v-if** o un **v-else-if** elemento; de lo contrario, no se reconocerá.

Ejemplos de uso correcto **v-else**:

```
<script setup>
    const name = "Vue 3";
    const active = false;
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <h2 v-if="active">v-if en acción</h2>
    <h2 v-else>v-else en acción</h2>
</template>
```

```
<script setup>
    const name = "Vue 3";
    const active = null;
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <h2 v-if="active === true">Estoy en true</h2>
    <h2 v-else-if="active === false">Estoy en false</h2>
    <h2 v-else>Estoy indeciso</h2>
</template>
```

Uso incorrecto de **v-else**:

```
<template>
    <h1>Hola {{ name }}!</h1>
    <h2 v-if="active === true">Estoy en true</h2>
    <h2 v-else-if="active === false">Estoy en false</h2>
    <p>Lorem ipsum dolor sit</p>
    <h2 v-else>Estoy indeciso</h2>
</template>
```

v-show

- La directiva **v-show** se usa para mostrar o ocultar un elemento de la plantilla.
- **v-show** solo cambia la propiedad display CSS del elemento.
- En términos generales, **v-if** tiene costos de alternancia más altos mientras que **v-show** tiene costos de renderización inicial más altos. Así que prefiera **v-show** si necesita alternar algo con mucha frecuencia, y prefiera **v-if** si es poco probable que la condición cambie en el tiempo de ejecución.

```
<script setup>
    const name = "Vue 3";
    const active = false;
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <h2 v-show="active">Estoy en true</h2>
</template>
```

v-for

- doc v-for
- La directiva **v-for** se usa para iterar sobre una lista de elementos.
- Para darle a Vue una pista para que pueda rastrear la identidad de cada nodo y, por lo tanto, reutilizar y reordenar los elementos existentes, debe proporcionar un atributo **key** único para cada elemento.

```
<script setup>
    const name = "Vue 3";
    const arrayFrutas = ["", "", "", "", ""];
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <ul>
        <li v-for="(fruta, index) in arrayFrutas" :key="index">
            {{ fruta }}
        </li>
    </ul>
</template>
```

```
<script setup>
    const name = "Vue 3";
    const arrayFrutas = [{
            name: "Manzana",
            price: "$1.00",
            description: "Una manzana",
        },{
            name: "Pera",
            price: "$2.00",
            description: "Una pera",
        },
        {
            name: "Naranja",
            price: "$3.00",
            description: "Una naranja",},];
</script>
```

```
<template>
    <h1>Hola {{ name }}!</h1>
    <ul>
        <li v-for="fruta in arrayFrutas" :key="name">
            {{ fruta.name }} - {{ fruta.price }} - {{ fruta.description }}
        </li>
    </ul>
</template>
```

Iterar objetos:

```
<script setup>
    const name = "Vue 3";

    const fruta = {
        name: "Naranja",
        price: "$3.00",
        description: "Una naranja",
    };
</script>

<template>
    <h1>Hola {{ name }}!</h1>
    <ul>
        <li v-for="(value, propiedad) in fruta">
            {{ propiedad }} : {{ value }}
        </li>
    </ul>
</template>
```

v-for v-if

- doc v-for v-if(opens new window)
- Cuando existen en el mismo nodo, **v-if** tiene una prioridad más alta que **v-for**.
- Eso significa que la condición **v-if** no tendrá acceso a las variables del alcance de **v-for**:

```
<script setup>
const name = "Vue 3";
const arrayFrutas = [
    {
        name: "Manzana",
        price: "$1.00",
        description: "Una manzana",
        stock: 0,
    },
    {
        name: "Pera",
        price: "$2.00",
        description: "Una pera",
        stock: 10,
    },
    {
        name: "Naranja",
        price: "$3.00",
        description: "Una naranja",
        stock: 20,
    },
];
</script>
<template>
    <h1>Hola {{ name }}!</h1>
    <ul>
        <li
            v-for="fruta in arrayFrutas"
            :key="fruta.name"
            v-if="fruta.stock > 0">
            {{ fruta }}
        </li>
    </ul>
</template>
```

Solución:

```
<ul>
    <template v-for="fruta in arrayFrutas" :key="fruta.name">
        <li v-if="fruta.stock > 0">
            {{ fruta }}
        </li>
    </template>
</ul>
```

Bibliografía

https://vuejs.org/guide/essentials/application.html

https://vuejs.org/guide/essentials/template-syntax.html

Fundamentos Vue 3 (composition api) | Vue Udemy (bluuweb.github.io)

GRACOSOFT ES EXCELENCIA EDUCATIVA