We Learn Together

Vue.js

GDG Pontevedra

Antes de comezar

Nos exemplos de código mostrados nesta guía omítense liñas como a importación da libraría de Vue.js, para non repetir certo código unha e outra vez.

Este documento está baseado na guía oficial de Vue.js e no tutorial da páxina tutorialspoint.

Primeiros pasos

Para usar Vue.js nun arquivo HTML, o primeiro que debes facer é importar a librería mediante a etiqueta <script>:

```
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
</script>
```

Podemos asegurarnos de que funciona correctamente creando unha aplicación "Hello World". Escribe o seguinte código dentro do <body> no arquivo HTML:

```
<div id="app">
  {{ mensaxe }}
</div>

<script>
  var app = new Vue({
    el: '#app',
    data: {
      mensaxe: 'Ola mundo!'
    }
  });
</script>
```

Neste código creamos unha instancia de Vue asignada ao id "app" e cun dato "mensaxe". Dentro do elemento co id "app" podemos utilizar as funcionalidades de Vue.js, como imprimir o dato.

O constructor

Toda aplicación de Vue.js comézase creando unha instancia da clase Vue da seguinte forma:

```
var app = new Vue({
  // Opcións
})
```

A continuación, imos explicar que se inclúe nesta clase cun exemplo:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <h1>Nome: {{ nome }}</h1>
      <h1>Apelido: {{ apelido }}</h1>
      <h1>{{ nomeCompleto() }}</h1>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          nome: "Xoán",
          apelido: "Vilas"
        },
        methods: {
          nomeCompleto: function() {
            return "Son " + this.nome + " " + this.apelido;
          }
        }
      });
    </script>

  </body>
</html>
```

O parámetro el é o id do elemento DOM. No exemplo anterior, temos o id "app". Este é o id do elemento que está no arquivo HTML.

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

Agora todo o que fagamos afectará a todos os elementos dentro do .

O seguinte que imos facer é emplear o obxecto "data" na vista. Este contén os valores de nome e apelido. No obtemos os datos da seguinte forma:

```
<div id="app">
  <h1>Nome: {{ nome }}</h1>
  <h1>Apelido: {{ apelido }}</h1>
</div>
```

Tamén podemos invocar métodos ou funcións na vista para imprimir, por exemplo, datos compostos. A función "nomeCompleto", dentro do obxecto "methods", permítenos imprimir o nome completo no :

```
<h1>{{ nomeCompleto() }}</h1>
```

Binding (asociar)

Imos ver como manipular ou asignar valores a atributos HTML, cambiar o estilo ou asignar clases coa directiva "v-bind". Vexamos un exemplo:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>

    <style>
      .fondo {
        background: red;
      }
    </style>

  </head>
  <body>

    <div id="app">
      <div v-bind:class="{ fondo: activo }">
        <b>{{ titulo }}</b>
      </div>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          titulo: "Asignación de clase",
          activo: true
        }
      });
    </script>

  </body>
</html>
```

Podemos observar un que contén a directiva "v-bind:class", á que se lle asigna un obxecto de JavaScript cunha propiedade "fondo" co valor da variable "activo" definida na instancia de Vue, que pode ser verdadeiro ou falso. Hai unha clase CSS "fondo" definida na etiqueta <style>, que transforma o fondo en vermello. Se a variable "activo" é verdadeira, aplícase a clase "fondo" ao . En caso contrario, non.

Tamén podemos asociar estilos:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <div v-bind:style="{ color: cor, fontSize: tamanhoFonte }">
        <b>{{ titulo }}</b>
      </div>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          titulo: "Asignación de estilo",
          cor: 'red',
          tamanhoFonte: '30pt'
        }
      });
    </script>

  </body>
</html>
```

O exemplo anterior aplicará os parámetros de estilo definidos en "data" ao coa directiva "v-bind:style".

Podemos realizar o mesmo creando unha variable "obxectoEstilo" na que gardemos os valores e despois a asignemos ao da seguinte forma:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <div v-bind:style="obxectoEstilo">
        <b>{{ titulo }}</b>
      </div>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          titulo: "Asignación de estilo",
          obxectoEstilo: {
            cor: 'red',
            tamanhoFonte: '30pt'
          }
        }
      });
    </script>

  </body>
</html>
```

Ademais de clases e mais estilos, tamén podemos asociar campos de texto (<input>). Vexamos un exemplo:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <input v-model="nome" />
      <h3>O nome introducido é {{ nome }}</h3>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          nome: ''
        }
      });
    </script>

  </body>
</html>
```

Cando escribimos no <input>, a directiva "v-model" asígnalle o valor do que escribimos á variable "nome". Posteriormente, mostramos o valor desta variable.

Eventos

"v-on" é o atributo utilizado no DOM para escoitar eventos en Vue.js. Vexamos algúns exemplos:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <button v-on:click="sumar">
        Pulsado {{ suma }} vece(s)
      </button>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          suma: 0
        },
        methods: {
          sumar: function(event) {
            this.suma += 1;
          }
        }
      });
    </script>

  </body>
</html>
```

A directiva "v-on:click" permítenos que a función "sumar" se execute cada vez que se faga clic no botón.

Dita directiva tamén se pode escribir da seguinte forma:

```
<button @click="sumar">
  Pulsado {{ suma }} vece(s)
</button>
```

Modificadores de eventos

Vue.js ofrece diferentes modificadores para o atributo "v-on". Un deles é "once", que permite executar o evento unha soa vez:

```
<button v-on:click.once="sumar">
  Pulsado {{ suma }} vece(s)
</button>
```

Outro modificador que pode resultar útil é "prevent", que evita que se abra o enlace definido na etiqueta e no seu lugar executar o método definido.

```
<a href="https://gdgpontevedra.me" v-on:click.prevent="evento">
  Enlace
</a>
```

Modificadores de teclado

Vue.js permite engadir modificadores de teclado para determinar se se pulsou algunha tecla. Realízase do seguinte xeito:

```
<!-- Só invoca a función se o código da tecla é 13 (Enter) -->
<input v-on:keyup.13="enviar">
```

Para as teclas máis usadas existen aliases:

```
<input v-on:keyup.enter="enviar">

<!-- Tamén se pode escribir así -->
<input @keyup.enter="enviar">
```

Preme aquí para ler máis sobre os eventos que ofrece Vue.js

Rendering

A continuación veremos as diferentes formas de renderizar contido dinámicamente con Vue.js.

v-if

Como en calquera linguaxe de programación, en Vue.js pódense construir sentencias "if" para renderizar ou non contido dependendo dunha condición. Vexamos un exemplo:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <button @click="mostrarTexto">
        Alternar visible/oculto
      </button>
      <p v-if="mostrar">Texto</p>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          mostrar: true
        },
        methods: {
          mostrarTexto: function() {
            this.mostrar = !this.mostrar;
          }
        }
      });
    </script>

  </body>
</html>
```

Se a variable "mostrar" é verdadeira, o <p> se mostra. Pulsando o botón, pode alternarse entre visible e oculto.

v-else

Tamén se pode mostrar contido se unha condición previa non se cumpre:

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <button @click="mostrarTexto">
        Alternar visible/oculto
      </button>
      <p v-if="mostrar">Texto mostrado</p>
      <p v-else>Texto oculto</p>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          mostrar: true
        },
        methods: {
          mostrarTexto: function() {
            this.mostrar = !this.mostrar;
          }
        }
      });
    </script>

  </body>
</html>
```

Se a condición "mostrar" non é verdadeira, entón móstrase o <p> co texto "Texto oculto".

v-show

A directiva "v-show" funciona da mesma forma de "v-if", só que a primeira non borra o elemento do DOM e a segunda si.

```
<p v-show="mostrar">Texto mostrado</p>
```

v-for

Esta directiva permite recorrer e renderizar os elementos dunha lista.

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <input @keyup.enter="engadirItem">
      <ul>
        <li v-for="item in items">{{ item }}</li>
      </ul>
    </div>

    <script>
      var app = new Vue({
        el: '#app',
        data: {
          items: []
        },
        methods: {
          engadirItem: function(event) {
            this.items.push(event.target.value);
          }
        }
      });
    </script>

  </body>
</html>
```

Neste exemplo se introduce texto nun <input> e se garda na lista "items" cando se pulsa a tecla "Enter" mediante o método "engadirItem", que colle o valor do obxecto sobre o que se realizou o evento (o <input>). Para recorrer os elementos da lista, utilízase a directiva "v-for" da seguinte forma:

```
<ul>
  <li v-for="item in items">{{ item }}</li>
</ul>
```

Compoñentes

No último apartado do tutorial veremos os compoñentes, unha das características máis importantes de Vue.js. Explicado de forma sinxela, os compoñentes permiten crear etiquetas HTML propias. A súa estructura en código é similar á dunha instancia Vue como as anteriores.

A continuación móstrase un exemplo de compoñente, <nome-completo>:

```
Vue.component('nome-completo', {
    template: '\
      <div v-on:click = "cambiarApelidos()">\
        {{ nome }} {{ primeiroApelido }} {{ segundoApelido }}\
      </div>\
    ',
    data: function() {
        return {
            nome: "Xoán",
            primeiroApelido: "Vilas",
            segundoApelido: "Vázquez"
        }
    },
    methods: {
        cambiarApelidos: function() {
            primeiroApelidoAnterior = this.primeiroApelido
            this.primeiroApelido = this.segundoApelido
            this.segundoApelido = primeiroApelidoAnterior
        }
    }
});
```

Dentro do compoñente aprécianse tres propiedades:

- template: código HTML que se insertará onde se utilice a etiqueta. Neste caso se crea un onde se imprimen as variables correspondentes ao nome dunha persoa. Ao facer clic no se executará un método do compoñente, que veremos máis abaixo.
- data: é como o obxecto "data" da instancia principal de Vue.js, pero neste caso non é un obxecto senón unha función. Isto é porque poden existir varias instancias do mesmo compoñente, e se non o facemos desta forma o obxecto "data" compartiríase por todas elas.
- methods: o mesmo que a instancia principal, inclúe funcións para executar sobre o compoñente.

Para utilizar este compoñente, basta con escribir a etiqueta no arquivo HTML.

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <nome-completo />
    </div>

    <script>
      Vue.component('nome-completo', {
        // ...
      });

      var app = new Vue({
        el: '#app'
      });
    </script>

  </body>
</html>
```

Nótese que o compoñente está definido antes da instancia principal "app". Se non o facemos así, a instancia non recoñecerá a etiqueta <nome-completo>.

Props

Agora mesmo o compoñente <nome-completo> utiliza datos estáticos, que xa están definidos. Se queremos que use datos proveñentes do exterior, podemos especificar "props". Isto faise da seguinte forma:

```
Vue.component('nome-completo', {
    template: '\
      <div>\
        {{ nome }} {{ primeiroApelido }} {{ segundoApelido }}\
      </div>\
    ',
    props: [ 'nome', 'primeiroApelido', 'segundoApelido' ]
});
```

Definimos os props "nome", "primeiroApelido" e "segundoApelido". Para asignarlles valores a estos props, se fai como nas etiquetas HTML normales:

```
<div id="app">
  <nome-completo
    nome="Xoán"
    primeiro-apelido="Vilas"
    segundo-apelido="Vázquez"
  />
</div>
```

Os nomes dos atributos da etiqueta non coinciden cos nomes dos props: isto é así por un requerimento de Vue.js. Se os nomes dos props están en "camelCase" (a primeira palabra empeza por minúscula e as outras por maiúscula), os nomes dos atributos da etiqueta deben estar en "kebab-case" (palabras separadas por guións).

Tamén podemos introducir como props valores da instancia principal de Vue. Véxase a seguinte instancia:

```
var app = new Vue({
  el: '#app',
  data: {
    items: [
      { id: 0, nome: "Xoán" },
      { id: 1, nome: "Sofía" },
      { id: 2, nome: "Rosa" },
    ]
  }
});
```

A propiedade "items" é unha lista con tres obxectos, cada un deles cun id e un nome. Imos emplear o compoñente <nome-completo> para cada un dos tres obxectos, utilizando as directivas "v-for" e "v-bind".

```
<html>
  <head>
    <title>Exemplo de Vue.js</title>
  </head>
  <body>

    <div id="app">
      <nome-completo
        v-for="item in items"
        v-bind:key="item.id"
        v-bind:nome="item.nome"
        primeiro-apelido="Vilas"
        segundo-apelido="Vázquez"
      />
    </div>

    <script>
      Vue.component('nome-completo', {
        template: '\
          <div>\
            {{ nome }} {{ primeiroApelido }} {{ segundoApelido }}\
          </div>\
        ',
        props: [ 'nome', 'primeiroApelido', 'segundoApelido' ]
      });

      var app = new Vue({
        el: '#app',
        data: {
          items: [
            { id: 0, nome: "Xoán" },
            { id: 1, nome: "Sofía" },
            { id: 2, nome: "Rosa" },
          ]
        }
      });
    </script>

  </body>
</html>
```

Imos ver o código resaltado da páxina anterior liña a liña:

- Creamos unha etiqueta <nome-completo>.
- Empleamos a directiva "v-for" para repetir a etiqueta <nome-completo> tantas veces como ítems haxa: a cada etiqueta lle corresponde un ítem. Os ítems se corresponden co obxecto "items" definido na instancia principal de Vue.
- Un requerimento de Vue.js nas etiquetas onde se emplea a directiva "v-for" é especificar un prop "key". O obxecto "items" é unha lista de obxectos coas propiedades "id" e "nome": pasaremos como prop "key" a propiedade "id". Para pasar dito prop debemos usar a directiva "v-bind": isto é porque o valor especificado entre comillas é un obxecto de JavaScript (neste caso, a propiedade "id" do obxecto "item" actual dentro do bucle).
- O prop "nome" pásase da mesma forma que o prop "key", coa directiva "v-bind" porque o valor é un obxecto de JavaScript (a propiedade "nome" do obxecto "item" actual).
- Respecto aos props de primeiro e segundo apelido, ao ser valores estáticos e non obxectos de JavaScript, non temos que usar a directiva "v-bind".

O resultado do código é o seguinte:

- Xoán Vilas Vázquez
- Sofía Vilas Vázquez
- Rosa Vilas Vázquez

Ata aquí todo o que trataremos neste tutorial sobre os compoñentes de Vue.js. Preme aquí para ler máis sobre esta potente funcionalidade.

Proxecto

Parabéns, completaches o tutorial! Diríxete a gdgpontevedra.me para demostrar os teus novos coñecementos de Vue.js mediante un divertido proxecto.