EBook Gratis

APRENDIZAJE Vue.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#vue.js

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: vue-js

It is an unofficial and free Vue.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Vue.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con Vue.js

Observaciones

`Vue.js` es un marco de front-end en rápido crecimiento para `JavaScript` , inspirado en `Angular.js` , `Reactive.js` y `Rivets.js` que ofrece un diseño simplificado de interfaz de usuario, manipulación y profunda reactividad.

Se describe como un marco modelado `MVVM` , `Model-View View-Model` , que se basa en el concepto de datos de *enlace bidireccional* a componentes y vistas. Es increíblemente rápido, supera las velocidades de otros marcos `JS` primer nivel y es muy fácil de usar para una fácil integración y creación de prototipos.

Versiones

Versión	Fecha de lanzamiento
2.4.1	2017-07-13
2.3.4	2017-06-08
2.3.3	2017-05-09
2.2.6	2017-03-26
2.0.0	2016-10-02
1.0.26	2016-06-28
1.0.0	2015-10-26
0.12.0	2015-06-12
0.11.0	2014-11-06

Examples

"¡Hola Mundo!" Programa

Para comenzar a usar Vue.js , asegúrese de tener el archivo de script incluido en su HTML. Por ejemplo, agregue lo siguiente a su HTML.

```
<script src="https://npmcdn.com/vue/dist/vue.js"></script>
```

Ejemplo simple

Plantilla HTML

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

JavaScript

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Vea una demostración en vivo de este ejemplo.

También puede consultar el ejemplo de "Hello World" creado por Vue.js.

Hola Mundo en Vue 2 (a la manera JSX)

JSX no debe interpretarse por el navegador. Primero debe ser transpilado a Javascript estándar. Para usar JSX necesitas instalar el plugin para babel `babel-plugin-transform-vue-JSX`

Ejecute el comando a continuación:

```
npm install babel-plugin-syntax-jsx babel-plugin-transform-vue-jsx babel-helper-vue-jsx-merge-
props --save-dev
```

y `.babelrc` a tu `.babelrc` así:

```
{
  "presets": ["es2015"],
  "plugins": ["transform-vue-jsx"]
}
```

Código de muestra con VUE JSX:

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  methods: {
    handleClick () {
      alert('Hello!')
```

```
    }
  },
  render (h) {
    return (
      <div>
        <h1 on-click={this.handleClick}>Hello from JSX</h1>
        <p> Hello World </p>
      </div>
    )
  }
})
```

Al utilizar JSX, puede escribir estructuras HTML / XML concisas en el mismo archivo que escribe el código JavaScript.

Felicitaciones, has terminado :)

Manejo de entrada de usuario

VueJS también se puede usar para manejar fácilmente la entrada del usuario, y el enlace de dos vías con v-model hace que sea muy fácil cambiar los datos fácilmente.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<div id="app">
    {{message}}
<input v-model="message">
</div>
```

JS:

```
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
```

Es muy fácil realizar un enlace bidireccional en VueJS utilizando la directiva `v-model` .

Echa un vistazo a un ejemplo en vivo aquí.

Lea Empezando con Vue.js en línea: https://riptutorial.com/es/vue-js/topic/1057/empezando-con-vue-js

Capítulo 2: Accesorios

Observaciones

camelCase <=> kebab-case

Al definir los nombres de sus `props` , recuerde siempre que los nombres de atributos HTML no distinguen entre mayúsculas y minúsculas. Eso significa que si define un `prop` en caso de camello en la definición de su componente ...

```
Vue.component('child', {
    props: ['myProp'],
    ...
});
```

... debes llamarlo en tu componente HTML como my-prop.

Examples

Pasando datos de padres a hijos con accesorios

En Vue.js, cada instancia de componente tiene ***su propio alcance aislado*** , lo que significa que si un componente principal tiene un componente secundario, el componente secundario tiene su propio alcance aislado y el componente primario tiene su propio alcance aislado.

Para cualquier aplicación de tamaño mediano a grande, seguir las convenciones de mejores prácticas previene muchos dolores de cabeza durante la fase de desarrollo y luego después del mantenimiento. Una de esas cosas a seguir es que ***evite hacer referencia / mutar datos primarios directamente desde el componente secundario*** . Entonces, ¿cómo hacemos referencia a los datos principales desde un componente secundario?

Cualquier información de los padres que se requiera en un componente secundario se debe pasar al niño como `props` de los padres.

Caso de uso : supongamos que tenemos una base de datos de usuarios con dos tablas, `users` y `addresses` con los siguientes campos:
tabla de `users`

nombre	teléfono	correo electrónico
John Mclane	(1) 234 5678 9012	john@dirhard.com
James Bond	(44) 777 0007 0077	bond@mi6.com

tabla de `addresses`

bloquear	calle	ciudad
Torres Nakatomi	Broadway	Nueva York
Casa mi6	Buckingham Road	Londres

y queremos tener tres componentes para mostrar la información del usuario correspondiente en cualquier lugar de nuestra aplicación

user-component.js

```
export default{
    template:`<div class="user-component">
                <label for="name" class="form-control">Name: </label>
                <input class="form-control input-sm" name="name" v-model="name">
                <contact-details :phone="phone" :email="email"></contact-details>
              </div>`,
    data(){
        return{
            name:'',
            phone:'',
            email:''
        }
    },
}
```

contact-details.js

```
import Address from './address';
export default{
    template:`<div class="contact-details-component>
                <h4>Contact Details:</h4>
                <label for="phone" class="form-control">Phone: </label>
                <input class="form-control input-sm" name="phone" v-model="phone">
                <label for="email" class="form-control">Email: </label>
                <input class="form-control input-sm" name="email" v-model="email">

                <h4>Address:</h4>
                <address :address-type="addressType"></address>
                //see camelCase vs kebab-case explanation below
            </div>`,
    props:['phone', 'email'],
    data:(){
        return:{
            addressType:'Office'
        }
    },
    components:{Address}
}
```

dirección.js

```
export default{
    template:`<div class="address-component">
                <h6>{{addressType}}</h6>
                <label for="block" class="form-control">Block: </label>
```

```
                <input class="form-control input-sm" name="block" v-model="block">
                <label for="street" class="form-control">Street: </label>
                <input class="form-control input-sm" name="street" v-model="street">
                <label for="city" class="form-control">City: </label>
                <input class="form-control input-sm" name="city" v-model="city">
            </div>`,
    props:{
        addressType:{
            required:true,
            type:String,
            default:'Office'
        },
    data(){
        return{
            block:'',
            street:'',
            city:''
        }
    }
}
```

main.js

```
import Vue from 'vue';

Vue.component('user-component', require'./user-component');
Vue.component('contact-details', require'./contact-details');

new Vue({
    el:'body'
});
```

index.html

```
...
<body>
    <user-component></user-component>
        ...
</body>
```

Estamos mostrando los datos del `phone` y del `email` , que son propiedades del `user-component` de `user-component` en `contact-details` que no tienen datos del teléfono o correo electrónico.

Pasando datos como props

Entonces, dentro de `user-component.js` en la propiedad de la ***plantilla*** , donde incluimos el componente `<contact-details>` , pasamos el ***teléfono*** y los datos de ***correo electrónico*** de `<user-component>` (componente primario) a `<contact-details>` (componente secundario) vinculándolo dinámicamente a los **accesorios** - `:phone="phone"` y `:email="email` que es igual que `v-bind:phone="phone"` y `v-bind:email="email"`

Puntales - Enlace dinámico

Dado que estamos vinculando dinámicamente las propuestas, cualquier cambio en el ***teléfono*** o ***correo electrónico*** dentro del componente principal, es decir, `<user-component>` se reflejará

inmediatamente en el componente secundario, es decir, `<contact-details>` .

Accesorios - como literales

Sin embargo, si hubiéramos pasado los valores de ***teléfono*** y ***correo electrónico*** como valores literales de cadena como `phone="(44) 777 0007 0077" email="bond@mi6.com"` entonces no reflejaría ningún cambio de datos que ocurra en el padre. componente.

Encuadernación unidireccional

De forma predeterminada, la dirección de los cambios es de arriba a abajo, es decir, cualquier cambio en los accesorios vinculados dinámicamente en el componente principal se propagará al componente secundario, pero cualquier cambio en los valores de prop en un componente secundario no se propagará al principal.

Por ejemplo: si dentro de `<contact-details>` cambiamos el correo electrónico de `bond@mi6.com` a `jamesbond@mi6.com` , los datos de los padres, es decir, la propiedad de los datos del teléfono en `<user-component>` seguirán teniendo un valor de `bond@mi6.com` .

Sin embargo, si cambiamos el valor del correo electrónico de `bond@mi6.com` a `jamesbond@mi6.co` en el componente principal (`<user-component>` en nuestro caso de uso), entonces el valor del correo electrónico en el componente secundario (`<contact-details>` en nuestro caso de uso) cambiará a `jamesbond@mi6.com` automáticamente - el cambio en el padre se propagará instantáneamente al niño.

Enlace de dos vías

Si queremos un enlace bidireccional, debemos especificar explícitamente el enlace bidireccional como `:email.sync="email"` lugar de `:email="email"` . Ahora, si cambiamos el valor de prop en el componente secundario, el cambio también se reflejará en el componente principal.

En una aplicación de tamaño mediano a grande, cambiar el estado padre del estado hijo será muy difícil de detectar y controlar, especialmente durante la depuración: **tenga cuidado** .

No habrá ninguna opción .sync disponible en Vue.js 2.0. **El enlace bidireccional para los accesorios está en desuso en Vue.js 2.0** .

Vinculante de una sola vez

También es posible definir **un** enlace explícito de **una sola vez** como `:email.once="email` , es más o menos similar a pasar un literal, porque cualquier cambio posterior en el valor de la propiedad principal no se propagará al elemento secundario.

ADVERTENCIA
Cuando se pasa un **objeto o una matriz** como prop, se pasarán **SIEMPRE POR REFERENCIA** , lo que significa que independientemente del tipo de enlace definido explícitamente `:email.sync="email"` o `:email="email"` o `:email.once="email"` , Si el correo electrónico es un objeto o una matriz en el principal, independientemente del tipo de enlace, cualquier cambio en el valor de prop dentro del componente secundario también afectará el valor en el primario.

Apoyos como Array

En el archivo `contact-details.js` hemos definido `props:['phone', 'email']` como una matriz, lo cual está bien si no queremos un control detallado con props.

Apoyos como objeto

Si queremos un control más preciso sobre los accesorios, como

- Si queremos definir qué tipo de valores son aceptables como prop.
- ¿Cuál debería ser un valor por defecto para el prop
- si un valor DEBE ser (obligatorio) para ser aprobado para la propuesta o es opcional

entonces necesitamos usar la notación de objetos para definir los accesorios, como hemos hecho en `address.js` .

Si estamos creando componentes reutilizables que pueden ser utilizados por otros desarrolladores en el equipo, entonces es una buena práctica definir objetos como objetos para que cualquiera que use el componente tenga una idea clara de cuál debería ser el tipo de datos y si Es obligatorio u opcional.

También se conoce como ***validación de accesorios*** . El **tipo** puede ser cualquiera de los siguientes constructores nativos:

- Cuerda
- Número
- Booleano
- Formación
- Objeto
- Función
- o un constructor personalizado

Algunos ejemplos de validación de propiedades como tomados de http://vuejs.org/guide/components.html#Props

```
Vue.component('example', {
   props: {
       // basic type check (`null` means accept any type)
       propA: Number,
       // multiple possible types (1.0.21+)
       propM: [String, Number],
       // a required string
       propB: {
         type: String,
         required: true
       },
       // a number with default value
       propC: {
          type: Number,
          default: 100
       },
       // object/array defaults should be returned from a
       // factory function
```

```
       propD: {
          type: Object,
          default: function () {
             return { msg: 'hello' }
         }
       },
       // indicate this prop expects a two-way binding. will
       // raise a warning if binding type does not match.
       propE: {
          twoWay: true
       },
       // custom validator function
       propF: {
          validator: function (value) {
             return value > 10
          }
       },
       // coerce function (new in 1.0.12)
       // cast the value before setting it on the component
       propG: {
          coerce: function (val) {
            return val + '' // cast the value to string
          }
       },
       propH: {
          coerce: function (val) {
            return JSON.parse(val) // cast the value to Object
          }
       }
    }
});
```

camelCase vs kebab-case

Atributos HTML son entre mayúsculas y minúsculas, lo que significa que no puede diferenciar entre `addresstype` y `addressType` , por lo que cuando se utiliza nombres camelCase prop como atributos que necesitamos para utilizar su kebab de los casos equivalentes (guión) delimitado por:

`addressType` debe escribirse como `address-type` en el atributo HTML.

Apoyos dinámicos

Al igual que puede vincular los datos de una vista al modelo, también puede vincular accesorios utilizando la misma directiva v-bind para pasar información de los componentes principales a los secundarios.

JS

```
new Vue({
    el: '#example',
    data: {
        msg: 'hello world'
    }
});
```

```
Vue.component('child', {
    props: ['myMessage'],
    template: '<span>{{ myMessage }}</span>
});
```

HTML

```
<div id="example">
    <input v-model="msg" />
    <child v-bind:my-message="msg"></child>
    <!-- Shorthand ... <child :my-message="msg"></child> -->
</div>
```

Resultado

```
hello world
```

Pasando apoyos mientras usas Vue JSX

Tenemos un componente principal: importando un componente secundario en él pasaremos las propuestas a través de un atributo. Aquí el atributo es 'src' y también estamos pasando el 'src'.

ParentComponent.js

```
import ChildComponent from './ChildComponent';
export default {
    render(h, {props}) {
        const src = 'https://cdn-images-1.medium.com/max/800/1*AxRXW2j8qmGJixIYg7n6uw.jpeg';
        return (
           <ChildComponent src={src} />
        );
    }
};
```

Y un componente infantil, donde tenemos que pasar props. Necesitamos especificar qué accesorios estamos pasando.

ChildComponent.js:

```
export default {
    props: ['src'],
    render(h, {props}) {
        return (
            <a href = {props.src} download = "myimage" >
                Click this link
            </a>
        );
```

```
    }
};
```

Lea Accesorios en línea: https://riptutorial.com/es/vue-js/topic/3080/accesorios

Capítulo 3: Bus de eventos

Introducción

Los buses de eventos son una forma útil de comunicación entre componentes que no están directamente relacionados, es decir, no tienen una relación padre-hijo.

Es solo una instancia de `vue` vacía, que se puede usar para `$emit` eventos `$on` o escuchar `$on` dichos eventos.

Sintaxis

1. exportar por defecto nuevo Vue ()

Observaciones

Use vuex si su aplicación tiene muchos componentes que requieren los datos de cada uno.

Examples

eventoBus

```
// setup an event bus, do it in a separate js file
var bus = new Vue()

// imagine a component where you require to pass on a data property
// or a computed property or a method!

Vue.component('card', {
  template: `<div class='card'>
    Name:
      <div class='margin-5'>
          <input v-model='name'>
    </div>
    <div class='margin-5'>
      <button @click='submit'>Save</button>
    </div>
  </div>`,
  data() {
    return {
      name: null
    }
  },
  methods: {
    submit() {
      bus.$emit('name-set', this.name)
    }
  }
})
```

```
// In another component that requires the emitted data.
var data = {
  message: 'Hello Vue.js!'
}

var demo = new Vue({
  el: '#demo',
  data: data,
  created() {
    console.log(bus)
    bus.$on('name-set', (name) => {
      this.message = name
    })
  }
})
```

Lea Bus de eventos en línea: https://riptutorial.com/es/vue-js/topic/9498/bus-de-eventos

Capítulo 4: Complementos

Introducción

Los complementos de Vue agregan funcionalidad global como, métodos globales, directivas, transiciones, filtros, métodos de instancia, objetos e inyectan algunas opciones de componentes utilizando mixins

Sintaxis

- MyPlugin.install = function (Vue, opciones) {}

Parámetros

Nombre	Descripción
Vue	Constructor Vue, inyectado por Vue.
opciones	Opciones adicionales si es necesario

Observaciones

En la mayoría de los casos, deberá indicar explícitamente a Vue que use un complemento

```
// calls `MyPlugin.install(Vue)`
Vue.use(MyPlugin)
```

Pasar opciones

```
Vue.use(MyPlugin, { someOption: true })
```

Examples

Registrador simple

```
//myLogger.js
export default {

  install(Vue, options) {
     function log(type, title, text) {
       console.log(`[${type}] ${title} - ${text}`);
     }

     Vue.prototype.$log = {
```

```
      error(title, text) { log('danger', title, text) },
      success(title, text) { log('success', title, text) },
      log
    }
  }
}
```

Antes de tu instancia principal de Vue, dile que registre tu plugin

```
//main.js
import Logger from './path/to/myLogger';

Vue.use(Logger);

var vm = new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

Ahora puede llamar a `this.$log` en cualquier componente secundario

```
//myComponent.vue
export default {
  data() {
    return {};
  },
  methods: {
    Save() {
      this.$log.success('Transaction saved!');
    }
  }
}
```

Lea Complementos en línea: https://riptutorial.com/es/vue-js/topic/8726/complementos

Capítulo 5: Componentes

Observaciones

En Componente (s):

props es una matriz de literales de cadena o referencias de objetos que se utilizan para pasar datos del componente principal. También puede estar en forma de objeto cuando se desea tener un control más preciso, como la especificación de valores predeterminados, el tipo de datos aceptados, si es necesario u opcional.

los datos tienen que ser una función que devuelve un objeto en lugar de un objeto plano. Esto es así porque requerimos que cada instancia del componente tenga sus propios datos para fines de reutilización.

eventos es un objeto que contiene escuchas para eventos a los que el componente puede responder por cambio de comportamiento

Métodos objeto que contiene funciones que definen el comportamiento asociado con el componente.

las propiedades **computadas** son como observadores u observables, cada vez que cualquier dependencia cambia, las propiedades se recalculan automáticamente y los cambios se reflejan en DOM al instante si DOM usa alguna propiedad computada

listo es un gancho de ciclo de vida de una instancia de Vue

Examples

Componente con ámbito (no global)

Manifestación

HTML

```
<script type="x-template" id="form-template">
    <label>{{inputLabel}} :</label>
  <input type="text" v-model="name" />
</script>

<div id="app">
  <h2>{{appName}}</h2>
  <form-component title="This is a form" v-bind:name="userName"></form-component>
</div>
```

JS

```
// Describe the form component
// Note: props is an array of attribute your component can take in entry.
// Note: With props you can pass static data('title') or dynamic data('userName').
// Note: When modifying 'name' property, you won't modify the parent variable, it is only
descendent.
// Note: On a component, 'data' has to be a function that returns the data.
var formComponent = {
  template: '#form-template',
  props: ['title', 'name'],
  data: function() {
    return {
      inputLabel: 'Name'
    }
  }
};

// This vue has a private component, it is only available on its scope.
// Note: It would work the same if the vue was a component.
// Note: You can build a tree of components, but you have to start the root with a 'new
Vue()'.
var vue = new Vue({
  el: '#app',
  data: {
    appName: 'Component Demo',
    userName: 'John Doe'
  },
  components: {
    'form-component': formComponent
  }
});
```

¿Qué son los componentes y cómo definir los componentes?

Los componentes en Vue son como widgets. Nos permiten escribir elementos personalizados reutilizables con el comportamiento deseado.

No son más que objetos que pueden contener cualquiera o todas las opciones que puede contener la raíz o cualquier instancia de Vue, incluida una plantilla HTML para representar.

Los componentes consisten en:

- Marca HTML: la plantilla del componente
- Estilos CSS: cómo se mostrará el marcado HTML
- Código JavaScript: los datos y el comportamiento.

Estos pueden escribirse en un archivo separado o como un solo archivo con la extensión `.vue` . A continuación hay ejemplos que muestran ambas formas:

.VUE - como un solo archivo para el componente

```
<style>
    .hello-world-compoment{
        color:#eeeeee;
        background-color:#555555;
    }
</style>
```

```
<template>
    <div class="hello-world-component">
        <p>{{message}}</p>
        <input @keyup.enter="changeName($event)"/>
    </div>
</template>

<script>
    export default{
        props:[ /* to pass any data from the parent here... */ ],
        events:{ /* event listeners go here */},
        ready(){
            this.name= "John";
        },
        data(){
           return{
              name:''
           }
        },
        computed:{
            message(){
                return "Hello from " + this.name;
            }
        },
        methods:{
            // this could be easily achieved by using v-model on the <input> field, but just
to show a method doing it this way.
            changeName(e){
                this.name = e.target.value;
            }
        }
    }
</script>
```

Archivos separados

hello-world.js - el archivo JS para el objeto componente

```
export default{
    template:require('./hello-world.template.html'),
    props:[ /* to pass any data from the parent here... */ ],
    events:{ /* event listeners go here */ },
    ready(){
        this.name="John";
    },
    data(){
        return{
            name:''
        }
    },
    computed:{
        message(){
            return "Hello World! from " + this.name;
        }
    },
    methods:{
        changeName(e){
            let name = e.target.value;
            this.name = name;
```

```
        }
    }
}
```

hello-world.template.html

```
<div class="hello-world-component">
    <p>{{message}}</p>
    <input class="form-control input-sm" @keyup.enter="changeName($event)">
</div>
```

hola-mundo.css

```
 .hello-world-compoment{
    color:#eeeeee;
    background-color:#555555;
}
```

Estos ejemplos utilizan la sintaxis de es2015, por lo que se necesitará a Babel para compilarlos en es5 para navegadores más antiguos.
Babel junto con **Browserify + vueify** o **Webpack + vue-loader** deberán compilar `hello-world.vue` .

Ahora que tenemos el componente `hello-world` definido, debemos registrarlo con Vue.

Esto se puede hacer de dos formas:

Registrarse como un componente global
En el archivo `main.js` (punto de entrada a la aplicación) podemos registrar cualquier componente globalmente con `Vue.component` :

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
Vue.component('hello-world', require('./hello-world');  // global registeration

new Vue({
    el:'body',

    // Templates can be defined as inline strings, like so:
    template:'<div class="app-container"><hello-world></hello-world></div>'
});
```

O regístrelo localmente dentro de un componente principal o componente raíz

```
import Vue from 'vue'; // Note that 'vue' in this case is a Node module installed with 'npm
install Vue'
import HelloWorld from './hello-world.js';

new Vue({
    el:'body',
    template:'<div class="app-container"><hello-world></hello-world></div>",

    components:{HelloWorld}  // local registeration
```

```
});
```

Los componentes globales se pueden utilizar en cualquier lugar dentro de la aplicación Vue.

Los componentes locales solo están disponibles para su uso en el componente principal con el que están registrados.

Componente del fragmento
Puede obtener un error de consola que le dice que no puede hacer algo porque el suyo es un *componente de fragmento* . Para resolver este tipo de problema, simplemente ajuste su plantilla de componente dentro de una sola etiqueta, como un `<div>` .

Registro local de componentes.

Un componente se puede registrar global o localmente (enlazar a otro componente específico).

```
var Child = Vue.extend({
    // ...
})

var Parent = Vue.extend({
    template: '...',
    components: {
         'my-component': Child
    }
})
```

El nuevo componente () solo estará disponible dentro del alcance (plantilla) del componente principal.

Registro en línea

Puede ampliar y registrar un componente en un solo paso:

```
Vue.component('custom-component', {
    template: '<div>A custom component!</div>'
})
```

También cuando el componente está registrado localmente:

```
var Parent = Vue.extend({
    components: {
        'custom-component': {
            template: '<div>A custom component!</div>'
        }
    }
})
```

Registro de datos en componentes.

Pasar un objeto a la propiedad de `data` al registrar un componente causaría que todas las

instancias del componente apunten a los mismos datos. Para resolver esto, necesitamos devolver `data` de una función.

```
var CustomComponent = Vue.extend({
    data: function () {
        return { a: 1 }
    }
})
```

Eventos

Una de las formas en que los componentes pueden comunicarse con sus ancestros / descendientes es a través de eventos de comunicación personalizados. Todas las instancias de Vue también son emisores e implementan una interfaz de eventos personalizada que facilita la comunicación dentro de un árbol de componentes. Podemos utilizar lo siguiente:

- `$on` : escucha los eventos emitidos por estos componentes ancestros o descendientes.
- `$broadcast` : emite un evento que se propaga hacia abajo a todos los descendientes.
- `$dispatch` : emite un evento que se activa primero en el componente y se propaga hacia arriba a todos los antepasados.
- `$emit` : activa un evento en uno mismo.

Por ejemplo, queremos ocultar un componente de botón específico dentro de un componente de formulario cuando se envía el formulario. En el elemento padre:

```
var FormComponent = Vue.extend({
  // ...
  components: {
    ButtonComponent
  },
  methods: {
    onSubmit () {
        this.$broadcast('submit-form')
    }
  }
})
```

En el elemento hijo:

```
var FormComponent = Vue.extend({
  // ...
  events: {
    'submit-form': function () {
        console.log('I should be hiding');
    }
  }
})
```

Algunas cosas para tener en mente:

- Cada vez que un evento encuentra un componente que lo escucha y se activa, se detendrá la propagación a menos que la función de devolución de llamada en este componente

devuelva `true` .

- `$dispatch()` siempre activa primero el componente que lo ha emitido.
- Podemos pasar cualquier número de argumentos al controlador de eventos. Haciendo `this.$broadcast('submit-form', this.formData, this.formStatus)` nos permite acceder a estos argumentos como `'submit-form': function (formData, formStatus) {}`

Lea Componentes en línea: https://riptutorial.com/es/vue-js/topic/1775/componentes

Capítulo 6: Componentes dinámicos

Observaciones

`<component>` es un elemento de componente reservado, no se confunda con la instancia de componentes.

`v-bind` es una directiva. Las directivas tienen el prefijo *v-* para indicar que son atributos especiales proporcionados por Vue.

Examples

Ejemplo de componentes dinámicos simples

Cambie dinámicamente entre varios componentes utilizando el elemento `<component>` y pase los datos a *v-bind: es* atributo:

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home'
  },
  components: {
    home: {
      template: "<p>Home</p>"
    },
    about: {
      template: "<p>About</p>"
    },
    contact: {
      template: "<p>Contact</p>"
    }
  }
})
```

HTML:

```
<div id="app">
   <component v-bind:is="currentPage">
       <!-- component changes when currentPage changes! -->
       <!-- output: Home -->
   </component>
</div>
```

Retazo:

Demo en vivo

Páginas de navegación con keep-alive

A veces desea mantener los componentes apagados en la memoria, para que esto ocurra, debe usar el elemento `<keep-alive>` :

Javascript:

```
new Vue({
  el: '#app',
  data: {
    currentPage: 'home',
  },
  methods: {
     switchTo: function(page) {
            this.currentPage = page;
     }
  },
  components: {
    home: {
      template: `<div>
                 <h2>Home</h2>
                 <p>{{ homeData }}</p>
                 </div>`,
      data: function() {
         return {
            homeData: 'My about data'
         }
       }
     },
    about: {
      template: `<div>
                 <h2>About</h2>
                 <p>{{ aboutData }}</p>
                 </div>`,
      data: function() {
         return {
            aboutData: 'My about data'
         }
      }
    },
    contact: {
      template: `<div>
             <h2>Contact</h2>
             <form method="POST" @submit.prevent>
             <label>Your Name:</label>
             <input type="text" v-model="contactData.name" >
             <label>You message: </label>
             <textarea v-model="contactData.message"></textarea>
             <button type="submit">Send</button>
             </form>
             </div>`,
       data: function() {
          return {
            contactData: { name:'', message:'' }
         }
      }
```

```
    }
  }
})
```

HTML:

```
<div id="app">
  <div class="navigation">
    <ul>
      <li><a href="#home" @click="switchTo('home')">Home</a></li>
      <li><a href="#about" @click="switchTo('about')">About</a></li>
      <li><a href="#contact" @click="switchTo('contact')">Contact</a></li>
    </ul>
  </div>

  <div class="pages">
    <keep-alive>
      <component :is="currentPage"></component>
    </keep-alive>
  </div>
</div>
```

CSS:

```
.navigation {
  margin: 10px 0;
}

.navigation ul {
  margin: 0;
  padding: 0;
}

.navigation ul li {
  display: inline-block;
  margin-right: 20px;
}

input, label, button {
  display: block
}

input, textarea {
  margin-bottom: 10px;
}
```

Retazo:

Demo en vivo

Lea Componentes dinámicos en línea: https://riptutorial.com/es/vue-js/topic/7702/componentes-dinamicos

Capítulo 7: Componentes personalizados con v-modelo

Introducción

Muchas veces tenemos que crear algunos componentes que realizan algunas acciones / operaciones en los datos y requerimos eso en el componente principal. La mayoría de las veces, `vuex` sería una mejor solución, pero en los casos en que el comportamiento del componente secundario no tiene nada que ver con el estado de la aplicación, por ejemplo: un control deslizante de rango, selector de fecha / hora, lector de archivos

Tener tiendas individuales para cada componente cada vez que se usan se complica.

Observaciones

Para tener `v-model` en un componente, debe cumplir dos condiciones.

1. Debe tener un prop llamado 'valor'
2. Debe emitir un evento de `input` con el valor esperado por los componentes principales.

```
<component v-model='something'></component>
```

es solo azúcar sintáctica para

```
<component
    :value="something"
    @input="something = $event.target.value"
>
</component>
```

Examples

v-modelo en un componente contador

Aquí el `counter` es un componente secundario al que se accede mediante una `demo` que es un componente principal que usa `v-model` .

```
// child component
Vue.component('counter', {
  template: `<div><button @click='add'>+1</button>
  <button @click='sub'>-1</button>
  <div>this is inside the child component: {{ result }}</div></div>`,
  data () {
    return {
      result: 0
```

```
    }
  },
  props: ['value'],
  methods: {
    emitResult () {
      this.$emit('input', this.result)
    },
    add () {
      this.result += 1
      this.emitResult()
    },
    sub () {
      this.result -= 1
      this.emitResult()
    }
  }
})
```

Este componente hijo emitirá `result` cada vez que se llamen los métodos `sub()` o `add()` .

```
// parent component
new Vue({
  el: '#demo',
  data () {
    return {
      resultFromChild: null
    }
  }
})

// parent template
<div id='demo'>
  <counter v-model='resultFromChild'></counter>
  This is in parent component {{ resultFromChild }}
</div>
```

Dado que `v-model` está presente en el componente secundario, al mismo tiempo se envió un prop con `value` nombre, hay un evento de entrada en el `counter` que, a su vez, proporcionará el valor del componente secundario.

Lea Componentes personalizados con v-modelo en línea: https://riptutorial.com/es/vue-js/topic/9353/componentes-personalizados-con-v-modelo

Capítulo 8: Directivas personalizadas

Sintaxis

- `Vue.directive(id, definition);`
- `Vue.directive(id, update); //when you need only the update function.`

Parámetros

Parámetro	Detalles
`id`	Cadena: el ID de la directiva que se utilizará sin el prefijo `v-` . (Agregue el prefijo `v-` cuando lo use)
`definition`	Objeto: un objeto de definición puede proporcionar varias funciones de `bind` (todas opcionales): `bind` , `update` y `unbind`

Examples

Lo esencial

Además del conjunto predeterminado de directivas enviadas en el núcleo, Vue.js también le permite registrar directivas personalizadas. Las directivas personalizadas proporcionan un mecanismo para asignar cambios de datos a un comportamiento arbitrario de DOM.

Puede registrar una directiva personalizada global con el `Vue.directive(id, definition)` , pasando una id de directiva seguida de un objeto de definición. También puede registrar una directiva personalizada local incluyéndola en la opción de `directives` un componente.

Funciones de gancho

- **bind** : se llama solo una vez, cuando la directiva se enlaza por primera vez al elemento.
- **actualización** : se llama por primera vez inmediatamente después del `bind` con el valor inicial, y luego nuevamente cuando el valor del enlace cambia. El nuevo valor y el valor anterior se proporcionan como el argumento.
- **desvincular** : se llama solo una vez, cuando la directiva no está vinculada al elemento.

```
Vue.directive('my-directive', {
     bind: function () {
       // do preparation work
       // e.g. add event listeners or expensive stuff
       // that needs to be run only once
     },
     update: function (newValue, oldValue) {
       // do something based on the updated value
       // this will also be called for the initial value
     },
```

```
    unbind: function () {
      // do clean up work
      // e.g. remove event listeners added in bind()
    }
})
```

Una vez registrado, puede usarlo en plantillas Vue.js como esta (recuerde agregar el prefijo `v-`):

```
<div v-my-directive="someValue"></div>
```

Cuando solo necesita la función de `update` , puede pasar una sola función en lugar del objeto de definición:

```
Vue.directive('my-directive', function (value) {
  // this function will be used as update()
})
```

Propiedades de la instancia directiva

Todas las funciones de enganche se copiarán en el objeto real de la Directiva, que se puede acceder dentro de estas funciones como su `this` contexto. El objeto directivo expone algunas propiedades útiles:

- **el** : el elemento al que está dirigida la directiva.
- **vm** : el contexto ViewModel que posee esta directiva.
- **expresión** : la expresión del enlace, excluyendo argumentos y filtros.
- **arg** : el argumento, si está presente.
- **nombre** : el nombre de la directiva, sin el prefijo.
- **modificadores** : un objeto que contiene modificadores, si los hay.
- **descriptor** : un objeto que contiene el resultado del análisis de toda la directiva.
- **params** : un objeto que contiene atributos param. Explicado a continuación.

  Debe tratar todas estas propiedades como de solo lectura y nunca modificarlas. También puede adjuntar propiedades personalizadas al objeto directivo, pero tenga cuidado de no sobrescribir accidentalmente las existentes internas.

Un ejemplo de una directiva personalizada que usa algunas de estas propiedades:

HTML

```
<div id="demo" v-demo:hello.a.b="msg"></div>
```

JavaScript

```
Vue.directive('demo', {
  bind: function () {
    console.log('demo bound!')
  },
  update: function (value) {
    this.el.innerHTML =
```

```
      'name - '       + this.name + '<br>' +
      'expression - ' + this.expression + '<br>' +
      'argument - '   + this.arg + '<br>' +
      'modifiers - '  + JSON.stringify(this.modifiers) + '<br>' +
      'value - '      + value
  }
})
var demo = new Vue({
  el: '#demo',
  data: {
    msg: 'hello!'
  }
})
```

Resultado

```
name - demo
expression - msg
argument - hello
modifiers - {"b":true,"a":true}
value - hello!
```

Objeto Literal

Si su directiva necesita varios valores, también puede pasar un objeto literal de JavaScript. Recuerda, las directivas pueden tomar cualquier expresión válida de JavaScript:

HTML

```
<div v-demo="{ color: 'white', text: 'hello!' }"></div>
```

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value.color) // "white"
  console.log(value.text) // "hello!"
})
```

Modificador literal

Cuando se usa una directiva con el modificador literal, su valor de atributo se interpretará como una cadena simple y se pasará directamente al método de `update` . El método de `update` también se llamará una sola vez, porque una cadena simple no puede ser reactiva.

HTML

```
<div v-demo.literal="foo bar baz">
```

JavaScript

```
Vue.directive('demo', function (value) {
  console.log(value) // "foo bar baz"
```

```
})
```

Lea Directivas personalizadas en línea: https://riptutorial.com/es/vue-js/topic/2368/directivas-personalizadas

Capítulo 9: El enlace de datos

Examples

Texto

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

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

La etiqueta del bigote se reemplazará con el valor de la propiedad `msg` en el objeto de datos correspondiente. También se actualizará cada vez que cambie la propiedad `msg` del objeto de datos.

También puede realizar interpolaciones de una sola vez que no se actualizan en el cambio de datos:

```
<span>This will never change: {{* msg }}</span>
```

HTML sin procesar

Los bigotes dobles interpretan los datos como texto plano, no HTML. Para generar HTML real, deberá usar los bigotes triples:

```
<div>{{{ raw_html }}}</div>
```

Los contenidos se insertan como HTML simple: se ignoran los enlaces de datos. Si necesita reutilizar piezas de plantilla, debe usar parciales.

Atributos

Los bigotes también se pueden usar dentro de los atributos HTML:

```
<div id="item-{{ id }}"></div>
```

Tenga en cuenta que las interpolaciones de atributos no están permitidas en las directivas Vue.js y los atributos especiales. No te preocupes, Vue.js generará advertencias cuando los bigotes se usen en lugares equivocados.

Filtros

Vue.js le permite agregar "filtros" opcionales al final de una expresión, denotados por el símbolo "pipe":

```
{{ message | capitalize }}
```

Aquí estamos "canalizando" el valor de la expresión del `message` través del filtro de `capitalize` incorporado, que de hecho es solo una función de JavaScript que devuelve el valor capitalizado. Vue.js proporciona una serie de filtros integrados, y hablaremos sobre cómo escribir sus propios filtros más adelante.

Tenga en cuenta que la sintaxis de canalización no forma parte de la sintaxis de JavaScript, por lo tanto, no puede mezclar filtros dentro de las expresiones; solo puedes añadirlos al final de una expresión.

Los filtros pueden ser encadenados:

```
{{ message | filterA | filterB }}
```

Los filtros también pueden tomar argumentos:

```
{{ message | filterA 'arg1' arg2 }}
```

La función de filtro siempre recibe el valor de la expresión como primer argumento. Los argumentos entre comillas se interpretan como una cadena simple, mientras que los que no están entre comillas se evaluarán como expresiones. Aquí, la cadena simple `'arg1'` pasará al filtro como segundo argumento, y el valor de la expresión `arg2` se evaluará y pasará como el tercer argumento.

Lea El enlace de datos en línea: https://riptutorial.com/es/vue-js/topic/1213/el-enlace-de-datos

Capítulo 10: enrutador vue

Introducción

vue-router es la biblioteca de enrutamiento soportada oficialmente para vue.js.

Sintaxis

- `<router-link to="/path">Link Text</router-link> <!-- Creates a link to the route that matches the path -->`
- `<router-view></router-view> <!-- Outlet for the currently matched route. It's component will be rendered here. -->`

Examples

Enrutamiento básico

La forma más fácil de comenzar a utilizar vue-router es usar la versión provista a través de CDN.

HTML:

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>

<div id="router-example">
    <router-link to="/foo">Link to Foo route</router-link>
    <router-view></router-view>
</div>
```

JavaScript (ES2015):

```
const Foo = { template: <div>This is the component for the Foo route</div> }

const router = new VueRouter({
    routes: [
       { path: '/foo', component: Foo}
    ]
})

const routerExample = new Vue({
    router
}).$mount('#router-example')
```

Lea enrutador vue en línea: https://riptutorial.com/es/vue-js/topic/9654/enrutador-vue

Capítulo 11: Eventos

Examples

Sintaxis de eventos

Para enviar un evento: `vm.$emit('new-message');`

Para capturar un evento: `vm.$on('new-message');`

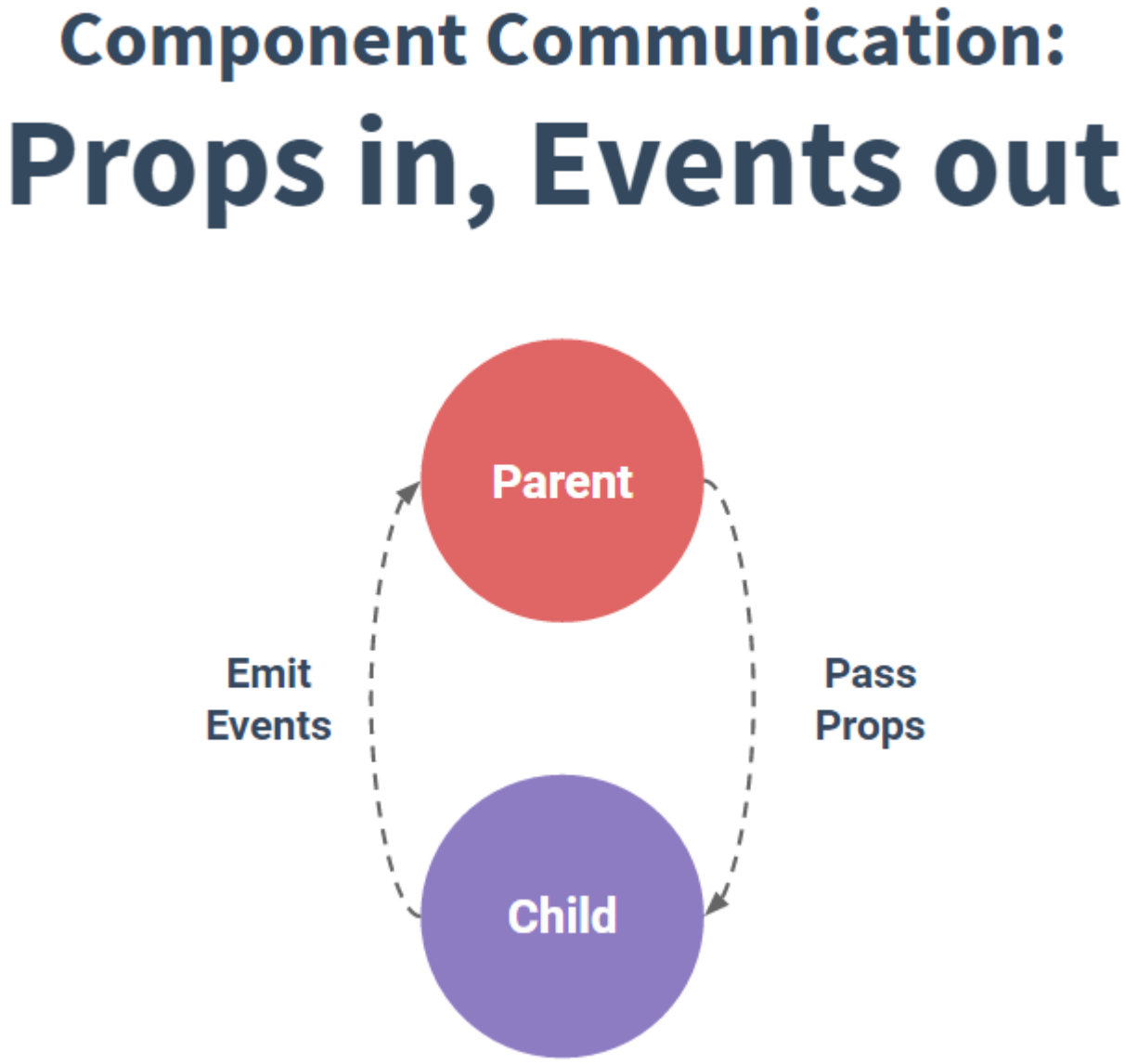
Para enviar un evento para todos los componentes **abajo:** `vm.$broadcast('new-message');`

Para enviar un evento para todos los componentes **arriba:** `vm.$dispatch('new-message');`

Nota: `$broadcast` y `$dispatch` están en desuso en Vue2. (ver características de Vue2)

¿Cuándo debo usar los eventos?

La siguiente imagen ilustra cómo debería funcionar la comunicación de componentes. La imagen proviene de las diapositivas de The Progressive Framework de Evan You (desarrollador de VueJS).



Aquí hay un ejemplo de cómo funciona:

MANIFESTACIÓN

HTML

```
<script type="x-template" id="message-box">
    <input type="text" v-model="msg" @keyup="$emit('new-message', msg)" />
</script>

<message-box :msg="message" @new-message="updateMessage"></message-box>
<div>You typed: {{message}}</div>
```

JS

```
var messageBox = {
  template: '#message-box',
  props: ['msg']
};

new Vue({
  el: 'body',
```

```
  data: {
    message: ''
  },
  methods: {
    updateMessage: function(msg) {
      this.message = msg;
    }
  },
  components: {
    'message-box': messageBox
  }
});
```

El ejemplo anterior se puede mejorar!

El ejemplo anterior muestra cómo funciona la comunicación de componentes. Pero en el caso de un componente de entrada personalizado, para sincronizar la variable principal con el valor escrito, deberíamos usar `v-model` .

DEMO vue1

DEMO vue2

En Vue1, debe usar `.sync` en la propiedad enviada al componente `<message-box>` . Esto le indica a VueJS que sincronice el valor en el componente secundario con el principal.

> **Recuerde:** Cada instancia de componente tiene su propio alcance aislado.

HTML Vue1

```
<script type="x-template" id="message-box">
  <input v-model="value" />
</script>

<div id="app">
  <message-box :value.sync="message"></message-box>
  <div>You typed: {{message}}</div>
</div>
```

En Vue2, hay un evento especial de `'input'` que puede `$emit` . El uso de este evento le permite poner un `v-model` directamente en el componente `<message-box>` . El ejemplo se verá como sigue:

HTML Vue2

```
<script type="x-template" id="message-box">
  <input :value="value" @input="$emit('input', $event.target.value)" />
</script>

<div id="app">
  <message-box v-model="message"></message-box>
  <div>You typed: {{message}}</div>
</div>
```

JS Vue 1 y 2

```
var messageBox = {
  template: '#message-box',
  props: ['value']
};

new Vue({
  el: '#app',
  data: {
    message: ''
  },
  components: {
    'message-box': messageBox
  }
});
```

Observe qué tan rápido se actualiza la entrada.

¿Cómo lidiar con la desaprobación de $ despacho y $ transmisión? (patrón de evento de bus)

Es posible que se haya dado cuenta de que `$emit` está dentro del alcance del componente que emite el evento. Eso es un problema cuando se quiere comunicar entre componentes alejados entre sí en el árbol de componentes.

Nota: En Vue1 puede usar `$dispatch` o `$broadcast` , pero no en Vue2. La razón es que no se escala bien. Hay un patrón de `bus` popular para gestionar esto:

MANIFESTACIÓN

HTML

```
<script type="x-template" id="sender">
  <button @click="bus.$emit('new-event')">Click me to send an event !</button>
</script>

<script type="x-template" id="receiver">
  <div>I received {{numberOfEvents}} event{{numberOfEvents == 1 ? '' : 's'}}</div>
</script>

<sender></sender>
<receiver></receiver>
```

JS

```
var bus = new Vue();

var senderComponent = {
  template: '#sender',
  data() {
    return {
      bus: bus
    }
```

```
  }
};

var receiverComponent = {
  template: '#receiver',
  data() {
    return {
      numberOfEvents: 0
    }
  },
  ready() {
    var self = this;

    bus.$on('new-event', function() {
      ++self.numberOfEvents;
    });
  }
};

new Vue({
  el: 'body',
  components: {
    'sender': senderComponent,
    'receiver': receiverComponent
  }
});
```

Solo debe comprender que cualquier instancia de `Vue()` puede `$emit` y capturar (`$on`) un evento. Simplemente declaramos un `bus` llamadas de instancia de `Vue` global y luego cualquier componente con esta variable puede emitir y capturar eventos desde él. Solo asegúrese de que el componente tenga acceso a la variable de `bus` .

Lea Eventos en línea: https://riptutorial.com/es/vue-js/topic/5941/eventos

Capítulo 12: Filtros personalizados

Sintaxis

- `Vue.filter(name, function(value){});` //BASIC
- `Vue.filter(name, function(value, begin, end){});` // Basic con valores de envoltura
- `Vue.filter(name, function(value, input){});` //Dinámica
- `Vue.filter(name, { read: function(value){}, write: function(value){} });` //De doble sentido

Parámetros

Parámetro	Detalles
nombre	Cadena - nombre deseado del filtro
valor	[Devolución de llamada] Cualquiera - valor de los datos que pasan al filtro
empezar	[Devolución de llamada] Cualquier valor por venir antes de los datos pasados
fin	[Devolución de llamada] Cualquier valor después de los datos pasados
entrada	[Devolución de llamada] Cualquiera: entrada del usuario vinculada a la instancia de Vue para obtener resultados dinámicos

Examples

Filtros de dos vías

Con un `two-way filter` , podemos asignar una operación de `read` *y* `write` para un solo `filter` que cambia el valor de los *mismos* datos entre la `view` y el `model` .

```
//JS
Vue.filter('uppercase', {
    //read : model -> view
    read: function(value) {
        return value.toUpperCase();
    },

    //write : view -> model
    write: function(value) {
        return value.toLowerCase();
    }
});

/*
 * Base value of data: 'example string'
 *
 * In the view : 'EXAMPLE STRING'
```

```
 * In the model : 'example string'
 */
```

BASIC

Los filtros personalizados en `Vue.js` se pueden crear fácilmente en una sola función de llamada a `Vue.filter` .

```
//JS
Vue.filter('reverse', function(value) {
    return value.split('').reverse().join('');
});

//HTML
<span>{{ msg | reverse }}</span> //'This is fun!' => '!nuf si sihT'
```

Es una buena práctica almacenar todos los filtros personalizados en archivos separados, por ejemplo, en `./filters` ya que es fácil reutilizar su código en su próxima aplicación. Si vas de esta manera tienes que **reemplazar la parte JS** :

```
//JS
Vue.filter('reverse', require('./filters/reverse'));
```

También puede definir sus propios envoltorios de `begin` y `end` .

```
//JS
Vue.filter('wrap', function(value, begin, end) {
    return begin + value + end;
});

//HTML
<span>{{ msg | wrap 'The' 'fox' }}</span> //'quick brown' => 'The quick brown fox'
```

Lea Filtros personalizados en línea: https://riptutorial.com/es/vue-js/topic/1878/filtros-personalizados

Capítulo 13: Ganchos de ciclo de vida

Examples

Ganchos para vue 1.x

- **`init`**

  Se llama de forma síncrona después de que la instancia se haya inicializado y antes de cualquier observación de datos inicial.

- **`created`**

  Llamado sincrónicamente después de que se crea la instancia. Esto ocurre antes de la configuración de `$el` , pero después de `data observation` , las `computed properties` , las `watch/event callbacks` y los `methods` se han configurado.

- **`beforeCompile`**

  Inmediatamente antes de la compilación de la instancia de Vue.

- **`compiled`**

  Inmediatamente después de completada la compilación. Todas las `directives` están vinculadas pero aún antes de que `$el` esté disponible.

- **`ready`**

  Ocurre después de que la compilación *y* `$el` se completan y la instancia se inyecta en el DOM por primera vez.

- **`attached`**

  Ocurre cuando `$el` se adjunta al DOM por una `directive` o instancia llama a `$appendTo()` .

- **`detached`**

  Se llama cuando `$el` se elimina / se separa del DOM o método de instancia.

- **`beforeDestroy`**

  Inmediatamente antes de que se destruya la instancia de Vue, todavía es completamente funcional.

- **`destroyed`**

  Llamado después de que se destruye una instancia. Todas las `bindings` y `directives` ya se

han liberado y las instancias secundarias también se han destruido.

Usando en una instancia

Dado que *todos los* ganchos de ciclo de vida en `Vue.js` son solo `functions` , puede colocar *cualquiera* de ellos directamente en la instancia declaraction.

```
//JS
new Vue({

    el: '#example',

    data: {
        ...
    },

    methods: {
        ...
    },

    //LIFECYCLE HOOK HANDLING
    created: function() {
        ...
    },

    ready: function() {
        ...
    }

});
```

Errores comunes: Acceso a DOM desde el enlace `ready ()`

Un caso de uso común para el enlace `ready()` es acceder al DOM, por ejemplo, para iniciar un complemento de Javascript, obtener las dimensiones de un elemento, etc.

El problema

Debido al mecanismo asíncrono de actualización de DOM de Vue, no se garantiza que el DOM se haya actualizado completamente cuando se llame al gancho `ready()` . Esto generalmente resulta en un error porque el elemento no está definido.

La solución

Para esta situación, el método de instancia `$nextTick()` puede ayudar. Este método difiere la ejecución de la función de devolución de llamada proporcionada hasta después del siguiente tick, lo que significa que se activa cuando se garantiza que todas las actualizaciones de DOM finalizarán.

Ejemplo:

```
module.exports {
  ready: function () {
```

```
    $('.cool-input').initiateCoolPlugin() //fails, because element is not in DOM yet.

    this.$nextTick(function() {
      $('.cool-input').initiateCoolPlugin() // this will work because it will be executed
after the DOM update.
    })
  }
}
```

Lea Ganchos de ciclo de vida en línea: https://riptutorial.com/es/vue-js/topic/1852/ganchos-de-ciclo-de-vida

Capítulo 14: Las advertencias de detección de cambio de matriz

Introducción

Cuando intenta establecer un valor de un elemento en un índice particular de una matriz inicializada en la opción de datos, vue no puede detectar el cambio y no desencadena una actualización al estado. Para superar esta advertencia, debe usar Vue. $ Set o usar el método Array.prototype.splice.

Examples

Usando Vue. $ Set

En su método o en cualquier gancho de ciclo de vida que cambie el elemento de la matriz en el índice particuar

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
            //this will not work
            //myArr[2] = 'strawberry';

            //Vue.$set(array, index, newValue)
            this.$set(this.myArr, 2, 'strawberry');
        }
    }
})
```

Aquí está el enlace al violín.

Usando Array.prototype.splice

Puede realizar el mismo cambio en lugar de usar `Vue.$set` utilizando el `splice()` del prototipo Array `splice()`

```
new Vue({
    el: '#app',
    data:{
        myArr : ['apple', 'orange', 'banana', 'grapes']
    },
    methods:{
        changeArrayItem: function(){
            //this will not work
            //myArr[2] = 'strawberry';
```

```
            //Array.splice(index, 1, newValue)
            this.myArr.splice(2, 1, 'strawberry');
        }
    }
})
```

Para matriz anidada

Si tiene una matriz anidada, se puede hacer lo siguiente

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            ['apple', 'banana'],
            ['grapes', 'orange']
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1], 1, 'strawberry');
        }
    }
})
```

Aquí está el enlace al jsfiddle

Array de objetos que contienen arrays

```
new Vue({
    el: '#app',
    data:{
        myArr : [
            {
                name: 'object-1',
                nestedArr: ['apple', 'banana']
            },
            {
                name: 'object-2',
                nestedArr: ['grapes', 'orange']
            }
        ]
    },
    methods:{
        changeArrayItem: function(){
            this.$set(this.myArr[1].nestedArr, 1, 'strawberry');
        }
    }
})
```

Aquí está el enlace al violín.

Lea Las advertencias de detección de cambio de matriz en línea: https://riptutorial.com/es/vue-js/topic/10679/las-advertencias-de-deteccion-de-cambio-de-matriz

Capítulo 15: Mixins

Examples

Mixin global

También puedes aplicar una mezcla globalmente. ¡Con precaución! Una vez que aplique una mezcla globalmente, afectará a **cada** instancia de Vue creada posteriormente. Cuando se usa correctamente, se puede usar para inyectar lógica de procesamiento para opciones personalizadas:

```
// inject a handler for `myOption` custom option
Vue.mixin({
  created: function () {
    var myOption = this.$options.myOption
    if (myOption) {
      console.log(myOption)
    }
  }
})

new Vue({
  myOption: 'hello!'
})
// -> "hello!"
```

> Utilice los mixins globales de forma dispersa y cuidadosa, ya que afecta a cada instancia de Vue creada, incluidos los componentes de terceros. En la mayoría de los casos, solo debe usarlo para el manejo de opciones personalizadas como se muestra en el ejemplo anterior.

Opción personalizada Combinar estrategias

Cuando las opciones personalizadas se combinan, utilizan la estrategia predeterminada, que simplemente sobrescribe el valor existente. Si desea que una opción personalizada se combine mediante la lógica personalizada, debe adjuntar una función a `Vue.config.optionMergeStrategies` :

```
Vue.config.optionMergeStrategies.myOption = function (toVal, fromVal) {
  // return mergedVal
}
```

Para la mayoría de las opciones basadas en objetos, simplemente puede usar la misma estrategia utilizada por los `methods` :

```
var strategies = Vue.config.optionMergeStrategies
strategies.myOption = strategies.methods
```

Lo esencial

Los mixins son una forma flexible de distribuir funcionalidades reutilizables para componentes de Vue. Un objeto mixin puede contener cualquier opción de componente. Cuando un componente usa una mezcla, todas las opciones en la mezcla se "mezclan" en las propias opciones del componente.

```
// define a mixin object
var myMixin = {
  created: function () {
    this.hello()
  },
  methods: {
    hello: function () {
      console.log('hello from mixin!')
    }
  }
}

// define a component that uses this mixin
var Component = Vue.extend({
  mixins: [myMixin]
})

var component = new Component() // -> "hello from mixin!"
```

Opción de fusión

Cuando un mixin y el componente en sí contienen opciones superpuestas, se "fusionan" utilizando las estrategias apropiadas. Por ejemplo, las funciones de enganche con el mismo nombre se combinan en una matriz para que todas ellas sean llamadas. Además, los ganchos de mezcla se llamarán **antes de** los ganchos propios del componente:

```
var mixin = {
  created: function () {
    console.log('mixin hook called')
  }
}

new Vue({
  mixins: [mixin],
  created: function () {
    console.log('component hook called')
  }
})

// -> "mixin hook called"
// -> "component hook called"
```

Las opciones que esperan valores de objetos, por ejemplo, `methods` , `components` y `directives` , se fusionarán en el mismo objeto. Las opciones del componente tendrán prioridad cuando haya claves en conflicto en estos objetos:

```
var mixin = {
  methods: {
    foo: function () {
```

```
      console.log('foo')
    },
    conflicting: function () {
      console.log('from mixin')
    }
  }
}

var vm = new Vue({
  mixins: [mixin],
  methods: {
    bar: function () {
      console.log('bar')
    },
    conflicting: function () {
      console.log('from self')
    }
  }
})

vm.foo() // -> "foo"
vm.bar() // -> "bar"
vm.conflicting() // -> "from self"
```

Tenga en cuenta que las mismas estrategias de combinación se utilizan en `Vue.extend()` .

Lea Mixins en línea: https://riptutorial.com/es/vue-js/topic/2562/mixins

Capítulo 16: Modificadores

Introducción

Hay algunas operaciones de uso frecuente como `event.preventDefault()` o `event.stopPropagation()` dentro de los controladores de eventos. Aunque podemos hacer esto fácilmente dentro de los métodos, sería mejor si los métodos fueran puramente lógicos de datos en lugar de tener que lidiar con los detalles del evento DOM.

Examples

Modificadores de eventos

Vue proporciona modificadores de eventos para `v-on` llamando a las directivas postfixes denotadas por un punto.

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

Por ejemplo:

```
<!-- the click event's propagation will be stopped -->
<a v-on:click.stop="doThis"></a>

<!-- the submit event will no longer reload the page -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- use capture mode when adding the event listener -->
<div v-on:click.capture="doThis">...</div>

<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div v-on:click.self="doThat">...</div>
```

Modificadores clave

Cuando escuchamos eventos de teclado, a menudo necesitamos verificar códigos de teclas comunes. Recordar todos los keyCodes es una molestia, por lo que Vue proporciona alias para las claves más utilizadas:

- `.enter`
- `.tab`
- `.delete` (captura las teclas "Eliminar" y "Retroceso")
- `.esc`
- `.space`
- `.up`

- `.down`
- `.left`
- `.right`

Por ejemplo:

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

Modificadores de entrada

- `.trim`

Si desea que las entradas del usuario se recorten automáticamente, puede agregar el modificador de `trim` a las entradas administradas de su `v-model` :

```
<input v-model.trim="msg">
```

- `.number`

Si desea que la entrada del usuario se escriba automáticamente como un número, puede hacer lo siguiente:

```
<input v-model.number="age" type="number">
```

- `.lazy`

En general, `v-model` sincroniza la entrada con los datos después de cada evento de entrada, pero puede agregar el modificador `lazy` para sincronizar en su lugar después de los eventos de cambio:

```
<input v-model.lazy="msg" >
```

Lea Modificadores en línea: https://riptutorial.com/es/vue-js/topic/8612/modificadores

Capítulo 17: Plantilla "webpack" de polyfill

Parámetros

Archivos o paquetes	Comando o configuración a modificar.
babel-polyfill	`npm i -save babel-polyfill`
karma.conf.js	`files: ['../../node_modules/babel-polyfill/dist/polyfill.js','./index.js'],`
webpack.base.conf.js	`app: ['babel-polyfill', './src/main.js']`

Observaciones

Las configuraciones descritas anteriormente, el ejemplo que usa una función no estandarizada funcionará en "internet explorer" y la `npm test` pasará.

Examples

Uso de funciones para polyfill (ex: find)

```
<template>
  <div class="hello">
    <p>{{ filtered() }}</p>
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      list: ['toto', 'titi', 'tata', 'tete']
    }
  },
  methods: {
    filtered () {
      return this.list.find((el) => el === 'tata')
    }
  }
}
</script>
```

Lea Plantilla "webpack" de polyfill en línea: https://riptutorial.com/es/vue-js/topic/9174/plantilla--webpack--de-polyfill

Capítulo 18: Propiedades calculadas

Observaciones

Datos vs propiedades calculadas

La principal diferencia de caso de uso para los `data` y las propiedades `computed` de una instancia de `Vue` depende del estado potencial o la probabilidad de cambio de los datos. Al decidir qué categoría debe ser un determinado objeto, estas preguntas pueden ayudar:

- ¿Es este un valor constante? (**datos**)
- ¿Tiene esto la posibilidad de cambiar? (**computado** o **datos**)
- ¿Su valor depende del valor de otros datos? (**calculado**)
- ¿Necesita datos o cálculos adicionales para completar antes de ser utilizado? (**calculado**)
- ¿El valor solo cambiará bajo ciertas circunstancias? (**datos**)

Examples

Ejemplo básico

Modelo

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

JavaScript

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    // a computed getter
    b: function () {
      // `this` points to the vm instance
      return this.a + 1
    }
  }
})
```

Resultado

```
a=1, b=2
```

Aquí hemos declarado una propiedad computada `b` . La función que proporcionamos se utilizará

como la función getter para la propiedad `vm.b` :

```
console.log(vm.b) // -> 2
vm.a = 2
console.log(vm.b) // -> 3
```

El valor de `vm.b` siempre depende del valor de `vm.a`

Puede enlazar datos a propiedades computadas en plantillas como una propiedad normal. Vue es consciente de que `vm.b` depende de `vm.a` , por lo que actualizará todos los enlaces que dependan de `vm.b` cuando `vm.a` cambie.

Propiedades calculadas vs reloj

modelo

```
<div id="demo">{{fullName}}</div>
```

ver ejemplo

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar',
    fullName: 'Foo Bar'
  }
})

vm.$watch('firstName', function (val) {
  this.fullName = val + ' ' + this.lastName
})

vm.$watch('lastName', function (val) {
  this.fullName = this.firstName + ' ' + val
})
```

Ejemplo calculado

```
var vm = new Vue({
  el: '#demo',
  data: {
    firstName: 'Foo',
    lastName: 'Bar'
  },
  computed: {
    fullName: function () {
      return this.firstName + ' ' + this.lastName
    }
  }
})
```

Setters computados

Las propiedades calculadas se volverán a calcular automáticamente cada vez que se modifique cualquier dato del que dependan los cálculos. Sin embargo, si necesita cambiar manualmente una propiedad computada, Vue le permite crear un método de establecimiento para hacer esto:

Plantilla (del ejemplo básico anterior):

```
<div id="example">
  a={{ a }}, b={{ b }}
</div>
```

Javascript:

```
var vm = new Vue({
  el: '#example',
  data: {
    a: 1
  },
  computed: {
    b: {
      // getter
      get: function () {
        return this.a + 1
      },
      // setter
      set: function (newValue) {
        this.a = newValue - 1
      }
    }
  }
```

Ahora puedes invocar el getter o el setter:

```
console.log(vm.b)       // -> 2
vm.b = 4                // (setter)
console.log(vm.b)       // -> 4
console.log(vm.a)       // -> 3
```

`vm.b = 4` invocará el setter, y configurará this.a en 3; por extensión, vm.b evaluará a 4.

Usando setters computados para v-model

Es posible que necesite un `v-model` en una propiedad computada. Normalmente, el modelo v no actualizará el valor de la propiedad computada.

La plantilla:

```
<div id="demo">
  <div class='inline-block card'>
    <div :class='{onlineMarker: true, online: status, offline: !status}'></div>
    <p class='user-state'>User is {{ (status) ? 'online' : 'offline' }}</p>
  </div>

  <div class='margin-5'>
```

```
    <input type='checkbox' v-model='status'>Toggle status (This will show you as offline to
others)
  </div>
</div>
```

Estilo:

```
#demo {
  font-family: Helvetica;
  font-size: 12px;
}
.inline-block > * {
  display: inline-block;
}
.card {
  background: #ddd;
  padding:2px 10px;
  border-radius: 3px;
}
.onlineMarker {
  width: 10px;
  height: 10px;
  border-radius: 50%;
  transition: all 0.5s ease-out;
}

.online {
  background-color: #3C3;
}

.offline {
  background-color: #aaa;
}

.user-state {
  text-transform: uppercase;
  letter-spacing: 1px;
}

.margin-5 {
  margin: 5px;
}
```

El componente:

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            }
        }
    }
})
```

Fiddle Aquí verás, al hacer clic en el botón de radio no tiene ningún uso, tu estado aún está en línea.

```
var demo = new Vue({
    el: '#demo',
    data: {
        statusProxy: null
    },
    computed: {
        status: {
            get () {
                return (this.statusProxy === null) ? true : this.statusProxy
            },
            set (val) {
                this.statusProxy = val
            }
        }
    }
})
```

violín Y ahora puede ver el cambio cuando la casilla de verificación está activada / desactivada.

Lea Propiedades calculadas en línea: https://riptutorial.com/es/vue-js/topic/2371/propiedades-calculadas

Capítulo 19: Ranuras

Observaciones

¡Importante! Las ranuras después del render no garantizan el orden para las posiciones de las ranuras. La ranura, que fue la primera, puede tener una posición diferente después del render.

Examples

Uso de ranuras individuales

Las ranuras únicas se utilizan cuando un componente secundario solo define una `slot` dentro de su plantilla. El componente de la `page` anterior utiliza una única ranura para distribuir contenido.

A continuación se muestra un ejemplo de la plantilla del componente de la `page` que utiliza una única ranura:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <slot>
            This will only be displayed if there is no content
            to be distributed.
        </slot>
    </body>
</html>
```

Para ilustrar cómo funciona la ranura, podemos configurar una página de la siguiente manera.

```
<page>
    <p>This content will be displayed within the page component</p>
</page>
```

El resultado final será:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <p>This content will be displayed within the page component</p>
    </body>
</html>
```

Si no colocáramos nada entre las etiquetas de `page` su lugar teníamos `<page></page>` , obtendríamos el siguiente resultado, ya que hay un contenido predeterminado entre las etiquetas de la `slot` en la plantilla del componente de la `page` .

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        This will only be displayed if there is no content
        to be distributed.
    </body>
</html>
```

¿Qué son las tragamonedas?

Las ranuras ofrecen una forma conveniente de distribuir contenido de un componente principal a un componente secundario. Este contenido puede ser cualquier cosa, desde texto, HTML o incluso otros componentes.

A veces puede ser útil pensar en las ranuras como un medio para inyectar contenido directamente en la plantilla de un componente secundario.

Las ranuras son especialmente útiles cuando la composición del componente debajo del componente principal no es siempre la misma.

Tomemos el siguiente ejemplo donde tenemos un componente de `page` . El contenido de la página podría cambiar en función de si esa página muestra, por ejemplo, un artículo, una publicación de blog o un formulario.

Artículo

```
<page>
    <article></article>
    <comments></comments>
</page>
```

Entrada en el blog

```
<page>
    <blog-post></blog-post>
    <comments></comments>
</page>
```

Formar

```
<page>
    <form></form>
</page>
```

Observe cómo el contenido del componente de la `page` puede cambiar. Si no usáramos las ranuras, esto sería más difícil ya que la parte interna de la plantilla se arreglaría.

Recuerde: *"Todo en la plantilla principal se compila en el ámbito primario; todo en la plantilla secundaria se compila en el ámbito secundario".*

Usando ranuras nombradas

Las ranuras con nombre funcionan de manera similar a las ranuras individuales, pero en su lugar le permiten distribuir contenido a diferentes regiones dentro de su plantilla de componente secundaria.

Tome el componente de la `page` del ejemplo anterior, pero modifique su plantilla para que sea como sigue:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <slot name="sidebar"></slot>
        </aside>
        <main>
            <slot name="content"></slot>
        </main>
    </body>
</html>
```

Al usar el componente de la `page` , ahora podemos determinar dónde se coloca el contenido a través del atributo de `slot` :

```
<page>
    <p slot="sidebar">This is sidebar content.</p>
    <article slot="content"></article>
</page>
```

La página resultante será:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <aside>
            <p>This is sidebar content.</p>
        </aside>
        <main>
            <article></article>
        </main>
    </body>
</html>
```

Si se define una `slot` sin un atributo de `name` , cualquier contenido que se coloque dentro de etiquetas de componentes que no especifiquen un atributo de `slot` se colocará en esa ranura.

Vea el ejemplo de inserción múltiple en los documentos oficiales de Vue.js.

Usando Slots en Vue JSX con 'babel-plugin-transform-vue-jsx'

Si estás usando VueJS2 y te gusta usar JSX junto con él. En este caso, para usar la ranura, a continuación se muestra la solución con el ejemplo. Tenemos que usar `this.$slots.default` Es casi como `this.props.children` en React JS.

Componente.js:

```
export default {
    render(h) {  //eslint-disable-line
        return (
            <li>
                { this.$slots.default }
            </li>
        );
    }
};
```

ParentComponent.js

```
import Component from './Component';

export default {
    render(h) {  //eslint-disable-line
        return (
            <ul>
                <Component>
                    Hello World
                </Component>
            </ul>
        );
    }
};
```

Lea Ranuras en línea: https://riptutorial.com/es/vue-js/topic/4484/ranuras

Capítulo 20: Representación condicional

Sintaxis

- `<element v-if="condition"></element>` // v-if
- `<element v-if="condition"></element><element v-else="condition"></element>` // v-if | v-else
- `<template v-if="condition">...</template>` // v-if con plantilla
- `<element v-show="condition"></element>` // v-show

Observaciones

Es muy importante recordar la diferencia entre `v-if` y `v-show` . Si bien sus usos son casi idénticos, un elemento vinculado a `v-if` *solo se procesará en el DOM* cuando su condición sea `true` por **primera vez** . Cuando se usa la directiva `v-show` , *todos los elementos **se** representan en el DOM* pero se ocultan usando el estilo de `display` si la condición es `false` .

Examples

Visión general

En Vue.js, la representación condicional se logra mediante el uso de un conjunto de directivas en los elementos de la plantilla.

`v-if`

El elemento se muestra normalmente cuando la condición es `true` . Cuando la condición es `false` , solo se produce *una* compilación *parcial* y el elemento no se representa en el DOM hasta que la condición se vuelve `true` .

`v-else`

No acepta una condición, sino que procesa el elemento si la condición `v-if` del elemento anterior es `false` . Solo se puede utilizar después de un elemento con la directiva `v-if` .

`v-show`

Sin embargo, se comporta de manera similar a `v-if` , sin embargo, el elemento *siempre* se representará en el DOM, incluso cuando la condición sea `false` . Si la condición es `false` , esta directiva simplemente establecerá el estilo de `display` del elemento en `none` .

v-if / v-else

Asumiendo que tenemos una instancia de `Vue.js` definida como:

```
var vm = new Vue({
```

```
    el: '#example',
    data: {
        a: true,
        b: false
    }
});
```

Puede representar condicionalmente cualquier elemento html incluyendo la directiva v-if; el elemento que contiene v-if solo se procesará si la condición se evalúa como verdadera:

```
<!-- will render 'The condition is true' into the DOM -->
<div id="example">
    <h1 v-if="a">The condition is true</h1>
</div>
```

El elemento `<h1>` se representará en este caso, porque la variable 'a' es verdadera. v-if puede usarse con cualquier expresión, propiedad computada o función que se evalúe como un valor booleano:

```
<div v-if="0 === 1">                  false; won't render</div>
<div v-if="typeof(5) === 'number'">   true; will render</div>
```

Puede usar un elemento de `template` para agrupar varios elementos para una sola condición:

```
<!-- in this case, nothing will be rendered except for the containing 'div' -->
<div id="example">
    <template v-if="b">
        <h1>Heading</h1>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </template>
</div>
```

Al usar `v-if` , también tiene la opción de integrar una condición de contador con la directiva `v-else` . El contenido contenido dentro del elemento solo se mostrará si la condición del v-if anterior era falsa. Tenga en cuenta que esto significa que un elemento con v-else debe aparecer inmediatamente después de un elemento con v-if.

```
<!-- will render only 'ELSE' -->
<div id="example">
    <h1 v-if="b">IF</h1>
    <h1 v-else="a">ELSE</h1>
</div>
```

Al igual que con v-if, con v-else puede agrupar varios elementos html en un `<template>` :

```
<div v-if="'a' === 'b'"> This will never be rendered. </div>
<template v-else>
    <ul>
      <li> You can also use templates with v-else. </li>
      <li> All of the content within the template </li>
      <li> will be rendered. </li>
    </ul>
```

```
</template>
```

v-show

El uso de la directiva `v-show` es casi idéntico al de `v-if` . Las únicas diferencias son que `v-show` *no* admite la sintaxis `<template>` y no existe una condición "alternativa".

```
var vm = new Vue({
    el: '#example',
    data: {
        a: true
    }
});
```

El uso básico es el siguiente ...

```
<!-- will render 'Condition met' -->
<div id="example">
    <h1 v-show="a">Condition met</h1>
</div>
```

Si bien `v-show` no admite la directiva `v-else` para definir condiciones "alternativas", esto se puede lograr al anular la anterior ...

```
<!-- will render 'This is shown' -->
<div id="example">
    <h1 v-show="!a">This is hidden</h1>
    <h1 v-show="a">This is shown</h1>
</div>
```

Lea Representación condicional en línea: https://riptutorial.com/es/vue-js/topic/3465/representacion-condicional

Capítulo 21: Representación de lista

Examples

Uso básico

Se puede representar una lista usando la directiva `v-for` . La sintaxis requiere que especifique la matriz de origen para iterar, y un alias que se usará para hacer referencia a cada elemento en la iteración. En el siguiente ejemplo, usamos `items` como la matriz de origen, y `item` como el alias para cada elemento.

HTML

```
<div id="app">
  <h1>My List</h1>
  <table>
    <tr v-for="item in items">
      <td>{{item}}</td>
    </tr>
  </table>
</div>
```

Guión

```
new Vue({
  el: '#app',
  data: {
    items: ['item 1', 'item 2', 'item 3']
  }
})
```

Puedes ver una demostración de trabajo aquí .

Solo renderizar elementos HTML

En este ejemplo se renderizarán cinco etiquetas `<li>`

```
<ul id="render-sample">
  <li v-for="n in 5">
    Hello Loop
  </li>
</ul>
```

Lista de cuenta atrás de cerdo

```
<ul>
  <li v-for="n in 10">{{11 - n}} pigs are tanning at the beach. One got fried, and
```

```
</ul>
```

https://jsfiddle.net/gurghet/3jeyka22/

Iteración sobre un objeto

`v-for` puede utilizarse para iterar sobre un objeto claves (y valores):

HTML:

```
<div v-for="(value, key) in object">
  {{ key }} : {{ value }}
</div>
```

Guión:

```
new Vue({
  el: '#repeat-object',
  data: {
    object: {
      FirstName: 'John',
      LastName: 'Doe',
      Age: 30
    }
  }
})
```

Lea Representación de lista en línea: https://riptutorial.com/es/vue-js/topic/1972/representacion-de-lista

Capítulo 22: Usando "esto" en Vue

Introducción

Uno de los errores más comunes que encontramos en el código Vue en StackOverflow es el mal uso de `this` . Los errores más comunes generalmente se dividen en dos áreas, utilizando `this` en devoluciones de llamadas para promesas u otras funciones asíncronas y utilizando funciones de flecha para definir métodos, propiedades computadas, etc.

Examples

¡INCORRECTO! Usando "esto" en una devolución de llamada dentro de un método Vue.

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomethingAsynchronous(){
      setTimeout(function(){
        // This is wrong! Inside this function,
        // "this" refers to the window object.
        this.foo = "baz";
      }, 1000);
    }
  }
})
```

¡INCORRECTO! Usando "esto" dentro de una promesa.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Again, this is wrong! "this", here, refers to the window.
      this.people = data.results;
    })
  }
})
```

¡CORRECTO! Use un cierre para capturar "esto"

Puede capturar la correcta `this` usando un cierre .

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    // Before executing the web service call, save this to a local variable
    var self = this;
    $.getJSON("http://swapi.co/api/people/", function(data){
      // Inside this call back, because of the closure, self will
      // be accessible and refers to the Vue object.
      self.people = data.results;
    })
  }
})
```

¡CORRECTO! Utilice vinculación.

Puede enlazar la función de devolución de llamada.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted:function(){
    $.getJSON("http://swapi.co/api/people/", function(data){
      this.people = data.results;
    }.bind(this));
  }
})
```

¡CORRECTO! Usa una función de flecha.

```
new Vue({
  el:"#star-wars-people",
  data:{
    people: null
  },
  mounted: function(){
    $.getJSON("http://swapi.co/api/people/", data => this.people = data.results);
  }
})
```

¡Precaución! Las funciones de flecha son una sintaxis introducida en Ecmascript 2015. Todavía no es compatible con *todos* los navegadores modernos, así que solo úselo si está apuntando a un navegador que *sabe que lo* soporta, o si está compilando su javascript hasta la sintaxis de ES5 usando algo como babel .

¡INCORRECTO! Usando una función de flecha para definir un método que se refiere a "esto"

```
new Vue({
```

```
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    // This is wrong! Arrow functions capture "this" lexically
    // and "this" will refer to the window.
    doSomething: () => this.foo = "baz"
  }
})
```

¡CORRECTO! Definir métodos con la sintaxis de función típica.

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething: function(){
      this.foo = "baz"
    }
  }
})
```

Alternativamente, si está utilizando un compilador de javascript o un navegador que admita Ecmascript 2015

```
new Vue({
  el:"#app",
  data:{
    foo: "bar"
  },
  methods:{
    doSomething(){
      this.foo = "baz"
    }
  }
})
```

Lea Usando "esto" en Vue en línea: https://riptutorial.com/es/vue-js/topic/9350/usando--esto--en-vue

Capítulo 23: Vigilantes

Examples

Cómo funciona

Puedes ver la propiedad de datos de cualquier instancia de Vue. Al ver una propiedad, activa un método al cambiar:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' () {
            console.log('The watched property has changed')
        }
    }
}
```

Puede recuperar el valor antiguo y el nuevo:

```
export default {
    data () {
        return {
            watched: 'Hello World'
        }
    },
    watch: {
        'watched' (value, oldValue) {
            console.log(oldValue) // Hello World
            console.log(value) // ByeBye World
        }
    },
    mounted () {
        this.watched = 'ByeBye World'
    }
}
```

Si necesita ver propiedades anidadas en un objeto, deberá usar la propiedad `deep` :

```
export default {
    data () {
        return {
            someObject: {
                message: 'Hello World'
            }
        }
    },
    watch: {
        'someObject': {
```

```
            deep: true,
            handler (value, oldValue) {
                console.log('Something changed in someObject')
            }
        }
    }
}
```

¿Cuándo se actualizan los datos?

Si necesita activar el observador antes de realizar algunos cambios nuevos en un objeto, debe usar el método `nextTick()` :

```
export default {
    data() {
        return {
            foo: 'bar',
            message: 'from data'
        }
    },
    methods: {
        action () {
            this.foo = 'changed'
            // If you juste this.message = 'from method' here, the watcher is executed after.
            this.$nextTick(() => {
                this.message = 'from method'
            })
        }
    },
    watch: {
        foo () {
            this.message = 'from watcher'
        }
    }
}
```

Lea Vigilantes en línea: https://riptutorial.com/es/vue-js/topic/7988/vigilantes

Capítulo 24: Vue componentes de un solo archivo

Introducción

Describe cómo crear componentes de un solo archivo en un archivo .vue.

Especialmente las decisiones de diseño que se puedan tomar.

Examples

Ejemplo de archivo componente .vue

```
<template>
  <div class="nice">Component {{title}}</div>
</template>

<script>
export default {
    data() {
        return {
            title: "awesome!"
        };
    }
}
</script>

<style>
.nice {
    background-color: red;
    font-size: 48px;
}
</style>
```

Lea Vue componentes de un solo archivo en línea: https://riptutorial.com/es/vue-js/topic/10118/vue-componentes-de-un-solo-archivo

Capítulo 25: VueJS + Redux con Vua-Redux (la mejor solución)

Examples

Cómo utilizar Vua-Redux

Instalación de Vua Redux desde NPM:

Instalar a través de:

```
npm i vua-redux --save
```

Inicializar:

==========

// main.js

```
import Vue from 'vue';
import { reduxStorePlugin } from 'vua-redux';
import AppStore from './AppStore';
import App from './Component/App';

// install vua-redux
Vue.use(reduxStorePlugin);

new Vue({
    store: AppStore,
    render(h) {
        return <App />
    }
});
```

// AppStore.js

```
import { createStore } from 'redux';

const initialState = {
  todos: []
};

const reducer = (state = initialState, action) => {
  switch(action.type){
    case 'ADD_TODO':
      return {
        ...state,
        todos: [...state.todos, action.data.todo]
      }
```

```
    default:
      return state;
    }
}

const AppStore = createStore(reducer);

export default AppStore;
```

Use en su componente:

// componentes / App.js

```
import { connect } from 'vua-redux';

const App = {
    props: ['some-prop', 'another-prop'],

    /**
     * everything you do with vue component props
     * you can do inside collect key
     */
    collect: {
        todos: {
            type: Array,
        },
        addTodo: {
            type: Function,
        },
    },

    methods: {
        handleAddTodo() {
            const todo = this.$refs.input.value;
            this.addTodo(todo);
        }
    },

    render(h) {
        return <div>
            <ul>
                {this.todos.map(todo => <li>{todo}</li>)}
            </ul>

            <div>
                <input type="text" ref="input" />
                <button on-click={this.handleAddTodo}>add todo</button>
            </div>
        </div>
    }
};

function mapStateAsProps(state) {
    return {
        todos: state.todos
    };
}

function mapActionsAsProps(dispatch) {
    return {
```

```
        addTodo(todo) {
            dispatch({
                type: 'ADD_TODO',
                data: { todo }
            })
        }
    }
}

export default connect(mapStateAsProps, mapActionsAsProps)(App);
```

Lea VueJS + Redux con Vua-Redux (la mejor solución) en línea: https://riptutorial.com/es/vue-js/topic/7396/vuejs-plus-redux-con-vua-redux--la-mejor-solucion-

Capítulo 26: Vuex

Introducción

Vuex es una biblioteca de patrones de administración de estado para aplicaciones Vue.js. Sirve como un almacén centralizado para todos los componentes de una aplicación, con reglas que garantizan que el estado solo se puede mutar de manera predecible. También se integra con la extensión de herramientas de desarrollo oficial de Vue para proporcionar funciones avanzadas como la depuración de viajes en tiempo de configuración cero y la exportación / importación de instantáneas de estado.

Examples

¿Qué es Vuex?

Vuex es un complemento oficial para Vue.js que ofrece un almacén de datos centralizado para usar dentro de su aplicación. Está muy influido por la arquitectura de la aplicación Flux, que presenta un flujo de datos unidireccional que conduce a un diseño y un razonamiento más simples.

Dentro de una aplicación Vuex, el almacén de datos contiene todo el **estado de la aplicación compartida** . Este estado se altera por las **mutaciones** que se realizan en respuesta a una **acción que** invoca un evento de mutación a través del **despachador** .

Un ejemplo del flujo de datos en una aplicación Vuex se describe en el diagrama a continuación.

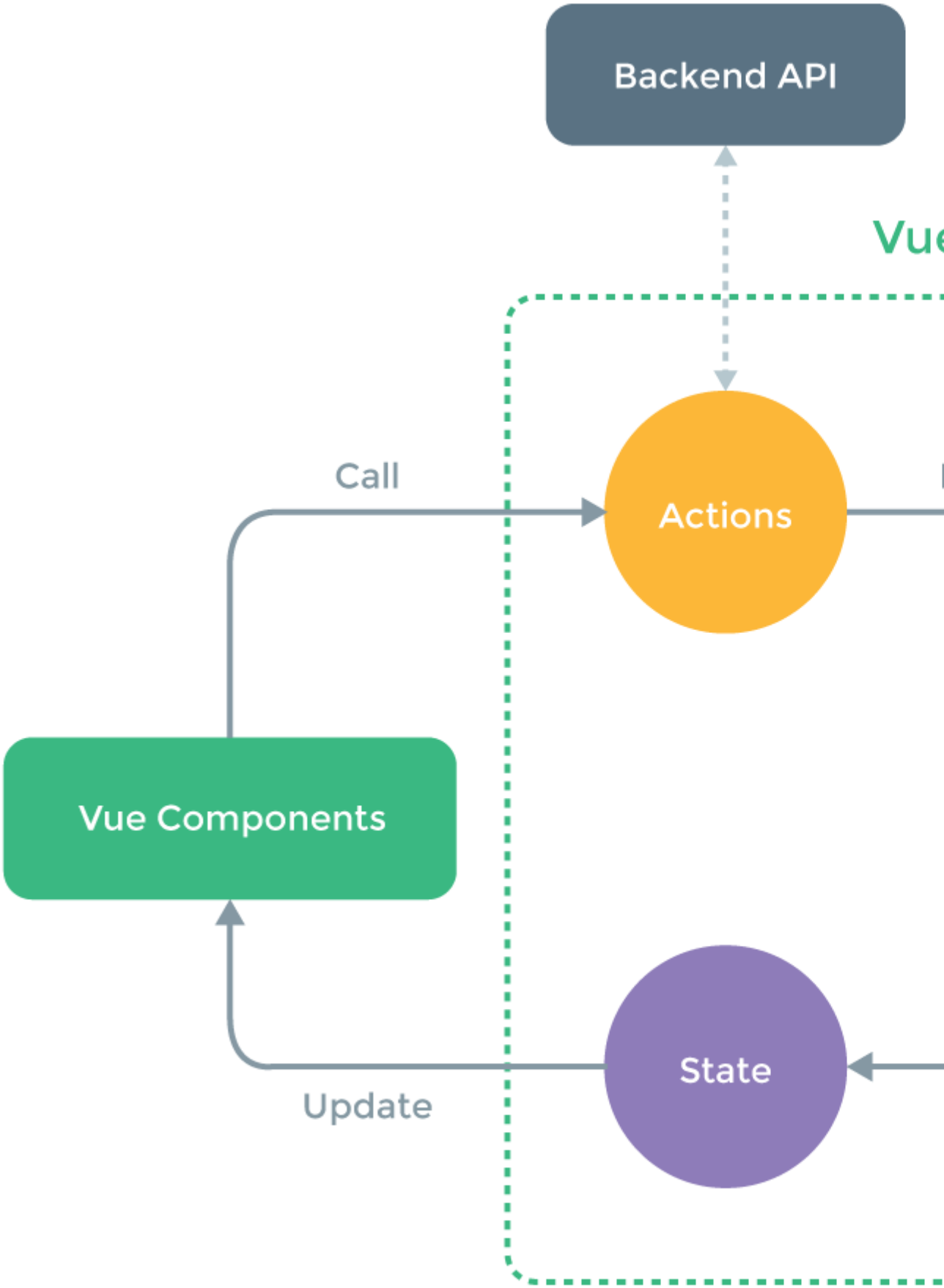

Backend API
Vue
Call
Actions
Vue Components
State
Update

Diagrama utilizado bajo la licencia MIT , originalmente del repositorio oficial de Vuex GitHub .

Los componentes individuales de la aplicación Vue.js pueden acceder al objeto de la tienda para recuperar datos a través de **captadores** , que son funciones puras que devuelven una copia de solo lectura de los datos deseados.

Los componentes pueden tener **acciones** que son funciones que realizan cambios en la propia copia de los datos del componente, luego usan el **despachador** para enviar un evento de mutación. Este evento es manejado por el almacén de datos que actualiza el estado según sea necesario.

Los cambios se reflejan automáticamente en toda la aplicación, ya que todos los componentes están vinculados de forma reactiva a la tienda a través de sus captadores.

Un ejemplo que ilustra el uso de vuex en un proyecto de vue.

```
const state = {
    lastClickTime: null
}

const mutations = {
    updateLastClickTime: (state, payload) => {
      state.lastClickTime = payload
  }
}

const getters = {
  getLastClickTime: state => {
    return new Date(state.lastClickTime)
  }
}

const actions = {
    syncUpdateTime: ({ commit }, payload) => {
    commit("updateLastClickTime", payload)
  },
  asyncUpdateTime: ({ commit }, payload) => {
    setTimeout(() => {
      commit("updateLastClickTime", payload)
    }, Math.random() * 5000)
  }
}

const store = new Vuex.Store({
  state,
  getters,
  mutations,
  actions
})

const { mapActions, mapGetters } = Vuex;

// Vue
const vm = new Vue({
    el: '#container',
  store,
```

```
  computed: {
      ...mapGetters([
        'getLastClickTime'
    ])
  },
  methods: {
      ...mapActions([
        'syncUpdateTime',
      'asyncUpdateTime'
    ]),
    updateTimeSyncTest () {
        this.syncUpdateTime(Date.now())
    },
    updateTimeAsyncTest () {
        this.asyncUpdateTime(Date.now())
    }
  }
})
```

Y la plantilla HTML para el mismo:

```
<div id="container">
  <p>{{ getLastClickTime || "No time selected yet" }}</p>
  <button @click="updateTimeSyncTest">Sync Action test</button>
  <button @click="updateTimeAsyncTest">Async Action test</button>
</div>
```

1. Aquí el **estado** contiene la propiedad **lastClickTime** inicializada como nula. Esta configuración de valores predeterminados es importante para mantener las propiedades **reactivas** . **Las propiedades no mencionadas en el estado** estarán disponibles, pero los cambios que se realicen posteriormente **no serán accesibles** mediante el uso de captadores.

2. El captador utilizado proporciona una propiedad calculada que se actualizará cada vez que una mutación actualice el valor de la propiedad estatal.

3. **Solo** se permite que las **mutaciones** cambien el estado y sus propiedades, dicho esto, solo lo hace de forma **síncrona** .

4. Se puede usar una Acción en caso de actualizaciones asíncronas, donde la llamada a la API (aquí simulada por el setTimeout temporizado al azar) se puede realizar en la acción, y después de obtener la respuesta se puede comprometer una mutación, para realizar el cambio al estado .

¿Por qué usar Vuex?

Al crear aplicaciones de gran tamaño, como las aplicaciones de una sola página (SPA), que suelen constar de muchos componentes reutilizables, se pueden volver difíciles de construir y mantener rápidamente. El intercambio de datos y el estado entre estos componentes también puede descomponerse rápidamente y ser difícil de depurar y mantener.

Al utilizar un almacén de datos de aplicaciones centralizado, el estado completo de la aplicación

se puede representar en un solo lugar, lo que hace que la aplicación esté más organizada. Mediante el uso de un flujo de datos unidireccional, las mutaciones y el acceso a los datos del componente de alcance solo a los datos requeridos, se vuelve mucho más sencillo razonar sobre el rol del componente y cómo debe afectar el estado de la aplicación.

Los componentes de VueJS son entidades separadas y no pueden compartir datos entre sí fácilmente. Para compartir datos sin vuex necesitamos `emit` evento con datos y luego escuchar y capturar ese evento con `on` .

componente 1

```
this.$emit('eventWithDataObject', dataObject)
```

componente 2

```
this.$on('eventWithDataObject', function (dataObject) {
    console.log(dataObject)
})
```

Con vuex instalado, simplemente podemos acceder a sus datos desde cualquier componente sin necesidad de escuchar los eventos.

```
this.$store.state.myData
```

También podemos cambiar los datos de forma sincronizada con *mutadores,* utilizar *las acciones* asíncronos y obtener datos con funciones *getter.*

Las funciones de Getter podrían funcionar como funciones globales computadas. Podemos acceder a ellos desde componentes:

```
this.$store.getters.myGetter
```

Las acciones son métodos globales. Podemos enviarlos desde componentes:

```
this.$store.dispatch('myAction', myDataObject)
```

Y las mutaciones son la única forma de cambiar los datos en vuex. Podemos confirmar los cambios:

```
this.$store.commit('myMutation', myDataObject)
```

El código de Vuex se vería así

```
state: {
  myData: {
    key: 'val'
  }
},
getters: {
```

```
  myGetter: state => {
     return state.myData.key.length
  }
},
actions: {
  myAction ({ commit }, myDataObject) {
    setTimeout(() => {
      commit('myMutation', myDataObject)
    }, 2000)
  }
},
mutations: {
  myMutation (state, myDataObject) {
    state.myData = myDataObject
  }
}
```

¿Cómo instalar Vuex?

La mayor parte del tiempo que utilizará Vuex estará en aplicaciones basadas en componentes más grandes donde probablemente esté utilizando un paquete de paquetes como Webpack o Browserify junto con Vueify si está usando archivos individuales.

En este caso, la forma más fácil de obtener Vuex es desde NPM. Ejecute el siguiente comando para instalar Vuex y guárdelo en las dependencias de su aplicación.

```
npm install --save vuex
```

Asegúrese de cargar el enlace de Vuex con su configuración de Vue colocando la siguiente línea después de su declaración de `require('vue')` .

```
Vue.use(require('vuex'))
```

Vuex también está disponible en CDN; Puede obtener la última versión de cdnjs aquí .

Notificaciones de despido automático

Este ejemplo registrará un módulo vuex dinámicamente para almacenar notificaciones personalizadas que se pueden descartar automáticamente

notificaciones.js

Resolver vuex store y definir algunas constantes.

```
//Vuex store previously configured on other side
import _store from 'path/to/store';

//Notification default duration in milliseconds
const defaultDuration = 8000;

//Valid mutation names
const NOTIFICATION_ADDED = 'NOTIFICATION_ADDED';
const NOTIFICATION_DISMISSED = 'NOTIFICATION_DISMISSED';
```

establecer el estado inicial de nuestro módulo

```
const state = {
  Notifications: []
}
```

establecer nuestros getters módulo

```
const getters = {
  //All notifications, we are returning only the raw notification objects
  Notifications: state => state.Notifications.map(n => n.Raw)
}
```

configura nuestro modulo de acciones

```
const actions = {
  //On actions we receive a context object which exposes the
  //same set of methods/properties on the store instance
  //{commit} is a shorthand for context.commit, this is an
  //ES2015 feature called argument destructuring
  Add({ commit }, notification) {
    //Get notification duration or use default duration
    let duration = notification.duration || defaultDuration

    //Create a timeout to dismiss notification
    var timeOut = setTimeout(function () {
      //On timeout mutate state to dismiss notification
      commit(NOTIFICATION_DISMISSED, notification);
    }, duration);

    //Mutate state to add new notification, we create a new object
    //for save original raw notification object and timeout reference
    commit(NOTIFICATION_ADDED, {
      Raw: notification,
      TimeOut: timeOut
    })
  },
  //Here we are using context object directly
  Dismiss(context, notification) {
    //Just pass payload
    context.commit(NOTIFICATION_DISMISSED, notification);
  }
}
```

establecer nuestras mutaciones de módulo

```
const mutations = {
  //On mutations we receive current state and a payload
  [NOTIFICATION_ADDED](state, notification) {
    state.Notifications.push(notification);
  },
  //remember, current state and payload
  [NOTIFICATION_DISMISSED](state, rawNotification) {
    var i = state.Notifications.map(n => n.Raw).indexOf(rawNotification);
    if (i == -1) {
      return;
    }
```

```
    clearTimeout(state.Notifications[i].TimeOut);
    state.Notifications.splice(i, 1);
  }
}
```

Registre nuestro módulo con estado definido, captadores, acciones y mutación.

```
_store.registerModule('notifications', {
  state,
  getters,
  actions,
  mutations
});
```

Uso

componenteA.vue

Este componente muestra todas las notificaciones como alertas de bootstrap en la esquina superior derecha de la pantalla, también permite descartar manualmente cada notificación.

```
<template>
<transition-group name="notification-list" tag="div" class="top-right">
  <div v-for="alert in alerts" v-bind:key="alert" class="notification alert alert-dismissible"
v-bind:class="'alert-'+alert.type">
    <button v-on:click="dismiss(alert)" type="button" class="close" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
    <div>
      <div>
        <strong>{{alert.title}}</strong>
      </div>
      <div>
        {{alert.text}}
      </div>
    </div>
  </div>
</transition-group>
</template>

<script>
export default {
  name: 'arc-notifications',
  computed: {
    alerts() {
      //Get all notifications from store
      return this.$store.getters.Notifications;
    }
  },
  methods: {
    //Manually dismiss a notification
    dismiss(alert) {
      this.$store.dispatch('Dismiss', alert);
    }
  }
}
</script>
<style lang="scss" scoped>
```

```
$margin: 15px;

.top-right {
    top: $margin;
    right: $margin;
    left: auto;
    width: 300px;
    //height: 600px;
    position: absolute;
    opacity: 0.95;
    z-index: 100;
    display: flex;
    flex-wrap: wrap;
    //background-color: red;
}
.notification {
    transition: all 0.8s;
    display: flex;
    width: 100%;
    position: relative;
    margin-bottom: 10px;
    .close {
        position: absolute;
        right: 10px;
        top: 5px;
    }

    > div {
        position: relative;
        display: inline;
    }
}
.notification:last-child {
    margin-bottom: 0;
}
.notification-list-enter,
.notification-list-leave-active {
    opacity: 0;
    transform: translateX(-90px);
}
.notification-list-leave-active {
    position: absolute;
}
</style>
```

Fragmento para agregar notificación en cualquier otro componente

```
//payload could be anything, this example content matches with componentA.vue
this.$store.dispatch('Add', {
  title = 'Hello',
  text = 'World',
  type = 'info',
  duration = 15000
});
```

Lea Vuex en línea: https://riptutorial.com/es/vue-js/topic/3430/vuex

Creditos

S. No	Capítulos	Contributors
1	Empezando con Vue.js	Community, Erick Petrucelli, ironcladgeek, J. Bruni, James, Lambda Ninja, m_callens, MotKohn, rap-2-h, Ru Chern Chong, Sankalp Singha, Shog9, Shuvo Habib, user1012181, user6939352, Yerko Palma
2	Accesorios	asemahle, Donkarnash, FlatLander, m_callens, rap-2-h, Shuvo Habib
3	Bus de eventos	Amresh Venugopal
4	Complementos	AldoRomo88
5	Componentes	Donkarnash, Elfayer, Hector Lorenzo, Jeff, m_callens, phaberest, RedRiderX, user6939352
6	Componentes dinámicos	Med, Ru Chern Chong
7	Componentes personalizados con v-modelo	Amresh Venugopal
8	Directivas personalizadas	Mat J, Ogie Sado
9	El enlace de datos	gurghet, Jilson Thomas
10	enrutador vue	AJ Gregory
11	Eventos	Elfayer
12	Filtros personalizados	Finrod, M U, m_callens
13	Ganchos de ciclo de vida	Linus Borg, m_callens, PatrickSteele, xtreak
14	Las advertencias de detección de cambio de matriz	Vamsi Krishna
15	Mixins	Ogie Sado

16	Modificadores	sept08
17	Plantilla "webpack" de polyfill	Stefano Nepa
18	Propiedades calculadas	Amresh Venugopal, cl3m, jaredsk, m_callens, Theo, Yerko Palma
19	Ranuras	Daniel Waghorn, Elfayer, Shuvo Habib, Slava
20	Representación condicional	jaredsk, m_callens, Nirazul, user6939352
21	Representación de lista	chuanxd, gurghet, Mahmoud, Theo
22	Usando "esto" en Vue	Bert
23	Vigilantes	El_Matella
24	Vue componentes de un solo archivo	jordiburgos, Ru Chern Chong
25	VueJS + Redux con Vua-Redux (la mejor solución)	Aniko Litvanyi, FlatLander, Shuvo Habib, Stefano Nepa
26	Vuex	AldoRomo88, Amresh Venugopal, Daniel Waghorn, Matej Vrzala M4, Ru Chern Chong