Introducción a Vue.js

abril 27, 2017 Mauro Caffaratto Grandes Sin comentarios Tutoriales

1. Introducción

En este tutorial veremos qué es y cómo funciona el framework vue.js, y cuáles son sus fortalezas y debilidades respecto a otros frameworks similares, como angular.js o react.

Índice de contenidos

2. Entorno

Cualquier editor de texto y cualquier navegador compatible con ES5 te valdrá para ejecutar el código que encontrarás en este tutorial.

3. ¿Qué es vue.js?

Vue.js es, en palabras de Evan You, su creador, un **framework progresivo para crear interfaces de usuario**.

Vue se basa en una implementación muy ligera del patrón view-model que nos ayudará a relacionar nuestra capa de presentación con nuestra capa de negocio de forma sencilla y eficiente.

A lo que se refiere You con 'progresivo', es a que es **muy fácil añadir Vue.js a cualquier proyecto ya existente** y poder aprovechar su funcionalidad casi sin complicaciones. Por el contrario, otros frameworks como Angular son mucho más opinionados y están más orientados a comenzar proyectos desde cero, y con una arquitectura predeterminada que viene dirigida por el framework.

You era un ingeniero front-end de google, y precisamente esto es lo que no le gustaba de Angular, aunque sí le parecía que el data-binding y las directivas de Angular eran una gran idea. Vue comenzó originariamente como un proyecto que aspiraba a reproducir este set mínimo de funcionalidades de Angular sin obligar al usuario a amoldarse a una determinada arquitectura. Comenzó a trabajar en Vue en 2013 y lo publicó en github en febrero de 2014.

Actualmente Vue ha tenido un gran impacto y hay varias compañías de renombre que utilizan Vue en su front-end, como **Gitlab, Alibaba o Nintendo**. Vue, seguramente debido a que Evan You es chino, ha tenido mucho impacto en compañías asiáticas.

Vue.js se caracteriza por ser **uno de los frameworks javaScript de mayor rendimiento**, gracias a una implementación muy ligera del llamado virtual dom. Aunque su núcleo se centra unicamente en la capa de vista y no provee herramientas de routing o control de

estado, hay muchas herramientas del ecosistema de Vue para escarlarlo y crear SPA's complejas. A sus fortalezas añade su adaptabilidad: se puede añadir a aplicaciones web ya existentes y aprovecharlo sin mayores problemas de instalación, simplemente cargándolo desde un `<script>`, pero también utilizarlo para crear aplicaciones complejas con un proceso de build que utilice commonJS o módulos ES6, hot module replacement y demás ventajas de bundlers como webpack o browserify durante el proceso de desarrollo.

4. Introducción a los view-models

El patrón view-model o *Model-view-view-model* consiste en añadir una capa de lógica entre nuestra capa de negocio y nuestra capa de UI para evitar que una tenga que comunicarse directamente con la otra: cuando la vista -la representación de una entidad en la capa de presentación-, o el modelo -su representación en la capa de negocio- cambien, **el view-model se encargará de actualizar su contrapartida en la otra capa de forma automática**.

esto facilita el testing y permite crear programas más modificables y reusables, a parte de evitar una gran cantidad de código 'boilerplate' que actualiza manualmente uno en función del otro.

Supongamos, por ejemplo, que en una página web de venta online de algún tipo de producto, tenemos una entidad 'pedido' tal que, en función de su estado, mostramos o no determinados bloques de html con determinado contenido. Cuando el usuario interactúa con la página, añadimos o modificamos campos de algún formulario que representa el estado final del pedido y, cuando el usuario quiere realizar la compra, enviamos ese formulario, que tendrá toda la información relevante, a algún controlador de nuestra página que procese la transacción.

Realizando este tipo de programas sin utilizar el patrón view-model es fácil que cometamos los siguientes errores:

- Mezclemos lógica de negocio con lógica de presentación.
- Almacenemos temporalmente información de negocio en el DOM.
- Tengamos que gestionar manualmente cómo los cambios en una de las dos capas se reflejan en la otra.
- Tengamos que preocuparnos constantemente de declarar y eliminar `handlers` que escuchen eventos.
- Hagamos que lógica que debería ser pura, y por tanto fácil de probar en tests, dependa del estado del DOM, lo que dificulta la testabilidad y reusabilidad.

Las librerías de interfaz de usuario como vue.js que implementan este patrón, utilizan mecanismos que permiten, **declarativa y automáticamente**, comunicar cambios entre la capa de negocio y la capa de presentación. De este modo hay una fuente única de verdad, el modelo, y el DOM se ajusta a él en función del comportamiento de la aplicación.

Esto implica un cambio de paradigma respecto a la forma en la que se solía trabajar en javaScript con interfaces de usuario. Si antes era habitual utilizar jQuery constantemente para leer y manipular el DOM, ahora manipulamos únicamente modelos de datos en javaScript, y dejamos que sea el viewModel el que ajuste el DOM en consonancia.

5. la instancia de Vue

La forma más immediata de trabajar con Vue es instanciar Vue para un tag determinado de nuestra aplicación. Simplemente, crearemos una nueva instancia de Vue llamándolo como un constructor:

```
1     new Vue(params);
```

Es fácil ver cómo funciona la instanciación de Vue a través de este ejemplo en codePen.

En el ejemplo, `params` es un objeto javaScript con determinadas claves. Este objeto puede tener definidas muchas propiedades, aunque la fundamental es `el` que debe ser un string con un selector css indicando para qué elemento aplicará esta instancia de Vue que estamos creando.

El resto de propiedades de una instancia Vue indican sus datos, métodos que operan sobre estos datos, handlers de eventos que afecten a ese bloque html, o diversos hooks que podemos utilizar para realizar tareas en algún momento específico de su ciclo de vida.

Una vez hecho esto, podemos utilizar Vue para modificar y completar el html de la aplicación mediante templates y directivas:

Los 'templates' son expresiones escapadas entre corchetes en el contenido de nuestro html que Vue interpreta y renderiza de acuerdo a un modelo de datos. De este modo, podemos utilizar Vue en el front-end de manera similar a cualquier motor de templates de back-end, como Freemarker, Pug o Jade. En el caso de Vue, todo lo que se encuentra entre '{{' y '}}' en determinados tags html se interpreta como javascript y su valor es modificado en función del valor de nuestros objetos.

Las directivas son atributos html que dirigen el comportamiento de la aplicación. Las directivas en Vue.js son muy similares a las de Angular 1. Por ejemplo, `v-if` añade o no un tag al DOM en función del valor de algún campo, o `v-model` enlaza controles de formularios con variables de nuestro modelo de datos.

Tanto los templates como las directivas están reactivamente enlazados con el modelo de datos: cuando uno cambia, el otro también lo hace. Esto es lo que se llama two way data binding, y hace que trabajar con formularios usando Vue sea mucho más cómodo que manejando handlers a mano.

Cuando una aplicación o un caso de uso empieza a tener un tamaño considerable, el two way databinding se considera una mala práctica, porque podemos tener múltiples elementos de la página modificando el modelo sin control. Por este motivo, cuando se utilizan componentes, esto cambia y el binding entre los componentes fluye unicamente de los padres a los hijos.

6. Propiedades de una instancia de Vue

Hay varias opciones que podemos especificar al crear una instancia de Vue. Un pequeño ejemplo más o menos completo sería como el que sigue:

```
1    new Vue({
2       el : '#app',
3       template : `<div v-bind:style="estilos" v-on:click="log()"`,
4       data : { titulo : 'Rectángulo', largo : 100, alto : 40 },
5       computed : {
6           area : function(){ return this.largo * this.alto; } ,
7           estilos : function(){
8           return {
9           width : this.largo + 'px',
10          height : this.alto + 'px'
11        }
12
13       },
14      methods : { log : function(){ console.log('Soy un rectángulo con un area de ' + this.area + ' pixels' ); } },
        created : function(){ console.log('se crea la instancia: todavía no se ha reemplazado "#app" por el template de esta
15  instancia'); },
        mounted : function(){ console.log('ya se ha renderizado el template en el DOM'); }
16
17  });
18
```

Podéis ver ese mismo script funcionando en este codepen. La especificacíon completa de las propiedades de Vue se encuentra en la documentación de la API de Vue.

Como se vé en el ejemplo, aquí se está instanciando Vue con un objeto que tiene varias propiedades. Vamos a verlos por orden:

El

La propiedad `el` determina, como ya hemos dicho, el tag html sobre los que operará nuestra instancia Vue. Es un selector CSS como los que se utilizan en las hojas de estilo o jQuery.

Es importante notar que **cada instancia de Vue apunta a un solo nodo html y sus hijos**. Podemos utilizar como selector una clase o una etiqueta, pero la instancia solamente operará sobre el primer nodo que encuentre.

Template

Como `template`, se indica una cadena con html que reemplazará el html que hemos fijado como `el`. Nota que hay dos cosas distintas denominadas 'template': todo contenido entre `{{}}` y la propiedad `template` de la instancia. Son dos cosas distintas, aunque generalmente van juntas.

No es necesario usar la propiedad `template` para trabajar con Vue. Podríamos haber puesto ese mismo código directamente en el html dentro del `div#app` y hubiera funcionado de manera similar. A veces es deseable mantener el html en ficheros .html y no llevarlo a javaScript. Pero la opción `template` también tiene sus ventajas:

- Si el html y el javaScript van a cambiar a la vez a menudo, los mantenemos en el mismo fichero.
- Si el html que estamos escribiendo no es válido **antes** de que Vue lo parsee y lo modifique, evitamos que el navegador lo elimine antes de que Vue pueda operar sobre él.
- Podemos usar javaScript para generar el template, lo que en según que casos puede ser cómodo y evitar duplicidades en el html.

Data

La propiedad `data` es un objeto con propiedades que podemos utilizar en nuestra instancia. En este caso, tanto la directiva `v-bind` como el método `area()` refieren estas propiedades. También se utiliza entre corchetes en el `template`. **Básicamente, es el estado de nuestro objeto**, el `model` del patrón `view-model`.

Cuando se crea la instancia, el motor de Vue lee todas las propiedades en `data` y les añade getters y setters, utilizando la 'magia' de `Object.defineProperty` para que sean reactivas y quedar a la escucha de sus cambios. La reactividad en Vue tiene ciertas limitaciones: si añadimos propiedades a un objeto dentro de `data` directamente con javascript, Vue no podrá escuchar estos cambios.Así que si uno de los campos de `data` es una lista de objetos y renderizamos html en función de estas listas, añadir un objeto a la lista o cambiar a mano el índice puede provocar que los cambios no se propaguen a la vista. De todos modos, en esos casos, el método $set nos tiene cubiertos.

Computed

Computed es un objeto que contiene una serie de métodos que devuelven valores calculados a partir de las propiedades del objeto. En nuestro ejemplo, valores derivados como el área, o el perímetro del rectángulo son los típicos casos en los que usaríamos `computed`.

Además tenemos un campo `estilos` que indica algunos estilos que se aplicarán al componente. Esto se consigue mediante el atributo `v-bind` que hay en el `template`. Más abajo veremos el tema de las directivas.

Methods

Methods es un objeto con funciones que podemos invocar, y cuyo contexto está enlazado con el propio objeto. Cuando queramos modificar las propiedades de nuestro objeto, o enviar eventos a otros componentes o instancias de la página, lo haremos desde alguno de estos métodos.

Nota que, moviendo `area` a `methods` y añadiendo `()` a su llamada en el método `log`, podríamos usar un método en vez de una `computed property` para indicar el área del rectángulo. La cosa quedaría como:

```
1     methods : {
2          log : function(){
3                console.log('Soy un rectángulo con un area de ' + this.area() + ' pixels' );
4          },
5          area : function(){
6              return this.largo * this.alto;
7          }
8
9      }
10     ...
```

Esto es una mala práctica y dará problemas de rendimiento, dado que el resultado de `computed` queda guardado en memoria y solamente cambia cuando cambia alguna de sus dependencias -en este caso, las propiedades `alto` y `largo`, mientras que las funciones en `methods` se invocan siempre.

Hooks

`Created` y `mounted` son funciones que se ejecutan en determinados momentos del ciclo de vida de la instancia. `Created` se invoca cuando se crea el objeto, mientras que `mounted` se invoca justo después de que el motor de Vue reemplaze el html que hubiera en `el`.

Hay otros hooks que podemos consumir, para realizar acciones cuando una instancia es destruida, es modificada… etc. etc. Generalmente `mounted` es el más utilizado y de alguna manera es similar a `$( function(){} )` en jQuery: nos asegura que el código no se va a ejecutar hasta que Vue haya terminado de inicializar el html de la instancia.

La siguiente imagen es una representación esquemática bastante completa del ciclo de vida de una instancia -o un componente- y de sus hooks:

Directivas

Habrás notado que en el html del `template` hay dos atributos: `v-bind:style` y `v-on:click`. Esto son directivas: atributos específicos de Vue.js que tienen asociada una determinada funcionalidad. En este caso, como es fácil ver, `v-bind` enlaza atributos -en este caso style- con valores de la instancia, y `v-on` añade un listener en un evento determinado, en este caso `click`. En el caso particular del atributo html `style`, como contiene css y la sintaxis de css es básicamente la de un json, podemos pasar un objeto como parámetro a `v-bind`: en este caso nuestra `computed property` 'estilos'.

El string que se pasa a las directivas es interpretado como javascript, con el contexto de `this` asociado a la propia instancia.

Al igual que en angular.js, hay bastantes directivas predeterminadas para modificar el markup en función de nuestro modelo de datos. `v-for`, que genera listas, y `v-if`, que genera html condicionalmente, tal vez sean las más utilizadas, así que he creado codepens para ilustrarlas:

- v-for
- v-if

Una lista exhaustiva de todas las directivas se puede encontrar en la documentación.

7. Componentes

Los componentes son elementos html reusables y configurables, y nos permiten definir etiquetas que podemos usar en nuestro markup dotadas de una determinada api y comportamiento. Aunque su estructura no es idéntica a la de los custom elements, es posible convertir vue components a custom elements con facilidad mediante vue-custom-elements.

Aunque la Api más inmediata de Vue es su constructor, **utilizando meras instancias de Vue no podemos montar con facilidad instancias dentro de otras ni reutilizar código para casos de uso similares**. Para esto, deberemos usar componentes. Un componente es, básicamente, una instancia Vue que será utilizada dentro de otra, y que será referenciada mediante una etiqueta específica. De este modo, podemos extender el html para crear nuevas etiquetas con un comportamiento y unos datos dirigidos por Vue. Por ejemplo, si definimos un componente como `<usuario>`, referenciar esa etiqueta dentro de código html parseado por Vue generará html válido -en función de la propia lógica del componente-.

Los componentes son componibles, de tal modo que se pueden añadir componentes dentro de componentes, y tienen un **orden jerárquico**: los padres pueden modificar los datos de los hijos, pero no al revés. Los hijos únicamente pueden notificar eventos a los padres, pero no modificar su estado directamente. Esto puede parecer una molestia, pero definitivamente mejora el workflow y evita

muchos bugs y condiciones de carrera, y asegura que cuando modifiquemos nuestros modelos haya un punto de entrada centralizado que gestiona estos cambios.

El uso de los componentes va más allá de este tutorial en concreto, pero he creado un codepen para ilustrar un caso de ejemplo muy básico.

8. Ecosistema

El core de Vue es adecuado para aplicaciones no muy grandes, y especialmente para aplicaciones web donde cada página se sirve por separado desde el back-end. Para aplicaciones SPA, sin embargo, necesitaremos alguna herramienta para gestionar las rutas y el estado de nuestra aplicación. Además, en el ecosistema de Vue hay muchas herramientas para utilizar webpack o browserify a la hora de ensamblar una aplicación basada en vue.js.

Precisamente una de las razones de ser de Vue es no atar a los desarrolladores a una herramienta o arquitectura en concreto, por lo que podemos utilizar casi cualquier librería de routing, de control de estado o relacionada con cualquier otro problema con Vue. De hecho, es bastante común usar Vue con redux, un motor de control de estado desarrollado originalmente para React.

Con todo, Vue tiene su propia librería de control de estado: Vuex, y su propio router oficial: vue-router.

Originalmente Vue tenía su propia librería oficial para hacer peticiones http, vue-resource, pero fue retirada por considerarse que, mientras que el router y el gestor de estado pueden estar acoplados al core de Vue, y por tanto tiene sentido tener herramientas específicas para Vue, ese no es el caso de un cliente http. La más popular entre los usuarios de Vue es Axios.

He encontrado muy útil el webpack-vue-loader, que permite codificar componentes con una sintaxis similar a la de los template elements, y dotar al componente de CSS contextual, que permite una privacidad de estilos similar a la que provee el shadow DOM sin necesidad de polyfills.

Por otro lado, hace poco los ingenieros de Alibaba y el equipo de Vue empezaron a colaborar en la herramienta weex, aún en desarrollo, cuyo objetivo es renderizar componentes vue de forma nativa en plataformas móviles, de forma similar a cómo funciona React native.

9. Conclusiones

Lo malo

Aunque Vue hace muchas cosas bien y puede ser la herramienta adecuada para muchos proyectos, también tiene sus contrapartidas.

Para proyectos con equipos muy grandes y mucha rotación de gente, el hecho de que Vue no tenga opiniones sobre la arquitectura puede llevar a la anarquía o al uso de anti patrones. En el mundo de la consultoría, donde los programadores rotan mucho y los desarrollos se delegan en terceros, esto hace que soluciones como Angular sean más apreciadas.

Vue es, además, un producto muy nuevo, y aunque tiene un ecosistema muy rico, algunas herramientas ligadas a él no son tan maduras como en otras alternativas. La herramienta de scaffolding por línea de comandos de Vue, vue-cli, tiene fama de no ser muy completa. La extensión de depuración de Vue, por otro lado, no provee toda la información que muchas veces sería deseable.

A veces, los mensajes de error de Vue no son todo lo informativos que uno desearía: eventualemente el motor no puede renderizar un componente por algún error de sintaxis y falla silenciosamente, o no se comporta como debería por errores en la declaración de un componente o una instancia que deberían producir errores más claros. **No obstante, muchos problemas de este tipo que he tenido mientras he ido probando Vue se han ido resolviendo según han ido apareciendo nuevas versiones**.

Por otro lado, Vue no tiene el apoyo de gigantes como Google o Facebook, que es un motivo que muchas compañías utilizan para decidirse por Angular o React, dado que temen que Vue sea abandonado por sus desarrolladores. Este es un problema serio para una gran compañía que está preparando un producto que debe mantenerse durante años.

Lo bueno

Vue tiene una gran documentación, lo que junto con la simplicidad de su diseño hace que **aprenderlo y empezar a ser productivos con él sea muy sencillo**. Es de los mejores trabajos de documentación de una herramienta de software que he leído en mi vida, y llevo unos cuantos.

Vue tiene de los mejores rendimientos entre frameworks js que hay en el mercado: esto es algo que en el front-end no es tan crítico como en el back-end, pero siempre es un punto a favor.

A pesar de no contar con el apoyo de compañías tecnológicas tan grandes como Google o Facebook, **Vue es utilizado intensamente por Alibaba**, y hay bastantes compañías importantes que dependen de Vue, lo que lleva a pensar que es difícil que su desarrollo sea abandonado. Su adopción en este último año ha crecido enormemente, y Evan You creó un patreon para financiar Vue que ha tenido bastante éxito y ha ayudado a contribuir al desarrollo de la herramienta.

Vue sea seguramente de las mejores soluciones posibles para refactorizar aplicaciones webs que llevan varios años en desarrollo en las que el front-end no está todo lo limpio que debiera, pero donde los recursos, las presiones de tiempo o la acumulación de deuda técnica impiden rehacer la aplicación de una vez como una SPA. Por otro lado, es una opción perfectamente viable para hacer grandes SPA's, aunque en ese nicho hay muchas más alternativas.

10. Referencias

- Post de presentación de Vue 2.0.
- Awesome vue, un listado muy completo de recursos relacionados con Vue.
- Comparación de Vue con otras alternativas. Es una página de la documentación de Vue, por lo que tal vez no sea imparcial, pero es muy interesante -además, la comparación con React está consensuada con el propio equipo de React-.
- Why we chose Vue. Un artículo de Jacob Schatz, desarrollador de Gitlab, sobre por qué eligieron Vue para su front-ent.
- Computed properties internals, un artículo sobre las `computed properties` en Vue y en general sobre reactividad en javaScript.
- Entrevista a Evan You, habla del origen y el futuro de Vue y de varios temas.
- Vue.js, la revolución sencilla. Una charla de Rafa Casuso de introducción a Vue. es de los mejores recursos en castellano sobre Vue.