React.JS

Introducción

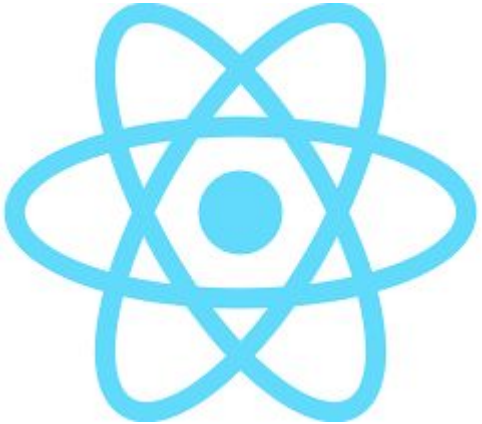
Módulo 01

Conceptos Básicos

¿Qué es y para qué sirve React.JS?

React (también llamada React.js o ReactJS) es una librería Javascript de código abierto para crear interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones en una sola página. Es mantenido por Facebook, Instagram y una comunidad de desarrolladores independientes y compañías 1.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; está construida para poder ser utilizado con el patrón de diseño modelo–vista–controlador (MVC), y puede ser utilizada conjuntamente con otras bibliotecas de Javascript o más grandes #MVC como AngularJS. También puede ser utilizado con las extensiones de React-based que se encargan de las partes no-UI (no gráficas) de una aplicación web.



Es importante señalar que ReactJS es una librería enfocada en la visualización. Si estamos iniciando un proyecto podemos basarnos en la arquitectura Flux, pero si ya tenemos un proyecto usando un Framework MVC como AngularJS podemos dejar AngularJS como Controlador y que ReactJS se encargue de las vistas.

Esto tiene sentido pues ReactJS tiene un performance superior al momento para manipular el DOM, y esto tiene un gran impacto cuando se trata con listas largas que cambian constantemente en nuestra visualización 2.

ReactJs es una librería enfocada en la visualización

El secreto de ReactJS para tener un performance muy alto, es que implementa algo llamado Virtual DOM y en vez de renderizar todo el DOM en cada cambio, que es lo que normalmente se hace, este hace los cambios en una copia en memoria y después usa un algoritmo para comparar las propiedades de la copia en memoria con las de la versión del DOM y así aplicar cambios exclusivamente en las partes que varían. Esto puede sonar como mucho trabajo, pero en la práctica es mucho más eficiente que el método tradicional pues si tenemos una lista de dos mil elementos en la interfaz y ocurren diez cambios, es más eficiente aplicar diez cambios, ubicar los componentes que tuvieron un cambio en sus propiedades y renderizar estos diez elementos, que aplicar diez cambios y renderizar dos mil elementos.

El secreto de ReactJS es impactar en un DOM en memoria (Virtual DOM) y luego hacer los cambios en el DOM real, en vez de impactar en el DOM real directamente

Son más pasos a planear y programar, pero ofrece una mejor experiencia de usuario y una planeación muy lineal. Una característica importante de ReactJS es que promueve el flujo de datos en un solo sentido, en lugar del flujo bidireccional típico en Frameworks modernos, esto hace más fácil la planeación y detección de errores en aplicaciones complejas, en las que el flujo de información puede llegar a ser muy complejo, dando lugar a errores difíciles de ubicar que pueden hacernos la vida muy triste.

ReactJs promueve el flujo de datos en un solo sentido

VirtualDOM

El principal problema es que DOM nunca fue optimizado para crear una IU dinámica. Podemos trabajar con él usando JavaScript y bibliotecas como jQuery (gracias a Dios que lo tenemos). Pero jQuery y otros hicieron poco para resolver problemas de rendimiento.

Piensa en redes sociales modernas como Twitter, Facebook o Pinterest. Después de desplazarse un poco, el usuario tendrá decenas de miles de nodos. Interactuar con ellos de manera eficiente es un gran problema. Trate de mover un 1000 divs 5 píxeles a la izquierda, por ejemplo. Puede tomar más de un segundo. Es mucho tiempo para la web moderna.

Puede optimizar el guión y utilizar algunos trucos, pero al final, es un dolor trabajar con páginas enormes y una interfaz de usuario dinámica. Para solucionar esto se proponen las siguientes alternativas.

> **El DOM nunca fue optimizado para Interfaces de Usuario dinámicas, como las que requieren las aplicaciones web modernas como Facebook, Instagram o Netflix**

VirtualDOM - Ejemplo

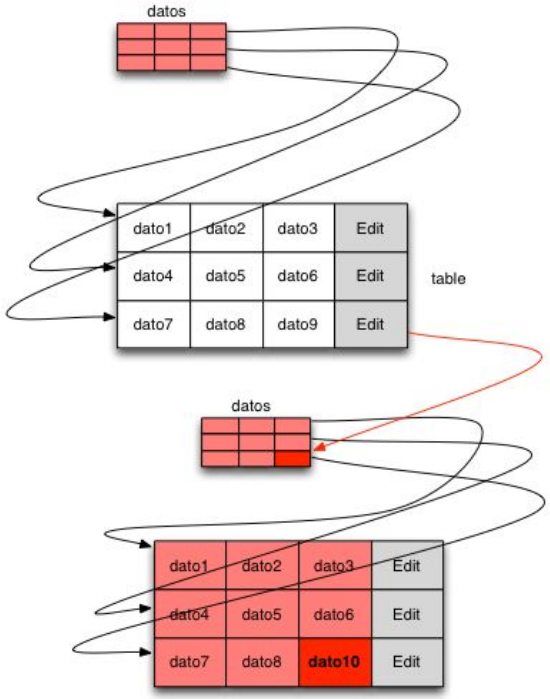
Imaginémonos que disponemos de una tabla con datos en una página HTML. Esta tabla dispone de en cada fila de un botón de edición. Este botón nos permite cambiar de forma rápida los datos de la tabla. Por ejemplo podemos pulsar en la última fila y cambiar el dato9 por dato10.

dato1	dato2	dato3	Edit
dato4	dato5	dato6	Edit
dato7	dato8	**dato10**	Edit

VirtualDOM - Ejemplo

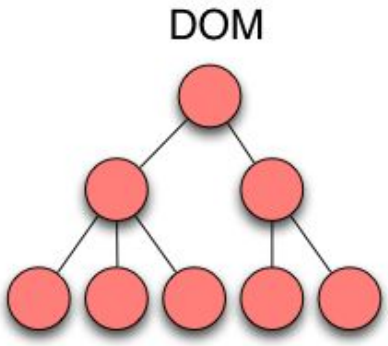
¿Cómo realizamos esta operación?

En la mayor parte de las situaciones los desarrolladores optan por una solución similar a lo siguiente. La tabla se crea a partir de un array de objetos, este array de objetos es actualizado cuando editamos. Finalizada la edición, la tabla se vuelve a generar.



VirtualDOM - Ejemplo

dato1	dato2	dato3	Edit
dato4	dato5	dato6	Edit
dato7	dato8	dato10	Edit



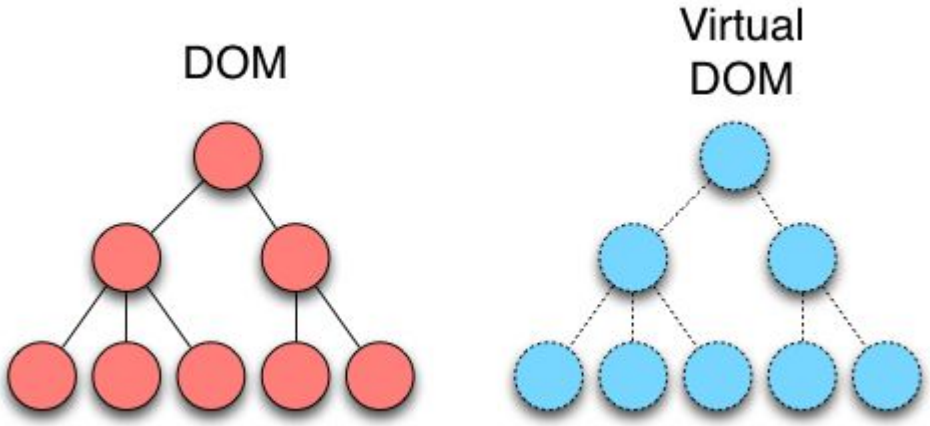
Nos encontramos ante un código que funciona correctamente pero que tiene un problema importante. ¿Era necesario volver a generar la tabla entera? Se trata de una pregunta interesante ya que solo hemos cambiado un dato puntual.
La respuesta es NO.
Acabamos de forzar al navegador a redibujar todo el bloque cuando solo un mínimo de información cambiaba.

Cambiar los nodos de un árbol DOM tiene un coste , cuantos más nodos cambiemos mayor será el coste para el navegador. En situaciones sencillas los problemas de rendimiento no se notan pero cuanto más compleja y grande sea nuestra web estas cosas serán importantes.
En este caso nos encontramos con una tabla de datos que define una parte del árbol DOM

VirtualDOM - Ejemplo

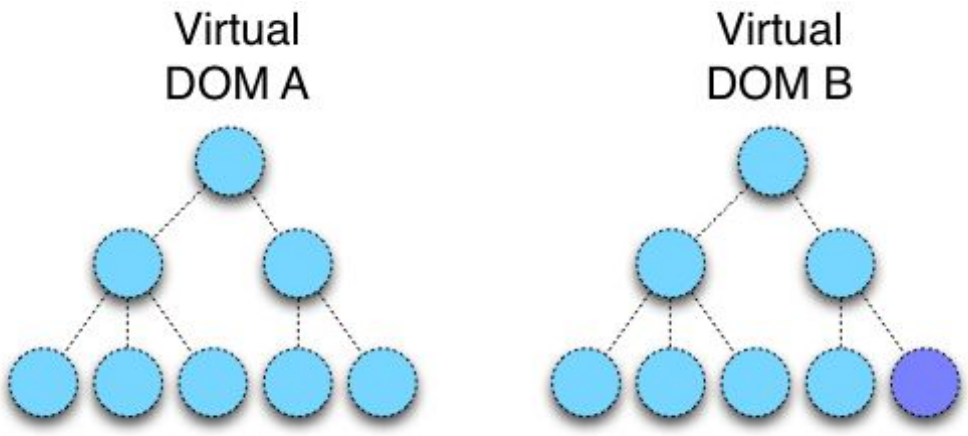
Lamentablemente cuando volvemos a dibujar la tabla estamos cambiando todos los nodos del árbol DOM y volviéndola a renderizar por completo. ¿Cómo funciona un enfoque de virtual DOM?

En este caso el framework o librería que corresponda se encarga de tener una copia de nuestro árbol DOM simplificado en memoria a la cual se denomina VirtualDOM.



VirtualDOM - Ejemplo

Una vez que dispone de esta versión del árbol el framework se encarga de estar pendiente de los cambios que se generan sobre él. Siendo capaz de ver las diferencias entre un estado A y un estado B cuando nosotros actualizamos información. En este caso solo hemos cambiado un nodo del árbol:



JSX

Para hacernos la vida mucho más fácil, ReactJS nos ofrece un pseudolenguaje llamado JSX que facilita el desarrollo de aplicaciones web con su sintaxis para crear elementos en el DOM. Recordemos que ReactJS se enfoca en la visualización y literalmente debemos armar el HTML que deseamos directamente en JSX, con el fin de describir nuestros componentes por medio de etiquetas y de una sintaxis muy parecida a la de HTML, que es compilada a Javascript. No podemos usar directamente el JSX, pero tenemos dos opciones para compilarlo.

1. Usando JSXTransformer que hace la conversión directamente en el navegador del programador.
2. Para producción se recomienda usar el precompilador basado en NodeJS que nos genera JavaScript nativo y evita que el JSX tenga que ser compilado por el usuario.

Para facilitar el manejo del Virtual Dom, ReactJs ofrece una sintaxis alternativa para definir elementos llamada JSX

JSX

Algo importante de señalar es su curva de aprendizaje, pues al principio puede parecer bastante complicado, pero gracias a la sintaxis de JSX y al hecho de que todo se va armando por módulos compilados de forma secuencial, uno se adapta muy rápido y lo domina en poco tiempo, pues es relativamente sencillo, sobre todo si se compara con AngularJS. Además su estructura es más manejable que Backbone. El truco para dominar ReactJS es aprender a dividir todo en pequeños componentes que se agrupan para formar una aplicación. Personalmente me gusta ir de lo general a lo particular, pero es posible (y recomendable en aplicaciones complejas) iniciar con una serie de pequeños componentes, e irlos ensamblando hasta crear una aplicación compleja.

El truco para dominar ReactJS es aprender a dividir todo en pequeños componentes que se agrupan para formar una aplicación

¡Muchas gracias!

¡Sigamos trabajando!