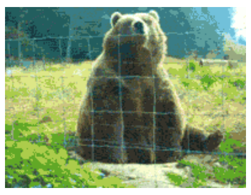


The one thing that no one properly explains about React — Why Virtual DOM

Sai Kishore Komanduri

Published 3 years ago
JavaScript React

The other day a friend had this React question for me: “Composition through components, one way data binding; I understand all that, but why Virtual DOM?”.

I’ve given him the usual answer. "Because, direct DOM manipulation is inefficient, and slow."

“There’s always news on how JavaScript engines are getting performant; what makes adding something directly to the DOM slow?”

…

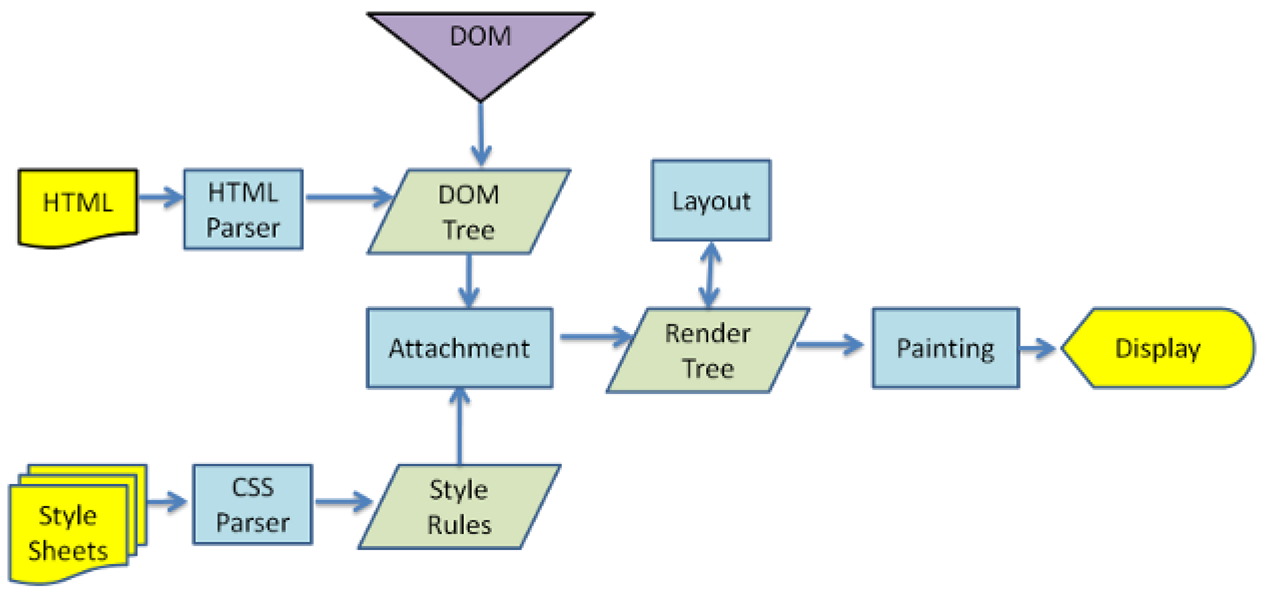
That is a great question. Surprisingly, I’ve not found any article that properly pieces it all together, making the case for the need of a Virtual DOM rock solid.

It’s not just the dir
happens after.

To understand the need for a Virtual DOM, lets take a quick detour, a 30000 feet level view on a browser's workflow, and what exactly happens after a DOM change.

A Browser's Workflow

NOTE: The following diagram, and the corresponding explanation uses Webkit engine's terminology. The workflow is almost similar across all browsers, save for a couple of nuances.



Creation of the DOM tree

- Once the browser receives a HTML file, the render engine parses it and creates a DOM tree of nodes, which have a one-one relation with the HTML elements.

Creation of the Render tree

- Meanwhile, the styles both from external CSS files, and inline styles from the elements are parsed. The style information, along with the nodes in the DOM tree, is used to create another tree, called the render tree

Creation of the Render Tree — Behind the scenes

- In WebKit, the process of resolving the style of a node is called "attachment". All nodes in the DOM tree have an "attach" method, which takes in the calculated style information, and return a render object (a.k.a. renderer)
- **Attachment is synchronous, node insertion to the DOM tree calls the new node "attach" method**
- Building a render tree, consisting of these render objects, requires calculating the visual properties of each render object; which is done by using the calculated style properties of each element.

The Layout (also referred to as reflow)

- After the construction of the render tree, it goes through a "layout" process. Every node in the render tree is given the screen coordinates, the exact position where it should appear on the screen.

The Painting

- The next stage is to paint the render objects — the render tree is traversed and each node's "paint()" method is called (using browser's platform agnostic UI backend API), ultimately displaying the content on the screen.

Enter the Virtual DOM

So, as you can see from the above flow of steps, whenever you make a DOM change all the following steps in the flow, right from the creation of the render tree (which requires recalculation of all the style properties of all the elements), to the layout, to the painting step, all are redone.

In a complex SPA, often involving a large number of DOM manipulations, this would mean multiple computational steps (which could be avoided) which make the whole process inefficient.

This is where the Virtual DOM abstraction truly shines; when there's a change in your view; all the supposed changes that are to be made on the real DOM, are first made on the Virtual DOM, and then sent on to the real DOM, thus reducing the number of following computational steps involved.

Update: The following comment from redditor ugwe43to874nf4 does more justice to the prominence of Virtual DOM 👏

👍 65

> *The real problem with DOM manipulation is that each manipulation can trigger layout changes, tree modifications and rendering. Each of them. So, say you modified 30 nodes, one by one. That would mean 30 (potential) re-calculations of the layout, 30 (potential) re-renderings, etc.*
>
> *Virtual DOM is actually nothing new, but the application of "double buffering" to the DOM. You do each of those changes in a separate, offline DOM tree. This does not get rendered at all, so changes to it are cheap. Then, you dump those changes to the "real" DOM. You do that once, with all the changes grouped into 1. Layout calculation and re-rendering will be bigger, but will be done only once. That, grouping all the changes into one is what reduces calculations.*
>
> *But actually, this particular behaviour can be achieved without a virtual DOM. You can manually group all the DOM modifications in a DOM fragment yourself and then dump it into the DOM.*
>
> *So, again, what does a Virtual DOM solve? It automates and abstracts the management of that DOM fragment so you don't have to do it manually. Not only that, but when doing it manually you have to keep track of which parts have changed and which ones haven't (because if you don't you'd end up refreshing huge pieces of the DOM tree that may not need to be refreshed). So a Virtual DOM (if implemented correctly) also automates this for you, knowing which parts need to be refreshed and which parts don't.*
>
> *Finally, by relinquishing DOM manipulation for itself, it allows for different components or pieces of your code to request DOM modifications without having to interact among themselves, without having to go around sharing the fact that they've modified or want to modify the DOM. This means that it provides a way to avoid having to do synchronization between all those parts that modify the DOM while still grouping all the modifications into one.*

Further Reading

The above Browser workflow has been excerpted from this document on the internals of browser operations. It delves deeper into a browser engine's hood, explaining everything in detail; definitely worth your time to read it from end to end. It helped me a great deal in understanding the "why", and justifying the the need for a Virtual DOM abstraction.

Hope this was of help. Let me know if you have any questions in the comments.

Sai Kishore Komanduri

Engineering an eGovernance Product | Hashnode Alumnus | I love pixel art

Bangalore, India   fatman-   @_saikishore

FOLL

65

Write your comment…

POPULAR RECENT 11 comments

Hipkiss · · 3 years ago

So, does the vDOM do the WHOLE DOM? or just a specified singular element? I assume vDOM is based on something like MutationSummary (https://github.com/rafaelw/mutation-summary) for picking up changes to it? Furthermore is the vDOM essentially just a JSON/similar representation of the real DOM? Is it possible to view/access the vDOM in react?

👍 3

Show all replies

Hipkiss 3y

@@saiki Thanks for the pointer! Unfortunately I did actually already create a DOM serialiser! Thanks for the interest too :). I just thought I might be barking up the wrong tree (see what I did there). Thanks again

👍 1

Your reply

Reply to this…

Share your programming knowledge and learn from the best developers on Hashnode

Get started

Mev-Rael · · 3 years ago

Have seen this "explanation" and diagram many times. But still, useless and actually doesn't explain everything.

- When a Virtual DOM
otherwise it is just in
when I can do just A

👍 65

for example?
- When exactly a browser is doing "rerendering"? What about "forced reflow", "read first and write second"? Why you think modern browsers are not doing a lot of optimization already.
- Any real business code examples where everyone can compare good vanilla JS DOM manipulation and same code with Virtual DOM? Please no more useless innerHTML in the loop. I've been loading and rerendering 10000+ comments with plain JS easily.
- Any benchmarks?

These simple questions in normal situations developer can answer quickly, however in this "virtual" problem even core author of React couldn't answer any of them.

DOM isn't slow, you are - https://korynunn.wordpress.com/2013/03/19/the-dom-isnt-slow-you-are/

👍 1

Show all replies

Kyle Agronick 2m

Of course it stores the VDOM in memory. What would it compare against when it does a rerender?

The virtual dom will always be slower than manual dom updates. Just try making a list of 10,000 items and appending to the list. The virtual dom will have to do an expensive diff each time only to see that one element has changed. In the end it will call the same appendChild method that you would of called in Vanilla JS. A few times I've had to bail out of the virtual dom because it was just too slow.

The virtual dom is about making things easier for developers. It is not faster and it doesn't unleash some magical API that only the virtual DOM can access. If you don't believe me React core developers have said the same thing. Namely that doing what the virtual dom does in vanilla js will always be faster.

Batching updates doesn't make it faster either. Browsers wait for all tasks and microtasks to finish before rendering the next frame. So even if the virtual dom does dom updates all at once and you do it in vanilla JS mixed in with your business logic, it wont make a difference unless you are using certain properties and methods that trigger a reflow. If you use fragments you can beat the virtual dom every time.

👍

Your reply

Reply to this…

A A Karim · 2 years ago

Easy enough to understai

👍 65

👍 1

Your reply

Reply to this…

🔥 Posts You Must Read

FeathersJS Tip — On feathers transpilation, when using babel-register at server-side

Sai Kishore Komanduri · a year ago

👍 4 💬 Add comment

React Redux Hooks API is out!

Juni Brosas · 21 hours ago

👍 12 💬 Add comment

Part 2. Clock-in/out System: Basic backend (I) — AuthModule published: true

Carlos Caballero · a day ago

👍 17 💬 Add comment

Top 4 Tags You Should Follow on Hashnode This Week

Milica Maksimović · 20 hours ago

👍 25 💬 1

👍 65