Virtual DOM in ReactJS?

React is an **isomorphic/universal** framework. That means that there is a virtual representation of the UI component tree, and that is separate from the actual rendering that it outputs in the browser. From the documentation:

React is so fast because it never talks to the DOM directly. React maintains a fast in-memory representation of the DOM – Virtual DOM.

However, that in-memory representation is not tied directly to the DOM in the browser (even though it is called Virtual DOM, which is an unfortunate and confusing name for an universal apps framework).

Virtua DOM is just a DOM-like data-structure that represents all the UI components hierarchy and additional meta-data.

Q. Calling setState() in ComponentDidMount() does not causes infinite loop ? Whereas calling setState() in ComponentDidUpdate() does causes infinite loop ?

<https://stackoverflow.com/questions/43095112/react-setstate-in-componentdidmount-not-causing-infinite-loop>

Ans. When you setState in componentDidUpdate, the latter is being called again because the component should update, which results in endless recursion. On the other hand, componentDidMount is only called on the initial render.

Theory on componentWillMount() & componentDidMount()

**componentWillMount()**

One big snag that I didn’t realize until an online study group session is that componentWillMount() was deprecated in 2018. While you can continue to use it (until React 17.0), it isn’t best practice because it is not safe for async rendering. If you do choose to continue to use it, you should use UNSAFE\_componentWillMount().

The Reason

Using a fetch call within componentWillMount() causes the component to render with empty data at first, because componentWillMount() will NOT return before the first render of the component.

**Due to the fact that JavaScript events are async, when you make an API call, the browser continues to do other work while the call is still in motion. With React, while a component is rendering it doesn’t wait for componentWillMount() to finish, so the component continues to render.**

With all that being said, you would need to to create a component that still looks presentable without the data that you are hoping to display. There is no way (not even a timer) to stop the component from rendering until the data is present.

When I was building my project, I didn’t know about the deprecation of componentWillMount() and I didn’t understand why I kept getting the “cannot read property ‘map’ of undefined” error. I was so confused because I had an array and it should be getting populated by data, but it wasn’t.

Turns out this makes complete and total sense since the component is rendered with empty data at first, so the array was empty and couldn’t be iterated over. It was only at this point that I learned that it is best practice to use componentDidMount().

**componentDidMount()**

The best place to make calls to fetch data is within componentDidMount(). componentDidMount() is only called once, on the client, compared to componentWillMount() which is called twice, once to the server and once on the client. It is called after the initial render when the client received data from the server and before the data is displayed in the browser. Due to the fact that the data won’t be loaded until after the initial render, the developer NEEDS to set up the components initial state properly.

On a side note, with componentDidMount() you can use a timer and have the data updated every so often without the user ever having to refresh the page. I’d say that is a really neat feature that can be useful in projects/websites.

componentDidMount:

Invoked once, only on the client (not on the server), immediately after the initial rendering occurs. At this point in the lifecycle, the component has a DOM representation which you can access via React.findDOMNode(this).

You can contrast this with componentDidUpdate function, which is called everytime that React renders (except for the initial mount).

Component Mounting and Un-mounting in React

**Mounting:** Adding nodes to the DOM

**Updating:** Making changes to nodes already in the DOM

**Unmounting:**  Removing them from the DOM

For example:

I have a web page which is initially loaded with first tab: Reviews. The table that we see with Network, Date , Review and Rating columns is mounted when this web page is loaded, and we can say Review table is **MOUNTED onto the DOM.**

This table will get updated if I enter some value in the Search box above this table (which is supposed to trigger render()), then we can say the **DOM has been UPDATED.**

When I click on second tab named Network Share, then Reviews table will said to be **UNMOUNTED from the DOM** and Network share table or pie chart is mounted at same time onto the DOM.

