**Refactoring React ES6 Class Components with Property Initialisers**

“Proprty initialisers” is an experimental ES6 JavaScript feature that allows you to write React class components without constructor and bind this. It creates cleaner React class component. The use of this syntax is so wide-spread and calling it experimental (because it is not yet ratified) feels almost misleading.

Before getting into refactoring, we first need to add the babel plugin called @babel/plugin-proposal-class-properties so that babel can compile react. Without this plugin, babel will complain about arrow functions inside the class component. You can check out this pos for more reference ([Clarifying which babel to use for compiling React today](https://www.mydatahack.com/clarifying-which-babel-to-use-for-compiling-react-today/)).

**Example Classic Class Component**

Here is the classic class component. For the demo purpose, it has local state that captures id value as well as dispatchers that are mapped to props.

The first thing you do is to create a constructor and pass props to super. Then, create functions and bind them to this. This is necessary because of the (almost pathological) nature of JS global scoping. Without binding to this, these functions do not exist in the global this object.

<pre class=" toolbar:2 lang:js font-size:12px decode:true" >

</pre>

**Refectored Class Component with Property Initialisers**

When we use the property initialiser feature, we can get rid of constructor. Props are accessed by this.props. The local state can be defined as a class property. Functions can be arrow function, which means you can take a full advantage of lexical scoping. Hence, you do not need to bind them to this. In the render method, you do not need to call it with arrow function.

This feels cleaner, doesn’t it?

<pre class=" toolbar:2 lang:js font-size:12px decode:true" >

</pre>

Before wrapping it up, there is one more thing you may be interested in….

**Refectored Class Component to Functional Component**

Not exactly the same as the original class component, but you can entirely scrap the class component and make it to a functional component. Now, the onInputHandler dispatches action connected to props. It takes name and value and keep updating the global store. onSubmitHandler is the same. It dispatches action.

There is a catch. The general consensus of the community is that we should not put the form state in Redux. It is wasteful to dispatch action on every key stroke. Makes sense. It is actually in the Redux official documentation [here](https://redux.js.org/faq/organizing-state#should-i-put-form-state-or-other-ui-state-in-my-store). So, this is the anti-pattern that keep form input state in the global redux store unless you have a good use case that requires other changes depending on it.

<pre class=" toolbar:2 lang:js font-size:12px decode:true" >

</pre>