React Hooks

Understanding hooks

TOC

Demo: https://jolly-perlman-98367f.netlify.com

Overview

Hooks are a new feature proposal that lets you use

state and other React features without writing a class. They're currently

in React v16.7.0-alpha and being discussed in an open RFC

Understanding the problems

01 Hooks lets us use state inside functional component, previously if we need state in any component we need to make class based component.

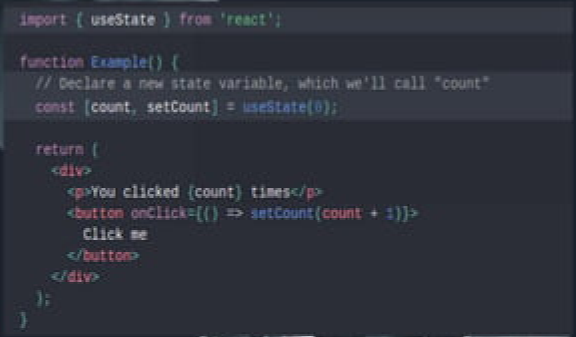
02 In functional component we were unable to use component's life cycle hooks

Like componentDidUnmount, componentDidUpdate.

03 With hooks now our functional components are no more stateless component

04 Classes confuse both people and machines, In addition to making code reuse and code organization more difficult, we've found that classes can be a large barrier to learning React. You have to understand how this works in JavaScript, which is very different from how it works in most languages. You have to remember to bind the event handlers.

useState and uses



```
import { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Here, useState is a Hook. We call it inside a function component to add some local state to it. React will preserve this state between re-renders. useState returns a pair: the current state value and a function that lets you update it. You can call this function from an event handler or somewhere else. It's similar to this.setState in a class, except it doesn't merge the old and new state together.

The only argument to useState is the initial state. In the example above, it is 0 because our counter starts from zero. Note that unlike this.state, the state here doesn't have to be an object — although it can be if you want. The initial state argument is only used during the first render.

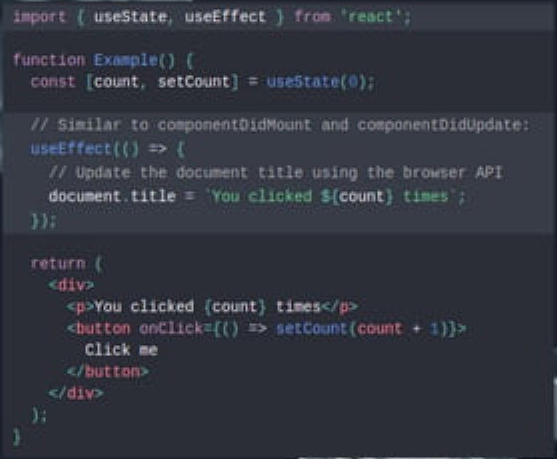
Some key points

- we need to use useState function to have state
- We can use String, Number, Array or Object in useState
- useState return array which we destructure, which is also called array destructuring.

```
const [state,setState] = useState('samundra') // ['samundra', function(state){ }]
```

- Doesn't work with class
- Doesn't work outside of component, on any vanilla javascript function description
- uses Linked List data structure
- Placement of useState matters

useEffect and uses



```
import { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

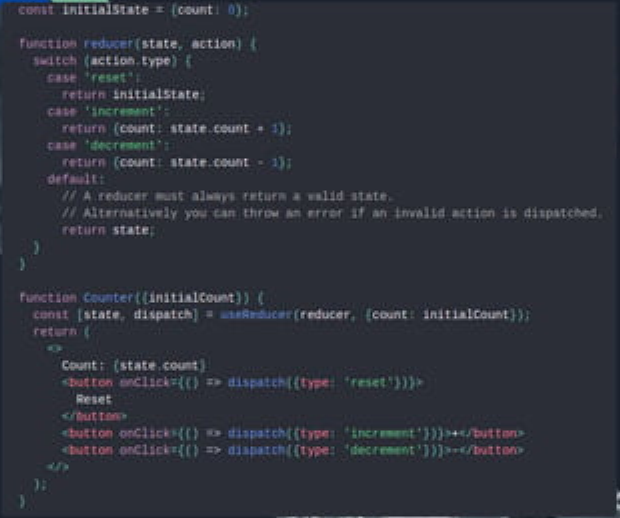
You've likely performed data fetching, subscriptions, or manually changing the DOM from React components before. We call these operations "side effects" (or "effects" for short) because they can affect other components and can't be done during rendering.

The Effect Hook, `useEffect`, adds the ability to perform side effects from a function component. It serves the same purpose as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` in React classes, but unified into a single API.

Some key points

- useEffect is effect which is equal to componentDidMount and ComponentDidUpdate.
- which means useEffect will run on mount and update
- will run only after the DOM is applied or DOM mutation is done.
- every time whenever a local state is changed this effect will run.
- we can make our effect run depending on other state by passing it in second argument as array
- We can make effect run only once by passing empty array in second argument

useReducer and uses



```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'reset':
      return initialState;
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      // A reducer must always return a valid state.
      // Alternatively you can throw an error if an invalid action is dispatched.
      return state;
  }
}

function Counter({initialCount}) {
  const [state, dispatch] = useReducer(reducer, {count: initialCount});
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'reset'})}>
        Reset
      </button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
    </>
  );
}
```

An alternative to useState. Accepts a reducer of type (state, action) => newState, and returns the current state paired with a dispatch method. (If you're familiar with Redux, you already know how this works.)

Some key points

- useReducer is an another hook provided by react
- when we want to manage state in single object we can use it
- Inspired by redux, we may not need redux if our app is small and we need some state management

Some key points

- useReducer is an another hook provided by react
- when we want to manage state in single object we can use it
- Inspired by redux, we may not need redux if our app is small and we need some state management

Custom hooks

When we want to share logic between two JavaScript functions, we extract it to a third function. Both components and Hooks are functions, so this works for them too!

A custom Hook is a JavaScript function whose name starts with "use" and that may call other Hooks.

Some key points

- Before there used to be mixins in react where we can share logic among components But later it was removed because of the state we have to share between components.
- Custom hooks are somewhat like mixins but without sharing state between components. Instead they will have their own state so it is less prone to bug.
- Can be used any number of time.
- Its name should always start with use so that you can tell at a glance that the rules of Hooks apply to it.

Useful hooks

- Basic Hooks
  - useState
  - useEffect
  - useContext
- Additional Hooks
  - useReducer
  - useCallback
  - useMemo
  - useRef
  - useImperativeMethods
  - useLayoutEffect

useContext

Accepts a context object (the value returned from React.createContext) and returns the current context value, as given by the nearest context provider for the given context.

When the provider updates, this Hook will trigger a rerender with the latest context value.

```
const context = useContext(Context);
```

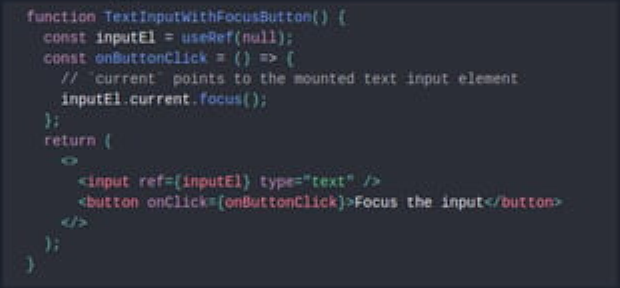
useCallback

```
const memoizedCallback = useCallback(
  () => {
    doSomething(a, b);
  },
  [a, b],
);
```

Returns a memoized callback.

Pass an inline callback and an array of inputs. `useCallback` will return a memoized version of the callback that only changes if one of the inputs has changed. This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders (e.g. `shouldComponentUpdate`).

useRef



```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

useRef returns a mutable ref object whose .current property is initialized to the passed argument (initialValue). The returned object will persist for the full lifetime of the component.

FAQ

- Is React hook future?
  - Maybe, currently its in alpha version and soon will be in stable version.
- Will React deprecate or plan to remove class base components?
  - No! Currently React team doesn't deprecate class base components or there are any plan in future to remove it but encourages you to use hooks in upcoming components.
- Do I need to rewrite all my class components?
  - No!

Thank you!