React Notes

Mohamed Emary

September 13, 2024

1 State Vs Variables

variable	state
loses its values after rerendering	preserves its values after rerendering
doesn't rerender when its value changes	rerenders when its value changes
can be used anywhere	can be used in the top level of the function components only

2 Function Components

1. Return JSX
2. Starts with a capital letter

3 Component Lifecycle

1. Mounting
2. Updating
3. Unmounting

3.1 Mounting

You mount the component either by adding its selector like `<About />` or by going to its route defined in the `App.jsx` file.

- The first thing to get called in the mounting phase is the `constructor()` method (if you use class components)
- The second is the render method which is the only required method in a class component, in a functional component you just return the JSX.

- The third method is `componentDidMount()` which is called after the component is rendered. It is the best place to make API calls.

3.2 Updating

It happens when a parent component passes new props to the child component or when the state of the component changes. It also happens when you call use `forceUpdate()` method.

Then the render method is called again to re-render the component and the `componentDidUpdate()` method is called after the component is rendered.

`componentDidUpdate()` tells you the update phase was triggered by whome?

3.3 Unmounting

Happens when you delete a component or moving from one route to another.

The `componentWillUnmount()` method is called before the component is removed from the DOM.

The `componentWillUnmount()`:

1. Cleans up the component
2. Removes event listeners

3.4 How to control the component in each of the 3 phases

1. In `componentDidMount()` you can use `useEffect()` hook with an empty array as the second argument.

```
useEffect(() => {
  // componentDidMount code
}, []);
```

2. In `componentDidUpdate()` you can use the `useEffect()` hook with a dependency array.

```
useEffect(() => {
  // componentDidUpdate code
}, [dependency]);
```

Notice that this code will also run in the `componentDidMount()` phase, to prevent this you can use a flag to check if the component is mounted or not.

```
const [isMounted, setIsMounted] = useState(true)

useEffect(() => {
  if(isMounted){
    console.log("Mounting code");
    setIsMounted(false);
  }else{
    console.log("Updating code");
  }
}, [dependency]);
```

1. In `componentWillUnmount()` you can return a function from the `useEffect()` hook.

```
useEffect(() => {
  // componentDidMount code

  return () => {
    // componentWillUnmount code
  };
}, []);
```

3.5 Important Notes

- Always add the empty array as the second argument to the `useEffect()` hook to make it run only once and not every time any state changes.

```
useEffect(() => {
  // This code will run in all phases
  // It will also run with every state change
});
```

- When using `useEffect` it's better to keep the logic of the `componentDidMount()` and `componentWillUnmount()` in separate `useEffect()` hooks since both don't need any dependencies. And make a separate `useEffect()` hook for the `componentDidUpdate()` logic.

- When using `addEvenListener` inside the `useEffect` function you should clear those event listeners in the return of that `useEffect`

```
// Inside the component
function clickEvent() {
  console.log('click');
}


useEffect(
  ()=>{
    window.addEventListener('click', clickEvent);

    return () => {
      window.removeEventListener('click', clickEvent);
    };
  },
  []
)
```

And notice that you can't use an anonymous function instead of `clickEvent` because you need to pass the same function with the same reference to the `removeEventListener` function.

You can apply the above to `setInterval`, `setTimeout` and `clearInterval`, `clearTimeout` functions.

- To make your `useEffect` function work only as component did mount you should pass an empty array as the second argument.

```
useEffect(() => {
  setName('Ahmed'); // infinte loop
});

useEffect(() => {
  setName('Ahmed');
}, []);
```