ReactJs

Nodejs 🡪nodejs and npm

1. App creation npx create-react-app my-react-app
   1. Single page application
   2. Everything is component
   3. React create virtual DOM
   4. Invented by facebook
   5. React is library
2. State vs props
3. Statefull component or class component

import React from 'react';

import ReactDOM from 'react-dom/client';

class Header extends React.Component {

constructor(props) {

super(props);

this.state = {favoritecolor: "red"};

}

static getDerivedStateFromProps(props, state) {

return {favoritecolor: props.favcol };

}

render() {

return (

<h1>My Favorite Color is {this.state.favoritecolor}</h1>

);

}

}

1. Stateless component or functional component

import React from 'react';

import ReactDOM from 'react-dom/client';

function Car(props) {

return <h2>I am a { props.brand }!</h2>;

}

const myElement = <Car brand="Ford" />;

1. React lifecycle

Mounting

1. constructor() 🡪componentWillMount
2. getDerivedStateFromProps() 🡪 ComponentWillreceiveProps
3. render()
4. componentDidMount()

Update

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

Unmounting

* componentWillUnmount()

What is PureComponent?

1. Routes

import ReactDOM from "react-dom/client";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Layout from "./pages/Layout";

import Home from "./pages/Home";

import Blogs from "./pages/Blogs";

import Contact from "./pages/Contact";

import NoPage from "./pages/NoPage";

export default function App() {

return (

<BrowserRouter>

<Routes>

<Route path="/" element={<Layout />}>

<Route index element={<Home />} />

<Route path="blogs" element={<Blogs />} />

<Route path="contact" element={<Contact />} />

<Route path="\*" element={<NoPage />} />

</Route>

</Routes>

</BrowserRouter>

);

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);

1. Hook
   1. useState
   2. useEffect
   3. useContext 05/07
   4. useRef 05/07
   5. useCallback 05/07
   6. useMemo 06/07
   7. custom hook 06/07
2. Jquery 07/07 and 08/07

2. Useeffect:

 fetching data, directly updating the DOM, and timers.

import { useState, useEffect } from "react";

import ReactDOM from "react-dom/client";

function Timer() {

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

useEffect(() => {

setTimeout(() => {

setCount((count) => count + 1);

}, 1000);

});

return <h1>I've rendered {count} times!</h1>;

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Timer />);

no dependency passed – render every time

useEffect(() => {

//Runs only on the first render

}, []);

empty arreay ---run only initial load\

import { useState, useEffect } from "react";

import ReactDOM from "react-dom/client";

function Counter() {

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

const [calculation, setCalculation] = useState(0);

useEffect(() => {

setCalculation(() => count \* 2);

}, [count]); // <- add the count variable here

return (

<>

<p>Count: {count}</p>

<button onClick={() => setCount((c) => c + 1)}>+</button>

<p>Calculation: {calculation}</p>

</>

);

}

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<Counter />);

2. Context :

React Context is a way to manage state globally.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

 we will need to pass the state as "props" through each nested component. This is called "prop drilling".

function Component1() {

const [user, setUser] = useState("Jesse Hall");

return (

<UserContext.Provider value={user}>

<h1>{`Hello ${user}!`}</h1>

<Component2 user={user} />

</UserContext.Provider>

);

}

import { useState, createContext, useContext } from "react";

function Component5() {

const user = useContext(UserContext);

return (

<>

<h1>Component 5</h1>

<h2>{`Hello ${user} again!`}</h2>

</>

);

}

UseRef:

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

If we tried to count how many times our application renders using the useState Hook, we would be caught in an infinite loop since this Hook itself causes a re-render.

To avoid this, we can use the useRef Hook.

import { useRef } from "react";

import ReactDOM from "react-dom/client";

function App() {

const inputElement = useRef();

const focusInput = () => {

inputElement.current.focus();

};

return (

<>

<input type="text" ref={inputElement} />

<button onClick={focusInput}>Focus Input</button>

</>

);

}

**UseMemo Hook and useCallback hook:**

Performance optimization

Usememo recalculate only cached value

Restrict to impact each element of page

Use calback changes only changes when one of the dependencies has chane

* 1. React -> render every time when state change
  2. React.Memo -> render only if dependent state get changes
  3. React hook – useCallBack -> render method when method dependent variable get change
  4. React hook – useMemo-> render when specified variable/state get changed.

**JQUERY:**

* Jquery is library and most of the Jquery feature are replaced by modern Javascript so if only required then only we look into Jquery

Some basic features of JQUery

* **AJAX**☹Replaced by fetch) Call API by asynchronously
* **SELECTORS**: select the element id, tag,or class
* **EFFECTS**: Hide, Show, Toggle, Slide, Fade, and Animate.
* **EVENTS**: The term **"fires/fired"** is often used with events. Example: "The keypress event is fired, the moment you press a key".Anything which perform on DOM or node of DOM.

noConflict() – replace the $ responsibility to other variable (any variable you can provide)

if you don’t want to use $ then you can use JQuery name

* **TRAVERSING:** parent()
* parents()
* parentsUntil() – inbetween two tag

$(document).ready(function(){  
  $("span").parentsUntil("div");  
});

* first(), last()
* siblings()
* next()
* nextAll()
* nextUntil()
* prev()
* prevAll()
* prevUntil()

SELECTOR:

<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script>

$(document).ready(function(){

$("button").click(function(){

$("p").hide();

});

});

</script>

</head>

<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>

<p>This is another paragraph.</p>

<button>Click me to hide paragraphs</button>

</body>

</html>

Traversing:

<!DOCTYPE html>

<html>

<head>

<style>

.ancestors \* {

display: block;

border: 2px solid lightgrey;

color: lightgrey;

padding: 5px;

margin: 15px;

}

</style>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>

<script>

$(document).ready(function(){

$("span").parent().css({"color": "red", "border": "2px solid red"});

});

</script>

</head>

<body>

<div class="ancestors">

<div style="width:500px;">div (great-grandparent)

<ul>ul (grandparent)

<li>li (direct parent)

<span>span</span>

</li>

</ul>

</div>

<div style="width:500px;">div (grandparent)

<p>p (direct parent)

<span>span</span>

</p>

</div>

</div>

</body>

</html>

react 16.8

1. State hook

Hook state is the new way of declaring a state in React app. Hook uses useState() functional component for setting and retrieving state.

. The useState is similar to this.setState in class

2. Effect Hook

The Effect Hook allows us to perform side effects (an action) in the function components. It does not use components lifecycle methods which are available in class components. In other words,

Effects Hooks are equivalent to componentDidMount(), componentDidUpdate(), and componentWillUnmount() lifecycle methods.

Side effects have common features which the most web applications need to perform, such as:

Updating the DOM,

Fetching and consuming data from a server API,

3. Context Hook

React Context is a way to manage state globally.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

4. Ref Hook

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.

The useRef Hook allows you to persist values between renders.

It can be used to store a mutable value that does not cause a re-render when updated.

It can be used to access a DOM element directly.