هوکهای ری‌اکت و نکست‌جی‌اس

مقدمه

از کتابخانه‌ها و فریمورکهای محبوب جاوااسکریپت برای ساخت اپلیکیشن‌های تحت (Next.js) و نکست‌جی‌اس (React) ری‌اکت در ری‌اکت امکانات قدرتمندی برای مدیریت وضعیت و چرخه زندگی کامپونت‌ها ارائه می‌دهند .در اینجا (Hooks) وب هستند .هوکها .به معرفی تمام هوکهای اصلی ری‌اکت و نکست‌جی‌اس به همراه مثال و توضیحات می‌پردازیم

هوکهای ری‌اکت

useState

یک جفت مقدار برمی‌گرداند :مقدار کنونی useState .یک کامپونت استفاده می‌شود (local state) برای مدیریت وضعیت محلی .برای مدیریت چندین وضعیت محلی در یک کامپونت استفاده کرد useState وضعیت و تابعی برای به‌روزرسانی آن .می‌تواز چندین

```javascript
import React, { useState } from 'react';

function Counter() {
 const [count, setCount] = useState(۰);

 return (
  <div>
   <p>Count: {count}</p>
   <button onClick={() => setCount(count + ۱)}>Increment</button>
  </div>
 );
}
```

useEffect

تابعی است که بعد از useEffect .استفاده می‌شود DOM تایمرها و دسترسی به ،API برای مدیریت اثرات جانبی مانند درخواست‌های رندر کردن کامپونت اجرا می‌شود .می‌تواز از آرایه‌ای به‌عنواز دومین آرگوماز استفاده کرد تا مشخص شود چه زمانی این اثر جانبی باید اجرا .شود)وابستگی‌ها(

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

function DataFetcher() {
 const [data, setData] = useState(null);

 useEffect(() => {
  fetch('https://api.example.com/data')
   .then(response => response.json())
   .then(data => setData(data));
```

```
}, []); // این آرایه‌ی خالی یعنی فقط یک بار بعد از اولین رندر اجرا شود.

 return <div>{data ? data : 'Loading...'}</div>;
}
```

useContext

برای به‌اشتراک‌گذاری داده‌ها بین کامپوننت‌های مختلف، Context های ری‌اکت استفاده می‌شود. Context برای استفاده از مقادیر در بدون نیاز به ارسال پراپ‌ها در هر سطح، کاربرد دارد.

```javascript
import React, { useContext, createContext } from 'react';

const MyContext = createContext();

function MyComponent() {
 const value = useContext(MyContext);

 return <div>{value}</div>;
}

function App() {
 return (
  <MyContext.Provider value="Hello, World!">
   <MyComponent />
  </MyContext.Provider>
 );
}
```

useReducer

مشابه یک ماشین useReducer. برای مدیریت وضعیت‌های پیچیده‌تر که شامل چندین اکشن و حالت مختلف است استفاده می‌شود. برای وضعیت‌های useState، عمل می‌کند که با استفاده از اکشن‌ها وضعیت را تغییر می‌دهد. در مقایسه با حالت (state machine) پیچیده‌تر و وابسته به هم مناسب‌تر است.

```javascript
import React, { useReducer } from 'react';

const initialState = { count: ۰ };

function reducer(state, action) {
 switch (action.type) {
  case 'increment':
   return { count: state.count + ۱ };
  case 'decrement':
   return { count: state.count - ۱ };
  default:
```

```
    throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
}
```

useMemo

مقداری را فقط زمانی محاسبه می‌کند که یکی از useMemo .برای محاسبات پرهزینه که باید در رندر مجدد حفظ شوند، استفاده می‌شود .وابستگی‌ها تغییر کند

```javascript
import React, { useMemo, useState } from 'react';

function ExpensiveComponent({ number }) {
  const computeExpensiveValue = (num) => {
    console.log('Computing...');
    // شبیه‌سازی محاسبات سنگین
    return num * ۲;
  };

  const expensiveValue = useMemo(() => computeExpensiveValue(number), [number]);

  return <div>{expensiveValue}</div>;
}

function App() {
  const [count, setCount] = useState(۰);

  return (
    <div>
      <ExpensiveComponent number={count} />
      <button onClick={() => setCount(count + ۱)}>Increment</button>
    </div>
  );
}
```

useCallback

.برای حفظ مراجع توابع بین رندرها استفاده می‌شود، به خصوص زمانی که این توابع به عنوان پراپ به کامپونت‌های فرزند ارسال می‌شوند

```javascript
import React, { useCallback, useState } from 'react';

function Child({ onClick }) {
 return <button onClick={onClick}>Click me</button>;
}

function Parent() {
 const [count, setCount] = useState(۰);

 const handleClick = useCallback(() => {
  setCount(count + ۱);
 }, [count]);

 return (
  <div>
   <Child onClick={handleClick} />
   <p>Count: {count}</p>
  </div>
 );
}
```

useRef

.و ذخیره مقادیر که باعث رندر مجدد نمی‌شوند، استفاده می‌شود DOM برای دسترسی به عناصر

```javascript
import React, { useRef } from 'react';

function TextInputWithFocusButton() {
 const inputEl = useRef(null);

 const onButtonClick = () => {
  // با استفاده از useRef به عنصر DOM دسترسی پیدا می‌کنیم
  inputEl.current.focus();
 };

 return (
  <div>
   <input ref={inputEl} type="text" />
   <button onClick={onButtonClick}>Focus the input</button>
  </div>
 );
```

}
```

## useLayoutEffect

انجام می‌شود .به‌طور معمول برای اثرات جانبی که نیاز به اندازه‌گیری یا تغییرات DOM اما بعد از همه تغییرات ،useEffect شبیه به .دارند استفاده می‌شود DOM

```javascript
import React, { useLayoutEffect, useRef } from 'react';

function LayoutEffectExample() {
 const divRef = useRef(null);

 useLayoutEffect(() => {
  console.log(divRef.current.getBoundingClientRect());
 }, []);

 return <div ref={divRef}>Hello, World!</div>;
}
```

useImperativeHandle

های کامپوننت‌های API برای ایجاد forwardRef استفاده می‌شود .همراه با ref برای سفارشی‌سازی مقادیر قابل دسترس از طریق .سفارشی استفاده می‌شود

```javascript
import React, { useImperativeHandle, forwardRef, useRef } from 'react';

const FancyInput = forwardRef((props, ref) => {
 const inputRef = useRef();

 useImperativeHandle(ref, () => ({
  focus: () => {
   inputRef.current.focus();
  }
 }));

 return <input ref={inputRef} />;
});

function Parent() {
 const ref = useRef();

 return (
  <div>
   <FancyInput ref={ref} />
   <button onClick={() => ref.current.focus()}>Focus the input</button>
```
```

```
  </div>
 );
}
```

هوک‌های نکست‌جی‌اس

useRouter

در کامپوننت‌های فانکشنال نکست‌جی‌اس استفاده می‌شود .این هوک امکاناتی مانند پیمایش (router) برای دسترسی به شیء مسیریاب .و دسترسی به پارامترهای مسیر را فراهم می‌کند (programmatic navigation) برنامه‌ای

```javascript
import { useRouter } from 'next/router';

function MyComponent() {
 const router = useRouter();

 const goToHome = () => {
  router.push('/');
 };

 return <button onClick={goToHome}>Go to Home</button>;
}
```

useSWR

داده‌ها است که توسط تیم fetching یک کتابخانه برای SWR .برای مدیریت درخواست‌های داده و کش کردن نتایج استفاده می‌شود .ورسل توسعه داده شده است و با نکست‌جی‌اس به خوبی سازگار است

```javascript
import useSWR from 'swr';

function Fetcher() {
 const fetcher = (url) => fetch(url).then((res) => res.json());
 const { data, error } = useSWR('/api/data', fetcher);

 if (error) return <div>Failed to load</div>;
 if (!data) return <div>Loading...</div>;

 return <div>{data.message}</div>;
}
```