React 前端开发

第四讲 Effect、Context、其他补充内容与常用库

PDLi

2023 年 12 月 17 日

目录

Effect

Effect I

在理想状态下，我们希望 React 组件的渲染结果只与 props 和 state 相关，但是实际上，我们经常需要在组件渲染的时候执行一些额外的操作，比如获取数据、订阅事件等。这些操作被称为 side effect(副作用)，React 为我们提供了 useEffect Hook 来处理这些 side effect。

Effect II

useEffect

useEffect 接收一个函数作为参数，这个函数就是我们需要执行的副作用。在组件渲染的时候，React 会保存这个函数，然后在组件渲染完成后执行这个函数。如果我们需要在组件卸载的时候执行一些清理操作，我们可以在这个函数中返回一个函数，React 会在组件卸载的时候执行这个函数。useEffect 接受一个可选的第二个参数，这个参数是一个数组，数组中的每个元素都是一个依赖项。如果依赖项发生了变化，React 会重新执行这个副作用函数。如果没有传入依赖项，那么每次组件渲染的时候都会执行这个副作用函数。

Effect III

示例代码：

```
 1 import React, { useState, useEffect } from 'react';
 2
 3 function Example() {
 4   const [count, setCount] = useState(0);
 5   const [data, setData] = useState('');
 6
 7   useEffect(() => {
 8     // 使用 setTimeout 模拟异步请求
 9     setTimeout(() => {
10       setData('111111');
11     }, 1000);
12   }, []);
13
14   return (
15     <div>
16       <p>You clicked {count} times</p>
17       <h1>{data}</h1>
```

Effect IV

```
18         <button onClick={() => setCount(count + 1)}>
19           Click me
20         </button>
21       </div>
22     );
23   }
```

设置依赖项：

```
 1  import React, { useState, useEffect } from 'react';
 2
 3  function Example() {
 4    const [count, setCount] = useState(0);
 5
 6    useEffect(() => {
 7      document.title = `You clicked ${count} times`;
 8    }, [count]);
 9
10    return (
```

Effect V

```
11      <div>
12        <p>You clicked {count} times</p>
13        <button onClick={() => setCount(count + 1)}>
14          Click me
15        </button>
16      </div>
17    );
18  }
```

设置定时器：

```
1  import React, { useState, useEffect } from 'react';
2
3  function Example() {
4    const [count, setCount] = useState(0);
5
6    useEffect(() => {
7      const timer = setInterval(() => {
8        setCount(count + 1);
```

Effect VI

```
 9       }, 1000);
10       return () => clearInterval(timer);
11     }, [count]);
12
13     return (
14       <div>
15         <p>You clicked {count} times</p>
16       </div>
17     );
18   }
```

Effect VII

useEffect 的执行时机

- React 会在每次渲染后调用副作用函数，包括第一次渲染的时候。
- 之后每次渲染前，React 会先清除上一次渲染时的副作用函数。
- React 会在组件卸载的时候执行副作用函数，清除副作用。

依赖项的设置

- 如果没有传入依赖项，那么每次组件渲染的时候都会执行这个副作用函数。
- 如果传入了空数组，那么副作用函数只会在组件渲染的时候执行一次，之后不会再执行。
- 如果传入了非空数组，那么副作用函数会在组件渲染的时候执行一次，之后只有当依赖项发生变化的时候才会执行。

Context

Context I

Context 提供了一种在组件之间共享值的方式，而不必通过组件树的逐层传递 props。在一些场景下，这种做法会使得组件的传递变得很复杂，Context 可以帮助我们解决这个问题。

Context II

Context 的使用

- 首先，我们需要创建一个 Context 对象，这个对象包含一个 Provider 组件和一个 Consumer 组件。
- 然后，我们需要在 Provider 组件中传入一个值，这个值可以是任意类型的数据。
- 最后，我们可以在 Consumer 组件中获取到 Provider 组件中传入的值。

Context III

示例代码：

```
 1 import React, { useState, useEffect, createContext,
       useContext } from 'react';
 2
 3 const CountContext = createContext();
 4
 5 function Counter() {
 6   const count = useContext(CountContext);
 7   return <h2>{count}</h2>;
 8 }
 9
10 function Example() {
11   const [count, setCount] = useState(0);
12
13   useEffect(() => {
14     const timer = setInterval(() => {
15       setCount(count + 1);
16     }, 1000);
```

Context IV

```
17      return () => clearInterval(timer);
18    }, [count]);
19
20    return (
21      <div>
22        <p>You clicked {count} times</p>
23        <CountContext.Provider value={count}>
24          <Counter />
25        </CountContext.Provider>
26      </div>
27    );
28  }
```

其他补充内容

在 JSX 中使用 if 语句 I

if 语句

JSX 本身并不支持 if 语句，但是我们可以在 JSX 中使用三元运算符或 && 来实现 if 语句的功能。

在 JSX 中使用 if 语句 II

示例代码:

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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      {count % 2 === 0 ? <h1>偶数</h1> : <h1>奇数</h1>}
      {count > 5 && <h1>大于5</h1>}
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

在 JSX 中使用 for 循环 I

for 循环

JSX 本身并不支持 for 循环，但是我们可以在 JSX 中使用 map 来实现 for 循环的功能。

在 JSX 中使用 for 循环 II

示例代码：

```
 1 import React, { useState } from 'react';
 2
 3 function Example() {
 4   const [count, setCount] = useState(0);
 5
 6   return (
 7     <div>
 8       <p>You clicked {count} times</p>
 9       {[1, 2, 3, 4, 5].map((item) => {
10         return <h1 key={item}>{item}</h1>;
11       })}
12       <button onClick={() => setCount(count + 1)}>
13         Click me
14       </button>
15     </div>
16   );
17 }
```

在 JSX 中使用 for 循环 III

key 属性

在使用 map 的时候，我们需要为每个元素添加一个 key 属性，这个属性的值应该是唯一的，这样 React 才能够正确地识别每个元素。key 属性的作用是帮助 React 识别哪些元素发生了变化，从而减少 DOM 操作的次数，提高性能。

受控组件与非受控组件 I

受控组件

受控组件是指表单元素的值由 React 组件来控制的组件，比如下面的代码：

```
 1 import React, { useState } from 'react';
 2
 3 function Example() {
 4   const [value, setValue] = useState('');
 5
 6   return (
 7     <div>
 8       <input value={value} onChange={(e) => setValue(e.
             target.value)} />
 9     </div>
10   );
11 }
```

受控组件与非受控组件 II

非受控组件

非受控组件是指表单元素的值由 DOM 元素来控制的组件，比如下面的代码：

```
 1 import React, { useRef } from 'react';
 2
 3 function Example() {
 4   const inputRef = useRef(null);
 5
 6   return (
 7     <div>
 8       <input ref={inputRef} />
 9     </div>
10   );
11 }
```

受控组件与非受控组件 III

受控组件与非受控组件

- 受控组件的值由 React 组件来控制，所以我们可以通过 props 来控制表单元素的值。
- 非受控组件的值由 DOM 元素来控制，所以我们只能通过 DOM 元素的方法来控制表单元素的值。

使用 受控组件

在一般情况下，我们应该尽量使用受控组件，因为这样可以让表单元素的值与 React 组件的状态保持一致，从而减少出错的可能性。

常用库

React Router 简介与安装 I

React Router

React Router 是一个用于 React 的路由库，它可以帮助我们实现页面之间的跳转。

通过 React Router，我们可以在 SPA 应用中实现页面之间的跳转，而不需要刷新页面。

React Router 简介与安装 II

安装

```
1 npm install react-router-dom
```

ReactRouter 的使用 I

ReactRouter 的使用

- 首先，我们需要在 App.js 中导入 BrowserRouter 组件，然后将 App 组件包裹在 BrowserRouter 组件中。
- 然后，我们需要在 App.js 中导入 Routes 组件，然后在 BrowserRouter 组件中添加 Routes 组件。
- 最后，我们需要在 Routes 组件中添加 Route 组件，其中 path 属性表示路由的路径，element 属性表示路由对应的组件。

ReactRouter 的使用 II

示例代码：

```
 1 import React from 'react';
 2 import { BrowserRouter, Routes, Route } from 'react-
       router-dom';
 3 import Home from './Home';
 4 import About from './About';
 5
 6 function App() {
 7   return (
 8     <BrowserRouter>
 9       <Routes>
10         <Route path="/" element={<Home />} />
11         <Route path="/about" element={<About />} />
12       </Routes>
13     </BrowserRouter>
14   );
15 }
16
```

ReactRouter 的使用 III

```
17 export default App;
```

ReactRouter 的使用 IV

Route 组件的属性

- path 属性表示路由的路径。
- element 属性表示路由对应的组件。
- caseSensitive 属性表示是否区分大小写。
- index 属性表示索引路由。
- children 属性表示嵌套路由。

ReactRouter 的使用 V

caseSensitive 属性

caseSensitive 属性表示是否区分大小写，如果设置了 caseSensitive 属性，那么只有当路径大小写完全匹配的时候才会渲染对应的组件。

示例代码：

```
 1 import React from 'react';
 2 import { BrowserRouter, Routes, Route } from 'react-
       router-dom';
 3 import Home from './Home';
 4 import About from './About';
 5
 6 function App() {
 7   return (
 8     <BrowserRouter>
 9       <Routes>
10         <Route path="/" element={<Home />} />
```

ReactRouter 的使用 VI

```
11          <Route path="/about" element={<About />} />
12          <Route path="/About" element={<About />}
                caseSensitive />
13        </Routes>
14      </BrowserRouter>
15    );
16  }
17
18  export default App;
```

ReactRouter 的使用 VII

index 属性

index 属性表示索引路由，如果设置了 index 属性，那么这个组件将被渲染到父级路由的 Outlet 中。

嵌套路由

我们可以在 Route 组件下添加子组件来实现嵌套路由。

示例代码：

```
1 import React from 'react';
2 import { BrowserRouter, Routes, Route } from 'react-
      router-dom';
3 import Home from './Home';
4 import About from './About';
5 import AboutMe from './AboutMe';
6 import AboutYou from './AboutYou';
7
```

ReactRouter 的使用 VIII

```
 8 function App() {
 9   return (
10     <BrowserRouter>
11       <Routes>
12         <Route path="/" element={<Home />} />
13         <Route path="/about" element={<About />}>
14           <Route path="me" element={<AboutMe />} />
15           <Route path="you" element={<AboutYou />} />
16         </Route>
17       </Routes>
18     </BrowserRouter>
19   );
20 }
21
22 export default App;
```

路由表 I

路由表

我们可以将路由表单独抽离出来，然后在 App.js 中导入路由表。并使用 useRoutes 函数来渲染路由表。

示例代码：

```
 1 import React from 'react';
 2 import { BrowserRouter as Router, Route, useRoutes }
       from 'react-router-dom';
 3 import Home from './Home';
 4 import About from './About';
 5 import AboutMe from './AboutMe';
 6 import AboutYou from './AboutYou';
 7
 8 const routes = [
 9   {
10     path: '/',
11     element: <Home />,
```

路由表 II

```
12    },
13    {
14      path: '/about',
15      element: <About />,
16    },
17    {
18      path: '/about/me',
19      element: <AboutMe />,
20    },
21    {
22      path: '/about/you',
23      element: <AboutYou />,
24    },
25  ];
26
27  function App() {
28    const routing = useRoutes(routes);
29
30    return (
```

路由表 III

```
31      <Router>
32        {routing}
33      </Router>
34    );
35  }
36
37  export default App;
```

路由参数 I

路由参数

我们可以在路由中添加参数，然后在组件中通过 useParams 或 useSearchParams 函数来获取路由参数。

示例代码：

```
 1 import React from 'react';
 2 import { BrowserRouter as Router, Route, useRoutes,
       useParams } from 'react-router-dom';
 3 import Home from './Home';
 4 import About from './About';
 5 import AboutMe from './AboutMe';
 6 import AboutYou from './AboutYou';
 7
 8 const routes = [
 9   {
10     path: '/',
```

路由参数 II

```
11      element: <Home />,
12    },
13    {
14      path: '/about',
15      element: <About />,
16    },
17    {
18      path: '/about/me',
19      element: <AboutMe />,
20    },
21    {
22      path: '/about/you',
23      element: <AboutYou />,
24    },
25    {
26      path: '/about/:name',
27      element: <AboutName />,
28    },
29  ];
```

路由参数 III

```
30
31 function AboutName() {
32   const { name } = useParams();
33   return <h1>{name}</h1>;
34 }
35
36 function App() {
37   const routing = useRoutes(routes);
38
39   return (
40     <Router>
41       {routing}
42     </Router>
43   );
44 }
45
46 export default App;
```

search 参数：

路由参数 IV

```
 1 import React from 'react';
 2 import { BrowserRouter as Router, Route, useRoutes,
       useSearchParams } from 'react-router-dom';
 3 import Home from './Home';
 4 import SearchResults from './SearchResults';
 5
 6 const routes = [
 7   {
 8     path: '/',
 9     element: <Home />,
10   },
11   {
12     path: '/search',
13     element: <SearchResults />,
14   },
15 ];
16
17 function SearchResults() {
```

路由参数 V

```
18    const [searchParams] = useSearchParams();
19    const query = searchParams.get('query');
20
21    return <h1>Search Results for: {query}</h1>;
22  }
23
24  function App() {
25    const routing = useRoutes(routes);
26
27    return (
28      <Router>
29        {routing}
30      </Router>
31    );
32  }
33
34  export default App;
```

Link 组件 I

Link 组件

Link 组件可以帮助我们实现页面之间的跳转。

示例代码：

```
 1 import React from 'react';
 2 import { BrowserRouter as Router, Route, useRoutes, Link
       } from 'react-router-dom';
 3 import Home from './Home';
 4 import About from './About';
 5 import AboutMe from './AboutMe';
 6 import AboutYou from './AboutYou';
 7
 8 const routes = [
 9   {
10     path: '/',
11     element: <Home />,
12   },
```

Link 组件 II

```
13    {
14      path: '/about',
15      element: <About />,
16    },
17    {
18      path: '/about/me',
19      element: <AboutMe />,
20    },
21    {
22      path: '/about/you',
23      element: <AboutYou />,
24    },
25    {
26      path: '/about/:name',
27      element: <AboutName />,
28    },
29  ];
30
31  function AboutName() {
```

Link 组件 III

```
32    const { name } = useParams();
33    return <h1>{name}</h1>;
34  }
35
36  function App() {
37    const routing = useRoutes(routes);
38
39    return (
40      <Router>
41        <nav>
42          <Link to="/">Home</Link>
43          <Link to="/about">About</Link>
44          <Link to="/about/me">About Me</Link>
45          <Link to="/about/you">About You</Link>
46        </nav>
47        {routing}
48      </Router>
49    );
50  }
```

Link 组件 IV

```
51
52 export default App;
```