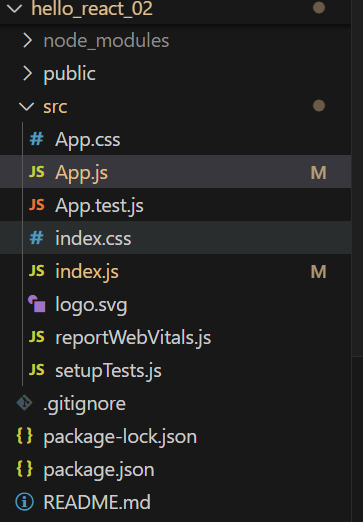
npx create-react-app XXX 创建一个新的react-app

npm start 启动服务

npm build 打包

package.json 依赖 react版本

index.js 是项目的入口

import React from 'react';

import ReactDOM from 'react-dom/client';

是React的两个核心包

App.js为项目的根组件

import App from './App';

渲染流程为：App.js渲染到index.js，然后index.js把内容渲染到public中的index.html

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

root.render(

  <App />

);

将App渲染到id为root的根组件上。

JSX为Javascript和HTML的缩写，表示在JS代码中编写HTML模板结构

return (

    <div className="App">

      This is a react app.

    </div>

  );

Return中的就是JSX，优势1. HTML声明式模板 2. Javascript的可编程能力

JSX并不是标准的JS语法，是JS的语法扩展，需要特定的解析工具解析运行。通过{}识别Javascript的表达式，比如常见的变量，函数调用和方法调用。

1. 使用引号传递字符串

2. 使用Javascript变量

3. 函数调用和方法调用

4. 使用Javascript对象

{/\* 使用引号传递字符串 \*/}

{"This is a message."}

{/\* 使用Javascript变量 \*/}

{"Count is " + count}

{/\* 函数调用 \*/}

{getCount()}

{/\* 方法调用 \*/}

{new Date().getDate()}

{/\* 使用Javascript对象 \*/}

<div style={{color: "blue"}}>This is a div</div>

JSX实现列表渲染

需要使用原生Javascript中的map方法遍历渲染 注意需要key值

const list = [

  {id: 1001, name: "Vue"},

  {id: 1002, name: "React"},

  {id: 1003, name: "Angular"}

]

{/\* 渲染列表 \*/}

      <ul>

        {list.map(item => <li key={item.id}>{item.name}</li>)}

      </ul>

JSX实现条件渲染

通过逻辑运算符&&或者三元表达式实现

{/\* 条件渲染 \*/}

{/\* 逻辑与 && \*/}

{isLogin && <span>Please Login in</span>}

{/\* 三元运算符 \*/}

{isLogin ? <span>Username</span> : <span>Please Login in</span>}

JSX实现复杂条件渲染

通过自定义函数+if语句

const flag = 3;

// 定义函数，根据flag返回模板

function getFlagTemplate(){

  if(flag === 0){

    return <div>flag = 0</div>;

  } else if(flag === 1){

    return <div>flag = 1</div>;

  } else {

    return <div>flag = 3</div>;

  }

}

{/\* 复杂情况渲染 \*/}

{getFlagTemplate()}

React进行事件绑定

on + 事件名称 = {事件处理程序} 小驼峰

const clickButton = () => {

  console.log("click");

}

<button onClick={clickButton}>Click</button>

使用事件对象参数

通过设置形参e实现

// 事件参数

const clickButton1 = (e) => {

  console.log("click", e);

}

{/\* 获得对象参数 \*/}

<button onClick={clickButton1}>Click1</button>

传递自定义参数

通过箭头函数实现，事件需要绑定一个函数引用

// 传递自定义参数

const clickButton2 = (id) => {

  console.log("click", id++);

}

{/\* 传递自定义参数 \*/}

<button onClick={() => clickButton2(0)}>Click2</button>

同时传递事件对象和自定义参数

改造两处

// 传递事件参数和自定义参数

const clickButton3 = (id, e) => {

  console.log("click", ++id, e);

}

{/\* 传递事件参数和自定义参数 \*/}

<button onClick={(e) => clickButton3(0, e)}>Click3</button>