第 3 章

React 快速入门

3.1 React 简介

前端技术日新月异，尤其 Web 前端技术的能力和应用领域不断增多，Web 前端开发工作的广度和深度也随之日益提升，这就要求 Web 前端工程师必须扩展自己的知识技能体系。

React 是一款由 Facebook 开发并开源的 JavaScript 库，主要用于帮助开发者构建高质量的用户界面，特别是单页面应用。相比 Vue 和 Angular 等前端框架，React 具备组件化设计、虚拟 DOM 和单向数据流等特性，说明如下。

- 声明式编程：开发者只需要声明需要渲染的内容，React 会自动处理页面的更新。
- 组件化设计：将页面拆分成多个独立的组件，每个组件拥有自己的状态和生命周期，可以被复用和组合。
- 虚拟 DOM：React 使用虚拟 DOM 来提高性能和操作效率，可以在不重新渲染整个页面的情况下更新特定部分的内容。
- 单向数据流：React 采用单向数据流的设计思想，父组件可以通过 props 属性将数据传递给子组件，子组件通过回调函数将数据回传给父组件。
- 高性能：React 采用虚拟 DOM 和优化算法等方式来提高性能和操作效率。

React 拥有广泛的资源和社区支持，可以帮助开发者更快更好地学习和使用 React。总的来说，无论是构建单页应用还是搭建复杂的前端项目，React 都因其简洁的设计和强大的生态系统成为现代前端开发的首选工具之一。

3.2 虚拟 DOM

React 框架的主旨是推动简化复杂的任务并把不必要的复杂实现从开发人员身上抽离出来。React 试图提升前端的性能，从而让研发人员更专注于应用业务的开发，React 鼓励开发人员使用声明式编程而不是命令式编程。React 要求开发者通过“声明组件在不同状态下的外观和行为”来构建 UI，而非直接操作 DOM，这也是 React 相比传统命令式编程更

加高效的核心原因。

而驱动这些的主要技术之一就是虚拟 DOM(Virtual DOM,VDOM)。虚拟 DOM 是一种编程概念,开发者通过操作虚拟 DOM,React 会将其与真实 DOM 进行比较,并通过高效的算法计算出最小的更新操作,从而减少页面重绘和重新布局的开销。

在 React 的虚拟 DOM 工作原理中,React 组件在执行首次渲染时,会根据 UI 组件生成一棵虚拟 DOM 树,该树的结构与真实的 DOM 结构一一对应。当组件的状态或属性发生变化时,React 会重新创建一个新的虚拟 DOM 树,新的虚拟 DOM 树与旧的虚拟 DOM 树进行比较,并找出两者之间的差异。在计算出差异后,React 会将必要的更新操作应用到真实 DOM 上,整个流程如图 3-1 所示。

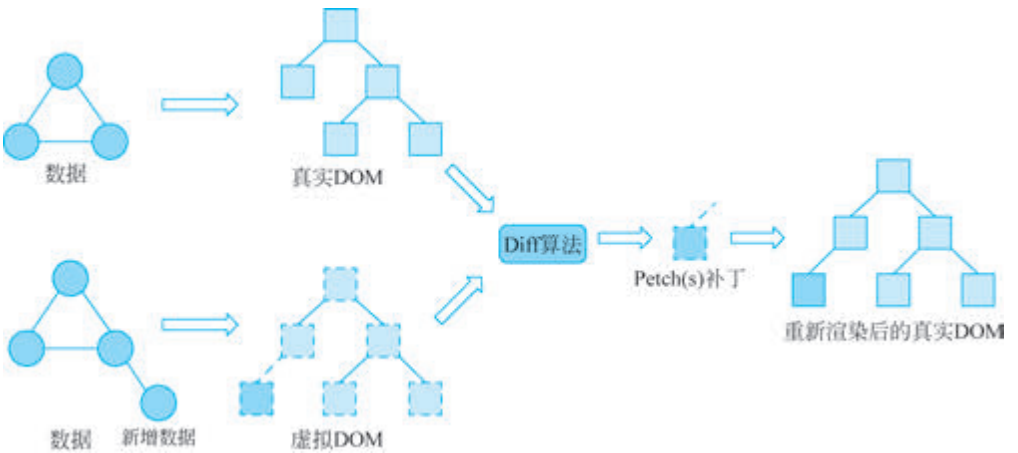

图 3-1 虚拟 DOM 工作原理示意图

在 React 框架中,虚拟 DOM 的实现离不开 JSX 与 React.createElement 函数的配合。事实上,在 React 的渲染流程中,每个 JSX 元素都会转化为一个包含节点类型、节点属性、子节点等信息的虚拟 DOM 对象,虚拟 DOM 的优化则主要体现在 Diffing 算法和更新策略上。

对于类组件来说,可以使用 shouldComponentUpdate 生命周期方法来控制组件是否需要重新渲染。而对于函数组件来说,则可以使用 React.memo 来避免不必要的重新渲染。

总的来说,虚拟 DOM 的出现是为了高效地比较和更新 UI,减少对真实 DOM 的直接操作,从而提高页面性能。不过,虚拟 DOM 也引入了一些额外的开销,如需要额外的内存来存储虚拟 DOM 树和 DOM 树的差异比较。此外,对于一些对性能要求极高的应用(如游戏或需要精细渲染的图形应用),虚拟 DOM 的 Diffing 操作和更新机制可能无法满足需求。

3.3 JSX 语法基础

3.3.1 JSX 简介

JSX(JavaScript XML)并不算是一门编程语言,而是 JavaScript 的一种语法扩展,一种

可以在 JavaScript 代码中使用 HTML 元素来编写 JavaScript 对象的语法糖，所以 JSX 本质上来说还是 JavaScript。JSX 语法的出现，使得组件的结构和逻辑更加直观，目前绝大多数 React 和 React Native 应用都使用 JSX 进行开发。

当然，开发 React 和 React Native 应用程序不一定非要使用 JSX 语法，也可以继续使用 JavaScript 进行开发。不过，因为 JSX 在定义上类似 HTML 这种树形结构，所以使用 JSX 可以极大地提高阅读和开发效率，减少代码维护的成本。例如，下面是使用 typescript 模版创建的 React 示例项目的主页面代码。

```
function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.tsx</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer">
          Learn React
        </a>
      </header>
    </div>
  );
}
export default App;
```

上面的代码使用 JSX 进行编写，其中 return 方法返回的就是需要渲染的视图对象。之所以没有看到前端开发中创建对象和设置属性的代码，是因为 JSX 提供的 JSXTransformer 可以把代码中的 XML-Like 语法编译转换成 JavaScript 代码，进而被语言解析器所识别。

JSX 语法作为 JavaScript 的一种语法扩展，不仅可以帮助开发者创建视图对象、样式和布局，还可以用它构建视图的树形结构。更重要的是，JSX 语法的可读性、可维护性非常好，非常适合用来开发前端页面。

3.3.2 嵌入表达式

JSX 可以在标签内部嵌入 JavaScript 表达式，通过使用大括号{}将表达式包裹在标签内部，实现动态 UI 渲染效果。例如，我们声明了一个名为 name 的变量，然后可以在 JSX 中使用它。

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

事实上，可以在大括号内放置任何有效的 JavaScript 表达式。例如，在下面的示例中，我们将调用 JavaScript 函数 formatName(user)方法的运行结果，并将结果嵌入< h1 >元素中。

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  < h1 > Hello, {formatName(user)} </h1 >
);
```

3.3.3 绑定属性

在 JSX 语法中，我们可以使用引号来将属性值指定为字符串字面量，或者使用大括号在属性值中插入一个 JavaScript 表达式，如下所示。

```
const element = < a href = "https://www.reactjs.org"> link </a >;
const element = < img src = {user.avatarUrl}></img >;
```

在使用 JSX 语法开发 React 页面的过程中，当给元素绑定 style 时，外层的大括号表示可传入变量或者表达式，而内部的大括号是一个对象，表示元素的样式属性及属性值。

```
const MyComponent = () => {
  return (
    < div >
      < span className = {["tag", "span"].join(" ")}> span 标签</span >
      < span style = {{ color: "red", fontSize: 16 }}>红色颜色的字体</span >
    </div >
  );
};
```

需要说明的是，当 React 元素的属性由多个单词组成时，需要使用驼峰命名法进行表示。

3.3.4 绑定事件

React 元素的事件处理和 DOM 元素的事件处理是很相似的，处理 React 元素的事件时，需要传入一个函数作为事件处理函数，事件的命名采用驼峰式，如下所示。

```
const MyComponent = () => {
  const onClick = () => {
    window.alert("您单击了按钮 1");
```

```
  };

  return (
    <div>
      <button onClick={onClick}>按钮 1</button>
      <button
        onClick={() => {
          window.alert("您单击了按钮 2");
        }}>
        按钮 2
      </button>
    </div>
  );
};
```

如果函数过于简单,可以直接在 JSX 中直接编写函数的内容。

3.3.5 条件渲染

JSX 扩展了 JavaScript 的功能,从而可以直接使用 JSX 实现条件渲染。JSX 的条件渲染支持常见的条件判断语句、三元运算符和与运算符 &&,如下所示。

```
const MyComponent = () => {
  const renderTitle = (key) => {
    if (key > 1) {
      return <span>标题 1</span>;
    }
    return <span>标题 2</span>;
  };

  return (
    <div>
      {renderTitle(2)}
      {2 > 1 ? <span>显示</span> : <span>隐藏</span>}
      {2 > 1 && <span>显示</span>}
    </div>
  );
};
```

3.4 开发工具

目前支持 React 项目开发的工具有很多,常见的有 Visual Studio Code、Sublime Text、WebStorm 和 Eclipse 等,推荐使用 Visual Studio Code(简称 VS Code)。

VS Code 是微软于 2015 年发布的一款可以运行在 macOS、Windows 和 Linux 操作系统之上的跨平台源代码编辑器,支持主流的 C++、C#、Python、PHP 和 Dart 等开发语言,同

时还内置了对 JavaScript、TypeScript 和 Node.js 的支持，是一款真正轻量且强大的代码开发工具。如果还没有安装 VS Code，可以从微软官网下载并安装 VS Code，如图 3-2 所示。

图 3-2　下载 VS Code

安装完成之后，双击桌面快捷图标打开 VS Code，其主页面如图 3-3 所示。

图 3-3　VS Code 主页面

可以看到，VS Code 的主页面由活动栏、侧边栏、代码编辑区和控制台等部分组成。在平时的项目开发过程中，使用最多的就是侧边栏、代码编辑区和控制台。在 React 项目开发中，为了方便运行与调试 TypeScript 代码，我们需要全局安装一个 ts-node 的软件包，安装

命令如下。

```
npm install -g ts-node
```

事实上，使用集成开发工具，不仅可以在开发过程中对语法进行校验，减少程序可能的语法错误，还能在保存后立即执行编译，提高程序的开发效率。

除了 VS Code 外，WebStorm 因其强大的功能也被很多的开发者用来开发 React 项目。WebStorm 是 JetBrains 公司旗下一款 JavaScript 开发工具，大家熟知的后端开发工具 IntelliJ IDEA 就是该公司的产品。WebStorm 继承了 IntelliJ IDEA 强大的功能，被广大开发者誉为 Web 前端开发神器。至于选择哪款开发工具，大家可以根据自己的习惯和喜好进行选择。

3.5 项目示例

3.5.1 安装 Node.js

由于 React 项目的运行需要 JavaScript 环境的支持，所以在开发 React 项目之前需要先安装 Node.js。Node.js 本身并不属于任何一门开发语言，也不属于任何的 JavaScript 框架，它是 Ryan Dahl 开发的一款基于 Google Chrome V8 引擎的 JavaScript 运行环境。

如果还没有安装 Node.js，可以从 Node.js 官网下载对应操作系统的安装包并进行安装。下载安装包时推荐下载 LTS 版本，因为 LTS 版本是 Node.js 最稳定的版本，出现问题的概率较低。

下载完成之后，双击安装包，然后根据安装向导依次单击【继续】按钮即可完成 Node.js 的安装。安装完成之后，可以使用 node -v 命令来验证是否安装成功，如图 3-4 所示。

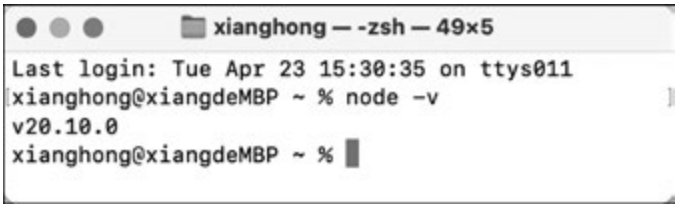

图 3-4 查看 Node.js 版本信息

3.5.2 创建 React 项目

React 支持使用命令行和 IDE 开发工具两种方式来创建项目，其中使用命令行方式创建 React 项目之前需要全局安装 create-react-app 工具，命令如下。

```
npm install -g create-react-app
```

然后就可以使用 npx 工具创建 React 项目了，在创建项目的过程中可以添加模板，如下

所示。

```
npx create-react-app react-ts-app
```

或者

```
npx create-react-app react-ts-app --template typescript
```

需要说明的是，在初始化 React 项目时，项目名称不能包含中文、空格和特殊符号，也不能使用 JavaScript 关键字作为项目名，如 class、new 等。

如果项目使用 TypeScript 语言进行开发，那么还需要在项目的根目录下新建一个 tsconfig.json 文件来配置编译器选项。好在 create-react-app 工具已经默认创建好了 tsconfig.json 文件，如下所示。

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
}
```

上述配置中，target 和 module 指定了编译后的代码兼容的目标环境和模块系统，jsx 设置为 react-jsx 时便于 React 组件的编译，strict 选项开启时会执行一系列严格的类型检查。

3.5.3 项目结构

使用 create-react-app 工具创建 React 项目时，工具会默认根据模板帮助开发者创建好必要的工程结构，如图 3-5 所示。

上面这些目录和文件构成了 React 项目的基础结构，开发者可以根据项目需要合理添

图 3-5 React 项目结构示意图

加或修改这些文件结构和配置，说明如下。

- node_modules：存放项目所有依赖包的地方。
- public：用于存放静态资源，如 index.html。
- src：项目源代码目录，包含组件、页面、上下文、钩子等。
- src/components：用于存放所有可复用的 React 组件。
- src/contexts：用于存放 React Context，用于状态管理。
- src/hooks：用于存放自定义的 React Hooks。
- src/pages：用于存放页面组件，通常与路由相关联。
- src/services：用于存放服务层代码，如 API 调用。
- src/utils：用于存放工具函数。
- App.tsx：应用的根组件。
- index.tsx：应用的入口文件。
- package.json：项目的依赖和配置文件，如启动、测试和构建等脚本。
- package-lock.json：依赖版本锁定文件。
- README.md：项目说明文件。

3.5.4 运行项目

任何应用程序都有一个执行的入口，React 项目的执行入口在 index.tsx 文件中，源码如下。

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

其中，App.tsx 是项目的根组件，ReactDOM.createRoot 的作用是创建一个根容器用于渲染 React 应用，root.render 则用于将根组件 App 渲染到创建的根容器中。同时，React 会默认启动严格模式，目的是检测潜在的问题。

使用 yarn start 命令启动应用，然后打开浏览器输入地址 http://localhost:3000/即可看到运行结果，如图 3-6 所示。

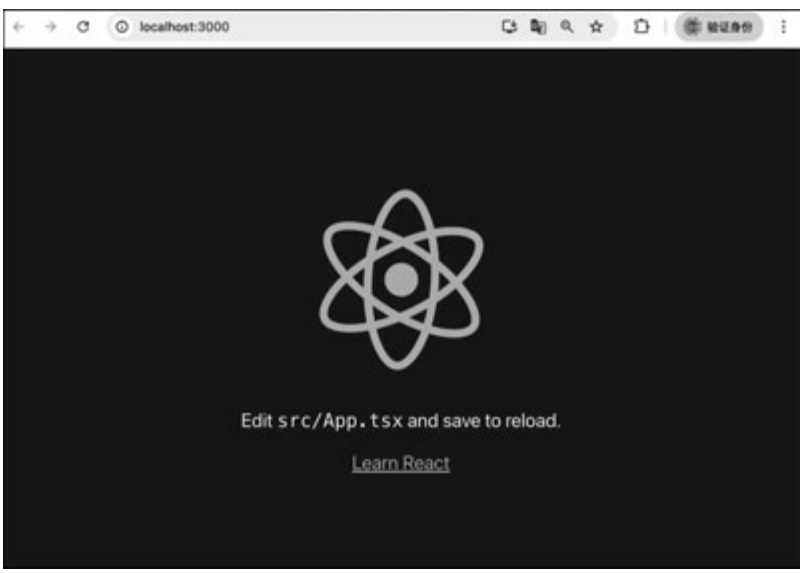

图 3-6　运行 React 示例项目

3.5.5　项目调试

React 项目的调试需要用到 React DevTools 插件，默认情况下 Chrome 等浏览器已经

集成了 DevTools 插件，所以 React 项目的调试基本上都是通过浏览器来完成的。

在 VS Code 中启动 React 项目，然后在 Chrome 浏览器中访问该项目，依次单击【Chrome 菜单】→【更多工具】→【开发者工具】，或者使用快捷键【Command＋Option＋I】打开 Chrome 浏览器的调试窗口，如图 3-7 所示。

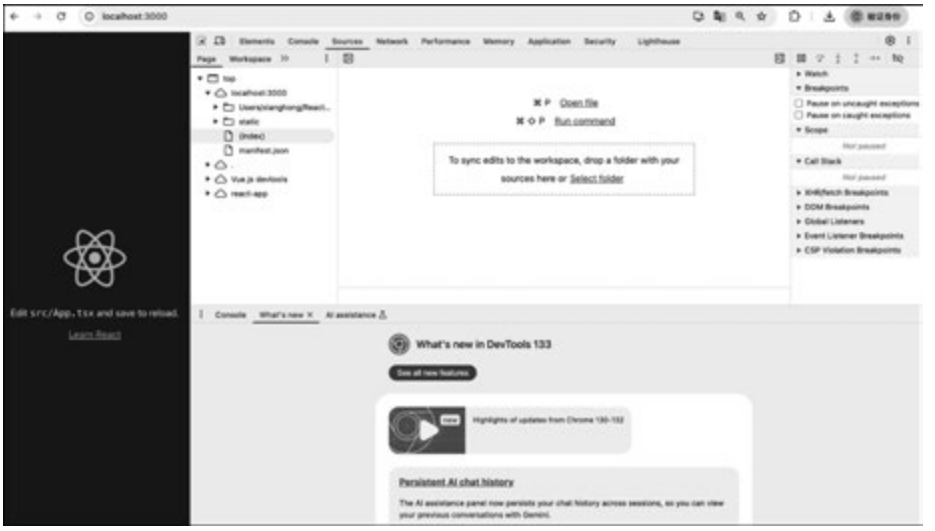

图 3-7 Chrome 浏览器调试窗口

单击调试面板顶部的卡片切换到 Sources 选项，使用快捷键【Command＋O】找到需要调试的源代码文件，然后在需要调试的地方添加一个断点，再次运行程序即可执行断点调试操作，如图 3-8 所示。

图 3-8 React 项目断点调试

可以看到，当程序运行到断点的地方时，程序就会自动挂起。此时，可以在调试面板的右侧获取应用的线程状态、变量值、调用栈、全局监听器等信息。

此时也可以通过调试工具控制区来控制断点的暂停、执行和终止，以及执行单步进入、单步跳过，单步跳出等操作，如图 3-9 所示。

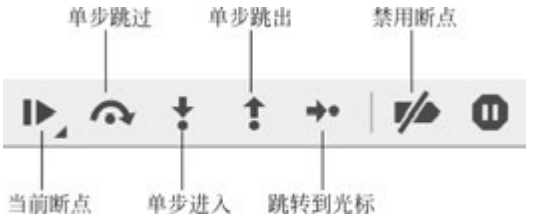

图 3-9 Chrome 浏览器调试工具栏

除了上述的代码调试功能外，Chrome DevTools 还提供元素定位、网络分析、控制台日志、性能分析、内存管理和安全管理等基本的模块，能够满足开发者各种日常的分析需求。

3.6 React Hook

3.6.1 Hook 简介

Hook 是从 React 16.8 版本推出的新特性，目的是解决 React 的状态共享以及组件生命周期管理混乱的问题。Hook 的出现，可以让开发者在不编写类组件的情况下使用组件的状态及其他的 React 特性，并且 Hook 极大地简化和优化了函数组件的开发过程。事实上，Hook 提供了一种更简洁、更灵活的方式来管理组件的状态和生命周期。

在 React 前端开发中，组件的状态管理是一件很麻烦的事情。而 React Hook 的出现，使得组件的状态管理只共享数据处理逻辑，并不会共享数据本身，因此开发者也就不需要关心数据与生命周期绑定的问题。例如，下面是使用类组件实现计数器的示例代码。

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
          Click me
        </button>
```

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

可以看到，如果是使用类组件来开发 React 页面，类组件需要自己声明状态并编写操作状态的方法，并且还需要维护状态的生命周期，开发起来显得特别烦琐。如果使用 React Hook 提供的 State Hook 来处理状态，那么上面的代码将会简洁许多，如下所示。

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

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

在上面的代码中，Example 从一个类组件变成了一个函数组件，此函数组件拥有自己的状态，并且不需要开发者再手动调用 setState()方法来更新变量的状态，这也正是 useState 的魅力所在。除了 useState 外，React 的常用 Hook 还包括 useEffect、useContext、useReducer、useCallback、useMemo、useRef、useLayoutEffect、useImperativeHandle 和 useDebugValue 等。这些 Hooks 涵盖了状态管理、副作用处理、性能优化、DOM 操作等各个方面。

总的来说，Hook 是一种让函数组件拥有状态和副作用处理能力的特性。它的实现原理基于 React 的 Fiber 架构和 JavaScript 的闭包机制，通过 Hooks 链表来管理组件的状态和副作用。使用 Hooks 可以让开发者更方便地编写清晰、简洁且易于维护的组件代码，进而提高代码的可读性和可维护性。

3.6.2 useState

useState 是 React 内置的最基础的 Hook 之一，主要用于在函数组件中添加状态管理。useState 接受一个初始值作为参数，并返回一个状态数组，数组的第一个元素表示当前的状态，第二个元素表示一个更新状态的函数。在具体使用时，可以通过解构赋值来获取这两个值，如下所示。

```
import React, { useState } from "react";

function Counter() {
```

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

上面的代码中使用 useState 方法声明了一个 count 变量和 setCount 方法，count 的初始状态是 0。当单击按钮时会调用 setCount 方法实现 count 值的累加。

3.6.3 useEffect

useEffect 的作用是处理函数组件中的副作用。在计算机科学中，如果一个函数或其他操作修改了其局部环境之外的状态变量的值，那么它就被称为有副作用，副作用是相对于主作用来说的。对于 React 组件来说，主作用就是根据数据渲染 UI，除此之外其他的操作都是副作用。

正常情况下，网络请求、模块订阅以及 DOM 操作都属于副作用的范畴，官方不建议开发者在函数体中编写带有副作用的代码。为了便于开发者处理这些副作用，React 提供了 useEffect 函数。为了便于说明，我们还是以计数器为例，首先在类组件的 componentDidMount 和 componentDidUpdate 生命周期函数添加如下代码。

```
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  componentDidMount() {
    document.title = `You clicked ${this.state.count} times`;
  }

  componentDidUpdate() {
    document.title = `You clicked ${this.state.count} times`;
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>
```

```
            Click me
          </button>
        </div>
      );
    }
  }
```

可以看到，componentDidMount 和 componentDidUpdate 两个生命周期函数中的代码是一样的。之所以出现同样的代码，是因为在很多情况下，我们希望在组件加载和更新时执行同样的操作。因此，我们希望可以对上述的代码进行合并处理，遗憾的是类组件并没有提供这样的方法。

不过，现在可以借助函数式组件提供 useEffect 钩子来避免这种问题。useEffect 也正是函数式组件中为了替换类组件的 componentDidMount、componentDidUpdate 和 componentWillUnmount 生命周期函数而出现的，如下所示。

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

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  },[count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

事实上，useEffect 只会在组件被渲染后（即 DOM 更新之后）才会执行副作用函数，这意味着它的执行过程是异步的，因此我们不用担心它会阻塞页面的渲染。

对于需要处理消息订阅的场景，类组件通常会在 componentDidMount 生命周期中设置订阅消息，然后在 componentWillUnmount 生命周期中清除订阅。而在类组件中，则可以使用 useEffect 的第二个参数来清除消息的订阅，如下所示。

```
import React, { useState, useEffect } from "react";

function Timer() {
  const [seconds, setSeconds] = useState(0);
```

```
  useEffect(() => {
    const intervalId = setInterval(() => {
      setSeconds((seconds) => seconds + 1);
    }, 1000);
    return () => clearInterval(intervalId);
  }, []);

  return <div>Seconds: {seconds}</div>;
}
```

在上面的代码中，使用 setInterval 的方法实现了一个计时器。为了防止内存泄漏和不必要的资源消耗，我们需要使用 useEffect 来清除计时器，为了达到清除计时器的目的，需要将 useEffect 的第二个参数设置成空数组[]，即回调函数仅执行一次。

除了 useEffect 外，useLayoutEffect 也可以用于处理函数组件中的副作用。不同之处在于，useEffect 在浏览器渲染完成后执行，而 useLayoutEffect 则在浏览器渲染前执行。

3.6.4 useRef

在 React 开发中，Ref 的主要作用是获取组件实例或者 DOM 元素。创建 Ref 主要有两种方式，即 createRef 和 useRef。其中，使用 createRef 方式创建的 Ref 每次渲染都会返回一个新的引用，而 useRef 每次渲染都会返回相同的引用。

useRef 作为 React 内置的一个钩子，主要用于在函数组件中创建一个可变的引用对象。useRef 返回的对象含有一个名为 current 的属性，可以用来存储任何值，并且在组件重新渲染时也会保持不变。例如，我们可以使用 current 属性来保存 DOM 节点的信息，以便在组件的生命周期中访问它。

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

function Example() {
  const inputRef = useRef(null);

  useEffect(() => {
    if (inputRef.current) {
      inputRef.current.focus();
    }
  }, []);

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

在上面的代码中，创建了一个 inputRef，然后在组件挂载后调用 focus 方法使输入框获

得焦点。

需要说明的是，过度使用 useRef 可能会导致代码难以理解和难以维护。因此，只有在确实需要直接访问 DOM 节点或保存不依赖于组件状态的值时，才建议使用 useRef。

3.6.5 useContext

在类组件中，组件之间的数据共享是通过属性 props 来实现的，跨级组件的数据共享只能通过逐级传递和使用状态管理框架来实现。而在函数组件中，跨级组件之间实现数据共享可以直接通过使用 useContext 来实现。

事实上，useContext 是 React Hook 提供的一种针对跨级组件数据传递场景的主要方式，避免了使用 props 数据层层传递的麻烦，能够有效保持代码整洁。例如，下面是使用 useContext 实现父子组件数据共享的示例。

```
import React, { useContext } from "react";

const MyContext = React.createContext();

function Child() {
  const { count, setCount } = useContext(MyContext);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}

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

  return (
    <MyContext.Provider value={{ count, setCount }}>
      <Child />
    </MyContext.Provider>
  );
}
```

useContext 接收一个上下文对象 context，并返回该上下文对象的当前值。当前上下文对象的值由上层组件中距离当前组件最近的数据提供者决定。

在上面的代码中，使用 createContext 方法创建了一个全局的上下文对象，并在 Parent 组件中声明了状态 count 和 setCount 方法。然后使用 Provider 包裹需要传递的变量，最后在 Child 组件中使用 useContext 方法获取上下文对象的当前值。获取到的上下文对象的值是由距离当前组件最近的数据提供者决定的，此处指的是 Parent 组件。

3.6.6 useReducer

useReducer 是 React 提供的一个用于管理复杂状态逻辑的钩子，在 React 开发中，可以使用它替代 useState 进行组件状态的管理。useReducer 的设计思想源自 Redux 状态管理框架的 Reducer 模式，将状态更新逻辑集中到一个函数中，使代码更易于维护和测试。Reducer 本质上是一个纯函数，接收当前的 state 和 action，并返回一个新的 state。

作为 useState 的升级版本，useReducer 可以用来在某些复杂的场景中替换 useState，如嵌套场景中的 state 和 state 包含多个子状态时，如下所示。

```
import React, { useReducer } from "react";

const initialState = { count: 0 };

function reducer(state, action) {
  switch (action.type) {
    case "increment":
      return { count: state.count + 1 };
    case "decrement":
      return { count: state.count - 1 };
    default:
      throw new Error();
  }
}

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

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

在上面的代码中，定义了一个 reducer 函数，它接收 state 和 action 两个参数，并返回一个状态对象。然后又提供了一个初始状态 initialState 以及调用 useReducer 的方法，当单击增加和减少按钮时就会通过 action 分别触发增加或减少 count 值的操作。

在某些特定的场景中，可能需要惰性地初始化 state 的内容，此时需要将初始化函数作为 useReducer 的第三个参数进行传入，如下所示。

```
function init(initialCount) {
    return {count: initialCount};
}
```

```
function reducer(state, action) {
    …. //省略代码
}

function Counter({initialCount}) {
    const [state, dispatch] = useReducer(reducer, initialCount, init);
    return (
      … //省略代码
    );
}
```

可以看到，useReducer 作为 React 提供的用于管理复杂状态的工具，特别适合用来处理多状态关联、复杂逻辑或需要替代 Redux 的场景。它通过 reducer 函数集中处理状态更新逻辑，使得代码更具可维护性和可预测性。

3.6.7 自定义 Hook

React 官方虽然内置了诸如 useState、useContext、useEffect 和 useReducer 等钩子，但它们并不能满足所有的开发场景，如记录用户是否在线或者连接聊天室。对此，React 允许开发者根据应用需求创建属于自己的 Hook。

在 React 开发中，我们可以将组件中共用的状态逻辑提取出来，使其在多个组件中复用。事实上，自定义 Hook 其实就是一个以 use 关键字开头的函数，函数内部可以调用其他的 Hook。比如，在聊天室程序中，我们可以将显示好友状态提取成可重用的函数，如下所示。

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });
  return isOnline;
}
```

可以发现，需要共享的好友状态逻辑已经被提取到 useFriendStatus 的自定义 Hook 中。然后，我们就可以在个人中心页面和聊天列表页面中，像使用普通函数一样调用自定义的 Hook 函数，如下所示。

```
//个人中心
function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}

//聊天列表
function FriendListItem(props) {
  const isOnline = useFriendStatus(props.friend.id);

  return (
    <li style={{ color: isOnline ? 'green' : 'black' }}>
      {props.friend.name}
    </li>
  );
}
```

需要说明的是，Hooks 的设计极度依赖事件定义的顺序，如果在后续的渲染环节中 Hooks 的调用顺序发生变化，就可能会出现不可预知的问题。在 React 应用开发过程中，为了保证 Hooks 调用顺序的稳定性，官方开发了一个名为 eslint-plugin-react-hooks 的 ESLint 插件来进行静态代码检测。

3.7 习题

一、选择题

1. 在 JSX 语法中，可以在大括号内放置任何有效的 JavaScript 表达式，以下哪些类型的描述是正确的？（　　）（多选）

A. 算术，主要以算术运算符计算结果为数字的表达式

B. 逻辑，主要以逻辑运算符计算结果为真假值的表达式

C. 右侧表达式，用于给目标赋值的表达式

D. 字符串，主要以字符串运算符计算结果为字符串的表达式

2. 以下哪些属于 React 提供的 Hook？（　　）（多选）

A. useState　　B. useContext　　C. useMemo　　D. useCallBack

3. useEffect 可以模拟类组件的哪些生命周期？（　　）（多选）

A. componentDidMount　　B. componentWillUnmount

C. componentDidUpdate　　D. componentWillMount

4. 以下哪些 Hook 可以用于在函数组件中处理路由？（　　）（多选）

A. useState　　B. useContext　　C. useHistory　　D. useEffect

二、简述题

1. 什么是虚拟 DOM？简单介绍 diff 算法实现原理。
2. 什么是 JSX 语法？它与 JavaScript、HTML 有什么联系？
3. 什么是 React Hook，以及常见的 Hook 有哪些？
4. useMemo 和 UseCallBack 有什么区别？