React组件

Created and last modified by 王文磊 on Feb 14, 2022

学习资料：

https://overreacted.io/zh-hans/how-are-function-components-different-from-classes/

https://juejin.cn/post/6944312020825014302

https://cloud.tencent.com/developer/article/1771344

一、函数组件

本质上就是javascript的函数，与普通函数的区别是接收一个props（代表属性）对象并返回一个 React 元素，早期并没有 React-Hooks 的加持，函数组件内部无法定义和维护 state，因此它还有一个别名叫“无状态组件”。

- 箭头函数形式

```
const Hello = (props) => { return <div>{props.message}</div>
// 可以简写成  const Hello = props => <div>{props.message}</div>
```

- function 形式

```
function Hello(props) {
  return <div>{props.message}</div>
}
```

特点：

- **只负责接收 props，渲染 DOM**
- **没有 state**
- **返回了一个 React 元素**
- **不能访问生命周期方法**
- **不需要声明类：可以避免 extends 或 constructor 之类的代码，语法上更加简洁。**
- **不会被实例化：因此不能直接传 ref（可以使用 React.forwardRef 包装后再传 ref）。**
- **不需要显示声明 this 关键字：在 ES6 的类声明中往往需要将函数的 this 关键字绑定到当前作用域，而因为函数式声明的特性，我们不需要再强制绑定。**
- **更好的性能表现：因为函数式组件中并不需要进行生命周期的管理与状态管理，因此React并不需要进行某些特定的检查或者内存分配，从而保证了更好地性能表现。**

二、类组件

所谓类组件，就是基于 ES6 Class 这种写法，通过继承 React.Component 得来的 React 组件。以下是一个典型的类组件：

```
class DemoClass extends React.Component {
  // 初始化类组件的 state
  state = {
    text: ""
  };

  componentDidMount() {
    // 省略业务逻辑
  }
  changeText = (newText) => {
    // 更新 state
    this.setState({
      text: newText
    });
  };
  // 编写生命周期方法 render
  render() {
    return (
      <div>
        <p>{this.state.text}</p>
        <button onClick={()=>this.changeText("newText")}>点我修改</button>
      </div>
    );
  }
}
```

特点：

- **为了避免代码冗余，提高代码利用率，组件可以重复调用**
- **组件的属性props是只读的，调用者可以传递参数到props对象中定义属性，调用者可以直接将属性作为组件内的属性或方法直接调用。往往是组件调用方调用组件时指定props定义属性，往往定值后就不改边了，注意组件调用方可赋值被调用方。**
- **通过props的方式进行父子组件交互,通过传递一个新的props属性值使得子组件重新render，从而达到父子组件通讯。**
- **{...this.props}可以传递属性集合，...为属性扩展符**
- **组件必须返回了一个 React 元素**
- **组件中state为私有属性，是可变的，一般在construct()中定义，使用方法：不要直接修改 state(状态)**
- **修改子组件还有一种方式，通过 ref属性，表示为对组件真正实例引用，其实就是ReactDOM.render()返回的组件实例**

三、受控 / 非受控组件

在 HTML 中，表单元素如 <input>，<textarea> 和 <select> 表单元素通常保持自己的状态，并根据用户输入进行更新。而在 React 中，可变状态一般保存在组件的 state(状态) 属性中，并且只能通过 setState() 更新。

我们可以通过使 React 的 state 成为 “单一数据源原则” 来结合这两个形式。然后渲染表单的 React 组件也可以控制在用户输入之后的行为。

这种形式，其值由 React 控制的输入表单元素称为“受控组件”。

那么相反的，值并不由 React 进行控制，该组件自己输入，减少等等，该元素称为非受控组件。

四、Q&A

Q：函数组件和类组件的区别

A：

React中组件主要可分为函数组件和类组件，两者区别是函数组件没有state和生命周期，故函数组件也称为stateless functional components， 适用于仅进行简单渲染操作的组件。
设计思想层面

> - **类组件的根基是 OOP（面向对象编程），所以它有继承、有属性、有内部状态的管理。**
> - **函数组件的根基是 FP (函数式编程)。它属于“结构化编程”的一种，与数学函数思想类似。也就是假定输入与输出存在某种特定的映射关系，那么输入一定的情况下，输出必然是确定的。**

渲染差异

函数组件会捕获render内部的状态

能力

- **类组件通过生命周期包装业务逻辑**
- **函数组件可以通过React Hooks 钩子函数来模拟类组件中的生命周期（useState ，useEffect，useContext，useCallback 等）**

> **Hooks 的本质：一套能够使函数组件更强大、更灵活的“钩子”**

设计模式

在设计模式上，因为类本身的原因，类组件是可以实现继承的，而函数组件缺少继承的能力。

当然在 React 中也是不推荐继承已有的组件的，因为继承的灵活性更差，细节屏蔽过多，所以有这样一个铁律，组合优于继承。

发展趋势

函数组件更加契合 React 框架的设计理念

React 组件本身的定位就是函数，一个吃进数据、吐出 UI 的函数。作为开发者，我们编写的是声明式的代码，而 React 框架的主要工作，就是及时地把声明式的代码转换为命令式的 DOM 操作，把数据层面的描述映射到用户可见的 UI 变化中去。这就意味着从原则上来讲，React 的数据应该总是紧紧地和渲染绑定在一起的，而类组件做不到这一点。

由于 React Hooks 的推出，函数组件成了社区未来主推的方案。

React 团队从 Facebook 的实际业务出发，通过探索时间切片与并发模式，以及考虑性能的进一步优化与组件间更合理的代码拆分结构后，认为类组件的模式并不能很好地适应未来的趋势。 他们给出了3 个原因：

> - **this 的模糊性；**
> - **业务逻辑散落在生命周期中；**
> - **React 的组件代码缺乏标准的拆分方式。**

而使用 Hooks 的函数组件可以提供比原先更细粒度的逻辑组织与复用，且能更好地适用于时间切片与并发模式。

Q：react是如何实现函数组件的

A：

与JSX的原理有关，当组件的props发生变化时React会重新调用整个函数返回新的JSX以更新页面

```
// 标签的方式使用函数式组件：
<ComFn name={name} />
// 基本上等价于：
{ComFn({name})} //组件的方式使用，就是在调用函数
```

Q：函数组件/类组件使用场景

A：

- **在不使用 Recompose 或者 Hooks 的情况下，如果需要使用生命周期，那么就用类组件，限定的场景是非常固定的；**
- **但在 recompose 或 Hooks 的加持下，这样的边界就模糊化了，类组件与函数组件的能力边界是完全相同的，都可以使用类似生命周期等能力。**

Q：受控 / 非受控组件区别，应用场景

A:

区别：

1. **受控组件 受控组件依赖于状态 受控组件的修改会实时映射到状态值上，此时可以对输入的内容进行校验 受控组件只有继承React.Component才会有状态 受控组件必须要在表单上使用onChange事件来绑定对应的事件**

2. 非受控组件 非受控组件不受状态的控制 非受控组件获取数据就是相当于操作DOM 非受控组件可以很容易和第三方组件结合，更容易同时集成 React 和非 React 代码

总结：

- 共同点，都是指表单元素，或者表单组件
- 不同点，被react的state控制，就是受控组件。不会state控制，就是非受控。
- 受控组件的实现方式，就是设置state，使用事件调用setstate，更新数据和视图。
- 非受控组件，避开state，使用ref等等方式，更新数据和视图。

使用场景：

1. 受控组件使用场景：一般用在需要动态设置其初始值的情况。例如：某些form表单信息编辑时，input表单元素需要初始显示服务器返回的某个值然后进行编辑。
2. 非受控组件使用场景：一般用于无任何动态初始值信息的情况。例如：form表单创建信息时，input表单元素都没有初始值，需要用户输入的情况。