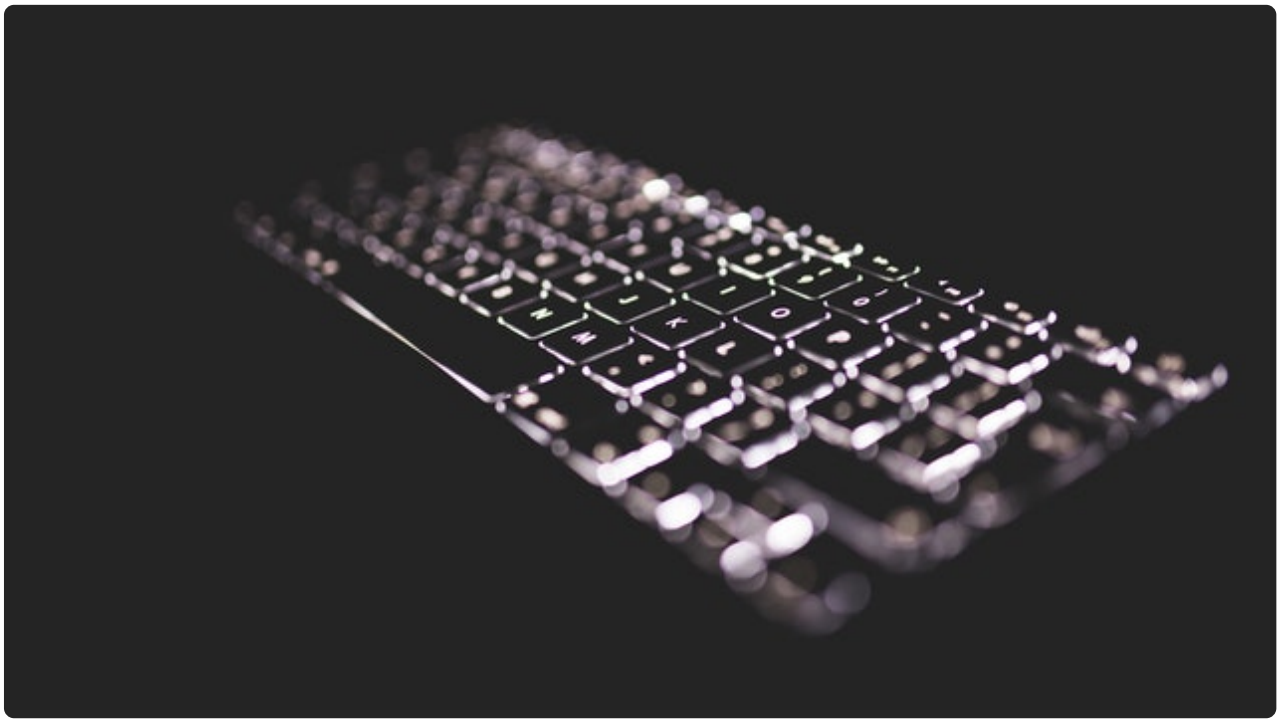
05 组件：从设计者的角度理解 React 组件

更新时间：2020-08-20 09:52:43



> 世上无难事,只要肯登攀。——毛泽东

什么是组件？

React 官方文档 - 组件 & Props 给出了组件的描述：组件允许你将 UI 拆分为独立可复用的代码片段，并对每个片段进行独立构思。组件，从概念上类似于 JavaScript 函数，它接受任意的入参（即 `props`），并返回用于描述页面展示内容的 React 元素。那么，一个 React 组件的构成可以简单描述成下面的形式。

```
React组件 = UI + 逻辑（处理 props，state以及事件等）
```

使用函数定义组件

对于开发者来说，定义组件最简单的方式就是编写 JavaScript 函数，如代码示例 2.2.1。

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

代码示例 2.2.1 函数组件的定义方式

该函数是一个有效的 React 组件，它接收唯一带有数据的 props 对象与并返回 **React 元素**。这类组件被称为 **函数组件**，因为它本质上就是 JavaScript 函数。

使用 class 定义组件

也可以使用 ES6 的 class 来定义组件，如代码示例 2.2.2。

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

代码示例 2.2.2 class 组件的定义方式

函数组件相对于 class 组件有其自身的特点，比如下面几个方面：

- 函数组件不会被实例化，整体渲染性能得到提升；
- 函数组件不能访问 `this` 对象；
- 函数组件无法继承 `React.Component` 上的属性，因此无法访问生命周期的方法；
- 无状态函数组件只能访问输入的 `props` 。

高阶组件

什么是高阶组件？

一个高阶组件只是一个包装了另外一个 React 组件的 **React 组件**。注意，这里说的是「包装」而不是嵌套成父子组件那样。那么如何定义一个高阶组件呢？见代码示例 2.2.3。

```
function ppHOC(WrappedComponent) {
  return class PP extends React.Component {
    render() {
      return <WrappedComponent {...this.props}/>
    }
  }
}
```

代码示例 2.2.3 定义一个高阶组件

高阶组件的本质就是一个 JavaScript 函数，该函数的参数就是我们想要包装的组件并且最终在不改动UI的情况下作为函数内部组件的元素返回，那么为什么要包装这个组件呢？因为这样的处理可以绑定函数内部组件的一些 `props` 。高阶组件的实现离不开组件就是函数的本质，因为 JavaScript 支持高阶函数，所以，高阶组件的设计可以认为是高阶函数的上层拓展。

高阶组件有什么作用呢？**高阶组件可以达到代码复用，逻辑抽象，底层代码抽离，渲染劫持，state 抽象和 props 更改等目的。**

组件的设计思想—数据驱动更新的优雅实现

所谓数据驱动更新指的是当一个组件内部的数据发生变化时，组件中返回的 UI 也随之变化并更新到屏幕。传统的 UI 模式中，我们一般是怎么保证组件内部的 UI 处于最新信息的状态呢？见代码示例 2.2.4。

```
class Form extends TraditionalObjectOrientedView {
  render() {
    const { isSubmitted, buttonText } = this.attrs;
    if (!isSubmitted && !this.button) {
      // 如果form还没有被提交，则创建button
      this.button = new Button({
        children: buttonText,
        color: 'blue'
      });
      this.el.appendChild(this.button.el);
    }

    if (this.button) {
      // 如果button已经存在，则更新button的文本
      this.button.attrs.children = buttonText;
      this.button.render();
    }

    if (isSubmitted && this.button) {
      // 如果form已经被提交，则删除button
      this.el.removeChild(this.button.el);
      this.button.destroy();
    }

    // ...
  }
}
```

代码示例 2.2.4 传统方式定义组件

上面是以传统方式定义 Form 组件的伪代码，其主要思想就是根据属性值的变化进行对应的 DOM 处理以保证组件能够及时返回最新的 UI 状态。这种设计组件的方式存在的问题就是，**随着属性值的种类增多组件变得愈发难维护，代码行数将会朝着组件可能状态数的平方量级增长**。那么，React 是如何优雅的解决这个问题的呢？

```
class Form extends React.Component {
  constructor(props) {
  super(props);
  this.state = {
     button: false
    };
  }

  render() {
    const { isSubmitted, buttonText } = this.props;
    return (
      <div className="wrap-box">
      {
        !isSubmitted && !this.state.button && (
        <button key="1">{buttonText}</button>
        )
      }
      </div>
    )
  }
}
```

代码示例 2.2.5 使用 React 定义组件

React 使用 state 为组件维护了自己的内部状态，使用 props 为组件维护了自己的外部状态。state 和 props 的变化意味着组件的 UI 需要更新。事实上，React 组件中的 UI 就是它的「元素」，React 元素是一种普通的 JavaScript 对象。因此，state 和 props 的变化只需要更新 JavaScript 对象即可。由于更新 JavaScript 对象要远比直接 DOM 树更加轻巧便利也使得 React 组件的更新渲染性能达到了很高的标准。

小结

本节介绍了 React 对组件的定义方式以及 React 如何以比较优雅的方式实现数据驱动更新的思想。在下一节将会介绍如何深入理解组件的生命周期。

}

04 React 世界中那些重要的概念

06 生命周期：如何理解 React 组件生命周期？