



2-7 Hooks

Hook 简介

Hook 是 **React 16.8** 的新增特性。它可以让你在不编写 `class` 的情况下使用 `state` 以及其他的 `React` 特性。

Hooks 带来的变革，让函数组件有了状态和其他的 `React` 特性，可以替代类组件。

JavaScript

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

function Example() {

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

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(0)}>
        Click me {count}
      </button>
    </div>
  );
}
```

Hook 解决了什么问题

Hook 解决了我们这些年来编写和维护成千上万的组件时遇到的各种各样看起来不相关的问题。无论你正在学习 React，或每天使用，或者更愿尝试另一个和 React 有相似组件模型的框架，你都可能对这些问题似曾相识。

在组件之间复用状态逻辑很难

React 没有提供将可复用性行为“附加”到组件的途径（例如，把组件连接到 store）。如果你使用过 React 一段时间，你也许会熟悉一些解决此类问题的方案，比如 `render props` 和 `高阶组件`。但是这类方案需要重新组织你的组件结构，这可能会很麻烦，使你的代码难以理解。如果你在 React DevTools 中观察过 React 应用，你会发现由 `providers`，`consumers`，`高阶组件`，`render props` 等其他抽象层组成的组件会形成“嵌套地狱”。尽管我们可以在 DevTools 过滤掉它们，但这说明了一个更深层次的问题：React 需要为共享状态逻辑提供更好的原生途径。

你可以使用 Hook 从组件中提取状态逻辑，使得这些逻辑可以单独测试并复用。

Hook 使你在无需修改组件结构的情况下复用状态逻辑。 这使得在组件间或社区内共享 Hook 变得更便捷。

具体将在 `自定义 Hook` 中对此展开更多讨论。

复杂组件变得难以理解

我们经常维护一些组件，组件起初很简单，但是逐渐会被状态逻辑和副作用充斥。每个生命周期常常包含一些不相关的逻辑。例如，组件常常在 `componentDidMount` 和 `componentDidUpdate` 中获取数据。但是，同一个 `componentDidMount` 中可能也包含很多其它的逻辑，如设置事件监听，而之后需在 `componentWillUnmount` 中清除。相互关联且需要对照修改的代码被进行了拆分，而完全不相关的代码却在同一个方法中组合在一起。如此很容易产生 bug，并且导致逻辑不一致。

在多数情况下，不可能将组件拆分为更小的粒度，因为状态逻辑无处不在。这也给测试带来了一定挑战。同时，这也是很多人将 React 与状态管理库结合使用的原因之一。但是，这往往会引入了很多抽象概念，需要你在不同的文件之间来回切换，使得复用变得更加困难。

为了解决这个问题，Hook 将组件中相互关联的部分拆分成更小的函数（比如设置订阅或请求数据），而并非强制按照生命周期划分。你还可以使用 `reducer` 来管理组件

的内部状态，使其更加可预测。

我们将在[使用 Effect Hook](#) 中对此展开更多讨论。

难以理解的 class

除了代码复用和代码管理会遇到困难外，我们还发现 class 是学习 React 的一大屏障。你必须去理解 JavaScript 中 `this` 的工作方式，这与其他语言存在巨大差异。还不能忘记绑定事件处理器。没有稳定的[语法提案](#)，这些代码非常冗余。大家可以很好地理解 props, state 和自顶向下的数据流，但对 class 却一筹莫展。即便在有经验的 React 开发者之间，对于函数组件与 class 组件的差异也存在分歧，甚至还要区分两种组件的使用场景。

另外，React 已经发布五年了，我们希望它能在下一个五年也与时俱进。就像 [Svelte](#), [Angular](#), [Glimmer](#) 等其它的库展示的那样，组件预编译会带来巨大的潜力。尤其是在它不局限于模板的时候。最近，我们一直在使用 [Prepack](#) 来试验 [component folding](#)，也取得了初步成效。但是我们发现使用 class 组件会无意中鼓励开发者使用一些让优化措施无效的方案。class 也给目前的工具带来了一些问题。例如，class 不能很好的压缩，并且会使热重载出现不稳定的情况。因此，我们想提供一个使代码更易于优化的 API。

为了解决这些问题，**Hook 使你在非 class 的情况下可以使用更多的 React 特性。**从概念上讲，React 组件一直更像是函数。而 Hook 则拥抱了函数，同时也没有牺牲 React 的精神原则。Hook 提供了问题的解决方案，无需学习复杂的函数式或响应式编程技术。

Hook API

- [基础 Hook](#)

- `useState`
- `useEffect`
- `useContext`

- [额外的 Hook](#)

- `useReducer`
- `useCallback`
- `useMemo`
- `useRef`
- `useImperativeHandle`
- `useLayoutEffect`
- `useDebugValue`
- `useDeferredValue`
- `useTransition`
- `useId`

- Library Hooks

- `useSyncExternalStore`
- `useInsertionEffect`