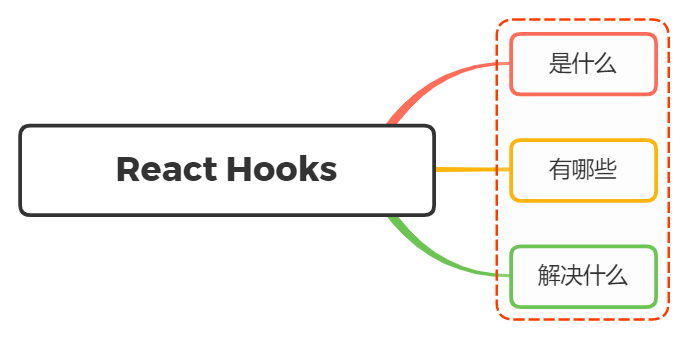
# 面试官：说说对React Hooks的理解？解决了什么问题？



## 一、是什么

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

至于为什么引入hook，官方给出的动机是解决长时间使用和维护react过程中常遇到的问题，例如：

* 难以重用和共享组件中的与状态相关的逻辑
* 逻辑复杂的组件难以开发与维护，当我们的组件需要处理多个互不相关的 local state 时，每个生命周期函数中可能会包含着各种互不相关的逻辑在里面
* 类组件中的this增加学习成本，类组件在基于现有工具的优化上存在些许问题
* 由于业务变动，函数组件不得不改为类组件等等

在以前，函数组件也被称为无状态的组件，只负责渲染的一些工作

因此，现在的函数组件也可以是有状态的组件，内部也可以维护自身的状态以及做一些逻辑方面的处理

## 二、有哪些

上面讲到，Hooks让我们的函数组件拥有了类组件的特性，例如组件内的状态、生命周期

最常见的hooks有如下：

* useState
* useEffect
* 其他

### useState

首先给出一个例子，如下：

import React, { useState } from 'react';  
  
function Example() {  
 // 声明一个叫 "count" 的 state 变量  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <p>You clicked {count} times</p >  
 <button onClick={() => setCount(count + 1)}>  
 Click me  
 </button>  
 </div>  
 );  
}

在函数组件中通过useState实现函数内部维护state，参数为state默认的值，返回值是一个数组，第一个值为当前的state，第二个值为更新state的函数

该函数组件等价于的类组件如下：

class Example 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>  
 );  
 }  
}

从上述两种代码分析，可以看出两者区别：

* state声明方式：在函数组件中通过 useState 直接获取，类组件通过constructor 构造函数中设置
* state读取方式：在函数组件中直接使用变量，类组件通过this.state.count的方式获取
* state更新方式：在函数组件中通过 setCount 更新，类组件通过this.setState()

总的来讲，useState 使用起来更为简洁，减少了this指向不明确的情况

### useEffect

useEffect可以让我们在函数组件中进行一些带有副作用的操作

同样给出一个计时器示例：

class Example 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>  
 );  
 }  
}

从上面可以看见，组件在加载和更新阶段都执行同样操作

而如果使用useEffect后，则能够将相同的逻辑抽离出来，这是类组件不具备的方法

对应的useEffect示例如下：

import React, { useState, useEffect } from 'react';  
function Example() {  
 const [count, setCount] = useState(0);  
   
 useEffect(() => { document.title = `You clicked ${count} times`; });  
 return (  
 <div>  
 <p>You clicked {count} times</p >  
 <button onClick={() => setCount(count + 1)}>  
 Click me  
 </button>  
 </div>  
 );  
}

useEffect第一个参数接受一个回调函数，默认情况下，useEffect会在第一次渲染和更新之后都会执行，相当于在componentDidMount和componentDidUpdate两个生命周期函数中执行回调

如果某些特定值在两次重渲染之间没有发生变化，你可以跳过对 effect 的调用，这时候只需要传入第二个参数，如下：

useEffect(() => {  
 document.title = `You clicked ${count} times`;  
}, [count]); // 仅在 count 更改时更新

上述传入第二个参数后，如果 count 的值是 5，而且我们的组件重渲染的时候 count 还是等于 5，React 将对前一次渲染的 [5] 和后一次渲染的 [5] 进行比较，如果是相等则跳过effects执行

回调函数中可以返回一个清除函数，这是effect可选的清除机制，相当于类组件中componentwillUnmount生命周期函数，可做一些清除副作用的操作，如下：

useEffect(() => {  
 function handleStatusChange(status) {  
 setIsOnline(status.isOnline);  
 }  
  
 ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);  
 return () => {  
 ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);  
 };  
});

所以， useEffect相当于componentDidMount，componentDidUpdate 和 componentWillUnmount 这三个生命周期函数的组合

### 其它 hooks

在组件通信过程中可以使用useContext，refs学习中我们也用到了useRef获取DOM结构......

还有很多额外的hooks，如：

* useReducer
* useCallback
* useMemo
* useRef

## 三、解决什么

通过对上面的初步认识，可以看到hooks能够更容易解决状态相关的重用的问题：

* 每调用useHook一次都会生成一份独立的状态
* 通过自定义hook能够更好的封装我们的功能

编写hooks为函数式编程，每个功能都包裹在函数中，整体风格更清爽，更优雅

hooks的出现，使函数组件的功能得到了扩充，拥有了类组件相似的功能，在我们日常使用中，使用hooks能够解决大多数问题，并且还拥有代码复用机制，因此优先考虑hooks

## 参考文献

* https://zh-hans.reactjs.org/docs/hooks-state.html
* https://zh-hans.reactjs.org/docs/hooks-effect.html
* https://www.cnblogs.com/lalalagq/p/9898531.html