14. forwardRef和 useImperativeHandle 是用来解决什么问题的？（上）

面试题与参考答案：React.forwardRef

主题：React.forwardRef 如何打通父子组件的 ref 通道

Q1: 什么是 React中的 `ref`？通常在哪些场景下会使用它?
A1:
在 React 中，`ref` 提供了一种访问组件渲染出的 DOM 元素或 React 组件实例的方式。它通常用于进行命令式的操作，例如：

- 管理焦点、文本选择或媒体播放。
- 触发强制动画。
- 集成第三方 DOM 库。

Q2: 为什么在父组件中直接给一个自定义的函数组件传递 `ref` 属性通常无法获取到子组件内部的 DOM 元素?
A2:
`ref` (和 `key` 一样) 是 React 特殊处理的 prop。默认情况下，函数组件是不能直接接收 `ref` prop 的，React 会自动忽略它。这是因为函数组件不像类组件那样有实例（instance）。React 这样设计，部分原因是为了保护组件的封装性，即父组件不应该过多地干涉子组件的内部实现。

Q3: `React.forwardRef` 是用来解决什么核心问题的?
A3:
`React.forwardRef` 主要用于解决一个场景：当父组件需要获取其子函数组件内部的特定 DOM 元素或子类组件实例的 `ref` 时。由于函数组件默认不接收 `ref` 属性，`forwardRef` 提供了一种机制来将父组件的 `ref` “转发”到子组件内部的特定元素上。

Q4: 当你使用 `React.forwardRef` 包裹一个函数组件时，该函数组件会接收哪些参数？它们分别是什么?
A4:
当使用 `React.forwardRef` 包裹一个函数组件时，该函数组件会接收两个参数：

1. `props`： 这是常规的 props 对象，包含了父组件传递给子组件的所有属性（除了 `ref`）。
2. `ref`： 这是父组件传递过来的 `ref`。通过 `forwardRef`，这个 `ref` 才能被函数组件接收到。

Q5: 请简要描述一下 `React.forwardRef` 的工作机制。
A5:
`React.forwardRef` 是一个高阶组件（或者说是一个函数，它接收一个组件作为参数并返回

一个新的组件）。它接收你的函数组件作为参数，并返回一个新的、增强版的 React 组件。这个增强后的组件，其函数定义会包含两个参数：`props` 和 `ref`。在组件内部，你可以将这个接收到的 `ref` 参数显式地附加到你想要引用的 DOM 元素或类组件实例上，从而实现了 `ref` 的转发。

Q6: 在面试中，你会如何向面试官解释 `React.forwardRef` 的核心价值?
A6:
我会这样解释：
“`React.forwardRef` 主要用于解决当父组件需要获取其子函数组件内部的特定 DOM 元素或子类组件实例的 `ref` 时的场景。

默认情况下，函数组件不能直接通过 `props` 接收 `ref`，`ref` 属性会被 React 忽略。`React.forwardRef` 提供了一种机制，它允许函数组件在其参数中接收一个 `ref`（通常是第二个参数），然后组件可以将这个 `ref` 向下传递（即‘转发’）给其内部的某个子元素（通常是 DOM 节点）或子组件。

它的核心价值在于：它使得自定义的函数组件在 `ref` 的行为上能够像原生 HTML 元素或其他类组件一样，允许父组件获取对其内部特定节点的引用，从而进行必要的命令式操作，比如聚焦、尺寸测量等，同时又保持了组件的封装性。可以把它比作给函数组件开了一个特定的通道，让父组件的 `ref` 可以安全地传递到子组件内部的目标元素上。”

Q7: 请用一个简单的代码示例说明如何使用 `React.forwardRef` 来创建一个自定义输入框组件，并允许父组件聚焦该输入框。

A7:

```
// CustomInput.js
import React from 'react';

// 使用 React.forwardRef 包裹我们的函数组件
const CustomInput = React.forwardRef((props, ref) => {
  // 第一个参数是 props 对象
  // 第二个参数 ref，就是父组件传递过来的 ref
  return (
    <input
      type="text"
      placeholder="我是自定义输入框"
      ref={ref} // 在这里，我们将接收到的 ref 转发给原生的 input 元素
      {...props} // 其他 props 正常传递
    />
  );
});

export default CustomInput;

// App.js (父组件)
import React, { useRef, useEffect } from 'react';
import CustomInput from './CustomInput';
```

```
function App() {
  const myCustomInputRef = useRef(null);

  useEffect(() => {
    // 组件挂载后，让自定义输入框聚焦
    if (myCustomInputRef.current) {
      myCustomInputRef.current.focus(); // 调用原生 input 的 focus 方法
    }
  }, []);

  const handleFocusButtonClick = () => {
    if (myCustomInputRef.current) {
      myCustomInputRef.current.focus();
    }
  };

  return (
    <div>
      <CustomInput ref={myCustomInputRef} />
      <button onClick={handleFocusButtonClick} style={{ marginTop: '10px'
}}>
        点击聚焦自定义输入框
      </button>
    </div>
  );
}

export default App;
```

在这个例子中，`CustomInput` 组件使用 `React.forwardRef` 接收 `ref` 并将其转发给内部的 `input` 元素。父组件 `App` 则可以通过 `myCustomInputRef` 来引用这个 `input` DOM 元素，并调用其 `focus()` 方法。

Q8: 讲义中提到 `forwardRef` 解决了父组件获取子组件 DOM 元素引用的问题。那么，如果父组件希望调用的不是子组件 DOM 元素的标准方法 (如 `focus`)，而是子组件自己封装的一些定制化方法，`forwardRef` 能直接满足这种需求吗?
A8:
`forwardRef` 本身主要用于将 `ref` 转发到 DOM 元素或类组件实例。如果直接将 `ref` 转发到 DOM 元素，父组件拿到的是 DOM 元素的引用，可以调用其所有原生方法。

如果希望父组件调用的是子组件自己封装的、更定制化的接口（而不是直接操作 DOM），单纯使用 `forwardRef` 将 `ref` 指向 DOM 元素是不够的。父组件直接拿到的还是 DOM 元素，而不是子组件的自定义方法。

有时候，我们可能不希望父组件直接操作 DOM 的所有方法，或者想暴露一些我们自己封装的、更定制化的接口给父组件。这时候，另一个 Hook—— `useImperativeHandle` 就要登场

了。`forwardRef` 通常需要与 `useImperativeHandle` 配合使用，来实现更精细地控制暴露给父组件的 `ref` 的行为和接口。