Higher-Order Components (HOCs)

Higher-Order Components (HOC) — это один из шаблонов в React, который позволяет повторно использовать логику компонентов. HOC берет компонент и возвращает новый компонент с дополнительными свойствами или поведением. Это позволяет избежать дублирования кода, делая систему более чистой и легко поддерживаемой.

Как создать HOC?

Создание HOC довольно простое. Это функция, которая принимает компонент и возвращает новый компонент:

```
function withExample(WrappedComponent) {
    return function EnhancedComponent(props) {
        // здесь можно добавить дополнительную логику
        return <WrappedComponent {...props} />;
    };
}
```

Пример: HOC для логирования

Предположим, у нас есть простой компонент `Hello`:

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

Теперь создадим HOC, который будет логировать пропсы, переданные компоненту:

```
function withLogging(WrappedComponent) {
    return function EnhancedComponent(props) {
        console.log("Props:", props);
        return <WrappedComponent {...props} />;
    };
}
```

Использование этого HOC:

```
const HelloWithLogging = withLogging(Hello);

// Использование нового компонента
<HelloWithLogging name="John" />;
```

Когда этот компонент будет отрендерен, в консоли увидим:

```
Props: { name: "John" }
```

Пример: HOC для работы с состоянием

Допустим, у нас есть компонент, который отображает число. Мы хотим создать HOC, который добавляет к этому компоненту кнопки для инкремента и декремента числа.

```
// Компонент, который отображает число
function CounterDisplay({ count }) {
    return <div>{count}</div>;
}

// HOC, который добавляет инкремент/декремент
function withCounter(WrappedComponent) {
    return function EnhancedComponent(props) {
        const [count, setCount] = React.useState(0);

        return (
            <div>
                <button onClick={() => setCount(count - 1)}>-</button>
                <WrappedComponent count={count} {...props} />
                <button onClick={() => setCount(count + 1)}>+</button>
```

```
            </div>
        );
    };
}

const CounterWithControls = withCounter(CounterDisplay);

// Использование
<CounterWithControls />;
```

Теперь у нас есть компонент `CounterWithControls`, который включает в себя логику управления счетчиком, а сам этот счетчик отображается через компонент `CounterDisplay`.

Higher-Order Components (HOC) — это продвинутый паттерн в React для повторного использования логики компонента. HOC сам по себе не является частью API React; скорее, это паттерн, вытекающий из композиционной природы компонентов в React.

HOC берет компонент и возвращает новый компонент с дополнительной функциональностью или пропсами.

Простой пример HOC

Давайте создадим HOC, который добавляет "hover" функциональность к компоненту. Этот HOC будет отслеживать, когда пользователь наводит мышь на компонент, и обновлять состояние `isHovered`.

Вот как это может выглядеть:

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

// HOC функция
const withHover = (WrappedComponent) => {
    return (props) => {
        const [isHovered, setIsHovered] = useState(false);

        const handleMouseOver = () => setIsHovered(true);
        const handleMouseOut = () => setIsHovered(false);

        return (
            <div onMouseOver={handleMouseOver} onMouseOut={handleMouseOut}>
                <WrappedComponent isHovered={isHovered} {...props} />
            </div>
        );
    };
```

```
};

// Обычный компонент
const Button = ({ isHovered, label }) => {
    return (
        <button style={{ background: isHovered ? "blue" : "white" }}>
            {label}
        </button>
    );
};

// Оборачиваем обычный компонент в HOC
const HoverableButton = withHover(Button);

// Использование обернутого компонента
const App = () => {
    return <HoverableButton label="Нажми на меня" />;
};

export default App;
```

В этом примере мы создали HOC `withHover`, который добавляет свойство `isHovered` к любому компоненту, который он оборачивает. Затем мы использовали этот HOC для создания нового компонента `HoverableButton`, который изменяет свой фон, когда на него наводят мышь.

Преимущества

1. **Повторное использование кода**: HOC позволяет повторно использовать код, логику и bootstrap компонентов.
2. **Разделение ответственности**: HOC можно использовать для изоляции и разделения логики от представления.

Недостатки

1. **Сложность**: Использование HOC может сделать дерево компонентов более сложным, что затрудняет отладку и тестирование.
2. **Коллизия пропсов**: Пропсы, передаваемые через HOC, могут перезаписывать существующие пропсы.