Hook的出现是为了解决Class存在的一些问题，但这并不是替换Class

**Hook解决了什么问题**

如下代码

class A extends React.Component {

    state = {

        page: 1,

        size: 10,

        total: 100,

        orders: [],

        text: ''

    }

intervalH;

    componentDidMount() {

        this.intervalH = setInterval(() => {

            // ...

        }, 1000);

        this.setState({

            total: 10000,

            text: this.props.text

        });

}

    shouldComponentUpdate(nextProps) {

        if(this.props.text != nextProps.text){

            this.state.text = nextProps.text;

            return false;

        }

        return true;

}

    componentWillUnmount() {

        clearInterval(this.intervalH);

}

    render() {

        return <div>A -- page: {this.state.page}, total: {this.state.total}, text: {this.state.text}</div>

    }

}

class B extends React.Component {

    state = {

        page: 1,

        size: 10,

        total: 100,

        articles: []

}

    componentDidMount() {

        this.setState({

            total: 10000,

        });

    }

    render() {

        return <div>B -- page: {this.state.page}, total: {this.state.total}</div>

    }

}

这里存在3个问题

1. A、B组件这部分的逻辑是相同的，是否可以分离

class A extends React.Component {

    state = {

        page: 1,

        size: 10,

        total: 100,

    }

    componentDidMount() {

        this.setState({

            total: 10000,

        });

    }

}

class B extends React.Component {

    state = {

        page: 1,

        size: 10,

        total: 100

}

    componentDidMount() {

        this.setState({

            total: 10000,

        });

    }

}

1. 这2个代码逻辑相关，应该放在一个函数内，但由于生命周期的原因，不得以拆分到2个方法内

componentDidMount() {

    this.intervalH = setInterval(() => {

        this.setState({page: this.state.page + 1});

    }, 1000);

}

componentWillUnmount() {

    clearInterval(this.intervalH);

}

1. 这2个代码似乎有逻辑上的关联

    componentDidMount() {

        this.setState({

            text: this.props.text

        });

}

    shouldComponentUpdate(nextProps) {

        if(this.props.text != nextProps.text){

            this.state.text = nextProps.text;

            return false;

        }

        return true;

    }

为了更好的解决上面的问题，react推出了Hook

我们使用Hook解决上面的问题

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

const usePage = () => {

    let [page, setPage] = useState(1);

    let [size, setSize] = useState(10);

let [total, setTotal] = useState(100);

    useEffect(() => {

        setTotal(1000);

}, []);

    return {

        page, setPage,

        size, setSize,

        total, setTotal,

    }

}

const A = (props: any) => {

    let pager = usePage();

let [text, setText] = useState('');

    useEffect(() => {

        setText(props.text);

}, [props.text]);

    useEffect(() => {

        let intervalH = setInterval(() => {

            pager.setPage(pager.page + 1);

        }, 1000);

        return () => {

            clearInterval(intervalH);

        }

});

    return <div>A -- page: {pager.page}, total: {pager.total}, text: {text}</div>

}

const B = (props: any) => {

let pager = usePage();

    return <div>B -- page: {pager.page}, total: {pager.total}</div>

}

所有，Hook主要是解决class的生命周期内逻辑被拆分和状态复用问题

**何时使用Hook**

你可以在任何时候使用Hook，但在一个复杂的组件里，Hook会使渲染逻辑变得复杂

我的实践方案是：容器组件使用类组件，展示组件使用函数式组件（Hook）