React创建单页应用

npx create-react-app my-app

cd my-app

npm start

npm run build

会在 build 文件夹内生成你应用的优化版本。

怪事

function Clock(props){

return <p>现在是{props.time.toString()}</p>

}

setInterval(ReactDOM.render(<Clock time={new Date()}></Clock>,document.getElementById('root')), 1000);

这样写不行，时间永远不变

function tick(){

ReactDOM.render(<Clock time={new Date()}></Clock>, document.getElementById('root'));

}

setInterval(tick, 1000)

这样写就会实时更新当前时间

艹，你是傻逼吗。React.DOM.render()返回值不是函数，放到setInterval()里肯定不会执行！

props.children

Component.defaultProps

## 目录结构

my-app/

README.md

node\_modules/

package.json

public/

index.html

favicon.ico

src/

App.css

App.js

App.test.js

index.css

index.js

logo.sv

* index.html里<body>有一个<div id="root"></div>

## React 组件类

是React.Component的子类

import React from "react";

import ReactDOM from "react-dom";

import "./index.css";

class Nihao extends React.Component {

render() {

return <div>你好</div>

}

}

render()方法返回了一个 React 元素，这是一种对渲染内容的轻量级描述

当组件的状态数据props、state改变时，组件会再次调用 render() 方法重新渲染。所以render()里的const也不恒定

每次组件更新时 render 方法都会被调用，但只要在相同的 DOM 节点中渲染，就仅有一个 组件的 class 实例被创建使用。这就使得我们可以使用如 state 或生命周期方法等很多其他特性。

ReactDOM.render(<Nihao />, document.getElementById("root"));

加载

ReactDOM.render(<h1>Hello, world!</h1>,document.getElementById('root'))

可以直接render <h1>等标签

class Father extends React.Component{

render(){

return <Nihao name="TianCheng"/>

}

}

ReactDOM.render(<Father />, document.getElementById("root"));

render()返回值中可以有其他组件，形成父子关系

组件接收一些参数props，在render() 返回的视图中可以引用

<div> 你好 {this.props.name} </div>

显示

你好 TianCheng

在 React 应用中，数据通过 props 的传递，从父组件流向子组件。

不同子组件相互通信，需要都传到父组件中，然后，父组件通过props传回子组件

所有 React 组件都必须像纯函数一样保护它们的 props 不被更改，即组件中不能出现类似props.start=1这样的语句

return <Nihao name={this.state.n}/>

如果像这样传值，父组件的state改变之后，子组件的props.name属性也相应改变。应该是重新绘制了

除了render()，其他方法也可以返回一个视图，然后再组件类中渲染

renderSquare(i) {

return <Square value={i} />;

}

如上，renderSquare返回一个<Square>组件。并且没有加 ()

render()中可以调用

<div> {this.renderSquare(0)} </div>

constructor(props) {

super(props);

}

组建类constructor写法

super()之后已经完成了父组件给子组件的属性传值

constructor(props) {

super(props);

this.state = {

value: null,

};

}

this.state 应该被视为一个组件的私有属性。

好像就是不能通过父组件传值的props

State 的更新可能是异步的，出于性能考虑，React 可能会把多个 setState() 调用合并成一个调用。也就是说，setState之后，并没有立即setState，而是等着多个setState一起调用，而这个时候state可能早就变了

因为 this.props 和 this.state 可能会异步更新，所以你不要依赖他们的值来更新下一个状态。例如不要这样更新

this.setState({

counter: this.state.counter + this.props.increment,

});

要解决这个问题，可以让 setState() 接收一个函数而不是一个对象。这个函数用上一个 state 作为第一个参数，props 做为第二个参数：

this.setState((state, props) => ({

counter: state.counter + props.increment

}));

注意，箭头函数返回一个对象时要在对象外边加小括号，不然按代码块编译了

说实话，咱也不知道为什么这样就解决问题了

你setState调用之后不立即更新，而是等几秒更新，那还玩儿个屁啊，早尼玛耽误了

如果state是{a:1}, 那么调用this.setState({b:2})并不会把a这个属性去掉，而是变为{a:1, b:2}

不要直接操作state，用this.setState()方法

除了构造函数中，给state赋值必须用setSate()。构造函数中可以this.state={a:1}这样

* 函数组件

如果组件只包含一个 render 方法，不包含 state，那么使用函数组件就会更简单。不需要定义一个继承于 React.Component 的类，可以定义一个函数，这个函数接收 props 作为参数，然后返回需要渲染的元素。

function Square(props) {

return (

<button className="square" onClick={props.onClick}>

{props.value}

</button>

);

}

this.props 都替换成 props

## JSX

const element = <h1>Hello, world!</h1>;

JSX是一个 JavaScript 的语法扩展

JSX 可以生成 React “元素”。element就是元素变量

JSX必须有一个父元素包裹所有的元素，不能const e = <p></p><p></p>

const name = 'Josh Perez';

const element = <h1>Hello, {name}</h1>;

在 JSX 语法中，可以在大括号内放置任何有效的 JavaScript 表达式。例如，2 + 2，user.firstName 或 formatName(user)

const element = (

<h1>

Hello, {formatName(user)}!

</h1>

);

建议将内容包裹在括号中，再把JSX 拆分为多行

通过使用引号，来将属性值指定为字符串字面量：

const element = <div tabIndex="0"></div>;

也可以使用大括号，在属性值中插入一个 JavaScript 表达式：

const element = <img src={user.avatarUrl}></img>;

或者

const element = <img src={user.avatarUrl} />;

Babel 会把 JSX 转译成一个名为 React.createElement() 函数调用。

以下两种代码完全等效：

const element = (

<h1 className="greeting">

Hello, world!

</h1>

);

const element = React.createElement(

'h1',

{className: 'greeting'},

'Hello, world!'

);

Babel 是一个工具链，主要用于将 ECMAScript 2015+ 版本的代码转换为向后兼容的 JavaScript 语法，以便能够运行在当前和旧版本的浏览器或其他环境中

* 条件渲染

let btn = this.state.isLogin ? <Logout onClick={this.clickLogout} /> : <Login onClick={this.clickLogin}/>;

也可以用与运算符

{ unreadMessages.length > 0 && <h2> You {unreadMessages.length} </h2> }

function WarningBanner(props) {

if (!props.warn) {

return null;

}

return <div className="warning">Warning! </div>;

}

可以通过return null阻止条件渲染，虽然可能组件已经被渲染了出来，这样做就隐藏了组件

## 生命周期

当组件第一次被渲染到 DOM 中的时候，挂载mount

当 DOM 中组件被删除的时候，卸载unmount

componentDidMount() {

}

componentWillUnmount() {

}

componentDidMount() 方法会在组件已经被渲染到 DOM 中后运行

componentWillMount has been renamed, and is not recommended for use.

componentDidMount() {

this.timerID = 1;

}

可以向class添加任意额外字段。好像不用事先声明

componentWillMount()

componentDidMount()

componentWillUpdate(object nextProps, object nextState)

componentDidUpdate(object prevProps, object prevState)

componentWillUnmount()

getInitialState

render

componentDidMount

componentWillUnmount

销毁

## 事件处理

<button onClick={this.activateLasers}>

Activate Lasers

</button>

需要传入一个事件处理函数，不能是字符串

注意，这么传不能带小括号，onClick={this.handle}，不能onClick={this.handle(e)}

function activateLasers (e) {

e.preventDefault();

}

使用e.preventDefault() 阻止默认事件

render(i) {

return (

<Child value={this.state.nihao}

onClick={() => this.handleClick(i)} />

);

}

注意，给组件注册onClick事件，这个onClick就有特殊意义了。不是整个组件点击任何部分都能触发，而是可以绑定给组件的元素。

其实吧，叫了个onClick，感觉和value是一样的，都是props的属性

在子组件内部，可以使用props.onClick绑定事件处理方法

function Child(props){

<button onClick={props.onClick}>登录</button>

}

* 事件绑定通常的做法是将事件处理函数声明为 class 中的方法

class Toggle extends React.Component {

constructor(props) {

this.handleClick = this.handleClick.bind(this);

}

handleClick() {

this.setState(state => ({

isToggleOn: !state.isToggleOn

}));

}

render() {

return <button onClick={this.handleClick}>show</button>

}

}

只要handleClick方法中有类似this.setState()的语句，用到了this，就必须在constructor中设置this.handleClick = this.handleClick.bind(this);

* 还有另外一种方法，用箭头函数绑定事件

<button onClick={ () => this.handleClick() }>show</button>

因为箭头函数中的this指向声明时的this，而不是使用时的

## key

function Nihao (props){

const arr = [324, 32, 3214, 432, 432];

let items = arr.map( a => (<li>这里是{a}</li>))

return (<ul>{items}</ul>)

}

报错：Warning: Each child in a list should have a unique "key" prop

let items = arr.map( a => (<li key={a.toString()} >这里是{a}</li>))

给予每个列表项一个独一无二的key，warning消失

## 表单

受控组件

this.state = {

name: "woshjinizuzong"

};

this.aaa=this.aaa.bind(this);

aaa(e){

this.setState({

name: e.target.value

})

}

render() {

return (

<input name="name" value={this.state.name} onChange={this.aaa}></input>

);

}

* 把<input>标签的value属性绑定this.state中的属性
* 此时运行会警告此组件是只读的。必须设置onChange事件回调或readOnly属性
* <input>标签onChange事件绑定回调函数。当然不要忘了在constructor中设置bind(this)
* 事件处理函数中，要用event.target.value提取当前空间的值。注意，即使在设置this.setState()之后，state.name和event.target.value仍然不相等，event.target.value是改变后的值，state.name是改变之前的值。有可能是因为setState是异步执行的，所以慢一点。但是使用setState(() => ({ name : e.target.value})) 形如这样的方法来同步执行setState，也还是不行。会直接报错，在有e.target的情况下不让这样子使用setState

this.setState({name: event.target.value.toUpperCase()});

如果这样设置，很奇怪。输入fa，输入框显示FA，console.log(this.state.name)显示F

## 状态提升

这一章教程举了一个很长很长的例子，用摄氏度和华氏度的同步转换，告诉了大概以下几个道理

* 只要组件的state和props发生改变，不管render()返回值中有没有用到{this.state}或{this.props}，都要重新绘制一遍DOM。所以形如

render(){

const x = this.state.name;

return <div>{x}</div>

}

在pros或state发生改变后，不管发生了什么改变，x值有没有变化，都要重新绘制一遍。当然如果state.name发生变化，就会将结果反映到DOM中

* 并列的子组件之间不能直接通信，必须通过状态提升，把值传给父组件，然后在父组件中处理，再将结果返回给子组件。

将值传给父组件也不能直接传递，必须通过把父组件的方法绑定到子组件的属性当中，然后利用子组件调用父组件传来的函数，将变化后的值以参数形式传回父组件，这样父组件就接收到了值的改变。举个例子

//父组件中

changeMethod(newValue){

this.setState({ name: newValue });

}

<Child method={this.changeMethod}</Child>

//子组件中

changeMethod(e){

this.props.method(e.target.value)

}

<input type="text" onChange={this.changeMethod}></input>

这样，子组件input改变一次值，父组件state.name就跟着变化一次

* .js文件中可以直接编写函数，相当于全局作用域。class里也可以直接用