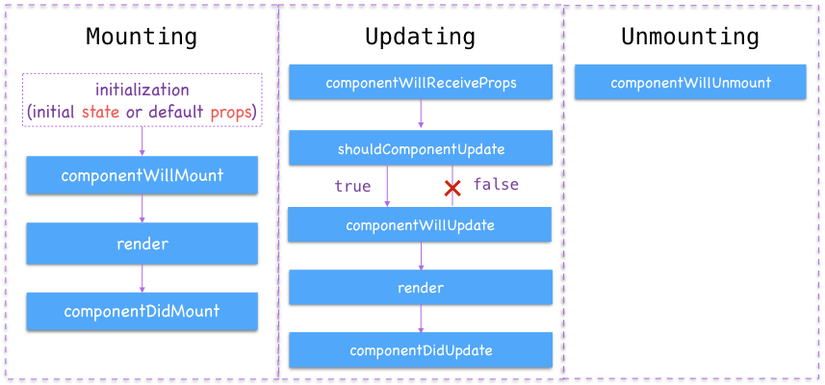
**Một số khái niệm cần biết trong React**

[Lifecycle](https://viblo.asia/tags/lifecycle)[Immutability](https://viblo.asia/tags/immutability)[es2015](https://viblo.asia/tags/es2015)[Data Binding](https://viblo.asia/tags/data-binding)[ReactJS](https://viblo.asia/tags/reactjs)[this](https://viblo.asia/tags/this)

**Giới thiệu**

Trong bài viết nay mình sẽ giới thiệu đến các bạn một số khái niệm quan trọng trong react. Đây là những khái niệm cần lắm rõ để trong quá trình phát triển với react tránh mắc phải việc mất thời gian cho những lỗi cơ bản mà mình đã gặp phải do chưa lắm vững kiến thức. Mình cũng ko lan man nữa. Phần đâu tiên là vòng đời của react component.

**Vòng đời trong React Component**

Khái niệm component lifecycle trong react là một trong những khái niệm quan trọng nhất, hiểu rõ lifecycle của component sẽ giúp bạn biết khi component được tạo, thay đổi, những method nào được gọi và làm những công việc gì. Để có thể hiểu rõ về lifecycle thì chúng ta cần nắm được sự khác nhau của lifecycle khi component được khởi tạo (mounting), khi state, props thay đổi(Updating) và khi unmount component như hình dưới đây:

1. Mounting

Trong giai đoạn nay sẽ bao gồm các phương thức khởi tạo state, props như sau :

constructor(props) {

super(props);

this.state = {

arrayIdea: [],

textIdea: "",

booleanIdea: false

};

// props sẽ được truyền từ class cha xuống

console.log(props);

}

Tiếp theo phương thức componentWillMount sẽ được gọi ở đây và lưu ý hàm này chỉ được gọi 1 lần duy nhất trước khi thực hiện phương thức render(). Phương thức rendersẽ trả về nội dung mà bạn muốn hiển thị bao gồm code html, css. Dưới đây là ví dụ một hàm render

render() {

return (

<div className="tile">

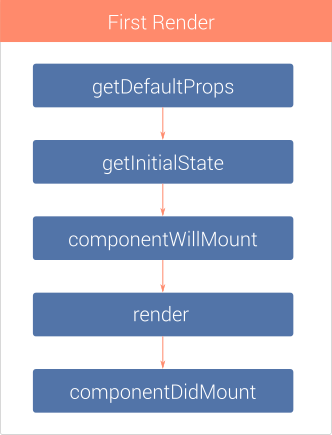
<h2>Hello world</h2>

</div>

);

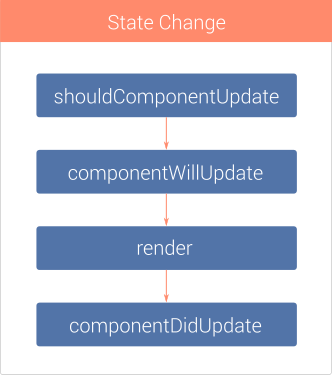
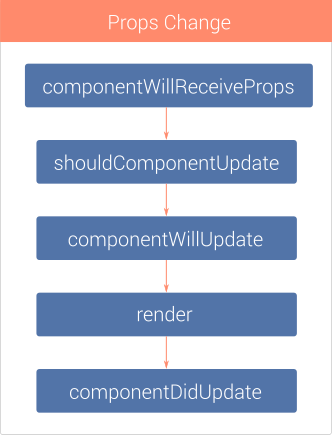
}

Ngay sau khi phương thức render được gọi thì phương thức componentDidMount sẽ được gọi. Tại đây khi mà mọi thứ đã được xây dựng hoàn chỉnh, ta có thể thực hiện các thao tác với component vừa mới tạo ví dụ: thực hiện các hàm bất đồng bộ để lấy dữ liệu, thao các với DOM.



1. Updating

Khi state thay đổi, hoặc props sẽ trigger đến những phương thức như sau:

Trường hợp thay khi props thay đổi( từ lớp cha chuyền dữ liệu mới lớp con) để có thể so sánh giữa props mới với props cũ, hoặc đơn giản là cập nhật lại state từ props mới ta có thể xử lý ngay tại đây như sau:

componentWillReceiveProps(nextProps) {

this.setState({

idea: nextProps.newIdea

});

//or showNotification()

}

Tiếp theo shouldComponentUpdate sẽ được thực hiện vả trả về kết quả true/false . Vậy ta có thể sử dụng để kiểm tra xem có thỏa mãn điều kiện ko nếu có thì trả về true để update còn không sẽ trả về false như sau:

shouldComponentUpdate(nextProps, nextState) {

console.log(nextState)

if (someCondition) {

return true;

}

return false;

}

Ngay sau khi shouldComponentUpdate return true thì componentWillUpdate sẽ được gọi:

componentWillUpdate(nextProps, nextState) {

// shouldComponentUpdate return true

}

Cuối cùng componentDidUpdate sẽ được gọi sau khi render xong.

componentDidUpdate(prevProps, prevState) {

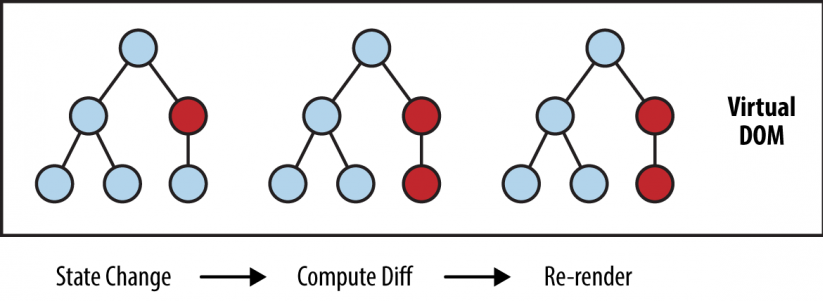
}

1. Unmounting

Unmouting là khi component được xóa khỏi DOM.Chỉ có một method duy nhất được gọi trước khi Component được xóa khỏi DOM. Được sử dụng trong trường hợp muốn cleanup event hay timer chẳng hạn.



**Immutability Helpers**

Trong javascript làm việc với dữ liệu có dạng immutable rất là khó so với các ngôn ngữ khác. React cũng không phải ngoại lệ và chịu ảnh hường đến hiệu năng nếu ko giải quyết được việc xử lý immutable data nguyên nhân là do cơ chế render() như sau

Mình cũng đã nói ở phần vòng đời của react component phương thức render() chỉ được gọi lại khi có sự thay đổi từ state hoặc props. Trong trường hợp nếu state trong class react có cấu trúc đơn giản thì việc so sánh sẽ không tốn kém nhưng tưởng tượng object của bạn có dạng như sau:

this.state.myData.x.y.z = 7;

// or...

this.state.myData.a.b.push(9);

Và bạn phải thực hiện lần lượt các thao tác sau để tạo new state

const newData = deepCopy(this.state.myData);

newData.x.y.z = 8;

newData.a.b.push(10);

...

function deepCopy(data) {}

Việc xử lý như trên gây ra tốn kém về hiệu năng quá lớn đồng thời sẽ có những trường hợp ko thể làm như trên được và sinh bug cho hệ thống. Do đó React cung cấp cho ta một thư viện để xử lý việc deepCopy và thay đổi dữ liệu vừa được copy: [Immutability Helpers](https://reactjs.org/docs/update.html)

Để sử dụng thư việc Immutability Helpers của react trong đường dẫn dự án ta sử dụng npm để cài đặt

npm install immutability-helper --save

Tiếp theo là các câu lệnh trong immutability-helper:

* {$push: array} thêm tất cả items vào cuối array được chỉ định
* {$unshift: array} thêm tất cả items vào đầu array được chỉ định
* {$splice: array of arrays} với một array được truyền vào để thửc hiện remove phần tử trong array được chỉ định vào thay thế phần tử đó bằng số mảng khác
* {$set: any} thay thế đối tượng chỉ định bằng đối tượng mới
* {$merge: object} dùng cho đối tượng dangh hash (key: value)
* {$apply: function}: áp dụng function cho đối tượng được chỉ định

Ví dụ: mình có viết thử 1 đoạn để cập nhât state của component sử dụng immutability-helper

import update from 'react-addons-update';

import React from 'react';

import ReactDOM from 'react-dom';

class Hello extends React.Component {

constructor(props) {

super(props);

this.state = {

ideas: [{

"id": 1,

"title": "A new cake recipe",

"body": "Made of chocolate"

},

{

"id": 2,

"title": "A twitter client idea",

"body": "Only for replying to mentions and DMs"

}]

};

}

helperReact = () => {

let index = 0;

let newData = update(this.state.ideas, {

[index]: {"title": {$set: "this is a new title"}}

});

this.setState({

ideas: newData

});

}

showState = () => {

console.log(this.state.ideas);

}

componentDidMount() {

this.helperReact();

}

render() {

this.showState();

return (

<div>Hello {this.props.name}</div>

);

}

}

ReactDOM.render(

<Hello name="World" />,

document.getElementById('container')

);

Hoặc bạn có thể chạy thử trên các ứng dụng editor online: đây là [ví dụ](https://codesandbox.io/s/54q3lwxj7l) mình viết trên online editor. Ngoài ra bạn có thể tham khảo các ví dụ khác trên trang chủ của react

**AUTOBIND không còn hoạt động với react + es6**

Trước tiên mình muốn nói qua về con trỏ this trong javascript. Trong các ngôn ngữ hướng đối tượng thông thường thì con trỏ this luôn để chỉ tới class chứa nó tuy nhiên trong javascript trở nên phức tạp hơn rất nhiều khi mà có thêm các khái niệm scoped function, callbacks, closured và promises khiến cỏn trỏ ta không thể kiểm soát được this đang tham chiếu ở đâu.

Quay lại với chủ đề, React đã chứng tỏ rằng nó là một thư viện thân thiện và tuyệt vời khi làm việc và việc xử lý từ khóa this cũng không ngoại lệ. React mặc định this sẽ được tham chiếu tới bối cảnh(context) bên trong của mỗi component React, ví dụ:

this.setState({ someKey: someValue })

Tuy nhiên vấn đề sẽ xảy ra ki bạn viết code như sau:

this.setState({ dataFetched: false })

fetch('/data').then(function fetchComplete() {

this.setState({ dataFetched: true })

})

Bạn đã thấy vấn đề xảy ra không? khi mà dòng đầu chạy sẽ không có lỗi gì xảy ra. Tuy nhiên khi chạy đến dòng setState thứ 2 kết quả lỗi sẽ bị raise. Nguyên nhân là dó con trỏ this không còn tham chiếu tới React component nữa mà tham chiếu vào hàm fetchComplete.

Trước khi giải quyết vấn đề trên mình xin quay lại một chút kiến thức về javascript. Giả sử ta định nghĩa một đối tượng như sau:

//Định nghĩa 1 đối tượng với phương thức hiển thị thuộc tính ra màn hình

var Student = {

name: 'John',

printName: function() {

console.log(this.name);

}

};

//Bắt sự kiện click chuột lên button, sẽ thực thi hàm hiển thị tên của Student lên màn hình

$('button').click(Student.printName); //kq: undefined

Kết quả hiện thị sẽ là undefined nguyên nhân là do con trỏ this không còn trỏ vào đối tượng Student nữa mà nó trỏ vào đối tượng $(‘button’). Để đoạn code trên chạy đúng như mong muốn thì ta phải đảm bảo context của hàm callback Student.printName phải được tham chiếu tới đối tượng Student. Cụ thể ta sẽ sử dụng hàm bind() để gắn context vào hàm callback đó.

$('button').click(Student.printName.bind(Student));

Các bạn có thể thử tại codepen:

Đó cũng là lí do tại sao khi viết code trong es6 mặc dù cú pháp hoàn toàn đúng nhưng không thể chạy như mong muốn được

class ExampleComponent extends React.Component {

onClickHander() {

this.setState({ clicked: true });

}

render() {

return <div onClick={this.onClickHander}>Click me</div>;

}

Bạn có thể kiểm chứng tại [đây](https://codesandbox.io/s/or4qx720z). Cuối cùng là một số giải pháp cho vấn đề trên

1. Sử dụng bind khi render

render() {

return <div onClick={this.onClickHander}>AAAA</div>;

}

1. Sử dụng arrow function trong JSX

render() {

return (

<div onClick={() => this.onClickHandler()}>AAAA</div>;

)

};

1. Bind ngay khi khởi tạo

class ExampleComponent extends React.Component {

constructor() {

super();

this.onClickHandler = this.onClickHandler.bind(this);

}

...

}

1. Sử dụng thư viện cho phép autobinding

import autobind from 'autobind-decorator';

class ExampleComponent extends React.Component {

@autobind

update() {

...

}

...

}

1. Sử dụng arrow function làm class method

import React from 'react';

class ExampleComponent extends React.Component {

update = () => {

...

}

...

}câu hỏi phỏng vấn

## 1. React hoạt động như thế nào ? Virtual-DOM hoạt động như thế nào trong React ?

React tạo một DOM ảo. Khi trạng thái trên component thay đổi, trước tiên, nó sẽ chạy thuật toán "diffing" (so sánh) và xác định những gì đã thay đổi trong virtual DOM. Sau đó, bước thứ 2 là đối chiếu, react sẽ cập nhật DOM với kết quả của việc so sánh ở trên.

HTML DOM luôn có cấu trúc dạng cây được cho phép bởi cấu trúc của tài liệu HTML. Ngày nay, cây DOM rất lớn bởi vì các ứng dụng lớn ra đời. Từ khi chúng ta đẩy mạnh hơn vào các ứng dụng web động (Single Page Application), chúng ta cần sửa đổi cây DOM không ngừng và nhiều. Và đây là một vấn đề lớn của hiệu suất và phát triển ứng dụng.

Virtual DOM là một bản thu gọn của HTML DOM. Nó rất nhẹ và tách rời khỏi các chi tiết triển khai dành riêng cho trình duyệt. Tuy Virtual DOM không được phát minh bởi React nhưng nó được sử dụng và cung cấp miễn phí.

Bất cứ khi nào ReactComponent thay đổi trạng thái, thuật toán so sánh trong React sẽ hoạt động và kiểm tìm ra phần tử đã thay đổi. Sau đó nó sẽ cập nhật DOM với kết quả cuả việc so sánh trên. Vấn đề là nó được thực hiện nhanh hơn so với DOM thông thường do chỉ thay đổi những thứ cần thiết mà không cần phải load lại toàn bộ DOM.

## 2. JSX là gì ?

JSX là một phần mở rộng cú pháp cho JavaScript và đi kèm với toàn bộ sức mạnh vốn có của JavaScript. JSX tạo ra các "elements" của React. Bạn có thể nhúng bất kì biểu thức JavaScript nào trong JSX bằng cách đóng gói trong dấu ngoặc nhọn. Sau khi biên dịch, JSX sẽ trở thành các đối tượng JavaScript thông thường. Điều này có nghĩa là bạn có thể sử dụng JSX bên trong các câu lệnh if và cho các vòng lặp, gán biến, chấp nhận làm đối số và retun trong các hàm.

Theo dõi ví dụ bên dưới, một ví dụ đơn giản về cú pháp JSX và một ví dụ tương tự khi không sử dụng JSX.

const element = (

<h1 className="greeting">

Hello world!

</h1>

)

Tương tự như trên sử dụng React.createElement:

const element = React.CreateElement(

'h1',

{"className": "greeting"},

'Hello world!'

)

## 3. React.createClass là gì ?

React.createClass cho phép chúng ta tạo ra các component "class". Khi sử dụng clreateClass(), chúng ta truyền vào một đối tượng làm đối số. Do đó, chúng ta có thể viết một component sử dụng createClass tương tự như:

import React from 'React';

const Contacts = React.createClass({

render() {

return (

<div></div>

)

}

});

export default Contacts;

Sử dụng ES6 class để viết mộc component có một số thay đổi. Thay vì sử dụng phương thức trong thư viện của react, chúng ta sẽ mở rộng một lớp mà thư viện định nghĩa, Contacts component sẽ được viết dưới dạng:

import React from 'react';

class Contacts extends React.Component({

constructor(props){

super(props);

}

render() {

return (

<div></div>

)

}

});

constructor() là một chức năng đặc biệt trong một lớp JavaScript. JavaScript gọi constructor() bất cứ khi nào một đối tượng được tạo thông qua một lớp.

## 4. Sự khác nhau giữa ReactDOM và React

Trước phiên bản v0.14, tất cả chức năng ReactDOM là một phần của React. Nhưng sau đó, React và ReactDOM được chia thành 2 thư viện khác nhau.

Giống như cái tên của nó, ReactDOM là chất kết dính giữa React và DOM. Thông thường , chúng ta chr sử dụng nó cho một mục đích đó là gắn kết với ReacDOM. Một số trường hợp sử dụng ReactDOM là ReactDOM.findDOMNode() để có thể truy cập trực tiếp vào phần tử DOM.

Trong tất cả các trường hợp khác, đó sẽ là React. Chúng ta sử dụng React để định nghĩa và tạo các elements, lifecycle hooks, .. để hoàn thiện một ứng dụng React.

## 5.Class component và Functional component

Class component cho phép chúng ta sử dụng các tính năng bổ sung để có thể can thiệp vào state và vòng đời của một đối tượng trong React. Ngoài ra, có thẻ cho phép component có thể trực tiếp truy cập vào store và lưu trữ lại trạng thái của ứng dụng.

Khi component được truyền props và render chúng ra trang, đó là stateless component, đây là thành phần phi trạng thái, mà chức năng pure function có thể được sử dụng. Chúng cũng được gọi với cái tên dump component hoặc presentational component.

Từ câu hỏi trước, chúng ta có thể gọi Booklist component là một functional components và là một stateless

// Booklist.js

import React from "react";

const Booklist = books => (

<ul>

{books.map(

)}

</ul>

)

Mặt khác, BookListContainer sẽ là một class component và sử dụng lại functional component Booklist ở trên.

## 6. Sự khác nhau giữa state và props

State là trạng thái cấu trúc dữ liệu được khởi tại với dự liệu ban đầu khi mà Componentkhởi tạo. Nó có thể thay đổi được theo thời gian, chủ yếu là kết quả của các tương tác người dùng với DOM.

props là một cấu hình của Component. props là cách mà các component nói chuyện với nhau. Chúng được nhận từ component cha và không thay đổi sau khi nhận được giá trị. Một Component không thể thay đổi props của nó, nhưng nó có trách nhiệm đẩy props cho các Component con của nó. Props không chỉ lbao gồm dữ liệu - nó có thể là các callback function.

Cũng có trường hợp chúng ta có thể có các props mặc định để có thể sử dụng ngay cả khi chúng không được truyền từ component cha xuống.

Class SearchBar extends Component {

constructor(props){

supper(props);

this.state = { term: '' };

}

onInputChange(term){

this.setState(term);

this.props.onSearchTermChange(term);

}

render() {

return(

<div className="search-bar">

<input

value={this.state.term}

onChange={event => this.onInputChange(event.target.value)} />

</div>

)

}

}

props và state làm những việc tương tự nhay nhưng lại được sử dụng theo những cách khác nhau. props được sử dụng để truyền dữ liệu từ component cha xuống con hoặc được sử dụng cho chính nó. Chúng bất biến và do đó sẽ không được thay đổi. stte được sử dụng cho dữ liệu có thể thay đổi hoặc dữ liệu sẽ thay đổi. Điều này đặc biệt hữu ích khi tương tác với người dùng.

## 7. Controlled components là gì ?

Trong HTML, các thành phần của form tương tự như <input>, <textarea>, và <select>thường được duy trì trạng thái của riêng chúng và được cập nhật khi người dùng tương tác với ứng dụng. Khi người dùng gửi một form lên (submit forrm), các giá trị nói trên được gửi cùng với form. Với React thì nó hoạt động khác nhau. Component chứa form sẽ giữ giá trị của người dùng trong state của nó và sẽ render ra component với giá trị đó khi được gọi lại. Ví dụ : onChange sẽ bị mất giá trị vì state sẽ được cập nhật. Một phần từ form có giá trị được React kiểm soát theo cách này được gọi là controlled component

Với một controlled component, mọi sự thay đổi trạng thái sẽ có chức năng xử lý liên quan. Điều này làm cho nó đơn giản để sửa đổi hoặc xác nhận đầu vào của người dùng.

## 8. Higher-order component ?

Một component bậc cao (higher-order component) HOC là một kĩ thuật nâng cao trong React để sử dụng lại các component logic. HOCs không phải là một thành phần của React API. Chúng là một mô hình xuật hiện từ bản chất cấu thành vốn có của React.

Một higher-order component là một phương thức lấy một component và trả về mộtcomponent mới.

HOCs giúp bạn có thể tái sử dụng lại code, logic và cấu trúc ứng dụng. HOCs rất phổ biến trong các thư viện của React và bên thứ ba. Phổ biến nhất là connect function của Redux. Ngoài việc chia sẻ đơn giản các thư viện tiên ích và các thành phần đơn giản, HOCs là một cách tốt để có thể chia sẻ hành vi giữa các component. Nếu bạn thấy mình viết rất nhiều mã ở những nơi khác nhau làm cùng một công việc, bạn có thể tái cấu trúc thành HOC để có thể sử dụng lại.

Tham khảo thêm: <https://reactjs.org/docs/higher-order-components.html>

## 9. create-react-app ?

create-react-app là một CLI (Command Line Interface) chính thức cho react để tạo các ứng dụng React không có cấu hình xây dựng.

Thật đơn giản, chúng ta không cần phải cài đặt hoặc cấu hình các tools như Webpackhay Babel để có thể sử dụng React. Chúng được cấu hình sẵn để chúng ta có thể tập trung vào công việc chính, đó là code. Chúng ta có thể dễ dàng cài đặt và sử dụng giốngnhw bất cứ modules nào khác. Và chỉ với một command để có thể bắt đầu một project React.

crate-react-app my-app

Trong đó đã bao gồm mọi thứ cần thiết để có thể xây dựng một ứng dụng React:

* Hỗ trợ cú pháp React, JSX, ES6 và Flow
* Ngôn ngữ bổ sung ngoài ES6 như toán tử trải rộng đối tượng.
* CSS tự động kết hợp, vì vậy bạn không cần tiền tố -webkit- hoặc các tiền tố khác.
* Một máy chủ phát triển trực tiếp cảnh báo về những sai lầm phổ biến.
* Một tập lệnh xây dựng để đóng gói JS, CSS và hình ảnh để sản xuất, với các giá trị băm và mã nguồn.

## 10. Redux là gì ?

Ý tưởng cơ bản của Redux là toàn bộ trạng thái của ứng dụng được giữ trong một store. store là một object javascript đơn giản. Cách duy nhất để thay đổi trạng thái là sử dụng actions trong ứng dụng và sau đó viết các reducers cho actions để có thể thay đổi trạng thái đó. Toàn bộ quá trình chuyển đổi trạng thái được giữ bên trong các reducers và khong nên có bất kì các tác dụng phụ nào khác như gửi request, đẩy ra DOM.,.. bên trong reducers

Redux dựa trên ý tưởng rằng chỉ nên có một nguồn đáng tin cậy trong ứng dụng của bạn, có thể là trạng thái UI giống như tab nào đang hoạt động hoặc trạng thái chi tiết profile người dùng.

first\_name: "Sun",

last\_name: "Asterisk",

age: 7

Tất các dữ liệu này được giữ lại bởi Redux bên trong một bao đóng mà chúng ta gọi là store. Redux cũng cung cấp cho chúng ta một phương thức để có thể nhận và khởi tạo store gọi là createStore(x).

createStore chấp nhạn một hàm khác, x là đối số.

Chứng năng được thông qua có trách nhiệm trả về lại trạng thái của ứng dụng tịa thời điểm đó, sau đó được duy trì trong store. Điều này thông qua chức năng được gọi là reducer.

Một ví dụ điển hình về reducer:

export default function reducer(state= {}, action){

return state;

}

store chỉ có thể được cập nhật khi chúng ta dispatching một action. Ứng dụng sử gửi một action, được chuyển vào reducer, reducer sẽ trả về một thể hiện mới của state; sau đó, store sẽ thông báo cho ứng dụng của chúng ta hoặc có thể render lại từ đầu.