ReactJS

Công dụng

Giải quyết được vấn đề của tầng View

- Giải quyết vấn đề của tầng View trong mô hình MVC (Model-View-Controller)
- Giúp viết mã Javascript dễ dàng hơn với JSX
- Dễ dàng hơn khi viết mã, và thân thiện hơn với các lập trình viên

"Thành phần hóa" giao diện

- Cho phép lập trình viên tạo ra các Component tương ứng với các phần của giao diện
- Có thể tái sử dụng, hoặc kết hợp với các Component khác để tạo ra một giao diện hoàn chỉnh
- Chìa khóa giải quyết vấn đề khó khăn khi dự án ngày càng lớn

Tăng hiệu năng với Virtual-DOM

- Tạo ra thay đổi trên một mô hình DOM ảo (Virtual DOM)
- Mang lại hiệu năng cho ứng dụng

Thân thiện với SEO

Dễ dàng viết UI Testcases.

Giới thiệu

Khái niệm

- React (ReactJS, React.js) là một thư viện Javascript mã nguồn mở để xây dựng các thành phần giao diện có thể tái sử dụng

Được tạo ra bởi Jordan Walke, một kỹ sư phần mềm tại Facebook

Lần đầu tiên được triển khai cho ứng dụng Newsfeed của Facebook năm 2011

Được triển khai cho Instagram.com năm 2012

Mở mã nguồn (open-sourced) tại JSConf US tháng 5 năm 2013

Mục tiêu chính

- nhanh, đơn giản, hiệu năng cao và dễ dàng mở rộng

Cách sử dụng

Tạo project

- create-react-app <name project>

Webpack

Ưu điểm

- Có khả năng xử lý asset tĩnh, đặc biệt là CSS. Tất cả các hình ảnh và CSS được đóng gói vào 1 thư mục "dist/"
- Dễ dàng thực hiện chia tách mã nguồn
- Kiểm soát quá trình xử lý asset
- Giảm nguy cơ deploy code mà lại thiếu file ảnh và đăng nhầm file CSS cũ giúp quá trình triển khai sản phẩm được ổn định

Cài đặt

Trên Global

- Trên Yarn: yarn global add webpack webpack-cli
- Trên Npm: npm i -g webpack webpack-cli
- Khi đã hoàn tất, có thể cho chạy lệnh: webpack-cli

Trên máy Local

- Trên Yarn: yarn add webpack webpack-cli -D
- Trên Npm: npm i webpack webpack-cli –save-dev
- Khi đã hoàn tất cài đặt, thêm vào file package.json đoạn code sau.

React router

Ưu điểm

- Cho phép trang web mỗi khi thay đổi, chỉ tải lại các phần cần thiết
- Giúp bạn hiển thị component tương ứng với từng đường dẫn
- Route sẽ chỉ định component tương ứng hiển thị lên giao diện
- Giúp tiết kiệm thời gian hơn

Cách dùng

Cài đặt react-router-dom

- npm install react-router-dom

- Mở tệp index.js và nhập BrowserRouter, Switch và Route từ react -router-dom
- Vào tệp index.js
- Tạo các liên kết nav cơ bản từ react-router, vào tệp header.js
- Chạy dự án và làm mới trình duyệt để xem các thay đổi

Thành phần chính

- Router
- Route matchers
- Chuyển hướng

Các đặc trưng

JSX

Ưu điểm

- Tốc độ nhanh hơn vì JSX thực hiện tối ưu hóa khi biên dịch mã sang JavaScript.
- Hầu hết các lỗi đều có thể được phát hiện ngay trong quá trình biên dịch mã.
- JSX giúp viết các mẫu (templates) nhanh và dễ dàng hơn.

JSX trong React

- Gán một biểu thức
- JSX là một biểu thức
- Chỉ định attributes với JSX
- Phần tử con trong JSX

Redux

Cấu trúc

- Store : Là 1 object lưu trữ state của toàn bộ ứng dụng, cho phép truy cập State qua getState(), update State qua dispatch(action).
- Action creators : Là nơi tạo ra các action dùng để mô tả event do người dùng tạo ra.
- Reducer : Là 1 fuction nhận đầu vào là state và các mô tả về event và dựa trên đó để trả về state tiếp theo nhưng ko thay đổi state cũ
- View: Hiển thị dữ liệu được cung cấp bởi Store.

Cài đặt

- Sau khi khởi tạo một dự án ReactJS, để có thể sử dụng Redux chúng ta cần phải cài đặt 2 module là redux và react-redux bằng cách sử dụng npm

Tích hợp Redux vào ReactJS

- Chia nhỏ Redux ra nhiều thư mục
- Khởi tạo các hằng
- Khởi tạo actions (Actions là một object chứa các hành động muốn gửi đến reducers.)
- Khởi tạo reducers (Reducers có nhiệm vụ thay đổi state của ứng dụng dựa trên từng hành động được gửi đến)
- Tích hợp Redux
- Lấy và cập nhật giá trị của state từ Store

Single-way data flow (Luồng dữ liệu một chiều)

Virtual DOM

Khái niệm

- DOM hay Document Object Model (Mô hình Đối tượng Tài liệu), là một chuẩn được định nghĩa bởi W3C dùng để truy xuất và thao tác trên code HTML hay XML bằng các ngôn ngữ lập trình thông dịch (scripting language) như Javascript.
- Virtual DOM là một định dạng dữ liệu JavaScript nhẹ được dùng để thể hiện nội dung của DOM tại một thời điểm nhất định nào đó.

Công dụng

- Sử dụng thuật toán so sánh thay đổi để biết những gì đã thay đổi
- Cập nhật danh mục DOM đồng thời
- Cập nhật hàng loạt vào DOM

Props và state

- Props (properties) là 1 đối tượng lưu trữ các giá trị của các attribute (thuộc tính) của một thẻ (Tag).Là cách mà component có thể nhận được các giá trị của thuộc tính truyền vào từ bên ngoài và là cách mà các component có thể nói chuyện với nhau.
- State State là dữ liệu động cho phép một component theo dõi thông tin thay đổi giữa các kết xuất (render) và làm cho nó có thể tương tác.

Components

Tổng quan

- Chức năng : components trong React App thường nhận về các props và trả về React elements từ đó hiển thị ra cho UI.
- Components trong React thường được viết theo 2 loại chính đó là functional component và class components.

Khởi tạo

- khởi tạo một thư mục có tên components trong thư mục src để chứa tất cả các component trong dự án.
- Functional Component
- Class Component
- Hỗ trợ xây dựng các ứng dụng SPA (Single-Page Application)

Lifecycle Of Components

Mounting

- constructor() method được gọi đầu tiên khi component được khởi tạo, và đây cũng là nơi chúng ta initial các state của component.
- getDerivedStateFromProps() method được gọi ngay trước khi render element vào DOM.
- render() là method bắt buộc và nó thực hiện việc đưa các output HTML vào DOM.
- componentDidMount() method được gọi sau khi component đã được render.

Updating

- getDerivedStateFromProps() method thực hiện công việc set lại state theo prop được initial trước đó.
- shouldComponentUpdate() method có thể trả về giá trị boolean để chỉ định rằng có thực thi việc rendering hay không (giá trị mặc định: true).
- render() method thực hiện render lại HTML khi một component đã được update
- Với getSnapshotBeforeUpdate() method ta có thể lấy được giá trị props và state trước khi update.
- componentDidUpdate() method được gọi sau khi component đã được updated

Unmounting

- componentWillUnmount() method được gọi khi một component được remove khỏi DOM.

ReactJS

1. Công dụng

1.1. Giải quyết được vấn đề của tầng View

1.1.1. Giải quyết vấn đề của tầng View trong mô hình MVC (Model-View-Controller)

1.1.2. Giúp viết mã Javascript dễ dàng hơn với JSX

1.1.3. Dễ dàng hơn khi viết mã, và thân thiện hơn với các lập trình viên

1.2. "Thành phần hóa" giao diện

1.2.1. Cho phép lập trình viên tạo ra các Component tương ứng với các phần của giao diện

1.2.2. Có thể tái sử dụng, hoặc kết hợp với các Component khác để tạo ra một giao diện hoàn chỉnh

1.2.3. Chìa khóa giải quyết vấn đề khó khăn khi dự án ngày càng lớn

1.3. Tăng hiệu năng với Virtual-DOM

1.3.1. Tạo ra thay đổi trên một mô hình DOM ảo (Virtual DOM)

1.3.2. Mang lại hiệu năng cho ứng dụng

1.4. Thân thiện với SEO

1.5. Dễ dàng viết UI Testcases.

2. Giới thiệu

2.1. Khái niệm

2.1.1. React (ReactJS, React.js) là một thư viện Javascript mã nguồn mở để xây dựng các thành phần giao diện có thể tái sử dụng

2.2. Được tạo ra bởi Jordan Walke, một kỹ sư phần mềm tại Facebook

2.3. Lần đầu tiên được triển khai cho ứng dụng Newsfeed của

Facebook năm 2011

2.4. Được triển khai cho Instagram.com năm 2012

2.5. Mở mã nguồn (open-sourced) tại JSConf US tháng 5 năm 2013

2.6. Mục tiêu chính

2.6.1. nhanh, đơn giản, hiệu năng cao và dễ dàng mở rộng

3. Cách sử dụng

3.1. Tạo project

3.1.1. create-react-app <name project>

4. Webpack

4.1. Ưu điểm

4.1.1. Có khả năng xử lý asset tĩnh, đặc biệt là CSS. Tất cả các hình ảnh và CSS được đóng gói vào 1 thư mục “dist/”

4.1.2. Dễ dàng thực hiện chia tách mã nguồn

4.1.3. Kiểm soát quá trình xử lý asset

4.1.4. Giảm nguy cơ deploy code mà lại thiếu file ảnh và đăng nhầm file CSS cũ giúp quá trình triển khai sản phẩm được ổn định

4.2. Cài đặt

4.2.1. Trên Global

4.2.1.1. Trên Yarn: yarn global add webpack webpack-cli

4.2.1.2. Trên Npm: npm i -g webpack webpack-cli

4.2.1.3. Khi đã hoàn tất, có thể cho chạy lệnh: webpack-cli

4.2.2. Trên máy Local

4.2.2.1. Trên Yarn: yarn add webpack webpack-cli -D

4.2.2.2. Trên Npm: npm i webpack webpack-cli –save-dev

4.2.2.3. Khi đã hoàn tất cài đặt, thêm vào file package.json đoan code sau.

```
{
 //...
 "scripts": {
    "build": "webpack"
  }
}
```

5. React router

> React Router là một API cho các ứng dụng React

5.1. Ưu điểm

5.1.1. Cho phép trang web mỗi khi thay đổi, chỉ tải lại các phần cần thiết

5.1.2. Giúp bạn hiển thị component tương ứng với từng đường dẫn

5.1.3. Route sẽ chỉ định component tương ứng hiển thị lên giao diện

5.1.4. Giúp tiết kiệm thời gian hơn

5.2. Cách dùng

```
import { BrowserRouter as Router, Switch, Route } from "react-router-dom";



```

5.2.1. Cài đặt react-router-dom

5.2.1.1. npm install react-router-dom

5.2.2. Mở tệp index.js và nhập BrowserRouter, Switch và Route từ react -router-dom

```
import { BrowserRouter as Router, Switch,
Route } from "react-router-dom";
```

5.2.3. Vào tệp index.js

```
import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import { BrowserRouter as Router, Switch, Route} from
"react-router-dom";

import Header from './components/header'

import Footer from './components/footer'

import About from './components/about'

import Service from './components/service'

const Routing = () => {

  return(

    <Router>

      <Header/>

      <Switch>

        <Route exact path="/" component={App} />

        <Route path="/about" component={About} />

        <Route path="/service" component={Service} />
```

```
      </Switch>
      <Footer/>
    </Router>
  )
}
ReactDOM.render(
  <React.StrictMode>
    <Routing />
  </React.StrictMode>,
  document.getElementById('root')
);
```

5.2.4. Tạo các liên kết nav cơ bản từ react-router, vào tệp header.js

```
import React from 'react'

import {Link} from 'react-router-dom'

export default function header() {

    return (

        <>

            <ul>

                <li><Link to="/">Home</Link></li>

                <li><Link to="/about">About</Link></li>

                <li><Link to="/service">Service</Link>
</li>

            </ul>

        </>

    )

}
```

5.2.5. Chạy dự án và làm mới trình duyệt để xem các thay đổi

5.3. Thành phần chính

5.3.1. Router

như là <BrowserRouter> và <HashRouter>

5.3.2. Route matchers

> như là <Route> và <Switch>

5.3.3. Chuyển hướng

> như <Link>, <NavLink>, và <Redirect>

6. Các đặc trưng

6.1. JSX

> JSX (Javascript XML). Là một template languges nhưng lại mang hầu hết tính năng của Javascript

6.1.1. Ưu điểm

6.1.1.1. Tốc độ nhanh hơn vì JSX thực hiện tối ưu hóa khi biên dịch mã sang JavaScript.

6.1.1.2. Hầu hết các lỗi đều có thể được phát hiện ngay trong quá trình biên dịch mã.

6.1.1.3. JSX giúp viết các mẫu (templates) nhanh và dễ dàng hơn.

6.1.2. JSX trong React

> Trong **ReactJS** không bắt buộc phải sử dụng **JSX** nhưng hầu hết mọi người đều sử dụng nó bởi đây là cách hữu ích nhất để làm việc với các UI bên trong Javascript code.

6.1.2.1. Gán một biểu thức

```
//gán biến name vào trong JSX
  const name = 'Freetuts.net';
  const element = <h1>Welcome to {name}</h
  1>;

  ReactDOM.render(
     element,
     document.getElementById('root')
  );
```

6.1.2.2. JSX là một biểu thức

```
Sau khi complie, các đoạn đoạn mã JSX sẽ
như các object Javasript thông thường, có thể
gọi hoặc làm bất cứ gì với nó.
//Hàm trả về một JSX
  function sayHi(name) {
       if (name) {
            return <p>Xin chào, {name} !</
  p>
       }else{
            return <p>Xin chào bạn !</p>
       }
  }
```

6.1.2.3. Chỉ định attributes với JSX

```
//Cú pháp giống như HTML thông thường
const element = <div tabIndex="0"></div>;
// Hoặc bằng biểu thức javascript
const element = <img src={user.avatarUrl}></
img>;
```

6.1.2.4. Phần tử con trong JSX

```
// Nếu chỉ có 1 tag
const element = <img src={user.avatarUrl} />
;
// Trường hợp trong tag có nhiều phần tử con cần phải bọc ngoài nó bằng một JSX tags
  const element = (
    <div>
      <h1>Hello</h1>
      <p>Welcome to Freetuts</p>
    </div>
  );
```

6.2. Redux

Redux là 1 thư viện giúp quản lý các **state** 1 cách tốt hơn. Thay vì phải truyền state qua từng Component thì **Redux** sẽ tạo ra **1 store duy nhất** dùng để thay đổi dữ liệu.

6.2.1. Cấu trúc

6.2.1.1. Store : Là 1 object lưu trữ state của toàn bộ ứng dụng, cho phép truy cập State qua getState(), update State qua dispatch(action).

```
import {createStore} from 'redux';
import rootReducers from './reducers';

const store = createStore(rootReducers);
```

6.2.1.2. Action creators : Là nơi tạo ra các action dùng để mô tả event do người dùng tạo ra.

```
export const selectPost = (post) => {
  return{
    type: "SELECT_POST",
    payload: post
  }
}
```

6.2.1.3. Reducer : Là 1 fuction nhận đầu vào là state và các mô tả về event và dựa trên đó để trả về state tiếp theo nhưng ko thay đổi state cũ

```
export default function activePostReducer (s
tate = initialState, action) {
  switch (action.type){
    case "SELECT_POST":
      return action.payload;
    default:
      return state;
  }
}
```

6.2.1.4. View: Hiển thị dữ liệu được cung cấp bởi Store.

6.2.2. Cài đặt

6.2.2.1. Sau khi khởi tạo môt dự án ReactJS, để có thể sử dụng Redux chúng ta cần phải cài đặt 2 module là redux và react-redux bằng cách sử dụng npm

```
npm install redux react-redux --save
```

6.2.3. Tích hợp Redux vào ReactJS

6.2.3.1. Chia nhỏ Redux ra nhiều thư mục

```
src/
....const/
        index.js
.....actions/
        index.js
....reducers/
        ...
        index.js
----components

....App.js
```

6.2.3.2. Khởi tạo các hằng

```
// const/index.js
export const ADD_NEW_NOTE = "ADD_NEW_NOTE";
```

6.2.3.3. Khởi tạo actions (Actions là một object chứa các hành động muốn gửi đến reducers.)

```
// actions/index.js
import { ADD_NEW_NOTE, REMOVE_NOTE, EDIT_NOTE } from "../const/index";
export const actAddNote = (content) => {
    return {
      type: ADD_NEW_NOTE,
      content,
    };
};
```

6.2.3.4. Khởi tạo reducers (Reducers có nhiệm vụ thay đổi state của ứng dụng dựa trên từng hành động được gửi đến)

```
// reducers/noteReducers.js
import { ADD_NEW_NOTE, REMOVE_NOTE, EDIT
_NOTE } from "../const/index";

const noteReducers = (state = [], action
) => {
   switch (action.type) {
     case ADD_NEW_NOTE:
       const generateID = new Date().ge
tTime();
       state = [...state, { id: generate
ID, content: action.content }];
       return state;
     default:
       return state;
   }
};

export default noteReducers
```

6.2.3.5. Tích hợp Redux

```
// src/index.js

import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import * as serviceWorker from "./servic
eWorker";


import { Provider } from "react-redux";
import { createStore } from "redux";

//Gọi reducers
import reducers from "./reducers/index";
//Tạo store
const store = createStore(reducers);

ReactDOM.render(
   <React.StrictMode>
      <Provider store={store}>
         <App />
      </Provider>
   </React.StrictMode>,
   document.getElementById("root")
);
serviceWorker.unregister();
```

6.2.3.6. Lấy và cập nhật giá trị của state từ Store

```
// src/App.js

//Import kết nối tới react-redux
import { connect } from 'react-redux'
//Import action dùng để dispatch
```

```
import {actAddNote} from './actions/index'

function App(props) {

    return (
        ...
    );
}

//Gán dispatch thành props
const mapDispatchToProps = (dispatch) =>
  {
    return {
      addNote: (content) => {
        dispatch(actAddNote(content))
      }
    }
}

//Gán giá trị của state thành props
const mapStateToProps = (state, ownProps
) => {
    return {
      note: state.note
    }
}

//Export component với két nối redux.
export default connect(mapStateToProps,
 mapDispatchToProps)(App)
```

6.3. Single-way data flow (Luồng dữ liệu một chiều)

6.4. Virtual DOM

6.4.1. Khái niệm

6.4.1.1. DOM hay Document Object Model (Mô hình Đối tượng Tài liệu), là một chuẩn được định nghĩa bởi W3C dùng để truy xuất và thao tác trên code HTML hay XML bằng các ngôn ngữ lập trình thông dịch (scripting language) như Javascript.

6.4.1.2. Virtual DOM là một định dạng dữ liệu JavaScript nhẹ được dùng để thể hiện nội dung của DOM tại một thời điểm nhất định nào đó.

6.4.2. Công dụng

6.4.2.1. Sử dụng thuật toán so sánh thay đổi để biết những gì đã thay đổi

6.4.2.2. Cập nhật danh mục DOM đồng thời

6.4.2.3. Cập nhật hàng loạt vào DOM

6.5. Props và state

6.5.1. Props (properties) là 1 đối tượng lưu trữ các giá trị của các attribute (thuộc tính) của một thẻ (Tag).Là cách mà component có thể nhận được các giá trị của thuộc tính truyền vào từ bên ngoài và là cách mà các component có thể nói chuyện với nhau.

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

const element = <Welcome name="ReactJS" />;
ReactDOM.render(
  element,
  document.getElementById('root')
);
```

6.5.2. State State là dữ liệu động cho phép một component theo dõi thông tin thay đổi giữa các kết xuất (render) và làm cho nó có thể tương tác.

6.6. Components

> **Components** giúp phân chia các UI (giao diện người dùng) thành các phân nhỏ để dễ dàng quản lý và tái sử dụng.

6.6.1. Tổng quan

6.6.1.1. Chức năng : components trong React App thường nhận về các props và trả về React elements từ đó hiển thị ra cho UI.

6.6.1.2. Components trong React thường được viết theo 2 loại chính đó là functional component và class components.

6.6.2. Khởi tạo

6.6.2.1. khởi tạo một thư mục có tên components trong thư mục src để chứa tất cả các component trong dự án.

```
public/
node_modules/
src/
-----components/
---------------Components sẽ viết ở trong thư
mục này
-----App.js
-----index.js
---- vv....
packages.json
packages-lock.json
```

6.6.2.2. Functional Component

```
//Import react vào trong dự án
import React from "react";

const Welcome = function(props) {
    return (
      <div>
        <h1>Welcome ! I am a functional c
omponent </h1>
      </div>
    )
}

export default Welcome;
```

6.6.2.3. Class Component

```
import React, { Component } from "react"
;
class Welcome extends Component {
    render() {
      return (
        <div>
          <h1>Welcome ! I am a class comp
onent </h1>
        </div>
      );
    }
}
export default Welcome;
```

6.6.2.4. Hỗ trợ xây dựng các ứng dụng SPA (Single-Page Application)

6.7. Lifecycle Of Components

Mỗi component trong React đều có một vòng đời với 3 giai đoạn
chính: **Mounting**, **Updating** và **Unmounting**

6.7.1. Mounting

Mounting được hiểu là việc put các element vào DOM.

6.7.1.1. constructor() method được gọi đầu tiên khi component được khởi tạo, và đây cũng là nơi chúng ta initial các state của component.

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.f
avoritecolor}</h1>
    );
  }
}
```

6.7.1.2. getDerivedStateFromProps() method được gọi ngay trước khi render element vào DOM.

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, sta
te) {
    return {favoritecolor: props.favcol };
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.f
avoritecolor}</h1>
    );
  }
}
```

6.7.1.3. render() là method bắt buộc và nó thực hiện việc đưa các output HTML vào DOM.

```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header
component</h1>
    );
  }
}
```

6.7.1.4. componentDidMount() method được gọi sau khi component đã được render.

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"
})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.f
avoritecolor}</h1>
    );
  }
}
```

6.7.2. Updating

> Một component được update khi có sự thay đổi về `state` hay `props` của nó.

6.7.2.1. getDerivedStateFromProps() method thực hiện công việc set lại state theo prop được initial trước đó.

6.7.2.2. shouldComponentUpdate() method có thể trả về giá trị boolean để chỉ định rằng có thực thi việc rendering hay không (giá trị mặc định: true).

6.7.2.3. render() method thực hiện render lại HTML khi một component đã được update

6.7.2.4. Với getSnapshotBeforeUpdate() method ta có thể lấy được giá trị props và state trước khi update.

6.7.2.5. componentDidUpdate() method được gọi sau khi component đã được updated

6.7.3. Unmounting

6.7.3.1. componentWillUnmount() method được gọi khi một component được remove khỏi DOM.