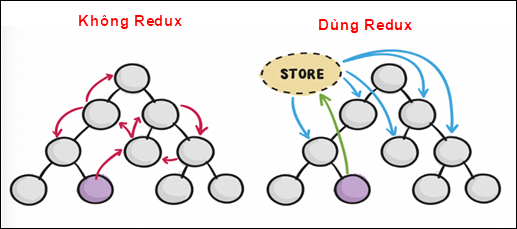
Sử dụng Redux trong React

23/07/2021

Sử dụng Redux trong React là bài viết giới thiệu về thư viện Redux và hướng dẫn cách dùng Redux trong một ứng dụng React.

Với các ứng dụng lớn thì quản lý state trong React sẽ có nhiều phức tạp. Ví dụ khi cần chuyển state giữa các component với nhau, thì phải đưa state lên các component cha để đến component gốc rồi chuyển xuống dần đến đích.



Redux là một thư viện quản lý trạng thái (dữ liệu) cho các ứng dụng Javascript, trong đó có React. Redux tổ chức 1 đối tượng **store** chứa toàn bộ dữ liệu của ứng dụng. Mỗi component có thể truy xuất trực tiếp đến state đê lưu trữ dữ liệu và liên lạc nhau thay vì phải chuyển dữ liệu từ component này đến component khác.

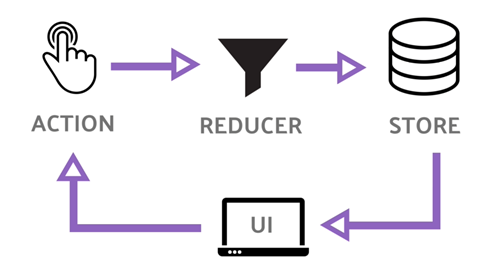
Redux là độc lập với react, cho nên muốn dùng redux  thì bạn phải cần thêm thư viện nữa là**react-redux**để làm cầu nối. Nhờ nó mà bạn mới có thể sử dụng Redux trong React.



Cài đặt 2 thư viện trong project như sau: **npm install redux react-redux**

Trong Redux có ba thành phần cơ bản là **store** ( lưu dữ liệu ứng dụng),  **action** (các thông để thực thực hiện thay đổi) , và **reducer** (thực thi các thay đổi theo thông tin từ action)

Các thành phần cơ bản trong Redux



1. Action

Mỗi action chứa thông tin cho biết cần phải làm gì với **store**, ví dụ như thêm sản phẩm, xóa sản phẩm, thêm loại tin, xóa loại tin… Mỗi action có 1 thuộc tính tên là **type** cho biết hành động cần thực hiện trên store. Ngoài type bạn có thể thêm các thông tin khác tùy ý. Ví dụ:

const actThemLoaiTin = (ten,thutu) => { //action thêm loại tin

return { **type:** THEM\_LOAITIN, ten, thutu, };

};

const actXoaLoaiTin = (id) => { //action xóa loại tin

return {**type:** XOA\_LOAITIN, id, };

};

2. Reducer

Reducer thực thi các t**hay đổi** **state**(trong store) dựa theo các thông tin trong action. Ví dụ: thêm dữ liệu, cập nhật, xóa dữ liệu trong store. Reducer nhận tham số là state hiện tại và action cần thi hành. Các thông tin trong action sẽ giúp reducer thực thi công việc và trả về state mới.  Ví dụ:

const **loaitinReducer** = (state = [], action) => {

switch (**action.type**) {

case THEM\_LOAITIN:

const idLT = new Date().getTime();

state = [...state, { id: idLT, ten: action.ten }];

**return state;**

case SUA\_LOAITIN:

const indexLT = state.findIndex((row) => row.id === action.id);

if (indexLT !== -1) state[indexLT].ten = action.ten;

**return state;**

case XOA\_LOAITIN:

const idLT = action.id;

state = state.filter(loaitin => {

if (loaitin.id === idLT) return false

return true

})

**return state;**

default: **return state;**

}

};

3. Store

**Store** là nơi lưu trạng thái (dữ liệu) của ứng dụng. Có thể truy xuất dữ liệu trong state, update state…Ví dụ: Tạo store cho ứng dụng:

const store = createStore(reducers); //Tạo store

Trong store có hàm dispatch(), là cách dùng để cập nhật state: **store.dispatch().**

Những thông tin cần biết trước khi bắt đầu

Ngoài ba thành phần cơ bản action, reducer, store. Bạn cần biết thêm một số thông tin trước khi bắt đầu.

Provider

Provider là component được cung cấp bởi thư viện react-redux. Cung cấp store cho những component con của nó, các component trong ứng dụng muốn dùng store phải đặt trong provider. Ví dụ:

//index.js

const store = createStore(reducers); //Tạo store

ReactDOM.render(

<React.StrictMode>

<Provider store={store}>

<App />

<Counter/>

</Provider>

</React.StrictMode>,

document.getElementById("root")

);

Connect đến store

Trong mỗi component, bạn dùng hàm **connect**để kết nối đến store. Chỉ những component bên trong provider mới có thể connect được. Hàm connect có 2 tham số  là **mapStateToProps**và **mapDistpachToProps**, hai hàm này định nghĩa cách tương tác giữa component hiện tại với **store**và định nghĩa trong component trước khi gọi connect. Ví dụ:

export default connect(mapStateToProps, mapDispatchToProps)(HienLoaiTin);

Component không bao giờ truy xuất trực tiếp đến store, mà hàm connect sẽ thực hiện giúp với hai tham số như trên.

Hàm mapStateToProps

Tham số đầu tiên của hàm connect là hàm dùng để lấy dữ liệu từ store đưa vào component dùng như props. Tài liệu thường đặt tên là **mapStateToProps** cho rõ nghĩa chứ thật ra bạn đặt tên gì cũng được, ví dụ như mapState. Nhờ hàm này, bạn có thể dùng các giá trị từ state trong component như props bình thường.

Hàm này được gọi mỗi lần có thay đổi state trong store. Nó nhận toàn bộ state của store và trả về đối tượng data mà component cần dùng. Ví dụ:

// chuyển state từ store thành props của component

const **mapState** = (state, ownProps) => {

return { lt: state.loaitin, };

};

export default connect(**mapState**, mapDispatchToProps)(HienLoaiTin);

Khi store có thay đổi, hàm **mapState**sẽ tự chạy và lấy dữ liệu loaitin đã lưu trong store lưu vào biến lt. Khi đó trong component, sử dụng **props.l**t sẽ có dữ liệu mới.

Hàm mapDispatchToProps

Tham số thứ hai của hàm connect là hàm dùng để báo (lúc connect) về các action trong component sẽ gửi đến store. Các action này dùng trong component như các props. Tài liệu thường đặt tên là **mapDispatchToProps** chứ thật ra bạn đặt tên gì cũng được, ví dụ như mapDispatch. Nhờ hàm này, bạn có thể send các action và thông tin lên store để chỉnh state.

// chuyển dispatch thành props

const mapDispatchToProps = (dispatch) => {

return {

suaLoaiTin: (id, ten) => {

dispatch(actSuaLoaiTin(id, ten));

},

xoaLoaiTin: id => {

dispatch(actXoaLoaiTin(id));

}

};

};

export default connect(mapState, mapDispatchToProps)(HienLoaiTin);

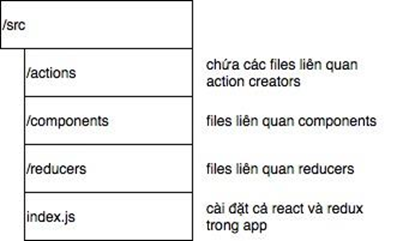
**dispatch** là hàm của **store**. Bạn có thể gọi store.dispatch để gửi 1 action. Đó là cách để làm cho state thay đổi

Tạo ứng dụng với React với redux

Sau đây chúng ta sẽ thực hiện thử một ứng dụng react với việc sử dụng redux. Ứng dụng này để quản lý ghi chú và loại tin.

0. Cấu trúc project với redux

Khi đã dùng đến redux, ứng dụng không phải đơn giản nữa mà ít nhiều có sự phức tạp, từ nghiệp vụ, dữ liệu cho đến các file code trong project. Cho nên cấu trúc folder bạn nên bổ sung thêm để dễ quản lý code. Ví dụ trong src, tạo folder **actions** để chứa các action, tạo folder **components** để chứa các component, tạo folder **reducers** để chứa các reducer…File index.js là nơi bạn thực hiện các khởi tạo ban đầu như nạp redux, tạo store,  khai báo provider…



1. Chuẩn bị:

a**. Tạo project React**

Thực hiện tạo project như thông thường

b. Cài thư viện redux và react-redux

npm install redux react-redux

c. Tạo cấu trúc folder

Trong folder **src**, tạo các folder con sau

src/

...const

...actions

...reducers

...components

Trong đó:

* const: folder chứa các hằng dùng trong project
* actions: chứa các *action*
* reducers: chứa file *reducers* trong redux.

d. Nhúng bootstrap

Chúng ta có dùng bootstrap cho định dạng, việc nhỏ làm trước . Bạn mở file public/index.html và nhúng thư viện bootstrap vào:

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css" rel="stylesheet">

e. Chạy ứng dụng

npm start

Bạn sẽ thấy ứng dụng chạy và bây giờ chúng ta sẽ thực hiện những cái mới: Tạo store, provider, tạo reducer…

2. Import redux, tạo store, reducer, provider

Mở file src/index.js và code

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

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

**import { createStore } from "redux";**

**import reducers from "./reducers/index"; // import reducer thay đổi store**

**const store = createStore(reducers); //Tạo store**

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

**<Provider store={store}>**

<App />

**</Provider>**

);

console.log("Xem store:", store.getState());

3. Tạo các reduder để thay đổi dữ liệu trong store

Để sử dụng Redux trong React, bạn cần phải tạo các reducer để cập nhật các loại dữ liệu mình có trong ứng dụng, nhưng phải có một reducer đóng vai trò root reducer để gom các reducer lại với nhau.

– Trong folder reducers, tạo file index.js để dùng làm rooter reducer , gom 2 reducer là note và loaitin:

// src/reducers/index.js

import {combineReducers} from 'redux'

import noteReducer from './noteReducer';

import loaitinReducer from './loaitinReducer';

export default combineReducers({

note: noteReducer,

loaitin: loaitinReducer,

})

– Tiếp tục tạo file reducers/noteReducer.js để quản lý dữ liệu node trong store

// reducers/noteReducer.js

import { ADD\_NEW\_NOTE, REMOVE\_NOTE, EDIT\_NOTE } from "../const/index";

const noteReducers = (state = [], action) => {

switch (action.type) {

case ADD\_NEW\_NOTE:

const id\_Note = new Date().getTime();

state= [...state, { id: id\_Note, content: action.content }];

console.log("Thêm note:" , state);

return state;

case EDIT\_NOTE:

const indexNote = state.findIndex((row) => row.id === action.id);

if (indexNote !== -1)

state[indexNote].content = action.content;

console.log("Chỉnh note: ", state);

return state;

case REMOVE\_NOTE:

const idNote = action.id;

state = state.filter(row => {

if (row.id === idNote) return false; else return true

})

console.log("Xóa node:", state);

return state;

default:

return state;

}

};

export default noteReducers

– Tạo file reducers/loaitinReducer.js để quản lý dữ liệu loaitin trong store

// reducers/loaitineducer.js

import { THEM\_LOAITIN, XOA\_LOAITIN, SUA\_LOAITIN } from "../const/index";

const loaitinReducer = (state = [], action) => {

switch (action.type) {

case THEM\_LOAITIN:

const id\_LT = new Date().getTime();

state = [...state, { id: id\_LT, ten: action.ten }];

console.log("Thêm LT:" , state);

return state;

case SUA\_LOAITIN:

const indexLT = state.findIndex((row) => row.id === action.id);

if (indexLT !== -1) state[indexLT].ten = action.ten;

console.log("Chỉnh LT: ", state);

return state;

case XOA\_LOAITIN:

const idLT = action.id;

state = state.filter(loaitin => {

if (loaitin.id === idLT) return false

return true

})

console.log("Xóa LT:", state);

return state;

default: return state;

}

};

export default loaitinReducer

– Giiờ thì tạo các hằng để dùng cho thân thiện, hãy tạo file const/index.js và code

// const/index.js

export const ADD\_NEW\_NOTE = "ADD\_NEW\_NOTE";

export const REMOVE\_NOTE = "REMOVE\_NOTE";

export const EDIT\_NOTE = "EDIT\_NOTE";

export const ADD\_LOAITIN = "ADD\_LOAITIN";

export const REMOVE\_LOAITIN = "REMOVE\_LOAITIN";

export const EDIT\_LOAITIN = "EDIT\_LOAITIN";

4. Tạo các action để chứa thông tin cho reducer hoạt động

– Các hằng giúp code bạn thân thiện trong lập trình, không nhất thiết phải tạo. Ở đây chúng ta dùng luôn. Bạn tạo file actions/actNote.js

// actions/actNote.js

import { ADD\_NEW\_NOTE, REMOVE\_NOTE, EDIT\_NOTE } from "../const/index";

//action thêm note

export const actAddNote = (content) => {

return { type: ADD\_NEW\_NOTE, content,};

};

//action xóa note

export const actRemoveNote = (id) => {

return {type: REMOVE\_NOTE, id,};

};

//action sửa note

export const actEditNote = (id, content) => {

return { type: EDIT\_NOTE, id, content,};

};

– Các hằng liên quan đến loạitin, khai báo bằng cách tạo file actions/actLoaiTin.js

// actions/actLoaiTin.js

import { ADD\_LOAITIN, REMOVE\_LOAITIN, EDIT\_LOAITIN } from "../const/index";

//action thêm loại tin

export const actAddLoaiTin = (ten) => {

return { type: ADD\_LOAITIN, ten};

};

//action xóa loại tin

export const actRemoveLoaiTin = (id) => {

return {type: REMOVE\_LOAITIN, id,};

};

//action sửa loại tin

export const actEditLoaiTin = (id, ten) => {

return { type: EDIT\_LOAITIN, id, ten,};

};

5. Tạo các component quản lý ghi chú

a. Tạo component/TaoNote.js

// conponents/TaoNote.js

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

import { connect } from "react-redux";

import { actAddNote } from "../actions/actNote"; //Import các actions

function TaoNote(props) {

const [content, hamGanContent] = useState();

const refNoteContent = useRef();

const hamThemNote = () => {

if (content.trim()=="") {

alert("Bạn chưa nhập nội dung");

return;

}

props.addNote(content); //Props này tạo bởi hàm mapDispatch

refNoteContent.current.value = '';

hamGanContent('');

};

return (

<div className="col-md-12">

<div className="input-group mb-8">

<input type="text" className="form-control"

placeholder="Nội dung ghi chú"

onChange={(e) => { hamGanContent(e.target.value) }}

ref={refNoteContent}

/>

<div className="input-group-append">

<button type="button" className="btn btn-primary"

onClick={hamThemNote}>

Thêm ghi chú

</button>

</div>

</div>

</div>

);

}

const mapDispatch = (dispatch) => {

return {

addNote: (content) => {

dispatch(actAddNote(content));

},

};

};

export default connect(null, mapDispatch)(TaoNote);

b. Tạo component/HienNote.js

// components/HienNote.js

import React, { useState } from "react";

import "./HienNote.css";

import { connect } from "react-redux";

import { actEditNote, actRemoveNote } from "../actions/actNote";

function HienNote(props) {

let noidungGhiChu = props.noteData.content;

let noteID = props.noteData.id

const [noteContent, hamGanNoiDungGhiChu] = useState(noidungGhiChu);

const hamSuaGhiChu = (e) => {

hamGanNoiDungGhiChu(e.target.value)

props.editNote(noteID, e.target.value)

}

const hamXoaGhiChu = () => { props.removeNote(noteID) }

return (

<div className="mt-2 card bg-warning">

<textarea className="form-control" value= {noteContent} onChange = {hamSuaGhiChu}></textarea>

<div className="card-footer p-1">

<button className="btn btn-danger btn-sm float-right"

onClick={hamXoaGhiChu}>Xóa</button>

</div>

</div>

);

}

// chuyển dispatch thành props

const mapDispatchToProps = (dispatch) => {

return {

editNote: (id,content) => {dispatch(actEditNote(id, content)); },

removeNote: id => { dispatch(actRemoveNote(id)); }

};

};

// chuyển state từ store thành props của component

const mapStateToProps = (state, ownProps) => {

return { note: state.note, };

};

export default connect(null, mapDispatchToProps)(HienNote);

File components/HienNote.css

/\* src/components/HienNote.css \*/

textarea {

background-color: rgba(0, 0, 0, 0);

border-width: 0;

overflow: hidden;

resize: none;

}

textarea:focus{ outline: none;}

6. Layout hiện thông tin

File src/App.js, code lại để được như sau:

import { connect } from 'react-redux';

import HienNote from './components/HienNote';

import TaoNote from './components/TaoNote';

function App(props) {

const kq=

<div className="container">

<div className="row">

<div className="col-6 bg-dark">

<TaoNote />

<div className="row">

{props.note.map((n, index) => { // Render các ghi chú.

return <HienNote noteData = {n} key={n.id}/>

})}

</div>

</div>

<div className="col-6 bg-secondary">Loại tin </div>

</div>

</div>

return (kq);

}

const mapStateToProps = (state, ownProps) => {

//console.log("Toàn bộ state " , state);

return {

note: state.note,

loaitin: state.loaitin,

};

};

export default connect(mapStateToProps, null)(App);

7. Test

Giờ thì bạn chạy ứng dụng, thêm, sửa xóa ghi chú, phải hoạt động tốt. Quan sát console.log để phân tích các hoạt động của từng chức năng thêm, sửa, xóa. Phân tích hoạt động củia các hàm mapSateToProps và mapDispatchToProps

8. Tạo các component quản lý loại tin

Bạn tự thực hiện thử xem nhé

Tóm tắt, để sử dụng Redux trong React bạn cần nhớ một số thông tin sau:

1. Cần tạo các reducer (reducer giống như model trong thành phần MVC) để thay đổi dữ liệu trong store
2. Bạn phải tạo các action : là các đối tượng chứa thông tin gửi đến cho reducer dùng.
3. Store thì giống như database vậy, bạn tạo nói trong index.js và bao quanh conponents bởi tag <Provider>
4. Mỗi component phải connect đến **store**thì mới dùng được.
5. Redux thì sẽ tự động gửi thông tin thay đổi trong store về cho component nếu bạn có dùng hàm mapStateToprops, còn hàm mapDispathToProps sẽ giúp gửi thông tin từ component lên reducer để cập nhập vào store. Xem thêm tài liệu ở đây : <https://react-redux.js.org/using-react-redux/connect-mapstate>