???

1. Sự khác nhau giữa khai báo hàm

Function () {

}

Và

Var a = function(){

}

1. Arrow function

()=> {} -> hàm này không truyền tham số, this trong hàm này là this của cha

1. Async/await

Hooks sử dụng cho function componnent

1. useState(initialState)

output là 1 mảng gồm 2 phần tử [state, setState]

Sử dụng: const [status, setStatus] = useState(0)

Chú ý:

useState() sử dụng REPLACING thay vì MERGING Ví dụ: setStatus({…status, color: ‘green’})

initial state chỉ sử dụng cho lần đầu, những lần sau nó bị bỏ rơi

* Sử dụng callback function ()=> {const initColor = getCOmplicatedColor(); return initiColor}
* Đảm bảo initial State chỉ chạy 1 lần duy nhất

1. useEffect

side effects: những thứ tác động và chạy từ bên ngoài component ví dụ: gọi api lấy dữ liệu, tương tác với DOM-> không cần clean up, subscriptions, setTimeout, setInterval -> cần clean up

useEffect: được thực thi sau mỗi lần render

useEffect (callback, dependencies) {}

* ko có dependencies: luôn luôn chạy
* dependencies = [] : chạy 1 lần duy nhất
* dependencies = [value]: chạy lịa hay không tùy thuộc vào việc biến có thay đổi hay không

useEffect(() => {

console.log(‘component did mount’)

return () => {

console.log(‘component will unmount’)

}

})

1. Custom hooks

* 1 hooks do mình tự định nghĩa ra
* Có thể sử dụng các hooks khác như useState, useEffect, .. hoặc 1 custom hook khác
* Output trả ra là dữ liệu
* Lưu Redux state vào local storage với redux-persist

Redux Persist nhận vào Redux state object của bạn và lưu nó vào các bộ lưu trữ nội bộ. Sau đó khi app khởi động thì nó sẽ lấy các state này ra và lưu trở lại vào Redux.

**Bắt đầu sử dụng**

**Dependencies**

npm install --save redux-persist - hoặc - yarn add redux-persist

**Implementation**

Khi tạo Redux store, truyền vào hàm createStore của bạn một hàm persistReducer có tác dụng đóng gói reducer gốc trong app của bạn. Một khi store đã được khởi tạo, truyền nó vào hàm persistStore để đảm bảo Redux state sẽ được lưu vào storage mỗi khi nó thay đổi.

// src/store/index.js

import { createStore } from 'redux';

import { persistStore, persistReducer } from 'redux-persist';

import storage from 'redux-persist/lib/storage';

import autoMergeLevel2 from 'redux-persist/lib/stateReconciler/autoMergeLevel2';

import rootReducer from './reducers'; // giá trị trả về từ combineReducers

const persistConfig = {

key: 'root',

storage: storage,

stateReconciler: autoMergeLevel2 // Xem thêm tại mục "Quá trình merge".

};

const pReducer = persistReducer(persistConfig, rootReducer);

export const store = createStore(pReducer);

export const persistor = persistStore(store);

Nếu bạn đang dùng React, gói root component của bạn trong PersistGate. Nó sẽ delay quá trình render UI app của bạn cho đến khi state đã được lấy ra và lưu trở lại vào Redux.

import React from 'react';

import { Provider } from 'react-redux';

import { PersistGate } from 'redux-persist/lib/integration/react';

import { persistor, store } from './store';

import { RootComponent, LoadingView } from './components';

const App = () => {

return (

<Provider store={store}>

// 2 props loading và persistor đều yêu cầu phải có

<PersistGate loading={<LoadingView />} persistor={persistor}>

<RootComponent />

</PersistGate>

</Provider>

);

};

export default App;

**Tùy chỉnh những gì sẽ được lưu**

Nếu bạn không muốn lưu 1 phần nào đó trong state, bạn có thể cho nó vào blacklist. blacklist sẽ được thêm vào config object mà chúng ta sẽ sử dụng để cài đặt PersistReducer.

const persistConfig = {

key: 'root',

storage: storage,

blacklist: ['navigation']

};

const pReducer = persistReducer(persistConfig, rootReducer);

export const store = createStore(pReducer);

export const persistor = persistStore(store);

blacklist nhận vào một mảng string. Mỗi string cần khớp với 1 phần của state được quản lý bởi reducer mà bạn truyền vào persistReducer. Với ví dụ trên, nếu rootReducer được tạo thông qua hàm combineReducers thì chúng ta sẽ expect là navigation là 1 trong các reducer, như thế này:

combineReducers({

auth: AuthReducer,

navigation: NavReducer,

notes: NotesReducer

});

Ngược lại, nếu bạn chỉ muốn lưu 1 phần nào đó trong state thì chúng ta có thể dùng whitelist. Ví dụ với các reducer trên chúng ta chỉ muốn lưu auth thôi chẳng hạn:

const persistConfig = {

key: 'root',

storage: storage,

whitelist: ['auth']

};

Vậy nếu chúng ta muốn blacklist một nested property thì sao? Ví dụ, giả định rằng state object của bsnj có một key auth và bạn muốn lưu auth.currentUser nhưng không lưu auth.isLoggingIn. Để làm được điều này, hãy gói AuthReducer vào một PersistReducer, và sau đó cho isLoggingIn vào blacklist.

// AuthReducer.js

import storage from 'redux-persist/lib/storage';

import { persistReducer } from 'redux-persist';

const INITIAL\_STATE = {

currentUser: null,

isLoggingIn: false

};

const AuthReducer = (state = INITIAL\_STATE, action) => {

// reducer implementation

};

const persistConfig = {

key: 'auth',

storage: storage,

blacklist: ['isLoggingIn']

};

export default persistReducer(persistConfig, AuthReducer);

Nếu bạn muốn tất cả các config được để ở cùng một chỗ, thì thay vì để nó trong reducer tương ứng, bạn có thể đưa hết chúng vào hàm combineReducers:

// src/reducers/index.js

import { combineReducers } from 'redux';

import storage from 'redux-persist/lib/storage';

import { persistReducer } from 'redux-persist';

import { authReducer, navReducer, notesReducer } from './reducers'

const rootPersistConfig = {

key: 'root',

storage: storage,

blacklist: ['navigation']

};

const authPersistConfig = {

key: 'auth',

storage: storage,

blacklist: ['isLoggingIn']

};

const rootReducer = combineReducers({

auth: persistReducer(authPersistConfig, authReducer),

navigation: navReducer,

notes: notesReducer

});

export default persistReducer(rootPersistConfig, rootReducer);

**Quá trình merge**

Khi app của bạn khởi động, Redux sẽ set một state mặc định. Ngay sau đó, Redux Persist sẽ lấy state đã lưu từ storage. State này sau đó sẽ được ghi đè lên trên các state mặc định.

Quá trình merge được cho là sẽ hoạt động một cách tự động cho bạn. Tuy vậy, bạn cũng có thể có những xử lý riêng cho quá trình này. Ví dụ, ở những bản cũ hơn của Redux Persist thì chúng ta cần tự quản lý quy trình lấy lại state (rehydration) bằng cách bắt action REHYDRATE trong reducer và sau đó lưu payload của action này vào Redux state.

import { REHYDRATE } from 'redux-persist';

const INITIAL\_STATE = {

currentUser: null,

isLoggingIn: false

};

const AuthReducer = (state = INITIAL\_STATE, action) => {

switch (action.type) {

case REHYDRATE:

return {

...state,

currentUser: action.payload.currentUser

};

// ...handle other cases

Action REHYDRATE sẽ được dispatch bởi Redux Persist ngay sau khi nó đã lấy được state từ storage. Nếu bạn trả về một state object mới từ action REHYDRATE, nó sẽ là state cuối cùng bạn nhận được. Nhưng như đã nói ở trên thì ở phiên bản mới của Redux Persist thì bạn không cần phải bắt action bằng tay nữa, trừ khi bạn muốn kiểm soát việc state được khôi phục như thế nào.

**Tuy vậy, hãy cẩn thận với điều này...**

Có một vấn đề mà bạn cần nắm được trong quá trình merge, nó liên quan đến việc quá trình merge state sẽ sâu đến mức nào. Ở trên chúng ta đã biết quá trình merge sẽ ghi đè state mặc định bằng state được lấy ra từ storage. Sau đây là cách nó hoạt động:

Giả sử state mặc định của chúng ta nhìn như thế này, và chúng ta đang lưu toàn bộ các property:

// initial state

{

auth: {

currentUser: null,

isLoggingIn: false

},

notes: []

}

App của chúng ta khởi động, và đây là state đã được lưu:

// persisted state

{

auth: {

currentUser: { firstName: 'Mark', lastName: 'Newton' },

isLoggingIn: false

},

notes: [noteA, noteB, noteC]

}

Mặc định thì quá trình merge sẽ đơn giản chỉ thay các state ở cấp cao nhất thôi. Trong code thì nó nhìn sẽ tương tự thế này:

const finalState = { ...initialState };

finalState['auth'] = persistedState['auth']

finalState['notes'] = persistedState['notes']

Trong hầu hết các trường hợp thì như thế là đủ, nhưng nếu bạn release version mới của app của bạn và set state mặc định của auth như thế này thì sao:

const INITIAL\_STATE = {

currentUser: null,

isLoggingIn: false,

error: ''

};

Bạn chắc chắn sẽ muốn state object cuối cùng có chứa key error này. Nhưng state object được lưu lại không có key đó, và nó sẽ thay đổi hoàn toàn state mặc định trong quá trình merge. Tạm biệt key error.

Cách sửa cho trường hợp này là yêu cầu PersistReducer merge sâu 2 cấp. Trong mục Bắt đầu sử dụng, bạn có thể đã thấy setting stateReconciler trong root PersistReducer.

import autoMergeLevel2 from 'redux-persist/lib/stateReconciler/autoMergeLevel2';

const persistConfig = {

key: 'root',

storage: storage,

\*\* stateReconciler: autoMergeLevel2\*\*

};

autoMergeLevel2 là config để merge sâu 2 cấp. Đối với state auth, điều đó có nghĩa là quá trình merge đầu tiên sẽ tạo một bản copy state mặc định của auth, và sau đó chỉ ghi đè những key trong object auth nếu nó đã được lưu. Bởi vì key error chưa được lưu, nó sẽ được giữ nguyên.

Tóm lại, bạn cần phải nhớ rằng PersistReducers mặc định sẽ sử dụng autoMergeLevel1, nghĩa là nó sẽ thay các state ở cấp cao nhất bằng những gì đã được lưu. Nếu bạn không có những PersistReducer riêng để quản lý việc lưu state nào cho những key ở cấp cao này thì bạn chắc chắn sẽ cần phải sử dụng autoMergeLevel2. Tôi thì thích tự set cấp merge và ko dùng hàm này, nhưng tùy vào bạn thôi.

**Tùy biến nâng cao**

**Transforms**

Transform cho phép bạn tùy biến state object mà bạn đã lưu và phục hồi.

Khi state object được lưu, nó đầu tiên sẽ được serialize bằng JSON.stringify(). Nếu có những phần trong state object không thể map với JSON object, quá trình serialize có thể sẽ biến đổi những phần này theo những cách mà chúng ta không lường trước được. Ví dụ, kiểu Set trong Javascript không tồn tại trong JSON. Khi bạn serialize một Set bằng JSON.stringify(), nó sẽ trở thành một object rỗng. Chắc chắn đây không phải là điều bạn muốn.

Dưới đây là một Transform có thể lưu một property thuộc kiểu Set đơn giản bằng cách convert nó thành một mảng và ngược lại. Với cách này, Set sẽ được convert thành Array, là một cấu trúc dữ liệu được support trong JSON. Khi được lấy ra từ local storage, mảng lại được convert ngược lại thành Set trước khi lưu vào Redux store.

import { createTransform } from 'redux-persist';

const SetTransform = createTransform(

// transform state trước khi nó được serialize và lưu

(inboundState, key) => {

// convert mySet thành một mảng.

return { ...inboundState, mySet: [...inboundState.mySet] };

},

// transform state đang được phục hồi

(outboundState, key) => {

// convert mySet trở lại thành Set.

return { ...outboundState, mySet: new Set(outboundState.mySet) };

},

// định nghĩa reducer nào sẽ áp dụng transform này.

{ whitelist: ['someReducer'] }

);

export default SetTransform;

Hàm createTransform nhận vào 3 tham số:

* Một hàm được gọi ngay trước khi lưu state.
* Một hàm được gọi ngay trước khi phục hồi state.
* Một config object

Sau đó, transform cần được thêm vào config object của PersistReducer:

import storage from 'redux-persist/lib/storage';

import { SetTransform } from './Transforms';

const persistConfig = {

key: 'root',

storage: storage,

transforms: [SetTransform]

};

// ...remaining implementation

Bài viết được dịch từ [The Definitive Guide to Redux Persist](https://blog.reactnativecoach.com/the-definitive-guide-to-redux-persist-84738167975) của tác giả **Mark Newton**.