D:\应用程序案例\state\state.module.ts

@NgModule({})

export class StateModule {

static forRoot(): ModuleWithProviders<StateModule> {

return {

ngModule: StateModule,

providers: [

...stateMetaReducers,

provideDefaultConfig(defaultStateConfig),

{ provide: StateConfig, useExisting: Config },

],

};

}

}

D:\应用程序案例\\state\reducers\index.ts

import { Optional, PLATFORM\_ID, Provider, InjectionToken } from '@angular/core';

import { TransferState } from '@angular/platform-browser';

import { META\_REDUCERS } from '@ngrx/store';

import { Config } from '../../config/config.module';

import { WindowRef } from '../../window/window-ref';

import { getStorageSyncReducer } from './storage-sync.reducer';

import { getTransferStateReducer } from './transfer-state.reducer';

export { getStateSlice } from '../utils/get-state-slice';

export { getStorageSyncReducer } from './storage-sync.reducer';

export \* from './transfer-state.reducer';

export const TRANSFER\_STATE\_META\_REDUCER = new InjectionToken(

'TransferStateMetaReducer'

);

export const STORAGE\_SYNC\_META\_REDUCER = new InjectionToken(

'StorageSyncMetaReducer'

);

export const stateMetaReducers: Provider[] = [

{

provide: TRANSFER\_STATE\_META\_REDUCER,

useFactory: getTransferStateReducer,

deps: [

PLATFORM\_ID,

[new Optional(), TransferState],

[new Optional(), Config],

],

},

{

provide: STORAGE\_SYNC\_META\_REDUCER,

useFactory: getStorageSyncReducer,

deps: [WindowRef, [new Optional(), Config]],

},

{

provide: META\_REDUCERS,

useExisting: TRANSFER\_STATE\_META\_REDUCER,

multi: true,

},

{

provide: META\_REDUCERS,

useExisting: STORAGE\_SYNC\_META\_REDUCER,

multi: true,

},

];

D:\应用程序案例\state\reducers\transfer-state.reducer.ts

export const CX\_KEY: StateKey<string> = makeStateKey<string>('cx-state');

export function getTransferStateReducer(

platformId,

transferState?: TransferState,

config?: StateConfig

) {

if (

transferState &&

config &&

config.state &&

config.state.ssrTransfer &&

config.state.ssrTransfer.keys

) {

if (isPlatformBrowser(platformId)) {

return getBrowserTransferStateReducer(

transferState,

config.state.ssrTransfer.keys

);

} else if (isPlatformServer(platformId)) {

return getServerTransferStateReducer(

transferState,

config.state.ssrTransfer.keys

);

}

}

return (reducer) => reducer;

}

export function getServerTransferStateReducer(

transferState: TransferState,

keys: { [key: string]: StateTransferType }

) {

const transferStateKeys = filterKeysByType(

keys,

StateTransferType.TRANSFER\_STATE

);

return function (reducer) {

return function (state, action: any) {

const newState = reducer(state, action);

if (newState) {

const stateSlice = getStateSlice(transferStateKeys, [], newState);

transferState.set(CX\_KEY, stateSlice);

}

return newState;

};

};

}

export function getBrowserTransferStateReducer(

transferState: TransferState,

keys: { [key: string]: StateTransferType }

) {

const transferStateKeys = filterKeysByType(

keys,

StateTransferType.TRANSFER\_STATE

);

return function (reducer) {

return function (state, action: any) {

if (action.type === INIT) {

if (!state) {

state = reducer(state, action);

}

// we should not utilize transfer state if user is logged in

const authState = (state as StateWithAuth)[AUTH\_FEATURE];

const isLoggedIn =

authState && authState.userToken && authState.userToken.token;

if (!isLoggedIn && transferState.hasKey(CX\_KEY)) {

const cxKey = transferState.get(CX\_KEY, {});

const transferredStateSlice = getStateSlice(

transferStateKeys,

[],

cxKey

);

state = deepMerge({}, state, transferredStateSlice);

}

return state;

}

return reducer(state, action);

};

};

}

D:\应用程序案例\\state\reducers\storage-sync.reducer.ts

export function getStorageSyncReducer<T>(

winRef: WindowRef,

config?: StateConfig

): MetaReducer<T, Action> {

if (

!winRef.nativeWindow ||

!config ||

!config.state ||

!config.state.storageSync ||

!config.state.storageSync.keys

) {

return (reducer) => reducer;

}

const storageSyncConfig = config.state.storageSync;

return (reducer: ActionReducer<T, Action>): ActionReducer<T, Action> => {

return (state, action): T => {

const newState = reducer(state, action);

if (action.type === INIT || action.type === UPDATE) {

const rehydratedState = rehydrate(config, winRef);

return deepMerge({}, newState, rehydratedState);

}

if (action.type !== INIT) {

// handle local storage

const localStorageKeys = filterKeysByType(

storageSyncConfig.keys,

StorageSyncType.LOCAL\_STORAGE

);

const localStorageExclusionKeys = filterKeysByType(

storageSyncConfig.excludeKeys,

StorageSyncType.LOCAL\_STORAGE

);

const localStorageStateSlices = getStateSlice(

localStorageKeys,

localStorageExclusionKeys,

newState

);

persistToStorage(

config.state.storageSync.localStorageKeyName,

localStorageStateSlices,

winRef.localStorage

);

// handle session storage

const sessionStorageKeys = filterKeysByType(

storageSyncConfig.keys,

StorageSyncType.SESSION\_STORAGE

);

const sessionStorageExclusionKeys = filterKeysByType(

storageSyncConfig.excludeKeys,

StorageSyncType.SESSION\_STORAGE

);

const sessionStorageStateSlices = getStateSlice(

sessionStorageKeys,

sessionStorageExclusionKeys,

newState

);

persistToStorage(

config.state.storageSync.sessionStorageKeyName,

sessionStorageStateSlices,

winRef.sessionStorage

);

}

return newState;

};

};

}

export function rehydrate<T>(config: StateConfig, winRef: WindowRef): T {

const localStorageValue = readFromStorage(

winRef.localStorage,

config.state.storageSync.localStorageKeyName

);

const sessionStorageValue = readFromStorage(

winRef.sessionStorage,

config.state.storageSync.sessionStorageKeyName

);

return deepMerge(localStorageValue, sessionStorageValue);

}

export function exists(value: Object): boolean {

if (value != null) {

if (typeof value === 'object') {

return Object.keys(value).length !== 0;

}

return value !== '';

}

return false;

}

export function getStorage(

storageType: StorageSyncType,

winRef: WindowRef

): Storage {

let storage: Storage;

switch (storageType) {

case StorageSyncType.LOCAL\_STORAGE: {

storage = winRef.localStorage;

break;

}

case StorageSyncType.SESSION\_STORAGE: {

storage = winRef.sessionStorage;

break;

}

case StorageSyncType.NO\_STORAGE: {

storage = undefined;

break;

}

default: {

storage = winRef.sessionStorage;

}

}

return storage;

}

export function persistToStorage(

configKey: string,

value: any,

storage: Storage

): void {

if (!isSsr(storage) && value) {

storage.setItem(configKey, JSON.stringify(value));

}

}

export function readFromStorage(storage: Storage, key: string): any {

if (isSsr(storage)) {

return;

}

const storageValue = storage.getItem(key);

if (!storageValue) {

return;

}

return JSON.parse(storageValue);

}

export function isSsr(storage: Storage): boolean {

return !Boolean(storage);

}