D:\应用程序案例\loader\loader.reducer.ts

import { Action } from '@ngrx/store';

import { LoaderState } from './loader-state';

import { LoaderAction } from './loader.action';

export const initialLoaderState: LoaderState<any> = {

loading: false,

error: false,

success: false,

value: undefined,

};

\*将通用加载标志(loading)添加到状态块的高阶reducer

\* 利用“loader”元字段的动作来设置特定的标志动作(加载、成功、失败、复位)

\* action (LOAD, SUCCESS, FAIL, RESET)

\*/

export function loaderReducer<T>(

entityType: string,

reducer?: (state: T, action: Action) => T

): (state: LoaderState<T>, action: LoaderAction) => LoaderState<T> {

return (

state: LoaderState<T> = initialLoaderState,

action: LoaderAction

): LoaderState<T> => {

if (

action.meta &&

action.meta.loader &&

action.meta.entityType === entityType

) {

const entity = action.meta.loader;

if (entity.load) {

return {

...state,

loading: true,

value: reducer ? reducer(state.value, action) : state.value,

};

} else if (entity.error) {

return {

...state,

loading: false,

error: true,

success: false,

value: reducer ? reducer(state.value, action) : undefined,

};

} else if (entity.success) {

return {

...state,

value: reducer ? reducer(state.value, action) : action.payload,

loading: false,

error: false,

success: true,

};

} else {

// reset state action

return {

...initialLoaderState,

value: reducer

? reducer(initialLoaderState.value, action)

: initialLoaderState.value,

};

}

}

if (reducer) {

const newValue = reducer(state.value, action);

if (newValue !== state.value) {

return { ...state, value: newValue };

}

}

return state;

};

}

D:\应用程序案例\entity\entity.reducer.ts

export const initialEntityState: EntityState<any> = { entities: {} };

/\*\*

\* Higher order reducer for reusing reducer logic for multiple entities

\*用于重用多个实体的减速机逻辑的高阶减速机

\* Utilizes entityId meta field to target entity by id in actions

利用entityId元字段在操作中通过id来目标实体

\*/

export function entityReducer<T>(

entityType: string,

reducer: (state: T, action: Action) => T

) {

return (

state: EntityState<T> = initialEntityState,

action: EntityAction

): EntityState<T> => {

let ids: string[];

let partitionPayload = false;

if (

action.meta &&

action.meta.entityType === entityType &&

action.meta.entityId !== undefined

) {

ids = [].concat(action.meta.entityId);

// remove selected entities

if (action.meta.entityRemove) {

if (action.meta.entityId === null) {

return initialEntityState;

} else {

let removed = false;

const newEntities = Object.keys(state.entities).reduce((acc, cur) => {

if (ids.includes(cur)) {

removed = true;

} else {

acc[cur] = state.entities[cur];

}

return acc;

}, {});

return removed ? { entities: newEntities } : state;

}

}

partitionPayload =

Array.isArray(action.meta.entityId) && Array.isArray(action.payload);

} else {

ids = Object.keys(state.entities);

}

const entityUpdates: { [id: string]: T } = {};

for (let i = 0; i < ids.length; i++) {

const id = ids[i];

const subAction = partitionPayload

? { ...action, payload: action.payload[i] }

: action;

const newState = reducer(state.entities[id], subAction);

if (newState) {

entityUpdates[id] = newState;

}

}

if (Object.keys(entityUpdates).length > 0) {

return {

...state,

entities: { ...state.entities, ...entityUpdates },

};

}

return state;

};

}

D:\应用程序案例\entity-loader\entity-loader.reducer.ts

/\*\*

\* Higher order reducer that wraps LoaderReducer and EntityReducer enhancing

\* single state reducer to support multiple entities with generic loading flags

\*/

export function entityLoaderReducer<T>(

entityType: string,

reducer?: (state: T, action: LoaderAction) => T

): (

state: EntityLoaderState<T>,

action: EntityLoaderAction

) => EntityLoaderState<T> {

return entityReducer(entityType, loaderReducer(entityType, reducer));

}

D:\应用程序案例\processes-loader\processes-loader.reducer.ts

export const initialProcessesState: ProcessesLoaderState<any> = {

processesCount: 0,

};

/\*\*

\* Higher order reducer that adds processes count

\*/

export function processesLoaderReducer<T>(

entityType: string,

reducer?: (state: T, action: Action) => T

): (

state: ProcessesLoaderState<T>,

action: ProcessesLoaderAction

) => ProcessesLoaderState<T> {

return (

state: ProcessesLoaderState<T> = {

...initialProcessesState,

...initialLoaderState,

},

action: ProcessesLoaderAction

): ProcessesLoaderState<T> => {

const loaderState = loaderReducer(entityType, reducer)(state, action);

if (action.meta && action.meta.entityType === entityType) {

const processesCountDiff = action.meta.processesCountDiff;

if (isDevMode() && state.processesCount + processesCountDiff < 0) {

console.error(

`Action '${action.type}' sets processesCount to value < 0!\n` +

'Make sure to keep processesCount in sync.\n' +

'There should always be only one decrement action for each increment action.\n' +

"Make sure that you don't reset state in between those actions.\n",

action

);

}

if (processesCountDiff) {

return {

...loaderState,

processesCount: state.processesCount

? state.processesCount + processesCountDiff

: processesCountDiff,

};

} else if (processesCountDiff === null) {

// reset action

return {

...loaderState,

...initialProcessesState,

};

}

}

return loaderState;

};

}

D:\应用程序案例\entity-processes-loader\entity-processes-loader.reducer.ts

/\*\*

\* Higher order reducer that wraps ProcessesLoaderReducer and EntityReducer enhancing

\* single state reducer to support multiple entities with generic processesCount flag

\*/

export function entityProcessesLoaderReducer<T>(

entityType: string,

reducer?: (state: T, action: ProcessesLoaderAction) => T

): (

state: EntityProcessesLoaderState<T>,

action: EntityProcessesLoaderAction

) => EntityProcessesLoaderState<T> {

return entityReducer(entityType, processesLoaderReducer(entityType, reducer));

}

D:\应用程序案例\scoped-loader\scoped-loader.reducer.ts

/\*\*

\* Higher order reducer designed to add scope support for loader reducer

\*

\* For backward compatibility, we accommodate scopes alongside current

\* loading/error/success/value flags, thus those names can't be used as scope

\* names.

\*

\* TODO: Improve, issue #5445

\*

\* @param entityType

\* @param reducer

\*/

export function scopedLoaderReducer<T>(

entityType: string,

reducer?: (state: T, action: Action) => T

): (

state: ScopedLoaderState<T> | LoaderState<T>,

action: EntityScopedLoaderAction

) => ScopedLoaderState<T> | LoaderState<T> {

const loader = loaderReducer<T>(entityType, reducer);

return (

state: ScopedLoaderState<T> | LoaderState<T> = initialLoaderState,

action: EntityScopedLoaderAction

): ScopedLoaderState<T> | LoaderState<T> => {

if (

action &&

action.meta &&

action.meta.entityType === entityType &&

action.meta.scope

) {

return {

...state,

[action.meta.scope]: loader(state[action.meta.scope], action),

};

} else {

return loader(state, action);

}

};

}

D:\应用程序案例\scoped-loader\entity-scoped-loader.reducer.ts

/\*\*

\* 包装了scopedloaderreduce和entityreduce的更高阶的减速器，增强了单个状态的减速器，以支持具有通用加载标志和范围的多个实体

\*/

export function entityScopedLoaderReducer<T>(

entityType: string,

reducer?: (state: T, action: LoaderAction) => T

): (

state: EntityLoaderState<T>,

action: EntityLoaderAction

) => EntityLoaderState<T> {

return entityReducer<ScopedLoaderState<T> | LoaderState<T>>(

entityType,

scopedLoaderReducer<T>(entityType, reducer)

);

}