**D:\应用程序案例\core\src\process\process.module.ts**

@NgModule({

imports: [ProcessStoreModule],

})

export class ProcessModule {

static forRoot(): ModuleWithProviders<ProcessModule> {

return {

ngModule: ProcessModule,

providers: [],

};

}

}

**D:\src\process\store\process-store.module.ts**

mport { StateModule } from 'zyapp/core/state/state.module';

import { PROCESS\_FEATURE } from './process-state';

import { reducerProvider, reducerToken } from './reducers/index';

@NgModule({

imports: [StateModule, StoreModule.forFeature(PROCESS\_FEATURE, reducerToken)],

providers: [reducerProvider],

})

export class ProcessStoreModule {}

**D:\process\store\process-state.ts**

import { EntityLoaderState } from '../../state/utils/entity-loader/entity-loader-state';

export const PROCESS\_FEATURE = 'process';

export interface StateWithProcess<T> {

[PROCESS\_FEATURE]: EntityLoaderState<T>;

}

export type EntityLoaderState<T> = EntityState<LoaderState<T>>;

export interface EntityState<T> {

entities: {

[id: string]: T;

};

}

export interface LoaderState<T> {

loading?: boolean;

error?: boolean;

success?: boolean;

value?: T;

}

**D:src\process\store\reducers\index.ts**

export function getReducers<T>(): ActionReducer<EntityLoaderState<T>> {

return entityLoaderReducer(PROCESS\_FEATURE);

}

export const reducerToken: InjectionToken<ActionReducerMap<

EntityLoaderState<any>

>> = new InjectionToken<ActionReducerMap<EntityLoaderState<any>>>(

'ProcessReducers'

);

export const reducerProvider: Provider = {

provide: reducerToken,

useFactory: getReducers,

};

1. **entityLoaderReducer**

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));

}

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

1. **entityReducer**

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:\selectors\index.ts**

import \* as ProcessSelectors from './process.selectors';

export { ProcessSelectors };

D:\应用程序案例\process\store\selectors\process-group.selectors.ts

export \* from './feature.selector';

export \* from './process.selectors';

**D:store\selectors\feature.selector.ts**

import { MemoizedSelector, createFeatureSelector } from '@ngrx/store';

import { StateWithProcess, PROCESS\_FEATURE } from '../process-state';

import { EntityLoaderState } from '../../../state/utils/entity-loader/entity-loader-state';

export function getProcessState<T>(): MemoizedSelector<

StateWithProcess<T>,

EntityLoaderState<T>

> {

return createFeatureSelector<EntityLoaderState<T>>(PROCESS\_FEATURE);

}

D:\应用程序案例\process\store\selectors\process.selectors.ts

import { createSelector, MemoizedSelector } from '@ngrx/store';

import { StateEntityLoaderSelectors } from '../../../state/utils/index';

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

import {

loaderErrorSelector,

loaderLoadingSelector,

loaderSuccessSelector,

} from '../../../state/utils/loader/loader.selectors';

import { StateWithProcess } from '../process-state';

import { getProcessState } from './feature.selector';

export function getProcessStateFactory<T>(

processId: string

): MemoizedSelector<StateWithProcess<T>, LoaderState<T>> {

return createSelector(getProcessState(), (entityState) =>

StateEntityLoaderSelectors.entityStateSelector(entityState, processId)

);

}

export function getProcessLoadingFactory<T>(

processId: string

): MemoizedSelector<StateWithProcess<T>, boolean> {

return createSelector(getProcessStateFactory(processId), (loaderState) =>

loaderLoadingSelector(loaderState)

);

}

export function getProcessSuccessFactory<T>(

processId: string

): MemoizedSelector<StateWithProcess<T>, boolean> {

return createSelector(getProcessStateFactory(processId), (loaderState) =>

loaderSuccessSelector(loaderState)

);

}

export function getProcessErrorFactory<T>(

processId: string

): MemoizedSelector<StateWithProcess<T>, boolean> {

return createSelector(getProcessStateFactory(processId), (loaderState) =>

loaderErrorSelector(loaderState)

);

}