**books.module.ts**

import \* as fromBooks from '@example-app/books/reducers';

StoreModule.forFeature(fromBooks.booksFeatureKey, fromBooks.***reducers***),

**books\reducers\index.ts**

export const ***booksFeatureKey*** = 'books';

export interface ***BooksState*** {

[fromSearch.searchFeatureKey]: fromSearch.State;

[fromBooks.booksFeatureKey]: fromBooks.State;

[fromCollection.collectionFeatureKey]: fromCollection.State;

}

export interface State extends fromRoot.State {

[***booksFeatureKey***]: BooksState;

}

/\*\* Provide reducer in AoT-compilation happy way \*/

export function ***reducers***(state: ***BooksState*** | undefined, action: Action) {

return combineReducers({

[fromSearch.searchFeatureKey]: fromSearch.reducer,

[fromBooks.booksFeatureKey]: fromBooks.reducer,

[fromCollection.collectionFeatureKey]: fromCollection.reducer,

})(state, action);

}

/\*\*

\*选择器函数是一个映射函数工厂。

我们向它传递参数，它返回一个从较大的状态树映射到较小的状态块的函数。

这个选择器只是选择“books”状态。

选择器与' select '操作符一起使用。

\*

\* ```ts

\* class MyComponent {

\* constructor(state$: Observable<State>) {

\* this.booksState$ = state$.pipe(select(getBooksState));

\* }

\* }

\* ```

\*/

/\*\*

createFeatureSelector函数从状态对象的根中选择一段状态。

这用于选择主动加载或惰性加载的特性状态。

\*/

export const selectBooksState = createFeatureSelector<State, BooksState>(

booksFeatureKey

);

/\*\*

\* 每个reducer模块导出selector函数，但是reducers不知道整个状态树。

为了使它们可用，我们需要创建新的选择器来包装它们。

\*

createSelector函数创建了非常高效的选择器，这些选择器是记忆的，只有当参数改变时才会重新计算。

\* 还可以将创建的选择器组合在一起，以选择不同的状态块。

\*/

export const selectBookEntitiesState = createSelector(

selectBooksState,

(state) => state.books

);

export const selectSelectedBookId = createSelector(

selectBookEntitiesState,

fromBooks.selectId

);