定义实体

一个EntityMetadataMap告诉NgRx关于实体的数据。向每个实体名称的集合中添加一个属性。

import { EntityMetadataMap } from '@ngrx/data';

const entityMetadata: EntityMetadataMap = {

Hero: {},

Villain: {}

};

// 因为"hero"的复数不是"heros"

const pluralNames = { Hero: 'Heroes' };

export const entityConfig = {

entityMetadata,

pluralNames

};

导出在AppModule中注册时要使用的实体配置。

注册实体存储

import { NgModule } from '@angular/core';

import { HttpClientModule } from '@angular/common/http';

import { EffectsModule } from '@ngrx/effects';

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

import { DefaultDataServiceConfig, EntityDataModule } from '@ngrx/data';

import { entityConfig } from './entity-metadata';

@NgModule({

imports: [

HttpClientModule,

StoreModule.forRoot({}),

EffectsModule.forRoot([]),

EntityDataModule.forRoot(entityConfig)

]

})

export class AppModule {}

创建实体数据服务

NgRx数据通过在你的服务类中扩展EntityCollectionServiceBase来处理服务器上的创建、检索、更新和删除数据。

import { Injectable } from '@angular/core';

import {

EntityCollectionServiceBase,

EntityCollectionServiceElementsFactory

} from '@ngrx/data';

import { Hero } from '../core';

@Injectable({ providedIn: 'root' })

export class HeroService extends EntityCollectionServiceBase<Hero> {

constructor(serviceElementsFactory: EntityCollectionServiceElementsFactory) {

super('Hero', serviceElementsFactory);

}

}

在组件中使用NgRx数据

要访问实体数据，组件应该注入实体数据服务。

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

import { Hero } from '../../core';

import { HeroService } from '../hero.service';

@Component({

selector: 'app-heroes',

templateUrl: './heroes.component.html',

styleUrls: ['./heroes.component.scss']

})

export class HeroesComponent implements OnInit {

loading$: Observable<boolean>;

heroes$: Observable<Hero[]>;

constructor(private heroService: HeroService) {

this.heroes$ = heroService.entities$;

this.loading$ = heroService.loading$;

}

ngOnInit() {

this.getHeroes();

}

add(hero: Hero) {

this.heroService.add(hero);

}

delete(hero: Hero) {

this.heroService.delete(hero.id);

}

getHeroes() {

this.heroService.getAll();

}

update(hero: Hero) {

this.heroService.update(hero);

}

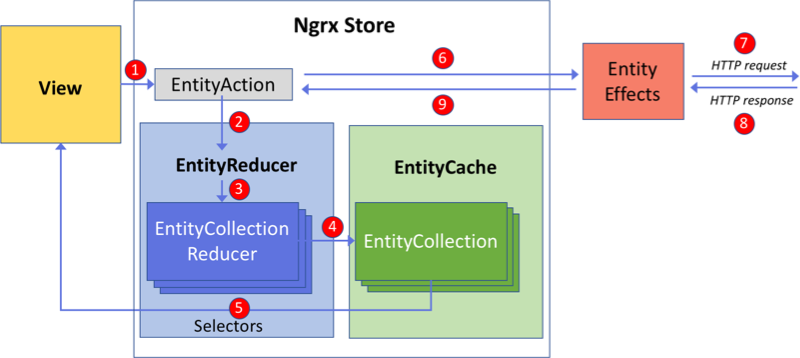
}

你用几行实体元数据向NgRx数据描述你的实体模型，然后让库来完成剩下的工作。

你的组件会注入一个NgRx Data EntityCollectionService，并调用一个或多个标准的命令方法来分派动作。

你的组件还订阅了一个或多个服务的可观察选择器，以便响应性地处理和显示由这些命令产生的实体状态变化。

NgRx数据实际上只是NgRx的外壳。数据以典型的NgRx方式流动。下面的图表演示了持久化实体操作的历程，比如Hero实体类型的QUERY\_ALL。



1 - 视图/组件调用EntityCollectionService.getAll()，它会把英雄的QUERY\_ALL实体动作分派到存储中。

2 - NgRx启动…

（1）NgRx Data EntityReducer读取动作的entityName属性(本例中是Hero)，并将动作和现有实体的集合状态转发给heroes的EntityCollectionReducer。

（2）collection reducer根据操作的entityOp (operation)属性选择一个开关箱。这个例子将动作和集合状态处理成一个新的(更新后的)英雄集合。

（3）store使用更新后的集合更新状态树中的实体缓存。

（4）NgRx可观察对象选择器会检测并向视图中的订阅者报告这些变化(如果有的话)。

3 - 原始的EntityAction然后转到entityeeffects。

4 - 效果为该实体类型选择一个EntityDataService。数据服务向服务器发送一个HTTP请求。

5 - 这个效果会把HTTP响应变成一个新的带有英雄的成功动作(如果请求失败，则会变成一个错误动作)。

6 - NgRx特效把这个动作分派给商店，这就重复了第2步，用英雄来更新集合，刷新视图。