* 为每个实体类型自动的创建actions, reducers, effects, dispatchers, and selectors。
* 为每种实体类型提供默认的HTTP GET、PUT、POST和DELETE方法
* holds entity data as collections within a cache which is a slice of NgRx store state.
* 支持乐观和悲观的保存策略
* enables transactional save of multiple entities of multiple types in the same request.
* makes reasonable default implementation choices
* offers numerous extension points for changing or augmenting those default behaviors.

NgRx数据的目标是管理持久的实体数据，如客户和订单，许多应用程序查询并保存到远程存储。这是它的最佳位置。

它不适用于非实体数据。使用标准的NgRx可以更好地管理值类型、枚举、会话数据和高度特殊的数据。现实世界中的应用程序将受益于NgRx技术的组合，它们共享一个公共的存储。

#### Entity

实体是具有长期数据值的对象，您可以从数据库中读取数据值并将其写入数据库。实体指的是应用程序域中的某些“东西”。示例包括Customer、Order、LineItem、Product、Person和User。

在Ngrx数据中，我们通过主键来维护实体对象的标识。NgRx数据中的每个实体都必须有一个主键。主键通常是对象的单个属性。例如，“Sally”实体对象可能是“Customer”实体类型的实例，该实例永久不变的主键是id属性，值为42。

主键不必是单个属性。如果需要该特性，它可以由对象的多个属性组成。重要的是，主键在相同类型的实体的永久集合中惟一地标识该对象。只能有一个id为42的客户实体，该实体为“Sally”。

#### Entity Collection

实体集合的概念也是NgRx数据的基础。所有实体都属于同一实体类型的集合。Customer实体属于Customers集合。

即使实体类型只有一个实例，它也必须保存在实体集合中:可能是包含单个元素的集合。

#### Defining the entities

EntityMetadataMap告诉NgRx Data有关您的实体。为每个实体名称在集合中添加一个属性。

const entityMetadata: EntityMetadataMap = {

Hero: {},

Villain: {}

};

// because the plural of "hero" is not "heros"

const pluralNames = { Hero: 'Heroes' };

export const entityConfig = {

entityMetadata,

pluralNames

};

导出在AppModule中注册实体配置时使用的实体配置。

#### **Registering the entity store**

创建实体配置后，需要将其放入NgRx的根存储中。这是通过导入entityConfig，然后将其传递给EntityDataModule.forRoot()函数来完成的。

@NgModule({

imports: [

HttpClientModule,

StoreModule.forRoot({}),

EffectsModule.forRoot([]),

*EntityDataModule.forRoot(entityConfig)*

]

})

export class AppModule {}

## Creating entity data services

NgRx数据通过在服务类中扩展EntityCollectionServiceBase来处理服务器上的数据创建、检索、更新和删除。

@Injectable({ providedIn: 'root' })

export class HeroService extends EntityCollectionServiceBase<Hero> {

constructor(serviceElementsFactory: EntityCollectionServiceElementsFactory) {

super('Hero', serviceElementsFactory);

}

}

## Using NgRx Data in components

要访问实体数据，组件应该注入实体数据服务。

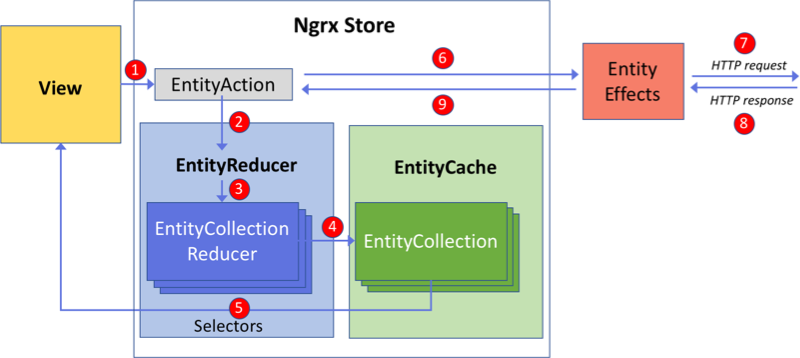
***架构总述***

您可以用几行实体元数据向NgRx数据描述您的实体模型，然后让库来完成其余的工作。

您的组件注入一个NgRx Data EntityCollectionService，并调用一个或多个标准命令方法集来调度操作。

您的组件还订阅了服务的一个或多个Observable选择器，以反应地处理和显示由这些命令产生的实体状态更改。

NgRx data实际上只是NgRx的底层。数据以典型的NgRx方式流动。下图演示了持久化EntityAction(比如Hero实体类型的QUERY\_ALL)的过程。



1 - 视图/组件调用EntityCollectionService.getAll()，它将英雄的QUERY\_ALL EntityAction分派到存储。

2 - NgRx启动…

1. NgRx数据EntityReducer读取操作的entityName属性(本例中为Hero)，并将操作和现有实体集合状态转发给英雄的EntityCollectionReducer。

(2)集合reducer根据操作的entityOp (operation)属性选择开关用例。这种情况下将操作和集合状态处理为一个新的(更新的)英雄集合。

(3)存储区使用更新后的集合更新状态树中的实体缓存。

(4)NgRx observable选择器检测并向视图中的订阅者报告变化(如果有的话)。

3 - 初始的EntityAction然后进入到EntityEffects。

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

5 - 其 effec是将HTTP响应转换为带有英雄的新的成功操作(或者请求失败时的错误操作)。

6 - NgRx效果将该操作分派到存储，该存储重申第2步，即使用英雄更新集合并刷新视图。

**Entity Metadata**

NgRx数据库在NgRx存储中维护实体收集数据的缓存。

您可以通过Entity Metadat告诉NgRx数据库关于这些集合及其包含的实体的信息。

集合中的实体属于相同的实体类型。每个实体类型都显示为NgRx Data EntityMetadata<T>接口的命名实例。

您可以在EntityMetadataMap中同时为多个实体指定元数据。

下面是一个EntityMetadataMap示例，它类似于演示应用程序中定义了两个实体(Hero和Villain)的元数据的示例。

export const appEntityMetadata: EntityMetadataMap = {

Hero: {

/\* optional settings \*/

filterFn: nameFilter,

sortComparer: sortByName

},

Villain: {

villainSelectId, // necessary if key is not `id`

/\* optional settings \*/

entityName: 'Villain', // optional because same as map key

filterFn: nameAndSayingFilter,

entityDispatcherOptions: { optimisticAdd: true, optimisticUpdate: true }

}

};

**Register metadata**

必须向NgRx Data EntityDefinitionService注册元数据。

注册元数据最简单的方法是为整个应用程序定义一个单一的EntityMetadataMap，并在初始化NgRx数据库的地方指定它:

EntityDataModule.forRoot({

...

entityMetadata: appEntityMetadata,

...

})

如果你在几个不同的、快速加载的Angular模块中定义了实体，你就可以通过多提供程序为每个模块添加元数据。

{ provide: ENTITY\_METADATA\_TOKEN, multi: true, useValue: someEntityMetadata }

这种技术对延迟加载的模块不起作用。在延迟加载模块到达时，ENTITY\_METADATA\_TOKEN提供程序已经设置并使用了。

模块应该注入EntityDefinitionService，并直接使用其中一个注册方法注册元数据。

@NgModule({...})

class LazyModule {

constructor(eds: EntityDefinitionService) {

eds.registerMetadataMap(this.lazyMetadataMap);

}

...

}

**Metadata Properties**

EntityMetadata<T>接口描述了实体类型的各个方面，这些方面告诉NgRx数据库如何管理类型为T的实体数据集合。

类型T是应用程序对该实体的TypeScript表示;它可以是接口或类。