点这里进入ABP系列文章总目录

基于DDD的现代ASP.NET开发框架--ABP系列之11、ABP领域层——仓储（Repositories）

ABP是“ASP.NET Boilerplate Project (ASP.NET样板项目)”的简称。

ABP的官方网站：http://www.aspnetboilerplate.com

ABP在Github上的开源项目：https://github.com/aspnetboilerplate

本文由**台湾-小张**提供翻译

仓储定义：“在领域层和数据映射层的中介,使用类似集合的接口来存取领域对象”(Martin Fowler)。

实际上，仓储被用于领域对象在数据库上的操作(实体Entity和值对象Value types)。一般来说,我们针对不同的实体(或聚合根Aggregate Root)会创建相对应的仓储。

IRepository接口

在ABP中,仓储类要实现IRepository接口。最好的方式是针对不同仓储对象定义各自不同的接口。

针对Person实体的仓储接口声明的示例如下所示:

```
public interface IPersonRepository : IRepository<Person>
```

{

}

IPersonRepository继承自IRepository<TEntity>，用来定义Id的类型为int(Int32)的实体。如果你的实体Id数据类型不是int,你可以继承IRepository<TEntity, TPrimaryKey>接口，如下所示:

public interface IPersonRepository : IRepository<Person, long>

{

}

对于仓储类，IRepository定义了许多泛型的方法。比如: Select,Insert,Update,Delete方法(CRUD操作)。在大多数的时候,这些方法已足已应付一般实体的需要。如果这些方对于实体来说已足够,我们便不需要再去创建这个实体所需的仓储接口/类。在Implementation章节有更多细节。

(1)查询(Query)

IRepository定义了从数据库中检索实体的常用方法。

取得单一实体(Getting single entity)

TEntity Get(TPrimaryKey id);Task<TEntity> GetAsync(TPrimaryKey id);TEntity Single(Expression<Func<TEntity, bool>> predicate);TEntity FirstOrDefault(TPrimaryKey id);Task<TEntity> FirstOrDefaultAsync(TPrimaryKey id);TEntity FirstOrDefault(Expression<Func<TEntity, bool>> predicate);Task<TEntity> FirstOrDefaultAsync(Expression<Func<TEntity, bool>> predicate);TEntity Load(TPrimaryKey id);

Get方法被用于根据主键值(Id)取得对应的实体。当数据库中根据主键值找不到相符合的实体时,它会抛出例外。Single方法类似Get方法,但是它的输入参数是一个表达式而不是主键

值(Id)。因此,我们可以写Lambda表达式来取得实体。示例如下:

var person = _personRepository.Get(42);var person = _personRepository.Single(p => o.Name == "Halil ibrahim Kalkan");

注意,Single方法会在给出的条件找不到实体或符合的实体超过一个以上时,都会抛出例外。

FirstOrDefault也一样,但是当没有符合Lambda表达式或Id的实体时,会回传null(取代抛出异常)。当有超过一个以上的实体符合条件,它只会返回第一个实体。

Load并不会从数据库中检索实体,但它会创建延迟执行所需的代理对象。如果你只使用Id属性,实际上并不会检索实体,它只有在你存取想要查询实体的某个属性时才会从数据库中查询实体。当有性能需求的时候,这个方法可以用来替代Get方法。Load方法在NHibernate与ABP的整合中也有实现。如果ORM提供者(Provider)没有实现这个方法,Load方法运行的会和Get方法一样。

ABP有些方法具有异步(Async)版本，可以应用在异步开发模型上(见Async方法相关章节)。

取得实体列表(Getting list of entities)

List<TEntity> GetAllList();Task<List<TEntity>> GetAllListAsync();List<TEntity> GetAllList(Expression<Func<TEntity, bool>> predicate);Task<List<TEntity>> GetAllListAsync(Expression<Func<TEntity, bool>> predicate);IQueryable<TEntity> GetAll();

GetAllList被用于从数据库中检索所有实体。重载并且提供过滤实体的功能,如下:

var allPeople = _personRespository.GetAllList();var somePeople = _personRepository.GetAllList(person => person.IsActive && person.Age > 42);

GetAll返回IQueryable<T>类型的对象。因此我们可以在调用完这个方法之后进行Linq操作。示例:

//例子一var query = from person in _personRepository.GetAll()where person.IsActiveorderby person.Nameselect person;var people = query.ToList();//例

子二List<Person> personList2 = _personRepository.GetAll().Where(p => p.Name.Contains("H")).OrderBy(p => p.Name).Skip(40).Take(20).ToList();

如果调用GetAll方法,那么几乎所有查询都可以使用Linq完成。甚至可以用它来编写Join表达式。

说明：关于IQueryable<T>

当你调用GetAll这个方法在Repository对象以外的地方,必定会开启数据库连接。这是因为IQueryable<T>允许延迟执行。它会直到你调用ToList方法或在forEach循环上(或是一些存取已查询的对象方法)使用IQueryable<T>时,才会实际执行数据库的查询。因此,当你调用ToList方法时,数据库连接必需是启用状态。我们可以使用ABP所提供的UnitOfWork特性在调用的方法上来实现。注意,Application Service方法预设都已经是UnitOfWork。因此,使用了GetAll方法就不需要如同Application Service的方法上添加UnitOfWork特性。

有些方法拥有异步版本,可应用在异步开发模型(见关于async方法章节)。

自定义返回值(Custom return value)

ABP也有一个额外的方法来实现IQueryable<T>的延迟加载效果,而不需要在调用的方法上添加UnitOfWork这个属性卷标。

T Query<T>(Func<IQueryable<Tentity>,T> queryMethod);

查询方法接受Lambda(或一个方法)来接收IQueryable<T>并且返回任何对象类型。示例如下:

var people = _personRepository.Query(q => q.Where(p => p.Name.Contains("H")).OrderBy(p => p.Name).ToList());

因为是采用Lambda(或方法)在仓储对象的方法中执行,它会在数据库连接开启之后才被执行。你可以返回实体集合,或一个实体,或一个具部份字段(注: 非Select *)或其它执行查询后的查询结果集。

(2)新增(insert)

IRepository接口定义了简单的方法来提供新增一个实体到数据库:

TEntity Insert(TEntity entity);Task<TEntity> InsertAsync(TEntity entity);TPrimaryKey InsertAndGetId(TEntity entity);Task<TPrimaryKey> InsertAndGetIdAsync(TEntity entity);TEntity InsertOrUpdate(TEntity entity);Task<TEntity> InsertOrUpdateAsync(TEntity entity);TPrimaryKey InsertOrUpdateAndGetId(TEntity entity);Task<TPrimaryKey> InsertOrUpdateAndGetIdAsync(TEntity entity);

新增方法会新增实体到数据库并且返回相同的已新增实体。InsertAndGetId方法返回新增实体的标识符(Id)。当我们采用自动递增标识符值且需要取得实体的新产生标识符值时非常好用。InsertOfUpdate会新增或更新实体,选择那一种是根据Id是否有值来决定。最后,InsertOrUpdatedAndGetId会在实体被新增或更新后返回Id值。

所有的方法都拥有异步版本可应用在异步开发模型(见关于异步方法章节)

(3)更新(UPDATE)

IRepository定义一个方法来实现更新一个已存在于数据库中的实体。它更新实体并返回相同的实体对象。

TEntity Update(TEntity entity);Task<TEntity> UpdateAsync(TEntity entity);

(4)删除(Delete)

IRepository定了一些方法来删除已存在数据库中实体。

void Delete(TEntity entity);Task DeleteAsync(TEntity entity);void Delete(TPrimaryKey id);Task DeleteAsync(TPrimaryKey id);void Delete(Expression<Func<TEntity, bool>> predicate);Task DeleteAsync(Expression<Func<TEntity, bool>> predicate);

第一个方法接受一个现存的实体,第二个方法接受现存实体的Id。

最后一个方法接受一个条件来删除符合条件的实体。要注意,所有符合predicate表达式的实体会先被检索而后删除。因此,使用上要很小心,这是有可能造成许多问题,假如果有太多实体符合条件。

所有的方法都拥有async版本来应用在异步开发模型(见关于异步方法章节)。

(5)其它方法(others)

IRepository也提供一些方法来取得数据表中实体的数量。

int Count();Task<int> CountAsync();int Count(Expression<Func<TEntity, bool>> predicate);Task<int> CountAsync(Expression<Func<TEntity, bool>> predicate);Long LongCount();Task<long> LongCountAsync();Long LongCount(Expression<Func<TEntity, bool>> predicate);Task<long> LongCountAsync(Expression<TEntity, bool>> predicate);

所有的方法都拥有async版本被应用在异步开发模型(见关于异步方法章节)。

(6)关于异步方法(About Async methods)

ABP支持异步开发模型。因此,仓储方法拥有Async版本。在这里有一个使用异步模型的application service方法的示例:

public class PersonAppService : AbpWpfDemoAppServiceBase, IPersonAppService{ private readonly IRepository<Person> _personRepository; public PersonAppService(IRepository<Person> personRepository) { _personRepository = personRepository; } public async Task<GetPeopleOutput> GetAllPeople() { var people = await _personRepository.GetAllListAsync(); return new GetPeopleOutput { People = Mapper.Map<List<PersonDto>>(people) }; }}

GetAllPeople方法是异步的并且使用GetAllListAsync与await保留关键字。

Async不是在每个ORM框架都有提供。

上例是从EF所提供的异步能力。如果ORM框架没有提供Async的仓储方法则它会以同步的方式操作。同样地,举例来说,InsertAsync操作起来和EF的新增是一样的,因为EF会直到单元作业(unit of work)完成之后才会写入新实体到数据库中(DbContext.SaveChanges)。

仓储的实现

ABP在设计上是采取不指定特定ORM框架或其它存取数据库技术的方式。只要实现IRepository接口,任何框架都可以使用。

仓储要使用NHibernate或EF来实现都很简单。见实现这些框架在ABP仓储对象上一文:

- NHibernate
- EntityFramework

当你使用NHibernate或EntityFramework,如果提供的方法已足够使用,你就不需要为你的实体创建仓储对象了。我们可以直接注入IRepository<TEntity>(或IRepository<TEntity, TPrimaryKey>)。下面的示例为application service使用仓储对象来新增实体到数据库:

```
public class PersonAppService : IPersonAppService{    private readonly IRepository<Person> _personRepository;    public PersonAppService(IRepository<Person> personRepository)    { _personRepository = personRepository;    }    public void CreatePerson(CreatePersonInput input)    {                person = new Person { Name = input.Name, EmailAddress = input.EmailAddress }; _personRepository.Insert(person);    }}
```

PersonAppService的建构子注入了IRepository<Person>并且使用其Insert方法。当你有需要为实体创建一个客制的仓储方法,那么你就应该创建一个仓储类给指定的实体。

管理数据库连接

数据库连接的开启和关闭，在仓储方法中,ABP会自动化的进行连接管理。

当仓储方法被调用后,数据库连接会自动开启且启动事务。当仓储方法执行结束并且返回以后,所有的实体变化都会被储存, 事务被提交并且数据库连接被关闭,一切都由ABP自动化的控制。如果仓储方法抛出任何类型的异常,事务会自动地回滚并且数据连接会被关闭。上述所有操作在实现了IRepository接口的仓储类所有公开的方法中都可以被调用。

如果仓储方法调用其它仓储方法(即便是不同仓储的方法),它们共享同一个连接和事务。连接会由仓储方法调用链最上层的那个仓储方法所管理。更多关于数据库管理,详见UnitOfWork文件。

储的生命周期

所有的仓储对象都是暂时性的。这就是说,它们是在有需要的时候才会被创建。ABP大量的使用依赖注入，当仓储类需要被注入的时候,新的类实体会由注入容器会自动地创建。见相根据注入文件有更多信息。

仓储的最佳实践

- 对于一个T类型的实体,是可以使用IRepository<T>。但别任何情况下都创建定制化的仓储,除非我们真的很需要。预定义仓储方法已经足够应付各种案例。
- 假如你正创建定制的仓储(可以实现IRepository<TEntity>)
    - 仓储类应该是无状态的。这意味着, 你不该定义仓储等级的状态对象并且仓储方法的调用也不应该影响到其它调用。
    - 当仓储可以使用相根据注入，尽可较少或是不相根据于其它服务。

希望更多国内的架构师能关注到ABP这个项目，也许这其中有能帮助到您的地方，也许有您的参与，这个项目可以发展得更好。

欢迎加ABP架构设计交流QQ群：134710707

ABP架构设计交流群

点这里进入ABP系列文章总目录