**安装**

要使用Abp封装的EFCore，你需要安装如下包

*Volo.Abp.EntityFrameworkCore*

然后在你的Module类上指定依赖

[DependsOn(typeof(AbpEntityFrameworkCoreModule))]

public class MyModule : AbpModule

{

    //...

}

然后创建你的DBContext

public class MyDbContext : AbpDbContext<MyDbContext>

{

    //...在这里添加 DbSet properties

    public MyDbContext(DbContextOptions<MyDbContext> options)

        : base(options)

    {

    }

}

注：如果你使用启动模板生成的项目，默认已集成EFCore

**配置实体**

你可以在DbContext的OnModelCreating中配置你的实体

protected override void OnModelCreating(ModelBuilder builder)

{

    //Always call the base method

    base.OnModelCreating(builder);

    builder.Entity<Book>(b =>

    {

        b.ToTable("Books");

        //Configure the base properties

        b.ConfigureByConvention();

        //Configure other properties (if you are using the fluent API)

        b.Property(x => x.Name).IsRequired().HasMaxLength(128);

    });

}

其中ConfigureByConvention是一个封装的方法，其按照约定配置实体的基本属性，建议调用他

**配置连接字符串**

你可以使用ConnectionStringName指定在appsettings.json中的连接字符串名

[ConnectionStringName("MySecondConnString")]

public class MyDbContext : AbpDbContext<MyDbContext>

{

}

**AbpDbContextOptions**

你可以在Module的ConfigureServices方法中对DbContext进行配置

Configure<AbpDbContextOptions>(options =>

{

    // 指定DbContext的数据库为SqlServer

    options.UseSqlServer();

});

**添加DbContext到依赖注入中**

使用如下方法注册我们的上下文

[DependsOn(typeof(AbpEntityFrameworkCoreModule))]

public class MyModule : AbpModule

{

    public override void ConfigureServices(ServiceConfigurationContext context)

    {

        context.services.AddAbpDbContext<MyDbContext>(options =>

        {

            // 为MyDbContext创建一个默认仓储

            options.AddDefaultRepositories();

        });

        //...

    }

}

默认情况下只有聚合根才有仓储，如果你想实体也可以使用，请做如下配置

options.AddDefaultRepositories(includeAllEntities: true);

**使用默认仓储**

进行如上配置后即可使用默认仓储了

public class BookManager : DomainService

{

    private readonly IRepository<Book, Guid> \_bookRepository;

    //inject default repository to the constructor

    public BookManager(IRepository<Book, Guid> bookRepository)

    {

        \_bookRepository = bookRepository;

    }

}

**自定义仓储**

如果你想自定义仓储，你可以进行如下操作

首先继承仓储接口

public interface IBookRepository : IRepository<Book, Guid>

{

    Task DeleteBooksByType(BookType type);

}

然后继承EfCoreRepository

public class BookRepository

    : EfCoreRepository<BookStoreDbContext, Book, Guid>, IBookRepository

{

    public BookRepository(IDbContextProvider<BookStoreDbContext> dbContextProvider)

        : base(dbContextProvider)

    {

    }

    public async Task DeleteBooksByType(BookType type)

    {

        var dbContext = await GetDbContextAsync();

        await dbContext.Database.ExecuteSqlRawAsync(

            $"DELETE FROM Books WHERE Type = {(int)type}"

        );

    }

}

**加载关联实体**

1. 首先你需要配置实体间的关系

builder.Entity<Order>(b =>

{

    b.ToTable("Orders");

    b.ConfigureByConvention();

    //定义关系

    b.HasMany(x => x.Lines)

        .WithOne(x => x.Order)

        .HasForeignKey(x => x.OrderId)

        .IsRequired();

});

builder.Entity<OrderLine>(b =>

{

    b.ToTable("OrderLines");

    b.ConfigureByConvention();

});

1. 然后使用IRepository的WithDetailsAsync方法加载

public async Task TestWithDetails(Guid id)

{

    //通过包含子集合获取一个 IQueryable<T>

    var queryable = await \_orderRepository.WithDetailsAsync(x => x.Lines);

    //应用其他的 LINQ 扩展方法

    var query = queryable.Where(x => x.Id == id);

    //执行此查询并获取结果

    var order = await AsyncExecuter.FirstOrDefaultAsync(query);

}

如果你需要包含多个集合，WithDetailsAsync方法可以接收多个表达式参数

1. 默认加载关联

如果你想默认就自动加载这些关联，那你可以在你的EntityFrameworkCore项目中进行如下配置

Configure<AbpEntityOptions>(options =>

{

    options.Entity<Order>(orderOptions =>

    {

        orderOptions.DefaultWithDetailsFunc = query => query.Include(o => o.Lines);

    });

});

这样你在使用WithDetailsAsync方法时可以不用传任何参数

**Get/Find 方法**

仓储的GetAsync 和 FindAsync 方法上有一个参数includeDetails，其有默认值为true，表示自动加载实体下的单个关联对象

**EnsurePropertyLoadedAsync / EnsureCollectionLoadedAsync**

仓储的这2个方法用于显示加载实体的单个导航属性和集合

public async Task TestWithDetails(Guid id)

{

    var order = await \_orderRepository.GetAsync(id, includeDetails: false);

    //order.Lines 此时是空的

    await \_orderRepository.EnsureCollectionLoadedAsync(order, x => x.Lines);

    //order.Lines 被填充

}

**延时加载**

如果你不想显示加载导航属性，想要在访问时自动加载，那么你可以进行如下配置

* 安装 Microsoft.EntityFrameworkCore.Proxies 包到你的项目
* 为你的 DbContext 配置 UseLazyLoadingProxies (在 EF Core 项目的模块的 ConfigureServices 方法中)

Configure<AbpDbContextOptions>(options =>

{

    options.PreConfigure<MyCrmDbContext>(opts =>

    {

        opts.DbContextOptions.UseLazyLoadingProxies(); //启用延时加载

    });

    options.UseSqlServer();

});

* 使你的导航属性和集合是 virtual

public virtual ICollection<OrderLine> Lines { get; set; } //虚集合

public virtual Order Order { get; set; } //虚导航属性

* 现在你可以自由访问导航属性和集合了

public async Task TestWithDetails(Guid id)

{

    var order = await \_orderRepository.GetAsync(id);

    //order.Lines 此时是空的

    var lines = order.Lines;

    //order.Lines 被填充 (延时加载)

}

**访问 EF Core API**

如果你想要方法EFCore的Api，你可以使用仓储的如下2个方法

public async Task TestAsync()

{

    var dbContext = await \_orderRepository.GetDbContextAsync();

    var dbSet = await \_orderRepository.GetDbSetAsync();

}