

从零开始实现DDD应用

第四章 Persistence

作者：贺传珺

仓储

从本章开始要连接数据库，把我们的聚合根持久化（Persistence）到数据库中。本章使用PostgreSQL作为数据库，安装过程略过，自行去查阅资料。其他关系型数据库也可，代码上可能会有一些细微的区别。ORM使用EF Core（下一章会用到Dapper）。

仓储

现在很多人觉得仓库层（Repository）是对ORM的过度封装。确实，如果你在写三层架构它就毫无意义。因为你能在服务层直接去调用ORM的功能和数据库进行通信。但是在DDD里，如果不使用仓储层，领域层可能要处理和数据库相关的事务，这会让领域层不够干净。

仓储

首先，我们在Domain项目里新建一个IApartmentRepository接口。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace RJRentalOfHousing.Domain
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public interface IPartmentRepository
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        Task<Apartment> Load(ApartmentId id);
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        Task Add(Apartment entity);
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        Task<bool> Exists(ApartmentId id);
    }
}
```

工作单元

为保证业务事务的一致性，我们需要一个工作单元来保存或者回滚操作。在Framework项目中新建一个IUnitOfWork接口。

```
lentalOfHousing.Framework
1 namespace RJRentalOfHousing.Framework
2 {
3     0 个引用 | 0 项更改 | 0 名作者, 0 项更改
4     public interface IUnitOfWork
5     {
6         0 个引用 | 0 项更改 | 0 名作者, 0 项更改
7         Task Commit();
8     }
```

应用仓储和工作单元

修改ApartmentApplicationService，把我们刚才写的接口加上。

```
using RJRentalOfHousing.Domain;
using RJRentalOfHousing.Framework;
using static RJRentalOfHousing.Contracts.Apartments;

namespace RJRentalOfHousing
{
    3 个引用 | HCJ, 19 小时前 | 1 名作者, 1 项更改
    public class ApartmentsApplicationService : IApplicationService
    {
        private readonly IApartmentRepository _repository;
        private readonly IUnitOfWork _unitOfWork;
        private readonly ICurrencyLookup _currencyLookup;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public ApartmentsApplicationService(IApartmentRepository repository, IUnitOfWork unitOfWork, ICurrencyLookup currencyLookup)
        {
            _repository = repository;
            _unitOfWork = unitOfWork;
            _currencyLookup = currencyLookup;
        }
    }
}

1 个引用 | HCJ, 19 小时前 | 1 名作者, 1 项更改
private async Task HandleCreate(V1.Create cmd)
{
    if (await _repository.Exists(new ApartmentId(cmd.Id)))
        throw new InvalidOperationException($"{cmd.Id} 已存在");
    var apartment = new Apartment(new ApartmentId(cmd.Id), new UserId(cmd.OwnerId));
    await _repository.Add(apartment);
    await _unitOfWork.Commit();
}
```

应用仓储和工作单元

6 个引用 | HCJ, 19 小时前 | 1 名作者, 1 项更改

```
private async Task HandleUpdate(Guid ApartmentId, Action<Apartment> operation)
{
    var apartment = await _repository.Load(new ApartmentId(ApartmentId));
    if (apartment == null)
        throw new InvalidOperationException($"{ApartmentId} 不存在");
    operation(apartment);
    await _unitOfWork.Commit();
}
```

应用仓储和工作单元

```
public async Task Handle(object command)
{
    switch(command)
    {
        case V1.Create cmd:
            await HandleCreate(cmd);
            break;
        case V1.SetArea cmd:
            await HandleUpdate(cmd.Id, x => x.SetArea(new Area(cmd.Areas)));
            break;
        case V1.SetAddress cmd:
            await HandleUpdate(cmd.Id, x => x.SetAddress(new Address(cmd.Address)));
            break;
        case V1.SetRent cmd:
            await HandleUpdate(cmd.Id, x => Price.FromDecimal(cmd.Rent, cmd.CurrencyCode, _currencyLookup));
            break;
        case V1.SetDeposit cmd:
            await HandleUpdate(cmd.Id, x => Price.FromDecimal(cmd.Deposit, cmd.CurrencyCode, _currencyLookup));
            break;
        case V1.SetRemark cmd:
            await HandleUpdate(cmd.Id, x => x.SetRemark(cmd.Remark));
            break;
        case V1.SentForReview cmd:
            await HandleUpdate(cmd.Id, x => x.RequestToPublish());
            break;
        default:
            throw new InvalidOperationException($"未知的命令类型: {command.GetType().FullName}");
    }
}
```


应用仓储和工作单元

由于程序处理过程中可能会出现各种意外情况，我们不能直接忽略这些信息，而要把它捕捉并记录。现在修改 ApartmentCommandsApi。首先我们要安装一个日志包：Serilog。安装过程略过。

```
using Microsoft.AspNetCore.Mvc;
using static RJRentalOfHousing.ContractsApartments;

namespace RJRentalOfHousing
{
    [Route("/apartment")]
    public class ApartmentCommandsApi : Controller
    {
        private readonly ApartmentsApplicationService _applicationService;
        private static Serilog.ILogger Log = Serilog.Log.ForContext<ApartmentCommandsApi>();

        public ApartmentCommandsApi(ApartmentsApplicationService applicationService) => _applicationService = applicationService;
    }
}
```

应用仓储和工作单元

```
private async Task<IActionResult> HandleRequest<T>(T request, Func<T, Task> handler)
{
    try
    {
        Log.Debug("正在处理类型为{0}的HTTP请求", typeof(T).Name);
        await handler(request);
        return Ok();
    }
    catch (Exception e)
    {
        Log.Error("处理请求失败", e);
        return new BadRequestObjectResult(new { error = e.Message, stackTrace = e.StackTrace });
    }
}
```

[HttpPost]

0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改

```
public async Task<IActionResult> Post([FromBody] V1.Create request) => await HandleRequest(request, _applicationService.Handle);
```

[Route("area")]

[HttpPut]

0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改

```
public async Task<IActionResult> Put([FromBody] V1.SetArea request) => await HandleRequest(request, _applicationService.Handle);
```

[Route("address")]

[HttpPut]

0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改

```
public async Task<IActionResult> Put([FromBody] V1.SetAddress request) => await HandleRequest(request, _applicationService.Handle);
```

应用仓储和工作单元

```
[Route("rent")]
[HttpPut]
0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public async Task<IActionResult> Put([FromBody] V1.SetRent request) => await HandleRequest(request, _applicationService.Handle);

[Route("deposit")]
[HttpPut]
0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public async Task<IActionResult> Put([FromBody] V1.SetDeposit request) => await HandleRequest(request, _applicationService.Handle);

[Route("remark")]
[HttpPut]
0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public async Task<IActionResult> Put([FromBody] V1.SetRemark request) => await HandleRequest(request, _applicationService.Handle);

[Route("publish")]
[HttpPut]
0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public async Task<IActionResult> Put([FromBody] V1.SentForReview request) => await HandleRequest(request, _applicationService.Handle);
```

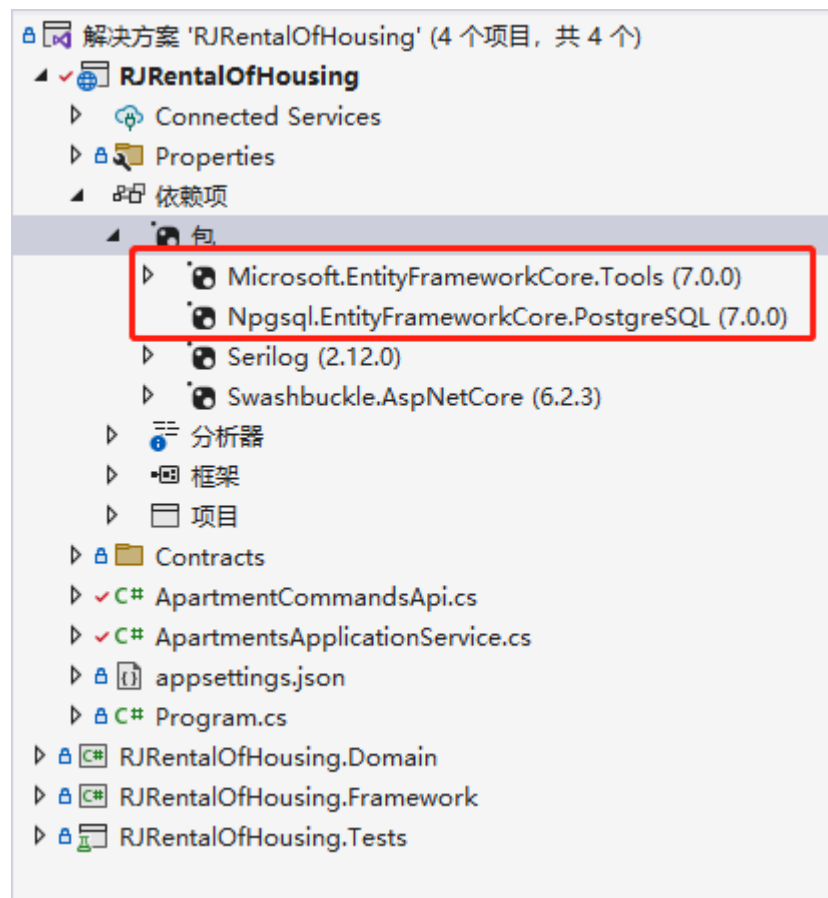
实现Entity Framework Core仓储

在实现仓储前我们需要做点准备，在RJRentalOfHousing项目中安装两个包：

- Npgsql.EntityFrameworkCore.PostgreSQL
- Microsoft.EntityFrameworkCore.Tools

版本号最新即可，在写这个教程的时候最新的版本为7.0.0。

实现Entity Framework Core仓储



实现Entity Framework Core仓储

现在我们需要用到基础设施，由于我们只有HTTP一个端点和演示方便，我们把基础设施层和API层放在一起。在API项目中新建一个文件夹，命名为Infrastructure，并且在里面新建DbContext，命名为ApartmentDbContext。

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using RJRentalOfHousing.Domain;

namespace RJRentalOfHousing.Infrastructure
{
    8 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class ApartmentDbContext : DbContext
    {
        private readonly ILoggerFactory _loggerFactory;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public ApartmentDbContext(DbContextOptions<ApartmentDbContext> options, ILoggerFactory loggerFactory) : base(options)
        {
            => _loggerFactory = loggerFactory;

            3 个引用 | 0 项更改 | 0 名作者, 0 项更改
            public DbSet<Apartment> Apartments { get; set; }
        }
    }
}
```

实现Entity Framework Core仓储

在继续实现DbContext之前还有件事需要做，EFCore需要知道每个实体的主键，这个主键不能在基类或者其他地方，所以我们需要修改Apartment类，加上一个字段和做一些处理。

```
2 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Apartment(ApartmentId id, UserId ownerId)
{
    Pictures = new List<Picture>();
    Apply(new Events.ApartmentCreated
    {
        Id = id,
        OwnerId = ownerId
    });
}
```

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public Guid ApartmentId { get; internal set; }
```

```
3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Area Areas { get; internal set; }
```

```
3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Address Address { get; internal set; }
```

```
3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
protected override void When(object @event)
{
    switch(@event)
    {
        case Events.ApartmentCreated e:
            Id = new ApartmentId(e.Id);
            Owner = new UserId(e.OwnerId);
            State = ApartmentState.Created;
            ApartmentId = e.Id;
            break;
        case Events.ApartmentAreaUpdated e:
            Areas = new Area(e.Areas);
            break;
        // ...
    }
}
```

实现Entity Framework Core仓储

现在继续实现DbContext，把下面的代码加入到DbContext文件中。

```
public class ApartmentEntityTypeConfiguration : IEntityTypeConfiguration<Apartment>
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public void Configure(EntityTypeBuilder<Apartment> builder)
    {
        builder.HasKey(x => x.ApartmentId);
    }
}

0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public static class AppBuilderDatabaseExtensions
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public static void EnsureDatabase(this IApplicationBuilder app)
    {
        var context = app.ApplicationServices.GetService<ApartmentDbContext>();
        if (!context.Database.EnsureCreated())
            context.Database.Migrate();
    }
}
```


实现Entity Framework Core仓储

```
public class ApartmentDbContext : DbContext
{
    private readonly ILoggerFactory _loggerFactory;

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public ApartmentDbContext(DbContextOptions<ApartmentDbContext> options, ILoggerFactory loggerFactory) : base(options)
        => _loggerFactory = loggerFactory;

    3 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public DbSet<Apartment> Apartments { get; set; }

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseLoggerFactory(_loggerFactory);
        optionsBuilder.EnableSensitiveDataLogging();
    }

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new ApartmentEntityTypeConfiguration());
    }
}
```

特别注意：请勿在正式环境中加上EnableSensitiveDataLogging(), 会引发严重的安全问题。

实现Entity Framework Core仓储

现在去用EFCore实现工作单元（UnitOfWork）。在API项目中的Infrastructure文件夹中新建一个EFUnitOfWork类。

```
using RJRentalOfHousing.Infrastructure.EFUnitOfWork;

using RJRentalOfHousing.Framework;

namespace RJRentalOfHousing.Infrastructure
{
    1 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class EFUnitOfWork : IUnitOfWork
    {
        private readonly ApartmentDbContext _dbContext;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public EFUnitOfWork(ApartmentDbContext dbContext) => _dbContext = dbContext;

        3 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Task Commit() => _dbContext.SaveChangesAsync();
    }
}
```

实现Entity Framework Core仓储

继续用EFCore来实现仓储，在Infrastructure文件夹中新建一个ApartmentRepository类。

```
using RJRentalOfHousing.Domain;

namespace RJRentalOfHousing.Infrastructure
{
    1 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class ApartmentRepository : IApartmentRepository
    {
        private readonly ApartmentDbContext _dbContext;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public ApartmentRepository(ApartmentDbContext dbContext) => _dbContext = dbContext;

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public async Task Add(Apartment entity) => await _dbContext.Apartments.AddAsync(entity);

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public async Task<bool> Exists(ApartmentId id) => await _dbContext.Apartments.FindAsync(id.Value) != null;

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public async Task<Apartment> Load(ApartmentId id) => await _dbContext.Apartments.FindAsync(id.Value);
    }
}
```

实现Entity Framework Core仓储

为让程序能跑起来，我们还需要实现一下ICurrencyLookup接口，在API项目中的Infrastructure新建一个FixedCurrencyLookup类。

```
using RJRentalOfHousing.Domain;

namespace RJRentalOfHousing.Infrastructure
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class FixedCurrencyLookup : ICurrencyLookup
    {
        private static readonly IEnumerable<CurrencyDetails> _currencies =
            new[]
            {
                new CurrencyDetails
                {
                    CurrencyCode = "CNY",
                    DecimalPlaces = 2,
                    InUse = true
                },
                new CurrencyDetails
                {
                    CurrencyCode = "EUR",
                    DecimalPlaces = 2,
                    InUse = true
                },
                new CurrencyDetails
                {
                    CurrencyCode = "USD",
                    DecimalPlaces = 2,
                    InUse = true
                }
            };
    }
}
```

2 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
public CurrencyDetails FindCurrency(string currencyCode)
{
    var currency = _currencies.FirstOrDefault(x => x.CurrencyCode == currencyCode);
    return currency ?? CurrencyDetails.None;
}
```

实现Entity Framework Core仓储

还有最后一步要做：修改Program类，注册我们的服务。

```
using Microsoft.EntityFrameworkCore;
using RJRentalOfHousing;
using RJRentalOfHousing.Domain;
using RJRentalOfHousing.Framework;
using RJRentalOfHousing.Infrastructure;

var builder = WebApplication.CreateBuilder(args);

const string connectionString = "Host=localhost;Database=Chapter4;Username=postgres;Password=123456";

builder.Services.AddEntityFrameworkNpgsql().AddDbContext<ApartmentDbContext>(options => options.UseNpgsql(connectionString), ServiceLifetime.Singleton);
builder.Services.AddSingleton<IUnitOfWork, EFUnitOfWork>();
builder.Services.AddSingleton<ICurrencyLookup, FixedCurrencyLookup>();
builder.Services.AddScoped<IApartmentRepository, ApartmentRepository>();
builder.Services.AddScoped<ApartmentsApplicationService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.EnsureDatabase();

app.UseHttpsRedirection();
```

阻抗不匹配

按下F5，哦豁，报错了！

0 个引用 | 0 项更改 | 0 名作者，0 项更改

```
public static class AppBuilderDatabaseExtensions
```

```
{
```

1 个引用 | 0 项更改 | 0 名作者，0 项更改

```
    public static void EnsureDatabase(this IApplicationBuilder app)
```

```
    {
```

```
        var context = app.ApplicationServices.GetService<ApartmentDbContext>();
```

```
        if (!context.Database.EnsureCreated())
```

```
            context.Database.Migrate();
```

```
    }
```

```
}
```

未经处理的异常

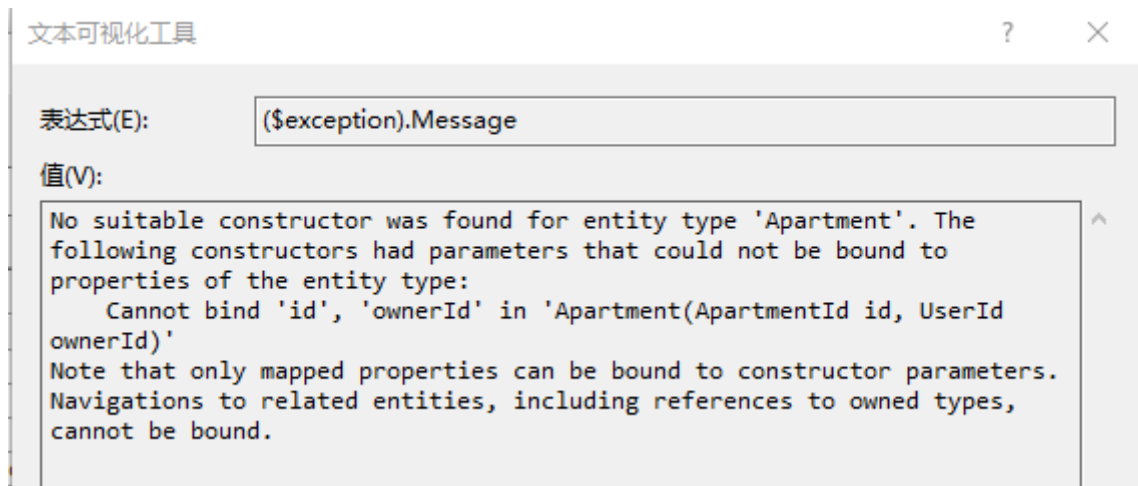
System.InvalidOperationException: "No suitable constructor was found for entity type 'Apartment'. The following constructors had parameters that could not be bound to properties of the entity type:
 Cannot bind 'Id' to 'model' in 'Apartment(ApartmentId id'.

[显示调用堆栈](#) | [查看详细信息](#) | [复制详细信息](#) | [启动 Live Share 会话](#)

▸ 异常设置

阻抗不匹配

报错的原因很简单：代码写得有问题。当然不是你敲得有问题，是因为现在遇到了ORM的老大难：阻抗不匹配。意思是持久化的模型和业务模型不一致。让我们细看一下报错信息。



这个错误的意思是，Apartment类没有合适的构造函数给ORM调用，它需要一个无参数的构造函数。这简单，修改Apartment类，给它加上一个无参数的构造函数，然后再次运行。

```
public class Apartment:AggregateRoot<ApartmentId>  
{  
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改  
    protected Apartment() { }  
    2 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改  
    public Apartment(ApartmentId id, UserId ownerId)  
    {  
        Pictures = new List<Picture>();  
        Apply(new Events.ApartmentCreated  
        {  
            Id = id,  
            OwnerId = ownerId  
        });  
    }  
}
```

阻抗不匹配

继续按下F5运行，又报错了，看看这次是报了啥错。



和刚才一样，Picture类也需要一个无参数的构造函数给ORM调用，还是像刚才一样，在Picture中加一个。

阻抗不匹配

修改Picture类，因为Picture继承Entity类，而Entity类定义了一个带一个委托类型参数的构造函数，所以在Entity类里也要加上一个构造函数。

```
4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改  
public abstract class Entity<TId>: IInternalEventHandler  
{  
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改  
    protected Entity() {}  
  
    private readonly Action<object> _applier;  
  
    1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改  
    protected Entity(Action<object> applier) => _applier = applier;  
  
    0 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
```

```
public class Picture : Entity<PictureId>  
{  
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改  
    protected Picture() {}  
  
    1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改  
    internal Picture(Action<object> applier) : base(applier) {}  
}
```

阻抗不匹配

继续按F5运行，又报错了，还是一样的内容，修复方法同上，把所有的值对象都加上一个Protected的无参数构造函数，过程略，可以查看本章的配套代码。

0 个引用 | 0 项更改 | 0 名作者，0 项更改

```
public static class AppBuilderDatabaseExtensions
```

```
{
```

```
1 个引用 | 0 项更改 | 0 名作者，0 项更改
```

```
public static void EnsureDatabase(this IApplicationBuilder app)
```

```
{
```

```
var context = app.ApplicationServices.GetService<ApartmentDbContext>();
```

```
if (!context.Database.EnsureCreated())
```

```
context.Database.Migrate();
```

```
}
```

```
}
```

未经处理的异常

System.InvalidOperationException: "No suitable constructor was found for entity type 'Price'. The following constructors had parameters that could not be bound to properties of the entity type:
Cannot bind 'currencyCode' 'currencyCode' in 'Price'.

[显示调用堆栈](#) | [查看详细信息](#) | [复制详细信息](#) | [启动 Live Share 会话](#)

异常设置

阻抗不匹配

修改完成之后继续按F5运行项目，又出现了一个新的错误。

```
public static void EnsureDatabase(this IApplicationBuilder app)
{
    var context = app.ApplicationServices.GetService<ApartmentDbContext>();
    if (!context.Database.EnsureCreated())
        context.Database.Migrate();
}
```

未经处理的异常

System.InvalidOperationException: "The entity type 'Address' requires a primary key to be defined. If you intended to use a keyless entity type, call 'HasNoKey' in 'OnModelCreating'. For more information on keyless entity types, see <https://go.microsoft.com/fwlink/?linkid=2141943>."

[显示调用堆栈](#) | [查看详细信息](#) | [复制详细信息](#) | [启动 Live Share 会话...](#)

▸ 异常设置

这次总算是来了点新鲜的。这个报错是要求我们给Address指定一个主键，但是Address是值对象，回忆一下第一章的内容，值对象是不应该有身份标识的，接下来我们通过配置解决这个问题。

阻抗不匹配

修改ApartmentDbContext文件中的ApartmentEntityTypeConfiguration类。

```
public class ApartmentEntityTypeConfiguration : IEntityTypeConfiguration<Apartment>
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public void Configure(EntityTypeBuilder<Apartment> builder)
    {
        builder.HasKey(x => x.ApartmentId);
        builder.OwnsOne(x => x.Id);
        builder.OwnsOne(x => x.Address);
        builder.OwnsOne(x => x.Areas);
        builder.OwnsOne(x => x.Rent, r => r.OwnsOne(c => c.Currency));
        builder.OwnsOne(x => x.Deposit, r => r.OwnsOne(c => c.Currency));
        builder.OwnsOne(x => x.Owner);
        builder.OwnsOne(x => x.ApprovedBy);
    }
}
```

现在运行仍然会报错，我们需要把Picture实体也配置到Context中才能继续运行。

现在看到了一个平时很少用到的方法：OwnsOne。这个方法告诉EFCore，这个对象只是这个实体的一部分，具体请查看微软文档。

阻抗不匹配

首先要在Picture中加上所属房屋的字段和它自身的标识。

```
8 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public class Picture : Entity<PictureId>
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    protected Picture() { }

    1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
    internal Picture(Action<object> applier) : base(applier) { }

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public Guid PictureId { get; internal set; }
    4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
    public PictureSize Size { get; internal set; }
    1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
    public Uri Location { get; internal set; }
    3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
    public int Order { get; internal set; }
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public ApartmentId ParentId { get; internal set; }
```

阻抗不匹配

然后在ApartmentDbContext中加上Picture的配置。

```
public class PictureEntityTypeConfiguration : IEntityTypeConfiguration<Picture>
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public void Configure(EntityTypeBuilder<Picture> builder)
    {
        builder.HasKey(x => x.PictureId);
        builder.OwnsOne(x => x.Id);
        builder.OwnsOne(x => x.ParentId);
        builder.OwnsOne(x => x.Size);
    }
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.ApplyConfiguration(new ApartmentEntityTypeConfiguration());
    modelBuilder.ApplyConfiguration(new PictureEntityTypeConfiguration());
}
```

阻抗不匹配

按下F5启动项目，这下总算是成功启动了。

RJRentalOfHousing ^{1.0} OAS3


<https://localhost:7294/swagger/v1/swaggerjson>


ApartmentCommandsApi ^

POST	/apartment	✓
PUT	/apartment/area	✓
PUT	/apartment/address	✓
PUT	/apartment/rent	✓
PUT	/apartment/deposit	✓
PUT	/apartment/remark	✓
PUT	/apartment/publish	✓

阻抗不匹配

让我们去看看数据库里的表和字段。

名	类型	长度	小数点	不是 null	键
▶ ApartmentId	uuid	0	0	<input checked="" type="checkbox"/>	 1
Areas_Value	numeric	0	0	<input checked="" type="checkbox"/>	
Address_Value	text	0	0	<input checked="" type="checkbox"/>	
Rent_Amount	numeric	0	0	<input checked="" type="checkbox"/>	
Rent_Currency_CurrencyCode	text	0	0	<input checked="" type="checkbox"/>	
Rent_Currency_InUse	bool	0	0	<input checked="" type="checkbox"/>	
Rent_Currency_DecimalPlaces	int4	32	0	<input checked="" type="checkbox"/>	
Deposit_Amount	numeric	0	0	<input checked="" type="checkbox"/>	
Deposit_Currency_CurrencyCode	text	0	0	<input checked="" type="checkbox"/>	
Deposit_Currency_InUse	bool	0	0	<input checked="" type="checkbox"/>	
Deposit_Currency_DecimalPlaces	int4	32	0	<input checked="" type="checkbox"/>	
Owner_Value	uuid	0	0	<input checked="" type="checkbox"/>	
Remark	text	0	0	<input checked="" type="checkbox"/>	
ApprovedBy_Value	uuid	0	0	<input checked="" type="checkbox"/>	
State	int4	32	0	<input checked="" type="checkbox"/>	
Id_Value	uuid	0	0	<input type="checkbox"/>	

名	类型	长度	小数点	不是 null	键	注
▶ PictureId	uuid	0	0	<input checked="" type="checkbox"/>	 1	
Size_Height	int4	32	0	<input checked="" type="checkbox"/>		
Size_Width	int4	32	0	<input checked="" type="checkbox"/>		
Location	text	0	0	<input checked="" type="checkbox"/>		
Order	int4	32	0	<input checked="" type="checkbox"/>		
ParentId_Value	uuid	0	0	<input checked="" type="checkbox"/>		
ApartmentId	uuid	0	0	<input type="checkbox"/>		
Id_Value	uuid	0	0	<input type="checkbox"/>		

阻抗不匹配

好像一切错误都解决了，那让我们来调用一下创建租房信息的接口。

ApartmentCommandsApi

POST

/apartment

Parameters

Cancel

No parameters

Request body

application/json

```
{  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  "ownerId": "3fa85f64-5717-4562-b3fc-2c963f66afa6"}}
```

阻抗不匹配

发现运行的时候会报错，我们来看看错误信息。

```
e", a."Owner_Value", a."Rent_Amount", a."Rent_Currency_CurrencyCode", a."Rent_Currency_DecimalPlaces", a."Rent_Currency_InUse"
FROM "Apartments" AS a
WHERE a."ApartmentId" = @__get_Item_0
LIMIT 1
fail: Microsoft.EntityFrameworkCore.Database.Command[20102]
Failed executing DbCommand (23ms) [Parameters=[@p0='3fa85f64-5717-4562-b3fc-2c963f66afa6', @p1=NULL (Nullable = false), @p2='0', @p3='3fa85f64-5717-4562-b3fc-2c963f66afa6' (Nullable = true), @p4='3fa85f64-5717-4562-b3fc-2c963f66afa6', CommandType='Text', CommandTimeout='30']
INSERT INTO "Apartments" ("ApartmentId", "Remark", "State", "Id_Value", "Owner_Value")
VALUES (@p0, @p1, @p2, @p3, @p4);
fail: Microsoft.EntityFrameworkCore.Update[10000]
An exception occurred in the database while saving changes for context type 'RJRentalOfHousing.Infrastructure.ApartmentDbContext'.
Microsoft.EntityFrameworkCore.DbUpdateException: An error occurred while saving the entity changes. See the inner exception for details.
--> Npgsql.PostgresException (0x80004005): 23502: 在字段 "Areas_Value" 中空值违反了非空约束
```

这个错误提示得很明显，就是Areas不能为空，但是我们在创建的时候没有提供这个参数，我们看看要如何解决这个问题。

阻抗不匹配

为解决上面的问题，我们需要修改Apartment类。

```
2 个引用 | 0 项更改 | 0 名作者, 0 项更改
public Guid ApartmentId { get; internal set; }

4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Area? Areas { get; internal set; }

4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Address? Address { get; internal set; }

4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Rent? Rent { get; internal set; }

4 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Deposit? Deposit { get; internal set; }

3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public UserId? Owner { get; internal set; }

1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public string? Remark { get; internal set; }

2 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public UserId? ApprovedBy { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public ApartmentState State { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public List<Picture> Pictures { get; internal set; }
```

这里可能会有点疑问，这些字段明明都不能为空。为什么还在字段里设为可空类型？因为可不可空也是业务中的一部分，而这部分应该完全由代码控制而不是由基础设施来控制。

阻抗不匹配

修改完成后再次启动项目和调用创建接口，看看效果。

```
warn: Microsoft.EntityFrameworkCore.Update[10001]
      The same entity is being tracked as different entity types 'Apartment.Rent#Price' and 'Apartment.Deposit#Price' with defining navigations. If a property value changes, it will result in two store changes, which might not be the desired outcome.
warn: Microsoft.EntityFrameworkCore.Update[10001]
      The same entity is being tracked as different entity types 'Apartment.Rent#Price' and 'Apartment.Deposit#Price' with defining navigations. If a property value changes, it will result in two store changes, which might not be the desired outcome.
fail: Microsoft.EntityFrameworkCore.Update[10000]
      An exception occurred in the database while saving changes for context type 'RJRentalOfHousing.Infrastructure.ApartmentDbContext'.
      System.InvalidOperationException: Cannot save instance of 'Apartment.Rent#Price' because it is an owned entity without any reference to its owner. Owned entities can only be saved as part of an aggregate also including the owner entity.
```

这个意思是说，有两个字段引用了同一个对象，当一个字段修改的时候另一个字段也会受影响。为解决这个问题，我们把Money类拆分，分为Rent和Deposit。

阻抗不匹配

在Domain项目中新建一个Rent类。

```
namespace RJRentalOfHousing.Domain
{
    8 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public record Rent : Money
    {
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected Rent() { }

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        private Rent(decimal amount, string currencyCode, ICurrencyLookup currencyLookup) : base(amount, currencyCode, currencyLookup)
        {
            if (amount < 0)
                throw new ArgumentException("租金不能为负数", nameof(amount));
        }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        internal Rent(decimal amount, string currencyCode) : base(amount, new CurrencyDetails { CurrencyCode = currencyCode }) { }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public static Rent FromDecimal(decimal amount, string currency, ICurrencyLookup currencyLookup) => new Rent(amount, currency, currencyLookup);

        public static Rent NoRent = new Rent();
    }
}
```

阻抗不匹配

在Domain项目中新建一个Deposit类。

```
using Domain
RJRentalOfHousing.Domain.Deposit
NoDeposit

namespace RJRentalOfHousing.Domain
{
    8 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public record Deposit : Money
    {
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected Deposit() {}

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        private Deposit(decimal amount, string currencyCode, ICurrencyLookup currencyLookup) : base(amount, currencyCode, currencyLookup)
        {
            if (amount < 0)
                throw new ArgumentException("押金不能为负数", nameof(amount));
        }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        internal Deposit(decimal amount, string currencyCode) : base(amount, new CurrencyDetails { CurrencyCode = currencyCode }) {}

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public static Deposit FromDecimal(decimal amount, string currency, ICurrencyLookup currencyLookup) => new Deposit(amount, currency, currencyLookup);

        public static Deposit NoDeposit = new Deposit();
    }
}
```

阻抗不匹配

修改Apartment中的字段类型。

```
public Guid ApartmentId { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Area Areas { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Address Address { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Rent Rent { get; internal set; }

5 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public Deposit Deposit { get; internal set; }

3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public UserId Owner { get; internal set; }

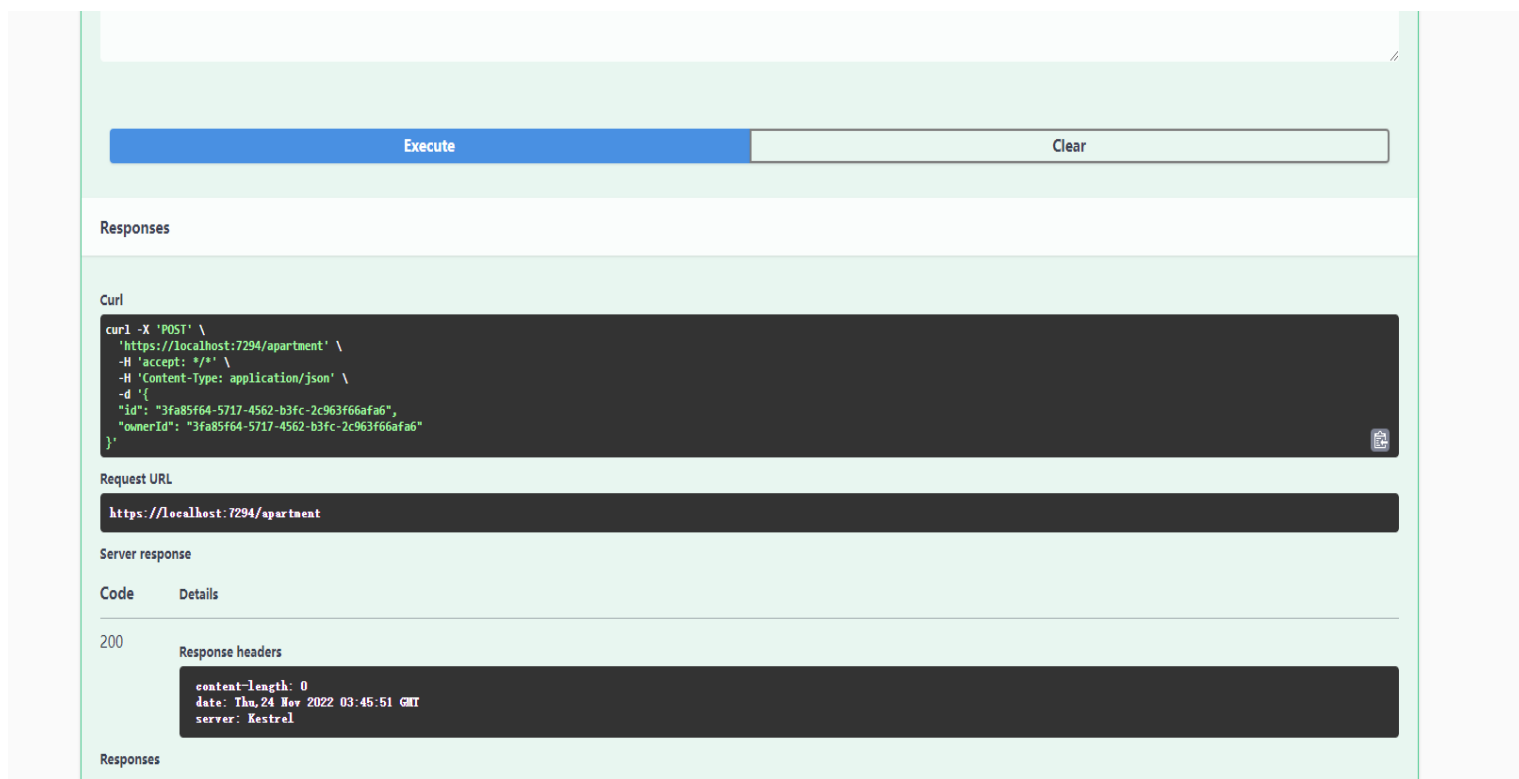
1 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public string Remark { get; internal set; }

3 个引用 | HCJ, 1 天前 | 1 名作者, 1 项更改
public UserId ApprovedBy { get; internal set; }
```

```
protected override void When(object @event)
{
    switch(@event)
    {
        case Events.ApartmentCreated e:
            Id = new ApartmentId(e.Id);
            Owner = new UserId(e.OwnerId);
            State = ApartmentState.Created;
            Areas = Area.NoArea;
            Address = Address.NoAddress;
            Rent = Rent.NoRent;
            Deposit = Deposit.NoDeposit;
            ApprovedBy = UserId.NoUser;
            ApartmentId = e.Id;
            break;
        case Events.ApartmentAreaUpdated e:
            Areas = new Area(e.Areas);
            break;
        case Events.ApartmentAddressUpdated e:
            Address = new Address(e.Address);
            break;
        case Events.ApartmentRentUpdated e:
            Rent = new Rent(e.Rent, e.CurrencyCode);
            break;
        case Events.ApartmentDepositUpdated e:
            Deposit = new Deposit(e.Deposit, e.CurrencyCode);
            break;
        case Events.ApartmentRemarkUpdated e:
            Remark = e.Remark;
            break;
        case Events.ApartmentSentForReview e:
    }
```

阻抗不匹配

打开数据库，把我们刚才生成的数据库删除，然后重新运行项目和调用接口，看看效果。



总算是成功了，数据库里面也应该有数据，接下来看看其他接口是不是能正常调用。

阻抗不匹配

调用设置面积的接口。

PUT

/apartment/area

Parameters

No parameters

Request body

application/json

```
{  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  "areas": 100}
```

Code

Details

200

Response headers

```
content-length: 0
date: Thu, 24 Nov 2022 03:47:11 GMT
server: Kestrel
```

Responses

Code	Description	Links
200	Success	No links

阻抗不匹配

看看控制台的输出和数据库里的数据。

```
VALUES (@p0, @p1, @p2, @p3, @p4);  
info: Microsoft.EntityFrameworkCore.Database.Command[20101]  
      Executed DbCommand (18ms) [Parameters=[@p1='3fa85f64-5717-4562-b3fc-2c963f66afa6', @p0='100' (Nullable = true)], CommandType='Text', CommandTimeout='30']  
      UPDATE "Apartments" SET "Areas_Value" = @p0  
      WHERE "ApartmentId" = @p1;
```

开始事务	文本	筛选	排序	导入	导出
ApartmentId	Areas_Value	Address_Value	Rent_		
▶ 3fa85f64-5717-45	100	(Null)			

其他接口自行实验，应该也是可以的。

结束语

本章讲解了仓储、工作单元和如何用Entity Framework Core来实现仓储和工作单元，并且讲解如何解决在使用ORM的时候遇到的阻抗不匹配的问题。下一章也是最后一章，将会讲解CQRS和查询端。