从零开始实现DDD应用

第二章 WebAPI

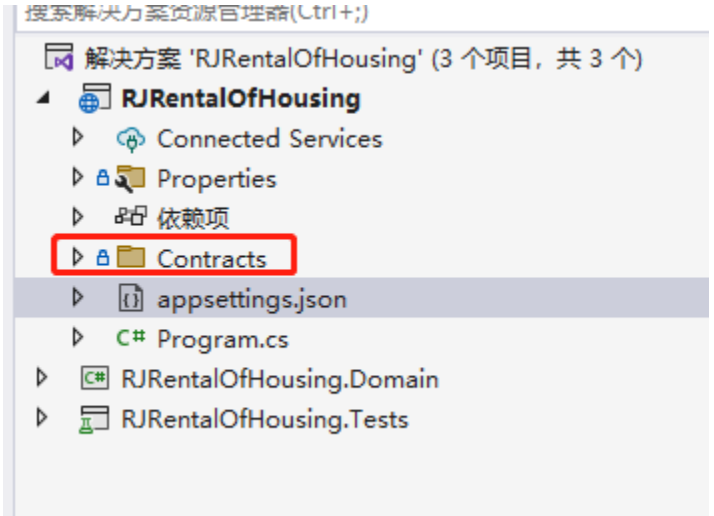
作者：贺传珺

合约

在开始之前先把我们的API项目清理一下，删除API项目中的Controller文件夹和WeatherForecast。

合约

合约（Contract）就是我们所熟知的DTO对象，也是一个POCO（Plain-old C# objects）对象。因此它不会包含任何业务上的逻辑，仅包含字段。能直接被序列化和反序列化。我们在API项目中新建一个文件夹来保存这些文件，命名为Contracts。



合约

我们公开出去的API有很多时候不能做中断变更（Breaking changes）也叫作破坏性更新。因为API的改变很多时候不能及时通知调用者。所以合约中必须包括有版本的信息。接下来我们来看看如何在代码中实现它。在Contracts文件夹中新建一个类，命名为Apartments。

```
namespace RJRentalOfHousing.Contracts
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public static class Apartments
    {
        0 个引用|0 项更改|0 名作者，0 项更改
        public static class V1
        {
            0 个引用|0 项更改|0 名作者，0 项更改
            public class Create
            {
                0 个引用|0 项更改|0 名作者，0 项更改
                public Guid Id { get; set; }
                0 个引用|0 项更改|0 名作者，0 项更改
                public Guid OwnerId { get; set; }
            }
        }
    }
}
```

合约

在代码中我们看到合约被定义在两个Class包裹之中，最外层的Class表示的是所属的业务，第二层的Class表示的是版本号，最里面的才是合约。可能到目前为止会觉得这个设计很多余。这很正常，带着问题接着往下看。

HTTP端点

现在要新建第一个控制器，在API项目中新建一个API控制器，命名为ApartmentCommandsApi，并且继承Controller和增加一个Post接口来接收命令。

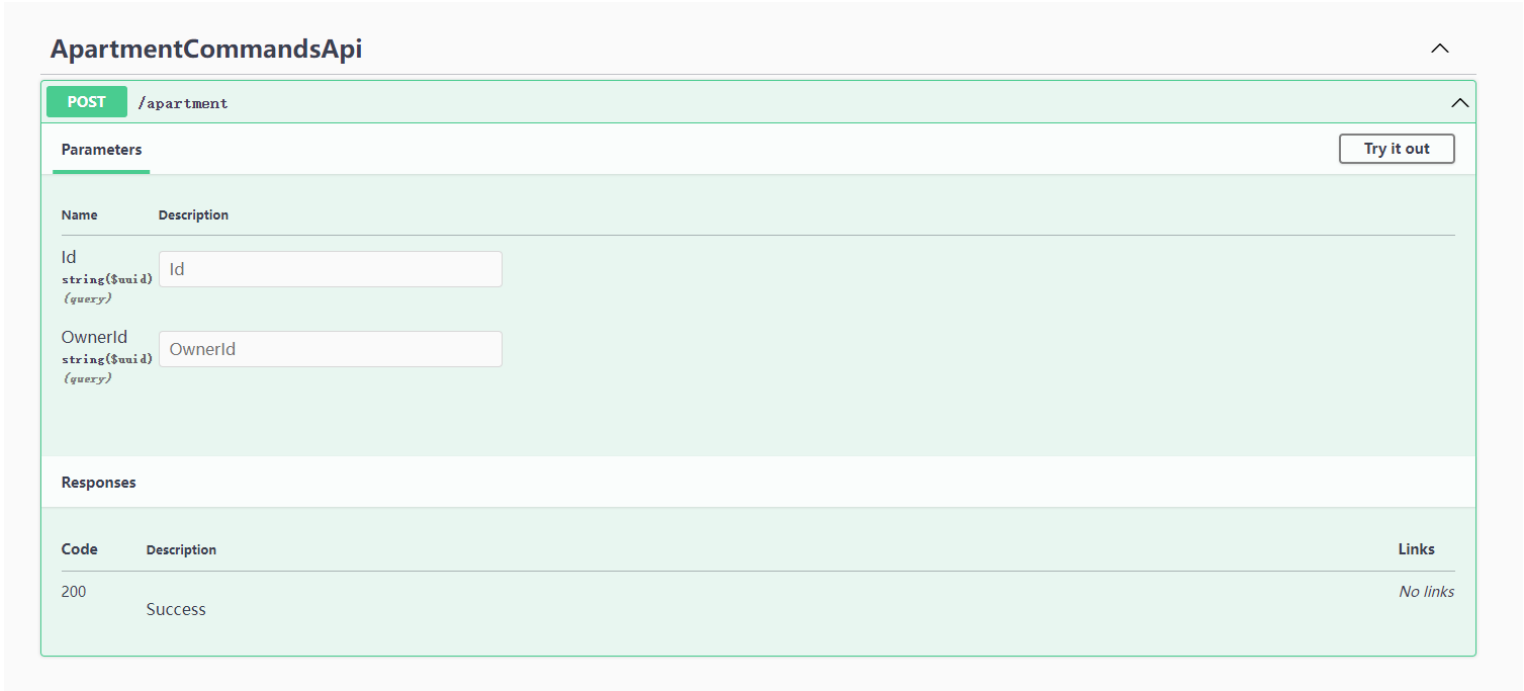
```
using Microsoft.AspNetCore.Mvc;

namespace RJRentalOfHousing
{
    [Route("/apartment")]
    0 个引用|0 项更改|0 名作者，0 项更改
    public class ApartmentCommandsApi : Controller
    {
        [HttpPost]
        0 个引用|0 项更改|0 名作者，0 项更改
        public async Task<IActionResult> Post(Contracts.Apartments.V1.Create request)
        {
            return Ok();
        }
    }
}
```

注意观察这个接口的参数，就是我们在上面定义的类型。如果我们有新的版本的接口需要上线且需要更改参数。我们只需要在合约里新建一个版本号的合约而不会影响现在运行的版本。

运行项目

按下F5或者Ctrl+F5运行项目，一切正常的情况下我们会有如下页面。这时候无论输入什么值这个接口智只会返回200，接下来我们要让这些接口和领域之间产生互动。



应用服务

应用服务和领域服务不同，它不包含任何业务代码。可以把它看做是介于边缘和领域之间的管道。首先新建一个接口，命名为IApplicationService。由于我们目前只有一种边缘（HTTP），所以我不会把应用服务和API放在不同的项目里。

JtHousing

```
namespace RJRentalOfHousing
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public interface IApplicationService
    {
        0 个引用|0 项更改|0 名作者，0 项更改
        Task Handle(object command);
    }
}
```

应用服务

新建一个类，命名为ApartmentsApplicationService，并且继承IApplicationService。然后把我们刚才写的合约加进去。

```
using static RJRentalOfHousing.Contracts.Apartments;

namespace RJRentalOfHousing
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public class ApartmentsApplicationService : IApplicationService
    {
        1 个引用|0 项更改|0 名作者，0 项更改
        public async Task Handle(object command)
        {
            switch(command)
            {
                case V1.Create cmd:
                    break;
                default:
                    throw new InvalidOperationException($"未知的命令类型:{command.GetType().FullName}");
                    break;
            }
        }
    }
}
```

应用服务

现在回头去看看我们的实体，看看有那些需要实现的命令，并把它们加入到合约中。修改Contracts文件夹中的Apartments类。

```
public class SetArea
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public decimal Areas { get; set; }
}

0 个引用|0 项更改|0 名作者，0 项更改
public class SetAddress
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public string Address { get; set; }
}

0 个引用|0 项更改|0 名作者，0 项更改
public class SetRent
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public decimal Rent { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public string CurrencyCode { get; set; }
}
```

```
public class SetDeposit
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public decimal Deposit { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public string CurrencyCode { get; set; }
}

0 个引用|0 项更改|0 名作者，0 项更改
public class SetRemark
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
    0 个引用|0 项更改|0 名作者，0 项更改
    public string Remark { get; set; }
}

0 个引用|0 项更改|0 名作者，0 项更改
public class SentForReview
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public Guid Id { get; set; }
}
```

应用服务

现在修改HTTP端点，根据刚才所新增的合约新增一些接口。

```
using Microsoft.AspNetCore.Mvc;
using static RJRentalOfHousing.Contracts.Apartments;

namespace RJRentalOfHousing
{
    [Route("/apartment")]
    1 个引用|0 项更改|0 名作者，0 项更改
    public class ApartmentCommandsApi : Controller
    {
        private readonly ApartmentsApplicationService _applicationService;

        0 个引用|0 项更改|0 名作者，0 项更改
        public ApartmentCommandsApi(ApartmentsApplicationService applicationService) => _applicationService = applicationService;

        [HttpPost]
        0 个引用|0 项更改|0 名作者，0 项更改
        public async Task<IActionResult> Post(V1.Create request)
        {
            await _applicationService.Handle(request);
            return Ok();
        }
```

应用服务

```
[Route("area")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SetArea request)
{
    await _applicationService.Handle(request);
    return Ok();
}

[Route("address")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SetAddress request)
{
    await _applicationService.Handle(request);
    return Ok();
}

[Route("rent")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SetRent request)
{
    await _applicationService.Handle(request);
    return Ok();
}
```

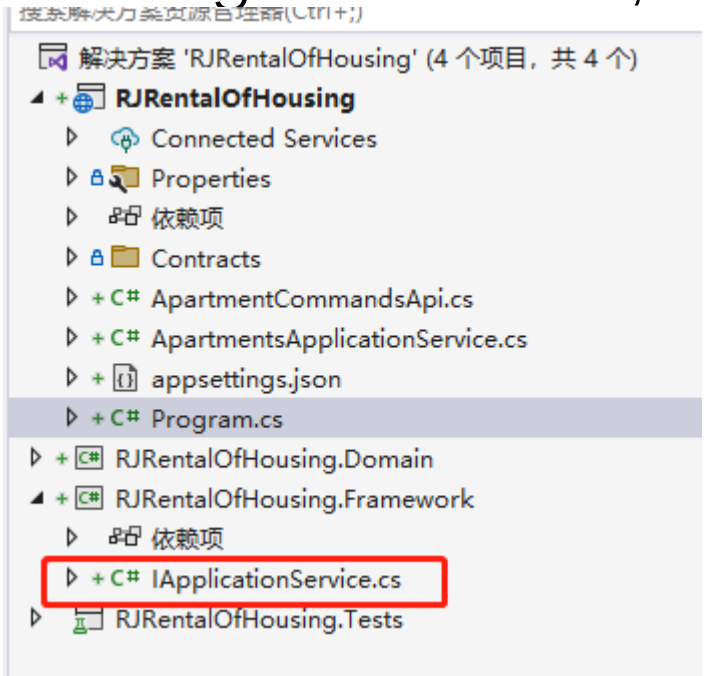
```
[Route("deposit")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SetDeposit request)
{
    await _applicationService.Handle(request);
    return Ok();
}

[Route("remark")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SetRemark request)
{
    await _applicationService.Handle(request);
    return Ok();
}

[Route("publish")]
[HttpPut]
0 个引用|0 项更改|0 名作者，0 项更改
public async Task<IActionResult> Put(V1.SentForReview request)
{
    await _applicationService.Handle(request);
    return Ok();
}
```

应用服务

现在我们需要一个接口来实现保存数据的功能，但是这些代码是属于框架的功能，因此我们需要新建一个类库项目，命名为RJRentalOfHousing.Framework，并且把IApplicationService接口给移过去。



应用服务

让API项目添加Framework项目的引用，然后修改ApartmentsApplicationService类。

lousing　　RJRentalOfHousing.ApartmentsApplicationService

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

namespace RJRentalOfHousing
{
    2 个引用|0 项更改|0 名作者，0 项更改
    public class ApartmentsApplicationService : IApplicationService
    {
        8 个引用|0 项更改|0 名作者，0 项更改
        public async Task Handle(object command)
        {
            switch(command)
            {
                case V1.Create cmd:
                    break;
                default:
                    throw new InvalidOperationException($"未知的命令类型:{command.GetType().FullName}");
                    break;
            }
        }
    }
}
```

领域服务

在Framework项目中新建一个接口，命名为IEntityStore。

```
namespace RJRentalOfHousing.Framework
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public interface IEntityStore
    {
        0 个引用|0 项更改|0 名作者，0 项更改
        Task<T> Load<T>(string entityId);

        0 个引用|0 项更改|0 名作者，0 项更改
        Task Save<T>(T entity);

        0 个引用|0 项更改|0 名作者，0 项更改
        Task<bool> Exists<T>(string entityId);
    }
}
```

现在还不需要去实现这个接口，我们在第四章才会持久化这些数据到数据库里。

领域服务

现在万事具备，开始在领域服务中添加一些代码，修改ApartmentsApplicationService类。

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

namespace RJRentalOfHousing
{
    3 个引用|0 项更改|0 名作者，0 项更改
    public class ApartmentsApplicationService : IApplicationService
    {
        private readonly IEntityStore _entityStore;
        private readonly ICurrencyLookup _currencyLookup;

        0 个引用|0 项更改|0 名作者，0 项更改
        public ApartmentsApplicationService(IEntityStore entityStore,ICurrencyLookup currencyLookup)
        {
            _entityStore = entityStore;
            _currencyLookup = currencyLookup;
        }
```

领域服务

```
1 个引用 | 0 项更改 | 0 名作者，0 项更改
private async Task HandldCreate(V1.Create cmd)
{
    if (await _entityStore.Exists(cmd.Id.ToString()))
        throw new InvalidOperationException($"{cmd.Id}已存在");
    var apartment = new Apartment(new ApartmentId(cmd.Id), new UserId(cmd.OwnerId));
    await _entityStore.Save(apartment);
}

6 个引用 | 0 项更改 | 0 名作者，0 项更改
private async Task HandleUpdate(Guid ApartmentId,Action<Apartment> operation)
{
    var apartment = await _entityStore.Load<Apartment>(ApartmentId.ToString());
    if (apartment == null)
        throw new InvalidOperationException($"{ApartmentId}不存在");
    operation(apartment);
    await _entityStore.Save(apartment);
}
```

领域服务

```
public async Task Handle(object command)
{
    switch(command)
    {
        case V1.Create cmd:
            await HandldCreate(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}");
    }
}
```

结束语

本章的内容较为简单。介绍了WebAPI和应用服务，WebAPI作为边缘应该很“薄”，不应该含有任何业务逻辑。应用服务只是作为边缘和领域之间的管道。下一章将是整个教程的重点：聚合根（Aggregate）。