

从零开始实现DDD应用

第三章 Aggregate

作者：贺传珺

聚合

聚合是指一组一致的数据，它们在特定的业务下不能脱离聚合根单独存在。它们需要一起更新状态，一起保存，意味着一组聚合需要有相同至少需要类似的更新频率。我们以生活中常见的事物举例：个人电脑（PC）。

聚合

一台PC都由什么零件组成？CPU、主板、电源、内存、硬盘、显卡、键盘、鼠标、显示器、风扇等。它又有哪些属性？品牌、型号、生产日期、各种部件的型号、CPU主频、内存颗粒、内存频率、电源功率、主板芯片组、GPU型号、显存等，除此之外还有各种状态：CPU频率、CPU温度、GPU温度、硬盘转速、内存占用率、显存占用率、CPU负载、GPU负载等。如果我们对电脑进行建模，按以往的经验这些字段是要被设计到一张表里。这似乎很正常，这些都是电脑自身的属性。

聚合

我们刚才设计了一个“完美”的PC表。这个表包含PC的方方面面，但是这真的科学吗？在我们使用的场景下，我们是不关心电脑的品牌、型号、生产日期的，只关心它的运行状态，其他的数据在这个场景下毫无意义。在保修的时候我们也只关心品牌、型号、生产日期而不关心它运行时的状态。这意味着我们刚才设计的PC表的更新的频率是不一致的。DDD不需要一个大而全的对象，但是需要你找到一组一致的对象。接下来我们在代码里实现一个聚合。

聚合

现在开始实现聚合根的基类，在Framework项目中新建一个AggregateRoot类。

```
namespace RJRentalOfHousing.Framework
{
    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public abstract class AggregateRoot<TId>
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TId Id { get; protected set; }

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected abstract void When(object @event);

        private readonly List<object> _changes;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected AggregateRoot() => _changes = new List<object>();

        7 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected void Apply(object @event)
        {
            When(@event);
            EnsureValidState();
            _changes.Add(@event);
        }
    }
}
```

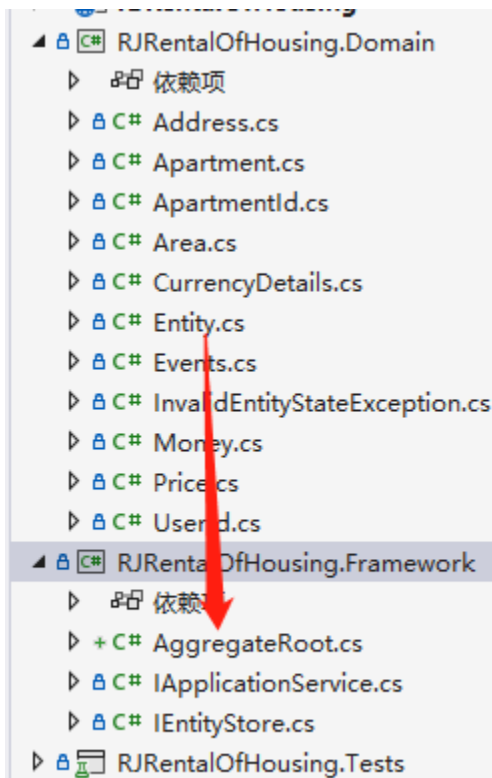
```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public IEnumerable<object> GetChanges() => _changes.AsEnumerable();

0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public void ClearChanges() => _changes.Clear();

1 个引用 | 0 项更改 | 0 名作者, 0 项更改
protected abstract void EnsureValidState();
```

聚合

把Entity类从Domain项目中移动到Framework项目中，并且修改命名空间。



```
namespace RJRentalOfHousing.Framework;

1 个引用 | 0 项更改 | 0 名作者, 0 项更改
public abstract class Entity
{
    private readonly List<object> _events;

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    protected Entity() => _events = new List<object>();

    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
```

聚合

修改Domain中的Apartment类，让它继承AggregateRoot。

```
using RJRentalOfHousing.Framework;

namespace RJRentalOfHousing.Domain
{
    7 个引用 | HCJ, 4 天前 | 1 名作者, 1 项更改
    public class Apartment: AggregateRoot<ApartmentId>
    {
        2 个引用 | HCJ, 4 天前 | 1 名作者, 1 项更改
        public Apartment(ApartmentId id, UserId ownerId)
            => Apply(new Events.ApartmentCreated
            {
                Id = id,
                OwnerId = ownerId
            });

        //public ApartmentId Id { get; internal set; }

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

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

聚合

修改Entity类。

```
namespace RJRentalOfHousing.Framework
{
    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public abstract class Entity<TId>
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TId Id { get; protected set; }

        private readonly List<object> _events;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected Entity() => _events = new List<object>();

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected void Apply(object @event)
        {
            When(@event);
            _events.Add(@event);
        }

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected abstract void When(object @event);

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public IEnumerable<object> GetChanges => _events;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public void ClearChanges() => _events.Clear();
    }
}
```


聚合

在第一章我们提到，出租的房子里还得有图片，现在我们把图片加上，并且应用到聚合根里。在Domain项目中新建Picture类。

```
using RJRentalOfHousing.Framework;

namespace RJRentalOfHousing.Domain
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class Picture : Entity<PictureId>
    {
        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void When(object @event)
        {
            throw new NotImplementedException();
        }
    }

    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public record PictureId
    {
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Guid Value { get; internal set; }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public PictureId(Guid value) => Value = value;
    }
}
```

聚合

让我们想象一下一张图片有什么属性？有宽、高、地址、排序等。这里为简化只实现这4个字段，并且默认排序最靠前的为默认图片。现在有个隐藏的业务信息：图片的宽高都不能小于零，因此我们需要为图片的尺寸新建一个值对象。在Domain中新建一个类，命名为PictureSize。

聚合

```
namespace RJRentalOfHousing.Domain
{
    7 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public record PictureSize
    {
        4 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public int Height { get; internal set; }
        4 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public int Width { get; internal set; }

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public PictureSize(int width, int height)
        {
            if (width <= 0)
                throw new ArgumentOutOfRangeException(nameof(width), "图片宽度必须大于0");
            if (height <= 0)
                throw new ArgumentOutOfRangeException(nameof(height), "图片长度必须大于0");
            Width = width;
            Height = height;
        }

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        internal PictureSize() { }
    }
}
```

聚合

修改Picture类。

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public class Picture : Entity<PictureId>
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public PictureSize Size { get; internal set; }
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public Uri Location { get; internal set; }
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Order { get; internal set; }

    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    protected override void When(object @event)
    {
    }
}
```

聚合

聚合内的所有动作都是在聚合根里完成的，因此我们要在Events类中新建一个事件。

2 个引用 | HCJ, 5 天前 | 1 名作者, 1 项更改

```
public class ApartmentSentForReview
```

```
{
```

1 个引用 | HCJ, 5 天前 | 1 名作者, 1 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 class PictureAddedToApartment
```

```
{
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

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

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
    public Guid PictureId { get; set; }
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
    public string Url { get; set; }
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
    public int Height { get; set; }
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
    public int Width { get; set; }
```

```
}
```

```
}
```

聚合

在Apartment中新建一个方法，用来新增图片。

8 个引用 | HCJ, 5 天前 | 1 名作者, 1 项更改

```
public void RequestToPublish()
=> Apply(new Events.ApartmentSentForReview
{
    Id = Id,
});
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
public void AddPicture(Uri pictureUri, PictureSize size)
=> Apply(new Events.PictureAddedToAApartment
{
    PictureId = new Guid(),
    ApartmentId = Id,
    Url = pictureUri.ToString(),
    Height = size.Height,
    Width = size.Width
});
```

2 个引用 | HCJ, 5 天前 | 1 名作者, 1 项更改

```
protected override void EnsureValidState()
{
    var valid =
        Id != null &&
```

聚合

修改Apartment类，新建一个字段保存图片，并且在构造函数中赋值防止出现空引用。

```
7 个引用 | HCJ, 5 天前 | 1 名作者, 1 项更改
public class Apartment: AggregateRoot<ApartmentId>
{
    2 个引用 | HCJ, 5 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
        });
    }
    //public ApartmentId Id { get; internal set; }

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

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

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

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

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

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

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

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

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

1 个引用 | 0 项更改 | 0 名作者, 0 项更改
public List<Picture> Pictures { get; internal set; }
}
```

聚合

在When方法中，新增一个模式匹配，应用刚才新建的事件。

```
        break;
    case Events.ApartmentDepositUpdated e:
        Deposit = new Price(e.Deposit, e.CurrencyCode);
        break;
    case Events.ApartmentRemarkUpdated e:
        Remark = e.Remark;
        break;
    case Events.ApartmentSentForReview _:
        State = ApartmentState.PendingReview;
        break;
    case Events.PictureAddedToApartment e:
        var newPicture = new Picture
        {
            Size = new PictureSize(e.Width, e.Height),
            Location = new Uri(e.Url),
            Order = Pictures.Max(x => x.Order) + 1
        };
        Pictures.Add(newPicture);
        break;
```


聚合

前面那些看上去很完美，但是并非如此。首先，Apartment执行了Picture本身的逻辑。而且我们实体往往不止有一个事件，实体需要执行自己的事件，但是聚合根也得感知这些事件，这时候应该如何处理？首先在Framework项目中新建一个InternalEventHandler接口，这个接口用来在聚合根和实体之间传递事件。

```
using Framework
namespace RJRentalOfHousing.Framework
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public interface IInternalEventHandler
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        void Handler(object @event);
    }
}
```

聚合

接下来对Entity和AggregateRoot类进行一些修改。

```
namespace RJRentalOfHousing.Framework
{
    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public abstract class Entity<TId>: IInternalEventHandler
    {
        private readonly Action<object> _applier;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected Entity(Action<object> applier) => _applier = applier;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TId Id { get; protected set; }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected void Apply(object @event)
        {
            When(@event);
            _applier(@event);
        }

        3 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected abstract void When(object @event);

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        void IInternalEventHandler.Handler(object @event) => When(@event);
    }
}
```

聚合

```
using Framework
namespace RJRentalOfHousing.Framework
{
    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public abstract class AggregateRoot<TId> : IInternalEventHandler
    {
        9 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TId Id { get; protected set; }

        3 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected abstract void When(object @event);

        private readonly List<object> _changes;

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected AggregateRoot() => _changes = new List<object>();

        8 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected void Apply(object @event)
        {
            When(@event);
            EnsureValidState();
            _changes.Add(@event);
        }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public IEnumerable<object> GetChanges() => _changes.AsEnumerable();
    }
}
```

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public IEnumerable<object> GetChanges() => _changes.AsEnumerable();

0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public void ClearChanges() => _changes.Clear();

2 个引用 | 0 项更改 | 0 名作者, 0 项更改
protected abstract void EnsureValidState();

0 个引用 | 0 项更改 | 0 名作者, 0 项更改
protected void ApplyToEntity(IInternalEventHandler entity, object @event)
    => entity?.Handler(@event);

2 个引用 | 0 项更改 | 0 名作者, 0 项更改
void IInternalEventHandler.Handler(object @event) => When(@event);
}
```

聚合

重构Picture，把属于它自己的事件移到它自身。

```
using RJRentalOfHousing.Framework;

namespace RJRentalOfHousing.Domain
{
    5 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class Picture : Entity<PictureId>
    {
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Picture(Action<object> applier) : base(applier) {}

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public PictureSize Size { get; internal set; }
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Uri Location { get; internal set; }
        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public int Order { get; internal set; }

        3 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void When(object @event)
        {
            switch(@event)
            {
                case Events.PictureAddedToAApartment e:
                    Id = new PictureId(e.PictureId);
                    Location = new Uri(e.Url);
                    Size = new PictureSize { Height = e.Height, Width = e.Width };
                    Order = e.Order;
                    break;
            }
        }
    }
}
```

```
3 个引用 | 0 项更改 | 0 名作者, 0 项更改
public class PictureAddedToAApartment
{
    1 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public Guid ApartmentId { get; set; }
    2 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public Guid PictureId { get; set; }
    3 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public string Url { get; set; }
    3 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Height { get; set; }
    3 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Width { get; set; }
    1 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Order { get; set; }
}
```

聚合

修改Apartment类，把事件通过刚才的接口从聚合根传播到实体。

0 个引用 | 0 项更改 | 0 名作者，0 项更改

```
public void AddPicture(Uri pictureUri, PictureSize size)
=> Apply(new Events.PictureAddedToApartment
{
    PictureId = new Guid(),
    ApartmentId = Id,
    Url = pictureUri.ToString(),
    Height = size.Height,
    Width = size.Width,
    Order = Pictures.Max(x => x.Order)
});
```

```
break;
case Events.PictureAddedToApartment e:
    var picture = new Picture(Apply);
    ApplyToEntity(picture, e);
    Pictures.Add(picture);
    break;
```

聚合

现在我们在实体里添加一个行为：改变图片的尺寸。要求第一章作为封面的图片宽必须大于等于800像素，高必须大于等于600像素。首先在Events中新增一个事件，然后在聚合根（Apartment）中新建一个查询图片的方法。

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public class ApartmentPictureResized
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public Guid PictureId { get; set; }
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Width { get; set; }
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public int Height { get; set; }
}
```

```
...
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
public void AddPicture(Uri pictureUri, PictureSize size)
=> Apply(new Events.PictureAddedToAApartment
{
    PictureId = new Guid(),
    ApartmentId = Id,
    Url = pictureUri.ToString(),
    Height = size.Height,
    Width = size.Width,
    Order = Pictures.Max(x => x.Order)
});
```

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
private Picture FindFirstPicture(PictureId id) => Pictures.FirstOrDefault(x => x.Id == id);
```

```
0 个引用 | 0 项更改 | 0 名作者, 0 项更改
```

聚合

然后，在Picture中添加相应的方法和事件。

```
using RJRentalOfHousing.Framework;

namespace RJRentalOfHousing.Domain
{
    5 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class Picture : Entity<PictureId>
    {
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Picture(Action<object> applier) : base(applier) { }

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public PictureSize Size { get; internal set; }
        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public Uri Location { get; internal set; }
        2 个引用 | 0 项更改 | 0 名作者, 0 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 void Resize(PictureSize newSize)
            => Apply(new Events.ApartmentPictureResized
            {
                PictureId = Id.Value,
                Height = newSize.Height,
                Width = newSize.Width,
            });
    }
}
```

```
});

3 个引用 | 0 项更改 | 0 名作者, 0 项更改
protected override void When(object @event)
{
    switch(@event)
    {
        case Events.PictureAddedToApartment e:
            Id = new PictureId(e.PictureId);
            Location = new Uri(e.Url);
            Size = new PictureSize{ Height = e.Height, Width = e.Width };
            Order = e.Order;
            break;
        case Events.ApartmentPictureResized e:
            Size = new PictureSize { Height = e.Height, Width = e.Width };
            break;
    }
}
```

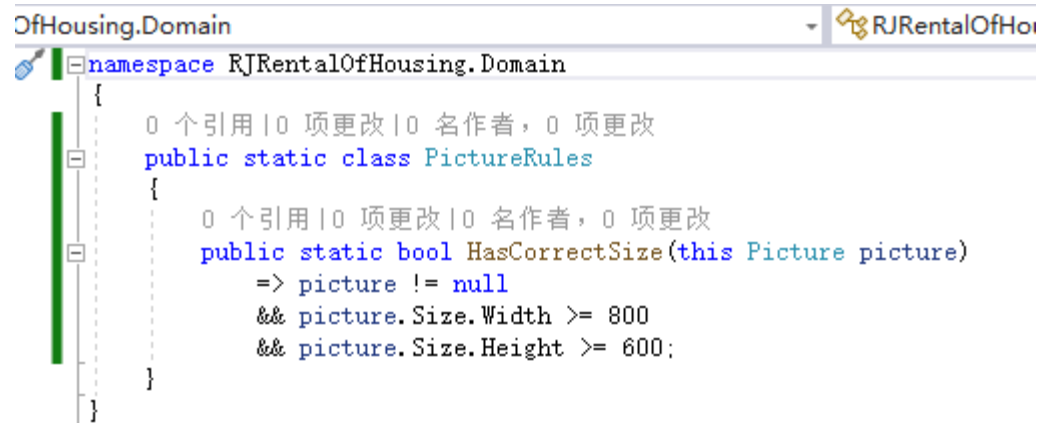
聚合

在聚合根中添加方法来引发实体中的事件。

```
//  
1 个引用 | 0 项更改 | 0 名作者， 0 项更改  
private Picture FindFirstPicture(PictureId id) => Pictures.FirstOrDefault(x => x.Id == id);  
  
0 个引用 | 0 项更改 | 0 名作者， 0 项更改  
public void ResizePicture(PictureId pictureId, PictureSize newSize)  
{  
    var picture = FindFirstPicture(pictureId);  
    if (picture == null)  
        throw new InvalidOperationException("不能修改不存在的图片的尺寸");  
    picture.Resize(newSize);  
}
```


聚合

刚才似乎漏了我们前面的限制，这个限制本来应该加在聚合根中的EnsureValidState方法里，但是如果这样做每个状态的检查都必须复制一次代码，这样就违背了DRY原则，我们用扩展方法来处理这个问题。在Domain中新建一个PictureRules类。



```
OffHousing.Domain - RJRentalOfHousing.Domain
namespace RJRentalOfHousing.Domain
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public static class PictureRules
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public static bool HasCorrectSize(this Picture picture)
        => picture != null
        && picture.Size.Width >= 800
        && picture.Size.Height >= 600;
    }
}
```

聚合

在聚合根里添加一个寻找第一张图片的方法，并且加上业务限制。

1 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
private Picture FindFirstPicture(PictureId id) => Pictures.FirstOrDefault(x => x.Id == id);
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
private Picture FirstPicture => Pictures.OrderBy(x => x.Order).FirstOrDefault();
```

0 个引用 | 0 项更改 | 0 名作者, 0 项更改

```
public void ResizePicture(PictureId pictureId, PictureSize newSize)
```

```
{
```

```
    var picture = FindFirstPicture(pictureId);
```

```
    if (picture == null)
```

```
        throw new InvalidOperationException("不能修改不存在的图片的尺寸");
```

```
    picture.Resize(newSize);
```

```
}
```

聚合

```
protected override void EnsureValidState()
{
    var valid =
        Id != null &&
        Owner != null &&
        (
            State switch
            {
                ApartmentState.PendingReview =>
                    Areas != null &&
                    Address != null &&
                    Rent?.Amount > 0 &&
                    Deposit?.Amount > 0 &&
                    FirstPicture.HasCorrectSize(),
                ApartmentState.Renting =>
                    Areas != null &&
                    Address != null &&
                    Rent?.Amount > 0 &&
                    Deposit?.Amount > 0 &&
                    ApprovedBy != null &&
                    FirstPicture.HasCorrectSize(),
                _ => true
            }
        );
    if (!valid)
        throw new InvalidEntityStateException(this, $"实体提交状态 {State} 检查失败");
}
```

结束语

到现在为止我们已经介绍了实体、值对象、聚合根之间的关系。还介绍了领域服务、WebAPI、合约等。但是我们目前都只停留在建模的阶段。下一章将开始连接数据库，把我们的聚合根持久化到数据库中。