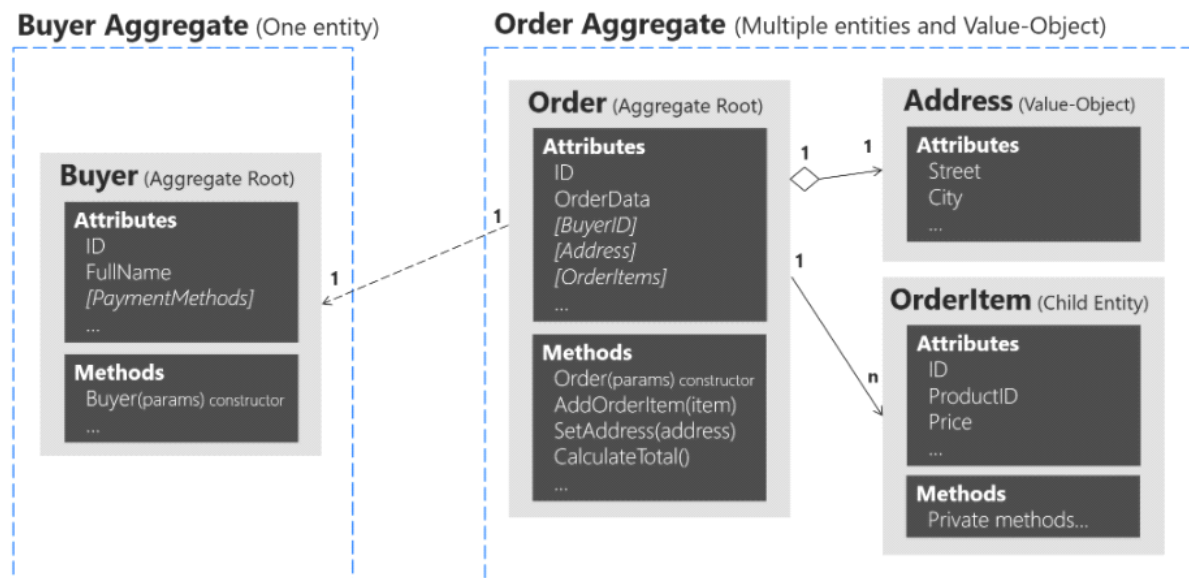


第43期-理解实体与聚合根

2020年11月16日 15:32

实体+聚合根+值对象



实体定义

```
public class Book : Entity<Guid>
{
    public string Name { get; set; }

    public float Price { get; set; }
}
```

如果你不想继承基类 Entity<TKey> 类，也可以直接实现 IEntity<TKey> 接口。

```
public class Book : IEntity<Guid>
{
    public string Name { get; set; }
    public float Price { get; set; }
    protected Book()
    {
    }
    public Book(Guid id): base(id)
    {
    }
}
```

```
Book book = new Book(IGuidGenerator.Create());
```

具有复合主键的实体

```
public class UserRole : Entity
{
    public Guid UserId { get; set; }

    public Guid RoleId { get; set; }

    public DateTime CreationTime { get; set; }

    public UserRole()
    {
    }

    public override object[] GetKeys()
    {
        return new object[] { UserId, RoleId };
    }
}
```

除此之外还是需要在 EF Core 模型映射上设置复合主键。

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<UserRole>().HasKey(t => new { t.UserId, t.RoleId });
}
```

复合主键实体不可以使用 IRepository<TEntity, TKey>, 只能使用 IRepository<TEntity> 仓储。

聚合根

eShopOnContainers AggregatesModel

```
public class Order : AggregateRoot<Guid>
{
    public virtual string ReferenceNo { get; protected set; }

    public virtual int TotalItemCount { get; protected set; }

    public virtual DateTime CreationTime { get; protected set; }

    public virtual List<OrderLine> OrderLines { get; protected set; }

    protected Order()
    {
    }
}
```

```

public Order(Guid id, string referenceNo)
{
    Check.NotNull(referenceNo, nameof(referenceNo));

    Id = id;
    ReferenceNo = referenceNo;

    OrderLines = new List<OrderLine>();
}

public void AddProduct(Guid productId, int count)
{
    if (count <= 0)
    {
        throw new ArgumentException(
            "You can not add zero or negative count of
products!",
            nameof(count)
        );
    }

    var existingLine = OrderLines.FirstOrDefault(ol => ol.ProductId ==
productId);

    if (existingLine == null)
    {
        OrderLines.Add(new OrderLine(this.Id, productId, count));
    }
    else
    {
        existingLine.ChangeCount(existingLine.Count + count);
    }

    TotalItemCount += count;
}

}

public class OrderLine : Entity
{
    public virtual Guid OrderId { get; protected set; }

    public virtual Guid ProductId { get; protected set; }

    public virtual int Count { get; protected set; }

    protected OrderLine()
    {
    }
}

```

```

internal OrderLine(Guid orderId, Guid productId, int count)
{
    OrderId = orderId;
    ProductId = productId;
    Count = count;
}

internal void ChangeCount(int newCount)
{
    Count = newCount;
}

public override object[] GetKeys()
{
    return new Object[] {OrderId, ProductId};
}
}

```

聚合根继承 `AggregateRoot<TKey>` 类，也可以直接实现 `IAggregateRoot<TKey>` 接口。

带有组合键的聚合根

不常见，也不推荐，如果非要这么做，继承 `AggregateRoot` 基类，实现组合键。

关于 `BasicAggregateRoot` 类

`AggregateRoot` 类实现了 `IHasExtraProperties` 和 `IHasConcurrencyStamp` 接口。如果你不需要这些功能，可以继承 `BasicAggregateRoot<TKey>`(或 `BasicAggregateRoot` 类)

基类和接口的审计属性

像 `CreationTime`, `CreatorId`, `LastModificationTime` 很常见，标准化和自动化。

- `IHasCreationTime` 定义了以下属性：
 - `CreationTime`
- `IMayHaveCreator` 定义了以下属性：
 - `CreatorId`
- `ICreationAuditedObject` 继承 `IHasCreationTime` 和 `IMayHaveCreator`，所以它定义了以下属性：
 - `CreationTime`
 - `CreatorId`
- `IHasModificationTime` 定义了以下属性：
 - `LastModificationTime`
- `IModificationAuditedObject` 扩展 `IHasModificationTime` 并添加了 `LastModifierId` 属性。所以它定义了以下属性：
 - `LastModificationTime`
 - `LastModifierId`

- `IAuditedObject` 扩展 `ICreationAuditedObject` 和 `IModificationAuditedObject`, 所以它定义了以下属性:
 - `CreationTime`
 - `CreatorId`
 - `LastModificationTime`
 - `LastModifierId`
- `ISoftDelete` (参阅 数据过滤文档) 定义了以下属性:
 - `IsDeleted`
- `IHasDeletionTime` 扩展 `ISoftDelete` 并添加了 `DeletionTime` 属性. 所以它定义了以下属性:
 - `IsDeleted`
 - `DeletionTime`
- `IDeletionAuditedObject` 扩展 `IHasDeletionTime` 并添加了 `DeleterId` 属性. 所以它定义了以下属性:
 - `IsDeleted`
 - `DeletionTime`
 - `DeleterId`
- `IFullAuditedObject` 继承 `IAuditedObject` 和 `IDeletionAuditedObject`, 所以它定义了以下属性:
 - `CreationTime`
 - `CreatorId`
 - `LastModificationTime`
 - `LastModifierId`
 - `IsDeleted`
 - `DeletionTime`
 - `DeleterId`

审计基类简化接口实现

- `CreationAuditedEntity<TKey>` 和 `CreationAuditedAggregateRoot<TKey>` 实现了 `ICreationAuditedObject` 接口.
- `AuditedEntity<TKey>` 和 `AuditedAggregateRoot<TKey>` 实现了 `IAuditedObject` 接口.
- `FullAuditedEntity<TKey>` and `FullAuditedAggregateRoot<TKey>` 实现了 `IFullAuditedObject` 接口.

所有这些基类都有非泛型版本, 例如: 可以使用 `AuditedEntity` 和 `FullAuditedAggregateRoot` 来支持复合主键。

在实体或者聚合中添加用户导航属性: `FullAuditedAggregateRootWithUser<TUser>` 和 `FullAuditedAggregateRootWithUser<TKey, TUser>`

扩展属性支持

`AggregateRoot` 基类已经实现了 `IHasExtraProperties` 接口, 实体需要自己实现。

对于 EF Core 创建一个 `ExtraProperties` 字段, 里边存储的 JSON 实体序列化结果。

对于 MongoDB 数据库, 天生支持扩展属性存储位 JSON 文档。

最佳实践

实体也是聚合根，在领域层中定义，提供一个构造函数给ORM用，使用 ID 引用聚合根而不是导航属性，总是将属性和方法定义为 virtual 保护级，不推荐在聚合根中使用复合主键，推荐使用 Guid 作为主键，聚合尽可能小一些会提高内聚性和性能。

参考资料

[聚合（根）、实体、值对象精炼思考总结](#)