**值对象**

值对象没有对应的数据库表

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {

            modelBuilder.Entity<Person>().OwnsOne(e => e.ValueObject);

        }

        public DbSet<Person> Persons { get; set; }

}

    public class Person

    {

        public int Id { get; set; }

        public ValueObject ValueObject { get; set; }

}

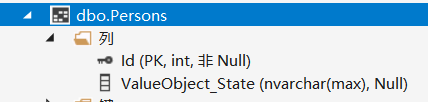
    public class ValueObject

    {

        public string State { get; set; }

    }

数据库字段



**继承模式**

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Person>().HasDiscriminator(e => e.Type).HasValue<SmallPerson>("SmallPerson").HasValue<BigPerson>("BigPerson");

        }

        public DbSet<Person> Persons { get; set; }

}

    public abstract class Person

    {

        public int Id { get; set; }

        public string Type { get; set; }

}

    public class SmallPerson : Person

    {

        public string SmallPersonField { get; set; }

}

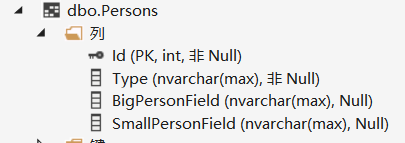
    public class BigPerson : Person

    {

        public string BigPersonField { get; set; }

    }

数据库字段



注：多继承的数据保存在一张表上，但在 EF6 和 EFCore 5.0（2020.11发布） 上可以支持每个实体对应一张表

另一种多表保存的设计

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

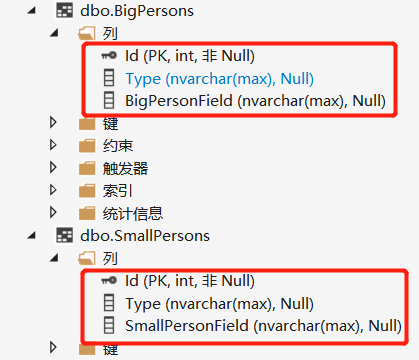
        }

        public DbSet<SmallPerson> SmallPersons { get; set; }

        public DbSet<BigPerson> BigPersons { get; set; }

    }

数据库字段



这种设计使用每个实体都有各自的数据库表，但这种设计只是在代码上实现的继承，但数据库中并没有实现继承，举个例子：查找最近添加的10个Person，怎么查找？

**状态模式**

原本状态类是想使用值对象，但EFCore值对象模式不支持集成，所以只能使用实体

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {

            modelBuilder.Entity<Person>();

            modelBuilder.Entity<StateBase>().HasDiscriminator(e => e.State).HasValue<State1>("State1").HasValue<State2>("State2");

        }

        public DbSet<Person> Persons { get; set; }

}

    // 人

    public class Person

    {

        public int Id { get; set; }

        // 人的状态

        public StateBase State { get; set; }

}

    // 状态基类

    public abstract class StateBase

    {

        public int Id { get; set; }

        public string State { get; set; }

}

    // 状态1

    public class State1: StateBase

    {

        public string FieldState1 { get; set; }

}

    // 状态2

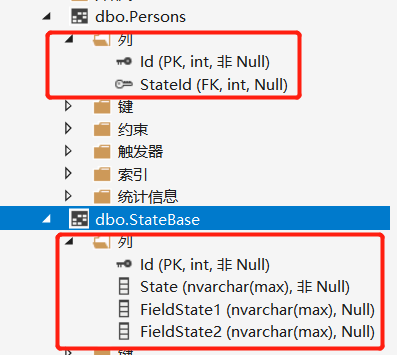
    public class State2: StateBase

    {

        public string FieldState2 { get; set; }

    }

数据库字段



**规格**

如，有一个需求，绳索的载重量必须是大于电梯载重量的2倍

实际上这是一个规格的设计，但规格并不保存再数据库中，所以EF实现起来也是比较简单

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

        }

        public DbSet<Lift> Lifts { get; set; }

}

    /// <summary>

    /// 电梯

    /// </summary>

    public class Lift

    {

        public int Id { get; set; }

        // 载重量

        public int Carry { get; set; }

        public Line Line { get; set; }

}

    /// <summary>

    /// 挂载电梯的绳锁

    /// </summary>

    public class Line

    {

        public int Id { get; set; }

        // 载重量

        public int Carry { get; set; }

}

    /// <summary>

    /// 电梯载重量约束（规格）

    /// </summary>

    public class LiftCarryCheck

    {

        public bool Check(Lift lift) {

            return lift.Carry \* 2 < lift.Line.Carry;

        }

    }

**组合模式**

哎，没什么好说的，看代码

    class MyContext : DbContext

    {

        public MyContext(DbContextOptions<MyContext> options) : base(options) { }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            modelBuilder.Entity<Menu>().HasDiscriminator(e => e.MenuType).HasValue<LiftMenu>("LiftMenu").HasValue<ComposeMenu>("ComposeMenu");

        }

        public DbSet<Menu> Menus { get; set; }

}

    /// <summary>

    /// 菜单

    /// </summary>

    public abstract class Menu

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string MenuType { get; set; }

}

    /// <summary>

    /// 叶子菜单

    /// </summary>

    public class LiftMenu : Menu

    {

}

    /// <summary>

    /// 组合菜单

    /// </summary>

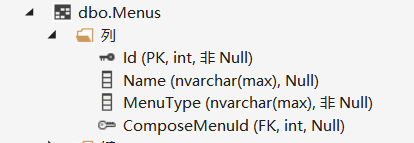
    public class ComposeMenu: Menu

    {

        public IQueryable<Menu> Childs { get; set; }

    }

数据库字段



值得一提的是，上面的MyContext中，我一般只DBSet一个实体，因为大多数情况我们只公布根实体，具体原因查看领域驱动设计