**术语定义**

相关实体： 这是包含外键属性的实体。 有时称为关系的 "子级"。

主体实体： 这是包含主/备用键属性的实体。 有时称为关系的 "父项"。

主体密钥： 唯一标识主体实体的属性。 这可能是主键或备用密钥。

外键： 用于存储相关实体的主体键值的依赖实体中的属性。

导航属性： 在主体和/或从属实体上定义的属性，该属性引用相关实体。

集合导航属性： 一个导航属性，其中包含对多个相关实体的引用。

引用导航属性： 保存对单个相关实体的引用的导航属性。

反向导航属性： 讨论特定导航属性时，此术语是指关系另一端的导航属性。

自引用关系： 依赖关系和主体实体类型相同的关系。

**下面的代码显示 Blog 与 Post 之间的一对多关系。**

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public int BlogId { get; set; }

public Blog Blog { get; set; }

}

Post 是依赖实体

Blog 是主体实体

Blog.BlogId 是主体键（在本例中为主键而不是备用密钥）

Post.BlogId 为外键

Post.Blog 是一个引用导航属性

Blog.Posts 是集合导航属性

Post.Blog 是 Blog.Posts 的反向导航属性（反之亦然）

注：public属性是必须的

**约定**

如下：EF为这些类建立一对多关系

示例1：

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public int BlogId { get; set; }

public Blog Blog { get; set; }

}

示例2：

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public Blog Blog { get; set; }

}

示例3：

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

}

**自己配置关系**

使用如下Api实体间配置关系

HasOne/WithOne：一对一

HasOne/WithMany：一对多

HasMany/WithOne：多对一（一对多）

class MyContext : DbContext

{

public DbSet<Blog> Blogs { get; set; }

public DbSet<Post> Posts { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Post>()

// 一个Blog

.HasOne(p => p.Blog)

// 对应多个Post

.WithMany(b => b.Posts);

}

}

**单个导航属性**

我们一般不会在两个实体中都配置导航属性，如下配置当导航属性

class MyContext : DbContext

{

public DbSet<Blog> Blogs { get; set; }

public DbSet<Post> Posts { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>()

.HasMany(b => b.Posts)

.WithOne();

}

}

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

}

**指定外键**

class MyContext : DbContext

{

public DbSet<Blog> Blogs { get; set; }

public DbSet<Post> Posts { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Post>()

.HasOne(p => p.Blog)

.WithMany(b => b.Posts)

.HasForeignKey(p => p.BlogForeignKey);

}

}

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public List<Post> Posts { get; set; }

}

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public int BlogForeignKey { get; set; }

public Blog Blog { get; set; }

}

指定约束名称

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Post>()

.HasOne(p => p.Blog)

.WithMany(b => b.Posts)

.HasForeignKey(p => p.BlogId)

.HasConstraintName("ForeignKey\_Post\_Blog");

}

**主体密钥**

如果你希望外键引用主键之外的属性，则可以使用如下 API 来配置关系的主体键属性。

class MyContext : DbContext

{

public DbSet<Car> Cars { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<RecordOfSale>()

.HasOne(s => s.Car)

.WithMany(c => c.SaleHistory)

// 指定外键

.HasForeignKey(s => s.CarLicensePlate)

// 对应另一个实体的LicensePlate字段，而不是Id

.HasPrincipalKey(c => c.LicensePlate);

}

}

public class Car

{

public int CarId { get; set; }

public string LicensePlate { get; set; }

public string Make { get; set; }

public string Model { get; set; }

public List<RecordOfSale> SaleHistory { get; set; }

}

public class RecordOfSale

{

public int RecordOfSaleId { get; set; }

public DateTime DateSold { get; set; }

public decimal Price { get; set; }

public string CarLicensePlate { get; set; }

public Car Car { get; set; }

}

**配置外键是否必须的**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Post>()

.HasOne(p => p.Blog)

.WithMany(b => b.Posts)

.IsRequired();

}

**级联删除**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Post>()

.HasOne(p => p.Blog)

.WithMany(b => b.Posts)

.OnDelete(DeleteBehavior.Cascade);

}

DeleteBehavior.Cascade 为删除关联实体

**一对一**

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

public BlogImage BlogImage { get; set; }

}

public class BlogImage

{

public int BlogImageId { get; set; }

public byte[] Image { get; set; }

public string Caption { get; set; }

public int BlogId { get; set; }

public Blog Blog { get; set; }

}

**一对一Api配置**

class MyContext : DbContext

{

public DbSet<Blog> Blogs { get; set; }

public DbSet<BlogImage> BlogImages { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>()

.HasOne(b => b.BlogImage)

.WithOne(i => i.Blog)

.HasForeignKey<BlogImage>(b => b.BlogForeignKey);

}

}

**多对多**

目前尚不支持多对多关系