**安装**

新建项目，在引用中右键->管理nuget程序包，搜索 entity framework, 安装，默认会引用 EntityFramework 和EntityFramework.SqlServer

**基本概念**

通过实体类来映射表结构，通过实体类的对象来操作数据，DbContext来管理实体类的上下文状态，SaveChanges 方法来根据状态提交sql到数据库。

**Dbfirst方式**

在sqlserver数据库中新建表

Nuget获取最新版ef

从已有的表中新建实体，新建项 -> 数据->实体数据模型，选择“来自数据库的EF设计器”，最终生成了一个edmx文件。完成从数据库到实体的映射（通过 csdl msl ssdl 3个xml文件完成映射）。

代码实现增删查改

当数据库表结构有变化的时候，打开edmx文件，在设计器的空白位置右键->从数据库更新模型

Dbfirst方式会生成 csdl msl ssdl 等映射文件，因此连接字符串需要包含此3个文件的元数据，**有较多的依赖，不推荐使用**

**CodeFirst方式**

新建一个空数据库（如果数据库已经存在，则用已有的）

Nuget获取最新版ef

新建项->ADO.NET实体数据模型，选择来自数据库的CodeFirst,选择要映射的表

最终每个表生成了一个.cs文件，数据库也生成一个.cs文件。完成从数据库到实体的映射

可以通过操作实体类来实现增删查改来操作数据库了

当实体有变化，需要映射到数据库的时候

打开 程序包管理器控制台(工具-Nuget包管理器中)

执行 Enable-Migrations

当需要将实体的变化提交到数据库时 执行 Add-Migration 变更名称 (名称自己取)

执行Update-DataBase –Verbose (-Verbose为可选项，会输出执行的sql)

可以通过操作实体类来实现增删查改来操作数据库了

**Data Annotations**

Key 主键，支持多个字段组成联合主键，当只有一个属性为[Key]时，默认会是自增字段

ForeignKey 外键， 用在n 表中，指向1表的主键

ConcurrencyCheck 更新的时候的检查列，生成update语句是会加此列的判断

Table 表名称

Column 列名称，可以设置列的类型

DatabaseGenerated 自增字段 ，主键默认自增，非主键才需要设置

Required 必填项

MaxLength 字符串长度

NotMapped 不映射到表

ComplexType 映射到多个字段的组合

TimeStamp rowvesion,查看列更新的版本字段

Index 索引， 多字段，是否聚集，是否唯一

代码示例：

using System;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.ComponentModel.DataAnnotations;

using System.Collections.Generic;

public partial class Model1 : DbContext

{

public Model1()

: base("name=Model1")

{

}

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

public DbSet<Artcle> Artcles { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

}

}

[Table(name: "NewPerson")]

public class Person

{

[Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public long PersonNumber { get; set; }

[Index("PersonIndex", 1, IsClustered = false, IsUnique = false, Order = 1)]

[Required, MaxLength(20), ConcurrencyCheck]

public string PersonName { get; set; }

[Column(name: "Remark2", Order = 1, TypeName = "varchar")]

public string Remark { get; set; }

public DetailInfo Detail { get; set; }

[Timestamp]

public byte[] Version { get; set; }

[NotMapped]

public string NickName { get; set; }

}

public class Artcle

{

[Key, Column(Order = 1)]

public int ArtcleType { get; set; }

[Key, Column(Order = 2)]

public int ArtcleOrder { get; set; }

[Index("ArticleIndex", 1, IsClustered = true, IsUnique = true, Order = 1)]

public int ArtcleNo { get; set; }

[Index("ArticleIndex", 2, IsClustered = true, IsUnique = true, Order = 2)]

public int ArtcleSubNo { get; set; }

public long WriterPersonNumber { get; set; }

[ForeignKey("WriterPersonNumber")]

public Person Writer { get; set; }

}

[ComplexType]

public class DetailInfo

{

public string Home { get; set; }

public string School { get; set; }

}

**Fluent API**

Data Annotations 会对 实体类有一定的侵入性， 而且有些映射不方便使用Data Annotations实现，此时可以通过Fluent API实现，通过override dbContext的 OnModelCreating 方法，在其中书写Fluent API建立映射

代码示例

using System;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.ComponentModel.DataAnnotations;

using System.Collections.Generic;

using System.Data.Entity.Infrastructure.Annotations;

public partial class Model1 : DbContext

{

public Model1()

: base("name=Model1")

{

}

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

public DbSet<Artcle> Artcles { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Person>().HasKey(t => t.PersonNumber);

modelBuilder.Entity<Person>().Property(t => t.PersonNumber).HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

modelBuilder.Entity<Person>().Property(t => t.PersonName).HasMaxLength(20).IsRequired().IsConcurrencyToken();

modelBuilder.Entity<Person>().Property(t => t.PersonName).HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute("PersonIndex") { IsClustered= false, IsUnique = false,Order=1}));

modelBuilder.Entity<Person>().Ignore(t => t.NickName);

modelBuilder.Entity<Person>().Property(t => t.Remark).HasColumnName("Remark2").HasColumnType("varchar");

modelBuilder.Entity<Person>().Property(t => t.Version).IsRowVersion();

modelBuilder.Entity<Artcle>().HasKey(t => new { t.ArtcleType, t.ArtcleOrder });

modelBuilder.Entity<Artcle>().Property(t => t.ArtcleNo).HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute("ArticleIndex") { IsClustered = true, IsUnique = true , Order = 1 }));

modelBuilder.Entity<Artcle>().Property(t => t.ArtcleSubNo).HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute("ArticleIndex") { IsClustered = true, IsUnique = true, Order = 2 }));

modelBuilder.Entity<Artcle>().HasRequired(c => c.Writer)

.WithMany(t => t.Artcles)

.Map(m => m.MapKey("WriterPersonNumber"));

}

}

[Table(name: "NewPerson")]

public class Person

{

public long PersonNumber { get; set; }

public string PersonName { get; set; }

public string Remark { get; set; }

public DetailInfo Detail { get; set; }

public byte[] Version { get; set; }

public string NickName { get; set; }

public virtual ICollection<Artcle> Artcles { get; private set; }

}

public class Artcle

{

public int ArtcleType { get; set; }

public int ArtcleOrder { get; set; }

public int ArtcleNo { get; set; }

public int ArtcleSubNo { get; set; }

public Person Writer { get; set; }

}

public class DetailInfo

{

public string Home { get; set; }

public string School { get; set; }

}

**最佳实践**

最开始在数据中建立表结构，完成codefirst映射，当有变更或补充的时候，使用Data Annotations 作为主要映射手段，Fluent API为辅助手段，使用业务层的实体，做业务处理和界面显示（如果需要，使用显示层的实体）

**使用枚举**

可以使用定义一个枚举，并将其作为属性放在实体类中，EF会自动完成映射

**乐观并发的控制和处理**

当对实体类的某个属性设置了乐观并发，当更新的时候，数据库中的值比加载时有变更，则会抛出异常，此时需要捕获异常，并进行处理。 参考：https://msdn.microsoft.com/en-us/library/jj592904(v=vs.113).aspx

**执行原生sql**

Dbcontext 的SqlQuery 方法 和 DbSet 的SqlQuery 方法可以执行sql并返回值和列表

示例1： 返回实体列表

DbSqlQuery<Artcle> curQuery = curModel.Artcles.SqlQuery("select \* from Artcles ");

List<Artcle> tempList = curQuery.ToList();、

示例2: 返回值类型

DbRawSqlQuery<int> curResult = curModel.Database.SqlQuery<int>("select count(\*) from Artcles ");

List<int> intList = curResult.ToList();

int value = intList(0);

**事务**

**默认的事务:**

using (var db = new BloggingContext())

{

db.Blogs.Add(new Site { Url = "http://msdn.com/data/ef" });

db.Blogs.Add(new Site { Url = "http://blogs.msdn.com/adonet" });

db.SaveChanges();

}

**手动控制的事务**

using (var db = new BloggingContext())

{

MyConfiguration.SuspendExecutionStrategy = true;

using (var trn = db.Database.BeginTransaction())

{

db.Blogs.Add(new Blog { Url = "http://msdn.com/data/ef" });

db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });

db.SaveChanges();

db.Blogs.Add(new Blog { Url = "http://twitter.com/efmagicunicorns" });

db.SaveChanges();

trn.Commit();

}

MyConfiguration.SuspendExecutionStrategy = false;

}

**日志输出**

Interceptors,植入的操作，可以用系统自带的，也可以自己定义。

例如：使用系统自带的interceptors输出sql日志，在<entityFramework> 中添加如下配置完成 将执行的sql输出到对应的文件中

<interceptors>

<interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">

<parameters>

<parameter value="C:\Temp\LogOutput.txt"/>

</parameters>

</interceptor>

</interceptors>