## 处理模型更改—CodeFirst迁移

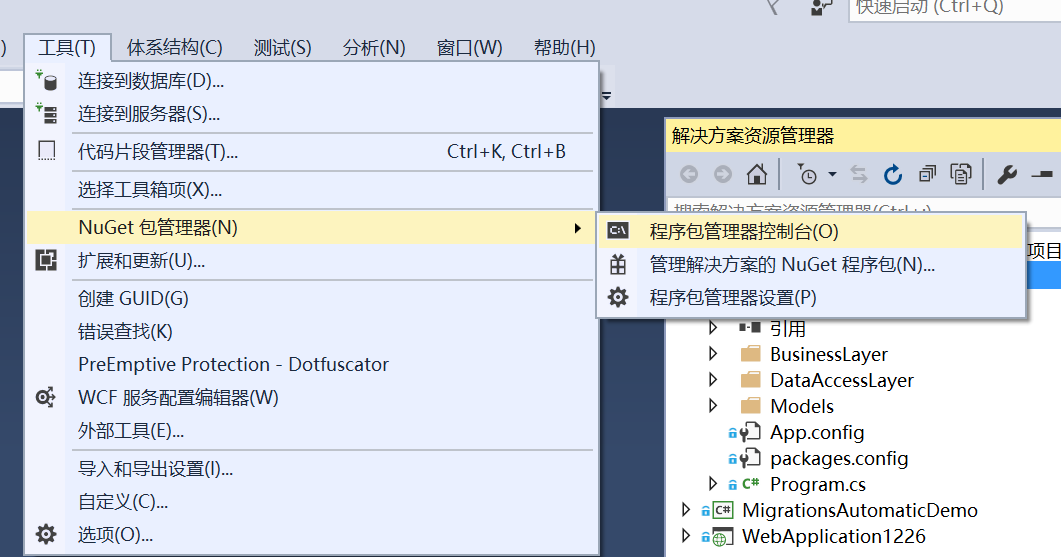
实现模型变更和数据库架构更新需要使用 Code First 迁移功能。迁移使我们可以有序的集的步骤，进行升级 （和降级） 我们数据库架构。

迁移工具支持：

* 创建可用于 EF 模型的初始数据库
* 生成迁移以跟踪对 EF 模型所做的更改
* 使数据库随时掌握这些更改

### 启用 Code First 迁移

**（1）工具->NuGet包管理器->程序包管理器控制台**



**（2）在包管理器控制台中运行 Enable-Migrations 命令**

|  |
| --- |
|  |

一个新的 Migrations 文件夹已添加到我们包含两个项目的项目：

|  |
| --- |
|  |

Configuration.cs – 此文件包用于迁移的设置 bloggingcontext。

<时间戳>\_InitialCreate.cs –记录迁移历史，文件名上的时间戳用于排序目的。

### 生成和运行迁移

Code First 迁移有两个主要命令。

**Add-Migration** 将基于自上次迁移创建以来对模型所做的更改来构建下一次迁移

**Update-Database** 将对数据库应用任意挂起的迁移

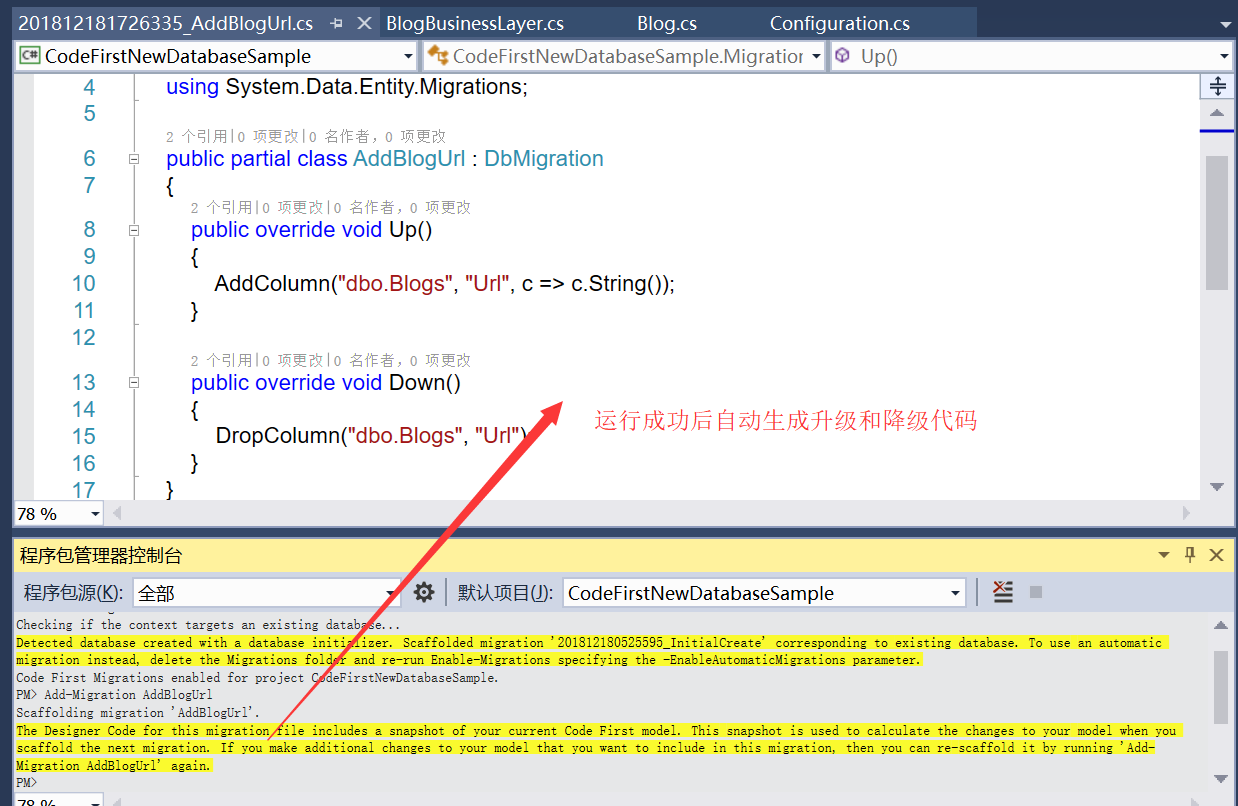
### 引入新属性

**步骤1将 Url 属性引入Blog类**

|  |
| --- |
|  |

如果要再次运行应用程序，则会收到一个 InvalidOperationException报错，指出“创建数据库后，支持‘BlogContext’上下文的模型已发生变化。请考虑使用 Code First 迁移更新数据库 (<http://go.microsoft.com/fwlink/?LinkId=238269)>”。

**步骤2 在包管理器控制台中运行 Add-Migration AddBlogUrl 命令**



**步骤3  在包管理器控制台运行 Update-Database 命令将迁移应用到数据库。**

|  |
| --- |
|  |

在数据库中可见新增属性成功

|  |
| --- |
|  |

### 自定义迁移

设置新增属性Rating的默认值

步骤1为博客类引入一个评价等级属性Rating

|  |
| --- |
|  |

步骤2 运行运行Add-migration AddBlogRating程序包管理器控制台命令。

|  |
| --- |
|  |

步骤3 修改更新日志类，设置默认值

|  |
| --- |
|  |

步骤4运行Update-database程序包管理器控制台命令更新数据库

|  |
| --- |
|  |

检验数据库变更

|  |
| --- |
|  |

### 强行删除类/属性

**步骤1删除Post类和Blog类中的相关导航属性**

|  |
| --- |
|  |

**步骤2 设置配置文件自动变更数据库属性为真**

|  |
| --- |
|  |

**步骤3控制台运行强制更新数据库命令**

|  |
| --- |
| **Update-database -force** |

|  |
| --- |
|  |

查看数据库

|  |
| --- |
|  |

删除Blog两个新增属性也可以使用相同的方法

### 增加类（本例中Post类在3.4.5中被删除了，本节中重新增加）

**步骤1 新增Post类 及设置Blog类中的导航属性**

|  |
| --- |
| public class Post  {  public int PostId { get; set; }  [MaxLength(200)]  public string Title { get; set; }  public string Content { get; set; }  public int BlogId { get; set; }  public Blog Blog { get; set; }  } |
| public class Blog  {  public int BlogId { get; set; }  public string Name { get; set; }  **public virtual ICollection<Post> Posts { get; set; }**  } |

非自动更新法AutomaticMigrationsEnabled = false;

步骤2在包管理器控制台中运行 Add-Migration AddPostClass 命令

|  |
| --- |
|  |

步骤3在包管理器控制台中运行 Update-Database –Verbose 命令

–Verbose参数更新数据库的同时可以查看对应的SQL命令

检查数据库变更

## CodeFirst约定

Code First 允许使用类描述模型。 通过使用约定来检测到该模型的基本形状。 约定是用于自动配置基于类定义时使用 Code First 的概念模型的规则集。 约定 在System.Data.Entity.ModelConfiguration.Conventions 命名空间中定义。

<https://docs.microsoft.com/zh-cn/previous-versions/visualstudio/gg696316(v=vs.113)>

**规则优先级 fluent API>跟数据注释>约定**

### 类型发现

使用 Code First 开发时通常首先编写.NET Framework 类定义概念 （域） 模型。 除了定义类，还需要让DbContext知道想要在模型中包含的类型。 若要执行此操作，得定义派生上下文类DbContext ，并公开DbSet想要将模型的一部分的类型的属性。 代码首先将包括这些类型，并还会中拉入任何引用的类型，即使不同程序集中定义引用的类型。

### 主要密钥约定

代码首先将推断属性为主键，是否一个类上的属性名为"ID"（不区分大小写），或类名后, 跟"ID"。 如果主键属性的类型是数值或 GUID 它就是配置为标识列。

### 关系约定

在实体框架中，导航属性提供一种导航两个实体类型之间的关系方法。 针对对象参与到其中的每个关系，各对象均可以具有导航属性。 导航属性，您可以导航和管理两个方向，返回引用对象中的关系 (在重数为两者中任何一个或零或一) 或集合 （如果重数为多个）。 代码首先会推断出关系基于您的类型上定义的导航属性。

### 复杂类型约定

当 Code First 发现其中主键不能推断出，并通过数据注释或 fluent API 注册没有主键的类定义时，然后该类型自动注册为复杂类型中。

将以下代码置入一个新控制台项目中，引用EF，设置连接串，使用上下文。观察数据库。理解约定。

|  |
| --- |
| public class SchoolEntities : DbContext  {  public DbSet<Department> Departments { get; set; }  }  public class Department  {  // Primary key  public int DepartmentID { get; set; }  public string Name { get; set; }  // Navigation property  public virtual ICollection<Course> Courses { get; set; }  }  public class Course  {  // Primary key  public int CourseID { get; set; }  public string Title { get; set; }  public int Credits { get; set; }  // Foreign key  public int DepartmentID { get; set; }  // Navigation properties  public virtual Department Department { get; set; }  }  public partial class OnlineCourse : Course  {  public string URL { get; set; }  }  public partial class OnsiteCourse : Course  {  public string Location { get; set; }  public string Days { get; set; }  public System.DateTime Time { get; set; }  } |

## 数据注释

<https://docs.microsoft.com/zh-cn/ef/ef6/modeling/code-first/data-annotations>