**创建初始模型和数据库**

　　在开始使用迁移（Migrations）之前，我们需要一个 Project 和一个 Code First Model, 对于本文将使用典型的 Blog 和 Post 模型

* 创建一个新的控制台应用程序 **MigrationsDemo**;
* 添加最新的 EntityFramework 到项目
  + Tools –> Library Package Manager –> Package Manager Console;
  + 运行命令 **Install-Package EntityFramework**
* 创建 **Blog.cs** 和 DbContext 的派生类 **BlogContext.cs**

public class Blog

{

public int BlogId { get; set; }

public string Name { get; set; }

}

public class BlogContext : DbContext

{

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

}

　　更改 Program.cs 以调用

[IMG_256](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

static void Main(string[] args)

{

using (var db = new BlogContext())

{

db.Blogs.Add(new Blog { Name = "Another Blog" });

db.SaveChanges();

foreach (var blog in db.Blogs)

{

Console.WriteLine(blog.Name);

}

}

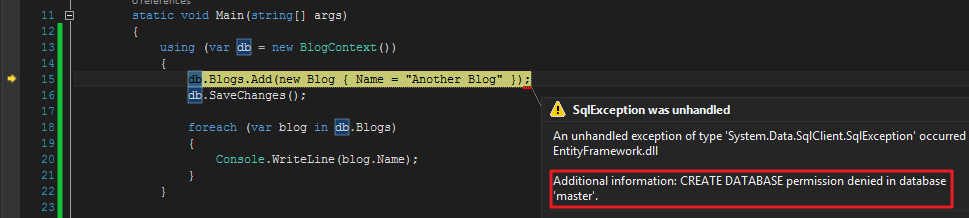
Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

[IMG_257](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

　　运行查看结果

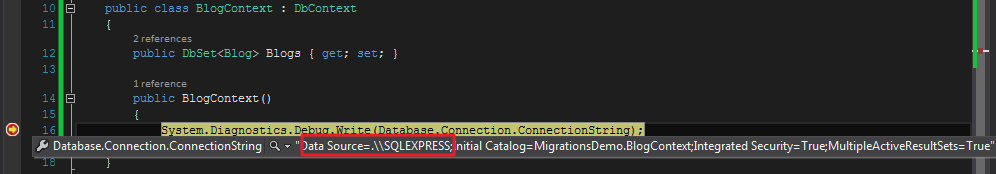


　　发现如上错误"CREATE DATABSE permission denied in databse 'master'"

　　我们在 BlogContext 上的无参构造函数上添加诊断代码并设置调试断点

System.Diagnostics.Debug.Write(Database.Connection.ConnectionString);

　　再次运行



　　我们注意到 Data Scource 竟然是 .\\SQLEXPRESS 而不是我们想要的 localDB , 这是因为：

* 如果我们安装了 SQL Express，那么 database 将会安装在 local SQL Express instance，否则 Code First 才将尝试使用 localDB;
* SQL Express 总是具有优先权，只要安装了它

　　知道了原因我们就好解决了：

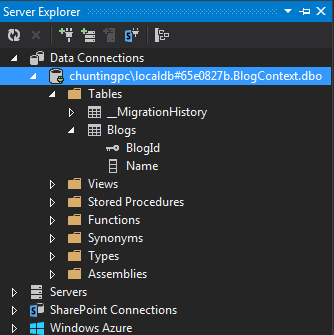
* 如果想继续使用 SQL Express，那么就配置相应地权限，请参考 http://odetocode.com/Blogs/scott/archive/2012/08/14/a-troubleshooting-guide-for-entity-framework-connections-amp-migrations.aspx;
* 如果想改用 localDB, 只需在.config 配置即可（放在 configSections 节点后面）

<connectionStrings>

<add name="BlogContext" connectionString="Data Source=(LocalDb)\v11.0;Initial Catalog=BlogContext;Integrated Security=SSPI;" providerName="System.Data.SqlClient"/>

</connectionStrings>

　　再次运行就行了，让我们看一下后台生成的数据库

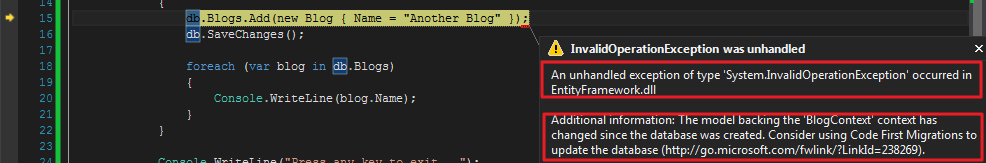


**启用迁移**

　　我们对模型 Blog 做一些更改：增加一个 Url 属性

public string Url { get; set; }

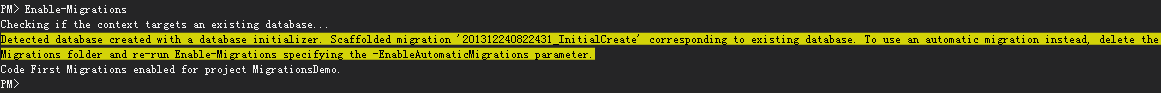
　　我们此时再次运行程序，发现如下错误

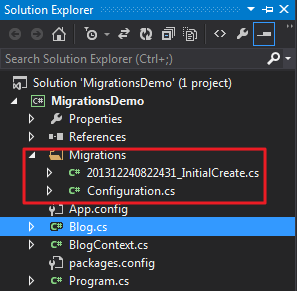


　　'InvalidOperationException' was unhandled. The model backing the 'BlogContext' context has changed since the database was created. Consider using Code First Migrations to update the database ( <http://go.microsoft.com/fwlink/?LinkId=238269>)

　　正如错误消息提示的那样，是时候使用 Code First Migrations，第一步是运行如下的命令：

* 在 Package Manager Console 下运行命令 **Enable-Migrations**

****



　　这个命令将在项目下创建文件夹 **Migrations**

* **The Configuration class** 这个类允许你去配置如何迁移，对于本文将使用默认的配置（在本文中因为只有一个 Context, Enable-Migrations 将自动对 context type 作出适配）;
* An InitialCreate migration（本文为 201312240822431\_InitialCreate.cs）这个迁移之所以存在是因为我们之前用 Code First 创建了数据库, 在启用迁移前，scaffolded migration 里的代码表示在数据库中已经创建的对象，本文中即为表 Blog （列 BlogId 和 Name）. 文件名包含一个 timestamp 以便排序（如果之前数据库没有被创建，那么 InitialCreate migration 将不会被创建，相反，当我们第一次调用 Add-Migration 的时候所有表都将归集到一个新的 migration 中）

**多个实体锁定同一数据库**

　　当使用 EF6 之前的版本时，只会有一个 Code First Model 被用来生成/管理数据库的 Schema, 这将导致每个数据库只会有一张 **\_\_MigrationsHistory** 表，从而无法辨别实体与模型的对应关系。

　　 从 EF6 开始，**Configuration** 类将会包含一个 **ContextKey** 属性，它将作为每一个 Code First Model 的唯一标识符， **\_\_MigrationsHistory** 表中一个相应地的列允许来自多个模型（multiple models）的实体共享表（entries），默认情况下这个属性被设置成 context 的完全限定名。

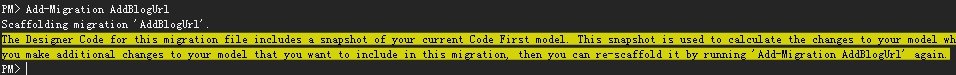
**生成、运行迁移**

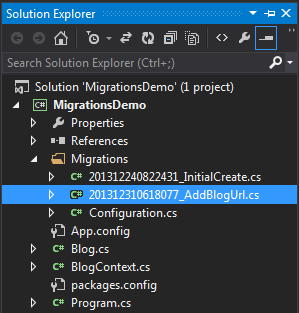
　　Code First Migrations 有两个你需要熟悉的命令：

* Add-Migration 将 scaffold 创建下一次基于上一次迁移以来的更改的迁移;
* **Update-Databse** 将任何挂起的迁移应用到数据库

　　我们需要脚手架（scaffold 直译）一个迁移，以上面的 Url 属性为例，命令 **Add-Migration** 允许我们对迁移命名，我们姑且称之为 **AddBlogUrl**

* 在 Package Manager Console 中运行命令 Add-Migration AddBlogUrl;
* 一个新的迁移（名称包含 timestamp 前缀）在目录 **Migrations** 中创建成功





[IMG_266](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

namespace MigrationsDemo.Migrations

{

using System;

using System.Data.Entity.Migrations;

public partial class AddBlogUrl : DbMigration

{

public override void Up()

{

AddColumn("dbo.Blogs", "Url", c => c.String());

}

public override void Down()

{

DropColumn("dbo.Blogs", "Url");

}

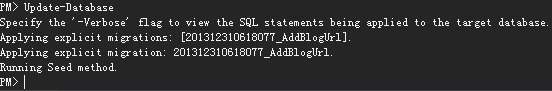
}

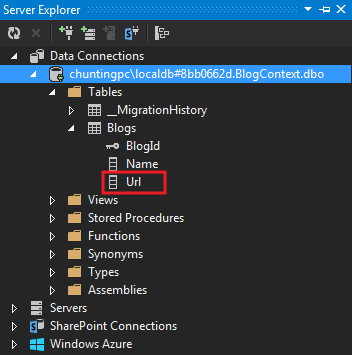
}

[IMG_267](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

　　我们现在可以对这个迁移进行编辑或者增加，但似乎看起来还不错，那我们就直接用 **Update-Database** 来应用到数据库吧

* 在 Package Manager Console 中运行命令 Update-Database ;
* **AddBlogUrl** 迁移将会被应用到数据库（表 Blogs 增加一列 Url）





**定制化迁移**

　　到目前为止我们生成并运行了一个迁移，但是没有对迁移做任何更改，下面我们将尝试做一些更改：在类 Bolg 上增加一属性 **Rating**

public int Rating { get; set; }

　　新建 Post

[IMG_270](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

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; }

}

[IMG_271](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

　　在 **Blog** 中添加 **Post**的集合

public virtual ICollection<Post> Posts { get; set; }

　　在 Package Manager Console 中运行命令 **Add-Migration AddPostClass**

　　生成的迁移如下

IMG_272 View Code

　　接下来我们对迁移做些更改：

* 在 **Posts.Title** 列上增加唯一索引;
* 使 **Blogs.Rating** 列非空，对于表中已经存在的数据，新列都会被赋值成 CLR 的默认数据类型（如 Rating 是整型，故默认值为0），但是我们想指定默认值为3，这样存在的记录将会有一个合理的评分。

　　更改后的代码如下

[IMG_273](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

namespace MigrationsDemo.Migrations

{

using System;

using System.Data.Entity.Migrations;

public partial class AddPostClass : DbMigration

{

public override void Up()

{

CreateTable(

"dbo.Posts",

c => new

{

PostId = c.Int(nullable: false, identity: true),

Title = c.String(maxLength: 200),

Content = c.String(),

BlogId = c.Int(nullable: false),

})

.PrimaryKey(t => t.PostId)

.ForeignKey("dbo.Blogs", t => t.BlogId, cascadeDelete: true)

.Index(t => t.BlogId)

.Index(p => p.Title, unique: true);

AddColumn("dbo.Blogs", "Rating", c => c.Int(nullable: false, defaultValue: 3));

}

public override void Down()

{

DropIndex("dbo.Posts", new[] { "Title" });

DropForeignKey("dbo.Posts", "BlogId", "dbo.Blogs");

DropIndex("dbo.Posts", new[] { "BlogId" });

DropColumn("dbo.Blogs", "Rating");

DropTable("dbo.Posts");

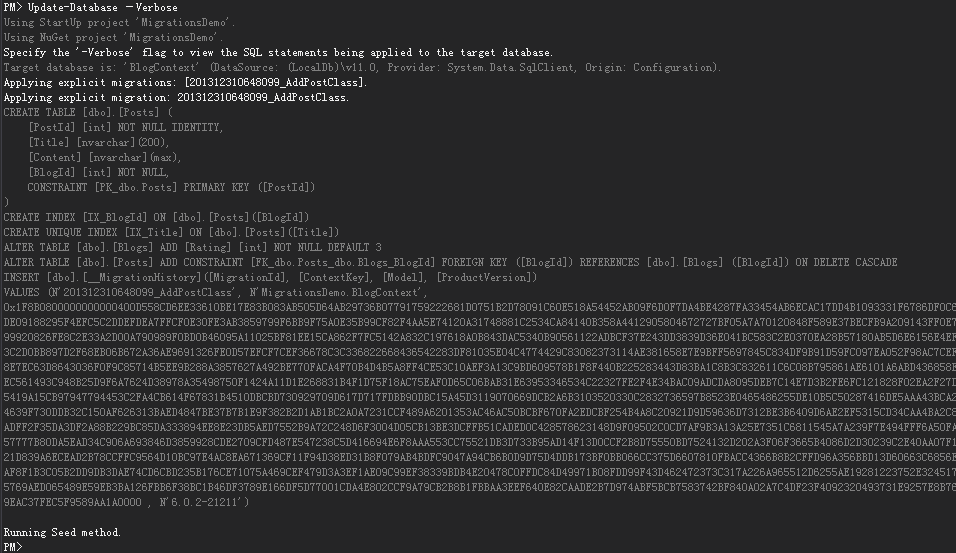
}

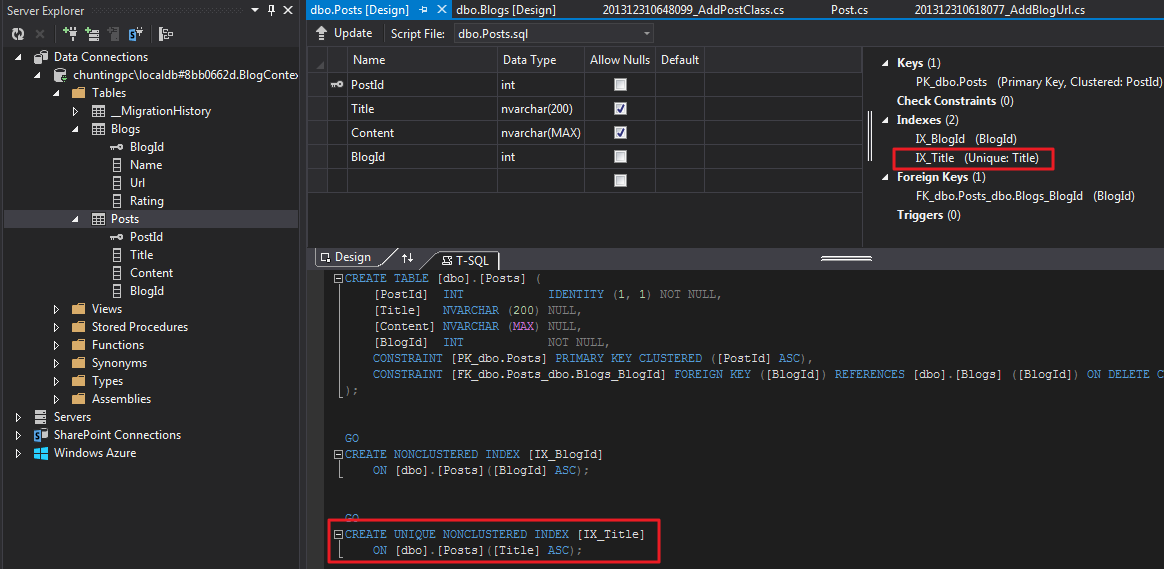
}

}

[IMG_274](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

　　在 Package Manager Console 中运行命令 **Update-Database –Verbose**





**数据移动 / 定制SQL**

　　迄今为止，迁移都没有更改或移动数据，现在让我们看一下需要移动数据的例子。虽然没有对数据移动的原生支持，但是我们可以随意运行 SQL 脚本。

　　让我们在 Post 中增加一个属性 Abstract, 稍后我们使用列 Content 的开头来填充此列（数据库已有记录）

public string Abstract { get; set; }

* 在 Package Manager Console 中运行命令 **Add-Migration AddPostAbstract**;
* 生成的迁移非常好，但是我们想使用 Content 的前 100 个字符来预填充 Abstract 列，我们可对迁移做如下更改

[IMG_277](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

namespace MigrationsDemo.Migrations

{

using System;

using System.Data.Entity.Migrations;

public partial class AddPostAbstract : DbMigration

{

public override void Up()

{

AddColumn("dbo.Posts", "Abstract", c => c.String());

Sql("UPDATE dbo.Posts SET Abstract = LEFT(Content, 100) WHERE Abstract IS NULL");

}

public override void Down()

{

DropColumn("dbo.Posts", "Abstract");

}

}

}

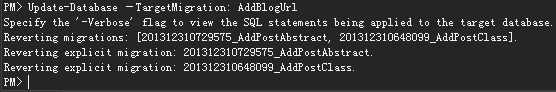
[IMG_278](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

　　在 Package Manager Console 中运行命令 **Update-Database –Verbose**

**迁移至指定版本（包括后退）**

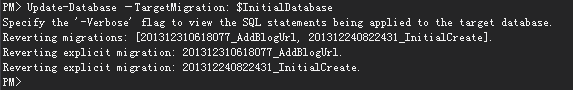
　　迄今为止，我们总是升级至最新迁移，然而某些时候我们需要升级/降级至指定版本，例如我们想迁移数据库至运行 **AddBlogUrl** 迁移之后的状态，此时我们就可以使用 **–TargetMigration** 来降级到这个版本

　　在 Package Manager Console 中运行命令 **Update-Database –TargetMigration: AddBlogUrl**



　　这个命令将会运行 **AddBlogAbstract** and AddPostClass 的 Down 命令

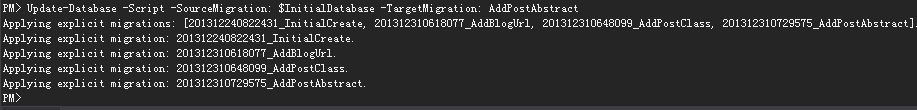
　　如果你想回滚一切至空数据库，可以使用命令 **Update-Database –TargetMigration: $InitialDatabase**



**得到SQL脚本**

　　如果其它开发人员也希望在他们自己的机器上拥有这些更改，他们只需在我们 check in 代码至 source control 的时候做一次同步即可，一旦他们拥有了这些迁移，只需运行命令 Update-Database 就可以把这些更改应用于本地。但是如果我们想把这些更改推送至测试服务器或生产服务器，我们也许需要一份 SQL 脚本提供给 DBA

* 在运行 Update-Database 的时候指定 -Specify 标记，我们就能够使得这些更改被写入一个脚本中而不是被应用，我们同时也会为此脚本指定源迁移和目标迁移，例如我们希望产生的脚本是从一个空数据库（**$InitialDatabase**）到最新的版本（**AddPostAbstract**迁移）;（注意：如果你没有指定目标迁移，那么迁移将始终更新至最新版本；如果你没有指定源迁移，那么迁移将以数据库目前状态为初始）
* 在 Package Manager Console 中运行命令 **Update-Database -Script -SourceMigration: $InitialDatabase -TargetMigration: AddPostAbstract**

****

　　产生的 SQL 脚本如下

IMG_282 View Code

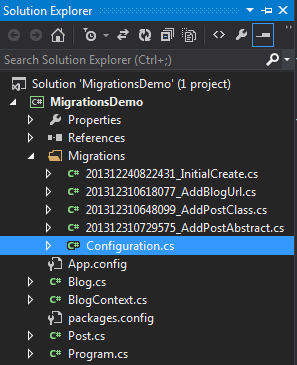
**产生幂等脚本（EF6+）**

　　从 EF6 开始，如果你使用 **–SourceMigration $InitialDatabase**， 产生的脚本将是幂等的，**幂等脚本意味着无论数据库当前处于什么版本/状态，都能升级至最新版本或指定版本**（指定 **–**TargetMigration），生成的脚本包括检查表 **\_\_MigrationsHistory** 的逻辑以及只更新之前从未更新的

**在应用程序启动时自动升级（MigrateDatabaseToLatestVersion初始化器）**

　　当你发布部署应用程序的时候，可能希望当程序启动的时候它自动更新数据库（更新应用任何未更新的迁移），你可以通过注册 MigrateDatabaseToLatestVersion 数据库初始化器来实现这一点，数据库初始化器只包含一些逻辑检查用于确保数据库被正确设置，这个逻辑检查将会在AppDomain 的 context 第一次被使用的时候执行。

　　当我们创建一个初始化器的实例时，需要指定 context type（**BlogContext**）以及 migrations configuration （**Configuration**）- 这个迁移配置类是在我们启用迁移时生成的 **Migrations** 目录下增加的



[IMG_284](http://www.cnblogs.com/panchunting/p/javascript:void(0);" \o "复制代码)

using System;using System.Collections.Generic;using System.Data.Entity;using System.Linq;using System.Text;using System.Threading.Tasks;using MigrationsDemo.Migrations;

namespace MigrationsDemo

{

class Program

{

static void Main(string[] args)

{

Database.SetInitializer(new MigrateDatabaseToLatestVersion<BlogContext, Configuration>());

using (var db = new BlogContext())

{

db.Blogs.Add(new Blog { Name = "Another Blog " });

db.SaveChanges();

foreach (var blog in db.Blogs)

{

Console.WriteLine(blog.Name);

}

}

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

}

}

}

http://www.cnblogs.com/panchunting/p/entity-framework-code-first-migrations.html