第一個程式碼至新的資料庫

發行項 • 2023/09/12

這段影片和逐步解說提供以新資料庫為目標的程式碼優先開發簡介。 此案例包括以不存在的資料庫為目標，而 Code First 將會建立，或 Code First 將新增資料表的空白資料庫。 Code First 可讓您使用 C# 或 VB.Net 類別來定義模型。 您也可以選擇性地使用類別和屬性上的屬性或使用 Fluent API 來執行其他設定。

觀賞影片

這段影片提供以新資料庫為目標的程式碼優先開發簡介。 此案例包括以不存在的資料庫為目標，而 Code First 將會建立，或 Code First 將新增資料表的空白資料庫。 Code First 可讓您使用 C# 或 VB.Net 類別來定義模型。 您也可以選擇性地使用類別和屬性上的屬性或使用 Fluent API 來執行其他設定。

主講人Rowan Miller

影片 ： WMV | MP4 | WMV （ ZIP ）

必要條件

您必須至少安裝 Visual Studio 2010 或 Visual Studio 2012，才能完成本逐步解說。

如果您使用 Visual Studio 2010，您也必須 安裝 NuGet 。

1.建立應用程式

為了簡化工作，我們將建置使用 Code First 執行資料存取的基本主控台應用程式。

- 開啟 Visual Studio
- **檔案 - > 新增 - > 專案...**
- **從左側功能表選取 [Windows] 和 [主控台應用程式]**
- 輸入 **CodeFirstNewDatabaseSample** 作為名稱
- 選取**確定**

2.建立模型

讓我們使用類別來定義非常簡單的模型。 我們只是在 Program.cs 檔案中定義它們，但在真實世界應用程式中，您會將您的類別分割成個別的檔案，而且可能是個別的專案。

在 Program.cs 中的 Program 類別定義下方，新增下列兩個類別。

C#

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual 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 virtual Blog Blog { get; set; }
}
```

您會發現我們正在建立兩個導覽屬性 （ Blog.Posts 和 Post.Blog ） 虛擬。 這會啟用 Entity Framework 的延遲載入功能。 延遲載入表示當您嘗試存取這些屬性時，會自動從資料庫載入這些屬性的內容。

3.建立內容

現在是時候定義衍生內容，其代表與資料庫的會話，讓我們能夠查詢和儲存資料。 我們會定義衍生自 System.Data.Entity.DbCoNtext 的內容，並公開模型中每個類別的具型別 DbSet < TEntity > 。

我們現在開始使用 Entity Framework 中的類型，因此我們需要新增 EntityFramework NuGet 套件。

- **專案 – > 管理 NuGet 套件...** 注意：如果您沒有 **[管理 NuGet 套件...**] 選項，您應該安裝 最新版本的 NuGet
- 選取 [線上] **索引** 標籤
- **選取 EntityFramework** 套件
- 按一下 [安裝]

在 Program.cs 頂端新增 System.Data.Entity 的 using 語句。

C#

```
using System.Data.Entity;
```

在 Program.cs 的 Post 類別下方，新增下列衍生內容。

C#

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

以下是 Program.cs 現在應該包含的完整清單。

C#

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

namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual 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 virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}
```

這就是開始儲存和擷取資料所需的所有程式碼。 顯然，幕後發生了相當多的事情，我們將看看這在一刻，但首先讓我們看看它的行動。

4.讀取 & 寫入資料

在 Program.cs 中實作 Main 方法，如下所示。 此程式碼會建立內容的新實例，然後使用它來插入新的部落格。 然後使用 LINQ 查詢，從依 Title 依字母順序排序的資料庫擷取所有部落格。

C#

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
```

```
                Console.WriteLine(item.Name);
            }

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

您現在可以執行應用程式並進行測試。

主控台

```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...
```

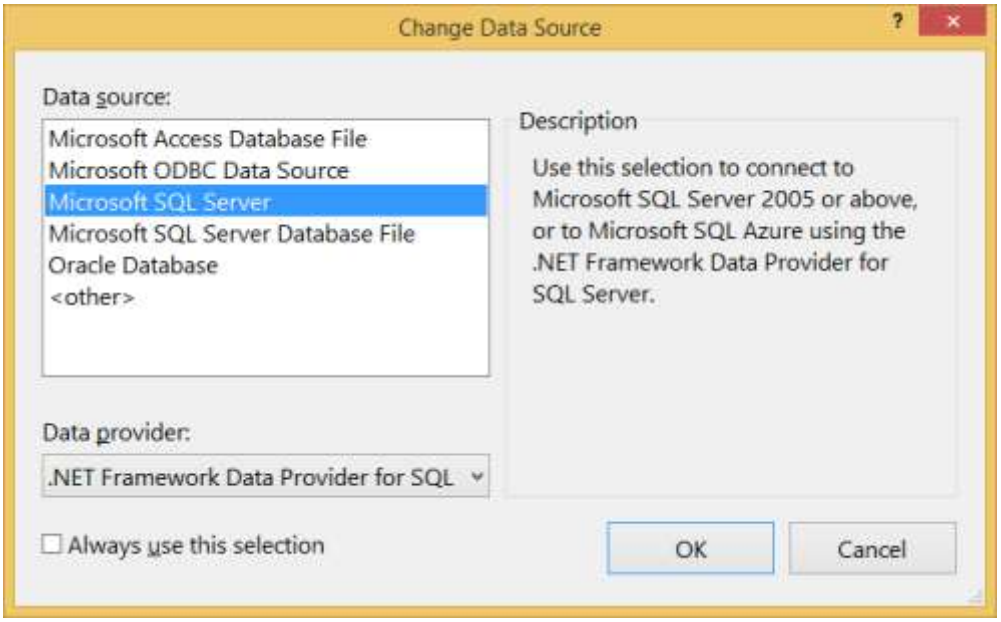
我的資料在哪裡？

根據慣例，DbCoNtext 已為您建立資料庫。

- 如果本機 SQL Express 實例可用（預設使用 Visual Studio 2010 安裝），Code First 就會在該實例上建立資料庫
- 如果無法使用 SQL Express，Code First 會嘗試並使用 LocalDB （預設會隨 Visual Studio 2012 安裝）
- 資料庫是以衍生內容的完整名稱命名，在我們的案例中為 **CodeFirstNewDatabaseSample.BloggingCoNtext**

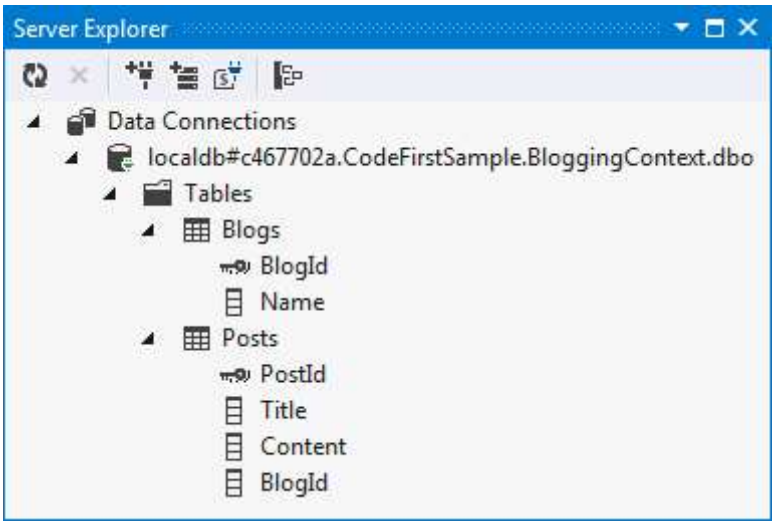
這些只是預設慣例，而且有各種方式可以變更 Code First 使用的資料庫，如需詳細資訊，請參閱 **DbCoNtext 探索模型和資料庫連線主題** 。 您可以在 Visual Studio 中使用伺服器總管連線到此資料庫

- **檢視 - > 伺服器總管**
- 以滑鼠右鍵按一下 **[資料連線]** ，然後選取 [**新增連線...**
- 如果您尚未從 [伺服器總管] 連線到資料庫，則必須先選取 [Microsoft SQL Server] 作為資料來源



- 連線至 LocalDB 或 SQL Express，視您已安裝的專案而定

我們現在可以檢查 Code First 建立的架構。



DbCoNtext 查看我們定義的 DbSet 屬性，找出要包含在模型中的類別。 然後，它會使用預設的 Code First 慣例集來判斷資料表和資料行名稱、判斷資料類型、尋找主鍵等等。 稍後在本逐步解說中，我們將探討如何覆寫這些慣例。

5.處理模型變更

現在是時候對模型進行一些變更，當我們進行這些變更時，我們也需要更新資料庫架構。 若要這樣做，我們將使用稱為「Code First 移轉」或「移轉」的功能。

移轉可讓我們有一組已排序的步驟，說明如何升級或降級資料庫架構。 這些步驟，稱為移轉，包含一些程式碼，描述要套用的變更。

第一個步驟是為 BloggingCoNtext 啟用Code First 移轉。

- **工具 - > 程式庫封裝管理員 - > 封裝管理員 主控台**
- 在套件管理員主控台中執行 **Enable-migrations** 命令
- 新的 Migrations 資料夾已新增至包含兩個專案的專案：
  - **Configuration.cs –** 此檔案包含移轉將用於移轉 BloggingCoNtext 的設定。 我們不需要變更本逐步解說的任何專案，但您可以在這裡指定種子資料、註冊其他資料庫的提供者、變更移轉所產生的命名空間等等。
  - **<timestamp > _InitialCreate.cs –** 這是您的第一個移轉，它代表已經套用至資料庫的變更，使其從空白資料庫轉換成包含 Blogs 和貼文資料表的變更。 雖然我們讓 Code First 自動為我們建立這些資料表，但現在我們已選擇移轉它們已轉換成移轉。 Code First 也記錄在我們的本機資料庫中已套用此移轉。 檔案名上的時間戳記用於排序目的。

  現在讓我們變更模型，將 Url 屬性新增至 Blog 類別：

C#

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

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

- 在 **封裝管理員 主控台中執行 Add-Migration AddUrl** 命令。 [新增移轉] 命令會檢查您上次移轉之後的變更，並使用找到的任何變更來建立新的移轉。 我們可以為移轉提供名稱;在此情況下，我們會呼叫移轉 'AddUrl'。 Scaffold 程式碼指出，我們需要將可保存字串資料的 Url 資料行新增至 dbo。部落格資料表。 如有需要，我們可以編輯 Scaffolded 程式碼，但在此情況下並非必要。

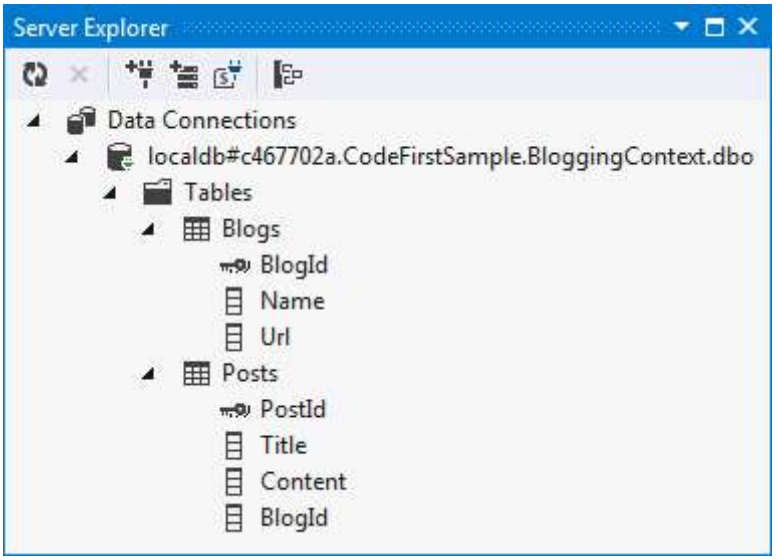
C#

```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddUrl : DbMigration
    {
```

```
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

- 在 **封裝管理員 主控台中執行 Update-Database** 命令。 此命令會將任何擱置的移轉套用至資料庫。 我們的 InitialCreate 移轉已套用，因此移轉只會套用新的 AddUrl 移轉。 秘訣：呼叫 Update-Database 時，您可以使用 **–Verbose** 參數來查看針對資料庫執行的 SQL。

新的 Url 資料行現在會新增至資料庫中的 Blogs 資料表：



6. 資料批註

到目前為止，我們只是讓 EF 使用其預設慣例來探索模型，但有時候我們的類別不會遵循慣例，我們需要能夠執行進一步的設定。 這有兩個選項：我們將在本節中查看資料批註，然後在下一節中查看 Fluent API。

- 讓我們將 User 類別新增至模型

C#

```
public class User
{
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- 我們也需要將集合新增至衍生內容

C#

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

- 如果我們嘗試新增移轉，我們會收到錯誤，指出「 *EntityType 'User' 未定義索引鍵。定義此 EntityType 的索引鍵。* 因為 EF 無法知道 Username 應該是 User 的主鍵。
- 我們現在將使用資料批註，因此我們需要在 Program.cs 頂端新增 using 語句

C#

```
using System.ComponentModel.DataAnnotations;
```

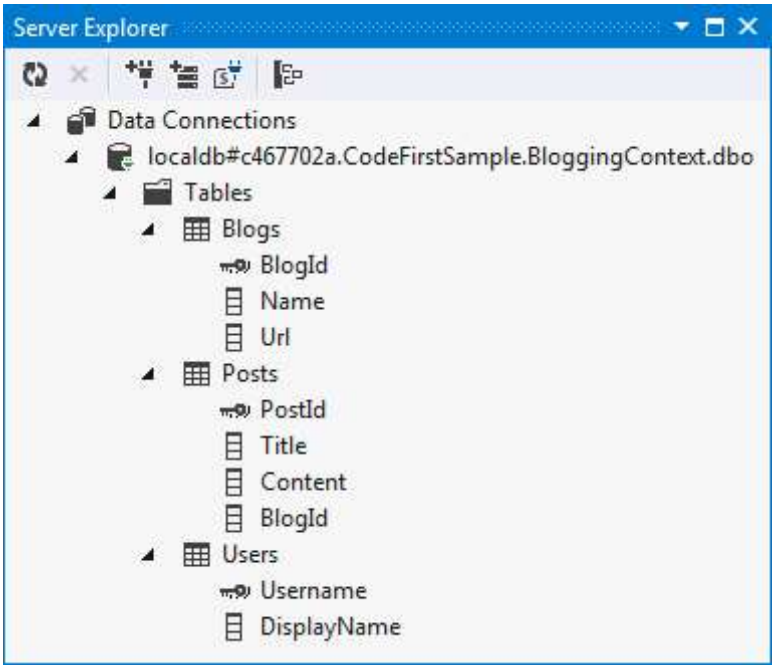
- 現在標注 Username 屬性，以識別其為主鍵

C#

```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- **使用 Add-Migration AddUser** 命令來建立移轉，將這些變更套用至資料庫
- **執行 Update-Database** 命令，將新的移轉套用至資料庫

新的資料表現在已新增至資料庫：



EF 支援的注釋完整清單如下：

- KeyAttribute
- StringLengthAttribute
- MaxLengthAttribute
- ConcurrencyCheckAttribute
- RequiredAttribute
- TimestampAttribute
- ComplexTypeAttribute
- ColumnAttribute
- TableAttribute
- InversePropertyAttribute
- ForeignKeyAttribute
- DatabaseGeneratedAttribute
- NotMappedAttribute

7. Fluent API

在上一節中，我們探討如何使用資料批註來補充或覆寫慣例偵測到的內容。 設定模型的另一種方式是透過 Code First Fluent API。

大部分的模型設定都可以使用簡單的資料批註來完成。 Fluent API 是指定模型組態的更進階方式，其涵蓋資料批註除了無法透過資料批註進行一些更進階的設定之外，還能執行的所有作業。 資料批註和 Fluent API 可以一起使用。

若要存取 Fluent API，請覆寫 DbCoNtext 中的 OnModelCreating 方法。 假設我們想要將 User.DisplayName 儲存在 中的資料行重新命名為display_name。

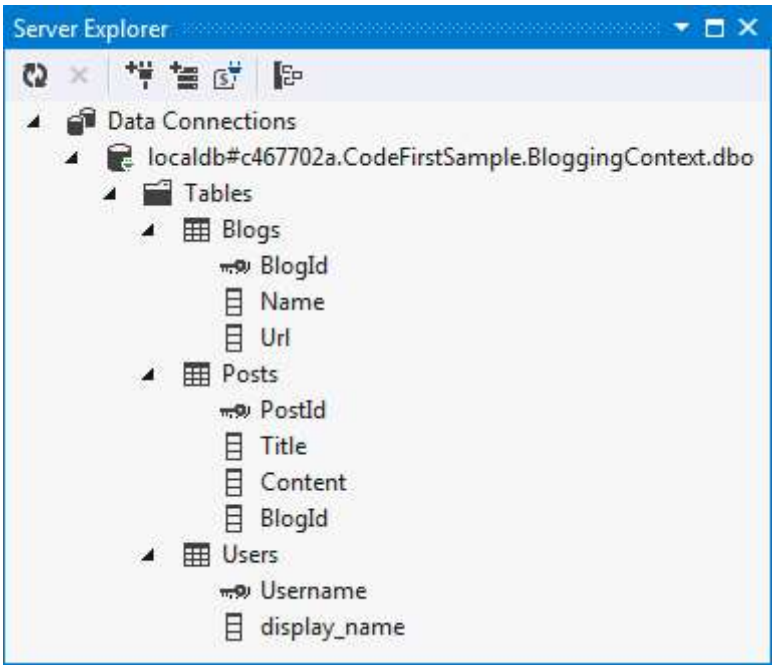
- 使用下列程式碼覆寫 BloggingCoNtext 上的 OnModelCreating 方法

C#

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>()
            .Property(u => u.DisplayName)
            .HasColumnName("display_name");
    }
}
```

- **使用 Add-Migration ChangeDisplayName** 命令來 Scaffold 移轉，將這些變更套用至資料庫。
- **執行 Update-Database** 命令，將新的移轉套用至資料庫。

DisplayName 資料行現在已重新命名為 display_name：



摘要

在本逐步解說中，我們已探討使用新資料庫進行 Code First 開發。 我們使用類別定義模型，然後使用該模型來建立資料庫並儲存和擷取資料。 建立資料庫之後，我們使用Code First 移轉隨著模型演進而變更架構。 我們也瞭解如何使用資料批註和 Fluent API 來設定模型。