Конвенции

Последнее обновление: 26.09.2016

По умолчанию в Entity Framework действует ряд условностей. В частности, таблицы должны называться по имени моделей во множественном числе и т.д. С помощью аннотаций данных и Fluent API мы можем переопределить это поведение. Однако Entity Framework позволяет не только просто переопределить это поведение, но и создать свои собственные условности или конвенции (conventions).

К примеру, возьмем следующие модели:

```
public class Company
{
    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Phone> Phones { get; set; }
    public Company()
    {
        Phones = new List<Phone>();
    }
}

public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }

    public int CompanyId { get; set; }
    public Company Company { get; set; }
}
```

Эти модели содержат свойство Name, которое в нашем случае обязательно должно быть установлено. И, возможно, мы захотим ограничить это свойство по длине строки. Для этого мы могли бы использовать атрибуты `[Required]` и `[MaxLength]`. Однако здесь мы сталкиваемся с практически однотипными настройками. И если моделей десяток и более, то установка атрибутов и последующее возможное изменение (например, изменение предельной длины строки) может занять некоторое время. И в этом случае удобнее использовать конвенции, то есть создать свои условности в отношении свойства Name.

Для этого определим следующий класс контекста данных:

```
public class PhoneContext : DbContext
{
    public PhoneContext() : base("DefaultConnection")
    { }

    public DbSet<Company> Companies { get; set; }
    public DbSet<Phone> Phones { get; set; }
```

```
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Properties<string>().Configure(s => s.HasMaxLength(150));

        modelBuilder.Properties<string>()
            .Where(s => s.Name == "Name")
            .Configure(s => s.HasMaxLength(30).IsRequired());
    }
}
```

С помощью метода `modelBuilder.Properties()` мы можем установить конвенции для определенного рода свойств. Например, для текстовых свойств. Сами условности устанавливаются с помощью метода `Configure()`. Так, выражение:

```
modelBuilder.Properties<string>().Configure(s => s.HasMaxLength(150));
```

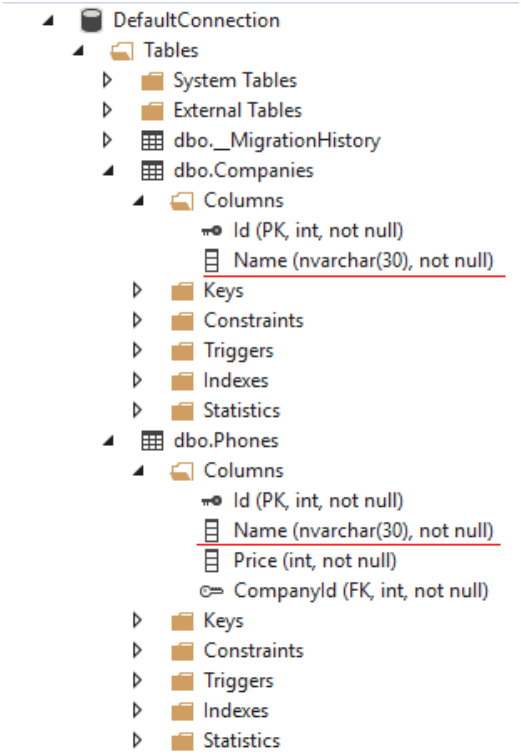
указывает, что все строковые столбцы в базе данных будут иметь максимальную длину в 150 символов.

Если необходимо установить правила для столбцов с определенным условием (например, с названием "Name"), то можно применить выражение `Where`:

```
modelBuilder.Properties<string>().Where(s => s.Name == "Name")
```

Важную роль играет порядок определения конвенций: все последующие конвенции переопределяют предыдущие. Так, в примере выше сначала для всех сток устанавливается максимальная длина в 150 символов, и только потом отдельно для свойства Name устанавливает максимальная длина в 30 символов. Если бы определение конвенций в данном случае поменялось бы местами, то правило для 150 символов переопределило бы правило для Name.

В итоге в базе данных для столбцов будут действовать описанные ограничения:



Подобных конвенций можно насоздавать множество. И чтобы сделать код метода `OnModelCreating()`, мы можем вынести все конвенции в отдельные классы.

Для этого нам надо определить классы, производные от класса **Convention** из пространства имен `System.Data.Entity.ModelConfiguration.Conventions`:

```
using System.Data.Entity.ModelConfiguration.Conventions;

public class PhoneContext : DbContext
{
    public PhoneContext() : base("DefaultConnection")
```

```
    { }

    public DbSet<Company> Companies { get; set; }
    public DbSet<Phone> Phones { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Add(new StringConvention());
        modelBuilder.Conventions.Add(new NameConvention());
    }
}

public class StringConvention : Convention
{
    public StringConvention()
    {
        Properties<string>().Configure(s => s.HasMaxLength(150));
    }
}

public class NameConvention : Convention
{
    public NameConvention()
    {
        Properties<string>()
            .Where(s => s.Name == "Name")
            .Configure(s => s.HasMaxLength(30).IsRequired());
    }
}
```

С помощью метода `modelBuilder.Conventions.Add()` объекты конвенций добавляются в контекст. И опять же здесь играет большое значение порядок определения конвенций: последующие переопределяют предыдущие.

Переопределение существующих конвенций

По умолчанию Entity Framework уже применяет ряд конвенций, в частности, конвенцию IdKeyDiscoveryConvention, которая требует наличия свойства под названием Id или [Название_модели]Id. Но, допустим, мы не хотим называть идентификатор id, а как-то по другому, например, "Key". В этом случае мы можем переопределить существующую конвенцию. Для этого создадим следующий класс:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Data.Entity.Core.Metadata.Edm;

public class CustomKeyConvention : KeyDiscoveryConvention
{
    private const string key = "Key";
    protected override IEnumerable<EdmProperty> MatchKeyProperty
        (EntityType entityType, IEnumerable<EdmProperty> primitiveProperties)
    {
        var matches = primitiveProperties
                            .Where(p => key.Equals(p.Name, StringComparison.OrdinalIgnoreCase));
        if(!matches.Any())
        {
            matches = primitiveProperties
                .Where(p => (entityType.Name + key).Equals(p.Name, StringComparison.OrdinalIgnoreCase));
        }
        if (matches.Count() < 1)
            throw new InvalidOperationException("Свойство с ключом Key или [имя_класса]Key отсутствует");
        return matches;
```

```
    }
}
```

Далее применим эту конвенцию:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Data.Entity.Core.Metadata.Edm;
using System.ComponentModel.DataAnnotations.Schema;

public class PhoneContext : DbContext
{
    public PhoneContext() : base("DefaultConnection")
    { }

    public DbSet<Company> Companies { get; set; }
    public DbSet<Phone> Phones { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Add(new CustomKeyConvention());
        modelBuilder.Conventions.Remove<IdKeyDiscoveryConvention>();

        modelBuilder.Entity<Company>()
            .Property(c => c.Key)
            .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
        modelBuilder.Entity<Phone>()
            .Property(p => p.PhoneKey)
            .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
        }
}
public class Company
{
    public int Key { get; set; }
    public string Name { get; set; }

    public ICollection<Phone> Phones { get; set; }
    public Company()
    {
        Phones = new List<Phone>();
    }
}

public class Phone
{
    public int PhoneKey { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }

    public int CompanyId { get; set; }
    public Company Company { get; set; }
}
```

В результате теперь вместо "Id" в качестве идентификатора глобально будет использоваться "Key".

Назад **Содержание** **Вперед**