

# 第16期-具有构造函数的实体类型

2019年4月27日 08:24

## 使用构造函数的实体类型

从 EF Core 2.1 开始，可定义有参数的构造函数，并让 EF Core 在创建实体的实例时调用此构造函数。

当查询数据时，需要创建类型的实例，EF首先调用默认无参的构造函数，然后设置每个属性的值，但是，如果查找到有参构造函数，而且参数名称和类型与EF映射属性相匹配，则使用有参构造函数实例化这个对象，而且不会通过SET访问器显式设置属性。

```
public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; set; }

    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public Post(int id, string title, DateTime postedOn)
    {
        Id = id;
        Title = title;
        PostedOn = postedOn;
    }

    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

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

需要注意的事项：

- 构造函数未包括的属性，将不会被初始化。例如 Post.Content 属性。
- 参数类型和名称必须匹配，只不过属性遵守 Pascal-cased 命名规则，而参数遵守 camel-cased 命名规则。
- 无法通过构造函数设置导航属性。
- 构造函数可以是公共、私有的或具有任何其他可访问性。

## 只读属性

通过构造函数中设置只读属性，在控制外部访问性上很有意义。

```
public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; private set; }

    public string Name { get; private set; }
    public string Author { get; private set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}
```

可以设置私有字段为主键，然后设置值。

```
public class Blog
{
    private int _id;

    public Blog(string name, string author)
    {
        Name = name;
        Author = author;
    }

    public string Name { get; }
    public string Author { get; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}
```

```
}
```

```
modelBuilder.Entity<Blog>(
    b =>
    {
        b.HasKey("_id");
        b.Property(e => e.Author);
        b.Property(e => e.Name);
    });
```

## 在实体中依赖注入服务

EF Core 还可以将服务注入到实体类型的构造函数。

```
public class Blog
{
    public Blog()
    {
    }

    private Blog(BloggingContext context)
    {
        Context = context;
    }

    private BloggingContext Context { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }
    public string Author { get; set; }

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

    public int PostsCount
        => Posts?.Count
        ?? Context?.Set<Post>().Count(p => Id == EF.Property<int?>
(p, "BlogId"))
        ?? 0;
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

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

}