

# 第33期-预先加载&显式加载&延迟加载

2019年6月5日 05:57

Entity Framework Core 允许使用导航属性来加载相关实体。

## 加载关联数据三种方式

**预先加载：**表示从数据库中加载关联数据，作为初始查询的一部分。

**显式加载：**稍后手动控制时，从数据库中显式加载导航数据。

**延迟加载：**当访问导航属性时，才从数据库中加载导航属性数据。

IQueryable 未使用时不会执行查询行为，生成一个表达式树，需要时执行。

## 默认情况下导航属性不加载任何数据

```
var blog = _context.Blogs.Find(1);
```

## 预先加载

可以使用 Include 方法来指定要包含在查询结果中的关联数据。

```
var blogs = _context.Blogs.Include(blog => blog.Posts).ToList();
```

```
SELECT [blog.Posts].[PostId], [blog.Posts].[BlogId],
FROM [Posts] AS [blog.Posts]
INNER JOIN (
    SELECT [blog0].[BlogId]
    FROM [Blogs] AS [blog0]
) AS [t] ON [blog.Posts].[BlogId] = [t].[BlogId]
ORDER BY [t].[BlogId]
```

## 多个导航属性预先加载

```
var blogs = _context.Blogs.Include(blog => blog.Posts).Include(blog => blog.Owner).ToList();
```

## 多层级导航属性加载

使用 ThenInclude 方法可以依循关系包含多个层级的关联数据。

```
var blogs = _context.Blogs.Include(blog => blog.Posts)
    .ThenInclude(post => post.Author)
    .ThenInclude(author => author.Photo)
    .ToList();
```

可通过链式调用 `ThenInclude` 方法，进一步包含更深级别的关联数据。

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
    .ThenInclude(post => post.Author)
    .ThenInclude(author => author.Photo)
    .ToList();
```

可以将来自多个级别和多个根的关联数据合并到同一查询中。

```
var blogs = context.Blogs
    .Include(blog => blog.Posts)
        .ThenInclude(post => post.Author)
            .ThenInclude(author => author.Photo)
    .Include(blog => blog.Owner)
        .ThenInclude(owner => owner.Photo)
    .ToList();
```

这些 `Include` 在大多数情况下，会在生成 SQL 的 JOIN 联接查询。

## 派生类型上的导航属性加载

可以使用 `Include` 和 `ThenInclude` 包括来自仅在派生类型上定义的导航的相关数据。

使用强制转换方式：

```
context.Peoples.Include(person => ((Student)person).School).ToList()
```

使用 `as` 运算符方式：

```
context.Peoples.Include(person => (person as Student).School).ToList()
```

使用采用类型 `string` 的参数的 `Include` 的重载：

```
context.Peoples.Include("School").ToList()
```

## 忽略导航属性加载

如果更改查询，从而使其不再返回查询以之为开头的实体类型的实例，则会忽略 `include` 运算符。

以下示例中，`include` 运算符基于 `Blog`，但 `Select` 运算符将查询改变为返回匿名类型。在这种情况下，`include` 运算符没有任何效果。

```
var blogs = _context.Blogs
    .Include(blog => blog.Posts)
    .Select(b => new
    {
        b.Name,
        Posts = b.Posts.ToList()
    });
```

默认情况下，当忽略 include 运算符时，EF Core 将记录警告。

```
warn: Microsoft.EntityFrameworkCore.Query[10106]
      The Include operation for navigation '[blog].Posts' is unnecessary and was ignored because the navigation is not reachable in the final query results. See https://go.microsoft.com/fwlink/?linkid=850303 for more information.
```

```
dbContextOptionsBuilder..ConfigureWarnings(warnings =>
warnings.Throw(CoreEventId.IncludeIgnoredWarning)))
```



## 显式加载

```
var blog = _context.Blogs.Single(b => b.BlogId == 1);
```

```
_context.Entry(blog).Collection(b => b.Posts).Load();
_context.Entry(blog).Reference(b => b.Owner).Load();
```

导航属性加载时，可生成远程 SQL 查询条件。

```
var postCount = _context.Entry(blog).Collection(b => b.Posts).Query().Count();
```

```
SELECT TOP(2) [b].[BlogId], [b].[Name], [b].[OwnerId], [b].[Ur1]
FROM [Blogs] AS [b]
WHERE [b].[BlogId] = 1
```

```
SELECT COUNT(*)
FROM [Posts] AS [e]
WHERE [e].[BlogId] = @__get_Item_0
```

```
var goodPosts = _context.Entry(blog).Collection(b => b.Posts).Query().Where(p => p.Rating > 3).ToList();
```

```
SELECT [e].[PostId], [e].[BlogId], [e].[Content], [e].[Title]
FROM [Posts] AS [e]
WHERE ([e].[BlogId] = @__getItem_0) AND ([e].[BlogId] > 3)
```

## 延迟加载

### 使用代理的最简单延迟加载

使用延迟加载的最简单方式是通过安装 `Microsoft.EntityFrameworkCore.Proxies` 包，并通过调用 `UseLazyLoadingProxies` 来启用该包。

```
Install-Package Microsoft.EntityFrameworkCore.Proxies
```

```
optionsBuilder.UseLazyLoadingProxies().UseSqlServer(myConnectionString);
```

EF Core 将为可重写的任何导航属性（必须是 `virtual` 修饰符，且可被继承）的导航属性上启用延迟加载。

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

### 不使用代理的延迟加载

不使用代理进行延迟加载的工作方式是将 `ILazyLoader` 通过构造函数注入到实体中。

```
public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }
```

```

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

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}

```

这样，将不要求实体类型为可继承的类型，也不要求导航属性必须是虚拟的。

ILazyLoader 接口依赖于 Microsoft.EntityFrameworkCore.Abstractions 包。

Install-Package Microsoft.EntityFrameworkCore.Abstractions

**不过，可以将 ILazyLoader.Load 方法以委托的形式注入，这样可以不依赖于任何包。**

```

public class Blog
{
    private ICollection<Post> _posts;

```

```

public Blog()
{
}

private Blog(Action<object, string> lazyLoader)
{
    LazyLoader = lazyLoader;
}

private Action<object, string> LazyLoader { get; set; }

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

public ICollection<Post> Posts
{
    get => LazyLoader.Load(this, ref _posts);
    set => _posts = value;
}
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}

public static class PocoLoadingExtensions
{
    public static TRelated Load<TRelated>(

```

```

        this Action<object, string> loader,
        object entity,
        ref TRelated navigationField,
        [CallerMemberName] string navigationName = null)
        where TRelated : class
    {
        loader?.Invoke(entity, navigationName);

        return navigationField;
    }
}

```

**延迟加载委托的构造函数参数必须名为“lazyLoader”。未来的一个版本中的配置将计划采用另一个名称。**

## 关联数据和序列化

由于 EF Core 具有导航属性和反向导航属性，因此在对象图中可能会产生循环引用。

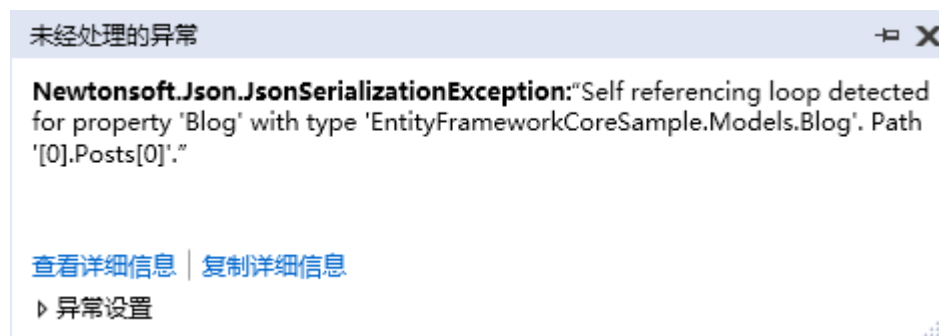
某些序列化框架不允许使用循环引用。

Install-Package Newtonsoft.Json

```

var blogs = _context.Blogs.Include(blog => blog.Posts).ToList();
string jsonString = JsonConvert.SerializeObject(blogs);

```



可设置 JSON.NET 行为，忽略并防止循环引用。

```

JsonSerializerSettings settings = new JsonSerializerSettings();
settings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
string jsonString = JsonConvert.SerializeObject(blogs, settings);

```

另一种方法是使用 [JsonIgnore] 特性修饰其中一个导航属性。

```

[JsonIgnore]
public ICollection<Post> Posts { get; set; }

```

ASP.NET CORE 中可通过扩展方式配置 MVC 默认自带的 JSON 框架设置。

```
services.AddMvc().AddJsonOptions(options => options.SerializerSettings.ReferenceLoopHandling  
= ReferenceLoopHandling.Ignore);
```