

# 第31期-远程数据查询支持

2019年5月27日 23:17

## 查询数据您必须知道

**IEnumerable** 是在服务器本地内存中查询的，不支持远程查询。

**IQueryable** 支持远程查询，延迟加载，除此还提供很多的 **XXXAsync** 方法支持异步远程查询。

**DbSet** 也提供一些方法便于查询。

### [IEnumerable Class](#)

### [Queryable Class](#)

### [EntityFrameworkQueryableExtensions](#)

### [DbSet Class](#)

**IEnumerable** 使用委托查询，**IQueryable** 使用表达式查询，表达式可以生成 SQL 语句。

```
IEnumerable<TSource> Where<TSource>(Func<TSource, bool> predicate);  
IQueryable<TSource> Where<TSource>(Expression<Func<TSource, bool>> predicate);
```

EF Core 应该优先支持远程查询，无法生成对应 SQL 语句时，将无法进行远程查询，这就无法利用数据库的查询优势（索引，查询引擎），将全部数据查询到服务器内存中后，再本地查询，这样性能会受到一些影响，服务器内存过小，还会导致系统奔溃，我们最好是知道那些 LINQ 查询能够转换成何种 SQL 语句，对 EF Core 才能了如指掌，以便于进行性能优化，写出更高质量的 LINQ 语句，EF Core 的性能并不差，可以成倍提高开发效率，不能驾驭 EF CORE 的人才觉得性能差，因为他们不知道每句 LINQ 后，数据库究竟做了何种查询，只是认为生成了一堆性能低下的垃圾代码。

## 打印控制台日志

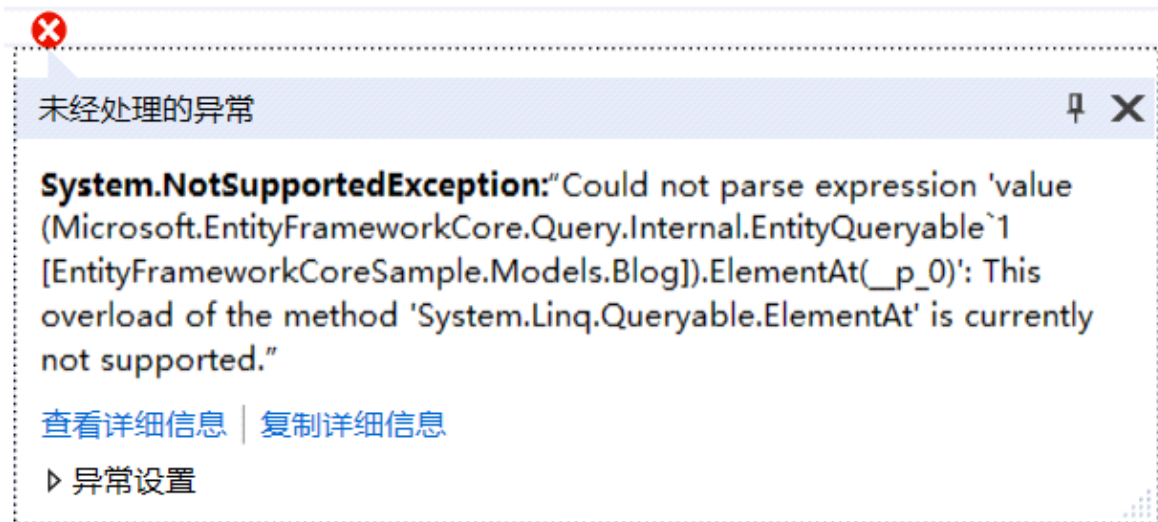
```
Install-Package Microsoft.Extensions.Logging.Console
```

```
new LoggerFactory().AddConsole(LogLevel.Debug)
```

## 元素操作

**ElementAt**、**Find**、**ElementAtOrDefault**、**First**、**FirstOrDefault**、**Last**、**LastOrDefault**、**Single** 和 **SingleOrDefault**

```
var result = _context.Blogs.ElementAt(1);
```



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

```
SELECT TOP(1) [e].[BlogId], [e].[Name], [e].[OwnerId], [e].[Uri]
FROM [Blogs] AS [e]
WHERE [e].[BlogId] = @__get_Item_0
```

主键具有唯一索引，性能会更好。

```
var blog = _context.Blogs.First(b => b.OwnerId == 2);
```

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

并不是所有的方法都翻译成SQL语句，比如：Last 和 LastOrDefault 方法将在本地执行。

```
var blog = _context.Blogs.Last();
```

```
SELECT [b].[BlogId], [b].[Name], [b].[OwnerId], [b].[Uri]
FROM [Blogs] AS [b]
```

```
var blog = _context.Blogs.Single(b => b.OwnerId == 2);
```

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

## 聚合运算

Aggregate、Average、Count、LongCount、Max、Min 和 Sum

```
var result = _context.Blogs.Select(b => b.OwnerId).Aggregate((total, next) =>
total + next);
```

**System.NotImplementedException:**"Remotion.Linq.Clauses.ResultOperators.AggregateResultOperator"

```
var result = _context.Blogs.Average(b=>b.OwnerId);
```

```
SELECT AVG(CAST([b].[OwnerId] AS float))
FROM [Blogs] AS [b]
```

```
var result = _context.Blogs.Count(b=>b.BlogId>1);
```

```
SELECT COUNT(*)
FROM [Blogs] AS [b]
WHERE [b].[BlogId] > 1
```

```
var result = _context.Blogs.Max(b => b.OwnerId);
```

```
SELECT MAX([b].[OwnerId])
FROM [Blogs] AS [b]
```

```
var result = _context.Blogs.Where(b => b.OwnerId > 1).Min(b => b.OwnerId);
```

```
SELECT MIN([b].[OwnerId])
FROM [Blogs] AS [b]
WHERE [b].[OwnerId] > 1
```

```
var result = _context.Blogs.Where(b => b.OwnerId > 1).Sum(b => b.OwnerId);
```

```
SELECT SUM([b].[OwnerId])
FROM [Blogs] AS [b]
WHERE [b].[OwnerId] > 1
```

## 数据筛选

### Where 和 OfType

```
var result = _context.Blogs.Where(b => b.OwnerId > 1 && b.BlogId < 10);
```

```
SELECT [b].[BlogId], [b].[Name], [b].[OwnerId], [b].[Url]
FROM [Blogs] AS [b]
WHERE ([b].[OwnerId] > 1) AND ([b].[BlogId] < 10)
```

|| 翻译成 OR 条件, && 翻译成 AND 条件, 注意延迟查询, 并不会立即执行。

```
var result = _context.Animals.Where(a => a.Age > 3).OfType<Cat>();
```

```
SELECT [a].[ID], [a].[Age], [a].[Discriminator], [a].[CatName]
FROM [Animals] AS [a]
```

```
SELECT [a].[ID], [a].[Age], [a].[Discriminator], [a].[CatName]
FROM [Animals] AS [a]
WHERE ([a].[Discriminator] = N'Cat') AND ([a].[Age] > 3)
```

```
var result = _context.Animals.Cast<Dog>();
```

```
SELECT [a].[ID], [a].[Age], [a].[Discriminator], [a].[CatName], [a].[DogName]
FROM [Animals] AS [a]
WHERE [a].[Discriminator] IN (N'Dog', N'Cat', N'Animal')
```

```
var result = _context.Dogs.Cast<Animal>();
```

```
SELECT [a].[ID], [a].[Age], [a].[Discriminator], [a].[DogName]
FROM [Animals] AS [a]
WHERE [a].[Discriminator] = N'Dog'
```

EF Core 不支持利用 OfType 和 Cast 转换原始类型。

```
var result = _context.Animals.Select(a => a.Age).OfType<string>();
```

#### 未经处理的异常

**System.InvalidOperationException:** "No coercion operator is defined between types 'System.Int32' and 'System.String'."

C# 中可以使用 IS 和 AS 来进行类型转换。IS 判断指定的对象是某个类型或其父类，如果是，就返回 True，否则返回 False，IS 永远不会抛出异常，而 AS 会抛出异常。

```
var result = _context.Animals.Where(a => a is Dog);
```

```
SELECT [a].[ID], [a].[Age], [a].[Discriminator], [a].[CatName], [a].[DogName]
FROM [Animals] AS [a]
WHERE [a].[Discriminator] IN (N'Dog', N'Cat', N'Animal') AND ([a].[Discriminator] = N'Dog')
```

[C#中的 is 和 as 操作符](#)