

# 第60期-EfCore3.0新特性

2019年10月30日 22:34

## 客户端评估受限

在前几个版本中，EF Core 明确查询的哪些部分可以转换为 SQL，并在客户端上执行查询的其余部分。在某些情况下，这种客户端执行是可取的，但在其他许多情况下，这可能会导致低效的查询。

例如，如果 EF Core 2.2 无法转换 Where() 调用中的谓词，则会执行一个不带筛选器的 SQL 语句，从数据库传输所有行，然后在内存中对其进行筛选：

```
var specialCustomers = context.Customers .Where(c => c.Name.StartsWith(n) && IsSpecialCustomer(c));
```

如果数据库包含少量行，那么这可能是可以接受的，但如果数据库包含大量行，则可能导致严重的性能问题甚至应用程序故障。

在 EF Core 3.0 中，当检测到无法转换为 SQL 查询表达式时，它会引发运行时异常，除非显式确认。

```
var specialCustomers =  
    context.Customers  
        .Where(c => c.Name.StartsWith(n))  
        .AsEnumerable() // switches to LINQ to Objects  
        .Where(c => IsSpecialCustomer(c));
```

## 每个 LINQ 查询有一个 SQL 语句

在之前的版本中，在某些场景下我们通常生成多个 SQL 语句，例如：在集合导航属性上转换 Include() 调用，以及转换遵循某些包含子查询的模式查询。尽管在某些场景下这种做法很方便，并且对于 Include() 而言，这甚至有助于避免通过线路发送冗余数据，但实现较为复杂，这会导致一些效率极为低下的行为（N+1 个查询），并且在一些情况下多个查询返回的数据可能不一致。

与客户端评估类似，若 EF Core 3.0 无法将 LINQ 查询转换为单个 SQL 语句，它将引发运行时异常。但我们已使 EF Core 能够借助 JOIN 联接将用于生成多个查询的许多常见模式转换为单个查询。

## EF Core 3.0 支持 C# 8.0 特性

### C# 8.0 中的新增功能

## 异步流

异步查询结果现在使用新的标准 `IAsyncEnumerable<T>` 接口公开，并且可以通过 `await foreach` 使用。

```
var orders = context.Orders.Where(o=>o.Status == OrderStatus.Pending);
```

```
await foreach(var o in orders.AsAsyncEnumerable())  
{
```

```
        Process(o);  
    }
```

## 可为空引用类型

在代码中启用此新功能后，EF Core 将检查引用类型属性的为 Null 性，并将它应用到数据库中的相应列和关系：将按照不可为 Null 的引用类型的属性具有 [Required] 数据注释属性来处理它们。

```
<Nullable>enable</Nullable>  
<LangVersion>8.0</LangVersion>
```

需要将计算机设置为运行 .NET Core，包括 C# 8.0 编译器。Visual Studio 2019 或 .NET Core 3.0 随附 C# 8.0 编译器。

例如，在下面的类中，将把标记为类型 string? 的属性配置为可选，而将 string 配置为必需：

```
public class Customer  
{  
    public int Id { get; set; }  
    public string FirstName { get; set; }  
    public string LastName { get; set; }  
    public string? MiddleName { get; set; }  
}
```

## 截获数据库操作

EF Core 3.0 中的新截获 API 允许提供自定义逻辑，以便在发生低级别数据库操作时作为 EF Core 正常运行的一部分自动调用它们。例如，打开连接、提交事务或执行命令时。

与 EF 6 中的截获功能相似，借助侦听器，你可以在操作发生之前或之后拦截它们。在操作发生前截获它们时，将允许你绕过执行，并从截获逻辑提供备用结果。

```
public class HintCommandInterceptor : DbCommandInterceptor  
{  
    public override InterceptionResult ReaderExecuting(  
        DbCommand command,  
        CommandEventData eventData,  
        InterceptionResult result)  
    {  
        // Manipulate the command text, etc. here...  
        command.CommandText += " OPTION (OPTIMIZE FOR UNKNOWN)";  
        return result;  
    }  
}
```

```
services.AddDbContext(b => b  
    .UseSqlServer(connectionString)  
    .AddInterceptors(new HintCommandInterceptor()));
```

## 数据库视图的反向工程

```
dotnet ef dbcontext scaffold "connectionString" Microsoft.EntityFrameworkCore.SqlServer
```

此工具现在将自动为无键的视图和表类型：

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Names>(entity =>
    {
        entity.HasNoKey();
        entity.ToView("Names");
    });

    modelBuilder.Entity<Things>(entity =>
    {
        entity.HasNoKey();
    });
}
```

## 在 .NET Core 运行 EF 6.3 + 框架

EF 6.3 + 微软将所有库都改为兼容 .NET Standard 2.1 标准了，所以 .NET Core 中可以直接安装 EF 6.3.0 + 框架了，很大程度是为了兼容一些老项目迁移到新项目时的问题，但我们还是推荐 EF Core 框架，必定它才是微软轻量级最彻底的跨平台框架。