.NET Conf China 2023

2023/12/16
09:30 - 18:00

中国・北京

.NET中文社区

中国 · 北京

.NET Conf China 2023

EFCore 8 的新变化
及走向拥抱开源的国产数据库

姚圣伟

个人介绍

https://github.com/JaneConan

姚圣伟

在职开发者，盛派开发者社区主创人员，中国DevOps社区理事会成员天津地区核心组织者，华为云云享专家，MSBuildTour2019北京场分享嘉宾，首届.Net Conf 黑客松北京赛区汗八里小队队长、.NET20周年云原生开发挑战赛获奖团队选手。

热衷于学习和分享可落地的新技术和新文化。目前致力于在信创项目中实践新技术，并立志于将所习得的新技术和思想分享给更多开发者去解决实际问题。

Agenda

一、EF Core 8 的新变化

二、走向拥抱开源的国产数据库

三、NCC 社区孵化国产数据库驱动项目介绍

四、Q & A

EF Core 8 的新变化

1.值对象可使用复杂类型：现在支持“复杂类型”来覆盖为保存多个值而构造的对象，但对象没有用键定义的标识。

2.原始集合的增强

3.JSON列映射的增强

4..NET和EF Core中的 HierarchyId

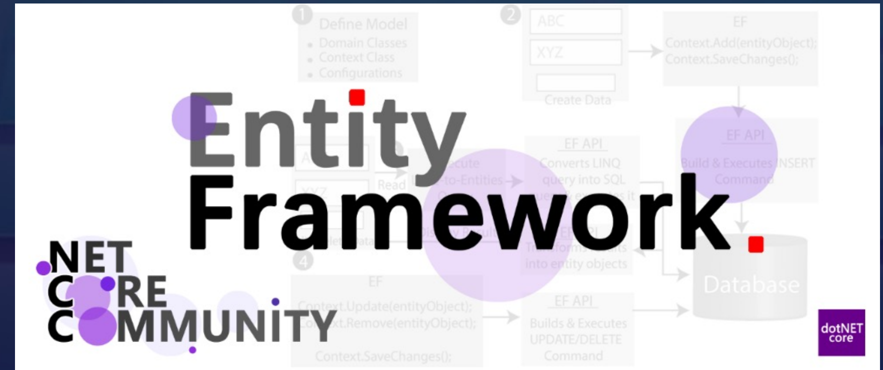
5.对未映射类型的原始 SQL 查询的支持

6.更好的 ExecuteUpdate 和 ExecuteDelete

7.延迟加载的增强

……more

https://learn.microsoft.com/zh-cn/ef/core/what-is-new/ef-core-8.0/whatsnew



https://github.com/dotnet/efcore

Entity Framework (EF) Core

轻量化、可扩展、开源和跨平台版的常用 Entity Framework 数据访问技术。

EF Core 可用作**对象关系映射程序 (O/RM)**

这可以实现以下两点：

•使 .NET 开发人员能够使用 .NET 对象处理数据库

•无需再像通常那样编写大部分数据访问代码(SQL)

EFCore 8 的新变化
——使用复杂类型的值对象

保存到数据库的对象可以分为三大类：
1、非结构化并保存单个值的对象。 例如，int、Guid、string、IPAddress。
这些类型（有点宽泛）称为“基元类型”。

2、为保存多个值而构造的对象，对象的标识由键值定义。 例如：Blog、Post、Customer。
它们称为“实体类型”。

3、为保存多个值而构造的对象，但对象没有用键定义的标识。 例如：Address、Coordinate。

EF8 现在支持“复杂类型”来涵盖此第三种类型的对象。 复杂类型对象：

- 未按键值标识或跟踪。
- 必须定义为实体类型的一部分。 （换句话说，你不能有复杂类型的 DbSet）
- 可以是 .NET 值类型或引用类型。
- 实例可以由多个属性共享。

.NET中文社区

.NET Conf China 2023
中国 • 北京

.NET Conf 2023

Document vs Relational Databases

- Document Databases are good at:
  - Point read of a document
  - Scale out
  - And more...
- Relational Databases are good at:
  - General-purpose queries
  - Transactions and consistency
  - And more...

- Entity Framework Core:
  - Leverage common patterns (LINQ, unit-of-work)
  - Leverage the specific power of each database system
  - Promote best practices

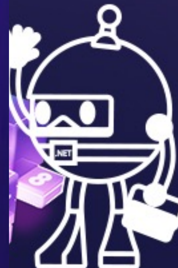
.NET中文社区

.NET Conf China 2023
中国 • 北京

Customer Table

Id	Name	Visits	MemberSince	Details
de662b02-adb4-4aaf-3eaa-08dbe1e40d4b	Willow	null	2008-05-20	{"Notes":null,"Addresses":[{"City":"Walpole St Peter","Country":"UK","IsPrimary":true,"Postcode":"PE157ZN","Street":"Baking Gate"}],"PhoneNumbers":[{"CountryCode":1,"IsPrimary":false,"Number":"(555) 234-2333"},{"CountryCode":44,"IsPrimary":true,"Number":"(1234) 1234-333"}]}
7dc44e57-5087-400c-3eab-08dbe1e40d4b	Toast	["2014-05-01T12:00:00","2014-06-01T12:30:00","2018-01-01T12:15:00"]	2018-01-21	{"Notes":null,"Addresses":[{"City":"Walpole St Andrews","Country":"UK","IsPrimary":false,"Postcode":"PE157AN","Street":"The Old Pub"},{"City":"Walpole St Peter","Country":"UK","IsPrimary":true,"Postcode":"PE 59ZN","Street":"Peacock Ma [{"CountryCode":44,"IsPrimary":true,"Number":"(1234) 1234-989"},{"CountryCode" (123) 444-3543"}]}
db146f30-c372-47ef-3eac-08dbe1e40d4b	Baxter	["2014-05-01T12:00:00","2014-06-01T12:30:00","2018-01-01T12:15:00"]	2005-12-24	{"Notes":["Consultant for PCC"],"Addresses":[{"City":"Redmond","Country":"US","IsPrimary":true,"Post ode":"98052","Street":"Ed Mountain","Country":"US","IsPrimary":false,"Postcode":"9 051","Street":"Mountain [{"CountryCode":1,"IsPrimary":true,"Number":"(123) 555-5 33"},{"CountryCode":44 (1234) 1234-133"}]}

```json
{
  "Notes": null,
  "Region": "Europe",
  "Addresses": [
    {
      "City": "Walpole St Peter",
      "Country": "UK",
      "IsPrimary": true,
      "Postcode": "PE157ZN",
      "Street": "Baking Gate"
    }
  ],
  "PhoneNumbers": [
    {
      "CountryCode": 1,
      "IsPrimary": false,
      "Number": "(555) 234-2333"
    },
    {
      "CountryCode": 44,
      "IsPrimary": true,
      "Number": "(1234) 1234-333"
    }
  ]
}
```



```
{
    "Id": "ea3b5aeb-05ae-40dd-a285-08dbe508841a",
    "Discriminator": "Customer",
    "MemberSince": "2008-05-20",
    "Name": "Willow",
    "Visits": null,
    "id": "Customer|ea3b5aeb-05ae-40dd-a285-08dbe508841a",
    "Details": {
        "Notes": null,
        "Region": "Europe",
        "Addresses": [
            {
                "City": "Walpole St Peter",
                "Country": "UK",
                "IsPrimary": true,
                "Postcode": "PE157ZN",
                "Street": "Baking Gate"
            }
        ],
        "PhoneNumbers": [
            {
                "CountryCode": 1,
                "IsPrimary": false,
                "Number": "(555) 234-2333"
            },
            {
                "CountryCode": 44,
                "IsPrimary": true,
                "Number": "(1234) 1234-333"
            }
        ]
    },
    "_rid": "vfYrAJ4FyrkBAAAAAAAAAA==",
    "_self": "dbs/vfYrAA==/colls/vfYrAJ4Fyrk=/docs/vfYrAJ4FyrkBAAAAAAAAAA==/",
    "_etag": "\"00000000-0000-0000-16f1-61c2c5c201da\"",
    "_attachments": "attachments/",
    "_ts": 1699962894
}
```


如何构造?

如何CRUD?

……

可变性 跟踪
查询性能
值的操作

简单示例 如何构造?

例如，想一想 `Address` 类型：

C#

```
public class Address
{
    public required string Line1 { g
    public string? Line2 { get; set;
    public required string City { ge
    public required string Country {
    public required string PostCode
}
```

然后，`Address` 用于一个简单的客户/订单模型中的三个位置：

C# 复制

```
public class Customer
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required Address Address { get; set; }
    public List<Order> Orders { get; } = new();
}

public class Order
{
    public int Id { get; set; }
    public required string Contents { get; set; }
    public required Address ShippingAddress { get; set; }
    public required Address BillingAddress { get; set; }
    public Customer Customer { get; set; } = null!;
}
```

让我们使用其地址创建并保存客户：

C#

```csharp
var customer = new Customer
{
    Name = "Willow",
    Address = new() { Line1 = "Barking Gate", City = "Walpole St Peter",
Country = "UK", PostCode = "PE14 7AV" }
};

context.Add(customer);
await context.SaveChangesAsync();
```

```sql
INSERT INTO [Customers] ([Name], [Address_City], [Address_Country],
[Address_Line1], [Address_Line2], [Address_PostCode])
OUTPUT INSERTED.[Id]
VALUES (@p0, @p1, @p2, @p3, @p4, @p5);
```

请注意，复杂类型不会有自己的表。 相反，它们会被内联保存到 `Customers` 表的列。 这与从属类型的表共享行为相匹配。

ⓘ **备注**

我们并不打算允许将复杂类型映射到其自己的表。 但是，在将来的版本中，我们确实计划允许将复杂类型另存为单列的 JSON 文档。 如果这对你很重要，请为**问题 #31252** 投票。

https://github.com/dotnet/efcore/issues/31252

.NET中文社区

.NET Conf China 2023
中国 • 北京

EFCore 8 的新变化
使用复杂类型的值对象

现在假设我们要将订单寄送给客户，并使用客户的地址作为默认的计费地址和发货地址。执行此操作的自然方法是将 `Address` 对象从 `Customer` 复制到 `Order`。 例如：

C#

```
customer.Orders.Add(
    new Order { Contents = "Tesco Tasty Treats", BillingAddress =
customer.Address, ShippingAddress = customer.Address, });

await context.SaveChangesAsync();
```

对于复杂类型，此过程会按预期工作，地址会插入 `Orders` 表中：

SQL

```
INSERT INTO [Orders] ([Contents], [CustomerId],
    [BillingAddress_City], [BillingAddress_Country], [BillingAddress_Line1],
[BillingAddress_Line2], [BillingAddress_PostCode],
    [ShippingAddress_City], [ShippingAddress_Country],
[ShippingAddress_Line1], [ShippingAddress_Line2],
[ShippingAddress_PostCode])
OUTPUT INSERTED.[Id]
VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7, @p8, @p9, @p10, @p11);
```

看到这里，你可能会说，
“从属类型也可以做到这一点！”

但是，从属类型的“实体类型”语义很快就会成为障碍。

EFCore 8 的新变化
使用复杂类型的值对象

```
warn: 8/20/2023 12:48:01.678
CoreEventId.DuplicateDependentEntityTypeInstanceWarning[10001]
(Microsoft.EntityFrameworkCore.Update)
      The same entity is being tracked as different entity types
'Order.BillingAddress#Address' and 'Customer.Address#Address' with defining
navigations. If a property value changes, it will result in two store
changes, which might not be the desired outcome.
warn: 8/20/2023 12:48:01.687
CoreEventId.DuplicateDependentEntityTypeInstanceWarning[10001]
(Microsoft.EntityFrameworkCore.Update)
      The same entity is being tracked as different entity types
'Order.ShippingAddress#Address' and 'Customer.Address#Address' with defining
navigations. If a property value changes, it will result in two store
changes, which might not be the desired outcome.
warn: 8/20/2023 12:48:01.687
CoreEventId.DuplicateDependentEntityTypeInstanceWarning[10001]
(Microsoft.EntityFrameworkCore.Update)
      The same entity is being tracked as different entity types
'Order.ShippingAddress#Address' and 'Order.BillingAddress#Address' with
defining navigations. If a property value changes, it will result in two
store changes, which might not be the desired outcome.
fail: 8/20/2023 12:48:01.709 CoreEventId.SaveChangesFailed[10000]
(Microsoft.EntityFrameworkCore.Update)
      An exception occurred in the database while saving changes for context
type 'NewInEfCore8.ComplexTypesSample+CustomerContext'.
      System.InvalidOperationException: Cannot save instance of
'Order.ShippingAddress#Address' because it is an owned entity without any
reference to its owner. Owned entities can only be saved as part of an
aggregate also including the owner entity.
         at
Microsoft.EntityFrameworkCore.ChangeTracking.Internal.InternalEntityEntry.Pr
epareToSave()
```

问题：
使用从属类型运行上述代码会导致大量警告，
然后出现错误：
这是因为 Address 实体类型的单个实例
（**具有相同的隐藏键值 key id**）
被用于 *三个不同* 的实体实例。

解决方案：
EFCore8 允许在复杂属性之间共享同一实例，
因此使用复杂类型时代码会按预期工作。

复杂类型的配置

必须使用映射属性或通过调用 OnModelCreating 中的 ComplexProperty API 来在模型中配置复杂类型。 复杂类型不按约定发现。

例如，可以使用 ComplexTypeAttribute 配置 `Address` 类型：

C# 复制

```
[ComplexType]
public class Address
{
    public required string Line1 { get; set; }
    public string? Line2 { get; set; }
    public required string City { get; set; }
    public required string Country { get; set; }
    public required string PostCode { get; set; }
}
```

或在 `OnModelCreating` 中：

C# 复制

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Customer>()
        .ComplexProperty(e => e.Address);

    modelBuilder.Entity<Order>(b =>
    {
        b.ComplexProperty(e => e.BillingAddress);
        b.ComplexProperty(e => e.ShippingAddress);
    });
}
```

.NET中文社区

造成的可变性问题

在上面的示例中，我们最后在三个位置中使用了相同的 `Address` 实例。 这是被允许的，并且不会在使用复杂类型时对 EF Core 造成任何问题。 但是，共享同一引用类型的实例意味着，如果修改了实例上的属性值，则该更改将反映在所有三个使用中。 例如，接着上面的例子，让我们更改客户地址的 `Line1` 并保存更改：

C# 复制

```csharp
customer.Address.Line1 = "Peacock Lodge";
await context.SaveChangesAsync();
```

这会导致使用 SQL Server 时对数据库进行以下更新：

SQL 复制

```sql
UPDATE [Customers] SET [Address_Line1] = @p0
OUTPUT 1
WHERE [Id] = @p1;
UPDATE [Orders] SET [BillingAddress_Line1] = @p2, [ShippingAddress_Line1] = @p3
OUTPUT 1
WHERE [Id] = @p4;
```

请注意，三个 `Line1` 列都已更改，因为它们都共享同一实例。 这通常不是我们想要的。

问题：
不想统一被改变

怎么办？

造成的可变性问题

处理此类问题的一个好办法是 **使类型不可变** 。

事实上，当某个类型很适合成为复杂类型时，这种不可变性通常是很自然的。

例如，提供一个 ***复杂且新*** 的 Address 对象，**比仅仅改变国家/地区，并让其余部分保持不变**，更说得通。

引用和值类型都可设为不可变。

我们一起来看一些示例。

不可变 类 class

我们在上面的示例中使用了一个简单、可变的 `class`。 为了防止上述意外突变问题，我们可以使类不可变。 例如：

C#

```csharp
public class Address
{
    public Address(string line1, string? line2, string city, string country,
string postCode)
    {
        Line1 = line1;
        Line2 = line2;
        City = city;
        Country = country;
        PostCode = postCode;
    }

    public string Line1 { get; }
    public string? Line2 { get; }
    public string City { get; }
    public string Country { get; }
    public string PostCode { get; }
}
```

C#　　复制

```csharp
[ComplexType]
public class Address
{
    public required string Line1 { get; set; }
    public string? Line2 { get; set; }
    public required string City { get; set; }
    public required string Country { get; set; }
    public required string PostCode { get; set; }
}
```

只跟踪值变更的列

现在无法更改现有地址上的 `Line1` 值。 相反，我们需要创建一个具有已更改值的新实例。 例如：

C#

```
var currentAddress = customer.Address;
customer.Address = new Address(
    "Peacock Lodge", currentAddress.Line2, currentAddress.City,
currentAddress.Country, currentAddress.PostCode);

await context.SaveChangesAsync();
```

这次对 `SaveChangesAsync` 的调用仅更新了客户地址：

SQL

```
UPDATE [Customers] SET [Address_Line1] = @p0
OUTPUT 1
WHERE [Id] = @p1;
```

请注意，即使 Address 对象不可变且整个对象已更改，EF 仍会跟踪对各个属性的更改，因此只会更新具有更改值的列。

问题：
太复杂了！

怎么办？

不可变 记录 record

C# 9 推出了记录类型，它使得创建和使用不可变对象变得更加容易。 例如，可以将 `Address` 对象设为记录类型：

C# 复制

```csharp
public record Address
{
    public Address(string line1, string? line2, string city, string country, string
    {
        Line1 = line1;
        Line2 = line2;
        City = city;
        Country = country;
        PostCode = postCode;
    }

    public string Line1 { get; init; }
    public string? Line2 { get; init; }
    public string City { get; init; }
    public string Country { get; init; }
    public string PostCode { get; init; }
}
```

替换可变对象和调用 `SaveChanges` 现在所需的代码更少了：

C#

```csharp
customer.Address = customer.Address with { Line1 = "Peacock Lodge" };

await context.SaveChangesAsync();
```

不可变 结构记录 record struct

…同理……

嵌套的复杂类型

一个复杂类型可以包含其他复杂类型的属性。 例如，让我们将上述的 `Address` 复杂类型与 `PhoneNumber` 复杂类型一起使用，并将它们嵌套在另一个复杂类型中：

C#

```
public record Address(string Line1, string? Line2, string City, string
Country, string PostCode);

public record PhoneNumber(int CountryCode, long Number);

public record Contact
{
    public required Address Address { get; init; }
    public required PhoneNumber HomePhone { get; init; }
    public required PhoneNumber WorkPhone { get; init; }
    public required PhoneNumber MobilePhone { get; init; }
}
```

我们在此处使用不可变记录，因为这些记录非常适合复杂类型的语义，但复杂类型的嵌套可以使用任何 .NET 类型来做到。

ⓘ **备注**

我们没有对 `Contact` 类型使用主构造函数，因为 EF Core 尚不支持复杂类型值的构造函数注入。 如果这对你很重要，请为**问题** #31621 投票。

我们会将 `Contact` 添加为 `Customer` 的属性：

C#

```
public class Customer
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required Contact Contact { get; set; }
    public List<Order> Orders { get; } = new();
}
```

并将 `PhoneNumber` 添加为 `Order` 的属性：

C#

```
public class Order
{
    public int Id { get; set; }
    public required string Contents { get; set; }
    public required PhoneNumber ContactPhone { get; set; }
    public required Address ShippingAddress { get; set; }
    public required Address BillingAddress { get; set; }
    public Customer Customer { get; set; } = null!;
}
```

嵌套的复杂类型

可以使用 ComplexTypeAttribute 再次配置嵌套的复杂类型：

C#

```
[ComplexType]
public record Address(string Line1, string? Line2, string City, string Country, string PostCode);

[ComplexType]
public record PhoneNumber(int CountryCode, long Number);

[ComplexType]
public record Contact
{
    public required Address Address { get; init; }
    public required PhoneNumber HomePhone { get; init; }
    public required PhoneNumber WorkPhone { get; init; }
    public PhoneNumrequired ber MobilePhone { get; init; }
}
```

或在 `OnModelCreating` 中：

C# 复制

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Customer>(
        b =>
        {
            b.ComplexProperty(
                e => e.Contact,
                b =>
                {
                    b.ComplexProperty(e => e.Address);
                    b.ComplexProperty(e => e.HomePhone);
                    b.ComplexProperty(e => e.WorkPhone);
                    b.ComplexProperty(e => e.MobilePhone);
                });
        });

    modelBuilder.Entity<Order>(
        b =>
        {
            b.ComplexProperty(e => e.ContactPhone);
            b.ComplexProperty(e => e.BillingAddress);
            b.ComplexProperty(e => e.ShippingAddress);
        });
}
```

查询

实体类型上复杂类型的属性被视为和实体类型的任何其他非导航属性同等。这意味着，加载实体类型时，会始终加载它们。这也适用于任何嵌套的复杂类型属性。例如，某个客户的查询：

C#

```
var customer = await context.Customers.FirstAsync(e => e.Id == customerId);
```

在使用 SQL Server 时被转换为以下 SQL：

SQL

```
SELECT TOP(1) [c].[Id], [c].[Name], [c].[Contact_Address_City], [c].
[Contact_Address_Country],
    [c].[Contact_Address_Line1], [c].[Contact_Address_Line2], [c].
[Contact_Address_PostCode],
    [c].[Contact_HomePhone_CountryCode], [c].[Contact_HomePhone_Number],
[c].[Contact_MobilePhone_CountryCode],
    [c].[Contact_MobilePhone_Number], [c].[Contact_WorkPhone_CountryCode],
[c].[Contact_WorkPhone_Number]
FROM [Customers] AS [c]
WHERE [c].[Id] = @__customerId_0
```

请注意此 SQL 中的两件事：

- 返回了所有内容以填充客户以及所有嵌套的 Contact、Address、PhoneNumber 复杂类型。
- 所有复杂类型值都存储为实体类型的表中的列。复杂类型永远不会映射到单独的表。

投影 Projections

可以从查询投影复杂类型。 例如，仅从订单中选择发货地址：

C#

```
var shippingAddress = await context.Orders
    .Where(e => e.Id == orderId)
    .Select(e => e.ShippingAddress)
    .SingleAsync();
```

使用 SQL Server 时，这会转换为以下内容：

SQL

```
SELECT TOP(2) [o].[ShippingAddress_City], [o].[ShippingAddress_Country],
[o].[ShippingAddress_Line1],
    [o].[ShippingAddress_Line2], [o].[ShippingAddress_PostCode]
FROM [Orders] AS [o]
WHERE [o].[Id] = @__orderId_0
```

请注意，无法跟踪复杂类型的投影，因为复杂类型对象没有用于跟踪的标识。

EFCore 8 的新变化
使用复杂类型的值对象

在谓词中使用

复杂类型的成员可用于谓词。 例如，查找前往特定城市的所有订单：

C#

```csharp
var city = "Walpole St Peter";
var walpoleOrders = await context.Orders.Where(e => e.ShippingAddress.City
== city).ToListAsync();
```

这会转换为 SQL Server 上的以下 SQL：

SQL

```sql
SELECT [o].[Id], [o].[Contents], [o].[CustomerId], [o].
[BillingAddress_City], [o].[BillingAddress_Country],
    [o].[BillingAddress_Line1], [o].[BillingAddress_Line2], [o].
[BillingAddress_PostCode],
    [o].[ContactPhone_CountryCode], [o].[ContactPhone_Number], [o].
[ShippingAddress_City],
    [o].[ShippingAddress_Country], [o].[ShippingAddress_Line1], [o].
[ShippingAddress_Line2],
    [o].[ShippingAddress_PostCode]
FROM [Orders] AS [o]
WHERE [o].[ShippingAddress_City] = @__city_0
```

EFCore 8 的新变化
使用复杂类型的值对象

在谓词中使用

还可以在谓词中使用完整的复杂类型实例。 例如，查找具有给定电话号码的所有客户：

C#

```csharp
var phoneNumber = new PhoneNumber(44, 7777555777);
var customersWithNumber = await context.Customers
    .Where(
        e => e.Contact.MobilePhone == phoneNumber
             || e.Contact.WorkPhone == phoneNumber
             || e.Contact.HomePhone == phoneNumber)
    .ToListAsync();
```

使用 SQL Server 时，这会转换为以下 SQL：

SQL

```sql
SELECT [c].[Id], [c].[Name], [c].[Contact_Address_City], [c].
[Contact_Address_Country], [c].[Contact_Address_Line1],
     [c].[Contact_Address_Line2], [c].[Contact_Address_PostCode], [c].
[Contact_HomePhone_CountryCode],
     [c].[Contact_HomePhone_Number], [c].[Contact_MobilePhone_CountryCode],
[c].[Contact_MobilePhone_Number],
     [c].[Contact_WorkPhone_CountryCode], [c].[Contact_WorkPhone_Number]
FROM [Customers] AS [c]
WHERE ([c].[Contact_MobilePhone_CountryCode] =
@__entity_equality_phoneNumber_0_CountryCode
    AND [c].[Contact_MobilePhone_Number] =
@__entity_equality_phoneNumber_0_Number)
OR ([c].[Contact_WorkPhone_CountryCode] =
@__entity_equality_phoneNumber_0_CountryCode
    AND [c].[Contact_WorkPhone_Number] =
@__entity_equality_phoneNumber_0_Number)
OR ([c].[Contact_HomePhone_CountryCode] =
@__entity_equality_phoneNumber_0_CountryCode
    AND [c].[Contact_HomePhone_Number] =
@__entity_equality_phoneNumber_0_Number)
```

请注意，通过扩展复杂类型的每个成员，执行了相等性。

这与没有标识键的复杂类型保持一致，

因此**当且仅当成员全部相等**时，一个复杂类型实例**完全等于**另一个复杂类型实例。

这也符合 .NET 为记录类型定义的相等性

复杂类型值的操作

EF8 提供了跟踪信息（例如复杂类型的当前值和原始值），以及属性值是否已修改的信息。 API 复杂类型是已用于实体类型的更改跟踪 API 的扩展。

EntityEntry 的 `ComplexProperty` 方法返回整个复杂对象的条目。 例如，要获取 `Order.BillingAddress` 的当前值：

C# 复制

```
var billingAddress = context.Entry(order)
    .ComplexProperty(e => e.BillingAddress)
    .CurrentValue;
```

可以添加对 `Property` 的调用以访问复杂类型的属性。 例如，要只获取计费邮政编码的当前值：

C# 复制

```
var postCode = context.Entry(order)
    .ComplexProperty(e => e.BillingAddress)
    .Property(e => e.PostCode)
    .CurrentValue;
```

使用对 `ComplexProperty` 的嵌套调用访问嵌套的复杂类型。 例如，要在 `Customer` 上从 `Contact` 的嵌套的 `Address` 中获取城市：

C# 复制

```
var currentCity = context.Entry(customer)
    .ComplexProperty(e => e.Contact)
    .ComplexProperty(e => e.Address)
    .Property(e => e.City)
    .CurrentValue;
```

还有其他方法可用于读取和更改状态。 例如，PropertyEntry.IsModified 可用于将复杂类型的属性设置为已修改：

C# 复制

```
context.Entry(customer)
    .ComplexProperty(e => e.Contact)
    .ComplexProperty(e => e.Address)
    .Property(e => e.PostCode)
    .IsModified = true;
```

当前限制

复杂类型代表着跨 EF 堆栈的重大投资。 我们无法在此版本中做到尽善尽美，但我们计划在未来的版本中补上一些缺口。 如果修复其中的任何限制对你很重要，请务必对相应的 GitHub 问题进行投票 (👍)。

EF8 中的复杂类型限制包括：

- 支持复杂类型的集合。 （问题 #31237）
- 允许复杂类型属性为 null。 （问题 #31376）
- 将复杂类型属性映射到 JSON 列。 （问题 #31252）
- 复杂类型的构造函数注入。 （问题 #31621）
- 为复杂类型添加种子数据支持。 （问题 #31254）
- 映射 Cosmos 提供程序的复杂类型属性。 （问题 #31253）
- 为内存中数据库实现复杂类型。 （问题 #31464）

https://github.com/dotnet/efcore/issues/

.NET Conf 2023

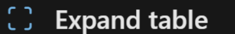
Summary

- Document and relational databases both have strengths
  - Competitive environment is pushing both to do better
- EF8:
  - Expands on previous versions with common patterns for both
  - Uses
    - First-class JSON support in the database
    - With the rich EF model
  - To enable powerful patterns:
    - Primitive collections
    - Collections of entities
    - Parameter and inline collections (Improved "Contains" queries)

目标 Framework

EF Core 8 面向 .NET 8。 面向旧版 .NET、.NET Core 和 .NET Framework 版本的应用程序需要更新到面向 .NET 8。

总结



中断性变更	影响
LINQ 查询中的 Contains 可能会在较旧的 SQL Server 版本上停止工作	高
JSON 中的枚举默认存储为整数而不是字符串	高
SQL Server date 和 time 现已搭建到 .NET DateOnly 和 TimeOnly	中
具有数据库生成值的布尔列不再搭建基架为可为 null	中
SQLite Math 方法现在转换为 SQL	低
ITypeBase 在某些 API 中替换 IEntityType	低
ValueGenerator 表达式必须使用公共 API	低
ExcludeFromMigrations 不再排除 TPC 层次结构中的其他表	低
非阴影整数键保留到 Cosmos 文档	低

可参考和访问以下链接：

主要的文档：aka.ms/efdocs
What’s New in EF Core 8：aka.ms/ef8-new
What’s New in EF Core 7：aka.ms/ef7-new
EF Core 的 issue 和功能请求：github.com/dotnet/efcore/issues
Entity Framework 路线图：aka.ms/efroadmap
每两周一次的更新：aka.ms/ef-news

The .NET Data Community Standup
EF Core Community Standup 播放列表：aka.ms/efstandups

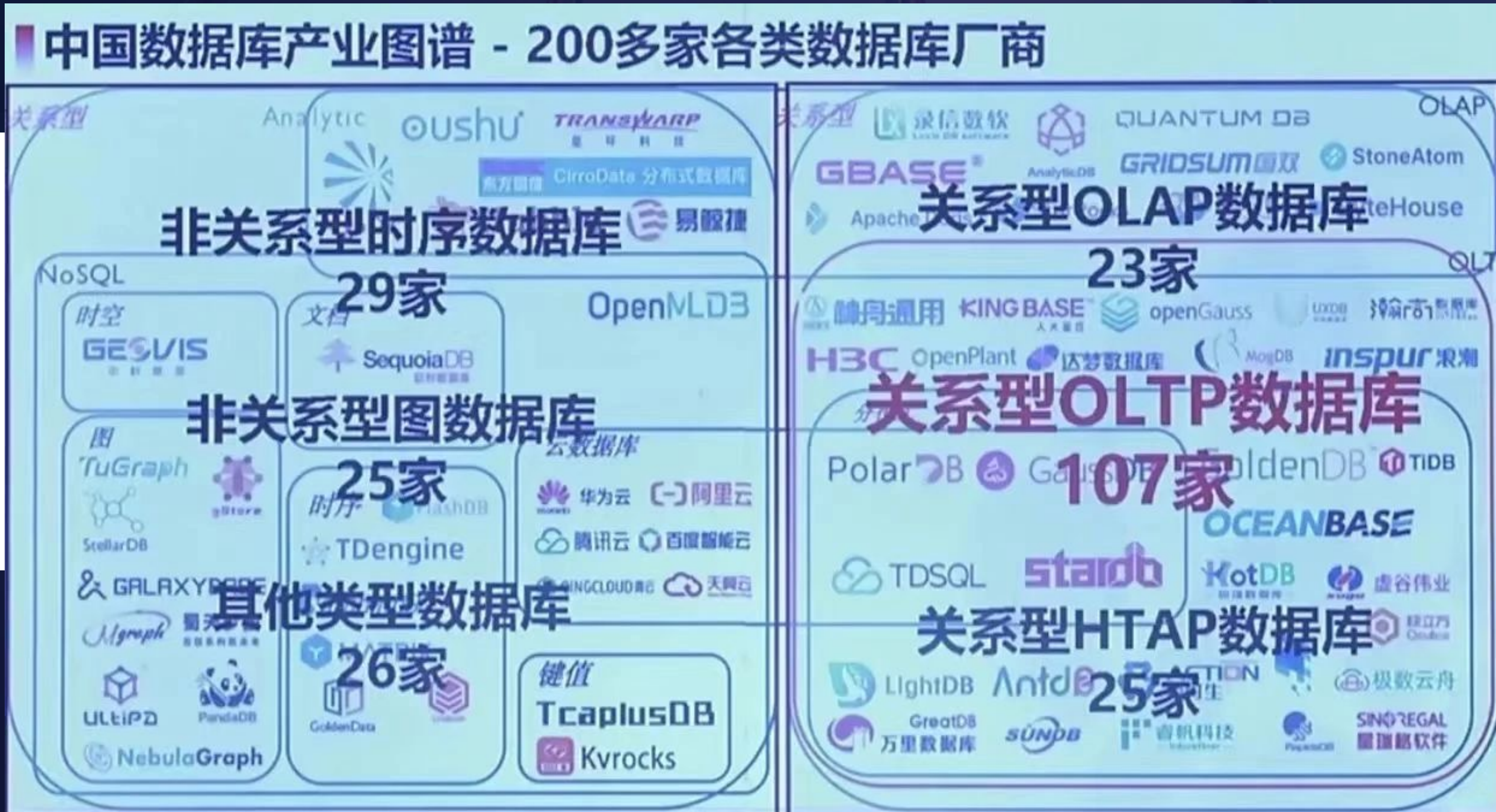
.NET 数据访问团队现在每星期三太平洋时间上午10点、东部时间下午1点或 UTC 时间18:00进行直播。
加入直播学习并询问有关 .NET 数据相关的问题。

走向拥抱开源的 国产数据计算底座

基础软硬件作为数字经济发展底座，是必然趋势。
当前，我国以CPU、GPU、服务器等为代表的硬件基础设施
以及以操作系统、**数据库**、中间件等为代表的基础软件均已实现从“可用”向“好用”的转变，将为我国数字经济的长期持续发展提供有力支撑。

走向拥抱开源的国产数据库

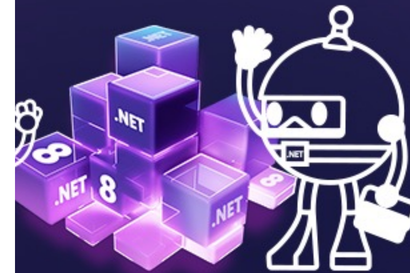
人大金仓
TDSQL
TiDB
GBase
达梦
AnalyticDB
OceanBase
openGauss
PolarDB



走向拥抱开源的国产数据库

OLTP数据库溯源

- PostgreSQL
  - 华为 Open Gauss
    - 海量 VastBase
    - 人大金仓 KingBase
    - 云和恩墨 MogDB
    - 南大通用 Gbase 8c
    - 神舟通用 神通V7
    - 中国移动 磐维数据库
    - 超图软件 Yukon（禹贡）
    - 超聚变
  - PGXC
    - 华为 GaussDB
    - 腾讯 TBASE
    - 亚信 AntDB
  - PG社区版
    - 瀚高 ivorySQL
    - 优炫 UXDB
    - 阿里 Porlar DB-PG
    - 恒生电子 LightDB
    - 浪潮 kaiwuDB
    - 中国电信 TeleDB PG
- 自研
  - Oracle兼容
    - 达梦 DM8
    - 浪潮 K-DB
    - 科蓝软件 SunDB
  - RocksDB分布式
    - 平凯星辰 TiDB
    - 奥星贝斯 OceanBase
  - 巨杉数据库 SequoiaDB
- MySQL
  - 阿里RDS MySQL
  - 腾讯TDSQL
  - 万里开源 GreatSQL
  - 中兴通讯 GoldenDB
  - 热璞 HotDB
  - 星环 KunDB
  - 百度 GaiaDB
  - 阿里 PolarDB-mysql
  - 中国电信 TeleDB-mysql
  - 石原子科技 StoneDB
- Informix
  - 南大通用 Gbase 8s
  - 星瑞格SinoDB
  - 华胜 xigemaDB



.NET中文社区

.NET Conf China 2023
中国 • 北京

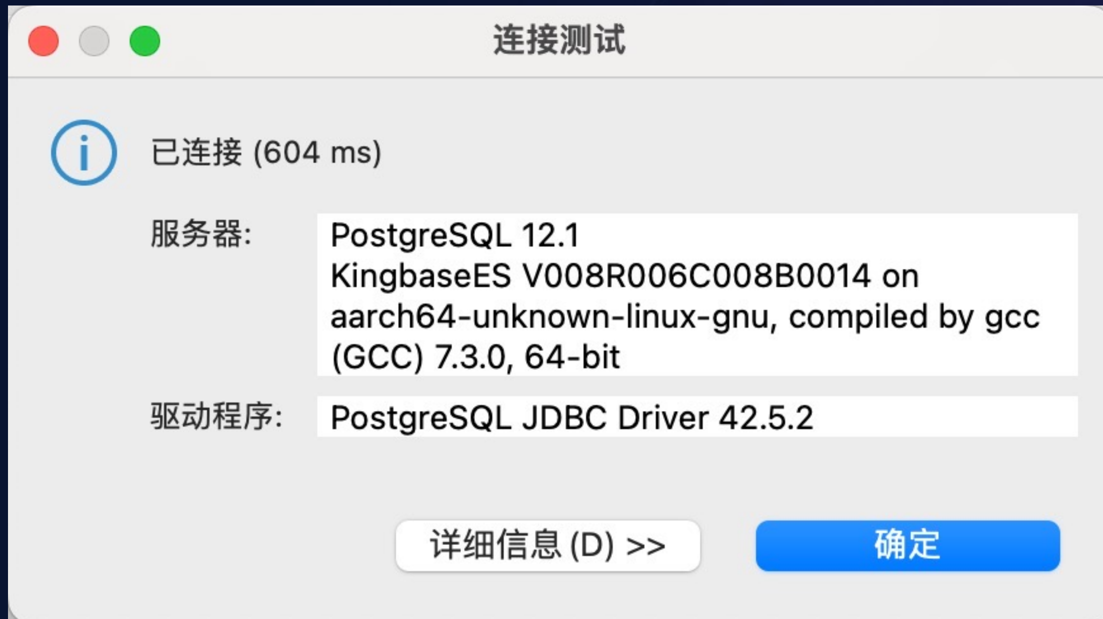
走向拥抱开源的国产数据库

公司名称	数据库产品	管理工具	成立时间
达梦	DM8，对应Oracle单实例	DMDFM，数据融合管理平台	2000年
	DMDSC 数据共享集群，对应Oracle RAC	DMDIS，数据迁移同步	
	DMDataWatch 数据主备集群，对应Oracle DataGuard	DMDRS，数据复制软件	
	DMMPP 分析型大规模数据处理集群	DMDVS，数据校验软件	
	DMRWC 读写分离集群		
	DMDPC 分布式数据库		
	云数据库		
	图数据库		
	DMCDM 新云缓存数据库		
人大金仓	KingbaseES，关系型数据库	Kingbase FlySync，数据同步软件	1999年
	金仓数据守护集群软件 （Kingbase Data Watch）	KDMS，金仓数据库迁移评估系统	
	金仓数据库读写分离集群软件 （KingbaseRWC）	KDTS，数据库迁移工具	
	金仓KingbaseRAC集群数据库软件 （KingbaseRAC）		
	金仓高可用软件 （KingbaseHA）		
	金仓集群资源管理软件 (KingbaseES Clusterware)		
	KingbaseAnalyticsDB，MPP分析型数据库		
	KSOne，分布式数据库		
南大通用	Gbase 8s，关系型数据库	GBase Enterprise Manager 工具	2004年
	Gbase 8a，MPP大型批处理	GBase8sMTK迁移工具	
	Gbase 8c，分布式数据库	GBase 8s V8.8 数据变更捕获(CDC)	
	Gbase UP，融合产品		
	Gbase 8d，LDAP		
	GBase Cloud Vector DB		
神州通用	神通数据库管理系统	神通K-Fusion数据集成系统	2008年
	神通数据库(MPP集群)	神通K-Miner数据挖掘分析系统	
	神通数据库(openGauss版)	神通K-Cuber联机分析处理系统	
	神通KSTORE海量数据管理系统	神通T-Miner文本挖掘系统	
	神通BDC大数据中心解决方案		

什么是人大金仓 KingbaseES

KingbaseES 是北京人大金仓信息技术股份有限公司（简称“人大金仓”）旗下的产品，是一款高性能分布式数据库产品，具有以下特点：

- 高可用性：采用了分布式架构，支持水平扩展和垂直扩展，可以通过增加节点或提高节点配置来提高系统的处理能力和存储容量。
- 高可靠性：采用了数据多副本存储、数据自动恢复、数据一致性校验等技术，确保了数据的安全性和可靠性。
- 高性能：采用了分布式并行计算、内存计算、列式存储等技术，提高了系统的处理性能和响应速度。
- 易用性：提供了丰富的管理工具和接口，方便用户对系统进行管理和维护。
- 兼容性：支持多种操作系统、多种编程语言和多种数据库协议，可以方便地与其他系统进行集成和交互。




```
root@ecs-kp-arm64-ai:/opt/license_29404# systemctl status kingbase
● kingbased.service - LSB: Start and stop the kingbase server
   Loaded: loaded (/etc/init.d/kingbased; generated)
   Active: active (exited) since Thu 2023-11-30 14:11:43 CST; 20s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 28776 ExecStart=/etc/init.d/kingbased start (code=exited, status=0/SUCCESS)

11月 30 14:11:43 ecs-kp-arm64-ai systemd[1]: Starting LSB: Start and stop the kingbase server...
11月 30 14:11:43 ecs-kp-arm64-ai kingbased[28776]: Starting KingbaseES V8:
11月 30 14:11:43 ecs-kp-arm64-ai su[28786]: Successful su for kingbase by root
11月 30 14:11:43 ecs-kp-arm64-ai su[28786]: + ??? root:kingbase
11月 30 14:11:43 ecs-kp-arm64-ai su[28786]: pam_unix(su:session): session opened for user kingbase by (uid=0)
11月 30 14:11:43 ecs-kp-arm64-ai kingbased[28776]: 等待服务器进程启动 .... 完成
11月 30 14:11:43 ecs-kp-arm64-ai kingbased[28776]: 服务器进程已经启动
11月 30 14:11:43 ecs-kp-arm64-ai su[28786]: pam_unix(su:session): session closed for user kingbase
11月 30 14:11:43 ecs-kp-arm64-ai kingbased[28776]: KingbaseES V8 started successfully
11月 30 14:11:43 ecs-kp-arm64-ai systemd[1]: Started LSB: Start and stop the kingbase server.
root@ecs-kp-arm64-ai:/opt/license_29404#
```

什么是华为 GaussDB

GaussDB 是华为公司自主研发的分布式数据库产品，它具有高性能、高可靠性、高扩展性、易用性等特点。

以下是一些关于 GaussDB 的特点和优势：
•高性能：采用了分布式架构和并行计算技术，能够支持海量数据的存储和处理，具有出色的性能表现。
•高可靠性：采用了多副本存储、数据一致性校验、数据恢复等技术，确保了数据的安全性和可靠性。
•高扩展性：支持水平扩展和垂直扩展，可以根据业务需求灵活地扩展系统的存储容量和处理能力。
•易用性：提供了丰富的管理工具和接口，方便用户对系统进行管理和维护。
•兼容性：支持多种操作系统、多种编程语言和多种数据库协议，可以方便地与其他系统进行集成和交互。

区域：乌兰察布一 I 可用区1
规格：
AI加速型 I kai1s.xlarge.1 I 4vCPUs I 4GiB I '1 * HUAWEI Ascend 310'
镜像：Ubuntu 18.04 server 64bit for Kai1s
系统盘：高IO，40GiB
虚拟私有云：vpc-default(192.168.0.0/16)
安全组：Sys-WebServer
Sys-FullAccess
default
源/目的检查：开启
网卡：subnet-default(192.168.0.0/24)
弹性公网IP：规格：全动态BGP
计费方式：按带宽计费
带宽：5 Mbit/s

目前官方收费版本：
gaussdb 100 (主打OLTP，在线事务处理)
gaussdb 200 (主打OLAP 在线分析处理)
gaussdb 300 (100+200混合，未来可能废弃的版本)
目前免费开源版本：
openGauss

NCC 社区孵化国产数据库驱动项目介绍

分别是 Entity Framework Core 的人大金仓 KingbaseES 和华为 GaussDB 驱动，由汪鹏（Jeffcky）老师亲自操刀，他是《你必须掌握的 Entity Framework 6.x 与 Core 2.0》一书的作者。

KingbaseES 项目：
https://github.com/dotnetcore/EntityFrameworkCore.KingbaseES

GaussDB 项目：
https://github.com/dotnetcore/EntityFrameworkCore.GaussDB

NCC 社区孵化
国产数据库驱动项目
KingbaseES

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"/>
  </ItemGroup>

  <ItemGroup>
    <ProjectReference Include="..\..\src\EFCore.KingbaseES\EFCore.KingbaseES.csproj" />
  </ItemGroup>

</Project>
```

解决方案资源管理器

- EFCore.KingbaseES
  - Properties
  - 依赖项
  - Design
  - Diagnostics
  - Extensions
  - Infrastructure
  - Internal
  - Lib
  - Metadata
  - Migrations
  - Query
  - Scaffolding
  - Storage
  - Types
  - Update
  - Utilities
  - ValueGeneration
  - Check.cs
  - CodeAnnotations.cs
  - KdbndpRetryingExecutionStrategy.cs
  - NonCapturingLazyInitializer.cs
  - README.md
  - SharedTypeExtensions.cs
  - Strings.resx
- test
  - KingbaseES.BasicTest
    - 依赖项
    - Migrations
    - Blog.cs
    - BlogContext.cs
    - BlogContextFactory.cs
    - Program.cs

.NET Conf China 2023

中国 • 北京

NCC 社区孵化
国产数据库驱动项目
KingbaseES

```csharp
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;

namespace KingbaseES.BasicTest
{
    internal static class Program
    {
        static void Main(string[] args)
        {
            var services = new ServiceCollection();

            services.AddDbContext<BlogContext>(options =>
            {
                options.UseKdbndp(@"host=localhost;port=54321;database=test;user id=system;password=123456;");
            });

            var serviceProvider = services.BuildServiceProvider();

            var context = serviceProvider.GetRequiredService<BlogContext>();

            context.Database.EnsureCreated();

            // get list
            var blogs = context.Blogs.ToList();
            Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "当前 Blogs 表中条目数: " + blogs.Count);
            if (blogs.Any())
            {
                Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "当前 Blogs 表中条目数大于1，进行清理...");
                context.Blogs.RemoveRange(blogs);
                context.SaveChanges();
                Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "清理完成！");
                Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "当前 Blogs 表中条目数: " + blogs.Count);
            }
            //add
            Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "新增 Blog 实体 值为I love EFCore!");
            context.Add(new Blog() { Name = "I love EFCore!" });
            var result = context.SaveChanges();

            //update
            var first = context.Blogs.FirstOrDefault();
            Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "查询结果:" + first.Name);

            Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "变更 Blog 实体 值为I love EFCore too!");
            first.Name = "I love EFCore too!";
            context.SaveChanges();

            // get
            var testselect = context.Blogs.FirstOrDefault();
            Console.WriteLine(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.fff") + "查询结果:" + testselect.Name);
            Console.ReadKey();
        }
    }
}
```

```
C:\Users\JaneC\source\repos\.
2023-12-08 01:35:36.983当前 Blogs 表中条目数: 1
2023-12-08 01:35:36.995当前 Blogs 表中条目数大于1, 进行清理...
2023-12-08 01:35:37.090清理完成!
2023-12-08 01:35:37.090当前 Blogs 表中条目数: 1
2023-12-08 01:35:37.090新增 Blog 实体 值为I love EFCore!
2023-12-08 01:35:37.203查询结果:I love EFCore!
2023-12-08 01:35:37.204变更 Blog 实体 值为I love EFCore too!
2023-12-08 01:35:37.256查询结果:I love EFCore too!
```

EFCore 8 的新变化
及走向拥抱开源的国产数据库

Q & A