عنوان: **خلاصه‌ای از اعمال متداول با AutoMapper و Entity Framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۲/۱۹ ۲۱:۱۳
آدرس: www.dotnettips.info
گروه‌ها: MVC, Entity framework, AutoMapper

فرض کنید کلاس‌های مدل برنامه از سه کلاس مشتری، سفارشات مشتری‌ها و اقلام هر سفارش تشکیل شده‌است:

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

    public virtual ICollection<Order> Orders { get; set; }

    [Computed]
    [NotMapped]
    public string FullName
    {
        get { return FirstName + ' ' + LastName; }
    }
}

public class Order
{
    public int Id { set; get; }
    public string OrderNo { get; set; }
    public DateTime PurchaseDate { get; set; }
    public bool ShipToHomeAddress { get; set; }

    public virtual ICollection<OrderItem> OrderItems { get; set; }

    [ForeignKey("CustomerId")]
    public virtual Customer Customer { get; set; }
    public int CustomerId { get; set; }

    [Computed]
    [NotMapped]
    public decimal Total
    {
        get { return OrderItems.Sum(x => x.TotalPrice); }
    }
}

public class OrderItem
{
    public int Id { get; set; }
    public decimal Price { get; set; }
    public string Name { get; set; }
    public int Quantity { get; set; }

    [ForeignKey("OrderId")]
    public virtual Order Order { get; set; }
    public int OrderId { get; set; }

    [Computed]
    [NotMapped]
    public decimal TotalPrice
    {
        get { return Price * Quantity; }
    }
}
```

در اینجا برای پیاده سازی خواص محاسباتی، از نکته‌ی مطرح شده‌ی در مطلب « نگاشت خواص محاسبه شده به کمک AutoMapper و DelegateDecompiler » استفاده شده‌است.
در ادامه می‌خواهیم اطلاعات حاصل از این کلاس‌ها را با شرایط خاصی به ViewModelهای مشخصی جهت نمایش در برنامه نگاشت کنیم.

نمایش اطلاعات مشتری‌ها

می‌خواهیم اطلاعات مشتری‌ها را مطابق فرمت کلاس ذیل بازگشت دهیم:

```
public class CustomerViewModel
{
    public string Bio { get; set; }
    public string CustomerName { get; set; }
}
```

با این شرایط که

- اگر Bio نال بود، بجای آن N/A نمایش داده شود.

- CustomerName از خاصیت محاسباتی FullName کلاس مشتری تامین گردد.

برای حل این مساله، نیاز است نگاشت زیر را تهیه کنیم:

```
this.CreateMap<Customer, CustomerViewModel>()
   .ForMember(dest => dest.CustomerName, opt => opt.MapFrom(entity => entity.FullName))
   .ForMember(dest => dest.Bio, opt => opt.MapFrom(entity => entity.Bio ?? "N/A"));
```

AutoMapper برای جایگزین کردن خواص با مقدار نال، با یک مقدار مشخص، از متدی به نام NullSubstitute استفاده می‌کند. اما در این حالت خاص که قصد داریم از Project To استفاده کنیم، این روش پاسخ نمی‌دهد و محدودیت‌هایی دارد . به همین جهت از روش map from و بررسی مقدار خاصیت، استفاده شده‌است.
همچنین در اینجا مطابق نگاشت فوق، خاصیت CustomerName از خاصیت FullName کلاس مشتری دریافت می‌شود.

کوئری نهایی استفاده کننده‌ی از این اطلاعات به شکل زیر خواهد بود:

```
using (var context = new MyContext())
{
    var viewCustomers = context.Customers
        .Project()
        .To<CustomerViewModel>()
        .Decompile()
        .ToList();
    // don't use
    // var viewCustomers = Mapper.Map<IEnumerable<Customer>,
IEnumerable<CustomerViewModel>>(dbCustomers);
    foreach (var customer in viewCustomers)
    {
        Console.WriteLine("{0} - {1}", customer.CustomerName, customer.Bio);
    }
}
```

در اینجا از متدهای Project To و همچنین Decompile استفاده شده‌است (جهت پردازش خاصیت محاسباتی).

نمایش اطلاعات سفارش‌های مشتری‌ها

در ادامه قصد داریم اطلاعات سفارش‌ها را با فرمت ViewModel ذیل نمایش دهیم:

```
public class OrderViewModel
{
    public string CustomerName { get; set; }
    public decimal Total { get; set; }
    public string OrderNumber { get; set; }
    public IEnumerable<OrderItemsViewModel> OrderItems { get; set; }
}

public class OrderItemsViewModel
{
    public string Name { get; set; }
    public int Quantity { get; set; }
    public decimal Price { get; set; }
}
```

با این شرایط که
- CustomerName از خاصیت محاسباتی FullName کلاس مشتری تامین گردد.
- خاصیت OrderNumber آن از خاصیت OrderNo تهیه گردد.

به همین جهت کار را با تهیه‌ی نگاشت ذیل ادامه می‌دهیم:

```
this.CreateMap<Order, OrderViewModel>()
  .ForMember(dest => dest.OrderNumber, opt => opt.MapFrom(src => src.OrderNo))
  .ForMember(dest => dest.CustomerName, opt => opt.MapFrom(src => src.Customer.FullName));
```

بر این اساس کوئری مورد استفاده نیز به نحو ذیل تشکیل می‌شود:

```
using (var context = new MyContext())
{
    var viewOrders = context.Orders
        .Project()
        .To<OrderViewModel>()
        .Decompile()
        .ToList();
    // don't use
    // var viewOrders = Mapper.Map<IEnumerable<Order>, IEnumerable<OrderViewModel>>(dbOrders);
    foreach (var order in viewOrders)
    {
        Console.WriteLine("{0} - {1} - {2}", order.OrderNumber, order.CustomerName, order.Total);
    }
}
```

در اینجا چون از خاصیت OrderItems کلاس ViewModel صرفنظر نشده‌است، اطلاعات آن نیز به همراه viewOrders موجود است. یعنی می‌توان چنین کوئری را نیز جهت نمایش اطلاعات تو در توی اقلام هر سفارش نیز نوشت:

```
using (var context = new MyContext())
{
    var viewOrders = context.Orders
        .Project()
        .To<OrderViewModel>()
        .Decompile()
        .ToList();
    // don't use
    // var viewOrders = Mapper.Map<IEnumerable<Order>, IEnumerable<OrderViewModel>>(dbOrders);
    foreach (var order in viewOrders)
    {
        Console.WriteLine("{0} - {1} - {2}", order.OrderNumber, order.CustomerName, order.Total);
        foreach (var item in order.OrderItems)
        {
            Console.WriteLine("({0}) {1} - {2}", item.Quantity, item.Name, item.Price);
        }
    }
}
```

اگر می‌خواهید OrderItems به صورت خودکار واکشی نشود، نیاز است در نگاشت تهیه شده، توسط متد Ignore از آن صرفنظر کنید:

```
this.CreateMap<Order, OrderViewModel>()
  .ForMember(dest => dest.OrderNumber, opt => opt.MapFrom(src => src.OrderNo))
  .ForMember(dest => dest.OrderItems, opt => opt.Ignore())
  .ForMember(dest => dest.CustomerName, opt => opt.MapFrom(src => src.Customer.FullName));
```

نمایش اطلاعات یک سفارش، با فرمتی خاص

تا اینجا نگاشت‌های انجام شده بر روی لیستی از اشیاء صورت گرفتند. در ادامه می‌خواهیم اولین سفارش ثبت شده را با فرمت

ذیل نمایش دهیم:

```
public class OrderDateViewModel
{
    public int PurchaseHour { get; set; }
    public int PurchaseMinute { get; set; }
    public string CustomerName { get; set; }
}
```

به همین منظور ابتدا نگاشت ذیل را تهیه می‌کنیم:

```
this.CreateMap<Order, OrderDateViewModel>()
  .ForMember(dest => dest.PurchaseHour, opt => opt.MapFrom(src => src.PurchaseDate.Hour))
  .ForMember(dest => dest.PurchaseMinute, opt => opt.MapFrom(src => src.PurchaseDate.Minute))
  .ForMember(dest => dest.CustomerName, opt => opt.MapFrom(src => src.Customer.FullName));
```

در اینجا ساعت و دقیقه‌ی خرید، از خاصیت PurchaseDate استخراج شده‌اند. همچنین CustomerName نیز از خاصیت FullName کلاس مشتری دریافت گردیده‌است.
پس از این تنظیمات، کوئری نهایی به شکل ذیل خواهد بود:

```
using (var context = new MyContext())
{
    var viewOrder = context.Orders
        .Project()
        .To<OrderDateViewModel>()
        .Decompile()
        .FirstOrDefault();
    // don't use
    // var viewOrder = Mapper.Map<Order, OrderDateViewModel>(dbOrder);

    if (viewOrder != null)
    {
        Console.WriteLine("{0}, {1}:{2}", viewOrder.CustomerName, viewOrder.PurchaseHour,
viewOrder.PurchaseMinute);
    }
}
```

فرمت کردن سفارشی اطلاعات در حین نگاشت‌ها

در مورد فرمت کننده‌های سفارشی و تبدیلگرها پیشتر بحث کرده‌ایم. اما اغلب آن‌ها را در حالت خاص LINQ to Entities نمی‌توان بکار برد، زیرا قابلیت تبدیل به SQL را ندارند. برای مثال فرض کنید می‌خواهیم خاصیت ShipToHomeAddress کلاس Order را به خاصیت ShipHome کلاس ذیل نگاشت کنیم:

```
public class OrderShipViewModel
{
    public string ShipHome { get; set; }
    public string CustomerName { get; set; }
}
```

با این شرط که اگر مقدار آن True بود، Yes را نمایش دهد. با توجه به ساختار مدنظر، نگاشت ذیل را می‌توان تهیه کرد که در آن فرمت کردن سفارشی، به متد MapFrom واگذار شده‌است:

```
this.CreateMap<Order, OrderShipViewModel>()
   .ForMember(dest => dest.ShipHome, opt => opt.MapFrom(src=>src.ShipToHomeAddress? "Yes": "No"))
   .ForMember(dest => dest.CustomerName, opt => opt.MapFrom(src => src.Customer.FullName));
```

با این کوئری جهت استفاده‌ی از این تنظیمات:

```
using (var context = new MyContext())
```

```
{
    var viewOrders = context.Orders
        .Project()
        .To<OrderShipViewModel>()
        .Decompile()
        .ToList();
    // don't use
    // var viewOrders = Mapper.Map<IEnumerable<Order>, IEnumerable<OrderShipViewModel>>(dbOrders);
    foreach (var order in viewOrders)
    {
        Console.WriteLine("{0} - {1}", order.CustomerName, order.ShipHome);
    }
}
```

کدهای کامل این مطلب را از اینجا می‌توانید دریافت کنید.