Cleanest Way To Map Entity To DTO With Linq Select?

Asked 4 years, 5 months ago Active 4 years, 5 months ago Viewed 29k times

15

6

I've been trying to come up with a clean and reusable way to map entities to their DTOs. Here is an example of what I've come up with and where I'm stuck.

Entities

```
public class Person
{
    public int ID { get; set; }
    public string Name { get; set; }
    public Address Address { get; set; }
    // Other properties not included in DTO
}

public class Address
{
    public int ID { get; set; }
    public string City { get; set; }
    // Other properties not included in DTO
}
```

DTOs

```
public class PersonDTO
{
    public int ID { get; set; }
    public string Name { get; set; }
    public AddressDTO Address { get; set; }
}

public class AddressDTO
{
    public int ID { get; set; }
```

This is how I began to handle the mapping. I wanted a solution that wouldn't execute the query before mapping. I've been told that if you pass a `Func<in, out>` instead of `Expression<Func<in, out>>` that it will execute the query before mapping.

```
public static Expressions
{
    public static Expression<Func<Person, PersonDTO>> = (person) => new PersonDTO()
    {
        ID = person.ID,
        Name = person.Name,
        Address = new AddressDTO()
        {
            ID = person.Address.ID,
            City = person.Address.City
        }
    }
}
```

One issue with this is that I already have an expression that maps an `Address` to an `AddressDTO` so I have duplicated code. This will also break if `person.Address` is null. This gets messy very quick especially if I want to display other entities related to person in this same DTO. It becomes a birds nest of nested mappings.

I've tried the following but Linq doesn't know how to handle it.

```
public static Expressions
{
    public static Expression<Func<Person, PersonDTO>> = (person) => new PersonDTO()
    {
        ID = person.ID,
        Name = person.Name,
        Address = Convert(person.Address)
    }

    public static AddressDTO Convert(Address source)
    {
        if (source == null) return null;
        return new AddressDTO()
        {
            ID = source.ID,
            City = source.City
```

c# linq entity-framework linq-to-sql linq-to-entities

asked Apr 13 '15 at 19:45

Jeff
294 2 6 23

AutoMapper: automapper.org – Christian Apr 13 '15 at 19:57

I've used AutoMapper before but I was under the assumption that it had to execute the query before mapping. After looking further into the documentation it looks like it may have something close to what I'm looking for HERE – Jeff Apr 13 '15 at 20:03

1 Your query will execute when the mapping is performed but if there are fields in the entity that you're not interested use `Project().To<>` which is available both for NHibernate and EntityFramework. It will effectively do a `select` on the fields specified in the mapping configurations. – Christian Apr 13 '15 at 20:20

1 Yes, `Project().To<>` is definitely the way to go. Also, AutoMapper can deal with nested collections, and it has limited support for flattening. – Gert Arnold Apr 13 '15 at 20:33

3 Answers

8

Just use AutoMapper.

Example:

```
Mapper.CreateMap<Address, AddressDTO>();
Mapper.CreateMap<Person, PersonDTO>();
```

> Your query will execute when the mapping is performed but if there are fields in the entity that you're not interested use `Project().To<>` which is available both for NHibernate and EntityFramework. It will effectively do a select on the fields specified in the mapping configurations.

6 Might add some value: rogeralsing.com/2013/12/01/... – Vaibhav Sep 22 '16 at 12:29

8

If you want to create mappings manually then you can use Select on the collection in the following way:

Some test data:

```
    var persons = new List<Person>
    {
        new Person() {ID = 1, Name = "name1", Address = new Address() {ID = 1, City =
"city1"}},
        new Person() {ID = 2, Name = "name2", Address = new Address() {ID = 2, City =
"city2"}},
        new Person() {ID = 3, Name = "name3", Address = new Address() {ID = 1, City =
"city1"}}
    };
```

Mapping methods:

```
public static PersonDTO ToPersonDTOMap(Person person)
{
    return new PersonDTO()
    {
        ID = person.ID,
        Name = person.Name,
        Address = ToAddressDTOMap(person.Address)
    };
}

public static AddressDTO ToAddressDTOMap(Address address)
{
    return new AddressDTO()
    {
        ID = address.ID,
        City = address.City
    };
}
```

Keep in mind that if this was a real query is would not get executed as long as it was IQueryable, it would be executed once you materialize it (using ToList() for example).

However, I would consider using some framework which could do it (the mappings) for you automatically (if your mapping are as simple as provided example(.

edited Apr 18 '15 at 10:41

answered Apr 13 '15 at 20:04

Andrew B

707 1 10 20

1

You could either use AutoMapper or write extension methods like these:

```
public static class PersonMapper
{
    public static PersonDTO ConvertToDTO(this Person person)
    {
        return new PersonDTO { ID = person.ID, Name = person.Name, Address =
person.Address.ConvertToDTO() };
    }

    public static IEnumerable<PersonDTO> ConvertToDTO(this IEnumerable<Person> people)
    {
        return people.Select(person => person.ConvertToDTO());
    }
}

public static class AddressMapper
{
    public static AddressDTO ConvertToDTO(this Address address)
    {
        return new AddressDTO { ID = address.ID, City = address.City };
    }

    public static IEnumerable<AddressDTO> ConvertToDTO(this IEnumerable<Address>
addresses)
    {
        return addresses.Select(address => address.ConvertToDTO());
```

```
public class Program
{
    static void Main(string[] args)
    {
        Person person = new Person { ID = 1, Name = "John", Address = new Address { ID =
1, City = "New Jersey" } };
        PersonDTO personDTO = person.ConvertToDTO();
        Console.WriteLine(personDTO.Name);
    }
}
```

answered Apr 13 '15 at 20:03

chomba
1,134 10 13

Would using these extension methods `SELECT * FROM Persons` and then map or would this actually modify the output SQL to only `SELECT` those properties included in the extension. – Jeff Apr 13 '15 at 20:13