Hướng dẫn sử dụng AutoMapper

**AutoMapper là gì?**

AutoMapper là một object to object mapper. Object-Object mapping hoạt động bằng cách chuyển đổi dữ liệu của Object này sang dữ liệu của Object khác. Điều làm cho AutoMapper thú vị là nó cung cấp một số quy ước để loại bỏ công việc khó khăn khi mapping từ A sang B.

**Tại sao sử dụng AutoMapper?**

Mapping dữ liệu giữa các đối tượng là một công việc nhàm chặn. Kiểm tra lại việc Mapping thậm chí còn nhàm chặn hơn. AutoMapper cung cập cấu hình đơn giản, cũng như kiểm tra đơn giản việc mapping. Câu hỏi thực tế là: “Tại sao sử dụng object-object mapping?”. Mapping có thể xuất hiện nhiều nơi trong ứng dụng, nhưng

Trong quá trình phát triển một ứng dụng web Asp.net core bạn thường cần chuyển đối dữ liệu từ một object này sang một object khác. Bạn có thể hoàn thành nhiệm vụ này bằng cách thực hiện thiết lập thủ công những thuộc tính của object từ object nguồn. Những sẽ đơn giản và tránh mất thời gian hơn là khi công việc này thực hiện một các tự động. Và AutoMapper sinh ra để giúp ta làm việc mapping từ Object này sang Object khác một cách tự động. Nhưng hầu hết xuất hiện giữa các lớp như giữa lớp UI/Domain hoặc giữa lớp Service/Domain. Mối lo ngại sự xung đột giữa các file khác, vì vậy object-object mapping dẫn đến mô hình tách biệt nơi mà những file này chỉ có tác dụng trong layer của nó.

**Làm thế nào để sử dụng AutoMapper?**

Đầu tiên, bạn cần cả dữ liệu nguồn và dữ liệu đích để làm việc. Thiết kế của dữ liệu đích có thể bị ảnh hưởng bởi lớp mà nó thuộc. Nếu dữ liệu nguồn có trường FirstName thì sẽ tự động mapping với dữ trường có tên FirstName của dữ liệu đích.

AutoMapper sẽ bỏ qua tham chiếu null khi mapping. Nếu không muốn cách tiếp cận này thì bạn có thể áp dụng custome value resolvers.

Khi bạn đã có dữ liệu của minh thì bạn có thể sử dụng MapperConfiguration và Createmap. Bạn chỉ cần một instance của MapperConfiguration ở AppDomain và bạn cài đặt trong khi khởi động.

Ví dụ:

var config = **new** MapperConfiguration(cfg => cfg.CreateMap<Order, OrderDto>());

Trong đó Order là nguồn, OrderDto là đích.

Để thực hiện mapping ta gọi Map như sau:

var mapper = config.CreateMapper();

*// or*

var mapper = **new** Mapper(config);

OrderDto dto = mapper.Map<OrderDto>(order);

Hầu hết ứng dụng có thể sử dụng phương pháp inject để tạo instance của IMapper.

**Nơi cấu hình AutoMapper ở đâu?**

Cấu hình nên được thực hiện ở AppDomain. Có nghĩa là nơi tốt nhất để đặt code configuration ở nơi ứng dụng bắt đầu, như Global.asax ở ASP.NET, Startup.cs ở Asp.net core.

Làm thế nào để kiểm tra mapping?

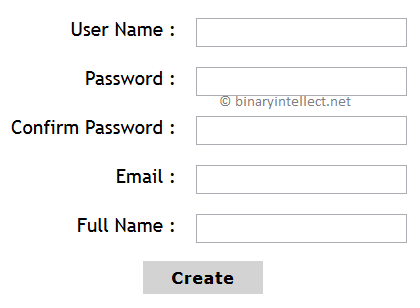
Để kiểu tra mapping của bạn, bạn có thể tạo ra test bằng 2 cách:

* Gọi boostrap của bạn để tạo tất cả các ánh xạ
* Gọi mapperConfiguration.AsertConfigurationIsValid

var config = AutoMapperConfiguration.Configure();

config.AssertConfigurationIsValid();

Ví dụ khi ta làm nhiệm vụ quản lí user trên web. Ta có form như sau:



Như vậy có thể thấy, có 5 trường trong form được đặt tên lần lượt là UserName, password, Confirm password, Email, Full Name. Thông thường bạn sẽ lấy dữ liệu được nhập bởi người dùng trong trang này bằng cách sử dụng model binding. Và khi đó bạn sẽ tạo ra class view model có dạng như sau;

public class UserViewModel

{

public string UserName { set; get; }

public string Password { set; get; }

public string Email { set; get; }

public string Name { set; get; }

}

Bây giờ trong class UserViewModel đang cso một thuộc tính Pasword và ConfirmPassword là cần thiết bởi vì hệ thống thực hiện Validate dữ liệu để bạn xác nhận chính xác Password của bạn vừa nhập không bị sai.

Nhưng Api nhận dữ liệu thì để thêm User thì lại không cần ConfirmPassword mà chỉ cần minh trường Password và lưu Password trong cơ sở dữ liệu. Tuy nhiên, nó lại cần thêm một thuộc tính khác như UserId để xác định xem chúng ta thêm mới hay update dữ liệu và trường này thì không được năm bắt bởi view. Như vậy ta có class mà service sử dụng sẽ như sau:

public class AppUser

{

public int UserId { set; get; }

public string UserName { set; get; }

public string Password { set; get; }

public string Email { set; get; }

public string FullName { set; get; }

}

Bây giờ ta thấy rằng giữa client và service đang sử dụng 2 định dạng dữ liệu không tương đồng nhau. Bình thường, ta chuyển đổi qua lại giữa hai định dạng này một cách thủ công bằng cách tạo 1 đối tượng và gán các giá trị từ đối tượng kia để thu được giá trị mong muốn.

public IActionResult Register (UserViewModel model)

{

if (ModelState.IsValid)

{

AppUser user = new AppUser()

{

UserName = model.UserName,

Password = model.Password,

Email = model.Email,

FullName = model.Name

};

db.Users.Add(user);

db.SaveChanges();

}

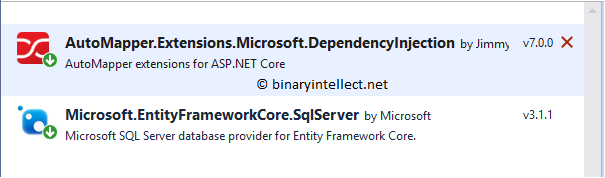
return View();

}

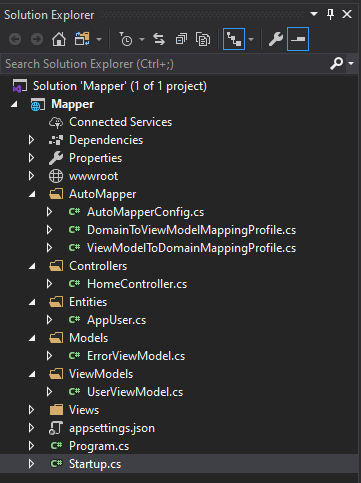
Ở trên ta thấy đoạn code gán thủ công từ object này cho object kia. Đây chỉ là một tình huống đơn giản nơi mà việc ánh xạ (mapping) với ít dữ liệu. Nếu với object nhiều thuộc tính thì ta phải làm rất mất thời gian. Và công việc chuyển đổi qua lại giữa các định dạng cũng diễn ra rất nhiều trong lập trình hàng ngày nên ta sẽ thực hiện sử dụng AutoMapper để giải quyết.

Cấu hình:

Đầu tiên ta thực hiện cài AutoMapper trong nuget package: AutoMapper.Extensions.Microsoft.DependencyInjection



Tiếp theo ta thực hiện:



Trong đó:

- AutoMapper: Thư mục chứa những config giúp ta mapping.

+ AutoMapperConfig.cs: File chứa MapperConfiguration nơi đăng kí nhưng Profile

+ DomainToViewModelMappingProfile.cs: Nơi cấu hình khi ta mapping từ file Domain trên service về sang file ViewModel cho client.

+ ViewModelToDoMainMappingProfile.cs: Nơi cấu hình khi ta mapping từ file ViewModel ở client về lại file Domain trên service.

- Controllers:

+ HomeController: Nơi thực hiện mapping dữ liệu

- Entities: Nơi chứa file tương ứng dữ liệu Domain ở trên server

- Models: Nơi chứa file tương ứng dữ liệu ViewModel ở client

Nội dung của Entities/AppUser.cs

namespace Mapper

{

public class AppUser

{

public int UserId { set; get; }

public string UserName { set; get; }

public string Password { set; get; }

public string Email { set; get; }

public string FullName { set; get; }

}

}

Nội dung của file Models/UserViewModel.cs

namespace Mapper

{

public class UserViewModel

{

public string UserName { set; get; }

public string Password { set; get; }

public string Email { set; get; }

public string Name { set; get; }

}

}

Nội dung của file AutoMapper/AutoMapperConfig.cs

using AutoMapper;

using Mapper.AutoMapper;

namespace Mapper

{

public class AutoMapperConfig

{

public static MapperConfiguration RegisterMappings()

{

return new MapperConfiguration(cfg =>

{

cfg.AddProfile(new DomainToViewModelMappingProfile());

cfg.AddProfile(new ViewModelToDomainMappingProfile());

});

}

}

}

Nội dung của file AutoMapper/DomainToViewModelMappingProfile.cs

using AutoMapper;

namespace Mapper.AutoMapper

{

public class DomainToViewModelMappingProfile: Profile

{

public DomainToViewModelMappingProfile()

{

CreateMap<AppUser, UserViewModel>();

}

}

}

Nội dung của file AutoMapper/ViewModelToDomainMappingProfile.cs

using AutoMapper;

namespace Mapper.AutoMapper

{

public class ViewModelToDomainMappingProfile: Profile

{

public ViewModelToDomainMappingProfile()

{

CreateMap<UserViewModel, AppUser>().ForMember(x => x.FullName, option => option.MapFrom(source => source.Name));

}

}

}

Trong Startup.cs ta thêm cấu hình như sau:

public void ConfigureServices(IServiceCollection services)

{

…

IMapper mapper = AutoMapperConfig.RegisterMappings().CreateMapper();

services.AddSingleton(mapper);

…

}

Để sử dụng trong HomeController ta dùng như sau:

public IActionResult Index()

{

var userViewModel = new UserViewModel() { Email = "Email", Name = "Name", Password = "Password", UserName = "UserName" };

AppUser domainUser = \_mapper.Map<AppUser>(userViewModel);

domainUser.UserId = 10;

domainUser.Email = "Hehe";

var appUser = \_mapper.Map<UserViewModel>(domainUser);

return View(domainUser);

}

Khi mapping dữ liệu giữa các object không phải lúc nào dữ liệu giữa các object cũng trùng tên nhau. Như ở ví dụ của ta ta thấy ở AppUser là FullName nhưng ở UserViewModel thì lại là Name. Như vậy thì ta cần phải config, không thì khi mapping dữ liệu sẽ không nhận.

Để cấu hình mapping khi không trùng tên nhau ta thực hiện như sau:

Trong file ViewModelToDomainMappingProfile.cs

using AutoMapper;

namespace Mapper.AutoMapper

{

public class ViewModelToDomainMappingProfile: Profile

{

public ViewModelToDomainMappingProfile()

{

CreateMap<UserViewModel, AppUser>().ForMember(x => x.FullName, option => option.MapFrom(source => source.Name));

}

}

}

Ta cũng có thể thêm điều khi khi mapping ví dụ bạn chỉ muốn ánh xạ DisplayName đến FullName khi nó khác UserName ta thực hiện như sau:

public class AutoMapperProfile : Profile

{

public AutoMapperProfile()

{

CreateMap<UserViewModel, AppUser>()

.ForMember(destination => destination.FullName,

options => options.MapFrom(source => source.DisplayName))

.ForMember(destination => destination.FullName,

options => options.Condition(source => source.DisplayName

!= source.UserName));

}

Một số cấu hình hay dùng:

1. Projection: Phép chiếu

Dùng để chuyển dữ liệu nguồn thành đích bằng cách lấy các giá trị trong đối tượng,

**public** **class** **CalendarEvent**

{

**public** DateTime Date { **get**; **set**; }

**public** string Title { **get**; **set**; }

}

**public** **class** **CalendarEventForm**

{

**public** DateTime EventDate { **get**; **set**; }

**public** int EventHour { **get**; **set**; }

**public** int EventMinute { **get**; **set**; }

**public** string Title { **get**; **set**; }

}

*// Model*

var calendarEvent = **new** CalendarEvent

{

Date = **new** DateTime(2008, 12, 15, 20, 30, 0),

Title = "Company Holiday Party"

};

*// Configure AutoMapper*

var configuration = **new** MapperConfiguration(cfg =>

cfg.CreateMap<CalendarEvent, CalendarEventForm>()

.ForMember(dest => dest.EventDate, opt => opt.MapFrom(src => src.Date.Date))

.ForMember(dest => dest.EventHour, opt => opt.MapFrom(src => src.Date.Hour))

.ForMember(dest => dest.EventMinute, opt => opt.MapFrom(src => src.Date.Minute)));

*// Perform mapping*

CalendarEventForm form = mapper.Map<CalendarEvent, CalendarEventForm>(calendarEvent);

form.EventDate.ShouldEqual(**new** DateTime(2008, 12, 15));

form.EventHour.ShouldEqual(20);

form.EventMinute.ShouldEqual(30);

form.Title.ShouldEqual("Company Holiday Party");

2. Nested Mapping: Mapping lồng nhau

Khi mapping engine thực hiện mapping, nó có thể sử dụng một trong nhiều cách để phân giải giá trị thành viên đích. Một trong những cách này là sử dụng một map khác, nơi mà kiểu nguồn và kiểu đích cũng được cấu hình trong ánh xạ.

**public** **class** **OuterSource**

{

**public** int Value { **get**; **set**; }

**public** InnerSource Inner { **get**; **set**; }

}

**public** **class** **InnerSource**

{

**public** int OtherValue { **get**; **set**; }

}

**public** **class** **OuterDest**

{

**public** int Value { **get**; **set**; }

**public** InnerDest Inner { **get**; **set**; }

}

**public** **class** **InnerDest**

{

**public** int OtherValue { **get**; **set**; }

}

var config = **new** MapperConfiguration(cfg => {

cfg.CreateMap<OuterSource, OuterDest>();

cfg.CreateMap<InnerSource, InnerDest>();

});

config.AssertConfigurationIsValid();

var source = **new** OuterSource

{

Value = 5,

Inner = **new** InnerSource {OtherValue = 15}

};

var mapper = config.CreateMapper();

var dest = mapper.Map<OuterSource, OuterDest>(source);

dest.Value.ShouldEqual(5);

dest.Inner.ShouldNotBeNull();

dest.Inner.OtherValue.ShouldEqual(15);