# AutoMapper使用培训

在应用开发的过程中，首先要了解整个系统中各个系统的组件的作用，然后了解系统的工作流(workflow)，最后需要梳理一遍数据流（dataflow），而在整理数据流的过程中，数据的转化常常是最难理解写起来相当枯燥乏味的部分，AutoMapper的功能就是实现映射一个对象到另一个对象的自动化工具，最常见的就是DTO (Data Transfer Object)与MO(ModelObject)之间的转换，这里我先介绍一下DTO与MO的关系，然后才介绍AutoMapper工具的使用。

## 理解DTO

### 以下是我总结的几个不同平台的解释：

1. 百度百科：

数据传输对象（DTO)(Data Transfer Object)，是一种设计模式之间传输数据的软件应用系统。数据传输目标往往是数据访问对象从数据库中检索数据。数据传输对象与数据交互对象或数据访问对象之间的差异是一个以不具有任何行为除了存储和检索的数据（访问和存取器）

1. 维基百科：

数据传输对象（DTO）是在两个进程之间承载数据的对象。数据传输和业务对象或数据访问对象之间的区别在于，DTO除了存储和检索其自身的数据（调制器和访问器）之外没有任何行为。DTO是不应该包含任何需要测试的业务逻辑的简单对象。

1. 博客园dax.net：

表现层于应用层之间是通过DTO来进行交互的，数据传输对象是没有行为的POCO对象，他的目的是为了对领域对象进行数据封装，实现层与层之间的数据传递。简单来说Model面向业务，我们是通过业务来定义Model的。而DTO是面向UI，通过UI的需求来定义的，通过DTO我们实现了表现层与Model层之间的解耦，表现层不引用Model。如果开发过程中我们的模型变了，而界面没变，我们只需改Model而不需要去改动表现层。

看了这些解释，应该还是有点蒙？

### 先来看看为什么要使用DTO，我总结了三点

1. 隔离领域模型，使用封装（包括数据结构的精简和合并）的DTO实现改动领域模型而不影响UI，保持领域模型的安全，不暴露业务逻辑。
2. 在分布式模式下，相同的数据结构在不同的场景使用相同的数据结构有不同的需求。
3. DTO的存在也为了帮助减少客户端请求而降低服务器压力，提升效率。

不是所有的时候都需要用到DTO，如果为所有的不同需求都建一个DTO，那么一个概念所对应的DTO可能多达5，6种，非常难于管理，当页面所需要的数据与数据库中的数据不一致的情况如果常常出现，那么我们就需要开始着手准备添加对DTO的支持，并且DTO可以设当的冗余设计来减少DTO种类

### 在系统中添加DTO主要有以下几部分工作需要完成

1. 设计和添加DTO类。
2. 添加从MO到DTO的转化逻辑。
3. 将原本对MO的使用转换为对DTO的使用。（这里可以看出，要尽早的预判是否需要DTO，以免将来麻烦）

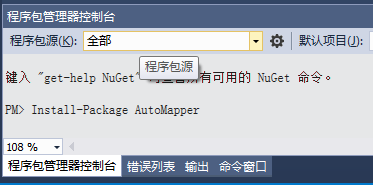
关于DTO就了解到这，下面开始回归正题，介绍AutoMapper工具的使用

## AutoMapper入门

### 安装AutoMapper

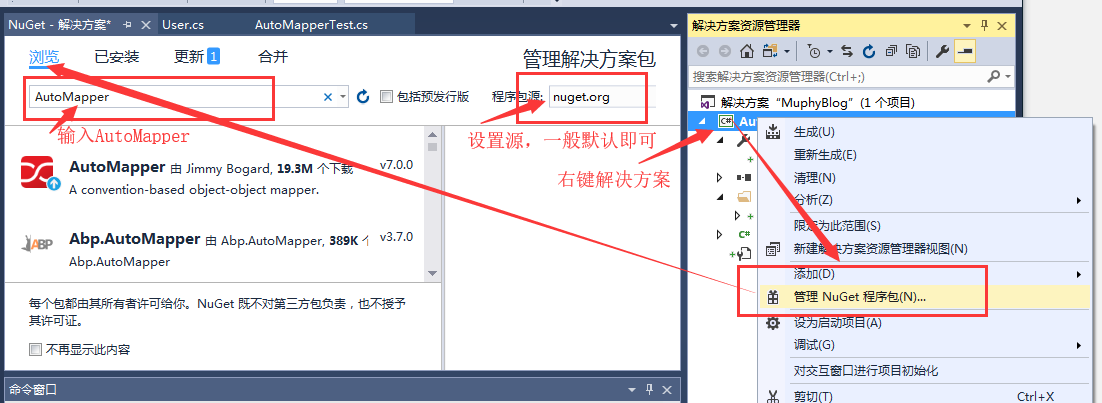
1. 从包管理器控制台安装

打开程序包管理器控制台，然后输入命令：Install-Package AutoMapper



1. 从包管理器界面安装

在界面中搜索需要安装的程序集，如AutoMapper.dll



安装完成之后添加到程序集引用中即可。

### 如何使用AutoMapper

首先，您需要使用源类型和目标类型。目标类型的设计可能受其所在层的影响，但只要成员的名称与源类型的成员匹配，AutoMapper就可以发挥最佳效果。如果您有一个名为“FirstName”的源成员，它将自动映射到名称为“FirstName”的目标成员。AutoMapper也支持拼合。

一旦你有了你的类型，你可以使用MapperConfiguration或者静态Mapper实例和CreateMap为这两种类型创建一个映射。MapperConfiguration通常每个AppDomain只需要一个实例，并且应该在启动过程中实例化。

1. 创建映射的两种方式（左侧的类型是源类型，右侧的类型是目标类型）：
2. 静态方式：

Mapper.Initialize(cfg => cfg.CreateMap<Order, OrderDto>());

如果您使用的是静态Mapper方法，则每个AppDomain只能进行一次配置。这意味着放置配置代码的最佳位置是在应用程序启动时，例如ASP.NET应用程序的Global.asax文件

1. 实例方式

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

要执行映射又可以使用静态或实例映射器方法：

1. IMapper mapper = config.CreateMapper();
2. IMapper mapper = new Mapper(config);

大多数应用程序可以使用依赖注入来注入创建的IMapper实例。

1. 可以添加配置到配置类中，方便统一管理（AddProfile），然后通过自己写一下泛型返回自己想要的类型转换，这样可以一次性实例化多种类型映射：
2. Mapper.Initialize(cfg => cfg.AddProfile(new MyProfile()));
3. Mapper.Initialize(cfg => cfg.AddProfile<MyProfile>());
4. var config = new MapperConfiguration(fig => fig.AddProfile(new MyProfile()));
5. var config = new MapperConfiguration(fig => fig.AddProfile< MyProfile>());

### 处理下面几种情况的方式

属性名称相同

扁平化映射

投影(指定字段)

简单类型映射

使用Profile配置

属性名称不同

条件映射：

空值替换NullSubstitute

忽略属性Ignore

目标属性多于源属性，可以进行预设值

类型转换ITypeConverter

using AutoMapper;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AutoMapperDemo

{

public static class AutoMapperInitialize

{

static AutoMapperInitialize()

{

Mapper.Initialize(fig => fig.AddProfile<MyProfile>());

}

public class User

{

public int Id { set; get; }

public string Name { set; get; }

public int Age { set; get; }

public User(int i)

{

Id = i;

Name = "muphy-" + i;

Age = 15 + i;

}

public User()

{

Id = 1;

Name = "muphy";

Age = 15;

}

public override string ToString()

{

return "[Name:" + Name + ",Age:" + Age + "]";

}

}

public class UserDto

{

public string Name { set; get; }

public int Age { set; get; }

public override string ToString()

{

return "[Name:" + Name + ",Age:" + Age + "]";

}

}

public class Azi : User

{

public string Like { set; get; }

public Azi(int i) : base(i) { Like = "QiaoFeng"; }

public Azi() { }

public override string ToString()

{

return "[Name:" + Name + ",Age:" + Age + "]";

}

}

public class Product

{

public Supplier Supplier { set; get; }

public decimal GetPrice()

{

return 10;

}

public Product()

{

Supplier = new Supplier();

}

}

public class Supplier

{

public string Name { get; set; }

public Supplier()

{

Name = "伟创力";

}

}

public class ProductDto

{

public string SupplierName { set; get; }

public decimal Price { set; get; }

public override string ToString()

{

return "[Price:" + Price + ",SupplierName:" + SupplierName + "]";

}

}

public class Book : Product

{

public DateTime BuyTime { get { return DateTime.Now; } }

}

public class BookDto : ProductDto

{

public DateTime Date { get; set; }

public int Hour { get; set; }

public int Minute { get; set; }

public override string ToString()

{

return "[Price:" + Price + ",SupplierName:" + SupplierName + ",Date:" + Date + "]";

}

}

public class Source

{

public string Name { get { return "muphy"; } }

public int Width { set; get; }

}

public class Destination

{

public string Name { get; set; }

public string like { get; set; }

public override string ToString()

{

return Name + " Like " + like;

}

}

public class MyProfile : Profile

{

public MyProfile()

{

//类型字段系统

CreateMap<User, UserDto>();

//扁平化映射

CreateMap<Product, ProductDto>();

//投影

CreateMap<Book, BookDto>()

.ForMember(desc => desc.Date, opt => opt.MapFrom(src => src.BuyTime.Date))

.ForMember(desc => desc.Hour, opt => opt.MapFrom(src => src.BuyTime.Hour))

.ForMember(desc => desc.Minute, opt => opt.MapFrom(src => src.BuyTime.Minute));

// 条件映射

CreateMap<Azi, UserDto>().ForMember(desc => desc.Age, opt => opt.MapFrom(src => src.Age < 21));

// 可设置默认值

CreateMap<Source, Destination>().ForMember(desc => desc.like, opt => opt.MapFrom(src => "azi"));

//可忽视多余值

CreateMap< Destination, Source>().ForMember(desc => desc.Width, opt => opt.Ignore());

//空值替换

CreateMap<Source, Destination>().ForMember(desc => desc.Name, opt => opt.NullSubstitute("azi"));

//其他设置

CreateMap<User, Azi>()

.BeforeMap((src,desc) => src.Age += 50)

.AfterMap((src,desc)=>desc.Age -= 35)// 设置转换前后的行为

.ReverseMap();//设置反向映射

}

}

// 1.简单静态的方式初始化映射 使用Initialize 两个方法：CreateMap AddProfile（稍后）

public static UserDto AutoMapperConfig1()

{

//Mapper.Initialize(cfg => cfg.CreateMap<User, UserDto>());

//Mapper.Initialize(cfg => cfg.AddProfile(new MyProfile()));

//Mapper.Initialize(cfg => cfg.AddProfile<MyProfile>());

UserDto dto = Mapper.Map<UserDto>(new User(1));

return dto;

}

// 2.简单实例的方式初始化方式映射 使用MapperConfiguration 也有两种方式：CreateMapper方法和Mpapper构造函数

public static UserDto AutoMapperConfig2()

{

MapperConfiguration config = new MapperConfiguration(fig => fig.CreateMap<User, UserDto>());

var mapper = config.CreateMapper();

//IMapper mapper = new Mapper(config);

return mapper.Map<UserDto>(new User(2));

}

// 3.使用继承Profile的配置属性方式映射 AddProfile方法

public static UserDto AutoMapperConfig3()

{

MapperConfiguration config = new MapperConfiguration(fig => fig.AddProfile<MyProfile>());

IMapper mapper = new Mapper(config);

return mapper.Map<UserDto>(new User(3));

}

// 4.使用继承Profile的配置属性 + 泛型方式映射 效果会更好

public static T AutoMapperConfig4<K, T>(K k)

{

return Mapper.Map<T>(k);

}

// 5.扁平化映射

public static ProductDto AutoMapperConfig5()

{

MapperConfiguration config = new MapperConfiguration(fig => fig.AddProfile<MyProfile>());

var mapper = config.CreateMapper();

return mapper.Map<ProductDto>(new Product());

}

}

}