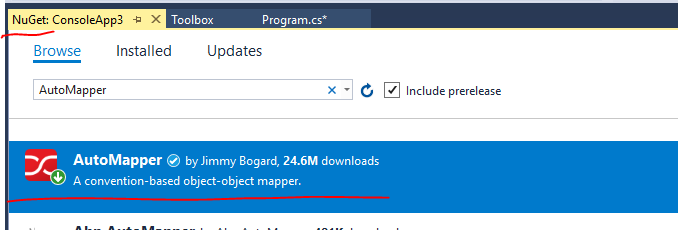
AutoMapper Learning

安装AutoMapper



public static void Initialize(Action<IMapperConfigurationExpression> config);

public static void Initialize(MapperConfigurationExpression config);

public static TDestination Map<TSource, TDestination>(TSource source, TDestination destination);

public static TDestination Map<TSource, TDestination>(TSource source, TDestination destination, Action<IMappingOperationOptions<TSource, TDestination>> opts);

public static TDestination Map<TSource, TDestination>(TSource source);

public static object Map(object source, Type sourceType, Type destinationType, Action<IMappingOperationOptions> opts);

public static object Map(object source, object destination, Type sourceType, Type destinationType);

public static object Map(object source, object destination, Type sourceType, Type destinationType, Action<IMappingOperationOptions> opts);

public static TDestination Map<TDestination>(object source, Action<IMappingOperationOptions> opts);

public static TDestination Map<TDestination>(object source);

public static TDestination Map<TSource, TDestination>(TSource source, Action<IMappingOperationOptions<TSource, TDestination>> opts);

public static object Map(object source, Type sourceType, Type destinationType);

public static void Reset();

AutoMapper 初始化：

public class User

{

public string FName { get; set;}

public string LName { get; set; }

public string Address { get; set; }

public int Level { get; set;}

public DateTime Birthday { get; set; }

}

public class UserPro

{

public string fname { get; set; }

public int level { get; set; }

}

**使用静态 Mapper 类的方法**

Mapper.Initialize(

cfg => cfg.CreateMap<User, UserPro>()

.BeforeMap((p, q) => Console.WriteLine("Before Map:{0} - {1}", p.Level, q.level))

.AfterMap((p, q) => { q.level += 1000; Console.WriteLine("After Map:{0} - {1}", p.Level, q.level); })

);

UserPro upp = Mapper.Map<UserPro>(u);

**使用对象：**

MapperConfiguration mconf = new MapperConfiguration(cfg => cfg.CreateMap<User, UserPro>());

var mmp = mconf.CreateMapper();

IMapper mp = new Mapper(mconf);

UserPro upp1 = mp.Map<UserPro>(u);

**Initialization**

You now must use either Mapper.Initialize or new MapperConfiguration() to initialize AutoMapper. If you prefer to keep the static usage, use Mapper.Initialize.

If you have a lot of Mapper.CreateMap calls everywhere, move those to a Profile, or into Mapper.Initialize, called once at startup.

Mapper.Initialize(cfg => {

cfg.AddProfile<UserMapPF>(); 一个 Profile 包含多个映射

});

注意：如果使用静态Mapper.Initialize 只能执行一次，否则会抛出错误。

如果想配置文件根据情况，有各种不同应用的配置，可以使用对象的方式。

MapperConfiguration mconf = new MapperConfiguration(

cfg => cfg.CreateMap<User, UserPro>()

);

var mmp = mconf.CreateMapper();

**AssertConfigurationIsValid**

Mapper.AssertConfigurationIsValid();

如果目标有属性没有被映射到，则会抛出错误， 如果源数据有属性没有被使用则不会抛出错误。

如果目标属性没有设置映射，可以使用 Ignore() 来忽略映射，避免抛出异常

## cfg.CreateMap<Employee, Manager>(MemberList.Source) – 只检查源数据对象的所有属性是否有映射。不检查目标对象属性是否有映射关系。

cfg.CreateMap<Employee, Manager>(MemberList.Source)

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

;

## Employee -> Manager (Source member list) Unmapped properties: fullname

## 出错： Source.FullName 并没有映射到 Dest.FullName, 因为它已经被映射了

## 如果想要源数据的某个属性跳过检查， 可以使用

cfg.CreateMap<Employee, Manager>(MemberList.Source)

## .ForSourceMember(s=>s.fullname, opt=>opt.Ignore())

cfg.CreateMap<Employee, Manager>(**MemberList.Destination**) – 只检查目标属性是否映射，不检查源

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

.ForMember(dst=>dst.Level, src=>src.Ignore());

cfg.CreateMap<Employee, Manager>(**MemberList.None**)

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

//.ForMember(dst=>dst.Level, src=>src.Ignore())

;

## Profile的用法

Profile提供了一个命名的映射类，所有继承自Profile类的子类都是一个映射集合。

我们来看一下Profile的用法，这个例子中仍然使用上面的Source类和Destination类。

注意： 旧版本是通过 protected override void Configure(), 在新版本里取消了，直接放在构造函数里完成映射定义

public class MapProfile: Profile

{

public MapProfile():base("GoodProfile")

{

CreateMap<User, UserPro>(MemberList.Destination).ReverseMap()

.BeforeMap((s, d) => Console.WriteLine("MapProfile Before: {0} - {1}", s.fname, d.FName))

.AfterMap((s, d) => Console.WriteLine("MapProfile After: {0} - {1}", s.fname, d.FName));

RecognizePrefixes("m\_");

}

}

我们可以在Profile中重写Configure方法，从而完成映射规则的配置。从Profile初始化Mapper规则：

public class User

{

public string FName { get; set;}

public string LName { get; set; }

public string Address { get; set; }

public int Level { get; set;}

public DateTime Birthday { get; set; }

}

public class UserPro

{

public string fname { get; set; }

public int level { get; set; }

public int age { get; set; }

}

public class Rule

{

public string RuleName { get; set; }

public int width { get; set; }

public int height { get; set; }

public Range size { get; set; }

}

public class RuleDto

{

public int Width { get; set; }

public int Height { get; set; }

public int Size { get; set; }

}

public enum Range {

low = 1,

medium = 2,

high = 3

}

可以把多个Mapping 放到一个Profile 里， 也可以单独分开

public class UserMapPF: Profile

{

public UserMapPF():base()

{

CreateMap<User, UserPro>()

.BeforeMap((s, d) => Console.WriteLine("UserProfile Before: {0} - {1}", s.FName, d.fname))

.AfterMap((s, d) => Console.WriteLine("UserProfile After: {0} - {1}", s.FName, d.fname));

CreateMap<Rule, RuleDto>()

.BeforeMap((s, d) => Console.WriteLine("MapProfile Before: {0} - {1}", s.size, d.Size))

.AfterMap((s, d) => Console.WriteLine("MapProfile After: {0} - {1}", s.size, d.Size));

}

}

User user1 = new User()

{

FName = "William",

LName = "Liu",

Address = "D3 Security",

Level = 9,

Birthday = DateTime.Now

};

Rule rule1 = new Rule

{

RuleName = "Good Rule",

width = 101,

height = 202,

size = Range.high

};

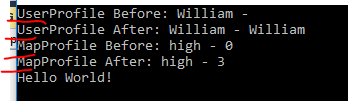
Mapper.Initialize(cfg => {

cfg.AddProfile<UserMapPF>(); 一个 Profile 包含多个映射

});

UserPro user2 = Mapper.Map<UserPro>(user1);

RuleDto rule2 = Mapper.Map<RuleDto>(rule1);



如果单独分开 Profile:

public class UserMapPF: Profile

{

public UserMapPF():base()

{

CreateMap<User, UserPro>()

.BeforeMap((s, d) => Console.WriteLine("UserProfile Before: {0} - {1}", s.FName, d.fname))

.AfterMap((s, d) => Console.WriteLine("UserProfile After: {0} - {1}", s.FName, d.fname));

}

}

public class RuleMapPF: Profile

{

public RuleMapPF()

{

CreateMap<Rule, RuleDto>()

.BeforeMap((s, d) => Console.WriteLine("MapProfile Before: {0} - {1}", s.size, d.Size))

.AfterMap((s, d) => Console.WriteLine("MapProfile After: {0} - {1}", s.size, d.Size));

}

}

Mapper.Initialize(cfg =>

{

cfg.AddProfile<UserMapPF>();

cfg.AddProfile<RuleMapPF>();

});

UserPro user2 = Mapper.Map<UserPro>(user1);

RuleDto rule2 = Mapper.Map<RuleDto>(rule1);

注意： 一个有趣的现象

Mapper.Initialize(cfg =>

{

cfg.AddProfile<UserMapPF>(); -- 如果只添加一个 Profile

});

UserPro user2 = Mapper.Map<UserPro>(user1);

RuleDto rule2 = Mapper.Map<RuleDto>(rule1); -- 此Profile并没有被添加，也一样可以映射成功

### Profiles设置

除了使用以上两总方式类配置映射关系，也可以使用Profie配置来实现映射关系。

创建自定义的Profile需要继承Profile类：

|  |  |
| --- | --- |
| public class MyProfile : Profile  {      public MyProfile()      {          CreateMap<ProductEntity, ProductDTO>();          // Other mapping configurations      }  } |  |

例子：

|  |  |
| --- | --- |
| [TestMethod]  public void TestProfile()  {      var productEntity = new ProductEntity()      {          Name = "Product" + DateTime.Now.Ticks,          Amount = 10      };        var configuration = new MapperConfiguration(cfg => cfg.AddProfile<MyProfile>());      var productDTO = configuration.CreateMapper().Map<ProductDTO>(productEntity);        Assert.IsNotNull(productDTO);      Assert.IsNotNull(productDTO.Name);      Assert.IsTrue(productDTO.Amount > 0);  } |  |

除了使用AddProfile，也可以使用AddProfiles添加多个配置；同样，可以同时使用Mapper和Profile，也可以添加多个配置：

var configuration = new MapperConfiguration(cfg =>

{

    cfg.AddProfile<MyProfile>();

    cfg.CreateMap<ProductEntity, ProductDTO>();

});

**扁平化映射**

AutoMapper先映射名字一致的字段，如果没有，则会尝试使用以下规则来映射：

* 目标中字段去掉前缀“Get”后的部分
* 分割目标字段(根据Pascal命名方式)为单个单词

**Source:**

public DateTime GETAmount()

{

return DateTime.Now.AddDays(-10);

}

**Target:**

public string AMOUNT { get; set; }

只要可以兼容转化： 则不会出错：

Source:

public int GETAmount() - 可以是方法，也可以是 property like { get { return 100; } }

{

return 100+200; return (int)100 -> (byte)100 则不抛出错误

}

Target:

public byte AMOUNT { get; set; }

抛出错误： int -> byte 从大转化成小，超界

Source:

public int GETAmount()

{

return 100;

}

Target:

public int getamount { get; set; } – 同时被赋值

public byte AMOUNT { get; set; } – 同时被赋值

同时被赋值

* 分割目标字段(根据Pascal命名方式)为单个单词

public class Person

{

public string FullName { get; set; }

public Address MyAddr { get; set; }

}

public class Parent

{

public string fullname { get; set; }

public string MyAddrCountry { get; set; } - 但是Country必须是大写字符开头

public int myaddrState { get; set; }

public string myaddrCity { get; set; }

}

public class Address {

public string country { get; set; } - 即使这里定义是小写字符

public int state { get; set; }

public string city { get; set; }

}

MyAddr.country -> myaddrCountry ok

MyAddr.country -> myaddrCOUNTRY ok

MyAddr.country -> MYADDRCOUNTRY ok

MyAddr.country -> myADDrCOUNTRY ok

MyAddr.country -> myddRCOUNTRY fail

MyAddr.country -> myaddRCOUNTRY ok

MyAddr.country -> MyAddrCOuntry ok

MyADDR.country -> MyaddRCOuntry fail

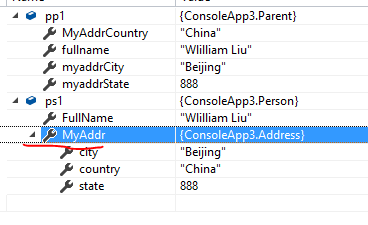
MyADDR.country -> MyaddRCountry ok

MyADDR.country -> MYADDRCountry ok

MyADDR.country -> MYaddrCOuntry ok

总结： 只要 Country 是 Capital, 前缀的对象名称则大小写无所谓。

反向映射也是可以的：

Person ps1 = new Person

{

FullName = "Tommy Yang",

};

Parent pp1 = new Parent

{

fullname = "Wlilliam Liu",

MyAddrCountry = "China",

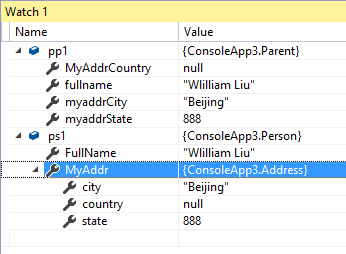
myaddrState = 888,

myaddrCity = "Beijing"

};

Mapper.Map<Parent, Person>(pp1, ps1);

-----------------------------------------------------------------



Person ps1 = new Person

{

FullName = "Tommy Yang", - 会被覆盖

MyAddr = new Address - 对象会被覆盖

{

country = "USA",

state = 1020

}

};

Parent pp1 = new Parent

{

fullname = "Wlilliam Liu",

myaddrState = 888,

myaddrCity = "Beijing"

};

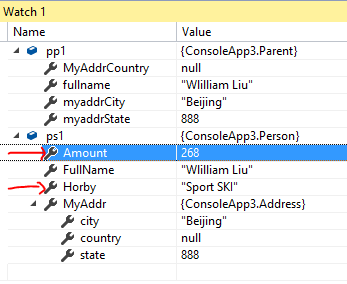
Mapper.Map<Parent, Person>(pp1, ps1);

-------------------------------------------------------------------

public class Person

{

public string FullName { get; set; }

 public Address MyAddr { get; set; }

public int Amount { get; set; }

public string Horby { get; set; }

}

Person ps1 = new Person

{

FullName = "Tommy Yang",

Amount = 268, - 由于源数据没有此属性，所以保留

Horby = "Sport SKI", - 由于源数据没有此属性，所以保留

MyAddr = new Address

{

country = "USA",

state = 1020

}

};

Parent pp1 = new Parent

{

fullname = "Wlilliam Liu",

myaddrState = 888,

myaddrCity = "Beijing"

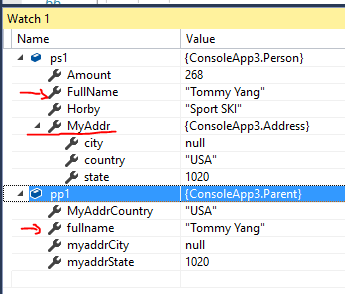
};

Mapper.Map<Parent, Person>(pp1, ps1);

正向映射：

--------------------------------------------------------------------

Person ps1 = new Person

{

FullName = "Tommy Yang",

Amount = 268,

Horby = "Sport SKI",

MyAddr = new Address

{

country = "USA",

state = 1020

}

};

Parent pp1 = new Parent

{

fullname = "Wlilliam Liu",

myaddrState = 888,

myaddrCity = "Beijing"

};

Mapper.Map<Person, Parent>(ps1, pp1);

**集合验证**

AutoMapper除了可以映射单个对象外，也可以映射集合对象。AutoMapper源集合类型支持以下几种：

* IEnumerable
* IEnumerable<T>
* ICollection
* ICollection<T>
* IList
* IList<T>
* List<T>
* Arrays

对于集合的映射关系， 只是定义类型映射即可：

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Person, Parent>().ReverseMap();

});

注意：对于集合映射，不需要使用集合，只需要集合的类型即可。

cfg.CreateMap<Person[], List<Parent>>().ReverseMap();

public class Person

{

public string FullName { get; set; }

public Address MyAddr { get; set; }

public int Amount { get; set; }

public string Horby { get; set; }

}

public class Parent

{

public string fullname { get; set; }

public int level { get; set; }

public string MyAddrCountry { get; set; }

public int myaddrState { get; set; }

public string myaddrCity { get; set; }

}

public class Address {

public string country { get; set; }

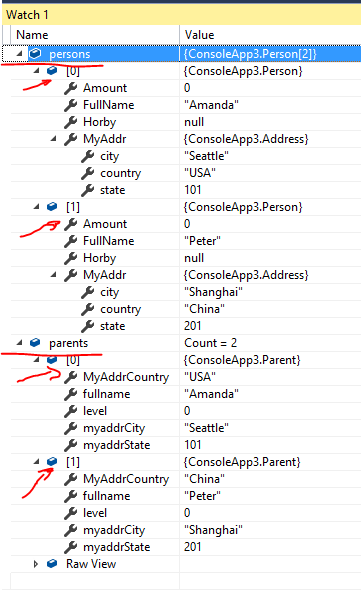
public int state { get; set; }

public string city { get; set; }

}

-----------------------------------------------------------------------------------------------------

Person[] persons = new Person[]

{

new Person

{

FullName="Amanda",

MyAddr = new Address

{

country = "USA",

state = 101,

city = "Seattle"

}

},

new Person

{

FullName="Peter",

MyAddr = new Address

{

country = "China",

state = 201,

city = "Shanghai"

}

}

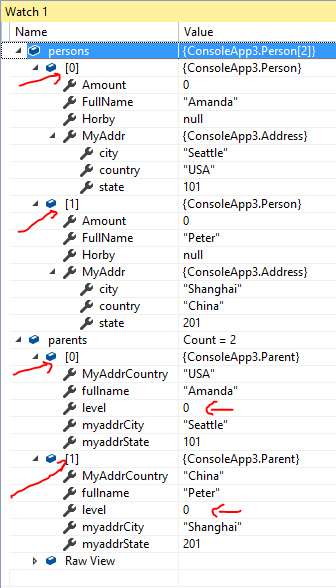
};

IList<Parent> parents = Mapper.Map<IList<Parent>>(persons);

集合没有合并的问题，按照源集合转化成目标集合。不存在合并计算的问题

Mapper.Initialize(cfg =>

{

 cfg.CreateMap<Person, Parent>().ReverseMap();

});

Person[] persons = new Person[] - 源集合只有 2 条记录

{

new Person

{

FullName="Amanda",

MyAddr = new Address

{

country = "USA",

state = 101,

city = "Seattle"

}

},

new Person

{

FullName="Peter",

MyAddr = new Address

{

country = "China",

state = 201,

city = "Shanghai"

}

}

};

IList<Parent> parents=new List<Parent>-目标集合有4条记录

{

new Parent{level = 800 },

new Parent{level = 801 },

new Parent{level = 802 },

new Parent

{

level = 803,

fullname = "Tommy",

myaddrCity = "Vancouver",

myaddrState = 399,

MyAddrCountry = "Canada"

}

};

Mapper.Map<Person[], IList<Parent>>(persons, parents);

从结果可以看出， 并不是对目标做合并计算，只是根据源集合转化成目标集合。

原来的目标集合被完全覆盖掉。

复杂类型：

public class Teacher

{

public Teacher()

{

this.Kids = new List<Student>();

}

public string FullName { get; set; }

public string AddrCountry { get; set; } - 拆分映射 Guard.addr.country

public string AddrState { get; set; } - 拆分映射 Guard.addr.state

public string AddrCity { get; set; } - 拆分映射 Guard.addr.city

public List<Student> Kids { get; set;} - 必须同名映射，大小写无关

public void addStud(string name, int score, int level)

{

this.Kids.Add(new Student { Name = name, Score = score, Level = level });

}

}

public class Student

{

public string Name { get; set; }

public int Score { get; set; }

public int Level { get; set; } -- 多一个属性

}

public class Guard

{

public Guard()

{

this.kids = new List<Child>();

}

public string fullname { get; set; }

public Address addr { get; set; }

public List<Child> kids { get; set; }

public void addChild(string name)

{

this.kids.Add(new Child { name = name });

}

}

public class Child

{

public string name { get; set; }

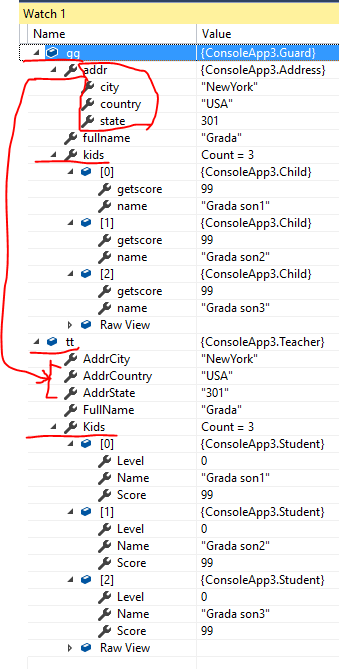
public int getscore { get { return 99; } }

}

-------------------------------------------------------------

Mapper.Initialize(cfg =>

{

 cfg.CreateMap<Guard, Teacher>().ReverseMap();

cfg.CreateMap<Child, Student>().ReverseMap();

- 子类型也需要定义映射， 否则出错

});

Guard gg = new Guard

{

fullname = "Grada",

addr = new Address

{

country = "USA",

state = 301,

city = "NewYork"

}

};

gg.addChild("Grada son1");

gg.addChild("Grada son2");

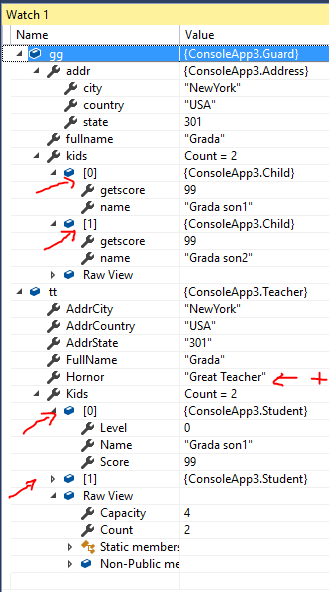
gg.addChild("Grada son3");

Teacher tt = Mapper.Map<Teacher>(gg);

合并类型：

对于复杂类型里的集合类型，不会执行合并，只是源集合转化并覆盖目标集合

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Guard, Teacher>().ReverseMap();

cfg.CreateMap<Child, Student>().ReverseMap();

});

Guard gg = new Guard

{

fullname = "Grada",

addr = new Address

{

country = "USA",

state = 301,

city = "NewYork"

}

};

gg.addChild("Grada son1");

gg.addChild("Grada son2");

Teacher tt = new Teacher

{

FullName = "Teacher Yuan",

AddrCity = "Beijing",

AddrState = "990",

AddrCountry = "China",

Hornor = "Great Teacher"

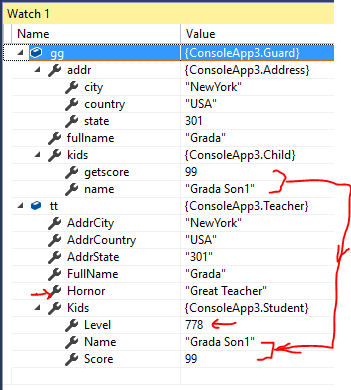
};

tt.addStud("Tea Stu1", 333, 501);

tt.addStud("Tea Stu2", 444, 502);

tt.addStud("Tea Stu3", 555, 503);

Mapper.Map<Guard,Teacher>(gg, tt);

如果是类型里面的类型

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Guard, Teacher>().ReverseMap();

cfg.CreateMap<Child, Student>().ReverseMap();

});

Guard gg = new Guard

{

fullname = "Grada",

addr = new Address

{

country = "USA",

state = 301,

city = "NewYork"

},

kids = new Child { name = "Grada Son1" }

};

Teacher tt = new Teacher

{

FullName = "Teacher Yuan",

AddrCity = "Beijing",

AddrState = "990",

AddrCountry = "China",

Hornor = "Great Teacher",

Kids = new Student { Name = "Tea Stu1", Score = 556, Level = 778 }

};

Mapper.Map<Guard,Teacher>(gg, tt);

## 投影及条件映射

### 投影(指定字段)

除了以上使用的自动映射规则，AutoMapper还可以指定映射方式。下面使用ForMemeber指定字段的映射，将一个时间值拆分映射到日期、时、分：

public class Calendar

{

    public DateTime CalendarDate { get; set; }

    public string Title { get; set; }

}

public class CalendarModel

{

    public DateTime Date { get; set; }

    public int Hour { get; set; }

    public int Minute { get; set; }

    public string Title { get; set; }

}

[TestMethod]

public void TestProjection()

{

    var calendar = new Calendar()

    {

        Title = "2018年日历",

        CalendarDate = new DateTime(2018, 1, 1, 11, 59, 59)

    };

    Mapper.Initialize(cfg => cfg

        .CreateMap<Calendar, CalendarModel>()

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

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

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

    var calendarModel = Mapper.Map<CalendarModel>(calendar);

    Assert.AreEqual(calendarModel.Date.Ticks, new DateTime(2018, 1, 1).Ticks);

    Assert.AreEqual(calendarModel.Hour, 11);

    Assert.AreEqual(calendarModel.Minute, 59);

}

自定义字段投影：

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Guard, Teacher>()

.ForMember(dst=>dst.FName, opt=>opt.MapFrom(p=>p.fullname))

.ForMember(dst=>dst.cnty, opt=>opt.MapFrom(p=>p.addr.country))

.ForMember(dst=>dst.st, opt=>opt.MapFrom(p=>p.addr.state))

.ForMember(dst=>dst.ct, opt=>opt.MapFrom(p=>p.addr.city))

.ReverseMap();

cfg.CreateMap<Child, Student>()

.ForMember(dst=>dst.Level, opt=>opt.MapFrom(p=>p.getscore))

.ReverseMap();

});

**AutoMapper的匹配**

     1，智能匹配

　　　 　AutoMapper能够自动识别和匹配大部分对象属性:

* + 如果源类和目标类的属性名称相同，直接匹配，不区分大小写
  + 目标类型的CustomerName可以匹配源类型的Customer.Name
  + 目标类型的Total可以匹配源类型的GetTotal()方法

    2，自定义匹配

　　　　Mapper.CreateMap<CalendarEvent, CalendarEventForm>()

               //属性匹配，匹配源类中WorkEvent.Date到EventDate

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

//忽略目标类中的属性

　　　　.ForMember(dest => dest.SomeValue, opt => opt.Ignore())

//复杂的匹配

　　　　.ForMember(dest => dest.TotalAmount, opt => opt.MapFrom(src => src.TotalAmount ?? 0))

//固定值匹配

　　　　.ForMember(dest => dest.OrderDate, opt => opt.UserValue<DateTime>(DateTime.Now));

MapFrom

映射字段：

.ForMember(dst => dst.cnty, opt => opt.MapFrom(p => p.addr.country))

可以是表达式：

.ForMember(dst => dst.st, opt => opt.MapFrom(p => p.addr.state + "dldld"))

忽略映射：是指不管源数据有没有，都不映射，但是目标数据的属性值还是保留目标原值

.ForMember(dst => dst.Age, opt => opt.Ignore())

先忽略再映射： 还是正常工作

.ForMember(dst => dst.Age, opt => opt.Ignore())

.ForMember(dst => dst.Age, opt => opt.MapFrom(s => s.age))

.ForMember(dst => dst.Age, opt => opt.Ignore())

.ForMember(dst => dst.Age, opt => opt.MapFrom(s => s.Hornor)) – 再映射其他的也可以

// 实际应用，没有必要先 Ignore 再映射别的。 直接映射别的即可。

NullSubstitute

空值替换： 注意只是验证源数据的空值，并不验证目标是否是空值

public class Manager

{

public string FullName { get; set; }

public string Title { get; set; }

public int Level { get; set; }

}

public class Employee

{

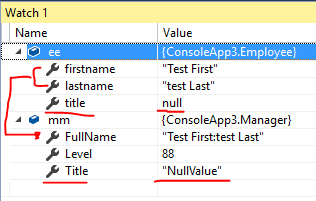
public string firstname { get; set; }

public string lastname { get; set; }

public string title { get; set; }

}

----------------------------------------------------------------

Source.title = null

Employee ee = new Employee

{

firstname = "Test First",

lastname = "test Last",

title = null

};

Manager mm = new Manager

{

FullName = "Manager Name",

Title = "CEO",

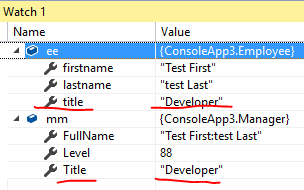
Level = 88

};

Mapper.Map<Employee, Manager>(ee, mm);

-----------------------------------------------------------------

Source.title = "Developer"



Employee ee = new Employee

{

firstname = "Test First",

lastname = "test Last",

title = "Developer"

};

Manager mm = new Manager

{

FullName = "Manager Name",

Level = 88,

Title = "CEO"

};

Mapper.Map<Employee, Manager>(ee, mm);

-----------------------------------------------------------------

Source.title not defined

public class Manager

{

public string FullName { get; set; }

public string Title { get; set; }

public int Level { get; set; }

}

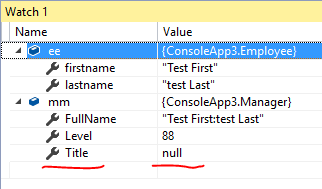
public class Employee

{

public string firstname { get; set; }

public string lastname { get; set; }

}

Employee ee = new Employee

{

firstname = "Test First",

lastname = "test Last",

};

Manager mm = new Manager

{

FullName = "Manager Name",

Level = 88

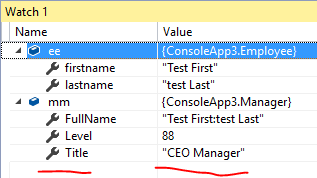
};

Mapper.Map<Employee, Manager>(ee, mm);

此案例证明，不验证目标属性是否为 null 值

------------------------------------------------

Destination.Title = "CEO Manager"



Employee ee = new Employee

{

firstname = "Test First",

lastname = "test Last",

};

Manager mm = new Manager

{

FullName = "Manager Name",

Level = 88,

Title = "CEO Manager"

};

Mapper.Map<Employee, Manager>(ee, mm);

UseValue – 使用固定值填充目标属性

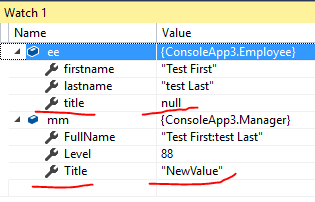
cfg.CreateMap<Employee, Manager>()

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst=>dst.Title, src=>src.UseValue("NewValue"))

.ForMember(dst => dst.Title, src => src.NullSubstitute("NullValue"))

;



UseValue 比NullSubstitute 占上风

## UseDestinationValue

UseDestinationValue告诉AutoMapper不要为某个成员创建一个新的对象，而是使用目标对象的现有属性。 它以前是默认的。

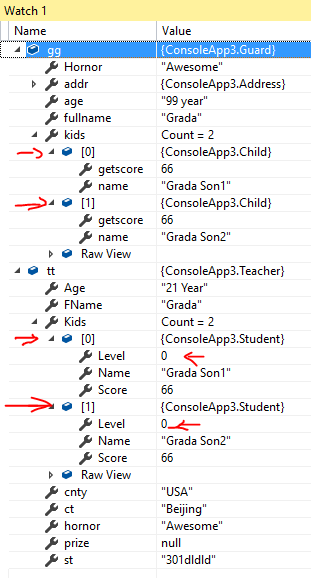
对于属性是对象， 其实是使用合并操作，

对于属性是集合，其实是使用源数据集合来转化成目标数据集合（是新的目标集合对象，原来的目标集合对象被删除覆盖了）

----------------------------------------------------------------------

集合的案例

Guard gg = new Guard

{

age= "99 year",

fullname = "Grada",

Hornor = "Awesome",

addr = new Address

{

country = "USA",

state = 301,

city = "NewYork"

},

};

gg.Add("Grada Son1");

gg.Add("Grada Son2");

Teacher tt = new Teacher

{

Age = "21 Year",

FName = "Teacher Yuan",

ct = "Beijing",

st = "990",

cnty = "China",

};

tt.Add("Team1", 1001, 2001);

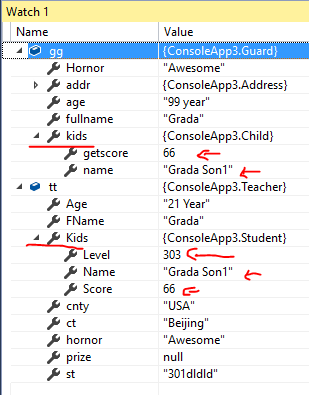
tt.Add("Team2", 1002, 2002);

tt.Add("Team3", 1003, 2003);

tt.Add("Team4", 1004, 2004);

Mapper.Map(gg, tt);

------------------------------------------------

对象的案例

Guard gg = new Guard

{

age= "99 year",

fullname = "Grada",

Hornor = "Awesome",

addr = new Address

{

country = "USA",

state = 301,

city = "NewYork"

},

kids = new Child { name = "Grada Son1" }

};

Teacher tt = new Teacher

{

Age = "21 Year",

FName = "Teacher Yuan",

ct = "Beijing",

st = "990",

cnty = "China",

Kids = new Student {

Name = "Teams",

Score = 876,

Level =303 }

};

Mapper.Map(gg, tt);

ResolveUsing:

void ResolveUsing<TValueResolver, TSourceMember>(Expression<Func<TSource, TSourceMember>> sourceMember) where TValueResolver : IMemberValueResolver<TSource, TDestination, TSourceMember, TMember>;

void ResolveUsing<TValueResolver>() where TValueResolver : IValueResolver<TSource, TDestination, TMember>;

void ResolveUsing<TResult>(Func<TSource, TDestination, TMember, ResolutionContext, TResult> resolver);

void ResolveUsing<TResult>(Func<TSource, TDestination, TMember, TResult> resolver);

案例：

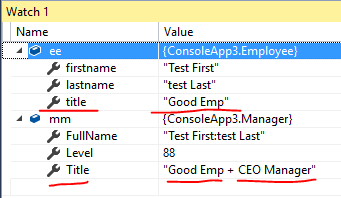
cfg.CreateMap<Employee, Manager>()

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

-----------------------------------------------

Employee ee = new Employee

{

title = "Good Emp",

firstname = "Test First",

lastname = "test Last",

};

Manager mm = new Manager

{

FullName = "Manager:Name",

Level = 88,

Title = "CEO Manager"

};

Mapper.Map<Employee, Manager>(ee, mm);

ReverseMap

cfg.CreateMap<Employee, Manager>()

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.NullSubstitute("NullValue"))

.ReverseMap() - 反向

.ForMember(dst => dst.firstname, src => src.MapFrom(s => s.FullName)) –反向映射

.ForMember(dst => dst.lastname, src => src.MapFrom(s => s.FullName)) –反向映射

;

Config:

## 值转换: 属于全局设置

如果配置了值转换，AutoMapper会将修改转换后的值以符合配置的规则。比如，配置目标对象中的值添加符号“@@”：

Mapper.Initialize(cfg =>

{

cfg.CreateMap<Guard, Teacher>(MemberList.None)

.ForMember(dst => dst.cnty, opt => opt.MapFrom(p => p.addr.country))

.ForMember(dst => dst.st, opt => opt.MapFrom(p => p.addr.state + "dldld"))

.ForMember(dst => dst.Age, opt => opt.Ignore())

.ForMember(dst => dst.FName, opt => opt.MapFrom(p => p.fullname))

.ForMember(dst=>dst.Kids, opt=>opt.UseDestinationValue())

;

cfg.CreateMap<Child, Student>(MemberList.None)

.ReverseMap();

cfg.CreateMap<Employee, Manager>(MemberList.None)

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

;

cfg.**ValueTransformers**.Add<string>(val => $"@{val}@d3.com");

cfg.**ValueTransformers**.Add<Student>(val => new Student {Name = "FiveStar", Level=665, Score = val.Score + 8000 });

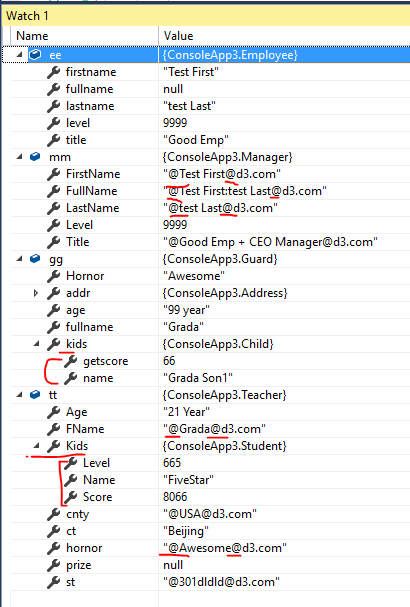
});

Mapper.AssertConfigurationIsValid();

第一 所有字符类型的值都被转化成 @{val}@d3.com

第二 所有类型Student都被转化生成新的对象：

{Name = "FiveStar", Level=665, Score = val.Score + 8000 }



### **条件映射：可以应用于单个Member 或者 AllMembers 或者AllOtherMembers 上**

 有些情况下，我们会考虑添加映射条件，比如，某个值不符合条件时，不允许映射。针对这种情况可以使用ForMember中的Condition

cfg.CreateMap<Employee, Manager>(MemberList.Destination)

.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

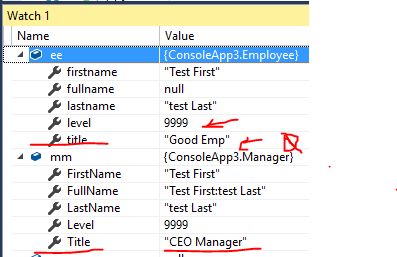
.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

.ForMember(dst=>dst.Title, src=>src.Condition(s=>s.level==88))

;

.ForMember(dst=>dst.Title, src=>src.Condition(s=>s.level==88)) – 这是不符合条件

Employee ee = new Employee

{

title = "Good Emp",

firstname = "Test First",

lastname = "test Last",

level = 9999

};

Manager mm = new Manager

{

FullName = "Manager:Name",

Level = 88,

Title = "CEO Manager"

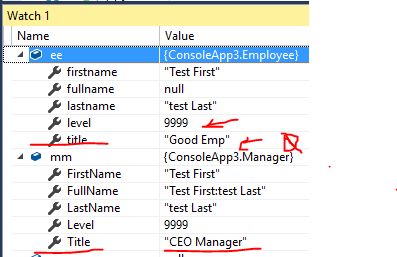
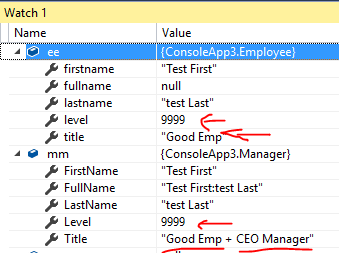
};

Mapper.Map<Employee, Manager>(ee, mm);

------------------------------------------------------------------------------------------------

.ForMember(dst=>dst.Title, src=>src.Condition(s=>s.level==9999)) – 这是符合条件

Employee ee = new Employee

{

title = "Good Emp",

firstname = "Test First",

lastname = "test Last",

level = 9999

};

Manager mm = new Manager

{

FullName = "Manager:Name",

Level = 88,

Title = "CEO Manager"

};

Mapper.Map<Employee, Manager>(ee, mm);

对于Member 的条件映射几点说明：

1. 设置判断条件的源属性， 和目标属性可以是没有映射关系的。条件满足则按照事先设置好的映射转化。

2. 如果条件不满足，则目标的属性仍然使用目标的原值。

3. ForMember 所设置的条件，只影响ForMember里设置的属性，对其他的属性映射不产生影响。

ForAllMembers: 的条件是对整个目标对象是否映射，

如果条件为真，这执行映射， 如果为假，则整个目标使用原值

cfg.CreateMap<Employee, Manager>(MemberList.Destination)

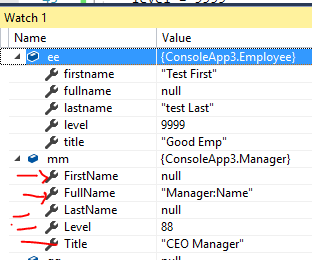
.ForMember(dst=>dst.FullName, src=> src.MapFrom(s=>s.firstname + ":" + s.lastname))

.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

.ForAllMembers(opt=>opt.Condition(src=>src.level==88))

;

.ForAllMembers(opt=>opt.Condition(src=>src.level==88)) – 条件不满足



Employee ee = new Employee

{

title = "Good Emp",

firstname = "Test First",

lastname = "test Last",

level = 9999

};

Manager mm = new Manager

{

FullName = "Manager:Name",

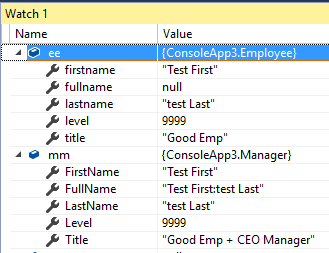
Level = 88,

Title = "CEO Manager"

};

Mapper.Map<Employee, Manager>(ee, mm);

.ForAllMembers(opt=>opt.Condition(src=>src.level==9999)) – 条件满足



ForAllOtherMembers:

定义了映射关系的属性不受影响，条件不满足，则目标（没有映射关系的）属性全部保留原值

cfg.CreateMap<Employee, Manager>(MemberList.Destination)

.ForMember(dst=>dst.FullName, src=>src.MapFrom(s=>s.fullname))

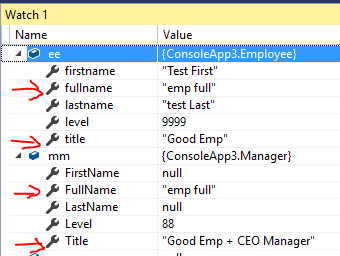
.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

.ForAllOtherMembers(opt=>opt.Condition(src=>src.level==88))

;

.ForAllOtherMembers(opt=>opt.Condition(src=>src.level==88)) – 条件不符合

Employee ee = new Employee

{

title = "Good Emp",

fullname = "emp full",

firstname = "Test First",

lastname = "test Last",

level = 9999

};

Manager mm = new Manager

{

FullName = "Manager:Name",

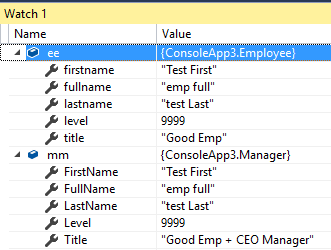
Level = 88,

Title = "CEO Manager"

};

Mapper.Map<Employee, Manager>(ee, mm);

.ForAllOtherMembers(opt=>opt.Condition(src=>src.level==9999)) – 条件符合



cfg.CreateMap<Employee, Manager>(MemberList.Destination)

.ForMember(dst=>dst.FullName, src=>src.MapFrom(s=>s.fullname))

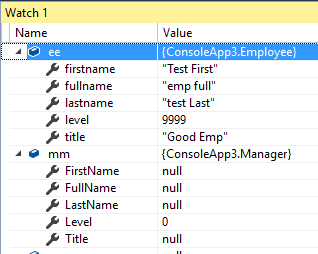
.ForMember(dst => dst.Title, src => src.ResolveUsing( (s, d)=> $"{s.title} + {d.Title}") )

.ForAllMembers(opt=>opt.Condition(src=>src.level==88))

;

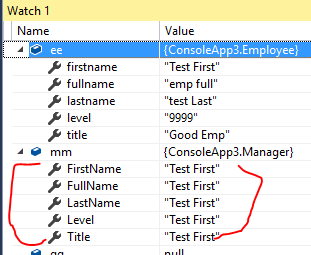
.ForAllMembers(opt=>opt.Condition(src=>src.level==88)) – 条件不符合

var mm = Mapper.Map<Manager>(ee); - 转化为新对象，一切取默认值



.ForAllMembers(opt=>opt.ResolveUsing( src => src.firstname))

在所有成员里这是对所有成员进行转化，逻辑上是没有什么用处的， 如果类型不兼容，则会抛出错误



#### 条件对象映射器

条件对象映射器基于源类型和目标类型之间的条件生成新的类型映射。

首先将User和UserDto的构造函数去掉，配置文件中Configure方法中的代码只写入下面这句代码：

AddConditionalObjectMapper().Where((s, d) => d.Name == s.Name + "Dto");

解释：如果目标类型的名称等于源类型的名称加上“Dto”,那么就生成一个源类型和目标类型的对象映射器。添加了这一句，就不需要CreateMap方法了。

在这连个类中加入属性：

//对象映射规则: 通过以下配置，可以映射所有”目标对象的名称“等于“源对象名称+Model”的类，而不用单个添加CreateMapper映射

cfg.AddConditionalObjectMapper().Where((s, d) => d.Name == s.Name + "Model");

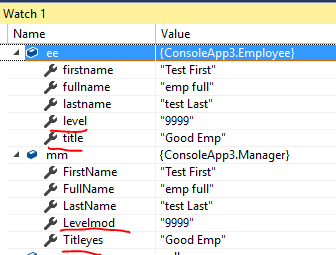
//字段映射规则: 通过以下配置，可以映射“源字段”与“目标字段+Model或Mod”的字段

cfg.AddMemberConfiguration().AddName<PrePostfixName>(\_ => \_.AddStrings(p => p.DestinationPostfixes, "Model", "Mod"));

cfg.AddMemberConfiguration().AddName<PrePostfixName>( s =>

s.AddStrings(p => p.DestinationPostfixes, "Model", "Mod", "Yes"));

Employee ee = new Employee

{

title = "Good Emp",

fullname = "emp full",

firstname = "Test First",

lastname = "test Last",

level = "9999"

};

Manager mm = new Manager

{

FullName = "Manager:Name",

Levelmod = "88",

Titleyes = "CEO Manager"

};

mm = Mapper.Map<Manager>(ee);

cfg.AddConditionalObjectMapper().Where((s, d) => d.Name == s.Name + "Model");

未知如何使用

**ForPath – 反向映射:**

**AutoMapper -** ReverseMap() **通常会自动处理反向映射和反向扁平化 unflatten**

cfg.CreateMap<Order, OrderDto>()

.ForMember(d => d.CustomerName, opt => opt.MapFrom(src => src.Customer.Name))

.ReverseMap();

AutoMapper 会自动反向扁平： (CustomerName => Customer.Name).

但是如果我们想自定义反向映射，可以是用 ForPath:

cfg.CreateMap<Order, OrderDto>()

.ForMember(d => d.CustomerName, opt => opt.MapFrom(src => src.Customer.Name))

.ReverseMap()

.**ForPath**(s => s.Customer.Name, opt => opt.MapFrom(src => src.CustomerName));

cfg.CreateMap<Person, Parent>(MemberList.None)

.ReverseMap()

.**ForMember**(dst=>dst.FullName, opt=>opt.MapFrom(s=>s.fullname + " Reverse"))

;

ForPath 和 ForMember 都可以，实例证明他们是通用的。

**UseDestinationValue 对比 Ignore:**

.ForMember(dst => dst.Kids, opt => opt.Ignore())

.ForMember(dst => dst.Kids, opt => opt.UseDestinationValue())

**Ignore：** 是完全不映射，目标数据属性保持原值

**UseDestinationValue：**这个是默认值，对于属性是类型对象则是使用源数据合并到目标数据。但是对于集合，也是使用源数据集合转化成目标数据集合，并覆盖目标数据集合。

## Inline validation

var source = new Source();

var dest = Mapper.Map<Source, Dest>(source, opt => opt.ConfigureMap(MemberList.None);

Mapper.Initialize(cfg => cfg.ValidateInlineMaps = false);

## Disabling inline maps

Mapper.Initialize(cfg => cfg.CreateMissingTypeMaps = false);

## Handling null collections

Mapper.Initialize(cfg => {

cfg.AllowNullCollections = true;

cfg.CreateMap<Source, Destination>();

});

## Polymorphic element types in collections

public class ParentSource

{

public int Value1 { get; set; }

}

public class ChildSource : ParentSource

{

public int Value2 { get; set; }

}

public class ParentDestination

{

public int Value1 { get; set; }

}

public class ChildDestination : ParentDestination

{

public int Value2 { get; set; }

}

Mapper.Initialize(c=> {

c.CreateMap<ParentSource, ParentDestination>()

.Include<ChildSource, ChildDestination>();

c.CreateMap<ChildSource, ChildDestination>();

});

var sources = new[]

{

new ParentSource(),

new ChildSource(),

new ParentSource()

};

var destinations = Mapper.Map<ParentSource[], ParentDestination[]>(sources);

destinations[0].ShouldBeInstanceOf<ParentDestination>();

destinations[1].ShouldBeInstanceOf<ChildDestination>();

destinations[2].ShouldBeInstanceOf<ParentDestination>();

# **Custom Type Converters**

[Test]

public void Example()

{

Mapper.Initialize(cfg => {

cfg.CreateMap<string, int>().ConvertUsing(s => Convert.ToInt32(s));

cfg.CreateMap<string, DateTime>().ConvertUsing(new **DateTimeTypeConverter**());

cfg.CreateMap<string, Type>().ConvertUsing<**TypeTypeConverter**>();

cfg.CreateMap<Source, Destination>();

});

Mapper.AssertConfigurationIsValid();

var source = new Source

{

Value1 = "5",

Value2 = "01/01/2000",

Value3 = "AutoMapperSamples.GlobalTypeConverters.GlobalTypeConverters+Destination"

};

Destination result = Mapper.Map<Source, Destination>(source);

result.Value3.ShouldEqual(typeof(Destination));

}

public class **DateTimeTypeConverter** : ITypeConverter<string, DateTime>

{

public DateTime Convert(string source, DateTime destination, ResolutionContext context)

{

return System.Convert.ToDateTime(source);

}

}

public class **TypeTypeConverter** : ITypeConverter<string, Type>

{

public Type Convert(string source, Type destination, ResolutionContext context)

{

return Assembly.GetExecutingAssembly().GetType(source);

}

}

------------------------------------------------

Mapper.Initialize(cfg => {

cfg.CreateMap<string, int>().**ConvertUsing**(s => Convert.ToInt32(s));

cfg.CreateMap<string, DateTime>().**ConvertUsing**(new DateTimeTypeConverter());

cfg.CreateMap<string, Type>().**ConvertUsing**<TypeTypeConverter>();

cfg.CreateMap<Source, Destination>();

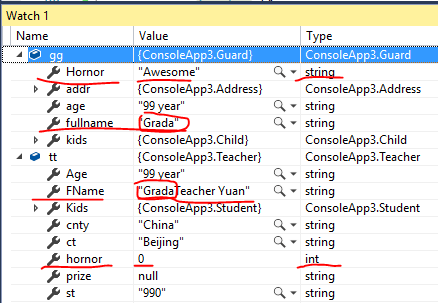
});

cfg.CreateMap<string, int>().**ConvertUsing**(s=> {int it=0; int.TryParse(s, out it); return it;});

cfg.CreateMap<Guard, Teacher>(MemberList.None)

.ForMember(dst => dst.FName, opt => opt.**ResolveUsing**((s,d)=> {return s.fullname + d.FName; }))

;



## 全局类型转换

cfg.CreateMap<string, int>().ConvertUsing<IntTypeConvert>();

cfg.CreateMap<DateTime, ExpireDate>().ConvertUsing<DateTypeConvert>();

public class IntTypeConvert : ITypeConverter<string, int>

{

int ITypeConverter<string, int>.Convert(string source, int destination, ResolutionContext context)

{

int temp = 0;

int.TryParse(source, out temp);

destination += 50000 + temp;

return destination;

}

}

public class DateTypeConvert : ITypeConverter<DateTime, ExpireDate>

{

ExpireDate ITypeConverter<DateTime, ExpireDate>.Convert(DateTime source, ExpireDate destination, ResolutionContext context)

{

destination = new ExpireDate();

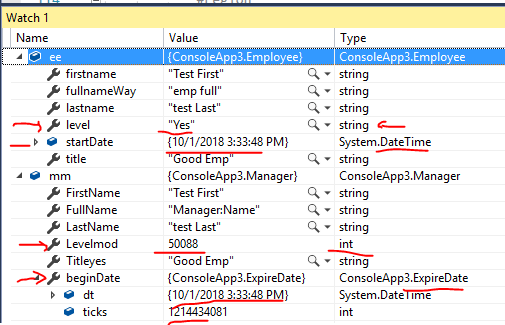
destination.ticks = (int)source.Ticks;

destination.dt = source;

return destination;

}

}



Employee ee = new Employee

{

startDate = DateTime.Now.AddDays(-10),

title = "Good Emp",

fullnameWay = "emp full",

firstname = "Test First",

lastname = "test Last",

level = "Yes"

};

Manager mm = new Manager

{

FullName = "Manager:Name",

Levelmod = 88,

Titleyes = "CEO Manager"

};

mm = Mapper.Map<Manager>(ee);

# **Custom Value Resolvers**

* ResolveUsing<TValueResolver>
* ResolveUsing(typeof(CustomValueResolver))
* ResolveUsing(aValueResolverInstance)

public interface IValueResolver<in TSource, in TDestination, TDestMember>

{

TDestMember Resolve( TSource source,

TDestination destination,

TDestMember destMember,

ResolutionContext context );

}

public class CustomResolver : IValueResolver<Source, Destination, int>

{

public int Resolve(

Source,

Destination,

int member,

ResolutionContext context)

{

return source.Value1 + source.Value2;

}

}

public class NameResolver : IValueResolver<Employee, Manager, int>

{

int IValueResolver<Employee, Manager, int>.Resolve(Employee source, Manager destination, int destMember, ResolutionContext context)

{

destMember = 1000 + destination.Levelmod;

return destMember + 100;

}

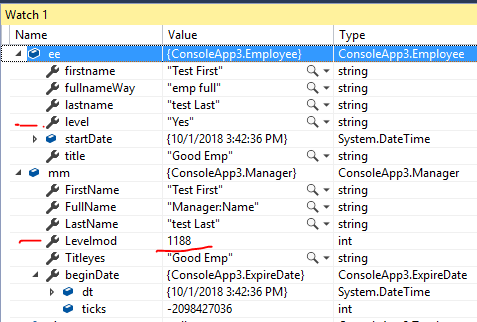
}

cfg.CreateMap<Employee, Manager>(MemberList.None)

.ForMember(dst=>dst.beginDate, opt=>opt.MapFrom(s=>s.startDate))

.ForMember(dst=>dst.Levelmod, opt=>opt.ResolveUsing<NameResolver>())

;



如果有重复定义： IValueResolver 优先级别比 ITypeConverter 的高

## Custom constructor methods

Mapper.Initialize(cfg => cfg.CreateMap<Source, Destination>()

.ForMember( dest => dest.Total,

opt => opt.ResolveUsing(new CustomResolver())

));

## Customizing the source value supplied to the resolver

IMemberValueResolver

public class CustomResolver : IMemberValueResolver<object, object, decimal, decimal> {

public decimal Resolve( object source,

object destination,

decimal sourceMember,

decimal destinationMember,

ResolutionContext context)

{

// logic here

}

}

**自定义字典 key-value 参与到映射中：**

Mapper.CreateMap<Source, Dest>()

.ForMember(dest => dest.Foo,

opt => opt.ResolveUsing((src, dest, destMember, context) => context.Items["Foo"]));

Mapper.Map<Source, Dest>(src, opt => opt.Items["Foo"] = "Bar");

cfg.CreateMap<Employee, Manager>(MemberList.None)

.ForMember(dst=>dst.beginDate, opt=>opt.MapFrom(s=>s.startDate))

.ForMember(dst=>dst.FullName, opt=>opt.ResolveUsing(

(src,dst,dstMember,context) =>

context.Items["myname"] + " & " +

context.Items["hisname"] + " : " +

dstMember + " ~ " +

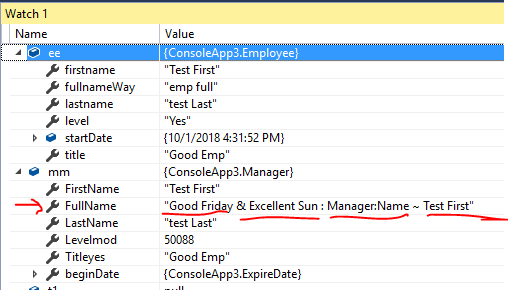
src.firstname

)

);

Mapper.Map<Employee, Manager>(ee, mm, opt=> { opt.Items["myname"] = "Good Friday";

opt.Items["hisname"] = "Excellent Sun"; });



映射的对象内部还包含集合映射：

private void InitMapper()

{

var config = new MapperConfiguration(cfg =>

{

cfg.CreateMap<JObject, McAfeeEventEntity>(MemberList.None)

.ForMember(d => d.Severity, opt => opt.MapFrom(s => s.Path<string>("$.return.severity")))

.ForMember(d => d.IpsId, opt => opt.MapFrom(s => s.Path<string>("$.return.ipsId.id")))

.ForMember(d => d.AlertId, opt => opt.MapFrom(s => s.Path<string>("$.return.alertId")))

.ForMember(d => d.SigId, opt => opt.MapFrom(s => s.Path<string>("$.return.sigId")))

.ForMember(d => d.SigDesc, opt => opt.MapFrom(s => s.Path<string>("$.return.sigDesc")))

.ForMember(d => d.RuleName, opt => opt.MapFrom(s => s.Path<string>("$.return.ruleName")))

.ForMember(d => d.Reviewed, opt => opt.MapFrom(s => s.Path<string>("$.return.reviewed")))

.ForMember(d => d.SubType, opt => opt.MapFrom(s => s.Path<string>("$.return.subtype")))

.ForMember(d => d.FirstTime, opt => opt.MapFrom(s => s.Path<string>("$.return.firstTime")))

.ForMember(d => d.LastTime, opt => opt.MapFrom(s => s.Path<string>("$.return.lastTime")))

.ForMember(d => d.NormId, opt => opt.MapFrom(s => s.Path<string>("$.return.normId")))

.ForMember(d => d.NormMessage, opt => opt.MapFrom(s => s.Path<string>("$.return.normMessage")))

.ForMember(d => d.NormDesc, opt => opt.MapFrom(s => s.Path<string>("$.return.normDesc")))

.ForMember(d => d.SrcUser, opt => opt.MapFrom(s => s.Path<string>("$.return.srcUser")))

.ForMember(d => d.SrcIp, opt => opt.MapFrom(s => s.Path<string>("$.return.srcIp")))

.ForMember(d => d.SrcPort, opt => opt.MapFrom(s => s.Path<string>("$.return.srcPort")))

.ForMember(d => d.SrcMac, opt => opt.MapFrom(s => s.Path<string>("$.return.srcMac")))

.ForMember(d => d.DestIp, opt => opt.MapFrom(s => s.Path<string>("$.return.destIp")))

.ForMember(d => d.DestPort, opt => opt.MapFrom(s => s.Path<string>("$.return.destPort")))

.ForMember(d => d.DestMac, opt => opt.MapFrom(s => s.Path<string>("$.return.destMac")))

.ForMember(d => d.CaseList, opt => opt.MapFrom<CaseListResolver>() )

;

});

this.mapper = config.CreateMapper();

}

public class CaseListResolver : IValueResolver<JObject, McAfeeEventEntity, IList<McAfeeCaseEntity>>

{

public IList<McAfeeCaseEntity> Resolve(

JObject src, McAfeeEventEntity dst, IList<McAfeeCaseEntity> member, ResolutionContext context

) {

member = new List<McAfeeCaseEntity>();

JArray jt = src.Path<JArray>("$.return.cases");

foreach(JObject jo in jt)

{

var obj = new McAfeeCaseEntity();

obj.CaseId = jo.Path<string>("$.id.value");

obj.StatusId = jo.Path<string>("$.statusId.value");

obj.AssignedTo = jo.Path<string>("$.assignedTo");

obj.OrgId = jo.Path<string>("$.orgId");

obj.Severity = jo.Path<string>("$.severity");

obj.Summary = jo.Path<string>("$.summary");

obj.OpenTime = jo.Path<string>("$.openTime");

obj.CloseTime = jo.Path<string>("$.closeTime");

member.Add(obj);

}

return member;

}

}

还有一种方法是使用全局转化：

cfg.CreateMap<JArray, IList<McAfeeCaseEntity>>().ConvertUsing<**CaseListConvert**>();

cfg.CreateMap<JObject, McAfeeEventEntity>(MemberList.None)

.ForMember(d => d.Severity, opt => opt.MapFrom(s => s.Path<string>("$.return.severity")))

.ForMember(d => d.IpsId, opt => opt.MapFrom(s => s.Path<string>("$.return.ipsId.id")))

.ForMember(d => d.AlertId, opt => opt.MapFrom(s => s.Path<string>("$.return.alertId")))

.ForMember(d => d.SigId, opt => opt.MapFrom(s => s.Path<string>("$.return.sigId")))

.ForMember(d => d.SigDesc, opt => opt.MapFrom(s => s.Path<string>("$.return.sigDesc")))

.ForMember(d => d.RuleName, opt => opt.MapFrom(s => s.Path<string>("$.return.ruleName")))

.ForMember(d => d.Reviewed, opt => opt.MapFrom(s => s.Path<string>("$.return.reviewed")))

.ForMember(d => d.SubType, opt => opt.MapFrom(s => s.Path<string>("$.return.subtype")))

.ForMember(d => d.FirstTime, opt => opt.MapFrom(s => s.Path<string>("$.return.firstTime")))

.ForMember(d => d.LastTime, opt => opt.MapFrom(s => s.Path<string>("$.return.lastTime")))

.ForMember(d => d.NormId, opt => opt.MapFrom(s => s.Path<string>("$.return.normId")))

.ForMember(d => d.NormMessage, opt => opt.MapFrom(s => s.Path<string>("$.return.normMessage")))

.ForMember(d => d.NormDesc, opt => opt.MapFrom(s => s.Path<string>("$.return.normDesc")))

.ForMember(d => d.SrcUser, opt => opt.MapFrom(s => s.Path<string>("$.return.srcUser")))

.ForMember(d => d.SrcIp, opt => opt.MapFrom(s => s.Path<string>("$.return.srcIp")))

.ForMember(d => d.SrcPort, opt => opt.MapFrom(s => s.Path<string>("$.return.srcPort")))

.ForMember(d => d.SrcMac, opt => opt.MapFrom(s => s.Path<string>("$.return.srcMac")))

.ForMember(d => d.DestIp, opt => opt.MapFrom(s => s.Path<string>("$.return.destIp")))

.ForMember(d => d.DestPort, opt => opt.MapFrom(s => s.Path<string>("$.return.destPort")))

.ForMember(d => d.DestMac, opt => opt.MapFrom(s => s.Path<string>("$.return.destMac")))

.ForMember(d => d.CaseList, opt => opt.MapFrom(s=>s.Path<JArray>("$.return.cases")))

;

public class **CaseListConvert** : ITypeConverter<JArray, IList<McAfeeCaseEntity>>

{

IList<McAfeeCaseEntity> ITypeConverter<JArray, IList<McAfeeCaseEntity>>.Convert(

JArray ja, IList<McAfeeCaseEntity> caseList, ResolutionContext context

) {

caseList = new List<McAfeeCaseEntity>();

foreach (JObject jo in ja)

{

var obj = new McAfeeCaseEntity();

obj.CaseId = jo.Path<string>("$.id.value");

obj.StatusId = jo.Path<string>("$.statusId.value");

obj.AssignedTo = jo.Path<string>("$.assignedTo");

obj.OrgId = jo.Path<string>("$.orgId");

obj.Severity = jo.Path<string>("$.severity");

obj.Summary = jo.Path<string>("$.summary");

obj.OpenTime = jo.Path<string>("$.openTime");

obj.CloseTime = jo.Path<string>("$.closeTime");

caseList.Add(obj);

}

return caseList;

}

}