ObjectMapper Overview

리얼웹	개발본부	배성혁	2010-08-26 오전 11:03

Object Mapper란 서로 다른 Object (수형이 같던, 다르건) 간에 정보를 복사하는 기능을 수행합니다.

Mapping 이란 말에는 값을 변형 없이, 단순 복사 형태도 있고, 여러 처리 과정을 거치거나, 원본 객체의 두 개 이상의 속성의 조합 등으로 새로운 값을 대상 객체의 속성 값에 설정 할 수도 있습니다.

Object Mapper의 필요성은 같은 수형에 대해서는 그리 필요가 없습니다. BinarySerializer를 통한 메모리 통째 복사를 수행하면 Deep Copy 까지 되므로 아주 좋은 방식입니다. RCL에서는 RCL.Core.BinarySerializer<T> 를 사용하면 됩니다.

```
5       [TestFixture]
6       public class BinarySerializerFixture
7       {
8           private static readonly ISerializer<UserInfo> UserSerializer = new BinarySerializer<UserInfo>();
9
10          [Test]
11          public void ObjectTest()
12          {
13              var user = UserInfo.GetSample();
14
15              var data = UserSerializer.Serialize(user);
16              var user2 = UserSerializer.Deserialize(data);
17
18              Assert.AreEqual(user.FirstName, user2.FirstName);
19              Assert.AreEqual(user.FavoriteMovies.Count, user.FavoriteMovies.Count);
20          }
21      }
```

RCL.Core.BinarySerializer<T> 사용 예

그럼 다른 수형이라면? 일반적으로 외부 데이터 서비스를 수행할 경우, 내부 시스템에서 사용하는 수형을 그대로 전달하지는 않습니다. Class가 최대한 정보를 숨겨야 하듯이, 한 시스템도 최대한 정보를 숨겨야, 시스템 안정성을 해치지 않습니다. 이에 한 시스템에서 외부로 정보를 제공할 때에는 꼭 필요한 정보만을 제공하기 위해 DTO (Data Transfer Object) 를 사용합니다. (이 것을 DTO 패턴이라고도 합니다.)

예를 들어 인사시스템의 User class 에는 일반적인 정보 외에 인사시스템 고유의 정보 (상벌관계, 인맥관계, 출신학교 등등 개인으로서는 민감한 사안) 는 외부에 유출하는 것 자체가 인사시스템의 신뢰성을 무너뜨리게 됩니다.
이렇게 외부 시스템에 정보를 제공하기 위해서는 UserDTO 라는 새로운 클래스를 만들고, 이 클래스의 인스턴스에 User 정보를 설정해서 서비스 해주면 됩니다.

이 때 현실적으로 개발자가 절망하는 대부분의 이유는 다음과 같습니다.

1. 전달해야 할 DTO 가 상당히 많을 때 (어쩔 수 없이 다 만들어야죠)
2. DTO가 내부 Object의 유연한 조합을 수행하여야 할 때 (Transform(변환) 수준이죠)
3. 통일된 DTO가 아닌 소비자 중심의 다양한 버전의 DTO를 원할 때
4. DTO 버전이 바뀔 때...

어디서 많이 본 듯한 문제점이죠? WebService나 WCF 도 결국 Data 통신이므로, 위와 같은 문제에 대한 고민을 가지고 있습니다. 다만, 일반 객체를 binary array로, Xml 로, JSON이냐 등 통신 상의 Stream의 형식에 문제와 관련된 것은 여기서는 제외하고 (향후 이 문제도 논의하겠습니다.)

여기서는 내부 정보를 외부에 전달하기 위한 방안으로서 DTO를 선택했을 경우, RCL에서 제공하는 ObjectMapper를 이용하면, 상당히 쉽게 DTO에 대한 작업을 수행할 수 있게 되겠습니다.

그럼 몇 가지 방법을 생각해 봅시다.

1. **시스템 내부의 객체를 DTO 객체에 값 매핑하기.**
2. **DB 의 정보를 직접 DTO에 설정하게 한다.**
   a. **ADO DataReader를 이용하여 AdoMapExtensions.Map<T> 을 통해**
   b. **Nhibernate의 Transfromer 또는 RCL의 INHRepository.ReportAll<T> 을 통해**

위 3가지 방식은 나름대로, 모두 활용가치가 있습니다.

1. 시스템 내부의 객체를 DTO 객체로 매핑하기

확장성 및 성능 때문에 캐시를 도입한 경우라면, DB나 파일보다는 캐시로부터 정보를 읽어 DTO로 매핑하는 것이 더욱 빠를 것입니다.

```
80          /// <summary>
81          /// 서로 다른 형식의 인스턴스의 속성 정보를 복사한다.
82          /// </summary>
83          [Test]
84          public void ObjectMapProperty()
85          {
86              using(new OperationTimer("Object MapProperty"))
87              {
88                  foreach(var source in SourceDatas)
89                  {
90                      var target = source.MapProperty(() => new TargetData());
91                      //var target = new TargetData();
92                      //source.MapProperty(target, true);
93
94                      Assert.AreEqual(source.Id, target.Id);
95                      Assert.AreEqual(source.Name, target.Name);
96                      Assert.AreEqual(source.Guid, target.Guid);
97                      Assert.AreEqual(source.CreatedDate, target.CreatedDate);
98                      Assert.AreEqual(source.ElapsedTimeSpan, target.ElapsedTimeSpan);
99                      Assert.AreEqual(source.NumberLong.GetValueOrDefault(),
100                                      (float) target.NumberLong.GetValueOrDefault());
101
102                     Assert.AreNotEqual(source.DataOnly, target.OtherDataOnly);
103                 }
104             }
105         }
```

형식이 다른 객체끼리 속성 정보를 복사하기 (속성 값 매핑)

위 예제는 RCL.Core.ObjectMapper 클래스의 확장 메소드를 이용한 것입니다. 두 객체 간 속성 명이 일치하면, 원본 객체의 속성값을 대상 객체의 속성 값에 설정해 줍니다. 이러한 객체의 속성 값 조작은 IDynamicAccessor 를 구현한 클래스에서 작업합니다.

기존에는 수형이 정확히 일치했어야 했는데, R**CL-3.6.9 부터는 수형의 정확히 일치하지 않아도, 변환이 가능한 수형이라면, 매핑이 가능하게 되었습니다.**

2. DB 정보를 직접 Class 인스턴스로 매핑하기

이 방식은 외부 서비스용 DTO에 국한된 것이 아니라 내부 persistent object도 적용되는 기술입니다.
지금까지 웹 개발자들은 DataSet, DataTable을 많이 사용해왔지만, 리얼웹 개발자들은 알게 모르게, RCL-1.1.0 부터 이 기술이 적용된 persistent object 들을 사용하고 있었습니다. 문제는 생 노가다였죠 (그래서 Code Generator를 이용하였구요)

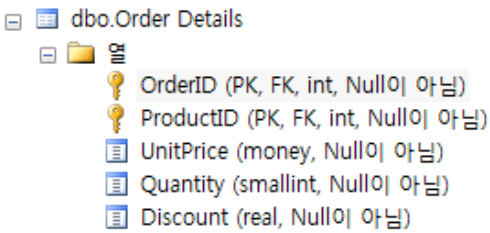
지금은 이런 방식 자체를 사용하지 않습니다. NET-3.0 부터 Auto-Implement Property (getter, setter만 있는 속성) 기능도 추가되었고 해서 persistent object 를 정의하는 작업이 그리 어렵지도, 노가다도 아닙니다. 그렇죠?

BPA 팀에서는 2008년부터 이러한 DB 정보를 class로 자동 매핑할 수 있도록 Upgrade 했습니다.
코딩량은 엄청 줄었습니다.

단점은 DB 컬럼 수형과 Persistent object 속성 수형이 정확히 일치해야 한다는 점 이였습니다.

장점은

1. Mapping 방식이 속성명이 굳이 일치하지 않아도, 속성명 Mapper를 통해 DB의 Naming 규칙과 Class의 Naming 규칙이 달라도 유연하게 대처가 가능한 점. (4가지 방식을 지원합니다)
2. ORM 처럼 매핑파일을 만들 필요가 없다.
3. Persistet Object에 값 설정을 자동으로 한다.
4. Reflection을 이용하지 않고, Dynamic Method를 이용하므로 속도가 빠르다.
5. **여기에 RCL-3.6.9 부터는 DB컬럼 수형과 Persistent Object 의 수형이 정확히 일치하지 않아도 됩니다.**




```
[Serializable]
internal class OrderDetail
{
    public virtual int OrderID { get; set; }
    public virtual int ProductID { get; set; }
    public virtual decimal? UnitPrice { get; set; }
    public virtual int? Quantity { get; set; }
    public virtual float? Discount { get; set; }
}
```

DB 테이블 컬럼과 Persistent class 의 속성

Fluent AD...

첨부된 Fluent ADO.NET 자료를 보면, DB 컬럼 명의 명명규칙과 Class의 명명 규칙이 상이해서 생기는 문제를 해결하는 방법 (Mapper, Persister, Converter 등)을 설명했습니다. 이에 따라, 일정한 규칙에 따른 두 시스템간의 정보의 매핑을 자동으로 수행 할 수 있도록 하였습니다.

```
[RowTest]
[Row(0, 10)]
[Row(3, 20)]
[Row(5, 10)]
public void ExecuteReaderWithPaging(int pageIndex, int pageSize)
{
    using(var cmd = SqlRepository.GetCommand(OrderDetailSql))
    {
        var orderDetails = SqlRepository.ExecuteReader<OrderDetail>(TrimMapper, cmd, pageIndex, pageSize);

        orderDetails.All(od => od.UnitPrice.GetValueOrDefault() > 0);
        orderDetails.All(od => od.Quantity.GetValueOrDefault() > 0);
        orderDetails.All(od => od.Discount.GetValueOrDefault() > 0.0);
    }
}
```

OrderDetail 정보를 DataReader로 읽고, 그 정보를 OrderDetail class의 인스턴스로 매핑하는 코드

위의 코드는 RealBPA는 기본으로 사용하는 코드입니다. DataReader를 사용하므로, DataSet, DataTable을 사용하는 것보다 빠르고, DB->DataSet->Persistent Object 로 매핑하려면, 쓸데없이 메모리만 낭비하는 꼴이 됩니다.

만약 Persistent Object를 사용하기로 결정했다면 AdoRepository.ExecuteReader<T>() 를 사용하는 것이 최상의 방법입니다. 물론 ORM을 사용한다면 다른 얘기가 되지만, Mapping 파일을 만들 필요가 없으므로, 공수가 줄어듭니다.

3. Nhibernate 기반의 RCL.Core.NH.Repository.ReportAll<T> 사용하기

NHibernate 는 ORM 툴로서 가장 유명한 Open Source Library이고, 리얼웹 제품의 서버 모듈에는 거의 모두 채택되어 사용되고 있습니다. 그 만큼 ORM으로서 충실한 역할을 수행하고 있습니다.
다만, 외부 시스템에 정보를 전달하기 위해서, 새로운 유형의 Persistent Object를 만들기 위해서 약간의 조작이 필요합니다.
이런 조작에는 NHibernate 가 기본으로 제공하는 Transformer 를 상속받아 이용하던가, 1번 방법처럼 NHibernate Persistent Object를 DTO로 만들면 됩니다.
또 다른 방법은 RCL.Data.NH.Repository.ReportAll<T>() 메소드를 사용하는 것입니다. 이 방법은, Icriteria나 Iquery에서 집계 함수등을 이용하여, 새로운 정보를 만들게 되면 그것을 Mapping 된 Entity가 아닌 일반 Data용 Class로 매핑해 준다는 것입니다.

```
390         [Test]
391         public void ReportAll()
392         {
393             var dtos = Repository<Parent>.ReportAll<ParentDTO>(ProjectByNameAndAge);
394
395             Assert.AreEqual(parentsInDB.Count, dtos.Count);
396             AssertDTOsCreatedFrom(parentsInDB, dtos);
397         }
```

Parent Name으로 Parent 정보를 찾아 ParentDTO로 매핑해서 반환

여기서 ParentDTO는 Mapping된 Entity가 아니므로, 외부 전달용 DTO로 활용할 수 있다.
ReportAll<T> 메소드의 핵심 코드도 결국은 IDynamicAccessor 를 이용한 속성 설정에 있고, 이를 간편하게 한 것이 ObjectMapper 입니다.

4. 결론

결국 3가지 대표적 방식의 핵심은 Reflection 또는 Dynamic Method를 통해 동적으로 객체의 속성값을 설정하는 것입니다. 이를 활용하여, Data Mapping / Reducing 에 범용적으로 활용할 수 있습니다.

여러분들도 시스템 연계 시, 웹서비스에서 DataSet 등을 사용하지 마시고, Class를 활용해 보시기 바랍니다.

델파이(Object Pascal)에서도 웹서비스 활용 시 Persistent Object의 수형을 파싱해서 Delphi용 class를 제공해줍니다. 이렇게 된다면, 굳이 내부를 알 수 없는 DataSet이나 DataTable을 사용하지 않고, Class를 사용하는 것이 견고하고, 명확한 서비스를 제공한다고 할 수 있습니다.