最近一直对DDD(Domain Driven Design)很感兴趣，于是去网上找了一些文章来看看，发现它确实是个好东西。于是我去买了两本关于领域驱动设计的书本和一本企业应用架构模式的书。看了之后也掌握了一些理论基础。但总感觉需要通过做一个实际项目来测试自己所学到的知识。因为以前我开发过一个叫做“蜘蛛侠论坛”的网站，官方演示地址：[http://www.entityspider.com/](http://www.entityspider.com/" \t "https://www.cnblogs.com/netfocus/archive/2010/12/20/_blank)（论坛目前已关闭，需要源代码的可以联系我），但在我学习了DDD之后，才明白原来之前我所做的设计是贫血模型+事务脚本的设计方法。这种设计方法有很多不足，最大的不足就是业务逻辑不能重用，业务逻辑没有组织为一个可重用的自封闭的业务模型。所以我想用DDD的思想来重新设计我的论坛。

大家都知道，一般我们在做DDD架构的应用时，一般会分成四层：

1. Presentation Layer：展现层，负责显示和接受输入；
2. Application Layer：应用层，很薄的一层，只包含工作流控制逻辑，不包含业务逻辑；
3. Domain Layer：领域层，包含整个应用的所有业务逻辑；
4. Infrastructure Layer：基础层，提供整个应用的基础服务；

 一开始接触这样的架构时，觉得确实很好。但后来在不断实践中遇到了不少问题，下面几个就是我所遇到的几个关键问题，在接下来的随笔中我将会一一和大家分享我是如何思考和解决这些问题的。

1. 是否应该使用ORM；
2. 持久化透明；
3. 高性能；
4. 事务支持；

**是否应该使用ORM？**

大家都知道ORM能将对象和数据库进行映射，它可以让开发人员完全按照面向对象的思维去设计实现软件，并且你无需关心对象如何从数据库取出来或保存到数据库。但是我发现几乎我所见过的所有ORM框架都基于同一个前提，那就是将对象的属性映射到数据库字段，将对象之间的引用映射到数据库表的关系。然后如果当你需要一些高级功能支持时，ORM会要求你对你的对象做一些设置，比如在NHibernate中，你如果要进行Lazy Load，你就必须将属性设置为virtual，如果是一对多，好像还有个叫ISet的东东吧，我不是太了解。还有，像LINQ to SQL，如果要支持Lazy Load，需要使用EntitySet或EntityRef，这两个类型的侵入性太强了，搞的你的模型不像模型。对于微软最新的ORM框架ADO.NET Entity Framework，我还没有真正用过，所以不怎么知道它的明显缺点，但据说ORM的映射能力目前还不如NHibernate。撇开ORM的高级特性不说，我们只谈基本特性我想我就能问倒你。是谁告诉你ORM的映射就是对象的属性和数据库表字段的映射？面向对象博大精深，继承，多态，重载，重写，等等。很多对象可能只有方法而没有属性，不要问我为什么。至少只要用属性可以实现的地方一般用方法也能做到，使用方法合适还是属性合适应该根据你当前的具体情况而定。但你跟我说，如果你要用ORM，那你就必须用属性，否则不能做到ORM的自动映射！那我岂不是很郁闷。另外，对于领域驱动设计来说，领域对象都是很丰富的，是充血的。对象不仅有很多的属性也会有很多的方法。可以说是“活”的东西，是能完成很多职责以及和别的对象进行交流的东西。但数据库中的数据是“死”的，是静态的。数据只有状态而没有行为。我很难想象为什么大家一定要将这种“活”的有生命的对象映射到一个“死”的数据库记录上去。在我看来，对象和数据是两个不同的概念，它们不应该被直接进行映射。

但是问题是，我们无法规避两个问题：1）将**对象的状态**保存到数据库；2）从数据库获取**对象的状态**并重新构建出一个对象；这是任何一个基于关系型数据库的面向对象应用程序无法回避的两个问题。不知道大家注意到了没有，我特地将对象的状态进行了加粗。目的就是想强调，真正应该需要映射的是：对象和状态和数据库之间的映射，而不是对象和数据库之间的映射。弄清楚了这个概念后，我就可以谈一下我对ORM映射的具体做法的观点。

在我看来对象由状态和行为组成，所谓的状态可以这样来形容：什么东西的什么是什么。比如人是一个对象，一个人的身高是什么，体重是什么；在比如一个订单，它的创建日期是什么，它有那些订单明细，他的订货人是谁，这些都是对象的状态。可以看出对象的状态在某个特定的时刻一定是静态的，没有任何动态的成分，可以用某种标准进行衡量。因此，我觉得我们应该将对象的状态和数据库中的表进行映射。那如何从对象的状态进一步创建出一个对象呢？这个应该很简单吧，虽然简单，但是目前来说要做这样的事情，只能人工去做，因为目前世界上还没有所谓的工具能将**对象的状态**映射到**对象**的。

如果用C#语言，该怎样表示对象的状态以及对象呢？这个问题对大家来说实在是在简单了，对吧！你肯定会说，对象的状态就是一些简单的类，这些类只有简单的属性，或者是其他一个简单的类，并且也不需要实现什么接口或继承什么基类。每个这种简单的类型和一个数据库表对应。而对象就是面向对象分析与设计中的一个概念。虽然它也是用类来表示，但它的实现并不是只有简单的属性。而是可能会包含很多丰富的元素，比如它的构造函数也可能很复杂，会接受很多参数，它可能会包含事件、方法、属性、内部类，也可能会实现多个接口，等等。总之按照我的话来说，它就是一个“活”的东西。

现在大家了解了什么是对象的状态，什么是对象了。那么该怎样根据对象的状态来创建一个对象呢？刚才说了，你自己能做到，工具做不到。这里涉及到两个问题：

1. 如何根据多个状态类创建出一个对象，或者根据一个状态类创建出多个对象；
2. 如何将一个对象的状态拆分为多个状态类，或者将多个对象的状态保存到一个状态类中；

理解了这两点后，我们就能很容易的按照这个思路来创建或持久化对象了。前面说了一大段一大段的废话，该是举个例子的时候了。

假设我现在有一个论坛，论坛有版块分组和版块两个概念。就像CSDN这个论坛一样，.NET就是一个版块分组，而ASP.NET或C#等就是版块。其中，一个版块分组可以包含多个版块，一个版块只能属于一个版块分组；也就是一对多的关系。

数据库表的设计：

1. 版块分组表：tb\_Groups，包含这些字段：Id, Subject, Enabled
2. 版块表：tb\_Sections，包含这些字段：Id, Subject, Enabled, GroupId

这些字段应该不用解释了吧，很容易理解。其中GroupId是一个外键，指向tb\_Groups表中的Id.

关于对象的设计，以版块为例，可能会设计成下面这样：

[IMG_256](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

  1     public class Section : AggregateRoot<Guid>  
  2     {  
  3         #region Private Variables  
  4   
  5         private Group group;  
  6         private int? totalThreadCount;  
  7         private List<User> adminUserList;  
  8   
  9         #endregion  
 10   
 11         #region Constructors  
 12   
 13         public Section(Guid id, Group group) : base(id)  
 14         {  
 15             this.group = group;  
 16         }  
 17   
 18         #endregion  
 19   
 20         #region Public Properties  
 21   
 22         [TrackingProperty]  
 23         public string Subject { get; set; }  
 24         [TrackingProperty]  
 25         public bool Enabled { get; set; }  
 26         public Group Group  
 27         {  
 28             get  
 29             {  
 30                 return group;  
 31             }  
 32             set  
 33             {  
 34                 if (group != value && value != null)  
 35                 {  
 36                     group = value;  
 37                     RaiseEvent(new SectionGroupChangedEvent { Id = Id, Group = group });  
 38                 }  
 39             }  
 40         }  
 41         public int TotalThreadCount  
 42         {  
 43             get  
 44             {  
 45                 if (totalThreadCount == null)  
 46                 {  
 47                     RaiseEvent(new SectionTotalThreadCountQueryEvent  
 48                     {  
 49                         Id = Id,  
 50                         SetTotalThreadCount = new Action<int>(count => totalThreadCount = count)  
 51                     });  
 52                 }  
 53                 return totalThreadCount.Value;  
 54             }  
 55         }  
 56         public ReadOnlyCollection<User> AdminUsers  
 57         {  
 58             get  
 59             {  
 60                 return AdminUserList.AsReadOnly();  
 61             }  
 62         }  
 63   
 64         #endregion  
 65   
 66         #region Public Methods  
 67   
 68         public void AddAdminUser(User user)  
 69         {  
 70             if (!AdminUserList.Contains(user))  
 71             {  
 72                 AdminUserList.Add(user);  
 73                 RaiseEvent(new SectionAdminUserAddedEvent { Id = Id, User = user });  
 74             }  
 75         }  
 76         public void RemoveAdminUser(User user)  
 77         {  
 78             if (AdminUserList.Contains(user))  
 79             {  
 80                 AdminUserList.Remove(user);  
 81                 RaiseEvent(new SectionAdminUserRemovedEvent { Id = Id, User = user });  
 82             }  
 83         }  
 84   
 85         #endregion  
 86   
 87         #region Private Properties  
 88   
 89         private List<User> AdminUserList  
 90         {  
 91             get  
 92             {  
 93                 if (adminUserList == null)  
 94                 {  
 95                     RaiseEvent(  
 96                         new SectionAdminUsersQueryEvent  
 97                         {  
 98                             Id = Id,  
 99                             SetUsers = new Action<IEnumerable<User>>(users => adminUserList = users.ToList())  
100                         });  
101                 }  
102                 return adminUserList;  
103             }  
104         }  
105   
106         #endregion  
107     }

[IMG_257](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

先不说这个对象设计的好坏，但至少在我看来，他至少已经不是一个简单的状态类型了。比如它的构造函数接收一个其他的对象（Group对象），它还有一些方法，它的某些属性的值被修改后会触发事件，它有良好的封装性，即不能对外界公开的就声明为私有，可以被外界访问但不能被修改的就设计成只读。还有因为一个版块必须要有一个所属的版块组，如果不提供版块组就不允许被创建。所以告诉我们必须在构造函数中传递给它一个Group对象。等等这些都说明它是一个既有状态也有行为的对象。

接下来我们再来看看对象状态类型是怎么样的？

[IMG_258](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

 1     public class GroupObject  
 2     {  
 3         public Guid Id { get; set; }  
 4         public string Subject { get; set; }  
 5         public bool Enabled { get; set; }  
 6     }  
 7     public class SectionObject  
 8     {  
 9         public Guid Id { get; set; }  
10         public string Subject { get; set; }  
11         public bool Enabled { get; set; }  
12         public Guid GroupId { get; set; }  
13     }  
14     public class SectionAndGroupObject  
15     {  
16         public SectionObject SectionObject { get; set; }  
17         public GroupObject GroupObject { get; set; }  
18     }

[IMG_259](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

上面的代码中，GroupObject代表Group对象的状态，SectionObject代表Section对象的状态，而SectionAndGroupObject则是一个组合状态对象。但是大家不要误会，它也只是一个状态类型。之所以我搞出一个组合状态类型，是为了可以避免重复写属性而已，没有其他特别目的。

好了，现在对象有了，表示对象的状态的类型有了，数据库表也有了。接下来谈谈该如何来有效的组织这三者，让对象的状态能进行保存和恢复。

前面大家看了我的观点后，可能会觉得，ORM我们不应该用，其实不是这样，虽然不能直接用它来将对象和数据库之间进行映射，但却可以用它来将对象状态和数据库进行映射。比如上面的例子中，GroupObject和SectionObject是两个状态对象，他们都是扁平的，仅仅保存了对象的全部或部分的状态，也就是说状态对象保存的都是状态，都是数据，是静态的。它的这个特性正好和数据库表完全一致。因此，我觉得我们可以将状态对象和数据库表通过任意一种成熟的ORM工具来建立映射。当然，如果你想人工去建立状态对象和数据库表之间的映射也没有问题，就是写起来比较麻烦，很多繁琐的工作都要你自己去完成。比如你用ADO.NET去写，那么你必须自己去创建Connection，Command，CommanParameter，等等繁琐累人的本不应该由你去做的任务，或者如果你连ADO.NET都不用，直接自己写SQL，那就更麻烦了。最后，如果你用ORM映射来实现，那么由于状态对象设计的时候往往已经兼顾了表的结构以及如何比较好的保存对象状态的两方面的问题，所以当我们在进行映射时，往往会非常简单直接，基本上状态对象的每个属性都是和数据库表一一对应的。以LINQ to SQL为例：

[IMG_260](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

 1     <Table Name="tb\_Groups">  
 2         <Type Name="CompanyName.ProductName.Modules.Forum.LinqToSqlDataProvider.GroupObject">  
 3             <Column Name="Id" Member="Id" DbType="UniqueIdentifier NOT NULL" IsPrimaryKey="true" />  
 4             <Column Name="Subject" Member="Subject" DbType="NVarChar(128) NOT NULL" CanBeNull="false" />  
 5             <Column Name="Enabled" Member="Enabled" DbType="Bit NOT NULL" />  
 6         </Type>  
 7     </Table>  
 8     <Table Name="tb\_Sections">  
 9         <Type Name="CompanyName.ProductName.Modules.Forum.LinqToSqlDataProvider.SectionObject">  
10             <Column Name="Id" Member="Id" DbType="UniqueIdentifier NOT NULL" IsPrimaryKey="true" />  
11             <Column Name="Subject" Member="Subject" DbType="NVarChar(128) NOT NULL" CanBeNull="false" />  
12             <Column Name="Enabled" Member="Enabled" DbType="Bit NOT NULL" />  
13             <Column Name="GroupId" Member="GroupId" DbType="UniqueIdentifier NOT NULL" />  
14         </Type>  
15     </Table>

[IMG_261](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

大家可以看到上面的映射非常清晰明了，可以说，只要是个ORM框架，就一定能完成这样简单的映射。

好了，现在一切就绪，我们来看看如何来实现对象的状态的保存（也就是对象持久化）以及如何从数据库中保存的状态重建一个对象。

对象持久化：

[IMG_262](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

 1         public void PersistANewSection(Section section)  
 2         {  
 3             //首先，将Section的部分状态保存到SectionObject中  
 4             SectionObject sectionObject = new SectionObject  
 5             {  
 6                 Id = section.Id,  
 7                 Subject = section.Subject,  
 8                 Enabled = section.Enabled,  
 9                 GroupId = section.Group.Id  
10             };  
11   
12             //然后，利用LINQ to SQL将状态对象（SectionObject）持久化到数据库  
13             dataContext.GetTable<SectionObject>().InsertOnSubmit(sectionObject);  
14             dataContext.SubmitChanges();  
15         }

[IMG_263](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

上面的这个函数可以将一个Section对象的部分状态持久化到数据库中。之所以说是部分，是因为Section对象还包含了一些其他的状态，不如版主信息等。大家可以看到，我们先Section对象的状态保存到一个SectionObject的状态对象中，然后在利用ORM框架（LINQ to SQL）来完成持久化。

从数据库中保存的状态重建一个对象：

[IMG_264](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

 1         public Section GetSectionFromDatabase(Guid sectionId)  
 2         {  
 3             //首先，利用LINQ to SQL这个牛逼的ORM工具来定义一个LINQ查询，它相当于是一个SQL语句，  
 4             //只不过我们用的不是表的字段，而是状态对象的属性而已。  
 5             var query = from s in dataContext.GetTable<SectionObject>()  
 6                         join g in dataContext.GetTable<GroupObject>()  
 7                         on s.GroupId equals g.Id  
 8                         where s.Id == sectionId  
 9                         select new SectionAndGroupObject { SectionObject = s, GroupObject = g };  
10   
11             //执行查询  
12             SectionAndGroupObject sectionAndGroupObject = query.FirstOrDefault();  
13   
14             //先根据GroupObject状态对象创建一个Group对象  
15             Group group = new Group(sectionAndGroupObject.GroupObject.Id)  
16             {  
17                 Subject = sectionAndGroupObject.GroupObject.Subject,  
18                 Enabled = sectionAndGroupObject.GroupObject.Enabled  
19             };  
20             //然后根据SectionObject状态对象和Group对象创建一个Section对象  
21             Section section = new Section(sectionAndGroupObject.SectionObject.Id, group)  
22             {  
23                 Subject = sectionAndGroupObject.SectionObject.Subject,  
24                 Enabled = sectionAndGroupObject.SectionObject.Enabled  
25             };  
26   
27             //最后返回Section对象  
28             return section;  
29         }

[IMG_265](https://www.cnblogs.com/netfocus/archive/2010/12/20/javascript:void(0);)

上面的代码里面都已经加了注释了，我想已经比较清楚了，所以就不做更多解释了。

好了，到这里，我想大家应该都已经比较清楚我想表达的意思了。**就是我们可以用ORM框架，但不是用它来直接将对象和数据库进行映射，而是我们需要先定义一些描述对象状态的状态对象，然后将状态对象和数据库进行映射。**

你可能还会问，我不想这么麻烦，我还是想直接将对象和数据库进行映射。这样我就可以不用再去重复定义状态对象了。没错，也许这样更好，但你必须做出一些牺牲，比如你的对象的状态必须用属性实现，或者如果要实现某些高级功能比如延迟加载，则可能还要按照某些特定ORM框架的要求将属性声明为virtual或EntitySet/EntityRef/ISet之类的。其实应该还有很多其他的限制，本人对ORM框架不太熟悉，也只能想到这些了。如果你都能接受这些限制出现在你的那些可爱的，活的，被报以厚望的，想要被重用N多次的对象中的话。那没有问题，你可以直接将对象和数据库进行映射。对了，我还想到一个不应该直接将对象映射到数据库而应该将状态对象映射到数据库的理由。就是对象之间往往很复杂，比如在领域驱动设计中，一个聚合对象（Aggregate Root）可能会聚合很多子对象（Aggregate Child），它们之间往往会有各种各样复杂的关联。因此当你在做ORM映射时，你会发现将对象映射到数据库表也不是一件那么容易的事情，一大堆的XML文件，搞的你眼睛看花。当然如果用工具自动生成映射那会简单很多，但我表示很感慨那个工具尽然能将如此复杂的具有各种内部关联的对象映射到数据库，并且是自动的，我想要想让工具可以自动完成那样的事情，不知道该告诉它多少信息啊！表示感慨！

相反，如果你设计一个扁平的静态的一维的状态对象类，设计的时候尽量多兼顾考虑表结构和对象状态拆分这两个因素，那我想设计出来的状态对象类一定可以很容易的和数据库进行映射，而且基本上99%的情况下都是类和表一对一映射。我现在在开发蜘蛛侠论坛的第三个版本，关于对象持久化和重建或新建的设计就是上面这些了。目前自我感觉良好，因为无论是多么复杂的对象，即使它没有一个属性而只有方法，我也能够很轻而易举的将它的状态保存到数据库或者从数据库取出来。

好了，以上就是我对ORM的一些个人观点。欢迎大家批评指正。本来还想把接下来的三个问题一起写完的，但感觉如果把所有的内容都写在一起，会太长，影响大家阅读，所以决定分成不同的随笔来写。我会尽快在接下来的随笔中和大家分享其他的几个问题以及我的解决方案。