

# PlinqO Nhibernate 新后端框架上手指南 V1.0

By ys

## 目录

前言.....	2
框架介绍.....	2
Demo 简介和使用说明 .....	2
简单的增删改查.....	8
新增一条数据.....	9
新增多条数据.....	10
更新一条数据.....	11
删除单条和多条数据.....	11
查找数据.....	12
分页数据的获取.....	12
一对多关系的处理.....	14
一对多关系实体的新增.....	14
一对多关系中关系的更新.....	15
一对多关系中关系的部分解除.....	17
多对多关系的处理.....	18
多对多关系实体新增.....	18
多对多关系中解除单个关系.....	21
解除全部关系.....	23
视图的使用.....	24
存储过程的使用.....	26
Nhibernate 常见异常处理 .....	30
Cascade 属性说明 .....	31
AutoMapper 的使用和异常处理 .....	35

## 前言

Plinq Nhibernate 框架已经在逐渐替代老的 Subsonic 三层架构，而且公司目前已经有三个项目正在采用该后端框架，为方便开发人员上手本套新框架并且规范使用，提高开发效率，特编写此指南。

## 框架介绍

本框架的全称是 Professional Linq to Object based Nhibernate，是 Codesmith tool 公司，对 Nhibernate 的封装和扩展，加入了大量的 Linq 特性，使 Nhibernate 更易上手，更易用，开发更便捷。Plinq Nhibernate 并不是在创造，而是在集成而已，所以它并不是全新的框架，也不是全新的技术。没有那个聪明的公司会重复的造轮子。

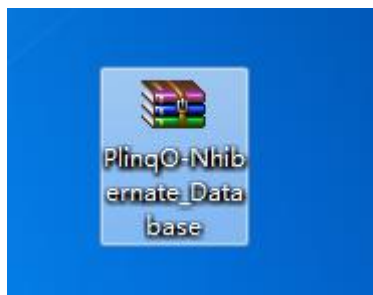
关于本上手指南附带的 demo，所有后端的代码，经过亲测，基本上没有问题的，学习过程中，如果有问题，欢迎找华通项目组成员，也欢迎私下找我

## Demo 简介和使用说明

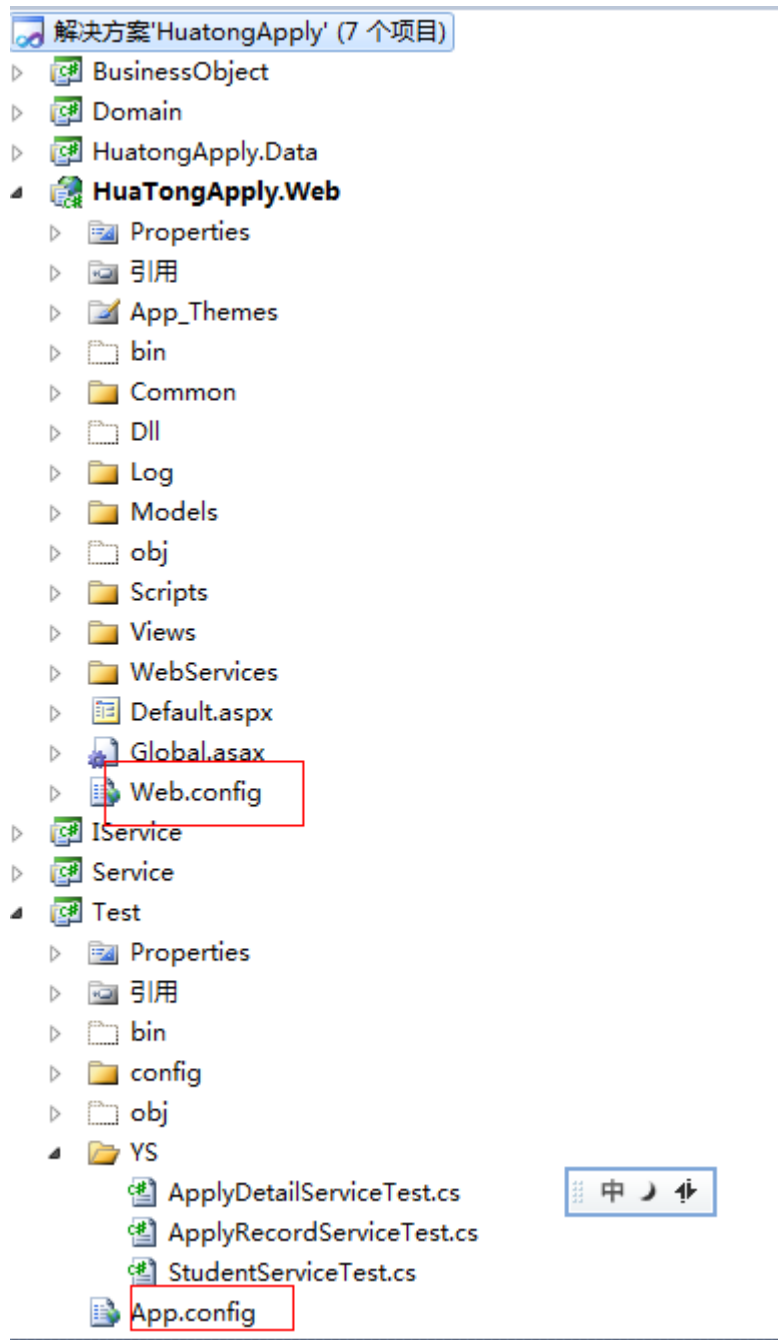
这个 demo 是从华通项目中提取的一个业务小模块。报销功能。为了便于大家直接能用数据库，我把数据库从 mysql 迁移到了 Sql Server，便于数据库数据和数据表以文件的形式迁移。数据库可以直接附加到本地使用

你只需要：

1. 附加 demo 中的数据库到 sql server。确保能打开 management studio，数据库正常使用即可。



2, 附加完数据库以后, 修改 web 层和 test 层的 config 文件, 修改连接字符串即可。

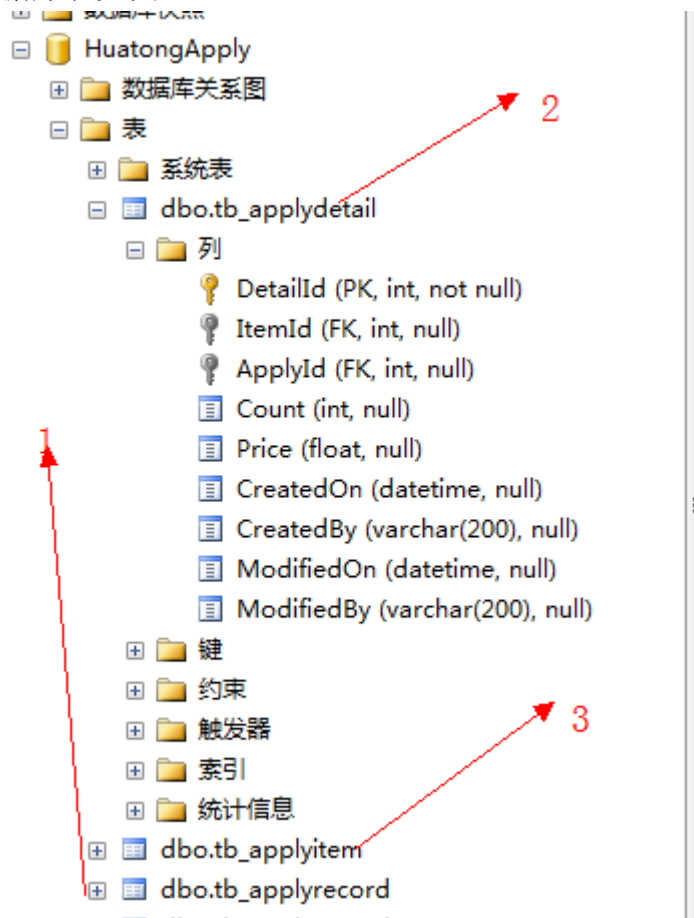


更改以上两个 config 文件中的连接字符串。

不知道怎么写 sql server 连接字符串的, 请到这里=>[ConnStrings](#)  
这里有 90%以上数据库的各种连接字符串写法。

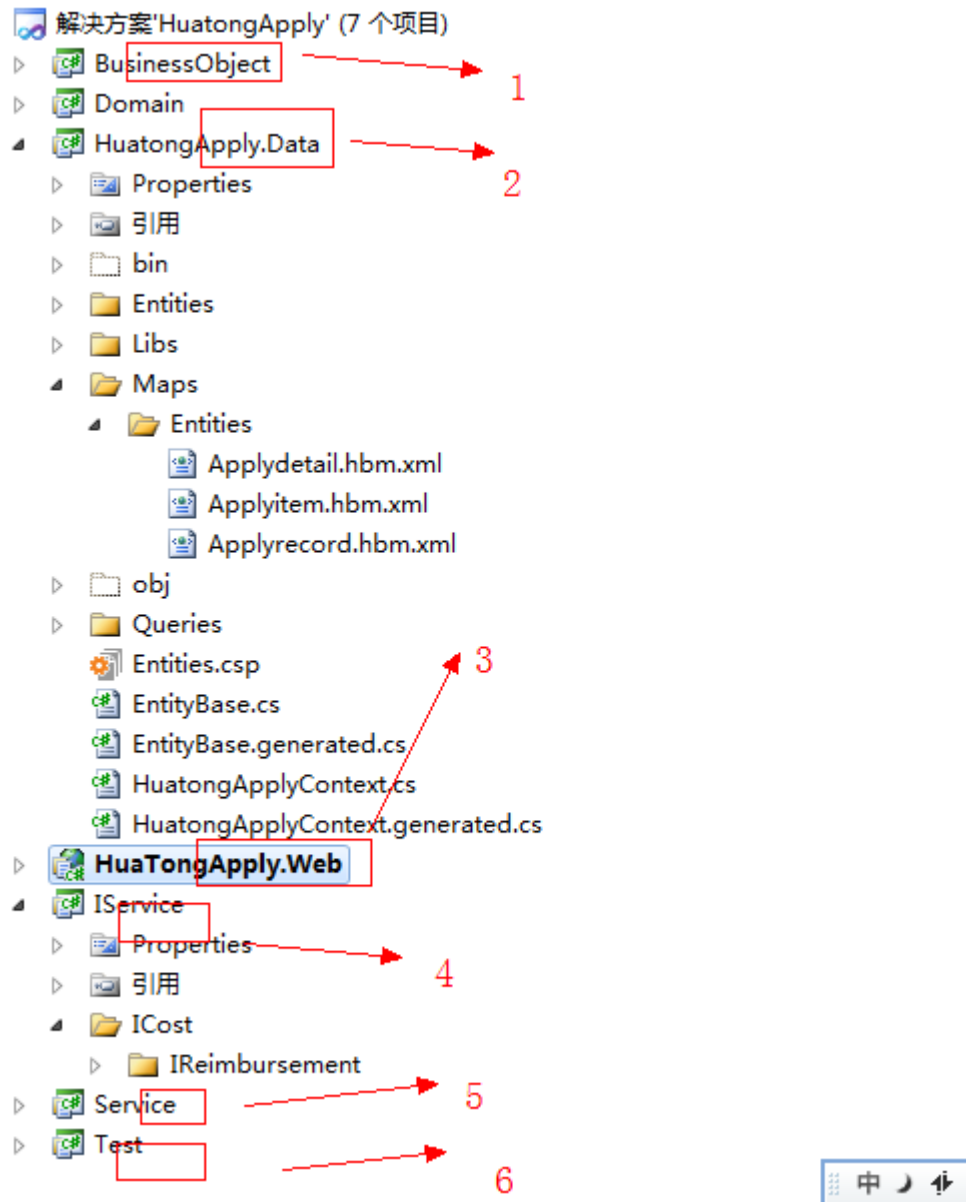
接下来更改编译, 直接运行就 ok

数据库表如图，



- 1, 报销记录表, 记录报销单
- 2, 报销详情表, 记录每个报销单下的报销详情记录。
- 3, 报销类目表, 记录所有的报销类目

Demo 各项目, 如图



1. bo 实体层
2. 数据访问层，本层代码由 Codesmith 生成，可扩展添加
3. Web 层
4. 服务接口层
5. 服务层
6. 单元测试层

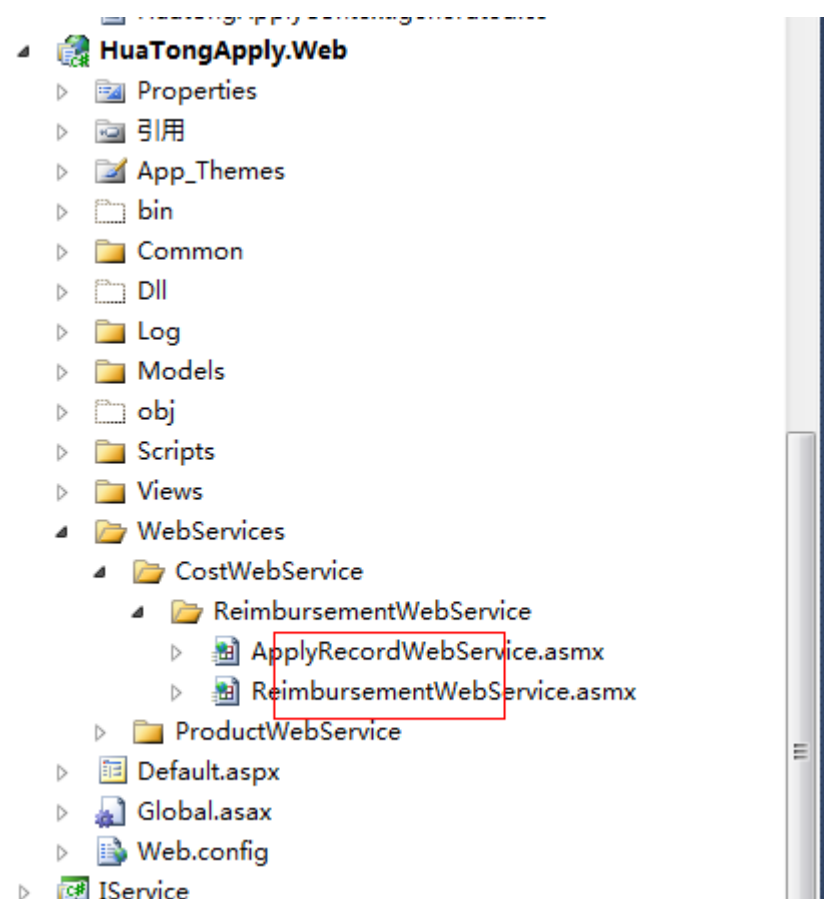
Web 层效果



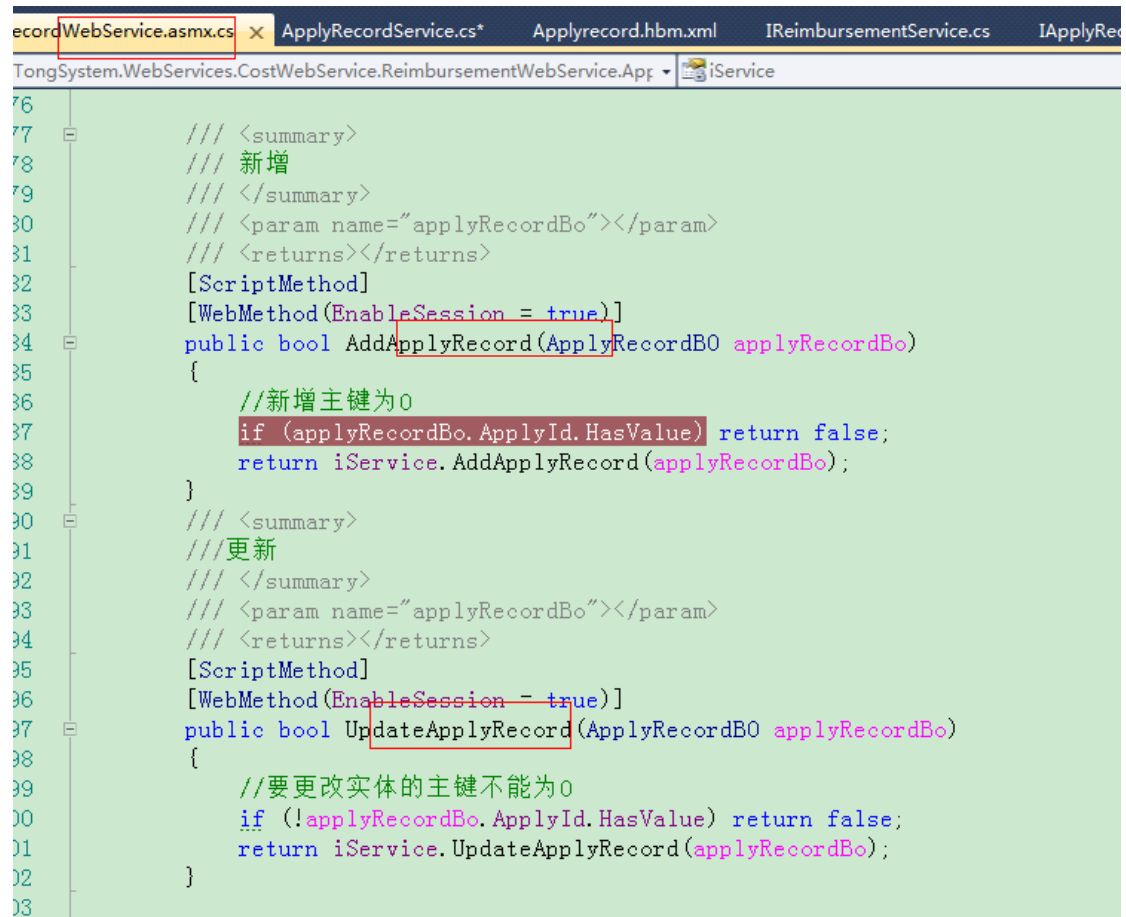
成本管理=>报销管理=>报销项目列表 || 报销记录列表

是本 Demo 示例代码的前端入口，可从这两个地方，新增数据，更新数据，查看数据，删除数据

后台的方法入口



Web 层的两个 Webservice 服务文件



```
76
77     /// <summary>
78     /// 新增
79     /// </summary>
80     /// <param name="applyRecordBo"></param>
81     /// <returns></returns>
82     [ScriptMethod]
83     [WebMethod(EnableSession = true)]
84     public bool AddApplyRecord(ApplyRecordBo applyRecordBo)
85     {
86         //新增主键为0
87         if (applyRecordBo.ApplyId.HasValue) return false;
88         return iService.AddApplyRecord(applyRecordBo);
89     }
90     /// <summary>
91     /// 更新
92     /// </summary>
93     /// <param name="applyRecordBo"></param>
94     /// <returns></returns>
95     [ScriptMethod]
96     [WebMethod(EnableSession = true)]
97     public bool UpdateApplyRecord(ApplyRecordBo applyRecordBo)
98     {
99         //要更改实体的主键不能为0
100        if (!applyRecordBo.ApplyId.HasValue) return false;
101        return iService.UpdateApplyRecord(applyRecordBo);
102    }
103
```

本 Demo 从华通系统中抠出，是没有权限控制功能的。所以这里不需要启用 Session，不纠结

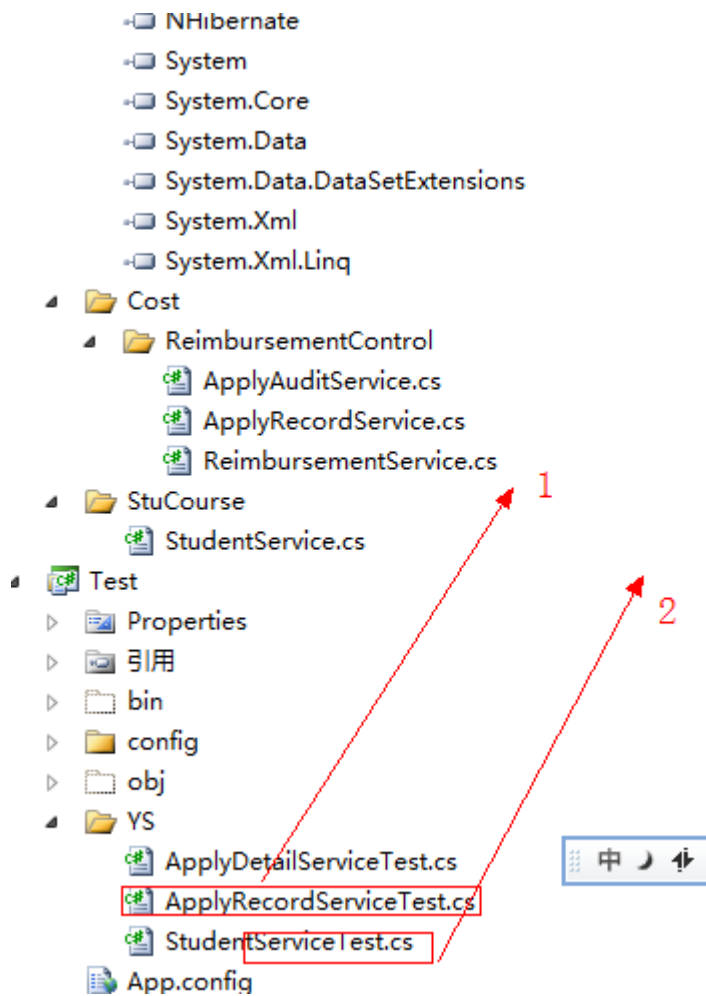
从 WebService 做入口，打断点调试，看代码如何一步步执行即可。

## 单元测试入口

1 文件，包含报销模块的方法测试，报销单新增，报销单详情的更新和删除，单条记录查找。分页数据查找，批量删除

2 文件，这个是我后来加上去的，经典的学生-课程，主要是测试，多对多关系业务处理，在本测试文件中，测试了，关系的部分解除，关系的全部解除，关系的插入操作。

便于大家对多对多关系实体，正确的的进行处理，在方法里已经标注了常见异常和错误的关系解除代码。也给出了正确的解除关系代码示例。  
方法编写形式不唯一，也不保证是最优。

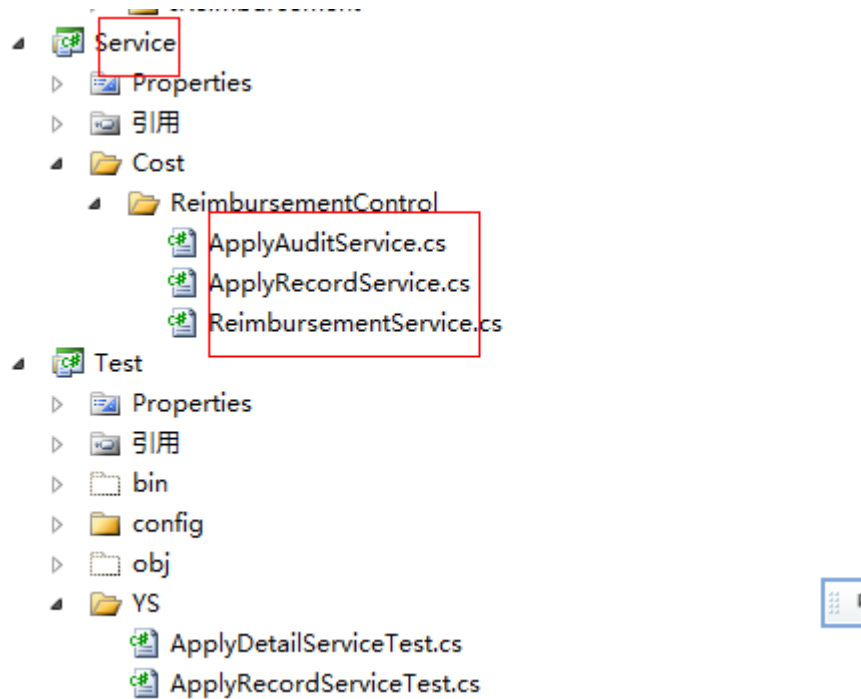


另外本框架需要结合 Nhibernate profile 工具使用，用法见 [Nhibernate Profile 的使用](#)。那里面有程序的破解版，运行即可。

## 简单的增删改查

本处代码示例所在位置如图





## 新增一条数据

```
{
    try
    {
        Mapper.Reset();
        Mapper.CreateMap<ApplyitemBO, Applyitem>();
        var entity = Mapper.Map<ApplyitemBO, Applyitem>(applyitemBo);
        entity.CreatedBy = "非系统访问";
        entity.CreatedOn = DateTime.Now;
        using (var context=new HuatongApplyContext())
        {
            context.BeginTransaction();
            context.Applyitem.InsertOnSubmit(entity);
            context.CommitTransaction();
            result = true;
        }
    }
    catch (Exception e)
    {
        result = false;
        LogHelper.WriteLog(string.Format("ReimbursementService.AddApplyItem({0})", applyitemBo), e);
    }
    return result;
}
```

还有一种写法如下

```

try
{
    Mapper.Reset();
    Mapper.CreateMap<ApplyitemBO, Applyitem>();
    var entity = Mapper.Map<ApplyitemBO, Applyitem>(applyitemBo);
    class HuatongApply.Data.Applyitem 统访问";
    entity.CreatedOn = DateTime.Now;
    using (var context=new HuatongApplyContext())
    {
        context.Applyitem.InsertOnSubmit(entity);
        context.SubmitChanges();
        result = true;
    }
}
catch (Exception e)
{
    result = false;
    LogHelper.WriteLog(string.Format("ReimbursementService.AddApplyItem({0})", applyitemBo), e);
}
return result;

```

上面的代码也可以不用事务，因为是无关系的简单数据写入操作

## 新增多条数据

```

try
{
    Mapper.Reset();
    Mapper.CreateMap<ApplyitemBO, Applyitem>();
    var entity = Mapper.Map<ApplyitemBO, Applyitem>(applyitemBo);
    entity.CreatedBy = "非系统访问";
    entity.CreatedOn = DateTime.Now;
    var entity2=entity;
    List<Applyitem>entities=new List<Applyitem>() {entity, entity2};
    using (var context=new HuatongApplyContext())
    {
        context.BeginTransaction();
        context.Applyitem.InsertAllOnSubmit(entities);
        context.CommitTransaction();
        result = true;
    }
}
catch (Exception e)
{
    result = false;
    LogHelper.WriteLog(string.Format("ReimbursementService.AddApplyItem({0})", applyitemBo), e);
}
return result;

```

将要插入的数据纳入一个集合里，传入，InsertAllOnSubmmit 即可注意，多记录插入，需要开启显示事务。

因为是单表无关系数据写入，所以这里的多条数据插入基本上是不会有异常的。所以，using 内部一般也无需 try -catch 捕获异常并回滚的操作

## 更新一条数据

```
try
{
    using (var context = new HuatongApplyContext())
    {
        context.BeginTransaction();
        var entity=context.Applyitem.FirstOrDefault(p => p.ItemId == applyitemBo.ItemId);
        if(entity!=null)
        {
            entity.ItemName = applyitemBo.ItemName;
            entity.ModifiedBy = applyitemBo.ModifiedBy;
            entity.ModifiedOn = DateTime.Now;
            result = true;
        }
        else
        {
            result = false;
        }
    }
    context.CommitTransaction();
}
```

更新操作，要做的是先把你需要更新的对象查出来，再把更新信息赋值，再Submmmit。这里因为是两次数据库操作，所以开启了显示事务，会比创建两个隐式事务好点。

## 删除单条和多条数据

```
/// <returns></returns>
public bool DeleteApplyitemByIds(Int32[] ids)
{
    bool result = false;
    var entities = ids.Select(p => new Applyitem() {ItemId = p});
    try
    {
        using (var context=new HuatongApplyContext())
        {
            context.BeginTransaction();
            context.Applyitem.DeleteAllOnSubmit(entities);
            context.CommitTransaction();
            result = true;
        }
    }
    catch (Exception e)
    {
        result = false;
        LogHelper.WriteLog(string.Format("ReimbursementService.DeleteApplyitemByIds({0})", ids));
    }
    return result;
}
```

删除数据的时候不需要去挨个查，直接构造要删除的对象，传入deleteAllOnSubmmmit即可。因为删除只需要知道记录的主键id即可。这个方法可以删除一条记录，也可以删除多条，跟插入多条数据差不多

## 查找数据

```
{
    ApplyRecordBO bo = null;
    Applyrecord entity = null;
    try
    {
        using (var context = new HuatongApplyContext())
        {
            context.BeginTransaction();
            entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == id);
            var temp1 = context.Applyrecord.ByApplyId(id).ToList();
            var temp2 = context.Applyrecord.ByAuditStatus(0).ToList();
            var temp3 = context.Applyrecord.ByCreatedOn(DateTime.Now).ToList();
            context.CommitTransaction();
        }
        Mapper.Reset();
        Mapper.CreateMap<Applyrecord, ApplyRecordBO>();
        bo = Mapper.Map<Applyrecord, ApplyRecordBO>(entity);
    }
    catch (Exception e)
    {
        LogHelper.WriteLog(string.Format("ApplyRecordService.GetApplyRecordById({0})", id), e);
    }
}
```

本后端框架，功能最齐全的就是查找。针对表的每一个字段都生成了对应的扩展方法，很方便、

```
try
{
    using (var context = new HuatongApplyContext())
    {
        context.BeginTransaction();
        entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == id);
        var temp1 = context.Applyrecord.ByApplyId(id);
        var temp2 = context.Applyrecord.ByAuditStatus(0);
        var temp3 = context.Applyrecord.ByCreatedOn(DateTime.Now);
        var temp4 = context.Applyrecord.ByCreatedOn(DateTime.Now);
        context.CommitTransaction();
    }
    Mapper.Reset();
    Mapper.CreateMap<Applyrecord, ApplyRecordBO>();
    bo = Mapper.Map<Applyrecord, ApplyRecordBO>(entity);
}
catch (Exception e)
{
    LogHelper.WriteLog(string.Format("ApplyRecordService.GetApplyRecordById({0})", id), e);
}
return bo;
```

## 分页数据的获取

使用已经封装好的 Linq 分页的扩展方法

```

public static Page<TResult> GetPagedListQuery<TEntity, IEntity, TOrderBy, TResult>(
    this IQueryable<TEntity> query,
    int index,
    int pageSize,
    List<Expression<Func<TEntity, IEntity, bool>>> where,
    Expression<Func<TEntity, TOrderBy>> orderBy,
    Func<IQueryable<TEntity>, List<TResult>> selector,
    bool isAsc)
{

```

使用示例如下

1 处是接收查询的条件的 bo

2 处条件表达式集合

3 填充查询条件到条件表达式集合

```

public Page<ApplyRecordBO> GetApplyRecords(int pageIndex, int pageSize, string order, string sort, ApplyRecordBO applyRecordBo)
{
    Page<ApplyRecordBO> applyRecordsPage = null;
    Page<ApplyRecord> page = null;
    try
    {
        var wheres = new List<Expression<Func<Applyrecord, bool>>>();
        if (!string.IsNullOrEmpty(applyRecordBo.ApplyBy))
        {
            Expression<Func<Applyrecord, bool>> where = p => p.ApplyBy.Contains(applyRecordBo.ApplyBy);
            wheres.Add(where);
        }
        if (applyRecordBo.OrId!=null&&applyRecordBo.OrId!=0&&applyRecordBo.OrId!=-1)
        {
            Expression<Func<Applyrecord, bool>> where = p => p.OrId==applyRecordBo.OrId;
            wheres.Add(where);
        }
        if (!string.IsNullOrEmpty(applyRecordBo.ApplyBy))
        {
            Expression<Func<Applyrecord, bool>> where = p => p.ApplyBy .Contains(applyRecordBo.ApplyBy);
            wheres.Add(where);
        }
        if (!string.IsNullOrEmpty(applyRecordBo.OrName))
        {
            Expression<Func<Applyrecord, bool>> where = p => p.OrName.Contains(applyRecordBo.OrName);
            wheres.Add(where);
        }
        //排序,默认创建时间来排序
        Expression<Func<Applyrecord, Object>> orderBy = null;
        switch (sort)
        {

```

指定排序字段

```

//排序,默认创建时间来排序
Expression<Func<Applyrecord, Object>> orderBy = null;
switch (sort)
{
    case "ApplyId":
        orderBy = o => o.ApplyId;
        break;
    default:
        orderBy = o => o.CreatedOn;
        break;
}
//分页筛选器
Func<IQueryable<Applyrecord>, List<Applyrecord>> selector =
    u => u.Skip((pageIndex - 1) * pageSize).Take(pageSize).ToList();
using (var context = new HuatongApplyContext())
{
    context.BeginTransaction();
    page = context.Applyrecord.GetPagedListQuery(pageIndex, pageSize, wheres, orderBy, selector, false);
    context.CommitTransaction();
}
Mapper.CreateMap<Page<Applyrecord>, Page<ApplyRecordBO>>();
Mapper.CreateMap<Applyrecord, ApplyRecordBO>();
applyRecordsPage = Mapper.Map<Page<Applyrecord>, Page<ApplyRecordBO>>(page);
}

```

## 一对多关系的处理

一对多关系实体，关系的新增，更改，删除和全部删除。

本 demo 示例中，报销记录和报销详情是一对多的关系，一个报销单下有多  
个报销详情。

```
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2" namespace="HuatongApply.Data" assembly="HuatongApply.Data">
  <class name="ApplyRecord" table="[dbo].[tb_applyrecord]" >
    <id name="ApplyId" column="[ApplyId]" type="Int32" >
      <generator class="native" />
    </id>
    <property name="OrId" column="[Or_Id]" type="Int32" not-null="false" />
    <property name="Price" column="[Price]" type="Double" not-null="false" />
    <property name="Currency" column="[Currency]" type="AnsiString" not-null="false" length="10" />
    <property name="ApplyBy" column="[ApplyBy]" type="AnsiString" not-null="false" length="200" />
    <property name="ApplyOn" column="[ApplyOn]" type="DateTime" not-null="false" />
    <property name="Description" column="[Description]" type="AnsiString" not-null="false" length="500" />
    <property name="IsCar" column="[IsCar]" type="Int32" not-null="false" />
    <property name="CarNo" column="[CarNo]" type="AnsiString" not-null="false" length="20" />
    <property name="AuditStatus" column="[AuditStatus]" type="Int32" not-null="false" />
    <property name="DepAuditBy" column="[DepAuditBy]" type="AnsiString" not-null="false" length="200" />
    <property name="FinanceAuditBy" column="[FinanceAuditBy]" type="AnsiString" not-null="false" length="200" />
    <property name="FinalAuditBy" column="[FinalAuditBy]" type="AnsiString" not-null="false" length="200" />
    <property name="CreatedOn" column="[CreatedOn]" type="DateTime" not-null="false" />
    <property name="CreatedBy" column="[CreatedBy]" type="AnsiString" not-null="false" length="200" />
    <property name="ModifiedOn" column="[ModifiedOn]" type="DateTime" not-null="false" />
    <property name="ModifiedBy" column="[ModifiedBy]" type="AnsiString" not-null="false" length="200" />
    <property name="OrName" column="[OrName]" type="AnsiString" not-null="false" length="200" />
    <bag name="ApplydetailList" inverse="true" cascade="all-delete-orphan" lazy="true" >
      <key>
        <column name="[ApplyId]" />
      </key>
      <one-to-many class="Applydetail" />
    </bag>
  </class>
</hibernate-mapping>
```

那么 ApplyRecord 就是一对多关系的“一”方实体

## 一对多关系实体的新增

```
1
List<Applydetail> details=new List<Applydetail>();
double recordTotalPrice = default(float); //报销单总价
try
{
    Mapper.Reset();
    var map=Mapper.CreateMap<ApplyRecordBO, Applyrecord>();
    Mapper.CreateMap<ApplydetailBO, Applydetail>();
    var entity = Mapper.Map<ApplyRecordBO, Applyrecord>(applyRecordBo);
    entity.CreatedBy = "杨双";
    entity.CreatedOn = DateTime.Now;
    entity.AuditStatus = 0; //新增报销项的状态为‘未审核’
    #region
```

准备报销单数据

```

entity.AuditStatus = 0; //新增报销项的状态为‘未审核’
#region
using (var context = new HuatongApplyContext())
{
    try
    {
        context.BeginTransaction();
        context.Applyrecord.InsertOnSubmit(entity);
        context.SubmitChanges(); //先Submit产生报销单
        foreach (var applydetailBo in applyRecordBo.ApplydetailBoList)
        {
            Applydetail singleDetail = new Applydetail();
            singleDetail.Applyitem =
                context.Applyitem.FirstOrDefault(p => p.ItemId == applydetailBo.ItemId);
            singleDetail.Applyrecord = entity;
            singleDetail.CreatedOn = entity.CreatedOn;
            singleDetail.CreatedBy = entity.CreatedBy;
            singleDetail.Count = applydetailBo.Count;
            singleDetail.Price = applydetailBo.Price;
            details.Add(singleDetail);
            recordTotalPrice = recordTotalPrice +
                (singleDetail.Count.GetValueOrDefault()) * (singleDetail.Price);
        }
        entity.ApplydetailList = details;
        entity.Price = recordTotalPrice; //提交事务产生报销详情记录
        context.CommitTransaction();
        result = true;
    }
    catch (Exception e)
    {
        context.RollbackTransaction();
    }
}

```

Diagram annotations in the image:

- 1: Points to `context.BeginTransaction();`
- 2: Points to the block of code that creates `singleDetail` and adds it to `details`.
- 3: Points to `entity.ApplydetailList = details;`
- 4: Points to `context.CommitTransaction();`
- 5: Points to `context.RollbackTransaction();`

准备报销详情数据(一对多关系多方的数据实体)

如果在准备子数据的过程中,不需要访问数据库,那么请将此过程放到 using 块外。

如果在准备子数据的时候需要访问数据库,请将此过程纳入 using 块,共享同一个 context, 并且将此过程置于显式声明的事务下

1 处提交父数据到数据库, 让报销单产生, 并让 NH 开始跟踪它

2, 准备报销详情数据,

3 将报销详情实体集合关联到报销单的报销详情属性上, 赋值给它

4 处, 提交事务, 写入父子数据

注意, using 内层的异常捕获和回滚是必须有的, 操作多表需要异常捕获, 事务开启->提交/回滚的处理过程是需要的。

## 一对多关系中关系的更新

代码有点长, 分两个图说明

以下代码是, 更新指定报销单下的报销详情数据

```

try
{
    context.BeginTransaction();
    entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == applyRecordBo.ApplyId);
    if(entity!=null&&entity.ApplydetailList.Count>=0)
    {
        entity.ApplydetailList.Clear();
    }
    context.SubmitChanges();
    foreach (var applydetailBo in applyRecordBo.ApplydetailBoList)
    {
        Applydetail singleDetail = new Applydetail();
        singleDetail.Applyitem =
            context.Applyitem.FirstOrDefault(p => p.ItemId == applydetailBo.ItemId.GetValueOrDefault());
        singleDetail.Applyrecord = entity;
        singleDetail.CreatedOn = DateTime.Now;
        singleDetail.CreatedBy = "杨双";
        singleDetail.Count = applydetailBo.Count;
        singleDetail.Price = applydetailBo.Price;
        details.Add(singleDetail);
        recordTotalPrice = recordTotalPrice +
            (singleDetail.Count.GetValueOrDefault()) * (singleDetail.Price.GetValueOrDefault());
    }
}

```

做法是，先将关联的报销详情清空，并 submit，让数据库执行删除操作，通常这一步，数据库处理有两种，删除孤儿节点即解除关系删除报销详情数据，或者设置报销详情数据关联的报销单为 null，此处理只解除了关系，但是没有删除数据。具体视 xml 中 cascade 属性设置而定。

```

{
    context.BeginTransaction();
    entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == applyRecordBo.ApplyId);
    if(entity!=null&&entity.ApplydetailList.Count>=0)
    {
        entity.ApplydetailList.Clear();
    }
    context.SubmitChanges();
    foreach (var applydetailBo in applyRecordBo.ApplydetailBoList)
    {
        Applydetail singleDetail = new Applydetail();
        singleDetail.Applyitem =
            context.Applyitem.FirstOrDefault(p => p.ItemId == applydetailBo.ItemId.GetValueOrDefault());
        singleDetail.Applyrecord = entity;
        singleDetail.CreatedOn = DateTime.Now;
        singleDetail.CreatedBy = "杨双";
        singleDetail.Count = applydetailBo.Count;
        singleDetail.Price = applydetailBo.Price;
        details.Add(singleDetail);
        recordTotalPrice = recordTotalPrice +
            (singleDetail.Count.GetValueOrDefault()) * (singleDetail.Price.GetValueOrDefault());
    }
    context.Applydetail.InsertAllOnSubmit(details);
    context.SubmitChanges(); //Submit清除所有报销子项
    //创建报销子项
    entity.Price = recordTotalPrice; //提交事务产生报销详情记录
    entity.ModifiedBy = "杨双";
    entity.ModifiedOn = DateTime.Now;
    context.CommitTransaction();
    result = true;
}

```

这图接着上图。

和插入一样，1 处重新准备报销详情数据

2 将报销单和刚创建的报销详情实体关联起来，建立关系。

3 写入所有报销详情数据，（此时报销单和报销详情的关系尚未同步到数据库）

4 更新报销单到报销详情的新的关联关系

注意，这个也是操作多表，事务，事务异常回滚必不可少。



上面是更新的一对多关系型的全部多方实体，其实已经包括了全部关系实体的移除操作。

## 一对多关系中关系的部分解除

如果是不确定长度的 id 集合，请如下写

```
List<Applydetail> details = new List<Applydetail>();
double recordTotalPrice = default(float); //报销单总价
using (var context = new HuatongApplyContext())
{
    try
    {
        int[] deleteIds=new int[] {1,2,3,4};
        context.BeginTransaction();
        entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == applyRecordBo.ApplyId);
        if(entity!=null&&entity.ApplydetailList.Count>0)
        {
            var realtedDetails = entity.ApplydetailList.ToList(); //取所有关系实体
            //取要删除的关系实体
            foreach (var deleteId in deleteIds)
            {
                var deletedEntity=realtedDetails.FirstOrDefault(p => p.DetailId == deleteId);
                if(deletedEntity!=null)
                {
                    entity.ApplydetailList.Remove(deletedEntity);
                }
            }
        }
    }
}
```



确定长度并且数量少的 id 集合，如下写

```
int[] deleteIds=new int[] {1,2,3,4};
context.BeginTransaction();
entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == applyRecordBo.ApplyId);
if(entity!=null&&entity.ApplydetailList.Count>0)
{
    var realtedDetails = entity.ApplydetailList.ToList(); //取所有关系实体
    //取要删除的关系实体
    realtedDetails.RemoveAll(p => p.DetailId == 1 || p.DetailId == 2 || p.DetailId == 3 || p.DetailId == 4);
}
```

当然也还有其它的写法，本处只做实例说明。

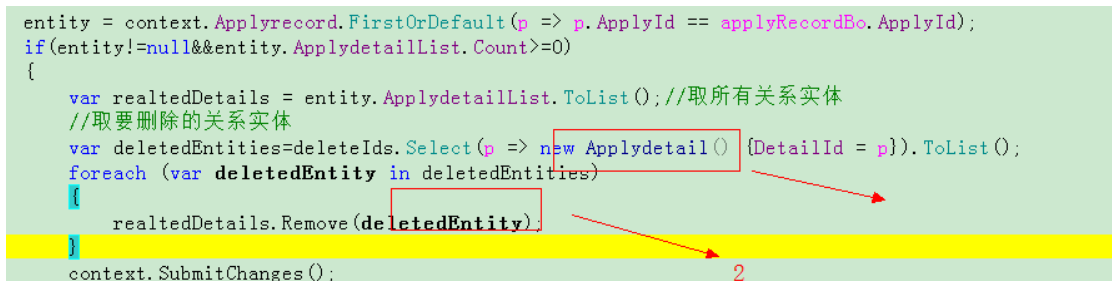
这段代码的目的是，将报销单 a 下的主键为 1234 的报销详情记录删除

1 要删除报销子项的 id 集合

2 处的操作，需要把要删除的实体，从 Nhibernate 跟踪到的关联实体集合中查出来，再删除它，否则是删不掉的。

这么写会报错

```
entity = context.Applyrecord.FirstOrDefault(p => p.ApplyId == applyRecordBo.ApplyId);
if(entity!=null&&entity.ApplydetailList.Count>0)
{
    var realtedDetails = entity.ApplydetailList.ToList(); //取所有关系实体
    //取要删除的关系实体
    var deletedEntities=deleteIds.Select(p => new Applydetail() {DetailId = p}).ToList();
    foreach (var deletedEntity in deletedEntities)
    {
        realtedDetails.Remove(deletedEntity);
    }
}
context.SubmitChanges();
```



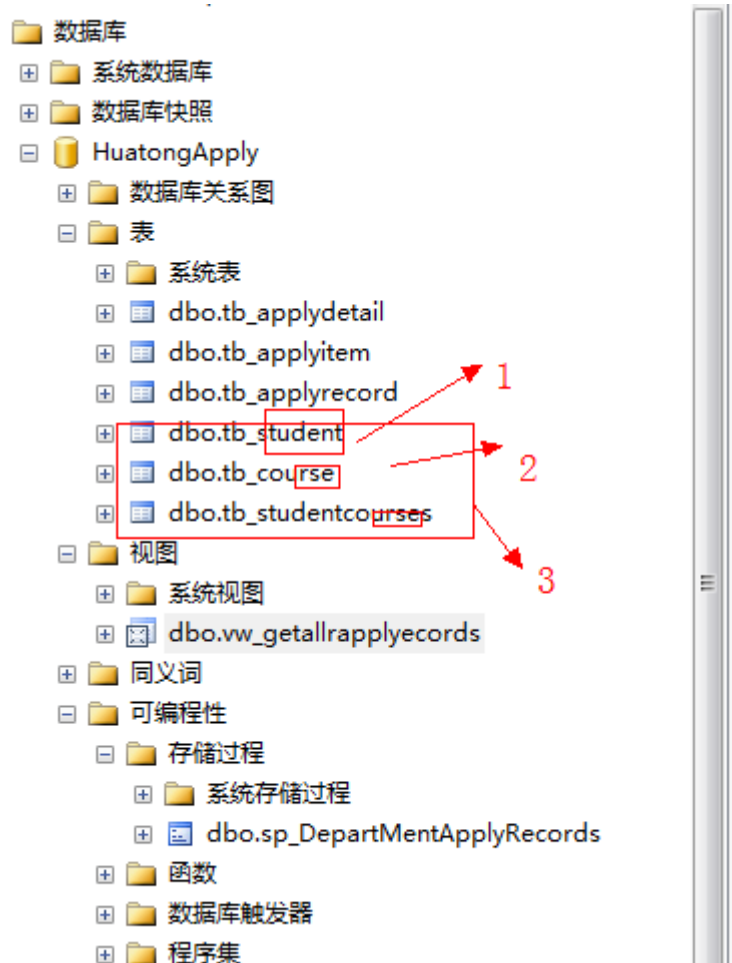
以上代码删不掉关系实体。NHibernate 只能管理它跟踪起来的实体。直接

new 出来的对象不在它的管理范围内，不会产生删除的 sql，无法删除。

## 多对多关系的处理

多对多关系的处理相对来说难度要大一些，为了方便大家学习掌握，少走弯路，我采用经典的学生-选课，这一业务场景来说明对多关系中如何处理关系实体。这一场景也适用于角色-权限多对多场景的处理。

数据库表



## 多对多关系实体新增

多对多关系插入数据的时候，最常见的代码如下

```

bool result = false;
//准备学生数据
Student entity=new Student();
Mapper.Reset();
Mapper.CreateMap<StudentBo, Student>();
entity=Mapper.Map<StudentBo, Student>(studentBo);
//准备课程数据
List<Course>courses=new List<Course>();
if (studentBo.CourseListBos != null)
{
    foreach (var courseListBo in studentBo.CourseListBos)
    {
        //注意，加入的课程必须是已经存在的课程。否则插入数据的时候
        //课程外键依赖检查，会抛异常
        courses.Add(new Course() {Identification = courseListBo.Identification});
    }
}

using (var context=new HuatongApplyContext())
{
    try
    {
        context.BeginTransaction();
        context.Student.InsertOnSubmit(entity);
        context.SubmitChanges(); //写入学生数据
        entity.CourseList = courses; //写入关系
        context.CommitTransaction();
        result = true;
    }
    catch (Exception e)
    {
    }
}

```

Nhibernate 在加载对象的关系集合的时候，NH 会跟踪这个集合，如果多对多关系型实体，需要在写入数据的时候，同时写入关系。

那么，请向 NH 跟踪的关系实体集合中添加新的关系对象，再 Submit

而不是，用一个新的集合赋值给实体的集合属性，就像 2 处，这是不怎么正确的写法，因为 2 处 courses 集合，nh 并没有对它进行跟踪管理。

但是实际上，我在用 NH 的过程中，有的 xml 配置，确实看到了代码这么写，关系数据又可以正常写入的情况。有点费解。

按照以下这么写，相对合理

```

bool result = false;
//准备学生数据
Student entity=new Student();
Mapper.Reset();
Mapper.CreateMap<StudentBo, Student>();
entity=Mapper.Map<StudentBo, Student>(studentBo);
//准备课程数据
List<Course>courses=new List<Course>();
if (studentBo.CourseListBos != null)
{
    courses=studentBo.CourseListBos.Select(p => new Course() {Identification= p.Identification}).ToList();
}
using (var context=new HuatongApplyContext())
{
    try
    {
        context.BeginTransaction();
        entity.CourseList = courses;//关联起关系
        context.Student.InsertOnSubmit(entity);//写入学生
        context.CommitTransaction();
        result = true;
    }
    catch (Exception e)
    {
        context.RollbackTransaction();
        result = false;
        //异常日志写入
    }
}

```

关系关联再前，数据写入操作在后。或者如下写也是可以的。

```

Mapper.Reset();
Mapper.CreateMap<StudentBo, Student>();
entity=Mapper.Map<StudentBo, Student>(studentBo);
entity.CourseList=new List<Course>();
//准备课程数据
var courseids=new List<Int32>();
if (studentBo.CourseListBos != null)
{
    courseids=studentBo.CourseListBos.Select(p => p.Identification).ToList();
}
using (var context=new HuatongApplyContext())
{
    try
    {
        context.BeginTransaction();
        foreach (var courseid in courseids)
        {
            //向NH跟踪的集合中添加关系数据
            //否则关系数据无法写入
            entity.CourseList.Add(new Course() { Identification = courseid });
        }
        context.Student.InsertOnSubmit(entity);
        context.CommitTransaction();
        result = true;
    }
    catch (Exception e)
    {
        context.RollbackTransaction();
        result = false;
        //异常日志写入
    }
}
}

```

写法有多种，但是所有的操作，都应该遵循一条法则，关联操作在前，写入再后。就 ok

另外，要插入的关系数据，请手动进行赋值构造，而非使用 AutoMapper 直接从 bo=》entity 映射。

# 多对多关系中解除单个关系

id 为 8 的学生不再想选修 id 为 1 的课程，则需要从关系表-选课表中删除 studentId=8 并且 courseId=1 的记录。

▶	1	2
	2	4
	2	5
	3	5
	3	6
	4	2
	8	1
	8	2
	8	3
	8	4
	8	5
	11	1
	11	3
	12	1
	12	3
	16	1
	16	3
	17	1
	17	3
	18	1
	18	3
	20	1
	20	3
	21	1
	21	3
*	NULL	NULL

但是多对多关系表,Nhibernate 是不会生成相应实体的,所以不能直接删除。

解除单个关系，示例如下

```

/// <returns></returns>
public bool DeleteStudentCourse(int studentId, int courseId)
{
    bool result = false;
    using (var context=new HuatongApplyContext())
    {
        try
        {
            context.BeginTransaction();
            var tempStudent = context.Student.FirstOrDefault(p => p.Identification == studentId);
            if (tempStudent != null)
            {
                var tempCourses = tempStudent.CourseList;
                var deleteEntity = tempCourses.FirstOrDefault(p => p.Identification == courseId);
                tempCourses.Remove(deleteEntity);
            }
            context.CommitTransaction();
            result = true;
        }
        catch (Exception)
        {
            //
        }
    }
}

```

注意，一定要从 Nhibernate 已经跟踪到的关系实体集合中移除你要删除的数据

Nhibernate Profile 检测到的执行结果如下

Short SQL	Row Count	Duration
begin transaction with isolation level: Unspecified		
SELECT ... FROM [dbo].[tb_student] student0_ WHERE student0_[StudentId] = 8	1	72 ms /
SELECT ... FROM [dbo].[tb_studentcourses] courselist0_ left out... WHERE courselist0_[StudentId] = 8	5	1 ms / 2
DELETE FROM [dbo].[tb_studentcourses] WHERE [StudentId] = 8		36 ms
INSERT INTO [dbo].[tb_studentcourses] ...		32 ms
INSERT INTO [dbo].[tb_studentcourses] ...		
INSERT INTO [dbo].[tb_studentcourses] ...		
INSERT INTO [dbo].[tb_studentcourses] ...		
commit transaction		


Details	Stack Trace
<pre> 1 INSERT INTO [dbo].[tb_studentcourses] 2     ([StudentId], 3     [CourseId]) 4 VALUES 5     (8 /* @p0_0 */, 6     2 /* @p1_0 */) 7 8 9 INSERT INTO [dbo].[tb_studentcourses] 10    ([StudentId], 11    [CourseId]) 12 VALUES 13    (8 /* @p0_1 */, 14    3 /* @p1_1 */) 15 16 17 INSERT INTO [dbo].[tb_studentcourses] 18    ([StudentId], 19    [CourseId]) 20 VALUES 21    (8 /* @p0_2 */, 22    4 /* @p1_2 */) 23 24 </pre>	

Nhibernate 对于多对多关系实体的关系解除的处理是，先删除指定学生的所有选课信息，然后再将未移除的选课信息，再次插入。注意，我反复强调过。显示事务的使用。如果一些连续的操作，不用显示事务，每条语句都会自动使用隐式事务处理。而显示事务可以将事务的创建和销毁次数削减到最小。

以下的移除关系的写法是错的，关系不会解除

```
try
{
    context.BeginTransaction();
    var tempStudent = context.Student.FirstOrDefault(p => p.Identification == studentId);
    if (tempStudent != null)
    {
        #region 正确写法
        /*var tempCourses = tempStudent.CourseList;
        var deleteEntity = tempCourses.FirstOrDefault(p => p.Identification == courseId);
        tempCourses.Remove(deleteEntity);*/
        #endregion

        #region 错误写法，这种写法不会解除关系
        var tempCourses = tempStudent.CourseList.ToList();
        tempCourses.RemoveAll(p => p.Identification == courseId);
        #endregion
    }
    context.CommitTransaction();
    result = true;
}
```



前面已经强调过，所有关系实体的移除，应该是从Nh跟踪的关系实体集合中移除，上面的错误代码。执行了 tolist 操作，集合已经是另一个集合对象了，进行 remove 操作的时候，并没有从NHibernate跟踪的关系实体集合中删除数据，所以，上面的代码不能解除关系。

## 解除全部关系

Session #5			
Statements Entities Session Usage			
Short SQL			
		Row Count	Duration
begin transaction with isolation level: Unspecified			
SELECT ... FROM [dbo].[tb_student] student0_ WHERE student0_[StudentId] = 8		1	71 ms / 159 ms
SELECT ... FROM [dbo].[tb_studentcourses] courselist0_ left out... WHERE courselist0_[StudentId] = 8		4	10 ms / 30 ms
DELETE FROM [dbo].[tb_studentcourses] WHERE [StudentId] = 8			13 ms
commit transaction			
Details Stack Trace			
1 DELETE FROM [dbo].[tb_studentcourses]		Param	Value
2 WHERE [StudentId] = 8 /* @p0 */		@p0	8

Id 为 8 的学生所有的选课信息均被删除。

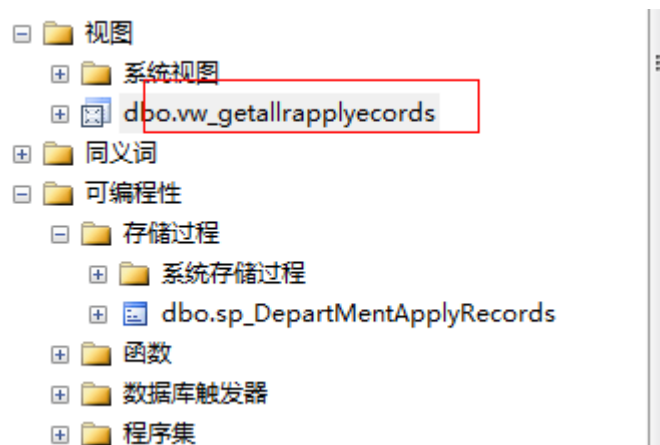
查看数据库，查看选课关系表，看有无被删除

	StudentId	CourseId
▶	1	2
	2	4
	2	5
	3	5
	3	6
	4	2
	11	1
	11	3
	12	1
	12	3
	16	1
	16	3
	17	1
	17	3
	18	1
	18	3
	20	1
	20	3
	21	1
	21	3
*	NULL	NULL

已经被成功删除。

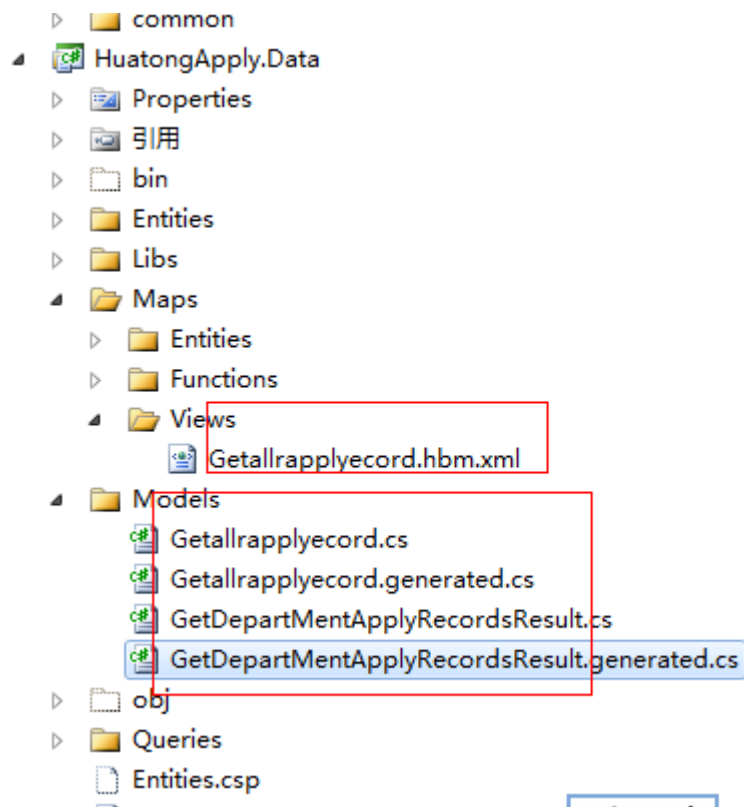
## 视图的使用

Demo 中数据库我编写了一个视图，获取所有报销单数据



生成代码后，对应的文件在 views 和 models 中。





说视图的使用，

很简单，像用表实体一样去用视图。没有什么难的，返回类型为集合的约定返回空集合。可以给上层方法节省不少判断代码。

```

List<ApplyRecordBO> recordBos=new List<ApplyRecordBO>();
try
{
    using (var context = new HuatongApplyContext())
    {
        var tempViewDatas = context.Getallrapplyecord.ToList();
        recordBos=tempViewDatas.Select(p => new ApplyRecordBO()
        {
            ApplyBy = p.ApplyBy,
            ApplyOn = p.ApplyOn,
            ApplyId = p.ApplyId,
            AuditStatus = p.AuditStatus,
            FinalAuditBy = p.FinalAuditBy,
            DepAuditBy = p.DepAuditBy,
            CreatedBy = p.CreatedBy,
            OrId = p.OrId,
            OrName = p.OrName,
            CreatedOn = p.CreatedOn,
            Currency = p.Currency
        }).ToList();
    }
}
catch (Exception)
{
    //异常写入
}
return recordBos;

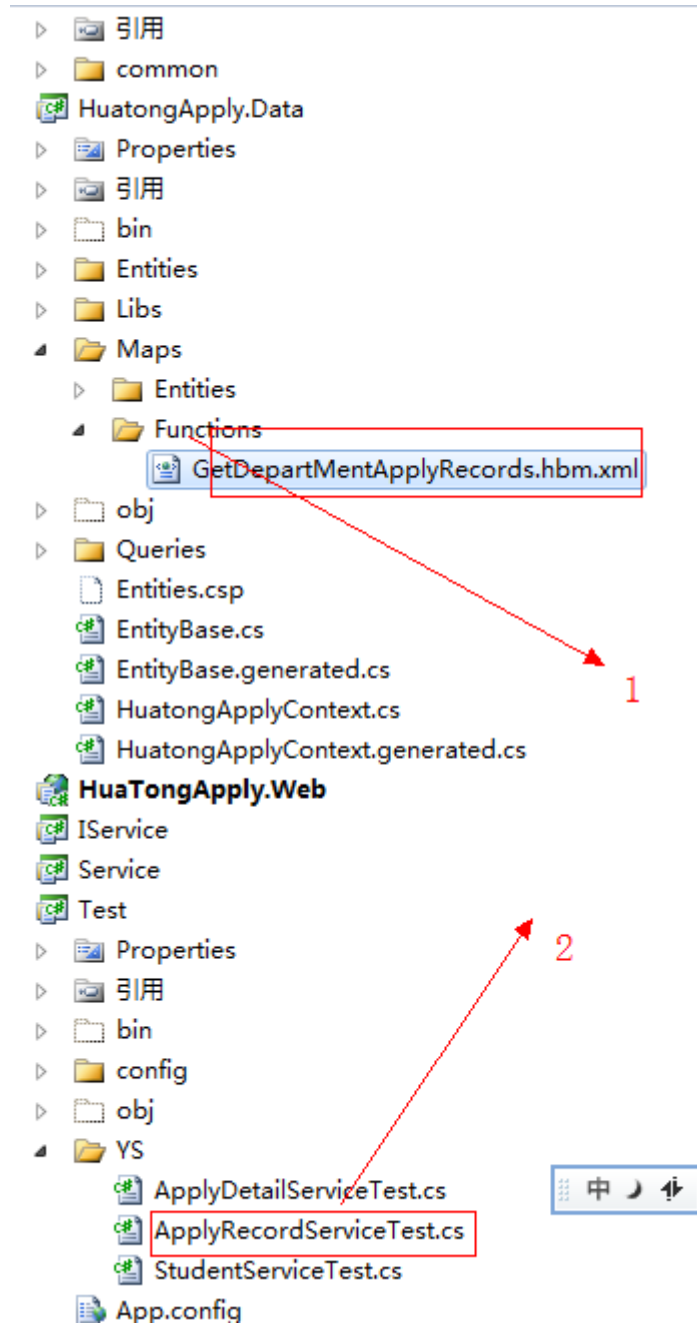
```

## 存储过程的使用

目前尚未找到让 Codesmith Generator 能正确生成调用存储过程代码的方法  
目前华通项目中都是手动配的。目前认定可能是需要改模版

Plinq0 NH 如果需要通过存储过程获取数据，则需要使用原生 Nhibernate 对存储过程支持的特性。

而 Plino 在封装的过程中，也是保留了直接使用 Nhibernate 对存储过程原生支持的特性，具体使用如下。



1 处，命名查询的映射文件


2 处，本 demo 中，测试存储过程调用方法的单元测试文件

手动编写调用存储过程的 hbm.xml 文件

```

<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2" namespace="HuatongApply.Data" assembly="HuatongApp]
  <sql-query name="GetDepartMentApplyRecords" >
    <return-scalar column="[ApplyId]" type="System.Int32" />
    <return-scalar column="[OrId]" type="System.Int32" />
    <return-scalar column="[Price]" type="System.Double" />
    <return-scalar column="[Currency]" type="System.String" />
    <return-scalar column="[ApplyBy]" type="System.String" />
    <return-scalar column="[ApplyOn]" type="System.DateTime" />
    <return-scalar column="[Description]" type="System.String" />
    <return-scalar column="[IsCar]" type="System.Int32" />
    <return-scalar column="[CarNo]" type="System.String" />
    <return-scalar column="[AuditStatus]" type="System.Int32" />
    <return-scalar column="[DepAuditBy]" type="System.String" />
    <return-scalar column="[FinanceAuditBy]" type="System.String" />
    <return-scalar column="[FinalAuditBy]" type="System.String" />
    <return-scalar column="[CreatedOn]" type="System.DateTime" />
    <return-scalar column="[CreatedBy]" type="System.String" />
    <return-scalar column="[ModifiedOn]" type="System.DateTime" />
    <return-scalar column="[ModifiedBy]" type="System.String" />
    <return-scalar column="[OrName]" type="System.String" />
    exec sp_DepartMentApplyRecords :pOrId
  </sql-query>
</hibernate-mapping>

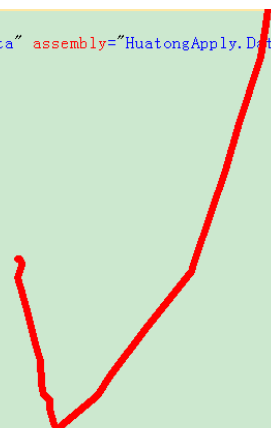
```



```

<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2" namespace="HuatongApply.Data" assembly="HuatongApply.Data">
  <sql-query name="GetDepartMentApplyRecords" >
    <return-scalar column="ApplyId" type="System.Int32" />
    <return-scalar column="Or_Id" type="System.Int32" />
    <return-scalar column="Price" type="System.Double" />
    <return-scalar column="Currency" type="System.String" />
    <return-scalar column="ApplyBy" type="System.String" />
    <return-scalar column="ApplyOn" type="System.DateTime" />
    <return-scalar column="Description" type="System.String" />
    <return-scalar column="IsCar" type="System.Int32" />
    <return-scalar column="CarNo" type="System.String" />
    <return-scalar column="AuditStatus" type="System.Int32" />
    <return-scalar column="DepAuditBy" type="System.String" />
    <return-scalar column="FinanceAuditBy" type="System.String" />
    <return-scalar column="FinalAuditBy" type="System.String" />
    <return-scalar column="CreatedOn" type="System.DateTime" />
    <return-scalar column="CreatedBy" type="System.String" />
    <return-scalar column="ModifiedOn" type="System.DateTime" />
    <return-scalar column="ModifiedBy" type="System.String" />
    <return-scalar column="OrName" type="System.String" />
    exec sp_DepartMentApplyRecords :pOrId
  </sql-query>
</hibernate-mapping>

```



上面的 xml 文件的编写，不要给字段加上[]，否则读不到任何数据并且，类型和名称应该和存储过程返回的数据信息一模一样，否则一定会报错。

存储过程使用中常见异常如下

1. 字段名称和数据库返回的数据名称不一样，其实就是我在写的时候，给字段加上了【】，而导致 reader 读不到对应的数据。
2. 数据库字段的顺序和 retur-scalar 中定义的字段顺序不一致，导致异常  
如果不一致，则会抛以下类型的异常。

```

message-could not execute query
exec sp_DepartMentApplyRecords @p0 ]
Name:pOrId - Value:1
SQL: exec sp_DepartMentApplyRecords @p0]
Source:NHibernate
QueryString=exec sp_DepartMentApplyRecords @p0
StackTrace:
在 NHibernate.Loader.Loader.DoList(ISessionImplementor session, QueryParameters queryParameters)
在 NHibernate.Loader.Loader.ListIgnoreQueryCache(ISessionImplementor session, QueryParameters queryParameters)
在 NHibernate.Loader.Loader.List(ISessionImplementor session, QueryParameters queryParameters, ISet`1 querySpaces, IType[] re:
在 NHibernate.Loader.Custom.CustomLoader.List(ISessionImplementor session, QueryParameters queryParameters)
在 NHibernate.Impl.SessionImpl.ListCustomQuery(ICustomQuery customQuery, QueryParameters queryParameters, IList results)
在 NHibernate.Impl.SessionImpl.List(NativeSQLQuerySpecification spec, QueryParameters queryParameters, IList results)
在 NHibernate.Impl.SessionImpl.List[T](NativeSQLQuerySpecification spec, QueryParameters queryParameters)
在 NHibernate.Impl.SqlQueryImpl.List[T]()
在 Service.Cost.ReimbursementControl.ApplyRecordService.GetApplyRecordBosByCallProcedures(Int32 orId) 位置 C:\Users\Administr:
atongApply\Service\Cost\ReimbursementControl\ApplyRecordService.cs:行号 384
InnerException: System.IndexOutOfRangeException
Message=[ApplyId]
Source=System.Data
StackTrace:
在 System.Data.ProviderBase.FieldNameLookup.GetOrdinal(String fieldName)
在 System.Data.SqlClient.SqlDataReader.GetOrdinal(String name)
在 NHibernate.Driver.NHybridDataReader.GetOrdinal(String name)
在 NHibernate.Type.NullableType.NullSafeGet(IDataReader rs, String name)
在 NHibernate.Type.NullableType.NullSafeGet(IDataReader rs, String name, ISessionImplementor session, Object owner)
在 NHibernate.Loader.Custom.CustomLoader.ScalarResultColumnProcessor.Extract(Object[] data, IDataReader resultSet, ISess:
在 NHibernate.Loader.Custom.CustomLoader.ResultRowProcessor.BuildResultRow(Object[] data, IDataReader resultSet, Boolean
在 NHibernate.Loader.Custom.CustomLoader.GetResultColumnOrRow(Object[] row, IResultTransformer resultTransformer, IDataR:
在 NHibernate.Loader.Loader.GetRowFromResultSet(IDataReader resultSet, ISessionImplementor session, QueryParameters quer:
tityKey optionalObjectKey, IList hydratedObjects, EntityKey[] keys, Boolean returnProxies)
在 NHibernate.Loader.Loader.DoQuery(ISessionImplementor session, QueryParameters queryParameters, Boolean returnProxies)
在 NHibernate.Loader.Loader.DoQueryAndInitializeNonLazyCollections(ISessionImplementor session, QueryParameters queryPar:
在 NHibernate.Loader.Loader.DoList(ISessionImplementor session, QueryParameters queryParameters)
InnerException:

```

3. retur-scalar 定义的字段比数据库返回的字段要多，导致异常。
4. 存储过程传参，正确传参的形式如下

```

<return-scalar column= CreatedBy type= System. String //
<return-scalar column="ModifiedOn" type="System. DateTime" //
<return-scalar column="ModifiedBy" type="System. String" //
<return-scalar column="OrName" type="System. String" //
exec sp_DepartMentApplyRecords :pOrId
</sql-query>
</hibernate-mapping>

```

或者

```

<return-scalar column= AuditStatus type= System. Int32 //
<return-scalar column="DepAuditBy" type="System. String" //
<return-scalar column="FinanceAuditBy" type="System. String" //
<return-scalar column="FinalAuditBy" type="System. String" //
<return-scalar column="CreatedOn" type="System. DateTime" //
<return-scalar column="CreatedBy" type="System. String" //
<return-scalar column="ModifiedOn" type="System. DateTime" //
<return-scalar column="ModifiedBy" type="System. String" //
<return-scalar column="OrName" type="System. String" //
exec sp_DepartMentApplyRecords :?
</sql-query>
</hibernate-mapping>

```

在 Service 调用存储过程的示例如下：

```

using (var context = new HuatongApplyContext())
{
    var list = context.Advanced.DefaultSession.GetNamedQuery("GetDepartMentApplyRecords")
        .SetInt32("pOrId", orId)
        .List<object[]>();

    result = list.Select(q => new ApplyRecordBO()
    {
        ApplyId = Convert.ToInt32(q[0]),
        OrId = Convert.ToInt32(q[1]),
        Price = Convert.ToDouble(q[2]),
        Currency = Convert.ToString(q[3]),
        ApplyBy = Convert.ToString(q[4]),
        ApplyOn = Convert.ToDateTime(q[5]),
        Description = Convert.ToString(q[6]),
        IsCar = Convert.ToInt32(q[7]),
        CarNo = Convert.ToString(q[8]),
        AuditStatus = Convert.ToInt32(q[9]),
        DepAuditBy = Convert.ToString(q[10]),
        FinanceAuditBy = Convert.ToString(q[10]),
        FinalAuditBy = Convert.ToString(q[10]),
        CreatedOn = Convert.ToDateTime(q[10]),
        CreatedBy = Convert.ToString(q[10])
    }).ToList();
}

```

要注意的是 123 处，

1 处，如果采用匿名传参，参数顺序和名称必须和上图 exec sp... 语句中的参数顺序一致，强烈建议命名传参。命名传参参数顺序也必须一致。

2 处，此处查询命名需要和以下地方的 name 保持一致

```

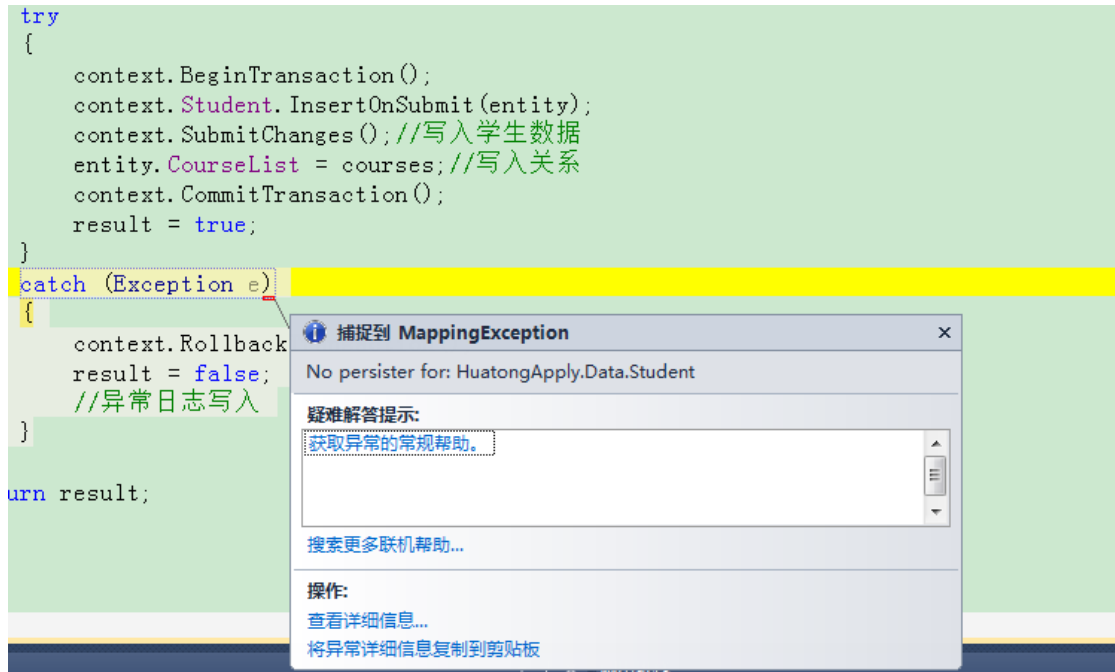
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" namespace="Huato:
    <sql-query name="GetDepartMentApplyRecords">
        <return-scalar column="ApplyId" type="System.Int32" />
        <return-scalar column="Or_Id" type="System.Int32" />
        <return-scalar column="Price" type="System.Double" />
        <return-scalar column="Currency" type="System.String" />
        <return-scalar column="ApplyBy" type="System.String" />
        <return-scalar column="ApplyOn" type="System.DateTime" />
        <return-scalar column="Description" type="System.String" />
        <return-scalar column="IsCar" type="System.Int32" />
        <return-scalar column="CarNo" type="System.String" />
    </sql-query>

```

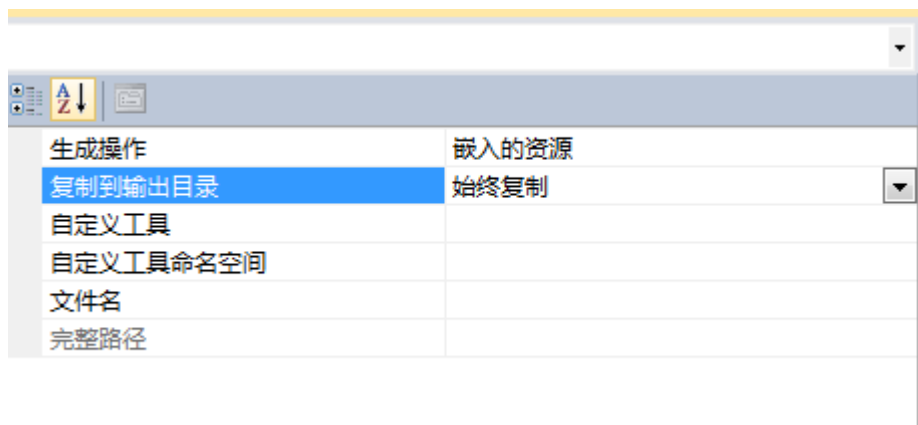
3 处，顺序一定要正确，防止类型转换抛异常。

## Nhibernate 常见异常处理

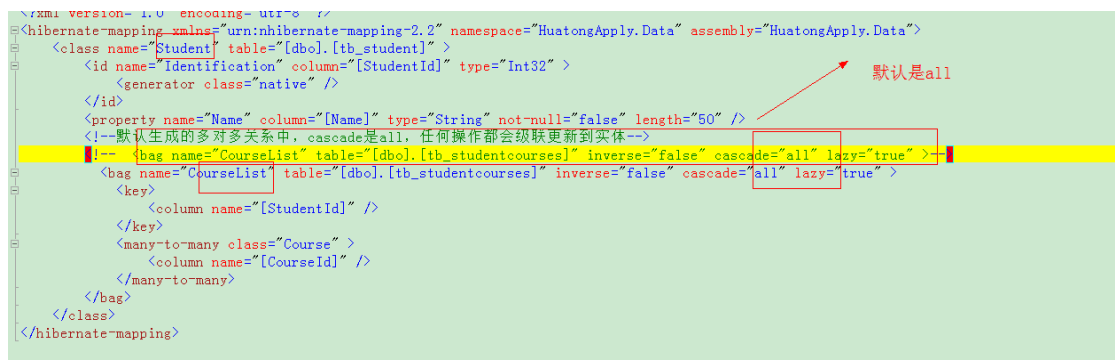
未找到映射文件



解决办法，将选中对应的 hbm.xml 文件，属性设置为嵌入资源，每次都复制。



## Cascade 属性说明



如上图，学生-课程的关系是多对多，生成的 hbm.xml 中，cascade 属性是

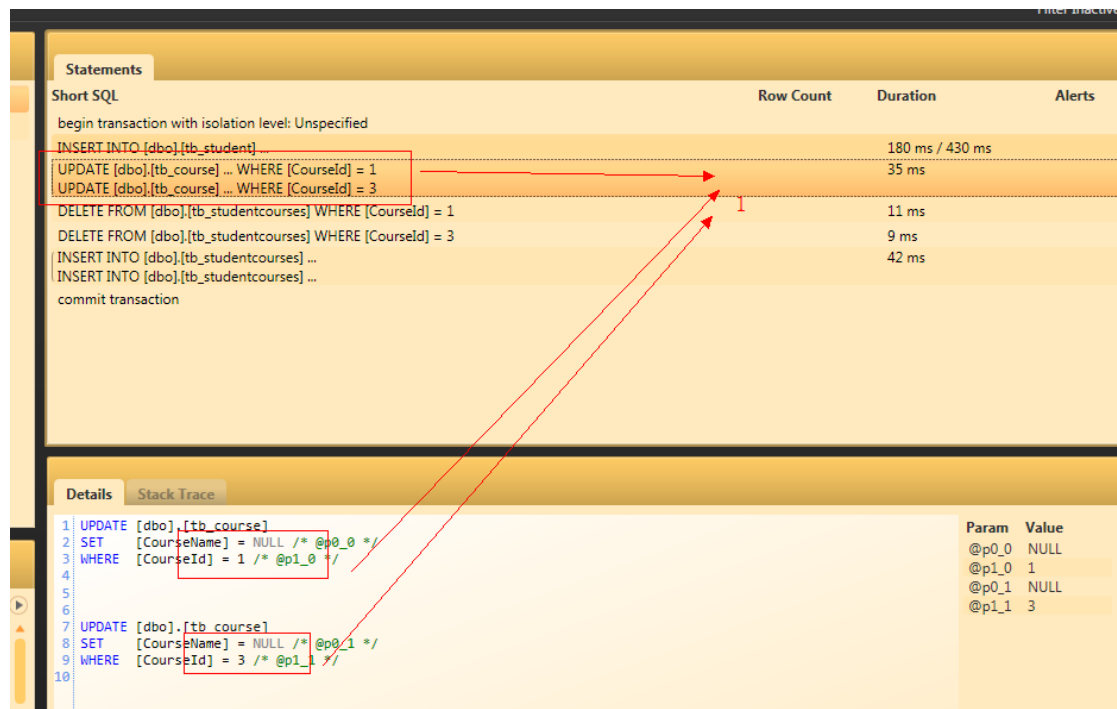
all，也就是任何操作都会级联更新到关系实体

测试 Demo 中的方法 AddStudent

```
[Test]
public void AddStudentTest()
{
    StudentBo tempBo = new StudentBo();
    tempBo.Name = "张三";
    tempBo.CourseListBos=new List<CourseBo>() {new CourseBo() {Identification = 1},new CourseBo() {Identification = 3}};
    bool temp=iService.AddStudent(tempBo);
    Assert.AreEqual(true,temp);
}
```

在添加课程集合的时候，我们只设置了 id，而没有设置 Name，那么 Nhibernate 如何执行？id 为 1 和 3 的课程是存在的

用 Nhibernate 监测发现如下结果



Statements	Row Count	Duration	Alerts
begin transaction with isolation level: Unspecified			
INSERT INTO [dbo].[tb_student]		180 ms / 430 ms	
UPDATE [dbo].[tb_course] ... WHERE [CourseId] = 1	1	35 ms	
UPDATE [dbo].[tb_course] ... WHERE [CourseId] = 3			
DELETE FROM [dbo].[tb_studentcourses] WHERE [CourseId] = 1		11 ms	
DELETE FROM [dbo].[tb_studentcourses] WHERE [CourseId] = 3		9 ms	
INSERT INTO [dbo].[tb_studentcourses] ...		42 ms	
INSERT INTO [dbo].[tb_studentcourses] ...			
commit transaction			

Details	Stack Trace	Param	Value
1 UPDATE [dbo].[tb_course]		@p0_0	NULL
2 SET [CourseName] = NULL /* @p0_0 */		@p1_0	1
3 WHERE [CourseId] = 1 /* @p1_0 */		@p0_1	NULL
4		@p1_1	3
5			
6			
7 UPDATE [dbo].[tb_course]			
8 SET [CourseName] = NULL /* @p0_1 */			
9 WHERE [CourseId] = 3 /* @p1_1 */			
10			

发现，更新 id 为 1，3 课程的名称为 null，查看两条 delete 语句



```
UPDATE [dbo].[tb_course] ... WHERE [CourseId] = 1
UPDATE [dbo].[tb_course] ... WHERE [CourseId] = 3
DELETE FROM [dbo].[tb_studentcourses] WHERE [CourseId] = 1
DELETE FROM [dbo].[tb_studentcourses] WHERE [CourseId] = 3
INSERT INTO [dbo].[tb_studentcourses] ...
INSERT INTO [dbo].[tb_studentcourses] ...
commit transaction
```

Details

Stack Trace

```
1 DELETE FROM [dbo].[tb_studentcourses]
2 WHERE [CourseId] = 3 /* @p0 */
```

发现从选课表中删除了所有 id 为 1 和 3 的选课

查看两条 Insert 语句

```
INSERT INTO [dbo].[tb_studentcourses] ...
INSERT INTO [dbo].[tb_studentcourses] ...
commit transaction
```

Details

Stack Trace

```
1 INSERT INTO [dbo].[tb_studentcourses]
2 ([StudentId],
3 [CourseId])
4 VALUES (7 /* @p0_0 */,
5 1 /* @p1_0 */)
6
7
8
9 INSERT INTO [dbo].[tb_studentcourses]
10 ([StudentId],
11 [CourseId])
12 VALUES (7 /* @p0_1 */,
13 3 /* @p1_1 */)
14
```

两条 Insert 是插入了新的选课关系

所以，当 cascade=all 的时候，插入带课程信息的学生数据，直接级联更新了课程信息，并且从选课表中删除了所有 id 为 1 和 3 的选课信息，这是不合理的。因为这个操作会删除别人的选课数据。

这一个场景，也适用与角色-权限的授权关系。

**Tips: 多对多关系中，如果不希望发生级联操作，请设置 cascade=none，无级联操作。**

本 Demo 中 Student 和 Course 的 cascade 设置如下

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2" namespace="HuatongApply.Data" assembly="HuatongApply.Data">
  <class name="Course" table="[dbo].[tb_course]">
    <id name="Identification" column="[CourseId]" type="Int32">
      <generator class="native" />
    </id>
    <property name="Name" column="[CourseName]" type="String" not-null="false" length="50" />
    <bag name="StudentList" table="[dbo].[tb_studentcourses]" inverse="false" cascade="all" lazy="true">
      <key>
        <column name="[CourseId]" />
      </key>
      <many-to-many class="Student">
        <column name="[StudentId]" />
      </many-to-many>
    </bag>
  </class>
</hibernate-mapping>
```

```
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2" namespace="HuatongApply.Data" assembly="HuatongApply.Data">
  <class name="Student" table="[dbo].[tb_student]">
    <id name="Identification" column="[StudentId]" type="Int32">
      <generator class="native" />
    </id>
    <property name="Name" column="[Name]" type="String" not-null="false" length="50" />
    <!-- 默认生成的多对多关系中，cascade是all，任何操作都会级联更新到关联实体-->
    <!-- <bag name="CourseList" table="[dbo].[tb_studentcourses]" inverse="false" cascade="all" lazy="true" -->
    <bag name="CourseList" table="[dbo].[tb_studentcourses]" inverse="false" cascade="none" lazy="true">
      <key>
        <column name="[StudentId]" />
      </key>
      <many-to-many class="Course">
        <column name="[CourseId]" />
      </many-to-many>
    </bag>
  </class>
</hibernate-mapping>
```

再插入一条带选课信息的学生数据

查看 sql 的执行结果。

一条数据，插入到学生表

两条数据写入到选课关系表。

未发生级联操作。正常

Short SQL	Row Count	Duration
begin transaction with isolation level: Unspecified		
INSERT INTO [dbo].[tb_student] ...		82 ms / 172 ms
INSERT INTO [dbo].[tb_studentcourses] ...		30 ms
INSERT INTO [dbo].[tb_studentcourses] ...		
commit transaction		

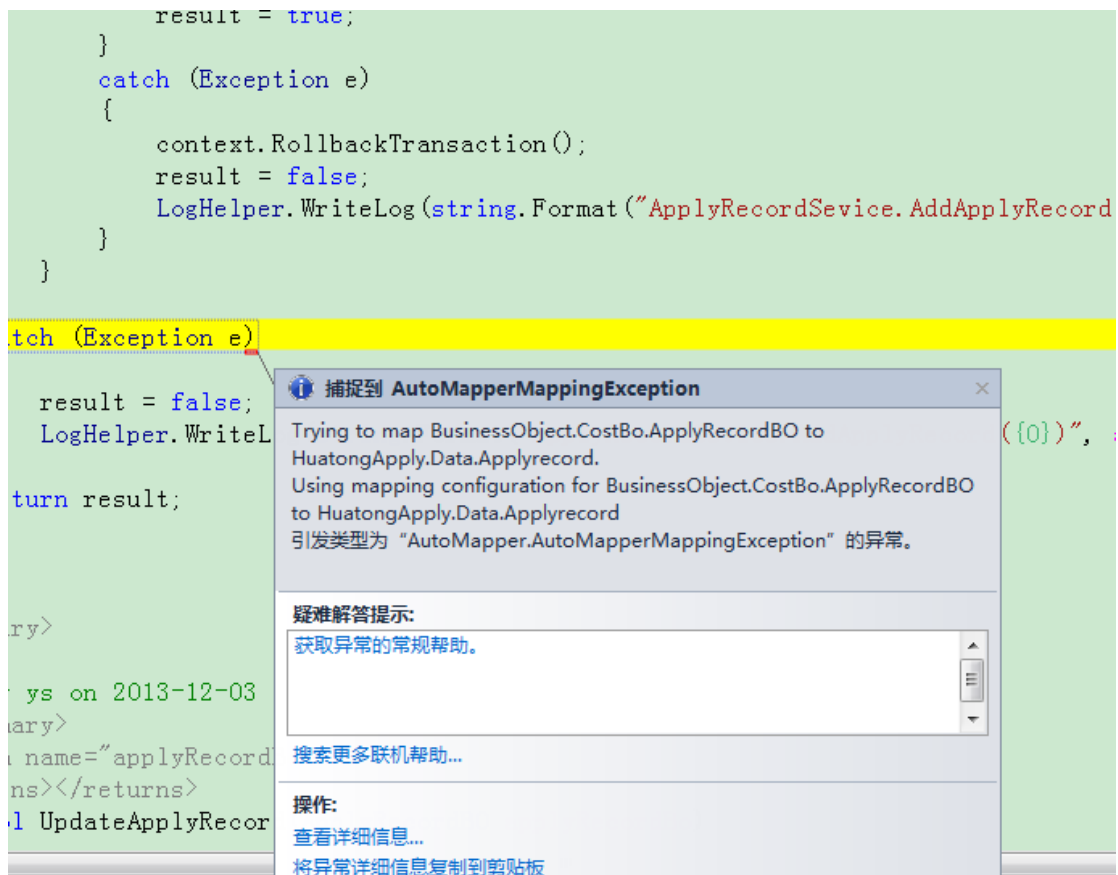
  

Details	Stack Trace										
<pre> 1 INSERT INTO [dbo].[tb_studentcourses] 2     ([StudentId], 3      [CourseId]) 4 VALUES 5     (9 /* @p0_0 */, 6      1 /* @p1_0 */) 7 8 9 INSERT INTO [dbo].[tb_studentcourses] 10     ([StudentId], 11      [CourseId]) 12 VALUES 13     (9 /* @p0_1 */, 14      3 /* @p1_1 */) </pre>	<table> <tr> <th>Param</th><th>Value</th></tr> <tr> <td>@p0_0</td><td>9</td></tr> <tr> <td>@p1_0</td><td>1</td></tr> <tr> <td>@p0_1</td><td>9</td></tr> <tr> <td>@p1_1</td><td>3</td></tr> </table>	Param	Value	@p0_0	9	@p1_0	1	@p0_1	9	@p1_1	3
Param	Value										
@p0_0	9										
@p1_0	1										
@p0_1	9										
@p1_1	3										

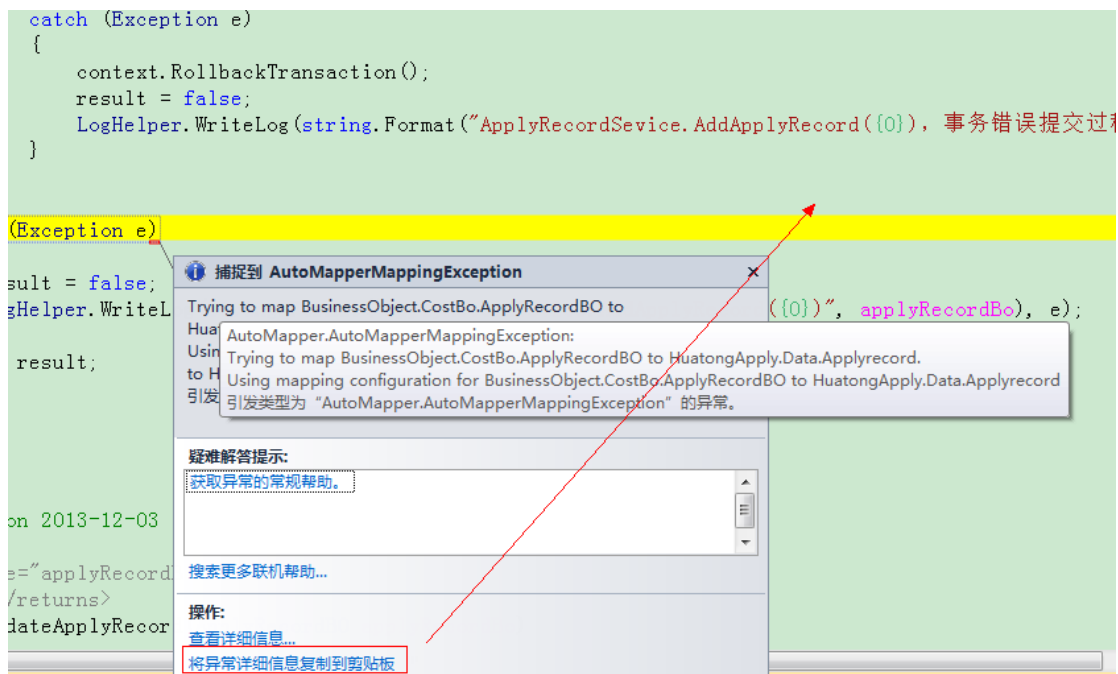
Tips, 一对多关系, 多对多关系, 如果实体更新和插入出现了级联的异常  
 请检查 hbm.xml 中的 cascade 和 inverse 属性设置。

## AutoMapper 的使用和异常处理

当 AutoMapper 抛异常的时候, 不要惊慌。



请按照如下操作



将异常信息复制到剪贴板

打开记事板，粘贴，然后直接看 InnerException。你看到的异常信息是整个异常堆栈，是非常详细的，如果 InnerException 是 null，那么，请往上看，看父 Exception。

如下

```

捕捉到 AutoMapper.AutoMapperMappingException
Message=Trying to map BusinessObject.CostBo.ApplyRecordBO to HuatongApply.Data.ApplyRecordBO.Using mapping configuration for BusinessObject.CostBo.ApplyRecordBO to
HuatongApply.Data.ApplyRecordBO引发类型为“AutoMapper.AutoMapperMappingException”的异常。
Source=AutoMapper
StackTrace:
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
    在 AutoMapper.MappingEngine.Map(Object source, Type sourceType, Type destinationType)
    在 AutoMapper.MappingEngine.Map[TSource,TDestination](TSource source)
    在 AutoMapper.Mapper.Map[TSource,TDestination](TSource source)
    在 Service.Cost.ReimbursementControl.ApplyRecordService.AddApplyRecord(ApplyRecordBO applyRecordBO) 位置 C:\Users\Administrator\Documents\CodeSmith Generator\Templates
\PLINQO NH\HuatongApply\Service\Cost\ReimbursementControl\ApplyRecordService.cs:行号 61
InnerException: AutoMapper.AutoMapperMappingException
Message=Trying to map System.Single to System.Nullable`1[System.Double,mscorlib,Version=4.0.0.0,Culture=neutral,PublicKeyToken=b77a5c561934e089]].Using mapping
configuration for BusinessObject.CostBo.ApplyRecordBO to HuatongApply.Data.ApplyRecordBO引发类型为“AutoMapper.AutoMapperMappingException”的异常。
Source=AutoMapper
StackTrace:
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.MapPropertyValue(ResolutionContext context, IMappingEngineRunner mapper, Object
mappedObject, PropertyMap propertyMap)
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.Map(ResolutionContext context, IMappingEngineRunner mapper)
    在 AutoMapper.Mapper.Map(ResolutionContext context, IMappingEngineRunner mapper)
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
InnerException: System.InvalidCastException
Message=指定的转换无效。
Source=HuatongApply.Data
StackTrace:
    在 SetPrice(Object , Object )
    在 AutoMapper.Internal.PropertyAccessor.SetValue(Object destination, Object value)
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.AssignValue(PropertyMap propertyMap, Object mappedObject, Object
propertyValue[Assign])
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.MapPropertyValue(ResolutionContext context, IMappingEngineRunner mapper, Object
mappedObject, PropertyMap propertyMap)
InnerException:
|

```

1 处为 InnerException,直接看他就可以了。这是引发的内部异常。

2 处外层 Exception，如果没有 InnerException，看这个。

3 处。这里的消息，是我们解决问题的入口。

```

捕捉到 AutoMapper.AutoMapperMappingException
Message=Trying to map System.Collections.Generic.List`1[[HuatongApply.Data.Applydetail, HuatongApply.Data, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null]] to System.Collections.Generic.List`1[[BusinessObject.CostBo.ApplydetailBO, BusinessObject,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]].引发类型为“AutoMapper.AutoMapperMappingException”的异常。
Source=AutoMapper
StackTrace:
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
    在 AutoMapper.MappingEngine.Map(Object source, Type sourceType, Type destinationType)
    在 AutoMapper.MappingEngine.Map[TSource,TDestination](TSource source)
    在 AutoMapper.Mapper.Map[TSource,TDestination](TSource source)
    在 Service.Cost.ReimbursementControl.ApplyRecordService.GetApplyDetailsByRecordId(Int32 id) 位置 C:\Users\Administrator
\Documents\CodeSmith Generator\Templates\PLINQO NH\HuatongApply\Service\Cost\ReimbursementControl\ApplyRecordService.cs:行号 356
InnerException: AutoMapper.AutoMapperMappingException
Message=Trying to map HuatongApply.Data.Applydetail to BusinessObject.CostBo.ApplydetailBO.Using mapping configuration for
HuatongApply.Data.Applydetail to BusinessObject.CostBo.ApplydetailBO引发类型为“AutoMapper.AutoMapperMappingException”的异常。
Source=AutoMapper
StackTrace:
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
    在 AutoMapper.Mappers.EnumerableMapperBase`1.Map(ResolutionContext context, IMappingEngineRunner mapper)
    在 AutoMapper.Mappers.CollectionMapper.Map(ResolutionContext context, IMappingEngineRunner mapper)
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
InnerException: AutoMapper.AutoMapperMappingException
Message=Trying to map System.Double to System.Nullable`1[System.Single,mscorlib,Version=4.0.0.0,Culture=neutral,
PublicKeyToken=b77a5c561934e089]].Using mapping configuration for HuatongApply.Data.Applydetail to
BusinessObject.CostBo.ApplydetailBODestination property: Price引发类型为“AutoMapper.AutoMapperMappingException”的异常。
Source=AutoMapper
StackTrace:
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.MapPropertyValue(ResolutionContext
context, IMappingEngineRunner mapper, Object mappedObject, PropertyMap propertyMap)
    在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.Map(ResolutionContext context,
IMappingEngineRunner mapper)
    在 AutoMapper.Mapper.Map(ResolutionContext context, IMappingEngineRunner mapper)
    在 AutoMapper.MappingEngine.AutoMapper.IMappingEngineRunner.Map(ResolutionContext context)
InnerException: System.InvalidCastException
Message=指定的转换无效。

```

上面的 3 处的英语异常信息的意思是，Price 映射出现了异常。所以我们需要看下，Price 的数据类型定义，还有我们 bo 中 Price 属性的定义，包括类型和名称，名称要一致，类型要平台兼容。

来看图，看 ApplyRecord 表的 xml 文件

发现 Price 是 double 类型

```

class name="Applyrecord" table=" [dbo].[tb_applyrecord]" >
  <id name="ApplyId" column="[ApplyId]" type="Int32" >
    <generator class="native" />
  </id>
  <property name="OrId" column="[Or_Id]" type="Int32" not-null="false" />
  <property name="Price" column="[Price]" type="Double" not-null="false" />
  <property name="Currency" column="[Currency]" type="AnsiString" not-null="false" len
  <property name="ApplyBy" column="[ApplyBy]" type="AnsiString" not-null="false" lengt
  <property name="ApplyOn" column="[ApplyOn]" type="DateTime" not-null="false" />
  <property name="Description" column="[Description]" type="AnsiString" not-null="fals
  <property name="IsCar" column="[IsCar]" type="Int32" not-null="false" />
  <property name="CarNo" column="[CarNo]" type="AnsiString" not-null="false" length="2
  <property name="AuditStatus" column="[AuditStatus]" type="Int32" not-null="false" />
  <property name="DepAuditBy" column="[DepAuditBy]" type="AnsiString" not-null="false"
  <property name="FinanceAuditBy" column="[FinanceAuditBy]" type="AnsiString" not-null
  <property name="FinalAuditBy" column="[FinalAuditBy]" type="AnsiString" not-null="fa
  <property name="CreatedOn" column="[CreatedOn]" type="DateTime" not-null="false" />
  <property name="CreatedBy" column="[CreatedBy]" type="AnsiString" not-null="false" l
  <property name="ModifiedOn" column="[ModifiedOn]" type="DateTime" not-null="false" /
  <property name="ModifiedBy" column="[ModifiedBy]" type="AnsiString" not-null="false"
  <bag name="ApplydetailList" inverse="true" cascade="all-delete-orphan" lazy="true" >
    <key>

```

再看我们定义的 bo 中类型

```

    /// <summary>
    /// 报销部门主键
    /// </summary>
    public Int32? OrId { set; get; }

    /// <summary>
    /// 部门名称
    /// </summary>
    public string OrName { get; set; }

    /// <summary>
    /// 报销金额
    /// </summary>
    public Single? Price { set; get; }

    /// <summary>
    /// 报销币种
    /// </summary>
    public string Currency { set; get; }

    /// <summary>
    /// 经手人
    /// </summary>
    public string ApplyBy { get; set; }

    /// <summary>
    /// 报销单填写时间

```

Bo 中的是 Single 类型，不一致，改为一致即可。

注意，我说的数据兼容是由方向决定的，entity 中是 double，bo 是 float。AutoMapper 在映

赋值的时候，double 类型的数据赋给 float 类型，是会报错的。  
而，反过来，float 赋给 double 则是兼容的。不会出异常。

**Tips:** 而业界一条规则，约定强过配置。entity 和 bo 中属性类型要完全一致，这样双向映射就不会有类型异常的问题了。

再如

```
捕捉到 AutoMapper.AutoMapperMappingException
Message=Trying to map System.Collections.Generic.List`1[[HuatongApply.Data.Applydetail, HuatongApply.Data, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]] to System.Collections.Generic.List`1[[BusinessObject.CostBo.ApplydetailBO, BusinessObject, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]].引发类型为“A AutoMapper.AutoMapperMappingException”的异常。
Source=A AutoMapper
StackTrace:
在 AutoMapper.MappingEngine.Automapper.IMappingEngineRunner.Map(ResolutionContext context)
在 AutoMapper.MappingEngine.Map(Object source, Type sourceType, Type destinationType)
在 AutoMapper.MappingEngine.Map[TSource,TDestination](TSource source)
在 AutoMapper.Mapper.Map[TSource,TDestination](TSource source)
在 Service.Cost.ReimbursementControl.ApplyRecordService.GetApplyDetailsByRecordId(Int32 id) 位置 C:\Users\Administrator\Documents\CodeSmith Generator\Templates\FLINQO NH\HuatongApply\Service\Cost\ReimbursementControl\ApplyRecordService.cs:行号 356
InnerException: AutoMapper.AutoMapperMappingException
Message=Trying to map HuatongApply.Data.Applydetail to BusinessObject.CostBo.ApplydetailBO.Using mapping configuration for huatongApply.Data.Applydetail to BusinessObject.CostBo.ApplydetailBO引发类型为“A AutoMapper.AutoMapperMappingException”的异常。
Source=A AutoMapper
StackTrace:
在 AutoMapper.MappingEngine.Automapper.IMappingEngineRunner.Map(ResolutionContext context)
在 AutoMapper.Mappers.EnumerableMapperBase`1.Map(ResolutionContext context, IMappingEngineRunner mapper)
在 AutoMapper.Mappers.CollectionMapper.Map(ResolutionContext context, IMappingEngineRunner mapper)
在 AutoMapper.MappingEngine.Automapper.IMappingEngineRunner.Map(ResolutionContext context)
InnerException: AutoMapper.AutoMapperMappingException
Message=Trying to map System.Double to System.Nullable`1[[System.Single, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]].Using mapping configuration for HuatongApply.Data.Applydetail to BusinessObject.CostBo.ApplydetailBODestination property: Price引发类型为“A AutoMapper.AutoMapperMappingException”的异常。
Source=A AutoMapper
StackTrace:
在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.MapPropertyValue(ResolutionContext context, IMappingEngineRunner mapper, Object mappedObject, PropertyMap propertyMap)
在 AutoMapper.Mappers.TypeMapObjectMapperRegistry.PropertyMapMappingStrategy.Map(ResolutionContext context, IMappingEngineRunner mapper)
在 AutoMapper.Mappers.TypeMapMapper.Map(ResolutionContext context, IMappingEngineRunner mapper)
在 AutoMapper.MappingEngine.Automapper.IMappingEngineRunner.Map(ResolutionContext context)
InnerException: System.InvalidCastException
Message=指定的转换无效。
```

上面圈起来的地方，就已经说明，出问题的地点。

AutoMapper 出现最常见的几种异常中，没有映射规则，和属性类型不一致是非常常见的，注意即可。

## 1. Nhibernate Profile 工具的使用

相关使用和配置见 [wiki=>Nhibernate 性能分析利器](#)