**一、映射定义概括**

**1.1 映射定义（Mapping declaration）**

　　对象和关系数据库之间的映射是用一个XML文档(XML document)来定义的。这个映射文档被设计为易读的，并且可以手工修改。映射语言是以对象为中心的， 意味着映射是按照持久化类的定义来创建的，而非表的定义。

　　让我们打开上节课的映射例子：

IMG_256

[IMG_257](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

<?xml version="1.0" encoding="utf-8" ?>  
  
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Product" table="T\_Product" lazy="true" >  
    <id name="ID" column="ID" type="Guid" >  
      <generator class="assigned" />  
    </id>  
  
    <property name="Code" type="string">  
      <column name="Code" length="50"/>  
    </property>  
  
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
      
    <property name="QuantityPerUnit" type="string">  
      <column name="QuantityPerUnit" length="50"/>  
    </property>  
  
    <property name="Unit" type="string">  
      <column name="Unit" length="50"/>  
    </property>  
  
  
    <property name="SellPrice" type="decimal">  
      <column name="SellPrice" precision="14" scale="2"/>  
    </property>  
  
    <property name="BuyPrice" type="decimal">  
      <column name="BuyPrice" precision="14" scale="2"/>  
    </property>  
  
    <property name="Remark" type="string">  
      <column name="Remark" length="200"/>  
    </property>  
  
  </class>  
</hibernate-mapping>

[IMG_258](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

 　　其中xmlns为xml命名空间，我们能够在NHibernate的分发包src\nhibernate-mapping.xsd里找到XSD文件。我们可以将.xsd文件拷贝到Visual Studio安装目录的\Xml\Schemas中，以便带来NHibernate持久化类配置的提示功能。

**二、基本映射常用标签**

　　作为NHibernate这个ORM框架来说，一个实体类对应的是数据库中的一张表；类中的一个属性对应表中的一个字段；一个对象对应的是表中的一条记录。

**2.1 hibernate-mapping**

　　<hibernate-mapping>标签是NHibernate映射文件的根节点。

　　<hibernate-mapping                                     
　      schema="schemaName"                         数据库schema名称。

         default-cascade="none|save-update"      默认的级联风格，(可选 - 默认为 none):。

         auto-import="true|false"   　　　　　      指定我们在使用查询语句的时候是否可以使用非全限定名。

         assembly="AssemblyName"

         namespace="Namespace"　　　　　        指定映射文件中的类的应用程序集名称和其所在的名称空间名，用来生成类的非全限定名。

　　/>

**2.2** **class**

　　<class>标签是定义一个持久化类的。

　　<class

name="ClassName" 持久化类的类名，这里可以是类的全名。

table="tableName" 对应的数据库表名。

discriminator-value="discriminator\_value" 辨别值，一个用于区分不同的子类的值，在多态行为时使用（在后面继承映射的课程中会讲到）。

mutable="true|false" 表明该类的实例可变。

schema="owner" 覆盖在根元素中指定的schema名字。

proxy="ProxyInterface" 指定一个接口，在延迟装载时作为代理使用。你可以在这里使用该类自己的名字。

dynamic-update="true|false" 指定用于UPDATE 的SQL将会在运行时动态生成，并且只更新那些改变过的字段。

dynamic-insert="true|false" 指定用于 INSERT的 SQL 将会在运行时动态生成，并且只包含那些非空值字段。

select-before-update="true|false" 指定NHibernate除非确定对象的确被修改了，UPDATE操作。

polymorphism="implicit|explicit" 界定是隐式还是显式的使用查询多态。

where="arbitrary sql where condition" 指定一个附加的SQL WHERE 条件，在抓取这个类的对象时会一直增加这个条件。

persister="PersisterClass" 指定一个定制的 IClassPersister。

batch-size="N" 指定一个用于根据标识符抓取实例时使用的"batch size"（批次抓取数量），默认值为1。

optimistic-lock="none|version|dirty|all" 乐观锁定，决定乐观锁定的策略。

lazy="true|false" 是否启用延迟加载。

abstract="true|false" 是否为抽象类。

　　/>

**2.3 id**

　　<id>标签定义了该属性到数据库表主键字段的映射。

　　<id

　　  name="PropertyName"                                   标识属性的名字。  
        type="typename"                                          NHibernate类型的名字  
        column="column\_name"                                主键字段的名字。  
        unsaved-value="any|none|null|id\_value"         一个特定的标识属性值，用来标志该实例是刚刚创建的，尚未保存。  
        access="field|property|nosetter|ClassName"    NHibernate用来访问属性值的策略。  
        <generator class="generatorClass"/>  
　　</id>

 　　generator：主键生成策略。NHibernate提供了以下几种生成策略：

|  |  |  |
| --- | --- | --- |
| **名称** | **说明** | **是否常用** |
| **increment** | 用于为int类型生成 唯一标识。只有在没有其他进程往同一张表中插入数据时才能使用 | N |
| **identity** | 对DB2,MySQL, MS SQL Server, Sybase和HypersonicSQL的内置标识字段提供支持。数据库返回的主键值 返回的标识符是int类型的。 | N |
| **sequence** | 在DB2,PostgreSQL, Oracle, SAP DB, McKoi中使用序列（sequence)， 而在Interbase中使用生成器(generator)。返回的标识符是int类型的。 | Y |
| **seqhilo** | 使用一个高/低位算法来高效的生成int类型的标识符，给定一个数据库序列（sequence)的名字。 | N |
| **uuid.hex** | 用一个System.Guid的ToString()方法法生成字符串类型的标识符， 字符串的长度由format参数定义。 | N |
| **uuid.string** | 用一个新的System.Guid实例的byte[]转化为字符串作为标示符。 | N |
| **guid** | 使用新的System.Guid实例作为标示符。 |  |
| **guid.comb** | 使用Jimmy Nilsson的算法生成一个新的System.Guid标示符。 |  |
| **native** | 根据底层数据库的能力选择identity, sequence 或者hilo中的一个。 | Y |
| **assigned** | 让应用程序在 Save()之前为对象分配一个标示符。 | Y |
| **foreign** | 使用另外一个相关联的对象的标识符。通常和<one-to-one>联合起来使用。 | Y |

**2.4 composite-id**

　　<composite-id>为联合主键。

　　<composite-id                                                          
        name="PropertyName"                                                              
        class="ClassName"                                               联合主键类的类名。  
        unsaved-value="any|none"                                     
        access="field|property|nosetter|ClassName">  
  
        <key-property name="PropertyName" type="typename" column="column\_name"/>             联合主键的属性  
        <key-many-to-one name="PropertyName class="ClassName" column="column\_name"/>      联合主键多对一属性  
        ......  
　　</composite-id>

　　注意的是，若使用联合主键，你的持久化类必须重载 Equals()和GetHashCode()方法。例子如下：

[IMG_259](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

<?xml version="1.0" encoding="utf-8" ?>  
  
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Product" table="T\_Product" lazy="true" >  
      
    <composite-id name="ID" class="ProductID">  
      <key-property name="Name" type="string">  
        <column name="Name" length="50"/>  
      </key-property>  
  
      <key-property name="QuantityPerUnit" type="string">  
        <column name="QuantityPerUnit" length="50"/>  
      </key-property>  
  
    </composite-id>  
  
    <property name="Unit" type="string">  
      <column name="Unit" length="50"/>  
    </property>  
  
  
    <property name="SellPrice" type="decimal">  
      <column name="SellPrice" precision="14" scale="2"/>  
    </property>  
  
    <property name="BuyPrice" type="decimal">  
      <column name="BuyPrice" precision="14" scale="2"/>  
    </property>  
  
    <property name="Remark" type="string">  
      <column name="Remark" length="200"/>  
    </property>  
  
  </class>  
</hibernate-mapping>

[IMG_260](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

[IMG_261](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace Domain  
{  
    public class Product  
    {  
        public virtual ProductID ID { get; set; }  
  
        public virtual string Unit { get; set; }  
  
        public virtual decimal SellPrice { get; set; }  
  
        public virtual decimal BuyPrice { get; set; }  
  
        public virtual string Remark { get; set; }  
    }  
  
    public class ProductID  
    {  
        public virtual string Name { get; set; }  
  
        public virtual string QuantityPerUnit { get; set; }  
  
        public override bool Equals(object obj)  
        {  
            var entity = obj as ProductID;  
            if (entity == null)  
            {  
                return false;  
            }  
  
            return entity.Name == this.Name  
                && entity.QuantityPerUnit == this.QuantityPerUnit;  
        }  
  
        public override int GetHashCode()  
        {  
            return base.GetHashCode();  
        }  
    }  
}

[IMG_262](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

　　运行效果如图2.4.1所示，包含联合主键的表已经生成成功。

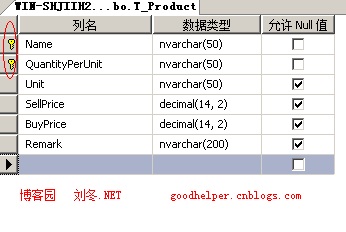


图2.4.1

**2.5 property**

　　<property>标签是类定义了一个持久化类的属性。

　　<property  
        name="propertyName"                          属性名  
        column="column\_name"                        对应的数据库字段名。  
        type="typename"                                 NHibernate类型的名字。  
        update="true|false"                              update时是否包含该字段。  
        insert="true|false"                                insert时时候包含该字段。  
        formula="arbitrary SQL expression"       一个SQL表达式，定义了这个计算 （computed） 属性的值。计算属性没有和它对应的数据库字段  
        access="field|property|ClassName"        NHibernate用来访问属性值的策略。  
        optimistic-lock="true|false"                   乐观锁定  
        generated="never|insert|always"           表明此属性值是否是由数据库生成。  
　　/>

**三、NHibernate的类型**

**3.1 值类型映射表**

| **NHibernate类型** | **.NET类型** | **Database类型** | **备注** |
| --- | --- | --- | --- |
| AnsiChar | System.Char | DbType.AnsiStringFixedLength - 1 char |  |
| Boolean | System.Boolean | DbType.Boolean | 在没有指定类型(type) 属性时的默认值。 |
| Byte | System.Byte | DbType.Byte | 在没有指定类型(type) 属性时的默认值。 |
| Char | System.Char | DbType.StringFixedLength - 1 char | 在没有指定类型(type) 属性时的默认值。 |
| DateTime | System.DateTime | DbType.DateTime - ignores the milliseconds | 在没有指定类型(type) 属性时的默认值。 |
| Decimal | System.Decimal | DbType.Decimal | 在没有指定类型(type) 属性时的默认值。 |
| Double | System.Double | DbType.Double | 在没有指定类型(type) 属性时的默认值。 |
| Guid | System.Guid | DbType.Guid | 在没有指定类型(type) 属性时的默认值。 |
| Int16 | System.Int16 | DbType.Int16 | 在没有指定类型(type) 属性时的默认值。 |
| Int32 | System.Int32 | DbType.Int32 | 在没有指定类型(type) 属性时的默认值。 |
| Int64 | System.Int64 | DbType.Int64 | 在没有指定类型(type) 属性时的默认值。 |
| PersistentEnum | System.Enum | 潜在类型对应的DbType | 不用在映射文件指定type="PersistentEnum".而是提供枚举的程序集全名, 让NHibernate用反射来猜测类型。枚举使用的潜在类型决定适当的DbType.。 |
| Single | System.Single | DbType.Single | 在没有指定类型(type) 属性时的默认值。 |
| Ticks | System.DateTime | DbType.Int64 | type="Ticks"必须被指定。 |
| TimeSpan | System.TimeSpan | DbType.Int64 | 在没有指定类型(type) 属性时的默认值。 |
| Timestamp | System.DateTime | DbType.DateTime - 取决于数据库支持 | type="Timestamp"必须被指定。 |
| TrueFalse | System.Boolean | DbType.AnsiStringFixedLength -一个字符,'Y' 或者'N' | type="TrueFalse"必须被指定。 |
| YesNo | System.Boolean | DbType.AnsiStringFixedLength -一个字符,'Y' 或者'N' | type="YesNo"必须被指定。 |

**3.2 应用类型映射表**

| **NHibernate Type** | **.NET Type** | **Database Type** | **Remarks** |
| --- | --- | --- | --- |
| AnsiString | System.String | DbType.AnsiString | type="AnsiString"必须被指定。 |
| CultureInfo | System.Globalization.CultureInfo | DbType.String -表明文化(culture)的5个字符 | 在没有指定类型(type) 属性时的默认值。 |
| Binary | System.Byte[] | DbType.Binary | 在没有指定类型(type) 属性时的默认值。 |
| Type | System.Type | DbType.String保存应用程序集权限定名。 | 在没有指定类型(type) 属性时的默认值。 |
| String | System.String | DbType.String | 在没有指定类型(type) 属性时的默认值。 |

**3.3 二进制类型映射表**

| **NHibernate Type** | **.NET Type** | **Database Type** | **Remarks** |
| --- | --- | --- | --- |
| StringClob | System.String | DbType.String | type="StringClob"必须被指定。 整个字段在内存里可读。 |
| BinaryBlob | System.Byte[] | DbType.Binary | type="BinaryBlob"必须被指定。 整个字段在内存里可读。 |
| Serializable | Any System.Object 必须标注可序列化标签 | DbType.Binary | type="Serializable" 应该被指定. 如果不能为属性找到NHibernate类型，这是最后可依靠的类型。 |

**一、多对一关联映射概括**

　　关联关系是实体类与实体类之间的结构关系，分别为“多对一”、“一对一”、“多对多”。然而“多对一”是怎样描述的呢？让我们参考图1.1所示：



图1.1

　　学生表“T\_Student”有两条记录，它们分别对应了班级表“T\_Class”的主键值为“1”的同一条记录。所以这就是“多对一”的使用情景。

　　让我们看一下“多对一”持久化类的结构：

[IMG_257](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

    public class Class  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
    }  
  
    public class Student  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        public virtual Class Class { get; set; }  
    }

[IMG_258](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

　　在“Student”类中有类型为“Class”的属性，这就意味着在“Student”有一个与之对应的“Class”类。

　　再让我们看一下映射文件：

[IMG_259](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

<class name="Class" table="T\_Class" lazy="true" >  
    <id name="ID" type="int" column="ClassID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
  </class>  
  
  <class name="Student" table="T\_Student" lazy="true" >  
    <id name="ID" type="int" column="StudentID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <many-to-one name="Class" column="ClassID"/>  
      
  </class>

[IMG_260](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

　　我在“Student”类的映射文件中发现了“many-to-one”节点，<many-to-one>标签是描述“多对一”的关联映射。

　　该标签的name属性是“多对一”的类名，column是外键字段名。

　　我们配置好映射文件后运行程序，生成的表结构如图1.2所示：

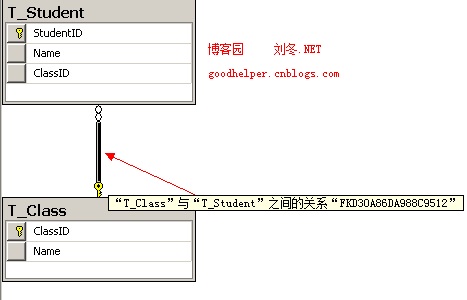


图1.2

　　发现NHibernate帮我们生成了一个外键。

**二、多对一关联映射插入和查询**

**2.1 多对一关联映射插入**

　　我们编写一个测试类，测试一下“多对一”关联映射的插入。

 　　代码如下：

[IMG_262](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

 [TestFixture]  
    public class NHibernateInit  
    {  
        private ISessionFactory sessionFactory;  
  
        [SetUp]  
        public void InitTest()  
        {  
            var cfg = new NHibernate.Cfg.Configuration().Configure("Config/hibernate.cfg.xml");  
            sessionFactory = cfg.BuildSessionFactory();  
        }  
  
        [Test]  
        public void Save()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var cls = new Class { Name = "1班" };  
  
                var liu = new Student { Name = "刘冬", Class = cls };  
                var zhang = new Student { Name = "张三", Class = cls };  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }  
    }

[IMG_263](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

　　使用NUnit运行测试类。如图2.1.1所示，发现抛出“object references an unsaved transient instance - save the transient instance before flushing. Type: Domain.Class, Entity: Domain.Class”这样的异常。

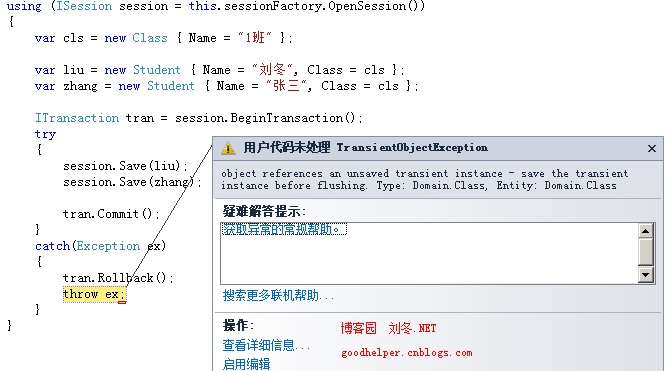


图2.1.1

　　这个异常是怎么回事呢？持久化“Student”类的实例后，“Student”类的实例变为**“持久态(Persistent)”**，而“Class”类的实例还是**临时态(Transient)**，当持久化一个持久态的实例时，若该实例引用了一个临时态的实例，就会抛出这样的异常。

　　我们修改一下代码：

[IMG_265](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

[Test]  
        public void Save()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var cls = new Class { Name = "1班" };  
  
                var liu = new Student { Name = "刘冬", Class = cls };  
                var zhang = new Student { Name = "张三", Class = cls };  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    //持久化“Class”类  
                    session.Save(cls);  
  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_266](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

 　　然后运行修改过的程序。如图2.1.2所示，运行成功。

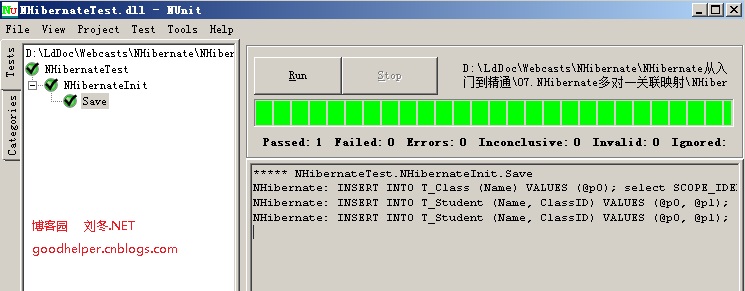


图2.1.2

　　同理，我们也可以修改<many-to-one>标签中的cascade属性为“all”,这样就能实现同样的效果。设置cascade属性后，当持久态的实例引用临时态的实例时，NHibernate就会帮我们把这个临时态的实例自动持久化到数据库中。

**2.2 多对一关联映射查询**

　　我们编写一个查询方法，当获取到“Student”时，可以通过其“Class”属性读取“Class”的信息。

[IMG_268](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

        [Test]  
        public void Select()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var student = session.CreateQuery(" from Student").List<Student>().First();  
  
                Console.WriteLine("学生名为：{0}", student.Name);  
  
                Console.WriteLine("班级名为：{0}", student.Class.Name);  
            }  
        }

[IMG_269](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/javascript:void(0);)

　　运行效果如图2.2.1所示，我们并没有再去调用查询方法，就能获取到关联的信息。



图2.2.1

**三、多对一关联映配置介绍**

　　<many-to-one

name="PropertyName" 属性名

column="column\_name" 数据库字段名

class="ClassName" 关联的类的名字

cascade="all|none|save-update|delete" 级联关系

fetch="join|select" 抓取

update="true|false"

insert="true|false"

property-ref="PropertyNameFromAssociatedClass" 指定关联类的一个属性，这个属性将会和本外键相对应

access="field|property|nosetter|ClassName"

unique="true|false" 是否生成一个唯一约束

optimistic-lock="true|false"

not-found="ignore|exception" 指定外键引用的数据不存在时如何处理

　　/>

**内容摘要**

　　单向主键关联映射

　　双向主键关联映射

　　唯一外键关联映射

 　　NHibernate的一对一关联映射有三种，单向主键关联映射、双向主键关联映射、唯一外键关联映射。

**一、单向主键关联映射**

 　　我们模拟一个现实情况：学生（Student）和家庭（Family）的关系。在中国，目前实行计划生育，一个家庭只有一个孩子，孩子上学后就成为了学生。学生和家庭的关系可以认为是一对一的。

 　　让我们看一下“一对一”的表结构,如图1.1所示：

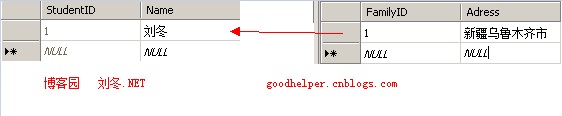


图1.1

　　让我们看一下“一对一”的实体类和映射文件：

[IMG_257](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

    public class Student  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
    }    
  
    public class Family  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Adress { get; set; }  
  
        public virtual Student Student { get; set; }  
    }

[IMG_258](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

[IMG_259](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

  <class name="Student" table="T\_Student" lazy="true" >  
  
    <id name="ID" column="StudentID" type="int">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string" length="50"/>     
      
  </class>  
  
  <class name="Family" table="T\_Family" lazy="true" >  
  
    <id name="ID" column="FamilyID" type="int">  
      <generator class="foreign">  
        <param name="property">Student</param>  
      </generator>  
    </id>  
  
    <property name="Adress" type="string" length="200"/>  
  
    <one-to-one name="Student" constrained="true"/>  
  </class>

[IMG_260](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　其中，我们设置了“Family”类的主键生成策略为：“foreign”，含义是通过外键查询到主键值。

　　param的name属性设置为property，值设置为“Student”，表示通过“Student”属性查询到主键值。

　　然后使用<one-to-one>标签来描述“一对一”关联映射，并设置constrained属性为true，来建立一个外键约束。

　　环境建立好了，生成的表结构，如图1.2所示，我们能够观察到“T\_Family”表的“FamilyID”字段既是主键又是外键。

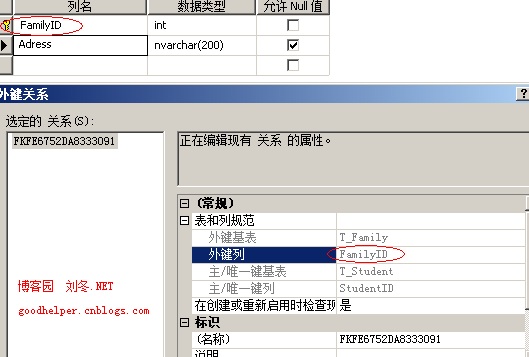


图1.2

　　我们编写单元测试类的代码测试一下插入和查询。

[IMG_262](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

    [TestFixture]  
    public class DomainTest  
    {  
        private ISessionFactory sessionFactory;  
  
        [SetUp]  
        public void InitTest()  
        {  
            var cfg = new NHibernate.Cfg.Configuration().Configure("Config/hibernate.cfg.xml");  
            sessionFactory = cfg.BuildSessionFactory();  
        }  
  
        [Test]  
        public void SaveFamilyTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var student = new Student { Name = "刘冬" };  
                var family = new Family { Adress = "新疆乌鲁木齐市", Student = student };  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    session.Save(family);  
                    tran.Commit();  
                }  
                catch (Exception ex)  
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }  
  
        [Test]  
        public void SelectFamilyTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var family = session.CreateQuery("from Family").List<Family>().First();  
  
                Console.WriteLine("家庭地址为：{0}", family.Adress);  
                Console.WriteLine("学生姓名为：{0}", family.Student.Name);  
            }  
        }  
    }

[IMG_263](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　插入的运行效果如图1.3所示，运行成功。奇怪的是，“Family”引用了一个**临时态(Transient)**的实例“Student” ，但并没有抛出异常。

　　这是因为<one-to-one>默认的cascade为all，这样NHibernate自动帮助我们把**临时态(Transient)**的实例持久化到数据库中了。

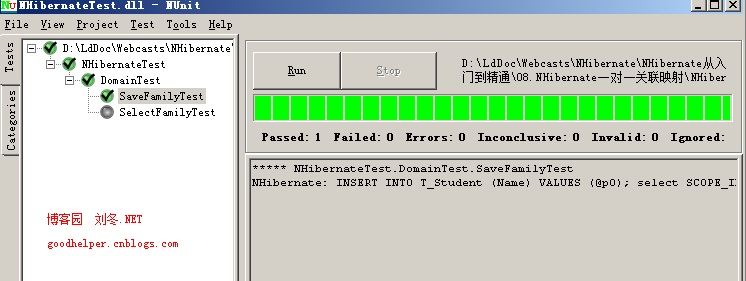


图1.3

　　查询的运行效果如图1.4所示，通过关联的属性带出了想要的信息。

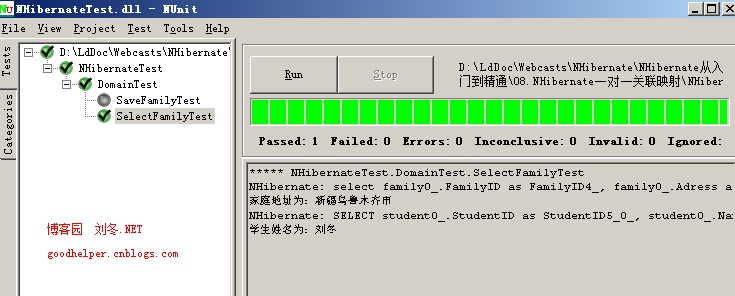


图1.4

**二、双向主键关联映射**

　　我们修改一下代码，来体现双向主键关联映射：在“Student”类中加入属性“Family”，这样“Student”中有“Family”，“Family”中也有“Student”的循环引用方式就描述了双向主键关联映射的实体类结构。

　　然后在“Student”的映射文件中加入“<one-to-one name="Family" class="Family"/>”。

[IMG_266](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

    public class Student  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        public virtual Family Family { get; set; }  
    }

[IMG_267](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

[IMG_268](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

<class name="Student" table="T\_Student" lazy="true" >  
  
    <id name="ID" column="StudentID" type="int">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string" length="50"/>  
  
    <one-to-one name="Family" class="Family"/>  
      
  </class>

[IMG_269](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　 然后编写一个查询的单元测试方法：

[IMG_270](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

        [Test]  
        public void SelecStudentTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var student = session.Get<Student>(1);  
  
                Console.WriteLine("学生姓名为：{0}", student.Name);  
                Console.WriteLine("家庭地址为：{0}", student.Family.Adress);  
            }  
        }

[IMG_271](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　运行效果如图2.1所示。奇怪的是仅生成了一条SQL语句就将两个表中的数据获取出来了。这是因为“一对一”主键关联映射默认的抓取(fetch)策略是“join”。

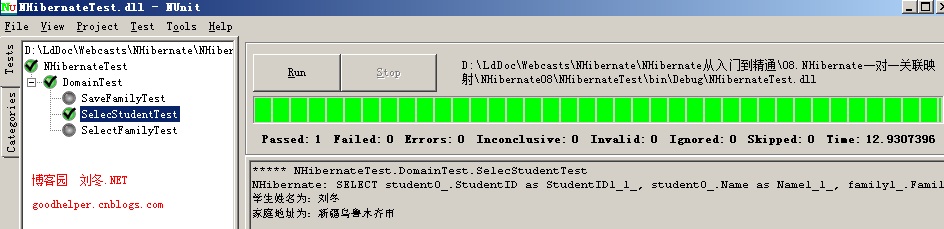


图2.1

**三、唯一外键关联映射**

　　唯一外键关联映射是非主键字段的“一对一”关联。

　　我们模拟一种“一对一”情况：一个班级对应了一个班主任老师，一个班主任老师管理一个班级。代码如下：

[IMG_273](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

    public class Class  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        /// <summary>  
        /// 班主任老师  
        /// </summary>  
        public virtual Teacher Teacher { get; set; }  
    }  
  
    public class Teacher  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        public virtual Class Class { get; set; }  
    }

[IMG_274](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

[IMG_275](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

 <class name="Class" table="T\_Class" >  
  
    <id name="ID" column="ClassID" type="int">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string" length="50"/>  
  
    <many-to-one name="Teacher" class="Teacher" column="TeacherID" unique="true"/>  
      
  </class>  
  
  
  <class name="Teacher" table="T\_Teacher">  
  
    <id name="ID" column="TeacherID" type="int">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string" length="50"/>  
  
    <one-to-one name="Class" class="Class" property-ref="Teacher"/>  
  
  </class>

[IMG_276](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　我们在“Class”类中使用<many-to-one>标签，并设置unique属性为true。然后在“Teacher”类中使用<one-to-one>标签，并设置属性“Class”的property-ref指向“Teacher”。

　　最后编写一个单元测试方法：

[IMG_277](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

[Test]  
        public void SaveTeacherTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var teacher = new Teacher { Name = "刘冬" };  
                var cls = new Class { Name = "1班", Teacher = teacher };  
                teacher.Class = cls;  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    session.Save(teacher);  
                    session.Save(cls);  
  
                    tran.Commit();  
                }  
                catch (Exception ex)  
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_278](http://www.cnblogs.com/GoodHelper/archive/2011/02/25/javascript:void(0);)

　　运行效果如图3.1所有，运行成功。

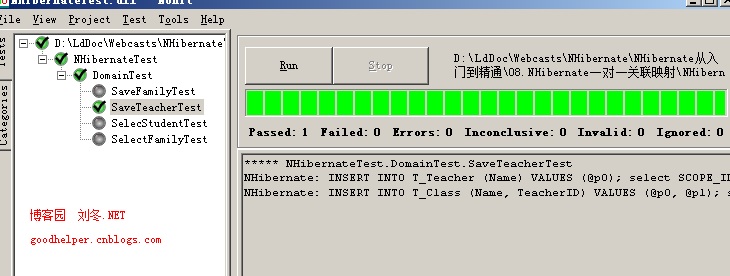


图3.1

　　然后观察生成的表结构，如图3.2所示，生成了唯一外键。

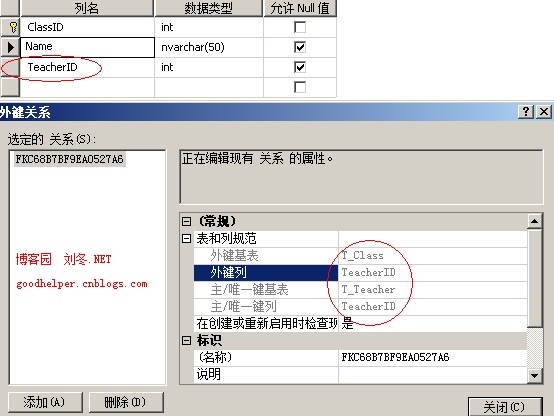


图3.2

　　但注意的是：唯一外键关联映射实际上使用的是<many-to-one>标签，所有说默认的cascade是“none”，这样必须确保在没有引用**临时态(Transient)**的实例下才能持久化数据。

**内容摘要**

　　　　单向关联映射

　　　　双向关联映射

**一、单向关联映射**

**1.1 单向关联映射的描述**

　　让我们回顾一下之前讲的《[多对一关联映射](http://www.cnblogs.com/GoodHelper/archive/2011/02/24/nhibernate07.html)》，如图1.1.1所示，其实“一对多”关联映射就是“多对一”关联映射相反的映射。



图1.1.1

　　至于“一对多”单向关联映射的代码如下：

[IMG_257](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

    public class Student  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
    }  
  
    public class Class  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        public virtual IList<Student> Students { get; set; }  
    }

[IMG_258](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　映射文件如下：

[IMG_259](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Student" table="T\_Student" lazy="true" >  
    <id name="ID" type="int" column="StudentID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
  </class>  
</hibernate-mapping>  
  
  
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Class" table="T\_Class" lazy="true" >  
    <id name="ID" type="int" column="ClassID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <bag name="Students">  
      <key column="ClassID"/>  
      <one-to-many class="Student"/>  
    </bag>  
      
  </class>  
</hibernate-mapping>

[IMG_260](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　我们看到“Class”类中，有名为“Students” 的属性，其类型是IList<Student>。在映射文件中，我们使用<bag>和<one-to-many>标签来描述“一对多”关联映射。

**1.2 单向关联映射的数据插入**

 　　单元测试类的代码如下：

[IMG_261](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

    [TestFixture]  
    public class OneToManyTest  
    {  
        private ISessionFactory sessionFactory;  
  
        public OneToManyTest()  
        {  
            log4net.Config.XmlConfigurator.Configure();  
        }  
  
        [SetUp]  
        public void Init()  
        {  
            var cfg = new NHibernate.Cfg.Configuration().Configure("Config/hibernate.cfg.xml");  
            sessionFactory = cfg.BuildSessionFactory();  
        }  
  
        [Test]  
        public void SaveTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var liu = new Student { Name = "刘冬" };  
                var zhang = new Student { Name = "张三" };  
  
                var cls = new Class { Name = "1班" };  
                cls.Students = new List<Student> { liu, zhang };  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    session.Save(cls);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }  
}

[IMG_262](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

 　　我们配置log4net输出SQL语句：

[IMG_263](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<?xml version="1.0"?>  
<configuration>  
  <configSections>  
  
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />  
  </configSections>  
  
  
  <!--log4net配置-->  
  <log4net debug="true">   
    <appender name="EventLogAppender" type="log4net.Appender.EventLogAppender">  
      <layout type="log4net.Layout.PatternLayout">  
        <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] - %m%n" />  
      </layout>  
    </appender>   
    <root>  
      <level value="ALL" />  
      <appender-ref ref="RollingLogFileAppender" />  
    </root>  
  </log4net>  
  
  
  <startup>  
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>  
  </startup>  
</configuration>

[IMG_264](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果如图1.2.1所示，先成“insert into”语句，然后生成“update”语句修改外键。

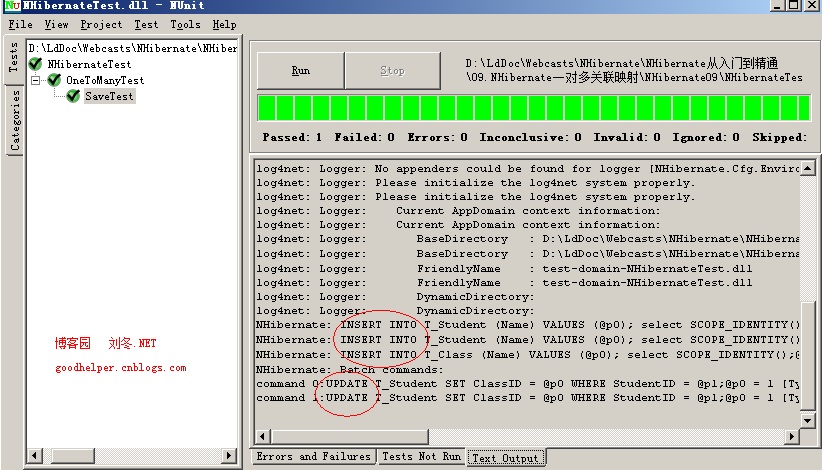


图1.2.1

　　从图1.2.1中，我们能够观察到，如果“一对多”的外键不允许空，就有可能插入不成功。

　　我们将映射文件稍作修改：

[IMG_266](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Class" table="T\_Class" lazy="true" >  
    <id name="ID" type="int" column="ClassID">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <!--设置为不可空-->  
    <bag name="Students">  
      <key column="ClassID" not-null="true"/>  
      <one-to-many class="Student"/>  
    </bag>  
  
  </class>  
</hibernate-mapping>

[IMG_267](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果如图1.2.2所示，抛出“ClassID”不允许插入NULL的异常。

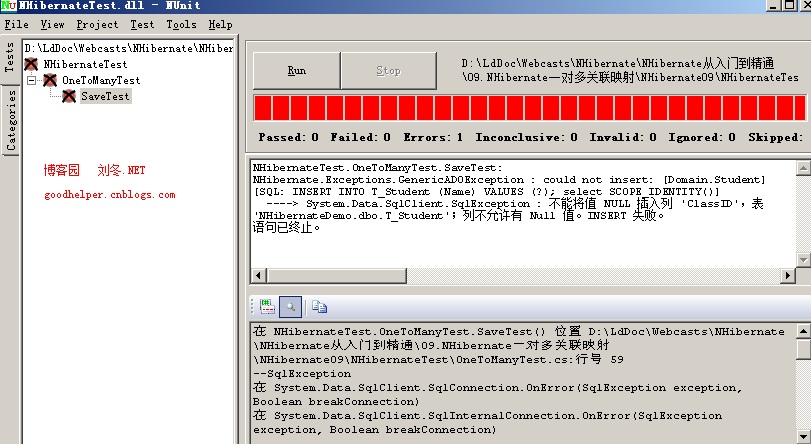


图1.2.2

**二、双向关联映射**

**2.1 双向关联映射的描述**

　　我们修改一下代码，来实现“一对多”双向关联映射：

[IMG_269](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

    public class Student  
    {  
        public virtual int? ID { get; set; }  
  
        public virtual string Name { get; set; }  
  
        public virtual Class Class { get; set; }  
    }

[IMG_270](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

[IMG_271](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Student" table="T\_Student" lazy="true" >  
    <id name="ID" type="int" column="StudentID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <many-to-one name="Class" column="ClassID" />  
      
  </class>  
</hibernate-mapping>

[IMG_272](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　“一对多”双向关联映射指的是：在“一”的这端（“Class”类这端）包含“多”的属性（“Students”）；在“多”的这端包含“一”的属性（“Class”属性）。这样两个类构成的循环引用就是双向关联映射。

**2.2 双向关联映射的数据插入**

　　插入数据的代码如下：

[IMG_273](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

        [Test]  
        public void SaveTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var liu = new Student { Name = "刘冬" };  
                var zhang = new Student { Name = "张三" };  
  
                var cls = new Class { Name = "1班" };  
                cls.Students = new List<Student> { liu, zhang };  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    session.Save(cls);  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_274](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果如图2.2.1所示，运行成功。

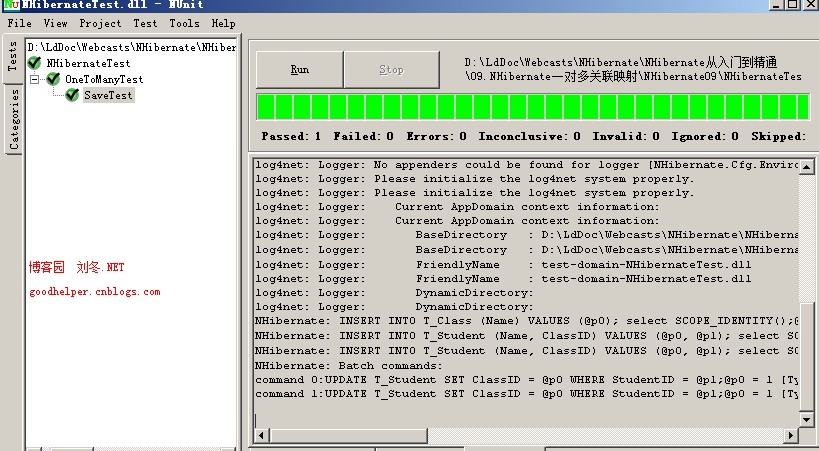


图2.2.1

　　修改“Student”的映射文件，将“Class”属性修改为不允许空：

[IMG_276](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Student" table="T\_Student" lazy="true" >  
    <id name="ID" type="int" column="StudentID">  
      <generator class="native"/>  
    </id>  
    
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <!--不允许空-->  
    <many-to-one name="Class" column="ClassID" not-null="true"/>  
      
  </class>  
</hibernate-mapping>

[IMG_277](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

 　　运行效果如图2.2.1所示，抛出“not-null property references a null or transient value”的异常。

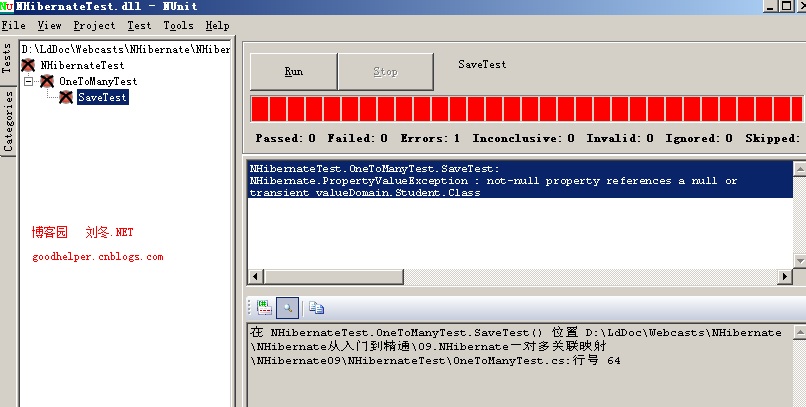


图2.2.1

　　我修改单元测试代码：

[IMG_279](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

        [Test]  
        public void SaveTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var liu = new Student { Name = "刘冬" };  
                var zhang = new Student { Name = "张三" };  
  
                var cls = new Class { Name = "1班" };  
                cls.Students = new List<Student> { liu, zhang };  
  
                liu.Class = cls;  
                zhang.Class = cls;  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    //先保存班级  
                    session.Save(cls);  
  
                    //后保存学生  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_280](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果图2.2.2所示，运行成功。

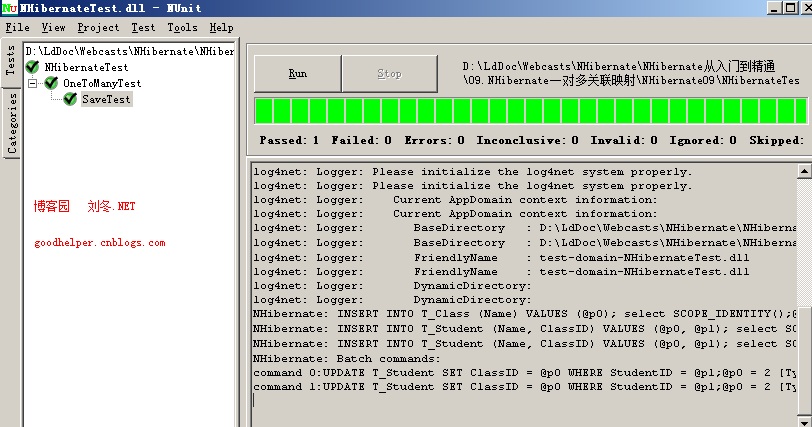


图2.2.2

　　我们修改保存“Student”和“Class”实例的先后顺序：

[IMG_282](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

        [Test]  
        public void SaveTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var liu = new Student { Name = "刘冬" };  
                var zhang = new Student { Name = "张三" };  
  
                var cls = new Class { Name = "1班" };  
                cls.Students = new List<Student> { liu, zhang };  
  
                liu.Class = cls;  
                zhang.Class = cls;  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    //先保存学生  
                    session.Save(liu);  
                    session.Save(zhang);  
  
                    //后保存班级  
                    session.Save(cls);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_283](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果图2.2.3所示，同样抛出“not-null property references a null or transient value”的异常。

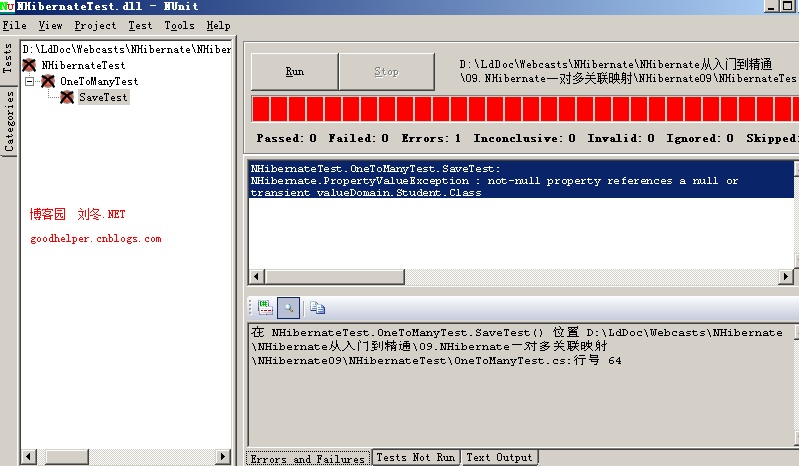


图2.2.3

　　我们修改一下“Class”的映射文件：

[IMG_285](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Domain" namespace="Domain">  
  <class name="Class" table="T\_Class" lazy="true" >  
    <id name="ID" type="int" column="ClassID">  
      <generator class="native"/>  
    </id>  
  
    <property name="Name" type="string">  
      <column name="Name" length="50"/>  
    </property>  
  
    <!--设置为不可空-->  
    <bag name="Students" inverse="true" cascade="all">  
      <key column="ClassID" not-null="true"/>  
      <one-to-many class="Student"/>  
    </bag>  
  
  </class>  
</hibernate-mapping>

[IMG_286](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　然后修改单元测试的代码：

[IMG_287](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

        [Test]  
        public void SaveTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var liu = new Student { Name = "刘冬" };  
                var zhang = new Student { Name = "张三" };  
  
                var cls = new Class { Name = "1班" };  
                cls.Students = new List<Student> { liu, zhang };  
  
                liu.Class = cls;  
                zhang.Class = cls;  
  
                ITransaction tran = session.BeginTransaction();  
                try  
                {  
                    //只保存班级  
                    session.Save(cls);  
  
                    tran.Commit();  
                }  
                catch(Exception ex)   
                {  
                    tran.Rollback();  
                    throw ex;  
                }  
            }  
        }

[IMG_288](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果如图2.2.4所示，运行成功，并且没有生成“update”语句，只生成“insert into”语句。

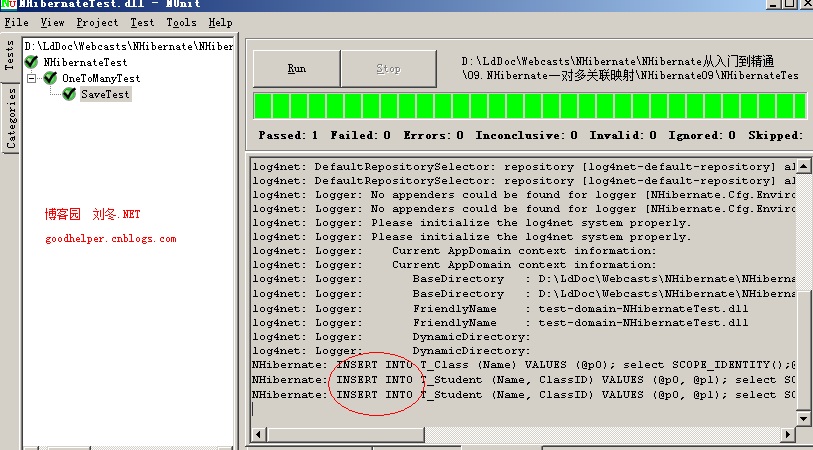


图2.2.4

　　这样，保存“一”的这端(“Class”实例)，就能够将“一”得那端（“Student”）连带保存。其中映射文件中的“inverse”的属性是反转的意思，就是将操作交给双向关联关系中的另一端。

　　我们细观察到生成的SQL语句，只生成了“insert into”语句，这样，执行效率就变的高了。

**2.3 双向关联映射的数据查询**

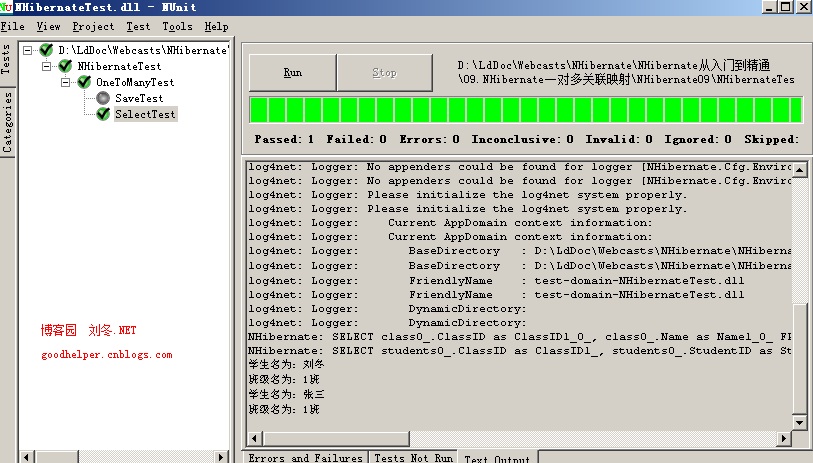
　　编写单元测试的代码：

[IMG_290](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

        [Test]  
        public void SelectTest()  
        {  
            using (ISession session = this.sessionFactory.OpenSession())  
            {  
                var cls = session.Get<Class>(1);  
  
                foreach (var item in cls.Students)  
                {  
                    Console.WriteLine("学生名为：{0}", item.Name);  
                    Console.WriteLine("班级名为：{0}", item.Class.Name);  
                }  
            }  
        }

[IMG_291](http://www.cnblogs.com/GoodHelper/archive/2011/03/03/javascript:void(0);)

　　运行效果如图2.3.1所示，比较以往使用SQL语句编程的代码后，发现调用“一对多”关联映射的集合变得如此方便。



<http://www.cnblogs.com/GoodHelper/archive/2011/02/24/nhibernate06.html>

该系列文章相当不错