FREE eBook

LEARNING nhibernate

Free unaffiliated eBook created from
Stack Overflow contributors.

#nhibernate

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: nhibernate

It is an unofficial and free nhibernate ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official nhibernate.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with nhibernate

Remarks

This section provides an overview of what nhibernate is, and why a developer might want to use it.

It should also mention any large subjects within nhibernate, and link out to the related topics. Since the Documentation for nhibernate is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Detailed instructions on getting nhibernate set up or installed.

Read Getting started with nhibernate online: https://riptutorial.com/nhibernate/topic/1663/getting-started-with-nhibernate

Chapter 2: Cascades

Syntax

- cascade="all-delete-orphan"

Remarks

Entities has associations to other objects, this may be an association to a single item (many-to-one) or an association to a collection (one-to-many, many-to-any).

At any rate, you are able to tell NHibernate to automatically traverse an entity's associations, and act according to the cascade option. For instance, adding an unsaved entity to a collection with save-update cascade will cause it to be saved along with its parent object, without any need for explicit instructions on our side.

https://ayende.com/blog/1890/nhibernate-cascades-the-different-between-all-all-delete-orphans-and-save-update

Examples

save-update

when the object is saved/updated, check the associations and save/update any object that require it (including save/update the associations in many-to-many scenario).

none

do not do any cascades, let the users handles them by themselves.

delete

when the object is deleted, delete all the objects in the association.

delete-orphan

when the object is deleted, delete all the objects in the association. In addition to that, when an object is removed from the association and not associated with another object (orphaned), also delete it.

all

when an object is save/update/delete, check the associations and save/update/delete all the objects found.

all-delete-orphan

when an object is save/update/delete, check the associations and save/update/delete all the objects found. In additional to that, when an object is removed from the association and not associated with another object (orphaned), also delete it.

Read Cascades online: https://riptutorial.com/nhibernate/topic/2754/cascades

Chapter 3: LINQ to NHibernate Queries

Remarks

The LINQ to NHibernate driver is centered on the `IQueryable<T>` interface.

Be sure to add `using NHibernate.Linq;` in order to use the NHibernate LINQ provider.

Examples

Basic query

```
IQueryable<Cat> cats = session.Query<Cat>()
    .Where(c => c.Name == "Max");
```

Read LINQ to NHibernate Queries online: https://riptutorial.com/nhibernate/topic/3544/linq-to-nhibernate-queries

Chapter 4: Mappings

Examples

A sample of Model to Map

NHibernate uses classes to map into tables or views. Creating a `Plain Old CLR Object` (POCOs, sometimes called Plain Ordinary CLR Objects) is a good practice for persistent classes. A POCO has its data accessible through the standard .NET property mechanisms, shielding the internal representation from the publicly visible interface.

```
namespace Project
{
    public class Customer
    {
        public virtual string Id { get; set; }

        public virtual string Name { get; set; }

        public virtual char Sex { get; set; }

        public virtual float Weight { get; set;}

        public virtual bool Active { get; set;}

        public virtual DateTime Birthday { get; set;}

        public Customer()
        {
        }
    }
}
```

NHibernate is not restricted in its usage of property types: all .NET types and primitives (like string, char and DateTime) can be mapped, including classes from the `System.Collections` and `System.Collections.Generics` namespaces. You can also map a relation between the entities, having properties that refer to another entity type. You can map them as values, collections of values, or associations to other entities. The property named `Id` here is a special property that represents the database identifier (primary key) of that class, which is highly recommended for entities like a Cat. NHibernate can use identifiers internally only, without having to declare them on the class, but we would lose some of the flexibility in our application architecture.

No special interface has to be implemented for persistent classes nor do we have to subclass from a special root persistent class. NHibernate also doesn't use any build time processing, such as IL manipulation; it relies solely on .NET reflection and runtime class enhancement. So, without any dependency in the POCO class on NHibernate, we can map it to a database table or view.

For the above mentioned runtime class enhancement to work, NHibernate requires that all public properties of an entity class are declared as `virtual`. The entity class must have a no-arguments constructor (`protected` or `public`) for NHibernate to create the objects.

Xml Mappings

The xml mapping uses a `hbm.xml` file which is a hibernate mapping file. It is a syntax xml file which contains the metadata required for the object/relational mapping. The metadata includes declaration of persistent classes and the mapping of properties (to columns and foreign key relationships to other entities) to database tables.

Add a file named Entity.hbm.xml into the project and set it as `embedded resource` on the properties tab. For sample, Customer.hbm.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
    namespace="Project" assembly="Project">

    <class name="Customer" table="CUSTOMERS">

        <id name="Id">
            <column name="Customer_Id" sql-type="int" not-null="true"/>
            <generator class="native" />
        </id>

        <!-- A cat has to have a name, but it shouldn' be too long. -->
        <property name="Name">
            <column name="Name" length="60" not-null="true" />
        </property>
        <property name="Sex" />
        <property name="Weight" />
        <property name="Active" />
        <property name="Birthday" />
    </class>

</hibernate-mapping>
```

The `hibernate-mapping` tag contains the namespace and assembly project information. The `class` tag contains the name of the entity on the project and the table which is been mapped. The `id` tag contains the mapping for the `primary key` where the column is specified by the `column` tag and `generator` tag define how the id is generated. The `property` tag contains information for the other columns in the database.

Fluent NHibernate Mappings

The `Fluent NHibernate` is a library to help you to map the entities using C# code instead of xml mappings. Fluent NHibernate uses the `fluent pattern` and it is based on conventions to create the mappings and it gives you the power of the visual studio tools (such as intellisense) to improve the way you map your entities.

Add the reference of the Fluent NHibernate from Nuget on your project and add a class CustomerMap.cs:

```
namespace Project.Mappings
{
    public class CustomerMap : ClassMap<Customer>
    {
```

```
        public CustomerMap()
        {
            Table("CUSTOMERS");

            Id(x => x.Id).Column("Customer_Id").GeneratedBy.Native();

            //map a property while specifying the max-length as well as setting
            //it as not nullable. Will result in the backing column having
            //these characteristics, but this will not be enforced in the model!
            Map(x => x.Name)
                .Length(16)
                .Not.Nullable();

            Map(x => x.Sex);

            Map(x => x.Weight);

            Map(x => x.Active);

            //Map a property while specifying the name of the column in the database
            Map(x => x.Birthday, "BIRTHDAY");

            //Maps a many-to-one relationship
            References(x => x.Company);

            //Maps a one-to-many relationship, while also defining which
            //column to use as key in the foreign table.
            HasMany(x => x.Orders).KeyColumn("CustomerPk");
        }
    }
}
```

The `CustomerMap` class inhirits from `ClassMap<T>` that is the base class for mapping and contains all methods necessary to create the map of your `T` entity. The method `Table` define the table name you are mapping. The `Id` method is used to map the `primery key` column. The `Map` method is used to map other columns.

Read Mappings online: https://riptutorial.com/nhibernate/topic/3543/mappings

Chapter 5: QueryOver Queries

Remarks

NHibernate 3.0 introduced the QueryOver API, which combines the use of extension methods and lambda expressions to provide a statically typesafe wrapper around the ICriteria API. The ICriteria API is NHibernate's implementation of the Query Object pattern.

Examples

Basic query

A basic `QueryOver` query is performed against an `ISession` using the `QueryOver<T>` method, where `T` is the type of a mapped entity.

```
IList<Customer> customers = session.QueryOver<Customer>()
    .Where(c => c.LastName == "Simpson")
    .List();
```

Query with join using JoinQueryOver

To join and and for instance filter on the joined table use `JoinQueryOver`.

```
IList<Customer> customers = session.QueryOver<Customer>()
    .Inner.JoinQueryOver(x => x.Organisation)
    .Where(y => y.Name == "Acme Inc")
    .List();
```

Query with join using JoinAlias

It's possible to use JoinAlias method to join several tables. It's useful when it's needed to specify some property from the joined table in the select statement:

```
Customer customerAlias = null;
Organization organizationAlias = null;

IList<Customer> customers = session.QueryOver(() => customerAlias)
    .Left.JoinAlias(x => x.Organization, () => organizationAlias)
    .Where(customer => customer.Name == "Customer Name")
    .And(() => customerAlias.Age > 18)
    .AndNot(() => organizationAlias.Name == "Forbidden Organization")
    .List();
```

Read QueryOver Queries online: https://riptutorial.com/nhibernate/topic/3545/queryover-queries

Credits

S. No	Chapters	Contributors
1	Getting started with nhibernate	Community, Felipe Oriani, Laurel
2	Cascades	dove
3	LINQ to NHibernate Queries	ngm
4	Mappings	aeliusd, Felipe Oriani, Nathan Tuggy
5	QueryOver Queries	aeliusd, ngm, Roman Koliada