EBook Gratuito

APPRENDIMENTO

nhibernate

Free unaffiliated eBook created from
Stack Overflow contributors.

#nhibernate

Sommario

Di

Capitolo 1: Iniziare con Nhibernate

Osservazioni

Questa sezione fornisce una panoramica di ciò che è nhibernate e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare eventuali soggetti di grandi dimensioni all'interno di Nibernato e collegarsi agli argomenti correlati. Poiché la documentazione di nhibernate è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Istruzioni dettagliate su come installare o installare Nhibernate.

Leggi Iniziare con Nhibernate online: https://riptutorial.com/it/nhibernate/topic/1663/iniziare-con-nhibernate

Capitolo 2: Cascades

Sintassi

- cascade = "all-delete-orphan"

Osservazioni

Le entità hanno associazioni ad altri oggetti, questa può essere un'associazione a un singolo oggetto (molti-a-uno) o un'associazione a una raccolta (uno a molti, molti a tutti).

Ad ogni modo, puoi dire a NHibernate di attraversare automaticamente le associazioni di un'entità e agire secondo l'opzione a cascata. Ad esempio, l'aggiunta di un'entità non salvata a una raccolta con save-update cascade farà sì che venga salvata insieme all'oggetto principale, senza bisogno di istruzioni esplicite dalla nostra parte.

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

Examples

save-update

quando l'oggetto viene salvato / aggiornato, controllare le associazioni e salvare / aggiornare qualsiasi oggetto che lo richieda (incluso salvare / aggiornare le associazioni nello scenario molti-a-molti).

nessuna

non fare cascate, lasciare che siano gli utenti a gestirle da sole.

Elimina

quando l'oggetto viene eliminato, elimina tutti gli oggetti nell'associazione.

delete-orphan

quando l'oggetto viene eliminato, elimina tutti gli oggetti nell'associazione. Inoltre, quando un oggetto viene rimosso dall'associazione e non associato a un altro oggetto (reso orfano), lo elimina anche.

tutti

quando un oggetto è salvato / aggiorna / cancella, controlla le associazioni e salva / aggiorna /

cancella tutti gli oggetti trovati.

all-delete-orphan

quando un oggetto è salvato / aggiorna / cancella, controlla le associazioni e salva / aggiorna / cancella tutti gli oggetti trovati. In aggiunta a ciò, quando un oggetto viene rimosso dall'associazione e non associato ad un altro oggetto (reso orfano), lo elimina anche.

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

Capitolo 3: Mapping

Examples

Un esempio di modello da mappare

NHibernate utilizza le classi per mappare in tabelle o viste. Creare un `Plain Old CLR Object` (POCOs, talvolta chiamato oggetti CLR ordinari semplici) è una buona pratica per le classi persistenti. Un POCO ha i suoi dati accessibili attraverso i meccanismi di proprietà .NET standard, proteggendo la rappresentazione interna dall'interfaccia visibile pubblicamente.

```
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 non è limitato nel suo utilizzo dei tipi di proprietà: tutti i tipi e le primitive .NET (come string, char e DateTime) possono essere mappati, incluse le classi dagli spazi `System.Collections.Generics` nomi `System.Collections` e `System.Collections.Generics` . Puoi anche mappare una relazione tra le entità, avendo proprietà che si riferiscono ad un altro tipo di entità. Puoi mapparli come valori, raccolte di valori o associazioni ad altre entità. La proprietà denominata `Id` qui è una proprietà speciale che rappresenta l'identificatore del database (chiave primaria) di quella classe, che è altamente raccomandato per entità come una Cat. NHibernate può utilizzare gli identificatori solo internamente, senza doverli dichiarare sulla classe, ma perderemmo parte della flessibilità nella nostra architettura dell'applicazione.

Nessuna interfaccia speciale deve essere implementata per le classi persistenti, né dobbiamo creare sottoclassi da una speciale classe persistente di root. NHibernate inoltre non utilizza alcuna elaborazione del tempo di costruzione, come la manipolazione di IL; si basa esclusivamente sulla riflessione .NET e sul miglioramento della classe di runtime. Quindi, senza alcuna dipendenza nella classe POCO su NHibernate, possiamo mapparlo a una tabella o a una vista del database.

Affinché il suddetto miglioramento della classe runtime funzioni, NHibernate richiede che tutte le

proprietà pubbliche di una classe di entità siano dichiarate come `virtual` . La classe di entità deve avere un costruttore no-arguments (`protected` o `public`) per NHibernate per creare gli oggetti.

Mapping Xml

La mappatura xml utilizza un file `hbm.xml` che è un file di mappatura in sospensione. È un file xml di sintassi che contiene i metadati richiesti per il mapping oggetto / relazione. I metadati includono la dichiarazione delle classi persistenti e la mappatura delle proprietà (alle colonne e alle relazioni con le chiavi esterne ad altre entità) alle tabelle del database.

Aggiungi un file denominato Entity.hbm.xml nel progetto e impostalo come `embedded resource` nella scheda delle proprietà. Per esempio, 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>
```

Il tag di `hibernate-mapping` contiene lo spazio dei nomi e le informazioni sul progetto dell'assieme. Il tag `class` contiene il nome dell'entità sul progetto e la tabella che è stata mappata. Il tag `id` contiene il mapping per la `primary key` cui la colonna è specificata dal tag `column` e il tag `generator` definisce come viene generato l'id. Il tag di `property` contiene informazioni per le altre colonne nel database.

Fluent NHibernate Mappings

The `Fluent NHibernate` è una libreria che ti aiuta a mappare le entità usando il codice C # invece dei mapping xml. Fluente NHibernate usa lo `fluent pattern` e si basa sulle convenzioni per creare le mappature e ti dà la potenza degli strumenti di studio visivo (come l'intellisense) per migliorare il modo in cui mappi le tue entità.

Aggiungi il riferimento di Fluent NHibernate da Nuget al tuo progetto e aggiungi una classe 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");
        }
    }
}
```

La classe `CustomerMap` eredita da `ClassMap<T>` che è la classe base per la mappatura e contiene tutti i metodi necessari per creare la mappa della tua entità `T` Il metodo `Table` definisce il nome della tabella che stai mappando. Il metodo `Id` viene utilizzato per mappare la colonna `primery key` . Il metodo `Map` viene utilizzato per mappare altre colonne.

Leggi Mapping online: https://riptutorial.com/it/nhibernate/topic/3543/mapping

Capitolo 4: Query LINQ a NHibernate

Osservazioni

Il driver LINQ to NHibernate è centrato sull'interfaccia `IQueryable<T>` .

Assicurati di aggiungere `using NHibernate.Linq;` per poter utilizzare il provider LINQ di NHibernate.

Examples

Query di base

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

Leggi Query LINQ a NHibernate online: https://riptutorial.com/it/nhibernate/topic/3544/query-linq-a-nhibernate

Capitolo 5: QueryOver Queries

Osservazioni

NHibernate 3.0 ha introdotto l'API QueryOver, che combina l'uso di metodi di estensione e espressioni lambda per fornire un wrapper staticamente tipicamente attorno all'ICriteria API. L'API ICriteria è l'implementazione di NHibernate del modello Oggetto query .

Examples

Query di base

Una query `QueryOver` base viene eseguita su `ISession` utilizzando il `QueryOver<T>` , dove `T` è il tipo di entità mappata.

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

Query con join usando JoinQueryOver

Per unire e per esempio il filtro sulla tabella unita usa `JoinQueryOver` .

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

Interrogarsi con join utilizzando JoinAlias

È possibile utilizzare il metodo JoinAlias per unire più tabelle. È utile quando è necessario specificare alcune proprietà dalla tabella unita nell'istruzione select:

```
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();
```

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

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Nhibernate	Community, Felipe Oriani, Laurel
2	Cascades	dove
3	Mapping	aeliusd, Felipe Oriani, Nathan Tuggy
4	Query LINQ a NHibernate	ngm
5	QueryOver Queries	aeliusd, ngm, Roman Koliada