



NHibernate Documentação de Referência

Versão: 1.0.2

Índice

Prefácio vi

1. Quickstart com o IIS e Microsoft SQL Server	1
1.1. Primeiros passos com o NHibernate	1
1.2. Primeira classe persistente 2	
1.3. Mapeamento do gato 3	
1.4. Brincar com gatos 4	
1.5. Finalmente 6	
2. Arquitetura 7	
2.1. Visão global 7	
3. ISessionFactory Configuração	10
3.1. Configuração programática	10
3.2. Obtenção de um ISessionFactory	11
3.3. Usuário desde conexão ADO.NET	11
3.4. NHibernate desde conexão ADO.NET	11
3.5. Propriedades de configuração opcional	13
3.5.1. Dialetos SQL 14	
3.5.2. Outer Join Fetching	15
3.5.3. Personalizado ICacheProvider	15
3.5.4. Substituição Query Language	16
3.6. Logging 16	
3.7. Implementação de um INamingStrategy	16
3.8. Arquivo de configuração XML	16
4. Classes persistentes 18	
4.1. Um exemplo simples POCO	18
4.1.1. Declare acessores e mutadores para campos persistentes	19
4.1.2. Implementar um construtor padrão	19
4.1.3. Fornecer um identificador de propriedade (opcional)	19
4.1.4. Preferem não seladas classes e métodos virtual (opcional)	19
4.2. Implementar a herança	20
4.3. Implementação de Equals () e GetHashCode ()	20
4.4. Callbacks do ciclo de vida 21	
4.5. Callback IValidatable 22	
5. Básicos Mapeamento O / R 23	
5.1. Declaração de mapeamento 23	
5.1.1. Namespace XML	23
5.1.2. hibernate-mapping	23
5.1.3. classe 24	
5.1.4. id 26	
5.1.4.1. gerador	26
5.1.4.2. Oi / Lo Algoritmo	27
5.1.4.3. Algoritmo Hex UUID	28
5.1.4.4. UUID Algoritmo Cordas	28
5.1.4.5. Algoritmos GUID	28
5.1.4.6. Colunas de identidade e seqüências	28
5.1.4.7. Identificadores atribuídos	29
5.1.5. composite-id 29	
5.1.6. discriminador 30	
5.1.7. versão (opcional)	30

5.1.8. timestamp (opcional)	30
5.1.9. propriedade	31
5.1.10. many-to-one	33
5.1.11. um-para-um	34
5.1.12. componente dinâmico de componentes	35
5.1.13. subclasse	36
5.1.14. joined-subclass	36
5.1.15. mapa, conjunto, lista de saco,	37
5.1.16. importação	37
5.2. Tipos NHibernate	38
5.2.1. Entidades e valores	38
5.2.2. Tipos de valores básicos	38
5.2.3. Tipos de valor personalizado	40
5.2.4. Qualquer tipo mapeamentos	41
5.3. Identificadores SQL	42
5.4. Modular arquivos de mapeamento	42
6. Mapeamento de coleção	43
6.1. Coleções persistentes	43
6.2. Mapeamento de uma coleção	44
6.3. Coleções de Valores e muitos-para-muitos	45
6.4. Um-para-muitos Associações	47
6.5. Inicialização lenta	47
6.6. Coleções classificadas	48
6.7. Usando um <idbag>	49
6.8. Associações bidirecional	50
6.9. Associações ternárias	51
6.10. Associações heterogêneos	51
6.11. Exemplos coleção	51
7. Mapeamento de Componentes	54
7.1. Objetos dependentes	54
7.2. Coleções de objetos dependentes	55
7.3. Componentes como índices de IDictionary	56
7.4. Componentes como um identificador composto	56
7.5. Componentes dinâmicos	58
8. Mapeamento de Herança	59
8.1. As três estratégias	59
8.2. Limitações	61
9. Manipulando dados persistentes	63
9.1. Criando um objeto persistente	63
9.2. Carregando um objeto	63
9.3. Consultando	64
9.3.1. Consultas escalares	65
9.3.2. A interface IQuery	66
9.3.3. Coleções de filtragem	67
9.3.4. Consultas critérios	67
9.3.5. Consultas em SQL nativo	67
9.4. Objetos de atualização	68
9.4.1. Atualização na mesma ISession	68
9.4.2. Atualizando objetos destacados	68
9.4.3. Recolocar objetos destacados	69
9.5. Exclusão de objetos persistentes	70
9.6. Resplendor	70

9.7. Encerrando uma Sessão	71
9.7.1. Flushing Sessão	71
9.7.2. Confirmar a transação de banco de dados	71
9.7.3. Fechando a ISession	71
9.8. Tratamento de exceção	72
9.9. Ciclos de vida e gráficos de objetos	72
9.10. Interceptores	73
9.11. API de metadados	75
10. Transações e Concorrência	76
10.1. Configurações, Sessões e Fábricas	76
10.2. Threads e conexões	76
10.3. Considerando a identidade do objeto	76
10.4. Controle de simultaneidade otimista	77
10.4.1. Longa sessão com versionamento automático	77
10.4.2. Muitas sessões com versionamento automático	77
10.4.3. Versão do aplicativo verificação	78
10.5. Desconexão da sessão	78
10.6. Bloqueio pessimista	79
11. HQL: O Hibernate Query Language	81
11.1. Sensibilidade caso	81
11.2. A cláusula from	81
11.3. Associações e junta-se	81
11.4. A cláusula select	82
11.5. Funções de agregação	83
11.6. Consultas polimórficas	83
11.7. A cláusula where	84
11.8. Expressões	85
11.9. A ordem pela cláusula	87
11.10. A cláusula group by	87
11.11. Subconsultas	88
11.12. Exemplos HQL	88
11.13. Dicas & Truques	90
12. Consultas critérios	92
12.1. Criação de uma instância ICriteria	92
12.2. Estreitamento do conjunto de resultados	92
12.3. Ordenar os resultados	93
12.4. Associações	93
12.5. Dinâmica buscar associação	93
12.6. Consultas de exemplo	94
13. Native Queries SQL	95
13.1. Criando um IQueryable baseada	95
13.2. Alias e referências propriedade	95
13.3. Chamado consultas SQL	95
14. Melhorar o desempenho	97
14.1. Compreensão do desempenho Coleção	97
14.1.1. Taxonomia	97
14.1.2. Listas, mapas e sets são as coleções mais eficiente para atualizar	97
14.1.3. Sacos e as listas são as coleções mais eficiente inversa	98
14.1.4. Um tiro excluir	98
14.2. Proxies para a inicialização lenta	99
14.3. Usando busca em lote	100
14.4. O cache de segundo nível	101

14.4.1. Mapeamentos de cache	102
14.4.2. Estratégia: somente leitura	102
14.4.3. Estratégia: leitura / escrita	102
14.4.4. Estratégia: não seja estrita de leitura / gravação	102
14.5. Gerenciando o cache ISession	103
14.6. Cache de Consultas 103	
15. Guia Toolset 105	
15.1. Geração de Esquema 105	
15.1.1. Personalizando o esquema	105
15.1.2. Executar a ferramenta	106
15.1.3. Propriedades 107	
15.1.4. Usando Ant 107	
15.1.5. Atualizações de esquema incremental	108
15.1.6. Usando Ant para atualizações de esquema incremental	108
15.2. Geração de Código 109	
15.2.1. O arquivo de configuração (opcional)	109
15.2.2. O atributo meta	110
15.2.3. Gerador finder básica	112
15.2.4. Velocidade baseado renderer / gerador	113
15.3. Mapeamento de geração do arquivo	113
15.3.1. Executar a ferramenta	114
16. Exemplo: Pai / Filho 116	
16.1. Uma nota sobre coleções	116
16.2. Bidirecional um-para-muitos	116
16.3. Ciclo de vida em cascata 117	
16.4. Usando Atualização em cascata ()	118
16.5. Conclusão 120	
17. Exemplo: Aplicação Weblog	121
17.1. Classes persistentes 121	
17.2. Mapeamentos Hibernate 122	
17.3. NHibernate Código 123	
18. Exemplo: Vários Mapeamentos	126
18.1. Empregador / empregado 126	
18.2. Autor / Trabalho 127	
18.3. Ordem do cliente // Produto	129
19. Melhores Práticas 132	
I. Documentação NHibernateContrib	134
Prefácio CXXXV	
20. NHibernate.Caches 136	
20.1. Como usar um cache?	136
20.2. Configuração da Cache prevalência	137
20.3. SysCache Configuração	137
21. NHibernate.Mapping.Attributes	138
21.1. Como usá-lo? 138	
21.2. Dicas 139	
21.3. Sabe questões e TODOs	140
21.4. Anotações Developer	141
22. NHibernate.Tool.hbm2net	143
23. Nullables 144	
23.1. Como usá-lo? 144	

Prefácio

Trabalhando com software orientado a objetos e um banco de dados relacional pode ser trabalhoso e demorado, em ambientes corporativos atuais. NHibernate é uma ferramenta de mapeamento objeto / relacional para .NET ambientes. O mapeamento objeto / relacional prazo (ORM) refere-se a técnica de mapeamento de uma representação de dados de um objeto modelo para um modelo de dados relacional com um esquema baseado em SQL.

NHibernate não só cuida do mapeamento de classes. NET às tabelas do banco de dados (e de .NET dados aos tipos de dados SQL), mas também fornece dados de consulta e instalações de recuperação e pode reduzir significativamente o desenvolvimento de outra forma o tempo gasto com os dados de movimentação manual de cargas em SQL e ADO.NET.

NHibernate objetivo é aliviar o desenvolvedor de 95 por cento da programação comum a persistência de dados relacionados tarefas. NHibernate pode não ser a melhor solução para aplicações centradas em dados que só use os procedimentos armazenados para implementar a lógica de negócios no banco de dados, é mais útil com os modelos orientados a objetos de domínio e de negócios lógica no .NET middle-tier. No entanto, NHibernate pode certamente ajudá-lo a remover ou encapsular fornecedor específico código SQL e vai ajudar com a tarefa comum da tradução de um conjunto de resultados representam tabulação a um gráfico de objetos.

Se você é novo para NHibernate e Mapeamento Objeto / Relacional, ou mesmo .NET Framework, siga estas passos:

1. Leia o Capítulo 1, **Quickstart com o IIS e Microsoft SQL Server** para um tutorial de 30 minutos, usando o Internet In-Serviços de formação (IIS) servidor web.
2. Leia o Capítulo 2, **Arquitetura** para entender o ambiente onde NHibernate pode ser usado.
3. Use esta documentação de referência como sua fonte primária de informação. Considere a leitura **Hibernate em Ação** ([Http://www.manning.com/bauer](http://www.manning.com/bauer)) se precisar de mais ajuda com o design do aplicativo ou se você preferir um passo a passo tutorial. Também visitar <http://nhibernate.sourceforge.net/NHibernateEg/> tutorial para NHibernate com exemplos.
4. FAQs são respondidas no site do NHibernate.
5. Demos terceiros, exemplos e tutoriais estão ligados no site NHibernate.
6. A Área da comunidade no site do NHibernate é uma boa fonte de padrões de projeto e vários inter-soluções (ASP.NET, Windows Forms).

Se você tiver dúvidas, use o fórum usuário vinculado no site do NHibernate. Nós também fornecemos uma questão JIRA rastros do sistema para relatórios de bugs e solicitações de recursos. Se você está interessado no desenvolvimento do NHibernate, juntar-se a lista de discussão de desenvolvedores. Se você estiver interessado em traduzir esta documentação em sua língua, contact-nos na lista de discussão de desenvolvedores.

Capítulo 1. Quickstart com IIS e SQL da Microsoft Servidor

1.1. Introdução ao NHibernate

Este tutorial explica a configuração do NHibernate 1.0.2 dentro de um ambiente Microsoft. As ferramentas utilizadas neste Tutorial são:

1. Microsoft Internet Information Services (IIS) - servidor web que suporte a ASP.NET.
2. Microsoft SQL Server 2000 - o servidor de banco de dados. Este tutorial usa a edição de desktop (MSDE), um livre download da Microsoft. Suporte para outros bancos de dados é apenas uma questão de mudar o SQL NHibernate dialeto e configuração do driver.
3. Microsoft Visual Studio .NET 2003 -. Ambiente de desenvolvimento.

Primeiro, temos que criar um novo projeto da Web. Usamos o nome `QuickStart`, O projeto web diretório virtual `http://localhost/QuickStart`. No projeto, adicione uma referência para `NHibernate.dll`. Visual Studio irá automaticamente copiar a biblioteca e suas dependências para o diretório de saída do projeto. Se você estiver usando um banco de dados do SQL Server, adicione uma referência ao conjunto driver para o seu projeto.

Vamos agora configurar as informações de conexão do banco de NHibernate. Para fazer isso, abra o arquivo `Web.config` automaticamente gerado para o seu projeto e adicionar elementos de configuração de acordo com a lista abaixo:

```
<? Xml version = "1.0" encoding = "utf-8"?>
<configuration>
    <!-- Adicionar este elemento -->
    <configSections>
        Seção <
            name = "hibernate-configuration"
            type = "NHibernate.Cfg.ConfigurationSectionHandler, NHibernate"
        />
    </ ConfigSections>

    <!-- Adicionar este elemento -->
    <!--> <hibernate-configuration
        <session-factory>
            <property name="dialect"> NHibernate.Dialect.MsSql2000Dialect </ property>
            <property name="connection.provider"> NHibernate.Connection.DriverConnectionProvider </ Property
            <property Servidor name="connection.connection_string"> = (local); catálogo inicial = quickstart; E
        </ Session-factory>
    </ Hibernate-configuration>

    <!-- - Deixe a seção system.web inalterada -->
    <system.web>
        ...
    </ System.web>
</ Configuration>
```

O `<configSections>` elemento contém definições de seções que seguem e manipuladores de usar para processar seus conteúdo. Nós declaramos o manipulador para a seção de configuração aqui. O `<hibernate-configuration>` seção contém a configuração propriamente dita, dizendo NHibernate que vamos usar um Microsoft SQL Server 2000 banco de dados e conectar a ele através da sequência de conexão especificado. O dialeto é um ajuste necessário, bases de dados diferem em suas interpretação do SQL "padrão". NHibernate vai cuidar das diferenças e vem com dia-

lects para grandes bases de dados open source e comerciais.

Um `ISessionFactory` é o conceito do NHibernate de um armazenamento de dados único, vários bancos de dados podem ser usados através da criação de múltiplos arquivos de configuração XML e criando múltiplas `Configuração` e `ISessionFactory` objetos em seu aplicação.

O último elemento da `<hibernate-configuration>` seção declara `QuickStart` como o nome de uma montagem contendo declarações de classe e arquivos de mapeamento. Os arquivos de mapeamento contém os metadados para o mapeamento do POCO classe para uma tabela do banco de dados (ou várias tabelas). Vamos voltar para arquivos de mapeamento em breve. Vamos escrever o POCO primeira classe e em seguida, declarar os metadados de mapeamento para ele.

1.2. Primeira classe persistente

NHibernate funciona melhor com o Plain Old CLR Objects (POCOs, às vezes chamado Plain Ordinary CLR Objects) modelo de programação para as classes persistentes. A POCO tem seus dados acessíveis através do padrão .NET mecanismos de propriedade, protegendo a representação interna da interface visível publicamente:

```
using System;

namespace QuickStart
{
    Cat public class
    {
        private string
        private string      id;
        private char       nome;
        private float      sexo;
        peso;

        Cat pública ()
        {
        }

        Id cadeia pública
        {
            get {return; }
            set {id = valor; }
        }

        Nome cadeia pública
        {
            get {return nome; }
            set {nome = valor; }
        }

        Sexo de char pública
        {
            get {sexo return; }
            {sexo = valor;} set
        }

        Peso flutuar pública
        {
            get {peso return; }
            {peso = valor;} set
        }
    }
}
```

NHibernate não está restrito em seu uso de tipos de propriedade, todos .NET e primitivos (como `corda,caractere` e `DateTIme`) Podem ser mapeados, incluindo aulas a partir do `System.Collections` namespace. Você pode mapeá-los como valores, as coleções de valores, ou associações a outras entidades. O `ID` é uma propriedade especial que representa o identificador de banco de dados (chave primária) da classe, é altamente recomendado para entidades como um `Gato`. NHibernate pode

usar identificadores apenas internamente, sem ter que declará-los na classe, mas perderíamos alguns dos flexibilidade em nossa arquitetura de aplicação.

Nenhuma interface especial tem de ser implementados para classes persistentes nem temos a subclasse de uma raiz especial classe persistente. NHibernate também não usa qualquer processamento tempo de construção, tais como manipulação de IL, se baseia unicamente em .NET reflexão e valorização da classe de tempo de execução (por meio de Castle.DynamicProxy biblioteca). Assim, sem qualquer dependência na classe POCO sobre NHibernate, podemos mapeá-lo para uma tabela do banco de dados.

1.3. Mapeamento do gato

O : `Cat.hbm.xml` arquivo de mapeamento contém os metadados necessários para o mapeamento objeto / relacional. Os metadados inclui declaração de classes persistentes eo mapeamento de propriedades (para colunas e chave estrangeira relacionamentos para outras entidades) para tabelas de banco de dados.

```
<? Xml version = "1.0" encoding = "utf-8"?>
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
    namespace = "QuickStart" assembly = "QuickStart">

    <class name="Cat" table="Cat">

        <! - Um personagem hex 32 é a nossa chave substituta. É automaticamente
            gerado pelo NHibernate com o padrão UUID. ->
        <id name="Id">
            <column name="CatId" sql-type="char(32)" not-null="true"/>
            <generator class="uuid.hex" />
        </ Id>

        <! - Um gato tem que ter um nome, mas devia ser muito longo. ->
        <property name="name">
            <column name="nome" length="16" not-null="true" />
        </ Property>
        <property name="Sex" />
        <property name="Weight" />
    </ Class>

</ Hibernate-mapping>
```

Cada classe persistente deve ter um atributo identificador (na verdade, apenas as classes que representam entidades, não dependentes objetos de valor, que são mapeadas como componentes de uma entidade). Esta propriedade é usada para distinguir persistente objetos: Dois gatos são iguais se `catA.Id.Equals(catB.Id)`. É verdade, este conceito é chamado **identidade banco de dados**. NHibernate vem com vários geradores identificador para diferentes cenários (incluindo geradores nativa para sequências de banco de dados, tabelas oi / lo identificador, e identificadores de aplicação atribuído). Nós usamos o gerador de UUID

(Recomendada apenas para testes, como inteiro chaves substitutas gerados pelo banco de dados deve ser preferido) e também especificar a coluna `Catid` da tabela `Gato` para o valor do identificador NHibernate gerados (como uma chave primária da tabela).

Todas as outras propriedades de `Gato` são mapeados para a mesma tabela. No caso da `Nome` propriedade, mapeamos-lo com um declaração explícita coluna banco de dados. Isto é especialmente útil quando o esquema de banco de dados é automaticamente gerado (como instruções SQL DDL) a partir da declaração de mapeamento com NHibernate `SchemaExport` ferramenta. Todos os outros propriedades são mapeados usando as configurações padrão do NHibernate, que é o que você precisa na maioria das vezes. A tabela `Gato` no banco de dados parecido com este:

Coluna	Tipo	Modificadores
Catid	char (32)	not null chave primária
Nome	nvarchar (16)	not null
Sexo	nchar (1)	
Peso		reais

Agora você deve criar o banco de dados e esta tabela manualmente, e depois leia o Capítulo 15, Guia Toolset se você quer automatizar este passo com a ferramenta SchemaExport. Esta ferramenta pode criar uma DDL SQL completo, incluindo tabela definição, restrições de coluna tipo personalizado, restrições de unicidade e índices. Se você estiver usando SQL Server, você também deve se certificar do ASP.NET usuário tem permissões para usar o banco de dados.

1.4. Brincar com gatos

Agora estamos prontos para começar do NHibernate `ISession`. É o persistência gerente interface, a usamos para armazenar e recuperar `Gatos` de e para o banco de dados. Mas primeiro, temos de obter uma `ISession` (NHibernate unidade de trabalho) a partir do

`ISessionFactory`:

```
ISessionFactory sessionFactory =
    . novo Configuration () Configurar () BuildSessionFactory () .;
```

Um `ISessionFactory` é responsável por um banco de dados e só pode usar um arquivo de configuração XML (`Web.config` ou `hibernate.cfg.xml`). Você pode definir outras propriedades (e até mesmo alterar os metadados de mapeamento) por acessando o Configuração antes você constrói a `ISessionFactory` (Que é imutável). Onde é que vamos criar a `ISessionFactory` e como podemos acessá-lo em nossa aplicação?

Um `ISessionFactory` normalmente só é construída uma vez, por exemplo, no interior de inicialização `Application_Start` manipulador de eventos. Este também significa que você não deve mantê-lo em uma variável de instância em suas páginas ASP.NET, mas em outro local. Além disso, precisamos de algum tipo de `Singleton`, para que possamos acessar a `ISessionFactory` facilidade na aplicação código. A abordagem mostrado a seguir resolve dois problemas: o acesso à configuração e fácil a uma `ISessionFactory`.

Nós implementamos um `NHibernateHelper` classe auxiliar:

```
utiliza System;
utiliza System.Web;
utiliza NHibernate;
utiliza NHibernate.Cfg;

namespace QuickStart
{
    public sealed class NHibernateHelper
    {
        privada const string CurrentSessionKey = "nhibernate.current_session";
        private static readonly ISessionFactory sessionFactory;

        estatica NHibernateHelper ()
        {
            . sessionFactory = new Configuration () Configurar () BuildSessionFactory () .;

        }

        getCurrentSession ISession public static ()
        {
            Contexto HttpContext HttpContext.Current =;
            ISession currentSession = Context.Items [CurrentSessionKey] como ISession;

            if (currentSession == null)
            {
                currentSession sessionFactory.openSession = ();
                Context.Items [CurrentSessionKey] = currentSession;
            }

            retorno currentSession;
        }

        CloseSession public static void ()
        {
            Contexto HttpContext HttpContext.Current =;
            ISession currentSession = Context.Items [CurrentSessionKey] como ISession;
```

```

        if (currentSession == null)
        {
            // Não sessão atual
            retorno;
        }

        currentSession.Close ();
        context.Items.Remove (CurrentSessionKey);
    }

    public static void CloseSessionFactory ()
    {
        if (sessionFactory != null)
        {
            sessionFactory.Close ();
        }
    }

}
}

```

Esta classe não apenas cuidar da `ISessionFactory` com seu atributo estático, mas também tem o código para lembrar o `ISession` para a solicitação HTTP atual.

Um `ISessionFactory` é `threadsafe`, muitos fios pode acessá-lo simultaneamente e solicitar `ISessions`. Um `ISession` é um objeto não-`threadsafe` que representa uma unidade-de-obra única com o banco de dados. `ISessions` são abertos por um `ISessionFactory` e são fechados quando todo o trabalho é concluído:

```

Sessão ISession NHibernateHelper.GetCurrentSession = ();

ITransaction tx = session.BeginTransaction ();

Cat Cat princesa = new ();
princesa.Name = "princesa";
princesa.Sex = 'F';
princesa.Weight = 7.4f;

Session.save (princesa);
tx.Commit ();

NHibernateHelper.CloseSession ();

```

Em um `ISession`, Toda operação de banco de dados ocorre dentro de uma transação que isola as operações de banco de dados (mesmo operações somente leitura). Nós usamos NHibernate `ITransaction` API para abstrair da transação subjacente estratégia (no nosso caso, as transações ADO.NET). Por favor, note que o exemplo acima não lidar com qualquer exceções.

Observe também que você pode chamar `NHibernateHelper.GetCurrentSession ()`; quantas vezes você quiser, você terá alternativas obter o atual `ISession` desta solicitação HTTP. Você tem que certificar-se o `ISession` é fechada após o seu unidade de trabalho completa-, seja em `Application_EndRequest` manipulador de eventos em sua classe de aplicação ou em um `HttpModule` antes da resposta HTTP é enviada. O bom efeito colateral deste último é a inicialização lenta fácil: a `ISession` ainda está aberta quando a visão é processada, de modo NHibernate pode carregar objetos unitialized enquanto você navega o gráfico.

NHibernate tem vários métodos que podem ser usadas para recuperar objetos do banco de dados. A forma mais flexível está usando o Hibernate Query Language (HQL), que é um fácil de aprender e poderosa orientada a objetos extensão para SQL:

```

ITransaction tx = session.BeginTransaction ();

Query = IQuery session.CreateQuery ("c selecionar Cat como c = onde c.Sex: sexo");
query.SetCharacter ("sexo", 'F');

```

```
foreach (cat gato em query.Enumerable ())
{
    Console.Out.WriteLine ("Cat Feminino:" + cat.Name);
}

tx.Commit ();
```

NHibernate também oferece uma orientada a objeto **consulta por critérios API** que podem ser usados para formular consultas tipo seguro.

NHibernate, naturalmente, utiliza `IDbCommands` e parâmetro de ligação para toda a comunicação com o banco de dados `SQL`. Você também pode usar o recurso do NHibernate consulta direta `SQL` ou obter uma conexão `ADO.NET` simples a partir de um `ISES-Comissão` em casos raros.

1.5. Finalmente

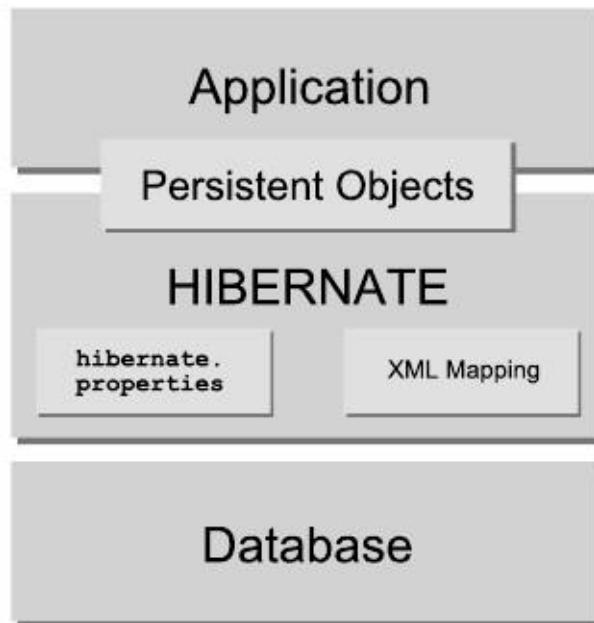
Nós apenas arranhamos a superfície do NHibernate neste pequeno tutorial. Por favor, note que não inclui qualquer `ASP.NET` código específico em nossos exemplos. Você tem que criar uma página `ASP.NET` você mesmo e insira o NHibernate código como você vê o ajuste.

Tenha em mente que NHibernate, como uma camada de acesso a dados, é totalmente integrado em sua aplicação. Normalmente, todos os outras camadas dependem do mecanismo de persistência. Certifique-se de compreender as implicações deste projeto.

Capítulo 2. Arquitetura

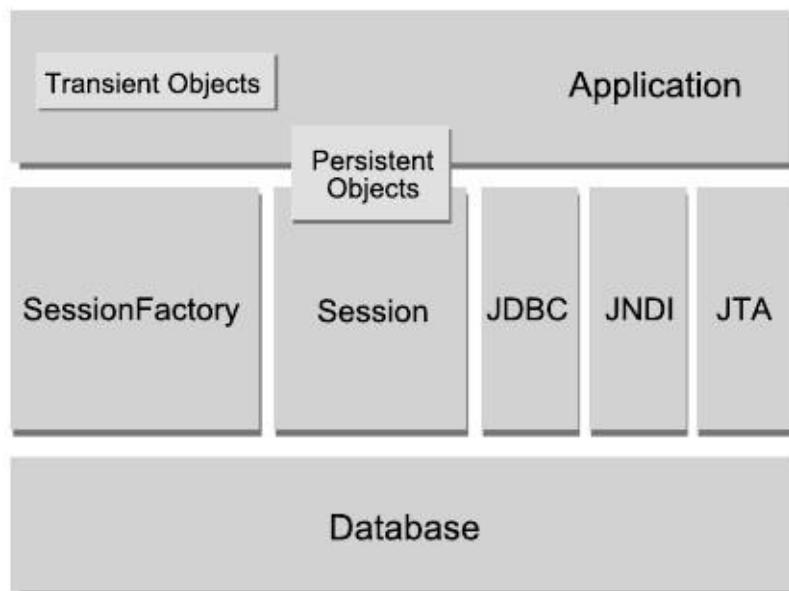
2.1. Visão global

A (muito) visão de alto nível da arquitetura do NHibernate:



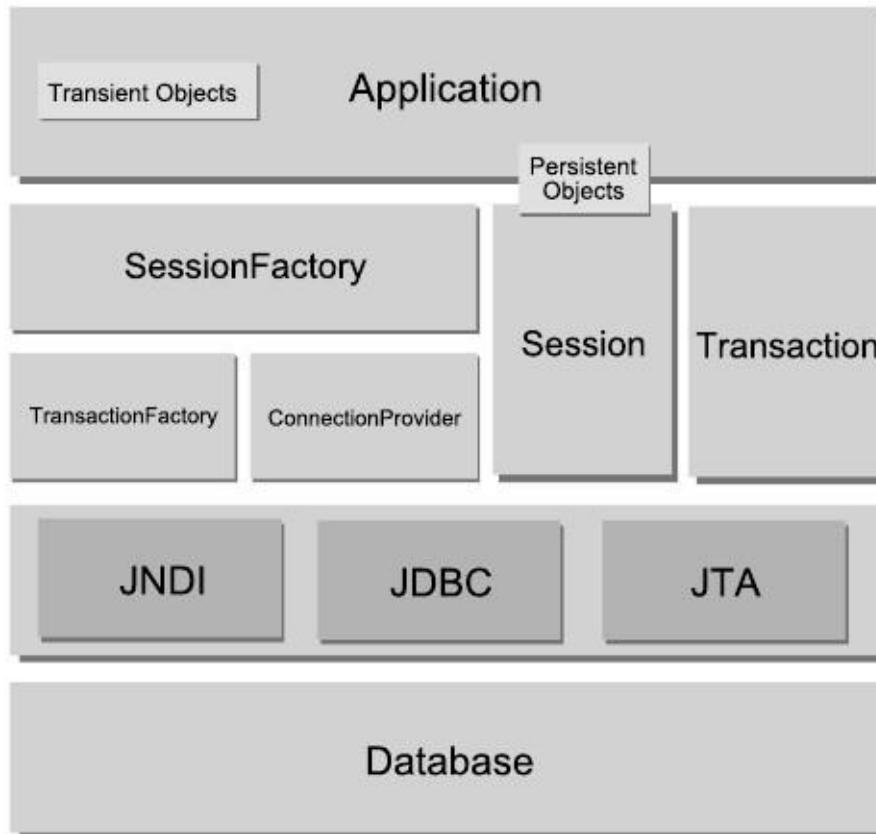
Este diagrama mostra NHibernate usando o banco de dados e dados de configuração para fornecer serviços de persistência (e objetos persistentes) para o aplicativo.

Nós gostaríamos de mostrar uma visão mais detalhada da arquitetura em tempo de execução. Infelizmente, NHibernate é flexível e suporta várias abordagens. Vamos mostrar os dois extremos. O "lite" a arquitetura tem a aplicação fornecer suas próprias conexões ADO.NET e gerenciar suas próprias transações. Esta abordagem utiliza um subconjunto mínimo das APIs do NHibernate:



O "full creme" arquitetura abstrai a aplicação da base ADO.NET APIs e deixa

NHibernate cuidar dos detalhes.



Herem algumas definições dos objetos nos diagramas:

ISessionFactory (`NHibernate.ISessionFactory`)

A threadsafe cache (imutável) de mapeamentos compilados para um único banco de dados. Uma fábrica para `ISession` e um cliente da `IConnectionProvider`. Pode realizar um cache (segundo nível) opcional de dados que é reutilizável entre as transações, a um processo-ou cluster-level.

ISession (`NHibernate.ISession`)

Um single-threaded, objeto de curta duração que representa uma conversa entre a aplicação e a persistente loja. Envolve uma conexão ADO.NET. Fábrica para `ITransaction`. Mantém um cache obrigatório (primeiro nível) de objetos persistentes, usado quando se navega o objeto gráfico ou à procura de objetos pelo identificador.

Objetos persistentes e Coleções

De curta duração, único objetos threaded contendo estado persistente e função de negócios. Estes podem ser Ordinary POCOs, a única coisa especial sobre eles é que eles são atualmente associado (exatamente um) `ISession`. Assim que a sessão é fechada, eles são separados e livres para uso em qualquer camada de aplicação (por exemplo, diretamente como objetos de transferência de dados de e para a apresentação).

Objetos transientes e Coleções

Instâncias de classes persistentes que não estão associados a um `ISession`. Eles podem ter sido instantiated pela aplicação e (ainda) não persistiu ou podem ter sido instanciado por uma fechada `ISession`.

ITransaction (`NHibernate.ITransaction`)

(Opcional) Um single-threaded, objeto de curta duração utilizado pela aplicação para especificar unidades atômicas de trabalho. Resumos de aplicação da transação ADO.NET subjacente. Um `ISession` pode abranger várias `ITransactions` em alguns casos.

IConnectionProvider (`NHibernate.Connection.IConnectionProvider`)

(Opcional) Uma fábrica para conexões ADO.NET e comandos. Resumos de aplicação do concreto vendedor implementações específicas de `IDbConnection` e `IDbCommand`. Não expostos à aplicação, mas pode ser prorrogado / implementado pelo desenvolvedor.

Idriver (`NHibernate.Driver.IDriver`)

(Opcional) Uma interface encapsular diferenças entre os prestadores de ADO.NET, tais como nomes de parâmetro convenções e recursos suportados ADO.NET.

ITransactionFactory (`NHibernate.Transaction.ITransactionFactory`)

(Opcional) Uma fábrica para `ITransaction` instâncias. Não expostos à aplicação, mas pode ser prorrogado / implementado pelo desenvolvedor.

Dado um "lite" arquitetura, a aplicação ignora a `ITransaction/ITransactionFactory` e / ou `IConnectionProvider` APIs para falar com ADO.NET diretamente.

Capítulo 3. ISessionFactory Configuração

Porque NHibernate é projetado para operar em vários ambientes diferentes, há um grande número de configuração. Felizmente, a maioria tem valores default lógicos e o NHibernate é distribuído com uma exemplo `App.config` arquivo (encontrado em `src \ NHibernate.Test`), Que mostra as várias opções. Você geralmente só tem para colocar esse arquivo no seu projeto e personalizá-lo.

3.1. Configuração programática

Uma instância de `NHibernate.Cfg.Configuration` representa um conjunto inteiro de mapeamentos de um aplicativo do. NET tipos de um banco de dados SQL. O `Configuração` é usado para construir um (imutáveis) `ISessionFactory`. Os mapeamentos são compilados a partir de vários arquivos de mapeamento XML.

Você pode obter uma `Configuração` exemplo, instanciando-lo diretamente. Heres um exemplo de criação de uma data-loja de mapeamentos definidos em dois arquivos de configuração XML:

```
Cfg configuração Configuration = new ()  
    . AddFile ("Item.hbm.xml")  
    . AddFile ("Bid.hbm.xml");
```

Uma alternativa forma (às vezes melhor) é deixar NHibernate carregar um arquivo de mapeamento de um recurso incorporado:

```
Cfg configuração Configuration = new ()  
    . AddClass (typeof (NHibernate.Auction.Item))  
    . AddClass (typeof (NHibernate.Auction.Bid));
```

Então NHibernate irá procurar por arquivos de mapeamento chamado `NHibernate.Auction.Item.hbm.xml` e `NHibernate.Auction.Bid.hbm.xml` incorporados como recursos na assembly que os tipos estão contidos dentro. Este apagam elimina qualquer nome de arquivo codificado.

Outra alternativa (provavelmente o melhor) maneira é deixar de carga NHibernate todos os arquivos contidos em um mapeamento Assembly:

```
Cfg configuração Configuration = new ()  
    . AddAssembly ("NHibernate.Auction");
```

Então NHibernate irá procurar através do conjunto de todos os recursos que terminam com `. Hbm.xml`. Esta abordagem elimina qualquer nome de arquivo codificado e garante os arquivos de mapeamento na montagem são adicionados.

Se uma ferramenta como o Visual Studio. NET ou NAnt é usado para construir a montagem, certifique-se que o `. Hbm.xml` arquivos
são compilados no assembly como Recursos Incorporados.

A `Configuração` também especifica várias propriedades opcionais:

```
Adereços IDictionary Hashtable = new ();  
...  
Cfg configuração Configuration = new ()  
    . AddClass (typeof (NHibernate.Auction.Item))  
    . AddClass (typeof (NHibernate.Auction.Bid))  
    . SetProperties (props);
```

A `Configuração` é concebida como um objeto de configuração de tempo, para ser descartada uma vez que um `ISessionFactory` é construído.

3.2. Obtenção de um ISessionFactory

Quando todos os mapeamentos têm sido analisados pela `Configuração`, O aplicativo deve obter uma factory para `ISession` instâncias. Esta fábrica é destinada a ser partilhada por todas as threads da aplicação:

```
Sessões ISessionFactory cfg.BuildSessionFactory = () ;
```

No entanto, NHibernate não permite sua aplicação instanciar mais de um `ISessionFactory`. Esse efeito é útil útil se você estiver usando mais de um banco de dados.

3.3. Usuário desde conexão ADO.NET

Um `ISessionFactory` pode abrir uma `ISession` em uma conexão fornecido pelo usuário ADO.NET. Esta escolha de design libera a aplicação para obter conexões ADO.NET onde quer:

```
IDbConnection conn myApp.GetOpenConnection = () ;
Sessão ISession sessions.OpenSession = (conn) ;

/ / Fazer algum trabalho de acesso a
dados
```

O pedido deve ser cuidado para não abrir dois concorrentes `ISessions` na conexão ADO.NET mesmo!

3.4. NHibernate desde conexão ADO.NET

Alternativamente, você pode ter o `ISessionFactory` conexões abertas para você. O `ISessionFactory` deve ser fornecido com as propriedades de conexão ADO.NET em uma das seguintes formas:

1. Passar uma instância de `IDictionary` mapeamento de nomes de propriedades para valores de propriedade para `Connection.SetProperties ()`.
2. Adicione as propriedades para uma seção de configuração no arquivo de configuração do aplicativo. A seção deve ser nomeado `nhibernate` e seu manipulador definido para `System.Configuration.NameValueSectionHandler`. Incluir `<property>` elementos em uma seção de configuração no arquivo de configuração do aplicativo. A seção
3. `shouldbenamed="nhibernate-configuration"` and `itshandlerset="NHibernate.Cfg.ConfigurationSectionHandler`. O namespace XML da seção deve ser definido como `urn: nhibernate-configuration-2.0`. Incluir `<property>` elementos em `nhibernate.cfg.xml` (Discutido mais tarde).
- 4.

Se você tomar essa abordagem, a abertura de uma `ISession` é tão simples como:

```
Sessão ISession sessions.OpenSession = () / / Abre uma nova sessão
/ / Fazer algum trabalho de acesso a dados, uma conexão ADO.NET será usado na procura
```

Todos os nomes de propriedade NHibernate e semânticas são definidas na classe `NHibernate.Cfg.Environment`. Vamos agora descrever as configurações mais importantes para a configuração de conexão ADO.NET.

NHibernate irá obter (e piscina) conexões usando um provedor de dados ADO.NET se você definir o seguinte propriedades:

Tabela 3.1. NHibernate ADO.NET Propriedades

Nome da propriedade	Propósito
hibernate.connection.provider_class	<p>O tipo de um costume <code>IConnectionProvider</code>.</p> <p>por exemplo. <code>full.classname.of.ConnectionProvider</code> se o ProviderisbuiltintoNHibernate, ou <code>full.classname.of.ConnectionProvider</code>, as- sestiver usando uma implementação do <code>IConnection-Provider</code> não incluídos no NHibernate.</p>
hibernate.connection.driver_class	<p>O tipo de um costume <code>IDriver</code>, Se usar <code>DriverCon-nectionProvider</code>.</p> <p><code>full.classname.of.Driver</code> se o driver é construído em NHibernate, ou <code>full.classname.of.Driver</code>, as- assembléia se estiver usando não uma implementação de <code>IDriver</code> in- concluíram em NHibernate.</p> <p>Isso geralmente não é necessário, na maioria das vezes o <code>oi-bernate.dialect</code> vai cuidar da definição do <code>IDriver</code> usando um padrão razoável. Veja a documentação da API do <code>IDialect</code> específicas para os padrões.</p>
hibernate.connection.connection_string	Seqüência de conexão para usar para obter a conexão.
hibernate.connection.isolation	<p>Definir o nível de isolamento da transação ADO.NET. Verificar <code>System.Data.IsolationLevel</code> para valores significativos e documentação do banco de dados para assegurar que o nível é suportada.</p> <p>por exemplo. <code>Caos</code>, <code>ReadCommitted</code>, <code>ReadUncommitted</code>, <code>Re-peatableRead</code>, <code>Serializable</code>, não especificado</p>

Este é um exemplo de como especificar as propriedades de conexão do banco de dados dentro de um `web.config`:

```

<? Xml version = "1.0" encoding = "utf-8"?>
<configuration>
    <configSections>
        <Nome da seção = "nhibernate" type = "System.Configuration.NameValueSectionHandler Sistema,>
            Version = 1.0.5000.0, Culture = neutral, PublicKeyToken = b77a5c561934e089 "/>
    </ConfigSections>

    <nhibernate>
        <Adicionar
            key = "hibernate.connection.provider"
            value = "NHibernate.Connection.DriverConnectionProvider"
        />
        <Adicionar
            key = "hibernate.dialect"
            value = "NHibernate.Dialect.MsSql2000Dialect"
        />
        <Adicionar
            key = "hibernate.connection.driver_class"
            value = "NHibernate.Driver.SqlClientDriver"
        />
        <Adicionar
            key = "hibernate.connection.connection_string"
            value = "Server = 127.0.0.1; Initial Catalog = thedatabase; Integrated Security = SSPI"
        />
        <Adicionar
            key = "hibernate.connection.isolation"
            value = "ReadCommitted"
        />
    </nhibernate>

```

```

    />

</ Nhibernate>

<-! Configuração outro aplicativo específico segue ->
</ Configuration>

```

NHibernate depende da implementação do provedor de dados ADO.NET da pool de conexões.

Você pode definir sua própria estratégia de plugin para obter conexões ADO.NET implementando a interface `NHibernate.Connection.IConnectionProvider`. Você pode escolher uma implementação customizada através da criação `hibernate.connection.provider_class`.

3.5. Propriedades opcionais de configuração

Há uma série de outras propriedades que controlam o comportamento do NHibernate em tempo de execução. Todos são opcionais e têm valores padrão razoável.

Em nível de sistema propriedades só podem ser definidas manualmente, definindo propriedades estáticas da `NHibernate.Cfg.Environment` classe ou ser definido no `<nhibernate>` seção do arquivo de configuração do aplicativo. Essas propriedades não podem ser definida usando `Configuration.setProperty` ou ser definido no `<hibernate-configuration>` seção do arquivo de configuração plicatura.

Quadro 3.2. Configuração Propriedades NHibernate

Nome da propriedade	Propósito
<code>hibernate.dialect</code>	A classe de um NHibernate Dialeto - Permite certas características dependente de plataforma. por exemplo. <code>full.classname.of.Dialect, montagem</code>
<code>hibernate.default_schema</code>	Qualificar nomes das tabelas não qualificado com o dado schema / tablespace no SQL gerado. por exemplo. <code>SCHEMA_NAME</code>
<code>hibernate.use_outer_join</code>	Permite junção exterior busca. Deprecado, uso <code>max_fetch_depth</code> . por exemplo. <code>verdadeiro falso</code>
<code>hibernate.max_fetch_depth</code>	Definir uma "profundidade" máxima para o outer join fetch árvore para single-ended associações (um-para-um, muitos-to-one). A opção desativa junção externa de busca. por exemplo. valores recomendados entre 0 e 3
<code>hibernate.use_reflection_optimizer</code>	Permite o uso de uma classe de tempo de execução gerado para definir ou obter propriedades de uma entidade ou componente em vez de usar runtime reflexão (nível de sistema de propriedade). O uso do otimizador de reflexão inflige um determinado custo de inicialização sobre a aplicação, mas deve levar a um melhor desempenho a longo prazo. Você não pode definir essa propriedade em <code>hibernate.cfg.xml</code> ou <code><hibernate-configuration></code>

Nome da propriedade	Propósito
	<p>seção do arquivo de configuração do aplicativo.</p> <p>por exemplo.</p>
<code>hibernate.cache.provider_class</code>	<p><code>verdadeiro falso</code></p> <p>O nome da classe de um costume <code>ICacheProvider</code>.</p> <p>por exemplo. <code>classname.of.CacheProvider, montagem</code></p>
<code>hibernate.cache.use_minimal_puts</code>	<p>Otimizar a operação de segundo nível de cache para minimizar escritas, ao custo de leituras mais freqüentes (útil para caches em cluster).</p> <p>por exemplo.</p>
<code>hibernate.cache.use_query_cache</code>	<p><code>verdadeiro falso</code></p> <p>Permitir que o cache de consultas, consultas individuais ainda tem a ser definido em cache.</p> <p>por exemplo.</p>
<code>hibernate.cache.query_cache_factory</code>	<p><code>verdadeiro falso</code></p> <p>O nome da classe de um costume <code>IQueryCacheFactory</code> em interface, defaultstothebuilt-instandardQueryCacheFactory.</p> <p>por exemplo. <code>classname.of.QueryCacheFactory, montagem</code></p>
<code>hibernate.cache.region_prefix</code>	<p>Um prefixo para usar nomes de segundo nível região cache.</p> <p>por exemplo.</p>
<code>hibernate.query.substitutions</code>	<p>Mapeamento de símbolos em consultas SQL para NHibernate tokens (símbolos devem ser nomes de função ou literal, para exemplo).</p> <p>por exemplo. <code>hqlLiteral = SQL_LITERAL, hqlFunc = SQLFUNC</code></p>
<code>hibernate.show_sql</code>	<p>Escreve todas as instruções SQL para console.</p> <p>por exemplo.</p>
<code>hibernate.hbm2ddl.auto</code>	<p><code>verdadeiro falso</code></p> <p>Exportar automaticamente DDL esquema para o banco de dados quando o <code>ISessionFactory</code> é criado. Com <code>create-gota</code>, O esquema de banco de dados serão descartados quando o <code>ISessionFactory</code> está fechado explicitamente.</p> <p>por exemplo. <code>criar criar-drop</code></p>

3.5.1. Dialetos SQL

Você sempre deve definir a `hibernate.dialect` propriedade para a correta `NHibernate.Dialect.Dialect` subclasse para o seu banco de dados. Isso não é estritamente essencial a menos que você deseja usar nativo ou seqüência gêneros-chave primáriação ou de bloqueio pessimista (com, por exemplo. `ISession.Lock ()` ou `IQuery.SetLockMode ()`). No entanto, se você especificar um dialeto, NHibernate usará padrões sensíveis para algumas das outras propriedades listadas abaixo, poupando-lhe o esforço de especificá-los manualmente.

Quadro 3.3. Dialetos NHibernate SQL (hibernate.dialect)

RDBMS	Dialetos
DB2	NHibernate.Dialect.DB2Dialect
PostgreSQL	NHibernate.Dialect.PostgreSQLDialect
MySQL	NHibernate.Dialect.MySQLDialect
Oracle (qualquer versão)	NHibernate.Dialect.OracleDialect
Oráculo 9/10g	NHibernate.Dialect.Oracle9Dialect
Sybase	NHibernate.Dialect.SybaseDialect
Microsoft SQL Server 2000	NHibernate.Dialect.MsSql2000Dialect
Microsoft SQL Server 7	NHibernate.Dialect.MsSql7Dialect
Firebird	NHibernate.Dialect.FirebirdDialect
SQLite	NHibernate.Dialect.SQLiteDialect

Dialetos adicionais podem estar disponíveis no pacote NHibernateContrib (ver Parte I, "NHibernateContrib Documentação"). No momento de escrever este pacote contém suporte para o Microsoft Access (Jet) motor de banco de dados.

3.5.2. Outer Join Fetching

Se o seu banco de dados suporta ANSI ou Oracle exterior une estilo, **outer join fetching** pode aumentar o desempenho limitando o número de idas e vindas de e para o banco de dados (ao custo de possivelmente mais trabalho desempenhado pelo próprio banco de dados). Buscar junção externa permite que um gráfico de objetos conectados por muitos-para-um, um-para-muitos ou um-to-one associações a ser recuperado em um único SQL `SELEÇÃO`.

Por padrão, o gráfico obtido ao carregar um objeto termina em objetos folha, coleções, objetos com proxies, ou circularidades onde ocorrem.

Para um **associação particular**, busca pode ser ativado ou desativado (o comportamento padrão e substituído) por setting da `outer-join` atributo no mapeamento XML.

Buscar junção externa pode ser desativado **globalmente** definindo a propriedade `hibernate.max_fetch_depth` para 0. Um setting de 1 ou superior habilita o outer join fetching para todas as associações um-para-um e muitos-para-um, que são, também por padrão, definida para `automático` junção externa. No entanto, um-para-muitos associações e coleções nunca são buscados com um outer-join, a menos que explicitamente declarado para cada associação particular. Este comportamento também pode ser anulado em runtime com consultas Hibernate.

Em NHibernate 1.0, `buscar` atributo pode ser usado em vez de `outer-join.fetch = "join"` é equivalente a `outer-join = "true"` E `fetch = "select"` corresponde a `outer-join = "false"`.

3.5.3. Personalizado ICacheProvider

Você pode integrar um nível de processo (ou cluster) sistema de cache de segundo nível através da implementação da interface `NHibernate.Cache.ICacheProvider`. Você pode selecionar a implementação personalizada através da criação `hibernate.cache.provider_class`.

3.5.4. Substituição Query Language

Você pode definir novos símbolos de consulta usando NHibernate `hibernate.query.substitutions`. Por exemplo:

```
hibernate.query.substitutions true = 1, false = 0
```

faria com que os tokens `verdadeiro` e `falso` ser traduzidos para literais inteiros no SQL gerado.

```
hibernate.query.substitutions toLowerCase = LOWER
```

Isto permitiria mudar o nome da função SQL `LOWER`.

3.6. Logging

NHibernate logs vários eventos usando log4net Apache.

Você pode baixar a partir log4net <http://logging.apache.org/log4net/>. Para usar log4net você precisará de uma seção de configuração no arquivo de configuração do aplicativo. Um exemplo da seção de configuração é distribuído com o NHibernate no `src / NHibernate.Test` projeto.

Recomendamos fortemente que você se familiarize com as mensagens de log do NHibernate. Muito trabalho tem sido posto em fazer o log NHibernate tão detalhado quanto possível, sem torná-lo ilegível. É um elemento essencial de solução de problemas do dispositivo. Também não se esqueça de habilitar o SQL registro conforme descrito acima (`hibernate.show_sql`). Ele é o primeiro passo quando se olha para os problemas de performance.

3.7. Implementação de um INamingStrategy

A interface `NHibernate.Cfg.INamingStrategy` permite especificar um "padrão de nomeação" para banco de dados objetos e elementos do esquema.

Você pode fornecer regras para geração automática de identificadores do banco de dados. Identificadores NET ou para processamento

Coluna "lógica" e mesa de nomes dados no arquivo de mapeamento na tabela "físico" e nomes de coluna. Este FEA-
tura ajuda a reduzir a verbosidade do documento de mapeamento, eliminando interferências repetitivas (`TBL_` prefixos, por exemplo). A estratégia padrão usado pelo NHibernate é muito mínima.

Você pode especificar uma estratégia diferente chamando `Configuration.setNamingStrategy ()` antes de adicionar-mapeamentos:

```
ISessionFactory sf = new Configuration ()
    . SetNamingStrategy (ImprovedNamingStrategy.Instance)
    . AddFile ("Item.hbm.xml")
    . AddFile ("Bid.hbm.xml")
    . BuildSessionFactory ();
```

`NHibernate.Cfg.ImprovedNamingStrategy` é uma estratégia interna que pode ser um ponto de partida útil para algumas aplicações.

3.8. Arquivo de configuração XML

Uma abordagem alternativa é especificar uma configuração completa em um arquivo chamado `hibernate.cfg.xml`. Este arquivo pode ser usado como um substituto para o `<nhibernate;>` ou `<hibernate-configuration>` seções da aplicação con-

arquivo configuração.

O arquivo de configuração XML é, por padrão esperado para a sua diretório do aplicativo. Aqui está um exemplo:

```
<? Xml version = 'encoding = ' 1 .0 utf-8 '>
xmlns="urn:nhibernate-configuration-2.0"> <hibernate-configuration>

<-! Uma instância ISessionFactory ->
<session-factory>

    <! - Propriedades ->
    <property name="connection.provider"> NHibernate.Connection.DriverConnectionProvider </ property>
    <property name="connection.driver_class"> NHibernate.Driver.SqlClientDriver </ property>
    <property name="connection.connection_string"> Servidor = localhost; catálogo inicial = nhibernate; Usu
    <property name="show_sql"> false </ property>
    <property name="dialect"> NHibernate.Dialect.MsSql2000Dialect </ property>
    <property name="use_outer_join"> true </ property>

    <-! Arquivos de mapeamento ->
    <Mapping resource="NHibernate.Auction.Item.hbm.xml" assembly="NHibernate.Auction" />
    <Mapping resource="NHibernate.Auction.Bid.hbm.xml" assembly="NHibernate.Auction" />

</ Session-factory>

</ Hibernate-configuration>
```

Configurando o NHibernate é, então, tão simples como

```
.. ISessionFactory sf = new Configuration () Configurar () BuildSessionFactory ();
```

Você pode escolher um arquivo de configuração XML usando diferentes

```
ISessionFactory sf = new Configuration ()
    . Configure ("/ caminho / para /
config.cfg.xml")
    . BuildSessionFactory ();
```

Capítulo 4. Classes persistentes

Classes persistentes são classes em uma aplicação que implementam as entidades do problema de negócio (por exemplo, Customer e Order em um aplicativo de E-commerce). Classes persistentes têm, como o nome indica, transitória e também instância persistente armazenados no banco de dados.

NHibernate funciona melhor se essas classes seguir algumas regras simples, também conhecido como o objeto de Plain Old CLR (POCO) modelo de programação.

4.1. Um exemplo simples POCO

Mais .NET aplicações requerem uma classe persistente representando felinos.

```
using System;
using Iesi.Collections;

namespace Eg
{
    Cat public class
    {
        id longo privado; / identificador /
        private string name;
        data de nascimento DateTime privado;
        companheiro Cat privado;
        privada gatinhos ISET
        Cor color privado;
        sexo private char;
        peso da bôia privado;

        Id pública virtual longa
        {
            get {return;}
            set {id = valor;}
        }

        Nome cadeia pública virtuais
        {
            get {return nome;}
            set {nome = valor;}
        }

        Cat-Mate públicas virtuais
        {
            get {companheiro de retorno;}
            set {companheiro = valor;}
        }

        public virtual DateTime Aniversário
        {
            get {return data de nascimento;}
            set {data de nascimento = valor;}
        }

        Peso flutuar públicas virtuais
        {
            get {peso return;}
            {peso = valor;} set
        }

        Color públicas virtuais
        {
            começar {color return;}
            {color = valor;} set
        }
    }
}
```

```

public virtual gatinhos ISET
{
    get {gatinhos return; }
    set {gatinhos = valor; }
}

// AddKitten não necessário por NHibernate
public void virtuais AddKitten (Cat gatinho)
{
    kittens.Add (gatinho);
}

sexo virtual public char
{
    get {sexo return; }
    {sexo = valor;} set
}

}
}

```

Há quatro principais regras a seguir aqui:

4.1.1. Declare acessores e mutadores para campos persistentes

Gat declara métodos de acesso para todos os seus campos persistentes. Muitas ferramentas de ORM outros diretamente persistem exemplo variáveis. Acreditamos que é muito melhor para dissociar este detalhe de implementação do mecanismo de persistência. NHibernate persistir propriedades, usando seus métodos getter e setter.

Propriedades necessidade não ser declarado público - NHibernate pode persistir uma propriedade com uma interno, protegido, protegida ou privado visibilidade.

4.1.2. Implementar um construtor padrão

Gat tem uma implícita construtor padrão (sem argumentos). Todas as classes persistentes devem ter um construtor padrão (Que pode ser não-pública) para NHibernate pode instanciá-los usando `ConstructorInfo.Invoke (null)`.

4.1.3. Fornecer um identificador de propriedade (opcional)

Gat tem uma propriedade chamada `ID`. Esta propriedade contém a coluna de chave primária de uma tabela do banco de dados. A propriedade poderia ter sido chamado de qualquer coisa, e seu tipo poderia ter sido qualquer tipo primitivo, corda ou `System.DateTime`. (Se sua tabela de banco de dados legado tem chaves compostas, você ainda pode usar uma classe definida pelo usuário com propriedades de estes tipos - veja a seção sobre identificadores compostos abaixo).

A propriedade identificador é opcional. Você pode deixá-lo para e deixar NHibernate manter o controle de identificadores de objetos em externamente. No entanto, para muitas aplicações ainda é uma decisão de bom design (e muito popular).

Além do mais, algumas funcionalidade está disponível apenas para classes que declarar uma propriedade de identificador:

- Atualizações em cascata (ver "objetos do Ciclo de Vida")
- `Session.SaveOrUpdate ()`

Recomendamos que você declare propriedades de identificação de forma consistente, nomeado em classes persistentes.

4.1.4. Preferem não seladas classes e métodos virtual (opcional)

A característica central do NHibernate, **proxies**, depende da classe persistente sendo não-fechado e todos os seus métodos de ods, propriedades e eventos declarado como virtual. Outra possibilidade é para a classe para implementar uma interface que declara que todos os membros públicos.

Você pode persistir `selado` classes que não implementam uma interface e não têm membros virtuais com NHibernate, mas você não será capaz de usar proxies - o que irá limitar as suas opções para ajuste de desempenho algum- o quê.

4.2. Implementar a herança

A subclasse também deve observar as regras de primeiro e segundo. Ele herda sua propriedade identificador de Gato.

```
using System;
namespace Eg
{
    DomesticCat public class: Cat
    {
        private string name;

        Nome cadeia pública virtuais
        {
            get {return nome;}
            set {nome = valor;}
        }
    }
}
```

4.3. Implementação Equals () e GetHashCode ()

Você tem que substituir o `Equals ()` e `GetHashCode ()` métodos, se você pretende misturar objetos de persistência classes (por exemplo, em um `ISET`).

Isso só se aplica se esses objetos são carregados em duas diferentes `ISessions`, como NHibernate somente garante a identidade do

`(a == b, A implementação padrão de Equals () Dentro de um único ISession!)`

Mesmo se ambos os objecs `ume` bsão a linha mesmo banco de dados (eles têm o valor primário mesma chave como a sua identifi- er), não podemos garantir que eles são a mesma instância do objeto fora de uma determinada `ISession` contexto.

A maneira mais óbvia é implementar `Equals ()/GetHashCode ()` comparando o valor do identificador de ambos os ob- jectos. Se o valor é o mesmo, ambos devem ser a linha mesmo banco de dados, por isso, são iguais (se ambos forem adicionados a um `ISET`, Só teremos um elemento no `ISET`). Infelizmente, não podemos usar essa abordagem. NHibernate só irá atribuir valores identificador para os objetos que são persistentes, uma instância recém-criada não terá qualquer identi- valor fier! Recomendamos a implementação `Equals ()` e `GetHashCode ()` utilização Igualdade de negócios.

Igualdade de negócios significa que o `Equals ()` método compara apenas as propriedades que formam o negócio chave, uma chave que identificam o nosso exemplo, no mundo real (a natural chave candidata):

```
Cat public class
{
    ...
    public override bool Equals (outro objeto)
    {
        if (this == outras) return true;

        Cat cat = outro como Cat;
        if (gato == null) return false; / / null ou não um gato
```

```

        if (! Nome cat.Name ==) return false;
        Se retornar falso (Birthday.Equals (cat.Birthday) !);

        return true;
    }

    public override int GetHashCode ()
    {
        não verificado
        {
            int resultado;
            resultado Name.GetHashCode = ();
            resultado = 29 * + resultado Birthday.GetHashCode ();
            resultado de retorno;
        }
    }

}

```

Tenha em mente que a nossa chave candidata (neste caso, um composto de nome e data de nascimento) tem de ser válido apenas para um operação de comparação particular (talvez até mesmo apenas em um único caso de uso). Nós não precisamos de os critérios de estabilidade que normalmente se aplicam a uma verdadeira chave primária!

4.4. Callbacks do ciclo de vida

Opcionalmente, uma classe persistente pode implementar a interface `ILifecycle` que fornece alguns callbacks que al- objeto a baixa persistência para executar a inicialização necessária / limpeza após salvar ou carregar e antes de exclusão ou atualização.

O NHibernate `IInterceptor` oferece uma alternativa menos invasiva, no entanto.

```

public interface ILifecycle
{
    LifecycleVeto OnSave (ISession s); (1)
    LifecycleVeto OnUpdate (ISession s); (2)
    LifecycleVeto onDelete (ISession s); (3)
    OnLoad void (s ISession, objeto id); (4)
}

```

- (1) `OnSave` - Chamado apenas antes que o objeto é salvo ou inseridos
- (2) `OnUpdate` - Chamado logo antes de um objeto é atualizado (quando o objeto é passado para `ISession.Update ()`)
- (3) `onDelete` - Chamado logo antes de um objeto é excluído
- (4) `OnLoad` - Chamado logo após um objeto é carregado

`OnSave ()`, `OnDelete ()` pode ser usado em cascata salva e exclusões de objetos dependentes. Este é uma alternativa para declarar operações em cascata no arquivo de mapeamento. `OnLoad ()` pode ser usado para inicializar transient propriedades do objeto de seu estado persistente. Não pode ser usado para carregar objetos dependentes desde o `ISession` interface não pode ser invocada por dentro deste método. A posterior utilização pretendida de `OnLoad ()`, `OnSave ()` e `OnUpdate ()` é armazenar uma referência para a corrente `ISession` para uso posterior.

Note-se que `OnUpdate ()` não é chamado toda vez que estado persistente do objeto é atualizado. Ele é chamado somente quando um objeto transiente é passado para `ISession.Update ()`.

Se `OnSave ()`, `OnUpdate ()` ou `onDelete ()` retorno `LifecycleVeto.Veto`, A operação é silenciosamente vetado. Se um `CallbackException` é lançada, a operação é vetado e a exceção é passada de volta para a aplicação.

Note-se que `OnSave ()` é chamado depois de um identificador é atribuído ao objeto, exceto quando a geração da chave é nativo

utilizada.

4.5. Callback IValidatable

Se a classe persistente precisa verificar invariantes antes de seu estado é mantido, pode aplicar as seguintes interface:

```
public interface IValidatable
{
    void Validar ();
}
```

O objeto deve lançar uma `ValidationFailure` se um invariante foi violada. Uma instância de `Validatable` deveria Não mude seu estado a partir do interior `Validate ()`.

Ao contrário dos métodos callback do `ILifecycle` interface, `Validate ()` poderia ser chamado às vezes imprevisível. O aplicativo não deve depender de chamadas para `Validate ()` para a funcionalidade de negócios.

Capítulo 5. Básicos Mapeamento O / R

5.1. Declaração de mapeamento

Mapeamentos objeto / relacional são definidas em um documento XML. O documento de mapeamento é projetado para ser de leitura capazes e mão-editável. A linguagem de mapeamento é objeto-centric, o que significa que os mapeamentos são construídos em torno de persistentes declarações de classe, as declarações não mesa. Note-se que, embora muitos usuários NHibernate optar por definir mapeamentos XML manualmente, uma série de ferramentas de ex- ist para gerar o documento de mapeamento, incluindo NHibernate.Mapping.Attributes biblioteca e vários template-baseada geradores de código (CodeSmith, MyGeneration).

Vamos começar com um mapeamento de exemplo:

```
<? Xml version = "1.0"?>
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0" assembly = "Ex"
    namespace = "Ex.:">

    <class name="Cat" table="CATS" discriminator-value="C">
        <id name="Id" column="uid" type="Int64">
            <generator class="hilo"/>
        </ Id>
        <discriminator column="subclass" type="Char"/>
        <property name="BirthDate" type="Date"/>
        <property name="color" not-null="true"/>
        <property name="Sex" not-null="true" update="false"/>
        <property name="Weight"/>
        <many-to-one name="Mate" column="mate_id"/>
        <set name="Kittens">
            <key column="mother_id"/>
            <one-to-many class="Cat"/>
        </ Set>
        <subclass name="DomesticCat" discriminator-value="D">
            <property name="nome" type="String"/>
            <Subclasse />
        </ Class>
    </ Class>

    <class name="Dog"> <class
        <! - Mapeamento para cão poderia ir aqui ->
    </ Class>

</ Hibernate-mapping>
```

Vamos agora discutir o conteúdo do documento de mapeamento. Iremos apenas descrever os elementos do documento e atributos que são usados pelo NHibernate em tempo de execução. O documento de mapeamento também contém alguns extra optional at- tributos e elementos que afetam os esquemas de banco de dados exportados pela ferramenta de exportação do esquema. (Por exemplo, o não-nulo atributo.)

5.1.1. XML Namespace

Todos os mapeamentos de XML devem declarar o namespace XML mostrado. A definição do esquema atual pode ser encontrado em

O src \ nhibernate-mapping 2.0.xsd- arquivo na distribuição NHibernate.

Dica: para habilitar o IntelliSense para arquivos de mapeamento e configuração, copiar o apropriado .Xsd arquivos para <VS.NET in- lação diretório> \ Common7 \ Packages \ schemas \ xml.

5.1.2. hibernate-mapping

Este elemento tem diversos atributos opcionais. O `esquema` atributo especifica que as tabelas de que trata este mapa de ping pertencem ao esquema chamado. Se especificado, tabelas irão ser qualificados com o nome determinado esquema. Se missing, tabelas não serão qualificados. O `default-cascade` atributo especifica qual estilo de cascata deve ser assumidos pelas propriedades e coleções que não especificar um `cascata` atributo. O `importação automática` atributo permite nos utilizar nomes de classes não qualificados na linguagem de consulta, por padrão. O `montagem` e `namespace` atributos esperar as opções de montagem, onde estão localizadas as classes persistentes eo namespace eles são declarados dentro

```
<Mapeamento hibernate-
    schema = "do-esquema"
    default-cascade = "none | save-update"
    auto-import = "true | false"
    assembly = "Ex"
    namespace = "Ex"
/>
```

- (1) `esquema`(Opcional): O nome de um esquema de banco de dados.
- (2) `default-cascade` (Opcional - valor default para `nenhum`): Um estilo cascata padrão.
- (3) `importação automática` (Opcional - valor default para `verdadeiro`): Especifica se podemos usar nomes de classes não qualificadas (de classes deste mapeamento) na linguagem de consulta.
- (4) `montagem` e `namespace`(Opcional): Especifique montagem e namespace para assumir para a classe não qualificado nomes no documento de mapeamento.
- (5)

(4))
Se você não estiver usando `montagem` e `namespace` atributos, você tem que especificar nomes de classe totalmente qualificado, incluindo-

ing o nome do conjunto que as classes são declaradas dentro

Se você tem duas classes persistentes com o nome (não qualificadas), você deve definir `auto-import = "false"`. NHibernate irá lançar uma exceção se você tentar setar duas classes para o mesmo nome "importado".

5.1.3. classe

Você pode declarar uma classe persistente utilizando o `classe` elemento:

```
Class <
    name = "ClassName"
    table = "tableName"
    discriminator-value = "discriminator_value"
    mutável = "true | false"
    schema = "dono"
    proxy = "ProxyInterface"
    dynamic-update = "true | false"
    dinâmica insere= "true | false"
    select-before-update = "true | false"
    polimorfismo = "implicit | explícito"
    onde = "sql arbitrária onde a condição"
    persistor = "PersistorClass"
    batch-size = "N"
    optimistic-lock = "none | Versão | sujo | all"
    lazy = "true | false"
/>
```

- (1) `nome`: O totalmente qualificado. NET nome da classe da classe persistente (ou interface), incluindo a sua montagem nome.
- (2) `tabela`: O nome da sua tabela de banco de dados.
- (3) `discriminador valor` (Opcional - valor default para o nome da classe): Um valor que distingue cada sub-classes, usadas para o comportamento polimórfico. Valores aceitáveis incluem `nulo` e `não nulo`.
- (4) `mutável` (Opcional, o padrão é `verdadeiro`): Especifica que as instâncias da classe são (ou não) mutáveis.
- (5) `esquema` (Opcional): Sobrepõe o nome do esquema especificado pela raiz `<hibernate-mapping>` elemento.
- (6) `procuração` (Opcional): Especifica uma interface para usar os proxies de inicialização. Você pode especificar o nome do

- a própria classe.
- (7) `dynamic-update` (Opcional, o padrão é `false`): Especifica que ATUALIZAÇÃO SQL devem ser gerados em tempo de execução e conter apenas aquelas colunas cujos valores foram alterados.
 - (8) `dynamic-insert` (Opcional, o padrão é `false`): Especifica que INSERIR SQL devem ser gerados em tempo de execução e conter apenas aquelas colunas cujos valores não estão nulos.
 - (9) `select-before-update` (Opcional, o padrão é `false`): Especifica que deve NHibernate nunca executar uma SQL ATUALIZAÇÃO a menos, é certo que um objeto é realmente modificado. Em certos casos (na verdade, somente quando um objeto transiente foi associado com uma nova sessão usando `update()`), Isso significa que vai NHibernate executar uma SQL extras SELEÇÃO para determinar se um ATUALIZAÇÃO é realmente necessária.
 - (10) `polimorfismo` (Opcional, o padrão é `implícito`): Determina se consulta implícita ou explícita polimorfismo é usado.
 - (11) `onde` (Opcional) especifica um comando SQL arbitrárias ONDE condição a ser usado quando da recuperação de objetos desta classe
 - (12) `persister` (Opcional): Especifica um personalizado `IClassPersister`.
 - (13) `batch-size` (Opcional, o padrão é 1) Especificar um "tamanho de lote" para a recuperação instâncias desta classe pelo identificador.
 - (14) `optimistic-lock` (Opcional, o padrão é `versão`): Determina a estratégia de bloqueio otimista.
 - (15) `preguiçoso` (Opcional): Configuração `lazy = "true"` é um atalho para equalvalent especificando o nome da própria classe como o procuração interface.

É perfeitamente aceitável para a classe persistente chamado para ser uma interface. Você deverá então declarar implementar aulas de ing dessa interface usando o `<subclass>` elemento. Você pode persistir qualquer classe interna. Você deve especificar o nome da classe usando a forma padrão. Eg. `Foo + Bar`, Eg. Devido a um analisador aulas HQL limitação interna pode não devem ser utilizadas em consultas em NHibernate 1.0.

Classes imutáveis, `mutable = "false"`, Não podem ser atualizados ou excluídos pela aplicação. Isto permite-NHibernate comeu fazer algumas otimizações de desempenho menor.

O opcional `procuração` atributo permite a inicialização lenta de instâncias persistentes da classe. NHibernate irá inicialmente retorno proxies que implementam a interface chamada. O objeto persistente atual será carregado quando um método do proxy é invocado. Consulte "Inicializando coleções e proxies" abaixo.

Implícito polimorfismo significa que as instâncias da classe será retornada por uma consulta que dá nome a qualquer superclasse ou implementadas interface ou a classe e as instâncias de qualquer subclasse da classe será retornada por uma consulta que os nomes da própria classe. **Explícito** polimorfismo significa que instâncias da classe será devolvida apenas consultas que explicitamente nomeiam a classe e que queries que nomeiam as classes irão retornar apenas instâncias de subclasses mapeadas dentro da `<class>` declaração como um `<subclass>` ou `<joined-subclass>`. Para a maioria dos propósitos do padrão, `polimorfismo = "implícito"`, É apropriado. Polimorfismo explícito é útil quando duas classes diferentes são mapeado para a mesma tabela (isso permite que uma classe "leve" que contém um subconjunto das colunas da tabela).

O `persister` atributo permite que você personalize a estratégia de persistência utilizada para a classe. Você pode, por exemplo, especificar sua própria subclasse de `NHibernate.Persister.EntityPersister` ou você pode até mesmo fornecer um-com completamente nova implementação da interface `NHibernate.Persister.IClassPersister` que implementa persistência através, por exemplo, chamadas de procedimento armazenado, serialização de arquivos flat ou LDAP. Ver `NHibernate.DomainModel.CustomPersister` para um exemplo simples (de "persistencia" para uma `Hashtable`).

Note que o `dynamic-update` e `dynamic-insert` definições não são herdadas por subclasses e assim também podem ser especificada no `<subclass>` ou `<joined-subclass>` elementos. Estas configurações podem aumentar o desempenho em alguns casos, mas pode realmente diminuir o desempenho em outros. Use criteriosamente.

Uso de `select-before-update` normalmente irá diminuir o desempenho. É muito útil para evitar que uma atualização de banco de dados trigger que está sendo chamado desnecessariamente.

Se você permitir `dynamic-update`, Você terá uma escolha de estratégia de bloqueio otimista:

- `versão` verifica as colunas versão / timestamp

- todo verificar todas as colunas
- sujo verifica as colunas modificadas
- nenhum não utiliza o bloqueio otimista

Nós muito Recomendamos que você use a versão / timestamp colunas para o bloqueio otimista com NHibernate. Esta é a estratégia ótima em relação ao desempenho e é a única estratégia que maneja modificações feitas fora da sessão (isto é, quando `ISession.Update ()` é usado). Tenha em mente que uma versão ou propriedade timestamp nunca deve ser nulo, não importa o que `unsaved valor` estratégia, ou uma instância será detectada como transiente.

5.1.4. id

Classes mapeadas preciso declarar a coluna de chave primária da tabela do banco de dados. A maioria das classes também terá um propriedade segurando o identificador único de uma instância. O `<id>` elemento define o mapeamento desta propriedade para a coluna de chave primária.

```
<Id
    name = "PropertyName" (1)
    type = "typename" (2)
    coluna = "column_name" (3)
    unsaved valor = "qualquer | none | null | id_value" (4)
    access = "field | imóvel | nosetter | ClassName (5)">

    <generator class="generatorClass"/>
</ Id>
```

- (1) nome (Opcional): O nome da propriedade identificador.
- (2) tipo (Opcional): Um nome que indica o tipo NHibernate.
- (3) coluna (Opcional - valor default para o nome da propriedade): O nome da coluna de chave primária.
- (4) unsaved valor (Opcional - default para um valor "sensível"): Um valor de propriedade de identificação que indica que uma instância é uma nova instanciada (unsaved), distinguindo de instâncias transitória que foram salvas ou carregado em uma sessão anterior.
- (5) acesso (Opcional - valor default para propriedade): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

Se o `nome` atributo está ausente, assume-se que a classe não tem nenhuma propriedade identificador.

O `unsaved valor` atributo quase nunca é necessária em NHibernate 1.0.

Há uma alternativa `<composite-id>` declaração para permitir o acesso a dados legados com chaves compostas. Nós desencorajam o seu uso para qualquer outra coisa.

5.1.4.1. gerador

Necessária `<generator>` elemento filho nomeia uma classe .NET usado para gerar identificadores únicos para os casos da classe persistente. Se todos os parâmetros necessários para configurar ou inicializar a instância geradora, eles são passado com o `<param>` elemento.

```
<id name="Id" type="Int64" column="uid" unsaved-value="0">
    <generator class="NHibernate.Id.TableHiLoGenerator">
        <param name="table"> uid_table </ param>
        <param name="column"> next_hi_value_column </ param>
    <Gerador />
</ Id>
```

Todos os geradores de implementar a interface `NHibernate.Id.IIdentifierGenerator`. Este é um muito simples interface; algumas aplicações podem optar por fornecer as suas próprias implementações especializadas. No entanto, NHibernate oferece uma gama de built-in implementações. Há nomes de atalhos para os geradores de built-in:

incremento

gera identificadores dos tipos `Int64`, `Int16` ou `Int32` que são únicos apenas quando nenhum outro processo está inserindo dados na mesma tabela. **Não use em um cluster.**

identidade

suporta colunas de identidade em DB2, MySQL, MS SQL Server e Sybase. O identificador retornado pela banco de dados é convertido para o tipo de propriedade usando `Convert.ChangeType`. Qualquer tipo de propriedade integral é, portanto, suportados.

seqüência

usa uma seqüência em DB2, PostgreSQL, Oracle ou um gerador no Firebird. O identificador retornado pela data-base é convertido para o tipo de propriedade usando `Convert.ChangeType`. Qualquer tipo de propriedade integral é, portanto, suportados.

hilo

utiliza um algoritmo de oi / lo para gerar de forma eficiente identificadores do tipo `Int16`, `Int32` ou `Int64`. Dada uma mesa e coluna (por padrão `hibernate_unique_key` e `next_hi` respectivamente) como fonte de valores oi. O oi / lo algoritmo gera identificadores que são únicos apenas para um banco de dados particular. **Não use este gerador com uma fornecido pelo usuário de conexão.**

seqhilo

utiliza um algoritmo de oi / lo para gerar de forma eficiente identificadores do tipo `Int16`, `Int32` ou `Int64`. Dada uma chamada de dados seqüência de base.

uuid.hex

usa `System.Guid` e sua `ToString` (formato) método para gerar identificadores do tipo string. O comprimento da corda retornado depende do configurado formato.

uuid.string

utiliza um novo `System.Guid` para criar um `byte []` que é convertido em uma string.

guid

utiliza um novo `System.Guid` como identificador.

guid.comb

usa o algoritmo para gerar um novo `System.Guid` descrito por Jimmy Nilsson no artigo [ht tp://www.informit.com/articles/article.asp?p=25862](http://www.informit.com/articles/article.asp?p=25862).

nativo

picaretas `identidade`, `seqüência` ou `hilo` dependendo das capacidades do banco de dados subjacente.

atribuído

permite que o aplicativo para definir um identificador para o objeto antes `Save ()` é chamado.

estrangeiro

usa o identificador de um outro objeto associado. Normalmente usado em conjunto com um `<one-to-one>` primário chave de associação.

5.1.4.2. Oi / Lo Algoritmo

O `hilo` e `seqhilo` geradores fornecem duas implementações alternativas do algoritmo `oi / lo`, um ap favorito OCDE para a geração de identificadores. A primeira implementação requer uma tabela de banco de dados "especiais" para manter nos próximos disponíveis "oi" de valor. A segunda utiliza uma seqüência do estilo Oracle (quando suportado).

```
<id name="Id" type="Int64" column="cat_id">
    <generator class="hilo">
        <param name="table"> hi_value </ param>
        <param name="column"> next_value </ param>
        <param name="max_lo"> 100 </ param>
    <Gerador />
</ Id>

<id name="Id" type="Int64" column="cat_id">
    <generator class="seqhilo">
        <param name="sequence"> hi_value </ param>
        <param name="max_lo"> 100 </ param>
    <Gerador />
</ Id>
```

Infelizmente, você não pode usar `hilo` ao fornecer seus próprios `IDbConnection` para NHibernate. NHibernate deve ser capaz de buscar o "oi" de valor em uma nova transação.

5.1.4.3. Algoritmo Hex UUID

```
<id name="Id" type="String" column="cat_id">
    <generator class="uuid.hex">
        <param name="format"> format_value </ param>
        <param name="seperator"> seperator_value </ param>
    <Gerador />
</ Id>
```

O UUID é gerado pelo telefone `Guid.NewGuid ()`. `ToString (formato)`. Os valores válidos para `formato` são descrito na documentação do MSDN. O padrão `seperator` é `-` e raramente deve ser modificado. O `formato` determina se o configurado `seperator` pode substituir o separador padrão usado pelo `formato`.

5.1.4.4. UUID Algoritmo Cordas

O UUID é gerado pelo telefone `Guid.NewGuid ()`. `ToByteArray ()` e, em seguida, converter o `byte []` em um `char []`. O `char []` é retornado como um `Corda` composto por 16 caracteres.

5.1.4.5. Algoritmos GUID

O `guid` identificador é gerado pelo telefone `Guid.NewGuid ()`. Para abordar algumas das preocupações com o desempenho usar GUIDs como chaves primárias, chaves estrangeiras, e como parte de índices com o MS SQL `guid.comb` pode ser usado. O benefício de usar o `guid.comb` com outros bancos de dados que GUIDs apoio não foi medido.

5.1.4.6. Colunas de identidade e seqüências

Para bancos de dados que suportam colunas de identidade (DB2, MySQL, Sybase, MS SQL), você pode usar `identidade` chave geração. Para bancos de dados que suportam sequencias (DB2, Oracle, PostgreSQL, Interbase, McKoi, SAP DB), você pode usar `seqüência` geração de chaves de estilo. As duas estratégias requerem duas consultas SQL para inserir um novo objeto.

```
<id name="Id" type="Int64" column="uid">
    <generator class="sequence">
        <param name="sequence"> uid_sequence </ param>
    <Gerador />
</ Id>
```

```
<id name="Id" type="Int64" column="uid" unsaved-value="0">
    <generator class="identity"/>
</ Id>
```

Para desenvolvimento multi-plataforma, o nativo estratégia será escolher a partir da identidade, seqüência e hilo estratégias, depende da capacidade do banco de dados subjacente.

5.1.4.7. Identificadores atribuídos

Se você deseja que o aplicativo para atribuir identificadores (em oposição a ter NHibernate gerá-los), você pode usar o atribuído gerador. Este gerador especial irá utilizar o valor do identificador atribuído ao objeto de identidade como também um identificador. Tenha muito cuidado ao usar esse recurso para atribuir teclas com significado de negócios (quase sempre uma decisão de projeto terrível).

Devido à sua natureza, entidades que utilizam este gerador não pode ser salva através de saveOrUpdate () do método. Em vez disso você tem que especificar explicitamente para NHibernate se o objeto deve ser salvo ou atualizado pelo telefone ou o Save () ou Update () método da ISession.

5.1.5. composite-id

```
<Composite-id
    name = "PropertyName"
    class = "ClassName"
    unsaved valor = "qualquer | none"
    access = "field | imóvel | nosetter | ClassName">

    <key-property name="PropertyName" type="typename" column="column_name"/>
    <key-many-to-one name="PropertyName" class= column="column_name"/>
    .....
</ Composite-id>
```

Para uma tabela com uma chave composta, você pode mapear múltiplas propriedades da classe como propriedades de identificação. O

`<composite-id>` elemento aceita `<key-property>` mapeamentos de propriedade e `<key-many-to-one>` mapeamentos como elementos filho.

```
<composite-id>
    <key-property name="MedicareNumber"/>
    <key-property name="Dependent"/>
</ Composite-id>
```

Sua classe persistente preciso substituir `Equals ()` e `GetHashCode ()` para implementar a igualdade identificador composto. Ele também deve ser `Serializable`.

Infelizmente, essa abordagem para um identificador composto significa que um objeto persistente é seu próprio identificador. Lá não é "handle" que não seja o próprio objeto. Você deve criar uma instância da classe persistente it-autu e preencher suas propriedades de identificação antes de poder `load ()` o estado persistente associado com uma composição chave. Iremos descrever uma abordagem muito mais conveniente, onde o identificador composto é implementado como um classe separada na Seção 7.4, "Componentes como identificadores compostos". Os atributos descritos abaixo aplicam-se apenas a esta abordagem alternativa:

- `nome` (Opcional, necessária para esta abordagem): Uma propriedade do tipo componente que armazena o composto identificador (ver secção seguinte).
- `acesso` (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade valor.
- `classe` (Opcional - default para o tipo de propriedade determinada por reflection): classe A componente usado como um identificador composto (veja próxima seção).

5.1.6. discriminador

O <discriminador> elemento é necessário para persistência polimórfica utilizando o mapa-table-per-class-hierarchy estratégia de ping e declara uma coluna discriminadora da tabela. A coluna discriminadora contem marcador valores que contam a camada de persistência qual subclasse instanciar para uma linha particular. Um conjunto restrito de tipos podem ser usado: `Corda,Char,Int32,Byte,Curto,Boolean,YesNo,TrueFalse`.

```
<Discriminador
    coluna = "discriminator_column"      (1)
    type = "discriminator_type"         (2)
    force = "true | false"             (3)
    insert = "true | false"           (4)
/>
```

- (1) `coluna` (Opcional - valor default para `classe`) O nome da coluna discriminadora.
- (2) `tipo` (Opcional - valor default para `Corda`) Um nome que indica o tipo NHibernate
- (3) `força` (Opcional - valor default para `falso`) "Força" NHibernate para especificar valores discriminadores permitidos mesmo ao recuperar todas as instâncias da classe raiz.
- (4) `inserir` (Opcional - valor default para `verdadeiro`) Este conjunto de `falso` se sua coluna discriminadora é também parte de um mapeamento identificador composto.

Valores reais da coluna discriminadora são especificados pelo `discriminador valor` atributo do <class> e <subclass> elementos.

O `força` atributo é (apenas) útil se a tabela contém linhas com "extra" valores discriminadores que não são mapeados para uma classe persistente. Isso não será normalmente o caso.

5.1.7. versão (opcional)

O <versão> elemento é opcional e indica que a tabela contém dados versionados. Isto é particularmente uso útil se você planeja usar transações longas (Veja abaixo).

```
Versão <
    coluna = "version_column"          (1)
    name = "PropertyName"            (2)
    tipo = "typename"                (3)
    access = "field | imóvel | nosetter | ClassName" (4)
    unsaved-value = "null | negativos | undefined | Valor" (5)
/>
```

- (1) `coluna` (Opcional - valor default para o nome da propriedade): O nome da coluna mantendo o número da versão.
- (2) `nome`: O nome de uma propriedade da classe persistente.
- (3) `tipo` (Opcional - valor default para `Int32`): O tipo do número da versão.
- (4) `acesso` (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade `valor`.
- (5) `unsaved valor` (Opcional - default para um valor "sensível"): um valor a propriedade versão que indica que um instância é uma nova instaciada (`unsaved`), distinguindo de instâncias transitória que foram salvas ou carregado em uma sessão anterior. (`undefined` especifica que o valor da propriedade de identificação deve ser usado.)

Números de versão podem ser do tipo `Int64,Int32,Int16,Carrapatos,Timestamp` Ou `TimeSpan`.

5.1.8. timestamp (opcional)

O opcional <timestamp> elemento indica que a tabela contém dados timestamped. Esta pretende ser uma alternativa a versão. Timestamps são por natureza uma implementação menos segura do locking otimista. No entanto,

por vezes, a aplicação pode usar timestamps em outros caminhos.

```
<Timestamp
    coluna = "timestamp_column" (1)
    name = "PropertyName" (2)
    access = "field | imóvel | nosetter | Clas (3) sName"
    unsaved-value = "null | Valor | undefined" (4)
/>
```

- (1) **coluna** (Opcional - valor default para o nome da propriedade): O nome da coluna que mantem o timestamp.
- (2) **nome**: O nome de uma propriedade do tipo NET `DateTime` da classe persistente.
- (3) **acesso** (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade valor.
- (4) **unsaved valor** (Opcional - valor default para `null`): Um valor da propriedade timestamp que indica que uma instância é nova instanciada (`unsaved`), distinguindo de instâncias transitória que foram salvas ou carregadas em uma pré-sessão prévia. (`undefined` especifica que o valor da propriedade de identificação deve ser usado.)

Note-se que `<timestamp>` é equivalente a `<version type="timestamp">`.

5.1.9. propriedade

O `<property>` elemento declara uma propriedade da classe persistente.

```
Propriedade <
    name = "propertyName" (1)
    coluna = "column_name" (2)
    type = "typename" (3)
    update = "true | false" (4)
    insert = "true | false" (4)
    fórmula = "expressão SQL arbitrária" (5)
    access = "field | imóvel | ClassName" (6)
/>
```

- (1) **nome** o nome da propriedade da sua classe.
- (2) **coluna** (Opcional - valor default para o nome da propriedade): o nome da coluna mapeada do banco de dados de mesa.
- (3) **tipo** (Opcional): um nome que indica o tipo NHibernate.
- (4) **update, insert** (Opcional - valor default para `verdadeiro`): Especifica que as colunas mapeadas devem ser incluídas no SQL ATUALIZAÇÃO e / ou INSERIR declarações. Definição tanto para `falso` permite um puro propriedade "derivados", cuja valor é inicializado de outra propriedade que mapeie a mesma coluna (s) ou por uma trigger ou ap-outras cação.
- (5) **fórmula** (Opcional): uma expressão SQL que define o valor para uma computadorizada propriedade. Computadorizada adequada-
- (6) **laços** não tem um mapeamento de coluna própria.
- acesso** (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

typename poderia ser:

1. O nome de um tipo NHibernate básica (ex. `Int32, String, Char, DateTime, Timestamp, Solteiro, Byte [], Object, ...`).
2. O nome de um tipo .NET com um tipo de padrão básico (ex.: `System.Int16, System.Single, System.Char, System.String, System.DateTime, System.Byte [], ...`).
3. O nome de um tipo de enumeração (ex. `Eg.Color, Eg`).
4. O nome de um tipo serializável .NET.
5. O nome da classe de um tipo customizado (ex. `Illflow.Type.MyCustomType`).

Note-se que você tem que especificar completa conjunto-qualificado nomes para todos, exceto tipos básicos NHibernate (a menos que você conjunto montagem e / ou namespace atributos do `<hibernate-mapping>` elemento).

Se você não especificar um tipo, NHibernate irá utilizar reflexão sobre a propriedade nomeada para ter uma idéia do correct tipo NHibernate. NHibernate irá tentar interpretar o nome da classe de retorno da propriedade getter usando regras 2, 3, 4, nessa ordem. No entanto, isso nem sempre é suficiente. Em certos casos, você ainda vai precisar do `tipo` atributo. (Por exemplo, para distinguir entre `NHibernate.DateTime` e `NHibernate.Timestamp`, Ou para especificar um tipo personalizado.)

O `acesso` atributo permite que você controle como NHibernate irá aceder ao valor da propriedade em tempo de execução. O valor do `acesso` atributo deve ser um texto formatado como `acesso strategy.naming` estratégia. O `. Nomeando-stragey` nem sempre é necessária.

Tabela 5.1. Estratégias de acesso

Nome Estratégia de Acesso	Descrição
<code>propriedade</code>	A implementação padrão. NHibernate usa o / get definir os assessores da propriedade. Nenhuma estratégia de nomeação devem ser usados com essa estratégia porque o acesso valor do <code>nome</code> atributo é o nome do prop-pobreza.
<code>campo</code>	NHibernate irá acessar o campo diretamente. NHibernate usa o valor da <code>nome</code> atributo como o nome do de campo. Isto pode ser usado quando uma propriedade getter e setter conter ações extras que você não quer ocorrer quando NHibernate é preencher ou ler o object. Se quiser que o nome da propriedade e não o campo para ser o que os consumidores de seu uso com API HQL, então uma estratégia de nomeação é necessária.
<code>nosetter</code>	NHibernate irá acessar o campo diretamente ao definir o valor e vai usar o imóvel quando chegar o valor. Isto pode ser usado quando uma propriedade só expõe um acessador get porque os consumidores de sua API não pode alterar o valor diretamente. A estratégia de nomeação é necessário porque NHibernate usa o valor da <code>nome</code> atributo como o nome da propriedade e precisa ser disso que o nome do campo é.
<code>ClassName</code>	Se NHibernate é construído em estratégias de acesso não são o que é necessária para a sua situação, então você pode construir seu própria, implementar a interface <code>NHibernate.Property.IPropertyAccessor</code> . O valor da <code>acesso</code> atributo deve ser um qualificado-montagem nome que pode ser carregado com <code>Ati-or.CreateInstance (string assemblyQualified-Nome).</code>

Tabela 5.2. Estratégias de nomeação

Naming Nome Estratégia	Descrição
CamelCase	O <code>nome</code> atributo é convertido em caso de camel para encontrar o campo. <code><Nome da propriedade = "Foo" ... ></code> usa o campo <code>foo</code> .
CamelCase-sublinhado	O <code>nome</code> atributo é convertido em caso de camel e pré-fixa com um sublinhado para encontrar a campo. <code>Propriedade <name = "Foo" ... ></code> usa o campo <code>_foo</code> .
minúsculas	O <code>nome</code> atributo é convertido para minúsculas para encontrar Campo. <code><Nome da propriedade = "foobar" ... ></code> usa o campo <code>foobar</code> .
minúsculas, sublinhado	O <code>nome</code> atributo é convertido para minúsculas e pré-fixa com um sublinhado para encontrar a campo. <code>Propriedade <name = "foobar" ... ></code> usa o campo <code>_foobar</code> .
pascalcase-sublinhado	O <code>nome</code> atributo é prefixado com um sublinhado para encontrar o campo. <code><Nome da propriedade = "Foo" ... ></code> usa o campo <code>_foo</code> .
pascalcase-m-sublinhado	O <code>nome</code> atributo é prefixado com o caráter <code>me</code> um sublinhado para encontrar a campo. <code>Propriedade <name = "Foo" ... ></code> usa o campo <code>m_Foo</code> .

5.1.10. many-to-one

Uma associação ordinária para outra classe persistente é declarada usando um `many-to-one` elemento. O relacional modelo é uma associação many-to-one. (É realmente apenas uma referência a um objeto.)

```

<Many-to-one
    name = "PropertyName"
    coluna = "column_name"
    class = "ClassName"
    cascade = "all | none | save-update | delete"
    fetch = "join | escolher"
    update = "true | false"
    insert = "true | false"
    property-ref = "PropertyNameFromAssociatedClass"
    access = "field | imóvel | nosetter | ClassName"
    unique = "true | false"
/>

```

- (1) `nome` O nome da propriedade.
- (2) `coluna` (Opcional): O nome da coluna.
- (3) `classe` (Opcional - default para o tipo de propriedade determinada por reflection): O nome do associado classe.
- (4) `cascata` (Opcional): Especifica que as operações devem ser em cascata do objeto pai para o assunto ated.
- (5) `buscar` (Opcional - valor default para `selecionar`): Escolhe entre ir buscar-junção externa ou recuperação seqüencial.
- (6) `update, insert` (Opcional - valor default para `verdadeiro`) Especifica que as colunas mapeadas devem ser incluídas no

SQL ATUALIZAÇÃO e / ou INSERIR declarações. Definição tanto para `falso` permite uma pura associação "derivados", cuja valor é inicializado de outra propriedade que mapeie a mesma coluna (s) ou por uma trigger ou ap-outras cação.

- (7) **property-ref:** (Opcional) O nome de uma propriedade da classe associada que é ligada a esta chave estrangeira.
- (8) Se não for especificado, a chave primária da classe associada é utilizada.
- acesso (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade valor.
- único (Opcional): Habilita a geração de DDL de uma restrição de unicidade para a coluna de chave externa.
- (9)

O `cascata` atributo permite os seguintes valores: `todos,save-update,excluir,nenhum`. Definir um valor diferente do que `nenhum` irá propagar certas operações para o objeto associado (filho). Ver "objetos Lifecycle" abaixo.

O `buscar` atributo aceita dois valores diferentes:

- `juntar` Buscar a associação com uma associação externa
- `selecionar` Buscar a associação usando uma consulta separada

Um típico `many-to-one` declaração parece tão simples como

```
<many-to-one name="product" class="Product" column="PRODUCT_ID"/>
```

O `property-ref` atributo deve ser usado apenas para dados de mapeamento legados onde uma chave estrangeira se refere a uma única chave da tabela associada que não seja a chave primária. Este é um modelo relacional desagradável. Por exemplo, suponha `O Produto` classe teve um número de série único, que não é a chave primária. (A `único` controles atributo Geração de DDL NHibernate com a ferramenta SchemaExport.)

```
<property name="serialNumber" unique="true" type="string" column="SERIAL_NUMBER"/>
```

Em seguida, o mapeamento para `OrderItem` pode usar:

```
<many-to-one name="product" property-ref="serialNumber" column="PRODUCT_SERIAL_NUMBER"/>
```

Isto certamente não é incentivada, entretanto.

5.1.11. um-para-um

A associação de um-para-um para outra classe persistente é declarada usando um `um-para-um` elemento.

```
<One-to-one
    name = "PropertyName"
    class = "ClassName"
    cascade = "all | none | save-update | delete"
    constrained = "true | false"
    fetch = "join | escolher"
    property-ref = "PropertyNameFromAssociatedClass"
    access = "field | imóvel | nosetter | ClassName"
/>
```

- (1) nome O nome da propriedade.
- (2) classe (Opcional - default para o tipo de propriedade determinada por reflection): O nome do associado classe.
- (3) cascata (Opcional) especifica que as operações devem ser em cascata do objeto pai para o associado objeto.
- (4) constrangido (Opcional) especifica que uma restrição de chave estrangeira na chave primária da tabela mapeada referências da tabela da classe associada. Esta opção afeta a ordem em que `Save ()` e `Delete ()` são em cascata (e também é usado pela ferramenta schema export).

- (5) `busca` (Opcional - valor default para selecionar): Escolhe entre ir buscar-junção externa ou recuperação seqüencial.
- (6) `property-ref`: (Opcional) O nome da propriedade da classe associada que é ligada a chave primária desta classe. Se não for especificado, a chave primária da classe associada é utilizada.
- (7) `acesso` (Opcional - valor default para propriedade): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

Há duas variedades de um-para-uma associação:

- associação de chave primária
- única estrangeira principais associações

Associação de chave primária não precisa de uma coluna da tabela extra; se duas linhas são relacionadas pela associação, em seguida, o

duas linhas da tabela dividem o mesmo valor de chave primária. Então, se você quer que dois objetos sejam relacionados por uma chave primária as-
sociação, você deve se certificar de que eles são atribuídos o mesmo valor identificador!
Para uma associação de chave primária, adicione os seguintes mapeamentos em `Empregado` e `Pessoa`,
Respectivamente.

```
<one-to-one name="Person" class="Person"/>
```

```
<one-to-one name="Employee" class="Employee" constrained="true"/>
```

Agora devemos assegurar que as chaves primárias das linhas relacionadas em tabelas PERSON e EMPLOYEE são iguais.
Nós usamos uma estratégia de geração de identificador chamado especial NHibernate `estrangeiro`:

```
<class name="Person" table="PERSON">
    <id name="Id" column="PERSON_ID">
        <generator class="foreign">
            <param name="property"> Employee </ param>
        <Gerador />
    </ Id>
    ...
    <One-to-one name = "Funcionário"
        class = "Funcionário"
        constrained = "true" />
</ Class>
```

A instância recém-salva de `Pessoa` é então atribuído o mesmo valor de chave primaria como o `Empregado` exemplo referido com o `Empregado` propriedade de que `Pessoa`.

Alternativamente, uma chave estrangeira com uma restrição de unicidade, de `Empregado` para `Pessoa`, Pode ser expressa como:

```
<many-to-one name="Person" class="Person" column="PERSON_ID" unique="true"/>
```

E essa associação pode ser feita bidirecional, adicionando o seguinte ao `Pessoa` mapeamento:

```
<one-to-one name="Employee" class="Employee" property-ref="Person"/>
```

5.1.12. componente, componente dinâmico

O `<component>` mapas elemento propriedades de um objeto filho para colunas da tabela de uma classe pai. COMPONENTS pode, por sua vez, declarar suas próprias propriedades, componentes ou coleções. Consulte "Componentes" abaixo.

```
Componente <
    name = "PropertyName"
    class = "ClassName"
    insert = "true | false"                                (1)
    (2)
    (3)
```

```

update = "true | false" (4)
access = "field | imóvel | nosetter | Clas (5) sName">

<property ..../>
<Many-to-one .... />
.....
</ Component>

```

- (1) nome (O nome da propriedade).
- (2) classe (Opcional - default para o tipo de propriedade determinada por reflection): O nome do componente Classe (filha).
- (3) inserir: As colunas mapeadas aparecem em SQL INSERIRS?
- (4) atualizar: As colunas mapeadas aparecem em SQL ATUALIZAÇÃOS?
- (5) acesso (Opcional - valor default para propriedade): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

A criança `<property>` Tag mapear propriedades da classe filha para colunas da tabela.

O `<component>` elemento permite que um `<parent>` subelemento que mapeia uma propriedade da classe do componente como um referência de volta para a entidade que contém.

O `<dynamic-component>` elemento permite que um `IDictionary` a ser mapeado como um componente, onde a propriedade nomes se referem às chaves do dicionário.

5.1.13. subclasse

Finalmente, a persistência polimórfica requer a declaração de cada subclasse da classe raiz persistente. Para o (Recomendado) table-per-class-hierarchy estratégia de mapeamento, o `<subclass>` declaração é usada.

```

Subclasse <
    name = "ClassName"
    discriminator-value = "discriminator_value"          (1)
    proxy = "ProxyInterface"                            (2)
    lazy = "true | false"                             (3)
    dynamic-update = "true | false"                   (4)
    dinâmica insere= "true | false">

    <Propriedade .... />
    ....
<Subclasse />

```

- (1) nome (O totalmente qualificado. NET nome da classe da subclasse, incluindo seu nome montagem).
- (2) discriminador valor (Opcional - valor default para o nome da classe): Um valor que distingue cada sub-classes.
- (3) procuração (Opcional): Especifica a classe ou interface para usar os proxies de inicialização.
- (4) preguiçoso (Opcional): Configuração `lazy = "true"` é um atalho para equalivalent especificando o nome da própria classe como o procuração interface.

Cada subclasse deve declarar suas próprias propriedades persistentes e subclasses. `<versão>` e `<id>` propriedades são assumido ser herdado da classe raiz. Cada subclasse numa hierarquia deve definir um único discriminador-valor. Se nenhum for especificado, o totalmente qualificado. NET nome da classe é usado.

5.1.14. joined-subclass

Alternativamente, uma subclasse que é mantido a sua própria tabela (table-per-subclass estratégia de mapeamento) é declarado nos-que estabelece um `<joined-subclass>` elemento.

```

<Subclasse juntou-
  name = "ClassName"                                (1)
  proxy = "ProxyInterface"                         (2)
  lazy = "true | false"
  dynamic-update = "true | false"
  dinâmica insere= "true | false">

  <Chave .... >

  <Propriedade .... />
  ....
</ Joined-subclass>

```

- (1) nomeO nome da classe totalmente qualificada da subclasse.
- (2) procuração (Opcional): Especifica a classe ou interface para usar os proxies de inicialização.
- (3) preguiçoso (Opcional): Configuração `lazy = "true"` é um atalho para equalivalent especificando o nome da própria classe como o procuração interface.

Não coluna discriminadora é necessário para esta estratégia de mapeamento. Cada subclasse deve, no entanto, declarar uma tabela coluna com o identificador do objeto usando o `<key>` elemento. O mapeamento no início do capítulo seria re-escrita como:

```

<? Xml version = "1.0"?>
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0" assembly = "Ex"
  namespace = "Ex.:>

  <class name="Cat" table="CATS">
    <id name="Id" column="uid" type="Int64">
      <generator class="hilo"/>
    </ Id>
    <property name="BirthDate" type="Date"/>
    <property name="color" not-null="true"/>
    <property name="Sex" not-null="true"/>
    <property name="Weight"/>
    <many-to-one name="Mate"/>
    <set name="Kittens">
      <key column="MOTHER"/>
      <one-to-many class="Cat"/>
    </ Set>
    <joined-subclass name="DomesticCat" table="DOMESTIC_CATS">
      <key column="CAT"/>
      <property name="nome" type="String"/>
    </ Joined-subclass>
  </ Class>

  name="Dog"> <class
    <! - Mapeamento para cão poderia ir aqui ->
  </ Class>

</ Hibernate-mapping>

```

5.1.15. mapa, conjunto, lista de saco,

Coleções são discutidos mais tarde.

5.1.16. importação

Suponha que seu aplicativo tem duas classes persistentes com o mesmo nome, e você não quer especificar o pleno nome qualificado em consultas NHibernate. Classes podem ser "importados" de forma explícita, do que depender de `auto-import = "true"`. Você pode até importar classes e interfaces que não estão explicitamente mapeadas.

```

<import class="System.Object" rename="Universe"/>

```

```

Importação <
    class = "ClassName"          (1)
    renomear = "ShortName"       (2)
/>

```

- (1) `class`: nome da classe totalmente qualificado de qualquer classe .NET, incluindo o seu nome de montagem.
(2) `renomear` (Opcional - valor default para o nome da classe não qualificado): Um nome que pode ser usado na consulta language.

5.2. Tipos NHibernate

5.2.1. Entidades e valores

Para entender o comportamento de vários .NET nível de linguagem objetos em relação ao serviço de persistência, nós necessidade de classificá-los em dois grupos:

Um **entidade** existe independentemente de qualquer outro objeto guardando referências para a entidade. Em contraste com o usual Java modelo em que um objeto não referenciado é lixo coletado. Entidades devem ser explicitamente salvas ou deletada (Exceto que salva e deleções podem ser **em cascata** a partir de uma entidade-mãe para seus filhos). Este é diferente do ODMG modelo de persistência do objeto por acessibilidade - e corresponde mais a objetos como a aplicação são geralmente utilizados em grandes sistemas. Entidades de apoio referências circulares e compartilhada. Eles também podem ser versionados.

Estado persistente de uma entidade é composta de referências a outras entidades e instâncias de **valor tipos**. Valores são prim-
itives, coleções, componentes e certos objetos imutáveis. Entidades distintas, valores (em coleções particulares e componentes) **são** persistidos e apagados por acessibilidade. Uma vez que objetos de valor (e primitivos) são persistentes e apagados junto com as entidades que contenham não podem ser versionados independentemente. Valores não têm inde-
identidade independente, de modo que não pode ser compartilhada por duas entidades ou coleções.

Todos os tipos NHibernate exceto coleções suportam semânticas nulo se o tipo .NET é anulável (ou seja, não derivado a partir de `System.ValueType`).

Até agora, estamos usando a "classe persistente" para se referir a entidades. Continuaremos a fazer isso. Estritamente falando, porém, não todas as classes definidas pelo usuário com estados persistentes são entidades. A componente é um usuário classe definida com valores semânticos.

5.2.2. Tipos básicos de valor

O **tipos básicos** pode ser grosseiramente classificados em três grupos - `System.ValueType` tipos, `System.Object` tipos e `System.Object` tipos de objetos grandes. Assim como os tipos .NET, colunas para `System.ValueType` tipos **não pode** loja nulo valores e tipos `System.Object` pode loja nulo valores.

Quadro 5.3. Mapeamento de Tipos `System.ValueType`

Tipo NHibernate	.NET Tipo	Tipo de banco de dados	Observações
<code>AnsiChar</code>	<code>System.Char</code>	<code>DbType.AnsiStringFixedLength - 1 char</code>	
<code>Boolean</code>	<code>System.Boolean</code>	<code>DbType.Boolean</code>	Padrão quando nenhuma tipo atributo especificado.

Tipo NHibernate	.NET Tipo	Tipo de banco de dados	Observações
Byte	System.Byte	DbType.Byte	Padrão quando nenhuma tipo atributo especificado.
Char	System.Char	DbType.StringFixedLength - 1 char	Padrão quando nenhuma tipo atributo especificado.
DateTime	System.DateTime	DbType.DateTime	Padrão quando nenhum tipo atores o millisecondes ou nanoseconds especificado.
Decimal	System.Decimal	DbType.Decimal	Padrão quando nenhuma tipo atributo especificado.
Duplo	System.Double	DbType.Double	Padrão quando nenhuma tipo atributo especificado.
Guid	System.Guid	DbType.Guid	Padrão quando nenhuma tipo atributo especificado.
Int16	System.Int16	DbType.Int16	Padrão quando nenhuma tipo atributo especificado.
Int32	System.Int32	DbType.Int32	Padrão quando nenhuma tipo atributo especificado.
Int64	System.Int64	DbType.Int64	Padrão quando nenhuma tipo atributo especificado.
PersistentEnum	ASystem.Enum	O DbType para menores de Donotspecify mentindo valor.type = "PersistentEnum"	no mapeamento. Em vez especificar a Assembléia Nome qualificado do Enum ou deixar NHibernate usar a reflexão para "adivinar" o Type. A-underly ingType do Enum é usados para determinar a cor rect DbType.
Único	System.Single	DbType.Single	Padrão quando nenhuma tipo atributo especificado.
Carrapatos	System.DateTime	DbType.Int64	type = "Carrapato" deve ser especificado.
TimeSpan	System.TimeSpan	DbType.Int64	Padrão quando nenhuma tipo atributo especificado.
Timestamp	System.DateTime	DbType.DateTime	Como type = "Timestamp" preciso específicas como banco de dados sup-ser especificado. portos.
TrueFalse	System.Boolean	DbType.AnsiStringFixedLength - 1 char ou 'T' ou 'F'	type = "TrueFalse" preciso ser especificado.

Tipo NHibernate	.NET Tipo	Tipo de banco de dados	Observações
YesNo	System.Boolean	DbType.AnsiStringFixedLength - 1 char ou 'Y' ou 'N'	type = "YesNo" deve ser especificado.

Tabela 5.4. Mapeamento de Tipos de System.Object

Tipo NHibernate	.NET Tipo	Tipo de banco de dados	Observações
AnsiString	System.String	DbType.AnsiString	type = "AnsiString" preciso ser especificado.
CultureInfo	System.Globalization.CultureInfo	DbType.String - 5 chars padrão quando nenhum tipo atributo para a cultura	- 5 chars padrão quando nenhum tipo atributo especificado.
Binário	System.Byte []	DbType.Binary	Padrão quando nenhuma tipo atributo especificado.
Tipo	System.Type	DbType.String Montagem Nome.	segurando padrão quando nenhum tipo atributo qualificado especificado.
Corda	System.String	DbType.String	Padrão quando nenhuma tipo atributo especificado.

Tabela 5.5. Tipos de mapeamento de objetos grandes

Tipo NHibernate	.NET Tipo	Tipo de banco de dados	Observações
StringClob	System.String	DbType.String	type = "StringClob" ser especificado. Campo inteiro é lido na memória.
BinaryBlob	System.Byte []	DbType.Binary	type = "BinaryBlob" ser especificado. Campo inteiro é lido na memória.
Serializable	Qualquer System.Object que é marcada com SerializableAttribute.	DbType.Binary	type = "Serializable" deve ser especificado. Este é o tipo de reserva, caso não Tipo NHibernate pode ser encontrado para a propriedade.

NHibernate suporta alguns nomes de tipos adicionais para compatibilidade com Hibernate (útil para quem vem ao longo de hibernação ou usando algumas das ferramentas para gerar hbm.xml arquivos). A type = "inteiro" ou type = "int" será mapeado para um Int32 Tipo de NHibernate, type = "short" a um Int16 NHibernateType. Para ver todas as conversões você pode visualizar a fonte do construtor estático da classe NHibernate.Type.TypeFactory.

5.2.3. Personalizado tipos de valor

É relativamente fácil para os desenvolvedores a criar seus próprios tipos de valor. Por exemplo, você pode querer persistir propriedades do tipo `Int64` para `VARCHAR` colunas. NHibernate não fornece um tipo built-in para isso. Mas tipos personalizados não estão limitados a mapeamento de uma propriedade (ou elemento de coleção) a uma única coluna da tabela. Assim, por exemplo, você pode ter uma propriedade `Nome {get;}` do tipo `Corda` que é mantido para as colunas `FIRST_NAME,INITIAL,SOBRENOME`.

Para implementar um tipo personalizado, implemente tanto `NHibernate.IUserType` ou `NHibernate.ICompositeUserType` e declare propriedades usando o nome totalmente qualificado do tipo. Confira `NHibernate.DomainModel.DoubleString` para ver o tipo de coisas que são possíveis.

```
<property name="TwoStrings" type="NHibernate.DomainModel.DoubleStringType, NHibernate.DomainModel">
    <column name="first_string"/>
    <column name="second_string"/>
</ Property>
```

Observe o uso da `<column>` etiquetas para mapear uma propriedade para várias colunas.

Apesar de rica variedade NHibernate de built-in tipos e suporte para componentes significa que você terá muito raramente necessidade usar um tipo personalizado, é no entanto considerada boa forma de usar os tipos personalizados para (não-entidade) classes

que ocorrem com freqüência em sua aplicação. Por exemplo, um `MonetaryAmount` classe é um bom candidato para uma `ICompositeUserType`, Mesmo que ele poderia facilmente ser mapeado como um componente. Uma motivação para isso é abstrair. Com um tipo de costume, os seus documentos de mapeamento seria à prova de futuro contra possíveis mudanças no sua maneira de representar valores monetários.

5.2.4. Qualquer tipo de mapeamentos

Há um tipo de propriedade de mapeamento. O `<any>` elemento de mapeamento define uma associação polimórfica às aulas de várias tabelas. Este tipo de mapeamento sempre requer mais de uma coluna. A primeira coluna possui o tipo da entidade associada. As restantes colunas possuem o identificador. É impossível especificar uma restrição de chave estrangeira para este tipo de associação, de modo que este não é certamente entendido como a maneira usual de mapeamento (polimórfico) associações. Você deve usar isso só em casos muito especiais (exemplo: audit logs, sessão do usuário de dados, etc).

```
<qualquer name="AnyEntity" id-type="Int64" meta-type="Eg.Custom.Class2TablenameType">
    <column name="table_name"/>
    <column name="id"/>
</ Qualquer>
```

O `meta-type` atributo permite a aplicação especificar um tipo personalizado que os valores da coluna mapas banco de dados para persistir-entidades que tem propriedades identificadoras do tipo especificado por `id-type`. Se as instâncias meta-type retorna de `System.Type`, Nada mais é necessário. Por outro lado, se é um tipo básico como `Corda` ou `Char`, Você deve especificar o mapeamento de valores para classes.

```
<qualquer name="AnyEntity" id-type="Int64" meta-type="String">
    <meta-value value="TBL_ANIMAL" class="Animal"/>
    <meta-value value="TBL_HUMAN" class="Human"/>
    <meta-value value="TBL_ALIEN" class="Alien"/>
    <column name="table_name"/>
    <column name="id"/>
</ Qualquer>
```

```
<Qualquer
    name = "PropertyName" (1)
    id-type = "idtypename" (2)
    meta-type = "metatypename" (3)
    cascade = "none | todos | save-update" (4)
    access = "field | imóvel | nosetter | ClassName (5)"
```

```
>
<Meta de valor ... />
<Meta de valor ... />
.....
Coluna <.... />
Coluna <.... />
.....
</ Qualquer>
```

- (1) nome o nome da propriedade.
- (2) id-type: O tipo de identificador.
- (3) meta-type (Opcional - valor default para tipo): Um tipo que mapeia System.Type a uma coluna de banco de dados único ou, alternativamente, um tipo que é permitido para um mapeamento discriminador.
- (4) cascata (Opcional - valor default para nenhum): O estilo em cascata.
- (5) acesso (Opcional - valor default para propriedade): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

5.3. Identificadores SQL citada

Você pode forçar NHibernate para citar um identificador no SQL gerado colocando o nome da tabela ou coluna em backticks no documento de mapeamento. NHibernate irá usar o estilo citação correta para o SQL Dialeto (Geralmente entre aspas, mas suportes para SQL Server e backticks para MySQL).

```
<class name="LineItem" table="`Line Item`">
    <id name="Id" column="`Item Id`"/> <generator class="assigned"/> </ id>
    <property name="ItemNumber" column="`Item #`"/>
    ...
</ Class>
```

5.4. Modular arquivos de mapeamento

É possível definir subclasses e joined-subclass em documentos de mapeamento separados, diretamente estabelecendo hibernate-mapping. Isso permite que você estender uma hierarquia de classes apenas adicionando um novo arquivo de mapeamento. Você deve especificar um estende-se atributo no mapeamento da classe subclasse, nomeando uma superclasse previamente mapeada. O uso deste característica faz com que a ordenação dos documentos de mapeamento importante!

```
<hibernate-mapping>
    <Nome subclasses = "Eg.Subclass.DomesticCat, por exemplo"
        estende = "Eg.Cat, por exemplo" discriminador valor = "D">
        <property name="nome" type="string"/>
    <Subclasse />
</ Hibernate-mapping>
```

Capítulo 6. Mapeamento de coleção

6.1. Coleções persistentes

Esta seção não contém muito exemplo de código C#. Nós assumimos que você já sabe como usar .NET coleções quadro. Se assim for, não há realmente mais nada saber - com uma advertência simples, você pode usar .NET collections da mesma forma que você sempre tem.

NHibernate pode persistir instâncias de `System.Collections.IDictionary`, `Iesi.Collections.ISet`, `System.Collections.IList`, E qualquer conjunto de entidades persistentes ou valores. Propriedades do tipo `System.Collections.ICollection` ou `System.Collections.IList` também pode ser mantido com o "saco" semântica.

Agora, a advertência: coleções persistentes não reter qualquer semântica extra adicionado pela classe que implementa a coleta de interface (por exemplo, ordem de iteração de um `ListDictionary`). As coleções persistentes realmente se comportam como

`Hashtable`, `HashSet` e `ArrayList` respectivamente (com exceção de `SortedList` e `SortedSet` que fazer manter a ordem de classificação). Além disso, o tipo de uma propriedade segurando uma coleção deve ser o tipo de interface (ou seja,

`IDictionary`, `ISET` ou `IList`; Nunca `Hashtable`, `SortedSet` ou `ArrayList`). Esta restrição existe porque, quando você não está olhando, NHibernate sneakily substitui as instâncias do `IDictionary`, `ISET` e `IList` com no-posturas de seus próprios implementações persistente de `IDictionary`, `ISET` ou `IList`. (Assim também ter cuidado ao usar == em suas coleções.)

```
Cat cat = new DomesticCat ();
Gato gatinho = new DomesticCat ();
...
Gatinhos ISET HashSet = new ();
kittens.Add (gatinho);
cat.Kittens = gatinhos;
Session.save (cat);
gatinhos = cat.Kittens // Ok, é uma coleção de gatinhos Set
HashSet hs = (HashSet) cat.Kittens; // Erro!
```

Coleções de obedecer as regras usuais para tipos de valor: não há referências compartilhadas, criado e apagados junto com conterridade ing. Devido ao modelo relacional subjacente, eles não suportam semântica de valor nulo; NHibernate faz não distinguir entre uma referência coleção nula e uma coleção vazia.

Coleções são automaticamente persistentes quando referenciado por um objeto persistente e automaticamente excluído quando sem referência. Se uma coleção é passada de um objeto persistente para outro, seus elementos podem ser movidos de uma tabela para outra. Você não deveria ter que se preocupar muito com nada disso. Basta usar coleções NHibernate é a mesma maneira que usa ordinária coleções NET, mas certifique-se de compreender a semântica de bidirecional associações (discutido mais tarde) antes de usá-los.

Casos coleção distinguem-se no banco de dados por uma chave estrangeira para a entidade proprietária. Esta chave estrangeira é referido como o **entrega das chaves**. A chave coleção é mapeada pelo `<key>` elemento.

Coleções podem conter quase qualquer outro tipo NHibernate, incluindo todos os tipos básicos, tipos personalizados, tipos de entidade e componentes. Esta é uma definição importante: um objeto em uma coleção pode ser tratada com "passar por valor" semântica (que, portanto, depende totalmente de o proprietário do conjunto) ou pode ser uma referência para outra entidade com um ciclo de vida própria. Coleções não podem conter outras coleções. O tipo contido é referido como o **tipo de elemento de coleta**. Elementos da coleção são mapeados por `<element>`, `<composite-element>`, `<one-to-many>`, `<many-to-many>` ou `<many-to-any>`. Os dois primeiros elementos do mapa com valores semânticos, a ouer três são usados para mapear associações entidade.

Todos os tipos de coleção, exceto `ISET` e saco de ter um índice coluna - uma coluna que mapeia para um array ou `IList` índice

ou `IDictionary` chave. O índice de um `IDictionary` pode ser de qualquer tipo básico, um tipo de entidade ou mesmo uma composição tipo (não pode ser uma coleção). O índice de um array ou lista é sempre do tipo `Int32`. Índices são mapeados nos-
ing `<index>`, `<index-many-to-many>`, `<composite-index>` ou `<index-many-to-any>`.

Há uma gama bastante de mapeamentos que podem ser gerados para as coleções, que abrangem relacional comum muitos modelos. Sugerimos que você experimente com a ferramenta de geração de esquema para obter uma idéia de como mapeamento de diversos declarações traduzem-se em tabelas de banco de dados.

6.2. Mapeamento de uma coleção

Coleções são declarados pelo `<set>`, `<list>`, `<map>`, `<bag>`, `<array>` e `<primitive-array>` elementos.
`<map>` é representativa:

```

<Mapa
    name = "propertyName"                                (1)
    table = "nome_tabela"                               (2)
    schema = "schéma_name"                            (3)
    lazy = "true | false"                             (4)
    inverse = "true | false"                           (5)
    cascade = "all | none | save-update | delete | all-delete-orphan" (6)
    sort = "indiferenciados | natural | comparatorClass" (7)
    fim-by = "column_name ASC | DESC"                (8)
    onde = "sql arbitrária onde a condição"          (9)
    fetch = "select | join"                           (10)
    batch-size = "N"                                  (11)
    access = "field | imóvel | ClassName"            (12)

>

    <Chave .... />
    Index <.... />
    Elemento <.... />
</ Mapa>

```

- (1) nome nome de propriedade de coleção
- (2) tabela (Opcional - o padrão é nome da propriedade) o nome da tabela de coleção (não utilizado para um-para-muitos associações)
- (3) esquema (Opcional) o nome de um esquema de tabela para substituir o esquema declarado no elemento raiz
- (4) preguiçoso (Opcional - valor default para `falso`) Permite a inicialização lenta (não utilizado para arrays)
- (5) inverso (Opcional - valor default para `falso`) Marcam esta coleção como o fim "inversa" de um bidirecional assocoation
- (6) cascata (Opcional - valor default para `nenhum`) Permitir operações em cascata para entidades filho
- (7) espécie (Opcional) especificar uma coleção ordenada, com `natural` ordem de classificação, ou uma classe de comparação dada
- (8) fim-by (Opcional) especificar uma coluna de tabela (ou colunas) que definem a ordem de iteração do `IDiction-`
`ary`, `ISET` ou saco, juntamente com um opcional `asc` ou `desc`
- (9) onde (Opcional) especifica um comando SQL arbitrárias `ONDE` condição a ser utilizado ao recuperar ou remover o coleção (útil se a coleção deve conter apenas um subconjunto dos dados disponíveis)
- (10) buscar (Opcional) Escolha entre outer-join e busca por seleção seqüencial.
- (11) batch-size (Opcional, o padrão é 1) Especifica um "tamanho de lote" para buscar preguiçosamente instâncias desta coleção.
- (12) acesso (Opcional - valor default para `propriedade`): O NHibernate estratégia deve utilizar para acessar a propriedade valor.

O mapeamento de uma `IList` ou matriz requer uma coluna de tabela em separado segurando o índice de array ou lista (o `euem` `foo [i]`). Se o seu modelo relacional não tem uma coluna de índice, por exemplo, se você estiver trabalhando com dados legados, use um `unordered ISET` em seu lugar. Este parece colocar as pessoas fora que assumem que `IList` deve ser apenas um mais conveniente maneira de acessar uma coleção não ordenada. Coleções NHibernate obedecer estritamente a semântica real anexado ao o `ISET`, `IList` e `IDictionary` interfaces. `IList` elementos não apenas reorganizar-se espontaneamente!

Por outro lado, as pessoas que planejava usar o `IList` para emular `SACO` semântica tem uma queixa legítima aqui. A bolsa é uma coleção não ordenada, não indexados, que podem conter o mesmo elemento várias vezes. O .NET framework coleções carece de uma `IBAG` interface, portanto você tem que imitá-la com um `IList`. NHibernate permite mapear propriedades do tipo `IList` ou `ICollection` com o `<bag>` elemento. Note que a semântica bag não são realmente parte do `ICollection` contrato e que o conflito realmente com a semântica da `IList` contrato (No entanto, você pode classificar o saco de forma arbitrária, discutido mais adiante neste capítulo).

Nota: Grande sacos NHibernate mapeado com `inverse = "false"` são ineficientes e devem ser evitados; NHibernate não pode criar, excluir ou atualizar linhas individualmente, porque não há nenhuma chave que pode ser usado para identificar um linha individual.

6.3. Coleções de Valores e muitos-para-muitos

A tabela da coleção é necessária para qualquer conjunto de valores e toda a coleção de referências a outras entidades mapeada como uma associação muitos-para-muitos (a semântica natural para uma coleção .NET). A tabela requer (Estrangeiros) de coluna de chave (s), coluna elemento (s) e, possivelmente, coluna do índice (s).

A chave estrangeira da tabela de coleção para a mesa da classe proprietária é declarada usando um `<key>` elemento.

```
<key column="column_name"/>
```

(1) `coluna` (Obrigatório): O nome da coluna de chave estrangeira.

Para coleções indexadas como mapas e listas, que exigem uma `<index>` elemento. Para listas, esta coluna contém se-inteiros seqüenciais numeradas de zero. Certifique-se que o seu índice realmente começa do zero se você tiver que lidar com dados legados. Para os mapas, a coluna pode conter quaisquer valores de qualquer tipo NHibernate.

```
<índice
    coluna = "column_name"                                (1)
    type = "typename"                                     (2)
/>
```

(1) `coluna` (Obrigatório): O nome da coluna mantendo os valores de índice de coleta.

(2) `tipo` (Opcional, o padrão é `Int32`): O tipo de índice da coleção.

Alternativamente, um mapa pode ser indexado por objetos do tipo entidade. Usamos o `<index-many-to-many>` elemento.

```
<índice-many-to-many
    coluna = "column_name"                                (1)
    class = "ClassName"                                   (2)
/>
```

(1) `coluna` (Obrigatório): O nome da coluna de chave estrangeira para os valores de índice de coleta.

(2) `classe` (Obrigatório): A classe de entidade usada como índice da coleção.

Para uma coleção de valores, usamos o `<element>` tag.

```
Elemento <
    coluna = "column_name"                                (1)
    type = "typename"                                     (2)
/>
```

(1) `coluna` (Obrigatório): O nome da coluna mantendo os valores dos elementos de coleta.

(2) `tipo` (Obrigatório): O tipo de elemento da coleção.

Uma coleção de entidades com sua própria tabela corresponde à noção relacional de muitos-para-muitos associação. A muitos a associação de muitos é o mapeamento mais natural de uma coleção. NET mas geralmente não é a melhor relacional modelo.

```
<Many-to-many
    coluna = "column_name"
    class = "ClassName"
    fetch = "join | escolher"
/>
```

(1)
(2)
(3)

- (1) **coluna** (Obrigatório): O nome da coluna da chave estranho.
- (2) **classe** (Obrigatório): O nome da classe associada.
- (3) **buscar** (Opcional, o padrão é juntar): Habilita o outer-join ou seqüencial selecionar fetching para esta associação.
Este é um caso especial, pois a busca ansiosa completo (em uma única instrução SELECT) de uma entidade e seus muitos-para-muitos relacionamento a outras entidades, que permitiria join fetching, não só da coleção em si, mas também com este atributo na <many-to-many> elemento aninhado.

Alguns exemplos, primeiro, um conjunto de strings:

```
<set name="Names" table="NAMES">
    <key column="GROUPID"/>
    <element column="NAME" type="String"/>
</ Set>
```

Um saco contendo números inteiros (com uma ordem determinada pela iteração fim-by atributo):

```
<bag name="Sizes" table="SIZES" order-by="SIZE ASC">
    <key column="OWNER"/>
    <element column="SIZE" type="Int32"/>
<Saco />
```

Um conjunto de entidades - neste caso, uma nota que muitos associação de muitos (que as entidades são objetos do ciclo de vida, cascade = "all"):

```
<array name="Foos" table="BAR_FOOS" cascade="all">
    <key column="BAR_ID"/>
    <index column="I" />
    <many-to-many column="FOO_ID" class="Eg.Foo, Eg"/>
</ Array>
```

Um mapa de índices de string para datas:

```
<MAP name="Holidays" table="holidays" schema="dbo" order-by="hol_name asc">
    <key column="id"/>
    <index column="hol_name" type="String"/>
    <element column="hol_date" type="Date"/>
</ Mapa>
```

A lista de componentes (discutido no próximo capítulo):

```
<enumerar name="CarComponents" table="car_components">
    <key column="car_id"/>
    <index column="posn"/>
    class="Eg.Car.CarComponent"> <composite-element
        <property name="Price" type="float"/>
        <property name="Type" type="Eg.Car.ComponentType, Eg"/>
        <property name="SerialNumber" column="serial_no" type="String"/>
    </ Elemento composto->
</ List>
```

6.4. Um-para-muitos Associações

Aum a associação de muitos vincula as tabelas de duas classes diretamente, sem tabela de coleção intervenientes. (Este implementa um um-para-muitos modelo relacional.) Este modelo relacional perde um pouco da semântica. NET coletações:

- Não valores nulos podem ser contidas em um dicionário definido, ou lista
- Uma instância da classe de entidade contida não pode pertencer a mais de uma instância da coleção
- Uma instância da classe de entidade contida pode não aparecer em mais de um valor do índice de coleta de

Uma associação de `Foo` para `Bar` requer a adição de uma coluna de chave e, possivelmente, uma coluna de índice para a tabela da classe de entidade contida, `Bar`. Estas colunas são mapeados usando o `<key>` e `<index>` elementos descritos acima.

O `<one-to-many>` tag indica uma associação de muitos.

```
<one-to-many class="ClassName"/>
```

(1) `classe`(Obrigatório): O nome da classe associada.

Exemplo:

```
<set name="Bars">
  <key column="foo_id"/>
  <one-to-many class="Eg.Bar, Eg"/>
</ Set>
```

Observe que o `<one-to-many>` elemento não precisa declarar todas as colunas. Também não é necessário especificar o tabela nome em qualquer lugar.

Nota muito importante: Se o `<key>` coluna de uma `<one-to-many>` Associação é declarada `NOT NULL`, NHibernate pode causar violações de restrição quando se cria ou atualiza a associação. Para evitar esse problema, você deve use uma associação bidirecional com o fim muitas valorizado (o conjunto ou saco) marcado como `inverse = "true"`. Veja a discussão das associações bidirecionais mais adiante neste capítulo.

6.5. Inicialização lenta

Coleções (exceto matrizes) pode ser inicializado preguiçosamente, o que significa que seu estado de carga da base de dados única quando o aplicativo precisa para acessá-lo. Inicialização acontece de forma transparente para o usuário para que a aplicação seria normalmente não precisa se preocupar com isso (na verdade, a inicialização lenta transparente é a principal razão NHibernate suas necessidades de implementações própria coleção). No entanto, se o aplicativo tenta algo assim:

```
s sessions.OpenSession = ();
ITransaction tx = sessions.BeginTransaction ();
U = usuário (User) s.Find ("de onde u usuário u.Name =?", UserName, NHibernateUtil.String) [0];
Permissões IDictionary = u.Permissions;
tx.Commit ();
s.close ();

int accessLevel = (int) Permissões ["contas"];           // Erro!
```

Poderia ser uma surpresa desagradável. Como a coleção de permissões não foi inicializado quando o `ISession` foi comprometida, a coleção nunca vai ser capaz de carregar seu estado. A correção é para mover a linha que lê a coleção para pouco antes do commit. (Há outras maneiras mais avançadas para resolver este problema, no entanto.)

Alternativamente, use uma coleção não-preguiçoso. Desde a inicialização lenta pode levar a erros como esse acima, a preguiça não é o padrão. No entanto, pretende-se que a inicialização lenta ser usado para quase todas as coleções, especialmente para coleções de entidades (por razões de eficiência).

Exceções que ocorrem enquanto preguiçosamente inicializar uma coleção estão envoltos em uma `LazyInitializationException`.

Declare uma coleção preguiçosa usando o opcional `preguiçoso` atributo:

```
<set name="Names" table="NAMES" lazy="true">
  <key column="group_id"/>
  <element column="NAME" type="String"/>
</ Set>
```

Em algumas arquiteturas de aplicações, especialmente quando o código que acessa os dados usando NHibernate, e os código que usa-lo estão em diferentes camadas da aplicação, ele pode ser um problema para garantir que o `ISession` é aberta quando uma coleção é inicializado. Existem duas maneiras básicas de lidar com esse problema:

- Em uma aplicação web-based, um manipulador de eventos pode ser usado para fechar a `ISession` apenas no final de um solicitação do usuário, uma vez que o processamento da visão é completa. É claro, isso coloca grandes exigências sobre a exactidão do tratamento de exceção de sua infra-estrutura de aplicativos. É de vital importância que o `ISession` é fechada ea transação terminada antes de retornar para o usuário, mesmo quando ocorre uma exceção durante renderização da view. O manipulador de eventos tem que ser capaz de acessar a `ISession` para esta abordagem. Nós re- Recomendo que a corrente `ISession` é armazenado no `HttpContext.Items` coleta (ver capítulo 1, Sec- ção 1.4, "Brincar com os gatos", para uma implementação de exemplo).
- Em uma aplicação com uma camada de negócios separada, a lógica de negócios deve "preparar" todas as coleções que serão necessários pela camada web antes de retornar. Isto significa que a camada de negócios deve carregar todos os dados e retornar todos os dados já inicializada para a camada de apresentação / web que é necessário para um caso de uso particular. Normalmente, o aplicativo chama `NHibernateUtil.Initialize ()` para cada coleção que serão necessários na camada web (Esta chamada deve ocorrer antes da sessão é fechada) ou recupera a coleção usando um NHibernate consulta com um `FETCH` cláusula.
- Você também pode anexar um objeto previamente carregado para uma nova `ISession` com `Update ()` ou `Lock ()` antes do acesso- ing uninitialzed coleções (ou outros proxies). NHibernate não pode fazer isso automaticamente, como seria introduce ad hoc operação semântica!

Você pode usar o `Filter ()` método do NHibernate `ISession` API para obter o tamanho de uma coleção sem inicializing-lo:

```
ICollection countColl = s.Filter (recolha, "select count (*)");
CountEn Ienumerator countColl.GetEnumerator = ();
countEn.MoveNext ();
int count = (int) countEn.Current;
```

`Filter` ou `CreateFilter ()` também são usados para recuperar de forma eficiente subconjuntos de um conjunto sem a necessidade de initialize toda a coleção.

6.6. Coleções classificadas

NHibernate `collections` implemented by `System.Collections.SortedList` e `Iesi.Collections.SortedSet`. Você deve especificar um comparador no arquivo de mapeamento:

```
<set name="Aliases" table="person_aliases" sort="natural">
  <key column="person"/>
```

```

<element column="name" type="String"/>
</ Set>

<MAP name="Holidays" sort="My.Custom.HolidayComparer, MyAssembly" lazy="true">
  <key column="year_id"/>
  <index column="hol_name" type="String"/>
  <element column="hol_date" type="Date"/>
</ Mapa>

```

Valores permitidos da `espécie` atributo são indiferenciados,natural eo nome de uma classe que implementa `System.Collections.IComparer`.

Se quiser que o próprio banco de dados para a ordem os elementos da coleção usar o `fim-by` atributo de `conjunto,saco` ou `mapa` mapeamentos. Este executa a ordenação na consulta SQL, não na memória.

Definir o `fim-by` atributo diz para usar NHibernate `ListDictionary` ou `ListSet` classe internamente para dicção-Áries e conjuntos, mantendo a ordem dos elementos. Note-se que operações de pesquisa sobre essas coleções são muito lenta se contiverem mais de alguns elementos.

```

<set name="Aliases" table="person_aliases" order-by="name asc">
  <key column="person"/>
  <element column="name" type="String"/>
</ Set>

<MAP name="Holidays" order-by="hol_date, hol_name" lazy="true">
  <key column="year_id"/>
  <index column="hol_name" type="String"/>
  <element column="hol_date" type="Date"/>
</ Mapa>

```

Note-se que o valor do `fim-by` atributo é uma ordenação SQL, e não um ordenamento HQL!

As associações podem ainda ser classificados por alguns critérios arbitrários em tempo de execução usando um `Filter ()`.

```
sortedUsers = s.Filter (group.Users, "order by this.Name");
```

6.7. Usando um `<idbag>`

Se você já abraçou totalmente a nossa visão de que chaves compostas são uma coisa ruim e que as entidades devem ter sintético identificadores (chaves substitutas), então você pode achar um pouco estranho que a muitos a muitas associações e coleções de valores que temos mostrado até agora todos os mapa para tabelas com chaves compostas! Agora, esse ponto é bastante discutível, uma tabela de associação pura não parecem se beneficiar muito de uma chave substituta (embora uma coleção de compostos values `pode`). No entanto, NHibernate oferece um recurso que permite mapear muitos de muitas associações e coleções de valores de uma tabela com uma chave substituta.

O `<idbag>` elemento permite mapear uma `Lista` (Ou Coleção) Com semântica saco.

```

<idbag name="Lovers" table="LOVERS" lazy="true">
  <collection-id column="ID" type="Int64">
    <generator class="hilo"/>
  </ Recolha-id>
  <key column="PERSON1"/>
  <many-to-many column="PERSON2" class="Eg.Person" fetch="join"/>
<Idbag />

```

Como você pode ver, um `<idbag>` tem um gerador de id sintético, assim como uma classe de entidade! Uma chave diferente substituto é atribuído a cada linha de coleta. NHibernate não prevê qualquer mecanismo para descobrir a chave substituta valor de uma linha particular, no entanto.

Note que o desempenho de atualização de um <idbag> é muito melhor do que um regular <bag>! NHibernate pode localizar linhas individual de forma eficiente e atualizar ou excluí-los individualmente, assim como uma lista de mapa, ou definir.

Na implementação atual, o nativo estratégia de geração de identificador não é suportado para <idbag> coleção identificadores.

6.8. Associações bidirecionais

Aassociação bidirecional permite a navegação de ambos os "lados" da associação. Dois tipos de bidirecional associação são suportados:

um-para-muitos

conjunto ou bolsa no valor de um final, de valor único para o outro

many-to-many

conjunto ou bolsa no valor de ambas as extremidades

Por favor note que NHibernate não suporta bidirecional um-para-muitos com uma coleção indexada (Mapa da lista, ou array) como o fim "muitos", você tem que usar um conjunto de mapeamento ou saco.

Você pode especificar uma associação many-to-many bidirecional simplesmente por duas mapeamento many-to-many associações para a mesa mesmo banco de dados e declarando um fim como **inverso** (Qual é a sua escolha). Aqui está um exemplo de uma associação many-to-many bidirecional de uma classe para trás **se** (Cada categoria pode ter muitos itens e cada item pode ser em muitas categorias):

```
<class name="NHibernate.Auction.Category, NHibernate.Auction">
    <id name="Id" column="ID"/>
    ...
    <bag name="Items" table="CATEGORY_ITEM" lazy="true">
        <key column="CATEGORY_ID"/>
        <many-to-many class="NHibernate.Auction.Item, NHibernate.Auction" column="ITEM_ID"/>
        <Saco />
    </ Class>

    <class name="NHibernate.Auction.Item, NHibernate.Auction">
        <id name="id" column="ID"/>
        ...

        <!-- Final inversa -->
        <bag name="categories" table="CATEGORY_ITEM" inverse="true" lazy="true">
            <key column="ITEM_ID"/>
            <many-to-many class="NHibernate.Auction.Category, NHibernate.Auction" column="CATEGORY_ID"/>
            <Saco />
        </ Class>
```

As alterações feitas apenas até o fim inverso da associação são **não persistiu**. Isto significa que NHibernate tem dois representações na memória para cada associação bidirecional, uma ligação de A para B e outro link de B para A. Este é mais fácil de entender se você pensar sobre o modelo de objeto. NET e como criar uma relação muitos-para-muitos relationship em C #:

```
category.Items.Add (item);           // A categoria agora "sabe" sobre o relacionamento
item.Categories.Add (categoria);     // O item agora "sabe" sobre o relacionamento

Session.update (item);              // Nenhum efeito, nada será salvo!
Session.update (categoria);         // A relação será salvo
```

O lado não-inversa é usada para salvar a representação em memória ao banco de dados. Obteríamos um-unne

INSERIR sária / UPDATE e provavelmente até mesmo uma violação de chave estrangeira se ambos provocaria mudanças! A mesma coisa é, naturalmente, também é verdade para bidirecional um-para-muitos.

Você pode mapear uma associação bidirecional um-para-muitos através de uma associação de um-para-muitos para a mesma tabela coluna (s) como many-to-one associação e anuncio o fim de muitos valores `inverse = "true"`.

```
<class name="Eg.Parent, Eg">
  <id name="Id" column="id"/>
  ...
  <set name="Children" inverse="true" lazy="true">
    <key column="parent_id"/>
    <one-to-many class="Eg.Child, Eg"/>
  </ Set>
</ Class>

<class name="Eg.Child, Eg">
  <id name="Id" column="id"/>
  ...
  <many-to-one name="Parent" class="Eg.Parent, Eg" column="parent_id"/>
</ Class>
```

Mapeamento de uma extremidade de uma associação com `inverse = "true"` não afeta o funcionamento das cascatas, ambos são divididos conceitos diferentes!

6.9. Associações ternárias

Existem duas abordagens possíveis para mapear uma associação ternária. Uma abordagem é usar elementos de componimento (discutido abaixo). Outra é usar um `IDictionary` com uma associação como o seu índice:

```
<MAP name="Contracts" lazy="true">
  <key column="employer_id"/>
  <index-many-to-many column="employee_id" class="Employee"/>
  <one-to-many column="contract_id" class="Contract"/>
</ Mapa>
```

```
<MAP name="Connections" lazy="true">
  <key column="node1_id"/>
  <index-many-to-many column="node2_id" class="Node"/>
  <many-to-many column="connection_id" class="Connection"/>
</ Mapa>
```

6.10. Associações heterogêneos

O `<many-to-any>` e `<index-many-to-any>` elementos prevêem verdadeiras associações heterogêneas. Estes elementos de mapeamento de trabalho da mesma forma como o `<any>` elemento - e também deve ser usado raramente, ou nunca.

6.11. Exemplos coleção

As seções anteriores são bastante confusas. Então, vamos olhar um exemplo. Esta classe:

```
using System;
using System.Collections;

namespace Eg

  Parent public class
  {
```

```

    id longo privado;
    privada crianças Iset;

    Id longa pública
    {
        get {return;}
        set {id = valor;}
    }

    Crianças privadas ISET
    {
        get {crianças return;}
        set {crianças = valor;}
    }

    ....
    ....
}
}

```

tem uma coleção de `Eg.Child` instâncias. Se cada criança tem, no máximo, um dos pais, o mapeamento mais natural é um one-para-muitos associação:

```

<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
assembly = "Eg" namespace = "Ex.:">

    name="Parent"> <class
        <id name="Id">
            <generator class="sequence"/>
        </ Id>
        <set name="Children" lazy="true">
            <key column="parent_id"/>
            <one-to-many class="Child"/>
        </ Set>
    </ Class>

    name="Child"> <class
        <id name="Id">
            <generator class="sequence"/>
        </ Id>
        <property name="name"/>
    </ Class>

</ Hibernate-mapping>

```

Isso mapeia para as definições de tabela a seguir:

```

criar tabela pai (Id bigint not null chave primária)
criar filho tabela (Id bigint not null chave primária, Nome varchar (255), bigint parent_id)
alter table filho acrescentar uma restrição de childfk0 (parent_id) pai referências

```

Se o pai é necessário, use uma associação bidirecional um-para-muitos:

```

<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
assembly = "Eg" namespace = "Ex.:">

    name="Parent"> <class
        <id name="Id">
            <generator class="sequence"/>
        </ Id>
        <set name="Children" inverse="true" lazy="true">
            <key column="parent_id"/>
            <one-to-many class="Child"/>
        </ Set>
    </ Class>

    name="Child"> <class

```

```

<id name="Id">
    <generator class="sequence"/>
</ Id>
<property name="name"/>
<many-to-one name="parent" class="Parent" column="parent_id" not-null="true"/>
</ Class>

</ Hibernate-mapping>

```

Observe o NOT NULL restrição:

```

criar tabela pai (Id bigint not null chave primária)
criar filho tabela (Id bigint not null
                    chave primária,
                    Nome varchar (255),
                    bigint parent_id não nulo)
alter table filho acrescentar uma restrição de childfk0 (parent_id) pai referências

```

Por outro lado, se uma criança pode ter vários pais, uma associação muitos-para-muitos é o caso:

```

<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
assembly = "Eg" namespace = "Ex.:">

    name="Parent"> <class
        <id name="Id">
            <generator class="sequence"/>
        </ Id>
        <set name="Children" lazy="true" table="childset">
            <key column="parent_id"/>
            <many-to-many class="Child" column="child_id"/>
        </ Set>
    </ Class>

    name="eg.Child"> <class
        <id name="Id">
            <generator class="sequence"/>
        </ Id>
        <property name="name"/>
    </ Class>

</ Hibernate-mapping>

```

Definições de tabela:

```

criar tabela pai (Id bigint not null chave primária)
criar filho tabela (Id bigint not null chave primária, nome varchar (255))
childset criar tabela (bigint parent_id não nulo,
                      bigint child_id não nulo,
                      chave primária (parent_id, child_id))
alter table childset restrição adicionar childsetfk0 (parent_id) pai referências
childset alter table add restrição childsetfk1 (child_id) filho referências

```

Capítulo 7. Mapeamento de Componentes

A noção de um **componente** é re-utilizado em vários contextos diferentes, para diferentes fins, ao longo NHibernate.

7.1. Objetos dependentes

Um componente é um objeto contido que é mantido como um tipo de valor, não uma entidade. O "componente" refere-se à noção orientada a objeto de composição (para nível de arquitetura de componentes). Por exemplo, você pode modelar de uma pessoa como esta:

```
public class Pessoa
{
    aniversário DateTime privado;
    nome Nome privado;
    chave private string;

    Chave cadeia pública
    {
        get {return chave;}
        set {chave = valor;}
    }

    Aniversário DateTime pública
    {
        get {aniversário return;}
        {aniversário = valor;} set
    }

    Nome pública
    {
        get {return nome;}
        set {nome = valor;}
    }
    .....
}

}
```

```
Nome public class
{
    caractere inicial;
    primeira corda;
    última seqüência de
    caracteres;

    public string Primeiro
    {
        get {return primeiro;}
        set {primeiro = valor;}
    }

    Última cadeia pública
    {
        get {return passado;}
        set {valor = passado;}
    }

    pública inicial de char
    {
        get {return inicial;}
        set {valor = inicial;}
    }
}
```

Agora `Nome` pode ser mantido como um componente de `Pessoa`. Note-se que `Nome` define os métodos getter e setter para suas propriedades persistentes, mas não precisa declarar nenhuma interface ou propriedades de identificação.

Nosso mapeamento NHibernate ficaria assim:

```
<class name="Eg.Person, Eg" table="person">
    <id name="Key" column="pid" type="string">
        <generator class="uuid.hex"/>
    </ Id>
    <property name="Birthday" type="date"/>
    <component> name="nome" class="Eg.Name, Eg"> <-! atributo de classe opcional ->
        <property name="Initial"/>
        <property name="First"/>
        <property name="Last"/>
    </ Component>
</ Class>
```

A tabela a pessoa teria as colunas `pid`, `Aniversário`, `Início`, `Primeiro` e `Passado`.

Como todos os tipos de valor, componentes não suportam referências compartilhadas. O nulo valor semântico de um componente são **ad hoc**. Ao recarregar o objeto que contém, NHibernate irá assumir que se todas as colunas de componentes são null, então todo o componente é nulo. Isso deve ser bom para a maioria dos propósitos.

As propriedades de um componente pode ser de qualquer tipo NHibernate (coleções, muitos-para-um, outros componentes, etc). Componentes aninhados devem **não** ser considerado um uso exótico. NHibernate se destina a suporta um modelo de objeto muito refinado.

O `<component>` elemento permite que um `<parent>` subelemento que mapeia uma propriedade da classe do componente como um referência de volta para a entidade que contém.

```
<class name="Eg.Person, Eg" table="person">
    <id name="Key" column="pid" type="string">
        <generator class="uuid.hex"/>
    </ Id>
    <property name="Birthday" type="date"/>
    <component> name="nome" class="Eg.Name, Eg">
        <parent name="NamedPerson"/> <-! referência de volta à Pessoa ->
        <property name="Initial"/>
        <property name="First"/>
        <property name="Last"/>
    </ Component>
</ Class>
```

7.2. Coleções de objetos dependentes

Coleções de componentes são suportadas (ex.: uma matriz do tipo `Nome`). Declare sua coleção componente por re-colocar o `<element>` tag com um `<composite-element>` tag.

```
<set name="SomeNames" table="some_names" lazy="true">
    <key column="id"/>
    <composite-element class="Eg.Name, Eg"> <-! atributo de classe requerida ->
        <property name="Initial"/>
        <property name="First"/>
        <property name="Last"/>
    </ Elemento composto>
</ Set>
```

Nota: se você definir um `ISET` de elementos compostos, é muito importante para implementar `Equals()` e `GetHashCode()` corretamente.

Elementos compostos podem conter componentes, mas não coleções. Se o seu elemento composto em si contém componentes, use o `<nested-composite-element>` tag. Este é um caso bastante exótico - uma coleção de COMPOENTES que se têm componentes. Nesta fase você deve estar se perguntando se um um-para-muitos associação é mais apropriado. Tente remodelar o elemento composto como uma entidade - mas note que, embora o modelo de objeto é o mesmo, o modelo relacional e persistência semântica ainda são um pouco diferentes.

Por favor, note que um mapeamento elemento composto não suporta nulo propriedades capazes de se você estiver usando uma `<set>`.

NHibernate tem que usar cada valor de colunas para identificar um registro ao excluir objetos (não há cobrança coluna de chave primária na tabela elemento composto), que não é possível com valores nulos. Você tem que quer use apenas as propriedades de não-nulo em um elemento composto ou escolher uma `<list>`, `<map>`, `<bag>` ou `<idbag>`.

Um caso especial de um elemento composto é um elemento composto com uma nested `<many-to-one>` elemento. Um mapa-de-ping como esta permite mapear colunas extras de uma tabela de associação de muitos-para-muitos para o elemento composto classe. O seguinte é uma associação muitos-para-muitos de `Ordem` para `Item` onde `PurchaseDate`, `Preço` e `Quantidade` são propriedades da associação:

```
<Nome da classe = "Ordem" .... >
...
<set name="PurchasedItems" table="purchase_items" lazy="true">
    <key column="order_id">
        class="Purchase"> <composite-element
            <property name="PurchaseDate"/>
            <property name="Price"/>
            <property name="Quantity"/>
            <many-to-one name="Item" class="Item"/> <-! atributo de classe é opcional ->
        </ Elemento composto->
    </ Set>
</ Class>
```

Mesmo associações ternárias (ou quaternária, etc) são possíveis:

```
<Nome da classe = "Ordem" .... >
...
<set name="PurchasedItems" table="purchase_items" lazy="true">
    <key column="order_id">
        class="OrderLine"> <composite-element
            <many-to-one name="PurchaseDetails class="Purchase"/>
            <many-to-one name="Item" class="Item"/>
        </ Elemento composto->
    </ Set>
</ Class>
```

Elementos compostos podem aparecer em pesquisas usando a mesma sintaxe que as associações a outras entidades.

7.3. Componentes como índices de `IDictionary`

O `<composite-index>` elemento permite mapear uma classe de componente como a chave de um `IDictionary`. Certifique-se de substituir `GetHashCode ()` e `Equals ()` corretamente na classe de componente.

7.4. Componentes como um identificador composto

Você pode usar um componente como um identificador de uma classe de entidade. Sua classe componente deve satisfazer determinados requisitos mentos:

- Deve ser `Serializable`.
- É preciso re-implementar `Equals ()` e `GetHashCode ()`, De forma consistente com a noção do banco de dados de composição

igualdade fundamental.

Você não pode usar um `IIdentifierGenerator` para gerar chaves compostas. Ao invés, a aplicação deve atribuir a sua identificadores próprios.

Uma vez que um identificador composto deve ser atribuído ao objeto antes de salvá-lo, não podemos usar `unsaved` valor do identificador para distinguir entre instâncias recém instanciado e instâncias salvas em uma sessão anterior.

Você pode implementar em vez `IInterceptor.IsUnsaved()` se você deseja usar `SaveOrUpdate()` ou em cascata salvar / Update. Como alternativa, você também pode definir o `unsaved` valor atributo em uma `<versão>` (Ou `<timestampe>`) Elemento para especificar um valor que indica uma nova instância transitória. Neste caso, a versão da entidade é usado em lugar do identificador (atribuído) e você não tem que implementar `IInterceptor.IsUnsaved()` si mesmo.

Usar o `<composite-id>` tag (mesmos atributos e elementos como `<component>`) No lugar de `<id>` para a declaração de uma classe identificador composto:

```
<class name="Foo" table="FOOS">
    <composite-id name="CompId" class="FooCompositeID">
        <key-property name="String"/>
        <key-property name="Short"/>
        <key-property name="Date" column="date_" type="Date"/>
    </ Composite-id>
    <property name="name"/>
    ...
</ Class>
```

Agora, todas as chaves estrangeiras na tabela `FOOS` também são compostas. Você deve declarar isto em seu mapeamentos para outras classes. Uma associação para `Foo` seria declarado assim:

```
<many-to-one name="Foo" class="Foo">
    <! - O atributo "class" é opcional, como de costume ->
    <column name="foo_string"/>
    <column name="foo_short"/>
    <column name="foo_date"/>
</ Many-to-one>
```

Este novo `<column>` tag também é usado por várias colunas tipos personalizados. Na verdade, é uma alternativa para o `coluna` atributo em todos os lugares. Uma coleção com elementos do tipo `Foo` usaria:

```
<set name="Foos">
    <key column="owner_id"/>
    <many-to-many class="Foo">
        <column name="foo_string"/>
        <column name="foo_short"/>
        <column name="foo_date"/>
    </ Many-to-many>
</ Set>
```

Por outro lado, `<one-to-many>`, Como de costume, não declara colunas.

Se `Foo` em si contém coleções, que também vai precisar de uma chave composta estrangeiros.

```
name="Foo"> <class
    ...
    ...
    <set name="Dates" lazy="true">
        <key> <! - uma coleção herda o tipo de chave composta ->
            <column name="foo_string"/>
            <column name="foo_short"/>
            <column name="foo_date"/>
        </ Key>
        <element column="foo_date" type="Date"/>
    </ Set>
```

```
</ Class>
```

7.5. Componentes dinâmicos

Você pode até mesmo mapear uma propriedade do tipo

IDictionary:

```
<dynamic-component name="UserAttributes">
  <property name="Foo" column="FOO"/>
  <property name="Bar" column="BAR"/>
  <many-to-one name="Baz" class="Baz" column="BAZ"/>
</ Componente dinâmico>
```

A semântica de um `<dynamic-component>` mapeamento são idênticos aos `<component>`. A vantagem deste tipo de mapeamento é a capacidade de determinar as propriedades reais do componente no momento da implantação, apenas editando o documento de mapeamento. (Manipulação de tempo de execução do documento de mapeamento também é possível, usando um DOM parser).

Capítulo 8. Mapeamento de Herança

8.1. As três estratégias

NHibernate suporta as três estratégias básicas de mapeamento de herança.

- tabela por hierarquia de classe
- tabela por subclasse
- tabela por classe concreta (algumas limitações)

É até possível usar diferentes estratégias de mapeamento para diferentes ramos da mesma hierarquia de herança, mas as mesmas limitações aplicáveis a aplicar a tabela de mapeamentos-per-classe concreta. NHibernate não suporta mixing <subclass> mapeamentos e <joined-subclass> mapeamentos dentro da mesma <class> elemento.

Suponha que temos uma interface `IPayment`, Com sua implementação `CreditCardPayment`, `CashPayment`, `ChequePayment`. O mapeamento table-per-hierarquia seria parecido com:

```
<class name="IPayment" table="PAYMENT">
    <id name="Id" type="Int64" column="PAYMENT_ID">
        <generator class="native"/>
    </ Id>
    <discriminator column="PAYMENT_TYPE" type="String"/>
    <property name="Amount" column="AMOUNT"/>
    ...
    <subclass name="CreditCardPayment" discriminator-value="CREDIT">
        ...
        <Subclasse />
    <subclass name="CashPayment" discriminator-value="CASH">
        ...
        <Subclasse />
    <subclass name="ChequePayment" discriminator-value="CHEQUE">
        ...
        <Subclasse />
    </ Class>
```

Exatamente uma tabela é necessária. Há uma grande limitação dessa estratégia de mapeamento: colunas declaradas pelo sub-classes não podem ter NOT NULL restrições.

Um mapeamento table-per-subclass ficaria assim:

```
<class name="IPayment" table="PAYMENT">
    <id name="Id" type="Int64" column="PAYMENT_ID">
        <generator class="native"/>
    </ Id>
    <property name="Amount" column="AMOUNT"/>
    ...
    <joined-subclass name="CreditCardPayment" table="CREDIT_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </ Joined-subclass>
    <joined-subclass name="CashPayment" table="CASH_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </ Joined-subclass>
    <joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        <key column="PAYMENT_ID"/>
        ...
    </ Joined-subclass>
```

```
</ Class>
```

Quatro mesas são obrigatórios. As três tabelas subclasses possuem associação de chave primária para a tabela da superclasse (então o modelo relacional é atualmente uma associação um-para-um).

Note que a implementação do NHibernate da subclasse table-per-não requer coluna discriminadora. Outro objeto / mappers relacional usa uma implementação diferente de tabela por subclasse, que requer um tipo discriminador coluna na tabela da superclasse. A abordagem adoptada pelo NHibernate é muito mais difícil de implementar, mas arguably mais correto do ponto de vista relacional.

Para qualquer uma dessas duas estratégias de mapeamento, uma associação polimórfica para `IPayment` é mapeada usando `<many-to-one>`.

```
<Many-to-one name = "Pagamento"
  coluna = "PAGAMENTO"
  class = "IPayment" />
```

A estratégia table-per-concrete-class é muito diferente.

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
  <id name="Id" type="Int64" column="CREDIT_PAYMENT_ID">
    <generator class="native"/>
  </ Id>
  <property name="Amount" column="CREDIT_AMOUNT"/>
  ...
</ Class>

<class name="CashPayment" table="CASH_PAYMENT">
  <id name="Id" type="Int64" column="CASH_PAYMENT_ID">
    <generator class="native"/>
  </ Id>
  <property name="Amount" column="CASH_AMOUNT"/>
  ...
</ Class>

<class name="ChequePayment" table="CHEQUE_PAYMENT">
  <id name="Id" type="Int64" column="CHEQUE_PAYMENT_ID">
    <generator class="native"/>
  </ Id>
  <property name="Amount" column="CHEQUE_AMOUNT"/>
  ...
</ Class>
```

Três tabelas foram necessárias. Repare que em nenhum lugar mencionamos o `IPayment` interface explicitamente. Em vez disso, fazer uso do NHibernate polimorfismo implícito. Notar também que as propriedades de `IPayment` são mapeados em cada das subclasses.

Neste caso, uma associação polimórfica para `IPayment` é mapeada usando `<any>`.

```
<Nome de nenhum = "Pagamento"
  meta-type = "classe"
  id-type = "Int64">
  <column name="PAYMENT_CLASS"/>
  <column name="PAYMENT_ID"/>
</ Qualquer>
```

Seria melhor se nós definirmos um `IUserType` como o `meta-type`, Para lidar com o mapeamento do tipo discriminador cordas para `IPayment` subclasse.

```
<= Nome qualquer "pagamento"
  meta-type = "PaymentMetaType"
  id-type = "Int64">
```

```
<column name="PAYMENT_TYPE"/> <! - CAIXA DE CRÉDITO, ou CHEQUE ->
<column name="PAYMENT_ID"/>
</ Qualquer>
```

Há uma coisa para ser observada sobre este mapeamento. Desde que as subclasses sejam mapeadas em seu próprio `<class>` elemento (e desde que `IPayment` é apenas uma interface), cada uma das subclasses pode ser facilmente parte de outro table-per-class ou table-per-subclass hierarquia de herança! (E você ainda pode usar pesquisas polimórficas em `O IPayment interface.`)

```
<class name="CreditCardPayment" table="CREDIT_PAYMENT">
    <id name="Id" type="Int64" column="CREDIT_PAYMENT_ID">
        <generator class="native"/>
    </ Id>
    <discriminator column="CREDIT_CARD" type="String"/>
    <property name="Amount" column="CREDIT_AMOUNT"/>
    ...
    <subclass name="MasterCardPayment" discriminator-value="MDC"/>
    <subclass name="VisaPayment" discriminator-value="VISA"/>
</ Class>

<class name="NonelectronicTransaction" table="NONELECTRONIC_TXN">
    <id name="Id" type="Int64" column="TXN_ID">
        <generator class="native"/>
    </ Id>
    ...
    <joined-subclass name="CashPayment" table="CASH_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="Amount" column="CASH_AMOUNT"/>
        ...
    </ Joined-subclass>
    <joined-subclass name="ChequePayment" table="CHEQUE_PAYMENT">
        <key column="PAYMENT_ID"/>
        <property name="Amount" column="CHEQUE_AMOUNT"/>
        ...
    </ Joined-subclass>
</ Class>
```

Mais uma vez, nós não mencionamos `IPayment` explicitamente. Se executar uma consulta contra a `IPayment` interface - para exemplo, de `IPayment` - NHibernate retorna automaticamente instâncias de `CreditCardPayment` (E seus sub-classes, desde que elas também implementem `IPayment`), `CashPayment` e `ChequePayment` mas não os casos de `NonelectronicTransaction`.

8.2. Limitações

NHibernate assume que uma associação de mapas a exatamente uma coluna de chave estrangeira. Múltiplas associações por foreign-estrangeiras fundamentais são toleradas (talvez seja necessário especificar `inverse = "true"` ou `insert = update = "false" = "false"`). Mas não há forma de mapear qualquer associação a várias chaves estrangeiras. Isso significa que:

- quando uma associação é modificada, é sempre a mesma chave estrangeira que é atualizado
- quando uma associação é trazida de forma tardia, uma consulta de banco de dados único é usado
- quando uma associação é buscada com avidez, pode ser obtida usando uma única associação externa

Em particular, isso implica que polimórficas de um-para-muitos para classes mapeadas usando a tabela-per-concrete-class estratégia são **não suportado**. (Buscando essa associação exigiria várias consultas ou multiple junta.)

A tabela a seguir mostra as limitações de table-per-concrete-class mapeamentos, e do polimorfismo implícito,

em NHibernate.

Tabela 8.1. Características de mapeamentos de herança

Herdar-dade estratégia	Poli-mórfica muitos- para-um	Poli-mórfica um-para-um	Poli-mórfica um para-muitos	Poli-mórfica muitos- para-muitos	Poli-mórfica Load () / Get ()	Poli-mórfica márvica consultas	Poli-mórfica junta-se
table-per- classe hierarquia	<Many-to-o ne>	<One-to-on e>	<One-to-ma ny>	<Many-to-m qualquer>	s.Get tipo (de Pay- de (IPaymen t), id)	da Ordem o participar o.payment p	
table-per- subclasse	<Many-to-o ne>	<One-to-on e>	<One-to-ma ny>	<Many-to-m qualquer>	s.Get tipo (de iPay- de (IPaymen t), id)	da Ordem o participar o.Payment p	
table-<any> per- concreto classe (Implícita polimorfismo)			não suppor- ted	não suppor- ted	usar uma consulta de Pay- mento p		não suppor- ted

Capítulo 9. Manipulando dados persistentes

9.1. Criando um objeto persistente

Um objeto (instância da entidade) ou é transitório ou persistente com relação a um determinado `ISession`. Recém-instantiados objetos são, naturalmente, transitórios. A sessão oferece serviços para a poupança (ou seja, persistindo) transitória instâncias:

```
DomesticCat fritz = new DomesticCat ();
fritz.Color = Color.Ginger;
fritz.Sex = 'M';
fritz.Name = "Fritz";
longo generatedId = (long) sess.Save (fritz);
```

```
DomesticCat pk = new DomesticCat ();
pk.Color = Color.Tabby;
pk.Sex = 'F';
pk.Name = "PK";
pk.Kittens = new HashSet ();
pk.AddKitten (fritz);
sess.Save (pk, 1234L);
```

O único argumento-`Save ()` gera e atribui um identificador único para `fritz`. A forma de dois argumentos-atenta persistir `pk` usando o identificador especificado. Em geral, desencorajar o uso da forma de dois argumentos já ele pode ser usado para criar chaves primárias com significado de negócios.

Objetos associados podem ser feitas persistentes em qualquer ordem que você gosta menos que você tenha um `NOT NULL` restrição sobre um coluna de chave estrangeira. Nunca há um risco de violar restrições de chaves estrangeiras. No entanto, você pode violar uma `NOT NULL` restrição se `Save ()` os objetos na ordem errada.

9.2. Carregando um objeto

O `Load ()` métodos de `ISession` dar-lhe uma maneira de recuperar uma instância persistente se você já sabe a sua identificadora. Uma versão leva um objeto de classe e carregar o estado em um objeto recém instaciado. A segunda versão permite que você forneça uma instância em que o estado vai ser carregado. A forma que toma uma instância só é útil em situações especiais (exemplo DIY pooling, etc)

```
Gato fritz = (Cat) sess.Load (typeof (Cat), generatedId);
```

```
longo pkId = 1234;
DomesticCat pk = (DomesticCat) sess.Load (typeof (Cat), pkId);
```

```
Cat cat = new DomesticCat ();
Estado / pk carga / do em gato
sess.Load (cat, pkId);
Gatinhos ISET = cat.Kittens;
```

Note-se que `Load ()` irá lançar uma exceção irrecuperável se não houver nenhuma linha de banco de dados correspondente. Se a classe é mapeada com um proxy, `Load ()` retorna um objeto que é um proxy não inicializado e na verdade não atingiu a database até que você invocar um método do objeto. Este comportamento é muito útil se você deseja criar uma associação a um objeto sem realmente carregá-lo do banco de dados.

Se você não é certo que uma linha correspondente existe, você deve usar o `Get ()` método, que atinge o banco de dados imediatamente e retorna nulo se não houver nenhuma linha correspondente.

```
Cat = gato (Cat) sess.Get (typeof (Cat), id);
if (gato == null) {
    cat = new Gato ();
    sess.Save (cat, id);
}
retorno gato;
```

Você também pode carregar um objeto usando um SQL `SELECT ... FOR UPDATE`. Consulte a próxima seção para uma discussão sobre

NHibernate `LockModeS`.

```
Cat = gato (Cat) sess.Get (typeof (Cat), id, LockMode.Upgrade);
```

Note-se que todas as instâncias associadas ou coleções contidas são **não** selecionado `FOR UPDATE`.

É possível re-carregar um objeto e todas as suas coleções a qualquer momento, usando o `Refresh ()` método. Isso é útil quando os gatilhos do banco de dados são usados para inicializar algumas das propriedades do objeto.

```
sess.Save (cat);
sess.Flush (); / force / SQL INSERT
sess.Refresh (cat); / / re-leitura do Estado (após o gatilho é executado)
```

9.3. Consultando

Se você não sabe o identificador (s) do objeto (s) que você está procurando, utilize o `Find ()` métodos de `ISession`. NHibernate suporta um objeto simples, mas poderosa linguagem de consulta orientada.

```
Gatos IList sess.Find = (
    "De Cat como cat = cat.Birthdate onde?",
    data,
    NHibernateUtil.Date
);

Mates IList sess.Find = (
    "Select companheiro de Gato como o gato se juntar cat.Mate como
mate" +
    "Where = cat.name?",
    nome,
    NHibernateUtil.String
);

Gatos IList = sess.Find ("de Cat como gato onde cat.Mate.Birthdate é nulo");

IList MoreCats sess.Find = (
    "De Cat como gato onde" +
    "Cat.Name = 'Fritz' ou cat.id =? Ou cat.id =?",
    novo objeto [] {id1, id2},
    nova ITipo [] {NHibernateUtil.Int64, NHibernateUtil.Int64}
);

Mates IList sess.Find = (
    "De Cat como cat = cat.Mate onde?",
    izi,
    NHibernateUtil.Entity (typeof (Cat))
);

Problemas IList sess.Find = (
    "A partir de GoldFish como peixes" +
    "Onde fish.Birthday> fish.Deceased ou fish.Birthday é nulo"
);
```

O segundo argumento para `Find ()` aceita um objeto ou array de objetos. O terceiro argumento aceita um NHibernate tipo ou variedade de tipos de NHibernate. Estes tipos de dados são usados para vincular os objetos dado ao ?lugar-query

titulares (que mapa para parâmetros de entrada de um ADO.NET `IDbCommand`). Assim como no ADO.NET, você deve usar este mecanismo obrigatório em preferência a manipulação de strings.

O `NHibernateUtil` classe define uma série de métodos estáticos e constantes, proporcionando acesso à maioria dos built-in tipos, como instâncias de `NHibernate.Type.IType`.

Se você espera sua consulta para retornar um número muito grande de objetos, mas você não espera usá-los de tudo, você poderia obter um melhor desempenho do `Enumeráveis ()` métodos, que retornam um `System.Collections.IEnumerable`. O iterador irá carregar objetos sob demanda, utilizando os identificadores retornados por uma consulta SQL inicial ($n + 1$ seleção total).

```
// Busca ids
IQueryable en sess.Enumerable = ("de fim eg.Qux q por q.Likeliness");
foreach (Qux qux em pt)
{
    / Algo / que não poderia expressar na consulta
    if (qux.CalculateComplicatedAlgorithm ()) {
        / Não precisa processar o resto
        break;
    }
}
```

O `Enumeráveis ()` método também funciona melhor se você esperar que muitos dos objetos já estão carregados e armazenada em cache pela sessão, ou se os resultados da consulta contêm os mesmos objetos muitas vezes. (Quando não há dados em cache ou repetido, `Find ()` é quase sempre mais rápido.) Heres um exemplo de uma consulta que deve ser chamado utilizando `Enumeráveis ()`:

```
IQueryable en = sess.Enumerable (
    "Select cliente, produto" +
    "A partir do cliente do cliente," +
    "Produto do produto" +
    "Join compra customer.Purchases" +
    "Onde o produto = purchase.Product"
);
```

Chamando a consulta anterior utilizando `Find ()` retornaria um conjunto de resultados muito grande ADO.NET contendo o mesmo vezos muitos dados.

Consultas NHibernate vezos retornam tuplas de objetos, caso em que cada tupla é retornada como um array:

```
FoosAndBars IQueryable = sess.Enumerable (
    "Select foo, bar de Foo foo, bar Bar" +
    "Onde bar.Date = foo.Date"
);
foreach (object [tupla] em foosAndBars)
{
    Foo foo = tupla [0]; bar Bar = tupla [1];
    ....
}
```

9.3.1. Consultas escalares

Consultas podem especificar uma propriedade de uma classe na `selecionar` cláusula. Eles podem até chamar funções de agregação SQL.

Propriedades ou agregados são considerados "escalar" os resultados.

```
Resultados IQueryable = sess.Enumerable (
    "Select cat.Color, min (cat.Birthdate), count (gato) de gato do gato" +
    "Grupo de cat.Color"
);
foreach (object [linha] nos resultados)
```

```
{
    Tipo de cor = linha (cor) [0];
    Mais antiga DateTime = linha (DateTime) [1];
    int count = (int) row [2];
    ....
}
```

```
IEnumerable en = sess.Enumerable (
    "Select cat.Type, cat.Birthdate, cat.Name de gato DomesticCat"
);
```

```
IList list = sess.Find (
    "Select cat.Mate.Name gato, de gato DomesticCat"
);
```

9.3.2. A interface IQuery

Se você precisar especificar limites em cima de seu conjunto de resultados (o número máximo de linhas que você deseja recuperar e / ou

a primeira linha que você deseja recuperar), você deve obter uma instância de `NHibernate.IQuery`:

```
IQuery sess.CreateQuery q = ("de gato DomesticCat");
q.SetFirstResult (20);
q.setMaxResults (10);
Gatos IList q.List = ();
```

Você pode até mesmo definir uma consulta nomeada no documento de mapeamento. (Lembre-se de usar um `CDATA` seção se o seu consulta contém caracteres que poderiam ser interpretados como marcação.)

```
<query name="Eg.DomesticCat.by.name.and.minimum.weight"> <! [CDATA [
    de Eg.DomesticCat como gato
    = onde cat.Name?
    e cat.Weight?
]]> </ Query>
```

```
IQuery q = sess.GetNamedQuery ("Eg.DomesticCat.by.name.and.minimum.weight");
q.SetString (0, nome);
q.SetInt32 (1, minWeight);
Gatos IList q.List =();
```

A interface de consulta suporta o uso de parâmetros nomeados. Parâmetros nomeados são identificadores da forma : `Nome` na cadeia de consulta. Existem métodos de `IQuery` para valores de ligação com o nome ou parâmetros posicionais. Números NHibernate parâmetros de zero. As vantagens de parâmetros nomeados são:

- parâmetros nomeados são insensíveis à ordem em que ocorrem na sequência de consulta
- eles podem ocorrer várias vezes na mesma consulta
- eles são auto-documentando

```
/ / Chamado parâmetro (de preferência)
IQuery sess.CreateQuery q = ("de gato DomesticCat onde cat.Name =: nome");
q.SetString ("nome", "Fritz");
Gatos IEnumerable = q.Enumerable ();
```

```
/ / Parâmetro posicional
IQuery sess.CreateQuery q = ("de gato DomesticCat onde cat.Name =?");
q.SetString (0, "Izi");
Gatos IEnumerable = q.Enumerable ();
```

```
/ / Chamada lista de parâmetros
IList nomes = new ArrayList ();
```

```

names.Add ("Izi");
names.Add ("Fritz");
IQuery sess.CreateQuery q = ("de gato DomesticCat onde cat.Name em (: namesList)");
q.setParameterList ("namesList", nomes);
Gatos IList q.List = ();

```

9.3.3. Coleções de filtragem

Uma coleção filtro é um tipo especial de consulta que pode ser aplicado a uma coleção persistente ou array. A consulta string pode se referir a este, Significando que o elemento da coleção atual.

```

BlackKittens ICollection session.Filter = (
    pk.Kittens ", onde this.Color =?", Color.black, NHibernateUtil.Enum (typeof (Cor))
);

```

A coleção retornada é considerado um saco.

Observe que os filtros não requerem um a partir de cláusula (embora possam ter uma se necessário). Filtros não são limitados de devolver os elementos da coleção si.

```

BlackKittenMates ICollection session.Filter = (
    pk.Kittens ", selecione this.Mate onde this.Color = Eg.Color.Black"
);

```

9.3.4. Consultas critérios

HQL é extremamente poderoso, mas algumas pessoas preferem construir consultas dinamicamente, usando uma API orientada a objeto, em vez de incorporar strings em seu código. NET. Para essas pessoas, NHibernate fornece uma interface intuitiva `ICriteria` API de consulta.

```

ICriteria crit = session.CreateCriteria (typeof (Cat));
crit.Add (Expression.Eq ("cor", Eg.Color.Black));
crit.SetMaxResults (10);
Gatos IList crit.List = ();

```

Se você estiver confortável com o SQL-como sintaxe, esta é talvez a maneira mais fácil para começar com o NHibernate. Esta API também é mais extensível que o HQL. Aplicativos podem fornecer suas próprias implementações do `ICriterion` interface.

9.3.5. Consultas em SQL nativo

Você pode expressar uma consulta no SQL, usando `CreateSQLQuery ()`. Você deve colocar aliases SQL em chaves.

```

Gatos IList session.CreateSQLQuery = (
    "SELECT {cat *} FROM CAT {cat} onde ROWNUM <10",
    "Cat",
    typeof (Cat)
.) Lista ();

```

```

Gatos IList session.CreateSQLQuery = (
    "SELECT {cat}. ID AS {cat.Id}, {cat} SEX. AS {cat.Sex}, " +
    "{Cat} MATE. Cat.Mate AS {}, {cat} subclasse. AS {} cat.class, ..." +
    "FROM CAT {cat} onde ROWNUM <10",
    "Cat",
    typeof (Cat)
). List ()

```

Consultas SQL pode conter parâmetros nomeados e posicionais, assim como consultas NHibernate.

9.4. Objetos de atualização

9.4.1. Atualização na mesma ISession

Transacional instâncias persistentes (Ou seja, objetos carregados, salvos, criados ou consultado pela `ISession`) Pode ser manipulated pela aplicação e quaisquer alterações ao estado persistente será mantido quando o `ISession` é lavada (Discutido mais tarde neste capítulo). Assim, a maneira mais simples para atualizar o estado de um objeto é a `Load ()` ele, e depois manipulá-lo diretamente, enquanto o `ISession` está aberta:

```
DomesticCat cat = (DomesticCat) sess.Load (typeof (Cat), 69L);
cat.Name = "PK";
sess.Flush (); / / alterações do gato são automaticamente detectados e persistiu
```

Às vezes, este modelo de programação é ineficiente, pois exigiria tanto uma SQL SELEÇÃO (Para carregar um object) e um SQL ATUALIZAÇÃO (Para preservar seu estado atualizado) na mesma sessão. Portanto NHibernate oferece uma abordagem ternate.

9.4.2. Atualizando objetos destacados

Muitas aplicações precisam recuperar um objeto em uma transação, enviá-lo para a camada de interface do usuário para a manipulação, em seguida, salvar as alterações em uma nova transação. (Aplicativos que usam este tipo de abordagem em uma alta concorrência de ambiente geralmente usam dados versionados para assegurar o isolamento de transação.) Esta abordagem requer um pouco diferente modelo de programação ao descrito na última seção. NHibernate suporta este modelo, fornecendo a método `Session.update ()`.

```
/ / Na primeira sessão
Cat = gato (Cat) firstSession.Load (typeof (Cat), catid);
Cat Cat potentialMate = new ();
firstSession.Save (potentialMate);

/ / Em um maior nível de aplicação
cat.Mate = potentialMate;

/ / Mais tarde, em uma nova sessão
secondSession.Update (cat); cat / update /
secondSession.Update (mate); companheiro / update /
```

Se o `Gato` com identificador `catid` já tinha sido carregado pelo `secondSession` quando a aplicação tentou atualizar ele, uma exceção teria sido lançada.

A aplicação deve individualmente `Update ()` casos transitórios acessível a partir do exemplo dado transitória se e apenas se ele quer que seu estado também atualizado. (Exceto para objetos do ciclo de vida, discutido mais tarde).

Hibernate usuários têm solicitado um método de uso geral que seja salva uma instância transiente, gerando um novo identificador ou atualizar o estado persistente associado com seu identificador atual. O `SaveOrUpdate ()` método agora implementa esta funcionalidade.

NHibernate distingue "novo" (não salvas) exemplos de "existentes" (salvas ou carregadas em uma sessão anterior), em posições pelo valor do seu identificador (ou versão, ou timestamp) propriedade. O `unsaved valor` atributo do `<id>` (Ou `<versão>`Ou `<timestamp>`) Mapeamento especifica que os valores devem ser interpretados como representando uma Instância "novo".

```
<id name="Id" type="Int64" column="uid" unsaved-value="0">
  <generator class="hilo"/>
</ Id>
```

Os valores permitidos de `unsaved valor` são:

- `qua` Sempre guardar
- `nenhum` - Sempre atualização
- `nulo` - Economias quando identificador é nulo
- `valor` identificador válido - salvo quando identificador é nulo ou o valor dado
- `undefined` - Se estiver definido para `versão` ou `timestamp`, Em seguida, verificar identificador é usado

Se `unsaved valor` não é especificado para uma classe, NHibernate irá tentar adivinhar-lo, criando uma instância da classe usando o construtor sem argumentos e ler o valor da propriedade da instância.

```
/ / Na primeira sessão
Cat = gato (Cat) firstSession.Load (typeof (Cat), catid);

/ / Em um maior nível de aplicação
Cat = gato companheiro new ();
cat.Mate companheiro =;

/ / Mais tarde, em uma nova sessão
secondSession.SaveOrUpdate (cat);           / / Atualiza estado existente (gato tem um ID de non-null)
secondSession.SaveOrUpdate (mate);           / / Salva a nova instância (mate tem um id null)
```

O uso da semântica de `SaveOrUpdate ()` parece ser confuso para novos usuários. Em primeiro lugar, contanto que você está não tentar usar instâncias de uma sessão em outra sessão nova, você não deve precisar usar `Update ()` ou `SaveOrUpdate ()`. Algumas aplicações toda nunca vai usar um desses métodos.

Geralmente `Update ()` ou `SaveOrUpdate ()` são usados no seguinte cenário:

- o aplicativo carrega um objeto na primeira sessão
- o objeto é passado para a camada de interface do usuário
- algumas modificações são feitas para o objeto
- o objeto é passado de volta para a lógica camada de negócios
- a aplicação persiste estas modificações chamando `update ()` em uma segunda sessão

`SaveOrUpdate ()` faz o seguinte:

- se o objeto já está persistente nesta sessão, não fazer nada
- se o objeto não tem nenhuma propriedade identificador, `save ()` ele
- se o identificador do objeto corresponde aos critérios especificados pelo `unsaved valor`, `save ()` ele
- se o objeto é versionado (`versão` ou `timestamp`), Então a versão terá precedência para verificar identificador, a menos que as versões `unsaved-value = "undefined"` (Valor padrão)
- se outro objeto associado com a sessão tem o mesmo identificador, lançar uma exceção

O último caso pode ser evitado pelo uso de `SaveOrUpdateCopy (Object o)`. Esse método copia o estado do dado objeto para o objeto persistente com o mesmo identificador. Se não houver nenhuma instância persistente atualmente associado com a sessão, que será carregado. O método retorna a instância persistente. Se o exemplo dado é não salvos ou não existe no banco de dados, NHibernate irá salvá-lo e devolvê-lo como uma instância recém-persistente. Caso contrário, o exemplo dado não se torne associada com a sessão. Na maioria das aplicações com objetos individual, você necessidade ambos os métodos, `SaveOrUpdate ()` e `SaveOrUpdateCopy ()`.

9.4.3. Recolocar objetos destacados

O `Lock()` método permite que o aplicativo reassocie um objeto modificado com uma nova sessão.

```
// Apenas reassocie:  
sess.Lock (fritz, LockMode.NONE);  
// Fazer uma verificação de versão, então reassocie:  
sess.Lock (izi, LockMode.READ);  
// Fazer uma verificação de versão, usando SELECT ... FOR UPDATE, então  
reassocie:  
sess.Lock (pk, LockMode.Upgrade);
```

9.5. Exclusão de objetos persistentes

`ISession.Delete()` irá mover estado de um objeto do banco de dados. Claro, o aplicativo pode ainda manter uma referência a ele. Por isso é melhor pensar em `Delete()` como fazer uma instância persistente transitória.

```
sess.Delete (cat);
```

Você também pode excluir muitos objetos de uma só vez, passando uma string de consulta para NHibernate `Delete()`.

Agora você pode excluir objetos em qualquer ordem que desejar, sem risco de estrangeiros violações restrição de chave. É claro, ainda é possível violar uma `NOT NULL` restrição em uma coluna de chave estrangeira, eliminando objetos no errado ou-der.

9.6. Resplendor

De tempos em tempos o `ISession` irá executar os comandos SQL necessários para sincronizar o ADO.NET conexão do estado com o estado de objetos mantidos na memória. Este processo, `flush`, ocorre por padrão no seguinte pontos

- de algumas invocações de `Find()` ou `Enumeráveis()`
- a partir de `NHibernate.ITransaction.Commit()`
- a partir de `ISession.Flush()`

As instruções SQL são emitidas na seguinte ordem

1. todas as inserções entidade, na mesma ordem os objetos correspondentes foram salvos com `ISession.Save()`
2. todas as atualizações entidade
3. todas as exclusões coleção
4. toda a coleta de exclusões elemento, atualizações e inserções
5. todas as inserções coleção
6. todas as exclusões entidade, na mesma ordem os objetos correspondentes foram excluídos usando `ISession.Delete()`

(Uma exceção é que os objetos usando nativo Geração de ID são inseridas quando eles são salvos.)

Exceto quando explicitamente você `Flush()`, Não há absolutamente nenhuma garantia sobre quando o `Sessão` executa o JDBC chamadas, apenas os `ordem` em que são executados. No entanto, NHibernate não garante que o `ISession.Find(...)` métodos nunca retornam dados antigos, nem eles vão retornar os dados errados.

É possível alterar o comportamento padrão para que lave ocorre com menos freqüência. O `FlushMode` classe define três modos diferentes: só lave na hora da confirmação (e somente quando o NHibernate `ITransaction` API é usado), lave automaticamente usando a rotina explicada, ou nunca lave a menos `Flush()` é chamado explicitamente. O último modo é útil para longas unidades de execução do trabalho, onde um `ISession` é mantido aberto e desligado por um longo tempo (Veja Seção 10.4, "controle de simultaneidade otimista").

```

sess sf.OpenSession = ();
ITransaction tx = sess.BeginTransaction ();
sess.FlushMode FlushMode.Commit = / / permitir consultas para retornar estado obsoleto
Cat izi = (Cat) sess.Load (typeof (Cat), id);
izi.Name = "iznizi";
/ / Execute algumas consultas ....
sess.Find ("from Cat cat como junção externa esquerda cat.Kittens gatinho");
/ / Altere para izi não é lavada!
...
tx.Commit () / / descarga ocorre

```

9.7. Encerrando uma Sessão

Terminar uma sessão envolve quatro fases distintas:

- lave a sessão
- confirmar a transação
- fechar a sessão
- lidar com exceções

9.7.1. Rubor da Sessão

Se acontecer de você estar usando a `ITransaction` API, você não precisa se preocupar com esta etapa. Será realizada implicitamente quando a transação é confirmada. Caso contrário, você deve chamar `Session.flush ()` para garantir que todos alterações serão sincronizadas com o banco de dados.

9.7.2. Confirmar a transação de banco de dados

Se você estiver usando o NHibernate `ITransaction` API, esta parece ser:

```
tx.Commit (); / / esvazia a sessão e confirmar a transação
```

Se você estiver gerenciando transações ADO.NET-se que você deve manualmente `Commit ()` a transacções ADO.NET ção.

```
sess.Flush ();
currentTransaction.Commit ();
```

Se você se decidir **não** para confirmar as alterações:

```
tx.Rollback (); / / Rollback da transação
```

ou:

```
currentTransaction.Rollback ();
```

Se você reverter a transação deve ser fechado imediatamente e descartar a sessão atual para garantir que Estado interno do NHibernate é consistente.

9.7.3. Fechando a ISession

Uma chamada para `ISession.Close ()` marca o fim de uma sessão. A principal implicação `close ()` é que o ADO.NET conexão será abandonada pela sessão.

```
tx.Commit ();
sess.Close ();
```

```
sess.Flush ();
currentTransaction.Commit ();
sess.Close ();
```

Se você desde que a sua própria conexão, `Close ()` retorna uma referência a ele, assim você pode fechá-la manualmente ou devolvê-lo para a piscina. Caso contrário `Close ()` retorna para a piscina.

9.8. Tratamento de exceção

Usar NHibernate pode levar a exceções, geralmente `HibernateException`. Essa exceção pode ter um interior aninhadas exceção (a causa), use o `InnerException` propriedade para acessá-lo.

Se o `ISession` lança uma exceção você deve imediatamente reverter a transação, chame `ISession.Close ()` e descartar a `ISession` exemplo. Certos métodos de `ISession` vontade não deixar a sessão em um estado consistente.

Para exceções lançadas pelo provedor de dados enquanto interage com o banco de dados, NHibernate vai embrulhar o erro em uma instância do `ADOException`. A exceção subjacente é acessível pelo telefone `ADOException.InnerException`.

A maneira de se manipular exceção seguinte mostra o caso típico em aplicações NHibernate:

```
usando (sess ISession factory.OpenSession = ())
usando (ITransaction tx sess.BeginTransaction = ())
{
    / / Fazer algum trabalho
    ...
    tx.Commit ();
}
```

Ou, quando manualmente gerenciar transações ADO.NET:

```
Sess ISession factory.openSession = ();
tentar
{
    / / Fazer algum trabalho
    ...
    sess.Flush ();
    currentTransaction.Commit ();
}
catch (Exception e)
{
    currentTransaction.Rollback ();
    throw;
}
finalmente
{
    sess.Close ();
}
```

9.9. Ciclos de vida e gráficos de objetos

Para salvar ou atualizar todos os objetos em um gráfico de objetos associados, você deve

- `Save (), SaveOrUpdate ()` ou `Update ()` cada objeto individual ou

- mapear objetos associados com `cascade = "all"` ou `cascade = "save-update"`.

Da mesma forma, para excluir todos os objetos em um gráfico, seja

- ~~Delete~~ cada objeto individual ou
- mapear objetos associados com `cascade = "all",cascade = "all-delete-orphan"` ou `cascade = "delete"`.

Recomendação:

- Se a vida útil do objeto filho é limitada pelo tempo de vida do objeto pai torná-lo um **ciclo de vida do objeto** especificando `cascade = "all"`.
- Caso contrário, `Save ()` e `Delete ()` explicitamente do código do aplicativo. Se você realmente quiser salvar a si mesmo a digitação extra, use `cascade = "save-update"` explícita e `Delete ()`.

Mapeamento de uma associação (many-to-one, ou coleção) com `cascade = "all"` marca a associação como um **pai / criança** relacionamento estilizado onde guardar / atualização / exclusão dos resultados mãe em save / update / exclusão do filho (s). Além disso, uma simples referência a uma criança de um pai persistente resultará em save / update do infantil. A metáfora é incompleta, no entanto. Uma criança que se torna não referenciado pelo seu pai é **não automaticamente eliminado**, exceto no caso de um `<one-to-many>` associação mapeada com `cascade = "all-delete-orphan"`. A semântica precisa de operações em cascata são os seguintes:

- Se um pai é salvo, todas as crianças são passados para `SaveOrUpdate ()`
- Se um pai é passado para `Update ()` ou `SaveOrUpdate ()`, Todas as crianças são passados para `SaveOrUpdate ()`
- Se uma criança se torna transitória referenciado por um pai persistente, ela é passada para `SaveOrUpdate ()`
- Se um pai é excluído, todas as crianças são passados para `Delete ()`
- Se uma criança é transitória dereferenced por um pai persistente, **nada de especial acontece** (O aplicativo deve excluir explicitamente a criança se necessário), a menos `cascade = "all-delete-orphan"`, Caso em que o "Órfãos" da criança é excluída.

NHibernate não implementa plenamente "persistência de acessibilidade", o que implicaria (ineficiente) persistente coleta de lixo. No entanto, devido à demanda popular, NHibernate oferece suporte para a noção de tornar-se entidades persistente quando referenciado por outro objeto persistente. Associações marcadas `cascade = "save-update"` comportar-se desta forma. Se você quiser usar essa abordagem em toda a sua aplicação, é mais fácil para especificar o `padrão-cascata` atributo do `<hibernate-mapping>` elemento.

9.10. Interceptores

O `IInterceptor` interface fornece retornos de chamada de sessão para a aplicação permitindo que a aplicação inspecionar e / ou manipular as propriedades de um objeto persistente antes de ser salvos, atualizados, deletados ou carregado. Um possível uso para isso é para rastrear as informações de auditoria. Por exemplo, o seguinte `IInterceptor` automaticamente define o `CreateTimestamp` quando um `IAuditable` é criada e atualiza o `LastUpdateTimestamp` propriedade quando um `IAuditable` é atualizado.

```
using System;
usando NHibernate.Type;

namespace NHibernate.Test
{
    [Serializable]
    AuditInterceptor public class: IInterceptor
    {

        privada atualizações int;
        int privada cria;

        onDelete public void (entidade objeto,
            identificação do objeto,
```

```

        objeto [estado],
        string [] propertyNames,
        ITipo tipos [])

    {
        / / Não faz nada
    }

    public boolean OnFlushDirty (entidade objeto,
                                identificação do objeto,
                                object [] currentState,
                                object [] previousState,
                                string [] propertyNames,
                                ITipo tipos []) {

        if (entidade é IAuditable)
        {
            atualizações + +;
            for (int i = 0; i < propertyNames.Length <; i + +)
            {
                if ("lastUpdateTimestamp" == propertyNames [i])
                {
                    currentState [i] = DateTime.Now;
                    return true;
                }
            }
        }
        return false;
    }

    OnLoad public boolean (entidade objeto,
                          identificação do objeto,
                          objeto [estado],
                          string [] propertyNames,
                          ITipo tipos [])
    {
        return false;
    }

    public boolean OnSave (entidade objeto,
                          identificação do objeto,
                          objeto [estado],
                          string [] propertyNames,
                          ITipo tipos [])
    {
        if (entidade é IAuditable)
        {
            cria + +;
            for (int i = 0; i < propertyNames.Length <; i + +)
            {
                if ("createTimestamp" == propertyNames [i])
                {
                    estado [i] = DateTime.Now;
                    return true;
                }
            }
        }
        return false;
    }

    PostFlush public void (entidades ICollection)
    {
        Console.Out.WriteLine ("Creations: {0}, Updates: {1}", cria, atualizações);
    }

    PreFlush public void (entidades ICollection) {
        updates = 0;
        cria = 0;
    }

    .....
    .....

```

```

    }
}
```

O interceptor será especificado quando uma sessão é criada.

```
Sessão ISession sf.OpenSession = (novo AuditInterceptor ());
```

Você também pode definir um interceptador em um nível global, usando o `Configuração`:

```
. novo Configuration () SetInterceptor (AuditInterceptor new ());
```

9.11. API de metadados

NHibernate requer um modelo muito rico meta-nível de todos os tipos de entidade e valor. De tempos em tempos, este modelo é muito útil para a aplicação em si. Por exemplo, a aplicação pode usar metadados NHibernate para implemento de um "smart" algoritmo deep-cópia que compreende quais objetos devem ser copiados (por exemplo, tipos mutáveis valor) e que não devem (por exemplo, tipos de valor imutável e, possivelmente, entidades associadas).

NHibernate expõe metadados via `IClassMetadata` e `ICollectionMetadata` interfaces e os `ITipo` hierarquia. Casos das interfaces de metadados podem ser obtidos a partir do `ISessionFactory`.

```

Gato fritz = ....;
IClassMetadata catMeta sessionfactory.GetClassMetadata = (typeof (Cat));
id = longa (long) catMeta.GetIdentifier (fritz);
object [] = propertyValues catMeta.GetPropertyValues (fritz);
string [] = propertyNames catMeta.PropertyNames;
ITipo [] = propertyTypes catMeta.PropertyTypes;

// Obtém um IDictionary de todas as propriedades que não são coleções ou associações
// TODO: o que dizer de componentes?

NamedValues IDictionary = new HashMap ();
for (int i = 0; i < propertyNames.Length; i + +)
{
    if (! propertyTypes [i]. IsEntityType & & ! propertyTypes [i]. IsCollectionType)
    {
        namedValues [propertyNames [i]] = propertyValues [i];
    }
}
```

Capítulo 10. Transações e Concorrência

NHibernate não é em si um banco de dados. É uma ferramenta de mapeamento objeto-relacional leve. Gerenciamento de transações é delegada a conexão do banco de dados subjacente. Se a conexão está inscrito com uma transação distribuída, operações realizadas pelo `ISession` são atomicamente parte da maior transação distribuída. NHibernate pode ser visto como um adaptador fino para ADO.NET, acrescentando orientada a objetos semântica.

10.1. Configurações, Sessões e Fábricas

Um `ISessionFactory` é um cara-a-criar, threadsafe objectos destinados a ser compartilhada por todas as aplicações threads. Um `ISession` é um barato, objeto não-threadsafe que deve ser usado uma vez, por uma única empresa processo, e depois descartados. Por exemplo, ao usar NHibernate em um aplicativo ASP.NET, as páginas podem obter-se `ISessionFactory` usando:

```
ISessionFactory sf = Global.SessionFactory;
```

Cada chamada para um método de serviço poderia criar um novo `ISession`, `Flush()` ele, `Commit()` sua operação, `Close()` ele e finalmente, descartá-lo. (A `ISessionFactory` também podem ser mantidos em uma estática `Singleton` variável auxiliar.)

Usamos o NHibernate `ITransaction` API como discutido anteriormente, um único `Commit()` de um NHibernate `ITransaction` flushes do estado e compromete qualquer conexão banco de dados subjacente (com tratamento especial de dis-transações contribuiu).

Certifique-se de entender a semântica da `Flush()`. Flushing sincroniza com o armazenamento persistente na memória mudanças, mas não vice-versa. Note-se que para todas as conexões ADO.NET NHibernate / transações, a transação nível de isolamento para essa conexão se aplica a todas as operações executadas pelo NHibernate!

As próximas seções irão discutir abordagens alternativas que utilizam versões para garantir a atomicidade da transação. Estes são considerados "avançados" abordagens para ser usado com cuidado.

10.2. Threads e conexões

Você deve observar as seguintes práticas ao criar Sessions NHibernate:

- Nunca crie mais de um concorrente `ISession` ou `ITransaction` instância por conexão com o banco.
- Tenha muito cuidado ao criar mais de uma `ISession` banco de dados por cada transação. O `ISession` it-automa mantém o controle de atualizações feitas para objetos carregados, de modo diferente `ISession` pode ver dados desatualizados.
- O `ISession` é não threadsafe! Nunca o mesmo acesso `ISession` em dois threads simultâneos. Um `ISession` é normalmente só unidade de trabalho-um único!

10.3. Considerando a identidade do objeto

A aplicação pode acessar concorrentemente o mesmo estado persistente em duas unidades de trabalho diferentes. No entanto, um instância de uma classe persistente nunca é compartilhada entre dois `ISession` instâncias. Portanto, há duas diferentes noções de identidade:

Identidade banco de dados

```
foo.Id.Equals (bar.Id)
```

CLR Identidade

```
foo == bar
```

Em seguida, para os objetos acoplados a um particular `Sessão`, As duas noções são equivalentes. No entanto, enquanto a aplicação pode acessar concorrentemente o "mesmo" objeto de negócio (identidade persistente) em duas sessões diferentes, os dois instâncias serão realmente "diferente" (CLR identidade).

Esta abordagem deixa NHibernate eo banco de dados que se preocupar com a concorrência. A aplicação não precisa sincronizar qualquer objeto de negócio, desde que adere a uma única thread por `ISESSION` ou identidade de objeto (dentro um `ISESSION` o aplicativo pode usar com segurança `==` comparar objetos).

10.4. Controle de simultaneidade otimista

Muitos processos de negócio requerem toda uma série de interações com o usuário intercaladas com acessos a banco de dados. Em aplicações web e corporativas não é aceitável para uma transação de banco de dados abrange uma interação do usuário.

Manter o isolamento dos processos de negócio torna-se responsabilidade parcial da camada de aplicativo, por isso, vamos chamar este processo de uma corrida de longa **transação da aplicação**. Uma transação única aplicação normalmente se estende por vários operações de banco de dados. Será atômica se somente uma destas transações (a última) armazena os-up datado de dados, todos os outros simplesmente ler os dados.

A única abordagem que seja coerente com alta concorrência e alta escalabilidade é con-simultaneidade otimista controle com controle de versões. NHibernate prevê três abordagens possíveis para escrever aplicações que usa simultaneidade otimista.

10.4.1. Longa sessão com versionamento automático

Um único `ISESSION` instância e suas instâncias persistentes são usados para a transação aplicação inteira.

O `ISESSION` utiliza o bloqueio otimista com a versão para garantir que as transações de banco de dados muitos aparecem para o aplicação como uma transação única aplicação lógica. O `ISESSION` está desconectado de qualquer subjacentes ADO.NET conexão enquanto espera a interação do usuário. Esta abordagem é a mais eficiente em termos de dados acesso base. A aplicação não precisa preocupar-se com a verificação de versão ou com reattaching destacada em posições.

```
/ / Foo é uma instância carregadas anteriormente pela Sessão
session.Reconnect ();
transação session.BeginTransaction = ();
foo.Property = "bar";
Session.flush ();
transaction.Commit ();
session.Disconnect ();
```

O `foo` objeto ainda sabe qual `ISESSION` ele foi carregado ele. Assim que o `ISESSION` ADO.NET tem um confiável Bluetooth, que confirmar as alterações para o objeto.

Este padrão é problemático se os nossos `ISESSION` é grande demais para ser armazenado durante usuário acha que o tempo, por exemplo, um `HttpSession` deve ser mantido o menor possível. Como o `ISESSION` é também a (obrigatório) cache de primeiro nível e contém todos os objetos carregados, podemos propably usar esta estratégia somente para alguns solicitação / resposta ciclos. Este é de fato recomendado, como o `ISESSION` em breve também com dados velhos.

10.4.2. Muitas sessões com versionamento automático

Cada interação com o armazenamento persistente ocorre em um novo `ISession`. No entanto, as mesmas instâncias persistentes são reutilizados para cada interação com o banco de dados. A aplicação manipula o estado das instâncias destacadas originalmente colocado em outra `ISession` e depois "reassociates-las" usando `ISession.Update ()` ou `ISession.SaveOrUpdate ()`.

```
/ / Foo é uma instância carregada por uma sessão anterior
foo.Property = "bar";
sessão factory.OpenSession = ();
transação session.BeginTransaction = ();
session.SaveOrUpdate (foo);
Session.flush ();
transaction.Commit ();
Session.close ();
```

Você também pode ligar para `Lock ()` em vez de `Update ()` e usar `LockMode.READ` (Realizando uma verificação de versão, ignorando todos os caches) se tiver certeza de que o objeto não foi modificado.

10.4.3. Versão do aplicativo de verificação

Cada interação com o banco de dados ocorre em um novo `ISession` que recarrega todas as instâncias persistentes a partir da database antes de manipulá-los. Esta abordagem força a aplicação a realizar sua própria versão de verificação para assegurar o isolamento de transações de aplicativos. (Claro, NHibernate ainda atualizar números de versão para você.) Este abordagem é o menos eficiente em termos de acesso banco de dados.

```
/ / Foo é uma instância carregada por uma sessão anterior
sessão factory.OpenSession = ();
transação session.BeginTransaction = ();
int = oldVersion foo.Version;
Session.load (foo, foo.Key);
if (! oldVersion foo.Version =) throw StaleObjectStateException new ();
foo.Property = "bar";
Session.flush ();
transaction.Commit ();
Session.close ();
```

Claro, se você estiver operando em um ambiente de baixa concorrência de dados e não requerem verificação de versão, você pode usar essa abordagem e simplesmente ignorar a verificação de versão.

10.5. Desconexão da sessão

A primeira abordagem descrita acima é para manter um único `ISession` para isso é um todo processo de negócio abrange usuário acha que o tempo. (Por exemplo, um servlet pode manter um `ISession` na do usuário `HttpSession`.) Para um desempenho razões pelas quais você deve

1. cometer o `ITransaction` e depois
2. desconecte o `ISession` a partir da conexão ADO.NET

antes de aguardar a atividade do usuário. O método `ISession.Disconnect ()` desconectará a sessão do ADO.NET conexão e retornar a ligação para a piscina (a menos que você forneceu a conexão).

`ISession.Reconnect ()` obtém uma nova conexão (ou você pode fornecer um) e reinicia a sessão. Depois de reconexão, para forçar uma checagem de versão em dados que não estão atualizando, você pode chamar `ISession.Lock ()` em todos os objetos que poderia ter sido atualizado por outra transação. Você não precisa bloquear todos os dados que você **são** atualização.

Está aqui um exemplo:

```

ISessionFactory sessões;
IList fooList;
Bar bar;
...
Sessions.OpenSession ISession s = ();
ITransaction tx = null;

tentar
{
    tx s.BeginTransaction = ()

    fooList s.Find =
        "Select foo foo de Eg.Foo onde foo.Date date = atual"
        / / Usa a função data db2
    );

    bar = new Bar ();
    s.Save (bar);

    tx.Commit ();
}
catch (Exception)
{
    if (tx = null!) tx.Rollback ();
    s.close ();
    throw;
}
s.Disconnect ();

```

Mais tarde:

```

s.Reconnect ();

tentar
{
    tx s.BeginTransaction = ();

    bar.FooTable = new HashMap ();
    foreach (Foo foo em fooList)
    {
        s.Lock (foo, LockMode.READ) / / verificar se foo não é obsoleto
        bar.FooTable.Put (foo.Name, foo);
    }

    tx.Commit ();
}
catch (Exception)
{
    if (tx = null!) tx.Rollback ();
    throw;
}
finalmente
{
    s.close ();
}

```

Você pode ver isto como a relação entre `ITransactions` e `ISessions` é muitos-para-um. Um `ISession` representa uma conversa entre o aplicativo e o banco de dados. O `ITransaction` quebra essa conexão até em unidades atômicas de trabalho ao nível do banco de dados.

10.6. Bloqueio pessimista

Não se pretende que os usuários gastam muito tempo preocupando estratégias de bloqueio. Geralmente é o suficiente para especificar um nível de isolamento para as conexões ADO.NET e depois simplesmente deixar o banco de dados fazer todo o trabalho. No entanto, ad-

usuários avançadas podem às vezes desejar obter locks pessimistas exclusivos, ou re-obter bloqueios no início de um novo transação.

NHibernate irá sempre usar o mecanismo de travamento do banco de dados, nunca trava objetos na memória!

O `LockMode` classe define os níveis de bloqueio diferentes que podem ser adquiridos pelo NHibernate. Um bloqueio é obtido pela dos seguintes mecanismos:

- `LockMode.READ` é adquirido automaticamente quando atualizações NHibernate ou insere uma linha.
- `LockMode.Upgrade` podem ser adquiridos mediante solicitação explícita do usuário usando `SELECT ... FOR UPDATE` em bancos de dados que suportam essa sintaxe.
- `LockMode.UpgradeNowait` podem ser adquiridos mediante solicitação explícita do usuário usando um `SELECT ... FOR UPDATE NOWAIT` em Oracle.
- `LockMode.READ` é adquirido automaticamente quando o NHibernate lê dados em Repetitivo Ler ou Serializar nível de isolamento capazes. Pode ser re-adquirido por solicitação explícita do usuário.
- `LockMode.NONE` representa a ausência de um bloqueio. Todos os objetos mudar para este modo de bloqueio no final de uma `ITransaction`. Objetos associados com a sessão através de uma chamada para `Update()` ou `SaveOrUpdate()` também começam neste modo de bloqueio.

O "pedido explícito do usuário" é expresso em uma das seguintes formas:

- Uma chamada para `ISession.Load()`, Especificando um `LockMode`.
- Uma chamada para `ISession.Lock()`.
- Uma chamada para `IQuery.SetLockMode()`.

Se `ISession.Load()` é chamado com `Melhorar` ou `UpgradeNowait`, E o objeto requisitado ainda não foi carregado pelo a sessão, o objeto é carregado usando `SELECT ... FOR UPDATE`. Se `Load()` é chamado para um objeto que já está carregado com um bloqueio menos restritivo do que o solicitado, as chamadas NHibernate `Lock()` para esse objeto.

`ISession.Lock()` executa um número de versão verificar se o modo de bloqueio especificado é `Ler`, `Melhorar` ou `-UpgradeNowait`. (No caso de `Melhorar` ou `UpgradeNowait`, `SELECT ... FOR UPDATE` é usado.)

Se o banco não oferece suporte ao modo de bloqueio solicitado, NHibernate irá usar um modo alternativo apropriado (Em vez de lançar uma exceção). Isto assegura que aplicações serão portáteis.

Capítulo 11. HQL: O Hibernate Query Language

NHibernate é equipado com uma linguagem de consulta extremamente poderoso que (muito intencionalmente) se parece muito como SQL. Mas não se deixe enganar pela sintaxe; HQL é totalmente orientada a objetos, compreendendo noções como herdar-polimorfismo referência, e de associação.

11.1. Sensibilidade caso

As consultas são insensíveis ao caso, exceto para nomes de classes, NET e propriedades. Assim `SELECCIONE` é o mesmo que `sELect`, `sELECT` é o mesmo que `SELECCIONE`, mas `Eg.FOO` não é `Eg.Foo` e `foo.barSet` não é `foo.BARSET`.

Este manual utiliza minúsculas palavras-chave HQL. Alguns usuários acham consultas com palavras-chave em maiúsculas mais legível, mas nós achamos essa convenção feia quando embutidos em código Java.

11.2. A cláusula from

A consulta NHibernate mais simples possível é da forma:

```
de Eg.Cat
```

que simplesmente retorna todas as instâncias da classe `Eg.Cat`.

Na maioria das vezes, você precisará atribuir um alias, já que você vai querer se referir ao `Gato` em outras partes do consulta.

```
de Eg.Cat como gato
```

Esta consulta atribui o alias `gato` para `Gato` instâncias, para que pudéssemos usar esse alias depois na consulta. O `como` palavra chave é opcional; também poderíamos escrever:

```
de gato Eg.Cat
```

Várias classes podem aparecer, resultando em um produto cartesiano ou "cross" join.

```
Parâmetro da Fórmula,
```

```
da Fórmula como forma, como parâmetro param
```

É considerada uma boa prática nome aliases consulta usando uma minúscula inicial, consistente com a nomeação stand-SARA para as variáveis locais (eg. `DomesticCat`).

11.3. Associações e junta-se

Podemos também atribuir aliases em uma entidade associada, ou mesmo a elementos de uma coleção de valores, utilizando uma `juntar`.

```
de Eg.Cat como gato
    cat.Mate junção interna como companheiro
    LEFT OUTER JOIN cat.Kittens como gatinho
```

```
de Eg.Cat como gato left join cat.Mate.Kittens como gatinhos
```

```
da forma Formula full join param form.Parameter
```

Os tipos de junção suportados são emprestados do SQL ANSI

- inner join
- LEFT OUTER JOIN
- direito de associação
- externa
full join (Geralmente não é útil)

O inner join, LEFT OUTER JOIN e direito de associação externa construções podem ser abreviados.

```
de Eg.Cat como gato
    juntar cat.Mate como companheiro
    left join cat.Kittens como gatinho
```

Além disso, um "fetch" join permite que associações ou coleções de valores a serem inicializados junto com seus pais objetos, utilizando uma única escolha. Isso é particularmente útil no caso de uma coleção. Ele substitui a junção externa e declarações preguiçoso do arquivo de mapeamento para as associações e coleções.

```
de Eg.Cat como gato
    junção interna buscar cat.Mate
    left join fetch cat.Kittens
```

A juntar-se buscar, normalmente, não precisa atribuir um alias, porque os objetos associados não deve ser usado na onde cláusula (ou de qualquer outra cláusula). Além disso, os objetos associados não são retornados diretamente nos resultados da consulta. In-lugar, eles podem ser acessados através do objeto pai.

Note-se que, na implementação atual, apenas um papel de coleta podem ser obtidos em uma consulta (tudo o resto seria não-performance). Note também que o buscar construção não podem ser utilizadas em consultas chamado usando Enumer-capazes (). Finalmente, note que full join fetch e direito join fetch não são significativas.

11.4. A cláusula select

O selecionar cláusula que pega objetos e propriedades para retornar no conjunto de resultados da consulta. Considerar:

```
selecionar companheiro
de Eg.Cat como gato
    cat.Mate junção interna como
    companheiro
```

A consulta irá selecionar Companheiros de outros Gatos. Na verdade, você pode expressar esta consulta de forma mais compacta como:

```
selecionar cat.Mate de Eg.Cat gato
```

Você pode até escolher elementos da coleção, usando o especial elementos função. A consulta a seguir retorna todos os gatinhos de qualquer gato.

```
elementos select (cat.Kittens) de gato Eg.Cat
```

Consultas podem retornar propriedades de qualquer tipo de valor, incluindo propriedades de tipo de componente:

```
selecionar cat.Name de Eg.DomesticCat gato
onde cat.Name like '% sex'
```

```
selecionar cust.Name.FirstName do Cliente como cust
```

Consultas podem retornar vários objetos e / ou propriedades como um array de tipo `Object []`

```
selecionar mãe, offspr, mate.Name
de Eg.DomesticCat como mãe
    mother.Mate junção interna como companheiro
    LEFT OUTER JOIN mother.Kittens como offspr
```

ou como um objeto real typesafe

```
seleccione nova família (mãe, companheiro, offspr)
de Eg.DomesticCat como mãe
    juntar mother.Mate como companheiro
    left join mother.Kittens como offspr
```

assumindo que a classe `Família` tem um construtor apropriado.

11.5. Funções de agregação

Consultas HQL pode até retornar os resultados de funções agregadas em propriedades:

```
SELECT AVG (cat.Weight), sum (cat.Weight), max (cat.Weight), count (gato)
de gato Eg.Cat
```

Coleções também podem aparecer dentro de funções de agregação na `selecionar` cláusula.

```
selecionar gato, count (elementos (cat.Kittens))
de Eg.Cat grupo gato por gato
```

As funções suportadas são agregadas

- `avg (...), sum (...), min (...), max (...)`
- `count (*)`
- `count (...), count (distintas ...), count (all. ...)`

O `distinto` e `todos` palavras-chave podem ser usadas e têm a mesma semântica como em SQL.

```
selecionar cat.Name distintas de gato Eg.Cat
select count (cat.Name distintas), count (gato) de gato Eg.Cat
```

11.6. Consultas polimórficas

Uma consulta como:

```
de Eg.Cat como gato
```

retorna instâncias não só de `Gato`, Mas também de subclasses como `DomesticCat`. Consultas NHibernate pode nomear qualquer .NET classe ou interface na a partir de cláusula. A consulta retornará instâncias de todas as classes persistentes que se estendem classe ou implementar a interface. A seguinte consulta retornaria todos os objetos persistentes:

```
de System.Object o
```

A interface `Inamed` pode ser implementada por várias classes persistentes:

```
Eg.Named de n, onde m Eg.Named n.Name = m.Name
```

Note-se que estas duas últimas consultas exigirão mais do que um SQL SELECIONE. Isto significa que o por fim cláusula não corretamente fim do conjunto de resultados inteiro.

11.7. A cláusula where

O onde cláusula permite restringir a lista de instâncias retornadas.

```
de Eg.Cat como gato onde cat.Name = 'Fritz'
```

retorna instâncias de Gato chamado 'Fritz'.

```
select foo
de Eg.Foo foo, bar Eg.Bar
onde foo.StartDate = bar.Date
```

irá retornar todas as instâncias do Foo para a qual existe uma instância de Bar com um Data propriedade igual ao StartDate propriedade da Foo. Expressões caminho composto fazer a onde cláusula extremamente poderoso. Consider:

```
de gato Eg.Cat onde cat.Mate.Name não é nulo
```

Isso se traduz consulta para uma consulta SQL com uma tabela (interno) de junção. Se você fosse escrever algo como

```
de Eg.Foo foo
onde foo.Bar.Baz.Customer.Address.City não é nulo
```

você iria acabar com uma consulta que exigiria quatro junções de tabela no SQL.

O =operador pode ser usado para comparar não apenas propriedades, mas também instâncias:

```
de gato Eg.Cat, rival Eg.Cat onde cat.Mate = rival.Mate
selecionar gato, companheiro
Eg.Cat de gato, companheiro Eg.Cat
onde cat.Mate companheiro =
```

A propriedade especial (em minúsculas) id pode ser usado para fazer referência ao identificador único de um objeto. (Você também pode usar o seu nome da propriedade.)

```
de Eg.Cat como gato onde cat.id = 123
de Eg.Cat como gato onde cat.Mate.id = 69
```

A segunda consulta é eficiente. Não participar de mesa é necessário!

Propriedades de identificadores compostas também podem ser usados. Supor Pessoa tenha um identificador composto que consiste em

Pais e MedicareNumber.

```
de pessoa Bank.Person
onde person.id.Country = "UA"
e person.id.MedicareNumber = 123456

da conta Bank.Account
onde account.Owner.id.Country = "UA"
e account.Owner.id.MedicareNumber = 123456
```

Mais uma vez, a segunda consulta não requer tabela de junção.

Da mesma forma, a propriedade especial `classe` acessa o valor discriminador de uma instância no caso de polimórficos persistência. Um nome de classe Java embutido na cláusula onde será traduzido para o seu valor discriminador.

```
de gato Eg.Cat onde cat.class = Eg.DomesticCat
```

Você também pode especificar propriedades de componentes ou tipos de usuário composto (e de componentes de componentes, etc). Nunca tente usar um caminho de expressão que termina em uma propriedade do tipo componente (em oposição a uma propriedade de um componente). Por exemplo, se `store.Owner` é uma entidade com um componente `Endereço`

```
store.Owner.Address.City      / / Ok
store.Owner.Address          / / Erro!
```

Um tipo "any" tem as propriedades especiais `id` e `classe`, O que nos permite expressar uma junção da seguinte maneira (Onde `AuditLog.Item` é uma propriedade mapeada com `<any>`).

```
Eg.AuditLog de log, o pagamento Eg.Payment
onde Eg.Payment log.Item.class = ', por exemplo, Versão =...' e log.Item.id = payment.id
```

Note-se que `log.Item.class` e `payment.class` remete para os valores da base de dados completamente diferente colunas na consulta acima.

11.8. Expressões

Expressões permitidas na `onde` cláusula incluem a maioria do tipo de coisas que você poderia escrever em SQL:

- operadores matemáticos `+, -, *, /`
- operadores de comparação binária `=, >=, <=, <>, !=`, Como
- operações lógicas `e, ou, não`
- concatenação `||`
- Funções escalares SQL como `upper()` e `inferior()`
- Parênteses `()` indicam agrupamento
- `em, entre, é nulo`
- parâmetros de posição `?`
- parâmetros nomeados : `Nome, : Start_date, : X1`
- Literais SQL `'Foo', 69, '1970-01-01 10:00:01.0'`
- Valores de enumeração e constantes `Eg.Color.Tabby`

`em` e `entre` pode ser usado da seguinte forma:

```
de gato Eg.DomesticCat onde cat.Name entre 'A' e 'B'
de gato Eg.DomesticCat onde cat.Name in ('Foo', 'Bar', 'Baz')
```

e as formas negada pode ser escrito

```
de gato Eg.DomesticCat onde cat.Name não entre 'A' e 'B'
de gato Eg.DomesticCat onde cat.Name não in ('Foo', 'Bar', 'Baz')
```

Da mesma forma, `é nulo` e `não é nula` podem ser usados para testar valores nulos.

Booleanos podem ser facilmente usados em expressões, declarando as substituições consulta HQL na configuração NHibernate:

```
<property name="hibernate.query.substitutions"> true 1, false 0 </ property>
```

Isso irá substituir as palavras-chave verdadeiro e falso com os literais 1 e 0 no SQL traduzida a partir deste HQL:

```
de gato Eg.Cat onde cat.Alive = true
```

Você pode testar o tamanho de uma coleção com a propriedade especial tamanho. Ou o especial size () função.

```
de gato Eg.Cat onde cat.Kittens.size> 0
de gato Eg.Cat onde o tamanho (cat.Kittens)> 0
```

Para coleções indexadas, você pode referir-se ao mínimo e máximo usando índices minindex e maxindex. Da mesma forma, você pode referir-se aos elementos mínimos e máximos de uma coleção de tipos básicos usando minElement e maxElement.

```
Calendário de cal onde a data> atual cal.Holidays.maxElement
```

Há também formas funcionais (que, diferentemente das construções acima, não são case sensitive):

```
da ordem de Ordem, onde maxindex (order.Items)> 100
da ordem Ordem minelement onde (order.Items)> 10000
```

As funções SQL algum, alguns, todos, existe, em são suportados quando passado o elemento ou conjunto de índices de um col-
lecão (elementos e indices funções) ou o resultado de uma subconsulta (veja abaixo).

```
selecionar mãe de Eg.Cat como mãe, como kit Eg.Cat
onde kit em elementos (mother.Kittens)

selecionar p da lista Eg.NameList, Eg.Person p
onde p.Name = alguns elementos (list.Names)

de gato Eg.Cat onde existe elementos (cat.Kittens)

de Eg.Player p onde 3> todos os elementos (p.Scores)

de mostrar Eg.Show onde 'fizard "nos indices (show.Acts)
```

Note-se que essas construções - tamanho,elementos,índices,minindex,maxindex,minelement,maxelement - Têm certas restrições de uso:

- em um onde cláusula: apenas para bancos de dados com subconsultas
- em um selecionar cláusula: apenas elementos e índices fazer sentido

Elementos de coleções indexadas (matrizes, listas, mapas) podem ser referidos pelo índice (em uma cláusula onde apenas):

```
da ordem de Ordem, onde order.Items [0]. id = 1234

selecionar pessoa de pessoa Pessoa, Calendar calendário
onde calendar.Holidays ['dia nacional'] = person.BirthDay
e person.Nationality.Calendar = calendário

seleccione o item a partir do item Item, a fim Ordem
onde order.Items [order.DeliveredItemIndices [0]] = item = 11 e order.id

seleccione o item a partir do item Item, a fim Ordem
onde order.Items [maxindex (order.items)] = item = 11 e order.id
```

A expressão dentro [] pode até ser uma expressão aritmética.

```
seleciona o item a partir do item Item, a fim Ordem
onde order.Items [size (order.Items) - 1] item =
```

HQL também fornece o built-in `index ()` função, para os elementos de uma associação um-para-muitos ou coleção de valores.

```
selecionar indice de item, (item) da ordem de Ordem
juntar-se item de order.Items
onde o indice (item) <5
```

Funções escalares SQL suportados pelo banco de dados subjacente pode ser usado

```
de gato, onde Eg.DomesticCat superior (cat.Name) like '% sex'
```

Se você ainda não estão convencidos por tudo isso, acho que quanto mais tempo e menos legível a seguinte consulta seria em SQL:

```
cust selecione
Produto de prod,
Loja loja
inner join store.Customers cust
onde prod.Name = 'widget'
e store.Location.Name in ('Melbourne', 'Sydney')
e prod = todos os elementos (cust.CurrentOrder.LineItems)
```

Dica: algo parecido

```
SELECCIONE cust.name, cust.address, cust.phone, cust.id, cust.current_order
FROM cust_clientes,
loja_lojas,
loc_locais,
store_customers sc,
produto_prod
ONDE prod.name = 'widget'
E store.loc_id = loc.id
E loc.name IN ('Melbourne', 'Sydney')
E sc.store_id = store.id
E sc.cust_id = cust.id
E prod.id = ALL (
    SELECCIONE item.prod_id
    FROM line_items, ordens o
    ONDE item.order_id = o.id
    E cust.current_order = o.id
)
```

11.9. A ordem pela cláusula

A lista retornada por uma consulta pode ser ordenada por qualquer propriedade de uma classe devolvidos ou componentes:

```
de gato Eg.DomesticCat
por fim cat.Name asc, desc cat.Weight, cat.Birthdate
```

O opcional `asc` ou `desc` indicam ordem ascendente ou descendente, respectivamente.

11.10. A cláusula group by

Uma consulta que retorna valores agregados podem ser agrupados por qualquer propriedade de uma classe devolvidos ou componentes:

```

selecionar cat.Color, sum (cat.Weight), count (gato)
de gato Eg.Cat
grupo por cat.Color

selecionar foo.id, avg (elementos (foo.Names)), max (índices (foo.Names))
de Eg.Foo foo
grupo por foo.id

```

Nota: Você pode usar o `elementos` e `índices` constrói dentro de uma cláusula select, mesmo em bancos de dados sem subselects.

A ter cláusula também é permitido.

```

selecionar cat.color, sum (cat.Weight), count (gato)
de gato Eg.Cat
grupo por cat.Color
tendo em cat.Color (Eg.Color.Tabby, Eg.Color.Black)

```

Funções SQL e funções de agregação são permitidos no `ter` e `por fim` cláusulas, se suportado pela un-banco de dados sustém (isto é, não no MySQL).

```

selecionar gato
de gato Eg.Cat
    juntar cat.Kittens gatinho
grupo por gato
ter avg (kitten.Weight)> 100
por fim count (gatinho) asc, sum (kitten.Weight) desc

```

Note-se que nem o `grupo por` cláusula, nem o `por fim` cláusula pode conter expressões aritméticas.

11.11. Subconsultas

Para bancos de dados que suportam subselects, NHibernate suporta subqueries dentro de queries. Uma subconsulta deve ser sur-arredondado por parênteses (muitas vezes por uma chamada de função agregada SQL). Mesmo subqueries correlacionadas (subconsultas que referem-se a um alias na consulta externa) são permitidos.

```

de Eg.Cat como fatcat
onde fatcat.Weight> (
    SELECT AVG (cat.Weight) de Eg.DomesticCat gato
)

de Eg.DomesticCat como gato
onde cat.Name = alguns (
    selecionar name.NickName de Eg.Name como nome
)

de Eg.Cat como gato
onde não existe (
    de eg.Cat como companheiro onde mate.Mate = cat
)

de Eg.DomesticCat como gato
onde cat.Name não em (
    selecionar name.NickName de Eg.Name como nome
)

```

11.12. Exemplos HQL

Consultas NHibernate pode ser muito poderosa e complexa. Na verdade, o poder da linguagem de consulta é um dos

Principais pontos do NHibernate venda. Aqui estão alguns exemplos de consultas muito semelhantes às consultas que eu usei em uma recente projeto. Note-se que a maioria das consultas que você vai escrever são muito mais simples do que estes!

A consulta a seguir retorna a ID de ordem, número de itens e o valor total da encomenda para todos os pedidos não pagos por um determinado cliente e dado valor mínimo total, ordenando os resultados por valor total. Na determinação do preços, ele usa o catálogo atual. O resultado da consulta SQL, contra o ORDER, ORDER_LINE, PRODUTO, CATÁLOGO e PREÇO tabelas tem quatro junta interior e um subselect (não correlacionadas).

```

selecionar order.id, sum (price.Amount), count (item)
de Ordem como forma
    juntar order.LineItems como item
    juntar item.Product como produto,
    Catálogo como catálogo
    juntar catalog.Prices como preço
onde order.Paid = false
    e Order.Customer =: cliente
    e price.Product produto =
    e catalog.EffectiveDate <sysdate
    e catalog.EffectiveDate> = all (
        selecionar cat.EffectiveDate
        Catalogo de como gato
        onde cat.EffectiveDate <sysdate
    )
grupo, por ordem
ter sum (price.Amount)>: minAmount
fim de soma (price.Amount) desc

```

O que é um monstro! Na verdade, na vida real, eu não estou muito interessado em subqueries, então minha query seria mais parecida com esta:

```

selecionar order.id, sum (price.amount), count (item)
de Ordem como forma
    juntar order.LineItems como item
    juntar item.Product como produto,
    Catálogo como catálogo
    juntar catalog.Prices como preço
onde order.Paid = false
    e Order.Customer =: cliente
    e price.Product produto =
    e catalogar =: currentCatalog
grupo, por ordem
ter sum (price.Amount)>: minAmount
fim de soma (price.Amount) desc

```

A próxima consulta conta o número de pagamentos em cada status, excluindo todos os pagamentos no AwaitingApproval estado onde a mudança de status mais recente foi feita pelo usuário atual. Isso se traduz em uma consulta SQL com dois junções internas e um subselect correlacionado contra o PAGAMENTO, PAYMENT_STATUS e PAYMENT_STATUS_CHANGE tabelas.

```

select count (pagamento), status.Name
de pagamento como forma de pagamento
    juntar payment.CurrentStatus como o status
    juntar payment.StatusChanges como StatusChange
onde payment.Status.Name <PaymentStatus.AwaitingApproval>
    ou (
        statusChange.TimeStamp = (
            select max (change.TimeStamp)
            da mudança PaymentStatusChange
            onde change.Payment pagamento =
        )
        e statusChange.User <>: CurrentUser
    )
grupo por status.Name, status.SortOrder
por fim status.SortOrder

```

Se eu tivesse mapeado o `StatusChanges` coleção como uma lista, em vez de um conjunto, a consulta teria sido muito mais simples de escrever.

```
select count (pagamento), status.Name
de pagamento como forma de pagamento
    juntar payment.CurrentStatus como o status
onde payment.Status.Name <PaymentStatus.AwaitingApproval>
    ou payment.StatusChanges [maxindex (payment.StatusChanges)] Usuário <>:. CurrentUser
grupo por status.Name, status.SortOrder
por fim status.SortOrder
```

A próxima consulta usa o MS SQL Server `IsNull ()` função para retornar todas as contas e pagamentos por pagar para a organização a qual o usuário atual pertence. Isso se traduz em uma consulta SQL com três junções internas, uma exterior juntar-se e um subselect contra o `CONTA,PAGAMENTO,PAYMENT_STATUS,ACCOUNT_TYPE,ORGANIZAÇÃO e ORG_USER` tabelas.

```
conta de selecionar, o pagamento
de conta como conta
    LEFT OUTER JOIN account.Payments como pagamento
onde: CurrentUser em elementos (account.Holder.Users)
    e PaymentStatus.Unpaid = isNull (payment.CurrentStatus.Name, PaymentStatus.Unpaid)
por fim account.Type.SortOrder, account.AccountNumber, payment.DueDate
```

Para alguns bancos de dados, nós precisaríamos acabar com o subselect (correlacionados).

```
conta de selecionar, o pagamento
de conta como conta
    juntar-se como usuário account.Holder.Users
    LEFT OUTER JOIN account.Payments como pagamento
onde: CurrentUser = user
    e PaymentStatus.Unpaid = isNull (payment.CurrentStatus.Name, PaymentStatus.Unpaid)
por fim account.Type.SortOrder, account.AccountNumber, payment.DueDate
```

11.13. Dicas & Truques

Você pode contar o número de resultados da consulta sem realmente devolvê-los:

```
IEnumerable countEn session.Enumerable = ("select count (*) de ....");
countEn.MoveNext ();
int count = (int) countEn.Current;
```

Para ordenar um resultado pelo tamanho de uma coleção, use a seguinte consulta:

```
selecionar usr.id, usr.Name
de Usuário como usr
    left join usr.Messages como msg
grupo por usr.id, usr.Name
ordem por count (msg)
```

Se o seu banco de dados suporta subselects, você pode colocar uma condição sobre tamanho da seleção na cláusula where da sua consulta:

```
Usuário de usr onde o tamanho (usr.Messages)> = 1
```

Se o seu banco de dados não suporta subselects, use a seguinte consulta:

```
selecionar usr.id, usr.Name
do Usuário usr
    juntar usr.Messages msg
```

```
grupo por usr.id, usr.Name
ter count (msg) > = 1
```

Como essa solução não pode retornar um `Usuário` com zero mensagens por causa da junção interna, o seguinte formulário também é de uso ful:

```
selecionar usr.id, usr.Name
de Usuário como usr
    left join usr.Messages como msg
grupo por usr.id, usr.Name
ter count (msg) = 0
```

Propriedades de um objeto pode ser vinculada a parâmetros de consulta nomeada:

```
IQuery s.CreateQuery q = ("from foo na classe Foo onde foo.Name =: Nome e foo.Size =: Size");
q.SetProperties (fooBean) // fooBean tem propriedades Nome e Tamanho
IList foos q.List = ();
```

Coleções são paginável usando o `IQuery` interface com um filtro:

```
IQuery q = s.CreateFilter (recolha, ""); // filtro do trivial
q.setMaxResults (PageSize);
q.setFirstResult (PageSize pageNumber *);
IList page = q.List ();
```

Elementos da coleção podem ser ordenados ou agrupados usando um filtro de consulta:

```
OrderedCollection ICollection = s.Filter (recolha, "order by this.Amount");
Conta ICollection = s.Filter (coleção ", selecione this.Type, count (isso) por grupo this.Type");
```

Capítulo 12. Consultas critérios

NHibernate agora possui uma interface intuitiva, API consulta extensível critérios. Por enquanto, essa API é menos poderoso do que o mais maduro facilidades de consultas HQL. Em particular, as consultas critérios não suportam projeção ou agregação.

12.1. Criação de um ICriteria exemplo

A interface `NHibernate.ICriteria` representa uma consulta contra uma classe particular persistente. O `ISession` é uma fábrica para `ICriteria` instâncias.

```
ICriteria crit = sess.CreateCriteria (typeof (Cat));
crit.SetMaxResults (50);
Gatos crit.List lista = ();
```

12.2. Estreitamento do conjunto de resultados

Um critério de consulta individual é uma instância da interface `NHibernate.Expression.ICriterion`. A classe `NHibernate.Expression.Expression` define métodos de fábrica para a obtenção de certas built-in `ICriterion` tipos.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
    . Add (Expression.Like ("Nome", "% Fritz"))
    . Add (Expression.Between ("Peso", minWeight, maxWeight))
    . List ();
```

Expressões podem ser agrupadas de maneira lógica.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
    . Add (Expression.Like ("Nome", "% Fritz"))
    . Add (Expression.Or (
        Expression.Eq ("Idade", 0),
        Expression.IsNotNull ("Idade")
    ))
    . List ();
```

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
    . Add (Expression.In ("Nome", new String [] {"Fritz", "Izi", "Pk"}))
    . Add (Expression.Disjunction ()
        . Add (Expression.IsNotNull ("Idade"))
        . Add (Expression.Eq ("Idade", 0))
        . Add (Expression.Eq ("Idade", 1))
        . Add (Expression.Eq ("Idade", 2))
    )
    . List ();
```

Há uma gama bastante de built-in tipos critério (`Expression` subclasses), mas que é especialmente útil permite você especificar SQL diretamente.

```
/ / Cria um parâmetro de seqüência para o SqlString abaixo
Parâmetro paramName = novo parâmetro ("AlgumNome", new StringSqlType ());

Gatos IList sess.CreateCriteria = (typeof (Cat))
    . Add (Expression.Sql (
        nova SqlString (novo objeto [] {
            "Inferior ({alias} Nome.) Como inferiores (",
            paramName,
            ")"),
            "Fritz%"
```

```
NHibernateUtil.String)
. List ();
```

O `{Alias}` espaço reservado é substituído pelo apelido de linha da entidade consultada.

12.3. Ordenar os resultados

Você pode ordenar os resultados usando `NHibernate.Expression.Order`.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
. Add (Expression.Like ("Nome", "% F"))
. AddOrder (Order.Asc ("Nome"))
. AddOrder (Order.Desc ("Idade"))
. SetMaxResults (50)
. List ();
```

12.4. Associações

Você pode facilmente especificar restrições sobre entidades relacionadas ao navegar usando associações `CreateCriteria ()`.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
. Add (Expression.Like ("Nome", "% F"))
. CreateCriteria ("gatinhos")
. Add (Expression.Like ("Nome", "% F"))
. List ();
```

note que o segundo `CreateCriteria ()` retorna uma nova instância `ICriteria`, Que se refere aos elementos da coleção `Gatinhos`.

A seguinte forma, alternativa é útil em determinadas circunstâncias.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
. CreateAlias ("gatinhos", "KT")
. CreateAlias ("Mate", "mt")
. Add (Expression.EqProperty ("kt.Name", "mt.Name"))
. List ();
```

(`CreateAlias ()` não cria uma nova instância `ICriteria`.)

Note-se que as coleções gatinhos realizada pela `Gato` instâncias retornadas pelo anterior duas consultas são **não** pré-filtrada pelos critérios! Se você deseja recuperar apenas os gatinhos que correspondem aos critérios, você deve usar `SetResultTransformer (CriteriaUtil.AliasToEntityMap)`.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
. CreateCriteria ("gatinhos", "KT")
. Add (Expression.Eq ("Nome", "% F"))
. SetResultTransformer (CriteriaUtil.AliasToEntityMap)
. List ();
foreach (mapa IDictionary em gatos)
{
    Cat = gato (Cat) map [CriteriaUtil.RootAlias];
    Gato gatinho = (Cat) map ["kt"];
}
```

12.5. Buscar associação dinâmica

Você pode especificar a semântica associação busca em tempo de execução usando `SetFetchMode()`.

```
Gatos IList sess.CreateCriteria = (typeof (Cat))
    . Add (Expression.Like ("Nome", "% Fritz"))
    . SetFetchMode ("Mate", FetchMode.Eager)
    . SetFetchMode ("gatinhos", FetchMode.Eager)
    . List ();
```

Esta consulta vai buscar tanto `Companheiro` e `Gatinhos` pela junção externa.

12.6. Consultas de exemplo

A classe `NHibernate.Expression.Example` permite construir um critério de consulta a partir de uma determinada instância.

```
Cat = new Gato ();
cat.Sex = 'F';
cat.Color = Color.black;
Resultados da lista = session.CreateCriteria (typeof (Cat))
    . Add (Example.Create (gato))
    . List ();
```

Propriedades versão, identificadores e associações são ignorados. Por padrão, null propriedades avaliadas e propriedades que retornar uma string vazia da chamada para `ToString ()` são excluídos.

Você pode ajustar a forma como o `Exemplo` é aplicada.

```
Exemplo Example.Create = (gato)
    . ExcludeZeroes () // excluir nulo ou zero propriedades avaliadas
    . ExcludeProperty ("cor") // excluir a propriedade chamada "cor"
    . IgnoreCase () // case insensitive realizar comparações de strings
    . Utilizar ()// EnableLike como para comparações
Resultados IList session.CreateCriteria = (typeof (Cat))
    . Add (exemplo)
    . List ();
```

Você ainda pode usar exemplos para colocar critérios sobre os objetos associados.

```
Resultados IList session.CreateCriteria = (typeof (Cat))
    . Add (Example.Create (gato))
    . CreateCriteria ("Mate")
        . Add (Example.Create (cat.Mate))
    . List ();
```

Capítulo 13. Native Queries SQL

Você também pode expressar consultas no dialeto SQL nativo de seu banco de dados. Isto é útil se você quiser utilizar características do banco de dados específicos, tais como a palavra-chave CONNECT em Oracle. Isso também permite uma migração mais limpa caminho a partir de uma aplicação SQL / ADO.NET direta baseada em NHibernate.

13.1. Criar um baseado em SQL IQuery

Consultas SQL são expostos através do mesmo `IQuery` interface, assim como consultas HQL normal. A única diferença é o uso de `ISession.CreateSQLQuery ()`.

```
IQuery SQLQuery = sess.CreateSQLQuery ("select {cat .*} {cat de gatos}", "gato", typeof (Cat));
sqlQuery.SetMaxResults (50);
Gatos IList sqlQuery.List = ();
```

Os três parâmetros fornecidos para `CreateSQLQuery ()` são:

- a string de consulta SQL
- uma tabela nome de alias
- a classe persistente retornadas pela consulta

O nome do alias é usado dentro da string SQL para se referir às propriedades da classe mapeada (neste caso `Gato`). Você pode recuperar vários objetos por linha, fornecendo uma `Corda` array de nomes de alias e um `System.Type` disposição das classes correspondentes.

13.2. Alias e referências propriedade

O `{Cat .*}` notação usada acima é uma abreviação para "todas as propriedades". Você pode até listar as propriedades explicitamente, mas você deve deixar NHibernate fornecer aliases SQL coluna para cada propriedade. Os espaços reservados para esses coluna aliases são o nome da propriedade qualificado pelo alias da tabela. No exemplo a seguir, nós recuperamos `Gatos` de uma diretabela diferentes (`cat_log`) declarada nos metadados de mapeamento. Observe que nós ainda pode usar a propriedade aliases na cláusula where.

```
string sql = "select cat.originalId como {} cat.Id"
+ "Cat.mateid como {} cat.Mate, cat.sex como {} cat.Sex"
+ "Cat.weight * 10 como {} cat.Weight cat.name, como {} cat.Name"
+ "De cat_log gato onde {} = cat.Mate: catid"
IList loggedCats = sess.CreateSQLQuery (sql, "gato", typeof (Cat))
. SetInt64 ("catid", catid)
. List ();
```

Nota: se você listar cada propriedade explicitamente, você deve incluir todas as propriedades da classe e suas subclasses!

13.3. Chamado consultas SQL

Consultas nomeadas SQL podem ser definidas no documento de mapeamento e chamou exatamente da mesma maneira como um chamado Consulta HQL.

```
Pessoas IList = sess.GetNamedQuery ("mySqlQuery")
```

```
. SetMaxResults (50)
. List ();
```

```
<sql-query name="mySqlQuery">
  <Voltar alias="person" class="Eg.Person, Eg"/>
  SELECT {} NOME pessoa. AS {} person.Name,
         {} Pessoa}. IDADE AS {} person.Age,
         {} SEX pessoa. AS {} person.Sex
    FROM PESSOA {pessoa} onde {} NOME pessoa. LIKE '% Hiber'
</ Sql-query>
```

Capítulo 14. Melhorar o desempenho

14.1. Compreensão do desempenho Colecção

Já passou algum tempo conversando sobre coleções. Nesta seção vamos destacar um par mais questões sobre como se comportar durante a execução coleções.

14.1.1. Taxonomia

NHibernate define três tipos básicos de coleções:

- coleções de valores
- um a muitas associações
- muitos a muitas associações

Esta classificação distingue a mesa várias e relacionamentos de chave estrangeira, mas não nos dizem muito tudo o que precisamos saber sobre o modelo relacional. Para entender completamente a estrutura relacional e performance, devemos também considerar a estrutura da chave primária que é usada pelo NHibernate para atualizar ou excluir linhas de coleta. Isto sugere a seguinte classificação:

- coleções indexadas
- conjuntos
- sacos

Todas as coleções indexadas (mapas, listas, arrays) tem uma chave primária que consiste na `<key>` e `<index>` colunas. Neste caso, as atualizações coleção são geralmente extremamente eficiente - a chave primária pode ser eficientemente indexados e uma determinada linha pode ser eficientemente localizado quando NHibernate tenta atualizar ou excluí-lo.

Conjuntos de ter uma chave primária composta por `<key>` e colunas elemento. Isso pode ser menos eficaz para alguns tipos de elemento da coleção, composta principalmente de texto ou elementos de grandes dimensões ou campos binários, o banco de dados pode não ser capaz

o índice de uma chave de complexo primário de forma tão eficiente. Por outro lado, para um para muitos ou muitos para muitos associações, particularmente no caso dos identificadores sintéticos, é provável que seja tão eficiente. (Side-note: se você quiser `SchemaExport` para realmente criar a chave primária de uma `<set>` para você, você deve declarar todas as colunas como `não-null = "true"`.)

Sacos são o pior caso. Uma vez que um saco de licenças de valores de elementos duplicados e não tem coluna do índice, nenhuma chave primária podem ser definidos. NHibernate não tem nenhuma maneira de distinguir entre linhas duplicadas. NHibernate resolve este problema por remover completamente (em um único APAGAR) E recriando a coleção sempre que ela muda. Este pode ser muito ineficiente.

Note que para uma associação de um-para-muitos, a "chave primária" pode não ser a chave primária física do banco de dados tabela - mas mesmo neste caso, a classificação acima ainda é útil. (Ele ainda reflete como NHibernate "localiza" individual linhas da coleção.)

14.1.2. Listas, mapas e sets são as coleções mais eficiente de atualização

A partir da discussão acima, deve ficar claro que as coleções indexadas e (geralmente) conjuntos permitem o mais eficiente operação em termos de adição, remoção e atualização de elementos.

Há, sem dúvida, mais uma vantagem que têm sobre coleções indexadas conjuntos, para muitos, muitas associações ou coleções de valores. Por causa da estrutura de um `ISET`, NHibernate não é nunca ATUALIZAÇÃO uma linha quando um elemento é "mudou". Alterações em um `ISET` sempre funciona via `INSERIR` e `APAGAR` (De linhas individuais). Uma vez novamente, esta consideração não se aplica a um para muitas associações.

Depois de observar que as matrizes não pode ser preguiçoso, podemos concluir que as listas, mapas e sets são as de melhor desempenho tipos de coleção. (Com a ressalva de que um conjunto pode ser menos eficiente para algumas coleções de valores.) Conjuntos devem ser o tipo mais comum de cobrança em aplicações NHibernate.

Existe um recurso não documentado nesta versão do NHibernate. O `<idbag>` mapeamento implementa saco de semantics para uma coleção de valores ou uma associação de muitos para muitos e é mais eficiente que qualquer outro estilo de coleta, neste caso!

14.1.3. Sacos e as listas são as coleções mais eficiente inversa

Pouco antes de abandonar sacos para sempre, não é um caso particular em que sacos (e também listas) são muito mais performante de sets. Para uma coleção com `inverse = "true"` (O padrão de relacionamento one-to-many bidirecional idioma, por exemplo), podemos adicionar elementos a um saco ou uma lista sem a necessidade de inicializar (fetch) os elementos saco!

Isto é porque `IList.Add ()` ou `IList.AddRange ()` sempre deve ter êxito para um saco ou `IList` (Ao contrário de um conjunto). Isso pode fazer o seguinte código comum muito mais rápido.

```
P = pai (Parent) sess.Load (typeof (pai), id);
Criança c = Criança new ();
c.Parent = p;
p.Children.Add (c) / / não há necessidade de buscar a coleção!
sess.Flush ();
```

14.1.4. Um tiro apagar

Ocasionalmente, excluir elementos da coleção, um por um pode ser extremamente ineficiente. NHibernate não é completamente estúpida, por isso sabe que não deve fazer isso no caso de uma coleção de recém-vazia (se você chamassem `list.Clear ()`, Para exemplo). Neste caso, NHibernate irá emitir um único `APAGAR` e estamos a fazer!

Suponha que nós adicionamos um único elemento a uma coleção de tamanho vinte e então remover dois elementos. NHibernate irá uma questão `INSERIR` declaração e dois `APAGAR` declarações (a menos que a coleção é um saco). Este é certamente desejável-capaz.

No entanto, suponha que nós removemos dezoito elementos, deixando dois e depois adicionar-te novos elementos. Há duas maneiras possíveis para continuar

- eighteen excluir registros um a um e depois inserir três linhas
- remover toda a coleção (em um SQL `APAGAR`) E inserir todos os cinco elementos atuais (um por um)

NHibernate não é inteligente o suficiente para saber que a segunda opção é provavelmente mais rápido neste caso. (E seria provavelmente indesejável para NHibernate que ser inteligente, tal comportamento pode confundir triggers de banco de dados, etc)

Felizmente, você pode forçar este comportamento (ou seja, a segunda estratégia) a qualquer momento, descartando (ie. dereferencing) coleção original e retornando uma coleção recém instanciado com todos os elementos actuais. Isto pode ser

muito útil e poderosa de tempos em tempos.

Já mostramos como você pode usar a inicialização lenta para coleções persistentes no capítulo sobre col-mapeamentos de lição. Um efeito semelhante é possível para referências de objetos comuns, usando proxies. Temos também mencionou como NHibernate caches objetos persistentes no nível de um `ISession`. Cache mais agressivos estratégias podem ser configurados em uma base de classe por classe.

Na próxima seção, vamos mostrar como usar esses recursos, que podem ser utilizados para alcançar muito mais elevado desempenho, se necessário.

14.2. Proxies para a inicialização lenta

NHibernate implementa proxies de inicialização para objetos persistentes usando geração IL tempo de execução (através do ex-biblioteca Castle.DynamicProxy cellent).

O arquivo de mapeamento declara uma classe ou interface para usar como interface de proxy para essa classe. A abordagem recomendada

é especificar a classe em si:

```
<class name="Eg.Order" proxy="Eg.Order">
```

O tipo de tempo de execução do proxies será uma subclasse de `Ordem`. Observe que a classe deve implementar um proxy-de construtor de falha com pelo menos visibilidade protegido e que todos os métodos, propriedades e eventos da classe deve ser declarada `virtual`.

Existem alguns truques que você deve saber quando extender essa abordagem para classes polimórficas, por exemplo.

```
<class name="Eg.Cat" proxy="Eg.Cat">
    .....
    <subclass name="Eg.DomesticCat" proxy="Eg.DomesticCat">
        .....
        <Subclasse />
    </ Class>
```

Em primeiro lugar, os casos de `Gato` nunca serão concretos para `DomesticCat`, Mesmo se a instância de base é uma instância de `DomesticCat`.

```
Cat = gato (Cat) Session.load (typeof (Cat), id) // instanciar um proxy (não atingiu o db)
if (cat.IsDomesticCat) // acertar o db para inicializar o proxy
{
    DomesticCat dc = cat (DomesticCat); // Erro!
    ....
}
```

Em segundo lugar, é possível quebrar o proxy ==.

```
Cat = gato (Cat) Session.load (typeof (Cat), id);           // Instanciar um proxy Cat
DomesticCat dc =                                         // Necessário procuração DomesticCat novo!
    (DomesticCat) Session.load (typeof (DomesticCat), id); // False
Console.Out.WriteLine (cat == dc);
```

No entanto, a situação não é tão ruim como parece. Mesmo que agora temos duas referências a diferentes objetos proxy, a instância de base ainda será o mesmo objeto:

```
cat.Weight = 11,0 // acertar o db para inicializar o proxy
Console.Out.WriteLine (dc.Weight) // 11,0
```

Terceiro, você não pode usar um proxy para um selado classe ou uma classe com qualquer selado ou nãovirtual métodos.

Finalmente, se o seu objeto persistente adquirir quaisquer recursos durante a instanciação (em inicializadores ou padrão construtor), então esses recursos também serão adquiridos pelo proxy. A classe de proxy é uma subclasse da classe persistente.

Estes problemas são todos devido a limitações fundamentais em .NET modelo de herança simples. Se você deseja evitar esses problemas em suas classes persistentes devem cada implementar uma interface que declara os seus métodos de negócios. Você deve especificar essas interfaces no arquivo de mapeamento. por exemplo.

```
<class name="Eg.Cat" proxy="Eg.ICat">
    .....
    <subclass name="Eg.DomesticCat" proxy="Eg.IDomesticCat">
        .....
        <Subclasse />
    </ Class>
```

onde Gato implementa a interface ICat e DomesticCat implementa a interface IDomesticCat. Em seguida, proxies para instâncias de Gato e DomesticCat podem ser devolvidos por Load () ou Enumeráveis (). (Note que Find () não retornar proxies.)

```
Cat = iCat (ICAT) Session.load (typeof (Cat), catid);
IEnumarable en = session.Enumerable ("de gato em Eg.Cat classe onde cat.Name = 'fritz'");
en.MoveNext ();
ICat fritz = en.Current (ICAT);
```

Relacionamentos também são preguiçosamente inicializado. Isto significa que você deve declarar todas as propriedades para ser do tipo ICat, Não Gato.

Fazer certas operações não requer inicialização de proxy

- Equals(a,classe persistente não sobrescrever Equals ())
- GetHashCode (), Se a classe persistente não sobrescrever GetHashCode ()
- O método getter do identificador (se a classe não usa um assessor personalizada para a propriedade identificador)

NHibernate irá detectar classes persistentes que se sobreponham Equals () ou GetHashCode () .

Exceções que ocorrem durante a inicialização de um proxy estão envoltos em uma LazyInitializationException.

Às vezes precisamos garantir que o proxy ou coleção é inicializado antes de fechar o ISession. É claro, podemos always forçar a inicialização chamando cat.Sex ou cat.Kittens.Count, Por exemplo. Mas isso é confuso para os leitores do código e não é conveniente para códigos genéricos. Os métodos estáticos NHibernateUtil.Initialize () e NHibernateUtil.IsInitialized () proporcionar a aplicação com uma maneira conveniente de trabalhar com coleções lazyily inicializado ou proxies. NHibernateUtil.Initialize (gato) forçará a initialization de um proxy, gato, Enquanto a sua ISession ainda está em aberto. NHibernateUtil.Initialize (cat.Kittens) tem um efeito similar para a recolha de gatinhos.

14.3. Usando busca em lote

NHibernate pode fazer uso eficiente de busca em lote, isto é, NHibernate pode carregar vários proxies não inicializada se um proxy é acessado. A busca em lote é uma otimização para a estratégia de carregamento lento. Há duas maneiras você pode usar busca em lote: na classe e nível do conjunto.

Busca em lote para classes / entidades é mais fácil de entender. Imagine que você tem a seguinte situação em tempo de execução: Você tem 25 Gato instâncias carregadas em um ISession, Cada Gato tem uma referência ao seu Proprietário, Um Pessoa. O Pessoa classe é mapeada com um proxy, lazy = "true". Se você agora percorrer todos os gatos e obter o Proprietário de cada um,

NHibernate irá por padrão executar 25 SELECCIONE declarações, para recuperar os proprietários proxy. Você pode melhorar esse comportamento especificando um batch-size no mapeamento de Pessoa:

```
<class name="Person" lazy="true" batch-size="10"> classe ...</>
```

NHibernate irá executar agora apenas três consultas, o padrão é 10, 10, 5. Você pode ver que busca em lote é uma acho que cego, na medida em que a otimização do desempenho vai, depende do número de procurações utilizadas em uma particular ISession.

Você também pode habilitar busca em lote de coleções. Por exemplo, se cada Pessoa tem uma coleção de preguiçosos Gatos, e 10 pessoas estão atualmente carregados no ISession, Iterando por todas as pessoas serão gerados 10 SELECCIONES, uma para cada leitura de Person.Cats. Se você habilitar busca em lote para o Gatos coleção no mapeamento de Perfil, NHibernate pode pré-carga das coleções:

```
name="Person"> <class
    <set name="Cats" lazy="true" batch-size="3">
        ...
    </set>
</class>
```

Com um batch-size de 3, NHibernate irá carregar 3, 3, 3, 1 em 4 coleções SELECCIONES. Novamente, o valor do atributo depende do número esperado de coleções não inicializadas em um determinado ISession.

A busca em lote de coleções é particularmente útil se você tem uma árvore encadeada de itens, ie. o projeto de lei típica de materiais padrão.

14.4. O cache de segundo nível

A NHibernate ISession é um cache de nível de transação de dados persistentes. É possível configurar um cluster ou nível de processo (ISessionFactoryNível cache) em uma classe por classe e por coleta de coleta de base. Você pode até ligar em um cache cluster. Tenha cuidado. Caches não estão cientes das mudanças feitas no armazenamento persistente, por outra aplicação (embora possam ser configurados para expirar regularmente dados em cache). Em NHibernate 1.0 segundo cache de nível não funciona corretamente em combinação com transações distribuídas.

Por padrão, NHibernate usa HashtableCache para o processo de nível de cache. Você pode escolher uma implementação, especificando o nome de uma classe que implementa NHibernate.Cache.ICacheProvider usando a propriedade hibernate.cache.provider_class.

Tabela 14.1. Provedores de cache

Esconderijo	Classe de provedor	Tipo	Cluster Seguro	Cache de Consultas Suportados
Hashtable (Não destinados para produção de uso)	NHibernate.Cache.HashtableCacheProvider	memória		sim
ASP.NET Esconderijo (System.Web.NHibernate.Caches.SysCache) Cache	NHibernate.Caches.SysCache.SysCacheProvider, (System.Web.NHibernate.Caches.SysCache) Cache	memória		sim
Predomínio Esconderijo	NHibernate.Caches.Prevalence.PrevalenceCacheP	memória, disco		sim

Esconderijo	Classe de provedor	Tipo	Cluster Seguro	Cache de Consultas Suportados
	rovider, NHibernate.Caches.Prevalence			

14.4.1. Mapeamentos de cache

O `<cache>` elemento de um mapeamento de classe ou coleção tem a seguinte forma:

```
Cache <
    uso = "read-write | não seja estrita-read-write | read-only"
/>
```

(1) uso especifica a estratégia de cache: leitura e escrita, não seja estrita-leitura e escrita ou read-only

Como alternativa (de preferência?), Você pode especificar `<class-cache>` e `<collection-cache>` elementos em `hibernate.cfg.xml`.

O uso atributo especifica uma estratégia de simultaneidade cache.

14.4.2. Estratégia: somente leitura

Se sua aplicação precisa ler, mas nunca modificar instâncias de uma classe persistente, um `read-only` cache pode ser utilizada. Esta é a estratégia mais simples e melhor desempenho. É ainda perfeitamente seguro para uso em um cluster.

```
<class name="Eg.Immutable" mutable="false">
    <cache usage="read-only"/>
    ...
</ Class>
```

14.4.3. Estratégia: leitura / gravação

Se o aplicativo precisa atualizar os dados, um `leitura e escrita` cache pode ser apropriado. Esta estratégia de cache deve nunca ser usado se o nível de isolamento serializável operação é necessária. Se você quiser usar esta estratégia em um cluster, você deve garantir que a implementação de cache suportava locking. Os provedores de built-in de cache do não.

```
<Nome da classe = "eg.Cat" .... >
    <cache usage="read-write"/>
    ...
    <Nome do conjunto = "gatinhos" ... >
        <cache usage="read-write"/>
    ...
</ Set>
</ Class>
```

14.4.4. Estratégia: não seja estrita de leitura / gravação

Se o aplicativo precisa apenas ocasionalmente para atualizar dados (ou seja, se ele é extremamente improvável que duas transações tentaria atualizar o mesmo item simultaneamente) e de isolamento da transação estrito não é necessário, uma `não seja estrita-leitura e escrita` cache pode ser apropriado.

A tabela a seguir mostra que os fornecedores são compatíveis com as estratégias que a concorrência.

Tabela 14.2. Suporte Cache estratégia de simultaneidade

Esconderijo	read-only	não seja estrita-leitura e escrita	leitura e escrita	
Hashtable (não intendiam para a pro-use ção)	sim	sim	sim	
SysCache	sim	sim	sim	
PrevalenceCache	sim	sim	sim	

Consulte o Capítulo 20, **NHibernate.Caches** para mais detalhes.

14.5. Gestão do `ISession` Esconderijo

Sempre que você passar um objeto para `Save ()`, `Update ()` ou `SaveOrUpdate ()` e sempre que você recuperar um objeto-nos `ing Load ()`, `Find ()`, `Enumeráveis ()` Ou `Filter ()`, Esse objeto é adicionado ao cache interno do `ISession`. Quando `Flush ()` posteriormente é chamado, o estado desse objeto será sincronizado com o banco de dados. Se você fizer Não quero isso de sincronização para ocorrer ou se você está processando um grande número de objetos e necessidade de gerenciar memória de forma eficiente, o `Evict ()` método pode ser usado para remover o objeto e suas coleções a partir do cache.

```
Gatos I Enumerable = sess.Enumerable ("de Eg.Cat como gato") / / um resultado enorme
foreach (cat gato em gatos)
{
    DoSomethingWithACat (cat);
    sess.Evict (cat);
}
```

NHibernate irá expulsar entidades associadas automaticamente se a associação é mapeado com `cascade = "all"` ou `cascade = "all-delete-orphan"`.

O `ISession` também fornece uma `Contains ()` método para determinar se uma instância pertence ao cache de sessão.

Completamente expulsar todos os objetos do cache de sessão, chamada `ISession.Clear ()`

Para o cache de segundo nível, há métodos definidos na `ISessionFactory` para expulsar o estado em cache de um exemplo, classe inteira, instância de coleção ou função coleção inteira.

14.6. Cache de Consultas

Conjuntos de resultados da consulta também pode ser armazenada em cache. Isto só é útil para consultas que são executadas com freqüência com o mesmo para-

metros. Para usar o cache de consultas você deve primeiro habilitá-lo definindo a propriedade `hibernate.cache.use_query_cache = true`. Isso faz com que a criação de duas regiões cache - uma consulta que mantém em cache conjuntos de resultados (`NHibernate.Cache.IQueryCache`), A realização de timestamps outras mais recentes atualizações para consultado

tabelas (`NHibernate.Cache.UpdateTimestampsCache`). Note que o cache de consultas não coloca em cache o estado de qualquer entidades no conjunto de resultados, mas apenas valores de identificador de caches e resultados de tipo de valor. Assim, o cache de consultas é normalmente

usado em conjunto com o cache de segundo nível.

A maioria das consultas não beneficiam de caching, então por consultas padrão não são armazenados em cache. Para habilitar o cache, chamada

`IQuery.SetCacheable (true)`. Este convite permite a consulta a olhar para os resultados de cache existente ou adicionar seus resultados

o cache quando ele é executado.

Se você precisar de um controle refinado sobre as políticas de expiração da cache de consultas, você pode especificar uma região cache chamado para uma determinada consulta pelo telefone `IQuery.SetCacheRegion ()`.

```
Blogs IList sess.CreateQuery = ("do blog Blog onde blog.Blogger =: blogger")
    . SetEntity ("blogger", blogger)
    . SetMaxResults (15)
    . SetCacheable (true)
    . SetCacheRegion ("FrontPages")
    . List ();
```

Se a consulta deve forçar uma atualização de sua região cache de consultas, você pode chamar `IQuery.SetForceCacheRefresh ()` para

verdadeiro. Isso é particularmente útil nos casos em que os dados subjacentes podem ter sido atualizados através de um processo separado

(Ou seja, não modificados através NHibernate) e permite que o aplicativo de atualização seletivamente as regiões cache de consultas

com base em seu conhecimento desses eventos. Esta é uma alternativa ao despejo de uma região cache de consultas. Se você precisa controle refinado de atualização para muitas consultas, use esta função ao invés de uma nova região para cada consulta.

Capítulo 15. Guia Toolset

Engenharia de ida e volta com NHibernate é possível usando um conjunto de ferramentas de linha de comando mantida como parte do NHibernate projeto, juntamente com o apoio NHibernate construído em várias ferramentas de geração de código (MyGeneration, CodeSmith, ObjectMapper, AndroMDA).

O pacote NHibernate principal vem com a ferramenta mais importante (que pode até mesmo ser utilizado a partir de "dentro" NHibernate on-the-fly):

- Geração de esquema DDL a partir de um arquivo de mapeamento (aka `SchemaExport,hbm2ddl`)

Outras ferramentas diretamente fornecido pelo projeto NHibernate são entregues com um pacote separado, `NHibernateContrib`. Este pacote inclui ferramentas para as seguintes tarefas:

- C # a partir de fonte de geração de um arquivo de mapeamento (aka `hbm2net`)
- mapeamento de geração do arquivo de .NET marcados com atributos (`NHibernate.Mapping.Attributes` Ou NHMA para o short)

Ferramentas de terceiros, com o apoio NHibernate são:

- CodeSmith, MyGeneration e ObjectMapper (geração de mapeamento de arquivo de um esquema de banco de dados existente)
- AndroMDA (MDA código de abordagem (Model-Driven Architecture) para gerar as classes persistentes da Diagramas UML e sua representação XML / XMI)

Estas ferramentas de terceiro não são documentadas nesta referência. Por favor, consulte o site NHibernate para cima to-date informações.

15.1. Geração de Esquema

O esquema gerado inclui restrições de integridade referencial (chaves primárias e estrangeiras) para a entidade e recolher os ção tabelas. Tabelas e sequências também são criados para geradores identificador mapeada.

Você preciso especificar um `SQL Dialeto` através do `hibernate.dialect` propriedade ao utilizar esta ferramenta.

15.1.1. Personalizando o esquema

Muitos elementos de mapeamento NHibernate definir um atributo opcional chamado `comprimento`. Você pode ajustar o comprimento de um coluna com este atributo. (Ou, para tipos numéricos / decimais de dados, a precisão).

Algumas marcas também aceitar uma `não-nulo` atributo (para gerar uma `NOT NULL` restrição sobre colunas da tabela) e um único atributo (para a geração de `UNIQUE` restrição sobre colunas da tabela).

Algumas tags aceitar uma `índice` atributo para especificar o nome de um índice para essa coluna. A `unique-chave` atributo pode ser usado para colunas de grupo em uma única restrição unidade de chave. Atualmente, o valor especificado do `unique-chave` atributo é não usado para nomear a restrição, apenas para agrupar as colunas no arquivo de mapeamento.

Exemplos:

```
<property name="Foo" type="String" length="64" not-null="true"/>
```

```
<many-to-one name="Bar" foreign-key="fk_foo_bar" not-null="true"/>

<element column="serial_number" type="Int64" not-null="true" unique="true"/>
```

Alternativamente, esses elementos também aceitar uma criança `<element>` elemento. Isto é particularmente útil para várias colunas tipos:

```
<property name="Foo" type="string">
  <column name="foo" length="64" not-null="true" sql-type="text"/>
</ Property>

<property name="Bar" type="My.CustomTypes.MultiColumnType, My.CustomTypes"/>
  <column name="fee" not-null="true" index="bar_idx"/>
  <column name="fi" not-null="true" index="bar_idx"/>
  <column name="fo" not-null="true" index="bar_idx"/>
</ Property>
```

O `sql-type` atributo permite que o usuário substitua o mapeamento padrão do tipo NHibernate para SQL datatype.

O `verificar` atributo permite especificar uma restrição de verificação.

```
<property name="Foo" type="Int32">
  <column name="foo" check="foo> 10 "/>
</ Property>

<Nome da classe = "Foo" table = "foos" check = "bar <100.0">
  ...
  <property name="Bar" type="Single"/>
</ Class>
```

Tabela 15.1. Sumário

Atributo	Valores	Interpretação
comprimento	número	tamanho da coluna / precisão decimal
não-nulo	true false	specifies que a coluna devem ser não-nulo
único	true false	especifica que a coluna deve ter uma restrição de unicidade
índice	index name	especifica o nome de um índice (multi-coluna)
unique-chave	unique_key_name	especifica o nome de uma restrição de multi-coluna única
de chave estrangeira	foreign key name	especifica o nome da restrição de chave estrangeira gerado para uma associação, usá-lo em <code><one-to-one></code> , <code><many-to-one></code> , <code><key></code> , e elementos de mapeamento <code><many-to-many></code> . Note-se que <code>inverse = "true"</code> os lados não serão consideradas pela SchemaExporto.
sql-type	column_type	substitui o tipo de coluna padrão (atributo de <code><element></code> único elemento)
verificar	Expressão SQL	criar uma restrição de verificação SQL em qualquer coluna ou da tabela

15.1.2. Execução da ferramenta

O SchemaExport ferramenta grava um script DDL para a saída padrão e / ou executa os comandos DDL.

```
java -cp hibernate_classpaths net.sf.hibernate.tool.hbm2ddl.SchemaExport opções mapping_files
```

Tabela 15.2. SchemaExport Opções de linha de comando

Opção	Descrição
- Quiet	não a saída do script para stdout
- Queda	apenas uma gota as tabelas
- Texto	não exportar para o banco de dados
- Output = my_schema.ddl	a saída do script ddl para um arquivo
- Config = hibernate.cfg.xml	ler Hibernate configuração de um arquivo XML
- Properties = hibernate.properties	ler as propriedades de um arquivo de banco de dados
- Formato	formatar o SQL gerado muito bem no roteiro
- Delimitador = x	definir um delimitador de fim de linha para o script

Você pode até mesmo incorporar SchemaExport na sua aplicação:

```
Cfg configuração = ....;
nova SchemaExport (cfg) criar (false, true);
```

15.1.3. Propriedades

Propriedades de banco de dados pode ser especificado

- como propriedades do sistema com -D<property>
- em hibernate.properties
- em um arquivo de propriedades nomeadas com -Propriedades

As propriedades necessárias são:

Tabela 15.3. Connection Properties SchemaExport

Nome da propriedade	Descrição
hibernate.connection.driver_class	jdbc classe do driver
hibernate.connection.url	url jdbc
hibernate.connection.username	usuário de banco de dados
hibernate.connection.password	senha do usuário
hibernate.dialect	dialeto

15.1.4. Usando Ant

Você pode chamar `SchemaExport` a partir do seu script Ant de compilação:

```

name="schemaexport"> <target
  <Nome taskdef = "SchemaExport"
    classname = "net.sf.hibernate.tool.hbm2ddl.SchemaExportTask"
    classpathref = "class.path" />

  <SchemaExport
    properties = "hibernate.properties"
    calma = "no"
    text = "não"
    queda = "no"
    delimitador = "";
    output = "esquema export.sql">
    <fileset dir="src">
      <include name="**/*.hbm.xml"/>
    <Conjunto de arquivos />
  </ SchemaExport>
</ Target>
```

Se você não especificar `propriedades` ou um `configuração` arquivo, o `SchemaExportTask` tentará usar projeto Ant normais propriedades em seu lugar. Em outras palavras, se você não quiser ou precisar de uma configuração externa ou arquivo de propriedades, você pode colocar `hibernate.*` propriedades de configuração em seu `build.xml` ou `build.properties`.

15.1.5. Incremental atualizações de esquema

O `SchemaUpdate` ferramenta irá atualizar um esquema existente com "incremental" muda. Note-se que `SchemaUpdate` depende muito da API de metadados JDBC, por isso não irá funcionar com todos os drivers JDBC.

`java -cp hibernate_classpaths net.sf.hibernate.tool.hbm2ddl.SchemaUpdate opções mapping_files`

Tabela 15.4. SchemaUpdate Opções de linha de comando

Opção	Descrição
<code>- Quiet</code>	não a saída do script para stdout
<code>- Properties = hibernate.properties</code>	ler as propriedades de um arquivo de banco de dados

Você pode incorporar `SchemaUpdate` na sua aplicação:

```
Cfg configuração = ....;
. SchemaUpdate nova (cfg) executar (false);
```

15.1.6. Usando Ant para atualizações de esquema incremental

Você pode chamar `SchemaUpdate` a partir do script Ant:

```

name="schemaupdate"> <target
  <Nome taskdef = "SchemaUpdate"
    classname = "net.sf.hibernate.tool.hbm2ddl.SchemaUpdateTask"
    classpathref = "class.path" />

  <SchemaUpdate
    properties = "hibernate.properties"
    quiet = "no">
    <fileset dir="src">
      <include name="**/*.hbm.xml"/>
    <Conjunto de arquivos />
```

```
</ SchemaUpdate>
</ Target>
```

15.2. Geração de Código

O gerador de código Hibernate pode ser usado para gerar classes de implementação Java do esqueleto de um Hibernate mapeamento de arquivo. Essa ferramenta está incluída no pacote de extensões Hibernate (um download separado).

`hbm2java` analisa os arquivos de mapeamento e gera totalmente funcional arquivos de origem Java a partir destes. Assim, com `hbm2java` pode-se "apenas" fornecer o `.Hbm` arquivos, e então não se preocupe com hand-writing/coding os arquivos Java.

```
java -cp hibernate_classpaths net.sf.hibernate.tool.hbm2java.CodeGenerator opções mapping_files
```

Tabela 15.5. Código Gerador de Linha Opções de comando

Opção	Descrição
<code>- Output =output_dir</code>	diretório raiz para o código gerado
<code>- Config =config_file</code>	arquivo opcional para configurar hbm2java

15.2.1. O arquivo de configuração (opcional)

O arquivo de configuração fornece uma maneira de especificar vários "renderizadores" para o código fonte e para declarar `<meta>` atributos que é "global" no escopo. Veja mais sobre isso no `<meta>` seção atributo.

```
<codegen>
  <meta attribute="implements"> codegen.test.IAuditable </ meta>
  <generate rendererm="net.sf.hibernate.tool.hbm2java.BasicRenderer"/>
  <Gerar
    pacote = "autofinders.only"
    sufixo = "Finder"
    renderer = "net.sf.hibernate.tool.hbm2java.FinderRendererm" />
</ Codegen>
```

Este arquivo de configuração declara um atributo meta global de "implements" e especificar dois processadores, o padrão (BasicRenderer) e um processador que gera Finder (Veja mais em "geração do Finder Basic" abaixo).

O renderizador segundas é fornecido com um pacote e atributo sufixo.

O atributo pacote especifica que os arquivos-fonte gerado a partir deste processador deve ser colocado aqui, em vez do escopo do pacote especificado no `.Hbm` arquivos.

O atributo especifica o sufixo sufixo para os arquivos gerados. Por exemplo, aqui um arquivo chamado `Foo.java` seria `FooFinder.java` em seu lugar.

Também é possível enviar para baixo parâmetros arbitrários para o torna adicionando `<param>` atributos para a `<generate>` elementos.

`hbm2java` atualmente tem suporte para um parâmetro tal, ou seja, `gerar-concrete-vazia-classes` que informa a BasicRenderer apenas gerar vazios classes concretas que estende uma classe base para todas as suas classes. O exemplo a seguir config.xml ilustrar esse recurso

```

<codegen>
  <generate prefix="Base" renderer="net.sf.hibernate.tool.hbm2java.BasicRenderer"/>
  <generate renderer="net.sf.hibernate.tool.hbm2java.BasicRenderer">
    <param name="generate-concrete-empty-classes"> true </ param>
    <param name="baseclass-prefix"> Base </ param>
  </ Generate>
</ Codegen>

```

Observe que esse config.xml configurar 2 (dois) representantes. Aquela que gera as classes base, e um segundo que só gera vazio classes concretas.

15.2.2. O meta atributo

O `<meta>` tag é uma maneira simples de anotar o `hbm.xml` com a informação, para que as ferramentas têm um lugar natural para armazenar / ler a informação que não está directamente relacionado com o núcleo Hibernate.

Você pode usar o `<meta>` tag para contar `hbm2java` apenas para gerar "protegido" setters, têm aulas sempre implemento de um determinado conjunto de interfaces ou mesmo tê-los estender uma classe base certas e até mais.

O exemplo a seguir:

```

name="Person"> <class
  <meta attribute="class-description">
    Javadoc para a classe Pessoa
    @ Author Frodo
  </ Meta>
  <meta attribute="implements"> IAuditable </ meta>
  <id name="id" type="long">
    <meta attribute="scope-set"> protegidos </ meta>
    <generator class="increment"/>
  </ Id>
  <property name="nome" type="string">
    <meta attribute="field-description"> O nome da pessoa </ meta>
  </ Property>
</ Class>

```

irá produzir algo como o seguinte (código encurtado para melhor compreensão). Observe a com-Javadoc
mento e os métodos estabelecidos de conservação:

```

/ Default / pacote

importação java.io.Serializable;
importação org.apache.commons.lang.builder.EqualsBuilder;
importação org.apache.commons.lang.builder.HashCodeBuilder;
importação org.apache.commons.lang.builder.ToStringBuilder;

/ **
 * Javadoc para a classe Pessoa
 * @ Author Frodo
 *
 */
public class Pessoa implements Serializable, IAuditable {

  / ** Campo identificador * /
  Long id público;

  / ** Campo nulo persistente * /
  String nome pública;

  / *** Construtor completa /
  Pessoa pública (java.lang.String name) {
    this.name name =;
  }
}

```

```

    / ** * Construtor padrão /
    Pessoa pública () {
    }

    pública java.lang.Long getId () {
        retorno this.id;
    }

    protected void setId (java.lang.Long id) {
        this.id = id;
    }

    / **
     * O nome da pessoa
     */
    getName java.lang.String pública () {
        retorno this.name;
    }

    setName public void (java.lang.String name) {
        this.name name =;
    }

}

```

Tabela 15.6. Suportados meta tags

Atributo	Descrição
classe descrição	inserido no javadoc para as classes
campo de descrição-	inserido no javadoc para os campos / propriedades
interface	Se uma interface verdade é gerado, em vez de uma classe.
implementa	interface da classe deve implementar
estende-se	classe a classe deve estender (ignorado para subclasses)
gerado classe	ignorar o nome da classe real gerado
escopo de classe	alcance para a classe
âmbito set-	alcance para o método setter
escopo de obter	alcance para o método getter
alcance em campo	espaço para campo real
use-in-tostring	incluir essa propriedade no <code>toString()</code>
implementar-iguais	incluir uma <code>equals()</code> e <code>hashCode()</code> método nesta classe.
use-in-iguais	incluir essa propriedade no <code>equals()</code> e <code>hashCode()</code> métodos de od.
obrigado	adicionar suporte para uma propriedade <code>PropertyChangeListener</code>
constrangido	apoio <code>vetoChangeListener</code> ligado + de um imóvel
gen-propriedade	propriedade não será gerado se falsa (use com cuidado)
propriedade de tipo-	Substitui o tipo padrão de propriedade. Use isso com qualquer tag de para especificar o tipo de concreto em vez de apenas objeto.

Atributo	Descrição
classe code-extra-importação	Código extra que será inserido no final da classe
finder método-sessão método de-	Importação extra que será inserido no final de todas as outras importações ver "gerador finder Basic" abaixo
	ver "gerador finder Basic" abaixo

Atributos declarados através do `<meta>` tag são por padrão "herdou" dentro de um `hbm.xml` arquivo.

O que significa isso? Isso significa que se você por exemplo quer ter todas as suas classes implementam `IAuditable` então você basta adicionar uma `<meta attribute="implements"> IAuditable </ meta>` no topo da `hbm.xml` arquivo, logo após `<hibernate-mapping>`. Agora todas as classes definidas no que `hbm.xml` arquivo irá implementar `IAuditable!` (Exceto se um classe também tem um "implements" atributo meta, porque locais meta tags especificado sempre se sobrepõe / substitui qualquer herdados meta tags).

Nota: Isto se aplica a todos `<meta>`-Tags. Assim, por exemplo, também pode ser usado para especificar que todos os campos devem ser declarar protegida, em vez do padrão privado. Isto é feito através da adição `<Metaattribute = "escopo campo"> protegida </ meta>` por exemplo, em apenas sob o `<class>` tag e todos os campos da classe será protegida.

Para evitar ter um `<meta>`-Tag herdou então você pode simplesmente especificar `herdar = "false"` para o atributo, por exemplo, `<meta attribute="scope-class" inherit="false"> public abstract </ meta>` irá restringir a "classe-escopo" para a classe atual, não o subclasses.

15.2.3. Gerador finder básico

Agora é possível ter `hbm2java` gerar básica finders para propriedades Hibernate. Isso exige duas coisas em o `hbm.xml` arquivos.

A primeira é uma indicação de quais campos você deseja gerar para finders. Você indica que, com um bloco de meta dentro de uma marca de propriedade, tais como:

```
<property name="nome" column="name" type="string">
  <meta attribute="finder-method"> findByName </ meta>
</ Property>
```

O nome do método finder será o texto incluído na meta tags.

A segunda é criar um arquivo de configuração para `hbm2java` do formato:

```
<codegen>
  <generate renderer="net.sf.hibernate.tool.hbm2java.BasicRenderer"/>
  <generate suffix="Finder" renderer="net.sf.hibernate.tool.hbm2java.FinderRenderer"/>
</ Codegen>
```

E depois use o param para `hbm2java - config = xxx.xml` onde `xxx.xml` é o arquivo de configuração que você acabou de criar.

Um parâmetro opcional é meta tag no nível de classe do formato:

```
<meta attribute="session-method">
  com.whatever.SessionTable.getSession () getSession () .;
</ Meta>
```

Qual seria a maneira em que você começa as sessões se você usar o Sessão Thread Local padrões (documentados em Design Patterns área do site Hibernate).

15.2.4. Velocidade baseado renderer / gerador

Agora é possível usar a velocidade como um mecanismo de renderização alternativa. O config.xml following mostra como hbm2java configurar para utilizar a sua velocidade renderizador.

```
<codegen>
<generate renderer="net.sf.hibernate.tool.hbm2java.VelocityRenderer">
  <param name="template"> pojo.vm </ param>
  </ Generate>
</ Codegen>
```

O parâmetro nomeado `modelo` é um caminho de recurso para o arquivo de macro velocidade que você deseja usar. Este arquivo deve ser disponível através do classpath para hbm2java. Assim, não se esqueça de adicionar o diretório onde está localizado `pojo.vm` para o seu formiga tarefa ou script shell. (O local padrão é `. / Tools / src / velocidade`) Estar ciente de que a corrente `pojo.vm` gera apenas as partes mais básicas do feijão java. Não é tão completa e rico em recursos como o renderizador padrão - principalmente um monte de `meta` tags não são suportados.

15.3. Geração do arquivo de mapeamento

Um arquivo de mapeamento esquelético pode ser gerado a partir compilado classes persistentes usando um utilitário de linha de comando chamado `MapGenerator`. Este utilitário é parte do pacote de extensões Hibernate.

O gerador de mapeamento Hibernate oferece um mecanismo para a produção de mapeamentos de classes compiladas. Ele usa Java reflexão para encontrar **propriedades** e usa heurísticas para adivinhar um mapeamento adequado do tipo de propriedade. O mapeamento gerado é destinado a ser um ponto de partida só. Não há nenhuma maneira de produzir um mapa completo Hibernate-de ping sem entrada extra do usuário. No entanto, a ferramenta faz tirar um pouco do trabalho repetitivo "grunt" envolvidos na produção de um mapeamento.

As aulas são adicionados à um mapeamento de cada vez. A ferramenta irá rejeitar as classes que os juízes são não são **Hibernar persistente**.

Para ser **Hibernate persistente** uma classe

- não deve ser um tipo primitivo
- não deve ser um array
- não deve ser uma interface
- não deve ser uma classe aninhada
- deve ter um construtor padrão (sem argumentos).

Note que as interfaces e classes aninhadas são realmente persistente pelo Hibernate, mas isso não seria normalmente intendiam pelo usuário.

`MapGenerator` vai subir a cadeia de superclasse de todas as classes acrescentou tentando adicionar como muitos Hibernate persistir-superclasses capazes quanto possível para a mesa mesmo banco de dados. A busca pára assim que uma propriedade é encontrada que tem um nome que aparece em uma lista de candidato nomes UID.

A lista padrão de nomes de candidatos propriedade UID é: `uid,UID,id,ID,chave,KEY,pk,PK`.

Propriedades são descobertas quando há dois métodos na classe, um setter e um getter, onde o tipo de

único argumento setter é o mesmo que o tipo de retorno do getter argumento zero, e os retornos setter vazio. Além disso, o nome do setter deve começar com a string `conjunto` e o nome do getter começa com `obter` ou o getter nome começa com `e` eo tipo da propriedade é boolean. Em ambos os casos, o restante de seus nomes devem corresponder. Esta parte de correspondência é o nome da propriedade, exceto que o caráter inicial da propriedade nome é feita minúsculas se a segunda letra é minúscula.

As regras para determinar o tipo de banco de dados de cada propriedade são as seguintes:

1. Se o tipo de Java é `Hibernate.basic ()`, Então a propriedade é uma coluna simples desse tipo.
2. Para `hibernate.type.Type` tipos de costume e `PersistentEnum` uma coluna simples é usado também.
3. Se o tipo de propriedade é um array, então um array Hibernate é utilizado, e `MapGenerator` tentativas de refletir sobre o tipo de elemento da matriz.
Se a propriedade tem o tipo `java.util.List,java.util.Map`Ou `java.util.Set`, Então o Hi-correspondente
4. Bernate tipos são usados, mas `MapGenerator` não pode mais processar o interior destes tipos.
Se o tipo de propriedade é qualquer outra classe, `MapGenerator` adia a decisão sobre a representação do banco de dados até que todas as classes tenham sido processados. Neste ponto, se a classe foi descoberto através da superclasse
5. pesquisa descrita acima, então a propriedade é uma `many-to-one` associação. Se a classe tem todas as propriedades, então é uma `componente`. Caso contrário, é serializável, ou não persistente.

15.3.1. Execução da ferramenta

A ferramenta grava mapeamentos XML padrão para fora e / ou para um arquivo.

Ao invocar a ferramenta que você deve colocar suas classes compiladas no classpath.

```
java -cp hibernate_and_your_class_classpaths net.sf.hibernate.tool.class2hbm.MapGenerator opções
e classnames
```

Existem dois modos de operação: linha de comando ou interativo.

O modo interativo é selecionado, fornecendo o argumento única linha de comando – `Interagem`. Este modo fornece um console de resposta imediata. Através dela você pode definir o nome da propriedade UID para cada classe usando o `uid = xxx` comando onde `xxx` é o nome da propriedade UID. Alternativas de comando outras são simplesmente uma completamente qual- nome da classe cados, ou o comando feito que emite o XML e termina.

No modo de linha de comando os argumentos são as opções abaixo intercaladas com nomes de classe totalmente qualificado do classes a serem processados. A maioria das opções são destinadas a ser usado várias vezes; cada uso afeta posteriormente adicionado classes.

Tabela 15.7. Linha MapGenerator Opções de comando

Opção	Descrição
<code>- Quiet</code>	não a saída do Mapeamento O-R para stdout
<code>- Uid = SetUID</code>	definir a lista de UIDs candidato à uid singleton
<code>- Uid = adduid</code>	uid adicionar à frente da lista de UIDs candidato
<code>- Select =modo</code>	Modo de usar o modo de escolha <code>modo</code> (por exemplo, <code>distintas</code> ou <code>todos</code>) para posteriormente adicionada aulas
<code>- Profundidade = <small-int></code>	limitar a profundidade de recursão de dados de componentes para posteriormente adicionada aulas

Opção	Descrição
- Output = my_mapping.xml	saída do Mapeamento O-R para um arquivo
full.class.Name	adicionar a classe para o mapeamento
- Abstract =full.class.Name	veja abaixo

O interruptor abstrata dirige a ferramenta de gerador de mapas para ignorar classes específicas super assim que as classes comuns com herança não são mapeados para uma tabela grande. Por exemplo, considere estas hierarquias de classe:

Animal -> Mamífero -> Humanos

Animal -> Mamífero -> marsupial -> Kangaroo

Se o - Abstract interruptor é **não** usado, todas as classes serão mapeadas como subclasses de Animal, Resultando em um grande tabela contendo todas as propriedades de todas as classes, mais uma coluna discriminadora para indicar qual a subclasse é atualmente armazenados. Se Mamífero é marcado como abstrato, Humano e Marsupial serão mapeados para separar <class> declarations e armazenados em tabelas separadas. Canguru ainda será uma subclasse de Marsupial a menos que Marsupial também é marcado como abstrato.

Capítulo 16. Exemplo: Pai / Filho

Uma das primeiras coisas que os novos usuários tentam fazer com NHibernate é modelar um tipo de pai / filho-relação navio. Existem duas abordagens diferentes para isso. Por várias razões a abordagem mais conveniente, especialmente para novos usuários, é modelar tanto `Principal` e `Criança` como classes de entidade com uma `<one-to-many>` associação de `Parent` para `Criança`. (A abordagem alternativa é declarar o `Criança` como um `<composite-element>`.) Agora, ao que parece que a semântica padrão de uma associação de muitos para um (em NHibernate) são muito menos perto da semântica usual de uma relação pai / filho do que aqueles de um mapeamento de elemento composto. Vamos explicar como usar um bidirectional a associação de muitos com cascatas para modelar uma relação pai / filho de forma eficiente e elegantly. Não é nada difícil!

16.1. Uma nota sobre coleções

NHibernate coleções são consideradas uma parte lógica da sua entidade proprietária, nunca das entidades contidas. Esta é uma distinção crucial! Ela tem as seguintes consequências:

- Quando remover / adicionar um objeto de / para uma coleção, o número da versão do proprietário do conjunto é incrementado.
- Se um objeto que foi removido de uma coleção é uma instância de um tipo de valor (por exemplo, um elemento composto), que objeto deixará de ser persistente e seu estado será completamente removido do banco de dados. Da mesma forma, adicionando uma instância de tipo de valor à coleção fará com que seu estado para ser imediatamente persistente.
- Por outro lado, se uma entidade é removida de uma coleção (a associação de um-para-muitos ou muitos-para-muitos), não será excluída, por padrão. Este comportamento é completamente consistente - uma mudança para o estado interno da outra entidade não deve fazer com que a entidade associada a desaparecer! Da mesma forma, adicionando uma entidade a uma coleção não causa essa entidade se tornar persistente, por padrão.

Em vez disso, o comportamento padrão é que a adição de uma entidade a uma coleção apenas cria um link entre os dois entitatis, enquanto remove ele remove o link. Isto é muito apropriado para todos os tipos de casos. Quando não for apropriado, é preciso explicitar.

16.2. Bidirecional um-para-muitos

Suponha que começar com uma simples `<one-to-many>` associação de `Principal` para `Criança`.

```
<set name="Children">
  <key column="parent id" />
  <one-to-many class="Child" />
</ Set>
```

Se fôssemos para executar o seguinte código

```
Pai p = ....;
Criança c = Criança new ();
p.Children.Add (c);
Session.save (c);
Session.flush ();
```

NHibernate iria emitir duas instruções SQL:

- um `INSERIR` para criar o registro de `c`
- um `ATUALIZAÇÃO` para criar o link de `p` para `c`

Isto não só é ineficaz, mas também viola qualquer `NOT NULL` restrição sobre o `parent_id` coluna.

A causa subjacente é que o link (a chave estrangeira `parent_id`) De `p` para `c` não é considerado parte do estado do `Criança` objeto e, portanto, não é criado no `INSERIR`. Portanto, a solução é fazer a peça de ligação do `Criança` mapeamento.

```
<many-to-one name="Parent" column="parent_id" not-null="true"/>
```

(Nós também precisamos adicionar o `Principal` propriedade para a `Criança` classe).

Agora que o `Criança` entidade está a gerir o estado do link, nós dizemos a coleção não para atualizar o link. Usamos o `inverso` atributo.

```
<set name="Children" inverse="true">
  <key column="parent_id"/>
  <one-to-many class="Child"/>
</ Set>
```

O código a seguir seria usado para adicionar um novo `Criança`.

```
P = pai (Parent) Session.load (typeof (pai), pid);
Criança c = Criança new ();
c.Parent = p;
p.Children.Add (c);
Session.save (c);
Session.flush ();
```

E agora, apenas um SQL `INSERIR` seriam emitidas!

Para apertar um pouco as coisas, poderíamos criar um `AddChild ()` método de `Principal`.

```
AddChild public void (c Criança)
{
    c.Parent = this;
    children.Add (c);
}
```

Agora, o código para adicionar um `Criança` parece que

```
P = pai (Parent) Session.load (typeof (pai), pid);
Criança c = Criança new ();
p.AddChild (c);
Session.save (c);
Session.flush ();
```

16.3. Ciclo de vida em cascata

A chamada explícita para `Save ()` ainda é chato. Iremos abordar este usando cascatas.

```
<set name="Children" inverse="true" cascade="all">
  <key column="parent_id"/>
  <one-to-many class="Child"/>
</ Set>
```

Isto simplifica o código acima para

```
P = pai (Parent) Session.load (typeof (pai), pid);
Criança c = Criança new ();
p.AddChild (c);
Session.flush ();
```

Da mesma forma, não precisamos para iterar sobre os filhos ao salvar ou excluir um `Principal`. A seguir remove `p` e todos os seus filhos a partir do banco de dados.

```
P = pai (Parent) Session.load (typeof (pai), pid);
session.Delete (p);
Session.flush ();
```

No entanto, este código

```
P = pai (Parent) Session.load (typeof (pai), pid);
// Pega um filho fora do conjunto
ChildEnumerator IEnumarator p.Children.GetEnumerator = ();
childEnumerator.MoveNext ();
Criança c = (Child) childEnumerator.Current;

p.Children.Remove (c);
c.Parent = null;
Session.flush ();
```

não removerá `c` do banco de dados, ele só irá remover o link para `p` (E causar um `NOT NULL` viola-restriçãoção, neste caso). Você precisa explicitamente `Delete ()` o `Criança`.

```
P = pai (Parent) Session.load (typeof (pai), pid);
// Pega um filho fora do conjunto
ChildEnumerator IEnumarator p.Children.GetEnumerator = ();
childEnumerator.MoveNext ();
Criança c = (Child) childEnumerator.Current;

p.Children.Remove (c);
session.Delete (c);
Session.flush ();
```

Agora, no nosso caso, um `Criança` não pode realmente existir sem seu pai. Então, se nós remover um `Criança` da coleção, nós realmente quer que ele seja eliminado. Para isso, devemos usar `cascade = "all-delete-orphan"`.

```
<set name="Children" inverse="true" cascade="all-delete-orphan">
  <key column="parent id"/>
  <one-to-many class="Child"/>
</ Set>
```

Nota: mesmo que o mapeamento da coleção especifica `inverse = "true"`, Cascatas ainda são processados por iteração os elementos da coleção. Então, se você exigir que um objeto ser salvo, apagados ou atualizados pelo cascata, você deve adicionar à coleção. Não é suficiente para simplesmente definir seu pai.

16.4. Utilizando em cascata `Update ()`

Suponha que nós carregamos um `Principal` em um `ISession`, Fez algumas mudanças em uma ação UI e desejo de persistir esses mudanças em uma nova `ISession` (chamando `Update ()`). O `Principal` conterá um conjunto de crianças e, uma vez atualização em cascata é ativada, NHibernate precisa saber que as crianças são recém instanciado e que reressentir-se as linhas existentes na base de dados. Vamos supor que ambos os `Principal` e `Criança` tem (sintético) identificador adequada-laços do tipo `longo`. NHibernate irá utilizar o valor da propriedade identificador para determinar qual dos filhos são novos.

(Você também pode usar a versão ou a propriedade timestamp, veja Secção 9.4.2, "Atualizando objetos soltos").

O `unsaved` valor atributo é usado para especificar o valor do identificador de uma instância mais recente. Em NHibernate não é necessário especificar `unsaved` valor explicitamente.

O código a seguir serão atualizados `principal` e `criança` e inserir `newChild`.

```
/ / Pai e filho foram ambos carregados em uma sessão anterior
parent.AddChild (criança);
NewChild Criança = new ();
parent.AddChild (newChild);
Session.update (pai);
Session.flush ();
```

Bem, é tudo muito bem para o caso de um identificador gerado, mas que sobre identificadores atribuídos e composto identificadores? Isso é mais difícil, uma vez `unsaved` valor não pode distinguir entre um objeto recém instanciado (Com um identificador atribuído pelo usuário) e um objeto carregado em uma sessão anterior. Nestes casos, você vai provavelmente terá que dar uma dica NHibernate: ou

- definir uma `unsaved` valor em um <versão> ou <timestamp> mapeamento da propriedade para a classe.
- conjunto `unsaved valor = "none"` e explicitamente `Save ()` crianças recém instanciado antes de chamar `-Up data (pai)`
- conjunto `unsaved-value = "qualquer"` e explicitamente `Update ()` crianças previamente persistente antes de chamar `-Up data (pai)`

`null` é o padrão `unsaved` valor para identificadores atribuídos, `nenhum` é o padrão `unsaved` valor para o compósito identificadores.

Há uma outra possibilidade. Há um novo `IInterceptor` método chamado `IsUnsaved ()` que permite que o aplicatura implementar sua própria estratégia para distinguir objetos recém instanciado. Por exemplo, você poderia definir uma classe base para suas classes persistentes.

```
Persistente public class
{
    private bool _saved = false;

    OnSave public void ()
    {
        _saved = true;
    }

    public void OnLoad ()
    {
        _saved = true;
    }

    .....

    public bool IsSaved
    {
        get {return _saved;}
    }
}
```

(A `salvo` propriedade não é persistente.) Agora, implementar `IsUnsaved ()`, Juntamente com `OnLoad ()` e `OnSave ()` como segue.

```
pública objeto IsUnsaved (entidade objeto)
{
    if (entidade é persistente)
```

```

{
    !. return ((Persistente) entidade) IsSaved;
}
outro
{
    return null;
}

}

OnLoad bool público (entidade objeto,
    identificação do objeto,
    objeto [estado],
    string [] propertyNameNames,
    ITipo tipos [])
{
    if (entidade é persistente) ((Persistente) entidade) OnLoad () .;
    return false;
}

public boolean OnSave (entidade objeto,
    identificação do objeto,
    objeto [estado],
    string [] propertyNameNames,
    ITipo tipos [])
{
    if (entidade é persistente) ((Persistente) entidade) OnSave () .;
    return false;
}

```

16.5. Conclusão

Há um pouco para digerir aqui e pode parecer confuso primeira vez ao redor. No entanto, na prática, tudo funciona muito bem. A maioria dos aplicativos NHibernate usar o padrão pai / filho em muitos lugares.

Mencionamos uma alternativa no primeiro parágrafo. Nenhuma das questões acima existe no caso de `<composite-element>` mapeamentos, que têm exatamente a semântica de uma relação pai / filho. Infelizmente, existem duas grandes limitações para as classes elemento composto: elementos compostos não podem possuir coleções, e não deve ser o filho de qualquer entidade que não seja o pai único. (No entanto, eles pode ter um surrogado com chave primária, utilizando uma `<idbag>` mapeamento.)

Capítulo 17. Exemplo: Aplicação Weblog

17.1. Classes persistentes

As classes persistentes representam um weblog, e um item publicado em um weblog. Eles devem ser modelado como um standard relação pai / filho, mas vamos usar um saco ordenou, em vez de um conjunto.

```
using System;
using System.Collections;

namespace Eg
{
    Blog public class
    {
        _id longo privado;
        _name private string;
        privada _items IList;

        Id pública virtual longa
        {
            get {return _id;}
            set {_id = valor;}
        }

        public virtual Itens IList
        {
            get {_items return;}
            set {_items = valor;}
        }

        Nome cadeia pública virtuais
        {
            get {_name return;}
            set {_name = valor;}
        }
    }
}
```

```
using System;

namespace Eg
{
    public class BlogItem
    {
        _id longo privado;
        _dateTime DateTime privado;
        private string _TEXT;
        _title private string;
        _Blog blog privada;

        Blog Blog públicas virtuais
        {
            get {return _blog;}
            set {_blog = valor;}
        }

        pública DateTime DateTime virtuais
        {
            get {return _dateTime;}
            set {_dateTime = valor;}
        }

        Id pública virtual longa
        {
            get {return id;}
        }
}
```

```

        set {_id = valor;}
    }

Texto cadeia pública virtuais
{
    get {return _TEXT;}
    set {_TEXT = valor;}
}

Título cadeia pública virtuais
{
    get {return _title;}
    set {_title = valor;}
}

}
}

```

17.2. Mapeamentos Hibernate

Os mapeamentos de XML devem agora ser bastante simples.

```

<? Xml version = "1.0"?>
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
assembly = "Eg" namespace = "Ex.:>

Class <
    name = "Blog"
    table = "blogs"
    lazy = "true">

    <Id
        name = "Id"
        coluna = "blog id">

        <generator class="native"/>

    </ Id>

    Propriedade <
        name = "Nome"
        coluna = "NAME"
        not-null = "true"
        unique = "true" />

    Saco de <
        name = "Items"
        inverse = "true"
        lazy = "true"
        fim-by = "DATE_TIME"
        cascade = "all">

        <key column="BLOG_ID"/>
        <one-to-many class="BlogItem"/>

    <Saco />

</ Class>

</ Hibernate-mapping>

```

```

<? Xml version = "1.0"?>
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
assembly = "Eg" namespace = "Ex.:>

Class <
    name = "BlogItem"

```

```





```

17.3. NHibernate Código

A classe a seguir demonstra alguns dos tipos de coisas que podemos fazer com essas classes, usando NHibernate.

```

using System;
using System.Collections;

usando NHibernate.Tool.hbm2ddl;

namespace Eg
{
    BlogMain public class
    {
        privada sessions ISessionFactory;

        Configure public void ()
        {
            _sessions Configuração = new ()
                . AddClass (typeof (Blog))
                . AddClass (typeof (BlogItem))
                . BuildSessionFactory ();
        }

        pública ExportTables void ()
        {
            Cfg configuração Configuration = new ()
                . AddClass (typeof (Blog))
                . AddClass (typeof (BlogItem));
            . nova SchemaExport (cfg) criar (true, true);
        }

        Blog pública CreateBlog (string nome)
    }
}

```

```

{
    Blog blog = Blog new ();
    blog.Name name =;
    blog.Items = new ArrayList ();

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        Session.save (blog);
        tx.Commit ();
    }

    retorno blog;
}

pública BlogItem CreateBlogItem (blog Blog, título string, string texto)
{
    BlogItem item = nova BlogItem ();
    item.Title title =;
    item.Text = texto;
    item.Blog = blog;
    item.DateTime = DateTime.Now;
    blog.Items.Add (item);

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        Session.update (blog);
        tx.Commit ();
    }

    retorno item;
}

pública BlogItem CreateBlogItem (blogID longa, título string, string texto)
{
    BlogItem item = nova BlogItem ();
    item.Title title =;
    item.Text = texto;
    item.DateTime = DateTime.Now;

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        Blog blog = (Blog) Session.load (typeof (Blog), blogID);
        item.Blog = blog;
        blog.Items.Add (item);
        tx.Commit ();
    }

    retorno item;
}

public void UpdateBlogItem (item BlogItem, string texto)
{
    item.Text = texto;

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        Session.update (item);
        tx.Commit ();
    }
}

UpdateBlogItem public void (itemId longa, string texto)
{
    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        BlogItem item = (BlogItem) Session.load (typeof (BlogItem), itemId);
}

```

```

        item.Text = texto;
        tx.Commit ();
    }

}

pública listAllBlogNamesAndItemCounts IList (int max)
{
    IList resultado = null;

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        IQuery session.CreateQuery q = (
            "Select blog.id, blog.Name, count (blogItem)" +
            "Blog de como blog" +
            "Left outer join blog.Items como blogItem" +
            "Grupo de blog.Name, blog.id" +
            "Order by max (blogItem.DateTime)"
        );
        q.setMaxResults (max);
        resultado q.List = ();
        tx.Commit ();
    }
}

resultado de retorno;
}

pública GetBlogAndAllItems Blog (longa blogID)
{
    Blog blog = null;

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        IQuery session.createQuery q = (
            "Blog de como blog" +
            "Left outer join fetch blog.Items" +
            "Onde blog.id =: blogID"
        );
        q.setParameter ("blogID", blogID);
        blog = (Blog) q.List () [0];
        tx.Commit ();
    }
}

retorno blog;
}

pública ListBlogsAndRecentItems IList ()
{
    IList resultado = null;

    usando (sessão ISession _sessions.OpenSession = ())
    usando (ITransaction tx session.BeginTransaction = ())
    {
        IQuery session.CreateQuery q = (
            "Blog de como blog" +
            "Inner join blog.Items como blogItem" +
            "Onde blogItem.DateTime >: minDate"
        );
        DateTime data = DateTime.Now.AddMonths (-1);
        q.setDateTime ("minDate", data);

        resultado q.List = ();
        tx.Commit ();
    }
}

resultado de retorno;
}
}

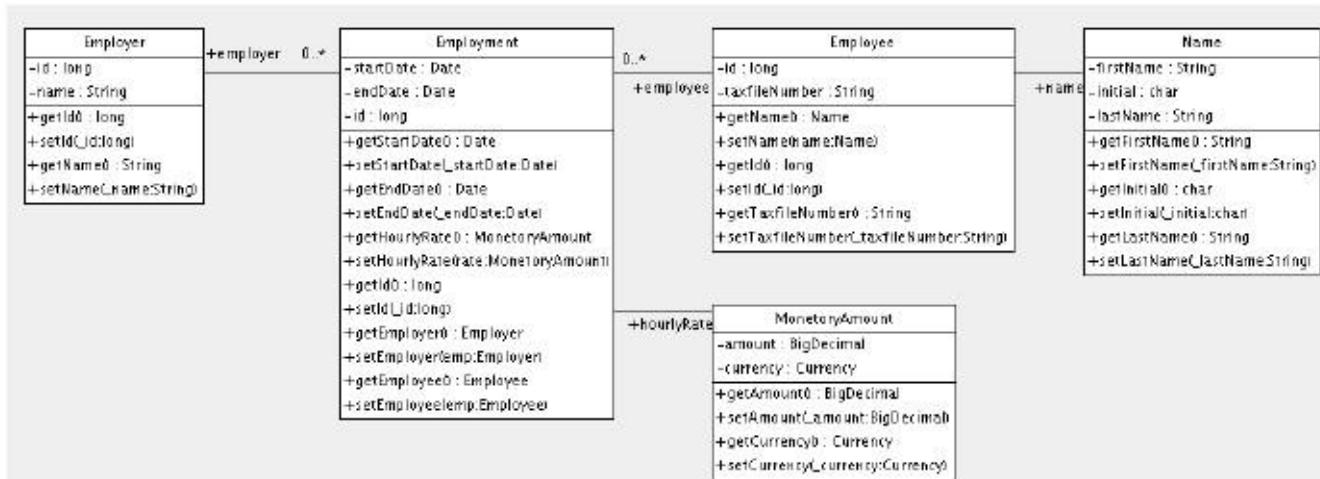
```

Capítulo 18. Exemplo: Vários Mapeamentos

Este capítulo mostra alguns mapeamentos de associação mais complexas.

18.1. Empregador / empregado

O seguinte modelo da relação entre `Empregador` e `Empregado` usa uma classe de entidade real (`Empregamento`) Para representar a associação. Isso é feito porque pode haver mais de um período de emprego para os mesmos dois partidos. Componentes são usados para modelar valores monetários e nomes de funcionários.



Aqui está um documento de mapeamento possível:

```
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
    montagem =..." namespace =...">

    <class name="Employer" table="employers">
        <id name="Id">
            <generator class="sequence">
                <param name="sequence"> employer_id_seq </ param>
                <Gerador />
            </ Id>
            <property name="name"/>
        </ Class>

        <class name="Employment" table="employment_periods">

            <id name="Id">
                <generator class="sequence">
                    <param name="sequence"> employment_id_seq </ param>
                    <Gerador />
                </ Id>
                <property name="StartDate" column="start_date"/>
                <property name="EndDate" column="end_date"/>

                componente> name="HourlyRate" class="MonetaryAmount">
                    <property name="Amount">
                        <column name="hourly_rate" sql-type="NUMERIC(12, 2)"/>
                    </ Property>
                    <property name="Currency" length="12"/>
                </ Component>

                <many-to-one name="Employer" column="employer_id" not-null="true"/>
                <many-to-one name="Employee" column="employee_id" not-null="true"/>

            </ Class>

            <class name="Employee" table="employees">
```

```

<id name="Id">
    <generator class="sequence">
        <param name="sequence"> employee_id_seq </ param>
        <Gerador />
    </ Id>
    <property name="TaxfileNumber"/>
    <component name="nome" class="name">
        <property name="FirstName"/>
        <property name="Initial"/>
        <property name="LastName"/>
    </ Component>
</ Class>

</ Hibernate-mapping>

```

E aqui está o esquema da tabela gerada pelo SchemaExport.

```

empregadores criar tabela (
    BIGINT id não nulo,
    Nome VARCHAR (255),
    chave primária (Id)
)

employment_periods criar tabela (
    BIGINT id não nulo,
    hourly_rate NUMERIC (12, 2),
    Moeda VARCHAR (12),
    BIGINT employee_id não nulo,
    BIGINT employer_id não nulo,
    TIMESTAMP end_date,
    TIMESTAMP start_date,
    chave primária (Id)
)

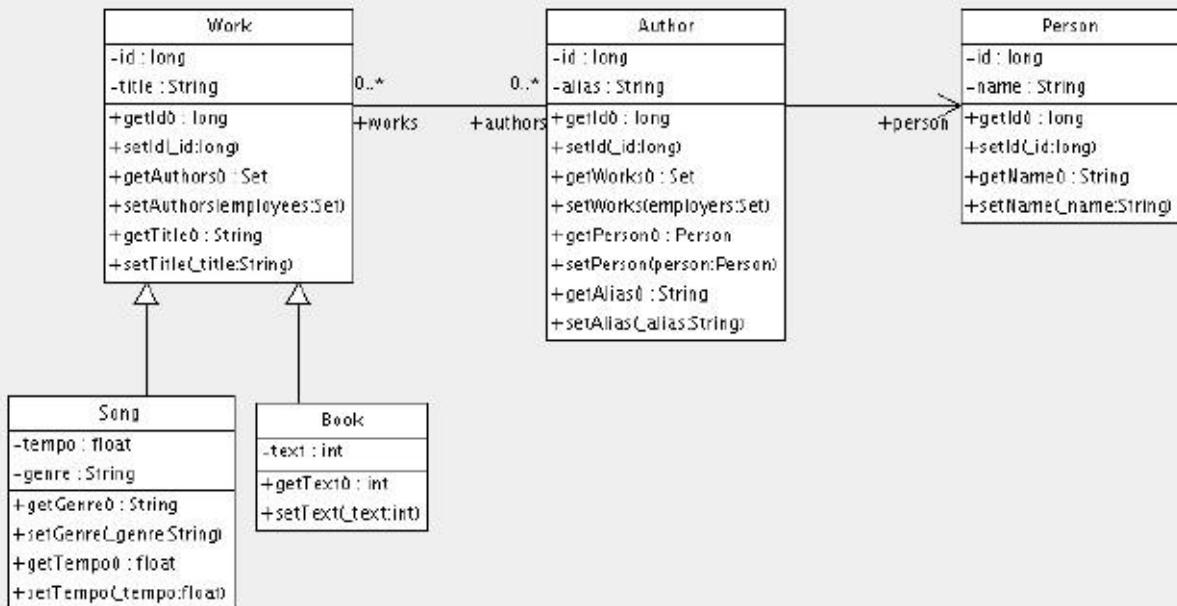
criar empregados de mesa (
    BIGINT id não nulo,
    Nome VARCHAR (255),
    CHAR inicial (1),
    LastName VARCHAR (255),
    TaxfileNumber VARCHAR (255),
    chave primária (Id)
)

employment_periods alter table
    adicionar restrição de chave estrangeira employment_periodsFK0 (employer_id) empregadores
    referências
employment_periods alter table
    acrescentar uma restrição de chave estrangeira employment_periodsFK1 (employee_id) empregados
    referências
criar employee_id_seq seqüência
criar employment_id_seq seqüência
criar employer_id_seq seqüência

```

18.2. Autor / Trabalho

Considere o seguinte modelo das relações entre Trabalho, Autor e Pessoa. Nós representamos a relação-navio entre Trabalho e Autor como uma associação muitos-para-muitos. Nós escolhemos para representar a relação entre Autor e Pessoa como um-para-um de associação. Outra possibilidade seria a de ter Autor ampliar Pessoa.



O documento seguinte mapeamento representa corretamente estes relacionamentos:

```

<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
  montagem ="..." namespace ="...">

  <class name="Work" table="works" discriminator-value="W">

    <id name="Id" column="id">
      <generator class="native"/>
    </ Id>
    <discriminator column="type" type="character"/>

    <property name="Title"/>
    <set name="Authors" table="author_work" lazy="true">
      <key>
        <column name="work_id" not-null="true"/>
      </ Key>
      <many-to-many class="Author">
        <column name="author_id" not-null="true"/>
      </ Many-to-many>
    </ Set>

    <subklass name="Book" discriminator-value="B">
      <property name="Text" column="text" />
    <Subclasse />

    <subklass name="Song" discriminator-value="S">
      <property name="Tempo" column="tempo" />
      <property name="Genre" column="genre" />
    <Subclasse />

  </ Class>

  <class NAME="author" table="authors">

    <id name="Id" column="id">
      <! - O autor deve ter o mesmo identificador como a Pessoa ->
      <generator class="assigned"/>
    </ Id>

    <property name="Alias" column="alias" />
    <one-to-one name="Person" constrained="true"/>

    <set name="Works" table="author_work" inverse="true" lazy="true">
      <key column="author_id"/>
    </ Set>

  </ class>

```

```

        <many-to-many class="Work" column="work_id"/>
    </ Set>

    </ Class>

    <class name="Person" table="persons">
        <id name="Id" column="id">
            <generator class="native"/>
        </ Id>
        <property name="nome" column="name" />
    </ Class>

</ Hibernate-mapping>

```

Há quatro tabelas neste mapeamento. `obras`, `autores` e `pessoas` segurar autor do trabalho, e pessoa de dados-respeito vamente. `author_work` é uma tabela de associação que liga autores de obras. Heres o esquema da tabela, como gerada por `SchemaExport`.

```

criar obras de mesa (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    FLOTAT tempo,
    gênero VARCHAR (255),
    INTEGER texto,
    título VARCHAR (255),
    tipo CHAR (1) não nulo,
    chave primária (id)
)

author work criar tabela (
    BIGINT author_id não nulo,
    BIGINT work_id não nulo,
    chave primária (work_id, author_id)
)

criar autores da tabela (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    apelido VARCHAR (255),
    chave primária (id)
)

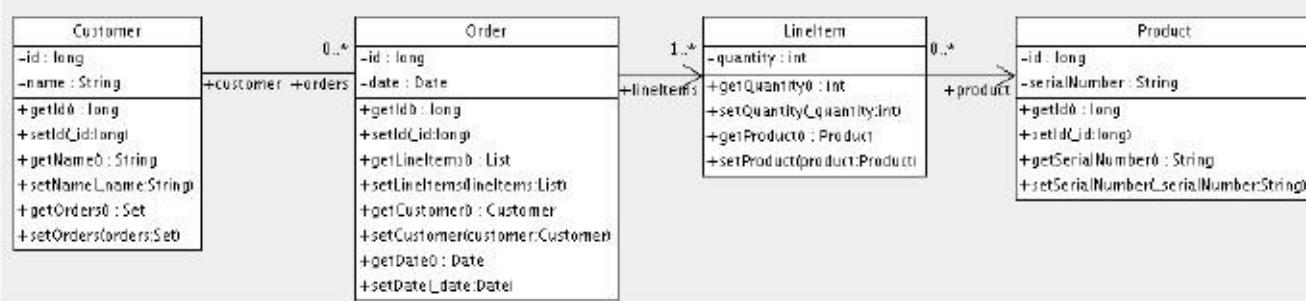
criar pessoas Mesa (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    nome VARCHAR (255),
    chave primária (id)
)

alter autores tabela
    adicionar restrição de chave estrangeira authorsFK0 (id) pessoas referências
author work alter table
    adicionar restrição de chave estrangeira author_workFK0 (author_id) autores referências
author_work alter table
    acrescentar uma restrição de chave estrangeira author_workFK1 (work_id) referências
funciona

```

18.3. Ordem do cliente // Produto

Agora considere um modelo das relações entre `Cliente`, `Ordem` e `LineItem` e `Produto`. Há um um-para-muitos associação entre `Cliente` e `Ordem`, Mas como devemos representar `Ordem / LineItem / Produto`? Eu escolhi para mapear `LineItem` como uma classe de associação que representa a associação many-to-many entre `Ordem` e `Produto`. Em NHibernate, isso é chamado de um elemento composto.



O documento de mapeamento:

```
<Hibernate-mapping xmlns = "urn: nhibernate-mapping-2.0"
    montagem =..." namespace =...">

    <class name="Customer" table="customers">
        <id name="Id" column="id">
            <generator class="native"/>
        </ Id>
        <property name="nome" column="name"/>
        <set name="Orders" inverse="true" lazy="true">
            <key column="customer_id"/>
            <one-to-many class="Order"/>
        </ Set>
    </ Class>

    <class name="Order" table="orders">
        <id name="Id" column="id">
            <generator class="native"/>
        </ Id>
        <property name="Date" column="date"/>
        <many-to-one name="Customer" column="customer_id"/>
        <enumerar name="LineItems" table="line_items" lazy="true">
            <key column="order_id"/>
            <index column="line number"/>
            class="LineItem" > <composite-element
                <property name="Quantity" column="quantity"/>
                <many-to-one name="Product" column="product_id"/>
            </ Elemento composto->
        </ List>
    </ Class>

    <class name="Product" table="products">
        <id name="Id" column="id">
            <generator class="native"/>
        </ Id>
        <property name="SerialNumber" column="serial_number" />
    </ Class>
</ Hibernate-mapping>
```

`clientes`, `ordens`, `line_items` manter clientes, a ordem, item de linha da ordem e os dados do produto respeitivamente. `line_items` também atua como uma tabela de associação associando as encomendas com os produtos.

```
criar clientes tabela (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    nome VARCHAR (255),
    chave primária (id)
)

criação de ordens de mesa (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    BIGINT customer_id,
    TIMESTAMP data,
    chave primária (id)
)

line_items criar tabela (
```

```
line_number INTEGER não nulo,
BIGINT order_id não nulo,
BIGINT product_id,
INTEGER quantidade,
chave primária (order_id, line_number)

)

criar produtos de mesa (
    id BIGINT NOT NULL gerado por padrão como a identidade,
    num_série VARCHAR (255),
    chave primária (id)
)

alterar ordens tabela
    adicionar restrição de chave estrangeira ordersFK0 (customer_id) clientes referências
line_items alter table
    adicionar restrição de chave estrangeira line_itemsFK0 (product_id) Produtos referências
line_items alter table
    acrescentar uma restrição de chave estrangeira line_itemsFK1 (order_id) ordens de
referências
```

Capítulo 19. Melhores Práticas

Escreva de grão fino classes e mapeá-los usando `<component>`.

Use uma `Endereço` classe para encapsular `rua,subúrbio,estado,CEP`. Isto promove a reutilização de código e simplifica o refactoring.

Declare propriedades identificadoras em classes persistentes.

NHibernate faz propriedades identificador opcional. Há todos os tipos de razões pelas quais você deve usá-los.

Nós recomendamos que os identificadores sejam 'sintéticos' (gerados, sem significado de negócios) e de um não-primitivo tipo. Para uma flexibilidade máxima, use `Int64` ou `Corda`.

Coloque cada classe de mapeamento em seu próprio arquivo.

Não use um documento de mapeamento único e monolítico. Mapa `Eg.Foo` no arquivo `Eg / Foo.hbm.xml`. Isso faz com que particularmente bom senso em um ambiente de equipe.

Incorporar mapeamentos em assembléias.

Colocar os arquivos de mapeamento, juntamente com as classes que mapa e declará-los como Recurso Incorporados em Visual Studio.

Considerar seqüências de consulta externalização.

Esta é uma boa prática se suas consultas chamar funções não-padrão ANSI SQL. Externalizar a consulta cordas para arquivos de mapeamento fará com que a aplicação mais portável.

Use parâmetros.

Como no ADO.NET, sempre substitua valores não constantes por "?". Nunca use manipulação de strings para ligar um não-valor constante em uma consulta! Melhor ainda, considerar o uso de parâmetros nomeados nas consultas.

Não gerencie suas conexões ADO.NET.

NHibernate permite que o aplicativo gerenciar conexões ADO.NET. Esta abordagem deve ser considerada a-última resort. Se você não pode usar os provedores de built-in conexões, considerar a possibilidade de sua própria implementação de `NHibernate.Connection.IConnectionProvider`.

Considere o uso de um tipo personalizado.

Suponha que você tenha um tipo de, digamos, de alguma biblioteca, que precisa ser persistentes, mas não fornece os assessores necessários para mapeá-la como um componente. Você deve considerar a implementação de `NHibernate.IUserType`. Este ap-gem libera o código da aplicação de implementar transformações de / para um tipo de NHibernate.

Uso da mão-coded ADO.NET em pontos de estrangulamento.

No desempenho crítico áreas do sistema, alguns tipos de operações (por exemplo massa update / delete) pode beneficiarem direto de ADO.NET. Mas, por favor, espere até saber se é um gargalo. E não pense que direta ADO.NET é necessariamente mais rápido. Se precisar usar direto ADO.NET, pode valer a pena abrir uma NHibernate `ISession` e usando essa conexão SQL. Dessa forma, você ainda pode usar a mesma transação estratégia e provedor de conexão subjacente.

Compreender `ISession` rubor.

De tempos em tempos o `ISession` sincroniza seu estado persistente com o banco de dados. Desempenho será afetadas se esse processo ocorre com muita freqüência. Você pode, por vezes minimizar o fluxo desnecessário desabilitando lavagem automática ou até mesmo mudando a ordem das consultas e outras operações dentro de um determinado transação.

Em uma arquitetura de três camadas, considerar o uso de `SaveOrUpdate ()`.

Ao usar uma arquitetura distribuída, você poderia passar objetos persistentes carregado na camada intermediária para e da camada de interface do usuário. Use uma nova sessão para atender a cada solicitação. Uso `ISession.Update ()` ou `ISES-`

`sion.SaveOrUpdate ()` para atualizar o estado persistente de um objeto.

Em uma arquitetura de duas camadas, considerar o uso de desconexão sessão.

Transações de banco de dados tem que ser o mais curto possível para melhor escalabilidade. No entanto, é frequentemente necessário

implementar operações de longo Aplicação em execução, uma unidade-de-obra única do ponto de vista de um usuário. Esta operação pode se estender aplicações diversas solicitações do cliente e ciclos resposta. Usar Moradia Objetos ou, em duas arquiteturas em camadas, basta desligar a sessão NHibernate do ADO.NET conexão Bluetooth e reconectá-lo para cada solicitação subsequente. Nunca use uma única sessão por mais de um Application Transaction usecase, caso contrário, você vai correr em dados desatualizados.

Não tratar exceções como recuperável.

Esta é mais uma prática necessária do que uma prática "melhor". Quando ocorre uma exceção, reverte o `ITransaction` e fechar o `ISession`. Se você não fizer isso, NHibernate não pode garantir que em memória de estado automaticamente representa o estado persistente. Como um caso especial deste, não utilize `ISession.Load ()` para determinar se um exemplo, com o identificador dado existe no banco de dados, uso `Get ()` ou uma consulta em seu lugar.

Preferem a busca preguiçosa para as associações.

Use ansiosos (outer-join) buscar com moderação. Use proxies e / ou coleções preguiçoso para a maioria das associações classes que não são armazenados em cache no cache de segundo nível. Para as associações de classes de cache, onde há uma alta probabilidade de acerto de cache, desabilite a busca ansiosa usando `fetch = "select"`. Quando uma junção externa-fetch é apropriado para um caso de uso particular, use uma consulta com um `left join fetch`.

Considere abstrair a lógica de negócios de NHibernate.

Hide (NHibernate) código de acesso a dados por trás de uma interface. Combine o DAO e Sessão Thread Local patterninhos. Você pode até ter algumas aulas persistiu por handcoded ADO.NET, associado ao NHibernate através de um `IUserType`. (Este conselho destina-se "suficientemente grandes" aplicações, não é apropriado para uma aplicação com cinco mesas!)

Executar `Equals ()` e `GetHashCode ()` usando uma chave de negócio único.

Se você comparar objetos fora do escopo `ISession`, você tem que implementar `Equals ()` e `GetHashCode ()`. Dentro do escopo `ISession`, a identidade do objeto é garantida. Se você implementar esses métodos, Nunca, jamais use o identificador de banco de dados! Um objeto transiente não tem um valor de identificador e NHibernate seria atribuir um valor quando o objeto é salvo. Se o objeto estiver em um ISET ao ser salvo, o código de hash mudanças, quebrando o contrato. Para implementar `Equals ()` e `GetHashCode ()`, Use uma chave de negócio único, isto é, comparar uma combinação única de propriedades de classe. Lembre-se que esta chave tem que ser estável e única só não enquanto o objeto estiver em um ISET, não para a vida inteira (tão estável como um banco de dados primário chave). Nunca use coleções no `Equals ()` comparação (carregamento lento) e ter cuidado com outros associados classes que pode ser proxy.

Não utilize mapeamentos de associação exóticos.

Casos de uso bom para um verdadeiro many-to-many associações são raros. Na maioria das vezes você precisa adicionais informar-

ation armazenadas na "tabela de link". Neste caso, é muito melhor usar duas associações um-para-muitos para um entre-médios classe link. Na verdade, pensamos que a maioria das associações são um-para-muitos e muitos-para-um, você deve ter cuidado ao utilizar qualquer estilo de outra associação e perguntar-se se é realmente necessário.

Parte I. NHibernateContrib Documentação

Prefácio

O NHibernateContrib é vários programas contribuíram para NHibernate por membros da equipe ou NHibernate pelos utilizadores finais. Os projetos aqui não são considerados peças fundamentais do NHibernate mas estendê-lo em um uso forma ful.

Capítulo 20. NHibernate.Caches

O que é NHibernate.Caches?

NHibernate.Caches são suplementos para NHibernate [<http://www.nhibernate.org>] contribuiu por Kevin Williams (aka k-dub). A cache é um local onde as entidades são mantidas (em sua primeira carga); vez em cache, eles podem ser retado em sem ter que consultá-los (de novo) no armazenamento de back-end. Isso significa que eles são mais rápidos de (Re) carga.

Uma sessão NHibernate tem um cache interno (primeiro nível), onde ele mantém suas entidades. Não há partilha entre estes esconderijos, de modo que uma sessão é destruída com o seu cache. NHibernate fornece uma **segundo nível de cache** sistema, que

trabalha no nível SessionFactory. Por isso, é compartilhada por todas as sessões criadas pelo SessionFactory mesmo.

Um ponto importante é que o cache de segundo nível **não** instâncias de cache do tipo de objeto que está sendo armazenada em cache, em-

lugar ele armazena em cache os valores individuais das propriedades desse objeto. Isto proporciona dois benefícios. Um, NHibernate não precisa se preocupar que seu código cliente irá manipular os objetos de uma maneira que irá perturbar o cache. Dois, as relações e associações não se tornar obsoleto, e são fáceis de manter-se atualizado, porque eles são simplesmente identificadores. O cache não é uma árvore de objetos, mas sim um mapa de matrizes.

Com o **por sessão-request** modelo, um elevado número de sessão podem acessar concorrentemente com a mesma entidade sem bater o banco de dados de cada vez, daí a ganhar a performance.

Estas contribuições tornam possível usar provedores de cache diferente para NHibernate:

- NHibernate.Caches.Prevalence torna possível a utilização da base `Bamboo.Prevalence` implementação como provedor de cache. Abra o arquivo `Bamboo.Prevalence.license.txt` para mais informações sobre a sua licenciar, você também pode visitar o seu site [<http://bbooprevalence.sourceforge.net/>].
- NHibernate.Caches.SysCache torna possível a utilização da base `System.Web.Caching.Cache` implementação como provedor de cache. Isto significa que você pode confiar em ASP.NET recurso de cache para entender como ele funciona. Para mais informações, leia (no MSDN): Cache de dados de aplicativos [[Http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcacheapis.asp](http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcacheapis.asp)].

20.1. Como usar um cache?

Aqui estão os passos a seguir para permitir que o cache de segundo nível em NHibernate:

- Escolher o provedor de cache você deseja usar e copiar a sua montagem em seu diretório de assemblies (`NHibernate.Caches.Prevalence.dll` ou `NHibernate.Caches.SysCache.dll`).
- Para dizer NHibernate qual provedor de cache para usar, adicione no seu arquivo de configuração do NHibernate (pode ser `YourAssembly.exe.config` ou `web.config` ou um `.Cfg.xml` arquivo):

```
<add key="hibernate.cache.provider_class" value="XXX" />(1)
<add key="expiration" value="120" />(2)
```

(1) "XXX" Pode ser `NHibernate.Caches.Prevalence.PrevalenceCacheProvider`,
ou `NHibernate.Caches.Prevalence` Ou `NHibernate.Caches.SysCache.SysCacheProvider`,
`NHibernate.Caches.SysCache`.

NHibern-
ate.
NHibern-

- (2) O `expiração` valor é o número de segundos que você deseja cache de cada entrada (aqui dois minutos). Este exemplo se aplica a SysCache só.
- Adicionar `<cache usage="read-write|nonstrict-read-write|read-only"/>` (Logo após `<class>`) no mapeamento de as entidades que você deseja armazenar em cache. Ele também funciona para as coleções (saco, lista, mapa, jogo, ...).

Tenha cuidado. Caches não estão cientes das mudanças feitas no armazenamento persistente, por outro processo (embora pode ser configurado para expirar regularmente dados em cache). Como os caches são criados no nível SessionFactory, eles são destruídos com a instância SessionFactory; assim que você deve mantê-los vivos, desde que você precisar deles.

20.2. Configuração da Cache prevalência

Há apenas um parâmetro configurável: `prevalenceBase`. Este é o diretório no sistema de arquivos onde o Motor de prevalência irá salvar os dados. Ele pode ser relativo ao diretório atual ou um caminho completo. Se o diretório não existir, ele será criado.

20.3. SysCache Configuração

Como SysCache depende `System.Web.Caching.Cache` para a implementação subjacente, a configuração é com base nas opções disponíveis que fazem sentido para NHibernate para utilizar.

- `timeToLive` = número de expiração de segundos de espera antes de expirar cada item
- `priority` = um custo numérico de expirar cada item, onde 1 é um custo baixo, 5 é o mais alto, e 3 é normal. Apenas valores de 1 a 5 são válidos.

SysCache tem um manipulador da seção de arquivo de configuração para permitir a configuração de vencimentos diferentes e as prioridades para diferentes regiões. Aqui está um exemplo:

```
<? Xml version = "1.0" encoding = "utf-8"?>
<configuration>
    <configSections>
        <Nome da seção = "syscache" type = "NHibernate.Caches.SysCache.SysCacheSectionHandler", NHibernate.Caches.SysCacheSectionHandler>
    </ configSections>

    <syscache>
        <cache region="foo" expiration="500" priority="4" />
        <cache region="bar" priority="3" expiration="300" />
    </ Syscache>
</ Configuration>
```

Capítulo 21. NHibernate.Mapping.Attributes

O que é NHibernate.Mapping.Attributes?

NHibernate.Mapping.Attributes é um add-in para o NHibernate [<http://www.nhibernate.org>] contribuiu por Pierre Henri Kuate (aka KPixel); a implementação foi feita pelo ex-John Morris. NHibernate requer fluxos de mapeamento para vincular o seu modelo de domínio para seu banco de dados. Normalmente, eles são escritos (e mantidos) em arquivos separados hbm.xml.

Com NHibernate.Mapping.Attributes, você pode usar. Atributos .NET para decorar suas entidades e seus atributos será usado para gerar esses hbm.xml mapeamento. (como arquivos ou fluxos). Então, você não terá mais que se preocupar com este desagradável xml;).

Conteúdo dessa biblioteca.

- NHibernate.Mapping.Attributes: Que o projeto que você precisa (como usuário final)
- Teste: um exemplo de funcionamento usando atributos e HbmSerializer como TestFixture NUnit
- Gerador: O programa usado para gerar os atributos e HbmWriter
- Refly [[Http://mbunit.tigris.org/](http://mbunit.tigris.org/)]: Graças a Jonathan de Halleux [<http://www.dotnetwiki.org/>] para esta biblioteca o que o torna tão fácil de gerar o código

Importante

Este biblioteca é ~~gerado~~ utilização <src/NHibernate.Mapping.Attributes/nhibernate-mapping-2.0.xsd> (Que é embutido no assembly a ser capaz de validar os fluxos gerados XML). Como este arquivo pode mudar a cada nova versão do NHibernate, você deve regenerá-lo antes de usá-lo com uma versão diferente (abra o gerador de solução, compilar e executar o projeto Generator). Mas, nenhum teste foi feito com versões anteriores à 0.8.

21.1. Como usá-lo?

O usuário final da classe é NHibernate.Mapping.Attributes.HbmSerializer. Esta classe **seriar** seu domínio modelo para mapeamento de fluxos. Você pode serializar classes, uma a uma ou um conjunto. Olhar para `NHibernate.Mapping.Attributes.Test` projeto de uma amostra de trabalho.

O primeiro passo é decorar suas entidades com os atributos, você pode usar: `[Classe]`, `[Subclasse]`, `[JoinedSubclass]` ou `[Componente]`. Então, você decorar seus membros (campos / propriedades), pois eles podem levar quantos atributos required pelo seu mapeamento. Por exemplo:

```
[NHibernate.Mapping.Attributes.Class]
Exemplo public class
{
    [NHibernate.Mapping.Attributes.Property]
    Nome cadeia pública;
}
```

Após esta etapa, você usa `NHibernate.Mapping.Attributes.HbmSerializer`: (Aqui, usamos Padrão que é um exemplo, você pode usar se você não precisa / quer criá-lo).

```

System.IO.MemoryStream fluxo = new System.IO.MemoryStream () / / onde o xml será escrito
NHibernate.Mapping.Attributes.HbmSerializer.Default.Validate = true / / Ativar validação (OptionA
/ / Aqui, nós serializar todas as classes decorados (mas você também pode fazer isso de classe por classe)
NHibernate.Mapping.Attributes.HbmSerializer.Default.Serialize (
    stream, System.Reflection.Assembly.GetExecutingAssembly ());
stream.Position = 0; / / Rewind
Cfg NHibernate.Cfg.Configuration NHibernate.Cfg.Configuration = new ();
cfg.Configure ();
cfg.AddInputStream (stream) / / Use o fluxo aqui
stream.Close ();
/ / Agora você pode usar essa configuração para construir sua SessionFactory ...

```

Nota

Como você pode ver aqui: NHibernate.Mapping.Attributes é **não** (Realmente) intrusiva. Definição de atributos em seu objetos não força você a usá-los com NHibernate e não quebra qualquer restrição em seu arquivo. Atributos são puramente informativo!

21.2. Dicas

- Uso `HbmSerializer.Validate` para ativar / desativar a validação dos fluxos gerados xml (contra-NHibernate comeu esquema de mapeamento), o que é útil para encontrar rapidamente os erros (que estão escritos no `StringBuilder HbmSerializer.Error`). Se o erro é devido a esta biblioteca, em seguida, ver se é uma questão de conhecer e relatá-lo, você pode contribuir uma solução se você resolver o problema:)
- Suas classes, campos e propriedades (membros) pode ser privado; apenas tenha certeza que você tem a permissão para acessar membros privados usando a reflexão (`ReflectionPermissionFlag`).
- Membros de uma classes mapeadas são também buscam em suas classes base (até chegarmos mapeados classe base). Então você pode decorar alguns membros de uma classe base (não mapeados) e usá-lo em sua classe sub (mapeado) (es).
- Para um nome de tomar uma `System.Type`, Defina o tipo com `Name = "xxx"` (Como corda) Ou `NameType = typeof (xxx);` (Acrescentar "Tipo"A"Nome")
- . Por padrão, os atributos NET não manter a ordem dos atributos, de modo que você precisa para defini-la a si mesmo quando a ordem matéria (usando o primeiro parâmetro de cada atributo), é **altamente** recomendado para defini-la quando você tem mais de um atributo sobre o mesmo membro.
- Enquanto não há ambigüidade, você pode decorar um membro com muitos atributos não relacionados. A ex-boa ampla é colocar classe atributos relacionados (como `<discriminator>`) No membro identificador. Mas não se esqueça que as questões de ordem (o `<discriminator>` deve ser posterior à `<id>`). A ordem utilizada vem do fim de elementos no esquema de mapeamento do NHibernate. Pessoalmente, eu prefiro usar números negativos para estes atributos (se eles vêm antes!).
- Você pode adicionar `[HibernateMapping]` em suas classes para especificar `<hibernate-mapping>` atributos (usado quando serialização da classe em seu fluxo). Você também pode usar * `HbmSerializer.Hbm` propriedades (usado quando a serialização uma montagem ou um tipo que não é decorado com `[HibernateMapping]`).
- Em vez de usar uma string para `DiscriminatorValue` (Em [Classe] e [Subclasse]), Você pode usar qualquer objeto que você quer. Exemplo:

```
[Subclasse (DiscriminatorValueEnumFormat = "d", DiscriminatorValueObject DiscEnum.Vall =)]
```

Aqui, o objeto é um Enum, e você pode definir o formato que você quiser (o valor padrão é "g"). Note que você

deve colocá-lo **antes!** Para os tipos de outros, Ele simplesmente usar o `Tostring()` método do objeto.

- Se você estiver usando membros do tipo `Nullables.NullableXXX` (A partir do `Nullables` biblioteca), então eles vão ser mapeados para `Nullables.NHibernate.NullableXXXTyoe` automaticamente; não definir `Tipo = "..."` em `[Propriedade]` (Deixe-a nula). Graças a **Michael Terceiro** para a idéia:)
- Cada fluxo gerado por `NHibernate.Mapping.Attributes` pode conter um comentário com a data da geração; Você pode ativar / desativar esta usando o método `WriteDateComment`.
- Se você se esqueça de fornecer um atributo xml necessário, ele irá, obviamente, lançar uma exceção ao gerar o mapeamento.
- A maneira recomendada e mais fácil de mapa `[Componente]` é usar `[ComponentProperty]`. O primeiro passo é colocar `[Componente]` na classe de componente e mapear seus campos / propriedades. Observe que você não deve definir o `Nome` em `[Componente]`. Então, em cada membro em sua classes, adicionar `[ComponentProperty]`. Mas você não pode substituir `Acesso,Atualizar ou Inserir` para cada membro.

Há um exemplo de trabalho em `NHibernate.Mapping.Attributes.Test` (Olhar para a classe `CompAddress` e sua uso em aulas de outros).

Uma última coisa: `ComponentPropertyAttribute` herda `DynamicComponentAttribute` facilmente escrevê-lo logo após `<component>` elementos no fluxo de XML.

- Outra forma de mapa `[Componente]` é usar a maneira como essa biblioteca funciona: Se uma classe contém uma mapeadas mapeado componente, então este componente será incluir na classe. `NHibernate.Mapping.Attributes.Test` contém as classes `JoinedBaz` e `Coisas` que ambos utilizam o componente `Endereço`.

Basicamente, é feito através da adição

```
[Componente (Name = "MyComp")] classe privada SubComp: Comp {}
```

em cada classe. Uma das vantagens é que você pode substituir `Acesso,Atualizar ou Inserir` para cada membro. Mas você tem que adicionar o componente na subclasse **cada** classe (e não pode ser herdada).

- **Sobre a personalização.** `HbmSerializer` usa `HbmWriter` para serializar cada tipo de atributos. Seus métodos são virtuais, assim você pode criar uma subclasse e sobrescrever qualquer método que você deseja (para alterar seu comportamento padrão). Use a propriedade `HbmSerializer.HbmWriter` para mudar o escritor usado (você pode definir uma subclasse de `HBM-Escritor`).

Exemplo usando algumas dicas esta: (0, 1 e 2 são índices posição)

```
[NHibernate.Mapping.Attributes.Id (0, TypeType = typeof (int))] // Não colocá-lo depois de [ManyToOne]!
[NHibernate.Mapping.Attributes.Generator (1, Classe = "uuid.hex")]
[NHibernate.Mapping.Attributes.ManyToOne (2, ClassType = typeof (Foo), OuterJoin = OuterJoinStrategy.True)
Entidade Foo privado;
```

Gera:

```
<id type="Int32">
<generator class="uuid.hex" />
</ Id>
<many-to-one name="Entity" class="Namespaces.Foo, SampleAssembly" outer-join="true" />
```

21.3. Sabe questões e TODOs

Primeiro, leia TODOs no código fonte;)

A `Posição` propriedade foi adicionada a todos os atributos para ordená-las. Mas ainda há um problema:

Quando um elemento pai "p" tem um elemento filho "x" que é também o elemento filho de outro elemento filho "c" do "P" (anterior "x"): Ilustração D:

```
<p>
  <c>
    <x />
  </ c>
  <x />
</ p>
```

Neste caso, ao escrever:

```
[Attributes.P (0)]
[Attributes.C (1)]
[Attributes.X (2)]
[Attributes.X (3)]
pública MyType MyProperty;
```

X (3) será sempre pertencem a C (1)! (Como X (2)).

É o caso de `<dynamic-component>` e `<nested-composite-element>`.

Outra má notícia é que, atualmente, elementos XML que vem depois dessa elementos não podem ser incluídos neles. Por exemplo: Não há como colocar uma coleção em `<dynamic-component>`. A razão é que o arquivo `nhibernate-mapping-2.0.xsd` diz como os elementos são construídos e em que ordem, e NHibernate.Mapping.Attributes usar esse ou-der.

De qualquer forma, a solução seria adicionar um `int ParentNode` propriedade para BaseAttribute de modo que você pode criar um verdadeiro gráfico ...

Na verdade, não há nenhum problema, nem conhecer outras modificações planejadas. Esta biblioteca deve ser estável e completo, mas se você encontrar um bug ou pensar em uma melhoria útil, entre em contato conosco!

Em nota à parte, seria bom para escrever melhor do que TestFixture NHibernate.Mapping.Attributes.Test : D

21.4. Anotações Developer

Qualquer alteração ao esquema (`nhibernate-mapping-2.0.xsd`) Implica:

- Verificar se há alguma mudança a fazer no Generator (como atualização KnowEnums / AllowMultipleValue / IsRoot / IsSystemType / IsSystemEnum / CanContainItself)
- Atualização / `Src/NHibernate.Mapping.Attributes/nhibernate-mapping-2.0.xsd` (Copy / paste) e execut-ning o gerador de novo (mesmo que não foi modificado)
- Executando o projeto de teste e certifique-se que nenhuma exceção é lançada. A propriedade de classe / deve ser modified/ad-deed neste projeto para ter certeza de que qualquer alteração de quebra novo será travado (=> atualizar a referêcia arquivos `hbm.xml` e / ou o projeto `NHibernate.Mapping.Attributes 1.1.csproj`-)

Esta implementação é baseada em esquema de mapeamento do NHibernate, assim provavelmente há muitos "esquema padrão FEA-

ras "que não são suportadas ...

A versão do NHibernate.Mapping.Attributes deve ser a versão do esquema NHibernate utilizado para gerar o código (=> versão do NHibernate biblioteca).

Na concepção deste projecto, o desempenho é uma meta (muito) menor:) Mais fácil implementação e manutenção são muito mais importante.

Capítulo 22. NHibernate.Tool.hbm2net

O que é NHibernate.Tool.hbm2net?

NHibernate.Tool.hbm2net é um add-in para o NHibernate [<http://www.nhibernate.org>]. Torna-se possível gerar arquivos de origem dos arquivos de mapeamento hbm.xml.

No diretório `NHibernate.Tasks`, Há uma ferramenta chamada `Hbm2NetTask` que você pode usar para automatizar a sua processo de construção (usando NAnt)

Capítulo 23. Nullables

O que é Nullables?

Nullables é um add-in contribuiu para NHibernate [<http://www.nhibernate.org>] por Donald L Mull Jr. (Aka bagagem). A maioria dos sistemas de banco de dados permitem que tipos de base (como `int` ou `bool`) Para ser nulo. Isto significa que um boolean coluna pode assumir os valores 0, 1 ou null, onde nulo não tem o mesmo significado que 0. Mas não é possível NET com 1.x; `bool` um é sempre verdadeiro ou falso.

Nullables torna possível a utilização de tipos de base anulável em NHibernate. Note-se que .NET 2.0 tem esse recurso.

23.1. Como usá-lo?

Aqui está um exemplo simples que usa um `Nullables.NullableDateTime` para (opcionalmente) armazena a data de nascimento para uma Pessoa.

```
público classe Pessoa
{
    int _id;
    name string;
    Nullables.NullableDateTime _dateOfBirth;

    Pessoa pública ()
    {
    }

    Id public int
    {
        get {return this._id;}
    }

    Nome cadeia pública
    {
        get {this._name return;}
        set {this._name = valor;}
    }

    público Nullables.NullableDateTime DateOfBirth
    {
        get {this._dateOfBirth return;}
        set {this._dateOfBirth = valor;}
    }
}
```

Como você pode ver, `DateOfBirth` tem o tipo de `Nullables.NullableDateTime` (Em vez de `System.DateTime`).

Aqui é o mapeamento

```
<? Xml version = "1.0" encoding = "utf-8"?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
    <class name="Example.Person, Example" table="Person">
        <id name="Id" access="field.camelcase-underscore" unsaved-value="0">
            <generator class="native" />
        </ Id>
        <property name="nome" type="String" length="200" />
        <Nome da propriedade = "DateOfBirth" type = "Nullables.NHibernate.NullableDateTimeType, Nullables.NHiber
    </ Class>
</ Hibernate-mapping>
```

Importante

No mapeamento, o tipo de **DateOfBirth** deve ser `Nullables.NHibernate.NullableDateTimeType`. Note-se que `NHibernate.Mapping.Attributes` alças que automaticamente.

`Nullables.NHibernate.NullableXXXTypes` são tipos wrapper usado para traduzir os tipos de `Nullables`
Tipos de banco de dados.

Aqui é um pedaço de código usando este exemplo:

```
Por pessoa = new Pessoa ();  
  
textBox1.Text = per.DateOfBirth.Value.ToString () / / irá lançar uma exceção quando não há valor.  
  
textBox1.Text = per.DateOfBirth.ToString () / / vai funcionar. ele irá retornar uma string vazia se não houver  
  
textBox1.Text = (per.DateOfBirth.Value.ToShortDateString per.DateOfBirth.HasValue (): "Desconhecido") / /  
  
per.DateOfBirth = new System.DateTime (1979, 11, 8); / cast / implícito da System.DateTime "plain".  
per.DateOfBirth = new NullableDateTime (novo System.DateTime (1979, 11, 8)) / / o longo caminho.  
  
per.DateOfBirth = null; / / isso funciona.  
per.DateOfBirth NullableDateTime.Default = / / isso é mais correto.
```