02

Cacheando dados

Uma das formas de se otimizar um sistema é minimizar o acesso ao banco de dados. Podemos fazer isso de várias formas: melhorando nossa lógica, agrupando buscas em uma única query, entre outros métodos.

O NHibernate já possui mecanismos que nos ajudam a minimizar a quantidade de queries executadas na aplicação. Nesse capítulo, estudaremos como funcionam os caches do NHibernate.

O cache do ISession

Quando carregamos uma entidade com o método Get do ISession, o NHibernate busca os dados do banco de dados e devolve um objeto do tipo pedido no estado persistent, mas antes de entregar o objeto para o código da aplicação, ele guarda a referência para a entidade dentro do objeto ISession.

Por exemplo, vamos simular o carregamento da categoria com id 1.

```
ISession session = // abre um ISession
Categoria categoria = session.Get<Categoria>(1);
```

Na linha do `Get`, o NHibernate, inicialmente, procura a entidade pedida dentro do `ISession`, como estamos realizando o primeiro `Get` na categoria de id 1, o NHibernate ainda não possui a referência para essa categoria, portanto ele deve executar uma query para buscá-la no banco de dados. Quando o banco devolve o resultado da busca, o NHibernate primeiro guarda a referência para a categoria devolvida e depois entrega essa referência para a aplicação.

Agora vamos tentar carregar mais uma vez a categoria de id 1.

```
ISession session = // abre um ISession
Categoria categoria = session.Get<Categoria>(1);
Categoria categoria2 = session.Get<Categoria>(1);
```

No segundo `Get`, o NHibernate procura novamente a categoria dentro do `ISession`. Como essa categoria já está armazenada, não é necessário executar a query no banco de dados, ele simplesmente devolve a entidade guardada, ou seja, o `ISession` do NHibernate funciona como um cache de entidades para a aplicação

```
ISession session = // abre um ISession
Categoria categoria = session.Get<Categoria>(1);
Categoria categoria2 = session.Get<Categoria>(1);

if(categoria == categoria2)
{
    Console.WriteLine("categorias iguais")
}
```

Logo a execução do código acima mostra a mensagem `categorias iguais` no terminal.

O Cache compartilhado

O Cache da sessão ajuda no desempenho de uma aplicação, porém toda vez que a fechamos, seu cache é destruído. Em um aplicação web, temos, normalmente, um `ISession` por requisição, ou seja, a sessão do NHibernate tem um tempo de vida curto, o que faz com que o cache do `ISession` não seja muito efetivo. Precisamos de um cache que será compartilhado entre todas as sessões do NHibernate. Esse cache compartilhado entre as sessões do NHibernate fica dentro da classe `ISessionFactory`.

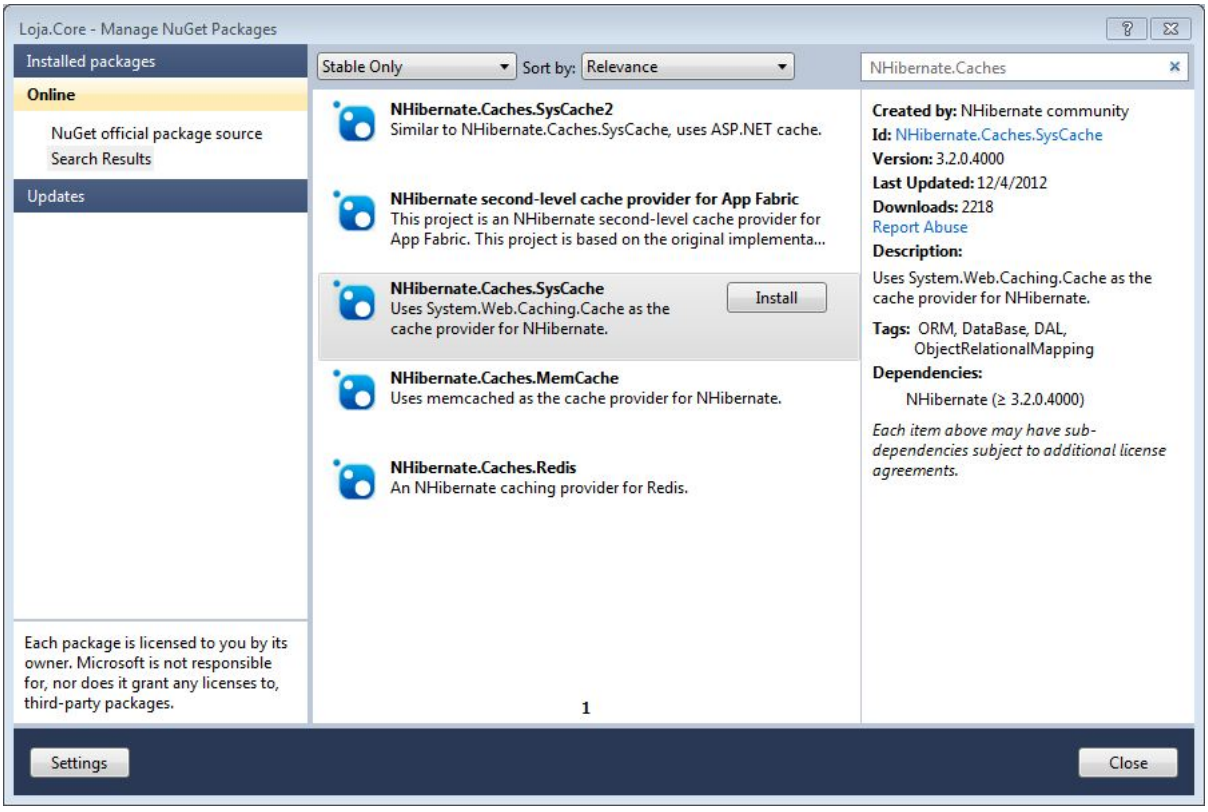
O `ISession` funciona como um primeiro nível de cache para a aplicação, e por isso é chamado de Cache de primeiro nível ou First Level Cache, um cache que está sempre habilitado na aplicação. Já o compartilhado, que fica no `ISessionFactory` é o Cache de segundo nível ou Second Level Cache, um cache muito mais complexo e por isso, deve ser habilitado no arquivo de configurações do NHibernate, o `hibernate.cfg.xml`.

Para habilitar o uso do second level cache, devemos adicionar uma nova propriedade chamada `cache.use_second_level_cache` com o valor `true` no `hibernate.cfg.xml`.

```
<property name="cache.use_second_level_cache">true</property>
```

Além disso, como já existem implementações muito boas de cache no mercado, o NHibernate não fornece sua própria implementação, ele utiliza as que já estão prontas. Uma dessas implementações é o SysCache, que utiliza o cache do Asp.Net.

Para utilizarmos o cache do Asp.Net, devemos baixar o pacote NHibernate.Caches.SysCache utilizando o Nuget.



Após a instalação, precisamos configurar o NHibernate para que ele utilize o SysCache como provedor de cache. Para realizar essa configuração, precisamos apenas adicionar a propriedade `cache.cache_provider` no `hibernate.cfg.xml`:

```
<property name="cache.provider_class">
    NHibernate.Caches.SysCache.SysCacheProvider, NHibernate.Caches.SysCache
</property>
```

Com isso, conseguimos habilitar o second level cache no NHibernate, mas como esse cache é global, precisamos configurar quais entidades podem ser armazenadas, uma configuração que é feita no arquivo de mapeamento da entidade.

Para configurarmos que a categoria pode ser cacheada, precisamos colocar a tag `cache` dentro do arquivo de mapeamento da entidade (Categoria.hbm.xml). Dentro dessa tag, devemos configurar como o NHibernate lidará com as situações de concorrência ao acessar a entidade. Temos 3 configurações possíveis:

- read-only: A entidade só pode ser lida, qualquer modificação gera uma exceção no código. Essa estratégia pode ser utilizada quando temos entidades que representam constantes no sistema
- nonstrict-read-write: A entidade pode ser lida e modificada, porém, em uma situação de alta concorrência, o NHibernate não garante que o valor armazenado no cache é a última versão da entidade. Essa estratégia pode ser utilizada em situações em que temos baixa concorrência ou quando a entidade não for modificada com muita frequência
- read-write: Essa é uma estratégia mais abrangente e permite que a entidade possa ser modificada com mais frequencia. Garante que os dados do cache são os mesmos do banco de dados.

Vamos configurar a categoria para utilizar a estratégia `nonstrict-read-write`

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Loja"
    namespace="Loja.Entidades">
  <class name="Categoria">
    <cache usage="nonstrict-read-write"/>

    <!-- resto do mapeamento do produto -->
  </class>
</hibernate-mapping>
```

Agora que configuramos a categoria, vamos testar o second level cache, para isso carregaremos a categoria utilizando duas sessões diferentes do NHibernate:

```
ISession session1 = // Abre uma session
ISession session2 = // Abre outra session

Categoria c1 = session1.Get<Categoria>(1);
Categoria c2 = session2.Get<Categoria>(1);
```

Nesse programa, o NHibernate envia apenas uma query para o banco de dados.

Cache de coleções

Mesmo quando armazenamos uma entidade no cache, ao acessarmos um relacionamento do tipo to many, o NHibernate ainda é obrigado a executar uma query no banco de dados. Podemos configurar o NHibernate para que ele também armazene as coleções de uma entidade.

A categoria possui um relacionamento do tipo one to many com os produtos, representado por sua lista de produtos. Para pedirmos para o NHibernate armazenar a lista de produtos dentro do second level cache, precisamos mais uma vez utilizar a tag `cache`, porém dessa vez, a colocaremos dentro da tag `bag` que representa o relacionamento one to many.

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Loja"
    namespace="Loja.Entidades">
```

```
    <class name="Categoria">
      <cache usage="nonstrict-read-write"/>
      <!-- mapeamento das outras propriedades -->
      <bag name="Produtos">
        <cache usage="nonstrict-read-write"/>
        <key column="CategoriaId"/>
        <one-to-many class="Produto"/>

      </bag>
    </class>
  </hibernate-mapping>
```

Agora toda vez que o NHibernate carregar a categoria, ele colocará tanto a categoria quanto sua lista de produtos dentro do cache, porém o NHibernate não guarda os produtos da lista, ele guarda apenas uma lista com o id de cada um dos produtos da categoria.

Quando o NHibernate carrega a lista de produtos, ele busca os ids que foram armazenados no cache. Para cada id, ele fará um get na session, portanto executaremos uma query para cada produto da lista!

Para resolvermos esse problema, precisamos fazer com que o produto também entre no cache do NHibernate:

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Loja"
    namespace="Loja.Entidades">
  <class name="Produto">
    <cache usage="nonstrict-read-write"/>
    <!-- resto do mapeamento -->
  </class>
</hibernate-mapping>
```

Cache de queries

Além de cachear entidades e seus relacionamentos, o NHibernate pode também cachear o resultado das queries mais utilizadas do sistema.

Para habilitarmos o cache de queries do NHibernate, devemos adicionar a propriedade `cache.use_query_cache` no `hibernate.cfg.xml`

```
<property name="cache.use_query_cache" >true</property>
```

Essa propriedade apenas habilita o cache, ela não faz com que o NHibernate faça o cache das queries automaticamente. Quando queremos que uma query seja cacheada, temos que configurá-la através do método `SetCacheable` do objeto `IQuery`.

Por exemplo, vamos fazer com que o NHibernate faça o cache da query que lista as categorias do banco de dados:

```
ISession session = // Abre uma session
IQuery query = session.CreateQuery("from Categoria");
query.SetCacheable(true);
IList<Categoria> categorias = query.List<Categoria>();
```

Esse é todo o código que precisamos para cachear uma query!

Também podemos colocar o resultado de criterias dentro do cache de queries, para isso utilizamos o método `SetCacheable` da interface `ICriteria` :

```
ISession session = // abre a session
ICriteria criteria = session.CreateCriteria<Produto>();
criteria.SetCacheable(true);
IList<Produto> produtos = criteria.List<Produto>();
```

Quando pedimos para o NHibernate cachear o resultado de uma query que retorna uma coleção de entidades, ele, assim como no cache de coleções, armazena apenas os ids das entidades devolvidas e, para evitarmos o problema de desempenho, precisamos adicionar a entidade no cache de segundo nível.