NHIBERNATE FRAMEWORK

LICA Ion, SARANCIUC Dorian

Universitatea Tehnică a Moldovei

Rezumat: *În acest articol se prezintă o descriere generală despre conceptul de persistență și importanța NHibernate ca o tehnică de implementare pentru aceasta. Se va discuta despre baze de date relaționale, cu o viziune mai aprofundată asupra SQL. Se va prezenta un mecanism de abordare a persistenței în .NET și va fi discutat NHibernate ca soluție utilă. La final se va vorbi despre maparea obiect/relațională, ca o trecere de la modelul relațional la cel obiectual.*

Cuvinte Cheie: *NHibernate, persistență, SQL, entități, ORM, Asimetrie, CRUD, .NET, interogări.*

1. Privire generală asupra Persistenței

Persistență este o preocupare fundamentală în dezvoltarea de aplicații. Aproape toate aplicațiile necesită date persistente. Utilizați persistența pentru a permite ca datele să fie stocate chiar și atunci când programele pe care le folosesc nu rulează.

Pentru a ilustra, să presupunem că doriți să creați o aplicație care permite utilizatorilor să stocheze numerele de telefon ale companiilor lor și datele de contact ale acestora și să le preluați aceste date oricând este necesar. Cu excepția cazului în care doriți ca utilizatorul să părăsească programul de funcționare tot timpul, vă veți da seama curând că aplicația are nevoie pentru a salva într-un fel contactele undeva. Te vei confruntat cu o decizie de persistență: ai nevoie de a te decide cu care mecanism de persistența dorești să îl utilizezi. Aveți posibilitatea de a stoca datele dvs. în multe locuri, cea mai simplă metodă fiind un fișier text. Cel mai adesea, puteți alege o bază de date relațională, deoarece astfel de baze de date sunt înțelese pe scară largă și oferă caracteristici extraordinare pentru stocarea și regăsirea datelor în mod fiabil.

Persistența obiectelor semnifică că unele obiecte pot avea ciclu de viață mai mare decât procesul aplicației care le-a creat. Acestea pot fi salvate și recreate mai apoi pentru utilizarea ulterioară.

2. Baza de date relaționale

Probabil că ai lucrat deja cu o bază de date relațională, cum ar fi Microsoft SQL Server, MySQL sau Oracle. Majoritatea dezvoltatorilor folosesc baze de date relaționale în fiecare zi; ele au o largă acceptare și sunt considerate soluție robustă și matură la provocările moderne de gestionare a datelor.

Un sistem de management al bazelor de date relaționale (RDBMS) nu este specific .NET, și o bază de date relațională nu este neapărat specifică unei singure aplicații. Puteți avea mai multe aplicații care accesează o singură bază de date, unele scrise în .net, unele scrise în Java sau Ruby, și așa mai departe. Tehnologia relaționala oferă o modalitate de partajare a datelor între multe aplicații diferite.

Tehnologia relaționala este un numitor comun al multor sisteme și platforme tehnologice care sunt fără legătură. Modelul de date relaționale este adesea comună la nivel de întreprindere pentru reprezentarea obiectelor de afaceri: o afacere are nevoie pentru a stoca informații despre diverse lucruri cum ar fi: clienți, conturi și produse (obiectele de afaceri), și bazele de date relaționale sunt, de obicei, locul central unde acestea sunt definite și stocate.

3. Viziune asupra SQL

Orice tip de dezvoltare a bazelor de date în .NET, necesită o înțelegere solidă a bazelor de date relaționale și SQL care sunt o condiție esențială atunci când utilizați NHibernate. Vei folosi SQL pentru a regla performanța din aplicația NHibernate. NHibernate automatizează multe sarcini codificare repetitive, dar cunoștințele asupra tehnologiei de persistență trebuie să se extindă dincolo de NHibernate dacă doriți să profitați de întreaga putere a bazelor de date moderne SQL. Amintiți-vă că obiectivul de bază este o gestionare solidă, eficientă a datelor persistente.

4. Persistența în aplicațiile obiect-orientate

Într-o aplicație orientată spre obiect, persistența permite unui obiect să dureze mai mult decât procesul sau aplicația care l-a creat. Starea obiectului poate fi stocată pe disc și un obiect, cu aceeași stare de recreat la un moment dat în viitor.

Bazele de date relaționale moderne oferă o reprezentare structurată a datelor persistente, care să permită sortarea, căutarea și gruparea de date. Sistemele de management al bazelor de date sunt responsabile

pentru gestionarea lucrurilor cum ar fi integritatea datelor şi concurenţa; acestea sunt responsabile pentru schimbul de date între mai mulţi utilizatori şi aplicaţii multiple. Un sistem de management a bazelor de date oferă, de asemenea, un nivel de securitate a datelor. Când discutăm persistenţa ne gândim toate aceste lucruri:

- Stocare, organizare şi regăsire a datelor structurate
- Simultaneitatea şi integritatea datelor
- Distribuirea datelor

În special, ne gândim la aceste probleme, în contextul unei cereri orientate spre obiecte care utilizează un model de domeniu. O aplicaţie cu un model de domeniu nu funcţionează în mod direct cu reprezentarea tabelară a entităţilor business (folosind Datasets); aplicaţia are propriul model, obiect-orientat al entităţilor business.

4.1 Persistenţa şi arhitectura pe nivele

O arhitectură pe nivele împarte un sistem în mai multe grupuri, în care fiecare grup conţine cod care abordează o anumită zonă de problemă. Aceste grupuri sunt numite nivele. Un beneficiu major al abordării pe nivele este că puteţi face de multe ori modificări la un singur strat, fără o perturbare semnificativă a celorlalte straturi, făcând astfel sisteme mai puţin fragile şi mai mentenabile.

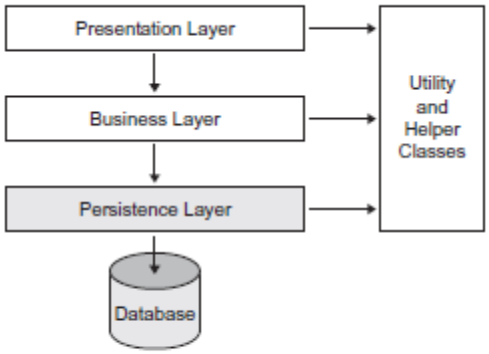

Figure 1.1 Layered architecture highlighting the persistence layer

Nivelul de Persistenţă – Stratul de persistenţă este un grup de clase şi componente responsabile pentru salvarea datelor aplicaţiilor şi preluarea din unul sau mai multe baze de date. Acest strat defineşte o mapare între entităţile domeniului de afaceri şi baza de date. Astfel NHibernate va fi folosit în primul rând în acest strat.

4.2 Abordări ale persistenţei în .NET

Am discutat despre modul în care, în orice aplicaţie considerabilă, aveţi nevoie de un strat de persistenţă să se ocupe de încărcarea şi de salvarea datelor. Multe abordări sunt disponibile atunci când are loc construirea acestui strat de persistenţă, şi fiecare are avantaje şi dezavantaje. Unele alegeri populare sunt după cum urmează:

- Codificare de mână (Hand Coding)
- Datasets
- LINQ-la-SQL
- NHibernate (sau similar)
- ADO.NET Entity Framework

În ciuda faptului că am recomanda NHibernate, este întotdeauna înţelept să se ia în considerare alternativele. Pe măsură ce veţi afla în curând, construirea de aplicaţii cu NHibernate este simplă, dar asta nu înseamnă că e perfect pentru fiecare proiect. Vom examina şi compara aceste strategii în detaliu, discutarea implicaţiilor pentru accesul bazelor de date şi interfaţa cu utilizatorul.

5. De ce avem nevoie de NHibernate

În lumea reală, rareori se fac aplicaţii simple. O aplicaţie pentru o întreprindere are multe entităţi cu obiective, logica de afaceri şi design complex: productivitatea, mentenanţa, şi performanţa sunt toate esenţiale.

În această secţiune, vom trece prin unele caracteristici indispensabile pentru implementarea unei aplicaţii de succes. În primul rând, vom da câteva exemple care ilustrează diferenţele fundamentale dintre obiecte şi baze de date relaţionale. Veţi afla, de asemenea, modul în care NHibernate ajută la crearea unei punţi între aceste reprezentări. Apoi, ne întoarcem la stratul de persistenţă pentru a descoperi modul în care se ocupă

NHibernate cu entități complexe și numeroase. Vei învață modelele și caracteristicile oferă pentru a obține cele mai bune performanțe. În cele din urmă, vom acoperi interogări complexe; veți vedea că puteți utiliza NHibernate pentru a scrie un puternic motor de căutare.

6. Unit of Work and Conversation

Atunci când utilizatorii lucrează la aplicații, acestea efectuează operații unitare distincte. Aceste operațiuni pot fi denumite conversații sau tranzacții comerciale. De exemplu, plasarea unei oferte pe un element este o tranzacție. Programatorii cu experiență știu cât de greu poate fi a vă asigura că multe operații conexe efectuate de către utilizator sunt tratate ca și cum ar fi o singură tranzacție de afaceri mai mare (o unitate).

7. The Unit of Work Pattern

Atunci când lucrați cu o bază de date relațională, s-ar putea să tinzi să te gândești la comenzi: de economisire sau de încărcare. Dar, o aplicație poate efectua operațiuni care implică mai multe entități. Când aceste entități sunt încărcate sau salvate depinde de context.

NHibernate folosește *Identity Map* pattern pentru a vă asigura că un element de utilizator este același obiect, pe care utilizatorul l-a avut înainte de a încărca elementul (atâta timp cât lucrați în aceeași tranzacție).

Acum, imaginați-vă că sunteți implicat într-o conversație complexă care implică multe actualizări și ștergeri. În cazul în care trebuie să urmăriți manual care entități trebuie salvate sau șterse, în timp ce vă asigurați că încărcați fiecare entitate doar o singură dată, lucrurile pot deveni rapid foarte dificil.

NHibernate urmează Unit of Work pattern pentru a rezolva această problemă și facilitează punerea în aplicare a convorbirilor. Aveți posibilitatea să creați entități și să le asociați cu NHibernate; apoi, NHibernate ține evidența tuturor, încărcând și salvând modificărilor numai atunci când este necesar. La finalul tranzacției, NHibernate aplică toate modificările în ordinea lor corectă.

8. Transparent Persistence and Lazy Loading

Pentru că NHibernate ține evidența tuturor entităților, se poate simplifica foarte mult aplicația dumneavoastră și crește performanța aplicației. Aici sunt două exemple simple.

Atunci când se lucrează pe un element în cererea de licitație, utilizatorii pot adăuga, modifica sau șterge ofertele lor. Ar fi dureros să urmăriți manual aceste schimbări, unul câte unul. În schimb, puteți utiliza caracteristica *transparent persistence* a lui NHibernate: rogi NHibernate să salveze toate modificările în colecția ofertelor atunci când elementul este persistat. El automat detectează care operațiune CRUD trebuie să fie executate.

Acum, dacă doriți să modificați un utilizator, încărcați, modificați și îl salvați. Dar, cum facem cu colecția de elemente ale acestui utilizator ? Ar trebui să încărcați aceste elemente sau să lăsați colecția ne-inițializată? Încărcarea elementelor ar fi ineficientă, dar lăsând colecția neinițializată va limita posibilitatea de a manipula utilizatorul.

NHibernate suportă o caracteristică numită lazy loading pentru a rezolva această problemă. Atunci când are loc încărcarea utilizatorul, puteți decide dacă să încărcați articolelor sau nu. Dacă alegeți să nu să facă acest lucru, colecția este inițializată în mod transparent atunci când aveți nevoie de ea.

9. Maparea obiect - relațională

Timpul a dovedit că bazele de date relaționale furnizează un bun mijloc de stocare a datelor, și că programarea orientată pe obiecte este o bună abordare pentru construirea de aplicații complexe. Cu cartografierea obiect/relațională, este posibil să se creeze un strat de traducere care să poată transforma cu ușurință obiecte în date relaționale și înapoi din nou.

Pe scurt, ORM este automatizarea (și, eventual, transparentă) persistenței obiectelor într-o aplicație la tabele dintr-o bază de date relațională, folosind meta date care descriu cartografierea între obiecte și baza de date. ORM, în esență, lucrează prin transformarea unor date de la o reprezentare la alta.

10. De ce anume ORM?

Timpul a dovedit că bazele de date relaționale furnizează un bun mijloc de stocare a datelor, și că programarea orientată pe obiecte este o bună abordare pentru construirea de aplicații complexe. Cu cartografierea obiect/relațională, este posibil să se creeze un strat de traducere care să poată transforma cu ușurință obiecte în date relaționale și înapoi din nou. Cum acest pod va manipula obiecte, el poate oferi multe dintre caracteristicile de care avem nevoie (cum ar fi punerea în cache, tranzacție și controlul concurenței). Tot ce trebuie să facem este de a oferi informații cu privire la modul de a mapa obiectelor la tabele.

Pe scurt, cartografierea obiect/relațională este automatizarea (și, eventual, transparentă) persistenței obiectelor într-o aplicație la tabele dintr-o bază de date relațională, folosind meta date care descriu cartografierea între obiecte și baza de date. ORM, în esență, lucrează prin transformarea unor date de la o reprezentare la alta.

Probleme rezolvate de ORM

Asimetria în modelare

Unul dintre costurile majore este în domeniul modelării. Modelele relaționale și obiectuale trebuie să cuprindă aceleași entități de afaceri. Dar, un programator orientat pe obiecte va modela aceste entități în mod diferit decât un modelator experimentat de date relaționale. Soluția este obișnuită constând din aplecarea și răsucirea modelul de obiect până când se potrivește cu tehnologia relațională care stă la bază.

Acest lucru se poate face cu succes, dar numai cu costul de a pierde o parte din avantajele obiect orientate. Țineți minte că modelarea relațională este susținută de teoria relațională. Orientarea obiectuală nu are o astfel de definiție matematică riguroasă sau un corp de lucrări teoretice.

Productivitatea și mentenanța

Asimetria din domeniul de modelare nu este singura problemă rezolvată de către ORM. Un instrument ca NHibernate te face mai productiv. Se elimină o mare parte din munca groh (mai mult decât te-ai aștepta) și vă permite să te concentrezi pe probleme de afaceri. Indiferent de strategia de dezvoltare a aplicației preferați-sus-jos, pornind de la un model de domeniu; sau de jos în sus, pornind de la o schemă a unei bază de date existentă - NHibernate utilizat împreună cu instrumentele adecvate va reduce semnificativ timpul de dezvoltare. Asimetria din domeniul de modelare nu este singura problemă rezolvată de către ORM. Un instrument ca NHibernate te face mai productiv. Se elimină o mare parte din munca groh (mai mult decât te-ai aștepta) și vă permite să te concentrezi pe probleme de afaceri. Indiferent de strategia de dezvoltare a aplicației preferați-sus-jos, pornind de la un model de domeniu; sau de jos în sus, pornind de la o schemă a unei bază de date existentă - NHibernate utilizat împreună cu instrumentele adecvate va reduce semnificativ timpul de dezvoltare.

Performanța

O afirmație comună este că persistența scrisă în hand-code poate fi întotdeauna cel puțin la fel de rapidă, și de multe ori mai rapidă, decât persistența automată. Acest lucru este valabil și în același sens că este adevărat că codul de asamblare poate fi întotdeauna cel puțin la fel de rapid ca și codul în .Net, cu alte cuvinte este pe lângă punctul dat rapiditatea.

Independența bazelor de date

NHibernate abstractizează aplicația dumneavoastră departe de baza de date SQL și dialectul acestuia. Faptul că susține o serie de baze de date diferite conferă un nivel de portabilitate a aplicației.

Nu ar trebui să vizezi în mod necesar să scrii aplicații complet independente față de bazele de date, deoarece capacitățile bazelor de date sunt diferite și obținerea de portabilitate completă s-ar realiza prin sacrificarea la o parte din puterea unor platforme mai puternice. Dar, un ORM poate contribui la atenuarea unora dintre riscurile asociate cu furnizorul de blocare. În plus, independenta de baze de date ajută în scenarii de dezvoltare pe care utilizați o bază de date locală ușoară, dar o dislocați pentru producție pe o platformă de baze de date diferite.

Concluzii

În această lucrare am vorbit despre conceptul de persistență și importanța NHibernate ca o tehnică de implementare pentru acesta. Persistența obiectelor semnifică că unele obiecte pot avea ciclu de viață mai mare decât procesul aplicației care le-a creat. Acestea pot fi salvate și recreate mai apoi pentru utilizarea ulterioară. Am vorbit despre arhitectura pe nivele care ajută la implementarea persistenței.

S-a făcut cunoștință cu multe caracteristici, care vor fi greu de scris prin hand-code. De asemenea, s-a discutat despre cum NHibernate rezolvă discrepanța dintre modelul obiectul și relațional. Aceasta apare când în joc apar SGBD bazate pe SQL. În prima instanță un graf mare de obiecte nu poate fi salvat într-o tabelă a bazei de date. Acestea trebuie dezasamblate și salvate în coloanele tabelei SQL.

În sfârșit s-a studiat conceptul de ORM. S-a discutat ce avantaje prezintă abordarea dată. ORM are ca scop ușurarea lucrului ce cade asupra programatorului, spre exemplu scrierea de scripturi sau joncțiuni asupra tabelelor. Astfel ORM și NHibernate oferă o portabilitate a bazelor de date, anumite tehnici de optimizare, cum ar fi cache.

ORM este cea mai buna soluție disponibilă la momentul de astăzi și este un colac de salvare pentru programatorii care se întâlnesc cu asimetria obiect/ relațională. NHibernate este un instrument fantastic al ORM-ului care permite combinarea beneficiilor ambelor concepte, cel obiectual și relațional în același timp.

Bibliografie

1. NHibernate Community 2015. Home. *http://nhibernate.info/.* [Online] [Cited: May 13, 2016.] http://nhibernate.info/.

2. Schenker, Gabriel. Your first NHibernate based application. *http://www.nhforge.org.* [Online] [Cited: May 14, 2016.] http://www.nhforge.org/wikis/howtonh/your-first-nhibernate-based-application.aspx.

3. Pierra Henri Kuate, Tobin Harris, Christian Bauer, Gavin King. *NHibernate in Action.* s.l. : Manning Publications Co., 2009. ISBN 978-1-932394-92-4.

4. Kazembe, Rodrick. NHibernate Tutorial C#. *http://www.kode-blog.com.* [Online] [Cited: May 14, 2016.] http://www.kode-blog.com/2014/01/nhibernate-tutorial-csharp/.