Tecnologie di Sviluppo per il Web

Programmazione su Basi di Dati: Framework Hibernate – Parte a

versione 1.0

G. Mecca – Università della Basilicata – mecca@unibas.it

Sommario

- Introduzione
- Introduzione a Hibernate
  - ⇨Componenti dell'API
  - ⇨Configurazione e Metadati

Introduzione

- Framework per la persistenza
  - ⇨“persistence framework”
  - ⇨infrastruttura di classi e di regole per lo sviluppo di applicazioni client-server su basi di dati (relazionali)
- Obiettivo
  - ⇨snellire significativamente la scrittura del codice relativo allo strato di persistenza

Introduzione

- Obiettivi specifici
  - ⇨eliminare la necessità di scrivere codice SQL per effettuare le operazioni CRUD
  - ⇨supportare la gestione del “mapping” tra classi del modello e tabelle della base di dati (riferimenti e caricamento)
  - ⇨fornire soluzione ai problemi ricorrenti di programmazione (generazione id, utilizzo pool di connessioni, gestione delle versioni e locking ottimistico)

Introduzione

- Obiettivi specifici (continua)
  - ⇨consentire di utilizzare dbms diversi astraendo rispetto ai vari “dialetti”
  - ⇨eliminare la necessità di scrivere codice accurato di test sulla persistenza
- Tutti questi obiettivi
  - ⇨implementando le operazioni nel modo più robusto ed efficiente possibile

Introduzione

- Attenzione
  - ⇨si tratta di sistemi molto complessi (molto più complessi di un framework per i test di regressione o per il modello 2/MVC)
  - ⇨di complessità comparabile ai problemi di programmazione sulla persistenza
  - ⇨quindi sono da considerarsi risorse ad alto valore aggiunto nel ciclo di sviluppo

Introduzione

- Svantaggi (come tutti i framework)
  - ⇨richiedono il rispetto di alcune regole, e quindi irrigidiscono lo sviluppo
  - ⇨complicano l'infrastruttura complessiva (introducendo nel sistema dipendenze rispetto a varie librerie)
- Al solito
  - ⇨l'adozione è consigliata solo in sistemi di dimensione medio-grande

Introduzione

- Funzionamento tipico del framework
  - ⇨lo sviluppatore fornisce una collezione di "metadati" che descrivono il "mapping" desiderato tra bean ed ennuple della base di dati
  - ⇨il sistema utilizza il mapping per fornire il servizio di persistenza
- Classificazione dei framework
  - ⇨due categorie principali
  - ⇨generatori di codice
  - ⇨sistemi di persistenza ("persistence manager")

Introduzione

- Generatori di codice
  - ⇨sistemi che, fornita una descrizione della base di dati, generano automaticamente a tempo di costruzione il codice di DAO (ed eventualmente DTO)
  - ⇨il codice generato viene poi utilizzato nell'applicazione
  - ⇨essendo stato generato automaticamente non richiede test di regressione (se non verifiche molto rapide)

Introduzione

>> it.unibas.acitorque

- Un esempio di generatore
  - ⇨Apache Torque, progetto open-source della Apache Software Foundation
  - ⇨appartiene al sottoprogetto db.apache.org
  - ⇨si tratta in sintesi di un generatore di DAO (chiamati "peer" nella terminologia di Torque)
  - ⇨l'architettura applicativa è quella tradizionale con DAO-DTO, ma non è necessario scrivere una parte consistente del codice

Introduzione

- Persistence manager
  - ⇨API che consente di rendere persistenti gli oggetti dell'applicazione a tempo di esecuzione senza necessità di sviluppare codice
- Idea
  - ⇨fornire un componente che sia sufficiente generico da effettuare operazioni CRUD su JavaBeans generici

Introduzione

- Esempi di persistence manager
  - ⇨Enterprise JavaBeans CMP ("Container Managed Persistence")
  - ⇨JDO ("Java Data Objects") – tecnologia standard sviluppata nell'ambito dello Java Communiti Process
  - ⇨OJB ("Object Relational Bridge") – sottoprogetto di db.apache.org
  - ⇨Hibernate, il leader incontrastato

Introduzione

- Differenze tra i due sistemi
  - ⇨sono abbastanza sfumate; sintetizziamo
  - ⇨i persistence manager sono più facilmente utilizzabili nei casi in cui la base di dati deve essere sviluppata ex-novo (tipicamente il processo procede dai bean alla base di dati)
  - ⇨i generatori di codice sono utili nei casi in cui si lavora con basi di dati esistenti (“legacy”) (il processo procede dalla base di dati ai bean)

Introduzione a Hibernate

- Hibernate
  - ⇨progetto open source
  - ⇨disponibile su www.hibernate.org
- In sintesi
  - ⇨un persistence manager che ha profondamente influenzato le tecniche di persistenza nella piattaforma Java
  - ⇨le nuove specifiche EJB 3.0 sono fortemente basate su Hibernate

Introduzione a Hibernate

- L'idea alla base di Hibernate
  - ⇨il framework fornisce un componente, chiamato Session (org.hibernate.Session) che è in sostanza un "super DAO" generico
  - ⇨fornisce metodi save(), update(), delete(), find() attraverso i quali è possibile realizzare operazioni CRUD su qualsiasi JavaBean
  - ⇨senza dover specificare codice SQL

Componenti dell'API

- L'API di Hibernate
  - ⇨tre componenti fondamentali
  - ⇨SessionFactory (org.hibernate.SessionFactory)
  - ⇨Session
  - ⇨Transaction (org.hibernate.Transaction)
- SessionFactory
  - ⇨è il componente che fornisce le Session; deve essere unico per tutta l'applicazione
  - ⇨analogo alla DataSource di un'appl. JDBC

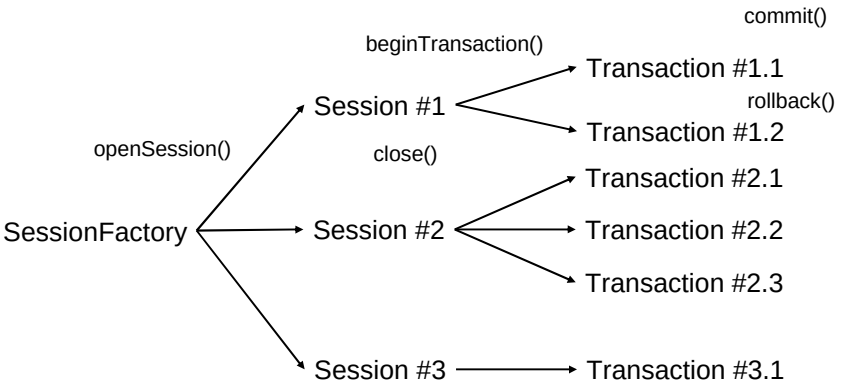
Componenti dell'API

- Session
  - ⇨gli oggetti di tipo Session vengono utilizzati per realizzare tutte le operazioni CRUD
  - ⇨si ottengono dalla SessionFactory utilizzando il metodo Session openSession()
  - ⇨incapsulano al loro interno la gestione dei componenti JDBC (che quindi il programmatore non manipola direttamente)
  - ⇨normalmente: una sessione per ogni azione

Componenti dell'API

- Transaction
  - ⇨Hibernate è orientato alla modalità concatenata e richiede una gestione esplicita delle transazioni
  - ⇨gli oggetti Transaction si ottengono dalle sessioni con il metodo beginTransaction() e si gestiscono con commit() e rollback()
  - ⇨incapsulano al loro interno una connessione JDBC
  - ⇨normalmente una o più transazioni per ciascuna Session

Componenti dell'API



Componenti dell'API

- Tipico funzionamento dell'applicazione
  - ⇨ un componente con visibilità globale (es: PersistenceManager) si occupa di inizializzare la SessionFactory all'avvio
  - ⇨ per eseguire le azioni viene acquisito il riferimento alla SessionFactory
  - ⇨ viene aperta una Session
  - ⇨ vengono avviate una o più Transaction utilizzando la Session, di cui l'azione gestisce il commit e il rollback
  - ⇨ al termine l'azione chiude la Session

Componenti dell'API

- Un esempio
  - ⇨consideriamo il modello dell'applicazione dell'ACI con Hibernate
  - ⇨Proprietari ed Automobili
  - ⇨l'azione che inserisce un proprietario
  - ⇨utilizza una classe PersistenceManager nel cui blocco di inizializzazione statico viene inizializzata la SessionFactory

Componenti dell'API

```
public void InserisciProprietario (Proprietario proprietario) {
      SessionFactory factory = PersistenceManager.getSessionFactory();
      Session session = null;
      Transaction transaction = null;
      try {
         session = factory.openSession();
         transaction = session.beginTransaction();
         session.save(proprietario);
         transaction.commit();
      } catch (HibernateException e) {
         if (transaction != null) { transaction.rollback(); }
      } finally {
         session.close();
      }
   }
```

Componenti dell'API

- HibernateException
  - si tratta di una eccezione non controllata (come tutte le eccezioni lanciate da Hibern.)
  - per deliberata scelta degli sviluppatori
- In sintesi
  - rappresenta una condizione di errore non recuperabile nel framework
  - se si verifica è necessario effettuare subito il rollback(), chiudere la Sessione e riavviare

Componenti dell'API

- I metodi di Session (>>)
  - save(Object o) (o persist(Object o))
  - update(Object o)
  - delete(Object o)
  - saveOrUpdate(Object o)
  - refresh(Object o)
  - lock(Object o, int lockMode)

Componenti dell'API

- Per le operazioni di "retrieve"
  - Session fornisce due metodi utili nel caso in cui si conosca la chiave primaria dell'oggetto
  - void load(Object o, T id): se non esiste una ennupla con l'id specificato solleva eccez.
  - void get(Object o, T id): se non esiste una ennupla assegna ad o il valore null
- Nota
  - la chiave primaria è normalmente sintetica e quindi raramente viene manipolata dal programmatore

Componenti dell'API

- In alternativa
  - è possibile effettuare interrogazioni sulla base di dati
- Varie possibilità
  - specificare query SQL (poco usata)
  - costruire interrogazioni utilizzando esclusivamente l'API Criteria (macchinoso)
  - utilizzare HQL

Componenti dell'API

- Hibernate Query Language (HQL)
  - ⇨il linguaggio di interrogazione fornito da Hibernate
  - ⇨è semplice e ha una sintassi compatta
  - ⇨è orientato agli oggetti e non alle ennuple
  - ⇨consente di effettuare interrogazioni sulla base di dati specificando condizioni sugli oggetti da estrarre e non sulle tabelle

Componenti dell'API

- Un esempio
  - ⇨con riferimento all'esempio precedente, supponiamo di voler trovare un proprietario per codice fiscale
  - ⇨essendo la chiave primaria un identificatore sintetico non possiamo usare load() e get()
  - ⇨costruiamo una query HQL

Componenti dell'API

```
public Proprietario cercaProprietario (String codiceFiscale) {
      SessionFactory factory = PersistenceManager.getSessionFactory();
      Session session = null;
      try {
          session = factory.openSession();
          Query query = session.createQuery("from Proprietario where codicefiscale = ?");
          query.setString(0, codiceFiscale);
          List proprietari = query.list();
          if (proprietari.size() != 0) {
              return (Proprietari)proprietari.get(0);
          }
      } catch (HibernateException e) {
          logger.logSevere(e);
      } finally {
          session.close();
      }
      return null;
  }
```


Componenti dell'API

- Sintassi di HQL
  - ⇨simile alla sintassi SQL
- Ma...
  - ⇨attenzione alla differenza: si specificano condizioni su oggetti e non su tabelle
  - ⇨è possibile omettere la clausola SELECT nel caso in cui sia necessario estrarre tutti gli oggetti che soddisfano la query

Componenti dell'API

- Nella clausola where
  - ⇨per specificare le condizioni si utilizza una sintassi basata su quella dei PreparedStatements
  - ⇨con una importante differenza: i parametri vengono numerati a partire da 0 e non da 1
- Nota
  - ⇨c'è anche una sintassi alternativa in cui ai parametri viene attribuito un nome esplicito

Configurazione e Metadati

- Una domanda spontanea
  - ⇨ma come può funzionare ?
- Il principio alla base di Hibernate
  - ⇨utilizzo estensivo della riflessione per ispezionare le caratteristiche degli oggetti
- Una regola fondamentale
  - ⇨gli oggetti del modello devono essere JavaBeans

Configurazione e Metadati

- Nonostante questo
  - ⇨Hibernate ha bisogno di informazione aggiuntiva
- Due categorie di informazioni
  - ⇨parametri di configurazione (es: base di dati, driver, configurazione del pool)
  - ⇨informazioni aggiuntive sul mapping tra oggetti e base di dati (es: generazione identificatori)

Configurazione e Metadati

- Informazioni sul mapping
  - ⇨si tratta di metadati (dati che descrivono come trattare i dati dell'applicazione)
- File hbm (Hibernate mapping)
  - ⇨metadati usati da Hibernate per il mapping
  - ⇨file .xml conformi ad uno specifico DTD
  - ⇨uno per ciascun bean dell'applicazione
  - ⇨normalmente nella stessa cartella in cui è contenuto il file .class

Configurazione e Metadati

- Funzione del file .hbm
  - completare le informazioni che Hibernate è in grado di ottenere dalla riflessione
- In particolare
  - quali proprietà rendere persistenti
  - che tipo di strategia utilizzare per gli identificatori
  - eventuale strategia per le versioni
  - come gestire i riferimenti ad altre classi persistenti (>>)

Configurazione e Metadati

- Nella maggioranza dei casi
  - contiene l'elenco delle proprietà della classe
  - una elemento di tipo id che specifica la strategia di generazione (è possibile usare hilo, ma ci sono molte alternative)
  - la definizione dei riferimenti
- Esempi di file di configurazione
  - Proprietario.hbm.xml
  - Automobile.hbm.xml

```
<?xml version="1.0" ?>

<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="it.unibas.esempihibernate.es1mbidir">

    <class name="Automobile" table="automobili">

        <id name="id" type="long">
          <generator class="hilo" />
        </id>

        <property name="targa" not-null="true" unique="true" />
        <property name="modello" not-null="true" />
        <property name="cilindrata" not-null="true" />

      <many-to-one name="proprietario" class="Proprietario" column="idproprietario" />

    </class>

</hibernate-mapping>
```

>> Proprietario.hbm.xml
>> Automobile.hbm.xml

Configurazione e Metadati

- Nota
  - ⇨Hibernate cerca di stabilire il tipo più adatto con cui salvare ciascuna proprietà nel DBMS utilizzando la riflessione
  - ⇨è possibile fornire altre informazioni per orientare questa scelta
  - ⇨es: nome della colonna e lunghezza del dato
  - ⇨es: tipo di dato nel DBMS, indispensabile per esempio per le date

Configurazione e Metadati

- Specificare i tipi per le proprietà
  - ⇨nel file .hbm è possibile specificare per la proprietà uno “hibernate mapping type”
  - ⇨insieme di tipi neutro rispetto al DBMS, che hibernate traduce nello specifico dialetto
- Esempio

```
<property name=“date” type=“timestamp”
          column=“DATA_EVENTO” />
<property name=“targa” type=“string”  length=“7”
          column=“TARGA” />
```


Configurazione e Metadati

- id degli oggetti
  - ⇨tutti i bean gestiti da Hibernate devono aver una proprietà id che contiene la chiave primaria della ennupla corrispondente
  - ⇨tipicamente di tipo long (ma può essere anche int o String, a seconda della strategia di generazione)
  - ⇨nel caso in cui non si voglia cambiare l’interfaccia del bean, i metodi get e set relativi possono essere privati

Configurazione e Metadati

- Al solito, quindi
  - ⇨due identificatori per lo stesso oggetto
  - ⇨OID assegnato dalla macchina virtuale
  - ⇨id prelevato dalla base di dati
- Inoltre, al solito
  - ⇨l'id viene assegnato dal metodo save() e quindi è indefinito fino al momento dell'inserimento della ennupla nella base di dati

Configurazione e Metadati

- Configurazione del framework
  - ⇨ci sono vari modi per fornire parametri di configurazione
- I due modi principali
  - ⇨fornire un file di properties hibernate.properties
  - ⇨fornire un file xml hibernate.cfg.xml
  - ⇨questo secondo modo è più flessibile

Configurazione e Metadati

- hibernate.cfg.xml
  - ⇨fornisce tutti i parametri per la configurazione della SessionFactory
  - ⇨deve essere disponibile in una cartella radice del classpath
  - ⇨contiene in aggiunta un elenco di tutti i file .hbm da utilizzare per il mapping

>> hibernate.cfg.xml

Configurazione e Metadati

- Per l'inizializzazione della SessionFactory
  - ⇨è necessario creare un oggetto di tipo Configuration, che provoca il caricamento del file hibernate.cfg.xml
  - ⇨dall'oggetto Configuration è possibile ottenere la SessionFactory chiamando il metodo configure()
  - ⇨e successivamente SessionFactory buildSessionFactory()

```
public class PersistenceManager {

    private static SessionFactory sessionFactory;

    static {
        try {
            Configuration configuration = new Configuration();
            sessionFactory =
                    configuration.configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
```


Riassumendo

- Introduzione
- Introduzione a Hibernate
  - ⇨Componenti dell'API
  - ⇨Configurazione e Metadati

Termini della Licenza