Chado API via Java & Hibernate

January 19th, 2007

Robert Bruggner
VectorBase.org

Overview

- Background
- Quick Hibernate Overview
- Hibernate Connectivity and O/R Mapping Example
- GMOD Demo

Background

- VectorBase
  - A bioinformatic resource center for invertebrate vectors of human pathogens
- Responsible for storage and display of multiple organisms' genomes
  - *Anopheles gambiae, Aedes aegypti, Ixodes scapularis, Culex pipiens* and so on....
- Want to store data for many organisms- Chado a natural choice
- Ensembl Genome Browser already used for *A. gambiae*
  - Wrote Ensembl API Database adaptor for Chado... Not maintainable.
- Use Both Databases
  - Transfer genomic data from Ensembl to Chado
    - Search Engine and Indexer
    - Run DAS
    - Export data via ChadoXML and GFF3
- Need API for Database I/O

Hibernate Background

- Hibernate
  - They say: “A powerful, high performance object/relational persistence and query service.”
- Automates the persistence of plain old Java objects (POJO)
  - User maps their POJO properties to database tables via XML (HBM File).
  - Persist a specific object by storing it the database.
- Intelligent Database I/O
  - Smart detection of “Dirty Properties” when performing Save / Update / Delete.
  - Cascadable Save / Update / Delete for complex objects.

Hibernate Database Connectivity

- Configure Hibernate in hibernate.cfg.xml
- Define a Data Source
  - We use a simple, single JDBC connection Chado
    - Can be configured to use a connection pool or data source accessible by the Java Naming and Directory Interface (JNDI).
  - Define a connection “dialect”
    - `org.hibernate.dialect.PostgreSQLDialect`
- Describe the relationship between Java objects and database tables
  - Use XML to describe where to store POJO property data in the database
- Create a new Hibernate Session based on the configuration
- Begin a transaction to start performing work

POJO and HBM Example file - CV

```
public class CV {
     private int cv_id;
     private String name;
     private String definition;

     public property gettersandsetters() {
          ....
     }
     public boolean equals(CV comparaCV){
          ....
     }
     public int hashCode(){
          ...
     }
}

<hibernate-mapping>
     <class name="org.vectorbase.chadoAPI.chadoObjects.CV" table="cv">
          <id name="cv_id" column="cv_id" unsaved-value="undefined">
               <generator class="sequence">
                    <param name="sequence">cv_cv_id_seq</param>
               </generator>
          </id>
          <property name="name" column=”name” type="java.lang.String” not-null="true"/>
          <property name="definition" column=”definition” type="java.lang.String”/>
     </class>
</hibernate-mapping>
```

HBM Example CVTerm

```
public class CVTerm {

     private int cvterm_id;
     private CV cv;
     private String name;
     private String definition;
     private DBXref dbxref;
     private int is_obsolete;
     private int is_relationshiptype;

.....

<hibernate-mapping>
     <class name="org.vectorbase.chadoAPI.chadoObjects.CVTerm" table="cvterm">
          <id name="cvterm_id" column="cvterm_id" unsaved-value="undefined">
               <generator class="sequence">
                    <param name="sequence">cvterm_cvterm_id_seq</param>
               </generator>
          </id>
          <many-to-one name="cv" class="org.vectorbase.chadoAPI.chadoObjects.CV" column="cv_id"
not-null="true" cascade="save-update"/>
          <property name="name" not-null="true" type="java.lang.String"/>
          <property name="definition"/>
          <one-to-one name="dbxref" class="org.vectorbase.chadoAPI.chadoObjects.DBXref"
cascade="all"/>
          <property name="is_obsolete"/>
          <property name="is_relationshiptype"/>
     </class>
</hibernate-mapping>
```

Hibernate Object Retrieve

```
import org.hibernate.Session;
import org.vectorbase.chadoAPI.CVTerm;
import org.vectorbase.chadoAPI.CV;

// Load the configuration from hibernate.cfg.xml

// Build a session factory first (not shown)

// Get the session based on the configuration and begin transaction
Session session = HibernateSessionFactory.getCurrentSession();
session.beginTransaction();

// Load a CVTerm by its ID
CVTerm cvt = (CVTerm) session.get(CVTerm.class,1);

// Load a CVTerm using HQL
CVTerm cvt = session.createQuery("from CVTerm where name=?").setString(0,"name").uniqueResult();

// Print out the name of the cvterm
System.out.println(cvt.getName());

// Get the cv that the cvterm is associated with
// Hibernate doesn't return the cv_id - it returns a CV Object.
CV cv = cvt.getCv();

// Print out the cv's name
System.out.println(cv.getName());
```

Hibernate Object Update

```
import org.hibernate.Session;
import org.vectorbase.chadoAPI.CVTerm;


// Load the configuration from hibernate.cfg.xml

// Build a session factory first (not shown)

// Get the session based on the configuration and begin transaction
Session session = HibernateSessionFactory.getCurrentSession();
session.beginTransaction();


// Load a CVTerm by its ID
CVTerm cvt = (CVTerm) session.get(CVTerm.class,1);

// Change cvt's name
cvt.setName("New CVTerm name");

// Save!
// Generated SQL updates "Dirty" properties (name, in this case)
session.save(cvt);

// Commit data to database
session.commit();
```

Hibernate Save

```
import org.hibernate.Session;
import org.vectorbase.chadoAPI.CVTerm;
import org.vectorbase.chadoAPI.CV;

// Load the configuration from hibernate.cfg.xml
// Build a session factory first and get begin transaction (not shown)

// Make a new CV
CV new_cv = new CV();
new_cv.setName("New CV");
new_cv.setDefinition("New CV Def");

// Make a new cvterm for that cv
CVTerm new_cvterm = new CVTerm();
new_cvterm.setName("New CVTerm Name");
// ..... save dbxref etc......

// Add that CVTerm to our new CV
new_cv.addCVTerm(new_cvterm);

// Save the new data...
// Hibernate recognizes that it has to first save new_cv, then save new_cvterm.
session.save(new_cvterm);

session.commit();

// You can see the new id's assigned by the database
System.out.println(new_cv.getCv_id());
System.out.println(new_cvterm.getCvterm_id());
```

Inheritance

```
<hibernate-mapping>
     <class name="org.vectorbase.chadoAPI.chadoObjects.Feature" table="feature" discriminator-
value="not null">
          <id name="feature_id" column="feature_id" unsaved-value="undefined">
               <generator class="sequence">
                    <param name="sequence">feature_feature_id_seq</param>
               </generator>
          </id>
          <discriminator column="type_id" type="integer" insert="false"/>

          <many-to-one name="dbxref" class="org.vectorbase.chadoAPI.chadoObjects.DBXref"
          column="dbxref_id" cascade="all"/>
          <many-to-one name="organism" class="org.vectorbase.chadoAPI.chadoObjects.Organism"
          column="organism_id" not-null="true" cascade="save-update"/>
          <property name="name"/>
          .....

<hibernate-mapping>
     <subclass name="org.vectorbase.chadoAPI.chadoFeatures.Gene"
     extends="org.vectorbase.chadoAPI.chadoObjects.Feature" discriminator-value="767">
     </subclass>
</hibernate-mapping>
```

Write custom methods for specific sub-classes

ChadoAPI

- POJO Mappings
  - CV, CVTerm, DB, DBXref, Feature, FeatureCVTerm, FeatureDBXref, FeatureLoc, FeatureProp, FeatureRelationship, FeatureSynonym, Organism, Pub, Synonym
- Extended Features
  - Chromosome, Gene, Transcript, Exon, Protein
- Constants
  - CVTerms, FeatureFeatureRelationships, Ontologies
- Special
  - ChadoAdapter

GMOD Example

```
// Set up our session and begin transaction
Session session = HibernateUtil.getSessionFactory().getCurrentSession();
session.beginTransaction();

// Make a chado adpator and load up some utility objects
ChadoAdaptor ca = new ChadoAdaptor();
Chromosome c = ca.fetchChromosomeByUniqueName("fake_chromosome");
Pub null_pub = ca.fetchPubByPubID(1);
Organism agambiae = ca.fetchOrganismByScientificName("Anopheles","gambiae");

// Begin GMOD Demo Code

// Make our new gene;
Gene xfile = new Gene();
xfile.setOrganism(agambiae);
xfile.setUniquename("xfile");
xfile.setDescription("A test gene for GMOD meeting");

/* Set the location of our gene. No need to set coordinates because they'll be updated
* based on the exon boundaries.
*/
FeatureLoc xfile_loc = new FeatureLoc();
xfile_loc.setSrcfeature(c);
xfile_loc.setStrand(1);
xfile.setFeatureLoc(xfile_loc);

// Add synonyms to xfile
xfile.createNewFeatureSynonym("mulder", null_pub, CVTerms.EXACT_SYNONYM);
xfile.createNewFeatureSynonym("scully", null_pub, CVTerms.EXACT_SYNONYM);
```

GMOD Example

```
// Create a new transcript for our gene.
Transcript t = xfile.createGeneTranscript("xfile-RA");

// Create some exons for that transcript.
t.createTranscriptExon("xfile:1", 13691, 13767);
t.createTranscriptExon("xfile:2", 14687, 14720);

// Save our new gene
session.save(xfile);
System.out.println("xfile feature_id is " + xfile.getFeature_id());

// Fetch our saved gene from the database
Gene xfile_r = ca.fetchGeneByUniqueName("xfile");
System.out.println("symbol: " + xfile_r.getUniquename());
System.out.print("synonyms: ");
for (FeatureSynonym fs : xfile_r.getFeatureSynonyms()){
      System.out.print(fs.getSynonym().getName() + " ");
}

System.out.println("description: " + xfile_r.getDescription());
System.out.println("type: " + xfile_r.getType().getName());

for (Transcript tx : xfile_r.fetchAllTranscripts()){
   for (Exon e : tx.fetchAllExons()){
      System.out.println(e.getUniquename() + " Start:\t" + e.getFeatureLoc().getFmin());
      System.out.println(e.getUniquename() + " End:\t" + e.getFeatureLoc().getFmax());
      System.out.println("\tSrcFeatureID: " + e.getFeatureLoc().getSrcfeature().getFeature_id());
   }
   System.out.println(">" + tx.getUniquename());
   System.out.println(tx.generateTranscriptSequenceFromExons().toUpperCase());
}
```

GMOD Update & Delete

```
// Lets update our name...
xfile_r.setUniquename("x-file");

session.save(xfile_r);

// Not part of the ChadoAdaptor utility object, but a good example of HQL
List<Gene> genes = (List<Gene>)session.createQuery("from Gene where uniquename like ?").setString(0,”x-%”).list();

for (Gene g : genes){
      System.out.println(g.getFeature_id() +
                         "\t" + g.getUniquename() +
                         "\t" + g.getOrganism().getGenus() +
                         " " + g.getOrganism().getSpecies());
}

// Deleting... hmm...
Gene delete_me = ca.fetchGeneByUniqueName("x-ray");
session.delete(delete_me);


// All Finished
session.getTransaction().commit();
```

To Do...

- Completeness
- Exception Handling
- Performance Tuning

Thanks!

- VectorBase People
  - Frank Collins, EO Stinson, Ryan Butler
- GMOD
- NIAID