Introduction to Hibernate

Overview

About the Presenter

- Cliff Meyers
  - cliff.meyers@gmail.com
- Application Developer
  - US Department of State, Bureau of Information Resource Management
- ColdFusion, Java, Oracle, MS SQL

What is Hibernate?

- Hibernate is an object-relational mapping tool (ORM) that allows for persisting Java objects in a relational database
- Driven by XML configuration files to configure data connectivity and map classes to database tables
- Not a Java/SQL code generation tool
  - Developer writes code to call API
  - API executes necessary SQL at runtime

Why Use Hibernate?

- Eliminate need for repetitive SQL
- Work with classes and objects instead of queries and result sets
  - More OO, less procedural
- Mapping approach can resist changes in object/data model more easily
- Strong support for caching

Why Use Hibernate?

- Handles all create-read-update-delete (CRUD) operations using simple API; no SQL
- Generates DDL scripts to create DB schema (tables, constraints, sequences)
- Flexibility to hand-tune SQL and call stored procedures to optimize performance
- Supports over 20 RDBMS; change the database by tweaking configuration files

Introduction to Hiberate

The Basics

Simple Object Model

- AuctionItem
  - description
  - type
  - successfulBid
- Bid
  - amount
  - datetime
- AuctionItem has zero or more bids
- Auction item has zero or one successfulBid

Plain Old Java Object (POJO)

- Default constructor
- Identifier property
- Get/set pairs
- Collection property is an interface type

```
public class AuctionItem {
    private Long _id;
    private Set _bids;
    private Bid _successfulBid
    private String _description;

    public Long getId() {
          return  _id;
    }
    private void setId(Long id) {
          _id = id;
    }
    public String getDescription() {
          return _description;
    }
    public void setDescription(String desc) {
          _description = desc;
    }
    …
}
```

XML Mapping File

- Readable metadata
- Column / table mappings
- Surrogate key generation strategy
- Collection metadata
- Fetching strategies

```
<class name="AuctionItem"
   table="AUCTION_ITEM">
   <id name="id" column="ITEM_ID">
        <generator class="native"/>
   </id>
   <property name="description"
   column="DESCR"/>
   <many-to-one name="successfulBid"
        column="SUCCESSFUL_BID_ID"/>
   <set name="bids"
        cascade="all"
        lazy="true">
        <key column="ITEM_ID"/>
        <one-to-many class="Bid"/>
   </set>
</class>
```

Creating Objects

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

AuctionItem item = new AuctionItem();
item.setDescription(“Batman Begins”);
item.setType(“DVD”);
session.save(item);

tx.commit();
session.close();
```

Updating Objects

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
item.setDescription(newDescription);

tx.commit();
session.close();
```

Deleting Objects

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
session.delete(item);

tx.commit();
session.close();
```

Selecting Objects

- Hibernate Query Language (HQL), similar to SQL

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

List allAuctions = session.createQuery("
   select item
   from AuctionItem item
        join item.bids bid
   where item.description like 'Batman%'
        and bid.amount < 15
").list();

tx.commit();
session.close();
```

Introduction to Hibernate

The Details

Key Hibernate Classes

- Configuration – uses mapping and database connection metadata to create SessionFactory
- **SessionFactory – thread-safe cache of compiled mappings for database; created once at application startup (expensive)**
- Session – represents a “conversation” between application and database; holds 1st level cache of objects
- Transaction – an atomic unit of work

Configuration

hibernate.properties

```
hibernate.dialect =
   org.hibernate.dialect.SQLServerDialect
hibernate.connection.driver_class = net.sf.jtds.Driver
hibernate.connection.url =
   jdbc:sqlserver://localhost/db:1433
hibernate.connection.username = myuser
hibernate.connection.password = mypass
```

- Also configurable via using XML
- Several ways to add mapping files to configuration, including XML or API-based

SessionFactory

- Once the Configuration is prepared, obtaining the SessionFactory is easy:

```
Configuration cfg = new Configuration();
// … do some configuration …
cfg.configure();
SessionFactory sf =
  cfg.buildSessionFactory();
```

Typical Usage Pattern

```
Session s = sessionFactory.openSession();
Transaction tx = s.beginTransaction();
// … perform some operation …
tx.commit();
s.close();
```

- Although some additional boilerplate code is required for proper error handling***

Hibernate Querying Options

- HQL
  - Syntax similar to SQL
  - Unlike SQL, HQL is still database-agnostic
- Criteria
  - Java-based API for building queries
  - Good for queries that are built up using lots of conditional logic; avoids messy string manipulation
- SQL / PLSQL
  - Often needed to optimize for performance or leverage vendor-specific features

Example Queries

- Criteria API

```
List auctionItems =
session.createCriteria(AuctionItem.class)
     .setFetchMode(“bids”, FetchMode.EAGER)
     .add( Expression.like(“description”, description) )
     .createCriteria(“successfulBid”)
            .add( Expression.gt(“amount”, minAmount) )
     .list();
```

- Equivalent HQL:

```
from AuctionItem item
     left join fetch item.bids
where item.description like :description
     and item.successfulbid.amount > :minAmount
```

Example Queries

- "Query by example" approach

```
AuctionItem item = new AuctionItem();
item.setDescription("hib");
Bid bid = new Bid();
bid.setAmount(1.0);
List auctionItems =
   session.createCriteria(AuctionItem.class)
     .add( Example.create(item).enableLike(MatchMode.START) )
     .createCriteria("bids")
              .add( Example.create(bid) )
     .list();
```

- Equivalent HQL:

```
from AuctionItem item
     join item.bids bid
where item.description like 'hib%'
     and bid.amount > 1.0
```

Introduction to Hibernate

More than Hibernate

Simplifying Hibernate Boilerplate

```
public void deleteItem(Long itemId) throws HibernateException
   {
    Session session = null;
    try {
       session = sessionFactory.openSession();
       tx = session.beginTransaction();
       AuctionItem item =
         (AuctionItem) session.get(ActionItem.class, itemId);
       session.delete(item);
       tx.commit();
    catch(HibernateException ex) {
       tx.rollback();
       // … do something useful, like log and rethrow exception
    } finally {
       session.close();
    }
}
```

Spring to the Rescue

- Spring offers a convenient `HibernateSupportDao` class that offers “free” convenience methods and exception handling

```
public void deleteItem(Long itemId)
   throws DataAccessException {
   AuctionItem item = (AuctionItem)
   getHibernateTemplate().get(ActionItem.class,
   itemId);
   session.delete(item);
}
```

Hibernate with ColdFusion

- Write Java classes, mapping files and Hibernate configuration
- Bundle into a .jar file
- Deploy to ColdFusion server along with Hibernate’s required .jar files

Hibernate with ColdFusion

- Use the “multiserver” install for CF
- Create a separate instance for the hybrid application
- Deploy files to this location:

```
{jrun.home}/servers/myServer/cfusion.ear/
cfusion.war/WEB-INF/cfusion/lib
```

Hibernate with ColdFusion

- Hibernate’s required .jar files:

```
hibernate3.jar
antlr.jar
asm.jar
asm-attrs.jar
cglib.jar
dom4j.jar
ehcache.jar
jta.jar
commons-collections.jar
commons-logging.jar
log4j.jar
```

Hibernate with ColdFusion

- Database connectivity:
  - Hibernate can use JNDI datasources (similar to CF datasources) instead of connection information in `hibernate.properties`
  - JNDI can be configured through the JRun administration console (JMC)
    - Only available in CF “multiserver” install

Hibernate with ColdFusion

- Java is a strong-typed language whereas ColdFusion is weakly-typed
  - Use of ColdFusion’s `javaCast()` function is required to distinguish between methods like `getAuctionItem(Long itemId)` and `getAuctionItem(String description)`
  - Quantity of casting can get tedious
- ColdFusion’s logging infrastructure (Log4J) suppresses all of Hibernate’s logging info, making it difficult to debug deployed code

Any Questions?

References

- Hibernate docs
  - http://hibernate.org
- Hibernate in Action
  - http://manning.com/bauer/
- Spring docs
  - http://springframework.org

Thank You!