Course Booklet for Hibernate framework

Know how of Hibernate frameworks

By Emertxe Trainers

version1.0 (June 20, 2012)

All rights reserved. Copyright @ 2012
Emertxe Information Technologies Pvt Ltd
(http://www.emertxe.com)

Table of Contents

Introduction to Hibernate framework

Hibernate was started in 2001 by Gavin King as an alternative to using EJB2-style entity beans. Its mission back then was to simply offer better persistence capabilities than offered by EJB2 by simplifying the complexities and allowing for missing features.

Early in 2003, the Hibernate development team began Hibernate2 releases which offered many significant improvements over the first release.

JBoss, Inc. (now part of Red Hat) later hired the lead Hibernate developers and worked with them in supporting Hibernate. Hibernate is part of JBoss (a division of Red Hat) Enterprise Middleware System (JEMS) suite of products.

Hibernate is an Object-relational mapping (ORM) tool. Object-relational mapping or ORM is a programming method for mapping the objects to the relational model where entities/classes are mapped to tables, instances are mapped to rows and attributes of instances are mapped to columns of table.

A “virtual object database” is created that can be used from within the programming language.

Hibernate is a persistence framework which is used to persist data from Java environment to database. Persistence is a process of storing the data to some permanent medium and retrieving it back at any point of time even after the application that had created the data ended.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

Hibernate Advantages:

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides Simple querying of data.

Supported Databases:

Hibernate supports almost all the major RDBMS. Following is list of few of the database engines supported by Hibernate.

- HSQL Database Engine
- DB2/NT
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

Supported Technologies:

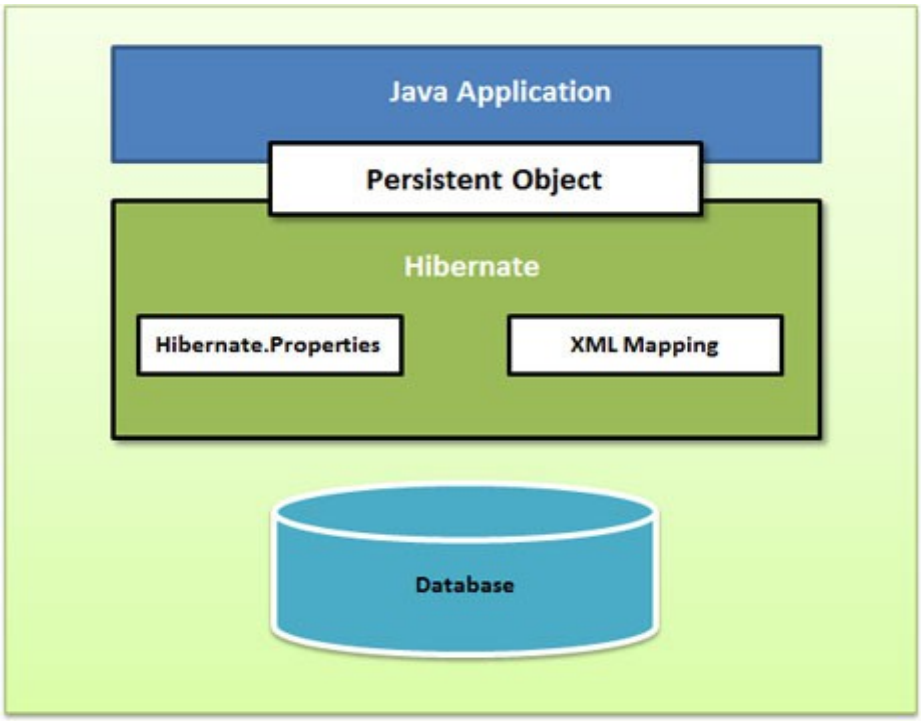
Hibernate supports a variety of other technologies, including the following:

- XDoclet Spring
- J2EE
- Eclipse plug-ins
- Maven

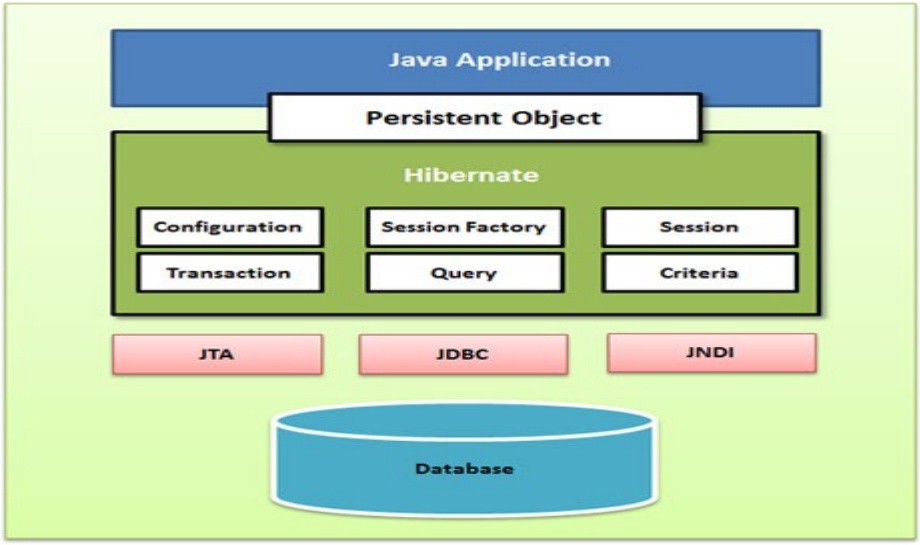
Hibernate Architecture

The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.



Following is a detailed view of the Hibernate Application Architecture with few important core classes.



Configuration Object:

SessionFactory Object:

Session Object:

Transaction Object:

Query Object:

Criteria Object:

Hibernate Environment Setup

Downloading Hibernate:

It is assumed that you already have latest version of Java is installed on your machine. Following are the simple steps to download and install Hibernate on your machine.

- Make a choice whether you want to install Hibernate on Windows, or Unix and then proceed to the next step to download .zip file for windows and .tz file for Unix.
- Download the latest version of Hibernate from http://www.hibernate.org/downloads.

Hibernate Configuration

Hibernate configuration is managed by an instance of `org.hibernate.cfg.Configuration`. An instance of org.hibernate.cfg.Configuration represents an entire set of mappings of an application's Java types to an SQL database. The

org.hibernate.cfg.Configuration is used to build an immutable `org.hibernate.SessionFactory`. The mappings are compiled from various XML mapping files or from Java 5 Annotations.

Hibernate provides following types of configurations

1. *hibernate.cfg.xml* – A standard XML file which contains hibernate configuration and which resides in root of application's CLASSPATH
2. *hibernate.properties* – A Java compliant property file which holds key value pair for different hibernate configuration strings.
3. *Programmatic configuration* – This is the manual approach. The configuration can be defined in Java class.

Hibernate Properties:

Following is the list of important properties you would require to configure for a databases in a standalone situation:

S.N.	Properties and Description
1	**hibernate.dialect** This property makes Hibernate generate the appropriate SQL for the chosen database.
2	**hibernate.connection.driver_class** The JDBC driver class.
3	**hibernate.connection.url** The JDBC URL to the database instance.
4	**hibernate.connection.username** The database username.
5	**hibernate.connection.password** The database password.
6	**hibernate.connection.pool_size** Limits the number of connections waiting in the Hibernate database connection pool.
7	**hibernate.connection.autocommit** Allows autocommit mode to be used for the JDBC connection.

Hibernate Sessions

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. The main function of the Session is to

offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of the following three states at a given point in time:

1. **transient:** A new instance of a a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
2. **persistent:** You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
3. **detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

Hibernate Persistent Class

The entire concept of Hibernate is to take the values from Java class attributes and persist them to a database table. A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields.

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model. There are following main rules of persistent classes, however, none of these rules are hard requirements.

- All Java classes that will be persisted need a default constructor.
- All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.
- All attributes that will be persisted should be declared private and have **getXXX** and **setXXX** methods defined in the JavaBean style.
- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.
- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

Hibernate Mapping Files

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables.

Though many Hibernate users choose to write the XML by hand, a number of tools exist to generate the mapping document. These include **XDoclet, Middlegen** and **AndroMDA** for advanced Hibernate users.

Hibernate Mapping Types

When you prepare a Hibernate mapping document, we have seen that you map Java data types into RDBMS data types. The **types** declared and used in the mapping files are not Java data types; they are not SQL database types either. These types are called Hibernate mapping types, which can translate from Java to SQL data types and vice versa.

Primitive types:

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) ('Y' or 'N')
true/false	boolean or java.lang.Boolean	CHAR(1) ('T' or 'F')

Date and time types:

Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

Binary and large object types:

Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or

		BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB

JDK-related types:

Mapping type	Java type	ANSI SQL Type
class	java.lang.Class	VARCHAR
locale	java.util.Locale	VARCHAR
timezone	java.util.TimeZone	VARCHAR
currency	java.util.Currency	VARCHAR

Hibernate O/R Mappings

There are three most important mapping topics, These are the mapping of collections, the mapping of associations between entity classes and Component Mappings:

Collections Mappings:

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java. Hibernate can persist instances of **java.util.Map, java.util.Set, java.util.SortedMap, java.util.SortedSet, java.util.List**, and any **array** of persistent entities or values.

Association Mappings:

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the cardinality of the relationship between the objects can be expressed. An association mapping can be unidirectional as well as bidirectional.

Examples on :

1)one-one
2)one-many
3)many-many
4)many-one

Component Mappings:

It is very much possible that an Entity class can have a reference to another class as a member variable. If the referred class does not have it's own life cycle and completely depends on the life cycle of the owning entity class, then the referred class hence therefore is called as the Component class.

The mapping of Collection of Components is also possible in a similar way just as the mapping of regular Collections with minor configuration differences.

Hibernate Annotations

Hibernate annotations is the newest way to define mappings without a use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.

Hibernate Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

If you going to make your application portable to other EJB 3 compliant ORM applications, you must use annotations to represent the mapping information but still if you want greater flexibility then you should go with XML-based mappings.

Environment Setup for Hibernate Annotation

Hibernate Query Language

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries which in turns perform action on database.

Although you can use SQL statements directly with Hibernate using Native SQL but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.

Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table

and column names are case sensitive in HQL.

Hibernate Criteria Queries

Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables. One of the methods is Criteria API which allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.

The Hibernate **Session** interface provides **createCriteria()** method which can be used to create a **Criteria** object that returns instances of the persistence object's class when your application executes a criteria query.

Pagination using Criteria:

There are two methods of the Criteria interface for pagination.

S.N.	Method & Description
1	**public Criteria setFirstResult(int firstResult)** This method takes an integer that represents the first row in your result set, starting with row 0.
2	**public Criteria setMaxResults(int maxResults)** This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

Projections & Aggregations:

The Criteria API provides the **org.hibernate.criterion.Projections** class which can be used to get average, maximum or minimum of the property values. The Projections class is similar to the Restrictions class in that it provides several static factory methods for obtaining **Projection** instances.

Hibernate Native SQL

If you want to utilize database-specific features such as query hints or the CONNECT keyword in Oracle. Hibernate 3.x allows you to specify handwritten SQL, including stored procedures, for all create, update, delete, and load operations.

Your application will create a native SQL query from the session with the **createSQLQuery()** method on the Session interface.:

```
public SQLQuery createSQLQuery(String sqlString) throws HibernateException
```

After you pass a string containing the SQL query to the createSQLQuery() method, you can associate the SQL result with either an existing Hibernate entity, a join, or a scalar result using addEntity(), addJoin(), and addScalar() methods respectively.

Scalar queries:

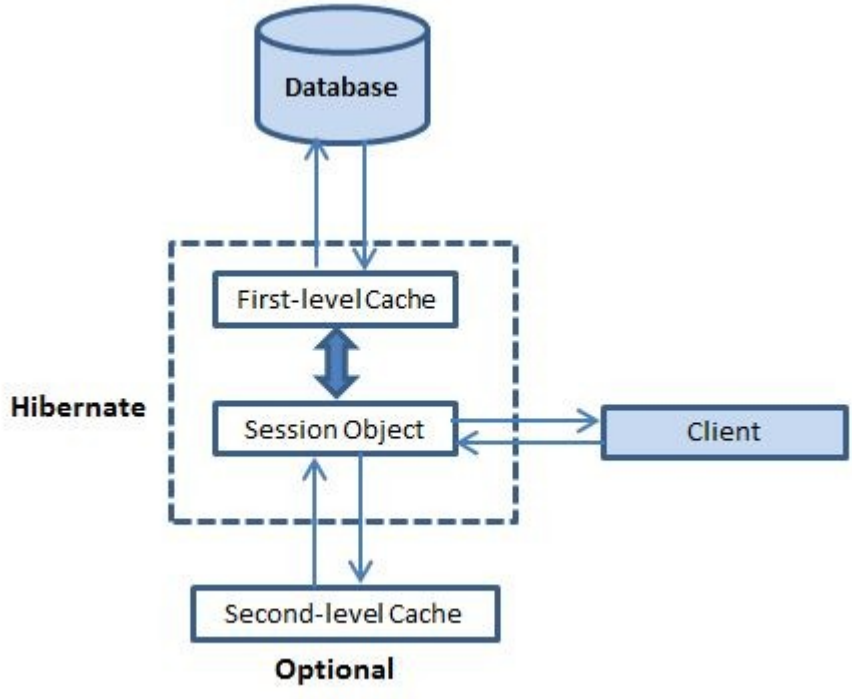
Entity queries:

Named SQL queries:

Hibernate Caching

Caching is all about application performance optimization and it sits between your application and the database to avoid the number of database hits as many as possible to give a better performance for performance critical applications.

Caching is important to Hibernate as well which utilizes a multilevel caching schemes as explained below:



Hibernate Interceptors

Once the object has been changed, it must be saved back to the database. This process continues until the next time the object is needed, and it will be loaded from the persistent store.

Thus an object passes through different stages in its life cycle and **Interceptor Interface** provides methods which can be called at different stages to perform some required tasks. These methods are callbacks from the session to the application, allowing the application to inspect and/or manipulate properties of a persistent object before it is saved, updated, deleted or loaded. Following is the list of all the methods available within the Interceptor interface:

How to use Interceptors?

To build an interceptor you can either implement **Interceptor** class directly or extend **EmptyInterceptor** class. Following will be the simple steps to use Hibernate Interceptor

functionality.