Hibernate Framework

Hibernate is one of the most widely used ORM frameworks in the industry. It provides all the benefits of an ORM solution and implements the specification of JPA for data persistence.

Advantages of Hibernate Framework

- **Open Source and Lightweight**
  Hibernate framework is open source under the LGPL license and lightweight.
- **Fast Performance**
  The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.
- **Database Independent Query**
  HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So, you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.
- **Automatic Table Creation**
  Hibernate framework provides the facility to create the tables of the database automatically. So, there is no need to create tables in the database manually.
- **Simplifies Complex Join**
  Fetching data from multiple tables is easy in hibernate framework.
- **Provides Query Statistics and Database Status**
  Hibernate supports Query cache and provide statistics about query and database status.

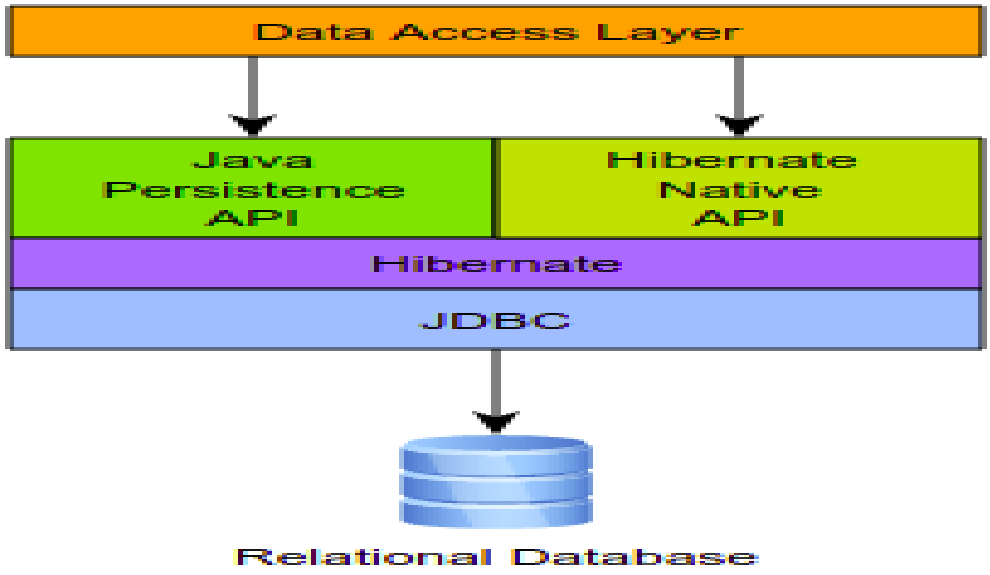
The Hibernate Components.

The Hibernate project suite includes the following: -

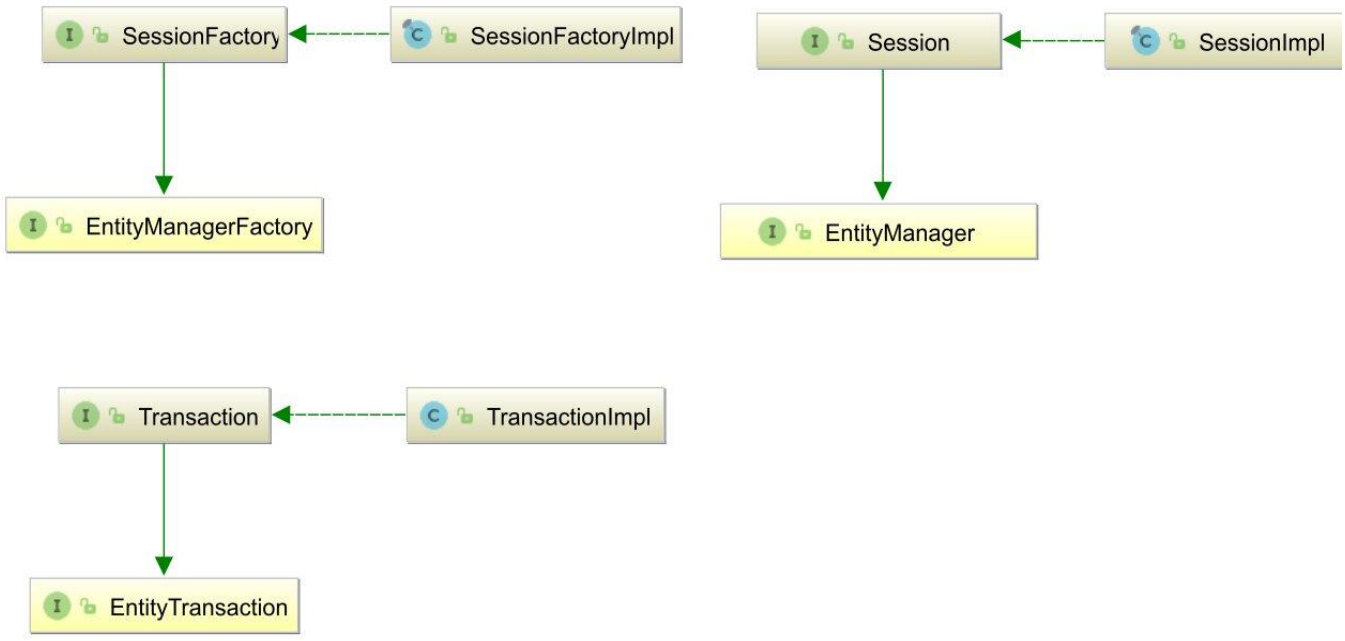
- **Hibernate ORM —**Hibernate ORM consists of a core, a base service for persistence with SQL databases, and a native proprietary API. Hibernate ORM is the foundation for several of the other projects and is the oldest Hibernate project. You can use Hibernate ORM on its own, independent of any framework or any particular runtime environment with all JDK s. It works in every Java EE/J2EE application server, in Swing applications, in a simple servlet container, and so on. As long as you can configure a data source for Hibernate, it works.

- **Hibernate EntityManager—**This is Hibernate's implementation of the standard Java Persistence API s, an optional module you can stack on top of Hibernate ORM. You can fall back to Hibernate when a plain Hibernate interface or even a JDBC Connection is needed. Hibernate's native features are a superset of the JPA persistence features in every respect.
- **Hibernate Validator—**Hibernate provides the reference implementation of the Bean Validation (JSR 303) specification. Independent of other Hibernate projects, it provides declarative validation for your domain model (or any other) classes.
- **Hibernate Envers—**Envers is dedicated to audit logging and keeping multiple versions of data in your SQL database. This helps you add data history and audit trails to your application, similar to version control systems you might already be familiar with such as Subversion and Git.
- **Hibernate Search—**Hibernate Search keeps an index of your domain model data up to date in an Apache Lucene database. It lets you query this database with a powerful and naturally integrated API. Many projects use Hibernate Search in addition to Hibernate ORM, adding full-text search capabilities. If you have a free text search form in your application's user interface, and you want happy users, work with Hibernate Search. Hibernate Search isn't covered in this book; you can find more information in Hibernate Search in Action by Emmanuel Bernard (Bernard, 2008).
- **Hibernate OGM —**The most recent Hibernate project is the object/grid mapper. It provides JPA support for NoSQL solutions, reusing the Hibernate core engine but persisting mapped entities into a key/value-, document-, or graph-oriented data store. Hibernate OGM isn't covered in this book.

Hibernate Architecture



As a JPA provider, Hibernate implements the Java Persistence API specifications and the association between JPA interfaces and Hibernate specific implementations can be visualized in the following diagram:



What is JPA?

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

The **javax.persistence** package contains the JPA classes and interfaces.

Elements of Hibernate Architecture

- **SessionFactory (org.hibernate.SessionFactory)**

  A thread-safe (and immutable) representation of the mapping of the application domain model to a database. Acts as a factory for org.hibernate.Session instances. The EntityManagerFactory is the JPA equivalent of a SessionFactory and basically, those two converge into the same SessionFactory implementation.

  A SessionFactory is very expensive to create, so, for any given database, the application should have only one associated SessionFactory.

  The SessionFactory maintains services that Hibernate uses across all Session(s) such as second level caches, connection pools, transaction system integrations, etc.

- **Session (org.hibernate.Session)**

  A single-threaded, short-lived object conceptually modeling a "Unit of Work" (PoEAA). In JPA nomenclature, the Session is represented by an EntityManager.

  Behind the scenes, the Hibernate Session wraps a JDBC java.sql.Connection and acts as a factory for org.hibernate.Transaction instances. It maintains a generally "repeatable read" persistence context (first level cache) of the application domain model.

- **Transaction (org.hibernate.Transaction)**

  A single-threaded, short-lived object used by the application to demarcate individual physical transaction boundaries. EntityTransaction is the JPA equivalent and both act as an abstraction API to isolate the application from the underlying transaction system in use (JDBC or JTA).

- **ConnectionProvider**

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

- **TransactionFactory**

  It is a factory of Transaction, it is optional.

The Hibernate Modules/Artifacts

Hibernate's functionality is split into a number of modules/artifacts meant to isolate dependencies (modularity).

1. **hibernate-core: -** The main (core) Hibernate module. Defines its ORM features and APIs as well as the various integration SPIs.
2. **hibernate-envers:-** Hibernate's historical entity versioning feature
3. **hibernate-spatial: -** Hibernate's Spatial/GIS data-type support
4. **hibernate-osgi:-** Hibernate support for running in OSGi containers.
5. **hibernate-agroal:-** Integrates the Agroal connection pooling library into Hibernate
6. **hibernate-c3p0: -** Integrates the C3P0 connection pooling library into Hibernate
7. **hibernate-hikaricp:-** Integrates the HikariCP connection pooling library into Hibernate
8. **hibernate-vibur:-** Integrates the Vibur DBCP connection pooling library into Hibernate
9. **hibernate-proxool:-** Integrates the Proxool connection pooling library into Hibernate
10. **hibernate-jcache:-** Integrates the JCache caching specification into Hibernate, enabling any compliant implementation to become a second-level cache provider.
11. **hibernate-ehcache:-** Integrates the Ehcache caching library into Hibernate as a second-level cache provider.

Hibernate Uses at Development

We can use Hibernate as following manner: -

1. Using Native Hibernate APIs and hbm.xml Mapping
2. Using Native Hibernate APIs and Annotation Mappings
3. Using the Java Persistence API (JPA)
4. Using Envers

Note: - Hibernate Envers is a framework for auditing. Though Hibernate is an ORM technology, auditing tasks based on Hibernate entities means changes on the entity is audited and saved on the database. Auditing of all mappings is defined by the JPA specification. Revision of each entity log is saved by Hibernate Envers. Hibernate Envers gives the way to read historical data log.

Configuration

We can configure Hibernate in three ways:

1. **Programmatic configuration:** Use the API to load the **hbm** file, load the database driver, and specify the database connection details.
2. **XML configuration:** Specify the database connection details in an XML file that's loaded along with the **hbm** file. The default file name is **hibernate.cfg.xml.** You can use another name by specifying the name explicitly.
3. **Properties file configuration:** Similar to the XML configuration, but uses a **.properties** file. The default name is **hibernate.properties**.