Hibernate

It was started in 2001 by **Gavin King** as an alternative to EJB2 style entity bean.

It is an ORM (Object Relational Mapping) Framework.

## Pros and Cons of JDBC

|  |  |
| --- | --- |
| **Pros of JDBC** | **Cons of JDBC** |
| * **Clean and simple SQL processing** * **Good performance with large data** * **Very good for small applications** * **Simple syntax so easy to learn** | * Complex if it is used in large projects * Large programming overhead * No encapsulation * Hard to implement MVC concept * Query is DBMS specific |

**Difference between JDBC and HIBERNATE?**

|  |  |
| --- | --- |
| **JDBC** | **HIBERNATE** |
| **JDBC is a persistence technology (API)** | Hibernate is a persistence Framework. |
| **In JDBC, developer is responsible to taking and closing the connection and also writes the SQL statement.** | In Hibernate **HRS**(Hibernate Runtime System) is responsible for taking the connections,creating the statement and releasing the connections. |
| **JDBC does not have any ORM tool.** | Hibernate have ORM (Object relational Mapping) tools which simplifies all the works (like caching mechanism,   supports Object Oriented Query Language etc.  ). That is why Hibernate is the King of among all persistence Frameworks. |
| **JDBC does not support any type of mapping** | Hibernate Support various mapping styles (like simple mapping, collection mapping, inheritance mapping, association mapping and other mappings), optimization facility (like dirty checking, lazy associating fetching etc.), locking mechanism (optimistic and pessimistic lock) and also provide facility to express joins. |

### Hibernate Framework

Hibernate framework simplifies the development of java application to interact with the database. Hibernate is an open source, lightweight, [ORM (Object Relational Mapping)](http://en.wikipedia.org/wiki/Object-relational_mapping) tool.

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

### Advantages of Hibernate Framework

There are many advantages of Hibernate Framework. They are as follows:

**1) Open source and Lightweight:** Hibernate framework is open source under the LGPL license and lightweight.

**2) 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.

**3) 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.

**4) 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.

**5) Simplifies complex join:** To fetch data from multiple tables is easy in hibernate framework.

**6) Provides query statistics and database status:** Hibernate supports Query cache and provide statistics about query and database status.

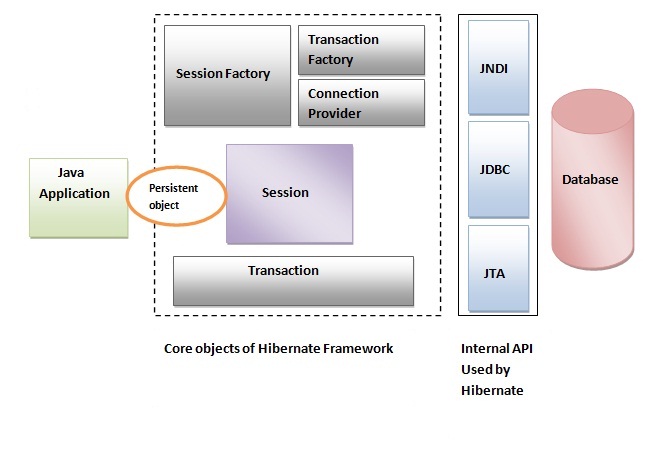
# Hibernate Architecture

The Hibernate architecture includes many objects persistent object, session factory, transaction factory, connection factory, session, transaction etc.

There are **4 layers** in hibernate architecture java application layer, hibernate framework layer, backend api layer and database layer.Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

### Elements of Hibernate Architecture

|  |
| --- |
| For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows: |

#### SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The **org.hibernate.SessionFactory** interface provides factory method to get the object of Session.

#### Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The **org.hibernate.Session** interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

#### Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

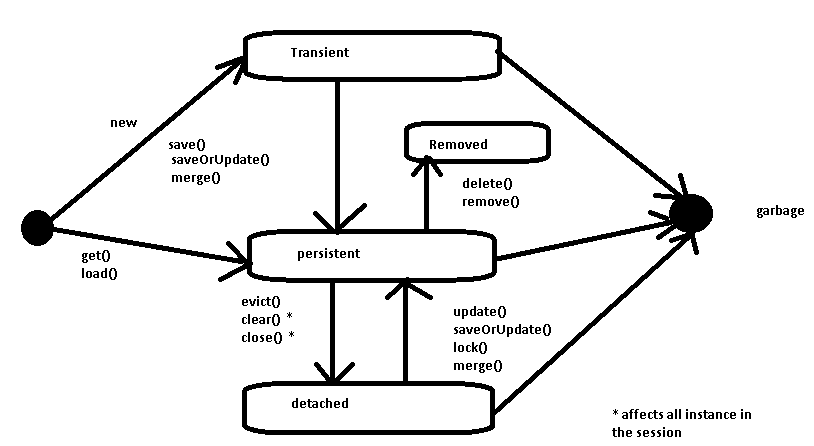
#### ConnectionProvider

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.

#### Hibernate states of an Instance



# Example to create the Hibernate Application in Eclipse IDE

Here, we are going to create a simple example of hibernate application using eclipse IDE. For creating the first hibernate application in Eclipse IDE, we need to follow following steps:

1. Create the java project
2. Add jar files for **hibernate 4.0**
3. **Create the Persistent class (POJO)**
4. **Create the mapping file for Persistent class**
5. **Create the Configuration file**
6. **Create the class that retrieves or stores the persistent object (client application)**
7. Run the application

### 1) Create the java project

|  |
| --- |
| Create the java project by **File Menu** - **New** - **project** - **java project**. Now specify the project name e.g. firsthb then **next**-**finish**. |

### 2) Add jar files for hibernate

|  |
| --- |
| To add the jar files **Right click on your project** - **Build path** - **Add external archives**. Now select all the jar files. |

In this example, we are connecting the application with oracle database. So you must add the ojdbc14.jar file.

### 3) Create the Persistent class (POJO – plain old java object)

To create the persistent class, Right click on src - New - Class - specify the class with package name (e.g. mypackage) - finish.

#### Employee.java

**public** **class** Employee {

**private** **int** id;

**private** String firstName,lastName;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

}

### 4) Create the mapping file for Persistent class

To create the mapping file, Right click on src - new - file - specify the file name (e.g. employee.hbm.xml) - ok. It must be outside the package.

#### employee.hbm.xml

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-mapping PUBLIC

 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

 <hibernate-mapping>

  <**class** name="mypackage.Employee" table="emp1000">

    <id name="id">

     <generator **class**="assigned"></generator>

    </id>

    <property name="firstName"></property>

    <property name="lastName"></property>

  </**class**>

 </hibernate-mapping>

### 5) Create the Configuration file

The configuration file contains all the informations for the database such as connection\_url, driver\_class, username, password etc. The hbm2ddl.auto property is used to create the table in the database automatically. We will have in-depth learning about Dialect class in next topics. To create the configuration file, right click on src - new - file. Now specify the configuration file name e.g. hibernate.cfg.xml.

#### hibernate.cfg.xml

<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <property name="hbm2ddl.auto">update</property>

        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>

        <property name="connection.username">hr</property>

        <property name="connection.password">hr</property>

        <property name="connection.driver\_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping resource="employee.hbm.xml"/>

    </session-factory>

</hibernate-configuration>

### 6) Create the class that retrieves or stores the persistent object (client program)

In this class, we are simply storing the employee object to the database.

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**public** **class** StoreData {

**public** **static** **void** main(String[] args) {

    //creating configuration object

    Configuration cfg=**new** Configuration();

    cfg.configure("hibernate.cfg.xml");//populates the data of the configuration file

    //creating seession factory object

    SessionFactory factory=cfg.buildSessionFactory();

    //creating session object

    Session session=factory.openSession();

    //creating transaction object

    Transaction t=session.beginTransaction();

    Employee e1=**new** Employee();

    e1.setId(115);

    e1.setFirstName("sonoo");

    e1.setLastName("jaiswal");

    session.persist(e1);//persisting the object

    t.commit();//transaction is committed

    session.close();

    System.out.println("successfully saved");

}

}

### 7) Run the application

|  |
| --- |
| Before running the application, determine that directory structure is like this. |
| Generator classes in Hibernate  The <generator> subelement of id used to generate the unique identifier for the objects of persistent class. There are many generator classes defined in the Hibernate Framework.  All the generator classes implements the **org.hibernate.id.IdentifierGenerator interface**. The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface. Hibernate framework provides many built-in generator classes:   1. **assigned** 2. **increment** 3. **sequence** 4. hilo 5. native 6. identity 7. seqhilo 8. uuid 9. guid 10. select 11. foreign 12. sequence-identity  1) assigned It is the default generator strategy if there is no <generator> element . In this case, application assigns the id. For example:   1. .... 2. <hibernate-mapping> 3. <**class** ...> 4. <id ...> 5. <generator **class**="assigned"></generator> 6. </id> 7. ..... 8. </**class**> 9. </hibernate-mapping>  2) increment It generates the unique id only if no other process is inserting data into this table. It generates **short**, **int** or **long** type identifier. The first generated identifier is 1 normally and incremented as 1. Syntax:   1. .... 2. <hibernate-mapping> 3. <**class** ...> 4. <id ...> 5. <generator **class**="increment"></generator> 6. </id> 7. ..... 8. </**class**> 9. </hibernate-mapping>  3) sequence It uses the sequence of the database. if there is no sequence defined, it creates a sequence automatically e.g. in case of Oracle database, it creates a sequence named HIBERNATE\_SEQUENCE. In case of Oracle, DB2, SAP DB, Postgre SQL or McKoi, it uses sequence but it uses generator in interbase. Syntax:   1. ..... 2. <id ...> 3. <generator **class**="sequence"></generator> 4. </id> 5. .....   For defining your own sequence, use the param subelement of generator.   1. ..... 2. <id ...> 3. <generator **class**="sequence"> 4. <param name="sequence">your\_sequence\_name</param> 5. </generator> 6. </id> 7. .....  4) hilo It uses high and low algorithm to generate the id of type short, int and long. Syntax:   1. ..... 2. <id ...> 3. <generator **class**="hilo"></generator> 4. </id> 5. .....  5) native It uses identity, sequence or hilo depending on the database vendor. Syntax:   1. ..... 2. <id ...> 3. <generator **class**="native"></generator> 4. </id> 5. .....  6) identity It is used in Sybase, My SQL, MS SQL Server, DB2 and HypersonicSQL to support the id column. The returned id is of type short, int or long. 7) seqhilo It uses high and low algorithm on the specified sequence name. The returned id is of type short, int or long. 8) uuid It uses 128-bit UUID algorithm to generate the id. The returned id is of type String, unique within a network (because IP is used). The UUID is represented in hexadecimal digits, 32 in length. 9) guid  |  | | --- | | It uses GUID generated by database of type string. It works on MS SQL Server and MySQL. |  10) select  |  | | --- | | It uses the primary key returned by the database trigger. |  11) foreign  |  | | --- | | It uses the id of another associated object, mostly used with <one-to-one> association. |  12) sequence-identity  |  | | --- | | It uses a special sequence generation strategy. It is supported in Oracle 10g drivers only. | |

SQL Dialects in Hibernate

For connecting any hibernate application with the database, you must specify the SQL dialects. There are many Dialects classes defined for RDBMS in the org.hibernate.dialect package. They are as follows:

|  |  |
| --- | --- |
| **RDBMS** | **Dialect** |
| **Oracle (any version)** | org.hibernate.dialect.OracleDialect |
| **Oracle9i** | org.hibernate.dialect.Oracle9iDialect |
| **Oracle10g** | org.hibernate.dialect.Oracle10gDialect |
| **MySQL** | org.hibernate.dialect.MySQLDialect |
| **MySQL with InnoDB** | org.hibernate.dialect.MySQLInnoDBDialect |
| **MySQL with MyISAM** | org.hibernate.dialect.MySQLMyISAMDialect |
| **DB2** | org.hibernate.dialect.DB2Dialect |
| **DB2 AS/400** | org.hibernate.dialect.DB2400Dialect |
| **DB2 OS390** | org.hibernate.dialect.DB2390Dialect |
| **Microsoft SQL Server** | org.hibernate.dialect.SQLServerDialect |
| **Sybase** | org.hibernate.dialect.SybaseDialect |
| **Sybase Anywhere** | org.hibernate.dialect.SybaseAnywhereDialect |
| **PostgreSQL** | org.hibernate.dialect.PostgreSQLDialect |
| **SAP DB** | org.hibernate.dialect.SAPDBDialect |
| **Informix** | org.hibernate.dialect.InformixDialect |
| **HypersonicSQL** | org.hibernate.dialect.HSQLDialect |
| **Ingres** | org.hibernate.dialect.IngresDialect |
| **Progress** | org.hibernate.dialect.ProgressDialect |
| **Mckoi SQL** | org.hibernate.dialect.MckoiDialect |
| **Interbase** | org.hibernate.dialect.InterbaseDialect |
| **Pointbase** | org.hibernate.dialect.PointbaseDialect |
| **FrontBase** | org.hibernate.dialect.FrontbaseDialect |
| **Firebird** | org.hibernate.dialect.FirebirdDialect |

Hibernate Inheritance Mapping

We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

1. Table Per Hierarchy
2. Table Per Concrete class
3. Table Per Subclass

#### Table Per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table .

#### Table Per Concrete class

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

#### Table Per Subclass

In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.