# Hibernate Concepts

**NOTE**: since there are lots many API changes from version to version in hibernate framework. SessionFactory creation may be different the version you are using. This document targeting the version 4.3.6

Reference: <http://vladmihalcea.com/tutorials/hibernate/>

Steps to work with Hibernate

1. Create your maven project
2. Write your **hibernate.cfg.xml** file and place it in classpath
3. Write utility class to create SessionFactory object as shown below
4. Finally create Session as described below and work with database.

**Project Creation:**

Create maven based java project and add below dependency

Use below maven dependency to create maven project for quick start

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.6.Final</version>

</dependency>

**Hibernate Configuration File:**

Configuration file looks like below and there can be additional configuration that we will discuss in the next topics

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE hibernate-configuration SYSTEM "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>

<property name=*"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"hibernate.connection.username"*>root</property>

<property name=*"hibernate.connection.password"*>admin</property>

<property name=*"hibernate.hbm2ddl.auto"*>create-drop</property>

<property name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/test</property>

<property name=*"show\_sql"*>true</property>

<property name=*"format\_sql"*>false</property>

<mapping class=*"com.tvajjala.domain.User"*/>

</session-factory>

</hibernate-configuration>

**SessionFactory Creation:**

Create **SessionFactory** as shown below using hibernate 4.3.6 version.

It is a Factory class through which we can get the sessions and peforms database operations.

**Below code looks for xml file in the classpath**

**final** Configuration configuration = **new** Configuration().configure();

**final** StandardServiceRegistryBuilder builder = **new** StandardServiceRegistryBuilder().applySettings(configuration.getProperties());

*sessionFactory* =

configuration.buildSessionFactory(builder.build());

NOTE: SessionFactory creation varies version to version. It seems to be there is no standarddization yet.

**Session Creation**

There are three methods through which we can get the Session object.

1. **getCurrentSession**: It returns the Session object bound to current context. Inorder to work with this method we need to make some configuration. If not it results below exception message.

No CurrentSessionContext configured!

Add below property to your configuration file.

<property name="hibernate.current\_session\_context\_class">thread</property>

This is useful in single- threaded environment and we don’t need to close the session everytime. It is bound to single thread and we can use session quickly.

1. **openSession**: It creates new session on every method call and once the DB operatios are done we have to close this. This is best suitable in web applications and multithreaded environments.

In web applications we create session per web request/ per web session.

1. **openStatelessSession**: StatelessSession does not implement first-level cache and it doesn’t interact with any second-level cache. Since it’s stateless, it doesn’t implement transactional write-behind or automatic dirty checking or do cascading operations to associated entities.

Collections are also ignored by a stateless session. Operations performed via a stateless session bypass Hibernate’s event model and interceptors. It’s more like a normal JDBC connection and doesn’t provide any benefits that come from using hibernate framework.

However stateless session can be a good fit in certain situations, for example where we are loading bulk data into database and we don’t want hibernate session to hold huge data in first-level cache memory.

It has two over loaded methods:

* openStatlessSession: with zero arguments
* openStatelessSession (Connection connection): java.sql.Connection as an argument.

**NOTE**: never class **session.close()** method on the session object which was created using the openCurrentSession method.

**Q) What is flush method will do?**

**Flush will just clears the memory and insert into database but you don’t see data until you commit the transaction.**

This should be called every unit of work to execute database statements

Session.persist() only stores into session cache but doesn’t hit the database. Session.flush() will fire the database query but since transaction not commited it is not inserted into database as well.

When we do transaction.commit() it will commit into datbase .

🡺 Commit= flush + commit

NOTE: Session object container setFlushMode () method if you set this value to MANUAL. You have to call manually to save into databse otherwise it won’t insert the record into databse.

**Q) How can you work on direct JDBC connection object if you have hibernate session available?**

There are two important methods on session object that allows users to work with JDBC related work.

1. Method with void return type session.doWork (Work work);
2. Method with ‘T’ return type session.doReturningWork (ReturningWork<T> returningWork);

**Q) Define Transaction?**

Transaction is sequence of operations that are succeeded only if all the operations are successful.

**Q) What are the Transaction Characteristics?**

Transaction has four characters called ACID

* **Atomicity**: It is individual operations turns into unit of work that are success or fail as a group. With this system left as consistent state after group of operatioins executed.
* **Consistency:** Transaction should take system one consistent state to other consistent state.if one operation triggers secondary actions (TRIGGERS, CASCADE) those must also be succeed otherwise transacton fails. If the system has multiple nodes, all the changes should propagate to all the nodes. If system propagting nodes asynchronosly then we are breaking Consistency. Then system called eventually consistent. Ex: Hazelcast has option to async data propagation setting
* **Isolation:** In concurrent programming world the result of an incomplete transaction should not show data to the other external world. SQL Standard defines FOUR Isolatioin levels.

**READ\_COMMITTED,**

**READ\_UNCOMMITTED,**

**REPEATABLE\_READ,**

**SERIALIZATION**

* **Durability:** Successful transaction must persist the changes permenantly. If the system crashes those successful transaction data should be replayed.

**Q) What are dirty reads, repeatable reads and phonton reads?**

* **Dirty read:** reading un-commtted data from the other transaction.
* **Non-Repeatable reads**: during the course of transaction, if we read the same row multiple times it returns different data each time. (Same row returns different data)
* **Phanton-reads**: During the course of transaction, if we executed two identical queries, rows returned from the first query different from the second query. (No.of records different each time)

**Q) What is the isolation level of the MYSQL, ORACLE, and SQL Sever Databases?**

* MYSQL it is **REPEATABLE\_READ**
* ORACLE it is READ\_COMMITTED
* SQL SERVER it is READ\_COMMITTED

**Q) How do you change Isolation level in Hibernate?**

Using property hibernate.connection.isolation

We can change the isolation level of the transaction

**Q) What are the different transactions hibernate provides?**

Hibernate provides two types of transactions

1. Resource local
2. JTA Transactions

**JTA Transactions**: JTA transactions require an [XAConnection](https://docs.oracle.com/javase/7/docs/api/javax/sql/XAConnection.html) and it’s the [JTA transaction manager](https://github.com/bitronix/btm/wiki/Transaction-manager-configuration) responsibility to provide XA compliant connections.

**Q) What are the different Connection providers hibernate gives?**

Hibernate offers multiple connection provider options:

* **Driver Manager Connection Provider** (doesn’t pool connections and therefore it’s only meant for simple testing scenarios)

The Driver Manager Connection Provider offers a rudimentary DataSource wrapper for the configured database driver. You should only use it for test scenarios since it doesn’t offer a professional connection pooling mechanism.

**Hibernate Overrides Isolation level**

* **C3P0 Connection Provider** (delegating connection acquiring calls to an internal C3P0 connection pooling DataSource)

We need to provide the driver configuration settings and Hibernate instantiate the C3P0 connection pool on our behalf.

**Hibernate Overrides Isolation level**

* **DataSource Connection Provider** (delegating connection-acquiring calls to an external DataSource.

Hibernate doesn’t force you to use a specific connection provider mechanism. You can simply supply a DataSource and Hibernate will use it whenever a new Connection is being requested.

**IN THIS CASE ISOLATION LEVEL is NOT OVERRIDEN BY THE HIBERNATE**

**Q) Different ways of primary generations possible in hibernate?**

**Sequence Generation:** database sequences are very convenient because they are both flexible and efficient for most use cases

**Q) What is use of javaassist library that we see more often in the any project like spring or hibernate related?**

|  |  |
| --- | --- |
|  | A common application is to generate proxy classes at runtime, i.e. to create a subclass at runtime that intercepts all method invocations. Examples:   1. Hibernate uses Proxies to intercept method invocations on entities to implement lazy loading, i.e. fetching the object from the database when it is first accessed. 2. The Spring Framework [uses Proxies to implement its AOP support](http://static.springsource.org/spring/docs/3.1.0.M2/spring-framework-reference/html/aop.html#aop-understanding-aop-proxies), which among other things powers its support for declarative transactions. It also uses proxies to enforce proper scoping. 3. EJB uses proxies to implement container-managed transactions, authorization checking, and to apply user-defined interceptors. 4. CDI implementations must also proxy the managed beans to ensure proper scoping. I suspect they use a byte code-engineering library, too. |

* **Hibernate currently uses Javassist (in the past it used GCLib, but it was deprecated) to instrument classes at runtime.**
* **Hibernate does creates subclasses at runtime that work has proxy for you persistent entities.**

**Q) Compare diffent code gen libraries**

[**http://zeroturnaround.com/rebellabs/testing-the-performance-of-4-java-runtime-code-generators-cglib-javassist-jdk-proxy-byte-buddy/**](http://zeroturnaround.com/rebellabs/testing-the-performance-of-4-java-runtime-code-generators-cglib-javassist-jdk-proxy-byte-buddy/)

**Q) Which java API involved in JDK Dynamic Proxying?**

Java.lang.reflect.InvocationHandler interface generates new proxy class which has single method called invoke () using that Proxy.newProxyInstance (classLoader, interfaces [], proxyImpl) will delegates the calls to target class.

**Q) What are differences between persist and save methods in hibernate?**

* First difference is save method returnsn PK as a Serializable value and persist method returns void.
* Second major difference is save method creates new row for the detached object where as persist method throws org.hibernate.PersistentObjectException: detached entity passed to persist
* ~~Third difference is save method works without transaction whereas persist works within the boundary of the transaction.~~

~~Save method works without transaction but it dependent (mapped) objects are not saved unless you~~ **~~flush~~** ~~that session.~~

Q) What is the difference between update and merge methods?

When trying to update object with the same id within the session it throws org.hibernate.NonUniqueObjectException: A different object with the same identifier value was already associated with the session. Where as merge method will merges successfully.

**Q) What is the difference Value Object and Entity Object? How to use value object hibernate?**

**Entity Object:** Has its own database identity (Student table)  
**Value Object:** Does n’t not have its own database identity

For readability we can move some of the properties new Java class and embedd into Entity Bean. For this we can use @Embedded and @Embeddable annotations we can use. @AttributesOverrides used to change the database column names

**Q) What is the difference between @Entity (name=””) and @Table (name=””)?**

**@Table (name=””)** will change the name of the table @Entity (name=””) will be used to in the HQL queries. If you want to change DB table name only then it would be good to use **@Table (name=””).**

**Q) What is @Basic annotation and when can be useful?**

Any java primitive types it is default annotation used on the properties.

**Q) Which annotation used to ignore as a DB column?**

There is annotation called @Transient which will be ignored form DB

**Q) How do you customize table Date column value (time, date, timestamp)?**

**Q) How does @Lob works?**

@Lob annotation that is used on top of String type can be chosen as @Lob whereas if we use char [] array that can be chosen as @Clob

**Q) What is the default FetchType?**

Default fetch type javax.persistence.FetchType.LAZY

**Q) What are surrogate keys?**

That serve any business data they just incremental values

**Q) When do we need to use mappedBy attribute?**

This attribute can be used to tell that other entity is maintaining the joining column information. This entity creates standalone table that doesn’t have any relations with other tables. Using this we can reduce the redundacy column in the **@OneToMany**, **@OneToOne** relations. And reduces redundancy table in the **@ManyToMany** relation.

Q) What are positional parameters and named parameters in HQL?

Without parameter binding, you have to concatenate the parameter String like this (bad code):

String hql = "from Stock s where s.stockCode = '" + stockCode + "'";

List result = session.createQuery (hql).list();

Pass an unchecked value from user input to the database will raise security concern, because it can easy get hack by SQL injection. You have to avoid the above bad code and using parameter binding instead.

There are two ways to parameter binding:

1. **Named parameters:** This is the most common and user-friendly way. It use colon followed by a parameter name (: example) to define a named parameter.
2. **Positional parameters:** It’s use question mark (?) to define a named parameter, and you have to set your parameter according to the position sequence.

**Q) What are differences between Criteria and HQL?**

Using HQL we can do select as well as non-select operations (like dalete, update etc) Where as **Criteria** we can only do select operations.

HQL used to execute static queries where as Criteria used to execute Dynamic Query.

Criteria supports pagination but HQL doesn’t support pagination

Criteria’s are show in execution but HQL are faster

Criteria safe from SQL Injection as the query created dynamically where are HQL (both named, positional) can be hacked by the SQL Injection.

**Q) What are Restrictions and Projections?**

Projections are used in the below two scenarios

* We can **load** partial object from the database
* We can find the Result of Aggregate functions

**Q) How do you configure second level cache in hibernate?**

First level cache bounded to session. But second level cache shared across different sessions.

SECOND LEVEL CACHE

Second level cache is a local store of entity data managed by the perisitence provider to improve application performance.

A second-level cache helps improve performance by avoiding expensive database calls, keeping the entity data local to the application.

A second-level cache is typically tranparent to the application as it is managed by the persistence provider and underlies the persistence context of an application.

Persistence providers are not required to support a second-level cache.

Second-level cache for a persistence unit may be configured to one of several second-level cache modes.

Following mode settings are defined by the Java persistence API.

|  |  |
| --- | --- |
| ALL | All entity data is stored in the second-level cache for this persistence unit |
| NONE | No data is cached in the UI. PU must not cache any data |
| ENABLE\_SELECTIVE | Enable caching for entities that have been explicity set with the @Cacheable annotation |
| DISABLE\_SELECTIVE | Enable caching for all enetities those that have been explicitly set with the @Cacheable (false) annotation |
| UNSPECIFIED | Caching behaviour for the PU is undefined. PU Provider’s default caching behaviour will be used. |

**Second level cache can be set in two ways**

1. Adding **shared-cache-mode** property to **DISABLE\_SELECTIVE**
2. Setting **javax.persistence.sharedCache.mode** property to one of the shared cache mode settings

Javax.persistence.sharedCache.mode = ENABLE\_SELECTIVE

**Setting the Cache Retrieval and Store modes**

If the second-level cache has been enabled for a persistence unit by setting the shared cache mode, the behaviour of the second-level cache can be further modified by setting the below property

* Javax.persistence.cache.retrieveMode
* Javax.persistence.cache.storeMode

**Cache Retrieval Mode:** The cache retrieval mode, set by the javax.persistence.retrieveMode property, controls how data is read form the cache for calls to the **EntityManger.find** method and from queries.

The retrieveMode property can be set to one of the constants defined by the javax.persistence.CacheRetrieveMode-enumerated type either USE (default) or BYPASS.

* **USE**: data retrieved from the cache if available, otherwise from DB.
* **BYPASS**: second-level cache is bypassed and a call to the DB is made to retrieve the data.

Cache Store Mode: it controls how data is stored in the cache using below property.

Javax.persistence.storeMode

* **USE**: the cache data is created or updated when data is read from or committed to the database. If data is already in the cache, setting the store mode to USE will not force a refresh when data is read form the DB
* **BYPASS**: data read form or committed to the DB is not inserted or updated in the cache. That is, the cache unchanged.
* **REFRESH**: the cache data is created or updated when data is read form or committed to the DB, and a refresh is forced on data in the cache upon DB reads.