GENERAL QUESTIONS

## Q1. What is Hibernate?

Hibernate is one of the most popular Java frameworks that simplify the development of Java application to interact with the database. It is an Object-relational mapping (ORM) tool. Hibernate also provides a reference implementation of Java API.

It is referred as a framework which comes with an abstraction layer and also handles the implementations internally. The implementations include tasks like writing a query for CRUD operations or establishing a connection with the databases, etc.

Hibernate develops persistence logic, which stores and processes the data for longer use. It is a lightweight tool and most importantly open-sourced which gives it an edge over other frameworks.

## Q2. What are the major advantages of Hibernate Framework?

• It is open-sourced and lightweight.

• Performance of Hibernate is very fast.

• Helps in generating database independant queries.

• Provides facilities to automatically create a table.

• It provides query statistics and database status.

## Q3. What are the advantages of using Hibernate over JDBC?

Major advantages of using Hibernate over JDBC are:

1. Hibernate eliminates a lot of boiler-plate code that comes with JDBC API, the code looks cleaner and readable.

2. This Java framework supports inheritance, associations, and collections. These features are actually not present in JDBC.

3. HQL (Hibernate Query Language) is more object-oriented and close to Java. But for JDBC, you need to write native SQL queries.

4. Hibernate implicitly provides transaction management whereas, in JDBC API, you need to write code for transaction management using commit and rollback.

5. JDBC throws SQLException that is a checked exception, so you have to write a lot of try-catch block code. Hibernate wraps JDBC exceptions and throw JDBCException or HibernateException which are the unchecked exceptions, so you don’t have to write code to handle it has built-in transaction management which helps in removing the usage of try-catch blocks.

## Q4. What is an ORM tool?

It is basically a technique that maps the object that is stored in the database. An ORM tool helps in simplifying data creation, manipulation, and access. It internally uses the Java API to interact with the databases.

## Q5. Why use Hibernate Framework?

Hibernate overcomes the shortcomings of other technologies like JDBC.

• It overcomes the database dependency faced in the JDBC.

• Changing of the databases cost a lot working on JDBC, hibernate overcomes this problem with flying colors.

• Code portability is not an option while working on JDBC. This is easily handled by Hibernate.

• Hibernate strengthens the object level relationship.

• It overcomes the exception-handling part which is mandatory while working on JDBC.

• It reduces the length of code with increased readability by overcoming the boilerplate problem.

## Q6. What are the different functionalities supported by Hibernate?

• Hibernate is an ORM tool.

• Hibernate uses Hibernate Query Language(HQL) which makes it database-independent.

• It supports auto DDL operations.

• This Java framework also has an Auto Primary Key Generation support.

• Supports cache memory.

• Exception handling is not mandatory in the case of Hibernate.

## Q7. What are the technologies that are supported by Hibernate?

Hibernate supports a variety of technologies, like:

• XDoclet Spring

• Maven

• Eclipse Plug-ins

• J2EE

## Q8. What is HQL?

HQL is the acronym of Hibernate Query Language. It is an Object-Oriented Query Language and is independent of the database.

## Q9. How to achieve mapping in Hibernate?

Association mappings are one of the key features of Hibernate. It supports the same associations as the relational database model. They are:

• One-to-One associations

• Many-to-One associations

• Many-to-Many associations

You can map each of them as a uni- or bidirectional association.

## Q10. Name some of the important interfaces of Hibernate framework?

Hibernate interfaces are:

• SessionFactory (org.hibernate.SessionFactory)

• Session (org.hibernate.Session)

• Transaction (org.hibernate.Transaction)

## Q11. What is One-to-One association in Hibernate?

In this type of mapping, you only need to model the system for the entity for which you want to navigate the relationship in your query or domain model. You need an entity attribute that represents the association, so annotate it with an @OneToOne annotation.

## Q12. What is One-to-Many association in Hibernate?

In this type of association, one object can be associated with multiple/different objects. Talking about the mapping, the One-to-Many mapping is implemented using a Set Java collection that does not have any redundant element. This One-to-Many element of the set indicates the relation of one object to multiple objects.

## Q13. What is Many-to-Many association in Hibernate?

Many-to-Many mapping requires an entity attribute and a @ManyToMany annotation. It can either be unidirectional and bidirectional. In Unidirectional, the attributes model the association and you can use it to navigate it in your domain model or JPQL queries. The annotation tells Hibernate to map a Many-to-Many association. The bidirectional relationship, mapping allows you to navigate the association in both directions.

## Q14. How to integrate Hibernate and Spring?

Spring is also one of the most commonly used Java frameworks in the market today. Spring is a JavaEE Framework and Hibernate is the most popular ORM framework. This is why Spring Hibernate combination is used in a lot of enterprise applications.

Following are the steps you should follow to integrate Spring and Hibernate.

1. Add Hibernate-entity manager, Hibernate-core and Spring-ORM dependencies.

2. Create Model classes and corresponding DAO implementations for database operations. The DAO classes will use SessionFactory that will be injected by the Spring Bean configuration.

3. Note that you don’t need to use Hibernate Transaction Management, as you can leave it to the Spring declarative transaction management using @Transactional annotation.

## Q15. What do you mean by Hibernate Configuration File?

Hibernate Configuration File mainly contains database-specific configurations and are used to initialize SessionFactory. Some important parts of the Hibernate Configuration File are Dialect information, so that hibernate knows the database type and mapping file or class details.

1) Hibernate Interview Questions for intermediate

## Q16. Mention some important annotations used for Hibernate mapping?

Hibernate supports JPA annotations. Some of the major annotations are:

1. javax.persistence.Entity: This is used with model classes to specify they are entity beans.

2. javax.persistence.Table: It is used with entity beans to define the corresponding table name in the database.

3. javax.persistence.Access: Used to define the access type, field or property. The default value is field and if you want Hibernate to use the getter/setter methods then you need to set it to a property.

4. javax.persistence.Id: Defines the primary key in the entity bean.

5. javax.persistence.EmbeddedId: It defines a composite primary key in the entity bean.

6. javax.persistence.Column: Helps in defining the column name in the database table.

7. javax.persistence.GeneratedValue: It defines the strategy to be used for the generation of the primary key. It is also used in conjunction with javax.persistence.GenerationType enum.

## Q17. What is Session in Hibernate and how to get it?

Hibernate Session is the interface between Java application layer and Hibernate. It is used to get a physical connection with the database. The Session object created is lightweight and designed to be instantiated each time an interaction is needed with the database. This Session provides methods to create, read, update and delete operations for a constant object. To get the Session, you can execute HQL queries, SQL native queries using the Session object.

## Q18. What is Hibernate SessionFactory?

SessionFactory is the factory class that is used to get the Session objects. The SessionFactory is a heavyweight object so usually, it is created during application startup and kept for later use. This SessionFactory is a thread-safe object which is used by all the threads of an application. If you are using multiple databases then you would have to create multiple SessionFactory objects.

## Q19. What is the difference between openSession and getCurrentSession?

This getCurrentSession() method returns the session bound to the context and for this to work, you need to configure it in Hibernate configuration file. Since this session object belongs to the context of Hibernate, it is okay if you don’t close it. Once the SessionFactory is closed, this session object gets closed.

openSession() method helps in opening a new session. You should close this session object once you are done with all the database operations. And also, you should open a new session for each request in a multi-threaded environment.

## Q20. What do you mean by Hibernate configuration file?

The following steps help in configuring Hibernate file:

1. First, identify the POJOs (Plain Old Java Objects) that have a database representation.

2. Identify which properties of POJOs need to be continued.

3. Annotate each of the POJOs in order to map the Java objects to columns in a database table.

4. Create a database schema using the schema export tool which uses an existing database, or you can create your own database schema.

5. Add Hibernate Java libraries to the application’s classpath.

6. Create a Hibernate XML configuration file that points to the database and the mapped classes.

7. In the Java application, you can create a Hibernate Configuration object that refers to your XML configuration file.

8. Also, build a Hibernate SessionFactory object from the Configuration object.

9. Retrieve the Hibernate Session objects from the SessionFactory and write down the data access logic for your application (create, retrieve, update, and delete).

## Q21. What are the key components of a Hibernate configuration object?

The configuration provides 2 key components, namely:

• Database Connection: This is handled by one or more configuration files.

• Class Mapping setup: It helps in creating the connection between Java classes and database tables.

## Q22. Discuss the Collections in Hibernate

Hibernate provides the facility to persist the Collections. A Collection basically can be a List, Set, Map, Collection, Sorted Set, Sorted Map. java.util.List, java.util.Set, java.util.Collection, etc, are some of the real interface types to declared the persistent collection-value fields. Hibernate injects persistent Collections based on the type of interface. The collection instances generally behave like the types of value behavior.

## Q23. What are the collection types in Hibernate?

There are five collection types in hibernate used for one-to-many relationship mappings.

• Bag

• Set

• List

• Array

• Map

## Q24. What is a Hibernate Template class?

When you integrate Spring and Hibernate, Spring ORM provides two helper classes – HibernateDaoSupport and HibernateTemplate. The main reason to use them was to get two things, the Session from Hibernate and Spring Transaction Management. However, from Hibernate 3.0.1, you can use the SessionFactory getCurrentSession() method to get the current session. The major advantage of using this Template class is the exception translation but that can be achieved easily by using @Repository annotation with service classes.

## Q25. What are the benefits of using Hibernate template?

The following are the benefits of using this Hibernate template class:

• Automated Session closing ability.

• The interaction with the Hibernate Session is simplified.

• Exception handling is automated.

## Q26. Which are the design patterns that are used in Hibernate framework?

There are a few design patterns used in Hibernate Framework, namely:

• Domain Model Pattern: An object model of the domain that incorporates both behavior as well as data.

• Data Mapper: A layer of the map that moves data between objects and a database while keeping it independent of each other and the map itself.

https://www.sourcecodeexamples.net/2018/04/data-mapper.html

• Proxy Pattern: It is used for lazy loading.

• Factory Pattern: Used in SessionFactory.

## Q27. Define Hibernate Validator Framework

Data validation is considered as an integral part of any application. Also, data validation is used in the presentation layer with the use of Javascript and the server-side code before processing. It occurs before persisting it in order to make sure it follows the correct format. Validation is a cross-cutting task, so we should try to keep it apart from the business logic. This Hibernate Validator provides the reference implementation of bean validation specs.

## Q28. What is Dirty Checking in Hibernate?

Hibernate incorporates Dirty Checking feature that permits developers and users to avoid time-consuming write actions. This Dirty Checking feature changes or updates fields that need to be changed or updated, while keeping the remaining fields untouched and unchanged.

Hibernate provides as feature called Automatic Dirty checking whereby changes to a persistent object are automatically saved to the database when the session is flushed or the transaction is committed. So the code does not need to invoke an explicit save or update. In this article, I will be covering this feature in detail.

Introduction

Hibernate supports 3 entity states, transient, persistent and detached. In order to understand more about entity states, you can refer to this article.

So basically, an object can move into the persistent state when any one of the following happens:

When the code invokes session.save, session.persist or session.saveorUpdate

When the code invokes session.load or session.get

Any changes to a persistent object are automatically saved to the database when the session in flushed. Flushing is the process of synchronizing the underlying database with the objects in the session. There is a session.flush method available but you generally don’t need to invoke it explicitly. A session gets flushed when the transaction is commited. The process of automatically updating the database with the changes to the persistent object when the session is flushed is known as automatic dirty checking.

An Example

Suppose we have a database table ‘Person’ which fields corresponding to ‘id’, ‘name’ and ‘age’. Suppose it has some records as follows:

Id Name Age

1 Bill 27

2 Jane 31

3 Anna 29

4 Dave 34

Now consider the following code:

SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Person person = session.load(Person.class, 2); //loads Person object for id 2

person.setAge(32);

tx.commit();

session.close();

HibernateUtil.closeSessionFactory();

So this code invokes the session.load method to load the person object corresponding to the id ‘2’. So this moves the person record with id 2 to the persistent state. It then invokes the setAge method and updates the value of the age field to 32. Then the code directly invokes the tx.commit method without invoking the session.save or session.update. Since the person object is in the persistent state, the change to the age field will automatically get saved to the database.

Conclusion

So in this article, we saw what is the automatic dirty checking feature in Hibernate and how you can use it to save changes to persistent objects.

## Q29. How can you share your views on mapping description files?

• Mapping description files are used by the Hibernate to configure functions.

• These files have the \*.hbm extension, which facilitates the mapping between database tables and Java class.

• Whether to use mapping description files or not this entirely depends on business entities.

Q30. What is meant by Light Object Mapping?

The means that the syntax is hidden from the business logic using specific design patterns. This is one of the valuable levels of ORM quality and this Light Object Mapping approach can be successful in case of applications where there are very fewer entities, or for applications having data models that are metadata-driven.

2) Hibernate Interview Questions for experienced

## Q31. What is meant by Hibernate tuning???

Optimizing the performance of Hibernate applications is known as Hibernate tuning.

The performance tuning strategies for Hibernate are:

1. SQL Optimization

2. Session Management

3. Data Caching

<https://vladmihalcea.com/hibernate-performance-tuning-tips/>

## Q32. What is Transaction Management in Hibernate? How does it work?

Transaction Management is a property which is present in the Spring framework. Now, what role does it play in Hibernate?

Transaction Management is a process of managing a set of commands or statements. In hibernate, Transaction Management is done by transaction interface. It maintains abstraction from the transaction implementation (JTA, JDBC). A transaction is associated with Session and is instantiated by calling session.beginTransaction().

## Q33. How do you integrate Hibernate with Struts2 or Servlet web applications?

You can integrate any Struts application with Hibernate. There are no extra efforts required.

1. Register a custom ServletContextListener.

2. In the ServletContextListener class, first, initialize the Hibernate Session, store it in the servlet context.

3. Action class helps in getting the Hibernate Session from the servlet context, and perform other Hibernate task as normal.

## Q34. What are the different states of a persistent entity?

It may exist in one of the following 3 states:

• Transient: This is not associated with the Session and has no representation in the database.

• Persistent: You can make a transient instance persistent by associating it with a Session.

• Detached: If you close the Hibernate Session, the persistent instance will become a detached instance.

## Q35. How can the primary key be created by using Hibernate?

A Primary key is a special relational database table column designated to uniquely identify all table records. It is specified in the configuration file hbm.xml. The generator can also be used to specify how a Primary key can be created in the database.

1

2

3

4 <id name="ClassID" type="string" >

<column name= "columnID" length="10" >

<generator/>

</id>

## Q36. Explain about Hibernate Proxy and how it helps in Lazy loading?

• Hibernate uses a proxy object in order to support Lazy loading.

• When you try loading data from tables, Hibernate doesn’t load all the mapped objects.

• After you reference a child object through getter methods, if the linked entity is not present in the session cache, then the proxy code will be entered to the database and load the linked object.

• It uses Java assist to effectively and dynamically generate sub-classed implementations of your entity objects.

## Q37. How can we see Hibernate generated SQL on console?

In order to view the SQL on a console, you need to add following in Hibernate configuration file to enable viewing SQL on the console for debugging purposes:

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

## Q38. What is Query Cache in Hibernate?

Hibernate implements a separate cache region for queries resultset that integrates with the Hibernate second-level cache. This is also an optional feature and requires a few more steps in code.

<property name="hibernate.cache.use\_query\_cache">true</property>

And in code, we need to use setCacheable(true) method of Query, quick example looks like below.

Query query = session.createQuery("from Employee");

query.setCacheable(true);

query.setCacheRegion("ALL\_EMP");

Next

Note: This is only useful for queries that are run frequently with the same parameters.

## Q39. What is the benefit of Native SQL query support in Hibernate?

Hibernate provides an option to execute Native SQL queries through the use of the SQLQuery object. For normal scenarios, it is however not the recommended approach because you might lose other benefits like Association and Hibernate first-level caching.

Native SQL Query comes handy when you want to execute database-specific queries that are not supported by Hibernate API such query hints or the Connect keyword in Oracle Database.

## Q40. What is Named SQL Query?

Hibernate provides another important feature called Named Query using which you can define at a central location and use them anywhere in the code.

You can create named queries for both HQL as well as for Native SQL. These Named Queries can be defined in Hibernate mapping files with the help of JPA annotations @NamedQuery and @NamedNativeQuery.

## Q41. When do you use merge() and update() in Hibernate?

This is one of the tricky Hibernate Interview Questions asked.

update(): If you are sure that the Hibernate Session does not contain an already persistent instance with the same id .

merge(): Helps in merging your modifications at any time without considering the state of the Session.

## Q42. Difference between get() vs load() method in Hibernate?

This is one of the most frequently asked Hibernate Interview Questions. The key difference between the get() and load() method is:

load(): It will throw an exception if an object with an ID passed to them is not found.

get(): Will return null.

load(): It can return proxy without hitting the database unless required.

get(): It always goes to the database.

So sometimes using load() can be faster than the get() method.

## Q43. Difference between the first and second level cache in Hibernate?

The first-level cache is maintained at Session level while the second level cache is maintained at a SessionFactory level and is shared by all sessions.

## Q44. Difference between Session and SessionFactory in Hibernate?

This is yet another popular Hibernate Interview Question asked.

• A Session is a single-threaded, short-lived object. It provides the first-level cache.

• SessionFactory is immutable and shared by all Session. It also lives until the Hibernate is running. It also provides the second-level cache.

## Q45. Difference between save() and saveOrUpdate() method of Hibernate?

Even though save() and saveOrUpdate() method is used to store an object into Database, the key difference between them is that save() can only Insert records but saveOrUpdate() can either Insert or Update records.

## Q46. Difference between sorted and ordered collection in Hibernate?

sorted collection sort the data in JVM’s heap memory using Java’s collection framework sorting methods. The ordered collection is sorted using order by clause in the database itself.

Note: A sorted collection is more suited for small dataset but for a large dataset, it’s better to use ordered collection to avoid

## Q47. Difference between the transient, persistent and detached state in Hibernate?

Transient state: New objects are created in the Java program but are not associated with any Hibernate Session.

Persistent state: An object which is associated with a Hibernate session is called Persistent object. While an object which was earlier associated with Hibernate session but currently it’s not associate is known as a detached object. You can call save() or persist() method to store those object into the database and bring them into the Persistent state.

Detached state: You can re-attach a detached object to Hibernate sessions by calling either update() or saveOrUpdate() method.

## Q48. Difference between managed associations and Hibernate associations?

Managed associations: Relate to container management persistence and are bi-directional.

Hibernate Associations: These associations are unidirectional.

## Q49. What are the best practices that Hibernate recommends for persistent classes?

• 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.

## Q50. What are the best practices to follow with Hibernate framework?

• Always check the primary key field access, if it’s generated at the database layer then you should not have a setter for this.

• By default hibernate set the field values directly, without using setters. So if you want Hibernate to use setters, then make sure proper access is defined as @Access(value=AccessType.PROPERTY).

• If access type is property, make sure annotations are used with getter methods and not setter methods. Avoid mixing of using annotations on both filed and getter methods.

• Use native sql query only when it can’t be done using HQL, such as using the database-specific feature.

• If you have to sort the collection, use ordered list rather than sorting it using Collection API.

### • Use named queries wisely, keep it at a single place for easy debugging. Use them for commonly used queries only. For entity-specific query, you can keep them in the entity bean itself.

• For web applications, always try to use JNDI DataSource rather than configuring to create a connection in hibernate.

• Avoid Many-to-Many relationships, it can be easily implemented using bidirectional One-to-Many and Many-to-One relationships.

• For collections, try to use Lists, maps and sets. Avoid array because you don’t get benefit of lazy loading.

### • Do not treat exceptions as recoverable, roll back the Transaction and close the Session. If you do not do this, Hibernate cannot guarantee that the in-memory state accurately represents the persistent state.

• Prefer DAO pattern for exposing the different methods that can be used with entity bean

• Prefer lazy fetching for associations

## What will happen if we don’t have no-args constructor in Entity bean?

### Hibernate uses Reflection API to create instance of Entity beans, usually when you call get() or load() methods. The method Class.newInstance() is used for this and it requires no-args constructor. So if you won’t have no-args constructor in entity beans, hibernate will fail to instantiate it and you will get HibernateException

## How to implement Joins in Hibernate?

There are various ways to implement joins in hibernate.

Using associations such as one-to-one, one-to-many etc.

Using JOIN in the HQL query. There is another form “join fetch” to load associated data simultaneously, no lazy loading.

We can fire native sql query and use join keyword.

## Why we should not make Entity Class final?

Hibernate use proxy classes for lazy loading of data, only when it’s needed. This is done by extending the entity bean, if the entity bean will be final then lazy loading will not be possible, hence low performance.

## Criteria API

Hibernate provides Criteria API that is more object oriented for querying the database and getting results. We can’t use Criteria to run update or delete queries or any DDL statements. It’s only used to fetch the results from the database using more object oriented approach.

Some of the common usage of Criteria API are:

Criteria API provides Projection that we can use for aggregate functions such as sum(), min(), max() etc.

Criteria API can be used with ProjectionList to fetch selected columns only.

Criteria API can be used for join queries by joining multiple tables, useful methods are createAlias(), setFetchMode() and setProjection()

Criteria API can be used for fetching results with conditions, useful methods are add() where we can add Restrictions.

Criteria API provides addOrder() method that we can use for ordering the resu

## Cascading

When we have relationship between entities, then we need to define how the different operations will affect the other entity. This is done by cascading and there are different types of it.

Here is a simple example of applying cascading between primary and secondary entities.

import org.hibernate.annotations.Cascade;

@Entity

@Table(name = "EMPLOYEE")

public class Employee {

@OneToOne(mappedBy = "employee")

@Cascade(value = org.hibernate.annotations.CascadeType.ALL)

private Address address;

}

Note that Hibernate CascadeType enum constants are little bit different from JPA javax.persistence.CascadeType, so we need to use the Hibernate CascadeType and Cascade annotations for mappings, as shown in above example.

Commonly used cascading types as defined in CascadeType enum are:

None: No Cascading, it’s not a type but when we don’t define any cascading then no operations in parent affects the child.

ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. Basically everything

SAVE\_UPDATE: Cascades save and update, available only in hibernate.

DELETE: Corresponds to the Hibernate native DELETE action, only in hibernate.

DETATCH, MERGE, PERSIST, REFRESH and REMOVE – for similar operations

LOCK: Corresponds to the Hibernate native LOCK action.

REPLICATE: Corresponds to the Hibernate native REPLICATE action.

## Hibernate merge,save and other methods

The Session interface has several methods that eventually result in saving data to the database: persist, save, update, merge, saveOrUpdate. To understand the difference between these methods, we must first discuss the purpose of the Session as a persistence context and the difference between the states of entity instances in relation to the Session.

We should also understand the history of Hibernate development that led to some partly duplicated API methods.

Apart from object-relational mapping itself, one of the problems that Hibernate was intended to solve is the problem of managing entities during runtime. The notion of “persistence context” is Hibernate's solution to this problem. Persistence context can be thought of as a container or a first-level cache for all the objects that you loaded or saved to a database during a session.

The session is a logical transaction, which boundaries are defined by your application’s business logic. When you work with the database through a persistence context, and all of your entity instances are attached to this context, you should always have a single instance of entity for every database record that you’ve interacted during the session with.

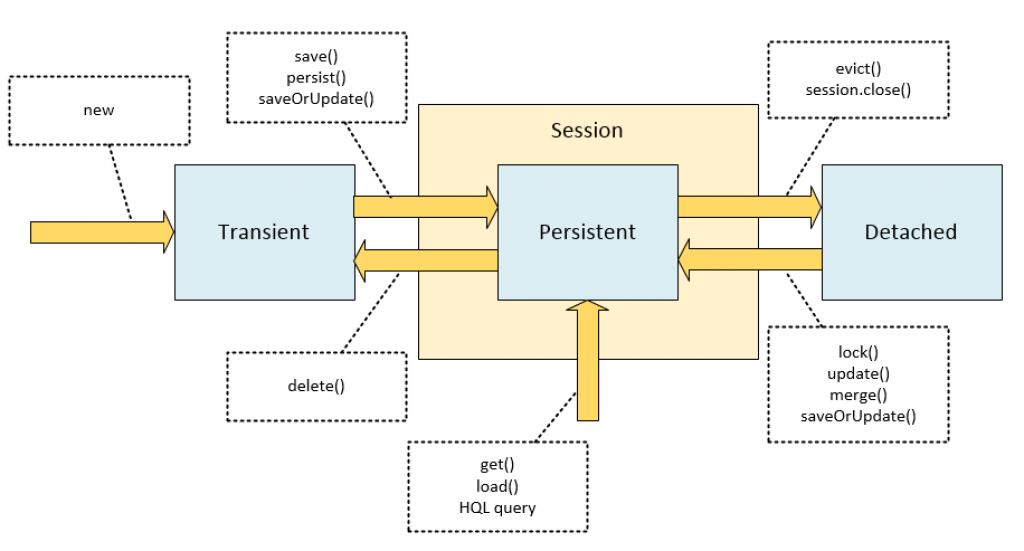
In Hibernate, the persistence context is represented by org.hibernate.Session instance. For JPA, it is the javax.persistence.EntityManager. When we use Hibernate as a JPA provider and operate via EntityManager interface, the implementation of this interface basically wraps the underlying Session object. However, Hibernate Session provides a richer interface with more possibilities so sometimes it is useful to work with Session directly.

### **2.2. States of Entity Instances**

Any entity instance in your application appears in one of the three main states in relation to the Session persistence context:

* transient — this instance is not, and never was, attached to a Session; this instance has no corresponding rows in the database; it's usually just a new object that you have created to save to the database;
* persistent — this instance is associated with a unique Session object; upon flushing the Session to the database, this entity is guaranteed to have a corresponding consistent record in the database;
* detached — this instance was once attached to a Session (in a persistent state), but now it’s not; an instance enters this state if you evict it from the context, clear or close the Session, or put the instance through serialization/deserialization process.

Here is a simplified state diagram with comments on Session methods that make the state transitions happen.



2.3. Conformity to JPA Specification

Hibernate was the most successful Java ORM implementation. No wonder that the specification for Java persistence API (JPA) was heavily influenced by the Hibernate API. Unfortunately, there were also many differences: some major, some more subtle.

To act as an implementation of the JPA standard, Hibernate APIs had to be revised. Several methods were added to Session interface to match the EntityManager interface. These methods serve the same purpose as the “original” methods, but conform to the specification and thus have some differences.

3. Differences Between the Operations

It is important to understand from the beginning that all of the methods (persist, save, update, merge, saveOrUpdate) do not immediately result in the corresponding SQL UPDATE or INSERT statements. The actual saving of data to the database occurs on committing the transaction or flushing the Session.

The mentioned methods basically manage the state of entity instances by transitioning them between different states along the lifecycle.

As an example entity, we will use a simple annotation-mapped entity Person:

### **3.1.**Persist

The persist method is intended for adding a new entity instance to the persistence context, i.e. transitioning an instance from transient to persistent state.

We usually call it when we want to add a record to the database (persist an entity instance):

Person person = new Person();

person.setName("John");

session.persist(person);

What happens after the persist method is called? The person object has transitioned from transient to persistent state. The object is in the persistence context now, but not yet saved to the database. The generation of INSERT statements will occur only upon commiting the transaction, flushing or closing the session.

Notice that the persist method has void return type. It operates on the passed object “in place”, changing its state. The person variable references the actual persisted object.

This method is a later addition to the Session interface. The main differentiating feature of this method is that it conforms to the JSR-220 specification (EJB persistence). The semantics of this method is strictly defined in the specification, which basically states, that:

a transient instance becomes persistent (and the operation cascades to all of its relations with cascade=PERSIST or cascade=ALL),

if an instance is already persistent, then this call has no effect for this particular instance (but it still cascades to its relations with cascade=PERSIST or cascade=ALL),

if an instance is detached, you should expect an exception, either upon calling this method, or upon committing or flushing the session.

Notice that there is nothing here that concerns the identifier of an instance. The spec does not state that the id will be generated right away, regardless of the id generation strategy. The specification for the persist method allows the implementation to issue statements for generating id on commit or flush, and the id is not guaranteed to be non-null after calling this method, so you should not rely upon it.

You may call this method on an already persistent instance, and nothing happens. But if you try to persist a detached instance, the implementation is bound to throw an exception. In the following example we persist the entity, evict it from the context so it becomes detached, and then try to persist again. The second call to session.persist() causes an exception, so the following code will not work:

Person person = new Person();

person.setName("John");

session.persist(person);

session.evict(person);

session.persist(person); // PersistenceException!

3.2. Save

The save method is an “original” Hibernate method that does not conform to the JPA specification.

Its purpose is basically the same as persist, but it has different implementation details. The documentation for this method strictly states that it persists the instance, “first assigning a generated identifier”. The method is guaranteed to return the Serializable value of this identifier.

Person person = new Person();

person.setName("John");

Long id = (Long) session.save(person);

The effect of saving an already persisted instance is the same as with persist. Difference comes when you try to save a detached instance:

Person person = new Person();

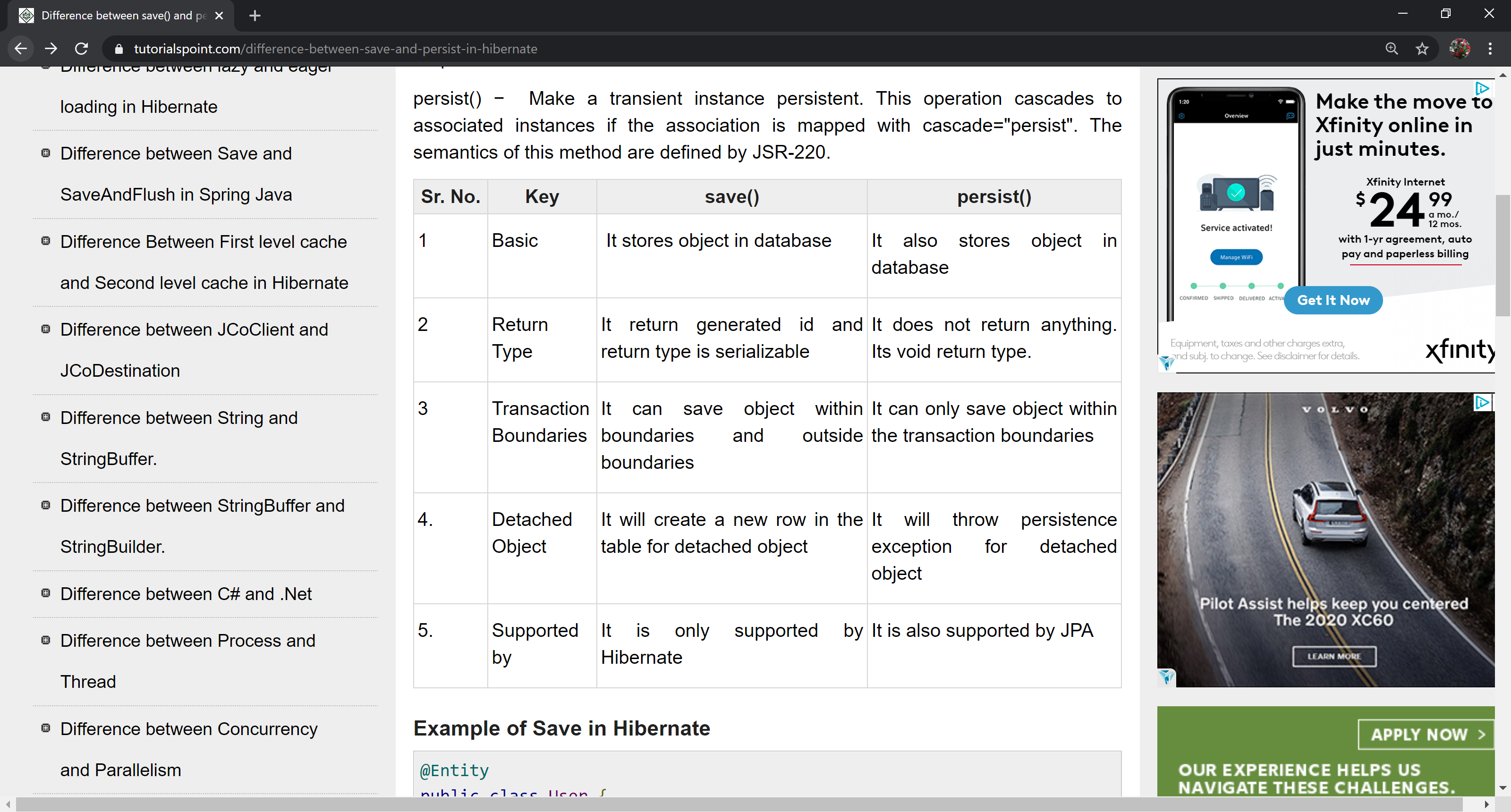
person.setName("John");

Long id1 = (Long) session.save(person);

session.evict(person);

Long id2 = (Long) session.save(person);

The id2 variable will differ from id1. The call of save on a detached instance creates a new persistent instance and assigns it a new identifier, which results in a duplicate record in a database upon committing or flushing.



3.3. Merge

The main intention of the merge method is to update a persistent entity instance with new field values from a detached entity instance.

For instance, suppose you have a RESTful interface with a method for retrieving an JSON-serialized object by its id to the caller and a method that receives an updated version of this object from the caller. An entity that passed through such serialization/deserialization will appear in a detached state.

After deserializing this entity instance, you need to get a persistent entity instance from a persistence context and update its fields with new values from this detached instance. So the merge method does exactly that:

finds an entity instance by id taken from the passed object (either an existing entity instance from the persistence context is retrieved, or a new instance loaded from the database);

copies fields from the passed object to this instance;

returns newly updated instance.

In the following example we evict (detach) the saved entity from context, change the name field, and then merge the detached entity.

Person person = new Person();

person.setName("John");

session.save(person);

session.evict(person);

person.setName("Mary");

Person mergedPerson = (Person) session.merge(person);

Note that the merge method returns an object — it is the mergedPerson object that was loaded into persistence context and updated, not the person object that you passed as an argument. Those are two different objects, and the person object usually needs to be discarded (anyway, don't count on it being attached to persistence context).

As with persist method, the merge method is specified by JSR-220 to have certain semantics that you can rely upon:

if the entity is detached, it is copied upon an existing persistent entity;

if the entity is transient, it is copied upon a newly created persistent entity;

this operation cascades for all relations with cascade=MERGE or cascade=ALL mapping;

if the entity is persistent, then this method call does not have effect on it (but the cascading still takes place).

3.4. Update

As with persist and save, the update method is an “original” Hibernate method that was present long before the merge method was added. Its semantics differs in several key points:

it acts upon passed object (its return type is void); the update method transitions the passed object from detached to persistent state;

this method throws an exception if you pass it a transient entity.

In the following example we save the object, then evict (detach) it from the context, then change its name and call update. Notice that we don't put the result of the update operation in a separate variable, because the update takes place on the person object itself. Basically we're reattaching the existing entity instance to the persistence context — something the JPA specification does not allow us to do.

Person person = new Person();

person.setName("John");

session.save(person);

session.evict(person);

person.setName("Mary");

session.update(person);

Trying to call update on a transient instance will result in an exception. The following will not work:

Person person = new Person();

person.setName("John");

session.update(person); // PersistenceException!

3.5. SaveOrUpdate

This method appears only in the Hibernate API and does not have its standardized counterpart. Similar to update, it also may be used for reattaching instances.

Actually, the internal DefaultUpdateEventListener class that processes the update method is a subclass of DefaultSaveOrUpdateListener, just overriding some functionality. The main difference of saveOrUpdate method is that it does not throw exception when applied to a transient instance; instead, it makes this transient instance persistent. The following code will persist a newly created instance of Person:

Person person = new Person();

person.setName("John");

session.saveOrUpdate(person);

You may think of this method as a universal tool for making an object persistent regardless of its state wether it is transient or detached.

4. What to Use?

If you don't have any special requirements, as a rule of thumb, you should stick to the persist and merge methods, because they are standardized and guaranteed to conform to the JPA specification.

They are also portable in case you decide to switch to another persistence provider, but they may sometimes appear not so useful as the “original” Hibernate methods, save, update and saveOrUpdate.

<https://thorben-janssen.com/persist-save-merge-saveorupdate-whats-difference-one-use/>