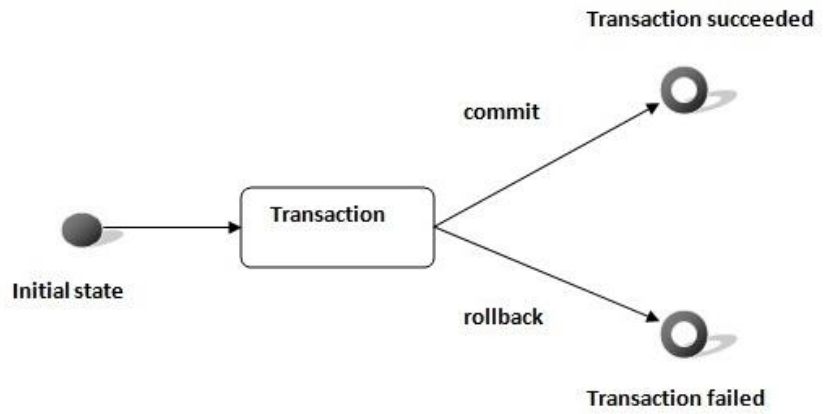
Hibernate Transaction Management Example

A **transaction** simply represents a unit of work. In such case, if one step fails, the whole transaction fails (which is termed as atomicity). A transaction can be described by ACID properties (Atomicity, Consistency, Isolation and Durability).



Transaction Interface in Hibernate

In hibernate framework, we have **Transaction** interface that defines the unit of work. It maintains abstraction from the transaction implementation (JTA,JDBC).

A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.

6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully.

Example of Transaction Management in Hibernate

In hibernate, it is better to rollback the transaction if any exception occurs, so that resources can be free. Let's see the example of transaction management in hibernate.

```
Session session = null;
Transaction tx = null;

try {
session = sessionFactory.openSession();
tx = session.beginTransaction();
//some action

tx.commit();

}catch (Exception ex) {
ex.printStackTrace();
tx.rollback();
}
finally {session.close();}
```

← prev next →