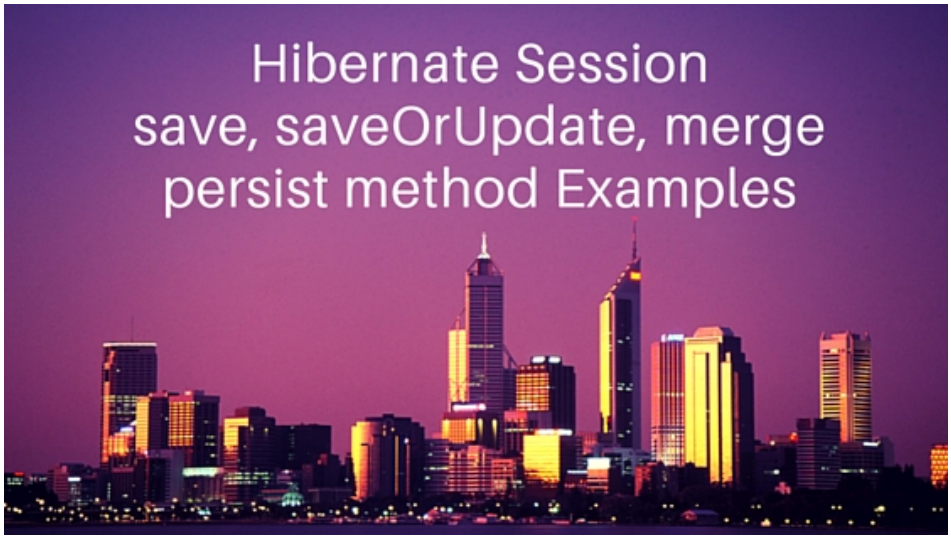
Hibernate Session merge, update, save, saveOrUpdate, persist example

APRIL 2, 2018 BY PANKAJ — 43 COMMENTS

Hibernate Session is the interface between java application and hibernate framework. Today we will look into Session important methods for saving and updating data in tables – **save**, **saveOrUpdate**, **persist**, **update** and **merge**.

Table of Contents [hide]

Hibernate Session



Hibernate Session save

As the method name suggests, **hibernate save()** can be used to save entity to database. We can invoke this method outside a transaction, that's why I don't like this method to save data. If we use this without transaction and we have cascading between entities, then only the primary entity gets saved **unless we flush the session**.

For our testing purposes we have two entity beans – `Employee` and `Address`.

```
package com.journaldev.hibernate.model;

import javax.persistence.Access;
import javax.persistence.AccessType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.Cascade;

@Entity
@Table(name = "EMPLOYEE")
@Access(value=AccessType.FIELD)
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
package com.journaldev.hibernate.model;

import javax.persistence.Access;
import javax.persistence.AccessType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name = "ADDRESS")
@Access(value=AccessType.FIELD)
public class Address {

	@Id
	@Column(name = "emp_id", unique = true, nullable = false)
```

Here is a simple hibernate program where we are invoking `save()` method in different cases.

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Address;
import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateSaveExample {

	public static void main(String[] args) {

		// Prep Work
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
```

```
//save example - without transaction
Session session = sessionFactory.openSession();
Employee emp = getTestEmployee();
long id = (Long) session.save(emp);
System.out.println("1. Employee save called without transaction,
```

When we execute above program, it produces following output.

```
Hibernate: insert into EMPLOYEE (emp_name, emp_salary) values (?, ?)
1. Employee save called without transaction, id=149
Hibernate: insert into ADDRESS (address_line1, city, zipcode, emp_id) values (?, ?,
?, ?)
*****
Hibernate: insert into EMPLOYEE (emp_name, emp_salary) values (?, ?)
2. Employee save called with transaction, id=150
3. Before committing save transaction
Hibernate: insert into ADDRESS (address_line1, city, zipcode, emp_id) values (?, ?,
?, ?)
4. After committing save transaction
*****
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as
city3_0_1_, address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer
join ADDRESS address1_ on employee0_.emp_id=address1_.emp_id where
employee0_.emp_id=?
Employee Details=Id= 20, Name= Kumar1, Salary= 1000.0, {Address= AddressLine1= Test
address1, City=Blr, Zipcode=12121}
5. Employee save called with transaction, id=20
6. Before committing save transaction
```

Few important points that we can confirm from above output are:

- We should avoid save outside transaction boundary, otherwise mapped entities will not be saved causing data inconsistency. It's very normal to forget flushing the session because it doesn't throw any exception or warnings.
- Hibernate save method returns the generated id immediately, this is possible because primary object is saved as soon as save method is invoked.
- If there are other objects mapped from the primary object, they gets saved at the time of committing transaction or when we flush the session.
- For objects that are in persistent state, save updates the data through update query. Notice that it happens when transaction is committed. If there are no changes in the object, there wont be any query fired. If you will run above program multiple times, you will notice that update queries are not fired next time because there is no change in the column values.

- Hibernate save load entity object to persistent context, if you will update the object properties after the save call but before the transaction is committed, it will be saved into database.

Hibernate Persist

Hibernate persist is similar to save (with transaction) and it adds the entity object to the **persistent context**, so any further changes are tracked. If the object properties are changed before the transaction is committed or session is flushed, it will also be saved into database.

Second difference is that we can use `persist()` method only within the boundary of a transaction, so it's safe and takes care of any cascaded objects.

Finally, persist doesn't return anything so we need to use the persisted object to get the generated identifier value. Let's look at hibernate persist with a simple program.

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernatePersistExample {

	public static void main(String[] args) {

		// Prep Work
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

		//persist example - with transaction
		Session session2 = sessionFactory.openSession();
		Transaction tx2 = session2.beginTransaction();
		Employee emp2 = HibernateSaveExample.getTestEmployee();
		session2.persist(emp2);
		System.out.println("Persist called");
```

Output produced by above code is:

```
Hibernate: insert into EMPLOYEE (emp_name, emp_salary) values (?, ?)
8. Employee persist called with transaction, id=158, address id=158
Hibernate: insert into ADDRESS (address_line1, city, zipcode, emp_id) values (?, ?, ?,
?)
Hibernate: update EMPLOYEE set emp_name=?, emp_salary=? where emp_id=?
*****
```

Notice that first employee object is inserted, then at the time of transaction commit, update query is executed to update the name value. Also mapped object address is saved into database.

Hibernate saveOrUpdate

Hibernate saveOrUpdate results into insert or update queries based on the provided data. If the data is present in the database, update query is executed.

We can use `saveOrUpdate()` without transaction also, but again you will face the issues with mapped objects not getting saved if session is not flushed.

Hibernate saveOrUpdate adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist.

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateSaveOrUpdateExample {

        public static void main(String[] args) {

                // Prep Work
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();

                //saveOrUpdate example - without transaction
                Session session5 = sessionFactory.openSession();
                Employee emp5 = HibernateSaveExample.getTestEmployee();
                session5.saveOrUpdate(emp5);
                System.out.println("*****");
```

Above program produces following output.

```
Hibernate: insert into EMPLOYEE (emp_name, emp_salary) values (?, ?)
*****
Hibernate: insert into EMPLOYEE (emp_name, emp_salary) values (?, ?)
9. Before committing saveOrUpdate transaction. Id=166
Hibernate: insert into ADDRESS (address_line1, city, zipcode, emp_id) values (?, ?, ?,
?)
Hibernate: update EMPLOYEE set emp_name=?, emp_salary=? where emp_id=?
10. After committing saveOrUpdate transaction
*****
11. Before committing saveOrUpdate transaction. Id=166
Hibernate: update ADDRESS set address_line1=?, city=?, zipcode=? where emp_id=?
12. After committing saveOrUpdate transaction
*****
```

Notice that without transaction, only Employee gets saved and address information is lost.

With transaction employee object is tracked for any changes, thats why in last call there is no update in Employee table even though the value was changed in between, final value remains same.

Hibernate update

Hibernate update should be used where we know that we are only updating the entity information. This operation adds the entity object to **persistent context** and further changes are tracked and saved when transaction is committed. Let's check this behavior with a simple program.

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateUpdateExample {

        public static void main(String[] args) {

                // Prep Work
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
```

```
                Session session = sessionFactory.openSession();
                Transaction tx = session.beginTransaction();
                Employee emp = (Employee) session.load(Employee.class, new
Long(101));
                System.out.println("Employee object loaded. " + emp);
                tx.commit();
```

When we execute above program for the first time, we get following output.

```
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer join ADDRESS
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
Employee object loaded. Id= 101, Name= Test Emp, Salary= 1000.0, {Address=
AddressLine1= Test address1, City=Test City, Zipcode=12121}
13. Before committing update transaction
Hibernate: update EMPLOYEE set emp_name=?, emp_salary=? where emp_id=?
Hibernate: update ADDRESS set address_line1=?, city=?, zipcode=? where emp_id=?
14. After committing update transaction
```

On further execution, we get following output.

```
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer join ADDRESS
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
Employee object loaded. Id= 101, Name= Final updated name, Salary= 1000.0, {Address=
AddressLine1= Test address1, City=Bangalore, Zipcode=12121}
13. Before committing update transaction
14. After committing update transaction
```

Notice that there are no updates fired after first execution because there are no update in values. Also notice the employee name is “Final updated name” that we set after invoking update() method. This confirms that hibernate was tracking the object for any changes and at the time of committing transaction, this value got saved.

Hibernate Merge

Hibernate merge can be used to update existing values, however this method create a copy from the passed entity object and return it. The returned object is part of persistent context and tracked for any changes, passed object is not tracked. This is the major difference with merge() from all other methods. Let's look at this with a simple program.

```
package com.journaldev.hibernate.main;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.journaldev.hibernate.model.Employee;
import com.journaldev.hibernate.util.HibernateUtil;

public class HibernateMergeExample {

        public static void main(String[] args) {

                // Prep Work
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                Session session = sessionFactory.openSession();
                Transaction tx = session.beginTransaction();
                Employee emp = (Employee) session.load(Employee.class, new
Long(101));
                System.out.println("Employee object loaded. " + emp);
                tx.commit();
```

Output in first execution is:

```
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer join ADDRESS
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
Employee object loaded. Id= 101, Name= Final updated name, Salary= 1000.0, {Address=
AddressLine1= Test address1, City=Bangalore, Zipcode=12121}
false
15. Before committing merge transaction
Hibernate: update EMPLOYEE set emp_name=?, emp_salary=? where emp_id=?
16. After committing merge transaction
```

In further execution, output produced is:

```
Hibernate: select employee0_.emp_id as emp_id1_1_0_, employee0_.emp_name as
emp_name2_1_0_, employee0_.emp_salary as emp_sala3_1_0_, address1_.emp_id as
emp_id1_0_1_, address1_.address_line1 as address_2_0_1_, address1_.city as city3_0_1_,
address1_.zipcode as zipcode4_0_1_ from EMPLOYEE employee0_ left outer join ADDRESS
address1_ on employee0_.emp_id=address1_.emp_id where employee0_.emp_id=?
Employee object loaded. Id= 101, Name= Kumar, Salary= 25000.0, {Address= AddressLine1=
Test address1, City=Bangalore, Zipcode=12121}
false
15. Before committing merge transaction
16. After committing merge transaction
```

Notice that the entity object returned by merge() is different from the passed entity. Also notice that in further execution, name is “Kumar”, this is because the returned object is tracked for any changes.

That’s all for **Hibernate Session** `save` and `update` methods, I hope that examples above will help you in clarifying any doubts you have.

« PREVIOUS
[Solved] org.hibernate.AnnotationException: No identifier specified for entity Class

NEXT »
Hibernate SessionFactory

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: HIBERNATE

Comments

Taufique Shaikh says

APRIL 9, 2018 AT 10:47 AM

Save method doesn't save anything until flush or commit happen. It only assigns identifier to the entity and saves only when flush or commit is called.

You mentioned in the tutorial that it saves the first level entity and doesn't save the mapped entity.

Please correct it.

Reply

Prince Gupta says

APRIL 10, 2018 AT 6:54 AM

It depends upon the identifier generator strategy, if it is IDENTITY then it creates first level entry, for sequence it doesn't create any entry. It creates the entity on flush or commit of transaction.

Reply

tekilaina says

NOVEMBER 4, 2017 AT 2:33 PM

this code works fine with out any error or flaw….

referenced entity get saved even without flush in save() method.

Person per=new Person();

Address add=new Address();

per.setName("sfdf");

add.setAddress("acadf");

per.setAddress(add);
add.setPerson1(per);
session.save(per);
Reply

tekilaina says
NOVEMBER 4, 2017 AT 1:31 PM
Well i'm doing the same thing,
Person per=new Person();
Address add=new Address();
add.setAddress("sohal");
add.setPerson1(per);
per.setAddress(add);
per.setName("hello");
session.save(per);
it give me exception : object references an unsaved transient instance – save the transient instance before flushing: model.Person.address -> model.Address
Reply

Akash Mohapatra says
JUNE 14, 2017 AT 8:49 AM
I am inserting my data into Oracle database using hibernate session.save() method but auto increment is provided by database side. In my bean class @Id showing me error how I can solve this problem???
Reply

Brian says
MAY 4, 2017 AT 10:21 AM
I gives me error on System.out.println("Employee Details="+emp6);, saying something about identifier.. An example who won't bother taking care of what he writes.
Reply

adrian says
JULY 10, 2017 AT 1:16 PM
issue is because emp6 is not being loaded properly !
Look at your database table and see exactly how the id's of the employees are being STORED.
You will notice that he loads the employee with ID = 20. My database did not store the employees with that id. Once I made that realization I was able to see exactly how to fix the code. (Hint: change

from 20 to 1 if your db is setup just to incrementally update the id)
Reply

Shauank says
NOVEMBER 20, 2016 AT 10:39 AM
Hi Pankaj,
Just tried to do save without transaction. But what I see, query getting executed but DB does not have new rows. Here is sample code which I did…
SessionFactory sessionFactory = getSessionFactory();
Session session = sessionFactory.openSession();
Addresses addresses = new Addresses("stree1", "city", "state", 12345);
session.save(addresses);
session.flush();
sessionFactory.close();
Output:
Hibernate: insert into addresses (city, pincode, state, street) values (?, ?, ?, ?)
Nov 21, 2016 12:07:21 AM
org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool
Reply

Jay Cheng says
AUGUST 20, 2016 AT 1:37 AM
Hi dear author! Can you attach the example code? I am learning but encounter some problem.
Reply

kaptan singh says
AUGUST 11, 2016 AT 2:18 AM
Dear Pankaj
session.flush(); //address will not get saved without this
In Save example does it mean employee details will save and address will not save
if i am not using above statement..
But i have try without above statement the employee details is not saved.
So Kindly explain me.
Reply

kaptan singh says
AUGUST 10, 2016 AT 10:31 PM
In save example without transaction .
statement – session.flush(); //address will not get saved without this
if i am not using this statement , does it mean employee details will be save and address details not save.
But try same without this statement employee will not be save.
Reply

Ravi Kumar says
JULY 21, 2016 AT 8:56 PM
You mentioned that :
One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist.
Is it really true , because in the example of Save method you are contradicting with your own words and saying that:
Hibernate save load entity object to persistent context, if you will update the object properties after the save call but before the transaction is committed, it will be saved into database.
What is the truth…
Reply

Pankaj says
JULY 21, 2016 AT 10:21 PM
Hi Ravi,
The description was confusing, both `save` and `saveOrUpdate` behaves in same way with Transaction.
I have corrected it.
Reply

Deepti chaudhary says
MAY 19, 2016 AT 12:08 AM
Nice explaination □
Reply

Anshuman Dwivedi says
MARCH 15, 2016 AT 11:28 AM
Hi Pankaj,
Are save() and saveOrUpdate() both are putting/storing data into persistence context ? By reading above article I understand whatever is being stored into persistence context is closely monitored by hibernate till txn.commit/session.flush to reflect remaining changes into DB.
In save() you said "save() puts/stores entity object to persistent context, if you will update the object properties after the save call but before the transaction is committed, it will be saved into database."
On contrary in saveOrUpdate() description you said "One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction."
Which one is correct ? Kindly suggest.
Reply

Bhargav says
MAY 30, 2016 AT 10:08 PM
I have same confusion.
"One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist." looks no difference to save and saveOrUpdate
Hi Pankaj –
Can you help us to erase all the doubt. ?
Thanks
Bhargav
Reply

rajeshkumar says
MARCH 10, 2016 AT 1:19 PM
Excellent and simple explanation
Reply

Ravi Kumar says
JULY 21, 2016 AT 9:00 PM
Have you read the tutorial carefully Rajesh, As I have some confusion If you understood well…
Pankaj mentioned that :
One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist.

Is it really true , because in the example of Save method Pankaj is contradicting with his own words and saying that:
Hibernate save load entity object to persistent context, if you will update the object properties after the save call but before the transaction is committed, it will be saved into database.
…I am asking this question to you because multiple people including you have commented that the tutorial is excellent. No doubt tutorial is excellent and I am learning a lot from this . but question should arise if you are reading carefully.
Reply

obulreddy says
MARCH 10, 2016 AT 6:18 AM
Thanks for explanation !!
Reply

Anu says
MARCH 1, 2016 AT 10:05 PM
Excellent tutorial. Concepts are very clearly defined with well-thought out examples.
Reply

tekilaina says
NOVEMBER 4, 2017 AT 2:38 PM
program run kr k dekha kabi…?
Reply

Rahul says
JANUARY 22, 2016 AT 11:56 AM
Very simple explanation, Thanks
Reply

Binh Thanh Nguyen says
OCTOBER 14, 2015 AT 3:21 AM
Thanks, nice post

Reply

Prithvi says
JUNE 24, 2015 AT 7:25 AM
HI
Reply

vlad says
MARCH 25, 2015 AT 8:14 AM
"The returned object is part of persistent context and tracked for any changes, passed object is not tracked."
I don't think that this is true. Try to switch the two lines:
emp4.setName("Kumar");
emp.setName("Test");
and you will get name "Test", which means that both objects are tracked.
Reply

levik says
MAY 2, 2015 AT 7:33 AM
Hi Vlad,
Actually this is true, I've try right test and verify what value was set and it is "Kumar".
You can run two tests
UserServiceTest#shouldSuccessfulMergeWithTransactionAfterMargeEmailOldValueAndNewValue
and shouldSuccessfulMergeWithTransactionAfterMargeEmailChangeOrderNewValueAndOldValue
After verification you see that Pankaj right.
Reply

Vaibhav Mittal says
MARCH 21, 2015 AT 7:32 AM
I found this line confusing
"One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction, like persist."
As both are doing same, if you look at the below code:
long id = (Long) session.save(emp);

emp.setName("Test");
session.flush();
In both case update will get called on employee. How you can say 'it adds the entity object to persistent context'?
Please specify I am getting it wrong.
Reply

Siva says
JANUARY 25, 2015 AT 7:55 PM
Transaction tx1 = session.beginTransaction();
Session session1 = sessionFactory.openSession();
Employee emp1 = getTestEmployee();
long id1 = (Long) session1.save(emp1);
tx1.commit();
How is the tx1 that belongs to earlier session updating the values of current session (session1)?
Reply

Kk Jackson says
DECEMBER 19, 2014 AT 12:09 PM
Hahaha Loser Ankit!!!
Reply

npk says
FEBRUARY 17, 2015 AT 11:28 PM
Learn to help others to rectify the mistake instead of laughing at them…you think you are Hero in every direction and none is above you…
nothing to laugh at others…beginner will always struggle at start.
Reply

Ankit says
SEPTEMBER 9, 2014 AT 6:09 AM
The post is showing difference correct.By default autocommit propertyis false this is reason the tables are not populated while using save and flush.Set autocommit true then you will see the difference.
Reply

Kapil says
SEPTEMBER 14, 2014 AT 10:23 PM
Hello Ankit ,
I have set true auto commit property. But there is no difference between save and persist() methods.
Please clear this
Reply

Ankit says
SEPTEMBER 9, 2014 AT 6:04 AM
Hi All,
I found the problem where is.The problem in hibernate.cfg.xml file ,by default autocommit property is false.Please set true auto commit property then save and flush will work.then you will see the difference between save and persist method.i was wrong in my last comment,in article is showing the difference correct.
Reply

Sevak says
JANUARY 6, 2016 AT 8:41 AM
Absolutely correct Ankit. I was somehow wondering if article is wrong as connection.autocommit was not present in my cfg file and employee was not saving.
So to check the correct difference as mentioned here:-
1) Set connection.autocommit to true. Default is false.
2) use session.flush(); not just session.close();
Reply

MANGALA says
SEPTEMBER 3, 2014 AT 10:45 AM
Hi ,
when I try to save an object using session.save(object) without a transaction it never gets reflected in the DB .Even with session.flush() It doesn't work.So I found your statement confusing
"We can invoke this method outside a transaction, that's why I don't like this method to save data. If we use this without transaction and we have cascading between entities, then only the primary entity gets saved unless we flush the session."
Could you please explain?
Thank You!
Reply

Sriram says
JUNE 8, 2016 AT 1:45 AM
use session.close() along with session.flush() when saving outside a transaction context,to see updates in the database.
Reply

Deepak Hassani says
JULY 18, 2014 AT 3:00 AM
Hi, Thanks for the article.
I have one doubt, in the HibernateSaveExample.java, the line :
emp.setName("New Name1"); // will not get updated in database
Should it be : emp6.setName("New Name1"); // will not get updated in database?
Reply

Pankaj says
JULY 18, 2014 AT 12:51 PM
Thanks for the typo, i have corrected the post and some explanations too.
Reply

Saurabh says
MARCH 26, 2016 AT 1:17 PM
Everytime during load() method select query is getting fired but i think that it will not get fired because it just give proxy
As you have shown in merge() method.
Are save() and saveOrUpdate() both are putting/storing data into persistence context ? By reading above article I understand whatever is being stored into persistence context is closely monitored by hibernate till txn.commit/session.flush to reflect remaining changes into DB.
In save() you said "save() puts/stores entity object to persistent context, if you will update the object properties after the save call but before the transaction is committed, it will be saved into database."
On contrary in saveOrUpdate() description you said "One important difference between save and saveOrUpdate is that it adds the entity object to persistent context and track any further changes. Any further changes are saved at the time of committing transaction."
Which one is correct ??
Reply

Khomeni says
JULY 15, 2014 AT 4:39 PM
Thanks Boss.
Reply

Ankit says
SEPTEMBER 8, 2014 AT 11:37 PM
As in your post ,difference between save and persist is not helpful example.because i tried to save the data into database using save but it doesn't get saved in db.until i use transaction.
Reply

Ankit says
SEPTEMBER 9, 2014 AT 5:36 AM
Hi Pankaj,
I found with testing there is no difference between save and persist method of session for persistence context.The save also treat as like persist in persistence context behavior.The only difference between them is save return generated id but persist doesn't return because the return type of persist is void.Or one more important thing is you can't save the mapped entity with session.flush().You have to use transaction to save the entity with mapped entity.
Or if you have concern about this topic.please comment on this.
Reply

James says
AUGUST 19, 2016 AT 5:46 AM
save works for detached instance but persist doesn't.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

HIBERNATE FRAMEWORK

Hibernate Tutorial

- Hibernate Example
- Hibernate SessionFactory
- Hibernate Session get load
- Hibernate Session save
- HQL Example
- Hibernate Criteria
- Hibernate SQL
- Hibernate Named Query
- Hibernate Log4J

RECOMMENDED TUTORIALS

- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial