Java-Success.com

Industrial strength Java Career Companion

Register | Login | Logout | Contact Us

search here … Go

Home | Java FAQs | 600+ Java Q&As | Career | Tutorials | Member | Why?

Can u Debug? | Java 8 ready? | Top X | Productivity Tools | Judging Experience?

Home › Interview › Spring, Hibernate, & Maven Interview Q&A › Hibernate › 08: Hibernate automatic dirty checking of persistent objects and handling detached objects

# 08: Hibernate automatic dirty checking of persistent objects and handling detached objects

Posted on December 22, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

**Q1.** What do you understand by automatic dirty checking in Hibernate?
**A1. Dirty checking** is a feature of hibernate that saves time and effort to update the database when states of objects are modified inside a transaction. All persistent objects are monitored by hibernate.It detects which objects have been modified and then calls update statements on all updated objects.

Hibernate Session contains a ***PersistenceContext*** object that maintains a cache of all the objects read from the database as a Map. So, when you modify an object within the same session, Hibernate compares the objects and triggers the updates when the session is flushed. The objects that are in the PersistenceContext are persistent objects.

**Q2.** How do you perform dirty checks for detached objects?
**A2.** When the session is closed, the ***PersistenceContext*** is lost and so is the cached copy, and the persistent object becomes a detached object. Detached objects can be passed all the way to the presentation layer. When you reattach a detached object through *merge( ), update( ),* or *saveOrUpdate( )* methods, a new session is created with an empty PersistenceContext, hence there is nothing to compare against to perform the dirty check. Here is how you overcome this scenario with the help of the following annotation **selectBeforeUpdate = true**, by default it is set to false.

To save the change to a detached object, you do something like

```
employee.setLastname("Smith");                    // mo
Session sess = sessionFactory.getSession(); // op
Transaction tx = sess.beginTransaction();   //beg
sess.update(employee);                            //The
employee.setFirstName("John");                   //mod
tx.commit();

```

When the update() call is made, hibernate issues an SQL UPDATE statement. This happens irrespective of weather the object has changed after detaching or not. One way to avoid this UPDATE statement while reattaching is by setting the select-before-update= “true”. If this is set, hibernates tries to determine if the UPDATE needed by executing the SELECT statement.

```
@Entity
@org.hibernate.annotations.Entity(selectBeforeUpd
@Table(name = "tbl_employee")
public class Employee extends MyAppDomainObject i
    .....
}

```

## 16 Technical Key Areas

## 80+ step by step Java Tutorials

In rare scenarios, where you are confident that a particular object can never be modified, hence you can tell hibernate that an UPDATE statement will never be needed by setting the following annotation

```
@Entity
@org.hibernate.annotations.Entity(mutable = false
@Table(name = "tbl_employee")
public class Employee extends MyAppDomainObject i
     .....
}

```

Alternatively, if you want to decide if an object's state has changed or not and then decide if a redundant update call to be made or not, you can implement your own ***DirtyCheckInterceptor*** by either implementing Hibenate's *Interceptor* interface or extending the *EmptyInterceptor* class. The interceptor can be bootstrapped as shown below

```
SessionFactory.openSession( new DirtyCheckInterce

```

and, Hibernate's ***FlushEntityEventListener's*** onFlushEntity implementation calls the registered interceptor before making an update call.

Another possible approach would be to clone the detached objects and store it somewhere in a HttpSession and then use that to populate the PersistenceContext when reattaching the object. For example,

```
Session sess = sessionFactory.getSession();
PersistenceContext persistenceContext = session i

if (persistenceContext != null) {
  addPreviouslyStoredEntitiesToPersistenceContext
}

```

**Q3.** What do you understand by the terms **optimistic locking** versus **pessimistic locking**?
**A3. Optimistic locking** means a specific record in the database table is open for all users/sessions. Optimistic

Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

## 100+ Java pre-interview coding tests

open all | close all
Can you write code? (
♦ Complete the given
Converting from A to I
Designing your classe
Java Data Structures
Passing the unit tests
What is wrong with th
Writing Code Home A
Written Test Core Jav
Written Test JEE (1)

## How good are your .....?

open all | close all
Career Making Know-
Job Hunting & Resum

locking uses a strategy where you read a record, make a note of the version number and check that the version number hasn't changed before you write the record back. When you write the record back, you filter the update on the version to make sure that it hasn't been updated between when you check the version and write the record to the disk. If the record is dirty (i.e. different version to yours) you abort the transaction and the user can re-start it.

You could also use other strategies like checking  the timestamp or all the modified fields (this is useful for legacy tables that don't have version number or timestamp column). **Note**: The strategy to compare version numbers and timestamp will work well with detached hibernate objects as well. Hibernate will  automatically manage the version numbers.

In Hibernate, you can use either long number or Date for versioning

```
@Version
private long id;

```

or

```
@Version
private Date version;

```

and mark

```
@Entity
@org.hibernate.annotations.Entity(selectBeforeUpd
@Table(name = "tbl_employee")
public class Employee extends MyAppDomainObject i
    .....
}

```

if you have a legacy table that does not have a version or timestamp column, then use either

```
@Entity
@org.hibernate.annotations.Entity(selectBeforeUpd
@Table(name = "tbl_employee")
public class Employee extends MyAppDomainObject i
    .....
}

```

for all fields and

```
@Entity
@org.hibernate.annotations.Entity(selectBeforeUpd
@Table(name = "tbl_employee")
public class Employee extends MyAppDomainObject i
    .....
}

```

for dirty fields only.

**Pessimistic locking** means a specific record in the database table is open for read/write only for that current session. The other session users can not edit the same because you lock the record for your exclusive use until you have finished with it. It has much better integrity than optimistic locking, but requires you to be careful with your application design to avoid deadlocks. In pessimistic locking, appropriate transaction isolation levels need to be set, so that the records can be locked at different levels. The general isolation levels are

- Read uncommitted isolation
- **Read committed isolation**
- **Repeatable read isolation**
- Serializable isolation

It can be dangerous to use “read uncommitted isolation” as it uses one transaction’s uncommitted changes in a different transaction. The “Serializable isolation” is used to protect phantom reads, phantom reads are not usually problematic, and this isolation level tends to scale very poorly. So, if you are using pessimistic locking, then read commited and repeatable reads are the most common ones.

## Popular Member Posts

♦ 11 Spring boot interview questions & answers

**904 views**

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

**816 views**

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

**427 views**

18 Java scenarios based interview Questions and Answers

**408 views**

♦ 7 Java debugging interview questions & answers

**324 views**

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

**311 views**

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

**304 views**

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

**301 views**

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

**251 views**

001B: ♦ Java architecture & design concepts interview questions & answers

**209 views**

0 Like Share submit reddit 1 G+1 Share

Bio | **Latest Posts**

**Arulkumaran Kumaraswamipillai**

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

**About** Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Top 5 reasons why a resume fails to generate a response

06: Hibernate First & second level cache interview questions and answers ›

**Posted in** Hibernate, member-paid

## Leave a Reply

Logged in as geethika. Log out?

**Comment**

Post Comment

## Empowers you to open more doors, and fast-track

### Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

### Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

## Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

## © Disclaimer