**Hibernate interceptor example – audit log**

Hibernate has a powerful feature called ‘**interceptor**‘ to intercept or hook different kind of Hibernate events, like database CRUD operation. In this article, i will demonstrate how to implement an application audit log feature by using Hibernate interceptor, it will log all the Hibernate save, update or delete operations into a database table named ‘**auditlog**‘.

**Hibernate interceptor example – audit log**

**1. Create a table**

Create a table called ‘auditlog’ to store all the application audited records.

DROP TABLE IF EXISTS `auditlog`;

CREATE TABLE `auditlog` (

`AUDIT\_LOG\_ID` bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

`ACTION` varchar(100) NOT NULL,

`DETAIL` text NOT NULL,

`CREATED\_DATE` date NOT NULL,

`ENTITY\_ID` bigint(20) unsigned NOT NULL,

`ENTITY\_NAME` varchar(255) NOT NULL,

PRIMARY KEY (`AUDIT\_LOG\_ID`)

)

**2. Create a marker interface**

Create a marker interface, any classes which implemented this interface will be audit. This interface requires that the implemented class to expose it identifier – **getId()** and the content to log – ‘**getLogDeatil()**‘. All exposed data will be store into database.

**public** **interface** IAuditLog {

**public** Long getId();

**public** String getLogDeatil();

}

**3. Map the ‘auditlog’ table**

A normal annotation model file to map with table ‘auditlog’.

@Entity

@Table(name = "auditlog”)

**public** **class** AuditLog **implements** java.io.Serializable {

**private** Long auditLogId;

**private** String action;

**private** String detail;

**private** Date createdDate;

**private** **long** entityId;

**private** String entityName;

}

**4. A class implemented the IAuditLog**

A normal annotation model file to map with table ‘stock’, which will use for interceptor demo later. It have to implemented the **IAuditLog** marker interface and implement the **getId()** and **getLogDeatil()** method.

@Entity

@Table(name = "stock", catalog = "mkyong"

**public** **class** Stock **implements** java.io.Serializable, IAuditLog {

@Transient

@Override

**public** Long getId(){

**return** **this**.stockId.longValue();

}

@Transient

@Override

**public** String getLogDeatil(){

StringBuilder sb = **new** StringBuilder();

sb.append(" Stock Id : ").append(stockId)

.append(" Stock Code : ").append(stockCode)

.append(" Stock Name : ").append(stockName);

**return** sb.toString();

}

}

**5. Create a Helper class**

A helper class to accept the data from interceptor and store it into database.

**public** **class** AuditLogUtil {

**public** **static** **void** LogIt(String action, IAuditLog entity, Connection conn) {

Session tempSession = HibernateUtil.getSessionFactory().openSession(

conn);

**try** {

AuditLog auditRecord = **new** AuditLog(action, entity.getLogDeatil(),

**new** Date(), entity.getId(), entity.getClass().toString());

tempSession.save(auditRecord);

tempSession.flush();

} **finally** {

tempSession.close();

}

}

}

**6. Create a Hibernate interceptor class**

Create a interceptor class by extends the Hibernate **EmptyInterceptor**. Here is the most popular interceptor function.

* onSave – Called when you save an object, the object is not save into database yet.
* onFlushDirty – Called when you update an object, the object is not update into database yet.
* onDelete – Called when you delete an object, the object is not delete into database yet.
* preFlush – Called before the saved, updated or deleted objects are committed to database (usually before postFlush).
* postFlush – Called after the saved, updated or deleted objects are committed to database.

The code is quite verbose, it should self-exploratory.

**public** **class** AuditLogInterceptor **extends** EmptyInterceptor {

Session session;

**private** Set inserts = **new** HashSet();

**private** Set updates = **new** HashSet();

**private** Set deletes = **new** HashSet();

**public** **void** setSession(Session session) {

**this**.session = session;

}

**public** **boolean** onSave(Object entity, Serializable id, Object[] state,

String[] propertyNames, Type[] types) **throws** CallbackException {

System.***out***.println("onSave");

**if** (entity **instanceof** IAuditLog) {

inserts.add(entity);

}

**return** **false**;

}

**public** **boolean** onFlushDirty(Object entity, Serializable id,

Object[] currentState, Object[] previousState,

String[] propertyNames, Type[] types) **throws** CallbackException {

System.***out***.println("onFlushDirty");

**if** (entity **instanceof** IAuditLog) {

updates.add(entity);

}

**return** **false**;

}

**public** **void** onDelete(Object entity, Serializable id, Object[] state,

String[] propertyNames, Type[] types) {

System.***out***.println("onDelete");

**if** (entity **instanceof** IAuditLog) {

deletes.add(entity);

}

}

// called before commit into database

**public** **void** preFlush(Iterator iterator) {

System.***out***.println("preFlush");

}

// called after committed into database

**public** **void** postFlush(Iterator iterator) {

System.***out***.println("postFlush");

**try** {

**for** (Iterator it = inserts.iterator(); it.hasNext();) {

IAuditLog entity = (IAuditLog) it.next();

System.***out***.println("postFlush - insert");

AuditLogUtil.LogIt("Saved", entity, session.connection());

}

**for** (Iterator it = updates.iterator(); it.hasNext();) {

IAuditLog entity = (IAuditLog) it.next();

System.***out***.println("postFlush - update");

AuditLogUtil.LogIt("Updated", entity, session.connection());

}

**for** (Iterator it = deletes.iterator(); it.hasNext();) {

IAuditLog entity = (IAuditLog) it.next();

System.***out***.println("postFlush - delete");

AuditLogUtil.LogIt("Deleted", entity, session.connection());

}

} **finally** {

inserts.clear();

updates.clear();

deletes.clear();

}

}

}

**7.Enabling the interceptor**

You can enable the interceptor by pass it as an argument to **openSession(interceptor);**.

Session session = **null**;

Transaction tx = **null**;

**try** {

AuditLogInterceptor interceptor = **new** AuditLogInterceptor();

session = HibernateUtil.getSessionFactory().openSession(interceptor);

interceptor.setSession(session);

//test insert

tx = session.beginTransaction();

Stock stockInsert = **new** Stock();

stockInsert.setStockCode("1111");

stockInsert.setStockName("mkyong");

session.saveOrUpdate(stockInsert);

tx.commit();

//test update

tx = session.beginTransaction();

Query query = session.createQuery("from Stock where stockCode = '1111'");

Stock stockUpdate = (Stock)query.list().get(0);

stockUpdate.setStockName("mkyong-update");

session.saveOrUpdate(stockUpdate);

tx.commit();

//test delete

tx = session.beginTransaction();

session.delete(stockUpdate);

tx.commit();

} **catch** (RuntimeException e) {

**try** {

tx.rollback();

} **catch** (RuntimeException rbe) {

// log.error("Couldn’t roll back transaction", rbe);

}

**throw** e;

} **finally** {

**if** (session != **null**) {

session.close();

}

}

In insert test

session.saveOrUpdate(stockInsert); //it will call onSave

tx.commit(); // it will call preFlush follow by postFlush

In update test

session.saveOrUpdate(stockUpdate); //it will call onFlushDirty

tx.commit(); // it will call preFlush follow by postFlush

In delete test

session.delete(stockUpdate); //it will call onDelete

tx.commit(); // it will call preFlush follow by postFlush

##### **Output**

onSave

Hibernate:

insert into stock

(STOCK\_CODE, STOCK\_NAME)

values (?, ?)

preFlush

postFlush

postFlush - insert

Hibernate:

insert into auditlog

(ACTION, CREATED\_DATE, DETAIL, ENTITY\_ID, ENTITY\_NAME)

values (?, ?, ?, ?, ?)

preFlush

Hibernate:

select ...

from stock stock0\_

where stock0\_.STOCK\_CODE='1111'

preFlush

onFlushDirty

Hibernate:

update stock

set STOCK\_CODE=?, STOCK\_NAME=?

where STOCK\_ID=?

postFlush

postFlush - update

Hibernate:

insert into auditlog

(ACTION, CREATED\_DATE, DETAIL, ENTITY\_ID, ENTITY\_NAME)

values (?, ?, ?, ?, ?)

onDelete

preFlush

Hibernate:

delete from stock where STOCK\_ID=?

postFlush

postFlush - delete

Hibernate:

insert into auditlog

(ACTION, CREATED\_DATE, DETAIL, ENTITY\_ID, ENTITY\_NAME)

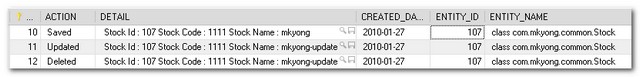
values (?, ?, ?, ?, ?)

##### **In database**

SQL

SELECT \* FROM auditlog a;

All audited data are inserted into database.



## Conclusion

The audit logs is a useful feature that is often handled in database by using triggers, but i would recommend to use application to implement it for the portability concern.