Hibernate Batch Processing

**Maven Configuration(pom.xml)**

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>hibernate-batch</groupId>

<artifactId>hibernate-batch</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-hibernate</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spring.version>4.1.2.RELEASE</spring.version>

<spring.security.version>3.2.3.RELEASE</spring.security.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.30</version>

</dependency>

<!-- C3P0 library -->

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.5</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.7.Final</version>

</dependency>

</dependencies>

</project>

**Spring-Hibernate Configuration(app-context.xml)**

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:jee=*"http://www.springframework.org/schema/jee"*

xmlns:tx=*"http://www.springframework.org/schema/tx"* xmlns:jdbc=*"http://www.springframework.org/schema/jdbc"*

xmlns:osgi=*"http://www.springframework.org/schema/osgi"* xmlns:security=*"http://www.springframework.org/schema/security"*

xsi:schemaLocation=*"http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*

*http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd*

*http://www.springframework.org/schema/osgi http://www.springframework.org/schema/osgi/spring-osgi.xsd*

*http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.0.3.xsd"*>

<tx:annotation-driven transaction-manager=*"discussionTransactionManager"* />

<bean id=*"dataSourceInternal"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*

destroy-method=*"close"*>

<property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost/test"* />

<property name=*"user"* value=*"deba"* />

<property name=*"password"* value=*"deba"* />

<!-- these are C3P0 properties -->

<property name=*"acquireIncrement"* value=*"5"* />

<property name=*"initialPoolSize"* value=*"5"* />

<property name=*"minPoolSize"* value=*"5"* />

<property name=*"maxPoolSize"* value=*"20"* />

<!-- <property name="maxIdleTime" value="${c3p0.maxIdleTime}" />

<property name="checkoutTimeout" value="${c3p0.checkoutTimeout}" />

<property name="preferredTestQuery" value="${c3p0.preferredTestQuery}" />

<property name="idleConnectionTestPeriod" value="${c3p0.idleConnectionTestPeriod}" />

<property name="maxIdleTimeExcessConnections" value="${c3p0.maxIdleTimeExcessConnections}" />

<property name="numHelperThreads" value="${c3p0.numHelperThreads}" />

<property name="unreturnedConnectionTimeout" value="${c3p0.unreturnedConnectionTimeout}" />

<property name="debugUnreturnedConnectionStackTraces" value="${c3p0.debugUnreturnedConnectionStackTraces}" />

<property name="testConnectionOnCheckin" value="${c3p0.testConnectionOnCheckin}" />

<property name="acquireRetryAttempts" value="${c3p0.acquireRetryAttempts}" />

<property name="acquireRetryDelay" value="${c3p0.acquireRetryDelay}" />

<property name="breakAfterAcquireFailure" value="${c3p0.breakAfterAcquireFailure}" /> -->

</bean>

<!-- This is the lazy DataSource proxy that interacts with the target DataSource once a real statement is sent to the database. Users use this DataSource to set up their Hibernate session factory, which in turn forces the Hibernate second-level cache and also everything that interacts with that Hibernate session factory to use it. -->

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.LazyConnectionDataSourceProxy"*>

<property name=*"targetDataSource"*><ref bean=*"dataSourceInternal"* /></property>

</bean>

<!-- <bean id="hibSessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"> -->

<bean id=*"hibSessionFactory"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"* />

<!--<property name="hibernateProperties"> <value> hibernate.show\_sql=true

</value> </property> -->

<property name=*"hibernateProperties"*>

<value>

hibernate.id.new\_generator\_mappings=true

<!-- hibernate.hbm2ddl.auto=update -->

hibernate.show\_sql=true

hibernate.format\_sql

<!-- hibernate.current\_session\_context\_class=thread -->

</value>

</property>

<property name=*"annotatedClasses"*>

<list>

<value>com.ddlab.rnd.orm.Employee</value>

</list>

</property>

</bean>

</beans>

**Java Source Code**

**Employee.java**

**package** com.ddlab.rnd.orm;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "selfjoinemp")

**public** **class** Employee {

@Id

@GeneratedValue(strategy=GenerationType.***IDENTITY***)

@Column(name = "empId")

**private** **long** empId;

@Column(name = "firstname")

**private** String firstName;

@Column(name = "city")

**private** String city;

**public** Employee() {

}

**public** Employee(String firstName, String city) {

**this**.firstName = firstName;

**this**.city = city;

}

**public** **long** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**long** empId) {

**this**.empId = empId;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

**TestBatch.java**

**package** com.ddlab.rnd.orm;

**import** org.hibernate.HibernateException;

**import** org.hibernate.ScrollableResults;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** TestBatch {

**public** **static** **void** bulkInsert(Session session) {

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

**for** (**int** i = 0; i < 3000; i++) {

Employee employee = **new** Employee("Name-" + i, "City-" + i);

session.save(employee);

// Otherwise it will throw OutOfMemory Error

**if** (i % 1000 == 0) {

// flush a batch of inserts and release memory:

session.flush();

session.clear();

}

}

transaction.commit();

} **catch** (HibernateException e) {

transaction.rollback();

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **static** **void** bulkUpdate(Session session) {

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

ScrollableResults empResults = session.createQuery("FROM Employee")

.scroll();

**int** count = 0;

**while** (empResults.next()) {

Employee employee = (Employee) empResults.get(0);

employee.setCity("My City-" + count);

count++;

session.update(employee);

// Otherwise it will throw OutOfMemory Error

**if** (count % 1000 == 0) {

// flush a batch of inserts and release memory:

session.flush();

session.clear();

}

}

transaction.commit();

} **catch** (HibernateException e) {

transaction.rollback();

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **static** **void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"app-context.xml");

SessionFactory sessionFactory = (SessionFactory) context

.getBean("hibSessionFactory");

Session session = sessionFactory.openSession();

*bulkInsert*(session);

*bulkUpdate*(session);

}

}

You can also use Hibernate Interceptor for audi log purposes.

Create an interceptor like the below.

**MyInterceptor.java**

**package** com.ddlab.rnd.orm;

**import** java.io.Serializable;

**import** java.util.Iterator;

**import** org.hibernate.EmptyInterceptor;

**import** org.hibernate.type.Type;

/\*

\* It can be used as an Audit Log Entry

\*/

**public** **class** MyInterceptor **extends** EmptyInterceptor {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

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

String[] propertyNames, Type[] types) {

// log delete events

System.***out***.println("Delete event");

}

// called when a Student gets updated.

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

Object[] currentState, Object[] previousState,

String[] propertyNames, Type[] types) {

**if** (entity **instanceof** Employee) {

System.***out***

.println("===========Employee Update Operation==============");

**return** **true**;

}

**return** **false**;

}

// called on load events

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

String[] propertyNames, Type[] types) {

// log loading events

System.***out***.println("Load Operation");

**return** **true**;

}

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

String[] propertyNames, Type[] types) {

**if** (entity **instanceof** Employee) {

System.***out***

.println("===========Employee Create Operation===========");

**return** **true**;

}

**return** **false**;

}

// called before commit into database

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

System.***out***.println("Before commiting");

}

// called after committed into database

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

System.***out***.println("After commiting");

}

}

**TestApp.java (For testing interceptor)**

**package** com.ddlab.rnd.orm;

**import** org.hibernate.HibernateException;

**import** org.hibernate.ScrollableResults;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App

{

**public** **static** **void** bulkInsert(Session session) {

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

**for** ( **int** i=0; i<3000; i++ ) {

Employee employee = **new** Employee("Name-"+i, "City-"+i);

session.save(employee);

//Otherwise it will throw OutOfMemory Error

**if**( i % 1000 == 0 ) {

//flush a batch of inserts and release memory:

session.flush();

session.clear();

}

}

transaction.commit();

} **catch** (HibernateException e) {

transaction.rollback();

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **static** **void** bulkUpdate(Session session) {

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

ScrollableResults empResults = session.createQuery("FROM Employee").scroll();

**int** count = 0;

**while** ( empResults.next() ) {

Employee employee = (Employee) empResults.get(0);

employee.setCity("My City-"+count);

count++;

session.update(employee);

//Otherwise it will throw OutOfMemory Error

**if**( count % 1000 == 0 ) {

//flush a batch of inserts and release memory:

session.flush();

session.clear();

}

}

transaction.commit();

} **catch** (HibernateException e) {

transaction.rollback();

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **static** **void** main( String[] args )

{

ApplicationContext context = **new** ClassPathXmlApplicationContext(

"app-context.xml");

SessionFactory sessionFactory = (SessionFactory) context

.getBean("hibSessionFactory");

//In case of Hibernate 4

MyInterceptor intercepter = **new** MyInterceptor();

Session session = sessionFactory.withOptions().interceptor(intercepter).openSession();

// bulkInsert(session);

*bulkUpdate*(session);

}

}