Пакетная обработка больших объемов

**Содержание**

1. [Пакетная вставка](http://hibernate-refdoc.3141.ru/ch15.BatchProcessing#p15.1)
2. [Пакетное обновление](http://hibernate-refdoc.3141.ru/ch15.BatchProcessing#p15.2)
3. [Интерфейс StatelessSession](http://hibernate-refdoc.3141.ru/ch15.BatchProcessing#p15.3)
4. [Операции в DML-стиле](http://hibernate-refdoc.3141.ru/ch15.BatchProcessing#p15.4)

Неправильный подход к вставке 100 000 строк в базу данных с использованием Hibernate может выглядеть так:

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {

Customer customer = new Customer(.....);

session.save(customer);

}

tx.commit();

session.close();

Сие упадёт с OutOfMemoryException где-то около 50 000-й строки. Это связано с тем, что Hibernate кэширует все вновь вставленные экземпляры клиента в кэше 1 уровня сессии. В этой главе мы покажем вам, как избежать этой проблемы.

Если вы выполняете пакетную обработку, вам необходимо включить использование пакетной обработки JDBC. Это абсолютно необходимо, если вы хотите достичь оптимальной производительности. Установите размер пакета JDBC на разумное число (например, 10-50):

hibernate.jdbc.batch\_size 20

Hibernate отключает пакетную вставку на уровне JDBC прозрачно, если вы используете генератор идентификаторов identity.

Вы также можете выполнять такую работу в процессе, когда взаимодействие с кэшем второго уровня полностью отключено:

hibernate.cache.use\_second\_level\_cache false

Однако это не является абсолютно необходимым, поскольку мы можем явно установить CacheMode для отключения взаимодействия с кэшем второго уровня.

15.1. Пакетная вставка

Когда делаете новые объекты постоянными, вызывайте flush(), а затем clear() регулярно, чтобы контролировать размер кеша первого уровня.

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

for ( int i=0; i<100000; i++ ) {

Customer customer = new Customer(.....);

session.save(customer);

if ( i % 20 == 0 ) { //20, такой же размер, как в JDBC

//сбросить пакет вставок и освободить память:

session.flush();

session.clear();

}

}

tx.commit();

session.close();

15.2. Пакетное обновление

Для получения и обновления данных применяются те же идеи. Кроме того, вам нужно использовать scroll(), чтобы получить преимущество от использования серверных курсоров запросов, которые возвращают много записей.

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")

.setCacheMode(CacheMode.IGNORE)

.scroll(ScrollMode.FORWARD\_ONLY);

int count=0;

while ( customers.next() ) {

Customer customer = (Customer) customers.get(0);

customer.updateStuff(...);

if ( ++count % 20 == 0 ) {

//сбросить пакет обновлений и освободить память:

session.flush();

session.clear();

}

}

tx.commit();

session.close();

15.3. Интерфейс StatelessSession

В качестве альтернативы Hibernate предоставляет командный API, который может использоваться для потоковой передачи данных в базу данных и из неё в виде отдельных объектов. StatelessSession не имеет контекста персистентности и не обеспечивает много семантик жизненного цикла более высокого уровня. В частности, сессия без состояния не реализует кэш первого уровня и не взаимодействует с кэшем второго уровня или запроса. Он не выполняет транзакционную запись write-behind или автоматическую «проверку». Операции, выполненные с использованием сессии без состояния, никогда не каскадируются на связанные экземпляры. Коллекции игнорируются сессией без состояния. Операции, выполняемые с помощью сессий без состояния обходят (bypass) модель событий и перехватчиков Hibernate. Из-за отсутствия кэша первого уровня сессии без состояния уязвимы для эффектов алиасинга (aliasing) данных. Сессия без состояния — это абстракция более низкого уровня, которая намного ближе к низлежащему JDBC.

**StatelessSession** session = sessionFactory.**openStatelessSession()**;

Transaction tx = session.beginTransaction();

ScrollableResults customers = session.getNamedQuery("GetCustomers")

.scroll(ScrollMode.FORWARD\_ONLY);

while ( customers.next() ) {

Customer customer = (Customer) customers.get(0);

customer.updateStuff(...);

session.update(customer);

}

tx.commit();

session.close();

В этом примере кода экземпляры Customer, возвращаемые запросом, немедленно отделяются. Они никогда не связаны с каким-либо контекстом постоянства (persistence).

Операции insert(), update() и delete(), определённые интерфейсом StatelessSession, рассматриваются как прямые операции уровня записи (row-level) базы данных. Они приводят к немедленному выполнению SQL INSERT, UPDATE или DELETE соответственно. Они имеют разную семантику операций save(), saveOrUpdate() и delete(), определённых интерфейсом сессии.

15.4. Операции в DML-стиле

Как уже обсуждалось, автоматическое и прозрачное объектно-реляционное отображение связано с управлением состоянием объекта. Состояние объекта доступно в памяти. Это означает, что манипулирование данными непосредственно в базе данных (с использованием языка манипулирования данными SQL (DML): инструкции INSERT, UPDATE или DELETE) не влияют на состояние в памяти. Тем не менее, Hibernate предоставляет методы для массового выполнения инструкций DML SQL-стиля, которые выполняются с помощью языка запросов Hibernate ([глава 16. HQL: язык запросов Hibernate](http://hibernate-refdoc.3141.ru/ch16.HQL)).

Псевдо-синтаксис для операторов UPDATE и DELETE: (UPDATE |  DELETE) FROM ? имя\_сущности (WHERE where-условие) ?.

Некоторые моменты:

* В секции from ключевое слово FROM является необязательным
* В секции from может быть только одно имя сущности. Однако оно может быть псевдонимом. Если имя сущности имеет псевдоним, любые ссылки на свойства должны быть квалифицированы с использованием этого псевдонима. Если имя сущности не является псевдонимом, то для любых ссылок на свойства оно является незаконным.
* Ни одна из [форм синтаксиса объединения](http://hibernate-refdoc.3141.ru/ch16.HQL#p16.4), явная или неявная, не моет быть указана в объёмном запросе HQL. Подзапросы могут использоваться в условии where-секции, где сами подзапросы могут содержать объединения.
* Секция where также необязательно.

В качестве примера, чтобы выполнить HQL UPDATE, используйте метод Query.executeUpdate(). Этот метод назван так для тех, кто знаком с JDBC PreparedStatement.executeUpdate():

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

String hqlUpdate = "update Customer c set c.name = :newName where c.name = :oldName";

// или String hqlUpdate = "update Customer set name = :newName where name = :oldName";

int updatedEntities = s.createQuery( hqlUpdate )

.setString( "newName", newName )

.setString( "oldName", oldName )

.executeUpdate();

tx.commit();

session.close();

В соответствии со спецификацией EJB3 операторы HQL UPDATE по умолчанию не влияют на [номер версии](http://hibernate-refdoc.3141.ru/ch5.BasicORMapping#p5.1.3.1) или на свойство [Timestamp](http://hibernate-refdoc.3141.ru/ch5.BasicORMapping" \l "p5.1.3.2) для затронутых сущностей. Тем не менее, вы можете заставить Hibernate сбросить значения свойств version или timestamp с помощью versioned update. Это достигается добавлением ключевого слова VERSIONED после ключевого слова UPDATE.

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

String hqlVersionedUpdate = "update versioned Customer set name = :newName where name = :oldName";

int updatedEntities = s.createQuery( hqlUpdate )

.setString( "newName", newName )

.setString( "oldName", oldName )

.executeUpdate();

tx.commit();

session.close();

Пользовательские типы версий, org.hibernate.usertype.UserVersionType, не допускаются в сочетании с инструкцией update versioned.

Чтобы выполнить HQL DELETE, используйте тот же метод Query.executeUpdate():

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

String hqlDelete = "delete Customer c where c.name = :oldName";

// или String hqlDelete = "delete Customer where name = :oldName";

int deletedEntities = s.createQuery( hqlDelete )

.setString( "oldName", oldName )

.executeUpdate();

tx.commit();

session.close();

Значение int, возвращаемое методом Query.executeUpdate(), указывает количество сущностей, выполняемых операцией. Это может или не может коррелировать с количеством записей, созданных в базе данных. Операция большого объема HQL может привести к выполнению нескольких действительных операторов SQL (например, для объединённого подкласса (joined-subclass)). Возвращаемое число указывает количество фактических сущностей, на которые влияет инструкция. Возвращаясь к примеру объединённого подкласса, удаление в отношении одного из подклассов может фактически привести к удалению не только таблицы, на которую этот подкласс отображён, но также «корневой» таблицы и потенциально подключенных подклассов далее вниз по иерархии наследования.

Псевдо-синтаксис для операторов INSERT: INSERT INTO имя\_сущности список\_свойств select-инструкция. Некоторые моменты:

* Поддерживается только форма INSERT INTO ... SELECT ...; а не INSERT INTO ... VALUES ... .

Свойство «список\_свойств» аналогично спецификации столбца в инструкции SQL INSERT. Для сущностей, участвующих в отображённом наследовании, только свойства, непосредственно определённые на данном заданном уровне класса, могут использоваться в «списке\_свойств». Свойства суперкласса недопустимы, а свойства подкласса не имеют смысла. Другими словами, инструкции INSERT по своей сути являются неполиморфными.

* «select-инструкция» может быть любым допустимым запросом select HQL, с оговоркой, что возвращаемые типы должны соответствовать типам, ожидаемым вставкой. В настоящее время это проверяется во время компиляции запросов. Это может, однако, вызвать проблемы между типами TypeHibernate, которые *эквивалентны*, а не *равны*. Это может вызвать проблемы с несоответствиями между свойством, определённым как org.hibernate.type.DateType, и свойство, определённым как org.hibernate.type.TimestampType, даже если база данных не может делать различия или может обрабатывать преобразование.
* Для свойства id инструкция insert предоставляет два варианта. Вы можете явно указать свойство id в «списке\_свойств», в этом случае его значение берётся из соответствующего выражения select, или опустить его из «списка\_свойств», и в этом случае используется сгенерированное значение. Этот последний параметр доступен только при использовании генераторов id, работающих в базе данных; попытка использовать эту опцию с любыми генераторами типа «в памяти (in memory)» вызовет исключение во время парсинга. Для целей этого обсуждения генераторы «из базы данных (in-database)» считаются org.hibernate.id.SequenceGenerator (и его подклассы) и любы реализующие org.hibernate.id.PostInsertIdentifierGenerator. Наиболее заметным исключением здесь является org.hibernate.id.TableHiLoGenerator, который нельзя использовать, поскольку он не предоставляет доступный способ для получения его значений.
* Для свойств, отображаемых в version или timestamp, инструкция insert предоставляет два варианта. Вы можете указать свойство в «списке\_свойств», в этом случае его значение берется из соответствующих выражений select, или опустить его из «списка\_свойств» и в этом случае используется начальное значение, определённое с помощью org.hibernate.type.VersionType.

Ниже приведен пример выполнения инструкции HQL INSERT:

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

String hqlInsert = "insert into DelinquentAccount (id, name) select c.id, c.name from Customer c where ...";

int createdEntities = s.createQuery( hqlInsert ).executeUpdate();

tx.commit();

session.close();