ОПТИМИЗАЦИЯ HIBERNATE ДЛЯ ОБРАБОТКИ БОЛЬШОГО ОБЪЕМА ДАННЫХ

Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь

Боркун А.А.

Сиротко С.И., к.ф-м.наук, доцент

В современных условиях при обработке больших данных часто возникают проблемы эффективного использования оперативной памяти необходимой для выполнения web-приложений. Одним из решений является увеличение выделенной оперативной памяти. Однако автором предлагается более эффективный способ, позволяющий достигнуть необходимой результата в рамках существующей конфигурации сервера.

По некоторым оценкам, объем информации в мире увеличивается вдвое каждые десять лет. Для эффективной работы с большими объёмами данных необходимы системы, которые отвечают высоким требованиям промышленного ПО (программного обеспечения). С помощью этих систем пользователи могут находить необходимые данные за минимамальное количество времени.

Для обработки большого объема данных в Java чаще всего используется JDBC(Java DataBase Connectivity - соединение с базами данных на Java, платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными базами данных), чтобы получить более высокую производительность вычислений. Если в приложении уже используется ORM (Object-relational mapping - объектно-реляционное отображение, технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая “виртуальную объектную базу данных”), то переход на использование JDBC может означать дополнительные финансовые вложения и перенос срока окончания разработки проекта. Также будут потеряны такие функции для разработчика, как оптимистический контроль параллельного доступа (optimistic concurrency control), кэширование, конвертация данных БД в Java-объекты, их сохранение, модификация и удаление.

Hibernate - библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (object-relational mapping - ORM). Она представляет собой свободное программное обеспечение с открытым исходным кодом. Данная библиотека предоставляет лёгкий в использовании каркас (фреймворк) для отображения объектно-ориентированной модели данных в традиционные реляционные базы данных.

Типовой задачей является обработка таблицы с 100,000 записей. Каждая запись состоит из одного столбца, содержащего некоторые случайные буквенно-цифровые данные размером 2 KB. Настройки JVM: максимальный размер heap (название структуры данных, с помощью которой реализована динамически распределяемая память приложения, а также объём памяти, зарезервированный под эту структуру) 250 МБ. HQL команда list() для выборки данных при заданных настройках памяти через несколько секунд после запуска происходит exception (исключительная ситуация): “java.lang.OutOfMemoryError”, свидетельствующий о недостаточно выделенной heap-памяти.

При обработке большого объёма данных в Hibernate ORM в рамках исследования автором предлагаются следующие способы оптимизации, позволяющие использовать оперативную память более эффективно:

Способ 1. Использование класса ScrollableResults, который позволяет преобразовывать записи в объекты по одному во время обработки данных в цикле. Использование данного класса аналогично работе с JDBC классом ResultSet. Однако в данном случае используется доменная модель, а не значения полей таблицы базы данных.

Способ 2. Повышение эффективности использования оперативной памяти за счет методов Session.evict(Class entityClass), Cache.evictEntityRegion(Class entityClass). Эти методы позволяют удалить объект из кэша Hibernate после обработки. При необходимости можно вызвать метод Session.clear(), который позволяет удалить все объекты ассоциированные с текущей Hibernate сессией.

Способ 3. Использование методов Query.setReadOnly(true) и Session.setReadOnly(true), которые позволяет Hibernate оптимизировать свою работу в связи с тем, что загруженные сущности из БД будут доступны только для чтения. Таким образом данные настройки ликвидируют возможность внесения каких-либо изменений.

Способ 4. Использование Stateless-сессии. Stateless Hibernate сессия является первой абстракцией над JDBC, в которой отстутствуют следующие функции ORM:

- Persistence Context;
- кэши первого и второго уровня;
- проверка на грязное “чтение” (чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится);
- lazy loading (загрузка объектов после прямого обращения);
- операции каскадного удаления и сохранения сущностей.

В этом режиме Hibernate использует память наиболее эффективно. Открытие и закрытие сессии происходит напрямую при помощи методов openStatelessSession() и close(), так как в этом режиме Hibernate не хранит своё состояние. После каждого открытия Stateless-сессии происходит создание нового соединения с БД.

Способ реализации предлагаемых автором выше решений с точки зрения программного кода представлен на рис. 1.

```
new TransactionTemplate(txManager).execute(new TransactionCallback<Void>() {
 @Override
 public Void doInTransaction(TransactionStatus status) {
  sessionFactory.getCurrentSession().doWork(new Work() {
   @Override
   public void execute(Connection connection) throws SQLException {
    StatelessSession statelessSession = sessionFactory.openStatelessSession(connection);
    try {
     ScrollableResults scrollableResults = statelessSession.createQuery("from
Entity").scroll(ScrollMode.FORWARD_ONLY);

     int count = 0;
     while (scrollableResults.next()) {
      if (++count > 0 && count % 100 == 0) {
       System.out.println("Fetched " + count + " entities");
      }
      Entity demoEntity = (Entity) scrollableResults.get()[0];
      //Process and write result
     }
    } finally {
     statelessSession.close();
    }
   }
  });
  return null;
 }
});
```

Рис.1 – Фрагмент программы по оптимизации Hibernate

Описанный выше вариант программного кода, использующий предложенные автором способы оптимизации позволяет получить следующую экономию оперативной памяти в рамках поставленной задачи (Рис.2).

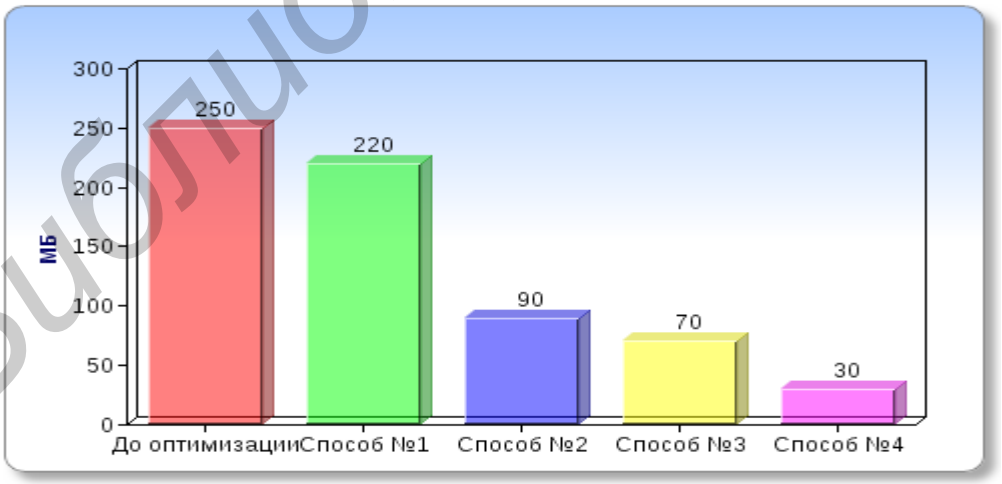

Рис. 2 – Экономия оперативной памяти за счет оптимизации Hibernate

Таким образом, в статье предложены решения оптимизации ORM Hibernate для обработки больших объёмов информации. Рассматриваемые решения позволяют снизить количество памяти, необходимой для работы приложения, использующего Hibernate для доступа к реляционной БД, без необходимости наращивать мощность сервера или переходить к более низким уровням абстракции – таким как JDBC.