Ответы на вопросы на собеседование Object relational mapping (ORM), Hibernate (часть 2).

Vasyl K 7:47:00 1 Комментарий

- **КАК НАСТРАИВАЕТСЯ КЭШ ВТОРОГО УРОВНЯ В HIBERNATE?**

Чтобы указать кэш второго уровня нужно определить hibernate.cache.provider_class в hibernate.cfg.xml:

```
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.cache.provider_class">org.hibernate.cache.EHCacheProvider</property>
    </session-factory>
</hibernate-configuration>
```

По-умолчанию используется EHCache.
Чтобы использвать кэш запросов нужно его включить установив свойство hibernate.cache.use_query_cache в true в hibernate.properties.

- **КАКАЯ РАЗНИЦА В РАБОТЕ МЕТОДОВ LOAD(); И GET();?**

Hibernate session обладает различными методами для загрузки данных из базы данных. Наиболее часто
используемые методы для этого - get() и load().

- Метод load(); обычно используется когда в не уверен что запрашиваемый объект уже находится в базе данных. Если объект не найден, то метод кидает исключение Если объект найден – метод возвращает прокси объект, который является ссылкой на объект находящийся в базе данных (запрос в базу данных еще не был осуществлен, своего рода lazy изъятие), непосредственный запрос к базе данных когда мы непосредственно обращаемся к необходимому объекту через прокси объект.
- Метод get(); используется тогда, вы на 100 процентов не уверены есть ли запрашиваемый объект в базе данных. В случае обращение к несуществующему объекту, метод get(); вернет null. В случае нахождения объект, метод get(); вернет сам объект и запрос в базу данных будет произведен немедленно.

- **КАКОВЫ СУЩЕСТВУЮТ РАЗЛИЧНЫЕ СОСТОЯНИЯ У ENTITY BEAN?**

Transient: состояние, при котором объект никогда не был связан с какой-либо сессией и не является персистентностью. Этот объект находится во временном состоянии. Объект в этом состоянии может стать персистентным при вызове метода save(),

persist() или saveOrUpdate(). Объект персистентности может перейти в transient состоянии после вызова метода delete().
Persistent: когда объект связан с уникальной сессией он находится в состоянии persistent (персистентности). Любой экземпляр, возвращаемый методами get() или load() находится в состоянии persistent.
Detached: если объект был персистентным, но сейчас не связан с какой-либо сессией, то он находится в отвязанном (detached) состоянии. Такой объект можно сделать персистентным используя методы update(), saveOrUpdate(), lock() или replicate(). Состояния transient или detached так же могут перейти в состояние persistent как новый объект персистентности после вызова метода merge().

- **ЧТО ПРОИЗОЙДЕТ, ЕСЛИ БУДЕТ ОТСУТСТВОВАТЬ КОНСТРУКТОР БЕЗ АРГУМЕНТОВ У ENTITY BEAN?**

Hibernate использует рефлексию для создания экземпляров Entity бинов при вызове методов get() или load(). Для этого используется метод Class.newInstance(), который требует наличия конструктора без параметров. Поэтому, в случае его отсутствия, вы получите ошибку HibernateException.

- **КАК ИСПОЛЬЗУЕТСЯ ВЫЗОВ МЕТОДА HIBERNATE SESSION MERGE()?**

Hibernate merge() может быть использован для обновления существующих значений, однако этот метод создает копию из переданного объекта сущности и возвращает его. Возвращаемый объект является частью контекста персистентности и отслеживает любые изменения, а переданный объект не отслеживается.

- **В ЧЕМ РАЗНИЦА МЕЖДУ HIBERNATE SAVE(), SAVEORUPDATE() И PERSIST()?**

Hibernate save() используется для сохранения сущности в базу данных. Проблема с использованием метода save() заключается в том, что он может быть вызван без транзакции. А следовательно если у нас имеется отображение нескольких объектов, то только первичный объект будет сохранен и мы получим несогласованные данные. Также save() немедленно возвращает сгенерированный идентификатор.
Hibernate persist() аналогичен save() с транзакцией. persist() не возвращает сгенерированный идентификатор сразу.
Hibernate saveOrUpdate() использует запрос для вставки или обновления, основываясь на предоставленных данных. Если данные уже присутствуют в базе данных, то будет выполнен запрос обновления. Метод saveOrUpdate() можно применять без транзакции, но это может привести к аналогичным проблемам, как и в случае с методом save().

- **ЧТО ТАКОЕ LAZY FETCHING(ИЗЪЯТИЕ) В HIBERNATE?**

Тип изъятия Lazy, в Hibernate, связан с листовыми(дочерними) сущностями и определяют политику совместного изъятия, если идет запрос на изъятие сущности

родителя.
Простой пример:
Есть сущность Дом. Он хранит информацию о своем номере, улице, количество квартир и информацию о семьях которые живут в квартирах, эти семьи формируют дочернюю сущность относительно сущности Дом. Когда мы запрашиваем информацию о Доме, нам может быть совершенно ненужным знать информацию семьях которые в нем проживают, тут нам на помощь приходит lazy(ленивое) изъятие(fetching) которая позволяет сконфигурировать сущность Дом, чтобы информацию о семьях подавалась только по востребованию, это значительно облегчает запрос и ускоряет работу приложения.

- **В ЧЕМ РАЗНИЦА МЕЖДУ SORTED COLLECTION И ORDERED COLLECTION? КАКАЯ ИЗ НИХ ЛУЧШЕ?**

При использовании алгоритмов сортировки из Collection API для сортировки коллекции, то он вызывает отсортированный список (sorted list). Для маленьких коллекций это не приводит к излишнему расходу ресурсов, но на больших коллекциях это может привести к потери производительности и ошибкам OutOfMemory. Так же entity бины должны реализовывать интерфейс Comparable или Comparator для работы с сортированными коллекциями.
При использовании фреймворка Hibernate для загрузки данных из базы данных мы можем применить Criteria API и команду order by для получения отсортированного списка (ordered list). Ordered list является лучшим выбором к sorted list, т.к. он использует сортировку на уровне базы данных. Она быстрее и не может привести к утечке памяти.

- **КАК РЕАЛИЗОВАНЫ JOIN'Ы HIBERNATE?**

Существует несколько способов реализовать связи в Hibernate.
- Использовать ассоциации, такие как one-to-one, one-to-many, many-to-many.
- Использовать в HQL запросе команду JOIN. Существует другая форма «join fetch«, позволяющая загружать данные немедленно (не lazy).
- Использовать чистый SQL запрос с командой join.

- **ПОЧЕМУ МЫ НЕ ДОЛЖНЫ ДЕЛАТЬ ENTITY CLASS КАК FINAL?**

Хибернейт использует прокси классы для ленивой загрузки данных (т.е. по необходимости, а не сразу). Это достигается с помощью расширения entity bean и, следовательно, если бы он был final, то это было бы невозможно. Ленивая загрузка данных во многих случаях повышает производительность, а следовательно важна.

- **ЧТО ВЫ ЗНАЕТЕ О HQL И КАКОВЫ ЕГО ПРЕИМУЩЕСТВА?**

Hibernate Framework поставляется с мощным объектно-ориентированным языком запросов – Hibernate Query Language (HQL). Он очень похож на SQL, за исключением,

что в нем используются объекты вместо имен таблиц, что делает язык ближе к объектно-ориентированному программированию.
HQL является регистронезависимым, кроме использования в запросах имен java переменных и классов, где он подчиняется правилам Java. Например, SelECt то же самое, что и select, но com.blogspot.jsehelper.MyClass отличен от com.blogspot.jsehelper.MyCLASS. Запросы HQL кэшируются (это как плюс так и минус).

- **ЧТО ТАКОЕ QUERY CACHE В HIBERNATE?**

Hibernate реализует область кэша для запросов resultset, который тесно взаимодействует с кэшем второго уровня Hibernate. Для подключения этой дополнительной функции требуется несколько дополнительных шагов в коде. Query Cache полезны только для часто выполняющихся запросов с повторяющимися параметрами. Для начала необходимо добавить эту запись в файле конфигурации Hibernate:

```
<property name="hibernate.cache.use_query_cache">true</property>
```

Уже внутри кода приложения для запроса применяется метод setCacheable(true), как показано ниже:

```
Query query = session.createQuery("from Employee");
query.setCacheable(true);
query.setCacheRegion("ALL_EMP");
```

- **МОЖЕМ ЛИ МЫ ВЫПОЛНИТЬ SQL (SQL NATIVE) ЗАПРОС В HIBERNATE?**

С помощью использования SQLQuery можно выполнять чистый запрос SQL. В общем случае это не рекомендуется, т.к. вы потеряете все преимущества HQL (ассоциации, кэширование). Выполнить можно примерно так:

```
Transaction tx = session.beginTransaction();
SQLQuery query = session.createSQLQuery("select emp_id, emp_name, emp_salary from Employee");
List<object[]> rows = query.list();
for(Object[] row : rows){
    Employee emp = new Employee();
    emp.setId(Long.parseLong(row[0].toString()));
    emp.setName(row[1].toString());
    emp.setSalary(Double.parseDouble(row[2].toString()));
    System.out.println(emp);
}
```

- **НАЗОВИТЕ ПРЕИМУЩЕСТВА ПОДДЕРЖКИ НАТИВНОГО SQL В HIBERNATE.**

Использование нативного SQL может быть необходимо при выполнении запросов к некоторым базам данных, которые могут не поддерживаться в Hibernate. Примером может служить некоторые специфичные запросы и «фишки» при работе с БД от Oracle.

- **ЧТО ТАКОЕ NAMED SQL QUERY?**

Hibernate поддерживает именованный запрос, который мы можем задать в каком-либо центральном месте и потом использовать его в любом месте в коде. Именованные запросы поддерживают как HQL, так и Native SQL. Создать именованный запрос можно с помощью JPA аннотаций @NamedQuery, @NamedNativeQuery или в конфигурационном файле отображения (mapping files).

- **КАКОВЫ ПРЕИМУЩЕСТВА NAMED SQL QUERY?**

- Именованный запрос Hibernate позволяет собрать множество запросов в одном месте, а затем вызывать их в любом классе.
- Синтаксис Named Query проверяется при создании session factory, что позволяет заметить ошибку на раннем этапе, а не при запущенном приложении и выполнении запроса.
- Named Query глобальные, т.е. заданные однажды, могут быть использованы в любом месте.

Однако одним из основных недостатков именованного запроса является то, что его очень трудно отлаживать (могут быть сложности с поиском места определения запроса).

- **КАК ДОБАВИТЬ ЛОГИРОВАНИЕ LOG4J В HIBERNATE ПРИЛОЖЕНИЕ?**

Добавить зависимость log4j в проект.
Создать log4j.xml или log4j.properties файл и добавить его в classpath.
Для веб приложений используйте ServletContextListener, а для автономных приложений DOMConfigurator или PropertyConfigurator для настройки логирования.
Создайте экземпляр org.apache.log4j.Logger и используйте его согласно задачи.

- **КАК ЛОГИРОВАТЬ СОЗДАННЫЕ HIBERNATE SQL ЗАПРОСЫ В ЛОГ-ФАЙЛЫ?**

Для логирования запросов SQL добавьте в файл конфигурации Hibernate строчку:

```
1 | <property name="hibernate.show_sql">true</property>
```

- **ЧТО ВЫ ЗНАЕТЕ О HIBERNATE ПРОКСИ И КАК ЭТО ПОМОГАЕТ В ЛЕНИВОЙ ЗАГРУЗКЕ (LAZY LOAD)?**

Hibernate использует прокси объект для поддержки отложенной загрузки. Обычно при загрузке данных из таблицы Hibernate не загружает все отображенные (замаппинные) объекты. Как только вы ссылаетесь на дочерний объект или ищите объект с помощью геттера, если связанная сущность не находиться в кэше сессии, то прокси код перейдет к базе данных для загрузки связанной сущности. Для этого используется javassist, чтобы эффективно и динамически создавать реализации подклассов ваших entity объектов.

- **КАК УПРАВЛЯТЬ ТРАНЗАКЦИЯМИ С ПОМОЩЬЮ HIBERNATE?**

Hibernate вообще не допускает большинство операций без использования транзакций. Поэтому после получения экземпляра session от SessionFactory необходимо выполнить beginTransaction() для начала транзакции. Метод вернет ссылку, которую мы можем использовать для подтверждения или отката транзакции.
В целом, управление транзакциями в фреймворке выполнено гораздо лучше, чем в JDBC, т.к. мы не должны полагаться на возникновение исключения для отката транзакции. Любое исключение автоматически вызовет rollback.

- **ЧТО ТАКОЕ КАСКАДНЫЕ СВЯЗИ (ОБНОВЛЕНИЯ) В HIBERNATE?**

Если у нас имеются зависимости между сущностями (entities), то нам необходимо определить как различные операции будут влиять на другую сущность. Это реализуется с помощью каскадных связей (или обновлений). Вот пример кода с использованием аннотации @Cascade:

```
1  import org.hibernate.annotations.Cascade;
2
3  @Entity
4  @Table(name = "EMPLOYEE")
5  public class Employee {
6
7  @OneToOne(mappedBy = "employee")
8  @Cascade(value = org.hibernate.annotations.CascadeType.ALL)
9  private Address address;
10 }
```

Есть некоторые различия между enum CascadeType в Hibernate и в JPA. Поэтому обращайте внимание какой пакет вы импортируете при использовании аннотации и константы типа.

- **КАКИЕ КАСКАДНЫЕ ТИПЫ ЕСТЬ В HIBERNATE?**

Наиболее часто используемые CascadeType перечисления описаны ниже.

- None: без Cascading. Формально это не тип, но если мы не указали каскадной связи, то никакая операция для родителя не будет иметь эффекта для ребенка.
- ALL: Cascades save, delete, update, evict, lock, replicate, merge, persist. В общем – всё.
- SAVE_UPDATE: Cascades save и update. Доступно только для hibernate.
- DELETE: передает в Hibernate native DELETE действие. Только для hibernate.
- DETATCH, MERGE, PERSIST, REFRESH и REMOVE – для простых операций.
- LOCK: передает в Hibernate native LOCK действие.
- REPLICATE: передает в Hibernate native REPLICATE действие.

- **ЧТО ТАКОЕ СЕСИИЯ И ФАБЛИКА СЕССИЙ В HIBERNATE? КАК НАСТРОИТЬ SESSION FACTORY В КОНФИГУРАЦИОННОМ ФАЙЛЕ SPRING?**

Hibernate сессия – это главный интерфейс взаимодействия Java-приложения и Hibernate. SessionFactory позволяет создавать сессии согласно конфигурации

hibernate.cfg.xml. Например:

```
// Initialize the Hibernate environment
Configuration cfg = new Configuration().configure();
// Create the session factory
SessionFactory factory = cfg.buildSessionFactory();
// Obtain the new session object
Session session = factory.openSession();
```

При вызове Configuration().configure() загружается файл hibernate.cfg.xml и происходит настройка среды Hibernate. После того, как конфигурация загружена, вы можете сделать дополнительную модификацию настроек уже на программном уровне. Данные корректировки возможны до создания экземпляра фабрики сессий. Экземпляр SessionFactory как правило создается один раз и используется во всем приложении.

Главная задача сессии – обеспечить механизмы создания, чтения и удаления для экземпляров примапленых к БД классов. Экземпляры могут находиться в трёх состояниях:

transient – никогда не сохранялись, не ассоциированы ни с одной сессией;

persistent – ассоциированы с уникальной сессией;

detached – ранее сохраненные, не ассоциированы с сессией.

Объект Hibernate Session представляет одну операцию с БД. Сессию открывает фабрика сессий. Сессия должна быть закрыта, когда все операции с БД совершены. Пример:

```
Session session = null;
UserInfo user = null;
Transaction tx = null;
try {
 session = factory.openSession();
 tx = session.beginTransaction();
 user = (UserInfo)session.load(UserInfo.class, id);
 tx.commit();
} catch(Exception e) {
 if (tx != null) {
  try {
   tx.rollback();
  } catch (HibernateException e1) {
   throw new DAOException(e1.toString()); }
 }
 throw new DAOException(e.toString());
} finally {
 if (session != null) {
  try {
   session.close();
  } catch (HibernateException e) { }
 }
}
```

- **КАК ИСПОЛЬЗОВАТЬ JNDI DATASOURCE СЕРВЕРА ПРИЛОЖЕНИЙ С HIBERNATE FRAMEWORK?**

В веб приложении лучше всего использовать контейнер сервлетов для управления пулом соединений. Поэтому лучше определить JNDI ресурс для DataSource и использовать его в веб приложении. Для этого в Hibernate нужно удалить все специфичные для базы данных свойства из и использовать указания свойства JNDI DataSource:

```
<property name="hibernate.connection.datasource">java:comp/env/jdbc/MyLocalDB</property>
```

- **КАК ИНТЕГРИРОВАТЬ HIBERNATE И SPRING?**

Лучше всего прочитать о настройках на сайтах фреймворков для текущей версии. Оба фреймворка поддерживают интеграцию из коробки и в общем настройка их взаимодействия не составляет труда. Общие шаги выглядят следующим образом.

- Добавить зависимости для hibernate-entitymanager, hibernate-core и spring-orm.
- Создать классы модели и передать реализации DAO операции над базой данных. Важно, что DAO классы используют SessionFactory, который внедряется в конфигурации бинов Spring.
- Настроить конфигурационный файл Spring (смотрите в офф. документации или из примера на этом сайте).
- Дополнительно появляется возможность использовать аннотацию @Transactional и перестать беспокоиться об управлением транзакцией Hibernate.

- **КАКИЕ ПАТТЕРНЫ ПРИМЕНЯЮТСЯ В HIBERNATE?**

Domain Model Pattern – объектная модель предметной области, включающая в себя как поведение так и данные.
Data Mapper – слой мапперов (Mappers), который передает данные между объектами и базой данных, сохраняя их независимыми друг от друга и себя.
Proxy Pattern – применяется для ленивой загрузки.
Factory pattern – используется в SessionFactory

- **РАССКАЖИТЕ О HIBERNATE VALIDATOR FRAMEWORK.**

Проверка данных является неотъемлемой частью любого приложения. Hibernate Validator обеспечивает эталонную реализацию двух спецификаций JSR-303 и JSR-349 применяемых в Java. Для настройки валидации в Hibernate необходимо сделать следующие шаги.

- Добавить hibernate validation зависимости в проект.

```
<dependency>
    <groupid>javax.validation</groupid>
    <artifactid>validation-api</artifactid>
    <version>1.1.0.Final</version>
</dependency>
<dependency>
    <groupid>org.hibernate</groupid>
    <artifactid>hibernate-validator</artifactid>
    <version>5.1.1.Final</version>
</dependency>
```

- Так же требуются зависимости из JSR 341, реализующие Unified Expression Language для обработки динамических выражений и сообщений о нарушении ограничений.

```
<dependency>
    <groupid>javax.el</groupid>
    <artifactid>javax.el-api</artifactid>
    <version>2.2.4</version>
</dependency>
<dependency>
    <groupid>org.glassfish.web</groupid>
    <artifactid>javax.el</artifactid>
    <version>2.2.4</version>
</dependency>
```

- Использовать необходимые аннотации в бинах.

```
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.CreditCardNumber;
import org.hibernate.validator.constraints.Email;

public class Employee {

    @Min(value=1, groups=EmpIdCheck.class)
    private int id;

    @NotNull(message="Name cannot be null")
    @Size(min=5, max=30)
    private String name;

    @Email
    private String email;

    @CreditCardNumber
    private String creditCardNumber;
....
```

- **КАКИЕ ПРЕИМУЩЕСТВА ДАЕТ ИСПОЛЬЗОВАНИЕ ПЛАГИНА HIBERNATE TOOLS ECLIPSE?**

Плагин Hibernate Tools упрощает настройку маппинга, конфигурационного файла. Упрощает работы с файлами свойств или xml тегами. Помогает минимизировать ошибки написания кода.