Ответы на вопросы на собеседование Java Persistence API (JPA) (часть 2).

Vasyl K 19:01:00 1 Комментарий

- **ЧТО ТАКОЕ ENTITYMANAGER И КАКИЕ ОСНОВНЫЕ ЕГО ФУНКЦИИ ВЫ МОЖЕТЕ ПЕРЕЧИСЛИТЬ?**

EntityManager это интерфейс, который описывает API для всех основных операций над Enitity, получение данных и других сущностей JPA. По сути главный API для работы с JPA. Основные операции:

- Для операций над Entity: persist (добавление Entity под управление JPA), merge (обновление), remove (удаления), refresh (обновление данных), detach (удаление из управление JPA), lock (блокирование Enity от изменений в других thread),
- Получение данных: find (поиск и получение Entity), createQuery, createNamedQuery, createNativeQuery, contains, createNamedStoredProcedureQuery, createStoredProcedureQuery
- Получение других сущностей JPA: getTransaction, getEntityManagerFactory, getCriteriaBuilder, getMetamodel, getDelegate
- Работа с EntityGraph: createEntityGraph, getEntityGraph
- Общие операции над EntityManager или всеми Entities: close, isOpen, getProperties, setProperty, clear.

- **КАКИЕ ЧЕТЫРЕ СТАТУСА ЖИЗНЕННОГО ЦИКЛА ENTITY ОБЪЕКТА (ENTITY INSTANCE'S LIFE CYCLE) ВЫ МОЖЕТЕ ПЕРЕЧИСЛИТЬ?**

У Entity объекта существует четыре статуса жизненного цикла: new, managed, detached, или removed. Их описание

- new – объект создан, но при этом ещё не имеет сгенерированных первичных ключей и пока ещё не сохранен в базе данных,
- managed – объект создан, управляется JPA, имеет сгенерированные первичные ключи,
- detached – объект был создан, но не управляется (или больше не управляется) JPA,

- removed – объект создан, управляется JPA, но будет удален после commit'а транзакции.

- **КАК ВЛИЯЕТ ОПЕРАЦИЯ MERGE НА ENTITY ОБЪЕКТЫ КАЖДОГО ИЗ ЧЕТЫРЕХ СТАТУСОВ?**

1. Если статус detached, то либо данные будет скопированы в существующей managed entity с тем же первичным ключом, либо создан новый managed в который скопируются данные,
2. Если статус Entity new, то будет создана новый managed entity, в который будут скопированы данные прошлого объекта,
3. Если статус managed, операция игнорируется, однако операция merge сработает на каскадно зависимые Entity, если их статус не managed,
4. Если статус removed, будет выкинут exception сразу или на этапе commit'а транзакции.

- **КАК ВЛИЯЕТ ОПЕРАЦИЯ REMOVE НА ENTITY ОБЪЕКТЫ КАЖДОГО ИЗ ЧЕТЫРЕХ СТАТУСОВ?**

1. Если статус Entity new, операция игнорируется, однако зависимые Entity могут поменять статус на removed, если у них есть аннотации каскадных изменений и они имели статус managed,
2. Если статус managed, то статус меняется на removed и запись объект в базе данных будет удалена при commit'е транзакции (так же произойдут операции remove для всех каскадно зависимых объектов),
3. Если статус removed, то операция игнорируется,
4. Если статус detached, будет выкинут exception сразу или на этапе commit'а транзакции.

- **КАК ВЛИЯЕТ ОПЕРАЦИЯ PERSIST НА ENTITY ОБЪЕКТЫ КАЖДОГО ИЗ ЧЕТЫРЕХ СТАТУСОВ?**

1. Если статус Entity new, то он меняется на managed и объект будет сохранен в базу при commit'е транзакции или в результате flush операций,
2. Если статус уже managed, операция игнорируется, однако зависимые Entity могут поменять статус на managed, если у них есть аннотации каскадных изменений,
3. Если статус removed, то он меняется на managed,
4. Если статус detached, будет выкинут exception сразу или на этапе commit'а транзакции.

- **КАК ВЛИЯЕТ ОПЕРАЦИЯ REFRESH НА ENTITY ОБЪЕКТЫ КАЖДОГО ИЗ ЧЕТЫРЕХ СТАТУСОВ?**

1. Если статус Entity managed, то в результате операции будут востановленны все изменения из базы данных данного Entity, так же произойдет refresh всех каскадно зависимых объектов,
2. Если статус new, removed или detached, будет выкинут exception.

- **КАК ВЛИЯЕТ ОПЕРАЦИЯ DETACH НА ENTITY ОБЪЕКТЫ КАЖДОГО ИЗ ЧЕТЫРЕХ СТАТУСОВ?**

1. Если статус Entity managed или removed, то в результате операции статус Entity (и всех каскадно-зависимых объектов) станет detached.
2. Если статус new или detached, то операция игнорируется.

- **ДЛЯ ЧЕГО НУЖНА АННОТАЦИЯ ACCESS?**

Она определяет тип доступа (access type) для класса entity, суперкласса, embeddable или отдельных атрибутов, то есть как JPA будет обращаться к атрибутам entity, как к полям класса (FIELD) или как к свойствам класса (PROPERTY), имеющие гетеры (getter) и сетеры (setter).

- **ДЛЯ ЧЕГО НУЖНА АННОТАЦИЯ BASIC?**

Basic – указывает на простейший тип маппинга данных на колонку таблицы базы данных. Также в параметрах аннотации можно указать fetch стратегию доступа к полю и является ли это поле обязательным или нет.

- **КАКОЙ АННОТАЦИЯМИ МОЖНО ПЕРЕКРЫТЬ СВЯЗИ (OVERRIDE ENTITY RELATIONSHIP) ИЛИ АТРИБУТЫ, УНАСЛЕДОВАННЫЕ ОТ СУПЕРКЛАССА, ИЛИ ЗАДАННЫЕ В EMBEDDABLE КЛАССЕ ПРИ ИСПОЛЬЗОВАНИИ ЭТОГО EMBEDDABLE КЛАССА В ОДНОМ ИЗ ENTITY КЛАССОВ И НЕ ПЕРЕКРЫВАТЬ В ОСТАЛЬНЫХ?**

Для такого перекрывания существует четыре аннотации:
- AttributeOverride чтобы перекрыть поля, свойства и первичные ключи,
- AttributeOverrides аналогично можно перекрыть поля, свойства и первичные ключи со множественными значениями,
- AssociationOverride чтобы перекрывать связи (override entity relationship),
- AssociationOverrides чтобы перекрывать множественные связи (multiple relationship).

- **КАКИЕ АННОТАЦИИ СЛУЖИТ ДЛЯ ЗАДАНИЯ КЛАССА ПРЕОБРАЗОВАНИЯ BASIC АТРИБУТА ENTITY В ДРУГОЙ ТИП ПРИ СОХРАНЕНИИ/ПОЛУЧЕНИИ ДАННЫХ ИХ БАЗЫ (НАПРИМЕР, РАБОТАТЬ С АТРИБУТОМ ENTITY BOOLEAN ТИПА, НО В БАЗУ СОХРАНЯТЬ ЕГО КАК ЧИСЛО)?**

Convert и Converts – позволяют указать класс для конвертации Basic атрибута Entity в другой тип (Converts – позволяют указать несколько классов конвертации). Классы для конвертации должны реализовать интерфейс AttributeConverter и могут быть отмечены (но это не обязательно) аннотацией Converter.

- **КАКОЙ АННОТАЦИЕЙ МОЖНО УПРАВЛЯТЬ КЕШИРОВАНИЕМ JPA ДЛЯ ДАННОГО ENTITY?**

Cacheable – позволяет включить или выключить использование кеша второго уровня (second-level cache) для данного Entity (если провайдер JPA поддерживает работу с кешированием и настройки кеша (second-level cache) стоят как ENABLE_SELECTIVE или DISABLE_SELECTIVE, см вопрос 41). Обратите внимание свойство наследуется и если не будет перекрыто у наследников, то кеширование измениться и для них тоже.

- **КАКОЙ АННОТАЦИЕЙ МОЖНО ЗАДАТЬ КЛАСС, МЕТОДЫ КОТОРОГО ДОЛЖЕН ВЫПОЛНИТСЯ ПРИ ОПРЕДЕЛЕННЫХ JPA ОПЕРАЦИЯХ НАД ДАННЫМ ENITITY ИЛИ MAPPED SUPERCLASS (ТАКИЕ КАК УДАЛЕНИЕ, ИЗМЕНЕНИЕ ДАННЫХ И Т.П.)?**

Аннотация EntityListeners позволяет задать класс Listener, который будет содержать методы обработки событий (callback methods) определенных Entity или Mapped Superclass.

- **ДЛЯ ЧЕГО НУЖНЫ CALLBACK МЕТОДЫ В JPA? К КАКИМ СУЩНОСТЯМ ПРИМЕНЯЮТСЯ АННОТАЦИИ CALLBACK МЕТОДОВ? ПЕРЕЧИСЛИТЕ СЕМЬ CALLBACK МЕТОДОВ (ИЛИ ЧТО ТОЖЕ САМОЕ АННОТАЦИЙ CALLBACK МЕТОДОВ).**

Callback методы служат для вызова при определенных событиях Entity (то есть добавить обработку например удаления Entity методами JPA), могут быть добавлены к entity классу, к mapped superclass, или к callback listener классу, заданному

аннотацией EntityListeners (см предыдущий вопрос). Существует семь callback методов (и аннотаций с теми же именами):

- PrePersist
- PostPersist
- PreRemove
- PostRemove
- PreUpdate
- PostUpdate
- PostLoad

- **КАКОЙ АННОТАЦЕЙ МОЖНО ИСКЛЮЧИТЬ ПОЛИ И СВОЙСТВА ENTITY ИЗ МАППИНГА (PROPERTY OR FIELD IS NOT PERSISTENT)?**

Для этого служит аннотация Transient.

- **КАКИЕ АННОТАЦИИ СЛУЖИТЬ ДЛЯ УСТАНОВКИ ПОРЯДКА ВЫДАЧИ ЭЛЕМЕНТОВ КОЛЛЕКЦИЙ ENTITY?**

Для этого служит аннотация OrderBy и OrderColumn.

- **КАКИЕ ШЕСТЬ ВИДОВ БЛОКИРОВОК (LOCK) ОПИСАНЫ В СПЕЦИФИКАЦИИ JPA (ИЛИ КАКИЕ ЕСТЬ ЗНАЧЕНИЯ У ENUM LOCKMODETYPE В JPA)?**

У JPA есть шесть видов блокировок, перечислим их в порядке увеличения надежности (от самого ненадежного и быстрого, до самого надежного и медленного):

- NONE – без блокировки
- OPTIMISTIC (или синоним READ, оставшийся от JPA 1) – оптимистическая блокировка
- OPTIMISTIC_FORCE_INCREMENT (или синоним WRITE, оставшийся от JPA 1) – оптимистическая блокировка с принудительным увеличением поля версионности
- PESSIMISTIC_READ – пессимистичная блокировка на чтение
- PESSIMISTIC_WRITE – пессимистичная блокировка на запись (и чтение)
- PESSIMISTIC_FORCE_INCREMENT – пессимистичная блокировка на запись (и чтение) с принудительным увеличением поля версионности.

- **КАКИЕ ДВА ВИДА КЭШЕЙ (CACHE) ВЫ ЗНАЕТЕ В JPA И ДЛЯ ЧЕГО ОНИ НУЖНЫ?**

JPA говорит о двух видов кэшей (cache):

- first-level cache (кэш первого уровня) – кэширует данные одной транзакции,

- second-level cache (кэш второго уровня) – кэширует данные дольше чем одна транзакция. Провайдер JPA может, но не обязан реализовывать работу с кэшем второго уровня. Такой вид кэша позволяет сэкономить время доступа и улучшить производительность, однако оборотной стороной является возможность получить устаревшие данные.

• КАКИЕ ЕСТЬ ВАРИАНТЫ НАСТРОЙКИ SECOND-LEVEL CACHE (КЭША ВТОРОГО УРОВНЯ) В JPA ИЛИ ЧТО АНАЛОГИЧНО ОПИШИТЕ КАКИЕ ЗНАЧЕНИЯ МОЖЕТ ПРИНИМАТЬ ЭЛЕМЕНТ SHARED-CACHE-MODE ИЗ PERSISTENCE.XML?

JPA говорит о пяти значениях shared-cache-mode из persistence.xml, который определяет как будет использоваться second-level cache:

- ALL – все Entity могут кэшироваться в кеше второго уровня
- NONE – кеширование отключено для всех Entity
- ENABLE_SELECTIVE – кэширование работает только для тех Entity, у которых установлена аннотация Cacheable(true) или её xml эквивалент, для всех остальных кэширование отключено
- DISABLE_SELECTIVE – кэширование работает для всех Entity, за исключением тех у которых установлена аннотация Cacheable(false) или её xml эквивалент
- UNSPECIFIED – кеширование не определенно, каждый провайдер JPA использует свою значение по умолчанию для кэширования

• КАК МОЖНО ИЗМЕНИТЬ НАСТРОЙКИ FETCH СТРАТЕГИИ ЛЮБЫХ АТРИБУТОВ ENTITY ДЛЯ ОТДЕЛЬНЫХ ЗАПРОСОВ (QUERY) ИЛИ МЕТОДОВ ПОИСКА (FIND), ТО ЕСЛИ У ENITY ЕСТЬ АТРИБУТ С FETCHTYPE = LAZY, НО ДЛЯ КОНКРЕТНОГО ЗАПРОСА ЕГО ТРЕБУЕТСЯ СДЕЛАТЬ EAGER ИЛИ НАОБОРОТ?

Для этого существует EntityGraph API, используется он так: с помощью аннотации NamedEntityGraph для Entity, создаются именованные EntityGraph объекты, которые содержат список атрибутов у которых нужно поменять fetchType на EAGER, а потом данное имя указывается в hits запросов или метода find. В результате fetchType атрибутов Entity меняется, но только для этого запроса. Существует две стандартных property для указания EntityGraph в hit:

- javax.persistence.fetchgraph – все атрибуты перечисленные в EntityGraph меняют fetchType на EAGER, все остальные на LAZY
- javax.persistence.loadgraph – все атрибуты перечисленные в EntityGraph меняют fetchType на EAGER, все остальные сохраняют свой fetchType (то есть если у атрибута, не указанного в EntityGraph, fetchType был EAGER, то он и останется EAGER)С помощью NamedSubgraph можно также изменить fetchType вложенных объектов Entity.

- **КАКИМ СПОСОБОМ МОЖНО ПОЛУЧИТЬ МЕТАДАННЫЕ JPA (СВЕДЕНИЯ О ENTITY ТИПАХ, EMBEDDABLE И MANAGED КЛАССАХ И Т.П.)?**

Для получения такой информации в JPA используется интерфейс Metamodel. Объект этого интерфейса можно получить методом getMetamodel у EntityManagerFactory или EntityManager.

- **КАКИМ СПОСОБОМ МОЖНО В КОДЕ РАБОТАТЬ С КЭШЕМ ВТОРОГО УРОВНЯ (УДАЛЯТЬ ВСЕ ИЛИ ОПРЕДЕЛЕННЫЕ ENTITY ИЗ КЕША, УЗНАТЬ ЗАКЭШИРОВАЛСЯ ЛИ ДАННОЕ ENTITY И Т.П.)?**

Для работы с кэшем второго уровня (second level cache) в JPA описан Cache интерфейс, содержащий большое количество методов по управлению кэшем второго уровня (second level cache), если он поддерживается провайдером JPA, конечно. Объект данного интерфейса можно получить с помощью метода getCache у EntityManagerFactory.

- **В ЧЕМ РАЗНИЦА В ТРЕБОВАНИЯХ К ENTITY В HIBERNATE, ОТ ТРЕБОВАНИЙ К ENTITY, УКАЗАННЫХ В СПЕЦИФИКАЦИИ JPA?**

1. Конструктор без аргументов не обязан быть public или protected, рекомендуется чтобы он был хотя бы package видимости, однако это только рекомендация, если настройки безопасности Java позволяют доступ к приватным полям, то он может быть приватным,
2. JPA категорически требует не использовать final классы, Hibernate лишь рекомендует не использовать такие классы чтобы он мог создавать прокси для ленивой загрузки, однако позволяет либо выключить прокси Proxy(lazy=false), либо использовать в качестве прокси интерфейс, содержащий все методы маппинга для данного класса (аннотацией Proxy(proxyClass=интерфейс.class))

- **КАКАЯ УНИКАЛЬНАЯ СТРАТЕГИЯ НАСЛЕДОВАНИЯ ЕСТЬ В HIBERNATE, НО НЕТ В СПЕЦИФИКАЦИИ JPA?**

В отличии JPA в Hibernate есть уникальная стратегия наследования, которая называется implicit polymorphism.

- **КАКИЕ ОСНОВНЫЕ НОВЫЕ ВОЗМОЖНОСТИ ПОЯВИЛИСЬ В СПЕЦИФИКАЦИИ JPA 2.1 ПО СРАВНЕНИЮ С JPA 2.0 (ПЕРЕЧИСЛИТЕ ХОТЯ БЫ ПЯТЬ-ШЕСТЬ НОВЫХ ВОЗМОЖНОСТЕЙ)?**

В спецификации JPA 2.1 появились:

- Entity Graphs – механизм динамического изменения fetchType для каждого запроса
- Converters – механизм определения конвертеров для задания функций конвертации атрибутов Entity в поля базы данных
- DDL генерация – автоматическая генерация таблиц, индексов и схем
- Stored Procedures – механизм вызова хранимых процедур из JPA
- Criteria Update/Delete – механизм вызова bulk updates или deletes, используя Criteria API
- Unsynchronized persistence contexts – появление возможности указать SynchronizationType
- Новые возможности в JPQL/Criteria API: арифметические подзапросы, generic database functions, join ON clause, функция TREAT
- Динамическое создание именованных запросов (named queries)
- Интерфейс EntityManager получил новые методы createStoredProcedureQuery, isJoinedToTransaction и createQuery(CriteriaUpdate или CriteriaDelete)
- Абстрактный класс AbstractQuery стал наследоваться от класса CommonAbstractCriteria, появились новые интерфейсы CriteriaUpdate, CriteriaDelete унаследованные CommonAbstractCriteria,
- PersistenceProvider получил новые функции generateSchema позволяющие генерить схемы,
- EntityManagerFactory получил методы addNamedQuery, unwrap, addNamedEntityGraph, createEntityManager (с указанием SynchronizationType)
- Появился новый enum SynchronizationType, Entity Graphs, StoredProcedureQuery и AttributeConverter интерфейсы.