Транзакции и сущности

Типы контекстов персистентности

Контексты персистентности бывают двух типов:
1) транзакционные (время жизни контекста ограничено контекстом транзакции);
2) расширенные (время жизни контекста может охватывать несколько транзакций).
Тип контекста указывается при создании менеджера сущностей. Один из способов это сделать – использовать элемент `type` аннотации `@PersistenceContext`, которая используется для внедрения ссылки на менеджер сущностей; допустимые значения элемента – `PersistenceContextType.TRANSACTION` и `PersistenceContextType.EXTENDED`.
По умолчанию используются транзакционные контексты персистентности.
Расширенные контексты можно использовать только в сессионных компонентах с состоянием.
Расширенный контекст персистентности существует с момента создания менеджера сущностей и до его закрытия. Время его жизни может охватывать несколько транзакционных и нетранзакционных обращений к менеджеру сущностей. Расширенный контекст персистентности вовлекается в текущую транзакцию, когда в ее рамках вызываются методы менеджера сущностей этого контекста или методы сессионного компонента с состоянием, с которым связан этот контекст.
При подтверждении транзакции все экземпляры сущностей, управляемые транзакционным контекстом персистентности, переходят в отсоединенное состояние.
Расширенный контекст персистентности оставляет все экземпляры сущностей в управляемом состоянии и после подтверждения транзакции. Методы менеджера сущностей `persist()`, `remove()`, `merge()` и `refresh()` можно вызывать независимо от того, имеется активная транзакция или нет. Действие этих методов будет сохранено в базе данных при подтверждении ближайшей транзакции, в которую будет вовлечен расширенный контекст персистентности. Также может пригодиться метод `refresh()`, так как при начале новой транзакции управляемые экземпляры сущностей не загружаются вновь из базы данных.
При откате транзакции все управляемые экземпляры сущностей (а также экземпляры сущностей, находившиеся в удаленном состоянии) становятся отсоединенными. Дальнейшее использование этих объектов не рекомендуется, так как они могут не соответствовать реальному состоянию базы данных, даже в какой-либо момент времени в прошлом (это вытекает из свойств транзакции).

Виды управления транзакциями

Есть два вида управления транзакциями при работе с менеджером сущностей: 1) с помощью JTA, 2) локальное управление транзакциями с помощью интерфейса `EntityTransaction`, вызовы которого отображаются на методы управления транзакциями источника данных.
Вид управления транзакциями устанавливается через дескриптор модуля персистентности (файл `persistence.xml`). По умолчанию на платформе Java EE выбирается управление с помощью JTA. При этом для модуля персистентности должен использоваться JTA-совместимый источник данных.
Так как в управляемом контейнером контексте персистентности управление транзакциями должно выполняться с помощью JTA, то далее рассмотрим только данный вид управления транзакциями.

Управление транзакциями через JTA

Менеджер сущностей с управлением транзакциями через JTA принимает участие в текущей JTA-транзакции, которая начинается и завершается извне менеджера сущностей, а также включает в транзакцию используемый им менеджер ресурсов (источник данных).
В рамках JTA-транзакции все управляемые контейнером менеджеры сущностей, относящиеся к одному модулю персистентности, используют общий контекст персистентности. Поэтому отсутствует необходимость передавать в качестве параметра ссылку на менеджера сущностей.
Контекст персистентности передается по менеджерам сущностей вместе с распространением JTA-транзакции. Если отсутствует активная JTA-транзакция или она не распространяется, то не передается и контекст персистентности. Но передача контекста действует только в локальном окружении – контекст персистентности не передается в другие JVM.
Рассмотрим далее ЖЦ контекстов персистентности разного типа.

1) Транзакционный контекст персистентности
Новый контекст создается при вызове метода управляемого контейнером менеджера сущностей в контексте активной JTA-транзакции, с которой еще не связан контекст персистентности. Затем созданный контекст связывается с JTA-транзакцией.
ЖЦ контекста персистентности завершается при подтверждении или откате связанной JTA-транзакции, а все управляемые им экземпляры сущностей становятся отсоединенными.
Если метод менеджера сущностей вызывается вне контекста транзакции, то все экземпляры сущностей, загруженные из БД, становятся отсоединенными непосредственно перед возвратом из метода.

2) Расширенный контекст персистентности
Расширенный контекст может быть создан только в рамках сессионного компонента с состоянием. Он существует с момента создания конкретного экземпляра компонента и привязан к нему.
Расширенный контекст персистентности закрывается после завершения `@Remove`-метода компонента или при удалении экземпляра сессионного компонента по какой-либо другой причине.
Если экземпляр сессионного компонента с состоянием (назовем его SFSB1) инициализирует экземпляр другого сессионного компонента с состоянием (SFSB2), и в обоих компонентах используются расширенные контексты персистентности, то расширенный контекст персистентности SFSB1 наследуется в (связывается с) SFSB2. Данное правило применяется рекурсивно и независимо от наличия активной транзакции в момент создания экземпляров компонентов.
Если расширенный контекст персистентности был унаследован какими-либо экземплярами сессионных компонентов с состоянием, то он закрывается только после удаления последнего из них.
Передача расширенного контекста персистентности вместе с распространением JTA-транзакции обладает следующей особенностью. Если расширенный контекст персистентности, связанный с экземпляром компонента, отличается от контекста, передаваемого с JTA-транзакцией, то контейнер выбрасывает исключение `EJBException`.

Рассмотрим пример использования расширенного контекста персистентности. Каждый метод компонента `ShoppingCartImpl` выполняется в отдельной транзакции (значение транзакционного атрибута `REQUIRES_NEW`), поэтому изменения в управляемых экземплярах сущностей сохраняются в БД при возврате из методов. Кроме того, подобный выбор

значения транзакционного атрибута позволяет избежать передачи контекста персистентности вместе с JTA-транзакцией и связанных с этим потенциальных ошибок.

```
@Stateful
@Transaction(REQUIRES_NEW)
public class ShoppingCartImpl implements ShoppingCart {
    @PersistenceContext(type=EXTENDED)
    EntityManager em;

    private Order order;

    private Product product;

    public void initOrder(Long id) {
        order = em.find(Order.class, id);
    }

    public void initProduct(String name) {
        product = (Product) em.createQuery("select p from Product p
                        where p.name = :name")
                    .setParameter("name", name)
                    .getSingleResult();
    }

    public LineItem createLineItem(int quantity) {
        LineItem li = new LineItem(order, product, quantity);
        order.getLineItems().add(li);
        return li;
    }
}
```

Оптимистическое и пессимистическое блокирование

Оптимистическое блокирование представляет собой стратегию блокирования набора данных, при которой изменяемая запись блокируется только на время внесения изменений в запись программой, но не пользователем. Пессимистическое блокирование – стратегия блокирования набора данных, при которой изменяемая запись блокируется на все время внесения изменений в запись пользователем и не доступна для редактирования другим пользователям.

Пессимистическое блокирование можно реализовать в языке SQL при помощи оператора `SELECT ... FOR UPDATE`, который блокирует любой доступ из других транзакций к записям, входящим в результирующую выборку.

При оптимистическом блокировании для изменения записи в предложении `WHERE` оператора `UPDATE` помимо идентификатора записи указываются также предыдущие значения других полей, что позволяет убедиться в отсутствии изменений, внесенных в запись другими транзакциями. Проще всего для этой цели выделить особое поле – поле версии, увеличиваемое на 1 при каждом очередном изменении записи. Например, оператор изменения записи может иметь следующий вид: `UPDATE LineItem SET version = version + 1, quantity = 10 WHERE id = 10638 AND version = 1`. Если запись таблицы `LineItem` с идентификатором 10638 была изменена в другой транзакции(-ях), то поле `version` будет содержать значение, отличное от 1. В этом случае количество записей, обработанных указанным запросом будет 0, а не 1, и приложение должно будет сообщить пользователю о невозможности изменения данных.

Оптимистическое блокирование обеспечивает решение проблемы потери обновлений, выполняемых конкурирующими транзакциями. Если транзакция А пытается переписать изменения экземпляра сущности, зафиксированные транзакцией Б, начатой после чтения

экземпляра сущности в рамках транзакции A, то возникает исключение `OptimisticLockException` и транзакция A откатывается.

Оптимистическое блокирование при работе с сущностями подразумевает, что провайдер персистентности ведет доступ к БД на уровне изоляции READ COMMITTED (чтение подтвержденных данных), а запись в БД происходит только при вызове метода `flush()` менеджера сущностей. Этот метод можно вызывать явно из приложения, либо провайдер вызывает его самостоятельно в соответствии с установленным режимом сброса (flush mode). В любом случае метод `flush()` вызывается также при завершении транзакции.

Чтобы использовать пессимистическое блокирование, требуется установить для соединения с БД более высокий уровень изоляции транзакций, чем READ COMMITTED.

Оптимистическое блокирование включается для конкретных классов сущностей путем добавления к ним версионного атрибута (поля или свойства) – атрибута типа `int`, `Integer`, `short`, `Short`, `long`, `Long` или `Timestamp`, помеченного аннотацией `@Version`. Приложение не должно изменять значение версионного атрибута, это происходит автоматически при записи экземпляра сущности в базу данных.

При выполнении метода `merge()` менеджер сущностей должен проверять значение версионного атрибута и выбрасывать исключение `OptimisticLockException`, если присоединяемый к контексту экземпляр сущности устарел.

Провайдер персистентности может откладывать фактическую запись в БД до окончания транзакции. Если в приложении требуется оперативно обрабатывать исключение `OptimisticLockException`, следует использовать метод `flush()` менеджера сущностей для немедленного сброса изменений в объектах контекста персистентности в БД.

Рекомендуется включить оптимистическое блокирование для всех сущностей, к которым ведется параллельный доступ, и которые присоединяются к контексту персистентности из отсоединенного состояния.

При изменении или удалении экземпляра сущности, содержащей версионный атрибут, провайдер персистентности обязан гарантировать отсутствие проблем «грязного» чтения и неповторяемого чтения.