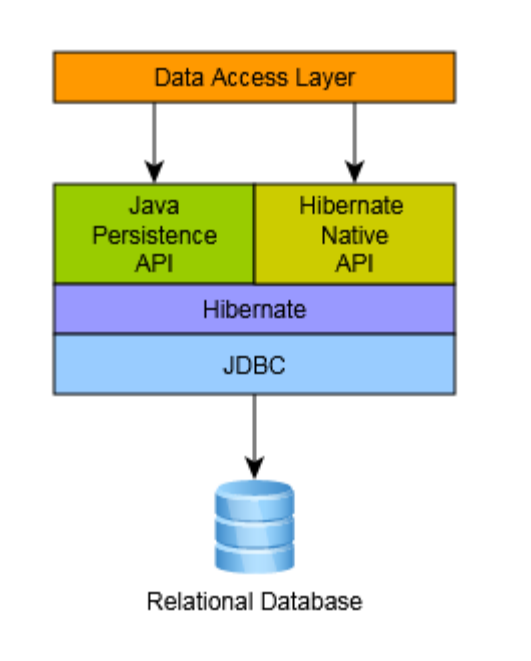
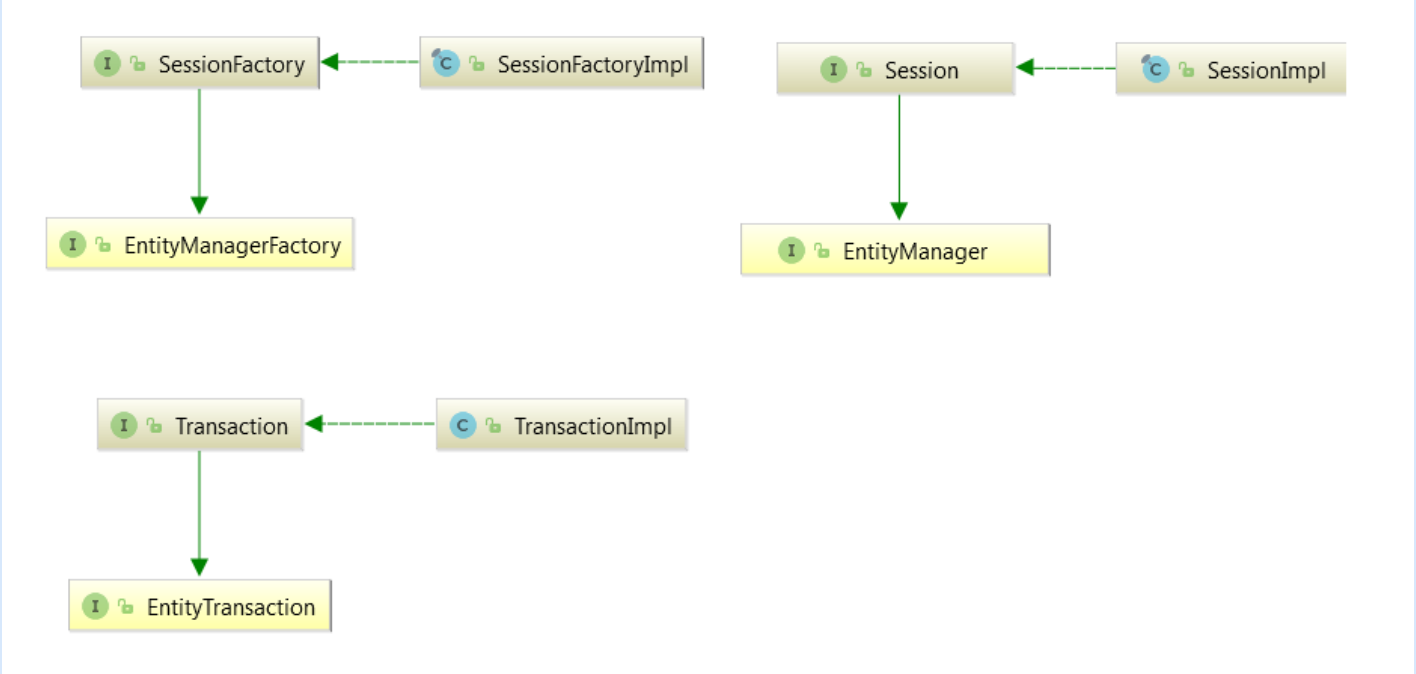
ЗНАКОМИМСЯ С HIBERNATE ПОБЛИЖЕ

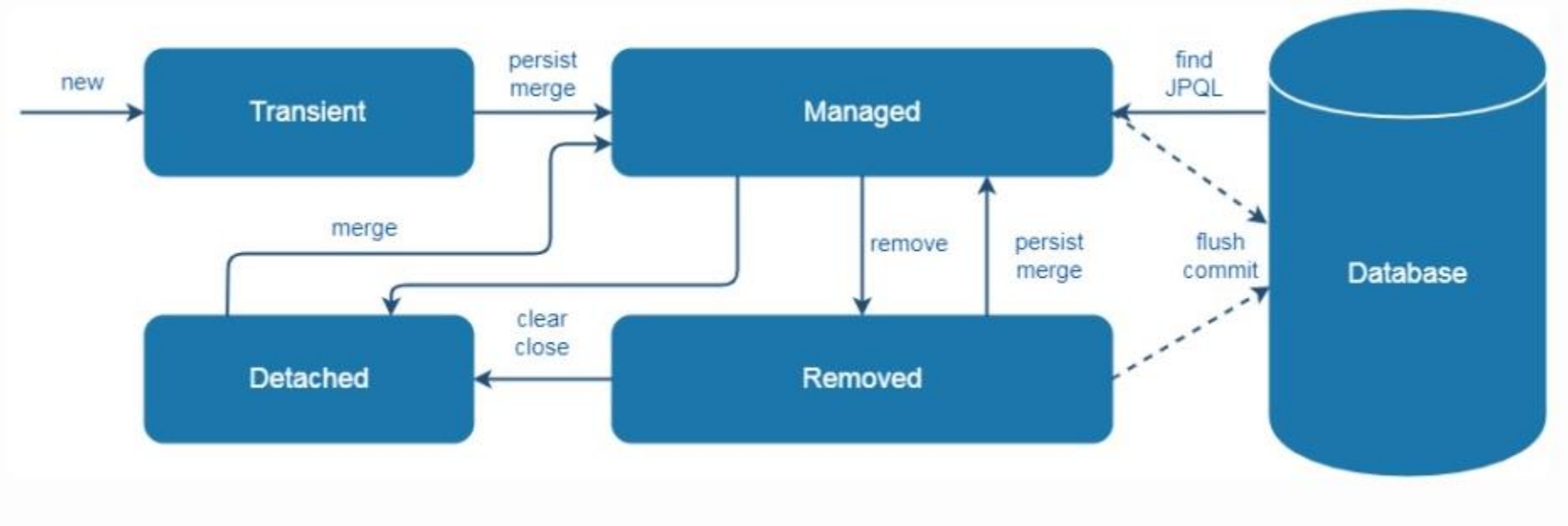
hibernate-core 5.6.9

Архитектура





Состояния, в которых может быть сущность



Виды отношений (связей):

- OneToOne
- OneToMany
- ManyToOne
- ManyToMany

Связь между сущностями может быть

- Односторонняя (unidirectional)
- Двусторонняя (bidirectional)

У двусторонней связи есть:

- владеющая связью сторона (owning side)
- противоположная или не владеющая связью сторона (inverse or non-owning side)

Односторонняя связь имеет только owning side

Parent/Child

Правила для установления двусторонней связи:

1. **Не владеющая связью сторона должна ссылаться на владеющую сторону, используя элемент mappedBy аннотаций OneToOne, OneToMany, ManyToMany. Элемент mappedBy указывает поле в сущности, которая является владельцем связи.**

2. **Сторона many-to в связи one-to-many/many-to-one должна быть владеющей стороной. Элемент mappedBy не может быть указан в аннотации ManyToOne.**

Правила для установления двусторонней связи (продолжение):

3. Для one-to-one связей владеющая сторона соответствует стороне, содержащей внешний ключ.

4. Для many-to-many связей любая сторона может быть владеющей связью.

Элементы аннотаций:

- **targetEntity.**

Применительно к связям to-many должна быть указана, если используется сырой тип (Raw Type) коллекции. Если коллекция java.util.Map, то применяется к map value:

```
@OneToMany(targetEntity=net.homer.Order.class, mappedBy="customer")
private Set orders;

@ManyToMany(targetEntity=net.homer.Customer.class, mappedBy="phones")
private Set customers;
```

Если указан тип элементов коллекции, то данный элемент является необязательным

```
@OneToMany(mappedBy="customer")
private Set<Order> orders;
```

Применительно к связям to-one для указания конкретного типа поля, владеющего связью (если в коде тип ссылки к примеру Object).

```
@OneToOne(targetEntity=net.homer.Order.class, mappedBy="customer")
private Object order;
```

По умолчанию тип элементов коллекции или тип поля, владеющего связью.

Элементы аннотаций (продолжение):

- cascade.

ALL, PERSIST, MERGE, REMOVE, REFRESH, DETACH.

Указывает какие операции должны применяться к child, если эта операция применяется к Parent. Спецификация JPA рекомендует применять Remove только для отношений one-to.
По умолчанию никакие операции не выполняются каскадно.

Для аннотаций @OneToMany и @ManyToMany если целевая коллекция является java.util.Map, то каскадная операция применяется к map value .

- fetch

Указывает должна ли загружать связанная сущность сразу (EAGER) или по требованию (LAZY)
По умолчанию:
- **EAGER для @OneToOne и @ManyToOne**
- **LAZY для @OneToMany и @ManyToMany**

Элементы аннотаций (продолжение):

- **mappedBy** (отсутствует в @ManyToOne)

Указывает поле в сущности, которая является владельцем связи. Обязательно для двусторонней связи

- **optional** (отсутствует в @ManyToMany и @OneToMany)

Возможные значения: true, false
Указывает является ли данная связь обязательной. Если указано false, то всегда должна существовать связь (не null)
По умолчанию true.

Элементы аннотаций (продолжение):

- **orphanRemoval (отсутствует в @ManyToMany и @ManyToOne)**

Возможные значения: true, false
Если задано, то при “отвязывании” дочерней сущности от родительской, она будет удалена из БД (например, при удалении сущности из вложенной коллекции внутри родительской сущности)

Для аннотации @OneToMany если целевая коллекция является java.util.Map, то данная операция применяется к map value

По умолчанию false

@OneToOne

Способы создания связи:

1. Через внешний ключ (@JoinColumn)
2. Через таблицу соединения (@JoinTable)
3. Через общий первичный ключ (@PrimaryKeyJoinColumn)

Если используется внешний ключ, то существует проблема загрузки родительской стороны и ее ленивых связей. Они загружаются не лениво.

Демо

@OneToMany

Способы создания связи:

1. Через внешний ключ (@JoinColumn)
2. Через таблицу соединения (@JoinTable)

@OneToMany по определению является родительской стороной.

Демо

@ManyToOne

Способы создания связи:

1. Через внешний ключ (@JoinColumn)
2. Через таблицу соединения (@JoinTable)

@ManyToMany

Создается через таблицу соединения.

Демо

Best practices использования one-to-many и many-to-one:

1. Не используйте одностороннюю @OneToMany.
2. Стараться избегать связи to-many для большого количества возможных сущностей на стороне many. Лучше использовать однонаправленную many-to-one.
3. Подумайте дважды перед использованием CascadeType.Remove для to-many. Чтобы удалить сущности, Hibernate сначала их загружает в контекст, а потом удаляет по одной.
4. Используйте orphanRemoval при проектировании отношений parent-child, когда child не может существовать без parent
5. Реализуйте helper-методы для обновления двусторонней связи
6. Устанавливайте тип загрузки Lazy для many-to-one связей

Демо

Best practices использования many-to-many:

1. Старайтесь избегать использования List в виде контейнера для связанных сущностей. При удалении сущности из List сначала удалятся все записи из таблицы связей, потом туда добавятся связи для сущностей, которые не были удалены. Лучше моделировать данные отношения, используя Set.
2. Реализуйте helper-методы для обновления двусторонней связи
3. Не изменяйте FetchType
4. Не используйте CascadeType.Remove и CascadeType.All. В лучшем случае вы получите проблемы с производительностью, в худшем удалите нужные данные.

Демо

Проблема загрузки ленивых связей:

При доступе к связям сущности, которые загружаются по требованию, необходимо гарантировать, что загрузка связи будет происходить в той же транзакции, где загружалась и сама сущность. Иначе будет сгенерировано LazyInitializationException.

LazyCollection
В Hibernate существует аннотация LazyCollection, которая позволяет не загружать коллекцию при выполнении тривиальниых запросов. Параметр LazyCollectionOption.EXTRA включает поддержку операций с коллекцией, не вызывающих ее инициализацию.

Демо

Dirty Checking

По умолчанию Hibernate проверяет все свойства сущностей, управляемых контекстом хранения. Когда сущность загружается, делается дополнительная копия всех значений ее свойств. Во время flush каждое свойство управляемой сущности сравнивается с ранее сделанной копией, чтобы определить изменения.

Данный механизм можно отключать через

- **указание аннотации Hibernate @Immutable над сущностью.**
- **отсоединение сущности от контекста хранения (em.detach)**
- **установку FlushModeType.Manual**

Демо

Маппинг в DTO

В Hibernate есть возможность создавать на основе результатов запросов экземпляры классов, которые не помечены аннотацией Entity

Можно использовать ResultTransformer или @SqlResultSetMapping

Существует несколько ResultTransformer, к примеру:

- AliasToEntityMapResultTransformer
- AliasToBeanResultTransformer
- ToListResultTransformer

Демо

План извлечения связей

Существует 2 плана извлечения связей:

1. **Отложенный или ленивый (Lazy)**
2. **Немедленный (Eager)**

Lazy загрузка связей ведет к проблеме N + 1 выражений Select, так как требуется дополнительный запрос Select, чтобы загрузить связанную сущность. Если мы выгрузили N сущностей одним запросом, то для загрузки связей потребуется еще N запросов (N + 1 Select)

Eager загрузка связей выполняется вместе с самой сущностью одним запросом select с использованием join. Это может привести к декартовому произведению. Существует также настройка Hibernate, которая указывает сколько максимально таблиц можно соединять через join (hibernate.max_fetch_depth), после достижения которой связь будет загружаться отдельным Select. (Cartesian Product)

Демо

Способы изменить план извлечения для решения N + 1:

1. **Join fetch в JPQL запросе**
2. **Batch fetching (предсказывание)**
3. **Использование подзапросов**

Способы изменить план извлечения для решения декартового произведения:

1. **Использование отдельных запросов для загрузки связей вместо JOIN**

Демо

Существует возможность изменить настроенный план извлечения с помощью графов сущностей.

Графы сущностей бывают:

- **Граф загрузки - EntityGraphType.LOAD (javax.persistence.loadgraph)**
- **Граф извлечения - EntityGraphType.FETCH (javax.persistence.fetchgraph)**

Граф загрузки загружает все атрибуты, указанные как вершины графа, с параметром Eager. Те, что не указаны, загружается в соответствии со своими параметрами

Граф извлечения загружает все атрибуты, указанные как вершины графа, с параметром Eager. Те, что не указаны, загружается с параметром Lazy.

При использовании Spring Data JPA по умолчанию у графа тип EntityGraphType.FETCH

Демо