**Правильное использование методов - persist, merge из JPA и методов save, update, saveOrUpdate из Hibernate**

**Выводы**

**persist –** для сохранения сущности следует использовать этот метод JPA**.**

**merge** - является предпочтительным для копирования состояния detached-сущности.

**update** - полезен только для задач пакетной обработки.

**save** и **saveOrUpdate** — это просто псевдонимы для **update**, и вам не следует использовать их вообще, т.к. они могут давать исключения **NonUniqueObjectException**.

Некоторые разработчики используют save, даже если объект уже управляется,  
но это ошибка и вызывает лишнее событие, так как для управляемых сущностей UPDATE автоматически обрабатывается *Persistence context* во время flush.

**Введение**

В этой статье я собираюсь показать вам, как работают методы persist, merge из JPA и сравнить их с методами save, update, saveOrUpdate из Hibernate.

Хотя лучше использовать JPA-методы для [изменения состояния сущности](https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/) ([рус.](http://akorsa.ru/2016/07/rukovodstvo-dlya-nachinayushhih-po-perehody-mezhdu-sostoyaniyami-v-jpa-hibernate/)),  
вы увидите, что метод save из Hibernate является хорошей альтернативой merge, когда вы хотите уменьшить количество SQL-запросов, выполняемых во время пакетной обработки (batch processing).

**Состояния сущностей**

Как описано в [этой статье](https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/) ([рус.](http://akorsa.ru/2016/07/rukovodstvo-dlya-nachinayushhih-po-perehody-mezhdu-sostoyaniyami-v-jpa-hibernate/)), сущность в JPA или Hibernate может быть в одном из следующих состояний:

* *Transient (New)* — Новая
* *Managed (Persistent)* — Управляемая
* *Detached* — Отсоединенная
* *Removed (Deleted)* — Удаленная

Переход из одного состояния в другое осуществляется с помощью методов EntityManager или Session.  
Например, EntityManager из JPA предоставляет следующие методы перехода состояния сущности.

Session в Hibernate реализует все методы EntityManager из JPA и предоставляет несколько дополнительных методов для изменения состояния сущностей, таких как save, saveOrUpdate и update.

**Модель предметной области**

Давайте рассмотрим сущность Book, которая использует [Fluent API](https://vladmihalcea.com/fluent-api-entity-building-with-jpa-and-hibernate/):

@Entity(name = "Book")

@Table(name = "book")

public class Book {

@Id @GeneratedValue private Long id;

private String isbn;

private String title;

private String author;

Getter

Setter

Теперь посмотрим, как мы можем сохранить и обновить сущность с помощью JPA и Hibernate.

**Метод persist**

Чтобы изменить состояние сущности с *Transient (New)* на *Managed (Persisted)*, мы можем использовать метод persist, предлагаемый JPA EntityManager, который также наследуется в Hibernate Session.

*Метод persist инициирует событие PersistEvent, которое обрабатывается обработчиком DefaultPersistEventListener.*

Поэтому при выполнении следующего примера:

doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

entityManager.**persist**(book);

LOGGER.info("Persisting the Book entity with the id: {}", book.getId());

});

*Примечание переводчика: метод [doInJPA](http://in.relation.to/2016/09/15/integration-tests-and-java-8-lambdas/) выполняет операции в JPA-транзакции.*

Hibernate генерирует следующие операторы SQL:

CALL NEXT VALUE FOR hibernate\_sequence

-- Persisting the Book entity with the id: 1

INSERT INTO book (author, isbn, title, id ) VALUES ('Vlad Mihalcea', '978-9730228236', 'High-Performance Java Persistence', 1 )

Обратите внимание, что идентификатор (id) присваивается до присоединения сущности Book к текущему *Persistence Context*. Это необходимо поскольку управляемые сущности хранятся в структуре Map, в которой ключ формируется из типа сущности и идентификатора, а значение является ссылкой на сущность. Именно по этой причине JPA EntityManager и Hibernate Session также называются кэшем первого уровня (First-Level Cache).

При вызове метода persist сущность только присоединяется к текущему *Persistence Context*, и INSERT может быть отложен до вызова flush.

Единственным исключением является генератор IDENTITY, который запускает INSERT сразу, так как это единственный способ получить идентификатор сущности. По этой причине Hibernate не может использовать пакетные запросы INSERT для сущностей, использующих IDENTITY-идентификаторы. Дополнительные сведения о этом см. в [этой статье](https://vladmihalcea.com/how-to-batch-insert-and-update-statements-with-hibernate/).

**Метод save**

Специфичный для Hibernate метод save был в нём еще до появления JPA, с начала проекта Hibernate.

*Метод save инициирует событие SaveOrUpdateEvent, которое обрабатывается обработчиком DefaultSaveOrUpdateEventListener. Следовательно, метод save эквивалентен методам update и saveOrUpdate.*

Чтобы увидеть как работает метод save, рассмотрим следующий пример:

doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

Session session = entityManager.unwrap(Session.class);

Long id = (Long) session.**save**(book);

LOGGER.info("Saving the Book entity with the id: {}", id);

});

При выполнении приведенного выше примера Hibernate генерирует следующий SQL:

CALL NEXT VALUE FOR hibernate\_sequence

-- Saving the Book entity with the id: 1

INSERT INTO book (author, isbn, title, id ) VALUES ('Vlad Mihalcea', '978-9730228236', 'High-Performance Java Persistence', 1 )

Как вы видите, результат идентичен вызову метода persist. Однако, в отличие от persist, метод save возвращает идентификатор сущности.

**Метод update**

Hibernate-специфичный метод update предназначен для обхода механизма [dirty checking](https://vladmihalcea.com/the-anatomy-of-hibernate-dirty-checking/) ([рус.](http://akorsa.ru/2016/09/chto-takoe-dirty-checking-mehanizm-hibernate/)), и принудительного обновления сущности во время flush (сброса).

*Метод update инициирует событие SaveOrUpdateEvent, которое обрабатывается обработчиком DefaultSaveOrUpdateEventListener. Следовательно, метод update эквивалентен методам save и saveOrUpdate.*

Чтобы увидеть как работает метод update, рассмотрим пример, в котором в транзакции сохраняется сущность Book, затем изменяется, пока сущность находится в состоянии *Detached*, и после этого принудительно вызывается SQL UPDATE, используя метод update.

Book \_book = doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

entityManager.persist(book);

return book; });

LOGGER.info("Modifying the Book entity");

\_book.setTitle( "High-Performance Java Persistence, 2nd edition" );

doInJPA(entityManager -> {

Session session = entityManager.unwrap(Session.class);

session.**update**(\_book);

LOGGER.info("Updating the Book entity"); });

При выполнении приведенного выше примера Hibernate генерирует следующие операторы SQL:

CALL NEXT VALUE FOR hibernate\_sequence

INSERT INTO book (author, isbn, title, id) VALUES ('Vlad Mihalcea', '978-9730228236', 'High-Performance Java Persistence', 1)

-- Modifying the Book entity

-- Updating the Book entity

UPDATE book SET author = 'Vlad Mihalcea', isbn = '978-9730228236', title = 'High-Performance Java Persistence, 2nd edition' WHERE id = 1

Обратите внимание, что UPDATE выполняется во время сброса (flush) *Persistence Context*, прямо перед коммитом, и поэтому сначала логгируется сообщение «Updating the Book entity».

**Использование @SelectBeforeUpdate для предотвращения не нужных обновлений**

Теперь UPDATE будет всегда выполняться, даже если сущность не была изменена в то время, когда она была в состоянии *Detached*. Чтобы предотвратить это, вы можете использовать аннотацию Hibernate @SelectBeforeUpdate, которая вызовет SELECT для загрузки сущности, для использования в механизме dirty checking.

Итак, если мы аннотируем сущность Book аннотацией @SelectBeforeUpdate:

@Entity(name = "Book")

@Table(name = "book")

@**SelectBeforeUpdate** public class Book { // Код опущен для краткости }

И выполним следующий пример:

Book \_book = doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

entityManager.persist(book);

return book; });

doInJPA(entityManager -> {

Session session = entityManager.unwrap(Session.class);

session.update(\_book); });

Hibernate выполнит следующие инструкции SQL:

INSERT INTO book (author, isbn, title, id) VALUES ('Vlad Mihalcea', '978-9730228236', 'High-Performance Java Persistence', 1 )

SELECT b.id, b.author AS author2\_0\_, b.isbn AS isbn3\_0\_, b.title AS title4\_0\_ FROM book b WHERE b.id = 1

Обратите внимание, что на этот раз UPDATE не выполняется, поскольку механизм dirty checking обнаружил, что сущность не была изменена.

**Метод saveOrUpdate**

Hibernate-специфичный метод saveOrUpdate — это просто псевдоним для сохранения и обновления.

*Метод saveOrUpdate инициирует событие SaveOrUpdateEvent, которое обрабатывается обработчиком DefaultSaveOrUpdateEventListener. Следовательно, метод update эквивалентен методам save и saveOrUpdate.*

Теперь вы можете использовать saveOrUpdate, когда хотите сохранить сущность или принудительно выполнить UPDATE, как показано в следующем примере.

Book \_book = doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

Session session = entityManager.unwrap(Session.class);

session.saveOrUpdate(book);

return book; });

\_book.setTitle("High-Performance Java Persistence, 2nd edition");

doInJPA(entityManager -> {

Session session = entityManager.unwrap(Session.class);

session.**saveOrUpdate**(\_book); });

**Опасайтесь исключений NonUniqueObjectException**

Одна из проблем, которая может возникнуть с save, update и saveOrUpdate, заключается в том, что *Persistence Context* уже содержит ссылку на сущность с тем же идентификатором и того же типа:

Book \_book = doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

Session session = entityManager.unwrap(Session.class);

session.saveOrUpdate(book);

return book; });

\_book.setTitle( "High-Performance Java Persistence, 2nd edition" );

try {

doInJPA(entityManager -> {

Book book = entityManager.find(Book.class, \_book.getId());

Session session = entityManager.unwrap(Session.class);

session.**saveOrUpdate**(\_book);

});

} catch (NonUniqueObjectException e) {

LOGGER.error("The Persistence Context cannot hold " + "two representations of the same entity", e);

}

Теперь Hibernate бросит исключение NonUniqueObjectException, потому что второй EntityManager уже содержит объект Book с тем же идентификатором, что и тот, который мы передали для обновления, и *Persistence Context* не может содержать два представления одной и той же сущности.

org.hibernate.NonUniqueObjectException: A different object with the same identifier value was already associated with the session : [com.vladmihalcea.book.hpjp.hibernate.pc.Book#1] at org.hibernate.engine.internal.StatefulPersistenceContext.checkUniqueness(StatefulPersistenceContext.java:651) at org.hibernate.event.internal.DefaultSaveOrUpdateEventListener.performUpdate(DefaultSaveOrUpdateEventListener.java:284) at org.hibernate.event.internal.DefaultSaveOrUpdateEventListener.entityIsDetached(DefaultSaveOrUpdateEventListener.java:227) at org.hibernate.event.internal.DefaultSaveOrUpdateEventListener.performSaveOrUpdate(DefaultSaveOrUpdateEventListener.java:92) at org.hibernate.event.internal.DefaultSaveOrUpdateEventListener.onSaveOrUpdate(DefaultSaveOrUpdateEventListener.java:73) at org.hibernate.internal.SessionImpl.fireSaveOrUpdate(SessionImpl.java:682) at org.hibernate.internal.SessionImpl.saveOrUpdate(SessionImpl.java:674)

**Метод merge**

Чтобы избежать NonUniqueObjectException, необходимо использовать метод merge из JPA EntityManager, который унаследован также в Hibernate Session.

Как объяснено в [этой статье](https://vladmihalcea.com/jpa-persist-and-merge/) ([рус.](http://akorsa.ru/2016/07/kak-rabotayut-persist-i-merge-v-jpa/)), метод merge извлекает сущность из базы данных, если в *Persistence Context* не найдена ссылка на эту сущность, и копирует состояние detached-сущности, переданной в метод merge, в извлечённую сущность.

*Метод merge инициирует событие MergeEvent, которое обрабатывается обработчиком DefaultMergeEventListener.*

Чтобы увидеть, как работает метод merge, рассмотрим пример, в котором сущность Book сохраняется в транзакции, затем модифицируется, пока она находится в состоянии *Detached*, после этого detached-сущность передается в merge.

Book \_book = doInJPA(entityManager -> {

Book book = new Book()

.setIsbn("978-9730228236")

.setTitle("High-Performance Java Persistence")

.setAuthor("Vlad Mihalcea");

entityManager.persist(book);

return book; });

LOGGER.info("Modifying the Book entity");

\_book.setTitle("High-Performance Java Persistence, 2nd edition" );

doInJPA(entityManager -> {

Book book = entityManager.**merge**(\_book);

LOGGER.info("Merging the Book entity");

assertFalse(book == \_book); });

При выполнении приведенного выше примера Hibernate выполнил следующие операторы SQL:

INSERT INTO book (author, isbn, title, id ) VALUES ('Vlad Mihalcea', '978-9730228236', 'High-Performance Java Persistence', 1)

-- Modifying the Book entity

SELECT b.id, b.author AS author2\_0\_, b.isbn AS isbn3\_0\_, b.title AS title4\_0\_ FROM book b WHERE b.id = 1

-- Merging the Book entity

UPDATE book SET author = 'Vlad Mihalcea', isbn = '978-9730228236', title = 'High-Performance Java Persistence, 2nd edition' WHERE id = 1

Обратите внимание, что ссылка на сущность, возвращаемая merge, отличается от отсоединенной (detached), которую мы передали методу merge.

Хотя при копировании состояния detached-сущности лучше использовать JPA-метод merge, дополнительный SELECT может быть проблематичным при выполнении пакетной обработки.

По этой причине лучше использовать update, когда вы уверены, что в *Persistence Context* нет ссылки на эту сущность, и что detached-сущность была изменена. Дополнительные сведения по этой теме см. [этой статье](https://vladmihalcea.com/how-to-optimize-the-merge-operation-using-update-while-batching-with-jpa-and-hibernate/).