A beginner's guide to entity state transitions with JPA and Hibernate

JULY 30, 2014 / VLADMIHALCEA

(Last Updated On: January 29, 2018)

Introduction

Hibernate shifts the developer mindset from *SQL* statements to entity state transitions. Once an entity is actively managed by *Hibernate*, all changes are going to be automatically propagated to the database.

Manipulating domain model entities (along with their associations) is much easier than writing and maintaining *SQL* statements. Without an ORM tool, adding a new column requires modifying all associated *INSERT*/*UPDATE* statements.

But *Hibernate* is no silver bullet either. *Hibernate* doesn't free us from ever worrying about the actual executed *SQL* statements. Controlling *Hibernate* is not as straightforward as one might think and it's mandatory to check all *SQL* statements *Hibernate* executes on our behalf.

The entity states

As I previously mentioned, *Hibernate* monitors currently attached entities. But for an entity to become managed, it must be in the right entity state.

First we must define all entity states:

- **New (Transient)**

  A newly created object that hasn't ever been associated with a *Hibernate Session* (a.k.a *Persistence Context*) and is not mapped to any database table row is considered to be in the **New (Transient)** state.

  To become persisted we need to either explicitly call the EntityManager#persist method or make use of the transitive persistence mechanism.
- **Persistent (Managed)**

  A **persistent** entity has been associated with a database table row and it's being managed by the current running *Persistence Context*. Any change made to such entity is going to be detected and propagated to the database (during the *Session* flush-time). With *Hibernate*, we no longer have to execute *INSERT/UPDATE/DELETE* statements. *Hibernate* employs a transactional write-behind working style and changes are synchronized at the very last responsible moment, during the current *Session* flush-time.
- **Detached**

  Once the current running *Persistence Context* is closed all the previously managed entities become **detached**. Successive changes will no longer be tracked and no automatic database synchronization is going to happen.

To associate a **detached** entity to an active *Hibernate Session*, you can choose one of the following options:

- **Reattaching**
  Hibernate (but not *JPA* 2.1) supports reattaching through the Session#update method.

  A *Hibernate Session* can only associate one *Entity* object for a given database row. This is because the *Persistence Context* acts as an in-memory cache (first level cache) and only one value (entity) is associated to a given key (entity type and database identifier).

  An entity can be reattached only if there is no other *JVM* object (matching the same database row) already associated to the current *Hibernate Session*.
- **Merging**
  The merge operaration is going to copy the **detached** entity state (source) to a managed entity instance (destination). If the merging entity has no equivalent in the current *Session*, one will be fetched from the database.

  The **detached** object instance will continue to remain detached even after the merge operation.

- **Removed**

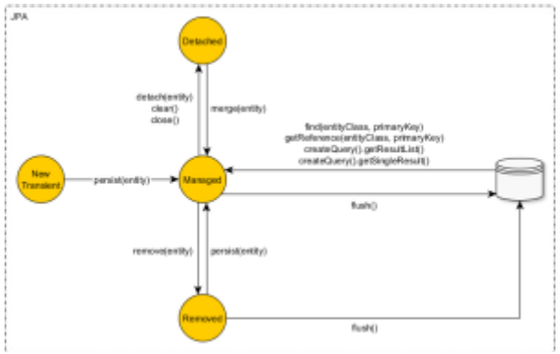
  Although *JPA* demands that managed entities only are allowed to be removed, *Hibernate* can also delete detached entities (but only through a Session#delete method call).

  A removed entity is only scheduled for deletion and the actual database *DELETE* statement will be executed during *Session* flush-time.

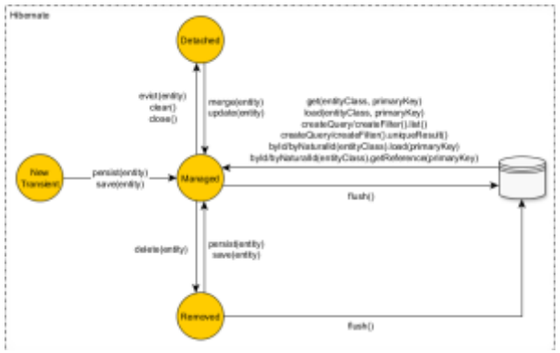
Entity state transitions

To change one *Entity* state, we need to use one of the following entity management interfaces:

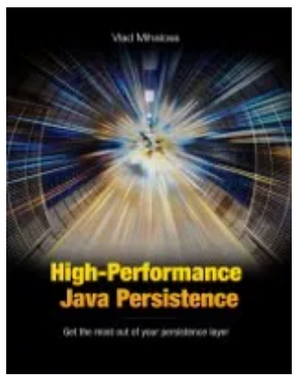
- EntityManager

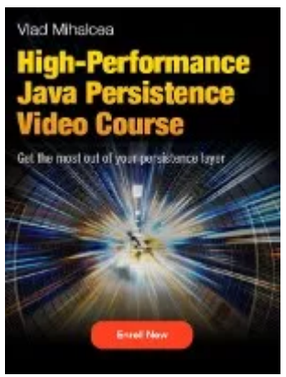


- Session



If you enjoyed this article, I bet you are going to love my **Book** and **Video Courses** as well.





Conclusion

These interfaces define the entity state transition operations we must explicitly call to notify *Hibernate* of the entity state change. At flush-time the entity state transition is materialized into a database *DML* statement.

For more about how to use persist and merge effectively, you should read this article as well.

Related

How does AUTO flush strategy work in JPA and Hibernate
August 13, 2014
In "Hibernate"

A beginner's guide to flush strategies in JPA and Hibernate
August 7, 2014
In "Hibernate"

How do JPA and Hibernate define the AUTO flush mode
January 19, 2016
In "Hibernate"

Categories: Hibernate Tags: entity state, hibernate, jpa, Training, Tutorial

← Hibernate pooled and pooled-lo identifier generators

A beginner's guide to flush strategies in JPA and Hibernate →

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Sign me up for the newsletter!

Post Comment

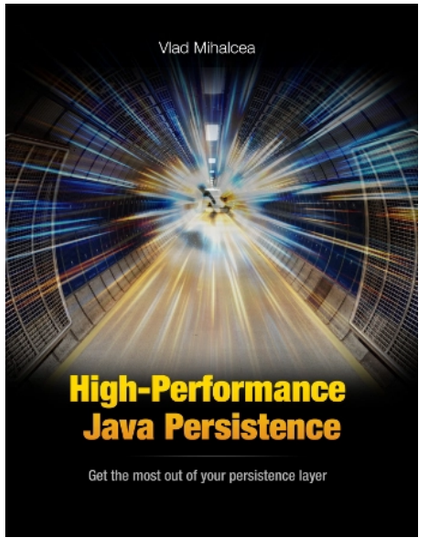
Notify me of follow-up comments by email.

This site uses Akismet to reduce spam. Learn how your comment data is processed.

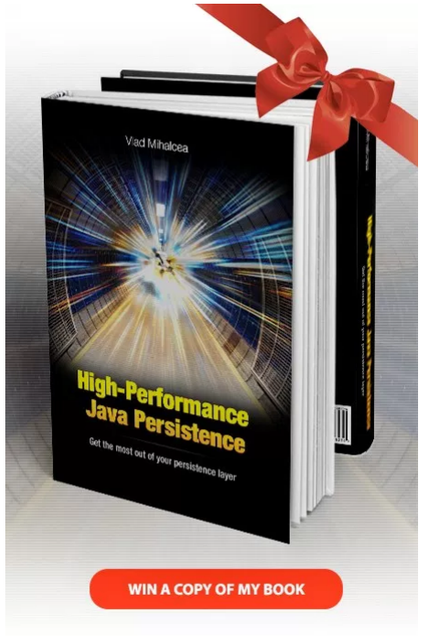
JDK.IO WORKSHOP – COPENHAGEN

VIDEO COURSE

HIGH-PERFORMANCE JAVA PERSISTENCE



WIN A COPY OF MY BOOK!

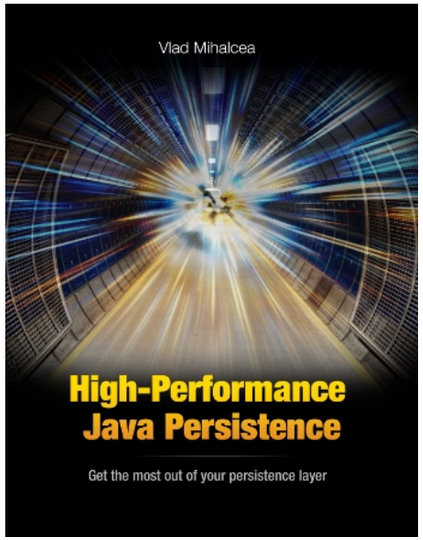


HIBERNATE PERFORMANCE TUNING TIPS

HYPERSISTENCE

WIN A COPY OF MY BOOK!

HIGH-PERFORMANCE JAVA PERSISTENCE



VIDEO COURSE

JDK.IO WORKSHOP – COPENHAGEN

JDKIO
0xCAFEBABE
CONFERENCE
2018

WORKSHOP

High-Performance Java Persistence

RSS - Posts

RSS - Comments

ABOUT

About

Privacy Policy

Terms of Service