

The JPA Entity Lifecycle

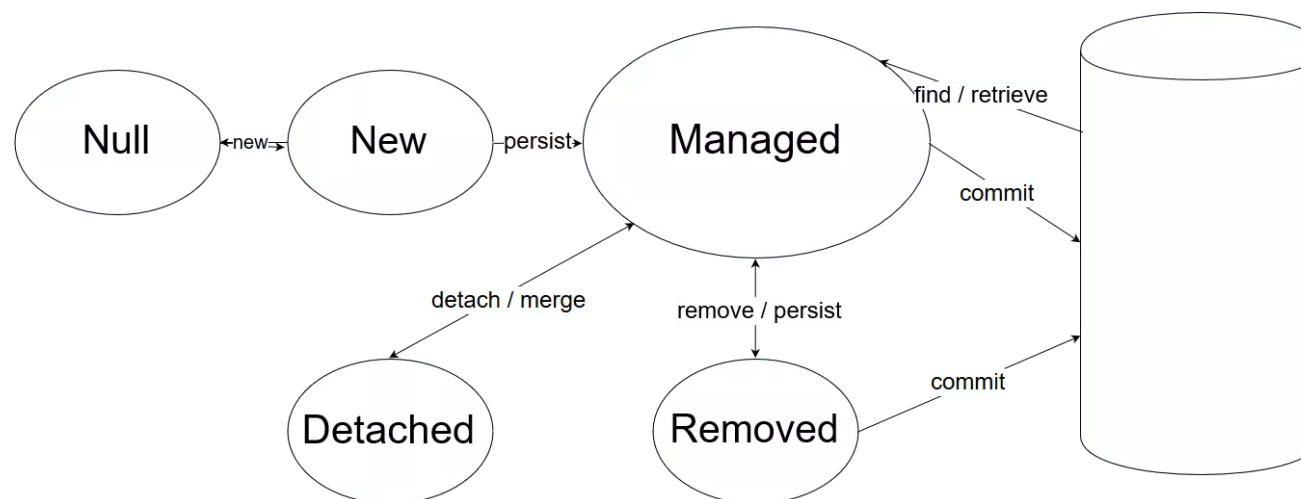
This look at the JPA entity lifecycle explores the lifecycle itself, the five stages of entity management, and callback methods on entity objects.

by **Martin Farrell**  MVB · Sep. 06, 17 · Java Zone · Tutorial

Download Microservices for Java Developers: A hands-on introduction to frameworks and containers. Brought to you in partnership with Red Hat.

The last blog post about the JPA EntityManager didn't cover the JPA entity lifecycle. This post builds on the original one by covering the JPA Entity lifecycle and associated lifecycle events.

JPA Entity Lifecycle



As a reminder, the purpose of the EntityManager is to the relationship between the JPA entity and the underlying datasource.

The above diagram shows the 5 key stages of JPA entity management:

- Object Doesn't Exist – This is a null object

```
1  MyObject myObject = null;
```

- New Object – Not associated with the EntityManager, and doesn't exist on database

```
1  MyObject myObject = new MyObject();
```

- Managed – This is the stage where the object becomes persisted and managed by the EntityManager. To do this, we need to call the persist method from within a transaction. The object is then persisted to the database when the commit method is called

```
1          entityManager.getTransaction().begin();
2          MyObject myObject = new MyObject();
3          entityManager.persist(myObject);
4          entityManager.getTransaction().commit();
```

- Detached – This state removes the object from the EntityManager, but the object **still** exists on the database. Some EntityManager methods on a detached object will result in an IllegalArgumentException. The object can be reattached to the EntityManager through the merge method

```
1  entityManager.detach(myObject);
```

- Removed – Deletes the object from the database. Like persist, this also needs to take place inside a transaction.

```
1          entityManager.getTransaction().begin();
2          entityManager.remove(myObject);
3          entityManager.getTransaction().commit();
```

Callback Methods on JPA Entities

The final part of the JPA lifecycle event is the optional callback methods, and their associated annotations:

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

• @POSTLOAD

The pre-methods are executed before the associated method action is executed (i.e. @PrePersist is executed before em.persist).

The post-methods are executed after the associated method action is executed (i.e. @PostPerist is executed after em.persist)

If either the pre- or post- methods/annotations throw an exception then the transaction will be rolled back.

What Can You Use These Callback Events For?

The pre- and post-persist methods are useful for setting timestamps for auditing or setting default values.

Download Building Reactive Microservices in Java: Asynchronous and Event-Based Application Design. Brought to you in partnership with Red Hat.

Like This Article? Read More From DZone



Contexts and Dependency Injection (CDI): Eager Extensions



What's New in HTTP-RPC?




Java Persistence API Introduction (Part 1)



**Free DZone Refcard
Getting Started With Kotlin**

Topics: [JAVA](#) , [JPA](#) , [LIFECYCLE MANAGEMENT](#) , [CALLBACK METHOD](#) , [TUTORIAL](#)

Published at DZone with permission of Martin Farrell , DZone MVB. [See the original article here.](#) 
Opinions expressed by DZone contributors are their own.

Java Partner Resources

Deep insight into your code with IntelliJ IDEA.

JetBrains



Migrating to Microservice Databases

Red Hat Developer Program



Modern Java EE Design Patterns: Building Scalable Architecture for Sustainable Enterprise Development

Red Hat Developer Program



Build vs Buy a Data Quality Solution: Which is Best for You?

Melissa Data

