JPA Implementation Patterns: Retrieving Entities

by Vincent Partington MVB · Jul. 29, 09 · Java Zone · Not set

---

Last time I talked about how to save an entity. And once we've saved an entity we'd also like to retrieve it. Compared to saving entities, retrieving entities is actually rather simple. So simple I doubted whether there would be much point in writing this blog 😉 *. However we did use a few nice patterns when writing code for this. And I'm interested to hear what patterns you use to retrieve entities.*

Basically, there are two ways to retrieve an entity with JPA:

- EntityManager.find will find an entity by its id or return null when that entity does not exists.
- If you pass a query string specified in Java Persistence Query Language to EntityManager.createQuery it will return a Query object that can then be executed to return a list of entities or a single entity.

A Query object can also be created by referring to a named query (using EntityManager.createNamedQuery), or by passing in an SQL query (using one of the three three flavours of EntityManager.createNativeQuery). And while the name implies otherwise, a Query can also be used to execute an update or delete statement.

A named query may seem like a nice way to keep the query with the entities it queries, I've found that not to work out very well. Most queries need parameters to be set with one of the variants of Query.setParameter. Keeping the query and the code that sets these parameters together makes them both easier to understand. That is why I keep them together in the DAO and shy away from using named queries.

A convention I've found to be useful is to differentiate between *finding* an entity and *getting* an entity. In the first case null is returned when an entity cannot be found, while in the latter case an exception is thrown. Using the latter method when your code expects an entity to be present prevents NullPointerExceptions from popping up later.

Finding and getting a single entity by id

An implementation of this pattern for the JpaDao base class we discussed a few blogs ago can look like this (I've included the find method for contrast):

```
public E findById(K id) {
	return entityManager.find(entityClass, id);
}

public E getById(K id) throws EntityNotFoundException {
	E entity = entityManager.find(entityClass, id);
	if (entity == null) {
		throw new EntityNotFoundException(
			"Entity " + entityClass.getName() + " with id " + id + " not found");
	}
	return entity;
}
```

Of course you'd also need to add this new method to the Dao interface:

```
E getById(K id);
```

Finding and getting a single entity with a query

A similar distinction can be made when we use a query to look for a single entity. The findOrderSubmittedAt method below will return null when the entity cannot be found by the query. The getOrderSubmittedAt method throws a NoResultException. Both methods will throw a NonUniqueResultException if more than one result is returned. To keep the getOrderSubmittedAt method consistent with the findById method we could map the NoResultException to an EntityNotFoundException. But since there are both unchecked exceptions, there is no real need.

Since these methods apply only to the Order object, there are a part of the JpaOrderDao:

```
public Order findOrderSubmittedAt(Date date) throws NonUniqueResultException {
	Query q = entityManager.createQuery(
		"SELECT e FROM " + entityClass.getName() + " e WHERE date = :date_at");
	q.setParameter("date_at", date);
	try {
		return (Order) q.getSingleResult();
```

```
        } catch (NoResultException exc) {
                return null;
        }
}

public Order getOrderSubmittedAt(Date date) throws NoResultException, NonUniqueResultException {
        Query q = entityManager.createQuery(
                "SELECT e FROM " + entityClass.getName() + " e WHERE date = :date_at");
        q.setParameter("date_at", date);
        return (Order) q.getSingleResult();
}
```

Adding the correct methods to the OrderDao interface is left as an exercise for the reader. 😉

Finding multiple entities with a query

Of course we also want to be able to find more than one entity. In that case I've found there to be no useful distinction between *getting* and *finding*. The findOrdersSubmittedSince method just return a list of entities found. That list can contain zero, one or more entities. See the following code:

```
public List<Order> findOrdersSubmittedSince(Date date) {
        Query q = entityManager.createQuery(
                        "SELECT e FROM " + entityClass.getName() + " e WHERE date >= :date_since");
        q.setParameter("date_since", date);
        return (List<Order>) q.getResultList();
}
```

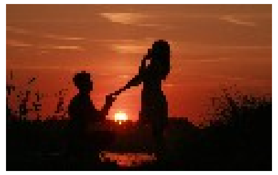
Observant readers will note that this method was already present in the first version of the JpaOrderDao.

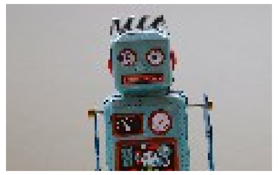
So while retrieving entities is pretty simple, there are a few patterns you can stick to when implementing finders and getters. Of course I'd be interested to know how you handle this in your code.

P.S. JPA 1.0 does not support it yet, but JPA 2.0 will include a Criteria API. The Criteria API will allow you to dynamically build JPA queries. Criteria queries are more flexible than string queries so you can build them depending on input in a search form. And because you define them using domain objects, they are easier to maintain as references to domain objects get refactored automatically. Unfortunately the Criteria API requires you to refer to your entity's properties by name, so your IDE will not help you when you rename those.

From http://blog.xebia.com

Like This Article? Read More From DZone


Proposed Jakarta EE Design Principles


Advanced Research Initiatives: Focus on the Future

How to Configure NGINX High Availability Cluster Using Pacemaker on Ubuntu 16.04

Free DZone Refcard
Getting Started With Kotlin

Topics:

Opinions expressed by DZone contributors are their own.

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