Published on *Java.net* (http://www.java.net)

Don't use @PersistenceContext in a web app...

By *ss141213*
Created *2005-12-19 15:18*
Posted by ss141213 [1] on December 19, 2005 at 3:18 PM EST

It's a common mistake to inject an EntityManager into a web application that uses Java Persistence API. Let's discuss why?

I wrote an example web application [2] that uses Java Persistence API [3]. My servlet code looked like this:

```
public class RegistrationServlet extends HttpServlet {
   // inject default EntityManager
   @javax.persistence.PersistenceContext private EntityManager em;
   @Resource private UserTransaction utx;
   public void service ( HttpServletRequest req , HttpServletResponse resp)
        throws ServletException, IOException {
      ...
      utx.begin();
      em.persist(credential);
      utx.commit();
      ...
   }
   ...
}
```

This code worked, but only by chance. I did not realize that I have committed a horrible mistake in my servlet code. As the servlet spec [4] suggests, unless explicitly mentioned in the deployment descriptor (web.xml) as **SingleThreadModel**, a single servlet instance by default can be shared to serve multiple requests concurrently. i.e. multiple threads can simultaneously enter the service() method of our servlet because we have not marked the service() as **synchronized**. As a result multiple threads will share the same **PersistenceContext** object via the instance variable *em*. A persistence context is not required to be thread safe as per the spec and is typically designed not to be used concurrently. Because the behavior is timing dependent and I had a tiny example, I did not see this issue during testing.

What is the fix

If we can't use @PersistenceContext [5] to inject an EntityManager [6], what is the fix? The fix is to either use @PersistenceUnit [7] to inject an EntityManagerFactory [8] and use it to get hold of an EntityManager,
or
declare a dependency on an EntityManager and use JNDI to look it up.
The former one is called **application managed entity manager** because application manages the life cycle (by calling EntityManagerFactory.create & EntityManager.close) where as the later one is called **container managed entity manager** because container manages life cycle. More discussion on differences between container managed vs. application managed will be done in a later article. Let's discuss the fix using both the approaches. Let's discuss each approach using code samples.

Container Managed Entity Manager

There are a couple of ways to use JNDI lookup of entity manager:
a) via annotation:

```
@PersistenceContext(name="persistence/LogicalName", unitName="ActualPUNameAsItAppearsInPersistence.xml")
public class RegistrationServlet extends HttpServlet {
   @Resource private UserTransaction utx;
   public void service ( HttpServletRequest req , HttpServletResponse resp)
        throws ServletException, IOException {
```

```
      Context envCtx = InitialContext().lookup("java:comp/env");
      EntityManager em =  (EntityManager) envCtx.lookup("persistence/LogicalName");
      ...
      utx.begin();
      em.persist(credential);
      utx.commit();
      ...
   }
   ...
}
```

Note that there is no injection going on here since the annotation @PersistenceContext appears at the class level. This is an alternative to declaring the persistence context dependency via a persistence-context-ref in web.xml as discussed below (in option #b).

b) via persistence-context-ref in web.xml

In web.xml, add an element like this:

```
 < persistence-context-ref>
   < persistence-context-ref-name>
       persistence/LogicalName
   </persistence-context-ref>
   < persistence-unit-name>
       ActualPUNameAsItAppearsInPersistence.xml
   </persistence-unit-name>
</persistence-context-ref>
```

Now do a JNDI lookup in your code as shown below:

```
public class RegistrationServlet extends HttpServlet {
   @Resource private UserTransaction utx;
   public void service ( HttpServletRequest req , HttpServletResponse resp)
        throws ServletException, IOException {
      Context envCtx = InitialContext().lookup("java:comp/env");
      EntityManager em =  (EntityManager) envCtx.lookup("persistence/LogicalName");
      ...
      utx.begin();
      em.persist(credential);
      utx.commit();
      ...
   }
   ...
}
```

While using container managed entity manager (whether option #a or #b is used), we did not call em.close() because container is managing the life cycle of underlying persistence context. We also did not have to call utx.rollback() because web container would automatically rollback a transaction at the end of http request processing if servlet does not end the tx.

Application Managed Entity Manager

```
public class RegistrationServlet extends HttpServlet {
   // inject EntityManagerfactory
   @javax.persistence.PersistenceUnit private EntityManagerFactory emf;
   @Resource private UserTransaction utx;
   public void service ( HttpServletRequest req , HttpServletResponse resp)
        throws ServletException, IOException {
      EntityManager em = emf.createEntityManager();
      try {
          ...
          utx.begin();
          em.persist(credential);
          utx.commit();
          ...
      } catch (Exception e){
          try {
              utx.rollback();
          } catch (Exception e) {}
      } finally {
          em.close();
      }
   }
```

```
    . . .
}
```

See we call close() to close the EntityManager. More over note the use of try catch finally block. Since em.close() can not be called as long as the associated transaction is complete either by calling commit() or rollback(), we have to write those try catch finally.

Is it mentioned any where in the spec?

In Transaction Management chapter, section #4.2.3 of Java EE 5 proposed final draft spec [9], it is mentioned that:

In web components not implementing SingleThreadModel, transactional resource objects should not be stored in class instance fields, and should be acquired and released within the same invocation of the service method.

If you want to draw an analogy with JDBC world then EntityManager is like a **Connection**, where as EntityManagerFactory is like a **DataSource**. So EntityManager (or a PersistenceContext) which is a transactional resource should not be stored in a instance field and hence should not be injecte into a web app that does not implement SingleThreadModel.EntityManagerFactory is thread safe, so it can be injected into the servlet.

What is the performance over head?

Creation of a EntityManagerFactory is typically a costly operation. But creation of EntityManager is not. So creating an EntityManager is inside service() does not have negative impact on performance.

What about thread safety of UserTransaction?

If you see the code above, it still injects a UserTransaction object and stores in an instance field. That is not issue because it is a stateless object and can be shared across multiple threads. Looking at the javadocs for UserTransaction [10], it is clear that it by itself does not represent the transaction object, instead it is an interface to the underlying transaction manager to begin a new transaction and associate that with current thread; and end a transaction associated with current thread.

More articles about glassfish [11] persistence [12].

Related Topics >> J2EE [13]

J2EE

Source URL: http://www.java.net/blog/ss141213/archive/2005/12/dont_use_persis.html

Links:
[1] http://www.java.net/blog/ss141213
[2] http://weblogs.java.net/blog/ss141213/archive/2005/12/introduction_to.html
[3] https://glassfish.dev.java.net/javaee5/persistence/entity-persistence-support.html
[4] http://jcp.org/en/jsr/detail?id=154
[5] https://glassfish.dev.java.net/source/browse/glassfish/persistence-api/src/java/javax/persistence/PersistenceContext.java?rev=HEAD&view=markup
[6] https://glassfish.dev.java.net/source/browse/glassfish/persistence-api/src/java/javax/persistence/EntityManager.java?rev=HEAD&view=markup
[7] https://glassfish.dev.java.net/source/browse/glassfish/persistence-api/src/java/javax/persistence/PersistenceUnit.java?rev=HEAD&view=markup
[8] https://glassfish.dev.java.net/source/browse/glassfish/persistence-api/src/java/javax/persistence/EntityManagerFactory.java?rev=HEAD&view=markup
[9] http://jcp.org/en/jsr/detail?id=244
[10] http://java.sun.com/j2ee/1.4/docs/api/javax/transaction/UserTransaction.html
[11] http://technorati.com/tag/glassfish

[12] http://technorati.com/tag/persistence
[13] http://www.java.net/topic/j2ee