Understand Persistence Context Collision in JEE

Sheng Wang 1:01 AM JEE, JPA No Comments

When deal with persistence layer in JEE application, persistence context is the magic that make your entity instances different from a normal POJO. Persistence context manages the entity instances to make it finally synchronized with database.

The persistence context collision happens when invoking stateful session bean from a stateless session method if both beans has their *EntityManager* defined. (BTW, Use stateful bean from stateless bean itself is not a good idea, but this article just focuses on persistence context)

In this article, for simplicity, when we mention stateless bean, we mean a stateless bean with *EntityManager* variable and its methods operate on the persistence. When we mention stateful bean, we mean a stateful bean with *EntityManager* variable and its methods operate on persistence. In short we only talk about beans in persistence layer.

0. Basic rules about persistence context

Here are some basic rules about persistence context:

There will be only one active persistence context at any time for a transaction.

JEE container can propagate persistence context between different *EntityManager* variables in a single transaction. (Different *EntityManager* field variables of different beans can use the same persistence context).

Stateless bean usually use **transaction-scoped** persistence context, which means when the transaction is over, the persistence context is also gone. (When the bean's method is over, transaction is gone, persistence context is also gone)

Stateful bean usually use **extended** persistence context. The persistence is created when the bean instance is created and only destroy when the stateful bean is removed. The extended persistence context will be associated to a transaction when a method of stateful bean is called. When the method is over, transaction is gone, but the persistence context doesn't go with the transaction but stay for the next transaction until the whole stateful bean is removed by the container.

Stateful bean always use his own **extended** persistence context, if the active persistence context is not the **extended** one, *javax.ejb.EJBException* will be thrown, this is called **persistence context collision**. This usually happens when call a stateful bean from a stateless bean.

Let see a simple example of persistence context collision.

1. Define pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.shengwang.demo</groupId>
  <artifactId>jee-persistence-context-collision</artifactId>
  <packaging>war</packaging>
```

```xml
  <version>1.0</version>
  <name>jee-persistence-context-collision Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!-- where glassfish 4.0 is installed -->
    <glassfish.home>D:\glassfish4.0</glassfish.home>
  </properties>

  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>jee-persistence-context-collision</finalName>
    <plugins>
      <!-- Use Java 1.7 -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>

      <!-- use mvn cargo:run to deploy and start server-->
      <plugin>
        <groupId>org.codehaus.cargo</groupId>
        <artifactId>cargo-maven2-plugin</artifactId>
        <inherited>true</inherited>
        <configuration>
          <container>
            <containerId>glassfish4x</containerId>
            <type>installed</type>
            <home>${glassfish.home}</home>
          </container>
          <configuration>
            <type>existing</type>
            <home>${glassfish.home}/glassfish/domains</home>
          </configuration>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

The pom has 1 dependency for JEE 7 and 2 plugins. The first plugin specify the Java version (Java 1.7), the second one is for deploying package to local glassfish 4.0 with maven command-line. Demo use glassfish 4.0 as JEE container.

2. Define entity class

A very simple entity class Client.java with 2 fields, int clientId and String name.

```java
package com.shengwang.demo.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Client {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "CLIENT_ID")
  private int clientId;

  private String name;

  /* getter setter omitted */

  @Override
  public String toString() {
    return "{" + clientId + "," + name + "}";
  }
}
```

The entity is trivial.

3. Define stateful bean

```java
package com.shengwang.demo.session;

import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.PersistenceContextType;

import com.shengwang.demo.entity.Client;

@Stateful
public class MyStatefulBean {
  @PersistenceContext (type=PersistenceContextType.EXTENDED) // extended
  EntityManager em;
  Client client;

  public String changeClientName() {
    if (client == null) {
      client = em.find(Client.class, 1);
    }
    client.setName(client.getName() + "_" + "hello");

    return client.toString();
  }
}
```

This stateful bean just for demo, so doesn't make much sense. Its only method change the first client's name, add suffix to the name.

4. Define stateless bean

```java
package com.shengwang.demo.session;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.shengwang.demo.entity.Client;

@Stateless
public class MyStatelessBean {
  @PersistenceContext
  EntityManager em;

  @EJB
  MyStatefulBean statefulBean;

  public String methodA() {
    String c1 = em.find(Client.class, 1).toString(); // any operation of em
    String c2 = statefulBean.changeClientName();

    return c1 + "," + c2;
  }
}
```

The stateless bean has one method, methodA. Call the em.find() first then call the stateful beans's method. Calling methodA() will cause **persistence context collision**!

Why? Because the transaction-scoped persistence is created **lazy**, so it(PC-A) will be created only when *em.find()* called, and this persistence context is now the active persistence context. But the stateful bean with extended persistence context init persistence context **eagerly**, which means the stateful bean already has a persistence context(PC-B) when it initialized. Now the active persistence context propagated to stateful bean,PC-A, is not the extended persistence context PC-B. So collision happens and exception will be thrown. We can see this when we try to run it below.

5. Define a servlet as EJB client

To use the beans we defined above, let's define a simple servlet.

```java
package com.shengwang.demo.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.shengwang.demo.session.MyStatefulBean;
import com.shengwang.demo.session.MyStatelessBean;


@WebServlet(name = "testServlet", urlPatterns = { "/test" })
public class TestServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  @EJB
  MyStatelessBean statelessBean;

  @EJB
  MyStatefulBean statefulBean;

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String str = statelessBean.methodA();

    PrintWriter out = resp.getWriter();
    out.printf("%s", str);
    out.flush();
  }
}
```

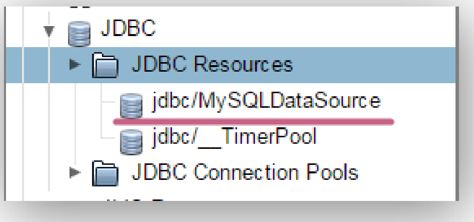
In the servlet, the stateless bean's only method is invoked.

6. Config persistence.xml

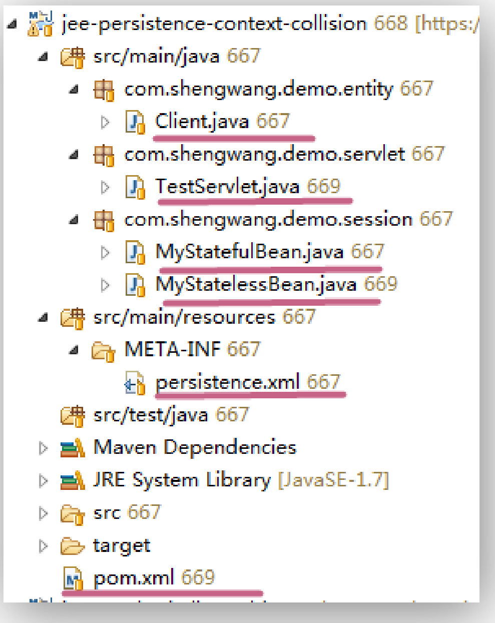
Usually the META-INFO/persistence.xml is simple for JEE applications, tell the server which data source application wants to use.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="demo-persistence-unit" transaction-type="JTA">
    <jta-data-source>jdbc/MySQLDataSource</jta-data-source>
  </persistence-unit>
</persistence>
```

There's a data source on server with JNDI name jdbc/MySqlDataSource.



Finally, before running the demo, the project hierarchy looks like below.

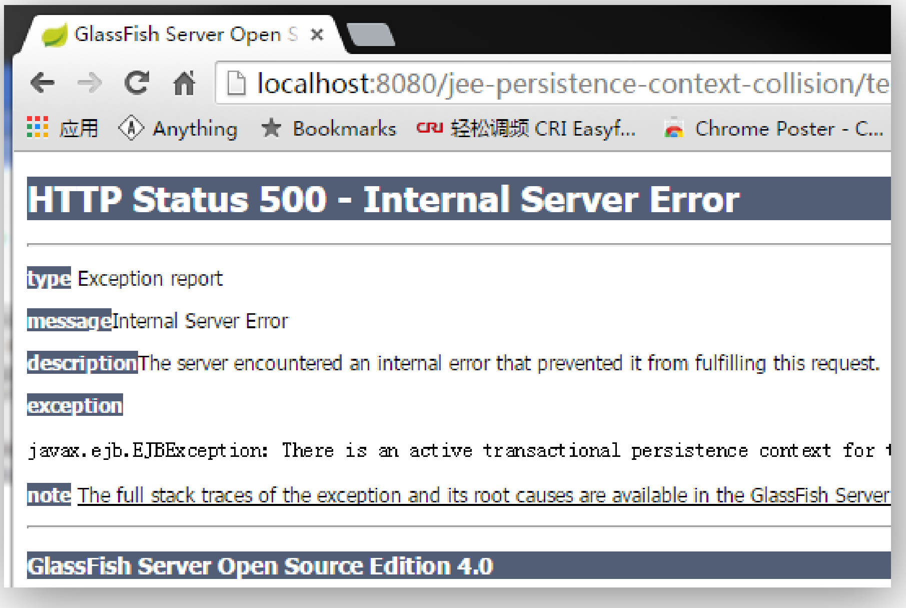


7. Run the demo

Run the demo with mvn command-line:

```
mvn clean verify cargo:run
```

This command will start glassfish server and deploy our application to it. Then access the servlet by browser. You can see the exception: **javax.ejb.EJBException: There is an active transactional persistence context for the same EntityManagerFactory as the current stateful session bean's extended persistence context**



HTTP Status 500 - Internal Server Error

type Exception report

message Internal Server Error

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
javax.ejb.EJBException: There is an active transactional persistence context for t
```

note The full stack traces of the exception and its root causes are available in the GlassFish Server

GlassFish Server Open Source Edition 4.0

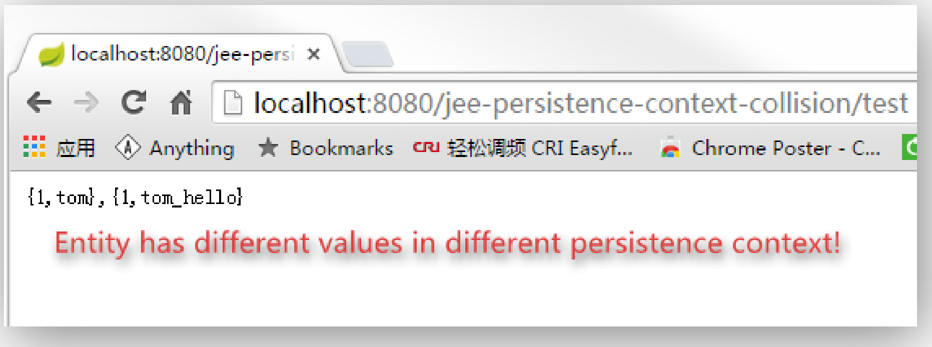
8. What's more

What will happen if we make a little change to the stateless bean.

```java
@Stateless
public class MyStatelessBean {
  @PersistenceContext
  EntityManager em;

  @EJB
  MyStatefulBean statefulBean;

  public String methodA() {

    // These 2 lines switch order
    String c2 = statefulBean.changeClientName();
    String c1 = em.find(Client.class, 1).toString();

    return c1 + "," + c2;
  }
}
```

The order of two lines are swithed. Now the demo can run without any exception.

Why? Because when invoke the stateful bean there is no persistence context yet, so the stateful bean's method can use its own extended persistence context to run. Then the *em.find()* is invoked, a transaction-scoped persistence create when first em operation invoked. So these 2 lines use different persistence context! If this is understood, then then output also makes sense to you.



```
{1, tom}, {1, tom_hello}
```

Entity has different values in different persistence context!

Furthermore, If you really need to call stateful bean from stateless bean in it original order, which cause the persistence context collision. Here are some detours (think twice when you want to do this):

1. Mark stateful bean (or its method invoked by stateless bean) **REQUIRE_NEW** for TransactionAttributeType. So there will be 2 transactions for stateless and stateful bean respectively. Each transaction can has its own persistence context. Similar scenario like above, entities may have different values on different context.

2. Mark stateful bean (or its method invoked by stateless bean) **NOT_SUPPORT** for TransactionAttributeType. If only invoke stateful bean's read-only operation from stateless bean. (Servers like Glassfish may need to config JDBC connection pool to enable Non Transactional Connections)

3. Use Application-Managed EntityManager instead of the default Container-Managed Entity Manager in the stateful bean.

0 comments:

Post a Comment