HOWTO - Use OJB in clustered environments

by Jason McKerr

Table of contents

1. How to use OJB in clustered environments

Object/Relational Bridge will work fine in environments that require robustness features such as load-balancing, failover, and clustering. However, there are some important steps that you need to take in order for your data to be secure, and to prevent isolation failures. These steps are outlined below.

I have tested this in a number of environments, and with Servlet engines and J2EE environments. If you run into problems, please post questions to the OJB users mail list.

This outline for clustering is based on an email from the OJB Users Mail List: This is that mail.

2. Three steps to clustering your OJB application

A lot of people keep asking for robust ways to run their OJB engines in multi-VM or clustered environments. This email covers how to do that safely and securely using Open Symphony's OSCache caching product.

OSCache is a high-performance, robust caching product that supports clustering. I've been using it for a while in production and it is excellent.

Back to the Topic: There are three main things that you should do in order to make your OJB application ready for using a cache in a multi-VM or distributed environment.

2.1. First: Take care of the sequence manager

that you define within *jdbc-connection-descriptor* element in your repository.xml file. If none was set OJB use per default the `SequenceManagerHighLowImpl` sequence manager implementation.

Note:

As of Release Candidate 5 (rc5), you can use SequenceManagerHighLowImpl in distributed (non-managed) environments. The SequenceManagerHighLowImpl now supports its own optimistic locking that makes the implementation cluster aware by versioning an entry in the OJB_HL_SEQ table.

However, the SequenceManagerHighLowImpl has not been heavily tested in clustered environments, so if you want absolute security use an sequence manager implementation which delegates key generation to database.

If your database supports database based sequence key generation (like PostgreSQL, Oracle, ...) it's recommended to use `SequenceManagerNextValImpl` (supports database based sequence keys). Using this sequence manager will prevent conflicts in multi-vm or clustered environments. More about sequence manager here.

2.1.1. Handling sequence names

If you are using `SequenceManagerNextValImpl` you have two possibilities:

- Do it by your own:
  - Create a sequence object in your database.
    - An Oracle sequence lookslike: "create sequence ackSequence increment by 1 start with 1;"
    - A Postgres sequence looks like: "CREATE SEQUENCE ackSequence START 1";
  - For other databases you're on your own.
  - To tell OJB to use that sequence for your table add in your repository.xml the sequence

name to the field-descriptor for your table's primary key field:

```
<field-descriptor
name="ackID"
column="ACKID"
jdbc-type="INTEGER"
primarykey="true"
autoincrement="true"
sequence-name="ackSequence"
/>
```

- Let OJB do that job for you:
  The `SequenceManagerNextValImpl` implementation create the sequence in database automatic if none was found. If you don't want to declare a `sequence-name` attribute in your *field-descriptor*, you can enable an automatic sequence name building by setting a specific *custom-attribute* , then `SequenceManagerNextValImpl` build an internal sequence name if none was found.

```
<sequence-manager
className="org.apache.ojb.broker.util.sequence.SequenceManagerNextValImpl">
    <attribute attribute-name="autoNaming" attribute-value="true"/>
</sequence-manager>
```

More about sequence manager here.

2.2. Second: Enable optimistic locking

You need to secure the data at the database. Thomas Mahler (lead OJB developer and considerable ORM guru) recommended in one email that you use the Optimistic Locking feature that is provided by OJB when using the **PB API** in a clustered environment. Sounds good to me. To do this you need to do three small steps:

Note:
When using one of the top-level API in most cases Pessimistic (Object) Locking is supported. In that case it is recommended to use a *distributed lock management* instead of optimistic locking. More information about ODMG API and Locking here.

- Add a database column to your table that is either an INTEGER or a TIMESTAMP
- Add the field to your java class, and getter/setter methods (depends on the used PersistentField implementation):

```
private Integer ackOptLock;

public Integer getAckOptLock()
{
return ackOptLock;
}

public void setAckOptLock(Integer ackOptLock)
{
this.ackOptLock = ackOptLock;
}
```

- Add the column to your table in your repository:

```
<field-descriptor
    name="ackOptLock"
    column="ACKOPTLOCK"
    jdbc-type="INTEGER"
    locking="true"/>
```

Now OJB will handle the locking for you. No explicit transactional code necessary!

2.3. Do The Cache

The detailed steps to setup the *OSCache* can be found in caching document

You're ready to go! Now just create two instances of your application and see how they communicate at the cache level. Very cool.

3. Notes

- For J2EE/Servlet users: I have tested this on a number of different application servers. If you're having problems with your engine, post an email to the OJB Users mail list.
- OSCache also supports JMS for clustering here, which I haven't covered. If you either don't have access to a multicast network, or just plain like JMS, please refer to the OSCache documentation for help with that, see OSCache Clustering with JMS).
- I have also tested this with Tangosol Coherence. Please refer to this Blog entry for that setup: Coherence Setup
- OJB also has ships with JCS. Feel free to try that one out on your own.