Enqueue Server in SAP NetWeaver 04 Java

Overview

SAP NetWeaver 04

Copyright

Enqueue Server in SAP NetWeaver 04 Java Overview

Abstract

The text spotlights the role of the Enqueue Server Java in the SAP NetWeaver 04 server infrastructure. Among others, based on the Enqueue Server Java, SAP NetWeaver provides a server-side locking API to Java applications as an alternative to the traditional database locking technique.

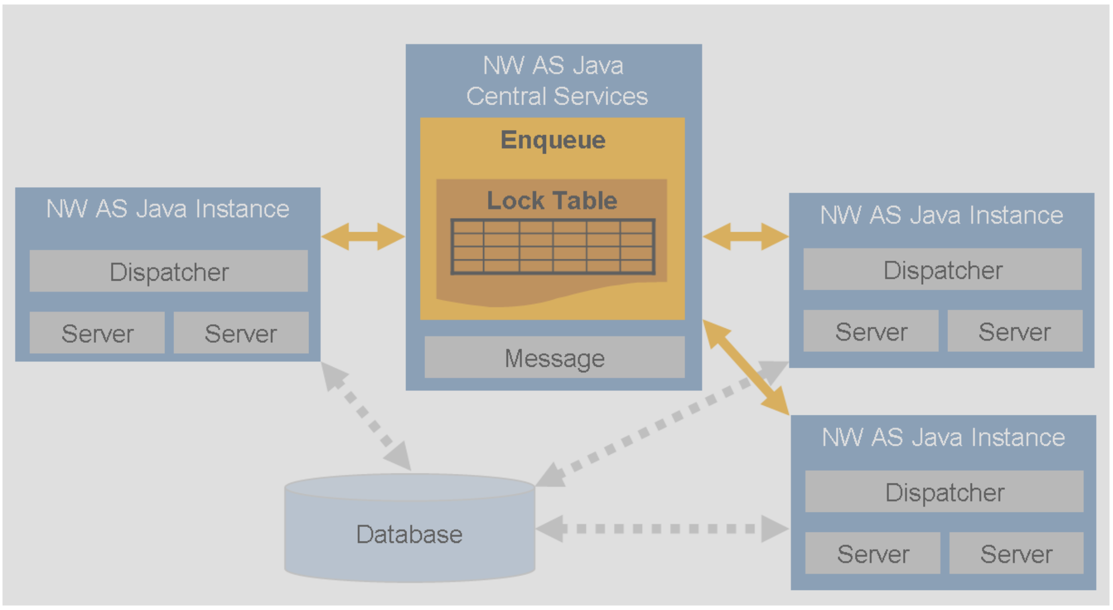
Introducing the Enqueue Server

As of release ´04 of SAP NetWeaver, SAP offers an essential ABAP concept also for Java: the Enqueue Server, also known as the Lock Server.

There is a single Enqueue Server instance running as a central service in any SAP NetWeaver Java cluster (Enqueue Java). Just like in ABAP, Enqueue Java forms a basis for cluster-wide synchronization as it enables server side locking of data, both for internal server data and application data.

The concept is very simple. Prior to modifying any shared data, such as database table entries, Java programs ask Enqueue Java to set an appropriate lock. The Enqueue Java checks whether the lock request collides with an existing lock. If not, it sets the lock, otherwise it rejects it.

The crucial point is that Enqueue Locking is pure server side locking. Enqueue Java does not communicate with any kind of persistent storage, like databases or file system, at all. Instead, it keeps the currently valid locks within an internal lock table in main memory. The Enqueue Server enables locking of arbitrary business objects in a DBMS independent way.



Enqueue Server is a central service in a SAP NetWeaver Java cluster.

A prominent user of Enqueue Locking is the SAP NetWeaver Java itself. Its frameworks typically rely on Enqueue locks, for example during the deployment of applications or while updating configuration data in the system database.

What is Locking for?

In general, the concept of locks is closely related to the concept of transaction isolation. To execute several statements with the “all or none” semantics, you surround them with appropriate begin, commit or rollback commands demarcating a JTA or local database transaction[1]. Transactions usually work on shared data which can be read and changed by several users of the system. But what happens if several transactions try to access the same piece of data simultaneously? Does a user see the changes made concurrently by others? Unfortunately, inconsistencies can arise when some transactions work on uncommitted, “dirty” data:

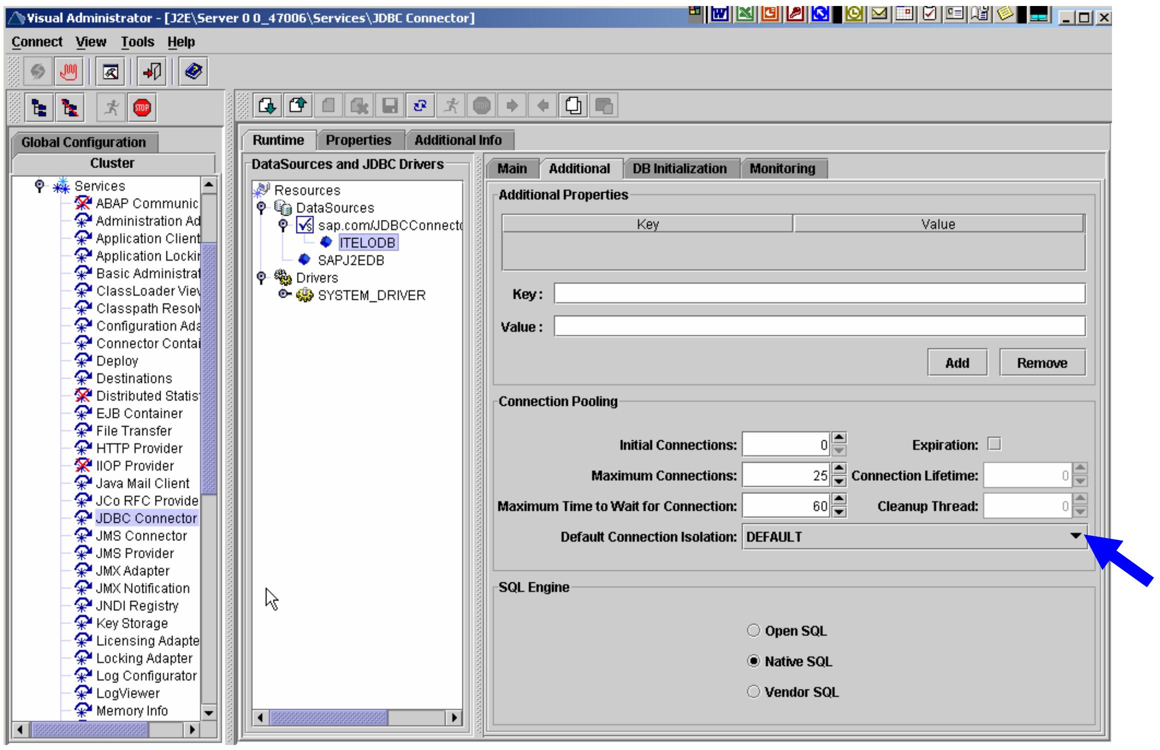
- **Dirty Read**: Transaction 1 changes a row. Transaction 2 reads this row before transaction 1 commits the change. Transaction 1 then rolls back a change. In this case, transaction 2 has read a row that never existed.
- **Non-Repeatable Read**: Transaction 1 reads a row. Transaction 2 then changes or deletes this row and commits these changes. If transaction 1 then reads the row again, but this time it either gets the modified row or a message indicating that the row no longer exists.
- **Phantom**: Transaction 1 performs an SQL statement S with a search condition that returns a result set M of rows. Transaction 2 then creates at least one additional row that meets this search condition, for example, by adding or changing data. If transaction 1 subsequently executes S again, then the new result set is different.

To avoid these phenomena, usually locks will be set on data items a transaction is intended to read or change. This will prevent other transactions to access the locked data until you release the lock.

Database Locking

Traditionally, applications rely on locks that databases set implicitly (database locking). With database locking work, developers decide to which extent database transactions see the changes made by others by setting transaction isolation levels. The JDBC standard adopts several isolation levels, for example `TRANSACTION_READ_UNCOMMITTED` and `TRANSACTION_READ_COMMITTED`. Usually, databases provide additional, proprietary isolation levels, too. You can set the default isolation level for example when defining a JDBC data source representing your database inside the SAP NetWeaver Java.

[1] Within the Java stack the term “transaction” addresses JTA and/or database transactions. There is no similarity to the traditional concept of SAP transactions (ABAP programs).



Key:
Value:

Initial Connections: 0
Expiration: ☐
Maximum Connections: 25
Connection Lifetime: 0
Maximum Time to Wait for Connection: 60
Cleanup Thread: 0
Default Connection Isolation: DEFAULT

SQL Engine: Open SQL [✓ Native SQL] Vendor SQL

Visual Administrator: Setting the default isolation level for a JDBC data source

In this way you define the type of lock the database will implicitly set for database tables or rows the transaction is accessing within SQL statements. For example, the MaxDB database applies shared, exclusive and optimistic locks. All SQL statements that result in a change (`INSERT`/`UPDATE`/`DELETE`) continuously and implicitly request an exclusive lock.

Applications can also acquire such a database lock explicitly using a proprietary interface (SQL) of the database, like a `LOCK` option in the `SELECT` statement in the case of the MaxDB database or a LOCK statement itself.

Enqueue Locking versus Database Locking

Experience shows that if the transaction isolation is stronger, throughput and scalability decrease. In other words the locking strategy is crucial for the response times, throughput and scalability of your system, so you should choose the locking strategy with care. Java applications running on top of SAP NetWeaver can adopt one of the two available locking techniques:

- Database locking (common)
- Enqueue Locking (unique, provided by SAP NetWeaver)

Originally, the Enqueue Locking technique inherently goes with the ABAP LUW concept which cannot be propagated to Java straightforwardly. But SAP's own Java applications continue this concept and apply Enqueue Locking all the time. To see why, consider the following specifics of Enqueue Locking:

- Enqueue locking allows you to run the shared resource (database) at the lowest isolation level to gain the best possible throughput, and, whenever you need more isolation, acquire well-aimed enqueue locks.
- Enqueue locks are portable among DBMS supported by SAP NetWeaver Java. As of today, database systems do not expose uniform semantics for locking by themselves.
- In the case of a lock collision, Enqueue Java immediately throws an exception. Applications can react appropriately and without delay. This is unfortunately not necessarily the case with database locks. Resolving collisions and deadlocks are DBMS specific procedures. Requesting transactions will typically wait some time for locks to be released, for example until the competing transaction terminates (the

waiting end users possibly do not get any notification why the response time is getting so long …).

- Enqueue Locking executes on a higher level, namely on the application level: You explicitly lock application objects, i.e. business objects, regardless of their database representation. Even abstract objects which are not intended to be stored anywhere can still be locked. The same holds for items which do not exist in the database yet. For example, prior to performing monthly billings or annual accounts, you can lock the related month's or year's business data by a single server side lock request. Locking mass data like this is hard to be done efficiently with database locking.

How to Apply Enqueue Locks?

The following basic rules can give you a first guideline for applying enqueue locks in Java:

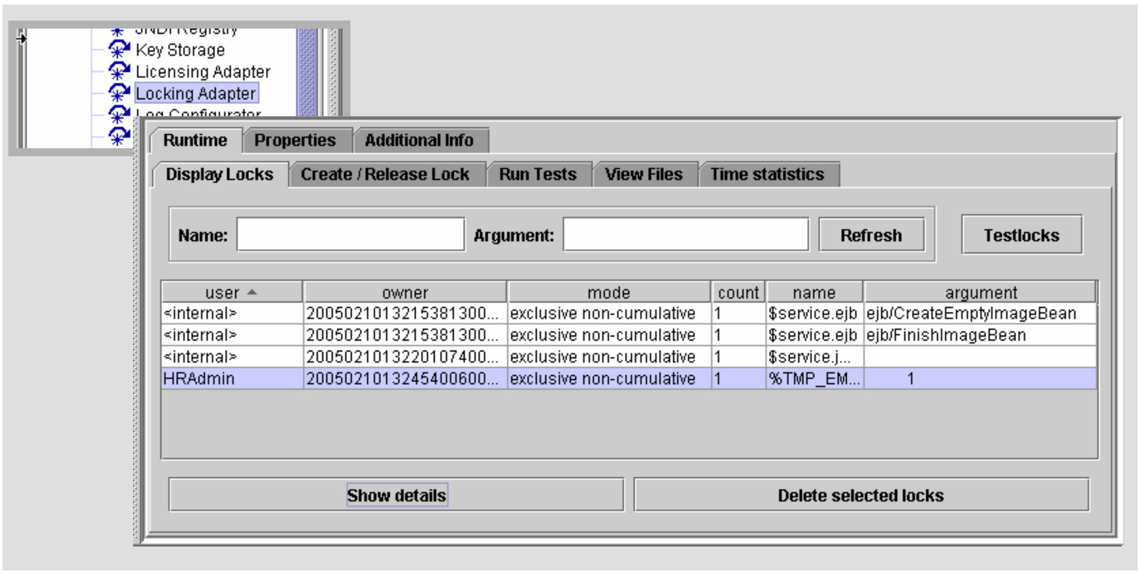
1. Enqueue Locking is applicable independent of whether you keep business data in a system database of SAP NetWeaver Java or in an external database.
2. Make sure all application components rely on Enqueue Locking (obviously, locking works only if all participants agree on the same locking principle).
3. Loosen database locking: Make sure that the JDBC data source operates on the lowest isolation level, `READ_UNCOMMITTED`.
4. Enqueue Locking works fine regardless of the Open SQL level assigned to the JDBC data source (`Open SQL`, `Native SQL` and `Vendor SQL` are all allowed).
5. Acquiring Enqueue locks is only possible within the scope of an active (JTA or database) transaction.
6. It is generally recommended that you stick to one persistence API whenever you change shared data.
   - With object relational mapping technologies: EJB CMP and JDO, you are usually not required to set and release locks explicitly. Per default, both frameworks implicitly lock persistent objects. In addition, both APIs provide their own means allowing developers to influence locking, for example the interface `com.sap.jdo.Locking` for JDO.
   - Otherwise (for example in the case of JDBC and SQLJ based components) you can set and release enqueue locks using simple interfaces provided by the Locking Adapter Service:
     - To lock concrete database entries, use the method `lock()` of the interface `com.sap.engine.services.applocking.TableLocking`:

       ```
       lock(byte lifetime, Connection connection,
                String tableName, Map primaryKeys, char mode)
       ```
     - Otherwise you can invoke the method `lock()` from the more abstract interface `com.sap.engine.services.applocking.LogicalLocking`:

       ```
       lock(byte lifetime, java.lang.String name,
               java.lang.String argument, char mode)
       ```
     - In SAP NetWeaver ´04, the lifetime of an Enqueue lock should be a transaction.
     - Each locking request implies a remote (network) communication between the application and the Enqueue Java instance. Deciding on an appropriate lock granularity helps reduce the network traffic. For example, you can embed wildcard characters when specifying locking arguments and block entire areas of business objects in a single lock request.

You should set locks only when necessary and for as short a time period as possible. There are several options how to release locks:

- In the application: using the `unlock()`method.
- Implicitly: Enqueue locks expire according to their `lifetime` attribute. In SAP NetWeaver 04, the lock lifetime should be a transaction, so that when the transaction ends, Enqueue Java automatically releases locks set during this transaction.
- In case of problems, you can also view and release Enqueue locks manually in the Visual Administrator tool, within the Locking Adapter service.



user	owner	mode	count	name	argument
<internal>	2005021013215381300...	exclusive non-cumulative	1	$service.ejb	ejb/CreateEmptyImageBean
<internal>	2005021013215381300...	exclusive non-cumulative	1	$service.ejb	ejb/FinishImageBean
<internal>	2005021013220107400...	exclusive non-cumulative	1	$service.j...	
HRAdmin	2005021013245400600...	exclusive non-cumulative	1	%TMP_EM...	1

Visual Administrator: Display and delete enqueue locks.

Can ABAP and Java Share Enqueue Locks?

A common Enqueue Locking mechanism for ABAP and Java applications is not implemented in the SAP NetWeaver 04. Therefore, ABAP and Java components cannot be synchronized this way. For example, it does not make sense for Java components running in SAP NetWeaver 04 to acquire Enqueue locks for data managed by ABAP systems.

Summary

SAP NetWeaver 04 offers the ABAP concept of server-side locking of business objects - Enqueue Locking - also for the Java stack. This essential element of the ABAP LUW concept provides an alternative to the database locking in the Java stack, too. Enqueue locking can offer better response times, throughput and scalability than the traditional database locking if applied properly. Moreover, Enqueue Locking enables a portable locking implementation across database systems supported by SAP NetWeaver Java. For more information, refer to the SAP NetWeaver Java documentation available on the SAP Help Portal at http://help.sap.com.