Glassfish 4.1.1 - Code inspection

January 5, 2016

POLITECNICO
MILANO 1863

Daniele Grattarola (Mat. 853101)

Ilyas Inajjar (Mat. 790009)

Andrea Lui (Mat. 850680)

Contents

1 Introduction

1.1 Purpose

This is the code inspection document for a subset of methods in the Glassfish 4.1.1 JEE web-server project. The purpose of this document is to report all code-related issues and compliances of the methods, in relation to some coding practices and standards that will be defined later in the document.

This document is intended for software engineers and programmers involved in the project, as it could possibly identify a necessity for a code cleanup or re-structuring.

1.2 Scope

The Glassfish 4.1.1 project is divided into different modules, each composed of numerous classes and methods.

The analysis that will be carried out in this document will only involve the following 5 methods of the StatefulSessionContainer class (located at Glassfish 4.1.1/appserver/ejb/ejb-container/src/main/java/com/sun/ejb/containers/ of the public Subversion repository of the project):

1. destroyExtendedEMsForContext(SessionContextImpl sc)
2. _getContextForInstance(byte [] instanceKey)
3. _getContext(EjbInvocation inv)
4. doVersionCheck(EjbInvocation inv, Object sessionKey, SessionContextImpl sc)
5. handleConcurrentInvocation(boolean allowSerializedAccess, EjbInvocation inv, SessionContextImpl sc, Object sessionKey)

1.3 Definitions, acronyms, and abbreviations

Throughout this document, the following definitions will be applied without further explanations:

- **JEE**: Java Enterprise Edition

1.4 Code standards and best practice

The analysis that will be performed in this document will be centered on the following code standards and best practice, which will be checked for each method. The conventions are given in the form of a checklist and can be found attached to this document in the “Assignment 3 - Code inspection.pdf” file.

2 Checklist

1. **OK**
2. **OK**
3. **OK**
4. **OK**
5. **2 founds:** _getContextForInstance@1454, _getContext@1605

3 Methods

3.1 destroyExtendedEMsForContext

```
1378    private void destroyExtendedEMsForContext(SessionContextImpl sc) {
1379        for (PhysicalEntityManagerWrapper emWrapper : sc.getExtendedEntityManagers()) {
1380            synchronized (extendedEMReferenceCountMap) {
1381                EntityManager em = emWrapper.getEM();
1382                if (extendedEMReferenceCountMap.containsKey(em)) {
1383                    EEMRefInfo refInfo = extendedEMReferenceCountMap.get(em);
1384                    if (refInfo.refCount > 1) {
1385                        refInfo.refCount--;
1386                        _logger.log(Level.FINE,
1387                                "Decremented RefCount ExtendedEM em: " + em);
1388                    } else {
1389                        _logger.log(Level.FINE, "DESTROYED ExtendedEM em: "
1390                                + em);
1391                        refInfo = extendedEMReferenceCountMap.remove(em);
1392                        eemKey2EEMMap.remove(refInfo.getKey());
1393                        try {
1394                            em.close();
1395                        } catch (Throwable th) {
1396                            _logger.log(Level.FINE,
1397                                    "Exception during em.close()", th);
1398                        }
1399                    }
1400                }
1401            }
1402        }
1403    }
```

3.2 _getContextForInstance

```
1454    private SessionContextImpl _getContextForInstance(byte[] instanceKey) {
1455
1456        Serializable sessionKey = (Serializable) uuidGenerator.byteArrayToKey(instanceKey, 0, -1);
1457
1458        if (_logger.isLoggable(TRACE_LEVEL)) {
1459            _logger.log(TRACE_LEVEL, "[SFSBContainer] Got request for: "
1460                    + sessionKey);
1461        }
1462        while (true) {
1463            SessionContextImpl sc = (SessionContextImpl)
1464                    sessionBeanCache.lookupEJB(sessionKey, this, null);
1465
1466            if (sc == null) {
1467                // EJB2.0 section 7.6
1468                // Note: the NoSuchObjectLocalException gets converted to a
1469                // remote exception by the protocol manager.
1470                throw new NoSuchObjectLocalException(
1471                        "Invalid Session Key ( " + sessionKey + ")");
1472            }
1473
1474            synchronized (sc) {
1475                switch (sc.getState()) {
1476                    case PASSIVATED:      //Next cache.lookup() == different ctx
1477                    case DESTROYED:    //Next cache.lookup() == null
1478                        break;
1479                    default:
1480
1481                        return sc;
1482                }
1483            }
1484        }
1485    }
```

3.3 _getContext

```
1601    /**
1602     * Called from preInvoke which is called from the EJBObject
1603     * for local and remote invocations.
1604     */
1605    public ComponentContext _getContext(EjbInvocation inv) {
1606        EJBLocalRemoteObject ejbo = inv.ejbObject;
1607        SessionContextImpl sc = ejbo.getContext();
1608        Serializable sessionKey = (Serializable) ejbo.getKey();
1609
1610        if (_logger.isLoggable(TRACE_LEVEL)) {
1611            logTraceInfo(inv, sessionKey, "Trying to get context");
1612        }
1613
1614        if (sc == null) {
1615            // This is possible if the EJB was destroyed or passivated.
1616            // Try to activate it again.
1617            sc = (SessionContextImpl) sessionBeanCache.lookupEJB(
1618                    sessionKey, this, ejbo);
1619        }
1620
1621        if ((sc == null) || (sc.getState() == BeanState.DESTROYED)) {
1622            if (_logger.isLoggable(TRACE_LEVEL)) {
1623                logTraceInfo(inv, sessionKey, "Context already destroyed");
1624            }
1625            // EJB2.0 section 7.6
1626            throw new NoSuchObjectLocalException("The EJB does not exist."
1627                    + " session-key: " + sessionKey);
1628        }
1629
1630        MethodLockInfo lockInfo = inv.invocationInfo.methodLockInfo;
1631        boolean allowSerializedAccess =
1632                (lockInfo == null) || (lockInfo.getTimeout() != CONCURRENCY_NOT_ALLOWED);
1633
1634        if( allowSerializedAccess ) {
1635
1636            boolean blockWithTimeout =
1637                    (lockInfo != null) && (lockInfo.getTimeout() != BLOCK_INDEFINITELY);
1638
1639            if( blockWithTimeout ) {
1640                try {
1641                    boolean acquired = sc.getStatefulWriteLock().tryLock(lockInfo.getTimeout(),
1642                            lockInfo.getTimeUnit());
1643                    if( !acquired ) {
1644                        String msg = "Serialized access attempt on method " + inv.beanMethod +
1645                            " for ejb " + ejbDescriptor.getName() + " timed out after " +
1646                             + lockInfo.getTimeout() + " " + lockInfo.getTimeUnit();
1647                        throw new ConcurrentAccessTimeoutException(msg);
1648                    }
1649
1650                } catch(InterruptedException ie)  {
1651                    String msg = "Serialized access attempt on method " + inv.beanMethod +
1652                            " for ejb " + ejbDescriptor.getName() + " was interrupted within " +
1653                             + lockInfo.getTimeout() + " " + lockInfo.getTimeUnit();
1654                    ConcurrentAccessException cae = new ConcurrentAccessTimeoutException(msg);
1655                    cae.initCause(ie);
1656                    throw cae;
1657                }
1658            } else {
1659                sc.getStatefulWriteLock().lock();
1660            }
```

```
1661
1662            // Explicitly set state to track that we're holding the lock for this invocation.
1663            // No matter what we need to ensure that the lock is released.   In some
1664            // cases releaseContext() isn't called so for safety we'll have more than one
1665            // place that can potentially release the lock.  The invocation state will ensure
1666            // we don't accidently unlock too many times.
1667            inv.setHoldingSFSBSerializedLock(true);
1668        }
1669
1670        SessionContextImpl context = null;
1671
1672        try {
1673
1674            synchronized (sc) {
1675
1676                SessionContextImpl newSC = sc;
1677                if (sc.getState() == BeanState.PASSIVATED) {
1678                    // This is possible if the EJB was passivated after
1679                    // the last lookupEJB. Try to activate it again.
1680                    newSC = (SessionContextImpl) sessionBeanCache.lookupEJB(
1681                            sessionKey, this, ejbo);
1682                    if (newSC == null) {
1683                        if (_logger.isLoggable(TRACE_LEVEL)) {
1684                            logTraceInfo(inv, sessionKey, "Context does not exist");
1685                        }
1686                        // EJB2.0 section 7.6
1687                        throw new NoSuchObjectLocalException(
1688                                "The EJB does not exist. key: " + sessionKey);
1689                    }
1690                    // Swap any stateful lock that was set on the original sc
1691                    newSC.setStatefulWriteLock(sc);
1692                }
1693                // acquire the lock again, in case a new sc was returned.
1694                synchronized (newSC) { //newSC could be same as sc
1695                    // Check & set the state of the EJB
1696                    if (newSC.getState() == BeanState.DESTROYED) {
1697                        if (_logger.isLoggable(TRACE_LEVEL)) {
1698                            logTraceInfo(inv, sessionKey, "Got destroyed context");
1699                        }
1700                        throw new NoSuchObjectLocalException
1701                                ("The EJB does not exist. session-key: " + sessionKey);
1702                    } else if (newSC.getState() == BeanState.INVOKING) {
1703                        handleConcurrentInvocation(allowSerializedAccess, inv, newSC, sessionKey);
1704                    }
1705                    if (newSC.getState() == BeanState.READY) {
1706                        decrementMethodReadyStat();
1707                    }
1708                    if (isHAEnabled) {
1709                        doVersionCheck(inv, sessionKey, sc);
1710                    }
1711                    newSC.setState(BeanState.INVOKING);
1712                    context = newSC;
1713                }
1714            }
1715
1716            // touch the context here so timestamp is set & timeout is prevented
1717            context.touch();
```

```
1718
1719            if ((context.existsInStore()) && (removalGracePeriodInSeconds > 0)) {
1720                long now = System.currentTimeMillis();
1721                long threshold = now - (removalGracePeriodInSeconds * 1000L);
1722                if (context.getLastPersistedAt() <= threshold) {
1723                    try {
1724                        backingStore.updateTimestamp(sessionKey, now);
1725                        context.setLastPersistedAt(System.currentTimeMillis());
1726                    } catch (BackingStoreException sfsbEx) {
1727                        _logger.log(Level.WARNING, COULDNT_UPDATE_TIMESTAMP_FOR_EXCEPTION,
1728                                new Object[]{sessionKey, sfsbEx});
1729                        _logger.log(Level.FINE,
1730                                "Couldn't update timestamp for: " + sessionKey, sfsbEx);
1731                    }
1732                }
1733            }
1734
1735            if (_logger.isLoggable(TRACE_LEVEL)) {
1736                logTraceInfo(inv, context, "Got Context!!");
1737            }
1738        } catch(RuntimeException t) {
1739
1740            // releaseContext isn't called if this method throws an exception,
1741            // so make sure to release any sfsb lock
1742            releaseSFSBSerializedLock(inv, sc);
1743
1744            throw t;
1745        }
1746
1747        return context;
1748    }
```

3.4 doVersionCheck

```
1754    private void doVersionCheck(EjbInvocation inv, Object sessionKey,
1755                                SessionContextImpl sc) {
1756        EJBLocalRemoteObject ejbLRO = inv.ejbObject;
1757        long clientVersion = SFSBVersionManager.NO_VERSION;
1758        if ((!inv.isLocal) && (sfsbVersionManager != null)) {
1759            clientVersion = sfsbVersionManager.getRequestClientVersion();
1760            sfsbVersionManager.clearRequestClientVersion();
1761            sfsbVersionManager.clearResponseClientVersion();
1762        }
1763
1764        if (ejbLRO != null) {
1765            if (clientVersion ==
1766                    sfsbVersionManager.NO_VERSION) {
1767                clientVersion = ejbLRO.getSfsbClientVersion();
1768            }
1769
1770            long ctxVersion = sc.getVersion();
1771            if (_logger.isLoggable(TRACE_LEVEL)) {
1772                _logger.log(TRACE_LEVEL, "doVersionCheck(): for: {" + ejbDescriptor.getName()
1773                        + "." + inv.method.getName() + " <=> " + sessionKey + "} clientVersion: "
1774                        + clientVersion + " == " + ctxVersion);
1775            }
1776            if (clientVersion > ctxVersion) {
1777                throw new NoSuchObjectLocalException(
1778                        "Found only a stale version " + " clientVersion: "
1779                                + clientVersion + " contextVersion: "
1780                                + ctxVersion);
1781            }
1782        }
1783    }
```

3.5 handleConcurrentInvocation

```
1785    private void handleConcurrentInvocation(boolean allowSerializedAccess,
1786                                            EjbInvocation inv, SessionContextImpl sc, Object sessionKey) {
1787        if (_logger.isLoggable(TRACE_LEVEL)) {
1788            logTraceInfo(inv, sessionKey, "Another invocation in progress");
1789        }
1790
1791        if( allowSerializedAccess ) {
1792
1793            // Check for loopback call to avoid deadlock.
1794            if( sc.getStatefulWriteLock().getHoldCount() > 1 ) {
1795
1796                throw new IllegalLoopbackException("Illegal Reentrant Access : Attempt to make " +
1797                        "a loopback call on method '" + inv.beanMethod + " for stateful session bean " +
1798                        ejbDescriptor.getName());
1799            }
1800
1801        } else {
1802
1803            String errMsg = "Concurrent Access attempt on method " +
1804                    inv.beanMethod + " of SessionBean " + ejbDescriptor.getName() +
1805                    " is prohibited.  SFSB instance is executing another request. "
1806                + "[session-key: " + sessionKey + "]";
1807            ConcurrentAccessException conEx = new ConcurrentAccessException(errMsg);
1808
1809            if (inv.isBusinessInterface) {
1810                throw conEx;
1811            } else {
1812                // there is an invocation in progress for this instance
1813                // throw an exception (EJB2.0 section 7.5.6).
1814                throw new EJBException(conEx);
1815            }
1816        }
1817    }
```

4 Code inspection

4.1 StatefulSessionContainer

Line	Checklist Number	Comment
150	12	Two spaces before equal
185	7	Last character not uppercase
278	25	public static variable after private static ones
304	7	non costant variable with uppercase name
323	7	constant with lowecase name
325	7	constant with lowecase name

4.2 destroyExtendedEMsForContext

Line	Checklist Number	Comment
1378		Instead using parameter SessionContextImpl use List of PhysicalEntityManagerWrapper
1395	52	Catch block catch Throwable instead of Illegal State Exception

4.3 _getContextForInstance

Line	Checklist Number	Comment
1454	5	Method name pattern should be [a-z][a-z0-9][a-zA-Z0-9_]*
1462	56	While(true) loop
1471	43	Incorrect output format
1476	54	Empty case block without break or return
1476	19	Commented line of code
1477	19	Commented line of code
1480	12	Useless blank line

4.4 _getContext

Line	Checklist Number	Comment
1602	18	Comment not explain adequately the method functions
1605	27	Duplication Code lines: 1621, 1677, 1696. Can be avoided with a proper method
1605	5	Method name pattern should be [a-z][a-z0-9][a-zA-Z0-9_]*
1645	17	Wrong indentation and duplicated '+'
1652	17	Wrong indentation and duplicated '+'
1673	12	Useless blank line
1676	6	Abbreviation in name must contain no more than one capital letter

4.5 doVersionCheck

Line	Checklist Number	Comment
1754	29	Useless paramether (sessionKey) used only for log
1766		Static refence to a non static object
1772	13	Line can be less then 80 characters
1778	43	String concatenation can be avoided

4.6 handleConcurrentInvocation

Line	Checklist Number	Comment
1786	14	Lines can be less then 80 characters
1786	29	Useless parameters: SessionContextImpl, 'sessionKey' only for log
1806	17	Splitted line not aligned

5 Additional comments

The production of this document has been a joint effort of all the authors, with a fair distribution of the mansions which caused each member of the group to work on all the parts of the document. The production has been carried out between 9/12/2015 and 4/1/2016 for a total time expense of:

- **Group work**: 6 hours
- **Individual work**:

Daniele Grattarola (Mat. 853101)	3hours
Ilyas Inajjar (Mat. 790009)	3hours
Andrea Lui (Mat. 850680)	5hours