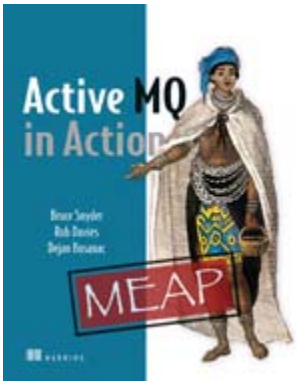
MANNING PUBLICATIONS

How are messages stored by ActiveMQ?

An article from



[ActiveMQ in Action](#)
EARLY ACCESS EDITION

Bruce Snyder, Dejan Bosanac, and Rob Davies
MEAP Release: August 2008
Softbound print: December 2010 (est.) | 375 pages
ISBN: 1933988940

This article is taken from the book ActiveMQ in Action. *The authors show how messages are stored for queues and topics and explain the various message store implementations, their configurations, and when to use each.*

An understanding of storage mechanisms for messages in an ActiveMQ message store aid in configuration and provide an awareness of what takes place in the ActiveMQ broker during the delivery of persistent messages. Messages sent to queues and topics are stored differently, because some storage optimizations can be made with Topics that do not make sense with Queues, as we will explain.

Storage for queues is straightforward. Messages are stored in a first-in, first-out (FIFO) order (see figure 1). One message is dispatched to a single consumer at a time. Only when that message has been consumed and acknowledged can it be deleted from the broker's message store.

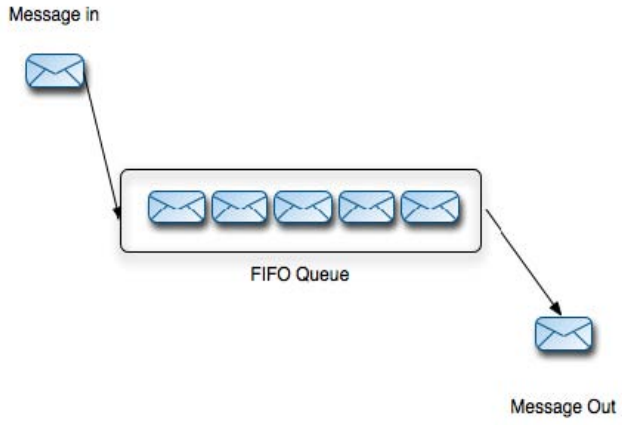


Figure 1 First-in, first-out message storage for queues

For durable subscribers to a topic, each consumer gets a copy of the message. In order to save storage space, only one copy of a message is stored by the broker. A durable subscriber object in the store maintains a pointer to its next stored message and dispatches a copy of it to its consumer, as shown in figure 2. The message store is implemented in this manner because each durable subscriber could be consuming messages at different rates or not running at the same time. Also, because every message can potentially have many consumers, a message cannot be deleted from the store until it has been successfully delivered to every interested durable subscriber.

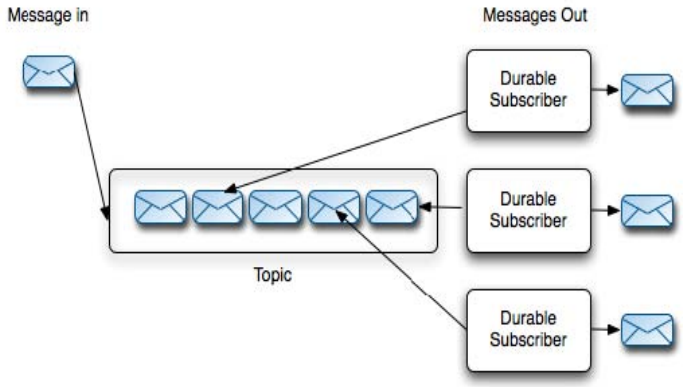


Figure 2 Messages stored for durable subscribers to Topics use message pointers

Every message store implementation for ActiveMQ supports storing messages for both queues and topics, though obviously the implementation differs between storage types; for example, the memory store holds all messages in memory.

The KahaDB Message Store

The recommended message store to use for general purpose message since ActiveMQ version 5.3 is KahaDB. This is a file-based message store that combines a transactional journal for very reliable message storage and recovery with good performance and scalability.

The KahaDB store is a file-based transactional store that has been tuned and designed for a very fast storage of messages. The aim of the KahaDB store is to be easy to use and as fast as possible. Its use of a file-based message database means there is no prerequisite for a third-party database. This message store enables ActiveMQ to be downloaded and running in literally minutes. In addition, the structure of the KahaDB store has been streamlined especially for the requirements of a message broker.

The KahaDB message store uses a transactional log for its indexes and only uses one index file for all its destinations. It has been used in production environments with 10,000 active connections, each connection having a separate queue. The configurability of the KahaDB store means that it can be tuned for most usage scenarios, from high throughput applications (trading platforms) to storing very large amounts of messages (GPS tracking).

To enable the KahaDB store for ActiveMQ, you need to configure the `<persistenceAdapter>` element in the activemq.xml configuration file. Below is a minimal configuration for the KahaDB message store:

```
<broker brokerName="broker" persistent="true" useShutdownHook="false">
...
    <persistenceAdapter>
        <kahaDB directory="activemq-data" journalMaxFileLength="16mb"/>
    </persistenceAdapter>
...
</broker>
```

If you wish to embed an ActiveMQ broker inside an application, the message store can also be configured programmatically. Below is an example of a programmatic configuration for KahaDB:

```
public class EmbeddedBrokerUsingAMQStoreExample {

    BrokerService createEmbeddedBroker() throws Exception {

        BrokerService broker = new BrokerService();                          #1
        File dataFileDir = new File("target/amq-in-action/kahadb");

        KahaDBStore kaha = new KahaDBStore();
        kaha.setDirectory(dataFileDir);                                      #2
        // Using a bigger journal file
        kaha.setJournalMaxFileLength(1024*100);

        // small batch means more frequent and smaller writes
        kaha.setIndexWriteBatchSize(100);
        // do the index write in a separate thread
        kaha.setEnableIndexWriteAsync(true);

        broker.setPersistenceAdapter(kaha);                                  #3
        //create a transport connector
        broker.addConnector("tcp://localhost:61616");                        #4
        //start the broker broker.start();                                   #5

    return broker; } }
```

#1 Initializing the ActiveMQ broker
#2 Creating an instance of the KahaDB message store, telling it to use the directory target/amq-in-action/kahadb to store its data
#3 Instructing the broker to use the KahaDB store
#4 Creating a transport connector to expose the broker to clients
#5 Starting the broker

Although this example seems small, it is enough to create an ActiveMQ broker using the KahaDB message store and listen for ActiveMQ clients connecting over TCP.

In order to better understand its use and configuration, it is important to examine the internals of the KahaDB message store, which we will do next.

The KahaDB Message Store Internals

The AMQ store is the fastest of all the provided message store implementations. Its speed results from a combination of a fast transactional journal consisting of data log files, the highly optimized indexing of message IDs, and in-memory message caching. Figure 3 provides a high-level diagram of the AMQ message store.

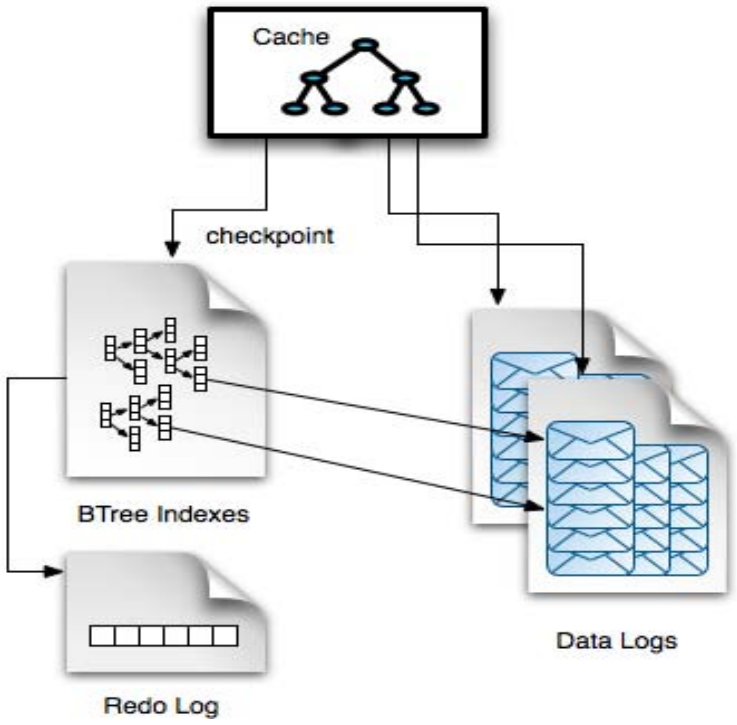


Figure 3 In KahaDB, messages are stored in indexed log files and cached for performance

The diagram above provides a view of the three distinct parts of the KahaDB message store including:

- **The Data Logs** act as a message journal that consists of a rolling log of messages and commands (such as transactional boundaries and message deletions) stored in data files of a certain length. When the maximum length of the currently used data file has been reached, a new data file is created. All the messages in a data file are reference counted so that, once every message in that data file is no longer required, the data file can be removed or archived. In the data logs, messages are only appended to the end of the current data file, so storage is very fast.
- **The Cache** holds messages for fast retrieval in the memory after they have been written to the data logs. Because messages are always persisted in the data logs first, ActiveMQ does not suffer from message loss on a sudden machine or broker failure. The cache will periodically update the reference store with its current message IDs and location in the data logs. This process is known as performing a checkpoint. Once the reference store has been updated, messages can be safely removed from the cache. The period of time between the cache updates to the reference store is configurable and can be set by the `checkpointInterval` property. A checkpoint will also occur if the ActiveMQ message broker is deemed to be reaching the ActiveMQ system usage memory limit, which can be set in the ActiveMQ broker configuration. When the amount of memory used for messages passes 70% of this memory limit, a checkpoint will also occur.
- **The BTree indexes** hold references to the messages in the data logs that are indexed by their message IDs. It is actually the indexes that maintain the FIFO data structure for queues and the durable subscriber pointers to their topic messages. The redo log is used only if the ActiveMQ broker has not shut down cleanly and to insure the BTree index's integrity is maintained.

KahaDB uses different files on disk for its data logs and indexes so we will show what a typical KahaDB directory structure should look like.

The KahaDB Message Store Directory Structure

When you start an ActiveMQ broker configured to use a KahaDB store, a directory will automatically be created in which the persistent messages are stored. This directory structure is shown in figure 4.

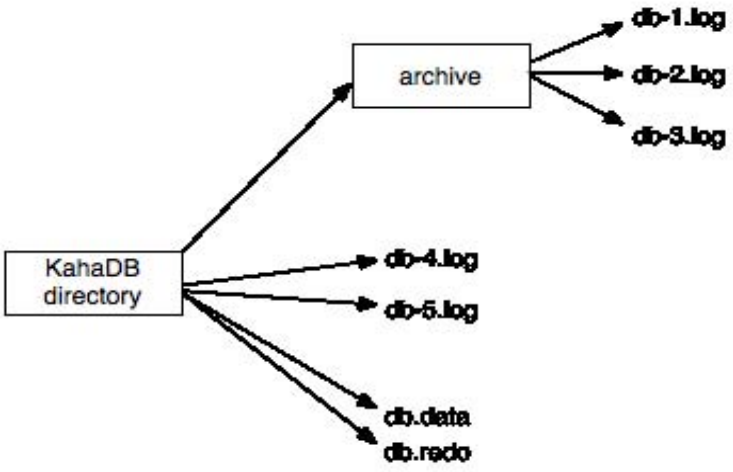


Figure 4 The KahaDB message store directory structure

Inside the KahaDB directory, following directory and file structures can be found:

- **db log files—**KahaDB stores messages into data log files named `db-<Number>.log` of a predefined size. When a data log is full, a new one will be created, the log number being incremented. When there are no more references to any of the messages in the data log file, it will be deleted or archived.
- **The archive directory—**Exists only if archiving is enabled. The archive is used to store data logs that are no longer needed by KahaDB, making it possible to replay messages from the archived data logs at a later point. If archiving is not enabled (the default), data logs that are no longer in use are deleted from the file system.
- **db.data—**This file contains the persistent BTree indexes to the messages held in the message data logs.
- **db.redo—**This is the redo file used for recovering the BTree indexes if the KahaDB message store starts after a hard stop.

Now that the basics of the KahaDB store have been covered, the next step is to review its configuration.

Configuring the KahaDB Message Store

The KahaDB message store can be configured in the ActiveMQ broker configuration file. Its configuration options control the different tuning parameters, as described in table 1.

Table 1 Configuration options available for the KahaDB message store

Property name	Default value	Description
directory	activemq-data	The directory path used by KahaDB
setIndexWriteBatchSize	1000	The number of indexes to write in a batch to disk
enableIndexWriteAsync	false	If set, will asynchronously write indexes

journalMaxFileLength	32mb	A hint to set the maximum size of each of the message data logs
maxCheckpointMessageAdd	Size4kb	The maximum number of messages to keep in a transaction before automatically committing
cleanupInterval	30000	Time (ms) before checking for a discarding/moving message data logs that are no longer used
checkpointInterval	5000	Time (ms) before checkpointing the journal
indexCacheSize	100	The maximum number of index pages to hold cached in memory
ignoreMissingJournalfiles	false	If enabled, will ignore a missing message log file
checkForCorruptJournalFile	sfalse	If enabled, on startup will validate that the message data logs have not been corrupted.
checksumJournalFiles	false	If enabled, will provide a checksum for each message data log.
archiveDataLogs	false	If enabled, will move a message data log to the archive directory instead of deleting it.
directoryArchive	null	Define the directory to move data logs to when they all the messages they contain have been consumed.

ActiveMQ also offers support for other message storage implementations, including another file based message store—the AMQ Message Store, a message store designed for relational databases—the JDBC Message Store, and a memory-only message store.

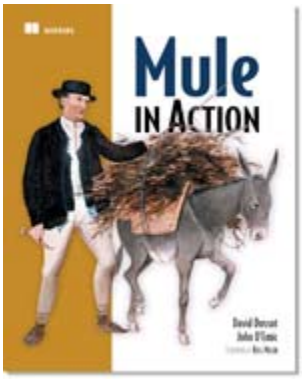
Summary

We discussed how messages are stored differently for queues and topics. Then, we explained the various message store implementations, their configurations, and when to use each.

Here are some other Manning titles you might be interested in:

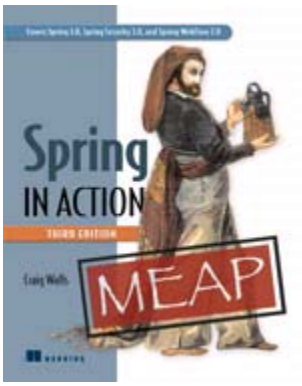
Open Source SOA
IN PRINT

Jeff Davis
May 2009 | 448 pages
ISBN: 1933988541



Mule in Action
IN PRINT

David Dossot and John D'Emic
Foreword by Ross Mason, Creator of Mule
July 2009 | 432 pages
ISBN: 1933988967



Spring in Action, Third Edition
EARLY ACCESS EDITION

Craig Walls
MEAP Release: June 2009
Softbound print: January 2011 (est.) | 700 pages
ISBN: 9781935182351