### 一、JDBC持久化方式

<beans>

<broker brokerName="test-broker" persistent="true" xmlns="http://activemq.apache.org/schema/core">

<persistenceAdapter>

<jdbcPersistenceAdapter dataSource="#mysql-ds" createTablesOnStartup="false"/>

</persistenceAdapter>

</broker>

<bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>

<property name="username" value="activemq"/>

<property name="password" value="activemq"/>

<property name="maxActive" value="200"/>

<property name="poolPreparedStatements" value="true"/>

</bean>

</beans>

数据库会创建3个表：activemq\_msgs，activemq\_acks和activemq\_lock。

activemq\_msgs用于存储消息，Queue和Topic都存储在这个表中：

ID：自增的数据库主键

CONTAINER：消息的Destination

MSGID\_PROD：消息发送者客户端的主键

MSG\_SEQ：是发送消息的顺序，MSGID\_PROD+MSG\_SEQ可以组成JMS的MessageID

EXPIRATION：消息的过期时间，存储的是从1970-01-01到现在的毫秒数

MSG：消息本体的Java序列化对象的二进制数据

PRIORITY：优先级，从0-9，数值越大优先级越高

activemq\_acks用于存储订阅关系。如果是持久化Topic，订阅者和服务器的订阅关系在这个表保存：

主要的数据库字段如下：

CONTAINER：消息的Destination

SUB\_DEST：如果是使用Static集群，这个字段会有集群其他系统的信息

CLIENT\_ID：每个订阅者都必须有一个唯一的客户端ID用以区分

SUB\_NAME：订阅者名称

SELECTOR：选择器，可以选择只消费满足条件的消息。条件可以用自定义属性实现，可支持多属性AND和OR操作

LAST\_ACKED\_ID：记录消费过的消息的ID。

表activemq\_lock在集群环境中才有用，只有一个Broker可以获得消息，称为Master Broker，

其他的只能作为备份等待Master Broker不可用，才可能成为下一个Master Broker。

这个表用于记录哪个Broker是当前的Master Broker。

### 二、KahaDB方式

配置方式：

<persistenceAdapter>

<kahaDB directory="${activemq.data}/activemq-data" journalMaxFileLength="16mb"/>

</persistenceAdapter>

directory : 指定持久化消息的存储目录

journalMaxFileLength : 指定保存消息的日志文件大小，具体根据你的实际应用配置

默认的持久化插件，也是我们项目现在使用的持久化方式。

KahaDb恢复时间远远小于其前身AMQ并且使用更少的数据文件，所以可以完全代替AMQ。

kahaDB的持久化机制同样是基于日志文件，索引和缓存。

（1）KahaDB主要特性

1、日志形式存储消息；

2、消息索引以B-Tree结构存储，可以快速更新；

3、完全支持JMS事务；

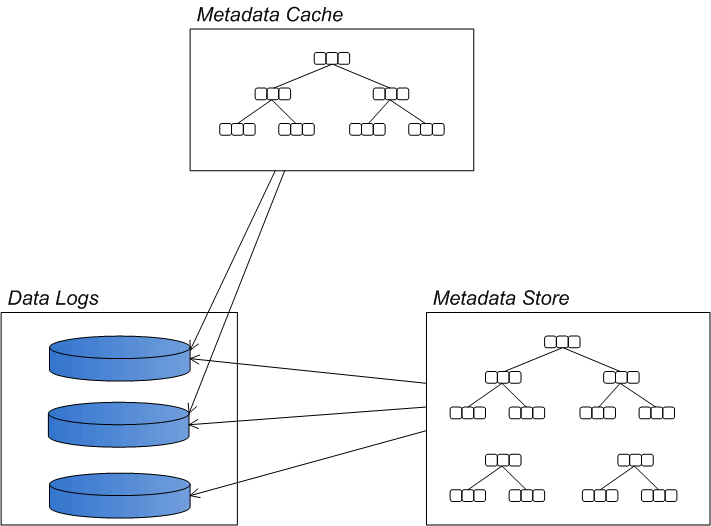
4、支持多种恢复机制；

（2）KahaDB的结构

消息存储在基于文件的数据日志中。如果消息发送成功，变标记为可删除的。系统会周期性的清除或者归档日志文件。

消息文件的位置索引存储在内存中，这样能快速定位到。定期将内存中的消息索引保存到metadata store中，避免大量消息未发送时，消息索引占用过多内存空间。

从ActiveMQ 5.6版本之后，又推出了LevelDB的持久化引擎。  
目前默认的持久化方式仍然是KahaDB，不过LevelDB持久化性能高于KahaDB，可能是以后的趋势。  
在ActiveMQ 5.9版本提供了基于LevelDB和Zookeeper的数据复制方式，用于Master-slave方式的首选数据复制方案。



**Data logs：**  
Data logs用于存储消息日志，消息的全部内容都在Data logs中。  
同AMQ一样，一个Data logs文件大小超过规定的最大值，会新建一个文件。同样是文件尾部追加，写入性能很快。  
每个消息在Data logs中有计数引用，所以当一个文件里所有的消息都不需要了，系统会自动删除文件或放入归档文件夹。

**Metadata cache ：**  
缓存用于存放在线消费者的消息。如果消费者已经快速的消费完成，那么这些消息就不需要再写入磁盘了。  
Btree索引会根据MessageID创建索引，用于快速的查找消息。这个索引同样维护持久化订阅者与Destination的关系，以及每个消费者消费消息的指针。

**Metadata store**   
在db.data文件中保存消息日志中消息的元数据，也是以B-Tree结构存储的，定时从Metadata cache更新数据。Metadata store中也会备份一些在消息日志中存在的信息，这样可以让Broker实例快速启动。  
即便metadata store文件被破坏或者误删除了。broker可以读取Data logs恢复过来，只是速度会相对较慢些。

### 三、AMQ方式

<persistenceAdapter>

<amqPersistenceAdapter directory="${activemq.data}/activemq-data" maxFileLength="32mb"/>

</persistenceAdapter>

性能高于JDBC，写入消息时，会将消息写入日志文件，由于是顺序追加写，性能很高。为了提升性能，创建消息主键索引，并且提供缓存机制，进一步提升性能。每个日志文件的大小都是有限制的（默认32m，可自行配置）。

当超过这个大小，系统会重新建立一个文件。当所有的消息都消费完成，系统会删除这个文件或者归档（取决于配置）。

主要的缺点是AMQ Message会为每一个Destination创建一个索引，如果使用了大量的Queue，索引文件的大小会占用很多磁盘空间。

而且由于索引巨大，一旦Broker崩溃，重建索引的速度会非常慢。

### 四、LevelDB方式

从ActiveMQ 5.6版本之后，又推出了LevelDB的持久化引擎。

目前默认的持久化方式仍然是KahaDB，不过LevelDB持久化性能高于KahaDB，可能是以后的趋势。

在ActiveMQ 5.9版本提供了基于LevelDB和Zookeeper的数据复制方式，用于Master-slave方式的首选数据复制方案。