



一、在linux系统上搭建activemq单机版

1、下载activemq解压版包

   网址为：<http://activemq.apache.org/download.html>

   下载之后分为64位操作系统与32位操作系统，根据安装环境进行判断 点击最新版  或者全部版本 根据自己的需求进行下载。



2、在linux系统上安装单机版

直接将解压包复制到linux系统中(用鼠标拉进去就可)  解压文件   命令如下：

http://img.blog.csdn.net/20170428154643681?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbHBwX2Rk/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

打开解压后的文件夹   在bin目录下通过命令 ./activemq start  开启activemq的服务

通过 ./activemq status 命令查看activemq的状态

或者通过 ps -ef | grep java 查看当前运行的java线程

通过访问   [www.ip:8161](http://www.ip:8161/)  访问网页版的activemq客户端

二、结合spring+maven搭建activemq客户端

1、编写pom.xml文件

在配置了中央仓库并联网的情况下

配置中央仓库的位置是修改maven包下的setting.xml配置文件

需要修改的中央仓库的地址，可以在百度上进行查询中央仓库的地址

1. <dependencies>
2. <!-- spring-jms -->
3. <dependency>
4. <groupId>org.springframework</groupId>
5. <artifactId>spring-jms</artifactId>
6. <version>4.3.8.RELEASE</version>
7. </dependency>
8. <!-- javax -jms -->
9. <dependency>
10. <groupId>javax.jms</groupId>
11. <artifactId>jms</artifactId>
12. <version>1.1</version>
13. </dependency>
14. <!-- active all -->
15. <dependency>
16. <groupId>org.apache.activemq</groupId>
17. <artifactId>activemq-all</artifactId>
18. <version>5.9.0</version>
19. <scope>compile</scope>
20. </dependency>
21. <dependency>
22. <groupId>org.apache.activemq</groupId>
23. <artifactId>activemq-client</artifactId>
24. <version>5.9.0</version>
25. </dependency>
26. <dependency>
27. <groupId>org.apache.activemq</groupId>
28. <artifactId>activemq-core</artifactId>
29. <version>5.7.0</version>
30. </dependency>
31. <dependency>
32. <groupId>org.apache.activemq</groupId>
33. <artifactId>activemq-pool</artifactId>
34. <version>5.9.0</version>
35. </dependency>
36. <dependency>
37. <groupId>org.apache.activemq.protobuf</groupId>
38. <artifactId>activemq-protobuf</artifactId>
39. <version>1.1</version>
40. </dependency>
42. <!-- spring 上下文 -->
43. <dependency>
44. <groupId>org.springframework</groupId>
45. <artifactId>spring-context</artifactId>
46. <version>4.3.8.RELEASE</version>
47. </dependency>
48. <!-- spring 测试包 项目上线后可以去掉 -->
49. <dependency>
50. <groupId>org.springframework</groupId>
51. <artifactId>spring-test</artifactId>
52. <version>4.0.4.RELEASE</version>
53. </dependency>
54. <!-- spring web包 -->
55. <dependency>
56. <groupId>org.springframework</groupId>
57. <artifactId>spring-webmvc</artifactId>
58. <version>4.3.8.RELEASE</version>
59. </dependency>
60. <!-- spring 核心包 -->
61. <dependency>
62. <groupId>org.springframework</groupId>
63. <artifactId>spring-core</artifactId>
64. <version>4.3.8.RELEASE</version>
65. </dependency>
66. <!-- spring bean包 -->
67. <dependency>
68. <groupId>org.springframework</groupId>
69. <artifactId>spring-beans</artifactId>
70. <version>4.3.8.RELEASE</version>
71. </dependency>
72. <!-- spring aop包 -->
73. <dependency>
74. <groupId>org.springframework</groupId>
75. <artifactId>spring-aop</artifactId>
76. <version>4.3.8.RELEASE</version>
77. </dependency>
78. </dependencies>

导入的maven约束如上所示

如果需要导入不一样版本的jar包约束  方式如下：

访问[http://mvnrepository.com/](http://mvnrepository.com/%E7%BD%91%E7%AB%99%E5%9C%A8%E9%87%8C%E9%9D%A2%E6%9F%A5%E6%89%BE%E9%9C%80%E8%A6%81%E7%9A%84jar%E5%8C%85maven%E7%BA%A6%E6%9D%9F)

或者直接在百度里面搜索   包名+maven   直接访问相应jar包的界面

2、编写spring配置文件

配置文件activemqconfig.properties

1. jms.broker.url=failover:(tcp://10.0.1.227:61616,tcp://10.0.1.227:61617,tcp://10.0.1.227:61618)
2. jms.queue.name=str.queue
3. jms.userName=admin
4. jms.password=admin
5. jms.connections.num=10

通过spring的jmsTemplate实现的消息发送端的配置文件编写  activemqProvider.xml

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:amq="http://activemq.apache.org/schema/core"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
6. http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core-5.8.0.xsd">
8. <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
9. <!-- 表示加载的顺序 -->
10. <property name="order" value="1" />
11. <!-- 是否忽略不可以解析的Placeholder 当加载多个配置文件时就需要配置true-->
12. <property name="ignoreUnresolvablePlaceholders" value="true" />
13. <property name="location">
14. <value>classpath:activemqconfig.properties</value>
15. </property>
16. </bean>
17. <!-- Activemq connection factory -->
18. <bean id="amqConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
19. <constructor-arg index="0" value="${jms.useName}"/>
20. <constructor-arg index="1" value="${jms.password}"/>
21. <constructor-arg index="2" value="${jms.broker.url}"/>
23. <!-- 表示异步发送消息 -->
24. <property name="useAsyncSend" value="true"/>
25. </bean>
26. <!-- 配置连接池 -->
27. <bean id="pooledConnectionFactoryBean" class="org.apache.activemq.pool.PooledConnectionFactoryBean">
28. <property name="connectionFactory" ref="amqConnectionFactory"/>
29. <property name="maxConnections" value="${jms.connections.num}"/>
30. </bean>
32. <!-- ConnectionFactory Definition -->
33. <bean id="connectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
34. <property name="targetConnectionFactory" ref="pooledConnectionFactoryBean"/>
35. </bean>
36. <!-- JmsTemplate Definition -->
37. <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
38. <property name="connectionFactory" ref="connectionFactory"/>
39. <property name="defaultDestinationName" value="${jms.queue.name}"/>
40. </bean>
42. </beans>

配置监听类的方式的消息消费方配置文件的编写：activemqCustomer.xml

1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:amq="http://activemq.apache.org/schema/core"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
6. http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core-5.8.0.xsd">
8. <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
9. <property name="order" value="1" />
10. <property name="ignoreUnresolvablePlaceholders" value="true" />
11. <property name="location">
12. <value>classpath:activemqconfig.properties</value>
13. </property>
14. </bean>
15. <!-- Activemq connection factory -->
16. <bean id="amqConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
17. <constructor-arg index="0" value="${jms.useName}"/>
18. <constructor-arg index="1" value="${jms.password}"/>
19. <constructor-arg index="2" value="${jms.broker.url}"/>
21. <!-- 表示异步发送消息 -->
22. <property name="useAsyncSend" value="true"/>
23. </bean>
24. <!-- 配置连接池 -->
25. <bean id="pooledConnectionFactoryBean" class="org.apache.activemq.pool.PooledConnectionFactoryBean">
26. <property name="connectionFactory" ref="amqConnectionFactory"/>
27. <property name="maxConnections" value="${jms.connections.num}"/>
28. </bean>

31. <!-- ConnectionFactory Definition -->
32. <bean id="connectionFactory" class="org.springframework.jms.connection.CachingConnectionFactory">
33. <property name="targetConnectionFactory" ref="pooledConnectionFactoryBean"/>
34. </bean>
36. <!-- 自定义消息监听类 -->
37. <bean id="messageReceiver" class="com.xxxx.receiver.ActiveMqReceiverListener"/>

40. <!-- 配置消息监听器 -->
41. <bean class="org.springframework.jms.listener.SimpleMessageListenerContainer">
42. <property name="connectionFactory" ref="connectionFactory"/>
43. <property name="destinationName" value="${jms.queue.name}"/>
44. <property name="messageListener" ref="messageReceiver"/>
45. </bean>
47. <!-- 自定义消息监听类 -->
48. <bean id="messageReceiverTwo" class="com.xxxx.receiver.ActiveMqReceiverListenerTwo"/>

51. <!-- 配置消息监听器 -->
52. <bean class="org.springframework.jms.listener.SimpleMessageListenerContainer">
53. <property name="connectionFactory" ref="connectionFactory"/>
54. <property name="destinationName" value="${jms.queue.nameTwo}"/>
55. <property name="messageListener" ref="messageReceiverTwo"/>
56. </bean>
58. </beans>

在这里配置了两个消息消费的监听类用于监听不同消息队列中传递的消息类型

知识点介绍：

activemq支持的消息类型为5种分别为：

session.createTextMessage();   String数据类型

session.createMapMessage();    Map键值对数据类型

session.createStreamMessage(); StreamIO流数据  可以传递文件

session.createObjectMessage(); 对象数据类型，要求对象类需要实现序列化接口

session.createBytesMessage();  传递字节数据数据类型

注意在传递流数据类型时 在向消息队列中存储数据时，发送方存储的时候是可以一直存储但是在消费方获取的时候只能取到最后一次存储的数据

所以需要在向streamMessage对象中存储一次数据就发送一次数据

1. MessageProducer producer = session.createrProducer("目标队列")
2. File file = new File("c:\\a.txt");
3. FileInputStream in = new FileInputStream(file);
4. byte[] bytes = new byte[1024];
5. int i = -1;
6. while((i=in.read(bytes))!=-1) {
7. message = session.createStreamMessage();
8. //这个方法实现在消息队列中存储键值对信息 文件名作为参数在消费方获取
9. message.setStringProperty("filename",filename);
10. //将messge中的信息进行清除
11. message.clearBody();
12. message.writeBytes(bytes);
13. producer.send(message);
14. }

消息流对象可以实现传递任意格式的文件，所以我们在传递大文件时，可以先实现将文件进行压缩然后再进行发送，在消息消费方再进行解压实现文件的快速传递。

**3、消息发送方与消息消费方**

消息消费方属性注入jmsTemplate对象调用send方法 编写匿名内部类MesageCreator类，重写里面的createMessage方法，获取session会话对象，实现消息的发送。

消息消费方，在配置文件里面配置了实体类的位置之后，在编写实体类时实现MessageListener类重写onMessage方法对接收的方法进行判断。

例子代码：

消息发送方：

1. import javax.jms.JMSException;
2. import javax.jms.MapMessage;
3. import javax.jms.Message;
4. import javax.jms.Session;
5. import org.springframework.jms.core.JmsTemplate;
6. import org.springframework.jms.core.MessageCreator;
7. public class Sender {
8. private JmsTemplate jmsTemplate;
9. //getter and setter
10. public JmsTemplate getJmsTemplate() {
11. return jmsTemplate;
12. }
13. public void setJmsTemplate(JmsTemplate jmsTemplate) {
14. this.jmsTemplate = jmsTemplate;
15. }
17. public void sendInfo() {
18. jmsTemplate.send(new MessageCreator() {
19. public Message createMessage(Session session) throws JMSException {
20. MapMessage message = session.createMapMessage();
21. message.setString("lastName", "ppp");
22. return message;
23. }
24. });
25. }
26. }

消息消费方：

1. public class QueueMessageListener implements MessageListener {
2. //当收到消息时，自动调用该方法。
3. public void onMessage(Message message) {
4. TextMessage tm = (TextMessage) message;
5. try {
6. System.out.println("ConsumerMessageListener收到了文本消息：\t"
7. + tm.getText());
8. } catch (JMSException e) {
9. e.printStackTrace();
10. }
11. }
12. }

**三、搭建activemq集群**

在linux上搭建activemq集群的方式   在这里是基于zookeeper实现的集群搭建

首先理解一下基于zookeeper的activemq集群的原理：

首先zookeeper搭建一个至少三个节点的集群，然后在集群基础上，我们每个mq节点会去主动连接zookeeper集群获取当前的的主机，这个主机的对外开放的地址与端口号为每个mq的bind值。zookeeper推选主机的机制是选举制。我们在客户端连接mq时，是通过tcp协议failover的方式去连接，连接配置上所有节点的mq对外开放的bind值，其实在zookeeper的选举下，此时的多个节点下的mq，只有一个被选举为主机master的mq的bind节点是对外开放的，其他的节点是默认拒绝连接的。这时，我们客户端连接时，就会挨个的去连接，直到遇见能够连接上的节点，如果所有的节点都不能连接，这时failover方式的连接会堵塞住。zookeeper内部的心跳机制会定时的查看各个mq节点的状态，一旦主机宕机了，就重新选举新的节点作为主机。这样，不仅做到了高可用，还可以利用zookeeper内部进行主备机之间的数据共享。数据分为异步与实时同步两种方式，一般情况下是主机将实时的共享到一台备机然后由这台备机异步的将数据共享到其他备机上。

**1、搭建zookeeper集群**

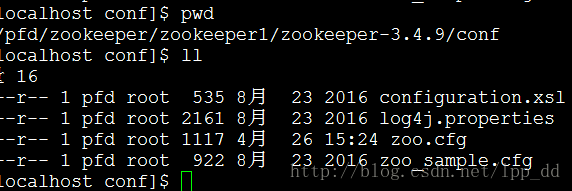
1、下载zookeeper解压包

网站<http://www.apache.org/dyn/closer.cgi/zookeeper/>       点击download按钮  选择[**http://mirrors.hust.edu.cn/apache/zookeeper/**](http://mirrors.hust.edu.cn/apache/zookeeper/)   这个下载地址(国内的下载地址)

2、解压zookeeper安装包

http://img.blog.csdn.net/20170428154742044?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbHBwX2Rk/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

3、修改配置文件

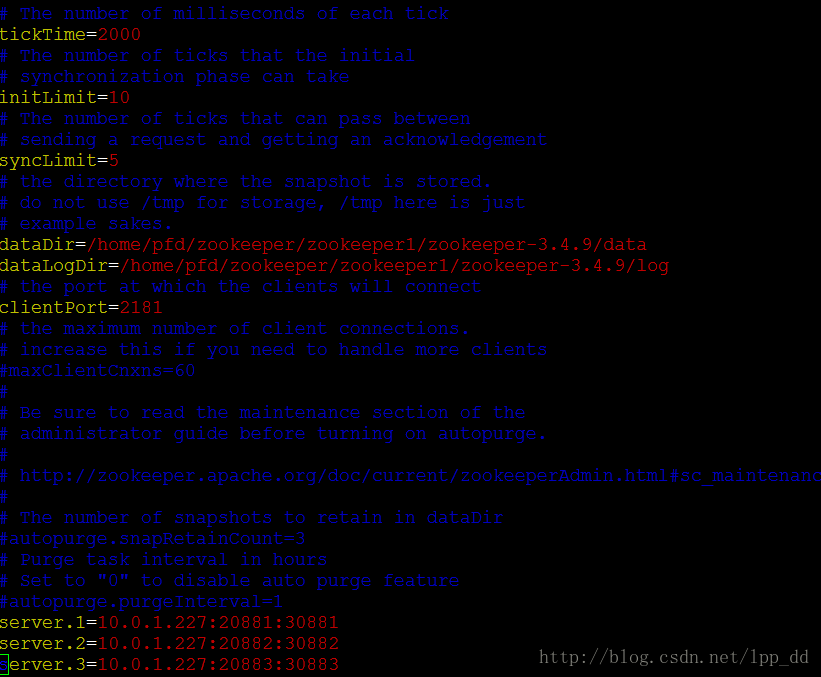


在安装包的conf路径下的zoo\_sample.cfg文件

复制一份到当前目录 改名为zoo.cfg

http://img.blog.csdn.net/20170428154837214?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbHBwX2Rk/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

修改zoo.cfg文件中的内容：



如上所示在这里设置三个(奇数个)zookeeper实现简单的集群搭建。

zookeeper默认是选举制保证奇数个数的成员实现选举出leader机器作为当前的主机，其余的作为follow为备机。

添加dataDir作为数据文件的路径

添加dataLogDir作为日志文件的路径

在文件底部添加service.n=ip:port:port 第一个端口是成员之间相互交流的端口号，第二个端口是选举leader时的端口号

参数详情：

tickTime这个时间是作为zookeeper服务器之间或客户端与服务器之间维持心跳的时间间隔,也就是说每个tickTime时间就会发送一个心跳。

initLimit这个配置项是用来配置zookeeper接受客户端（这里所说的客户端不是用户连接zookeeper服务器的客户端,而是zookeeper服务器集群中连接到leader的follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。

当已经超过10个心跳的时间（也就是tickTime）长度后 zookeeper 服务器还没有收到客户端的返回信息,那么表明这个客户端连接失败。总的时间长度就是 10\*2000=20秒。

syncLimit这个配置项标识leader与follower之间发送消息,请求和应答时间长度,最长不能超过多少个tickTime的时间长度,总的时间长度就是5\*2000=10秒。

dataDir顾名思义就是zookeeper保存数据的目录,默认情况下zookeeper将写数据的日志文件也保存在这个目录里；

clientPort这个端口就是客户端连接Zookeeper服务器的端口,Zookeeper会监听这个端口接受客户端的访问请求；

----来自 linux就该这么学 用户  http://www.cnblogs.com/linuxprobe/p/5851699.html

注意在这里的clientPort的参数 如果在同一台虚拟机上配置三台zookeeper，需要修改端口号

4、创建data文件夹 log文件夹 myid唯一标志文件

在zookeeper主目录下创建data与log文件夹   在data文件夹下创建myid文件

touch myid   在每个文件下插入内容   echo 1 > myid   这个数字表示当前的zookeeper的编号

5、复制当前修改好的zookeeper到两个新的文件夹下

修改myid的唯一标识，echo 2 > myid    如果在同一台虚拟机上搭建的集群  需要修改clientPort参数

6、分别运行三台zookeeper

./zkServer.sh start

运行前两台zookeeper时会报错，但是不要担心，运行第三台后会发生这样的显示：

http://img.blog.csdn.net/20170428154905500?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvbHBwX2Rk/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast

显示的状态为follower或者leader

表示搭建zookeeper集群成功！！！

通过命令

sh zkServer.sh status 查看运行状态

2、搭建activemq集群

1、下载activemq解压包  解压（单机版已经讲解）

2、修改conf文件夹下的activemq.xml文件

1. <persistenceAdapter>
2. <!--<kahaDB directory="${activemq.data}/kahadb"/>-->
3. <replicatedLevelDB
4. directory="${activemq.data}/leveldb"
5. replicas="3"
6. bind="tcp://10.0.1.227:61616"
7. hostname="localhost"
8. zkAddress="10.0.1.227:2181,10.0.1.227:2182,10.0.1.227:2183"
9. zkPath="/activemq/leveldb-stores"
10. sync="local\_disk"
11. />
12. </persistenceAdapter>

在activemq.xml文件中找到persistenceAdapter节点

步骤如下：

1、vim activemq.xml

2、shift+/

3、输入persistenceAdapter回车

（小写n向下找

大写N向上找）

将原内容修改成如上所示的内容

参数内容

directory：activemq数据的文件夹位置

不要改

replicas：表示成员的个数

bind：表示客户端连接的地址

brokerURL参数的内容

hostname：表示主机的名字

zkAddress：表示连接的zookeeper的ip

zkPath：表示zookeeper位置

不要改

sync 是否异步

不要改

3、复制activemq三份

分别修改activemq.xml文件中的bind值

3、客户端的连接配置

<property name=“brokerURL” value=“failover:(tcp://192.168.0.101:61616,tcp://192.168.0.101:61617, tcp://192.168.0.101:61618)" /> 

在配置ActiveMQConnectionFactory对象的时候需要注入brokerURL的值，当有多台activemq时，连接参数的写法如上。

4、多线程访问中发送方与消费方的配置

1、在发送方发送消息时

每个线程发送的消息，新定义一个消息队列。

session.createQueue("队列名字");

2、在消息消费方监听方式接收消息的时候是通过监听多个队列实现

配置监听多条队列的方式为：

jms.queue.nameTwo=queue1,queue2

**[java]** [view plain](http://blog.csdn.net/lpp_dd/article/details/70916076) [copy](http://blog.csdn.net/lpp_dd/article/details/70916076)

1. <span style="font-size:18px;"><!-- 配置消息监听器 -->
2. <bean **class**="org.springframework.jms.listener.DefaultMessageListenerContainer">
3. <property name="connectionFactory" ref="connectionFactory"/>
4. <property name="destinationName" value="${jms.queue.name}"/>
5. <property name="messageListener" ref="messageReceiver"/>
6. </bean></span>

多个队列之间用逗号隔开。

5、消息队列的管理

当我们频繁的创建了消息队列之后通过在activemq.xml文件中添加配置  实现定期删除不用的消息队列

当然，也可以通过配置，使得broker可以自动探测到无用的队列（一定时间内为空的队列）并删除掉，回收响应资源。

配置如下：

<broker xmlns="http://activemq.apache.org/schema/core" schedulePeriodForDestinationPurge="10000">

<destinationPolicy>

<policyMap>

<policyEntries>

<policyEntry queue=">" gcInactiveDestinations="true" inactiveTimoutBeforeGC="30000"/>

</policyEntries>

</policyMap>

</destinationPolicy>

</broker>

schedulePeriodForDestinationPurge：10000  每十秒检查一次，默认为0，此功能关闭

gcInactiveDestinations： true  删除掉不活动队列，默认为false

inactiveTimoutBeforeGC：30000 不活动30秒后删除，默认为60秒

6、消息队列的安全设置

1、控制台安全配置，打开conf/jetty.xml文件，找到

    <bean id="securityConstraint" class="org.eclipse.jetty.http.security.Constraint">

        <property name="name" value="BASIC" />

        <property name="roles" value="admin" />

        <property name="authenticate" value="false" />

    </bean>

   将“false”改为“true”即可。用户名和密码存放在conf/jetty-realm.properties文件中。

2、生产者和消费者连接MQ需要密码

   打开conf/activemq.xml文件，在<broker>标签里的<systemUsage>标签前加入：

<plugins>

<simpleAuthenticationPlugin>

<users>

<authenticationUser username="${activemq.username}" password="${activemq.password}" groups="users,admins"/>

</users>

</simpleAuthenticationPlugin>

</plugins>

注意必须在<systemUsage>标签前，否则启动ActiveMQ会报错。

连接activemq生产者与消费者的账户与密码配置位置为

conf路径下的credentials.properties文件

(手动试一下)

7、消息队列的持久化方式

1）KahaDb和AMQ Message Store两种持久方式如何选择？

官方：

From 5.3 onwards - we recommend you use KahaDB - which offers improved scalability and recoverability over the AMQ Message Store.

The AMQ Message Store which although faster than KahaDB - does not scales as well as KahaDB and recovery times take longer.

非官方：

kaha文件系统实际上上是一个文件索引系统，有两部分组成，一个是数据文件系统，由一个个独立的文件组成，缺省文件大小是32M大（可配置），另外一个是索引文件系统，记录消息在数据文件中的位置信息以及数据文件中的空闲块信息。数据文件是存储到硬盘上的，索引文件是缓存在内存中的。所以这个存储系统对大消息存储有利，象我们的memberId之类的文本消息，实际上是浪费，索引比消息还大，哈。

推荐： Amq持久方式

理由：虽然官方推荐使用KahaDB持久方式，但其提到的优势：可伸缩性和恢复性较好，对于我们实际的应用意义不大。从我们自己的使用经验来看，KahaDB持久方式，Data文件是一个大文件（感觉文件过大后，造成队列服务瘫死的可能性会增大），从官网的相关配置（附录1）也找不到哪里可以设置数据的文件的最大Size。）而Amq持久方式可以设置Data文件最大Size，这样可以保证即时消息积压很多，Data文件也不至于过大。

**[java]** [view plain](http://blog.csdn.net/lpp_dd/article/details/70916076) [copy](http://blog.csdn.net/lpp_dd/article/details/70916076)

1. <!-- 消息持久化方式 -->
2. <persistenceAdapter>
3. <amqPersistenceAdapter directory="${<SPAN class=hilite1>activemq</SPAN>.base}/data"/>
4. </persistenceAdapter>

8、java main方法方式发送与接收

发送方代码：

**[java]** [view plain](http://blog.csdn.net/lpp_dd/article/details/70916076) [copy](http://blog.csdn.net/lpp_dd/article/details/70916076)

1. **public** **static** **void** main(String[] args) {
2. //初始化消息队列名字
3. String queueName = "";
4. String username = "";
5. String password = "";
6. String brokerURL = "";
7. //连接mq并发送数据
8. **try** {
9. //获取连接工厂 通过构造方法初始化 用户名 密码 连接地址
10. ConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory(username,password,brokerURL);
11. //获取连接对象
12. Connection connection = connectionFactory.createConnection();
13. //开启连接对象
14. connection.start();
15. /\*  获取session会话对象
16. connection.createSession(paramA,paramB);
17. paramA 取值有：
18. 1、true：支持事务
19. 为true时：paramB的值忽略， acknowledgment mode被jms服务器设置为SESSION\_TRANSACTED 。
20. 2、false：不支持事务
21. 为false时：paramB的值可为Session.AUTO\_ACKNOWLEDGE、Session.CLIENT\_ACKNOWLEDGE、DUPS\_OK\_ACKNOWLEDGE其中一个。
22. paramB 取值有：
23. 1、Session.AUTO\_ACKNOWLEDGE：为自动确认，客户端发送和接收消息不需要做额外的工作。
24. 2、Session.CLIENT\_ACKNOWLEDGE：为客户端确认。客户端接收到消息后，必须调用javax.jms.Message的acknowledge方法。jms服务器才会删除消息。
25. 3、DUPS\_OK\_ACKNOWLEDGE：允许副本的确认模式。一旦接收方应用程序的方法调用从处理消息处返回，会话对象就会确认消息的接收；而且允许重复确认。在需要考虑资源使用时，这种模式非常有效。
26. 4、SESSION\_TRANSACTED
27. \*/
28. Session session = connection.createSession(Boolean.FALSE,Session.AUTO\_ACKNOWLEDGE);
29. //设置订阅的目的地  String参数表示 通过名字进行区分
30. Destination destination = session.createQueue(queueName);
31. //获取消息发送者
32. MessageProducer producer = session.createProducer(destination);
33. //获取发送消息的载体  这里不用多态是因为 父类Message里面没有子类的特有方法setText
34. TextMessage message = session.createTextMessage();
35. //向消息里面添加key-value键值对  可以作为接收时的参数
36. message.setStringProperty("key","value");
37. //需要发送的消息文本
38. message.setText("sendActivemqMessageContent");
39. //发送消息
40. producer.send(message);
41. //流格式的消息 创建与发送
42. /\*
43. 创建StreamMessage流消息 通过向流消息里面写文件流发送消息
44. byte[] content = new byte[bytelength];
45. int i = 0;
46. InputStream ins = new FileInputStream(new File(filepath));
48. BufferedInputStream bins = new BufferedInputStream(ins);
49. while ((i = bins.read(content)) > 0) {
50. message = session.createStreamMessage();
51. message.setStringProperty("FILE\_NAME", FILE\_NAME);
52. message.setStringProperty("COMMAND", "sending");
53. message.clearBody();
54. message.writeBytes(content,0,i);
55. producer.send(message);
56. }
57. bins.close();
58. ins.close();
60. \*/
61. } **catch** (Exception e) {
62. e.printStackTrace();
63. }

接收方代码：

**[java]** [view plain](http://blog.csdn.net/lpp_dd/article/details/70916076) [copy](http://blog.csdn.net/lpp_dd/article/details/70916076)

1. **public** List<Object> receiveMessage(Message message) **throws** Exception {
2. List<Object> result = **new** ArrayList<Object>();
3. String file\_name = "";
4. **boolean** appended = **false**;
5. **try** {
6. **if** (message == **null**) {
7. }
9. **if** (message **instanceof** StreamMessage) {
10. StreamMessage streamMessage = (StreamMessage) message;
11. String command = streamMessage.getStringProperty("COMMAND");
12. **if** ("start".equals(command)) {
13. appended = **false**;
14. result.add(appended);
15. //开始发消息时创建文件输出流
16. file\_name = message.getStringProperty("FILE\_NAME");
17. fos = **new** FileOutputStream(cfilepath + file\_name, appended);
18. bos = **new** BufferedOutputStream(fos);
20. }
21. **if** ("sending".equals(command)) {
22. **byte**[] content = **new** **byte**[bytelength];
23. **int** i = 0;
24. **try** {
25. **while** ((i = streamMessage.readBytes(content)) > 0) {
26. bos.write(content,0,i);
27. bos.flush();
28. }
30. } **catch** (MessageEOFException e) {
31. **throw** e;
32. }
34. result.add(appended);
35. }
36. **if** ("end".equals(command)) {
37. file\_name = message.getStringProperty("FILE\_NAME");
38. String sign = message.getStringProperty("type");
39. appended = **true**;
40. result.add(appended);
41. result.add(file\_name);
42. result.add(sign);
43. //关流
44. bos.close();
45. fos.close();
46. }
47. }
48. } **catch** (JMSException e) {
50. }
52. **return** result;
53. }