一、 ActiveMQ的部署

1.1 下载  
下载地址：http://activemq.apache.org   
1.2 安装  
υ 安装JDK（1.4以上即可）。  
υ 设置JAVA\_HOME环境变量。  
υ 直接解压ActiveMQ压缩包。  
默认配置下，执行解压目录下bin/activemq 即可正常运行。之后，可以通过访问http://localhost:8161/admin查看ActiveMQ的运行情况（默认用户名和密码为admin/admin）。

我们也可以把日志输出到指定的日志文件中

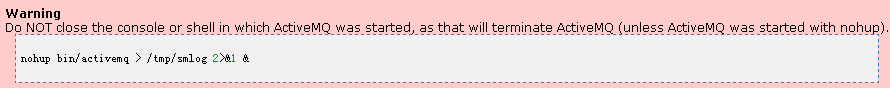
Shell代码  [收藏代码](javascript:void())

1. ./activemq start > /tmp/smlog  2>&1 &

然后日志就输出到/tmp/smlog文件中了

注意：官方文档中标明了如果按上述的方式启动后，如果我们把linux终端窗口关闭的话，则服务也会停止

具体如下：



不过，我尝试了一下关闭xshell终端，貌似服务依然正在，没有退出。

不过为了保险起见，我们在启动时最好还是加上nohup参数

启动命令如下：

**Xshell代码  [收藏代码](javascript:void())**

1. nohup ./activemq start

或者：

**Shell代码  [收藏代码](javascript:void())**

1. nohup ./activemq start > /tmp/smlog 2>&1 &

1.3 配置  
ActiveMQ的配置存放在安装目录的conf/activemq.xml文件中。  
因为ActiveMQ采用了Jetty作为容器，因此Jetty相关的配置在conf/jetty.xml文件中。

二、 ActiveMQ的使用

ActiveMQ的使用分为两种，一种是嵌入式，即把ActiveMQ作为内嵌的JMS provider集成到Tomcat或其他Web服务器中；另一种是独立部署，即ActiveMQ单独部署，独立运行，通过ActiveMQ提供的API进行进程外访问。  
我们采用独立部署的方式使用ActiveMQ，这样的做法更有利于业务的解耦和工程部署结构的分离；也有利于对ActiveMQ进行配置定制和优化。  
我们知道JMS规范中约定了两种消息发送方式：P2P和Publish/Subscribe。根据业务的需求，我们采用P/S方式更为合理，也能更好的发挥ActiveMQ本身的优势。  
2.1 消息的发布  
// 消息的发布

ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("tcp://192.168.1.175:61616");

Connection connection = factory.createConnection();

connection.start();

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 创建一个MQ

Queue queue = new ActiveMQQueue("testQueue");

MessageProducer producer = session.createProducer(queue);

producer.setDeliveryMode(DeliveryMode.PERSISTENT);

TextMessage message = session.createTextMessage("A persistent Message");

// 发布主题消息

producer.send(message);

producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);

producer.send(session.createTextMessage("A non persistent Message"));

System.out.println("Send messages sucessfully!");

2.2 消息的订阅  
// 消息的订阅

ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("tcp://192.168.1.175:61616");

Connection connection = factory.createConnection();

connection.start();

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

// 创建主题

Queue queue = new ActiveMQQueue("testQueue");

// 创建订阅

MessageConsumer consumer = session.createConsumer(queue);

consumer.setMessageListener(new MessageListener() {

// 订阅接收方法

public void onMessage(Message message) {

TextMessage tm = (TextMessage) message;

try {

System.out.println("Received message: " + tm.getText());

} catch (JMSException e) {

e.printStackTrace();

}

}

});

2.3 消息的持久订阅  
// 连接到ActiveMQ服务器  
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("tcp://localhost:61616");  
Connection connection = factory.createConnection();  
String ip = "";  
try {  
 // 获取本机IP  
 InetAddress addr = InetAddress.getLocalHost();  
 ip = addr.getHostAddress().toString();  
} catch (UnknownHostException ex) {  
 ex.printStackTrace();  
 ip = "";  
}  
if(!"".equals(ip)) {  
 System.out.println("CLIENT: " + ip);  
 // 设置订阅客户端ID  
 connection.setClientID(ip);  
 connection.start();  
Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);  
// 创建主题  
Topic topic = session.createTopic("myTopic.messages");  
// 创建订阅  
MessageConsumer consumer = session.createDurableSubscriber(topic, "test");

consumer.setMessageListener(new MessageListener() {  
// 订阅接收方法  
 public void onMessage(Message message) {  
 TextMessage tm = (TextMessage) message;  
 try {  
 System.out.println("Received message: " + tm.getText());  
 } catch (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
});  
}

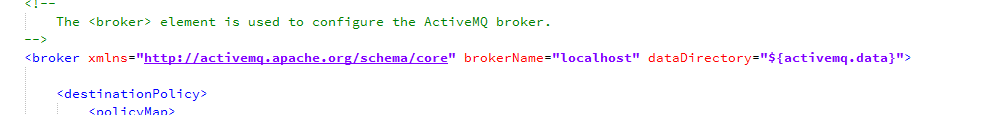
2.4 注意事项  
υ 普通订阅的情况下，客户端只有在连接到服务器的情况下，才能接收服务器上的主题消息。  
υ 持久订阅后，当客户端在线时，服务器端会把客户端在上次下线之后到本次上线之间的所有消息一并推送给客户端；这样就保证了客户端不会有丢失的消息。  
υ 持久订阅会引发另一个问题：当新增一个订阅客户端时，这个客户端会收到服务器上该主题下的所有未过期消息。

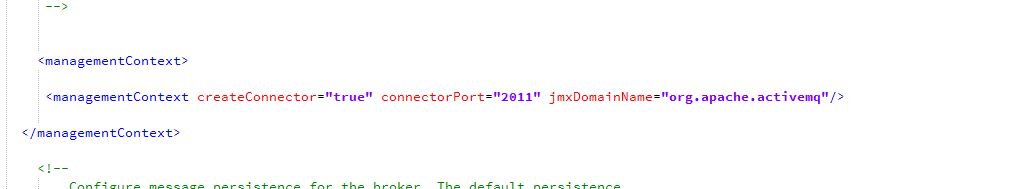
3 其他

3.1 消息持久化  
默认的情况下，ActiveMQ的消息持久化是基于文件系统的KahaDB。我们可以通过配置，让ActiveMQ使用MySQL实现消息持久化：  
υ 将MySQL的jar包复制到安装目录的lib下。  
υ 修改配置文件：  
<persistenceAdapter>  
<jdbcPersistenceAdapter dataDirectory="${activemq.base}/data" dataSource="#derby-ds" />  
</persistenceAdapter>  
υ 增加节点（与broker节点同级）：  
<bean id="derby-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>  
υ 然后，重启ActiveMQ即可。  
3.2 集群  
ActiveMQ的集群由服务器端和客户端共同完成。服务器端通过部署Master/Slaver机制，通过进行分布式部署，以实现服务器集群的平行扩展。而客户端则采取静态地址发现，或者动态地址发现的方式，实现服务器的负载均衡选择。  
3.2.1 服务器端的部署  
ActiveMQ支持Master/Slaver机制，但简单Master/Slaver方式有一定的局限性，不适合服务器集群的平行扩展（当然，简单Master/Slaver已经足够支撑一般的商业应用）。因此，ActiveMQ提供了支持大并发请求的集群方式：共享文件系统的集群，以及基于JDBC的集群。  
υ 共享文件系统的集群  
实际上就是基于文件系统进行集群部署（前面提到过，ActiveMQ默认的消息存储就是基于文件系统的），可以通过分布式存储系统或共享数据目录来实现。这种方式只需要修改conf/activemq.xml：  
<persistenceAdapter>  
<journaledJDBC dataDirectory="/sharedFileSystem/broker"/>  
</persistenceAdapter>  
υ 基于JDBC的集群  
原理与共享文件系统一致，只不过把文件系统换成了数据库平台。即：多台ActiveMQ连接同一个数据库，从而实现ActiveMQ的服务器集群。配置同5.1。  
3.2.2 客户端的使用  
服务器端的集群对客户端而言是透明的，但如果客户端希望得到集群和负载均衡的功能支持，则必须在代码中有所体现。  
最常规的方法就是failover协议，fileover支持客户端在当前服务器断开的情况下，自动重新连接到新的服务器上，而新的服务器地址可以来源于静态地址列表，也可以来源于动态地址广播。  
υ 静态地址发现的常规用法  
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("failover:(tcp://primary:61616,tcp://secondary:61616)?randomize=false");  
υ 动态地址发现的常规用法  
ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory("failover:(multicast://host:6255)");  
当然，ActiveMQ还支持更多的协议，如：fanout、discovery等。  
3.3 管理与监控  
ActiveMQ提供了一个Web后台用于查看服务器运行状态，并提供了对消息队列、主题及订阅者等进行管理的功能。  
另外，ActiveMQ也可以通过配置支持Nagios的集成监控。

4 activemq的监控

采用jmx监控，需要配置activemq.xml文件，在此节点下加入代码，





<managementContext>

<managementContext createConnector="true" connectorPort="2011" jmxDomainName="org.apache.activemq"/>

</managementContext>

Java代码如下：

**public** **class** JmsMonitoring {

**public** **static** Logger *logger* = LoggerFactory.*getLogger*(JmsMonitoring.**class**);

**public** **static** **void** main(String[] args) **throws** IOException, MalformedObjectNameException {

JMXServiceURL url = **new** JMXServiceURL("service:jmx:rmi:///jndi/rmi://192.168.1.175:2011/jmxrmi");

JMXConnector connector = JMXConnectorFactory.*connect*(url, **null**);

connector.connect();

MBeanServerConnection connection = connector.getMBeanServerConnection();

// 需要注意的是，这里的org.apache.activemq必须和上面配置的名称相同

ObjectName name = **new** ObjectName("org.apache.activemq:type=Broker,brokerName=localhost");

BrokerViewMBean mBean = (BrokerViewMBean) MBeanServerInvocationHandler.*newProxyInstance*(connection, name,

BrokerViewMBean.**class**, **true**);

**for** (ObjectName queueName : mBean.getQueues()) {

QueueViewMBean queueMBean = (QueueViewMBean) MBeanServerInvocationHandler.*newProxyInstance*(connection,

queueName, QueueViewMBean.**class**, **true**);

System.***out***.println("\n------------------------------\n");

// 消息队列名称

System.***out***.println("States for queue --- " + queueMBean.getName());

// 队列中剩余的消息数

System.***out***.println("Size --- " + queueMBean.getQueueSize());

// 消费者数

System.***out***.println("Number of consumers --- " + queueMBean.getConsumerCount());

// 出队数

System.***out***.println("Number of dequeue ---" + queueMBean.getDequeueCount());

}

}

}