## 高并发发送消息异常解决方法：

现象：使用10个线程每100ms发送一条消息，大约3000多条后，出现异常，所有线程停

止： javax.jms.JMSException: Could not connect to broker

URL: tcp://localhost:61616. Reason:java.net.BindException: Address already in use: connect; nested exception is

java.net.BindException: Address already in use: connect

原因：创建了太多jms连接没有来得及回收

解决方法：使用jms连接池

原来的配置：

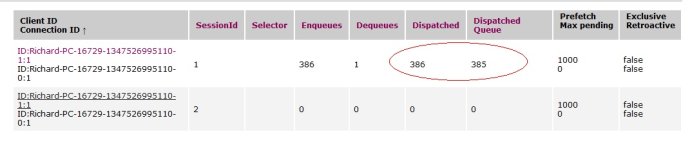
|  |
| --- |
| <bean>  <property name="environment">  <props>  <prop key="java.naming.factory.initial">  org.apache.activemq.jndi.ActiveMQInitialContextFactory  </prop>  <prop key="java.naming.provider.url">tcp://huzq-linux:61616</prop>  </props>  </property>  </bean>  <bean>  <property name="jndiName">  <value>ConnectionFactory</value>  </property>  <property name="jndiTemplate">  <ref local="jndiTemplate"></ref>  </property>  </bean> |

修改为：

|  |
| --- |
| <bean>  <property name="connectionFactory">  <bean>  <property name="brokerURL" value="tcp://huzq-linux:61616" />  </bean>  </property>  </bean> |

## 解决activemq多消费者并发处理

遇到一个现象，如果activemq队列积压了数据的话，如果在spring中启动listner，只有一个consumer执行，查阅了很多资料，无果，后来偶尔通过activemq的监控网页看到消费者列表中，只有一个消费者有等待处理的数据，其他都没有，如下图：

[](http://photo.blog.sina.com.cn/showpic.html#blogid=5376c7190101aiq9&url=http://s8.sinaimg.cn/orignal/5376c719t7a8e8f704037)  
由此得知，activemq有一定机制将队列中的数据交给consumer处理，这个机制就是数据的数量分配，查资料得知，默认是1000，因此，把这个值调小就可以了。

在客户端的连接url中，修改为tcp://ipaddr:61616?jms.prefetchPolicy.all=2

这样基本消费者就分配公平了，不会出现一个消费者忙死，另外的消费者闲死了。

## 为高并发程序部署ActiveMQ

使用ActiveMQ来扩展你的应用程序需要一些时间并要花一些精力.本节中我们将介绍三种技术用于扩展应用程序.我们将从垂直扩展开始,这种扩展方式中,单个代理需要处理成千上万的连接和消息队列.

接下来我们将介绍水平扩展,这种扩展方式需要处理比前一种方式更多的网络连接.最后,我们介绍的传输负载分流,可以在扩展和性能间得到平衡,但是会增加ActiveMQ程序的复杂性.

1. 垂直扩展:

垂直扩展是一种用于增加单个ActiveMQ代理连接数(因而也增加了负载能力)的技术.默认情况下,ActiveMQ的被设计成尽可高效的传输消息以确保低延迟和良好的性能.但是,你也可以进行一些配置使的ActiveMQ代理可以同时处理大量并发的连接以及大量的消息队列.

默认情况下,ActiveMQ使用阻塞IO来处理传输连接,这种方式为每一个连接分配一个线程.你可以为ActiveMQ代理使用非阻塞IO(同时客户端可以使用默认的传输)以减少线程的使用.可以在ActiveMQ的配置文件中通过传输连接器配置非阻塞IO.下面的是配置非阻塞IO的示例

代码：配置NIO传输连接器

|  |
| --- |
| <broker>  <transportConnectors>  <transportConnector name="nio" uri="nio://localhost:61616"/>  </<transportConnectors>  </broker> |

除了为每个连接使用一个线程的阻塞IO,ActiveMQ还可以为每一个客户端连接使用一个消息分发线程.你可以通过将系统参数org.apache.activemq.UseDedicatedTaskRunner设置为false来设置ActiveMQ使用一个搞线程池.下面是一个示例:

|  |
| --- |
| ACTIVEMQ\_OPTS="-Dorg.apache.activemq.UseDedicatedTaskRunner=false" |

确保ActiveMQ代理用于足够的内存来处理大量的并发连接,需要分两步进行:

首先,你需要确保运行ActiveMQ的JVM在启动之前已经配置了足够的内存.可以使用

JVM的-Xmx选项来配置,如下所示:

|  |
| --- |
| ACTIVEMQ\_OPTS="-Xmx1024M -Dorg.apache.activemq.UseDedicatedTaskRunner=false" |

其次,需要确保JVM配置了适量的专门供ActiveMQ代理使用的内存.这个配置可用通过<system-Usage> 元素的limit属性来配置.一个不错的根据经验得到的规则时,在连接数为几百个时配置512MB为最小内存.

如果测试发现内存不够用,可以增加内存配置.你可以按照下面代码示例来配置ActiveMQ使用的内存限制:

代码:为ActiveMQ代理设置内存使用限制

|  |
| --- |
| <systemUsage>  <systemUsage>  <memoryUsage>  <memoryUsage limit="512 mb"/>  </memoryUsage>    <storeUsage>  <storeUsage limit="10 gb" name="foo"/>  </storeUsage>    <tempUsage>  <tempUsage limit="1 gb"/>  </tempUsage>    </systemUsage>  </systemUsage> |

同样,简易减少每个连接的CPU负载.如果你正使用Open-Wire格式的消息,关闭tight encoding选项,开启该选项会导致CPU占有过多.Tight encoding选项可以通过客户端连接的URI中的参数设置以便关闭该选项.下面是示例代码:

|  |
| --- |
| String uri = "failover://(tcp://localhost:61616?" + wireFormat.tightEncodingEnabled=false)";  ConnectionFactory cf = new ActiveMQConnectionFactory(uri); |

了解了一些扩展ActiveMQ代理处理大量连接的调优选项之后,我们在了解一些让ActiveMQ处理大量消息队列的调优选项.

默认的消息队列配置中使用一个独立的线程负责将消息存储中的消息提取到消息队列中而后再被分发到对其感兴趣的消息消费者.如果有大量的消息队列,建议通过启用optimizeDispatch这个属性

改善这个特性,示例代码如下所示:

|  |
| --- |
| <destinationPolicy>  <policyMap>  <policyEntries>  <policyEntry queue=">" optimizedDispatch="true"/>  </policyEntries>  </policyMap>  </destinationPolicy> |

注意,代码清单中使用通配符>表示该配置会递归的应用到所有的消息队列中.

为确保扩展配置既可以处理大量连接也可以处理海量消息队列,请使用JDBC或更新更快的KahaDB消息存储.默认情况下ActiveMQ使用KahaDB消息存储.

到目前位置,我们关注了连接数扩展,减少线程使用以及选择正确的消息存储.下面的示例配置代码展示了ActiveMQ配置中为扩展进行了调优:

代码：为扩展进行调优的配置示例代码

|  |
| --- |
| <broker xmlns="http://activemq.apache.org/schema/core" brokerName="amq-broker" dataDirectory="${activemq.base}/data">    <persistenceAdapter>  <kahaDB directory="${activemq.base}/data" journalMaxFileLength="32mb"/>  </persistenceAdapter>    <destinationPolicy>  <policyMap>  <policyEntries>  <policyEntry queue="&gt;" optimizedDispatch="true"/>  </policyEntries>  </policyMap>  </destinationPolicy>    <systemUsage>  <systemUsage>    <memoryUsage>  <memoryUsage limit="512 mb"/>  </memoryUsage>    <storeUsage>  <storeUsage limit="10 gb" name="foo"/>  </storeUsage>    <tempUsage>  <tempUsage limit="1 gb"/>  </tempUsage>    </systemUsage>  </systemUsage>    <transportConnectors>  <transportConnector name="openwire" uri="nio://localhost:61616"/>  </transportConnectors>  </broker> |

注意示例代码中所有为调优而建议的配置条目,这些调优条目在默认的配置文件中并没有配置,所以请确保给予充分重视.

了解过如何扩展ActiveMQ后,现在是时候了解使用代理网络来进行横向扩展了.

1. 横向扩展

除了扩展单独的代理,你还可以使用代理网络来增加应用程序可用的代理数量.因为网络会自动传递消息给所有互联的具有对消息感兴趣的消息消费者的代理,所以你可以配置客户端连接到一个代理集群,随机的选择集群中的一个代理来连接.可以通过URI中的参数来配置,如下所示:

|  |
| --- |
| failover://(tcp://broker1:61616,tcp://broker2:61616)?randomize=true |

为了确保队列或持久化主题中的消息不会卡在某个代理上而不能进行转发,需要在配置网络连接时,将dynamicOnly配置成true并使用小一点的prefetchSize.下面是一个示例:

|  |
| --- |
| <networkConnector uri="static://(tcp://remotehost:61617)"  name="bridge" dynamicOnly="true" prefetchSize="1"  </networkConnector> |

示例代理网络来横向扩展并不会的代理更多的延迟,因为消息在传送到消息消费者在之前会经过多个代理.另外一种可选的部署方案可以提供更多的扩展性和更好的性能,但是需要在应用程序中做更多的计划.这种混合的解决方案被称为传输负载分流(traffic partitioning),这种方案通过在应用程序中分割消息目的地到不同的代理上以完成垂直扩展.

1. 传输负载分流

客户端的传输负载分流是一个垂直和水平混合的负载分流方案.通常不使用代理网络,因为客户端程序会决定将哪个负载发送到哪个(些)代理上.客户端程序需要维护多个JMS连接并且决定哪个JMS连接应该用于那个消息目的地.

没有直接使用网络连接的好处是降低了代理见过量的消息转发.你不需要像在传统程序中那样进行额外的负载的均衡处理.到这里,我们已经了解了垂直和水平扩展以及传输负载分流.你应该能够深刻了解如何使用ActiveMQ来处理大量的并发连接和海量的消息目的地的连接了.

## Activemq在大流量停出现内存耗尽的情况以及解决方案

在大量消息持续发送到broker的情况下，当broker到消费者之间的网络满了以后，broker的消息无法发送出去，导致在TransportConnection的dispatchQueue中堆积的消息越来越多。PendingMessageCursor中的消息不能被及时消费，导致broker判断消费者为慢消费者。当broker的内存被耗尽后JVM会频繁的进行full gc，由于消息不能被回收，所以消息对象会从年轻代转移到老年代而不会释放内存，导致broker几乎停止对外服务。

这个问题的根本原因是ActiveMQ只对接收消息作了流量控制，但是没有发送消息堵塞的情况。需要根据消息发送情况来控制消息的接收。

解决方案，在TransportConnection中的dispatchAsync对dispatchQueue中的消息数量做判断，当超过阈值就暂停dispatch，当前thread sleep，这样TopicSubscription就会暂停接收消息，避免内存耗尽