# ActiveMQ--6.消费者负载均衡与高可用

ActiveMQ Broker提供基于LevelDB复制的方式提供高可用服务，但是对负载均衡做的很弱，只支持Static的服务器之间转发。目前比较流行的消息分片竟然不支持。但是消费者的负载均衡和高可用还是比较完善的。另外说一下，生产者的高可用和负载均衡，一般是靠外围程序控制。比如，基于Tomcat的web程序作为生产者，那么这个web程序的高可用，需要靠tomcat等外围程序。所以一般所说的高可用，主要指Broker和Consumer。

下面介绍一下几个常用的消费者策略。

**Exclusive Consumer：**

用于处理Queue的高可用。如果同时使用多个消费者从同一个Queue消费消息，那么消息的顺序性将得不到保证。这时候可以使用Exclusive Consumer。使用Exclusive Consumer可以保证只有一个消费者在消费这个Queue，其他的消费者处于等待状态。一旦处理消费状态的消费者不可用，系统会自动使用失效转移机制，选择到一个新的消费者继续消费。

使用如下：

|  |  |
| --- | --- |
| 1 | queue =  new  ActiveMQQueue( "TEST.QUEUE?consumer.exclusive=true" ); |

只需要在Destination上增加consumer.exclusive=true参数即可。

还可以给消费者设置优先级，用于针对网络和服务器资源不同的情况。

|  |  |
| --- | --- |
| 1 | queue =  new  ActiveMQQueue( "TEST.QUEUE?consumer.exclusive=true&consumer.priority=10" ); |

**Message Groups：**

ActiveMQ 5.3版本新增加了这个功能。Message Groups可以看做Exclusive Consumer的升级版，是一个可以并行的Exclusive Consumer。原理是通过使用**JMSXGroupID**来定义消息组。拥有相同的JMSXGroupID的消息将发送到同一个Queue，这个技术类似于Sesssion Sticky技术。这样既可以保证消费者的高可用（因为可以有多个消费者消费同一队列），又可以保证消息按顺序消费，还不会像Exclusive Consumer一样浪费资源（需要额外的处于等待状态的消费者待命）。如果消费者被关闭或者消息组被关闭，这个拥有JMSXGroupID的消息会自动被发送到其他消费者。

设置JMSXGroupID的例子如下：

|  |  |
| --- | --- |
| 1  2  3  4 | Mesasge message = session.createTextMessage( "foo" );  message.setStringProperty( "JMSXGroupID" ,  "your business key" );  ...  producer.send(message); |

关闭消息组，通过设置JMSXGroupSeq的值为-1，例子如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | Mesasge message = session.createTextMessage( "foo" );  message.setStringProperty( "JMSXGroupID" ,  "your business key" );  message.setIntProperty( "JMSXGroupSeq" , - 1 );  // ...  producer.send(message); |

可以使用JMSXGroupFirstForConsumer来判断这个消费者是否是第一次消费这个JMSXGroupID的消息：

|  |  |
| --- | --- |
| 1  2  3 | if  (message.getBooleanProperty( "JMSXGroupFirstForConsumer" )) {      // flush cache for groupId  } |

 如果Broker中已经有消息了，这时由于启动消费者的速度不一致，可能会导致某些消费者先启动并率先消费消息，导致A消费者的负载不均匀。可以在Broker中配置timeBeforeDispatchStarts，让消费者延迟一段时间再开始负载消费。或者配置consumersBeforeDispatchStarts，让消费者达到一定数量再开始负载消费。

具体配置如下，修改${ACTIVEMQ\_HOME}\conf\activemq.xml：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | < destinationPolicy >     < policyMap >       < policyEntries >         < policyEntry  queue=">" consumersBeforeDispatchStarts="2" timeBeforeDispatchStarts="2000"/>       </ policyEntries >     </ policyMap >  </ destinationPolicy > |

设置有两个消费者都启动好或者2秒之后，在开始负载消费消息。

**Virtual Topics：**

Message Groups只能用于Queue，那么Topic对应的版本，就是这个Virtual Topics。他实现了和Message Groups类似的功能，负载均衡和失效转移。

对于消息的生产者来说，Virtual Topics只是一个普通的Topic，但是必须以**VirtualTopic.**（可配置）开头，如VirtualTopic.Foo。

而消息的消费者连接是一个队列。这个队列需要遵从以下规则，即**Consumer.ClientID.VirtualTopicName**。例如：Consumer.CilentA.VirtualTopic.Foo，表示消费者的Client id是ClientA，消费的是VirtualTopic.Foo这个Topic。

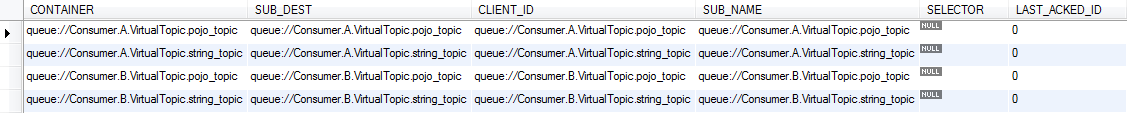
消息生产者代码示例：

|  |  |
| --- | --- |
| 1  2  3 | MessageProducer producer = session.createProducer( new  ActiveMQTopic( "VirtualTopic.FOO" ));  TextMessage message = session.createTextMessage( "foo" );  producer.send(message); |

消息消费者代码示例：

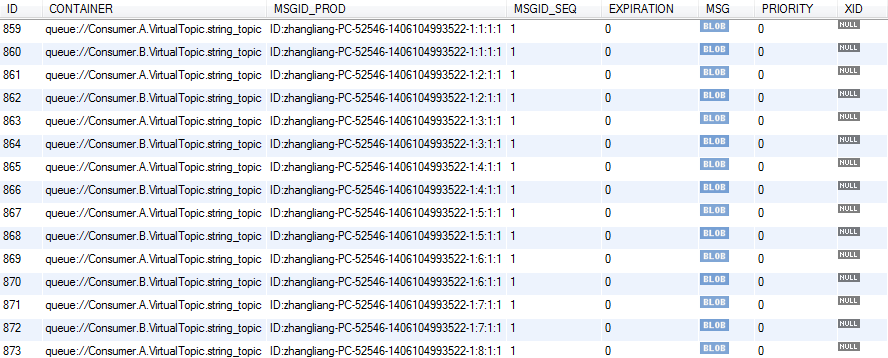
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | MessageConsumer consumerA = session.createConsumer( new  ActiveMQQueue( "Consumer.ClientA.VirtualTopic.FOO" ));  consumerA.setMessageListener( new  MessageListener() {       public  void  onMessage(Message message) {           // do something ...       }  }); |

可以看到，通过Virtual Topics注册的queue订阅关系如下：



这里使用了2个client，A和B，分别订阅了两个主题pojo\_topic和string\_topic。

而消息的存储格式如下：



通过上图可以看到，消息是以Queue的形式存储，并非Topic，但是Queue被直接根据所订阅的ClientID生成多份消息。比如，VirtualTopic.string\_topic有两个订阅者A和B，那消息就被分成Consumer.A.VirtualTopic.string\_topic和Consumer.B.VirtualTopic.string\_topic。这就是将Topic通过Virtual Topics转换成了Queue。

只要转换成了Queue，就可以结合使用刚才介绍的Exclusive Consumer或者Message Groups对这个VirtualTopic的队列进行高可用和负载均衡的配置，从而实现Topic的高可用和负载均衡。

可以配置Broker，改变虚拟主题的默认前缀，如下面的配置，则表示虚拟主题的前缀是VirtualTopicConsumers。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | < broker  xmlns = "<http://activemq.apache.org/schema/core>" >       < destinationInterceptors >         < virtualDestinationInterceptor >           < virtualDestinations >             < virtualTopic  name=">" prefix="VirtualTopicConsumers.\*." selectorAware="true"/>           </ virtualDestinations >         </ virtualDestinationInterceptor >       </ destinationInterceptors >  </ broker > |

从ActiveMQ 5.4版本开始，可以配置selectorAware属性控制只有符合订阅者规则的消息才被分发给相应的虚拟队列，用于防止分发不匹配的消息，提升效率。

说了优点，当然也要说缺点。优点是可以实现高可用和负载均衡；缺点是，如果订阅者很多，每个订阅者都需要复制一份消息，这样会占用过多的磁盘空间，造成消息爆炸。

**Composite Destinations：**

可以将消息发送给多个Destination。可以混合发送Queue和Topic。例如：

|  |  |
| --- | --- |
| 1 | Queue queue =  new  ActiveMQQueue( "FOO.A,<topic://NOTIFY.FOO.A>" ); |

个人感觉这个功能比较鸡肋，属于客户端控制服务器端的消息高可用。如果是纯粹的消息可用性复制，可以直接使用基于LevelDb的消息复制机制或基于JDBC的主从同步机制。