1. **应答模式选择**
2. 对于并发量比较大的数据，采用自动应答模式1（Session.AUTO\_ACKNOWLEDGE），因为自动应答模式，不管消费端消费的速度，只要消费端拿到消息，就会自动确认消息状态，队列就会删除消息，可以在程序中保证最终一致性，比如可以把消费过程中后续处理出现异常的消息存储在文件或者数据库中，用后台定时线程去取消息。
3. 对于并发量不大的消息数据，可以采用客户端手工确认应答模式2（Session.CLIENT\_ACKNOWLEDGE）或者单条消息确认4（INDIVIDUAL\_ACKNOWLEDGE ），可以在消费端消费的消息处理完成之后在手工确认消息的状态，可以一定程度上保证消息的可靠性，不易丢失消息。
4. **同步、异步消费**

1、同步消费

consumer.recieve()，调用时会一直阻塞，知道有消息过来，可以配置超时时间，防止一直阻塞。可以采用push和pull两种方式。

2、异步消费

采用spring监听器的方式，只能push方式，由broker推消息给监听者

使用同步方式receive()方法获取消息时，prefetch limit即可以设置为0，也可以设置为大于0

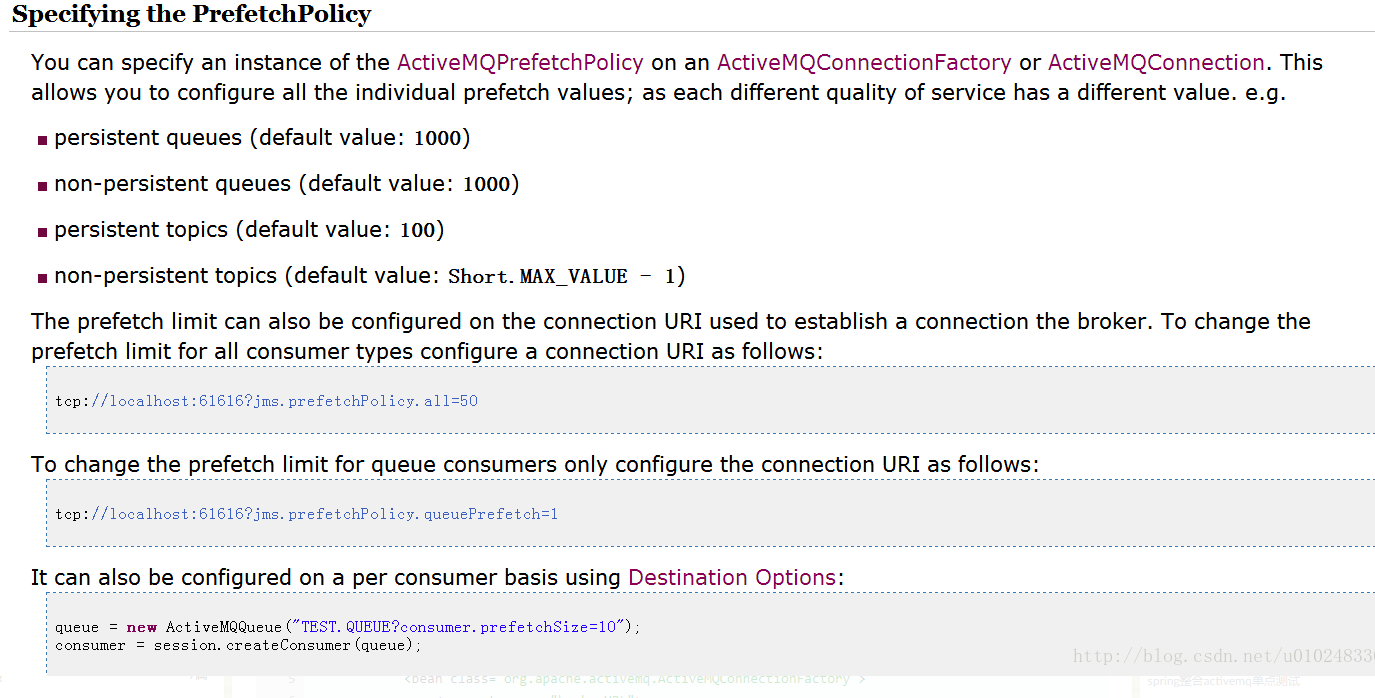
prefetch limit为零 意味着：“receive()方法将会首先发送一个PULL指令并阻塞，直到broker端返回消息为止，这也意味着消息只能逐个获取(类似于Request<->Response)”

prefetch limit 大于零 意味着：“broker端将会批量push给client 一定数量的消息(<= prefetch)，client端会把这些消息(unconsumedMessage)放入到本地的队列中，**只要此队列有消息，那么receive方法将会立即返回**，当一定量的消息ACK之后，broker端会继续批量push消息给client端。”

当使用MessageListener异步获取消息时，prefetch limit必须大于零了。因为，prefetch limit 等于零 意味着消息中间件不会主动给消费者Push消息，而此时消费者又用MessageListener被动获取消息(不会主动去轮询消息)。这二者是矛盾的。

1. **设置预取prefetch大小，改变pull和push方式**

1、java代码实现方式



1. spring方式

|  |
| --- |
| <!-- 定义PrefetchPolicy（预取策略） -->  <bean id="activeMQPrefetchPolicy" class="org.apache.activemq.ActiveMQPrefetchPolicy">  <property name="queuePrefetch" value="1"></property>  </bean>  <!-- 工厂 原厂商提供 -->  <bean id="activeMQConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">  <!-- 设置连接URL -->  <property name="brokerURL" value="tcp://192.168.20.210:61616"/>  <!-- 用户名、 -->  <property name="userName" value="admin"/>  <!-- 密码、 -->  <property name="password" value="admin"/>  <!-- 重发策略 -->  <!-- <property name="redeliveryPolicy" ref="activeMQRedeliveryPolicy"></property> -->  <!-- 预取策略 -->  <property name="prefetchPolicy" ref="activeMQPrefetchPolicy"></property>  </bean> |

**那prefetch limit的值设置为多少合适？视具体的应用场景而定。**

If you have very few messages and each message takes a very long time to process

you might want to set the prefetch value to 1 so that a consumer is given one message at a time.

如果消息的数量很少(生产者生产消息的速率不快)，但是每条消息 消费者需要很长的时间处理，那么prefetch limit设置为1比较合适。这样，消费者每次只会收到一条消息，当它处理完这条消息之后，向消息中间件发送ACK，此时消息中间件再向消费者推送下一条消息。

**prefetch limit 设置成0意味着什么？**

Specifying a prefetch limit of zero means the consumer will poll for more messages, one at a time,

instead of the message being pushed to the consumer.

意味着此时，消费者去轮询消息中间件获取消息。不再是Push方式了，而是Pull方式了。即消费者主动去消息中间件拉取消息。

perfetch limit是“消息预取”的值，这是针对消息中间件如何向消费者发消息 而设置的。与之相关的还有针对 消费者以何种方式向消息中间件返回确认ACK(响应)：比如消费者是每次消费一条消息之后就向消息中间件确认呢？还是采用“延迟确认”---即采用批量确认的方式(消费了若干条消息之后，统一再发ACK)。这就是 [Optimized Acknowledge](http://activemq.apache.org/performance-tuning.html)

ActiveMQ can acknowledge receipt of messages back to the broker in batches (to improve performance).

[引用 一段话](http://shift-alt-ctrl.iteye.com/blog/2020182)：“如果prefetchACK为true，那么prefetch必须大于0；当prefetchACK为false时，你可以指定prefetch为0以及任意大小的正数。  
不过，当prefetch=0是，表示consumer将使用PULL(拉取)的方式从broker端获取消息，broker端将不会主动push消息给client端，直到client端发送PullCommand时；  
当prefetch>0时，就开启了broker push模式，此后只要当client端消费且ACK了一定的消息之后，会立即push给client端多条消息。”

**那么，在程序中如何采用Push方式或者Pull方式呢？**

从是否阻塞来看，消费者有两种方式获取消息。同步方式和异步方式。

同步方式使用的是ActiveMQMessageConsumer的receive()方法。而异步方式则是采用消费者实现MessageListener接口，监听消息。

使用同步方式receive()方法获取消息时，prefetch limit即可以设置为0，也可以设置为大于0

prefetch limit为零 意味着：“receive()方法将会首先发送一个PULL指令并阻塞，直到broker端返回消息为止，这也意味着消息只能逐个获取(类似于Request<->Response)”

prefetch limit 大于零 意味着：“broker端将会批量push给client 一定数量的消息(<= prefetch)，client端会把这些消息(unconsumedMessage)放入到本地的队列中，**只要此队列有消息，那么receive方法将会立即返回**，当一定量的消息ACK之后，broker端会继续批量push消息给client端。”

当使用MessageListener异步获取消息时，prefetch limit必须大于零了。因为，prefetch limit 等于零 意味着消息中间件不会主动给消费者Push消息，而此时消费者又用MessageListener被动获取消息(不会主动去轮询消息)。这二者是矛盾的。

**此外，还有一个要注意的地方，即消费者采用同步获取消息(receive方法) 与 异步获取消息的方法(MessageListener) ，对消息的确认时机是不同的。**

1. **性能调节**
2. 异步生产

发送非持久化消息，需要设置投递模式为非持久化或者设置ActiveMQ ConnectionFactory的**useAsyncSend** 属性为true

|  |
| --- |
| producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);  或者  connectionFactory.setUseAsyncSend(true); |

1. 对消费者设置Pre-fetch Sizes

|  |
| --- |
| ActiveMQPrefetchPolicy prefetchPolicy = new ActiveMQPrefetchPolicy();  prefetchPolicy.setQueuePrefetch(10);  connectionFactory.setPrefetchPolicy(prefetchPolicy); |

1. 优化应答属性开启，批量应答

在连接工厂里设置optimizeAcknowledge=true

1. 基于文件的持久性

我们有一个基于文件的持久性存储，可用于增加持久性消息的吞吐量

