讨论课-复用解决方案

我们项目通讯采用的是 ActiveMQ，下面我们以 ActiveMQ 为基础分析几个问题的解决方案。

长连接心跳机制

ActiveMQ 支持多种通讯协议 TCP/UDP 等，我们选取最常用的 TCP 来分析 activeMQ 的通讯机制。

ActiveMQ 的核心通讯机制包括三个部分：建立连接、关闭连接、心跳。

一、建立连接

1．ActiveMQ 初始化时，通过 TcpTransportServer 类根据配置打开 TCP 侦听端口，客户通过该端口发起建立链接的动作。

2． 把 accept 的 Socket 放入阻塞队列中。

3． 另外一个线程 Socket handler 阻塞着等待队列中是否有新的 Socket，如果有则取出来。

4． 生成一个 TransportConnection 的实例。TransportConnection 类的主要作用是处理链路的状态信息，并实现 CommandVisitor 接口来完成各类消息的处理。

5． TransportConnection 会使用一个由多个 TransportFilter 实例组成的消息处理链条，负责对接收到的各类消息进行处理并发送相应的应答。这个链条的典型组成顺序：MutexTransport->WireFormatNegotiator->InactivityMonitor->TcpTransport。在这条链条中最后的一环就是 TcpTransport 类，它是实际和 Client 获取和发送数据的地方，该类的重要方法有 run()和 oneway()，一个负责读取，一个负责发送。

6． 建链完成，可以进行通讯操作。

二、关闭连接

activeMQ 发现 TCP 链接的关闭，最关键的代码在 TcpBufferedInputStream 类中

```
int n = in.read(buffer, position, buffer.length - position);
```

三、心跳

为了更好的维护 TCP 链路的使用，activeMQ 采用了心跳机制作为判断双方链路的健康情况。activeMQ 使用的是双向心跳，也就是 activeMQ 的 Broker 和 Client 双方都进行相互心跳，但不管是 Broker 或 Client 心跳的具体处理情况是完全一样的，都在 InactivityMonitor 类中实现，下面具体介绍。

心跳会产生两个线程“InactivityMonitor ReadCheck”和“InactivityMonitor WriteCheck”，它们都是 Timer 类型，都会隔一段固定时间被调用一次。ReadCheck 线程主要调用的方法是 readCheck()，当在等待时间内，有消息接收到，则该方法会返回 true。WriteCheck 线程主要调用的方法是 writeCheck()，当 WriteCheck 线程休眠时，有任何数据发送成功，则该线程被唤醒后，不用通过 TCP 向对方真的发送心跳消息，这样可以从一定程度上减少网络传输的数据量。

消息遗漏和重复

我们采用 ACK 来解决消息的遗漏和重复问题

ACK (Acknowledgement)，即确认字符，在数据通信中，接收站发给发送站的一种传输类控制字符。表示发来的数据已确认接收无误。

在 TCP/IP 协议中，如果接收方成功的接收到数据，那么会回复一个 ACK 数据。

通常 ACK 信号有自己固定的格式,长度大小,由接收方回复给发送方。其格式取决于采取的网络协议。当发送方接收到 ACK 信号时，就可以发送下一个数据。如果发送方没有收到信号，那么发送方可能会重发当前的数据包，也可能停止传送数据。具体情况取决于所采用的网络协议。

在 java 的 JMS API 中约定了 Client 端可以使用四种 ACK 模式,在 javax.jms.Session 接口中:

- *AUTO_ACKNOWLEDGE* = 1　　自动确认
- *CLIENT_ACKNOWLEDGE* = 2　　客户端手动确认
- *DUPS_OK_ACKNOWLEDGE* = 3　　自动批量确认
- *SESSION_TRANSACTED* = 0　　事务提交并确认

此外 ActiveMQ 补充了一个自定义的 ACK 模式：

- *INDIVIDUAL_ACKNOWLEDGE* = 4　　单条消息确认

ACK 模式描述了 Consumer 与 broker 确认消息的方式(时机),比如当消息被 Consumer 接收之后,Consumer 将在何时确认消息。对于 broker 而言，只有接收到 ACK 指令,才会认为消息被正确的接收或者处理成功了,通过 ACK，可以在 consumer（/producer）与 Broker 之间建立一种简单的“担保”机制.

Client 端指定了 ACK 模式,但是在 Client 与 broker 在交换 ACK 指令的时候,还需要告知 ACK_TYPE,ACK_TYPE 表示此确认指令的类型，不同的 ACK_TYPE 将传递着消息的状态，broker 可以根据不同的 ACK_TYPE 对消息进行不同的操作。

我们以 AUTO_ACKNOWLEDGE 为例详解：

AUTO_ACKNOWLEDGE : 自动确认,这就意味着消息的确认时机将有 consumer 择机确认."择机确认"似乎充满了不确定性,这也意味着,开发者必须明确知道"择机确认"的具体时机,否则将有可能导致消息的丢失,或者消息的重复接收.

1>对于 consumer 而言，optimizeAcknowledge 属性只会在 AUTO_ACK 模式下有效。

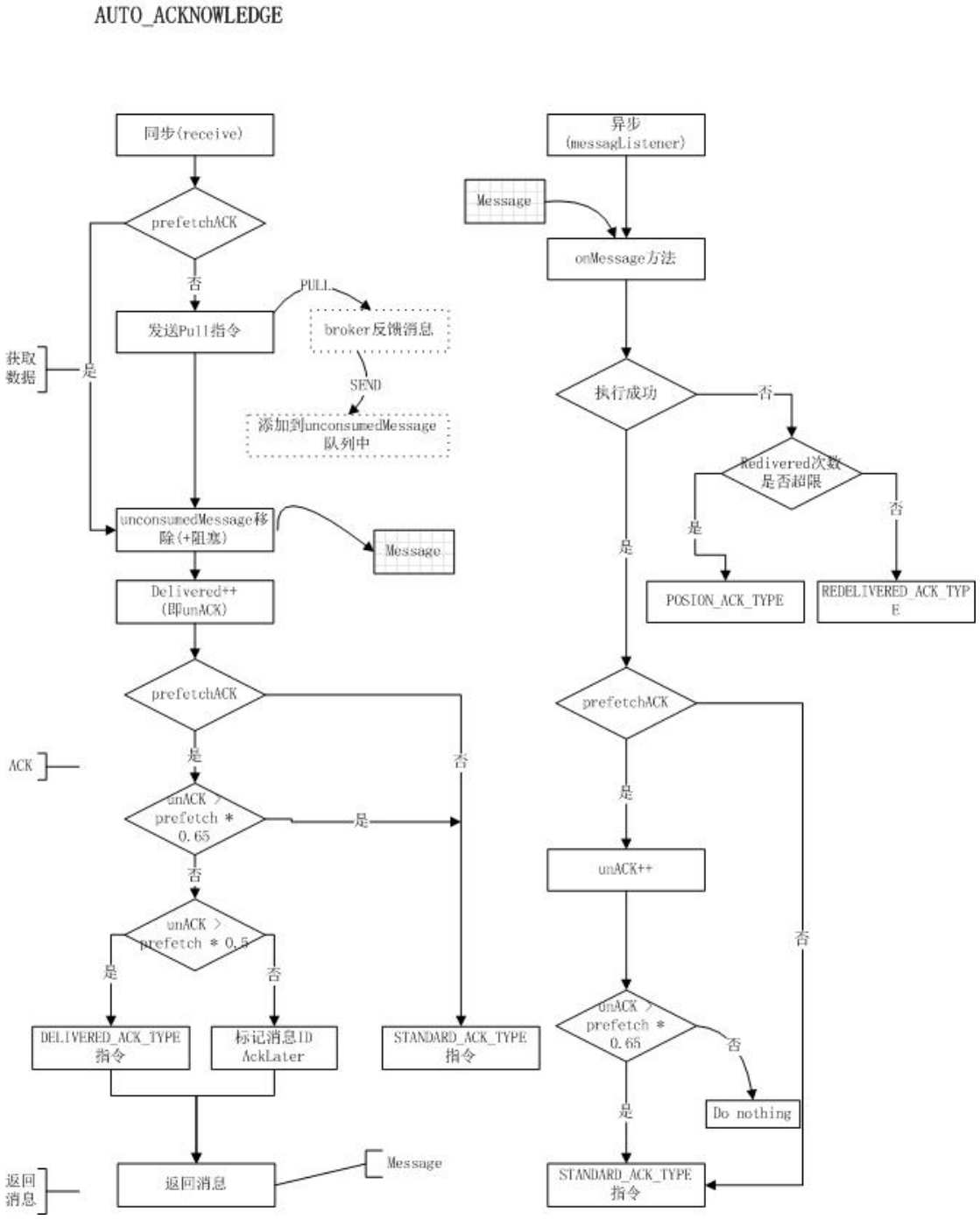
2>其中 DUPS_ACKNOWLEGE 也是一种潜在的 AUTO_ACK,只是确认消息的条数和时间上有所不同。

3>在“同步”(receive)方法返回 message 之前,会检测 optimizeACK 选项是否开启，如果没有开启，此单条消息将立即确认，所以在这种情况下，message 返回之后，如果开发者在处理 message 过程中出现异常，会导致此消息也不会 redelivery,即"潜在的消息丢失"；如果开启了 optimizeACK，则会在 unAck 数量达到 prefetch * 0.65 时确认，当然我们可以指定 prefetchSize = 1 来实现逐条消息确认。

4>在"异步"(messageListener)方式中,将会首先调用 listener.onMessage(message),此后再 ACK,如果 onMessage 方法异常,将导致 client 端补充发送一个 ACK_TYPE 为 REDELIVERED_ACK_TYPE 确认指令；如果 onMessage 方法正常,消息将会正常确认(STANDARD_ACK_TYPE)。此外需要注意，消息的重发次数是有限制的，每条消息中都会包含“redeliveryCounter”计数器，用来表示此消息已经被重发的次数，如果重发次数达到阀值，将会导致发送一个 ACK_TYPE 为 POSION_ACK_TYPE 确认指令,这就导致 broker 端认为此消息无法消费,此消息将会被删除或者迁移到"dead letter"通道中.

因此当我们使用 messageListener 方式消费消息时，通常建议在 onMessage 方法中使用 try-catch,这样可以在处理消息出错时记录一些信息，而不是让

consumer 不断去重发消息；如果你没有使用 try-catch,就有可能会因为异常而导致消息重复接收的问题,需要注意你的 onMessage 方法中逻辑是否能够兼容对重复消息的判断。



消息压缩

许多信息资料都或多或少的包含一些多余的数据。通常会导致在客户端与服务器之间，应用程序与计算机之间极大的数据传输量。最常见的解决数据存储和信息传送的方法是安装额外的存储设备和扩展现有的通讯能力。这样做是可以的，但无疑会增加组织的运作成本。一种有效的解决数据存储与信息传输的方法是通过更有效率的代码来存储数据。

我们可以利用 JAVA API 实现对数据的压缩与解压缩，其基本原理很简单，例如：

JJJJJJAAAAVVVVAAAAAA 这个字符串可以用更简洁的方式来编码，那就是通过替换每一个重复的字符串为单个的实例字符加上记录重复次数的数字来表示,上面的字符串可以被编码为下面的形式：

6J4A4V6A 在这里，"6J"意味着 6 个字符 J，"4A"意味着 4 个字符 A，以此类推。这种字符串压缩方式称为"行程长度编码"方式，简称 RLE。