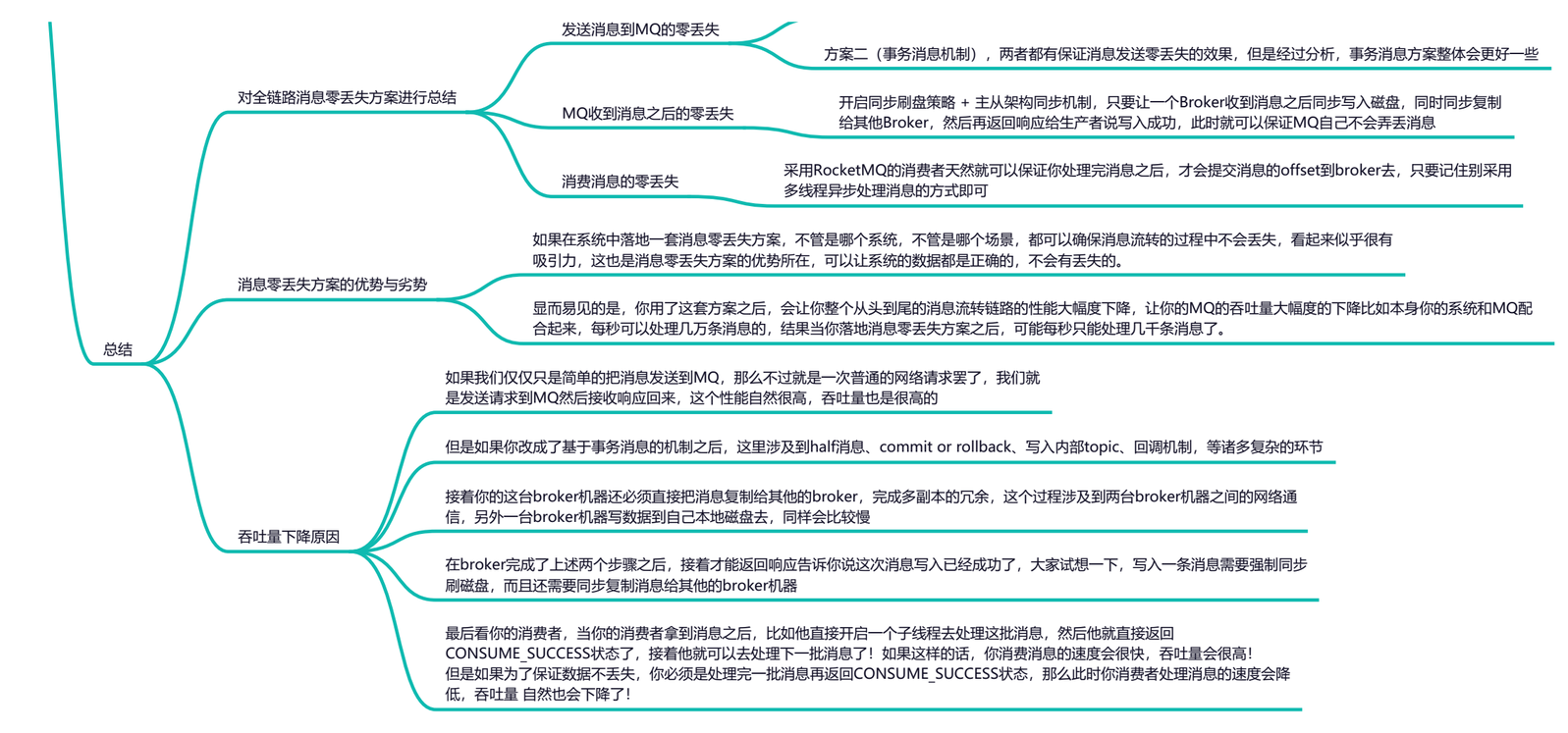
学习
RocketMQ
JVM


总结
对全链路消息零丢失方案进行总结
发送消息到MQ的零丢失
方案二（事务消息机制），两者都有保证消息发送零丢失的效果，但是经过分析，事务消息方案整体会更好一些
MQ收到消息之后的零丢失
开启同步刷盘策略 + 主从架构同步机制，只要让一个Broker收到消息之后同步写入磁盘，同时同步复制给其他Broker，然后再返回响应给生产者说写入成功，此时就可以保证MQ自己不会弄丢消息
消费消息的零丢失
采用RocketMQ的消费者天然就可以保证你处理完消息之后，才会提交消息的offset到broker去，只要记住别采用多线程异步处理消息的方式即可
消息零丢失方案的优势与劣势
如果在系统中落地一套消息零丢失方案，不管是哪个系统，不管是哪个场景，都可以确保消息流转的过程中不会丢失，看起来似乎很有吸引力，这也是消息零丢失方案的优势所在，可以让系统的数据都是正确的，不会有丢失的。
显而易见的是，你用了这套方案之后，会让你整个从头到尾的消息流转链路的性能大幅度下降，让你的MQ的吞吐量大幅度的下降比如本身你的系统和MQ配合起来，每秒可以处理几万条消息的，结果当你落地消息零丢失方案之后，可能每秒只能处理几千条消息了。
吞吐量下降原因
如果我们仅仅只是简单的把消息发送到MQ，那么不过就是一次普通的网络请求罢了，我们就是发送请求到MQ然后接收响应回来，这个性能自然很高，吞吐量也是很高的
但是如果你改成了基于事务消息的机制之后，这里涉及到half消息、commit or rollback、写入内部topic、回调机制，等诸多复杂的环节
接着你的这台broker机器还必须直接把消息复制给其他的broker，完成多副本的冗余，这个过程涉及到两台broker机器之间的网络通信，另外一台broker机器写数据到自己本地磁盘去，同样会比较慢
在broker完成了上述两个步骤之后，接着才能返回响应告诉你说这次消息写入已经成功了，大家试想一下，写入一条消息需要强制同步刷磁盘，而且还需要同步复制消息给其他的broker机器
最后看你的消费者，当你的消费者拿到消息之后，比如他直接开启一个子线程去处理这批消息，然后他就直接返回CONSUME_SUCCESS状态了，接着他就可以去处理下一批消息了！如果这样的话，你消费消息的速度会很快，吞吐量会很高！但是如果为了保证数据不丢失，你必须是处理完一批消息再返回CONSUME_SUCCESS状态，那么此时你消费者处理消息的速度会降低，吞吐量 自然也会下降了！