28 | 答疑解惑（二）：我的100元哪儿去了？

2019-09-28 李玥

你好，我是李玥。

今天这节课，是我们的“消息队列高手课第二阶段进阶篇的最后一节课，照例，我们在每一阶段的最后，安排一节课进行热点问题的答疑，针对同学们遇到的一些共同的问题，统一来进行详细的解答。

1. 我的100元哪儿去了？聊聊并发调用情况下的幂等性

在期中测试中，有这样一道题。

如果可以保证以下这些操作的原子性，哪些操作在并发调用的情况下具备幂等性？

- A. f(n, a)：给账户n转入a元
- B. f(n, a)：将账户n的余额更新为a元
- C. f(n, b, a)：如果账户n当前的余额为b元，那就将账户的余额更新为n元
- D. f(n, v, a)：如果账户n当前的流水号等于v，那么给账户的余额加a元，并将流水号加一

这道题的正确答案是D。很多同学都留言提问，选项B中，将账户n的余额更新为a元，这个操作不具备幂等性吗？

如果单单只是考虑这个操作，执行一次和执行多次，对系统的影响是一样的，账户n的余额都是a元。所以，这个操作确实是幂等的。但请你注意审题，我们的题目中说的是：“哪些操作**在并**

发调用的情况下具备幂等性？”在并发调用的情况下，我们再来看一下B这个选项的操作是否还具备幂等性。

假设，账户余额100元，依次执行2次转账：

1. 将账户余额设为200元；
2. 将账户余额设为300元；

经过两次转账后，账户余额应该是300元。

再次注意，我们的题目中说的是在并发调用的情况下。

按照时间顺序，就有可能会出现下面这种情况：

1. t0时刻客户端发送请求：将账户余额设为200元。
2. t1时刻服务端收到请求，账户余额由100元变为200元，然后服务端发出给客户端操作成功的响应，但是这个响应在网络传输过程中丢失了。
3. t2时刻客户端发送请求：将账户余额设为300元。
4. t3时刻服务端收到请求，账户余额由200元变为300元，然后服务端发出给客户端操作成功的响应。
5. t4时刻客户端：收到“将账户余额设为300元”这个请求的成功响应，本次调用成功。
6. t5时刻客户端由于没收到“将账户余额设为200元”这个请求的成功响应，重新发送请求：将账户余额设为200元。
7. t6时刻服务端收到请求，账户余额由300元变为200元，然后服务端给客户端发出操作成功的响应。
8. t7时刻客户端收到响应，本次重试调用成功。

结果，账户余额错误地变成了200元。

同学，请把我的100块钱还给我！通过这个题，我们可以总结出来，**一个操作是否幂等，还跟调用顺序有关系**，在线性调用情况下，具备幂等性的操作，在并发调用时，就不一定具备幂等性了。如果你在设计系统的时候，没有注意到这个细节，那系统就有可能出现我们上面这个例子中的错误，在生产系中，这是非常危险的。

2. Kafka和RocketMQ如何通过选举来产生新的Leader？

在《22 | Kafka和RocketMQ的消息复制实现的差异点在哪？》这节课中，我给你讲了这两个消息队列是如何通过复制来保证数据一致性的。当Broker节点发生故障时，它们都是通过选举机制，来选出新的Leader来继续提供服务。当时限于篇幅，我们并没有深入进去来讲选举的实现原理。那Kafka和RocketMQ（Dledger）都是怎么来实现的选举呢？

先来说Kafka的选举，因为Kafka的选举实现比较简单。严格地说，Kafka分区的Leader并不是选

举出来的，而是Controller指定的。Kafka使用ZooKeeper来监控每个分区的多个副本，如果发现某个分区的主节点宕机了，Controller会收到ZooKeeper的通知，这个时候，Controller会从ISR节点中选择一个节点，指定为新的Leader。

在Kafka中Controller本身也是通过ZooKeeper选举产生的。接下来我要讲的，Kafka Controller利用ZooKeeper选举的方法，你一定要记住并学会，因为这种方法非常简单实用，并且适用性非常广泛，在设计很多分布式系统中都可以用到。

这种选举方法严格来说也不是真正的“选举”，而是一种抢占模式。实现也很简单，每个Broker在启动后，都会尝试在ZooKeeper中创建同一个临时节点：/controller，并把自身的信息写入到这个节点中。由于ZooKeeper它是一个**可以保证数据一致性的分布式存储**，所以，集群中只会有一个Broker抢到这个临时节点，那它就是Leader节点。其他没抢到Leader的节点，会Watch这个临时节点，如果当前的Leader节点宕机，所有其他节点都会收到通知，它们会开始新一轮的抢Leader游戏。

这就好比有个玉玺，也就是皇帝用的那个上面雕着龙纹的大印章，谁都可以抢这个玉玺，谁抢到谁做皇帝，其他没抢到的人也不甘心，时刻盯着这个玉玺，一旦现在这个皇帝驾崩了，所有人一哄而上，再“抢”出一个新皇帝。这个算法虽然不怎么优雅，但胜在简单直接，并且快速公平，是非常不错的选举方法。

但是这个算法它依赖一个“玉玺”，也就是一个**可以保证数据一致性的分布式存储**，这个分布式存储不一定非得是ZooKeeper，可以是Redis，可以是MySQL，也可以是HDFS，只要是可以保证数据一致性的分布式存储，都可以充当这个“玉玺”，所以这个选举方法的适用场景也是非常广泛的。

再来说RocketMQ/Dledger的选举，在Dledger中的Leader真的是通过投票选举出来的，所以它不需要借助于任何外部的系统，仅靠集群的节点间投票来达成一致，选举出Leader。一般这种自我选举的算法，为了保证数据一致性、避免集群分裂，算法设计的都非常非常复杂，我们不太可能自己来实现这样一个选举算法，所以我在这里不展开讲。Dledger采用的是Raft一致性算法，感兴趣的同学可以读一下这篇经典的论文。

像Raft这种自我选举的算法，相比于上面介绍的抢占式选举，优点是不需要借助外部系统，完全可以实现自我选举。缺点也非常明显，就是算法实在是太复杂了，非常难实现。并且，往往集群中的节点要通过多轮投票才能达成一致，这个选举过程是比较慢的，一次选举耗时几秒甚至几十秒都有可能。

我们日常在设计一些分布式的业务系统时，如果需要选举Leader，还是采用Kafka的这种“抢玉玺”的方法更加简单实用。

3. 为什么说Pulsar存储计算分离的架构是未来消息队列的发展方向？

在上节课《27 | Pulsar的存储计算分离设计：全新的消息队列设计思路》中，我给你留的思考题是：为什么除了Pulsar以外，大多数的消息队列都没有采用存储计算分离的设计呢？这个问题其实是一个发散性的问题，并没有什么标准答案。因为，本来架构设计就是在权衡各种利弊，做出取舍和选择，并没有绝对的对错之分。

很多同学在课后的留言中，都已经给出了自己的思路和想法，而且有些同学的想法和我个人的观点不谋而合。在这里我也和你分享一下我对这个问题的理解和看法。

早期的消息队列，主要被用来在系统之间异步交换数据，大部分消息队列的存储能力都比较弱，不支持消息持久化，不提倡在消息队列中堆积大量的消息，这个时期的消息队列，本质上是一个数据的管道。

现代的消息队列，功能上看似没有太多变化，依然是收发消息，但是用途更加广泛，数据被持久化到磁盘中，大多数消息队列具备了强大的消息堆积能力，只要磁盘空间足够，可以存储无限量的消息，而且不会影响生产和消费的性能。这些消息队列，本质上已经演变成为分布式的存储系统。

理解了这一点，你就会明白，为什么大部分消息队列产品，都不使用存储计算分离的设计。为一个“分布式存储系统”做存储计算分离，计算节点就没什么业务逻辑需要计算的了。而且，消息队列又不像其他的业务系统，可以直接使用一些成熟的分布式存储系统来存储消息，因为性能达不到要求。分离后的存储节点承担了之前绝大部分功能，并且增加了系统的复杂度，还降低了性能，显然是不划算的。

那为什么Pulsar还要采用这种存储和计算分离的设计呢？我们还是需要用发展的眼光看问题。我在上节课说过，Pulsar的这种架构，很可能代表了未来消息队列的发展方向。为什么这么说呢？你可以看一下现在各大消息队列的Roadmap（发展路线图），Kafka在做Kafka Streams，Pulsar在做Pulsar Functions，其实大家都在不约而同的做同一件事儿，就是流计算。

原因是什么呢？现有的流计算平台，包括Storm、Flink和Spark，它们的节点都是无状态的纯计算节点，是没有数据存储能力的。所以，现在的流计算平台，它很难做大量数据的聚合，并且在数据可靠性保证、数据一致性、故障恢复等方面，也做得不太好。

而消息队列正好相反，它很好地保证了数据的可靠性、一致性，但是Broker只具备存储能力，没有计算的功能，数据流进去什么样，流出来还是什么样。同样是处理实时数据流的系统，一个只能计算不能存储，一个只能存储不能计算，那未来如果出现一个新的系统，既能计算也能存储，如果还能有不错的性能，是不是就会把现在的消息队列和流计算平台都给替代了？这是很有可能的。

对于一个“带计算功能的消息队列”来说，采用存储计算分离的设计，计算节点负责流计算，存储节点负责存储消息，这个设计就非常和谐了。

到这里，我们课程的第二个模块-进阶篇，也就全部结束了。进阶篇的中讲解知识有一定的难度，特别是后半部分的几节源码分析课，从评论区同学们的留言中，我也能感受到，有些同学学习起来会有些吃力。

我给同学们的建议是，除了上课时听音频和读文稿之外，课后还要自己去把源代码下载下来，每一个流程从头到尾读一遍源码，最好是打开单步调试模式，一步一步地跟踪一下执行过程。读完源码之后，还要把类图、流程图或者时序图画出来，只有这样才能真正理解实现过程。

从下节课开始，我们的课程就进入最后一个模块：案例篇。在这个模块中，我会带你一起动手来写代码，运用我们在课程中所学的知识，来做一些实践的案例。首先我会带你一起做一个消息队列和流计算的案例，你可以来体会一下现在的流计算平台它是什么样的。然后，我们还会用进阶篇中所学到的知识，来一起实现一个类似Dubbo的RPC框架。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言

leslie 2

Pulsar的计算性是真正的减负：就是觉得如果只是单纯的其它MQ那么做有点Cache的意思了，而且这个Cache还和No sql用的都是内存缓存。

刘文浩老师当时提的哪个我当时觉得最不一样的就在于提及的是CPU缓存，老师说的Pulsar的计算性是其它所不具备的不一样的特性-故而上堂课的留言我会提及既然有了Cpu干嘛要有GPU

：任何工具都有两面性吧，如果不充分发挥特性异中发挥其特性，是很难在未来走下去的。今年的十一长假基本上要和老师的课程一起度过了：一路走来学习的很辛苦，但是收获的喜悦替代了身体的辛苦；期待老师的案例篇吧：实战中度过十一长假，期待老师十一长假的更新；谢谢老师的分享，愿老师节日快乐。

2019-09-28

Geek_72a3d3 0

老师，spring里面集成的kafka客户端和官方kafka客户端您更推荐哪个，spring里面集成的kafka客户端您认为怎么样？常用么

2019-10-11

作者回复

你是说spring-kafka么？这个只是在Spring中对kafka-clients做了一些封装而已，使用的就是kafka的官方客户端。

2019-10-11

陶源 0

并发调用情况的第6条是不是应该为 “没收到‘将账户余额设为200元’这个请求的成功响应”

2019-10-10

作者回复

这个地方确实是笔误，我联系编辑小姐姐修改一下。感谢。

2019-10-10

jack 0

老师，之前您介绍kafka在zookeeper中的节点，似乎没有/controller这个临时节点啊？

2019-10-02

作者回复

是的，之前那张图中只有主要的节点，并不是Kafka在ZK中的全部节点

2019-10-04

Jen 0

期待案例篇 老师节日快乐

2019-09-30

Better me 0

提前祝老师节日快乐啊，假期有时间必须把课程相关源码过一下

2019-09-30

A9 0

请问老师后面会再讲到序列化相关的知识点吗

2019-09-29

作者回复

在实践篇，我们还有有半节课专门来讲序列化的最佳实践。

2019-09-30