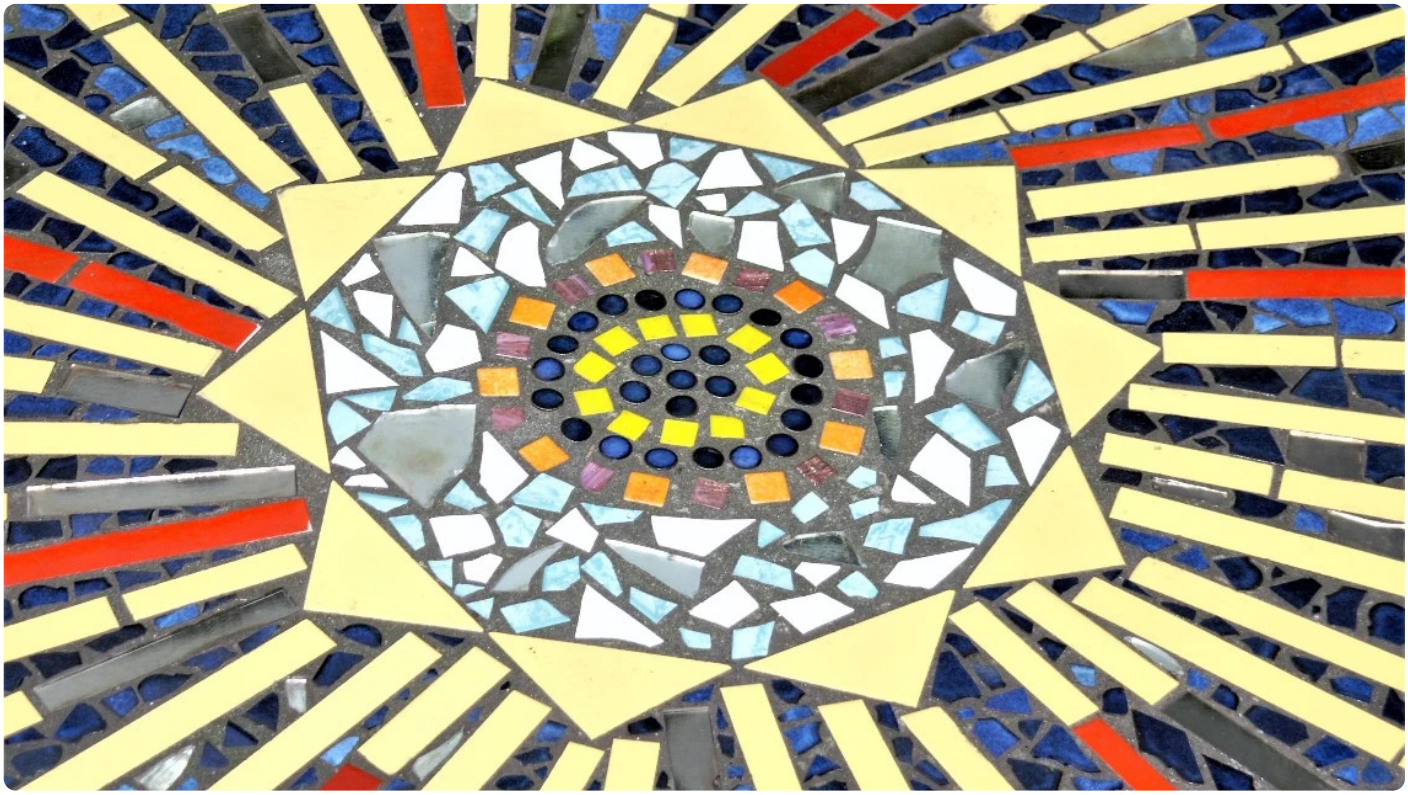
33 | 知识串联：以购买火车票的流程串联分布式核心技术

2019-12-13 聂鹏程 来自北京

《分布式技术原理与算法解析》



你好，我是聂鹏程。今天，我来继续带你打卡分布式核心技术。

还记得在专栏之初，我和你分享的“分布式四纵四横知识体系”吗？截止到目前，我已经带你学习了四横和三纵，包括分布式计算、分布式存储与管理、分布式通信、分布式资源池化、分布式协同、分布式调度和分布式追踪与高可用的关键技术（由于分布式追踪、分布式部署虽属于支撑技术，但并不会影响业务的构成，因此我没有在专栏中展开）。

但学以致用才是最终目的，所以在接下来的模块中，我将通过两篇总结性质的文章，为你串联起前面讲到的核心知识点，看看它们在业务中是如何应用的。

今天，我就先以购买火车票的流程，带你串联下整个专栏涉及的分布式核心技术吧。

首先，为方便你理解，并抓住其中涉及的核心技术，我对购买火车票的流程做了一个简化，大致划分为三大核心步骤：

首先，铁路局向购票系统发布火车票；

然后，用户通过系统查询火车票，找到需要的火车票后购买；

最后，购票系统给用户响应，完成购票。

这个流程看似简单，但涉及了我们之前讲过的很多知识。

这里，我有个小建议，在学习后面的内容前，你可以先自己思考下这个过程涉及了哪些知识点，然后再与我接下来的讲述进行对比，以验证自己对之前内容的掌握程度。这样一来，你可以加深对已掌握知识的理解深度，也可以查漏补缺进而有针对性地复习其他内容。

那么接下来，我就主要分为三部分进行讲解：铁路局发布火车票、用户查询火车票，以及用户购买火车票。

铁路局发布火车票

铁路局发布火车票的过程，主要涉及分布式数据存储这一站的知识，包括存储系统三要素、数据分布和数据复制等技术。

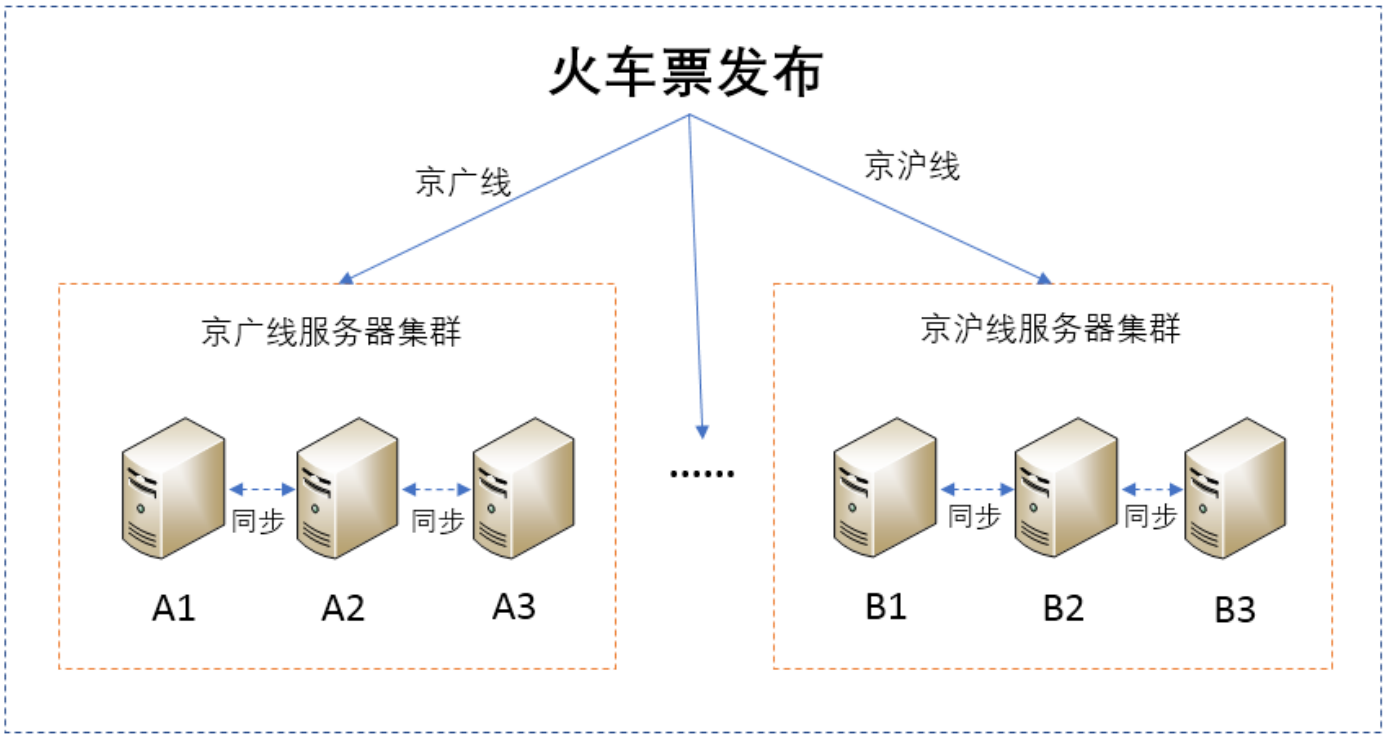
铁路局向购票系统发布火车票的过程，主要是将火车票信息发送到服务器集群进行存储，需要用到存储系统三要素的相关技术。其中，铁路局就是存储系统三要素中的“顾客”，火车票存储到具体哪个服务器需要构建数据索引，也就是我们说的三要素中的“导购”，而存储数据的服务器就是三要素中的“货架”。

由于涉及多个服务器存储火车票信息，不可避免地需要考虑服务器之间数据存储的均衡，以及快速确定火车票信息存储位置以方便后续查询和购买火车票。因此，这里需要用到分布式存储中的数据分布技术。

铁路局按照火车线路将火车票发布到不同的服务器上，即在数据分片技术中提到的按照数据范围进行分片。比如，京广线的车票信息存储在京广线服务器集群、京沪线的车票信息存储在京沪线服务器集群等。

除此之外，为了保证可靠性，也就是当一台服务器故障后，该服务器存储的火车票信息可以恢复或不丢失，通常会进行数据备份。也就是说，同一份数据可能会有多台服务器一起存储，比如京广线的火车票数据存储到服务器 A1、A2 和 A3 上，京沪线的火车票数据存储在服务器 B1、B2 和 B3 上。

而数据备份，用到的就是分布式存储中的数据复制技术。由于同一份数据被多台服务器存储，自然就需要保证数据的一致性。关于数据一致性，你可以参考CAP 理论这篇文章。



接下来，我们再看看用户查询火车票过程中涉及了哪些相关技术吧。

用户查询火车票

对于用户查询火车票来说，大致过程是用户向服务器发起查询请求，服务器根据用户请求，通过数据索引，也就是导购技术，定位到火车票信息存储的位置，然后获取数据返回给用户。

从这个流程可以看出，**服务器接收用户查询请求是第一步**。

正常情况下，用户并发请求量比较小，很少会出现服务器能力有限导致系统崩溃的情况。但，遇到节假日或春节，用户请求量通常会非常大，这时如果不采取一定策略的话，大概率会因为

服务器能力受限导致系统崩溃。

而这里的策略，通常就是保证分布式高可靠的负载均衡和流量控制。比如，每年春运，火车票发布的瞬间，就有大量的用户抢票，如果购票系统后台不使用负载均衡和流量控制的话，服务器一下子就被击垮了。

除此之外，服务器还不可避免地会出现一些小故障，比如磁盘损坏、网络故障等问题。如何检测服务器故障，以及如何进行故障恢复，就需要用到分布式高可用的故障隔离与故障恢复的相关技术了。

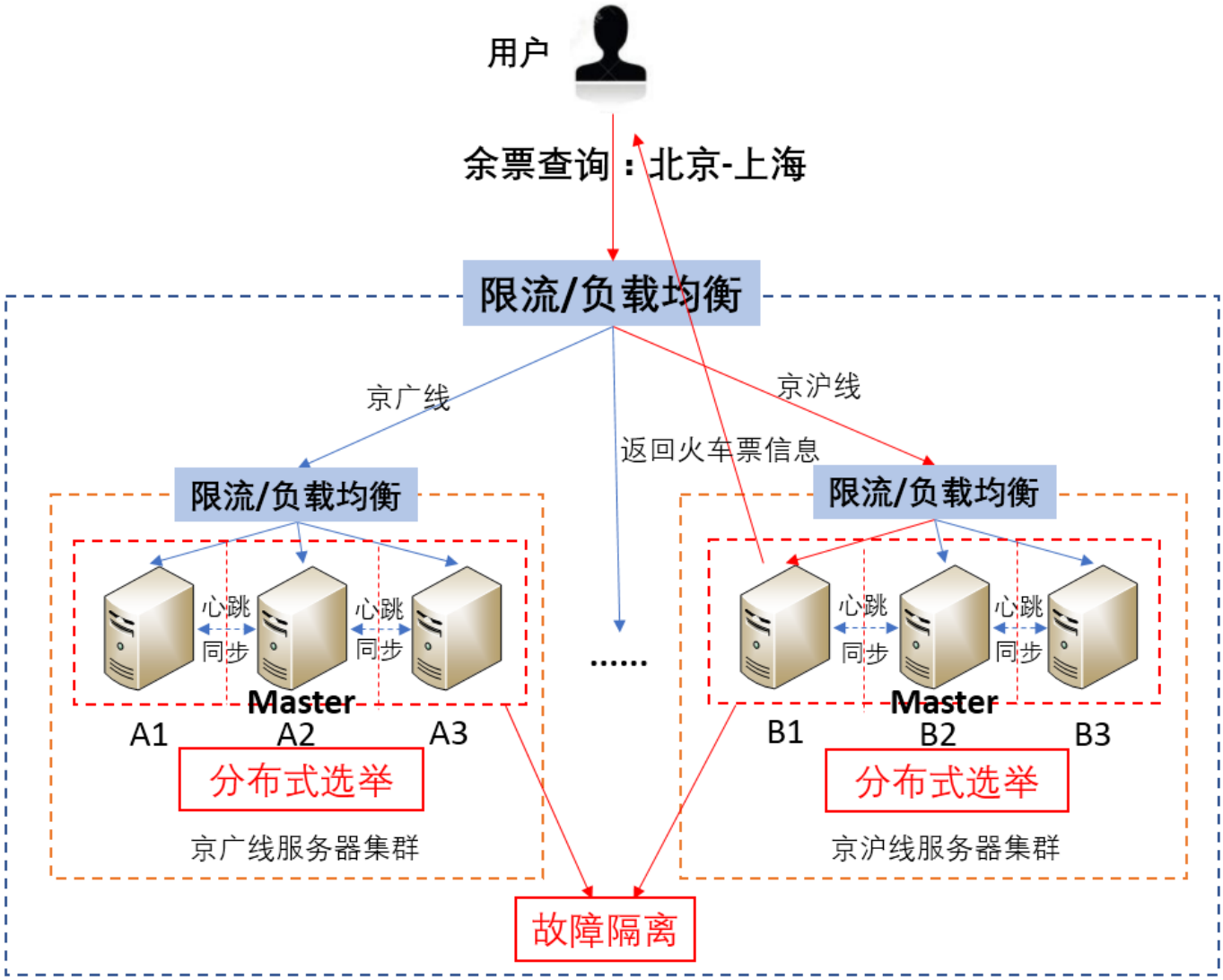
接收用户请求后，接下来需要将请求转发至相应的服务器集群，然后再从中选择某一台服务器处理用户请求，也就是获取数据并返回给用户。本质上，这就是在进行数据索引，设计分布式数据存储中的数据分布方式的相关技术。

以用户查询从北京到上海的火车票信息为例，查询流程如下：

首先，根据查询条件，系统将请求转发至存储京沪线火车票信息的服务器集群中；

然后，服务器集群再使用一次负载均衡，比如使用轮询算法， 将请求转发至某一台服务器；

最后，这台服务器将火车票的车次、余票等信息返回给用户。



在这个过程中，还可以使用限流算法，比如漏桶、令牌桶等策略来限制用户流量，保证系统高可用。

除此之外，在存储火车票信息的服务器中，各个服务器的服务单独运行，也就相当于做了一定的故障隔离，比如图中的服务器 A1~A3、B1~B3。

当然，这里**还需要注意一个问题**。在上面的过程中，我一直提到的是服务器集群。既然是集群，就会涉及集群架构、分布式选主等策略。

以集中式架构 Master/Slave 为例，服务器集群中会通过分布式选举算法选出一个 Master，Master 和其他服务器节点之间会维持心跳，并通过心跳来感知服务器节点的存活状态。比如，京广线服务器集群选出的 Master 节点为 A2，A2 与 A1 和 A3 之间一直维持心跳。

具体的集群架构原理，你可以参考“第二站：分布式资源管理与负载调度”；而分布式选主的原理，你可以再回顾“第一站：分布式协同与同步”的相关内容。

当集群中 Master 节点故障后，会从其他节点中重新选举出一个 Master 节点，继续为用户提供服务，也就是备升主，以保证服务的可用性。这里，备升主就是一种故障恢复策略。

最后，我们再来分析下用户购买火车票的过程。

用户购买火车票

用户购买火车票的过程与用户查询火车票的过程非常相似，**唯一不同的是会造成数据库的变化。**换句话说，用户查询火车票是读请求，而购买火车票相当于一个写请求。那么，写请求又会造成什么新问题呢？

写请求与读请求的区别是，写请求会造成数据的变化，因此相对于查询火车票，购买火车票的过程涉及的技术问题会多一些，但多出的无非就是数据的一致性问题。谈到数据的一致性，我们就会想到CAP 理论、数据复制技术、分布式事务、分布式锁等。其实，不仅仅是购买火车票，任何一个简单的购买操作或写操作中，都会涉及这些分布式知识。

本质上讲，每次购买火车票的操作，就是一个分布式事务，要么执行成功要么执行失败。

当用户购买了火车票时，该火车票对应时间的车次余票数量必须相应减 1，如果减少时发现原先票数就已经为 0 了，此时就应该提醒用户购买火车票失败，余票为 0；同样的，如果票数不为 0，则票数应该相应减 1，并提示用户购票成功。

不难看出，购买火车票会改变火车票数据，也不可避免地会存在多个用户同时购买相同路线、相同车次（比如京沪线的 T12）的场景。也就是说，这个购买过程存在多个进程同时访问共享资源的问题，因此还要用到分布式锁的相关技术。

另外，**用户购买火车票的过程还会涉及用户体验、数据一致性和网络故障等问题，因此还涉及 C、A、P 策略的选择问题。**

在铁路局发布火车票的流程中，为了保证可靠性，同一数据通常会备份到多个服务器上。当用户购买火车票导致火车票数据改变时，主节点上的数据必须与备节点上的数据保持一致，以防止主节点故障后，备升主，但数据不一致导致业务出错的情况。

比如，用户 A 购买 2019 年 10 月 12 日北京到上海的 T12 的火车票，已购买成功，座位号为 3 车厢 23B。假设主节点和备节点之间数据不一致，主节点上已经减去该火车票，但未在备节点上减去。此时，若主节点故障，备节点升主，用户 B 此时申请购买相同火车票，系统将 3 车厢 23B 火车票又卖给了用户 B。等到乘车时，用户 A 和 B 就难免“打架”了。

当然，通常因为网络故障或节点故障等原因导致主节点不能正常工作，才会发生备升主，而备升主其实就是故障恢复策略，而一致性问题涉及的是“第五站：分布式数据存储”的相关技术。

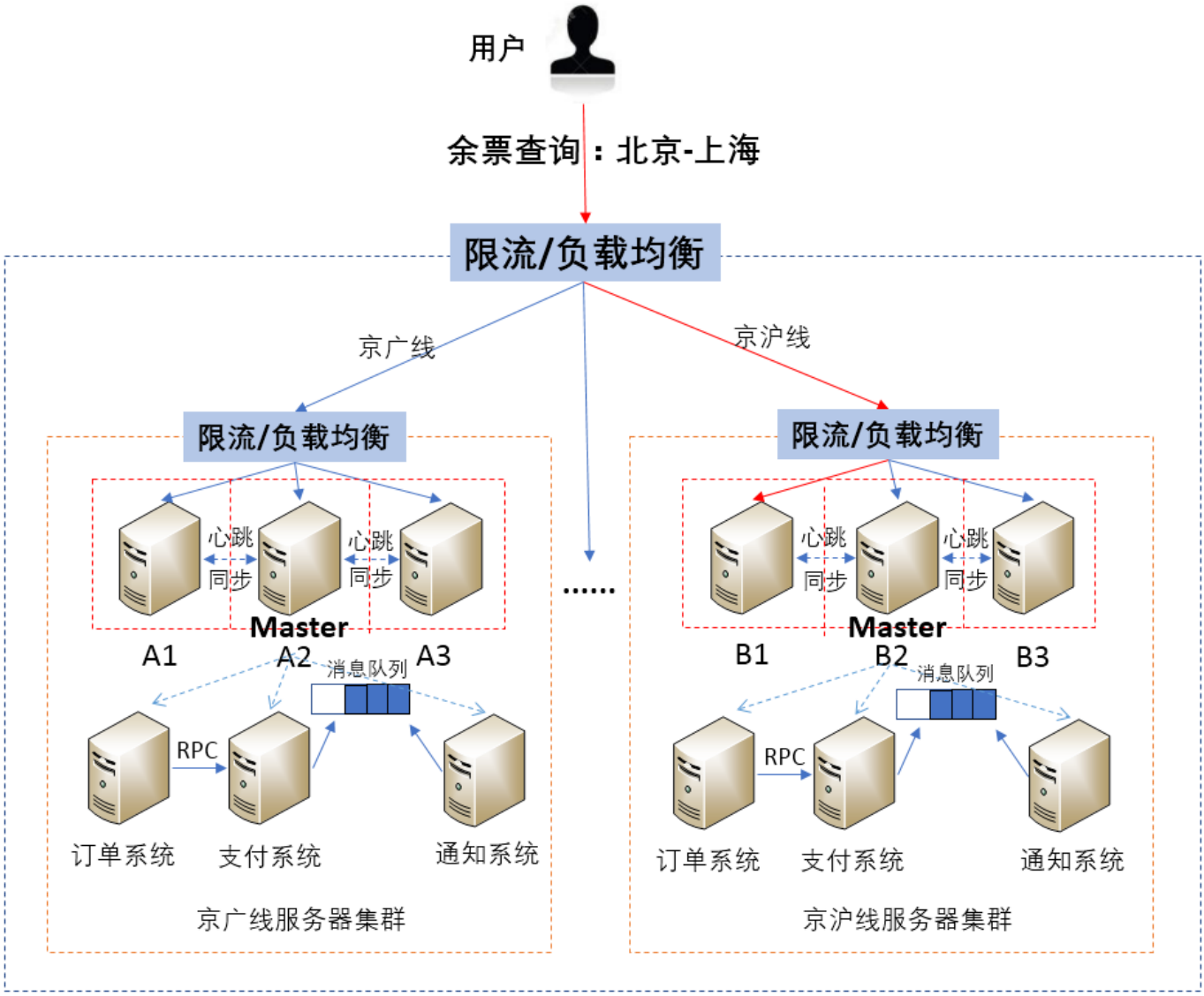
同时，购买火车票的场景需要快速响应用户，以保证用户体验。因此，通常优先保证系统的可用性，稍微降低对数据一致性的要求，但也必须保证最终一致性。这就是我们平常遇到的，查询火车票时还有余票，但下单后却提示余票为 0，无法购买。

导致这个结果的原因是，下单前你访问的数据库中，数据还未同步，显示有余票；而下单后，数据实现同步了，发现余票数量已经为 0，因此提示你无法购买该火车票。

实现上述策略的方法，通常会采用半同步复制技术，即将修改后的数据同步到多个备数据库中的某一个或几个后立即响应用户，而不用将数据同步到所有备数据库。

除此之外，业务量很大的情况下，为了让服务更加健壮、低耦合、便于管理，会根据功能拆分为不同的服务。比如，将整个购票系统拆分为订单系统、支付系统和通知系统，而当购票系统拆分为 3 个子系统后，子系统之间不可避免地存在信息的交互，子系统之间的交互就会涉及分布式通信的相关知识，比如远程调用 RPC、消息队列等。

如图所示，用户购买火车票后，会首先在订单系统下单，下单成功后会调用支付系统的支付操作进行支付，之后将支付成功的消息存放到消息队列中，通知系统到消息队列中获取消息，最后通知用户购买成功。



总结

今天，我主要以购买火车票为例，为你串联了分布式技术在实践中的应用。

为方便理解，我将购买火车票的模型简化为三个核心步骤，即铁路局发布火车票、用户查询火车票和用户购买火车票。

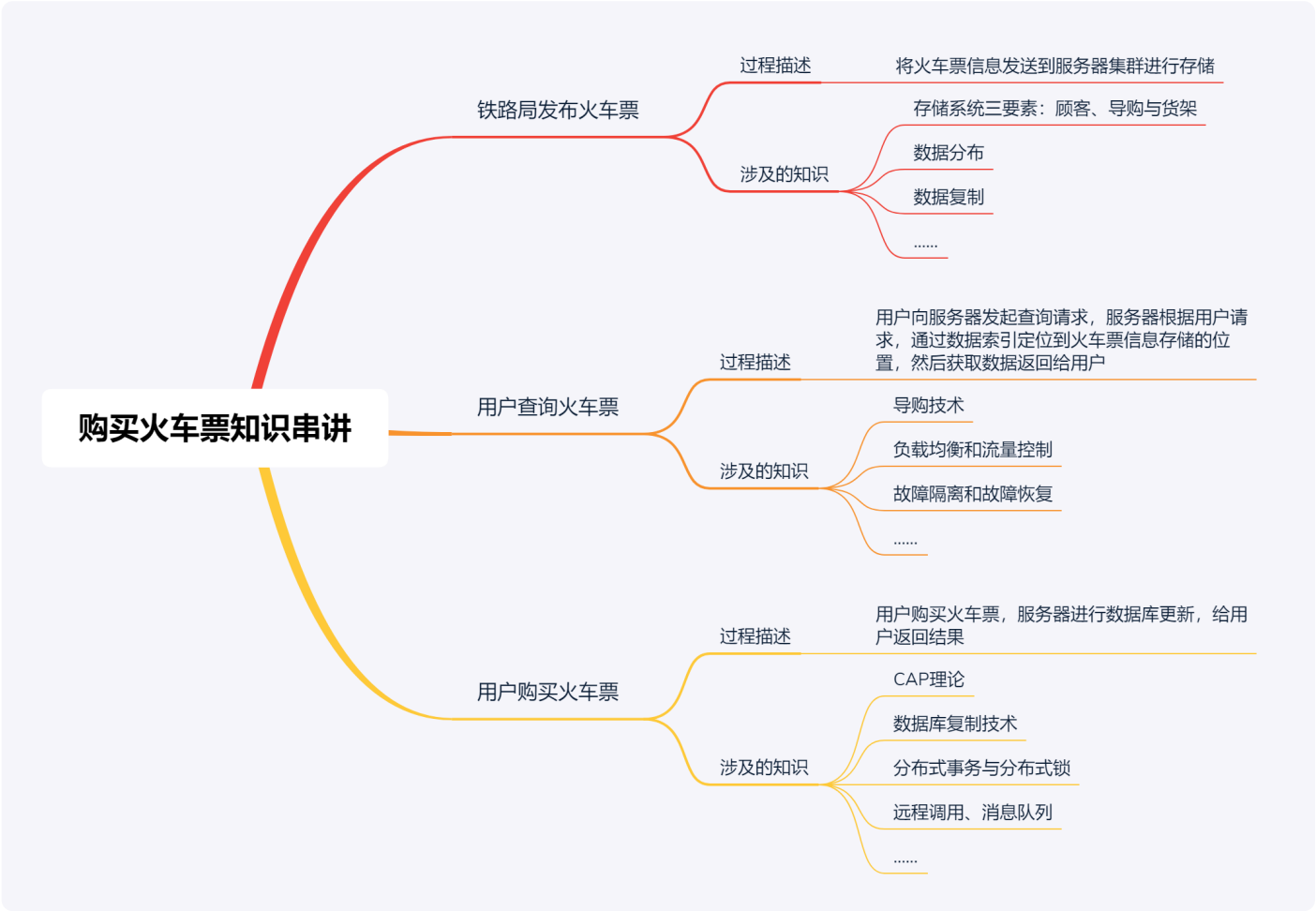
其次，我分别与你分析了这三个核心步骤涉及的关键的分布式技术。

对于铁路局发布火车票这个流程来说，铁路局是数据的生产者，需要将数据发布到服务器进行存储，主要涉及的是分布式数据存储相关技术，对应专栏“第五站：分布式数据存储”的内容。

对于用户查询火车票来说，主要是读请求，涉及负载均衡、流量控制、集群管理及选主等技术，对应专栏“第一站：分布式协调与同步”“第二站：分布式资源管理与负载调度”和“第六站：分布式高可靠”的内容。

而对于用户购买火车票来说，是写请求，涉及数据的改变，因此除了用户查询火车票涉及的技术外，还额外涉及一致性、分布式事务、远程通信等技术，对应专栏“第一站：分布式协调与同步”、“第四站：分布式通信技术”、“第五站：分布式数据存储”和第六站：“分布式高可用”等内容。

最后，我再通过一张思维导图来归纳一下今天的核心知识点吧。



思考题

结合购买火车票这个案例，你能和我分享你身边的应用场景或系统，都涉及或采用了哪些分布式技术吗？

我是聂鹏程，感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再会！

精选留言 (9)

任鹏斌
2019-12-13

老师能不能讲讲高并发下的库存扣减方案？如果完全依赖数据库的话容易出瓶颈吧？一般的大厂如何做这一块的？是不是要借助缓存？缓存的话要考虑缓存和数据库的一致性。感觉这一块要做好不容易。

共 1 条评论 >　　8

阿卡牛
2019-12-13

分布式系统里面涉及太多知识点了，细节才是魔鬼

4

Jackey
2019-12-13

发现前面的知识忘差不多了😂需要规划一下二刷了

3

leslie
2019-12-13

现在的系统RMDB真正承担的是和付款相关的事情，MQ和NOSQL 承担了上面的环节。我记得这套系统早期是单独用了sybase,然后跪了，换了oracle效果提升不明显。

如同老师之前课程所说的电商案例：拆分成了多个库，付款环节都有等待时间，这个时间才是真实与数据库的交互；最近看到的一些系统这方面做的很不好，导致数据库异常以及有时压力过大，这也是引发今年我仅仅是数据库性能优化问题然后就把组成原理和系统架构以及老师的分布式技术原理不惜一遍的原因。

从原理的角度去看：有时有些业务问题更加看的清楚更能明白问题的根源所在；知道问题的根据就知道后续如何真正去处理和解决。

谢谢老师今天的分享：一路跟随的过程其实很快，不知不觉发现课程居然快结束了，从初秋已至深冬了。谢谢老师一路的教诲，期待老师下周的分享。

3

钱
2020-02-21

回想一下我们的系统使用了那些分布式技术：
1：分布式负载均衡，几乎所有的服务都是分布式的，所以，负载均衡策略必不可少
2：分布式锁，比如控制只有一个业务用户可以导入数据或者刷新缓存
3：RPC、MQ、REDIS这些标配也少不了
4：有集群那分布式选主，故障隔离和恢复也必不可少
回想一下老师讲的这些都是会使用到的，在分布式系统中，这些好像是自行车的链条一般一个紧扣一个少了那个环节，整个链条就不再完整了，那自行车想跑起来就困难啦！
这一整套技术信息量比较多，也不是一个人或一个团队就能搞定的，就像分布式系统一样研发团队也是分布式的。有不同的研发小组就像一个小集群，里面会有一个小组长，然后有了需求组长会负责负载均衡的分发出去，根据不同组员的能力采用不同的策略。

2

安排
2019-12-22

主备和主从的区别是什么呢？ 为什么有时候说主备，但是又把被叫做slave节点，都搞混了

作者回复: Master、Slave其实在中文中对应的是主从的意思。节点间存在主从关系只有在集中式架构中才会出现。

在分布式架构中除了主从关系，还有主备关系，主备关系是一种角色对等但互为备份的关系。当主故障了，备会顶替主对外提供服务，有点类似我们足球队中的替补球员。主备关系在集中式架构和非集中式架构中都存在。

共 2 条评论>

3

大魔王汪汪
2019-12-16

老师问下，是否采用半同步复制技术这种需要视业务场景而来吧，如果对于数据一致性要求很高是否完全采用同步方案呢

2

蓝魔丶
2019-12-14

文中“比如，用户 A 购买 2019 年 10 月 12 日北京到上海的 T12 的火车票，已购买成功，座位号为 3 车厢 23B。假设主节点和备节点之间数据不一致，主节点上已经减去该火车票，但未在备节点上减去。此时，若主节点故障，备节点升主，用户 B 此时申请购买相同火车票，系统将 3 车厢 23B 火车票又卖给了用户 B。等到乘车时，用户 A 和 B 就难免“打架”了”这个问题的解决方案是什么呢?

共 3 条评论 >

2

随心而至
2019-12-13

重要的文章看三遍👀

3