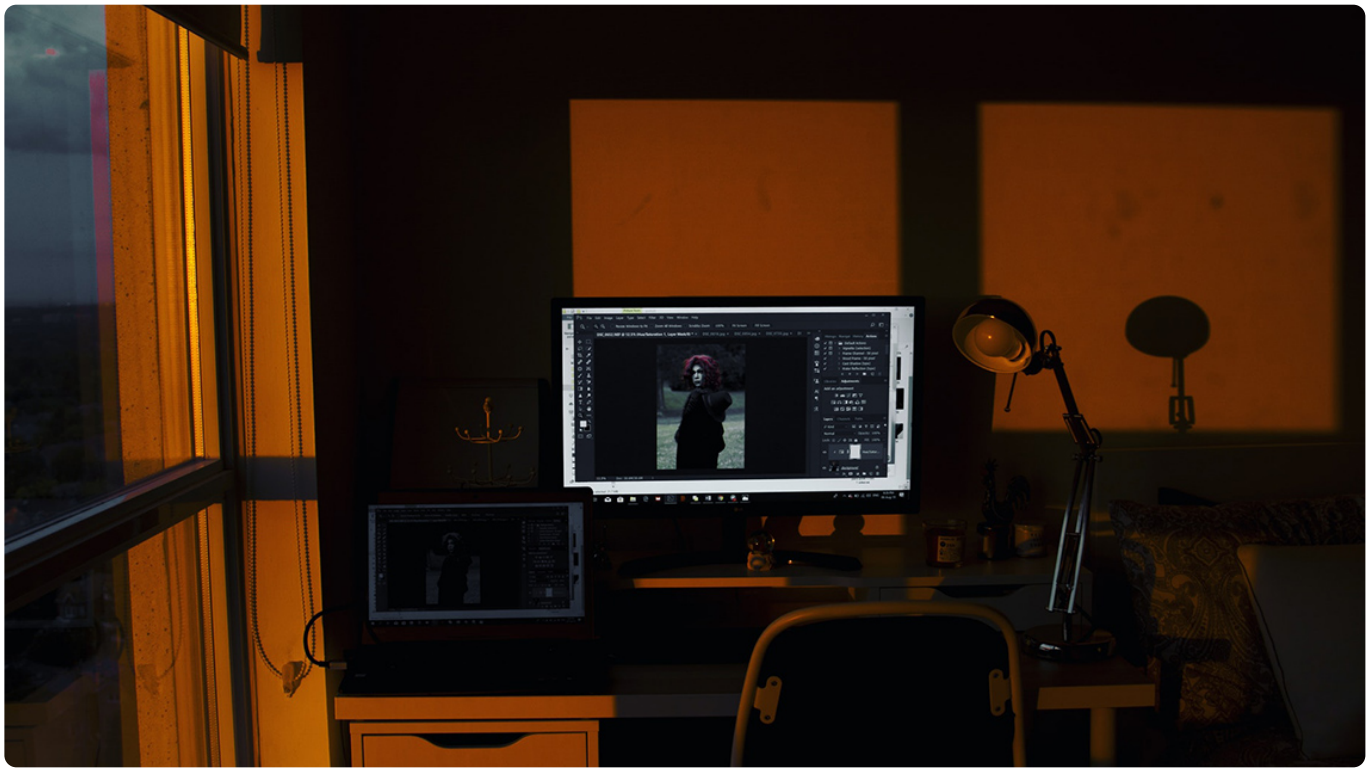
37 | 键值存储与数据库

2019-08-30 许式伟

许式伟的架构课 进入课程 >



讲述：姚迪迈
时长 10:53　大小 9.98M

你好，我是七牛云许式伟。

上一讲我们介绍了存储中间件的由来。今天我们就聊一下应用最为广泛的存储中间件：数据库。

数据库的种类

从使用界面（接口）的角度来说，通常我们接触的数据库有以下这些。

使用最为广泛的，是关系型数据库（Relational Database），以 MySQL、Oracle、SQLSever 为代表。

这类数据库把数据每个条目（row）的数据分成多个项目（column），如果某个项目比较复杂，从数据结构角度来说是一个结构体，那么就搞一个新的表（table）来存储它，在主表只存储一个 ID 来引用。

这类数据库的特点是强 schema，每个项目（column）有明确的数据类型。从业务状态的角度看，可以把一个表（table）理解为一个结构体，当遇到结构体里面套结构体，那么就定义一个子表。

第二类是文档型数据库（Document Database），以 MongoDB 为代表。这类数据库把数据每个条目（row）称为文档（document），每个文档用 JSON 或其他文档描述格式表示。

当前文档型数据库大部分是无 schema 的，也就是在插入文档时并不对文档的数据格式的有效性进行检查。

这有好有坏。好处是使用门槛低，升级数据格式方便。不好之处在于，质量保障体系弱化，数据可能被弄脏而不自知。可以预见的是，未来也会诞生强 schema 的文档型数据库。

第三类是键值存储（KV Storage），以 Cassandra 为代表。

键值存储从使用的角度来说，可以认为是数据库的特例。数据库往往是允许设定多个索引字段的，而键值存储明确只有唯一索引。

从实现角度来说，键值存储是数据库的基础。每一组数据库的索引，往往背后就是一组键值存储。

事务

无论是何种数据库，都面临一个重大选择：是否支持事务。这是一个艰难选择。从需求角度来说，事务功能非常强大，没道理不去支持。从实现角度来说，事务支持带来极大的负担，尤其是在分布式数据库的场景。

什么是事务？简单来说，事务就是把一系列数据库操作变成原子操作的能力。展开来说，事务的特性我们往往简称为 ACID，详细如下。

原子性（Atomicity）：在整个事务中的所有操作，要么全部完成，要么全部不做，没有中间状态。对于事务在执行中发生错误，所有的操作都会被回滚，整个事务就像从没被执行过一样。

一致性（Consistency）：事务的执行必须保证系统的一致性。这一点拿转账为例最容易理解。假设 A 有 500 元，B 有 300 元，如果在一个事务里 A 成功转给 B 50 元，那么不管并行发生了其他什么事，A 账户一定得是 450 元，B 账户一定得是 350 元。

隔离性（Isolation）：事务与事务之间不会互相影响，一个事务的中间状态不会被其他事务感知。

持久性（Durability）：一旦事务完成了，那么事务对数据所做的变更就完全保存在了数据库中，即使发生停电，系统宕机也是如此。

如果我们忽略性能要求，事务是很好实现的，只需要用一把能够 Lock/Unlock 整个数据库的大锁就够了。

但这显然不现实，一把大锁下来，整个数据库就废了。从 IOPS（IO 吞吐能力）角度来说，为什么分布式数据库很讨厌事务是很容易理解的：如果没有事务，一次数据库操作很容易根据数据的分区特征快速将操作落到某个分区实例，剩下来的事情就纯粹是一个单机数据库的操作了。

一种常见的事务实现方式是乐观锁。

什么是乐观锁?

常规的锁是先互斥，再修改数据。不管是不是发生了冲突，我们都会先做互斥。

但乐观锁不同，它是先计算出所有修改的数据，然后最后一步统一提交修改。提交时会进行冲突检查，如果没有冲突，也就是说，在我之前没有人提交过新版本，或者虽然有人提交过新版本，但是修改的数据和我所依赖的数据并不相关，那么提交会成功。否则就是发生了冲突，会放弃本次修改。

这意味着，每个数据有可能有多个值。如下：

KEY_i -> [(VER_0, VAL_0), (VER_1, VAL_1), ...]

其中，VER_0 对应当前已经提交的值 VAL_0，VER_1 对应事务$_1$ 中修改后的值 VAL_1，以此类推。

除了修改后的值外，每个事务还需要记录自己读过哪些数据。不幸的是，它并不是记录读过的 KEY 列表那么简单，而是要记录所有的读条件。

例如，对于 SELECT name, age, address WHERE age>17 这样一个查询，我们不是要记录读过哪些 name、age、address，而是认为我们读过所有 age>17 的条目（row）。

在事务提交的时候，锁住整个数据库（前面修改过程事务间不冲突，所以不需要锁数据库），检查所有记录的读条件，如果这些读条件对应的条目（row）的已提交版本都<=基版本（VER_0），那么说明不冲突，于是提交该事务所有的修改并释放锁。

如果事务提交的时候发现和其他已提交事务冲突，则放弃该事务，对所有修改进行回滚（其实是删除该事务产生的版本修改记录）。

到这里我们就可以理解为什么要用乐观锁了：至少它让锁数据库的粒度降到最低，判断冲突的逻辑也都是可预期的行为，这就避免了出现死锁的可能。

我们很容易可以推理得知，在所有并行执行的事务中，必然有一个事务的提交会成功。这样就避免了饥饿（永远都没人可以成功）。

主从结构

一旦我们考虑数据库的业务可用性和数据持久性，我们就需要考虑多副本存储数据。可用性（Availability）关注的是业务是否正常工作，而持久性（Durability）关注的是数据是否会被异常丢失。

当我们数据存在多个副本时，就有数据一致性的问题。因为不同副本的数据可能值不一样，我们到底应该听谁的。

我们的服务同时存在很多并发的请求，这就可能存在客户端 A 希望值是 VAL_a ，客户端 B 希望值是 VAL_b 的情况。

解决这个问题的方法之一是采用主从（Master-Slave）结构。主从结构采用的是一主多从模式，所有写操作都发往主（Master），所有从（Slave）都从主这边同步数据修改的操作。

这样，从（Slave）的数据版本只可能因为同步还没有完成，导致版本会比较旧，而不会出现比主（Master）还新的情况。

从（Slave）可以帮主（Master）分担一定的读压力。但是不是所有的读操作都可以被分担。大部分场景的读操作必须要读到最新的数据，否则就可能会出现逻辑错乱。只有那些纯粹用于界面呈现用途，而不是用于逻辑计算的场景，非敏感场景（比如财务场景是敏感场景）下能够接受读的旧版本数据，可以从从节点读。

从（Slave）最重要的是和主（Master）形成了互备关系。在主挂掉的时候，某个从节点可以替代成为新的主节点。这会发生一次选举行为，系统中超过一半的节点需要同意某个节点成为主，那么选举就会通过。

考虑选举的话，意味着集群的节点数为奇数比较好。比如，假设集群有 2 个节点，只有一主一从，那么在主挂掉后，因为只剩下一个节点参与选举，没有超过半数，选举不出新的主节点。

选择谁成为新的主是有讲究的，因为从的数据有可能不是最新的。一旦我们选择没有最新数据的从作为新的主节点，就意味着版本回退，也就意味着发生了数据丢失。

这是不能接受的事情。为了避免版本回退，写操作应该确保至少有一个从节点收到了最新的数据。这样在主挂掉后才可以确保能够选到一个拥有最新数据的节点成为新的主节点。

分布式

多副本让数据库的可用性和持久性有了保障，但是仍然有这样一些问题需要解决：

数据规模大到一定程度后，单个物理节点存放不了那么大的数据量；

主承受的读写压力太大，单台主节点承受不了这样高的 IOPS（吞吐能力）。

从目前存储技术的发展看，单台设备的存储量已经可以非常高，所以上面的第二种情况也会很常见。

怎么解决?

分布式。简单说，就是把数据分片存储到多台设备上的分片服务器一起构成一个单副本的数据库。分片的方式常见的有两种：

- 哈希分片（Hash based sharding）；
- 范围分片（Range based sharding）。

无论哪个分片方式，都会面临因为扩容缩容导致的重新分片过程。重新分片意味着需要做数据的搬迁。

数据迁移阶段对数据访问的持续有不低的挑战，因为这时候对正在迁移的分片来说，有一部分数据在源节点，一部分数据在目标节点。

在分布式存储领域，有一个著名（CAP）理论。其中，C、A、P 分别代表一个我们要追求的目标。

- 数据一致性 (Consistency)：如果系统对一个写操作返回成功，那么之后的读请求都必须读到这个新数据；如果返回失败，那么所有读操作都不能读到这个数据。
- 服务可用性 (Availability)：所有读写请求在一定时间内得到响应，可终止、不会一直等待。
- 分区容错性 (Partition-tolerance)：在网络分区的情况下，被分隔的节点仍能正常对外服务。

那么 CAP 理论说的是什么？简单说，就是 C、A、P 三个目标不能兼得，我们只能取其二。

假设我们不会放弃服务的可用性，那么我们决策一个分布式存储基本上在数据一致性（C）和分区容错性（P）之间权衡。

数据一致性（C）的选择基本上是业务特性决定的，业务要求是强一致，我们就不可能用最终一致性模型，相应的，我们只能在分区容错性（P）上去取舍。

结语

今天我们概要讨论了数据库相关的核心话题。我们第一关心的，当然还是使用界面（接口）。从使用界面角度，我们要考虑选择关系型数据库还是文档型数据库，以及是否需要事务特性。

确定了我们要使用什么样的数据库后，接着我们从实现角度，考虑主从结构和分布式方面的特性。

数据库是非常专业并且复杂的领域，限于篇幅我们这里不能展开太多，你如果有兴趣可以参考相关的资料。

如果你对今天的内容有什么思考与解读，欢迎给我留言，我们一起讨论。下一讲我们将聊聊对象存储。

如果你觉得有所收获，也欢迎把文章分享给你的朋友。感谢你的收听，我们下期再见。

上一篇　36 | 业务状态与存储中间件

精选留言 (13)

写留言

leslie
2019-08-30

老师的结语中其实有一点有误："从使用界面角度，我们要考虑关系型数据库还是文档型数据库，以及是否需要事务特性"，文档型数据库其实只是非关系型数据库中的一种，而不是一类，键值存储redis同样是非关系型数据库，早期的memcache；其实它们有个共同的名称-内存数据库。

关系型数据库和非关系数据库的关系不是相互取代而是相互补充：MYSQL 8.0已经...

展开

作者回复: 多谢补充。我目前的确没有把redis归类到数据库，而是归类的类似memcache的内存缓存。

7

兢
2019-09-02

文中提到”为了避免版本回退，写操作应该确保至少有一个从节点收到了最新的数据。”请问是如何确保至少有一个从节点收到了最新的数据，是每个写操作后都去验证一遍从库是否同步数据成功吗？如若是这样的话那如何去平衡效率问题？

展开

作者回复: 这个具体就仁者见仁智者见智了，方法总比困难多😁

1

靠人品去赢
2019-08-30

这个主从关系，我理解就是我们说的读写分离，可以分担一些压力，但有的时候确实需要最新的数据，比如提交订单了，显示余额，要最新的肯定是主库查。那问题来了什么时候主库什么时候分库呢？如果是浏览商品可以salve查余额，金额变化了就要查master主库，单纯的从业务上来判断吗？是不是做不到真正的读写分离？

展开

作者回复: 理论上读写分离是可行的，因为写的时候需要保证应该一主一从写成功，那么如果能够确认某个slave总是最新的话，可以分担读。

1

饭
2019-08-30

老师，一直不太明白，文档型数据库什么情况下应用会比较合适了？公司项目都是把他当临时缓存在用，把一些调用频繁的json格式的数据存上面。我不太理解，用redis不也可以了吗，还轻量级一些。

作者回复: 的确算误用。缓存和存储是两码事。

1

CrazyAirhead
2019-09-01

https://pingcap.com/blog-cn/tidb-internal-1/

https://pingcap.com/blog-cn/pessimistic-transaction-the-new-features-of-tidb/
一起看看这两篇效果更佳。

展开

作者回复: 多谢推荐

williamcai
2019-09-01

老师，CAP中的P，这个概念有点不太懂，百度了一下好多不一致的说法，老师能解释一下吗

作者回复: P简单来说，就是网络出现分区（变成两个相互独立的集群）时，是不是还可以正常提供服务。如果可以正常服务，说明分区容忍度高。

Charles
2019-08-31

老师说的，主从结构，主挂掉的情况下，两个以上从选举行为MySQL中是自动完成的吗？主恢复的时候，还需要额外做什么操作才能恢复原来的主吗？

展开

Jxin
2019-08-30

大佬的课，至今没有一篇能一遍看懂的，除了这篇。所以说这篇讲得太浅，无论是具体的技术，还是之上的思想都太浅。这有失大佬水准。之前的文章能力有限留言不了，这篇却是没什么点可以留言的。

作者回复: 我们大部分基础平台或基础软件相关的内容是以需求分析和平台的关键点解剖为主，并不是以内容深浅为准则。很难保证每一篇都有高深的东西。上一篇关于存储中间件的通用话题讨论完了，这一篇基本上就只能谈很具体的数据库相关的领域需求了。

1

Dean
2019-08-30

老师谈到CAP理论，目前感觉绝大多数系统都需要P，只能在C和A之间做取舍，对于A其实大部分场景也不能放弃，所以最后只能在C上退让。在出现网络分区后，仍然尽量处理请求，但各分区之间会有数据不一致的情况。老师可以说说哪些系统是绝对支持C的吗?

展开

作者回复: mongodb 就可以选择不同的一致性模型，可以选择强一致性。

许童童
2019-08-30

跟着老师一起精进。

展开

humor
2019-08-30

如果事务提交的时候发现和其他已提交事务冲突，则放弃该事务，对所有修改进行回滚（其实是删除该事务产生的版本修改记录）。
Mysql的InnoDB会有这种情况出现吗？我理解的InnoDB应该是在事务更新时会加行锁或者间隙锁，如果另外一个事务也对锁范围内的行做更新的话，会一直阻塞直到前一个事务执行完毕释放锁或者超时吧。

展开

作者回复: InnoDB 的确不是用乐观锁。

Aaron Cheung
2019-08-30

打卡 37

展开

大糖果
2019-08-30

老师好，看完文章对数据库有了一个整体概念，老师说如果有兴趣可以参考相关的资料。那么这个相关的资料有什么推荐吗？

展开

作者回复: 后面这一章的总结篇会给一些参考