读写分离
读写分离的数据不一致问题

订单系统读写分离方案设计与实现

使用Redis 作为MySQL的前置缓存，可以帮助MySQL挡住绝大部分的查询请求。这种方法对于像电商中的商品系统、搜索系统这类与用户关联不大的系统、效果特别好。因为在这些系统中、任何人看到的内容都是一样的，也就是说，对后端服来说，任何人的查询请求和返回的数据都是一样的。在这种情况下，Redis 缓存的命中率非常高，几乎所有的请求都可以命中缓存。

但是与用户相关的系统（不是用户系统本身，用户信息等相关数据在用户登录时进行缓存，就价值很高），使用缓存的效果就没有那么好了，比如，订单系统、账户系统、购物车系统、订单系统等等。对于这些系统而言，各个用户查询的信息与用户自身相关，即使同一个功能界面，用户看到的数据也是不一样的。

比如，“我的订单”这个功能，用户看到的都是自己的订单数据。在这种情况下，缓存的命中率就比较低了，会有相当一部分查询请求因为命中不了缓存,穿透到 MySQL 数据库中。

随着系统的用户数量越来越多,穿透到MySQL 数据库中的读写请求也会越来越多，当单个MySQL支撑不了这么多的并发请求时,该怎么办？最常见的解决思路就是读写分离和分库分表。这一章节我们先来分享第一种优化思路-读写分离。

读写分离

读写分离是提升 MySQL 并发能力的首选方案，当单个MySQL无法满足要求的时候，只能用多个MySQL实例来承担大量的读写请求。MySQL与大部分常用的关系型数据库一样,都是典型的单机数据库,不支持分布式部署。用一个单机数据库的多个实例组成一个集群,提供分布式数据库服务,是一件非常困难的事情。

一个简单且非常有效的是用多个具有相同数据的MySOL实例来分担大量查询请求，也就是“读写分离”。很多系统，特别是互联网系统，数据的读写比例严重不均衡，读写比例一般在9:1到几十比1，即平均每发生几十次查询请求，才会有一次更新请求，那就是说数据库需要应对的绝大部分请求都是只读查询请求。

分布式存储系统支持分布式写是非常困难的，因为很难解决好数据一致性的问题。但分布式读相对来说就简单得多，能够把数据尽可能实时同步到只读实例上，它们就可以分担大量的查询请求了。

读写分离的另一个好处是，实施起来相对比较简单。在数据层面，通常由数据库产品搭建主从同步集群。然后在应用层面，只需要将 SQL 语句根据读写逻辑路由到不同的数据库即可。

以 MySQL 数据库为例，通常通过 MySQL 提供的binlog 来实现实时数据同步。其他数据库产品通常也有类似的解决方案。

即在主库上打开 Binlog 日志。这个日志会记录主库上的每一次数据操作。然后在从库上打开 RelayLog 日志。从库会主动跟踪主库上的 Binlog 日志变化，并将日志中的数据操作同步记录到自己的 RelayLog 日志中。而从库接下来就会重演 RelayLog 中记录的数据操作，从而达到与主库数据同步的效果。

再到应用层面，把使用单机MySQL的系统升级为读写分离的多实例架构非常容易，一般不需要修改系统的业务逻辑，只需要简单修改DAO (Data Access Object,一般指应用程序中负责访问数据库的抽象层)层的代码,把对数据库的读写请求分开，请求不同的MySQL实例就可以了。

在电商项目当中，是采用的ShardingSphere框架实现的读写分离。这也是目前最常用的一种实现方式。

主库负责执行应用程序发来的数据更新请求，然后将数据变更同步到所有的从库中。这样，主库和所有从库中的数据一致，多个从库可以共同分担应用的查询请求。

通过读写分离这样一个简单的存储架构升级，数据库支持的并发数量就可以增加几倍到十几倍。所以，当系统的用户数越来越多时，读写分离应该是首要考虑的扩容方案。

读写分离的数据不一致问题

读写分离的一个副作用是，可能会存在数据不一致的问题。原因是数据库中的数据在主库完成更新后，是异步同步到每个从库上的，这个过程会有一个微小的时间差。正常情况下，主从延迟非常小，以几毫秒计。但即使是这样小的延迟，也会导致在某个时刻主库和从库上数据不一致的问题。

应用程序需要能够接受并克服这种主从不一致的情况，否则就会引发一些由于主从延迟而导致的数据错误。

回顾我们的订单系统业务，用户对购物车发起商品结算创建订单，进入订单页，打开支付页面进行支付，支付完成后，按道理应该再返回到支付之前的订单页。但如果这时马上自动返回到订单页，就很有可能会出现订单状态还是显示“未支付”的问题。因为支付完成后，订单库的主库中订单状态已经更新了，但订单页查询的从库中这条订单记录的状态可能还未更新，如何解决这种问题呢?

其实这个问题并没有特别好的技术手段来解决，所以可以看到，稍微上点规模的电商网站并不会支付完成后自动跳到到订单页，而是增加了一个支付完成页面，这个页面其实没有任何新的有效信息，就是告诉你支付成功的信息。如果想再查看一下刚刚支付完成的订单，需要手动选择，这样就能很好地规避主从同步延迟的问题。

如果是那些数据更新后需要立刻查询的业务，这两个步骤可以放到一个数据库事务中，同一个事务中的查询操作也会被路由到主库，这样就可以规避主从不一致的问题了，还有一种解决方式则是对查询部分单独指定进行主库查询。

总的来说，对于这种因为主从延迟而带来的数据不一致问题，并没有一种简单方便且通用的技术方案可以解决，对此，我们需要重新设计业务逻辑，尽量规避更新数据后立即去从库查询刚刚更新的数据。

> 有道云笔记链接：https://note.youdao.com/s/ZHrFR3yw