12 | 如何利用数据库实现并发扣减?

在后台开发领域，高并发的扣减一直是比较热门的话题。在各类技术博客、大会分享以及面试问题中，出现频率都非常高，可见它的重要性和技术知识点的密集性。从本讲开始，我将由浅入深、由简至繁地介绍三种能够支撑不同并发量级的解决方案，首先介绍的是基于纯数据库实现的扣减方案。

什么是扣减类业务

看到这个标题很多人可能会有疑惑，扣减类业务不就是指秒杀吗，为什么要取这么抽象的名字呢?

但其实秒杀只是扣减类业务中的一个有代表性、具备一定技术复杂度的场景，它并不能代表扣减类业务的全部场景。我将在“**第 16 讲**”中详细讲解秒杀相关的内容。除了秒杀之外，常见的扣减类业务有：

- 购买一个或多商品时扣减的库存
- 商家针对用户设置的某个或几个商品最多购买次数
- 支付订单时扣减的金额
- ...

上述业务场景有几个共性点：购买的或设置需要扣减的数量一次可以是一个或多个；数量是共享的，每个用户都可以扣减某一个数据的数量。基于上述分析，可以给扣减类业务下一个定义：

> 它是需要通过对一个或多个已有的、用户间或用户内共享的数量，精准扣减成功才能继续的业务。

通过定义，将我们要讨论的扣减类业务圈定了一个边界和清晰的概念。希望你在本模块里和我一起锚定这个定义，防止出现因为定义不清楚导致的认知偏差和讨论分歧。

在了解了扣减类业务的定义之后，再来看看它和前面几个模块中涉及的读业务与 UGC 写业务的区别。

- **读业务的特点是写少读多**，同时写入为非在线类运营操作，写入的 SLA（Service Level Agreement，服务等级协议）要求级别较低，对于读的 SLA 最高。读数据因为不会改或者频率很低，所以可以采用数据不断前置应对性能等的要求。
- **UGC 写业务则和扣减业务类似**。写入均是 C 端（客户）操作，对写入的 SLA 要求级别最高。但 UGC 写业务的特点是写入的数据是用户私有的而不是共享的，同时写入不需要依赖已有的数据。对于 UGC 写业务，只要尽最大可能将数据存储下来即可。

相比上两类业务的各自特点与技术实现关注点，扣减类业务着重关注对历史已有数据的增减上，接下来我们就来具体看一看对扣减类业务的特点如何应对。

扣减类业务的技术关注点

发生扣减必然就会存在归还。比如用户购买了商品之后因为一些原因想要退货，这个时候就需要将商品的库存、商品设置的购买次数及订单金额等进行归还。因此，在实现的时候还需要考虑归还。但是因归还的实现较通用，且归还是后置流程对并发性要求并不高，故本模块会先用三讲介绍如何应对高并发扣减，再来讲解如何实现归还。

基于扣减类业务的定义，我把关于扣减的实现，需要关注的技术点总结如下：

- 当前剩余的数量需要大于等于当次需要扣减的数量，即不允许超卖；

- 对同一个数据的数量存在用户并发扣减，需要保证并发一致性；
- 需要保证可用性和性能，性能至少是秒级；
- 一次的扣减会包含多个目标数量；
- 当次扣减有多个数量时，其中一个扣减不成功即不成功，需要回滚。

对于返还的实现需要关注的技术点如下：

- 必须有扣减才能返还；
- 返还的数量必须要加回，不能丢失；
- 返还的数据总量不能大于扣减的总量；
- 一次扣减可以有多次返还；
- 返还需要保证幂等。

在了解了扣减类业务的场景、定义，确定了在实现时需要包含的功能点，以及各个功能点的实现要求后，下面我将介绍三种不同方式的实现方案。这三个方案都能够满足上述要求的功能和对应的技术点要求，但三个方案的实现复杂度以及能够支撑的性能和并发量级均有一定的区别。

下面介绍的实现方案将直接以库存扣减为蓝本。其他扣减场景，比如：限次购买、支付扣减等技术方案基本类似，你可以举一反三。下面我们先来介绍第一种方案——纯数据库的扣减。

纯数据库式扣减实现

顾名思义，纯数据库的方案就是扣减业务的实现完全依赖数据库提供的各项功能，而不依赖其他额外的一些存储和中间件了。**纯数据库实现的好处是逻辑简单、开发及部署成本低。**

纯数据库的实现之所以能够满足扣减业务的各项功能要求，主要是依赖各类主流数据库提供的两个特性：

- 第一是基于数据库乐观锁的方式保证数据并发扣减的强一致性；
- 第二是基于数据库的事务实现批量扣减部分失败时的数据回滚。

基于上述特性实现的架构方案如下图 1 所示，它包含一个扣减服务和一个数量数据库。

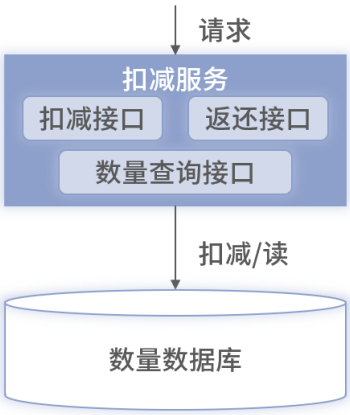

图 1：纯数据库扣减架构图

> 注意，扣减服务是一个后端服务，因本课程是介绍后台架构，对于扣减中涉及的前端技术不进行赘述，后续再提到的各种服务可直接默认为提供 RPC 接口的后端应用。

数量数据库存储扣减中的所有数据，主要包含两张表：扣减剩余数量表和流水表。扣减剩余数量表是最主要的表，包含实时的剩余数量。主要结构如下表 1 所示：

表 1：扣减剩余数量表

字段名	英文表示	含义
商品 ID	SKU	商品标识
当前剩余可购买数量	LeavedAmount	剩余可购买的商品数量，随着扣减实时变化

如上表所示，对于当前剩余可购买的数量，当用户进行取消订单、售后等场景时，都需要把数量加回到此字段。同时，当商家补齐库存时，也需要把数量加回。

从完成业务功能的角度看，只要扣减剩余数量表即可。但在实际场景中，会需要查看明细进行对账、盘货、排查问题等需求。其次，在扣减后需要进行返还时是非常依赖流水的。因为只能返还有扣减记录的库存数量。最后，在技术上的幂等性，也非常依赖流水表。下面我们来看一下流水表的主要结构，如下表 2 所示：

表 2：扣减流水表

字段名	英文表示	含义
扣减编号	uuid	表示一次成功的扣减记录
商品 ID	SKU	同上
此次扣减数量	num	此次调用扣减服务扣减对应商品的数量

1. 扣减接口实现

完成了存储的数据结构设计后，咱们再来学习一下扣减服务提供的扣减接口的实现。扣减接口接受用户提交的扣减请求，包含用户账号、一批商品及对应的购买数量，大致实现逻辑如下图 2 所示：

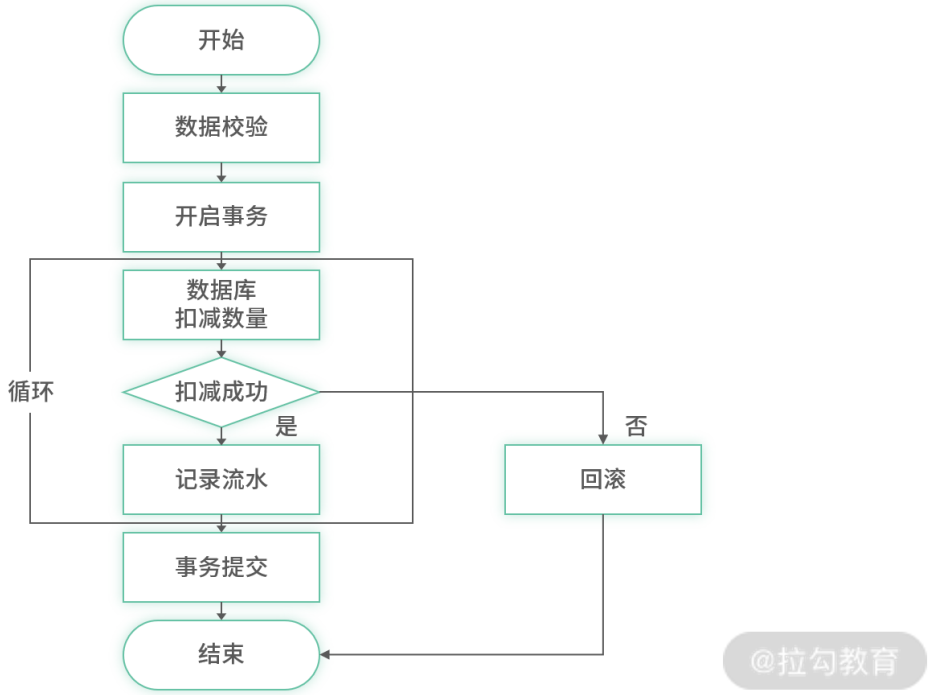

图 2：扣减实现流程

在图 2 的流程开始时，首先进行的是数据校验，在其中可以做一些常规的参数格式校验。其次，它还可以进行库存扣减的前置校验。比如当数据库中库存只有 8 个时，而用户要购买 10 个，此时在数据校验中即可前置拦截，减少对于数据库的写操作。纯读不会加锁，性能较高，可以采用此种方式提升并发量。

当用户只购买某商品 2 个时，如果在校验时剩余库存有 8 个，此时校验会通过。但在后续的实际扣减时，因为其他用户也在并发的扣减，可能会出现了幻读，即此用户实际去扣减时不足 2 个，导致失败。这种场景就会导致多一次数据库查询，降低了整体的扣减性能。其次，即使将校验放置在事务内，先查询数据库数量校验通过后再扣减，也会增加性能。

那是不是前置校验就不需要了呢？在实践中，前置校验是需要的。相比读，扣减的事务性能更差，两弊相衡取其轻，能避免则避免。此外，扣减服务提供的数量查询接口和校验中的反查底层实现是相同的，如果反查走库则都走库。在“**模块二：构建高性读服务**”中我们了解到，正常情况下，读比写的量级至少大十倍以上。因此，查询的性能问题仍须解决。关于如何规避性能问题、如何降低给数据库带来的压力，我们会在本讲后半部分详细讲解。

在事务之后，则是数据库更新操作。因为用户扣减的商品数量可以是一个或多个，只要其中一个扣减不成功，则判定用户不能购买。注意，因为在事务之后，对商品使用 for 循环进行处理，每一次循环都需要判断结果。如果一个扣减失败，则进行事务回滚。基于上述提供的两张表结构，单条商品的扣减 SQL 大致如下：

```
update stock set leavedAmount=leavedAmount-currentAmount where skuid='123456' and leavedAmount>=c
```

此 SQL 采用了类似乐观锁的方式实现了原子性，在 where 条件里判断此次需要的数量小于等于剩余的数量。在扣减服务的代码里，判断此 SQL 的返回值，如果值为 1 表示扣减成功，即用户此次购买的数量，当前的库存可以满足否则，返回 0 进行回滚即可。

扣减完成之后，需要记录流水数据。每一次扣减时，都需要外部用户传入一个 uuid 作为流水编号，此编号是全局唯一的。用户在扣减时传入唯一的编号有两个作用。

1. 当用户归还数量时，需要带回此编号，用来标识此次返还属于历史上的具体哪次扣减。
2. 进行幂等性控制。当用户调用扣减接口出现超时时，因为用户不知道是否成功，用户可以采用此编号进行重试或反查。在重试时，使用此编号进行标识防重。

当每一个 SKU 按上述流程都扣减成功了，则提交事务，说明整个扣减成功。

2. 扣减接口实现升级

在上面提到了前置校验的好处及存在的问题：多一次查询，就会增加数据库的压力，同时对整体服务性能也有一定影响。此外，对外提供的查询库存数量的接口也会对数据库产生压力，同时读的请求量要远大于写，由此带来的压力会更大。

根据业务场景分析，读库存的请求一般是顾客浏览商品时产生，而调用扣减库存的请求基本上是用户购买时才会触发。用户购买请求的业务价值相比读请求会更大，因此对于写需要重点保障。转换到技术上，价值相对低的读来说是可以降级的、有损的。对于写要尽可能性能好、尽量减少不必要的读与写请求（写本身非常消耗性能）等。

针对上述的问题，可以对整体架构进行升级，升级后的架构如下图 3 所示：

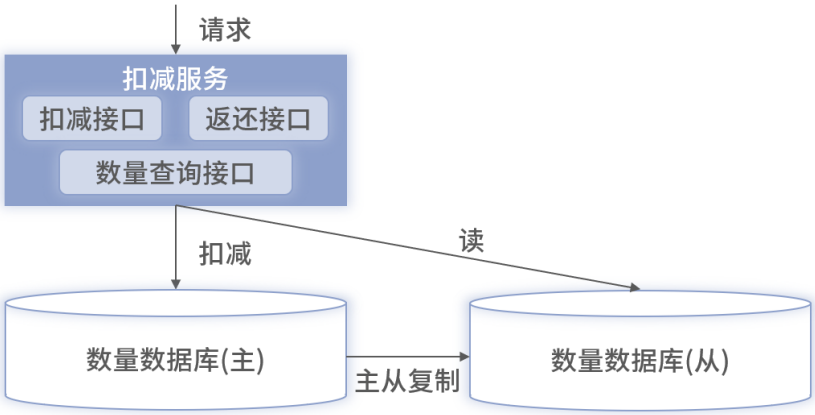

图 3：读写分离的扣减架构图

整体的升级策略采用了读写分离的方式，另外主从复制直接使用了 MySQL 等数据库已有功能，改动上非常小，只要在扣减服务里配置两个数据源。当客户查询剩余库存数量、扣减服务中的前置校验时，读取从数据库即可。而真正的数据扣减还是使用主数据库。

读写分离之后，根据二八原则，80% 的均为读流量，主库的压力降低了 80%。但采用了读写分离也会导致读取的数据不准确的问题，不过库存数量本身就在实时变化，短暂的差异业务上是可以容忍的，最终的实际扣减会保证数据的准确性。

不过，在上面提到的因为在扣减前，为了降低数据库的压力，增加的前置校验导致的性能下降问题，并没有得到太多实质性的升级解决。那么，接下来我们该从什么方向上解决这个问题呢？

3. 扣减接口实现再升级

在基于数据库的主从复制降低了主库流量压力之后，还需要升级的就是读取的性能了。这里用到了我们在“**模块二：构建高性能读服务**”里学习到的：使用 Binlog 实现简单、可靠的异构数据同步的技能，应用此方案后整体的架构如下图 4 所示：

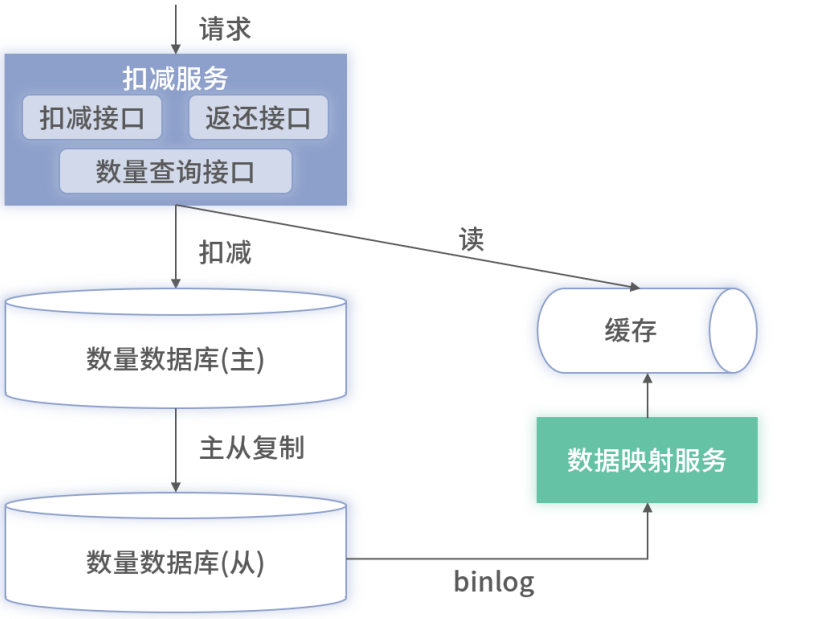

图 4：读写基于不同存储的架构图

和上面第 2 点实现的区别是增加了**缓存**，用来提升读取从库的性能。在技术实现上，采用了在“**模块二：构建高性能读服务**”里介绍的 Binlog 技术，这里不再赘述。

经过此次方案升级后，基本上解决了在前置扣减里校验环节及获取库存数量接口的性能问题，提高了用户体验性。

纯数据库扣减方案适用性

要知道，任何方案都是根据业务需求、实现成本进行综合分析和取舍，很难有一个实现方案将所有诉求 100% 满足，它都是有一定的优点也有对应的缺点。

对于采用数据库实现扣减服务的方案也不例外，整体实现方案上也是有它适用的场景以及它不适用的场景。

纯数据库方案主要有以下几个优点：

- 实现简单，即使读使用了前置缓存，整体代码工程就两个，即扣减服务与数据映射服务，在需求交付周期非常短、人力紧张的场景是非常适用的；
- 使用了数据库的 ACID 特性进行扣减。业务上，库存数量既不会出现超卖，也不会出现少卖。

但不足之处是，当扣减 SKU 数量增多时，性能非常差。因为对每一个 SKU 都需要单独扣减，导致事务非常大，极端情况下，可能出现几十秒的情况。

在上述的优点和不足背景下，请你思考以下两个问题：

- 此方案在落地上有适用场景吗？
- 或者有哪些适用场景呢？

在一些企业内部 ERP 系统里的次数限制、中小电商站点的库存管理、政府系统等场景里，其实此方案是比较适合的。因为此类系统的用户并发数、对于请求的耗时要求、购买商品的数量都比较小，如果一开始就采用后几**讲**里介绍的方案，其实是一种浪费。

当业务不断发展时，对上述指标有要求时，再去升级也不迟，毕竟系统是演化迭代来的，不是一天建成的。

到这里我们今天的重点内容就讲完了，此时你可能会有以下两个疑问，本讲的标题是“如何实现万级并发的扣减服务”，但从本篇所讲方案里和描述里，一没有提到具体数据，二从方案上看数据库并不能支持并发上万的扣减量，是不是有点标题党呢？

首先回答第一个疑问，在这里想做下正本清源，以后你看到任何类似单机过万的数字都要小心与仔细思考。因为任何没有机器配置的指标，都是耍流氓。如果我采用 Oracle 的数据库、100 多核的刀锋服务器、SSD 的硬盘，在一定情况下，即使是纯数据库的扣减方案，也是可以达到单机上万的 TPS 的。

对于第二个疑问，我想表达的思想是，任何架构都是不完美的，都是有取舍的。不要觉得单机过万或者并发过百万、千万就是最好，因为它在其他指标上会有更多的成本消耗。对于本讲标题设定的目标，请见后续章节分解。

总结

为了让你对扣减类业务有一个清晰的认知，我首先介绍了什么是扣减类业务、适用场景，以及它的实现需要满足的功能要求。之所以没有直接讲解各种实现方案的原因是：一个优秀的方案一定是建立在对本质问题的理解之上，也就是定义问题，偏离问题的解决方案是事倍功半的。这个思想是本讲的要点之一，希望你牢记并在未来进行系统架构时第一时间定义问题再进行设计。

在此之上，本讲还介绍了递进式的架构设计演化方案，先介绍了一个满足基本功能的纯数据库方案。当新的问题出现时，又递进地介绍了采用主从分离的方案以及进一步的主从 + 缓存前置的方案。在理解了这两个递进方案的实现之外，也希望你了解，方案是演化的，而不是一步到位的。

练习题

最后，我再给你留两道思考题。

1. 除了使用主从同步来提升读取性能，是否可以使用数据库索引来提升性能？另外，数量数据库应如何设置索引来满足如防重等诉求？
2. 能否将数据库进行分库或者分表，利用分库或分表来提升扣减写入的并发性？

你可以把你的答案、思路或者课后总结写在留言区，如果你觉得今天的内容对你有所启发，欢迎分享给身边的朋友。我们一起交流！