**分布式ID生成器**

1. **需求起源**

几乎所有的业务系统，都有生成一个唯一记录表示的需求，比如消息标识，订单标识，帖子表示等等，这个记录标识往往就是数据库的主键，数据库上会建立聚集索引，即在物理存储上以这个字段排序。

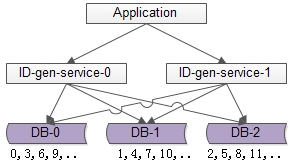
这个记录标识的查询，往往又有分页或者排序的业务需求，例如：拉取最新的一页消息、拉取最新的一页订单、拉取最新的一页帖子。所以往往要有一个time字段，并且在time字段上建立普通索引(non-cluster index)。

普通索引存储的是实际记录的指针，其访问效率会比聚集索引慢，如果记录标识在生成时能够基本按照时间有序，则可以省去这个time字段的索引查询，必须注意，能这么做的前提是id的生成基本是趋势时间递增的。

这就引出了记录标识生成的两大核心需求：全局唯一和趋势有序，所以分布式ID生成器的核心问题就是如何高效生成趋势有序的全局唯一ID。

1. **常见方法、不足与优化**

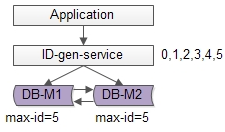
* ***方法一：使用数据库的auto\_increment来生成全局唯一递增ID***
* 优点
  + 简单，使用数据库已有功能
  + 能够保证唯一性
  + 能够保证递增性
  + 步长固定
* 缺点
  + 可用性难以保证：数据库常见架构是一主多从+读写分离，生成自增ID是写请求，主库挂了就宕了
  + 扩展性差，性能有上限：因为写入是单点，数据库主库的写性能决定ID的生成性能上线，并且难以扩展
* 改进方法
  + 冗余主库，避免写入单点
  + 数据水平切分，保证主库生成的 ID不重复。如下图，由一个写库变成3个，每个写库设置不同的auto\_increment初始值，以及相同的增长步长，以保证每个数据库生成的ID是不同的



* + 这样做虽然保证了可用性，但是仍然存在缺点：
    - 丧失了ID生成的“绝对递增性”：先访问库0生成0,3，再访问库1生成1，可能导致在短时间内，ID生成不是绝对递增的(问题不大，目标是趋势递增，不一定要求绝对递增)
    - 数据库的写压力依然很大，每次生成ID都要访问数据库
* ***方法二：单点批量ID生成服务***

分布式系统之所以难，很重要的原因是“没有一个全局时钟，难以保证绝对的时序”，要想保证绝对的时序，还是只能使用单点服务，用本地时钟保证“绝对时序”。

数据库写压力大，是因为每次生成ID都访问了数据库，可以使用批量的方式降低数据库写压力。



如上图，数据库使用双master保证可用性，数据库中只存储当前ID的最大值。ID生成服务假设每次批量拉取6个ID，服务可访问数据库，将当前ID的最大值，这样应用访问ID生成服务索要ID，ID生成服务不需要每次访问数据库，就能依次派发0,1,2,3,4,5这些ID了。

当ID发完后，再将ID的最大值修改为11，就能再次派发6,7,8,9,10,11,这些ID了，于是数据库的压力就降低为原来的1/6了

这样做的优点在于

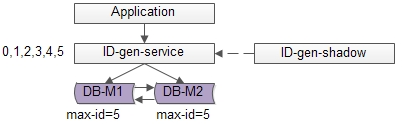
* + 保证了ID生成绝对递增有序
  + 大大的降低了数据库的压力，ID生成可以做到每秒生成(W)级别的

缺点在于

* + 服务仍然是单点
  + 如果服务挂了，服务重启之后，继续生成ID可能会不连续，中间出现空洞(服务内存是保存着0,1,2,3,4,5没数据库中max-id是5，分配到3时，服务重启了，下次会从6开始分配，4,5就成了空洞)
  + 虽然每秒可以生成几万几十万个ID，但毕竟还是有性能上限，无法进行水平扩展

改进方法

单点服务的常用高可用优化方案是“备用服务”，也叫“影子服务”，所以我们能用以下方法优化第一个缺点：



如上，对外提供服务的是主服务，但还有一个影子服务处于备用状态，当主服务挂了的时候，影子服务可以顶上。这个切换对调用方式透明的，可以自动完成，常用的技术vip+keepalived。

* ***uuid/guid***

不论是通过数据库还是通过服务生成ID，业务方Application总需要进行一次远程调用，比较耗时。而uuid(string ID=GenUUID())就是一种本地生成ID的方法，即高性能，又时延低，优点在于：

* + 本地生成ID，不需要进行远程调用，时延低
  + 扩展性好，基本可以认为没有性能上限

缺点

* + 无法保证趋势递增
  + uuid过长，往往用字符串表示，作为主键建立索引查询效率低，常见优化方案为“转化为两个uint64整数存储”或者“折半存储(不能保证唯一性)”
* ***取当前毫秒数***

uuid是一个本地算法，生成性能高，但无法保证趋势递增，且作为字符串ID检索效率低，取当前毫秒数是一种能够保证递增的本地算法(unit64 ID=GenTimeMS())

这样的优点在于

* + 本地生成ID，不需要进行远程调用，时延低
  + 生成的ID趋势递增
  + 生成的ID是整数，建立索引后查询效率高

缺点在于

* + 并发量超过1000，会生成重复ID

唯一性受到挑战。

* ***类snowflake算法***

snowflake是twitter开源的分布式ID生成算法，其核心思想是一个long型的ID：

* + 41bit作为毫秒数
  + 10bit作为机器编号
  + 12bit作为毫秒的内序列号

算法单机每秒内理论上最多生成1000\*(2^12)，也就是400W的ID，完全能满足业务的需求。借鉴这种方法，结合具体的业务逻辑和并发量，实现一个自己的分布式ID生成算法。

这样做也有缺点，由于“没有一个全局时钟”，每台服务器分配的ID是绝对递增的，但从全局看，生成的ID只是确实递增(各个服务器时间不一致)