目前采用的是分布式id生成器，系统已经运行了好几年了，但我们目前的数据库表的记录不到100w，我真不知道当时为啥选择分布式id生成器，用主键不香吗？一般用分布式ID主要是分库分表，但我们目前的业务增长量好像近期也不需要分库分表啊。而且分布式id生成器需要进行网络通讯，万一网络抖动了导致无法生成主键id，那岂不是很麻烦？凡是依赖于网络的都存在不可靠的因素。

有哪些可以被替换掉的?

代码，代码还是代码，代码做好兼容性就好了，也许会费点时间，那又何妨，换来一个清爽的，简约的结构不爽吗?

2. 技术方案选型
==========

先讲个故事，我有三个朋友，他们是大C，M，和小D。

大C做事情非常麻利从来不拖泥带水，交给他的事情他都能帮你办的妥妥的。

小D慢性子，社恐，但做事情非常仔细，循规蹈矩，不出格，你交给他的事情除了慢一点，没别的毛病。

M呢，和事佬，经常调节大C和小D之间的矛盾，大C总是嫌小D做事情拖拖拉拉，慢慢腾腾的，他们两个一旦有矛盾，M总会出现。

我这三个朋友他们分别是 CPU，Memory，Disk。

我偷偷的告诉你，后来大C觉得总是麻烦M，很不落忍，于是他经常把自己和小D的矛盾积攒在一起，然后一次性交给M来帮忙解决，积攒在一起的这个地方叫L Cache。

如果要做到高性能，业内通常的做法是加缓存，在快和慢之间。

**如果要做到高并发，那肯定不能一个人全把活干了，需要多个像CMD这样的

组合，这就是横向扩展。**

在上边的故事中，我们捋清楚了他们各自的角色，但有一点需要特别注意，小D的工作任务怎么能有条不紊的交给M呢，他们之间是不是得有条航线啊，这条航线叫操作 并不会存在因大量更新导致的性能低下。

如果采用策略2，如果有大量缓存失效，那将会有大量请求分发到数据库中，导致数据库压力上升，目前在读多写少的场景中，希望更多的命中缓存的方式。

如果采用策略1需要解决的问题是：消息的顺序性；容忍短暂的不一致

通过调研canal在同步binlog的机制中可以按照顺序进行同步 在高并发场景中不会出现错误，所以在业务场景中，我们选择了策略1。

但这样的方案仍然存在隐患，如果canal断了，我们该如何解决呢?

参考文章链接

[segmentfault.com/a/119000004...](http://cxyroad.com/”https://segmentfault.com/a/1190000041998615”)

[www.quora.com/Why-does-Fa...](http://cxyroad.com/”https://www.quora.com/Why-does-Facebook-use-delete-to-remove-the-key-value-pair-in-Memcached-instead-of-updating-the-Memcached-during-write-request-to-the-backend”)

[github.com/alibaba/can...](http://cxyroad.com/”https://github.com/alibaba/canal/wiki/Canal-Kafka-RocketMQ-QuickStart”)

原文链接: https://juejin.cn/post/7360498535076347944