https://mp.weixin.qq.com/s/IiFk5fnCjUueTA2deEJH3Q

**分布式ID（唯一性）的生成方法汇总**

## 1. JDK自带的UUID

程序设计语言开发工具包中都有生成主键的策略，以java语言的UUID为例（图1），它有着全球唯一的特性，可以做为分布式系统ID。核心思想是结合服务器的网卡、当地时间以及随记数来生成UUID。

* 优点是生成简单、性能好、全球唯一，在数据迁移、系统合并或者数据库变更的情况下都可以应对。
* 缺点就是生成的ID一般使用字符串存储，可读性性较差。在数据库中占有空间较大并且查询的效率比较低，在各微服务场景下，耗费的网络资源也响应增多。

## 2. 数据库自增ID

基于数据库的auto\_increment自增ID完全可以充当分布式ID，具体原理：创建一个单独的实例用来生成ID，用一张数据表存储目前的最大ID（图2）。

当我们需要一个ID的时候，向表中插入一条记录返回主键ID，但访问量激增时数据库本身就是系统的瓶颈，用它来实现分布式服务风险比较大，不推荐！

* 优点是数据库生成的ID绝对有序，高可用实现方式简单；
* 缺点是需要独立部署数据库实例，成本高，数据库压力大，性能有限（可通过DB集群设置不同步长改善，即部署N台数据库实例，每台设置成不同的初始值，自增步长为机器的台数，如图3所示，可实现多个ID实例自增），但这种方法可能会导致数据的安全性问题。

## 3. 号段模式

一次按需批量生成多个ID，每次生成都需要访问数据库，将数据库修改为最大的ID值，并在内存中记录当前值及最大值一般可以借助redis实现，如图4所示。

* 优点是避免了每次生成ID都要访问数据库并带来压力，提高性能；
* 缺点是属于本地生成策略，存在单点故障，服务重启造成ID不连续。

## 4. Redis生成

Redis服务器来也可以生成全局ID，这主要依赖于Redis是单线程的，所以也可以用生成全局唯一的ID 如图5。利用Redis的原子操作 INCR和INCRBY来实现。

* 优点是不依赖于数据库，灵活方便，性能高。数字ID天然排序，对分页或者需要排序的结果很有帮助。使用Redis集群也可以防止单点故障的问题；
* 缺点是依赖第三方组件Redis，增加系统复杂度。需要编码和配置的工作量比较大。

## 5. snowflake算法

snowflake 是 twitter 开源的分布式ID生成算法，其核心思想为，一个long型的ID：41 bit 作为毫秒数、10 bit 作为机器编号（10位的长度最多支持部署1024个节点）、12 bit 作为毫秒内序列号（12位的计数顺序号支持每个节点每毫秒产生4096个ID序号），如图6。

* 优点是简单高效，生成速度快。时间戳在高位，自增序列在低位，整个ID是趋势递增的，按照时间有序递增。灵活度高，可以根据业务需求，调整bit位的划分，满足不同的需求。不需要其他依赖，使用方便。
* 缺点是强依赖机器的时钟，如果服务器时钟回拨，会导致重复ID生成。在分布式环境上，每个服务器的时钟不可能完全同步，有时会出现不是全局递增的情况，不同机器配置不同worker id麻烦。

## 6. 百度UidGenerator

UidGenerator是Java实现的, 基于Snowflake算法的唯一ID生成器。UidGenerator以组件（图7）形式工作在应用项目中, 支持自定义workerId位数和初始化策略, 从而适用于docker等虚拟化环境下实例自动重启、漂移等场景。

* 优点是全局唯一，高可用、高性能解决了时钟回拨的问题；
* 缺点是内置WorkerID分配器, 依赖数据库，启动阶段通过DB进行分配; 如自定义实现, 则DB非必选依赖。

## 7. 美团Leaf

美团的Leaf分布式ID生成组件（图8）是在Snowflake算法的基础上做了两套优化的方案：Leaf-segment数据库方案（相比之前的方案每次都要读取数据库，该方案改用代理服务器批量获取，且做了双缓存的优化）与Leaf-snowflake方案（主要针对时钟回拨问题做了特殊处理。若发生时钟回拨则拒绝发号，并进行告警）。

* 优点是全局唯一，高可用、高性能用zookeeper解决了各个服务器时钟回拨的问题，弱依赖zookeeper；
* 缺点是依赖第三方组件，如zookeeper。

## 8. zookeeper生成唯一ID

zookeeper主要通过其节点的信息来生成序列号，可以生成32位或者64位的数据版本号，客户端可以使用这个版本号来作为唯一的序列号。

* 优点是实现原理较为简单，容易实现；
* 缺点是需要依赖zookeeper，并且是多步调用API，如果在竞争较大的情况下，需要考虑使用分布式锁。因此，性能在高并发的分布式环境下，也不甚理想。

总的来看，目前的实现方案主要分为两种：

第一有中心（如数据库，包括mysql，redis等），其中可以会利用约束条件来实现集群（起始步长）。

第二种就是无中心，通过生成足够散落的数据，来确保无冲突（如UUID等）。

中心化方案的优点是ID数据长度相对小一些、数据可以实现自增趋势等；缺点是容易发生并发瓶颈、集群需要实现约定、横向扩展困难等。非中心化方案的优点是实现简单、不会出现中心节点带来的性能瓶颈、扩展性较高（扩展的局限往往集中于数据的离散问题）；缺点是数据长度较长、无法实现数据的自增长。