分库分表后必然面对的一个问题，就是id怎么生成？因为要是分成多个表之后，每个表都是从1开始累加，那肯定不对，需要一个全局唯一的id来支持。

### 1、基于数据库的实现方案

#### 数据库自增id

你的系统里每次得到一个id，都是往一个库的一个表里插入一条没什么业务含义的数据，然后获取一个数据库自增的一个id。拿到这个id之后再往对应的分库分表里取写入。

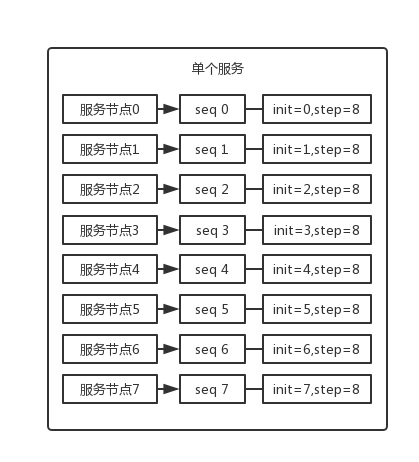
这个方案的好处就是方便简单，谁都会用：缺点就是单库生成自增id，要是高并发的话，就会有瓶颈；如果硬是要改进一下，那么久专门开一个服务出来，这个服务每次就拿到当前id最大值，然后自己递增几个id，一次性返回一批id，然后再把当前最大id值修改成递增几个id之后的一个值；但是无论如何都是基于单个数据库。

适合的场景：你分库分表就俩原因，要不就是单库并发太高，要不就是单库数据量太大；除非是你并发不高，但是数据量太大导致的分库分表扩容，你可以用这个方案，因为可能美妙最高并发最多就几百，那么就走单独的一个库和表生成自增主键即可。

#### 设置数据库sequence或者表自增字段步长

可以通过设置数据库sequence或者表的自增字段步长来进行水平伸缩。

比如说，现在有8个服务节点，每个服务节点使用一个sequence功能来产生ID，每个sequence的起始ID不同，并且一次递增，步长都是8。



适用场景：在用户防止产生的ID重复时，这种方案实现起来比较简单，也能达到性能目标。但是服务节点固定，步长也固定，将来如果还要增加服务节点，就不好搞了。

#### UUID

好处是本地生成，不要基于数据库；缺点就是UUID太长、占用空间大，作为主键性能太差；更重要的是，UUID不具有有序性，会导致B+树索引在写的时候有过多的随机写操作（连续的ID可以产生部分顺序写），还有，由于在写的时候不能产生有顺序的append操作，而需要进行insert操作，将会读取整个B+树节点到内存，在插入这条记录后会将整个节点写回磁盘，这种操作在记录占用空间比较大的情况下，性能下降明显。

适合的场景：如果你是要随机生成个什么文件名、编号之类的，你可以用UUID，但是作为主键不能用UUID的。

java.util.UUID.*randomUUID*().toString().replace("-", "");

#### 获取系统当前时间

这个就是获取当前时间即可，但是问题是，并发很高的时候，比兔一秒并发几千，会有重复的情况，这个肯定不合适的。基本就不考虑了。

适合的场景：一般如果用这个方案，是将当前时间跟很多其他的业务字段拼接起来，作为一个id，如果业务上你觉得可以接受，那么也是可以的。你可以将别的业务字段值跟当前时间拼接起来，组成一个全局唯一的编号。

#### snowflake算法

twitter开源的分布式id生成算法，采用Scala语言实现，是一个64位的long型id。

* 1bit：不用，二进制里第一个bit如果为1那么都是负数，我们生成的id都是正数，所以第一个bit统一都是0。
* 41bit：表示时间戳，单位是毫秒。41bit可以表示的数字多达2^41-1，也就是可以标识2^41-1个毫秒值，换算成年就是表示69年的时间。
* 10bit：记录工作机器id，代表的是这个服务最多可以部署在2^10台机器上，也就是1024台机器。但是10bit里5个bit代表机房id，5个bit代表机器id。最多代表2^5个机房（32个机房），每个机房里可以代表2^5个机器（32台机器）。
* 12bit：这个是用来记录同一个毫秒内产生的不同id，12bit可以代表的最大正整数是2^12 - 1 = 4096，也就是说可以用这个12bit代表的数字来区分同一毫秒内的4096个不同的id。



0 | 0001100 10100010 10111110 10001001 01011100 00 | 10001 | 1 1001 | 0000 00000000

41bit是当前毫秒单位的一个时间戳；然后5bit是你传递进来的一个机房id

（但是最大只能是32以内），另外5bit是你传递进来的机器id（最大32以内）；剩下的12bit序列号（如果跟你上次生成id的事件还在一个毫秒内，那么会把顺序给你累加，最多在4096个序号以内）。

所以你自己利用这个工具类，自己搞一个服务，然后对每个机房的每个机器都初始化这么一个东西，刚开始这个机房的这个机器的序号就是0。然后每次接收到一个请求，说这个机房的这个机器要生成一个id，你就找到对应的Worker生成。

利用这个snowflake算法，你可以开发自己公司的服务，甚至对于机房id和机器id，反正给你预留了5bit + 5bit，换成别的有业务含义的东西也可以。

分布式id生成，如果是高并发，那么用这个snowflake算法性能比较好，一般美妙几万并发的场景。