链滴

架构 - 一款永不重复的高性能分布式发号器

作者：someone33881

原文链接：https://ld246.com/article/1542543818923

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180225.jpg?imageView2/1/w/960/h/520/interlace/1/q/100">
<h3 id="零-基本术语">零、基本术语</h3>
    发号器：用于生成唯一流水号（也即俗称的唯一 ID）的
务系统，称之为发号器
<h3 id="一-技术选型">一、技术选型</h3>
UUID
    优点：能够保证唯一性
    缺点：（1）完全的时间数据=> 性能比较差、比较长
占用空间大、间接导致数据库性能下降；（2）无序=> 导致 B+ 树
引在写的时候会有过多的随机写操作，不会产生有顺序的 append 操作，而是需要进行 in
ert 操作，这将读取整个 B+ 树节点到内存并在插入该条记录后会将整个节点写会磁盘=> 在记录
用空间比较大的情况下，**写的性能会明显下降**
数据库
     单库时（自增字段）：局限性在于自增
段完全依赖数据库，会导致数据库移植、扩容、洗数据、分库分表
题
    分库分表时（水平伸缩=> 自增字段 + 数据
seq+ 步长 step）：缺陷在于服务节点固定（即 step 固定，继续
加服务节点难以进行扩展）、仍强依赖于数据库（对数据库造成压力）
开源项目-Snowflake
    Twitter 开源的发号器，缺点在于文档简单、发布模式单一、缺少支持和维护</
>
=> 自研：一款通用、高性能发号器产品，具有“全局唯一、粗略有序、可反解、可制造”特
，三种发布模式“嵌入发布模式(jar 包)、中心服务器发布模式(服务化场景)、REST 发布模式(rest api
接口)”
<h3 id="二-基本需求">二、基本需求</h3>
    全局唯一：一般悲观策略是使用锁或分布式锁（性能大大
低），而这里将利用时间的有序性，在时间的某个单元下采用自增序列达到全
唯一
    粗略有序：完全有序则涉及到数据的汇聚，需要用到锁或
布式锁，考虑到效率问题采用折中方案即粗略有序，将支持秒级有序和毫秒级有序两种方式
    可反解：ID 具有时间属性且可反解其他信息量，节省空间
/p>
    可制造：即支持手工处理，可复制、可恢复、可制造
    高性能：ID 生成取决于网络 I/O 和 CPU 的性能，网络 I/
一般不是瓶颈，根据经验单台机器 TPS 能达到 10000/s
    高可用：对等集群、重试机制、本地容错方案（即本地依
）
    可伸缩：水平伸缩，支持业务量增长
<h3 id="三-核心设计">三、核心设计</h3>
发布模式
    嵌入发布模式：仅限 java 客户端，通过嵌入 jar 包式的原生服务，需提前配置
地机器 ID
    中心服务器发布模式：仅限 java 客户端，提供一个服务的客户端 jar 包，java
序像调用本地 api 一样调用，但依赖于中心的 ID 产生服务器
    REST 发布模式：中心服务器通过 Restful API 导出服务，非 java 客户端可使用
/p>
ID 类型
    最大峰值型：秒级有序，秒级时间占用 30 位，序列号占用 20 位
<table>
<thead>
<tr>
<th>字段</th>

<th>版本</th>
<th>类型</th>
<th>生成方式</th>
<th>秒级时间</th>
<th>序列号</th>
<th>机器 ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>位数</td>
<td>63</td>
<td>62</td>
<td>60-61</td>
<td>30-59</td>
<td>10-29</td>
<td>0-9</td>
</tr>
</tbody>
</table>
    最小粒度型：秒级有序，秒级时间占用 40 位，序列号占用 10 位
<table>
<thead>
<tr>
<th>字段</th>
<th>版本</th>
<th>类型</th>
<th>生成方式</th>
<th>秒级时间</th>
<th>序列号</th>
<th>机器 ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>位数</td>
<td>63</td>
<td>62</td>
<td>60-61</td>
<td>20-59</td>
<td>10-19</td>
<td>0-9</td>
</tr>
</tbody>
</table>
数据结构
    版本：1 位，用做扩展位或扩容时的临时方案，默认值 0，1 则表示扩展或扩容
    ID 类型：1 位，0-最大峰值型，1-最小粒度型
    生成方式：2 位，00-嵌入发布模式，01-中心发布模式，02（即二进制 10）-RST 发布模式，03（即二进制 11）-保留未用
    时间：最大峰值型-秒级时间 30 位，2^30/60/60/24/365=34 年；最小粒度型毫秒级时间 40 位，2^40/1000/60/60/24/365=34 年
    序列号：最大峰值型 20 位，理论上每秒可最多产生 2^20=1048576 个 ID，

百万级别；最小粒度型 10 位，理论上每毫秒最多可产生 2^10=1024 个 ID，=> 最大 1024000/

    机器 ID：10 位，即最多支持 1024 个服务器，中心发布模式和 REST 发布模式般不会有太多数量机器，按照设计每台 TPS 为 1 万/s，10 台服务器则可达到 10 万/s，已经可以满绝大部分需求；而考虑到内嵌发布模式，对机器需求量很大，因此最多支持 1024 个服务器
并发
    中心服务器和 REST 发布模式，开销主要涉及网络 I/O 和 CPU 操作，ID 生成本上是内存到高速缓存操作，没有磁盘 I/O 操作，网络 I/O 是系统瓶颈；但相对于网络 I/O，CPU 算速度更是瓶颈，因此 ID 产生的服务采用多线程方式
    多种实现方式，实现 ID 生成过程中的竞争点 time 和 sequence：
    （1）使用 concurrent 包的 ReentrantLock 进行互斥，默认实现方式，追求能和稳定的折中方案
    （2）使用传统的 synchronized 进行互斥，性能较（1）稍逊色一些，通过传入 JVM 参数-Dvesta.sync.lock.impl.key=true 开启
    （3）使用 concurrent 包的原子变量进行互斥，性能非常高，但是高并发环境下 CPU 负载会很高，通过传入 JVM 参数-Dvesta.atomic.lock.impl.key=true 开启
机器 ID 的分配
    0-923：嵌入发布模式，预先配置机器 ID，最多支持 924 台内嵌服务器
    924-1023：中心服务器发布模式和 REST 发布模式，最多支持 100 台，最大持 100*1 万/s 即 100 万/s 的 TPS
=> 可以动态调整两个区间的值来适应，当然各个垂直业务具有天生的隔离性，每个业务都可适应最多 1024 台服务器的
=>3 种机器 ID 的分配方式：共享数据库方式、IP 方式、Spring 配置文件方式(测试时使用)<
p>
时间同步
    crontab，周期性地通过时间服务器虚拟集群核准服务器时间：ntpdate -u pool ntp.orgpool.ntp.org
    未重启机器调慢时间，Vesta 抛出异常拒绝产生 ID;重启机器调快时间，调整后常产生 ID；重启机器调慢时间，可能会产生重复的 ID，系统管理员需要保证避免这种情况；每 4 年次同步闰秒，由于是调快 1 秒，因此调整后不影响 ID 的产生
设计验证
（1）根据不同信息分段构建一个 ID，且 ID 全局唯一、可反解、可制造等
（2）使用秒级时间或毫秒级时间及时间单元内序列递增的方法保证 ID 粗略有序
（3）多线程处理中心服务器发布模式和 REST 发布模式，三种实现方式解决竞争点的互斥
（4）保证发布模式的最大可用，即某种模式不可用，可以回退到使用本地预配的机器 ID
（5）机器号分两个区段，一个从 0 开始向上，一个从 1024 开始向下，并且能够动态调整分界，满足可伸缩性
<h3 id="四-代码实现">四、代码实现</h3>
    代理模式
【读书系列】
    《可伸缩服务架构：框架与中间件》，李艳鹏、杨彪、李海亮、贾博岩、刘湜电子工业出版社