分布式全局ID

# 特性需求

1. 全局唯一

2. 支持高并发

3. 能够体现一定属性

4. 高可靠，容错单点故障

5. 高性能

# 常用方案

## 利用数据库递增，全数据库唯一。

优点：明显，可控。

缺点：单库单表，数据库压力大。

### 自增长实现

mysql>

SHOW VARIABLESLIKE'auto\_inc%';

| Variable\_name | Value |

| auto\_increment\_increment | 1 |  //查看步长

| auto\_increment\_offset | 1 |

2rowsinset(0.00sec)

mysql>

SET@@auto\_increment\_increment=10;  //设置步长

Query OK, 0rows affected (0.00sec)

mysql>

SHOW VARIABLESLIKE'auto\_inc%';

| Variable\_name | Value |

| auto\_increment\_increment | 10 |    //已设置为10

| auto\_increment\_offset | 1 |

2rowsinset(0.01sec)

数据表自增将以10为间隔自增

如果以上方法还不生效，或者重启mysql后，又变回来了。那肯定是在my.cnf里面设置了全局变量。这个必须到配置文件里面去修改了；这种修改永久有效。而且无法通过上面的操作再次被修改。

用vi编辑器打开配置文件，默认位置

#vi /etc/my.cnf

找到 auto\_increment\_increment 变量设置的地方；VI里面可以用 "/auto\_increment\_increment" 找到。

找到后设置

auto\_increment\_increment=1;即可

其他全局变量也可在此配置文件里面永久设置好。

配置文件修改好后，要重启mysql服务才会生效。

#/etc/init.d/mysql restart

## UUID，

生成的是length=32的16进制格式的字符串，如果回退为byte数组共16个byte元素，即UUID是一个128bit长的数字，一般用16进制表示。

优点：对数据库压力减轻了。

缺点：但是排序怎么办？

此外还有UUID的变种，增加一个时间拼接，但是会造成id非常长。

## 雪花算法

twitter在把存储系统从MySQL迁移到Cassandra的过程中由于Cassandra没有顺序ID生成机制，于是自己开发了一套全局唯一ID生成服务：Snowflake。

1. 41位的时间序列（精确到毫秒，41位的长度可以使用69年）  
2. 10位的机器标识（10位的长度最多支持部署1024个节点）   
3. 12位的计数顺序号（12位的计数顺序号支持每个节点每毫秒产生4096个ID序号）

最高位是符号位，始终为0。

优点：高性能，低延迟；独立的应用；按时间有序。

缺点：需要独立的开发和部署。

## Redis生成ID

当使用数据库来生成ID性能不够要求的时候，我们可以尝试使用Redis来生成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。可以用Redis的原子操作INCR和INCRBY来实现。

可以使用Redis集群来获取更高的吞吐量。假如一个集群中有5台Redis。可以初始化每台Redis的值分别是1,2,3,4,5，然后步长都是5。各个Redis生成的ID为：

A：1,6,11,16,21

B：2,7,12,17,22

C：3,8,13,18,23

D：4,9,14,19,24

E：5,10,15,20,25

比较适合使用Redis来生成每天从0开始的流水号。比如订单号=日期+当日自增长号。可以每天在Redis中生成一个Key，使用INCR进行累加。

优点：

不依赖于数据库，灵活方便，且性能优于数据库。

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：

如果系统中没有Redis，还需要引入新的组件，增加系统复杂度。

需要编码和配置的工作量比较大，多环境运维很麻烦，

在开始时，程序实例负载到哪个redis实例一旦确定好，未来很难做修改。

package com.imooc.activiti.springboot;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.redis.core.RedisTemplate;  
import org.springframework.stereotype.Service;  
  
import java.util.Calendar;  
import java.util.Date;  
  
*/\*\*  
 \** ***@author*** *syliu  
 \* 利用redis生成数据库全局唯一性id  
 \*/*@Service  
public class PrimaryKeyService {  
  
 @Autowired  
 private RedisTemplate redisTemplate;  
  
  
*/\*\*  
 \* 获取年的后两位加上一年多少天+当前小时数作为前缀  
 \** ***@param*** *date  
 \** ***@return*** *\*/* public String getOrderIdPrefix(Date date) {  
 Calendar calendar = Calendar.*getInstance*();  
 calendar.setTime(date);  
 int year = calendar.get(Calendar.*YEAR*);  
 int month = calendar.get(Calendar.*MONTH*);  
 int day = calendar.get(Calendar.*DAY\_OF\_MONTH*);  
 int hour = calendar.get(Calendar.*HOUR\_OF\_DAY*);  
 //补两位,因为一年最多三位数  
 System.*out*.println("格式化之前的月： "+month);  
 String monthFormat = String.*format*("%1$02d", month+1);  
 System.*out*.println("格式化之后的月： "+monthFormat);  
 //补两位，因为日最多两位数  
 System.*out*.println("格式化之前的日 ： "+monthFormat);  
 String dayFormat = String.*format*("%1$02d", day);  
 System.*out*.println("格式化之后的日 ： "+monthFormat);  
 //补两位，因为小时最多两位数  
 System.*out*.println("格式化之前的小时： "+monthFormat);  
 String hourFormat = String.*format*("%1$02d", hour);  
 System.*out*.println("格式化之后的小时： "+monthFormat);  
 return year + monthFormat + dayFormat+hourFormat;  
 }  
  
 */\*\*  
 \* 生成订单  
 \** ***@param*** *prefix  
 \** ***@return*** *\*/* public Long orderId(String prefix) {  
 String key = "DEMO\_ORDER\_ID\_" + prefix;  
 String orderId = null;  
 try {  
 Long increment = redisTemplate.opsForValue().increment(key,1);  
 //往前补6位  
 orderId=prefix+String.*format*("%1$06d",increment);  
 } catch (Exception e) {  
 System.*out*.println("生成订单号失败");  
 e.printStackTrace();  
 }  
 return Long.*valueOf*(orderId);  
 }  
}

## Flicker的解决方案

因为MySQL本身支持auto\_increment操作，很自然地，我们会想到借助这个特性来实现这个功能。

Flicker在解决全局ID生成方案里就采用了MySQL自增长ID的机制（auto\_increment + replace into + MyISAM）。

## 还有其他一些方案，

比如京东淘宝等电商的订单号生成。因为订单号和用户id在业务上的区别，订单号尽可能要多些冗余的业务信息，比如：

滴滴：时间+起点编号+车牌号

淘宝订单：时间戳+用户ID

其他电商：时间戳+下单渠道+用户ID，有的会加上订单第一个商品的ID。

而用户ID，则要求含义简单明了，包含注册渠道即可，尽量短。