ZK+SnowFlake每秒生成100W主键ID

面试题：分布式id 主键如何处理?

考点分析

其实这是分库分表之后你必然要面对的一个问题，就是 id 咋生成?

> 要求：既要数量（每秒百万），又要质量（连续递增）

基础知识：有些分布式的ID生成器方案呢?

《java高并发核心编程 卷1》

为什么不建议使用uuid作为主键?

分布式数据库当然也有主键的需求，但是为什么不直接使用uuid作为主键呢?

1. UUID生成速率低下

Java的UUID依赖于SecureRandom.nextBytes方法，而SecureRandom又依赖于操作系统提供的随机数源，在Linux系统下，它的默认依赖是/dev/random，而这个源是阻塞的。最可怕的是，这个nextBytes方法还是一个synchronized方法，也就是说，如果多线程调用UUID，生成速率不升反降。

测试结果：在一台64线程的服务器上，调用UUID.randomUUID方法，生成一千万个uuid平均耗时在130s，tps不到8w

2. UUID主键在innodb中会引发性能问题

a. innodb中的主键索引也是聚集索引，如果插入的数据是**顺序的**，那么b+树的叶子基本都是满的，缓存也可以很好的发挥作用。如果插入的数据是完全无序的，那么叶子节点会**频繁分裂**，缓存也基本无效了。这会减少tps

b. uuid占用的空间较大

3. UUID完全没有意义，如果有一个主键是全局自增的，那么数据排列顺序就是数据的插入顺序

ZooKeeper+SnowFlake ID 生成器

ZooKeeper生成节点ID

原理+源码

> 问题：为啥不用zk，生成数据库id？ 性能问题。
>
> zk是cp，强一致性的分布式系统，创建一个znode，只有所有的slave节点都创建了znode节点，该znode节点才是有效的, so，导致吞吐量很低，不适于每秒10W级甚至100W级别的ID生成场景。
>
> 但是，ZooKeeper 生成分布式系统的节点id，完全没有问题。而且，分布式系统节点id，本身就需要强一致性，需要的就是cp，而不是AP。

SnowFlake 雪花算法

原理+源码

封装成为Hibernate ID生成器

Hibernate ID生成器的基类

```
CommonSnowflakeIdGenerator  extends IncrementGenerator
```

以上的仅仅是对ZooKeeper+Snowflake算法分布式ID生成器的简单封装，

Hibernate ID生成器的使用

```
@Builder
public class SeckillSegmentStockPO implements Serializable
{
    //商品ID
    @Id

    @GenericGenerator(
            name = "snowflakeIdGenerator",
            strategy =
"com.crazymaker.springcloud.standard.hibernate.CommonSnowflakeIdGenerator")
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator =
"snowflakeIdGenerator")
    @Column(name = "SEG_STOCK_ID", unique = true, nullable = false, length = 8)
    private Long id;
```