# 第六节课交易分库分表详解

# 分库分表难点：

分布式事务。

**分布式主键。**

跨库的查询。

数据迁移的问题。

## 分库分表之分布式唯一ID解决方案：

## Uuid:

通用唯一识别码

组成部分：当前日期和时间+时钟序列+全局唯一网卡mac地址获取

执行任务数：10000

------------------------

所有线程共耗时：91.292 s

并发执行完耗时：1.221 s

单任务平均耗时：9.1292 ms

单线程最小耗时：0.0 ms

单线程最大耗时：470.0 ms

**优点：**

代码实现简单、不占用宽带、数据迁移不影响

**缺点：**

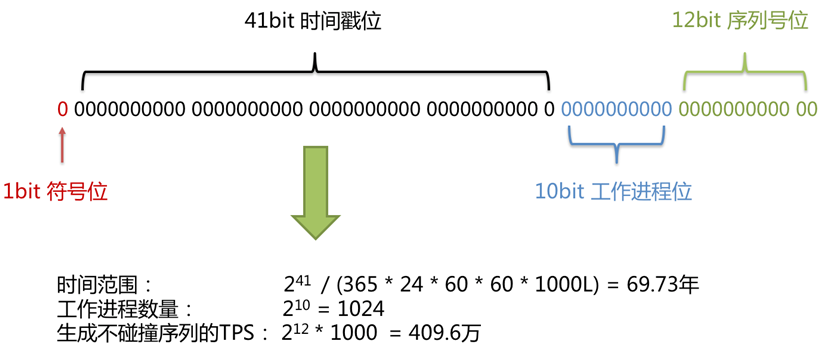
无序、无法保证趋势递增、字符存储、传输、查询慢

### Snowflke:

snowflake是Twitter开源的分布式ID生成算法。

传统数据库软件开发中，主键自动生成技术是基本需求。而各个数据库对于该需求也提供了相应的支持，比如MySQL的自增键，Oracle的自增序列等。 数据分片后，不同数据节点生成全局唯一主键是非常棘手的问题。同一个逻辑表内的不同实际表之间的自增键由于无法互相感知而产生重复主键。 虽然可通过约束自增主键初始值和步长的方式避免碰撞，但需引入额外的运维规则，使解决方案缺乏完整性和可扩展性。

io.shardingsphere.core.keygen.DefaultKeyGenerator



执行任务数：10000

------------------------

所有线程共耗时：15.111 s

并发执行完耗时：217.0 ms

单任务平均耗时：1.5111 ms

单线程最小耗时：0.0 ms

单线程最大耗时：97.0 ms

**优点：**

不占用宽带、本地生成、高位是毫秒，低位递增

**缺点：**

强依赖时钟，如果时间回拨，数据递增不安全

### Mysql:

利用数据库的步长来做的。

CREATE TABLE `bit\_table` (

`id` varchar(255) NOT NULL,//字符

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `bit\_num` (

`id` bigint(11) NOT NULL AUTO\_INCREMENT,

KEY (`id`) USING BTREE

) ENGINE=InnoDB auto\_increment=1 DEFAULT CHARSET=utf8;

SET global auto\_increment\_offset=1; --初始化

SET global auto\_increment\_increment=100; --初始步长

show global variables;

**缺点:**

受限数据库、单点故障、扩展麻烦

**优点：**

性能可以、可读性强、数字排序

### Redis:

redis原子性：对存储在指定key的数值执行原子的加1操作。如果指定的key不存在，那么在执行incr操作之前，会先将它的值设定为0

组成部分：年份+当天当年第多少天+天数+小时+redis 自增

执行任务数：10000

------------------------

所有线程共耗时：746.767 s

并发执行完耗时：9.381 s

单任务平均耗时：74.6767 ms

单线程最小耗时：0.0 ms

单线程最大耗时：4.119 s

**优点：**

有序递增、可读性强、符合刚才我们那个扩容方案的id

**缺点：**

占用宽带（网络）、依赖第三方、redis

一个小时内生产99万的订单 ？

### 场景：

|  |  |  |
| --- | --- | --- |
| **名称** | **场景** | **适用指数** |
| Uuid | Token、图片id | ★★ |
| Snowflake | ELK、MQ、业务系统 | ★★★★ |
| 数据库 | 非大型电商系统 | ★★★ |
| Redis | 大型系统 | ★★★★★ |

Dubbo 序列化bug

rango147