目录

[分布式锁 1](#_Toc10402162)

[数据库的唯一索引 1](#_Toc10402163)

[缓存实现分布式锁 1](#_Toc10402164)

[使用redis加锁与解锁 1](#_Toc10402165)

[基于ZooKeeper的分布式锁 3](#_Toc10402166)

# 分布式锁

在单体应用中，可以使用语言的内置锁来实现进程同步。但在分布式场景，需要同步的进程可能位于不同的节点上，那么就需要使用分布式锁。

## 数据库的唯一索引

通过利用数据库表唯一索引原理，获取锁时向数据库插入一条记录，删除该记录就释放锁，对于现在集群环境处理批扣数据原理一致，一台服务器进行数据跑批时，其他服务器不进行处理，主要通过表中唯一约束来实现。

缺点:

1. 锁没有失效时间，删除数据失败的话，会导致其他服务无法获取锁；
2. 非阻塞锁，获取锁失败时直接返回，不会进行等待；(使用数据库排它锁可实现阻塞，利用for update语句)
3. 不可重入，已经获取锁的进程必须重新插入记录来获得锁；

## 缓存实现分布式锁

redis，memcached等缓存实现分布式锁；

eg:

### 使用redis加锁与解锁

public class RedisTool {

private static final String LOCK\_SUCCESS = "OK";

private static final String SET\_IF\_NOT\_EXIST = "NX";

private static final String SET\_WITH\_EXPIRE\_TIME = "PX";

private static final Long RELEASE\_SUCCESS = 1L;

/\*\*

\* 尝试获取分布式锁

\* @param jedis Redis客户端

\* @param lockKey 锁

\* @param requestId 请求标识

\* @param expireTime 超期时间

\* @return 是否获取成功

\*/

public static boolean tryGetDistributedLock(Jedis jedis, String lockKey, String requestId, int expireTime) {

String result = jedis.set(lockKey, requestId, SET\_IF\_NOT\_EXIST, SET\_WITH\_EXPIRE\_TIME, expireTime);

if (LOCK\_SUCCESS.equals(result)) {

return true;

}

return false;

}

/\*\*

\* 释放分布式锁

\* @param jedis Redis客户端

\* @param lockKey 锁

\* @param requestId 请求标识

\* @return 是否释放成功

\*/

public static boolean releaseDistributedLock(Jedis jedis, String lockKey, String requestId) {

String script = "if redis.call('get', KEYS[1]) == ARGV[1] then return redis.call('del', KEYS[1]) else return 0 end";

Object result = jedis.eval(script, Collections.singletonList(lockKey), Collections.singletonList(requestId));

if (RELEASE\_SUCCESS.equals(result)) {

return true;

}

return false;

}

}

## 基于ZooKeeper的分布式锁

参考<https://blog.51cto.com/zero01/2107174>