**Java分布式锁的三种实现方案**

**http://www.jb51.net/article/103617.htm**

## **方案一：数据库乐观锁**

乐观锁通常实现基于数据版本(version)的记录机制实现的，比如有一张红包表（t\_bonus），有一个字段(left\_count)记录礼物的剩余个数，用户每领取一个奖品，对应的left\_count减1，在并发的情况下如何要保证left\_count不为负数，乐观锁的实现方式为在红包表上添加一个版本号字段（version），默认为0。

|  |
| --- |
| 异常  -- 可能会发生的异常情况  -- 线程1查询，当前left\_count为1，则有记录  select \* from t\_bonus where id = 10001 and left\_count > 0    -- 线程2查询，当前left\_count为1，也有记录  select \* from t\_bonus where id = 10001 and left\_count > 0    -- 线程1完成领取记录，修改left\_count为0,  update t\_bonus set left\_count = left\_count - 1 where id = 10001    -- 线程2完成领取记录，修改left\_count为-1，产生脏数据  update t\_bonus set left\_count = left\_count - 1 where id = 10001 |
| 正常  -- 添加版本号控制字段  ALTER TABLE table ADD COLUMN version INT DEFAULT '0' NOT NULL AFTER t\_bonus;    -- 线程1查询，当前left\_count为1，则有记录，当前版本号为1234  select left\_count, version from t\_bonus where id = 10001 and left\_count > 0    -- 线程2查询，当前left\_count为1，有记录，当前版本号为1234  select left\_count, version from t\_bonus where id = 10001 and left\_count > 0    -- 线程1,更新完成后当前的version为1235，update状态为1，更新成功  update t\_bonus set version = 1235, left\_count = left\_count-1 where id = 10001 and version = 1234    -- 线程2,更新由于当前的version为1235，udpate状态为0，更新失败，再针对相关业务做异常处理  update t\_bonus set version = 1235, left\_count = left\_count-1 where id = 10001 and version = 1234 |

## **方案二：基于Redis的分布式锁**

SETNX命令（SET if Not eXists）\  
语法：SETNX key value\  
功能：原子性操作，当且仅当 key 不存在，将 key 的值设为 value ，并返回1；若给定的 key 已经存在，则 SETNX 不做任何动作，并返回0。\  
Expire命令\  
语法：expire(key, expireTime)\  
功能：key设置过期时间\  
GETSET命令\  
语法：GETSET key value\  
功能：将给定 key 的值设为 value ，并返回 key 的旧值 (old value)，当 key 存在但不是字符串类型时，返回一个错误，当key不存在时，返回nil。\  
GET命令\  
语法：GET key\  
功能：返回 key 所关联的字符串值，如果 key 不存在那么返回特殊值 nil 。\  
DEL命令\  
语法：DEL key [KEY …]\  
功能：删除给定的一个或多个 key ,不存在的 key 会被忽略。

**第一种：使用redis的setnx()、expire()方法，用于分布式锁**

1. setnx(lockkey, 1) 如果返回0，则说明占位失败；如果返回1，则说明占位成功
2. expire()命令对lockkey设置超时时间，为的是避免死锁问题。
3. 执行完业务代码后，可以通过delete命令删除key。

这个方案其实是可以解决日常工作中的需求的，但从技术方案的探讨上来说，可能还有一些可以完善的地方。比如，如果在第一步setnx执行成功后，在expire()命令执行成功前，发生了宕机的现象，那么就依然会出现死锁的问题

**第二种：使用redis的setnx()、get()、getset()方法，用于分布式锁，解决死锁问题**

1. setnx(lockkey, 当前时间+过期超时时间) ，如果返回1，则获取锁成功；如果返回0则没有获取到锁，转向2。
2. get(lockkey)获取值oldExpireTime ，并将这个value值与当前的系统时间进行比较，如果小于当前系统时间，则认为这个锁已经超时，可以允许别的请求重新获取，转向3。
3. 计算newExpireTime=当前时间+过期超时时间，然后getset(lockkey, newExpireTime) 会返回当前lockkey的值currentExpireTime。
4. 判断currentExpireTime与oldExpireTime 是否相等，如果相等，说明当前getset设置成功，获取到了锁。如果不相等，说明这个锁又被别的请求获取走了，那么当前请求可以直接返回失败，或者继续重试。
5. 在获取到锁之后，当前线程可以开始自己的业务处理，当处理完毕后，比较自己的处理时间和对于锁设置的超时时间，如果小于锁设置的超时时间，则直接执行delete释放锁；如果大于锁设置的超时时间，则不需要再锁进行处理。

## 第三种方案：基于Zookeeper的分布式锁

1. **利用节点名称的唯一性来实现独占锁**
2. ZooKeeper机制规定同一个目录下只能有一个唯一的文件名，zookeeper上的一个znode看作是一把锁，通过createznode的方式来实现。所有客户端都去创建/lock/${lock\_name}\_lock节点，最终成功创建的那个客户端也即拥有了这把锁，创建失败的可以选择监听继续等待，还是放弃抛出异常实现独占锁。

**利用临时顺序节点控制时序实现**