锁的释放和获取先后关系

```
    }
    Boolean lock=false;
    try {
         lock = distributedLockService.lock(LockType.REDIS.getCode(), LockGroupType.UpgradePaasMetaData.getCode(), tenantOid.toString(),  expireTime:
        if (!lock) {
            return;
        }
        try {
            TriggerContext triggerContext = new TriggerContext();
            triggerContext.setOidList(Arrays.asList(Long.valueOf(tenantOid)));
            TenantService tenantService = KingKongContext.getBean(TenantService.class);
            tenantService.paasAppUpdate(triggerContext);
        } catch (Exception e) {
            log.error("", e);
        }
        tenant.set(TenantMetaInfo.status, KkTenantStatus.Active.getValue());
        tenant.set(TenantMetaInfo.unActiveOn, null);
        sObjectService.saveWithValidation(tenant);
        // 清理租户缓存
        if (tenant != null) {
            tenantDao.removeTenantCache(tenant);
        }
    } catch (Exception e) {
        log.error("", e);

    } finally {
        if(lock){
            distributedLockService.unLock(LockType.REDIS.getCode(), LockGroupType.UpgradePaasMetaData.getCode(), tenantOid.toString());
        }
    }
}
```

正确用法

下面错误用法，在这个错误用法中，即便lock获取锁失败了 会执行return ‘，但是执行return之前会执行finally 导致 释放了 redis锁，实际上 当前线程并没有获取到锁，他是释放了别人的锁

```
    }

    try {
        Boolean lock = distributedLockService.lock(LockType.REDIS.getCode(),
        if (!lock) {
            return;
        }
        try {
            TriggerContext triggerContext = new TriggerContext();
            triggerContext.setOidList(Arrays.asList(Long.valueOf(tenantOid)));
            TenantService tenantService = KingKongContext.getBean(TenantServi
            tenantService.paasAppUpdate(triggerContext);
        } catch (Exception e) {
            log.error("", e);
        }
        tenant.set(TenantMetaInfo.status, KkTenantStatus.Active.getValue());
        tenant.set(TenantMetaInfo.unActiveOn, null);
        sObjectService.saveWithValidation(tenant);
        // 清理租户缓存
        if (tenant != null) {
            tenantDao.removeTenantCache(tenant);
        }
    } catch (Exception e) {
        log.error("", e);

    } finally {
        distributedLockService.unLock(LockType.REDIS.getCode(), LockGroupType
    }
```