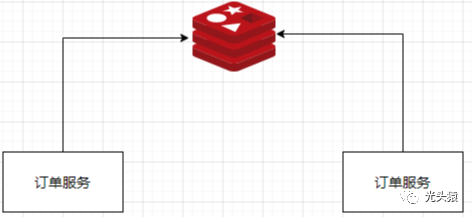
# 1、什么是分布式锁？

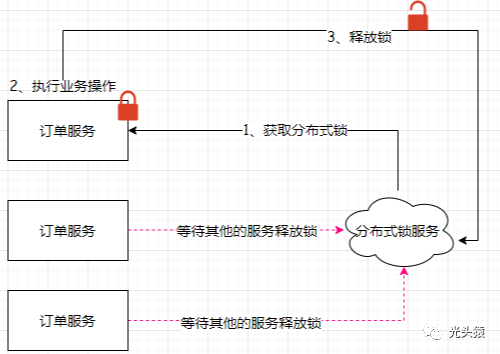
概述：在分布式系统中，多个线程访问共享数据就会出现数据安全性的问题。而由于jdk中的锁要求多个线程在同一个jvm中，因此在分布式系统中无法使用jdk中的锁保证数据的安全性，那么此时就需要使用分布式锁。

作用：可以保证在分布式系统中多个线程访问共享数据时数据的安全性

举例：在电商系统中，用户在进行下单操作的时候需要扣减库存。为了提高下单操作的执行效率，此时需要将库存的数据存储到Redis中。订单服务每一次生成订单之前需要查询一下库存数据，如果存在则生成订单同时扣减库存。在高并发场景下会存在多个订单服务操作Redis，此时就会出现线程安全问题。



分布式锁的工作原理：



分布式锁应该具备哪些条件：

1、在分布式系统环境下，一个方法在同一时间只能被一个机器的一个线程执行

2、高可用的获取锁与释放锁

3、高性能的获取锁与释放锁

4、具备可重入特性

5、具备锁失效机制，防止死锁

可重入特性：获取到锁的线程再次调用需要锁的方法的时候，不需要再次获取锁对象。

使用场景：遍历树形菜单的时候的递归调用。

# 2、分布式锁的实现方案都有哪些？

分布式锁的实现方案：

1、数据库

2、zookeeper

3、redis

# 3、Redis怎么实现分布式锁思路？

Redis实现分布式锁主要利用Redis的setnx命令。setnx是SET if not exists(如果不存在，则 SET)的简写。

127.0.0.1:6379> setnx lock value1 #在键lock不存在的情况下，将键key的值设置为value1(integer) 1127.0.0.1:6379> setnx lock value2 #试图覆盖lock的值，返回0表示失败(integer) 0127.0.0.1:6379> get lock #获取lock的值，验证没有被覆盖"value1"127.0.0.1:6379> del lock #删除lock的值，删除成功(integer) 1127.0.0.1:6379> setnx lock value2 #再使用setnx命令设置，返回0表示成功(integer) 1127.0.0.1:6379> get lock #获取lock的值，验证设置成功"value2"

上面这几个命令就是最基本的用来完成分布式锁的命令。

加锁：使用setnx key value命令，如果key不存在，设置value(加锁成功)。如果已经存在lock(也就是有客户端持有锁了)，则设置失败(加锁失败)。

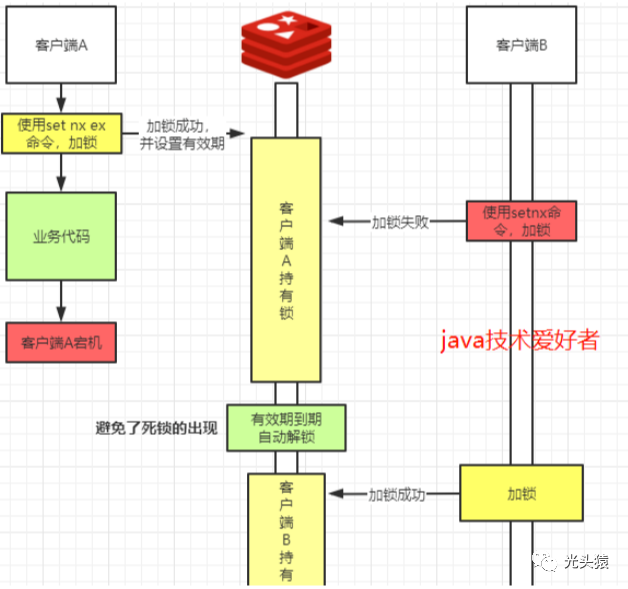
解锁：使用del命令，通过删除键值释放锁。释放锁之后，其他客户端可以通过setnx命令进行加锁。

# 4、Redis实现分布式锁如何防止死锁现象？

产生死锁的原因：如果一个客户端持有锁的期间突然崩溃了，就会导致无法解锁，最后导致出现死锁的现象。如下所示：

解决思路：在设置key的值时，需要加上有效时间，如果有效时间过期了，就会自动失效，就不会出现死锁。然后加锁的代码就会变成这样。

执行流程如下所示：



# 5、Redis实现分布式锁如何合理的控制锁的有效时长？

有效时间设置多长，假如我的业务操作比有效时间长？我的业务代码还没执行完就自动给我解锁了，不就完蛋了吗。

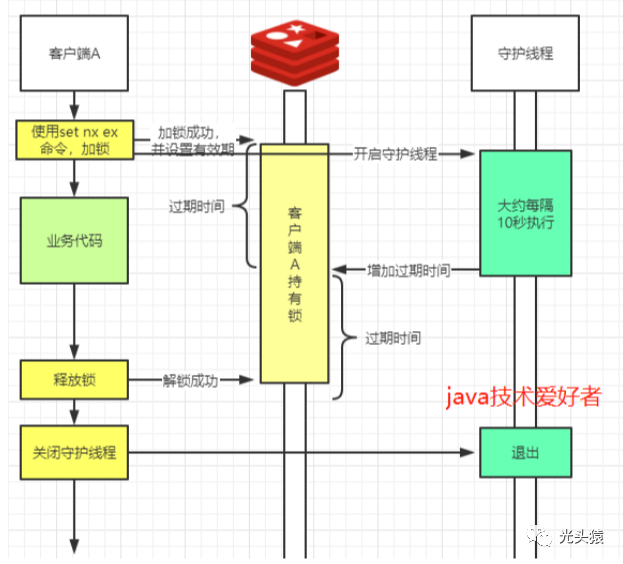
解决方案：

1、第一种：程序员自己去把握，预估一下业务代码需要执行的时间，然后设置有效期时间比执行时间长一些，保证不会因为自动解锁影响到客户端业务代码的执行。

2、第二种：给锁续期。

锁续期实现思路：当加锁成功后，同时开启守护线程，默认有效期是用户所设置的，然后每隔10秒就会给锁续期到用户所设置的有效期，只要持有锁的客户端没有宕机，就能保证一直持有锁，直到业务代码执行完毕由客户端自己解锁，如果宕机了自然就在有效期失效后自动解锁。

如下所示：



上述的第二种解决方案可以使用redis官方所提供的Redisson进行实现。

Redisson是Redis官方推荐的Java版的Redis客户端。它提供的功能非常多，也非常强大分布式服务，使用Redisson可以轻松的实现分布式锁。Redisson中进行锁续期的这种机制被称为"看门狗"机制。

# 6、Redis实现分布式锁如何保证锁服务的高可用？

解决方案：

1、使用Redis的哨兵模式构建一个主从架构的Redis集群

2、使用Redis Cluster集群

# 7、当同步锁数据到从节点之前，主节点宕机了导致锁失效，那么此时其他线程就可以再次获取到锁，这个问题怎么解决？

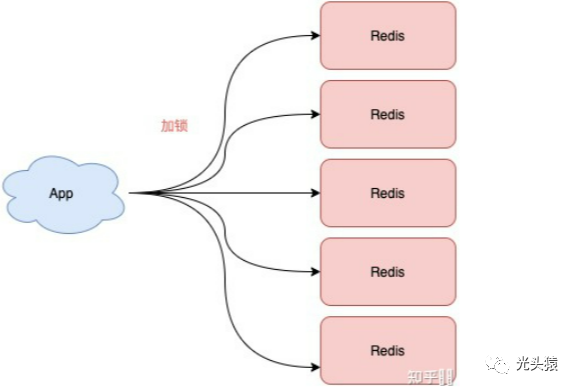
使用Redission框架中的RedLock进行处理。

RedLock的方案基于2个前提：

1、不再需要部署从库和哨兵实例，只部署主库

2、但主库要部署多个，官方推荐至少5个实例

也就是说，想使用RedLock，你至少要部署5个Redis实例，而且都是主库，它们之间没有任何关系，都是一个个孤立的实例。



工作流程如下所示：

1、客户端先获取【当前时间戳T1】

2、客户端依次向这个5个Redis实例发起加锁请求，且每个请求会设置超时时间(毫秒级，要远小于锁的有效时间)，如果某一个实例加锁失败(包括网络超时，锁被其他的人持有等各种异常情况)，就立即向下一个Redis实例申请加锁

3、如果客户端从 >=3 个(大多数)以上Redis实例加锁成功，则再次获取【当前时间戳T2】, 如果 T2 - T1 < 锁的过期时间，此时，认为客户端加锁成功，否则加锁失败

4、加锁成功，去操作共享资源

5、加锁失败，向【全部节点】发起释放锁请求

总结4个重点：

1、客户端在多个Redis实例上申请加锁

2、必须保证大多数节点加锁成功

3、大多数节点加锁的总耗时，要小于锁设置的过期时间

4、锁释放，要向全部节点发起释放锁请求

**7.1 为什么要在多个实例上加锁？**

本质上是为了【容错】, 部分实例异常宕机，剩余的实例加锁成功，整个锁服务依旧可用。

**7.2 为什么步骤3加锁成功后，还要计算加锁的累计耗时？**

因为操作的是多个节点，所以耗时肯定会比操作单个实例耗时更久，而且，因为是网络请求，网络情况是复杂的，有可能存在延迟、丢包、超时等情况发生，网络请求越多，异常发生的概率就越大。

所以，即使大多数节点加锁成功，如果加锁的累计耗时已经超过了锁的过期时间，那此时有些实例上的锁可能已经失效了，这个锁就没有意义了。