1. TPS、QPS、系统吞吐量的区别和理解
2. TPS:

Transaction Per Second,即每秒处理的事务数。一个事务是指客户端向服务器发送请求后服务器做出响应的过程，从发送请求开始到收到响应结果结束，以此来计算使用的时间和完成的事务数量,可以是一个接口、多个接口、一个业务流程等等。

如果每秒能够完成n次一个事务，TPS就是n。

1. QPS:

Queries Per Second,即每秒查询率，是一台服务器每秒能够响应的查询次数(数据库中的每秒执行查询sql的次数)，显然，这个不够全面，不能描述增、删、改。

1. TPS与QPS区别：

如果是对一个查询接口压测，且这个接口内部不会再去请求其他接口，那么TPS=QPS，否则TPS≠QPS，比如访问一个页面会请求三次，一次请求会产生一个“T”,产生三个“Q”。

1. 吞吐量：

或者叫做最大吞吐能力，每秒的响应请求数量,跟TPS一个意思。一个系统的吞吐量与请求对cpu的消耗、还不接口、IO等等紧密关联。单个请求对CPU消耗越高，外部系统接口、IO影响速度越慢，系统吞吐能力越低，反之越高。

吞吐量/TPS = 系统同时处理的请求并发数/平均响应时间。

1. 性能相关

<https://www.cnblogs.com/uncleyong/p/11059556.html>

1. Redis
2. Redis？
3. redis客户端

①jedis

采用**同步阻塞式IO**，**采用线程池时是线程安全的**。简单、灵活，api全面，缺点是某些redis高级功能需要自己封装。

②redisTemplete

是基于某个具体实现的再封装，比如springboot1.x时，具体实现是jedis；而到了springboot2.x时，具体实现变成了lettuce.封装的好处就是基于springboot自动装配原理，使得整合redis比较简单，隐藏了具体实现，调用更方便，但是**jedis的执行效率要比redisTemplete高10-30倍。**

③redisson

作为redis的**分布式客户端**，**基于netty采用异步非阻塞IO，是线程安全的**。提供了很多redis的分布式操作和高级功能，例如集群、哨兵、管道等，缺点是api抽象，学习成本高。

④lettuce

1. redis集群
2. 分布式架构

cdn:Content Delivery Network,内容分发网络。依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。安装在网络服务器上。

1. 分布式锁
2. 基于数据库的分布式锁
3. 基于redis的setnx实现分布式锁

①原理

通过setnx命令将名为key的锁设置到redis中，如果该key存在就插入失败，返回0，表示抢锁失败，否则插入成功，返回1，表示抢锁成功。抢锁成功的线程执行完任务，需要释放锁，以便其他线程可以进入。

②基于redis实现分布式锁的过程会遇到以下问题

问题1：**锁超时**。

当得到锁的线程再执行任务的过程中挂掉，来不及释放锁，这样会形成死锁，及这个key的锁一直存在于redis中，其他线程就会一直set不成功。所以，需要**设置超时时间**。

问题2：**必须保证set和设置超时时间两个操作的原子性**。

如果在设置超时时间时设置失败，同样会导致死锁。所以需要保证set和设置超时时间的原子性。可以**使用高版本的redis客户端工具，调用它的多参数set方法**。

问题3：**执行完任务需要删除锁**。

加入设置了超时时间为60s,但是我执行线程只换了3秒，如果执行完线程之后不主动删除锁，那么会导致其他线程白白等待57s,这样严重影响了效率。

问题4：**在并发情况下，可能会删除其他人的锁**。

比如，在多并发情况下，由于线程A的业务处理时间时间很长，锁自动过期了，但是线程A还是在处理。线程B此时获取了锁，在线程B执行的过程中，线程A业务处理完，然后执行删除所操作，线程C此时又能获取到锁了，导致达不到锁的效果。可以把值指定为uuid，**在删除锁时，判断锁的值是否等于当前线程生成的uuid**。同样也要保证判断和删除操作的原子性，我们此时需要**使用LUA脚本来保证判断和删除操作的原子性**。

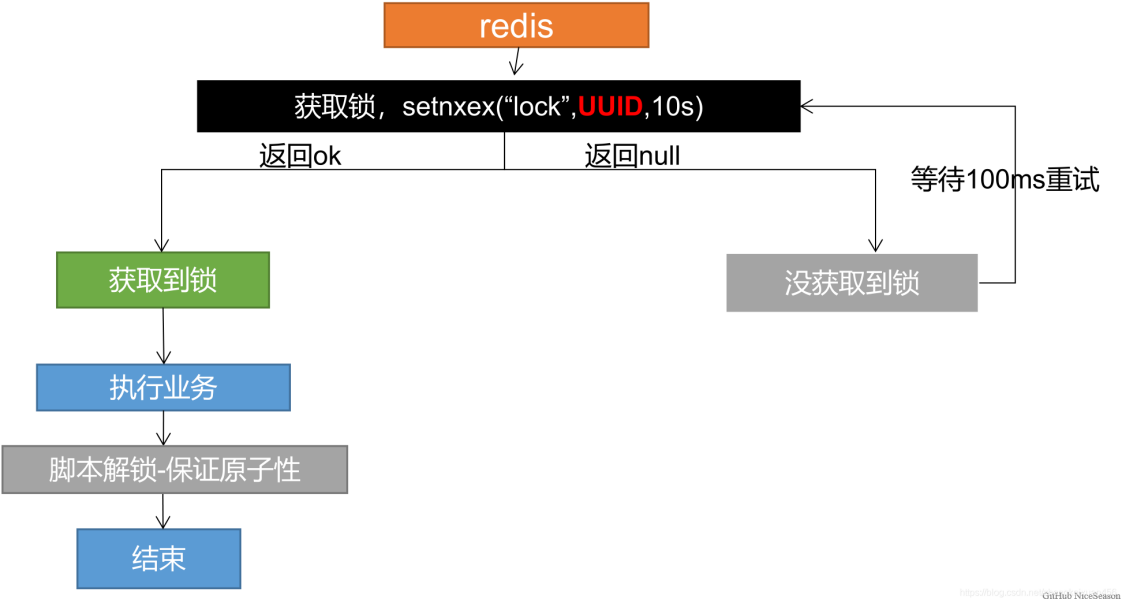
问题5：**续约问题**。

在多并发情况下，由于线程A的业务处理时间时间很长，锁自动过期了，但是线程A还是在处理，而线程B此时获取了锁，造成了同时有两个线程持有锁。这个问题归结为“续约问题”，即A线程每执行完时应该过期时间续约，执行完成才能释放锁。可以**让获得锁的线程开启一个守护线程，用来给快要过期的锁续约，或者通过Redisson的”看门狗”来实现**。问题4也属于续约问题。

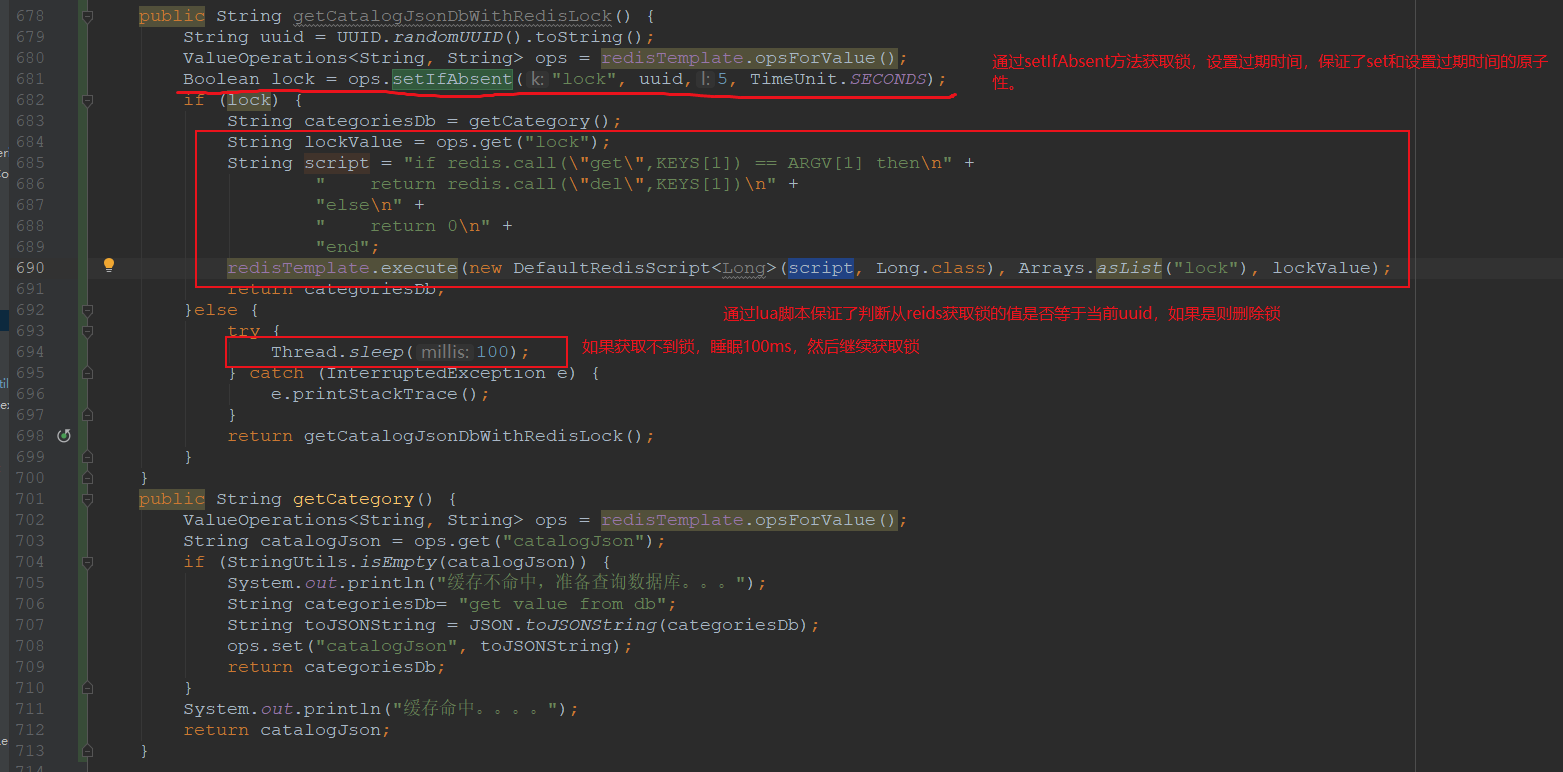
问题6：**集群同步延时问题**。

当项目需要实现rdis高可用时，就得部署多台redis，因为单机的redis，一旦redis挂掉，整个分布式锁就不没用了。在集群场景下，如果线程A在master拿到了锁，在没有把数据同步到slave时，master挂掉了。B线程再拿锁会从slave拿锁，此时线程B能获取到锁，又出现了多个线程同时获取到锁的情况。

最终逻辑如下图：



代码参考如下：



上述代码没有解决续约问题和集群的同步延时问题。

Redis

<https://www.jb51.net/article/125931.htm>

<https://blog.csdn.net/SharingOfficer/article/details/122117251>

https://blog.csdn.net/qq\_29860591/article/details/113243441?utm\_medium=distribute.pc\_aggpage\_search\_result.none-task-blog-2~aggregatepage~first\_rank\_ecpm\_v1~rank\_v31\_ecpm-3-113243441.pc\_agg\_new\_rank&utm\_term=redis%E4%B8%BB%E4%BB%8E%E5%90%8C%E6%AD%A5%E5%BB%B6%E8%BF%9F%E5%AF%BC%E8%87%B4%E5%88%86%E5%B8%83%E5%BC%8F%E9%94%81%E9%97%AE%E9%A2%98&spm=1000.2123.3001.4430

<https://blog.csdn.net/zjcjava/article/details/103025997>

https://blog.csdn.net/qq\_41653753/article/details/94979706?spm=1001.2101.3001.6661.1&utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.pc\_relevant\_default&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7Edefault-1.pc\_relevant\_default&utm\_relevant\_index=1

https://blog.csdn.net/Tangwp1214/article/details/102790497?spm=1001.2101.3001.6650.5&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.pc\_relevant\_aa&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-5.pc\_relevant\_aa&utm\_relevant\_index=10

1. 基于zk实现分布式锁

<https://www.php.cn/faq/466231.html>

<https://www.cnblogs.com/chenkeyu/p/8047811.html>

https://zhuanlan.zhihu.com/p/337052900

1. 分布式缓存

1、基于数据库的分布式锁