链滴

血的教训！千万别在生产使用这些 redis 指令

作者：9526xu

原文链接：https://ld246.com/article/1600217322329

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

哎，最近小黑哥又双叒叕犯事了。

事情是这样的，前一段时间小黑哥公司生产交易偶发报错，一番排查下来最终原因是因为 Redis 命令行超时。

可是令人不解的是，生产交易仅仅使用 Redis set 这个简单命令，这个命令讲道理是不可能会执行这么慢。

那到底是什么导致这个问题那？

为了找出这个问题，我们查看分析了一下 Redis 最近的慢日志，最终发现耗时比较多命令为 keys XX*

看到这个命令操作的键的前缀，小黑哥才发现这是自己负责的应用。可是小黑哥排查一下，虽然自己代码并没有主动去使用 keys命令，但是底层使用框架却在间接使用，于是就有了今天这个问题。

问题原因

小黑哥负责的应用是一个管理后台应用，权限管理使用 Shiro 框架，由于存在多个节点，需要使用分式 Session，于是这里使用 Redis 存储 Session 信息。

画外音：不知道分布式 Session ，可以看看小黑哥之前写的 一口气说出 4 种分布式一致性 Session 现方式，面试杠杠的~

由于 Shiro 并没有直接提供 Redis 存储 Session 组件，小黑哥不得不使用 Github 一个开源组件 shiro redis。

由于 Shiro 框架需要定期验证 Session 是否有效，于是 Shiro 底层将会调用 SessionDAO#getActiv Sessions 获取所有的 Session 信息。

而 shiro-redis 正好继承 SessionDAO 这个接口，底层使用用 keys 命令查找 Redis 所有存储的 Sessi

n key。

```
public Set<byte[]> keys(byte[] pattern){
    checkAndInit();
    Set<byte[]> keys = null;
    Jedis jedis = jedisPool.getResource();
    try{
        keys = jedis.keys(pattern);
    }finally{
        jedis.close();
    }
    return keys;
}
```

找到问题原因，解决办法就比较简单了，github 上查找到解决方案，升级一下 shiro-redis 到最新本。

在这个版本，shiro-redis 采用 scan命令代替 keys,从而修复这个问题。

```
public Set<byte[]> keys(byte[] pattern) {
    Set<byte[]> keys = null;
    Jedis jedis = jedisPool.getResource();

    try{
        keys = new HashSet<byte[]>();
        ScanParams params = new ScanParams();
        params.count(count);
        params.match(pattern);
        byte[] cursor = ScanParams.SCAN_POINTER_START_BINARY;
        ScanResult<byte[]> scanResult;
        do{
            scanResult = jedis.scan(cursor,params);
            keys.addAll(scanResult.getResult());
            cursor = scanResult.getCursorAsBytes();
        }while(scanResult.getStringCursor().compareTo(ScanParams.SCAN_POINTER_START) > 0);
    }finally{
        jedis.close();
    }
    return keys;

}
```

虽然问题成功解决了，但是小黑哥心里还是有点不解。

为什么 keys 指令会导致其他命令执行变慢?

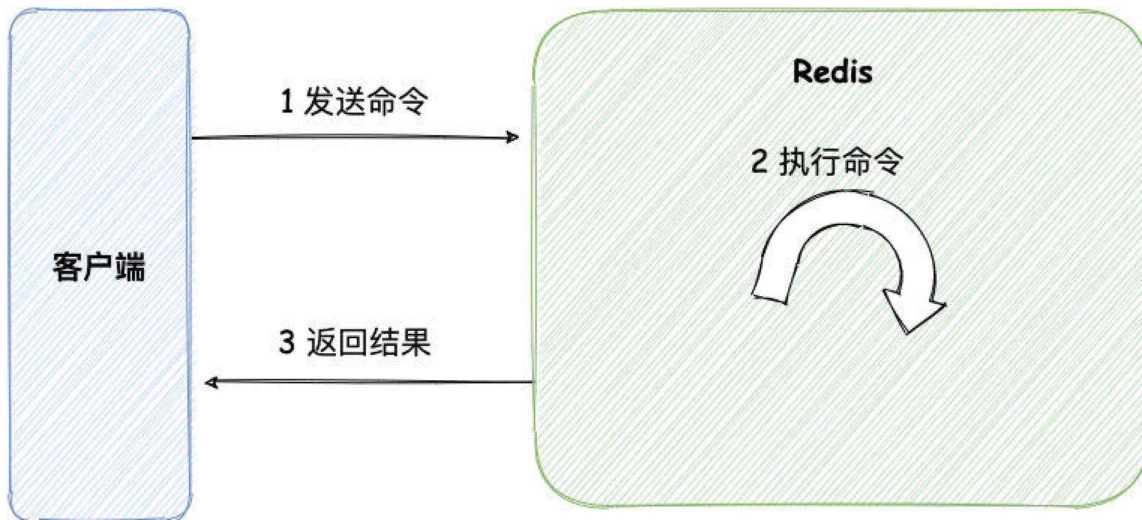
为什么 Keys 指令查询会这么慢?

为什么 Scan 指令就没有问题?

Redis 执行命令的原理

首先我们来看第一个问题，为什么 keys 指令会导致其他命令执行变慢?

回答这个问题，我们首先看下 Redis 客户端执行一条命令的情况：

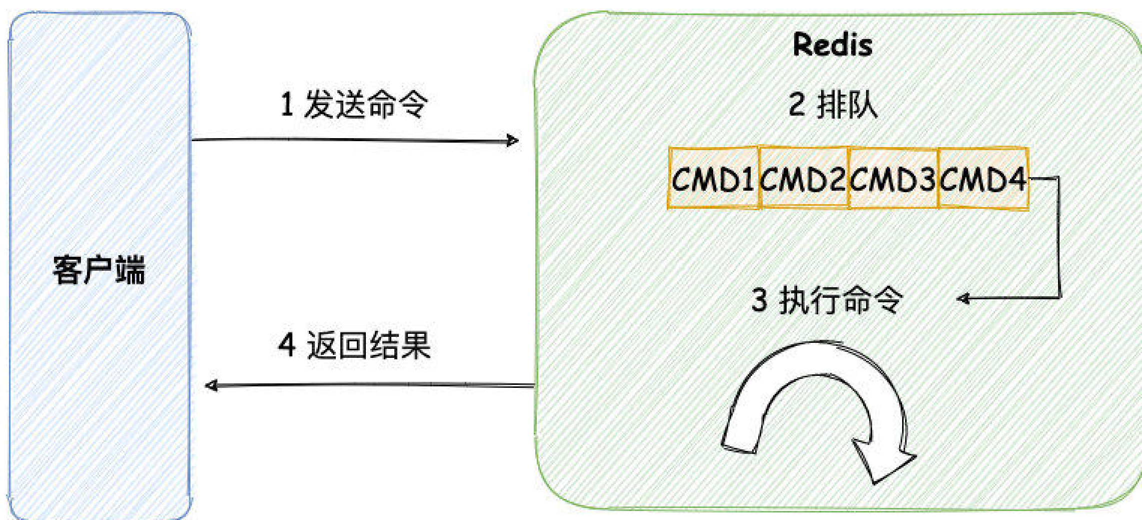


站在客户端的视角，执行一条命令分为三步：

1. 发送命令
2. 执行命令
3. 返回结果

但是这仅仅客户端自己以为的过程，但是实际上同一时刻，可能存在很多客户端发送命令给 Redis，而 Redis 我们都知道它采用的是单线程模型。

为了处理同一时刻所有的客户端的请求命令，Redis 内部采用了队列的方式，排队执行。



于是客户端执行一条命令实际需要四步：

1. 发送命令
2. 命令排队
3. 执行命令
4. 返回结果

原文链接：血的教训! 千万别在生产使用这些 redis 指令

由于 Redis 单线程执行命令，只能顺序从队列取出任务开始执行。

只要 3 这个过程执行命令速度过慢，队列其他任务不得不进行等待，这对外部客户端看来，Redis 好就被阻塞一样，一直得不到响应。

所以使用 Redis 过程切勿执行需要长时间运行的指令，这样可能导致 Redis 阻塞，影响执行其他指令。

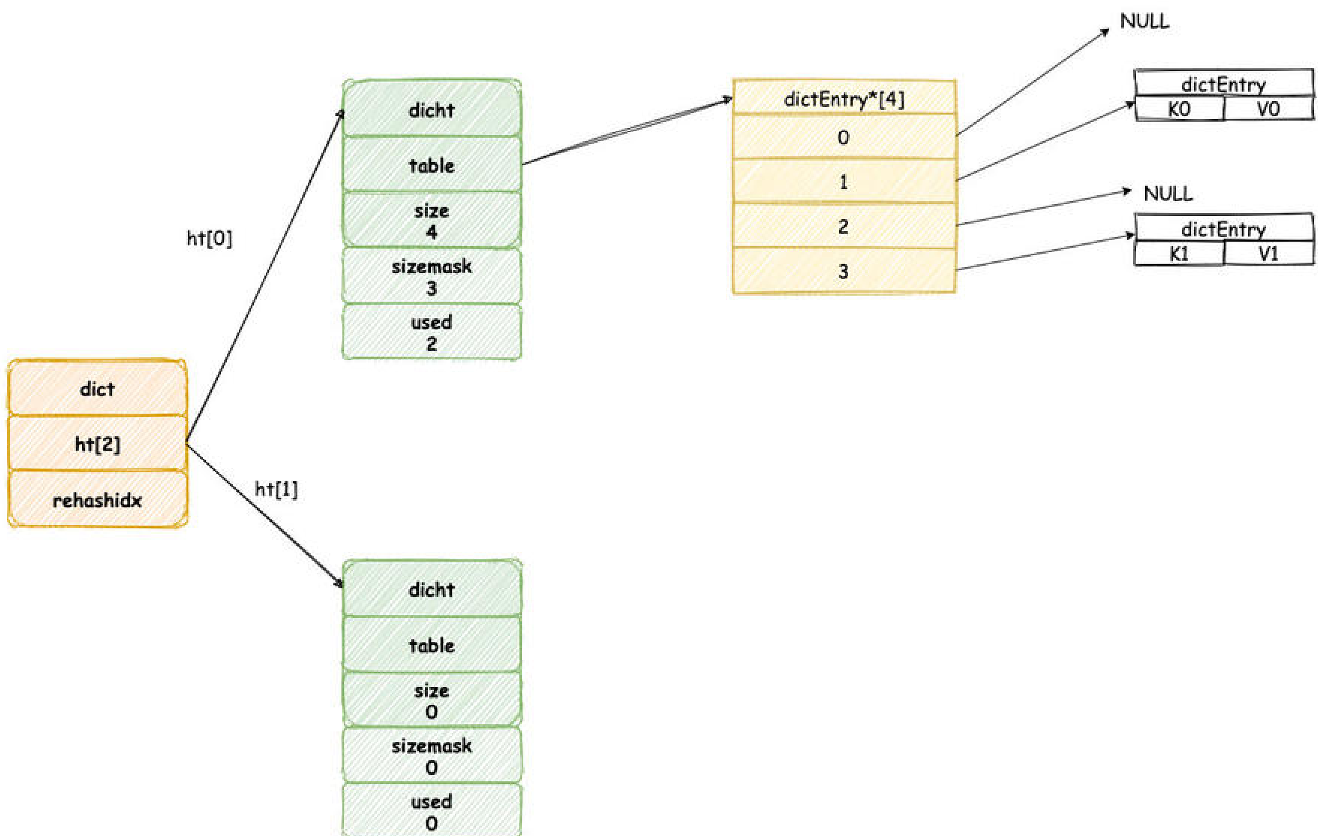
KEYS 原理

接下来开始回答第二个问题，为什么 Keys 指令查询会这么慢?

回答这个问题之前，请大家回想一下 Redis 底层存储结构。

不太清楚朋友的也没关系，大家可以回看一下小黑哥之前的文章「阿里面试官：HashMap 熟悉吧?的，那就来聊聊 Redis 字典吧！」。

这里小黑哥复制之前文章内容，Redis 底层使用字典这种结构，这个结构与 Java HashMap 底层比较似。



keys命令需要返回所有的符合给定模式 pattern 的 Redis 中键，为了实现这个目的，Redis 不得不历字典中 ht[0]哈希表底层数组，这个时间复杂度为 **O(N)**（N 为 Redis 中 key 所有的数量）。

如果 Redis 中 key 的数量很少，那么这个执行速度还是也会很快。等到 Redis key 的数量慢慢更加上升到百万、千万、甚至上亿级别，那这个执行速度就会很慢很慢。

下面是小黑哥本地做的一次实验，使用 lua 脚本往 Redis 中增加 10W 个 key，然后使用 keys 查询有键,这个查询大概会阻塞十几秒的时间。

```
eval "for i=1,100000  do redis.call('set',i,i+1) end" 0
```

这里小黑哥使用 Docker 部署 Redis，性能可能会稍差。

SCAN 原理

最后我们来看下第三个问题，为什么 scan 指令就没有问题？

这是因为 scan命令采用一种黑科技-**基于游标的迭代器。**

每次调用 scan 命令，Redis 都会向用户返回一个新的游标以及一定数量的 key。下次再想继续获取余的 key，需要将这个游标传入 scan 命令， 以此来延续之前的迭代过程。

简单来讲，scan 命令使用分页查询 redis 。

下面是一个 scan 命令的迭代过程示例：

scan 命令使用游标这种方式，巧妙将一次全量查询拆分成多次，降低查询复杂度。

虽然 scan 命令时间复杂度与 keys一样，都是 **O(N)**，但是由于 scan 命令只需要返回少量的 key，以执行速度会很快。

最后，虽然scan 命令解决 keys不足，但是同时也引入其他一些缺陷：

- 同一个元素可能会被返回多次，这就需要我们应用程序增加处理重复元素功能。
- 如果一个元素在迭代过程增加到 redis，或者说在迭代过程被删除，那个这个元素会被返回，也可不会。

以上这些缺陷，在我们开发中需要考虑这种情况。

除了 scan以外，redis 还有其他几个用于增量迭代命令：

- sscan:用于迭代当前数据库中的数据库键，用于解决 smembers 可能产生阻塞问题
- hscan命令用于迭代哈希键中的键值对，用于解决 hgetall 可能产生阻塞问题。
- zscan:命令用于迭代有序集合中的元素（包括元素成员和元素分值），用于产生 zrange 可能产生塞问题。

总结

Redis 使用单线程执行操作命令，所有客户端发送过来命令，Redis 都会现放入队列，然后从队列中序取出执行相应的命令。

如果任一任务执行过慢，就会影响队列中其他任务的，这样在外部客户端看来，迟迟拿不到 Redis 的应，看起来就很阻塞了一样。

所以不要在生产执行 keys、smembers、hgetall、zrange这类可能造成阻塞的指令，如果真需要执，可以使用相应的scan 命令渐进式遍历，可以有效防止阻塞问题。

原文链接：血的教训! 千万别在生产使用这些 redis 指令