Redis 为什么快？

1.纯内存 KV 操作

Redis 的操作都是基于内存的，CPU 不是 Redis 性能瓶颈,，Redis 的瓶颈是机器内存和网络带宽。

在计算机的世界中，CPU 的速度是远大于内存的速度的，同时内存的速度也是远大于硬盘的速度。redis 的操作都是基于内存的，绝大部分请求是纯粹的内存操作，非常迅速。

2.单线程操作

使用单线程可以省去多线程时 CPU 上下文会切换的时间，也不用去考虑各种锁的问题，不存在加锁释放锁操作，没有死锁问题导致的性能消耗。对于内存系统来说，多次读写都是在一个 CPU 上，没有上下文切换效率就是最高的！既然单线程容易实现，而且 CPU 不会成为瓶颈，那就顺理成章的采用单线程的方案了

Redis 单线程指的是网络请求模块使用了一个线程，即一个线程处理所有网络请求，其他模块该使用多线程，仍会使用了多个线程。

3.I/O 多路复用

为什么 Redis 中要使用 I/O 多路复用这种技术呢？

首先，Redis 是跑在单线程中的，所有的操作都是按照顺序线性执行的，但是由于读写操作等待用户输入或输出都是阻塞的，所以 I/O 操作在一般情况下往往不能直接返回，这会导致某一文件的 I/O 阻塞导致整个进程无法对其它客户提供服务，而 I/O 多路复用就是为了解决这个问题而出现的

4.Reactor 设计模式

Redis 基于 Reactor 模式开发了自己的网络事件处理器，称之为文件事件处理器(File Event Hanlder)。

Redis 合适的应用场景？

1、会话缓存（Session Cache）最常用的一种使用 Redis 的情景是会话缓存（sessioncache），用 Redis 缓存会话比其他存储（如 Memcached）的优势在于：Redis 提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？幸运的是，随着 Redis 这些年的改进，很容易找到怎么恰当的使用 Redis 来缓存会话的文档。甚至广为人知的商业平台 Magento 也提供 Redis 的插件。

2、全页缓存（FPC）除基本的会话 token 之外，Redis 还提供很简便的 FPC 平台。回到一致性问题，即使重启了 Redis 实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似 PHP 本地 FPC。再次以 Magento 为例，Magento 提供一个插件来使用 Redis 作为全页缓存后端。此外，对 WordPress 的用户来说，Pantheon 有一个非常好的插件 wp-redis，这个插件能帮助你以最快速度加载你曾浏览过的页面。

3、队列 Reids 在内存存储引擎领域的一大优点是提供 list 和 set 操作，这使得 Redis 能作为一个很好的消息队列平台来使用。Redis 作为队列使用的操作，就类似于本地程序语言（如 Python）对 list 的 push/pop 操作。如果你快速的在 Google 中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用 Redis 创建非常好的后端工具，以满足各种队列需求。例如，Celery 有一个后台就是使用 Redis 作为 broker，你可以从这里去查看。

4、排行榜/计数器 Redis 在内存中对数字进行递增或递减的操作实现的非常好。集合（Set）和有序集合（SortedSet）也使得我们在执行这些操作的时候变的非常简单，Redis 只是正好提供了这两种数据结构。微信搜索公众号：Java 专栏，获取最新面试手册所以，我们要从排序集合中获取到排名最靠前的 10 个用户－我们称之为“user_scores”，我们只需要像下面一样执行即可：当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：ZRANGE user_scores 0 10 WITHSCORESAgora Games 就是一个很好的例子，用 Ruby 实现的，它的排行榜就是使用 Redis 来存储数据的，你可以在这里看到。

5、发布/订阅最后（但肯定不是最不重要的）是 Redis 的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用 Redis 的发布/订阅功能来建立聊天系统！

Redis6.0 之前为什么一直不使用多线程？

官方曾做过类似问题的回复：使用 Redis 时，几乎不存在 CPU 成为瓶颈的情况，

Redis 主要受限于内存和网络。例如在一个普通的 Linux 系统上，Redis 通过使用 pipelining 每秒可以处理 100 万个请求，所以如果应用程序主要使用 O(N)或 O(log(N))的命令，它几乎不会占用太多 CPU。

使用了单线程后，可维护性高。多线程模型虽然在某些方面表现优异，但是它却引入了程序执行顺序的不确定性，带来了并发读写的一系列问题，增加了系统复杂度、同时可能存在线程切换、甚至加锁解锁、死锁造成的性能损耗。Redis 通过 AE 事件模型以及 IO 多路复用等技术，处理性能非常高，因此没有必要使用多线程。单线程机制使得 Redis 内部实现的复杂度大大降低，Hash 的惰性 Rehash、Lpush 等等,“线程不安全” 的命令都可以无锁进行。

Redis6.0 为什么要引入多线程？

Redis 将所有数据放在内存中，内存的响应时长大约为 100 纳秒，对于小数据包，Redis 服务器可以处理 80,000 到 100,000 QPS，这也是 Redis 处理的极限了，对于 80%的公司来说，单线程的 Redis 已经足够使用了。

但随着越来越复杂的业务场景，有些公司动不动就上亿的交易量，因此需要更大的 QPS。常见的解决方案是在分布式架构中对数据进行分区并采用多个服务器，但该方案有非常大的缺点，例如要管理的 Redis 服务器太多，维护代价大；某些适用于单个 Redis 服务器的命令不适用于数据分区；数据分区无法解决热点读/写问题；数据偏斜，重新分配和放大/缩小变得更加复杂等等。

Redis 有哪些高级功能？

消息队列、自动过期删除、事务、数据持久化、分布式锁、附近的人、慢查询分析、Sentinel 和集群等多项功能。

为什么要用 Redis？

使用缓存的目的就是提升读写性能。而实际业务场景下，更多的是为了提升读性能，带来更好的性能，带来更高的并发量。Redis 的读写性能比 Mysql 好的多，我们就可以把 Mysql 中的热点数据缓存到 Redis 中，提升读取性能，同时也减轻了 Mysql 的读取压力

Redis 与 memcached 相对有哪些优势？

(1) memcached 所有的值均是简单的字符串，redis 作为其替代者，支持更为丰富的数据类型

(2) redis 的速度比 memcached 快很多

(3) redis 可以持久化其数据

(4)Redis 支持数据的备份，即 master-slave 模式的数据备份。

(5) 使用底层模型不同，它们之间底层实现方式 以及与客户端之间通信的应用协议不一样。Redis 直接自己构建了 VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。

(6）value 大小：redis 最大可以达到 1GB，而 memcache 只有 1MB

怎么理解 Redis 中事务？

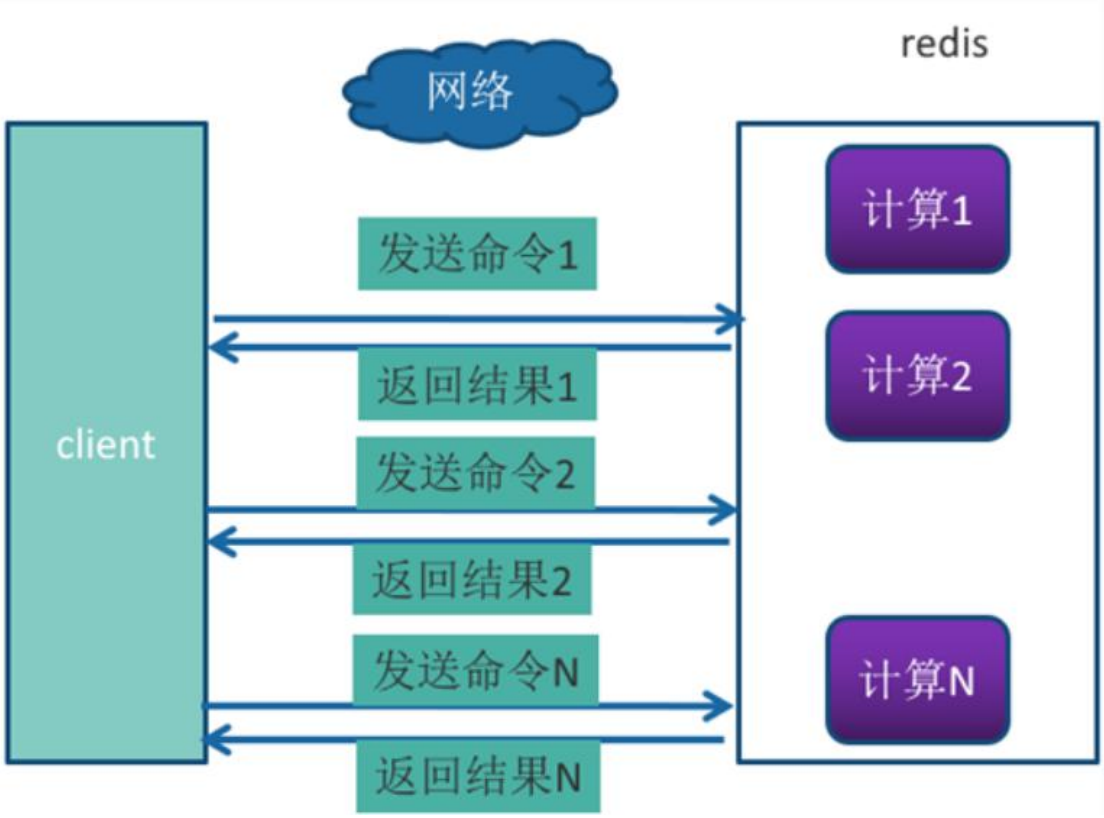
事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行，事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。不过 Redis 的是弱事物。

事务是 Redis 实现在服务器端的行为，用户执行 MULTI 命令时，服务器会将对应这个用户的客户端对象设置为一个特殊的状态，在这个状态下后续用户执行的查询命令不会被真的执行，而是被服务器缓存起来，直到用户执行 EXEC 命令为止，服务器会将这个用户对应的客户端对象中缓存的命令按照提交的顺序依次执行。

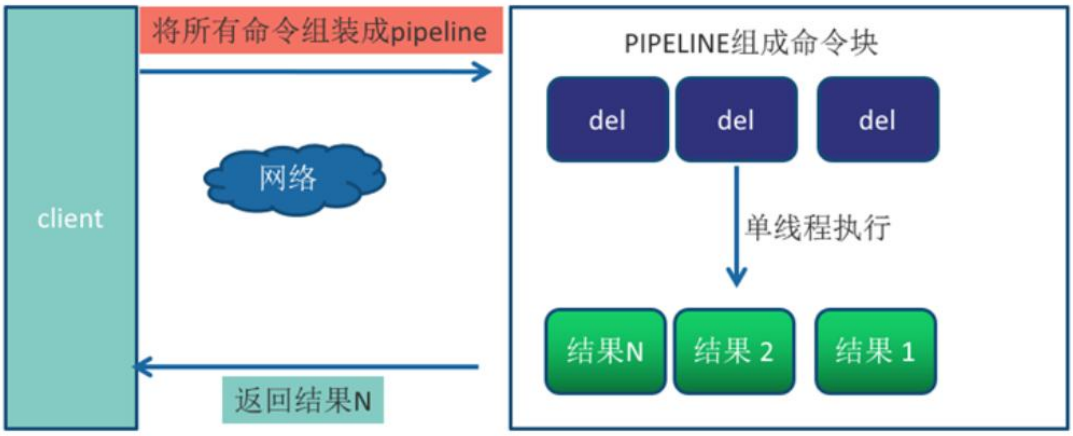
Redis 提供了简单的事务，之所以说它简单，主要是因为它不支持事务中的回滚特性，同时无法实现命令之间的逻辑关系计算，当然也体现了 Redis 的“keep it simple”的特性

为什么要使用 pipeline

Pipeline（流水线)机制能改善上面这类问题,它能将一组 Redis 命令进行组装,通过一次 RTT 传输给 Redis,再将这组 Redis 命令的执行结果按顺序返回给客户端,没有使用 Pipeline 执行了 n 条命令,整个过程需要 n 次 RTT。



使用 Pipeline 执行了 n 次命令，整个过程需要 1 次 RTT。



Redis 的过期策略以及内存淘汰机制？

redis 采用的是定期删除+惰性删除策略。 为什么不用定时删除策略? 定时删除,用一个定时器来负责监视 key,过期则自动删除。虽然内存及时释放，但是十分消耗 CPU 资源。在大并发请求下，CPU 要将时间应用在处理请求，而不是删除 key,因此没有采用这一策略. 定

期删除+惰性删除是如何工作的呢？定期删除，redis 默认每个 100ms 检查，是否有过期的 key,有过期 key 则删除。需要说明的是，redis 不是每个 100ms 将所有的 key 检查一次，而是随机抽取进行检查(如果每隔 100ms,全部 key 进行检查，redis 岂不是卡死)。因此，如果只采用定期删除策略，会导致很多 key 到时间没有删除。 于是，惰性删除派上用场。也就是说在你获取某个 key 的时候，redis 会检查一下，这个 key 如果设置了过期时间那么是否过期了？如果过期了此时就会删除

1.noeviction:返回错误当内存限制达到，并且客户端尝试执行会让更多内存被使用的命令。

2.allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。

3.volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。

4.allkeys-random: 回收随机的键使得新添加的数据有空间存放。

5.volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

6.volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。

什么是缓存穿透？如何避免？

缓存穿透：指查询一个一定不存在的数据，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到 DB 去查询，可能导致 DB 挂掉。

解决方案：1.查询返回的数据为空，仍把这个空结果进行缓存，但过期时间会比较短；2.布隆过滤器：将所有可能存在的数据哈希到一个足够大的 bitmap 中，一个一定不存在的数据会被这个 bitmap 拦截掉，从而避免了对 DB 的查询。

什么是缓存雪崩？如何避免？

缓存雪崩：设置缓存时采用了相同的过期时间，导致缓存在某一时刻同时失效，请求全部转发到 DB，DB 瞬时压力过重雪崩。与缓存击穿的区别：雪崩是很多 key，击穿是某一个 key 缓存。

解决方案：将缓存失效时间分散开，比如可以在原有的失效时间基础上增加一个随机值，比如 1-5 分钟随机，这样每一个缓存的过期时间的重复率就会降低，就很难引发集体失效的事件。

使用 Redis 如何设计分布式锁？

基于 Redis 实现的分布式锁，一个严谨的的流程如下：

1、加锁

SET lock_key $unique_id EX $expire_time NX

2、操作共享资源

3、释放锁：Lua 脚本，先 GET 判断锁是否归属自己，再 DEL 释放锁

怎么使用 Redis 实现消息队列？

基于 List 的 LPUSH+BRPOP 的实现

足够简单，消费消息延迟几乎为零，但是需要处理空闲连接的问题。

如果线程一直阻塞在那里，Redis 客户端的连接就成了闲置连接，闲置过久，服务器一般会主动断开连接，减少闲置资源占用，这个时候 blpop 和 brpop 或抛出异常，所以在编写客户端消费者的时候要小心，如果捕获到异常，还有重试。

其他缺点包括：

做消费者确认 ACK 麻烦，不能保证消费者消费消息后是否成功处理的问题（宕机或处理异常等），通常需要维护一个 Pending 列表，保证消息处理确认；不能做广播模式，如 pub/sub，消息发布/订阅模型；不能重复消费，一旦消费就会被删除；不支持分组消费。

基于 Sorted-Set 的实现

多用来实现延迟队列，当然也可以实现有序的普通的消息队列，但是消费者无法阻塞的获取消息，只能轮询，不允许重复消息。

PUB/SUB，订阅/发布模式

优点：

典型的广播模式，一个消息可以发布到多个消费者；多信道订阅，消费者可以同时订阅多个信道，从而接收多类消息；消息即时发送，消息不用等待消费者读取，消费者会自动接收到信道发布的消息。

缺点：

消息一旦发布，不能接收。换句话就是发布时若客户端不在线，则消息丢失，不能寻回；不能保证每个消费者接收的时间是一致的；若消费者客户端出现消息积压，到一定程度，会被强制断开，导致消息意外丢失。通常发生在消息的生产远大于消费速度时；可见，Pub/Sub 模式不适合做消息存储，消息积压类的业务，而是擅长处理广播，即时通讯，即时反馈的业务。

基于 Stream 类型的实现

基本上已经有了一个消息中间件的雏形，可以考虑在生产过程中使用。

什么是 bigkey？会有什么影响？

bigkey 是指 key 对应的 value 所占的内存空间比较大，例如一个字符串类型的 value 可以最大存到 512MB，一个列表类型的 value 最多可以存储 23-1 个元素。

如果按照数据结构来细分的话，一般分为字符串类型 bigkey 和非字符串类型 bigkey。

字符串类型：体现在单个 value 值很大，一般认为超过 10KB 就是 bigkey，但这个值和具体的 OPS 相关。

非字符串类型：哈希、列表、集合、有序集合,体现在元素个数过多。

bigkey 无论是空间复杂度和时间复杂度都不太友好，下面我们将介绍它的危害。

bigkey 的危害

bigkey 的危害体现在三个方面:

1、内存空间不均匀.(平衡):例如在 Redis Cluster 中，bigkey 会造成节点的内存空间使用不均匀。

2、超时阻塞:由于 Redis 单线程的特性，操作 bigkey 比较耗时，也就意味着阻塞 Redis 可能性增大。

3、网络拥塞:每次获取 bigkey 产生的网络流量较大

假设一个 bigkey 为 1MB，每秒访问量为 1000，那么每秒产生 1000MB 的流量,对于普通的千兆网卡(按照字节算是 128MB/s)的服务器来说简直是灭顶之灾，而且一般服务器会采用单机多实例的方式来部署,也就是说一个 bigkey 可能会对其他实例造成影响,其后果不堪设想。

Redis 如何解决 key 冲突？

遇到 hash 冲突采用链表进行处理

怎么提高缓存命中率？

需要在业务需求，缓存粒度，缓存策略，技术选型等各个方面去通盘考虑并做权衡。尽可能的聚焦在高频访问且时效性要求不高的热点业务上，通过缓存预加载（预热）、增加存储容量、调整缓存粒度、更新缓存等手段来提高命中率。

Redis 持久化方式有哪些？有什么区别？

RDB、AOF、混合持久化。

RDB 的优缺点：

优点：RDB 持久化文件，速度比较快，而且存储的是一个二进制文件，传输起来很方便。

缺点：RDB 无法保证数据的绝对安全，有时候就是 1s 也会有很大的数据丢失。

AOF 的优缺点：

优点：AOF 相对 RDB 更加安全，一般不会有数据的丢失或者很少，官方推荐同时开启 AOF 和 RDB。

缺点：AOF 持久化的速度，相对于 RDB 较慢，存储的是一个文本文件，到了后期文件会比较大，传输困难。

为什么 Redis 需要把所有数据放到内存中？

Redis 为了达到最快的读写速度,将数据都读到内存中,并通过异步的方式将数据写入磁盘,所以 Redis 具有快速和数据持久化的特征。如果不将数据放在内存中,磁盘 I/O 速度为严重影响 Redis 的性能。

如何保证缓存与数据库双写时的数据一致性

第一种方案：采用延时双删策略

具体的步骤就是：

先删除缓存；

再写数据库；

休眠 500 毫秒；

再次删除缓存。

第二种方案：异步更新缓存(基于订阅 binlog 的同步机制)

技术整体思路：

MySQL binlog 增量订阅消费+消息队列+增量数据更新到 redis

Redis 集群方案应该怎么做？

1. Redis Sentinel 体量较小时，选择 Redis Sentinel，单主 Redis 足以支撑业务。

2. Redis ClusterRedis 官方提供的集群化方案，体量较大时，选择 Redis Cluster，通过分片，使用更多内存。

3. TwemproxTwemprox 是 Twtter 开源的一个 Redis 和 Memcached 代理服务器，主要用于管理 Redis 和 Memcached 集群，减少与 Cache 服务器直接连接的数量。

4. CodisCodis 是一个代理中间件，当客户端向 Codis 发送指令时，Codis 负责将指令转发到后面的 Redis 来执行，并将结果返回给客户端。一个 Codis 实例可以连接多个 Redis 实例，也可以启动多个 Codis 实例来支撑，每个 Codis 节点都是对等的，这样可以增加整体的 QPS 需求，还能起到容灾功能。

5. 客户端分片在 Redis Cluster 还没出现之前使用较多，现在基本很少热你使用了，在业务代码层实现，起几个毫无关联的 Redis 实例，在代码层，对 Key 进行 hash 计算，然后去对应的 Redis 实例操作数据。这种方式对 hash 层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

Redis 集群方案什么情况下会导致整个集群不可用？

1.当访问一个 Master 和 Slave 节点都挂了的槽的时候，会报槽无法获取。

2.当集群 Master 节点个数小于 3 个的时候，或者集群可用节点个数为偶数的时候，基于 fail 的这种选举机制的自动主从切换过程可能会不能正常工作，一个是标记 fail 的过程，一个是选举新的 master 的过程，都有可能异常。

说一说 Redis 哈希槽的概念？

slot：称为哈希槽

Redis 集群中内置了 16384 个哈希槽，当需要在 Redis 集群中放置一个 key-value 时，redis 先对 key 使用 crc16 算法算出一个结果，然后把结果对 16384 求余数，这样每个 key 都会对应一个编号在 0-16383 之间的哈希槽，redis 会根据节点数量大致均等的将哈希槽映射到不同的节点。

使用哈希槽的好处就在于可以方便的添加或移除节点。

当需要增加节点时，只需要把其他节点的某些哈希槽挪到新节点就可以了；

当需要移除节点时，只需要把移除节点上的哈希槽挪到其他节点就行了；

Redis 集群会有写操作丢失吗？为什么？

以下情况可能导致写操作丢失：

过期 key 被清理

最大内存不足，导致 Redis 自动清理部分 key 以节省空间

主库故障后自动重启，从库自动同步

单独的主备方案，网络不稳定触发哨兵的自动切换主从节点，切换期间会有数据丢失

Redis 常见性能问题和解决方案有哪些？

一、缓存穿透：就是查询一个压根就不存在的数据，即缓存中没有，数据库中也没有

解决方案：使用布隆过滤器，把数据先加载到布隆过滤器中，访问前先判断是否存在于布隆过滤器中，不存在代表这笔数据压根就不存在。

缺点：布隆过滤器是不可变的，可能一开始过滤器和数据库数据时一致的，后面数据库数据变了，或变多或变少，而对应的布隆过滤器的数据也要改变，这时会比较麻烦。

二、缓存击穿：数据库中有，缓存中没有。缓存击穿实际就是一个并发问题，一般来说查询数据，先查询缓存，有直接返回，没有再查询数据库并放到缓存中之后返回，但这种场景在并发情况下就会有问题，假设同时又 100 个请求执行上面

逻辑的代码，则可能会出现多个请求都查询数据库，因为大家同时执行，都查到了缓存中没有数据。

解决方案：加锁。如果是单机部署，则可以使用 JVM 级别的锁，如 lock、synchronized。如果是集群部署，则需要使用分布式锁，如基于 redis、zookeeper、mysql 等实现的分布式锁。

三、缓存雪崩：大部分数据同时失效、过期，新的缓存又没来，导致大量的请求都去访问数据库而导致的服务器压力过大、宕机、系统崩溃。

解决方案：搭建高可用的 redis 集群，避免压力集中于一个节点；缓存失效时间错开，避免缓存同时失效而都去请求数据库。

热点数据和冷数据是什么

对于冷数据而言，大部分数据可能还没有再次访问到就已经被挤出内存，不仅占用内存，而且价值不 大。频繁修改的数据，看情况考虑使用缓存 对于上面两个例子，寿星列表、导航信息都存在一个特点，就是信息修改频率不高，读取通常非常高的 场景。

对于热点数据，比如我们的某 IM 产品，生日祝福模块，当天的寿星列表，缓存以后可能读取数十万次。 再举个例子，某导航产品，我们将导航信息，缓存以后可能读取数百万次。

什么情况下可能会导致 Redis 阻塞？

数据集中过期

不合理地使用 API 或数据结构

CPU 饱和

持久化阻塞

什么时候选择 Redis，什么时候选择 Memcached？

实际业务分析

如果业务中更加侧重性能的高效性，对持久化要求不高，那么应该优先选择 Memcached。

如果业务中对持久化有需求或者对数据涉及到存储、排序等一系列复杂的操作，比如业务中有排行榜类应用、社交关系存储、数据排重、实时配置等功能，那么应该优先选择 Redis

Redis 过期策略都有哪些？LRU 算法知道吗？

1.noeviction:返回错误当内存限制达到，并且客户端尝试执行会让更多内存被使用的命令。

2.allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。

3.volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键,使得新添加的数据有空间存放。

4.allkeys-random: 回收随机的键使得新添加的数据有空间存放。

5.volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

6.volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键,使得新添加的数据有空间存放。