1、什么是 Redis？简述它的优缺点？

Redis 的全称是：Remote Dictionary.Server，本质上是一个 Key-Value 类型的内存数据库，很像 memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据 flush 到硬盘上进行保存。

因为是纯内存操作，Redis 的性能非常出色，每秒可以处理超过 10 万次读写操作，是已知性能最快的 Key-Value DB。

Redis 的出色之处不仅仅是性能，Redis 最大的魅力是支持保存多种数据结构，此外单个 value 的最大限制是 1GB，不像 memcached 只能保存 1MB 的数据，因此 Redis 可以用来实现很多有用的功能。

比方说用他的 List 来做 FIFO 双向链表，实现一个轻量级的高性 能消息队列服务，用他的 Set 可以做高性能的 tag 系统等等。

另外 Redis 也可以对存入的 Key-Value 设置 expire 时间，因此也可以被当作一 个功能加强版的 memcached 来用。 Redis 的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此 Redis 适合的场景主要局限在较小数据量的高性能操作和运算上。

2、Redis 与 memcached 相比有哪些优势？

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 redis 的速度比 memcached 快很多

3.redis 可以持久化其数据 redis 可以持久化其数据

3、Redis 支持哪几种数据类型？

String、List、Set、Sorted Set、hashes

4、Redis 主要消耗什么物理资源？

内存。

5、Redis 有哪几种数据淘汰策略？

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。

6、Redis 官方为什么不提供 Windows 版本？

因为目前 Linux 版本已经相当稳定，而且用户量很大，无需开发 windows 版本，反而会带来兼容性等问题。

7、一个字符串类型的值能存储最大容量是多少？

512M

8、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。

在内存越来越便宜的今天，redis 将会越来越受欢迎， 如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

9、Redis 集群方案应该怎么做？都有哪些方案？

1.codis

2.目前用的最多的集群方案，基本和 twemproxy 一致的效果，但它支持在节点数量改变情况下，旧节点数据可恢复到新 hash 节点。

redis cluster3.0 自带的集群，特点在于他的分布式算法不是一致性 hash，而是 hash 槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

3.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。

10、Redis 集群方案什么情况下会导致整个集群不可用？

有 A，B，C 三个节点的集群,在没有复制模型的情况下,如果节点 B 失败了，那么整个集群就会以为缺少 5501-11000 这个范围的槽而不可用。

11、MySQL 里有 2000w 数据，redis 中只存 20w 的数据，如何保证 redis 中的数据都是热点数据？
redis 内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

12、Redis 有哪些适合的场景？

（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。

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 WITHSCORES

Agora 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！

13、Redis 支持的 Java 客户端都有哪些？官方推荐用哪个？

Redisson、Jedis、lettuce 等等，官方推荐使用 Redisson。

14、Redis 和 Redisson 有什么关系？

Redisson 是一个高级的分布式协调 Redis 客服端，能帮助用户在分布式环境中轻松实现一些 Java 的对象（Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog）。

15、Jedis 与 Redisson 对比有什么优缺点？

Jedis 是 Redis 的 Java 实现的客户端，其 API 提供了比较全面的 Redis 命令的支持；

Redisson 实现了分布式和可扩展的 Java 数据结构，和 Jedis 相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等 Redis 特性。Redisson 的宗旨是促进使用者对 Redis 的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

16、说说 Redis 哈希槽的概念？

Redis 集群没有使用一致性 hash,而是引入了哈希槽的概念，Redis 集群有 16384 个哈希槽，每个 key 通过 CRC16 校验后对 16384 取模来决定放置哪个槽，集群的每个节点负责一部分 hash 槽。

17、Redis 集群的主从复制模型是怎样的？

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型，每个节点都会有 N-1 个复制品.

18、Redis 集群会有写操作丢失吗？为什么？

Redis 并不能保证数据的强一致性，这意味这在实际中集群在特定的条件下可能会丢失写操作。

19、 Redis 集群之间是如何复制的？

异步复制

20、Redis 集群最大节点个数是多少？

16384 个

21、Redis 集群如何选择数据库？

Redis 集群目前无法做数据库选择，默认在 0 数据库。

22、Redis 中的管道有什么用？

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应，这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多 POP3 协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

23、怎么理解 Redis 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。

24、Redis 事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

25、Redis key 的过期时间和永久有效分别怎么设置？

EXPIRE 和 PERSIST 命令

26、Redis 如何做内存优化？

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。

比如你的 web 系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的 key,而是应该把这个用户的所有信息存储到一张散列表里面。

27、Redis 回收进程如何工作的？

一个客户端运行了新的命令，添加了新的数据。

Redi 检查内存使用情况，如果大于 maxmemory 的限制, 则根据设定好的策略进行回收。

一个新的命令被执行，等等。

所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地回收回到边界以下。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

28.加锁机制

咱们来看上面那张图，现在某个客户端要加锁。如果该客户端面对的是一个 redis cluster 集群，他首先会根据 hash 节点选择一台机器。**这里注意**，仅仅只是选择一台机器！这点很关键！紧接着，就会发送一段 lua 脚本到 redis 上，那段 lua 脚本如下所示：

为啥要用 lua 脚本呢？因为一大坨复杂的业务逻辑，可以通过封装在 lua 脚本中发送给 redis，保证这段复杂业务逻辑执行的**原子性**。

那么，这段 lua 脚本是什么意思呢？这里 **KEYS[1]**代表的是你加锁的那个 key，比如说：RLoc k lock = redisson.getLock("myLock");这里你自己设置了加锁的那个锁 key 就是“myLock”。

ARGV[1]代表的就是锁 key 的默认生存时间，默认 30 秒。**ARGV[2]**代表的是加锁的客户端的 I D，类似于下面这样：8743c9c0-0795-4907-87fd-6c719a6b4586:1

给大家解释一下，第一段 if 判断语句，就是用“**exists myLock**”命令判断一下，如果你要加锁的那个锁 key 不存在的话，你就进行加锁。如何加锁呢？很简单，**用下面的命令**：hset myLoc k

8743c9c0-0795-4907-87fd-6c719a6b4586:1 1，通过这个命令设置一个 hash 数据结构，这行命令执行后，会出现一个类似下面的数据结构：

上述就代表“8743c9c0-0795-4907-87fd-6c719a6b4586:1”这个客户端对“myLock”这个锁 key 完成了加锁。接着会执行“**pexpire myLock 30000**”命令，设置 myLock 这个锁 key 的**生存时间是 30 秒**。好了，到此为止，ok，加锁完成了。

29.锁互斥机制

那么在这个时候，如果客户端 2 来尝试加锁，执行了同样的一段 lua 脚本，会咋样呢？很简单，第一个 if 判断会执行“**exists myLock**”，发现 myLock 这个锁 key 已经存在了。接着第二个 if 判断，判断一下，myLock 锁 key 的 hash 数据结构中，是否包含客户端 2 的 ID，但是明显不是的，因为那里包含的是客户端 1 的 ID。

所以，客户端 2 会获取到 **pttl myLock** 返回的一个数字，这个数字代表了 myLock 这个锁 key 的**剩余生存时间。**比如还剩 15000 毫秒的生存时间。此时客户端 2 会进入一个 while 循环，不停的尝试加锁。

30.watch dog 自动延期机制

客户端 1 加锁的锁 key 默认生存时间才 30 秒，如果超过了 30 秒，客户端 1 还想一直持有这把锁，怎么办呢？

简单！只要客户端 1 一旦加锁成功，就会启动一个 watch dog 看门狗，**他是一个后台线程，会每隔 10 秒检查一下**，如果客户端 1 还持有锁 key，那么就会不断的延长锁 key 的生存时间。

31.可重入加锁机制

那如果客户端 1 都已经持有了这把锁了，结果可重入的加锁会怎么样呢？比如下面这种代码：

这时我们来分析一下上面那段 lua 脚本。**第一个 if 判断肯定不成立**，“exists myLock”会显示锁 key 已经存在了。**第二个 if 判断会成立**，因为 myLock 的 hash 数据结构中包含的那个 ID，就是客户端 1 的那个 ID，也就是“8743c9c0-0795-4907-87fd-6c719a6b4586:1”

此时就会执行可重入加锁的逻辑，他会用：

incrby myLock 8743c9c0-0795-4907-87fd-6c71a6b4586:1 1 ，通过这个命令，对客户端 1 的加锁次数，累加 1。此时 myLock 数据结构变为下面这样：

大家看到了吧，那个 myLock 的 hash 数据结构中的那个客户端 ID，就对应着加锁的次数

32.释放锁机制

如果执行 lock.unlock()，就可以释放分布式锁，此时的业务逻辑也是非常简单的。其实说白了，就是每次都对 myLock 数据结构中的那个加锁次数减 1。如果发现加锁次数是 0 了，说明这个客户端已经不再持有锁了，此时就会用：**“del myLock”命令**，从 redis 里删除这个 key。然后呢，另外的客户端 2 就可以尝试完成加锁了。这就是所谓的**分布式锁的开源 Redisson 框架的实现机制。**

一般我们在生产系统中，可以用 Redisson 框架提供的这个类库来基于 redis 进行分布式锁的加锁与释放锁。

33.上述 Redis 分布式锁的缺点

其实上面那种方案最大的问题，就是如果你对某个 redis master 实例，写入了 myLock 这种锁 key 的 value，此时会异步复制给对应的 master slave 实例。但是这个过程中一旦发生 redis master 宕机，主备切换，redis slave 变为了 redis master。

接着就会导致，客户端 2 来尝试加锁的时候，在新的 redis master 上完成了加锁，而客户端 1 也以为自己成功加了锁。此时就会导致多个客户端对一个分布式锁完成了加锁。这时系统在业务语义上一定会出现问题，**导致各种脏数据的产生**。

所以这个就是 redis cluster，或者是 redis master-slave 架构的主从异步复制导致的 redis 分布式锁的最大缺陷：**在 redis master 实例宕机的时候，可能导致多个客户端同时完成加锁**。

34.使用过 Redis 分布式锁么，它是怎么实现的？

先拿 setnx 来争抢锁，抢到之后，再用 expire 给锁加一个过期时间防止锁忘记了释放。
如果在 setnx 之后执行 expire 之前进程意外 crash 或者要重启维护了，那会怎么样？
set 指令有非常复杂的参数，这个应该是可以同时把 setnx 和 expire 合成一条指令来用的！

35.使用过 Redis 做异步队列么，你是怎么用的？有什么缺点？

一般使用 list 结构作为队列，rpush 生产消息，lpop 消费消息。当 lpop 没有消息的时候，要适当 sleep 一会再重试。

缺点：

在消费者下线的情况下，生产的消息会丢失，得使用专业的消息队列如 rabbitmq 等。

能不能生产一次消费多次呢？

使用 pub/sub 主题订阅者模式，可以实现 1:N 的消息队列。

36.什么是缓存穿透？如何避免？什么是缓存雪崩？何如避免？

缓存穿透

一般的缓存系统，都是按照 key 去缓存查询，如果不存在对应的 value，就应该去后端系统查找（比如 DB）。一些恶意的请求会故意查询不存在的 key,请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。

如何避免？

1：对查询结果为空的情况也进行缓存，缓存时间设置短一点，或者该 key 对应的数据 insert 了之后清理缓存。

2：对一定不存在的 key 进行过滤。可以把所有的可能存在的 key 放到一个大的 Bitmap 中，查询时通过该 bitmap 过滤。

缓存雪崩

当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，会给后端系统带来很大压力。导致系统崩溃。

如何避免？

1：在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个 key 只允许一个线程查询数据和写缓存，其他线程等待。

2：做二级缓存，A1 为原始缓存，A2 为拷贝缓存，A1 失效时，可以访问 A2，A1 缓存失效时间设置为短期，A2 设置为长期

3：不同的 key，设置不同的过期时间，让缓存失效的时间点尽量均匀