[Redis的那些最常见面试问题](https://www.cnblogs.com/Survivalist/p/8119891.html)

　　随笔：经过长达一周的奔波和面试，电话面试，回首今天终于成功的入职了，总共面试了大概10家公司，包括阿里，京东，IBM等等，京东技术过了，学历因为非统招就被pass了，阿里面了2次电话面试就没下文了，估计是我当时最后提问题的时候减分了吧，其他的也有一些offer，不是不想去，就是了无音讯了，眼看年关将近，也由不得我挑挑拣拣了，就直接进了我现在这家公司，主要是感觉公司人不错，薪水这方面也就没有计较太多。好了，书归正文，今天小编就大家送上我精心准备的关于Redis方面的面试题，希望可以帮到还在求职路上的你们。

**1.什么是redis?**

Redis 是一个基于内存的高性能key-value数据库。

**2.Reids的特点**

　　　Redis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。

**3.使用redis有哪些好处？**

   (1) 速度快，因为数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)   
   (2) 支持丰富数据类型，支持string，list，set，sorted set，hash   
   (3) 支持事务，操作都是原子性，所谓的原子性就是对数据的更改要么全部执行，要么全部不执行   
   (4) 丰富的特性：可用于缓存，消息，按key设置过期时间，过期后将会自动删除

**4.redis相比memcached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

**5.Memcache与Redis的区别都有哪些？**

　　　1)、存储方式 Memecache把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小。 Redis有部份存在硬盘上，这样能保证数据的持久性。   
　　　2)、数据支持类型 Memcache对数据类型支持相对简单。 Redis有复杂的数据类型。   
　　　3)、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.redis常见性能问题和解决方案：**

　　　1).Master写内存快照，save命令调度rdbSave函数，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以Master最好不要写内存快照。

　　　2).Master AOF持久化，如果不重写AOF文件，这个持久化方式对性能的影响是最小的，但是AOF文件会不断增大，AOF文件过大会影响Master重启的恢复速度。Master最好不要做任何持久化工作，包括内存快照和AOF日志文件，特别是不要启用内存快照做持久

　　　　化,如果数据比较关键，某个Slave开启AOF备份数据，策略为每秒同步一次。

　　　3).Master调用BGREWRITEAOF重写AOF文件，AOF在重写的时候会占大量的CPU和内存资源，导致服务load过高，出现短暂服务暂停现象。

　　　4). Redis主从复制的性能问题，为了主从复制的速度和连接的稳定性，Slave和Master最好在同一个局域网内

**7. 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**

　　　　相关知识：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（回收策略）。redis 提供 6种数据淘汰策略：

* volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰
* volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰
* volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰
* allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰
* allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰
* no-enviction（驱逐）：禁止驱逐数据

**8.请用Redis和任意语言实现一段恶意登录保护的代码，限制1小时内每用户Id最多只能登录5次。具体登录函数或功能用空函数即可，不用详细写出。**

　　　　用列表实现:列表中每个元素代表登陆时间,只要最后的第5次登陆时间和现在时间差不超过1小时就禁止登陆.用Python写的代码如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #!/usr/bin/env python3  import redis  import sys  import time    r = redis.StrictRedis(host=’127.0.0.1′, port=6379, db=0)  try:      id = sys.argv[1]  except:      print(‘input argument error’)      sys.exit(0)  if r.llen(id) >= 5 and time.time() – float(r.lindex(id, 4)) <= 3600:      print(“you are forbidden logining”)  else:      print(‘you are allowed to login’)      r.lpush(id, time.time())      # login\_func() |

**9.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。

**10.Redis是单进程单线程的**

　　　redis利用队列技术将并发访问变为串行访问，消除了传统数据库串行控制的开销

**11.redis的并发竞争问题如何解决?**

　　　Redis为单进程单线程模式，采用队列模式将并发访问变为串行访问。Redis本身没有锁的概念，Redis对于多个客户端连接并不存在竞争，但是在Jedis客户端对Redis进行并发访问时会发生连接超时、数据转换错误、阻塞、客户端关闭连接等问题，这些问题均是由于客户端连接混乱造成。对此有2种解决方法：

　　　1.客户端角度，为保证每个客户端间正常有序与Redis进行通信，对连接进行池化，同时对客户端读写Redis操作采用内部锁synchronized。

　　　2.服务器角度，利用setnx实现锁。  
　　　注：对于第一种，需要应用程序自己处理资源的同步，可以使用的方法比较通俗，可以使用synchronized也可以使用lock；第二种需要用到Redis的setnx命令，但是需要注意一些问题。

**12.redis事物的了解CAS(check-and-set 操作实现乐观锁 )?**

　　　　和众多其它数据库一样，Redis作为NoSQL数据库也同样提供了事务机制。在Redis中，MULTI/EXEC/DISCARD/WATCH这四个命令是我们实现事务的基石。相信对有关系型数据库开发经验的开发者而言这一概念并不陌生，即便如此，我们还是会简要的列出Redis中事务的实现特征：  
　　　　1). 在事务中的所有命令都将会被串行化的顺序执行，事务执行期间，Redis不会再为其它客户端的请求提供任何服务，从而保证了事物中的所有命令被原子的执行。  
　　　　2). 和关系型数据库中的事务相比，在Redis事务中如果有某一条命令执行失败，其后的命令仍然会被继续执行。  
　　　　3). 我们可以通过MULTI命令开启一个事务，有关系型数据库开发经验的人可以将其理解为"BEGIN TRANSACTION"语句。在该语句之后执行的命令都将被视为事务之内的操作，最后我们可以通过执行EXEC/DISCARD命令来提交/回滚该事务内的所有操作。这两个Redis命令可被视为等同于关系型数据库中的COMMIT/ROLLBACK语句。  
　　　　4). 在事务开启之前，如果客户端与服务器之间出现通讯故障并导致网络断开，其后所有待执行的语句都将不会被服务器执行。然而如果网络中断事件是发生在客户端执行EXEC命令之后，那么该事务中的所有命令都会被服务器执行。  
　　　　5). 当使用Append-Only模式时，Redis会通过调用系统函数write将该事务内的所有写操作在本次调用中全部写入磁盘。然而如果在写入的过程中出现系统崩溃，如电源故障导致的宕机，那么此时也许只有部分数据被写入到磁盘，而另外一部分数据却已经丢失。

　　　　　　Redis服务器会在重新启动时执行一系列必要的一致性检测，一旦发现类似问题，就会立即退出并给出相应的错误提示。此时，我们就要充分利用Redis工具包中提供的redis-check-aof工具，该工具可以帮助我们定位到数据不一致的错误，并将已经写入的部分数据进行回滚。修复之后我们就可以再次重新启动Redis服务器了。

**13.WATCH命令和基于CAS的乐观锁：**

　　　在Redis的事务中，WATCH命令可用于提供CAS(check-and-set)功能。假设我们通过WATCH命令在事务执行之前监控了多个Keys，倘若在WATCH之后有任何Key的值发生了变化，EXEC命令执行的事务都将被放弃，同时返回Null multi-bulk应答以通知调用者事务执行失败。例如，我们再次假设Redis中并未提供incr命令来完成键值的原子性递增，如果要实现该功能，我们只能自行编写相应的代码。其伪码如下：  
　　val = GET mykey  
　　val = val + 1  
　　SET mykey $val  
　　以上代码只有在单连接的情况下才可以保证执行结果是正确的，因为如果在同一时刻有多个客户端在同时执行该段代码，那么就会出现多线程程序中经常出现的一种错误场景--竞态争用(race condition)。比如，客户端A和B都在同一时刻读取了mykey的原有值，假设该值为10，此后两个客户端又均将该值加一后set回Redis服务器，这样就会导致mykey的结果为11，而不是我们认为的12。为了解决类似的问题，我们需要借助WATCH命令的帮助，见如下代码：  
　　WATCH mykey  
　　val = GET mykey  
　　val = val + 1  
　　MULTI  
　　SET mykey $val  
　　EXEC  
　　和此前代码不同的是，新代码在获取mykey的值之前先通过WATCH命令监控了该键，此后又将set命令包围在事务中，这样就可以有效的保证每个连接在执行EXEC之前，如果当前连接获取的mykey的值被其它连接的客户端修改，那么当前连接的EXEC命令将执行失败。这样调用者在判断返回值后就可以获悉val是否被重新设置成功。

**14.redis持久化的几种方式**

1、快照（snapshots）  
　　缺省情况情况下，Redis把数据快照存放在磁盘上的二进制文件中，文件名为dump.rdb。你可以配置Redis的持久化策略，例如数据集中每N秒钟有超过M次更新，就将数据写入磁盘；或者你可以手工调用命令SAVE或BGSAVE。  
　　工作原理  
　　． Redis forks.  
　　． 子进程开始将数据写到临时RDB文件中。  
　　． 当子进程完成写RDB文件，用新文件替换老文件。  
　　． 这种方式可以使Redis使用copy-on-write技术。  
2、AOF  
　　快照模式并不十分健壮，当系统停止，或者无意中Redis被kill掉，最后写入Redis的数据就会丢失。这对某些应用也许不是大问题，但对于要求高可靠性的应用来说，  
　　Redis就不是一个合适的选择。  
　　Append-only文件模式是另一种选择。  
　　你可以在配置文件中打开AOF模式  
3、虚拟内存方式  
　　当你的key很小而value很大时,使用VM的效果会比较好.因为这样节约的内存比较大.  
　　当你的key不小时,可以考虑使用一些非常方法将很大的key变成很大的value,比如你可以考虑将key,value组合成一个新的value.  
　　vm-max-threads这个参数,可以设置访问swap文件的线程数,设置最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的.可能会造成比较长时间的延迟,但是对数据完整性有很好的保证.

自己测试的时候发现用虚拟内存性能也不错。如果数据量很大，可以考虑分布式或者其他数据库

**15.redis的缓存失效策略和主键失效机制**

作为缓存系统都要定期清理无效数据，就需要一个主键失效和淘汰策略.  
　　在Redis当中，有生存期的key被称为volatile。在创建缓存时，要为给定的key设置生存期，当key过期的时候（生存期为0），它可能会被删除。  
　　1、影响生存时间的一些操作  
　　生存时间可以通过使用 DEL 命令来删除整个 key 来移除，或者被 SET 和 GETSET 命令覆盖原来的数据，也就是说，修改key对应的value和使用另外相同的key和value来覆盖以后，当前数据的生存时间不同。  
　　比如说，对一个 key 执行INCR命令，对一个列表进行LPUSH命令，或者对一个哈希表执行HSET命令，这类操作都不会修改 key 本身的生存时间。另一方面，如果使用RENAME对一个 key 进行改名，那么改名后的 key的生存时间和改名前一样。  
　　RENAME命令的另一种可能是，尝试将一个带生存时间的 key 改名成另一个带生存时间的 another\_key ，这时旧的 another\_key (以及它的生存时间)会被删除，然后旧的 key 会改名为 another\_key ，因此，新的 another\_key 的生存时间也和原本的 key 一样。使用PERSIST命令可以在不删除 key 的情况下，移除 key 的生存时间，让 key 重新成为一个persistent key 。  
　　2、如何更新生存时间  
　　可以对一个已经带有生存时间的 key 执行EXPIRE命令，新指定的生存时间会取代旧的生存时间。过期时间的精度已经被控制在1ms之内，主键失效的时间复杂度是O（1），  
　　EXPIRE和TTL命令搭配使用，TTL可以查看key的当前生存时间。设置成功返回 1；当 key 不存在或者不能为 key 设置生存时间时，返回 0 。  
　　最大缓存配置  
　　在 redis 中，允许用户设置最大使用内存大小  
　　server.maxmemory  
　　默认为0，没有指定最大缓存，如果有新的数据添加，超过最大内存，则会使redis崩溃，所以一定要设置。redis 内存数据集大小上升到一定大小的时候，就会实行数据淘汰策略。  
　　redis 提供 6种数据淘汰策略：  
　　． volatile-lru：从已设置过期时间的数据集（server.db[i].expires）中挑选最近最少使用的数据淘汰  
　　． volatile-ttl：从已设置过期时间的数据集（server.db[i].expires）中挑选将要过期的数据淘汰  
　　． volatile-random：从已设置过期时间的数据集（server.db[i].expires）中任意选择数据淘汰  
　　． allkeys-lru：从数据集（server.db[i].dict）中挑选最近最少使用的数据淘汰  
　　． allkeys-random：从数据集（server.db[i].dict）中任意选择数据淘汰  
　　． no-enviction（驱逐）：禁止驱逐数据  
　　注意这里的6种机制，volatile和allkeys规定了是对已设置过期时间的数据集淘汰数据还是从全部数据集淘汰数据，后面的lru、ttl以及random是三种不同的淘汰策略，再加上一种no-enviction永不回收的策略。  
　　使用策略规则：  
　　1、如果数据呈现幂律分布，也就是一部分数据访问频率高，一部分数据访问频率低，则使用allkeys-lru  
　　2、如果数据呈现平等分布，也就是所有的数据访问频率都相同，则使用allkeys-random  
　　三种数据淘汰策略：  
　　ttl和random比较容易理解，实现也会比较简单。主要是Lru最近最少使用淘汰策略，设计上会对key 按失效时间排序，然后取最先失效的key进行淘汰

**16.redis 最适合的场景**

Redis最适合所有数据in-momory的场景，虽然Redis也提供持久化功能，但实际更多的是一个disk-backed的功能，跟传统意义上的持久化有比较大的差别，那么可能大家就会有疑问，似乎Redis更像一个加强版的Memcached，那么何时使用Memcached,何时使用Redis呢?  
如果简单地比较Redis与Memcached的区别，大多数都会得到以下观点：  
　　1 、Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。  
　　2 、Redis支持数据的备份，即master-slave模式的数据备份。  
　　3 、Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。  
（1）、会话缓存（Session Cache）

　　最常用的一种使用Redis的情景是会话缓存（session cache）。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（Sorted Set）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！（不，这是真的，你可以去核

　　实）。  
　　Redis提供的所有特性中，我感觉这个是喜欢的人最少的一个，虽然它为用户提供如果此多功能。