

Redis 学习记录

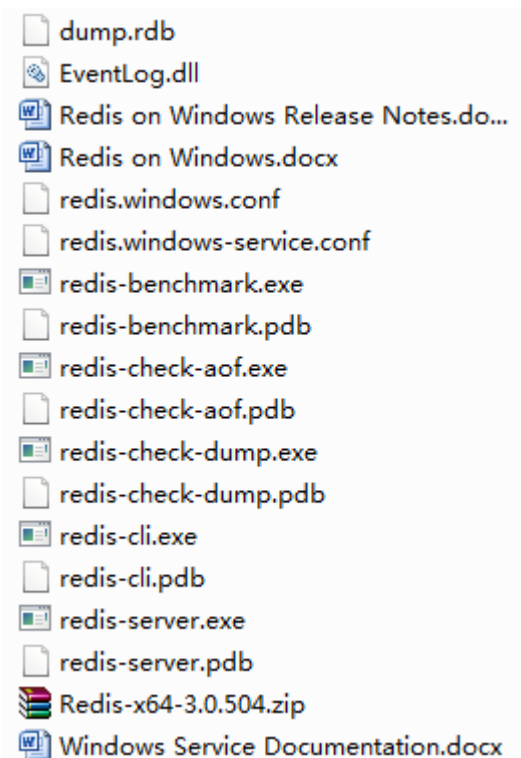
1、安装方式

官方没有发布支持 windows 平台的安装包,这里一般选择从 github 下载源码进行安装,下载地址为:

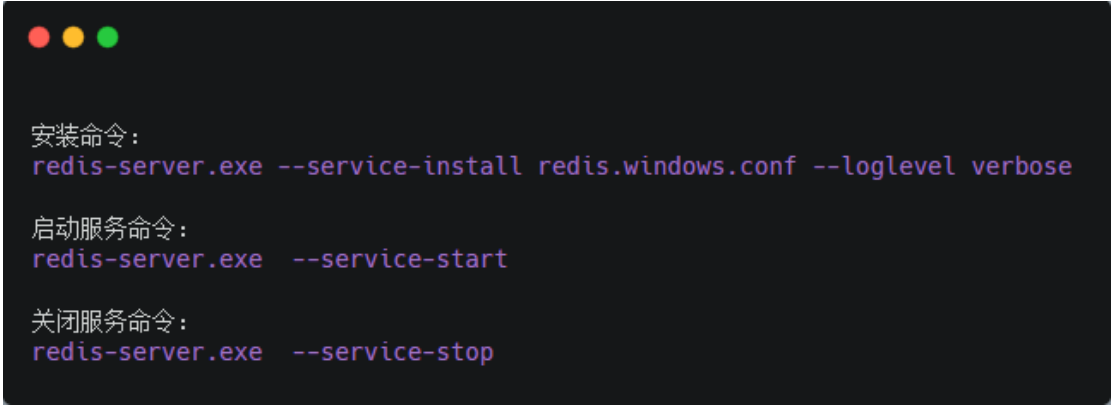
<https://github.com/MicrosoftArchive/redis/releases>



下载红色线框里面的版本解压缩即可,目录结构如下:



命令窗口安装方式如下:



```
安装命令：  
redis-server.exe --service-install redis.windows.conf --loglevel verbose  
  
启动服务命令：  
redis-server.exe --service-start  
  
关闭服务命令：  
redis-server.exe --service-stop
```

2、启动报错问题解决

如果在命令行窗口输入 `redis-server.exe redis.windows.conf` 指令执行报错误 [13164] 27 Dec 20:57:07.820 # Creating Server TCP listening socket 127.0.0.1:6379: bind: No error。

那么可以输入如下的命令依次执行：

第一条指令：`redis-cli.exe`

第二条指令：`shutdown`

第三条指令：`exit`

最后执行：`redis-server.exe redis.windows.conf`

如果要练习客户端命令等，需要另外启动一个命令窗口，同样在同一个目录里，输入命令 `Redis-cli.exe` 就可以连接上了（如果没有改默认配置的端口号），如果改了，就输入更加详细的命令：`redis-cli.exe -h 127.0.0.1 -p 6379` 就可以，根据这种格式对应改自己的修改的配置即可。

3、常用 Redis 操作命令

卸载服务: `redis-server --service-uninstall`

开启服务: `redis-server --service-start`

停止服务: `redis-server --service-stop`

重命名服务: `redis-server --service-name name`

4、 以下将会安装并启动三个不同的 Redis 实例作服务:

`redis-server --service-install --service-name redisService1 --port 10001`

`redis-server --service-start --service-name redisService1`

`redis-server --service-install --service-name redisService2 --port 10002`

`redis-server --service-start --service-name redisService2`

`redis-server --service-install --service-name redisService3 --port 10003`

`redis-server --service-start --service-name redisService3`

启动服务:

`redis-server --service-start`

精简模式:

`redis-cli.exe`

指定模式:

`redis-cli.exe -h 127.0.0.1 -p 6379 -a requirepass`

(-h 服务器地址 -p 指定端口号 -a 连接数据库的密码[可以在 redis.windows.conf 中配置], 默认无密码)

5.什么是 redis?

Redis 是一个基于内存的高性能 key-value 数据库。

6.Redis 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。

7.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

1) String

常用命令：set/get/decr/incr/mget 等；

应用场景：String 是最常用的一种数据类型，普通的 key/value 存储都可以归为此类；

实现方式：String 在 redis 内部存储默认就是一个字符串，被 redisObject 所引用，当遇到 incr、decr 等操作时会转成数值型进行计算，此时 redisObject 的 encoding 字段为 int。

2) Hash

常用命令：hget/hset/hgetall 等

应用场景：我们要存储一个用户信息对象数据，其中包括用户 ID、用户姓名、年龄和生日，通过用户 ID 我们希望获取该用户的姓名或者年龄或者生日；

实现方式：Redis 的 Hash 实际是内部存储的 Value 为一个 HashMap，

并提供了直接存取这个 Map 成员的接口。Key 是用户 ID, value 是一个 Map。这个 Map 的 key 是成员的属性名, value 是属性值。这样对数据的修改和存取都可以直接通过其内部 Map 的 Key(Redis 里称内部 Map 的 key 为 field), 也就是通过 key(用户 ID) + field(属性标签) 就可以操作对应属性数据。

当前 HashMap 的实现有两种方式: 当 HashMap 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储, 而不会采用真正的 HashMap 结构, 这时对应的 value 的 redisObject 的 encoding 为 zipmap, 当成员数量增大时会自动转成真正的 HashMap, 此时 encoding 为 ht。

3) List

常用命令: lpush/rpush/lpop/rpop/lrange 等;

应用场景: Redis list 的应用场景非常多, 也是 Redis 最重要的数据结构之一, 比如 twitter 的关注列表, 粉丝列表等都可以用 Redis 的 list 结构来实现;

实现方式: Redis list 的实现为一个双向链表, 即可以支持反向查找和遍历, 更方便操作, 不过带来了部分额外的内存开销, Redis 内部的很多实现, 包括发送缓冲队列等也都是用的这个数据结构。

4) Set

常用命令: sadd/spop/smembers/sunion 等;

应用场景: Redis set 对外提供的功能与 list 类似是一个列表的功能,

特殊之处在于 set 是可以自动排重的，当你需要存储一个列表数据，又不希望出现重复数据时，set 是一个很好的选择，并且 set 提供了判断某个成员是否在一个 set 集合内的重要接口，这个也是 list 所不能提供的；

实现方式：set 的内部实现是一个 value 永远为 null 的 HashMap，实际就是通过计算 hash 的方式来快速排重的，这也是 set 能提供判断一个成员是否在集合内的原因。

5) Sorted Set

常用命令：zadd/zrange/zrem/zcard 等；

应用场景：Redis sorted set 的使用场景与 set 类似，区别是 set 不是自动有序的，而 sorted set 可以通过用户额外提供一个优先级(score)的参数来为成员排序，并且是插入有序的，即自动排序。当你需要一个有序的并且不重复的集合列表，那么可以选择 sorted set 数据结构，比如 twitter 的 public timeline 可以以发表时间作为 score 来存储，这样获取时就是自动按时间排好序的。

实现方式：Redis sorted set 的内部使用 HashMap 和跳跃表(SkipList)来保证数据的存储和有序，HashMap 里放的是成员到 score 的映射，而跳跃表里存放的是所有的成员，排序依据是 HashMap 里存的 score，使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

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？

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 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？

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。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。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 适用于的场景？

Redis 最适合所有数据 in-memory 的场景，如：

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。

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。

3.队列

Redis 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。推荐阅读：Redis 的 8 大应用场景。

7、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 改名为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-eviction** (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-eviction** 永不回收的策略。

使用策略规则:

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** 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

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 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

10.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 Redis 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：

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。

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。

11、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。

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，如果要想实现该功能，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.使用过 Redis 分布式锁么，它是怎么回事？

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。

这时候对方会告诉你说你回答得不错，然后接着问如果在 `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？

这时候你要给予惊讶的反馈：唉，是喔，这个锁就永远得不到释放了。

紧接着你需要抓一抓自己得脑袋，故作思考片刻，好像接下来的结果是你主动思考出来的，然后回答：我记得 `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！对方这时会显露笑容，心里开始默念：嗯，这小子还不错。

15.假如 Redis 里面有 1 亿个 key，其中有 10w 个 key 是以某个固定的已知的前缀开头的，如果将它们全部找出来？

使用 keys 指令可以扫出指定模式的 key 列表。

对方接着追问：如果这个 redis 正在给线上的业务提供服务，那使用 keys 指令会有什么问题？

这个时候你要回答 redis 关键的一个特性：redis 的单线程的。keys 指令会导致线程阻塞一段时间，线上服务会停顿，直到指令执行完毕，服务才能恢复。这个时候可以使用 scan 指令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端做一次去重就可以了，但是整体所花费的时间会比直接用 keys 指令长。

16.使用过 Redis 做异步队列么，你是怎么用的？

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。

如果对方追问可不可以不用 sleep 呢？list 还有个指令叫 blpop，在没有消息的时候，它会阻塞住直到消息到来。

如果对方追问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。

如果对方追问 pub/sub 有什么缺点？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。

如果对方追问 redis 如何实现延时队列？我估计现在你很想把面试官一棒打死如果你手上有一根棒球棍的话，怎么问的这么详细。但是你很克制，然后神态自若的回答道：使用 sortedset，拿时间戳作为 score，消息内容作为 key 调用 zadd 来生产消息，消费者用 zrangebyscore 指令获取 N 秒之前的数据轮询进行处理。

到这里，面试官暗地里已经对你竖起了大拇指。但是他不知道的是此刻你却竖起了中指，在椅子背后。

17.如果有大量的 key 需要设置同一时间过期，一般需要注意什么？

如果大量的 key 过期时间设置的过于集中，到过期的那个时间点，redis 可能会出现短暂的卡顿现象。一般需要在时间上加一个随机值，使得过期时间分散一些。

18.Redis 如何做持久化的？

bgsave 做镜像全量持久化，aof 做增量持久化。因为 bgsave 会耗费较长时间，不够实时，在停机的时候会导致大量丢失数据，所以需要 aof 来配合使用。在 redis 实例重启时，会使用 bgsave 持久化文件

重新构建内存，再使用 aof 重放近期的操作指令来实现完整恢复重启之前的状态。

对方追问那如果突然机器掉电会怎样？取决于 aof 日志 sync 属性的配置，如果不要求性能，在每条写指令时都 sync 一下磁盘，就不会丢失数据。但是在高性能的要求下每次都 sync 是不现实的，一般都使用定时 sync，比如 1s1 次，这个时候最多就会丢失 1s 的数据。

对方追问 bgsave 的原理是什么？你给出两个词汇就可以了，fork 和 cow。fork 是指 redis 通过创建子进程来进行 bgsave 操作，cow 指的是 copy on write，子进程创建后，父子进程共享数据段，父进程继续提供读写服务，写脏的页面数据会逐渐和子进程分离开来。

19.Pipeline 有什么好处，为什么要用 pipeline？

可以将多次 IO 往返的时间缩减为一次，前提是 pipeline 执行的指令之间没有因果相关性。使用 redis-benchmark 进行压测的时候可以发现影响 redis 的 QPS 峰值的一个重要因素是 pipeline 批次指令的数目。

20.Redis 的同步机制了解么？

Redis 可以使用主从同步，从从同步。第一次同步时，主节点做一次 bgsave，并同时后续修改操作记录到内存 buffer，待完成后将

rd b 文件全量同步到复制节点，复制节点接受完成后将 rdb 镜像加载到内存。加载完成后，再通知主节点将期间修改的操作记录同步到复制节点进行重放就完成了同步过程。

21.是否使用过 Redis 集群，集群的原理是什么？

Redis Sentinel 着眼于高可用，在 master 宕机时会自动将 slave 提升为 master，继续提供服务。

Redis Cluster 着眼于扩展性，在单个 redis 内存不足时，使用 Cluster 进行分片存储。