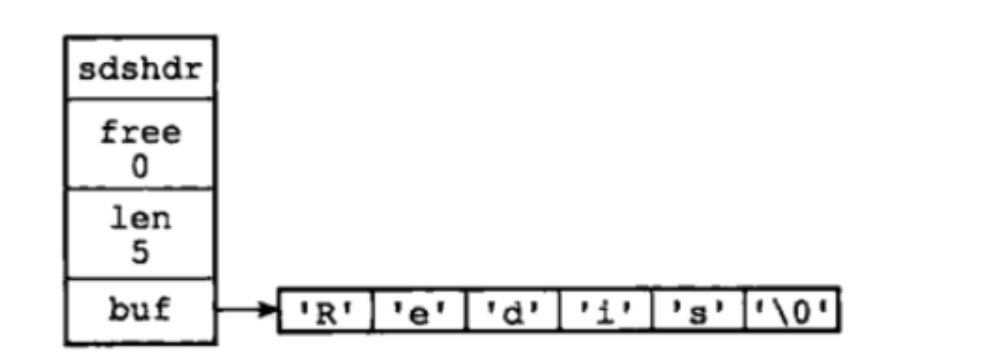
1. Redis 底层数据结构原理
2. String

Redis的底层是使用C语言来写的，它自己构建了一种简单动态字符串的抽象模型（SDS）用作redis的默认字符串表示。



我们看上面对于 SDS 数据类型的定义：

　1、len 保存了SDS保存字符串的长度

　2、buf[] 数组用来保存字符串的每个元素

　　 3、free j记录了 buf 数组中未使用的字节数量

好处：

1、常数复杂度获取字符串长度

由于 len 属性的存在，我们获取 SDS 字符串的长度只需要读取 len 属性，时间复杂度为 O(1)。而对于 C 语言，获取字符串的长度通常是经过遍历计数来实现的，时间复杂度为 O(n)。通过 strlen key 命令可以获取 key 的字符串长度。

　 ②、杜绝缓冲区溢出

　　我们知道在 C 语言中使用 strcat 函数来进行两个字符串的拼接，一旦没有分配足够长度的内存空间，就会造成缓冲区溢出。而对于 SDS 数据类型，在进行字符修改的时候，会首先根据记录的 len 属性检查内存空间是否满足需求，如果不满足，会进行相应的空间扩展，然后在进行修改操作，所以不会出现缓冲区溢出。

　　③、减少修改字符串的内存重新分配次数

C语言由于不记录字符串的长度，所以如果要修改字符串，必须要重新分配内存（先释放再申请），因为如果没有重新分配，字符串长度增大时会造成内存缓冲区溢出，字符串长度减小时会造成内存泄露。

而对于SDS，由于len属性和free属性的存在，对于修改字符串SDS实现了空间预分配和惰性空间释放两种策略：

　　1、空间预分配：对字符串进行空间扩展的时候，扩展的内存比实际需要的多，这样可以减少连续执行字符串增长操作所需的内存重分配次数。

　　2、惰性空间释放：对字符串进行缩短操作时，程序不立即使用内存重新分配来回收缩短后多余的字节，而是使用 free 属性将这些字节的数量记录下来，等待后续使用。（当然SDS也提供了相应的API，当我们有需要时，也可以手动释放这些未使用的空间。）

1. List

由于c语言没有内置链表的数据结构，所以redis自己构建了链表的实现。

　Redis链表特性：

　　①、双端：链表具有前置节点和后置节点的引用，获取这两个节点时间复杂度都为O(1)。

　　②、无环：表头节点的 prev 指针和表尾节点的 next 指针都指向 NULL,对链表的访问都是以 NULL 结束。

　　③、带链表长度计数器：通过 len 属性获取链表长度的时间复杂度为 O(1)。

　　④、多态：链表节点使用 void\* 指针来保存节点值，可以保存各种不同类型的值

1. Hash

哈希表是由table组成的，key用来保存键，val 属性用来保存值，值可以是一个指针，（也可以是uint64\_t整数，也可以是int64\_t整数）。这里还有一个指向下一个哈希表节点的指针，

我们知道哈希表最大的问题是存在哈希冲突，如何解决哈希冲突，有开放地址法和链地址法。这里采用的便是链地址法，通过next这个指针可以将多个哈希值相同的键值对连接在一起，用来解决哈希冲突。

扩容和收缩：当哈希表保存的键值对太多或者太少时，就要通过 rerehash(重新散列）来对哈希表进行相应的扩展或者收缩。具体步骤：

　　　　　　1、如果执行扩展操作，会基于原哈希表创建一个大小等于 ht[0].used\*2n 的哈希表（也就是每次扩展都是根据原哈希表已使用的空间扩大一倍创建另一个哈希表）。相反如果执行的是收缩操作，每次收缩是根据已使用空间缩小一倍创建一个新的哈希表。

　　　　　　2、重新利用上面的哈希算法，计算索引值，然后将键值对放到新的哈希表位置上。

　　　　　　3、所有键值对都迁徙完毕后，释放原哈希表的内存空间。

但是这种扩容操作会在redis数据量比较大的时候连续·占用很久的时间，所以提出了渐进式hash。

为了避免rehash对服务器性能造成影响，服务器不是一次性将ht[0]里面所有的值对都全部rehash到ht[1]，而是分多次，渐进式的将ht[0]里面的值对慢慢地rehash到ht[1]。

详细步骤：

1）为ht[1]分配空间，让字典同时持有ht[1]和ht[0]两个哈希表。

2）在字典中维持一个索引计数器变量rehashidx，并将它的值设置为0，表示rehash工作正式开始

3）在Rehash期间，每次对字典的执行添加，删除，查找或者更新操作时，程序除了执行指定的操作外，还会顺带将ht[0]哈希表在rehashidex索引上的所有键值对rehash到[1]，当rehash完成，程序将rehashidx属性的值增一。

4）随着字典操作的不断执行，最终在某个时间点上，ht[0]的所有键值对都会被rehash至ht[1]，这时程序将rehashidex属性的值设置为-1，表示rehash操作已完成。

渐进式Rehash的好处在于它采取分而治之的方式，将rehash键值对所需的计算工作均摊到了对字典的每个添加，删除，查找和更新操作上，从而避免了集中式rehash而带来的庞大计算量。

1. Set

　　　　跳跃表（skiplist）是一种有序数据结构，它通过在每个节点中维持多个指向其他节点的指针，从而达到快速访问节点的目的。跳跃表是一种随机化的数据,跳跃表以有序的方式在层次化的链表中保存元素，效率和平衡树媲美 ——查找、删除、添加等操作都可以在对数期望时间下完成，并且比起平衡树来说，跳跃表的实现要简单直观得多。

1. Zset

向有序集合中添加score和element 时间复杂度 o(logN)

Zset 查询个数时间复杂度O（1），获取element的分数 时间复杂度o(1)

zrange key start end [withscores] 查询有序集合成指定排名区间内的成员，下标从0开始 时间复杂度o(log(n)+m)

1. Redis的持久化方式？

Rdb（快照）（和aof。

快照的核心原理就是把redis在某个时间内存内的所有数据都写入硬盘。

1. 用户发送bgsave命令，此时redis会fork一个子进程，子进程负责生成硬盘文件，父进程负责继续接收命令
2. 用户发送save命令，使用了该命令后到系统创建快照完成之前系统不会再接收新的命令。

Save 60 1000；如果系统满足60秒内有1000次写入的这个条件，系统就会自动调用bgsave。

bgsave命令是针对save阻塞问题做的优化。因此Redis内部所有涉及到RDB操作都采用bgsave的方式，而save命令可以废弃。

在大量数据的时候，rdb会导致redis的服务暂停很久的时间，还有可能在更新的过程中系统崩溃了。‘

AOF：aof就是把所有的redis数据进行的写操作的命令都记录在硬盘上，aof的优势在于就算出现了问题，最多只会丢失一秒的数据。

Aof在系统重启的时候从里面读命令，如果aof文件太大，读命令可能也就要很久。

1. Redis与memcached的区别与比较/
2. 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；而后者只支持string的。。
3. redis支持数据的备份，即master-slaver模式的数据备份
4. redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载使用
5. redist 的速度快很多
6. Memcached是多线程，非阻塞io复用的网络模型；redis使用的是单线程的io复用模型
7. Redis的主从同步

旧版复制:sync

初次复制：从服务器之前没有复制过任何服务器，或者本次复制的服务器与上次复制的服务器不同。当断线后，处于命令传播阶段的主从复制中断了复制，但从服务器通过自动重新连接重连接上了主服务器，并继续复制主服务器。

但是这种对于断线的复制效率不高；为了让主服务器补足一小部分data，却让主服务器执行了一次sync命令。

因为每次执行sync命令：

1. 主服务器都会执行bgsave命令生成rdb文件，这个生成的操作十分消耗大量cpu资源；i/o资源和内存资源。
2. 主服务器需要将自己的生成的rdb文件发送给从服务器，这个发送操作会占用大量网络资源。
3. 从服务器载入rdb文件会因为阻塞而无法处理命令请求

新版复制：psync

完成同步：和sync一致

部分同步：允许断线后重新连接服务器可以将主从断开期间的写命令发送给从服务器。

1. Redis 缓存击穿

当大量请求不存在缓存中的数据，导致请求短时间直接落在了数据库上，导致数据库异常。

解决：

1. 接口限流与熔断降级
2. 使用互斥锁排队

当key获取value为空的时候锁上；从数据库中load数据后再释放锁。

Redis缓存雪崩

缓存同一时间内大量键过期；接下来一大波请求瞬间落在数据库中导致连接异常。

解决：

1. 加锁
2. 建立备份缓存；缓存A和缓存B。A设置超时时间，B不设置超时时间，如果从A中读取不到缓存就从B中读
3. 设置缓存超时时间时加上一个随机的时间长度。
4. redis中1亿个key，其中有10w个key以某个固定的前缀开头？
5. 使用keys指令可以扫描出制定模式key列表
6. 如果线上提供服务，使用keys指令会导致线程阻塞一段时间。这个时候可以使用scan指令，可以无阻塞取，但是有一定的概率会重复，我们可以在客户端中在做一次去重的工作
7. Redis怎么做异步队列?

一般使用list结构作为队列，rpush生产消息，lpop消费消息；当lpop没有消息时，要适当sleep一会再重试。也可以使用blpop，在没有消息时，会阻塞直到消息来。

1. Redis怎么实现延迟队列呢？

可以使用zset，拿时间戳作为socre；消息内容作为key|调用zadd来生产消息。消费者用zrangescore指令去获取N秒之前数据轮询处理。