Redis单线程：

主要是指网络IO和键值对读写是由一个线程完成的；其他功能，如持久化、异步删除、集群数据同步是由其他线程完成的

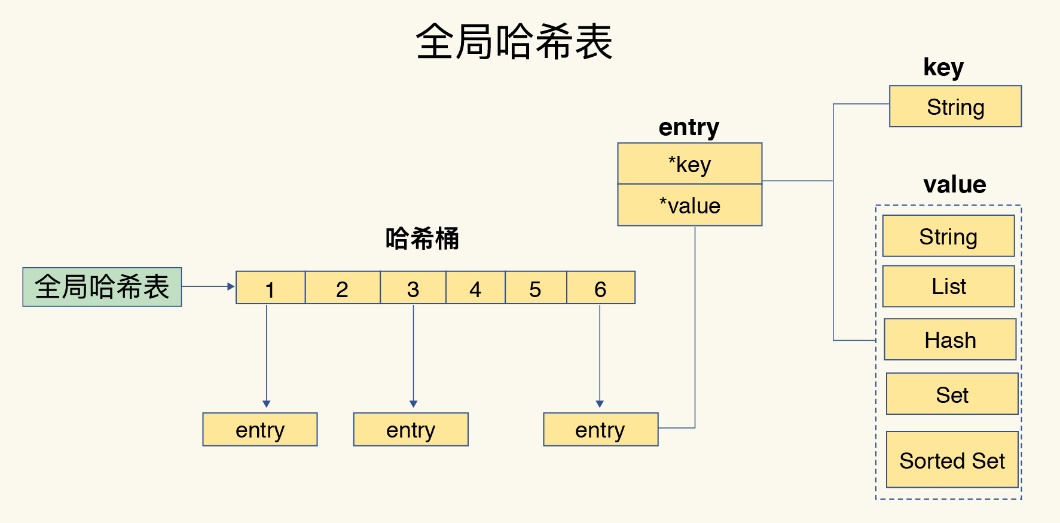
为什么使用单线程

多线程编程模式存在共享数据并发访问控制问题，需要获取访问共享资源的互斥锁；

为什么这么快？

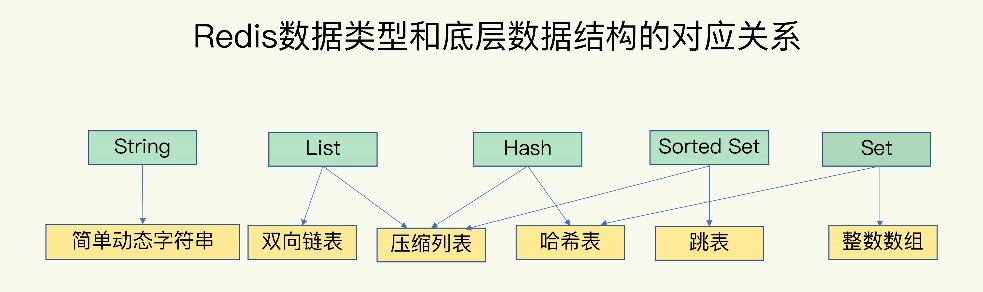
1. 数据结构

Redis使用**全局哈希表**保存所有键值对

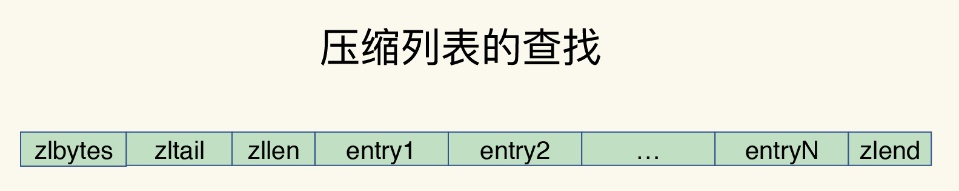


渐进式rehash：创建hash表2并分配2倍内存，将hash表1数据copy到hash表2中，释放hash表1；

在copy过程中，Redis仍然正常处理客户端请求，每处理一个请求时，就会处理一个索引下的所有entry到表2中；避免大量copy造成的开销，保证请求访问速



压缩列表



Zlbytes：列表长度

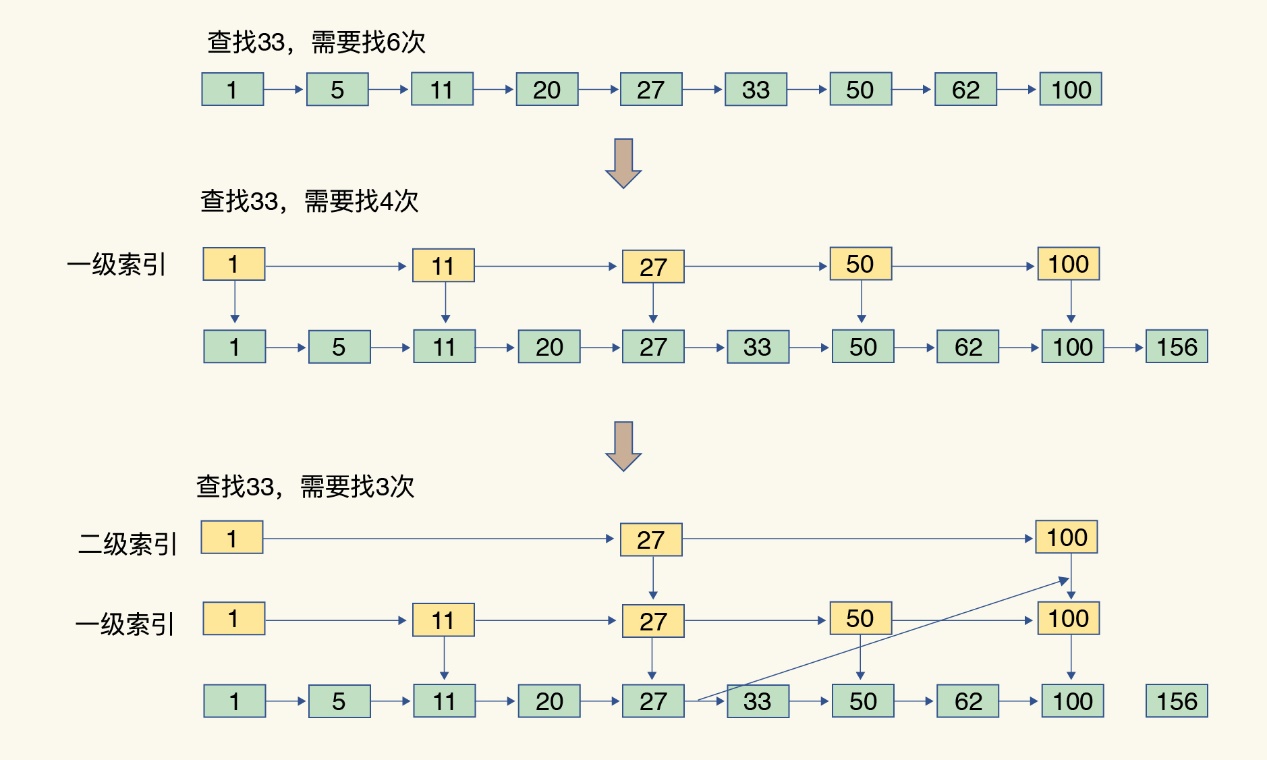
Zltail：列表尾部偏移量

Zllen：entry个数

Zlend：表示列表结束

优势：查找第一位元素和最后一个元素，可以通过表头信息直接O(1)查询

跳跃表：

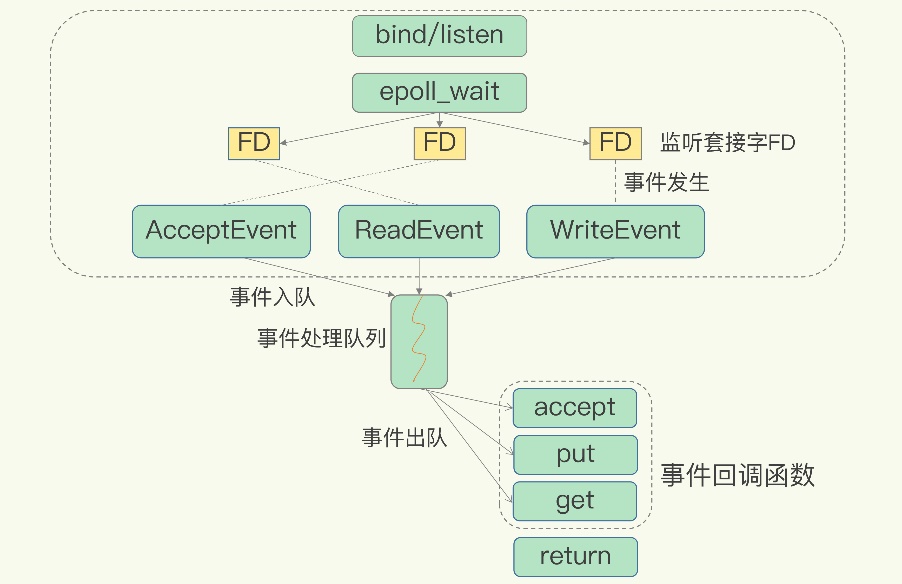


增加了多级索引，通过索引位置的几个跳转，实现数据快速定位



1. 多路复用机制（select/epoll）：在网络IO中能够并发处理大量的客户端请求，实现高吞吐率；该机制允许内核中，同时存在多个监听套接字和已连接套接字。内核会一直监听这些套接字上的连接请求或数据请求。Redis线程不会阻塞在某一个特定的监听或者套接字上（不会阻塞在某一客户端的请求处理上）；基于事件的回调机制（针对不同事件的发生，调用相应的处理函数）

优化点：避免accept/send/recv潜在的网络IO阻塞点

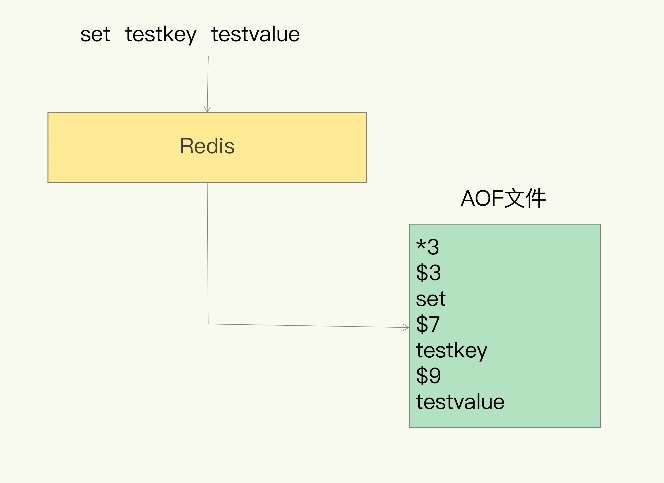


Redis持久方式

AOF

在主线程中执行

文本形式，记录内容是每一条命令



采用写后日志（与数据库相反）：

Redis先执行命令，把数据写入内存中，然后才写日志

原因：

1. 为了避免额外的开销，Redis写AOF时，不会做语法检查，所以如果写前日志的形式，日志中就可能会有错误的命令，导致无法使用；
2. 不会造成写操作的阻塞（数据直接存到了redis中）

风险点：

1. 刚执行完一个命令就宕机，造成aof日志缺失。
2. 因为Aof主线程中执行，如果磁盘写压力大，导致写磁盘慢，后续操作无法执行。

回写策略：控制一个写命令执行完后AOF日志写回磁盘的时机

AppendFsync可选值：

Always：同步回写，每次命令执行完，马上回写

Everysec：每秒回写，每个命令执行完，先把日志写到AOF的内存缓冲区，每个1s写入磁盘

No：操作系统控制，每个命令执行完，先把日志写到AOF的内存缓冲区，操作系统控制何时写入磁盘中



日志文件过大处理

AOF重写机制：在重写时，Redis针对现状，创建一个新AOF文件，读取所有数据库中键值对，写入新文件中；通过另一个文件，保证在此期间的写操作不会丢失

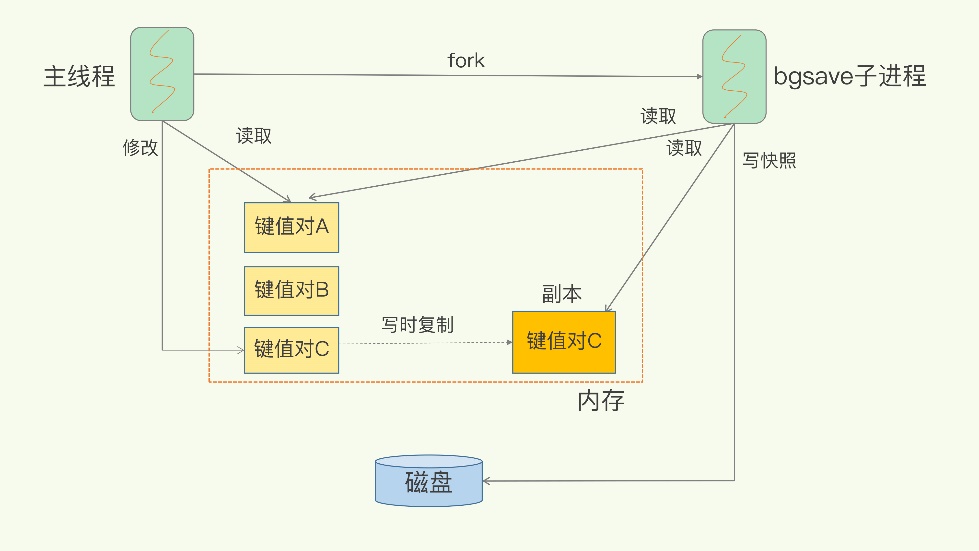
由后台子进程bgrewriteaof完成，避免阻塞主线程；

RDB：快照

Bgsave操作：创建一个子进程，专门用于写入RDB文件，避免阻塞主线程

写时复制（COW：copy on write），在执行快照同时，正常处理写操作。写时复制机制保证快照期间数据可修改

Bgsave子进程由主线程fork出来，可以共享主线程所有数据。



Redis4.0使用AOF和RDB混合方式。

主从数据同步：

一，全量同步：

**第一阶段**：主从库建立连接、协商同步过程，为全量同步做准备。从库给主库发送psync命令，表示要进行数据同步，主库根据psync命令启动复制。psync命令包含了主库的runId和复制进度的offset

runId：每个redis实例启动时自动生产的一个随机Id，第一次复制不知道则为？

offset：此时为-1， 因为全量

主库收到命令后，会用FULLRESTNC响应命令带上两个参数，主库runId和复制进度offset。

**第二阶段**：主库将所有数据同步给从库。从库收到数据后，在本地完成数据加载。这个过程依赖于内存快照生成的RDB文件。

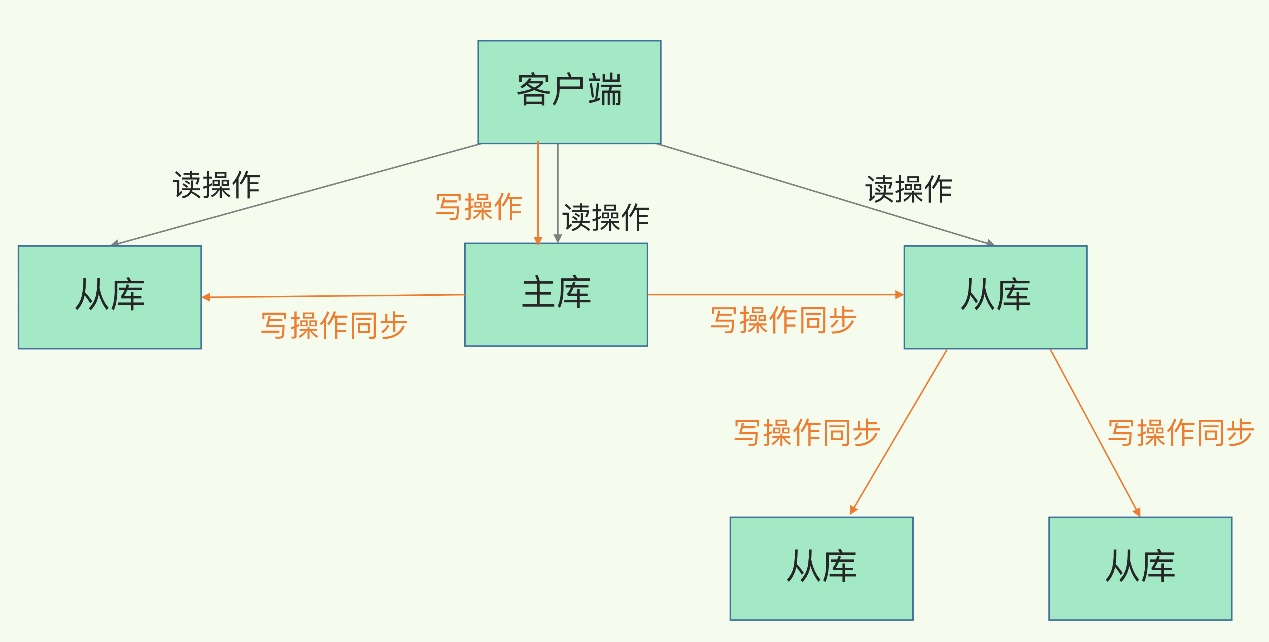
主库执行 bgsave 命令，生成 RDB 文件，接着将文件发给从库。从库接收到 RDB 文件后，会先清空当前数据库，然后加载 RDB 文件。这是因为从库在通过 replicaof 命令开始和主库同步前，可能保存了其他数据。为了避免之前数据的影响，从库需要先把当前数据库清空。

为了保证主从库的数据一致性，主库会在内存中用专门的 replication buffer，记录 RDB 文件生成后收到的所有写操作。

**第三阶段**：主库将replication buffer中的修改操作发给从库，从库再执行。

如果从库过多，主从数据同步时主库fork生成rdb时间过多，fork操作会阻塞主线程处理正常请求，如何分担主库压力？

解决：使用“主——从——从”模式



二，基于长连接的命令传播，可以避免频繁建立连接的开销

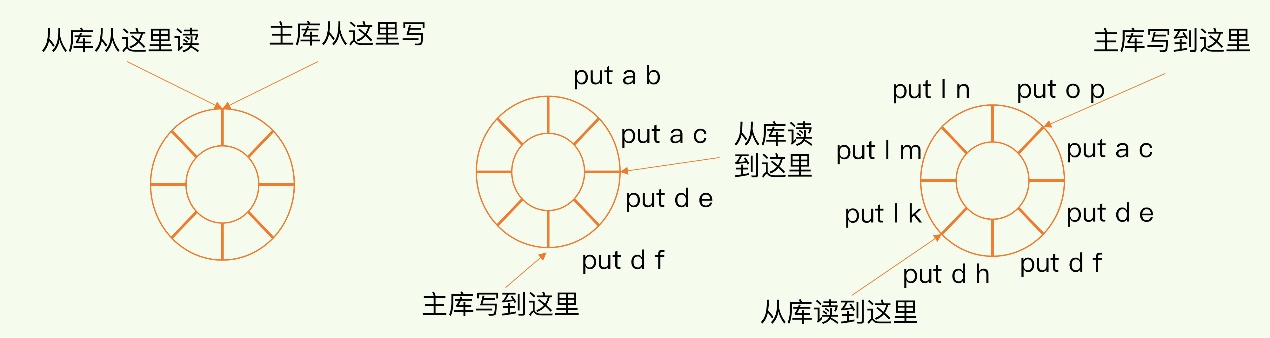
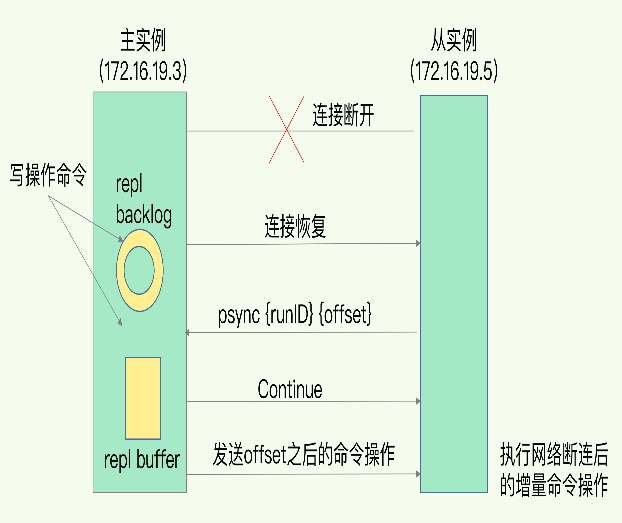
三，增量同步

Redis2.8后，网络断开，主从会采用增量复制的方式同步。

**repl\_backlog\_buffer 环形缓冲区：主库会记录自己写到的位置，从库会记录自己读到的位置**

主从断开连接后，主库会把期间的写操作命令，写入replication buffer中，同时也会写在repl\_backlog\_buffer中。

主从库的连接恢复之后，从库首先会给主库发送 psync 命令，并把自己当前的 slave\_repl\_offset 发给主库，主库会判断自己的 master\_repl\_offset 和 slave\_repl\_offset 之间的差距。在网络断连阶段，主库可能会收到新的写操作命令，所以，一般来说，master\_repl\_offset 会大于 slave\_repl\_offset。此时，主库只用把 master\_repl\_offset 和 slave\_repl\_offset 之间的命令操作同步给从库就行。



为避免主库写速度大于从库读速度，可调整参数repl\_backlog\_size，设置缓冲空间。

Redis性能

IO性能

1. 任意一个请求在server中一旦耗时，就会影响整个server的性能，后面的请求要等待，包括：
   1. 操作bigkey：写入bigkey分配内存时需消耗更多时间，删除bigkey释放空间一样
   2. 使用复杂度过高的命令：sort/sunion/zunionsort，或者基数很大的范围查询lrange key
   3. 大量key集中过期：Redis过期机制在主线程中执行，执行任务可能会耗时在删除过期key的过程中
   4. 淘汰策略：Redis淘汰策略在主线程执行，当redis内存达到上限后，每次写入都需要删除一些过期key
   5. AOF刷盘开启always机制：写磁盘速度远比写内存慢
   6. 主从全量同步生成RDB：虽然采用了fork子进程生成快照数据，但fork一瞬间会造成整个线程阻塞，且实例越大，阻塞时间越长
2. 并发量非常大的时候

解决：

1. 业务人员规避，Redis4.0推出lazy-free机制，把bigkey释放内存放在异步线程处理
2. Redis在6.0推出了多线程（但我还没有用过）

Redis实例：

数据库实例不宜太大，几个GB级别合适。减少RDB文件生成，传输，和重新加载的开销。