# Redis的基本使用

Redis的单线程和高性能

Redis是单线程吗

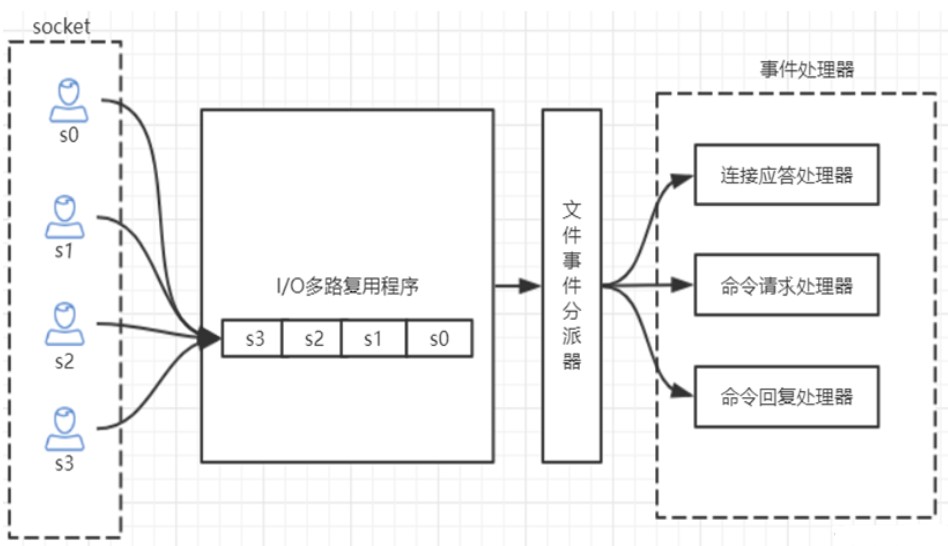
Redis的单线程主要是指Redis的网络IO和键值对读写是由一个线程来完成的，这也是Redis对外提供键值存储服务的主要流程。但Redis的其他功能，比如持久化、异步删除、集群数据同步等，其实是由额外的线程执行的。（在6.0版本后网络IO改成了多线）

Redis 单线程为什么还能这么快

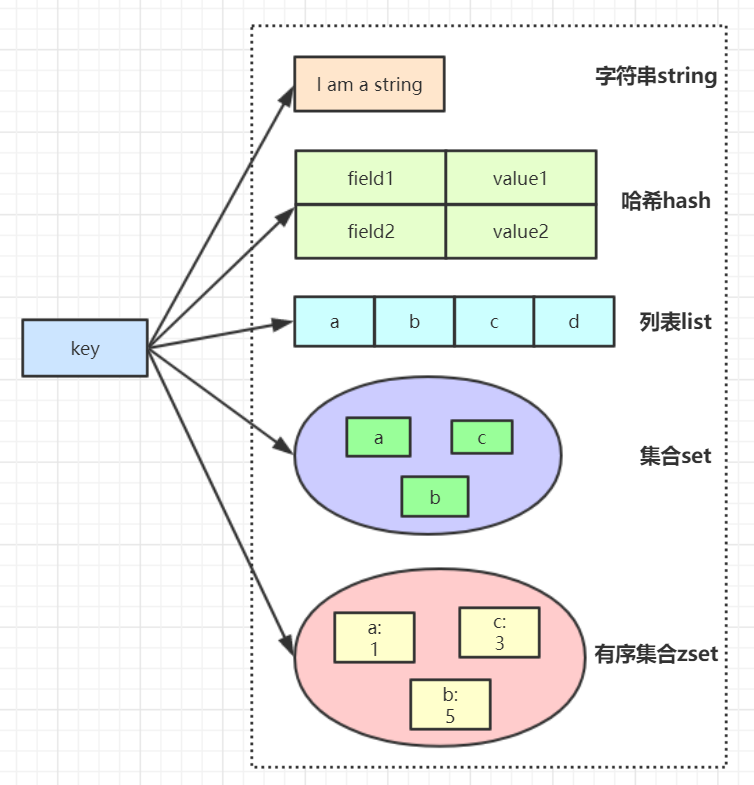
因为它所有的数据都在内存中，所有的运算都是内存级别的运算，而且单线程避免了多线程的切换性能损耗问题。正因为 Redis 是单线程，所以要小心使用 Redis 指令，对于那些耗时的指令(比如 keys)，一定要谨慎使用，一不小心就可能会导致 Redis 卡顿。

Redis 单线程如何处理那么多的并发客户端连接

Redis的IO多路复用：redis利用epoll来实现IO多路复用，将连接信息和事件放到队列中，依次放到文件事件分派器，事件分派器将事件分发给事件处理器。



五种数据结构



字符串常用操作

SET key value //存入字符串键值对

MSET key value [key value ...] //批量存储字符串键值对

SETNX key value //存入一个不存在的字符串键值对

GET key //获取一个字符串键值

MGET key [key ...] //批量获取字符串键值

DEL key [key ...] //删除一个键

EXPIRE key seconds //设置一个键的过期时间(秒)

原子加减

INCR key //将key中储存的数字值加1

DECR key //将key中储存的数字值减1

INCRBY key increment //将key所储存的值加上increment

DECRBY key decrement //将key所储存的值减去decrement

单值缓存

SET key value

GET key

对象缓存

1) SET user:1 value(json格式数据)

2) MSET user:1:name zhuge user:1:balance 1888

MGET user:1:name user:1:balance

分布式锁

SETNX product:10001 true //返回1代表获取锁成功

SETNX product:10001 true //返回0代表获取锁失败

。。。执行业务操作

DEL product:10001 //执行完业务释放锁

SET product:10001 true ex 10 nx //防止程序意外终止导致死锁

* Hash常用操作

HSET key field value //存储一个哈希表key的键值

HSETNX key field value //存储一个不存在的哈希表key的键值

HMSET key field value [field value ...] //在一个哈希表key中存储多个键值对

HGET key field //获取哈希表key对应的field键值

HMGET key field [field ...] //批量获取哈希表key中多个field键值

HDEL key field [field ...] //删除哈希表key中的field键值

HLEN key //返回哈希表key中field的数量

HGETALL  key //返回哈希表key中所有的键值

HINCRBY key field increment //为哈希表key中field键的值加上增量increment

对象缓存

HMSET user {userId}:name zhuge {userId}:balance 1888

HMSET user 1:name zhuge 1:balance 1888

HMGET user 1:name 1:balance

Hash结构优缺点

优点

1）同类数据归类整合储存，方便数据管理

2）相比string操作消耗内存与cpu更小

3）相比string储存更节省空间

缺点

过期功能不能使用在field上，只能用在key上

Redis集群架构下不适合大规模使用

keys：全量遍历键，用来列出所有满足特定正则字符串规则的key，当redis数据量比较大时，性能比较差，要避免使用

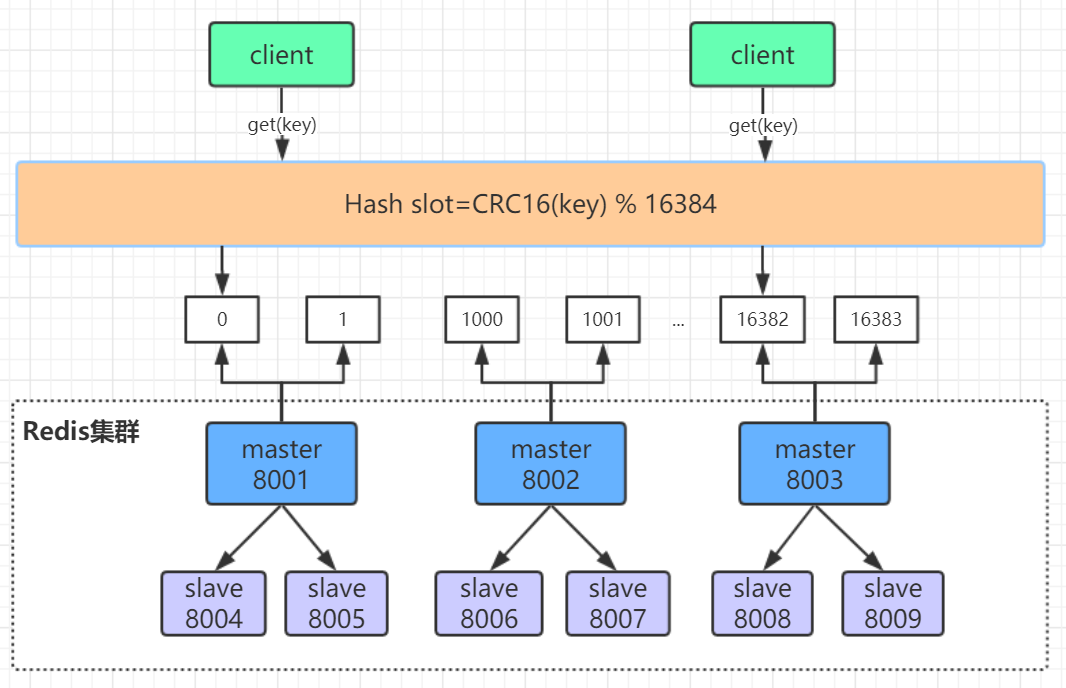
scan：渐进式遍历键

SCAN cursor [MATCH pattern] [COUNT count]

scan 参数提供了三个参数，第一个是 cursor 整数值(hash桶的索引值)，第二个是 key 的正则模式，第三个是一次遍历的key的数量(参考值，底层遍历的数量不一定)，并不是符合条件的结果数量。第一次遍历时，cursor 值为 0，然后将返回结果中第一个整数值作为下一次遍历的 cursor。一直遍历到返回的 cursor 值为 0 时结束。

注意：但是scan并非完美无瑕， 如果在scan的过程中如果有键的变化（增加、 删除、 修改） ，那么遍历效果可能会碰到如下问题： 新增的键可能没有遍历到， 遍历出了重复的键等情况， 也就是说 scan并不能保证完整的遍历出来所有的键， 这些是我们在开发时需要考虑的。

集群架构



List常用操作

LPUSH key value [value ...] //将一个或多个值value插入到key列表的表头(最左边)

RPUSH key value [value ...] //将一个或多个值value插入到key列表的表尾(最右边)

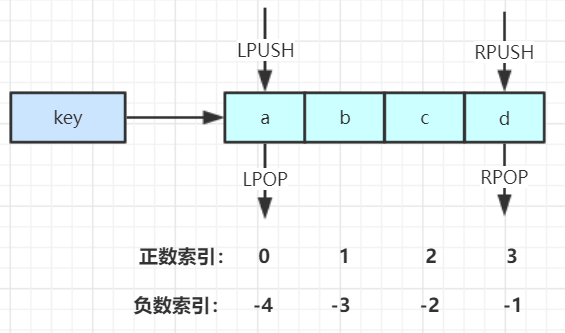
LPOP key //移除并返回key列表的头元素

RPOP key //移除并返回key列表的尾元素

LRANGE key start stop //返回列表key中指定区间内的元素，区间以偏移量start和stop指定

BLPOP key [key ...] timeout //从key列表表头弹出一个元素，若列表中没有元素，阻塞等待timeout秒,如果timeout=0,一直阻塞等待

BRPOP key [key ...] timeout //从key列表表尾弹出一个元素，若列表中没有元素，阻塞等待timeout秒,如果timeout=0,一直阻塞等待



Set常用操作

SADD key member [member ...] //往集合key中存入元素，元素存在则忽略，若key不存在则新建

SREM key member [member ...] //从集合key中删除元素

SMEMBERS key //获取集合key中所有元素

SCARD key //获取集合key的元素个数

SISMEMBER key member //判断member元素是否存在于集合key中

SRANDMEMBER key [count] //从集合key中选出count个元素，元素不从key中删除

SPOP key [count] //从集合key中选出count个元素，元素从key中删除

Set运算操作

SINTER key [key ...] //交集运算

SINTERSTORE destination key [key ..] //将交集结果存入新集合destination中

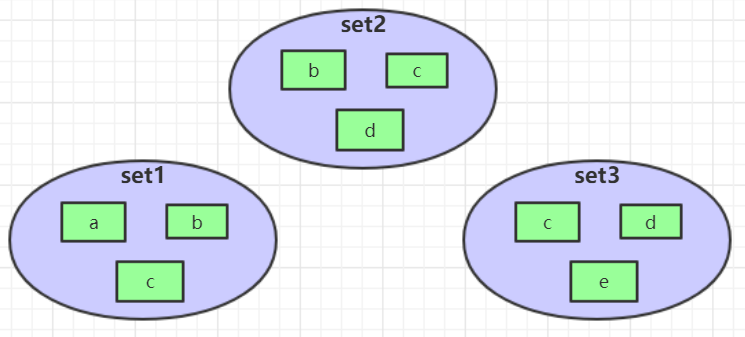
SUNION key [key ..] //并集运算

SUNIONSTORE destination key [key ...] //将并集结果存入新集合destination中

SDIFF key [key ...] //差集运算

SDIFFSTORE destination key [key ...] //将差集结果存入新集合destination中

集合操作



SINTER set1 set2 set3  { c }

SUNION set1 set2 set3  { a,b,c,d,e }

SDIFF set1 set2 set3  { a }

ZSet常用操作

ZADD key score member [[score member]…] //往有序集合key中加入带分值元素

ZREM key member [member …] //从有序集合key中删除元素

ZSCORE key member //返回有序集合key中元素member的分值

ZINCRBY key increment member //为有序集合key中元素member的分值加上increment

ZCARD key //返回有序集合key中元素个数

ZRANGE key start stop [WITHSCORES] //正序获取有序集合key从start下标到stop下标的元素

ZREVRANGE key start stop [WITHSCORES] //倒序获取有序集合key从start下标到stop下标的元素

Zset集合操作

ZUNIONSTORE destkey numkeys key [key ...] //并集计算

ZINTERSTORE destkey numkeys key [key …] //交集计算