redis基础

1. redis中的数据结构

- string：k-v结构，底层实现：整数值、embstr编码的动态字符串、动态字符串。使用场景如下：

1. 缓存功能:最基本的k-v缓存
2. 计数器：用redis快速实现实时计数器
3. 共享数据：用户的session或token等信息

- hash：hash表，底层实现：压缩列表、字典。使用场景如下：

1. 类似map的结构，可以存储对象，但这个对象不能有嵌套的其他对象。

- list：可用来实现消费者生产者队列，底层实现:压缩列表，双向链表

1. 消息队列：有序列表，生产者用lpush插入数据，消费者用brpop阻塞的“抢”消息消费
2. 文章列表或热门消息：可以分页的查询数据，非常方便。

- set：无序不可重复的集合，底层实现：整数集合、字典（hash表）。使用场景如下：

1. 去重：利用set不可重复的机制可以快速去重
2. 取交、并、差集：两个人的好友列表一起取交集查共同好友等

- sorted set：有序不可重复的集合，底层实现：跳表和字典、压缩列表

1. 排行榜：可以根据各个维度的数据进行排序
2. 带权重的队列：不同权重的消息score不同，线程可以按照权重从大到小消费消息。

2. redis底层数据结构

1. 动态字符串：SDS，是一个结构体，成员变量有长度len，剩余空间free，字符串数据buf[]，动态扩容和收缩，减少修改带来的内存重分配次数
2. 双向链表：链表节点都带有prev和next指针，获取某个节点的前置节点和后置节点都是O（N），链表中有记录链表长度的属性len，获取表头和表尾的时间复杂度都是O（1）
3. 字典：就是一个hash表
4. 跳表：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。
5. 整数集合：只能存储整数，数据量不大
6. 压缩列表：是一种为了节约内存而开发的顺序型数据结构。

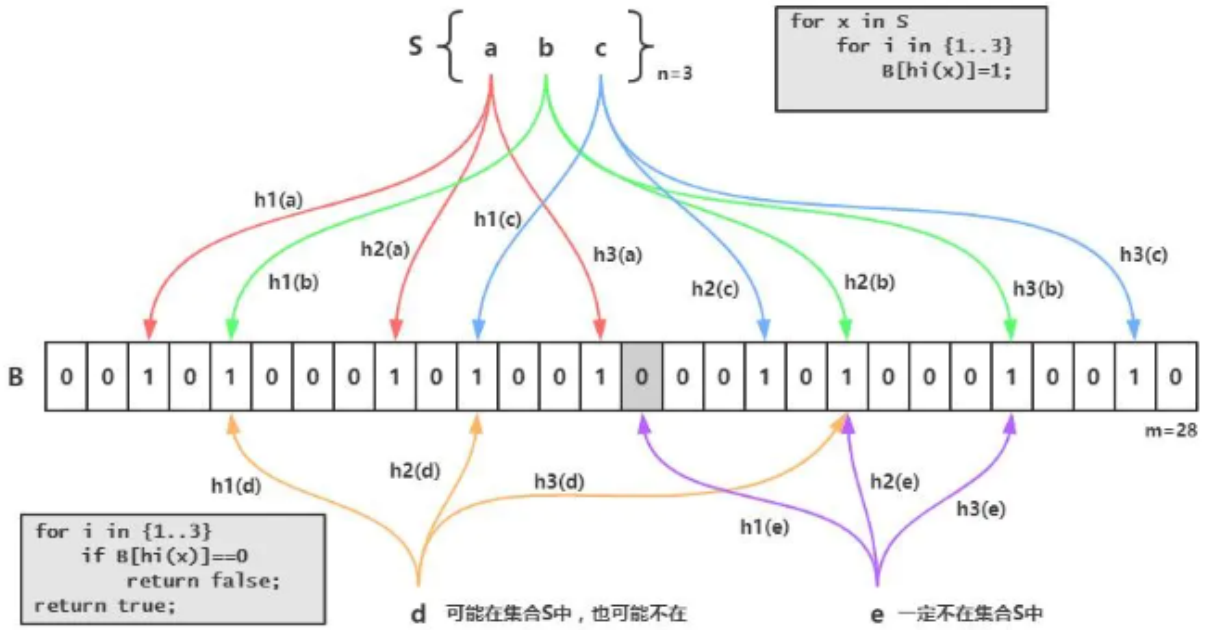
3. 布隆过滤器

3.1 布隆过滤器概念

实际上是一个很长的二进制向量和一系列随机映射函数。布隆过滤器可以用于检索一个元素是否在一个集合中，它的优点是空间效率和查询时间都远远超过一般的算法，缺点是有一定的误识别率和删除困难。

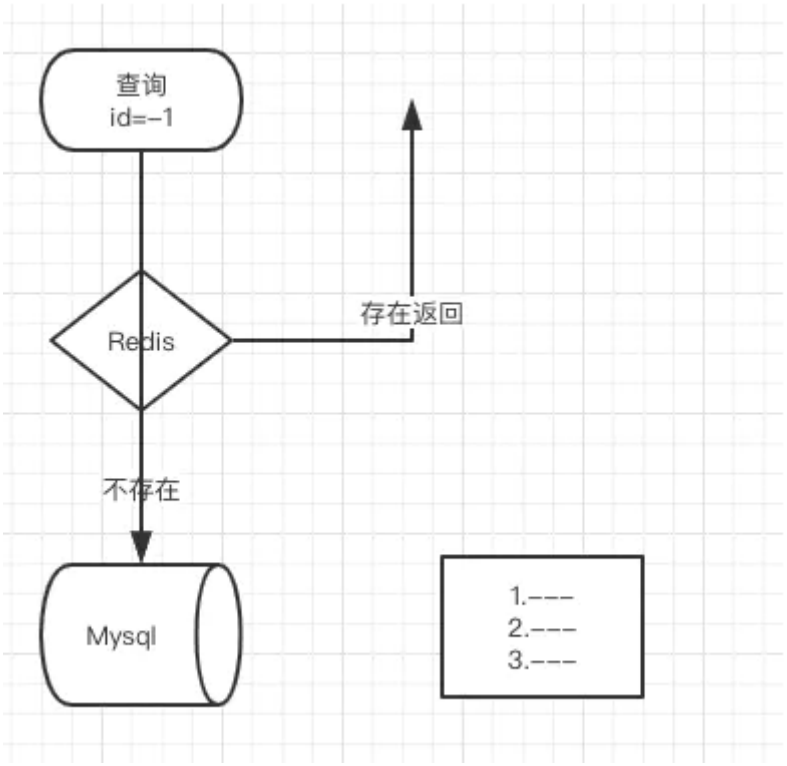
3.2 布隆过滤器原理

布隆过滤器的原理是当一个元素被加入集合时，通过k个散列函数将这个元素映射成一个位数组中的k个点，把它们置为1.检索时，我们只要看看这些点是不是都是1就（大约）知道集合中有没有这个元素，如果有任何一个0，则被检元素一定不存在。如果都是1，则被检索元素很可能在（有误差率）。布隆过滤器跟单哈希函数bit-map不同之处在于，布隆过滤器使用了k个哈希函数，每个字符串跟k个位对应。从而降低了冲突的概率。



3.3 布隆过滤器使用场景

布隆过滤器可以用在缓存击穿上，查询一个redis中不存在的数据，每次请求都会打到mysql，这个就叫缓存击穿。



用布隆过滤器进行恶意请求的过滤，防止恶意请求打垮数据库。

3.4 布隆过滤器的缺点

因为布隆过滤器在取舍中选择了空间和查询效率，就牺牲了准确性和删除的便利性。

- 存在误判，可能要查的元素并不在容器中，但是经过k个哈希函数计算出来是存在的，如果存储的是黑名单，那么可以通过建立一个白名单来存储可能会误判的元素
- 删除困难，因为元素是映射到k个位，删除的时候不能简单的把k个位置为0，会影响其他元素。可以采用counting布隆过滤器。

4. 缓存雪崩

如果大量的key的过期时间设置的过于集中，到过期的时间点可能会造成缓存雪崩，解决方案是在时间上假设一个随机值，是过期时间分散。

5. 假如redis有一亿个key，其中10万个是以固定的已知前缀开头的，如何将它们找出来?

用keys指令，因为redis是单线程的，keys指令会导致线程阻塞一段时间，如果不想阻塞。可以用scan指令，可以无阻塞的取出key列表，但存在重复概率。需要在客户端做去重处理，整体时间耗费会比keys指令长。

6. 使用redis做异步队列，怎么做

使用list作为队列，rpush生产消息，lpop消费消费。当lpop没有消息的时候，要适当sleep一会再重试。如果不想sleep，用指令blpop，没有消息的时候会阻塞直到有消息产生。 如果要实现生产一次

消费多次可以使用pub/sub主题订阅模式，实现1：N的消息队列。但如果消费者下线，生产的消息会丢失。 如果要实现延时队列，可以用sortedSet，用时间戳作为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。

7. redis持久化方案

1. RDB:镜像全量备份，默认5分钟备份一次，如果采用RDB持久化方案会造成5分钟内的数据丢失。

- 优点：会生成多个数据文件，每个数据文件dou都代表了某一时刻redis里面的数据，适合做冷备，可以恢复任一时刻的数据。对redis影响很小，实现是fork一个子进程进行备份，阻塞只发生在fork阶段,数据恢复的速度比AOF快。
- 缺点：数据完整性不如AOF，默认备份时间间隔是5分钟，会丢失一部分时间内的数据。RDB在生成数据快照的时候如果文件很大，客户端会暂停几毫秒甚至几秒。

2. AOF:类似日志记录，每当有写命令进来都通过write函数追加到文件中。如果选择每条命令都同步写入，性能较差，如果选择每隔一段时间内进行一次fsync，那么就会丢失这段时间间隔内的数据。

- 优点：时间间隔短，丢失的数据少。写日志是追加写的方式，少了磁盘寻址的开销，写入性能强，适合用来做灾后数据恢复。
- 缺点：一样的数据，AOF文件比RDB大，AOF开启后对性能有影响，同步写入日志时间间隔要根据业务进行评估。

3. 两种机制都开启的时候，默认用AOF去恢复数据，因为AOF的数据更完整。

8. redis的同步机制

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（RDB全量备份），并同时将后续修改记录到内存buffer，待完成后将RDB文件全量同步到复制节点，复制节点接受完成后将RDB镜像记载到内存，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（AOF日志）进行重放就完成了同步过程。后续的增量数据通过AOF日志同步即可。

9. redis集群的方式

- redis sentinal着眼于高可用，在master宕机时会自动将slave提升为master
- 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

redis缓存雪崩、击穿、穿透

1. 缓存雪崩

缓存雪崩是指大量同时过期的数据失效，导致所有的请求全部去到数据库，数据库被大量的请求打崩，重启数据库又马上崩的一种现象。 对所有的key设置过期时间的时候加上一个随机值，避免所有的key同时过期。热点数据也可以设置永不过期，发生更新时，用后台线程去更新缓存。

2. 缓存穿透

缓存穿透是指请求缓存和数据库中都不存在的数据，用户不断发起请求，导致所有的请求都去到数据库，引起数据库宕机（通常是恶意攻击） 解决方案是在接口层加上校验，校验请求数据的合法性，从缓存取不到的数据，在数据库也取不到的时候设置一个value为null的key，过期时间可以设置的短一点，例如30s或一分钟，在短时间内可以有效的过滤掉大量非法请求。如果是恶意攻击，可以对接口的请求的ip做频率限制，超过一定阀值的都拉黑。另外也可以用布隆过滤器做过滤，如果存在再去db拿数据，刷新缓存再返回。

3. 缓存击穿

缓存击穿和缓存雪崩有点类似，缓存雪崩是因为大面积的key集中失效，打崩数据库。缓存击穿是某个热点rey（例如微博热搜）一直承受高并发的访问，当这个key失效的瞬间，大量的请求打崩数据库。 最简单的解决方案是热点key永不过期，也可以开定时任务定时去更新热点key，在热点key过期前对内容进行更新。

redis哨兵和主从

1. redis哨兵机制

最少需要3个哨兵才能使用完善的哨兵功能，2个哨兵无法执行故障转移等操作。 哨兵功能如下：

- 集群监控：负责监控master和slave进程是否正常工作
- 消息通知：如果某个redis实例有故障，哨兵负责发送消息报警通知管理员
- 故障转移：如果master挂了，会自动转移到slave节点
- 配置中心：如果故障转移发生了，通知client新的master地址。

2. redis主从同步

启动一台slave的时候，它会发送一个psync命令给master，如果是这个slave首次连接到master，会触发一个全量复制。master就会启动一个线程，生成RDB快照，还会把新的写请求都缓存在内存中，RDB文件生成后，master会将这个RDB发给slave，slave拿到之后就写进磁盘，加载到内存，之后master就把内存里缓存的写操作同步发给slave。 如果传输过程中网络中断，重连之后会把缺少的数据补上。

3. redis key的过期策略

redis的key过期策略是定期删除+惰性删除。

- 定期删除：定时随机抽一些key检查，过期了就删除
- 惰性删除：查询这个key的时候判断是否有过期，过期了就删除，返回空。
- LRU:如果没有被定期和惰性删除的key，会使用LRU算法淘汰。（使用频率最少的最先淘汰）
-

4. redis双写一致性

以数据库写为准，优先保证数据库写入成功再去写redis，牺牲短暂时间的redis数据。如果要求强一致性，把读请求和写请求串行化，都放到一个内存队列里，按照顺序执行，数据就不会出现不一致。

5. 数据库更新为什么是删除缓存而不是更新缓存

假如一个数据更新的频率很高，查询频率很低，如果频繁的更新缓存实际上可能并不会被频繁的访问，所以优先是删除缓存，访问到时在重新查询插入缓存。

6.redis和memcached的区别

redis支持更复杂的数据结构，支持更丰富的数据操作。原生支持集群模式，memcached原生不支持集群，需要依靠客户端实现往集群中分片写入数据。redis使用单核，平均每一个核上存储小数据比memcached性能更高，而在100k以上的数据，memcached性能更高。

redis线程模型

1. redis为什么要使用io多路复用

redis是单线程模型，所有的操作都是按照顺序执行，由于读写操作等待用户输入输出都是阻塞的，所以I/O操作一般都不能直接返回，如果I/O阻塞就会导致整个进程无法对其他用户服务。I/O多路复用就是为了解决这个问题。

2. reactor设计模式

redis服务采用reactor的方式来实现文件事件处理器（每一个网络连接对应一个文件描述符）。文件事件处理器使用I/O多路复用模块同时监听多个FD，当accept、read、write和close文件事件产生时，文件事件处理器就会回调FD绑定的事件处理器。

3. I/O多路复用模块

I/O多路复用模块封装了底层的select（标准函数）、epoll（linux系统）、avport等I/O多路复用函数，为上层提供相同的接口。因为select函数的时间复杂度是O（n），其余函数都是O（1），redis会根据编译平台选择不同的函数，优先选择性能高的，select函数是最后的保底选择。