1.什么是Redis

redis是一种内存数据库，处理速度非常快，一般是用来做缓存处理。

此外还支持消息队列，分布式锁，排行榜等功能。

2.Redis为什么快

1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。

2、数据结构高效，Redis 中的数据结构是专门进行设计的。

3、采用单线程，避免了不必要的上下文切换和死锁等消耗。

4、使用多路 I/O 复用模型，非阻塞 IO。

多路IO复用实现

Redis基于Reactor模式开发了一套自己的**文件事件处理器**

文件事件处理器使用 I/O 多路复用程序来同时监听多个Socket，并根据socket目前执行的任务来为socket关联不同的事件处理器。

当被监听的socket准备好执行连接应答(`accept`)、读取(`read`)、写入(`write`)、关闭(`close`)等操作时，就会产生相对应的文件事件，Redis 将所有产生事件的socket都放到一个队列里面，以有序、同步、每次一个socket的方式向文件事件分派器传送socket，文件事件处理器根据socket对应的事件选择相应的处理器进行处理，从而实现了高效的网络请求。

3.Redis基本数据类型

数据类型	可以存储的值	操作	应用场景
STRING	字符串、整数或者浮点数	对整个字符串或者字符串的其中一部分执行操作 对整数和浮点数执行自增或者自减操作	做简单的键值对缓存
LIST	列表	从两端压入或者弹出元素 对单个或者多个元素进行修剪，只保留一个范围内的元素	存储一些列表型的数据结构，类似粉丝列表、文章的评论列表之类的数据
SET	无序集合	添加、获取、移除单个元素 检查一个元素是否存在于集合中 计算交集、并集、差集 从集合里面随机获取元素	交集、并集、差集的操作，比如交集，可以把两个人的粉丝列表整一个交集
HASH	包含键值对的无序散列表	添加、获取、移除单个键值对 获取所有键值对 检查某个键是否存在	结构化的数据，比如一个对象
ZSET	有序集合	添加、获取、删除元素 根据分值范围或者成员来获取元素 计算一个键的排名	去重但可以排序，如获取排名前几名的用户

Bitmap：

位图是支持按 bit 位来存储信息，可以用来实现 **布隆过滤器（BloomFilter）**。

HyperLogLog:

供不精确的去重计数功能，比较适合用来做大规模数据的去重统计，例如统计 UV.

Geo:

可以用来保存地理位置，并作位置距离计算或者根据半径计算位置等。实现附近的人或者计算最优地图路径

pub/sub:

订阅发布功能，可以用作简单的消息队列。

4.String实现原理

底层采用动态字符串实现（**SDS**）

01. len 保存字符串的长度
02. buf[] 数组用来保存字符串的每个元素

03. free 记录了 buf 数组中未使用的字节数量

好处：

01. 获取字符串长度的复杂度为O(1)
02. API是安全的，不会造成缓冲区溢出
03. 减少修改字符串造成的内存重分配
04. 可以保存文本或者二进制数据

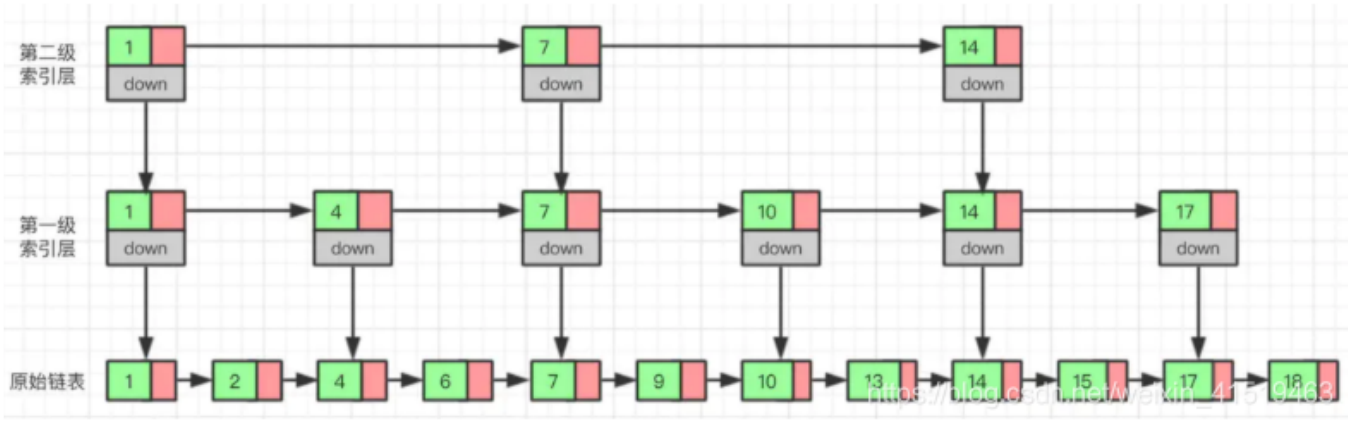
5.ZSet实现原理

01. 数据少时，使用**ziplist**

    ziplist由有序集合实现，每项元素都是（数据+score）的方式连续存储，按照score从小到大排序。ziplist为了节省内存，每个元素占用的空间可以不同，对于大的数据（long long），就多用一些字节来存储，而对于小的数据（short），就少用一些字节来存储。因此查找的时候需要按顺序遍历。ziplist省内存但是查找效率低。

02. 数据多时，使用**跳表**

    跳表是基于一条有序单向链表构造的，通过构建索引提高查找效率，空间换时间，查找方式是从最上面的链表层层往下查找，最后在最底层的链表找到对应的节点



插入：逐层查找位置，然后插入到最底层链表。注意需要维护索引与原始链表的大小平衡，如果底层结点大量增多了，索引也相应增加，避免出现两个索引之间结点过多的情况，查找效率降低。同理，底层结点大量减少时，索引也相应减少。

删除：如果这个结点在索引中也有出现，那么除了要删除原始链表中的结点，还要删除索引中的这个结点。

跳表查找的时间复杂度为O(log(n))。索引占用的空间复杂度为 O(n)。

时间复杂度：时间复杂度 = 索引的层数 * 每层索引遍历元素的个数。

6.Redis使用跳表而不是红黑树的原因

01. 跳表区间查找数据效率更高。
02. 跳表代码易实现、可读性好、不容易出错、更加灵活。
03. 红黑树插入和删除元素开销较大。

7.Redis持久化机制

7.1 AOF日志

工作原理： 记录操作命令

优势： 1.写后日志，redis先执行命令，在记录日志，可以防止错误的命令记录到日志中

2.由于是命令执行后再写日志，不会阻塞当前的写操作

劣势： 1.执行完命令还没写入日志宕机了 会造成数据丢失

2.虽然避免阻塞当前线程的写操作，但是由于要写入磁盘，AOF是主线程操作，可能会阻塞下一个操作

三种写回策略：

配置项	写回时机·	优点	缺点
Always	同步写回	可靠性高数据基本不丢失	每个写命令都要落盘，性能影响大
Everysec	每秒写回	性能适中	宕机时丢失1秒内数据
No	操作系统控制写回	性能好	宕机时丢失的数据比较多

AOF的重写机制

原理： AOF是记录文件形式的操作命令，随着时间的推移文件会越来越大，影响写入效率，这时重新开一个AOF日志，原本的AOF日志可能多条命令对同一键值进行修改，而重新放到新的AOF文件只保存最新状态的命令

重写的具体流程：

01. **一个拷贝：** 主线程fork一个后台bgrewriteadod线程，将主线程的内存拷贝一份给子线程，由这个子线程进行重写
02. **两个日志：** 一个日志主线程正在使用的，另一个是新的AOF重写日志，先将操作写进主线程AOF缓冲区和新AOF缓冲区，等到拷贝数据的所有操作记录在主线程中重写完成后，重写日志记录的这些最新操作也会写入新的AOF文件中.

7.2 RDB快照

- **工作原理**：以快照的形式记录redis某一时刻的数据
- **触发方式**

  save：在主线程下执行，会导致阻塞主线程

  bgsave：fork一个子进程，专门执行写入RDB文件，避免了主线程的阻塞，也是redisRDB的默认配置
- **优势**：数据恢复速度快，无需执行命令
- 缺点：每隔一段时间执行快照，如果在间隔时间内宕机会导致数据丢失，诺让数据间隔短一点会带来更高的性能消耗，诺长一点会导致更多数据的丢失

7.3 Redis4.0的混合持久化

一定频率执行快照，快照之间用AOF记录，这样避免RDB快照间宕机的数据丢失，也避免AOF记录过多的指令

8.Redis删除策略

- **惰性删除**：Redis只在使用key的时候才会检查key是否已经过期，对CPU友好，但是可能存在大量过期的Key未被清理。
- **定期删除**：Redis会定期抽取一批key执行删除过期key操作，并且Redis 底层会通过限制删除操作执行的时长和频率来减少删除操作对 CPU 时间的影响。
- **定时删除**：当Key过期事件到达时，定时器会立即删除Key。

9.Redis淘汰策略

01. volatile-lru（least recently used）：从已设置过期时间的数据集中挑选最近最少使用的数据淘汰
02. volatile-ttl：从已设置过期时间的数据集中挑选将要过期的数据淘汰
03. volatile-random：从已设置过期时间的数据集中任意选择数据淘汰
04. allkeys-lru（least recently used）：当内存不足以容纳新写入数据时，在键空间中，移除最近最少使用的 key
05. allkeys-random：从数据集中任意选择数据淘汰
06. no-eviction：禁止驱逐数据，也就是说当内存不足以容纳新写入数据时，新写入操作会报错。

4.0 版本后增加以下两种：

01. volatile-lfu（least frequently used）：从已设置过期时间的数据集中挑选最不经常使用的数据淘汰
02. allkeys-lfu（least frequently used）：当内存不足以容纳新写入数据时，在键空间中，移除最不经常使用的 key

10.BigKey问题

影响：消耗更多的内存空间，阻塞读取线程。

解决方法：

01. 拆分BigKey，通过multiGet查询，拆分单次操作的压力
02. 压缩BigKey

11.热点Key问题

影响：大量请求打到同一个key上，可能造成缓存击穿。

解决方法：

读方面

01. 人工预估热点Key
02. 采用多级缓存，例如本地HashMap缓存
03. 热点数据永不过期
04. 采用主从同步架构，分担读压力

写方面

01. Redis cluster集群，将key分布到不同节点，分担写压力

12.缓存击穿

描述：

缓存穿透是指大量请求同时打到同一个key上，但是由于他过期了，全部请求达到数据库上，造成宕机。

解决方案：

01. 设置热点数据永远不过期。

02. 加互斥锁，保证由一个线程完成key的重建。

13.缓存穿透

描述:

缓存穿透是指缓存和数据库中都没有的数据，而用户不断发起请求，导致数据库压力过大。

解决方案:

01. 接口层增加参数校验，过滤无效参数。
02. 缓存无效key，定时删除。
03. 使用布隆过滤器，存储合法的key，不存在直接返回空。

14.缓存雪崩

描述:

缓存雪崩是指缓存中大量Key同时过期，引起数据库压力过大甚至down机。

解决方案:

01. 过期时间打散，避免同时失效
02. 集群化处理，将key分布到不同机器上
03. 设置热点数据永远不过期

15.缓存同步一致性

解决方案:

01. 查询的时候首先判断缓存是否存在，不存在则去数据库查找，然后插入缓存。
02. 先修改数据库后删除缓存：可以通过监听数据库binlog的方式与业务解耦优化，可能存在短时间的不同步。
03. 先删除缓存后修改数据库再删除缓存（**延迟双删**）：避免修改数据库期间，有其他线程查询了数据库覆盖了旧缓存，延时时间根据修改时间决定。

16.Redis实现分布式锁

16.1 SETNX

01. 使用Redis的SETNX命令实现，返回0代表加锁成功，返回1则代表加锁失败，并加上过期时间，防止线程一直占有锁未释放。
02. 给value设为当前线程唯一uuid标识，防止其他线程释放锁。
03. 通过 `Lua` 脚本释放锁，保证原子性。

16.2 Redission

01. 通过看门狗机制实现锁自动续期
02. 支持可重入锁

看门狗实现：

01. 当加锁条件没有加上过期时间时，会触发看门狗机制
02. 开启异步线程获取锁
03. 获取到锁后，定时器每10s给锁续期30s，直到释放锁。

可重入实现：

01. 加锁代码首先使用 Redis `exists` 命令判断锁是否存在。
02. 如果锁不存在的话，创建一个 hash 表，并且为 Hash 表中key设为线程唯一uuid标识，value初始化为0，然后加 1，最后再设置过期时间。
03. 如果当前锁存在，则使用 `hexists` 判断当前锁对应的 hash 表中是否存在线程唯一uuid标识这个key，如果存在，则value自增1，最后再次设置过期时间。
04. 解锁操作会将value - 1，value为0则代表锁已释放

16.3 RedLock

01. 防止Redis单实例宕机导致锁失效提出的一种算法

实现：

01. 获取当前系统时间作为开始获取锁的时间
02. 客户端依次向n个redis实例执行加锁操作
03. 加锁操作完成后，客户端使用当前时间减去开始获取锁时间得到加锁总耗时。当且仅当从大多数的Redis节点都获取到锁，并且加锁总耗时小于锁失效时间，才算加锁成功。
04. 如果因为某些原因加锁失败，客户端应该在所有的Redis实例上进行解锁。

17. Redis消息队列实现

17.1 list

01. 使用list的左进右出实现队列先进先出
02. 使用阻塞拉取消息的命令：BRPOP / BLPOP，避免CPU空转问题，没有消息则阻塞等待，发布消息则通知拉取。

缺点:

01. 不支持多个消费者消费：队列消费完数据后，则直接删除。
02. 消息丢失：消费者发生异常宕机后，消息就丢失掉了。

17.2 发布订阅

01. 支持多个消费者绑定一个频道，实现多个消费者消费。
02. 支持阻塞式拉取消息

实现

01. 底层通过字典+链表结构实现，key为频道名，value为所有订阅客户端id
02. 若频道未创建，则创建频道和一个空链表，将订阅者加入到链表中
03. 若频道已创建，则直接获取对应频道的链表，插入到尾部
04. 发布消息时遍历频道对应链表订阅者即可

缺点

01. 消息丢失：消费者未上线，消息堆积，Redis宕机都会产生消息丢失情况。
02. 消息不支持持久化，没有做任何的数据存储，只是单纯转发。

17.3 Stream

01. 支持阻塞等待拉取消息。
02. 支持发布/订阅模式
03. 消费失败，可重新消费，消息不丢失。
04. 实例宕机，消息不丢失，数据可持久化。
05. 消息可堆积

缺点

01. Redis宕机，主从切换可能导致消息丢失
02. 消息积压导致内存压力大。

#	List	Pub/Sub	Stream
阻塞式消费	支持	支持	支持
发布 / 订阅	不支持	支持	支持
重复消费	不支持	不支持	支持
持久化	支持	不支持	支持
消息堆积	内存持续增长	缓冲区溢出，消费者被强制下线	可控制队列最大长度
消息会不会丢失？	Redis 本身不保证数据完整性，有可能存在数据丢失		
消息积压能力	Redis 数据存储在内存，消息堆积对内存压力较大		

18.Redis延迟消息实现

通过zset实现，value存储消息内容，score存储时间戳，按照时间戳进行排序，通过轮询方式取出消息执行。

19.Redis集群搭建

01. 主从复制
02. 哨兵模式
03. Redis-Cluster集群

20.Redis主从复制

20.1 什么是主从复制

将redis实例的数据复制到其他redis实例中，将实例划分为主从节点，可以用来实现数据冗余，故障恢复，读写分离等功能。

数据冗余：主从复制实现了数据的热备份，可以将主节点的数据复制到从节点。

故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复。

读写分离：采用主写从读的架构来实现读写分离，分担redis的读压力。

20.2 主从复制实现方式

01. **全量复制**

    第一次同步时，采用全量复制的方式进行复制。

02. **增量复制**

    把主从库网络断连期间主库收到的命令，同步给从库。

20.3 主从复制流程

01. 从库发送psync命令给主库，psync命令包含复制进度和主库runId等信息。
02. 主库收到psync命令后，会使用FULLRESYNC响应命令带上主库目前的复制进度和主库runId。
03. 主库执行bgsave命令生成RDB文件，放入缓冲区发送给从节点。
04. 从库收到RDB文件，为了保证数据完整度，先清空数据，才进行加载。
05. 如果在加载过程中，主库还收到其他写命令，先放在replication buffer，然后发送给从库，从库再重新执行这些操作。

22.Redis哨兵模式

22.1 什么是哨兵

哨兵是一个运行在特定模式下的Redis实例，主要用来负责监控，选主，通知。

22.2 监控

监控是指哨兵进程运行中会周期性的向主从库发送ping命令，来判断主从库的状态。如果从库在规定时间内没有回应，则会被标记为下线状态。如果主库在规定时间内没有回应，则需要一套重新选主的机制。

22.3 选主

当redis主库挂了后，需要到从库中重新选举出新的主库。

22.3.1 主观下线、客观下线

哨兵会使用ping命令来检测主从库的网络情况，判断实例状态。

主观下线

当主库或从库一段时间没有响应，就会被哨兵进程标记为主观下线状态。

客观下线

当大多数哨兵认为主库主观下线时，会判断主库为客观下线，执行选主流程。

22.3.2 哨兵集群通信机制

哨兵节点之间通过发布订阅模式来进行网络通信。哨兵会往频道中发送ip和端口号信息，绑定到该频道的哨兵节点能够获取到ip和端口号信息，然后与其建立网络连接。

22.3.2 哨兵集群选主机制

判断主库客观下线后，需要主哨兵来进行主从切换

成为主哨兵，要满足两个条件：

- 第一，拿到半数以上的赞成票；
- 第二，拿到的票数同时还需要大于等于哨兵配置文件中的 quorum 值

22.3.3 主库选举机制

- 过滤掉不健康的（下线或断线），没有回复过哨兵ping响应的从节点
- 选择 `salve-priority` 从节点优先级最高（redis.conf）的
- 选择复制偏移量最大，指复制最完整的从节点

22.4 通知

哨兵在选举完新主库后，会将新主库的信息发送给其他从库，让他们执行replicaof命令，和主库建立连接，进行数据复制。

23.Redis cluster

23.1 什么是Redis cluster

就是指启动多个Redis实例组成一个集群，然后按照一定的规则，把收到的数据划分成多份，每一份用一个实例来保存。

23.2 扩展方式

01. **纵向扩展**：升级redis单实例的配置。

    实现简单，但是导致持久化可能阻塞主线程。需要考虑硬件和成本的限制。

02. **横向扩展**：增加redis实例个数。

    需要考虑数据怎么分配到各个实例上，怎么进行访问。

23.3 实现方式

Redis cluster采用哈希槽的方式来处理数据和实例间的映射关系。一个集群有16384个哈希槽，哈希槽类似数据分区，每个键值对都会根据它的key，被映射到一个哈希槽中。

具体过程：

01. 根据Key，按照CRC16算法计算一个16bit的值。然后用这个值对16384取模，每个模数代表对应的哈希槽。
02. 集群有N个实例，每个实例有16384 / N 个哈希槽，可以手动进行分配。实例会保存哈希槽分配信息。

23.4 客户端如何定位数据

01. 客户端和集群实例建立连接后，实例就会把哈希槽的分配信息发给客户端。
02. 客户端收到哈希槽信息后，会把哈希槽信息缓存在本地。当客户端请求键值对时，会先计算Key所对应的哈希槽，然后就可以给相应的实例发送请求了。

23.4.1 重定向

哈希槽和实例的关系并不是一成不变的，最常见的变化有以下两个：

01. redis实例增多或减少，需要重新分配哈希槽。
02. 为了负载均衡，Redis需要把哈希槽在所有实例上重新分布一遍。

Redis cluster提供了**重定向机制**：

move重定向

当客户端把一个键值对的操作请求发给一个实例时，如果这个实例上并没有这个键值对映射的哈希槽，那么这个实例就会给客户端返回MOVED命令，这个命令中就包含了新实例的访问地址，并更改本地缓存。

ask重定向

ASK命令表示两层含义：第一，表明Slot数据还在迁移中；第二，ASK命令把客户端所请求数据的最新实例地址返回给客户端。

ASK命令的作用只是让客户端能给新实例发送一次请求，而不像MOVED命令那样，会更改本地缓存，

23.5 如何判断节点宕机

每一个节点都存有这个集群所有主节点以及从节点的信息。它们之间通过互相的ping-pong判断是否节点可以连接上。如果有一半以上的节点去ping一个节点的时候没有回应，集群就认为这个节点宕机了，然后去连接它的备用节点。

23.6 集群进入fail状态的必要条件

01. 某个主节点和所有从节点全部挂掉，集群就进入faill状态。
02. 如果集群超过半数以上master挂掉，无论是否有slave，集群进入fail状态.
03. 如果集群任意master挂掉,且当前master没有slave.集群进入fail状态

23.7 为什么hash槽是16384个

16384=16k，在发送心跳包时进行bitmap压缩后是2k，也就是说使用2k的空间创建了16k的槽数。