Redis基础

redis优缺点

优点：高性能、支持多种数据结构、支持多种数据持久化方式、支持分布式、支持高可用、简单易用等。

缺点：数据量受内存限制、不支持事务的回滚、数据持久化有数据丢失的风险、集群管理比较复杂等。

Redis 的数据结构及其使用场景

Redis 支持多种数据结构，主要包括：String、List、Set、Hash、Zset。

- String：存储字符串类型数据，常用于缓存和计数器等场景；
- List：存储列表类型数据，可以用于实现队列和栈等数据结构；
- Set：存储无序的唯一值，可以用于实现类似好友关系等场景；
- Hash：存储键值对类型的数据，常用于存储对象和实现计数器等场景；
- Zset：有序集合，可以用于实现排行榜等场景。

QuickList和Ziplist

1. Quicklist：
    - Quicklist 是 Redis 2.4 版本引入的一种列表数据结构的优化方式，主要用于解决列表数据过长导致的内存碎片问题。
    - Quicklist 是一个双向链表的数组，数组中的每个节点都是一个 ziplist，每个 ziplist 存储多个列表元素。
    - Quicklist 可以分成多个 ziplist 节点，每个节点都有固定大小的列表元素。这样，当列表元素较多时，Quicklist 会按需切分为多个 ziplist，避免了单个列表过长造成的内存碎片问题。
    - Quicklist 适用于存储较长的列表，它的设计使得在列表两端的 push 和 pop 操作非常高效。
2. Ziplist：
    - Ziplist（压缩列表）是 Redis 2.2 版本引入的一种紧凑存储列表的数据结构，它主要用于节约内存空间。
    - Ziplist 是一个紧凑的数组结构，可以将多个列表元素按顺序存储在连续的内存空间中。
    - Ziplist 使用一些特殊的编码方式来存储不同类型的数据，例如整数、字符串等，以节约存储空间。这种紧凑的存储方式对于小型列表非常高效。
    - Ziplist 适用于存储较小且元素数量不太多的列表。当列表较大时，Ziplist 可能会导致内存浪费，因为它需要为每个元素预留一定的额外空间。

Zset数据结构

ZSET（有序集合）是一种基于跳跃表（Skip List）实现的数据结构。跳跃表是一种随机化的数据结构，它通过在每个节点中维护多个指向其他节点的指针，从而实现快速的查找、插入和删除操作。

在Redis中，ZSET的每个元素都是一个有序的键值对，其中键是一个字符串，值是一个双精度浮点数。在跳跃表中，每个节点包括一个键、一个值和多个指向其他节点的指针。ZSET的每个元素都会在跳跃表中插入一个节点，并根据值的大小进行排序。

持久化

- RDB：以快照的形式将 Redis 内存中的数据写入磁盘，适用于大规模数据的备份和恢复；
- AOF：将 Redis 内存中的写操作记录到磁盘中，适用于对数据实时性要求较高的场景。（可以保证实时性和高可用）。

RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。

RDB持久化保存键空间的所有键值对(包括过期字典中的数据),并以二进制形式保存，符合rdb文件规范，根据不同数据类型会有不同处理。

AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

AOF持久化保存redis服务器所执行的所有写命令来记录数据库状态,在写入之前命令存储在aof_buf缓冲区。

Redis哨兵模式

Redis哨兵模式是一种高可用性的解决方案，它可以确保Redis集群在主节点宕机时能够自动进行故障转移，从而保证系统的稳定性和可靠性。

在Redis哨兵模式中，有一个或多个哨兵进程运行在独立的进程中，它们负责监控Redis主节点和从节点的状态，并在主节点宕机时自动将从节点切换为新的主节点，从而实现自动故障转移。

哨兵进程通过向Redis节点发送命令来监控节点的状态，例如PING命令用于检查节点是否在线，INFO命令用于获取节点的信息，SENTINEL命令用于获取哨兵的状态信息等。

当哨兵进程检测到主节点宕机时，它会通过一定的算法选出新的主节点，并将这个信息广播给其他哨兵进程和Redis客户端，从而实现自动故障转移。同时，哨兵进程还会监控新的主节点的状态，确保它能够正常工作。

Sentinel（哨兵）可以监听主服务器，并在主服务器进入下线状态时，自动从从服务器中选举出新的主服务器。

集群模式

- 主从复制：通过复制主节点数据到从节点，实现数据的备份和读写分离；
- Redis Cluster：将数据分散到多个节点中，实现数据的分片和高可用。

常见指令

1. `TTL（TIME TO LIVE）`：某个键值对的生存时间
2. `service redis start`: Linux启动redis
3. 设置键值对：`SET key value`，将key和value存储到Redis中，可以用`GET key`获取对应的value。
4. 获取键值对：`GET key`，获取key对应的value。
5. 设置过期时间：`EXPIRE key seconds`，将key设置为在seconds秒之后过期。

6. 判断key是否存在：`EXISTS key`，返回key是否存在的结果。
7. 自增：`INCR key`，将key对应的value自增1，支持浮点数自增`INCRBYFLOAT key increment`。
8. 列表操作：`LPUSH key value`，将value插入到key对应的列表的头部，可以用`LRANGE key start stop`获取指定范围的列表元素。
9. 集合操作：`SADD key member`，将member添加到key对应的集合中，可以用`SMEMBERS key`获取集合中的所有元素。
10. 有序集合操作：`ZADD key score member`，将member添加到key对应的有序集合中，并指定一个分数score，可以用`ZRANGE key start stop`获取指定范围的有序集合元素。
11. 批量操作：`MSET key1 value1 key2 value2 ...`，一次性设置多个键值对，可以用`MGET key1 key2 ...`获取多个key对应的value。
12. 事务操作：`MULTI`，开始一个事务，`EXEC`，提交一个事务，`DISCARD`，取消一个事务。

缓存穿透、缓存雪崩、缓存击穿

缓存穿透：我们使用Redis大部分情况都是通过Key查询对应的值，假如发送的请求传进来的key是不存在Redis中的，那么就查不到缓存，查不到缓存就会去数据库查询。假如有大量这样的请求，这些请求像“穿透”了缓存一样直接打在数据库上，这种现象就叫做缓存穿透。

缓存雪崩：当某一个时刻出现大规模的缓存失效的情况，那么就会导致大量的请求直接打在数据库上面，导致数据库压力巨大，如果在高并发的情况下，可能瞬间就会导致数据库宕机。这时候如果运维马上又重启数据库，马上又会有新的流量把数据库打死。这就是缓存雪崩。

缓存击穿：跟缓存雪崩有点类似，缓存雪崩是大规模的key失效，而缓存击穿是一个热点的Key，有大并发集中对其进行访问，突然间这个Key失效了，导致大并发全部打在数据库上，导致数据库压力剧增。这种现象就叫做缓存击穿。

淘汰策略

策略	描述
volatile-lru	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
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
volatile-random	从已设置过期时间的数据集中任意选择数据淘汰
allkeys-lru	从所有数据集中挑选最近最少使用的数据淘汰
allkeys-random	从所有数据集中任意选择数据进行淘汰
noeviction	禁止驱逐数据

Redis和Memcached

- 数据类型：Memcached仅支持字符串类型，Redis支持五种不同的数据类型
- 持久化：Memcached不支持持久化，Redis支持RDB快照和AOF日志
- 分布式：Memcached不支持分布式，Redis可以通过主存复制和Redis Cluster进行分布式
- 内存管理机制：Memcached一直在内存当中，Redis可以将一些很久没用的value交换到磁盘

Redis Cluster

Redis Cluster 是 Redis 分布式数据库的一种集群解决方案，用于在多个节点上分布数据，并提供高可用性和横向扩展。它允许将数据划分为多个分片，使每个节点只负责部分数据，从而实现负载均衡和数据的快速存储与检索。

1. **分片：** Redis Cluster 将数据划分为多个分片，每个分片由多个节点共同维护。数据分片的实现采用哈希槽（hash slot）的方式，每个分片对应一个或多个哈希槽。
2. **节点间通信：** Redis Cluster 中的节点之间通过内部通信进行数据交换和同步。节点间会维护复制关系，确保数据的备份和可用性。
3. **故障转移：** Redis Cluster 支持自动的故障转移，当某个主节点宕机时，系统会自动选择一个从节点晋升为新的主节点，保证系统的可用性。
4. **高可用性：** 由于 Redis Cluster 的分布式特性和故障转移机制，系统能够提供高可用性，减少因节点故障而导致的数据不可用时间。
5. **水平扩展：** 可以通过增加节点来扩展集群的处理能力，这可以在不中断服务的情况下实现。
6. **配置简化：** Redis Cluster 通过一个配置节点（cluster configuration node）来维护整个集群的配置信息，简化了集群管理的复杂性。

Redis Cluster和Redis Sentinel

区别：

1. **架构：**
    - Redis Cluster 是一种分布式解决方案，用于在多个节点之间分片和分布数据，以提供高可用性和横向扩展。每个节点都可以包含多个数据库分片。
    - Redis Sentinel 是一种监控和管理工具，用于在主从复制架构中监控 Redis 主节点的健康状态并执行自动故障切换。它不处理数据分片和分布。
2. **主要功能：**
    - Redis Cluster 主要关注数据分布、负载均衡和高可用性。它通过数据分片将数据分布在不同的节点上，并自动执行故障转移以保持数据的可用性。
    - Redis Sentinel 主要关注主从复制的高可用性。它监控 Redis 主节点的状态，如果主节点失效，它会自动选举一个从节点作为新的主节点，以确保系统的可用性。

联系：

1. **高可用性：** Redis Cluster 和 Redis Sentinel 都关注提供高可用性的解决方案。Redis Cluster 通过数据分片和自动故障转移来实现高可用性，而 Redis Sentinel 则通过监控主节点并执行故障切换来保证主从复制架构的高可用性。
2. **自动故障转移：** Redis Cluster 和 Redis Sentinel 都支持自动故障转移，但在不同的层面。Redis Cluster 在整个集群中进行数据迁移和故障切换，而 Redis Sentinel 在主从复制中仅处理主节点的切换。
3. **监控：** Redis Sentinel 提供了实时监控 Redis 主从节点的状态，并在发现故障时执行操作。Redis Cluster 也可以通过监控工具进行状态监控，但它更加关注分布式数据的管理。

多级缓存

高级缓存

内存缓存或者Redis缓存

低级缓存

1. **本地缓存：** 本地缓存是指将数据存储在应用程序的本地内存中，不涉及网络通信。它通常用于存储一些临时性的数据，适用于单个应用程序实例的情况。Java 中的 ConcurrentHashMap 可以看作是一种本地缓存。
2. **近程存储缓存：** 近程存储缓存是指将数据存储在应用程序运行的节点附近的存储介质中，例如计算机的硬盘或 SSD。尽管相对于内存来说速度较慢，但仍然比远程网络请求要快，可以有效地提高数据的访问速度。
3. **缓存框架的二级缓存：** 一些持久性框架（如 Hibernate）支持将数据存储在缓存中，以减少对数据库的访问。这种缓存通常使用本地或近程存储，可以显著提高数据库读取性能。

HyperLogLogs

HyperLogLog（HLL）是 Redis 中的一种数据结构，用于实现近似基数（cardinality）估计，即统计一个集合中不重复元素的个数，但占用的内存较少。HLL 可以在极小的内存开销下，估算大规模数据集合的元素个数，适用于需要统计独立元素数量的场景，如页面访问数、用户数量等。

HLL 的原理基于概率统计算法，它使用一些随机化的技巧来实现较小的内存占用。HLL 数据结构存储在 Redis 中的字符串类型值中，因此你可以在 Redis 中使用 HLL 数据类型。

底层原理

- 利用一组哈希函数对输入的元素进行映射，生成哈希值。
- 使用位数组（bit array）来存储哈希值中的位，每个位代表一个桶，可以用来判断某个桶是否被占用。
- 使用位操作来统计位数组中开启的位数，从而估计基数。