**1-Redis**

**Redis数据类型**

**常用数据结构**

* **ziplist（压缩列表）：**由一些特殊编码的连续内存块组成，按照一定规则编码在一块连续的内存区域，目的是节省内存。
* **intset（整数集合）：**数组结构，存数据的时候是有序的，查找数据的时候使用二分法。当数据量超过了当前数组大小，会开辟一个新的数组，然后把旧数据拷贝过去。
* **skiplist（跳表）：**是通过对链表建立索引，从而加快查找的速度（最快 O(logN)，最慢 O(N)）

**string 类型**

**主要作用：**统计计数，共享 session，分布式锁，限流

**具体使用：**

|  |
| --- |
| Bash set key value #设置值 get key #获得值 del key #删除值 SETNX key value #如果 key不存在，则SETNX成功返回1，如果这个key已经存在了，则返回0。 SETEX key seconds value #将值 value 关联到 key ，并将 key 的过期时间设为 seconds (以秒为单位)。 |

**底层数据结构（最大存储512M）：**

* **整数：**字符串长度小于21，且可以转化为整数字符串
* **emstr：**字符串长度小于39，在内存中只可读
* **sds：**以上其他情况来存储，它是一个结构体，里面成员有 free，len，buf，数据存放在 buf 中， buf 是一个字符串数组

**list 类型**

**主要作用：**消息队列，实现分页，博客评论列表

**具体使用：**

|  |
| --- |
| Plain Text lpush key value #左边推入数据 rpush key value #右边推入数据 lpop key #左边弹出数据 rpop key #右边弹出数据 lrange key start end #返回范围内的数据，0 开始，可以 -1 结尾 lindex key index #返回从左到右，第 index 个数据 |

**底层数据结构：**

* **ziplist：**对象保存的所有字符串元素长度小于64，且保存元素数量小于512个
* **linkedlist（双向循环列表）：**其它情况使用

**set 类型**

**主要作用：**用户标签，随机抽奖

**具体使用：**

|  |
| --- |
| Plain Text sadd key value #添加集合值 srem key value #删除集合值 smembers key #返回集合所有数据 sismember key value #检查值是否存在集合中 |

**底层数据结构：**

* **intset：**保存的元素都是整数，且保存的元素数量小于 512
* **hashtable：**其它情况使用

**hash 类型**

**主要作用：**存储用户信息

**具体使用：**

|  |
| --- |
| Plain Text hset key index\_key value #添加散列数据 hdel key index\_key #删除散列数据 hgetall key #返回所有散列数据 hget key index\_key #返回对应散列数据 |

**底层数据结构：**

* **ziplist：**对象保存的所有字符串元素长度小于 64，且保存元素数量小于 512
* **hashtable：**其它情况使用

**zset 类型**

**主要作用：**排行榜，点赞

**具体使用：**

|  |
| --- |
| Plain Text zadd key score value #添加有序集合数据 zrem key value #删除集合数据 zrange key start end withscores #根据一段范围内的数据，根据 score 排序返回 zrangebyscore key score1 score2 withscores #根据 score 区间返回数据 |

**底层数据结构：**

* **ziplist：**对象保存的所有字符串元素长度小于64，且保存元素数量小于 128
* **skiplist：**其它情况使用

**特殊三种类型**

* **Geo：**用于地理定位，位置存储，半径查询，距离查询
* **HyperLog：**基于统计算法的数据结构，用于统计网页的 UV，大数据的去重计算，实时分析，数据流分析
* **Bitmaps：**用一个比特位来映射某个元素的状态，可以用来做用户在线状态，日活用户统计，特征标签。布隆过滤器的基本组成包括一个很长的二进制向量（‌[位数组](https://www.baidu.com/s?wd=%E4%BD%8D%E6%95%B0%E7%BB%84&usm=4&ie=utf-8&rsv_pq=87eed3a000863d03&oq=%E5%B8%83%E9%9A%86%E8%BF%87%E6%BB%A4%E5%99%A8%E7%9A%84%E5%BA%95%E5%B1%82%E7%BB%93%E6%9E%84&rsv_t=223dWQfjMtEDbHBt2cyKMU7ejcudOsYFR5Y1pQhKWzE%2BnjEwnMwNXOLvUpg&sa=re_dqa_generate)）和一系列的随机映射函数（哈希函数）。布隆过滤器（Bloom Filter）是一种空间效率高的数据结构，用于检查一个元素是否可能在一个集合内，或者判断一个元素是否一定不在某个集合内。它可以告诉你 "某东西一定不在那里" 的可能性很高，但是 "某东西一定在那里" 的确定性不高

**常用命令**

**设置过期时间**

EXPIRE key time 单位是秒

**Redis 持久化方式**

**RDB（时间点存储）**

将 redis 某一时刻的数据持久化到磁盘中。持久化过程中会先创建一个临时文件来存储数据，等持久化之后，用该文件替代持久化好的文件。RDB 会 fork 一个子进程来操作。缺点是可能会在持久化间隔时间内丢失数据。

**AOF（追加方式存储）**

将执行的指令记录下来，数据恢复的时候按照指令执行一遍。当 AOF 文件大小超过设定大小的时候，会启动文件重写。文件重写也是 fork 一个子进程进行重写，重写会把现有指令进行压缩，重写的时候会先写临时文件，全部完成后再替换。但是同规模提及下，AOF 文件会比 RDB 体积大，恢复速度也慢于 RDB。

**Redis 高可用**

**主从模式**

**实现原理：**

一般主从同步中，主服务器会关闭数据持久化功能，只让从服务器进行持久化，且从服务器只可读。同步的时候，从服务器会向主服务器发出 SYNC 指令，从服务器收到后会创建一个子进程进行数据持久化。同步期间的写指令会记在缓存中，执行完快照以后，再将缓存的写指令同步给从服务器。如果有多个子服务器发过来 SYNC， 也只会执行一次操作，然后将持久化好的 RDB 发送给从服务器

**缺点：**

当主节点故障，需要人工将从节点晋升为主节点，同时还要通知应用改变了主节点地址。

**哨兵模式**

**实现原理：**

由一个或者多个哨兵组成。哨兵可以监视所有节点，包括其它哨兵节点。如果主节点下线，它会自动提升从节点为主节点。每个哨兵会以一定频率向不同哨兵节点发送 Ping 命令，当某一个节点被大部分哨兵标记下线，则该节点会被下线

**缺点：**

哨兵是基于主从，每个节点存储的数据是一样的，浪费内存，不好在线扩容

**Cluster 集群模式**

**实现原理：**

Redis Cluster 集群通过 Gossip 协议进行通信，节点之间不断交换信息，交换的信息内容包括节点出现故障、新节点加入，主从节点变更信息，slot 信息等等。集群节点缓存了所有集群信息，每个节点都相互连接，常用的 Gossip 消息分为 4 种，分别是：ping、pong、meet、fail

* ping：集群内交换最频繁的消息。集群内每个节点每秒向多个其它节点发送 ping 消息，用于检测节点是否在线，以及交换彼此信息
* pong：当接收到 ping、meet 消息时，作为响应消息回复给发送方，确认消息正常通信
* meet：通知新节点加入
* fail：当节点判断集群内另一个节点下线时，会向集群内广播一个 fail 消息，其它节点收到 fail 消息之后，把对应节点更新为下线状态

**分布式算法：**

使用了哈希槽，没有使用一致性 hash。将整个数据库分为了 16384 个槽（slot）。Redis 的键会根据 key 进行散列，分配到这 16384 中的一个槽位。使用的算法比较简单，用 CRC16 算法计算出一个 16 位的值，在对 16384 取模。集群中的每个节点负责一部分的槽位。为了保证高可用，集群引入了主从复制，cluster 模式下主节点会相互连接，当认为其中某一个主节点挂了后，会启动它的从节点。Redis 只保证了 AP，因为当主节点挂了后，写消息没来得及同步从节点，从节点升级到主节点后，会导致数据不一致。

**故障转移：**

Redis 集群通过 ping/pong 消息实现故障发现

**故障恢复：**

故障发现后，如果下线节点是主节点，则需要在它的从节点中选一个替换它，保证集群的高可用。只有持有槽的主节点才有投票权，当从节点收集到足够的选票（大于一半），会触发替换主节点操作。

**缺点：**

同时操作多个 key 不能使用事务，不能使用一个非常长的排序 key 存储数据集

**Codis 外部集群方式**

**Redis 常用功能**

**Redis 事务**

通过 MULTI，EXEC，DISCARD，WATCH 四个指令来操作事务

* MULTI：组装事务，标记事务开始
* EXEC：执行事务
* DISCARD：取消一个事务
* WATCH：监视一些key，如果这些 key 在事务执行之前被改变，则取消事务

**Redis 虚拟内存机制**

把不经常访问的数据（冷数据）从内存交换到磁盘，通过 VM 功能实现冷热数据分离

**Redis 过期策略**

实际情况会同时使用**惰性过期**和**定期过期**

**定时过期：**设置定时器，到过期时间后会进行清除

**惰性过期：**当访问 key 的时候，才会判断 key是否已经过期，如果过期则清除

**定期过期：**每隔一定时间，扫描一定的数据，并清除已经过期的 key

**Redis 发布/订阅 模型**

发布者：将消息发布到频道，不直接发消息给订阅者

订阅者：订阅一个或者多个频道，只能接收频道消息

支持匹配订阅：支持匹配特定模式的频道，如 news.\*

解耦：发布者和订阅者互不了解，提供系统的可扩展性和维护性

**Redis Streams**

**消息队列：**作为生产者和消费者模式下的消息队列，支持多个生产者和多个消费者

**日志收集：**收集不通过源的日志数据，支持按时间排序存储和查询

**实时消息系统：**支持消息的持久化和历史消息查询

**流处理：**支持对数据流进行复杂的处理和分析，适合实时数据处理的场景

**Redis 管道技术**

支持客户端可以一次性发送多个命令到服务器，无需等待每个命令回包

**Redis 常见问题**

**Redis 模型是什么**

Redis 是单线程模型，因为主要压力点是在网络 I/O 上

**缓存穿透**

**具体场景：**查询一个不存在的数据，导致不会缓存到 redis，每次都去查库，而数据库中也没有。

**解决办法：**

* 在请求入口处进行检查，过滤非法值
* 给缓存设置一个空值或者默认值，如果有写请求需要更新缓存，同时对缓存设置适当的过期时间
* 使用布隆过滤器判断数据是否存在，存在才允许访问

**缓存雪崩**

**具体场景：**当缓存中大量数据过期时间到期，查询数据量很大，导致请求都访问数据库

**解决办法：**

* 均匀设置过期时间，让过期时间相对离散
* redis 宕机也有可能引起该问题，构造高可用 redis 集群

**缓存击穿**

**具体场景：**热点 key 在某个时间刚刚好过期，这个时候对这个 key 有大量的请求发过来，从而大部分请求直接访问数据库

**解决办法：**

* 失效的时候，先使用带返回的原子操作，成功再去加载 db 和设置缓存，否则重试获取缓存
* redis 集群扩容，增加分片副本，均衡流量，将热 key 分散到不同的服务器，使用二级缓存

**如何保证 mysql 与 redis 双写一致性**

* 缓存延迟双删：写请求的时候，先删除缓存，再更新数据库，再删除一次缓存
* 删除缓存重试机制：写请求更新数据库，缓存因为某些原因，删除失败，把删除失败的 key 放到消息队列，等待一段时间后再取出消费队列的消息，获取要删除的 key，重试删除缓存操作
* 读取 binlog，异步删除缓存：将 binlog 日志采集发送到 MQ 队列里，然后通过 ACK 机制确认处理这条更新消息，删除缓存，保证数据缓存一致性

**Redis 什么情况下集群不可用**

A，B，C 三个节点集群，如果 B 挂了，则会导致一部分哈希槽不能使用

**如何保证 Redis 是热点数据**

使用内存淘汰机制

**Redis 集群会有写丢失吗**

Redis 不能保证强一致性，集群在特定情况下可能会导致写丢失，如写的时候主节点还没来得及同步从节点，然后挂了，将从节点升级为主节点，这个时候会导致写丢失

**Redis 缓存如何扩容**

使用一致性哈希实现动态扩容

**Redis 是前期做还是后期规模上来再做**

前期做比较好，可以都先放在一台机器上，后期压力上来了，可以直接多机器部署

**如何查看 Redis 运行状态以及基础配置**

使用 info 命令

**Redis 是单线程，如何提高效率**

同一台服务器部署多个 redis 实例，或者使用分片

**如何提高 Redis 性能**

* Master 不要做持久化工作：如 RDB 内存快照， AOF 日志文件
* Slave 开启 AOF 备份
* Master 和 Slave 在一个局域网内
* 主从可以使用单向链表来实现
* 避免在压力很大的主库上增加从库

**修改 Redis 配置可以不重启 Redis 生效吗？**

可以

**如何处理大量连接 Redis 的场景**

* 使用连接池
* 调整系统参数：如文件描述符
* 优化 Redis 配置： tcp-back-log 或者 timeout 设置

**如果同时使用 RDB 和 AOF ，Redis 会如何处理**

会优先加载 AOF 文件，因为 AOF 文件通常保存了更完整的操作

**Redis 如何保证性能和健康**

使用 info 查看 Redis 各种信息，使用 Redis 监控工具，日志分析，定期检查

**Redis 如何处理内存碎片**

监控内存碎片率，Redis 使用了 jemalloc，重启 Redis 服务

**如何优化 Redis 中 list 内存使用**

减小元素大小，避免 list 长度过小，list 中数据压缩后在存入

**Redis 如何处理读写冲突**

一般是单线程模式，不存在读写冲突。如果是事务可以通过操作实现原子操作，或者使用 发布/订阅 模式

**Redis 惰性如何工作**

键过期检查，节省资源，避免性能抖动

**Redis 如何保证主从复制的数据一致性**

先使用全量复制，然后使用增量复制（已命令方式将复制期间的变化传到从库）

**zset优化**

合理设置键值，使用范围查询，分片存储，定期清理

**Redis 集群处理什么命令会导致跨槽**

使用哈希标签，客户端支持

**Redis 如何实现延迟队列**

使用 zset 实现，每个元素带一个有时间戳的 score 表示其应该被处理的事件，使用 zadd 添加，zrangeByScore 来检查小于时间戳的数据，处理完后使用 zrem 移除数据

**Redis 的并发竞争是什么**

多个客户端同时写一个 key。解决办法：通过分布式锁，同一时间只有一个系统实例在操作一个 key，别人都不许读和写，每次写入缓存的数据都是从 mysql 查出来的，写入 mysql 的时候保存一个时间戳，如果系的时候时间戳比数据库中的早，则不写入。

**Redis 常见的机器配置**

部署 redis cluster 一共使用十台机器，五台机器部署了 redis 主实例，另外五台机器部署了 redis 的从实例，每个主实例下部署了 redis 的从实例。每个节点的读写高峰 qps 可以达到每秒 5 万。机器配置：32G内存 + 8 核 CPU + 1T硬盘，内存配置尽量不要超过 10G。200万条商品记录，每条大概 10 kb，占内存大概 20G。高峰期 qps 大概 3500左右。

**如何在 10 亿个 key 中找出某些前缀开头的数据**

使用 keys 命令扫出指定模式的 key 列表。keys 命令可能会导致线程阻塞，因为 redis 是单线程。可以使用 scan 命令，scan 指令可以无阻塞的提取出指定模式的 key 列表，但是会有一定的重复概率，在客户端再做一次去重即可，总体时间会比 keys 要长

**Redis 分布式锁实现**

**分布式锁特点**

* 「互斥性」: 任意时刻，只有一个客户端能持有锁。
* 「锁超时释放」：持有锁超时，可以释放，防止不必要的资源浪费，也可以防止死锁。
* 「可重入性」:一个线程如果获取了锁之后,可以再次对其请求加锁。
* 「高性能和高可用」：加锁和解锁需要开销尽可能低，同时也要保证高可用，避免分布式锁失效。
* 「安全性」：锁只能被持有的客户端删除，不能被其他客户端删除

**方案一：SETNX + EXPIRE**

setnx 和 expire 两个命令分开了，「不是原子操作」。如果执行完setnx加锁，正要执行expire设置过期时间时，进程crash或者要重启维护了，那么这个锁就“长生不老”了，「别的线程永远获取不到锁啦」。

**方案二：SETNX + value值是(系统时间+过期时间)**

把「过期时间放到setnx的value值」里面来。解决了方案一发生异常，锁得不到释放的问题。但是这个方案还有别的缺点：

* 过期时间是客户端自己生成的（System.currentTimeMillis()是当前系统的时间），必须要求分布式环境下，每个客户端的时间必须同步。
* 如果锁过期的时候，并发多个客户端同时请求过来，都执行jedis.getSet()，最终只能有一个客户端加锁成功，但是该客户端锁的过期时间，可能被别的客户端覆盖
* 该锁没有保存持有者的唯一标识，可能被别的客户端释放/解锁。

**方案三：使用Lua脚本(包含SETNX + EXPIRE两条指令)**

Lua 脚本中运行 SETNX 和 EXPIRE，但是还是会有方案二的问题

**方案四：SET的扩展命令（SET EX PX NX）**。

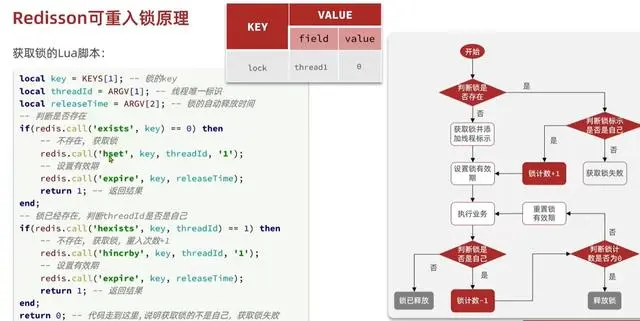
使用SET命令的扩展参数来实现分布式锁，EX表示过期时间，PX表示毫秒单位，NX表示只有当键不存在时才设置值。这种方案的优点是简单高效，可以一条命令完成加锁和设置过期时间123。

**方案五：SET EX PX NX + 校验唯一随机值,再释放锁**

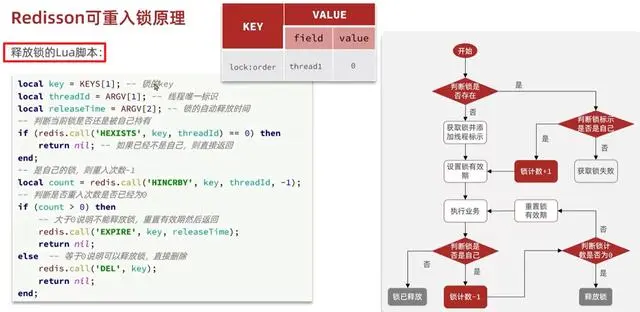
在方案四的基础上，增加了一个唯一随机值作为锁的持有者标识，只有持有者才能释放锁。这种方案的优点是增加了安全性，避免了其他客户端误删或者覆盖锁。

**方案六：Redisson**

使用Lua脚本获取锁如下：



使用Lua脚本释放锁的脚本：



**BIG KEY**

**如何识别Big Key？**

使用redis自带的命令识别。

例如可以使用Redis官方客户端redis-cli加上--bigkeys参数，可以找到某个实例5种数据类型(String、hash、list、set、zset)的最大key。

优点是可以在线扫描，不阻塞服务；缺点是信息较少，内容不够精确。

**Big Key的危害？**

1. 阻塞请求
2. 内存增大
3. 阻塞网络
4. 影响主从同步、主从切换

**如何解决Big Key问题？**

1. 对大Key进行拆分
2. 对大Key进行清理
3. 监控Redis的内存、网络带宽、超时等指标
4. 定期清理失效数据
5. 压缩value

**在Redis中，大批量删除hash数据问题**

在Redis中，大批量删除hash数据可以使用HDEL命令，但是由于Redis命令的原子性，如果要删除的field数量非常多，应避免一次性发送大量的命令，以免造成网络瓶颈或阻塞。