Redis

常用命令

Redis setnx

Redis在项目中是使用

缓存：提高查询速度，降低数据库压力Springboot中Redis做缓存

你用过redis中的哪些数据类型

String：计数场景

List：双向链表，消息队列，订阅模式

hash：键值对

set：存放不重复数据以及获得交集并集

sorted set：与set相比多了score字段，可以根据score字段进行排序

redis的场景数据类型和应用场景

zset 的实现

查询和插入的时间复杂度

持久化

RDB与AOF的对比

RDB：将某个时间点的所有数据都存放到硬盘上。会丢失一段时间的数据，Redis默认的持久化方案

AOF：将写命令添加到 AOF 文件，如果系统发生故障，将会丢失最后一次创建快照之后的数据。

Redis默认没有开启AOF，`appendonly yes`来开启。AOF可以选择每次数据改变/每秒/系统自己决定什么时候进行持久化。

Redis 4.0 开始支持 RDB 和 AOF 的混合持久化。如果把混合持久化打开，AOF 重写的时候就直接把 RDB 的内容写到 AOF 文件开头。这样做的 好处是可以结合 RDB 和 AOF 的优点, 快速加载同时避免丢失过多的数据。

如何利用Redis锁解决高并发问题

Redis集群如何同步

主从复制

使用流程：

工作流程：

1、从服务发送一个 sync 同步命令给主服务要求全量同步
2、主服务接收到从服务的 sync 同步命令时，会 fork 一个子进程后台执行 bgsave 命令（非阻塞）快照保存，生成 RDB 文件，并将
RDB 文件发送给从服务
3、从服务再将接收到的 RDB 文件载入自己的 redis 内存
4、待从服务将 RDB 载入完成后，主服务再将缓冲区所有写命令发送给从服务
5、从服务在将主服务所有的写命令载入内存从而实现数据的完整同步
6、从服务下次在需要同步数据时只需要发送自己的 offset 位置（相当于 mysql binlog 的位置） 即可，只同步新增加的数据，再不需要全量同步

哨兵模式

redis 的哨兵机制的作用?

1、监控:Sentinel 会不断的检查主服务器和从服务器是否正常运行。
2、通知:当被监控的某个 redis 服务器出现问题，Sentinel 通过 API 脚本向管理员或者其他的应用程序发送通知。
3、自动故障转移:当主节点不能正常工作时，Sentinel 会开始一次自动的故障转移操作，它会将与失效主节点是主从关系 的其中一个从节点升级为新的主节点，并且将其他的从节点指向新的主节点。

哨兵模式选举过程

1. 故障节点主观下线
2. 故障节点客观下线
3. Sentinel集群选举Leader
4. Sentinel Leader决定新主节点

1. 新建配置文件：sentinel.conf

```
sentinel monitor mastername host port number
```

- mastername,给master起个名字
- number，至少有number个从服务器的哨兵同意

2. `redis-sentinel 配置文件` 来启动哨兵

Master-Slave切换后，master_redis.conf、slave_redis.conf和sentinel.conf的内容都会自动发生改变，即master_redis.conf中会多一行slaveof的配置，sentinel.conf的监控目标会随之调换。

设置优先级

配置文件中`replice-priority`值越小，优先级越高

选举规则

1. 优先级高的(sentinel配置文件中，slave-priority 100,值越小，优先级越高)
2. 偏移量小的(跟主服务器最相近的)
3. runid(启动后自动生成)小的

RedisCluster

怎么进行数据插入和查询的

Redis分布式锁的实现
Redis的扩容机制

热点问题

Redis和Memcache 的区别

- 数据类型
- 持久化

内存淘汰策略

已过期/未过期，最近最少/随机，将过期/不删除,默认不再提供服务，直接报错。

1. volatile-lru(least recently used):从已设置过期时间的数据集(server.db[i].expires) 中挑选最近最少使用的数据淘汰
2. volatile-ttl:从已设置过期时间的数据集(server.db[i].expires)中挑选将要过期的数据淘汰
3. volatile-random:从已设置过期时间的数据集(server.db[i].expires)中任意选择数据淘汰
4. 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(这个是最常用的)
5. allkeys-random:从数据集(server.db[i].dict)中任意选择数据淘汰
6. 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。这个应该没人使用吧!

Redis是如何判断数据是否过期的呢

Redis 通过一个叫做过期字典（可以看作是hash表）来保存数据过期的时间。过期字典的键指向。Redis数据库中的某个key(键)，过期字典的值是一个long long类型的整数，这个整数保存了key所指向的数据库键的过期时间（毫秒精度的UNIX时间戳）

过期的数据的删除策略

- 惰性删除： 只会在取出key的时候才对数据进行过期检查。这样对CPU最友好，但是可能会 造成太多过期 key 没有被删除。
- 定期删除： 每隔一段时间抽取一批 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。

缓存穿透

访问不存在的key，每次都穿过缓存去查数据库

解决方案：

首先需要参数校验

1. 缓存无效的key：把无效的 ID，也在 redis 缓存起来，并设置一个很短的超时时间
2. 布隆过滤器：布隆过滤器说某个元素存在，某个元素可能小概率不存在，说某个元素不存咋，某个元素就一定不存在。

缓存击穿

解决方案：

1. 热点数据设置永不超时
2. 对访问的 Key 加上互斥锁，请求的 Key 如果不存在，则加锁，去数据库取，新请求过来，如果相同 KEy,则暂停 10s 再去缓存取值；如果 Key 不同，则直接去缓存取！

缓存雪崩

redis为什么单线程还这么快

1. 单线程编程容易并且更容易维护;
2. Redis 的性能瓶颈不再 CPU ， 主要在内存和网络;
3. 多线程就会存在死锁、线程上下文切换等问题，甚至会影响性能

Redis为什么又加入了多线程

为了提高网络IO读写能力

虽然，Redis6.0 引入了多线程，但是 Redis 的多线程只是在网络数据的读写这类耗时操作上使用了， 执行命令仍然是单线程顺序执行

[为什么 Redis 选择单线程模型]

Redis6新特性

redis缓存和数据库的数据不一致怎么解决

出现不一致的原因：两个请求，一个写一个读，写没刷入数据库时，一个请求进行了读取，刷新到了Redis

解决方案：***缓存双淘汰法***。先淘汰缓存

怎么保证数据一致性

一分钟只能发送一次验证码

一分钟只能发送3次验证码

redis的并发竞争问题如何解决