# REDIS

## 概念

redis是一个基于key-value格式存储noSql数据库（不仅仅是sql）,可以作为一个高速缓存缓解数据库和web服务器的压力。

数据存储使用类似java hashmap的字典结构，hash冲突时会以链表存储。

Redis操作时为单线程多路复用（省去了频繁创建销毁线程的开销），所以高并发下同时提交给redis服务器的命令为串行的。

## 可支持数据类型

1. String ：

数据结构：

Value底层为一个预分配内存一个动态字符串数组，超出容量后会扩容一倍,，类似与java的ArrayList

操作命令：

set get mset<v1>...<v10>（一次存入多值）

1. List: 底层为一个双向链表，

数据结构：

Value的底层为快速链表，一段一段的list ,list之间进行链表连接。数据量小时是一个list，数据大时变为多个list，进行链表连接。头尾操作效率高，中间节点操作效率低。优点是：比普通链表查找快，又比普通list插入快

操作命令：

lpush <key> <value> 从左边放值，后放的值在之前放入值左边

rpush <key> <value> 从右边放， 新值在旧值右边

Lpop <key> 从左边取

lpop <key> 从右边取

Lrange <start> <end> 查看数据下标范围的元素

Rpoplpush <key1> <key2> 从key1右边取出放到key2左边

Lset <index><value> 替换inde位置的值

Linsert <key> before <value> <newValue> 插入值

Lrem <key> <count> <value> 从最左边开始删除count个值为value的元素

Lindex <key> <index> 从左边获取第index个元素

1. Set

数据结构：Value为null的hash表

常用命令：可以求 交 、并、差集

1. Hash

数据结构：value为hash结构，类似于java中的Map<String,Object>。value底层为ziplist（压缩列表）或者hashtable。value的field少且值短则用ziplist否则用hashtable。

常用命令：存对象、取对象key、取对象值、所有key、所有值、给值做加法

1. Zset（存值时需要给分数）

数据结构：（key为值，value为分数）的跳跃表（用于快速查找元素）

常用命令：按照分数排名，按照分数范围取值，查询某值排名

容器启动命令：docker run -d --name redis -p 6379:6379 -v /home/docker/redis/:/etc/redis/ --restart=always --privileged=true redis:latest redis-server /etc/redis/redis.conf

## Lua脚本

Redis内置lua脚本解析器支持使用lua脚本操作redis,redis单线程的，保证lua脚本以类似于redis事务的方式执行，不会被别的客户端发送的命令插队。但是无法直接保证全部成功，全部失败。想要全部成功、全部失败需要自己再lua脚本中逻辑判断实现

## 发布订阅模式

订阅频道：subscribe <channel> 订阅频道

发布消息：publish <channel> <message> 发布消息

## 事务

定义：防止别的客户端提交的命令插队，命令放到队列中组队后进行序列化，然后再一次执行

事务特点：

1. 不保证原子性（事务被提交后某个命令失败，不影响其他命令）
2. 没有隔离级别的概念：事务提交之前命令不会真的提交给数据库（关系型数据库中事务没提交时命令会提交给数据库，所以有隔离级别的概念）
3. 命令不会被其他客户端提交的命令打断（命令序列化后一次提交给服务器，然后一次执行完）

事务补充：事务虽然本身没有原子性，但是当事务和乐观锁一起使用时，如果是因为数据被别的客户端更改导致事务命令失败，则事务所有命令失败具有原子性。如果是因为事务操作本身命令操作错误，则事务不具有原子性。

使用：开启事务multi 提交事务 exec 撤销事务discard

事务失败：1.组队时失败，则全部命令失败 （比如set时不给值）

1. 执行时失败，则只有执行失败的命令失败（比如给string类型做加减法）

事务的冲突问题：

多个客户端同时操作同一数据导致数据状态不正确的问题

事务冲突解决方法：

对数据访问加锁，锁分为悲观锁和乐观锁

悲观锁：认为自己操作数据时，别人一定会操作，所以自己操作时，对数据上锁，然后不让别人操作。别人只能等自己释放完锁 。Redis默认不让使用者使用。

乐观锁：通过版本号进行控制，获取数据时对数据设置一个版本号，对数据操作完毕后如果版本号没修改则更新数据，如果已经被别人修改，此时为防止数据出现不正确状态，所以此次操作无效。并且只有和事务一起使用时才生效

乐观锁使用：watch <key1> <key2> ：开启key的监视，unwatch 取消所有key的监视

## 数据持久化

Redis提供两种持久化技术（可以在配置文件中配置使用的持久化策略，默认只开启rdb）

RDB:备份周期时间达到条件则进行持久化，持久化时先创建一个子进程，然后把数据在内存中复制一份，然后子进程将数据持久化到一个dump.rdb文件中。(xx秒时间内,key变化xx个，则进行一次持久化,可以再redis.conf文件中配置多个策略)

RDB方式的问题：

1. 过度消耗内存（持久化时会复制一份数据到内存中）
2. 最后一次持久化后的数据可能会丢失（因为时间隔固定时间后检查数据变化再持久化，所以持久化后当有数据变化，而服务器挂掉，修改的数据就会丢失）

AOF：将所有写操作记录到一个 appendonly.aof 文件中（当文件太大时会进行文件重写，压缩文件命令，多条合并为一条），redis重启后根据此文件重新加载数据到内存中。（默认不开启需要在redis.conf文件中修改配置开启）

AOF同步频率：

appendfsync：always 每次写操作马上记录

appendfsync：everysec 每一秒记录一次

appendfsync：no redis不会主动同步，何时同步交给操作系统

Rdb和aof对比

优点：Aof数据丢失概率低（可以实时同步数据）；文件可读，可以修改误操作的数据。

缺点：比rdb更消耗存储（不光记录数据而且记录命令）；效率低（每次写操作要进行同步）；数据恢复慢（要重新执行操作命令恢复数据）

## 主从复制

master/slaver机制，实现读写分离（master写操作，slaver读操作。），灾难恢复。

命令：info replication 查看主从信息 ;slaveof <ip> <port> 设置当前节点的主节点

特点：

1. 从服务挂掉重启需要重新配置为丛节点，重新配置后会复制完整数据到从节点）（通过持久化的数据文件）。
2. 主服务挂掉，从服务还是从服务，主服务重启后还是主服务器。（需要自动切换主服务器则需要配置哨兵监视主机状态）

哨兵模式：

启动一个特殊的不提供服务的redis服务，监视主机状态，主机挂掉选从节点升级为主节点

新节点的选择根据1.配置文件中的优先级 2.和主节点数据的同步程度 3.服务启动时生成的runid。

问题：

复制延迟、无法解决数据量存储上限的问题所以需要redis集群

## redis集群

主要作用是解决容量不足问题和高并发的写操作

集群搭建：

1.早期：通过代理服务器解决服务入口问题 redis3.0退出了无中心化配置模式（redis集群节2.点可以互相连通，任何一个节点都可以作为服务入口）。

集群有多个主节点，每个主节点也可以有多个从节点。时

集群特点：

1. 每个主从节点保存1/N个数据（redis集群把数据分为16384个数据槽，根据hash取模来进行数据存储的分配）
2. 集群可以实现故障恢复
3. 集群支持水平扩展增加主从节点，并且数据存储会重新分配
4. 无中心化，每个节点都可以作为客户端的连接点，会根据数据槽的hash计算并分配给对应数据槽的节点
5. 主机挂掉后再恢复后会变为从机
6. 集群某个节点整个挂掉后会根据配置要么redis服务失效 要么当前节点数据槽不能服务其他数据槽还可以继续服务

## 经典问题

缓存穿透：

问题描述：web请求急剧增多，并且访问的数据都在数据库中无法查到，因而再redis中无缓存。导致每次请求都去访问数据库最终导致服务故障。

问题解决：：

1. redis中缓存空对象，返回空对象给客户端，以减少数据库服务的压力。

方案弊端：大量穿透key时会导致内存占用过多的问题。

1. 布隆过滤器

缓存击穿：

热度极高的key当失效的瞬间，大量的请求去访问数据库，导致大量请求阻塞

问题解决：

1. 代码中使用互斥锁，同一时间只有一个线程访问数库，其他线程阻塞，然后将数据写入缓存，其他线程读取缓存数据。

方案弊端：降低应用的吞吐量

1. 热点数据永不过期：

发现热点数据要过期，则启动一个线程去访问数据库然后重新写入缓存

方案弊端：除异步线程外其他线程在数据更新的时间段内只能访问到老值。

缓存雪崩：

问题描述：热点数据同时到达过期时间失效，导致大量请求访问数据库，致使数据库压力过大，应用故障

问题解决：

1.均匀过期：增加随机失效值，或者统一规划过期时间

2.加互斥锁

3.缓存永不过期

4.双层缓存策略

缓存预热：

常用的缓存数据再系统启动时直接加载到内存中

## Redis实现分布式锁

分布式互斥资源访问问题

1. 集群模式下共享资源访问
2. 分布式架构系统共享资源的访问

分布式锁要达到的目标：

1. 互斥性：同一时间只能有一个线程获得锁
2. 不能发生死锁
3. 自己上的锁只能自己解锁
4. 业务服务正常运行时，不能因为业务卡顿耗时，锁时间到期而自动解锁

实现思路：

使用redis的setnx 操作，互斥的上锁。

实现分布式锁会出现的问题

1. 上锁后宕机导致死锁：可以添加超时时间解决（但是上锁和添加超时时间要保证原子性，所以使用lua脚本实现）
2. 超时后锁自动被释放，别的线程已经获得了锁，然后业务部分代码执行完毕会释放锁，此时释放的是其他线程的锁：使用锁中存放uuid进行标识，释放锁时对uuid进行判断
3. 超时释放锁后导致其他线程获得了锁，多个线程操作了共享资源，导致数据安全问题：

添加锁时启动一个守护线程，每隔一段时间查看任务线程状态，并重置锁过期时间。任务线程执行完毕后给守护线程信号，守护线程退出。