Redis

单线程 + 多路 IO 复用

---vs. memcached---
1. 单一 String 类型
2. 不支持持久化
3. 多线程 + 锁

- **常用操作**
  1.
  Redis 是单线程的，因此单命令都是“原子操作”
  单命令执行多个 key 操作时，由于操作原子性，有一个 key 导致失败，则整个命令全部失败
  - shutdown

    关闭 Redis

    1. redis-cli shutdown（shell）
    2. shutdown (先进入终端)
  - config

    config set 设置配置值
    config get 获取配置值
  - auth pwd

    登录
  - key 操作
    - keys *

      查看当前库所有 key，* 可以做通配符匹配
    - exists key

      判断 key 的存在
    - type key

      查看 key 的类型
    - del key

      删除指定 key
    - unlink key

      非阻塞删除
      将 key 从 keyspace 元数据中删除，真正的删除在后续异步操作完成
    - expire key 10

      设置 10 秒后过期
    - ttl key

      查看过期秒数
      -1 - 永不过期
      -2 - 已过期
    - select 1

      切换到数据库 1
    - dbsize

      查看当前数据库 key 的数量
    - flushdb

      清空当前库
    - flushall

      清空所有库
  - 数据类型
    - String

      key - value
      二进制安全的，可以存储任意序列化对象
      value 最大 512MB

      ---场景---
      封锁一个 IP， incrby 命令计数
    - List

      单键多值
      双向链表，对两端的操作性能很高，通过索引下标的操作性能较差

      ---场景---
      模拟消息队列
    - Set

string 类型的无序集合
一个 Value 为 null 的 hash 表，增删查复杂度都是 O(1)

---场景---
自动排重，存放每个人的好友集合

- Hash

  String 类型的 field - value 映射表
  特别适合存储对象，类似于 Java 的 Map<String, Object>

  ---场景---
  存储用户信息（id，name，age）

- ZSet

  为每个成员关联一个评分
  集合成员是唯一的，评分可以是重复的

  报活，按权重排序等

- 新数据类型
  - Bitmaps

    key-value，value 是 bit 型数组
    数组下标即 bitmaps 的偏移量

  - HyperLogLog

    基数问题
    求集合中不重复元素个数

  - Geospatial

    Geographic，地理信息

- **配置** 2
  - units

    配置大小单位

  - includes

    被应用到 redis 实例的 redis.conf 配置
    后面覆盖前面

  - bind

    127.0.0.1 只本机访问
    不写 任意IP，protected-mode 开启，且为设置密码时，只本机访问

  - protected-mode
  - port

    默认 6378

  - tcp-backlog

    连接队列=未完成三次握手队列+已完成三次握手队列

    linux 内核 ->
    /proc/sys/net/core/somaxconn (128)
    /proc/sys/net/ipv4/tcp_max_syn_backlog (128)

  - timeout

    空闲客户端连接的超时时间，0表示永不关闭

  - tcp-keepalive

    对访问客户端的 心跳检测，0 表示不检测

  - daemanize

    守护进程

  - pidfile

    存放pid的文件位置

  - loglevel

    四个级别：debug/verbose/notice(default)/warning

  - logfile
  - database

    库的数量

  - requirepass

    密码

  - maxclients

    最大可连客户端数量，默认 10000，超出抛异常

  - maxmemory

    最大可用内存，不舍可能造成服务器宕机
    超过则可以通过 maxmemory-policy 移除内部数据，对读指令正常返回
    在集群模式下，要给系统预留一些用于同步队列缓存的内存空间

- maxmemory-policy

  ---内存淘汰策略---

  定时删除：对 CPU 不友好，用处理器性能换取存储空间（拿时间换空间）
  惰性删除：对内存不友好，用存储空间换取处理器性能（拿空间换时间）
  定期删除：每隔一段时间执行一次删除过期键操作，并通过限制删除操作执行的时长和频率来减少操作对CPU 时间的影响

  LRU Least Recently Used
  LFU least Frequently Used

  - noeviction

    不会驱逐任何 key，返回一个异常
  - allkeys-lru

    对所有key使用 LRU 算法进行删除
  - volatile-lru

    对所有设置了过期时间的key 使用 LRU 算法进行删除
  - allkeys-random

    对所有 key 随机删除
  - volatile-random

    对所有设置了过期时间的 key 随机删除
  - allkeys-lfu

    对所有 key 使用 LFU 算法进行删除
  - volatile-lfu

    对所有设置了过期时间的 key 使用 LFU 算法进行删除
  - volatile-ttf

    删除马上要过期的 key
- maxmemory-samples

  样本个数，对 lru/lfu/ttl 类的策略有效
  样本数越小越不准确，性能消耗越小

- **发布订阅**

  3

  1. 发布的消息无持久化
  2. 订阅者只能收到订阅后发布的消息

  - subscribe channel
  - publish channel value

- **事务**

  4

  串行多个命令，防止其他命令插队

  ---特性---
  1. 单独的隔离性
  2. 没有隔离级别概念
  3. 不保证原子性

  - 操作
    - Multi

      开始组队
      组队阶段报错，执行时整个队列都会取消
    - Exec

      提交执行
      执行阶段报错，只有报错的命令不会被执行，其他命令不会回滚
    - discard

      放弃组队
    - watch

      在 mutil 之前监视若干 key，在事务执行之前若这些 key 被其他命令改动，则事务被打断
    - unwatch

      取消对 key 的监视
      exec, discard 之后不需要在 unwatch
  - 应用

    秒杀系统

    1. 并发超卖
    2. 乐观锁，遗留库存(乐观锁导致多个秒杀操作失效)，连接超时
    3. 连接池, LUA

- **持久化**

  5

1. 默认只开启 rdb
2. 同时开启时系统默认读取 aof

官方推荐两个都启用
数据一致性不敏感使用 rdb
只做内存缓存可以都不用
不建议单独使用 aof，可能会出现bug

- RDB(Redis DataBase)

  ---备份---
  在指定时间间隔将内存中的数据集快照写入磁盘，fork 一个子进程，子进程将数据写入一个临时文件，在用临时文件替换上次持久化的文件，主进程无IO操作，性能极高
  ---恢复---
  将快照文件读入内存

  ---优势---
  1. 便于大规模数据恢复
  2. 对数据完整性和一致性要求不高更适合使用
  3. 节省磁盘空间
  4. 恢复速度快

  ---缺点---
  1. fork 导致大致 2 倍的内存膨胀
  2. fork 写时拷贝技术，数据庞大时，比较消耗性能
  3. 最后一次快照数据可能丢失

  ---动态停止---
  redis-cli config set save ""

  - fork

    1. 复制一个完全一样的子进程
    2. 写时复制技术
    3. 一般共享一段物理内存，只有当进程空间内容发生变化时，才会将父进程的内容复制一份给子进程

  - save

    * After 3600 seconds (an hour) if at least 1 key changed
    * After 300 seconds (5 minutes) if at least 100 keys changed
    * After 60 seconds if at least 10000 keys changed

    save 只管保存，全部阻塞
    bgsave 异步执行
    lastsave 最后一次执行快照的时间

    flushall 也会生成空 dump.rdb，无意义

  - stop-writes-on-bgsave-error

    无法写入磁盘时，直接关掉 Redis 写操作

  - rdbcompression

    压缩存储

    LZF 算法
    消耗 CPU

  - rdbchecksum

    检查文件完整性

    CRC64
    10% 性能消耗

- AOF(Append Only File)

  ---备份---
  以日志的形式记录每一个写操作（增量保存），读操作不记录，文件只追加不修改

  ---恢复---
  根据日志文件的内容将写指令从前到后执行依次

  ---优势---
  1. 备份机制稳健，数据丢失率低
  2. AOF 可读，可操作修复

  ---劣势---
  1. 占用更多磁盘
  2. 恢复速度慢
  3. 每次读写同步时，有性能压力
  4. 存在bug，造成恢复失败

  - appendonley

    开启 aof

    appendfilename "appendonly.aof"

  - redis-check-aof --fix appendonly.aof

    修复损坏的 aof 文件

  - appendfsync

    always - 每次写入立即记入日志，性能较差
    everysec - 每秒计入一次，可能丢失1秒的数据
    no - 不主动同步，把同步时机交给操作系统

  - bgrewriteaof

fork 一个子进程将文件重写，4.0 后就是把 rdb 的快照以二进制形式附在新的 aof 头部，作为已有的历史数据

---no-appendfsync-on-rewrite---
yes 只写缓存，数据安全低，性能高
no 写入磁盘，数据安全高，性能低

---auto-aof-rewrite-percentage---
默认 100%，即重写后 2倍时触发重写

---auto-aof-rewrite-min-size---
默认 64MB，达到这个值开始重写

aof_buf，aof_rewrite_buf

- **主从复制**

6

主机数据更新后根据配置和策略自动同步到备机

一主二仆
薪火相传
反客为主

---优势---
1. 读写分离，性能扩展
2. 容灾快速恢复

---劣势---
复制延迟

- 配置

daemonize yes
include /home/aurelius/data/redis/redis6379.conf
pidfile /var/run/redis_6379.pid
logfile /home/aurelius/data/redis/logs/redis6379.log
port 6379
dbfilename dump6379.rdb

关掉 appendonly 或者更改 appendfilename 的名字
appendonly no

从机优先级，越小优先级越高，用于选举，默认 100
replica-priority 100

主机配置密码时必配
replicaof masterip masterport
masterauth password

  - info replication

    查看运行状态

  - slaveof ip port

    成为某个实例的从服务器

  - slaveof no one

    从机变主机，反客为主

- 复制原理

1. slave 启动成功连接到 master 后，发送一个 sync 命令
2. master 接收到命令后启动后台存盘进程，同时收集所有写指令，在后台进程执行完后，将整个数据文件发送到 slave，以完成一次完全同步
3. slave 接收到全量数据文件后加载到内存中
4. master 继续将新收集的写指令发送给 slave，完成同步
只要重写连接 master，就要完成一次完全同步

- sentinel

后台监控主机，主机故障，自动切换从机为主机

---sentinel.conf---
sentinel master imaster 127.0.0.1 1
imaster - 监控对象的服务器名称
1 - 统一迁移的哨兵数量

---选举---
1. 选择优先级高的
2. 选择偏移量大的
3. 选择 runid 小的

---问题---
配置了 protected-mode yes & 密码

  - redis-sentinel

    redis-sentinel sentinel.conf

- **集群**

7

Redis 集群通过分区提供一定程度的可用性，即使集群中有一部分节点失效或无法通讯，集群也可以继续处理命令请求

扩容
并发分摊
去中心化，水平扩容

---优势---
1. 实现扩容
2. 分摊压力
3. 无中心配置相对简单

---劣势---
1. 不同 slots 的多键操作不被支持
2. 多键的事务不被支持
3. lua 脚本不被支持
4. 集群方案出现较晚，方案迁移难度大

- 配置

  include /home/aurelius/data/redis/redis.conf
  port 6379
  pidfile /var/run/redis_6379.pid
  dbfilename dump6379.rdb
  dir /home/aurelius/data/redis/cluster
  logfile /home/aurelius/data/redis/cluster/redis6379.log

  # 开启集群模式
  cluster-enabled yes
  # 节点配置文件名
  cluster-config-file nodes-6379.conf
  # 节点失联时间，毫秒，用于自动从主切从
  cluster-node-timeout 15000

- 合成集群

  redis-cli --cluster create --cluster-replicas 1 192.168.100.90:6379 192.168.100.90:6380 192.168.100.90:6381 192.168.100.90:6389 192.168.100.90:6390 192.168.100.90:6391

  ip 需要正式地址
  replicas 1 采用一主一从配置

- slots

  16384 个，键属于其中一个 slots

  key 所属 slots = CRC16(key) % 16384

  不同 slots 的键不能使用 mget, mset 等多键操作

  - redis-cli -c -p 6379

    -c 采用集群策略连接，自动切换响应主机

  - {}

    定义组，将多个 key 放在同一个 slot

- 故障恢复

  15 s超时，从变主
  主机恢复后，变为从

  - cluster-require-full-coverage

    yes - 某段 slots 的主从都挂掉时，整个集群都挂掉
    no - 某段 slots 的主从都挂掉时，该 slots 数据全部不可用，无法存储

- **应用问题**

8

- 缓存穿透

  对一个不存在的键反复请求，请求都作用在 DB 上
  在缓存和数据库都不存在该数据

  - 对空值缓存
  - 白/黑名单

    考虑使用 Bitmap

  - 布隆过滤器
  - 命中率监控

- 缓存击穿

  对一个过期的 key 做高并发请求，请求瞬间作用在 DB 上

  - 预设热门 key
  - 实时调整过期时间
  - 锁

    只放行一个 load db

- 缓存雪崩

  对大量过期的 key 高并发请求，请求瞬间作用在 DB 上

  - 多级缓存
  - 锁或队列
  - 过期更新

- 过期时间分散

- 分布式锁

  跨 JVM 的互斥机制，控制贡献资源的访问

  1. 互斥，只有一个客户端可以持有锁
  2. 无死锁，宕机崩溃有超时解锁机制
  3. 加锁解锁同一人
  4. 加锁解锁原子性

  ---实现---
  1. 数据库
  2. 缓存（Redis） - 性能好
  3. Zookeeper - 可靠性好

  ---Redis---
  set key value ex 100 nx
  del key
  ---缓存续期---
  Redisson
  RedLock

  - 锁释放
  - 锁过期设定（防宕机）
  - 锁操作（原子性）
  - 只删除自己的锁
  - Lua 或事务（multi）
  - 缓存续期（Redisson）
  - （AP）异步复制导致锁丢失
  - IllegalMonitorStateException

    to unlock lock, not locked by current thread by node id:xxxx

新特性

9

- ACL

  acl list - 用户权限列表
  acl cat - 权限指令类别
  acl cat string - 指定类型的指令类别
  acl whoami - 当前用户
  acl setuser user on/off

  auth user password

- IO 多线程

  io-threads-reads yes - 开启IO 多线程，默认不开启
  io-threads 4 - 线程数

- Cluster 支持

  Redis 5 后将 redis-trib.rb 集成到 redis-cli
  redis-benchmark 开始支持 cluster 模式，多线程多个分片压测

- RESP3 通信协议

  优化服务端与客户端通信

- Client side caching
- Proxy 集群代理模式
- Modules API