**基于内存的可持久化数据结构**

Redis is what is called a key-value store, often referred to as a NoSQL database.

1. redis是单线程的，其所有操作都是原子的，不会因并发产生数据异常

2.使用高耗时的Redis命令是很危险的，会占用唯一的单线程的大量处理时间，导致所有的请求都被拖慢

example: (命令不区分大小写)

> SET name "tbz" 增/改

> GET name 查

> DEL name 删

> EXPIRE name 10 设定时限

> TTL name 查时限

> EXISTS id1 是否存在KEY

> SELECT 2切换db/domain/keyspace

**容器**(KEY)支持的数据结构​：string, list, set, sorted set, hash, HyperLogLog(计数估算)

> RPUSH friends "Alice" 尾追加list

> LPUSH friends "Bob" 头追加list

> LRANGE friends 0 -1 遍历

> MSET id1 one id2 two 并列写

> MGET id1 id2 并列读

> HSET user phone 13211200213 哈希类型写

> HGET user phone 哈希类型读

> HMSET alarm alarm\_id 1 alarm\_name "test" rule\_id "aggregation" 模拟写对象

> HGETALL alarm 模拟读对象

> SADD rule\_type aggre shield rename autoack 集合

**订阅-发布：**(不推荐，缺少容错，消息传递不受限db)

> subscribe message\_queue

> publish message\_queue "finished"

**事务** (不推荐)

> MULTI标记事务的开始

> SET statue ok

> EXEC 清除标记并执行/ DISCARD 清除标记并丢弃

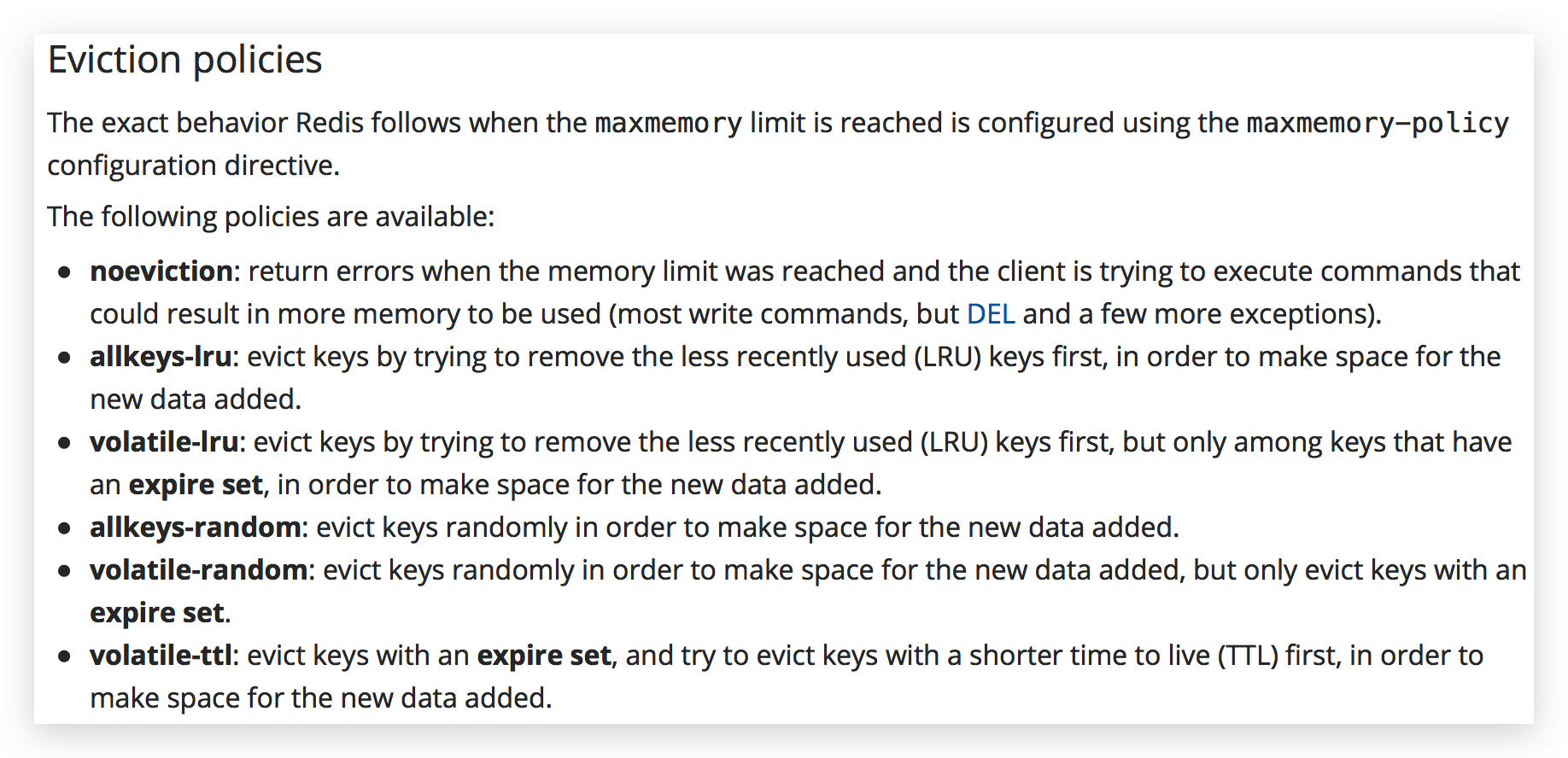
**运维**

> SAVE保存为文件，类似sqlite【定期的快照备份, dump.rdb】

> BGREWRITEAOF后台追加日志【记录写操作.aof】

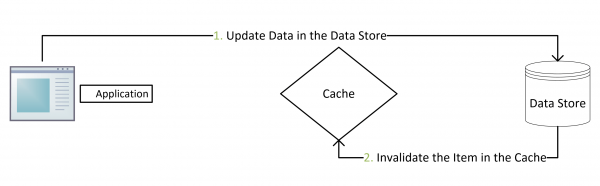
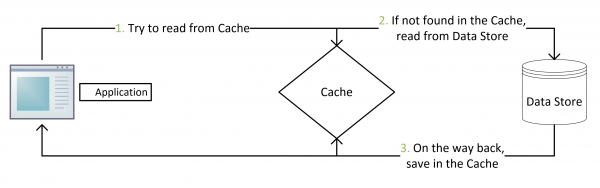
> CLIENT LIST当前所有客户端连接

> SLAVEOF host port指定master server并同步数据到自己(slave server)



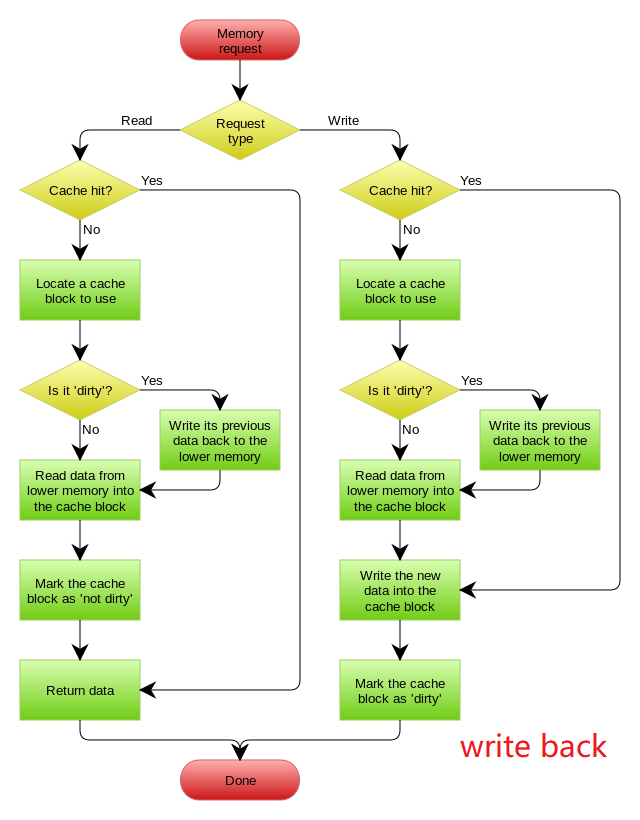
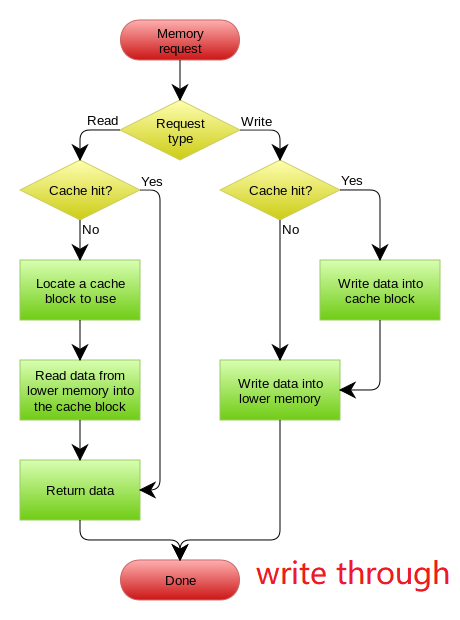
**流程设计**

1. 缓存的初衷：降低对数据来源的依赖。如果要求数据的强一致性，不建议使用缓存。
2. Cache Aside



更新DB完成后，才让cache失效的原因：避免cache失效后，更新DB执行前，有查询请求，导致cache在一段时间内的数据是过时的。

1. Read/Write Through写穿透
2. Write Back写返回



**cluster集群**

1. tcp协议
2. master-slave同步
3. shard数据分片(redis-cluster用hash slot定长取余，分配到对应master)
4. redis-cluster的Gossip协议P2P结构

**分布式锁**

·单节点

上锁：SET resource random NX PX 30000

释放：if GET resource == random:

then DEL resource

1. setnx，expire不能拆开，避免因故障永久占用不释放，造成死锁
2. 校验random，避免超时后释放在占用的锁，破坏锁的安全性
3. 主从复制未完成，master宕机，slave的锁不准确

·多节点

1. RedLock(在规定时间内，超过一半的节点上锁成功)
2. ZooKeeper(锁节点的心跳检测)