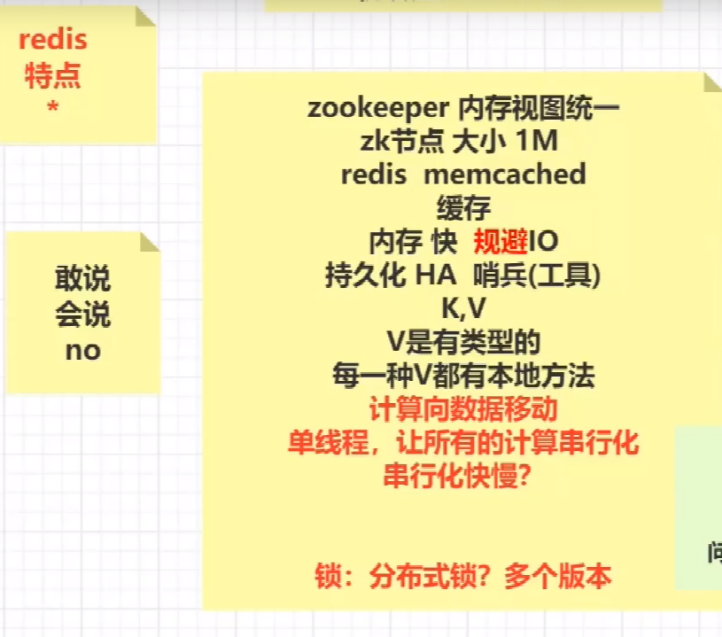
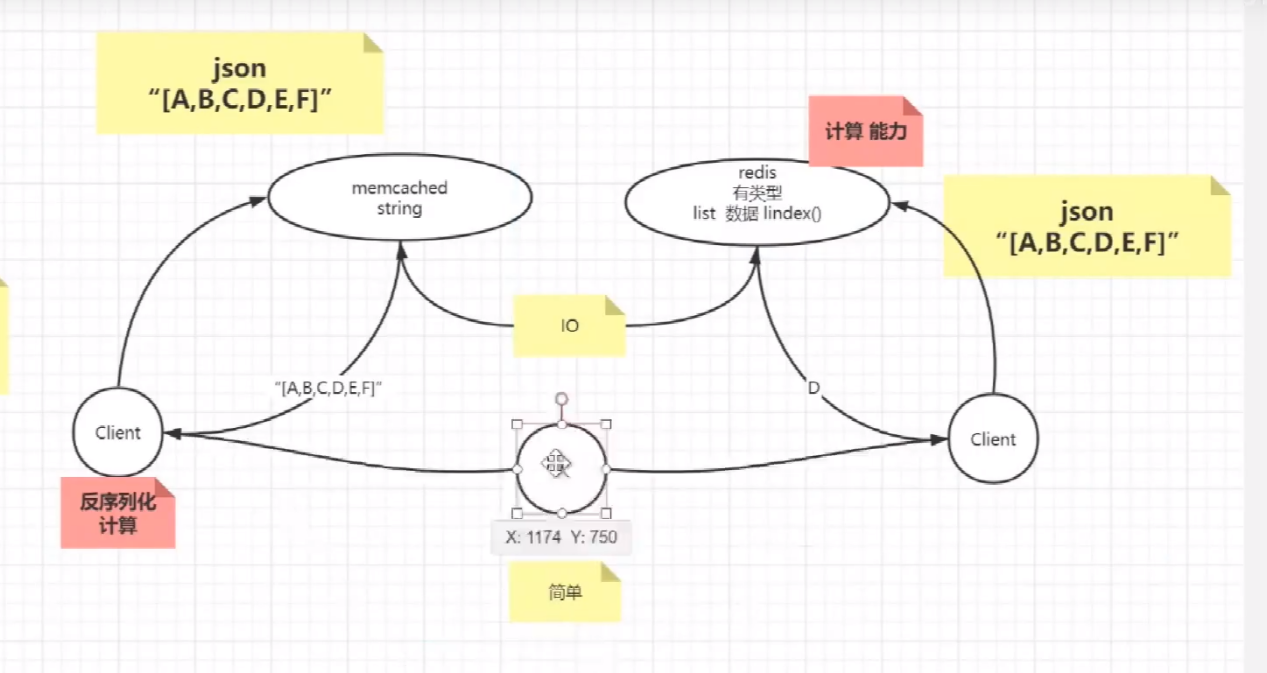
1. 特点
2. 单线程：不一定比多线程慢，因为多线程要加锁，最后也是串行化
3. Value：有类型、有本地方法（与memcache区别）





1. String

Key是一个对象，处理Key键值之外，还有它的type类型（可以判断操作的类型是否正确）、encoding、valueLength等等。在高并发的场景下，性能可以提高很多，因为可以先判断type或者encoding是否正确，再去做运算。

Redis收到的是字节流，不会去破坏其编码，因此是二进制安全的。

命令：

1. strlen key
2. type key : key类型（string、list、set、zset、hash）
3. object encoding key：value类型

Value为String类型命令：

1. set/mset key value [nx | xx]

nx：不存在的时候，才可以设置（创建，可用于分布式锁）

xx：存在的时候，才可以设置（更新）

1. setrange key start value
2. get/mget key value
3. getrange key start end
4. append key value

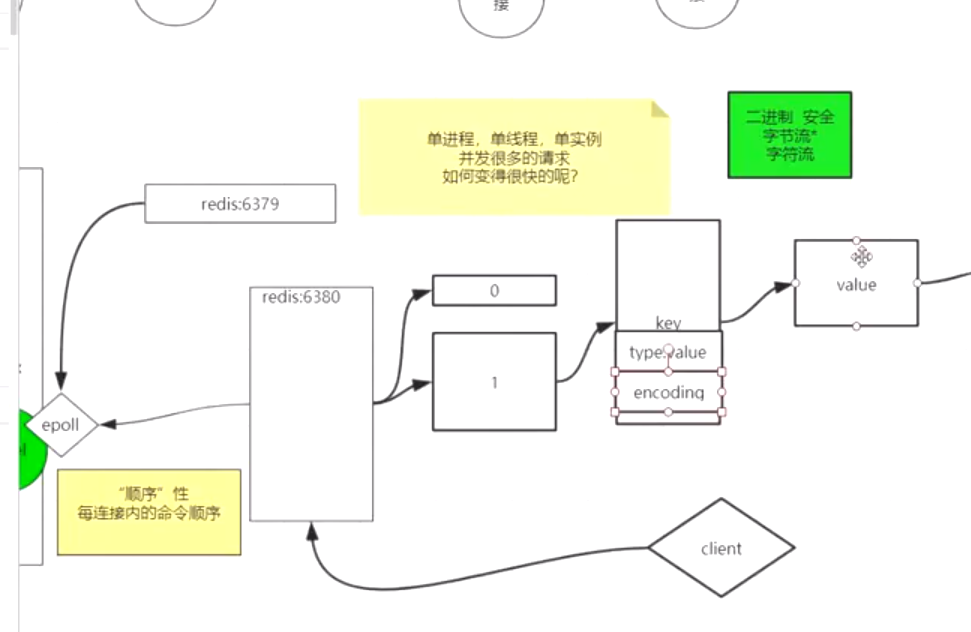
在key追加字符串

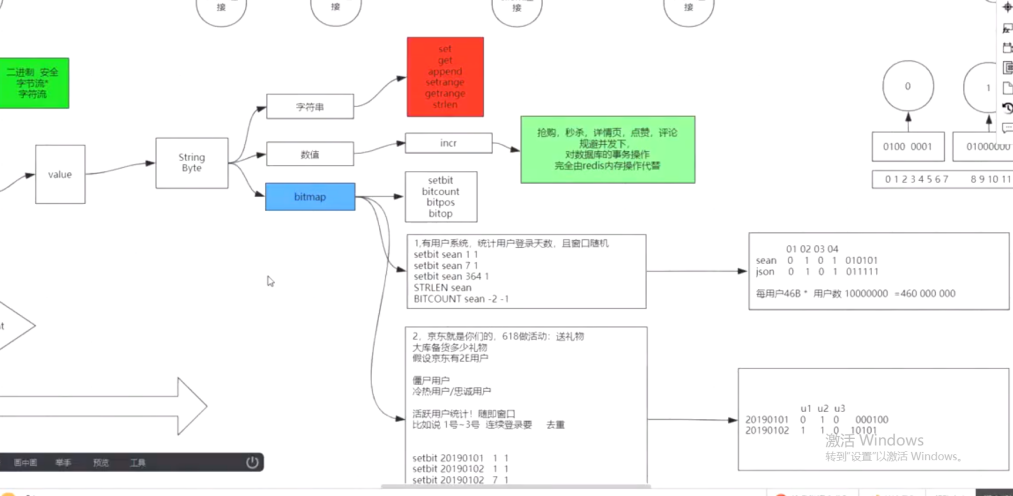
Value为INT类型命令：

1. incr key
2. incrby key increment
3. incrbyfloat key increment
4. decr key
5. decrby key decrment

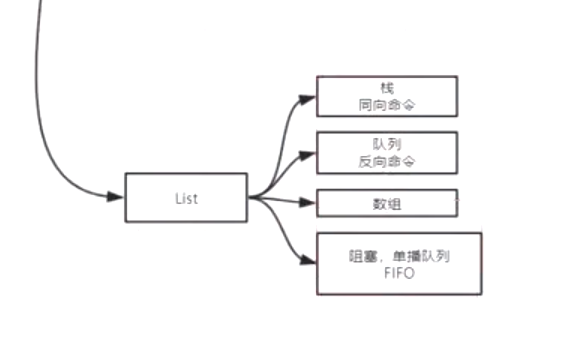
Value为BITMAP类型命令：

1. setbit 设置
2. bitop op key [start end]运算
3. bitpos key bit [start end]查找
4. bitcount key bit[start end]统计

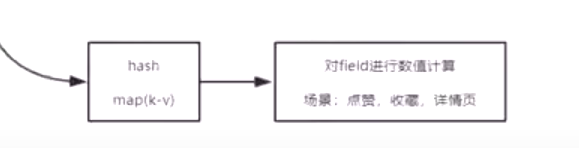




1. List
2. 增：lpush、rpush
3. 删：lpop、rpop、lrem key count element（删除指定数据）
4. 改：lset key index value
5. 查：lindex key index、lrange
6. 插：linsert key BEFORE|AFTER provit elemnt（在provit后面/前面插入）

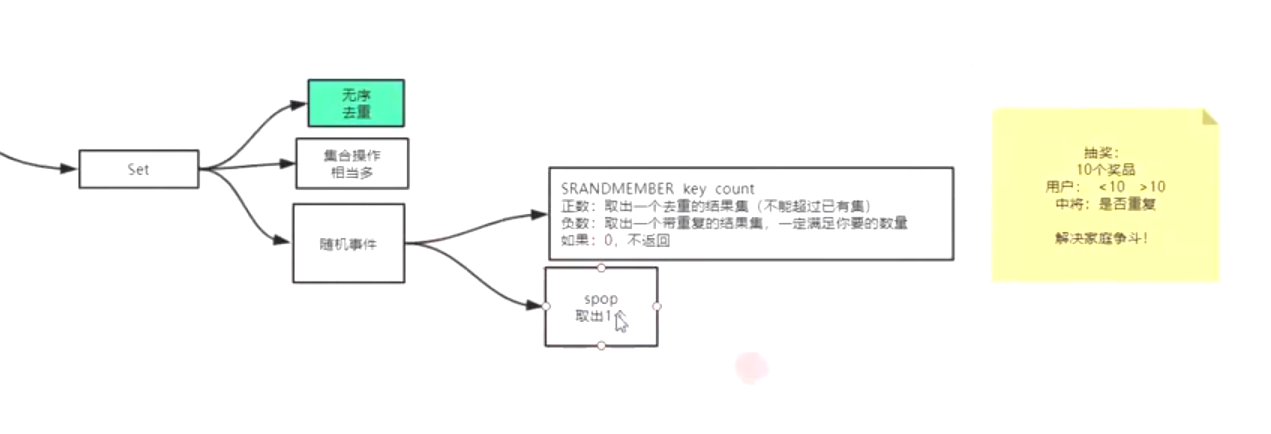


1. Hash
2. 增：hset、hmset
3. 查：hget、hmget、hkeys、hvals、hgetall
4. 运算：hincrby key filed increment、hincrbyfloat byfield increment



1. Set
2. 增：sadd
3. 查：smembers key、随机取出数据 srandmember key count
4. 删：srem key elem、spop
5. 运算：

* 交：sinter、sinterstore
* 并：sunion、suionstore
* 差：sdiff key1 key2(key1-key2)、sdiffstore



1. Zset
2. 增：zadd key score elem
3. 查：

分数：zsocre key elem

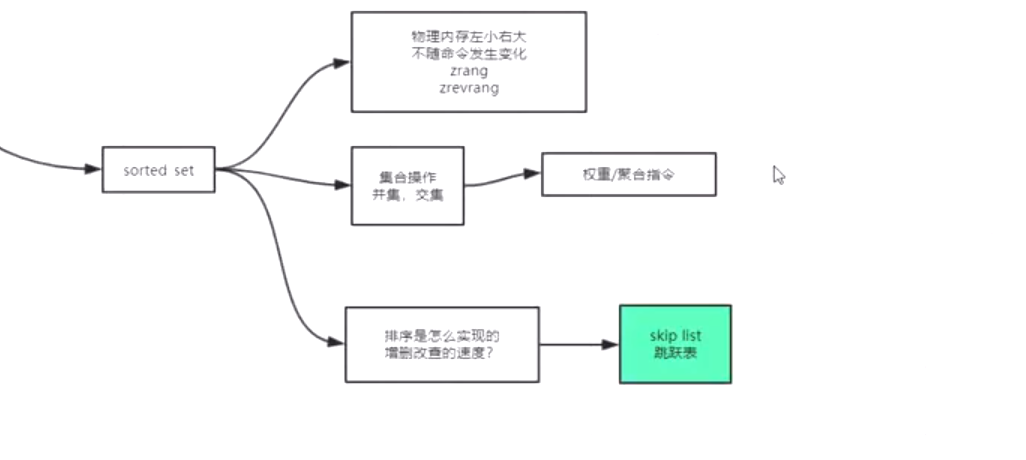
排名：zrank key elem

序列：

* 按照索引：zrange/zrevrange key min max
* 按照score：zrangebyscore/zrevrangebyscore key min max

1. 运算：zincrby key incrment elem

* 交：zinter[sum|min|max]、zinterstore[sum|min|max]
* 并：zunion[sum|min|max]、zuionstore[sum|min|max]
* 差：zdiff key1 key2(key1-key2)、zdiffstore



跳跃表：

Node

{

Node \*left;

Node \*right;

Node \*level;//为了缩小区间，让前面数据直接指向某个后面数据。

}  
 先找一级一级确定区间，最后到正确位置。（类似排序二叉树）

将区间存储下来。

