# 1、Redis数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| String  (常用命令：set,get,  decr,incr,  mget等) | 增 | jedis.set("key1","value1") |  |
| jedis.mset("key2","value2","key3","value3") | 一次性新增key2,key3及其对应值 |
| Jedis.setnx("key301", "value301") | 新增键值对时防止覆盖原先值 |
| Jedis.setex("key302", 2, "key302-2second") | 新增key303，并指定过期时间为2秒 |
| 删 | jedis.del("key3") |  |
| jedis.del(new String[]{"key1", "key2"}) |
| 改 | jedis.set("key1","value1-update") |  |
| jedis.append("key2","+appendString") |
| 查 | jedis.get("key1") |  |
| jedis.mget("key2","key3") | 一次性获取key201,key202各自对应的值 |
| Jedis.getrange("key302", 5, 7) | 获取子串 |
| Hash  (常用命令： hget,hset,  hgetall 等) | 增 | Jedis.hset("hashs", "key001", "value001")  Jedis.hset("hashs", "key002", "value002") |  |
| 删 | Jedis.hdel("hashs", "key002") |  |
| 改 | Jedis.hincrBy("hashs", "key001", 100l) |  |
| 查 | Jedis.hexists("hashs", "key003") | 判断key003是否存在 |
| Jedis.hget("hashs", "key004") | 获取key004对应的值 |
| Jedis.hmget("hashs", "key001", "key003") | 批量获取key001和key003对应的值 |
| Jedis.hkeys("hashs") | 获取hashs中所有的key |
| Jedis.hvals("hashs") | 获取hashs中所有的value |
| List  (常用命令: lpush,rpush,  lpop,rpop,  lrange等) | 增 | Jedis.lpush("stringlists", "vector")  Jedis.lpush("stringlists", "ArrayList") |  |
| 删 | Jedis.lpop("stringlists") | 出栈元素 |
| Jedis.lrem("stringlists", 2, "vector") | 删除列表中2个"vector"(后add进去的值先被删) |
| Jedis.ltrim("stringlists", 0, 3) | 删除下标0-3区间之外的元素 |
| 改 | Jedis.lset("stringlists", 0, "hello list!") | 修改列表中指定下标的值 |
| 查 | Jedis.lindex("stringlists", 2) | 获取下标为2的元素 |
| Jedis.lrange("stringlists", 0, -1) | 所有元素（-1表示倒数第一个元素） |
| Jedis.llen("stringlists") | 数组长度 |
| Jedis.sort("stringlists",sortingParameters) | 返回排序后的结果 |
| Set  (常用命令： sadd,spop,  smembers,  sunion 等) | 增 | jedis.sadd("sets", "element001")  jedis.sadd("sets", "element002") |  |
| 删 | jedis.srem("sets", "element003") |  |
| 查 | jedis.smembers("sets") | 查看sets集合中的所有元素 |
| jedis.sinter("sets1", "sets2") | sets1和sets2交集 |
| jedis.sunion("sets1", "sets2") | sets1和sets2并集 |
| jedis.sdiff("sets1", "sets2") | sets1和sets2差集 |
| ZSet  (常用命令： zadd,zrange,  zrem,zcard等) | 增 | Jedis.zadd("zset", 7.0, "element001") Jedis.zadd("zset", 9.0, "element002") |  |
| 删 | Jedis.zrem("zset", "element002") |  |
| 查 | Jedis.zrange("zset", 0, -1) | zset集合中的所有元素 |
| Jedis.zcard("zset") | 统计zset集合中的元素中个数 |
| Jedis.zcount("zset", 1.0, 8.0) | 统计zset集合中权重在 (1.0—8.0) 范围的元素个数 |
| Jedis.zscore("zset", "element002") | 查看zset集合中element004的权重 |
| Jedis.zrange("zset", 1, 2) | 查看下标1到2范围内的元素值 |

# 2、redis 和 map/guava 区别

（1）map/guava 轻量以及快速，生命周期随着 jvm 的销毁而结束，并且在多实例的情况下，每个实例都需要各自保存一份缓存，缓存不具有一致性

（2）redis 在多实例的情况下，各实例共用一份缓存数据，缓存具有一致性。

（3）redis缺点是需要保持服务的高可用，整个程序架构上较为复杂。

# 3、redis 和 Memecache 区别

（1）redis支持更丰富的数据类型（支持更复杂的应用场景）：Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，zset，hash等数据结构的存储。memcache支持简单的数据类型，String

（2）redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用,而Memecache把数据全部存在内存之中

（3）集群模式：memcached没有原生的集群模式，需要依靠客户端来实现往集群中分片写入数据；但是 redis 目前是原生支持 cluster 模式的

（4）Redis使用单线程的多路 IO 复用模型，而Memcached是多线程、非阻塞IO复用的网络模型；

# 4、过期时间

set key时，可以设置expire time (过期时间)

两种删除方式：

（1）定期删除：redis默认每隔 100ms 就随机抽取一些设置了过期时间的key，检查其是否过期，如果过期就删除。

（2）惰性删除：当系统查找key时，发现已过期则将其删除。(定期删除存在过期 key 到了时间并没有被删除掉)

# 5、淘汰机制

① volatile-lru：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：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。

4.0版本后增加以下两种：

⑦ volatile-lfu：从已设置过期时间的数据集(server.db[i].expires)中挑选最不经常使用的数据淘汰

⑧ allkeys-lfu：当内存不足以容纳新写入数据时，在键空间中，移除最不经常使用的key

# 6、持久化

① 快照RDB：将某时间点的数据存储到一个临时文件中（不安全(易丢失数据)，文件小、恢复速度快）

RDB默认配置：

save 900 1 #900秒后，若有1个key发生变化，则创建快照

save 300 10 #300秒后，若有10个key发生变化，则创建快照

save 60 10000 #60秒后，若有10000个key发生变化，则创建快照

② AOF：将“操作+数据”以格式化指令的方式存到日志文件尾部（安全，磁盘IO开支大、文件大、恢复速度慢）

AOF 三种方式：

always #每次有数据修改，都会写入AOF文件

everysec #每秒钟同步一次

no #由操作系统决定何时同步 (buffer填充情况/通道空闲…)

注：Redis 4.0 开始支持RDB和AOF的混合持久化（默认关闭，可以通过配置项 aof-use-rdb-preamble 开启）。

AOF重写：创建新的AOF文件，直接把 RDB 的内容写到 AOF 文件开头，并将重写缓冲区中的所有内容追加到新AOF文件的末尾（重写缓冲区用于记录服务器执行的所有写命令）

# 7、redis事务

Redis 事务可以一次执行多个命令， 并且带有以下三个重要的保证：

①批量操作在发送 EXEC 命令前被放入队列缓存。

②收到 EXEC 命令后进入事务执行，事务中任意命令执行失败，其余的命令依然被执行。

③在事务执行过程，其他客户端提交的命令请求不会插入到事务执行命令序列中

Redis 事务先以 MULTI 开始一个事务， 然后将多个命令入队到事务中， 最后由 EXEC 命令触发事务

# 8、缓存雪崩/缓存穿透问题

缓存雪崩：缓存同一时间大面积的失效，后面的请求都会落到数据库上，造成数据库短时间内承受大量请求而崩掉。

解决办法：①事前：尽量保证整个 redis 集群的高可用性，发现机器宕机尽快补上。选择合适的内存淘汰策略。

②事中：本地ehcache缓存 + hystrix限流&降级，避免MySQL崩掉

③事后：利用 redis 持久化机制保存的数据尽快恢复缓存

缓存穿透：故意请求缓存中不存在的数据，导致所有请求都落到数据库上，造成数据库短时间内承受大量请求而崩掉。

解决办法：采用布隆过滤器，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被 这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。

# 9、高并发情况下，key竞争

使用分布式锁（zookeeper 和 redis 都可以实现分布式锁）

# 10、双写数据一致性问题

读请求和写请求串行化，串到一个内存队列里去