Redis课程2

1. 过期时间的设置及原理分析
2. expire key seconds

ttl key 监控过期时间

persist key 取消过期时间

1. setex(String key, int seconds,String value)

原理：

1. 消极方法（passive way）

当应用访问key时发现已过期则删除。

1. 积极方法（active way）

周期性从设置了过期时间的key中选择一部分的key进行删除。

1. 随机测试20个带有timeout信息的key
2. 删除发送过期的key
3. 如果超过25%的key过期删除，则重复整个流程
4. 发布订阅模式

pub/sub模式

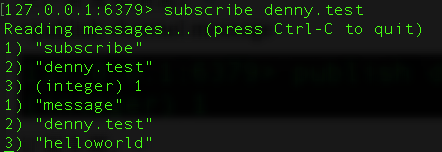
生产者发布消息：

>publish channel message



订阅主题：

>subscribe channel



只能作为简单的发布-订阅模式，不支持多协议、不支持消息持久化存在消息丢失、不能保障可靠性、不支持事务等。可以用于站内简单通信如聊天工具等。

1. Redis持久化及原理分析

Redis 支持数据持久化

RDB、AOF

RDB：

当符合【条件】的时候，fork子进程，生成rdb.dump.rdb快照文件，

1. 配置规则，redis.conf配置参数：

save seconds changes

save 900 1 //900秒内有一个或以上的key被更改过

save 300 10

save 60 10000

满足以上其中一个条件就会触发生成快照文件。

1. 用户主动执行save或者bgsave。

save：会阻塞所有客户端的请求。

bgsave：后台异步快照任务

1. flushall 清空内存中所有数据。执行时如快照规则配置存在就会执行快照任务。
2. 执行复制操作。

两次快照间隔期间，执行了内存操作会存在数据丢失。

AOF：

实时日志，每执行事务性操作会把执行变更的数据写入到磁盘中，消耗性能，保证数据尽

可能不丢失。在redis.conf配置参数：

|  |
| --- |
| appendonly yes  appendfilename “appendonly.aof” |

如果RDB和AOF同时开启以启用AOF的快照文件加载数据。AOF快照文件会记录所有

的数据事务操作日志，AOF文件会很大但实际数据量并不大。如何优化？

|  |
| --- |
| auto-aof-rewrite-percentage 100  auto-aof-rewrite-min-size 64mb |

auto-aof-rewrite-percentage：当前AOF文件大小超过上次重写文件的指定百分比时触发

重写。 默认100%。

auto-aof-rewrite-min-size：AOF文件允许重写的最小长度。

AOF重写(fork子进程进行)：重写可以去除数据的中间执行过程，直接保留最终数据命令。

参数appendfsync，在Redis中对AOF调用write写入后，何时再调用fsync将其写 到

磁盘上，通过appendfsync选项来控制

|  |
| --- |
| appendfsync everysec //默认每隔一秒进行一次fsync调用，将缓冲区中的数据写到磁盘  appendfsync always //每一次写操作都会调用一次fsync，这时数据是最安全的，性能受影响  appendfsync no //不主动调用fsync去将AOF日志内容同步到磁盘，依赖操作系统 |

1. Redis的内存回收策略

LRU 最近最少使用,淘汰数据。

LFU根据数据的历史访问频率来淘汰数据，如果数据过去被访问多次，那么将来被访问的

频率也更高。访问频率最低的淘汰。

|  |
| --- |
| maxmemory-policy  noeviction：不淘汰删除数据，如果内存不足，写入时返回错误。  volatile-lru：淘汰设置了过期时间并且最近最少使用的数据  allkeys-lru：最近最少使用数据淘汰  volatile-lfu：  allkeys-lfu：  volatile-random：随机淘汰设置了过期时间的key  allkeys-random：随机key淘汰  volatile-ttl：淘汰即将过期的数据。 |

1. Redis单线程为什么性能很高？

内存和网络的宽带，多路复用。同时处理大量的客户端请求。单线程避免多线程竞争和上下文切换。

同步阻塞、同步非阻塞、异步阻塞、异步非阻塞。

同步/异步：用户线程和内核交互方式

阻塞/非阻塞：线程调用内核做IO操作

同步阻塞：

用户进程或线程发送IO请求，等待内核空间返回数据到用户空间。

同步非阻塞：

用户进程或线程发送IO请求，内核空间接受到立即返回。用户进程或线程向内核空间循环请求获取数据。

异步阻塞(IO多路复用)：

用户进程或线程发送IO请求，同时事件监视操作状态（阻塞多个IO操作），内核空间返回操作状态，如可读。用户进程或线程执行相应操作。一个用户进程或线程可以同时处理多个IO请求。

1. Lua脚本在Redis中的应用

pileline

使用Lua脚本保证多个客户端执行多个命令的复用性、原子性、有序性和减少网络开销。

Lua安装:

|  |
| --- |
| curl -R -O http://www.lua.org/ftp/lua-5.3.5.tar.gz  tar zxf lua-5.3.5.tar.gz  cd lua-5.3.5  make linux test或make macosx test  sudo make install |

在Lua脚本中去执行Redis命令：

redis.call()

eval 执行脚本：

|  |
| --- |
| ./redis-cli --eval /Users/denny/Documents/LuaScripts/ratelimit.lua 192.168.3.14 , 10 10 |

ratelimit.lua 脚本：

|  |
| --- |
| local key = "ratelimit:"..KEYS[1]  local limit = tonumber(ARGV[1])  local expireTime = ARGV[2]  local times = redis.call('incr', key)  if times == 1 then  redis.call('expire', key, expireTime)  end  if times > limit then  return 0  end  return 1 |

客户端执行编写脚本执行：

