私塾在线《高级软件架构师实战培训阶段二》

——跟着cc学架构系列精品教程

本部分课程概览

- 根据实际的应用需要，学习Redis的相关知识，以快速上手、理解并掌握Redis，大致包括：

1：Redis简介、安装和基础入门
2：Redis的数据类型、基本操作和配置详解
3：Redis的持久化：RDB和AOF
4：Redis的事务
5：Redis的发布订阅模式
6：Redis的复制
7：Redis的集群
8：Lua脚本开发
9：杂项知识，包括：Redis的安全、管理等
10：适合使用Redis的场景
11：Redis的性能优化
12：Redis的Java客户端，以及和Spring的集成开发

Redis简介

- Redis介绍（REmote DIctionary Server 远程字典服务器）

  Redis是一个使用C编写的、开源的、Key-Value型、基于内存运行并支持持久化的NoSQL数据库，是当前最热门的NoSql数据库之一。

- Redis和其它数据库或缓存服务器的对比

Name	Type	Data storage options	Query types	Additional features
Redis	In-memory non-relational database	Strings, lists, sets, hashes, sorted sets	Commands for each data type for common access patterns, with bulk operations, and partial transaction support	Publish/Subscribe, master/slave replication, disk persistence, scripting (stored procedures)
memcached	In-memory key-value cache	Mapping of keys to values	Commands for create, read, update, delete, and a few others	Multithreaded server for additional performance
MySQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions	SELECT, INSERT, UPDATE, DELETE, functions, stored procedures	ACID compliant (with InnoDB), master/slave and master/master replication
MongoDB	On-disk non-relational document store	Databases of tables of schema-less BSON documents	Commands for create, read, update, delete, conditional queries, and more	Supports map-reduce operations, master/slave replication, sharding, spatial indexes

Redis的安装

- Redis安装

1：去官网下载最新的版本：http://redis.io/download ，这里用的是3.0.2

2：解压后，进入解压好的文件夹

3：redis的安装非常简单，因为已经有现成的Makefile文件，所以直接先make，然后make install就可以了

4：安装的位置在/usr/local/bin ，有：

（1）redis-benchmark：性能测试工具，测试Redis在你的系统及配置下的读写性能

（2）redis-check-aof：用于修复出问题的AOF文件

（3）redis-check-dump：用于修复出问题的dump.rdb文件

（4）redis-cli：Redis命令行操作工具

（5）redis-sentinel：Redis集群的管理工具

（6）redis-server：Redis服务器启动程序

5：启动Redis的时候，只有一个参数，就是指定配置文件redis.conf的路径。redis.conf在解压的文件夹里面有，复制一个出来，按需修改即可，也可--port来指定端口

6：连接Redis并操作，使用redis-cli，如果有多个实例，可以redis-cli -h 服务器ip -p 端口

7：关闭Redis，redis-cli shutdown，如果有多个实例，可以指定端口来关闭：redis-cli -p 6379 shutdown

Redis基础知识

- 单进程

Redis的服务器程序采用的是单进程模型来处理客户端的请求。对读写等事件的响应是通过对epoll函数的包装来做到的。

Redis的实际处理速度完全依靠主进程的执行效率，假如同时有多个客户端并发访问服务器，则服务器处理能力在一定情况下将会下降。假如你要提升服务器的并发能力，那么可以采用在单台机器部署多个redis进程的方式。

- 多数据库

1：Redis每个数据库对外都是以从0开始递增的数字来命名，默认16个数据库，默认使用0号数据库，可以使用Select 数字 来选择要使用的数据库
2：使用Dbsize可以查看当前数据库的key的数量
3：可以在多个数据库间移动数据，使用move key 目的数据库编号 就可以了
4：使用flushdb可以清除某个数据库的数据
5：Redis不支持自定义数据库名字
6：Redis不支持为每个数据库设置不同的访问密码
7：多个数据库之间并不是完全独立的，FlushAll可以清空全部的数据
8：Redis的数据库更像是一个命名空间

Redis的数据类型

- Redis的key

Redis的key是字符串类型，如果中间有空格或者转义字符等，要用“”。

1：命名建议：对象类型:对象ID:对象属性

2：多个单词之间以“.”来分隔

3：Key的命名，应该在可读的情况下，尽量简短

- Redis的Value支持五种类型

1：String：字符串，可以存储String、Integer、Float型的数据，甚至是二进制数据，一个字符串最大容量是512M

2：List：字符串List，底层实现上不是数组，而是链表，也就是说在头部和尾部插入一个新元素，其时间复杂度是常数级别的；其弊端是：元素定位比数组慢

3：Set：字符串Set，无序不可重复，是通过HashTable实现的

4：Hash：按Hash方式来存放字符串

5：ZSet：字符串Set，有序且不可重复，根据Score来排序。底层使用散列表和跳跃表来实现，所以读取中间部分数据也很快

Redis的基本操作-1

- 对Keys的操作命令

1：Keys：获得符合规则的键名列表

格式是keys pattern，pattern支持glob风格通配符格式：

（1）？ 匹配一个字符

（2）* 匹配任意个字符

（3）[] 匹配中括号内的任一字符，可以用-来表示一个范围

（4）\x 匹配字符x，用于转义符号

2：exists：判断键值是否存在，格式是exists key

3：del：删除key，格式是del key。

小技巧：Del命令不支持通配符，可以结合linux管道和xargs命令来自定义删除，示例如下：redis-cli keys k* | xargs redis-cli del

4：type：获得键值的数据类型，格式是type key

5：rename：改名，格式是rename oldKey newKey

6：renamenx：如果不存在则改名，格式是rename oldKey newKey

Redis的基本操作-2

- 对String类型的操作命令

1：get、set、del：获取key的值、设置key和值、删除key
2：incr、decr：递增和递减整数值，格式是incr key值
3：incrby、decrby：递增和递减整数值，可指定增减的数值，格式是incrby key值 正负数值
4：incrbyfloat：递增和递减浮点数值，格式是incrbyfloat key值 正负数值
5：append：在尾部追加值，格式是append key值 追加的值
6：getrange：获取指定索引范围内的值，格式是getrange key值 起始索引 结束索引
7：setrange：从索引位置开始设置后面的值，格式是setrange key值 offset索引 值
8：strlen：返回键值的长度，格式是strlen key值
9：mget：同时获得多个键的值，格式是mget 多个key值
10：mset：同时设置多个键值对，格式是mset key值 value ， key和value可以多对
11：bitcount：获取范围内为1的二进制位数，格式是bitcount key值 [start end]
12：getbit：获取指定位置的二进制位的值，格式是getbit key值 offset索引
13：setbit：设置指定位置的二进制位的值，格式是setbit key值 offset索引 值
14：bitop：对多个二进制值进行位操作，格式是bitop 操作 目的key key值1 key值2，操作有and、or、xor、not，key值可以是多个
15：getset：原子的设置key的值，并返回key的旧值 ，格式是getset key value

Redis的基本操作-3

- 对List类型的操作命令

1：lpush/rpush：添加值，格式是 rpush list的key item项的值，值可以是多个
2：lrange：按索引范围获取值，格式是 lrange list的key 起始索引 终止索引，-1表示最后一个索引
3：lindex：获取指定索引的值，格式是 lindex list的key 索引号
4：lpop/rpop：弹出值，格式是 lpop list的key
5：llen：获取元素个数，格式是llen list的key
6：lrem：删除元素，格式是lrem list的key 数量 item项的值，数量可正负，表示从左或右删除，如果数量为0，表示删除全部与给定值相等的项
7：ltrim：保留指定索引区间的元素，格式是ltrim list的key 起始索引 结束索引
8：blpop/brpop：弹出值，格式是blpop list的key值 过期时间，key可以是多个，如果没有值，会一值等待有值，直到过期
9：rpoplpush：将元素从一个列表转移到另外一个列表，格式是rpoplpush 源list的key值 目的list的key值
10：brpoplpush：将元素从一个列表转移到另外一个列表，格式是brpoplpush 源list的key值 目的list的key值 过期时间
11：lset：设置指定索引的值，格式是lset list的key 索引 新的值
12：linsert：插入元素，格式是linsert list的key before|after 定位查找的值 添加的值

Redis的基本操作-4

- 对Set类型的操作命令

1：sadd：添加元素，格式是 sadd set的key item项的值，item项可以多个
2：smembers：获取集合中所有元素，格式是 smembers set的key
3：sismember：判断元素是否在集合众，格式是 sismember set的key item项的值
4：srem：删除元素，格式是 srem set的key item项的值
5：scard：获取集合中元素个数，格式是scard set的key
6：srandmember：随机获取集合中的元素，格式是srandmember set的key [数量]，数量为正的时候，会随机获取这么多个不重复的元素；如果数量大于集合元素个数，返回全部；如果数量为负，会随机获得这么多个元素，可能有重复
7：spop：弹出元素，格式是spop set的key
8：smove：移动元素，格式是smove 源set的key 目的set的key item项的值
9：sdiff：差集，返回在第一个set里面而不在后面任何一个set里面的项，格式是sdiff set的key 用来比较的多个set的key
10：sdiffstore：差集并保留结果，格式是命令 存放结果的set的key set的key 用来比较的多个set的key
11：sinter：交集，返回多个set里面都有的项，格式是sinter 多个set的key
12：sinterstore：交集并保留结果，格式是sinter 存放结果的set的key 多个set的key
13：sunion：并集，格式是sunion 多个set的key
14：sunionstore：并集并保留结果，格式是sunionstore 存放结果的set的key 多个set的key

Redis的基本操作-5

- 对Hash类型的操作命令

1：hset：设置值，格式是hset Hash的Key 项的key 项的值
2：hmset：同时设置多对值，格式是hmset Hash的Key 项的key 项的值，项的key和值可多对
3：hgetall：获取该Key下所有的值，格式是hgetall Hash的Key
4：hget：获取值，格式是hget Hash的Key 项的key
5：hmget：同时获取多个值，格式是hmget Hash的Key 项的key，项的key可以是多个
6：hdel：删除某个项，格式是hdel Hash的Key 项的key
7：hlen：获取Key里面的键值对数量，格式是hlen Hash的Key
8：hexists：判断键值是否存在，格式是hexists Hash的Key 项的key
9：hkeys：获取所有Item的key，格式是hkeys Hash的Key
10：hvals：获取所有Item的值，格式是hvals Hash的Key
11：hincrby：增减整数数字，格式是hincrby Hash的Key 项的key 正负整数
12：hincrbyfloat：增减Float数值，格式是hincrbyfloat Hash的Key 项的key 正负float
13：hsetnx：如果项不存在则赋值，存在时什么都不做，格式是hsetnx Hash的Key 项的key 项的值

Redis的基本操作-6

- 对ZSet类型的操作命令

1：zadd：添加元素，格式是zadd zset的key score值 项的值，Score和项可以是多对，score可以是整数，也可以是浮点数，还可以是+inf表示正无穷大，-inf表示负无穷大

2：zrange：获取索引区间内的元素，格式是zrange zset的key 起始索引 终止索引 (withscores)

3：zrangebyscore：获取分数区间内的元素，格式是zrangebyscore zset的key 起始score 终止score (withscores)，默认是包含端点值的，如果加上“(”表示不包含；后面还可以加上limit来限制

4：zrem：删除元素，格式是zrem zset的key 项的值，项的值可以是多个

5：zcard：获取集合中元素个数，格式是zcard zset的key

6：zincrby：增减元素的Score，格式是zincrby zset的key 正负数字 项的值

7：zcount：获取分数区间内元素个数，格式是zcount zset的key 起始score 终止score

8：zrank：获取项在zset中的索引，格式是zrank zset的key 项的值

9：zscore：获取元素的分数，格式是zscore zset的key 项的值，返回项在zset中的score

10：zrevrank：获取项在zset中倒序的索引，格式是zrevrank zset的key 项的值

11：zrevrange：获取索引区间内的元素，格式是zrevrange zset的key 起始索引 终止索引 (withscores)

12：zrevrangebyscore：获取分数区间内的元素，格式是zrevrangebyscore zset的key 终止score 起始score (withscores)

13：zremrangebyrank：删除索引区间内的元素，格式是zremrangebyrank zset的key 起始索引 终止索引

14：zremrangebyscore：删除分数区间内的元素，格式是命令 zset的key 起始score 终止score

Redis的基本操作-7

15：zinterstore：交集，格式是ZINTERSTORE dest-key key-count key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]

16：zunionstore：并集，格式是ZUNIONSTORE dest-key key-count key [key ...] [WEIGHTS weight [weight ...]] [AGGREGATE SUM|MIN|MAX]

- 排序

1：sort：可以对List、Set、ZSet里面的值进行排序。格式是SORT source-key [BY pattern] [LIMIT offset count] [GET pattern [GET pattern ...]] [ASC|DESC] [ALPHA] [STORE dest-key]

2：by：设置排序的参考键，可以是字符串类型或者是Hash类型里面的某个Item键，格式是 Hash键名:*->Item键。设置了by参考键，sort将不再依据元素的值来排序，而是对每个元素，使用元素的值替换参考键中的第一个”*”，然后获取相应的值，再对获得的值进行排序。

如果参考键不存在，默认为0。

如果参考键值一样，再以元素本身的值进行排序。

3：get：指定sort命令返回结果包含的键的值，形如： Hash键名:*->Item键，可以指定多个get，返回的时候，一行一个。如果要返回元素的值，用get #。

4：对较大数据量进行排序会严重影响性能，使用建议：

（1）尽量减少待排序集合中的数据

（2）使用limit来限制获取的数据量

（3）如果要排序的数据量较大，可以考虑使用Store参数来缓存结果

Redis的基本操作-8

- 处理过期keys的机制

当client主动访问key时，会先对key进行超时判断，过时的key会立刻删除；另外Redis会在后台，每秒10次的执行如下操作： 随机选取100个key校验是否过期，如果有25个以上的key过期了，立刻额外随机选取下100个key(不计算在10次之内)。也就是说，如果过期的key不多，Redis最多每秒回收200条左右，如果有超过25%的key过期了，它就会做得更多，这样即使从不被访问的数据，过期了也会被删除掉。

- 处理过期keys的命令

1：expire：设置过期时间，格式是expire key值 秒数
2：expireat：设置过期时间，格式是expireat key值 到秒的时间戳
3：ttl：查看还有多少秒过期，格式是ttl key值，-1表示永不过期，-2表示已过期
4：persist：设置成永不过期，格式是persist key值，删除key的过期设置；另外使用set或者getset命令为键赋值的时候，也会清除键的过期时间
5：pttl：查看还有多少毫秒过期，格式是pttl key值
6：pexpire：设置过期时间，格式是pexpire key值 毫秒数
7：pexpireat：设置过期时间，格式是pexpireat key值 到毫秒的时间戳

Redis的配置详解-1

- Config命令

  可以在redis-cli里面使用config命令来获取或者设置Redis配置，可以不用重新启动Redis。命令是config get/set 配置名。注意并不是所有的配置参数都可以通过Config来在运行期修改，比如：daemonize、pidfile、port、database、dir、slaveof、rename-command等

- redis.conf的配置介绍

1：开头定义了一些基本的度量单位，只支持bytes，不支持bit

2：配置中对单位的大小写不敏感

3：配置中大致包含如下部分：

（1）通用部分

（2）快照部分：Redis的RDB持久化相关的配置

（3）复制部分

（4）安全部分

（5）限制部分

（6）追加模式部分

（7）LUA脚本部分

（8）慢日志部分

（9）事件通知部分

（10）高级配置部分

Redis的配置详解-2

- 通用部分

1：daemonize：是否以后台daemon方式运行
2：pidfile：pid文件位置，默认会生成在/var/run/redis.pid
3：bind：指定要绑定的IP，默认Redis会响应本机所有可用网卡的连接请求
4：port：监听的端口号，默认服务端口是6379，0表示不监听端口；如果redis不监听端口，可以通过unix socket方式来接收请求，可以通过unixsocket配置项来指定unix socket文件的路径，并通过unixsocketperm来指定文件的权限
5：tcp-backlog：设置tcp的backlog，backlog其实是一个连接队列，backlog队列总和=未完成三次握手队列 + 已经完成三次握手队列。在高并发环境下你需要一个高backlog值来避免慢客户端连接问题。注意Linux内核会将这个值减小到/proc/sys/net/core/somaxconn的值，所以需要确认增大somaxconn和tcp_max_syn_backlog两个值来达到想要的效果
6：timeout：连接空闲超时时间，0表示永不关闭
7：tcp-keepalive：单位为秒，如果设置为0，则不会进行Keepalive检测，建议设置成60
8：loglevel：log信息级别，共分四级，即debug、verbose、notice、warning
9：logfile：log文件位置，如果设置为空字符串，则redis会将日志输出到标准输出。假如你在daemon情况下将日志设置为输出到标准输出，则日志会被写到/dev/null中
10：syslog-enabled：是否把日志输出到syslog中
11：syslog-ident：指定syslog里的日志标志

Redis的配置详解-3

12：syslog-facility：指定syslog设备，值可以是USER或LOCAL0-LOCAL7

13：databases：开启数据库的数量，编号从0开始，默认的数据库是编号为0的数据库，可以使用select <DBid>来选择相应的数据库

- 限制部分

1：maxclients：设置redis同时可以与多少个客户端进行连接。默认情况下为10000个客户端。当你无法设置进程文件句柄限制时，redis会设置为当前的文件句柄限制值减去32，因为redis会为自身内部处理逻辑留一些句柄出来。如果达到了此限制，redis则会拒绝新的连接请求，并且向这些连接请求方发出“max number of clients reached”以作回应。

2：maxmemory：设置redis可以使用的内存量。一旦到达内存使用上限，redis将会试图移除内部数据，移除规则可以通过maxmemory-policy来指定。如果redis无法根据移除规则来移除内存中的数据，或者设置了“不允许移除”，那么redis则会针对那些需要申请内存的指令返回错误信息，比如SET、LPUSH等。但是对于无内存申请的指令，仍然会正常响应，比如GET等。

如果你的redis是主redis（说明你的redis有从redis），那么在设置内存使用上限时，需要在系统中留出一些内存空间给同步队列缓存，只有在你设置的是“不移除”的情况下，才不用考虑这个因素

3：maxmemory-samples：设置样本数量，LRU算法和最小TTL算法都并非是精确的算法，而是估算值，所以你可以设置样本的大小，redis默认会检查这么多个key并选择其中LRU的那个

Redis的配置详解-4

4：maxmemory-policy：设置内存移除规则，redis提供了多达6种的移除规则：

（1）volatile-lru：使用LRU算法移除key，只对设置了过期时间的键

（2）allkeys-lru：使用LRU算法移除key

（3）volatile-random：在过期集合中移除随机的key，只对设置了过期时间的键

（4）allkeys-random：移除随机的key

（5）volatile-ttl：移除那些TTL值最小的key，即那些最近要过期的key

（6）noeviction：不进行移除。针对写操作，只是返回错误信息

无论使用上述哪一种移除规则，如果没有合适的key可以移除的话，redis都会针对写请求返回错误信息

- 快照部分

1：save * *：保存快照的频率，第一个*表示多长时间，单位是秒，第二个*表示至少执行写操作的次数；

在一定时间内至少执行一定数量的写操作时，就自动保存快照；可设置多个条件。

如果想禁用RDB持久化的策略，只要不设置任何save指令，或者给save传入一个空字符串参数也可以

如果用户开启了RDB快照功能，那么在Redis持久化数据到磁盘时如果出现失败，默认情况下，Redis会停止接受所有的写请求。这样做的好处在于可以让用户很明确的知道内存中的数据和磁盘上的数据已经存在不一致了。如果下一次RDB持久化成功，redis会自动恢复接受写请求。

2：stop-writes-on-bgsave-error：如果配置成no，表示你不在乎数据不一致或者有其他的手段发现和控制这种不一致，那么在快照写入失败时，也能确保redis继续接受新的写请求

Redis的配置详解-5

3：rdbcompression：对于存储到磁盘中的快照，可以设置是否进行压缩存储。如果是的话，redis会采用LZF算法进行压缩。如果你不想消耗CPU来进行压缩的话，可以设置为关闭此功能

4：rdbchecksum：在存储快照后，还可以让redis使用CRC64算法来进行数据校验，但是这样做会增加大约10%的性能消耗，如果希望获取到最大的性能提升，可以关闭此功能

5：dbfilename：数据快照文件名（只是文件名，不包括目录），默认dump.rdb

6：dir：数据快照的保存目录（这个是目录），默认是当前路径

- 追加模式部分

1：appendonly：是否开启AOF

2：appendfilename：设置AOF的日志文件名

3：appendfsync：设置AOF日志如何同步到磁盘，fsync()调用，用来告诉操作系统立即将缓存的指令写入磁盘，有三个选项：

（1）always：每次写都强制调用fsync，这种模式下，redis会相对较慢，但数据最安全

（2）everysec：每秒启用一次fsync

（3）no：不调用fsync()。而是让操作系统自行决定sync的时间。这种模式下，redis的性能会最快

4：no-appendfsync-on-rewrite：设置当redis在rewrite的时候，是否允许appendsync。因为redis进程在进行AOF重写的时候，fsync()在主进程中的调用会被阻止，也就是redis的持久化功能暂时失效。默认为no，这样能保证数据安全

5：auto-aof-rewrite-min-size：设置一个最小大小，是为了防止在aof很小时就触发重写

Redis的配置详解-6

5：auto-aof-rewrite-percentage：设置自动进行AOF重写的基准值，也就是重写启动时的AOF文件大小，假如redis自启动至今还没有进行过重写，那么启动时aof文件的大小会被作为基准值。这个基准值会和当前的aof大小进行比较。如果当前aof大小超出所设置的增长比例，则会触发重写。如果设置auto-aof-rewrite-percentage为0，则会关闭此重写功能

- 复制部分

1：slaveof ：指定某一个redis作为另一个redis的从服务器，通过指定IP和端口来设置主redis。建议为从redis设置一个不同频率的快照持久化的周期，或者为从redis配置一个不同的服务端口

2：masterauth：如果主redis设置了验证密码的话（使用requirepass来设置），则在从redis的配置中要使用masterauth来设置校验密码，否则的话，主redis会拒绝从redis的访问请求

3：slave-serve-stale-data：设置当从redis失去了与主redis的连接，或者主从同步正在进行中时，redis该如何处理外部发来的访问请求 。

如果设置为yes（默认），则从redis仍会继续响应客户端的读写请求。如果设置为no，则从redis会对客户端的请求返回“SYNC with master in progress”，当然也有例外，当客户端发来INFO请求和SLAVEOF请求，从redis还是会进行处理。从redis2.6版本之后，默认从redis为只读。

4：slave-read-only：设置从Redis为只读

5：repl-ping-slave-period：设置从redis会向主redis发出PING包的周期，默认是10秒

6：repl-timeout：设置主从同步的超时时间，要确保这个时限比repl-ping-slave-period的值要大，否则每次主redis都会认为从redis超时。

Redis的配置详解-7

7：repl-disable-tcp-nodelay：设置在主从同步时是否禁用TCP_NODELAY，如果开启，那么主redis会使用更少的TCP包和更少的带宽来向从redis传输数据。但是这可能会增加一些同步的延迟，大概会达到40毫秒左右。如果关闭，那么数据同步的延迟时间会降低，但是会消耗更多的带宽。

8：repl-backlog-size：设置同步队列长度。队列长度（backlog)是主redis中的一个缓冲区，在与从redis断开连接期间，主redis会用这个缓冲区来缓存应该发给从redis的数据。这样的话，当从redis重新连接上之后，就不必重新全量同步数据，只需要同步这部分增量数据即可

9：repl-backlog-ttl：设置主redis要等待的时间长度，如果主redis等了这么长时间之后，还是无法连接到从redis，那么缓冲队列中的数据将被清理掉。设置为0，则表示永远不清理。默认是1个小时。

10：slave-priority：设置从redis优先级，在主redis持续工作不正常的情况，优先级高的从redis将会升级为主redis。而编号越小，优先级越高。当优先级被设置为0时，这个从redis将永远也不会被选中。默认的优先级为100。

11：min-slaves-to-write：设置执行写操作所需的最少从服务器数量，如果至少有这么多个从服务器， 并且这些服务器的延迟值都少于 min-slaves-max-lag 秒， 那么主服务器就会执行客户端请求的写操作

12：min-slaves-max-lag：设置最大连接延迟的时间， min-slaves-to-write和min-slaves-max-lag中有一个被置为0，则这个特性将被关闭。默认情况下min-slaves-to-write为0，而min-slaves-max-lag为10

Redis的配置详解-8

- 安全部分

当redis-server处于一个不太可信的网络环境中时，可以要求redis客户端在向redis-server发送请求之前，先进行密码验证

1：requirepass：设置请求的密码

2：rename-command：对命令进行重命名，只读的从redis并不适合直接暴露给不可信的客户端。为了尽量降低风险，可以使用rename-command指令来将一些可能有破坏力的命令重命名，避免外部直接调用。

如果要禁用某些命令，那就重命名为“”就可以了

- Lua脚本部分

1：lua-time-limit：设置ua脚本的最大运行时间，单位是毫秒，如果此值设置为0或负数，则既不会有报错也不会有时间限制

- 慢日志部分

1：slowlog-log-slower-than：判断是否慢日志的执行时长，单位是微秒，负数则会禁用慢日志功能，而0则表示强制记录每一个命令

2：slowlog-max-len：慢日志的长度。当一个新的命令被写入日志时，最老的一条会从命令日志队列中被移除。

Redis的配置详解-9

- 事件通知部分

1：notify-keyspace-events：设置是否开启Pub/Sub 客户端关于键空间发生的事件，有很多通知的事件类型，默认被禁用，因为用户通常不需要该特性，并且该特性会有性能损耗，设置成空字符串就是禁用

- 高级配置部分

1：有关哈希数据结构的一些配置项：当hash只有少量的entry，并且最大的entry所占空间没有超过指定的限制时，会用一种节省内存的数据结构来编码：

（1）hash-max-ziplist-entries：设置使用ziplist的最大的entry数

（2）hash-max-ziplist-value：设置使用ziplist的值的最大长度

2：有关列表数据结构的一些配置项：数据元素较少的list，可以用另一种方式来编码从而节省大量空间：

（1）list-max-ziplist-entries：设置使用ziplist的最大的entry数

（2）list-max-ziplist-value：设置使用ziplist的值的最大长度

3：有关Set数据结构的配置项：当set数据全是十进制64位有符号整型数字构成的字符串时，设置set使用一种紧凑编码格式来节省内存的最大长度：

（1）set-max-intset-entries：设置使用紧凑编码的最大的entry数

Redis的配置详解-10

4：有关有序集合数据结构的配置项：有序集合也可以用一种特别的编码方式来节省大量空间，这种编码只适合长度和元素都小于下面限制的有序集合：

（1）zset-max-ziplist-entries：设置使用ziplist的最大的entry数

（2）zset-max-ziplist-value：设置使用ziplist的值的最大长度

5：HyperLogLog 稀疏表示字节限制：这个限制包含了16个字节的头部，当一个HyperLogLog使用sparse representation，超过了这个限制，它就会转换到dense representation上

（1）hll-sparse-max-bytes：设置HyperLogLog 稀疏表示的最大字节数

6：关于是否启用哈希刷新的配置项：启用哈希刷新，每100个CPU毫秒会拿出1个毫秒来刷新Redis的主哈希表（顶级键值映射表），redis所用的哈希表实现采用延迟哈希刷新机制：对一个哈希表操作越多，哈希刷新操作就越频繁；反之，如果服务器是空闲的，那么哈希刷新就不会完成，哈希表就会占用更多的一些内存而已。默认是每秒钟进行10次哈希表刷新，用来刷新字典，然后尽快释放内存

（1）activerehashing：设置是否启用哈希刷新，默认是yes

7：有关重写aof的配置项：当一个子进程重写AOF文件时，如果启用下面的选项，则文件每生成32M数据会被同步。为了增量式的写入硬盘并且避免大的延迟，这个指令是非常有用的

（1）aof-rewrite-incremental-fsync：默认是yes

Redis的配置详解-11

8：关于客户端输出缓冲的控制项：可用于强制断开那些因为某种原因从服务器读取数据的速度不够快的客户端，一个常见的原因是一个发布/订阅客户端消费消息的速度无法赶上生产它们的速度，可以对三种不同的客户端设置不同的限制：

（1）normal：正常客户端

（2）slave：slave和 MONITOR 客户端

（3）pubsub：至少订阅了一个pubsub channel或pattern的客户端

语法是：client-output-buffer-limit <class><hard limit> <soft limit> <soft 持续时间单位秒>

默认normal客户端不做限制，因为他们在不主动请求时不接收数据；pubsub和slave客户端会有一个默认值；把硬限制和软限制都设置为0来禁用该功能，默认配置如下：

（1）client-output-buffer-limit normal 0 0 0

（2）client-output-buffer-limit slave 256mb 64mb 60

（3）client-output-buffer-limit pubsub 32mb 8mb 60

9：有关频率的配置项：Redis调用内部函数来执行许多后台任务，Redis依照指定的“hz”值来执行检查任务，默认情况下，“hz”的被设定为10。提高该值将在Redis空闲时使用更多的CPU时，但同时当有多个key同时到期会使Redis的反应更灵敏，以及超时可以更精确地处理。范围是1到500之间，但是值超过100通常不是一个好主意。大多数用户应该使用10这个默认值，只有在非常低的延迟要求时有必要提高到100

（1）hz：设置检查任务的频率

Redis的持久化-1

- Redis持久化概述

  Redis持久化分成两种方式：RDB（Redis DataBase）和AOF（Append Only File）

1：RDB是在不同的时间点，将Redis某一时刻的数据生成快照并存储到磁盘上

2：AOF是只允许追加不允许改写的文件，是将Redis执行过的所有写指令记录下来，在下次Redis重新启动时，只要把这些写指令从前到后再重复执行一遍，就可以实现数据恢复了

3：RDB和AOF两种方式可以同时使用，在这种情况下，如果Redis重启的话，则会优先采用AOF方式来进行数据恢复，这是因为AOF方式的数据恢复完整度更高

4：可以关闭RDB和AOF，这样的话，Redis将变成一个纯内存数据库，就像Memcache一样

5：通过配置redis.conf中的appendonly为yes就可以打开AOF功能

- RDB

  RDB方式，Redis会单独创建（fork）一个子进程来进行持久化，会先将数据写入到一个临时文件中，待持久化过程都结束了，再用这个临时文件替换上次持久化好的文件。整个过程中，主进程是不进行任何IO操作的，这就确保了极高的性能

  如果需要进行大规模数据的恢复，且对于数据恢复的完整性不是非常敏感，那RDB方式要比AOF方式更加的高效。RDB的缺点是最后一次持久化后的数据可能丢失。

Redis的持久化-2

- RDB的问题

1：fork一个进程时，内存的数据也被复制了，即内存会是原来的两倍

2：每次快照持久化都是将内存数据完整写入到磁盘一次，并不是增量的只同步脏数据。如果数据量大的话，而且写操作比较多，必然会引起大量的磁盘io操作，可能会严重影响性能。

3：由于快照方式是在一定间隔时间做一次的，所以如果redis意外down掉的话，就会丢失最后一次快照后的所有修改。

- 触发快照的情况

1：根据配置规则进行自动快照

2：用户执行save或bgsave命令

3：执行flushall命令

4：执行复制replication时

- save命令

执行Save命令时，Redis会阻塞所有客户端的请求，然后同步进行快照操作。

Redis的持久化-3

- bgsave命令

  执行bgsave命令时，Redis会在后台异步进行快照操作，快照同时还可以响应客户端请求。可以通过lastsave命令获取最后一次成功执行快照的时间。

- flushall命令

  这个命令会导致Redis清除内存中的所有数据，如果定义了自动快照的条件，那么无论是否满足条件，都会进行一次快照操作；如果没有定义自动快照的条件，那么不会进行快照

- AOF

  默认的AOF持久化策略是每秒钟fsync一次，fsync是指把缓存中的写指令记录到磁盘中，在这种情况下，Redis仍可以保持很高的性能。

  当然由于OS会在内核中缓存 write做的修改，所以可能不是立即写到磁盘上。这样aof方式的持久化也还是有可能会丢失部分修改。不过可以通过配置文件告诉Redis，想要通过fsync函数强制os写入到磁盘的时机。

  AOF方式在同等数据规模的情况下，AOF文件要比RDB文件的体积大，因此AOF方式的恢复速度也要慢于RDB方式。

Redis的持久化-4

- AOF日志恢复

如果在追加日志时，恰好遇到磁盘空间满或断电等情况，导致日志写入不完整，也没有关系，Redis提供了redis-check-aof工具，可以用来进行日志修复，基本步骤如下：

1：备份被写坏的AOF文件
2：运行redis-check-aof –fix进行修复
3：用diff -u来看下两个文件的差异，确认问题点
4：重启redis，加载修复后的AOF文件

- AOF重写

AOF采用文件追加方式，这会导致AOF文件越来越大，为此，Redis提供了AOF文件重写（rewrite）机制，即当AOF文件的大小超过所设定的阈值时，Redis就会启动AOF文件的内容压缩，只保留可以恢复数据的最小指令集。可以使用命令bgrewriteaof 。

Redis的持久化-5

- AOF重写的触发机制

  Redis是这样工作的：Redis会记录上次重写时的AOF大小。假如自启动至今还没有进行过重写，那么启动时AOF文件的大小会被作为基准值，这个基准值会和当前的AOF大小进行比较，如果当前AOF大小超出所设置的增长比例，则会触发重写。另外，你还需要设置一个最小大小，是为了防止在AOF很小时就触发重写

- AOF重写的基本原理

1：在重写开始前，redis会创建一个“重写子进程”，这个子进程会读取现有的AOF文件，并将其包含的指令进行分析压缩并写入到一个临时文件中。

2：与此同时，主进程会将新接收到的写指令一边累积到内存缓冲区中，一边继续写入到原有的AOF文件中，这样做是保证原有的AOF文件的可用性，避免在重写过程中出现意外。

3：当“重写子进程”完成重写工作后，它会给父进程发一个信号，父进程收到信号后就会将内存中缓存的写指令追加到新AOF文件中

4：当追加结束后，redis就会用新AOF文件来代替旧AOF文件，之后再有新的写指令，就都会追加到新的AOF文件中

5：重写aof文件的操作，并没有读取旧的aof文件，而是将整个内存中的数据库内容用命令的方式重写了一个新的aof文件，这点和快照有点类似

Redis的事务-1

- 概述

Redis中的事务就是一组命令的集合，被依次顺序的执行，当然你可以放弃事务的执行，那么所有事务里面的命令都不会执行。

关于Redis的事务有几点说明：

1：Redis的事务仅仅是保证事务里的操作会被连续独占的执行，因为是单线程架构，在执行完事务内所有指令前是不可能再去同时执行其他客户端的请求的

2：Redis的事务没有隔离级别的概念，因为事务提交前任何指令都不会被实际执行，也就不存在”事务内的查询要看到事务里的更新，在事务外查询不能看到”这种问题了

3：Redis的事务不保证原子性，也就是不保证所有指令同时成功或同时失败，只有决定是否开始执行全部指令的能力，没有执行到一半进行回滚的能力

- Redis事务的基本过程

1：发送一个事务的命令给Redis，命令是multi

2：依次发送要执行的命令给Redis，Redis接到这些命令，并不会立即执行，而是放到等待执行的事务队列里面

3：发送执行事务的命令给Redis，命令是exec

4：Redis会保证一个事务内的命令依次执行，而不会被其它命令插入

Redis的事务-2

- 事务过程中的错误处理

1：如果任何一个命令语法有错，Redis会直接返回错误，所有的命令都不会执行
2：如果某个命令执行错误，那么其它的命令仍然会正常执行，然后在执行后返回错误信息
3：Redis不提供事务回滚的功能，开发者必须在事务执行出错后，自行恢复数据库状态

- 事务操作的基本命令

1：multi：设置事务开始
2：exec：执行事务
3：discard：放弃事务
4：watch：监控键值，如果键值被修改或删除，后面的一个事务就不会执行
5：unwatch：取消watch

- Watch说明

1：Redis使用Watch来提供乐观锁定，类似于CAS(Check-and-Set)
2：WATCH 可以被调用多次
3：当 EXEC 被调用后，所有的之前被监视的键值会被取消监视，不管事务是否被取消或者执行。并且当客户端连接丢失的时候，所有东西都会被取消监视

Redis的发布订阅模式

- 介绍

  Redis的发布订阅模式可以实现进程间的消息传递

- 发布订阅模式的操作命令

1：publish：发布消息，格式是publish channel 消息

2：subscribe：订阅频道，格式是subscribe channel，可以是多个channel

3：psubscribe：订阅频道，格式是psubscribe channel，支持glob风格的通配符

3：unsubscribe：取消订阅，格式是unsubscribe channel，不指定频道表示取消所有subscribe命令的订阅

4：punsubscribe：取消订阅，格式是punsubscribe channel，不指定频道表示取消所有psubscribe命令的订阅，注意这里匹配模式的时候，是不会将通配符展开的，是严格进行字符串匹配的，比如： punsubscribe * 是无法退定 c1.*的，必须严格使用punsubscribe c1.*才可以

Redis的复制-1

- 复制

Redis支持复制的功能，以实现当一台服务器的数据更新后，自动将新的数据同步到其它数据库。

Redis复制实现中，把数据库分为主数据库master和从数据库slave，主数据库可以进行读写操作，从数据库一般是只读的，当主数据库数据变化的时候，会自动同步给从数据库。

- 复制带来的好处

1：可以实现读写分离

2：利于在主数据库崩溃时的数据恢复

- 复制的配置

主数据库不做配置；从数据库需要在配置中设置“slaveof 主数据库ip 主数据库端口”。

- 复制的基本操作命令

1：info replication ：可以查看复制节点的相关信息

2：slaveof：可在运行期间修改slave节点的信息，如果该数据库已经是某个主数据库的从数据库，那么会停止和原主数据库的同步关系，转而和新的主数据库同步

3：slaveof no one：使当前数据库停止与其他数据库的同步，转成主数据库

Redis的复制-2

- 复制的基本原理

1：slave启动时，会向master发送sync命令，2.8版后发送psync，以实现增量复制

2：主数据库接到sync请求后，会在后台保存快照，也就是实现RDB持久化，并将保存快照期间接收到的命令缓存起来

3：快照完成后，主数据库会将快照文件和所有缓存的命令发送给从数据库

4：从数据库接收后，会载入快照文件并执行缓存的命令，从而完成复制的初始化

5：在数据库使用阶段，主数据库会自动把每次收到的写命令同步到从服务器

- 乐观复制策略

Redis采用乐观复制的策略，容忍在一定时间内主从数据库的内容不同，当然最终的数据会是一样的。这个策略保证了性能，在复制的时候，主数据库并不阻塞，照样处理客户端的请求。

Redis提供了配置来限制只有当数据库至少同步给指定数量的从数据库时，主数据库才可写，否则返回错误。配置是：min-slaves-to-write、min-slaves-max-lag

Redis的复制-3

- 无硬盘复制

当复制发生时，主数据库会在后台保存RDB快照，即使你关闭了RDB，它也会这么做，这样就会导致：

1：如果主数据库关闭了RDB，现在强行生成了RDB，那么下次主数据库启动的时候，可能会从RDB来恢复数据，这可能是旧的数据。

2：由于要生成RDB文件，如果硬盘性能不高的时候，会对性能造成一定影响

因此从2.8.18版本，引入了无硬盘复制选项：repl-diskless-sync

- 哨兵（sentinel）

Redis提供了哨兵工具来实现监控Redis系统的运行情况，主要实现：

1：监控主从数据库运行是否正常

2：当主数据库出现故障时，自动将从数据库转换成为主数据库

3：使用Redis-sentinel，redis实例必须在非集群模式下运行

- 开启哨兵功能

建立一个sentinel.conf文件，里面设置要监控的主数据库的名字，形如：

sentinel monitor 监控的主数据库的名字 127.0.0.1 6379 1

1 表示选举主数据库的最低票数

（1）这个文件的内容，在运行期间会被sentinel动态进行更改

（2）可以同时监控多个主数据库，一行一个配置即可

Redis的集群-1

- 复制的问题

由于复制中，每个数据库都是拥有完整的数据，因此复制的总数据存储量，受限于内存最小的数据库节点，如果数据量过大，复制就无能为力了。

- 分片

分片（Partitioning）就是将你的数据拆分到多个Redis实例的过程，这样每个Redis实例将只包含完整数据的一部分。常见的分片方式：

1：按照范围分片

2：哈希分片，比如一致性哈希

- 常见的分片实现：

1：在客户端进行分片

2：通过代理来进行分片，比如：Twemproxy

3：查询路由：就是发送查询到一个随机实例，这个实例会保证转发你的查询到正确的节点，Redis集群在客户端的帮助下，实现了查询路由的一种混合形式，请求不是直接从Redis实例转发到另一个，而是客户端收到重定向到正确的节点

4：在服务器端进行分片， Redis采用哈希槽（hash slot）的方式在服务器端进行分片：Redis集群有16384个哈希槽，使用键的CRC16编码对16384取模来计算一个键所属的哈希槽

Redis的集群-2

- Redis分片的缺点

1：不支持涉及多键的操作，如mget，如果所操作的键都在同一个节点，就正常执行，否则会提示错误
2：分片的粒度是键，因此每个键对应的值不要太大
3：数据备份会比较麻烦，备份数据时你需要聚合多个实例和主机的持久化文件
4：扩容的处理比较麻烦
5：故障恢复的处理会比较麻烦，可能需要重新梳理Master和Slave的关系，并调整每个复制集里面的数据

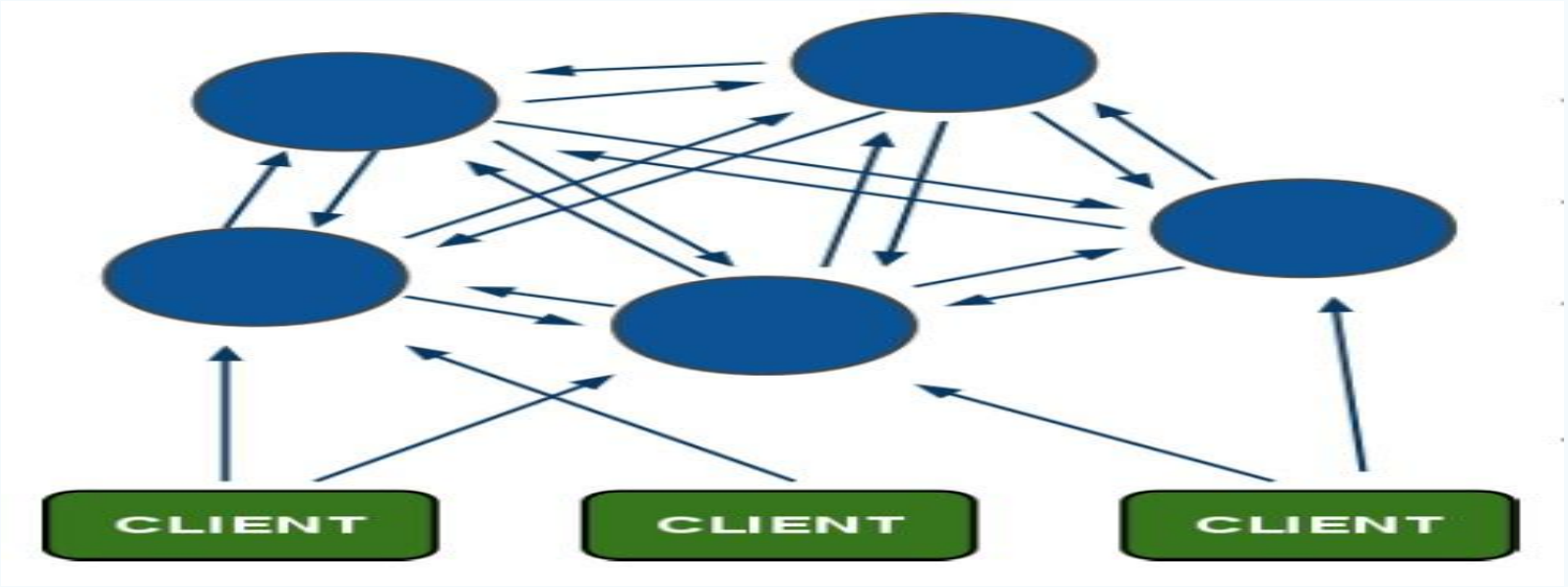
- 集群

由于数据量过大，单个复制集难以承担，因此需要对多个复制集进行集群，形成水平扩展，每个复制集只负责存储整个数据集的一部分，这就是Redis的集群。

1：在以前版本中，Redis的集群是依靠客户端分片来完成，但是这会有很多缺点，比如维护成本高，需要客户端编码解决；增加、移出节点都比较繁琐等
2：Redis3.0新增的一大特性就是支持集群，在不降低性能的情况下，还提供了网络分区后的可访问性和支持对主数据库故障的恢复。
3：使用集群后，都只能使用默认的0号数据库
4：每个Redis集群节点需要两个TCP连接打开，正常的TCP端口用来服务客户端，例如6379，加10000的端口用作数据端口，必须保证防火墙打开这两个端口
5：Redis集群不保证强一致性，这意味着在特定的条件下，Redis集群可能会丢掉一些被系统收到的写入请求命令。

Redis的集群-3

- 集群架构



1：所有的Redis节点彼此互联，内部使用二进制协议优化传输速度和带宽
2：节点的fail是通过集群中超过半数的节点检测失效时才生效
3：客户端与Redis节点直连，不需要中间proxy层。客户端不需要连接集群所有节点，连接集群中任何一个可用节点即可
4：集群把所有的物理节点映射到[0-16383]插槽上，集群负责维护：节点-插槽-值 的关系

Redis的集群-4

- 集群操作基本命令

1：CLUSTER INFO：获取集群的信息
2：CLUSTER NODES：获取集群当前已知的所有节点，以及这些节点的相关信息
3：CLUSTER MEET <ip> <port>：将ip和port所指定的节点添加到集群当中
4：CLUSTER FORGET <node_id>：从集群中移除 node_id 指定的节点
5：CLUSTER REPLICATE <node_id>：将当前节点设置为 node_id 指定的节点的从节点
6：CLUSTER SAVECONFIG：将节点的配置文件保存到硬盘里面
7：CLUSTER ADDSLOTS <slot> [slot ...]：将一个或多个槽分配给当前节点
8：CLUSTER DELSLOTS <slot> [slot ...]：从当前节点移除一个或多个槽
9：CLUSTER FLUSHSLOTS：移除分配给当前节点的所有槽
10：CLUSTER SETSLOT <slot> NODE <node_id>：将槽分配给 node_id 指定的节点，如果槽已经分配给另一个节点，那么先让另一个节点删除该槽>，然后再进行分配
11：CLUSTER SETSLOT <slot> MIGRATING <node_id>：将本节点的槽迁移到指定的节点中
12：CLUSTER SETSLOT <slot> IMPORTING <node_id>：从指定节点导入槽到本节点
13：CLUSTER SETSLOT <slot> STABLE：取消对槽的导入（import）或迁移（migrate）
14：CLUSTER KEYSLOT <key>：计算键 key 应该被放置在哪个槽
15：CLUSTER COUNTKEYSINSLOT <slot>：返回槽目前包含的键值对数量
16：CLUSTER GETKEYSINSLOT <slot> <count>：返回 count 个槽中的键
17：migrate 目的节点ip 目的节点port 键名 数据库号码 超时时间 [copy] [replace]：迁移某个键值对

Redis的集群-5

- 手工创建集群

1：首先进行集群配置

只需要将每个数据库的cluster-enabled配置选项打开，然后再修改如下内容：pidfile、port、logfile、dbfilename、cluster-config-file

2：分别启动这些Redis数据库，可以用info cluster查看信息

3：连接节点，使用cluster meet，把所有的数据库都放到一个集群中来

4：可以通过cluster info ，或者cluster nodes 查看信息

5：设置部分数据库为slave，使用cluster replicate

6：然后就该来分配插槽了，使用cluster addSlots，这个命令目前只能一个一个加，如果要加区间的话，就得客户端编写代码来循环添加。

有个实用的技巧：把所有的Redis停下来，然后直接修改node的配置文件，只需要配置master的数据库就可以，然后再重启数据库。

分配完插槽，可以使用cluster slots查看信息。

7：通过cluster info查看集群信息，如果显示ok，那就可以使用了

Redis的集群-6

- 什么是插槽

插槽是Redis对Key进行分片的单元。在Redis的集群实现中，内置了数据自动分片机制，集群内部会将所有的key映射到16384个插槽中，集群中的每个数据库实例负责其中部分的插槽的读写。

- 键与插槽的关系

Redis会将key的有效部分，使用CRC16算法计算出散列值，然后对16384取余数，从而把key分配到插槽中。键名的有效部分规则是：

1：如果键名包含{}，那么有效部分就是{}中的值
2：否则就是取整个键名

- 移动已分配的插槽

这个稍微麻烦点，尤其是有了数据过后，假设要迁移123号插槽从A到B，大致步骤如下：

1：在B上执行cluster setslot 123 importing A
2：在A上执行cluster setslot 123 migrating B
3：在A上执行cluster getkeysinslot 123 要返回的数量
4：对上一步获取的每个键执行migrate命令，将其从A迁移到B
5：在集群中每个服务器上执行cluster setslot 123 node B

Redis的集群-7

- 避免在移动已分配插槽过程中，键的临时丢失

上面迁移方案中的前两步就是用来避免在移动已分配插槽过程中，键的临时丢失问题的，大致思路如下：

1：当前两步执行完成后，如果客户端向A请求插槽123中的键时，如果键还未被转移，A将处理请求

2：如果键已经转移，则返回，把新的地址告诉客户端，客户端将发起新的请求以获取数据

- 获取插槽对应的节点

当客户端向某个数据库发起请求时，如果键不在这个数据库里面，将会返回一个move重定向的请求，里面包含新的地址，客户端收到这个信息后，需要重新发起请求到新的地址去获取数据。

当然，大部分的Redis客户端都会自动去重定向，也就是这个过程对开发人员是透明的。

redis-cli也支持自动重定向，只需要在启动时加入 -c 的参数。

Redis的集群-8

- 故障判定

1：集群中每个节点都会定期向其他节点发出ping命令，如果没有收到回复，就认为该节点为疑似下线，然后在集群中传播该信息

2：当集群中的某个节点，收到半数以上认为某节点已下线的信息，就会真的标记该节点为已下线，并在集群中传播该信息

3：如果已下线的节点是master节点，那就意味着一部分插槽无法写入了

4：如果集群任意master挂掉，且当前master没有slave，集群进入fail状态

5：如果集群超过半数以上master挂掉，无论是否有slave，集群进入fail状态

6：当集群不可用时，所有对集群的操作做都不可用，收到CLUSTERDOWN The cluster is down错误信息

Redis的集群-9

- 故障恢复

发现某个master下线后，集群会进行故障恢复操作，来将一个slave变成master，基于Raft算法，大致步骤如下：

1：某个slave向集群中每个节点发送请求，要求选举自己为master
2：如果收到请求的节点没有选举过其他slave，会同意
3：当集群中有超过节点数一半的节点同意该slave的请求，则该Slave选举成功
4：如果有多个slave同时参选，可能会出现没有任何slave当选的情况，将会等待一个随机时间，再次发出选举请求
5：选举成功后，slave会通过 slaveof no one命令把自己变成master
如果故障后还想集群继续工作，可设置cluster-require-full-coverage为no，默认yes

- 对于集群故障恢复的说明

1：master挂掉了，重启还可以加入集群，但挂掉的slave重启，如果对应的master变化了，是不能加入集群的，除非修改它们的配置文件，将其master指向新master
2：只要主从关系建立，就会触发主和该从采用save方式持久化数据，不论你是否禁止save
3：在集群中，如果默认主从关系的主挂了并立即重启，如果主没有做持久化，数据会完全丢失，从而从的数据也被清空

Redis的集群-10

- 使用redis-trib.rb来操作集群

  redis-trib.rb是Redis源码中提供的一个辅助工具，可以非常方便的来操作集群，它是用ruby写的，因此需要在服务器上安装相应环境

1：安装Ruby

（1）下载ruby安装包，地址https://www.ruby-lang.org/en/downloads/

（2）然后分别configure、make、make install

（3）安装后通过ruby -v 查看一下版本，看是否正常

2：还需要安装rubygems

（1）下载包，地址https://rubygems.org/pages/download

（2）解压后进入解压文件夹，运行 ruby setup.rb

（3）安装后通过gem -v查看一下版本，看是否正常

3：还需要安装redis的ruby library

（1）由于连接国外源不太稳定，请先删除，如gem sources --remove https://rubygems.org/ ，然后添加gem sources -a https://ruby.taobao.org/

（2）可以通过gem sources -l 查看源

（3）运行gem install redis

Redis的集群-11

4：使用redis-trib.rb来初始化集群，形如：

ruby redis-trib.rb create --replicas 1 127.0.0.1:6381 127.0.0.1:6382 127.0.0.1:6383 127.0.0.1:6384 127.0.0.1:6385 127.0.0.1:6386

create表示要初始化集群，--replicas 1表示每个驻数据库拥有的从数据库为1个

5：使用redis-trib.rb来迁移插槽，如下：

（1）执行ruby redis-trib.rb reshard ip:port ，这就告诉Redis要重新分片，ip:port可以是集群中任何一个节点

（2）然后按照提示去做就可以了

（3）这种方式不能指定要迁移的插槽号

Redis预分区

- 介绍

为了实现在线动态扩容和数据分区，Redis的作者提出了预分区的方案，实际就是在同一台机器上部署多个Redis实例，当容量不够时将多个实例拆分到不同的机器上，这样就达到了扩容的效果。拆分过程如下：

1：在新机器上启动好对应端口的Redis实例
2：配置新端口为待迁移端口的从库
3：待复制完成，与主库完成同步后，切换所有客户端配置到新的从库的端口
4：配置从库为新的主库
5：移除老的端口实例
6：重复上述过程把要迁移的数据库转移到指定服务器上

以上拆分流程是Redis作者提出的一个平滑迁移的过程，不过该拆分方法还是很依赖Redis本身的复制功能的，如果主库快照数据文件过大，这个复制的过程也会很久，同时会给主库带来压力。

Lua脚本开发-1

- Lua介绍

Lua是一个高效、简洁、轻量级、可扩展的脚本语言，可以很方便的嵌入到其它语言中使用，Redis从2.6版支持Lua。

- 使用脚本的好处

1：减少网络开销
2：原子操作：Redis会把脚本当作一个整体来执行，中间不会插入其它命令
3：复用功能

- Lua的数据类型

Lua是一个动态类型的语言，一个变量可以存储任何类型的值，类型有：

1：空：nil，也就是还没有赋值
2：字符串：用单引号 或者 双引号 引起来
3：数字：包含整数和浮点型
4：布尔：boolean
5：表：表是Lua唯一的数据结构，既可以当数组，也可以做Map，或被视为对象
6：函数：封装某个或某些功能
7：userData：用来将任意 C 数据保存在 Lua 变量中，这样的操作只能通过 C API
8：Thread：用来区别独立的执行线程，它被用来实现 coroutine （协同例程）

Lua脚本开发-2

- 变量

Lua的变量分成全局变量和局部变量。

1：全局变量

全局变量无需声明即可直接使用，默认值是nil。在Redis脚本中不允许使用全局变量，以防止脚本之间相互影响。

2：局部变量，声明方法为：local 变量名

3：变量名必须是非数字开头，只能包含字母、数字和下划线，不能是保留关键字，如：

```
and break do else elseif end false for function if in local nil
not or repeat return then true until while
```

4：Lua的变量名是区分大小写的

5：局部变量的作用域为从声明开始到所在层的语句块结尾

- 注释

1：单行：--

2：多行：--[[开始，到]] 结束

- 赋值

Lua支持多重赋值，如：local a,b = 1,2,3

Lua脚本开发-3

- 操作符

1：数学操作符：+、-、*、/、%、- 取反、^ 幂运算；如果操作数是字符串，会自动转换成数字进行操作

2：比较操作符：==、~=、〉、>=、<、<=；比较操作符不会转换类型，如果类型不同进行比较，会返回false；可以手动使用tonumber或者tostring进行转换

3：逻辑操作符：and、or、not

4：连接操作符：..；用来连接两个字符串

5：取长度操作符：#，例如：print(#’helloworld’)

6：操作符的优先级跟其它编程语言是类似的

- If语句

1：格式是：

```
if 条件 then
elseif 条件 then
else
end
```

2：注意：在Lua中，只有nil和false才是假，其它类型的值均被认为是真

Lua脚本开发-4

- 循环语句

Lua支持for、while和repeat三种循环语句。

1：for语句格式是：

```
for 变量=初值，终值，步长 do
end
```

步长可以省略，默认是1

2：增强for循环的格式是：

```
for 变量1，变量2…，变量N in 迭带器 do
end
```

3：while语句的格式是：

```
while 条件 do
end
```

4：repeat语句的格式是：

```
repeat
until条件
```

5：使用break来跳出循环块

Lua脚本开发-5

- 表类型

可以当作数组或者Map来理解，比如：

1：a = {}，报一个空表赋值给a

2：a[key]=value，把value赋值给表a中的字段key

3：a={ key1= ‘value1’， key2= ‘value2’ }

4：引用的时候，可以使用 . 操作符，如： a.key1

5：如果用索引来引用，跟数组是一样的，如：a[1]，注意Lua的索引是从1开始

6：可以使用增强for循环来遍历数组，如：

```
for k, v in ipairs(a) do
        print(k)
        print(v)
end
```

其中的ipairs是Lua的内置函数，实现类似迭带器的功能，从索引1开始递增遍历到最后一个不为nil的整数索引。类似的还有一个pairs，用来便利非数组的表值，它会遍历所有值不为nil的索引。

7：也可以使用for循环来按照索引遍历数组，如：

```
for i=1,#a do
end
```

Lua脚本开发-6

- 函数

1：定义格式为：

```
function(参数列表)
end
```

2：注意：就算没有参数，括号也不能省略

3：形参实参个数不用完全对应，如果想要得到所有的实参，可以把最后一个形参设置成…

4：函数内返回使用return

- Lua的标准库

Lua的标准库提供了很多使用的功能，Redis支持其中大部分，主要有：

1：Base：提供一些基础函数

2：String：提供用于操作字符串的函数

3：Table：提供用于表操作的函数

4：Math：提供数据计算的函数

5：Debug：提供用于调试的函数

Lua脚本开发-7

- 在Redis中常用的标准库函数

1：string.len(string)

2：string.lower(string)

3：string.upper(string)

4：string.rep(s, n)：返回重复s字符串n次的字符串

5：string.sub(string,start[,end])，索引从1开始，-1表示最后一个

6：string.char(n…)：把数字转换成字符

7：string.byte (s [, i [, j]])：用于把字符串转换成数字

8：string.find (s, pattern [, init [, plain]])：查找目标模板在给定字符串中出现的位置，找到返回起始和结束位置，没找到返回nil

9：string.gsub (s, pattern, repl [, n])：将所有符合匹配模式的地方都替换成替代字符串。并返回替换后的字符串，以及替换次数。四个参数，给定字符串，匹配模式、替代字符串和要替换的次数

10：string.match (s, pattern [, init])：将返回第一个出现在给定字符串中的匹配字符串，基本的模式有：. 所有字符，%a字母，%c控制字符，%d数字，%l小写字母，%p 标点符号字符，%s 空格，%u 大写字母，%w 文字数字字符，%x 16进制数字等

Lua脚本开发-8

11：string.reverse (s)：逆序输出字符串

12：string.gmatch (s, pattern)：返回一个迭代器，用于迭代所有出现在给定字符串中的匹配字符串

13：table.concat(table[,sep[,i[,j]]])：将数组转换成字符串，以sep指定的字符串分割，默认是空，i和j用来限制要转换的表索引的范围，默认是1和表的长度，不支持负索引

14：table.insert(table,[pos,]value)：向数组中插入元素，pos为指定插入的索引，默认是数组长度加1，会将索引后面的元素顺序后移

15：table.remove(table[,pos])：从数组中弹出一个元素，也就是删除这个元素，将后面的元素前移，返回删除的元素值，默认pos是数组的长度

table.sort(table[,sortFunction])：对数组进行排序，可以自定义排序函数

16：Math库里面常见的：abs、ceil、floor、max、min、pow、sqrt、sin、cos、tan等

17：math.random([m[,n]])：获取随机数，如果是同一个种子的话，每次获得的随机数是一样的，没有参数，返回0-1的小数；只有m，返回1-m的整数；设置了m和n，返回m-n的整数

18：math.randomseed(x)：设置生成随机数的种子

Lua脚本开发-9

- 其它库

  除了标准库外，Redis还会自动加载cjson和cmsgpack库，以提供对Json和MessagePack的支持，在脚本中分别通过cjson和cmsgpack两个全局变量来访问相应功能

1：cjson.encode(表)：把表序列化成字符串

2：cjson.decode(string)：把字符串还原成为表

3：cmsgpack.pack(表)：把表序列化成字符串

4：cmsgpack.unpack(字符串)：把字符串还原成为表

- Redis和Lua结合

1：redis.call：在脚本中调用Redis命令，遇到错误会直接返回

2：redis.pcall：在脚本中调用Redis命令，遇到错误会记录错误并继续执行

3：Lua数据类型和Redis返回值类型对应

（1）数字——整数

（2）字符串——字符串

（3）表类型——多行字符串

（4）表类型（只有一个ok字段存储状态信息）——状态回复

（5）表类型（只有一个err字段存储错误信息）——错误回复

Lua脚本开发-10

4：eval命令：在Redis中执行脚本

（1）格式是：eval 脚本内容 key参数数量 [key…] [arg…]

（2）通过key和arg两类参数来向脚本传递数据，在脚本中分别用KEYS和ARGV来获取

注意：

对于KEYS和ARGV的使用并不是强制的，也可以不从KEYS去获取键，而是在脚本中硬编码，比如：redis.call(‘get’ ,’ user:’ ..ARGV[1]) 0 key1 ，照样能取到”user:key1”对应的值。

但是这种写法，就无法兼容集群，也就是说不能在集群中使用。要兼容集群，建议的方式是在客户端获取所有的key，然后通过KEYS传到脚本中。

5：evalsha命令：可以通过脚本摘要来运行，其他同eval。执行的时候会根据摘要去找缓存的脚本，找到了就执行，否则会返回错误。

6：script load：将脚本加入缓存，返回值就是SHA1摘要

7：script exists：判断脚本是否已经缓存

8：script flush：清空脚本缓存

9：script kill：强制终止脚本的执行，如果脚本中修改了某些数据，那么不会终止脚本的执行，以保证脚本执行的原子性

Lua脚本开发-11

- 沙箱

为了保证Redis服务器的安全，并且要确保脚本的执行结果只和脚本执行时传递的参数有关，Redis禁止脚本中使用操作文件或系统调用相关的函数，脚本中只能对Redis数据进行操作，这就是沙箱。

Redis会禁用脚本的全局变量，以保证脚本之间是隔离的，互不相干的。

- Redis对随机数和随机结果的处理

1：为了确保执行结果可以重现，Redis对随机数的功能进行了处理，以保证每次执行脚本生成的随机数列都相同

2：Redis还对产生随机结果进行了处理，比如smembers或hkeys等，数据都是无序的，Redis会对结果按照字典进行顺序排序

3：对于会产生随机结果但无法排序的命令，比如指挥产生一个元素，Redis会在这类命令执行后，把该脚本标记为lua_random_dirty，此后只允许调用读命令，不许修改，否则返回错误，这类Redis命令有：spop、srandmember、randomkey、time。

Lua脚本开发-12

- MetaTable

用来实现重载操作符功能，基本示例如下：

1：自定义操作的函数，示例：

```
myAdd={}
function myAdd.__add(f1,f2)
        --具体的操作
end
```

2：为已有的table设置自定义的操作模板，示例：

```
setmetatable(tableA,myAdd)
setmetatable(tableB,myAdd)
```

3：对两个table做加的操作，示例：

```
tableA+tableB
```

这个时候就会调用自定义的myAdd了，等于重载了默认的_add方法，myAdd的__add方法就是MetaMethod

4：Lua内建约定的MetaMethod ：

__add(a, b) 、__sub(a, b)、__mul(a, b)、__div(a, b)、__mod(a, b)、__pow(a, b) 、__unm(a) 取反、__concat(a, b)、__len(a)、__eq(a, b)、__lt(a, b)、__le(a, b)、__index(a, b) 对应表达式 a.b、__newindex(a, b, c) 对应表达式 a.b = c、__call(a, ...)

Lua脚本开发-13

- 面向对象

Lua脚本的面向对象类似于JavaScript的面向对象，都是模拟的，比如：

1：直接创建对象：local user={userId='user1',userName='sishuok'}

2：添加新属性：user.age = 12

3：添加方法

```
function user:show(a)
  redis.log(redis.LOG_NOTICE,'a='..a..',age='..self['age'])
end
```

里面的self就相当于this

4：就可以调用方法了：user:show('abc')

5：做个子类来继承user：

```
local child={address='bj'}
setmetatable(child,{__index=user})
```

__index在这里起的作用就类似于JS中的Prototype

6：继承了自然就可以调用父类的属性和方法了：child:show('child')

7：当然你还可以定义自己的方法去覆盖父类的方法：

```
function child:show(a)
   redis.log(redis.LOG_NOTICE,'child='..a..',age='..self['age']..',address=='..self.address)
end
```

Lua脚本开发-14

- 模块化

注意：这种方式不能在Redis中使用，目前不支持

1：可以直接使用require(“model_name”)来载入别的lua文件，文件的后缀是.lua。载入的时候就会直接执行那个文件

2：载入同样的lua文件时，只有第一次的时候会去执行，后面的相同的都不执行了

3：如果要让每一次文件都执行，可使用dofile(“model_name”)函数

4：如果要载入后不执行，等需要的时候执行，可使用 loadfile(“model_name”)函数，这种是把loadfile的结果赋值给一个变量，比如：

```
local abc = loadfile(“载入的lua”)
```

后面需要运行时： abc()

Redis的安全

- 简述

  Redis在安全部分并没有做太多的工作，毕竟Redis是按照“Redis是运行在可信环境”这个假定来设计的。

- 安全相关的配置

1：bind：可以绑定允许访问数据库的地址，只能绑定一个地址

2：requirepass：设置数据库密码，如果设置了，那么客户端每次连接Redis的时候，都需要传入密码，形如：auth 密码，然后才能执行命令。
如果是复制集，就需要配置masterauth参数为主数据库的密码

- 安全相关的命令

1：rename-command：命令重命名

Redis的管理

- 常用的管理命令

1：slowlog get：获取慢日志，可以通过配置文件的slowlog-log-slower-than来设置时间限制，默认是10000微秒，slowlog-max-len来限制记录条数。

返回的记录包含四个部分：

（1）日志的id

（2）该命令执行的unix时间

（3）该命令消耗的时间，单位微秒

（4）命令和参数

2：monitor：监控Redis执行的所有命令，这个命令比较耗性能，建议仅用在开发调试阶段

- 常用的管理工具

1：phpRedisAdmin：地址https://github.com/ErikDubbelboer/phpRedisAdmin

2：rdbtools：地址https://github.com/sripathikrishnan/redis-rdb-tools ，这个是用python写的，可以提供生成内存报告、 转储文件到JSON、使用标准的diff工具比较两个dump文件等功能

3：Cacti：用来监控Redis服务的流量

Redis的虚拟内存-1

- 介绍

这个功能在2.4版本deprecated，现在的版本里面已经没有这个功能了。

Redis的虚拟内存跟OS的虚拟内存是两个不同的东西，但是思路和目的是一样的，就是暂时把不经常访问的数据从内存交换到磁盘中，从而腾出宝贵的内存空间。为何这么做呢，Redis作者给出的理由是：

1：OS的虚拟内存是以4K页面为最小单位进行交换的，而Redis的大多数对象都远小于4K，所以一个OS页面上可能有多个Redis对象

2：另外Redis的集合对象类型如list,set可能存在于多个OS页面上，最终可能造成只有10%的Key被经常访问，但是所有OS页面都会被OS认为是活跃的，这样只有内 存真正耗尽时，OS才会交换页面

3：相比于OS的交换方式，Redis可以将被交换到磁盘的对象进行压缩，保存到磁盘的对象可以去除指针和对象元数据信息，一般压缩后的对象会比内存中的对象小10倍，这样能少做很多IO操作

- 虚拟内存相关的配置

1：vm-enabled：配置成yes，开启vm功能

2：vm-swap-file：设置交换出来的value保存的文件路径，默认/tmp/redis.swap

3：vm-max-memory：设置Redis使用的最大内存上限，超过上限后开始交换

4：vm-page-size：设置每个页面的大小，默认32个字节

5：vm-pages：在文件中最多使用多少页面，默认是134217728，交换文件的大小=vm-page-size * vm-pages

6：vm-max-threads：用于执行交换的线程数量，默认4，0表示不使用工作线程，而在主线程完成，这会造成阻塞，建议设置成CPU核数

Redis的虚拟内存-2

- Redis的虚拟内存机制

1：为了保证key的查找速度，只会将value交换到swap文件

2：Redis也是按页面来交换对象的，Redis规定同一个页面只能保存一个对象，但是一个对象可以保存在多个页面中

3：在Redis使用的内存没超过vm-max-memory之前，是不会交换任何value的。当超过最大内存限制后，会选择较老的对象进行交换，如果两个对象一样老，会优先交换比较大的对象，精确的公式swappability = age*log(size_in_memory)

4： 对于vm-page-size的设置应该根据自己的应用，将页面的大小设置为可以容纳大多数对象的大小。太大了会浪费磁盘空间，太小了会造成交换文件出现碎片

5：对于交换文件中的每个页面，Redis会在内存中对应一个1bit值来记录页面的空闲状态，这个空间是vm-pages配置的

适合使用Redis的场景

- 缓存
- 取最新N个数据的操作，如：可以将最新的50条评论的ID放在List集合
- 排行榜类的应用，取TOP N操作，前面操作以时间为权重，这个是以某个条件为权重，比如按顶的次数排序
- 计数器应用
- 存储关系：比如社交关系，比如Tag等
- 获取某段时间所有数据排重值，使用set，比如某段时间访问的用户ID，或者是客户端IP
- 构建队列系统，List可以构建栈和队列，使用zset可以构建优先级队列
- 实时分析系统，比如：访问频率控制
- 模拟类似于HttpSession这种需要设定过期时间的功能
- Pub/Sub构建实时消息系统
- 记录日志

Redis的优化-1

- 使用管道（Pipeline）

  Redis的底层通讯协议对管道提供了支持，通过管道，可以一次性发送多条命令给Redis，在执行完后一次性将结果取回。

  使用管道，可以减少客户端和Redis的通信次数，降低网络延时，从而提供性能。Redis的管道功能在命令行中没有，但Redis是支持管道的，而且在各个语言版的client中都有相应的实现。
- 精简键名和键值
- 合理设计存储的数据结构和数据关系，尽量减少数据冗余
- 尽量使用mset来赋值，比set效率高一个数量级；类似的还有lpush、zadd等都可以一次输入多个指令
- 如果可能，尽量使用Lua脚本来辅助获取或操作数据
- 尽量使用hash结构来存储对象

  将一个对象存储在hash类型中会占用更少的内存，并且可以更方便的存取整个对象，省内存的原因是新建一个hash对象时开始是用zipmap来存储的。
- 使用hash结构时，应尽量保证每个key下面的<field, value>的数目不超过限制（默认值为64），否则插入效率下降十分明显，同样，内存开销也会显著增加

Redis的优化-2

- 配置使用ziplist以优化list

  如果list的元素个数小于配置值list-max-ziplist-entries且元素值字符串的长度小于配置值list-max-ziplist-value，则可以编码成ziplist类型存储，否则采用 Dict 来存储，Dict实际是Hash Table的一种实现。

- 配置使用intset以优化set

  当set集合中的元素为整数且元素个数小于配置set-max-intset-entries值时，使用intset数据结构存储，否则转化为Dict结构

- 配置使用ziplist以优化sorted set

  当sorted set的元素个数及元素大小小于一定限制时，它是用ziplist来存储。这个限制的配置如下：zset-max-ziplist-entries、zset-max-ziplist-value

- 配置使用zipmap以优化hash

  当entry数量没有超过hash-max-ziplist-entries指定的限制，并且值的最大长度没有超过hash-max-ziplist-value指定的限制时，会用zipmap来编码。

  注意：HashMap的优势就是查找和操作的时间复杂度都是O(1)的，而放弃Hash采用一维存储则是O(n)的时间复杂度，如果成员数量很少，则影响不大，否则会严重影响性能，所以要权衡好些个值的设置，在时间成本和空间成本上进行权衡。

Redis的优化-3

- 一定要设置maxmemory

设置Redis使用的最大物理内存，也就是使用了这么多物理内存后就开始拒绝后续的写入请求，该参数能保护Redis不会因为使用了过多的物理内存而严重影响性能甚至崩溃。

- 对排序的优化

1：尽量让要排序的Key存放在一个Server上

如果采用客户端分片，那么具体决定哪个key存在哪个服务器上，是由client端采用一定算法来决定的，因此可以通过只对key的部分进行hash。比如：client如果发现key中包含{}，那么只对key中{}包含的内容进行hash。

如果采用服务端分片，也可以通过控制key的有效部分，来让这些数据分配到同一个插槽中。

2：尽量减少Sort的集合大小

如果要排序的集合非常大，会消耗很长时间，Redis单线程的，长时间的排序操作会阻塞其它client的请求。解决办法是通过主从复制，将数据复制到多个slave上，然后只在slave上做排序操作，并尽可能的对排序结果缓存。

Redis的优化-4

- 考虑采用复制+RDB的方式

  使用复制机制来实现高可用，数据采用RDB的方式进行持久化备份，建议只在Slave上持久化RDB文件，而且只要在一个相对较长的时间备份一次就够了，比如只保留save 900 1这条规则，大致就是15分钟保存一次。

  这样的方式避免了AOF带来的持续的IO，也避免AOF Rewrite最后将rewrite过程中产生的新数据写到新文件所造成的阻塞。代价是如果Master/Slave同时倒掉，可能会丢失15分钟的数据。

- 考虑在一台服务器启动多个Redis实例

  由于Redis使用单线程，为了提高CPU利用率，可以在同一台服务器上启动多个Redis实例，但这可能会带来严重的IO争用，除非Redis不需要持久化，或者有某种方式保证多个实例不会在同一个时间重写AOF。

Redis的Java客户端-1

- Redis官方推荐的Java客户端是jedis
  网址：https://github.com/xetorthio/jedis
- 构建开发环境，在Maven中添加

```
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
    <version>2.7.2</version>
</dependency>
```

- 连接池

1：jedis的连接池基于apach的commons-pool，要确保添加了相应的包
2：配置连接池，一般使用JedisPoolConfig，常见的设置：
（1）maxTotal：最大实例数，-1表示不限制，默认8
（2）maxIdle/minIdle：最大/小空闲实例数，默认8/0
（3）whenExhaustedAction：当池中的实例被分配完时，要采取的操作，默认有三种：
WHEN_EXHAUSTED_FAIL：直接抛出NoSuchElementException
WHEN_EXHAUSTED_BLOCK：阻塞住，达到maxWaitMillis时抛出例外，默认选项
WHEN_EXHAUSTED_GROW：新建一个实例，也就说设置的maxTotal无用
（4）maxWaitMillis：获取一个实例时，最大的等待毫秒数，如果超时抛出JedisConnectionException，默认-1，表示永远等待
（5）testOnBorrow：在获取实例时，是否验证对象有效，如果无效，会重新选择一个，默认false
（6）testOnReturn：在归还连接给池时，是否验证对象有效性，默认false

Redis的Java客户端-2

（7）testWhileIdle：是否对空闲实例验证对象有效性，失效的对象会被删除，默认false，仅在timeBetweenEvictionRunsMillis设置为正值时有效

（8）timeBetweenEvictionRunsMillis：空闲实例验证，两次扫描之间要sleep的毫秒数；默认是-1

（9）numTestsPerEvictionRun：空闲实例验证，每次扫描的最多的对象数，默认是3，仅在timeBetweenEvictionRunsMillis设置为正值时有效

（10）minEvictableIdleTimeMillis：一个实例至少停留在idle状态的最短时间，然后才能被空闲实例验证扫描并驱逐；默认30分钟，仅在timeBetweenEvictionRunsMillis设置为正值时有效

（11）lifo：设置采用last in first out队列，默认true

- 连接集群

以前的老版本Redis不支持集群的功能，因此是在客户端采用一致性Hash来对数据进行分片，jedis中对应的是SharedJedis；既然现在Redis已经支持集群了，就先看新的集群的写法：

```
Set<HostAndPort> jedisClusterNodes = new HashSet<HostAndPort>();
//Jedis Cluster will attempt to discover cluster nodes automatically
jedisClusterNodes.add(new HostAndPort("127.0.0.1", 7379));
JedisCluster jc = new JedisCluster(jedisClusterNodes);
jc.set("foo", "bar");
String value = jc.get("foo");
```

Redis的Java客户端-3

- 客户端分片的方式

1：定义一个List，里面包含多个JedisShardInfo

2：创建ShardedJedis对象，然后就可以通过这个对象操作命令了，操作完后可以disconnect

3：可以通过getShardInfo(key)方法来获取使用的分片信息

4：可以通过设置key tag pattern来保证key 位于同一个shard

5：分片的连接池对象是：ShardJedisPool，配置对象仍然是JedisPoolConfig，例如：

```
ShardJedisPool  pool =new ShardedJedisPool(jedisPoolConfig, jedsInfoList,
Hashing.MURMUR_HASH,Sharded.DEFAULT_KEY_TAG_PATTERN);
```

Redis和Spring集成-1

- 构建开发环境，在Maven中添加spring-data-redis支持的jedis版本目前只是到了2.7.0，所以要修改一下前面的配置，然后添加：

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
    <version>1.5.1.RELEASE</version>
</dependency>
<dependency>
    <groupId>net.sourceforge.cobertura</groupId>
    <artifactId>cobertura</artifactId>
    <version>2.1.1</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
    <version>2.4</version>
</dependency>
```

Redis和Spring集成-2

- 在Spring的配置文件中添加

```
<bean id="poolConfig" class="redis.clients.jedis.JedisPoolConfig">
    <property name="maxTotal" value="8"></property>
    <property name="maxIdle" value="8"></property>
    <property name="maxWaitMillis" value="1000"></property>
</bean>
<bean id="jedisConnectionFactory"
    class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory">
    <property name="hostName" value="192.168.1.106" />
    <property name="port" value="6379" />
    <property name="usePool" value="true" />
    <property name="timeout" value="100000" />
    <constructor-arg index="0" ref="poolConfig" />
</bean>
<bean id="testClient" class="com.javass.TestClient">
    <property name="connectionFactory" ref="jedisConnectionFactory"></property>
</bean>
```

- 在Java程序中，就可以使用RedisTemplate了，例如：

```
String s = ""+this.execute(new RedisCallback(){
    public Object doInRedis(RedisConnection jr) throws DataAccessException {
          return new String(jr.get("k1".getBytes()));
}});
```