1、完全基于内存，绝大部分请求是纯粹的内存操作，非常快速。数据存在内存中，类似于HashMap，HashMap的优势就是查找和操作的时间复杂度都是O(1)；

2、数据结构简单，对数据操作也简单，Redis中的数据结构是专门进行设计的；

// 为什么redis是单线程的

3、采用单线程，避免了不必要的上下文切换和竞争条件，也不存在多进程或者多线程导致的切换而消耗 CPU，不用去考虑各种锁的问题，不存在加锁释放锁操作，没有因为可能出现死锁而导致的性能消耗；

4、使用多路I/O复用模型，非阻塞IO；

5、使用底层模型不同，它们之间底层实现方式以及与客户端之间通信的应用协议不一样，Redis直接自己构建了VM 机制 ，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求；

以上几点都比较好理解，下边我们针对多路 I/O 复用模型进行简单的探讨：

（1）多路 I/O 复用模型

多路I/O复用模型是利用 select、poll、epoll 可以同时监察多个流的 I/O 事件的能力，在空闲的时候，会把当前线程阻塞掉，当有一个或多个流有 I/O 事件时，就从阻塞态中唤醒，于是程序就会轮询一遍所有的流（epoll 是只轮询那些真正发出了事件的流），并且只依次顺序的处理就绪的流，这种做法就避免了大量的无用操作。

这里“多路”指的是多个网络连接，“复用”指的是复用同一个线程。采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗），且 Redis 在内存中操作数据的速度非常快，也就是说内存内的操作不会成为影响Redis性能的瓶颈，主要由以上几点造就了 Redis 具有很高的吞吐量。

————————————————

版权声明：本文为CSDN博主「人生丶几度」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/weixin_42189604/java/article/details/82703391>

、Redis功能介绍

高速读写

数据类型丰富 \*\*\*\*\*\*\*\*\*

注意:redis主要做数据库的存储,选一种与Mysql数据类型差不多的数据结构

支持持久化 \*\*\*\*\*\*\*\*

多种内存分配及回收策略

支持事务 \*\*\*\* \*\*\*\*

消息队列、消息订阅

支持高可用 \*\*\*\*

支持分布式分片集群 \*\*\*\*\*

缓存穿透\雪崩 \*\*\*\*\*

Redis API

2、企业缓存产品介绍

Memcached:

优点：高性能读写、单一数据类型、支持客户端式分布式集群、一致性hash

1

多核结构、多线程读写性能高。

缺点：无持久化、节点故障可能出现缓存穿透、分布式需要客户端实现、跨机房数据同步困难、架构扩容复杂度

高

Redis: 优点：高性能读写、多数据类型支持、数据持久化、高可用架构、支持自定义虚拟内存、支持分布式分片集群、单线程读写性能极高

缺点：多线程读写较Memcached慢

1

新浪、京东、直播类平台、网页游戏

memcache与redis在读写性能的对比

memcached 适合,多用户访问,每个用户少量的rw

redis 适合,少用户访问,每个用户大量rw

1234

Tair：

优点：高性能读写、支持三种存储引擎（ddb、rdb、ldb）、支持高可用、支持分布式分片集群、支撑了几乎所有淘宝业务的缓存。

缺点：单机情况下，读写性能较其他两种产品较慢

3、Redis使用场景介绍

Memcached：多核的缓存服务，更加适合于多用户并发访问次数较少的应用场景

Redis：单核的缓存服务，单节点情况下，更加适合于少量用户，多次访问的应用场景。

Redis一般是单机多实例架构，配合redis集群出现。

4、Redis安装部署

下载：

wget http://download.redis.io/releases/redis-3.2.12.tar.gz

解压：

上传至 /data

tar xzf redis-3.2.12.tar.gz

mv redis-3.2.12 redis

安装：

cd redis

make

启动：

src/redis-server &

环境变量：

vim /etc/profile

export PATH=/data/redis/src:$PATH

source /etc/profile

进入redis目录下启动,并且进入redis:

redis-server &

redis-cli

127.0.0.1:6379> set num 10

OK

127.0.0.1:6379> get num

10

12345678910111213141516171819202122232425

5、Redis基本管理操作

5.1 基础配置文件介绍：

[root@standby ~]# redis-cli shutdown

mkdir /data/6379

查看并且更改配置文件

cat >>/data/6379/redis.conf <<EOF

daemonize yes

port 6379

logfile /data/6379/redis.log

dir /data/6379

dbfilename dump.rdb

EOF

重启redis

redis-cli shutdown

redis-server /data/6379/redis.conf

netstat -lnp|grep 63

12345678910111213141516

+++++++++++++++++++++++++

配置文件说明

redis.conf

是否后台运行：

daemonize yes

默认端口：

port 6379

日志文件位置

logfile /var/log/redis.log

持久化文件存储位置

dir /data/6379

RDB持久化数据文件:

dbfilename dump.rdb

+++++++++++++++++++++++++

redis-cli

127.0.0.1:6379> set name zhangsan

OK

127.0.0.1:6379> get name

"zhangsan"

redis-cli 客户端命令常用参数说明

redis-cli 刚装完,可以在redis服务器上直接登录redis

-p 6379 指定端口号

-h 指定链接地址

-a 指定链接密码

redis-cli set num 10 ,无交互执行redis命令

cat /tmp/1.txt |redis-cli

[root@db01 ~]# redis-cli -h 10.0.0.51 -p 6379

10.0.0.51:6379>

1234567891011121314151617

5.2 redis安全配置

redis默认开启了保护模式，只允许本地回环地址登录并访问数据库。

protected-mode yes/no （保护模式是否开启，是否只允许本地访问）

(1)Bind :指定IP进行监听

echo "bind 10.0.0.200 127.0.0.1" >>/data/6379/redis.conf

(2)增加requirepass {password}

echo "requirepass 123" >>/data/6379/redis.conf

重启redis

redis-cli shutdown

redis-server /data/6379/redis.conf

12345678

验证:

方法一：

[root@db03 ~]# redis-cli -a 123

127.0.0.1:6379> set name zhangsan

OK

127.0.0.1:6379> exit

方法二：

[root@db03 ~]# redis-cli

127.0.0.1:6379> auth 123

OK

127.0.0.1:6379> set a b

12345678910

5.3在线查看和修改配置

CONFIG GET \*

CONFIG GET requirepass

CONFIG SET requirepass 123

123

5.4 redis持久化(内存数据保存到硬盘)

RDB和AOF

作用:可以有效防止,在redis宕机后,缓存失效的问题.

RDB持久化(只存更新之后的数据)

可以在指定的花四溅间隔内生成数据集的时间点快照(point-in-time snapshot)

优点：速度快，适合于用做备份，主从复制也是基于RDB持久化功能实现的。

缺点：会有数据丢失

rdb持久化核心配置参数：

vim /data/6379/redis.conf

dir /data/6379

dbfilename dump.rdb

save 900 1

save 300 10

save 60 10000

配置分别表示：

900秒（15分钟）内有1个更改

300秒（5分钟）内有10个更改

60秒内有10000个更改

1234567891011

AOF 持久化(append-only log file)

记录服务器执行的所有写操作命令,并在服务器启动时,通过重新执行这些命令来还原数据集

AOF 文件中的命令全部以 Redis 协议的/格式来保存，新命令会被追加到文件的末尾

优点：可以最大程度保证数据不丢

缺点：日志记录量级比较大

12

AOF持久化配置

appendonly yes

appendfsync everysec

appendfsync always

appendfsync no

vim /data/6379/redis.conf

appendonly yes

appendfsync everysec

12345678910

AOF日志功能执行了什么操作?

是否打开aof日志功能

每1个命令,都立即同步到aof (每秒写1次)

写入工作交给操作系统,由操作系统判断缓冲区大小,统一写入到aof.

注意:

redis 持久化方式有哪些？有什么区别？

rdb：基于快照的持久化，速度更快，一般用作备份，主从复制也是依赖于rdb持久化功能

aof：以追加的方式记录redis操作日志的文件。可以最大程度的保证redis数据安全，类似于mysql的binlog

123

6、Redis数据类型

6.1 介绍

数据类型表示

名称

存储格式

String

字符串

key value

Hash

字典类型(哈希)

key col1 value1 col2 value2

List

列表

key [a,b,c,d]

Set

集合

key1 (a,b,c,d,b)

Sorted set

有序集合(自动排序,去重)

key1 ( a, b, c, d)

6.2 键的通用操作

查询语句

查询的内容

举例

KEYS \* keys a keys a\*

查看已存在所有键的名字 \*\*\*\*

TYPE

返回键所存储值的类型 \*\*\*\*

EXISTS

检查是否存在 \*\*\*\*\*

EXPIRE\ PEXPIRE

以秒\毫秒设定生存时间 \*\*\*

EXPIRE name 60

TTL\ PTTL

以秒\毫秒为单位返回生存时间 \*\*\*

PERSIST

取消生存实现设置 \*\*\*

PERSIST a

DEL

删除一个key

RENAME

变更KEY名

6.3 string

\*\*应用场景:\*\*常规计数,微博数,粉丝数等.订阅,礼物,页游 (key:value形式)

详细例子:

增

set mykey "test" 为键设置新值，并覆盖原有值

getset mycounter 0 设置值,取值同时进行

setex mykey 10 "hello" 设置指定 Key 的过期时间为10秒,在存活时间可以获取value

setnx mykey "hello" 若该键不存在，则为键设置新值

mset key3 "zyx" key4 "xyz" 批量设置键

删

del mykey 删除已有键

改

append mykey "hello" 若该键并不存在,返回当前 Value 的长度

该键已经存在，返回追加后 Value的长度

incr mykey 值增加1,若该key不存在,创建key,初始值设为0,增加后结果为1

decrby mykey 5 值减少5

setrange mykey 20 dd 把第21和22个字节,替换为dd, 超过value长度,自动补0

查

exists mykey 判断该键是否存在，存在返回 1，否则返回0

get mykey 获取Key对应的value

strlen mykey 获取指定 Key 的字符长度

ttl mykey 查看一下指定 Key 的剩余存活时间(秒数)

getrange mykey 1 20 获取第2到第20个字节,若20超过value长度,则截取第2个和后面所有的

mget key3 key4 批量获取键

123456789101112131415161718192021222324

（1）

set name zhangsan

（2）

MSET id 101 name zhangsan age 20 gender m

等价于以下操作：

SET id 101

set name zhangsan

set age 20

set gender m

mget id name age gender

（3）计数器

每点一次关注，都执行以下命令一次

127.0.0.1:6379> incr fans\_count

(integer) 10003

127.0.0.1:6379> get fans\_count

"10003"

127.0.0.1:6379> incrby fans\_count 1000

(integer) 11003

127.0.0.1:6379> decr fans\_count

(integer) 11002

127.0.0.1:6379> decrby fans\_count 1000

1234567891011121314151617181920212223

6.4 hash类型

\*\*应用场景:\*\*存储部分变更数据,如:用户信息等, 最接近mysql表结构的一种类型

存数据：

hmset stu id 101 name zs age 18

id name age

101 zs 18

insert into stu(id,name,age) values (101,'zs',18);

取数据：

HMGET stu id name age -----> select id,name,age from stu;

hgetall stu ------> select \* from stu;

将mysql中world.city表前10行数据导入redis

hmset city\_1 id 1 name Kabul contrycode AFG district Kabol population 1780000

select concat("hmset city\_",id," id ",id," name ",name," countrycode ",countrycode," district ",district," population ",population) from world.city into outfile '/tmp/hmset1.txt'

cat /tmp/hmset.txt |redis-cli -a 123

hmset city\_1 id 1 name Kabul contrycode AFG district Kabol population 1780000

12345678910111213141516171819202122

具体例子:

增

hset myhash field1 "s"

若字段field1不存在,创建该键及与其关联的Hashes, Hashes中,key为field1 ,并设value为s ，若存在会覆盖原value

hsetnx myhash field1 s

若字段field1不存在,创建该键及与其关联的Hashes, Hashes中,key为field1 ,并设value为s， 若字段field1存在,则无效

hmset myhash field1 "hello" field2 "world 一次性设置多个字段

删

hdel myhash field1 删除 myhash 键中字段名为 field1 的字段

del myhash 删除键

改

hincrby myhash field 1 给field的值加1

查

hget myhash field1 获取键值为 myhash,字段为 field1 的值

hlen myhash 获取myhash键的字段数量

hexists myhash field1 判断 myhash 键中是否存在字段名为 field1 的字段

hmget myhash field1 field2 field3 一次性获取多个字段

hgetall myhash 返回 myhash 键的所有字段及其值

hkeys myhash 获取myhash 键中所有字段的名字

hvals myhash 获取 myhash 键中所有字段的值

1234567891011121314151617181920

6.5 LIST（列表）

应用场景:

消息队列系统

比如sina微博:在Redis中我们的最新微博ID使用了常驻缓存，这是一直更新的。

但是做了限制不能超过5000个ID，因此获取ID的函数会一直询问Redis。

只有在start/count参数超出了这个范围的时候，才需要去访问数据库。

系统不会像传统方式那样“刷新”缓存，Redis实例中的信息永远是一致的。

SQL数据库（或是硬盘上的其他类型数据库）只是在用户需要获取“很远”的数据时才会被触发，

而主页或第一个评论页是不会麻烦到硬盘上的数据库了。

微信朋友圈：

127.0.0.1:6379> LPUSH wechat "today is nice day !"

127.0.0.1:6379> LPUSH wechat "today is bad day !"

127.0.0.1:6379> LPUSH wechat "today is good day !"

127.0.0.1:6379> LPUSH wechat "today is rainy day !"

127.0.0.1:6379> LPUSH wechat "today is friday !"

[e,d,c,b,a]

0 1 2 3 4

127.0.0.1:6379> lrange wechat 0 0

1) "today is friday !"

127.0.0.1:6379> lrange wechat 0 1

1) "today is friday !"

2) "today is rainy day !"

127.0.0.1:6379> lrange wechat 0 2

1) "today is friday !"

2) "today is rainy day !"

3) "today is good day !"

127.0.0.1:6379> lrange wechat 0 3

127.0.0.1:6379> lrange wechat -2 -1

1) "today is bad day !"

2) "today is nice day !"

12345678910111213141516171819202122232425262728

具体例子:

增

lpush mykey a b 若key不存在,创建该键及与其关联的List,依次插入a ,b， 若List类型的key存在,则插入value中

lpushx mykey2 e 若key不存在,此命令无效， 若key存在,则插入value中

linsert mykey before a a1 在 a 的前面插入新元素 a1

linsert mykey after e e2 在e 的后面插入新元素 e2

rpush mykey a b 在链表尾部先插入b,在插入a

rpushx mykey e 若key存在,在尾部插入e, 若key不存在,则无效

rpoplpush mykey mykey2 将mykey的尾部元素弹出,再插入到mykey2 的头部(原子性的操作)

删

del mykey 删除已有键

lrem mykey 2 a 从头部开始找,按先后顺序,值为a的元素,删除数量为2个,若存在第3个,则不删除

ltrim mykey 0 2 从头开始,索引为0,1,2的3个元素,其余全部删除

改

lset mykey 1 e 从头开始, 将索引为1的元素值,设置为新值 e,若索引越界,则返回错误信息

rpoplpush mykey mykey 将 mykey 中的尾部元素移到其头部

查

lrange mykey 0 -1 取链表中的全部元素，其中0表示第一个元素,-1表示最后一个元素。

lrange mykey 0 2 从头开始,取索引为0,1,2的元素

lrange mykey 0 0 从头开始,取第一个元素,从第0个开始,到第0个结束

lpop mykey 获取头部元素,并且弹出头部元素,出栈

lindex mykey 6 从头开始,获取索引为6的元素 若下标越界,则返回nil

123456789101112131415161718192021

6.6 SET 集合类型（join union）

应用场景:

案例：在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。

Redis还为集合提供了求交集、并集、差集等操作，可以非常方便的实现如共同关注、共同喜好、二度好友等功能，

对上面的所有集合操作，你还可以使用不同的命令选择将结果返回给客户端还是存集到一个新的集合中。

127.0.0.1:6379> sadd lxl pg1 pg2 songlaoban oldnie oldchen marong

(integer) 6

127.0.0.1:6379> sadd jnl baoqiang yufan oldchen songzhe

(integer) 4

127.0.0.1:6379> sadd jnl baoqiang yufan oldchen songzhe oldguo alexdsb

(integer) 2

127.0.0.1:6379>

127.0.0.1:6379> smembers lxl

1) "pg2"

2) "pg1"

3) "oldnie"

4) "songlaoban"

5) "marong"

6) "oldchen"

127.0.0.1:6379> smembers jnl

1) "alexdsb"

2) "yufan"

3) "oldguo"

4) "songzhe"

5) "baoqiang"

6) "oldchen"

127.0.0.1:6379>

127.0.0.1:6379> SUNION lxl jnl

1) "marong"

2) "pg2"

3) "pg1"

4) "oldchen"

5) "alexdsb"

6) "yufan"

7) "songlaoban"

8) "baoqiang"

9) "oldnie"

10) "songzhe"

11) "oldguo"

127.0.0.1:6379> SINTER lxl jnl

1) "oldchen"

127.0.0.1:6379>

127.0.0.1:6379> SDIFF lxl jnl

1) "songlaoban"

2) "oldnie"

3) "pg1"

4) "pg2"

5) "marong"

127.0.0.1:6379> SDIFF jnl lxl

1) "alexdsb"

2) "yufan"

3) "songzhe"

4) "oldguo"

5) "baoqiang"

1234567891011121314151617181920212223242526272829303132333435363738394041424344454647484950515253

具体例子:

增

sadd myset a b c

若key不存在,创建该键及与其关联的set,依次插入a ,b,若key存在,则插入value中,若a 在myset中已经存在,则插入了 d 和 e 两个新成员。

删

spop myset 尾部的b被移出,事实上b并不是之前插入的第一个或最后一个成员

srem myset a d f 若f不存在, 移出 a、d ,并返回2

改

smove myset myset2 a 将a从 myset 移到 myset2，

查

sismember myset a 判断 a 是否已经存在，返回值为 1 表示存在。

smembers myset 查看set中的内容

scard myset 获取Set 集合中元素的数量

srandmember myset 随机的返回某一成员

sdiff myset1 myset2 myset3 1和2得到一个结果,拿这个集合和3比较,获得每个独有的值

sdiffstore diffkey myset myset2 myset3 3个集和比较,获取独有的元素,并存入diffkey 关联的Set中

sinter myset myset2 myset3 获得3个集合中都有的元素

sinterstore interkey myset myset2 myset3 把交集存入interkey 关联的Set中

sunion myset myset2 myset3 获取3个集合中的成员的并集

sunionstore unionkey myset myset2 myset3 把并集存入unionkey 关联的Set中

12345678910111213141516171819

6.7 SortedSet（有序集合）

应用场景：

排行榜应用，取TOP N操作

这个需求与上面需求的不同之处在于，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，

这时候就需要我们的sorted set出马了，将你要排序的值设置成sorted set的score，将具体的数据设置成相应的value，

每次只需要执行一条ZADD命令即可。

127.0.0.1:6379> zadd topN 0 smlt 0 fskl 0 fshkl 0 lzlsfs 0 wdhbx 0 wxg

(integer) 6

127.0.0.1:6379> ZINCRBY topN 100000 smlt

"100000"

127.0.0.1:6379> ZINCRBY topN 10000 fskl

"10000"

127.0.0.1:6379> ZINCRBY topN 1000000 fshkl

"1000000"

127.0.0.1:6379> ZINCRBY topN 100 lzlsfs

"100"

127.0.0.1:6379> ZINCRBY topN 10 wdhbx

"10"

127.0.0.1:6379> ZINCRBY topN 100000000 wxg

"100000000"

127.0.0.1:6379> ZREVRANGE topN 0 2

1) "wxg"

2) "fshkl"

3) "smlt"

127.0.0.1:6379> ZREVRANGE topN 0 2 withscores

1) "wxg"

2) "100000000"

3) "fshkl"

4) "1000000"

5) "smlt"

6) "100000"

127.0.0.1:6379>

123456789101112131415161718192021222324252627

具体例子:

增

zadd myzset 2 "two" 3 "three" 添加两个分数分别是 2 和 3 的两个成员

删

zrem myzset one two 删除多个成员变量,返回删除的数量

改

zincrby myzset 2 one 将成员 one 的分数增加 2，并返回该成员更新后的分数

查

zrange myzset 0 -1 WITHSCORES 返回所有成员和分数,不加WITHSCORES,只返回成员

zrank myzset one 获取成员one在Sorted-Set中的位置索引值。0表示第一个位置

zcard myzset 获取 myzset 键中成员的数量

zcount myzset 1 2 获取分数满足表达式 1 <= score <= 2 的成员的数量

zscore myzset three 获取成员 three 的分数

zrangebyscore myzset 1 2 获取分数满足表达式 1 < score <= 2 的成员

#-inf 表示第一个成员，+inf最后一个成员

#limit限制关键字

#2 3 是索引号

zrangebyscore myzset -inf +inf limit 2 3 返回索引是2和3的成员

zremrangebyscore myzset 1 2 删除分数 1<= score <= 2 的成员，并返回实际删除的数量

zremrangebyrank myzset 0 1 删除位置索引满足表达式 0 <= rank <= 1 的成员

zrevrange myzset 0 -1 WITHSCORES 按位置索引从高到低,获取所有成员和分数

#原始成员:位置索引从小到大

one 0

two 1

#执行顺序:把索引反转

位置索引:从大到小

one 1

two 0

#输出结果: two

one

zrevrange myzset 1 3 获取位置索引,为1,2,3的成员

#相反的顺序:从高到低的顺序

zrevrangebyscore myzset 3 0 获取分数 3>=score>=0的成员并以相反的顺序输出

zrevrangebyscore myzset 4 0 limit 1 2 获取索引是1和2的成员,并反转位置索引

123456789101112131415161718192021222324252627282930313233

7 Redis事务

redis的事务是基于队列实现的。

mysql的事务是基于事务日志实现的。

开启事务功能时（multi）

multi

command1

command2

command3

command4

4条语句作为一个组，并没有真正执行，而是被放入同一队列中。

如果，这是执行discard，会直接丢弃队列中所有的命令，而不是做回滚。

exec

当执行exec时，对列中所有操作提交(要不全成功要不全失败)

123456789101112

127.0.0.1:6379> set a b

OK

127.0.0.1:6379> MULTI

OK

127.0.0.1:6379> set a b

QUEUED

127.0.0.1:6379> set c d

QUEUED

127.0.0.1:6379> exec

1) OK

2) OK

1234567891011

8、redis乐观锁实现（模拟买票）

发布一张票

set ticket 1

窗口1：

watch ticket

multi

set ticket 0 1---->0

窗口2：

multi

set ticket 0

exec

窗口1：

exec

123456789101112131415

9、 服务器管理命令

Info

Clinet list

Client kill ip:port

config get \*

CONFIG RESETSTAT 重置统计

CONFIG GET/SET 动态修改

Dbsize

FLUSHALL 清空所有数据

select 1

FLUSHDB 清空当前库

MONITOR 监控实时指令

SHUTDOWN 关闭服务器

关闭数据库：

redis-cli -a root shutdown

12345678910111213141516

10、redis（master-replicaset）

原理：

从服务器向主服务器发送 SYNC 命令。

接到 SYNC 命令的主服务器会调用BGSAVE 命令，创建一个 RDB 文件，并使用缓冲区记录接下来执行的所有写命令。

当主服务器执行完 BGSAVE 命令时，它会向从服务器发送 RDB 文件，而从服务器则会接收并载入这个文件。

主服务器将缓冲区储存的所有写命令（广播形式）发送给从服务器执行。

注意:

1、在开启主从复制的时候，使用的是RDB方式的，同步主从数据的

2、同步开始之后，通过主库命令传播的方式，主动的复制方式实现

3、2.8以后实现PSYNC的机制，实现断线重连

主从数据一致性保证：

min-slaves-to-write 1 # 至少一个从库连接

min-slaves-max-lag # 数据响应延迟几秒

12

这个特性的运作原理：

从服务器以每秒一次的频率 PING 主服务器一次， 并报告复制流的处理情况。

主服务器会记录各个从服务器最后一次向它发送 PING 的时间。

用户可以通过配置， 指定网络延迟的最大值 min-slaves-max-lag ，

以及执行写操作所需的至少从服务器数量 min-slaves-to-write 。

如果至少有 min-slaves-to-write 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。

你可以将这个特性看作 CAP 理论中的 C 的条件放宽版本： 尽管不能保证写操作的持久性，

但起码丢失数据的窗口会被严格限制在指定的秒数中。

另一方面， 如果条件达不到 min-slaves-to-write 和 min-slaves-max-lag 所指定的条件， 那么写操作就不会被执行

主服务器会向请求执行写操作的客户端返回一个错误。

主库是否要开启持久化？

如果不开有可能，主库重启操作，造成所有主从数据丢失！

主从复制实现(自愈的体现)

1、环境：

准备两个或两个以上redis实例

mkdir /data/638{0..2}

配置文件示例：

cat >> /data/6380/redis.conf << EOF

port 6380

daemonize yes

pidfile /data/6380/redis.pid

loglevel notice

logfile "/data/6380/redis.log"

dbfilename dump.rdb

dir /data/6380

requirepass 123

masterauth 123

EOF

cp /data/6380/redis.conf /data/6381/redis.conf

cp /data/6380/redis.conf /data/6382/redis.conf

sed -i 's#6380#6381#g' /data/6381/redis.conf

sed -i 's#6380#6382#g' /data/6382/redis.conf

启动：

redis-server /data/6380/redis.conf

redis-server /data/6381/redis.conf

redis-server /data/6382/redis.conf

netstat -lnp|grep 638

主节点：6380

从节点：6381、6382

2、开启主从：

6381/6382命令行:

redis-cli -p 6381 -a 123 SLAVEOF 127.0.0.1 6380

redis-cli -p 6382 -a 123 SLAVEOF 127.0.0.1 6380

3、查询主从状态

redis-cli -p 6380 -a 123 info replication

redis-cli -p 6381 -a 123 info replication

redis-cli -p 6382 -a 123 info replication

4、从库切为主库(6382为主库)

模拟主库故障

redis-cli -p 6380 -a 123 shutdown

redis-cli -p 6381 -a 123

info replication

slaveof no one

6382连接到6381：

[root@db03 ~]# redis-cli -p 6382 -a 123

127.0.0.1:6382> SLAVEOF no one

127.0.0.1:6382> SLAVEOF 127.0.0.1 6381

redis内部操作分析(redis-sentinel（哨兵）)

1、监控

2、自动选主，切换（6381 slaveof no one）

3、2号从库（6382）指向新主库（6381）

4、应用透明

12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364

sentinel搭建过程

mkdir /data/26380

cd /data/26380

cat >> sentinel.conf << EOF

port 26380

dir "/data/26380"

sentinel monitor mymaster 127.0.0.1 6380 1

sentinel down-after-milliseconds mymaster 5000

sentinel auth-pass mymaster 123

EOF

启动：

redis-sentinel /data/26380/sentinel.conf &

分析:

如果有问题：

1、重新准备1主2从环境

2、kill掉sentinel进程

3、删除sentinel目录下的所有文件

4、重新搭建sentinel

12345678910111213141516171819202122

停主库测试：

[root@db01 ~]# redis-cli -p 6380

shutdown

[root@db01 ~]# redis-cli -p 6381

info replication

启动源主库（6380），看状态。

Sentinel管理命令：

redis-cli -p 26380

分析:

PING ：返回 PONG 。

SENTINEL masters ：列出所有被监视的主服务器

SENTINEL slaves <master name>

SENTINEL get-master-addr-by-name <master name> ： 返回给定名字的主服务器的 IP 地址和端口号。

SENTINEL reset <pattern> ： 重置所有名字和给定模式 pattern 相匹配的主服务器。

SENTINEL failover <master name> ： 当主服务器失效时， 在不询问其他 Sentinel 意见的情况下， 强制开始一次自动故障迁移。

1234567891011121314151617181920

11.高性能与高可用在redis中的应用：

高性能

1、在多分片节点中，将16384个槽位，均匀分布到多个分片节点中

2、存数据时，将key做crc16(key),然后和16384进行取模，得出槽位值（0-16383之间）

3、根据计算得出的槽位值，找到相对应的分片节点的主节点，存储到相应槽位上

4、如果客户端当时连接的节点不是将来要存储的分片节点，分片集群会将客户端连接切换至真正存储节点进行数据存储

高可用：

在搭建集群时，会为每一个分片的主节点，对应一个从节点，实现slaveof的功能，同时当主节点down，实现类似于sentinel的自动failover的功能。

1、redis会有多组分片构成（3组）

2、redis cluster 使用固定个数的slot存储数据（一共16384slot）

3、每组分片分得1/3 slot个数（0-5500 5501-11000 11001-16383）

4、基于CRC16(key) % 16384 ====》值 （槽位号）

规划搭建过程:

6个redis实例，一般会放到3台硬件服务器

注：在企业规划中，一个分片的两个分到不同的物理机，防止硬件主机宕机造成的整个分片数据丢失。

1、安装集群插件

EPEL源安装ruby支持

yum install ruby rubygems -y

使用国内源

gem sources -l

gem sources -a http://mirrors.aliyun.com/rubygems/

gem sources --remove https://rubygems.org/

gem sources -l

gem install redis -v 3.3.3

123456789

2、集群节点准备

mkdir /data/700{0..5}

cat >> /data/7000/redis.conf << EOF

port 7000

daemonize yes

pidfile /data/7000/redis.pid

loglevel notice

logfile "/data/7000/redis.log"

dbfilename dump.rdb

dir /data/7000

protected-mode no

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

EOF

cp /data/7000/redis.conf /data/7001/redis.conf

cp /data/7000/redis.conf /data/7002/redis.conf

cp /data/7000/redis.conf /data/7003/redis.conf

cp /data/7000/redis.conf /data/7004/redis.conf

cp /data/7000/redis.conf /data/7005/redis.conf

sed -i 's#7000#7001#g' /data/7001/redis.conf

sed -i 's#7000#7002#g' /data/7002/redis.conf

sed -i 's#7000#7003#g' /data/7003/redis.conf

sed -i 's#7000#7004#g' /data/7004/redis.conf

sed -i 's#7000#7005#g' /data/7005/redis.conf

启动节点：

redis-server /data/7000/redis.conf

redis-server /data/7001/redis.conf

redis-server /data/7002/redis.conf

redis-server /data/7003/redis.conf

redis-server /data/7004/redis.conf

redis-server /data/7005/redis.conf

[root@db01 ~]# ps -ef |grep redis

root 8854 1 0 03:56 ? 00:00:00 redis-server \*:7000 [cluster]

root 8858 1 0 03:56 ? 00:00:00 redis-server \*:7001 [cluster]

root 8860 1 0 03:56 ? 00:00:00 redis-server \*:7002 [cluster]

root 8864 1 0 03:56 ? 00:00:00 redis-server \*:7003 [cluster]

root 8866 1 0 03:56 ? 00:00:00 redis-server \*:7004 [cluster]

root 8874 1 0 03:56 ? 00:00:00 redis-server \*:7005 [cluster]

1234567891011121314151617181920212223242526272829303132333435363738394041424344

3、将节点加入集群管理

redis-trib.rb create --replicas 1 127.0.0.1:7000 127.0.0.1:7001 \

127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005

12

4、集群状态查看

集群主节点状态

redis-cli -p 7000 cluster nodes | grep master

集群从节点状态

redis-cli -p 7000 cluster nodes | grep slave

1234

5、集群节点管理

5.1增加新的节点

mkdir /data/7006

mkdir /data/7007

vim /data/7006/redis.conf

port 7006

daemonize yes

pidfile /data/7006/redis.pid

loglevel notice

logfile "/data/7006/redis.log"

dbfilename dump.rdb

dir /data/7006

protected-mode no

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

vim /data/7007/redis.conf

port 7007

daemonize yes

pidfile /data/7007/redis.pid

loglevel notice

logfile "/data/7007/redis.log"

dbfilename dump.rdb

dir /data/7007

protected-mode no

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

redis-server /data/7006/redis.conf

redis-server /data/7007/redis.conf

1234567891011121314151617181920212223242526272829303132333435

5.2 添加主节点：

redis-trib.rb add-node 127.0.0.1:7006 127.0.0.1:7000

1

5.3转移slot（重新分片）

redis-trib.rb reshard 127.0.0.1:7000

1

5.4 添加一个从节点

redis-trib.rb add-node --slave --master-id 1c98b2b2ce18f88c76821cdb82dba4defaa5eb48 127.0.0.1:7007 127.0.0.1:7000

1

6.删除节点

6.1 将需要删除节点slot移动走

redis-trib.rb reshard 127.0.0.1:7000

1

6.2 删除一个节点

删除master节点之前首先要使用reshard移除master的全部slot,然后再删除当前节点

redis-trib.rb del-node 127.0.0.1:7006 1c98b2b2ce18f88c76821cdb82dba4defaa5eb48

redis-trib.rb del-node 127.0.0.1:7007 00185d1cf069b23468d5863202ac651f0d02a9f8

123

设置redis最大内存

config set maxmemory 102400000

1

注意:

缓存穿透

概念

访问一个不存在的key，缓存不起作用，请求会穿透到DB，流量大时DB会挂掉。

解决方案

采用布隆过滤器，使用一个足够大的bitmap，用于存储可能访问的key，不存在的key直接被过滤；

访问key未在DB查询到值，也将空值写进缓存，但可以设置较短过期时间。

缓存雪崩

概念

大量的key设置了相同的过期时间，导致在缓存在同一时刻全部失效，造成瞬时DB请求量大、压力骤增，引起雪崩。

解决方案

可以给缓存设置过期时间时加上一个随机值时间，使得每个key的过期时间分布开来，不会集中在同一时刻失效。

缓存击穿

概念

一个存在的key，在缓存过期的一刻，同时有大量的请求，这些请求都会击穿到DB，造成瞬时DB请求量大、压力骤增。

解决方案

在访问key之前，采用SETNX（set if not exists）来设置另一个短期key来锁住当前key的访问，访问结束再删除该短期key。

————————————————

版权声明：本文为CSDN博主「初相识.」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_42737056/java/article/details/86531310