**redis学习**

# 简介

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件。

它支持多种类型的数据结构，如 字符串（strings）， 散列（hashes）， 列表（lists）， 集合（sets），有序集合（sorted sets） 与范围查询， bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。

Redis 内置了 复制（replication），LUA脚本（Lua scripting），LRU驱动事件（LRU eviction），事务（transactions）和不同级别的 磁盘持久化（persistence）， 并通过 Redis哨兵（Sentinel）

和自动分区（Cluster）提供高可用性（high availability）。

不要把 Redis 用作主要的数据存储数据库！！！！

不能存储太多的信息！！大数据量的信息不要存储到Redis

Redis 与其他 key - value 缓存产品有以下三个特点：

1、支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

2、不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。

3、支持数据的备份，即master-slave模式的数据备份。

优势

1、性能极高：Redis能读的速度是110000次/s，写的速度是81000次/s 。

2、丰富的数据类型：Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

3、原子：Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。

单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。

4、丰富的特性：Redis还支持 publish/subscribe, 通知, key 过期等等特性

# 文档

1、官网文档：https://redis.io/documentation

2、中文文档：http://www.redis.cn/documentation.html

# 安装

linux版本最新的是4.0.9

链接地址：https://redis.io/download

windows平台最好的 MicrosoftArchive 维护的版本：

目前最新的3.2.100

以 windows 平台下为例：

1、链接：https://github.com/MicrosoftArchive/redis/releases，下载对应版本的zip，解压缩版本

2、解压到 d:\redis

3、将 d:\redis 添加到 环境变量 path 中

4、打开 cmd 窗口，切换到 d:\redis 目录下，运行

redis-server.exe redis.windows.conf

这个 redis.windows.conf 就是相关的配置信息：

如：

bind 127.0.0.1

port 6379

protected-mode yes

等

5、可以查看到

Running in standalone mode

port：6379

PID : 13064

6、这个 cmd 窗口 就是启动服务的窗口，不要关闭

7、重新开启一个cmd窗口，运行

redis-cli.exe -h 127.0.0.1 -p 6379

就可以连接到 redis

# 配置

连接到 redis 命令窗口后，

可以通过 config get config\_key 查看，譬如：config get loglevel

可以通过 config set config\_key config\_value ，譬如：config set loglevel 'debug'

redis.windows.conf 配置项说明如下：

1. Redis默认不是以守护进程的方式运行，可以通过该配置项修改，使用yes启用守护进程

daemonize no

2. 当Redis以守护进程方式运行时，Redis默认会把pid写入/var/run/redis.pid文件，可以通过pidfile指定

pidfile /var/run/redis.pid

3. 指定Redis监听端口，默认端口为6379，作者选用6379作为默认端口，

因为6379在手机按键上MERZ对应的号码，而MERZ取自意大利歌女Alessia Merz的名字

port 6379

4. 绑定的主机地址，这个已经要注意，做测试都是绑定 127.0.0.1

bind 127.0.0.1

5.当 客户端闲置多长时间后关闭连接，如果指定为0，表示关闭该功能

timeout 300

6. 指定日志记录级别，Redis总共支持四个级别：debug、verbose、notice、warning，默认为verbose

loglevel verbose

7. 日志记录方式，默认为标准输出，如果配置Redis为守护进程方式运行，

而这里又配置为日志记录方式为标准输出，则日志将会发送给/dev/null

logfile stdout

8. 设置数据库的数量，默认数据库为0，可以使用SELECT <dbid>命令在连接上指定数据库id

databases 16

9. 指定在多长时间内，有多少次更新操作，就将数据同步到数据文件，可以多个条件配合

多个条件中，任意满足一个就会进行同步

save <seconds> <changes>

Redis默认配置文件中提供了三个条件：

save 900 1

save 300 10

save 60 10000

分别表示900秒（15分钟）内有1个更改，300秒（5分钟）内有10个更改以及60秒内有10000个更改。

10. 指定存储至本地数据库时是否压缩数据，默认为yes，Redis采用LZF压缩，

如果为了节省CPU时间，可以关闭该选项，但会导致数据库文件变的巨大

rdbcompression yes

11. 指定本地数据库文件名，默认值为dump.rdb

dbfilename dump.rdb

12. 指定本地数据库存放目录

dir ./

13. 设置当本机为slave服务时，设置master服务的IP地址及端口，在Redis启动时，它会自动从master进行数据同步

slaveof <masterip> <masterport>

14. 当master服务设置了密码保护时，slave服务连接master的密码

masterauth <master-password>

15. 设置Redis连接密码，如果配置了连接密码，

客户端在连接Redis时需要通过AUTH <password>命令提供密码，默认关闭

requirepass foobared

16. 设置同一时间最大客户端连接数，默认无限制，Redis可以同时打开的客户端连接数为Redis进程可以打开的最大文件描述符数，

如果设置 maxclients 0，表示不作限制。当客户端连接数到达限制时，

Redis会关闭新的连接并向客户端返回max number of clients reached错误信息

maxclients 128

17. 指定Redis最大内存限制，Redis在启动时会把数据加载到内存中，达到最大内存后，

Redis会先尝试清除已到期或即将到期的Key，当此方法处理 后，仍然到达最大内存设置，

将无法再进行写入操作，但仍然可以进行读取操作。

Redis新的vm机制，会把Key存放内存，Value会存放在swap区

maxmemory <bytes>

18. 指定是否在每次更新操作后进行日志记录，Redis在默认情况下是异步的把数据写入磁盘，

如果不开启，可能会在断电时导致一段时间内的数据丢失。

因为 redis本身同步数据文件是按上面save条件来同步的，

所以有的数据会在一段时间内只存在于内存中。默认为no

appendonly no

19. 指定更新日志文件名，默认为appendonly.aof

appendfilename appendonly.aof

20. 指定更新日志条件，共有3个可选值：

no：表示等操作系统进行数据缓存同步到磁盘（快）

always：表示每次更新操作后手动调用fsync()将数据写到磁盘（慢，安全）

everysec：表示每秒同步一次（折衷，默认值）

appendfsync everysec

21. 指定是否启用虚拟内存机制，默认值为no，简单的介绍一下，VM机制将数据分页存放，

由Redis将访问量较少的页即冷数据swap到磁盘上，访问多的页面由磁盘自动换出到内存中（

vm-enabled no

22. 虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享

vm-swap-file /tmp/redis.swap

23. 将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,

所有索引数据都是内存存储的(Redis的索引数据 就是keys),也就是说,

当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0

vm-max-memory 0

24. Redis swap文件分成了很多的page，一个对象可以保存在多个page上面，但一个page上不能被多个对象共享，vm-page-size是要根据存储的 数据大小来设定的，作者建议如果存储很多小对象，page大小最好设置为32或者64bytes；如果存储很大大对象，则可以使用更大的page，如果不 确定，就使用默认值

vm-page-size 32

25. 设置swap文件中的page数量，由于页表（一种表示页面空闲或使用的bitmap）是在放在内存中的，，在磁盘上每8个pages将消耗1byte的内存。

vm-pages 134217728

26. 设置访问swap文件的线程数,最好不要超过机器的核数,如果设置为0,那么所有对swap文件的操作都是串行的，可能会造成比较长时间的延迟。默认值为4

vm-max-threads 4

27. 设置在向客户端应答时，是否把较小的包合并为一个包发送，默认为开启

glueoutputbuf yes

28. 指定在超过一定的数量或者最大的元素超过某一临界值时，采用一种特殊的哈希算法

hash-max-zipmap-entries 64

hash-max-zipmap-value 512

29. 指定是否激活重置哈希，默认为开启（后面在介绍Redis的哈希算法时具体介绍）

activerehashing yes

30. 指定包含其它的配置文件，可以在同一主机上多个Redis实例之间使用同一份配置文件，

而同时各个实例又拥有自己的特定配置文件

include /path/to/local.conf

# 数据类型

Redis支持五种数据类型：string（字符串），hash（哈希），list（列表），set（集合）及zset(sorted set：有序集合)。

## string

字符串，单键最大512m

命令列表：

set name "test" # 设置 key

get name # 获取key

set name aaa nx # nx参数，name 这个key 不存在时，才能新增成功，否则失败，如果已经存在，那么不修改

set name aaa xx # xx参数，name 这个key 存在时，才能设置成功，否则失败，只有存在，才修改，不新增

set name aaa ex 10 # ex 是失效时间，单位 秒， 不能和 px 同时存在

set name aaa px 100 # px 是失效时间，单位 毫秒 ，不能和 ex同时存在

# incr 原子增量，多个客户端请求不会冲突

# 类似的 incrby ， decr， decrby

set counter 100

incr counter

incr counter

incrby counter 50

decr counter

# getset 设置新值并返回旧值

get name

getset name ccc

get name

# mget, mset 批量获取和修改

set age 10

mget name age

mset name 'ddd' age 20

# exists，检测键是否存在，返回 0和1

exists name

# del 删除key , 存在返回 1 ，不存在返回0

get age

del age

get age

# expire ，设置过期，单位 秒

set age 10

expire age 3

get age

# 完整的set , ex 和 px 不能同时设置

SET 键 值 [EX 秒] [PX 毫秒] [NX | XX]

set age 20 ex 10 nx

## hash

一个键值(key=>value)对集合，每个 hash 可以存储 2^32 -1 键值对（40多亿）

hmset hash\_test k1 "Hello" k2 "World" # 设置 hash 的 多个 key

hset hash\_test k3 100 # 设置单个key

hget hash\_test k1 # 获取单个 hash 的key

hget hash\_test k2

hmget hash\_test k1 k2 # 获取hash 的多个 key

hgetall # 获取所有value

# hincrby hash中的字段 增加

hincrby hash\_test k3 9

## list

string类型的链表（和python的list有区别），按照插入顺序排序。

可以添加一个元素到list的头部（左边）或者尾部（右边）

lpush list\_test one # 将元素添加到list的左边

lpush list\_test two

lpush list\_test three

lrange list\_test 0 10 # 获取列表的元素，下标从0开始，0 是起始位置（包含）， 10是结束位置（包含）

rpush list\_test four # 将元素添加到list的右边

lrange list\_test 0 10 # 获取一个list， 0 是起始位置（包含）， 10是结束位置（包含）

lrange list\_test 0 -1 # -1 表示最后

lpop list\_test # 从左边弹出元素，获取并删除

rpop list\_test # 从右边弹出元素

rpush mylist 1 2 3 4 5

ltrim mylist 0 2 # 删除给定范围之外的元素

# 等待列表中的元素tasks，但如果5秒后没有元素可用，则返回 ,设置为 0 秒，则永久等待 b: block

brpop tasks 5

blpop tasks 5

# llen 返回长度

llen tasks

# 不能更改已经存在key的类型，以下代码会出错

set foo bar

lpush foo 1 2 3 # 不能把一个 字符串 类型的key，修改为 list , 反之可以，从list可以通过set 修改为 字符串

# 弹出所有元素后，key会被删除

del foo

lpush foo 1 2

lpop foo

lpop foo

exists foo

## set

string类型元素的无序集合，最大的成员数为 2^32 - 1(40多亿)

sadd set\_test one # 在set 中增加 元素

sadd set\_test two three four

sadd set\_test three # 这条命令会被忽略，因为 set 中已经有 three 这个值存在了。

smembers set\_test # 获取set的所有值

sismember set\_test 3 # 检测 set 中是否存在某个 值

# sinter 不同的set之间的交集，2个集合中都存在的数据

sadd set\_test1 one 1 two 2

sinter set\_test set\_test1

# srandmember 随机获取一个元素，不会删除元素在集合中的存在，对比 spop

srandmember deck

# spop 弹出一个元素 ,会在集合中 删除该元素

spop set\_test1 [count]

# sunionstore 复制 set 的副本

sadd deck C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 CJ CQ CK D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 DJ DQ DK H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 HJ HQ HK S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 SJ SQ SK

sunionstore deck\_s deck

# SCARD 获取 set 中的元素数量

scard deck

## zset

string类型元素的有序集合，介于 hash和set 之间

每个元素都会关联一个double类型的分数，成员是唯一的,

但分数(score)却可以重复，根据分数从小到大排序

zadd zset\_test 0 one

zadd zset\_test 2 two

zadd zset\_test 1 three

zadd zset\_test 4 four

zadd zset\_test 2 five # 同样的分数，根据value的顺序排序，redis 2.8 之后新特性

zadd zset\_test 2 zero # 同样的分数，根据value的顺序排序，redis 2.8 之后新特性

zadd zset\_test 10001 six

zrangebyscore zset\_test 0 1000 # 根据分数范围查找元素, 不是下标！！！是分数！！

zrange zset\_test 0 1000 # 从小到大 获取元素

zrevrange zset\_test 0 -1 # 从大到小 获取元素

# withscores 连元素和分数一起返回

zrange zset\_test 0 -1 withscores

# zrangebyscore 获取分数小于等于这个值的所有元素

zrangebyscore zset\_test -inf 4

# zremrangebyscore 删除 分数范围之间的元素 ,并且返回删除的个数

zremrangebyscore zset\_test 1 3

# zrank 获取指定元素的位置 ，从 0 开始的

zrange zset\_test 0 -1

zrank zset\_test six

# 获取zset的长度

zcard zset\_test

## 其他

keys \* ： 查看所有的key

select num： 切换到指定数据库，默认是0：select 0

flushall ： 删除所有数据库

flushdb ： 删除当前数据库

# python操作redis

## 使用 redis 库

安装： pip install redis

当前最新版：2.10.6

官方文档：https://pypi.python.org/pypi/redis

## 连接：

# host是redis主机，需要redis服务端和客户端都启动 redis默认端口是6379

# decode\_responses=True，写入的键值对中的value为str类型，为 False 写入的则为字节类型，默认为False。

r = redis.Redis(host='localhost', port=6379, decode\_responses=True)

r.set('name', 'junxi') # key是"foo" value是"bar" 将键值对存入redis缓存

print(r['name'])

print(r.get('name')) # 取出键name对应的值

print(type(r.get('name')))

## 连接池

pool = redis.ConnectionPool(host='127.0.0.1', port=6379)

r = redis.Redis(connection\_pool=pool)

一般情况下，我们不需要指定连接池，Redis 会自动使用

## 管道技术

4.1 多行编写

pipe = r.pipeline(transaction=True)

pipe.set('name', 'python')

pipe.set('age', 22)

pipe.execute()

4.2 一行编写

pipe = r.pipeline()

pipe.set('foo', 'bar').sadd('faz', 'baz').incr('auto\_number').execute()

4.3 事务

with r.pipeline() as pipe:

while True:

try:

# 监控 OUR-SEQUENCE-KEY 密钥

pipe.watch('OUR-SEQUENCE-KEY')

# 监听后，管道会进行 立刻执行命令 模式，除非我们设置为缓存模式

# 这样允许我们获得序列的值

current\_value = pipe.get('OUR-SEQUENCE-KEY')

next\_value = int(current\_value) + 1

# 使用 multi 恢复到缓存模式

pipe.multi()

pipe.set('OUR-SEQUENCE-KEY', next\_value)

# 执行许多行动

# 最后执行 命令

pipe.execute()

# 一直执行到这里，所以命令全部实现

# 退出

break

except WatchError as e:

# 另一个客户端修改了 OUR-SEQUENCE-KEY 的值，

# 在我们监听之后，到最后 执行之前 这个时间段内

# 回滚，重新执行

continue