Redis

Redis(Re mote Di ctionary S erver)远程服务字典

Redis最新教程

Redis实战Golang

安装Redis

1. 首先有java环境，最好在CentOS系统下
2. 下载Redis文件，以及看看是否有 `make` 命令，没有则使用 `yum install gcc-c++`
3. 解压redis文件后进入输入 `make` 命令，得到一个 `src` 目录。进行 `make install`
4. 安装会默认安装再 `/usr/local/bin` 目录下

```
[root@localhost src]# cd /usr/local/bin/
[root@localhost bin]# ls
redis-benchmark  redis-check-rdb  redis-sentinel
redis-check-aof  redis-cli        redis-server
```

5. 将 `redis.conf` 复制到 `/usr/local/bin` 目录下，并且修改 `deamonize yes`
6. 通过指定配置文件启动redis服务。 `redis-server redis.conf`
7. 打开连接 `redis-cli -p 6379`
8. 查看redis进程 `ps -ef | grep redis`

性能测试

> redis-benchmark压力测试服务

```
-h <hostname>        Server hostname (default 127.0.0.1)
-p <port>            Server port (default 6379)
-s <socket>          Server socket (overrides host and port)
-a <password>        Password for Redis Auth
--user <username>  Used to send ACL style 'AUTH username pass'. Needs -a.
-c <clients>         Number of parallel connections (default 50)
-n <requests>        Total number of requests (default 100000)
-d <size>            Data size of SET/GET value in bytes (default 3)
--dbnum <db>         SELECT the specified db number (default 0)
--threads <num>     Enable multi-thread mode.
--cluster            Enable cluster mode.
--enable-tracking  Send CLIENT TRACKING on before starting benchmark.
-k <boolean>         1=keep alive 0=reconnect (default 1)
-r <keyspacelen>    Use random keys for SET/GET/INCR, random values for SADD,
                      random members and scores for ZADD.
 Using this option the benchmark will expand the string __rand_int__
 inside an argument with a 12 digits number in the specified range
 from 0 to keyspacelen-1. The substitution changes every time a command
 is executed. Default tests use this to hit random keys in the
```

```
 specified range.
-P <numreq>          Pipeline <numreq> requests. Default 1 (no pipeline).
-e                        If server replies with errors, show them on stdout.
                          (no more than 1 error per second is displayed)
-q                        Quiet. Just show query/sec values
--precision          Number of decimal places to display in latency output (default 0)
--csv                   Output in CSV format
-l                        Loop. Run the tests forever
-t <tests>           Only run the comma separated list of tests. The test
                          names are the same as the ones produced as output.
-I                        Idle mode. Just open N idle connections and wait.
--help                  Output this help and exit.
--version            Output version and exit.
```

比如：

```
redis-benchmark -h localhost -p 6379 -c 100 -n 50000
```

基础知识

- redis默认有16个数据库，使用`select [index]`选择第index个数据库。
- `DBSIZE` 查看数据库的大小
- `keys *` 查看所有的键值
- `flushdb` 清空当前数据库
- `flushall` 清空所有的数据库
- Redis是单线程的，Redis是基于内存操作的，Redis的瓶颈是根据机器的内存和网络带宽据欸的那个的。
- 为什么redis这么快？redis是将所有的数据全部放在内存中，所以使用单线程操作效率最高，不需要上下文切换。

五大数据类型

Redis可以用作数据库、缓存和消息中间件。

Redis-key

- `set key value [EX seconds | PX millonseconds]`
- `get key`
- `move key [index]` 将某个键值移动到第n的数据库
- `EXISTS key` 查看某个值是否存在
- `EXPIRE Key seconds` 让key值在n秒内消失。`ttl key`查看剩余的秒数
- `type key` 查看key值的类型

String类型

- `APPEND key value` 在已有key值的情况下，连接字符串value；如果key不存在相当于创建字符串
- `STRLEN key` 获得某个key的字符串长度
- `GETRANGE key start end` 相当于substring。但是范围为`[start, end]`
- `SETRANGE key start value` 替换字符串从start位置开始替换目标字符串的长度。
- `setnx key value` 如果key值不存在，则设置为value返回为1；存在则创建失败且返回为0。

- `mset key1 value1 key2 value2 ...` 批量设置key-value
- `mget key1 key2 key3` 批量获取
- `msetnx key1 value1 key2 value2 ...` 批量设置，原子操作，一个失败则全部失败
- key值可以巧妙设计：`set user:{id}:{name} value`
- `getset key value` 先get在set，返回get后的值。类似于CAS(compare And Swap)操作

对于数值类型：

- `INCR key` 相当于key++ `DECR key` 相当于key--
- `INCRBY key steps` 相当于key+=steps `DECRBY key steps` 相当于key-=steps

List类型

> 可以将list实现栈或者队列

`LPUSH` 和 `RPUSH`，放到list的第一个和最后一个

`LPOP` 和 `RPOP`，移除list的第一个和最后一个元素

`LRANGE list start end` 获取list中的值[start, end]，获取所有 `LRANGE list 0 -1`

`LINDEX list [index]` 获取list指定索引下标的元素

`LLEN list` 获取list列表长度

`LREM list count value` 删除list中指定个数的值

`LTRIM list start end` 截取原来数组指定index的元素，`[start end]` 闭区间

`RPOPLPUSH old new` 从原来列表中移除最后一个元素并放入新的列表中

`LSET list index value` 指定对应索引的值，必须对应索引存在

`LINSERT list [before|after] pivot value` 在list中某个pivot前后插入对应的值。

Set类型

不可重复

`sadd myset value`

`SMEMBERS set` 获取set内的值

`SISMEMBER set value` 查看value是否在set集合内

`SREM set value` 移除

`SCARD SET` 查看set的个数

`SPOP` 随机删除一个元素 `SRANDMEMBER set [count]` 随机选取n个元素

`SMOVE source dest element` 移动元素

集合操作

差集：`SDIFF SET1 SET2`

交集：`SINTER SET1 SET2` 找共同好友或者共同关注之类的

并集：`SUNION SET1 SET2`

Hash 类型

类似于存储js中的对象类型。

在redis类似于 `string` 类型

```
{js:{field1:value,field2:value2}}
```

`hset key field value` 设置值

`hget key field`

`hmset` `hmget` 批量设置和获取。 `hgetall` 获取所有键值对

`hdel obj field` 删除某个属性

`HLEN obj` 获取hash的长度

`HEXISTS obj field` 某个属性是否存在

`HKEYS` `HVALS` 获取所有的键或值

`HINCR` `HDECR` `HSETNX`

Zset

在set基础上进行排序。按照key来进行排序

`zadd myset [score] [values]`，可以根据score的大小进行排序

`zrange myset min max [withscores]`，在min和max区间的分数进行排序数据

`zrevrange myset 0 -1`，降序输出集合

`zcard myset`，获取集合中的元素个数。

`zcount myset min max`，获取指定区间分数的元素个数。

三种特殊数据类型

geospatial

> 定位问题，可以推算地理位置，两地距离等。可以用于找附近的人，底层原理结构就是zset

`geoadd china:city 116.40 39.90 beijing`，添加地理位置

`geopos china:city beijing shanghai`，获取某地理位置的经度纬度

`geodist china:city beijing shanghai [m | km]`，获取两个位置之间的直线距离

`georadius china:city longtitude latitude [dist] [m|km] count n` 获取某个经纬度dist距离的前n个

`georadiusbymember china:city [dist] [m|km] count n` 获取距离某个城市dist距离的前n个数据

hyperloglog

> 统计基数的算法，占用内存很小，允许容错。

2^{64}不同元素，只需要12KB的内存，0.81%的误差。

```
127.0.0.1:6379[1]> pfadd s1 a b c d e f     # 添加元素
(integer) 1
127.0.0.1:6379[1]> pfcount s1               # 获取元素数目
(integer) 6
127.0.0.1:6379[1]> pfadd s2 z x c v b
(integer) 1
127.0.0.1:6379[1]> pfcount s2
(integer) 5
127.0.0.1:6379[1]> pfmerge s1 s2            # 合并s1，s2到s1
OK
127.0.0.1:6379[1]> pfcount s1               # 查看数目
(integer) 9
```

Bitmap位存储类型

> 统计用活跃不活跃、登陆未登录，只有1和0两种状态。

`setbit sign bit value`

`getbit sign bit`

统计操作：

`bitcount key` 记录key中1的个数

Redis 的事务

开启事务

Redis的单条命令是原子性的，但是事务不保证原子性，也没有隔离级别的概念。

redis事务的三个阶段：

- 开始事务（`multi`）
- 命令入队
- 执行事务（`exec`）或者放弃事务（`discard`）

```
127.0.0.1:6379> MULTI   # 开启事务
OK
127.0.0.1:6379(TX)> set key1 v1
QUEUED
127.0.0.1:6379(TX)> set k2 v2
QUEUED
127.0.0.1:6379(TX)> exec # 执行事务
1) OK
2) OK
```

如果事务执行过程中出现错误：

- 如果出现语法命令型错误会直接放弃事务
- 运行时错误：其他正确的命令不影响

锁

加锁：`watch`，解锁：`unwatch`

悲观锁：

乐观锁：使用watch可以当作redis的乐观锁操作。

如果乐观锁执行失败，则先解锁再加锁，继续进行操作。

```
127.0.0.1:6379[1]> unwatch
OK
127.0.0.1:6379[1]> watch money
OK
127.0.0.1:6379[1]> multi
OK
127.0.0.1:6379[1](TX)> decrby money 20
QUEUED
127.0.0.1:6379[1](TX)> incrby out 20
QUEUED
127.0.0.1:6379[1](TX)> exec         # 如果事务失败会返回nil
1) (integer) 60
2) (integer) 40
127.0.0.1:6379[1]> unwatch              # 事务完毕，解锁
OK
```

Redis.conf详解

```
bind 127.0.0.1          # 绑定IP
protect-mode yes        # 保护模式
port 6379               # 端口
# ========================
daemonize yes           # 默认是no，需要我们自己手动开启
pidfile /var/runredis_6379.pid  # 如果是后台运行，则需要指定一个pid文件
loglevel notice         # 日志级别
logfile ""              # 为输出文件，默认为标准输出
database 16             # 默认的数据库数量
# =========SNAPSHOT=========
# 持久化，再规定时间内执行了多少次操作就会持久化到文件.rdb .aof
# redis是内存数据库，如果不持久化断电就会数据小时
save 900 1      # 如果900s内至少有1次对key进行了修改则进行持久化
save 300 10     # ...300s...10次

stop-writes-on-bgsave-error yes     # 持久化之后是否进行工作

rdbcompression yes      # 是否对rdb文件进行压缩
rdbchecksum yes         # 保存rdb文件，进行错误验证
dir ./                  # rdb文件保存目录

# ==========REPLICATION========
replicaof 127.0.0.1 6379    # 设置主从复制的主机信息
masterauth passwd           # 设置主机的密码
```

```
# ==========SECURITY===========
requirepass foobared        # 设置redis密码，使用auth登录

# ==========Clients============
maxclients 10000            # 设置redis可连接的最多客户端数目
maxmemory <bytes>           # 设置redis最大内存容量
maxmemory-policy noeviction # 设置redis内存满之后的操作

# ============AOF============
appendonly no               # 默认不开启aof模式，默认使用rdb方式持久化
# The name of the append only file (default: "appendonly.aof")
appendfilename "appendonly.aof"
# appendfsync always        # 每次修改都进行同步，但消耗性能
appendfsync everysec        # 每秒同步一次，可能丢失1s的数据
# appendfsync no            # 不执行sync，操作系统自己同步，速度最快
```

Redis持久化

RDB（Redis Database）

触发持久化的条件：

- 配置文件中的`save`
- `flushall`和`exit`命令

查看持久化文件目录：`config get dir`

特点：

- 适合大规模数据恢复
- 对数据完整性不高
- 需要间隔一段时间进程操作。如果redis宕机，最后一次修改的数据就没有了。
- fork进程的时候会占用内存空间。

AOF（Append Only File）

> 将所有的命令全部记录下来（读操作除外）

默认不开启，需要再redis.conf中进行手动配置。

如果appendonly.aof被擅自修改，使用`redis-check-aof --fix appendonly.aof`来进行修复。

特点：

- 每一次修改都同步`appendfsync everysec`
- 相对于数据文件来说，aof要比rdb文件大，修复速度慢
- 运行效率比rdb低

Redis订阅发布

接收端(`subscribe`)：

```
127.0.0.1:6379> subscribe kss 2mw       # 订阅频道
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "kss"
3) (integer) 1
1) "subscribe"
2) "2mw"
3) (integer) 2
1) "message"
2) "kss"
3) "aoligei"
1) "message"
2) "2mw"
3) "moslian"
```

发送端(`publish`):

```
127.0.0.1:6379> publish kss aoligei     # 发布消息
(integer) 1
127.0.0.1:6379> publish 2mw moslian
(integer) 1
```

Redis主从复制

概述

将一台redis服务器的数据，复制到其他的服务器节点。前者是主节点(master)，后者是从节点(slave)，数据是单向的，只能从主节点到从节点。Master以写为主，Slave以读为主，主从复制，读写分离。单台Redis最大使用内存不应该超过20G。

🔘查看配置：`info replication`，默认为主节点。

```
# Replication
role:master
connected_slaves:0      # 没有从机
master_failover_state:no-failover
master_replid:f02fb8eff5f4e69a6759cc4cef25b02d920de596
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:0
second_repl_offset:-1
repl_backlog_active:0
repl_backlog_size:1048576
repl_backlog_first_byte_offset:0
repl_backlog_histlen:0
```

🔘环境配置：

使用命令配置：

复制并且修改`redis.conf`文件，需要修改的信息比如`port` `log` `dumpfile` `pid`等

如何配置从机：`slaveof host port`去找主人即可。

如何由从机变到主机：`slaveof no one`

使用配置文件配置：

```
# ==========REPLICATION========
replicaof 127.0.0.1 6379    # 设置主从复制的主机信息
masterauth passwd           # 设置主机的密码
```

◍复制原理：

从机不能进行写信息。主机宕机之后数据信息在从机中仍有保存。

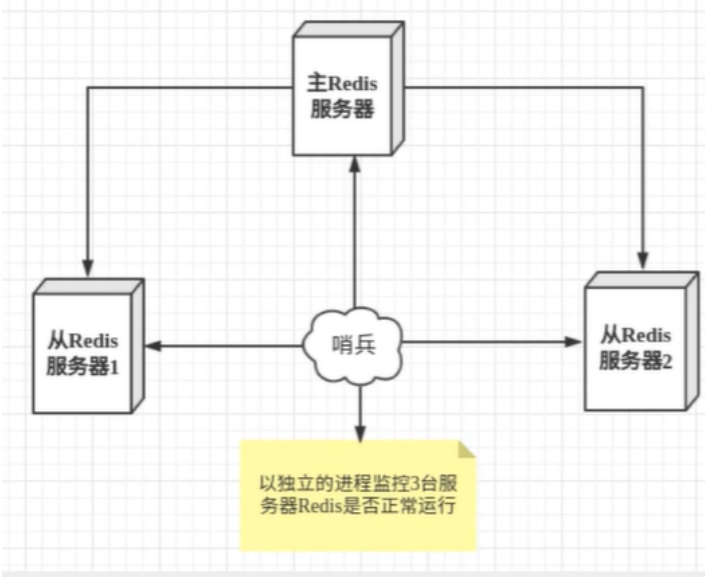
全量复制：从机第一次连接，主机就会将内存文件全部发送给从机。重新连接，数据仍会同步。

增量复制：主机数据发生变化的时候，从机进行同步主机的信息。

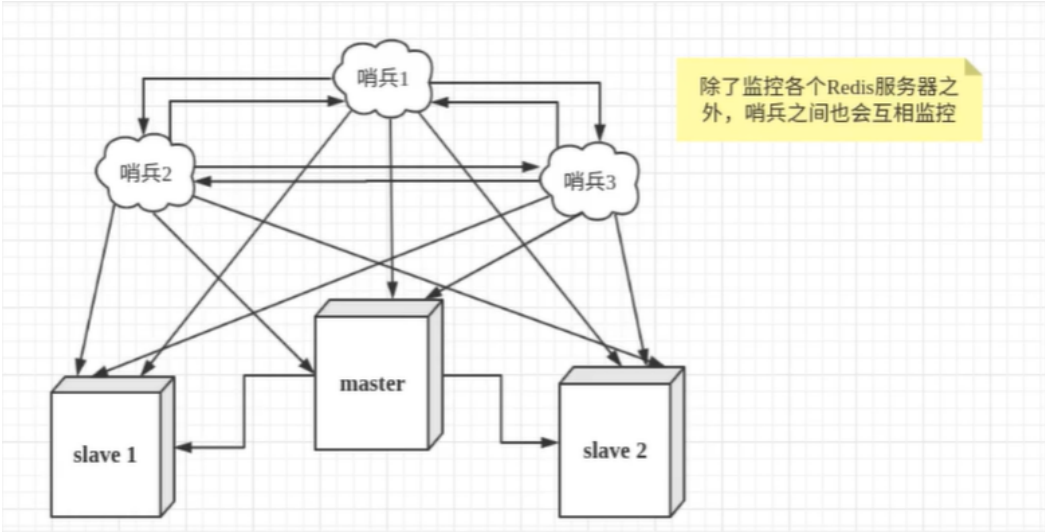
哨兵模式◈

> 自动选主机的模式，如果主机宕机，从机自动变主机

单哨兵模型：



多哨兵模型：



◍哨兵配置：

比如创建一个`sentinel.conf`

```
port 26379
dir /tmp        # 对应工作目录
# 监控主机，后面的1代表，当有1个哨兵认为主机连接不上之后，就认为客观死亡了。
sentinel monitor redis01 127.0.0.1 6379 1
sentinel auth-pass redis01 passwd       # 设置机器的密码
```

开启一个哨兵：

```
redis-sentinel sentinel.conf
```

如果宕机的主机重新恢复后，哨兵会自动重新将其转换为主机，其他转换为从机。

Redis缓存穿透、缓存击穿和雪崩

> 服务器的高可用问题

缓存穿透

> 大面积未命中，redis未命中，可能数据库中也不存在记录，可能为黑客攻击。

我们使用Redis大部分情况都是通过Key查询对应的值，假如发送的请求传进来的key是不存在Redis中的，那么就查不到缓存，查不到缓存就会去数据库查询。假如有大量这样的请求，这些请求像“穿透”了缓存一样直接打在数据库上，这种现象就叫做缓存穿透。

解决方法：

- 对在数据库中查询结果为空的数据，在redis进行缓存为null
- 白名单策略：布隆过滤器
- key加密

缓存击穿

> 某个key为热点，不停扛着高并发，在key失效瞬间，会击穿缓存，直接访问数据库。

Redis无大量的key过期，服务器平稳运行，数据库崩溃。redis某个key过期，而对于此key的访问激增，Redis数据库均未命中。相当于对某个点集中打击。

问题分析：单个key为高热数据，key过期

解决方案：

- 预先处理：加大key的时间
- 现场调整：手动延长过期时间，或者设置为永久key
- 后台刷新数据：启动定时任务，高峰期来临时候，刷新数据有效期
- 设置多级缓存：不同时淘汰即可。
- 加锁：分布式锁，慎用。

缓存雪崩

> key集中过期，服务器宕机、断网等等

系统在平稳运行的过程中，忽然数据库的访问量激增，应用服务器无法及时响应请求，大量408，500页面。数据库崩溃，应用服务器崩溃，Redis服务器崩溃，重启数据库之后再次被流量放倒。

问题排查：

- 较短时间内，缓存较多的key集中过期。短时间内访问过期数据，直接向数据库请求数据。

解决方案：

- 构建多级缓存：Nginx缓存+redis缓存+ehcache缓存
- 检查数据库查询是否超时
- 限流+降级：短时间范围内牺牲用户的体验，限制访问请求数量，降低压力。
- LRU和LFU切换使用
- 根据业务数据进行分类策略：A类90分钟，B类80分钟等
- 过期时间采用固定时间+随机值的形式，稀释集中过期的key数量
- 超热数据使用永久的key+定期维护
- 加锁