

# Redis

## 1. Redis-Cluster

- 集群是采用 **hash桶** 即 **slot** 的方式对数据进行分发存储的
  - 集群总共slot数量是 **16384** 个，**Redis** 会对 **key** 进行 **CRC16 MOD 16384** 计算得到 **slot** 编号，再存入到对应的 **slot** 中；
  - 集群又会把 **16384** 平分到每个节点上（**16384/N**）；

## 2. Redis

### 2.1 命令

#### 2.1.1 启动server

```
./redis-server /path/to/redis.conf
```

#### 2.1.2 启动client

```
redis-cli.exe -h 127.0.0.1 -p 6379
```

### 2.2 设置

#### 2.2.1 redis.conf文件

##### 2.2.1.1 内存大小

不区分大小写

```
# 1k => 1000 bytes
# 1kb => 1024 bytes
# 1m => 1000000 bytes
# 1mb => 1024*1024 bytes
# 1g => 1000000000 bytes
# 1gb => 1024*1024*1024 bytes
```

##### 2.2.1.2 文件引入

可以在主配置文件 **redis.conf** 中引入其他配置文件：

```
# include /path/to/local.conf
# include /path/to/other.conf
```

注意：这种配置下，redis默认会按列出的顺序，后面文件覆盖前面文件的配置；

##### 2.2.1.3 daemon

默认 **daemonize no**

如果此项设置为 **yes**，redis运行时，默认会生成 **/var/run/redis.pid** 文件；

#### 2.2.1.4 pidfile

如果启动daemon模式，默认生成 `/var/run/redis.pid` 文件，可通过 `pidfile` 指令修改：`pidfile /var/run/redis.pid`

#### 2.2.1.5 端口

```
port 6379
```

#### 2.2.1.6 backlog

设置已经建立的TCP连接的最大数量：`tcp-backlog 511`；

注意：`/proc/sys/net/ipv4/tcp_max_syn_backlog` 和 `/proc/sys/net/core/somaxconn` 都将影响到最大TCP连接数；

关于backlog，请移步[博客](#)

#### 2.2.1.7 客户端IP过滤

```
bind 192.168.1.100 10.0.0.1
```

#### 2.2.1.8 UnixSocket

如果没有外网使用的需求，我们还可以让 Redis 以 Unix Socket 的方式运行，以避免 TCP/IP 的性能瓶颈，在高访问场景实现 25% 的性能提升

```
# unixsocket /tmp/redis.sock
# unixsocketperm 700
```

具体设置使用方式自行搜索

#### 2.2.1.9 timeout

设置client失活多久后断开连接，单位秒（S），0表示设置无效：

```
timeout 0
```

#### 2.2.1.10 日志级别

```
loglevel notice
```

```
# debug (a lot of information, useful for development/testing)
# verbose (many rarely useful info, but not a mess like the debug level)
# notice (moderately verbose, what you want in production probably)
# warning (only very important / critical messages are logged)
```

#### 2.2.1.11 日志文件

```
logfile ""
```

- 空字符表示强制输出日志到 `standard output`
- Note that if you use standard output for logging but daemonize, logs will be sent to `/dev/null`

#### 2.2.1.12 syslog

- 打开syslog  
`syslog-enable yes` : 使用 `system logger`
- syslog identity  
`syslog-ident redis`

#### 2.2.1.13 多库

`databases 16`

[知识扩展](#)

#### 2.2.1.14 保存到disk

详细解释请查阅 `redis.conf` 文件

```
save 900 1
save 300 10
save 60 10000
```

#### 2.2.1.15 定时频率

查看 `hz` 配置属性:

#### 2.2.1.16 一致性延迟

查看 `repl-disable-tcp-nodelay no` 设置:

#### 2.2.1.17 复制积压缓冲区

查看 `repl-backlog-size` 配置:

#### 2.2.1.18 主从心跳检测

查看 `repl-ping-slave-period` 配置:

## 2.3 数据类型

### 2.3.1 Set

- set类型会根据score进行排序, 如果score相同, 会进行字典排序:

更多

TODO

1. `redis.conf` 配置文件还有很多配置模块 ( `SNAPSHOTTING`、`REPLICATION`、`SECURITY`、`APPEND ONLY MODE`、`REDIS CLUSTER`、`SLOW LOG`、`LATENCY MONITOR`、`EVENT NOTIFICATION`、`ADVANCED CONFIG` ), 太多了, 以后再搞吧!
2. redis可以通过Lua脚本实现原子性;
3. 可以了解一下[client-side-caching](#), 可能对解决热点key问题有所帮助
4. 聊聊Linux IO

## 遗留疑问

---

1. Hash类型的value的属性数量较少时，编码类型是zipmap，这个zipmap具体内部原理是啥？
2. HyperLog  
[走近源码：神奇的HyperLogLog](#)
3. GeoHash  
GeoHash貌似有点问题呀
4. Stream

## 参考

---

[redis原理 详解](#) |  
[redis为何采用单线程？](#) |  
[LFU淘汰策略](#) |  
[Redis中的LFU算法-详细版](#) |  
[LFU淘汰策略2](#) |  
[Redis持久化磁盘IO方式及其带来的问题](#) |  
[AOF重写](#) |  
[勿用redis的多摩](#) |  
[Redis 的多线程版本\(keyDB\)比 Redis 本身要快 5 倍](#) |  
[rehash实现](#) |  
[Redis基本数据类型](#) |  
[Redis内部数据结构详解\(1\)——dict](#) |  
[Redis内部数据结构详解\(7\)——intset](#) |  
[Redis持久化磁盘IO方式及其带来的问题](#) |  
[选择合适Redis数据结构，减少80%的内存占用](#) |  
[Redis内部数据结构详解\(5\)——quicklist](#) |  
[关闭持久化时，复制的安全性](#) |  
[Redis踩坑1](#) |  
[Redis数据结构（汇总）](#) |  
[Redis的内存淘汰策略](#) |  
[Redis热点Key发现及常见解决方案](#) |  
[如何快速定位 Redis 热 key](#) |  
[redis数据库结构](#) |  
[embstr为何是39或44](#) |  
[redis数据结构\(二\) - 字符串](#) |  
[Redis 中的数据持久化策略（AOF）](#) |  
[Redis数据持久化之RDB-AOF混合方式](#) |  
[主从复制](#) |  
[Redis 集群实现原理探讨](#) |  
[关于Redlock应该注意的点](#) |  
[对SDSHDR5是否使用的疑问](#) |  
[hash一致性-hash环](#)

如果master节点写并发很高，复制挤压缓冲区又设置的比较小的话，可能会每次向slave同步完数据以后，每次复制挤压缓冲区都会溢出，造成主从之间循环的全量复制。这确实是应该规避的问题！ [参考](#)