



作者 董泽润 (/users/fc79f7b7a47b) 2015.10.13 14:46*

写了8655字，被25人关注，获得了15个喜欢
(/users/fc79f7b7a47b)

write smart proxy step by step 1 (理论简介)

字数1673 阅读188 评论0 喜欢3



为什么写这系列？

前段时间《7月，redis迷情 (<http://www.jianshu.com/p/9fdb1aece269>)》里面提到，我司线上 Redis Cluster + Smart Proxy 模式。我把代码放到了个人 github (<https://github.com/dongzerun/smartproxy>)上，大家感兴趣的可以下载使用。如果遇到问题，随时开 Issue，或是直接找我。支持命令有限制，README (<https://github.com/dongzerun/smartproxy>)里有详细介绍。

```
# go get github.com/dongzerun/smartproxy  
  
# make && ./cmd/redis_proxy -config_file=example.ini
```

为什么要重写呢，有如下原因：



1. 当前实现需要对输入做字符串转化，大家知道在Golang中，大量 `Byte[] To String` 性能不高且对GC压力较大。可以参考 Bitly 对 NSQ 的优化官方文档。
2. 复用 Go Redis Driver，封装网络层，导致存在无效处理流程，重写后可以从底层优化。
3. Redis 在数据库领域里相当小巧，开发中间件有助于对数据库的理解。
4. 按模块开发，附以 Go 的单元测试，加深对 Go 语言的认知和性能调优。

新的 Proxy 目标

老的代理当前使用上没问题，性能也足够用。新的要实现如下功能

1. 支持 Redis Cluster 协议，屏蔽语言的差异，这是核心。
2. 命令检测，对危险或是不支持的命令提前检测。
3. 支持 Pipeline，这个使用场景还是很多，老的代理无法实现。
4. 更详细的性能统计，内建 Http DashBoard 以供查询。
5. 服务发现与注册，支持 Zookeeper 和 Etcd。

Redis Client Protocol

Redis 协议 (<http://redis.io/topics/protocol>) 相比 MySQL 协议 (<http://dev.mysql.com/doc/internals/en/client-server-protocol.html>) 简单太多，感兴趣的同学可以查看相应驱动源码，加深对协议的理解。Redis 属于文本协议，基于 Request-Response Model 模型，官方称做 RESP (REdis Serialization Protocol)。

Response Protol

RESP 协议有五种应答类型，每部分以 CRLF(\r\n) 结尾

1. Simple Strings: 简单字符应答以 "+" 开头，跟随字符串数据，并以 CRLF 结尾

```
[root@... dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*3
$3
SET
$6
FOOF00
$3
BAR
+OK
```

Simple Strings Reply

例中返回数据为+OK\\r\\n

2. Errors: 错误应答以"-"开头，跟随错误原因的字符串，并以 CRLF 结尾

```
[root@... dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*3
$3
get
$1
a
$1
a
-ERR wrong number of arguments for 'get' command
```

Errors Reply

例中返回数据为-ERR wrong number of arguments for 'get' command\\r\\n

3. Integers: 整数应答以":"开头，跟随整数值，并以 CRLF 结尾

```
[root@192.168.100.100 dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*2
$4
INCR
$3
aaa
:2
```

Integers Reply

例中返回数据为:2\r\n

4. Bulk Strings: 字符串应答以"\$"开头, 跟随真实数据的长度和数据

```
[root@192.168.100.100 dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*2
$3
get
$6
FOOF00
$3
BAR
```

Bulk String Reply

例中返回数据为:\$3\r\nBAR\r\n

5. Arrays: 数组类型应答以"*"开头, 跟随多个 Bulk Strings的应答

```
[root@LJ-OptiPlex-5090-100 dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*5
$4
mget
$1
a
$1
b
$1
c
$1
p
*4
$1
b
$1
a
$1
d
$-1
```

Array Reply

例中返回数据为 *4\r\n\$1\r\nnb\r\n\$1\r\n\$a\r\n\$1\r\nnd\r\n\$-1\r\n, 其中最后一个-1表示对应 command 中最后一个 key 不存在

Command Protol

举个最简单的命令 SET FOOFOO BAR, 那么客户端实际发送给 Server 的数据为

```
*3\r\n$3\r\nSET\r\n$6\r\nFOOFOO\r\n$3\r\nBAR\r\n
```

使用 telnet 模拟如下

```
[root@192.168.1.100 dongzerun]# telnet localhost 6379
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*3
$3
SET
$6
FOOF00
$3
BAR
+OK
```

set foofoo bar

最后的+OK表示命令执行成功，反回 simple string OK，也就是说 Command 行为和 Array Reply 一致。

Redis Cluster 简介

官网有两篇文章 cluster-spec (<http://redis.io/topics/cluster-spec>) 和 cluster-tutorial (<http://redis.io/topics/cluster-tutorial>)，详细讲解了 Redis Cluster 实现原理和搭建，湖南 TV 提供了 Python 版的 Redis-trib (<https://github.com/HunanTV/redis-trib.py>)，大家可以使用搭建和迁移 Cluster 数据很方便。如何搭建和 Cluster 内部选举算法这里不做讲解，需要注意 cluster-require-full-coverage 这个参数要设为 no

If this is set to yes, as it is by default, the cluster stops accepting writes if some percentage of the key space is not covered by any node. If the option is set to no, the cluster will still serve queries even if only requests about a subset of keys can be processed.

简单来说，Cluster 将 Key(或是根据 hash tag) 按照 CRC16 处理得到 hash value，对值按 16384 取模，这样会将数据均匀分到 16384 slots 中，属于逻辑槽，逻辑 slots 再根据 Cluster-nodes.conf 确定具体属于哪个物理主从对

```
[root@192.168.1.100 redis]# cat nodes-6494.conf | grep -i master | grep -v dddd
092452f9586dcf747a9f1a9e9ee2ec618ea44682 10.10.10.169:6494 master - 0 1445540692109 12 connected 8194-10923
c3113759d12b04fe3e45fdc89fe649de68dd2aac 10.10.10.92:6494 master - 0 1445540693008 3 connected 13654-16383
d087ff0a1e34d2618b3a1c9ed00ea06e8b6893f0 10.10.10.170:6494 master - 0 1445540687506 7 connected 5464-8193
94239ed02d1bcf2bdf17a8f99155b21099d0f249 10.10.10.112:6494 master - 0 1445540689006 11 connected 0-2733
f56700d77cf40a471cfbc1199b6516ce12a9ff04 10.10.10.94:6494 master - 0 1445540692009 9 connected 10924-13653
ae1aa7118aca4e5086ec26e88a53f50ed9413a0e 10.10.10.178:6494 master - 0 1445540691008 1 connected 2734-5463
```

cluster-nodes.conf

```
127.0.0.1:6494> cluster info
```

```
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:14
cluster_size:6
cluster_current_epoch:12
cluster_my_epoch:7
cluster_stats_messages_sent:14875447
cluster_stats_messages_received:14875364
```

cluster info

使用 Cluster Info 命令可以看到当前集群信息，`cluster_slots_assigned` 表示当前已分配物理节点的 slots 数量，`cluster_slots_ok` 表示当前有多少个 slots 正常服务，`fail` 和 `pfail` 分别表示当前节点谁为主观宕机和客观宕机的 slots 数量，`cluster_size` 表示集群有多少个物理主从对，对参数更详细的请参考官方文档。

由于采用无中心的设计，客户端随机连接一个节点，发起请求。Redis 根据 Key 获取 hash 值后取模，得到 Slot ID，在内存中查找 Slot ID 当前属于哪个物理节点。如果不是本机，那么 Redis 返回客户端一个 Errors Reply，错误内容是一个 MOVE 重定向，例如：

```
127.0.0.1:6494> get b
```

```
(error) MOVED 3300 10.10.10.178:6494
```

此时客户端根据 MOVED 指令，去连接 10.10.10.178:6494 实例，发起同样的请求，得到正确的数据。理想情况下，Smart 客户端只需 MOVED 跳转一次即可找到正确节点。

当 Slot 发生迁移时，Redis 会在源节点标记该 Slot 为 Migrating 状态，同时标记目标节点该 Slot 为 Importing 状态。当客户端发起请求时，所有流量都会转发到源节点，如果数据存在那么返回。如果数据不存在，那么可能已经迁移到目标节点。源节点返回一个 -ASK 错误附加目标节点信息，客户端连接目标节点，先发起一个 ASKING 命令，再发送正常的请求。如果不先发送 ASKING 命令，那么目标节点会认为是非法请求，返回 MOVED 重定向。

Smart Proxy 设计思路

明白了 Redis Cluster 原理，那么核心设计起来也就简单多了。其它功能都是添枝加叶。具体模块划分留到下一篇分享吧，每篇编写一个模块，并附件详细的 Go Test 和性能测试。

1. 定期获取 Cluster Config 配置信息，缓存到 Proxy 内存中。
2. 对请求 Key 取 Hash Value 后，查询 Slots 配置，发送到指定后端节点。
3. 处理好 MOVED 和 ASK 两个异常即可。

结语

这篇只讲些理论基础，后续再具体编码，希望自己能坚持写完~_~ 推荐大家一首歌《压缩饼干 (<http://y.qq.com/#type=song&mid=0044Y8UL0zFkXW&from=smartbox>)》来自大冰

● 推荐拓展阅读

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

本文已收到 0 次打赏

喜欢

分享到微博 分享到微信
更多分享 ▾

写下你的评论...

发表

😊⌘+Return 发表