



<http://www.onexsoft.com>

## ➤ OneCache — 针对 Redis 的代理服务

### ✧ 功能特性

- **高并发性**
  1. 测试时使用了多个 redis-benchmark 对同一个代理进行测试，每秒并发量可达 30w/s
- **完整的 Redis 命令支持**
  1. 覆盖常用的大部分命令，包括（ MGET,MSET ）这种多值操作
  2. 自带内置命令。例如查看当前监控的状态信息、查看 OneCache 命令支持 等等
- **通过代理的方式减少缓存服务器的连接数**
  1. 保持与 Redis 的长连接
  2. 连接可复用性
  3. 可设置代理与每个 Redis 的连接数目
- **RedisServerGroup 模型**
  1. 每个 Group 可以有不同的策略模式
  2. Group 内可配置若干 Master 和 Slave
  3. 每个 Group 可以动态配置散列范围和指定的 KEY
- **自动分片到后端多个 Redis 实例上**
  1. 使用散列方式将数据以 Key 的值进行散列化
  2. 可以控制后端实例的权重
- **详细的状态监控**
  1. 实时代理运行状态
  2. 实时后端节点运行状态
  3. 实时客户端连接状态
  4. TopKey 统计
- **HA 机制**
  1. 可部署多个 OneCache 以防止单台主机故障
  2. 轻量级，使用 VIP 方式实现

## ✧ 性能测试

### 测试环境描述

机器类型：物理机(16 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz 32G 内存 千兆网卡)

对比产品：Twemproxy

性能测试工具：redis-benchmark

注：Twemproxy 和 OneCache 都运行在同一台机器上(172.30.12.12)

### 指标测试 1

单个 redis-benchmark 进行测试，对比经过代理和直连的差异性

### Twemproxy 的启动配置

```
[root@localhost twemproxy]# cat nutcracker.yml
alpha:
  listen: 0.0.0.0:22121
  hash: fnv1a_64
  distribution: ketama
  auto_eject_hosts: true
  redis: true
  server_retry_timeout: 2000
  server_failure_limit: 1
  servers:
    - 172.30.12.8:6379:1
[root@localhost twemproxy]# ./nutcracker -c nutcracker.yml -v 0
[2015-09-02 00:16:57.224] nc.c:187 nutcracker-0.4.1 built for Linux 2.6.32-504.16.2.el6.x86_64 x86_64 started on pid 20020
[2015-09-02 00:16:57.224] nc.c:192 run, rabbit run / dig that hole, forget the sun / and when at last the work is done / don't sit down / it's time to dig another one
```

### OneCache 的启动配置

```
[root@localhost onecache]# cat 1.xml
<onecache port="8221" thread_num="8" hash_value_max="80">
  <group name="group1" hash_min="0" hash_max="80">
    <host host_name="host1" ip="172.30.12.8" port="6379" master="1"></host>
  </group>
</onecache>
[root@localhost onecache]# ./onecache 1.xml
[00:19:33] Message: Load config file '1.xml'...
[00:19:33] Message: Create connection pool (172.30.12.8:6379)...
[00:19:33] Message: Create the thread pool...
[00:19:33] Message: Thread pool created. size: 8
[00:19:33] Message: OneCache has running. Port: 8221
```

### 直连 Redis

```
[root@localhost redis]# ./redis-benchmark -h 172.30.12.8 -t get,set -n 1000000 -r 100 -q
SET: 94616.33 requests per second
GET: 110436.22 requests per second
```

### 通过 Twemproxy

```
[root@localhost redis]# ./redis-benchmark -h 172.30.12.12 -p 22121 -t get,set -n 1000000 -r 100 -q
SET: 84774.50 requests per second
GET: 81606.01 requests per second
```

### 通过 OneCache

```
[root@localhost redis]# ./redis-benchmark -h 172.30.12.12 -p 8221 -t get,set -n 1000000 -r 100 -q
SET: 123122.39 requests per second
GET: 120816.71 requests per second
```

### 结论：

通过测试结果可以看出，经过 Twemproxy 有一定的损失，但 OneCache 却没有损失，反而增加了。因为 OneCache 内部采用了多线程方式进行网络通信，提高了连接可复用性以及网络 IO 的利用率。

## 指标测试 2

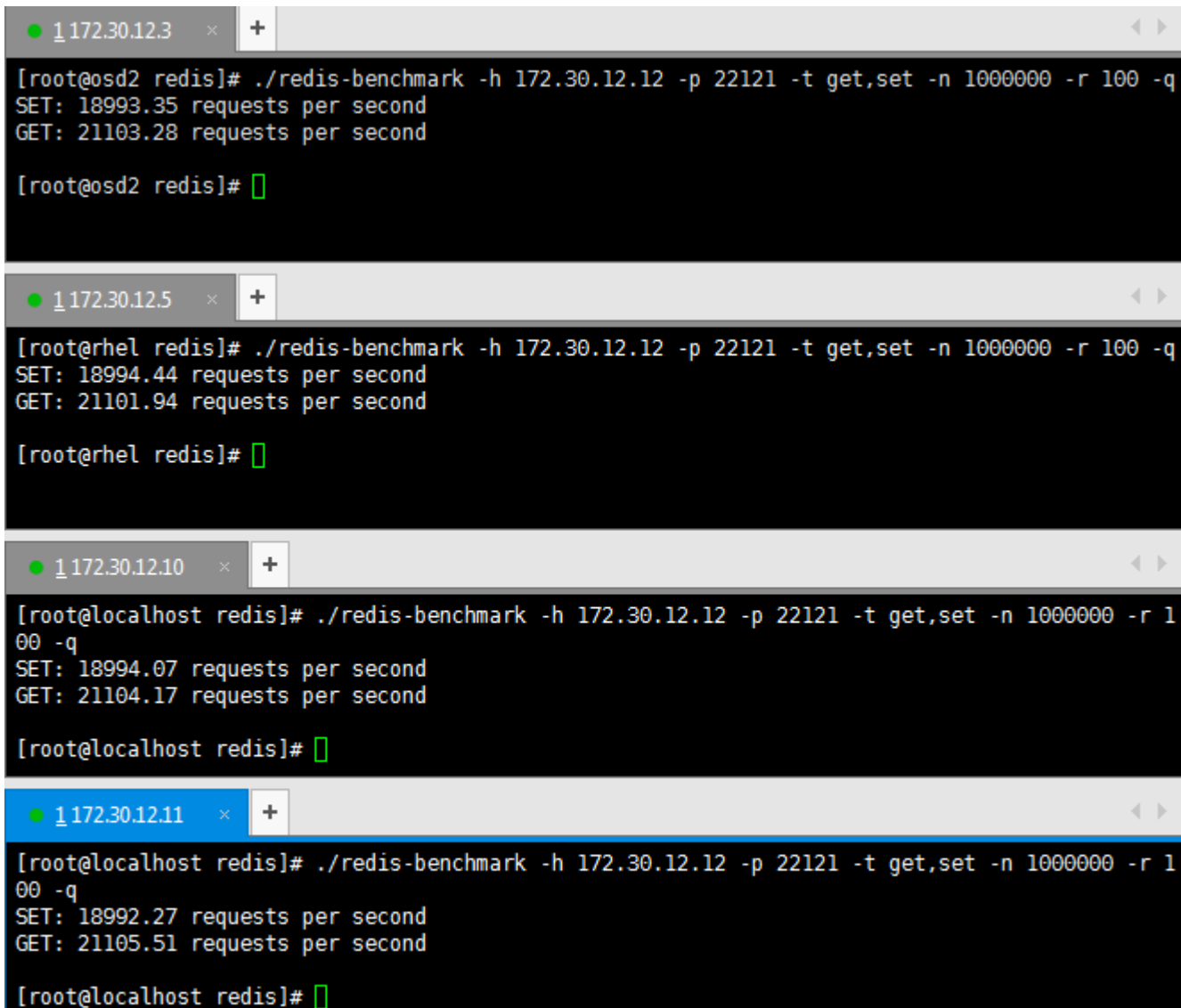
多个机器上各跑 redis-benchmark 并同时进行。

因为 redis-benchmark 是单线程方式测试的，单个测试并不能反映实际的并发量，所以现在用 4 个单独的机器来跑测试程序。OneCache 和 Twemproxy 后端各挂 4 个 Redis 节点，通过此方式来对比两者的每秒的并发量。

Twemproxy 启动配置

```
[root@localhost twemproxy]# cat nutcracker.yml
alpha:
  listen: 0.0.0.0:22121
  hash: fnv1a_64
  distribution: ketama
  auto_eject_hosts: true
  redis: true
  server_retry_timeout: 2000
  server_failure_limit: 1
  servers:
    - 172.30.12.6:6379:1
    - 172.30.12.8:6379:1
    - 172.30.12.9:6379:1
    - 172.30.12.2:6379:1
[root@localhost twemproxy]# ./nutcracker -c nutcracker.yml -v 0
[2015-09-02 00:31:46.937] nc.c:187 nutcracker-0.4.1 built for Linux 2.6.32-504.16.2.el6.x86_64 x86_64 started on pid 20123
[2015-09-02 00:31:46.937] nc.c:192 run, rabbit run / dig that hole, forget the sun / and when at last the work is done / don't sit down / it's time to dig another one
```

测试结果图



The image displays four terminal windows, each showing the output of the redis-benchmark command. The windows are titled with IP addresses: 172.30.12.3, 172.30.12.5, 172.30.12.10, and 172.30.12.11. Each window shows the command being executed and the resulting performance metrics for SET and GET operations.

IP Address	SET (requests per second)	GET (requests per second)
172.30.12.3	18993.35	21103.28
172.30.12.5	18994.44	21101.94
172.30.12.10	18994.07	21104.17
172.30.12.11	18992.27	21105.51

## OneCache 启动配置

```
[root@localhost onecache]# cat onecache.xml
<onecache port="8221" thread_num="15" hash_value_max="80">
  <vip if_alias_name="em1:0" vip_address="172.30.12.100" enable="0"></vip>

  <group name="group1" hash_min="0" hash_max="20">
    <host host_name="host1" ip="172.30.12.6" port="6379" master="1"></host>
  </group>
  <group name="group2" hash_min="21" hash_max="40">
    <host host_name="host1" ip="172.30.12.8" port="6379" master="1"></host>
  </group>
  <group name="group3" hash_min="41" hash_max="60">
    <host host_name="host1" ip="172.30.12.9" port="6379" master="1"></host>
  </group>
  <group name="group4" hash_min="61" hash_max="80">
    <host host_name="host1" ip="172.30.12.2" port="6379" master="1"></host>
  </group>
</onecache>
[root@localhost onecache]# ./onecache
[00:54:06] Warning: No config file specified. using the default config(onecache.xml)
[00:54:06] Message: Load config file 'onecache.xml'...
[00:54:06] Message: Create connection pool (172.30.12.6:6379)...
[00:54:06] Message: Create connection pool (172.30.12.8:6379)...
[00:54:06] Message: Create connection pool (172.30.12.9:6379)...
[00:54:06] Message: Create connection pool (172.30.12.2:6379)...
[00:54:06] Message: Create the thread pool...
[00:54:06] Message: Thread pool created. size: 15
[00:54:06] Message: OneCache has running. Port: 8221
```

## 测试结果图

The image displays four terminal windows, each showing the output of a Redis benchmark command. The command used is `./redis-benchmark -h 172.30.12.12 -p 8221 -t get,set -n 1000000 -r 100 -q`. The results show the number of requests per second for SET and GET operations.

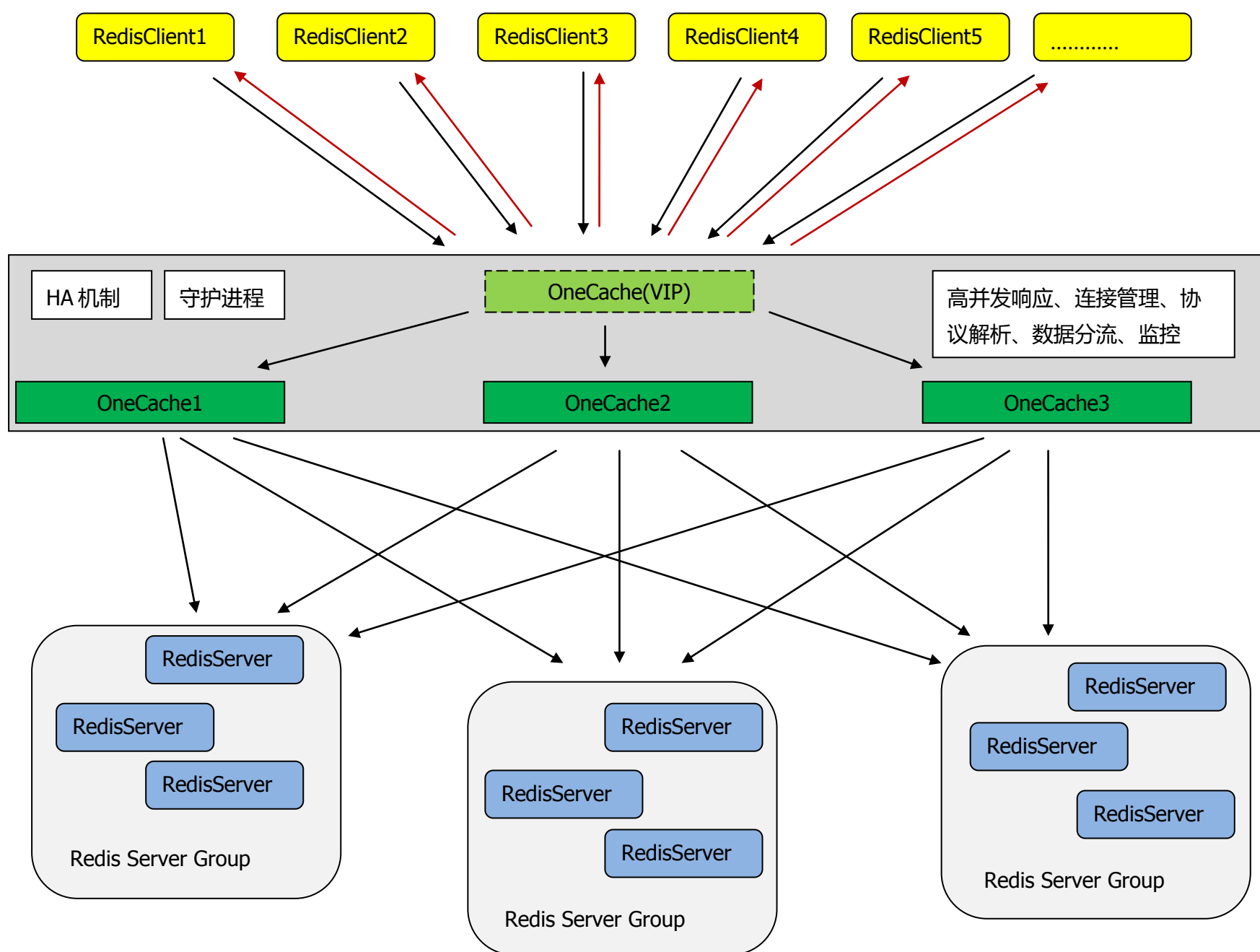
- Terminal 1 (172.30.12.3):** SET: 64758.45 requests per second, GET: 74194.98 requests per second.
- Terminal 2 (172.30.12.5):** SET: 66688.90 requests per second, GET: 77267.80 requests per second.
- Terminal 3 (172.30.12.10):** SET: 69686.41 requests per second, GET: 80295.48 requests per second.
- Terminal 4 (172.30.12.11):** SET: 69415.52 requests per second, GET: 78789.79 requests per second.

## 结论：

通过测试结果可以看出

1. Twemproxy 后端有多台 Redis，前端的单个 Twemproxy 的性能最大也只能和单台 Redis 性能差不多。
2. OneCache 后端有多个 Redis，前端单个 OneCache 的性能有很大的增量，GET 命令已经达到 30W/S。

## ✧ 功能模型



### 1. 转发规则

OneCache 采用 **HASH 分片** 方式进行转发。

OneCache 收到每个请求时都会快速的解析请求包的内容，例如是操作命令是什么，对应的操作 KEY 是什么，然后将 **KEY 进行散列化操作映射到某一个 Group 中去处理**。在这里，OneCache 担任着分流的作用，真正的处理是每个 Group 内部完成的。

### 2. RedisServerGroup 模型

传统的代理模型大部分是后端挂 N 个节点，节点之间没有很直接的关系。OneCache 中引入了 Group 模型，Group 内包含 master & slaver 节点。

对于每个 Group，都会有如下特性：

1. 每个 Group 都可以配置自己的散列范围，因此也可以控制每个 Group 的所占权重。
2. 每个 Group 也可以指定一些特定的 KEY，这些 KEY 最终会由该 Group 进行处理。
3. 每个 Group 可以有自己的策略模式。目前只提供了默认的策略。但基于这个结构模型，这些很容易得到扩展。

目前 OneCache 的默认策略是：master(s)负责写操作，slaver(s)负责读操作，采用均匀分配的方式去处理每个请求，所以也降低了单个节点的负载率。OneCache 能识别非活动的节点，从而跳过该节点，并能够正确寻找下一个可用的节点，如无节点可用，OneCache 将会返回错误的答复。



✧ 软件安装与配置

- 1. 下载软件。软件只有一个可执行文件 onecache
- 2. 启动软件。软件启动只依赖一个配置文件（XML 格式）。命令格式：onecache [cfg-file]。如果未指定配置文件路径名，默认会以当前目录下的 onecache.xml 做为配置，如果配置参数中有不正确的地方，软件将会报错，具体出错原因可以查看输出的日志信息。

这是一个典型的配置文件格式

```
<onecache port="8221" thread_num="15" hash_value_max="80" daemonize="0" guard="0" backendReconnInterval="5" >
  <!--软件端口为 8221 开启的线程数为 15 个 最大哈希范围为 80(最大不得超过 1024)-->
  <!--daemonize 表示是否后台运行 1=YES 0=NO-->
  <!--guard 表示是否启用守护进程 1=YES 0=NO-->
  <!--backendReconnInterval 表示后端断开后的重连间隔时间,单位: 秒-->
  <vip if_alias_name="eth0:0" vip_address="172.30.12.8" enable="0"></vip>
  <!--VIP 配置 if_alias_name 表示适配器别名 vip_address 表示虚拟地址 enable 表示是否启用 VIP 功能 1:启用 0:禁用-->
  <top_key enable="0"></top_key>
  <!--是否启用 TopKey 统计功能 1:启用 0:禁用-->
  <group name="group1" hash_min="0" hash_max="19">
    <!--组名为 group1 哈希映射的范围为 0~20 (包含 0,20)-->
    <host host_name="host1" ip="172.30.12.6" port="6379" master="1"></host>
    <host host_name="host1" ip="172.30.12.6" port="6380" master="0"></host>
    <!--host_name 表示主机别名 ip 为 Redis 地址 port 为 redis 启动端口 master 表示是否主备 1:主 0:备-->
  </group>
  <group name="group2" hash_min="20" hash_max="39">
    <host host_name="host1" ip="172.30.12.8" port="6379" master="1"></host>
  </group>
  <group name="group3" hash_min="40" hash_max="59">
    <host host_name="host1" ip="172.30.12.9" port="6379" master="1"></host>
  </group>
  <group name="group4" hash_min="60" hash_max="79">
    <host host_name="host1" ip="172.30.12.2" port="6379" master="1"></host>
  </group>
  <!--配置 HASH 值到 Group 的映射关系-->
  <hash_mapping>
    <hash value="0" group_name="group4"></hash>
    <hash value="1" group_name="group4"></hash>
  </hash_mapping>
  <!--配置指定 KEY 到 Group 的映射关系-->
  <key_mapping>
    <key key_name="key1" group_name="group1"></key>
    <key key_name="key2" group_name="group2"></key>
  </key_mapping>
</onecache>
```

正常启动 OneCache 会输出如下信息(主机地址：172.30.12.12 端口：8221)

```
[root@ol5-112 onecache]# ./onecache
OneCache 2.0, Copyright 2015 by OneXSoft

[2015-09-22 23:28:55] Message: No config file specified. using the default config(onecache.xml)
[2015-09-22 23:28:55] Message: Create connection pool (172.30.12.6:6379)...
[2015-09-22 23:28:55] Message: Creating successful
[2015-09-22 23:28:55] Message: Create connection pool (172.30.12.8:6379)...
[2015-09-22 23:28:55] Message: Creating successful
[2015-09-22 23:28:55] Message: Create connection pool (172.30.12.9:6379)...
[2015-09-22 23:28:55] Message: Creating successful
[2015-09-22 23:28:55] Message: Create connection pool (172.30.12.2:6379)...
[2015-09-22 23:28:55] Message: Creating successful
[2015-09-22 23:28:55] Message: Create the thread pool...
[2015-09-22 23:28:55] Message: Thread pool created. size: 15
[2015-09-22 23:28:55] Message: Start the OneCache on port 8221
```

✧ 软件使用

1. 支持的命令

启动软件后，可以使用 Redis 自带的客户端去连接代理，连接成功后跟正常访问 Redis 服务器一样。但 OneCache 并不是完全支持了所有的 Redis 命令。下表是目前所支持的命令以及对应的测试例程。

命令类型	输入测试	输出结果
APPEND	append a 1	(integer) 1
	append a 2	(integer) 2
	get a	"12"
BITCOUNT	set bitkey mybitval	OK
	bitcount bitkey 0 2	(integer) 13
	bitcount bitkey 0 1	(integer) 10
BITPOS	bitpos bitkey 0	(integer) 0
DUMP	set mykey myvalue	OK
	dump mykey	"\x00\amyvalue\x06\x00\x88^y\xc8\x10\xff\xb4b"
DEL	set del_test_key abc	OK
	del del_test_key	(integer) 1
	get del_test_key	(nil)
DECR	set decr_test_key 100	OK
	decr decr_test_key	(integer) 99
DECRBY	decrby decr_test_key 10	(integer) 89
EXPIREAT	set mykey 123	OK
	expireat mykey 5	(integer) 1
	get mykey	(nil)
EXISTS	set mykey abc	OK
	exists mykey	(integer) 1
	del mykey	(integer) 1
	exists mykey	(integer) 0
EXPIRE	set mykey myvalue	OK
	expire mykey 5	(integer) 1
	get mykey	"myvalue"
	get mykey	(nil)
GET	set a b	OK
	get a	"b"
GETBIT	set bitkey 123	OK
	getbit bitkey 0	(integer) 0
	getbit bitkey 7	(integer) 1
GETRANGE	set a helloworld	OK
	getrange a 1 2	"el"
	getrange a 1 5	"ellow"
GETSET	set a old	OK
	getset a new	"old"
	get a	"new"
HSET	hset myhash name abc	(integer) 1
HSETNX	hset myhash name abc	(integer) 1
	hsetnx myhash name abc	(integer) 0
	hsetnx myhash1 name abc	(integer) 1
HMSET	hmset myhash name abc age 12	OK
	hmget myhash name age	1) "abc" 2) "12"
HGET	hget myhash name	"abc"
HMGET	hmset myhash name abc age 12	OK
	hmget myhash name age	1) "abc" 2) "12"
HINCRBY	hset myhash age 10	(integer) 1
	hincrby myhash age 1	(integer) 11
HEXISTS	hexists myhash age	(integer) 1
	hexists myhash age1	(integer) 0
HLEN	hset myhash name abc	(integer) 1
	hset myhash age 10	(integer) 1
	hlen myhash	(integer) 2
HDEL	hdel myhash name	(integer) 1
HKEYS	hkeys myhash	1) "age"
HVALS	hvals myhash	1) "10"
HGETALL	hgetall myhash	1) "age" 2) "10"



HINCRBYFLOAT	hincrbyfloat myhash age 1.2	"11.2"
INCR	set a 1	OK
	incr a	(integer) 2
	get a	"2"
INCRBY	incrby a 3	(integer) 5
	get a	"5"
INCRBYFLOAT	set mykey 10.50	OK
	incrbyfloat mykey 0.1	"10.6"
LPUSH	lpush abclist a b c	(integer) 3
LRANGE	lrange abclist 0 -1	1) "c" 2) "b" 3) "a"
LPUSHX	lpushx abclist "hello"	(integer) 4
	lrange abclist 0 -1	1) "hello" 2) "c" 3) "b" 4) "a"
LPOP	lpop abclist	"hello"
	lrange abclist 0 -1	1) "c" 2) "b" 3) "a"
LREM	lrem abclist 1 a	(integer) 1
	lrange abclist 0 -1	1) "c" 2) "b"
LINDEX	lindex abclist 0	"c"
	lindex abclist 1	"b"
RPUSH	rpusth yourlist hello	(integer) 1
	lrange yourlist 0 -1	1) "hello"
LINSERT	linsert yourlist BEFORE "world" "hi"	(integer) 3
	lrange yourlist 0 -1	1) "hello" 2) "hi" 3) "world"
LLEN	lpush helloworld "hello"	(integer) 1
	lpush helloworld "world"	(integer) 2
	llen helloworld	(integer) 2
LSET	lpush job "cook food"	(integer) 1
	lrange job 0 0	1) "cook food"
	lset job 0 "play game"	OK
	lrange job 0 0	1) "play game"
LTRIM	lrange job 0 -1	1) "play game"
	ltrim job 1 -1	OK
	lrange job 0 -1	(empty list or set)
MSET	mset a 1 b 2 c 3	OK
MGET	mget a b c	1) "1" 2) "2" 3) "3"
PSETEX	psetex mykey 10000 "Hello"	OK
	get mykey	"Hello"
	get mykey	(nil)
PERSIST	set mykey "Hello"	OK
	exprie mykey 10	(integer) 1
	ttl mykey	(integer) 7
	persist mykey	(integer) 1
	ttl mykey	(integer) -1
PEXPIRE	pexpire mykey 15000	(integer) 1
TTL	ttl mykey	(integer) 10
PTTL	pttl mykey	(integer) 4683
PEXPIREAT	pexpireat mykey 1555555555005	(integer) 1
	ttl mykey	(integer) 113229494
PING	Ping	PONG
RESTORE	set greeting "hello, dumping world!"	OK
	dump greeting	"\x00\x15hello, dumping world!\x06\x00E\xa0Z\x82\xd8r\xcl\xde"
	restore greeting-again 0 "\x00\x15hello, dumping world!\x06\x00E\xa0Z\x82\xd8r\xcl\xde"	OK
	get greeting-again	"hello, dumping world!"
RPOP	rpusth testlist one	(integer) 1
	rpusth testlist two	(integer) 2
	lrange testlist 0 -1	1) "one" 2) "two"
	rpop testlist	"two"
	lrange testlist 0 -1	1) "one"
RPUSH	rpusth greet testlist	(integer) 1

	lrange greet 0 -1	1) "testlist"
SETBIT	setbit bit 10086 1	(integer) 0
SETRANGE	set greeting "hello world"	OK
	setrange greeting 6 Redis	(integer) 11
	get greeting	"hello Redis"
STRLEN	strlen mykey	(integer) 5
SET	set a 1	OK
SETEX	setex cache_user_id 60 10086	OK
	get cache_user_id	"10086"
	ttl cache_user_id	(integer) 52
SETNX	exists jobb	(integer) 0
	setnx jobb programmer	(integer) 1
	setnx jobb code-farmer	(integer) 0
	get jobb	"programmer"
TYPE	set weather sunny	OK
	type weather	String
SHOWCMD	Showcmd *** Support the command *** [Redis Command] APPEND BITCOUNT BITPOS DUMP DEL DECR DECRBY EXPIREAT EXISTS EXPIRE GET GETBIT GETRANGE GETSET HSET HSETNX HMSET HGET HMGET HINCRBY HEXISTS HLEN HDEL HKEYS HVALS HGETALL HINCRBYFLOAT INCR INCRBY INCRBYFLOAT LPUSH LPUSHX LPOP LRANGE LREM LINDEX LINSERT LLEN LSET LTRIM MGET MSET PSETEX PERSIST PEXPIRE PEXPIREAT PTTL PING RESTORE RPOP RPUSH RPUSHX SETBIT SETRANGE STRLEN SET SETEX SETNX TTL TYPE  [OneCache Command] SHOWCMD STATUS OUTPUTSTATUS TOPKEY HASHMAPPING ADDKEYMAPPING DELKEYMAPPING SHOWMAPPING	
STATUS	状态监控	
OUTPUTSTATUS	将状态监控输出到文件	
TOPKEY	查看当前 TopKey 状态，根据 KEY 的请求次数。使用方式 <b>TOPKEY [nums]</b> nums 表示要查看前多少个(默认 20 个)。	
TOPVALUE	查看当前 TopKey 状态，根据对应 KEY 的 VALUE 大小。使用方式 <b>TOPVALUE [nums]</b> nums 表示要查看前多少个(默认 20 个)。	
HASHMAPPING	配置 HASH 值到 Group 的映射。使用方式 <b>HASHMAPPING [hash value] [group name]</b>	
ADDKEYMAPPING	配置 KEY 到 Group 的映射。 使用方式 <b>ADDKEYMAPPING [group name] [key1] [key2]...</b>	
DELKEYMAPPING	删除 KEY 到 Group 的映射。 使用方式 <b>DELKEYMAPPING [key1] [key2]...</b>	
SHOWMAPPING	显示 Group 的映射关系。	

2. 配置 Group 映射关系

程序启动时必须给每个 Group 配置自己的散列范围（散列范围在 0~MaxHashValue 内），如果想在程序运行过程中去动态修改这种映射关系，则可以用如下方法：

首先通过 **SHOWMAPPING** 查看映射关系（这是笔者测试时使用的配置）

```
172.30.12.6:8221> showmapping

[HASH MAPPING]
HASH_VALUE    GROUP_NAME
0             group1
1             group1
2             group1
3             group1
4             group1
5             group2
6             group2
7             group2
8             group2
9             group2
10            group3
11            group3
12            group3
13            group3
14            group3
15            group4
16            group4
17            group4
18            group4
19            group4

[KEY MAPPING]
ID    NAME    KEYS
0     group1
1     group2
2     group3
3     group4
```

通过图中 **[HASH MAPPING]** 项可以看到一共有 4 个 Group，最大哈希值是 20（0~19），每个 Group 所占权重都是一样的。

通过图中 **[KEY MAPPING]** 项可以看出，目前没有配置特定 KEY 到 Group 的映射关系。

假设现在有如下需求：

- 1)将 HASH 值为 19 的映射组改为 group1
  - 2)在 group1 和 group2 各自增加一个特定的 KEY group1\_key 和 group2\_key
- 可以通过命令 **HASHMAPPING** 和 **ADDKEYMAPPING** 命令完成此需求。如下图

```
172.30.12.6:8221> hashmapping 19 group1
OK
172.30.12.6:8221> addkeymapping group1 group1_key
OK
172.30.12.6:8221> addkeymapping group2 group2_key
OK
```

修改完毕后再可以通过 **SHOWMAPPING** 进行查看。如图

```
172.30.12.6:8221> showmapping

[HASH MAPPING]
HASH_VALUE  GROUP_NAME
0           group1
1           group1
2           group1
3           group1
4           group1
5           group2
6           group2
7           group2
8           group2
9           group2
10          group3
11          group3
12          group3
13          group3
14          group3
15          group4
16          group4
17          group4
18          group4
19          group1

[KEY MAPPING]
ID  NAME      KEYS
0   group1   group1_key
1   group2   group2_key
2   group3
3   group4
```

可以看出配置已经生效了，这时如果有请求中 KEY 的 HASH 值为 19 或者 KEY 为 group1\_key 都会由 group1 进行处理。

提示：

此操作会将当前的配置以及当前的改动信息回写到新的配置文件中，以免程序意外退出造成配置的丢失。配置文件生成的路径在当前可执行文件目录下，命名格式为：onecache+时间戳.xml。

3. 查看 TOPKEY

目前可以按 KEY 的请求次数（对应命令 **TOPKEY**）和 KEY 对应的 VALUE 大小(对应命令 **TOPVALUE**)进行统计。如图：

这是按请求个数统计的排名，这里只显示前 10 个

```
172.30.12.12:8222> topkey 10
[KEY]                                     [COUNT]
key:000075657445                         3
key:000062986318                         3
key:000059412504                         3
key:000040874736                         2
key:000097231074                         2
key:000018766753                         2
key:000060149359                         2
key:000016886123                         2
key:000057565384                         2
key:000033091431                         2
```

下面是按 KEY 对应 VALUE 的大小进行统计，这里也是只显示前 10 个。

```
172.30.12.12:8222> topvalue 10
[KEY]                                     [VALUESIZE]
key:000062986318                         0.015KB
key:000075657445                         0.015KB
key:000059412504                         0.015KB
key:000088971456                         0.010KB
key:000098812864                         0.010KB
key:000028201088                         0.010KB
key:000078339090                         0.010KB
key:000045220107                         0.010KB
key:000017016621                         0.010KB
key:000030080340                         0.010KB
```

这两个命令比较简单，在此就不多做论述了。

提示： 启用此功能会带来一定性能损耗，如果不需要可以在配置文件中指定该功能不启用。

4. 状态监控

用 Redis 自带客户端 redis-cli 连接上 OneCache 后，可以用 STATUS 或者 OUTPUTSTATUS 命令查看当前状态。如下图所示

```
[root@localhost redis]# ./redis-cli -h 172.30.12.12 -p 8222
172.30.12.12:8222> status
[OneCache]
Version=2.0
Port=8222
StartTime=2015-9-25 17:16:36
UpTime=00:01:18
GroupCount=4
VipEnabled=No
TopKeyEnable=Yes

[Backends]
[GROUP]      [IP]      [PORT] [CONNPOOL] [MASTER] [ACTIVE]      [REQUESTS]      [RECV SIZE]      [SEND SIZE]      [SEND>1KB]      [SEND>1MB]      [COMMANDS]
group1       172.30.12.6 6379   50         Y         Y            289237          9.93MB           1.53MB           0               0               [GET]:289237
group2       172.30.12.8 6379   50         Y         Y            260387          8.94MB           1.47MB           0               0               [GET]:260387
group3       172.30.12.9 6379   50         Y         Y            180092          6.18MB           879.36KB         0               0               [GET]:180092
group4       172.30.12.2 6379   50         Y         Y            270284          9.28MB           1.29MB           0               0               [GET]:270284

[Clients]
[NUM]      [IP]      [CONNECTS] [REQUESTS] [RECV>1KB] [RECV>1MB]      [LAST CONNECT]      [COMMANDS]
1          172.30.12.11 52         1000000    0           0               2015-9-25 17:17:52  [GET]:1000000
```

字段名称	含义
[OneCache]	OneCache 全局信息
Version	软件版本
Port	软件启动所使用的端口
StartTime	OneCache 的启动时间
UpTime	OneCache 运行时间
GroupCount	后端组的个数
VipEnabled	是否启用 VIP
VipName	VIP 所使用的接口别名
VipAddress	VIP 地址
[Backends]	记录所有后端节点
GROUP	后端分组名
IP	后端 IP 地址
PORT	后端端口
MASTER	Y:主机 N:备机
ACTIVE	Y:活动状态 N:非活动状态
REQUESTS	请求数量
RECV SIZE	收到的字节数
SEND SIZE	发送的字节数
SEND>1KB	后端发送出去的包大小超过 1KB 的数量
SEND>1MB	后端发送出去的包大小超过 1MB 的数量
COMMANDS	所有的操作命令记录
[Clients]	记录所有客户端信息
NUM	客户端编号
IP	客户端 IP 地址
CONNECTS	客户端连接次数
REQUESTS	客户端请求次数
RECV>1KB	客户端接收的回复包大小超过 1KB 的数量
RECV>1MB	客户端接收的回复包大小超过 1MB 的数量
LAST CONNECT	客户端最后一次连接时间
COMMANDS	客户端所有命令记录