redis-cluster +sentinel+monitor tools

--动态实时监控*redis*集群

谷阳

2018-08-31

redis运维监控平台介绍

运维监控平台针对于redis集群的监控状态的动态分析

A redis集群特点

B 哨兵模式特点

C 监控redis的维度

D 监控redis的工具

PART ONE

01

redis集群的特点

特点介绍

1、所有的redis节点彼此互联(PING-PONG机制),内部使用二进制协议优化传输速度和带宽。

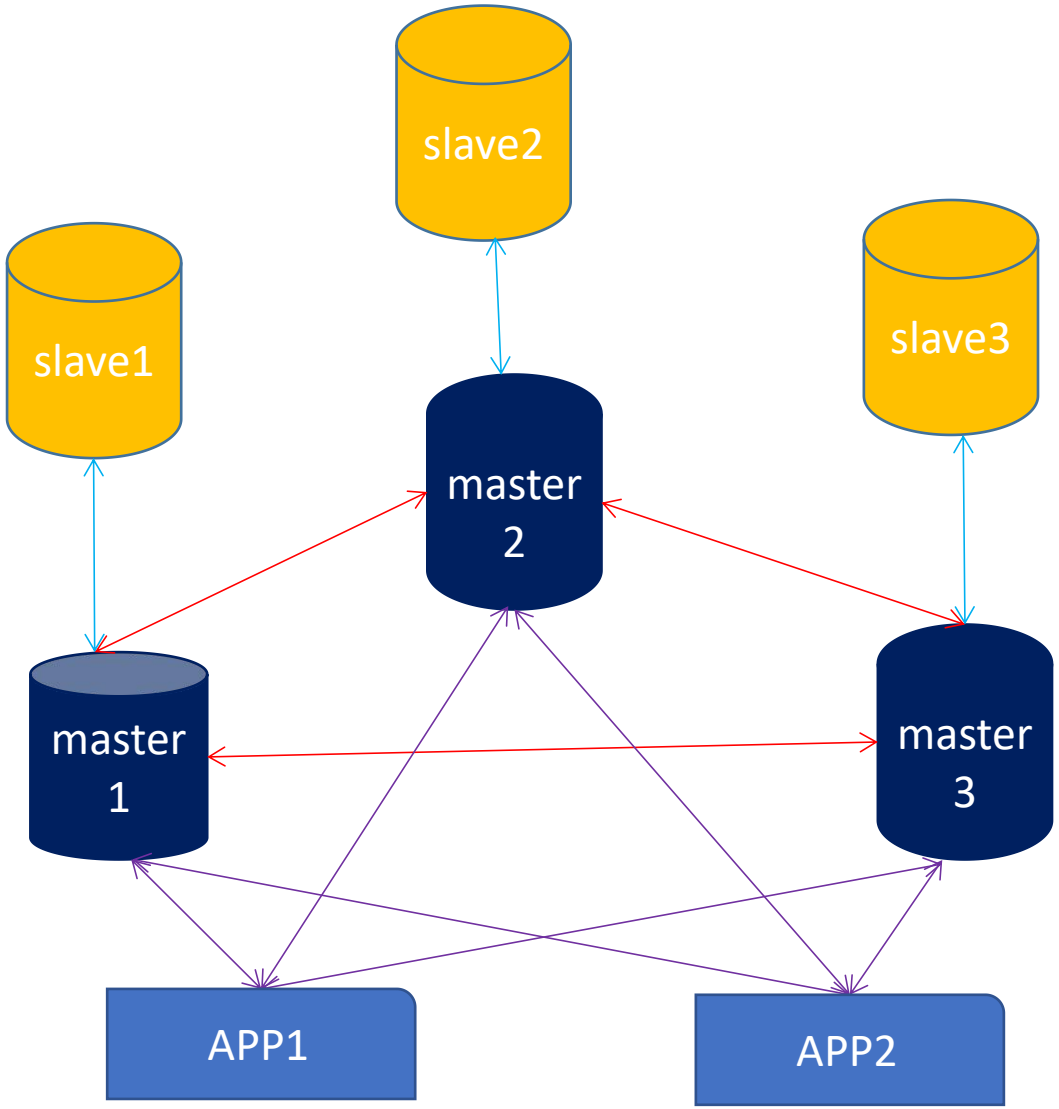
2、节点的fail是通过集群中超过半数的节点检测失效时才生效。

3、客户端与redis节点直连,不需要中间proxy层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。

4、redis-cluster把所有的物理节点映射到[0-16383]slot上（不一定是平均分配）,cluster 负责维护node<->slot<->value。

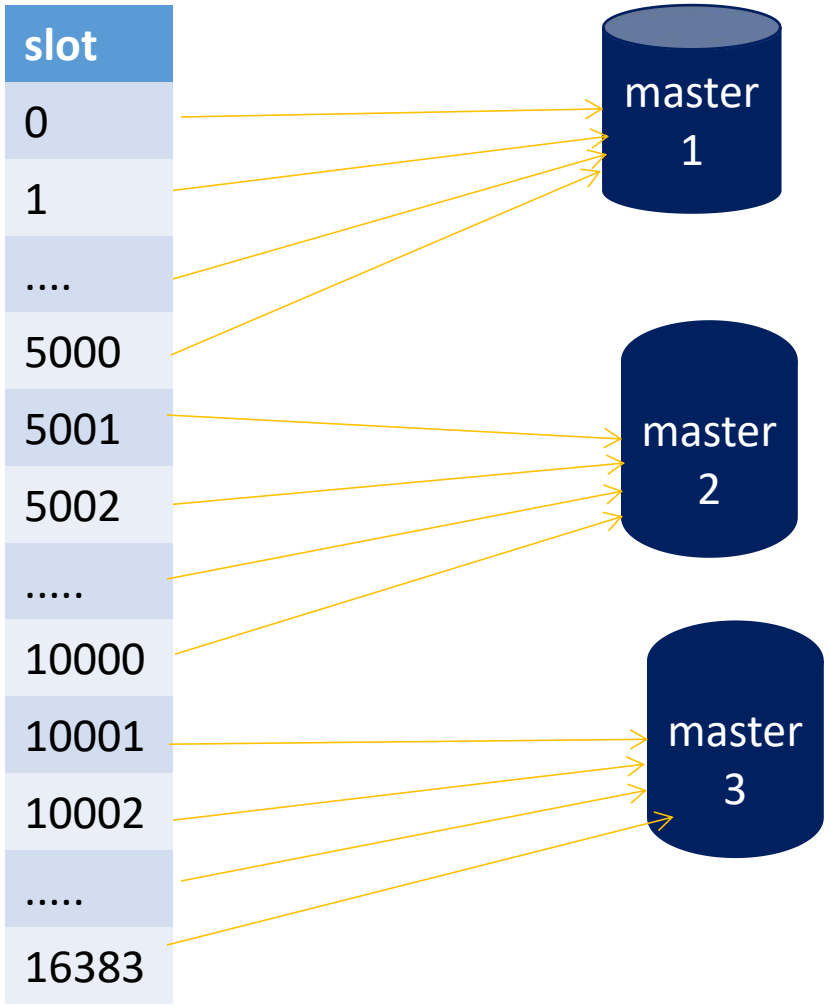
5、Redis集群预分好16384个slot，当需要在 Redis 集群中放置一个 key-value 时，根据 CRC16(key) mod 16384的值，决定将一个key放到哪个桶中。

集群拓扑图



redis-sharding

“



”

故障机制

1、 集群中的每个节点都会定期的向其它节点发送PING命令，并且通过有没有收到回复判断目标节点是否下线；

2、 集群中每一秒就会随机选择5个节点，然后选择其中最久没有响应的节点放PING命令；

3、 如果一定时间内目标节点都没有响应，那么该节点就认为目标节点疑似下线；

4、 当集群中的节点超过半数认为该目标节点疑似下线，那么该节点就会被标记为下线；

5、 当集群中的任何一个节点下线，就会导致插槽区有空档，不完整，那么该集群将不可用；

6、 如何解决上述问题?

a) 在Redis集群中可以使用主从模式实现某一个节点的高可用

b) 当该节点（master）宕机后，集群会将该节点的从数据库（slave）转变为（master）继续完成集群服务；

故障转移概述

"

集群中，当某个从节点发现其主节点下线时，就会尝试在未来某个时间点发起故障转移流程。具体而言就是先向其他集群节点发送CLUSTERMSG_TYPE_FAILOVER_AUTH_REQUEST包用于拉票，集群主节点收到这样的包后，如果在当前选举纪元中没有投过票，就会向该从节点发送CLUSTERMSG_TYPE_FAILOVER_AUTH_ACK包，表示投票给该从节点。

从节点如果在一段时间内收到了大部分主节点的投票，则表示选举成功，接下来就是升级为主节点，并接管原主节点所负责的槽位，并将这种变化广播给其他所有集群节点，使它们感知这种变化，并修改自己记录的配置信息。

"

主从切换

1.从节点在发现其主节点下线时，并不是立即发起故障转移流程，而是要等待一段时间，在未来的某个时间点才发起选举。这个时间点是这样计算的：{mstime() + 500ms + random()%500ms + rank*1000ms}，固定延时500ms，是为了留出时间，使主节点下线的消息能传播到集群中其他节点，这样集群中的主节点才有可能投票；随机延时是为了避免两个从节点同时开始故障转移流程；rank表示从节点的排名，排名是指当前从节点在下线主节点的所有从节点中的排名，排名主要是根据复制数据量来定，复制数据量越多，排名越靠前，因此，具有较多复制数据量的从节点可以更早发起故障转移流程，从而更可能成为新的主节点。

2.从节点发起故障转移，开始拉票，从节点的故障转移，是在函数clusterHandleSlaveFailover中处理的，该函数在集群定时器函数clusterCron中调用。本函数用于处理从节点进行故障转移的整个流程，包括：判断是否可以发起选举；发起选举；判断选举是否超时；判断自己是否拉到了足够的选票；使自己升级为新的主节点这些所有流程。

3.主节点投票， 集群中所有节点收到用于拉票的CLUSTERMSG_TYPE_FAILOVER_AUTH_REQUEST包后，只有负责一定槽位的主节点能投票，其他没资格的节点直接忽略掉该包，在clusterProcessPacket中，判断收到的是CLUSTERMSG_TYPE_FAILOVER_AUTH_REQUEST包后，就会调用clusterSendFailoverAuthIfNeeded函数，在满足条件的基础上，给发送者投票。

4.从节点统计投票、赢得选举，从节点CLUSTERMSG_TYPE_FAILOVER_AUTH_ACK包后，就会统计投票。

5.更新配置，故障转移后，某个从节点提升为主节点，并接手原主节点所负责的槽位。接下来更新所需要的配置信息。使得其他节点能感知到，这些槽位现在由新的节点负责。此时configEpoch发挥作用。

PART two

02

哨兵模式特点

特点介绍

1.监控：Sentinel不断的检查master和slave是否正常的运行。

2.通知：如果发现某个redis节点运行出现问题，可以通过API通知系统管理员和其他的应用程序。

3.自动故障转移：能够进行自动切换。当一个master节点不可用时，能够选举出master的多个slave中的一个来作为新的master,其它的slave节点会将它所追随的master的地址改为被提升为master的slave的新地址。

4.配置提供者：sentinel作为Redis客户端发现的权威来源：客户端连接到哨兵请求当前可靠的master的地址。如果发生故障，哨兵将报告新地址。

原理介绍

"

①sentinel集群通过给定的配置文件发现master，启动时会监控master。通过向master发送info信息获得该服务器下面的所有从服务器。

②sentinel集群通过命令连接向被监视的主从服务器发送hello信息(每秒一次)，该信息包括sentinel本身的ip、端口、id等内容，以此来向其他sentinel宣告自己的存在。

③sentinel集群通过订阅连接接收其他sentinel发送的hello信息，以此来发现监视同一个主服务器的其他sentinel；集群之间会互相创建命令连接用于通信，因为已经有主从服务器作为发送和接收hello信息的中介，sentinel之间不会创建订阅连接。

④sentinel集群使用ping命令来检测实例的状态，如果在指定的时间内（down-after-milliseconds）没有回复或则返回错误的回复，那么该实例被判为下线。

"

原理介绍

⑤当failover主备切换被触发后，failover并不会马上进行，还需要sentinel中的大多数sentinel授权后才可以进行failover，即进行failover的sentinel会去获得指定quorum个的sentinel的授权，成功后进入ODOWN状态。如在5个sentinel中配置了2个quorum，等到2个sentinel认为master死了就执行failover。

⑥sentinel向选为master的slave发送SLAVEOF NO ONE命令，选择slave的条件是sentinel首先会根据slaves的优先级来进行排序，优先级越小排名越靠前。如果优先级相同，则查看复制的下标，哪个从master接收的复制数据多，哪个就靠前。如果优先级和下标都相同，就选择进程ID较小的。

⑦sentinel被授权后，它将会获得宕掉的master的一份最新配置版本号(config-epoch)，当failover执行结束以后，这个版本号将会被用于最新的配置，通过广播形式通知其它sentinel，其它的sentinel则更新对应master的配置。

原理介绍

”

①到③是自动发现机制:

- 以10秒一次的频率，向被监视的master发送info命令，根据回复获取master当前信息。
- 以1秒一次的频率，向所有redis服务器、包含sentinel在内发送PING命令，通过回复判断服务器是否在线。
- 以2秒一次的频率，通过向所有被监视的master，slave服务器发送当前sentinel，master信息的消息。

④是检测机制

⑤和⑥是failover机制

⑦是更新配置机制。

”

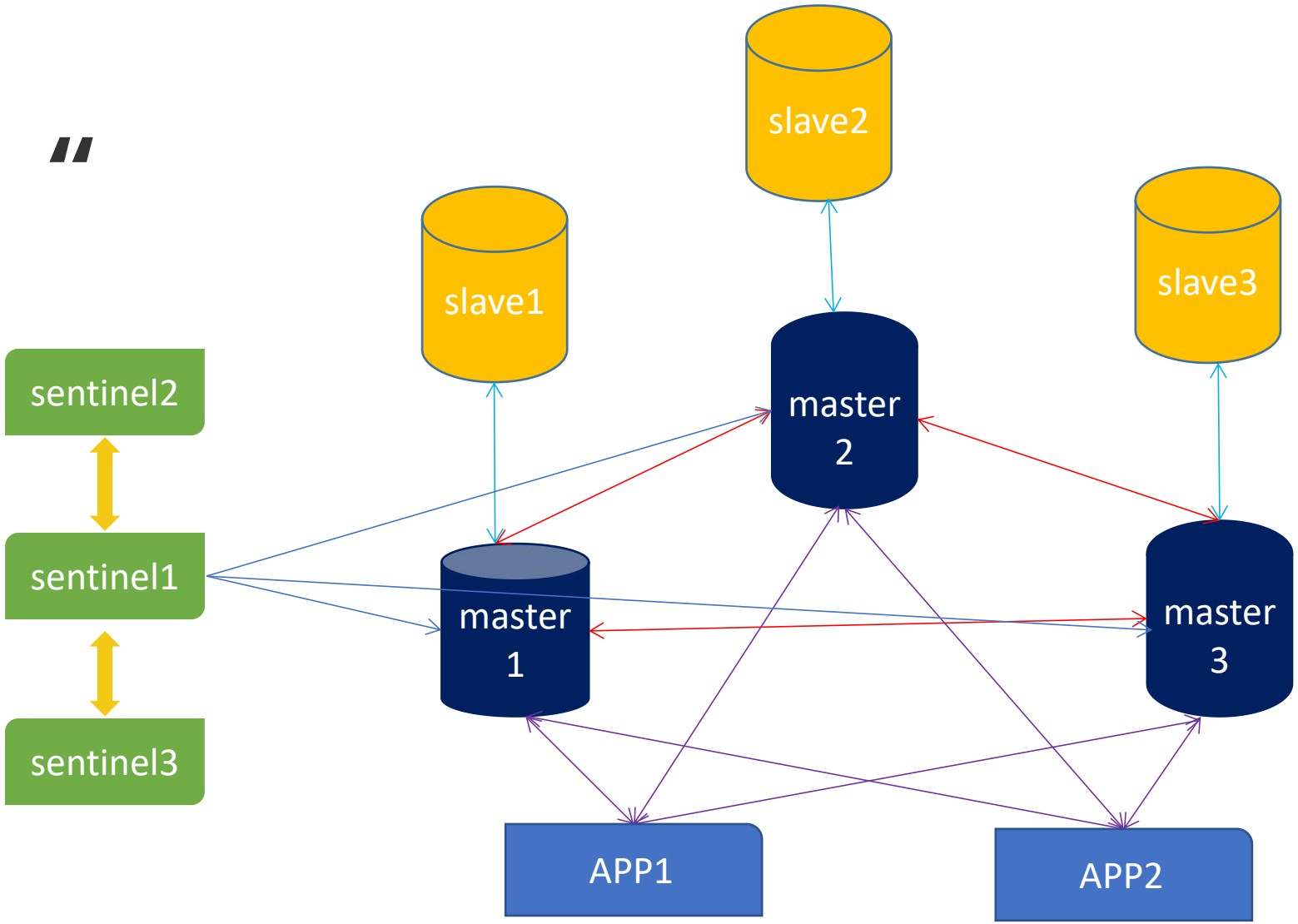
sentinel分布式特性

1.即使有一些sentinel进程宕掉了，依然可以进行redis集群的主备切换；

2.如果只有一个sentinel进程，如果这个进程运行出错，或者是网络堵塞，那么将无法实现redis集群的主备切换（单点问题）；

3.如果有多个sentinel，redis的客户端可以随意地连接任意一个sentinel来获得关于redis集群中的信息。

sentinel模式拓扑图



sentinel配置文件

1.原始配置文件：/usr/local/redis-4.0.11/sentinel.conf

2.配置参数

port 26379 //默认端口号
dir /tmp //默认路径
sentinel monitor mymaster 127.0.0.1 6379 2 //监控nodes的名称mymaster ip地址 端口 2表示当集群种有2个sentinel认为master down时，才能真正认为该master已经不可用了。（sentinel之间相互通信，通过gossip协议）
sentinel down-after-milliseconds mymaster 30000 //sentinel会向master发送心跳ping来确认master是否存活，如果master在“一定时间范围”内不回应PONG 或者是回复了一个错误消息，那么这个sentinel会主观地(单方面地)认为这个master已经不可用了(subjectively down, 也简称为SDOWN)。而这个down-after-milliseconds就是用来指定这个“一定时间范围”的，单位是毫秒。
sentinel parallel-syncs mymaster 1 //在发生failover主备切换时，这个选项指定了最多可以有多少个slave同时对新的master进行同步，这个数字越小，完成failover所需的时间就越长，但是如果这个数字越大，就意味着越多的slave因为replication而不可用。可以通过将这个值设为 1 来保证每次只有一个slave处于不能处理命令请求的状态。
sentinel failover-timeout mymaster 180000 //1. 同一个sentinel对同一个master两次failover之间的间隔时间。2. 当一个slave从一个错误的master那里同步数据开始计算时间。直到slave被纠正为向正确的master那里同步数据时。3.当想要取消一个正在进行的failover所需要的时间。4.当进行failover时，配置所有slaves指向新的master所需的最大时间。不过，即使过了这个超时，slaves依然会被正确配置为指向master，但是就不按parallel-syncs所配置的规则来了。

3.如果有多个sentinel，redis的客户端可以随意地连接任意一个sentinel来获得关于redis集群中的信息。

SDOWN和ODOWN

"

sentinel对于不可用有两种不同的看法，一个叫主观不可用(SDOWN),另外一个叫客观不可用(ODOWN)。SDOWN是sentinel自己主观上检测到的关于master的状态，ODOWN需要一定数量的sentinel达成一致意见才能认为一个master客观上已经宕掉，各个sentinel之间通过命令SENTINEL is_master_down_by_addr来获得其它sentinel对master的检测结果。

从sentinel的角度来看，如果发送了PING心跳后，在一定时间内没有收到合法的回复，就达到了SDOWN的条件。这个时间在配置中通过is-master-down-after-milliseconds参数配置。

"

sentinel的“仲裁会”

“

当ODOWN时，failover被触发。failover一旦被触发，尝试去进行failover的sentinel会去获得“大多数”sentinel的授权（如果票数比大多数还要大的时候，则询问更多的sentinel）
这个区别看起来很微妙，但是很容易理解和使用。例如，集群中有5个sentinel，票数被设置为2，当2个sentinel认为一个master已经不可用了以后，将会触发failover，但是，进行failover的那个sentinel必须先获得至少3个sentinel的授权才可以实行failover。

如果票数被设置为5，要达到ODOWN状态，必须所有5个sentinel都主观认为master为不可用，要进行failover，那么得获得所有5个sentinel的授权。

”

sentinel之间和slave之间的自动发现机制

1.每个sentinel通过向每个master和slave的发布/订阅频道
__sentinel__:hello每秒发送一次消息，来宣布它的存在。

2.每个sentinel也订阅了每个master和slave的频道
__sentinel__:hello的内容，来发现未知的sentinel，当检测到了新的sentinel，则将其加入到自身维护的master监控列表中。

3.每个sentinel发送的消息中也包含了其当前维护的最新的master配置。如果某个sentinel发现
自己的配置版本低于接收到的配置版本，则会用新的配置更新自己的master配置。

PART ONE

03

监控redis的维度

监控指标

1.服务器系统数据采集

2.Redis Server数据采集

3.Redis响应时间数据采集

4.Redis监控Screen

监控对象

1.服务器存活监控

2.redis server监控采集数据

3.redis cluster监控

服务器存活监控

“

1.ping监控告警

2.CPU

3.丢包率

”

Redis Server监控数据采集

1.redis存活监控 (redis_alive):redis本地监控agent使用ping，如果指定时间返回PONG表示存活，否则redis不能响应请求，可能阻塞或死亡。当返回值不为1时，redis挂了，告警。
2.连接个数 (connected_clients)：客户端连接个数，如果连接数过高，影响redis吞吐量。
3.连接数使用率(connected_clients_pct): 连接数使用百分比，通过(connected_clients/maxclients)计算；如果达到1，redis开始拒绝新连接创建，告警。
4.list阻塞调用被阻塞的连接个数 (blocked_clients): 如果监控数据大于0，告警。
5.redis分配的内存大小 (used_memory): redis真实使用内存，不包含内存碎片。
6.redis进程使用内存大小(used_memory_rss): 进程实际使用的物理内存大小，包含内存碎片；如果rss过大导致内部碎片大，内存资源浪费，和fork的耗时和cow内存都会增大。
7.redis内存碎片率 (mem_fragmentation_ratio): 表示(used_memory_rss/used_memory)，碎片率过大，导致内存资源浪费，不设置告警。小于1，表示redis已使用swap分区，则告警。
8.键个数 (keys): redis实例包含的键个数。单实例键个数过大，可能导致过期键的回收不及时。
9.请求键的命中率 (keyspace_hit_ratio):使用keyspace_hits/(keyspace_hits+keyspace_misses)计算所得，命中率低于50%告警。

Redis集群监控

1.实例是否启用集群模式 (cluster_enabled): 通过info的cluster_enabled监控是否启用集群模式。不等于1则告警。

2.集群健康状态 （clusster_state):cluster_state不为OK则告警。

3.集群数据槽slots分配情况 (cluster_slots_assigned):集群正常运行时，默认16384个slots不等于16384则告警。

4.检测下线的数据槽slots个数 (cluster_slots_fail):集群正常运行时，应该为0. 如果大于0说明集群有slot存在故障。

5.集群的节点数 （cluster_known_nodes）：集群中redis节点的个数。

监控工具

1

Redis-stat（Ruby）

3

lepus（python）

2

Redis live（python）

Redis-stat（ruby）

rdis-stat安装部署

redis-stat is a simple Redis monitoring tool written in Ruby.
It is based on INFO command of Redis, and thus generally won't affect the performance of the Redis instance unlike the other monitoring tools based on MONITOR command.
redis-stat allows you to monitor Redis instances
either with vmstat-like output from the terminal

地址：https://github.com/junegunn/redis-stat

开箱即用

安装步骤

"

1.yum install ruby
  sudo apt-get install ruby-full

2.get https://github.com/junegunn/redis-stat

3.gem install redis-stat

"

基本使用

"

usage: redis-stat [HOST[:PORT] ...] [INTERVAL [COUNT]]

-a, --auth=PASSWORD 设置密码
-v, --verbose 显示更多信息
--style=STYLE 输出编码类型: unicode | ascii
--no-color 取消ANSI颜色编码
--csv=OUTPUT_CSV_FILE_PATH 以CSV格式存储结果
--es=ELASTICSEARCH_URL 把结果发送到 ElasticSearch: [http://]HOST[:PORT][/INDEX] //启动web端展示

--server[=PORT] 运行redis-stat的web server (默认端口号: 63790)
--daemon 使得redis-stat成为进程。必须使用 --server 选项
--version 显示版本号
--help 显示帮助信息

"

redis-stat运行命令行监控

"

```
redis-stat
redis-stat 1
redis-stat 1 10
redis-stat --verbose
redis-stat localhost:6380 1 10
redis-stat localhost localhost:6380 localhost:6381 5
redis-stat localhost localhost:6380 1 10 --csv=/tmp/outpu.csv --verbose
```

"

Web界面中的redis-stat

“

```
redis-stat --server
redis-stat --verbose --server=8080 5

# redis-stat server can be daemonized
redis-stat --server --daemon

# Kill the daemon
killall -9 redis-stat-daemon
```

”

redis-stat效果图

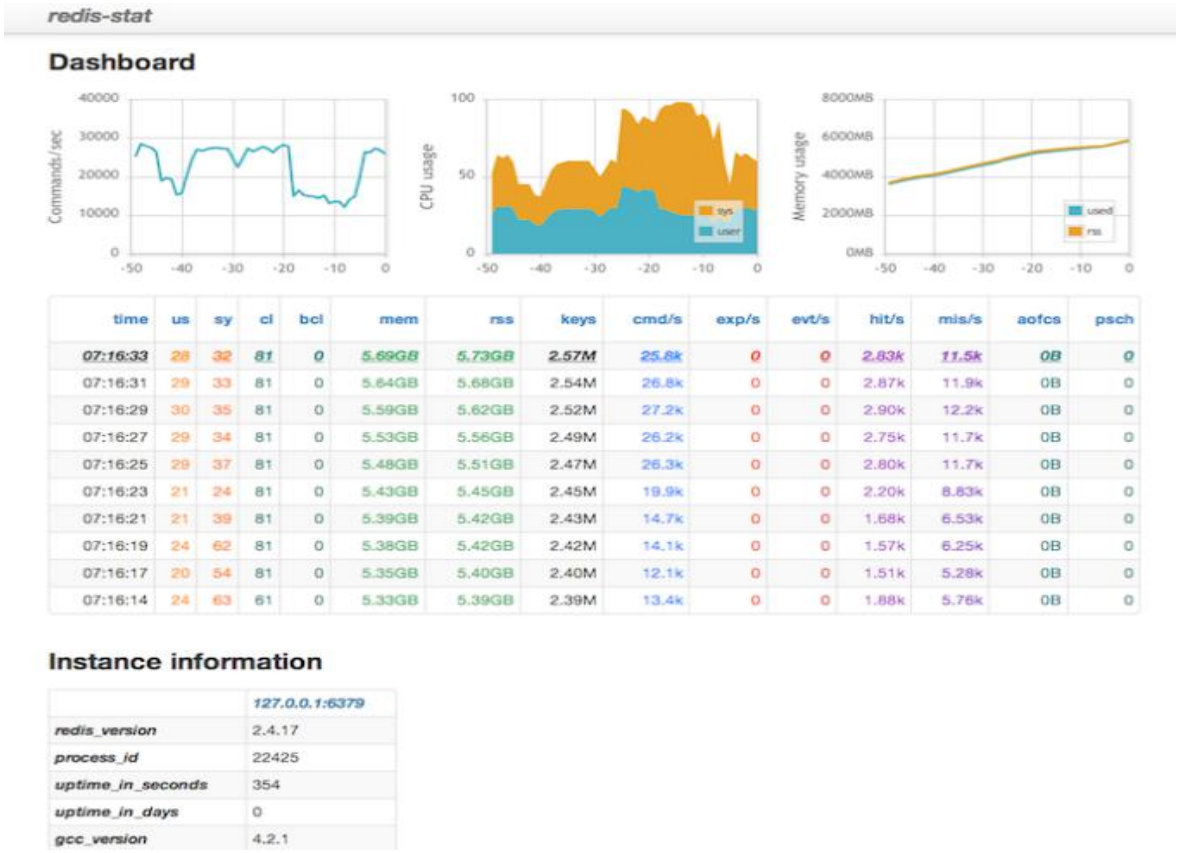
>> Listening on 0.0.0.0:8081, CTRL+C to stop

redis_version	4.0.11
redis_mode	cluster
process_id	17006
uptime_in_seconds	615312
uptime_in_days	7
role	slave
connected_slaves	0
aof_enabled	1
rdb_bgsave_in_progress	0
rdb_last_save_time	1535713931

time	us	sy	cl	bcl	rej/s	mem	rss	frag	keys	cmd/s	cmd	exp/s	exp	evt/s	evt	hit%/s	hit/s	hit	mis/s	mis	aofcs	aofbs	chsv	psch	psp
22:31:58	-	-	19	0	-	8.12MB	10.2MB	1.26	3.34k	-	7.90M	-	0	-	0	-	-	421k	-	3.34k	12.0MB	1.31MB	0	1	0
22:32:03	0	0	19	0	0	8.04MB	10.2MB	1.27	3.34k	11.4	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:08	0	0	19	0	0	8.12MB	10.2MB	1.26	3.34k	10.8	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:13	0	0	19	0	0	8.16MB	10.2MB	1.26	3.34k	13.2	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:18	0	0	19	0	0	8.12MB	10.2MB	1.26	3.34k	10.8	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:23	0	0	19	0	0	8.12MB	10.2MB	1.26	3.34k	11.2	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:28	0	0	19	0	0	8.12MB	10.2MB	1.26	3.34k	11.2	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:33	0	0	19	0	0	8.16MB	10.2MB	1.26	3.34k	10.8	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:38	0	0	19	0	0	8.16MB	10.2MB	1.26	3.34k	11.4	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:43	0	0	19	0	0	8.08MB	10.2MB	1.27	3.34k	12.8	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0
22:32:48	0	0	19	0	0	8.08MB	10.2MB	1.27	3.34k	11.4	7.90M	0	0	0	0	-	0	421k	0	3.34k	12.0MB	1.31MB	0	1	0

web效果图

" http://你的Redis IP:63790



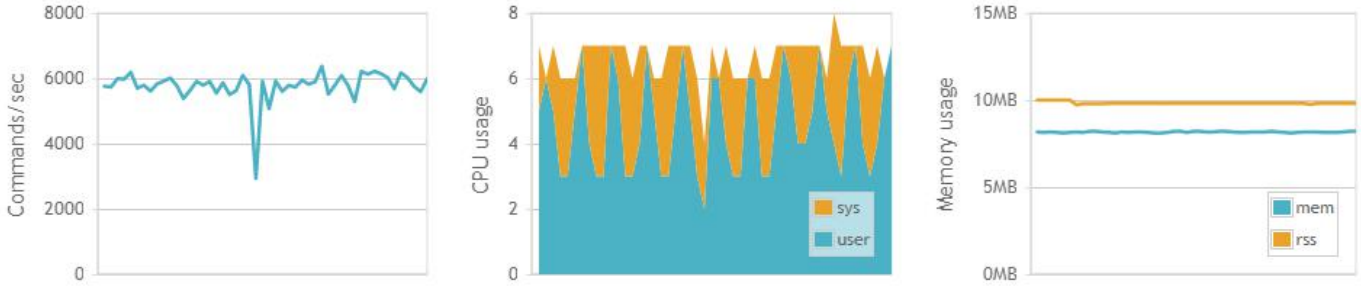
time	us	sy	cl	bcl	mem	rss	keys	cmd/s	exp/s	evt/s	hit/s	mis/s	aofcs	psch
07:16:33	28	32	81	0	5.69GB	5.73GB	2.57M	25.8k	0	0	2.83k	11.5k	0B	0
07:16:31	29	33	81	0	5.64GB	5.68GB	2.54M	26.8k	0	0	2.87k	11.9k	0B	0
07:16:29	30	35	81	0	5.59GB	5.62GB	2.52M	27.2k	0	0	2.90k	12.2k	0B	0
07:16:27	29	34	81	0	5.53GB	5.56GB	2.49M	26.2k	0	0	2.75k	11.7k	0B	0
07:16:25	29	37	81	0	5.48GB	5.51GB	2.47M	26.3k	0	0	2.80k	11.7k	0B	0
07:16:23	21	24	81	0	5.43GB	5.45GB	2.45M	19.9k	0	0	2.20k	8.83k	0B	0
07:16:21	21	39	81	0	5.39GB	5.42GB	2.43M	14.7k	0	0	1.68k	6.53k	0B	0
07:16:19	24	62	81	0	5.38GB	5.42GB	2.42M	14.1k	0	0	1.57k	6.25k	0B	0
07:16:17	20	54	81	0	5.35GB	5.40GB	2.40M	12.1k	0	0	1.51k	5.28k	0B	0
07:16:14	24	63	61	0	5.33GB	5.39GB	2.39M	13.4k	0	0	1.88k	5.76k	0B	0

"

web效果图

--monitor write

Dashboard



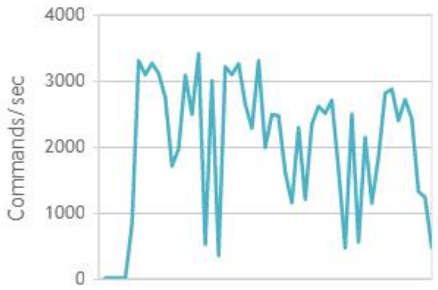
time	us	sy	cl	bcl	rej/s	mem	rss	frag	keys	cmd/s	cmd	exp/s	exp	evt/s	evt	hit%/s	hit/s	hit	mis/s	mis	aofcs	aofbs	chsv	psch	psp
22:26:26	7	0	19	0	0	8.21MB	9.82MB	1.20	3.34k	5.98k	2.45M	0	0	0	0	100	1.99k	765k	0	3.34k	14.5MB	1.32MB	53.4k	1	0
22:26:21	6	0	19	0	0	8.21MB	9.82MB	1.20	3.34k	5.59k	2.42M	0	0	0	0	100	1.86k	755k	0	3.34k	12.6MB	1.32MB	43.5k	1	0
22:26:16	4	3	19	0	0	8.17MB	9.82MB	1.20	3.34k	5.76k	2.39M	0	0	0	0	100	1.92k	746k	0	3.34k	10.9MB	1.32MB	34.2k	1	0
22:26:11	3	3	19	0	0	8.14MB	9.80MB	1.21	3.34k	6.02k	2.36M	0	0	0	0	100	2.00k	736k	0	3.34k	9.20MB	1.32MB	24.6k	1	0
22:26:06	4	3	19	0	0	8.14MB	9.80MB	1.20	3.34k	6.17k	2.33M	0	0	0	0	100	2.05k	726k	0	3.34k	7.38MB	1.32MB	14.6k	1	0
22:26:01	7	0	19	0	0	8.14MB	9.80MB	1.20	3.34k	5.68k	2.30M	0	0	0	0	100	1.89k	716k	0	3.34k	5.51MB	1.32MB	4.35k	1	0
22:25:56	6	1	19	0	0	8.18MB	9.80MB	1.20	3.34k	6.02k	2.28M	0	0	0	0	100	2.00k	707k	0	3.34k	3.79MB	1.32MB	115k	1	0
22:25:51	3	4	19	0	0	8.18MB	9.75MB	1.19	3.34k	6.14k	2.25M	0	0	0	0	100	2.04k	697k	0	3.34k	1.97MB	1.32MB	105k	1	0
22:25:46	4	4	19	0	0	8.18MB	9.82MB	1.20	3.34k	6.22k	2.21M	0	0	0	0	100	2.07k	687k	0	3.34k	62.9MB	1.43MB	95.2k	1	0
22:25:41	5	1	19	0	0	8.14MB	9.82MB	1.21	3.34k	6.13k	2.18M	0	0	0	0	100	2.04k	676k	0	3.34k	61.0MB	1.43MB	84.8k	1	0

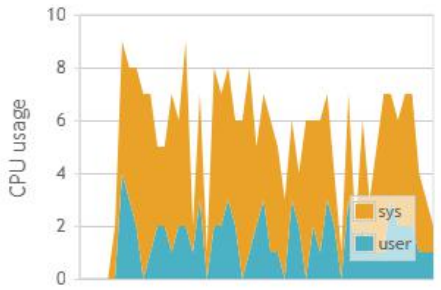
web效果图

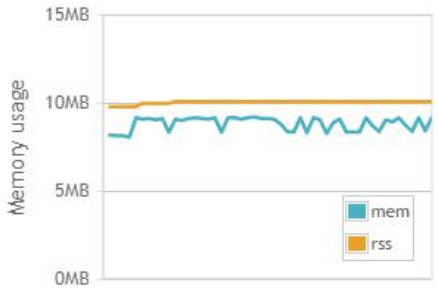
--monitor read

"

Dashboard

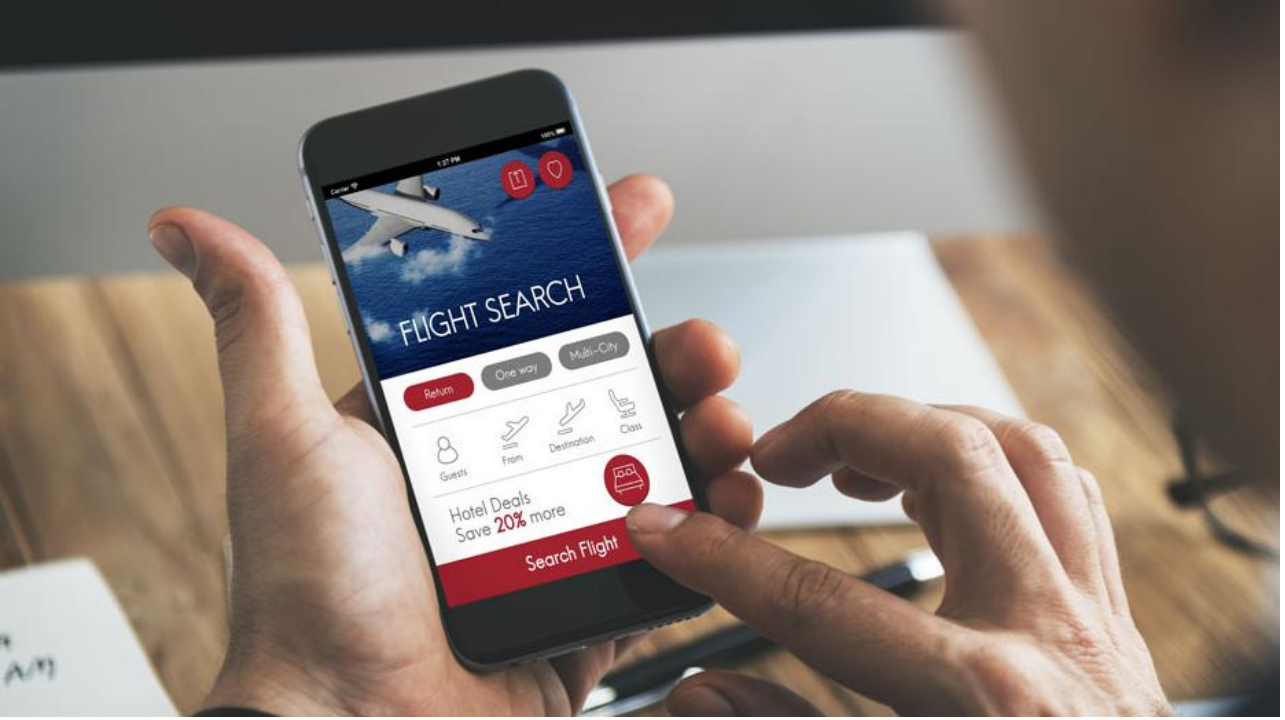






time	us	sy	cl	bcl	rej/s	mem	rss	frag	keys	cmd/s	cmd	exp/s	exp	evt/s	evt	hit%/s	hit/s	hit	mis/s	mis	aofcs	aofbs	chsv	psch	psp
23:08:46	1	1	29	0	0	9.14MB	10.1MB	1.10	3.34k	474	8.08M	0	0	0	0	100	464	2.97M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:41	1	2	29	0	0	8.40MB	10.1MB	1.20	3.34k	1.23k	8.08M	0	0	0	0	100	1.22k	2.96M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:36	1	3	29	0	0	9.14MB	10.1MB	1.10	3.34k	1.32k	8.08M	0	0	0	0	100	1.31k	2.96M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:31	2	5	29	0	0	8.36MB	10.1MB	1.20	3.34k	2.42k	8.07M	0	0	0	0	100	2.41k	2.95M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:26	2	5	29	0	0	8.71MB	10.1MB	1.16	3.34k	2.72k	8.06M	0	0	0	0	100	2.71k	2.94M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:21	2	4	29	0	0	9.14MB	10.1MB	1.10	3.34k	2.39k	8.04M	0	0	0	0	100	2.38k	2.93M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:16	3	4	29	0	0	8.91MB	10.1MB	1.13	3.34k	2.87k	8.03M	0	0	0	0	100	2.86k	2.91M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:11	1	6	29	0	0	9.03MB	10.1MB	1.12	3.34k	2.81k	8.02M	0	0	0	0	100	2.79k	2.90M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:06	1	4	29	0	0	8.36MB	10.1MB	1.20	3.34k	1.84k	8.00M	0	0	0	0	100	1.83k	2.89M	0	3.34k	10.6MB	1.57MB	4.87k	1	0
23:08:01	1	2	29	0	0	8.68MB	10.1MB	1.16	3.34k	1.15k	7.99M	0	0	0	0	100	1.13k	2.88M	0	3.34k	10.6MB	1.57MB	4.87k	1	0

"



redis-live monitor cluster

Monitoring the health and performance of clusters

Analysis

redis-live monitor Single node

Monitoring the health and performance status of a single node

Analysis

redis-live 简介

"

Redis Live is a dashboard application with a number of useful widgets. At it's heart is a monitoring script that periodically issues INFO and MONITOR command to the redis instances and stores the data for **analytics**.

"长时间运行对Redis性能有所影响"

"

redis-live 部署

1.Github地址：https://github.com/nkrode/RedisLive

2.依赖tornado：pip install tornado
依赖redis.py：pip install redis
依赖python-dateutil：pip install python-dateutil

3.下载：git clone https://github.com/kumarnitin/RedisLive.git

4.配置文件：cp /RedisLive/src/redis-live.conf.example redis-live.conf

redis-live 配置文件

vim redis-live.conf

```
{
        "RedisServers":
        [
                {
                        "server": "154.17.59.99",    //监控的IP地址
                        "port" : 6379          // 监控的端口
                },
                ..... //可以多写几个IP地址和端口，可将集群的节点都填写
                {
                        "server": "localhost",   //监控机的IP地址
                        "port" : 6380,  //监控机的端口
                        "password" : "some-password"  //监控机的 口令
                }
        ],

        "DataStoreType" : "redis",

        "RedisStatsServer":
        {
                "server" : "ec2-184-72-166-144.compute-1.amazonaws.com", //web地址
                "port" : 6385  //web端口
        },

        "SqliteStatsStore" :
        {
                "path":  "to your sql lite file"
        }
}
```

启动RedisLive

1.启动监控脚本，监控120秒，duration参数是以秒为单位

2. ./redis-monitor.py --duration=120

3.启动webserver。
RedisLive使用tornado作为web服务器，所以不需要单独安装服务器
Tornado web server 是使用Python编写出来的一个极轻量级、高可伸缩性和非阻塞IO的Web服务器软件

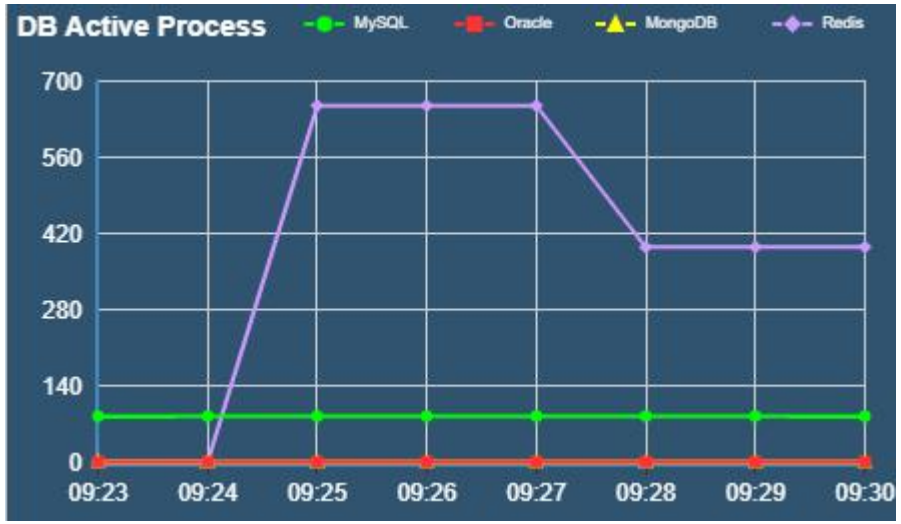
4.启动web：./redis-live.py

RedisLive web端效果图

lepus monitor
redis single or cluster

lepus 3.8 beta版本

下载地址：http://www.lepus.cc/soft/18



monitor health and performance and
Abnormal alarm

部署lepus简单介绍

"

1.下载xampp依赖环境
https://downloadsapachefriends.global.ssl.fastly.net/xampp-files/5.6.37/xampp-linux-x64-5.6.37-0-installer.run?from_af=true
Xampp帮助文档：https://www.apachefriends.org/faq_linux.html
2. chmod +x xampp-linux-x64-5.6.37-0-installer.run
3../xampp-linux-x64-1.8.2-5-installer.run
4. /opt/lampp/lampp start
5.ln -s /opt/lampp/lampp /etc/init.d/lampp
6.wget http://cdn.lepus.cc/cdncache/software/MySQLdb-python.zip
7.wget http://cdn.lepus.cc/cdncache/software/redis-py-2.10.3.tar.gz
8.tar zxvf redis-py-2.10.3.tar.gz
9.python setup.py install
10.python test_driver_redis.py

"

lepus 易操作

”


保存
列表
服务器
监控开关
告警项目
主机
端口
密码
标签
监控
发送邮件
发送短信
连接客户端数
命令执行数
阻塞客户端数
主机
6379
密码
标签
开
开
开
开
开
开

”

lepus 效果图

连接	角色	运行时间	版本	已连接	已拒绝	总计	已过期	已删除	命中次数	丢失次数	图表
成功	M	63 天	4.0.8	525	0	100382	14386	0	27282	18331	
成功	M	109 天	4.0.8	525	0	209367	15441	0	78339	27103	
成功	M	63 天	4.0.8	525	0	100103	12880	0	31387	16412	
成功	M	107 天	4.0.8	153	0	251783	104295	0	6188009	15658	
成功	S	107 天	4.0.8	141	0	253647	0	0	6259743	1230	
成功	M	31 天	4.0.8	154	0	82426	100015	0	90305	13484	
成功	M	63 天	4.0.8	587	0	199437	2152474	0	6355470	185276	
成功	S	36 天	4.0.8	574	0	80031	0	0	8182792	1171166	

THANKS