运维监控 redis 集群的工具

——分享自_2018 年 9 月 1 日_谷阳_首汽_DBA

总体，分四个模块

第一部分:首先简单介绍一些 redis 集群

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。

"

redis 集群特点，大体上分 5 点
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。

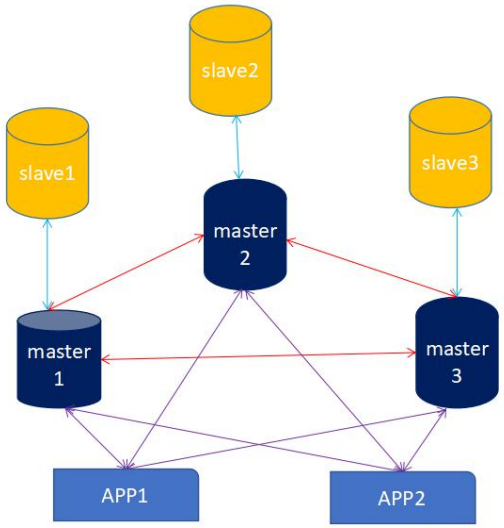
3、客户端与 redis 节点直连,不需要中间 proxy 层.客户端不需要连接集群所有节点,连接集群中任何一个可用节点即可。

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。
redis-cluster3.0 版本的经典架构拓扑图

集群拓扑图

"



"

以前：

1. redis cpu 使用率>80%， 拆分 redis 实例，修改代码，指向新的 redis 实例。
2 redis 内存使用超过标准，继续拆分实例！
3 redis 流量增长，拆!
4. 单实例的高可用问题。

现在：

只需要 分配一组新的 redis 实例 加入 cluster， 迁移 slot 即可解决 资源 使用率问题。

Redis-cluster 缺点：

1. 无法查看 几号 slot 里 存有什么类型的 keys，只能查看实例里存有多少 slot 号。
2. 当 redis-cluster 中 一组节点全部挂掉， 将 丢失 指向 已经挂掉节点的 keys。（根据 crc16 算法）

Redis-cluster 无法处理的问题：

A. 在遇到 被爬虫，强刷部分模块， 容易出现 redis 线程上涨，堵塞 响应请求， 其根因是 存储的 redis key 不合 理.

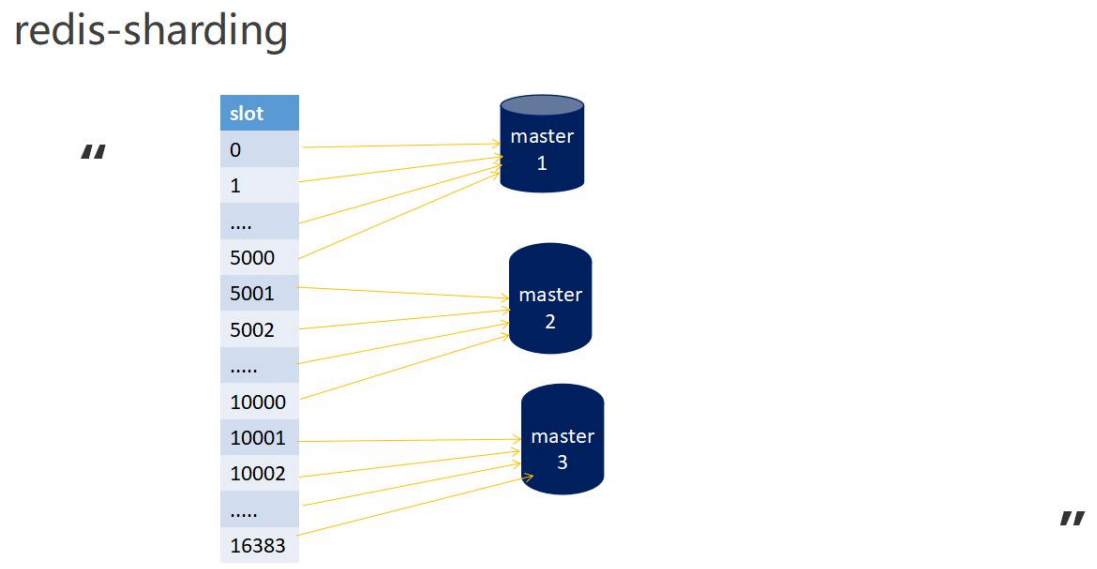
B. 部分 hash key 存的过大，单个 key 里 存储数据 超过 1W。降低 redis 响应时间。

C. 程序逻辑问题， 导致 redis 实列 频繁刷新 部分业务 key.

D. 程序设计漏洞。

伴随着，技术场景的不断发展，和业务场景对“快速响应”的要求，因此，redis-cluster 应运而生了。

下面，简单分析一下，redis 集群种，slot 与 nodes 的关系



分片（Sharding）

集群将整个数据库分为 16384（2 的 14 次方）个槽（slot）

每个主节点可以负责处理 0 个至 16384 个槽

注意：集群只有在所有槽位均有主节点处理时，才能进入上线状态并处理数据命令

那么，问题来了。集群中，是如何将“故障机制”进行转嫁的？

首先，看“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；
故障转移的原理是:
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。

那么，主从切换，是如何完成的呢？

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。

自此，redis-cluster 主从切换的原理，大体上分享到这里，大家有疑问的话，欢迎多多交流。

虽然是集群模式，但不一定能保证，主从转嫁，完全的 OK，那还有什么方案？能进一步改善吗？

是有的。接下来，进入第二部分。sentinel（哨兵模式）

第二部分。sentinel（哨兵模式）

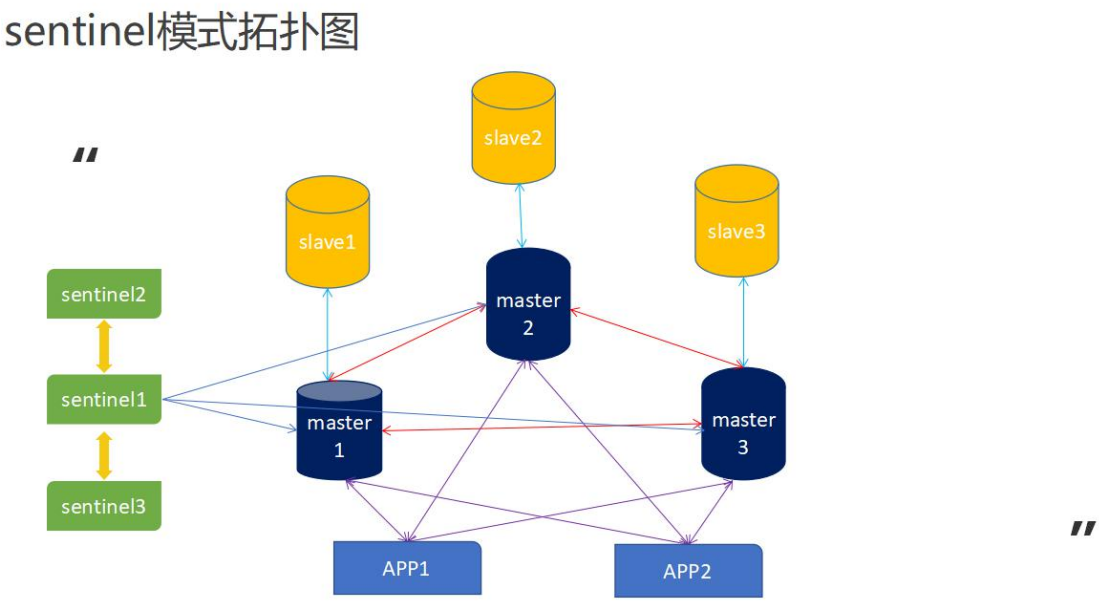
我们，先来看看哨兵模式的特点

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。

首先，在这里，说明。sentinel 可以作分布式的

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。

让我们来看看，哨兵模式+redis 集群的拓扑图



sentinel 可以分布在不同的服务上，用于做 HA 架构，负责对 redis 集群的每个 nodes 进行监控和负责主从切换工作.这里，要重点说一下，sentinel 的配置文件参数，对整个集群的影响.

1. 原始配置文件：/usr/local/redis-4.0.11/sentinel.conf。redis-4.0 中，自带了，sentinel 的配置文件.核心参数有 port 、dir、monitor、down-after-milliseconds parallel-syncs failover-timeout failover-timeout

逐一介绍一下，这些参数

port 26379 //默认端口号

dir /tmp //用于存放 sentinel 日志信息文件等

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。

首先，介绍个概念 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。

第三部分：监控 redis 維度

1.服务器存活监控

2.redis server 监控采集数据

2. redis cluster 监控
服务器系统数据采集
Redis Server 数据采集
Redis 响应时间数据采集
Redis 监控 Screen
一、服务器存活监控
1>ping 监控告警
2>CPU
3>丢包率
二、Redis Server 监控数据采集
ping,info all, slowlog get/len/reset/cluster info/config get
Redis 存活监控

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。

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

连接数使用率(connected_clients_pct): 连接数使用百分比，通過(connected_clients/maxclients)计算；如果达到 1，redis 开始拒绝新连接创建，告警
拒绝的连接个数(rejected_connections): redis 连接个数达到 maxclients 限制，拒绝新连接的个数。

新创建连接个数 (total_connections_received): 如果新创建连接过多，过度地创建和销毁连接对性能有影响，说明短连接严重或连接池使用有问题

edis 分配的内存大小 (used_memory): redis 真实使用内存，不包含内存碎片
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

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
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

redis 处理的命令数 (total_commands_processed): 监控采集周期内的平均 qps

请求键的命中率 (keyspace_hit_ratio):使用 keyspace_hits/(keyspace_hits+keyspace_misses)计算所得

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

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

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

集群的节点数 （cluster_known_nodes）

大体上了解，我们所需要监控的项目

那么，接下来，该如何监控呢？写脚本？做平台？我今天给大家分享给大家的是，自动化脚本监控+web 端呈现

有三个工具平台，第一个是 redis-stat（开箱即用）。第二是，python 写的 redis-live，能够做 redis 监控和性能分析的。第三个是，常用的 redis 监控平台，当然也是 python 写的，lepus，天兔。当然，天兔可以结合 go 语言来改写的。有兴趣的小伙伴，可以后续沟通。

首先，聊聊，开箱即用的，redis-stat。

Redis-stat（ruby）

redis-stat 是 ruby 语言撰写的，熟悉 ruby 语言的小伙伴，可以后期改造的。下载地址 很方便，在 github 既可以下载了。而且，很贴心，可以 web 端呈现。当然，不足之处，就是单节点监控。针对于集群的话，需要二次开发。不过，动态分析集群中的任意节点，也是不错的。

安装步骤，还是简单明了

1.yum install ruby

sudo apt-get install ruby-full

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

3.gem install redis-stat

使用起来，也比较方便

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:

很温馨的使用了 ANS 编码处理技术，所以，可以颜色鲜明的呈现

web 启动命令

[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

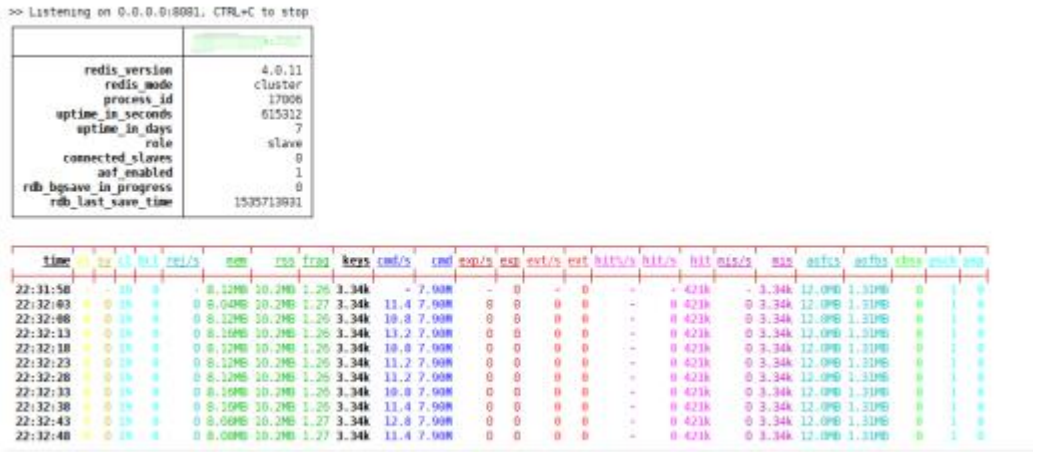
--version 显示版本号

--help 显示帮助信息

redis-stat 运行命令行监控

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

让我们来看看效果图



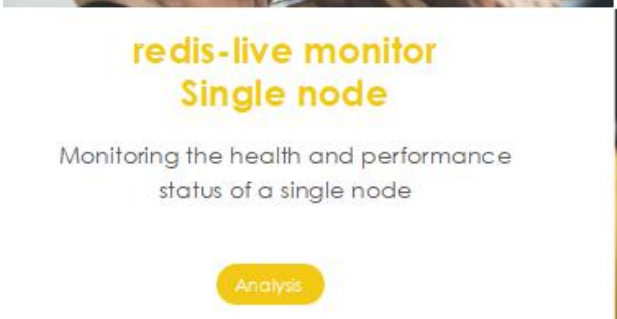
以下是我们做 项目测试节点的 监控呈现图。温馨提示，web 启动是 http://你的 Redis IP:63790。默认端口是 63790。如果，本地 redis 的端口，不是的话，需要指定本地的端口

web效果图

--monitor write

"

接下来，来说一说 redis-live



这个工具的擅长的地方，是做分析

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.

温馨提示　“长时间运行对 Redis 性能有所影响”部署时，还是很友好的

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

在此，重点说一下配置文件

```
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"
        }
}
```

上半部分是，监控节点的信息
中间是监控机的信息
server 部分是 启动 web 的信息
如果，仅需监控单点的话，仅需配置单点信息
如果是集群的话，可以填写集群中所有节点的信息
让我来看看，如何启动
1.启动监控脚本，监控 120 秒，duration 参数是以秒为单位

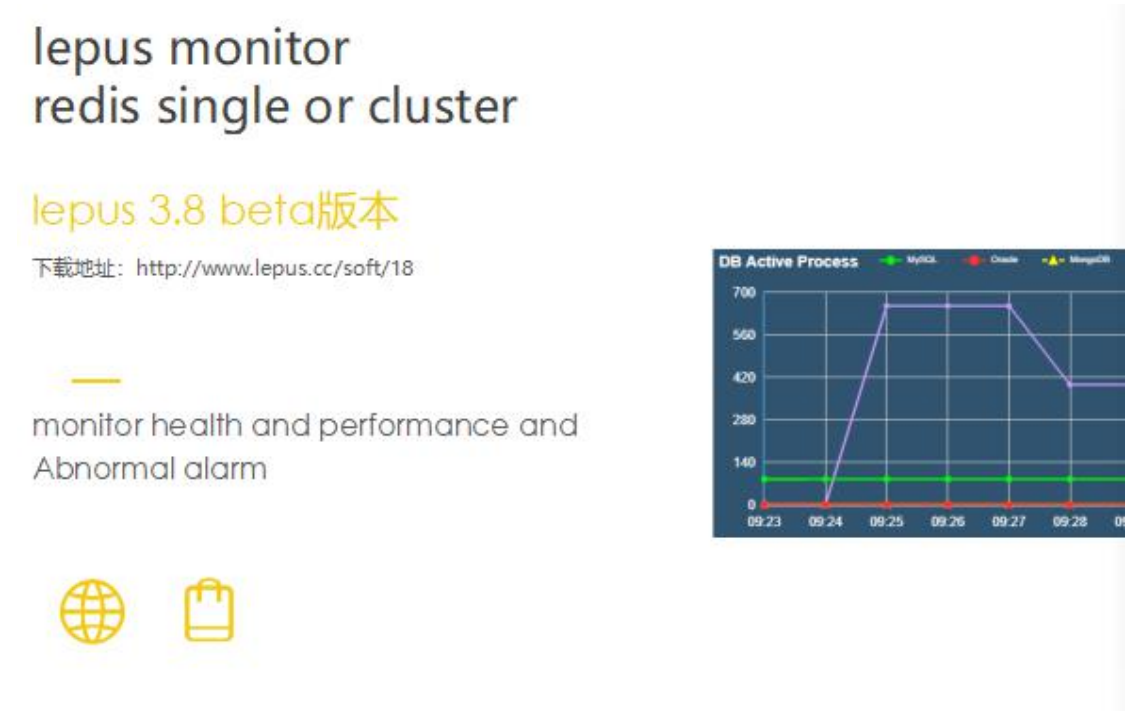
2. ./redis-monitor.py --duration=120

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
让我们来看 web 效果图

RedisLive web端效果图

最后，让我来看 天兔 的监控效果



天兔的优势在于，结合了，常规监控 redis（单和集群）性能和健康指标的动态呈现，并将预警机制，可以通过短信或者邮件等发给接收信息的相关人员。部署也非常简单，集成了 lamp 环境包，可以形成“一键安装”，官网上也有具体操作步骤的相关介绍。日常管理，也相对易操作

添加节点和删除节点，异操作。让我们来看看健康指标监控图。

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	

在此，简单总结以下，分析动态分析单点的 redis-stat，开箱即用，很方便，缺点时监控集群，需要二次开发。redis-live，做性能分析是一个不错的选择，缺点是不能做实时动态分析，也需要二次开发。lepus 的优点是监控了常规指标，能添加报警通道，短信和邮件均可以，当然微信平台，需要稍加改造。可以后续，改造为 go 语言。此次，分享第一站，暂时“休息”，稍后，我会把分享内容整理成文档，贴到群里。当然，自动化运维，需要不断修缮。期待和小伙伴们，一起开创，更好，更敏捷的工具平台，一起学习，一起进步。

感谢大家！

Ask：想问下你们监控对比 zabbix 如何？

An：各有优势把，当然，问题是，由于 redis 是内存数据库，zabbix 底层存储是 mysql（目前版本是 mariadb），收集信息久了，可能会存在误报或者相应慢，不够灵敏。当然，如果，用 zabbix 做 redis 周期性 性能分析，不太友好。建议，因为 zabbix 优势在于，它可以监控各种平台维度的服务，可以以 zabbix 为基础。Erving:

然后，做 redis 专项分析（问题排查和新项目压测等）可以用 redis-stat、redis-live