**京东618如何支持容器百万级域名解析服务？**

2017-06-19 20:24 阅读 725 评论 0

社区广播：运维派（Yunweipai.com）是国内最早成立的IT运维社区，欢迎大家[**投稿**](http://www.yunweipai.com/tougao)，让运维人不再孤寂的成长！



作者:鲍永成、陈书刚

编辑:木环

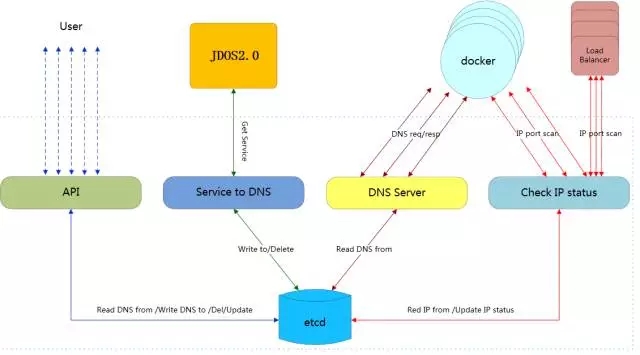
InfoQ 报道京东大促技术已有三年的时间，这三年的陪伴见证了京东技术的成长。2015 年京东开启容器技术，拥有数千实例；2016 年，该数字上升至 15 万；2017 年，容器实例规模继续稳步上升并开源数个项目。

随着京东业务的高速增长，以及 JDOS2.0 的线上大规模运营，进而容器集群的编排成为常态，Pod 失效也成为常态，RS（Replication Set）在处理失效 Pod 时候会带来 IP 的变化。这样容器之间基于 IP 相互访问就有可能存在问题。所以一个强大的能支持百万级 hostname 域名解析服务，可以很好地解决这个问题。

**一、系统概述**

本文介绍的 DNS 命名为 ContainerDNS，作为京东商城软件定义数据中心的关键基础服务之一，具有以下特点：

* 分布式，高可用
* 自动发现服务域名
* 后端探活
* 易于维护、易于动态扩展llllllll,\* 容器化部署



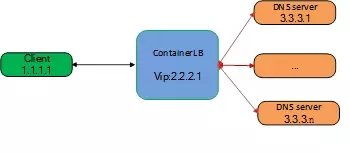
图一 ContainerDNS 架构图

   IP status check 模块用于对系统中域名所对应的 IP 做探活处理，数据状态也会存入到 etcd 数据库中。如果某一个域名对应的某一个 IP 地址不能对外提供服务，DNS Server 会在查询这个域名的时候，将这个不能提供服务的 IP 地址自动过滤掉。λ    User API 组件提供 restful API，用户可以创建自己的域名信息，数据同样保持到 etcd 数据库中。λ    Service to DNS 组件是 JDOS 集群与 DNS Server 的中间环节，会实时监控 JDOS 集群的服务的创建，将服务转化为域名信息，存入 etcd 数据库中。λ    DNS Server 用于提供 DNS 查询服务的主体，目前支持了大部分常用的查询类型（A、AAAA、SRV、NS、TXT、MX、CNAME 等）。λContainerDNS 包括四大组件 DNS Server、Service to DNS 、User API 、IP status check。这四个组件通过 etcd 集群结合在一起，彼此独立，完全解耦，每个模块可以单独部署和横向扩展。

**二、系统设计与实现**

**（1）DNS Server**

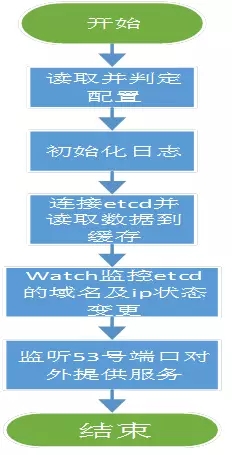
DNS Server 是提供 DNS 的主体模块，系统中是挂载在项目 ContainerLB（一种基于 DPDK 平台实现的快速可靠的软件网络负载均衡系统）之后，通过 VIP 对外提供服务。结构如下：



图二 DNS Server 与 ContainerLB

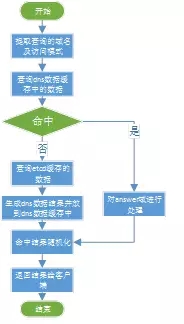
如上图所示，DNS Server 通过 VIP 对外提供服务，通过这层 LB 可以对 DNS Server 做负载均衡，DNS Server 的高可用、动态扩展都变得很容易。同时 DNS Server 的数据源依赖于 etcd 数据库，所以对 DNS Server 的扩展部署十分简单。由于 etcd 是一种强一致性的数据库，这也有效保障挂在 LB 后面的 DNS Server 对外提供的数据一致性。

DNS Server 作为 JDOS 集群的 DNS 服务，所以需要把服务器的地址传给容器。我们知道 JDOS 的 POD 都是由 JDOS Node 节点创建的，而 POD 指定 DNS 服务的地址和域名后缀。最终体现为 Docker 容器的 /etc/resolv.conf 中。

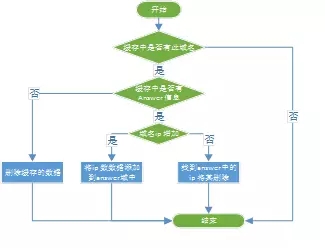


DNS Server 的启动过程

DNS Server 首先根据用户的配置，链接 etcd 数据库，并读取对应的域名信息放在程序的缓存中。然后启动 watch 监听 etcd 的变化，同步数据库与缓存中的数据。新的 DNS 请求不用在查询 etcd 数据库直接使用缓存中的数据，从而提高响应的速度。启动后监听用户配置的端口（默认 53 号），对收到的数据包进行处理。同时查出过得结果会缓存的 DNS-Server 的内存缓存中，对于缓存的数据不老化删除，就是说查询过的域名会一直在缓存中以提高查询的速度，从而达到很高的响应性能。如果域名信息发生变化，DNS Server 通过监听 etcd 随时感知这种变化，从而更新缓存中的数据，从而提供很好的实时性。测试发现，从发生变化到能查出变更预期的结果一般在 20ms 以内，坏的情况不超过 50-60ms。



上图是 DNS Server 响应一次查询的过程。首先根据域名和查询的类型生成一个数据缓存的索引，然后查询 DNS 数据缓存如果命中，简单处理返回给用户。没有命中从数据库查询结果，并将返回的结果插入到数据缓存中，下次查询直接从缓存中取得，提高响应速度。为了进一步提高性能，缓存的数据不会老化删除，只有到了缓存的数量限制才会随机删除一些释放空间。不删除缓存，缓存中的数据和实际的域名数据的一致性就是一个关键的问题。我们采用 etcd 监控功能实时抓取变更，从而更新缓存的数据，经过几个星期的不停地循环，增、删、改、查域名，近 10 亿次测试，未出现数据不一致的情况。下面是 DNS Server 监控到域名信息变化的处理流程。



下面是 DNS Server 的配置文件：



其中 DNS 域主要是对 DNS 的配置，DNS-domains 提供可查询的域名的 zone，支持多组用 % 分隔。ex-nameServers 如果不是配置的域名，DNS Server 会将请求转发到这个地址进行解析。解析的结果再通过 DNS Server 转给用户。inDomainServers 选择做已知域名 zone 的转发功能。首先如果访问的域名匹配到 inDomainServers， 则交给 inDomainServers 指定的服务器处理，其次如果匹配到 DNS-domains 则查询本地数据，最后如果都不匹配则交给 ex-nameServers 配置的 DNS 服务器处理。IP-monitor-path 是用于和探活模块做数据交互的，系统中的 IP 状态会存在 etcd 此目录下。DNS Server 读取其中的数据，并监控数据的变化，从而更新自己缓存中的数据。

DNS Server 另外提供两个附加的功能，可以根据访问端的 IP 地址做不同的处理。Hold-one 如果使能，同一个客户端访问同一个域名会返回一个固定的 IP。而 random-one 相反，每次访问返回一个不同的 IP。当然这两个功能在一个域名对应多个 IP 的时候才能体现出来。为了提高查询速度，查询的域名会放在缓存中，cacheSize 用于控制缓存的大小，以防止内存的无限之扩张。DNS Server 由于采用的是 Go 语言，cache 被设计为普通的字典，字典的 key 就是域名和访问类型的组合生成的结果。

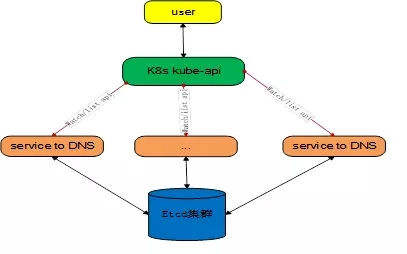
DNS Server 提供统计数据的监控，通过 restful API 用户可以读取 DNS 的历史数据，访问采用了简单的认证，密码通过配置文件配置。用户可以访问得到 DNS Server 启动后查询域名的总的次数、成功的次数、查询不到次数等信息。用户同样可以得到某一个域名的查询次数和最后一次访问的时间等有效信息。通过 DNS Server 统计信息，方便做集群的数据统计。效果如下：

http://tektea-img.b0.upaiyun.com/blog/2017/06/08.webp_.jpg

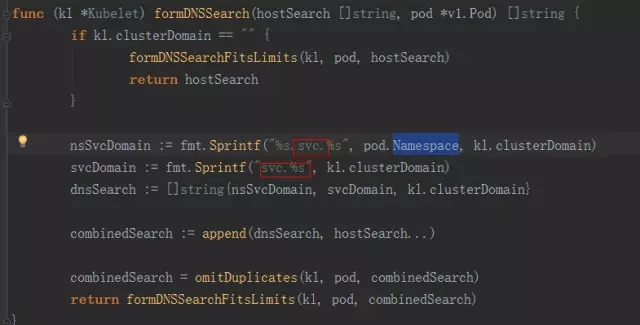
http://tektea-img.b0.upaiyun.com/blog/2017/06/1..webp_.jpg

**（2）Service to DNS**

这个组件的主要功能是通过 JDOS 的 JDOS-APIServer 的 watch-list 接口监控用户创建的 Service 和以及 endpoint 的变化，从而生成一条域名记录，并将域名记录导入到 etcd 数据库中。简单的结构如下图。Service to DNS 进程，支持多点冗余，防止单点故障。



Service to DNS 生成的域名主要目的是给 Docker 容器内部访问，域名的格式是 ServiceName.nameSpace.svc. clusterDomain。这个格式的要求和 JDOS 有密切的关系，我们知道 JDOS 创建 POD 的时候，传递数据生成容器的 resolv.conf 文件。下面是 JDOS 的代码片段及 Docker 容器的 resolv.conf 文件的内容。



http://tektea-img.b0.upaiyun.com/blog/2017/06/12.webp_11.jpg

可以看到域名采用的是 ServiceName.NameSpace.svc.clusterDomain 的命名格式，故而Service to DNS 需要监控 JDOS 集群的 Service 的变化，以这种格式生成相关的域名。由于系统对用户创建的服务会自动的创建 load-balance 的服务，所以域名的 IP 对应的是这个服务关联的 lb 的 IP，而 lb 的后端才是对应着的是真正提供服务的 POD。

**Service to DNS 进程有两种任务：分别做数据增量同步和数据全量同步。**

增量同步调用 JDOS-API 提供的 watch 接口，实时监控 JDOS 集群 Service 和 endpoint 数据的变化，将变化的结果同步到 etcd 数据库中，从而得到域名的信息。由于各种原因，增量同步有可能失败，比如操作 etcd 数据库，由于网络原因发生失败。正如此全量同步才显得有必要。全量同步是个周期性的任务，这个任务首先同步 JDOS-API 的 list 接口得到，集群中的 Service 信息，然后调用 etcd 的 get 接口得到 etcd 中存储域名数据信息，然后将两边的数据左匹配，从而保证 JDOS 集群中的 Service 数据和 etcd 的域名数据完全匹配起来。

另外，Service to DNS 支持多点部署的特性，所以有可能同时多个 Service to DNS 服务监听到 JDOS 集群数据的变化，从而引起了同时操作 etcd 的问题。这样不利于数据的一致性，同时对相同的数据，多次操作 etcd，会多次触发 etcd 的变更通知，从而使得 DNS Server 监听到一些无意义的变更。为此 etcd 的读写接口采用了 Golang 的 Context 库管理上下文，可以有效地实现多个任务对 etcd 的同步操作。比如插入一条数据，会首先判断数据是否存在，对于已经存在的数据，插入操作失败。同时支持对过个数据的插入操作，其中有一个失败，本次操作失败。配置文件如下：



其中 etcd-Server 为 etcd 集群信息，这个要与 DNS Server 的配置文件要一致。Host 字段用于区别 Service to DNS 的运行环境的地址，此数据会写到 etcd 数据库中，可以很方便看到系统运行了多少个冗余服务。IP-monitor-path 写入原始的 IP 数据供探活模块使用。JDOS-domain 域名信息，这个要和 DNS Server 保持一致，同时要和 JDOS 启动的 –cluster-domain 选项保持一致，数据才能被 Docker 容器正常的访问。JDOS-config-file 文件是 JDOS-API 的访问配置信息，包括认证信息等。

**（3）User API**

User API 提供 restful API，用户可以配置自己域名信息。用户可以对自己的域名信息进行增、删、改、查。数据结果会同步到 etcd 数据库中，DNS Server 会通过监听 etcd 的变化将用户的域名信息及时同步到 DNS Server 的缓存中。从而使得用户域名数据被查询。简单的配置如下：



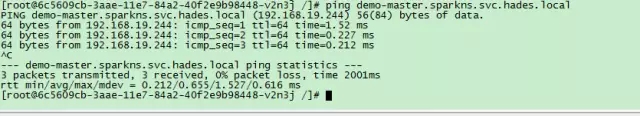
API-domains 支持多个域名后缀的操作，API-auth 用于 API 认证信息。其他信息 IP-monitor-path 等和 Service to DNS 模块的功能相同。具体的 API 的使用见

**（4）IP status check**

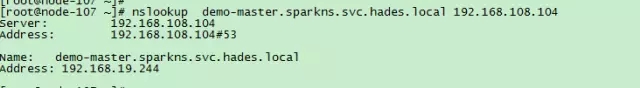
IP status check 组件对域名的 IP 进行探活，包括 DNS-scheduler 和 DNS-scanner 两个模块。DNS-scheduler 模块监控 Service to DNS 和 uer API 组件输入的域名 IP 的信息，并将相关的 IP 探活合理地分配给不用的 DNS-scanner 任务；DNS-scanner 模块负责对 IP 的具体的周期探活工作，并将实际的结果写到指定的 etcd 数据库指定的目录。DNS Server 组件会监听 etcd IP 状态的结果，并将结果及时同步到自己的缓存中。

**三、功能验证**

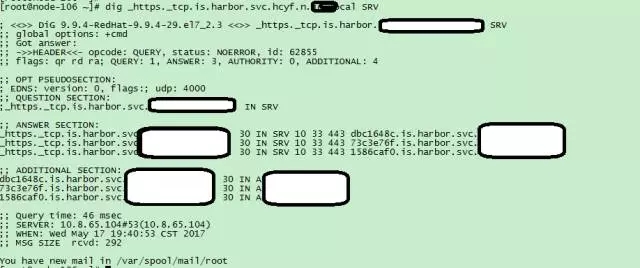
Docker 容器中验证



服务器验证：typeA

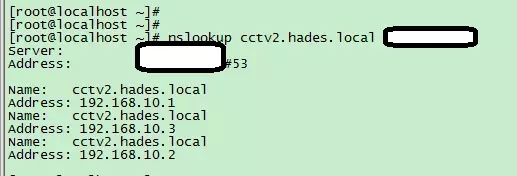


SRV 格式：

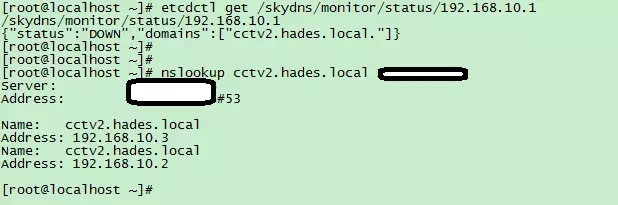


API 验证：

API



IP status check  验证：

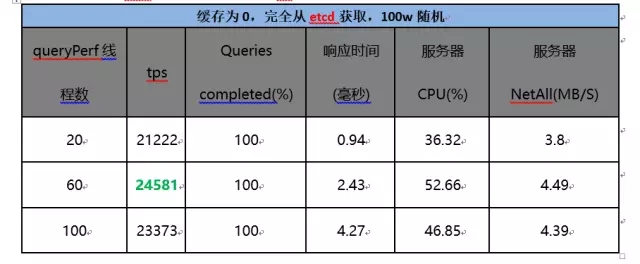


可以当 192.168.10.1 的状态变成 DOWN 后，查询 DNS Server，192.168.10.1 的地址不会再出现在返回结果中。

**性能优化ContainerDNS 的组件的交互依赖于 etcd，etcd 是由 Go 语言开发了。ContainerDNS 也采用 Go 语言。**

测试环境：CPU： Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHzNIC：  Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)测试工具：queryperf域名数据：1000W 条域名记录

性能数据：





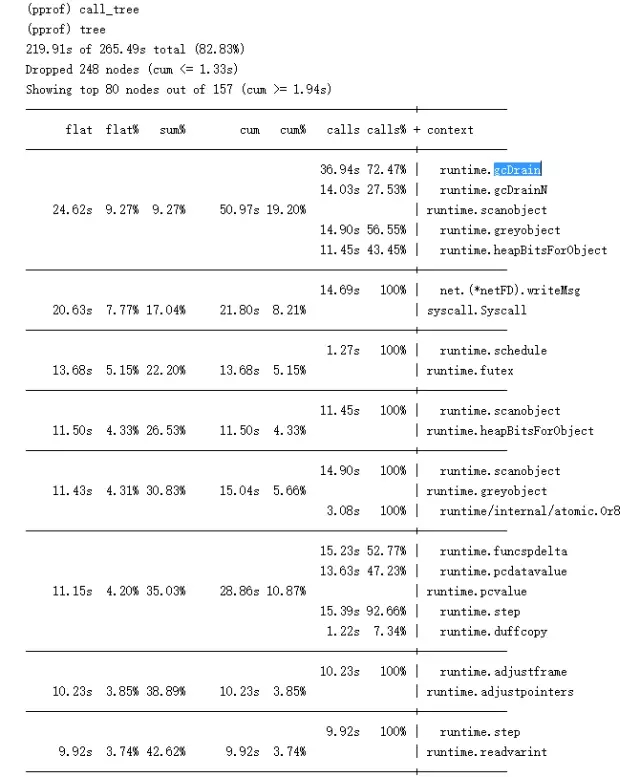


从上面三个表中可以清晰地看出，走 etcd 查询速度最慢，走缓存查询速度提升很多。同样，不存在缓存老化。所以程序优化的第一步，就是采用了全缓存，不老化的实现机制。就是说 DNS Server 启动的时候，将 etcd 中的数据全量读取到内存中，后期 watch 到 etcd 数据的变更，实时更新内存中的数据。全缓存一个最大的挑战就是 etcd 的数据要和缓存中的数据的一致性。为此代码中增加了很多对域名变更时，对缓存的处理流程。同时为了防止有 watch 不到的变更（一周稳定性测试 10 亿次变更，出现过一次异常），增加了周期性全量同步数据的过程，这个同步粒度很细，是基于域名的，程序中会记录每次域名变更的时间，如果发现同步的过程中这个域名的数据发生变化，这个域名本次不会同步，从而保证了缓存数据的实时性，不会因为同步导致新的变更丢失。

同时我们采集了每一秒的响应情况，发现抖动很大。而且全缓存情况下 queryperf 测试虽然平均能达到 10W TPS，但是抖动从 2W-14W 区间较大。



通过实验测试进程 CPU 损耗，我们发现 golang GC 对 CPU 的占用很大。



同时我们采集了 10 分钟内存的情况，如下



可以发现，系统动态申请了好多内存大概 200 多个 G，而 golang GC 会动态回收内存。

gc 18 @460.002s 0%: 0.030+44+0.21 ms clock, 0.97+1.8/307/503+6.9 ms cpu, 477->482->260 MB, 489 MB goal, 32 P

gc 19 @462.801s 0%: 0.046+50+0.19 ms clock, 1.4+25/352/471+6.3 ms cpu, 508->512->275 MB, 521 MB goal, 32 P

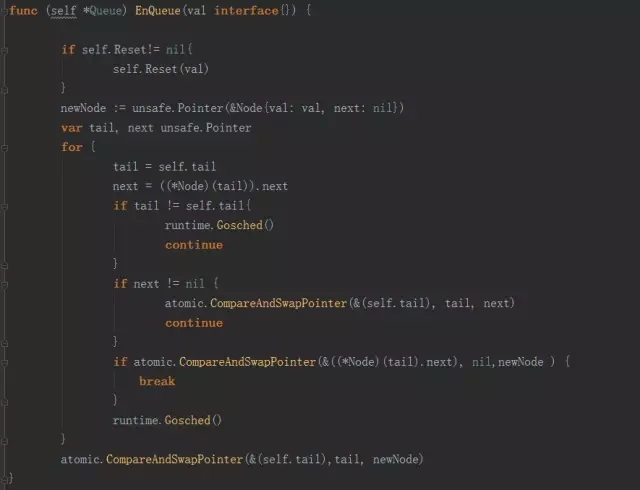
gc 20 @465.164s 0%: 0.067+50+0.41 ms clock, 2.1+64/351/539+13 ms cpu, 536->541->287 MB, 550 MB goal, 32 P

gc 21 @467.624s 0%: 0.10+54+0.20 ms clock, 3.2+65/388/568+6.2 ms cpu, 560->566->302 MB, 574 MB goal, 32 P

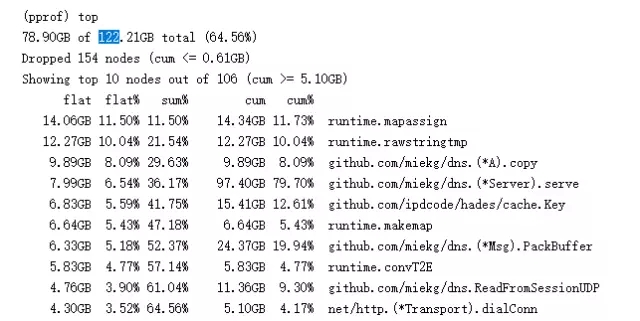
gc 22 @470.277s 0%: 0.050+57+0.23 ms clock, 1.6+73/401/633+7.3 ms cpu, 590->596->313 MB, 605 MB goal, 32 P

…

由于 golang GC 会 STW(Stop The World)，导致 GC 处理的时候有一段时间所有的协程停止响应。这也会引起程序的抖动。高级语言都带有 GC 功能，只要是有内存的动态使用，最终会触发 GC，而我们可以做的事是想办法减少内存的动态申请。为此基于 pprof 工具采集的内存使用的结果，将一些占用大的固定 size 的内存放入缓存队列中，申请内存首先从缓存重申请，如果缓存中没有才动态申请内存，当这块内存使用完后，主动放在缓存中，这样后续的申请就可以从缓存中取得。从而大大减少对内存动态申请的需求。由于各个协程都可能会操作这个数据缓存，从而这个缓存队列的设计就要求其安全和高效。为此我们实现了一个无锁队列的设计，下面是入队的代码片段。



目前对 512 字节的 msg 数据结构做了缓存。用 pprof 采集内存使用情况如下：



可以看到内存由原来的 200G 减少到 120G，动态申请内存的数量大大减小。

同时性能也有所提升：



10 分钟内的采集结果可以看出，抖动从原来的 2-10W 变成现在的 10-16W，抖动相对变小。同时 queryperf 测试每秒大概 14W TPS，比原来提高了 4W。

**写在最后**

本文主要介绍了 ContainerDNS 在实际环境中的实践、应用和一些设计的思路。全部的代码已经开源在 GitHub 上（详见 https://github.com/ipdcode/skydns ）。我们也正在做一些后续的优化和持续的改进。

**作者介绍**

鲍永成，京东商城 基础平台部技术总监。2013 年加入京东，负责京东容器集群平台（JDOS）研发，带领团队完成京东容器大规模落地战略项目，有效承载京东全部业务系统和 80% 数据库，特别在大促期间 scale up 秒级弹性应对高峰流量。目前聚焦在京东容器集群 JDOS 2.0 以及京东敏捷智能数据中心研发。服务过土豆网（TUDOU.COM），思科（CRDC）等，在分布式、虚拟化、容器、数据中心建设有丰富的实践经验。

陈书刚，京东商城基础平台部软件工程师，有着多年从事数通产品的开发、协议报文解析的工作的经验，目前主要从事基础网络功能的开发与维护。

文章来自微信公众号：高效运维开发