# 1 导论

## 1.1 简介

性能优化一直都是大多数软件工程师头上的“紧箍咒”

本文思路

1 从liunx系统资源角度

CPU 性能、磁盘 I/O 性能、内存性能以及网络性能。

2 从篇章角度

基础篇:介绍 Linux 必备的基本原理以及对应的性能指标和性能工具。

案例篇:通过模拟案例，帮你分析高手在遇到资源瓶颈时，是如何观测、定位、分析并优化这些性能问题的。

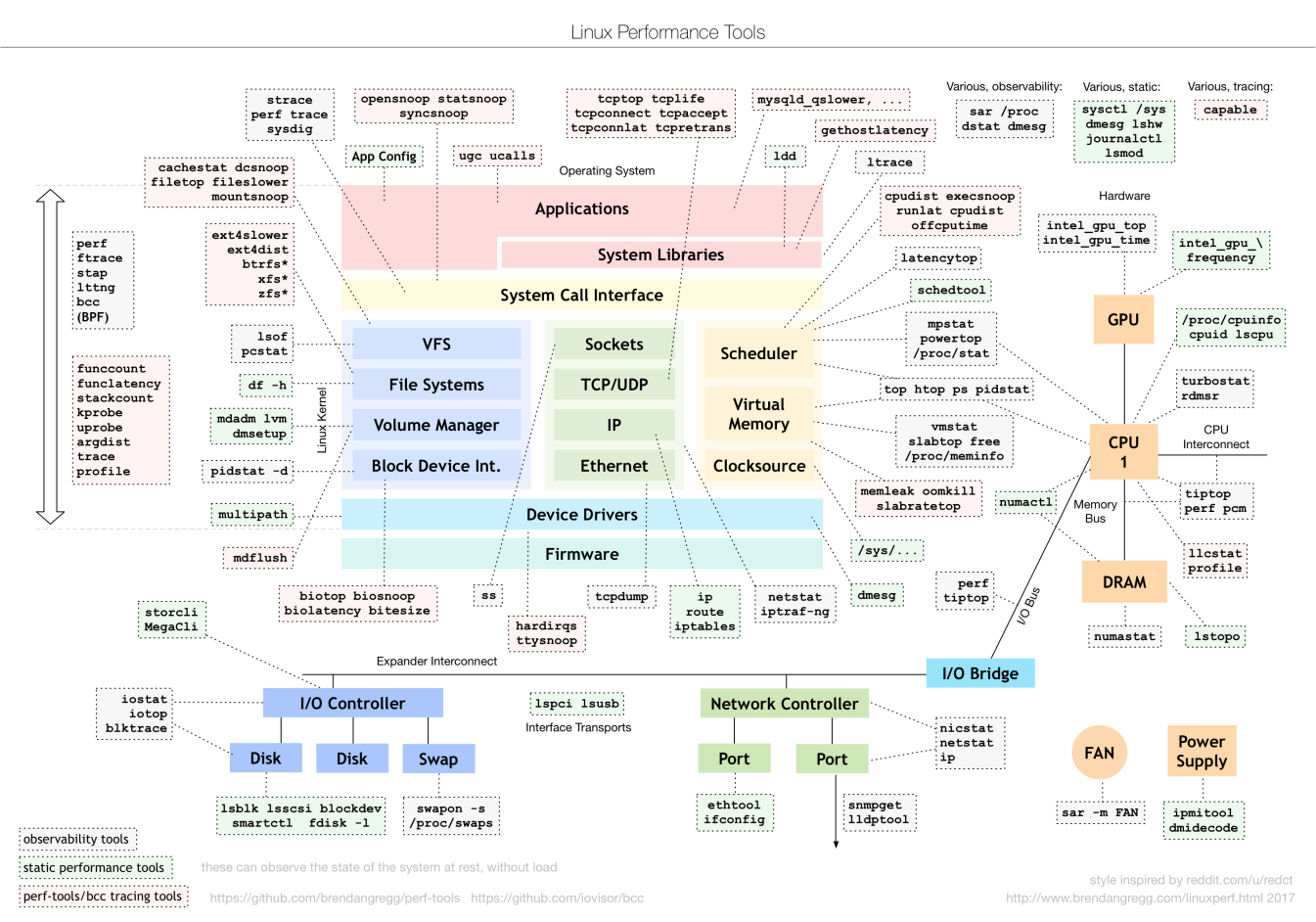
套路篇:梳理出排查问题的整体思路，也就是检查性能问题的一般步骤

答疑篇:常见问题解答

## 1.2 如何学习

|  |
| --- |
| 性能分析六个步骤 |
| 1 选择指标评估应用程序和系统的性能；  2 为应用程序和系统设置性能目标；  3 进行性能基准测试；  4 性能分析定位瓶颈；  5 优化系统和应用程序；  6 性能监控和告警。 |

|  |
| --- |
| 学习重点  1 理解最基础的系统知识原理  2 掌握必要的性能工具  3 实际场景演练，贯通不同组件  重点：建立整体系统性能的全局观 |
| 怎么学更高效  1 技巧1：抓重点(HOW)  虽然系统的原理很重要，但在刚开始一定不要试图抓住所有的实现细节。  2 技巧2：边学边实践  通过大量的案例演习掌握 Linux 性能的分析和优化。  3 技巧3：勤思考，多反思，善总结，多问为什么。(Why)  想真正学懂一门知识，最好的方法就是问问题。当你能提出好的问题时，就说明你已经深入了解了它。 |



**Liunx性能图谱**



# 2 CPU性能

## 2.1 概念篇

|  |
| --- |
| **命令 $ uptime**  负载指标 $ uptime  02:34:03 up 2 days, 20:14, 1 user, load average: 0.63, 0.83, 0.88  **--解析**  当前时间 04:03:58  系统已运行的时间 10 days, 13:19  当前在线用户 1 user  平均负载：0.54, 0.40, 0.20，最近1分钟、5分钟、15分钟系统的负载 |
| **命令 $cat /proc/loadavg**  0.10 0.06 0.01 1/72 29632  **--解析**  前3个数字表示平均进程数量外，  后面的1个分数，分母表示系统进程总数，分子表示正在运行的进程数；  最后一个数字表示最近运行的进程ID |

|  |
| --- |
| **平均负载：**  指单位时间内，系统处于可运行状态和不可中断状态的平均进程数，也就是平均活跃进程数，它和 CPU 使用率并没有直接关系。 |
| **可运行状态的进程:**  1 正在使用 CPU 或者正在等待 CPU 的进程  2 常用 ps 命令看到的，处于 R 状态（Running 或 Runnable）的进程。 |
| **不可中断状态的进程:**  1 处于内核态关键流程中的进程，并且这些流程是不可打断的  2 最常见的是等待硬件设备的 I/O 响应.  ps 命令中看到的 D 状态（Uninterruptible Sleep，也称为 Disk Sleep）的进程 |

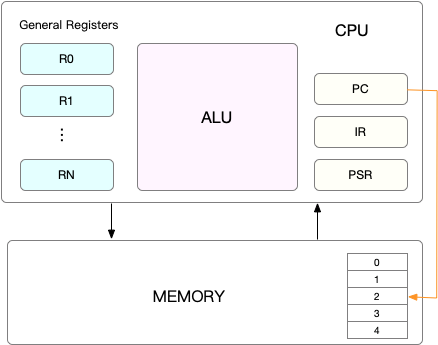
|  |
| --- |
| #查看CPU数  $ grep 'model name' /proc/cpuinfo |wc -l |
|  |

平均负载与 CPU 使用率

|  |
| --- |
| iostat、mpstat、pidstat |
| mpstat 是一个常用的多核 CPU 性能分析工具  用来实时查看每个 CPU 的性能指标，以及所有 CPU 的平均指标。  pidstat 是一个常用的进程性能分析工具  用来实时查看进程的 CPU、内存、I/O 以及上下文切换等性能指标。 |

### 2.1.1 CPU切换

**CPU 寄存器**，是 CPU 内置的容量小、但速度极快的内存。而**程序计数器**，则是用来存储 CPU 正在执行的指令位置、或者即将执行的下一条指令位置。它们都是 CPU 在运行任何任务前，必须的依赖环境，因此也被叫做 **CPU 上下文**。



**CPU上下文切换**，就是先把前一个任务的 CPU 上下文（也就是 CPU 寄存器和程序计数器）保存起来，然后加载新任务的上下文到这些寄存器和程序计数器，最后再跳转到程序计数器所指的新位置，运行新任务。

保存下来的上下文，会存储在系统内核中，并在任务重新调度执行时再次加载进来。这样就能保证任务原来的状态不受影响，让任务看起来还是连续运行。

**任务分类:**进程上下文切换、线程上下文切换以及中断上下文切换。

|  |
| --- |
| **进程上下文切换** |
| Linux 按照特权等级，把进程的运行空间分为内核空间和用户空间.CPU 特权等级的 Ring 0 和 Ring 3  1 **内核空间（Ring 0）**具有最高权限，可以直接访问所有资源；  2 **用户空间（Ring 3）**只能访问受限资源，不能直接访问内存等硬件设备，必须通过系统调用陷入到内核中，才能访问这些特权资源。  IMG_256 |
| 进程既可以在用户空间运行，又可以在内核空间中运行。  进程在用户空间运行时，被称为进程的用户态，而陷入内核空间的时候，被称为进程的内核态。 |
| **示例**：文件打开，系统调用的过程有没有发生 CPU 上下文的切换呢？答案自然是肯定的。  **答：**CPU 寄存器里原来用户态的指令位置，需要先保存起来。接着，为了执行内核态代码，CPU 寄存器需要更新为内核态指令的新位置。最后才是跳转到内核态运行内核任务。  而系统调用结束后，CPU 寄存器需要恢复原来保存的用户态，然后再切换到用户空间，继续运行进程。所以，一次系统调用的过程，其实是发生了两次 CPU 上下文切换。 |
| 进程上下文切换特性  1 进程是由内核来管理和调度的，进程的切换只能发生在内核态  进程的上下文不仅包括了虚拟内存、栈、全局变量等用户空间的资源，还包括了内核堆栈、寄存器等内核空间的状态。  2 保存上下文和恢复上下文的过程并不是“免费”的，需要内核在 CPU 上运行才能完成。  3 每次上下文切换都需要**几十纳秒到数微秒**的 CPU 时间。  这个时间还是相当可观的，特别是在进程上下文切换次数较多的情况下，很容易导致 CPU 将大量时间耗费在寄存器、内核栈以及虚拟内存等资源的保存和恢复上，进而大大缩短了真正运行进程的时间。  4 Linux 通过 TLB（Translation Lookaside Buffer）来管理虚拟内存到物理内存的映射关系。  当虚拟内存更新后，TLB 也需要刷新，内存的访问也会随之变慢。特别是在多处理器系统上，缓存是被多个处理器共享的，刷新缓存不仅会影响当前处理器的进程，还会影响共享缓存的其他处理器的进程。  5 进程切换时才需要切换上下文，换句话说，只有在进程调度的时候，才需要切换上下文。 |
| **进程在什么时候才会被调度到 CPU 上运行呢？**  1 为了保证所有进程可以得到公平调度，CPU 时间被划分为一段段的时间片，这些时间片再被轮流分配给各个进程。这样，当某个进程的时间片耗尽了，就会被系统挂起，切换到其它正在等待 CPU 的进程运行。  2 进程在系统资源不足（比如内存不足）时，要等到资源满足后才可以运行，这个时候进程也会被挂起，并由系统调度其他进程运行。  3 当进程通过睡眠函数 sleep 这样的方法将自己主动挂起时，自然也会重新调度。  4 当有优先级更高的进程运行时，为了保证高优先级进程的运行，当前进程会被挂起，由高优先级进程来运行。  5 发生硬件中断时，CPU 上的进程会被中断挂起，转而执行内核中的中断服务程序。 |

|  |
| --- |
| **线程上下文切换** |
| 1 线程是调度的基本单位，而进程则是资源拥有的基本单位  2 线程和进程理解  1) 当进程只有一个线程时，可以认为进程就等于线程。  2) 当进程拥有多个线程时，这些线程会共享相同的虚拟内存和全局变量等资源。这些资源在上下文切换时是不需要修改的。  3) 线程也有自己的私有数据，比如栈和寄存器等，这些在上下文切换时也是需要保存的。  3 线程的上下文切换情况  1) 前后两个线程属于不同进程。此时，因为资源不共享，所以切换过程就跟进程上下文切换是一样。  2) 前后两个线程属于同一个进程。此时，因为虚拟内存是共享的，所以在切换时，虚拟内存这些资源就保持不动，只需要切换线程的私有数据、寄存器等不共享的数据。 |

|  |
| --- |
| **中断上下文切换** |
| 1 中断:快速响应硬件的事件，中断处理会打断进程的正常调度和执行，转而调用中断处理程序，响应设备事件。而在打断其他进程时，就需要将进程当前的状态保存下来，这样在中断结束后，进程仍然可以从原来的状态恢复运行  2 中断上下文切换并不涉及到进程的用户态。中断上下文，其实只包括内核态中断服务程序执行所必需的状态，包括 CPU 寄存器、内核堆栈、硬件中断参数等。  3 对同一个 CPU 来说，中断处理比进程拥有更高的优先级，所以中断上下文切换并不会与进程上下文切换同时发生。  4 跟进程上下文切换一样，中断上下文切换也需要消耗 CPU，切换次数过多也会耗费大量的 CPU，甚至严重降低系统的整体性能。 |

### 2.1.2 如何分析CPU切换

|  |
| --- |
| vmstat |
| vmstat命令是最常见的Linux/Unix监控工具，可以展现给定时间间隔的服务器的状态值,包括服务器的CPU使用率，内存使用，虚拟内存交换情况,IO读写情况。  $ vmstat 2 1  #第一个参数是采样的时间间隔数，单位是秒，第二个参数是采样的次数 |
| ---procs---  r 表示运行队列(就是说多少个进程真的分配到CPU)  3 比较高 >5 高， >10 不正常了,危险  b 表示阻塞的进程  ---memory---  swpd 虚拟内存已使用的大小，  如果大于0，表示你的机器物理内存不足了。  free 空闲的物理内存的大小。  buff Linux/Unix系统是用来存储，目录里面有什么内容，权限等的缓存  cache cache直接用来记忆我们打开的文件,给文件做缓冲，当程序使用内存时，buffer/cached会很快地被使用。  ---swap---  si 每秒从磁盘读入虚拟内存的大小，如果这个值大于0，表示物理内存不够用或者内存泄露了，要查找耗内存进程解决掉。  so 每秒虚拟内存写入磁盘的大小，如果这个值大于0，同上。  ---io---  bi 块设备每秒接收的块数量，这里的块设备是指系统上所有的磁盘和其他块设备，默认块大小是1024byte，我本机上没什么IO操作，所以一直是0，但是我曾在处理拷贝大量数据(2-3T)的机器上看过可以达到140000/s，磁盘写入速度差不多140M每秒  bo 块设备每秒发送的块数量，例如我们读取文件，bo就要大于0。bi和bo一般都要接近0，不然就是IO过于频繁，需要调整。  ---system---  in 每秒CPU的中断次数，包括时间中断  cs 每秒上下文切换次数，例如我们调用系统函数，就要进行上下文切换，线程的切换，也要进程上下文切换。  ---cpu---  us 用户CPU时间，我曾经在一个做加密解密很频繁的服务器上，可以看到us接近100,r运行队列达到80。  sy 系统CPU时间，如果太高，表示系统调用时间长，例如是IO操作频繁。  id 空闲 CPU时间，一般来说，id + us + sy = 100,一般我认为id是空闲CPU使用率，us是用户CPU使用率，sy是系统CPU使用率。  wt 等待IO CPU时间。 |
| cs（context switch）是每秒上下文切换的次数。  in（interrupt）则是每秒中断的次数。  r（Running or Runnable）是就绪队列的长度，也就是正在运行和等待 CPU 的进程数。  b（Blocked）则是处于不可中断睡眠状态的进程数。 |

|  |
| --- |
| pidstat |
| vmstat 给出了系统总体的上下文切换情况,查看每个进程的详细情况，使用pidstat。  pidstat -w 选项，你就可以查看每个进程上下文切换的情况 |
| # 每隔 5 秒输出 1 组数据  $ pidstat -w 5  **cswch** 每秒自愿上下文切换（voluntary context switches）的次数  **nvcswch** 表示每秒非自愿上下文切换（non voluntary context switches）的次数  **自愿上下文切换**:指进程无法获取所需资源，导致的上下文切换。  **非自愿上下文切换**:指进程由于时间片已到等原因，被系统强制调度，发生的上下文切换。 |
| 1 自愿上下文切换高，可能发生了 I/O 等其他问题；  2 非自愿上下文切换高，说明进程都在被强制调度，都在争抢 CPU，CPU成了瓶颈；  3 中断高，CPU被中断处理程序占用，还需要通过查看/proc/interrupt文件来分析具体类型。 |

|  |
| --- |
| **典型情况** |
|  |

## 2.2 实战篇

|  |
| --- |
| **#stress安装#** |
| sudo yum install -y epel-release  sudo yum install -y stress |

### 2.2.1 实验1 CPU典型场景

|  |
| --- |
| 场景一：CPU 密集型进程 |
| 1 终端1 模拟一个 CPU 使用率 100% 的场景  $ stress --cpu 1 --timeout 600  2 终端2 查看平均负载的变化情况  # -d 参数表示高亮显示变化的区域  $ watch -d uptime    3 终端3 终端运行 mpstat 查看 CPU 使用率的变化情况  $ mpstat -P ALL 5  4 终端4 查看到底哪个进程占用  $ pidstat -u 5 1 |
| 分析  1 CPU 的使用率为 100%，但它的 iowait 只有 0。这说明，平均负载的升高正是由于 CPU 使用率为 100%  2 到底是哪个进程导致了 CPU 使用率为 100% 呢？你可以使用 pidstat 来查询 |

|  |
| --- |
| 场景二：I/O 密集型进程 |
| 1 终端1 模拟 I/O 压力，即不停地执行 sync  stress -i 1 --timeout 600  2 终端2 运行 uptime 查看平均负载的变化情况  mpstat -P ALL 5 1  3 终端3 运行 mpstat 查看 CPU 使用率的变化情况  pidstat -u 5 1 |

|  |
| --- |
| 场景三：大量进程的场景 |
| 1 终端1 模拟的是 16 个进程  stress -c 16 --timeout 600  2 终端2 查看过载情况  uptime  3 终端3 运行pidstate查看进程情况  pidstat -u 5 1 |

### 2.2.2 CPU切换分析

|  |
| --- |
| **CPU 切换分析** |
| 1 安装必备软件  预先安装 sysbench 和 sysstat 包  curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.rpm.sh | sudo bash  sudo yum -y install sysbench  yum install sysbench sysstat  2 操作步骤  2.1 终端1 模拟系统多线程调度  # 以 10 个线程运行 5 分钟的基准测试，模拟多线程切换的问题  $ sysbench --threads=10 --max-time=300 threads run  2.2 终端2 运行vmstat,观察上下文切换情况  # 每隔 1 秒输出 1 组数据（需要 Ctrl+C 才结束）  $ vmstat 1  r 列：就绪队列的长度已经到了 8，远远超过了系统 CPU 的个数 2，所以肯定会有大量的 CPU 竞争。  us（user）和 sy（system）列：这两列的 CPU 使用率加起来上升到了 100%，  系统 CPU 使用率sy 列高达 84%，说明 CPU 主要是被内核占用了。  in 列：中断次数也上升到了 1 万左右，说明中断处理也是个潜在的问题。    2.3 终端3 用 pidstat 来看一下,CPU 和进程上下文切换的情况  #每隔 1 秒输出 1 组数据（需要 Ctrl+C 才结束）  #-w 参数表示输出进程切换指标，而 -u 参数则表示输出 CPU 使用指标  $pidstat -w -u 1  终端3  # 每隔 1 秒输出一组数据（需要 Ctrl+C 才结束）  # -wt 参数表示输出线程的上下文切换指标  $ pidstat -wt 1    终端3 观察中断情况  # -d 参数表示高亮显示变化的区域  $ watch -d cat /proc/interrupts  重调度中断（RES）: 中断类型表示，唤醒空闲状态的 CPU 来调度新的任务运行。多处理器系统（SMP）中，调度器用来分散任务到不同 CPU 的机制，也称为处理器间中断（Inter-Processor Interrupts，IPI）。 |

# 3 内存性能

## 3.1 概念篇

## 3.2 实战篇

## 3.3 套路篇

# 4 网络性能

## 4.1 概念篇

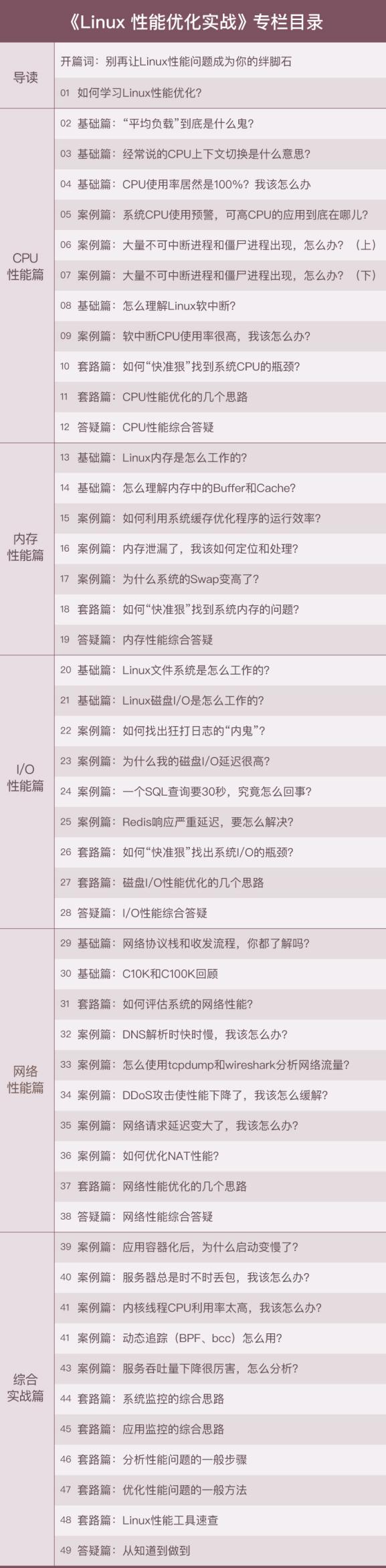
## 4.2 实战篇

## 4.3 套路篇

# 5 综合实战

## 5.1 实战篇

## 5.2 套路篇



# 来源

https://time.geekbang.org/column/intro/140