

## 16 | 基础篇：怎么理解内存中的Buffer和Cache？

2018-12-26 倪朋飞



讲述：冯永吉

时长 13:55 大小 12.75M



你好，我是倪朋飞。

上一节，我们梳理了 Linux 内存管理的基本原理，并学会了用 free 和 top 等工具，来查看系统和进程的内存使用情况。

内存和 CPU 的关系非常紧密，而内存管理本身也是很复杂的机制，所以感觉知识很硬核、很难啃，都是正常的。但还是那句话，初学时不用非得理解所有内容，继续往后学，多理解相关的概念并配合一定的实践之后，再回头复习往往会容易不少。当然，基本功不容放弃。

在今天的内容开始之前，我们先来回顾一下系统的内存使用情况，比如下面这个 free 输出界面：

```

1 # 注意不同版本的 free 输出可能会有所不同
2 $ free
3
4      total        used        free      shared  buff/cache   available
5 Mem:    8169348    263524    6875352         668     1030472     7611064
6 Swap:          0           0           0

```

显然，这个界面包含了物理内存 Mem 和交换分区 Swap 的具体使用情况，比如总内存、已用内存、缓存、可用内存等。其中缓存是 Buffer 和 Cache 两部分的总和。

这里的大部分指标都比较容易理解，但 Buffer 和 Cache 可能不太好区分。从字面上来说，Buffer 是缓冲区，而 Cache 是缓存，两者都是数据在内存中的临时存储。那么，你知道这两种“临时存储”有什么区别吗？


注：今天内容接下来的部分，Buffer 和 Cache 我会都用英文来表示，避免跟文中的“缓存”一词混淆。而文中的“缓存”，则通指内存中的临时存储。

## free 数据的来源

在我正式讲解两个概念前，你可以先想想，你有没有什么途径来进一步了解它们？除了中文翻译直接得到概念，别忘了，Buffer 和 Cache 还是我们用 free 获得的指标。

还记得我之前讲过的，碰到看不明白的指标时该怎么办吗？

估计你想起来了，不懂就去查手册。用 man 命令查询 free 的文档，就可以找到对应指标的详细说明。比如，我们执行 man free，就可以看到下面这个界面。

 复制代码

```

1 buffers
2      Memory used by kernel buffers (Buffers in /proc/meminfo)
3
4      cache  Memory used by the page cache and slabs (Cached and SReclaimable in /proc/
5
6      buff/cache
7      Sum of buffers and cache

```

从 free 的手册中，你可以看到 buffer 和 cache 的说明。

Buffers 是内核缓冲区用到的内存，对应的是 `/proc/meminfo` 中的 Buffers 值。

Cache 是内核页缓存和 Slab 用到的内存，对应的是 `/proc/meminfo` 中的 Cached 与 SReclaimable 之和。

这里的说明告诉我们，这些数值都来自 `/proc/meminfo`，但更具体的 Buffers、Cached 和 SReclaimable 的含义，还是没有说清楚。

要弄明白它们到底是什么，我估计你第一反应就是去百度或者 Google 一下。虽然大部分情况下，网络搜索能给出一个答案。但是，且不说筛选信息花费的时间精力，对你来说，这个答案的准确性也是很难保证的。

要注意，网上的结论可能是对的，但是很可能跟你的环境并不匹配。最简单来说，同一个指标的具体含义，就可能因为内核版本、性能工具版本的不同而有挺大差别。这也是为什么，我总在专栏中强调通用思路和方法，而不是让你死记结论。对于案例实践来说，机器环境就是我们的最大限制。

那么，有没有更简单、更准确的方法，来查询它们的含义呢？

## proc 文件系统

我在前面 CPU 性能模块就曾经提到过，`/proc` 是 Linux 内核提供的一种特殊文件系统，是用户跟内核交互的接口。比方说，用户可以从 `/proc` 中查询内核的运行状态和配置选项，查询进程的运行状态、统计数据等，当然，你也可以通过 `/proc` 来修改内核的配置。

`proc` 文件系统同时也是很多性能工具的最终数据来源。比如我们刚才看到的 `free`，就是通过读取 `/proc/meminfo`，得到内存的使用情况。

继续说回 `/proc/meminfo`，既然 Buffers、Cached、SReclaimable 这几个指标不容易理解，那我们还得继续查 `proc` 文件系统，获取它们的详细定义。

执行 `man proc`，你就可以得到 `proc` 文件系统的详细文档。

注意这个文档比较长，你最好搜索一下（比如搜索 `meminfo`），以便更快定位到内存部分。

```
1 Buffers %lu
2     Relatively temporary storage for raw disk blocks that shouldn't get tremendously lar
3
4 Cached %lu
5     In-memory cache for files read from the disk (the page cache). Doesn't include SwapC
6 ...
7 SReclaimable %lu (since Linux 2.6.19)
8     Part of Slab, that might be reclaimed, such as caches.
9
10 SUnreclaim %lu (since Linux 2.6.19)
11     Part of Slab, that cannot be reclaimed on memory pressure.
```

通过这个文档，我们可以看到：

Buffers 是对原始磁盘块的临时存储，也就是用来**缓存磁盘的数据**，通常不会特别大（20MB 左右）。这样，内核就可以把分散的写集中起来，统一优化磁盘的写入，比如可以把多次小的写合并成单次大的写等等。

Cached 是从磁盘读取文件的页缓存，也就是用来**缓存从文件读取的数据**。这样，下次访问这些文件数据时，就可以直接从内存中快速获取，而不需要再次访问缓慢的磁盘。

SReclaimable 是 Slab 的一部分。Slab 包括两部分，其中的可回收部分，用 SReclaimable 记录；而不可回收部分，用 SUnreclaim 记录。

好了，我们终于找到了这三个指标的详细定义。到这里，你是不是长舒一口气，满意地想着，总算弄明白 Buffer 和 Cache 了。不过，知道这个定义就真的理解了吗？这里我给你提了两个问题，你先想想能不能回答出来。

第一个问题，Buffer 的文档没有提到这是磁盘读数据还是写数据的缓存，而在很多网络搜索的结果中都会提到 Buffer 只是对**将要写入磁盘数据**的缓存。那反过来说，它会不会也缓存从磁盘中读取的数据呢？

第二个问题，文档中提到，Cache 是对从文件读取数据的缓存，那么它是不是也会缓存写文件的数据呢？

为了解答这两个问题，接下来，我将用几个案例来展示，Buffer 和 Cache 在不同场景下的使用情况。

# 案例

## 你的准备

跟前面实验一样，今天的案例也是基于 Ubuntu 18.04，当然，其他 Linux 系统也适用。我的案例环境是这样的。

机器配置：2 CPU，8GB 内存。


预先安装 sysstat 包，如 apt install sysstat。

之所以要安装 sysstat，是因为我们要用到 vmstat，来观察 Buffer 和 Cache 的变化情况。虽然从 /proc/meminfo 里也可以读到相同的结果，但毕竟还是 vmstat 的结果更加直观。

另外，这几个案例使用了 dd 来模拟磁盘和文件的 I/O，所以我們也需要观测 I/O 的变化情况。

上面的工具安装完成后，你可以打开两个终端，连接到 Ubuntu 机器上。

准备环节的最后一步，为了减少缓存的影响，记得在第一个终端中，运行下面的命令来清理系统缓存：

 复制代码

```
1 # 清理文件页、目录项、Inodes 等各种缓存
2 $ echo 3 > /proc/sys/vm/drop_caches
```

这里的 /proc/sys/vm/drop\_caches，就是通过 proc 文件系统修改内核行为的一个示例，写入 3 表示清理文件页、目录项、Inodes 等各种缓存。这几种缓存的区别你暂时不用管，后面我们都会讲到。

## 场景 1：磁盘和文件写案例

我们先来模拟第一个场景。首先，在第一个终端，运行下面这个 vmstat 命令：

 复制代码

```
1 # 每隔 1 秒输出 1 组数据
- - - - -
```



```

2 $ vmstat 1
3 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
4 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
5 0  0       0 7743608   1112  92168    0    0     0     0   52  152  0  1 100  0  0
6 0  0       0 7743608   1112  92168    0    0     0     0   36   92  0  0 100  0  0

```


输出界面里，内存部分的 buff 和 cache，以及 io 部分的 bi 和 bo 就是我们要关注的重点。

buff 和 cache 就是我们前面看到的 Buffers 和 Cache，单位是 KB。

bi 和 bo 则分别表示块设备读取和写入的大小，单位为块 / 秒。因为 Linux 中块的大小是 1KB，所以这个单位也就等价于 KB/s。


正常情况下，空闲系统中，你应该看到的是，这几个值在多次结果中一直保持不变。

接下来，到第二个终端执行 dd 命令，通过读取随机设备，生成一个 500MB 大小的文件：

 复制代码

```
1 $ dd if=/dev/urandom of=/tmp/file bs=1M count=500
```

然后再回到第一个终端，观察 Buffer 和 Cache 的变化情况：

 复制代码

```

1 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
2 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
3 0  0       0 7499460   1344 230484    0    0     0     0   29  145  0  0 100  0  0
4 1  0       0 7338088   1752 390512    0    0   488     0   39  558  0 47  53  0  0
5 1  0       0 7158872   1752 568800    0    0     0     4   30  376  1 50  49  0  0
6 1  0       0 6980308   1752 747860    0    0     0     0   24  360  0 50  50  0  0
7 0  0       0 6977448   1752 752072    0    0     0     0   29  138  0  0 100  0  0
8 0  0       0 6977440   1760 752080    0    0     0   152   42  212  0  1  99  1  0
9 ...
10 0  1       0 6977216   1768 752104    0    0     4 122880   33  234  0  1  51  49  0
11 0  1       0 6977440   1768 752108    0    0     0 10240   38  196  0  0  50  50  0

```

通过观察 `vmstat` 的输出，我们发现，在 `dd` 命令运行时，Cache 在不停地增长，而 Buffer 基本保持不变。

再进一步观察 I/O 的情况，你会看到，

在 Cache 刚开始增长时，块设备 I/O 很少，`bi` 只出现了一次 488 KB/s，`bo` 则只有一次 4KB。而过一段时间后，才会出现大量的块设备写，比如 `bo` 变成了 122880。

当 `dd` 命令结束后，Cache 不再增长，但块设备写还会持续一段时间，并且，多次 I/O 写的结果加起来，才是 `dd` 要写的 500M 的数据。


把这个结果，跟我们刚刚了解到的 Cache 的定义做个对比，你可能会有点晕乎。为什么前面文档上说 Cache 是文件读的页缓存，怎么现在写文件也有它的份？

这个疑问，我们暂且先记下来，接着再来看另一个磁盘写的案例。两个案例结束后，我们再统一进行分析。

不过，对于接下来的案例，我必须强调一点：


下面的命令对环境要求很高，需要你的系统配置多块磁盘，并且磁盘分区 `/dev/sdb1` 还要处于未使用状态。如果你只有一块磁盘，千万不要尝试，否则将会对你的磁盘分区造成损坏。

如果你的系统符合标准，就可以继续在第二个终端中，运行下面的命令。清理缓存后，向磁盘分区 `/dev/sdb1` 写入 2GB 的随机数据：

 复制代码

```
1 # 首先清理缓存
2 $ echo 3 > /proc/sys/vm/drop_caches
3 # 然后运行 dd 命令向磁盘分区 /dev/sdb1 写入 2G 数据
4 $ dd if=/dev/urandom of=/dev/sdb1 bs=1M count=2048
```

然后，再回到终端一，观察内存和 I/O 的变化情况：

 复制代码

```
1 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
2  r  b   swpd   free   buff  cache   si   so    bi    bo    in   cs us sv id wa st
```

|   |   |   |   |         |         |        |   |   |     |        |    |     |   |    |    |    |   |
|---|---|---|---|---------|---------|--------|---|---|-----|--------|----|-----|---|----|----|----|---|
| 3 | 1 | 0 | 0 | 7584780 | 153592  | 97436  | 0 | 0 | 684 | 0      | 31 | 423 | 1 | 48 | 50 | 2  | 0 |
| 4 | 1 | 0 | 0 | 7418580 | 315384  | 101668 | 0 | 0 | 0   | 0      | 32 | 144 | 0 | 50 | 50 | 0  | 0 |
| 5 | 1 | 0 | 0 | 7253664 | 475844  | 106208 | 0 | 0 | 0   | 0      | 20 | 137 | 0 | 50 | 50 | 0  | 0 |
| 6 | 1 | 0 | 0 | 7093352 | 631800  | 110520 | 0 | 0 | 0   | 0      | 23 | 223 | 0 | 50 | 50 | 0  | 0 |
| 7 | 1 | 1 | 0 | 6930056 | 790520  | 114980 | 0 | 0 | 0   | 12804  | 23 | 168 | 0 | 50 | 42 | 9  | 0 |
| 8 | 1 | 0 | 0 | 6757204 | 949240  | 119396 | 0 | 0 | 0   | 183804 | 24 | 191 | 0 | 53 | 26 | 21 | 0 |
| 9 | 1 | 1 | 0 | 6591516 | 1107960 | 123840 | 0 | 0 | 0   | 77316  | 22 | 232 | 0 | 52 | 16 | 33 | 0 |

从这里你会看到，虽然同是写数据，写磁盘跟写文件的现象还是不同的。写磁盘时（也就是 bo 大于 0 时），Buffer 和 Cache 都在增长，但显然 Buffer 的增长快得多。


这说明，写磁盘用到了大量的 Buffer，这跟我们在文档中查到的定义是一样的。

对比两个案例，我们发现，写文件时会用到 Cache 缓存数据，而写磁盘则会用到 Buffer 来缓存数据。所以，回到刚刚的问题，虽然文档上只提到，Cache 是文件读的缓存，但实际上，Cache 也会缓存写文件时的数据。

## 场景 2：磁盘和文件读案例


了解了磁盘和文件写的情况，我们再反过来想，磁盘和文件读的时候，又是怎样的呢？

我们回到第二个终端，运行下面的命令。清理缓存后，从文件 /tmp/file 中，读取数据写入空设备：

 复制代码

```
1 # 首先清理缓存
2 $ echo 3 > /proc/sys/vm/drop_caches
3 # 运行 dd 命令读取文件数据
4 $ dd if=/tmp/file of=/dev/null
```

然后，再回到终端一，观察内存和 I/O 的变化情况：

 复制代码


```
1 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
2 r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs us sy id wa st
3 0  1    0 7724164  2380 110844  0  0 16576  0  62 360 2  2 76 21  0
4 0  1    0 7691544  2380 143472  0  0 32640  0  46 439 1  3 50 46  0
5 0  1    0 7658736  2380 176204  0  0 32640  0  54 407 1  4 50 46  0
6 0  1    0 7626052  2380 208908  0  0 32640 40  44 422 2  2 50 46  0
```



观察 vmstat 的输出，你会发现读取文件时（也就是 bi 大于 0 时），Buffer 保持不变，而 Cache 则在不停增长。这跟我们查到的定义“Cache 是对文件读的页缓存”是一致的。


那么，磁盘读又是什么情况呢？我们再运行第二个案例来看看。

首先，回到第二个终端，运行下面的命令。清理缓存后，从磁盘分区 /dev/sda1 中读取数据，写入空设备：

 复制代码

```
1 # 首先清理缓存
2 $ echo 3 > /proc/sys/vm/drop_caches
3 # 运行 dd 命令读取文件
4 $ dd if=/dev/sda1 of=/dev/null bs=1M count=1024
```

然后，再回到终端一，观察内存和 I/O 的变化情况：

 复制代码

```
1 procs -----memory----- ---swap-- -----io----- -system-- -----cpu-----
2  r  b    swpd   free   buff  cache   si   so    bi    bo    in   cs  us  sy  id  wa  st
3  0  0        0 7225880   2716 608184    0    0     0     0   48  159  0  0 100  0  0
4  0  1        0 7199420   28644 608228    0    0 25928     0   60  252  0  1 65 35  0
5  0  1        0 7167092   60900 608312    0    0 32256     0   54  269  0  1 50 49  0
6  0  1        0 7134416   93572 608376    0    0 32672     0   53  253  0  0 51 49  0
7  0  1        0 7101484  126320 608480    0    0 32748     0   80  414  0  1 50 49  0
```

观察 vmstat 的输出，你会发现读磁盘时（也就是 bi 大于 0 时），Buffer 和 Cache 都在增长，但显然 Buffer 的增长快很多。这说明读磁盘时，数据缓存到了 Buffer 中。

当然，我想，经过上一个场景中两个案例的分析，你自己也可以对比得出这个结论：读文件时数据会缓存到 Cache 中，而读磁盘时数据会缓存到 Buffer 中。

到这里你应该发现了，虽然文档提供了对 Buffer 和 Cache 的说明，但是仍不能覆盖到所有的细节。比如说，今天我们了解到的这两点：

Buffer 既可以用作“将要写入磁盘数据的缓存”，也可以用作“从磁盘读取数据的缓存”。

Cache 既可以用作“从文件读取数据的页缓存”，也可以用作“写文件的页缓存”。

这样，我们就回答了案例开始前的两个问题。

简单来说，**Buffer 是对磁盘数据的缓存，而 Cache 是文件数据的缓存，它们既会用在读请求中，也会用在写请求中。**

## 小结

今天，我们一起探索了内存性能中 Buffer 和 Cache 的详细含义。Buffer 和 Cache 分别缓存磁盘和文件系统的读写数据。

从写的角度来说，不仅可以优化磁盘和文件的写入，对应用程序也有好处，应用程序可以在数据真正落盘前，就返回去做其他工作。

从读的角度来说，既可以加速读取那些需要频繁访问的数据，也降低了频繁 I/O 对磁盘的压力。

除了探索的内容本身，这个探索过程对你应该也有所启发。在排查性能问题时，由于各种资源的性能指标太多，我们不可能记住所有指标的详细含义。那么，准确高效的手段——查文档，就非常重要了。

你一定要养成查文档的习惯，并学会解读这些性能指标的详细含义。此外，proc 文件系统也是我们的好帮手。它为我们呈现了系统内部的运行状态，同时也是很多性能工具的数据来源，是辅助排查性能问题的好方法。

## 思考

最后，我想给你留一个思考题。

我们已经知道，可以使用 ps、top 或者 proc 文件系统，来获取进程的内存使用情况。那么，如何统计出所有进程的物理内存使用量呢？

提示：要避免重复计算多个进程同时占用的内存，像是页缓存、共享内存这类。如果你把 ps、top 得到的数据直接相加，就会出现重复计算的问题。

这里，我推荐从 /proc/< pid >/smaps 入手。前面内容里，我并没有直接讲过 /proc/< pid >/smaps 文件中各个指标含义，所以，需要你自己动手查 proc 文件系统的文档，解读并回答这个问题。

欢迎在留言区和我讨论，也欢迎你把这篇文章分享给你的同事、朋友。我们一起在实战中演练，在交流中进步。

 极客时间

# Linux 性能优化实战

## 10 分钟帮你找到系统瓶颈



**倪朋飞**  
微软资深工程师  
Kubernetes 项目维护者

新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得转载

上一篇 15 | 基础篇：Linux内存是怎么工作的？

下一篇 17 | 案例篇：如何利用系统缓存优化程序的运行效率？

## 精选留言 (62)

写留言



倪朋飞 置顶  
2018-12-26

74

关于磁盘和文件的区别，本来以为大家都懂了，所以没有细讲。磁盘是一个块设备，可以划分为不同的分区；在分区之上再创建文件系统，挂载到某个目录，之后才可以在这个目录中读写文件。

其实 Linux 中 “一切皆文件”，而文章中提到的“文件”是普通文件，磁盘是块设备文...  
展开 ∨



**Geek\_5258f...**

2018-12-26

👍 29

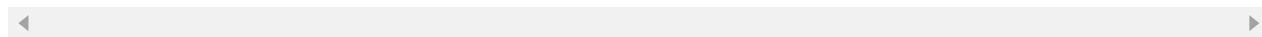
理论上，一个文件读首先到Block Buffer, 然后到Page Cache。有了文件系统才有了Page Cache.

在老的Linux上这两个Cache是分开的。那这样对于文件数据，会被Cache两次。这种方案虽然简单，

但低效。后期Linux把这两个Cache统一了。对于文件，Page Cache指向Block Buffer...

展开 ∨

作者回复: 🖱



**JJ**

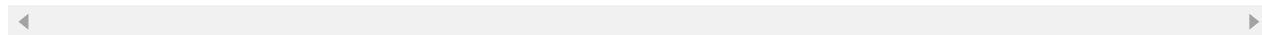
2018-12-26

👍 11

还是有点困惑，感觉读写磁盘上的数据不就是读写磁盘上的文件里的数据嘛，难道读磁盘上的数据可以不经过文件系统吗，可以直接读裸磁盘？有点没理解buffer是磁盘上的数据缓存，cache是文件数据缓存，求大神解答下。。

展开 ∨

作者回复: 请参考置顶回复



**Dr. ZZZ**

2019-01-01

👍 3

老师，关于buffer是对直接写磁盘的缓存，我想问下。现实中有哪些是直接写磁盘的场景。写读写文件不也最终是写到磁盘上吗？谢谢



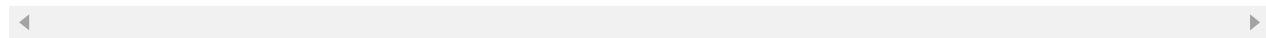
**David.cui**

👍 3

2018-12-26

数据库使用裸设备是明显的磁盘读写；如果数据库的数据文件在文件系统上就是文件读写。这样理解对么

作者回复: 对的



我来也

2018-12-26

👍 2

[D16打卡]

只有一块磁盘,就没轻易的试第二个案例.

-----  
以前应该只接触到了文件数据的缓存cache,没接触到磁盘数据的缓存buffer.

1.vim一个大文件,在第一次加载时较慢,之后再次打开时,会明显感觉到加载速度更快,应该...

展开 ∨

作者回复: 前面2是C库的缓存，跟系统的缓存没关系

ls的文档参考 info coreutils 'ls invocation'



虎虎♥

2018-12-26

👍 2

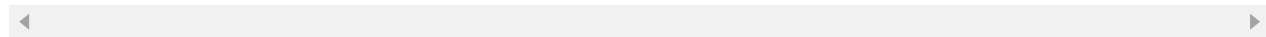
通过读csapp，又复习了下虚拟内存。其概念为“虚拟内存组织为一个由存放在磁盘上的N个连续的字节大小的单元组成的数组。”访问虚拟内存时，MMU通过访问页表，来索引到实际的存储地址。如果在物理内存中有缓存，直接从物理内存中读取数据。否则，从磁盘中读取，并选择牺牲一个物理页，并替换为新读取的页（当然，我觉得这种应该是...

展开 ∨

作者回复: 1. 物理内存的分布由系统管理，没有类似于虚拟内存这样的分布

2. 不是

3. LRU回收的是缓存，Swap换出的是不可回收的内存，比如进程的堆内存

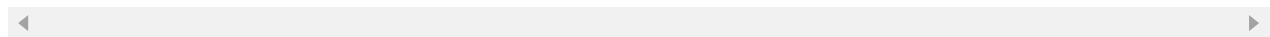


爱学习的小...

👍 1







往事随风, ...

2018-12-26

👍 1

# 首先清理缓存

```
echo 3 > /proc/sys/vm/drop_caches # 然后运行 dd 命令向磁盘分区 /dev/sdb1 写入 2G 数据 dd  
if=/dev/urandom of=/dev/sdb1 bs=1M count=2048
```

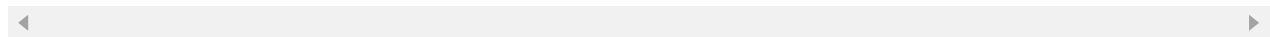
、

这个测试，在centos下是cache比buffer增长的快，和你说ub下正好相反，这是为什么？

procs -----memory----- ---swap-- -----io----- --system-- -----cpu-----...

展开 ▾

作者回复: 系统什么版本？是不是比较老？



科学Jia

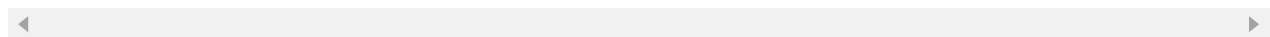
2018-12-26

👍 1

老师，女同学我今天上班时间终于追到这里了。写的真真清楚，想知道您花了多少时间学这些？

展开 ▾

作者回复: 也花了挺多时间，有些基础的原理在学校就学过了，也有很多是实践中学到的经验



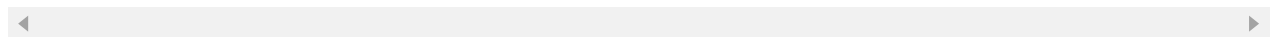
C家族的铁...

2018-12-26

👍 1

另外，Linux里的块设备，可以直接访问(比如数据库应用程序)，也可以存储文件系统然后被访问吧。

作者回复: 是的👉



金波

2018-12-26

👍 1

请问/tmp/file 是磁盘下的一个文件吗？没详细说明，可能是内存文件系统。磁盘下和

tmpfs的读对Cache是否一样？

作者回复: /tmp/file 是一个普通文件，确切的说是文件系统管理的文件，而没有磁盘下的哪个文件这一说。略过文件系统之后到了磁盘就都是Block了



豆腐居士

2019-02-26



top 查看16内存被吃光 但是没有看到占用内存比较高的进程是什么情况？free 查看是 buffers/cache占用内存比较多

作者回复: 这不是看到了吗，被缓存占了



AI杜嘉嘉

2019-02-21



想请教一下老师，怎么看待一个系统buffer和cache使用率过高的问题。是好是坏，如果这些缓存没及时回收可能会导致，程序异常

作者回复: 通常来说都是好事，不过也不是绝对的，还要看具体的场景的。比如，内存紧张的时候，回收缓存会对性能有一定影响；不合理的应用占用大量缓存，也可能会导致内存不足。



吴林辉

2019-02-18



“因为 Linux 中块的大小是 1KB，所以这个单位也就等价于 KB/s。”关于这一点，想请问下老师，linux block的大小不是4KB呢？

作者回复: 这句话来自vmstat的文档：<https://linux.die.net/man/8/vmstat>

通常说的的 Block Size 是磁盘分区的块大小，的确都是 4KB 了。



贼道



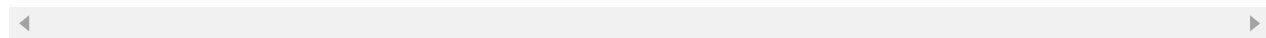
2019-02-15



老师，centos7怎么能查看系统里的cache和buffer被哪些进程占用？

展开 ∨

作者回复: 还没有这样的工具😏



lq

2019-02-02



线上出问题，往往没有太多时间查手册找含义，大神有什么好的快速的思路，

展开 ∨

作者回复: 这就需要多实践总结了，实践多了自然也就记住了。另外，碰到的问题和解决方法最好记录下来，方便后续参考

