top命令

top命令可以从宏观上观察操作系统的cpu，内存使用情况，以及每个进程的使用cpu情况，内存情况。

top命令的输出可以分为两个部分：前半部分是系统统计信息，后半部分是进程信息

```
top - 12:56:47 up  1:53,  6 users,  load average: 0.12, 0.14, 0.13
Tasks: 134 total,   1 running, 133 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.3 us,  4.0 sy,  0.0 ni, 93.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3863576 total,  2591188 free,   879044 used,   393344 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  2708080 avail Mem


   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
101041 root      20   0  161176   5948   4292 S   1.0  0.2   0:00.27 sshd
    11 root      rt   0       0      0      0 S   0.3  0.0   0:00.03 watchdog/0
   663 root      20   0  298928   6276   4928 S   0.3  0.2   0:05.88 vmtoolsd
 10301 root      20   0  161300   6004   4292 S   0.3  0.2   0:29.39 sshd
 31233 root      20   0  161176   5980   4292 S   0.3  0.2   0:24.77 sshd
 51720 root      20   0 3538028 739780  14372 S   0.3 19.1   0:17.11 java
 84916 root      20   0       0      0      0 S   0.3  0.0   0:00.45 kworker/0:2
100376 root      20   0  161960   2264   1592 S   0.3  0.1   0:00.05 top
     1 root      20   0  193456   6532   4076 S   0.0  0.2   0:02.44 systemd
```

统计信息

```
第1行是任务队列信息，它的结果等同于uptime命令。从左到右依次表示
系统当前时间、
系统运行时间、
当前登录用户数。
load average表示系统的平均负载，即任务队列的长度，这三个值分别表示1分钟、5分钟、15分钟到现在的平均值
```

```
第2行表示
总进程数,
正在运行的进程数,
休眠的进程数,
停止的进程数,
僵尸进程数。
```

```
第3行是cpu信息:
us 用户空间占用CPU百分比
sy 内核空间占用CPU百分比
ni 用户进程空间内改变过优先级的进程占用CPU百分比
id 空闲CPU百分比
wa 等待输入输出的CPU时间百分比
hi 硬件中断
si 软件中断
st: 实时
```

```
第4行是内存信息:依次显示
物理内存总量,
空闲物理内存,
已经使用的内存,
内核缓冲使用量,
```

```
第5行显示的交换空间信息:依次表示
交换空间总量
空闲空间总量
已经使用空间总量
缓冲区使用总量
```

进程信息

```
PI      : 进程ID
USE     : 进程所有者
PR      : 进程优先级
NI      : nice值.负值表示高优先级,正值表示低优先级
VIRT    : 进程使用的虚拟内存总量,单位KB.VIRT=SWAP+RES
RES     : 进程使用的,违背换出的物理内存大小,单位KB.RES=CODE+DATA
SHR     : 共享内存大小,单位KB
S       : 进程状态.D(不可中断的睡眠状态),R,S,T(跟踪/停止),Z
%CPU    : 上次更新到现在的CPU时间占用百分比
 %MEM   : 进程使用的物理内存百分比
 TIME+  : 进程使用的CPU时间总计,单位1/100秒
 COMMAND: 进程名称(命令行/命令名)
```

vmstat命令

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
Linux/Unix最喜爱的命令，一个是Linux/Unix都支持，二是相比top，我可以看到整个机器的CPU,内存,IO的使用情况，而不是单单看到各个
景不一样)。

一般vmstat工具的使用是通过两个数字参数来完成的，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，如:

```
# 2表示每个两秒采集一次服务器状态，1表示只采集一次。如果不输入第二个参数就表示一直采集
root@ubuntu:~# vmstat 2 1
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0      0 3498472 315836 3819540    0    0     0     1    2    0  0  0 100  0
```

参数说明

```
r 表示运行队列(就是说多少个进程真的分配到CPU)，我测试的服务器目前CPU比较空闲，没什么程序在跑，当这个值超过了CPU数目，

b 表示阻塞的进程,这个不多说，进程阻塞，大家懂的。

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，如果不是程序内存泄露的原因，那么你该升级内存了或者把

free 空闲的物理内存的大小，我的机器内存总共8G，剩余3415M。

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，我本机大概占用300多M

cache cache直接用来记忆我们打开的文件,给文件做缓冲，我本机大概占用300多M(这里是Linux/Unix的聪明之处，把空闲的物理

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。我的机器内

so 每秒虚拟内存写入磁盘的大小，如果这个值大于0，同上。

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，

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。

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，这个值要越小越好，太

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(机器在做压力测试，性能表现

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

```

iostat命令

iostat 命令可以提供详细的io信息

基本用法

```
# -d表示输出磁盘使用情况
# 1 表示每秒钟采样一次
# 2 表示采集两次
iostat -d 1 2
```

输出：

```
Linux 3.10.0-514.el7.x86_64 (localhost.localdomain)     2017年11月10日     _x86_64_    (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.05    0.00    0.11    0.02    0.00   99.82

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.48         3.49        14.65     295477    1240842
dm-0              0.54         3.12        14.61     264528    1237637
dm-1              0.01         0.02         0.01       1640       1084
```

说明：

```
tps：该设备每秒的传输次数
kB_read/s：每秒从设备（drive expressed）读取的数据量；
kB_wrtn/s：每秒向设备（drive expressed）写入的数据量；
kB_read：读取的总数据量；
kB_wrtn：写入的总数量数据量；这些单位都为Kilobytes。
```

如果要看更加详细的可以用-x

```
iostat -x 1 2
```

pidstat组件

```
pidstat是sysstat的一个组件，不仅仅可以检测进程使用cpu，io，内存情况，甚至可以深入到进程中的线程使用情况。
```

需要安装sysstat

```
yum -y install sysstat
```

检查cpu性能

它不仅仅可以监测进程的性能，还可以监测线程的性能。

运行一个Java程序，这里打成了jar包运行（jar名字1.jar），源码如下（T1的线程不休眠消耗cpu，T2会休眠不怎么消耗cpu）

```
public class HoldCpuMain {
    static class T1 implements Runnable{
        @Override
        public void run() {
            while(true){
                System.out.println("t1");
            }
        }
    }
     static class T2 implements Runnable{
         @Override
         public void run() {
             try {
                 while(true){
                     System.out.println("t2");
                     Thread.sleep(1000);
                 }
             } catch (InterruptedException e) {
                 e.printStackTrace();
             }
```

```
        }
    }
    public static void main(String[] args) {
        new Thread(new T1()).start();
        new Thread(new T2()).start();
        new Thread(new T2()).start();
    }
}
```

输入命令jps（jps是jdk自带的工具，配置好了环境变量即可使用）

```
[no1@localhost ~]$ jps
8908 1.jar
8924 Jps
```

可以看到有1.jar 在运行，进程ID是8908
输入pidstat命令查看进程信息， -p 进程ID -u表示对cpu的监控， 1 和 2 代表每秒钟采集一次，一共采集二次

```
[no1@localhost ~]$ pidstat -p 8908 -u 1 2
Linux 3.10.0-514.el7.x86_64 (localhost.localdomain)     2017年11月13日     _x86_64_    (2 CPU)

10时01分24秒   UID       PID    %usr %system  %guest    %CPU   CPU  Command
10时01分25秒  1000      8908   19.00   27.00    0.00   46.00     1  java
10时01分26秒  1000      8908   24.00   27.00    0.00   51.00     1  java
平均时间:  1000      8908   21.50   27.00    0.00   48.50     -  java
```

更进一步使用pidstat命令查看线程信息

```
[no1@localhost ~]$ pidstat -p 8908 -u 1 2 -t
Linux 3.10.0-514.el7.x86_64 (localhost.localdomain)     2017年11月13日     _x86_64_    (2 CPU)

10时02分04秒   UID      TGID       TID    %usr %system  %guest    %CPU   CPU  Command
10时02分05秒  1000      8908         -   21.00   27.00    0.00   48.00     1  java
10时02分05秒  1000         -      8908    0.00    0.00    0.00    0.00     1  |__java
10时02分05秒  1000         -      8909    0.00    0.00    0.00    0.00     1  |__java
10时02分05秒  1000         -      8910    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8911    0.00    0.00    0.00    0.00     1  |__java
10时02分05秒  1000         -      8912    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8913    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8914    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8915    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8916    0.00    0.00    0.00    0.00     1  |__java
10时02分05秒  1000         -      8917    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8918    0.00    0.00    0.00    0.00     0  |__java
10时02分05秒  1000         -      8919    1.00    0.00    0.00    1.00     1  |__java
10时02分05秒  1000         -      8920   21.00   26.00    0.00   47.00     1  |__java
10时02分05秒  1000         -      8921    0.00    0.00    0.00    0.00     1  |__java
10时02分05秒  1000         -      8922    0.00    0.00    0.00    0.00     1  |__java

10时02分05秒   UID      TGID       TID    %usr %system  %guest    %CPU   CPU  Command
10时02分06秒  1000      8908         -   26.00   24.00    0.00   50.00     1  java
10时02分06秒  1000         -      8908    0.00    0.00    0.00    0.00     1  |__java
10时02分06秒  1000         -      8909    0.00    0.00    0.00    0.00     1  |__java
10时02分06秒  1000         -      8910    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8911    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8912    0.00    0.00    0.00    0.00     0  |__java
```

```
10时02分06秒  1000         -      8913    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8914    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8915    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8916    0.00    0.00    0.00    0.00     1  |__java
10时02分06秒  1000         -      8917    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8918    0.00    0.00    0.00    0.00     0  |__java
10时02分06秒  1000         -      8919    0.00    0.00    0.00    0.00     1  |__java
10时02分06秒  1000         -      8920   26.00   24.00    0.00   50.00     1  |__java
10时02分06秒  1000         -      8921    0.00    0.00    0.00    0.00     1  |__java
10时02分06秒  1000         -      8922    0.00    0.00    0.00    0.00     1  |__java

平均时间:   UID      TGID       TID    %usr %system  %guest    %CPU   CPU  Command
平均时间:  1000      8908         -   23.50   25.50    0.00   49.00     -  java
平均时间:  1000         -      8908    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8909    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8910    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8911    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8912    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8913    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8914    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8915    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8916    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8917    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8918    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8919    0.50    0.00    0.00    0.50     -  |__java
平均时间:  1000         -      8920   23.50   25.00    0.00   48.50     -  |__java
平均时间:  1000         -      8921    0.00    0.00    0.00    0.00     -  |__java
平均时间:  1000         -      8922    0.00    0.00    0.00    0.00     -  |__java
```

可以看到线程id 8920 的线程占用的cpu很高（这里有很多线程，有些是jdk自带的线程）。
通过jstack工具查看（jstack是jdk自带的工具，配置jdk环境变量即可使用）

```
[no1@localhost ~]$ jstack -l 8908 > /home/no1/temp.txt
```

查看temp内容

```
2017-11-13 10:09:16
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.151-b12 mixed mode):

"Attach Listener" #12 daemon prio=9 os_prio=0 tid=0x00007f40a4001000 nid=0x2310 waiting on condition
   java.lang.Thread.State: RUNNABLE

   Locked ownable synchronizers:
    - None

"DestroyJavaVM" #11 prio=5 os_prio=0 tid=0x00007f40d4008800 nid=0x22cd waiting on condition [0x0000
   java.lang.Thread.State: RUNNABLE

   Locked ownable synchronizers:
    - None

"Thread-2" #10 prio=5 os_prio=0 tid=0x00007f40d418d000 nid=0x22da waiting on condition [0x00007f40
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldCpuMain$T2.run(HoldCpuMain.java:17)
```

```
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None

"Thread-1" #9 prio=5 os_prio=0 tid=0x00007f40d418b800 nid=0x22d9 waiting on condition [0x00007f40a9
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldCpuMain$T2.run(HoldCpuMain.java:17)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None

"Thread-0" #8 prio=5 os_prio=0 tid=0x00007f40d418a000 nid=0x22d8 runnable [0x00007f40a97d2000]
   java.lang.Thread.State: RUNNABLE
    at java.io.FileOutputStream.writeBytes(Native Method)
    at java.io.FileOutputStream.write(FileOutputStream.java:326)
    at java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:82)
    at java.io.BufferedOutputStream.flush(BufferedOutputStream.java:140)
    - locked <0x00000000e380dd60> (a java.io.BufferedOutputStream)
    at java.io.PrintStream.write(PrintStream.java:482)
    ...
    ...
    ...
```

可以看到Thread-0的 nid（Native id） 是 0x22d8， 0x 表示 16进制， 22d8十六进制转成十进制 正好是 8920也就是线程上面pidstat检测

检测io

编写下面大量使用IO的程序，打成可运行jar包（命名：2.jar） 进行执行

```
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class HoldIOMain {
    static class T1 implements Runnable{
        @Override
        public void run() {
            try {
                while(true){
                     FileOutputStream fos = new FileOutputStream("temp");
                     for(int i = 0 ; i < 100000 ;i ++){
                         fos.write(i);//大量的写
                     }
                     fos.flush();
                     fos.close();
                     FileInputStream fis = new FileInputStream("temp");
                     int b = 0;
                     while((b=fis.read()) != -1){//大量的读

                     }
                     fis.close();
```

```
                    b = 0;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    static class T2 implements Runnable{
        @Override
        public void run() {
            try {
                while(true){
                    Thread.sleep(1000);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) {
        new Thread(new T1()).start();
        new Thread(new T2()).start();
        new Thread(new T2()).start();
    }
}
```

同样jps查看 Java执行程序，发现9415进程ID 是执行的 2.jar

```
[no1@localhost ~]$ jps
9430 Jps
9415 2.jar
```

查看2.jar的进程io使用情况

```
[no1@localhost ~]$ pidstat -p 9415 -d 1 2
Linux 3.10.0-514.el7.x86_64 (localhost.localdomain)     2017年11月13日     _x86_64_    (2 CPU)

10时56分01秒   UID       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10时56分02秒  1000      9415      0.00    840.00      0.00  java
10时56分03秒  1000      9415      0.00    812.00     20.00  java
平均时间:  1000      9415      0.00    826.00     10.00  java
```

查看更加详细的 线程使用情况

```
[no1@localhost ~]$ pidstat -p 9415 -d 1 2 -t
Linux 3.10.0-514.el7.x86_64 (localhost.localdomain)     2017年11月13日     _x86_64_    (2 CPU)

10时56分40秒   UID      TGID       TID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10时56分41秒  1000      9415         -      0.00    896.00      0.00  java
10时56分41秒  1000         -      9415      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9416      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9417      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9418      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9419      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9420      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9421      0.00      0.00      0.00  |__java
```

```
10时56分41秒  1000         -      9422      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9423      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9424      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9425      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9426      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9427      0.00    892.00      0.00  |__java
10时56分41秒  1000         -      9428      0.00      0.00      0.00  |__java
10时56分41秒  1000         -      9429      0.00      0.00      0.00  |__java

10时56分41秒   UID      TGID       TID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
10时56分42秒  1000      9415         -      0.00    836.00      0.00  java
10时56分42秒  1000         -      9415      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9416      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9417      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9418      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9419      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9420      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9421      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9422      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9423      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9424      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9425      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9426      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9427      0.00    836.00      0.00  |__java
10时56分42秒  1000         -      9428      0.00      0.00      0.00  |__java
10时56分42秒  1000         -      9429      0.00      0.00      0.00  |__java

平均时间:   UID      TGID       TID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
平均时间:  1000      9415         -      0.00    866.00      0.00  java
平均时间:  1000         -      9415      0.00      0.00      0.00  |__java
平均时间:  1000         -      9416      0.00      0.00      0.00  |__java
平均时间:  1000         -      9417      0.00      0.00      0.00  |__java
平均时间:  1000         -      9418      0.00      0.00      0.00  |__java
平均时间:  1000         -      9419      0.00      0.00      0.00  |__java
平均时间:  1000         -      9420      0.00      0.00      0.00  |__java
平均时间:  1000         -      9421      0.00      0.00      0.00  |__java
平均时间:  1000         -      9422      0.00      0.00      0.00  |__java
平均时间:  1000         -      9423      0.00      0.00      0.00  |__java
平均时间:  1000         -      9424      0.00      0.00      0.00  |__java
平均时间:  1000         -      9425      0.00      0.00      0.00  |__java
平均时间:  1000         -      9426      0.00      0.00      0.00  |__java
平均时间:  1000         -      9427      0.00    864.00      0.00  |__java
平均时间:  1000         -      9428      0.00      0.00      0.00  |__java
平均时间:  1000         -      9429      0.00      0.00      0.00  |__java
```

9427 线程id 使用情况，同样可以通过jstack工具 验证，Thread-0 nid 0x24d3转换成十进制等于 9427

```
"Thread-2" #10 prio=5 os_prio=0 tid=0x00007f92e8175800 nid=0x24d5 waiting on condition [0x00007f92d87f6000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldIOMain$T2.run(HoldIOMain.java:34)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None

"Thread-1" #9 prio=5 os_prio=0 tid=0x00007f92e8173800 nid=0x24d4 waiting on condition [0x00007f92d88f7000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at HoldIOMain$T2.run(HoldIOMain.java:34)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None

"Thread-0" #8 prio=5 os_prio=0 tid=0x00007f92e8172000 nid=0x24d3 runnable [0x00007f92d89f8000]
   java.lang.Thread.State: RUNNABLE
    at java.io.FileInputStream.read0(Native Method)
    at java.io.FileInputStream.read(FileInputStream.java:207)
    at HoldIOMain$T1.run(HoldIOMain.java:18)
    at java.lang.Thread.run(Thread.java:748)

   Locked ownable synchronizers:
    - None
```

http://blog.csdn.net/jiuxiao199132

检查内存

检测内存 通过-r 命令即可。

mpstat 监控CPU使用率

mpstat是Multiprocessor Statistics的缩写，是实时监控工具，报告与cpu的一些统计信息这些信息都存在/proc/stat文件中，在多CPU系统里况的信息，而且能够有查看特定的cpu信息，mpstat最大的特点是:可以查看多核心的cpu中每个计算核心的统计数据；而且类似工具vmstat

绝大多数linux系统都需要安装 sysstat才能使用mpstat

```
yum install -y sysstat
```

查看多核cpu当前运行的状况，每两秒更新一次，一共更新5次

```
[root@VM_0_16_centos ~]# mpstat 2 5
Linux 3.10.0-693.el7.x86_64 (VM_0_16_centos)        08/06/2018      _x86_64_        (1 CPU)

03:56:42 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
03:56:44 PM  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
03:56:46 PM  all    0.50    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   99.50
03:56:48 PM  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
03:56:50 PM  all    0.50    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   98.99
03:56:52 PM  all    0.00    0.00    0.50    0.00    0.00    0.00    0.00    0.00    0.00   99.50
Average:     all    0.20    0.00    0.20    0.00    0.00    0.00    0.00    0.00    0.00   99.60
```

- %user 在internal时间段里，用户态的CPU时间(%)，不包含nice值为负进程 (usr/total)*100
- %nice 在internal时间段里，nice值为负进程的CPU时间(%) (nice/total)*100
- %sys 在internal时间段里，内核时间(%) (system/total)*100
- %iowait 在internal时间段里，硬盘IO等待时间(%) (iowait/total)*100
- %irq 在internal时间段里，硬中断时间(%) (irq/total)*100
- %soft 在internal时间段里，软中断时间(%) (softirq/total)*100
- %idle 在internal时间段里，CPU除去等待磁盘IO操作外的因为任何原因而空闲的时间闲置时间(%) (idle/total)*100