

目录

[简介](#)

[背景信息](#)

[案例分析：地址解析服务\(ARP\)中断](#)

[步骤 1：识别消耗CPU周期的进程](#)

[步骤 2：确定导致高CPU使用情况情况的CPU队列](#)

[步骤 3：转存数据包发送对CPU](#)

[步骤 4：请使用联邦机关跟踪](#)

[采样Cisco Catalyst 3850系列交换机的嵌入式活动管理器\(EEM\)脚本](#)

[相关信息](#)

简介

本文描述如何排除故障CPU使用情况注意事项，主要由于中断，在新的Cisco IOS XE平台。另外，本文介绍数是缺一不可的为了排除故障这样问题的新建的on命令此平台。

背景信息

知道是重要的Cisco IOS XE如何被构件。使用Cisco IOS XE，思科移动向Linux内核，并且所有子系统是被分解为的进程。是内部的Cisco IOS在的所有子系统-例如模块驱动程序，高可用性(HA)，等等-当前前运行作为在Linux操作系统(OS)内的软件进程。Cisco IOS运行作为在Linux OS (IOSd)内的一守护程序。Cisco IOS XE保留标准的Cisco IOS的不仅同一个外观，而且其操作、支持和管理。

这是一些有用的定义：

- **转发引擎驱动程序(联邦机关)**：这是Cisco Catalyst 3850系列交换机的重点并且对所有硬件编程/转发负责。
- **IOSd**：在Linux内核运行的这是Cisco IOS守护程序。它运行作为在内核内的软件进程。
- **数据包传输系统(PDS)**：这是体系结构和进程数据包如何到/从多种子系统传送。为例，它控制数据包如何从联邦机关传送到IOSd反之亦然。
- **把柄**：把柄可以设想作为指示器。它是方法发现关于在输出中使用方框生成的特定变量的详细信息。这类似于Local Target Logic (LTL)索引的概念在Cisco Catalyst 6500系列交换机的。

案例分析：地址解析服务(ARP)中断

排除故障和验证进程在此部分可以宽广地用于高CPU使用情况由于中断。

步骤 1：识别消耗CPU周期的进程

`show process cpu`命令自然显示CPU如何当前查找。注意Cisco Catalyst 3850系列交换机使用四个核心和您为所有四个核心看到列出的CPU使用情况：

```
3850-2#show processes cpu sort | exclude 0.0
Core 0: CPU utilization for five seconds: 53%; one minute: 39%; five minutes: 41%
Core 1: CPU utilization for five seconds: 43%; one minute: 57%; five minutes: 54%
Core 2: CPU utilization for five seconds: 95%; one minute: 60%; five minutes: 58%
Core 3: CPU utilization for five seconds: 32%; one minute: 31%; five minutes: 29%
PID    Runtime(ms) Invoked  uSecs  5Sec   1Min   5Min   TTY   Process
8525   472560     2345554  7525   31.37  30.84  30.83  0     iosd
5661   2157452     9234031  698    13.17  12.56  12.54  1088  fed
6206   19630      74895   262    1.83   0.43   0.10   0     eicored
6197   725760     11967089 60     1.41   1.38   1.47   0     pdsd
```

从输出，很清楚Cisco IOS守护程序与联邦机关一起消耗CPU的大部分，是此方框的重点。当CPU使用情况是高由于中断时，您看到IOSd和联邦机关使用CPU的大部分，并且这些子流程(或这些的一子集)使用CPU：

- 联邦机关Punject TX
- 联邦机关Punject RX
- 联邦机关Punject重新补充
- 联邦机关Punject TX完成

您能迅速移动到任何这些进程用`show process CPU`被选派的`<process>`命令。因为IOSd对CPU使用情况的多数负责，这是仔细观察到那。

```
3850-2#show processes cpu detailed process iosd sort | ex 0.0
Core 0: CPU utilization for five seconds: 36%; one minute: 39%; five minutes: 40%
Core 1: CPU utilization for five seconds: 73%; one minute: 52%; five minutes: 53%
Core 2: CPU utilization for five seconds: 22%; one minute: 56%; five minutes: 58%
Core 3: CPU utilization for five seconds: 46%; one minute: 40%; five minutes: 31%
PID    T C  TID  Runtime(ms) Invoked uSecs  5Sec   1Min   5Min   TTY   Process
                                (%)    (%)    (%)
8525   L           556160    2356540 7526   30.42  30.77  30.83  0     iosd
8525   L 1   8525  712558    284117  0     23.14  23.33  23.38  0     iosd
59     I           1115452  4168181 0     42.22  39.55  39.33  0     ARP Snoop
198    I           3442960  4168186 0     25.33  24.22  24.77  0     IP Host Track Proce
30     I           3802130  4168183 0     24.66  27.88  27.66  0     ARP Input
283    I           574800    3225649 0     4.33   4.00   4.11   0     DAI Packet Process
```

```
3850-2#show processes cpu detailed process fed sorted | ex 0.0
Core 0: CPU utilization for five seconds: 45%; one minute: 44%; five minutes: 44%
Core 1: CPU utilization for five seconds: 38%; one minute: 44%; five minutes: 45%
Core 2: CPU utilization for five seconds: 42%; one minute: 41%; five minutes: 40%
Core 3: CPU utilization for five seconds: 32%; one minute: 30%; five minutes: 31%
PID    T C  TID  Runtime(ms) Invoked uSecs  5Sec   1Min   5Min   TTY   Process
                                (%)    (%)    (%)
5638   L           612840    1143306 536   13.22  12.90  12.93  1088  fed
5638   L 3   8998  396500    602433  0     9.87   9.63   9.61   0     PunjectTx
5638   L 3   8997  159890    66051   0     2.70   2.70   2.74   0     PunjectRx
```

输出(输出的IOSd CPU)显示该ARP监听、IP主机跟踪流程和ARP输入高。当CPU中断归结于ARP数据包时，这编解码器。

步骤 2：

Cisco Catalyst 3850系列交换机有迎合不同种类的数据包的一定数量的队列(联邦机关维护32个RX

CPU队列，是队列去直接地CPU)。监控这些队列为了发现是重要的哪些数据包被踢对CPU，并且哪些由IOSd处理。这些队列是每个ASIC。

注意：有两个ASIC：0和1。Ports1通过24属于ASIC 0。

为了查看队列，请输入显示平台平底船统计信息波尔特asic <port-asic> cpuq <queue>方向<rx/>命令。

在显示平台平底船统计信息波尔特asic 0 cpuq -1方向rx命令，队列的-1个变元表全部。所以，此命令一览表波尔特ASIC的0所有接收队列。

现在，您必须识别以高速率排队推进很大数量的数据包。在本例中，队列的考试显示了此罪犯：

```
<snip>
RX (ASIC2CPU) Stats (asic 0 qn 16 lqn 16):
RXQ 16: CPU_Q_PROTO_SNOOPING
-----
Packets received from ASIC      : 79099152
Send to IOSd total attempts    : 79099152
Send to IOSd failed count      : 1240331
RX suspend count                : 1240331
RX unsuspend count              : 1240330
RX unsuspend send count        : 1240330
RX unsuspend send failed count : 0
RX dropped count                : 0
RX conversion failure dropped  : 0
RX pkt_hdr allocation failure  : 0
RX INTACK count                 : 0
RX packets dq'd after intack   : 0
Active RxQ event                : 9906280
RX spurious interrupt          : 0
<snip>
```

队列编号是16，并且队列名称是CPU_Q_PROTO_SNOOPING。

另一个方式发现罪犯队列将输入client命令显示平台的平底船。

```
3850-2#show platform punt client
tag      buffer      jumbo  fallback  packets  received  failures
         alloc    free   bytes    conv    buf
27       0/1024/2048  0/5    0/5       0        0         0        0        0
65536    0/1024/1600  0/0    0/512     0        0         0        0        0
65537    0/ 512/1600  0/0    0/512     1530    1530      244061   0        0
65538    0/  5/5      0/0    0/5       0        0         0        0        0
65539    0/2048/1600  0/16   0/512     0        0         0        0        0
65540    0/ 128/1600  0/8    0/0       0        0         0        0        0
65541    0/ 128/1600  0/16   0/32     0        0         0        0        0
65542    0/ 768/1600  0/4    0/0       0        0         0        0        0
65544    0/  96/1600  0/4    0/0       0        0         0        0        0
65545    0/  96/1600  0/8    0/32     0        0         0        0        0
65546    0/ 512/1600  0/32   0/512     0        0         0        0        0
65547    0/  96/1600  0/8    0/32     0        0         0        0        0
65548    0/ 512/1600  0/32   0/256     0        0         0        0        0
65551    0/ 512/1600  0/0    0/256     0        0         0        0        0
65556    0/  16/1600  0/4    0/0       0        0         0        0        0
65557    0/  16/1600  0/4    0/0       0        0         0        0        0
65558    0/  16/1600  0/4    0/0       0        0         0        0        0
65559    0/  16/1600  0/4    0/0       0        0         0        0        0
65560    0/  16/1600  0/4    0/0       0        0         0        0        0
s65561  421/ 512/1600  0/0    0/128  79565859 131644697 478984244 0 37467
```

```

65563 0/ 512/1600 0/16 0/256 0 0 0 0 0
65564 0/ 512/1600 0/16 0/256 0 0 0 0 0
65565 0/ 512/1600 0/16 0/256 0 0 0 0 0
65566 0/ 512/1600 0/16 0/256 0 0 0 0 0
65581 0/ 1/1 0/0 0/0 0 0 0 0 0
131071 0/ 96/1600 0/4 0/0 0 0 0 0 0

```

fallback pool: 98/1500/1600

jumbo pool: 0/128/9300

确定多数数据包分配了的标记。在本例中，它是**65561**。

然后，请输入此命令：

```

3850-2#show pds tag all | in Active|Tags|65561
Active Client Client
Tags Handle Name TDA SDA FDA TBufD TBytD
65561 7296672 Punt Rx Proto Snoop 79821397 79821397 0 79821397 494316524

```

此ouput显示队列是Rx原始监听。

s，在**65561**在client命令显示平台的平底船的输出中意味着前联邦机关把柄由流入数据包数量暂停并且淹没。如果**s**不消失，含义队列永久被滞留。

步骤 3：

在all命令显示pds的标记的结果，请注意把柄，**7296672**，在平底船Rx原始监听旁边报告。

请使用此把柄在显示pds客户端<handle>数据包为时接收器命令。注意您必须启用调试pds pktbuf为时，在您使用命令前。否则您遇到此错误：

```

3850-2#show pds client 7296672 packet last sink
% switch-2:pdsd:This command works in debug mode only. Enable debug using
"debug pds pktbuf-last" command

```

当调试启用，您看到此输出：

```

3850-2#show pds client 7296672 packet last sink
Dumping Packet(54528) # 0 of Length 60
-----
Meta-data
0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0010 00 00 16 1d 00 00 00 00 00 00 00 00 55 5a 57 f0 .....UZW.
0020 00 00 00 00 fd 01 10 df 00 5b 70 00 00 10 43 00 .....[p...C.
0030 00 10 43 00 00 41 fd 00 00 41 fd 00 00 00 00 00 ..C..A...A.....
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0050 00 00 00 3c 00 00 00 00 00 01 00 19 00 00 00 00 ...<.....
0060 01 01 b6 80 00 00 00 4f 00 00 00 00 00 00 00 00 .....O.....
0070 01 04 d8 80 00 00 00 33 00 00 00 00 00 00 00 00 .....3.....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 .....
Data
0000 ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01 .....
0010 08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a .....
0020 ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05 .....
0030 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 .....

```

此命令转存接收器接收的最后数据包，是在本例中的IOSd。这显示转存报头，并且可以解码与基于终端的Wireshark (TShark)。元数据是对于内部使用由系统，但是数据输出提供实际数据包信息。元数据，然而，保持非常有用。

从0070启动的公告线路。以后请使用如显示此处的前16个位：

```
3850-2#show platform port-asic ifm iif-id 0x0104d88000000033
```

```
Interface Table
Interface IIF-ID      : 0x0104d88000000033
Interface Name      : Gi2/0/20
Interface Block Pointer : 0x514d2f70
Interface State       : READY
Interface Stauts      : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt     : 6
Interface Epoch       : 0
Interface Type      : ETHER
    Port Type        : SWITCH PORT
Port Location         : LOCAL
    Slot              : 2
    Unit              : 20
Slot Unit             : 20
Acitve                : Y
SNMP IF Index         : 22
GPN                   : 84
EC Channel            : 0
EC Index              : 0
    ASIC              : 0
ASIC Port             : 14
Port LE Handle        : 0x514cd990
```

Non Zero Feature Ref Counts

```
FID : 48 (AL_FID_L2_PM), Ref Count : 1
FID : 77 (AL_FID_STATS), Ref Count : 1
FID : 51 (AL_FID_L2_MATM), Ref Count : 1
FID : 13 (AL_FID_SC), Ref Count : 1
FID : 26 (AL_FID_QOS), Ref Count : 1
```

Sub block information

```
FID : 48 (AL_FID_L2_PM), Private Data &colon; 0x54072618
FID : 26 (AL_FID_QOS), Private Data &colon; 0x514d31b8
```

罪犯接口识别此处。**Gi2/0/20**是有数据流生成器该泵ARP流量的地方。如果关闭此请向下，则将解决问题并且最小化CPU使用情况。

步骤 4：

与在最后一部分讨论的方法的唯一的缺点是只转存进入接收器的最后数据包，并且它也许不是罪犯。

一个更加好的方式排除故障此将使用呼叫**联邦机关跟踪**的功能。跟踪是数据包捕获方法(使用多种过滤器)由对CPU的联邦机关推送。然而联邦机关跟踪不是一样简单象在Cisco Catalyst 6500系列交换机的Netdr功能。此处进程分成步骤：

1. Enable (event)详细信息跟踪。默认情况下，事件追踪打开。您必须使详细信息跟踪为了获取实际数据包：

```
3850-2#set trace control fed-punject-detail enable
```

2. 优化捕获缓冲区。确定多么深深您的缓冲区是为详细信息跟踪和增加当必要时。

```
3850-2#show mgmt-infra trace settings fed-punject-detail
```

```
One shot Trace Settings:
```

```
Buffer Name: fed-punject-detail
```

```
Default Size: 32768
Current Size: 32768
Traces Dropped due to internal error: No
Total Entries Written: 0
One shot mode: No
One shot and full: No
Disabled: False
```

您能更改缓冲区大小用此命令：

```
3850-2#set trace control fed-punject-detail buffer-size <buffer size>
```

值可用对您是：

```
3850-2#set trace control fed-punject-detail buffer-size ?
<8192-67108864> The new desired buffer size, in bytes
default          Reset trace buffer size to default
```

3. 添加捕获过滤器。您当前需要捕获的添加多种过滤器。您能添加不同的过滤器和选择**匹配所有**或为您的捕获**匹配其中每一个**。

过滤器用此命令添加：

```
3850-2#set trace fed-punject-detail direction rx filter_add <filter>
```

这些选项是现在可以得到的：

```
3850-2#set trace fed-punject-detail direction rx filter_add ?
cpu-queue  rxq 0..31
field      field
offset     offset
```

现在您必须一起连接事。切记在此步骤2识别排除故障进程的罪犯队列？因为队列16是推送往CPU的很大数量的数据包队列，有意义跟踪此队列和发现什么数据包被踢对CPU由它。

您能选择跟踪所有队列用此命令：

```
3850-2#set trace fed-punject-detail direction rx filter_add cpu-queue <start queue>
<end queue>
```

这是此示例的命令：

```
3850-2#set trace fed-punject-detail direction rx filter_add cpu-queue 16 16
```

您必须选择**匹配全部**或**一匹配其中任一**您的过滤器的然后启用trace：

```
3850-2#set trace fed-punject-detail direction rx match_all
3850-2#set trace fed-punject-detail direction rx filter_enable
```

4. 显示已过滤数据包。您能显示用显示mgmt在下trace消息提供punject详细信息命令捕获的数据包。

```
3850-2#show mgmt-infra trace messages fed-punject-detail
```

[11/25/13 07:05:53.814 UTC 2eb0c9 5661]
00 00 00 00 00 00 4e 00 40 07 00 02 08 00 00 51 3b
00 00 00 00 00 01 00 00 03 00 00 00 00 00 00 01
00 00 00 00 20 00 00 0e 00 00 00 00 00 01 00 74
00 00 00 04 00 54 41 02 00 00 00 00 00 00 00 00

[11/25/13 07:05:53.814 UTC 2eb0ca 5661]
ff ff ff ff ff ff aa bb cc dd 00 00 08 06 00 01
08 00 06 04 00 01 aa bb cc dd 00 00 c0 a8 01 0a
ff ff ff ff ff ff c0 a8 01 14 00 01 02 03 04 05
06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 f6 b9 10 32

[11/25/13 07:05:53.814 UTC 2eb0cb 5661] Frame descriptors:

[11/25/13 07:05:53.814 UTC 2eb0cc 5661]

=====

fdFormat=0x4 systemTtl=0xe
loadBalHash1=0x8 loadBalHash2=0x8
spanSessionMap=0x0 forwardingMode=0x0
destModIndex=0x0 skipIdIndex=0x4
srcGpn=0x54 qosLabel=0x41
srcCos=0x0 ingressTranslatedVlan=0x3
bpdu=0x0 spanHistory=0x0
sgt=0x0 fpeFirstHeaderType=0x0
srcVlan=0x1 rcpServiceId=0x2
wccpSkip=0x0 srcPortLeIndex=0xe
cryptoProtocol=0x0 debugTagId=0x0
vrfId=0x0 saIndex=0x0
pendingAfdLabel=0x0 destClient=0x1
appId=0x0 finalStationIndex=0x74
decryptSuccess=0x0 encryptSuccess=0x0
rcpMiscResults=0x0 stackedFdPresent=0x0
spanDirection=0x0 egressRedirect=0x0
redirectIndex=0x0 exceptionLabel=0x0
destGpn=0x0 inlineFd=0x0
suppressRefPtrUpdate=0x0 suppressRewriteSideEffects=0x0
cmi2=0x0 currentRi=0x1
currentDi=0x513b dropIpUnreachable=0x0
srcZoneId=0x0 srcAsicId=0x0
originalDi=0x0 originalRi=0x0
srcL3IfIndex=0x2 dstL3IfIndex=0x0
dstVlan=0x0 frameLength=0x40
fdCrc=0x7 tunnelSpokeId=0x0

=====

[11/25/13 07:05:53.814 UTC 2eb0cd 5661]
[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: **Q: 16, Tag: 65561**
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: **Physical IIF-id 0x104d88000000033**
[11/25/13 07:05:53.814 UTC 2eb0d0 5661] PUNT PATH (fed_punject_get_src_l3if_index:
434):RX: L3 IIF-id 0x101b6800000004f
[11/25/13 07:05:53.814 UTC 2eb0d1 5661] PUNT PATH (fed_punject_fd_2_pds_md:478):
RX: l2_logical_if = 0x0
[11/25/13 07:05:53.814 UTC 2eb0d2 5661] PUNT PATH (fed_punject_get_source_cos:638):
RX: Source Cos 0
[11/25/13 07:05:53.814 UTC 2eb0d3 5661] PUNT PATH (fed_punject_get_vrf_id:653):
RX: VRF-id 0
[11/25/13 07:05:53.814 UTC 2eb0d4 5661] PUNT PATH (fed_punject_get_src_zoneid:667):
RX: Zone-id 0
[11/25/13 07:05:53.814 UTC 2eb0d5 5661] PUNT PATH (fed_punject_fd_2_pds_md:518):
RX: get_src_zoneid failed
[11/25/13 07:05:53.814 UTC 2eb0d6 5661] PUNT PATH (fed_punject_get_acl_log_direction:
695): RX: : Invalid CMI2
[11/25/13 07:05:53.814 UTC 2eb0d7 5661] PUNT PATH (fed_punject_fd_2_pds_md:541):RX:
get_acl_log_direction failed

```

[11/25/13 07:05:53.814 UTC 2eb0d8 5661] PUNT PATH (fed_punject_get_acl_full_direction:
724):RX: DI 0x513b ACL Full Direction 1
[11/25/13 07:05:53.814 UTC 2eb0d9 5661] PUNT PATH (fed_punject_get_source_sgt:446):
RX: Source SGT 0
[11/25/13 07:05:53.814 UTC 2eb0da 5661] PUNT PATH (fed_punject_get_first_header_type:680):
RX: FirstHeaderType 0
[11/25/13 07:05:53.814 UTC 2eb0db 5661] PUNT PATH (fed_punject_rx_process_packet:916):
RX: fed_punject_pds_send packet 0x1f00 to IOSd with tag 65561
[11/25/13 07:05:53.814 UTC 2eb0dc 5661] PUNT PATH (fed_punject_rx_process_packet:744):
RX: **** RX packet 0x2360 on qn 16, len 128 ****
[11/25/13 07:05:53.814 UTC 2eb0dd 5661]
buf_no 0 buf_len 128

<snip>

```

此输出提供大量信息，并且应该典型地是发现数据包来自的地方，并且什么的足够在他们包含。

报头转储的第一部分再是系统使用的元数据。第二部分是实际数据包。

```

ff ff ff ff ff ff - destination MAC address
aa bb cc dd 00 00 - source MAC address

```

您能选择跟踪此源MAC地址为了发现罪犯端口(一旦识别这是从队列16被踢数据包的多数;此输出只显示数据包的一个实例，并且其他输出/数据包截去)。

然而，有一个更加好的方式。注意在报头信息以后是存在的日志：

```

[11/25/13 07:05:53.814 UTC 2eb0ce 5661] PUNT PATH (fed_punject_rx_process_packet:
830):RX: Q: 16, Tag: 65561
[11/25/13 07:05:53.814 UTC 2eb0cf 5661] PUNT PATH (fed_punject_get_physical_iif:
579):RX: Physical IIF-id 0x104d88000000033

```

第一本日志清楚地告诉您从哪些队列和标记此数据包来。如果不知道队列更早的，这是识别的简单的方法排队它是。

第二本日志是更加有用的，因为提供物理接口ID出厂(IIF)-源接口的ID。十六进制值是能使用为了转存关于该端口的信息的把柄：

```

3850-2#show platform port-asic ifm iif-id 0x0104d88000000033
Interface Table
Interface IIF-ID          : 0x0104d88000000033
Interface Name         : Gi2/0/20
Interface Block Pointer  : 0x514d2f70
Interface State          : READY
Interface Stauts         : IFM-ADD-RCVD, FFM-ADD-RCVD
Interface Ref-Cnt       : 6
Interface Epoch         : 0
Interface Type        : ETHER
  Port Type           : SWITCH PORT
  Port Location         : LOCAL
  Slot                : 2
  Unit                : 20
  Slot Unit            : 20
  Acitve               : Y
  SNMP IF Index       : 22
  GPN                  : 84

```



```

EC Channel          : 0
EC Index           : 0
  ASIC              : 0
ASIC Port          : 14
Port LE Handle     : 0x514cd990
Non Zero Feature Ref Counts
FID : 48(AL_FID_L2_PM), Ref Count : 1
FID : 77(AL_FID_STATS), Ref Count : 1
FID : 51(AL_FID_L2_MATM), Ref Count : 1
FID : 13(AL_FID_SC), Ref Count : 1
FID : 26(AL_FID_QOS), Ref Count : 1
Sub block information
FID : 48(AL_FID_L2_PM), Private Data &colon; 0x54072618
FID : 26(AL_FID_QOS), Private Data &colon; 0x514d31b8

```

您再次识别source接口和罪犯。

跟踪是关键为了排除故障高CPU使用情况问题和提供大量信息为了成功地解决这样情况的一个强大的工具。

示例Cisco Catalyst 3850系列交换机的被嵌入的活动管理器(EEM)脚本

请使用此命令为了触发将生成的日志在一特定阈值：

```

process cpu threshold type total rising <CPU %> interval <interval in seconds>
switch <switch number>

```

用命令生成的日志如下所示：

```

*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]: Threshold: :
50, Total CPU Utilization(total/Intr) :50/0, Top 3 processes(Pid/Util) : 8622/25,
5753/12, 9663/0

```

生成的日志提供此信息：

- 在触发时的总CPU利用率。这由托塔尔CPU Utilization (总计/Intr)识别：50/0在本例中。
- 顶部进程-这些在格式列出PID/CPU%。因此在本例中，这些是：

```

*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]: Threshold: :
50, Total CPU Utilization(total/Intr) :50/0, Top 3 processes(Pid/Util) : 8622/25,
5753/12, 9663/0

```

EEM脚本显示此处：

```

*Jan 13 00:03:00.271: %CPUMEM-5-RISING_THRESHOLD: 1 CPUMEMd[6300]: Threshold: :
50, Total CPU Utilization(total/Intr) :50/0, Top 3 processes(Pid/Util) : 8622/25,
5753/12, 9663/0

```

注意：threshold命令进程的cpu在3.2.X系列当前不运转。记住的另一个点是此命令查看在四个核心中的平均值CPU utilization并且生成日志，当此平均值到达在命令定义的百分比时。

相关信息

- [什么是Cisco IOS XE ?](#)
- [思科Catalyst 3850交换机-数据表或宣传单页和文件](#)
- [技术支持和文档 - Cisco Systems](#)