

对EEM脚本进行故障排除和测试

目录

[简介](#)

[先决条件](#)

[要求](#)

[使用的组件](#)

[背景信息](#)

[使用Show命令进行EEM验证](#)

[确认计时器处于活动状态](#)

[确认触发事件正在触发](#)

[查看事件历史记录](#)

[使用手动触发器进行EEM验证](#)

[操作注意事项](#)

[问题：CLI命令无法执行](#)

[问题：EEM操作花费的时间超过最大运行时](#)

[问题：EEM触发过于频繁](#)

[相关信息](#)

简介

本文档介绍嵌入式事件管理器(EEM)脚本验证，并介绍常见操作注意事项和故障场景。

先决条件

要求

本文档假设读者已经熟悉Cisco IOS/IOS XE嵌入式事件管理器(EEM)功能。如果您不熟悉此功能，请阅读 [EEM功能概述](#) 首先。

Catalyst 9K系列交换机上的EEM需要Network Essentials许可证级别的DNA插件。Network Advantage完全支持EEM。

使用的组件

本文档中的信息涉及在Catalyst系列交换机上实施的EEM版本4.0。

本文档中的信息都是基于特定实验室环境中的设备编写的。本文档中使用的所有设备最初均采用原始（默认）配置。如果您的网络处于活动状态，请确保您了解所有命令的潜在影响。

背景信息

EEM在有效部署时是一个有用的功能，但重要的是要确保EEM能够准确完成作者的目标。审查不力的脚本可能会导致生产中出现灾难性问题。脚本最多只能以不必要的方式执行。本文档提供有关如何使用CLI show命令测试和验证EEM的有用信息，还介绍了一些常见的故障场景和用于识别和纠正问题的调试。

使用Show命令进行EEM验证

确认计时器处于活动状态

部署由计时器触发的EEM脚本时，如果脚本未按预期触发，请确认计时器处于活动状态并倒计时。

考虑分别名为test和test3的以下EEM脚本：

```
<#root>

event manager

  applet test

    authorization bypass
    event timer watchdog time 60
    action 0010 syslog msg "Test script running"

  event manager

  applet test3

    authorization bypass
    event timer watchdog name test3 time 300
    action 0010 syslog msg "test3 script running"
```

- 第一个脚本（测试）使用60秒（未命名）监视器计时器来触发脚本。
- 第二个脚本(test3)使用名为test3的300秒监视程序计时器来触发脚本。

可以使用命令show event manager statistics server查看已配置的计时器以及这些计时器的当前值。

示例

```
<#root>

Switch#

show event manager statistics server
```

EEM Queue Information

Client	Triggered Events	Dropped Events	Queue Size	Queue Max	Average Run Time
Call Home	5	0	0	64	0.021
EEM Applets	181	0	0	64	0.003
EEM IOS .sh Scripts	0	0	0	128	0.000
EEM Tcl Scripts	0	0	0	64	0.000

```
iosp_global_eem_proc    30      0      0      16      0.004
onep event service init 0      0      0      128     0.000
```

```
EEM Policy Counters
Name Value
```

```
-----
EEM Policy Timers
```

```
Name                Type
Time Remaining <-- EEM Countdown timer
```

```
-----
_EEMinternalname0
```

```
      watchdog                53.328
```

```
<--- Unnamed timers receive an internal name - this timer is for the 'test' policy
```

```
_EEMinternalname1      watchdog                37.120
```

```
test3
```

```
      watchdog                183.232
```

```
<--- Named timers use their configured name - this is the named timer configured for policy 'test3'
```

确认触发事件正在触发

如本文档的“确认计时器处于活动状态”部分所述，每次触发EEM小程序时，IOS XE都会在show event manager statistics server的输出中增加EEM Applets客户端行的“触发事件”列。要验证EEM脚本是否按预期运行，请多次执行触发事件并检查show event manager statistics server的输出以确认此值增量。如果没有，则说明脚本未触发。

当该命令按顺序多次运行时，计时器值将倒计时。当计时器达到零且脚本运行时，EEM小应用的触发事件计数也会增加。

```
<#root>
```

```
Switch#
```

```
show event manager statistics server
```

```
EEM Queue Information
```

```
Triggered
```

```
  Dropped Queue Queue Average
Client
```

```
Events
```

```

      Events  Size  Max  Run Time
-----
Call Home                5      0      0      64      0.021
EEM Applets              183
      0      0      64      0.003

```

<--- "Triggered Events" column is incremented by 2 due to 2 timers firing

```

EEM IOS .sh Scripts      0      0      0      128      0.000
EEM Tcl Scripts          0      0      0      64      0.000
iosp_global_eem_proc    30      0      0      16      0.004
onep event service init 0      0      0      128      0.000

```

EEM Policy Counters
Name Value

```


EEM Policy Timers
Name                               Type
Time Remaining

```

```

-----
_EEMinternalname0
      watchdog          56.215
_EEMinternalname1      watchdog          100.006
test3
      watchdog          126.117

```

 注意：如果未发生这种情况，请调查您的脚本以验证已配置的计时器。

查看事件历史记录

对于不是由计时器触发的脚本，show event manager history events命令在确认小程序按预期触发时非常有用。

考虑以下EEM脚本：

```

<#root>
event manager
  applet test_manual
    authorization bypass
  event none                                <-- manual trigger type for testing

action 0010

```

```
syslog msg "I am a manually triggered script!" <-- message that is printed when script runs
```

此脚本在执行CLI事件管理器运行test_manual并打印系统日志消息时运行。除了系统日志中的输出外，可以通过查看show event manager history events的输出来验证此脚本的执行情况，如下所示：

```
<#root>
```

```
Switch#
```

```
show event manager history events
```

```
No. Job Id Proc Status Time of Event
```

```
Event Type
```

```
Name  
1 5 Actv success Fri Nov 6 15:45:07 2020
```

```
timer countdown
```

```
callback: Call Home process <-- timer bases event that fired
```

```
2 18 Actv success Mon Nov 9 14:12:33 2020 oir callback: Call Home process  
3 19 Actv success Mon Nov 9 14:12:40 2020 oir callback: Call Home process  
4 20 Actv success Fri Nov13 14:35:49 2020
```

```
none
```

```
applet: test_manual <-- manually triggered event
```

使用手动触发器进行EEM验证

在某些情况下，需要手动触发EEM脚本，以测试执行流程或执行一次性操作。这可以通过具有事件无触发的EEM脚本实现，如以下输出所示：

```
<#root>
```

```
event manager
```

```
applet test_manual
```

```
authorization bypass
```

```
event none
```

```
action 0010 syslog msg "I am a manually triggered script!"
```

使用命令event manager run test_manual从enable提示符手动触发脚本：

```
<#root>
```

```
Switch#
```

```
event manager run test_manual <-- Manually runs the script
```

```
Switch#
```

```
show log <-- Check for the log from action 10.
```

```
*Oct 26 21:24:40.762:
```

```
%HA_EM-6-LOG: test_manual: I am a manually triggered script! <-- %HA_EM logs are from EEM events. The s
```

操作注意事项

确保EEM脚本在生产中使用之前经过验证。通常，脚本无法按预期工作有几种主要方式，下面将讨论其中三种方式。

本节介绍如何使用EEM脚本检查以下3个常见问题：

1. CLI命令失败：该命令无法解析，因此无法执行。
2. 脚本运行时间过长：EEM脚本的默认运行时间限制为20秒。如果超过此时间，脚本将在所有命令运行之前停止。
3. 脚本运行过于频繁：有时，脚本使用的触发事件可能过于频繁，导致脚本快速触发。最好控制脚本的触发频率和触发速率。

问题：CLI命令无法执行

此示例脚本包含多个问题。它是一个简单的小程序，可将多个show命令的输出附加到本地闪存介质中的文本文件：

```
<#root>
```

```
event manager
```

```
applet Data_Collection
```

```
auth bypass
```

```
event timer
```

```
watchdog time 60
```

```
action 1.0 cli command "enable"
```

```
action 1.1 cli command "show clock | append flash:DataCollection.txt"
```

```
action 1.2 cli command "show interfaces brief | append flash:DataCollection.txt"
```

```
action 1.3 cli command "show ip route | append flash:DataCollection.txt"
```

```
action 1.4 cli command "show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt"
```

```
action 1.5 cli command "show platform hardware fed switch active qos stats internal cpu policer | appen
```

```
action 2.0 syslog msg "Data Capture Complete"
```

小程序已成功运行，但未生成预期结果：

```
<#root>
```

```
Switch#
```

```
show logging | in Capture
```

```
<-- Our script-generated syslog contains the string "Capture".
```

```
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete
```

```
<-- Action 2.0 successfully ran.
```

```
Switch#
```

```
dir flash: | in .txt
```

```
<-- We only expected one .txt file, however two appear in flash:
```

```
32792 -rw- 36 Mar 11 2021 20:40:01 +00:00 DataCollection.txt
```

```
32798 -rw- 807 Mar 11 2021 20:40:01 +00:00 Datacollection.txt
```

```
Switch#
```

```
more flash:DataCollection.txt
```

```
<-- the output of our expected .txt file is empty except for the output of "show clock
```

```
"
```

```
*20:40:01.343 UTC Thu Mar 11 2021
```

使用debug embedded event manager action cli协助验证applet。

```
<#root>
```

```
Switch#
```

```
debug embedded event manager action cli
```

```
*Mar 11 20:40:01.175: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_open called.
```

```
<-- The applet is called.
```

```
*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch>
```

```
*Mar 11 20:40:01.275: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch>enable
```

```
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
```

```
*Mar 11 20:40:01.285: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show clock | appen
```

```

*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.396: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#show interfaces br
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

show interfaces breif

| append flash:DataCollection.txt

<-- Here is our first problem. "brief" is misspelled, so the command does not run.

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
^

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

% Invalid input detected at '^' marker. <-- CLI parser failure

*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.507: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#

show ip route | append flash:Datacollection.txt <-- This created the second .txt file. The file name is

*Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#

*Mar 11 20:40:01.618: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#

show processes cpu sorted | exclude 0.0 | append flash:DataCollection.txt

<-- This problem is less intuitive.

*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : CPU utilization for five
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : PID Runtime(ms) Invoked

the "exclude" argument reads everything beyond the pipe as the value that is to be excluded

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 117 57246 448028 127 0.0

A problem like this will likely not be evident in debugging

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 2 4488 16816 266 0.07% 0

This underscores the importance of pre-production testing to ensure the script performs as expected

.
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 173 829 44093 18 0.07% 0
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 205 22271 1313739 16 0.0
*Mar 11 20:40:01.729: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 467 238 2238 106 0.07% 0

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 81 12793 151345 84 0.07%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 232 22894 2621198 8 0.07%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 7 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 6 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 8 17 2804 6 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 9 33511 11402 2939 0.00%

```



```

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 12 0 2 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 10 106 1402 75 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 14 439 42047 10 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 11 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 16 0 1 0 0.00% 0.00% 0.00%

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 5 0 1 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 18 0 3 0 0.00% 0.00% 0.00%
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : 20+ lines read from cli

*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : IN : Switch#

show platform hardware fed switch active qos stats internal cpu policer

| append flash:DataCollection.txt
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : show platform hardware f
<-- Here, the syntax of the command was not properly parsed out before implementation. We are missing an

*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

^ <-- missing word queue

*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :

% Invalid input detected at '^' marker. <-- CLI parser failure

*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT :
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : Switch#
*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection: Data Capture Complete

<-- The syslog from Action 2.0 writes.

*Mar 11 20:40:01.941: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : CTL : cli_close called.

<-- The applet closes out as expected after executing all configured actions.

```

结论：正确检查所有EEM操作，并使用调试来防止错误配置和排版错误。

问题：EEM操作花费的时间超过最大运行时

在此方案中，使用简单的EEM收集以120秒间隔捕获的控制平面数据包。它将新的捕获数据附加到本地存储介质中的输出文件中。

```
<#root>
```

```
event manager
```

```
applet Capture
```

```
event timer
```

```
watchdog time 120 <-- 120 second countdown timer
```

```

action 1.0 cli command "enable"
action 1.1 cli command "no monitor capture CPUCapture"
action 2.0 cli command "monitor capture CPUCapture control-plane in match any buffer circular"
action 2.1 cli command "monitor capture CPUCapture start"
action 3.0 wait 45
action 4.0 cli command "monitor capture CPUCapture stop"
action 4.1 cli command "show clock | append flash:CPUCapture.txt"
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"

action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"

```

您可以轻松确定EEM未按预期完成。从操作5.0检查系统日志的本地日志。此系统日志会在小程序的每个成功迭代上打印。该日志未打印在缓冲区中，并且文件CPUCapture.txt未写入闪存：

```
<#root>
```

```
Switch#
```

```
show logging | include "CPUCapture Complete"
```

```
Switch#
```

```
dir flash: | include CPUCapture.txt
```

启用调试以进行调查。最常用的调试是debug event manager action cli。此实用程序按顺序打印操作对话框。

调试输出：调试输出显示成功调用的小程序。初始操作运行没有问题，但捕获无法结束。

```
<#root>
```

```
Switch#
```

```
debug event manager action cli
```

```
*Jan 28 22:55:54.742: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_open called.
```

```
<-- This is the initial message seen when the applet is called.
```

```
*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch>
```

```
The applet name can be seen within the line.
```

```
*Jan 28 22:55:54.843: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch>enable
```

```
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
```

```
*Jan 28 22:55:54.854: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#no monitor capture CPU
```

```
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Capture does not exist
```

```
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT :
```

```
*Jan 28 22:55:54.964: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
```

```
*Jan 28 22:55:54.965: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCap
Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:55:55.075: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : IN : CoreSwitch#monitor capture CPUCap
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : Started capture point : CPUCaptu

<-- The applet successfully creates and starts the capture.
```

```
*Jan 28 22:55:55.185: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : OUT : CoreSwitch#
*Jan 28 22:56:15.187: %HA_EM-6-LOG: Capture : DEBUG(cli_lib) : : CTL : cli_close called.
```

```
<-- After 20 seconds, cli_close is called and the applet begins to exit.
```

```
*Jan 28 22:56:15.187: fh_server: fh_io_ipc_msg: received msg FH_MSG_CALLBACK_DONE from client 27 pcli
*Jan 28 22:56:15.187: fh_io_ipc_msg: EEM callback policy Capture has ended with abnormal exit status of
```

```
FF
```

```
*Jan 28 22:56:15.187:
```

```
EEM policy Capture has exceeded it's elapsed time limit of 20.0 seconds <-- We are informed that the pol
```

```
*Jan 28 22:56:15.187: fh_io_ipc_msg: received FH_MSG_API_CLOSE client=27
*Jan 28 22:56:15.187: tty is now going through its death sequence
```

```
*Note "
```

```
debug event manager all
```

```
" is used to enable all debugs related to event manager.
```

解决方案：默认情况下，EEM策略运行不超过20秒。如果EEM中的操作运行时间超过20秒，则EEM无法完成。确保EEM的运行时间足以使您的Applet操作运行。配置maxrun以指定更合适的最大运行时值。

示例

```
<#root>
```

```
event manager
```

```
applet Capture
```

```
event timer watchdog time 120
```

```
maxrun 60
```

```
<-- Maxrun 60 specifies the capture will run for a maximum of 60 seconds.
```

```
action 1.0 cli command "enable"
```

```
action 1.1 cli command "no monitor capture CPUcapture"
```

```
action 2.0 cli command "monitor capture CPUcapture control-plane in match any buffer circular"
```

```
action 2.1 cli command "monitor capture CPUcapture start"
```

```
action 3.0 wait 45
```

```
<-- The altered maxrun allows the capture to run for the necessary time.
```

```
action 4.0 cli command "monitor capture CPUCapture stop"  
action 4.1 cli command "show clock | append flash:CPUCapture.txt"  
action 4.2 cli command "show mon cap CPUCapture buff dump | append flash:CPUCapture.txt"  
action 5.0 syslog msg "CPUCapture Complete - Next capture in 2 minutes"
```

问题：EEM触发过于频繁

有时，给定触发器的多个实例会在短时间内发生。这可能导致小程序的过度迭代，并在最坏的情况下产生严重后果。

此applet在特定系统日志模式上触发，然后收集show命令输出并将此输出附加到文件。具体而言，当已识别接口的线路协议丢弃时，小程序将触发：

```
<#root>
```

```
event manager
```

```
applet MonitorLinkFlap
```

```
event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"  
action 1.0 cli command "enable"  
action 1.1 cli command "show ip route | append flash:MonitorLinkFlap.txt "  
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"  
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"  
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Moni"  
action 5.0 syslog msg "Link has flapped - Data gathered"
```

每次观察系统日志时，小程序都会触发。接口抖动之类的事件会在短时间内迅速发生。

```
<#root>
```

```
Switch#
```

```
sh log | in Data gathered
```

```
*Jan 29 04:19:06.678: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

```
<-- The applet generates this syslog each time it fires.
```

```
*Jan 29 04:19:27.367: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:19:36.779: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:19:57.472: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:20:06.570: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:20:27.671: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:20:36.774: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered  
*Jan 29 04:20:57.264: %HA_EM-6-LOG: MonitorLinkFlap: Link has flapped - Data gathered
```

Applet在几分钟时间内运行了多次，导致不需要的输出文件包含无关数据。文件还继续增大，并继续填充本地介质。此简单示例如果重复运行，EEM不会造成太大的操作威胁，但此场景可能导致使用更复杂脚本的崩溃。

在这种情况下，限制小应用的触发频率会很有用。

解决方案：应用速率限制来控制小应用的运行速度。raterlimit关键字附加在触发器语句中，并与以秒为单位的值相关联。

示例

```
<#root>
```

```
event manager
```

```
applet MonitorLinkFlap
```

```
event syslog pattern "Interface GigabitEthernet1/0/23, changed state to down"
```

```
raterlimit 60
```

```
<-- Raterlimit
```

```
specifies a minimum amount of time that must pass before the applet will again trigger.
```

```
action 1.0 cli command "enable"
```

```
action 1.1 cli command "show clock | append flash:MonitorLinkFlap.txt "
```

```
action 2.0 cli command "show interface gig1/0/23 | append flash:MonitorLinkFlap.txt"
```

```
action 3.0 cli command "show process cpu sorted | append flash:MonitorLinkFlap.txt"
```

```
action 4.0 cli command "show platform hardware fed active fwd-asic drops exceptions | append flash:Moni"
```

```
action 5.0 syslog msg "Link has flapped - Data gathered"
```

相关信息

[思科IOS嵌入式事件管理器4.0](#)

[EEM的最佳实践和有用的脚本](#)

关于此翻译

思科采用人工翻译与机器翻译相结合的方式将此文档翻译成不同语言，希望全球的用户都能通过各自的语言得到支持性的内容。

请注意：即使是最好的机器翻译，其准确度也不及专业翻译人员的水平。

Cisco Systems, Inc. 对于翻译的准确性不承担任何责任，并建议您总是参考英文原始文档（已提供链接）。