對EEM指令碼進行故障排除和測試

目錄

簡介

本文檔介紹嵌入式事件管理器(EEM)指令碼驗證，並介紹常見操作注意事項和故障場景。

必要條件

需求

本檔案假設讀取器已熟悉Cisco IOS/IOS XE內嵌式事件管理員(EEM)功能。如果您還不熟悉此功能，請閱讀 EEM功能概述 首先。

Catalyst 9K系列交換機上的EEM需要Network Essentials許可證級別的DNA外掛。Network Advantage完全支援EEM。

採用元件

本檔案中的資訊與在Catalyst系列交換器上實作的EEM版本4.0相關。

本文中的資訊是根據特定實驗室環境內的裝置所建立。文中使用到的所有裝置皆從已清除（預設）的組態來啟動。如果您的網路運作中，請確保您瞭解任何指令可能造成的影響。

背景資訊

EEM在有效部署時是一個有用的功能，但確保EEM能夠準確完成作者的目標非常重要。未經仔細稽核的指令碼可能會導致生產中出現災難性的問題。指令碼充其量只能以不必要的方式執行。本文檔提供了有關如何使用CLI show命令測試和驗證EEM的有用資訊，並且還介紹了一些常見故障場景和用於識別和糾正問題的調試。

使用Show命令進行EEM驗證

確認計時器處於活動狀態

部署由計時器觸發的EEM指令碼時，如果指令碼未按預期觸發，請確認計時器處於活動狀態並倒計時。

請考慮分別名為test和test3的這些EEM指令碼：

<#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"
```

- 第一個指令碼（測試）使用60秒（未命名）的監視器計時器觸發指令碼。
- 第二個指令碼(test3)使用名為test3的300秒監視程式計時器來觸發指令碼。

使用show event manager statistics server命令可以檢視已配置的計時器以及這些計時器的當前值。

範例

<#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           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 timed configured for policy 'test3'
```

確認觸發事件正在觸發

如本文檔的「確認計時器處於活動狀態」部分所述，每次觸發EEM小程式時，IOS XE都會在show event manager statistics server的輸出中增加EEM Applets客戶端行的「觸發的事件」列。要驗證EEM指令碼是否按預期運行，請多次執行觸發事件並檢查show event manager statistics server的輸出以確認此值增量。如果未觸發，則說明指令碼未觸發。

當命令按順序多次運行時，計時器值將倒計時。當計時器達到零且指令碼運行時，EEM小程式觸發事件計數也會增加。

```
<#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：如果未發生這種情況，請調查您的指令碼以驗證已配置的計時器。

檢視事件歷史記錄

對於不是由計時器觸發的指令碼，show event manager history events命令對於確認按預期觸發小程式非常有用。

請考慮以下EEM指令碼：

```
<#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
```

此指令碼在執行CLI事件管理器運行test_manual並列印系統日誌消息時運行。除了系統日誌中的輸出外，可以通過檢視show event manager history events的輸出來驗證此指令碼的執行，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
```

使用手動觸發器的EEM驗證

在一些情況下，需要手動觸發EEM指令碼，以測試執行流程或執行一次性操作。可以使用帶有事件無觸發的EEM指令碼來完成此操作，如以下輸出所示：

<#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從enable提示符手動觸發指令碼：

```
<#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
```

操作注意事項

確保EEM指令碼在生產中使用之前經過驗證。通常，指令碼無法按預期工作有幾種主要方式，下面將討論其中三種方式。

本節介紹如何使用EEM指令碼檢查以下3個常見問題：

1. CLI命令失敗：該命令無法分析，因此無法執行。
2. 指令碼運行時間過長：EEM指令碼的預設運行時間限製為20秒。如果超出此時間，指令碼將在所有命令運行之前停止。
3. 指令碼運行過於頻繁：有時，指令碼使用的觸發事件可能會發生過於頻繁，從而導致指令碼快速觸發。 最好控制指令碼的觸發頻率和觸發速率。

問題：CLI命令無法執行

此示例指令碼包含幾個問題。它是一個簡單的小程式，可將多個show命令的輸出附加到本地快閃記憶體介質中的文本檔案：

```
<#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 breif | 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"
```

小程式已成功運行，但未生成預期結果：

<#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協助驗證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% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 10 106 1402 75 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 14 439 42047 10 0.00% 0.(
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 11 0 1 0 0.00% 0.00% 0.0
*Mar 11 20:40:01.730: %HA_EM-6-LOG: Data_Collection : DEBUG(cli_lib) : : OUT : 16 0 1 0 0.00% 0.00% 0.0

*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% 0.0
*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 ar

*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.
```

結論：正確稽核所有EEM操作，並使用調試來防止配置錯誤和列印錯誤。

問題：EEM操作花費的時間比最大運行時長

在此案例中，使用簡單的EEM以120秒的間隔收集控制平面資料包捕獲。 它將新的捕獲資料附加到本地儲存介質中的輸出檔案中。

<#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"
```

您可以輕鬆確定EEM沒有按預期完成。從操作5.0檢查系統日誌的本地日誌。 此系統日誌將列印在每次成功的小程式迭代上。該日誌未列印在緩衝區中，並且檔案CPUCapture.txt未寫入快閃記憶體：

```
<#root>
Switch#
show logging | include "CPUCapture Complete"

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

啟用調試以調查。最常用的調試是debug event manager action cli。此實用程式按順序列印操作對話方塊。

調試輸出：調試輸出顯示已成功呼叫小程式。初始操作運行沒有問題，但捕獲無法結束。

```
<#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 pclien
*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.

解決方案：預設情況下，EEM策略運行不超過20秒。如果EEM中的操作運行時間超過20秒，則EEM無法完成。確保EEM的運行時間足以使您的Applet操作運行。配置maxrun以指定更合適的最大運行時值。

範例

<#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"
```

問題：EEM觸發過於頻繁

有時，給定觸發器的多個例項會在短時間內發生。這可能導致小程式的過度迭代，並在最壞的情況下產生嚴重的後果。

此applet觸發特定系統日誌模式，然後收集show命令輸出並將此輸出附加到檔案。具體來說，當識別介面的線路協定丟棄時，小程式將觸發：

<#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"
```

每次觀察系統日誌時，都會觸發小程式。諸如介面翻動之類的事件可以在短時間內迅速發生。

<#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在幾分鐘時間內運行了多次，導致不需要的輸出檔案包含無關資料。檔案還繼續增大，並繼續填充本地介質。此簡單示例EEM如果重複運行，不會造成太大的操作威脅，但這種情況可能導致使用更複雜指令碼的崩潰。

在這種情況下，最好限制小程式的觸發頻率。

解決方案：應用速率限制來控制小程式的運行速度。raterlimit關鍵字附加在觸發器語句中，並與以秒為單位的值相關聯。

範例

```
<#root>

event manager

applet MonitorLinkFlap


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

ratelimit 60


<-- Ratelimit


      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"
```

相關資訊

Cisco IOS內嵌式事件管理員4.0

EEM的最佳實踐和有用的指令碼

關於此翻譯

思科已使用電腦和人工技術翻譯本文件，讓全世界的使用者能夠以自己的語言理解支援內容。請注意，即使是最佳機器翻譯，也不如專業譯者翻譯的內容準確。Cisco Systems, Inc. 對這些翻譯的準確度概不負責，並建議一律查看原始英文文件（提供連結）。