如何识别Cisco CallManager重新启动作为失败或服务关闭

目录

简介

本文描述在Cisco CallManager失败和服务关闭之间的区别。本文也提供步骤报告Cisco CallManager失败和使思科技术支持排除故障问题。

先决条件

要求

本文档没有任何特定的要求。

使用的组件

本文档中的信息基于以下软件版本：

- Cisco CallManager 3.x 和 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。如果您使用的是真实网络，请确保您已经了解所有命令的潜在影响。

规则

有关文档规则的详细信息，请参阅 Cisco 技术提示规则。

在Cisco CallManager故障和关闭之间的区别

故障

在Cisco CallManager代码的一bug引起CallManager失败。有失败三种主要类型：

- 访问违例
- 除零
- 未知例外

故障生成Dr.Watson条目，被添附对现有Dr.Watson文件的末端。故障也生成user.dmp文件。这些文件的位置是C:\Documents and Settings\All用户\文档\ DrWatson。

Dr.Watson文件的名称，是文本文件，是drwtsn32.log。

从Run窗口选择**drwtsn32**配置设置。

如何读Dr.Watson文件

完成这些步骤读Dr.Watson文件：

1. 搜索词"当"，用小写，并且查找问题发生的日期和时间。Dr.Watson文件记录所有应用程序失败。一些坠毁记录可能不是Cisco CallManager失败。的坠毁记录示例不是Cisco CallManager失败包括RisDC.exe和aupair.exe。
2. 在您找出问题的日期和时间后，请找出进程标识符(PID)编号，并且搜索任务列表确定哪应用程序失败了。任务列表在此步骤的示例发表。在本例中，失败的应用程序有PID 752和应用程序的名称是SCAN32.exe ：Application exception occurred:
   App: (**pid=752**) When: 9/1/2000 @ 10:23:40.836 Exception number: c0000005 (access violation) *----> System Information <----* Computer Name: CISCO-8VCUWBLUJ User Name: SYSTEM Number of Processors: 1 Processor Type: x86 Family 6 Model 8 Stepping 3 Windows 2000 Version: 5.0 Current Build: 2195 Service Pack: None Current Type: Uniprocessor Free Registered Organization: Cisco Systems Inc. Registered Owner: Cisco User *----> Task List <----* 0 Idle.exe 8 System.exe 144 smss.exe 168 csrss.exe 164 winlogon.exe 216 services.exe 228 lsass.exe 336 ibmpmsvc.exe 380 svchost.exe 424 svchost.exe 576 regsvc.exe 592 MSTask.exe 924 Explorer.exe 992 cmd.exe 972 msiexec.exe 928 tp4mon.exe 856 ibmpmsvc.exe 852 ltmsg.exe 408 RunDll32.exe 428 RunDll32.exe 328 PDirect.exe 620 TP98.exe 968 tphkmgr.exe 948 PRPCUI.exe 668 AUTOCHK.exe 744 tponscr.exe 868 KIX32.exe 520 spoolsv.exe 1164 Avsynmgr.exe 1136 VsStat.exe 1192 Vshwin32.exe 1224 Mcshield.exe 1024 MCUPDATE.exe **752 SCAN32.exe** 1292 drwtsn32.exe 0 _Total.exe
3. 如果失败是Cisco CallManager失败，请注释例外编号确定故障类型。**注意：** 如果需要，路由给适当的开发组不是Cisco CallManager失败的应用程序失败。Application exception occurred:

   App: (pid=752)

   When: 9/1/2000 @ 10:23:40.836

   Exception number: **c0000005 (access violation)** 在本例中，例外编号是c0000005，是。此意味着应用程序尝试了对访问存储器在该应用程序的存储器极限外面操作系统的集。另一种可能的崩溃类型是分界由零。因为此示例显示，除的例外编号零是c0000094 ：Application exception occurred:

   App: (pid=1564)

   When: 1/7/2003 @ 13:16:15.051

Exception number: **c0000094 (divide by zero)** 崩溃类型可以也是未知例外。因为此示例显示，未知例外的例外编号是e06d7363 ：Application exception occurred:

App: (pid=2784)

When: 12/10/2002 @ 09:02:58.202

Exception number: **e06d7363** 当您确定时失败是否是访问违例、分界由零或者未知例外，您能匹配失败用现有Cisco Bug。如果不查找匹配，开发工程师有确定一个好的开始什么发生。

4. 搜索在下，文件的部分词开始定义“签名”失败。 **注意：** 用大写字母出现。文件的此部分包含重要信息六个片段，是：遇到问题线索的编号登记的内容在失败时的此线索在失败时执行的功能导致失败的集合代码语句显示功能地址执行，按顺序，在失败之前的堆叠上一步trace原始堆叠转储，提供关于什么的更多信息在运行时堆叠在失败时此代码提供是访问违例失败Cisco CallManager失败的示例。 黑体文本突出显示六关键要素，以及词，指示文件的此部分：State Dump for **Thread Id 0x998** *!--- This number is the number of the thread that experienced the problem.* eax=00cae82c ebx=02070000 ecx=00e95da0 edx=346984d8 esi=34698970 edi=346984d8
eip=77fcb9b3 esp=05cef34c ebp=05cef358 iopl=0 nv up ei ng nz na pe cy cs=001b
ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000283 *!--- This provides the contents of the registers at the time of the crash.* **function: RtlSizeHeap** *!--- This function executed at the time of the crash.* 77fcb992 0f87aafeffff jnbe
RtlFreeHeap+0x20f (77fcb842) 77fcb998 807d1400 cmp byte ptr
[ebp+0x14],0x0 ss: 0650c92a=?? 77fcb99c 0f8586300000 jne
RtlZeroHeap+0x327 (77fcea28) 77fcb9a2 57 push edi 77fcb9a3
53 push ebx 77fcb9a4 e8646cfbff call
RtlConsoleMultiByteToUnicodeN+0x348 (77f8260d) 77fcb9a9 8b4f0c mov
ecx,[edi+0xc] ds: 34eb5aaa=00003781 77fcb9ac 8b4708 mov
eax,[edi+0x8] ds: 34eb5aaa=00003781 77fcb9af 3bc1 cmp
eax,ecx 77fcb9b1 8901 mov [ecx],eax ds:
00e95da0=00cae82c **FAULT ->77fcb9b3 894804 mov [eax+0x4],ecx ds:
014cbdfe=ec810000** *!--- This is the assembly code statement that resulted in the crash.*
77fcb9b6 744d jz 77fd4405 77fcb9b8 8a4705
mov al,[edi+0x5] ds: 34eb5aaa=81 77fcb9bb a804
test al,0x4 77fcb9bd 0f8521310000 jne RtlZeroHeap+0x3e3 (77fceae4)
77fcb9c3 8a4605 mov al,[esi+0x5] ds: 34eb5f42=d5
77fcb9c6 2410 and al,0x10 77fcb9c8 a810 test
al,0x10 77fcb9ca 884705 mov [edi+0x5],al ds:
34eb5aaa=81 77fcb9cd 0f8555030000 jne RtlSizeHeap+0x3ef (77fcbd28)
77fcb9d3 0fb70f movzx ecx,word ptr [edi] ds: 346984d8=0093
77fcb9d6 8b4510 mov eax,[ebp+0x10] ss: 0650c92a=???????? ***----> Stack Back Trace <----*** *!--- This shows, in order, the addresses of the functions that executed !--- just before the crash.* FramePtr ReturnAd Param#1 Param#2 Param#3 Param#4 Function Name 05CEF358 77FCB733 02070000 34698970 05CEF3D0 00000000 **ntdll!RtlSizeHeap** 05CEF400 7800115C 02070000 00000000 34698978 05CEF454 **ntdll!RtlFreeHeap** 05CEF448 00C0304F 34698978 00545EC2 34698978 34698978 **!free** 05CEF460 00B66F85 00000001 00B6626C 033B3D58 025A6720 !<nosymbols> 05CEFF34 018E736B 025A6720 77E964CB 033C6B20 033C6B20 !<nosymbols> 05CEFF80 780060CE 033B3D58 77E964CB 00000018 033C6B20 !ACE_OS_Thread_Adapter:: invoke µµ 05CEFFEC 00000000 00000000 00000000 00000000 00000000 kernel32!TlsSetValue ***----> Raw Stack Dump <----*** *!--- This provides more information about what was on the run-time stack !--- at the time of the crash.* 05cef34c 00 00 07 02 70 89 69 34 - 00 00 00 00 00 f4 ce 05
....p.i4........ 05cef35c 33 b7 fc 77 00 00 07 02 - 70 89 69 34 d0 f3 ce 05
3..w....p.i4.... 05cef36c 00 00 00 00 54 f4 ce 05 - 78 89 69 34 20 67 5a 02 ....T...x.i4
gZ. 05cef37c 44 5b e3 09 94 f3 ce 05 - 30 e6 b5 00 fc f3 ce 05 D[......0....... 05cef38c
38 29 6a 09 40 5b e3 09 - a8 f3 ce 05 65 e5 b5 00 8)j.@[......e... 05cef39c fc f3 ce 05
38 29 6a 09 - 40 5b e3 09 c4 f3 ce 05 ....8)j.@[...... 05cef3ac 39 e2 b5 00 57 92 89 01 -
30 db 55 02 f5 50 5b 00 9...W...0.U..P[. 05cef3bc e0 f3 ce 05 cc f3 ce 05 - 0f 4f 5b 00
e0 f3 ce 05 ....O[..... 05cef3cc 00 00 07 02 19 00 00 00 - 01 f4 ce 05 f8 2b cf 21
.............+.! 05cef3dc f8 2b cf 21 01 f1 ce 05 - 28 ff ce 05 70 f3 ce 05
.+.!....(...p... 05cef3ec 98 ef ce 05 38 f4 ce 05 - a7 9d fb 77 90 26 f8 77
....8......w.&.w 05cef3fc 01 00 00 00 48 f4 ce 05 - 5c 11 00 78 00 00 07 02

```
....H...\..x.... 05cef40c  00 00 00 00 78 89 69 34 - 54 f4 ce 05 04 fa ce 05
....x.i4T....... 05cef41c  20 67 5a 02 02 00 00 00 - 64 00 00 00 5c 00 00 00
gZ.....d...\... 05cef42c  fe 08 00 00 00 00 00 00 - 98 ef ce 05 28 ff ce 05
............(... 05cef43c  b8 ff 00 78 50 32 03 78 - ff ff ff ff 60 f4 ce 05
...xP2.x....`... 05cef44c  4f 30 c0 00 78 89 69 34 - c2 5e 54 00 78 89 69 34
O0..x.i4.^T.x.i4 05cef45c  78 89 69 34 34 ff ce 05 - 85 6f b6 00 01 00 00 00
x.i44....o...... 05cef46c  6c 62 b6 00 58 3d 3b 03 - 20 67 5a 02 98 f6 e6 36  lb..X=;.
gZ....6 05cef47c  98 f6 e6 36 00 00 00 00 - 00 00 00 00 00 00 00 00  ...6............
```

这六个位信息构成一部分的，但是不是所有，失败签名。 定义了签名信息的其余是：故障类型(访问违例、除零或者未知例外)在失败前执行的最后信号分配层(SDL)跟踪语句**注意：** 有使用在失败前的最后SDL文件，以及Dr.Watson文件，所有失败的随员烦扰。分布式缺陷跟踪系统(DDTS)的此签名信息(最后SDL文件、最后Cisco CallManager文件和Dr.Watson文件)随员记录，当您创建一新的失败DDTS。如果匹配已经存在并且有同一个根本原因与DDTS的新的失败，此信息是相同的：例外种类在失败时执行功能的名称功能的名称在堆叠上一步trace的**注意：** 这些名称在Dr.Watson文件总是没出现。在标记旁边出现的集合代码语句最后SDL跟踪线路，应该是非常类似的寄存器、内存地址和其他信息的内容能与存在另一DDTS的信息有所不同，即使失败有同一个根本原因。 如果运行Cisco CallManager，一个不同的版本地址变化。 如果运行Cisco CallManager同一个版本，在集合的地址编码部分，并且在堆栈跟踪部分是同样。

5. 收集这些文件调试失败：drwtsn32.loguser.dmp最后SDL和Cisco CallManager跟踪文件，从大约5分钟在失败前和5分钟在重新启动以后。proglog文件**注意：** 收集这些仅文件在Cisco CallManager版本3.2及以后。若有事件日志，系统和应用程序。性能监控程序(perfmon)若有日志。

DBLException错误

您看到在Cisco CallManager发布器和用户应用程序日志的此错误消息：

```
Error: DBLException - DBL Exception.
  ErrorCode: 8
  ExceptionString: Invalid parameter
  UNKNOWN_PARAMNAME:Text: addDevice
  App ID: Cisco CallManager
  Cluster ID: XXXX-Cluster
  Node ID: 192.168.0.2
Explanation: Severe database layer interface error occurred.
Recommended Action: Contact TAC..
```

或者：

```
A COM error occurred during processing. (6)

Details:
Error No. -2147219962 (0x80040606):
CDBLException Dump: [COM Error] COM Error Description = []
```

当IP电话拒绝从注册或由于在发布服务器和用户数据库之间时的一残破的订阅此种错误出现。通过使用DBLHelper工具，此问题可以解决。关于DBLHelper的更多信息，参考使用DBLHelper重建中断的Cisco CallManager集群SQL预订。

此错误能也生成由于Cisco数据库层箴言报的服务失败。执行这些步骤解决问题：

1. 去**Start > Programs > Administrative Tools** >Component**服务**。
2. 展开**组件服务 > 计算机 > 我的电脑 > COM+ 应用程序**。
3. 开始**MSDTC** (分布式处理协调员)服务，如果它显示终止。

关闭

其他种Cisco CallManager重新启动是关闭。关闭是，当Cisco CallManager无法有效运行并且关闭自己时。 关闭归入两个类别：

- 初始化超时
- SDL计时器和SDL路由器线索死亡

如果Cisco CallManager关闭自己，您查找在Callmanager跟踪的最后几种跟踪线路的关闭。 示例如下：

```
03/22/2003 14:32:16.562 Cisco CallManager|CallManagerFailure - Indicates
some failure in the Cisco CallManager system. Host name of hosting
node.:NEROCM1 IP address of hosting node.:172.27.27.224 Reason code.:4 Additional Text
[Optional]: App ID:Cisco CallManager Cluster ID:NEROCM1-Cluster Node
ID:172.27.27.224|<CLID::NEROCM1-Cluster><NID::172.27.27.224><CT::Alarm>
```

在本例中，原因代码是4 此列表提供从Cisco CallManager代码的关闭原因代码：

```
class CallManagerFailureAlarm : public CallManagerAlarmCatalog {
public:
            enum Reason {
                        Unknown = 1,
                        HeartBeatStopped = 2,
                        RouterThreadDied =3 ,
                        TimerThreadDied = 4,
                        CriticalThreadDied = 5,
                        DeviceMgrInitFailed = 6,
                        DigitAnalysisInitFailed = 7,
                        CallControlInitFailed = 8,
                        LinkMgrInitFailed = 9,
                        DbMgrInitFailed = 10,
                        MsgTranslationInitFailed = 11,
                        SupServiceInitFailed = 12,
                        DirectoryInitFailed = 13
            };
```

而其他原因更加普通，原因1和原因2是少见的情况内部关闭。原因3表明SDL路由器线索终止了答复。 原因4表明SDL计时器线索终止了答复。 原因5 – 13与初始化计时器火关连。

初始化超时

当Cisco CallManager服务首先开始时， CallManager进程箴言报(CMProcMon)线索开始。然后，MmmanInit线索开始，产生所有其他进程。其次， SDL路由器线索开始。此线索处理从一进程发送到另一个的信号。全部三线索同时开始。当MmmanInit线索开始其他进程时， CMProcMon线索和SDL路由器线索启用并且运行。

当MmmanInit开始多种进程时， CMProcmon和SDL需要是正在运行的。MmmanInit线索开始这些进程，按此顺序：

1. 数据库(ProcessDb)**注意：** ProcessDb是Cisco CallManager接口对Database Layer (DBL)代码。同时， MmmanInit代码也启动内部一定数量的其他的Cisco CallManager，独立进程。这些进程包括H225Handler、MGCPBhHandler和部门经理。
2. 地区
3. AARNeighborhood
4. 位置
5. 路由计划

6. 数字分析
7. 呼叫控制
8. 附加服务功能包括呼叫暂留、转发、会议和转移。
9. 设备
10. 目录
11. 呼叫搜索空间管理器(CSSManager)
12. 每日定时管理器(TODManager)

这些任务的成就发生系列。 每一个十二任务有与任务产生关联的一个计时器。当任务开始，此计时器启动。如果计时器火，在读的任务完成前， Cisco CallManager终止并且打印SDL trace ：

```
Critical thread death: name of the timer which fired
```

此列表显示其中每一个计时器，以及开始计时器和SDL信号终止计时器的SDL信号。如果适当，设置跟踪级别“InitDone”信号在SDL trace出现。(您在0x8000CB15的集Sdltracetypeflag。)

这些默认计时器根据Cisco CallManager版本4.1(2)。如果运行另外版本，也许是有些不同的。

1. 数据库初始化时间(默认为900秒) -此时间的起动信号是“启动”信号被发送对MmmanInit进程。您在SDL trace看到此。
2. 地区初始化时间(默认为120秒)。
3. AARNeighborhoods初始化时间(默认为90秒)。
4. 位置初始化时间(默认为90秒)。
5. 路由计划初始化时间(默认为600秒)。
6. 数字分析初始化时间(默认为900秒)。
7. 呼叫控制初始化时间(默认为90秒)。
8. 附加服务初始化时间(默认为900秒) -起动信号是CcInitDone，并且末端信号是SsInitDone。
9. 设备初始化时间(默认为360秒)。
10. 目录初始化时间(默认为90秒)
11. CSSManager初始化时间(默认为900秒)。
12. 定期TODManager的初始化(默认为900秒)。

在所有任务完成后， Cisco CallManager打开SDL链路对在网络的其他节点运作的CallManager服务。Cisco CallManager也打开SDL链路对计算机电话集成(CTI)在同一节点或另外节点运作在网络的管理器服务。

然后， MmmanInit发送CMInitComplete信号回到CMProcMon线索。 当CMProcMon首先开始时，启动Cisco CallManager初始化的一个60分钟硬编码的计时器。 计时器有命名CMInitComplete_WaitTime。(此计时器不是服务参数;计时器不可配置。)如果CMProcMon线索在60分钟内不收到CMInitComplete信号， Cisco CallManager终止并且发表读的跟踪语句：

```
Timed out waiting for CMInitComplete signal
```

如果任何一个十二项初始化任务发生故障，或者，如果这些任务的总时间超出60分钟， Cisco CallManager终止。

注意： CMInitComplete_WaitTime计时器曾经是硬编码对10分钟。 作为Cisco Bug ID CSCdx31622 (仅限注册用户)一部分，此hard code更改对60分钟。 更改进入3.1(3) Engineering Special (ES)系列，有ES的38作为开始。更改也在3.2(2) ES系列，与ES 11作为开始和在Cisco CallManager 3.3。

如果遇到与初始化计时器火的问题，您也许只需要增加计时器设置解决启动。 如果此更改不解决问题，问题可以是一个缓慢的数据库响应时间造成操作计时。collect选派了DBL跟踪、以及SDL和Cisco CallManager跟踪，如果需要。

收集这些文件调试初始化问题：

- 详细的Cisco CallManager跟踪
- SDL trace**注意：** 设置Sdltracetypeflag为0x8000CB15。
- 详细的DBL trace

SDL计时器和SDL路由器线索死亡

SDL路由器线索是执行最重要的线索在Cisco CallManager应用程序里面。它控制呼叫处理信号发送。 CMProcMon线索一次检查SDL路由器线索的健康每两秒。正如你在这些语句，看到Cisco CallManager跟踪显示此健康检查：

```
02/05/2003 00:30:32.790 Cisco CallManager|CMProcMon - ------Entered Router
Verification|<CLID::USNYTSVOIPPB01-Cluster><NID::10.2.40.11>

02/05/2003 00:30:32.790 Cisco CallManager|CMProcMon - ----Exited Router
Verification|<CLID::USNYTSVOIPPB01-Cluster><NID::10.2.40.11>
```

如果CMProcMon线索输入并且退出路由器验证， SDL路由器线索响应对健康检查和优良是。

然而，如果SDL路由器线索不响应，您看到，在Cisco CallManager跟踪的，作为此显示时：

```
CMProcMon - ----Entered While loop ++++ TimeAtWhileEntry: [some number here],
TimeBeforeSleep: [another number], TimeAfterSleep: [a third number], sleepTimeWas :
[4th number"
```

在此紧急， SDL路由器线索一次接收检查每秒钟期限20秒。 在20第二个期限，如果线索在任何时间响应，正常操作恢复，并且SDL路由器线索的健康再次接收验证每2秒。 如果，然而， SDL路由器线索不响应对检查在20秒期间， Cisco CallManager应用程序关闭了。此语句在SDL trace出现：

```
000177508| 01/12/31 07:28:40.389| 001| AlarmErr  |
 |                 |                 |                 |
| AlarmClass: CallManager, AlarmName: CallManagerFailure, AlarmSeverity: Error
AlarmMessage: , AlarmDescription: Indicates some failure in the Cisco CallManager
 system.,
AlarmParameters:  HostName:CCM-PUB, IPAddress:10.5.162.180, Reason:3, Text:, AppID:Cisco
CallManager, ClusterID:CCM-PUB-Cluster, NodeID:10.5.162.180,
```

注意在此跟踪语句文本的原因代码3。 代码意味着SDL路由器线索终止了答复，因此Cisco CallManager关闭了。

SDL路由器线索关闭的最可能原因是缺乏系统资源。另一应用程序长期以来使用了多数或所有CPU时间，至少20秒。 此活动是性能监控程序为什么是重要调试此种关闭。

测试的其他种关闭是SDL计时器线索关闭。 当在内部Cisco CallManager时钟和外部操作系统的时钟之间的差别超出16秒， SDL计时器线索关闭发生。 当SDL计时器线索关闭发生时，此trace在Cisco CallManager跟踪出现：

```
03/22/2003 14:32:16.562 Cisco CallManager|CallManagerFailure - Indicates
some failure in the Cisco CallManager system. Host name of hosting
node.:NEROCM1 IP address of hosting node.:172.27.27.224 Reason code.:4 Additional Text
[Optional]: App ID:Cisco CallManager Cluster ID:NEROCM1-Cluster Node
ID:172.27.27.224|<CLID::NEROCM1-Cluster><NID::172.27.27.224><CT::Alarm>
```

Cisco CallManager一次通常检查计时器线索每秒钟。 重视离开作为“期望时间的Cisco CallManager添加1秒到当前操作系统的时间和存储”。 然后， 1秒的Cisco CallManager睡眠。 在Cisco CallManager醒后，检查新的操作系统的时间并且减去期望时间。 如果这两时间之间的差异是超过1秒，此警告语句在Cisco CallManager跟踪出现：

```
CMProcMon::star_sdlVerification - Test Timer exceeded minimum timer latency
threshold of  1000 milliseconds, Actual latency: 1630 milliseconds
```

在此语句的显示内部Cisco CallManager SDL计时器线索运行减慢。 这里，在Cisco CallManager期望时间和实际操作系统的时间之间的区别是1.63秒。

如果此差异超出16秒， Cisco CallManager关闭并且提供关闭原因代码4

SDL计时器线索关闭的最可能原因是缺乏Cisco CallManager的CPU时间。 另一应用程序，例如VirusScan或STI备份，使用了大多CPU资源至少16秒。 Perfmon日志是重要确定此种关闭的根本原因。

如果Cisco IP电话应用程序备份长期以来运行时间在高CPU利用率，系统崩溃能出现。关于如何避免此系统崩溃的信息，参考：

- 检查在备份工具的设置避免本文Cisco CallManager服务崩溃的高CPU部分

收集这些文件一旦SDL路由器线索或SDL计时器线索关闭了：

- 详细的Cisco CallManager跟踪
- SDL trace**注意：** 设置Sdltracetypeflag为0x8000CB15。
- 若有， Perfmon跟踪显示所有进程CPU利用率百分比在方框运行**注意：** 您能远程捕获这些跟踪减少在Cisco CallManager服务器的CPU的性能影响。

如何报告Cisco CallManager故障到思科技术支持

Cisco CallManager事故的实际原因的诊断是困难。为了确定原因和加快解决方案，思科技术支持要求您收集跟踪和Dr.Watson日志和上传信息到思科技术支持案例笔记。您发送案例注释对attach@cisco.com并且提供在电子邮件标题栏的案例编号。步骤为：

1. 从30分钟和以前15分钟的收集的Cisco CallManager跟踪文件在失败以后。跟踪的位置是C:\Program Files\Cisco\Trace\CCM。
2. 从30分钟和以前15分钟的收集的SDL跟踪文件在失败以后。跟踪的位置是C:\Program Files\Cisco\Trace\SDL\CCM。
3. 收集user.dmp和drwtsn32.log文件。文件的位置是C:\Documents and Settings\All用户\文档\DrWatson。
4. 选择**Start > Programs > Administrative Tools > Event Viewer**搜集从事件查看器的系统和应用事件日志日志文件。如果事件日志数据不是必要的，您能跳到此步骤。 但是，请转存系统和应用程序事件并且在失败前过滤从大约30分钟的仅事件。在您发送他们对思科技术支持前，请调查这些事件。您能找到值得更多注意的事件。**警告：** 小心，如果使用事件查看器，一内置的Microsoft程序，转存这些事件到文本文件。在有高CPU利用率的系统中，此使用事件查看器能容易地使从CPU的其他进程挨饿。这些进程包括维护电话注册的Cisco CallManager Keepalive进程。您在各自的日志能以名称elogdmp.exe使用共享件工具转存所有条目到文本文件。当您使用elogdmp.exe工具时， CPU暗示是微不足道的。发出从DOS提示符的此命令：
   elogdmp *COMPUTER_NAME* **Application > AppEvents.txt elogdmp** *COMPUTER_NAME* **System > SysEvents.txt**
5. 压缩所有文件作为压缩文件按此步骤显示，在您电子邮件并且复制文件前的顺序。请使用Winzip版本8压缩文件。 (思科有此工具的一个站点许可证。)一般来说，对一个本地设备的文件副本更加快速的评估的。 您压缩使用较少空间的文件和您比原始文件格式能快速移动这些文件。一起压缩user.dmp和drwtsn32.log文件。立即请发送并且复制此压缩文件。提供一个说明性症状定义并且包括确切的Cisco CallManager版本、适当的设备加载和Cisco IOS软件版本。如果任何特殊补丁程序是在使用中的，请保证您做此事实结算。一起压缩Cisco

CallManager和SDL跟踪文件。当您等待联系方式时，请发送并且复制此压缩文件。一起压缩perfmon日志。当您等待联系方式时，请发送并且复制此压缩文件。一起压缩事件日志条目。当您等待联系方式时，请发送并且复制此压缩文件。

6. 在您采集了所有跟踪和日志后，请压缩文件并且发送压缩文件对attach@cisco.com。提供在电子邮件标题栏的案例编号。

相关信息

- Cisco CallManager 服务崩溃
- 排除Cisco CallManager崩溃故障
- 语音技术支持
- 语音和统一通信产品支持
- Cisco IP 电话故障排除
- 技术支持和文档 - Cisco Systems