CHAPTER 10

CSM での TCL スクリプトの使用

この章では、コンテント スイッチングの設定方法について説明します。

Content Switching Module（CSM; コンテント スイッチング モジュール）により、Toolkit Command Language（TCL）スクリプトをアップロードし、CSM 上で実行できます。TCL は、ネットワーク コミュニティで普及しているスクリプト言語です。また、TCL には開発されたスクリプトに関連する大量のライブラリがあり、さまざまなサイトから容易にアクセスできます。TCL スクリプトを使用すると、カスタム TCL スクリプトを作成し、カスタム ヘルス プローブまたはスタンドアロン タスクを開発することができます。

CSM の TCL インタープリタ コードは、標準 TCL の Release 8.0 に準拠しています。スクリプトを作成してヘルス プローブを設定するか（「ヘルス モニタリング用プローブの設定」[p.9-2] を参照）、またはヘルス プローブに含まれないタスクを CSM 上で実行することができます。CSM はユーザ側で設定可能な間隔で、定期的にスクリプトを実行します。

CSM Release 3.1(1a) までは、基本ヘルス モニタリング コードに含まれないプロトコル用にヘルス プローブを設定することはできませんでした。現在は、プローブを作成し、特定のアプリケーションに合わせて CSM をカスタマイズできます。CSM Release 3.2 は、UDP ソケット機能をサポートします。

CSM は現在、次の 2 種類のスクリプト モードをサポートします。

- プローブ スクリプト モード — このタイプのスクリプトは、ある種の単純なルールに従って作成する必要があります。このスクリプトの実行は、ヘルス モニタリング モジュールが制御します。

  スクリプトは、スクリプト プローブの一部として定期的に実行され、実行中のスクリプトが返した終了コードによって、特定の実サーバについて、相対的な状態と可用性がわかります。スクリプト プローブの動作は、現在の CSM ソフトウェアで利用できる他のヘルス プローブと同様です。
- スタンドアロン スクリプト モード — このタイプのスクリプトは、一般的な TCL スクリプトです。このスクリプトの実行は、CSM のコンフィギュレーションによって制御します。プローブ スクリプトはスタンドアロン タスクとして実行できます。

TCL 機能をサポートするために、サンプル スクリプトを利用できます。その他のカスタム スクリプトも使用できますが、これらのサンプル スクリプトはシスコシステムズの TAC がサポートしています。サンプル スクリプトのファイルには、次の URL からアクセスしてください。

http://www.cisco.com/cgi-bin/tablebuild.pl/cat6000-intellother

スクリプト ファイルの名前は、c6slb-script.3-3-1.tcl です。

スクリプトのロード

スクリプトは、スクリプト ファイルによって CSM にロードします。スクリプト ファイルには、スクリプトがない場合もあれば、1 つまたは複数のスクリプトが含まれている場合もあります。1 つのスクリプトに 128 KB のスタック スペースが必要です。ヘルス スクリプトは最大 50 ありますので、スクリプト プローブのスタックスペースは、最大で 6.4 MB になります。スタンドアロン スクリプトも実行できますが、そのためにさらにスタック スペースを消費します。

スクリプトのロード例

スクリプトは TFTP サーバ、ブートフラッシュ、スロット 0、またはその他のストレージ デバイスから、**script file** [*file-url*] コマンドを使用してロードできます。

次に、スクリプトをロードする例を示します。

```
Router(config)# module csm 4
Router(config-module-csm)# script file tftp://192.168.1.1/httpProbe.test
```

スクリプト名は、スクリプトのファイル名またはスクリプト ファイル内部で符号化された特殊な名前です。各スクリプト ファイルは、同じファイルに複数のスクリプトが収められている場合があります。スクリプトを実行する、またはスクリプトを使用してヘルス プローブを作成するには、スクリプトのロード元となったスクリプトファイルではなく、スクリプト名を参照する必要があります。

各関連スクリプトを識別できるように、次の行から各スクリプトを始める必要があります。

```
#!name = script_name
```

スクリプトをバンドルしたマスター スクリプト ファイルの例を示します。

```
#!name = SCRIPT1
puts "this is script1"
!name = SCRIPT2
puts "this is script2"
```

マスター スクリプト ファイルに含まれているファイルを調べる例を示します。

```
Router(config)# configure terminal
Router(config-t)# module csm 4
Router(config-module-csm)# script file tftp://192.168.1.1/script.master
Router(config-module-csm)# end
```

次の例では、script.master ファイルにスクリプトが 3 つあります。

```
Router(config)# show module csm 4 file tftp://192.168.1.1/script.master
script1, file tftp://192.168.1.1/script.master
  size = 40, load time = 03:49:36 UTC 03/26/93
script2, file tftp://192.168.1.1/script.master
  size = 40, load time = 03:49:36 UTC 03/26/93
```

ロードしたスクリプト ファイルの内容を表示するには、次のコマンドを使用します。

```
Router(config)# show module csm slot script full_file_URL code
```

指定のスクリプト内のコードを表示する例を示します。

```
router1# show module csm 6 script name script1 code
script1, file tftp://192.168.1.1/script.master
  size = 40, load time = 03:04:36 UTC 03/06/93
#!name = script1
```

スタンドアロン スクリプト タスクとスクリプト プローブの大きな相違は、ヘルス モニタリング用の CSM モジュールによって、ヘルス スクリプトがスケジューリングされるかどうかです。次の条件が当てはまります。

- スクリプト プローブがアクティブでも、スクリプトを変更できます。変更点は、次のスクリプト実行時に、コマンド ライン引数に自動的に適用されます。
- プローブの設定時に、特定のスクリプトがプローブに結合されます。その時点でスクリプトが使用できないと、プローブは NULL スクリプトを使用して実行されます。この状況が発生すると、警告フラグが生成されます。ただし、スクリプトのリロード時には、プローブ オブジェクトとスクリプト間のバインディングは自動的には実行されません。バインディングを実行するには、もう一度、**no script** および **script** コマンドを使用する必要があります。
- スクリプトがロードされたあとは、システムに残り、削除できません。スクリプトを変更するには、スクリプトを変更してから再び **no script file** および **script file** コマンドを入力します。
- 各スクリプトは常に固有の名前で識別されます。同名のスクリプトが 2 つ以上ある場合、CSM は最後にロードされたスクリプトを使用します。スクリプト名が重複していると、CSM によって警告メッセージが生成されます。

TCL スクリプトのリロード

スクリプト ファイルをロードすると、スクリプトのロード元ファイルとは無関係に、そのファイルに含まれていたスクリプトが CSM に組み込まれます。その後、スクリプト ファイルを変更する場合には、**script file** コマンドを使用して、スクリプト ファイルをリロードし、CSM 上で変更できるようにします（詳細については、『*Catalyst 6500 Series Content Switching Module Command Reference*』を参照してください）。次に、スクリプトをリロードする例を示します。

```
router(config)# module csm 4
router(config-module-csm)# no script file tftp://192.168.1.1/script.master
router(config-module-csm)# script file tftp://192.168.1.1/script.master
Loading script.master from 192.168.1.1 (via Vlan100): !!!!!!!!!!!!!!!
[OK - 74804 bytes]
router(config-module-csm)# end
```

実行コンフィギュレーションから **script file** コマンドを削除する場合は、**no script file** コマンドを使用します。このコマンドによってファイルに含まれていたスクリプトがアンロードされるわけではありません。また、CSM 上で実行中のスクリプトも影響を受けません。ロードしたスクリプトをアンロードすることはできません。ロードしたスクリプトが不要になっても、スクリプトを削除する必要はありません。

TCL スクリプトおよび CSM

CSM Release 4.1(1) の TCL スクリプト機能は、TCL 8.0 ソース ディストリビューション ソフトウェアに準拠しています。CSM TCL の変更により、スタンダード TCL ライブラリとは異なる他のプロセスを呼び出すために中断したり、同時に TCL インタープリタを実行したりすることが可能になりました。CSM TCL ライブラリは、**file**、**fcopy**、およびその他の標準 TCL ファイル入出力コマンドをサポートしません。

表 10-1 に、CSM がサポートする TCL コマンドを示します。

表 10-1　CSM がサポートする TCL コマンド

コマンド			
一般的な TCL コマンド			
append	array	binary	break
catch	concat	continue	error
eval	exit	expr	fblocked
for	foreach	format	global
gets	if	incr	info
join	lappend	lindex	linsert
list	llength	lrange	lreplace
lsearch	lsort	proc	puts
regexp	regsub	rename	return
set	split	string	subst
switch	unset	uplevel	upvar
variable	while	namespace	
時間関連のコマンド			
after	clock	time	
ソケット コマンド			
close	blocked	fconfigured	fileevent
flush	eof	read	socket
update	vwait		

表 10-2 に、CSM がサポートしない TCL コマンドを示します。

表 10-2　CSM がサポートしない TCL コマンド

一般的な TCL コマンド			
cd	fcopy	file	open
seek	source	tell	filename
load	package		

表 10-3 に、CSM 固有の TCL コマンドを示します。

UDP コマンド セットにより、Scotty ベースの TCL スクリプトが CSM 上で実行されます。Scotty はソフトウェア パッケージの名前で、高レベル、ストリングベースの API を使用するサイト特有のネットワーク管理ソフトウェアの実装を可能にします。すべての UDP コマンドは、その他の CSM TCL コマンド同様、スレッド セーフ（複数のプログラム間でデータ共有が可能）です。

表 10-3　CSM 固有の TCL コマンド

コマンド	定義
disable_real *serverfarmName realIp port ,-1* | **all** *probeNumId probeNameId*	PROBE_FAIL ステートにすることで、サーバ ファームの実サーバをディセーブルにします。成功した場合、このコマンドは 1 を返します。失敗した場合は 0 を返します。 `disable_real SF_TEST 1.1.1.1 -1 10 cisco` （注）　サーバ ファーム名は、CSCec72471 ごとに大文字を使用する必要があります。
enable_real *serverfarmName realIp port ,-1* | **all** *probeNumId probeNameId*	PROBE_FAIL ステートから動作可能な状態に実サーバをイネーブルにします。成功した場合、このコマンドは 1 を返します。失敗した場合は 0 を返します。 `enable_real SF_TEST 1.1.1.1 -1 10 cisco` （注）　サーバ ファーム名は、CSCec72471 ごとに大文字を使用する必要があります。
gset *varname value*	同じスクリプトで実行されているすべてのプローブ スレッドに対してグローバルな変数を設定することで、プローブ状態を保存できます。このコマンドは、プローブ スクリプトに対してのみ機能します。スタンドアロン スクリプトには適用されません。 プローブ スクリプトの変数は、1 つのプローブ スレッド内でしか認識できません。プローブが終了するたびに、すべての変数は消去されます。たとえば、プローブ スクリプトに「gset x 1 ; incr x」が含まれている場合、変数 x にはプローブごとに 1 が追加されます。 • スクリプトから変数の値を取得するには、*var* または $*var* を設定します。 • スクリプトから変数の値をリセットするには、unset *var* を設定します。 • 現在の変数の値を表示するには、**show module csm** *slot* **tech script** コマンドを使用します。詳細については、「プローブ スクリプトのデバッグ」（p.10-13）を参照してください。

表 10-3　CSM 固有の TCL コマンド（続き）

コマンド	定義
socket -graceful host A.B.C.D port	デフォルトでは、すべての CSM スクリプト プローブがリセットの送信によって TCP ソケットをクローズしています。このアクションは、CSM がアクティブ TCP クローズを初期化するときに、TIME_WAIT ステートを回避するために実行されます。 VxWork で使用できるソケット数が 255 に制限されているため、多くのプローブが同時に実行されている場合、CSM のシステム リソースが足りなくなり、ソケットのオープン時に次のプローブを実行できなくなります。 ソケットに適切なコマンドが入力されると、CSM はリセットの代わりに FIN を使用して TCP 接続をクローズします。このコマンドは、システムのプローブが 250 未満の場合のみ使用します。 `set sock [socket -graceful 192.168.1.1 23]`
ping [*numpacket*] *host A.B.C.D*	このコマンドは CSM Release 3.2 では現在使用できません。 スクリプトのホストに ping を実行できます。成功した場合、このコマンドは 1 を返します。失敗した場合は 0 を返します。 `set result [ping 3 1.1.1.1]` （注）　リモート ホストが各 CSCea67098 の CSM と同じサブネットにない場合、このコマンドはスクリプトをブロックします。

xml *xmlConfigString*	TCL スクリプトから CSM に XML 設定ストリングを送信します。このコマンドは、XML サーバが CSM でイネーブルになっている場合のみ機能します。XML 設定の章を参照してください。
	
	このコマンドは、XML 設定の結果のストリングを返します。
	
	`set cfg_result [ xml {`
	`<config>`
	`<csm_module slot="6">`
	`<serverfarm name="SF_TEST">`
	`</serverfarm>`
	`</config>`
	`}`
	`]`

表 10-4 に、CSM で使用する UDP コマンドを示します。

表 10-4　UDP コマンド

コマンド	定義
udp binary send *handle* [*host port*] *message*	ホストおよびポートで指定された宛先へメッセージを含むバイナリ データを送信します。UDP ハンドルが転送終端にすでに接続されている場合は、*host* および *port* 引数は使用されていない可能性があります。UDP ハンドルが接続されていない場合は、これらの任意の引数を使用して、データグラムの宛先を指定してください。
udp bind *handle* **readable** [*script*] **udp bind** *handle* **writable** [*script*]	スクリプトを UDP ハンドルにバインドできます。UDP ハンドルが読み取り可能または書き込み可能になると、**udp bind** コマンドの 3 番めの引数に応じて、スクリプトが評価されます。*script* 引数なしで **udp bind** コマンドを呼び出すことにより、現在 UDP ハンドルにバインドされているスクリプトを読み出すことができます。バインドを削除するには、空のストリングをバインドします。
udp close *handle*	ハンドルに関連付けられた UDP ソケットをクローズします。
udp connect *host port*	UDP のデータグラム ソケットをオープンして、リモート ホスト上のポートに接続します。接続された UDP ソケットは、メッセージを単一の宛先にしか送信できません。このため、通常は接続された UDP ソケットでは各 **udp send** コマンドの宛先アドレスを指定する必要がないので、コードを短縮することができます。コマンドは、UDP ハンドルを返します。
udp info [*handle*]	このコマンドは、*handle* 引数なしですべての既存の UDP ハンドルのリストを返します。有効な UDP ハンドルを指定することにより、UDP ハンドルのステート情報を取得できます。その結果、送信元 IP アドレス、送信元ポート、宛先 IP アドレス、および宛先ポートを含むリストを取得します。
udp open [*port*]	UDP のデータグラム ソケットをオープンして、UDP ハンドルを返します。ソケットは、指定されたポート番号またはポート名にバインドされます。*port* 引数を指定しなかった場合は、未使用のポート番号が使用されます。
udp receive *handle*	ハンドルに関連付けられた UDP ソケットからデータグラムを受信します。このコマンドは、データグラムを受信する準備ができるまでブロックされています。
udp send *handle* [*host port*] *message*	ホストおよびポートで指定された宛先へメッセージを含む ASCII データを送信します。UDP ハンドルが転送終端にすでに接続されている場合は、*host* および *port* 引数は使用されていない可能性があります。UDP ハンドルが接続されていない場合は、これらの任意の引数を使用して、データグラムの宛先を指定してください。

プローブ スクリプト

多様なアプリケーション セットおよびヘルス プローブを使用してネットワークを管理する必要がある場合、CSM では HTTP ヘルス プローブ、TCP ヘルス プローブ、および ICMP ヘルス プローブなどいくつかの特定タイプのヘルス プローブをサポートします。現在の CSM ソフトウェア リリースでサポートされる基本的なヘルス プローブ タイプは、ネットワークの求める特定のプローブ動作をサポートしないことがあります。より柔軟なヘルス プローブ機能をサポートするために、CSM は現在、TCL スクリプトをアップロードし、CSM 上で実行できるようになっています。

プローブに関連付けられたサーバ ファーム内の個々の実サーバに対して、CSM が定期的に実行するスクリプト プローブを作成できます。スクリプトの終了コードに基づいて、実サーバはヘルシー、サスペクト、失敗と判断されます。プローブ スクリプトは、サーバへのネットワーク接続を確立し、サーバにデータを送信し、応答を確認することによって、実サーバの状態をテストします。この TCL スクリプト環境は柔軟性が高いので、プローブ機能が使用できるようになります。

各タイム インターバルを設定すると、CSM の内部スケジューラがヘルス スクリプトをスケジューリングします。単一のプローブ実行が目的となるように、スクリプトを作成してください。**exit** コマンドを使用して、プローブの結果を宣言する必要があります。

一般に、ヘルス スクリプトが実行する動作は次のとおりです。

- IP アドレスに対してソケットをオープンします。
- 1 つまたは複数の要求を送信します。
- 応答を読み取ります。
- 応答を分析します。
- ソケットをクローズします。
- exit 5000（成功）または exit 5001（失敗）を使用することによって、スクリプトを終了します。

新しい **probe** *probe-name* **script** コマンドを使用すると、Cisco IOS ソフトウェアでスクリプト プローブを作成できます。このコマンドによって、従来の CSM ヘルス プローブ サブモード（HTTP、TCP、DNS、SMTP など）と同様のプローブ サブモードが開始されます。プローブ スクリプト サブモードには、従来のプローブ サブモード コマンド（**failed**、**interval**、**open**、**receive**、および **retries**）が含まれます。

さらに、新しい **script** *script-name* コマンドがプローブ スクリプト サブモードに追加されました。このコマンドでは 5 つの引数を使用できます。各引数は、実行時にヘルス プローブ機能の一部としてスクリプトに渡されます。

プローブ スクリプトの記述例

次に、ヘルス スクリプトを使用して HTTP サーバを調べる記述例を示します。

```
Router(config)# !name = HTTP_TEST

# get the IP address of the real server from a predefined global array csm_env
set ip $csm_env(realIP)
set port 80
set url "GET /index.html HTTP/1.0\n\n"

# Open a socket to the server. This creates a TCP connection to the real server
set sock [socket $ip $port]
fconfigure $sock -buffering none -eofchar {}

# Send the get request as defined
puts -nonewline $sock $url;

# Wait for the response from the server and read that in variable line
set line [ read $sock ]

# Parse the response
if { [ regexp "HTTP/1.. (\[0-9\]+) " $line match status  ] } {
  puts "real $ip server response : $status"
}

# Close the socket. Application must close the socket once the
# is over. This allows other applications and tcl scripts to make
# a good use of socket resource. Health monitoring is allowed to open
# only 200 sockets simultaneously.
close $sock

# decide the exit code to return to control module.
# If the status code is OK then script MUST do exit 5000
# to signal successful completion of a script probe.
# In this example any other status code means failure.
# User must do exit 5001 when a probe has failed.
if { $status == 200 } {
    exit 5000
} else {
    exit 5001
}
```

環境変数

ヘルス プローブ スクリプトは、前もって定義された TCL 配列によって、さまざまな設定項目にアクセスできます。この配列の最も一般的な用途は、スクリプトの起動中に疑わしいとされたもの（サスペクト）の、現在の実サーバ IP アドレスを調べることです。

CSM 上でスクリプト プローブが実行されるたびに、csm_env という特殊な配列がスクリプトに渡されます。この配列には、スクリプトが使用する重要なパラメータが格納されます。

（注）　ここで使用する環境変数の情報は、プローブ スクリプトのみに適用されます。スタンドアロン スクリプトには適用できません。

表 10-5 に、csm_env 配列のメンバーを示します。

表 10-5　csm_env 配列のメンバー

メンバー名	内容
realIP	サスペクトの IP アドレス
realPort	サスペクトの IP ポート
intervalTimeout	設定されているプローブ インターバル（秒単位）
openTimeout	このプローブに設定されているソケット オープン タイムアウト
recvTimeout	このプローブに設定されているソケット受信タイムアウト
failedTimeout	設定されている障害タイムアウト
retries	設定されている再試行回数
healthStatus	現在のサスペクトのヘルス ステータス

終了コード

プローブ スクリプトは終了コードを使用して、各種の内部状態を示します。終了コード情報は、スクリプトが正常に動作しなかった場合のトラブルシューティングに役立ちます。使用できる終了コードは、**exit 5000** および **exit 5001** だけです。プローブ スクリプトは、スクリプトの終了コードを使用して、実サーバの相対的な正常さとアベイラビリティを示します。スクリプトは exit（5000）を呼び出すことによって、サーバがプローブに正常に応答したことを示します。exit（5001）を呼び出した場合、サーバがヘルス プローブに正しく応答しなかったことを示します。

プローブ スクリプトが失敗して 5001 で終了した場合、対応するサーバは PROBE_FAILED とマーキングされ、一時的にサーバ ファームからディセーブルにされます。CSM はサーバの調査を継続します。プローブが正常に再接続されて 5000 で終了した場合、CSM はサーバのステータスに OPERATIONAL とマーキングして、サーバ ファームからそのサーバを再度イネーブルにします。

exit 5001 のスクリプトに加え、次の状態はスクリプトを失敗させて、PROBE_FAILED（サスペクト）とマーキングされる可能性があります。

- TCL エラー — スクリプト内に、TCL インタープリタで検出されたエラーが含まれる場合に発生します（構文エラーなど）。構文エラー メッセージは特殊な変数 **erroInfo** に保存されるため、**show mod csm X tech script** コマンドで表示できます。
- スクリプトの停止 — 無限ループまたはスクリプトが無効な IP アドレスに接続しようとしたことが原因です。各スクリプトは設定された時間内にタスクを完了しなければなりません。スクリプトがタスクを完了できなかった場合、スクリプト コントローラがスクリプトを中止します。サスペクトは暗黙の失敗とみなされます。
- エラー条件 — 接続のタイムアウトまたはピアによる接続拒否も、暗黙の失敗として扱われる場合に発生します。

表 10-6 に、CSM で使用される終了コードをすべて示します。

表 10-6　CSM の終了コード

終了コード	サスペクトの意味および動作上の影響
5000	サスペクトは正常。ユーザが制御。
5001	サスペクトは失敗。ユーザが制御。
4000	スクリプト打ち切り。ステートの変化はそのときのシステム ステータスによって決まります。システム用に予約。
4001	スクリプト終了。サスペクトは失敗。システム用に予約。
4002	スクリプトはパニック。サスペクトは失敗。システム用に予約。

表 10-6　CSM の終了コード（続き）

終了コード	サスペクトの意味および動作上の影響
4003	スクリプトは内部操作またはシステムコールに失敗。サスペクトは失敗。システム用に予約。
不明	変化なし。

EXIT_MSG 変数

デバッグが目的であれば、特殊な変数 EXIT_MSG にスクリプト デバッグ情報を設定すると効果的です。EXIT_MSG 変数を使用して特定の Cisco IOS **show** コマンドを入力することで、スクリプトの実行ポイントを追跡できます。

次に、EXIT_MSG 変数を使用して、スクリプトの終了点を追跡し、スクリプトが機能しなかった理由を調べる方法を示します。

```
set EXIT_MSG "opening socket"
set s [socket 10.2.0.12 80]
set EXIT_MSG "writing to socket"
puts -nonewline $sock $url
```

EXIT_MSG 変数を調べるには、**show module csm** *slot* **tech script** コマンドを使用します。

次に、終了前に実行した最後のスクリプトが EXIT_MSG であるために、EXIT_MSG が「opening socket」に設定された例を示します。

```
router1# show module csm 4 tech script
SCRIPT CONTROLLER STATS
: ==========================================================================
SCRIPT(0xcbcfb50) stat blk(0xcbcfbb0): TCL_test.tclcbcfb50
CMDLINE ARGUMENT:
 curr_id 1 argc 0 flag 0x0::
 type = PROBE
 task_id = 0x0: run_id = 512 ref count = 2
 task_status = TASK_DONE run status = OK
 start time = THU JAN 01 00:15:47 1970
 end time = THU JAN 01 00:17:02 1970
 runs = 1   +0
 resets = 1   +0
 notrel = 0   +0
 buf read err = 0   +0
 killed = 0   +0
 panicd = 0   +0
last exit status= 4000 last Bad status = 4000
Exit status history:
 Status (SCRIPT_ABORT) occured #(1) last@ THU JAN 01 00:17:02 1970
**TCL Controller:
------------------------
tcl cntrl flag = 0x7fffffff
#select(0) close_n_exit(0) num_sock(1)
 MEM TRACK last alloc(0) last size(0) alloc(0) size(0)
 hm_ver (1) flag(0x0) script buf(0xcbf8c00) new script buf(0x0)  lock owner(0x0) sig
taskdel:0 del:0 syscall:0 syslock:0 sig_select  script ptr (0xcbf88f0) id(0)
Config(0xcbcdd78) probe -> 10.1.0.105:80
 tclGlob(0xcbad050) script resource(0xcbcfa28)
#Selects(0) Close_n_exit(0) #Socket(1)
OPEN SOCKETS:
Last erroInfo = couldn't open socket: host is unreachable
    while executing
"socket 10.99.99.99 80 "
    (file "test.tcl" line 2)
Last errorCode = 65
Last panicInfo =
EXIT_MSG = opening socket
```

プローブ スクリプトの実行

プローブ スクリプトを実行するには、スクリプト プローブ タイプを設定し、スクリプト名をプローブ オブジェクトに関連付けます（『*Catalyst 6500 Series Content Switching Module Command Reference*』を参照）。

サーバ ファームおよび仮想サーバにスクリプトをロード、生成、適用してプローブ スクリプトを実行し、結果を表示する手順は、次のとおりです。

ステップ 1　スクリプトをロードします。

```
router1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
router1(config)# module csm 6

router1(config-module-csm)# script file tftp://192.168.10.102/csmTcl.tcl
Loading csmTcl.tcl from 192.168.10.102 (via Vlan100): !
[OK - 1933 bytes]
```

ステップ 2　スクリプト プローブを作成します。

```
router1(config-module-csm)# probe test1 script
rout(config-slb-probe-script)# script CSMTCL
rout(config-slb-probe-script)# interval 10
rout(config-slb-probe-script)# exit
```

ステップ 3　サーバ ファームおよび仮想サーバにプローブを適用します。

```
router1(config-module-csm)# serverfarm test
router1(config-slb-sfarm)# real 10.1.0.105
router1(config-slb-real)# ins
router1(config-slb-real)# probe test1
router1(config-slb-sfarm)# exit
```

ステップ 4　サーバ ファームを仮想サーバに適用します。

```
router1(config-module-csm)# vserver test
router1(config-slb-vserver)# virtual 10.12.0.80 tcp 80
router1(config-slb-vserver)# serverfarm test
router1(config-slb-vserver)# ins
router1(config-slb-vserver)# exit
```

この時点でスクリプト プローブが設定されます。**show module csm slot tech probe** コマンドを使用して、実行されているスクリプトを確認できます。

ステップ 5　スクリプト プローブを停止します。

```
router1(config-module-csm)# serverfarm test
router1(config-slb-real)# no probe test1
router1(config-slb-sfarm)# exit
```

ここでは、スクリプト コマンドの結果を検証する例を示します。

次に、スクリプト情報を表示する例を示します。

```
router1# show module csm 6 script
CSMTCL, file tftp://192.168.10.102/csmTcl.tcl
  size = 1933, load time = 03:09:03 UTC 01/01/70
```

次に、プローブ スクリプトに関する情報を表示する例を示します。

```
router1# show module csm 6 probe
probe           type    port  interval retries failed  open   receive
------------------------------------------------------------------------
TEST1           script        10       3       300     10     10
router1#
```

次に、特定のプローブ スクリプトに関する詳細情報を表示する例を示します。

```
router1# show module csm 6 probe name TEST1 detail
probe           type    port  interval retries failed  open   receive
------------------------------------------------------------------------
TEST1           script        10       3       300     10     10
 Script: CSMTCL
 real                  vserver         serverfarm      policy          status
 --------------------------------------------------------------------------------
 10.1.0.105:80         TEST1           TEST            (default)       OPERABLE
router1#
```

次に、実サーバのプローブ情報を表示する例を示します。

```
router1# show module csm 6 probe real
  real = 10.1.0.105:80, probe = TEST1, type = script,
    vserver = TEST, sfarm = TEST
    status = FAILED, current = 03:26:04 UTC 01/01/70,
    successes = 1, last success = 03:15:33 UTC 01/01/70,
    failures = 4, last failure = 03:26:04 UTC 01/01/70,
    state = Unrecognized or invalid response
    script CSMTCL
    last exit code = 5001
```

プローブ スクリプトのデバッグ

スクリプト プローブのデバッグ方法は次のとおりです。

- verbose モードで実行しているスクリプトに TCL の **puts** コマンドを使用します。

  verbose モードで **puts** コマンドを使用すると、各プローブ サスペクトは CSM コンソールにストリングを出力します。システムで実行しているサスペクトが多いと、リソースもそれだけ必要になるため、CSM コンソールがハングする可能性があります。この機能は、システムに設定されたサスペクトが 1 つである場合にのみイネーブルにすることを推奨します。

- スクリプトで特殊変数 EXIT_MSG を使用します。

  各プローブ サスペクトに自身の EXIT_MSG 変数が含まれています。この変数によりスクリプトのステータスを追跡して、プローブの状態を確認できます。

  次に、スクリプトの EXIT_MSG 変数の使用例を示します。

  ```
  set EXIT_MSG "before opening socket"
  set s [ socket $ip $port]
  set EXIT_MSG " before receive string"
  gets $s
  set EXIT_MSG "before close socket"
  close $s
  ```

  メッセージ受信時にプローブ サスペクトが失敗した場合、ストリングを受信する前に EXIT_MSG = を確認する必要があります。

- **show module csm** *slot* **probe real** [*ip*] コマンドを使用します。

  このコマンドは、システム内で現在アクティブなプローブ サスペクトを表示します。

```
router1# show module csm 6 probe real
  real = 10.1.0.105:80, probe = TEST1, type = script,
    vserver = TEST, sfarm = TEST
    status = FAILED, current = 04:06:05 UTC 01/01/70,
    successes = 1, last success = 03:15:33 UTC 01/01/70,
    failures = 12, last failure = 04:06:05 UTC 01/01/70,
    state = Unrecognized or invalid response
    script CSMTCL
    last exit code = 5001
```

（注）　最後の終了コードには、表 10-6（p.10-10）に示す終了コードのうち 1 つが表示されます。

- **show module csm** *slot* **tech probe** コマンドを使用します。

  このコマンドは現在のプローブ ステータスを表示します（標準およびスクリプト プローブの両方）。

```
router1# show module csm 6 tech probe

Software version: 3.2(1)
------------------------------------------------------------
------------------ Health Monitor Statistics ------------
------------------------------------------------------------
 Probe templates: 1
Suspects created: 1
    Open Sockets in System : 8 / 240
    Active Suspect(no ICMP): 0 / 200
    Active Script Suspect  : 0 / 50
    Num events  : 1

Script suspects: 1
  Healthy suspects: 0
Failures suspected: 0
Failures confirmed: 1

Probe attempts:         927  +927
Total recoveries:         3  +3
Total failures:           6  +6
Total Pending:            0  +0
```

- **show module csm** *slot* **tech script** コマンドを使用して、最後の終了ステータス、固定変数、errorInfo、および EXIT_MSG の出力を検出します。

```
router1# show module csm 6 tech script
SCRIPT(0xc25f7e0) stat blk(0xc25f848): TCL_csmTcl.tclc25f7e0
CMDLINE ARGUMENT:
 curr_id 1 argc 0 flag 0x0::
 type = PROBE
 task_id = 0x0: run_id = 521 ref count = 2
 task_status = TASK_DONE run status = OK
 start time = THU JAN 01 03:51:04 1970
 end time = THU JAN 01 03:51:04 1970

 runs = 13   +11
 resets = 13   +11
 notrel = 0   +0
 buf read err = 1   +1
 killed = 0   +0
 panicd = 0   +0

last exit status= 5001 last Bad status = 5001

Exit status history:

**TCL Controller:
------------------------
tcl cntrl flag = 0x7fffffff
#select(0) close_n_exit(0) num_sock(2)
 MEM TRACK last alloc(0) last size(0) alloc(0) size(0)
 hm_ver (3) flag(0x0) script buf(0xc25ad80) new script buf(0xc25ad80)
 lock owner(0x0) sig taskdel:0 del:0 syscall:0 syslock:0 sig_select
 script ptr (0xc25f038) id(0)
Config(0xc2583d8) probe -> 10.1.0.105:80
 tclGlob(0xc257010)
SCRIPT RESOURCE(0xc25af70)-------
#Selects(0) Close_n_exit(0) #Socket(2)
OPEN SOCKETS:

Persistent Variables

--------------------
   x = 11

Last erroInfo =

Last errorCode =
Last panicInfo =
EXIT_MSG = ping failed : invalid command name "ping"
```

最後の終了ステータスには、終了コード番号（表 10-6[p.10-10] に示す）が表示されます。

固定変数情報は、**gset varname value** コマンドで設定されます（表 10-3[p.10-5] を参照）。

erroInfo 変数は TCL コンパイラで生成されたエラーのリストです。スクリプトに TCL のランタイム エラーが含まれている場合、TCL インタープリタはスクリプトの実行を停止して、erroInfo 変数のエラー情報を保存します。

EXIT_MSG（「EXIT_MSG 変数」[p.10-11] を参照）は、失敗の恐れがある各プローブの詳細なデバッグ情報を表示します。出力が非常に長くなる可能性があるため、次の例のようにキーワードで最初にフィルタリングすることもできます。

```
router1# show module csm slot tech script inc keyword
```

スタンドアロン スクリプト

スタンドアロン スクリプトは、CSM にロードして実行する一般的な TCL スクリプトです。スタンドアロン スクリプトはプローブ スクリプトのように設定してサーバ ファームに適用しないため、CSM の定期的な実行タスクとしてのスケジューリングは行われません。タスクを実行するには、**script task** コマンドを使用する必要があります。

csm_env 環境変数はスタンドアロン スクリプトには適用されません。ただし、プローブ スクリプトの終了コードにスタンドアロン スクリプトに対する特別な意味が含まれていない場合、**exit** コマンドを使用できます。

スタンドアロン スクリプトの記述例

次に、一般的な TCL スクリプトの記述例を示します。

```
#!name = STD_SCRIPT
set gatewayList "1.1.1.1 2.2.2.2"
foreach gw $gatewayList {
    if { ![ ping $gw ]  } {
    puts "-WARNING : gateway $gw is down!!"
  }
}
```

スタンドアロン スクリプトの実行

スタンドアロン スクリプトはスクリプト プローブと異なり、タスクを 1 度だけ実行する TCL スクリプトです。スクリプトのロードが完了すると、実行が終了します。スタンドアロン スクリプトはタスクとしてスクリプトが設定されていないかぎり、定期的に CSM が実行することはありません。**script file** コマンドをスタートアップ コンフィギュレーションに組み込むと、CSM の起動時にコマンドが実行されます。スクリプトは CSM の稼働中、引き続き実行されます。

スタンドアロン スクリプトを実行するには、次の手順を実行します。

ステップ 1　スクリプトをロードします。

```
router1# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
router1(config)# module csm 6
router1(config-module-csm)# script file tftp://192.168.10.102/stdcsm.tcl
Loading stdcsm.tcl from 192.168.10.102 (via Vlan100): !
[OK - 183 bytes]
```

ステップ 2　スタンドアロン タスクとしてスクリプトを実行します。

```
router1(config-module-csm)# script task 1 STD_SCRIPT
```

ステップ 3　スクリプトを再度実行します。

古いタスクを削除して、次のように再度実行できます。

```
router1(config-module-csm)# no script task 1 STD_SCRIPT
router1(config-module-csm)# script task 1 STD_SCRIPT
```

次のように新しいタスク ID を指定することで、新規タスクを開始することもできます。

```
router1(config-module-csm)# script task 2 STD_SCRIPT
```

ステップ 4　スクリプトを停止します。

```
router1(config-module-csm)# no script task 1 STD_SCRIPT
```

ステップ 5　スクリプトのステータスを表示するには、**show** コマンドを使用します。

```
router1#sh mod csm 6 script
STD_SCRIPT, file tftp://192.168.10.102/stdcsm.tcl
router1#sh mod csm 6 script task

task  script                          runs       exit code  status
--------------------------------------------------------------------------
1     STD_SCRIPT                      1          4000       Not Ready
2     STD_SCRIPT                      1          4000       Not Ready
```

特定の実行スクリプトに関する情報を表示するには、**show module csm** *slot* **script task index** *script-index* **detail** または **show module csm** *slot* **script name** *script-name* **code** コマンドを使用します。

スタンドアロン スクリプトのデバッグ

スタンドアロン スクリプトのデバッグは、プローブ スクリプトのデバッグと類似しています（「プローブ スクリプトのデバッグ」[p.10-13] を参照）。複数のスレッドを実行しても問題は起きないので、スクリプトで **puts** コマンドを使用して、デバッグに役立てることができます。

TCL スクリプトの FAQ

ここでは、CSM の TCL スクリプトに関するよくある質問（FAQ）を取り上げます。

- システム リソースはどのように使用されますか？

  VxWorks サポート アプリケーションには、ファイル ディスクリプタが 255 個あり、それが標準入力、標準出力、あらゆるソケット接続（受信または送信）など、すべてのアプリケーション間で分配されます。スタンドアロン スクリプトを作成する場合は、いつソケットをオープンするかについて、慎重に配慮する必要があります。リソース不足を防ぐために、動作の完了後、ただちにソケットをクローズすることを推奨します。ヘルス モニタリング モジュールは、活発に実行しているスクリプトの数を制御することによって、オープン ソケット数を制御します。スタンドアロン スクリプトには、このような制御がありません。

  メモリは考慮事項ではあるものの、大きな制約をもたらすことはありません。モジュールには通常、十分なメモリが装備されているからです。スクリプトごとに 128 KB のスタックを 1 つずつ使用し、残りのメモリが実行時にスクリプトによって割り当てられます。

  スクリプトの実行中もシステムのリアルタイム特性がある程度一定に保てるように、スクリプト タスクにはシステムで最下位のプライオリティが与えられます。スクリプトのプライオリティが低いということは残念ながら、システムが TCL 以外の動作で忙しい場合に、どの TCL スレッドも完了に時間がかかることを意味します。このような状況では、一部のヘルス スクリプトが打ち切られ、未完了のスレッドにエラーのマークが設定されることがあります。スクリプトがエラーにならないように、すべてのスクリプト プローブで再試行値を 2 以上に設定する必要があります。可能なかぎり、固有の CSM プローブ（HTTP、DNS など）を使用してください。スクリプトのヘルス プローブは、サポート外のアプリケーションをサポートする場合に使用してください。ヘルス プローブの目的は、高速化ではなく、機能を提供することです。

  TCL は同期と非同期の両方のソケット コマンドをサポートします。非同期のソケット コマンドは、実接続を待たずに、ただちに戻ります。非同期バージョンのスクリプトを内部で実装するには、この種のコマンドごとに多数のシステムコールを使用する、非常に複雑なコード パスが必要になります。一般にこのような状況は、他のコマンドによるシステム コールの処理中にいくつかのクリティカル リソースが待機する原因となるので、システム速度を低下させます。どうしても避けられない場合以外、スクリプト プローブに非同期ソケットを使用することは推奨できません。ただし、スタンドアロン システムでこのコマンドを使用することは可能です。

- 設定したプローブが実行されているかどうかを知る方法は？

  ネットワークの実サーバ側で Sniffer を実行できます。また、次の **show** コマンドを使用して、プローブが CSM 上で実行されているかどうかを判断できます。

  - プローブが実行されていれば、この例で示すようにプローブを試行した回数が増加していきます。

```
router1# show module csm 6 tech probe
router1#sh mod csm 6 tech probe
Software version: 3.2(1)
------------------------------------------------------------
------------------ Health Monitor Statistics ----------------
------------------------------------------------------------
 Probe templates: 8
Suspects created: 24
    Open Sockets in System : 10 / 240
    Active Suspect(no ICMP): 2 / 200
    Active Script Suspect  : 2 / 50
    Num events  : 24
Script suspects: 24
  Healthy suspects: 16
Failures suspected: 0
Failures confirmed: 8
Probe attempts:        321  +220
Total recoveries:       16  +0
Total failures:          8  +2
Total Pending:           0  +0
```

– プローブが実行されていれば、この例で示すように成功または失敗のカウントが増加します。

```
router1# show module csm 6 probe real
 real = 10.12.0.108:50113, probe = SCRIPT2_2, type = script,
   vserver = SPB_SCRIPT2, sfarm = SCRIPT2_GOOD, policy = SCRIPT2_GOOD,
   status = OPERABLE, current = 22:52:24 UTC 01/04/70,
   successes = 18, last success = 22:52:24 UTC 01/04/70,
   failures = 0, last failure = 00:00:00 UTC 01/01/70,
   state = Server is healthy.
   script httpProbe2.tcl GET /yahoo.html html 1.0 0
   last exit code = 5000
 real = 10.12.0.107:50113, probe = SCRIPT2_2, type = script,
   vserver = SPB_SCRIPT2, sfarm = SCRIPT2_GOOD, policy = SCRIPT2_GOOD,
   status = OPERABLE, current = 22:52:42 UTC 01/04/70,
   successes = 19, last success = 22:52:42 UTC 01/04/70,
   failures = 0, last failure = 00:00:00 UTC 01/01/70,
   state = Server is healthy.
   script httpProbe2.tcl GET /yahoo.html html 1.0 0
   last exit code = 5000
```

また、リセット（RST）の代わりに FIN を使用してソケットをクローズすることもできます。

- リモート ホストが到達不能の場合、UDP プローブが実サーバを PROBE_FAIL ステートにできない理由は？

  UDP プローブは「icmp port unreachable」メッセージを受信して、サーバに PROBE_FAIL をマーキングする必要があります。リモート ホストがダウンした場合、または応答しない場合、UDP プローブは ICMP メッセージを受信しないために、そのプローブのパケットは紛失したとみなされ、サーバは正常の状態にあると判断されます。

  UDP プローブは Raw UDP プローブなので、CSM はプローブの応答ペイロードに単一のバイトを使用します。CSM は UDP アプリケーションから意味のある応答がくることを想定していません。CSM は ICMP 到達不能メッセージを使用して、UDP アプリケーションが到達可能かどうかを判断します。

  受信タイムアウトで ICMP 到達不能の応答がない場合、CSM はプローブが正常に実行されていると判断します。実サーバの IP インターフェイスがダウンまたは切断された場合、UDP プローブは自身で UDP アプリケーションが到達不能にあることを判断できません。指定のサーバの UDP プローブのほかに ICMP プローブを設定する必要があります。

  回避策：常に ICMP を UDP のプローブ タイプで設定します。

- ダウンロード可能なサンプル スクリプトはどこにありますか？

  サンプル スクリプトを使用して TCL 機能をサポートできます。その他のカスタム スクリプトも使用できますが、これらのサンプル スクリプトはシスコシステムズの TAC がサポートしています。サンプル スクリプトのファイルには、次の URL からアクセスしてください。

  http://www.cisco.com/cgi-bin/tablebuild.pl/cat6000-intellother

  スクリプト ファイルの名前は、c6slb-script.3-3-1.tcl です。

- TCL スクリプト情報はどこにありますか？

  TCL 8.0 のコマンド リファレンスには、次の URL からアクセスできます。

  http://www.tcl.tk/man/tcl8.0/TclCmd/contents.html

  TCL UDP コマンド リファレンスには、次の URL からアクセスできます。

  http://wwwhome.cs.utwente.nl/~schoenw/scotty/