式の使用方法

Cisco プライムネットワーク レジストラーは、クライアント クラスのサポートを強化します。クライアントデータベースにクライアントを登録しなくても、要求の内容に基づいてクライアント クラスに要求を配置できるようになりました。また、サブスクライバのアクティブなリース数に基づいてクライアントクラスに要求を配置できるようになり、さまざまな加入者に提供されるサービスのレベルに制限が与えることができるようになります。これは、式を使用した特別な DHCP オプションの処理によって可能です。

DHCPリレー エージェント情報オプション (RFC 3046 で説明されているオプション 82) の値に基づいて、加入者アドレスの制限を設定できます。これらの値は、機密性の高いアドレスを明らかにする必要はありません。オプション 82 サブオプション (リモート IDまたは回線 ID)またはその他の DHCP オプションに対して着信 DHCPDISCOVER 要求パケットを評価する式を作成することによって、個々の加入者に関連付ける値を作成できます。この式は、パケット内で評価される内容に応じて異なる値を返す一連のifステートメントです。これは、事実上、サブスクライバが属するクライアントクラスを計算し、アドレスの割り当てをそのクライアントクラスのスコープに制限します。

（注） 式は DHCP 拡張と同じではありません。式は、クライアント ID の作成やクライアントの検索に一般的に使用されます。拡張 (拡張ポイントの使用を参照) は、要求パケットまたは応答パケットを変更するために使用されます。ここで説明する式も正規表現と同じではありません。

式の使用方法

式処理は、次の場所で使用されます。

- — クライアントクラス検索 ID .Calculating a client-class この式は、着信パケットの内容に基づいてクライアントクラスを決定します。
- Creatingthe key to look —upクライアント検索 ID in . the client-entry database 式の評価結果のキーを使用して、クライアント エントリ データベースにアクセスします。
- Creatingthe ID to use to —limit制限 ID clients . of the same subscriber これは、他のクライアントがこのサブスクライバに関連付けられているかどうかを確認するために使用する ID です。これは DHCPv4 (DHCPv6 ではない) に対してのみサポートされます。
- **オプション値の作成**：オプションに対して式を使用する （37 ページ） を参照してください。

この種の処理は、次のシナリオで発生します。

1. DHCP サーバーは、クライアント クラスルックアップ ID 式に基づいてクライアント クラスを取得しようとします。クライアントクラスを計算できない場合は、通常のMACアドレスメソッドを使用してクライアントを検索します。
2. サーバーがクライアントクラスを計算できる場合は、クライアント参照IDを返すクライアントルックアップ ID式の評価に基づいて、クライアント エントリ検索を実行する必要があるかどうかを判断します。そのような ID を持つ場合は、それを使用してクライアントを検索します。そのような ID がない場合は、計算されたクライアント クラス値を使用してアドレスを割り当てます。
3. サーバーがクライアントルックアップIDを使用し、クライアント・エントリーを見つけた場合、クライアントのデータを使用します。クライアント エントリが見つからない場合は、計算されたクライアント クラス データまたは既定のクライアント クラス データが使用されます。

DHCPv4 の場合、割り当てられたアドレスの上限を、ポリシー・レベルで同一の制限 id値を持つネットワークまたはLANセグメント上のクライアントに設定することもできます。ポリシーの制限カウント属性を使用して、この上限を正の整数として設定します。同様の処理は、v6 クライアント クラスルックアップ IDとv6 クライアントルックアップ ID式を使用して DHCPv6 で可能です。

IP アドレスを加入者に制限するために設定する値は次のとおりです。

- ポリシーの場合は、制限カウント属性を正の整数に設定します。
- クライアント クラスの場合、limit-id属性とクライアントルックアップ ID属性を式に設定し、クライアント クラスに対してlimit-limit-client-class-name属性を設定します。
- クライアントの場合は、クライアント クラスに対して、クライアント クラス名の上限属性を設定します。

使用する式については、 を式の作成 （4 ページ）参照してください。

式の入力

属性定義に単純な式を含めるか、式ファイルに複雑な式を含め、属性定義でファイルを参照することができます。いずれの場合も、最大許容文字は 16 KB です。

CLI で設定されるほとんどの式はテキスト ファイルに格納され、その後、必要な設定属性に関連付けられます。このファイルのデフォルト パスは、現在の作業ディレクトリです。テキスト ファイルに格納せずに、CLI で単純な式を直接設定できます。単純な式は、CLI に入力する際に、次の規則に従う必要があります。

- 1 つのコマンド ラインに制限する必要があります。
- 式全体を二重引用符 ( )" "で囲む必要があります。
- 埋め込まれた二重引用符はバックスラッシュ(\) でエスケープする必要があります。

クライアントクラスルックアップ IDを設定する単純な式の例を次に示します。

```
\"limit\"
```

クライアントクラスの制限 idを設定するために、もう少し詳しい例を使用する場合は、

```
(request option 82 "circuit-id")
```

CLI のコマンド解析に制限があるため、この式を CLI に直接入力することはできません。複雑な式をテキスト ファイルに配置して入力し、そのファイルを属性定義内の "at" 記号 ( )@で参照する必要があります。たとえば、その式が cclookup.txt ファイルに置かれている場合、CLI コマンドは次のようになります。

```
nrcmd> dhcp set client-class-lookup-id=@cclookup.txt
```

ファイル内の式の構文には、単純な式の余分な要件(文字の間隔とエスケープ)はありません。また、シャープ記号(#)、ダブルスラッシュ(/)、セミコロン(;))、行末で終了するコメント行を含めることもできます。次の例を参考にしてください。

```
// Expression to set client-class based on remote-id
(if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
  "no-limit"
  "limit")


// Expression to calculate client-class based on remote-id
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
    "cm-client-class"
    "cpe-client-class")
  "<none>")
```

前の例の IPv6 バージョン (オプション番号を使用) は、次のとおりです。

```
// Expression to calculate client-class based on DOCSIS 3.0 cm-mac-address
(try
  (if (equal (request option 17 enterprise-id 4491 36)
```

```
          (or (request relay option 17 enterprise-id 4491 1026) "none"))
    "v6-cm-client-class"
    "v6-cpe-client-class")
  "<none>")
```

数値の代わりにオプション名を置き換えて、前の式を記述することもできます。

```
// Expression to calculate client-class based on DOCSIS 3.0 cm-mac-address
(try
  (if
    (equal
      (or
        (request option
        "vendor-opts" enterprise-id "dhcp6-cablelabs-config" "device-id")
         (substring (request option "client-linklayer-address") 3 8))
      (or
        (request relay option
          "vendor-opts" enterprise-id "dhcp6-cablelabs-config" "cm-mac-address")
        "none"))
    "v6-cm-client-class"
    "v6-cpe-client-class")
  "<none>")
```

例orの機能により、パケットがリレーされなかった場合、またはリレー エージェントがオプションを追加しなかった場合、サーバーはクライアントをCPEと見なし、ケーブルモデム(CM)ではないと見なします。

式の作成

DHCP式を使用すると、受信したDHCPパケットのデータに基づいて、取得、処理、および決定を行うことができます。着信パケットのクライアントクラスを決定するために使用し、オプション82制限サポート用の同等キーを作成することができます。パケットと個々のオプションから情報を取得する方法、パケット内の情報に基づく決定を可能にするさまざまな条件関数、およびクライアントクラスの名前またはキーを作成できるデータ合成機能を提供します。

例を記述する式ファイルに含める式一般的な制限シナリオは次のようになります。

```
// Begins the try function
(try
  (or
    (if (equal
          (request option "relay-agent-info" "remote-id")
          (request chaddr))
      "cm-client-class")
    (if (equal
          (substring (request option "dhcp-class-identifier") 0 6)
          "docsis")
      "docsis-cm-client-class")
    (if (equal
          (request option "user-class")
          "alternative-class")
      "alternative-cm-client-class"))
  "<none>")

// Ends the try function
```

式は関数をor使用し、3ifつの関数を評価します。より簡単な形式では、クライアントクラスを計算し、この式を cclookup.txt ファイルに含めることができます。

```
// Expression to calculate client-class based on remote-id
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
    "cm-client-class"
    "cpe-client-class")
  "<none>")
```

式を使用してサーバーのクライアント クラスルックアップ ID を設定するには、次のファイルを参照してください。

```
nrcmd> dhcp set client-class-lookup-id=@cclookup.txt
```

制限キーは、オプション 82 からremote-idサブオプションを取得し、できない場合は標準 MAC BLOB キーを使用して、制限キーを生成できます。ファイルに式を含め、ファイル内の制限 IDをcclimit.txt設定します。

```
// Expression to use remote-id or standard MAC
(try (request option "relay-agent-info" "remote-id") 00:d0:ba:d3:bd:3b)
```

式の構文

式は、関数とリテラルだけで構成されます。その構文は、Lispの構文に似ています。それは同じ規則の多くに従い、可能であればLisp関数名を使用します。基本のシンタックスは次のとおりです。

```
(function argument-0 ... argument-n)
```

より便利な例は次のとおりです。

```
(try
  (if (equal (request option "relay-agent-info" "remote-id") (request chaddr))
    "cm-client-class"
    "cpe-client-class")
  "<none>")
```

この例では、Relay エージェント情報オプション (オプション 82) のremote-idサブオプションをパケット内の MAC アドレスと比較し、それらが同じ場合は "cm-client-class" を返し、異なる場合は "cpe-client-class" を返します。(式がデータを評価できない場合、try関数は "<none>" 値を返式が失敗する可能性 （7 ページ）します。目的は、デバイスがケーブル モデムであるかどうかを判断すること (リモートIDが MAC アドレスと等しいと考えられます) を確認し、その場合は、デバイスを顧客宅内の機器や PC とは別のクライアント クラスに配置します。関数とリテラルの両方が式であることに注意してください。前の例では、関数を式として示しています。リテラルについては、「」を式のリテラル （6 ページ）参照してください。

式のデータタイプ

式でサポートされるデータ型は次のとおりです。

- Blob- カウントされた一連のバイト数、推奨される最大長は 1 KBの。

- String- 数え切られた一連の NVT ASCII 文字は 0 バイトで終わらず、推奨される最大長は 1 KBのです。
- Signed integer：32 ビット符号付き整数。
- Unsigned integer：32 ビットの符号なし整数。

IP アドレスデータ型はありません。IPv4 アドレスは 4 バイトの BLOB で、IPv6 アドレスは 16 バイトの BLOB です。すべての数字はネットワークバイト順です。データタイプの変換 （7 ページ） を参照してください。

式のリテラル

式機能には、次のようなさまざまなリテラルが含まれています。

- Signed 32 ビットに収まる必要がある標準integersの数値。
- Unsigned 32 ビットに収まる符号なしのintegers正規数。
- Blobs：コロン区切りの 16 進バイト。たとえば、01:02:03:04:05:06 は、バイト 1 から 6 までの 6 バイトの BLOB です。これは"01:02:03:04:05:06" (17 バイトの文字列) とは異なります。文字列は、BLOB のテキスト表現によって BLOB に関連付けられています。たとえば、式(to-blob "01:02:03")は BLOB 01:02:03 を返します。01 は整数に変わるので、1 バイトの BLOB のリテラル表現を作成できないことに注意してください。1 を含む 1 バイトの BLOB を(byte 1)取得するには、01 の BLOB を返すように使用できます。または、(substring(to-blob1)3式1)を使用することもできます。3 は、4 バイト整数の 4 バイト目 (00:00:00:01) を抽出するオフセットを示し、1 は抽出されたバイト数で、結果は "01" です。
- String：二重引用符で囲まれた文字。たとえば、"example.com" は文字列で、"01:02:03:04:05:05"と入力します。リテラル文字列に引用符を入れるには、次の例に示す円記号 (\) を使用してエスケープします。

```
"this has one \"quote"
```

整数リテラル(符号付きおよび符号なし)は、10の底にあると見なされます。0から始まる場合は 8 進数とみなされます。0x で始まる場合は、16 進数と見なされます。リテラルの例を次に示します。

- "hello world" は文字列リテラル (および完全に有効な式) です。
- 1 は符号なし整数リテラルです (完全に有効な式でもあります)。この値には 4 バイトが含まれ、最初の 3 バイトは 0 で、最後のバイトは最下位ビットに 1 を含みます。
- 01:02:03 は、3 バイト、01、02、および 03 を含む BLOB リテラルです。
- –10 は、10 進数 -10 の 2 の補数表現を持つ 4 バイトを含む符号付き整数リテラルです。

式の戻り型の値

例外が少ない場合は、式のポイントは値を返す点です。クライアントクラスを決定するように構成された式は、 DHCP サーバー プロパティクライアント クラス検索 IDで構成されます。この式が評価されると、DHCP サーバーは、クライアントクラスの名前または文字列を含む文字列"<none>"を返すことを DHCP サーバーが想定します。

すべての関数は値を返します。値のデータ型は、引数のデータ型によって異なります。式によっては、特定のデータ型の引数しか受け付けなれないものがあります。例えば：

```
(+ argument0  argument1)
```

ほとんどの場合、特定の引数に特定のデータ型を必要とする関数は、取得した引数を適切なデータ型に変換しようとします。たとえば(+"1"、2)文字列リテラル "1" を数値 1 に変換できたため、3 を返します。ただし、「1」(+ "one" 2)は正常に数値に変換されないため、エラーが発生します。一般に、式エバリュエーターは、データ型変換の決定を行う際に、可能な限り正しいことを行おうとします。

式が失敗する可能性

式を構成する関数の中には、データ型や値に対して正しく動作するものもありますが、多くの関数は正しく動作しません。前のセクションでは、+この関数は文字列リテラル "one" を有効な数値に変換しなかったため、その関数の評価に失敗しました。関数が評価に失敗すると、その呼び出し関数も失敗し、式全体が失敗するまで失敗します。式の評価が失敗した場合、関係する式によって結果が異なります。場合によっては、パケットがドロップされる可能性がありますが、警告メッセージを生成する場合もあります。

(try 式の失敗式) 関数を使用して、評価が失敗するのを防ぐことができます。関数tryは式を評価し、成功した場合は関数の値が式の値になります。評価が失敗した場合(何らかの理由で)、関数の値は失敗式の値になります。関数自体が失敗するtry唯一の状況は、失敗式の評価が失敗した場合です。したがって、どの式をエラー式として定義するか注意する必要があります。文字列リテラルは安全な賭けです。したがって、関数を使用してクライアントクラスルックアップ IDの評価をtry保護することをお勧めします。前に引用した例は、これがどのように機能するかを示しています。

```
(try
  (if (equal (request option "relay-agent-info" "remote-id")
             (request chaddr))
    "cm-client-class"
    "cpe-client-class")
  "<none>")
```

この場合、関数ifの評価が失敗した場合、クライアントクラスルックアップID式の値は"<none>"になります。もちろん、代わりにクライアントクラスの名前だったかもしれません。

データタイプの変換

関数が特定のデータ型の引数を必要とする場合、そのデータ型に値を変換しようとします。このエラーが発生する場合が多いため、関数全体が失敗することがあります。データ型変換は、to-string 、 to-blob、to-sintおよびto-uint関数によっても実行されます。関数が特定のデータ型の引数を必要とするたびに、外部から利用できる関数の内部バージョンを呼び出します。

また、as-string、 as-blob、as-sint、およびas-uint変換関数もあり、値のデータは目的のデータ型として再ラベル付けされます。次の表に、両方の関数セットの変換マトリックスが表示されます。

to-stringとas-stringの違いに注意してください。たとえば、BLOB形式のデータがあるとします。このデータは、要求パケットからデータを取得する関数評価 (**要求 get オプション**) の結果、または blob データをサブ文字列で処理した結果として使用される場合があります。このデータが BLOB 型であっても、実際に ASCII 文字列データを表す場合は、文字列として使用することをお勧めします。変換には as-string と to-string の 2 つの選択肢があります。どちらを選ぶべきでしょうか? データが ASCII バイトで構成されており、そのデータ型を文字列としてそのまま認識し、基本的にリセットする場合は、as-string関数を使用します。つまり、BLOB のバイトを文字列として使用します。BLOB 00:01 は文字列に変換できず、試してみるとエラーがスローされます。blob 68:65:6c:6c:6f は、as-string で文字列に変換して "hello" を生成します。一方、ASCIIデータである可能性もない可能性もある一連のバイトがある場合で、データをBLOBの文字列形式で表すには、 to-string を使用する必要があります。たとえば、to-string は最初が 0 次が 1 から成る 2 バイトの BLOB を文字列 "00:01" に変換します。

表 *1:* データ型変換行列

機能	文字列	BLOB	符号付き整数	符号なし整数
as-blob	失敗することはできません。ASCII文字にBLOB バイトとして再ラベルを付けます。	—	失敗することはできません。は、整数の4 バイトから 4 バイトのBLOBを生成します。	失敗しません。4 バイトの整数 から 4 バイトの BLOB を生成します。
as-sint	通常は役に立ちません。は、1 バイト、2 バイト、3バイト、または 4 バイトの文字列を BLOB に変換し、それを符号付き整数にパックします。	通常は役に立ちません。は 1 バイト、2バイト、3バイト、または 4 バイトの BLOB のみを変換します。	—	失敗することはできません。より大きな符号なし整数が正符号付き整数に収まる場合は、負の符号付き整数に変換されます。
as-string	—	文字列バイト (印刷可能な文字の場合) として再ラベル付けする	4 バイトの BLOB に変換し、それを BLOB として処理します (いくつかの特殊な整数を除いて失敗します)。	4 バイトの BLOB に変換し、BLOB として処理します(いくつかの特殊な整数を除いて失敗します)。
as-uint	通常は役に立ちません。1 バイト、2 バイト、3バイト、または 4 バイトの文字列を blob に変換し、次に符号付き整数に変換します。	通常は有用ではありません。1、2、3、4 バイトの BLOB のみを変換します。	失敗することはできません。は符号なし整数に変換され、負の符号付き整数は大きな符号なし整数になります。	—

機能	文字列	BLOB	符号付き整数	符号なし整数
to-blob	"01:02:03" の形式である必要があります。	—	失敗することはできません。は、整数の 4 バイトから 4 バイトの BLOB を生成します。	失敗しません。4 バイトの整数 から 4 バイトの BLOB を生成します。
to-sint	nまたは–nの形式でなければなりません。	1 バイト、2 バイト、3 バイト、または 4 バイトの BLOB のみ。	—	大きすぎて符号付き整数に収まらない場合にのみ変換します。
to-string	—	失敗しません	失敗しません	失敗しません
to-uint	形式n である必要があります。	1、2、3、4 バイトの BLOB のみ。	非負のみ。	—

式の関数

以下のセクションでは、式関数をリストします。式はかっこで囲む必要があります。

+、-、*、/、%

構文：

(+ arg1 ... argn)

(– arg1 ... argn)

(* arg1 ... argn)

(/ arg1 ... argn)

(% arg1 arg2)

説明：

符号付き整数または式の算術演算は、符号付き整数に変換できます。符号付き整数に変換できない (かつ null でない) 引数は、エラーを返します。null に評価される引数は無視されます (ただし、– および / の最初の引数は null に評価できません)。これらの関数は常に符号付き整数を返します (オーバーフローとアンダーフローは現在捕捉されないことに注意してください)。

- +引数を合計します。引数がない場合、結果は 0 になります。
- –単一の引数の値を否定するか、または複数の引数の場合は、残りの値を最初の引数から連続して減算します。たとえば、(– 3 4 5)は -6 になります。
- *引数の値の積を取ります。引数がない場合、結果は 1 になります。
- /連続して最初の引数を他のすべての引数で除算します。例えば、(/ 100 4 5)は 5 になります。最初の引数以外の引数が 0 の場合は、エラーが返されます。

- %は、最初の引数の結果の残りを 2 番目の引数で除算した剰余を決定する剰余算術演算子です。例えば、(% 12 7) は 5 (12 / 7 = 1 * 7 + 5) となります。

例：

(+ 1 2 3 4) は 10 を返します

(- 10 5 2) は 3 を返します

(* 3 4 5) は 60 を返します

(/ 20 2 5) は 2 を返します

(/ 20 0) はエラーを返します

(% 12 7) は 5 (12/7=1*7+5) を返します

---

and

構文：

(and arg1 ... argn)

説明：

引数を左から右の順に評価します。引数が null と評価された場合、引数の評価を停止し、null を返します。それ以外の場合は、最後の引数argnの値を返します。

例：

(and "hello" "world") は "world" を返します

(and (request option 82 1) (request option 82 2)) は、オプション 82 サブオプション 1 とサブオプション 2 の両方が要求に存在する場合は、オプション 82 サブオプション 2 を返し、それ以外の場合は null を返します。

---

as-blob

構文：

(as-blob expr)

説明：

exprを BLOB として扱います。exprが文字列に評価された場合、その文字列を構成するバイトは返される BLOB のバイトになります。exprが BLOB に評価される場合、その BLOB は変更されずに返されます。exprがいずれかの種類の整数に評価された場合、整数のバイトを含む 4 バイトの BLOB が返されます。

例：

(as-blob "hello world") は、blob の 68:65:6c:6c:6c:6f:20:77:6f:72:6c:64 を返します

as-sint

構文：

(as-sint expr)

説明：

exprを符号付き整数として扱います。exprが 4 バイト以下の文字列または BLOB に評価された場合、関数はそれらのバイトから構築された符号付き整数を返します (4 バイトより長い場合はエラーを返します)。exprが符号付き整数に評価された場合、値は変更されずに返されます。符号なし整数の場合、同じビット値を持つ符号付き整数を返します。

例：

(as-sint ff:ff:ff:ff) は -1 を返します

(as-sint 2147483648) はエラーを返します

as-string

構文：

(as-string expr)

説明：

exprを文字列として扱います。exprが文字列に評価された場合、その文字列を返します。exprが BLOB に評価された場合、出力できない ASCII 値でない限り、BLOB 内のバイトから構築された文字列を返します。exprが整数に評価された場合、その値は単一文字の ASCII 値であると見なされ、それがエラーを返す印字出来ない文字列でない限り、その 1 文字から成る文字列が返されます。

例：

(as-string 97) は "a" を返します

(as-string 68:65:6c:6c:6f:20:77:6f:72:6c:64) は "hello world" を返します

(as-string 0) はエラーを返します

as-uint

構文：

(as-uint expr)

説明：

exprを整数として扱います。exprが 4 バイト以下の文字列または BLOB に評価された場合、それらのバイトから構築された符号なし整数を返します。4 バイトより長い場合は、エラーを返します。結果が符号なし整数の場合は、引数をそのまま返します。符号付き整数の場合、同じビット値を持つ符号なし整数を返します。

例：

(as-uint-2147483648) は、符号なし整数 2147483648を返します

(as-uint-1) は、符号なし整数 4294967295を返します

(as-uintff:ff:ff:ff) は、符号なし整数 4294967295を返します

---

ash

構文：

(ash expr shift)

(lshift expr shift)

説明：

shift 量によってビットがシフトされた整数または BLOB を返します。exprは、整数、BLOB、または文字列に評価できます。exprが文字列に評価された場合、この関数は文字列を符号付き整数に変換しようとします。両方とも失敗した場合は、エラーを返します。shiftは、符号付き整数に変換可能なものに評価する必要があります。shiftが正の値の場合、シフトは左になります。負の値を指定すると、シフトは右になります。exprの結果が符号付き整数の場合、右シフトは符号拡張を伴います。exprの結果が符号なし整数またはBLOBになる場合、右シフトは最上位ビットで 0 ビットシフトします。

例：

(ash00:01:001) は、ブロブ 00:02:00 を返します

(lshift00:01:00-1) は、ブロブ 00:00:80 を返します

(ash11) は、符号なし整数 2 を返します

---

bit

構文：

(bit-and arg1 arg2)

(bit-andc1 arg1 arg2)

(bit-andc2 arg1 arg2)

(bit-eqv arg1 arg2)

(bit-or arg1 arg2)

(bit-orc1 arg1 arg2)

(bit-orc2 arg1 arg2)

(bit-xor arg1 arg2)

説明：

2 つの引数に対するビット単位のブール演算の結果を返します。結果のデータ型は、両方の引数がいずれかの種類の整数を返す場合は符号付き整数になります。arg1引数とarg2引数は、2 つの整数、2 つの同じ長さの BLOB、または 1 つの整数と 1 つの長さ 4 の blob に評価される必要があります。いずれかの引数が文字列に評価された場合、関数は文字列を符号付き整数に変換し、失敗した場合はBLOBに変換しようとします。この変換後、結果は上記の条件に一致する必要があります。これらの条件が満たされない場合は、エラーを返します。

演算c1とc2、それぞれ第 1 および第 2 引数が、演算の前に補完されることを示します。

例：

(bit-and 00:20 00:ff) は、00:20 を返します

(bit-or 00:20 00:ff) は、00:ff を返します

(bit-xor 00:20 00:ff) は、00:df を返します

(bit-andc1 00:20 00:ff) は、00:df を返します

---

bit-not

構文：

(bit-not expr)

説明：

exprのビットごとの補数である値を返します。式は、型またはBLOB のいずれかの整数に評価する必要があります。文字列に評価される場合、関数は文字列を符号付き整数に変換しようとします。それが失敗した場合は、BLOBに対して、失敗した場合はエラーを返します。結果のデータ型は、exprとその後の変換を評価した結果と同じです。

例：

(bit-not ff:ff) は、00:00を返します

(bit-not 1) は 4294967295を返します

(bit-not "hello world") は、エラーを返します

---

byte

構文：

(byte arg1)

説明：

1 バイトの BLOB の作成を容易にします。データ型に応じて、この BLOB を返します。

- sint,uint—整数の下位バイトを返します。
- blob—BLOB の最後のバイトを返します。
- string—文字列の最後のバイトを返します。

例：

(byte 150) は、96 の BLOBを返します

(byte 0x96) は、96 の BLOBを返します

---

comment

構文：

(comment comment expr1 ... exprn)

説明：

最初の引数は評価されず、引数が 1 つしかない場合は null を返します。引数が複数ある場合は、引数 expr1からexprnを評価し、 exprnの値を返します。

例：

(comment "this is a comment that won’t get lost" (request option 82 1))

---

concat

構文：

(concat arg1 ... argn)

説明：

引数の値を文字列または BLOB に連結します (null 引数は無視)。最初の引数 (arg1) は、文字列または BLOB に評価する必要があります。評価が整数の場合、関数はそれを BLOB に変換します。arg1のデータ型 (任意の変換後) は、結果のデータ型を決定します。この関数は、後続のすべての引数を結果のデータ型に変換し、この変換が失敗した場合はエラーを返します。

例：

(concat "hello" "world") は、"helloworld" を返します

(concat -1 "world") はエラーを返します

(concat -1 00:01:02) は、blob の ff:ff:ff:ff:00:01:02 を返します

---

datatype

構文：

(datatype expr)

説明：

式の結果のデータ型を返します(expr)式がエラーなしで評価された場合、データ型を文字列として返します。

- "未設定" (内部、null と見なされます)
- "null"
- "uint"
- "sint"
- "string"
- "blob"

---

dotimes

構文：

(dotimes (var count-expr [result-expr]) exp1 ... expn)

説明：

最初にゼロに設定された単一のローカル整数変数varを持つ環境を作成し、expnを通じてexp1を評価します。次に、varを 1 ずつインクリメントし、count-exprより小さい場合は、expnを通じてexp1を再度評価します。varがcount-expr以上の場合、関数はresult-exprを評価し、dotimes全体の結果として返します。result-exprがない場合、関数は null を返します。

varはローカル変数を定義し、アルファベットの名前でなければなりません。count-expr は、整数に評価するか、1 に変換可能でなければなりません。exp1からexpnは、任意のデータ型に評価できる式です。result-exprはオプションであり、表示される場合は任意のデータ型に評価できます。関数がcount-exprを評価すると、varはバインドされず、count-exprに出現することはできません。あるいは、varはresult-exprの評価にバインドされ、count-exprの値を持ちます。result-exprを省略すると、この関数は null を返します。

（注） exp1のvarの値をexpnを通じて変更する場合は、無限ループを簡単に作成できるので注意してください (例を参照)。

例：

(let (x y) (setq x 01:02:03) (dotimes (i (length x)) (setq y (concat (substring x i 1) y)))) は 03:02:01 を返します

(dotimes (i 10) (setq i 1)) は無限ループとなります!

---

environmentdictionary

構文：

(environmentdictionary {get | put val | delete} attr)

説明：

DHCP 拡張環境ディクショナリ属性値を取得、配置、または削除します。valは属性の値で、attrは属性名です。両方とも、初期データ型に関係なく文字列に変換されます。初期環境ディクショナリは変更できませんが、シャドウすることができます (最初のディクショナリ内の何かを再定義することはできますが、それを削除すると、元の初期値が残っています)。get キーワードは "get" のオプションではありません。また、これらの例では、初期環境ディクショナリが使用され、式を「設定」するために使用できる一方で、この関数は、すべての環境ディクショナリを介して拡張機能と通信するためにも使用できます。要求と応答のペア。

例：

nrcmd> dhcp setinitial-environment-dictionary=first=one, second=2

(environmentdictionary get "first") は "one" を返します

(environmentdictionary get "second") は "2" を返します(文字列の 2 です)

(environmentdictionary put "two" "second") は "second" を返します

(environmentdictionary delete "first") は null を返します

---

equal, equali

構文：

(equal expr1 expr2 expr3)

(equali expr1 expr2 expr3)

説明：

このequal関数は、expr1とexpr2を評価した結果の等価性を評価します。等しい場合は、次の値が返されます。

1. 指定されている場合はexpr3の値を返します。
2. expr2の値 (および可能な文字列変換後のデータ型) は、expr2が null でない限り、それ以外の値です。
3. 文字列 "*T*" (null を返すと、比較が失敗したことを誤って示すため)。

expr1とexpr2が等しくない場合、この関数は null を返します。

引数には任意のデータ型を指定できます。異なる場合、関数はこれらを比較する前に文字列に変換します (これは失敗できません)。文字列変換は、同等の(to-string..)を使用して無効にすることができます。したがって、blob 61:62 は "ab" 文字列と等しくありません。また、1 バイトの BLOB 01 はリテラル整数 1 と等しくないことに注意してください (どちらも文字列に変換され、"01" と "1" の文字列は等しくありません)。

関数equaliはequal関数と同じですが、比較が文字列に対する比較の場合(文字列引数が使用されたか、引数が文字列に変換されたため)、大文字と小文字を区別しない比較が使用されます。

例：

(equal (request option "dhcp-class-identifier") "docsis") は、オプションの値 dhcp-class-identifier が "docsis" と同じ文字列である場合、文字列 "docsis" を返します

(equali "abc" "ABC") は "ABC" を返します

(equal "abc" "def") は null を返します

(equal "ab" (as-string 61:62)) "this is true") は "this is true" を返します

(equal "ab" 61:62 "this is not true") は null を返します

(equal 01:02:03 01:02:03) は 01:02:03 を返します

(equal (as-blob "ab") 61:62) は 61:62 を返します

(equal 1 (to-blob 1)) は null を返します

(equal (null) (request option 20)) は、パケットにオプション 20 がない場合、"*T*" を返します

---

error

構文：

(error)

説明：

error 関数の評価の上に try 関数がない限り、式の評価全体が失敗する"回復なし"エラーを返します。

---

if

構文：

(if cond [then else])

説明：

if-then-elseの意味で条件式を評価します。condが null 以外の値に評価された場合、then引数を評価した結果を返します。それ以外の場合は else引数を評価した結果を返します。then および else は、オプションの引数です。then引数とelse引数を省略すると、cond引数を評価した結果が返されます。else引数を省略し、condが null に評価された場合、この関数は null を返します。3 つの引数のいずれにもデータ型に制限はありません。

例：

```
(if (equali
```

```
    (substring (request option "dhcp-class-identifier") 0 6)
    "docsis"
(request option 82 1))
```

いずれの場合も、dhcp クラス識別子の最初の 6 文字が "docsis" である場合は、オプション 82 のサブオプション 1 を返します。それ以外の場合は null を返します。

ip-string

構文：

(ip-string blob)

説明：

4 バイトの IP アドレスBLOBの文字列表現を"a.b.c.d"の形式で返します。単一の引数BLOBは、BLOB に評価するか、または 1 つに変換可能である必要があります。BLOB が 4 バイトを超える場合、この関数は最初の4つのバイトのみを使用してIPアドレス文字列を作成します。BLOB のバイト数が少ない場合、関数は IP アドレス文字列を作成するときに右端のバイトをゼロと見なします。

例：

(ip-string 01:02:03:04) は "1.2.3.4" を返します

(ip-string -1) は "255.255.255.255" を返します

(ip-string (as-blob "hello world")) は "104.101.108.108" を返します

ip6-string

構文：

(ip6-string blob)

説明：

16 バイトの IPv6 アドレスBLOBの文字列表現を"a:b:c:d:e:f:g:h"の形式で返します。引数 blob は、blob に評価されるか、blob に変換可能である必要があります。BLOB が 16 バイトを超える場合、この関数は最初の16バイトのみを使用してIPv6アドレス文字列を作成します。BLOB のバイト数が少ない場合、関数は IPv6 文字列を作成するときに右端のバイトをゼロと見なします。

（注） IPv6 アドレスを文字列として表す方法は複数あるため、IPv6 アドレスの文字列形式を比較すると、結果が不整合になる可能性があります。IPv6 アドレスを BLOB 値と比較するのが最善であり、アドレスの表現にあいまいさはありません。文字列形式の IPv6 アドレスが既にある場合は、to-ip6を参照してください。

例：

(ip6-string (as-blob "hello world")) は "6865:6c6c:6f20:776f:726c:6400::" を返します

---

is-string

構文：

(is-string expr)

説明：

exprの評価結果が文字列であるか、文字列として使用できる場合は、exprの値を返します。つまり、as-string がエラーを返さない場合、is-string は expr の値を返します。

例：

(is-string 01:02:03:04) は null を返します

(is-string "hello world") は "hello world" を返します

(is-string 68:65:6c:6c:6f:20:77:6f:72:6c:64) は blob 68:65:6c:6c:6f:20:77:6f:72:6c:64 を返します

---

length

構文：

(length expr)

説明：

値が expr の値の長さ（バイト単位）である整数値を返します。引数exprは任意のデータ型に評価できます。整数は常に長さ 4 を持ちます。文字列の長さには、文字列を終了する可能性のあるゼロバイトは含まれません。

例：

(length 1) は 4 を返します

(length 01:02:03) は 3 を返します

(length "hello world") は 11 を返します

---

let

構文：

(let (var1..varn) expr1 ..expn)

説明：

null 値に初期化されるローカル変数var1からvarnを持つ環境を作成します (setq関数を使用して他の値を指定できます)。ローカル変数が null に初期化されると、関数は式expr1からexprnを順番に評価します。その後、最後の式exprnの値を返します。この関数の利点は、値を一度計算し、ローカル変数に代入してから、その値を再計算せずに他の式で再利用できることです。変数では大文字と小文字が区別されます。

例：

```
(let (x)
  (setq x (substring (request option "dhcp-class-identifier") 0 6))
  (or (if (equali x "docsis") "client-class-1")
      (if (equali x "something else") "client-class-2")))
```

log

構文：

(log severity expr)

説明：

exprを文字列に変換した結果をログに記録します。severityとexprは文字列でなければならず、評価が 1 でない場合は 1 に変換されます。severityは null にすることもできます。文字列の場合、次のいずれかの値を持つ必要があります。

- "debug"
- "severity" (severityが null の場合のデフォルト)
- "info"
- "warning"
- "error"

（注） ログ記録はサーバーリソースを大量に消費するため、式に入れるlog関数評価の数を制限します。「error」の重大度がログに記録された場合でも、ログ関数はエラーを返しません。これは、ログメッセージにエラーを示すタグのみを付けます。関数評価の一部としてエラーを返すerror関数を参照してください。

mask-blob

構文：

(mask-blob mask-size length)

説明：

lengthの blob 長さで、BLOB の上位ビットから始まる長さmask-sizeのマスクを含む BLOB を返します。mask-sizeは、整数に評価される式、または変換可能な式です。同様にlengthはmask-size

より小さくすることはできませんが、0 または正の値を指定する必要があるという点以外は、固定の制限はありません。mask-sizeが 0 より小さい場合は、BLOB の右端から計算されたマスク長を示します。

例：

(mask-blob 1 4) は 80:00:00:00 を生成します

(mask-blob 4 2) は f0:00 を生成します

(mask-blob 31 4) は ff:ff:ff:fe を生成します

(mask-blob -1 4) は 00:00:00:01 を生成します

---

mask-int

構文：

(mask-int mask-size)

説明：

整数の上位ビットから始まる mask-size のマスクを含む整数を返します。mask-size は整数に評価されるか、または整数に変換される式である必要があります。mask-sizeが 0 より小さい場合は、整数の右端から計算されたマスク長を示します。

例：

(mask-int 1) は 0x80000000 を生成します

(mask-int 4) は 0xf0000000 を生成します

(mask-int 31) は 0xfffffffe を生成します

(mask-int -1) は 0x00000001 を生成します

---

not

構文：

(not expr)

説明：

exprは、文字列、BLOB、または整数に評価できる式です。その評価の結果が NULL でない場合は、null が返されます。その評価の結果が null の場合、null 以外の値が返されます。exprの値が null の場合に返される null 以外の値は、2 回の呼び出しで同じままであるとは保証されません。

例：

(not "hello world") は null を返します

null

構文：

(null [expr1 ... exprn])

説明：

null を返し、その引数を評価しません。

or, pick-first-value

構文：

(or arg1... argn)

(pick-first-value arg1... argn)

説明：

引数を順番に評価します。引数の評価が null 以外の値を返す場合、その値が返されます。1 つの引数が null 以外の値を返した後、他の引数は評価されません。それ以外の場合は、最後の引数argnの値を返します。データ型は同じである必要はありません。

例：

```
(or
  (request option 82 1)
  (request option 82 2)
  01:02:03:04)
```

はオプション 82 のサブオプション 1 の値を返し、それが存在しない場合はサブオプション 2 の値を返し、存在しない場合は 01:02:03:04 を返します。

parse

構文：

(parse expr1 expr2)

説明：

expr2 で指定されたデータ型として解析された文字列 expr1 を解析した BLOB 結果を返します。expr1 が文字列でない場合は、文字列に変換されます。expr2 は、Cisco Prime Network Registrar でサポートされる AT_* data types（文字列またはその数値）のいずれかである必要があります（オプションの検証タイプ を参照してください）。

この機能は、Cisco Prime Network Registrar 11.0 で導入されました。

例：

(parse 1234 "AT_INT") は d2:04:00:00 を返します。

(parse "cisco.com" "AT_DNSNAME") は、05:63:69:73:63:67:03:63:6f:6d:00 を返します。

---

progn, return-last

構文：

(progn arg ... argn)

(return-last arg ... argn)

説明：

引数を順番に評価し、最後の引数argnの値を返します。

例：

```
(progn
  (log (null) "I was here")
  (request option 82 1))

(return-last
  (log (null) "I was here")
  (request option 82 1))
```

---

regex

構文：

(regex expr1 expr2 var1... varn)

(regex expr1 expr2)

説明：

指定した target-string (expr2) で正規表現パターン (expr1) と一致するサブ文字列を検索し、指定された変数var1、var2、varnに設定します。つまり、指定されたターゲット文字列 (expr2) で正規表現パターン (expr1) で一致する最初のサブ文字列は、var1 に設定され、2 番目のサブ文字列はvar2に設定されます。変数を指定するときは、let 関数の前に置く必要があります。この関数は変数なしで使用することもできますが、この場合、正規表現パターン (expr1) で最初に一致するサブ文字列を、指定されたターゲット文字列 (expr2) で返します。

正規表現パターンの一致は文字列に対してのみ機能するので、パターン (expr1) とターゲット文字列 (expr2) の両方とも文字列である必要があります。そうでない場合、以下の例で使用されるように as-string 関数を使用する必要があります。

例：

(regex "[H][a-z]+" "Hello World") は "Hello" を返します

```
(let (x y z)
  (regex "[H][a-z]+" "Hello Hi World" x y z))
```

は x="Hello"、y="Hi"、z=null を設定し、"Hello"を返します

必要に応じて、let 内の regex の後に追加の式を配置して、x と y を操作できます。

---

request

構文：

(request [get | get-blob] [relay [number]] packetfield)

説明：

DHCPv4 packetfield の有効な値は次のとおりです。

op (blob 1)

htype (blob 1)

hlen (blob 1)

hops (blob 1)

xid (uint)

secs (uint)

flags (uint)

ciaddr (blob 4)

yiaddr (blob 4)

siaddr (blob 4)

giaddr (blob 4)

chaddr (blob hlen)

sname (string)

file (string)

request packetfield関数は、request パケットから指定されたフィールドの値を返します。DHCP request パケットには、オプション領域のオプションと同様に名前付きフィールドが含まれます。この形式の要求関数は、requestパケットから特定の名前付きフィールドを取得するために使用されます。relayキーワードは、request option関数に記述されています。

RFC 2131 で定義されている packetfield の値は、上記のとおりです。要求できるpacketfieldの値がいくつかありますが、未加工のDHCPパケットでは正確にこれらの方法で表示されません。これらはパケットに現れるデータを取り、よく使用される方法で結合します。これらの説明では、想定されるパケットの内容は次のとおりです。

hlen = 1 htype = 6 chaddr = 01:02:03:04:05:06

macaddress-string(string) - MAC アドレスをhlen、 htype、 chaddr形式で返します (たとえば、"1,6,01:02:03:04:05:06")

macaddress-blob(blob) - hlen:htype:chaddr形式の MAC アドレスを返します (たとえば、01:06:01:02:03:04:05:06)

macaddress-clientid(blob) - Microsoft htypeのMAC アドレスから作成されたクライアント ID を返します。

DHCPv6 packetfieldの有効な値は次のとおりです。

msg-type (uint)

msg-type-name (string)

xid (uint)

relay-count (uint)

hop-count (uint)

link-address (blob 16)

peer-address (blob 16)

DHCPv6 のmsg-typeパケット フィールドは、現在のリレーまたはクライアント メッセージの種類を示し、値を持ちます。

1=SOLICIT、2=ADVERTISE、3=REQUEST、4=CONFIRM、5=RENEW、6=REBIND、8=RELEASE、9=-DECLINE、11=INFORMATION-REQUEST、12=RELAY-FORWARD

msg-type-nameパケットフィールドは、メッセージタイプ名の文字列を返します。SOLICIT のように、文字列の値は常に大文字です。

xidは 24 ビット クライアント トランザクション IDで、relay-countは要求内のリレー メッセージの数です。

DHCPv4 パケットから DHCPv6 パケット フィールドが要求されると、エラーが返されます。その逆も同様です。

例：

(request get ciaddr) は存在する場合は ciaddr を返し、それ以外の場合は null を返します

(request ciaddr) は次と同等です (request get ciaddr)

(request giaddr) は、0 以外の場合は giaddr を返し、それ以外の場合は null を返します。

---

request dump

構文：

(request dump)

説明：

現在の要求パケットをログファイルにダンプします。すべての式の評価がdumpキーワードをサポートしているわけではないため、未サポートの場合は無視されます。

---

request option

構文：

(request [get | get-blob] option-request)

ここで option-request は次のとおりです。

1. IPv6 -relay [n] 用のオプションのリレー メッセージ セレクタ
2. 1 つ以上のオプション句 (複数のオプションが IPv6 でのみサポートされています) - option name | id [vendor name | enterprise-id name | id] [instance n]
3. 0 個以上のサブオプション句が続く - name | id [vendor name | enterprise-id name | id] [instance n]
4. オプションの句が続く - [instance-count | count | index n]

説明：

パケットからオプションの値を返します。キーワードは次のとおりです。

- get- 省略した場合は省略可能。
- get-blob- オプションバイトに直接アクセスできるデータを BLOB として返します。
- relay—IPv6 パケットにのみ適用され、それ以外の場合はエラーを返します。クライアント オプションの代わりにリレー オプションを要求します。nは、クライアントに最も近い n 番目のリレー エージェントを示します。省略すると、0 (クライアントに最も近いリレー エージェント) が想定されます。
- option— オプション (およびサブオプション) は、整数または文字列に評価される id または name引数で指定します。これらのいずれかに評価されない場合、関数は変換を行わないため、エラーを返します。名前指定子の有効な文字列値は、拡張機能に使用されるものと同じです。
- enterprise-id- オプションまたはサブオプションの後で、指定された enterprise-id を持つオプションまたはサブオプションのインスタンスを選択します。エンタープライズ ID は、整数または文字列に評価する必要があるidまたはname引数として指定できます。
- vendor- オプションまたはサブオプションの後で、オプションのデータをデコードするためにベンダーのカスタム・オプション定義を使用することを要求します。DHCPv6 オプションには適用されません。指定されたベンダ文字列に定義が存在しない場合、エラーは発行されず、オプションの標準定義が使用されます (なしの場合は BLOB と見なされます)。
- instance- 直前のオプションまたはサブオプションのn番目のインスタンスを選択します。インスタンスは 0 から始まります。(インスタンスとインスタンスカウントは、単一のリクエスト関数で一緒に使用することはできません。
- instance-count- 前のオプションまたはサブオプションのインスタンス数を返し、通常は、そのすべてのインスタンスをループ処理するために使用されます。オプションまたはサブオプションが存在しない場合は 0 を返します。
- index- 複数の値 (つまり、アドレスの配列または整数値) を含むオプションでn番目の値を選択します。インデックスは 0 から始まります。たとえば、index 0は最初の値を返し、index 1は 2 番目の値を返します。
- count- 前のオプションの関連するデータ項目の数を返し、通常はindexキーワードと共に使用して、オプションまたはサブオプションのすべてのデータ値をループします。

サブタブ (サブオプション) 指定子に定義されている唯一の文字列値サブオプション名は、リレーエージェント情報オプション (82) 用であり、復号化された DHCP パケット データ項目セクションのDHCPv4 および BOOTP オプションの表にリストされています。

このrequest option関数は、要求されたオプションに応じて、データ型を持つ値を返します。これは、テーブル内のデータ型がrequest関数によって返されるデータ型にどのように対応するかを示しています。

表 *2 : request*関数によって返されるデータ型

オプションデータ型	返されるデータ型
blob	blob
IP アドレス	4 バイトの BLOB
string	string
8 ビットの符号なし整数	uint
16 ビットの符号なし整数	uint
32 ビットの符号なし整数	uint
integer	sint
バイト値ブール型	sint=1 が true の場合は true、false の場合は null

例：

(request option 82) は relay-agent-info オプションを BLOB として返します。

(request option 82 1) は circuit-id (1) サブオプションだけを返します。

(request option 82 "circuit-id") は、 (request option 82 1) と同等です

(request option "domain-name- servers") は domain-name-servers オプションから最初の IP アドレスを返します

(request option 6 index 0) は、(request option 6 count) と同等で、IP アドレスの数を返します。

(request get-blob option "dhcp-class-identifier") は、文字列ではなく、BLOB として値を返します

(request option "IA-NA" instance 2 option "IAADDR" instance 3) は、IA-NA オプションの 3 番目のインスタンス、および IA-NA オプションにカプセル化された IAADDR オプションの 4 番目のインスタンスを返します

(request get-blob option "vendor-opts" enterprise-id 1234) はenterprise-id 1234 のオプション データの BLOB を返します

(request option "vendor-opts" enterprise-id 1234 3) は、要求されたベンダーオプションデータからサブオプション 3 を返します

DHCPv6 オプション 16 ベンダー クラス (長さ区切りフィールドを含む):

DHCPv6 メッセージのデータ:

```
00:10:00:11:00:00:00:7b:00:04:01:02:03:04:00:05:68:65:6c:6c:6f
^   ^ ^   ^ ^         ^ ^               ^ ^                 ^
|   | |   | |         | +--- field 0 ---+ +-- field 1 -------+
|   | |   | |         |
|   | |   | +---------+ enterprise-id 123(10)
|   | +---+ length 17
+---+ Option 16 Vendor-Class
```

(request option 16 enterprise-id 123) -> タイプ: blob 値: '01:02:03:04'

(request option 16 enterprise-id 456) -> タイプ: 値の設定解除: 'null'

(request get-blob option 16 enterprise-id 123) -> タイプ: blob 値: '00:00:7b:00:04:01:02:03:04:00:05:68:65:6c:6c:6c:6f'

(request option 16 enterprise-id 123 index 0) -> タイプ: blob 値: '01:02:03:04'

(request option 16 enterprise-id 123 index 1) -> タイプ: blob 値: '68:65:6c:6c:6f'

（注） DHCPv6 Option 15、User-Classは、同じように動作します。

DHCPv6 Option 17 Vendor Opts (サブオプションが含まれています):

DHCPv6 メッセージ内のデータ :

```
00:11:00:12:00:00:01:c8:00:01:00:04:0a:0b:0c:0d:00:05:00:02:01:02
^   ^ ^   ^ ^         ^ ^                     ^ ^               ^
|   | |   | |         | +---- suboption 1 ----+ +- suboption 5 -+
|   | |   | |         |
|   | |   | +---------+ enterprise-id 456(10),1c8(16)
|   | +---+ length 18
+---+ Option 17 Vendor-Opts
```

(request option 17 enterprise-id 456) -> タイプ: blob 値: '00:00:01:c8:00:01:00:04:0a:0b:0c:0d:00:05:00:02:01:02'

(request option 17 enterprise-id 0x1c8) -> タイプ: blob 値: '00:00:c8:00:01:00:00:04:0a:0b:0c:0d:00:00:05:00:02:01:02'

(request option 17 enterprise-id 123) -> タイプ: 値の設定解除: 'null'

(request option 17 enterprise-id 456 index 0) -> タイプ: blob 値: '00:00:c8:00:01:00:00:04:0a:0b:0c:0d:00:00:05:00:02:01:02'

(request option 17 enterprise-id 456 1) -> タイプ: blob 値: '0a:0b:0c:0d'

(request option 17 enterprise-id 456 2) -> タイプ: 値の設定解除: 'null'

(request option 17 enterprise-id 456 5) -> タイプ: blob 値: '01:02'

requestdictionary

構文：

(requestdictionary{get | put val | delete} attr)

説明：

DHCP 拡張要求ディクショナリ属性値を取得、配置、または削除します。valは属性の値で、attrは属性名です。両方とも、初期データ型に関係なく文字列に変換されます。get キーワードは、"get" のオプションではありません。

---

response

構文：

(response [get | get-blob] [relay [number]] packetfield)

説明：

応答パケットから指定されたpackefieldの値を返します。説明と有効な値は、request packetfield関数の説明と同じです。

---

response dump

構文：

(response dump)

説明：

現在の応答パケットをログファイルにダンプします。すべての式の評価がdumpキーワードをサポートしているわけではないため、未サポートの場合は無視されます。

---

response option

構文：

(response [get | get-blob] option-request)

ここで option-request は次のとおりです。

1. IPv6 -relay [n] 用のオプションのリレー メッセージ セレクタ
2. 1 つ以上のオプション句 (複数のオプションが IPv6 でのみサポートされています) - option name | id [vendor name | enterprise-id name | id] [instance n]
3. 0 個以上のサブオプション句が続く - name | id [vendor name | enterprise-id name | id] [instance n]
4. オプションの句が続く - [instance-count | count | index n]

説明：

パケットからオプションの値を返します。キーワードは、request関数のキーワードと同じです。

---

responsedictionary

構文：

(responsedictionary{get | put val | delete} attr)

説明：

DHCP 拡張応答ディクショナリ属性値を取得、配置、または削除します。valは属性の値で、attrは属性名です。両方とも、初期データ型に関係なく文字列に変換されます。get キーワードは、"get" のオプションではありません。

---

search

構文：

(search arg1 arg2 fromend)

説明：

arg1のバイト シーケンスと完全に一致するバイトのサブシーケンスのためのarg2の値を構成するバイトを検索します。見つかった場合、サブシーケンスが開始するarg2の要素のインデックスを返します(fromend引数を "true" またはその他の任意の null 以外の値に設定しない限り)。それ以外の場合は null を返します。(arg1が null の場合は 0 を返し、arg2が null の場合は null を返します。この関数は、両方の引数に対して暗黙のas-blob変換を行います。したがって、文字列と BLOB の実際のバイト シーケンスを比較し、sints と uints は比較の目的で 4 バイトの BLOB になります。

null 以外の fromend引数は、一番右の一致するサブシーケンスの左端の要素の index を返します。

例：

(search "test" "this is a test") は 10 を返します

(search "test" "this test test test" "true") は 15 を返します

---

setq

構文：

(setqvar expr)

説明：

let関数内でのみ有効です。varは、外側のlet関数で定義されたvar1からvarnのローカル変数のいずれかでなければなりません。

例：

例についてはlet 関数を参照してください。

---

starts-with

構文：

(starts-with expr prefix-expr)

説明：

prefix-expr の値が expr の先頭と一致する場合、exprの値を返します。prefix-expr が expr より長い場合は null を返します。この関数は、prefix-expr が expr (文字列または BLOB) と同じデータ型に変換できない場合、または expr が整数に評価された場合にエラーを返します。

例：

(starts-with "abcdefghijklmnop" "abc") は "abcdefghijklmnop" を返します

(starts-with "abcdefgji" "bcd") は null を返します

(starts-with 01:02:03:04:05:06 01:02:03) は 01:02:03:04:05:06 を返します

(starts-with "abcd" (as-string 61:62)) は "abcd" を返します

(starts-with "abcd" 61:62) は null を返します

(starts-with "abcd" (to-string 61:62)) は null を返します

---

substring

構文：

(substring expr offset len)

説明：

オフセットから始まる式exprのlenバイトを返します。exprは文字列またはBLOB です。整数の場合は、BLOB に変換されます。結果は文字列または BLOB、またはいずれかの引数が null と評価される場合は null になります。条件：

- offsetが長さlenより大きい場合、結果は null になります。
- offset + lenはexprの終わりを超えるデータで、関数は残りのデータをexprで返します。
- offsetが 0 より小さい場合、オフセットはデータの末尾から取得されます (最後の文字は、最初の文字を参照する –0=0 なので、インデックス -1 です)。

- これはデータの先頭を越えてデータを参照し、オフセットはゼロと見なされます。

例：

(substring "abcdefg" 1 6) は "bcdefg" を返します

(substring 01:02:03:04:05:06 3 2) は 04:05 を返します

---

synthesize-host-name

構文：

(synthesize-host-name method namestem)

説明：

構成されたメソッド(指定されていない場合)または指定されたmethodとnamestemに基づいてホスト名を生成します。

method 引数の有効なメソッドは、DHCPv4 要求または DHCPv6 要求が処理されているかどうかによって異なります。DHCPv4 の場合、有効なメソッドは、defaultまたは v4-synthetic-name-generator の列挙値の 1 つ:address、client-id、またはhashed-client-idです。DHCPv6 の場合、有効なメソッドは、defaultまたは v6-synthetic-name-generatorの列挙値の 1 つ:duid、hashed-duid、cablelabs-device-id、またはcablelabs-cm-mac-addrです。これらの列挙メソッドの詳細については、DHCPv4 と DHCPv6 での合成名の生成を参照してください。

namestem引数は、DNS 更新構成のsynthetic-name-stem値を指定します(DNS 更新設定の作成を参照してください)。

例：

(synthesize-host-name) は "dhcp-rhfxxi5pkjp6o" を返します。

(synthesize-host-name "duid" "test") は "test-00030001010203040506" を返します

(synthesize-host-name "client-id" "test") は "test-00030001010203040506" を返します

---

to-blob

構文：

(to-blob expr)

説明：

式を BLOB に変換します。条件：

- exprは文字列に評価され、 "nn:nn:nn" 形式である必要があります。この関数は、文字列を BLOB に変換した結果である BLOB を返します。関数が文字列を BLOB に変換できない場合は、エラーを返します。
- exprは、その BLOB を返す、BLOB に評価されます。

- exprは整数に評価され、ネットワーク順で整数のバイトを表す4バイトのBLOBを返します。（データタイプの変換 （7 ページ） を参照）。

例：

(to-blob 1) は 00:00:00:01 を返します

(to-blob "01:02") は 01:02 を返します

(to-blob 02:03) は 02:03 を返します

---

to-ip、to-ip6

構文：

(to-ip expr)

(to-ip6 expr)

説明：

式を文字列、BLOB、または整数として IP アドレスに変換します。条件：

- 文字列は、IPv4 の場合はドット付き 10 進法 IP アドレス形式、または IPv6 の場合はコロン形式の形式でなければなりません。文字列を解析して IP アドレスに変換することによって決定された BLOB IP アドレスを返します。
- 結果は BLOB で、(to-ip ..) の最初の 4 バイトと (to-ip6 ..) の最初の 16 バイトを返します。blob が to-ip の場合は 4 バイト未満、または to-ip6 の場合は 16 バイト未満の場合、引数 BLOB のバイト数は 0 バイトで高次バイトに埋め込まれます。
- 結果は整数で、(いずれかのタイプの)整数をブロブに変換します。整数と BLOB はネットワークの順序で並べ替えるため、順序の変更は必要ありません。

---

to-lower

構文：

(to-lower expr)

説明：

文字列を受け取り、小文字の文字列を生成します。client-lookup-id 属性を使用して、クライアント指定子を計算して、(LDAP ではなく) CNRDB ローカル ストア内の client-entry を検索する場合、結果の文字列は小文字である必要があります。この関数を使用すると、client-lookup-id の結果を小文字の文字列に簡単に作成できます。client-lookup-id を使用して LDAP にアクセスする場合、この機能を使用する場合と使用しない場合があります。

---

to-sint

構文：

(to-sint expr)

説明：

式を符号付き整数に変換します。

exprが文字列に評価される場合、符号付き整数に変換できる形式である必要があります。条件：

- exprが 1 ~ 4 バイトの BLOB に評価される場合、関数はそれを符号付き整数として返します。
- exprが 4 バイトを超える長さの BLOB に評価される場合、エラーを返します。
- exprが符号なし整数に評価される場合、符号なし整数の値が最大の正符号付き整数より大きい場合を除き、同じ値の符号付き整数を返します。
- exprが符号付き整数に評価される場合、その値を返します。

例：

(to-sint "1") は 1 を返します

(to-sint -1) は -1 を返します

(to-sint 00:02) は 2 を返します

(to-sint "00:02") はエラーを返します

(to-sint "4294967295") は 2147483647 を返します

---

to-string

構文：

(to-string expr)

説明：

式を文字列に変換します。exprが文字列に評価された場合は、その文字列を返します。BLOBまたは整数の場合は、その印字可能な表記を返します。すべての値が印字可能な表記であるため、expr自体がエラーなしで評価された場合、エラーは返されません。

例：

(to-string "hello world") は "hello world" を返します

(to-string -1) は "-1" を返します

(to-string 02:04:06) は "02:04:06" を返します

---

to-uint

構文：

(to-uint expr)

説明：

式を符号なし整数に変換します。条件:

- exprが文字列に評価される場合、符号なし整数に変換できる形式である必要があります。
- exprが 1 ~ 4 バイトの BLOB に評価される場合、符号なし整数として返されます。
- exprが 4 バイトよりも長い blob に評価される場合、エラーを返します。
- exprが符号付き整数に評価される場合、符号付き整数の値が 0 未満でない限り、同じ値の符号なし整数を返します。
- exprが符号なし整数に評価される場合、関数はその値を返します。

例：

(to-uint "1") は 1 を返します

(to-uint 00:02) は 2 を返します

(to-uint "4294967295") は 4294967295を返します

(to-uint "00:02") はエラーを返します

(to-uint -1) はエラーを返します

---

translate

構文：

(translate expr search replace)

説明：

文字列またはBLOB のシーケンスに回避する式を引数として受け取り、search に表示されるさまざまな文字またはバイトをreplaceの対応する値 (同じ位置) に置き換えます。条件：

- exprが文字列またはBLOB である場合、値はそのまま残され、それ以外の場合は強制的に文字列になります。処理後にexprが文字列である場合、search と replace は文字列である必要があります。
- expr が BLOB である場合、search と replace の両方が BLOB である必要があります。
- replace が search より短い場合、replaceに対応するバイトまたは文字がない search 内のバイトまたは文字は出力からドロップされます。
- replace が表示されない場合、search のバイトまたは文字はすべて expr から削除されます。

例：

(translate "Hello apple and eve" "abcdef" "123456") は "H5llo 1ppl5 1n4 5v5" を返します

(translate "a&b$c%d" "%$&") は "abcd" を返します

try

構文：

(try expr failure-expr)

説明：

評価中にエラーが検出されなかった場合、exprを評価し、その評価の結果を返します。exprの評価中にエラーが発生した場合は、次の手順を実行します。

- failure-exprがあり、エラーなしで評価された場合、try関数の結果としてその評価の結果を返します。
- failure-expr があり、関数が failure-expr を評価中にエラーが発生した場合、 エラーを返します。
- failure-expr がない場合、try は null を返します。

例：

(try (try (expr) (complex-failure-expr)) "string-constant") は外側の try がエラーを返さないことを保証します（"string-constant" の評価は失敗できないため）

(try (error) 01:02:03) は常に 01:02:03 を返します

(try 1 01:02:03) は常に 1 を返します

(try (request option 82) "failure") は "failure"を返しません。（request option 82）は、パケットに option-82 がなく、エラーを返さない場合に null になるためです。

(try (request option "junk") "failure") は "junk" が有効な option-name ではないため、"failure" を返します。

unparse

構文：

(unparse expr1 expr2 [expr3])

説明：

expr2 で指定されたデータ型として BLOB expr1 を解析した結果の文字列を返します。expr3 で指定されたとおりに変更されることがあります。expr1 が BLOB でない場合は、BLOB に変換されます。expr2 は、Cisco Prime Network Registrar でサポートされる AT_* data types（文字列またはその数値）のいずれかである必要があります（オプションの検証タイプを参照してください）。 expr3 はオプションで、値は「none」、「alternate」、または「feature」で、動作は expr2 に依存します。たとえば、AT_BOOL タイプの場合、「feature」は「enabled」または「disabled」を返し、「alternate」は「on」または「off」を返し、「none」（または no expr3）は「true」または「false」のいずれかを返します。

この機能は、Cisco Prime Network Registrar 11.0 で導入されました。

例：

(unparse 00 "AT_BOOL" "feature") は disabled を返します。

(unparse 05:63:69:73:63:67:03:63:6f:6d:00 "AT_DNSNAME") は 「cisco.com」 を返します。

---

validate-host-name

構文：

(validate-host-name hostname)

説明：

hostname 文字列を受け取り、検証済みのhostnameを返します。これは、入力 hostname と同じか、次のように変更できます。

- ハイフンに割り当てられたスペースと下線付き文字。
- 無効なhostname文字を削除。有効な文字は a ～ z、A ～ Z、0 ～ 9、およびハイフンです。
- Null ラベルを削除（「..」が「.」に変更される）。
- hostname の各ラベルは 63 文字に切り捨てられます。

例：

(validate-host-name "a b c d e f") は "a-b-c-d-e-f" を返します

(validate-host-name "_a_b_c_d_e_f_") は "a-b-c-d-e-f" を返します

(validate-host-name "abcdef") は "abcdef" を返します

(validate-host-name "a&b*c#d@!e()f") は "abcdef" を返します

オプションに対して式を使用する

Cisco Prime Network Registrar 11.0 以降では、式を使用して、オプションに値を返すことができます（DHCPv4 および DHCPv6）。

オプションに式を使用する場合は、次の点に注意してください。

- オプションインスタンスには、固定値または式を指定できますが、両方使用することはできません（ただし、式は固定値を返すことは可能）。
- 式であるオプションインスタンスは、そのオプションがクライアント要求の応答に追加されるたびに評価されます。
- 式であるオプションインスタンスは、リースクエリ（unitary、bulk、active）では評価されません（返されません）。これは、式を評価するためのコンテキストが使用できないためです。

- オプションの式は、次のいずれかを返す必要があります。
  - Null 値：この場合、オプションは応答に追加されません。
  - <none> の値 （大文字と小文字を区別しない）：この場合、オプションは応答に追加されません。
  - BLOB 値：この場合、値はこのオプションとして返されます。これは完全なオプションデータである必要があります。ベンダーオプション（DHCPv4 オプション 125 や DHCPv6 オプション 17 など）の場合は、最初の 4 バイトに企業 ID を含める必要があります。
  - 文字列値：この場合、値はオプションの定義に基づいて解析され、解析された値が返されます。解析が失敗した場合、オプションは応答に追加されません。

  結果に関係なく式を評価した後、サーバーはオプションの他のインスタンスのポリシー階層の検索を続行しないことに注意してください。
- 式追跡設定がオプション式に適用されます。
- オプションは予測できない順序で応答に追加されるため、式であるオプションそして、応答ディクショナリの他のポイントの値として使用するオプションは、予測できない結果が生じる場合があるため、推奨しません。
- オプション値のラウンドロビンは、式であるオプションで使用されます。式の結果の値はラウンドロビンされます。
- NULL 値によって、オプションが応答に追加されないため、式であるオプションは、長さが 0 のオプション値を生成できません。

（注） DHCPv4 オプション、dhcp-lease-time (51)、dhcp-renewal-time (58) および dhcp-rebinding-time (59) は、式ではサポートされていません。これらは値で設定する必要があります。式で設定されている場合、DHCP サーバーはこのオプションを無視します。

CLI の場合、**ポリシー**のヘルプには、オプションインスタンスを式として設定する方法の詳細が含まれています。

式を使用して、サブスクライバーにリースされるIPアドレスを制限する

これらの例では、クライアントを制限する、制限しないもの、および構成制限を超えて、クライアントクラスの制限超過に割り当てる必要があるものを設定します。クライアントの3つのクラスのそれぞれに、それぞれスコープと選択タグがあります。これらの例では、次のCisco Primeネットワーク レジストラー設定環境を想定しています(これは実際の環境とは異なり、図のためだけに使用されます)。

- Client-classes—制限、制限なし、および制限超過。
- Scopes10.0.1.0 (プライマリ)、10.0.2.0、10.0.3.0(セカンダリ)、サブネットの名前。
- Selection tags—制限タグ、制限なしタグ、および制限超過タグ。スコープは、それらが表すアドレス プールの名前が付けられます。選択タグは、範囲に割り当てられ、10.0.1.0 は制限タグ、10.0.2.0 は無制限タグ、10.0.3.0 は制限を超えるタグを取得します。

関連項目

制限事例 1: DOCSIS ケーブル モデム （39 ページ）

制限事例 2: 拡張 DOCSIS ケーブル モデム （40 ページ）

制限事例 3: 非同期転送モードでの DSL （41 ページ）

制限事例 1: DOCSIS ケーブル モデム

テストは、デバイスが DOCSIS ケーブル モデムと見なされるかどうかを判断し、各ケーブル モデムの背後にあるカスタマーデバイスの数を制限することです。クライアントクラスの制限 ID は、リレー エージェント情報オプションのremote-idサブオプションに含まれるケーブル モデムの MAC アドレスです。

サーバー上のクライアント・クラス・ルックアップ ID属性の式は、次のとおりです。

```
// Expression to set client-class to no-limit or limit based on remote-id
(if (equal (request option "relay-agent-info" "remote-id")
           (request chaddr))
  "no-limit"
  "limit")
```

上記の式は、relay-agent-infoオプションのremote-idサブオプション (2) の内容がパケットのchaddrと同じである場合、クライアントクラスは制限なしであることを示しています。

制限クライアントクラスの制限id式は次のとおりです。

```
(request option "relay-agent-info" "remote-id")
```

この式は、次の手順で使用します。

ステップ 1 クライアント クラスを定義します。

ステップ 2 スコープ、範囲、およびタグを定義し、それらがプライマリまたはセカンダリの場合に定義します。各スコープのホスト範囲は、すべてのホスト番号が同じである場合よりも、誤読される可能性が低いことを確認します。

ステップ 3 制限数を定義します。これは、デフォルトのポリシーに入ることができます。リクエストに制限 ID が表示されない場合、カウントはチェックされません。

ステップ 4 次の目的で、式ファイル cclookup1.txt に式を追加します。

```
// Expression to set limitation count based on remote-id
(if (equal (request option "relay-agent-info" "remote-id")
```

```
          (request chaddr))
  "no-limit"
  "limit")
```

ステップ 5 サーバー レベルでクライアント クラスの検索ID属性を設定する場合は、式ファイルを参照してください。

ステップ 6 クライアントの制限 ID に対する別の式を cclimit1.txt ファイルに追加します。

```
// Expression to set limitation ID based on remote-id
(request option "relay-agent-info" "remote-id")
```

ステップ 7 クライアントクラスの制限 id属性を設定する際は、この式ファイルを参照してください。

ステップ 8 サーバーをリロードします。

以前に使用されていない構成に対してこれを行うと、最初の 2 つの DHCP クライアントに共通のremote-id オプション 82 サブオプション値が設定されます。同じ値を持つ 3 番目のクライアントは、クライアント クラスの制限超過に入ります。サブスクライバが制限なしクライアント クラスに持つことができるデバイスの数には制限はありません。MACアドレスがremote-idサブオプションの値と等しいデバイスは、制限の目的で無視され、制限 ID が設定されていない制限なしクライアント クラスに入ります。

制限事例 2: 拡張 DOCSIS ケーブル モデム

この例は、制限事例 1: DOCSIS ケーブル モデム （39 ページ）で説明されている例の拡張です。後者の例では、デフォルトポリシーに対して制限数が2つ定義されているため、すべてのケーブル モデムがクライアント デバイスを 2 つ超えるだけで済みます。この例では、制限タグ選択タグを使用するスコープとは異なる数のデバイスに IP アドレスを付与できるように、特定のケーブル モデムを設定しています。

この場合、クライアントクラスデータベースで、2 つ以上のアドレスを持つケーブルモデムを明示的に設定する必要があります。この場合、Cisco Prime Network レジストラーまたは LDAP データベースでケーブル モデムのクライアント エントリを検索できるように、サーバー全体でのクライアント クラス処理を有効にする必要があります。ケーブル モデムが見つからない場合、デバイスの数は2に制限されます。この検出では、ケーブルモデムに設定されたポリシーの制限数が使用されます。

この例では、5つのデバイスを許可する 5 つの追加ポリシーが必要です。

ステップ 1 サーバー全体でクライアントクラスの処理を有効にします。

ステップ 2 5つのデバイスの制限数を持つ 5 つのポリシーを作成します。

ステップ 3 前の例と同様に、式を使用して、制限クライアントクラスの制限 ID を設定します。制限 ID を cclimit2.txt ファイルに、ルックアップ ID を cclookup2.txt ファイルに入れます。

```
cclimit2.txt file:
// Expression to set limitation ID
(request option "relay-agent-info" "remote-id")

cclookup2.txt file:
```

```
// Expression to set client-class lookup ID
(concat "1,6," (to-string (request option "relay-agent-info" "remote-id")))
```

ステップ 4 適切な属性を設定する際には、これらのファイルを参照してください。

ステップ 5 いくつかのケーブル モデム クライアントを定義し、5つのポリシーを適用します。

ステップ 6 サーバーをリロードします。

制限事例 3: 非同期転送モードでの DSL

この例では、式を使用して、非同期転送モード(ATM)ルーティングブリッジカプセル化(RBE)を使用してサービス プロバイダへの加入者のデジタル加入者線 (DSL) アクセスを構成する方法を示します。サービス プロバイダは、DSL サブスクライバーを構成する ATM RBE を使用するようになっています。Cisco IOS Release 12.2(2)T よりルーテッド ブリッジ カプセル化機能の DHCP オプション 82 サポートされるようになり、サービス プロバイダは DHCP を使用して IP アドレスを割り当てられるようになったほか、オプション 82 を使用してセキュリティおよび IP アドレス割り当てポリシーを実装できるようになりました。

このシナリオでは、DSL サブスクライバは Cisco 7401ASR ルータの個々の ATM サブインターフェイスとして識別されます。各顧客はルータに独自のサブインターフェイスを持ち、各サブインターフェイスには独自の仮想チャネル識別子(VCI)と仮想パス識別子(VPI)があり、ATM スイッチを通過する ATM セルの次の宛先を識別します。7401ASR ルータは、Cisco 7206 ゲートウェイ ルータにルーティングします。

ステップ 1 IOS を使用して、ルータの DHCP サーバーとインターフェイスを設定します。これは典型的な IOS 設定です:

```
Router#ip dhcp-server 170.16.1.2
Router#interface Loopback0
Loopback0(config)#ip address 11.1.1.129 255.255.255.192
Loopback0(config)#exit
Router#interface ATM4/0
ATM4/0(config)#no ip address
ATM4/0(config)#exit
Router#interface ATM4/0.1 point-to-point
ATM4/0.1(config)#ip unnumbered Loopback0
ATM4/0.1(config)#ip helper-address 170.16.1.2
ATM4/0.1(config)#atm route-bridged ip
ATM4/0.1(config)#pvc 88/800
ATM4/0.1(config)#encapsulation aal5snap
ATM4/0.1(config)#exit
Router#interface Ethernet5/1
Ethernet5/1(config)#ip address 170.16.1.1 255.255.0.0
Ethernet5/1(config)#exit
Router#router eigrp 100
eigrp(config)#network 11.0.0.0
eigrp(config)#network 170.16.0.0
eigrp(config)#exit
```

ステップ 2 IOS で、システムが Cisco IOS DHCP サーバーに転送される BOOTREQUEST メッセージに DHCP オプション 82 データを挿入できるようにします。

```
Router#ip dhcp relay information option
```

ステップ 3 IOS で、オプション 82 remote-idサブオプション(2)を使用して DHCP サーバーに送信される DHCP リレー エージェントのループバック インターフェイスの IP アドレスを指定します。

```
Router#rbe nasip Loopback0
```

ステップ 4 Cisco Prime Network レジストラーで、サーバー全体でのクライアント クラスの処理を有効にします。

ステップ 5 1つのデバイスの制限数を持つ 1 つのポリシーを作成します。

ステップ 6 パケットを適切なクライアント クラスに配置します。すべてのパケットは、クライアントクラスの制限内にあるべきです。値limitのみを含むルックアップ・ファイルを作成し、クライアント・クラスのルックアップ ID を設定します。cclookup3.txt ファイルで次の操作を行います。

```
// Sets client-class to limit
"limit"
```

ステップ 7 式を使用して、制限されたパケットに正しい制限IDがあることを確認します。ファイルに式を入れ、そのファイルを参照して制限 ID を設定します。サブストリング関数は、オプション 82 サブオプション 2 (remote-id) データ・フィールドのバイト 10 から 12 を抽出することによって VPI/VCI を取得します。cclimit3.txt ファイルで次の手順を実行します。

```
// Sets limitation ID
(substring (request option 82 2) 9 3)
```

ステップ 8 サーバーをリロードします。

デバッグ式

式に問題がある場合は、サーバー起動時に DHCP ログ ファイルを調べます。すべての式は、関数の入れ子を明確にするような形で印刷され、意図を確認するのに役立ちます。特に、ログ ファイルに出力された式をコピーして、エディタに貼り付けることができます。各行の先頭から文字を削除すると、結果の式が正しく入力されます (読み取りや変更が非常に簡単になります)。関数と引数のequalデータ型変換に特に注意してください。引数が同じデータ型でない場合、to-string関数と同様のコードを使用して文字列に変換されます。

DHCP サーバーの式トレース レベル属性を使用して、式のさまざまなデバッグ レベルを設定できます。実行されたすべての式は、属性によって設定された次数までトレースされます。最高のトレース レベルは 10 です。レベルを少なくとも 2 に設定すると、失敗した式はレベル 10 で再試行されます。

式トレース・レベルのトレース・レベルは次のとおりです (数値を使用)。

- 0— トレースなし
- 1— 失敗、(tryによって保護されたものを含む)
- 2— 失敗の再試行の合計 (再試行のトレース レベル = 6)

- 3— 関数呼び出しと戻り値
- 4— 関数の引数が評価される
- 5— 関数の引数を印刷する
- 6—データ型変換(すべて)

構成に問題がある式をトレースするために、式構成トレース・レベル属性も存在し、1から10までの任意のレベルに設定できます。レベルを2以上に設定すると、構成されていない式はレベル6に設定して再試行されます。番号付けのギャップは、将来のレベルの追加に対応するためです。式構成トレース・レベルのトレース・レベルは次のとおりです (number 値を使用)。

- 0— 追加のトレースなし
- 1— 追加のトレースなし
- 2— 失敗の再試行（デフォルト）
- 3— 関数定義
- 4— 関数の引数
- 5— 変数の検索とリテラルの詳細
- 6—すべて