# 手册:脚本编写

**适用RouterOS版本: v3, v4及以上**

## 脚本语言手册

这个手册对RouterOS内置的强大脚本语言进行介绍。
脚本主机提供了一种方法来自动执行一些路由器维护任务，通过执行与某些事件相关的用户定义的脚本。
脚本可以存放在脚本库中，也可以直接写入控制台。触发脚本执行的事件包括，但是并不局限系统计划System Scheduler、traffic monitor工具 netwatch工具产生的事件。

### 行结构

RouterOS脚本被分成多行命令行，命令行一条接一条地执行，直到脚本结束或则发生运行时错误。

#### 命令行

RouterOS 控制台使用以下的命令语法。

[prefix] [path] command [uparam] [param=[value]] .. [param=[value]]

• [prefix]-":" 或表明命令是ICE（接口配置环境）或者路径的"/"，可能需要或者不需要。
• [path] – 期望目录等级的相对路径，可能需要或者不需要。
• command –一条命令在特定的目录等级才可用。
• [uparam] –如果命令需要未命名参数，参数必须被指定。
• [params] – 命名参数和对应值的序列。

命令行结束通过通过 ";"表示，有时也通过换行(*NEWLINE*.)表示,命令行结束时换行(*NEWLINE*.)不是必须的。
单条命令在(), [] 或者{}中,不需要任何命令结束符，命令的结束是通过整个脚本内容来判断的。

```
:if ( true ) do={ :put "lala" }
```

每一条包含在另一条里的命令行，以"[ ]"(方括号)开始和结束。

```
:put [/ip route get [find gateway=1.1.1.1]];
```

请注意上面的命令行包含了三条命令行：

• :put
• /ip route get
• find gateway=1.1.1.1

命令行可以由多行物理行构成，通过以下的连接规则。

CONFIDENTIAL - amos

### 物理行

一个物理行是被行尾结束符 (EOL)终止的字符序列，任何标准平台的行尾结束符都可以使用。

- **unix** – ASCII LF;
- **windows** – ASCII CR LF;
- **mac** – ASCII CR;

标准C风格的换行字符也可以被使用。

### 注释

以#开始，在物理行结束。空格或者任何其他的符号都不允许在#之前出现。如果#出现在字符串中，就不会被当作注释，注释被语法忽略。

```
# this is a comment
  # bad comment
:global a; # bad comment

:global myStr "lala # this is not a comment"
```

### 行连接

两行或者多个物理行可以通过反斜杠(\)连接成为一个逻辑行，通过反斜杠结束的行不支持注释，反斜杠无法延续注释。反斜杠无法延续一个符号，字符串常量除外。当反斜杠在字符串常量之外的其他地方会被认为是非法的。

```
:if ($a = true \
           and $b=false) do={ :put “$a $b”; }
:if ($a = true \            # bad comment
           and $b=false) do={ :put “$a $b”; }
# comment \
       continued – invalid    (syntax error)
```

### 符号间的空格

空格可以用来分隔符号。当两个符号间的连接被理解为一个不同的符号时，空格就是必不可少的。例如：

```
{
     :local a true; :local b false;
# whitespace is not required
     :put (a&&b);
# whitespace is required
     :put (a and b);
}
```

下面的情况空格是不允许的

• between '<parameter>='
• between 'from=' 'to=' 'step=' 'in=' 'do=' 'else='

例如:

```
#incorrect:
:for i from = 1 to = 2 do = { :put $i }
#correct syntax:
:for i from=1 to=2 do={ :put $i }
:for i from= 1 to= 2 do={ :put $i }

#incorrect
/ip route add gateway = 3.3.3.3
#correct
/ip route add gateway=3.3.3.3
```

**范围**

变量只能在脚本的特定区域使用，这些区域就被称作scope(范围) ，范围决定了变量的可见度。这有两种类型的范围，全局和局部，一个变量在一个语句块内声明只能在该语句块内访问.

**全局范围**

全部范围或者根范围是脚本的默认范围，它是被自动创建的并且无法关闭。

**局部范围**

用户定义可以它自己的组来阻止特定变量的访问，这些范围被称作局部范围。每一个局部范围都是包含在花括号"{ }"内。

```
{
    :local a 3;
    {
        :local b 4;
        :put ($a+$b);
    }
#line below will generate error
    :put ($a+$b);
}
```

上面代码中的变量 b 是局部范围，在被花括号包含后无法被访问。

**请注意:** 控制台的每一行输入都被当作局部范围对待。

举个例子，定义好的局部变量在下一个命令行是不可见的，如果引用将会产生语法错误。

```
[admin@MikroTik] > :local myVar a;
[admin@MikroTik] > :put $myVar
syntax error (line 1 column 7)
```

**Warning警告：** 不要在局部范围内定义全局变量。

请注意，即使变量可以被定义为全局，变量也只能在局部范围内访问。

```
{
    :local a 3;
    {
        :global b 4;
    }
    :put ($a+$b);
}
```

上面的代码将会产生错误。

## 关键字

下面的字是关键字不能被用作变量和函数名：

and or not in

## 分隔符

下面的符号在语法中起分隔符的作：

() [] {} : ; $ /

## 数据类型

RouterOS 脚本语言有以下的数据类型：

| 类型 | 描述 |
|---|---|
| **number** | - 64位带符号整数，可能有十六进制输出； |
| **boolean** | - 值为 true或者false; |
| **string** | -字符序列; |
| **IP** | - IP 地址; |
| **internal ID** | - '*' 为前缀的十六进制值，每一个目录项都被分配了这样一个独一无二的值。 |
| **time** | -日期和时间值; |
| **array** | - 数值的序列被组织在一个数组里; |
| **nil** | - 在变量没有赋值时的默认变量类型; |

CDNA1-amo

### 转义序列常量

下面的转义序列可以用于在字符串内定义特殊的字符。

| | |
|---|---|
| \" | 插入双引号 |
| \\ | 插入反斜杠 |
| **\n** | 插入新的一行 |
| **\r** | 插入回车 |
| **\t** | 插入水平制表符 |
| **\$** | 输出 $ 字符. 否则 $ 用于引用变量. |
| **\?** | 输出 ? 字符. 否则 ? 是用于在控制台中输出 "help" |
| _ | 空格 |
| **\a** | BEL (0x07) |
| **\b** | 退格 (0x08) |
| **\f** | 换页 (0xFF) |
| **\v** | 插入垂直制表符 |
| **\xx** | 从16进制数值输出字符，十六进制数值应该用大写字母对应的。 |

```
:put "\48\45\4C\4C\4F\r\nThis\r\nis\r\na\r\ntest";
```

输出结果如下：

HELLO
This
is
a
test

## 运算符

### 算术运算符

常见的算术运算符，在RouterOS的脚本语言中都支持。

| Opearator | Description | Example |
|---|---|---|
| "+" | binary addition | :put (3+4); |
| "-" | binary subtraction | :put (1-6); |
| "*" | binary multiplication | :put (4*5); |
| "/" | binary division | :put (10 / 2); :put ((10)/2) |
| "-" | unary negation | { :local a 1; :put (-a); } |

**请注意：** 为了让除运算正常工作，你必须使用括号或者空格将操作对象分开，以免将操作对象误当作IP地址。

CDNAT - amos

### 关系运算符

| Opearator | Description | Example |
|---|---|---|
| "<" | 小于 | :put (3<4); |
| ">" | 大于 | :put (3>4); |
| "=" | 等于 | :put (2=2); |
| "<=" | 小于或等于 | |
| ">=" | 大于或等于 | |
| "!=" | 不等于 | |

### 逻辑运算符

| Opearator | Description | Example |
|---|---|---|
| **“!”， “not”** | logical NOT | :put (!true); |
| **“&&”，“and”** | logical AND | :put (true&&true) |
| **“||”， “or”** | logical OR | :put (true||false); |
| **“in”** | | :put (1.1.1.1/32 in 1.0.0.0/8); |

### 位运算符

位运算符对number和ip address数据类型起作用。

| Opearator | Description | Example |
|---|---|---|
| “~” | 比特位倒置 | put ， (~0.0.0.0) |
| “|” | 比特位或：执行逻辑的或运算。在每一对对应的比特位，如果连个比特位至少有一个值为1，则结果为“1”否则结果为“0”。 | |
| “^” | 比特位抑或：如果一对比特位的值不相等结果为1，比特位的值相等结果为0。 | |
| “&” | 逻辑和：如果比特对的值均为1结果为1，否则结果为0。 | |
| “<<” | 左移给定数量的比特位。 | |
| “>>” | 右移给定数量的比特位。 | |

### 连接运算符

CDNAT-amos

| Opearator | Description | Example |
|---|---|---|
| “.” | 连接两个字符串 | :put (“concatenate” . “ “ . “string”); |
| “,” | 连接两个数组或者增加新元素到数组 | :put ({1;2;3} , 5 ); |

不通过连接运算符添加变量值到字符串也是可能的。

```
:global myVar "world";

:put ("Hello " . $myVar);
# next line does the same as above
:put "Hello $myVar";
```

通过在字符串中使用 $[]和 $() 来添加变量值到字符串内是可能的。

```
:local a 5;
:local b 6;
:put " 5x6 = $($a * $b)";
```

```
:put " We have $[ :len [/ip route find] ] routes";
```

### 其他的运算符

| Opearator | Description | Example |
|---|---|---|
| “[]” | 命令替换，可以包含单个命令行 | :put [ :len "my test string"; ]; |
| “()” | 子表达式或者分组运算符 | :put ( "value is " . (4+5)); |
| “$” | 替换运算符 | :global a 5; :put $a; |
| “~” | 二进制操作符用来匹配扩展的正则表达式的值 | Print all routes which gateway ends with 202<br>/ip route print where gateway~"^[0-9<br>\\.]*202" |

## 变量

脚本语言有两种类型的变量:

- 全局 - 用global关键字定义，可以被当前用户创建的所有脚本访问。
- 局部 - 用local关键字定义，只能在当前定义的范围内才能被访问。

**请注意:** 从6.2版本开始可以有未定义的变量，当变量是未定义的，语法解析器会试着寻早变量集。例如通过DHCP lease脚本或者是Hotspot on-login脚本。除了RouterOS内置的变量每一个变量在使用前都必须通过local或者global关键字声明。未定义的变量将会被标记为未定义，并且会导致编译错误。

```
# following code will result in compilation error, because myVar is used without declaration
:set myVar "my value";
:put $myVar
```

错误代码

```
:local myVar;
:set myVar "my value";
```

```
:put $myVar;
```

例外是当使用变量集的时候。

```
/system script
add name=myLeaseScript policy=\
    ftp,reboot,read,write,policy,test,winbox,password,sniff,sensitive,api \
    source=":log info \$leaseActIP\r\
    \n:log info \$leaseActMAC\r\
    \n:log info \$leaseServerName\r\
    \n:log info \$leaseBound"

/ip dhcp-server set    myServer lease-script=myLeaseScript
```

变量命名的有效字符是字母和数字，如果变量名字包含了任何其他的字符，那么变量名应该放在双引号内，例如：

```
#valid variable name
:local myVar;
#invalid variable name
:local my-var;
#valid because double quoted
:global "my-var";
```

如果变量定义时没有赋值，变量类型会被设置成nil，否则数据类型会被脚本引擎自动判断。有时候数据类型的转换是需要的，这可以通过数据转换命令实现，例如：

```
#convert string to array
:local myStr "1,2,3,4,5";
:put [:typeof $myStr];
:local myArr [:toarray $myStr];
:put [:typeof $myArr]
```

变量名是大小写敏感的

```
:local myVar "hello"
# following line will generate error, because variable myVAr is not defined
:put $myVAr
# correct code
:put $myVar
```

不带值的set命令会让变量处于未定义状态（从environment中移去，v6.2新增）

```
#remove variable from environment
:global myVar "myValue"
:set myVar;
```

## 命令

每一个全局的命令应该以":" 符号开始，否则的话就会被视为变量。

| Command | Syntax | Description | Example |
|---|---|---|---|
| / | | 到根目录 | |
| .. | | 回退一个目录等级 | |
| ? | | 列出可用的目录命令和简单的描述 | |
| global | :global <var> [<value>] | 定义全局变量 | :global myVar "something"; :put $myVar; |
| local | :local <var> [<value>] | 定义局部变量 | { :local myLocalVar "I am local"; :put $myVar; } |
| beep | :beep <freq> <length> | 内置扬声器产生蜂鸣 | |
| delay | :delay <time> | 在给定的一段时间内不进行任何操作 | |
| put | :put <expression> | 输出参数到控制台 | |
| len | :len <expression> | 返回字符串长度或者数组元素计数值 | :put [:len "length=8"]; |
| typeof | :typeof <var> | 返回变量的数据类型 | :put [:typeof 4]; |
| pick | :pick <var> <start>[<end>] | 返回数组元素或者子字符串的范围，如果结束位置不明确，将会从数组中返回一个元素。 | :put [:pick "abcde" 1 3] |
| log | :log <topic> <message> | 将消息写入系统日志，可用的主题有 "debug, error, info and warning" | :log info "Hello from script"; |
| time | :time <expression> | 返回执行命令所需的时间间隔 | :put [:time {:for i from=1 to=10 do={ :delay 100ms }}]; |
| set | :set <var> [<value>] | 赋值给已声明的变量 | :global a; :set a true; |
| find | :find <arg> <arg> <start> | 返回子字符串或者数组元素的位置。 | :put [:find "abc" "a" -1]; |
| environment | :environment print <start> | 输出被初始化的变量信息 | :global myVar true; :environment print; |
| terminal | | 控制台相关命令 | |
| error | :error <output> | 产生控制台错误，停止脚本运行。 | |
| parse | :parse <expression> | 解析字符串并返回已解析的命令，可以被当作函数使用。 | :global myFunc [:parse ":put hello!"]; $myFunc; |
| resolve | :resolve <arg> | 返回给出DNS对应的IP地址 | :put [:resolve "www.mikrotik.com"]; |
| toarray | :toarray <var> | 转换变量类型为数组 | |
| tobool | :tobool <var> | 转换变量类型为布尔 | |
| toid | :toid <var> | 转换变量类型为 internal ID | |
| toip | :toip <var> | 转换变量类型IP 地址 | |
| toip6 | :toip6 <var> | 转换变量类型IPv6 地址 | |
| tonum | :tonum <var> | 转换变量类型为整数 | |

CDNA-amos

| | | |
|---|---|---|
| **tostr** | :tostr <var> | 转换变量类型为字符串c |
| **totime** | :totime <var> | 转换变量类型为time |

### 目录下具体命令

以下命令在大多数子目录下是可用的。

| Command | Syntax | Description |
|---|---|---|
| **add** | add <param>=<value>..<param>=<value> | 增加新条目 |
| **remove** | remove <id> | 移除选中条目 |
| **enable** | enable <id> | 启用选中条目 |
| **disable** | disable <id> | 禁用选中条目 |
| **set** | set <id> <param>=<value>..<param>=<value> | 改变选中条目的参数，同时可以指定多个参数。<br>Parameter can be unset by specifying '!' before parameter.<br>Example:<br>/ip firewall filter add chain=blah action=accept protocol=tcp port=123 nth=4,2<br>print<br>set 0 !port chain=blah2 !nth protocol=udp |
| **get** | get <id> <param>=<value> | 获取选定的条目的参数值 |
| **print** | print <param><param>=[<value>] | 输出目录条目，输出结果取决于选定的参数 |
| **export** | export [file=<value>] | 从当前目录以及其子目录导出配置，如果file参数被指定，输出将会被写入扩展名为'.rsc',的文件，否则输出将会显示在控制台，被导出的命令可以通过import命令导入。 |
| **edit** | edit <id> <param> | 在文本编辑器中编辑选中的条目属性 |
| **find** | find <expression> | 通过表达式找到条目 |

### 导入

导入命在根目录下可用，用于导入配置信息。

### 输出参数

输出命令可用的几个参数。

CDNA-amos

| Parameter | Description | Example |
|---|---|---|
| **append** | | |
| **as-value** | print output as array of parameters and its values | :put [/ip address print as-value] |
| **brief** | 输出简单描述 | |
| **detail** | 输出详细说明，可读性没有简单描述好，但是可以产看全部参数 | |
| **count-only** | 只输出条目计数 | |
| **file** | 将输出内容写入文件 | |
| **follow** | 输出所有当前条目并且追踪新条目，直到按下ctrl-c组合键，在查看日志条目时非常有用 | /log print follow |
| **follow-only** | 追踪和输出新条目直到按下ctrl-c组合键，，在查看日志条目时非常有用 | /log print follow-only |
| **from** | 从指定条目输出参数 | /user print from=admin |
| **interval** | 在选定的时间间隔内连续显示输出 | /interface print interval=2 |
| **terse** | 以紧凑和机器友好格式显示详细信息 | |
| **value-list** | 多个值一行输出 | |
| **without-paging** | 如果输出不符合控制台屏幕，输出就不停止,输出所有信息在一块 | |
| **where** | 表达式后跟where，可以用于过滤或者是匹配条目 | /ip route print where interface="ether1" |

一次可以使用多个参数，例如，/ip route print count-only interval=1 where interface="ether1"

## 循环和条件语句

| Command | Syntax | Description |
|---|---|---|
| **do..while** | :do { <commands> } while=( <conditions> ); :while ( <conditions> ) do={ <commands> }; | execute commands until given condition is met. |
| **for** | :for <var> from=<int> to=<int> step=<int> do={ <commands> } | execute commands over a given number of iterations |
| **foreach** | :foreach <var> in=<array> do={ <commands> }; | execute commands for each elements in list |

| Command | Syntax | Description |
|---|---|---|
| **if** | :if(<condition>) do={<commands>} else={<commands>} <expression> | If a given condition is true then execute commands in the do block, otherwise execute commands in the else block if specified. |

例如;

```
{
    :local myBool true;
    :if ($myBool = false) do={ :put "value is false" } else={ :put "value is true" }
}
```

CDNAT-amos

## 函数

脚本语言不允许直接创建函数，但是你可以把:parse命令当作工作区使用。从v6.2版本开始，新增的语法让定义函数和传递参数都变得更加简单。通过return命令返回函数值也是可能的。看下面的例子

```
#define function and run it
:global myFunc do={:put "hello from function"}
$myFunc

output:
hello from function

#pass arguments to the function
:global myFunc do={:put "arg a=$a"; :put "arg '1'=$1"}
$myFunc a="this is arg a value"    "this is arg1 value"

output:
arg a=this is arg a value
arg '1'=this is arg1 value
```

请注意传递参数的方式有两种。

•通过具体的名字传递传递参数  pass arg with specific name (如上例中的"a")

• 不通过具体的参数名传递值，在这种情况参数"1", "2" .. "n"  被使用。

```
.
你甚至可以从脚本environment复制脚本，并且把它当作函数使用。

#add script
  /system script add name=myScript source=":put \"Hello $myVar !\""

:global myFunc [:parse [/system script get myScript source]]
$myFunc myVar=world

output:
Hello world !
```

**警告:如果函数包含了与传递参数名字相同的已定义全局变量，那么已定义的全局变量将会被忽略，这么做是为了老版本的脚本兼容。这种特点在未来的版本中将会改变，请避免使用和全局变量名相同的参数名。例如：**

```
:global my2 "123"

:global myFunc do={ :global my2; :put $my2; :set my2 "lala"; :put $my2 }
$myFunc my2=1234
:put "global value $my2"
```

输出结果为:

```
1234
lala
```

global value 123

### 捕捉运行时错误

从v6.2开始，脚本可以捕捉运行时错误。例如：[code]:reslove[/code] 命令失败，将会抛出错误并且终止脚本。

```
[admin@MikroTik] > { :put [:resolve www.example.com]; :put "lala";}
failure: dns name does not exist
```

现在我们可以捕捉这个错误，并且通过调用我们脚本来处理。

```
:do {
        :put [:resolve www.example.com];
} on-error={ :put "resolver failed"};
:put "lala"

output:

resolver failed
lala
```

## 脚本库

**子目录等级:** /system script

包含所有的用户已创建的脚本，脚本可以以多种方式执行。

- **on event** – 脚本可以通过事件触发自动执行。
- **by another script** – 在脚本内运行脚本是允许的。
- **manually** –从控制台或者winbox运行执行命令。

| Property | | Description |
|---|---|---|
| **name** | (*string*; Default: *"Script[num]"*) | name of the script |
| **policy** | (*string*; Default: ) | list of applicable policies:<br>• **api** - api permissions<br>• **ftp** - can log on remotely via ftp and send and retrieve files from the router<br>• **local** - can log on locally via console<br>• **password** - change passwords<br>• **policy** - manage user policies, add and remove user<br>• **read** - can retrieve the configuration<br>• **reboot** - can reboot the router<br>• **sensitive** - see passwords and other sensitive information<br>• **sniff** - can run sniffer, torch etc<br>• **ssh** - can log on remotely via secure shell<br>• **telnet** - can log on remotely via telnet<br>• **test** - can run ping, traceroute, bandwidth test<br>• **web** - can log on remotely via http<br>• **winbox** - winbox permissions<br>• **write** - can retrieve and change the configuration |
| **source** | (*string*;) | Script source code |

CDNA - S

仅可读的状态属性

| Property | Description |
|---|---|
| **last-started** (*date*) | Date and time when the script was last invoked. |
| **owner** (*string*) | User who created the script |
| **run-count** (*integer*) | Counter that counts how many times script has been executed |

目录下具体命令

| Command | Description |
|---|---|
| **run** (*run [id\|name]*) | Execute specified script by ID or name |

## 运行环境

**子目录等级:**

- /system script environment
- /environment

包含所有用户的已定义变量和对应值。

```
[admin@MikroTik] > :global example;
[admin@MikroTik] > :set example 123
[admin@MikroTik] > /environment print
"example"=123
```

仅可读的状态属性：

| Property | Description |
|---|---|
| **name** (*string*) | Variable name |
| **user** (*string*) | User who defined variable |
| **value** () | Value assigned to variable |

## 工作

**子目录等级:** /system script job

包含所所有正在运行的脚本的列表。

仅可读的状态属性

| Property | Description |
|---|---|
| **owner** (*string*) | User who is running script |
| **policy** (*array*) | List of all policies applied to script |
| **started** (*date*) | Local date and time when script was started |

CDNA - mos

# 文章来源和参与者：

**Manual:Scripting** *Source*: http://wiki.mikrotik.com/index.php?oldid=25645 *参与者*s: Burek, Janisk, Marisb, Normis, Proofreader

**翻译：amos 邮箱1057905655@qq.com**

**寄语：将mikrotik官方的一些文档翻译出来，方便大家使用，由于自身水平不足，翻译不对的地方还望大家可以指出，可以直接发邮箱交流，也可以到bbs.mikrotik.com.cn论坛留言。大家有什么mikrotik官方文档需要翻译的也可以找我，我会酌情翻译的。**

CDNAT-amos