版本 S1.2

MikroTik Script 教程

该手册是提供对 MikroTik RouterOS 嵌入式脚本介绍，主机脚本提供了自动维护路由器任务的功能，通过借助用户自定义发生事件脚本。对于 RouterOS 的脚本操作，需要读者有一定的编程知识，而且对 RouterOS 各个功能熟悉和掌握。

内容如有更新,恕不通知。

第一章 RouterOS 脚本基本操作

在 RouterOS 中一个脚本配置构成由控制命令和表达式**(ICE - internal console expression 内部控制台表达式)**。

RouterOS 操作命令，例如：`/ip firewall filter add chain=forward protocol=gre action=drop` 这个是描述在防火墙中过滤 gre 协议的操作，即通过**“/”**来达到命令执行的目的。在脚本 **ICE** 表达式前缀需要用 **“：”**并能在任何目录路径下操作。

一个事件（event）用来触发脚本执行，在 RouterOS 下包括：System Scheduler, Traffic Monitoring Tool, Netwatch Tool 。

功能包需求: ***system***
操作路径: ***/system script***

注： RouterOS2.9 本版与 RouterOS 3.0 的脚本有一定的区别：

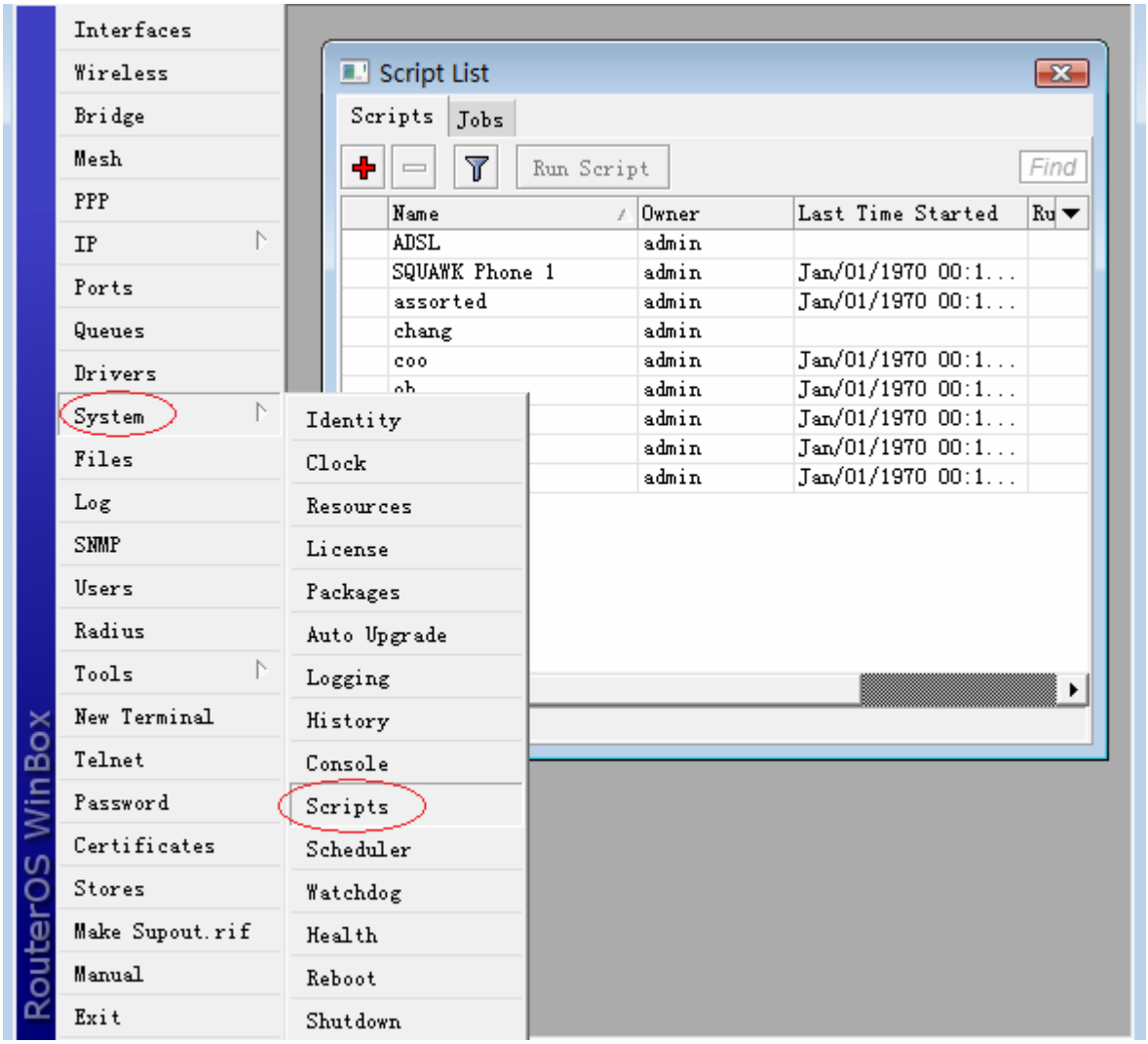
1、RouterOS3.0 字符参数后需要使用引号注明
如 **comment=”test”；name=”pppoe-out1”；:set i $”tx-current”**

2、RouterOS3.0 的变量定义不支持“中横杠”的定义
如 :**global test-address** 这样定义在 3.0 和 4.0 中是非法的

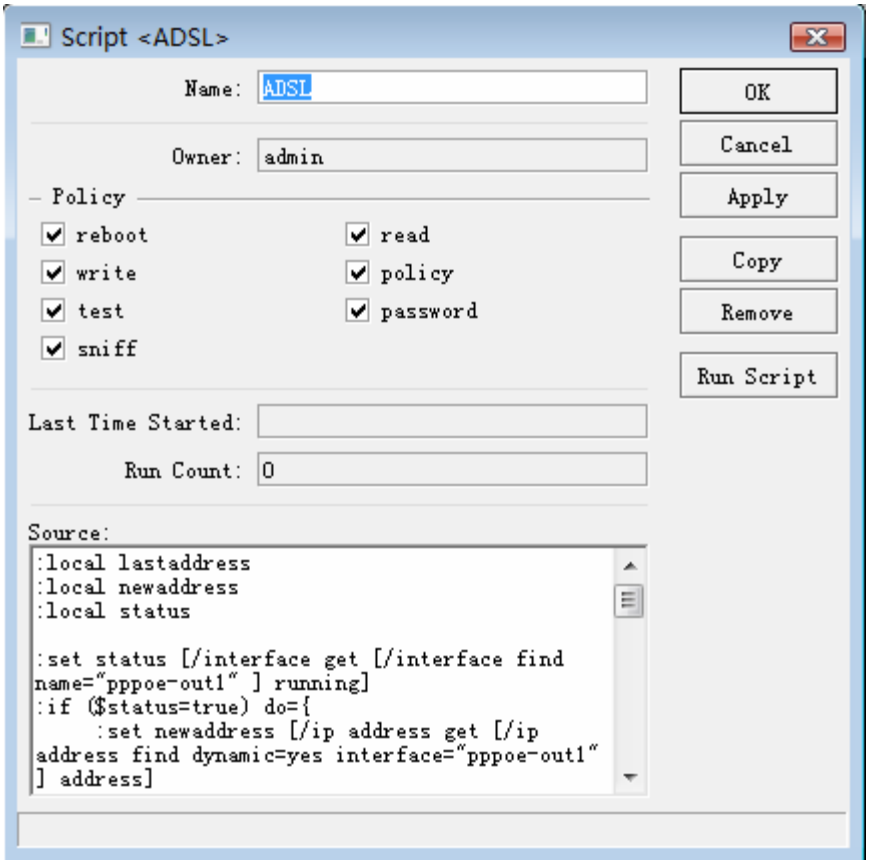
脚本在 RouterOS 中的调用

操作路径：/system script

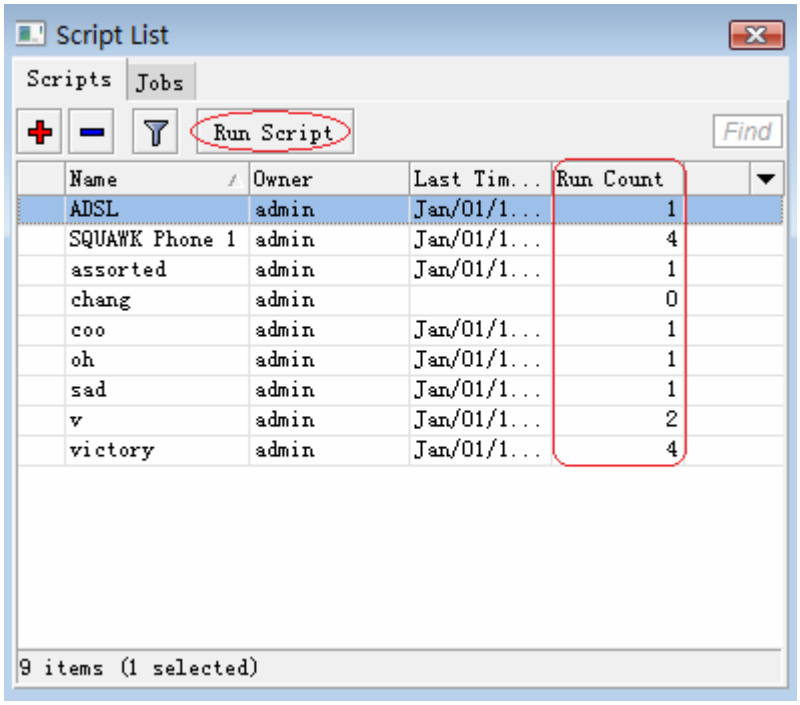
脚本编写进入 script 目录下，在 script 中我们可以定义多条脚本规则,如下图：



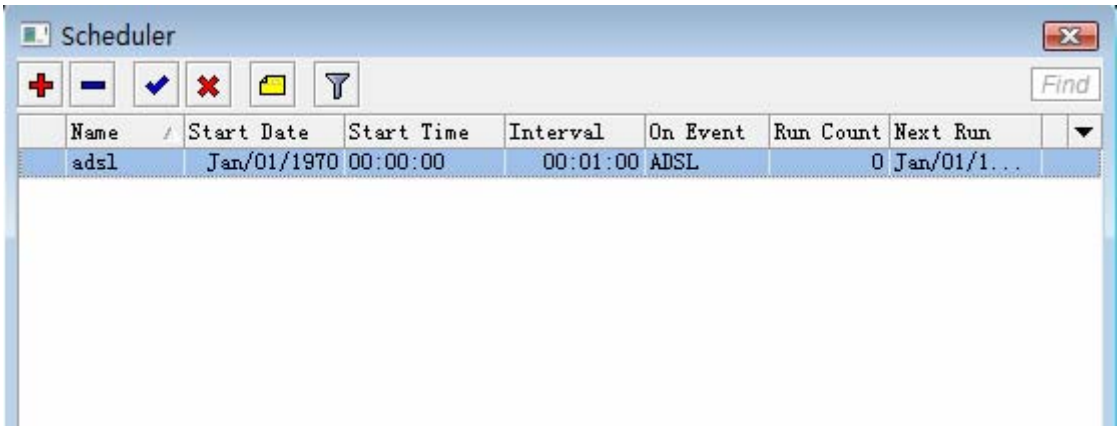
我通过 script 编辑器编辑脚本：



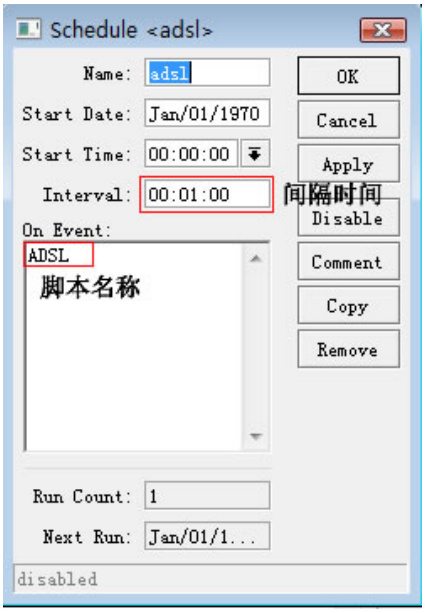
我们可以通过 Run Script 命令运行当前的脚本，在 Run Count 中会纪录脚本运行的次数：



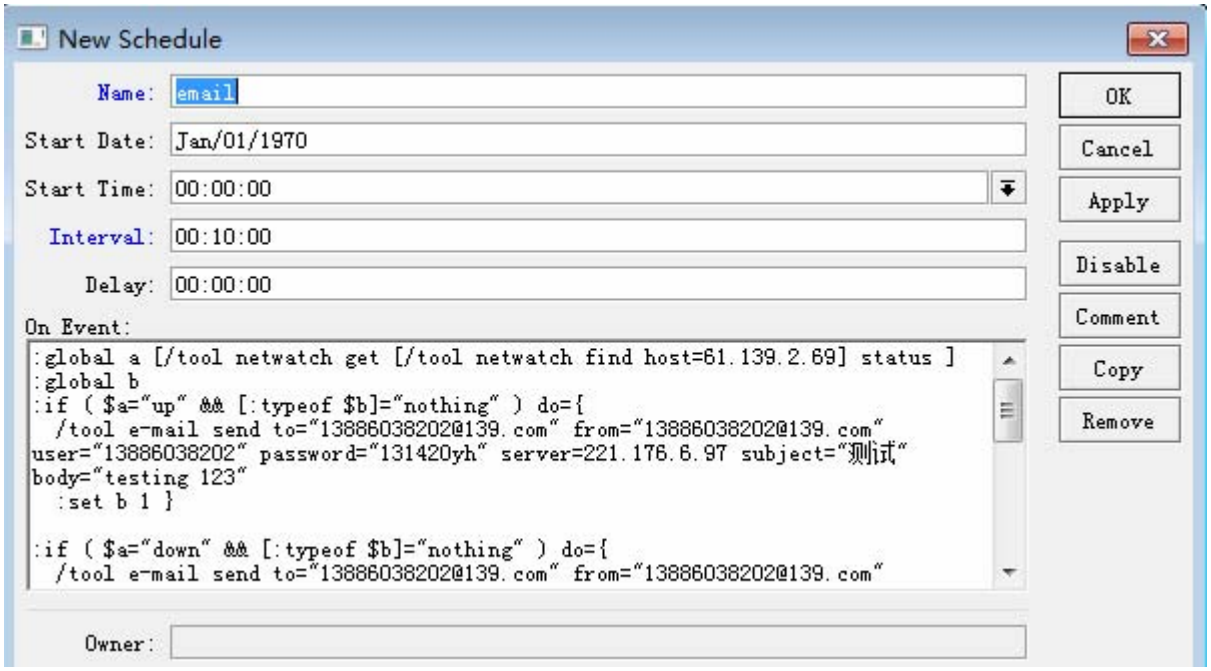
在 Script 中我们编辑好脚本后，我们可以通过 RouterOS 相应的功能调用脚本，并执行这些脚本，如/system scheduler（计划任务）、/tool 中的 netwatch、traffic-monitor 等。通常执行脚本在 scheduler 计划任务中最常用，如这里我们使用了 ADSL 脚本，需要每间隔 1 分钟执行一次，我们可以通过 scheduler 来完成，通过/system scheduler 进入计划任务目录：



添加一个名称为 adsl 的计划任务规则，设置 Interval 时间 1 分钟，On Event 中添加脚本名称：



注：我们也可以在 on event 中直接输入脚本的命令：



第二章 RouterOS 脚本语法

RouterOS 脚本被划分为多个命令行，命令行是一个接一个单元执行直到脚本结束或者直到 错误发生。

命令行结构

RouterOS 控制台是使用下面的命令语法：

[前缀] [路径] 命令 [未命名参数] [参数=[值]] .. [参数=[值]]

- **[前缀]** – 如果命令是 ICE 或者路径通过":" 或者 "/" 字符表示

- **[路径]** – 得到操作菜单的路径
- **命令** – 一个命令获取在指定的菜单路径下
- **[未命名参数]** – 即事先定义参数，如果命令需要必须指定该参数
- **[参数]** – 按先后顺序各自定义值

命令行结束以**“;”**标识为代表或者换行，在结束命令行有时不需要**“;”**或者换行

独立的命令包含 **(), []** 或者 **{}**不需要任何的结尾命令字符，命令结尾取决于脚本的内容

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

每条命令行包含其他命令行，起始通过方括号定义"[ ]"

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

注意，上面这条代码包含 3 条命令行：

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

命令行能通过多余一个行的方式建立，可以查看后面的行连接规则．几种常见的命令实例

- **Prefix（前缀）** - 指示那一个命令到一个 ICE，如：脚本**:put** 或者命令部分从根目录下执行，如“ **/** ”

```
[admin@MikroTik] ip firewall mangle> /ping 10.0.0.1
```

- **Path（路径）** – 希望到达目录的一个相关路径，如：**.. filter**

```
[admin@MikroTik] ip firewall mangle> .. filter print
```

- **Action（执行）** – 在指定的目录下一个可操作的执行命令，如：**add**

```
[admin@MikroTik] ip firewall mangle> /ip firewall filter add chain=forward
action=drop
```

- **unnamed parameter（无名参数）**- 需要通过一些执行和输入固定格式在命令后的执行名称，如 **10.0.0.1**

```
[admin@MikroTik] ip firewall mangle> /ping 10.0.0.1
```

- **name[=value]（参数值）**- 一个跟在参数名后的各自的值，如：**ssid=myssid**

```
/interface wireless set wlan1 ssid=myssid
```

事例

在下面的例子中解释了控制台内的部分命令:

```
/ping 10.0.0.1 count=5
```

前缀	/
执行	ping
未命名参数	10.0.0.1
name[=值]	count=5

```
.. ip firewall rule input
```

路径	.. ip firewall rule
路径项目	input

```
:for i from=1 to=10 do={:put $i}
```

前缀	:
执行	for
未命名参数	i
name[=值]	from=1 to=10 do={:put $i}

```
/interface monitor-traffic ether1,ether2,ipip1
```

前缀	/
路径	interface
执行	monitor-traffic
未命名参数	ether1,ether2,ipip1

变量

RouterOS 脚本语言支持两种类型的变量，**global**（系统变量）和 **local**（仅当前脚本运行的变量）取变量值使用'**$**'标记符号，但除了 **set** 和 **unset** 后面不需要' **$**'标记外，其他的都需要使用该标记。一个变量必须在脚本中首先被声明，下面有四种类型的变量：

- **全局变量** – 使用 **global** 关键字定义，全局变量可用被所有脚本和通过控制台登陆到的同一台路由器调用。注意，重启后全局变量无法保存。
- **本地变量** – 使用 **local** 关键字定义，本地变量不能和其他任何脚本或其他控制台登陆的共享。本地变量值会随脚本执行完成而丢失。

- **循环变量** – 在 **for** 和 **foreach** 内部定义，这里的变量仅能使用在 do 命令块中，在命令执行完成后将被删除掉
- **监听变量** – 一些 **monitor** 命令在 **do** 中能插入变量或控制命令额。

你分配一个新的变量值使用**:set** 命令，并定义两个为命名的参数：变量名称和新的变量值。如果一个变量不需要长时间被调用，可以通过**:unset** 命令释放变量。如果释放一个本地变量，该值会清空。如果你释放一个全局变量，该值仍然会保存在路由器中，但是在当前脚本无法调用。

*注：*循环变量会影响到前面已经声明过的同样名称的变量。

事例

```
[admin@MikroTik] ip route> /
[admin@MikroTik] > :global g1 "this is global variable"
[admin@MikroTik] > :put $g1
this is global variable
[admin@MikroTik] >
```

注释

一个注释从“#”号字符开始执行，并结束在一行的结尾，空格或者任何其他标示不允许在#标示之前。如果“#”字符出现在一个字符串中将不会考虑为一个注释内容。

```
# this is a comment （正确注释）
 # bad comment   （错误注释）
:global a; # bad comment  （错误注释）

:global myStr "lala # this is not a comment"   （不是注释）
```

行连接

通过“\”字符可以将两行或者多行连接到一个逻辑行。一行以反斜杠结束不能进行注释，即一个反斜杠不能连接一个注释。一个字符串不会继续一个指令。

```
:if ($a = true \
     and $b=false) do={ :put “$a $b”; }

:if ($a = true \     # bad comment  （错误注释）
     and $b=false) do={ :put “$a $b”; }

# comment \
   continued – invalid  (语法错误)
```

指令之间的空格

空格可以用于分隔指令。空格必须在两个指令之间仅在他们一系列互相关联的事物解释为一个不通过的指令，例如：

```
{
   :local a true; :local b false;
# 空格不需要
   :put (a&&b);
# 空格需要
   :put (a and b);
}
```

空格不允许

- 在 '<参数>=' 之间不允许
- 在'from=' 'to=' 'step=' 'in=' 'do=' 'else=' 之间不允许

```
#错误:
:for i from = 1 to = 2 do = { :put $i }

#正确:
:for i from=1 to=2 do={ :put $i }
:for i from= 1 to= 2 do={ :put $i }

#错误
/ip route add gateway = 3.3.3.3

#正确
/ip route add gateway=3.3.3.3
```

关键字

下面的字符是关键字，不能使用着变量和功能名称：

```
and      or       not      in
```

分隔符

下面记号作为分隔符的语法：

```
()  []  {}  :   ;   $   /
```

命令替换和返回值

一些终端命令是非常有用的，如他们可以输出一个变量值给其他命令。在 RouterOS 终端控制中通过命令得到返回值。返回值不会被显示出来。从一个命令中得到返回值，应包含在“[ ]”括号中。之前执行返回值的命令所得到的值包含在括号中，这个称为命令替换。

命令产生的返回值，但不限制 **find**，返回一个参考特殊项目 **ping**，返回 ping 成功的数目， **time**，返回测量时间长度值，**incr** 和 **decr**，返回新的变量值，**add**，返回内部最新建立项目编号

事例

find 命令的使用方法：

```
[admin@MikroTik] > /interface
[admin@MikroTik] interface> find type=ether
[admin@MikroTik] interface>
[admin@MikroTik] interface> :put [find type=ether]
*1,*2
[admin@MikroTik] interface>
```

这个方式你能看到内部控制台的项目编号。自然的，你能使用他们到其他的命令的操作中：

```
[admin@MikroTik] interface> enable [find type=ether]
[admin@MikroTik] interface>
```

运算符

RouterOS 控制台能对数值、时间值、IP 地址、字符串和表等做简单的运算。从一个表达式中得到结果。

命令描述

- - 一元减法。对一个数值做反运算。
- - 二进制减，扣除两个数值、两个时间值、两个 IP 地址或 IP 地址和其数值。
! – 逻辑非（**NOT**）。
/ - 除法运算符。
. – 连接符，连接两个字符串或拼接一个表到其他表上或拼接一个元素给一个表。
^ - 位运算移（**XOR**）。
~ - 按位反， which inverts bits in IP address
***** - 乘法运算符。
& - 位运算与（**AND**）
&& - 逻辑与（**AND**）
+ - 加法运算符。对两个数值、两个时间值或 IP 地址做加法运算。
< - 小于符。返回值为布尔型。
<< - 左移运算符。
<= - 小于等于符，返回值为布尔型。
> - 大于符。返回值为布尔型。
>= - 大于等于符，返回值为布尔型。
>> - 右移运算符。
| - 位运算或（**OR**）
|| - 逻辑或（**OR**），返回值为布尔型。

注： 当比较两个数组的时注意：如果他们各自元素是相等的，那么两个数组即相等。

事例：

运算符的优先级和求值命令

```
[admin@MikroTik] ip firewall rule forward> :put (10+1-6*2=11-12=2+(-3)=-1)
false
[admin@MikroTik] ip firewall rule forward> :put (10+1-6*2=11-12=(2+(-3)=-1))
true
[admin@MikroTik] ip firewall rule forward
```

逻辑非（NOT）

```
[admin@MikroTik] interface> :put (!true)
false
[admin@MikroTik] interface> :put (!(2>3))
true
[admin@MikroTik] interface>
```

逻辑运算

```
[admin@MikroTik] interface> :put (-1<0)
true
[admin@MikroTik] >
1
```

按位反

```
[admin@MikroTik] interface> :put (~255.255.0.0)
0.0.255.255
[admin@MikroTik] interface>
```

加法运算

```
[admin@MikroTik] interface> :put (3ms + 5s)
00:00:05.003
[admin@MikroTik] interface> :put (10.0.0.15 + 0.0.10.0)
cannot add ip address to ip address
[admin@MikroTik] interface> :put (10.0.0.15 + 10)
10.0.0.25
[admin@MikroTik] interface>
```

减法运算

```
[admin@MikroTik] interface> :put (15 - 10)
5
```

```
[admin@MikroTik] interface> :put (10.0.0.15 - 10.0.0.3)
12
[admin@MikroTik] interface> :put (10.0.0.15 - 12)
10.0.0.3
[admin@MikroTik] interface> :put (15h - 2s)
14:59:58
[admin@MikroTik] interface>
```

乘法运算

```
[admin@MikroTik] interface> :put (12s * 4)
00:00:48
[admin@MikroTik] interface> :put (-5 * -2)
10
[admin@MikroTik] interface>
```

除法运算

```
[admin@MikroTik] interface> :put (10s / 3)
00:00:03.333
[admin@MikroTik] interface> :put (5 / 2)
2
[admin@MikroTik] interface>
[admin@MikroTik] > :put (0:0.10 / 3)
00:00:02
[admin@MikroTik] >
```

各种逻辑比较运算

```
[admin@MikroTik] interface> :put (10.0.2.3<=2.0.3.10)
false
[admin@MikroTik] interface> :put (100000s>27h)
true
[admin@MikroTik] interface> :put (60s,1d!=1m,3600s)
true
[admin@MikroTik] interface> :put (bridge=routing)
false
[admin@MikroTik] interface> :put (yes=false)
false
[admin@MikroTik] interface> :put (true=aye)
false
[admin@MikroTik] interface>
```

逻辑与和或运算

```
[admin@MikroTik] interface> :put ((yes && yes) || (yes && no))
true
[admin@MikroTik] interface> :put ((no || no) && (no || yes))
false
[admin@MikroTik] interface>
```

按位与、或、异或运算

```
[admin@MikroTik] interface> :put (10.16.0.134 & ~255.255.255.0)
0.0.0.134
[admin@MikroTik] interface>
```

移位运算

```
[admin@MikroTik] interface> :put (~((0.0.0.1 << 7) - 1))
255.255.255.128
[admin@MikroTik] interface>
```

连接运算符

```
[admin@MikroTik] interface> :put (1 . 3)
13
[admin@MikroTik] interface> :put (1,2 . 3)
1,2,3
[admin@MikroTik] interface> :put (1 . 3,4)
13,4
[admin@MikroTik] interface> :put (1,2 . 3,4)
1,2,3,4
[admin@MikroTik] interface> :put ((1 . 3) + 1)
14
[admin@MikroTik] interface>
```

数据类型

RouterOS 区分几种数据类型，字符型、布尔型、数值型、时间型、IP 地址、内码和列表。 RouterOS 首先会试着将任何值转换为制定的类型。

内部脚本语言弥补了特殊函数之间类型转换的不足。通过内部的 **toarray**, **tobool**, **toid**, **toip**, **tonum**, **tostr** 和 **totime** 函数每个值转换到相应的列表（**list**）中，对应为：**boolean**, **internal number**, **IP address**, **number**, **string** 或 **time**.

数字类型在内部表示为 64 位带符号的整型， 因此一个数字类型值变量可用长度从 -9223372036854775808 到 9223372036854775807. 同样可用输入十六进制的数值，在前面加入 **0x** ，例如：

```
[admin@MikroTik] > :global MyVar 0x10
[admin@MikroTik] > :put $MyVar
16
[admin@MikroTik] >
```

列表通过逗号来区分值的次序，在空白出使用逗号间隔方式部推荐使用，因为这会让控制终端无法识别字符的边境。

Boolean 型的值为 **true** 或 **false**. 控制终端判断 **true** 为 **“yes”**，**false** 为 “**no**” 。

时间间隔可以输入 HH:MM:SS 例如：

```
[admin@MikroTik] > :put 01:12:1.01
01:12:01.010
[admin@MikroTik] >
```

或者通过累计数字，具体指明单位时间的标记(**d** 对应 days, **h** 对应 hours, **m** 对应 minutes, **s** 对应 seconds，以及 **ms** 对应 milliseconds)例如：

```
[admin@MikroTik] > :put 2d11h12
2d11:00:12
[admin@MikroTik] >
```

时间单位：

- **d, day, days** – 一天，或者 24 小时
- **h, hour, hours** – 一小时
- **m, min** – 一分钟
- **s** – 一秒
- **ms** – 一毫秒，等同 0.001 秒

控制终端同样接受时间为小数点的形式：

```
[admin@MikroTik] > :put 0.1day1.2s
02:24:01.200
[admin@MikroTik] >
```

命令参考文档

RouterOS 有多个嵌入式的控制终端命令和表达式 ICE 不依赖于当前操作目录。这些命令不能直接改变配置，但他们可以做日常的维护工作。所有 ICE 可以通过在操作符输入“ ： ”后敲击“ ？ ” 显示出。例如：

```
[admin@MikroTik] > :
beep        execute  global  list    pick    time    toip    typeof
```

```
delay       find    if     local    put     toarray  tonum   while
do          for     led    log      resolve  tobool   tostr
environment  foreach  len    nothing  set     toid    totime
[admin@MikroTik] >
```

命令属性

beep – 通过 PC 内置的蜂鸣器或者扬声器发出一个指定 **length** （时间长度）的 **frequency** Hz（频率 Hz）.

输入参数

frequency (*整型*; 默认: **1000**) – 信号频率大小用单位 Hz

length (时间; 默认: **100ms**) – 信号长度

```
[admin@MikroTik] > :beep length=2s frequency=10000

[admin@MikroTik] >
```

delay – 在一个给定的时间长度不做任何操作

输入参数

delay-time (*时间*) – 等待的时间长度

- **omitted** – 无限制延迟

do – 反复执行命令直到获取一个适当的值。如果没有参数获取，**do** 只执行有效操作一次，其中不会有什么作用。如果一逻辑条件被指定到 **while** 参数种,将会在命令执行后作判断，在该条件判断中为 ***true***，**do** 语句会被一次一次的执行直到满足 ***false*** 条件， **if** 参数，在做后面语句的任何操作时判断一次，如果 ***false*** 不会执行任何的操作

输入参数

unnamed (*文本*) – 反复执行的操作

while (yes | no) – 条件语句，这是判断每一次执行后的执行结果

if (yes | no) – 条件语句，这是判断一次执行之前的声明

```
[admin@MikroTik] > {:global i 10; :do {:put $i; :set i ($i - 1);} \
\... while (($i < 11) && ($i > 0)); :unset i;}
10
9
8
7
6
5
4
3
2
1
[admin@MikroTik] >
```

environment print – 显示关于当前变量的初始化情况。所有在系统中的全局变量 global variables 被以标题为 **Global Variables** 下列出。所有变量插入当前脚本（通过**:local**、通过**:for** 或者**:foreach**）被以标题为 **Local Variables** 下列出。

创建变量并显示出他们的列表：

```
 [admin@MikroTik] > :local A "This is a local variable"
```

```
[admin@MikroTik] > :global B "This is a global one"
[admin@MikroTik] > :environment print
Global Variables
B=This is a global one
Local Variables
A=This is a local variable
[admin@MikroTik] >
```

find – 查找字符串内的一个字符或者一个元素在项目内的值，根据变量类型并返回一个变量的位置。

输入参数

unnamed(*文本* | 列表) – 搜索字符或者字符列表值，并执行相关的操作

unnamed(*文本*) – 字符的搜索

unnamed（整型) – 搜索的起始位置，或搜索到目标返回的位置

```
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333"  ]
0
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333" 3  ]
1
[admin@MikroTik] interface pppoe-server> :put [:find "13sdf1sdfss1sfsdf324333" 3  3]
17
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 ]
4
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 3]
4
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 4]
5
[admin@MikroTik] interface pppoe-server> :put [:find "1,1,1,2,3,3,4,5,6,7,8,9,0,1,2,3"
3 5]
15
[admin@MikroTik]
```

for – 执行所给定的数次反复循环的命令，通过 **from** 和 **to** 设置起始和结算参数。

输入参数

unnamed (*名称*) – 循环计数器变量名称。

from (*整型*) – 循环起始的变量值

to (*整型*) – 循环结束的变量值

step (整型; 默认: **1**) – 递增变量. 在循环起始到结束中间每循环一次的间距变量

do (*文本*) – 执行包含在内的命令

```
[admin@MikroTik] > :for i from=1 to=100 step=37  do={:put ($i . " - " . 1000/$i)}
1 - 1000
38 - 26
75 - 13
[admin@MikroTik] >
```

foreach – 执行所提供在列表中的每一个元素

输入参数

unnamed (*名称*) -循环计数器变量名称。

in (*列表*) – 变量列表范围或路径

do (*text*) - 执行包含在内的命令

显示出一个 interface 中获得列表各自的 IP 地址

```
:foreach i in=[/interface find type=ether ] \
\... do={:put ("+--" . [/interface get $i name]); \
\... :foreach j in=[/ip address find interface=$i]
\... do={:put ("|  `--" . [/ip address get $j address])}}
+--ether1
|  `--1.1.1.3/24
|  `--192.168.50.1/24
|  `--10.0.0.2/24
+--ether2
|  `--10.10.0.2/24
[admin@MikroTik] >
```

global – 声明全局变量

输入参数

unnamed(*名称*) – 变量名称

unnamed(*文本*) – 值，分配给变量的内容

```
[admin@MikroTik] > :global MyString "This is a string"
[admin@MikroTik] > :global IPAddr 10.0.0.1
[admin@MikroTik] > :global time  0:10
[admin@MikroTik] > :environment print
Global Variables
IPAddr=10.0.0.1
time=00:10:00
MyString=This is a string
Local Variables
[admin@MikroTik] >
```

if – 条件语句. 如果一个逻辑判断为真，这时执行 **do** 分程序块中的命令，否则选择 **else** 分程序块中的执行。

输入参数

unnamed(yes | no) – 逻辑条件语句，在执行之后声明内容前判断一次

do(*文本*) – 如果 if 语句判断为真，在这个分程序块的命令会被执行。

else(*文本*) -如果 if 语句判断为假，在这个分程序块的命令会被执行。

通过 if 语句检查 firewall 中是否有任何规则被添加

```
[admin@MikroTik] > :if ([:len [/ip firewall filter find]] > 0) do={:put true} else={:put
false}
true
[admin@MikroTik] >
```

检查网关是否能到达。在这个事例中网关地址为 **10.0.0.254**

```
[admin@MikroTik] > :if ([/ping 10.0.0.254 count=1] = 0) do {:put "gateway unreachable"}
10.0.0.254 ping timeout
1 packets transmitted, 0 packets received, 100% packet loss
gateway unreachable
```

```
[admin@MikroTik] >
```

led – 允许控制系统内嵌的 LED（发光二极管）。这个命令仅能在 RouterBOARD 平台与安装 **routerboard** 或 **rb500** 功能包。LED 数量根据 RouterBOARD 型号不同而定。

输入参数

led1(yes | no) – 控制第一个 LED

led2(yes | no) -控制第二个 LED

led3(yes | no) -控制第三个 LED

led4(yes | no) -控制第四个 LED

length(*time*) – 具体指定操作的长度

- **omitted** – LED 长亮

打开 LED2 和 LED3 时间为 5 秒

```
[admin@MikroTik] > :led led2=yes led3=yes length=5s
```

len – 返回在字符串的字符数或列表中的元素数目

输入参数

unnamed(*name*) – 返回字符串或列表的长度

```
[admin@MikroTik] > :put [:len gvejimezyfopmekun]
17
[admin@MikroTik] > :put [:len gve,jim,ezy,fop,mek,un]
6
[admin@MikroTik] >
```

list – 显示一个能获得给定匹配关键字的列表

输入参数

unnamed(*文本*) – 第一个搜索关键字

unnamed(*文本*) -第二个搜索关键字

unnamed(*文本*) -第三个搜索关键字

显示控制台命令下含有 **hotspot**, **add** 和 **user** 部分的命令与路径

```
[admin@MikroTik] > :list user hotspot "add "
List of console commands under "/" matching "user" and "hotspot" and "add ":

ip hotspot profile add name= hotspot-address= dns-name= \
\... html-directory= rate-limit= http-proxy= smtp-server= \
\... login-by= http-cookie-lifetime= ssl-certificate= split-user-domain= \
\... use-radius= radius-accounting= radius-interim-update= copy-from=
ip hotspot user add server= name= password= address= mac-address= \
\... profile= routes= limit-uptime= limit-bytes-in= limit-bytes-out= \
\... copy-from= comment= disabled=
ip hotspot user profile add name= address-pool= session-timeout= \
\... idle-timeout= keepalive-timeout= status-autorefresh= \
\... shared-users= rate-limit= incoming-filter= outgoing-filter= \
\... incoming-mark= outgoing-mark= open-status-page= on-login= on-logout= copy-from=
[admin@MikroTik] >
```

local – 声明本地变量

输入参数

unnamed(*名称*) – 变量名称

unnamed(*文本*) – 值，分配给变量的内容

```
[admin@MikroTik] > :local MyString "This is a string"
[admin@MikroTik] > :local IPAddr 10.0.0.1
[admin@MikroTik] > :local time  0:10
[admin@MikroTik] > :environment print
Global Variables
Local Variables
IPAddr=10.0.0.1
time=00:10:00
MyString=This is a string
[admin@MikroTik] >
```

log – 通过参数添加一个指定的信息到系统 logs 中。

输入参数

unnamed(*名称*) – 记录日志的功能名称

unnamed(*文本*) – 被记录的文本信息

发送信息到 **info** 日志中

```
[admin@MikroTik] > :log info "Very Good thing happened. We have received our first packet!"
[admin@MikroTik] > /log print follow
...
19:57:46 script,info Very Good thing happened. We have received our first packet!
...
```

nothing – 没有任何操作，并返回值类型为“nothing”。 在条件语句中 nothing 等同“false”

从一个字符串中挑选一个不存在的符号

```
[admin@MikroTik] > :local string qwerty
[admin@MikroTik] > :if ([:pick $string 10]=[:nothing]) do={
{... :put "pick and nothing commands return the same value"}
pick and nothing commands return the same value
[admin@MikroTik] >
```

pick – 根据输入的值返回一个元素长度或一个子串值

输入参数

unnamed(*文本* | 列表) – 字符串或值列表的来源

unnamed(*整型*) – 字符串中子串的起始位置

unnamed(*整型*) -字符串中子串的结束位置

```
[admin@MikroTik] > :set a 1,2,3,4,5,6,7,8
[admin@MikroTik] > :put [:len $a]
8
[admin@MikroTik] > :put [:pick $a]
1
[admin@MikroTik] > :put [:pick $a 0 4]
1,2,3,4
[admin@MikroTik] > :put [:pick $a 2 4]
3,4
[admin@MikroTik] > :put [:pick $a 2]
3
[admin@MikroTik] > :put [:pick $a 5 1000000]
6,7,8
[admin@MikroTik] > :set a abcdefghij
```

```
[admin@MikroTik] > :put [:len $a]
10
[admin@MikroTik] > :put [:pick $a]
a
[admin@MikroTik] > :put [:pick $a 0 4]
abcd
[admin@MikroTik] > :put [:pick $a 2 4]
cd
[admin@MikroTik] > :put [:pick $a 2]
c
[admin@MikroTik] > :put [:pick $a 5 1000000]
fghij
```

put – 回复所提供的变量值到控制台

输入参数

unnamed(*文本*) – 需要回复的文本信息

显示 **ether1** 接口的 MTU 值

```
[admin@MikroTik] > :put [/interface get ether1 mtu]
1500
[admin@MikroTik] >
```

resolve – 解析 DNS 域名并返回主机的 IP 地址，首先需要配置好路由器的 DNS 参数(**/ip dns** 目录下)

输入参数

unnamed(*文本*) – 需要解析 IP 的主机域名

DNS 配置和 **resolve** 命令事例

```
[admin@MikroTik] ip route> /ip dns set primary-dns=159.148.60.2
[admin@MikroTik] ip route> :put [:resolve “www.example.com”]
192.0.34.166
```

set – 分配一个新值给变量

输入参数

unnamed(*name*) – 变量名称

unnamed(*text*) – 新的变量值

通过 **/ip route find dst 0.0.0.0** 的命令，查找路由表中 **dst-address** 返回值为 **0.0.0.0** 的值，这个值通常是路由器的默认网关，当查到后通过**/ip route set** 命令修改网关地址为 10.0.0.217

```
[admin@MikroTik] > /ip route set [/ip route find dst “0.0.0.0”] gateway 10.0.0.1
[admin@MikroTik] >
```

time – 计算出所给命令的执行时间总长度

输入参数

unnamed(*text*) – 控制台命令测量执行时间

计算出解析 www.example.com 需要的时间

```
[admin@MikroTik] > :put [:time [:resolve “www.example.com” ]]
00:00:00.006
[admin@MikroTik] >
```

while – 反复执行给定的控制命令，直到逻辑条件为 *true*

输入参数

unnamed(yes | no) – 条件，在每一次执行前判断声明范围

do(*文本*) – 反复执行的控制命令

```
[admin@MikroTik] > :set i 0; :while ($i < 10) do={:put $i; :set i ($i + 1)};
0
1
2
3
4
5
6
7
8
9
[admin@MikroTik] >
```

Typeof – 判断变量类型，返回值 num、str、nothing、time、ip、bool

输入参数

unnamed(*name*) – 变量名称

判断一个变量的类型

```
[admin@MikroTik] >:global a 192.168.1.1
[admin@MikroTik] > :put [:typeof $a]
ip
[admin@MikroTik] >:global b
[admin@MikroTik] >:put [:typeof $b]
nothing
```

第三章 Scripte 事例

自动创建多条策略

在 firewall **input** 规则中通过脚本添加接受从 1.1.1.1 开始到 1.1.1.100 地址的数据包，即 100 条规则 ：

```
:for e from 1 to 100 do={
  /ip firewall filte add \
    chain=input src-address=(“1.1.1.” . $e)
   }
```

我们使用类似的脚本编写，添加 200 个 IP 地址的流量规则：

```
:for e from 1 to 200 do={
  /queue simple add target-addresses=("192.168.1." . $e) max-limit=256000/512000
}
```

获取 bandwidth 测试参数

这个事例描述的是如果获取 **bandwidth-test** 命令的结果。 在事例中使用 global（全局变量）另外一个脚本在同一时间运行，并获取当前的 TX 参数。

```
:global i
/tool bandwidth-test 1.1.1.1 direction=transmit duration=14s
  do={
    :if ($status="running") do={
      :set i $”tx-current”
      }
    }
}
```

解析域名 IP 地址，并添加入 address-list

这个实例对一些域名有多 IP 地址的网站进行解析，我们可以通过设置计划任务每间隔一个周期执行脚本，比如 netbar.qq.com 进行解析，每次解析对比 **ip firewall address-list** 是否存在相同 IP，如果没有相同 IP 地址，则添加入 **address-list**。

```
:global a [:resolve netbar.qq.com]
:global b
:foreach i in=[/ip firewall address-list find list=qqgame] do={
   :if ($a = [/ip firewall address-list get $i address ]) do={
    :set b 1
  }
   else={
    :set b 0
  }

}
:if ($b = 0) do={ /ip firewall address-list add list=qqgame address=$a }
```

使用注释

如果有时候在一个规则中与其他规则许多属性相同，指定的参数就无法从规则中获取（例如：firewall 或者 routing 规则等）。这里我们可以通过编辑注释来解决。I

假设，我们需要从 ip firewall nat 中修改端口映射的目标地址，目标 IP 地址是 218.16.18.12，而我们做了多条端口映射规则，为方便查找和修改，我们可以通过注释标记。

```
[admin@MikroTik] ip firewall nat> set 0 comment=dst1
```

现在我们看看一个具体的应用实例，ADSL 的 IP 地址变动后，我们修改端口映射的 IP 地址：

```
:global adsl "pppoe-out1"   # 定义 ADSL 拨号接口名称变量: pppoe-out1
:global adsllast       # 之前的 ADSL IP 地址变量

:global adslip [ /ip address get [/ip address find interface=$adsl] address ] //获取当前 ADSL ip 地址

:if ([ :typeof $adsllast ] = nil ) do={ :set adsllast 0.0.0.0/0 } # 判断之前的 ADSL IP 地址是否为空,如果为空设置一个 0.0.0.0/0 的 IP 地址

:if ([ :typeof $adsl ] = nil ) do={

  :log info ("No ip address present on " . $adsl . ", please check.") # 判断当前 ADSL IP 是否为空,否则执行 else 的操作

} else={

  :if ($adslip != $adsllast) do={
    :log info [/ip firewall nat set [/ip firewall nat find comment = “dst1”] dst-address=$adslip ]    #  判断当前 IP 和之前 IP 是否相同，如果不同便修改 dst-address 的 IP 地址
    :log info "ADSL IP: UPDATE!"

    :set adsllast $adslip   # 交换 IP 地址

  } else={

    :log info "ADSL IP: No change"
 }

}
```

在这个事例中，我们可以看到通过添加注释，为编写脚本判断正确的规则。

DDNS 动态域名配置

这里我们来看看，当我们使用 ADSL 时由于重新拨号或者租约到期，IP 地址是动态变化的，而又需要让外网的客户通过动态域名来访问我们的内部服务器，这里我们需要使用到 DDNS 的动态域名脚本，注意 RouterOS 支持的动态域名服务器只有 www.changeip.com

```
:log info “DDNS: Begin“    #在 log 日志中显示 DDNS 开始运行

:global ddnsuser “ddns.test.com“   # 定义 ddnsuser 用户名变量
:global ddnspass “cdnat“              # 定义 ddnspass 密码变量
:global ddnshost “ddns.test.com“   # 定义 ddnshost 主机
:global ddnsinterface “pppoe-out1“  # 定义 ADSL 拨号接口，用户获取 IP
:global ddnsipt  #定义一个零时存储的 IP 地址变量
:global ddnsip [ /ip address get [/ip address find interface=$ddnsinterface] address ]
# 根据 ADSL 拨号的接口获取 IP 地址，并分配给 ddnsip
```

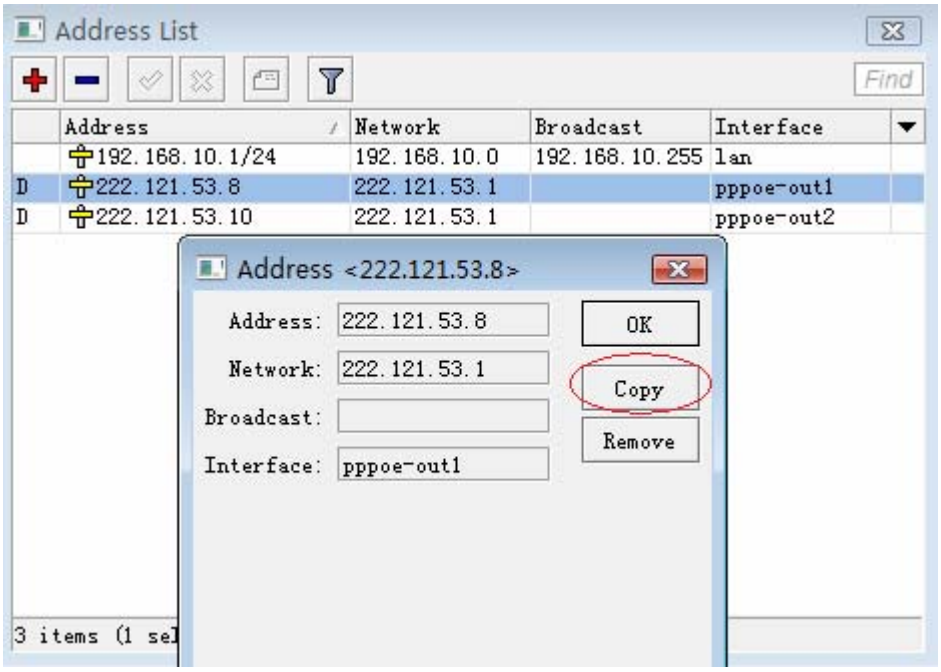
```
:if ([ :typeof $ddnslastip ] = nil ) do={ :global ddnslastip 0.0.0.0/0 }  # 查看 ddnslastip 变量是否为空，如果为空则分配 0.0.0.0/0 地址
:if ([ :typeof $ddnsip ] = nil ) do={

  :log info (“DDNS: No ip address present on ” . $ddnsinterface . “, please check.”)
# 查看 ddns-ip 变量是否为空，如果为空则在 log 中提示未获取 IP

} else={

  :if ($ddnsip != $ddnslastip) do={                                    # 否则执行新 IP 与以前的 IP 地址对比
    :set ddnsipt [:pick $a 0 ([:len $a] - 3)]                          # 去掉 IP 地址后面的子网掩码，并将修改后的 IP 分配给 ddns-ipt 变量
    :log info [/ip fi nat set [/ip fi nat find comment = s1] to-address $ddnsipt ]    # 根据注释名称修改相应的 nat 规则
    :log info [/ip fi nat set [/ip fi nat find comment = a1] dst-address $ddnsip ]
    :log info [/ip fi nat set [/ip fi nat find comment = a2] dst-address $ddnsip ]
    :log info [/ip fi nat set [/ip fi nat find comment = a3] dst-address $ddnsip ]
    :log info [/ip fi nat set [/ip fi nat find comment = a4] dst-address $ddnsip ]
    :log info [/ip fi nat set [/ip fi nat find comment = a5] dst-address $ddnsip ]
    :log info [/ip fi nat set [/ip fi nat find comment = a6] dst-address $ddnsip ]

    :log info “DDNS: Sending UPDATE!“     # log 中提示 DDNS 更新
    :log info [ /tool dns-update name=$ddnshost address=[:pick $ddnsip 0 [:find $ddnsip "/"] ] key-name=$ddnsuser key=$ddnspass ]
    # 发送最新的 IP 地址，到 DDNS 服务器
    :global ddnslastip $ddnsip   # 将新老 IP 地址交换

  } else={
    :log info “DDNS: No change”
 }
}
:log info "DDNS: End"
```

相同 ADSL 网关脚本修改

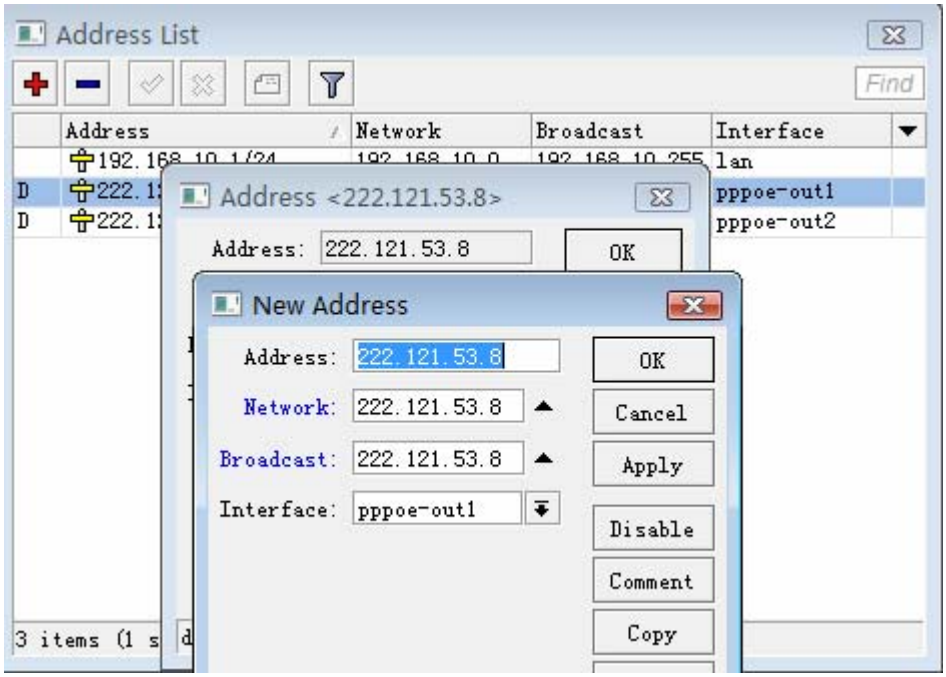
*注：*该脚本适用于 3.0 版本，4.0 在相同 ADSL 网关下可以不同脚本实现操作。

当我们使用到多线路的 ADSL 时，可能会遇到这些 ADSL 的网关是相同的，因为 RouterOS 只能识别不同网关的策略路由，因此在多 ADSL 相同网关的情况下，我们可以通过隧道协议的特点，用本地 IP 地址做为路由器的网关。

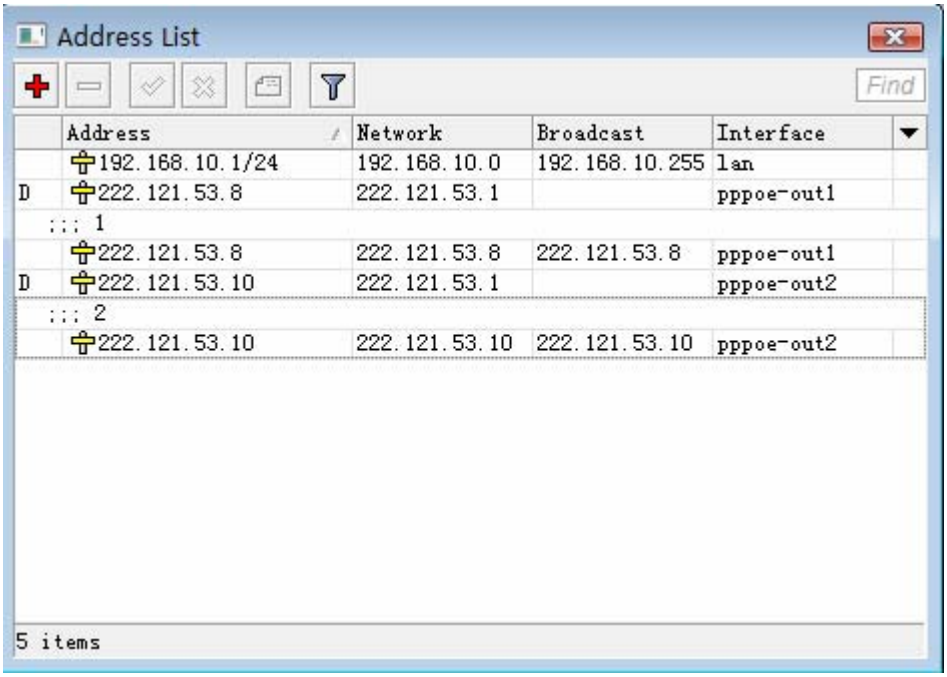
假设我们有 2 条 ADSL，分别为 pppoe-out1 和 pppoe-out2，做 ADSL 操作的时候需要将动态获取的 ADSL 地址修改为静态，通过在/ip address 中使用 copy 命令操作，如下图



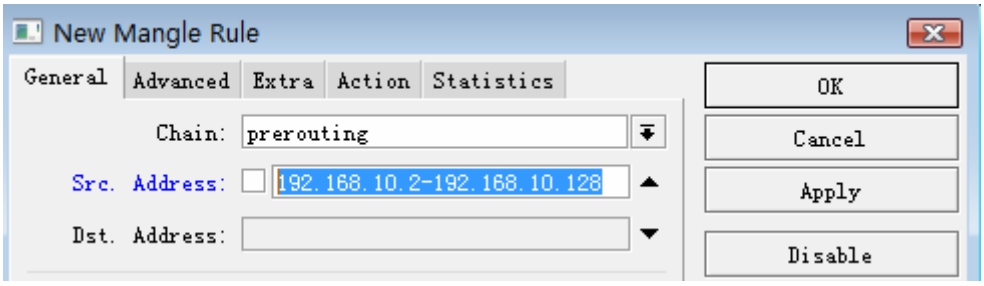
上图获取到的 IP 地址是 222.121.53.8，通过 copy 命令修改 Network 和 Boradcast 地址也为 222.121.53.8，如下：

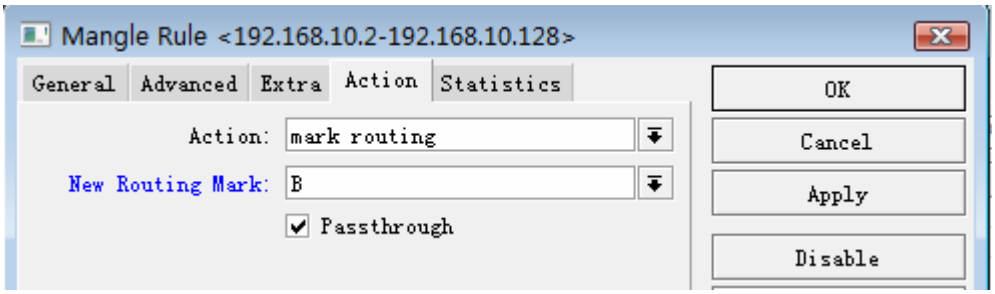


将下面 2 条 ADSL 的 IP 地址都修改为静态的，并给静态的 IP 地址标记上注释 1 和 2：



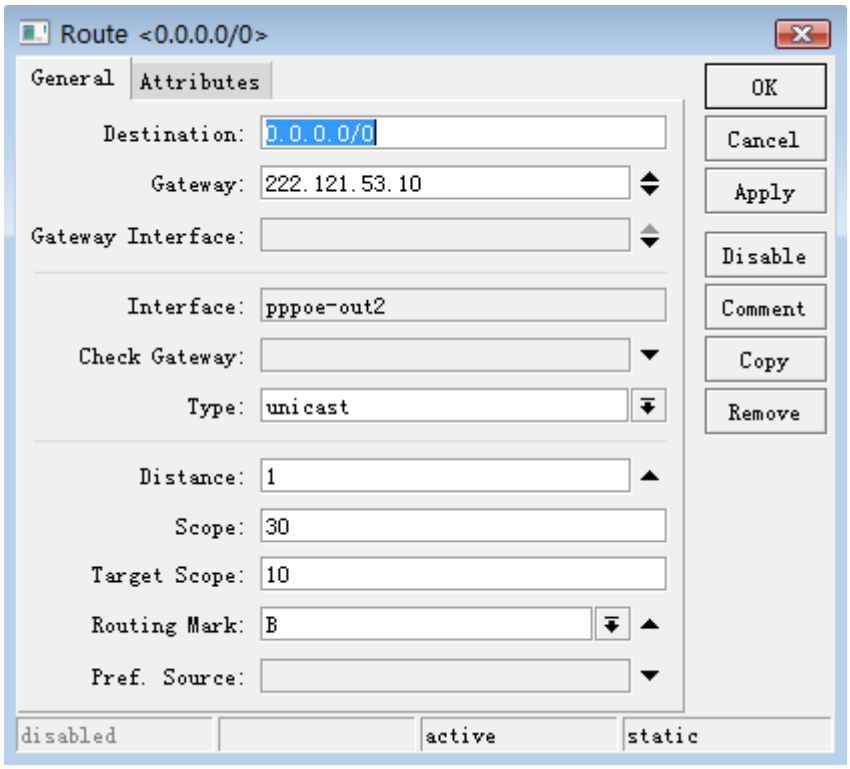
我们将网络的内的用户平均分组为 A 和 B，A 组走 1 号 ADSL 线路，而 B 组走 2 号 ADSL 线路。1 号 ADSL 设置为默认路由，即 RouterOS 的默认网关出口，这里我们将 2 号线作为用户 B 组的策略路由，我们在 ip firewall mangle 中配置，定义 chain=prerouting src-address=192.168.10.2-192.168.10.128 action=mark-routing new-routing-mark=B



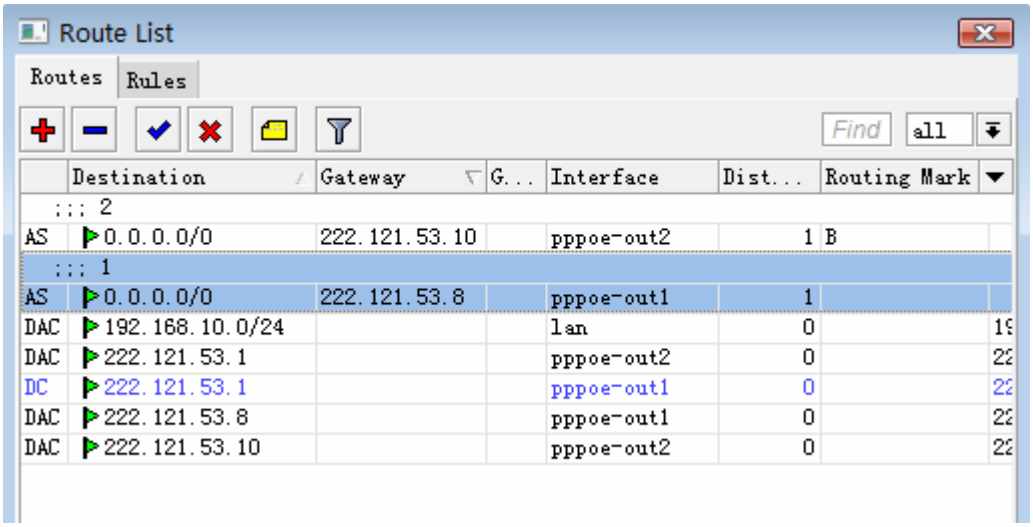


这里我们只需要定义 B 组用户，A 组用户只需要走 1 号 ADSL 的默认网关，定义 B 组用户走 2 号线路的 ADSL

配置路由，这里我们设置 1 号 ADSL 的 IP222.121.53.8 为默认网关，即 ADSL 的默认网关。2 号线路的 ADSL 我们用 222.121.53.10，做为 B 组线路的网关，并在 ip route 中添加 routing-mark 的标记



为 pppoe-out2 做注释标记 2，配置完成后的路由，B 组用户通过 22.121.53.10 的网关出去，剩下的用户通过默认网关如下图



接下来我们需要通过脚本，来判断 1 和 2 号线 ADSL 的 IP 地址是否变动，如果 IP 变动后用脚本会自动修改变动的 1 和 2 号线 ADSL 参数

```
:local lastaddress
:local newaddress
:local status

:set status [/interface get [/interface find  name="pppoe-out1" ] running]
:if ($status=true) do={
     :set newaddress [/ip address get [/ip address find dynamic=yes
interface="pppoe-out1" ] address]
     :set newaddress [:pick $newaddress 0 [:find $newaddress “/”]]
```

```
     :set lastaddress [/ip address get [/ip address find dynamic=no
interface="pppoe-out1"] address]
     :set lastaddress [:pick $lastaddress 0 [:find $lastaddress “/”]]
     :if ($lastaddress != $newaddress) do={
          :log info [/ip address set [/ip address find comment=”1”] address=$newaddress
network=$newaddress broadcast=$newaddress]
         :log info [/ip route set [/ip route find comment=”1”] gateway=$newaddress]
        }
 }
:set status [/interface get [/interface find  name="pppoe-out2" ] running]
:if ($status=true) do={
     :set newaddress [/ip address get [/ip address find dynamic=yes
interface="pppoe-out2" ] address]
     :set newaddress [:pick $newaddress 0 [:find $newaddress “/”]]
     :set lastaddress [/ip address get [/ip address find dynamic=no
interface="pppoe-out2"] address]
     :set lastaddress [:pick $lastaddress 0 [:find $lastaddress “/”]]
     :if ($lastaddress != $newaddress) do={
          :log info [/ip address set [/ip address find comment=”2”] address=$newaddress
network=$newaddress broadcast=$newaddress]
         :log info [/ip route set [/ip route find comment=”2”] gateway=$newaddress]
        }
 }
```

通过声控判断 WLAN 信号强度

我们可以使用 RouterOS 的脚本 beep 语句，配合循环操作，来判断 WLAN 的信号强度，该脚本主要应用在点对点的 WLAN 搜索信号使用。

下面是 ap-bridge 使用的声控信号强度脚本，注意 ap-bridge 使用的时候需要填写对方无线模块的 MAC 地址，才能获取信号强度

```
:local beep "10ms";
:local s85 "1350ms";
:local s80 "850ms";
:local s75 "650ms";
:local s70 "450ms";
:local s65 "350ms";
:local s60 "250ms";
:local s55 "200ms";
:local s50 "150ms";
:local s45 "100ms";
:local s40 "60ms";
:local s20 "20ms";
:global fr
:for i from=1 to=50 do={
:set fr [/interface wireless registration-table get [/interface wireless
registration-table find radio-name="000C4223D23E"] signal-strength ]
```

```
:set fr [:pick $fr 0 [:find $fr "d" ]]
 :if ($fr <= -85 && $fr > -88) do={
  :for i from=1 to=2 do={ :beep length=$beep; :delay $s85; }
 }
 :if ($fr <= -80 && $fr > -85) do={
  :for i from=1 to=3 do={ :beep length=$beep; :delay $s80; }
 }
 :if ($fr <= -75 && $fr > -80) do={
  :for i from=1 to=3 do={ :beep length=$beep; :delay $s75; }
 }
 :if ($fr <= -70 && $fr > -75) do={
  :for i from=1 to=6 do={ :beep length=$beep; :delay $s70; }
 }
 :if ($fr <= -65 && $fr > -70) do={
  :for i from=1 to=8 do={ :beep length=$beep; :delay $s65; }
 }
 :if ($fr <= -60 && $fr > -65) do={
  :for i from=1 to=11 do={ :beep length=$beep; :delay $s60; }
 }
 :if ($fr <= -55 && $fr > -60) do={
  :for i from=1 to=13 do={ :beep length=$beep; :delay $s55; }
 }
 :if ($fr <= -50 && $fr > -55) do={
  :for i from=1 to=18 do={ :beep length=$beep; :delay $s50; }
 }
 :if ($fr <= -45 && $fr > -50) do={
  :for i from=1 to=25 do={ :beep length=$beep; :delay $s45; }
 }
 :if ($fr <= -40 && $fr > -45) do={
  :for i from=1 to=31 do={ :beep length=$beep; :delay $s40; }
 }
 :if ($fr <= -20 && $fr > -40) do={
  :for i from=1 to=40 do={ :beep length=$beep; :delay $s20; }
 }

 }
}
```

该脚本在信号强度越强的情况下，发声频率越高

下面是 Station-wds 使用的声控信号强度脚本：

```
:local beep "10ms";
:local s85 "1350ms";
:local s80 "850ms";
:local s75 "650ms";
:local s70 "450ms";
:local s65 "350ms";
```

```
:local s60 "250ms";
:local s55 "200ms";
:local s50 "150ms";
:local s45 "100ms";
:local s40 "60ms";
:local s20 "20ms";
:for i from=1 to=100 do={
/interface wireless monitor wlan1 interval=1 do={
 :if ($"signal-strength" <= -85 && $"signal-strength" > -88) do={
  :for i from=1 to=2 do={ :beep length=$beep; :delay $s85; }
  }
 :if ($"signal-strength" <= -80 && $"signal-strength" > -85) do={
  :for i from=1 to=3 do={ :beep length=$beep; :delay $s80; }
  }
 :if ($"signal-strength" <= -75 && $"signal-strength" > -80) do={
  :for i from=1 to=4 do={ :beep length=$beep; :delay $s75; }
  }
 :if ($"signal-strength" <= -70 && $"signal-strength" > -75) do={
  :for i from=1 to=6 do={ :beep length=$beep; :delay $s70; }
  }
 :if ($"signal-strength" <= -65 && $"signal-strength" > -70) do={
  :for i from=1 to=8 do={ :beep length=$beep; :delay $s65; }
  }
 :if ($"signal-strength" <= -60 && $"signal-strength" > -65) do={
  :for i from=1 to=10 do={ :beep length=$beep; :delay $s60; }
  }
 :if ($"signal-strength" <= -55 && $"signal-strength" > -60) do={
  :for i from=1 to=12 do={ :beep length=$beep; :delay $s55; }
  }
 :if ($"signal-strength" <= -50 && $"signal-strength" > -55) do={
  :for i from=1 to=16 do={ :beep length=$beep; :delay $s50; }
  }
 :if ($"signal-strength" <= -45 && $"signal-strength" > -50) do={
  :for i from=1 to=24 do={ :beep length=$beep; :delay $s45; }
  }
 :if ($"signal-strength" <= -40 && $"signal-strength" > -45) do={
  :for i from=1 to=34 do={ :beep length=$beep; :delay $s40; }
  }
 :if ($"signal-strength" <= -20 && $"signal-strength" > -40) do={
  :for i from=1 to=48 do={ :beep length=$beep; :delay $s20; }
  }
 }
}
```

声音控制脚本

警报声

```
:for i from=1 to=3 step=1 do={
  :beep frequency=550 length=494ms;
  :delay 494ms;
  :beep frequency=400 length=494ms;
  :delay 494ms;
 }
```

电话铃声

```
:for i from=1 to=10 step=1 do={
  :beep frequency=1195 length=22ms;
  :delay 22ms;
  :beep frequency=2571 length=22ms;
  :delay 22ms;
 }
```

Coo 发声

```
:for i from=0 to=150 step=10 do={
  :beep frequency=(1295 - i) length=22ms;
  :delay 22ms;
  :beep frequency=(1095 + i) length=22ms;
  :delay 22ms;
 }
```

操作成功发声

```
:beep frequency=523 length=200ms;
 :delay 1000ms;

 :beep frequency=523 length=200ms;
 :delay 1000ms;

 :beep frequency=523 length=200ms;
 :delay 1000ms;

 :beep frequency=659 length=700ms;
 :delay 700ms;

 :beep frequency=784 length=500ms;
 :delay 500ms;

 :beep frequency=523 length=200ms;
 :delay 1000ms;

 :beep frequency=523 length=200ms;
```

```
:delay 1000ms;

:beep frequency=523 length=200ms;
:delay 1000ms;

:beep frequency=659 length=700ms;
:delay 700ms;

:beep frequency=784 length=500ms;
:delay 800ms;

:beep frequency=784 length=400ms;
:delay 400ms;

:beep frequency=884 length=200ms;
:delay 200ms;

:beep frequency=784 length=200ms;
:delay 200ms;

:beep frequency=687 length=200ms;
:delay 200ms;

:beep frequency=659 length=200ms;
:delay 200ms;

:beep frequency=579 length=200ms;
:delay 200ms;

:beep frequency=519 length=400ms;
:delay 400ms;
```