Opensips V2.2 中文手册

申　明

1、 本文收集于互联网，由 ligen119 整理，本人不拥有版权。

2、 本文内容的主要来源：csdn 流水夜月的博客，博客地址：

https://me.csdn.net/blog/long_longago 。

3、 想到了再加上。

目　录

一、 OpenSIPS 安装部署

1.1. 下载 OpenSIPS v2.2

有多中方法可以下载 opensips

1.1.1. 从网页获取分支文件

最新的 2.2 版本的 tar 包可以直接从项目网址下载

```
http://opensips.org/pub/opensips/2.2.x/
```

1.1.2. 从 sourceForge 获取 tar 包

从项目网址上获取的分支文件也可以在 sourceForge 下载。建议使用这种方式，下载更快

```
https://sourceforge.net/projects/opensips/files/OpenSIPS/
```

1.2. 编译和安装 opensips

1.2.1. 视频教程

opensips 团队已经举办了一次网络研讨会，将指导您快速安装 opensips（下载源码，编译，部署等）和 opensips 控制面板（安装和配置用户）。并向您展示怎么在几分钟内获取一个功能齐全的平台。如果你觉得基于文本的视频教程更容易，请试试访问 opensips 安装研讨会页面并下载研讨会记录

1.2.2. 编译

切换到 opensips 源码更目录，在这个目录运行如下命令

```
make all
```

那么 opensip 以及配置的模块都会被编译

1.2.2.1. 配置编译项

opensips 有各种能力相关的的编译时选项，例如你可以启用内存条是分配器或者启用 tls(默认情况下禁用)等等

为了更改这些编译时选项，你需要使用 menuconfig 工具，因为 menuconfig 工具依赖于 curses，在使用之前需要安装 nurses 开发库。在基于 debian 的系统中，应该运行如下命令

```
apt-get install libncurses5-dev
```

之后在 opensips 源码根目录，运行如下命令

```
make menuconfig
```

导航到 配置编译选项菜单，你只需要使用 arrows 键（UP + DOWN）去浏览遨游上午选项（在控制台底部简要介绍了他们）。启用或者禁用他们是通过空格键完成的。完成配置后，可以使用 q 键返回然后点击“保存配置”

修改编译选项后，你应该重新编译安装 opensips

1.2.2.2. 配置具有外部依赖的编译项

有些 opensips 模块默认不编译，因为他们需要一些外呼依赖，而这些依赖并非来自你的系统。因此当你下载并安装源码是，这些模块需要特别关注。比如 DB_MYSQL(依赖于 mysql 开发库)，JSON（依赖于外部的 JSON_parse）

为了使用这些模块，你需要使用 menuconfig 工具

在 opensips 源代码根目录，运行’make menuconfig’，并且切到’configure

excluded modules’，在这里你可以看到所有默认不开启的模块列表，并且模块功能的简明描述位于控制台底部。

使用空格启用或者禁用某个模块。一旦你选择了你需要的模块，按“q”返回上一个选项，然后点击“save changes”。工具将会展示你启用的模块，并展示成功编译这些模块锁需要的依赖。

更改之后需要重新编译安装。

1.2.3. 安装

为了安装 opensips，切到源代码根目录，运行如下指令

```
make install
```

默认情况下 opensips 将会安装在根目录\

1.2.3.1. 减少编译时间

为了减少 opensips 的编译时间，可以使用 FASTER 变量。改功能利用‘-jNR_OF_CORES’多核并行编译所有模块。鉴于此，该方法会只能用大量的资源和进程（不大于内核的数目）当然这个变量会减少大部分编译输出

例如在 4 核机器上安装 opensips，在根目录运行如下指令

```
FASTER=1 make -j4 install
```

1.2.3.2. 配置安装路径

因为各种原因（在同一台机器上安装两个不同版本的 opensips），有的时候我们需要更改 opensips 安装路径。为了达到该目的，你需要使用 menuconfig 工具运行“make menuconfig”并且切换到“configure install prefix”，然后输入你自定义

的 opensips 安装路径，然后向下导航到“save changes”并点击确定。之后你可以再进行“make install"，那么 opensisp 将会在你指定的目录上安装部署。

1.3. 部署数据库

1.3.1. 配置 db 证书

到[install_path]/etc/opensips/目录，打开文件 opensipsctlrc

并关注下面几行

- DBENGINE=
  - 当前可用的选项 MYSQL,PGSQL,ORACLE,DB_BERKELEY,或者 DBTEXT
- DBHOST=
  - 输入 DB engine 的主机
- DBPORT=
  - 输入 DB engine 的端口
- DBNAME=
  - 要创建的数据库的名字
- DBRWUSER=
  - 在数据库中为 opensips 创建的具有读写权限的用户名
- DBRWPW=
  - DBRWUSER 的密码
- DBROOTYSER=
  - 用于创建数据库，表和 DBRWUSER 的用户

1.3.2. 创建数据库

为了创建你上面配置的数据库，你需要运行

```
[Install_Path]/sbin/opensipsdbctl create
```

当 opensipsdbctl 工具提醒时，请输入你的 DBROOTUSER 密码

如果你想创建一个不同于默认 DBNAME 的数据库，你可以运行如下命令

```
[Install_Path]/sbin/opensipsdbctl create my_custom_db_name
```

opensipsdbctl 还可以用于执行备份，还原等等。如果你想要查看 opensipsdbctl 的功能和帮助手册，可以不带参数的运行

```
[Install_Path]/sbin/opensipsdbctl
```

二、 OpenSIPS 配置

2.1. RC 文件

opensipsctlrc 包含 opensipsctl,opensipsdbctl 和 osipsconsole 工具的所有配置选项。

安装后文件位于

```
[INSTALL_PATH]/etc/opensips/opensipsctlrc
```

这个文件包含如下配置项：和数据库的交互，和 opensips 的运行时交互，通过 opensipsctl 和 osipsconsole 工具的控制选项。

最相关或最常用的配置如下：

- SIP_DOMAIN - opensips 代理的 sip 域，用于系统添加一个新用户（通过 opensipsctl 工具）
- DB parameters - 包含创建数据库时 opensipsdbctl 使用的 db 认证信息，以及插入配置信息时 opensipsctl 需要的信息

- CTLENGINE - 当在 opensipsctl 工具运行 mi 命令时，fifo engine 将使用 transport。选项包括 FIFO,XMLRPC 和 UDP
- OSIPS_FIFO - 当前 opensips 实例的 opensips FIFO 路径。加入你再同一个服务器上部署了多个 opensips 实例，你应该修改他。
- STORE_PLAINTEXT_PW - 当通过“opensipsctl add username”明星添加用户时，password 以明文存储于在 db，或者仅仅包含提供的 password 的 hash 版本。

2.2. 配置文件

opensips 配置文件包含所有的 opensips 核心和模块的所有配置参数，以及用于路由 sip 流量的实际路由逻辑。

安装后，默认的配置文件路径:

```
[INSTALL_PATH]/etc/opensips/opensips.cfg
```

配置文件是基于文本的，用 opensips 自定义语音编写，非常类似于 c 语音。你讲找到各种变量（拥有不同的作用域，在手册中进一步说明）你也可以使用像 if/while/switch 这样的经典构造，也可以调用带参数的 sub-routines，这样脚本对于具有一些 sip 和变成技能的人非常易于阅读。

```
假如你修改了配置文件，必须重启 opensips 才能生效
```

由于每次修改 opensips 配置文件都需要重启 opensips，因此根据 opensips 语法确保所做的更改都是正确的非常重要。

你可以通过运行来检查 opensips 配置文件的有效性。

```
[INSTALL_PATH]/sbin/opensips -C [PATH_TO_CFG]
```

检查配置文件的有效性时，如果 cfg 是正确的返回 0，

如果配置文件包含任何错误，会在控制台展示，并返回-1

2.3. 产生配置文件

使用 menuconfig 工具可以生成 opensips 配置文件。因为图形界面是给予 ncurses 的，所以请首先确保安装了 ncurses 开发库（通常是 libncurses5-dev）。

2.3.1. 使用 menuconfig 工具

menuconfig 可以从 opensips 源码运行，也可以在安装后，再安装路径运行。

- 在源码，你可以运行

```
make menuconfig
```

- 安装之后，可以在安装路径直接运行 menuconfig

```
[install_path]/sbin/osipsconfig
```

进入 menuconfig 工具后，导航到“generate opensips script”选项，然后选择所需要的脚本类型。选择脚本类型后，你可以配置该脚本的各个可用选项（如下所述）。使用空格可以为每个脚本启动某些选项。配置好所需选项后，可以按"q"键回退到上个菜单 and 点击“save changes”。然后可以根据你的配置生成 opensips 脚本。最后，图形界面将提供新生成的配置文件的路径。

```
/usr/local/opensips_proxy_1.12/etc/opensips/opensips_residential_2013-5-21_12:39:48.cfg
```

2.3.2. 配置文件的类型

到目前为止，opensips1.12menuconfig 自动脚本生成器支持三种类型的脚本。一下是脚本类型以及每隔脚本的可选项。

- Residential Script
  - ENABLE_TCP: opensips 将在 tcp 上监听 sip 请求
  - ENABLE_TLS: opensips 将在 tcp 上监听 sip 请求
  - USE_ALIASES: opensips 将允许使用 sip 用户的别名
  - USE_AUTH: opensips 将认证 invite 和 register 请求
  - USE_DBACC: opensips 为所有的呼叫在 db 上存储 ACC 实体
  - USE_DBUSERLOC: opensips 将在数据库永久存储用户落地实体
  - USE_DIALOG: opensips 将跟踪活跃的对话
  - USE_MULTIDOMAIN: opensips 支持订阅者的多个域
  - USE_NAT: opensips 将通过修复 sip 消息并使用 rtpproxy 来应对 nat
  - USE_PRESENCE: opensips 将充当 presence 服务器
  - USE_DIALPLAN: opensips 使用 dialplana 转换本地号码
  - VM_DIVERSION: OpenSIPS will redirect to VM calls not reaching the subscribers
  - HAVE_INBOUND_PSTN: opensips 接收来自 pstn 的呼叫(带有静态 ip 认证)
  - HAVE_OUTBOUND_PSTN: opensips 向 pstn 发送数据拨号（带有静态 ip 认证）
  - USE_DR_PSTN: opensips 使用动态路由支持（LCR: dynamic routing support）和 pstn 互连
- Trunking Script

- ENABLE_TCP: opensips 将在 tcp 上监听 sip 请求
- ENABLE_TLS: opensips 将在 tcp 上监听 sip 请求
- USE_DBACC: opensips 为所有的呼叫在 db 上存储 ACC 实体
- USE_DIALPLAN: opensips 使用 dialplana 转换本地号码
- USE_DIALOG: opensips 将跟踪活跃的对话
- DO_CALL_LIMITATION: opensips 将限制每隔终极的并行呼叫数

- Load-Balancer Script
  - ENABLE_TCP: opensips 将在 tcp 上监听 sip 请求
  - ENABLE_TLS: opensips 将在 tcp 上监听 sip 请求
  - USE_DBACC: opensips 为所有的呼叫在 db 上存储 ACC 实体
  - USE_DISPATCHER: opensips 将使用 DISPATCHER 而不是负载均衡来分配流量
  - DISALBE_PINGING:

2.3.3. 签名产生的脚本的编辑

使用 menuconfig 产生 opensips 脚本后，你需要使用你喜欢的编辑器打开脚本，并且查看脚本中所有的“#CUSTOMIZE ME”注释。这些注释标记了用户需要关注的地方，并且通常是指自定义 opensips 监听地址或设置正确的数据库 url.
在进行适当的“#CUSTOMIZE ME”更改后，你可以保存脚本并试用。

2.4. 脚本格式

opensips 配置脚本主要有三个逻辑部分：全局参数，模块部分，路由逻辑。

2.4.1. 全局参数

通常，在第一部分定义全局变量-这些全局或者核心参数影响 opendips 核心和某些模块。

这些全局参数提供了：配置网络监听，可用的传输协议，forking(和线程数)，日志记录和其他全局的内容

例如：

```
disable_tcp = yes
listen = udp:192.268.2.20:5060
listen = udp:192.268.2.20:5070
fork = yes
children = 4
log_stderror = no
```

2.4.2. 模块部分

对于 opensips 模块，要加载的模块通过使用特定的 loadModule 加载。模块由名称和路径(到.so 文件)共同决定。如果没有提供路径（仅仅提供模块的名字），将使用默认路径（如果不是在编译时配置其他路径，那么默认路径是 /usr/lib/opensips/modules）。如果使用不同的路径，要么路径和名字结合起来使用（以获取每个模块的控制），要么可以通过全局参数 mpath 全局配置（对于所有模块生效）。

加载模块后，可以用 modparam 指令设置模块的参数-罗列每个模块的可用参数列表，参数的类型(integer 或者 string)可以在文档中找到。

例如

```
loadmodule "modules/mi_datagram/mi_datagram.so"
modparam("mi_datagram", "socket_name", "udp:127.0.0.1:4343")
modparam("mi_datagram", "children_count", 3)
mpath="/usr/local/opensips_proxy/lib/modules"
loadmodule "mi_datagram.so"
modparam("mi_datagram", "socket_name", "udp:127.0.0.1:4343")
modparam("mi_datagram", "children_count", 3)
loadmodule "mi_fifo.so"
modparam("mi_fifo", "fifo_name", "/tmp/opensips_fifo")
```

2.4.3. 路由逻辑

路由逻辑实际上是路由（script routes：包含 opensips 路由 sip 流量的逻辑）的总称。和 sip 路由相关的 opensips 行为的描述是通过这些 routes 实现的。

有两种不同的路由：

- top routes - 当某些事件（如 sip 请求，sip 应答，事务失败等）发生时，路由由 opensips 直接触发。
- sub-routes - 被其他 routes 在脚本中触发或者使用的 routes

  什么是现有的 top-routes，什么时候被触发，处理什么 sip 消息，什么 sip 操作被允许以及其他被记录在路由类型部分。

  sub-routes 有自己的名字并且其他 route（top or sub）在脚本中通过他们的名字调用 sub-routes。sub-routes 可能需要参数(调用时)或者返回一个数字代码（避免返回 0-因为这会打断你的整个脚本。sub-routes 类似于边冲中的函数或者过程。请参数 route 指令的说明）

2.4.4. 全局参数

本节列出了 opensips core 为脚本使用而导出的所有参数（在 opensips.cfg 中使用的）

2.4.5. 核心关键字 Core keywords

关键字特指在 sip 消息中用到的主要用于'if'表达式。

2.4.5.1. af

收到的 sip 消息的地址族。如果通过 ipv4 获取消息是 INET,如果通过 ipv6 获取消息则为 INET6。

使用用例

```
if(af==INET6) {
        log("Message received over IPv6 link\n");
};
```

2.4.5.2. dst_ip

接收 sip 消息的本地接口 ip。当代理监听很多网络接口时，可以检测是哪个地址接收数据包。

使用用例

```
if(dst_ip==127.0.0.1) {
    log("message received on loopback interface\n");
};
```

2.4.5.3. dst_port

接收 sip 消息的本地端口。当 opensips 正在监听多个端口时，可以检测到是哪个 端口接收数据包。

使用用例

```
if(dst_port==5061)
  {
        log("message was received on port 5061\n");
  };
```

2.4.5.4. from_uri

这个脚本变量对应"from"头信息的 URI 引用。可用于测试“from”头 URI。

使用用例

```
    if(is_method("INVITE") && from_uri=~".*@opensips.org")
    {
          log("the caller is from opensips.org\n");
    };
```

2.4.5.5. method

这个变量指消息的 sip 方法。

使用用例

```
    if(method=="REGISTER")
    {
        log("this SIP request is a REGISTER message\n");
    };
```

2.4.5.6. msg:len

这个变量是消息大小的引用。常用于’if’结构中以测试消息的大小。

使用用例

```
if(msg:len>2048)
{
        sl_send_reply("413", "message too large");
        exit;
};
```

2.4.5.7. $retcode

指代最近执行的函数的返回值（类似于 bash 的?——如果你愿意也可以用?--如果你愿意也可以用?——如果你愿意也可以用？在 opensips 中‘retcode’和'retcode’和'*retcode*’和'?'均支持）。如果在呼叫路由后进行测试，它就是路由返回的值。

使用用例

```
route {
        route(1);
        if($retcode==1)
        {
                log("The request is an INVITE\n");
        };
}


route[1] {
        if(is_method("INVITE"))
                return(1);
        return(2);
}
```

2.4.5.8. proto

该变量用于获取 sip 消息的传输协议。

使用用例

```
if(proto==UDP)
{
```

```
        log("SIP message received over UDP\n");
};
```

2.4.5.9. status

如果在 onreply_route 中使用，这个变量指响应的状态码。如果是在一个标准的路由块中使用，该变量指当前发起的请求的最后一个相应的状态。

使用用例

```
if(status=="200")
{
        log("this is a 200 OK reply\n");
};
```

2.4.5.10.src_ip

指代 sip 消息的源 ip 地址。

使用用例

```
if(src_ip==127.0.0.1)
{
        log("the message was sent from localhost!\n");
};
```

2.4.5.11.src_port

指代 sip 消息的源端口（消息由上一跳送到哪个端口）

使用用例

```
if(src_port==5061)
{
        log("message sent from port 5061\n");
}
```

2.4.5.12.to_uri

用于测试 TO 头的 URI 值。

使用用例

```
if(to_uri=~"sip:.+@opensips.org")
{
    log("this is a request for opensips.org users\n");
};
```

2.4.5.13.uri

用于测试请求 URI 的值

使用用例

```
if(uri=~"sip:.+@opensips.org")
{
    log("this is a request for opensips.org users\n");
};
```

2.4.6. 核心表达式 Core Values

常用于“if”表达式，以检查 core keywords。

2.4.6.1. INET

该 keyword 用于测试 sip 包是否可以通过 IPV4 连接接收。

使用用例

```
if(af==INET)
{
    log("the SIP message was received over IPv4\n");
};
```

2.4.6.2. INET6

该 keyword 用于测试 sip 包是否可以通过 IPV6 连接接收。

使用用例

```
if(af==INET6)
{
    log("the SIP message was received over IPv6\n");
};
```

2.4.6.3. TCP

该 keyword 用于测试传输协议。检查是否通过 TCP 接收 sip 包。

使用用例

```
if(proto==TCP)
{
    log("the SIP message was received over TCP\n");
};
```

2.4.6.4. UDP

该 keyword 用于测试传输协议，检测是否通过 UDP 接收 sip 包

使用用例

```
if(proto==UDP)
{
    log("the SIP message was received over TCP\n");
};
```

2.4.6.5. max_len

该 keyword 指代 DUP 包的大小上限。可以用于测试消息的大小。

使用用例

```
    if(msg:len>max_len)
```

```
    {
        sl_send_reply("413", "message too large to be forwarded over UDP without
fragmentation");
        exit;
    }
```

2.4.6.6. myself

用于罗列本地 ip 地址，主机名以及在 opensips 配置文件中设置的别名。列表包含 opensips 提供的域。

该变量用于测试 URI 的 host 是否在列表中。用于决定消息是在本地处理还是发送到另外一台服务器。

请参阅“alias”以添加 hostname,ip 地址和 alias 到列表中。

使用用例

```
    if(uri==myself) {
        log("the request is for local processing\n");
    };
```

2.4.6.7. null

用于赋值以便重置每个脚本变量的值或者删除一个 avp.

使用用例

```
    $avp(i:12) = null;
    $var(x) = null;
```

2.4.7. 核心参数 Core parameters

可以在配置文件中设置全局变量。可接受的值有：基于实际参数的字符串，数字以及“yes”或者“no”。如果你想"yes"和"no"作为字符串使用，请将其用双引号括起来。

2.4.7.1. abort_on_assert

默认值:false

只有启用 assert 才会生效。设置 assert 为 true 那么如果一个脚本断言失败，立即关闭 opensips。

使用用例

```
abort_on_assert = true
```

2.4.7.2. advertised_address

它可以是一个字符串或者 IP 地址，表示 via 头或者其他目标卷对应的地址。如果为空或者未设置(默认值)，那么使用发送请求的套接字地址。

```
WARNING:
 - don't set it unless you know what you are doing (e.g. nat traversal)
 - you can set anything here, no check is made (e.g. foo.bar will be
   accepted even if foo.bar doesn't exist)
```

使用用例

```
advertised_address="opensips.org"
```

注意：除了这种全局方法，你也可以在每一个接口方法中定义一个建议 IP 和端口（参阅"listen"）。当在每个接口中定义建议值时，

2.4.7.3. advertised_port

标识 via 头或者其他目标卷对应的地址。如果为空或者未设置（默认值），那么使用发送消息的端口。一些类似’advertised_address’的告警信息。

使用用例

```
advertised_port=5080
```

2.4.7.4. alias

用于为服务器设置主机别名。可以多次设置，每个值将会增加到一个列表：当检查'myself'时匹配主机名。

使用用例

```
alias=udp:other.domain.com:5060
alias=tcp:another.domain.com:5060
```

2.4.7.5. auto_aliases

这个参数控制在修复监听套接字期间是否自动发现并添加别名。自动发现的别名来自 dns 查询结果(如果监听是一个名字而不是 ip)或者监听 ip 上的反向 dns 查询结果。

由于兼容性的问题，默认值为'on'.

使用用例

```
auto_aliases=no
auto_aliases=0
```

2.4.7.6. bin_listen(Removed in OpenSIPS 2.2)

替换为 proto_bin 模块的 bin_port 模块参数（因为将 bin 实现从核心迁移到单独模块）

2.4.7.7. bin_children(Removed in OpenSIPS 2.2)

已过时，直接删除即可。

2.4.7.8. cfg_file

返回相应的 opensips 配置文件（一般应用于包含了多个配置文件的情形）

2.4.7.9. cfg_line

返回 OpenSIPS 配置文件中的相应行

2.4.7.10. check_via

检查回复中最顶层的 via 地址是否为本地地址，默认值为 0（禁用检查）。

使用用例

```
check_via=1
```

2.4.7.11. children

为每个 udp 或者 sctp 接口定义的创建工作进程数。默认值是 8。

使用用例

```
children=16
```

注意：如果在某个接口定义一个不同的 children，则可以覆盖全局设置（适用于所有 udp/sctp 接口）。因此你可以为每隔接口定义不同数量的 children.

2.4.7.12. chroot

取值必须是系统的真实路径。如果设置，openSIPS 将会据此设置根目录 (chroot)。

使用用例

```
chroot=/other/fakeroot
```

2.4.7.13. debug_mode

开始 debug_mode 是 debug openSIPS 最快的方式。这个选项会自动强制如下选项使能:

- 留在前台（不要离开控制台）
- 设置 logging level 到 4（debug）
- 设置 logging 为标准错误
- 允许 dump core
- 设置 UDP 进程数到 2
- 设置 TCP 进程数到 2

  默认值是 false/0(禁用)。

  注意：该选项会覆盖其他单个选项，如 前台模式，日志级别，子进程数。

  这个选项是作为 fork 选项的替代引入的。

2.4.7.14. db_version_table

在 DB API 中使用的 table version 的名字，用于检查使用的 tables 的版本。

默认值是’version’。

使用用例

```
db_version_table="version_1_8"
```

2.4.7.15. db_default_url

如果没有在每隔模块中指定 URL,那么将使用默认的 DB URL。默认值是 NULL(未定义)

使用用例

```
db_default_url="mysql://opensips:opensipsrw@localhost/opensips"
```

2.4.7.16. db_max_async_connections

从单个 opensips 工作模块针对每个独立的 sql 后端可以打开的 tcp 连接最大值。默认值是 10。

各个终端由如下元素决定 DB URL

```
[ scheme, user, pass, host, port, database ]
```

使用用例

```
db_max_async_connections=220
```

2.4.7.17. debug(Removed in OpenSIPS 2.2)

由 log_level 参数取代。

2.4.7.18. disable_503_translation

如果设置为’yes’，opensips 将不会将加收的 503 响应转化为 500 响应。（RFC 3261 明确指出代理不应该响应一个 503 响应，而是必须将其转化为 500）

默认值是’no’（转化）

2.4.7.19. disable_core_dump

可以设置为’yes’或者’no’。默认情况下，core dump 没有限制或者设置为一个足够高的值。将此配置变量设置为’yes’以禁用 core dump-ing(设置 core 限制为 0)。

默认值是’no’。

使用用例

```
disable_core_dump=yes
```

2.4.7.20. disable_dns_blacklist

当配置 failover 时，dns 解析器可以自动将故障目标存储在临时黑名单中。这将阻止 openSIPS 发送请求到已知会失败的目标。因此黑名单可以用于 dns 解析器的内存。

DNS 解析器创建的临时黑名单为“dns”，并且会默认选择使用（不需要使用 use_blacklist()）。该列表的时间规则为 4 分钟-你可以在编译阶段通过修改 resolve.c 解决。

2.4.7.21. disable_dns_failover

可以设置为’yes’或者’no’，默认基于 dns 的 failover 是开启的。设置该参数为‘yes’会禁用基于 DNS 的 failover。这是一个全局配置，影响核心以及其他模块。

默认值是’no’。

使用用例

```
disable_dns_failover=yes
```

2.4.7.22. disable_stateless_fwd

可以设置为’yes’或者’no’。这个参数控制无状态应答的处理

- yes - 如果 stateless fwd 函数(如 forward)在脚本中没有使用，则删除无状态回复。
- no - 转发无状态回复

  默认值是’yes’。

2.4.7.23. dns

此参数控制 SIP 服务器是否应尝试在 dns 中查找自己的域名。如果此参数设置为 yes 且域名不在 dns 中，将打印一个 warning 并且’received='域添加到 via header。

默认值是 no。

2.4.7.24. dns_retr_time

重试 dns 请求的时间间隔。默认值是系统指定的，依赖于配置文件’/etc/resolv.conf’的内容（通常是 4s）。

使用用例

```
dns_retr_no=3
```

2.4.7.25. dns_servers_no

使用’/etc/resolv.conf’中定义的 dns 服务器的数目。默认值是使用配置文件中定义的所以 dns 服务器。

使用用例

```
dns_servers_no=2
```

2.4.7.26. dns_try_ipv6

可以设置为’yes’或者’no’。如果设置为’yes’并且 dns 查询失败，那么将使用 ipv6（AAAA record）重试。默认值是’no’。

使用用例

```
dns_try_ipv6=yes
```

2.4.7.27. dns_try_napt

当为 sip 请求做基于 DNS 的路由时禁用 NAPTR（Naming Authority Pointer 名称权威指针）查询。如果禁用，DNS 查找将从 SRV 查询开始。可以设置为’yes’或者’no’。默认情况下是开启的，值为’yes’。

使用用例

```
dns_try_naptr=no
```

2.4.7.28. dns_use_search_list

可以设置为’yes’或者’no’,如果设置为’no’，’/etc/resolv.conf’的查询列表将被忽略（=》最少查找=》放弃最快）。默认值为’yes’。

提示：即便你没有定义查询列表，设置该选项为’no’，仍然"faster"，因为一个

空的查询列表实际上就是查询""（因此即使搜索列表为空/缺失，仍然有两个 dns 查询，例如 foo+'.'和 foo+""+'.' ）。

使用用例

```
dns_use_search_list=no
```

2.4.7.29. dst_blacklist

定义一个静态的（仅可读）ip/目标 黑名单。脚本可以在运行时选择这些列表基于 ip,协议，端口等过滤传出请求。

主要目的为阻止发送请求到关键 ip(如 GW)，或者避免发送到已知不可用的目标(临时或者永久的)。

使用用例

```
    # filter out requests going to ips of my gws
    dst_blacklist = gw:{( tcp , 192.168.2.200 , 5060 , "" ),( any , 192.168.2.201 , 0 , "" )}
    # block requests going to "evil" networks
    dst_blacklist = net_filter:{ ( any , 192.168.1.120/255.255.255.0 , 0 , "" )}
    # block message requests with nasty words
    dst_blacklist = msg_filter:{ ( any , 192.168.20.0/255.255.255.0 , 0 ,
"MESSAGE*ugly_word" )}
    # block requests not going to a specific subnet
    dst_blacklist = net_filter2:{ !( any , 192.268.30.0/255.255.255.0 , 0 , "" )}
```

每个规则定义如下

- protocol:TCP,UDP,TLS 或者所有协议即"any"
- 端口: 数字 0 标识所有端口
- ip/mark
- test pattr - 满足(查看"man 3 fnmatch")应用于传出请求缓存 (first_line+hdrs+body)。

2.4.7.30. enable_asserts

默认值：false

如果需要弃用 assert 脚本语句，设置为‘true’

使用用例

```
enable_asserts = true
```

2.4.7.31. event_pkg_threshold

E_CORE_PKG_THRESHOLD 事件引入的线程百分比阈值, 警告可用内存减少.

接受 0 到 100 的整数。

默认值是 0（禁用事件）

使用用例

```
event_pkg_threshold = 90
```

2.4.7.32. event_shm_threshold

E_CORE_SHM_THRESHOLD 事件引入的线程百分比阈值，讲稿可用内存减少。家饿瘦 0 到 100 的整数值

默认值是 0（禁用事件）

使用用例

```
event_shm_threshold = 90
```

2.4.7.33. exec_dns_threshold

一个 dns 查询将持续的最大微秒数。超过设置值的任何查询将触发告警并记录到日志工具。

默认值是 0（日志禁用）

使用用例

```
exec_dns_threshold = 60000
```

2.4.7.34. exec_msg_threshold

处理 sip 消息持续的最大微秒数。超过设定值的任何消息将触发告警并记录到日志工具上。除了消息和处理事件，最耗时的函数的事件也会记录。

默认值是 0（禁用日志）

使用用例

```
exec_msg_threshold = 60000
```

2.4.7.35. fork(Removed in OpenSIPS 2.2)

被参数 debug_mode 取代。

2.4.7.36. group gid（Removed in OpenSIPS 2.2）

使用 -u 命令行参数替代。

2.4.7.37. include_file

可以从外部路由块调用其他的 routes/blocks 或者简单的执行更多的函数。文件可以是绝对路径或者相对路径。如果不是绝对路径，首先在当前目录阐释去定位它。如果本地定位失败，第二次尝试包含该文件的目录。如果定位失败将抛出异常。

使用用例

```
include_file "proxy_regs.cfg"
```

2.4.7.38.import_file

类似于"include_file"，但是如果文件未找到不抛出异常。

使用用例

```
import_file "proxy_regs.cfg"
```

2.4.7.39.listen

设置 sip 服务器应该监听的网络地址。它的语法是 protocol: address[:port]。详解如下

- protocol: 应该是配置文件记载的传输协议中的一种(如 udp,tcp,tls)
- address: 可以是 ip 地址， 主机名， 网络接口 id，或者 '*'通配符（opensips 可以监听协议所有可能的接口）
- port: 可选的，用于监听的端口-如果不存在使用传输模块的默认端口。

  该参数可以在一个配置文件里设置多次，服务器将监听指定的所有地址。

  listen 定义可以接受一下几个可选参数:

- 仅仅为某个接口配置一个 advertised ip 和端口，语法"AS 11.22.33.44:5060"
- 仅仅为某个接口设置一个不同的 children(仅仅适用于 UDP,SCTP,和 HEP_UDP 接口)，这将覆盖全局的’children’参数。语法 “use_children 5”。

  注意上面的参数仅仅影响配置他们的接口；如果没有给定的接口中没有定义，那么将使用全局设置。

  使用用例

```
listen = udp:*
listen = udp:eth1
listen = tcp:eth1:5062
```

```
    listen = tls:localhost:5061
    listen = hep_udp:10.10.10.10:5064
    listen = ws:127.0.0.1:5060 use_children 5
    listen = sctp:127.0.0.1:5060 as 99.88.44.33:5060 use_children 3
```

在启动时，openSIPS 会报道它正在监听的所有接口。即便你仅仅指定了 UDP 接口，TCP 引擎进程也会创建。

2.4.7.40.log_facility

如果 opensips 记录日志到 syslog,你可以使用 facility 记录日志。当你想记录日志到不通文件时，非常有效。更多详情可以查看 man syslog

默认值是 LOG_DAEMON

使用用例

```
log_facility=LOG_LOCAL0
```

2.4.7.41.log_level

设置日志等级。值越高 opensips 打印越多的信息。

使用用例

```
log_level=1 -- print only important messages (like errors or more critical situations)
    - recommended for running proxy as daemon

log_level=4 -- print a lot of debug messages - use it only when doing debugging sessions
```

取值如下

```
- -3 - Alert level
- -2 - Critical level
- -1 - Error level
- 1 - Warning level
- 2 - Notice level
- 3 - Info level
- 4 - Debug level
```

参数’log_level’经常结合参数’log_stderror’一起使用。

'log_level’的值也可以通过 log_level core 函数或者$log_level 脚本变量进行动态获取设置。

2.4.7.42.log_name

设置打印在日志中的 id。取值必须是字符串并且仅仅在 daemon 模式下才会生效(fork=yes)。默认值是 argv[0]。

使用用例

```
log_name="osips-5070"
```

2.4.7.43.log_stderror

使用该参数你可以使 opensips 将日志写入标准错误。可能的取值如下:

- “yes” - write the messages to standard error
- “no” - write the messages to syslog

  默认值是"no"。

  使用用例

```
log_stderror=yes
```

2.4.7.44.max_while_loops

while 循环的最大循环次数。作为避免配置文件执行死循环的一个保护措施。默认值是 100

2.4.7.45. maxbuffer

自动探测过程中接收的 udp 消息缓冲区大小最大值。默认值是 262144。

使用用例

```
maxbuffer=65536
```

2.4.7.46. mem-group

定义一组模块（根据名字）以获取内存统计信息。opensips 将提供每组内存信息-分配的片段数，已使用内存和实际使用的内存（包括内存管理的开销）。如果你想监控某个模块（或者模块组）的内存使用情况会非常有效。

为了使用该功能项你需要运行’make generate-mem-stats’并定义变量’SHM_EXTRA_STATS’进行编译。

使用用例

```
    mem-group = "interest": "core" "tm"
    mem-group = "runtime": "dialog" "usrloc" "tm"
```

对于上面的实例，生成的统计信息将命名为：shmem_group_interest:fragments, shmem_group_interest:memory_used, shmem_group_interest:real_used.
可以定义多个组，但是组名不能重名。

如果你想生成默认组的统计信息（某个模块以外的所有模块），则必须定义变量 SHM_SHOW_DEFAULT_GROUP 进行编译。

2.4.7.47. mem_warning

What is memory fragmentation? Memory fragmentation is when the sum of the available space is a memory heap is large enough to satisfy a memory allocation request but the size of any individual fragment (or contiguous fragments) is too small to satisfy that memory allocation request.

默认值:off

仅仅在编译时启用 HP_MALLOC 时才生效。如果设置为"on",每次启动时，opensips 会尝试恢复停止/重启之前的 memory fragmentation 模式。如果没有找到先前运行的 pattern_file，则跳过内存告警，并且内存分配器像其他所有分配器一样初始分宜一块内存。

在处理大量流量时（多核机器上的数千 cps-核数越多，越有用） memory warming 非常有效。当在初始化的内存块上分配内存时，进程必须相互锁定。通过在启动时执行 fragmentation，opensip 在重启后的最初几分钟表现良好。碎片整理通常执行几秒 (e.g. ~5 seconds on an 8GB shm pool and 2.4Ghz CPU)，-在此期间不会处理流量。

使用用例

```
mem_warming = on
```

2.4.7.48. mem_warming_percentage

默认值：75

在重启时如果在编译时使能 mem_warming，opensips fragmentation 的百分比。如果使能 mem_warming 使能在启动时使用。

使用用例

```
mem_warming_percentage = 50
```

2.4.7.49.mem_warming_pattern_file

默认值：“CFG_DIR/mem_warming_pattern”

memory fragmentation pattern，如果使能 mem_warming，那么在启动时使用。

使用用例

```
mem_warming_pattern_file = "/var/tmp/my_memory_pattern"
```

2.4.7.50.memdump | mem_dump

内存状态信息的日志级别（运行时以及关闭时）。如果你想记录内存信息，其取值必须必’log_level’小。默认值: memdump=L_DUB(4)

使用用例

```
memdump=2
```

注意：如果设置 memlog 也需要设置 memdump.假如 memlog 和 memdump 要设置不同的值，那么需要先设置 memlog 后设置 memdump.

2.4.7.51.memlog | mem_log

内存 debug 信息的日志级别。如果想记录内存信息，其取值也必须小于参数’log_level’。默认值:=L_DBG(4)

使用用例

```
memlog=2
```

注意：如果设置了 memlog 那么 memdump 自动设置为相同的值。

2.4.7.52. mcast_loopback

可以设置为’yes’或者’no’。如果设置为’yes’，multicast datagram are sent over loopback。

使用用例

```
mcast_loopback=yes
```

2.4.7.53. mcast_ttl

Set the value for multicast ttl. Default value is OS specific (usually 1).

使用用例

```
mcast_ttl=32
```

2.4.7.54. mhomed

Set the server to try to locate outbound interface on multihomed host. By default is not (0) - it is rather time consuming.

使用用例

```
mhomed=1
```

2.4.7.55. mpath

设置 module 查询路径。用于简化 loadmodule 参数

使用用例

```
mpath="/usr/local/lib/opensips/modules"
loadmodule "mysql.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "sl.so"
loadmodule "tm.so"
```

2.4.7.56. open_file_limit

如果设置并且比当前打开文件限制高，那么 opensips 将尝试增加打开文件上限到指定值。注意：必须以 root 角色启动 opensips 才能增加限制超过系统限制（大多数系统，打开文件上限为 1024）

使用用例

```
open_files_limit=2048
```

2.4.7.57. poll_method

The poll method to be used by the I/O internal reactor - by default the best one for the current OS is selected. The available types are: poll, epoll_lt, sigio_rt, select, kqueue, /dev/poll.

Starting with version 2.2, epoll_et is deprecated and if it is used in the script, it will be automatically replaced by epoll_lt.

使用用例

```
poll_method=select
```

2.4.7.58. port

sip server 监听的端口。默认值是 5060

使用用例

```
port=5080
```

2.4.7.59. query_buffer_size

如果设置为大于 1 的值，数据库插入操作则不会逐个更新到数据库，要插入的行会保存在内存中，知道收集到多于 query_buffer_size 才会更新到数据库中。

使用 用例

```
query_buffer_size=5
```

2.4.7.60.query_flush_time

如果设置’query_buffer_size’为大于 1 的值，每隔 query_flush_time 将触发一个 trigger.保证内存中的插入行在内存中不会保持太长时间。

使用用例

```
query_flush_time=10
```

2.4.7.61.rev_dns

此参数控制 sip server 是否应尝试在 dns 中查找自己的地址。如果设置为 yes 并且 ip 地址不在 dns 中，会在 syslog 中打印一个警告信息并且添加字段 "received=“到 via header.

默认值是"no”。

2.4.7.62.server_header

当 opensips 作为 UAS 发送请求时，由 opensips 差生的 server header 的内容。默认为"OpenSips ((/))"。

使用用例

```
server_header="Server: My Company SIP Proxy"
```

请注意你必须添加标题名称"Server:",否则 OpenSIPS 将只写一个标题:

```
My Company SIP Proxy
```

2.4.7.63.server_signature

此参数控制所有本地产生的消息的"Server" header。

使用用例

```
server_signature=no
```

加入启用 server_signature 产生的 header 如下面所示

```
Server: OpenSIPS (0.9.5 (i386/linux))
```

2.4.7.64.shm_hash_split_percentage

仅在启用 hp_malloc 编译标志时才相关。该参数控制优化多少个 memroy buckerts.（例如设置为 2%，那么将优化前 81 个常用的 buckets）。默认值是 1.

2.4.7.65.shm_secondary_hash_size

仅在启用 HP_MALLOC 编译标志时才相关。该参数表示单个 bucket 的优化因子（e.g. setting it to 4 will cause the optimized buckets to be further split into 4）默认值为 8。

2.4.7.66.sip_warning

可以为 0 或者 1（默认值为 0）。在 opensips 产生的每个回复上添加'warning' header。The header contains several details that help troubleshooting using the network traffic dumps.

使用用例

```
sip_warning=0
```

2.4.7.67.tcp_children

Number of children processes to be created for reading from TCP connections. If no value is explicitly set, the same number of TCP children as UDP children (see “children” parameter) will be used.

使用用例

```
tcp_children=4
```

2.4.7.68.tcp_accept_aliases

默认值是 0（禁用）。如果启用，当检测到一个";alias" Via header 时将强制执行 RFC5923,并且在发送时重用为此类 sip 请求（source IP + Via prot + proto）打开的任意 TCP 连接（或者 tl,ws,wss）： 如 发送 sip 请求到或发送请求回同一个源时（source ip + Via port + proto）。毕竟,RFC5923 的最终目的是最小化 SIP 代码需要代开的 TLS 连接的数目，因为连接建立阶段的 cpu 开销很大。

RFC5923 连接重用（aliasing）机制上，opensips 在多个 sip 对话中的 tcp 连接也是持久的。这可以通过参数 tcp_connection_lifttime 进行控制。

Enabling the global tcp_accept_aliases parameter (RFC 5923) for end-user initiated connections (who are most likely grouped by one or more public IPs) is an open vector for call hijacking! In such platforms, we recommend using the force_tcp_alias() core function, in order to employ RFC 5923 behaviour only in conjunction with adjacent SIP proxies.

2.4.7.69.tcp_listen_backlog

backlog 参数定义 tcp 监听器挂起连接队列的最大长度。当队列满时，如果有连接请求到达，则客户端将受到带有 ECONNREFUSED 的错误，或者如果底层协议支持重传，则可以忽略改请求，以便稍后重连尝试成功。

默认值是 10.

2.4.7.70.tcp_connect_timeout

一个不间断的阻塞连接请求放弃的时间（单位毫秒）。默认值是 100ms.

使用用例

```
tcp_connect_timeout = 5
```

2.4.7.71.tcp_connection_lifetime

一个 tcp 会话的生命周期（单位秒）。openSips 将关闭持续 tcp_connection_lifttime 闲置的 tcp 连接默认值在 tcp_conn.h 中定义： # define DEFAULT_TCP_TIMEOUT 120.设置该值为 0，将快速关闭 tcp 连接。你还可以使用注册模块的 tcp_persistent_flag 参数将 tcp 的生命周期为注册的过期时间。

使用用例

```
tcp_connection_lifetime = 3600
```

2.4.7.72.tcp_max_connections

可接受的收到的活跃 tcp 连接最大值（即由远端初始化的）。一旦达到限制，将拒绝任何新来的 tcp 连接。默认值是 2018。对于 outgoing tcp 连接（opensips 初始化的）目前不受限制。

2.4.7.73.tcp_max_msg_time

sip 消息通过 tcp 连接完毕所用的最大秒数。如果一个 sip 包如果指定时间任然没有传输完毕，那么该连接将被丢弃。(either the connection is very overloaded and this leads to high fragmentation - or we are the victim of an ongoing attack where the attacker is sending the traffic very fragmented in order to decrease

our performance). 默认值是 4.

使用用例

```
tcp_max_msg_time = 8
```

2.4.7.74.tcp_no_new_conn_bflag

一个 branch flag 用于指示 opensips 传递请求是不要尝试建立新的 tcp 连接，而是仅仅复用现有的 tcp 连接（如果有的话）。如果没有可用的 tcp 连接，将发送一个一般发送错误。

该参数适用于 NAT 场景。在 nat 场景下创建 tcp 连接是没有意义的（如注册过程中创建的 TCP 连接丢失，那么在重新注册之前都没法连接该设备）。此外这也可以检测 NATed 注册用于核实断开了 tcp 连接，以便 opensips 可以将其注册无效。

使用用例

```
tcp_no_new_conn_bflag = TCP_NO_CONNECT
...
route {
    ...
    if (isflagset(DST_NATED) && $proto == "TCP")
        setbflag(TCP_NO_CONNECT);
    ...
    t_relay("0x02"); # no auto error reply
    $var(retcode) = $rc;
    if ($var(retcode) == -6) {
        #send error
        xlog("unable to send request to destination");
        send_reply("404", "Not Found");
        exit;
    } else if ($var(retcode) < 0) {
        sl_reply_error();
        exit;
    }
```

```
}
```

2.4.7.75.tcp_threshold

表示发送一个 tcp 请求预计会持续的最大微秒数。超过设定值时的任何 tcp 请求将触发在日志工具写入一条警告信息。

默认值是 0（日志关闭）

使用用例

```
tcp_threshold = 60000
```

2.4.7.76.tcp_keepalive

启用或者禁用 tcp keepalive(OS level).

默认开启。

使用用例

```
tcp_keepalive = 1
```

2.4.7.77.tcp_keepcount

关闭 tcp 连接前发送的 keepalives 的数目（仅限 linux）。

默认值是 0（不设置）。设置 tcp_keepcount 为任意值也会启用 tcp_keepalive。

使用用例

```
tcp_keepcount = 5
```

2.4.7.78.tcp_keepidle

如果连接空闲，opensips 发哦送 keepalive 的时间间隔（仅限 linux）。

默认值:0(不设置)。设置 tcp_keepilde 为任意值也会启用 tcp_keepalive。

使用用例

```
tcp_keepidle = 30
```

2.4.7.79.tcp_keepinterval

如果前一个失败，keepalive 探测的时间间隔（仅限 linux）。

默认值:0(不设置)。设置 tcp_keepinterval 为任意值也会启用 tcp_keepalive。

使用用例

```
tcp_keepinterval = 10
```

2.4.7.80.tls_ca_list

2.4.7.81.tls_certificate

2.4.7.82.tls_ciphers_list

2.4.7.83.tls_domain

2.4.7.84.tls_handshake_timeout

2.4.7.85.tls_log

2.4.7.86.tls_method

2.4.7.87.tls_port_no

2.4.7.88.tls_private_key

2.4.7.89.tls_require_certificate

2.4.7.90.tls_send_timeout

2.4.7.91.tls_verify

2.4.7.92.tos

TOS (Type Of Service) 用于发送 ip 包（支持 tcp 和 udp）。

使用用例

```
tos=IPTOS_LOWDELAY
tos=0x10
```

```
tos=IPTOS_RELIABILITY
```

2.4.7.93.user uid(在 opensips 2.2 弃用)

使用-u 命令行参数替代

2.4.7.94.user_agent_header

当 opensips 作为 uac 发送一个请求时产生的 User-Agent header 的内容。默认为"OpenSIPS((/))"

使用用例

```
user_agent_header="User-Agent: My Company SIP Proxy"
```

因为 opensips 不添加 因此必须包含 header 名 “User-Agent:” 否则你将得到如下错误的 header

```
My Company SIP Proxy
```

2.4.7.95.wdir

opensips 运行时使用的工作目录。当产生核心文件时，非常有用。

使用用例

```
wdir="/usr/local/opensips"
or
wdir=/usr/opensips_wd
```

2.4.7.96.xlog_buf_size

默认值: 4096

用于 opensips 日志打印的缓冲区大小。如果缓冲区太小，将打印溢出错误，并且跳过相关行。

2.4.7.97.xlog_force_color

默认值：false

仅在 xlog_force_color 时才相关。允许使用颜色转义序列，否则无效。

使用用例

```
xlog_force_color = true
```

2.4.7.98.xlog_default_level

默认值：-1

当没有设置 log_level 时，xlog 函数的日志等级默认值。

使用用例

```
xlog_default_level = 2 #L_NOTICE
```

2.4.8. 路由类型

opensips 路由逻辑使用几种类型的路由。每种路由类型由特定的事件触发并且允许你处理不同的类型的消息（请求或者回复）。

2.4.8.1. route

请求路由块：包含一系列针对 sip 请求的操作。

Triggered by:接收的来自网络的外呼请求。

Type:最初是无状态的，可以通过使用 TM 函数强制改为有状态的。

Default action:如果请求即没有被转发也没有回复，则路由最后将丢弃这个强求。

为每个 SIP 请求执行由"route {…}”"或者"route[0]{…}标识的 main"route’块。

执行主路由后会丢弃 sip 请求。为了发送回复或者抓饭请求，必须在路由块内调用明确的的操作。

使用用例

```
route {
        if(is_method("OPTIONS")) {
            # send reply for each options request
            sl_send_reply("200", "ok");
            exit();
        }
        route(1);
}
route[1] {
        # forward according to uri
        forward();
}
```

注意，如果从’branch_route[y]‘调用’route[X]’,那么在’route[X]'中值处理一个单独的分支而不是在 main route 中出现的所有分支。

2.4.8.2. branch_route

请求的分支路由块。包含一个 sip 请求的一个分支操作集合。

Triggered by:preparation a new branch (of a request); the branch is well formed, but not yet sent out.
Processing:the SIP request (with the branch particularities, like RURI, branch flags)
Type:stateful
Default action:如果 branch_route 没有 dropped(通过’drop()'语句)，branch_route 将自动发送出去。

It is executed only by TM module after it was armed via t_on_branch(“branch_route_index”).
使用用例

```
route {
      lookup("location");
```

```
        t_on_branch("1");
        if(!t_relay()) {
            sl_send_reply("500", "relaying failed");
        }
    }
    branch_route[1] {
        if(uri=~"10\.10\.10\.10") {
            # discard branches that go to 10.10.10.10
            drop();
        }
    }
```

2.4.8.3. failure_route

事务失败路由块。对于接受到所有否定答复的事务包含一系列操作。

2.5. 脚本转换

2.5.1. 前言

Transformation 一般用于变量（脚本变量，伪变量,AVPS,静态字符串）的函数，以获取特定值，原始变量的值不受影响。

在 opensips 脚本中使用不同变量的示例如下

```
# check if username in From header is equal with username in To header
if($fU==$tU) {
    ...
}

# r-uri username based processing
switch($ruri.user) {
    case "1234":
        ...
    break;
    case "5678":
        ...
    break;
    default:
```

```
        ...
}

# assign integer value to an AVP
$avp(i:11) = 1;

#assing string value to an AVP
$avp(i:22) = "opensips";

# write ruri in an AVP
$avp(i:33) = $ruri;

# concat "sip:" + From username + "@" + To domain in a script variable x
$var(x) = "sip:" + $fU +"@" + $td;
```

transformations 旨在便于访问变量的不同属性(字符串长度，变量的一部分，子串)或者变量的不同值（hexa,md5 编码，db 操作的 escape/unescape）。transformation 在'{'和'}'直接，后面跟变量名。当使用 transformations 时，变量名和 transformations 必须包含在'('和')'中间；

使用用例

```
# the length of From URI ($fu is pseudo-variable for From URI)

$(fu{s.len})
```

对于一个变量可以同时使用多个 transformations。

```
# the length of escaped 'Test' header body

$(hdr(Test){s.escape.common}{s.len})
```

返回字符串的所有 transformations 在错误或操作失败的情况下返回空字符串（例如 URI 的一个不存在的参数"{uri.param，name}"）。

transformations 可以在任何地方使用，作为脚本变量支持的一部分。在 xlog, avpops 或者其他模块的函数和参数中，在右侧赋值表达式中或在比较表达式中。

提醒：为了学习在 transformation 中使用的变量请查看 Scripting variables list.

2.5.2. 字符串转换

这些 transformation 的名字以’s’开始。可以对变零进行各种字符串操作。可用的字符串如下：

2.5.2.1. {s.len}

返回变量值的长度

```
$var(x) = "abc";
if($(var(x){s.len}) == 3)
{
    ...
}
```

2.5.2.2. {s.int}

转化给定字符串的初始部分转化为整数。如果没有数字返回 0。

```
$var(dur) = "2868.12 sec";
if ($(var(dur){s.int}) < 3600) {
   ...
}
```

2.5.2.3. {s.md5}

返回变量值的 md5

```
xlog("MD4 over From username: $(fU{s.md5})");
```

2.5.2.4. {s.substr,offset,length}

返回字符串从’offset’开始’length’个字符的子串。如果 offset 是负数，那么从字符串结尾开始计数：-1 是最后一个字符。如果是正数，0 是第一个字符。length 必须是正数：当取值为 0 时，返回到结尾的所有字符。offset 和 length 也

可以是变量。

使用用例

```
$var(x) = "abcd";
$(var(x){s.substr,1,0}) = "bcd"
```

2.5.2.5. {s.select,index,separator}

返回变量值的一个域。该域基于 separator 和 index.separator 用于标识域。index 必须为整数或者一个变量。如果 Index 是负数，域即计数从结尾开始：-1 指最后一个域。如果 index 是正数，0 指代第一个域。

使用用例

```
$var(x) = "12,34,56";
$(var(x){s.select,1,,}) => "34" ;

$var(x) = "12,34,56";
$(var(x){s.select,-2,,}) => "34"
```

2.5.2.6. {s.encode.hexa}

返回变量值的 hexa 编码。

2.5.2.7. {s.decode.hexa}

返回变量值的 hexa 解码

2.5.2.8. {s.escape.common}

返回变量值的转义字符串。转义的字符串有 ’," 和 0.在数据库查询时非常有用(应该注意非拉丁字符集)。

2.5.2.9. {s.unescape.common}

返回变量值的非转义字符串。

2.5.2.10.{s.escape.user}

返回变量值的转义字符串。将符合 RFC 要求的 SIP URI 的 user 部分中不允许的字符更改为'%hexa'。

2.5.2.11.{s.unescape.user}

返回变量值的非转义字符串。将'%hexa'更改为字符码。和上面的 transformation 相反。

2.5.2.12.{s.escape.param}

返回变量值的转移字符串。将符合 RFC 要求的 SIP URI 的 param 部分中不允许的字符更改为'%hexa'。

2.5.2.13.{s.unescape.param}

返回变量值的非转义字符串。将'%hexa'更改为字符码。和上面的 transformation 相反。

2.5.2.14.{s.tolower}

返回小写 ASCII 字母的字符串。

2.5.2.15.{s.toupper}

返回大写 ASCII 字母的字符串。

2.5.2.16.{s.index}

从另一个字符串开始位置搜索该字符串。如果发现返回字符串的开始 index,如果没有发现返回-1。变量 index 指定开始搜索的起始位置。支持负数并将 wrap。

```
$var(strtosearch) = 'onetwothreeone';
$var(str) = 'one';

# Search the string starting at 0 index
$(var(strtosearch){s.index, $var(str)}) # will return 0
$(var(strtosearch){s.index, $var(str), 0}) # Same as above
$(var(strtosearch){s.index, $var(str), 3}) # returns 11

# Negative offset
$(var(strtosearch){s.index, $var(str), -11}) # Same as above

# Negative wrapping offset
$(var(strtosearch){s.index, $var(str), -25}) # Same as above

#Test for existence of string in another
if ($(var(strtosearch){s.index, $var(str)}) >=0)
      xlog("found $var(sstr) in $var(strtosearch)");
```

2.5.2.17.{s.rindex}

从另一个字符串的结尾开始搜索一个字符。如果找到返回字符串的开始位置，如果没有找到返回-1.参数 index 指定了开始搜索的偏移位置。发现的字符串的位置要在提供的 offset 之前。支持负数并且将 wrap。

```
$(var(strtosearch){s.rindex, $var(str)}) # will return 11
$(var(strtosearch){s.rindex, $var(str), -3}) # will return 11
```

```
$(var(strtosearch){s.rindex, $var(str), 11}) # will return 11
$(var(strtosearch){s.rindex, $var(str), -4}) # will return 0
```

2.5.2.18.{s.fill.left, tok, len}

使用字符或者字符串向左填充字符串，直到达到给定的最终长度。如果初始字符串的长度大于或者等于给定的最终长度，则返回该字符串。

```
$var(in) = "485"; (also works for integer PVs)

$(var(in){s.fill.left, 0, 3})      => 485
$(var(in){s.fill.left, 0, 6})      => 000485
$(var(in){s.fill.left, abc, 8})    => bcabc485
为优化速度，不支持与变量参数或者连续的"s.fill"级联。
```

2.5.2.19.{s.fill.right, tok, len}

使用字符或者字符串向右填充字符串。如果字符串的长度大于或者等于给定的最终长度直接返回原始字符串。

```
$var(in) = 485; (also works for string PVs)

$(var(in){s.fill.right, 0, 3})     => 485
$(var(in){s.fill.right, 0, 6})     => 485000
$(var(in){s.fill.right, abc, 8}) => 485abcab
```

2.5.2.20.{s.width, len}

截取或者扩展输入到指定长度 len。扩展通过使用空格' '向右扩展实现。截取同样的从右边开始。

示例：

```
$var(in) = "transformation";

$(var(in){s.width, 14})     => "transformation"
$(var(in){s.width, 16})    => "transformation    "
$(var(in){s.width, 9})      => "transform"
```

2.5.2.21.{s.trim}

从输入字符串中删除任何前导或者尾随的空格。清理的字符包括 " "（空格）, \t (tab), \n (new line) 和 \r(回车符号)。

```
$var(in) = "\t \n input string    \r    ";

$(var(in){s.trim})      => "input string"
```

2.5.2.22.{s.trimr}

从输入字符串中删除任何尾随的空格。清理的字符包括 " "（空格）, \t (tab), \n (new line) 和 \r(回车符号)。

```
$var(in) = "\t \n input string    \r    ";

$(var(in){s.trimr})      => "\t \n input string"
```

2.5.2.23.{s.triml}

从输入字符串中删除任何前导的空格。清理的字符包括 " "（空格）, \t (tab), \n (new line) 和 \r(回车符号)。

```
$var(in) = "\t \n input string    \r    ";

$(var(in){s.triml})      => "input string    \r    "
```

2.5.2.24.{s.dec2hex}

将十进制（基数 10）转化为十六进制（基数 16），表示为字符串。

2.5.2.25.{s.hex2dec}

将十六进制（基数 16）转化为十进制（基数 10）。

2.5.2.26.{s.b64encode}

表示 ASCII 字符串格式的二进制输入数据。

```
$var(in) = "\x2\x3\x4\x5!@#%^&*";
$(var(in){s.b64encode})     => "AgMEBSFAIyVeJio="
```

2.5.2.27.{s.b64decode}

假设输入是 Base64 字符串并且家吗尽可能多的字符。

```
$var(in) = "AgMEBSFAIyVeJio=";
$(var(in){s.b64decode})     => "\x2\x3\x4\x5!@#%^&*"
```

2.5.2.28.{s.xor,secret}

根据两个字符串的长度，对’secret’字符串参数和输入字符串的执行一个或多个逻辑 XOR 操作。

```
$var(in) = "aaaaaabbbbbb";
$(var(in){s.xor,x})     => "!/>^P!/>^P!^U2^Q!^U2^Q"
```

2.5.3. URI 转换

transformation 的名字以’uri.'开始。变量的值视为 SIP URI。trnasformation 返回 SIP URI 的一部分(see struct sip_uri)。如果那部分不存在，则返回空串。

```
struct sip_uri {
        str user;         /* Username */
        str passwd;      /* Password */
        str host;         /* Host name */
        str port;         /* Port number */
        str params;      /* Parameters */
        str headers;
        unsigned short port_no;
        unsigned short proto; /* from transport */
        uri_type type; /* uri scheme */
        /* parameters */
```

```
        str transport;
        str ttl;
        str user_param;
        str maddr;
        str method;
        str lr;
        str r2; /* ser specific rr parameter */
        str gr; /* GRUU */
        /* values */
        str transport_val;
        str ttl_val;
        str user_param_val;
        str maddr_val;
        str method_val;
        str lr_val; /* lr value placeholder for lr=on a.s.o*/
        str r2_val;
        str gr_val;
        /* unknown params */
        str u_name[URI_MAX_U_PARAMS]; /* Unknown param names */
        str u_val[URI_MAX_U_PARAMS];   /* Unknown param valss */
        unsigned short u_params_no;     /* No of unknown params */
};
```

可用的 transformation 如下：

2.5.3.1. {uri.user}

返回 URI 模式的 user 部分。

2.5.3.2. {uri.host}

（类似于 {uri.domain}）

返回 URI 模式的 domain 部分。

2.5.3.3. {uri.passwd}

返回 URI 模式的 password 部分。

2.5.3.4. {uri.port}

返回 URI 模式的 port 部分。

2.5.3.5. {uri.params}

以字符串形式返回所有 URI 的 parameter

2.5.3.6. {uri.param,name}

返回名称为"name"的 parameter 的值。

2.5.3.7. {uri.headers}

返回 URI headers.

2.5.3.8. {uri.transport}

返回 transport URI 参数的值。

2.5.3.9. {uri.ttl}

返回 ttl URI 参数的值。

2.5.3.10.{uri.uparam}

返回 user URI 参数的值。

2.5.3.11.{uri.maddr}

返回 maddr URI 参数的值。

2.5.3.12.{uri.method}

返回 method URI 参数的值。

2.5.3.13.{uri.lr}

返回 lr URI 参数的值

2.5.3.14.{uri.r2}

返回 r2 URI 参数的值

2.5.3.15.{uri.schema}

返回给定 URI 的 schema 部分。

2.5.4. **VIA transformations**

这些 transformations 用于解析 Via header 并且全部以’via.'开始。变量的值被认为是 SIP Via header。transformation 返回 via header 的部分信息（see struct via_body）。如果没有找到请求的部分，返回空串。如果持有的 Via header 不存在，则 transformation 将失败(with script error)。除非在下面的描述中特殊说明，否则 transform 返回字符串（不是整数）。

```
/* Format: name/version/transport host:port;params comment */
  /* WARNING: keep in sync with tm/sip_msg.c via_body_cloner */
struct via_body {
```

```
        int error;
        str hdr;      /* Contains "Via" or "v" */
        str name;
        str version;
        str transport;
        str host;
        unsigned short proto; /* transport */
        unsigned short port;
        str port_str;
        str params;
        str comment;
        unsigned int bsize;                  /* body size, not including hdr */
        struct via_param* param_lst;    /* list of parameters*/
        struct via_param* last_param; /*last via parameter, internal use*/

        /* shortcuts to "important" params*/
        struct via_param* branch;
        str tid; /* transaction id, part of branch */
        struct via_param* received;
        struct via_param* rport;
        struct via_param* i;
        struct via_param* alias; /* alias see draft-ietf-sip-connect-reuse-00 */
        struct via_param* maddr;
        struct via_body* next; /* pointer to next via body string if
                                                  compact via or null */
};
```

示例:

```
    $var(upstreamtransport) = $(hdr(Via)[1]{via.transport}{s.tolower});
    $var(upstreamip) = $(hdr(Via)[1]{via.param,received});
    $var(clientport) = $(hdr(Via)[-1]{via.param,rport});
```

可用的 transformations 如下:

2.5.4.1. {via.name}

返回协议名称(RFC3261 BNF)，通常是 SIP。

2.5.4.2. {via.version}

返回协议版本号（RFC3261 BNF），通常是 2.0。

2.5.4.3. {via.transport}

返回传输协议（RFC3261 BNF），如 UDP,TCP,TLS。指用于发送请求信息的传输协议。

2.5.4.4. {via.host}

（类似于{via.domain}）

返回 send-by（RFC3261 BNF）的 host 部分。一般是请求消息发送方的 IP 地址，也是相应发送到的地址。

2.5.4.5. {via.port}

返回 send-by（RFC3261 BNF）的 port 部分。一般是请求消息发送方的端口，也是相应发送到的地址。transform 的接口可以是 integer 或者 string。

2.5.4.6. {via.comment}

与 via header 关联的 comment。结构体 via_body 包含此字段，但是不清楚 RFC3261 是否允许 Via 头具有 comment(请参阅 221 页，BNF 没有明确是否在 Via 允许 comment)。comment 是在 parens 中包含的文本。

2.5.4.7. {via.params}

返回所有的 Via headers parameters（RFC3261 的 via-param）。返回结果可以使用’{param.*}'转换。这基本上是 host 和 port 之后的所有内容。

2.5.4.8. {via.param,name}

返回名称为 name 的 Via header 参数的值。典型参数包含 branch,rport 和 received。

2.5.4.9. {via.branch}

返回 VIA header 的 branch 参数的值。

2.5.4.10.{via.received}

返回 VIA header 的 received 参数的值。

2.5.4.11.{via.rport}

返回 VIA header 的 rport 参数的值。

2.5.5. **Parameters List Transformations**

transformation 的名字以’param.'开始。变量的值是类似"name1=value1;name2=value2;…"的字符串。transformations 返回特定参数的值，或者指定位置参数的名字。

可用 transformations 如下

2.5.5.1. {param.value,name}

返回参数 name 的值。

示例

```
"a=1;b=2;c=3"{param.value,c} = "3"
```

'name’可以是变量。

2.5.5.2. {param.exist,name}

如果参数名字（不论是否有值）存在返回 1，否则返回 0。返回值可以是 integer 或者 string。name 可以是变量。可以用于检查没有只的参数的存在。

示例

```
"a=0;b=2;ob;c=3"{param.exist,ob};          # returns 1
"a=0;b=2;ob;c=3"{param.exist,a};           # returns 1
"a=0;b=2;ob;c=3"{param.exist,foo};         # returns 0
```

2.5.5.3. {param.valueat,index}

返回’index’指定位置参数值（从 0 开始）。

示例

```
"a=1;b=2;c=3"{param.valueat,1} = "2"
```

{param.name,index}

返回’index’指定位置参数名。

```
"a=1;b=2;c=3"{param.name,1} = "b"
```

2.5.5.4. {param.count}

返回列表中参数的个数。

```
"a=1;b=2;c=3"{param.count} = 3
```

2.5.6. Name-address Transformations

transformation 名称以’nameaddr.‘开始。变量值为类似’[display_name] uri’的字符串。transformations 返回特定域的值。

可用的 transformations 如下：

2.5.6.1. {nameaddr.name}

返回 display name 的值。

示例

```
'"test" <sip:test@opensips.org>' {nameaddr.name} = "test"
```

2.5.6.2. {nameaddr.uri}

返回 URI 的值

示例

```
'"test" <sip:test@opensips.org>' {nameaddr.uri} = sip:test@opensips.org
```

2.5.6.3. {nameaddr.len}

返回 name-addr 部分的长度。

2.5.6.4. {nameaddr.param,param_name}

返回名称为 param_name 的参数的值。

示例：

```
'"test" <sip:test@opensips.org>;tag=dat43h' {nameaddr.param,tag} = dat43h
```

2.5.6.5. {nameaddr.params}

返回所有参数对应的值。

2.5.7. IP Transformations

transformation 的名字以’ip.'开始。此类中可用的 transformations:

2.5.7.1. {ip.pton}

返回标识 IP 字符串的二进制表示形式。示例:

```
"192.268.2.234" {ip.pton} returns a 4 byte binary representation of the IP provided
```

2.5.7.2. {ip.ntop}

返回提供的二进制 ip 的字符串表示。示例:

```
"192.268.2.234"{ip.pton}{ip.ntop} = "192.268.2.234"
```

2.5.7.3. {ip.isip}

判断字符串是否是 IP,是返回 1 否则 0.示例:

```
"192.268.2.234" {ip.isip} = 1
"192.268.2.234.1" {ip.isip} = 0
```

2.5.7.4. {ip.family}

如果提供的二进制 IP 位 IPV4 或者 IPB6，相应返回 INET 或 INET6。示例:

```
"192.268.2.234" {ip.pton}{ip.family} = "INET"
```

2.5.7.5. {ip.resolve}

返回根据提供的字符串域解析的 IP 地址。如果提供的是字符串 IP，则 transformation 对字符串无影响。示例：

```
"opensips.org" {ip.resolve} = "78.46.64.50"
```

2.5.8. CSV Transformations

transformation 的名称以’csv.'开始。变量值假定为类似"field1,field2,…"的字符串。transformation 返回提供的 csv 的实体的个数，或者 csv 特定位置的字段。此类中可用的 transformation 如下：

2.5.8.1. {csv.count}

返回提供的 CSV 的条目数。示例：

```
"a,b,c" {csv.count} = 3
```

2.5.8.2. {csv.value}

返回 csv 特定位置的字段，计数从 0 开始。示例：

```
"a,b,c" {csv.value,2} = c
```

2.5.9. SDP Transformations

transformation 的名字以"sdp."开始。变量值假定为有效的 sdp body。transformation 返回 sdp body 的特定行。

此类中可用的 transformation 如下

2.5.9.1. {sdp.line}

返回 sdp body 的特定行。这个参数也接受第二个参数：指定从 sdp body 中获取的第一个参数类型的行号，索引从 0 开始。

示例：

```
if (is_method("INVITE"))
    {
        $var(aline) = $(rb{sdp.line,a,1});
        xlog("The second a line in the SDP body is $var(aline)\n");
    }

if (is_method("INVITE"))
    {
        $var(mline) = $(rb{sdp.line,m});
        xlog("The first m line in the SDP body is $var(mline)\n");
    }
```

2.5.10. **Regular Expression Transformations**

transformation 的名字一般以"re."开始。输入可以是任何字符串。

2.5.10.1.{re.subst,reg_exp}

reg_exp 可以是普通字符串或者变量。reg_exp 的格式如下：

```
/posix_match_expression/replacement_expression/flags
```

flag 可以是

- i - 匹配忽略大小写。
- s - 在多行字符串匹配
- g - 替换所有匹配项
  示例

```
$var(reg_input)="abc";
$var(reg) = "/a/A/g";
xlog("Applying reg exp $var(reg) to $var(reg_input) : $(var(reg_input){re.subst,$var(reg)})\n");
```

```
...
...
xlog("Applying reg /b/B/g to $var(reg_input) : $(var(reg_input){re.subst,/b/B/g})\n");
```

2.5.10.2. Examples

- parameter 在位置 1 对应值的长度（0 是第一个位置，1 是第二个位置）。

```
$var(x) = "a=1;b=22;c=333";
$(var(x){param.value,$(var(x){param.name,1})}{s.len}) = 2
```

- 1

- 判断是否未注册状态

```
if(is_method("REGISTER") && is_present_hf("Expires") && $(hdr(Expires){s.int})==0)
    xlog("This is an un-registration");
```

2.6. opensips 核心变量

2.6.1. Prev

Opensips 提供多种类型的变量，用于路由脚本。变量类型之间的差异来自 1）变量的可访问性（何处可访问）2）变量隶属于哪里（变量所在的位置）3）变量的读写状态（有些变量仅可读）4）多个值（for the same variable handled）

opensips 变量可以在脚本中轻松识别，因为他们的所有名称均以'$'符号开头。

语法：

伪变量的完整语法是：

```
$(<context>name(subname)[index]{transformation})
```

name 之外的部分是可选的，字段含义如下：

- name（必选）- 伪变量名(类型)
  例如 pvar,avp,ru,DLG_status 等等
- subname - 给定类型的某个 pv 的标识符
  例如 hdr(From), avp(name)。
- index - 一个 pv 可用存储多个值-可以一个值列表。如果指定 index 可以获取到 List 中的特定值。你也可以指定负数，-1 标识最后一个插入的，-2 标识标识倒数第二个插入的。

- transformation - 可以作用于伪变量的一系列操作。你可以在 Script-Tran 发现所有可用的 transformations.transformation 可以级联，使用一个 transformation 的输出作为另一个 transformation 的输入。
- context - the context in which the pseudo0variable will be evaluated. Now there are 2 pv contexts: reply and request. The reply context can be used in the failure route to request for the pseudo-variable to be evaluated in the context of the reply message. The request context can be used if in a reply route is desired for the pv to be evaluated in the context of the corresponding request.
  用法示例:
- Only name: $ru
- Name and 'subname: $hdr(Contact)
- Name and index: $(ct[0])
- Name, subname and index: $(avp(i:10)[2])
- Context - :
  $(<request>ru) from a reply route will get the Request-URI from the request
  $(<reply>hdr(Contact)) context can be used from failure route to access information from the reply
  变量类型:
- script variables - 顾名思义，这些变量严格的绑定到脚本语言。变量尽在路由块中可见-他们不与消息和事务相关，但是他们和进程相关（脚本变量将由同一 opensips 进程执行的脚本继承)。
- AVP- Attribute Value Pair - AVPs 是可以创建的动态变量-avps 和单个消息或者事务（如果使用有状态代理）相关联。最初（当接收或者创建）消息或者事务具有附加的 AVPs 的空白列表。在路由脚本中,脚本或者从脚本调用的函数可以创建新的 AVPS: 和事件或者事务关联。AVPS 对于事务处理的所有消息(回复或者请求)都是可访问的-branche_route, failure_route,noreply_route（对于 noreplay_route,需要开启 TM 参数 noreply_vap_mode)。AVPs 是可读写的并且现有的 AVP 可以删除（移除)。一个 AVP 可以包含多个值-新的赋值（或写操作）将向 AVP 添加新值；这个值保存在’最后添加最先使用’序列中。
- paeudo variables - 伪变量(或者 PV)提供已处理的 SIP 消息的信息（headers,RURI,transprot level info,a.s.o）或者 opensips 内部（time values, process PID,return code of a function)。根据他们提供的消息，PVs 即可以绑定到消息，也可以绑定到任何内容（全局)。大部分 PVs 是只读的并且仅仅少数允许写操作。一个 PV 可以返回几个值或仅返回一个值，具体取决于所引用的信息（可以有多个值)。标准 PV 是只读的并且仅仅返回一个值(如果没有另外说明).
- escape sequences - escape sequences 可以用于格式化字符串；他们实际上不是变量，而是格式化程序。

2.6.2. Script variables

Naming: **$var(name)**

Hints:

- 如果你想在一个路由中初始化一个脚本变量，最好使用相同的值初始化它(或重置它)，否则你可能从同一进程执行的先前路由继承数据。
- 脚本变量比 AVP 更快，因为他们直接引用内存位置。
- 脚本变量的值在 opensips 进程中持续存在。
- 脚本变量只能有一个值。
  使用用例

```
$var(a) = 2;    # sets the value of variable 'a' to integer '2'
$var(a) = "2";    # sets the value of variable 'a' to string '2'
$var(a) = 3 + (7&(~2)); # arithmetic and bitwise operation
$var(a) = "sip:" + $au + "@" + $fd; # compose a value from authentication username and From URI domain

# using a script variable for tests
if( [ $var(a) & 4 ] ) {
   xlog("var a has third bit set\n");
}
```

设置变量为 NULL 实际上是初始化为整数'0'。脚本变量没有 NULL 值。

2.6.3. AVP variables

Naming: **$avp(name)** or **$(avp(name)[N])**

使用索引"N"时，可以强制 AVP 返回某个值（第 N 个值）。如果没有指定

index,则返回第一个值。

Hints:

- 要在 onreply_route 中使用 AVPs,需要使用"modparam("tm", "onreply_avp_mode", 1)"
- 如果一个 VAPs 有多个值，值的顺序与添加的顺序相反。
- AVPs 是事务 context 的一部分，因此他们在当前事务的任何地方都是可访问的。
- AVP 的值可以删除。
  使用用例

2.6.3.1. 事务持久性示例

```
# enable avps in onreply route
modparam("tm", "onreply_avp_mode", 1)
...
```

```
route{
...
$avp(tmp) = $Ts ; # store the current time (at request processing)
...
t_onreply("1");
t_relay();
...
}

onreply_route[1] {
        if (t_check_status("200")) {
                # calculate the setup time
                $var(setup_time) = $Ts - $avp(tmp);
        }
}
```

2.6.3.2. 多个值示例

```
$avp(17) = "one";
# we have a single value
$avp(17) = "two";
# we have two values ("two","one")
$avp(17) = "three";
# we have three values ("three","two","one")

xlog("accessing values with no index: $avp(17)\n");
# this will print the first value, which is the last added value -> "three"

xlog("accessing values with no index: $(avp(17)[2])\n");
# this will print the index 2 value (third one), -> "one"

# remove the last value of the avp; if there is only one value, the AVP itself will be destroyed
$avp(17) = NULL;

# delete all values and destroy the AVP
avp_delete("$avp(17)/g");

# delete the value located at a certain index
$(avp(17)[1]) = NULL;

#overwrite the value at a certain index
```

```
$(avp(17)[0]) = "zero";
```

AVPOPS 模块提供了大量有效的方法操作 AVPS（如检查值，将值推送到不同的其他位置，删除 AVPs 等等）

2.6.4. **Pseudo Variables**

Naming: $name

Hints:

- PV tokens 可以作为不通脚本方法的参数给出，并且在函数执行前会赋值覆盖。
- 大多数 PVs 对于 opensips core 提供，但是也有模块导出 PV（以便为特定模块提供可用信息）-检查 modules 文档。预定义的 PVs（由 core 提供）按照字母顺序罗列。

2.6.4.1. URI in SIP Request’ s P-Asserted-Identity header

$ai - 请求的 P-Asserted-Identity header 的 URI 引用（RFC3325）。

2.6.4.2. Authentication Digest URI

$adu - Authorization 或者 Proxy-Authorization header 的 URI。用于计算 http Digest 响应。

2.6.4.3. Authentication realm

$ar - Authorization 或者 Proxy-Authorization header 中 realm。

2.6.4.4. Authentication user

$au - Authorization 或者 Proxy-Authorization header 中 username 的 user 部分。

2.6.4.5. Auth username domain

$ad - Authorization 或者 Proxy-Authorization header 中 username 的 domain 部分。

2.6.4.6. Auth nonce

$an - Authorization 或者 Proxy-Authorication header 的 nonce。

2.6.4.7. Auth response

$auth.resp - Authorization 或者 Proxy-Authorization header 的认证返回。

2.6.4.8. Auth nonce

$auth.nonce - Authorization 或者 Proxy-Authorization header 的 nonce 字符串。

2.6.4.9. Auth opaque

$auth.opaque - Authorization 或者 Proxy-Authorization header 的 opaque 字符串。

2.6.4.10. Auth algorithm

$auth.alg - Authorization 或者 Proxy-Authorization header 的 algorithm 字符串。

2.6.4.11. Auth QOP

$auth.qop - Authorization 或者 Proxy-Authorization header 的 qop 参数的值。

2.6.4.12. Auth nonce count(nc)

$auth.nc - Authorization 或者 Proxy-Authorization header 的 nonce count 参数的值。

2.6.4.13. Auth whole username

$aU - Authorization 或者 Proxy-Authorization header 的整个 username。

2.6.4.14. Acc username

$aU - 用于计费的 username。这是一个 selective pseudo variable（继承自 acc 模块）。如果存在返回$au，否则返回 From username。

2.6.4.15. Argument options

$argv - 提供对使用’-o’选项指定的命令行参数的访问。例子：

```
# for option '-o foo=0'
xlog("foo is $argv(foo) \n");
```

2.6.4.16. Branch flags

$bf - 展示当前 SIP 请求的 branch flags 的列表。

2.6.4.17. Branch

$branch - 该变量用于创建一个新的 branches：写入 SIP URI 的值。示例：

```
# creates a new branch
$branch = "sip:new@doamin.org";
```

```
    # print its URI
    xlog("last added branch has URI $(branch(uri)[-1]) \n");
```

2.6.4.18. Branch fields

$branch() - 该变量提供对现有 branch 的所有 fields/attributes 的读写权限（先前使用 append_branch()创建）。branch 的域如下：

- uri - the RURI of the branch (string value)
- duri - destination URI of the branch (outbound proxy of the branch) (string value)
- q - q value of the branch (int value)
- path - the PATH string for this branch (string value)
- flags - the branch flags of this branch (int value)
- socket - the local socket to be used for relaying this branch (string value)
  该变量接收 index$(branch(uri)[1])以访问特定的 branch(可以同时定义多个 branch)。
  index 从 0 开始（第一个 branch）。如果 index 是负数，则指倒数第 n 个（-1 表示最后一个 branch）。
  如果想要得到所有的 branch,使用 * index- $(branch(uri)[*])。
  示例：

```
    # creates the first branch
    append_branch();
    # creates the second branch
    force_send_socket(udp:192.268.1.12:5060);
    $du = "sip:192.268.2.20";
    append_branch("sip:foo@bar.com","0.5");

    # display branches
    xlog("----- branch 0: $(branch(uri)[0]) , $(branch(q)[0]), $(branch(duri)[0]),
$(branch(path)[0]), $(branch(flags)[0]), $(branch(socket)[0]) \n");
    xlog("----- branch 1: $(branch(uri)[1]) , $(branch(q)[1]), $(branch(duri)[1]),
$(branch(path)[1]), $(branch(flags)[1]), $(branch(socket)[1]) \n");

    # do some changes over the branches
    $branch(uri) = "sip:user@domain.ro";      # set URI for the first branch
    $(branch(q)[0]) = 1000;    # set to 1.00 for the first branch
    $(branch(socket)[1]) = NULL;    # reset the socket of the second branch
    $branch(duri) = NULL;    # reset the destination URI or the first branch
```

该变量是 R/W 变量（可以从路由逻辑中为期赋值）。

2.6.4.19. Call-Id

$ci - call-id header 内容的引用

2.6.4.20. Content-Length

$cl - content-length header 内容的引用

2.6.4.21. CSeq number

$cs - cseq header 的 cseq number 的引用。

2.6.4.22. Contact instance

$ct - contact header 的 contact instance/body 的引用。一个 contact instance 的样式：display_name + URI + contact_params。由于一个 contact header 可能包含多个 contact instance 并且一个消息可能包含多个 contact,因此在$ct 变量中添加索引 index。

- name - display name
- uri - contact uri
- q - q params(value only)
- expires - expires param(value only)
- methods - methods param(value only)
- received - received param(value only)
- params - all params (include names)

示例

```
$ct.fields(uri) - the URI of the first contact instance
$(ct.fields(name)[1]) - the display name of the second contact instance
```

2.6.4.23. contact-Type

$cT - Content-Type header 的内容引用。并且 Content-type headers 在多个 body 中。

- $cT - 消息的主要 Content-Type;headers 中的一个。
- $(cT[n]) - 消息多个信息中的第 n 个 Content-Type，索引以 0 开始。
- $(cT[-n]) - 消息多个信息中的倒数第 n 个 Content-Type，索引以-1 开始。
- $(cT[*]) - 所有的 Content-Type header 包含主要的以及其他的。

2.6.4.24. Domain of destination URI

$dd - destination uri 的 domain 引用

该变量是 R/W（可以在路由逻辑中赋值）。

2.6.4.25. Diversion header URI

$di - Diversion header URI 的引用。

2.6.4.26. Diversion “privacy” parameter

$dip - Diversion header "privacy"参数值的引用。

2.6.4.27. Diversion “reason” parameter

$dir - Diversion header "reason"参数值的引用。

2.6.4.28. Port of destination URI

$dp - destination uri 的 port 引用。

该变量可 R/W（可以在路由逻辑中修改）。

2.6.4.29. Transport protocol of destination URI

$dp - destination uri 的传输协议。

2.6.4.30. Destination set

$ds - destination set 的引用。

2.6.4.31. Destination URI

$du - destination uri 的引用(用于发送请求的 outbound proxy)。如果 loose_route()

返回 TRUE，依据第一个 Route header 为 destination uri 赋值。

该变量可 R/W（可以在路由逻辑中赋值）。

2.6.4.32. Error class

$err.class - error 对应的类名。（解析 erros 对应 index ‘1’）。

2.6.4.33. Error level

$err.level - error 的严重级别。

2.6.4.34. Error info

$err.info - error 的文本描述。

2.6.4.35. Error reply code

$err.rcode - recommended reply code

2.6.4.36. Error reply reason

$err.rreason - recommended reply reason phrase

2.6.4.37. From URI domain

$fd - 'From' header URI 的 domain。

2.6.4.38. From display domain

$fn - 'From' header 的 display name

2.6.4.39. Forced socket

$fs - 以 proto:ip:port 格式用于消息发送的强制 socket。

该变量是可 R/W 的（可以在路由脚本修改）。

2.6.4.40. From tag

$ft - 'From' header 的 tag 参数引用。

2.6.4.41. From URI

$fu - ‘From’ header 的 URI 引用。

2.6.4.42. From URI username

$fY ‘From’ header 中 URI 的 username 引用。

2.6.4.43. OpenSIPS Log Level

$log_level - 修改当前进程的日志级别；日志级别可以设置为一个新的值（请参阅可用的值或者可以将其重置为全局日志等级。如果进跟踪和调试特定代码，此功能非常有效）。

使用用例

```
log_level= -1 # errors only
.....
{
   ......
   $log_level = 4; # set the debug level of the current process to DBG
   uac_replace_from(....);
   $log_level = NULL; # reset the log level of the current process to its default level
   .......
}
```

2.6.4.44. SIP message buffer

$mb - reference to SIP message buffer

2.6.4.45. Message Flags

$mf - 展示当前 SIP 请求的 message/trnasaction flags set 的列表。

2.6.4.46. SIP message ID

$mi - SIP message id 的引用。

2.6.4.47. SIP message length

$ml - SIP message length 的引用。

2.6.4.48. Domain in SIP Request’ s original URI

$od - 请求的原始 R-URL 的 domain 引用。

2.6.4.49. Port of SIP request’ s original URI

$op - 原始 R-URI 的端口引用

2.6.4.50. Transport protocol of SIP request original URI

$oP - 原始 R-URI 的传输协议。

2.6.4.51. SIP Request’ s original URI

$ou - 请求原始 URI 引用。

2.6.4.52. Username in SIP Request’ s original URI

$oU - 请求原始 URI 的 username。

2.6.4.53. Route parameter

$param(idx) - 检索路由的参数。索引 index 可以是整数也可以是 pseudo-variable(index 从 1 开始)。

示例

```
route {
    ...
    $var(debug) = "DBUG:"
    route(PRINT_VAR, $var(debug), "param value");
    ...
}

route[PRINT_VAR] {
    $var(index) = 2;
    xlog("$param(1): The parameter value is <$param($var(index))>\n");
}
```

2.6.4.54. Domain in SIP Request’ s P-Preferred-Identity header URI

$pd - 请求的 P-Preferred-Identity header URI 的 domain 引用（参考 RFC3325）。

2.6.4.55. Display Name in SIP Request’ s P-Preferred-Identity header

$pn - 请求的 P-Preferred-Identity header URI 的 Display Name 引用（参考 RFC3325）。

2.6.4.56. Process id

$pp - process id 的引用(pid)

2.6.4.57. Protocol of received message

$pr or $proto - 接收的消息的协议(UDP,TCP,TLS,SCTP,WS)。

2.6.4.58. User in SIP Request’ s P-Preferred-Identity header URI

$pU - 请求的 P-Preferred-Identity header URI 的 user 引用（参考 RFC3325）。

2.6.4.59. URI in SIP Request’ s P-Preferred-Identity header

$pu - 请求的 P-Preferred-Identity header 的 URI 引用。

2.6.4.60. Domain in SIP Request’ s URI

$rd - 请求 URI 的 domain 引用。

2.6.4.61. Body of request/reply

$rb - 消息体的引用。

- $rb - 消息体
- $(rb[*]) - 类似于$rb
- $(rb[n]) - 消息体多个 body 中的第 n 个 body。计数从 0 开始。
- $(rb[-n]) - 消息体多个 body 中的倒数第 n 个 body，计数从-1 开始（对应最后一个）

2.6.4.62. Returned code

$rc - 上次调用的 function 返回的 code

$retcode - 和$rc 类似。

2.6.4.63. Remote-Party-ID header URI

$re - Remote-Party-ID header URI 的引用。

2.6.4.64. SIP request’ s method

$rm - 请求 method 的引用

2.6.4.65. SIP request’ s port

$rp - R-URI 的端口

该变量是 R/W 的（可以在路由脚本赋值）。

2.6.4.66. Transport protocol of SIP request URI

$rP - R-URI 的传输协议。

2.6.4.67. SIP reply’ s reason

$rr - reference to reply’s reason

2.6.4.68. SIP reply’ s status

$rs - reference to reply’s status

2.6.4.69. Refer-to URI

$rt - refer-to header 的引用。

2.6.4.70. SIP Request’ s URI

$ru - 请求 URI 的应用

该变量是 R/W 变量（可以在路由脚本赋值）。

2.6.4.71. Username in SIP Request’ s URI

$rU - 请求 URI 的 username 引用。

该变量是 R/W 变量（可以在路由脚本赋值）。

2.6.4.72. Q value of the SIP Request’ s URI

$ru_q - R-URI 中 q 的值。

2.6.4.73. Received IP address

$Ri - 收到请求的接口 IP 地址。

2.6.4.74. Received port

$Rp - 接收消息对应的端口。

2.6.4.75. Script flags（Removed in OpenSIPS 2.2）

$sf - 展示当前 sip 请求的 script flags set 的列表。

2.6.4.76. IP source address

$si - 消息的 ip 源地址。

2.6.4.77. Source port

$sp - 消息的源端口。

2.6.4.78. To URI Domain

$td - ‘To’ header URI 的 domain。

2.6.4.79. To display name

$tn - ‘To’ header 的 diaplay name。

2.6.4.80. To tag

$tt - ‘To’ header 的 tag 参数。

2.6.4.81. To URI

$tu - ‘To’ header 的 URI

2.6.4.82. To URI Username

$tU - ‘To’ header 的 URI 的 username

2.6.4.83. Formatted date and time

$time(format) - 依据 unix 时间返回格式化的字符串。

2.6.4.84. Branch index

$T_branch_idx - 执行 branch_route[]的 branch 的 index（索引从 1 开始）。如果在 branch_route[] 代码块外使用，值为 0。由 TM 模块提供。

2.6.4.85. String formatted time

$Tf - 字符串格式化时间。

2.6.4.86. Current unix time stamp in seconds

$Ts - 当前的 unix 时间戳(秒)。

2.6.4.87. Startup unix time stamp

$TS - 启动 unix 时间戳

2.6.4.88. User agent header

$ua - user agent header 字段

2.6.4.89. SIP Headers

$(hdr(name)[N]) - 表示由’name’标识的第 N 个 header 的正文。如果省略[N]，打印第一个 header 对应的正文。N=0 时获取第一个 header，N=1 时获取第二个 header。 打印最后一个 header，使用-1。现在还不支持其他负数。不允许存在空格(before }, before or after {, [,] symbols)。当 N=’*'时，打印指定类型的所有 headers。

该模块需要识别大多数 compact header 名字（所有 opensips 识别的名字）。如果不能，必须明确指定的 compact form。建议使用 headers 的专用说明符（比如%ua 对应 user agent header），如果可用，则更快。

$(hdrcnt(name)) - 返回由'name'标识的 headers 的数目。使用和上面$(hdr(name))相同规则的 headers name。很多 headers（如 Via, Path, Record-Route）可能出现很多次。该变量返回执行类型 headers 的数目。

请注意，一些 headers 可能用逗号连接在一起并显示未单个 header line。该变量返回 header lines 的数目，而不是 header 的值。

对于下面的消息碎片，$(hdrcn(Path))值为 2，并且$(hdr(Path)[0])返回 <a.com>：

```
Path: <a.com>
Path: <b.com>
```

对于下面的消息碎片，$(hdrcn(Path))值为 1，并且$(hdr(Path)[0])返回 <a.com>,<b.com>：

```
Path: <a.com>,<b.com>
```

注意上面两个例子语音上都是等效的，但是变量对应不同的值。

2.6.4.90. Route Type

$rT - 将当前路由类型保存为字符串。用于在另一个路由脚本中，确定原始路由类型，例如从 onreplay_route 调用的 route。允许更加重用可重用的 routes 如日志 route：在日志信息中包含 route type。

2.6.4.91. Current script line and file

$cfg_line - 保存正在执行的路由脚本的当前行，有益于记录日志

$cfg_file - 保存正在执行的 cfg 文件名，在通过 include 包含多个脚本时非常有效。

2.6.4.92. $shv(name)

存储在共享内存的 pseudo-variables。所有 opensips 进程都可以看到$shv(name)的值。每个"shv"都有单个值并且初始化为整数 0。可以使用"shvset"参数初始化共享变量。改模块提供了一组 MI 函数去获取/设置共享变量的值。

参考 cfgutils

使用示例

```
...
modparam("cfgutils", "shvset", "debug=i:1")
...
if ($shv(debug) == 1) {
        xlog("request: $rm from $fu to $ru\n");
}
...
```

2.6.5. **Escape Sequences（转移序列）**

这些序列主要由 xlog 模块提供并使用，使用转义序列用不通的颜色打印消息(前景和背景)。

2.6.5.1. Foreground and background colors

$C(xy) - 转义序列，¿x¿ represents the foreground color and ¿y¿ represents the background color.
支持的颜色如下:

- x : default color of the terminal
- s : Black
- r : Red
- g : Green
- y : Yellow
- b : Blue
- p : Purple
- c : Cyan
- w : White

2.6.5.2. 示例

使用示例

```
...
route {
...
        $avp(uuid)="caller_id";
        $avp(tmp)= $avp(uuid) + ": " + $fu;
        xlog("$C(bg)$avp(tmp)$C(xx) [$avp(tmp)] $C(br)$cs$C(xx)=[$hdr(cseq)]\n");
...
}
...
```