**kamailio源码修改指南**

|  |  |
| --- | --- |
| 文档编号： |  |
| 编 写 人： | 崔士阳 |
| 审 批 人： | [经理] |
| 生效日期： | 2015/10/20 |

**爱悠团队**

文档历史：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本 | 修订者 | 修订内容 | 修订日期 | 审批者 | 审批日期 |
| 0.1.0 | 崔士阳 | 文档初稿 | 2015.10.20 |  |  |
| 0.2.0 | 崔士阳 | 1. 重新修改文档的内容 2. 优化文档布局 | 2016.1.9 |  |  |
| 0.2.1 | 崔士阳 | 添加针对web控制功能的实现需要修改的源码内容 | 2016.03.09 |  |  |
| 0.2.2 | 崔士阳 | 删除无效的内容 | 2016.03.15 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

*填写说明：*

* *版本号规则：版本号采用三级生成规则，即0.0.0，第一位表示主版本号，每一次正式发布，主版本号加1，第二位表示从版本号，每一次对文档进行较大范围修订（增、删章节，修改内容主干等），从版本号加1；第三位为修订版本号，在不改变文档主体内容结构与思想，只做文字上的修饰与调整，则修订版本加1。版本号不采用进制策略，按规则累加，当上层版本变化时，次级版本归0。例如0.0.92，当从版本号升为1时，修订版本号归0，成为0.1.0。*
* *修订内容：详细描述修订及修订内容所在章节*

目录

[1 先决条件 3](#_Toc445819572)

[2 修改配置文件 3](#_Toc445819573)

[2.1 调试功能开启 3](#_Toc445819574)

[2.2 开启siptrace模块功能 4](#_Toc445819575)

[2.3 添加modules的路径 6](#_Toc445819576)

[3 修改siptrace模块 7](#_Toc445819577)

[3.1 添加Thrift通讯模块 7](#_Toc445819578)

[4 网关在线状态检测 10](#_Toc445819579)

[5 Kamailio与web server通讯 11](#_Toc445819580)

[6 TLS模块加载 12](#_Toc445819581)

[7 web控制指令的转发 14](#_Toc445819582)

[7.1 原理简介 14](#_Toc445819583)

[7.2 源码修改 14](#_Toc445819584)

[7.3 Kamailio-🡪web返回值 20](#_Toc445819585)

[7.4 备注 20](#_Toc445819586)

[8 不同版本源码的区别 21](#_Toc445819587)

# 先决条件

* 读者熟悉Ubuntu/Centos、Linux各种基本操作。
* 读者熟悉apt-get软件包管理工具。
* 读者熟悉编译安装Linux程序的一般过程。
* 操作用户具有sudo权限。

# 修改配置文件

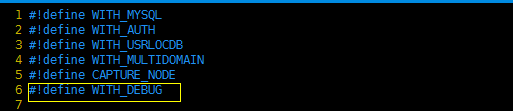
打开配置文件kamailio.cfg，**配置文件的路径默认是/usr/local/etc/kamailio/**

## 调试功能开启

如果想输出调试消息，则在该文件开头添加如下内容，

*#define WITH\_DEBUG*

如图，



## 开启siptrace模块功能

下面是引用siptrace模块需要修改的内容

如图，

*#ifdef CAPTURE\_NODE*

*Loadmodule “siptrace.so”*

*#endif*

*#ifdef CAPUTRE\_NODE*

*#Destination,where to send the traffic*

*#modparam(“siptrace”,”duplicate\_uri”,”自定义的URL”)*

//如果需要kamailio将消息包发送给web，则将该配置打开，否则直接关闭即可

*#Trace all traffic*

*modparam(“siptrace”,”trace\_on”,1)*

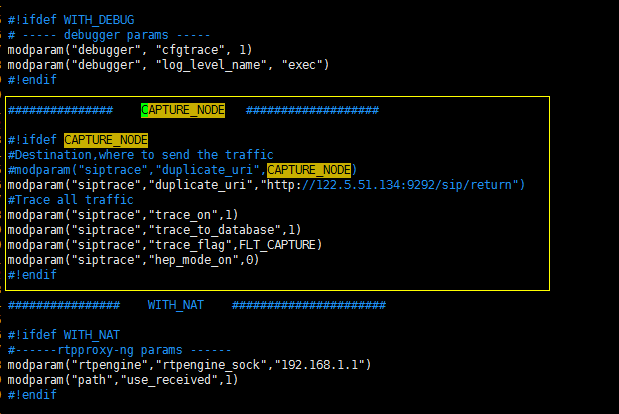
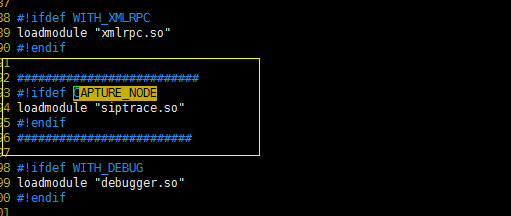
//将trace\_to\_database设置为1，表示保存到数据库；0，不保存

*modparam(“siptrace”,”trace\_to\_database”,1)*

*modparam(“siptrace”,”trace\_flag”,FLT\_CAPTURE)*

*modparam(“siptrace”,”hep\_mode\_on”,0)*

*#endif*



同时还要加上如下内容，

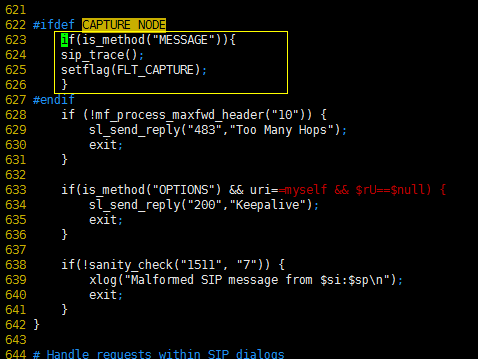
如图

*#ifdef CAPTURE\_NODE*

*sip\_trace();*

*setflag(FLT\_CAPTURE);*

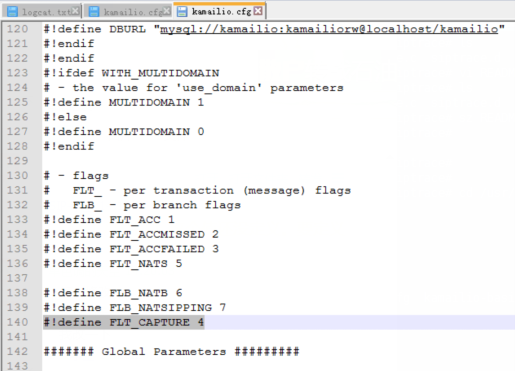
*#endif*



需要添加如下内容，定义FLT\_CAPTURE，

*#！define FLBT\_CAPTURE 4*

如下图：



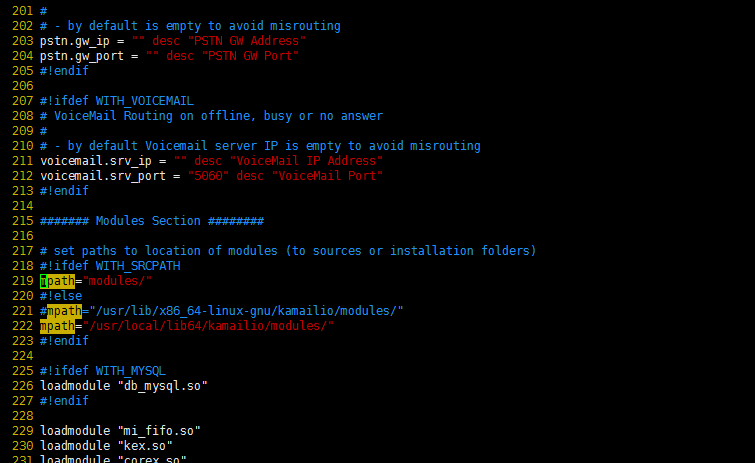
## 添加modules的路径

Kamailio源码编译安装之后，会将源码的modules目录下的各个模块编译生成的.so文件拷贝到/usr/local/lib64/kamailio/modules/路径下（一般是这个路径），所以需要在kamailio.cfg文件里添加该路径，才能保证kamailio的各个模块有效。

打开kamailio.cfg文件，添加如下内容：

*mpath = /usr/local/lib64/kamailio/modules/*

如图，



**注意，mpath后面的路径需要根据自己的实际情况来定，一般情况下路径就是上图所示。**

# 修改siptrace模块

## 添加Thrift通讯模块

在siptrace模块的siptrace.c文件中添加如下内容：

*#include <glib-object.h>*

*#include <thrift/c\_glib/protocol/thrift\_binary\_protocol.h>*

*#include <thrift/c\_glib/transport/thrift\_buffered\_transport.h>*

*#include <thrift/c\_glib/transport/thrift\_socket.h>*

*#include "CommunicateInterface.h"*

添加这些头文件的作用是为了使用thrift通讯，然后添加Evy\_thrift\_transfer()函数，内容如下：

*static int Evy\_thrift\_transfer(int length, char \*buf) {*

*GError \*error = NULL;*

*int exit\_status = 0;*

*ThriftSocket \*socket;*

*ThriftTransport \*transport;*

*ThriftProtocol \*protocol;*

*CalculatorIf \*client;*

*#if (!GLIB\_CHECK\_VERSION (2, 36, 0))*

*g\_type\_init ();*

*#endif*

*socket = g\_object\_new (THRIFT\_TYPE\_SOCKET,*

*"hostname", "localhost",*

*"port", 9090,*

*NULL);*

*transport = g\_object\_new (THRIFT\_TYPE\_BUFFERED\_TRANSPORT,*

*"transport", socket,*

*NULL);*

*protocol = g\_object\_new (THRIFT\_TYPE\_BINARY\_PROTOCOL,*

*"transport", transport,*

*NULL);*

*thrift\_transport\_open (transport, &error);*

*/\* In the C (GLib) implementation of Thrift, service methods on the*

*server are accessed via a generated client class that implements*

*the service interface. In this tutorial, we access a Calculator*

*service through an instance of CalculatorClient, which implements*

*CalculatorIf. \*/*

*client = g\_object\_new (TYPE\_CALCULATOR\_CLIENT,*

*"input\_protocol", protocol,*

*"output\_protocol", protocol,*

*NULL);*

*/\* Each of the client methods requires at least two parameters: A*

*pointer to the client-interface implementation (the client*

*object), and a handle to a GError structure to receive*

*information about any error that occurs.*

*On success, client methods return TRUE. A return value of FALSE*

*indicates an error occurred and the error parameter has been*

*set. \*/*

*if (!error && calculator\_if\_ping (client, &error)) {*

*puts ("ping()");*

*}*

*/\* Service methods that return a value do so by passing the result*

*back via an output parameter (here, "return\_value"). \*/*

*if (!error && calculator\_if\_message (client, &return\_value, buf, &error)) {*

*printf ("return = %d\n", return\_value);*

*}*

*/\* Thrift structs are implemented as GObjects, with each of the*

*struct's members exposed as an object property. \*/*

*/\* Struct objects can be reused across method invocations. \*/*

*if (error) {*

*printf ("ERROR: %s\n", error->message);*

*g\_clear\_error (&error);*

*exit\_status = 1;*

*}*

*thrift\_transport\_close (transport, NULL);*

*g\_object\_unref (client);*

*g\_object\_unref (protocol);*

*g\_object\_unref (transport);*

*g\_object\_unref (socket);*

*return exit\_status;*

*}*

然后，修改sip\_trace()函数，调用Evy\_thrift\_transfer()函数，内容如下：

*struct \_siptrace\_data sto;*

*struct onsend\_info \*snd\_inf = NULL;*

*str msg\_body;*

*msg\_body.s = get\_body(msg);//get the content of message\_body*

*if (msg\_body.s==0) {*

*LM\_ERR("failed to extract body from msg!\n");*

*return -1;*

*}*

*msg\_body.len = get\_content\_length(msg);//get the length of message\_body*

*LM\_DBG("%s, %d\n",msg\_body.s, msg\_body.len);*

*Evy\_thrift\_transfer(msg\_body.len, msg\_body.s);*

*int ret = return\_value;*

*LM\_DBG("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");*

*LM\_DBG("ret = %d\n",ret);*

*LM\_DBG("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");*

*if (ret == 1) {*

*LM\_DBG("authentication success!\n");*

*return 1;*

*}else if (ret == -2) {*

*LM\_DBG("authentication failed!\n");*

*return 2;*

*}else if (ret == -3) {*

*LM\_DBG("message content is wrong!\n");*

*return 3;*

*}if (dst){*

*if (dst->send\_sock == 0){*

*dst->send\_sock=get\_send\_socket(0, &dst->to, dst->proto);*

*if (dst->send\_sock==0){*

*LM\_ERR("can't forward to af %d, proto %d no corresponding"*

*" listening socket\n", dst->to.s.sa\_family, dst->proto);*

*return -1;*

*}*

*}*

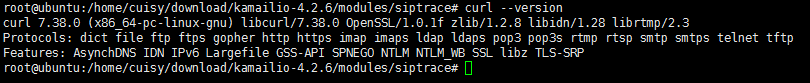
*}*

以上只是截取了sip\_trace()函数中的部分代码，后面的代码没有改动，所以就不添加了。

修改完毕之后，用本地kamailio源码里新的siptrace模块替换服务器中源码的siptrace模块，然后安装curl库，安装方法如下，



然后执行如下命令，看结果，



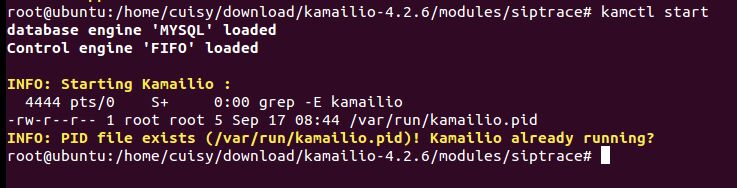
接下来，就是重新编译siptrace模块的过程了，进入到siptrace路径下，然后执行make，过程中可能出现相关报错，根据实际情况来解决问题。如果没有错误，则把新生成siptrace.so库复制到/usr/local/lib64/kamailio/modules/ 下。

**注意：一般情况下路径是/usr/local/lib64/kamailio/modules/，需要根据实际情况确定一下**。

如果都没有问题了，那么执行如下指令启动kamailio服务，

*kamctl start*

如果出现下面的错误，



则删除该路径下的pid文件并重新执行启动指令，

*rm /var/run/kamailio.pid*

*kamctl start*

注：4.3.2版本以后的kamailio将kamailio.pid文件的生成路径修改成了/var/run/kamailio/，所以如果在执行启动指令时报错，首先要看一下是否有这个路径，如果没有，则手动创建这个路径，然后再执行启动指令。

# 网关在线状态检测

网关对于kamailio来说，只是一个sip客户端而已，当网关联网时，在kamailio数据库表location中会插入一条数据，包括该sip客户端的sip账号、上线时间等信息，而当网关退网时，location表中关于该网关的数据会被删除。所以，可以通过遍历查询location表中username的值，确定网关是否在线。

# Kamailio与web server通讯

Kamailio和web server之间的互相通讯建立在http协议基础之上，具体实现如下：

1. **Kamailio🡪web server**

通过http建立连接，web server提供一个地址，Kamailio通过修改配置文件来实现该通道的建立，修改内容为

其中，红色部分为修改内容，<http://122.5.51.134.:9292/sip/return>为web端提供。

*#!ifdef CAPTURE\_NODE*

*#Destination,where to send the traffic*

*#modparam("siptrace","duplicate\_uri",CAPTURE\_NODE)*

*modparam("siptrace","duplicate\_uri","http://122.5.51.134.:9292/sip/return")*

*#Trace all traffic*

*modparam("siptrace","trace\_on",1)*

modparam("siptrace","trace\_to\_database",1)

…….

另外，在Kamailio源码的siptrace模块中，向siptrace.c中添加了curl\_post()函数，用于实现sip到web的通讯。

1. **web server🡪Kamailio**

也是通过http建立连接，web端将数据发送至<http://kamailioserver:5060/sip/return>

其中，kamailioserver为kamailio服务器的IP，5060为kamailio服务器的默认端口，实际操作中需要根据实际情况填写端口号。

修改kamailio.cfg文件，具体修改内容为：

*event\_route[xhttp:request] {*

*xlog("http:request hu= $rm => $hu\n");*

*xlog("http:request rb= $rm => $rb\n");*

}

其中，$hu代表的是<http://kamailioserver:5060/sip/return>

中的/sip/return这部分路径，$rb代表的是web server发送的具体内容，$rm代表的是使用何种方式发送，例如GET、POST、DELETE。

通过以上方式，即可实现web server和Kamailio的互通。

# TLS模块加载

首先需要修改TLS模块的源码，修改的文件为tls\_domain.c，修改的内容为：

*static int passwd\_cb(char \*buf, int size, int rwflag, void \*filename) {*

*char \*passwd=”everyoo2015”*;//此处需要根据生成的证书文件来修改

*strcpy(buf,passwd);*

*return strlen(buf);*

*}*

即需要将该处的passwd改成CA证书中使用的密码，否则会报错。然后修改tls.cfg文件，tls.cfg的内容如下：

*#*

*# Example Kamailio TLS Configuration File*

*#*

*# This is the default server domain, settings*

*# in this domain will be used for all incoming*

*# connections that do not match any other server*

*# domain in this configuration file.*

*#*

*# We do not enable anything else than TLSv1*

*# over the public internet. Clients do not have*

*# to present client certificates by default.*

*#*

*[server:default]*

*method = TLSv1*

*verify\_certificate = yes*

*require\_certificate = no*

*private\_key = /usr/local/etc/kamailio/test\_1.pem*

*certificate = /usr/local/etc/kamailio/test\_1.crt*

*#ca\_list = ./modules/tls/cacert.pem*

*#crl = ./modules/tls/crl.pem*

*# This is an example server domain for TLS connections*

*# received from the loopback interface. We allow*

*# the use of SSLv2 and SSLv3 protocols here, we do*

*# not require that clients present client certificates*

*# but if they present it it must be valid. We also use*

*# a special certificate and CA list for loopback*

*# interface.*

*#*

*[server:192.168.1.19:5061]*

*method = SSLv23*

*verify\_certificate = no*

*require\_certificate = no*

*#private\_key = ./modules/tls/local\_key.pem*

*#certificate = ./modules/tls/local\_cert.pem*

*private\_key = /usr/local/etc/kamailio/test\_1.pem*

*certificate = /usr/local/etc/kamailio/test\_1.crt*

*#verify\_depth = 3*

*#ca\_list = local\_ca.pem*

*#crl = local\_crl.pem*

*# Special settings for the iptel.org public SIP*

*# server. We do not verify the certificate of the*

*# server because it can be expired. The server*

*# implements authentication using SSL client*

*# certificates so configure the client certificate*

*# that was given to use by iptel.org staff here.*

*#*

*#[client:195.37.77.101:5061]*

*#verify\_certificate = no*

*#certificate = ./modules/tls/iptel\_client.pem*

*#private\_key = ./modules/tls/iptel\_key.pem*

*#ca\_list = ./modules/tls/iptel\_ca.pem*

*#crl = ./modules/tls/iptel\_crl.pem*

其中，红色标注的是需要修改的内容，test\_1.pem是生成的key，test\_1.crt是生成的CA文件。在tls.cfg文件中，server部分设置的端口是5061，此处需要注意。

# web控制指令的转发

## 原理简介

该设计目的是实现用户通过web控制设备的功能，具体的设计是web将指令包以JSON格式发送到kamailio，kamailio对发来的数据包进行解析，获取到其中有效的数据，包括userid、gatewayid和cmd等，然后通过调用msilo模块，将cmd发送给gatewayid代表的网关，网关接收到web发送的JSON指令之后，通过自己的解析模块完成对JSON指令的解析，然后将指令下发给具体设备，完成控制。

## 源码修改

对于msilo模块，在源码的msilo.c文件中添加两个函数，具体内容如下：添加m\_send\_message()函数，内容如下：

*static int m\_send\_message(str\* from,str\* to,str\* body,send\_result\_callback \_resultcallback,void\* userdata)*

*{*

*static char hdr\_buf[1024];*

*static char body\_buf[1024];*

*str ctype;*

*str extra\_hdr;*

*time\_t ttime;*

*myhandle \* phandle = NULL;*

*uac\_req\_t uac\_r;*

*uac\_req\_t uac\_r;*

*str hdr\_str,body\_str;*

*hdr\_str.s = hdr\_buf;*

*hdr\_str.len = 1024;*

*body\_str.s = body\_buf;*

*body\_str.len = 1024;*

*extra\_hdr.s = "";*

*extra\_hdr.len = 0;*

*ttime = time(NULL);*

*ctype.s = "text/plain";*

*ctype.len = strlen(ctype.s);*

*if(m\_build\_headers(&hdr\_str,ctype,\*from,ttime,extra\_hdr) < 0){*

*LM\_ERR("headers building failed by qiu \n");*

*return -1;*

*}*

*if(m\_build\_body(&body\_str,0,\*body,ttime) < 0){*

*LM\_ERR("error body \n");*

*return -1;*

*}*

*phandle = (myhandle\*)shm\_malloc(sizeof(myhandle));*

*if(phandle == NULL)*

*return -1;*

*phandle->\_callback = \_resultcallback;*

*phandle->param = userdata;*

*memset(&uac\_r,'\0',sizeof(uac\_r));*

*uac\_r.method = &msg\_type;*

*uac\_r.headers = &hdr\_str;*

*uac\_r.body =&body\_str;*

*uac\_r.cb\_flags = TMCB\_LOCAL\_COMPLETED;*

*uac\_r.cb = my\_tm\_callback;*

*uac\_r.cbp = (void\*)(long)phandle;*

*LM\_DBG("ljw output: \_callback 0x%08x --- 0x%08x \n",phandle->\_callback,\_resultcallback);*

*tmb.t\_request(&uac\_r,to,to,from,0);*

*return 1;*

*}*

添加回调函数m\_tm\_callback()，内容如下：

*void m\_tm\_callback( struct cell \*t, int type, struct tmcb\_params \*ps)*

*{*

*if(ps->param==NULL || \*ps->param==0)*

*{*

*LM\_DBG("message id not received\n");*

*goto done;*

*}*

*LM\_DBG("completed with status %d [mid: %ld/%d]\n",*

*ps->code, (long)ps->param, \*((int\*)ps->param));*

*if(!db\_con)*

*{*

*LM\_ERR("db\_con is NULL\n");*

*goto done;*

*}*

*if(ps->code >= 300)*

*{*

*LM\_DBG("message <%d> was not sent successfully\n", \*((int\*)ps->param));*

*msg\_list\_set\_flag(ml, \*((int\*)ps->param), MS\_MSG\_ERRO);*

*goto done;*

*}*

*LM\_DBG("message <%d> was sent successfully\n", \*((int\*)ps->param));*

*msg\_list\_set\_flag(ml, \*((int\*)ps->param), MS\_MSG\_DONE);*

*done:*

*return;*

*}*

然后修改msilo模块中的api.h，内容如下：

*/\* \* $Id$\*/*

*#ifndef \_MSILO\_API\_H\_*

*#define \_MSILO\_API\_H\_*

*//#include "../../sr\_module.h"*

*typedef int (\*send\_result\_callback)(int result,void\* userdata);*

*typedef int (\*msilo\_f)(struct sip\_msg\*, str\*);*

*typedef int (\*msilo\_f2)(str\* from,str\* to,str\* body,send\_result\_callback \_sendcallback,void\* userdata);*

*typedef struct msilo\_api {*

*msilo\_f m\_store;*

*msilo\_f m\_dump;*

*msilo\_f2 m\_dump2;*

*} msilo\_api\_t;*

*typedef int (\*bind\_msilo\_f)(msilo\_api\_t\* api);*

*/\*\**

*\* @brief Load the MSILO API*

*\*/*

*static inline int load\_msilo\_api(msilo\_api\_t \*api)*

*{*

*bind\_msilo\_f bindmsilo;*

*bindmsilo = (bind\_msilo\_f)find\_export("bind\_msilo", 1, 0);*

*if(bindmsilo == 0) {*

*LM\_ERR("cannot find bind\_msilo\n");*

*return -1;*

*}*

*if(bindmsilo(api)<0)*

*{*

*LM\_ERR("cannot bind msilo api\n");*

*return -1;*

*}*

*return 0;*

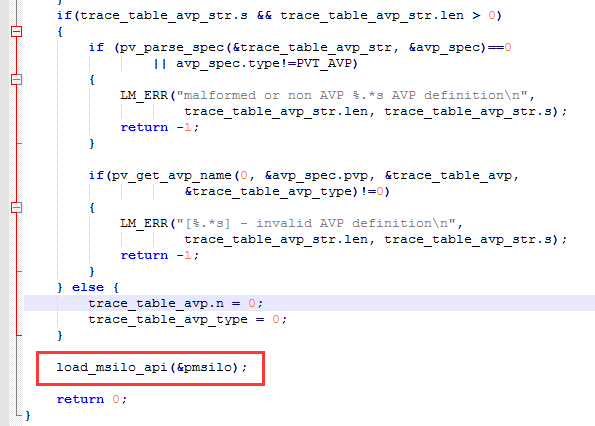
*}*

*#endif*

在siptrace模块中，修改siptrace.c文件，在该文件中包含msilo模块中的api.h头文件，然后在mod\_init()函数的最后添加如下内容：

*load\_msilo\_api(&pmsilo);*

如图所示：



添加Evy\_webCtrl()函数添加到siptrace.c中，Evy\_webCtrl()的内容如下：

*static int Evy\_webCtrl(struct sip\_msg\* msg, char\* webJson, char\* param){*

*char\* webMsg = webJson;*

*int webMsg\_len = strlen(webMsg);*

*LM\_DBG("webMsg = %s\n", webMsg);*

*LM\_DBG("webMsg\_len = %d\n",webMsg\_len);*

*int ret = -1;*

*int webMessage\_type = -1;*

*struct json\_object\* json\_type = NULL;*

*struct json\_object\* json\_body = NULL;*

*json\_body = json\_tokener\_parse(webJson);*

*if (unlikely(json\_body == NULL)) {*

*LM\_ERR("Error:json\_body = NULL!!!\n");*

*goto error;*

*}*

*if (unlikely(json\_object\_object\_get\_ex(json\_body,"type",&json\_type) == 0)) {*

*LM\_ERR("Error:not available message!!!\n");*

*goto error;*

*}*

*webMessage\_type = json\_object\_get\_int(json\_type);*

*switch(webMessage\_type) {*

*case 12:*

*ret = Evy\_parse\_webCtrl(json\_body,webMsg);*

*break;*

*case 13:*

*LM\_DBG("case = 13\n");*

*break;*

*default:*

*LM\_DBG("not available type!!!\n");*

*break;*

*}*

*json\_object\_put(json\_body);*

*return ret;*

*error:*

*json\_object\_put(json\_body);*

*return -1;*

*}*

添加Evy\_parse\_webCtrl()函数到siptrace.c中，Evy\_parse\_webCtrl()的内容如下：

*static int Evy\_parse\_webCtrl(struct json\_object\* json\_data,char \*webMessage) {*

*str webStr\_userid;*

*str webStr\_gatewayid;*

*str webStr\_cmd;*

*void\* userID;*

*struct json\_object\* webJson\_userid = NULL;*

*struct json\_object\* webJson\_gatewayid = NULL;*

*//struct json\_object\* webJson\_cmd = NULL;*

*if(unlikely(json\_object\_object\_get\_ex(json\_data,"userid", &webJson\_userid)) == 0) {*

*LM\_ERR("Error:webJson\_userid = NULL\n");*

*}*

*if(unlikely(json\_object\_object\_get\_ex(json\_data,"gatewayid", &webJson\_gatewayid)) == 0) {*

*LM\_ERR("Error:webJson\_gatewayid = NULL\n");*

*return -1;*

*}*

*webStr\_userid.s = (char\*)json\_object\_get\_string(webJson\_userid);*

*webStr\_userid.len = json\_object\_get\_string\_len(webJson\_userid);*

*webStr\_gatewayid.s = (char\*)json\_object\_get\_string(webJson\_gatewayid);*

*webStr\_gatewayid.len = json\_object\_get\_string\_len(webJson\_gatewayid);*

*webStr\_cmd.s = webMessage;*

*webStr\_cmd.len = strlen(webMessage);*

*LM\_DBG("webStr\_userid = %.\*s\n",webStr\_userid.len,webStr\_userid.s);*

*LM\_DBG("webStr\_gatewayid=%.\*s\n",webStr\_gatewayid.len,webStr\_gatewayid.s);*

*LM\_DBG("webStr\_cmd = %.\*s\n",webStr\_cmd.len,webStr\_cmd.s);*

*userID = (void\*)webStr\_userid.s;*

*pmsilo.m\_dump2(&webStr\_userid,&webStr\_gatewayid,&webStr\_cmd, callbackFunc, userID);*

*return 3;*

*}*

然后修改kamailio.cfg文件，将msilo模块加载进去，修改方式为：

*loadmodule "msilo.so"*

*# ---- msilio params -----*

*modparam("msilo","db\_url","mysql://kamailio:kamailiorw@localhost/kamailio")*

*modparam("msilo","from\_address","sip:registrar@kamailio.org")*

*modparam("msilo","contact\_hdr","Contact:registrar@192.168.1.19:5060;msilo=yes\r\n")*

*modparam("msilo","content\_type\_hdr","Content\_Type:text/plain\r\n")*

*modparam("msilo","offline\_message","\*\*\* User $rU is offline!")*

其中，192.168.1.19:5060是kamailio服务器的IP，需要根据具体情况而定。通过这样修改，即可将web发来的控制指令转发给对应的网关。

## Kamailio-🡪web返回值

web向Kamailio发送JSON之后，Kamailio需要向web返回结果，实现该部分需要修改Kamailio的配置文件，调用Kamailio的xhttp模块，修改内容如下：

*event\_route[xhttp:request] {*

*#rm= GET/POST hu= /sip/text rb= 123456*

*xdbg("http:request hu= $rm => $hu\n");*

*xdbg("http:request rb= $rm => $rb\n");*

*$var(webret) = Evy\_webCtrl($rb);*

*if ($var(webret) == 3 ) {*

*xhttp\_reply("200","{\"code\":\"200\",\"msg\":\"操作成功\"}",*

*"text/json","{\"code\":\"200\",\"msg\":\"操作成功\"}");*

*}else if($var(webret) == -1) {*

*xhttp\_reply("404","{\"code\":\"404\",\"msg\":\"操作失败\"}",*

*"text/json","{\"code\":\"404\",\"msg\":\"操作失败\"}");*

*}*

通过以上修改，web向Kamailio发送数据时，成功则收到*{"code":"200","msg":"操作成功"}*，失败则收到*{"code":"404","msg":"操作失败"}*，至此，web控制的功能完成。

## 备注

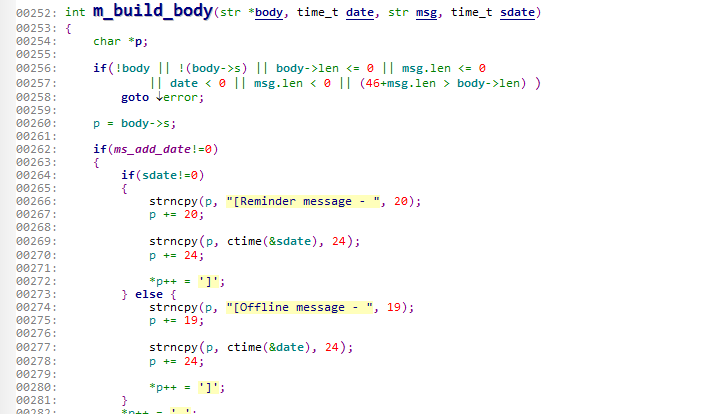
web向Kamailio发送的JSON数据格式如下：

*{"userid":"sip:1000@121.42.202.152","gatewayid":"sip:1001@121.42.202.152","type":"12","msg":{"value":"100","ctrlid":"385241f9-cded-4cc4-bd42-01c1ac8ef4ca","time":"2016\/03\/09"}}*

其中，userid和gatewayid的内容必须符合上面的格式，否则客户端接收不到，发送会失败。1000和1001是Kamailio的客户端的SIP账号，121.42.202.152是Kamailio服务器的IP。

# 修改Message的前缀

kamailio默认会在某些类型的message body前面增加特定的前缀，例如会在离线消息的body前添加“Offline message +日期”这样的内容，如果要想将前缀删除，需要修改msilo模块的msfunc.c文件，如图：



将需要去掉的前缀的代码注释掉即可。

# 不同版本源码的区别

**9.1**  Kamailio 4.1.2🡪 Kamailio 4.2.6的变化，目前遇到的不同之处如下所述，

（1）printf改为rpl\_printf

（2）ERROR\_LOG改为LM\_ERR

（3）LM\_MSG改为LM\_INFO

如果不修改的话，在重新编译siptrace模块的时候会报错。

**9.2** Kamailio 4.2.6🡪 Kamailio 4.3.2的变化

Kamailio.pid文件所在的路径发生了变化，由原来的/var/run/ Kamailio.pid变为了/var/run/ Kamailio / Kamailio.pid