目录

目录
需求分析
- 要求
- 架构设计
- 工作计划

前端
- 用户交互
- 接口定义

后端
- 接口定义
  - 用户
    - 登陆
    - 注销
    - 密码修改
  - 系统状态
    - 运行时间
    - 负载
    - 存储空间
  - 网络
    - PING
    - 测速
    - 网络状态
    - 网络设置
    - Wifi-列出可连接的网络
    - Wifi-连接网络
    - Wifi-热点状态
    - Wifi-热点开启
    - Wifi-热点关闭
  - 设备
    - 重启
    - 定时重启-开启
    - 定时重启-关闭
    - 固件升级
    - 快速复位
    - 设备名
    - 设备名-修改
  - 视频源
    - 信号-状态
    - 裁切-状态
    - 裁切-设置
    - 源选择和色彩-状态
    - 源选择和色彩-设置
  - 音频源
    - 状态
    - 源选择和音量-状态
    - 源选择和音量-设置
    - 编码引擎-状态
    - 编码引擎-设置
  - 流媒体
    - RTSP-状态
    - RTSP-设置
    - RTSP-用户-列表

- RTSP-用户-添加
- RTSP-用户-删除
- MJPEG流-状态
- MJPEG流-编码器-状态
- MJPEG流-编码器-设置
- 主/子码流-状态
- 主/子码流-编码-状态
- 主/子码流-编码-设置
- 主/子码流-录像-策略-状态
- 主/子码流-录像-策略-设置
- 主/子码流-录像-状态
- 主/子码流-录像-打开
- 主/子码流-录像-关闭
- 主/子码流-录像-列表
- 叠加层-状态
- 叠加层-打开
- 叠加层-关闭
- 叠加层-项-列表
- 叠加层-项-添加
- 叠加层-项-修改
- 叠加层-项-删除
- 叠加层-图片-列表
- 叠加层-图片-添加
- 叠加层-图片-删除

- 储存
  - 磁盘-列表
  - 磁盘CIFS-添加
  - 磁盘CIFS-删除
- 网络服务
  - WEB服务-状态
  - WEB服务-设置
  - TELNET服务-状态
  - TELNET服务-开启
  - TELNET服务-关闭
  - 静态ARP-查看
  - 静态ARP-添加
  - 静态ARP-删除
- 区域时间
  - 状态
  - 时间-设置
  - 时区-设置
  - 网络时间同步-开启
  - 网络时间同步-关闭

- C++端
  - 媒体源服务
    - 视频源采集模块
    - 音频源采集模块
    - 分流模块
  - 编&解码服务
    - 音视频复用
    - 视频色彩调节
    - 视频文字叠加
    - 视频图片叠加
    - 视频时间叠加
    - 视频裁切
    - 视频缩放
    - 视频旋转
    - 视频镜像

音频音量调节
流媒体服务
录像模块
SRT模块
RTSP模块
RTMP模块
TS-UDP模块
HLS模块
串口&USB服务

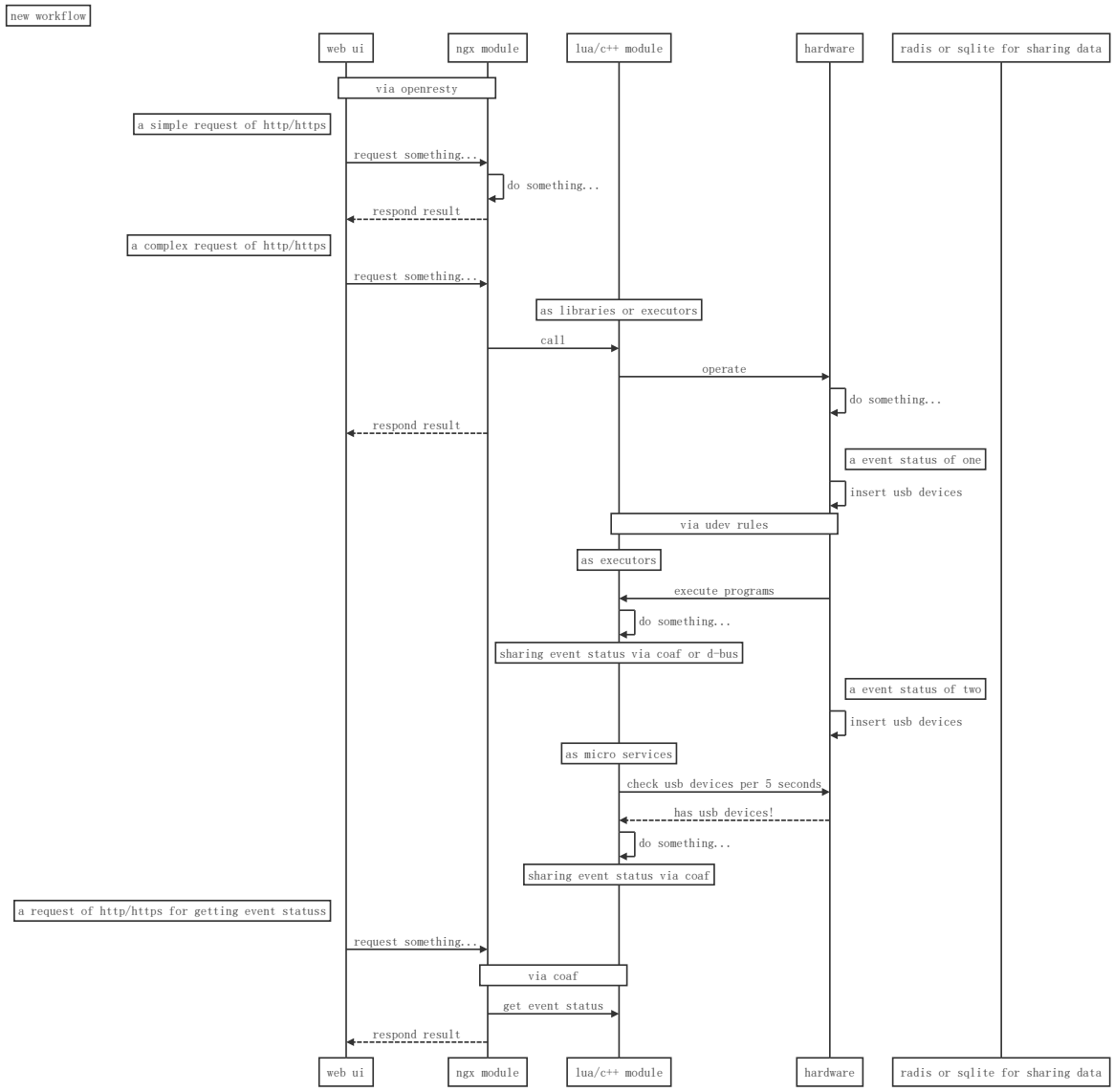
需求分析

要求

- 编写设计&接口文档。
- 重构前&后端接口。
- 简化代码&依赖。
- 规范命名法。

架构设计

- 工程代码三层归类，界面层（Web UI）+业务层（Ngx Module）+基础层（Lua/C++ Module）。
- 面向接口编写代码，减少代码封装层次，尽量简单直接。
- 简单的请求，不要涉及Coaf调用，减少代码量。
- Lua/C++模块之间减少依赖调用，保持简单&独立性，避免Coaf调用阻塞。
- 弃用D-Bus框架，仅使用Coaf框架进行RPC通信，降低复杂度。


new workflow
web ui
ngx module
lua/c++ module
hardware
radis or sqlite for sharing data
via openresty
a simple request of http/https
request something...
do something...
respond result
a complex request of http/https
request something...
as libraries or executors
call
operate
do something...
respond result
a event status of one
insert usb devices
via udev rules
as executors
execute programs
do something...
sharing event status via coaf or d-bus
a event status of two
insert usb devices
as micro services
check usb devices per 5 seconds
has usb devices!
do something...
sharing event status via coaf
a request of http/https for getting event statuss
request something...
via coaf
get event status
respond result
web ui
ngx module
lua/c++ module
hardware
radis or sqlite for sharing data

工作计划

项目	任务	问题	人员	时间
G2-Encoder	Web界面			
	Ngx模块-用户	接口-用户-登陆		
		接口-用户-修改密码		
		接口-用户-登出		
	Ngx模块-系统状态	接口-状态-运行时间		
		接口-状态-系统负载		
		接口-状态-存储空间		
	Ngx模块-网络	接口-网络-PING		
		接口-网络-网口测速		
		接口-网络-网络状态		
		接口-网络-Wifi-查找可连接的网络		
		接口-网络-Wifi-连接网络		
		接口-网络-Wifi-热点		
		接口-网络-网络设置		
	Ngx模块-设备	接口-设备-重启		
		接口-设备-定时重启		
		接口-设备-固件升级		
		接口-设备-快速复位		
		接口-设备-设备名		
	Ngx模块-视频	接口-视频-视频信号		
		接口-视频-裁切		
		接口-视频-源选择和色彩调节		
	Ngx模块-音频	接口-音频-音频信号		
		接口-音频-源选择与音量调节		

项目	任务	问题	人员	时间
		接口-音频-编码引擎状态		
	Ngx模块-流媒体	接口-流媒体-管理		
		接口-RTSP服务状态		
		接口-RTSP服务用户管理		
		接口-RTSP服务参数设置		
		接口-MJPEG流-状态		
		接口-MJPEG流-编码器-状态		
		接口-MJPEG流-编码器-设置		
		接口-H.264主/子码流-状态		
		接口-H.264主/子码流-编码器-状态		
		接口-H.264主/子码流-编码器-设置		
		接口-H.264主/子码流-录像-状态		
		接口-H.264主/子码流-录像-打开		
		接口-H.264主/子码流-录像-关闭		
		接口-H.264主/子码流-录像-策略-状态		
		接口-H.264主/子码流-录像-策略-设置		
		接口-H.264主/子码流-录像-列表		
	Ngx模块-区域时间	接口-区域时间-时间		
		接口-区域时间-位置		
		接口-区域时间-网络时间同步		
	Ngx模块-储存	接口-储存-状态		
		接口-储存CIFS-添加		
		接口-储存CIFS-删除		
	Ngx模块-服务	接口-服务-WEB服务		

项目	任务	问题	人员	时间
		接口-服务-Telnet服务		
		接口-服务-静态ARP服务		
	Lua/C++模块	流媒体服务		
		编码器服务		
		串口&USB服务		

前端

用户交互

接口定义

后端

接口定义

用户

登陆

```
HTTP GET /user/login
#输入
account: 'zhangsan'      #用户名
password: 'xf.sky.1'     #密码
#输出
{
    err: null,               #成功null，否则错误信息
    val: 'token123',         #登陆令牌
}
```

注销

```
HTTP GET /user/logout
#输入
#输出
{
    err: null,                   #成功null，否则错误信息
    val: true
}
```

密码修改

```
HTTP GET /user/password
#输入
password: 'xfskyl'       #旧密码
new_password: 'xfskyl2' #新密码
#输出
{
    err: null,                  #成功null，否则错误信息
    val: true
}
```

系统状态

运行时间

```
HTTP GET /system-status/persisted-time
#输入
#输出
{
    err: null,                  #成功null，否则错误信息
    val: 1231                   #运行的秒数
}
```

```
#命令
cat /proc/uptime
```

负载

```
HTTP GET /system-status/load
#输入
#输出
{
    err: null,                  #成功null，否则错误信息
    val: {
        cup: '35.9%',               #CPU负载百分比
        memory: {               #内存负载
            total: '15Gi',          #总共
            used: '4.1Gi',          #已用
            free: '7.5Gi',          #可用
            shared:'577Mi',         #共享
            buff_cache:'3.9Gi',     #缓存
            available:'10Gi',       #总共可用
        }
    }
}
```

```
#命令
top -n 1 | grep Cpu
free -h | grep Mem
```

存储空间

```
HTTP GET /system-status/disk-spaces
#输入
#输出
{
    err: null,                  #成功null，否则错误信息
    val: {
        data: {                 #数据储存空间
            size: '112.3M',     #总计
            used: '79.1M',      #已用
            available: '33.3M',         #可用
            use: '70%'          #使用负载
        },
        sys: {                  #系统储存空间
            size: '73.8M',
            used: '11.6M',
            available: '62.2M',
            use: '16%'
        }
    }
}
```

```
#命令
df -h | egrep '/$|/data$'
```

网络

PING

```
HTTP GET /network/ping
#输入
host: '192.168.1.1'     #目标主机IP
either_interface: ''    #可选，网卡名或网卡IP
#输出
{
    err: null,      #成功null，否则错误信息
    val: '--- 192.168.1.1 ping statistics ---                          \
          3 packets transmitted, 3 packets received, 0% packet loss \
          round-trip min/avg/max = 0.310/0.350/0.410 ms'        #测试报告，丢包百分比,响应时间（毫秒）
}
```

```
#命令
ping <目标主机IP> -W 1 -c 3 -q -I <网卡名或网卡IP>
```

测速

```
HTTP GET /network/speed
#输入
host: '192.168.1.1'     #目标主机IP
#输出
{
    err: null,          #成功null，否则错误信息
    val: '[ ID] Interval         Transfer     Bandwidth       Retr          \
          [  4] 0.00-4.00   sec  39.9 MBytes  83.6 Mbits/sec  0     sender  \
          [  4] 0.00-4.00   sec  39.5 MBytes  82.8 Mbits/sec        receiver'
#失败null

}
```

```
#命令
iperf3 -s   #作为服务运行
iperf3 -c <目标主机IP> [-b <网卡带宽>]  #客户端运行
```

网络状态

```
HTTP GET /network/status
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: {      #失败null
        ethernet: [     #有线网络
            {
                device: 'eth0',     #设备名
                hardware_address: 'de:10:79:11:eb:22',  #网卡地址
                state: 'connected', #连接状态
                name: 'ethernet-eth0',                  #连接名，修改配置用
                uuid: 'd873cce7-4bf7-4e04-89d7-2263fe2bc47a', #uuid，修改配置用，如果连接名重名，则用uuid,
                ipv4_method: 'manual',                          #地址获取方式，manual：手动，auto：自动
                ipv4_addresses: ['192.168.1.168/24','192.168.2.174/24'], #网络地址，含子网掩码
                ipv4_gateway: '192.168.2.1',    #网关
                dns: ['8.8.8.8'],               #dns地址，存在多个
            }
        ]
        wifi: [         #无线网络，存在多个设备
            {
                device: 'wlan0',
                hardware_address: '54:c9:df:9f:a8:d7',
                state: 'disconnected',
                name:'',
                uuid: '',
                ipv4_method: '',
                ipv4_addresses: [],
                ipv4_gateway: '',
                dns: []
            },{
                device: 'wlan1',
                hardware_address: '56:c9:df:9f:a8:d7',
                state: 'unmanaged',
```

```
                name:'',
                uuid: '',
                ipv4_method: '',
                ipv4_addresses: ['192.168.250.254/24'],
                ipv4_gateway: '',
                dns: []
            }
        ],
        usb: [          #USB网络
            {
                device: 'usb0',
                hardware_address: 'F6:3F:F6:0B:18:25',
                state: 'unmanaged',
                name:'usb0',
                uuid: 'd873cce7-4bf7-4e04-89d7-2263fe2bc47a',
                ipv4_method: 'manual',
                ipv4_addresses: ['10.10.10.10/24'],
                ipv4_gateway: '',
                dns: []
        ]
    }
}
```

```
#命令
nmcli device status
nmcli device show <设备名>
nmcli connection show <连接名|uuid <uuid>> | grep ipv4.method
```

网络设置

```
HTTP POST /network/setup
Content-Type: application/x-www-form-urlencoded
#输入
clone_mac: 'F6:3F:F6:0B:18:25'      #克隆MAC
ipv4_method: 'manual'               #地址获取方式，manual：手动，auto：自动，如果为
auto，所有ipv4_*的参数都忽略
ipv4_address: '192.168.1.190/24'    #网络地址/子网掩码
ipv4_gateway: '192.168.1.1'         #网关
ipv4_dns1: '8.8.8.8'                #网络解析地址
ipv4_dns2: '114.114.114.114'        #可选，网络解析地址
#输出
{
    err: null,      #成功null，否则错误信息
    val: true
}
```

```
#命令
nmcli connection show <连接名>
nmcli connection modify <连接名|uuid <uuid>> <key> <value>
```

Wifi-列出可连接的网络

```
HTTP GET /network/wifi-list
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: [
        {
            ssid: 'Kiloview03',      #网络名
            signal: 92,              #信号强度，百分比
            bars: '*****'            #信号强度，形象化
        },{
            ssid: 'HUAWEI-3920',
            signal: 64,
            bars: '****'
        },{
            ssid: 'NETGEAR_5G',
            signal: 47,
            bars: '**'
        }
    ]
}
```

```
#命令
nmcli device wifi list
```

Wifi-连接网络

```
HTTP GET /network/wifi-list
#输入
ssid: 'HUAWEI-3920'              # 网络名
password: 'xf.sky.1'             # 密码
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#命令
nmcli device wifi connect <ssid> <password>
```

Wifi-热点状态

```
HTTP GET /network/wifi-ap-status
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: ??
}
#命令
? ?
```

Wifi-热点开启

```
HTTP GET /network/wifi-ap-start
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#配置文件路径 /etc/hostapd/hostapd.conf
ctrl_interface=/var/run/hostapd
ctrl_interface_group=1
interface=wlan1
driver=nl80211
ssid=CAST_AP
channel=6
hw_mode=g
ignore_broadcast_ssid=0
auth_algs=1
wpa=3
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP

#命令
hostapd -B <配置文件路径>
```

Wifi-热点关闭

```
HTTP GET /network/wifi-ap-stop
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#命令
killall hostapd
```

设备

重启

```
HTTP GET /device/reboot
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
#命令
reboot
```

定时重启-开启

```
HTTP GET /device/period-reboot-open
#输入
hours: 5            # 每天的重启时间，小时：0~23
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#配置文件 reboot.task
* * * * * reboot
#命令
crontab reboot.task
```

定时重启-关闭

```
HTTP GET /device/period-reboot-close
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#命令
crontab -r
```

固件升级

```
HTTP POST /device/upgrade
Content-Type: application/x-www-form-urlencoded
#输入
filename: ''
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#命令
运行升级包内的'pre-upgrade.sh'
```

快速复位

```
HTTP GET /device/reset
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
--实现
function _M.Reset()
    local streamerProxy = GetCoAFProxy( "/", "streamer", "local", 1 )
    local codecProxy = GetCoAFProxy( "/Codec", "codec", "local", 1 )

    if streamerProxy then
        streamerProxy:Reset()
        streamerProxy:Destroy()
    end

    if codecProxy then
        codecProxy:Reset()
        codecProxy:Destroy()
    end
end
```

设备名

```
HTTP GET /device/hostname
#输入
#输出
{
    err: null,  #成功null，否则错误信息
    val: 'fedora-truman'    #主机名
}
```

```
#命令
cat /etc/hostname
```

设备名-修改

```
HTTP GET /device/hostname
#输入
name: 'fedora-truman'
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
#命令
echo <设备名> > /etc/hostname
```

视频源

信号-状态

```
HTTP GET /codec/video-status
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        video_freerun: 0,     #?
        video_frame_rate: 60, #视频帧率
        video_source: 'HDMI', #视频源，HDMI：高清接口
        video_source_detected: 1, #?
        video_format_seq: '', #?
        video_format_code: '1080p60', #显示格式&帧率
        video_format_name: '1080p 60Hz', #显示格式&帧率
        video_format: '1080p 60Hz', #显示格式&帧率
        video_interlaced: 0, # ?
        video_standard: 'HD', # 高清标准?
        video_width: '1920', #分辨率w
        video_height: '1080', #分辨率h
        video_locked: 1 #锁定状态，0：已锁定，1：未锁定
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers['codec/AVIO/00/VFE'] = codec:CreateProxy('/AVIO/00/VFE')
r, count, table = servers["codec/AVIO/00/AFE"]:Control("detect-source", {source
= "any"})
--结果
r = true
count = 8
table = {
    ["video.frame_rate"] = 60,
    ["video.source"] = "HDMI",
    ["video.freerun"] = 0,
    ["video.source_detected"] = 1,
    ["video.format_code"] = "1080p60",
    ["video.format"] = "1080p 60Hz",
    ["video.format_name"] = "1080p 60Hz",
    ["video.interlaced"] = 0,
    ["video.standard"] = "HD",
    ["video.width"] = 1920,
    ["video.format_seq"] = 0,
    ["video.height"] = 1080,
    ["video.locked"] = 1
}
```

裁切-状态

```
HTTP GET /codec/video-cut-status
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        source_crop_enable: 0,       #源图像裁切，0：不裁切，1：裁切，如果0,则忽略
source_crop*参数
        source_cropX: 0.0,           #左
        source_cropY: 0.0,           #上
        source_cropW: 240.0,         #右
        source_cropH: 240.0,         #下
        spec_out_format: "default", #缩放到特定分辨率
                                     # default
                                     # VIDOUT_1920x1080p60，VIDOUT_1920x1080p59，
VIDOUT_1920x1080p50
                                     # VIDOUT_1920x1080p30，VIDOUT_1920x1080p25
                                     # VIDOUT_1280x720p60，VIDOUT_1280x720p50，
VIDOUT_1280x720p30
                                     # VIDOUT_1280x720p25，VIDOUT_1280x1024p60
                                     # VIDOUT_640x480p60，VIDOUT_1024x768p60，
VIDOUT_1440x900p60
        spec_out_mode: "default",    #缩放模式
                                     # default：默认
                                     # OUTCTRL_ASPECT：自适应并维持纵横比
                                     # OUTCTRL_P2P：居中
                                     # OUTCTRL_STRETCH：拉伸到全屏
        out_crop_enable: 0,          #输出后图像裁切，0：不裁切，1：裁切，如果0,则忽略
out_crop*参数
        out_cropY: 0.0,              #左
        out_cropX: 0.0,              #上
        out_cropW: 0.0,              #右
        out_cropH: 0.0,              #下
        RMF: "0"                     #画面翻转，0：默认，90：90度，180：180度，270：270
度，mirror：水平镜像，flip：垂直翻转
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
codec = client.New("codec", "local", "192.168.2.174:14000")
servers['codec/AVIO'] = codec:CreateProxy('/AVIO')
params = {
    ["avio.video.source_crop.enable"] = 0,
    ["avio.video.source_crop.cropX"] = 0,
    ["avio.video.source_crop.cropY"] = 0,
    ["avio.video.source_crop.cropW"] = 0,
    ["avio.video.source_crop.cropH"] = 0,
    ["avio.video.spec_out.format"] = ",
    ["avio.video.spec_out.mode"] = "",
    ["avio.video.out_crop.enable"] = 0,
    ["avio.video.out_crop.cropX"] = 0,
    ["avio.video.out_crop.cropY"] = 0,
```

```
    ["avio.video.out_crop.cropW"] = 0,
    ["avio.video.out_crop.cropH"] = 0,
    ["avio.video.RMF"] = "0"
}
r, count, table = servers["codec/AVIO"]:Get(params)
--结果
r = true
count = 13
table = {
    ["avio.video.out_crop.cropH"] = 0.0,
    ["avio.video.out_crop.cropY"] = 0.0,
    ["avio.video.out_crop.cropX"] = 0.0,
    ["avio.video.source_crop.cropH"] = 240.0,
    ["avio.video.source_crop.cropX"] = 0.0,
    ["avio.video.source_crop.cropY"] = 0.0,
    ["avio.video.out_crop.cropW"] = 0.0,
    ["avio.video.out_crop.enable"] = 0,
    ["avio.video.source_crop.enable"] = 0,
    ["avio.video.spec_out.format"] = "default",
    ["avio.video.source_crop.cropW"] = 240.0,
    ["avio.video.spec_out.mode"] = "default",
    ["avio.video.RMF"] = "0"
}
```

裁切-设置

```
HTTP GET /codec/video-cut-set
#输入
source_crop_enable: 0        #源图像裁切，0：不裁切，1：裁切，如果0,则忽略source_crop*参数
source_cropX: 0.0            #左
source_cropY: 0.0            #上
source_cropW: 240.0          #右
source_cropH: 240.0          #下
spec_out_format: "default"   #缩放到特定分辨率
                             # default
                             # VIDOUT_1920x1080p60，VIDOUT_1920x1080p59，VIDOUT_1920x1080p50
                             # VIDOUT_1920x1080p30，VIDOUT_1920x1080p25
                             # VIDOUT_1280x720p60，VIDOUT_1280x720p50，VIDOUT_1280x720p30
                             # VIDOUT_1280x720p25，VIDOUT_1280x1024p60
                             # VIDOUT_640x480p60，VIDOUT_1024x768p60，VIDOUT_1440x900p60
spec_out_mode: "default"    #缩放模式
                             # default：默认
                             # OUTCTRL_ASPECT：自适应并维持纵横比
                             # OUTCTRL_P2P：居中
                             # OUTCTRL_STRETCH：拉伸到全屏
out_crop_enable: 0           #输出后图像裁切，0：不裁切，1：裁切，如果0,则忽略out_crop*参数
out_cropY: 0.0               #左
out_cropX: 0.0               #上
out_cropW: 0.0               #右
out_cropH: 0.0               #下
RMF: "0"                     #画面翻转，0：默认，90：90度，180：180度，270：270度，mirror：水平镜像，flip：垂直翻转
```

```
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers['codec/AVIO'] = codec:CreateProxy('/AVIO')

params = {
    ["avio.video.source_crop.enable"] = 0,
    ["avio.video.source_crop.cropX"] = 0,
    ["avio.video.source_crop.cropY"] = 0,
    ["avio.video.source_crop.cropW"] = 240,
    ["avio.video.source_crop.cropH"] = 240,
    ["avio.video.spec_out.format"] = "default",
    ["avio.video.spec_out.mode"] = "default",
    ["avio.video.out_crop.enable"] = 0,
    ["avio.video.out_crop.cropX"] = 0,
    ["avio.video.out_crop.cropY"] = 0,
    ["avio.video.out_crop.cropW"] = 0,
    ["avio.video.out_crop.cropH"] = 0,
    ["avio.video.RMF"] = "0"
}

r, count = servers["codec/AVIO"]:Set(params)
--结果
r = true
count = 13
```

源选择和色彩-状态

```
HTTP GET /codec/video-source-status
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        saturation: 128,    #饱和度 0～255
        hue: 128,           #色度 0～255
        source: 'AUTO',     #视频源，AUTH：自动选择，HDMI：高清源
        brightness: 128,    #亮度 0～255
        contrast: 128       #对比度 0～255
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
```

```
servers = { ['codec/AVIO'] = codec:CreateProxy('/AVIO') }

params = {
    ["avio.video.source"] = "",
    ["avio.video.brightness"] = 0,
    ["avio.video.contrast"] = 0,
    ["avio.video.saturation"] = 0,
    ["avio.video.hue"] = 0
}
r, count, table = servers["codec/AVIO"]:Get(params)
--结果
r = true
count = 5
table = {
    ["avio.video.saturation"] = 128,
    ["avio.video.hue"] = 128,
    ["avio.video.source"] = "AUTO",
    ["avio.video.brightness"] = 128,
    ["avio.video.contrast"] = 128
}
```

源选择和色彩-设置

```
HTTP GET /codec/video-source-set
#输入
saturation: 128 #饱和度 0～255
hue: 128        #色度 0～255
source: 'AUTO'  #视频源，AUTH：自动选择，HDMI：高清源
brightness: 128 #亮度 0～255
contrast: 128   #对比度 0～255
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/AVIO'] = codec:CreateProxy('/AVIO') }

params = {
    ["avio.video.saturation"] = 128,
    ["avio.video.hue"] = 128,
    ["avio.video.source"] = "AUTO",
    ["avio.video.brightness"] = 128,
    ["avio.video.contrast"] = 128
}
r, count, table = servers["codec/AVIO"]:Set(params)
--结果
r = true
count = 5
```

音频源

状态

```
HTTP GET /codec/audio-status
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        audio_sampling: 48000,  #采样率
        audio_sample_size: 0,   #采样大小，？
        audio_sample_width: 16, #编码位宽，值越大质量越好，8：单字节，16：两字节，32：四字节
        audio_source: 'DIGITAL', #采样源，'DIGITAL'：数字源
        audio_channels: 2,      #音频声道数，1：单声道，2：双声道
        audio_locked: 1         #锁定状态，0：已锁定，1：未锁定
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/AVIO/00/AFE'] = codec:CreateProxy('/AVIO/00/AFE') }
r, count, table = servers["codec/AVIO/00/AFE"]:Control("detect-source", {source = "any"})
--结果
r = true
count = 6
table = {
    ["audio.sampling"] = 48000,
    ["audio.sample_size"] = 0,
    ["audio.source"] = "DIGITAL",
    ["audio.channels"] = 2,
    ["audio.locked"] = 0,
    ["audio.sample_width"] = 16
}
```

源选择和音量-状态

```
HTTP GET /codec/audio-line-status
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        source: 'AUTO',     #音频源，自动选择：自动根据输入视频的选择音频源。HDMI：从HDMI获取，LINE：从外接线
        line_gain: '0db'    #音频增益，0db表示原始的音量；+/-3db表示增加/减少50%音频，+/-6db表示增加/减少1倍的音频。
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/AVIO'] = codec:CreateProxy('/AVIO') }

params = {
    ["avio.audio.source"] = "",
    ["avio.audio.line_gain"] = 0,
    ["avio.audio.mic_gain"] = 0
}

r, count, table = servers["codec/AVIO"]:Get(params)
--结果
r = true
count = 3
table = {
    ["avio.audio.source"] = "AUTO",
    ["avio.audio.mic_gain"] = 0.0,
    ["avio.audio.line_gain"] = 0.0
}
```

源选择和音量-设置

```
HTTP GET /codec/audio-line-set
#输入
source: 'AUTO'       #音频源，自动选择：自动根据输入视频的选择音频源。HDMI：从HDMI获取，
LINE：从外接线
                     # 这个原则是：如果视频输入信号带有内嵌音频（如HDMI/SDI），则自动从该视
频输入源中选择音频；
                     # 否则自动选择外接的模拟音频输入。
line_gain: '0db'     #音频增益，0db表示原始的音量；+/-3db表示增加/减少50%音频，+/-6db表示
增加/减少1倍的音频。
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/AVIO'] = codec:CreateProxy('/AVIO') }

params = {
    ["avio.audio.source"] = "AUTO",
    ["avio.audio.line_gain"] = 0,
    ["avio.audio.mic_gain"] = 0
}

r, count, table = servers["codec/AVIO"]:Set(params)
--结果
r = true
```

```
count = 3
```

编码引擎-状态

```
HTTP GET /codec/audio-engine-status
#输入
channel: 1 #音频编码通道，1：通道1，2：通道2，通道3
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        enable: 0,
        source_type: "ALSA",
        source_device: "default",
        source_resample: "fastest",
        encoder_codec: "AAC",
        encoder_sampling: 48000,
        encoder_channels: 2,
        encoder_bitrate: 64000,
        encoder_aac_type: "AAC-LC",
        encoder_aac_format: "RAW",
        encoder_g711_law: "ULAW"
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("streamer", "local", "192.168.2.174:14000")
servers = { ['streamer/AudioEngine'] = codec:CreateProxy('/AudioEngine') }

channel = 1
params = {
    [ "audioEngine.encoders[" .. channel .. "].enable" ] = 0,
    [ "audioEngine.encoders[" .. channel .. "].source.type" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].source.device" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].source.resample" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].encoder.codec" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].encoder.sampling" ] = 0,
    [ "audioEngine.encoders[" .. channel .. "].encoder.channels" ] = 0,
    [ "audioEngine.encoders[" .. channel .. "].encoder.bitrate" ] = 0,
    [ "audioEngine.encoders[" .. channel .. "].encoder.aac.type" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].encoder.aac.format" ] = "",
    [ "audioEngine.encoders[" .. channel .. "].encoder.g711.law" ] = ""
}
r, count, table = servers["streamer/AudioEngine"]:Get(params)
--结果
r = true
count = 11
table = {
    ["audioEngine.encoders[1].source.resample"] = "fastest",
    ["audioEngine.encoders[1].encoder.sampling"] = 48000,
    ["audioEngine.encoders[1].encoder.g711.law"] = "ULAW",
    ["audioEngine.encoders[1].source.device"] = "default",
    ["audioEngine.encoders[1].encoder.channels"] = 1,
```

```
    ["audioEngine.encoders[1].encoder.bitrate"] = 64000,
    ["audioEngine.encoders[1].enable"] = 1,
    ["audioEngine.encoders[1].encoder.codec"] = "AAC",
    ["audioEngine.encoders[1].encoder.aac.format"] = "ADTS",
    ["audioEngine.encoders[1].source.type"] = "ALSA",
    ["audioEngine.encoders[1].encoder.aac.type"] = "AAC-LC"
}
```

编码引擎-设置

```
HTTP PUT /codec/audio-engine-set
#输入
channel: 1                          #音频编码通道，1：通道1，2：通道2，通道3
enable: 1                           #启用音频编码通道，0：不启用，1：启用
source_type: "ALSA"                 #?
source_device: "default"            #?
source_resample: "fastest"          #重采样策略，fastest：快速/音质一般，common：高音质/较
高CPU消耗
encoder_codec: "AAC"                #编码，AAC，G711，如果是AAC,则忽略encoder_g711_*的参
数，否则忽略encoder_aac_*
encoder_sampling: 48000             #采样率，96000，88200，64000，48000，44100，32000，
22050,16000,8000
                                    # 编码G.711固定为8000
encoder_bitrate: 64000              #码率
                                    # 16000，24000，32000，48000，64000，72000
                                    # 80000，96000，128000，160000，192000，256000
                                    # 编码G.711固定为64000
encoder_channels: 2                 #声道数，1：单声道，2：立体声
                                    # 编码G.711固定为单声道
encoder_aac_type: "AAC-LC"          #编码AAC规则，AAC-LC：简单的，AAC-HE：高效的
encoder_aac_format: "RAW"           #编码AAC格式，RAW：原始格式
encoder_g711_law: "ULAW"            #编码G711规则，ULAW，ALAW
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("streamer", "local", "192.168.2.174:14000")
servers = { ['streamer/AudioEngine'] = codec:CreateProxy('/AudioEngine') }

channel = 1 --音频编码通道
params = {
    [ "audioEngine.encoders[" .. channel .. "].enable" ] = 1,
    [ "audioEngine.encoders[" .. channel .. "].source.type" ] = "ALSA",
    [ "audioEngine.encoders[" .. channel .. "].source.device" ] = "default",
    [ "audioEngine.encoders[" .. channel .. "].source.resample" ] = "fastest",
    [ "audioEngine.encoders[" .. channel .. "].encoder.codec" ] = "AAC",
    [ "audioEngine.encoders[" .. channel .. "].encoder.sampling" ] = 48000
    [ "audioEngine.encoders[" .. channel .. "].encoder.bitrate" ] = 64000,
    [ "audioEngine.encoders[" .. channel .. "].encoder.channels" ] = 2,
    [ "audioEngine.encoders[" .. channel .. "].encoder.aac.type" ] = "AAC-LC",
```

```
    [ "audioEngine.encoders[" .. channel .. "].encoder.aac.format" ] = "ADTS",
    [ "audioEngine.encoders[" .. channel .. "].encoder.g711.law" ] = "ULAW"
}
r, count, table = servers["streamer/AudioEngine"]:Set(params)
--实现结果
r = true
count = 11
```

流媒体

RTSP-状态

```
HTTP GET /stream-media/rtsp-status
#输入
stream_id: 2    #流编号，0~2，0：主码流，1：字码流，2：图片流
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        status: 'online',                              #流服务是否在线
        enabled: true,                                 #流服务是否启用
        service_uri: 'rtsp://%HOST%:554/mjpeg01',    #流服务地址，用正确的IP替换%HOST%
        service_type: 'RTSP/SERVER',      #流服务类型
        rtsp_server_port: 554,            #服务端口，默认554
        rtsp_server_http_tunnel_port: 0, #HTTP Tunnel端口，Tunnel端口保持为0，表示不开启HTTP Tunnel功能。
                                          # 注意HTTP Tunnel端口不要与其他服务端口相同，默认1080。
                                          # 修改HTTP Tunnel端口，仅当复位或重启后才能生效
        rtsp_server_session: 'mjpeg01',  #会话编号
        rtsp_server_no_adts: 0,       #对AAC音频策略，0：保持默认格式，1：去除ADTS头
        rtsp_auth: 0,                #RTSP身份认证，0：关闭，1：开启
        rtsp_server_ssm: 0,      #组播，0：关闭，1：开启，组播会话将固定为'<会话(Session) ID>/ssm'，即:mjpeg01/ssm
        rtsp_server_multicast_addr: "224.0.0.1", #若rtsp_server_ssm为1，则生效
                                                  # 组播地址，224.x.x.x～239.x.x.x
        rtsp_server_multicast_ttl: 127, #若rtsp_server_ssm为1，则生效
                                          # Time to Live 生存时间1～255，决定允许组播数据通过路由节点的个数。
                                          # TTL值越大表示允许通过路由节点的个数越多，根据实际网络环境更改值，默认值127
        rtsp_server_multicast_video_port: 3000, #若rtsp_server_ssm为1，则生效
                                                 # 组播视频端口
        rtsp_server_multicast_audio_port: 3002 #若rtsp_server_ssm为1，则生效
                                                 # 组播音频端口
    }
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
stream_id = 2 -- 流编号，0~2，0：主码流，1：字码流，2：图片流？

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')
```

```
r,count,table = streamer_DefaultApp:Get({
    ["streamer.rtsp_auth"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.port"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.httpTunnelPort"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.session"] = "",
    ["streamer.stream[".stream_id."].Rtsp_server.ssm"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_addr"] = "",
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_video_port"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_audio_port"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_ttl"] = 127,
    ["streamer.stream[".stream_id."].Rtsp_server.no_adts"] = 0
})

-- print(r,count,table)
-- print_table(table)

-- true    10      table: {
-- ["streamer.stream[2].Rtsp_server.session"] = "mjpeg01",
-- ["streamer.stream[2].Rtsp_server.multicast_ttl"] = 127,
-- ["streamer.rtsp_auth"] = 0,
-- ["streamer.stream[2].Rtsp_server.port"] = 554,
-- ["streamer.stream[2].Rtsp_server.multicast_addr"] = "224.0.0.1",
-- ["streamer.stream[2].Rtsp_server.httpTunnelPort"] = 0,
-- ["streamer.stream[2].Rtsp_server.multicast_video_port"] = 3200,
-- ["streamer.stream[2].Rtsp_server.ssm"] = 0,
-- ["streamer.stream[2].Rtsp_server.no_adts"] = 0,
-- ["streamer.stream[2].Rtsp_server.multicast_audio_port"] = 3202
-- }

streamer_DefaultApp_RTSP =
streamer:CreateProxy('/DefaultApp/0'..stream_id..'/Rtsp_server')
r,count,table = streamer_DefaultApp_RTSP:GetStatus()

-- 结果
-- true    1       table: {
-- ["stream.status"] = "online",
-- }

enabled,service_uri,service_type =
streamer_DefaultApp_RTSP._P("Enabled","ServiceUri", "ServiceType")

-- print(enabled,service_uri,service_type)

-- 结果
-- true    rtsp://%HOST%:554/ch01  RTSP/SERVER
```

RTSP-设置

```
HTTP GET /stream-media/rtsp-set
#输入
stream_id: 2    #流编号，0~2，0：主码流，1：字码流，2：图片流
rtsp_server_port: 554           #服务端口，默认554
rtsp_server_http_tunnel_port: 0 #HTTP Tunnel端口，Tunnel端口保持为0，表示不开启HTTP
Tunnel功能。
                                # 注意HTTP Tunnel端口不要与当前Web服务端口及Onvif服务
端口相同，建议值为1080。
                                # 修改HTTP Tunnel端口，仅当复位或重启后才能生效
```

```
rtsp_server_session: 'mjpeg01'      #会话编号
rtsp_server_no_adts: 0       #对AAC音频策略，0：保持默认格式，1：去除ADTS头
rtsp_auth: 0                 #RTSP身份认证，0：关闭，1：开启，由接口#RTSP-用户-*进行身份管理
rtsp_server_ssm: 0  #组播，0：关闭，1：开启，组播会话将固定为'<会话(Session) ID>/ssm'，即:mjpeg01/ssm
rtsp_server_multicast_addr: "224.0.0.1" #若rtsp_server_ssm为1，则生效
                                        # 组播地址，224.x.x.x～239.x.x.x
rtsp_server_multicast_ttl: 127  #若rtsp_server_ssm为1，则生效
                                # Time to Live 生存时间1～255，决定允许组播数据通过路由节点的个数。
                                # TTL值越大表示允许通过路由节点的个数越多，根据实际网络环境更改值，默认值127
rtsp_server_multicast_video_port: 3000 #若rtsp_server_ssm为1，则生效
                                       # 组播视频端口
rtsp_server_multicast_audio_port: 3002 #若rtsp_server_ssm为1，则生效
                                       # 组播音频端口
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
stream_id = 2 -- 流编号，0~2，0：主码流，1：字码流，2：图片流？

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')
r,count = streamer_DefaultApp:Set({
    ["streamer.stream[".stream_id."].Rtsp_server.port"] = 554,
    ["streamer.stream[".stream_id."].Rtsp_server.httpTunnelPort"] = 1080,
    ["streamer.stream[".stream_id."].Rtsp_server.session"] = "mjpeg1",
    ["streamer.stream[".stream_id."].Rtsp_server.no_adts"] = 0,
    ["streamer.rtsp_auth"] = 0,
    ["streamer.stream[".stream_id."].Rtsp_server.ssm"] = 1,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_addr"] = "224.0.0.1",
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_ttl"] = 127,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_video_port"] = 3000,
    ["streamer.stream[".stream_id."].Rtsp_server.multicast_audio_port"] = 3002
})

--print(r,count)
-- 结果
-- true    10
```

RTSP-用户-列表

```
HTTP GET /stream-media/rtsp-user-list
#输入
#输出
{
    err: null,
    val: ['张三','李四','王五']   #用户名列表
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_MediaUsers = streamer:CreateProxy('/MediaUsers')

item = 0 -- 用户项编号，0~15，最多16个用户

r,count,table = streamer_MediaUsers:Get({
    [ "media_users.users[" .. item .. "].username" ] = ""
})

-- print(r,count,table)
-- print_table(table)

-- true    1       table: {
-- ["media_users.users[0].username"] = "aaaa"
-- }
```

RTSP-用户-添加

```
HTTP PUT /stream-media/rtsp-user-add
#输入
username: '张三'  #用户名
password: 'zszs' #密码
#输出
{
    err: null,
    val: ['张三','李四','王五']   #用户名列表
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_MediaUsers = streamer:CreateProxy('/MediaUsers')

r,count = streamer_MediaUsers:Set({
    ["media_users.add.username"] = '张三',
    ["media_users.add.password"] = 'zszszs'
})

print(r,count)

-- true    2
```

RTSP-用户-删除

```
HTTP DELETE /stream-media/rtsp-user-erase
#输入
username: '张三' #被删除的用户名
#输出
{
    err: null,
    val: true
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_MediaUsers = streamer:CreateProxy('/MediaUsers')

r,count = streamer_MediaUsers:Set({
    ["media_users.delete.username"] = '张三'
})

print(r,count)

-- true    1
```

MJPEG流-状态

```
HTTP GET /stream-media/mjpeg-status
#输入
stream_id: 2    #流编号，0~2，0：主码流，1：字码流，2：图片流
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        audio_channel: 1,   #音频编码通道，可以通过#编码引擎-状态 获取音频信息
        video_codec: 'MJPEG',   #编码，MJPEG，H264
        video_scale: 'default', #缩放尺寸
        video_profile: 'default',   #编码体系，default，base，main，extend，high
                                    # MJPEG编码固定为default
        video_frame_rate: 30.0, #实时帧率
        video_bitrate: 2130,    #实时码率
    }
}
```

关联接口
编码引擎-状态

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

stream_id = 2 -- 流编号，0~2，0：主码流，1：字码流，2：图片流

streamer = client.New("streamer", "local", "192.168.2.174:14000")
```

```
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')

r, count, table = streamer_DefaultApp:Get({
    ["streamer.stream[" .. stream_id .. "].audioChannel"] = 0
})

-- 结果
-- true 1   table: {
--  ["streamer.stream[2].audioChannel"] = 1
-- }

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp_Source =
streamer:CreateProxy('/DefaultApp/0'..stream_id..'/Source')
r,count,table = streamer_DefaultApp_Source:GetStatus()

-- 结果
-- true    10      table: {
-- ["video.startPersis"] = 166581,
-- ["video.frameRate"] = 5.0,
-- ["video.lastResetPersis"] = 166581,
-- ["channel"] = 2,
-- ["video.bitrate"] = 958,
-- ["video.lastResetTime"] = "2020-11-09 16:19:57",
-- ["dataRateMeasure"] = 1000,
-- ["video.encodedFrames"] = 832943.0,
-- ["video.encodedBytes"] = 24240060000.0,
-- ["video.resets"] = 0,
-- ["video.startTime"] = "2020-11-09 16:19:57",
-- }

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

r, count, table = codec_Codec_Encoders:Get({
    ["encoder.stream[" .. stream_id .. "].enabled"] = 0,
    ["encoder.stream[" .. stream_id .. "].video.profile"] = "",
    ["encoder.stream[" .. stream_id .. "].video.scale"] = "",
    ["encoder.stream[" .. stream_id .. "].video.codec"] = ""
})

--结果
-- true    4       table: {
--  ["encoder.stream[2].video.profile"] = "high",
--  ["encoder.stream[2].enabled"] = 1,
--  ["encoder.stream[2].video.codec"] = "H264",
--  ["encoder.stream[2].video.scale"] = "default"
-- }

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders/0'..stream_id..')
r,count,table = codec_Codec_Encoders:GetVideoProperties()

-- 结果
-- true    6       table: {
--  ["video.progressive"] = true,
--  ["video.codec"] = "H264",
--  ["video.width"] = 1920,
```

```
--  ["video.height"] = 1080,
--  ["video.bitrate"] = 2000000,
--  ["video.framerate"] = 30.0
-- }
```

MJPEG流-编码器-状态

```
HTTP GET /stream-media/mjpeg-encoder-status
#输入
stream_id: 2    #流编号，0~2，0：主码流，1：字码流，2：图片流
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        snap_type: "preview",    #MJPEG画面设定
                                 # preview：仅用于预览的小画面 (且只有5fps)，
                                 # main：和H.264主码流画面相同，
                                 # sub：和H.264子码流画面相同
        kmp_ordering: "default", #KMP服务中占用第1个通道，default：否，snap_first：是
                                    # 考虑与老产品的兼容性原因，选择“是”表示强制让JPEG码流在KMP服务中，占用第1个通道
        video_quality: 80,  #画面质量，1-99，值越大，编码画面质量更好
        video_chrome: 0,         #色彩，0：彩色，1：黑白
        video_framerate_mode: "default",     #帧率规则
                                             # default：全帧率(与原始视频一致)
                                             # half：当原始视频帧率>=50时自动减半
                                             # custom：自定义帧率
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

stream_id = 2    --流编号，0~2，0：主码流，1：字码流，2：图片流？

r, count, table = codec_Codec_Encoders:Get({
    ["encoder.snap_type"] = "",
    ["encoder.stream["..stream_id.."].video.quality"] = 0,
    ["encoder.stream["..stream_id.."].video.chrome"] = 0,
    ["encoder.stream["..stream_id.."].video.framerate_mode"] = "",
})

--结果
-- true    4       table: {
-- ["encoder.snap_type"] = "preview",
-- ["encoder.stream[2].video.quality"] = 80,
-- ["encoder.stream[2].video.chrome"] = 1,
-- ["encoder.stream[2].video.framerate_mode"] = "default",
-- }

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')
```

```
r, count, table = streamer_DefaultApp:Get({
    ["streamer.kmp_ordering"] = ""
})

--结果
-- true    4       table: {
--  ["streamer.kmp_ordering"] = "default"
-- }
```

MJPEG流-编码器-设置

```
HTTP PUT /stream-media/mjpeg-encoder-set
#输入
stream_id: 2    #流编号，0~2，0：主码流，1：字码流，2：图片流
snap_type: "preview"    #MJPEG画面设定
                        # preview：仅用于预览的小画面 (且只有5fps)，
                        # main：和H.264主码流画面相同，
                        # sub：和H.264子码流画面相同
kmp_ordering: "default" #KMP服务中占用第1个通道，default：否，snap_first：是
                        # 考虑与老产品的兼容性原因，选择“是”表示强制让JPEG码流在KMP服务
中，占用第1个通道
video_quality: 80   #画面质量，1-99，值越大，编码画面质量更好
video_chrome: 0     #色彩，0：彩色，1：黑白
video_framerate_mode: "default"     #帧率规则
                                    # default：全帧率(与原始视频一致)
                                    # half：当原始视频帧率>=50时自动减半
                                    # custom：自定义帧率
video_framerate: 5,     #自定义帧率，帧率规则为custom生效
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

stream_id = 2   --流编号，0~2，0：主码流，1：字码流，2：图片流？

r, count = codec_Codec_Encoders:Set({
    ["encoder.snap_type"] = "preview",
    ["encoder.stream["..stream_id.."].video.quality"] = 80,
    ["encoder.stream["..stream_id.."].video.chrome"] = 1,
    ["encoder.stream["..stream_id.."].video.framerate_mode"] = "default",
})

--结果
-- true    4

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')
```

```
r, count = streamer_DefaultApp:Set({
    ["streamer.kmp_ordering"] = "default"
})

--结果
-- true    1
```

主/子码流-状态

```
HTTP GET /stream-media/status
#输入
stream_id: 0    #流编号，0~2，0：主码流，1：字码流，2：图片流
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        audio_channel: 1,    #音频编码通道，可以通过#编码引擎-状态 获取音频信息
        video_codec: 'H264',    #编码，MJPEG，H264
        video_scale: 'default', #缩放尺寸
                                # default，720x480，720x576，960x540
                                # 800x600，854x480，1024x576，1280x720，1920x1080
        video_profile: 'high',  #编码体系，default，base，main，extend，high
                                # MJPEG编码固定为default
        video_frame_rate: 30.0, #实时帧率
        video_bitrate: 2130,    #实时码率
    }
}
```

关联接口
编码引擎-状态

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
stream_id = 0 -- 流编号，0~2，0：主码流，1：字码流，2：图片流？

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp = streamer:CreateProxy('/DefaultApp')

r, count, table = streamer_DefaultApp:Get({
    ["streamer.stream[" .. stream_id .. "].audioChannel"] = 0
})

-- 结果
-- true 1   table: {
--  ["streamer.stream[0].audioChannel"] = 1
-- }

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_DefaultApp_Source =
streamer:CreateProxy('/DefaultApp/0'..stream_id..'/Source')
r,count,table = streamer_DefaultApp_Source:GetStatus()

-- 结果
-- true    10      table: {
```

```
-- ["video.startPersis"] = 166581,
-- ["video.frameRate"] = 5.0,
-- ["video.lastResetPersis"] = 166581,
-- ["channel"] = 2,
-- ["video.bitrate"] = 958,
-- ["video.lastResetTime"] = "2020-11-09 16:19:57",
-- ["dataRateMeasure"] = 1000,
-- ["video.encodedFrames"] = 832943.0,
-- ["video.encodedBytes"] = 24240060000.0,
-- ["video.resets"] = 0,
-- ["video.startTime"] = "2020-11-09 16:19:57",
-- }

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

r, count, table = codec_Codec_Encoders:Get({
    ["encoder.stream[" .. stream_id .. "].enabled"] = 0,
    ["encoder.stream[" .. stream_id .. "].video.profile"] = "",
    ["encoder.stream[" .. stream_id .. "].video.scale"] = "",
    ["encoder.stream[" .. stream_id .. "].video.codec"] = ""
})

--结果
-- true    4       table: {
--  ["encoder.stream[0].video.profile"] = "high",
--  ["encoder.stream[0].enabled"] = 1,
--  ["encoder.stream[0].video.codec"] = "H264",
--  ["encoder.stream[0].video.scale"] = "default"
-- }

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders/0'..stream_id)
r,count,table = codec_Codec_Encoders:GetVideoProperties()

-- 结果
-- true    6       table: {
--  ["video.progressive"] = true,
--  ["video.codec"] = "H264",
--  ["video.width"] = 1920,
--  ["video.height"] = 1080,
--  ["video.bitrate"] = 2000000,
--  ["video.framerate"] = 30.0
-- }
```

主/子码流-编码-状态

```
HTTP GET /stream-media/encoder-status
#输入
stream_id: 0    #流编号，0~2，0：主码流，1：字码流，2：图片流
#输出
{
    err: null, #成功null，否则错误信息
    val: {
        video.scale: "default", #缩放尺寸
                                # default，720x480，720x576，960x540
```

```
                                # 800x600，854x480，1024x576，1280x720，1920x1080
        video_chrome: 0,        #色彩，0：彩色，1：黑白
        video_profile: 'high',  #编码体系，default，base，main，extend，high
        video_br_ctrl: 'cbr',   #码率控制方式，cbr：CBR-恒定码率模式，vbr：VBR-动态码率
模式
        video_qp_min: 18,       #QP最小值，0～51
        video_qp_max: 51,       #QP最大值，0～51
        video_bitrate: 10000000,#编码码率
                                # 0，64000，128000，256000，512000，768000，
1000000，
                                # 1200000，1500000，1800000，2000000，2500000，
3000000，
                                # 4000000，6000000，8000000，10000000，12000000，
15000000，
                                # 20000000，25000000，30000000，35000000，40000000
        video_dynamic_bitrate: 0, #动态码率，0：关闭，1：根据网络带宽自动调节码率
        video_framerate_mode: "default",     #帧率规则
                                             # default：全帧率(与原始视频一致)
                                             # half：当原始视频帧率>=50时自动减半
                                             # custom：自定义帧率
        video_framerate: 5,     #自定义帧率，帧率规则为custom生效
        video_reduce_framerate: 0,  #降低帧率，0：关闭，1：当码率偏低时，自动降低帧率
        video_gop_size: 30,         #GOP大小(I帧间隔)，1～600
        video_ref_frames: 1, #编码参考，1：单一参考帧，2：多参考帧
                             # 使用多参考帧可以提高编码质量，但某些解码器可能不支持
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

stream_id = 0    --流编号，0~2，0：主码流，1：字码流，2：图片流？

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

r, count, table = codec_Codec_Encoders:Get({
    ["encoder.stream[".stream_id."].video.scale"] = "",
    ["encoder.stream[".stream_id."].video.chrome"] = 0,
    ["encoder.stream[".stream_id."].video.profile"] = "",
    ["encoder.stream[".stream_id."].video.br_ctrl"] = "",
    ["encoder.stream[".stream_id."].video.qp_min"] = 0,
    ["encoder.stream[".stream_id."].video.qp_max"] = 0,
    ["encoder.stream[".stream_id."].video.bitrate"] = 0,
    ["encoder.stream[".stream_id."].video.dynamic_bitrate"] = 0,
    ["encoder.stream[".stream_id."].video.framerate_mode"] = "",
    ["encoder.stream[".stream_id."].video.framerate"] = 0,
    ["encoder.stream[".stream_id."].video.reduce_framerate"] = 0,
    ["encoder.stream[".stream_id."].video.gop_size"] = 0,
    ["encoder.stream[".stream_id."].video.ref_frames"] = 0
})

--结果
-- true    13      table: {
--  ["encoder.stream[0].video.scale"] = "default",
```

```
--  ["encoder.stream[0].video.framerate_mode"] = "custom",
--  ["encoder.stream[0].video.dynamic_bitrate"] = 0,
--  ["encoder.stream[0].video.ref_frames"] = 1,
--  ["encoder.stream[0].video.qp_max"] = 51,
--  ["encoder.stream[0].video.br_ctrl"] = "cbr",
--  ["encoder.stream[0].video.bitrate"] = 10000000,
--  ["encoder.stream[0].video.reduce_framerate"] = 0,
--  ["encoder.stream[0].video.framerate"] = 50.0,
--  ["encoder.stream[0].video.gop_size"] = 50,
--  ["encoder.stream[0].video.profile"] = "high",
--  ["encoder.stream[0].video.chrome"] = 0,
--  ["encoder.stream[0].video.qp_min"] = 18
-- }
```

主/子码流-编码-设置

```
HTTP PUT /stream-media/encoder-set
#输入
stream_id: 0     #流编号，0~2，0：主码流，1：字码流，2：图片流
video.scale: "default" #缩放尺寸
                       # default，720x480，720x576，960x540
                       # 800x600，854x480，1024x576，1280x720，1920x1080
video_chrome: 0          #色彩，0：彩色，1：黑白
video_profile: 'high'    #编码体系，default，base，main，extend，high
video_br_ctrl: 'cbr'     #码率控制方式，cbr：CBR-恒定码率模式，vbr：VBR-动态码率模式
video_qp_min: 18         #QP最小值，0～51
video_qp_max: 51         #QP最大值，0～51
video_bitrate: 10000000 #编码码率
                        # 0，64000，128000，256000，512000，768000，1000000，
                        # 1200000，1500000，1800000，2000000，2500000，3000000，
                        # 4000000，6000000，8000000，10000000，12000000，15000000，
                        # 20000000，25000000，30000000，35000000，40000000
video_dynamic_bitrate: 0 #动态码率，0：关闭，1：根据网络带宽自动调节码率
video_framerate_mode: "default" #帧率规则
                                # default：全帧率(与原始视频一致)
                                # half：当原始视频帧率>=50时自动减半
                                # custom：自定义帧率
video_framerate: 30          #自定义帧率，帧率规则为custom生效
video_reduce_framerate: 0    #降低帧率，0：关闭，1：当码率偏低时，自动降低帧率
video_gop_size: 30           #GOP大小(I帧间隔)，1～600，默认为60
video_ref_frames: 1          #编码参考，1：单一参考帧，2：多参考帧，默认为1
                             # 使用多参考帧可以提高编码质量，但某些解码器可能不支持
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

stream_id = 0    --流编号，0~2，0：主码流，1：字码流，2：图片流？

codec = client.New("codec", "local", "192.168.2.174:14000")
codec_Codec_Encoders = codec:CreateProxy('/Codec/Encoders')

r, count, table = codec_Codec_Encoders:Set({
    ["encoder.stream["..stream_id.."].video.scale"] = "default",
    ["encoder.stream["..stream_id.."].video.chrome"] = 0,
    ["encoder.stream["..stream_id.."].video.profile"] = "high",
```

```
    ["encoder.stream[..stream_id..].video.br_ctrl"] = "cbr",
    ["encoder.stream[..stream_id..].video.qp_min"] = 18,
    ["encoder.stream[..stream_id..].video.qp_max"] = 51,
    ["encoder.stream[..stream_id..].video.bitrate"] = 10000000,
    ["encoder.stream[..stream_id..].video.dynamic_bitrate"] = 0,
    ["encoder.stream[..stream_id..].video.framerate_mode"] = "default",
    ["encoder.stream[..stream_id..].video.framerate"] = 30,
    ["encoder.stream[..stream_id..].video.reduce_framerate"] = 0,
    ["encoder.stream[..stream_id..].video.gop_size"] = 60,
    ["encoder.stream[..stream_id..].video.ref_frames"] = 1
})

--结果
-- true    13
```

主/子码流-录像-策略-状态

```
HTTP GET /stream-media/record-policy-status
#输入
stream_id: 0 --流编号，0~1，0：主码流，1：子码流
#输出
{
    err: null,
    val: {
        auto_record: 0, #(插入存储器时)自动录像，0：关闭，1：开启
        disk_type: "usb-first", #选择存储器，不管可用空间大小，录像到第一个插入的USB存储器分区上
                                # first：第一个插入的存储器，largest：可用空间最大的存储器，
                                # mark：特别标记的存储器，nas：可用空间最大的NAS存储器，
                                # nas-first：第一个NAS存储器，nas-mark：特别标记的NAS存储器
                                # usb：可用空间最大的USB存储器，usb-first：第一个插入的USB存储器，
                                # usb-mark：特别标记的USB存储器，sdmmc：可用空间最大的SD/MMC存储器，
                                # sdmmc-first：第一个插入的SD/MMC存储器，sdmmc-mark：特别标记的SD/MMC存储器
        format: "ts",           #录像文件格式，ts，mp4，mov，mkv，avi
        file_prefix: "REC",     #文件名前缀
        limit_type: "time-loop", #文件限制
                                 # size-loop：限制大小，自动切割多个文件，
                                 # time-loop：限制时长，自动切割多个文件，
                                 # size：单个文件限制大小，time：单个文件限制时长，none：不限大小和时长
        limit_size: 10000,       #若limit_type为size*，则生效，限制大小
        limit_time: 600,         #若limit_type为time*，则生效，限制时间
        disk_policy: "overwrite" #存储空间不够时的策略，overwrite：覆盖老的录像文件，full-stop：空间不够时停止录像
    }
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
```

```
streamer = client.New("record", "local", "192.168.2.174:14000")
record = streamer:CreateProxy('/')

stream_id = 0    --流编号，0~1，0：主码流，1：子码流
r, count, table = record:Get({
    ["record_m.stream[".stream_id."].auto_record"] = 0,
    ["record_m.stream[".stream_id."].disk_type"] = "",
    ["record_m.stream[".stream_id."].format"] = "",
    ["record_m.stream[".stream_id."].file_prefix"] = "",
    ["record_m.stream[".stream_id."].limit_type"] = "", -- 'time-loop' or
'size-loop' or 'time' or 'size'
    ["record_m.stream[".stream_id."].limit_size"] = 0,
    ["record_m.stream[".stream_id."].limit_time"] = 0,
    ["record_m.stream[".stream_id."].disk_policy"] = "" -- 'overwrite' or
'full-stop'
})

-- print(r,count,table)
-- print_table(table)

-- 结果
-- true    8    table: {
-- ["record_m.stream[0].disk_type"] = "usb-first",
-- ["record_m.stream[0].limit_type"] = "time-loop",
-- ["record_m.stream[0].disk_policy"] = "overwrite",
-- ["record_m.stream[0].limit_size"] = 10000,
-- ["record_m.stream[0].auto_record"] = 0,
-- ["record_m.stream[0].format"] = "ts",
-- ["record_m.stream[0].limit_time"] = 600,
-- ["record_m.stream[0].file_prefix"] = "REC"
-- }
```

主/子码流-录像-策略-设置

```
HTTP GET /stream-media/record-policy-set
#输入
stream_id: 0 #流编号，0~1，0：主码流，1：子码流
auto_record: 0  #(插入存储器时)自动录像，0：关闭，1：开启
disk_type: "usb-first"  #选择存储器，不管可用空间大小，录像到第一个插入的USB存储器分区上
                        # first：第一个插入的存储器，largest：可用空间最大的存储器，
                        # mark：特别标记的存储器，nas：可用空间最大的NAS存储器，
                        # nas-first：第一个NAS存储器，nas-mark：特别标记的NAS存储器
                        # usb：可用空间最大的USB存储器，usb-first：第一个插入的USB存储
器，
                        # usb-mark：特别标记的USB存储器，sdmmc：可用空间最大的SD/MMC存
储器，
                        # sdmmc-first：第一个插入的SD/MMC存储器，sdmmc-mark：特别标记
的SD/MMC存储器
format: "ts"            #录像文件格式，ts，mp4，mov，mkv，avi
file_prefix: "REC"      #文件名前缀
limit_type: "time-loop"     #文件限制
                            # size-loop：限制大小，自动切割多个文件，
                            # time-loop：限制时长，自动切割多个文件，
                            # size：单个文件限制大小，time：单个文件限制时长，none：不限
大小和时长
limit_size: 10000         #若limit_type为size*，则生效，限制大小
limit_time: 600           #若limit_type为time*，则生效，限制时间
```

```
disk_policy: "overwrite" #存储空间不够时的策略，overwrite：覆盖老的录像文件，full-stop：空间不够时停止录像
#输出
{
    err: null,
    val: true
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("streamer", "local", "192.168.2.174:14000")
streamer_MediaUsers = streamer:CreateProxy('/MediaUsers')

stream_id = 0    --流编号，0~1，0：主码流，1：字码流
r, count, table = servers["record/"]:Set({
    ["record_m.stream["..stream_id.."].auto_record"] = 1,
    ["record_m.stream["..stream_id.."].disk_type"] = "any",
    ["record_m.stream["..stream_id.."].format"] = "ts",
    ["record_m.stream["..stream_id.."].file_prefix"] = "REC",
    ["record_m.stream["..stream_id.."].limit_type"] = "none",
    ["record_m.stream["..stream_id.."].limit_size"] = 0,
    ["record_m.stream["..stream_id.."].limit_time"] = 0,
    ["record_m.stream["..stream_id.."].disk_policy"] = "overwrite"
})

-- print(r,count)
-- true    8
```

主/子码流-录像-状态

```
HTTP GET /stream-media/record-status
#输入
stream_id: 0 #流编号，0~1，0：主码流，1：子码流
#输出
{
    err: null,
    val: 0      #录像状态，0：没录像，1：录像中
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("record", "local", "192.168.2.174:14000")
record = streamer:CreateProxy('/')

stream_id = 0    --流编号，0~1，0：主码流，1：子码流
r, count, table = record:Get({
    ["record_m.stream["..stream_id.."].start"] = 0,
})

-- print(r,count,table)
-- print_table(table)
```

```
-- 结果
-- true    1    table: {
-- ["record_m.stream[0].start"] = 0
-- }
```

主/子码流-录像-打开

```
HTTP GET /stream-media/record-open
#输入
stream_id: 0 #流编号，0~1，0：主码流，1：子码流
#输出
{
    err: null,
    val: true
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("record", "local", "192.168.2.174:14000")
record = streamer:CreateProxy('/')

stream_id = 0    --流编号，0~1，0：主码流，1：子码流
r, count, table = record:Set({
    ["record_m.stream["..stream_id.."].start"] = 1,
})

print(r,count)

-- 结果
-- true    1
```

主/子码流-录像-关闭

```
HTTP GET /stream-media/record-close
#输入
stream_id: 0 #流编号，0~1，0：主码流，1：子码流
#输出
{
    err: null,
    val: true
}
```

```
-- 实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

streamer = client.New("record", "local", "192.168.2.174:14000")
record = streamer:CreateProxy('/')

stream_id = 0    --流编号，0~1，0：主码流，1：子码流
```

```
r, count, table = record:Set({
    ["record_m.stream["..stream_id.."].start"] = 0,
})

print(r,count)

-- 结果
-- true    1
```

主/子码流-录像-列表

```
HTTP GET /stream-media/record-list
#输入
#输出
{
    err: null,
    val: [
        {
            name: 'M-20201022_151257_0001',     #录像序列
            time_begun:'2020-10-22 15:12:59',   #开始时间
            time_ended:'2020-10-22 15:29:15',   #结束时间
            files: [
                {       #第一个文件
                    name: 'REC-20201022151257.ts',
                    size: '18.8 MB'
                }, {    #第二个文件
                    name: 'REC-20201022151400.ts',
                    size: '18.8 MB'
                }
            ],
            size: '37.6 MB' #总大小
        }, {
            name: 'M-20201022_151257_0002',
            time_begun:'2020-10-23 15:12:59',
            time_ended:'2020-10-23 15:29:15',
            files: [
                {
                    name: 'REC-20201023151257.ts',
                    size: '18.8 MB'
                }
            ],
            size: '18.8 MB'
        }
    ]
}
```

```
#命令
ls -l | egrep '\.ts|mov|mp4$'
```

叠加层-状态

```
HTTP GET /stream-media/overlay-status
#输入
stream_id: 0    #流编号，0~2，0：主码流，1：子码流，2：图片流？
#输出
{
    err: null,  #成功null，否则错误信息
    val: {
        enable: 0,  #是否启用，0：关闭，1：打开
        norotate: 1 #叠加层不随画面旋转，0：旋转，1：不旋转
    }
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0   --流编号，0~2，0：主码流，1：字码流，2：图片流？

params = {
    [ "osd.stream["..stream_id.."].enabled" ] = 0,
    [ "osd.stream["..stream_id.."].norotate" ] = 0
}

r, count, table = servers["codec/OSD"]:Get(params)
--结果
r: true    count: 2       table: {
    ["osd.stream[0].norotate"] = 1,
    ["osd.stream[0].enabled"] = 1
}
```

叠加层-打开

```
HTTP PUT /stream-media/overlay-open
#输入
stream_id: 0   #流编号，0~2，0：主码流，1：字码流，2：图片流？
norotate: 1    #叠加层不随画面旋转，0：旋转，1：不旋转
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0   --流编号，0~2，0：主码流，1：字码流，2：图片流？
```

```
params = {
    [ "osd.stream["..stream_id.."].enabled" ] = 1,  --启用
    [ "osd.stream["..stream_id.."].norotate" ] = 0  --不旋转
}

r, count = servers["codec/OSD"]:Set(params)

--结果
r: true    count: 2
```

叠加层-关闭

```
HTTP PUT /stream-media/overlay-close
#输入
stream_id: 0    #流编号，0~2，0：主码流，1：字码流，2：图片流？
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")
codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0   --流编号，0~2，0：主码流，1：字码流，2：图片流？

params = {
    [ "osd.stream["..stream_id.."].enabled" ] = 0,  --启用
    [ "osd.stream["..stream_id.."].norotate" ] = 0  --不旋转
}

r, count = servers["codec/OSD"]:Set(params)

--结果
r: true    count: 2
```

叠加层-项-列表

```
HTTP GET /stream-media/overlay-tiems-list
#输入
stream_id: 0    #流编号，0~2，0：主码流，1：字码流，2：图片流？
#输出
{
    err: null,  #成功null，否则错误信息
    val: [{
        overlay_type: 'text', #叠加类型，text：文字，time：系统时间，image：图片，
coordinates：GPS:经/纬度
                              # altitude：GPS:高度，speed：GPS:速度，bearing：GPS:方
位
        show_when: 'always',  #显示条件，always：总是显示，signal：当有视频输入时，
nosignal：当没有视频输入时
```

```
        position: "left-top", #叠加位置，left-top：左上角，top-center：上边居中，
right-top：右上角
                              # left-bottom：左下角，bottom-center：下边居中，right-
bottom：右下角
                              # center：正中心，special：自定义
        x: 0,   #如果叠加位置为special，则生效，叠加位置x
        y: 0,   #如果叠加位置为special，则生效，叠加位置y
        font: 'default', #字体,default：默认 (英文)，vera：Vera (英文)，lucida：
Lucida (英文)，song：宋体 (中文)
        font_color: 1,   #字体颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄色，
6：紫色，7：青色
        size: 1,         #字体大小，0：自动 - 较小，1：自动 - 中等，2：自动 - 大字体
                         # 9，12，16，20，24，28，32，36，40，48，56，64，72，80，88，
96：各种px的大小
        bold: 0,         #字体样式，0：正常，1：加粗
        outline: 2,      #字体轮廓厚度，0～10
        line_space: 'x0.5', #行间距，0，x0.1，x0.2，x0.3，x0.4，x0.5，x0.6，x0.7，
x0.8，x0.9
                            # x1，x1.2，x1.5，x1.8，x2，x2.5，x3
        outline_color: 0,   #边框颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄
色，6：紫色，7：青色
        text: "here is overlay of text",    #如果叠加类型为text，则生效，文字
        format: "", #如果叠加类型为time，则生效，时间格式，不填为YYYY-MM-DD hh:mm:ss
                    # %m/%d/%Y %H:%M:%S，%d/%m/%Y %H:%M:%S，%Y-%m-%d，%m/%d/%Y，
%d/%m/%Y，%H:%M:%S，%H:%M
        image_id: '' #如果叠加类型为image，则生效，图片编号，由“叠加层-图片-添加”接口上传
    },{
        overlay_type: 'time',   #叠加类型，time：系统时间
        show_when: 'always',
        position: "left-top",
        x: 0,
        y: 0,
        font: 'default',
        font_color: 1,
        size: 1,
        bold: 0,
        outline: 2,
        line_space: 'x0.5',
        outline_color: 0,
        text: "",
        format: "",
        image_id: ''
    },{
        overlay_type: 'image',  #叠加类型，image：图片
        show_when: 'always',
        position: "left-top",
        x: 0,
        y: 0,
        font: 'default',
        font_color: 1,
        size: 1,
        bold: 0,
        outline: 2,
        line_space: 'x0.5',
        outline_color: 0,
        text: "",
        format: "",
        image_id: 'overlay-of-image'
```

```
    }]
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0    --流编号，0~2，0：主码流，1：字码流，2：图片流？
item_id = 0      --叠加项编号，0~15

params = {
    --叠加类型，text：文字，time：系统时间，image：图片，coordinates：GPS:经/纬度，
altitude：GPS:高度，speed：GPS:速度，bearing：GPS:方位
    [ "osd.stream["..stream_id.."].items["..item_id.."].osd_type" ] = "",
    --显示条件，always：总是显示，signal：当有视频输入时，nosignal：当没有视频输入时
    [ "osd.stream["..stream_id.."].items["..item_id.."].show_when" ] = "always",
    --叠加位置，left-top：左上角，top-center：上边居中，right-top：右上角，left-bottom：
左下角，bottom-center：下边居中，right-bottom：右下角，center：正中心，special：自定义
    [ "osd.stream["..stream_id.."].items["..item_id.."].position" ] = "",
    --如果叠加位置为special，则生效，叠加位置x
    [ "osd.stream["..stream_id.."].items["..item_id.."].x" ] = 0,
    --如果叠加位置为special，则生效，叠加位置y
    [ "osd.stream["..stream_id.."].items["..item_id.."].y" ] = 0,
    --字体,default：默认 (英文)，vera：Vera (英文)，lucida：Lucida (英文)，song：宋体
(中文)
    [ "osd.stream["..stream_id.."].items["..item_id.."].font" ] = "",
    --字体颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄色，6：紫色，7：青色
    [ "osd.stream["..stream_id.."].items["..item_id.."].font_color" ] = 0,
    --字体大小，0：自动 - 较小，1：自动 - 中等，2：自动 - 大字体，9，12，16，20，24，28，
32，36，40，48，56，64，72，80，88，96：各种px的大小
    [ "osd.stream["..stream_id.."].items["..item_id.."].size" ] = 0,
    --字体样式，0：正常，1：加粗
    [ "osd.stream["..stream_id.."].items["..item_id.."].bold" ] = 0,
    --字体轮廓厚度，0～10
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline" ] = 2,
    --行间距，0，x0.1，x0.2，x0.3，x0.4，x0.5，x0.6，x0.7，x0.8，x0.9，x1，x1.2，
x1.5，x1.8，x2，x2.5，x3
    [ "osd.stream["..stream_id.."].items["..item_id.."].line_space" ] = "",
    --边框颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄色，6：紫色，7：青色
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline_color" ] = 0,
    --如果叠加类型为text，则生效，文字
    [ "osd.stream["..stream_id.."].items["..item_id.."].text" ] = "",
    --如果叠加类型为time，则生效，时间格式，YYYY-MM-DD hh:mm:ss，%m/%d/%Y %H:%M:%S，
%d/%m/%Y %H:%M:%S，%Y-%m-%d，%m/%d/%Y，%d/%m/%Y，%H:%M:%S，%H:%M
    [ "osd.stream["..stream_id.."].items["..item_id.."].format" ] = "",
    --如果叠加类型为image，则生效，图片编号，由“叠加层-图片-添加”接口上传
    [ "osd.stream["..stream_id.."].items["..item_id.."].image_id" ] = ""
}

r, count, table = servers["codec/OSD"]:Get(params)
--结果
r: true    count: 15      table: {
    ["osd.stream[0].items[0].text"] = "121212",
    ["osd.stream[0].items[0].bold"] = 0,
```

```
    ["osd.stream[0].items[0].format"] = "",
    ["osd.stream[0].items[0].size"] = 1,
    ["osd.stream[0].items[0].position"] = "left-top",
    ["osd.stream[0].items[0].font_color"] = 1,
    ["osd.stream[0].items[0].x"] = 0,
    ["osd.stream[0].items[0].font"] = "default",
    ["osd.stream[0].items[0].outline_color"] = 0,
    ["osd.stream[0].items[0].image_id"] = "",
    ["osd.stream[0].items[0].osd_type"] = "text",
    ["osd.stream[0].items[0].line_space"] = "x0.5",
    ["osd.stream[0].items[0].outline"] = 2,
    ["osd.stream[0].items[0].show_when"] = "always",
    ["osd.stream[0].items[0].y"] = 0
}
```

叠加层-项-添加

```
HTTP PUT /stream-media/overlay-items-set
#输入
stream_id: 0          #流编号，0~2，0：主码流，1：字码流，2：图片流？
item_id: 4      #叠加项编号，0~15
overlay_type: 'text' #叠加类型，text：文字，time：系统时间，image：图片，coordinates：GPS:经/纬度
                     # altitude：GPS:高度，speed：GPS:速度，bearing：GPS:方位
show_when: 'always'  #显示条件，always：总是显示，signal：当有视频输入时，nosignal：当没有视频输入时
position: "left-top" #叠加位置，left-top：左上角，top-center：上边居中，right-top：右上角
                     # left-bottom：左下角，bottom-center：下边居中，right-bottom：右下角
                     # center：正中心，special：自定义
x: 0    #如果叠加位置为special，则生效，叠加位置x
y: 0    #如果叠加位置为special，则生效，叠加位置y
font: 'default' #字体,default：默认（英文），vera：Vera（英文），lucida：Lucida（英文），song：宋体（中文）
font_color: 1   #字体颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄色，6：紫色，7：青色
size: 1         #字体大小，0：自动 - 较小，1：自动 - 中等，2：自动 - 大字体
                # 9，12，16，20，24，28，32，36，40，48，56，64，72，80，88，96：各种px的大小
bold: 0         #字体样式，0：正常，1：加粗
outline: 2       #字体轮廓厚度，0～10
line_space: 'x0.5'  #行间距，0，x0.1，x0.2，x0.3，x0.4，x0.5，x0.6，x0.7，x0.8，x0.9
                    # x1，x1.2，x1.5，x1.8，x2，x2.5，x3
outline_color: 0    #边框颜色，0：黑色，1：白色，2：红色，3：蓝色，4：绿色，5：黄色，6：紫色，7：青色
text: "here is overlay of text"      #如果叠加类型为text，则生效，文字
format: ""  #如果叠加类型为time，则生效，时间格式，不填为YYYY-MM-DD hh:mm:ss
            # %m/%d/%Y %H:%M:%S，%d/%m/%Y %H:%M:%S，%Y-%m-%d，%m/%d/%Y，%d/%m/%Y，%H:%M:%S，%H:%M
image_id: ''      #如果叠加类型为image，则生效，图片编号，由“叠加层-图片-添加”接口上传
#输出
{
    err: null,  #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0 --流媒体编号
item_id = 0   --叠加项编号

params = {
    [ "osd.stream["..stream_id.."].items["..item_id.."].osd_type" ] = "text",
    [ "osd.stream["..stream_id.."].items["..item_id.."].show_when" ] = "always",
    [ "osd.stream["..stream_id.."].items["..item_id.."].position" ] = "left-
top",
    [ "osd.stream["..stream_id.."].items["..item_id.."].x" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].y" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].font" ] = "default",
    [ "osd.stream["..stream_id.."].items["..item_id.."].size" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline" ] = 2,
    [ "osd.stream["..stream_id.."].items["..item_id.."].bold" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].italic" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].text" ] = "here is
overlay of text",
    [ "osd.stream["..stream_id.."].items["..item_id.."].image_id" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].font_color" ] = 1,
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline_color" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].line_space" ] = "x0.5",
    [ "osd.stream["..stream_id.."].items["..item_id.."].format" ] = ""
}
r, count = servers["codec/OSD"]:Set(params)
--结果
r: true     count: 18
```

叠加层-项-修改

```
HTTP PUT /stream-media/overlay-items-set
#输入
stream_id: 0     #流编号，0~2，0：主码流，1：字码流，2：图片流？
item_id: 2       #叠加项编号，0~15
overlay_type: 'image' #叠加类型，text：文字，time：系统时间，image：图片，coordinates：
GPS:经/纬度
                      # altitude：GPS:高度，speed：GPS:速度，bearing：GPS:方位
image_id: 'overlay-of-image' #如果叠加类型为image，则生效，图片编号，由“叠加层-图片-添
加”接口上传
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
```

```
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0 --流媒体编号
item_id = 0   --叠加项编号

params = {
    [ "osd.stream["..stream_id.."].items["..item_id.."].osd_type" ] = "none",
    [ "osd.stream["..stream_id.."].items["..item_id.."].image_id" ] = "",
}
r, count = servers["codec/OSD"]:Set(params)
--结果
r: true     count: 2
```

叠加层-项-删除

```
HTTP DELETE /stream-media/overlay-items-erase
#输入
stream_id: 0     #流编号，0~2，0：主码流，1：字码流，2：图片流？
item_id: 2       #叠加项编号，0~15
#输出
{
    err: null, #成功null，否则错误信息
    val: true
}
```

```
--实现
client = require "coaf.client"
client.InitSocket(nil, require "coaf.cli_adapter.simple")

codec = client.New("codec", "local", "192.168.2.174:14000")
servers = { ['codec/OSD'] = codec:CreateProxy('/OSD') }

stream_id = 0 --流媒体编号
item_id = 0   --叠加项编号

params = {
    [ "osd.stream["..stream_id.."].items["..item_id.."].osd_type" ] = "none",
    [ "osd.stream["..stream_id.."].items["..item_id.."].show_when" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].position" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].x" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].y" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].font" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].size" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].bold" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].italic" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].text" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].image_id" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].font_color" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].outline_color" ] = 0,
    [ "osd.stream["..stream_id.."].items["..item_id.."].line_space" ] = "",
    [ "osd.stream["..stream_id.."].items["..item_id.."].format" ] = ""
}
r, count = servers["codec/OSD"]:Set(params)
--结果
r = true
```

```
count = 18
```

叠加层-图片-列表

```
HTTP GET /stream-media/overlay/pictures-list
#输入
#输出
{
    err: null, #成功null，否则错误信息
    val: [{
        id: 'overlay-of-picture',       #图片编号
        file_path: '/data/osd_images/overlay-of-picture/orig.png'   #文件路径，必须是WEB静态资源，用于图片预览
    },{
        id: 'overlay2-of-picture',
        file_path: '/data/osd_images/overlay2-of-picture/orig.png'
    }]
}
```

叠加层-图片-添加

```
HTTP POST /stream-media/overlay/pictures-add
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary7MA4YWxkTrZu0gW
#输入
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="text"

title
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="image"; filename="picture.png"
Content-Type: image/png

< ./picture.png
------WebKitFormBoundary7MA4YWxkTrZu0gW--
#输出
{
    err: null,
    val: {
        id: 'overlay-of-picture',   #图片编号
        file_path: '/data/osd_images/overlay-of-picture/orig.png' #文件路径，必须是WEB静态资源，用于图片预览
    }
}
```

叠加层-图片-删除

```
HTTP DELETE /stream-media/overlay/pictures-erase
#输入
id: 'overlay-of-picture'        #图片编号
#输出
{
    err: null,
    val: true
}
```

储存

磁盘-列表

```
HTTP GET /network-service/disk-list
#输入
#输出
{
    err: null,
    val: [
        {
            mounted_on: '/media/sdmmc/mmcblk0p1',   #磁盘挂载点
            type: 'vfat',   #磁盘类型
            size: '7.4G',   #总计
            used: '7.4G',   #已用
            available: '0', #可用
            use: '100%'     #使用负载
        },{
            mounted_on: '/media/nas/a',     #磁盘挂载点
            type: 'cifs',       #磁盘类型
            size: '49.4G',      #总计
            used: '34.8G',      #已用
            available: '14.6G', #可用
            use: '71%'          #使用负载
        }
    ]
}
```

```
#命令
df -h | grep /media
mount
```

磁盘CIFS-添加

```
HTTP POST /network-service/disk-cifs-add
Content-Type: application/x-www-form-urlencoded
#输入
host: '192.168.2.202'           #主机地址
shared_directory: 'Desktop'     #共享目录
username: 'Everyone'            #用户名
password: ''                    #密码
#输出
{
    err: null,
    val: true
}
```

```
#命令
mount -t cifs -o username=<用户名>,password=<密码> //<主机地址>/<共享目录>
/media/nas/<挂载目录>
```

磁盘CIFS-删除

```
HTTP DELETE /network-service/disk-cifs-erase
#输入
host: '192.168.2.202'            #主机地址
shared_directory: 'Desktop'      #共享目录
#输出
{
    err: null,
    val: true
}
```

```
#命令
unmount /media/nas/<挂载目录>
```

网络服务

WEB服务-状态

```
HTTP GET /network-service/web-status
#输入
#输出
{
    err: null,
    val: {
        http_service_port: 80,  #http服务端口
        https_service_port: 443 #https服务端口
    }
}
```

```
#查看NGINX配置文件
cat /usr/local/openrest/nginx/conf/nginx.conf
```

WEB服务-设置

```
HTTP PUT /network-service/web-set
#输入
http_service_port: 80   #http服务端口默认为80
https_service_port: 443 #https服务端口默认为443
#输出
{
    err: null,
    val: true
}
```

```
#修改NGINX配置文件
vi /usr/local/openrest/nginx/conf/nginx.conf
#重启nginx
openrest/nginx -s reload
```

TELNET服务-状态

```
HTTP GET /network-service/telnet-status
#输入
#输出
{
    err: null,
    val: 'active'       #active：活动的，inactive：不活动的。
}
```

```
#命令
systemctl status telnet.socket
```

TELNET服务-开启

```
HTTP GET /network-service/telnet-open
#输入
#输出
{
    err: null,
    val: true
}
```

```
#命令
systemctl start telnet.socket
```

TELNET服务-关闭

```
HTTP GET /network-service/telnet-close
#输入
#输出
{
    err: null,
    val: true
}
```

```
#命令
systemctl stop telnet.socket
```

静态ARP-查看

```
HTTP GET /network-service/arp-statical-list
#输入
#输出
{
    err: null,
    val: [
        {
            ip: '192.168.2.174',
            mac: '54:c9:df:9f:a8:d7'
        },
        {
            ip: '192.168.2.174',
```

```
            mac: '36:b8:df:9f:a8:d1'
        }
    ]
}
```

静态ARP-添加

```
HTTP POST /network-service/arp-statical-add
Content-Type: application/x-www-form-urlencoded
#输入
ip: '192.168.2.174'
mac: '54:c9:df:9f:a8:d7'
#输出
{
    err: null,
    val: true
}
```

静态ARP-删除

```
HTTP DELETE /network-service/arp-statical-erase
#输入
ip: '192.168.2.174'
#输出
{
    err: null,
    val: true
}
```

区域时间

状态

```
HTTP GET /timezones/time
#输入
#输出
{
    err: null,
    val: {
        local_time:'1993-07-12 12:12:00',   #时间，格式为YYYY-MM-DD HH:MM:SS
        universal_time:'2020-11-09 08:12:18',
        rtc_time:'2020-11-09 16:12:18',
        time_zone: 'Asia/Shanghai',
        System clock synchronized: 'yes'
    }
}
```

```
#命令
timedatectl status
```

时间-设置

```
HTTP Put /timezones/time-set
#输入
time: '1993-07-12 12:12:00' #时间，格式为YYYY-MM-DD HH:MM:SS
#输出
{
    err: null,
    val: true
}
```

```
#命令
timedatectl set-time <设备时间:YYYY-MM-DD HH:MM:SS>
```

时区-设置

```
HTTP Put /timezones/zones-set
#输入
timezone: "Asia/Shanghai" #时区
```

```
#命令
timedatectl set-timezone "Asia/Shanghai"
```

网络时间同步-开启

```
HTTP Put /timezones/ntp-open
#输入
#输出
{
    err: null,
    val: true
}
```

```
#命令
timedatectl set-ntp true
```

网络时间同步-关闭

```
HTTP Put /timezones/ntp-close
#输入
#输出
{
    err: null,
    val: true
}
```

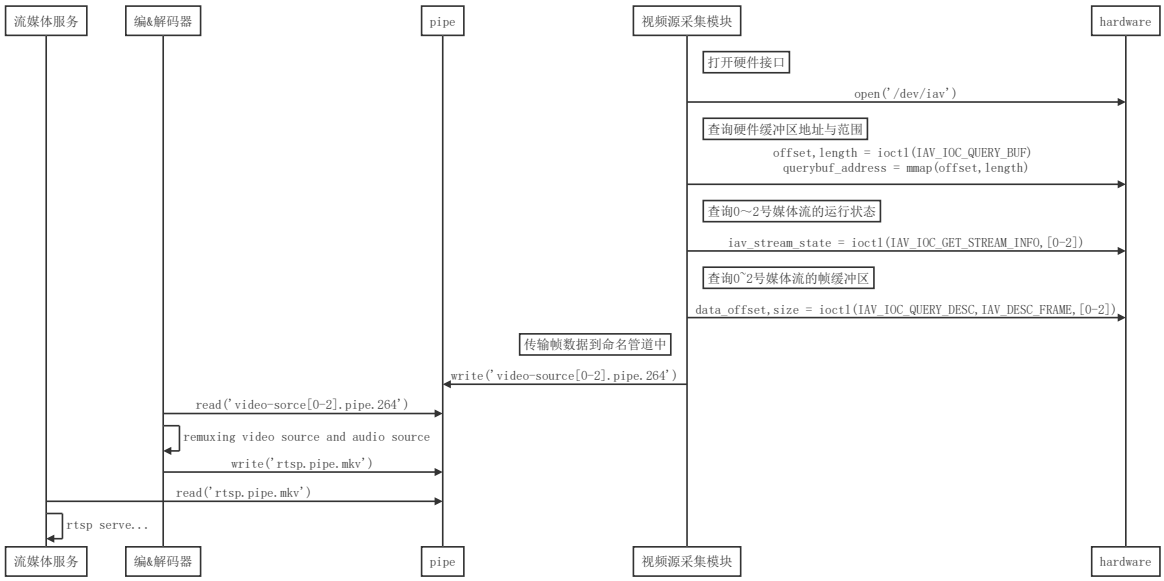
```
#命令
timedatectl set-ntp false
```

C++端

媒体源服务

视频源采集模块

```
#程序名
vrecord

#输入
stream id: 0 #流编号，0～2，0：主码流，1：子码流，2：MJPEG流
pipe name: 'video-source-0.pipe.264' #媒体输出管道名

#输出
将H264的媒体流写入命名管道中
```



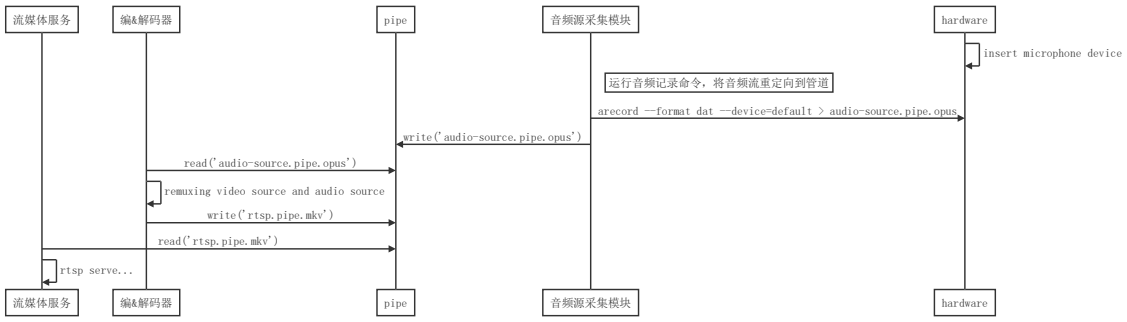

```
//视频源采集示例
extern "C"
{
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <signal.h>

#include <iav_ioctl.h>
}
#include <iostream>
#include <thread>
using namespace std;

void broken_pipe_callback(int) {}

int main()
{
    signal(SIGPIPE, broken_pipe_callback); //忽略管道关闭信号

    auto fd = open("/dev/iav", O_RDWR);
```

```
    cout << "open /dev/iav:" << fd << endl;

    iav_querybuf querybuf;
    querybuf.buf = IAV_BUFFER_BSB;
    cout << "ioctl iav_ioc_query_buf:" << ioctl(fd, IAV_IOC_QUERY_BUF,
&querybuf) << endl;
    auto querybuf_address = reinterpret_cast<char *>(mmap(nullptr,
querybuf.length, PROT_READ, MAP_SHARED, fd, querybuf.offset));
    cout << "querybuf.offset:" << querybuf_address << endl;
    cout << "querybuf.length:" << querybuf.length << endl;

    iav_stream_info stream_info;

    const auto MAX_IAV_CHANNELS = 3;
    for (auto i = 0; i < MAX_IAV_CHANNELS; i++)
    {
        stream_info.id = i; //查询流编号
        cout << "ioctl iav_ioc_stream_info:" << ioctl(fd,
IAV_IOC_GET_STREAM_INFO, &stream_info) << endl;
        cout << "stream_info.state:" << stream_info.state << endl;
    }

    iav_querydesc querydesc;
    auto &framedesc = querydesc.arg.frame;
    fill_n(reinterpret_cast<char *>(&querydesc), sizeof(querydesc), '\0');

    querydesc.qid = IAV_DESC_FRAME; //查询帧
    framedesc.id = 0;               //0：主码流H264，1：子码流H264，2：MJPEG

    cout << "mkfifo:" << mkfifo("test.264", S_IWUSR | S_IROTH) << endl;
    auto test_h264_pipe = open("test.264", O_WRONLY);
    cout << "open:" << test_h264_pipe << endl;

    while (true)
    {
        cout << "ioctl iav_ioc_query_desc:" << ioctl(fd, IAV_IOC_QUERY_DESC,
&querydesc) << endl;
        cout << "framedesc.id:" << framedesc.id << endl;
        cout << "framedesc.end:" << framedesc.stream_end << endl;
        cout << "framedesc.stream_type:" << framedesc.stream_type << endl;
        cout << "framedesc.data_address:" << querybuf.offset +
framedesc.data_addr_offset << endl;
        cout << "framedesc.size:" << framedesc.size << endl;

        auto framedesc_data_address = reinterpret_cast<char *>(querybuf_address
+ framedesc.data_addr_offset);
        auto count = write(test_h264_pipe, framedesc_data_address,
framedesc.size);
        cout << "write:" << count << endl;

        if (count == -1)
            this_thread::sleep_for(chrono::seconds(1));
    }

    close(test_h264_pipe);
    munmap(querybuf_address, querybuf.length);
    cout << "unlink:" << unlink("test.264") << endl;
```

```
    return 0;
}
```

音频源采集模块

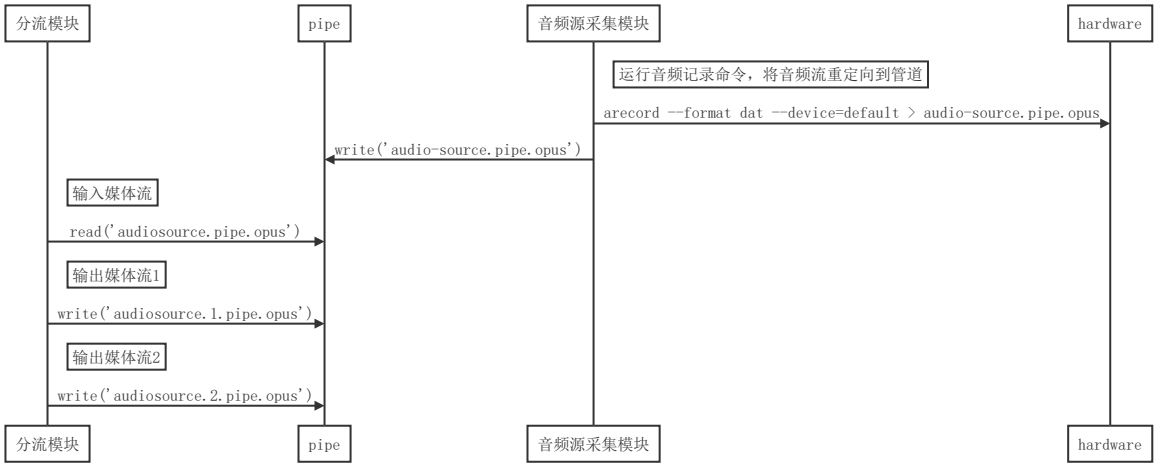
```
#程序名
arecord

#输入
format: 'dat' #采样质量48000
devide: 'default' #默认音频设备，通过arecord -l获取音频设备列表
pipe name: 'audio-source.pipe.opus' #媒体输出管道名

#输出
将音频媒体流写入命名管道中
```




```
#命令
mkfifo <文件名>
arecord --format <采样质量> --device=<音频设备> > <文件名>
```

分流模块

```
#程序名
fifo-fork
#输入
pipe name: 'audio-source.pipe.opus' #媒体流管道名
#输出
将媒体流写入多个命名管道中
#命令
fifo-fork <输入管道> -o [输出管道1] [输出管道2] ...
```




```
extern "C"
{
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>
}

#include <iostream>
#include <thread>
#include <vector>
#include <list>
#include <string>

using namespace std;

void broken_pipe_callback(int) {}

int main(int argc, char *argv[])
{
    signal(SIGPIPE, broken_pipe_callback); //忽略管道关闭信号

    auto input = argv[1];
    list<string> outputs;

    char arg = getopt(argc, argv, "o:");
    if (arg == -1)
    {
        cout << "Usage: " << argv[0] << " <input file> -o [output file] ..."
             << "Examples:" << argv[0] << " faded.264 -o 1.264 2.264 3.264"
             << endl;
        return 0;
    }

    for (auto begin = optind - 1, end = argc; begin < end; begin++)
        outputs.push_back(argv[begin]);

    mkfifo(input, S_IRUSR | S_IWUSR);
    auto input_fd = open(input, O_RDONLY);

    char buffer[1024];
    ssize_t buffer_size = 0;
```

```
    vector<int> output_fds;

    for (string output : outputs)
    {
        mkfifo(output.c_str(), S_IRUSR | S_IWUSR);
        output_fds.push_back(-1);
        auto i = output_fds.size() - 1;

        thread([i, output, &output_fds]() {
            do
            {
                if (output_fds[i] != -1)
                {
                    continue;
                }

                output_fds[i] = open(output.c_str(), O_WRONLY);
                cout << "open " << output << ' ' << output_fds[i] << endl;
            } while (this_thread::sleep_for(1s), true);
        }).detach();
    }

    while (true)
    {
        buffer_size = read(input_fd, buffer, sizeof(buffer));
        if (buffer_size == 0)
        {
            this_thread::sleep_for(1s);
            continue;
        }

        bool writen = false;

        do
        {
            for (auto &output_fd : output_fds)
            {
                if (output_fd == -1)
                    continue;

                if (-1 == write(output_fd, buffer, buffer_size))
                {
                    close(output_fd);
                    cout << "close:" << output_fd << endl;
                    output_fd = -1;
                }
                else
                {
                    writen = true;
                }
            }
        } while (!writen && (this_thread::sleep_for(1s), true));
    }

    return 0;
}
```

编&解码服务

音视频复用

```
#输入
video pipe: 'video-source-0.pipe.264' #h264视频流
audio pipe: 'audio-source.pipe.opus'  #opus音频流
#输出
media pipe: 'output.pipe.mkv'         #mkv媒体流
#命令
mkfifo <文件名>
ffmpeg -i <输入视频文件> -i <输入音频文件> -codec:v copy -codec:a copy -f matroska <
媒体文件>
ffmpeg -i <输入视频管道> -i <输入音频管道> -codec:v copy -codec:a copy -f matroska
pipe:1> <管道>
```

视频色彩调节

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
hue: 0                           #色度 0～360，默认0
brightness: 0                    #亮度 -2～2，默认0
saturation: 1                    #饱和度 -2～2，默认1
contrast: 1                      #对比度 -2～2，默认1
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v hue='h=<色度>',eq='brightness=<亮度>:saturation=<饱和
度>:contrast=<对比度>' -f h264 <视频文件>
```

视频文字叠加

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
text: "here is overlay of text"     #文本(默认)，时间：%{localtime}
x: 0    #叠加位置x，0（默认）
y: 0    #叠加位置y，0（默认）
fontfile: 'LiberationMono-Bold.ttf' #字体文件，Sans（默认）
fontsize: 16    #字体大小，各种px的大小：0～96，16（默认）
fontcolor: '0xffffffff' #字体RGB颜色，0x000000～0xffffff，黑色（默认）
borderw: 2       #文本边宽，0～10，0（默认）
bordercolor: '0xffffffff'   #文本边颜色，0x000000～0xffffff，黑色（默认）
#命令
ffmpeg -i <视频文件> -filter:v drawtext='text=<文本>:x=<X坐标>:y=<Y坐标>:fontfile=<
字体文件>:fontsize=<大小>:fontcolor=<颜色>borderw=<文本边宽>:brodercolor=<文本颜色>' -
f h264 <视频文件>
```

视频图片叠加

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
image: 'overlay.png' #叠加图片
x: 0    #叠加位置x，0（默认）
y: 0    #叠加位置y，0（默认）
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <输入视频文件> -filter:v movie='<叠加图片>',[in]overlay='<X坐标>:<Y坐标>' -
f h264 <输出视频文件>
```

视频时间叠加

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
x: 0    #叠加位置x，0（默认）
y: 0    #叠加位置y，0（默认）
fontfile: 'LiberationMono-Bold.ttf' #字体文件，Sans（默认）
fontsize: 16    #字体大小，各种px的大小：0～96，16（默认）
fontcolor: '0xffffffff' #字体RGB颜色，0x000000～0xffffff，黑色（默认）
borderw: 2       #文本边宽，0～10，0（默认）
bordercolor: '0xffffffff'   #文本边颜色，0x000000～0xffffff，黑色（默认）
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v drawtext='text=%{localtime}:x=<X坐标>:y=<Y坐标
>:fontfile=<字体文件>:fontsize=<大小>:fontcolor=<颜色>borderw=<文本边宽>:brodercolor=
<文本颜色>' -f h264 <视频文件>
```

视频裁切

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
w: 100  #宽
h: 100  #高
x: 0    #X坐标
y: 0    #Y坐标
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v crop='<宽>:<高>:<X坐标>:<Y坐标>' -f h264 <视频文件>
```

视频缩放

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
w: 100  #宽
h: 100  #高
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v scale='<宽>:<高>' -f h264 <视频文件>
```

视频旋转

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
angle: 90   #角度
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v rotate='<角度>*PI/180' -f h264 <视频文件>
```

视频镜像

```
#输入
video: 'video-source-0.pipe.264' #h264视频流
#输出
将处理后的视频流写入文件
#命令
ffmpeg -i <视频文件> -filter:v geq='p(W-X\,Y)' -f h264 <视频文件>
```

音频音量调节

```
#输入
audio: 'audio-source.pipe.opus' #opus音频流
volume: '0dB'   #音量，-10dB～10dB，0dB默认
#输出
#命令
ffmpeg  -i <视频文件> -filter:a volume='5dB' <视频文件>
```

流媒体服务

录像模块

SRT模块

RTSP模块

RTMP模块

TS-UDP模块

HLS模块

串口&USB服务