

FreeSWITCH

實例解析

杜金房 著

INVITE sip:+861234567890@example.com SIP/2.0
Via: SIP/2.0/UDP 10.20.30.40:5060;rport;branch=z9hG4bKj9Nm7v3047Ha
Max-Forwards: 70 FreeSWITCH-CN 出品
From: "Seven Du" <sip:+861234567890@example.com>;tag=B0U9ZQZQ124SK
To: <sip:+861234567890@example.com>
Call-ID: eb9f724e-9fed-1233-88ac-525400272cd0
CSeq: 77777777 INVITE

FreeSWITCH 实例解析

杜金房

版权所有，侵权必究

图书不在版编目 (NCIP) 数据

FreeSWITCH 实例解析 / 杜金房 著 / 2015.7

ISBN 7-DU-777777-7

本书包含 FreeSWITCH 的一些实例，一些大家经常问到的问题，一些相关的文章等。

本书更侧重于一些实例的实现，若你想更深入了解其中原理，读《FreeSWITCH 权威指南》。

FreeSWITCH 实例解析

作 者 杜金房

封面设计 杜金房

校 对 杜金房

排 版 杜金房

开 本 1890 毫米 × 2360 毫米

印 张 7.5

印 数 7

版 数 2015 年 7 月第 1 版 2016 年 8 月第 4 次发布

电子邮箱 freeswitch@dujinfang.com

前 言

自《FreeSWITCH 权威指南》出版以来，已经过去一年多了。在这一年多的时间里，我也收到很多的反馈，有批评的，有赞扬的，大部分还是赞扬的。

有一部分读者反映《指南》中的内容写得不够深入。我想，这部分读者应该是比较高级的读者，希望基于某个主题进行更深入地研究。但是，作为 FreeSWITCH 方面的第一本中文书，《指南》已经很厚了。不可否认，其中的某些知识点和案例，如 NAT、SIP、二次开发等，可能需要整整一章或者一本书才能写得详尽，但很显然为每个这样的知识点都写一本书是不现实的。

在设计书稿的时候，《指南》分为三个部分，其实应该分成三本书比较好，考虑到实际情况我们才把所有内容都放到一本书上了。而且，还有些内容放不下，我们只好以电子版附录的形式发布了。

说到电子版，其实还是有很多优势的。一是出版周期短，二是可以随时更新。好多朋友也都想读电子版。与此同时，我也又积累了一些文章和案例，不如索性整理一下再发出来，以飨读者。

本书的部分内容来自《指南》，部分来自于我们微信公众号或其它网络上公开发表的文章，部分来自于我在提供 FreeSWITCH 咨询过程中的一些实际案例和心得，当然，还有一些是专门写的。本书类似于一个 Cook Book，更侧重于实际应用与案例分析。如果你需要了解某些知识点的来龙去脉，再回去翻一翻《指南》，应该会更有收获。

如果你喜欢 FreeSWITCH，你肯定喜欢本书。

本书内容会不断更新，请关注本书的网站：<http://book.dujinfang.com>。你也可以订阅 FreeSWITCH-CN 微信公众号以随时了解 FreeSWITCH 和本书动态。



目 录

前 言	1
I 漫入佳境	9
1 FreeSWITCH 那些事	10
1.1 FreeSWITCH 是什么?	10
1.2 FreeSWITCH 的前世今生	11
1.3 FreeSWITCH 的版本	12
1.3.1 FreeSWITCH 1.0	12
1.3.2 FreeSWITCH 1.2	13
1.3.3 FreeSWITCH 1.4	13
1.3.4 FreeSWITCH 1.6	13
1.3.5 FreeSWITCH 1.8	13
1.3.6 FreeSWITCH 1.2 到 1.4 变化简析	13
1.4 FS-353	16
1.5 FreeSWITCH 常见问题解答	18
1.6 FreeSWITCH 基本概念和约定	29
1.6.1 信令和媒体	29
1.6.2 B2BUA	29
1.6.3 Session 和 Channel	29
1.6.4 回铃音与 Early Media	30
1.6.5 API 与 App	30

1.6.6 fs_cli	32
1.6.7 事件	32
2 多租户	34
2.1 Domain 简介	34
2.2 配置与实例	36
2.3 进阶	39
2.4 其它	39
3 FreeSWITCH 模块	41
3.1 模块列表	41
3.1.1 applications	41
3.1.2 asr_tts	44
3.1.3 codecs:	44
3.1.4 dialplans	45
3.1.5 directories	45
3.1.6 endpoints	45
3.1.7 event_handlers	46
3.1.8 formats	47
3.1.9 languages	47
3.1.10 loggers	47
3.1.11 say	48
3.1.12 timers	48
3.1.13 xml_int	48
3.2 mod_sofia	49
3.3 mod_dptools	62
3.4 mod_commands	64
3.5 mod_portaudio	65
3.6 mod_rtmp	67
3.7 mod_skypeopen	68

3.8 mod_cdr_csv	69
3.9 mod_cluechoo	70
3.10 mod_speex	72
3.11 mod_sonar	72
3.12 mod_rss	74
3.13 mod_conference	75
3.14 mod_enum	76
3.15 mod_spidermonkey	77
3.16 mod_lua和mod_v8	78
3.17 mod_RTC	79
3.18 mod_verto	80
3.19 mod_xml_rpc	84
3.20 mod_basic	86
3.21 mod_openh264	87
3.22 mod_blacklist	87

II 实例应用 92

4 技巧与实例 93	
4.1 来电转接	93
4.1.1 盲转	93
4.1.2 协商转	95
4.2 共享线路呈现	98
4.3 使用组播功能做网络广播	100
4.4 使用 FreeSWITCH 检测声音文件中的 DTMF 信息	102
4.5 号码连选	105
4.5.1 注册到运营商服务器	105
4.5.2 通过单个号码呼出	105
4.5.3 使用随机数做号码连选	106
4.5.4 使用 mod_distributor 进行连选	107

4.5.5 其它	108
4.6 收发传真	108
4.7 使用 loopback Endpoint 外呼	110
4.8 在 Web 浏览器中打电话	113
4.8.1 Flash	114
4.8.2 WebRTC	117
4.9 发送任意的 SIP 消息	122
4.9.1 SIPP	122
4.9.2 sipsak	122
4.9.3 Quaff	123
4.10 在呼叫失败的情况下如何播放语音提示	126
4.11 被叫忙的处理	128
4.12 话机上被叫号码被改变问题	131
4.13 在同一台电脑上启动多个 FreeSWITCH 实例	133
4.14 FreeSWITCH Bug 解决一例	135
5 NAT	137
5.1 FreeSWITCH NAT 问题之一	137
5.2 FreeSWITCH NAT 问题之二	143
5.3 一个 NAT 问题解决过程	144
III 他山之石	146
6 OpenSIPS	147
7 OpenSIPS 配置实例	148
7.1 负载均衡配置实例	148
IV 真刀真枪	155
8 搭建 SIPSIP 服务	157
8.1 设计规划	157

8.2 安装 FreeSWITCH	157
8.2.1 安装基本服务	157
8.2.2 安装依赖包	157
8.2.3 编译安装 FreeSWITCH	158
8.2.4 启动 FreeSWITCH	158
8.2.5 配置自启动脚本	159
8.3 配置最基本的服务	159
8.3.1 清空 Dialplan	160
8.3.2 密罐	161
8.3.3 180 和 183 服务	163
8.3.4 DNS	166
8.3.5 更多服务	168
8.3.6 Homer	169
8.3.7 CGRates	172
8.3.8 monit	172
8.3.9 iptables	174
8.3.10 从 Debian7 升级到 Debian8	174
8.3.11 munin	175
V 随笔杂谈	178
9 技术类话题	179
9.1 JSON API	179
9.2 视频直播	186
9.3 视频直播 Nginx 篇	193
9.4 FreeSWITCH 问题解决的几个例子	200
9.5 FreeSWITCH 代码最新变化	202
9.6 FreeSWITCH 与 FFmpeg	203
9.7 如何录制喜欢的电视节目	206
9.8 FreeSWITCH 精英群第 N 次活动小记	208

9.9 FreeSWITCH 第 N+1 次活动小记	211
9.10 解决一个诡异的呼叫问题	213
10 调试与排错	218
10.1 FreeSWITCH Bug 修复一例	218
10.2 将 SIP Trace 放入日志文件中	221
11 FreeSWITCH 文集	224
11.1 FreeSWITCH 开源社区指南	224
11.2 关于 FreeSWITCH 常用术语翻译的意见	226
11.3 FreeSWITCH 背后的故事(译)	228
11.4 FreeSWITCH 与 Asterisk	230
11.5 FreeSWITCH 的历史	235
11.6 我与 FreeSWITCH 的故事	241
11.7 The missing Link	246
11.8 在飞机上上网打电话	247
11.9 You pay what I know	248
11.10 180 还是 183?	249
11.11 从 3·15 看电话号码透传	254
11.12 FreeSWITCH 帮我找到了工作	256
11.13 Cluecon 2011 小记	259
11.14 ClueCon 2012	261
11.15 ClueCon 2014	269
11.16 为什么会是我?	271
11.17 为什么又是我?	272
11.18 ClueCon 2015	273
11.19 ClueCon 2016	275
VI 附录	286
1 通道变量	287

2 FreeSWITCH 架构详图	292
写在最后	293
作者简介	294
版权声明	295
广告	296
关于广告	296
FreeSWITCH 第五届开发者沙龙将于 2016 年 8 月 27 日在京举行	296
FreeSWITCH 培训 2016 夏季班（北京站）将于 2016 年 8 月 28-30 日在京举行	296
烟台小樱桃网络科技有限公司提供商业 FreeSWITCH 及 OpenSIPS 技术支持	296
烟台小樱桃网络科技有限公司是潮流网络（GrandStream）山东总代理	296
FreeSWITCH 相关图书	297
	299

第 I 部分 漸入佳境

第一章 FreeSWITCH 那些事

1.1 FreeSWITCH 是什么？

FreeSWITCH 是什么？其实很难给它下一个准确的定义。如果你是做 Web 开发的，那么，MySQL 和 Apache 大概是你最熟悉、最常用也是最基础、最不可或缺的软件，如果你是做通信的，那么，FreeSWITCH 就跟 MySQL 和 Apache 那么基础，那么鼎鼎大名。

是的，就跟上面提到的两款开源软件一样，FreeSWITCH 也是开源的。而且 FreeSWITCH 也是有定义的。FreeSWITCH 开源社区官方给它的定义是：The World's First Cross-Platform Scalable Free Multi-Protocol Soft Switch and Media Engine，翻译成人类能懂的语言就是：世界上第一个跨平台的、伸缩性极好的、开源免费的、多协议的软交换系统和媒体引擎。

首先它是**跨平台**的。它能原生地运行于 Windows、Max OS X、Linux、BSD 及 Solaris 等诸多 32/64 位平台，它具有很好的可移植性，也有人成功的将它移植到了 Linksys NLS2 平台及 Raspberry Pi¹上)。

它具有很强的**可伸缩性**—从一个简单的软电话客户端到运营商级的软交换设备几乎无所不能。

它是**免费的**，它采用 MPL 1.1²协议授权，意味着任何人都可以免费的使用并获取源代码，任何人都可以修改、发布、甚至出售自己的应用。

它支持 SIP、H323、Skype、Google Talk 等**多种通信协议**，并能很容易地与各种开源的 PBX 系统如 sipXecs、Call Weaver、Bayonne、YATE 及 Asterisk 等通信，它也可以与商用的交换系统如华为、中兴的交换机或思科、Avaya 的交换机等互通。

它可以用作一个简单的交换引擎、一个 PBX，一个媒体网关或媒体支持 IVR 的服务器，或在运营商的 IMS 网络中担当 CSCF 或 Application Server 等。

它遵循相关 RFC 并支持很多高级的 SIP 特性，如 presence、BLF、SLA 以及 TCP、TLS 和 sRTP 等。它也可以用作一个 SBC 进行透明的 SIP 代理（proxy）以支持其他媒体如 T.38 等。

它支持宽带及窄带语音编码，电话会议桥可同时支持 8、12、16、24、32 及 48kHz 的语音，可以支持立体声喇叭及麦克风，可以支持丰富的视频会议功能。

¹Raspberry Pi (<http://www.raspberrypi.org/>) 是一个装有 ARM CPU 的信用卡大小的计算机。事实上，笔者办公室的 IPPBX 就是一台运行着 FreeSWITCH 的 Raspberry Pi。

²Mozilla Public License。详见：<http://www.mozilla.org/MPL/1.1/>。

它一直紧跟时代步伐，永远支持最新的协议和最酷的技术，如最新的 WebRTC 技术、OPUS/VP8/VP9 语音和视频编码等。同时，它也提出 NDLB³，致力于支持最老旧的设备和通信技术，如传统的 TDM 话机和传真机等。

FreeSWITCH 从 1.6 版本开始支持视频转码和视频会议，FreeSWITCH 可以支持 4K 分辨率的视频会议，支持 64（甚至更大）方参会者融屏，支持录音录像流媒体推送（视频直播）以及向会议室中播放视频、PDF 文档等。

从技术上讲，它是一个**B2BUA**⁴。事实上，B2BUA 的概念会贯穿本书的始终，读者最好能好好理解它。

1.2 FreeSWITCH 的前世今生

话说混沌初开，那时还没有 FreeSWITCH，人类有史以来最早的通信方式基本上就是靠『吼』。当然，我们的老祖很是聪明，他们在生产劳动和相互协作中发明了历史上公认的最早的通信工具——『烽火台』，那都是史前的事了。



说话间就到了 19 世纪，苏格兰人亚历山大·贝尔发明了电话。从此，人们可以在被窝里窃窃私语，再也不用『吼』了。电话技术一直在进步，同时，计算机技术也一直在蓬勃发展，PC 机进入了千家万户。怎么在 PC 上打电话呢？Asterisk 以开源软件的姿态横空出世，解决了在 PC 机上打电话的问题，掀开了 21 世纪的新篇章。

2002 年，美国有一个叫 Anthony Minassale 的小伙子开始基于 Asterisk 编写一些呼叫中心类的应用。说起来，其实跟我们现在这些用 FreeSWITCH 写呼叫中心的码农差不多。Anthony 很有天分，对 Asterisk 研究的深了也经常发现一些 Bug、提交点补丁什么的。后来，它发现它写的补丁越来越

³No Device Left Behind，不让任何设备掉队。

⁴Back-to-back User Agent，背靠背的用户代理。。

多，可根本问题还是解决不了，老是死。他就跟社区里的人说，咱们写个 2.0 版吧，大家说好啊，然后就没有然后了。

Anthony 则不然，他回去后，盯着它的 Emacs 编辑器看了一个小时，敲入几行代码，他想重新写一个 Asterisk。总得起个名吧，第一个名叫 Choir，他希望该软件能像教堂里的唱诗班那样和谐。可是，没有人喜欢那个名字。他后来就改成 Pandora，也没有人喜欢。直到最后他改成 FreeSWITCH，大家都说，嗯，这个名高大上。立马注册商标，就挽起袖子开始干了。上面说的是 2005 年的事。

另外，2005 年 Anthony 还干了一件大事，那就是他搞了个网友见面会，把一批搞通信的技术大牛们聚在一起 Happy 了一把，这个网友见面会就叫做 ClueCon，从那年起每年一届都在芝加哥举行。

最初，FreeSWITCH 没什么功能，但是在 2006 年初的时候能跨平台了，至少能在 Linux、Mac 和 Windows 上原生的运行。后来，FreeSWITCH 还真能打电话了，经过努力，他们也有了第一批小白鼠用户，有不怕死的小白鼠还把 FreeSWITCH 应用到了生产系统上。

2008 年 5 月，FreeSWITCH 发布了 1.0 凤凰版。为什么叫凤凰？因为 Anthony 觉得开发的过程太痛苦了，1.0 发布有涅槃重生的感觉。

由于稳定、性能好，短短的时间，FreeSWITCH 收获了大批的用户。一晃到了 2012 年，Anthony 在 ClueCon 上发布了 FreeSWITCH 1.2 版。1.2 版是 FreeSWITCH 第一个稳定的发行版⁵。

世界瞬息万变，Google 发布了 WebRTC 标准并在 Chrome 浏览器里实现一下子改变了互联网的通信方式。自然，很多人都想让 FreeSWITCH 与 WebRTC 互通。在一些公司的赞助下，这一功能终于在 FreeSWITCH 里落地生根，2014 年春，FreeSWITCH 发布了 1.4 版，这是第一个支持 SIP over Websocket 和 WebRTC 的版本。

时间进入了 2015 年，FreeSWITCH 1.6，带来了全新的 FreeSWITCH 视频转码和视频会议功能，WebRTC 高清视频能够支持高达 4K 的分辨率。

2016 年 ClueCon，FreeSWITCH 又会给我们带来什么惊喜呢？让我们拭目以待。

1.3 FreeSWITCH 的版本

该选择哪一个版本呢？这个问题其实一言难尽。FreeSWITCH 已经发展 10 年了，在这 10 年里，有很多很多的故事，也有很多因素影响你的选择。下面，我们简单探讨一下。

1.3.1 FreeSWITCH 1.0

如果不是历史遗留项目，你肯定不想选择 FreeSWITCH 1.2 之前的版本。年代太久远了。

⁵之前的版本也稳定但是最稳定的版本永远都是开发分支的，也就是说发行版的发行速度远跟不上开发版。

1.3.2 FreeSWITCH 1.2

FreeSWITCH 1.2 很稳定，用得也多。而且，FreeSWITCH 编译起来也容易（因为还没有发生 FS-353⁶）。1.2 是基于 CentOS 5 开发的，对旧的操作系统兼容性较好。缺点是，1.2 版再也不会更新了，如果用于生产系统，将无法得到官方的安全更新和技术支持。

1.3.3 FreeSWITCH 1.4

这是当前的发行版，你应该总是用最新的版本（如，本书截稿时最新的版本是 1.4.20）。不过，由于发生了 FS-353，系统编译起来比较复杂，依赖于系统提供的一些第三方库。但是，你可以用 Debian 或 YUM 仓库中安装 FreeSWITCH，那样会自动安装依赖。1.4 版还会有安全更新，如果现在开始用的话，最好用这个版本。

1.3.4 FreeSWITCH 1.6

FreeSWITCH 1.6 增加了视频转码和视频会议支持，其它的如视频 Jitter Buffer 以及 RTCP 控制功能是新加的。所以，如果你做视频应用的话，肯定应该选择该版本。

至本书截稿时，FreeSWITCH 最新的版本是 1.6.9 版，Debian 8 上有已经编译好的安装包，参照官方的 Wiki 页面⁷很容易安装。

编译运行 FreeSWITCH 最好的环境也是 Debian 8，其它 Linux 发行版会麻烦一些，需要很多 Linux 系统方面的知识，不建议初学者使用。

如果没有特别说明，本书中的内容都是大部分版本都通用的（很显然，一些新的视频应用只有在 FreeSWITCH 1.6 中才通用）。

1.3.5 FreeSWITCH 1.8

2016 年 8 月，又一届 ClueCon 胜利闭幕。FreeSWITCH 1.8 很快就要发布了。届时，将给我们带来基于 Session 的实时文本聊天功能。

基于 Session 的实时文本聊天，将可以通过 RTP 传输，也可以通过 MSRP 传输，以及在 Verto 模块中通过 Websocket 传输。

让我们拭目以待。

1.3.6 FreeSWITCH 1.2 到 1.4 变化简析

这一节由一篇旧的文章整理而来。涵盖了从 FreeSWITCH 1.2.8 到 1.4.12 的变化。有一定参考价值。

⁶早期，FreeSWITCH 把一些第三方依赖库都放到 FreeSWITCH 代码库里，编译简单。我们后面还会专门讲这段故事。

⁷<https://freeswitch.org/confluence/display/FREESWITCH/FreeSWITCH+1.6+Video>。

FreeSWITCH 1.2.8 属于 1.2 版，而 1.4.12 是属于 1.4 版。FreeSWITCH 1.2.8 发布于 2013 年 3 月 30 日，而 1.4.12 则发布于 2014 年 10 月 10 日（都是美国时间）。下面，我们由简单来介绍一下两个版本的不同。

- FreeSWITCH 1.2 与 1.4 版最重要的差别就是 1.4 版支持 WebRTC。为了支持 WebRTC，FreeSWITCH 底层进行了很多的重构。
- 1.4 版的另一个重要变化与 FS-353 有关（参见第1.4节）。

基于上述变化，由于 WebRTC 依赖于更强的加密手段，这就依赖于更新的 OpenSSL 库，而且随着 OpenSSL 库连续爆出安全漏洞（如心脏滴血等），所以，目前 FreeSWITCH 建议使用较新版本的 OpenSSL 库（1.0.1j 以上）。

FreeSWITCH 最早是在 CentOS 5.2 上开发的，但 CentOS 5 已经太老了，FreeSWITCH 不再支持。而在更换到 CentOS 6（6.0 ~ 6.3）的过程中，遇到一些性能问题。FreeSWITCH 团队无心去解决 CentOS 的 Bug，因此，转向了更稳定的 Linux 发行版 Debian 7。在 1.4 版，FreeSWITCH 推荐的系统是 Debian 7（1.6 版引入视频特性，推荐 Debian 8）。

通过一些复杂的手段，FreeSWITCH 仍然可以在 CentOS 5 上编译。另外，FreeSWITCH 在 CentOS 6 上的性能问题从 6.4 以后应该有所改善，但没有具体的测试数据。

从 FreeSWITCH 1.2.8 到 1.4.12，一共有 3691 个提交，去掉 Merge 的提交（只提供一些合并信息，无实际修改），一共 3583 个提交。修复的 Bug（问题）大约有 1138 个（有些 Bug 是在写新代码的时候引入的）。

```
$ git log ^v1.2.8 v1.4.12 --no-merges |grep ^commit | wc -l  
3583
```

重构从 2012 年 12 月 18 日就开始了。

FreeSWITCH 首先进行了重构（commit: 592993e）。重构的核心是把原先在 `mod_sofia` (`sofia_glue.c`) 中的代码分跟媒体协商相关的代码移动到核心中去，增加了 `switch_core_media.c` 和 `switch_core_media.h`，并增加了一个 `switch_core_media_handle` 用于在外部模块中引用核心中的媒体功能。

另一个重构在于，把 RTP Flags 从一个 32 位的整数变成了一个数组实现。这是因为 32 位整数最多只能描述 32 个标志，已经不够用了。commit: 330f68d。

至 2013 年 1 月 22 日，该重构基本完成。commit: b1c855e1。

接下来，便是往 Sofia 协议栈里加入 SIP Over Websocket 支持。2013-01-24 commit: a70aa8f。另外，也在媒体引擎中增加了 DTLS 支持。DTLS 是 TLS（基于 TCP）的 UDP 版，以便支持加密的 RTP。

至 2013-02-26，在 FireFox 上可以用 WebRTC 打电话了。

由于重构，原先的一些通道变量失去了原来的意义，因此，原来的 `sip_` 开头的一些通道变量如 `sip_local_sdp_str` 统一改成了以 `rtp_` 开头的，如 `rtp_local_sdp_str`。这在从 1.2 转到 1.4 时，如果依赖了这些通道变量，要注意。commit: 4b3aa3。

2013-03-29，增加了 `mod_b64` 模块，FreeSWITCH 间可以考虑以 base64 的数据模式通信。

2013-04-02，SpanDSP 库也进行了大规模的重构，如支持彩色传真等 commit: 1757331。FreeSWITCH 使用该库支持传真，G726 编码等。

2013-04-01，增加了一个 `mod_translate` 模块，用于号码翻译。

2013-04-22，增加了 FreeSWITCH Portal 功能，是一个极其简单的内置于 FreeSWITCH 的图形界面。

2013 年 5 月份的时候，有一些比较大的 Code Review，修复了一些内存泄露等问题。

2013-06-17 及 07-11 重构了一个视频相关的代码，视频会单独在一个核心线程 (`core_video_thread`) 中处理，外转的视频代码就简单了很多。同时，构了视频刷新 (`video_refresh`) 的代码，这些代码用于向对端设备『要』一个关键帧，对于 SIP，使用 SIP 的 Video Fast Update 实现，对于 WebRTC，则使用 FIR (RTCP) 实现。

2013-07-14 增加了 `zh_CN` Say 接口，更加符合中文发音习惯。

2013-07-15 `mod_conference` 在 floor 的基础上增加了 `vid-floor`，将视频和音频的 floor 分开。

2013-07-16 WebRTC 相关的视频修改，Chrome 中有一个 Bug，在收到视频的源或时间戳变化时会卡住 5 分钟不动，而这种变化在 FreeSWITCH 的视频会议中非常常见，因此 FreeSWITCH 在 RTP 协议栈中重写了时间戳。该重写可能会影响一部分依赖于原始时间戳的客户端。

2013-07-19 增加了最后 5 分钟的 `peak_sps` 统计。

2013-10-01 默认使用 Event Dispatch，在内核中提高事件发送的性能。

2013-10-16 增加了 JSON API Interface，有一个大的重构。

2013-10-17 把 DTMF 的音量从 7 改为 13。

2013-11-08 有一次大的关于编码协商的重构。

2013-11-12 修复 ESL 库的一个内在泄漏问题。

2013-11-13 `mod_lua` 升级到 Lua 5.2。

2013-11-12 修正了与 Polycom 终端相关的媒体协商问题，支持在 200 OK 中返回多个编码的情况。

2013-12-15 `uuid_debug_audio` 改成 `uuid_debug_media` 并增加了 `vread|vwrite|vboth|all` 选项。

2014-01-14 增加了一个新模块 `mod_v8`。

2014-02-25 把mod_speex的功能移到了核心中。

2014-02 修正 FreeSWITCH 在 BSD 系列 UNIX 上的编译。

2014-02-27 大部分模块的编译都改为automake处理，原来是直接写的 Makefile。

2014-03-07 configure.in改为configure.ac，适应新的autotools。

2014-03-09 替换了核心的哈希代码，原来的哈希使用的是 SQLite 中的实现，现在改为使用 OpenZAP/FreeTDM 中的代码，因而核对对 SQLite 的依赖变小。对应的哈希函数原型也略有变化

2014-03-15 支持 srkdir build。

2014-03-17 stfu的代码原位于libs/ 中，现在移到了核心中，因为它不是一个 Lib。stfu是做 Jitter Buffer 用的。

2014-03 这个月主要是解决跨平台编译的问题。

2014-03-31 增加了 mod_basic。

2014-04 及 05，FreeSWITCH 团队进行了大量的 Code Review，修复了好多用肉眼能看出来的问题，如空指针、无效的赋值、内存泄露等。

2014-5-28 mod_fifo重构了一些代码，并增加了很多注释。

2014-06-12 开始增加立体声（双声道）支持。

2014-05-28 增加mod_rtc。

2014-06-11 增加mod_verto。

2014-06-30 mod_conference中增加openal，支持 3D 的音频会议。

2014-07-07 修改了默认的 Dialplan，如果不修改默认的密码 1234 的话，它会在日志中打印错误提示信息，并且电话会暂停 10s 再继续。

2014-07 主要是mod_verto模块对应的 Javascript 和 HTML 代码更新。

2014-08-26 增强了会议录音功能。

2014-09 增强了录音，避免出现两个声道不对称的情况。

2014-10 增强了 Jitter Buffer 相关的代码。

1.4 FS-353

FS-353可以算得上是 FreeSWITCH 史上最大的争论了，该争论的主要内容是，FreeSWITCH 用到了好多第三方的库，那么，这些库的代码到底应该是放在 FreeSWITCH 的代码库中，静态编译连接，还是应该动态连接操作系统上提供的库。

对于，该话题，仁者见仁。Roman 在它的文章中说⁸，在一个理想的世界里，应该只有一个操作系统，并且只有一个唯一实体管理着相同的环境。而 Anthony 说，我们所处的不是一个理想的世界，我们用到了许多第三方软件库，但不同的软件库在不同的操作系统上都有不同的版本，也有不同的运行期参数，因而，为了能让 FreeSWITCH 稳定的运行，最简单最直接的办法就是将这些依赖的库的代码直接放到 FreeSWITCH 的代码库中，这样，FreeSWITCH 可以选择适当的编译参数，并且打上相应的补丁。

但是，这便带来了 FS-353 的争论。FS-353 是在 2010 年 1 月份提出来的。它提到的不可否认的理由就是，很多操作系统发行套件（如 Debian），不允许有重复的库。也就是说，FreeSWITCH 将依赖的库静态编译，连接后的 FreeSWITCH 发布包将无法直接包括在操作系统的发行版里。这样，你就没法使用简单的方法如 apt-get 或 yum 方式安装 FreeSWITCH 软件包了。

当然，FreeSWITCH 的开发人员并不是非常固执，他们说，如果让 FreeSWITCH 不使用自己打包版本的库，而使用操作系统提供的库，那么为了能让 FreeSWITCH 在尽可能多的操作系统以及发行套件上运行，将会花很大的精力，所以，他们需要社区成员的帮助，这些帮助需要实实在在的提供解决方案，而不是只提需求和建议。因此，FS-353 便一直耽搁下来。

说到这里，我们不妨举个例子。FreeSWITCH 使用 SQLite 存储一些数据，并且使用了 SQLite 的哈希函数。而 SQLite 官方版有内存泄露，甚至在大并发的情况下会导致崩溃。因此，FreeSWITCH 自己为 SQLite 打了一些补丁，但由于各种原因，这些补丁并没有合并到官方的版本里去，只是在 FreeSWITCH 代码库中。简单来说，就是在用 FreeSWITCH 时，如果同时用 FreeSWITCH 代码库中的 SQLite 代码则不崩溃，而如果是用系统版本的，则会崩溃。

当然，上面只是个插曲，我们再接着说 FS-353。

后来，随着时间的推移，FreeSWITCH 社区中还是有人将 FreeSWITCH 打包到各 Linux 发行套件的发行版中。因而，即使没有人站出来说要对 FS-353 负责，FreeSWITCH 社区也挑起了这杆大旗。在这方面，最典型的代表就是 Travis Cross。Travis 也是 FreeSWITCH 社区里的专家，在 FreeSWITCH 代码库维护、安全及跨平台编译方面都有很大贡献（而且长得很帅）。在他的推动下，在 FreeSWITCH 开发者多次开会之后，终于决定了不再坚持把其它各种库的源代码再放到 FreeSWITCH 代码库中，而是使用系统上已有的库。当然，如前面所料到的，这一过程也是极其艰辛的。除了为了解决我们之前所说的 SQLite 的问题（升级到新版本的库，并重写了 FreeSWITCH 中的哈希表函数，不再依赖 SQLite 中的），还要解决的就是适应各种平台，也就是花很多时间写相关的 autoconf/automake 脚本以最大程度的做跨平台支持。

说到这里，如果你还是一头雾水的话，容我再多解释几句——之前，一些库代码如 SQLite、PCRE 等，FreeSWITCH 都有自己的版本，在自己的代码库里，下载源代码后，很容易编译成功。而现在，你需要自己在自己的操作系统上安装这些库，并且保证 FreeSWITCH 在编译时能找到他们，因而，对从源代码编译 FreeSWITCH 的用户来说提了更多的要求，也就是说，FreeSWITCH 比之前编译更复杂了。最常遇到的问题就是—在编译 FreeSWITCH 时它会提示你缺少各种库。

当然，由于这些操作系统上本来就有这些库，因而，一般情况下安装它们也不是太麻烦，一般

⁸参见第11.7节。

的`apt-get install`或`yum install`就能搞定。但二般的情况，就不一定了。比方说 FreeSWITCH 官方最新的版本（1.4 以上）不再支持 CentOS 5 以及 Debian 6 等旧的 Linux 版本（1.2 版仍然支持）⁹。

总之，各种方法各有各自的坏处和好处，但现在，我们看到，FreeSWITCH 1.4 有了自己的 Debian 库，直接用`apt-get install`安装起来已经很方便了。

最后，如果你对该话题还有疑问的话，下面作者当时跟 Anthony 的对话或许对你有帮助：

Seven: I think you meant to has the bundled versions default, and add options to use system versions, but now it looks like use system versions only and the bundled code will be get removed from the source tree, right?

杜：我想，你的本意是使用 FreeSWITCH 自带的（依赖库）版本，并且增加一些选项让别人可以自由选择使用系统提供的版本，但现在看起来，FreeSWITCH 将只支持系统版本，而 FreeSWITCH 自带的库代码将会被删除，是这样吗？

Anthony: yes that was the result of debating

安：是的，这是争论后的结果。

Anthony: we are not going to burden ourselves with making depends work everywhere

安：我们不再为了让这些依赖在任何地方都能运行而折磨自己了。

Seven: Well, that's a result, and anyway, when FS can goto distros less people will need to build from source for sure

杜：好吧，事已至此。不过，话又说回来，当 FreeSWITCH 能打包到各种发行版以后，肯定越来越少的人们需要从源代码自己编译了

Anthony: yes

安：是的

Anthony: that's the reasoning

安：就是这个原因

1.5 FreeSWITCH 常见问题解答

本文的部分内容选自 FreeSWITCH 官方 FAQ¹⁰，部分问题是大家在 QQ 群、微信或邮件列表中经常问到的问题，也有媒体对 Anthony 的采访等。我们简单整理了一下。

⁹这应该是理所当然的，因为这些 Linux 发行版的发行方都已经不再支持这些系统了。当然，并不是说新的 FreeSWITCH 版本不能在以上旧系统上使用，只是官方不再解答任何这方面的问题，如果你自己确实有需要的话自己想办法应该也是可以解决的。

¹⁰参见<https://freeswitch.org/confluence/display/FREESWITCH/FAQ>。

1. 什么是 FreeSWITCH? FreeSWITCH 能做什么?

根据官方的定义, FreeSWITCH 是世界上第一个跨平台的、伸缩性极好的、开源免费的、多协议的电话软交换平台。FreeSWITCH 中实现了通过 SIP、IAX2、H.323、Skype、Jingle (Google Talk)、本地声卡、以及 TDM 语音卡等协议及接口的互连互通。你可以把它用做一个 VoIP 通信服务器, 协议转换网关, 语音及视频会议服务器、执行由 JavaScript、Perl、Lua 或 C# 写的 IVR 语音菜单脚本的 IVR 服务器等。另外它也具有丰富的接口与其它系统集成。详见本书 1.2.1 节的有关内容。

2. FreeSWITCH™ 是用什么语言写的?

很多种语言。核心是用 C 写的。大部分的模块是 C 和 C++。某些模块则使用了其他一些语言, 包括但不限于 JavaScript/ECMAScript、Lua、Perl、Python、Ruby、Java 和.NET。

3. 你 (指 Anthony) 怎么评价你以前做过的 Asterisk 开发?

Anthony: 那些工作并没有白费。我有时还在使用它, 甚至有时我还提供这方面的咨询服务。我花了很多年贡献代码, 我也为 Asterisk 开发了许多第三方的模块, 现在在我的[asterisk stuff](#) 页面还可以找到。我只是简单地认为 FreeSWITCH 是电话的未来。

4. FreeSWITCH 与 Asterisk 有何异同?

Asterisk 是一个 PBX 而 FreeSWITCH 是一个软交换。参见附录 I (FreeSWITCH 与 Asterisk)。

5. PBX 与软交换的区别是什么? 只是语义问题吗?

一个 PBX 通常用于公司内部提供小型的语音信箱、分机、电话会议等服务。它主要关注于不同的分机间能互相通信。而一个软交换是一个连接多种网络的设备, 通常可以路由不同协议的电话, 把电话从一个终端路由到另一个终端 (如另一个 PBX)。诚然, FreeSWITCH 也可以作为一个 PBX 使用, 但它的功能远不止于此。FreeSWITCH 可以加载很多不同的模块, 运行起来相当于一个包含好多 PBX 的集群。通常的 PBX 都将全部 PBX 功能置于单一的核心中, 这使得它们在想用作软交换时比 FreeSWITCH 要困难地多。

6. 您怎么看待 FreeSWITCH 与其它同类竞争系统 (如 Asterisk、Yate) 的关系?

世界是多样性的, 这意味着 FreeSWITCH 的成功并不一定需要其它系统的牺牲。事实上, 很多人把 FreeSWITCH 用于他们现有的通信系统中, 作为一个重要组件或必要的补充。

7. FreeSWITCH 可以商用吗?

FreeSWITCH 采用 MPL1.2 授权协议。我们都不是法律专家, 不过, 它比 GPL 协议的商业限制要少很多。

8. 你们做该项目多久了？能简单说一下它的历史吗？

最早的发行版本是在 2006 年，只有很少的功能。实际上 Anthony Minessale 从 2005 年 10 月起就利用业余时间开发。FreeSWITCH 项目最初于 2006 年 1 月在 O'Reilly Media's ETEL 会议上发布。FreeSWITCH 的第一个官方的 1.0.0 版(Phoenix) 发布于 2008 年 5 月 26 日。2008 年 7 月 24 日发布了一个小的更新版。1.2.1 版发布于 2012 年 8 月。Anthony 在 ClueCon 2012 上宣布了 1.2.0 版的发布，FreeSWITCH 开发团队开始维护稳定版(1.2 版) 以及开发版(1.5 版) 两个分支。目前最新的版本是 1.5.7 版，大部分代码将进入稳定版 (1.4 版)。

9. 什么是 ClueCon？

ClueCon (<http://www.cluecon.com>) 是电话系统开发者的年度盛会，每年 8 月在芝加哥举行。会上会有来自不同的 VoIP 项目的领袖及其他开发者展示、讨论及交流意见。它为期三天，是一个可以在白天交流技术，晚上则享受芝加哥的美景的绝好的机会。

10. FreeSWITCH 能同时支持多少路通话？有基准测试吗？

这依赖于你的程序以及配置。你需要用你的程序进行压力测试来获取你的极限。你所能做的完全依赖于你的需求。请不要在 FreeSWITCH 邮件列表中问这样的问题，因为你总是会得到相同的官方回答：『我们只对每一个特定的 FreeSWITCH 部署进行基准测试，因为不同的部署方式会得到不同的值。如果你需要，你可以获取该项目的商业支持。我们曾经历过回答这种问题的教训，所以我们的策略是不在公共的论坛上对该问题发表任何意见』。

11. FreeSWITCH 一定要以 root 用户运行吗？

FreeSWITCH 可以以任何用户运行，只要它有相关目录的写入权限。

12. 在 Sofia 呼叫字符串中，『%』和『@』有何不同？

这很简单，你可以有多种选择。如果 `domain` 是一个 Profile 的别名，那你可以用 `sofia/domain_name/user`；如果 `domain` 不是别名，那可你可以使用 `sofia/profile_name/user%domain`；但如果你想呼叫一个远端的 SIP URI，并且对方不需要认证，那么你可以直接使用 `sofia/profile_name/remoteuser@remoteip`。详见《FreeSWITCH 权威指南》第 4.5.2 节及 10.4.1 节。

13. \${var}与\$\$var有何不同？

`${var}`会在每次执行到 Dialplan 时进行扩展，而 `$$var`则是在模块加载或 `reloadxml` 时一次性扩展的。更详细的信息见 `conf/vars.xml` 中的注释。

14. set 和 export App 有何区别？

`set` 在当前 Channel 上设置变量，而 `export` 在两个 Channel 上 (a-leg 和 b-leg) 都设置。当然，你也可以只在 b-leg 上设置，如：

```
<action application="export" data="nolocal:foo=bar"/>
```

15. 我的 FreeSWITCH 不响应任何 SIP 请求，我也用tcpdump检查了，发送端是正常的，但 FreeSWITCH 就是没有反应，怎么回事呢？

这是新手常见的问题，一般来说是你的防火墙惹得祸，在tcpdump中能看到防火墙之外的数据包。你最好关掉防火墙试试，如使用『`service iptables stop`』或『`/etc/init.d/iptables stop`』命令。在 Windows 上也可以尝试关掉防火墙。等你熟悉 FreeSWITCH 以后，再尝试自己添加相关的防火墙规则。

16. 我刚装好了 FreeSWITCH，但在启动的时候显示错误：SQL ERR[no such table: <table_name>

通常这条信息之后会有一条『`Auto Generating Table!`』的信息。那说明这是正常的。因为你是第一次使用，当 FreeSWITCH 找不到 SQLite 数据库或表时，它会自己创建。只要以后重启 FreeSWITCH 时不再出该错误，就没事。

17. `show channels`、`conference list`、以及其他控制台命令等什么也显示不出来。

可能是核心数据库的结构乱了，这些命令都是从数据库中获取数据的。删掉所有数据库（在 Linux/Unix 上用`rm -rf /usr/local/freeswitch/db/*`重启 FreeSWITCH 试试看。

18. 如何调试 SIP？

在控制台上执行『`sofia global siptrace on`』。另参见：<https://freeswitch.org/confluence/display/FREESWITCH/Debugging> 及<https://freeswitch.org/confluence/display/FREESWITCH/Sofia+SIP+Stack>。

19. 我收到『`Invalid Application <name>`』是什么意思？

该错误最可能的原因是你没有加载正确的模块。

20. ICMP error是什么错误？

由于某种原因，你的连接请求被拒绝了，详见http://wiki.freeswitch.org/wiki/Connection_Refused。

21. 我在日志中看到『`File has 2 channels, muxing to mono will occur.`』，是什么错误？

你的声音文件有两个声道，而某些通话（Channel）只支持一个，所以 FreeSWITCH 会自动替你混音。如果你不想看到那条警告或者在乎性能的话，把你的声音文件转成单声道的。

22. 我看到『Over Session Rate of 30』，是什么意思？

说明你呼的太快了。该数字是对每秒钟新产生的呼叫数量的限制。如果你的服务器足够强劲，你可以使用`fsctl sps 100`来将该限制改成100（临时生效），或者修改`switch.conf.xml`中对应的`max-session-per-second`让它永久生效。

23. 在哪里下载 FreeSWITCH？

在大多数平台上都有已编译好的安装包，见：http://wiki.freeswitch.org/wiki/Download_FreeSWITCH。

另外，FreeSWITCH 是开源的，因而可以通过源代码编译。源代码库（Git）的克隆地址是：<https://freeswitch.org/stash/scm/fs/freeswitch.git>。

24. Pastebin 是什么？

一个服务站点，用于粘贴日志，以便别人帮你查看问题。它的地址是<http://pastebin.freeswitch.org>。用户名是`pastebin`密码是`freeswitch`。

25. 我遇到『IP x.x.x.x Rejected by acl domains. Falling back to Digest auth.』是怎么回事。

一般是没事。如果开启了 ACL 认证方式，而被 ACL 拒绝的话，便会继续使用 Digest 进行认证。
参见第 10.5 节及 13.2.5 节。

26. 我遇到『NATIVE SQL ERR [database is locked]』。

可能是 SQLite 数据库损坏（如非法关机等），删掉数据表（`rm /usr/local/freeswitch/db/*.db`）并重启 FreeSWITCH。

27. 我遇到『The likely causes for this are: 1) Another application is already listening on the specified address. 2) The IP the profile is attempting to bind to is not local to this system.』，是什么错误？

说明 Sofia Profile 启动出错，可能是 Sofia 欲使用的端口已被别的程序占用，或者绑定 IP 不是本地的 IP。可以用`ifconfig`或`netstat`检查一下，如`netstat -an|grep 5060`等。

28. 我遇到『LUKE: I'm hit, but not bad. LUKE'S VOICE: ... saving the switch from certain doom.』，怎么办？

多半是你该换一台好一点的服务器。或者，不换的话改一个较小的`sps`值。参见问题 22。

29. 我在用源代码安装时出现『no usable zlib; please install zlib devel package』，是怎么回事？

说明你的系统上没有**zlib**的开发包，参照第 3.2.1 节的内容安装依赖的系统包。

30. FreeSWITCH 默认都会占用哪些端口？这些端口可以改变吗？

TCP/UDP:5060 (SIP)、TCP/UDP:5080 (SIP)，可以在**vars.xml**中修改；TCP:8021 (ESL)，可以在**event_socket.conf.xml**中修改。如果加载了**mod_xml_rpc**，还会有 TCP:8000 (HTTP)，在**xml_rpc.conf.xml**中修改。默认 RTP 使用的 UDP 端口范围是：16384 ~ 32768，可以在**switch.conf.xml**中修改。

31. FreeSWITCH 可以对接 IMS 或在 IMS 中使用吗？

是的，它可以与标准的 IMS 对接，参见第 14.2 节。也可以做为 IMS 中的一个 AS (Application Server)，当然，发挥一下创造性，用它当个 CSCF 也应该是可行的。

32. 我如何让 FreeSWITCH 运行在后台？

```
# freeswitch -nc
```

33. 当 FreeSWITCH 运行在后台时，是否有类似 telnet 的客户端能连上去呢？

是的。只要**mod_event_socket**模块已加载（默认），你就可以使用**fs_cli**连上去。它在 FreeSWITCH 的**bin**目录中。

34. FreeSWITCH 运行在后台时，我如何停止它呢？

在操作系统上使用命令：**freeswitch -stop**或在 FreeSWITCH 控制台中运行**shutdown**。

35. 如何让 FreeSWITCH 以更高的优先级运行？

FreeSWITCH 会自动选择最适合你服务器的模式运行，当然，你也可以强制它使用最高优先级，如：**freeswitch -hp**。

36. 如何将 FreeSWITCH 注册为一个 Windows 服务？

在你安装 FreeSWITCH 的路径中，执行**FreeswitchConsole - install**，或者，删除该服务：**FreeswitchConsole -uninstall**。

现在，该服务安装在『网络服务』项目中，在某些机器上，该项目可能没有足够的权限来运行 FreeSWITCH。在这种情况下，你需要修改它所属的用户。双击服务项目，到『登录』标签，将其修改为一个合适的用户，如『本地系统账户』或你为 FreeSWITCH 专门建立的新账户。

你可以在命令行模式下启动和停止 FreeSWITCH:

```
net start FreeswitchConsole  
net stop FreeswitchConsole
```

如果使用『`FreeswitchConsole -install`』建立的启动项目不能启动，还可以试试其他办法，如：<http://sw4me.com/wiki/Winserv>。下载 `winserv` 并放到某一位置，如『`C:\Program Files`』。然后可以使用以下命令安装服务：

```
C:\\\\Program Files\\\\winserv.exe\" install FreeSWITCH "C:\\Program Files\\FreeSWITCH\\FreeswitchConsole.exe" -nc
```

37. 如何在一台服务器上运行多个 FreeSWITCH 实例?

你需要有一组新的配置文件，并且有独立的 `conf`、`db`、`log` 等目录。注意，因为你想在同一个服务器上运行多个实例，每个实例中的配置文件中的端口等都不能冲突。配置好后，使用 `-conf`、`-db`、`-log`、`-run` 之类的指定特别的目录即可，如：

```
freeswitch -conf /tmp/conf -log /tmp/log -db /tmp/db -run /tmp/run
```

详情参见《FreeSWITCH 权威指南》第 13.1 节。

38. 如何对接多台 FreeSWITCH?

参见《FreeSWITCH 权威指南》第 13 章。

39. 它是否能运行在 Amazon Elastic Cloud (EC2) 上?

是的，只是注意选择其高版本的，以获得比较好的性能及低延迟。见：http://wiki.freeswitch.org/wiki/Amazon_EC2。

40. 它能运行在虚拟机里吗？如 Xen、VMWare 或 VirtualBox。

是的，EC2 就是用的 Xen。FreeSWITCH 是一个实时通信的程序，因此对系统时钟的精确度要求较高，而大部分虚拟机都会影响时钟精度，因此，如果在 VMWare 中运行，请调整 VMWare 的硬件时钟选项。有人也成功在 VirtualBox 中运行 FreeSWITCH。

如果你需要高并发但在虚拟机中又无法达到的话，请考虑在物理机中运行 FreeSWITCH。

41. 我不想安装到 `/usr/local/freeswitch/`，如何更改安装路径?

以使用以下命令进行配置：

```
./configure --prefix=/your/install/dir
```

42. 我如何选择编译哪些模块？

修改源代码目录中的 modules.conf，把你想要编译的模块前面的『#』去掉，把不想编译的模块前面加个『#』。在 Windows 系统上你需要使用 Visual Studio 提供的方法在 Configuration Manager 中修改模块依赖关系。

43. 我没有 Microsoft Visual C++，在 Windows 上是否有已经编译好的版本呢？

有：到http://files.freeswitch.org/windows_installer/installer/ 选择合适的版本。

44. 在长时间收不到 RTP 后可以自动挂断电话吗？

是的。在 Sofia Profile 有两个参数管这个事：rtp-timeout-sec 和 rtp-hold-timeout-sec。

45. 如何在 FS 控制台上看到 SIP 用户的注册情况？

可以在控制台上使用如下命令显示 Profile 的用户注册信息：

```
freeswitch> sofia status profile internal reg
```

46. FreeSWITCH 支持 TDM 硬件吗？如模拟线路（FXS/FSO）、ISDN BRI/BRA、PRI（E1/T1）等？

是的，通过 FreeTDM 模块（mod_freetdm）。详见<http://wiki.freeswitch.org/wiki/FreeTDM>。

47. 如何在 FreeSWITCH 中发起一个呼叫？

使用 originate 命令，参见《FreeSWITCH 权威指南》第 4.5 节及第 10.4 节。

48. reloadxml 能重载所有 XML 文件吗？

它只是将所有 XML 加载到内存，并不意味着所有的改变都生效。拨号计划和用户目录会刷新，它也会触发一个事件，如果某些模块认识这个事件，它可能会重新加载（如 ENUM 模块）。而 Sofia Profile 的设置不会更新。但你可以使用 Sofia 命令使其刷新（如『sofia profile internal rescan』），有些改变需要重启某个 Sofia Profile。会议设置则会在下次创建一个会议时生效。当会议正在进行时不会起作用。

49. 我如何把 Endpoints 放到不同的 Context 中，而不同的 Context 又有不同的 Extension ?

有几种不同的实现方法：

- 让每个 Profile 对应一个 Context，每一次都需要一个独立的 IP:Port。
- 在注册数据中使用不同的 Domain，它会自动路由 Context。
- 把所有电话都指到一个公共的 Context 中，并使用`execute_extension`或`transferApp`转移
到其他地方。
- 把它们送到一个 IVR，然后决定下一步去哪里。
- 使用`xml_curl`建立动态的 Dialplan，根据你知道的呼叫数据来决定下一步应该做什么。

50. 如何在整个服务器上使用单一的domain?

如果你想对所有请求提供服务，并且在单一的domain下，你可以找到`sip_profiles/internal.xml`中的`force-register-domain`一行，去掉该行的注释，并在`vars.xml`中设置对应的domain即可。如果你想让所有注册用户都能在同一个domain中列出来（或者排序），使用`force-register-db-domain`参数，如：

详见第二章多租户。

51. 是否有一个配置 FreeSWITCH 的图形界面？

有几个，参见《FreeSWITCH 权威指南》第 10.6 节。

52. 有帮助文档吗？

是的，FreeSWITCH 的 Wiki: <http://confluence.freeswitch.org>。另外，本书就是最好的中文文档。其它的中文的文档见：<http://www.freeswitch.org.cn>。

53. 有没有关于排错与汇报 BUG 的指南？我该从哪里开始呢？

从这里开始，Reporting Bugs: http://wiki.freeswitch.org/wiki/Reporting_Bugs，它会回答你很多问题。

54. 能不能讲一下配置 IVR，如何用 Javascript 控制 IVR 功能？比如电话进到 FreeSWITCH，如何控制电话到各个分机，通过 Javascript 来实现多级自动语音导航，并实现与数据库中的配置参数实现用户自动操作语音应答功能。

请看《FreeSWITCH 权威指南》，上面虽然讲的 Javascript 不多，但是参照 Lua 的例子做就好了。在本书后面我们可以多讲一些 Javascript 的例子。

55. 能不能讲讲 FreeSWITCH 里的消息或信息传递机制或设计原理讲讲呢？

在《FreeSWITCH 权威指南》里源代码相关的章节里已经讲了一些，本书后面可以再讲得深入一些。

56. FreeSWITCH 是否可以与 Polycom 相结合?

应该能，Polycom 支持标准的 SIP。另外，我们也有一个模块支持通过 H323 与 Polycom 互通。

57. 请问，想在 Web 页面做来电弹屏的话，有哪些方式？

所谓来电弹屏其实就是获取来电事件，在界面上弹出相关的客户信息。实现的方式有很多，可以说跟 FreeSWITCH 没有关系，这是个 Web 问题。

其实无外乎两点，用 Ajax 轮循或是用双向的 Socket 如 WebSocket 技术实时推送消息。Web 前端收到消息后再用 Javascript 渲染相关的页面。在后台，可以用 ESL 等方式获取消息再推送到前端。不过，现在，有了 WebSocket，也可以直接在 FreeSWITCH 中给前端推送消息了。具体可以参考《FreeSWITCH 权威指南》上提到的 FreeSWITCH Portal（支持 Ajax 轮循和 WebSocket，当然，Portal 并不支持所谓的『弹屏』，但如我上面所说，你收到相关的信息了，还不是想『弹』什么就弹什么吗？）或 mod_verto，Verto 的 Demo 里有直接推送事件的例子，但是文档不多，你可以需要好好看看源代码。

58. 录音两边不同步能解决吗？如何解决？

这是一个典型无法回答的问题。FreeSWITCH 以前确实出现过录音不同步的问题，但是，你的 FreeSWITCH 版本是什么？你的操作系统是什么？你的使用场景是什么？你都是使用默认的参数还是有自己的配置？录到本地硬盘上还是录到 NFS 上还是 NAS 上？我需要问你一百个问题才知道怎么回答你。建议你先看看 FreeSWITCH 新手指南¹¹。升级到最新版本的 FreeSWITCH，如果还有问题的话汇报 Jira（不知道什么是 Jira 的话也要看 FreeSWITCH 新手指南）。

59. 源码是否可以单独编译某一模块，如果可以的话，单独编译后如何加入主体程序中？

源代码可以单独编译一个模块的。如默认的 mp3 模块 FreeSWITCH 中是不编译的，你可以使用以下命令编译：

```
make mod_shout-install
```

另外，《FreeSWITCH 权威指南》上至少介绍了两种不同的方法能解决你的问题。

60. 杜老师，我对 FreeSWITCH 和 Asterisk 都学习了一些，想问一下这两个从开发，应用，性能，硬件支持上都有什么优劣势？谢谢！

¹¹<http://bbs.freeswitch.org.cn/t/freeswitchxin-shou-zhi-nan/46>

我对 Asterisk 了解不深，因此对比就算了吧。

当然，FreeSWITCH 的作者最初也是 Asterisk 开发者之一，因此，FreeSWITCH 与 Asterisk 还是有很深的渊源的，下面仅就我对 FreeSWITCH 理解的简单回答一下。

Asterisk 是 GPL 授权和商业版双授权，FreeSWITCH 是 MPL 授权。这意味着 FreeSWITCH 的 MPL 与 Asterisk 的 GPL 对商业应用比较宽松一些。当然，你可以直接买 Asterisk 的商业授权。
免责：我不是法律专家，各种协议我也说不清楚。

- 早些年，有人说自己用一台 FreeSWITCH 机器替换掉了 10 台 Asterisk。
- FreeSWITCH 比 Asterisk 年轻。
- FreeSWITCH 的 API 非常丰富。
- FreeSWITCH 的架构和代码写的很好。
- FreeSWITCH 的社区也好。
- FreeSWITCH 的商业支持也不错。

FreeSWITCH 可伸缩性很强，跨平台，软硬件支持都相当广，如各种 UNIX, Linux, Windows, Mac 都有原生支持。小到 ARM 型的小计算机如 Raspberry Pi，大到超级计算机硬件，应该都行的。

有人这么说：Asterisk 是一个 PBX，而 FreeSWITCH 是一个软交换平台。我觉得后者所指的大一些。另外，我觉得 FreeSWITCH 不仅仅是一个软交换，它的可扩展性很强，比方说我前期讲的基于 FreeSWITCH 做的 MCU 视频会议。

你可以读读本书附录中的《FreeSWITCH 的历史》了解更多。

61. 请问在 FreeSWITCH Conference 里有人退出或掉线，怎么捕获该事件呢？

你可以订阅 `CUSTOM conference::maintenance` 事件，所有跟 Conference 有关的都有。至于怎么捕获，你可以用 ESL，或者 Lua，或者直接在 FreeSWITCH 中写模块都行。《FreeSWITCH 权威指南》上有更多详细信息。

62. 如何实现内外网互通？

这个问题太过笼统，所以，你只能得到一个笼统的答案。

实际上，不管 FreeSWITCH 如何部署，都能实现内外网互通。可能的情况有：

- FreeSWITCH 放在公网上，内网话机都向 FreeSWITCH 注册即可。
- FreeSWITCH 放在内网上，内网都向 FreeSWITCH 注册，FreeSWITCH 通过 SIP 中继跟外面的世界互通（当然你的内网要能连到外面的世界才行）。这种方式的另一个版本就是用 E1 数字中继或模拟中继，也可以互通。
- 内网外网不通，把 FreeSWITCH 架在中间，双网卡一个连内网一个连外网，两个 Profile 一个绑定到内网网卡一个绑定到外网网卡上……

63. 请问 FreeSWITCH 的各类模块有什么区别？如果要添加一个模块是放到 application 还是 Endpoints 对开发者而言有什么本质区别？

各类模块都是实现了 FreeSWITCH 的某一个或多个 INTERFACE，如 endpoint、dialplan 等。一般都是根据该模块实现的主要 INTERFACE 分类存放的。但有些模块实现了很多 INTERFACE，又不确定哪个最重要，所就，就放在了 applications 目录里。你自己写个模块，放到哪儿都行。

64. 我安装了 FreeSWITCH，也能打通电话，但是接续时间非常慢，一般都要 10 秒以上，为什么？

这是一种保护措施，FreeSWITCH 故意这么做以引起你的注意。其实只要仔细看日志，就能在日志里看到红色的提示，提示你修改默认的密码。有以下两种方式可以避免该问题：

1) 修改 `vars.xml` 中的 `default_password`，修改为除 1234 外的其它值；2) 如果你就是想用 1234 做密码，可以修改 `dialplan/default.xml`，找到 1234，修改为其它值或把整个 `extension` 部分删掉。

65. 我还有很多问题在这里没有列出，请问在哪里可以获得帮助？

免费的有 FreeSWITCH 官方的邮件列表及 IRC，中文的有邮件列表及 QQ 群（参见附录 A）。如果你需要商业支持，可以联系 FreeSWITCH Solutions：<http://www.freeswitch.com>。

1.6 FreeSWITCH 基本概念和约定

在这一节，我们看一下复习一下 FreeSWITCH 需要了解的一些基本概念和约定，以便进行下一步的学习。

1.6.1 信令和媒体

信令负责话路的建立和释放，典型的信令如 SIP、H323 等。而实际通话的内容（音频、视频、数据等）被作为媒体进行传输，典型的媒体传输协议有 RTP 等。

1.6.2 B2BUA

FreeSWITCH 是一个典型的 B2BUA。RFC 中并没有准确定义 B2BUA 的概念，它实际上就是一对 UA 的串联。当 A 发起呼叫呼叫 B 时，通话会首先到达 FreeSWITCH，FreeSWITCH 再发起到 B 的呼叫。A 端的呼叫就称为 a-leg (A 腿)，B 端的呼叫就称为 b-leg (B 腿)。当然，并不是所有的呼叫都是有两条腿，如典型的 IVR 应用一般都只有一条腿（放音、接收按键等），这时候，FreeSWITCH 就相当于一个单独的 UA；也有超过两条腿的情况，如多人电话会议等。

1.6.3 Session 和 Channel

对每一次呼叫，FreeSWITCH 都会启动一个 Session（会话，它包含 SIP 会话，SIP 会在每对 UAC-UAS 之间生成一个 SIP Session），用于控制整个呼叫，它会一直持续到通话结束。其中，每个

Session 都控制着一个 Channel (信道)，Channel 是一对 UA 间通信的实体，相当于 FreeSWITCH 的一条腿 (leg)，每个 Channel 都有一个唯一的 UUID。另外，Channel 上可以绑定一些呼叫参数，称为 Channel Variable (信道变量)。Channel 中可能包含媒体 (音频或视频流)，也可能不包含。通话时，FreeSWITCH 的作用是将两个 Channel (a-leg 和 b-leg，通常先创建的或占主动的叫 a-leg) 桥接 (bridge) 到一起，使双方可以通话。

通话中，媒体 (音频或视频) 数据流在 RTP 包中传送。一般来说，Channel 是双向的，因此，媒体流会有发送 (Send/Write) 和接收 (Receive/Read) 两个方向。

1.6.4 回铃音与 Early Media

主叫 A 呼叫被叫 B 时，信令接通后，B 端开始振铃，A 端就听到回铃音 (或彩铃，或忙音等)。在 B 端话机真正接听前，A 是没有与 B 交谈的，A 端听到的回铃音一般不计费。

详细的信息可以参阅第11.10节。

1.6.5 API 与 App

API 和 App 是 FreeSWITCH 实现的两个基本的接口 (INTERFACE)。API 相当于命令行接口，它是独立存在的命令，App 则是与一个 Channel 相关的，在 Channel 上执行的。

API

我们在 FreeSWITCH 控制台上运行的命令就是 API，如

```
status
version
help
```

其中有的命令有参数，如，我们常用的有：

```
sofia status
originate user/1000 &echo
```

有一些 API 命令可以操作 Channel，这些 API 一般是以uuid_开头的，如uuid_kill可以强制一个 Channel 挂机。

常用的 API 是在mod_commands模块中实现的，dptools 即 Dialplan Tools 的缩写。参考：https://freeswitch.org/confluence/display/FREESWITCH/mod_commands。

App

App 是 Application 的缩写，通常会在 Dialplan 里见到，如

```
<action application="answer"/>
```

其中，answer 就是一个 App，它用于对来话进行应答。

其它的还有：

```
echo  
info
```

有的 App 也有参数，如：

```
<action application="log" data="INFO 这是一条测试日志"/>  
<action application="playback" data="/tmp/test.wav"/>
```

常用的 App 是在mod_dptools模块中实现的，dptools 即 Dialplan Tools 的缩写。参考：https://freeswitch.org/confluence/display/FREESWITCH/mod_dptools。

所有的 App 都是与 Channel 相关的，它作为通话中的一方与另一方交互，所以，如果你打一个电话到 FreeSWITCH，如呼叫9196，它就在 Dialplan 中执行echo，然后相当于你是在跟echo这个 App 在通话，不同的是，echo是一个回音 App，它不会说话，但它会把『听』到的（你说的）内容原样返回回来，因此你就听到了自己的回音。所以，记往：**跟 FreeSWITCH 通信，就是跟一个 App 在交互**，就很容易理解 App 了。

有些 App 能处理媒体，如echo、playback等；有些 App 仅处理信令，如ring_ready、answer等。

有些 App 是一条腿的，如echo、playback、record等；也有两条腿的，如bridge、record_session等；还有三条腿的，如eavesdrop、three_way、conference等；当然conference还能处理更多的腿（典型应用场是多人会议）。

API 与 App 的区别、联系和约定

大部分的 App 都有对应的 API，如answer App 对应uuid_answer API、hangup App 对应uuid_kill API 等。

但并不是所有的 API 都有对应的 App。

作为一个约定，在本书中，API 的例子我都都用`freeswitch>`提示符开头；App 的例子我们都用如下的 XML 格式表示，如：

```
<action application="log" data="INFO 这是一条测试日志"/>
```

在不引起混淆的情况下，我们也可以采用简写的形式，如：

```
record /tmp/test.wav          # Dialplan 中的 App 录音  
uuid_record <uuid> start /tmp/test.wav    # 使用 API 进行录音
```

1.6.6 fs_cli

FreeSWITCH 典型的应用是一个服务器，它提供一系列的接口可以在外部对 FreeSWITCH 进行管理和维护。`fs_cli`就是一个命令行客户端，它通过 FreeSWITCH 提供的 Event Socket 接口与 FreeSWITCH 交互。`fs_cli`可以随时打开和退出。

1.6.7 事件

FreeSWITCH 使用在内部使用消息和事件机制进行进程间和模块间通信。消息机制完全是内部的，在此我们不多讲。事件机制则即在内部使用，也可以在外部使用。FreeSWITCH 内部状态变化时，或收到一些新的消息时，会产生（Fire）一些事件。事件机制是一种『生产者-消费者』模型，它的产生和处理是异步的，是一种松耦合的关系。这些事件可以在 FreeSWITCH 内部通过绑定（Bind）一定的回调函数进行捕获，即 FreeSWITCH 的核心事件系统会依次回调这些回调函数，完成相应功能。另外，在嵌入式脚本（如 Lua）中也可以订阅相关的事件并进行处理。

在 FreeSWITCH 外部，也可以通过 Event Socket 等接口订阅相关的事件，以了解 FreeSWITCH 内部发生了什么，当前呼叫的状态等。如在上一节中讲到的`fs_cli`就是一个典型的外部程序，它通过 Event Socket 与 FreeSWITCH 通信。最简单的订阅事件的方法是：

```
fs_cli> /event plain ALL
```

上述命令在`fs_cli`中执行可以订阅所有的事件。FreeSWITCH 的事件主要有两大类，一类是主事件，根据事件的名字（Event-Name）区分，如`CHANNEL_ANSWER`（应答）、`CHANNEL_HANGUP`（挂机）等；另一类是自定义事件，它的 Event-Name 永远是 CUSTOM，不同的事件类型通过子类型（Event-Subclass）来区分，如`sofia::register`（SIP 注册）、`sofia::unregister`（SIP 注销）等。

可以单独订阅某类事件，如：

```
fs_cli> /event plain CHANNEL_ANSWER
fs_cli> /event plain CUSTOM sofia::register
```

关于这些基本概念我们就复习到这里，如果需要更详细的信息，请参阅《FreeSWITCH 权威指南》。

第二章 多租户

很多人将 FreeSWITCH 用于云计算平台，而 VoIP 云计算除了要支持大规模的并发呼叫外，更重要的是要支持『多租户¹』技术。简单来讲，多租户就是在一个系统中（或更简单点，一台 FreeSWITCH 服务器上），支持多个彼此相互独立的 PBX（如属于不同公司的）应用，这些不同的 PBX 中可能有相同的分机号，而不会产生冲突。

FreeSWITCH 在设计之初就考虑到了这一点，它是用 Domain 实现的。

2.1 Domain 简介

说起 Domain，大家一般都会联想到域名，或者更精确一点的说，联想到一个 FQDN²。但 FreeSWITCH 中的 Domain 是指一个域，或者，在这里，我们说到多租户的时候，可以说它指一个租户。实际上，它就是唯一标志一个域的字符串。当然，由于它可以是任意的字符串，那么，用实际的域名和 IP 地址也是没有问题的。但是，要记住，虽然你可以使用域名或实际的 IP 地址作为 Domain，但并不表示 Domain 跟域名或 IP 地址以及 DNS 有任何关系，FreeSWITCH 也永远不会把 Domain 通过 DNS 解析成域名，反之亦然。

讲到这里，读者可能听得一头雾水，那么，为什么把问题搞这么复杂呢？答案很简单，为了支持多租户。

我们首先来看一下 `conf/vars.xml` 中如下的配置：

```
<X-PRE-PROCESS cmd="set" data="domain=$${local_ip_v4}"/>
<X-PRE-PROCESS cmd="set" data="domain_name=$${domain}"/>
```

其中，上述配置配置了两个全局变量『`domain`』和『`domain_name`』，前者用作一个核心变量——当系统需要一个 Domain 值，而通过当前的环境又找不到这么一个值时，就用该变量的值顶替；而后者用作一个 Dialplan 中的变量，它只是跟每一通电话相关的。当然默认两者的值都等于

¹Multitenancy。参见<http://en.wikipedia.org/wiki/Multitenancy>。

²Fully qualified domain name，译作完全资格域名，或绝对域名。参见http://en.wikipedia.org/wiki/Fully_qualified_domain_name。

『local_ip_v4』的值。而『local_ip_v4』是一个动态计算出的变量值，它一般是当前能上网的网卡的 IP 地址（在取不到的情况下可能为『127.0.0.1』）。通过将『local_ip_v4』的值设成两个变量的默认值是因为这样的话在默认安装后不需要任何配置就可以在所有环境中都正常使用。

为了说明『domain』跟实际的 FQDN 域名没有任何关系，我们来做如下实验。首先，如图2.1所示，在 Domain 中填入『dujinfang.com』。注册时选择『Send outbound via Proxy』，并在 Proxy 中输入 FreeSWITCH 的 IP 地址。在本例中，笔者试验的客户端的服务器的 IP 地址都是『192.168.7.5』。

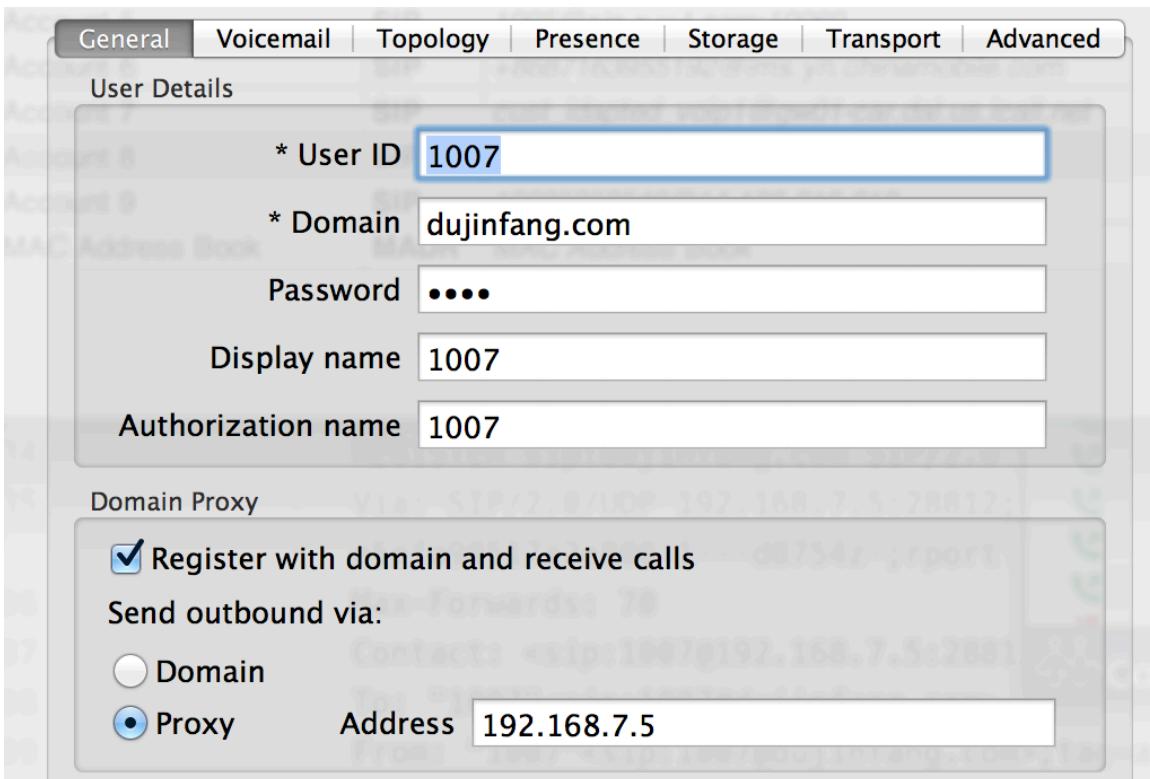


图 2.1: 通过 Proxy 注册

经过这样配置后，我们在 FreeSWITCH 中开启 SIP trace 可以看到如下的注册消息（为了方便讲解我们加了行号）：

```

01 -----
02 recv 789 bytes from udp/[192.168.7.5]:28812 at 15:36:21.277729:
03 -----
04 REGISTER sip:dujinfang.com SIP/2.0
05 Via: SIP/2.0/UDP 192.168.7.5:28812;branch=z9hG4bK-d8754z-...
06 Max-Forwards: 70
07 Contact: <sip:1007@192.168.7.5:28812;rinstance=af02322f2d5c82ba>
08 To: "1007"<sip:1007@dujinfang.com>
09 From: "1007"<sip:1007@dujinfang.com>;tag=a3e90a4e
10 Call-ID: ZTBhNGVkJMTJ10GQzMDZlNjJjN2E2MjQ5NGEzZTZmM2U
11 CSeq: 2 REGISTER

```

```

12 Expires: 3600
13 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO
14 User-Agent: Bria 3 release 3.5.0b stamp 69410
15 Authorization: Digest username="1007",realm="dujinfang.com",
    nonce="40dbf2e3-e5f6-4ee3-badf-855253b12c02",uri="sip:dujinfang.com",
    response="14337381cc0302574b6c39948fbea98f",
    cnonce="0c61ee029710dbf6b740dce279c651fa",
    nc=00000001,qop=auth,algorithm=MD5
16 Content-Length: 0

```

该消息不是第 1 个注册消息，而是在收到 401 响应后，计算出带有 Authorization 头的第 2 位注册消息。其中，我们可以看到，该消息确定发送到了我们在客户端上设置的 SIP Proxy 的 IP 地址上，而与我们以前经常使用的『Domain』中的『dujinfang.com』没有任何关系。在这时，『dujinfang.com』是一个合法的域名，但它没有经过任何形式的在 DNS 中解析到『192.168.7.5』这个 IP 地址上。由此，你可以看到，在此处的客户端上的『Domain』中，你在练习的时候可以填入任何的值，虽然填写不属于你自己的域名（如『apple.com』或『microsoft.com』，以及本例子中属于笔者的域名等）是非常不明智的。

第 4 行称为 Request Line，可以看到『dujinfang.com』也出现在这里，同理，它也出现在第 8 行、第 9 行，以及第 15 行等。

关于 Domain 我们就先介绍到这里，事实上，如果读者没有亲身的实践，在这里笔者是很难讲清楚的。就连 FreeSWITCH 的作者 Anthony Minessale 也花了一小时在邮件列表中讲这个问题，当然，他讲得是非常有参考意义的。参见<http://lists.freeswitch.org/pipermail/freeswitch-users/2013-August/099131.html>。下面我们通过实际的例子来看一下如何通过 Domain 设置多租户，希望读者最终能够理解这里的意思。

2.2 配置与实例

上面讲到的注册是可以成功的，而且我们在 SIP 消息中看到到处都是我们设置的 Domain（『dujinfang.com』）。但是是否注册成功了就说明这支持多租户了呢？答案是否定的。我们还需要做一些工作。首先，我们用以下 Dialplan 测试一下，看『domain』及『domain_name』两个变量能告诉我们什么。

```

<extension name="Domain">
    <condition field="destination_number" expression="^1234$">
        <action application="log" data="INFO domain=${domain}"/>
        <action application="log" data="INFO domain_name=${domain_name}"/>
    </condition>
</extension>

```

通过上述 Dialplan 的设置，在拨打1234后，我们在日志中可以看到如下输出：

```
[INFO] mod_dialplan_xml.c:558 Processing 1007 <1007>->1234 in context default
[INFO] mod_dptools.c:1595 domain=192.168.7.5
[INFO] mod_dptools.c:1595 domain_name=192.168.7.5
```

可以看出，由于『domain』是一个核心的变量，它继续保持『192.168.7.5』这个 IP 地址值倒也罢了，但『domain_name』也不变，我们根据什么来做多租户的路由呢？即，我们根据凭什么说这是某某域里的『1007』发起的呼叫而不是另一个域中的『1007』发起的呢？

其实，这是由于『FreeSWITCH 默认安装要在所有环境下都有运行』的根本宗旨决定的，要达到我们的要求，需要检查 Profile (internal) 中的如下参数：

```
<param name="force-register-domain" value="$$\{domain\}" />
<param name="force-subscription-domain" value="$$\{domain\}" />
<param name="force-register-db-domain" value="$$\{domain\}" />
```

可以看出，在默认的配置中，上述参数都是生效的，也就是说，不管你的 Domain 怎么配，都让它『force』（强制）成了默认的 Domain。因此，在此，我们需要删除这些参数，或者，在配置文件中注释掉这些行。

然后，重启该 Profile，就会看到如下的警告，而且，我们的 SIP 客户端再也注册不上了。

```
[WARNING] sofia_reg.c:2679 Can't find user [1007@dujinfang.com] from 192.168.7.5
You must define a domain called 'dujinfang.com' in your directory
and add a user with the id="1007" attribute
and you must configure your device to use the proper domain
in it's authentication credentials.
```

虽然是个警告，但是个好的警告，它至少告诉我们它在配置文件中找不到这个 Domain 和这个用户了。我们看一看默认的用户目录配置文件conf/directory/default.xml，它里面定义了一个 Domain，是默认的『\$\$\{domain\}』值，如下：

```
<include>
  <!--the domain or ip (the right hand side of the @ in the addr-->
  <domain name="$$\{domain\}">
    <params>
      ....
```

找到它后，我们只需要照着他复制一份新的就好了（这里当然也可以把『\${domain}』改成我们想要的『dujinfang.com』，但我们除此之外还要加别的 Domain，因而到后面总是要复制的）。

将上述文件复制到 conf/directory/dujinfang.com.xml，然后修改其中的『domain_name』为如下的值：

```
<domain name="dujinfang.com">
```

在 FreeSWITCH 控制台中执行『reloadxml』使之生效，然后，重新注册客户端，就发现能注册成功了。通过如下的命令也能列出注册信息：

```
freeswitch> sofia_contact 1007@dujinfang.com  
  
sofia/internal/sip:1007@192.168.7.5:34088;rinstance=d3ba18cdf1903f17  
  
freeswitch> sofia status profile internal reg 1007
```

Registrations:

```
=====  
Call-ID: ZWRhZjA3NjIONjYxODA1MDdkOGI0YzYxZWVkJGRmZjA  
User: 1007@dujinfang.com  
Contact: "1007" <sip:1007@192.168.7.5:34088;rinstance=d3ba18cdf1903f17>  
Agent: Bria 3 release 3.5.0b stamp 69410  
Status: Registered(UDP)(unknown) EXP(2013-10-20 00:56:40) EXPSECS(94)  
Host: seven.local  
IP: 192.168.7.5  
Port: 34088  
Auth-User: 1007  
Auth-Realm: dujinfang.com  
MWI-Account: 1007@dujinfang.com
```

接着，拨打『1234』进行测试，就可以看到如下的输出了：

```
[INFO] mod_dialplan_xml.c:558 Processing 1007 <1007>->1234 in context default  
[INFO] mod_dptools.c:1595 domain=192.168.7.5  
[INFO] mod_dptools.c:1595 domain_name=dujinfang.com
```

从中可以看出，这里的『domain_name』变量也变成我们期望的值了。

2.3 进阶

当然，我们新建的『dujinfang.com.xml』与原来的『default.xml』都装入了『conf/directory/default』目录下的用户配置文件。既然是不同的租户，我们就需要把它们分开，因此我们也把整个『default』目录复制一份，放到到新的目录『dujinfang.com』中，并且将『dujinfang.com.xml』中的『include』一行改为：

```
<users>
<X-PRE-PROCESS cmd="include" data="dujinfang.com/*.xml"/>
</users>
```

还没有完，我们要让我们自己域中的用户使用单独的 Dialplan 进行路由，因此，进入 dujinfang.com 目录，将里面所有的用户配置文件（如 1007.xml）中的『user_context』参数全部改成如下的值：

```
<variable name="user_context" value="dujinfang.com"/>
```

『reloadxml』后再拨打『1234』发现打不通了，原因很显然，我们还没有在 Dialplan 中设置『dujinfang.com』这个 Context。当然，设置它也很简单。进入『conf/dialplan』目录，将『default.xml』复制为『dujinfang.com.xml』，并修改里面的 Context 的名称为我们需要的值，如：

```
<include>
<context name="dujinfang.com">
```

再一次执行『reloadxml』并拨打测试，就可以看到，我们这里的路由也跟默认的配置完全分离了。这就基本上达到了我们的效果。读者接下来可以参照这里的步骤再配置几个新的域以对比测试。

2.4 其它

使用多 Domain (及多 Directory、多 Dialplan) 支持多租户以后，有以下几个事情是需要注意的：

- 在 Dialplan 中的呼叫字符串不能再使用类似『user/1000』这样的缩写形式了，而要写成如『user/1000@\${domain_name}』的形式。

- 在使用会议、`fifo`等类的应用时，也注意不要像本书中的例子一样图省事（当然，也不完全是为了省事，有时候是因为太长的行在书中不容易排版）把会议的名称写成『3000』或把`fifo`的名称写成『book』这样的短名字了，一定要写成如『3000-\$`{domain_name}`』或『book@\$`{domain_name}`』之类带 Domain 的长名字。
- 在实际应用中 Domain 可以与实际的 FQDN 相同，最好能使用 DNS 的 SRV 记录进行解析。不同的机构可以使用不同的域，如『dujinfang.com』、『microsoft.com』等；同一个机构不同的分支机构间也可以使用不同的域，如『beijing.dujinfang.com』、『shanghai.dujinfang.com』等。
- 如果需要计费的话，话单等也是尤其需要注意的。
- 除此之外，还可以进一步细分，将不同的租户分到不同的 Sofia Profile 中去。即每个租户都有自己的 Profile（分别占用不同的端口号），它们的 Profile 中的信息及各种参数都可以不同。当然，在所有的租户都想使用5060这样的端口号的话，也可以使用 DNS 的 SRV 记录来解决。关于 SRV 记录的用法超出了本书的范围，读者如果需要的话可以自行研究。

第三章 FreeSWITCH 模块

FreeSWITCH 的模块有很多，它们丰富了 FreeSWITCH 的功能。实际上，常用的模块本书已经基本都介绍过了。但总有好奇的读者会问『不知道mod_xxx』这个模块是干什么用的。因此，本章列出大部分已知的的模块，供广大读者参考。有的模块比较有代表性，我们也会详细介绍一下。

3.1 模块列表

关于模块的介绍，在官方的 Wiki 页面上大部分也都能查到。<https://freeswitch.org/confluence/display/FREESWITCH/Modules>，大部分模块也都有自己的页面，它们的 URL 也比较规范，如https://freeswitch.org/confluence/display/FREESWITCH/mod_dptools、https://freeswitch.org/confluence/display/FREESWITCH/mod_commands等。

我们基于 FreeSWITCH 1.6 版，以自然的顺序来讲解。这里，模块的顺序是用下面命令生成的：

```
cd src/mod
find . -type d -name mod_*
```

各个模块根据其主要功能和能提供的接口分到不同的目录中。

3.1.1 applications

该目录下的模块提供了大部分的应用功能，有的模块实现了多种 Interface 不好归类也会存在该目录中。

- **mod_abstraction**: 用于创建新的 API 命令。新的命令可以基于原有的 API 创建，相当于创建原有命令的别名或快捷方式。
- **mod_av**: 基于 libav.org 库的视频模块。支持 H263、H264 编解码，mp4 视频播放和录像，RTMP 推送等。也支持ffmpeg库（2.6、2.8、3.0）。

- **mod_avmd**: avmd 是 Advanced Voice Mail Detection 的缩写，即高级语音邮件检测。它是**mod_vmd**的高级版。详见**mod_vmd**。
- **mod_bert**: 一个用于音频测试的模块。
- **mod_blacklist**: 黑名单功能。它提供了一些通过 Dialplan 来添加、删除以及检查黑名单的方法。
- **mod_callcenter**: 一个比较强大的呼叫中心类应用。
- **mod_cidlookup**: 用于主叫号码查询，即可以根据主叫号码从本地数据库或网络上查询到主叫的名字。网络上有开放的服务，如：

```
$ curl https://api.opencnam.com/v2/phone/16502530000?format=pbx
GOOGLE INC
```

- **mod_cluechoo**: 该模块是一个例子模块，主要是教大家怎么写模块。另外，它还带了一个好玩的例子，如在命令行上执行 cluechoo 将会看到屏幕上开过一个小火车。
- **mod_commands**: 提供了系统大部分的命令 API。
- **mod_conference**: 多人语音及视频会议。
- **mod_curl**: 使用**libcurl**作为一个 HTTP 客户端向 Web 服务器发送请求，也可以得到返回的结果。
- **mod_cv**: 使用 OpenCV 库，对视频进行图像识别的模块。
- **mod_db**: 该模块提供一组接口，用于使用 API 或 App 对数据库表进行增、删、改、查。
- **mod_directory**: 该模块用于按姓名呼叫用户。如果不知道用户的分机号，但知道用户名，则可以通过输入该用户名的前几位进行拨叫。如，在下面的例子中，我们可以通过拨打 411 进入 directory 程序。

```
<extension name="directory" continue="true">
    <condition field="destination_number" expression="^411$">
        <action application="directory" data="default $$\{domain\} default"/>
    </condition>
</extension>
```

当系统提示输入名字时，我们输入9378可以找到 Brian West（默认是按**last_name**查找的，West 对应键盘按键9378）。

- **mod_distributor**: 如果需要通过多个网关出局时，该模块可以帮助将呼叫根据一定的比例分发到不同的网关。
- **mod_dptools**: 提供了系统大部分的 App。
- **mod_easyroute**: 用于根据号码路由，对大规模的 DID 比较有用。

- **mod_enum**: 通过 ENUM 查询可以根据 E164 号码找到用户的 SIP 地址。该模块也提供一个 enum Dialplan。
- **mod_esf**: ESF 是 Extended SIP Functionality 的缩写，即扩展的 SIP 功能。它提供通过 Multicast 方法进行组拨的功能。
- **mod_esl**: 该模块用于两个 FreeSWITCH 间的 ESL 对接。即一个 FreeSWITCH 可以作为另一个 FreeSWITCH 的 ESL 客户端访问它。
- **mod_expr**: 提供expr表达式计算。
- **mod_fifo**: 一个先入先出队列 (First In First Out)，可以用于简单的呼叫中心排队。
- **mod_fsk**: 收发 FSK (Frequency-shift keying, 移频键控) 信息。
- **mod_fsv**: FSV 是 FreeSWITCH Video 的缩写，它使用了一种私有的格式来进行视频录像，可以支持任何编码的视频格式。它只存储原始的 RTP 包，对视频流不进行任何处理。
- **mod_hash**: 用于操作系统内部的哈希表。可以存储一些简单数据。
- **mod_httapi**: 一种 HTTP 格式的 API 接口，可以通过 HTTP 方式写 IVR。
- **mod_http_cache**: 通过 http 方式上传和下载文件，并可以进行本地缓存。
- **mod_ladspa**: 使用 ladspa 库对声音进行处理，可以让声音更好听。
- **mod_lcr**: LCR 是 Least Cost Routing 的缩写，即最省钱的路由。它会根据数据库配置的路由信息和费率找到最省钱的路由。
- **mod_limit**: 用于系统资源限制。
- **mod_memcache**: 与 Memcache 交互，类似把**mod_hash**的数据库到远程的 Memcache 中。
- **mod_mongo**: 与 Mongodb 交互，类似**mod_memcache**。
- **mod_mp4**: 提供 MP4 文件的播放支持。
- **mod_mp4v2**: 提供 MP4 文件的录像支持。
- **mod_nibblebill**: 一些简单的计费功能，可用于预付费和电话卡类的应用。
- **mod_oreka**: 使用 oreka 进行录音。oreka 是一款开源的录音软件。
- **mod_osp**: 通过 Open Settlement Protocol 查找路由或上报 CDR。
- **mod_rad_auth**: 使用 radius 服务器进行鉴权。
- **mod_random**: 通过访问/dev/hwrandom 设备影响随机数的熵。
- **mod_redis**: 与 redis 服务器交互，类似**mod_memcache**。
- **mod_rss**: 访问 RSS (Really Simple Syndication, 简易信息聚合) 数据。
- **mod_skel**: 一个模块的例子框架。
- **mod_sms**: 处理文本消息。如收发 SIP MESSAGE 消息等。它实现了消息路由 (Chazplan)，类似 Dialplan。
- **mod_snapshot**: 可以截取一段声音的快照。
- **mod_snipe_hunt**: 一个简单的例子模块。
- **mod_snom**: 用于 snom 话机的一些特性。
- **mod_sonar**: 该模块类似于一个真正的声纳。首先你可以在远端启动一个服务器，能对来话执行echo App。然后在本地的 FreeSWITCH 上产生一些铃音，发送到远端的服务器上再反射

回来，然后使用 VAD 检测功能可以检测这些铃音，从而可以在某种程度上确定网络的质量。

- **mod_soundtouch**: 使用 soundtouch 库对声音进行处理，可以增加音效。
- **mod_spandsp**: 使用 spandsp¹支持一些语音编码及传真功能。

- **mod_spy**: 用于监视某个话机，当该话机有通话时，本机就振铃并可以监听。
- **mod_stress**: 使用快速傅立叶变换 (FFT, Fast Fourier Transform) 检测重音。
- **mod_translate**: 通过既定的规则对号码进行翻译。

- **mod_valet_parking**: 电话停靠。类似于泊车，有来电时可以将来电依靠在某个泊位上，然后通知某人拨开指定的号码将来话『接』走。

- **mod_vmd**: 提供 Voicemail 声音检测。在国外，好多电话都有自动应答功能，如『您好，主人不在家，请留言』。使用该模块可检测到这种声音，应用程序在自动外呼时就可以根据它的结果判断是人工接听的还是机器接听的。

- **mod_voicemail**: 语音邮箱。

- **mod_voicemail_ivr**: 带 IVR 导航的语音邮箱。

3.1.2 asr_tts

提供自动语音识别及语音合成的功能。

- **mod_cepstral**: 使用 Cepstra 语音库支持 TTS。
- **mod_flite**: 使用 Festival Lite 库支持 TTS。
- **mod_pocketsphinx**: 使用 pocketsphinx 库支持语音识别。
- ***mod_tts_commandline**: 通过命令行程序使用 TTS。
- **mod_unimrcp**: 通过 uniMRCP 协议与其他 ASR/TTS 产品对接。uniMRCP 是一个标准的协议，很多语音产品都支持它。

3.1.3 codecs:

各种类型的音、视频编码。大多数名称都很直观，不再多做解释。

- **mod_amr**: AMR 编解码，仅支持透传。
- **mod_amrwb**: AMR 编解码，宽带，仅支持透传。
- **mod_b64**: Base64 编码，可以传输任何数据。
- **mod_bv**: BroadVoice 的编码。

¹<http://www.soft-switch.org/>

- **mod_celt**: CELT 编解码。
- **mod_codec2**: Codec2 编解码，非常节省带宽，仅 2-4k。
- **mod_com_g729**: 商业的 G.729 编码，可转码，需要许可证。
- **mod_dahdi_codec**: 通过 DAHDI 库提供的编码。
- **mod_g723_1**: G723 编解码，仅支持透传。
- **mod_g729**: 仅支持透传。
- **mod_h26x**: 提供 H261, H263, H264 等视频编码，仅支持透传
- **mod_ilbc**: iLBC 编解码。
- **mod_isac**: iSac 编解码。
- **mod_mp4v**: mp4v 编解码，仅支持透传。
- **mod_openh264**: 使用思科 OpenH264 库支持 H264 编解码。
- **mod_opus**: OPUS 编解码。
- **mod_sangoma_codec**: 通过硬件板卡支持包括 G729、iLBC 等多种编码
- **mod_silk**: SILK 编解码。
- **mod_siren**: Siren 编解码。
- **mod_skel_codec**: 例子模块。
- **mod_speex**: SpeeX 编解码模块。在 1.4 版中已移至核心中，不再以独立模块的形式存在
- **mod_theora**: Theora 编解码。
- **mod_vp8**: 在 1.6 中已被**mod_vpx**代替。
- **mod_vpx**: VP8、VP9 编解码模块。

3.1.4 dialplans

拨号计划。

- **mod_dialplan_asterisk**: 类似于 Asterisk 格式的拨号计划。
- **mod_dialplan_directory**: 通过 LDAP 查询拨号计划。
- **mod_dialplan_xml**: XML 拨号计划。

3.1.5 directories

目录服务。

- **mod_ldap**: 通过 LDAP 提供目录服务。

3.1.6 endpoints

各种 Endpoint 的实现。

- **mod_alsa**: 使用 ALSA 声卡。
- **mod_dingaling**: 连接 Google Talk。
- **mod_gsmopen**: 使用无线上网卡上的 GSM 接口或使用手机上的 GSM 接口与外界发短信或通话。
- **mod_h323**: 连接 H.323 设备。使用 OpenH323 库实现。
- **mod_khomp**: 使用 KHOMP 板卡。
- **mod_loopback**: 提供 loopback 回环接口。
- **mod_opal**: 连接 H.323 设备，使用 OPAL 库实现。
- **mod_portaudio**: 通过 Portaudio 库支持本地声卡。
- **mod_reference**: 未知。
- **mod_rtmp**: 通过 Adobe 的 rtmp 协议与浏览器中的 Flash 电话进行通话。
- **mod_skinny**: 支持思科的 SCCP 协议话机。
- **mod_skypopen**: 与 Skype 互通。
- **mod_sofia**: SIP 模块。
- **mod_unicall**: 未知。
- **mod_verto**: FreeSWITCH 特定的基于 WebSocket 和 JSON 的协议，支持 WebRTC。

3.1.7 event_handlers

事件处理。

- **mod_amqp**: AMQP 模块，支持与 RabbitMQ 连接，发送事件，执行命令等。
- **mod_cdr_csv**: CSV 格式的话单。
- **mod_cdr_mongodb**: 将话单写入 Mongodb。
- **mod_cdr_pg_csv**: 将话单写入 PostgreSQL 数据库。
- **mod_cdr_sqlite**: 将话单写入 SQLite 数据库。
- **mod_erlang_event**: 对接 Erlang 节点，提供事件、日志、命令接口等。
- **mod_event_multicast**: 将事件通过组播方式发出去。
- **mod_event_socket**: 通过 ESL 库与第三方的接口。
- **mod_event_test**: 测试。
- **mod_event_zmq**: 使用 ZeroMQ 协议与第三方对接。
- **mod_format_cdr**: JSON 和 XML 格式的 CDR。
- **mod_json_cdr**: JSON 格式的 CDR。
- **mod_kazoo**: 2600Hz 使用的 Erlang 模块，解决了**mod_erlang_event**的一些问题。
- **mod_radius_cdr**: 将 CDR 写入 Radius 服务器。
- **mod_rayo**: Rayo 支持。
- **mod_smpp**: 使用 SMPP 协议发送即时消息。

- **mod_snmp**: SNMP 网管接口。

3.1.8 formats

格式。

- **mod_imagick**: 使用 ImageMagick 库支持 Gif、PDF 等。
- **mod_local_stream**: 从本地文件生成媒体流。
- **mod_native_file**: 支持原生文件读写，如直接读写.PCMU或.G729格式的文件。
- **mod_png**: 支持 PNG 文件。
- **mod_portaudio_stream**: 使用portaudio库从本地声卡生成媒体流。
- **mod_shell_stream**: 从 Shell 命令中生成媒体流。
- **mod_shout**: MP3 文件格式支持，远程 Shoutcast 服务器支持。
- **mod_sndfile**: 使用libsndfile支持大多数的声音文件。
- **mod_ssml**: SSML 格式的文件支持。
- **mod_tone_stream**: 生成铃流音。
- **mod_vlc**: 使用 libvlc 提供媒体文件格式的支持。
- **mod_webm**: 支持 webm 文件。

3.1.9 languages

各种嵌入式编码语言接口。

- **mod_basic**: Basic。
- **mod_java**: Java。
- **mod_lua**: Lua。
- **mod_managed**: 微软平台的语言接品，如 C#、VB.NET 等。
- **mod_perl**: Perl。
- **mod_python**: Python。
- **mod_spidermonkey**: Javascript，已被mod_v8替代。
- **mod_v8**: Javascript。
- **mod_yaml**: Yaml。

3.1.10 loggers

日志。

- **mod_console**: 控制台日志。

- **mod_graylog2**: 使用 graylog2 写日志文件。
- **mod_logfile**: 日志文件。
- **mod_syslog**: 将日志写到 Syslog。

3.1.11 say

多语种接口。

- **mod_say_de**: 德语。
- **mod_say_en**: 英语。
- **mod_say_es**: 西班牙语。
- **mod_say_es_ar**: 西班牙语。
- **mod_say_fa**: 波斯语。
- **mod_say_fr**: 法语。
- **mod_say_he**: 希伯来语。
- **mod_say_hr**: 克罗地亚语。
- **mod_say_hu**: 匈牙利语。
- **mod_say_it**: 意大利语。
- **mod_say_ja**: 日语。
- **mod_say_nl**: 荷兰语。
- **mod_say_pl**: 波兰语。
- **mod_say_pt**: 葡萄牙语。
- **mod_say_ru**: 俄语。
- **mod_say_sv**: 萨尔瓦多语。
- **mod_say_th**: 泰语。
- **mod_say_zh**: 汉语。

3.1.12 timers

定时器。

- **mod_posix_timer**: Posix 定时器。
- **mod_timerfd**: 使用 Linux 内核中的timerfd定时器。

3.1.13 xml_int

XML 接口。

- **mod_xml_cdr**: 使用 XML 格式写 CDR。
- **mod_xml_curl**: 从远程 HTTP 服务器获取 XML 配置。
- **mod_xml_ldap**: 从远程 LDAP 服务器获取 XML 配置。
- **mod_xml_radius**: 从远程 Radius 服务器获取 XML 配置。
- **mod_xml_rpc**: 使用 XMLRPC 接口与第三方交互，提供命令、日志及事件接口等，本身也是一个简单的 Web 服务器，并提供一个简单的 Web 管理界面。
- **mod_xml_scgi**: 使用 SCGI 协议获取 XML 配置。

3.2 mod_sofia

mod_sofia 是 FreeSWITCH 中最重量级的模块，不管是代码量还是功能上都很丰富。关于该模块我们已经在《FreeSWITCH 权威指南》上详细介绍过了，在此，我们仅来看一下 Sofia Profile 中的配置参数。

Sofia 的 Profile 有很多参数，下面是一些简要说明，供参考。其中，有的选项是取布尔值的，则一般来说**true/yes**通用，**false/no**通用。

- **media-option**: 媒体选项。该选项有两个取值。

resume-media-on-hold: 如果 FreeSWITCH 是没有媒体（No Media/Bypass Media）的，那么如果设置了该参数，当你在话机上按下 hold 键时，FreeSWITCH 将会回到有媒体的状态。如：

bypass-media-after-att-xfer: Attended Transfer 译即出席转移，也称协商转，它需要媒体才能完成工作。但如果在执行**att-xfer**之前没有媒体，该参数能让**att-xfer** 执行时有通过 **re-INVITE** 请求要回媒体，等到转移结束后再回到 Bypass Media 状态。

- **user-agent-string**: 该参数设置 SIP 消息中显示的 User-Agent 字段。如：

```
<param name="user-agent-string" value="FreeSWITCH Rocks!" />
```

- **debug**: 设置是否启用调试。取值有 0 和 1，如果是 1 则会输出更多的调试信息。如：

```
<param name="debug" value="1" />
```

- **shutdown-on-fail**: 由于各种原因（如端口被占用，IP 地址错误等），都可能造成 UA 在初始化时失败，该参数在失败时会停止 FreeSWITCH。如：

```
<param name="shutdown-on-fail" value="true"/>
```

- **sip-trace**: 是否开启 SIP 消息跟踪。如：

```
<param name="sip-trace" value="no"/>
```

另外，也可以在控制台上用以下命令开启和关闭 SIP 跟踪，如：

```
sofia profile internal siptrace on  
sofia profile internal siptrace off
```

- **log-auth-failures**: 是否将认证错误写入日志。

```
<param name="log-auth-failures" value="true"/>
```

- **context**: 设置来话到达 Dialplan 的 Context，注意，如果用户鉴权通过，则用户目录中的 user_context 比该参数优先级要高。如：

```
<param name="context" value="public"/>
```

- **rfc2833-pt**: 设置 SDP 中 RFC2833 的 Payload 值。如：

```
<param name="rfc2833-pt" value="101"/>
```

- **sip-port**: 设置监听的 SIP 端口号。如：

```
<param name="sip-port" value="5060"/>
```

- **ws-binding**: 设置 WebSocket 的监听地址和端口号（用于 SIP over WebSocket，一般是 WebRTC 呼叫）。如：

```
<param name="ws-binding" value=":5066"/>
```

- **wss-binding**: 设置安全 WebSocket 监听地址和端口号。该选择需要相关的安全证书 (`wss.pem` 存放在 `/usr/local/freeswitch/certs` 目录下)。如:

```
<param name="wss-binding" value=":7443"/>
```

- **dialplan**: 设置 Dialplan 的类型。如:

```
<param name="dialplan" value="XML"/>
```

- **dtmf-duration**: 设置 DTMF 的时长。如:

```
<param name="dtmf-duration" value="2000"/>
```

- **inbound-codec-prefs**: 支持的来话语音编码，用于语音编码协商。如:

```
<param name="inbound-codec-prefs" value="PCMU,PCMA,H264"/>
```

- **outbound-codec-prefs**: 支持的去话语音编码。用于语音编码协商。如:

```
<param name="outbound-codec-prefs" value="$$\{global_codec_prefs\}"/>
```

- **rtp-timer-name**: RTP 定时器名称，其他可先的定时器可以在 FreeSWITCH 中用『`show timers`』命令得到。

```
<param name="rtp-timer-name" value="soft"/>
```

- **rtp-ip**: RTP 使用的地址。如:

```
<param name="rtp-ip" value="$$\{local_ip_v4\}">
```

- **sip-ip**: SIP 监听的 IP 地址。如：

```
<param name="sip-ip" value="$$\{local_ip_v4\}">
```

- **hold-music**: UA 进入 hold 状态时默认播放的音乐。如：

```
<param name="hold-music" value="$$\{hold_music\}">
```

- **apply-nat-acl**: 用于判断哪些 IP 地址涉及到 NAT。如：

```
<param name="apply-nat-acl" value="nat.auto">
```

- **extended-info-parsing**: 是否启用扩展 INFO 解析支持，扩展 INFO 支持可用于向 FreeSWITCH 发送事件、API 命令等。如：

```
<param name="extended-info-parsing" value="true">
```

- **aggressive-nat-detection**: 用于 NAT 穿越，检测 SIP 消息中的 IP 地址与实际的 IP 地址是否相符，详见 9.4 节。

```
<param name="aggressive-nat-detection" value="true">
```

- **enable-100rel**: 设置是否使用 PRACK 对 SIP 183 消息进行证实。如：

```
<param name="enable-100rel" value="true">
```

- **enable-compact-headers**: 是否压缩 SIP 头。压缩 SIP 头域能使用 SIP 包变小一些。如：

```
<param name="enable-compact-headers" value="true"/>
```

- **enable-timer**: 是否启用时钟。默认是启用的。启用时钟后，在指定的时间内（如 20ms）如果收不到 RTP 数据，则返回静音（CNG）数据。如果不启用该功能，则会一直等待直到收到数据。如：

```
<param name="enable-timer" value="true"/>
```

- **minimum-session-expires**: SIP 会话超时最小值，在 SIP 消息中设置 Min-SE。如：

```
<param name="minimum-session-expires" value="120"/>
```

- **apply-inbound-acl**: 对来话启用哪个 ACL 进行鉴权。如：

```
<param name="apply-inbound-acl" value="domains"/>
```

- **local-network-acl**: 默认情况下，FreeSWITCH 会自动检测本地网络，并创建一条 localnet.auto ACL 规则。在 NAT 穿越时有用。也可以手工指定其它的 ACL。如：

```
<param name="local-network-acl" value="localnet.auto"/>
```

- **apply-register-acl**: 对注册请求采用哪个 ACL 进行鉴权。如：

```
<param name="apply-register-acl" value="domains"/>
```

- **dtmf-type**: DTMF 收号的类型。有三种方式，info、inband、rfc2833。如：
- **send-message-query-on-register**: 如何发送 message-waiting 消息。true 是每次都发送，而 first-only 只是首次注册时发送。如：

```
<param name="send-message-query-on-register" value="true"/>
```

- **caller-id-type**: 设置主叫号码显示的类型, rpid将会在 SIP 消息中设置Remote-Party-ID, 而pid则会设置P-*-Identity, 如果不需要这些, 可以设置成none。如:

```
<param name="caller-id-type" value="rpid"/>
```

- **record-path**: 录音文件的默认存放路径。如:

```
<param name="record-path" value="$$\{recordings_dir\}"/>
```

- **record-template**: 录音文件名模板。如:

```
<param name="record-template"
value="\$\{caller_id_number\}.\$\{target_domain\}.\$\{strftime(%Y-%m-%d-%H-%M-%S)\}.wav"/>
```

- **manage-presence**: 是否支持列席 (Presence)。如果不的话可以关掉以节省资源。如:

```
<param name="manage-presence" value="true"/>
```

- **manage-shared-appearance**: 是否支持 SLA (Shared Line Appearance)。如:

```
<param name="manage-shared-appearance" value="true"/>
```

与此相关的还有两个参数, 分别指定 Presence 的数据库名称及域, 如:

```
<param name="dbname" value="share_presence"/>
<param name="presence-hosts" value="$$\{domain\}"/>
```

- **bitpacking**: 设置 G726 的 bitpacking。如:

```
<param name="bitpacking" value="aal2"/>
```

- **max-proceeding**: 最大的开放对话 (SIP Dialog) 数。如:

```
<param name="max-proceeding" value="1000"/>
```

- **session-timeout**: 会话超时时间。

```
<param name="session-timeout" value="120"/>
```

- **multiple-registrations**:

是否支持多点注册，取值可以是**contact**或**true**。开启多点注册后多个 UA 可以用同一个分机注册上来，有人呼叫该分机时所有 UA 都会振铃。

```
<param name="multiple-registrations" value="contact"/>
```

- **inbound-codec-negotiation**: SDP 中的语音编协商，如果设成**greedy**，则自己提供的语音编码列表会有优先权。如:

```
<param name="inbound-codec-negotiation" value="generous"/>
```

- **bind-params**: 该参数设置的值会附加在 Contact 地址上。如:

```
<param name="bind-params" value="transport=udp"/>
```

- **unregister-on-options-fail**: 是否在 Ping 失败后取消分机注册。为了 NAT 穿越或支持 Keep Alive，FreeSWITCH 会通过 NAT 方式注册到它的分机 (**nat-options-ping**) 或所有注册到它的分机 (**all-reg-options-ping**) 周期性地发一些 OPTIONS 包，相当于 ping 功能。如:

```
<param name="unregister-on-options-fail" value="true"/>
<param name="nat-options-ping" value="true"/>
<!-- <param name="all-reg-options-ping" value="true"/> -->
```

- **tls**: 是否支持 TLS, 默认否。如:

```
<param name="tls" value="true"/>
```

- **tls-bind-params**: 设置 TLS 的其它绑定参数。如:

```
<param name="tls-bind-params" value="transport=tls"/>
```

- **tls-sip-port**: TLS 的监听端口号。如:

```
<param name="tls-sip-port" value="5061"/>
```

- **tls-cert-dir**: 存放 TLS 证书的目录。如:

```
<param name="tls-cert-dir" value="$$internal_ssl_dir"/>
```

- **sip_tls_versio**: 使用的 TLS 版本, 有`sslv23`(默认) 或`tlsv1`两种。如:

```
<param name="tls-version" value="sslv23"/>
```

- **rtp-autoflush-during-bridge**: 该选项默认为 true。即在桥接电话时是否自动清空缓存中的媒体数据(如果套接字上已有数据时, 它会忽略定时器睡眠, 能有效减少延迟)。如:

```
<param name="rtp-autoflush-during-bridge" value="false"/>
```

- **rtp-rewrite-timestamps**: 是否重写或透传 RTP 时间戳。如果透传, FreeSWITCH 有时会产生不连续的时间戳, 有的设备对此可能比较敏感, 该选项可以让 FreeSWITCH 产生自己的时间戳。如:

```
<param name="rtp-rewrite-timestamps" value="true"/>
```

- **pass-rfc2833**: 是否透传 RFC2833 DTMF 包。如：

```
<param name="pass-rfc2833" value="true"/>
```

- **odbc-dsn**: 使用 ODBC 数据库代替默认的 SQLite。如：

```
<param name="odbc-dsn" value="dsn:user:pass"/>
```

- **inbound-bypass-media**: 将所有来电设置为媒体绕过模式，即媒体流 (RTP) 不经过 FreeSWITCH。如：

```
<param name="inbound-bypass-media" value="true"/>
```

- **inbound-proxy-media**: 将所有来电设置为媒体透传。媒体经过 FreeSWITCH 但 FreeSWITCH 不处理，直接转发。如：

```
<param name="inbound-proxy-media" value="true"/>
```

- **inbound-late-negotiation**: 是否开启晚协商。默认情况下 FreeSWITCH 对来话会先协商媒体编码，然后再进入 Dialplan。开启晚协商有助于在协商媒体编码之前，先前电话送到 Dialplan，因而在 Dialplan 中可以进行个性化的媒体协商。

```
<param name="inbound-late-negotiation" value="true"/>
```

- **accept-blind-reg**: 该选项允许任何电话注册，而不检查用户和密码及其他设置。如：

```
<param name="accept-blind-reg" value="true"/>
```

- **accept-blind-auth**: 与上一条类似，该选项允许任何呼叫通过认证。如：

```
<param name="accept-blind-auth" value="true"/>
```

- **suppress-cng**: 抑制 CNG，即不使用静音包。如：

```
<param name="suppress-cng" value="true"/>
```

- **nonce-ttl**: 设置 SIP 认证中 nonce 的生存时间（秒）。如：

```
<param name="nonce-ttl" value="60"/>
```

- **disable-transcodin**: 禁止转码，如果该项为 true 则在 bridge 其他电话时，只提供与 a-leg 兼容或相同的语音编码列表进行协商，以避免引起转码。

```
<param name="disable-transcoding" value="true"/>
```

- **manual-redirect**: 允许在 Dialplan 中进行人工重定向。如：

```
<param name="manual-redirect" value="true"/>
```

- **disable-transfer**: 禁止转移。如：

```
<param name="disable-transfer" value="true"/>
```

- **disable-register**: 禁止注册。如：

```
<param name="disable-register" value="true"/>
```

- **NDLB-broken-auth-hash**: 有一些电话对 Chanllenge ACK 的回复在哈希值里会有 INVITE 方法，该选项容忍这种行为。如：

```
<param name="NDLB-broken-auth-hash" value="true"/>
```

```
<!-- add a ;received="<ip>:<port>\" to the contact when replying to register for nat handling -->
```

- **NDLB-received-in-nat-reg-contact**: 为支持某些 NAT 穿越，在 Contact 头域中增加;received=":"字符串。如：

```
<param name="NDLB-received-in-nat-reg-contact" value="true"/>
```

- **auth-calls**: 是否对来电进行鉴权。如：

```
<param name="auth-calls" value="true"/>
```

- **inbound-reg-force-matching-username**: 强制注册用户与 SIP 认证用户必须相同。如：

```
<param name="inbound-reg-force-matching-username" value="true"/>
```

- **auth-all-packets**: 对所有的 SIP 消息都进行鉴权，而不是仅仅是针对 INVITE 和 REGISTER 消息。如：

```
<param name="auth-all-packets" value="false"/>
```

- **ext-rtp-ip**: 在 NAT 环境中，设置外网 RTP IP。该设置会影响 SDP 中的 IP 地址。有以下几种可能：

- 一个 IP 地址，如 12.34.56.78
- 一个 stun 服务器，它会使用 stun 协议获得公网 IP，如 stun:stun.server.com
- 一个 DNS 名称，如 host:host.server.com
- auto，它会自动检测 IP 地址
- auto-nat，如果路由器支持 NAT-PMP 或 uPnP，则可以使用这些协议获取公网 IP。

如：

```
<param name="ext-rtp-ip" value="auto-nat"/>
```

- **ext-sip-ip**: 与上一条类似，设置外网的 SIP IP。如：

```
<param name="ext-sip-ip" value="auto-nat"/>
```

- **rtp-timeout-sec**: 设置 RTP 超时值（秒）。指定的时间内 RTP 没有数据收到，则挂机。如：

```
<param name="rtp-timeout-sec" value="300"/>
```

- **rtp-hold-timeout-sec**: RTP 处于保持状态的最大时长（秒）。如：

```
<param name="rtp-hold-timeout-sec" value="1800"/>
```

- **vad**: 语音活动状态检测，有三种可能，可设为入（in）、出（out），或双向（both），通常来说out是一个比较好的选择。如：

```
<param name="vad" value="out"/>
```

- **alias**: 给 Sip Profile 设置别名。如：

```
<param name="alias" value="sip:10.0.1.251:5555"/>
```

- **force-register-domain**: 对所有用户都强制使用某一域（Domain）。如：

```
<param name="force-register-domain" value="$$\{domain\}"/>
```

- **force-subscription-domain**: 对所有订阅都强制使用某一域（Domain）。如：

```
<param name="force-subscription-domain" value="$$\{domain\}"/>
```

- **force-register-db-domain**: 对所有经过认证的用户都使用该域（Domain）存入数据库。如：

```
<param name="force-register-db-domain" value="$$\{domain\}" />
```

- **force-subscription-expires**: 强制一个比较短的订阅超时时间。如：

```
<param name="force-subscription-expires" value="60" />
```

- **enable-3pcc**: 是否支持 3PCC 呼叫。该选项有两个值，`true`或`proxy`。`true`则直接接受 3PCC 来电；如果选`Proxy`，则会一直等待电话应答后才回送接受。

```
<param name="enable-3pcc" value="true" />
```

- **NDLB-force-rport**: 在 NAT 时强制 rport。除非你很了解该参数，否则后果自负。如：

```
<param name="NDLB-force-rport" value="true" />
```

- **challenge-realm**: 设置 SIP Challenge 使用的`realm`字段是从哪个域获取，`auto_from`和`auto_to`分别是从 From 和 To 中获取，除了这两者，也可以是任意的字符串值，如：

```
<param name="challenge-realm" value="freeswitch.org.cn" />
```

- **disable-rtp-auto-adjust**: 大多数情况下，为了更好的穿越 NAT，FreeSWITCH 会自动调整（适应）RTP 包的来源 IP 地址，但在某些情况下（尤其是在`mod_dingaling`中会有多个候选 IP 的时候），FreeSWITCH 可能会改变本来正确的 IP 地址。该参数禁用此功能。

```
<param name="disable-rtp-auto-adjust" value="true" />
```

- **inbound-use-callid-as-uuid**: 在 FreeSWITCH 是，每一个 Channel 都有一个 UUID，该 UUID 是由系统生成的全局唯一的。对于来话，你可以使用 SIP 中的 Call-ID 字段来做 UUID，在某些情况下对于信令的跟踪分析比较有用。

```
<param name="inbound-use-callid-as-uuid" value="true" />
```

- **outbound-use-uuid-as-callid**: 与上一个参数差不多，只是在去话时可以使用 UUID 作为 Call-ID。

```
<param name="outbound-use-uuid-as-callid" value="true"/>
```

- **rtp-autofix-timing**: 在某些情况下自动修复 RTP 时间戳。

```
<param name="rtp-autofix-timing" value="false"/>
```

- **pass-callee-id**: 在支持的话机或系统间传送相关消息以更新被叫号码（在多个 FreeSWITCH 实例间使用X-FS-Display-Name和X-FS-Display-NumberSIP 头域实现）。可以设置是否支持这种功能。如：

```
<param name="pass-callee-id" value="false"/>
```

- **auto-rtp-bugs**: 在跟某些不符合标准设备对接时，为了最大限度的支持这些设备，FreeSWITCH 在这方面进行了妥协。可能的取值有CISCO_SKIP_MARK_BIT_2833和SONUS_SEND_INVALID_TIMESTAMP_2833等。使用该参数时要小心。如：
- **disable-srv**或**disable-naptr**: 这两个参数可以规避 DNS 中某些错误的 SRV 或 NAPTR 记录。如：

最后要讲的这几个参数允许根据需要调整 Sofia-SIP 库中底层的时钟，一般情况下不需要改动。这几个参数是给高级用户用的，一般来说保持使用默认值即可。更详细的说明请参考 FreeSWITCH 默认的配置文件中的说明及相关的 RFC3261。这些参数和默认值如下：

```
<param name="timer-T1" value="500" />
<param name="timer-T1X64" value="32000" />
<param name="timer-T2" value="4000" />
<param name="timer-T4" value="4000" />
```

3.3 mod_dptools

mod_dptools这应该是 FreeSWITCH 中最常用的模块了。

本模块实现了系统大部分的 Application (即我们常说的 App)。dptools 是 Dialplan Tools 的简写。也就是说，这些 App 大部分是在 Dialplan 中用的。

我们可以通过在 FreeSWITCH 中重新加载该模块来查看一些信息，如：

```
freeswitch> reload mod_dptools
...
Adding Application 'blind_transfer_ack'
Adding Application 'bind_digit_action'
Adding Application 'capture'
Adding Application 'clear_digit_action'
Adding Application 'digit_action_set_realm'
Adding Application 'privacy'
Adding Application 'set_audio_level'
Adding Application 'set_mute'
...
```

从上面的例子可以看出，里面实现了很多我们不熟悉的 App。关于这些 App，我想，等所有模块都讲完了后我们也可以一个一个的讲。

关于 App，我们前面讲过了，最常见的如 `answer`、`echo`、`bridge` 等，它们都是执行一些功能跟通话的另一端进行交互。还记得吗？我们以前说过，跟 FreeSWITCH 通信其实就是在跟 FreeSWITCH 中的一个 App 在通信。

除 App 外，该模块还实现了一些 Endpoint、Dialplan、API、File、Chat 等 Interface。

其中，有一个我们最熟悉的 Endpoint——`user`，是的，可能大家都天天在使用，但没有注意到它是一个 Endpoint。实际上，它相当于一个虚拟的 Endpoint，我们先看一下下面的例子：

```
<action application="bridge" data="user/1000"/>
```

这里，`user/1000`是一个呼叫字符串，而 `user` 就是个 Endpoint，实际上，它（在默认的配置下）相当于如下的字符串：

```
<action application="bridge" data="sofia/internal/1000@192.168.x.x"/>
```

其中，`192.168.x.x`是一个 IP (实际上它是一个 Domain，参见第2.1节)。所以，这里的 `sofia` 才是真正的 endpoint，而 `user` 只是相当于到 `sofia` 的一个快捷方式而已。

别的不说了，将该模块多 `reload` 几遍相信你在日志中就有更多的发现。然后，有时间可以看看源代码啊，很好懂的。

3.4 mod_commands

接下来，我们来看一下 mod_commands 模块。该模块实现了系统大多数的 API 命令，如 `version`、`status` 以及我们常用的 `originate` 等。

类似于上一节，我们可以通过 `reload` 模块的方式查看模块的一些信息，不妨输入下列命令试试：

```
freeswitch> reload mod_commands
+OK Reloading XML
-ERR unloading module [Module is not unloadable]
```

我们看到，该命令无情地抛出了一个错误，说该模块是无法被卸载的，自然，就无法重新加载了。看来，我们昨天学到的技术不好用了。我们只得另想办法。

我们知道 `fs_cli` 可以做为一个客户端连接到 FreeSWITCH 上，看日志，执行命令什么的。其实，它也有一个小的技巧，就是，用 `-x` 参数执行一个命令立即即出，如：

```
$ fs_cli -x "status"
UP 0 years, 0 days, 13 hours, 46 minutes, 47 seconds, 330 milliseconds, 78 microseconds
FreeSWITCH (Version 1.5.8b git 0c8d8ba 2014-01-16 01:06:11Z 64bit) is ready
6 session(s) since startup
0 session(s) - peak 1, last 5min 0
0 session(s) per Sec out of max 200, peak 1, last 5min 0
3000 session(s) max
min idle cpu 0.00/100.00
Current Stack Size/Max 240K/8192K
```

哇，既然能这样用就好办了。我们可以用下面的命令看一看 `mod_commands` 模块中都实现了哪些 API 命令：

```
$ fs_cli -x "help" | grep mod_commands
....,Shutdown,mod_commands
acl,<ip> <list_name>,Compare an ip to an acl list,mod_commands
alias,[add|stickyadd] <alias> <command> | del [<alias>|*],Alias,mod_commands
banner,,Return the system banner,mod_commands
bg_system,<command>,Execute a system command in the background,mod_commands
bgapi,<command>[ <arg>],Execute an api command in a thread,mod_commands
... 此处省略 XXXX 字
```

其实，这里的`help`命令也是在`mod_commands`里实现的，同样，`load/unload/reload`也都是，这也是为什么该模块不能卸载的根本原因——`unload`依赖`mod_commands`，所以，它不能卸载自己所在的模块。

当然，另外一个方法就是到 Wiki 上找文档了。该模块的文档还是挺全的。见：https://freeswitch.org/confluence/display/FREESWITCH/mod_commands。

通过这两节你应该会注意到一些小技巧：如通过`reload`一个模块查看某模块中实现了哪些`Interface`这种技巧。如果想好好学习的话一定要注意发现。以后，我们就不详细详解这些技巧的细节了。

3.5 mod_portaudio

与`mod_sofia`模块类似，`mod_portaudio`也是一个`Endpoint`模块。与`mod_sofia`不同的是，`mod_sofia`实现了`SIP`，而`mod_portaudio`实现了驱动本地声卡，即，它可以从本地麦克风获取输入，并将声音从耳机/扬声器中播放出来。因而，配合`mod_sofia`，可以用作一个软电话。

该模块默认是不编译的。到你的源代码树下，执行如下命令编译安装：

```
make mod_portaudio
make mod_portaudio-install
```

然后到控制台中，执行：

```
freeswitch> load mod_portaudio
```

如果得到『Cannot find an input device』之类的错误可能是你的声卡驱动有问题。如果是提示『+OK』就是成功了，接着执行：

```
freeswitch> pa devlist
API CALL [pa(devlist)] output:
0;Built-in Microphone;2;0;
1;Built-in Speaker;0;2;r
2;Built-in Headphone;0;2;
3;Logitech USB Headset;0;2;o
4;Logitech USB Headset;1;0;i
```

以上是在笔者 Mac 笔记本上的输出，它列出了所有的声音设备。其中，3 和 4 最后的『o』和『i』分别代表声音输出(out)和输入(in)设备。在你的电脑上可能不一样，如果你想选择其他设备，可以使用命令：

```
freeswitch> pa indev #0
freeswitch> pa outdev #2
```

以上命令会选择我电脑上内置的麦克风和耳机。

接下来你就可以有一个可以用命令行控制的软电话了，酷吧？

```
freeswitch> pa looptest      (回路测试, echo)
freeswitch> pa call 9999
freeswitch> pa call 1000
freeswitch> pa hangup
```

如上所示，你可以呼叫刚才试过的所有号码。现在假设想从 SIP 分机 1000 呼叫到你，那需要修改拨号计划(Dialplan)。用你喜欢的编辑器编辑以下文件放到`conf/dialplan/default/portaudio.xml`中：

```
<include>
<extension name="call me">
<condition field="destination_number" expression="^(me|12345678)$">
<action application="bridge" data="portaudio"/>
</condition>
</extension>
</include>
```

然后，在控制台中按『F6』或输入以下命令使之生效：

```
FS> reloadxml
```

在分机 1000 上呼叫『me』或『12345678』(你肯定想为自己选择一个更酷的号码)，然后在控制台上应该能看到类似『[DEBUG] mod_portaudio.c:268 BRRRING! BRRRING! call 1』的输出(如果看不到的话按『F8』能得到详细的 Log)，这说明你的软电话在振铃。多打几个回车，然后输入『pa answer』就可以接听电话了。『pa hangup』可以挂断电话。

当然，你肯定希望在振铃时能听到真正的振铃音而不是看什么 BRRRING。好办，选择一个好听一点的声音文件(.wav格式)，编辑`conf/autoload_configs/portaudio.conf.xml`，修改下面一行：

```
<param name="ring-file" value="/home/your_name/your_ring_file.wav"/>
```

然后重新加载模块：

```
freeswitch> reloadxml  
freeswitch> reload mod_portaudio
```

再打打试试，看是否能听到振铃音了？

如果你用不惯字符界面，可以看一下 FreeSWITCH-Air²，它为 FreeSWITCH 提供一个简洁的软电话的图形界面。另外，如果你需要高清通话，除需要设置相关的语音编解码器(codec)外，你还需要有一幅好的耳机才能达到最好的效果。笔者使用的是一款 USB 耳机。

另外两款基于 FreeSWITCH 的软电话是[FSComm](#) (QT 实现) 和[FSClient](#) (C# 实现)。

3.6 mod_rtmp

`mod_rtmp`是另外一个 Endpoint。它实现了 RTMP 协议。RTMP 协议是 Adobe Flash 中使用的协议。

通过使用 Flash，便可以在浏览器中使用 Flash 版软电话了，用户再也不需要下载一个 SIP 客户端，这对于互联网应用来说是非常有吸引力的。

虽然近几年兴起的 WebRTC 着实令人激动，但由于顽固的 IE 至今不支持 WebRTC，因而，Flash 还是比较有效的解决方案。在此之前，大家一般是使用 ActiveX 控件的方式在 IE 浏览器中支持 SIP 客户端。

该模块的安装很简单，直接在 FreeSWITCH 源码目录中使用`make mod_rtmp-install`安装即可。

在源码目录的`client`目录下也包含了 Flex 软电话的源码和例子。该客户端也提供相关的 Javascript API 以便在网页上对其进行控制。

该模块也支持注册，它也是使用 FreeSWITCH 的 XML Directory 进行用户认证。因此，可以在 Javascript 中使用类似下面的代码注册：

```
flash.login('1000@192.168.1.2', '1234')
```

²<https://wiki.freeswitch.org/wiki/FsAir>。

Flash 注册后，就可以在 FreeSWITCH 中使用下列命令查看注册用户的情况：

```
rtmp status profile default reg
```

是不是看起来跟 Sofia 命令很像啊。

它也可以使用如下的 API 外呼：

```
flash.makeCall(number, account, options);
```

当然，它也可以做被叫，找到被叫的地址是靠 `rtmp_contact` API 实现的，如

```
rtmp_contact(user@domain)
```

详细情况参见https://freeswitch.org/confluence/display/FREESWITCH/mod_rtmp。也可以参考第4.8.1节的相关内容。

3.7 mod_skypeopen

`mod_skypeopen` 是一个有意思的模块，它提供 FreeSWITCH 与 Skype 的互通。

Skype 是世界上广泛使用的 VoIP 软件，不过，在被微软收购后表现不怎么样，最近又传出 Skype 将停止提供第三方 API 接口，也就是说 FreeSWITCH 将无法与新版的 Skype 互通。

不过，旧版的 Skype 好像不受影响，而且，不管以后 FreeSWITCH 是否能与 Skype 互通，`mod_skypopen` 对我们都有借鉴意义。

`mod_skypopen` 的前身是 `mod_skypiax`，后来才改为此名。该模块实现的很有意思。Skype 不是提供 API 吗？也就是说它允许外部的程序通过一定的 API 控制 Skype（相当于通信信令）。所以，理论上讲，就可以将控制 Skype 的部分代码放到 FreeSWITCH 中，而仍然需要启动一个 Skype 实例登录到实际的 Skype 网络中与其它 Skype 进行通信。在 Linux 中，通过使用内核中的虚拟声卡，可以指定不同的 Skype 实例使用不同的虚拟声卡，而在 FreeSWITCH 中通过访问这些虚拟声卡读写音频数据（相当于媒体流），进而完成信令和媒体的交互。

最初的 Skype 一个 Skype 账号在一台主机上只允许登录一次，所以我们最初使用时，注册了 20 个 Skype 账号，在一台 Linux 服务器上启动了 20 个 Skype 实例，并分别用不同的账号注册上去。对于 FreeSWITCH 而言，这 20 个实例就相当于 20 条外线，我们的 SIP 账号就可以通过这 20 条『Skype』外线打电话到其它的 Skype 账号上。

有什么用呢？当初我们在做在线一对一英语口语教学时，美国的老师使用 SIP 客户端，而学员一般使用手机（SIP 客户端在我国不怎么稳定）与老师对话。但问题是，有的学员手机信号不好，有的呢，千里迢迢漫游到北京去上学习班（但却买我们的网络口语教学课程），无法支付高昂的漫游费。而使用 Skype 就解决了这一问题。

后来，Skype 允许在一台主机上用同一账号多次登录，所以，我们就用同一个 Skype 账号登录 20 个 Skype 实例，实现了类似模块中继线的功能——打出去对外显示一个号。

当然，该模块是跨平台的，在 Windows 上也可以使用，也支持多账号。当前闲着没事的时候，笔者也实现了在 Mac 上的支持，只不过后来没有继续开发，也从来没有合并到主分支里去。

`mod_skype`也是一个 Endpoint。

它的详细安装和使用方法参见：https://freeswitch.org/confluence/display/FREESWITCH/mod_skypopen。

3.8 mod_cdr_csv

`mod_cdr_csv`是 FreeSWITCH 中写话单的模块。该模块会以 CSV (Comma-Separated Values, 即以逗号分隔的值) 格式写话单，当然，其它也可以通过配置模板选择将话单写成任何格式。

话单文件的默认存储位置是在`/usr/local/freeswitch/log/cdr-csv`目录下，当话单文件增长到一定程度，便会发生轮转 (rotate)，即，关闭原来的话单文件，重新生成新的话单文件。旧的话单文件会以日期时间的文件名格式保存，如：

1003.csv.2014-01-24-19-22-57

当然，如果话单文件还没有增长到一定程序时，也可以手工触发话单`rotate`，如在 FreeSWITCH 控制台上执行如下命令：

freeswitch> cdr_csv rotate

该模块本身没有什么值得多说的，下面说一下话单文件的处理。其实我想说的也不是如何处理话单，而是如何及时有效地处理话单。

笔者以前处理话单一直使用一个叫`iwatch`的工具，该工具是一个 perl 写的工具，通过配置正确的 XML 文件，可以让`iwatch`监听一个目录，`iwatch`使用 Linux 文件系统的`inotify`机制监视文件系统，一旦文件系统发生变化，便可以执行相应的脚本。

但是，该工具在 CentOS 上装起来特别麻烦，而且，它又依赖于无数的 Perl 模块，因此，对于一些上网不方便的环境部署起来特别麻烦。后来，又找到一个 C 语言写的工具：https://freeswitch.org/confluence/display/FREESWITCH/mod_cdr_csv。

//github.com/rvoicilas/inotify-tools/wiki，该工具看起来也不错，装起来也简单。安装后，我写了如下的脚本：

```
#!/bin/sh

CURPATH=`pwd`
FSPATH=/usr/local/freeswitch/log/cdr-csv

inotifywait -mr --timefmt '%d/%m/%y %H:%M' --format '%T %w %f' -e close_write $FSPATH | while read date time dir file
do
    FILECHANGE=${dir}${file}
    # convert absolute path to relative
    FILECHANGEREL=`echo "$FILECHANGE" | sed 's_'"$CURPATH"'/_'`"

    echo "At ${time} on ${date}, file $FILECHANGE was backed up to ..."
    echo mv $FILECHANGE /tmp/
done
```

当然，该脚本仅仅是个简单的例子，它使用了`inotifywait`监视一个目录（即`/usr/local/freeswitch/log/cdr-dsv`），当该目录中发生了`close_write`事件（即当文件被以写入方式打开又关闭时）时，即话单写完了，我们可以执行一个命令把它移动（或复制）到其它的目录中去。当然，没有人阻止你修改该脚本，解析 CDR 再写入数据库之类的。

总之，笔者认为这是很简单而又灵活的实现方式。UNIX 的哲学就是 KISS 嘛——Keep It Simple, Stupid。

3.9 mod_cluechoo

`mod_cluechoo`这个模块是个很好玩的模块。很多人对此很好奇，不知道它有什么用。实际上，它真没什么用。用当比较流行的一句网络流行语说就是：『然而它并没有什么卵用』。

从名字看，不知道该模块是干什么的，而 Wiki 上对它的介绍也就只有一句话：`mod_cluechoo` implements Toyoda Masashi's famous Steam Locomotive (SL) command for FreeSWITCH。即，该模块在 FreeSWITCH 中实现了 Toyoda Masashi 的蒸汽机车命令。那么，该蒸汽机车又到底是什么东东呢？

说来话长，通过搜索引擎，我们找到了这个页面：http://www.tkl.iis.u-tokyo.ac.jp/~toyoda/index_e.html。在 UNIX 系统上有一个`ls`命令，但有时候容易输错，成了`s1`，便会提示命令错误。而该蒸汽机车实现了一个`s1`命令，即蒸汽机车 (Steam Locomotive) 的英文缩写。实际上，该程序只是一个小小的玩笑，没什么实际用处。该程序手册上对 SL 的介绍是这样的：SL (Steam Locomotive) runs across your terminal when you type “s1” as you meant to type “ls”. It’s just a joke command, and **not usefull at all**. Put the binary to /usr/local/bin.

那么，不管有用没用，在 FreeSWITCH 中怎么玩呢？

该模块是默认编译和加载的，在 FreeSWITCH 控制台上重新加载一下该模块，便可以从日志中看出一些信息：

```
freeswitch> reload mod_cluechoo

[CONSOLE] switch_loadable_module.c:1464 Successfully Loaded [mod_cluechoo]
[NOTICE] switch_loadable_module.c:269 Adding Application 'cluechoo'
[NOTICE] switch_loadable_module.c:315 Adding API Function 'cluechoo'
```

从上面的日志中可以看出，该模块实现了一个 cluechoo App，以及一个 cluechoo API。如果对 App 和 API 是什么东东不了解的同学可以参考第1.6.5节。。

接着在命令控制台上输入以下命令，便会看到一辆小火车开过，如图3.1：

```
freeswitch> cluechoo
```

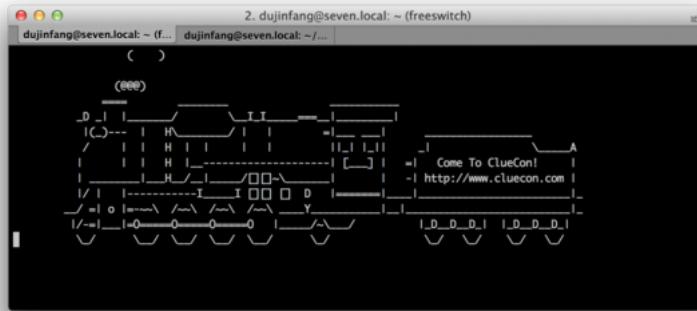


图 3.1: ClueChoo 小火车

那么 App 怎么使用呢？构造如下的 Dialplan，拨打 cluechoo 试一试。

```
<extension name="cluechoo">
    <condition field="destination_number" expression="^cluechoo$">
        <action application="answer" data="" />
        <action application="cluechoo" data="" />
    </condition>
</extension>
```

有兴趣的同学也要以看一看其源代码实现，挺有趣的哟:)。

3.10 mod_speex

Speex³是一种音频编码格式，并专门针对语音进行优化。它支持窄带（8KHz）及宽带（16kHz）的编码，支持可变比特率（VBR）以及丢包补偿等，因此非常适合 VoIP 应用。而且，由于其采用 BSD 许可证，而且宣称没有任何专利问题，因而得到了广泛的使用。比方在 RTMP 中就使用 speex 编码。

最初，FreeSWITCH 对 speex 编码的支持是在mod_speex中实现的，但是，在 2014 年 2 月份它被移到核心中去了⁴，代码提交说明是『move speex codec into the core since it already has speed anyway』。原因就是在 FreeSWITCH 的核心中已经加载了 Speex 代码库（用于 resample，即抽样频率转换），mod_speex的功能就无需要再存在于外部模块中了。

把该模块移到核心意味着它总是可用的，并不能选择被卸载。但是，该模块很小，如果不用，也占不了多少资源。

3.11 mod_sonar

Sonar 译为声纳，又译声呐，其英文全称为『Sound Navigation And Ranging』，是一种利用声波在水下的传播特性，通过电声转换和信息处理，完成水下探测和通讯任务的电子设备。它有主动式和被动式两种类型，属于声学定位的范畴。声纳系统的音频从次声波到超音波都有。

mod_sonar类似于一个真正的声纳。它的工作原理是这样的——首先你在远端启动一个服务器，能将收到的来话信息中的音频原样返回（可以使用 FreeSWITCH 中的echo App 实现）。然后在本地的 FreeSWITCH 上产生一些铃音，发送到远端的服务器上再反射回来，然后使用 VAD 检测功能可以检测这些铃音，从而可以在某种程度上确定网络的质量。

加载该模块后，将会在日志中看到如下的消息：

```
[NOTICE] switch_loadable_module.c:269 Adding Application 'sonar'
```

可见，该模块只实现了一个sonar App。

为进行测试，我在一台远端的 FreeSWITCH 服务器上使用如下 Dialplan 做了一个 Echo 服务：

```
<extension name="public_dialplan_echo">
<condition field="destination_number" expression="^(\echo)$">
<action application="answer"/>
```

³参见：<http://en.wikipedia.org/wiki/Speex>。

⁴参见：<http://fisheye.freeswitch.org/changelog/freeswitch.git?cs=ab56c276a0881f8db37e71d4f23d7fd682433ab0>。

```
<action application="echo"/>
</condition>
</extension>
```

然后，在笔者的电脑上使用如下的命令进行测试（其中x.x.x.x为笔者的远端服务器的 IP 地址）：

```
originate sofia/internal/echo@x.x.x.x:5080 &sonar
```

日志输出如下：

```
[DEBUG] switch_ivr_async.c:3087 TONE ping HIT 1/1
[DEBUG] switch_ivr_async.c:3093 TONE ping DETECTED
[NOTICE] mod_sonar.c:87 Sonar ping took 379 milliseconds
[NOTICE] mod_sonar.c:87 Sonar ping took 140 milliseconds
[NOTICE] mod_sonar.c:87 Sonar ping took 380 milliseconds
...
[INFO] mod_sonar.c:179 Sonar Ping (in ms):
      min:140 max:380 avg:303 sdev:103 mdev:83 sent:5 recv: 5 lost:0 lost/send:0.00%
```

从日志中可以看出，sonar 这个 App 向远端发送了 5 次 ping 请求（实际上发送 5 次音频数据），然后收到（检测到）5 次反射回来的音频，最小时长是 140ms，最大是 380ms，平均值是 303ms，其它的值是均方差等。看起来还算不错。

当然，为了测试更准确一些，也可以给 sonar 一个参数，表示你想测试的次数，如，下列命令将测试 10 次：

```
originate sofia/internal/echo@x.x.x.x:5080 &sonar(10)
```

下面是笔者用操作系统的 ping 命令得到的结果，可以看出，操作系统的 ping 包平均只需要 28ms 就能走一个来回。

```
7 packets transmitted, 7 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 16.233/28.495/80.739/21.465 ms
```

3.12 mod_rss

RSS 的全称是 Really Simple Syndication，即简易信息聚合，它是一种消息模式规范。虽然，大部分博客都输出 RSS 格式的信息，便于 RSS 客户端阅读，但并不能说 RSS 只是订阅博客的，因为任何信息都可以输出到 RSS。

该模块的功能是阅读本地的 RSS 文件。当然，这些 RSS 文件可以定期用外部程序甚至直接在 Dialplan 中通过 Javascript 或 Lua 等更新。

该模块默认是不安装的，到源代码目录中执行`make mod_rss-install`即可安装。

然后，下载一个 RSS 文件，如，可以下载笔者的博客提供的 RSS：

```
cd /tmp
wget http://www.dujinfang.com/feed.xml
```

把`conf/autoload_configs/rss.conf.xml`改成如下的样子：

```
<configuration name="rss.conf" description="RSS Parser">
<feeds>
  <feed name="Seven's Blog">/tmp/feed.xml</feed>
</feeds>
</configuration>
```

然后，创建如下的 Dialplan，拨打 rss，它便能朗读 RSS 了。当然，为了让它能朗读中文，我使用了`mod_tts_commandline`模块（在 FreeSWITCH 权威指南上有详细介绍），并编写了一个脚本让它使用我的 Mac 上提供的 TTS 引擎进行朗读。

```
<extension name="rss">
<condition field="destination_number" expression="^rss$">
  <action application="answer" data="" />
  <action application="rss" data="tts_commandline Ting-Ting"/>
</condition>
</extension>
```

在朗读过程中，还可以按 0 选择项目，按 4 听上一条，按 6 听下一条等。有兴趣的可以试一下，很有趣的。

3.13 mod_conference

FreeSWITCH 支持多人电话会议，该功能是由mod_conference模块实现的。

mod_conference是实现了一个conferenceApp 和 API。在默认的配置中，直接拨打 3000 就可以进入一个会议。

从下列 Dialplan 可以看出，它使用conference App 将来话送入一个会议：

```
<extension name="nb_conferences">
<condition field="destination_number" expression="^(30\d{2})$">
    <action application="answer"/>
    <action application="conference" data="$1-${domain_name}@default"/>
</condition>
</extension>
```

其中，conference App 的参数是一个会议的名称和一个会议的 Profile，它们之间是以@分隔的。`domain_name`一般是一个 IP 地址，所以，假设我们呼叫的是3000，FreeSWITCH 的 IP 地址是192.168.1.2的话，将上述参数进行变量替换后，结果是：3000-192.168.1.2@default。

@前面的就是会议的名称，后面的`default`是会议的一个 Profile。

会议的 Profile 定义了会议的一些特性，如各种提示音等。

为了简单起见，我们以3000作为会议的名称，修改 Dialplan 如下：

```
<action application="conference" data= "3000@default"/>
```

这个，拨打 3000 就可以进入3000这个会议（室）。通过conference API 命令，可以对会议执行一系列的操作，如下列命令将全部人员静音

```
conference 3000 mute all
```

取消静音：

```
conference 3000 unmute all
```

其中，会议中的每一路参与的电话称作一个成员（member），以`member_id`来标识，使用下列命令可以列出会议中所有的成员，进而知道成员的`member_id`：

```
conference 3000 list
```

得到了成员的 member_id 后，便可以对成员进行单独操作，如以下命令仅对member_id为2的成员静音：

```
conference 3000 mute 2
```

或将它踢出会议

```
conference 3000 kick 2
```

当然，该命令的参数还有很多，具体的可输入不带参数的conference命令查看帮助。

```
conference
```

除语音会议外，FreeSWITCH 也支持视频会议。1.4 版以前的 FreeSWITCH 不支持视频的转码，因而仅支持主持人式的视频会议。具体的讲，所有与会成员中会有一个成员会获得一个 video floor，持有该 floor 的人的视频会广播到其它所有成员的终端上，即，所有成员都会看到该成员的视频。在默认情况下，floor 的持有和转换是通过声音激励的方式实现的，即，通过一点的算法，根据当前声音的能量值来决定谁持有 floor，一般来说是当前说话的成员持有该 floor，即谁说话看谁。

自 1.6 版开始，FreeSWITCH 支持视频转码和视频会议，我们以后再讲。

3.14 mod_enum

说起enum，千万不要跟 C 语言里的枚举类型（enum）混了。ENUM 是 E.164 Number to URI Mapping 的缩写，即 E.164 号码到 URI 的映射关系。如果说到这里你还不理解的话，想象一下域名到 IP 地址的映射关系吧。当你在 Internet 上访问一个域名（网址）时，如www.freeswitch.org.cn，将先由 DNS 服务器将该域名翻译成一个 IP 地址，进而，你的注册品会访问该 IP 地址以获取它想要的网页内容。ENUM 也是一样，它只不过是使用了类似 DNS 的技术，将一个 E.164 的电话号码（也就是我们常用的电话号码）映射为一个相应的 VoIP 地址（可以是 SIP、H323 或 Jabber 的地址），这样，我们就可以通过拨打一个普通的电话号码来访问一个 VoIP 地址。

提供这个号码映射关系的网站有 e164.org 和 freenum.org 等，有兴趣读者可以试一下。

如果读者比较仔细的话，可以看到在 FreeSWITCH 默认的 Dialplan 中有如下的设置：

```
<extension name="enum">
<condition field="{{$module_exists(mod_enum)}} expression="true"/>
<condition field="destination_number" expression="^(.*)$">
<action application="transfer" data="$1 enum"/>
</condition>
</extension>
```

上述设置的意思是，如果以上所有 Dialplan 都没有相应的匹配规则的话，那么 Dialplan 的匹配将进入这里，该 Extension 将匹配所有被叫号码，并最终通过 transfer App 转入 enum Dialplan 进行进一步的路由查找

enum Dialplan 会根据相应的配置查找相应的 enum 服务器查找是否有匹配的号码，如果能找到与被叫号码相匹配的项（无论是 SIP 地址还是其它 VoIP 地址），电话将进而桥接到相应的地址。

3.15 mod_spidermonkey

SpiderMonkey 是第一款最早面世的 JavaScript 引擎，它是 Mozilla 使用的 Javascript 引擎。FreeSWITCH 也使用了该引擎来在 FreeSWITCH 内部支持 Javascript 嵌入式语言。

FreeSWITCH 中对 Javascript 脚本的支持是在 mod_spidermonkey 中实现的。通过该模块，可以使用 Javascript 控制呼叫流程，如，你可以使用如下 Dialplan 将电话交给一个 Javascript 脚本处理：

```
<action application="javascript" data="/tmp/test.js" />
```

然后，创建如下 js 文件：

```
session.answer();
session.sleep(1000);
session.streamFile("/tmp/hello-js.wav");
session.hangup();
```

在 js 脚本中，可以得到一个 session 对象（Object），然后就可以对该 session 进行操作了。一个 session 代表一路通话。从上面的 js 脚本中很容易看出，第一行用于对来话进行应答，第二行小睡一会（1000 毫秒），第三行则播放一个 .wav 文件，第四行挂机。

是的，它等于以下 Dialplan：

```
<action application= "answer" />
<action application= "sleep" data= "1000" />
<action application= "playback" data= "/tmp/hello-js.wav" />
<action application= "hangup" />
```

但是，大家都知道，在 Dialplan 中加入逻辑判断等功能都是有限的，但在 Javascript 中就不同了，你很容易写一些 if-else 之类的语句判断当前的场景进而执行不同的动作。关于这一点，我们在此就不举例子，有兴趣的可以自己研究一下。

除了 SpiderMonkey 外，另一个 Javascript 引擎是 V8，这是一个在 Google Chrome 中以及 Node.js 中使用的 Javascript 引擎，它提供了比 SpiderMonkey 更好的性能。FreeSWITCH 中也实现了一个 mod_v8 模块。该模块除了与 mod_spidermonkey 有一些小小的差别外，在 API 上完全兼容。也就是说，上述的 test.js 脚本在 mod_v8 下也能正常运行。

由于 mod_v8 的出现，现在 mod_spidermonkey 已经不推荐使用了，并被移到源代码库的 legacy 目录下。

3.16 mod_lua 和 mod_v8

这两个模块都是在 FreeSWITCH 支持使用嵌入式语言来控制呼叫流程的。我们在以前的文章中也曾不同程度地提到他们。

其中，mod_lua 支持 Lua 语言，Lua 是很有名的嵌入式语言，能非常好地嵌入其它语言的程序中。该模块有两个版本，最新的版本支持 Lua 5.2，而支持 Lua 5.1 的版本现在已被移动到源代码的 legacy 目录中。

而 mod_v8 支持 Javascript，由 Google 的 v8 库提供支持，v8 库已被成功用于 Chrome 及 node.js 中，性能强劲。其实对 Javascript 的支持也有两个版本，另一个版本也在 legacy 目录中，叫做 mod_spidermonkey。Spidermonkey 是 Firefox 中的 Javascript 引擎。

Lua 和 Javascript 两种语言有很多相似的地方，我们不必多说，先来看两个脚本：

test.lua

```
session:answer()
session:sleep(1000)
session:streamFile("/tmp/hello-lua.wav")
session:hangup()
```

test.js

```
session.answer();
session.sleep(1000);
session.streamFile("/tmp/hello-js.wav");
session.hangup();
```

可以看出，两个脚本的内容和功能基本上一样。第一行，对当前呼叫进行应答；然后，小睡一会儿（以确定能正确的建立媒体流）；接着，播放一个声音文件；最后，挂机。

可以分别使用如下 Dialplan 将来话路由到上述脚本：

```
<action application= "lua"  data= "/tmp/test.lua" />
```

或

```
<action application= "javascript"  data= "/tmp/test.js" />
```

当然，两个脚本本质上是 Lua 或 Script，因而你可以很容易地在脚本中加入条件判断、循环等，甚至也可以通过相关的 API 与数据库或其它系统交互。能实现什么样的程序，完全靠你的想象力，从现在起，就写个脚本练一练吧？

3.17 mod_rtc

mod_rtc是一个 Endpoint 模块，但它只处理媒体，不包含信令，因而，只能配合其它信令模块使用。在该提交信息里是这么说的：『add new stub module mod_rtc dummy signaling-free media engine endpoint』。意思是，该模块是一个空的 Endpoint 模块，而且，该模块是跟信令无关的。

但是，它却可以独立运行。笔者测试了一下，编译安装后，加载该模块可以看到如下的输出：

```
freeswitch> load mod_rtc
[CONSOLE] switch_loadable_module.c:1466 Successfully Loaded [mod_rtc]
[NOTICE] switch_loadable_module.c:149 Adding Endpoint 'rtc'
```

可以看出，它实现了一个名为 rtc 的 Endpoint，我们可以使用如下的字符串试一下建立一个 Channel：

```
freeswitch> bgapi originate rtc/test &echo  
+OK Job-UUID: 0f41a0a2-63a5-4229-a8e0-c8e5281cce79
```

下列命令可以看出它确实建立了一个 Channel:

```
freeswitch> show channels  
  
uuid,direction,created,created_epoch,name,state, ...  
725369b7-0731-497f-9500-1e3c128b7626,outbound,2014-05-31 21:07:39,1401541659,N/A,CS_CONSUME_MEDIA,
```

当然，`rtc/test` 是我们按规则随便造的一个呼叫字符串，从现在的情况来看，它只是生成了一个 Channel，现在还是什么都做不了。所以，我们也没什么可玩的，使用如下命令挂机：

```
freeswitch> hupall
```

当然，实现该模块的代码还是很有意思的，它现在仅有短短的 463 行，对搞不懂`mod_sofia`的庞大又想学习写一个 Endpoint 模块的读者来说，是极好的学习材料。这次提交的哈希值是`5138f4d`，可以用如下命令查看：

```
$ git show 5138f4d
```

其实，`mod_verto`就依赖该模块处理 RTP 媒体流，只是，用户不需要手工的调用，一般对终端用户而言是不可见的。但如果你是一位开发者，可以从源代码中一窥端倪。

我们将在下一节中讲`mod_verto`。

3.18 mod_verto

上一节，我们讲了`mod_rtc`。`mod_rtc`是一个纯媒体的模块，目的是为了支持 WebRTC。然而，任何的通信都需要一定的信令支持，`mod_verto`就是配合`mod_rtc`的信令模块。

众所周知，WebRTC 从诞生的第一天起就只定义了媒体的交互和传输，而把信令留给大家自己实现，以便有更大的自由度。最初，大部分 WebRTC 的例子都是基于 GAE 的，但在电信的 VoIP 领域，SIP 还是占统治地位的，因而，包括 FreeSWITCH 在内，又有一些 SIP 代理和软交换设备实现了配合 WebRTC 使用的 SIP 信令，这但是 SIP over WebSocket。FreeSWITCH 对 SIP over WebSocket 的支持是直接扩展了 Sofia-SIP 协议栈。

但无论如何，虽然 SIP 与传统的 VoIP 协议如 H323 相比，脱离了老式的电信信令思维，采用了类似 HTTP 协议的文本协议，但，它从电信领域诞生的基因决定了它还是很难融入互联网，退一步讲，人们还是认为 SIP 通信专业性太强了，SIP 就是 SIP，互联网就是互联网。

单从协议内容角度讲，SIP 对于浏览器尤其是对于移动浏览器来说，还是有些庞大了。而基于文本的 SIP 协议解析起来对浏览器来说，即使不是一种负担，也不是非常的优雅。对浏览器来说，最适合的数据格式是 JSON 已是不争的事实。

我们很高兴地看到，FreeSWITCH 团队开放了 `mod_verto`。它采用了 JSON 及 JSON-RPC 相关的信令协议，非常优雅的与 `mod_rtc` 相配合，将热闹的互连网与冷冰冰的 SIP 通信结合在了一起。也就是说，FreeSWITCH 不再是互联网从业者眼里专业的运动员，而跟 MySQL，Apache 一样，可以实实在在的融入互联网了。

在 WebRTC 设计之初，就非常重视安全问题，因而，一切都是加密的，不管是在媒体层还是在信令层，这是个好事，唯一比较麻烦的是，对于没有耐心的实践者来说，你在跑通 `mod_verto` 前要设置好你的 Web 服务器以及证书。

除了 Endpoint 功能外，`mod_verto` 其实还自己带了一个 HTTP/HTTPS 服务器。这是一个隐藏选项，下面，我们看一下其配置：

```
<vhosts>
    <vhost domain="localhost">
        <param name="alias" value="seven.local.freeswitch.org"/>
        <param name="root" value="/usr/local/freeswitch/htdocs"/>
        <param name="script_root" value="/usr/local/freeswitch/scripts"/>
        <param name="index" value="index.html"/>
        <!--
            <param name="auth-realm" value="FreeSWITCH"/>
            <param name="auth-user" value="freeswitch"/>
            <param name="auth-pass" value="rocks"/>
        -->
        <rewrites>
            <rule expression="^/api" value="/my_custom_api.lua"/>
            <rule expression="^/channels" value="/rest.lua"/>
        </rewrites>
    </vhost>
```

在上述配置中，`alias` 其实没什么用，备以后扩展；`root` 要指向 Web 服务的根目录；`index` 是默认的首页；`auth-` 相关的是 HTTP Basic 认证；`rewrites` 会重写请求地址，比如把请求重定向到一个 Lua 脚本进行处理（类似传统的 CGI）。

把上述配置加到 Verto 的 Profile 部分，就可以有一个内置的小型的 HTTP 服务器了。

下面的 Lua 脚本可以提供 REST 风格的 API，读者请参考 <https://freeswitch.org/confluence/display/FREESWITCH/Restful> 自行解析：

```
--[]

Restful by Seven Du.

GET    /channels
GET    /channels/uuid
POST   /channels
PUT    /channels/uuid
DELETE /channels/uuid
DELETE /channels
[]

function headers()
    stream:write("HTTP/1.0 200 OK\r\n")
    if (accept == "application/json") then
        stream:write("Content-Type: application/json\r\n")
    else
        stream:write("Content-Type: text/plain\r\n")
    end
    stream:write("\r\n")
end

-- print(env:serialize())

api = freeswitch.API()
method = env:getHeader("Request-Method")
http_uri = env:getHeader("HTTP-Request-URI")
http_query = env:getHeader("HTTP-QUERY")
accept = env:getHeader("Accept")
uid = string.sub(http_uri, "11") -- remove /channels/ from uri
freeswitch.consoleLog("ERR", "[" .. method .. "] \n")
-- freeswitch.consoleLog("ERR", http_uri .. "\n")
-- freeswitch.consoleLog("ERR", http_query .. "\n")

if (method == "GET") then
    if not (uid == "") then
        format = ""
        if accept == "application/json" then
            format = " json"
        end
        ret = api:execute("uuid_dump", uid .. format)
    else
        format = ""
        if (accept == "application/json") then
            format = " as json"
        end
        ret = api:execute("show", "channels" .. format)
    end
elseif (method == "POST") then
```

```

dest = env:getHeader("destNumber")
app = "echo"
dialstr = "user/" .. dest .. " &" .. app
print(dialstr)
ret = api:execute("originate", dialstr)

elseif method == "PUT" then
    if action == "nomedia" then
        cmd = "uuid_media"
        arg = uuid .. " off"
    elseif action == "media" then
        cmd = "uuid_media"
        arg = uuid .. " on"
    elseif action == "hold" then
        cmd = "uuid_hold"
        arg = "on" .. uuid
    elseif action == "media" then
        cmd = "uuid_media"
        arg = "off" .. uuid
    end
    ret = api:execute(cmd, arg)
elseif method == "DELETE" then
    if not (uuid == "") then
        cmd = "uuid_kill"
        arg = uuid
    else
        cmd = "hupall"
        arg = ""
    end
    ret = api:execute(cmd, arg)
end
headers()
stream:write(ret .. "\n")

```

使用 Curl 测试 RESTfulAPI 的例子如下：

```

curl -O localhost:8081/channels
curl -O -XPOST -d "destNumber=1000" localhost:8081/channels
curl -O localhost:8081/channels/1f0802b7-3568-4eb6-b372-182861b56d9b
curl -O -H "Accept: application/json" localhost:8081/channels/1f0802b7-3568-4eb6-b372-182861b56d9b
curl -O -XDELETE localhost:8081/channels

```

关于本模块更详细的信息，请参阅：https://confluence.freeswitch.org/display/FREESWITCH/mod_verto。

3.19 mod_xml_rpc

其实除了mod_verto外，在FreeSWITCH中很早就有另一个模块可以提供HTTP Server功能，这就是mod_xml_rpc。

该模块是最早的HTTP Server功能，在FreeSWITCH Console上执行

```
load mod_xml_rpc
```

可以看到类似如下输出：

```
mod_xml_rpc.c:1242 Starting HTTP Port 8080, DocRoot [/usr/local/freeswitch/htdocs]
```

表明一个监听8080端口的HTTP Server已经准备好了。

该模块默认使用HTTP Basic验证，用户名和密码都在xml_rpc.conf.xml中配置。

用浏览器打开FreeSWITCH服务器的8080端口，输入密码，就进入了一个小型的网站。里面的功能不多，不过，FreeSWITCH自己带了一个小小的FreeSWITCH Portal，这一块在《FreeSWITCH权威指南》里写得很详细，我们就不多说了。

我们来看看它提供的API：

```
$ curl --user freeswitch:works localhost:8080/api/status
<h1>FreeSWITCH Status</h1>
2015-06-30 18:42:49<br>
UP 0 years, 1 day, 21 hours, 31 minutes, 33 seconds, 325 milliseconds, 220 microseconds<br>
FreeSWITCH (Version 1.7.0 git 2a1195e 2015-06-26 19:51:40Z 64bit) is ready<br>
2 session(s) since startup<br>
0 session(s) - peak 1, last 5min 0 <br>
0 session(s) per Sec out of max 30, peak 1, last 5min 0 <br>
1000 session(s) max<br>
min idle cpu 0.00/100.00<br>
Current Stack Size/Max 240K/8192K
```

很熟悉吧，跟在Console上执行的结果基本一样。如果你熟悉Curl的话，你就明白以上命令向FreeSWITCH发了一个HTTP请求，并得到了结果。

其实我们完全可以执行更复杂的命令，如

```
$ curl --user freeswitch:works localhost:8080/api/sofia?status
      Name          Type           Data      State
=====
  192.168.7.6    alias        internal   ALIASED
  external       profile      sip:mod_sofia@192.168.7.6:5080  RUNNING (0)
....
```

甚至发起一个呼叫，如：

```
curl --user freeswitch:works "localhost:8080/api/originate?user/1000%20%26echo"
```

其中，参数经过 urlencode 处理，经过 FreeSWITCH urldecode 以后的命令就相当于：

```
originate usr/1000 &echo
```

当然，更强大的还在后台，因为，你可以调用 Lua

```
$ curl --user freeswitch:works "localhost:8080/api/lua?/tmp/test.lua"
Hi, 欢迎交流 FreeSWITCH-CN 微信公众号
```

对应的 Lua 脚本内容如下：/tmp/test.lua

```
stream:write("Hi, 欢迎交流 FreeSWITCH-CN 微信公众号\n")
```

有了 Lua，就有了无限可能。比方，你可以 POST 一些参数：

```
$ curl --user freeswitch:works -XPOST -d "a=1&b=2" "localhost:8080/api/lua?/tmp/test.lua"
Hi, 欢迎交流 FreeSWITCH-CN 微信公众号，我收到的参数是 a=1 b=2
```

对应的 Lua 脚本是：

```
a = env:getHeader("a")
b = env:getHeader("b")

stream:write("Hi, 欢迎交流 FreeSWITCH-CN 微信公众号，参数是 a=" .. a .. " b=" .. b .. "\n")
```

大家已经看到，FreeSWITCH 会将收到的 POST 参数存到env变量里，env是一个标准的 Event 变量，可以用 getHeader 取到参数的值。如果想查看全部的env变量，可以在脚本中加入以下代码：

```
stream:write(env:serialize());
```

你甚至可以 POST XML，如：

```
curl --user freeswitch:works -XPOST -d '<xml><a>100</a><b>200</b></xml>' -H "Content-Type: application/xml" "localhost:4567/execute?language=basic&script=hello.bas"
```

如果你像上面一个使用了stream:write的话，你就知道怎么取得这段 XML 字符串了。

3.20 mod_basic

FreeSWITCH 支持使用 Lua、Javascript、Perl、Java 等嵌入式语言脚本来控制呼叫流程。也有人说，为什么不支持 BASIC 呢？毕竟，BASIC 也是一门语言，而且，它足够简单。

在 2014 年愚人节那天，FreeSWITCH 开发团队宣布 FreeSWITCH 开始支持 BASIC。当然，这并不是一个玩笑，FreeSWITCH 是认真的。

通过使用如下 Diaplan，就可以在有电话呼入时执行一个 BASIC 脚本hello.bas。

```
<extension name="basic">
  <condition field="destination_number" expression="^basic$">
    <action application="basic" data="/tmp/hello.bas"/>
  </condition>
</extension>
```

在 FreeSWITCH 的源代码中也给出了一个示例脚本，内容如下：

```
s$ = "hello "
s$ = s$ + "world"

FS_LOG "WARNING" s$ + "!"

FS_EXECUTE "answer"
FS_EXECUTE "sleep" "1000"
FS_EXECUTE "playback" "misc/misc-cluecon_is_premier_conference.wav"
```

当然，因为它是 BASIC 嘛，所以，很容易理解，以上脚本就是在日志中输出hello world，并执行answer, sleep, playback 等 App。

从该模块的源代码中，可以看出它提供了以下函数：

```
mb_register_func(bi, "FS_EXECUTE", fun_execute);
mb_register_func(bi, "FS_GETARG", fun_getarg);
mb_register_func(bi, "FS_GETVAR", fun_getvar);
mb_register_func(bi, "FS_SETVAR", fun_setvar);
mb_register_func(bi, "FS_API", fun_api);
mb_register_func(bi, "FS_LOG", fun_log);
```

对 BASIC 有兴趣的小伙伴们可以自己试一下，我们就不深入研究了。该模块的发布声明在这里：<http://freeswitch.org/node/480>。

3.21 mod_openh264

使用思科（Cisco）的 OpenH264 库提供编解码支持。参考。

H264 编解码使用需要授权，所以使用起来有一些限制。思科提供了一种解决该问题的思路——如果使用思科提供的动态编码库（虽然 OpenH264 是开源的，但你只能使用其二进制版本），则如果需要交版税的话，思科会帮你交。所以你可以以某种形式分发思科的二进制编码库，而不需要交 H264 版税。

当前，FireFox 以及 Chrome 都使用了 OpenH264 库，不过，Chrome 可能使用的另外的授权方式。

3.22 mod_blacklist

前几天有网友问黑名单怎么做，我说自己写个 Lua 脚本查数据库吧。后来又查了一下，FreeSWITCH 竟然有一个mod_blacklist，这不就是黑名单嘛。

说实话我以前从来没注意过这个模块。遇到一个新模块怎么办呢？当然是查 FreeSWITCH 的 Wiki。那如果 Wiki 上也没有呢？Wiki 上一般都会有，如果没有的话，也不怕，下面，我就带大家在不看 Wiki 的情况下如何了解一个模块。

当然，这个模块比较简单。首先，这个模块默认是不编译的，要手动编译

```
$ cd src/mod/applications/mod_blacklist
$ make install
```

然后，就可以在 FreeSWITCH 里加载模块了：

```
freeswitch> load mod_blacklist
```

显示日志如下：

```
2016-04-28 09:12:29.508192 [CONSOLE] switch_loadable_module.c:1538 Successfully Loaded [mod_blacklist]
2016-04-28 09:12:29.508192 [NOTICE] switch_loadable_module.c:338 Adding API Function 'blacklist'
```

从日志中可以看出，该模块好像仅实现了一个 API: blacklist

直接输入这个命令试下

```
freeswitch@seven.local> blacklist
2016-04-28 16:53:23.913138 [ERR] mod_blacklist.c:182 Invalid usage
```

出错啦，继续试：

```
freeswitch@seven.local> blacklist test
-ERR: No such command: test (see 'blacklist help')
```

有门，继续：

```
freeswitch@seven.local> blacklist help

blacklist check <listname> <item>
blacklist add <listname> <item>
blacklist del <listname> <item>
blacklist save <listname>
blacklist reload
blacklist help
+OK
```

从命令行中看，它好像支持多个列表 (listname)，每个列表中都有相关的条目 (item)。看配置文件，`blacklist.conf.xml` 里有这么一行：

```
<list name="example" filename="$$conf_dir}/blacklists/example.list"/>
```

创建/usr/local/freeswitch/conf/blacklists/example.list

里面的内容怎么写呢？不知道，蒙一把，输入

```
1000  
1001
```

就是把电话号码每行一行。至于蒙的对不对，实际上可以看源代码的。在源代码中，有一个load_list()函数，看样子是用于加载这个列表的，主要内容如下：

```
while (fgets(buf, 1024, f)) {  
    trim(buf);  
    switch_core_hash_insert(bl->list, buf, (void *)SWITCH_TRUE);  
}
```

很明显，它加从文件中依次读取每一行，然后插入一个内容哈希表中。看样子我们蒙的还比较靠谱。

重新加载一下模块：

```
freeswitch@seven.local> reload mod_blacklist  
  
2016-04-28 15:09:18.868148 [INFO] mod_blacklist.c:104 Loaded list [example]  
2016-04-28 15:09:18.868148 [CONSOLE] switch_loadable_module.c:1538 Successfully Loaded [mod_blacklist]
```

哈哈，Loaded list [example] 显示名为 example 的列表已经加载了。试试检查 (check) 命令

```
freeswitch@seven.local> blacklist check example 1000  
  
true  
freeswitch@seven.local> blacklist check example 1001  
  
true  
freeswitch@seven.local> blacklist check example 1002  
  
false
```

由于1002不在列表中，返回了false。

看看能不能手工加

```
freeswitch@seven.local> blacklist add example 1002
```

```
+OK
```

```
2016-04-28 16:54:02.593169 [INFO] mod_blacklist.c:228 Added [1002] to list [example]
```

```
freeswitch@seven.local> blacklist check example 1002
```

```
true
```

这次是true了。试试save命令：

```
freeswitch@seven.local> blacklist save example
```

```
2016-04-28 16:54:17.973176 [INFO] mod_blacklist.c:280 Saving example to /usr/local/freeswitch/conf/blacklists/example.list
```

```
+OK
```

```
$ cat /usr/local/freeswitch/conf/blacklists/example.list
```

```
1000
```

```
1002
```

```
1001
```

可以看到1002已经加到列表里了。但由于内部实现是哈希表，没有排序，因此并没有特定的次序。

好了，但在实际打电话时怎么用呢？

创建如下 Dialplan，这个例子来自 FreeSWITCH 的 Wiki

```
<extension name="blacklist_check">
<condition field="${blacklist(check example ${destination_number})}" expression="^true$">
<action application="bridge" data="sofia/external/sip:lenny@sip.itslenny.com:5060"/>
</condition>
</extension>
```

上面的意思是说，测试条件是一个动态条件，该动态条件是由blacklist check listname item完成的。上面的例子测试了被叫号码 (destination_number) 是否在黑名单，若否则继续呼。

我们再来个稍复杂点的例子：

```

<extension name="echo">
    <condition field="destination_number" expression="^9196$">
        <condition field="\${blacklist(check example ${caller_id_number})}" expression="^true$">
            <action application="answer"/>
            <action application="echo"/>
        </condition>
    </condition>
</extension>

```

其实很简单，我们只不过是在默认的9196基础上增加了一层`blacklist`的判断。如果主叫号码是一个不在名单中的号码，则匹配失败，呼叫失败，打印日志如下：

```

Dialplan: sofia/internal/1004@192.168.7.6 Regex (PASS) [echo] destination_number(9196) =~ /~9196$/ break=on-false
|--- Dialplan: Processing recursive conditions level:1 [echo_recur_1] require-nested=TRUE
|--- Dialplan: sofia/internal/1004@192.168.7.6 Regex (FAIL) [echo_recur_1] \${blacklist(check example ${caller_id_number})

```

如果是在名单中的号码，则呼叫成功，日志显示如下：

```

Dialplan: sofia/internal/1000@192.168.7.6 Regex (PASS) [echo] destination_number(9196) =~ /~9196$/ break=on-false
|--- Dialplan: Processing recursive conditions level:1 [echo_recur_1] require-nested=TRUE
|--- Dialplan: sofia/internal/1000@192.168.7.6 Regex (PASS) [echo_recur_1] \${blacklist(check example ${caller_id_number
|--- Dialplan: sofia/internal/1000@192.168.7.6 Action answer()
|--- Dialplan: sofia/internal/1000@192.168.7.6 Action echo()

```

读到这里，细心的读者可能会问：你上面说错了吧？明明上面的逻辑是白名单……，呵呵，是的，天使跟恶魔只是一念之差，你也可以用黑名单来做白名单;)。当然，如果你确实需要黑名单，试试把上面的`true`改成`false`就行了，这个，留给读者自己练习吧？

第 II 部分 实例应用

第四章 技巧与实例

在本章，我们来讨论一些实例。

4.1 来电转接

来电转接是在企业 PBX 中的一项常用功能。来电转接分为盲转 (Blind Transfer) 和协商转 (Attended Transfer) 两种¹，下在我们分别来介绍一下。

4.1.1 盲转

顾名思意，盲转就是将来电直接转到某一分机。该业务一般用于电话已经接听的情况，举例来说，转接的步骤一般是这样：A 呼叫 B，B 接听，A 与 B 通话，A 要求转 C，B 通过一个操作将来电转接到 C，C 开始振铃，B 挂断，C 接听，A 与 C 通话。

FreeSWITCH 默认的 Dialplan 实现了这种转接，方法是，B 在通话中按『*1』两个键，听拨号音后，拨叫 C 的号码，B 挂机，A 与 C 通话。

这种转接魔术的根本原因在于下面这一行，在默认的 Dialplan 中的 Local_Extensions 中可以找到：

```
<action application="bind_meta_app" data="1 b s execute_extension::dx XML features"/>
```

首先，有电话打入后，如果是最终路由到一个内部分机，就会执行到这一行。至此，来话是 a-leg，现在还没有 b-leg。

bind_meta_app 是一个 App，它会在本次通话上绑定参数中指定的『1』这个按键，这个可以后续可由『*1』两个键激活。注意，1 后面的 b 表示这个功能要绑定到 b-leg 上，虽然 b-leg 现在还没有。但是到了以后，b-leg 可以按『*1』激活该项功能，而不是 a-leg。想想为什么²？

¹也有人分别叫一步转、两步转，或直接转移、出席转移等。

²因为 aleg 可能是外部来话，你肯定不想外面的人乱按按键执行各种功能。

接着说，b 后面的 s 表示什么呢？它表示，如果 b 按键激活该功能后，要在哪条腿上执行这个动作。这里的 s 表示 same，就是说，在哪条腿上接收到按键就在哪条腿上执行。

后面的`execute_extension`表示去 Dialplan 中找一个`extension`去执行一下，具体的，这里指到 XML `features`（它是一个 Context）中去找`dx`这个`extension`。在默认的配置中可以在`features.xml`文件中找到。

总之，这一行执行完毕后只是相当于设了这么一个『套』，后面该怎么执行怎么执行，直到 b-leg 按『*』才能激活该功能。接下来，Dialplan 最终会执行到后面的 `bridge(user/B)`，A 与 B 桥接并开始通话。

通话后，b-leg 就有了，它就是用户 B 所在的那条腿。在通话中，B 按下『*1』，就会执行到『`dx`』那儿了，我们到`features.xml`中去看具体的实现：

```
<extension name="dx">
  <condition field="destination_number" expression="^dx$">
    <action application="answer"/>
    <action application="read" data="11 11 'tone_stream://%(10000,0,350,440)' digits 5000 #"/>
    <action application="execute_extension" data="is_transfer XML features"/>
  </condition>
</extension>
```

这里的`answer`只是保证电话是应答的状态。后面的`read`表示等待用户按键，也就是等待用户输入一个分机号（如 C 的号码）。参数『11 11』表示最少接收 11 位，最大接收 11 位，如果按了『#』（最后一个参数）号，则接停止接收。`tone_stream`会产生一个拨号音，以提示 B 可以拨号了。B 拨完号之后，拨号的数据就存到了`digits`这个通道变量中。电话流程继续往下执行，接着又是`execute_extension`，它的参数`is_transfer`是另一个`extension`，我们可以在同一个文件中继续往下找到，内容如下：

```
<extension name="is_transfer">
  <condition field="destination_number" expression="^is_transfer$"/>
  <condition field="${digits}" expression="^(\d+)$">
    <action application="transfer" data="-bleg ${digits} XML default"/>
    <anti-action application="eval" data="cancel transfer"/>
  </condition>
</extension>
```

可以看到，到了『`is_transfer`』以后，通过正则表达式『`^(\d+)$`』判断收到的数字是不是合法的号码，如果是，则执行`transfer` App 进行转移。

先说上述正则表达式不匹配的情况，即用户 B 输入的不是一个合法的分机号，那么，Dialplan 就会执行到『`anti-action`』一行，取消本次操作。最终结果就是，B 在听了一会拨号音并瞎按了半天后继续与 A 通话。

再说 B 按对了 C 的号码的情况。执行到 transfer App 后。该 App 的参数中有一个『-bleg』，它什么意思呢？实际上，它的意思并不是绝对的 b-leg，而是一个相对的概念，即与当前那条腿桥接的另外一条腿。如果这么说还不明白的话，这样考虑一下：现在是 B 做了这么多操作，因而这些操作都是在原来的 b-leg 上执行的。而 transfer 这里是要把 A 转走，由于当前的是 b-leg，再加上『-bleg』后就相当于『负负得正』，因而意思是把 a-leg 转走。转到哪儿呢？转到『digits』变量所代表的值，即刚在 B 输入的 C 的号码。

这时 C 开始振铃，如果 C 接听，A 就可以与 C 通话了。同时，A 被转走后，B 就没事干了，电话自己就挂掉了。

当然，上面绕了这么一大圈才完成这个转接功能，看起来挺复杂的。实际上，聪明的读者也许已经注意到了，在『dx』那个extension中有这么一句注释：

```
<!-- In call Transfer for phones without a transfer button -->
```

它的意思是，该功能是给那些没有 Transfer 键的话机用的（也就是一般的模拟话机，由于没有 Transfer 按键，因此一般需要靠 DTMF 按键执行这些功能）。对于 SIP 话机而言，通常都会有一个 Transfer 键，一按就转了，很方便。那么这是怎么实现的呢？——答案是 SIP Refer。

Refer 是在 RFC3515³中定义的，它规定的 SIP 中电话转接的几种实现方式，具体的协议流程的实现跟具体话机终端有关，图4.1描述了一种典型的实现方式：

首先 A 与 B 已建立通话，这时候 B 想把 A 转接到 C。这里，B 称为 Transferor，他是转接的发起者；而 A 称为 Transferee，它是被转接的一方；C 称为 Target，是转接的目的地。转接成功后 B 与 C 通话。

B 首先发 re-INVITE 请求给 FS，请求将 B 的电话置为 Hold (保持) 状态，FS 收到请求后就给 A 播放保持音乐。同时，B 的话机放拨号音，以提示用户输入被叫号码。B 输入 C 的号码后，B 给 FS 发 Refer 请求。FS 收到后会释放 B，并同时呼叫 C。如果 C 正常接听，则 A 与 C 通话，转接完成。

4.1.2 协商转

读者读到这里就可能发现一个问题：在上述盲转的情况下，如果 C 长时间不接听（久叫不应），或 C 占线，则转接会失败，A 的电话会被挂断。A 可能会重新呼叫 B 要求转接，这对 A 的体验是很不好的。

FreeSWITCH 通过『att_xfer』App 支持协商转。在默认的拨号计划中也是可以实现的，与盲转不同的是，在进行转移时，B 按『*4』激活『att_xfer』功能，配置如下：

```
<action application="bind_meta_app" data="4 b s execute_extension::att_xfer XML features"/>
```

³参见 www.ietf.org/rfc/rfc3515.txt。

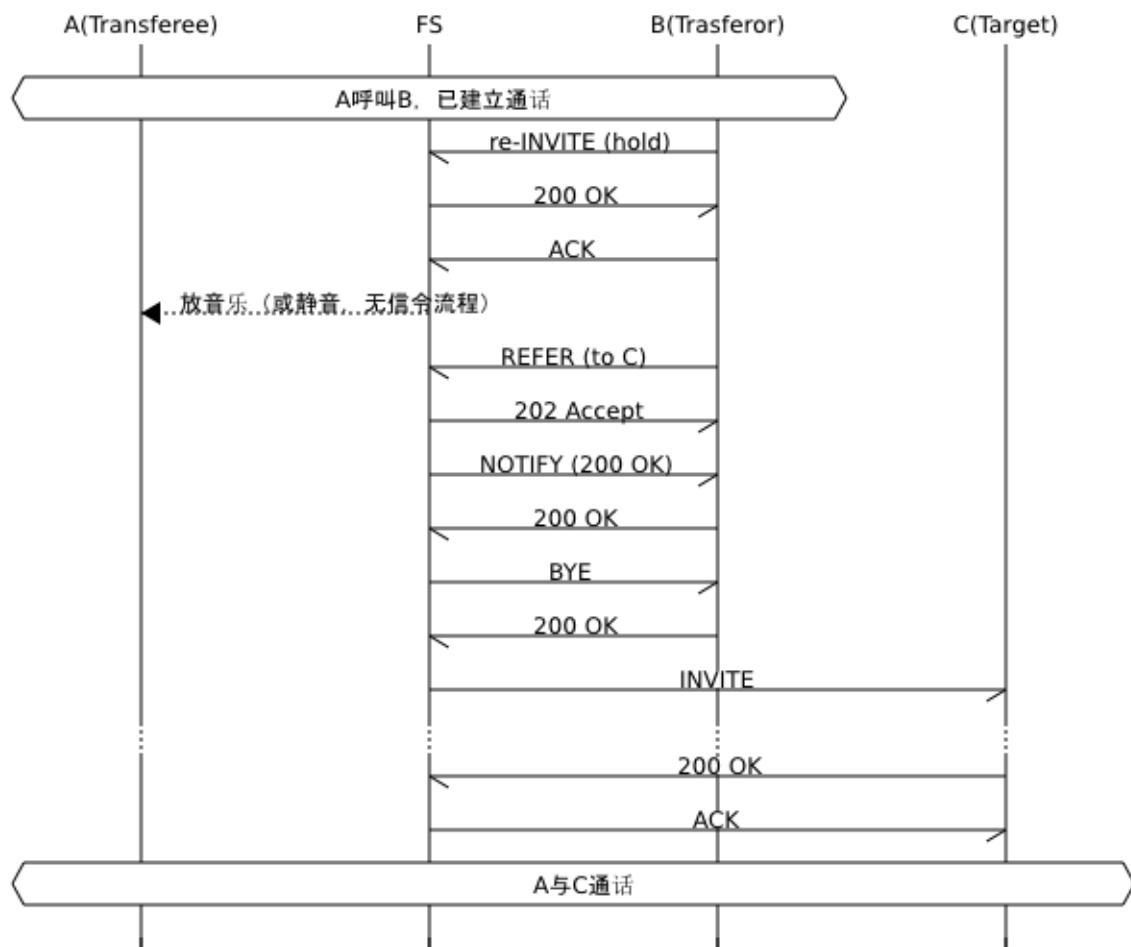


图 4.1: 使用 Refer 进行呼叫转接

同样，我们在 `features.xml` 中找到『`att_xfer`』条件：

```
<condition field="destination_number" expression="^att_xfer$">
    <action application="read" data="3 4 'tone_stream://%(10000,0,350,440)' digits 30000 #"/>
    <action application="set" data="origination_cancel_key="#" />
    <action application="att_xfer" data="user/${digits}@${domain}" />
</condition>
```

它照样使用 `read` 播放拨号音并等待 3-4 位按键，如果输入正确，则使用 `att_xfer` App 处理呼叫转移。`att_xfer` 它会呼叫 C 的号码，同时让 A 听等待音乐。如果呼叫 C 失败，则 B 仍然可以与 A 通话；如果 C 长时间不应答，则 B 可以按『#』（由『`origination_cancel_key`』设置）号键取消呼叫，继续与 B 通话；如果 C 接听后，B 与 C 通话，此时 B 可以询问 C 是否愿意接听电话⁴，如果 C 不愿意，则 C 挂机，B 仍然可以跟 A 通话；如果 C 接受通话，则 B 挂机，A 与 C 通话；如果 B 不挂机，并按 3，则可以形成三方通话，大家一起说。还有更有意思的，B 还可以随时按 1 与 A 通，按 2 与 C 通，而让 A 与 C 永远不通……

上述功能是在 FreeSWITCH 中通过 DTMF 按键实现的。某些话机支持多路通话，因而可以在话机端实现协商转。典型地，话机终端 B 可以把第一路电话置于 Hold 状态，然后再发起另外一路通话到 C，C 接听后 B 可以任意切换与 A 和 C 之间的通话，并可以通过本地会议桥进行混音以支持三方通话（也叫会议）。

此时 B 如果想退出 A 与 C 的通话，则可以发送 REFER 消息，让服务器把通话中的 B 替换为 C。该消息与盲转不同的是，它带了 Replaces 参数，如下：

```
Refer-To: sip:1002@192.168.1.118?Replaces=1388923627@192.168.1.110;to-tag=NDj261X80jpKF;from-tag=1013380895>
```

为了阅读方便，上面的消息是经过『`urldecode`』后的，实际的消息内容是用『`urlencode`』编码的字符串。它是这样产生的：A (1000) 呼叫 B (1004)，此处 B 是亿联话机，B 接听后，按下话机上的 Conf 软键（代表 Conference，会议）然后呼叫 C (1002)。C 接听后，B 与 C 通话。B 可以通过话机上的 Swap 按钮切换他与 A 或与 C 通话，也可以再次按 Conf 键启用本地混音实现 A、B、C 三方的电话会议。与我们上面讲的例子不同，上面的例子三方通话是由『`att_xfer`』实现的，参与方只有三方，即三个 Channel；而本例中是 4 个 Channel，因为 B 话机同时有两个 Channel 连接到 FreeSWITCH。

如果 B 想退出会议，但保持 A 与 C 的通话，则 B 发送 REFER 消息让 FreeSWITCH 把它替换掉，即让 FreeSWITCH 把 A 与 C Bridge 起来，翻译掉与 B 的两路通话。

完整的 REFER 消息如下：

⁴B 与 C 协商，这是也协商转的来源。

```

-----
recv 594 bytes from udp/[192.168.1.110]:5063 at 05:59:43.368543:
-----
REFER sip:mod_sofia@192.168.1.118:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.110:5063;branch=z9hG4bK1882546254
From: <sip:1004@192.168.1.110:5063>;tag=485693841
To: "Extension 1000" <sip:1000@192.168.1.118>;tag=H9cZZNttcFX8g
Call-ID: affaa293-566c-1231-e2ba-79a3b2b753b0
CSeq: 45647480 REFER
Contact: <sip:1004@192.168.1.110:5063>
Max-Forwards: 70
User-Agent: Yealink SIP-T22P 7.70.0.140
Refer-To: <sip:1002@192.168.1.118?Replaces=1388923627%40192.168.1.110%3Bto-tag%3DNDj261X80jpKF%3Bfrom-tag%3D1013380895
Referred-By: <sip:1004@192.168.1.110:5063>
Event: refer
Content-Length: 0

```

有兴趣的读者可以自行跟踪 SIP 消息看一下，详细的流程（及流程图）就不赘述了。

4.2 共享线路呈现

共享线路呈现 (SLA, Shared Lines Appearance) 也是企业应用中一个非常有用的功能。该功能在模拟话机中是没有的，只有 SIP 话机才能实现该功能。使用该功能可以在自己的话机上监视其他话机的状态，从而知道另一个电话是否处于忙或闲的状态。比方说在老板-秘书的场景中，如果有人打秘书的电话想找老板，而秘书转电话时碰巧老板的电话正在占线，就会导致转接不成功，耽误时间。而如果秘书事先知道老板的电话是否在忙，就可以直接判断是否要将电话转给老板，或者告诉主叫用户先等一会。

FreeSWITCH 支持这种功能。在使用时确认 Profile 中的设置开启了以下两项：

```

<param name="manage-presence" value="true"/>
<param name="manage-shared-appearance" value="true"/>

```

下面的例子以 Seven Du (607) 和 Sonic Gao (608) 为例。607 使用 X-Lite 注册。注册完毕后通过菜单项 【Contacts】->【Add Contact】添加一个联系人 608，并确保开启了 Presence 功能，如图4.2所示：

添加完成后，主界面上就能显示联系人的状态，如图4.3(左) 所示，现在 608 处于注册状态，显示是绿色的 Available 状态。如果 608 正在通话，则会变为红色的 Talk 状态，图4.3中右侧显示 608 (Sonic Gao) 正在跟 9196 通话。

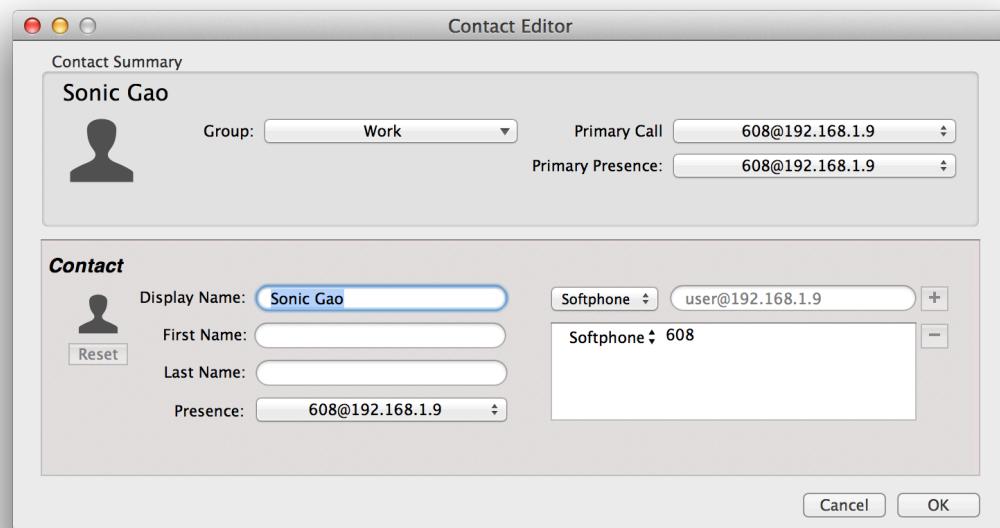


图 4.2: 在 XLite 中添加 Presence 联系人

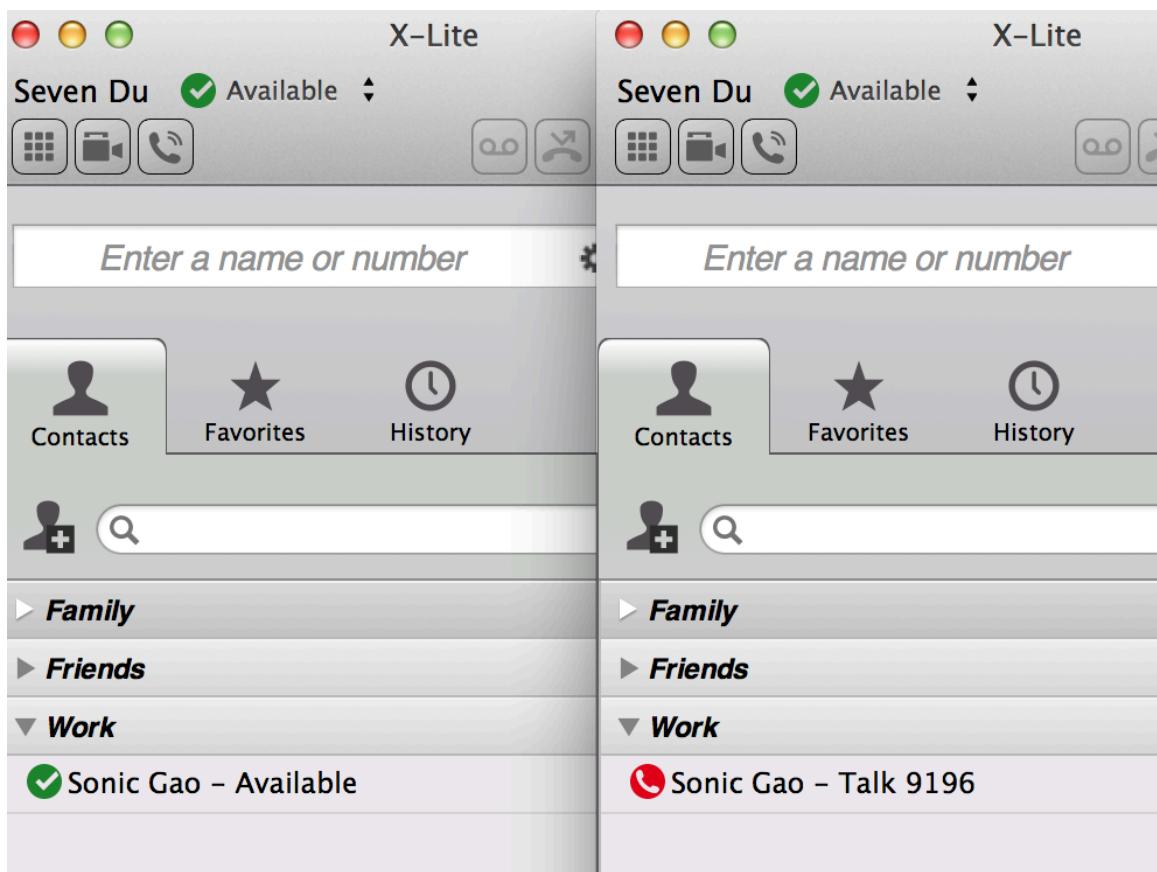


图 4.3: XLite 上的联系人状态

账号608是从一个亿联话机上注册的，该话机也支持线路共享功能。打开话机的 Web 设置界面，依次顺着菜单走到『话机设置』->『可编程按键』->『账号键』。如图4.4。



图 4.4: 在亿联话机上配置 SLA

在图4.4中所示的页面上，选择账号键 1，把类型由『线路』改为 SLA，『值』里输入你想监视的话机账号，这里是『sip:607@192.168.1.9』，注册『线路』一栏也选择账号 1，保存设置后，话机上账号 1 对应的灯就会常亮，表示 607 是正常空闲的。如果 607 正在通话，该灯就会闪烁。如图4.5所示，话机上第一个灯是常亮或闪烁。

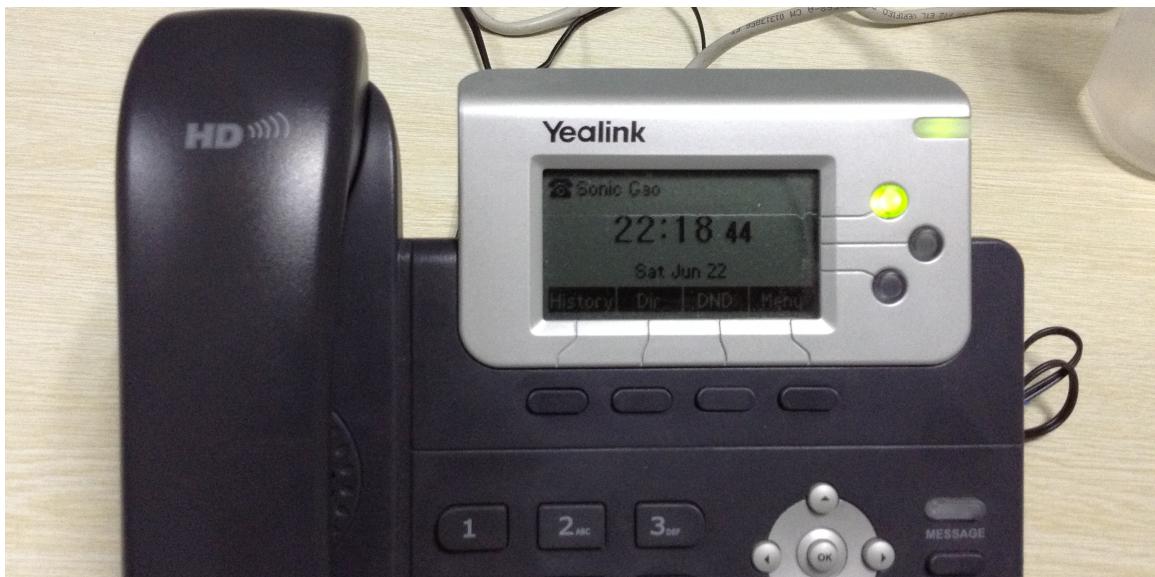


图 4.5: 话机『灯的状态』

4.3 使用组播功能做网络广播

在有些特殊的场景下，拿起电话拨打一个号码，就可以对一大群人喊话，广播功能可以大大提高工作效率。

广播功能可以使用会议实现，简单地发起 N 路通话加入一个会议也可以做到广播的效果。不过，那样实现要建立 N 路通话，需要消耗很多的网络资源；另外，也无法保证对方能及时接听，影响信息的送达。实现该业务模式最经济的方式就是使用组播（Multicast，或称多播）。组播只向组播地址发送一个 RTP 流，而监听该组播地址的所有主机就都能收到。

FreeSWITCH 有一个 `mod_esf` (Extra SIP Functionality) 模块，它提供一个 `esf_page_group` App 可以支持组播。

`esf_page_group` 有三个参数，分别是：

- 组播地址，默认为 224.168.168.168
- 端口号，默认为 35467
- 控制端口号，默认为 6001

要能收到组播包并播放声音，也需要配置亿联话机以启用这项功能⁵。配置界面如图4.6所示。



图 4.6: 配置亿联话机接受组播包

该配置项比较难找，依次找到菜单『电话簿』->『组播』，在第一个 IP 地址框里填入224.168.168.168:34567，标签栏可以任意填，起一个好记的名字就行。

在 FreeSWITCH 默认的配置中，拨打号码7243就直接向该地址发送组播，所以，现在你可以拿电话拨打7243试一下了。

默认的 Dialplan 配置如下：

```
<extension name="rtp_multicast_page">
<condition field="destination_number" expression="^pagegroup$|^7243$">
    <action application="answer"/>
    <action application="esf_page_group"/>
```

⁵笔者不知道其他话机是否具有这样的功能，最初亿联也没有，是笔者帮助 FreeSWITCH 的作者 Anthony Minassale 联系到亿联的工程师后他们才实现的。

```
</condition>
</extension>
```

如果你想发到其他的地址，可以配置相关参数，如下列配置可以将 RTP 包发到组播地址 224.0.0.100：

```
<action application="esf_page_group" data="224.0.0.100 34567 6001"/>
```

与普通的 IP 地址不同，组播需要配置组播地址。众所周知，在 IPv4 中，组播地址的范围是从 224.0.0.0 到 239.255.255.255，由于实际用到组播的业务却很少，因而好多人可能不是很熟悉。在有的系统上需要配置组播路由，如以下命令可以在 Linux 系统的 eth0 上配置组播路由：

```
ip route add 224.0.0.0/4 dev eth0 src 192.168.5.2
```

使用上述命令还需要 iproute2 软件包支持。另外，在实际应用中为避免组播风暴及潜在的冲突，可能需要把具有组播功能的话机都划分到一个 VLAN 上。再者，组播包一般不能穿越路由器，如果要跨路由器组播的话，需要支持组播的路由器，并进行适当的配置，这些都超出了本书的范围，有兴趣的读者可以自行参考相关资料。

4.4 使用 FreeSWITCH 检测声音文件中的 DTMF 信息

曾经，有网友问到一个问题——使用什么工具检测录音文件中的 DTMF 信息。其实 FreeSWITCH 本身就具备检测 DTMF 的功能，简单配置一下，写几个脚本就可以了。

先简单说一下 DTMF，DTMF 是 Double Tone Multiple Frequency 的缩写，即双音多频。在电话通话中，通过两个不同的频率的组合来传递按键信息，如题图中所显示的，1209 和 697 两种频率的组合就代表 1，其它依此类推。

在模拟电话以及传统的 PSTN 中，DTMF 与声音数据是混在一起的，因为它们根本没法分开。在 VoIP 中常常使用 DTMF2833 或 SIP INFO 来传输 DTMF，但那不是我们今天要讲的内容。

由于 DTMF 与声音都混在话路中，在录音时也就一块将 DTMF 信息录在了录音文件中，如果想从录音文件中提取这些 DTMF 信息，就需要对声音文件进行分析，也就是今天我们要解决的问题。

我们有了 FreeSWITCH，当然不需要去找别的工具，下面我们就来看一看怎么做。

为了做一次完整的实验，我们先得有个录音文件。首先把 SIP 电话设成使用 inband 方式发送 DTMF，以便能够录到 DTMF 信息，具体的设置方式因不同的话机（或软电话）而不同，我们就不多说了。然后，使用如下方法我们可以得到一个录音文件：

```
freeswitch> originate user/1008 &record(/tmp/dtmf.wav)
```

上面使用 originate 命令呼叫 1008，被叫接听后，开始录音。记得接听后要按几个键啊。在本次实验中，我按了 1234，并挂机。

挂机后找个工具播放一下 dtmf.wav，便能听到滴滴的按键音，虽然每个按键的声音不一样，但我们的耳朵认不出来，还得借助软件。

我们昨天刚讲了 Lua，今天正好进一步再来一个例子，因而我们写了一个 Lua 脚本来检测 DTMF，命名为 dtmf.lua，内容如下：

```
function onInputCBF(s, type, obj, arg)
    if (type == "dtmf") then
        freeswitch.consoleLog("INFO", "Got DTMF: " .. obj.digit .. " Duration: " .. obj.duration .. "\n")
    end
    return ''
end

session:answer()
session:execute("start_dtmf", "")
session:setInputCallback('onInputCBF', '')
session:streamFile("local_stream://moh』")
```

其中，我们设了一个回调函数 onInputCBF，当检测到 DTMF 时便进行回调，在日志中打印相关的 DTMF 信息。

- `session:answer()` 对 Channel 进行应答
- `session:execute()` 执行一个 App，这里我们执行了 `start_dtmf` 以启动对 inband 类型的 DTMF 的检测
- `session:setInputCallback()` 安装一个回调函数，在检测到 DTMF 时便执行该回调函数，就是我们上面写的那个 `onInputCBF`
- `session:streamFile()` 一行只是播放一个无限长的声音文件，防止挂机

通过该 Lua 脚本，当有电话呼入时，我们将来电路由到该脚本，便可以实时检测来电中的 DTMF 了。但是在这里我们有一个问题，那就是我们要检测的是录音文件里面的，它不是一路电话，即不是一个 Channel。

当然，这也难不住我们，既然我们有 FreeSWITCH，那我们可以弄两个 FreeSWITCH 实例，从一个中呼叫另一个，在其中一个执行 playback 以播放声音文件，另一个执行上面的 Lua 脚本检测，问题不就解决了？

是的，但我们还有更简单的解决办法。

在 FreeSWITCH 中，不管是播放声音文件还是检测 DTMF 都需要一个 Channel，在没有实际 Channel 的情况下，我们就可以生成一个假的 Channel。对于这一点，FreeSWITCH 早就帮我们想到了，那就是 loopback Interface。它其实也是一个 Endpoint，通过下面的命令生成一个 Channel，并执行我们的 Lua 脚本：

```
freeswitch> originate loopback/dtmf &lua(dtmf.lua)
```

其中，loopback/ 后面的 dtmf 是被叫号码，当一个 Channel 产生后，该 Channel 的一端（一头）会进入 Dialplan 查找路由，另一头则执行 lua App，即执行我们的 Lua 脚本。关于 loopback 我们就不多解释了，我们只需要知道它在查找 Dialplan 时需要在 Dialplan 中让它能找到，因而，我们在默认的 Dialplan 中(default.xml)加入以下内容：

```
<extension name="dtmf">
<condition field="destination_number" expression="dtmf">
<action application="answer" data="" />
<action application="playback" data="/tmp/dtmf.wav" />
</condition>
</extension>
```

上述 Dialplan 会匹配被叫号码 dtmf，然后应答，然后播放一个声音文件，就是我们刚才录的那一个。

在 Channel 的另一头执行我们的 Lua 脚本，就可以检测 DTMF 了，笔者测试时，日志输出如下：

```
[INFO] switch_cpp.cpp:1291 Got DTMF: 1 Duration: 1120
[INFO] switch_cpp.cpp:1291 Got DTMF: 2 Duration: 1120
[INFO] switch_cpp.cpp:1291 Got DTMF: 3 Duration: 1120
[INFO] switch_cpp.cpp:1291 Got DTMF: 4 Duration: 1120
```

帅不帅？

当然，以上我们的 Lua 脚本比较简单，通过增加一些语句，你也可以比较精确的打印 DTMF 在录音文件中的时间等信息，这些，自己练习一下吧。

4.5 号码连选

虽然有些运营商也开始尝试开放 SIP 号码，但大部分还是谨慎的，试探性的。因此一般不提供 SIP 对开中继的方式，而是开放单个的接入号码。虽然这不是理想的方式，但有总比没有好。在此，假设我们从运营商那里获得 10 个 SIP 账号，号码范围是xxxxxx30~xxxxxx39。我们来看一个如何有效地使用这些号码。

4.5.1 注册到运营商服务器

我们可以在 FreeSWITCH 中添加一些网关，以便注册到运营商的 SIP 服务器上去（应该是一个 SBC）。

网关的配置文件如下，为了使用方便，我们让网关名称（name）的后两位与号码的最后两位相同：

```
<gateway name="gw30">
    <param name="realm" value="218.56.x.x"/>
    <param name="username" value="xxxxxx30"/>
    <param name="password" value="xxxx"/>
    <param name="register" value="true"/>
</gateway>

<gateway name="yt31">
    <param name="realm" value="218.56.x.x"/>
    <param name="username" value="xxxxxx31"/>
    <param name="password" value="xxxx"/>
    <param name="register" value="true"/>
</gateway>
```

.....

上面我们仅列出了两个网关账号的配置，其它账号以此类推。注册成功后，我们就可以通过这线号码（又称为线路）打入打出电话了。

4.5.2 通过单个号码呼出

我们可以使用如下命令快速的测试是否能成功呼出：

```
freeswitch> originate sofia/gateway/gw30/1860535xxxx &echo
```

当然，也可以设置如下的 Dialplan 让所有分机号都可以通过一个网关呼出：

```
<extension name="Outbound Call">
<condition field="destination_number" expression="^(1[358].*)$">
<action application="bridge" data="sofia/gateway/gw30/$1"/>
</condition>
</extension>
```

注意，简单起见，我们本例中使用的正则表达式仅匹配手机号。上述配置将统一使用网关gw30（即通过号码xxxxxx30）呼出。很容易想到，如果有第二个人同时进行呼叫时，由于该网关对应的号码占线，因此，呼叫失败。

4.5.3 使用随机数做号码连选

为了能自动选择一个网关呼出，我们想办法从这 10 个网关中自动选择一个进行呼叫出。这种选择的过程就称为选线，也称为号码连选。当然，号码连选最简单的实现方法是使用一个随机数。见下面的 Dialplan：

```
<action application="set" data="gw=gw${expr(randomize(&x);ceil(random(30,39,&x)))}" />
<action application="bridge" data="sofia/gateway/${gw}/$1"/>
```

其中，『expr』是一个 API，我们用它的『randomize』方法产生一个从 30 到 39 之间的随机数，在该随机数前面加上『gw』字符，并把它赋值给一个『gw』通道变量（使用『set』）实现。有了该通道变量后，在『bridge』的参数中就可以使用『\${gw}』引用该变量，实现动态选择一个随机的网关。

当然，这种选线算法有一个缺点，就是它不记录实际号码的『忙闲』状态，因而在选到正在通话的号码时，通话会失败。通过下面的方式，我们可以做一个改进的算法：

```
<action application="set"
data="gw1=gw${expr(randomize(&x);ceil(random(30,39,&x)))}" />
<action application="set"
data="gw2=gw${expr(randomize(&x);ceil(random(30,39,&x)))}" />
<action application="bridge"
data="sofia/gateway/${gw1}/$1|sofia/gateway/${gw2}/$1"/>
```

该方法的思路是，同时选择两个网关（『gw1』和『gw2』），如果一个失败，则走另一个。

4.5.4 使用 mod_distributor 进行连选

首先，安装mod_distributor，进入 FreeSWITCH 的源代码目录，执行：

```
make mod_distributor-install
```

然后，在 conf/autoload_configs/distributor.conf.xml 中进行如下设置：

```
<list name="dist1" total-weight="10">
<node name="30" weight="1"/>
<node name="31" weight="1"/>
<node name="32" weight="1"/>
<node name="33" weight="1"/>
<node name="34" weight="1"/>
<node name="35" weight="1"/>
<node name="36" weight="1"/>
<node name="37" weight="1"/>
<node name="38" weight="1"/>
<node name="39" weight="1"/>
</list>
```

其中，我们配置了一个列表（『list』），它的名字是『dist1』，所有的权重（『total-weight』）是『10』。该列表有好多节点（『node』）组成，其中每个节点的权重（『weight』）为『1』⁶。可以看出，这些节点的名字跟我们线路号码的最后两位相同。

FreeSWITCH 加载该模块后，我们就可以先用如下的命令进行一下测试了：

```
freeswitch> distributor dist1
32
freeswitch> distributor dist1
35
```

『distributor』是该模块提供的一个 API 命令，它可用于从预定义的列表中根据权重选择一个节点，并返回该节点的名称。然后，我们就可以在 Dialplan 中使用它来帮助我们连线了：

```
<extension name="gw">
<condition field="destination_number" expression="^(01[358].*)$">
<action application="bridge"
```

⁶如果某条线路支持多线呼出，我们也可以增加它的权重

```

    data="sofia/gateway/gw${distributor(dist1)}/$1" loop="2"/>
</condition>
</extension>

```

其中，我们使用『\${distributor(dist1)}』让『mod_distributor』帮我们选择一个节点，并使用它作为网关的名字向外呼出。当然，我们上一节的使用随机数的方案相比，它也聪明不了许多，因为，该模块也没有记录哪条线路是空闲还是忙的。因此，我们使用了 Dialplan Action 中的『loop』属性，如果第一次呼叫失败，它将再试一次。

当然，与上一节的随机数方案比起来，它还是聪明一点的。例如，虽然我们配置了 10 个网关，但并不一定所有时间所有的网关都能正常注册上。通过如下方法，就可以让『distributor』在生成选择节点时，排除掉处于『down』状态（即不可用状态）的网关：

```

<action application="bridge"
  data="sofia/gateway/gw${distributor(dist1 ${sofia(profile internal gwlist down)})}/$1"/>

```

4.5.5 其它

上述的几种方案在一般话务量不是很高的情况下，应该问题不大。不过在很繁忙的场合（如线路占用率 90% 以上），可能就会出现较高的呼损（即呼叫失败，我们这里更多的是指在还有空闲线路的情况下呼叫失败）。因此，为了提高线路的利用率，降低呼损，我们需要更好的算法。

当然，没有现成的方案来解决该问题，这里，我们仅探讨一下实现的思路。首先，实现该算法的时候，应该能检测并记住线路的状态，避免选到坏线（如网关注册失败的线路）或忙线；其次，应该有冲突解决方案，即能够解决同抢的情况（如，刚刚选择了一个空间闲的线路，但在呼出前的瞬间忽然来了一通呼入的电话）。

当然，实现这种算法比较复杂，而且，如果实现不好的话，也不一定能减少呼损。所以有时候，多开一些网关线路，利用减少线路利用率来换取更低的呼损，选择一个折中的值还是可以接受的。

另外，在这些网关线路也会有大量呼入的情况下，呼损率肯定会更高。这时候，可以采取将呼入呼出分开的方法，即一部分线路只允许呼入，另一部分仅用于呼出（最好能在运营商的交换机上做限制，让该号码永远呼叫不进来）。具体两部分的比例应该根据实际情况设置，当然使用这种办法也会更进一步降低线路利用率。

4.6 收发传真

无论社会如何发展，老技术永远有它特殊的生命力，生生不息。传真技术也是一样。在互联网技术飞速发展的今天，电子邮件、即时通信、以及基于移动互联网的各种应用如微博、微信等铺天盖地

地发展，但我们在企业应用中，还是离不开传真机。当然，现代的传真机大多数都集打印、扫描等功能于一体了，但它还是使用 TDM 通信的方式收发传真。

为了将这些老旧的技术及设备融入 SIP 世界里，很多人也进行了各种努力，因而我们也可以非常方便的使用它。连接 TDM 传真机最简单的方式是使用一个模拟转 SIP 的网关，将它变成 SIP 后在 FreeSWITCH 中收发传真就变得容易了。如我们可以使用如下命令，呼叫一个号码，并开始收传真。

```
freeswitch> originate sofia/gateway/gw1/xxxx &rxfax(/tmp/test.tiff)
```

传真功能是在『mod_spandsp』⁷中实现的。它实现了一个『rxfax』App 用于收传真。普通的传真机是这样工作的，A 呼叫 B，首先建立正常的通话，电话打通以后 A 告诉 B 他想发一个传真，因此 B 会按下一个按钮将传真机切换到传真模式，这时 A 这一端将听到『吱吱』的传真音，然后 A 端按下传真机上的『发送键』（或传真开始键）发送传真。

有 SIP 参与的传真过程也是类似的，它们首先先建立正常的 SIP 连接，然后如果一方想发传真或收传真，则它的终端就会给另一方发送 SIP 『re-INVITE』消息，与对方协商将 RTP 媒体流切换到 T.38（或 T.30）传真图像模式，协商完成后开始发送传真。

上面我们讲到的『rxfax』是收传真的 App，它会将接收到的传真存到本地的一个 TIFF⁸格式的文件中。

发送传真与与此类似，只是需要预先将欲发送的内容转成 TIFF 格式的文件。如下命令可以呼叫一个号码并发送传真：

```
freeswitch> originate sofia/gateway/gw1/xxxx &txfax(/tmp/test.tiff)
```

可以使用 Gostscript 或 Imagemagick 图像处理工具将欲发送的传真内容从原来的格式（如 PDF）转换成 TIFF 格式。如，下列gs（即 Gostscript）命令可以将fax.pdf转换成fax.tiff：

```
gs -q -r204x98 -g1728x1078 -dNOPAUSE -dBATCH -dSAFER -sDEVICE=tiffg3 -sOutputFile=fax.tiff -- fax.pdf
```

注意，其中『204x98』表示分辨率。传真文件的分辨率就是比较奇怪——横向和纵向的分辨率是不同的。另外，传真页面也需要有特定的大小，大部分传真机都能接收页面大小为『1728x1078』的传真，其它尺寸的因传真机而异。

⁷基于 spandsp 库，当然除了传真之外它还实现了一些语音编解码。参考https://freeswitch.org/confluence/display/FREESWITCH/mod_spandsp。

⁸传真常用的格式，与常见的 JPEG 或 PNG 格式不同，但类似于 PDF，它主要的特点是支持多页，而这一点在传真文件中非常重要。

上面，我们只讨论了收发两端均是 SIP 的情况。如果在 FreeSWITCH 内部也使用如『`ftdm`』这样的 Endpoint（配合模拟或数字板卡），那么 FreeSWITCH 也可以通过『`t38gateway`』App 进行传真媒体的转换。关于这些我们在此就不多讲了，有需要的读者可以参阅相关的 wiki 页面。

最后，我们再来看一下默认的 Dialplan 的配置。配置的 Dialplan 配置了两个 Extension，『`9178`』用于收传真，『`9179`』用于发传真。内容如下：

```

<extension name="fax_receive">
    <condition field="destination_number" expression="^9178$">
        <action application="answer" />
        <action application="playback" data="silence_stream://2000"/>
        <action application="rxfax" data="/tmp/rxfax.tif"/>
        <action application="hangup"/>
    </condition>
</extension>

<extension name="fax_transmit">
    <condition field="destination_number" expression="^9179$">
        <action application="txfax" data="/tmp/txfax.tif"/>
        <action application="hangup"/>
    </condition>
</extension>

```

读者可以自行拨打测试一下。在此，我们就不多讲了。

4.7 使用 loopback Endpoint 外呼

如果要在 FreeSWITCH 中向外发起呼叫，需要一个呼叫字符串。在前面的章节中，我们学习过几种呼叫字符串，如『`user/1000`』、『`sofia/gateway/gw1/xx`』、『`ftdm/1/a/1000`』等。

但有些情况下，找到呼叫字符串不是那么容易。比方说，我们做了一套企业应用，该企业比较大，有很多分支机构、而且由于历史原因，号码分配比较乱，因此，我们需要在 Dialplan 中设置了几个 Extension，以匹配不同号码段正行正确的路由。如下：

```

<condition field="destination_number" expression="^0(.*)$">
<condition field="destination_number" expression="^11(2.*)$">
<condition field="destination_number" expression="^1234(852.*)$">
<condition field="destination_number" expression="^1234(52.*)$">
<condition field="destination_number" expression="^1234(.*)$">
.....

```

如果在我们的应用中，在某些场合下（如自动传真、催缴费等）需要在 FreeSWITCH 中主动呼叫这些号码。我们可以很容易地使用『originate』进行外呼，但在外呼时根据不同的号段我们需要生成不同的呼叫字符串。这样，我们的程序中就需要另外一套路由数据，不仅与 Dialplan 中的数据重复，而且不利维护。因此，我们就考虑，能否在用『originate』外呼的时候也使用 Dialplan 呢？

答案是肯定的，这里，我们使用一个新的 Endpoint，称为『loopback』。在理解它的实用原理之前，我们先来看一个简单的例子。

我们知道，路由到『9664』后会播放保持音乐，而路由到『9196』后会执行『echo』。以前的时候，只有有了一个 Channel 以后才能进入 Dialplan，并路由到『9664』，而现在，我们可以使用『loopback』提前进入 Dialplan 进行路由。如下面的命令：

```
freeswitch> originate loopback/9664 &bridge(loopback/9196)
```

这些 Channel 都是在 FreeSWITCH 内部发生的，运行完毕后可以使用下列命令看到，上述命令建立了 4 个 Channel（由于输出行太长，我们简单了输出，仅列出有代表性的字段）：

```
freeswitch> show channels
```



```
22937c99-7710-4f64-869f-4f441cc15b54, inbound, loopback/9664-b ...
66f83997-5115-43e9-b1b1-b786ac64a9d2, outbound, loopback/9664-a ...
d91f7824-b314-4996-acc4-e27fb846e66a, outbound, loopback/9196-a ...
57416b98-702b-4d38-bc1b-084966fd04e3, inbound, loopback/9196-b ...
```

那么为什么它有 4 个 Channel 呢，我们知道在普通情况下，我们如果执行如下呼叫，将只产生两个 Channel：

```
freeswitch> originate user/1000 &bridge(user/1001)
```

『loopback』Endpoint 的关键就在这里，当然它的实现原理很简单但讲起来却很复杂。为了能让呼叫先到达 Dialplan 查找路由，它首先创建一个假的腿（即 Channel，称为『loopback-a，我们称为第一条腿』），然后让该腿进入 Dialplan 进行路由，执行『playback』播放保持音乐，然后再创建另一条腿（称为『loopback-b』，我们称为第二条腿），以继续下一步的动作。在此，它执行『bridge』。我们在『bridge』中也使用了『loopback』，因而会发生同样的事情，先创建一条假腿（另一个『loopback-a』，第三条腿）到 Dialplan 中找到适当的路由，执行『echo』，然后又创建一条腿（第四条腿）再与原先的那个腿『bridge』起来。如果还是不理解的话，可以查看一下刚才呼叫的日志，就能比较清楚一些：

```
[NOTICE] switch_channel.c:1044 New Channel loopback/9664-a [66f83997-...]
[NOTICE] switch_channel.c:1042 Rename Channel loopback/9664-a->loopback/9664-a
[NOTICE] switch_channel.c:1044 New Channel loopback/9664-b [22937c99-...]
[NOTICE] mod_loopback.c:946 Channel [loopback/9664-a] has been answered
[NOTICE] mod_dptools.c:1225 Channel [loopback/9664-b] has been answered

[NOTICE] switch_channel.c:1044 New Channel loopback/9196-a [d91f7824-...]
[NOTICE] switch_channel.c:1042 Rename Channel loopback/9196-a->loopback/9196-a
[NOTICE] switch_channel.c:1044 New Channel loopback/9196-b [57416b98-...]
[NOTICE] mod_loopback.c:946 Channel [loopback/9196-a] has been answered
[NOTICE] mod_dptools.c:1225 Channel [loopback/9196-b] has been answered
```

从日志中可以看到，一共有 4 条腿（4 个 Channel）参与到该通话中，它们的名字分别是『loopback/9664-a』、『loopback/9664-b』、『loopback/9196-a』、『loopback/9196-b』。

当然，你也可以执行以下的例子，它将先呼叫 1000，再呼叫 1002，仍然会产生 4 条腿：

```
freeswitch> originate loopback/1000 &bridge(loopback/1001)
```

下面的命令的效果是一样的，但只会产生 3 条腿。

```
freeswitch> originate loopback/1000 1001
```

这是因为，『1001』那一端没有使用『loopback』，在『1001』那一端产生了两条腿以后，呼叫进入 Dialplan 进行路由，然后又『bridge』到『1001』，产生了第三条腿。

理解了这些之后，就可以对我们在本节最开始的时候提出的那些号码进行呼叫了，而不用管它们是具体怎么路由的，如，呼叫某个号码并自动发送传真：

```
freeswitch> originate loopback/123485234xxx &txfax(/tmp/test.tiff)
```

需要注意的是，由于使用 loopback 将比普通的呼叫多产生一条腿，因而话单可能受影响。为此，所有的『loopback』Channel 中都设置了一个『is_loopback』变量，以方便其它程序（如计费等）进行后续处理。

为了最大限度地减少它产生的额外的腿的影响，也可以通过一个通道变量让这条额外的腿在完成它的使命之后尽快释放（或者说两条腿合并为一条），如下：

```
freeswitch originate {loopback_bowout=true}loopback/1000 &echo
```

上述命令将先建立两条腿，呼叫『1000』，在成功执行『echo』时，两条腿将合并为一条。读者可以使用『show Channels』命令查看一下。

最后，我们本节花了很大篇幅给大家讲解『loopback』，除了它在某些场合确实有用外，我们还为了给大家讲清楚默认 Dialplan 中的『Local_Extension』部分的路由。在第 6.1.9 节中，我们为了方便讲解，把默认 Dialplan 中呼叫失败后自动进入 Voicemail 的 Dialplan Action 改写成了如下的形式：

```
<action application="voicemail" data="default ${domain_name} ${dialed_extension}" />
```

但实际上，它是如下的形式：

```
<action application="bridge" data="loopback/app=voicemail:default ${domain_name} ${dialed_extension}" />
```

后面这种方式使用了『loopback』。在大多数情况下，两者的效果是一样的。只不过，后者因为能在进入 Voicemail 后能产生一个 Channel，因而还可以配合『att_xfer』一起使用。

在这里，与前面讲的『loopback』会先进入 Dialplan 进行路由不同的是，它直接使用『app=』这样的语法直接执行一个 App，相当于『loopback』Dialplan 的 inline 形式。

4.8 在 Web 浏览器中打电话

多年来，随着互联网的发展，能在网页上『打电话』是人们的一个梦想。当然，这些年来，人们也实现了一些方案，这些方案一般都是基于浏览器的插件实现的，如基于 IE 浏览器的 ActiveX 插件、Java Applet 插件以及 Flash 插件等。其中，基于 Flash 插件的产品和解决方案由于其跨平台性和兼容性比较好，使用比较广泛，比较著名的 Red5 Media Server⁹以及 Wowza Media Server¹⁰等。

然而，随着 HTML5 技术的发展以及几年前苹果公司声明在 iOS 设备中不再支持 Flash，Flash 的使用量开始下滑，并且普通认为前景不好。而此时，谷歌提出的基于 HTML5 的 WebRTC 实时通信技术却受到了大家的关注，各在线通信平台也纷纷推出支持 WebRTC 的实时通信方案。

下看，我们来看一下 FreeSWITCH 对 Flash 和 WebRTC 的支持是如何实现的。

⁹参见<http://www.red5.org/>。

¹⁰参见<http://www.wowza.com/>。

4.8.1 Flash

基于 Flash 的实时多媒体通信是基于 Adobe 的 RTMP 协议进行的。FreeSWITCH 中通过『mod_rtmp』实现了一个基于 RTMP 协议的 Endpoint，可以支持用 Flash 实现的软电话。如果我们经常讲的一样，虽然 Flash 前景不再被看好，但是在一定范围内它还将顽强的存在。而且，作为有别于 SIP 模块（『mod_sofia』）的另外一个 Endpoint，也很有参考和借鉴意义。

在 FreeSWITCH 源代码目录中使用如下命令即可安装该模块：

```
# make mod_rtmp-install
```

在 FreeSWITCH 控制台上使用『load mod_rtmp』命令加载该模块后，它将监听 RTMP 协议默认的 1935 端口，并等待客户端连接，使用如下命令将可以显示它的该模块的有关状态：

```
freeswitch> rtmp status  
default tcp:0.0.0.0:1935 profile
```

上面的命令显示了一个 RTMP 的 Profile 运行在 1935 端口上。

FreeSWITCH 源代码中也实现了一个 Flash 版的软电话，并提供了客户端的例子。客户端及例子页面的源代码在 FreeSWITCH 源代码目录的『clients/flex/』目录中。要运行该例子，我们首先要进行一些修改，将freeswitch.html中的『rtmp_url』参数改为指向我们自己的 RTMP 服务器。如，原来的参数值如下：

```
rtmp_url: 'rtmp://my.ip.address.here/phone'
```

笔者在测试时修改后的值如下，它指向笔者本机上的 RTMP 服务（即我们上面打开的 1935 端口上的服务）：

```
rtmp_url: 'rtmp://127.0.0.1/phone'
```

当然，为了能访问该 HTML 页面，我们还需要把这些相关的 Web 资源都放到一个 Web 服务器上运行。python 语言提供了一个简单的方案用于运行一个简单的 Web 服务器，我们可以在客户端的源代码目录中使用下列命令启动一个简单的 HTTP Server，它将默认运行在 8000 端口上。

```
# python -m SimpleHTTPServer
```

然后，打开浏览器，访问该服务器对应的网址，如『<http://localhost:8000/freeswitch.html>』，就可以看到图4.7所示的界面：

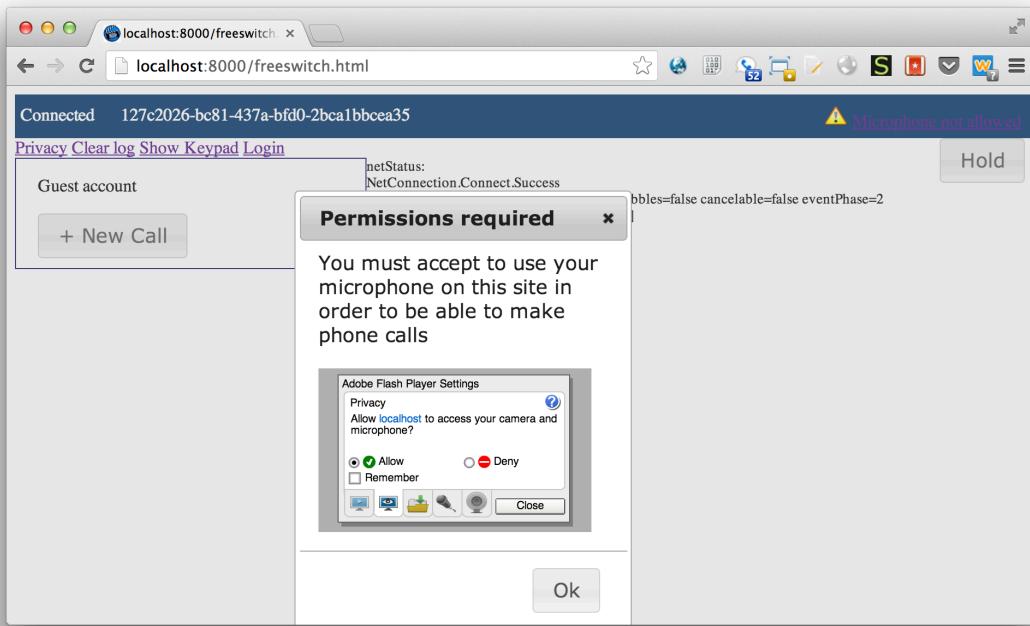


图 4.7: Flash 测试页面初始界面

其中，可以看到在最左上角有『Connected』字样，表示我们已经跟 RTMP 服务器（即 FreeSWITCH）连接上了。但是，由于该网页上的 Flash 软电话需要访问本地的麦克风，由于网页的安全性考虑，系统弹出一个窗口以提示用户是否允许访问这些设备。这里，我们选择『Allow』（允许）就可以了。

接下来，点击『Login』链接，输入用户及密码进行登录（即注册），笔者的输入如图4.8所示：

其中，『1008@192.168.7.5』为默认用户目录的配置，相当于『user@domain』，密码也是默认的『1234』。即，『mod_rtmp』也跟『mod_sofia』一样使用用户目录中的配置对用户进行验证。

注册成功后，我们就可以点击『New Call』按钮输入一个号码拨打电话了，如图 15-x 所以，作为第一个测试我们可以拨打『9196』。

当然，Flash 电话注册后，它是一个真正的电话，所以，你也可以呼叫它。

回忆一下 SIP 中的注册，FreeSWITCH 将记住客户端的 Contact 地址。在 RTMP 中也类似，每当有 RTMP 客户端向 FreeSWITCH 注册时，FreeSWITCH 也记录该客户端的地址，如，类似

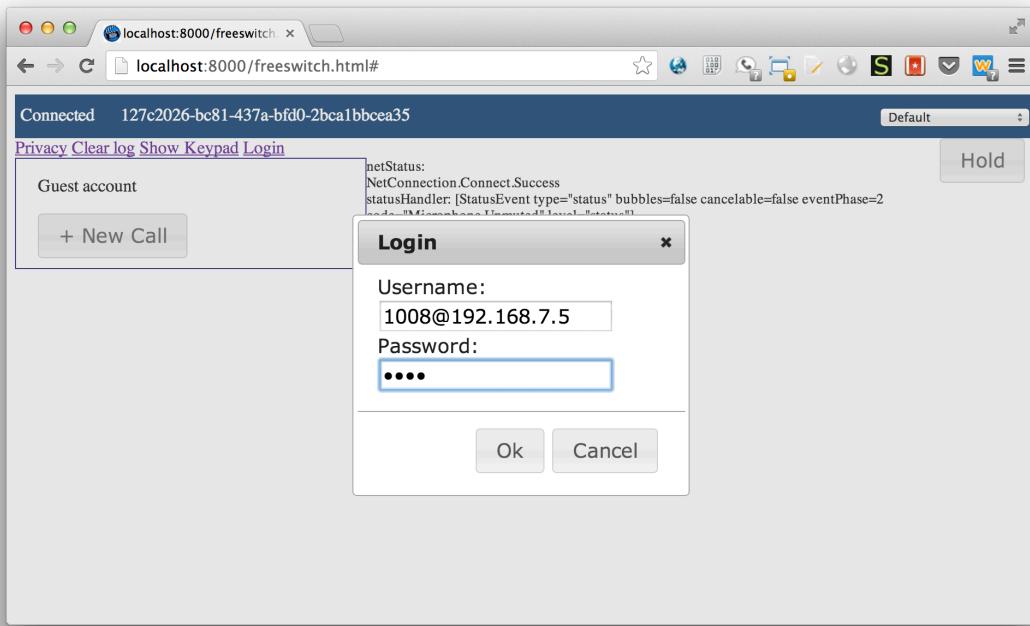


图 4.8: 将 Flash 软电话登录到 FreeSWITCH 上

『mod_sofia』中的『sofia_contact』,『mod_rtmp』提供了一个『rtmp_contact』命令可以查询客户端的注册情况:

```
freeswitch rtmp_contact default/1008@192.168.7.5

rtmp/127c2026-bc81-437a-bfd0-2bca1bbcea35/1008@192.168.7.5
```

其中,『default』是 RTMP Profile 的名称。我们看到一个『rtmp』的呼叫字符串,然后,就可以尝试呼叫它:

```
freeswitch> originate rtmp/127c2026-bc81-437a-bfd0-2bca1bbcea35/1008@192.168.7.5 &echo
```

执行上述命令后,就会在浏览器中听到来话的提示音,并且,可以看到如图4.9的来电提示页面:

测试成功后,就可以使用编辑如下的 Dialplan 让别的 Flash 电话或 SIP 电话呼叫到『1008』了:

```
<extension name="RTMP">
<condition field="destination_number" expression="^(1008)$">
```

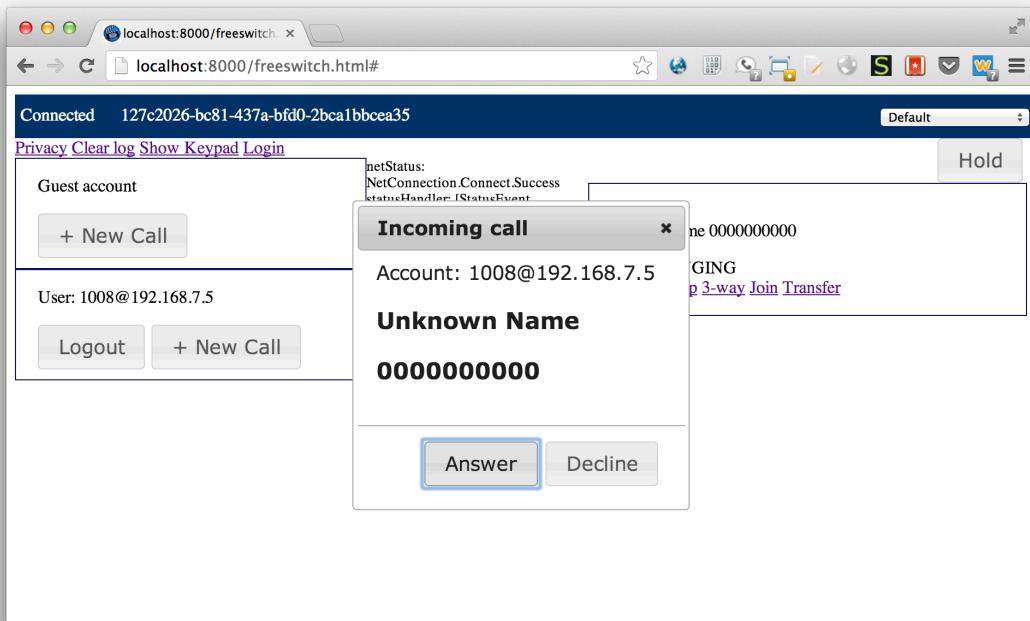


图 4.9: RTMP 来电提示页面

```

<action application="bridge"
  data="${rtmp_contact(default/${domain})}" />
</condition>
</extension>

```

该 Flash 电话客户端提供源代码，也提供 Javascript API 用于在 HTML 网页中调用，因而可以集成到任何系统中去。

不管浏览器如何发展，Flash 技术一直在顽强地生存着。FreeSWITCH 从 1.6 版开始也支持使用 Flash 加入视频会议，图4.10就是笔者做的一个例子（在源代码目录下可以找到`flash-video.html`）：

4.8.2 WebRTC

WebRTC¹¹的全称是 Web RealTime Communication，即基于 Web 的实时通信。实际上 WebRTC 提供了在浏览器中使用 Javascript API 访问本地的声音和视频设备的手段。由于保护隐私和安全性的原因，浏览器在每次使用你的声音或视频设备时都会询问你是否允许。

目前，只有 Chrome 和 FireFox 浏览器实现了 WebRTC。

值得注意的是，WebRTC 仅仅是提供了访问音视频设备这些『媒体』资源的方法，并没有规定

¹¹<http://www.webrtc.org/>。

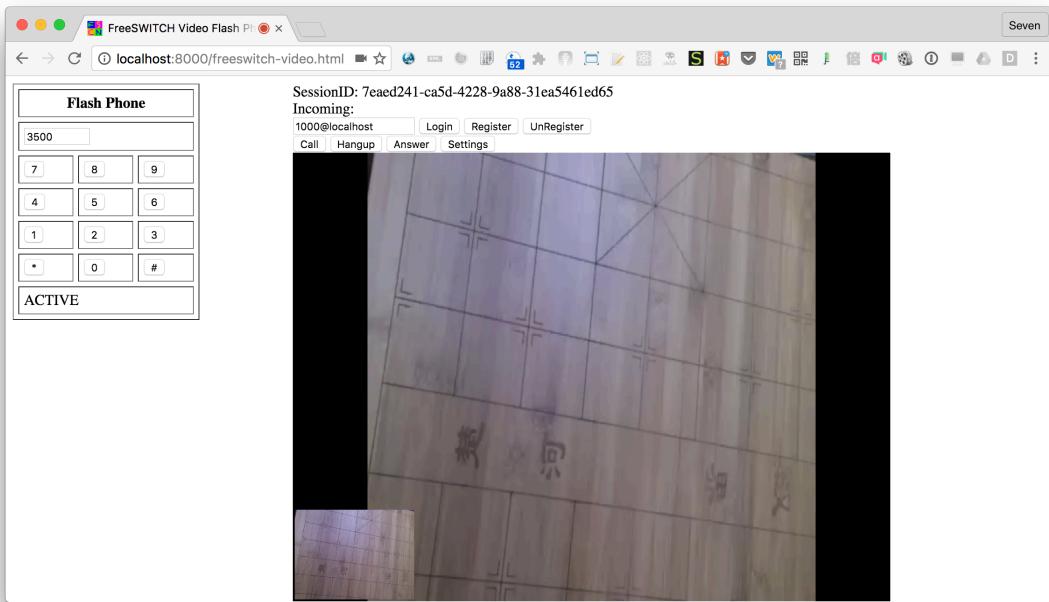


图 4.10: Flash 加入 FreeSWITCH 视频会议

信令是怎么走的。现有的例子大部分都使用了『Google App Engine Channel API』传输信息控制协议。另外一个可以传输信令的就是 WebSocket¹²。

WebSocket 是 HTML5 开始提供的一种浏览器与服务器间进行双向全双工通讯的网络技术。大家熟知的 HTTP 协议仅仅能用于通过浏览器单向地去请求服务器资源，而为了能使服务器主动向浏览器客户端『推送』资源，人们想尽了各种办法，如使用 Ajax 轮循，HTTP 长连接，使用 Flash 提供的 Socket 功能等，IE 中的 ActiveX 控件等。WebSocket 的出现解决了这些连接的混乱，它可以直接使用浏览器提供的功能和 API 与后端建立双向的 Socket 连接。包括 IE9 在内的现代浏览器都支持 WebSocket。

有了 WebSocket 以后，SIP 就的承载方式就又多了一个选择。大家都已经知道，SIP 一般通过 UDP 承载，也可以通过 TCP 或 TLS 承载，而出现了 WebSocket 以后，它也可以承载 SIP，这种技术就称为 SIP over WebSocket¹³，它目前还是一个 IETF 的草案。

信令和媒体都有了，我们就可以进行通信了。需要注意的是，浏览器中的视频编码有两个阵营。一个是以 Google Chrome 为代表的，它支持的视频格式是 VP8；另一个是以 Apple Safari 为代表的，支持 H264，而 FireFox 也支持 H264。由于 FreeSWITCH 不支持转码，目前这两种视频还是不能互通。

另我，在本书写作时，JsSIP 对 FireFox 的支持也不是很好，因而，以下的例子我们都使用 Chrome 浏览器。

¹²<http://zh.wikipedia.org/wiki/WebSocket> <http://tools.ietf.org/html/rfc6455>。

¹³<http://datatracker.ietf.org/doc/draft-ietf-sipcore-sip-websocket/>。

JsSIP

JsSIP¹⁴是一个开源的社区项目，它是一个比较早的 SIP over WebSocket 的 Javascript 实现。

JsSIP 的官方网站列出了这么一些功能：

- 在浏览器和 Node.js 中均能运行
- SIP over WebSocket (在 Web 应用中使用真正的 SIP)
- 通过 WebRTC 支持音频和视频呼叫，支持即时消息
- 轻量级！
- 100% 纯 JavaScript 从头打造
- 易于使用和强大的 API
- 支持 OverSIP、Kamailio、Asterisk、OfficeSIP 等
- 跟 RFC 7118 和 OverSIP 是同一些作者们写的

很多早期的 SIP over WebSoekt 和 WebRTC 项目都使用了它，不过，也有人反映它在跟 FreeSWITCH 对接时有不少问题。FreeSWITCH 社区推荐用另一个项目 SIP.js 替代它。

SIP.js

SIP.js是 JsSIP 的一个 Fork 版本，对原有的 API 有一些增强。更重要的是，它跟 FreeSWITCH 结合更好一些。

如果你想从 JsSIP 转换到 SIP.js，这里有一份指南：<http://sipjs.com/guides/convert-to-sipjs/>。

sipML5

sipML5[^sipml5]是由 Doubango 团队做出的 SIP Over Websocket 的开源实现。该项目可以在线试用，也可以在自己本地试用。sipML5 的源代码是他们 SVN 维护的，使用以下命令可以将该项目检出到本地：

```
svn checkout http://sipml5.googlecode.com/svn/trunk/ sipml5-read-only
```

如果你使用 Apache 或 Nginx Web 服务器，可以把这些代码放到你的 Web 服务器目录下，然后在浏览器上访问。如果你的机器上有 python 的话，可以使用它的 SimpleHTTPServer 模块很简单的启动一个 Web 服务器，如，以下命令将启动一个简单的 Web 服务器，默认使用 8000 端口：

¹⁴<http://jssip.net/>。

```
cd sipml5-read-only
python -m SimpleHTTPServer
```

如果你想改变端口，如改到 8888，则可以使用：

```
python -m SimpleHTTPServer 8888
```

一般系统上自带的都是 python2，如果使用的是 python3，则使用：

```
python3 -m http.server
```

打开你的浏览器，访问该服务器对应的端口，就能看到与 sipml5.org 同样的界面。不过，作者在测试中发现，由于 index.html 而面引用了一个外部的 Javascript，而该地址在国内是打不开的，因而会导致打开这个页面很慢。要解决这个问题，可以在 index.html 中找到包含下面地址的行并把它删掉：

```
https://apis.google.com/js/plusone.js
```

页面打开后，点击『Enjoy our live demo』按钮不可以看到 WebRTC 电话的页面了。当然如果读者知道的话，也可以直接访问下面这个地址：

```
http://localhost:8000/call.htm
```

由于你是在自己的服务器上使用，因此要先进入『Expert mode』，将『WebSocket Server URL』修改为你服务器 WebSocket 的监听地址，修改完毕后点击 Save 就保存了。这些值将保存到本地浏览器的 Local Storage 里，因而也可以直接通过浏览器的调试模式修改这些值，如图4.11。

保存完上述设置后，即可以返回原先的 call.htm 页面。输入相关信息就可以注册（Login）了：图 15-12 显示了笔者填写的注册信息，并演示了注册后测试呼叫 9196 的情况。

另外，FreeSWITCH 官方也提供了 sipml5 的例子，设置起来比较简单。请参考：<https://webrtc.freeswitch.org/sipml5/>。

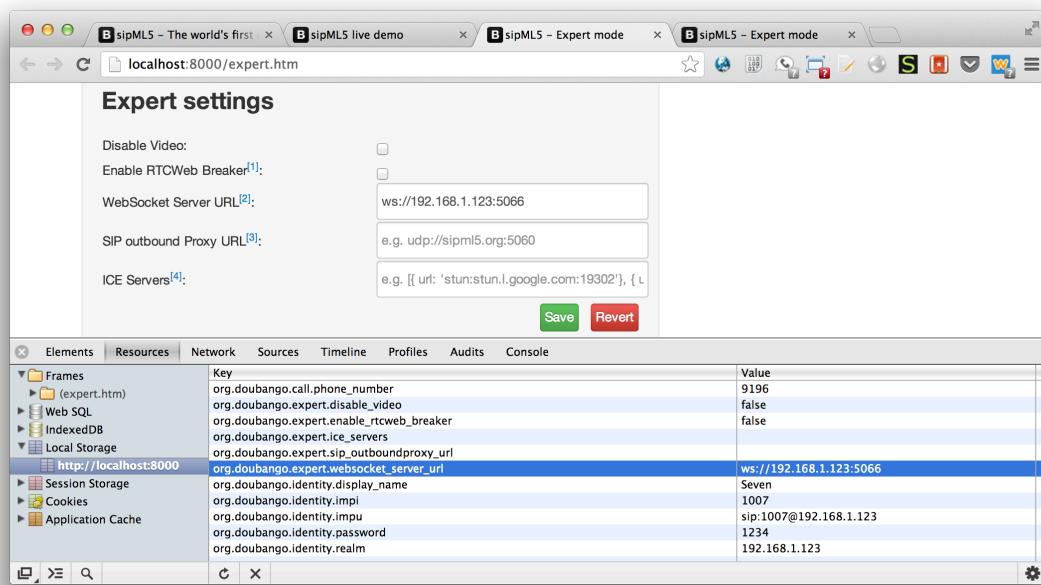


图 4.11: 在 Expert 模式进行设置

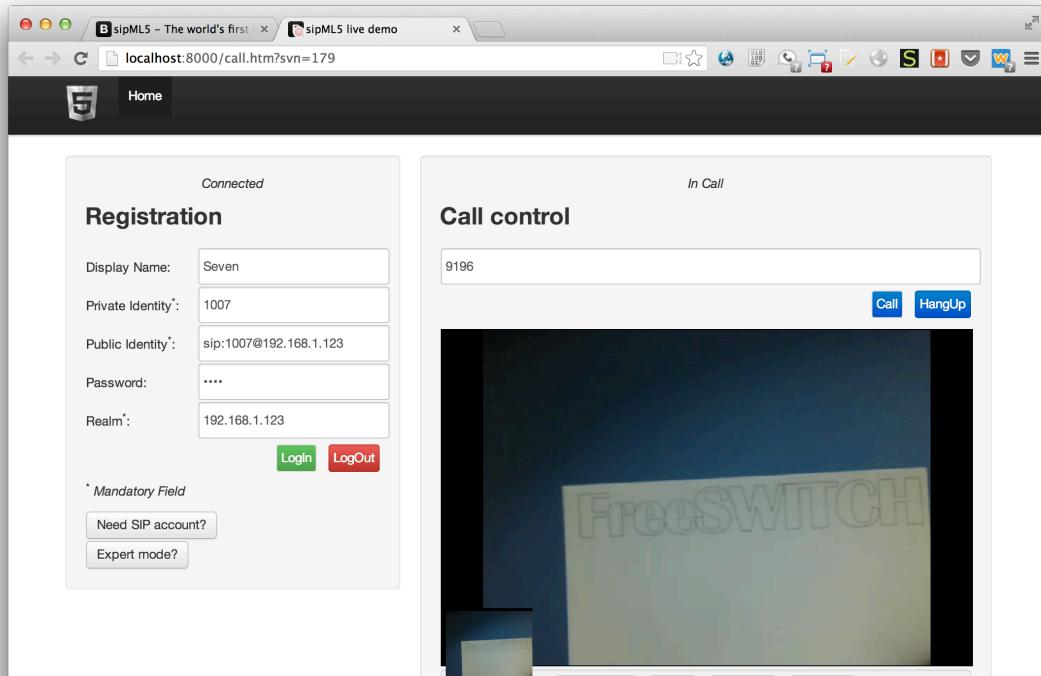


图 4.12: 图 15-12 使用 sipML5 进行 WebRTC 视频呼叫

Verto

我们已经在前面的章节中介绍过 Verto 了，简单地说，就是由于 WebRTC 未规定信令，而 SIP Over Websocket 方式的信令又有诸多问题，因而，FreeSWITCH 自己造了一个基于 JSON RPC 的信令（也是基于 WebSocket），用于支持 WebRTC。

在 2016 年的 ClueCon 上，Anthony 演示了四种浏览器（Microsoft Edge、Safari on Mac、Google Chrome 及 FireFox）同时加入同一个 FreeSWITCH 视频会议的情况（参见第11.19节）。

4.9 发送任意的 SIP 消息

有时候，为了做一些测试，我们可能想模拟发送一些 SIP 消息。在此，我们分记录几种实用的方法。

4.9.1 SIPP

SIPP 是一个很好的测试工具，我们在《FreeSWITCH 权威指南》中也多次讲到，暂时，把它列在这里占位。

4.9.2 sipsak

sipsak¹号称是 SIP 里的瑞士军刀。它可以发送任意的 SIP 消息。比如，我们想给 FreeSWITCH 发送一个 OPTIONS 消息，我们可以创建如下文件：

```
OPTIONS sip:192.168.7.5:5080 SIP/2.0
Contact: <sip:MGC1@192.168.7.5:5061>
To: <sip:MGC2@192.168.7.5:5080>
From: MGC1<sip:MGC1@10.0.45.97:5061>;tag=1707324631353641115CA100
Organization: ALCATEL-SBELL
Call-ID: 02077552298140000001ED3C@n25-sip12
Max-Forwards: 70
CSeq: 1 OPTIONS
X-Timestamp: 23569
Via: SIP/2.0/UDP 10.0.45.97:5061;branch=z9hG4bK42976A826C9B70505BF240E4E4326903
Content-Length: 0
```

上述 OPTIONS 消息是在一次跟阿尔卡特设备对接时抓下来的。把文件命名为 options.sip，然后使用如下命令就可以给 FreeSWITCH 发送上述 SIP 消息了。

```
dujinfang@seven:~/workspace/sipsak$ sipsak -vv -s sip:192.168.3.168 options.sip
warning: need raw socket (root privileges) to receive all ICMP errors

message received:
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.3.168:54013;branch=z9hG4bK.7c8bb448;rport=50761;alias
From: sip:sipsak@192.168.3.168:54013;tag=3ff47571
To: <sip:192.168.3.168>;tag=mygj3m7v8Z32Q
Call-ID: 1072985457@192.168.3.168
CSeq: 1 OPTIONS
Contact: <sip:192.168.3.168>
User-Agent: FreeSWITCH-mod_sofia/1.7.0+git~20160620T235421Z-a2ba998cea~64bit
Accept: application/sdp
Allow: INVITE, ACK, BYE, CANCEL, OPTIONS, MESSAGE, INFO, UPDATE, REGISTER, REFER, NOTIFY, PUBLISH,
       SUBSCRIBE
Supported: timer, path, replaces
Allow-Events: talk, hold, conference, presence, as-feature-event, dialog, line-seize, call-info, sla,
              include-session-description, presence.winfo, message-summary, refer
Content-Length: 0

** reply received after 0.375 ms **
SIP/2.0 200 OK
final received
```

好玩吧？

4.9.3 Quaff

sipsak 虽然很强大，但对于需要改变各种 SIP 参数的情况，还不是很方便。后来，笔者又找到了 Quaff <https://github.com/rkday/quaff>。

Quaff 是用 Ruby 写的。Ruby 语言非常简单，我们来试一把。

首先安装 Quaff

```
gem install quaff
```

gem 是 Ruby 的包管理工具。顺利的话，上述命令就安装完成了。

但如果不幸的话（由于众所周知的原因，在国内访问国外的网络可能会遇到困难），可以试试 rubygem 国内的镜像<https://ruby.taobao.org/>。

笔者使用如下命令切换到国内镜象：

```
$ gem sources --add https://ruby.taobao.org/ --remove https://rubygems.org/
$ gem sources -l
*** CURRENT SOURCES ***

https://ruby.taobao.org
```

Quaff 提供了一些脚本的例子：<https://github.com/rkday/quaff-examples>。为了发送 OPTIONS，笔者写了以下脚本 options.rb：

```
require 'quaff'

phone = Quaff::UDPSIPEndpoint.new("sip:1000@192.168.3.168", "1000", "1234", 5062, "192.168.3.168", 5060)
call = phone.outgoing_call("sip:uas@example.com")

call.send_request("OPTIONS")
call.recv_response("200")
puts "Successful call!"
call.end_call
```

执行一下：

```
dujinfang@seven:~/workspace/sipp/quaff$ ruby options.rb
Successful call!
```

FreeSWITCH 上收到的 SIP 消息如下：

```
freeswitch@seven.local> recv 397 bytes from udp/[192.168.3.168]:5062 at 11:47:13.048522:
-----
OPTIONS sip:uas@example.com SIP/2.0
From: <sip:1000@192.168.3.168>;tag=8bebe3c9e878e6ebd2b67bc0df604261
To: <sip:uas@example.com>
Call-ID: ce70334d5a1b9c33387bc772d73d6480
CSeq: 2 OPTIONS
Via: SIP/2.0/UDP 127.0.0.1:5062;rport;branch=z9hG4bK1467344833.048324
Max-Forwards: 70
Content-Length: 0
User-Agent: Quaff SIP Scripting Engine
Contact: <sip:quaff@127.0.0.1:5062;transport=UDP;ob>
```

```

send 758 bytes to udp/[192.168.3.168]:5062 at 11:47:13.048762:
-----
SIP/2.0 200 OK
Via: SIP/2.0/UDP 127.0.0.1:5062;rport=5062;branch=z9hG4bK1467344833.048324;received=192.168.3.168
From: <sip:1000@192.168.3.168>;tag=8bebe3c9e878e6ebd2b67bc0df604261
To: <sip:uas@example.com>;tag=pg336a932Hg8e
Call-ID: ce70334d5a1b9c33387bc772d73d6480
CSeq: 2 OPTIONS
Contact: <sip:192.168.3.168>
User-Agent: FreeSWITCH-mod_sofia/1.7.0+git-20160620T235421Z~a2ba998cea~64bit
Accept: application/sdp
Allow: INVITE, ACK, BYE, CANCEL, OPTIONS, MESSAGE, INFO, UPDATE, REGISTER, REFER, NOTIFY, PUBLISH, SUBSCRIBE
Supported: timer, path, replaces
Allow-Events: talk, hold, conference, presence, as-feature-event, dialog, line-seize, call-info, sla, include-session
Content-Length: 0

```

我们再看一个脚本 sub.rb:

```

require 'quaff'

ip = "192.168.3.168"

phone = Quaff::UDPSIPEndpoint.new("sip:1000@#{ip}", "1000", "1234", 5062, ip, 5060)
call = phone.outgoing_call("sip:1000@#{ip}")

call.send_request("SUBSCRIBE", "", {"Event" => "message-summary"})
call.recv_response("202")
puts "Successful call!"
call.end_call

```

第1行，加入 quaff。第3行，设置 ip 变量。第5行，初始化一个 SIP Endpoint，第6行，初始化一个 call。第7行，发送 SUBSCRIBE 消息，指定 Event 头为 message-summary。第8行，期待收到 202 消息。

执行此脚本，FreeSWITCH 端的消息如下：

```

freeswitch@seven.local> recv 431 bytes from udp/[192.168.3.168]:5062 at 11:49:29.604030:
-----
SUBSCRIBE sip:1000@192.168.3.168 SIP/2.0
From: <sip:1000@192.168.3.168>;tag=a8125a5c3a38c8f1d113ea820e71a85e
To: <sip:1000@192.168.3.168>
Call-ID: e9dfa7c5fb2d684eedc5df85c6e253d

```

```

CSeq: 2 SUBSCRIBE
Via: SIP/2.0/UDP 127.0.0.1:5062;rport;branch=z9hG4bK1467344969.603828
Max-Forwards: 70
Content-Length: 0
User-Agent: Quaff SIP Scripting Engine
Contact: <sip:quaff@127.0.0.1:5062;transport=UDP;ob>
Event: message-summary

-----
send 809 bytes to udp/[192.168.3.168]:5062 at 11:49:29.613898:

SIP/2.0 202 Accepted
Via: SIP/2.0/UDP 127.0.0.1:5062;rport=5062;branch=z9hG4bK1467344969.603828;received=192.168.3.168
From: <sip:1000@192.168.3.168>;tag=a8125a5c3a38c8f1d113ea820e71a85e
To: <sip:1000@192.168.3.168>;tag=M5DS4jtGqhhj
Call-ID: e9dfa7c5fdb2d684eedc5df85c6e253d
CSeq: 2 SUBSCRIBE
Contact: <sip:1000@192.168.3.168:5060>
Expires: 3600
User-Agent: FreeSWITCH-mod_sofia/1.7.0+git-20160620T235421Z~a2ba998cea~64bit
Allow: INVITE, ACK, BYE, CANCEL, OPTIONS, MESSAGE, INFO, UPDATE, REGISTER, REFER, NOTIFY, PUBLISH, SUBSCRIBE
Supported: timer, path, replaces
Allow-Events: talk, hold, conference, presence, as-feature-event, dialog, line-seize, call-info, sla, include-session
Subscription-State: active;expires=3600
Content-Length: 0

```

工欲善其事，必先利其器。在工作中找到合适的工具，必定事半功倍。

4.10 在呼叫失败的情况下如何播放语音提示

看到好多朋友问到这个问题。一般我们在打电话时会听到“您拨的电话正在通话中，请稍后再拨…”，或“电话无应答…”之类的提示，我们在 FreeSWITCH 里也可以这样做。

其实很简单，默认的配置在呼叫失败时会转到 voicemail (语音信箱)，我们只需要在这里修改，让他播放一个语音提示，然后再进入语音信箱（或直接挂断也行）。

在默认配置的 default Dialplan 中找到<extension name="Local_Extension">部分的最后几行：

```

<action application="bridge" data="user/${dialed_extension}@${domain_name}" />
<action application="answer" />
<action application="sleep" data="1000" />
<action application="bridge" data="loopback/app=voicemail:default ${domain_name} ${dialed_extension}" />

```

其中，第一个bridge是说明去呼叫被叫号码，如果呼叫失败，则 Dialplan 继续往下走，依次是

- 应答
- 睡一会 (sleep)
- 进入voicemail
- OK, 我们只需要把最后一个bridge那行改成

```
<action application="playback" data="${originate_disposition}.wav"/>
```

重新打电话试一下吧，如果被叫忙，就会播放对应的声音文件。

“骗人！”你肯定会这么说，“我照着你的做还是没有听到声音！”

别急，我还没说完。其实呢，说到这里问题的主要部分已经说清楚了。剩下的就是需要你自己练习。因为，你肯定在呼叫的时候在 FreeSWITCH 的日志中看到红色的错误日志，提示你找不到某个声音文件。接下来怎么办？当然是录一个放上去。

一盘来说，如果被叫忙，则originate_disposition变量的值就是USER_BUSY，用户没注册就是USER_NOT_REGISTERED之类的，你只需要保证相关目录下有相对应的声音文件即可。

当然，呼不通的原因可能有很多，你总不可能录上所有的声音文件是吧，有两种方法：

1) 使用一个 Lua (或其它语言) 的脚本

在 lua 脚本中可以拿到这个originate_disposition变量，从而可以使用if-then-else 之类的逻辑播放各种声音文件。一个示例脚本如下：

```
reason = session:getVariable("originate_disposition")
sound = "unknown_error.wav"

if reason == 'USER_BUSY' then
    sound = "busy.wav"
else if reason == 'USER_NOT_REGISTERED' then
    sound = "not_registered.wav"
end

session:streamFile(sound)
```

2) 当然，如果你脚本也不想编辑的话，实际上 FreeSWITCH 的 Dialplan 功能还算比较强大的，你只需要将呼叫转到播放不同声音文件的 Dialplan：

然后创建如下 Dialplan Extension:

```

<extension name="Local_Extension_play-cause">
<condition field="destination_number" expression="^play-cause-USER_BUSY$">
    <action application="playback" "/tmp/sounds/user-busy.wav"/>
</condition>
</extension>

<extension name="Local_Extension_play-cause">
<condition field="destination_number" expression="^play-cause-USER_NOT_REGISTERED$">
    <action application="playback" "/tmp/sounds/user-not-registered.wav"/>
</condition>
</extension>

<extension name="Local_Extension_play-cause">
<condition field="destination_number" expression="^play-cause0(.*)$">
    <!-- for all other reasons, play this file -->
    <action application="log" data="WARNING hangup cause: $1"/>
    <action application="playback" "/tmp/sounds/unknown-error.wav"/>
</condition>
</extension>

```

当然，能播放上面的声音文件还有一个前提，就是在第一个bridge前面要有以下两行：

```

<action application="set" data="hangup_after_bridge=true"/>
<action application="set" data="continue_on_fail=true"/>

```

其中第一行的作用是，如果第一个bridge成功了，被叫挂断电话后我们就没有必要再播放该声音了，因此直接挂机。当然这一行可以没有，那么你在后面的originate_disposition里如果发现值是“NORMAL_CLEARING”（正常挂机）的情况再决定是否播放相关语音。

第二行的作用是，如果呼叫失败（空号，拒接等），继续往下走，否则（值为false的情况）到这里就挂机了。该变量的值还可以有以下几种，表示只有遇到这几种情况才播放语音，其它的就直接挂机。

```

<action application="set" data="continue_on_fail=
NORMAL_TEMPORARY_FAILURE,USER_BUSY,NO_ANSWER,TIMEOUT,NO_ROUTE_DESTINATION"/>

```

4.11 被叫忙的处理

问题：1002 分机与 1001 分机正在通话，此时 1003 分机打给 1001，怎么让 1003 分机知道 1001 正忙，拨一段语音，Diaplan 要怎么配置？

类似的问题有很多同学问到，这里，我们来看一下解决方案。

在传统的 PSTN 电话中，一个电话只能接听一路呼叫，如果被叫忙，主叫就会听到忙音。而在 SIP 电话中，大多数的 SIP 话机或者软电话都可以支持多路通话，所以，被叫一般不会那么“忙”。但，总是有些人怀念原来的习惯，希望 SIP 电话也跟以前用的电话表现起来一模一样。

最理想的解决方案，就是什么都不干。如 A 通过 FS 呼叫 B，在 B 忙的情况下，B 的终端会回 486 BUSY HERE SIP 消息，FS 收到后转发给 A，A 收到 486 消息，播放指定的声音（这个声音可以在 A 的客户端配置，由于 SIP 终端本身就是智能终端，这不是什么事）。

然后，现实世界总是不那么理想。尤其是，现实世界是多样的。比方说，A 可能是个 PSTN 话机，这样的话，就需要依赖 FS 给 A 播放一段声音。

好吧，我们继续尝试其实的解决方案。其实解决该问题，最简单的办法在上一节中已经讲过了。找一个仅支持一路呼叫的 SIP 话机，如果第二路呼叫进来，话机就会拒绝，并发送 486（代表忙）SIP 消息。在 FreeSWITCH 中就能收到 USER_BUSY 的挂机原因，然后，……都在上一节讲过了。

说到这里，还有没有其它的解决方案呢？当然有，只有想不到，没有做不到。

FreeSWITCH 提供了一个 limit App，专门做限制。当然，要理解 limit，还需要多了解 FS 的一些基础知识。

好吧，我们先来做个实验。对于本节的问题，我们做如下 Dialplan：

```
<extension name="busy">
<condition field="destination_number" expression="^(100[1-3])$">
<action application="bridge" data="user/$1"/>
</condition>
</extension>
```

上面这个 Dialplan 大家都比较熟悉了，当 1002 呼叫 1001 时，使用 bridge 把两者桥接起来。

好了，祭出我们的 limit 大神。limit 的语法如下：

```
limit <backend> <realm> <resource> <max[/interval]> [<transfer_destination_number> [<dialplan> [<context>]]]
```

其中，**backend** 是 limit 的后台，可以是 hash（存在内存的哈希表里），db（存在数据库里）以及 redis（存在 redis 里）等；而 **realm** 是一个域，可以是任意值，在多租户环境下，一般用当前的 domain；**resource** 表示对哪些资源进行限制，可以是一个分机号；**max** 是限制的最大值，后面有个可选的 **interval**，表示在多长时间内该最大值有效；**transfer_destination_number**、**dialplan**、**context** 是可选的，表示，如果超出限制，要 transfer 到哪个 Dialplan。

参照上面的语法，我们做如下 Dialplan：

```

<extension name="busy">
  <condition field="destination_number" expression="^(100[1-3])$">
    <action application="limit" data="hash ${domain} $1 1 user_busy XML default"/>
    <action application="bridge" data="user/$1"/>
  </condition>
</extension>

<extension name="busy">
  <condition field="destination_number" expression="^user_busy$">
    <action application="playback" data="user_busy.wav"/>
  </condition>
</extension>

```

有了上述 Dialplan，当有用户呼叫1001时，会先执行limit，对当前的被叫号码1001的资源占用数加1。如果在通话过程中用第二个电话呼叫1001，则由于超出了资源限制，系统会自动转接到user_busy处，播放一个用户忙的语音提示user_busy.wav。

细心的读者可能发现，其实我们上面做的不够全面。如果1002呼叫1001，那么，当有用户呼叫主叫号码1002时，也应该播放忙音。原来的问题没有说明1001到底是主叫还是被叫，这是提问者提的不够完善的地方，不过，无论如何，我们可以同时用两个 limit 限制两个资源，如：

```

<action application="limit" data="hash ${domain} $1 1 user_busy XML default"/>
<action application="limit" data="hash ${domain} ${caller_id_number} 1 user_busy"/>

```

好了，打个电话，我们可以在日志里看到类似如下的内容：

```

[DEBUG] switch_limit.c:126 incr called: 192.168.7.6_1002 max:1, interval:0
[DEBUG] mod_hash.c:196 Usage for 192.168.7.6_1002 is now 1/1
[DEBUG] switch_limit.c:126 incr called: 192.168.7.6_1001 max:1, interval:0
[DEBUG] mod_hash.c:196 Usage for 192.168.7.6_1001 is now 1/1

```

挂断电话，看到资源被释放了：

```

[DEBUG] mod_hash.c:297 Usage for 192.168.7.6_1002 is now 0
[DEBUG] mod_hash.c:297 Usage for 192.168.7.6_1001 is now 0

```

在上述解决方案中，如果主被叫都是在内网上，当然是没有问题的。但是，如果主叫是 PSTN 呼入的，一般的运营商不允许你在应答（answer）前播放语音（即它们不会透传 Early Media），所以，

你需要在playback前加上一个answer，这是运营商的限制。不过，加了answer以后又破坏了这么做的语义，本来你是想在应答之前通过 Early Media 通知主叫被叫忙，电话打不通本来不应该收费的，加了answer会触发计费。

所以，如果你有PSTN的主叫的话，你可以尝试直接在SIP中返回用户忙，由运营商来给主叫用户播放忙音提示。Dialplan如下：

```
<extension name="busy">
  <condition field="destination_number" expression="^user_busy$">
    <action application="respond" data="486"/>
  </condition>
</extension>
```

部分日志供参考：

```
[INFO] mod_hash.c:183 Usage for 192.168.7.6_1002 is already at max value (1)
[NOTICE] switch_ivr.c:2167 Transfer sofia/internal/1000@192.168.7.6 to XML[user_busy@default]
Dialplan: sofia/internal/1000@192.168.7.6 parsing [default->busy] continue=false
[DEBUG] switch_core_state_machine.c:328 sofia/internal/1000@192.168.7.6 Standard EXECUTE
EXECUTE sofia/internal/1000@192.168.7.6 respond(486)
[DEBUG] switch_core_state_machine.c:60 sofia/internal/1000@192.168.7.6 Standard HANGUP, cause: USER_BUSY
send 869 bytes to udp/[192.168.7.6]:16664 at 14:37:53.273916:
-----
SIP/2.0 486 Busy Here
```

在本文完成后，笔者又发现了另一种用法。如果最后只是返回USER BUSY，不用单独写一个Dialplan Extension，而是直接在limit的参数中写上即可，如：

```
<action application="limit" data="hash ${domain} $1 1 !USER_BUSY"/>
```

注意，USER_BUSY前面应该有个感叹号。

小结：该问题是有人在QQ群里提出的。从本文可以看出，你觉得很简单的问题可能不是三言两语就能回答的，所以，我们建议类似这样的问题到[BBS](#)上提问，尽量把问题描述清楚，才方便解答。

4.12 话机上被叫号码被改变问题

在一次测试中，使用潮流（Grandstream）话机呼叫另一个本地分机，发现在振铃阶段被叫的电话号码显示有问题，明明呼叫的是1002，话机上却显示诡异的1002-0x7ffab3f18ea0，实在让人不明白。

测试使用的 Dialplan 如下：

```
<extension name="test">
<condition field="destination_number" expression="^(.*)$">
<action application="bridge" data="user/${dialed_extension}@${domain_name}" />
</condition>
</extension>
```

打开 SIP Trace 看一下，发现在 FS 回复主叫的 180 消息中有如下内容：

```
P-Asserted-Identity: "Outbound Call" <sip:1002-0x7ffab3f18ea0@192.168.3.166>
```

看来问题就出在这儿。FS 给潮流回的 180 消息中的信息影响了话机的显示。

其实这是一个好的特性。即如果发生被叫号码改变的情况下，可以在主叫端看到改变后的被叫号码，以及名称（这里是 FS 默认的 Outbound Call，且不管它）。但是，在此，我们明白不想看到 1002-后面那一长串信息。

那么，这个数据是哪儿来的呢？

继续看 SIP 消息，发现 FS 在呼叫被叫的 INVITE 消息里就是这样写的：

```
INVITE sip:1002-0x7ffab3f18ea0@192.168.3.166:59379 SIP/2.0
```

好吧，继续查：

```
freeswitch> sofia status profile internal reg
```

发现用户 1002 的注册信息如下：

Call-ID:	0f24bde35f5ec3c5
User:	1002@192.168.3.166
Contact:	"<sip:1002-0x7ffab3f18ea0@192.168.3.166:59379>"
Agent:	bairesip v0.4.13 (x86_64/darwin)

从注册信息中可以看出，笔者使用的是 baresip 客户端。那一串诡异的数字是从注册信息的 Contact 字段来的。

从 SIP 角度来讲，INVITE 消息是没有错的，它就应该按被叫的 Contact 发送。常见的客户端，Contact 中的消息一般是类似`1002@domain`的样子，当然，Contact 实际上可以任意写。比如，Blink 的注册信息如下：

```
User: 1003@192.168.3.166
Contact: "Seven Du" <sip:72615809@192.168.3.166:57823>
Agent: Blink Pro 4.4.1 (MacOSX)
```

如果呼叫 1003，则显示`72615809`，比起`1002-0x7ffab3f18ea0`更容易让人迷惑。当然，这是合法的。

那么，如果看来，FS 不应该把被叫的 Contact 内容带到它回复的 180 中去，可是，FS 为什么这样做呢？

不知道，继续试，后来发现，使用 FS 默认的 Dialplan 设置就不会发生这个现象。

通过对比我们使用的 Dialplan 和默认的 Dialplan，发现默认的 Dialplan 会设置`ringback`，进而，FS 在振铃阶段向主叫返回的是 183 消息还不是 180，而在 183 消息中，显示是正确的：

```
P-Asserted-Identity: "1002" <sip:1002@192.168.3.166>
```

既然如此，我们也在我们上面的测试 Dialplan 中加入了如下一行，问题解决：

```
<action application="set" data="ringback=${us-ring}" />
```

不过，至于 FS 为什么 180 和 183 消息中 P-Asserted-Identity 内容不一样，有待下一步研究。

4.13 在同一台电脑上启动多个 FreeSWITCH 实例

在《FreeSWITCH 权威指南》中，我们曾讲过一个在同一台电脑上启动多个 FreeSWITCH 实例的例子。当时所有实例都使用同一个 IP，下面，我们使用另一种方式实现，每个 FreeSWITCH 使用一个 IP。

在此，我们拟准备三个 FreeSWITCH 环境。以下操作在 Mac 上进行，在 Linux 上的操作也类似。

首先找到 Mac 当前的 IP 192.168.3.119。

在当前网卡上再设置两个静态 IP，通过别名方式实现（在 Linux 上可以使用 eth0:1 虚拟网卡实现）：

```
sudo ifconfig en1 inet 192.168.3.28 netmask 255.255.255.255 alias
sudo ifconfig en1 inet 192.168.3.29 netmask 255.255.255.255 alias
```

检查配置：

```
$ sudo ifconfig en1
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether b8:09:8a:d3:91:2f
      inet6 fe80::ba09:8aff:fed3:912f%en1 prefixlen 64 scopeid 0x5
        inet 192.168.3.119 netmask 0xffffffff broadcast 192.168.3.255
      inet6 fd88:be92:acb::ba09:8aff:fed3:912f prefixlen 64 autoconf
      inet6 fd88:be92:acb::e1d2:ee95:2da:d148 prefixlen 64 autoconf temporary
      inet6 fd77:2340:cb8b::ba09:8aff:fed3:912f prefixlen 64 autoconf
      inet6 fd77:2340:cb8b::81e:fc5d:30f3:3546 prefixlen 64 autoconf temporary
        inet 192.168.3.28 netmask 0xffffffff broadcast 192.168.3.28
        inet 192.168.3.29 netmask 0xffffffff broadcast 192.168.3.29
      nd6 options=1<PERFORMNUD>
      media: autoselect
      status: active
```

设置目录：

```
cd /usr/local
mkdir fs28
mkdir fs29
cp -R /usr/local/freeswitch/conf fs28/
cp -R /usr/local/freeswitch/conf fs29/
mkdir fs2{8,9}/{log,db}
```

分别修改vars.xml，使它们看起来如下：

```
<X-PRE-PROCESS cmd="set" data="local_ip_v4=192.168.3.28"/>
<X-PRE-PROCESS cmd="set" data="domain=$${local_ip_v4}"/>

<X-PRE-PROCESS cmd="set" data="local_ip_v4=192.168.3.29"/>
<X-PRE-PROCESS cmd="set" data="domain=$${local_ip_v4}"/>
```

分别修改autoload_configs/event_socket.conf.xml以避免 IP/端口冲突:

```
<param name="listen-ip" value="$$\{local_ip_v4\}" />
<param name="listen-ip" value="$$\{local_ip_v4\}" />
```

在fs28和fs29中分别启动 FreeSWITCH

```
cd fs28 && ../../freeswitch/bin/freeswitch -log log -db db -conf conf -nonat
cd fs29 && ../../freeswitch/bin/freeswitch -log log -db db -conf conf -nonat
```

好了，加上默认的 FreeSWITCH (我们放在fs14目录下，IP 为192.168.3.119) 我们现在一共有三个 FreeSWITCH，用电话客户端分别注册三个 FreeSWITCH 确保能正常注册。

收工。

4.14 FreeSWITCH Bug 解决一例

杜金房/2016.04.15

在前几天南京的 FreeSWITCH 培训班上，我给大家演示了通过 Flash 呼叫 FreeSWITCH，发现在有视频转码的情况下，Flash 端不显示视频。当时由于时间关系，没有深究。

后来，看到 FreeSWITCH 上有人也提到这个问题了。我就连夜调试了一下，把调试过程跟大家分享一下。

首先，确定现象。用 Flash 呼叫标准的 9196，执行 echo，Flash 端是有视频的，这说明 Flash 这层一般没什么问题。

呼叫 3500 进入转码的视频会议，Flash 端看不到视频。然后，再用其它客户端呼入 3500，能看到 Flash 端发到会议里的视频。看来，视频是单向的。

用uuid_debug_media证实，FreeSWITCH 只能收到视频数据，却没有发 (只有 R，没有 W)

因此，调试从发视频的地方开始。

找到 mod_rtmp 里的rtmp_write_video_frame 函数，该函数是负责发送视频的。在该函数里加入几行 Log

```
switch_log_printf(SWITCH_CHANNEL_LOG, SWITCH_LOG_ERROR, "write: %d CNG: %d marker: %d\n", frame->datalen, (frame->flag
```

发现该函数是被正常调用的，也有数据。只是在某些 if/else 分支判断中没有通过，最后没有把数据发送出去。

通过观察数据的 maker 和 CNG 标志，都不正常。

由于数据是经过 mod_av 编码的，因而判断可能出现在编码产生的 frame 标志有误。

找到 avcodec.c，找到 consume_h264_bitstream 函数，果然发现在某一些 refactor 的时候，有一个分支没有正确设置这些标志。

做了一个补丁如下：

```
@@ -767,6 +767,9 @@ static switch_status_t consume_h264_bitstream(h264_codec_context_t *context, swi
     if (pkt->size > 0) av_packet_unref(pkt);

+    switch_clear_flag(frame, SFF_CNG);
+    frame->m = 1;
+
```

打上这个补丁，果然 Flash 端的视频就出来了。

第五章 NAT

现在，我们可以拿出一章来讲 NAT 了。

5.1 FreeSWITCH NAT 问题之一

关于 FreeSWITCH 的学习，我一向推荐大家先从局域网上开始学，等到对 FreeSWITCH 熟悉以后，再研究与 NAT 有关的问题。我这样说不是没有道理的，首先，NAT 涉及很多网络基础知识，如果一个人连网络基础知识都不扎实，连 SIP 也不熟悉（不用说精通了），那 FreeSWITCH 的概念、原理、配置也不是很理解，那么，遇到 NAT 的问题后尝试去解决的困难是可想而知的。

不过，现在大家都已经熟读了《FreeSWITCH 权威指南》，我想，我们可以来研究一下 NAT 了。

当然，NAT 也不是那么可怕，FreeSWITCH 的默认配置已经能很好地应对很多情况了。只是，如果一旦媒体不通或者单通的话，就需要一定的知识和经验来解决这些问题了。

就在准备对 NAT 发表一篇长篇论时，笔者又回去翻了一下书，发现书上其实讲了很多细节了。只不过，大部分都停留在理论上，可能有些枯燥。下面，我们就从真正的例子中讲解一下 NAT。在讲得过程中，我也会提一下相关的原理，但是，更详细的，需要你再回去翻翻书了。

正好今天笔者遇到一个 NAT 的问题，该问题比较有代表性，在此，我来详细研究一下。

首先，我们来看一个正常的呼叫。

FreeSWITCH 部署在阿里云上（阿里应该给我广告费：）），默认配置，在服务器端设置 Dialplan 如下：

```
<extension name="play">
<condition field="destination_number" expression="^(play)">
    <action application="playback" data="tone_stream://$$\{cn-ring\}" />
    <action application="answer" data="" />
    <action application="playback" data="/wav/vacation.wav" />
</condition>
</extension>
```

上面的 Dialplan 的意思是，先放一段回铃音，然后应答，然后放一段声音（大约 15 秒），挂机。

笔者的客户端在 NAT 环境中，IP 地址是 192.168.7.6。客户端软件是 X-Lite 4.7.1 (74250) Mac 版。配置如图5.1所示：

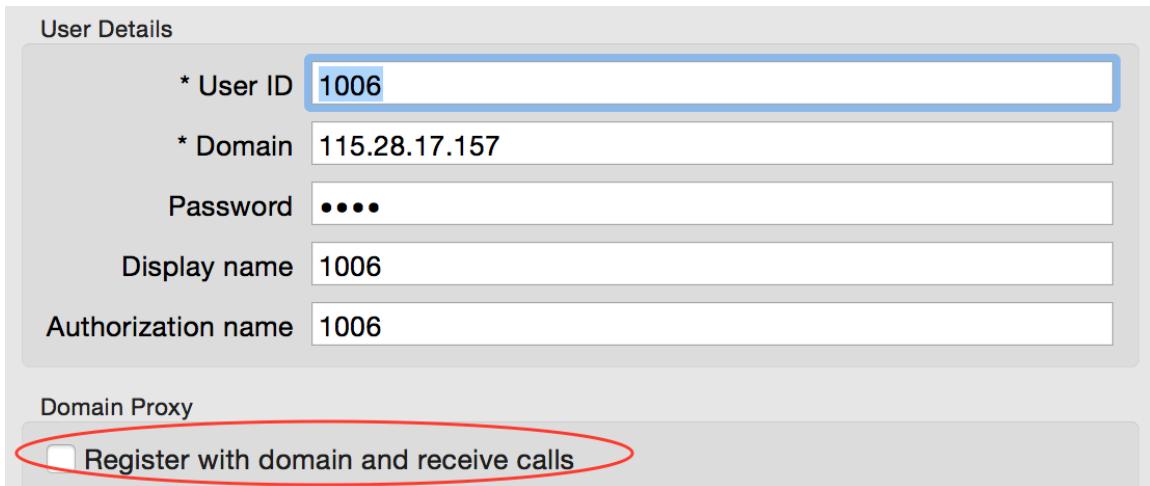


图 5.1: X-Lite 配置

注意，笔者并没有使用『Register Domain And Receive Calls』，因为我们这里不讨论注册，在客户端做主叫时就不需要向服务器注册。

设置好后在客户端侧开启 Wireshark 抓包，在客户端上呼叫 play，一切正常。经 Wireshark 分析，呼叫流程如图5.2。从图上可以看出（我们先忽略掉所有的 RTP 包），X-Lite 向 FreeSWITCH 发送 INVITE 请求，FreeSWITCH 返回 183（带回铃音）以及 200（应答）消息，在 vacation.wav 播放完毕后，FreeSWITCH 发送 BYE 消息通知 X-Lite 挂机。

这时候笔者改变 X-Lite 配置中的 Domain，在 IP 地址后增加:5080，即准备把 INVITE 发送到 FreeSWITCH 的 5080 端口。

另外，我们也需要重新配置 Dialplan，把本文最初那一段 Dialplan 放到 public.xml 中。

配好后重新呼叫 play，这次听到的声音还是正常的。不正常的是在语音播放完毕后电话并没有立即挂断。而是等了一会挂断的。而且，与上次的情况不同，从呼叫流程图5.3上看，X-Lite 只发送了一个 INVITE 请求，而图5.2中是两个。这是由于 FreeSWITCH 的默认配置中 5080 端口是不需要鉴权的，任何人都可以发起呼叫。真正令人费解的是，图 1.3 挂断是由客户端主动发起的，而不是像图 1.2 似的由服务器端发起的。

从图上我们可以看到，客户端连发 4 个 BYE，而服务端却没有反应，客户端果断将通话切断。

为什么呢？我们又没有主动点击挂机按钮？

下面，我们就着手分析。由于前半段呼叫流程好像没什么问题，我们从 BYE 消息开始看。

点开 BYE 消息，我们可以看下以下 SIP 头域：

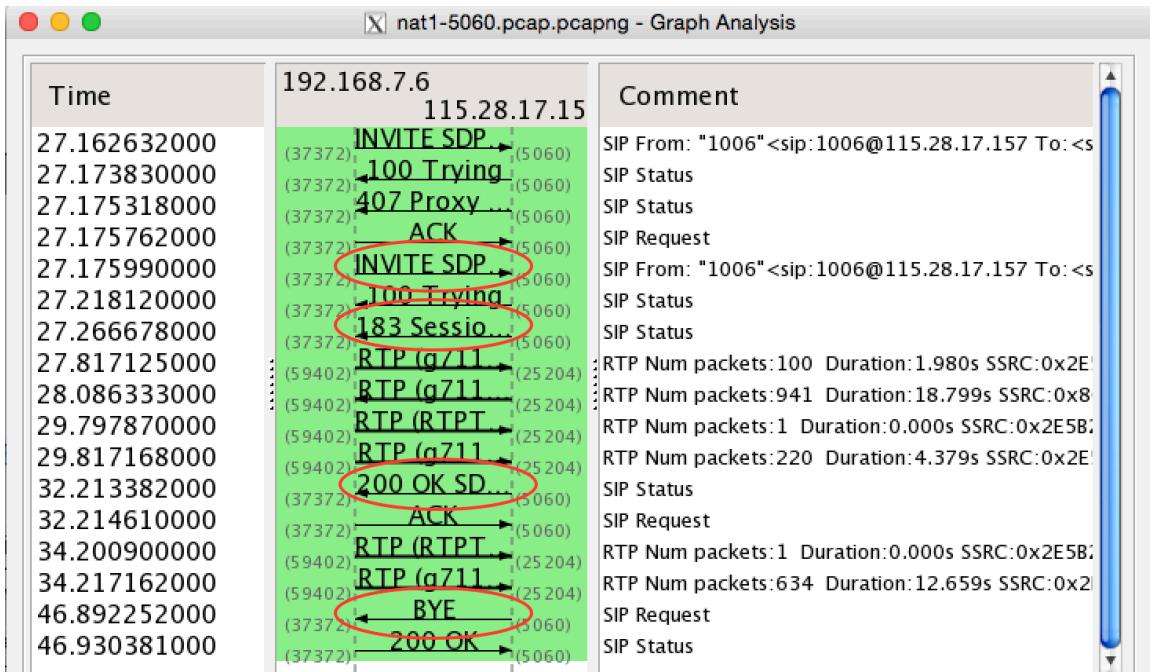


图 5.2: 第一个呼叫流程

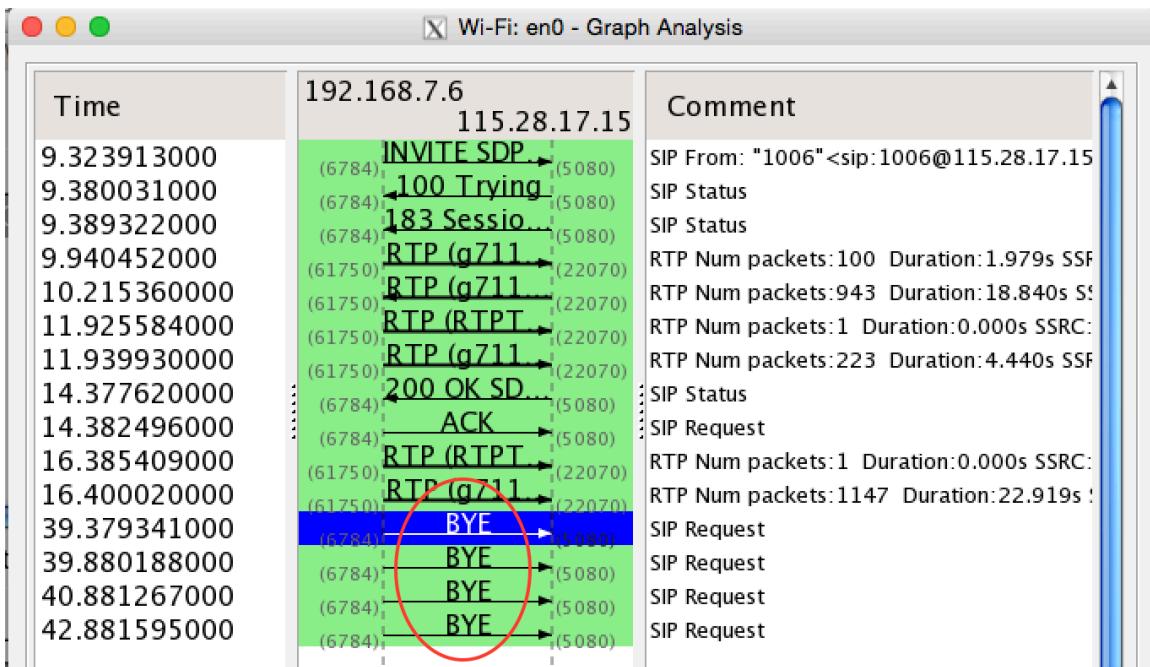


图 5.3: 第二个呼叫流程

```
Reason: SIP;description="RTP stream closed due to inactivity; mediaType=AUDIO"
```

这个 Reason 头域在 SIP 中是可选的，不过在这里它对我们很有用。字面意思是说，客户端检测到 RTP 不活动了（就是超过一定时间没收到 RTP 流），进而挂断了电话。

那么，问题来了。是否挂断电话不是由 SIP 决定的吗？怎么跟 RTP 还有关呢？我们先把这个问题放到这里，到最后再来回答这个问题。

那既然是由于没收到 RTP 挂断电话，有没有个选项可以设置是否启用这个功能呢？或者，这个等待的时长可否控制呢？

答案是肯定的，X-Lite 有一个设置如图5.4。把『Enable inactivity timers』前面的勾去掉，就可以不启用这个功能。当然，感兴趣的读者也可以试一下修改后面『RTP Timers』的值看有什么效果。

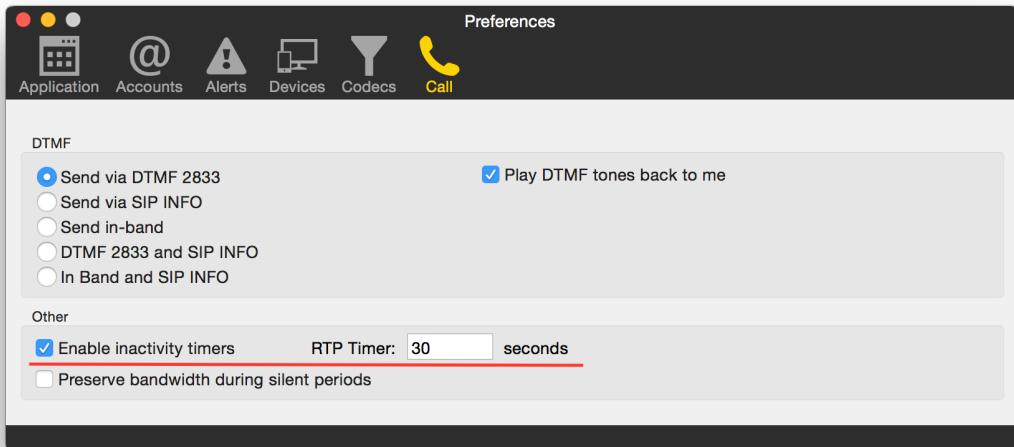


图 5.4: X-Lite RTP timer 设置

去掉我们上面说的勾以后，再重新打个电话，Wireshark 显示的流程图还是一样的，只是，X-Lite 不再自动挂断电话，而是我们受不了，手动按挂断按钮挂断了电话。

看来，我们需要从服务器端找原因了。

到 FreeSWITCH 控制台上，开启 SIP Trace:

```
sofia profile external siptrace on
```

再打一个电话，可以看到，FreeSWITCH 在播放完录音后，便不断地往客户端发 SIP BYE 消息，其中一条消息如下：

```
send 579 bytes to udp/[192.168.7.6]:6784 at 20:45:05.280651:
```

```
-----  
BYE sip:1006@192.168.7.6:6784 SIP/2.0  
-----
```

很显然，我们从上面第一行可以看出，该消息发给了192.168.7.6这个私网的地址，而在服务器端根本找不到这个地址，所以我们的客户端从来就没有收到过 BYE。

其实，在分析 FreeSWITCH 的信令相关的问题时，这样看 SIP Trace 就够了。只不过，为了让大家能看得更直观，我还是用 tcpdump 抓了一个包。

由于在这里我们不关注 RTP，因而只抓了5080端口的 SIP 包，命令如下：

```
tcpdump -i eth1 -s 0 -w /tmp/nat1-5080-server.pcap port 5080
```

用 Wireshark 打开这个抓包文件分析，流程如图5.5：

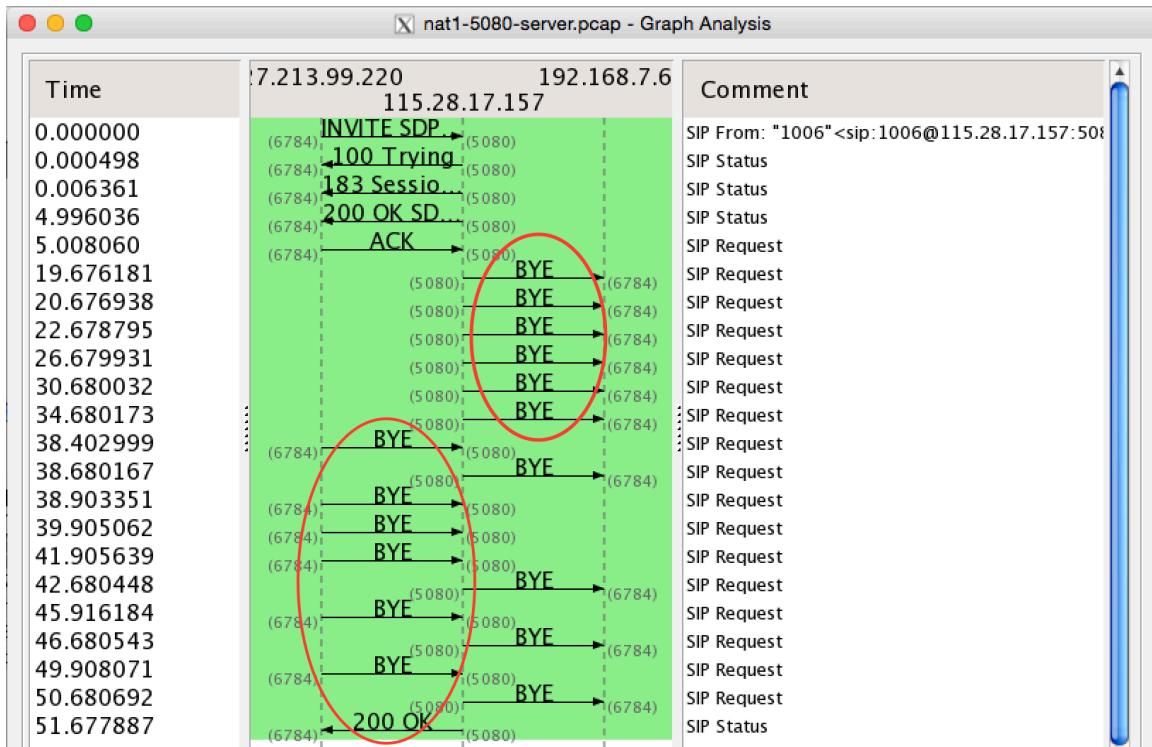


图 5.5: 服务器端的抓包

这个图很有意思，即使你稍微有点看图的经验，也可以一眼看出它是不正常的。其中99.220是 X-Lite 的公网地址，17.157是 FreeSWITCH 的地址，而7.6是 X-Lite 的私网地址。从图上可以看出。虽然 INVITE 请求是在 X-Lite 的私网地址 7.6 上发出来的，但在服务器端，只能看到公网地

址99.220。前面的交互流程都很正常，只是，为什么第一个 BYE 发到7.6上去了呢？而且，为什么发了那么多 BYE 呢？

为了找到答案，我们从 BYE 前面的一个消息入手。点开 BYE 前面的 ACK 消息，可以看到 SIP 头域里有一个 Contact 地址正是192.168.7.6。我们也可以在 FreeSWITCH 控制台上的 SIP Trace 消息里找到这条消息：

```
recv 438 bytes from udp/[27.213.99.220]:6784 at 20:44:49.611836:
-----
ACK sip:play@115.28.17.157:5080;transport=udp SIP/2.0
Contact: <sip:1006@192.168.7.6:6784>
```

FreeSWITCH 就是根据这个 Contact 地址发送的 BYE。由于 BYE 消息没有到达真正的目的地，因而对方也没有任何回应。我们的 SIP 消息是使用 UDP 承载的，为了防止丢包影响通话，SIP 有一个定时重传的机制，因而发送方会相应的重发多个 BYE（但不会无限的发）。

或许有人问，那99.220到17.157之间的好些 BYE 是怎么回事呢？那是因为在 X-Lite 上手工按了挂断，X-Lite 给 FreeSWITCH 发的 BYE，但 FreeSWITCH 还在忙着发 BYE 一直没顾得理它，因而 X-Lite 就一直重发。最后 FreeSWITCH 放弃发 BYE，才有时间给收到的 BYE 回了个 200 OK 消息。不过，最后这个消息倒是发到了正确的 IP 上。

至于为什么 FreeSWITCH 主动发的 BYE 在这里发到了 ACK 指定的 Contact 地址，就要去读 RFC3261 了。不过，我们更关心的问题是—为什么 X-Lite 呼叫 5060 端口（第一个例子，Domain 中只有 IP 地址不指定端口号则端口默认为 5060）的时候一切正常，到了 5080 就不行了呢？

我们对比了两个端口对应的 Profile—internal 和 external，发现 external 中少了如下一个参数：

```
<param name="apply-nat-acl" value="nat.auto"/>
```

FreeSWITCH 中有一些访问控制列表 (ACL, Access Control List)，其中**nat.auto**是 FreeSWITCH 自动生成的，它包含了 RFC1918 中指定的 IP 地址，而笔者的 IP 地址192.168.7.6正好在里面。如果上述参数在 Sofia Profile 里生效，那么 FreeSWITCH 就知道我们 ACK Contact 里指定的是一个私网地址（这里是192.168.7.6），它会根据它学习到的公网地址发送后续的 SIP 消息（即 BYE），而不再傻傻地遵循 RFC 的教导。

果然，笔者试了一下，在external Profile 中加上**apply-nat-acl**参数，再打电话就一切正常了。

最后，我们上面不是还留了一个问题吗？就是为什么 X-Lite 会在一段时间收不到 RTP 后挂断电话呢？其实，读到这里你应该已经明白了。这是客户端的一个保护机制，如果由于任何原因（包括我们例子中的原因）导致收到不正常的 SIP BYE，可以在 RTP 中断后一段时间挂断电话，因为再等也没有用了。当然，如你所知，该选项是可选的，而且，也并不是所有的客户端都有这样的机制。

今天就先写到这里，后面，我们还会分析更多的关于 NAT 的问题。

5.2 FreeSWITCH NAT 问题之二

这是一个真实的案例。

FreeSWITCH 在公网上，SIP 客户端在 NAT 内。客户端做主叫没问题，做被叫会出现“30 秒断”。

通过各种抓包对比，发现客户的网络很变态，有两层 NAT。去掉一层，还是不行，又各种查，发现还是两层 NAT。知道有多郁闷了吧？

客户的网络是 PPPOE 上网，拨号后得到一个外网地址，该外网地址后面还套着一个“外网”地址，就是这个意思。

到此，在网络上我们实在是没有招了。

不过，好在，至少还能通 30 秒。

大家估计已经知道了，30 秒断的问题一般是超时未收到 ACK。

SIP 呼叫有三次握手：

主叫		被叫
INVITE	->	
	<-	200
ACK	->	

而且，SIP 协议规定，发送方发送 ACK 时，要往 200 OK 中携带的 Contact 地址上发。但由于客户端不知道自己的外网地址，在做被叫时 Contact 中填的是自己的内网地址，故 FreeSWITCH 向它发送的 ACK 客户端收不到，表现出来就是电话通了，客户端不断发 200，FS 不断发 ACK。最后客户端 30 秒后因为收不到 ACK 握手不成功发 BYE 挂断电话。

经过研究，我们找到三种办法解决这个问题：

- SIP 协议用 TCP。由于 TCP 协议是面向连接的，因此，不存在 ACK 发不到的问题。
- 在 FS 的 SIP Profile 中启用`aggressive-nat-detection`，这样 FS 会更努力地检测 NAT，也有效。不过，该参数可能有副作用。原因很简单，没有一种方法是万能的，可能误检。
- 用 ACL。自己写一个 ACL，包含客户端最外层的外网地址，如

在 SIP Profile 中使用`apply-nat-acl`（默认的是`nat.auto`, 改成你想要的）。

这种方法也有效，但是，需要手工维护 ACL。

NAT 无处不在，但我们都无法解决它。处理 NAT 问题不是一句话两句话的事，往往需要很多次的测试、抓包、分析才行。但遇到 NAT 问题也不要慌，好好回忆一下所学的知识，耐心排查，以不变应万变，总有解决办法。

5.3 一个 NAT 问题解决过程

关于这个问题我们还是按照故事来讲吧。

首先是有一个网友在 QQ 群里私下问我问题，一般来说，我是不会私下回答任何问题的。因此，我让他到群里问，后来就看到群里有人问一个在 NAT 环境下电话不通的问题（不知道是不是跟私聊的一个人，私聊跟群里对不上号，呵呵）。

问题是：如果不开视频，则经过 NAT 设备的通话是通的，一开视频就不通了。

看起来很奇怪，原因也可能有很多，因此，我提议他打开 SIP Trace（使用`sofia global siptrace on`命令）将抓包的数据放到 Pastebin 上。接着他问什么是 Pastebin，我说先看《FreeSWITCH 新手指南》，接着他问什么是《新手指南》，我说那得看<http://www.freeswitch.org.cn>。是的，我建议所有提问题的人都先看新手指南。最后他还真看了。

正说到这里，群里还有个人说他也有同样的问题。这就是在群里问问题比私下问的好处，你能找到与你有同样问题的人，他们会跟你分享经验。

当日志贴到 Pastebin 上以后，我看了一下，客户端发了 INVITE 以后，FreeSWITCH 回了 407 要求认证，这时候客户端回了 ACK，然后客户端应该重新发带认证信息的 INVITE。结果 FreeSWITCH 等了半天没有收到，因此报`WRONG_CALL_STATE`错误，呼叫失败（如果有对 SIP 呼叫流程不清楚的读者可以看《FreeSWITCH 权威指南》第 7 章）。

这里顺便说一下，有了这个日志我们马上就定位到问题了，所以，贴日志很重要。

出现这个问题的原因可能是客户端根本没回下一个 INVITE（这不大可能，但也不是不可能），或者是路由器等 NAT 设备将该 INVITE 包拦截或丢掉了。

由于现象是音频电话通，视频电话不通。而这两种电话的区别一般是 INVITE 包中的 SDP 不同，后者消息长度比较大一些。从收到的第一个 INVITE 包来看，大小已有 1265 字节（`recv 1265 bytes from udp/[10.0.10.1]:62468 at ...`），也有可能是后续的 INVITE 包更大而超过了 MTU，被路由器分包或导致了 FreeSWITCH 收不到完整的 INVITE 包。

因此我建议他把 SIP 换成 TCP 试试，TCP 是面向连接的协议，具有丢包重传机制，因而能保证 IP 包的完整。但他的回答是试了以后问题依旧（这个是个迷，因为没有进一步的确认）。

后来，笔者在忙别的事情，再回来看时群里已经有朋友帮他解决了，说是路由器有 ALG（Application Layer Gateway），他改了 SIP 服务端口就好了。ALG 是一个看起来很美好但到处都是 Bug 的 NAT 解决方案，因此在使用 FreeSWITCH 的时候，我们都建议关掉它。不过，不知道该问题中的 ALG 为什么只对视频请求有问题，音频却没问题。总之，根本原因还不知道。但从这个问题中我们学到一些东西：

- 有问题在公开的地方（如 QQ 群里）问，这样，会有更多的人帮助你；
- 问题所在有时跟你猜的不一样，所以，一定要让别人看到你的日志和 SIP Trace；
- 仔细问题问题，多测试、对比，缩小问题的范围；

- UDP 通信在有大数据包时（超过 MTU 时）不靠谱，试试 TCP；
- 关掉 SIP ALG，大多数情况下它只会帮倒忙；
- NAT 问题无处不在，没有统一的解决方案。学好基础知识才能以不变应万变。

后来，又得到那位提问者的反馈。最终原因还是因为 SIP 包过长的原因，他最后的解决方案不是使用 TCP，但是他在客户央上去掉了几个不用的音、视频编码，把 SIP 包（其实是 SDP 的部分）减小了一点，最后问题解决了。由此，我们又学到以下三点：

- 其实这印证了我最初的猜测，根本原因是 SIP 包过长；
- 有时候，一个问题解决了你却不知道为什么的时候，好好回忆你都做了什么，多验证一下，找到真正的原因，才能进步；
- 向回答你问题的人反馈很重要，要不然，大家就看不到最后这一段了。

其实，最后这个问题根本不是 NAT 问题。

第 III 部分 他山之石

第六章 OpenSIPS

我们准备拿出一两章专门写 OpenSIPS 的内容。

第七章 OpenSIPS 配置实例

7.1 负载均衡配置实例

上一节，我们讲了大规模的集群的总体结构。具体到 FreeSWITCH 来讲，它最典型的应用就是作为集群中的一个节点提供具体的服务，而前面的 DNS、路由器以及各种代理服务等则主要是负责将相应的注册和呼叫等请求分配到相应的 FreeSWITCH 服务器上进行处理。

下面我们来做一个负载均衡的实验。如图，以一台 OpenSIPS 和两个 FreeSWITCH(A、B)节点为例¹。OpenSIPS 位于前端，做 Proxy；FreeSWITCH 在后端，进行话务处理；它们都共享同一个数据库。OpenSIPS 用于分发 SIP 请求，而后面的 FreeSWITCH 节点负责处理实际的通话，RTP 媒体流仍然在用户 UA 和 FreeSWITCH 之间直接传送²。

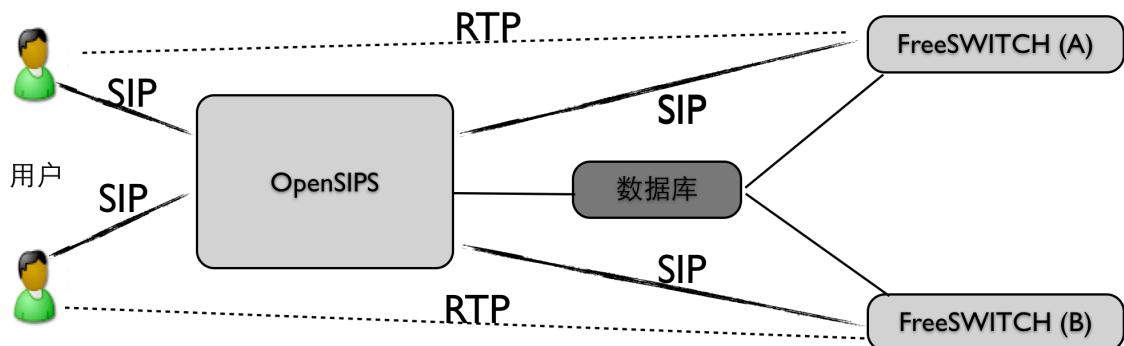


图 7.1: 图 15-15 OpenSIPS 与 FreeSWITCH 负载均衡示意图

OpenSIPS 是著名的 SIP Proxy，它的前身是 OpenSER，而 OpenSER 的前身则是 SER(SIP Express Router)。后来，由于版权的原因 OpenSER 更名为 OpenSIPS，而同时开发团队中的一部分人则 Fork 出了另外一个分支，叫做 Kamailio。我们无意于讨论它们之间的差别与优劣，总之它们的根是一样的，配置使用起来也差不多。

¹典型地，在生产应用中，OpenSIPS 也应该用双机做 HA，以避免单点故障。同理，后面的数据库也应该有相应的 HA 机制保证可靠。关于 OpenSIPS 及数据库的 HA 超出了本书的范围，在此，我们假设这些服务是可靠的。

²当然 OpenSIPS 也有配合 RTPProxy 的 RTP 代理方案，在此我们就不讨论了。

配置 FreeSWITCH

在配置 OpenSIPS 之前，我们应该配置两台一样的 FreeSWITCH (A 和 B)，分别测试注册和打电话保证都没问题。

然后，两个 FreeSWITCH 要连接一个共享的数据库，数据库可以参照 12.3.2 节中的 ODBC 配置，也可以使用原生的 PostgreSQL 驱动。简单起见，我们把数据库放到跟 OpenSIPS 相同的机器上，当然数据库也可以放到独立的数据库服务器上。

重要的一点是，两台 FreeSWITCH 要连接同一个数据库，并且，设置 `vars.xml` 中的 `domain` 参数为同一个名称，如 `sip.example.com`，将该 DNS 指向 OpenSIPS 服务器的 IP 地址。

修改 `domain`，将 `vars.xml` 中的

```
<X-PRE-PROCESS cmd="set" data="domain=$${local_ip_v4}"/>
```

改为：

```
<X-PRE-PROCESS cmd="set" data="domain=sip.example.com"/>
```

所以，这里需要用到 DNS 服务器支持。如果没有 DNS 服务器，也可以尝试将这里的 `domain` 设为 OpenSIPS 服务器的 IP 地址，或者，使用 15.7.1 节介绍的 Domain 知识进行相关配置。

安装配置 OpenSIPS

笔者在实验的时候用的是 OpenSIPS 1.8.2 版，如果版本号不同，配置文件也略有出入（其中『#』是注释起始符）。

OpenSIPS 的安装应该很简单，笔者在这里使用的是 PostgreSQL 数据库，如果你使用 `mysql`，也可以相应的进行替换。

使用如下命令进行安装：

```
cd opensips-1.8.2-tls  
make all include_modules="db_postgres"  
make include_modules="db_postgres" prefix="/usr/local" install
```

创建数据库 可以使用 `opensipsctl` 命令创建数据库。在使用之前需要修改如下配置文件 `/usr/local/etc/opensips/opensipsctlrc`, 以提供相应的参数。参数名称都很直观，在此就不多讲了，如：

```
# SIP_DOMAIN=opensips.org
DBENGINE=PGSQL
DBHOST=localhost
DBNAME=opensips
DBRWUSER=opensips
DBRPW="opensips"
DBROOTUSER="opensips"
```

然后运行以下命令创建数据库：

```
# opensipsdbctl create
```

在配置好所有参数之前，我们先将 OpenSIPS 启动到前台，这样便于调试。配置文件及解释如下，首先是基本的参数：

```
debug=5          # 输出详细的日志
memlog=1
fork=no         # 启动到前台，fork=yes 则启动到后台
children=2       # 在前台不起作用，如果 fork=yes，则控制启动多个进程
log_stderr=yes   # 将 LOG 输出到标准错误（控制台）
log_facility=LOG_LOCAL0 # 日志输出，后台模式有用
disable_tcp=yes  # 简单起见，我们只用 udp
disable_dns_blacklist = yes
auto_aliases=no
check_via=no
dns=off
rev_dns=off
listen=udp:192.168.1.118:7060    # 监听的 IP 地址和端口，可以是多行，但前台模式只支持第一行
```

然后是加载模块配置，在这里，我们使用 OpenSIPS 中的两个模块来做负载均衡，一个是 `dispatcher`，它用于均衡注册消息；另一个是 `load_balancer`，它用于均衡呼叫相关的消息。其它模块的详细说明参见相关文档。

```
mpath="/usr/local/lib64/opensips/modules/"      # 加载模块的路径
loadmodule "maxfwd.so"
```

```

loadmodule "sl.so"
loadmodule "db_postgres.so"
loadmodule "tm.so"
loadmodule "uri.so"
loadmodule "rr.so"
loadmodule "dialog.so"
loadmodule "mi_fifo.so"
loadmodule "signaling.so"
loadmodule "textops.so"
loadmodule "siptrace.so"
loadmodule "sipmsgops.so"          # 1.8 版本以后才有
loadmodule "dispatcher.so"        # 注册消息均衡(REGISTER)
loadmodule "load_balancer.so"      # 呼叫相关消息均衡(INVITE)

```

下面的模块的配置参数。模块参数配置一般是用`modparam`来指定的，其中第一个参数是模块名，第二个是参数名，不同的模块有不同的参数。如下：

```

# 默认的数据库的连接参数
db_default_url="postgres://opensips:opensips@localhost/opensips"
#fifo 模块的配置参数，其它模块类似
modparam("mi_fifo", "fifo_name", "/tmp/opensips_fifo")

modparam("dialog", "db_mode", 1)
<!-- modparam("dialog", "db_url", "postgres://opensips:opensips@localhost/opensips") -->

modparam("rr", "enable_double_rr", 1)
modparam("rr", "append_fromtag", 1)

modparam("siptrace", "trace_on", 1)

<!-- modparam("load_balancer", "db_url", "postgres://opensips:opensips@localhost/opensips") -->

<!-- modparam("dispatcher", "db_url", "postgres://opensips:opensips@localhost/opensips") -->

# dispatcher 模块的配置参数
modparam("dispatcher", "ds_ping_method", "OPTIONS")
modparam("dispatcher", "ds_ping_interval", 5)
modparam("dispatcher", "ds_probing_threshold", 2)
modparam("dispatcher", "ds_probing_mode", 1)

```

下面是路由的配置。路由的配置策略使用了类似 C 语言的脚本语言，看起来也比较直观，在此我们就不多解释了。路由配置内容如下：

```

route{
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
        exit;
    }

    if (!has_totag()) { # 初始请求
        record_route();
    } else { # 后续请求——遵循指定的路由
        loose_route();
        t_relay();
        exit;
    }

    if (is_method("CANCEL")) { # 处理 CANCEL 及重传
        if (t_check_trans()) t_relay();
        exit;
    }

    # 从现在开始，我们就只有初始请求了

    if (is_method("INVITE")) {
        # 如果是 INVITE 消息的话，将调用 load_balance 函数选择一台可用的后端节点
        if (!load_balance("1", "pstn", "1")) {
            send_reply("503", "Service Unavailable");
            exit;
        }
    } else if (is_method("REGISTER")) {
        # 如果是注册消息，则调用 dispatcher 模块的 ds_select_dst 函数分发请求
        # record_route();
        if (!ds_select_dst("1", "0")) {
            send_reply("503", "Service Unavailable");
            exit;
        }
    } else { # 我们暂时不转发其它消息
        send_reply("405", "Method Not Allowed");
        exit;
    }

    if (!t_relay()) { # 转发请求
        sl_reply_error();
    }
}

}

```

将上面的配置文件存为lb.conf(名字和路径可以任意，反正我们是测试)，然后运行如下命令启动 OpenSIPS：

```
# opensips -f lb.conf
```

接着就可以看到很多日志，启动另一个终端，在数据库中增加分发的目的服务器。`opensipsctl`程序可以动态控制各模块的行为，如，可以用如下命令在`dispatcher`中增加服务器：

```
# opensipsctl dispatcher addgw 1 sip:192.168.1.118 0 'FS1'
# opensipsctl dispatcher addgw 1 sip:192.168.1.119 0 'FS2'
```

然后启动一个 SIP 客户端注册到`sip.example.com`上，如果一切顺利的话将可以在 FreeSWITCH A 或 B 上收到注册消息。使用如下命令可以列出已注册的分机：

```
freeswitch> sofia status profile internal reg
```

由于我们的两个 FreeSWITCH 都使用了同一个数据库，因此可以看到，在两台服务器上显示的信息都是一样的，就如同是在一台上一样³。

另外，上面我们说到，对于呼叫相关的消息，由`load_balancer`这个模块来处理，我们这里可以直接将负载均衡配置数据直接写入数据库：

```
psql> insert into load_balancer (group_id, dst_uri, resources, description)
values (1,'sip:192.168.1.118', 'pstn=32', 'FS1');

psql> insert into load_balancer (group_id, dst_uri, resources, description)
values (1,'sip:192.168.1.119', 'pstn=32', 'FS1');
```

其中，`pstn=32`为服务器所能提供的资源，可以写成`'pstn=32;conf=20'`等，与`lb.conf`里`load_balance()`函数中的第二个参数对应。

假设有1000和1001两个分机都已经成功注册。我们可以做一个测试，呼叫流程如下：

1000 发 INVITE 请求到 OpenSIPS，OpenSIPS 转发 INVITE 到 A，A 查询数据库，到到 1001 的 Contact 地址，由于 A 是一个 B2BUA，因此重新发起一个新的 INVITE 请求直接到 1001 所在的地址，1001 接听后电话接通。

如果又有 1002 呼叫 1003，则`load_balance`模块会将 INVITE 路由到 B（因为 A 上已经有一路通话了），B 呼叫 1003，呼叫接通。

³SIP 注册的流程是：SIP 客户端发 REGISTER 到 OpenSIPS，然后 OpenSIPS 发到 A 或 B，A 或 B 将 SIP 客户端的 Contact 地址写入数据库。由于数据库是共享的，因此在 A 和 B 上看起来都一样。

这样就达到了负载均衡的目的。

所有配置都测试无误后可以把`1b.conf`改到`/usr/local/etc/opensips.conf`，并把其中的`fork`参数改为`yes`以使 OpenSIPS 启动到后台。

另外一个好处是，如果我们使用这种配置，可以随时添加或去掉一台 FreeSWITCH 服务器，而不影响使用。比方说在系统升级的时候，我们可以把所有的话务都定向到其中一台上，等待另一台负载为 0 后，进行升级。升级完成后，再将所有新的话务指到升级后的机器上，待旧机器话务降到 0 后即可升级旧的 FreeSWITCH。

在本例中，我们只是简单的介绍了 OpenSIPS 与 FreeSWITCH 配合使用的原理及基本配置，更多的示例可以参考http://wiki.freeswitch.org/wiki/Enterprise_deployment_OpenSIPS。

第 IV 部分 真刀真枪

我们来搭建一个真正的 FreeSWITCH 服务系统。全程记录。

第八章 搭建 SIPSIP 服务

在 FreeSWITCH 相关的开发测试中，我们经常需要测试与其它系统的对接，在特定的时候还需要对方返回特定的 SIP 消息如 180 或 183 等。当然，一般来说，我们会搭建另一台 FreeSWITCH 做为『其它系统』。我们很擅长此事。但是，用得多了，我们便想把它搭建成一个公共的服务，不仅方便我们，也方便大家。所以，该系统的建设构想便由此而来。

8.1 设计规划

最初，我们没有完整的建设规划。而是一步一步安装一步一步完善。因此，本书也基本按此步骤来记录。根据大家的反馈，我们再来做修正和完善。

8.2 安装 FreeSWITCH

我们首先申请了一台阿里云主机，以便安装我们的服务。主机的 IP 是『121.40.231.235』。虽然我们倾向于用 Debian8，但是阿里云不提供该版，因此，我们还是使用了 Debian7。

8.2.1 安装基本服务

在测试和使用时，我们经常用到一些效率工具，这里，先装上它们：

```
apt-get install tmux screen
```

8.2.2 安装依赖包

这些依赖包是编译安装 FreeSWITCH 要使用的。

```
apt-get install autoconf automake devscripts gawk g++ git-core libjpeg-dev \
libncurses5-dev libtool make python-dev gawk pkg-config libtiff5-dev \
libperl-dev libgdbm-dev libdb-dev gettext libssl-dev libcurl4-openssl-dev \
libpcre3-dev libspeex-dev libspeexdsp-dev libsqlite3-dev libedit-dev \
libldns-dev libtool-bin
```

8.2.3 编译安装 FreeSWITCH

虽然 FreeSWITCH1.6 已经布，但它在 Debian 7 上编译比较困难，而且，我们暂时也用不到 1.6，因此，我们安装 1.4 版本。

```
git clone https://stash.freeswitch.org/scm/fs/freeswitch.git
cd freeswitch
git checkout -b v1.4.21
```

初始化编译环境

```
./bootstrap.sh
```

配置编译环境

```
./configure --prefix=/usr/local/freeswitch
```

执行编译安装

```
make
make install
ln -sf /usr/local/freeswitch/bin/freeswitch /usr/bin/
ln -sf /usr/local/freeswitch/bin/fs_cli /usr/bin/
```

8.2.4 启动 FreeSWITCH

到这里，FreeSWITCH 已经安装完了。启动 FreeSWITCH：

```
freeswitch -nonat
```

由于我们的 FreeSWITCH 在公网上，因此我们使用-nonat 启动，以绕过启动期间的 NAT 检测，加速启动过程。

8.2.5 配置自启动脚本

在 FreeSWITCH 源代码目录内有各系统的启动脚本，我们要将脚本放到/etc/init.d/目录下，并设置启动用户和执行权限

```
cd freeswitch
cp ./debian/freeswitch-sysvinit.freeswitch.init /etc/init.d/freeswitch
cp ./debian/freeswitch-sysvinit.freeswitch.default /etc/default/freeswitch
chmod a+x /etc/init.d/freeswitch
useradd freeswitch
chown -R freeswitch:freeswitch /usr/local/freeswitch/{db,log,run,storage}
```

我们需要将/etc/init.d/freeswitch文件中对应的变量做些修改：

```
PATH=/sbin:/usr/sbin:/bin:/usr/bin
DESC=freeswitch
NAME=freeswitch
DAEMON=/usr/local/freeswitch/bin/freeswitch
USER=freeswitch
DAEMON_ARGS="-u $USER -ncwait"
CONFDIR=/usr/local/freeswitch/conf
RUNDIR=/usr/local/freeswitch/run
PIDFILE=$RUNDIR/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME
WORKDIR=/usr/local/freeswitch/lib
```

配置为开机自启动

```
chkconfig freeswitch on
```

8.3 配置最基本的服务

下面我们来一步步配置基本的 SIPSIP 服务。

8.3.1 清空 Dialplan

FreeSWITCH 的默认安装带了很多 Dialplan 的例子，可以说，装上 FreeSWITCH 后，基本不用什么特别的配置就是一个功能齐全的 PBX 了。但是，默认 Dialplan 目的主要是为了展示 FreeSWITCH 各种丰富的功能，在实际的应用中一般不要全部用到。因此，为了营造一个『干净』的环境，第一步，我们先清空所有的 Dialplan。然后再慢慢添加我们需要的服务。

清空后的 Dialplan 类似如下的样子：

default.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- http://wiki.freeswitch.org/wiki/Dialplan_XML -->
<include>
    <context name="default">

        <extension name="unloop">
            <condition field="${unroll_loops}" expression="^true$"/>
            <condition field="${sip_looped_call}" expression="^true$">
                <action application="deflect" data="${destination_number}"/>
            </condition>
        </extension>

    </context>
</include>
```

public.xml:

```
<!-- http://wiki.freeswitch.org/wiki/Dialplan_XML -->
<include>
    <context name="public">

        <extension name="unloop">
            <condition field="${unroll_loops}" expression="^true$"/>
            <condition field="${sip_looped_call}" expression="^true$">
                <action application="deflect" data="${destination_number}"/>
            </condition>
        </extension>

    </context>
```

我们保留了 unloop 部分是为了防止可能的 SIP 死循环。Dialplan 清空后，所有到达我们 FreeSWITCH 的呼叫由于在 Dialplan 中找不到对应的路由，因此都会呼叫失败。

8.3.2 密罐

公网上，一切都应该认为是不安全的。总会有人有意无意的使用你提供的服务，你的服务也会有意无意地遭到破坏。对付破坏者，通常有两种基本的策略：一种是高筑城墙，不要让破坏者进来；另一种更高级一点的策略是设个陷阱，给破坏者挖个坑，让他自己掉进去。当然，这两种策略其实不冲突，在实际应用中也经常混合使用。而且，为了更加迷惑破坏者，通过会把『坑』伪装的比较漂亮，尽量让破坏者多逗留一段时间，收集更多的信息更于更好地『逮』住他们。这种漂亮的『坑』又称为『密罐』。

因此，我们先来做一个密罐。我们无意去抓那些破坏者，相反，我们制作了各式各样的广告蜂蜜，供他们随时享用。

先修改internal.xml，设置允许任何人注册：

```
<param name="accept-blind-reg" value="true"/>
```

允许任何人呼叫：

```
<param name="accept-blind-auth" value="true"/>
```

修改 Context，来话时查找特定的 Dialplan。

```
<param name="context" value="honey"/>
```

创建相关的 Dialplan。在default.xml相同的目录中，创建honey.xml，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<include>
<context name="honey">

<extension name="unloop">
    <condition field="${unroll_loops}" expression="^true$"/>
    <condition field="${sip_looped_call}" expression="^true$">
        <action application="deflect" data="${destination_number}"/>
    </condition>
</extension>

<extension name="honey">
```

```

<condition field="destination_number" expression="^.*$">
    <action application="answer"/>
    <action application="playback" data="silence_stream://1000"/>
    <action application="playback" data="/usr/local/freeswitch/sounds/honey.wav"/>
</condition>
</extension>

</context>
</include>

```

在上面的配置中，我们配置了一个叫的 Dialplan Context。并增加了一条匹配任何被叫号码的路由（正则表达式『.*』）。该路由表示，不管呼叫任何被叫号码，都直接应答（**answer**），然后播放（**playback**）一秒静音（主要是为了等待 RTP 建立，防止后面的语音文件开头『丢字』）、然后播放一段广告。目前，我们的广告内容是：『您好，欢迎使用 FreeSWITCH 服务，广告位招租中…』。

当然，我们不会忘了录一个或用 TTS 转一个语音文件在/usr/local/freeswitch/sounds/目录下：

```
scp honey.wav root@talk:/usr/local/freeswitch/sounds/
```

重启 Profile：

```
freeswitch> sofia profile internal restart
```

或者，重启整个 sofia 模块：

```
freeswitch> reload mod_sofia
```

随便用一个账号注册到我们的服务器121.40.231.235上，呼叫123，看到如下日志：

```
[INFO] mod_dialplan_xml.c:635 Processing 1001 <1001>->123 in context default
[INFO] switch_core_state_machine.c:241 No Route, Aborting
```

不错。由于在 Dialplan 中找不到路由（我们已经清空了），上述呼叫失败了。我们从上述信息中看到 FreeSWITCH 对于这类『注册』过来的呼叫还是走默认的**default** Dialplan。因此，将下列内容添加到**default.xml**中：

```
<extension name="honey">
<condition field="destination_number" expression="^(.*)$">
<action application="transfer" data="$1 XML honey"/>
</condition>
</extension>
```

有了上面的路由配置，任何到**default** Context 的路由，都会转 (**transfer**) 到**honey** Context 进行处理。

好了，执行**reloadxml**，然后再打电话，所有电话都能听我们的广告了。

当然，我们上面绕了个圈从**default** Dialplan 转到**honey** Dialplan，不是没有道理的。如果主叫没有向我们注册 (REGISTER) 而直接向我们发起 INVITE 呼叫请求，那么，FreeSWITCH 就会直接从**honey** Dialplan Context 里查找路由了。你可以从另外的 FreeSWITCH 上用如下的命令直接向我们发送 INVITE：

```
freeswitch> originate sofia/external/test@121.40.231.235 &playback(/tmp/test.wav)
```

至此，我们的广告密罐就做好了。有人想租广告位吗？

8.3.3 180 和 183 服务

继续实现我们有用的服务。5060 端口是 SIP 的默认端口，是众所周知的，因此我们已经拿它做了密罐，希望有更多的人掉进去。但密罐并不是我们主要的业务。我们主要的业务还是提供正规的服务。我们的 180 和 183 服务将运行在 5080 端口上 (external Profile, 对呼叫不鉴权)。为了帮助 NAT 后面的客户端，我们先在**conf/sip_profiles/external.xml**中加入以下配置 (详情参见第5.1节)：

```
<param name="apply-nat-acl" value="nat.auto"/>
```

使用**sofia profile external rescan**命令使上述配置生效。

从 5080 端口进来的呼叫将走『public』 Dialplan，因此，我们把如下内容加到**dialplan/public.xml**中：

```
<extension name="180">
<condition field="destination_number" expression="^180$">
<action application="ring_ready"/>
<action application="sleep" data="3000"/>
```

```

<action application="answer"/>
<action application="playback" data="honey.wav"/>
</condition>
</extension>

<extension name="183">
<condition field="destination_number" expression="^183$">
<action application="ring_ready"/>
<action application="playback" data="honey.wav"/>
<action application="answer"/>
<action application="playback" data="honey.wav"/>
<action application="playback" data="honey.wav"/>
</condition>
</extension>

```

如果呼叫 180，则回 SIP 180 信令 (`ring_ready`)，等待 3 秒，然后应答，并放广告。

如果呼叫 183，则回 SIP 183 信令（我们直接用了`playback`）。183 信令是带媒体的，因此，在接听前客户端就能听到我们的广告（尚未计费）。播放完毕后，应答，继续放广告。

试呼

如果你想试呼一下，可以直接呼叫我们服务器的 5080 端口。如，你可以使用如图8.1的 X-Lite 配置：

由于 5080 端口无需注册，因此，用户名和密码之类的可以随意填。注意去掉『Register with domain and receive calls』前面的钩以防止 X-Lite 向 FreeSWITCH 注册。

配置完成后就可以在 X-Lite 上拨打 180 试呼测试了。

从 FreeSWITCH 中试呼

如果你在本地有个 FreeSWITCH，可以从你的 FreeSWITCH 中使用如下命令试呼：

```
originate sofia/external/180@121.40.231.235:5080 &playback(/tmp/sound_file.wav)
```

注意：如果你的 FreeSWITCH 处理 NAT 后面的话，在上面的命令中你应该使用`playback`而不是使用`echo`以便确保你能向我们的 FreeSWITCH 服务器发送 RTP 流以便帮助 NAT 穿越。

你也可以实现如下 Dialplan，你自己 FreeSWITCH 上的分机就可以呼叫 180 直接到我们的 FreeSWITCH 服务了：

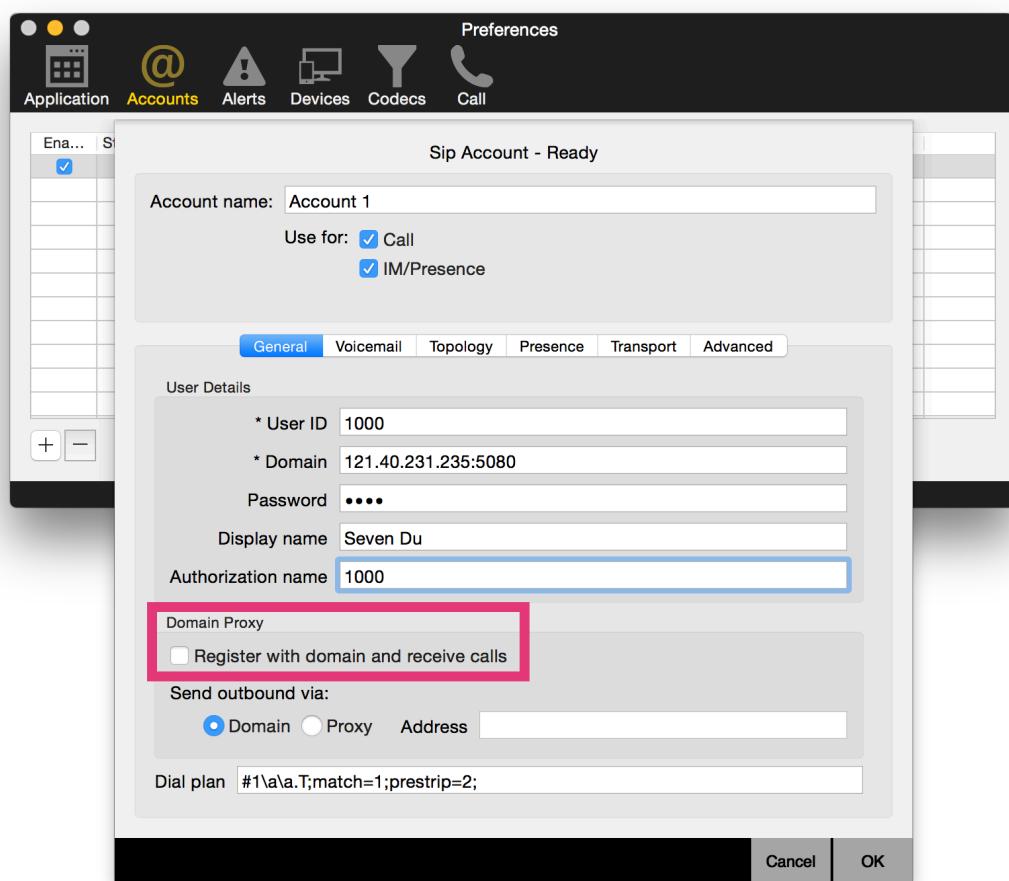


图 8.1: X-Lite 配置使用 180 服务

```
<extension name="my180">
<condition field="destination_number" expression="^180$">
<action application="bridge" data="sofia/external/180@121.40.231.235:5080"/>
</condition>
</extension>
```

8.3.4 DNS

在上面的例子中，我们直接使用了 IP 地址。但作为一项严肃的服务，没有个 DNS 显得我们也太不专业了。因此，下面我们将配置 DNS 指向我们的服务器。

首先在 DNS 服务器中添加 A 记录，将 sipsip.cn 指向我们的服务器。这样，大家就能更好的记住我们的服务器地址了。

如果你也有一个 FreeSWITCH，可以使用如下命令呼叫我们的 180 服务试一下：

```
freeswitch> originate sofia/external/180@sipsip.cn:5080 &echo
```

对了，sipsip.cn 好记，可还得记住 5080，不太理想。不过，不要着急。在 DNS 里，有专门的 SRV 记录用于解决这个问题。

我们又在 DNS 里添加了 SRV 记录。可以用dig工具查询 SRV 记录：

```
$ dig srv _sip._udp.sipsip.cn +short
20 50 5080 sip1.sipsip.cn.
20 50 5080 sip2.sipsip.cn.
$ dig srv _sip._tcp.sipsip.cn +short
20 50 5080 sip2.sipsip.cn.
20 50 5080 sip1.sipsip.cn.
```

我们可以看到。我们配置了两个 SIP 服务器，分别是sip1.sipsip.cn和sip2.sipsip.cn(这两个仍然是 A 记录)。这两台服务器可以分布式的分担请求。其中，dig输出结果中的第一个数字20表示记录的优先级；第二个数字50表示服务器的权重（以多大的概率向这台服务器发请求）；5080就是我们的服务器端口。

有了 SRV 记录 (SIP 服务会优先使用 SRV 记录，然后才是 A 记录)，就可以用如下方式使用我们的 180 服务了：

```
freeswitch> originate sofia/external/180@sipsip.cn &echo
```

当然，上面的例子是在 FreeSWITCH 做客户端情况下的例子。如果你的 SIP 客户端不支持 SRV 记录解析，那么，还是乖乖地记住我们的服务端口吧。

启用 SRV 后，我们可以从另外一台 FreeSWITCH（客户端）上做实验，简化的 SIP trace 如下：

```
freeswitch> originate sofia/external/180@sipsip.cn &echo

send 1172 bytes to udp/[121.40.231.235]:5080 at 21:34:33.708175:

-----  
INVITE sip:180@sipsip.cn SIP/2.0  
-----  
recv 369 bytes from udp/[121.40.231.235]:5080 at 21:34:33.782865:  
-----  
SIP/2.0 100 Trying  
-----  
recv 820 bytes from udp/[121.40.231.235]:5080 at 21:34:33.785259:  
-----  
SIP/2.0 180 Ringing
```

可见，客户端的 FreeSWITCH 成功地通过 SRV 记录查找到我们 180 服务的 IP 地址（121.40.231.235）和端口号（5080）并向其发送了 INVITE 请求。客户端收到了 180 Ringing 表示收到了我们的 180 服务。

当然，有时候，我们也会运气不好。我们的第二台 FreeSWITCH 服务器（sip2.sipsip.cn）还没有配置，万一 DNS 轮循时碰到第二台怎么办呢？我们先看下面的 SIP 消息：

```
freeswitch> originate sofia/external/180@sipsip.cn &echo

send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:24.962559:

-----  
INVITE sip:180@sipsip.cn SIP/2.0  
-----  
send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:25.963100:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0  
-----  
send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:27.963658:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0  
-----  
send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:31.964223:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0
```

```
send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:39.964435:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0  
  
send 1172 bytes to udp/[183.195.139.154]:5080 at 21:33:55.965819:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0  
  
send 1172 bytes to udp/[121.40.231.235]:5080 at 21:33:57.599331:  
-----  
INVITE sip:180@sipsip.cn SIP/2.0  
  
recv 369 bytes from udp/[121.40.231.235]:5080 at 21:33:57.643207:  
-----  
SIP/2.0 100 Trying  
  
recv 820 bytes from udp/[121.40.231.235]:5080 at 21:33:57.644749:  
-----  
SIP/2.0 180 Ringing
```

从上面的消息流程可以看出，客户端在发起呼叫时，从 DNS 记录中找到了两个 IP 地址，它向第一个 IP 地址（183.195.139.154）发送消息，服务器没任何反应。因而它使用 SIP 协议中的重传机制分别相隔 1、2、4、8、16 秒后重发 INVITE 请求。最后还是收不到响应，实在受不了了，它又向另外一个 IP 地址（121.40.231.235）发起呼叫请求，并最终收到了 180 Ringing 消息。

DNS 轮循在实际应用中也经常用到，如果其中一台服务器有问题（我们在此根本就没开），另一台仍然能够提供服务，只不过对某些通话来讲接续时间会有所延长（当然，在实际应用中如果知道某台服务器已经完蛋了，那么可以修改 DNS 把它从 DNS 列表里删除掉）。

DNS 轮循的好处是不同的服务器可以跨数据中心，实现异地多活。事实上，我们上面用的两台服务器分属于不同的城市和运营商（从上面的 IP 地址也可以看出来它们相差很多）。

8.3.5 更多服务

当然，如果我们费了半天劲只提供个 180 和 183 服务，那么我们就太小儿科了。有更多的业务等待我们去开发。

200 服务

200 服务在密罐里已经有了。我们只需要在 public Dialplan Context 里加一条路由转到密罐就可以了：

```
<extension name="200">
<condition field="destination_number" expression="^200$">
    <action application="trnasfer" data="200 XML honey"/>
</condition>
</extension>
```

302 服务

302 服务是一种临时转移服务。当有客户端向服务器发起呼叫时，服务器会通知客户端，我现在不能受理你的服务，但我告诉你另一台服务器，也许它能受理。这条通知消息就是在 302 消息里承载的。

我们的 302 服务如下：

```
<extension name="302">
<condition field="destination_number" expression="^302$">
    <action application="redirect" data="sip:302@sip.sip.cn"/>
</condition>
</extension>
```

当然，读者可以看到，302 消息又将主叫客户端引到了我们的蜜罐。

其它服务

其它的状态码都用 `response` 实现：

```
<extension name="other">
<condition field="destination_number" expression="^([456][0-9][0-9])$">
    <action application="response" data="$1"/>
</condition>
</extension>
```

8.3.6 Homer

开启了本服务后，便经常有人呼叫我到我们的服务器。为了统计到底有多少人呼叫我到我们的服务器，我们需要对信令做个统计。Homer 出场了。

Homer 一共有三部分组成。

- Captcher Agent：用于抓包，已内置于 FreeSWITCH 内部；

- Capther Server: 用于收集 Agent 发来的包存入数据库，使用 OpenSIPS 实现；
- WebHomer: Web 界面用于分析 SIP 包。

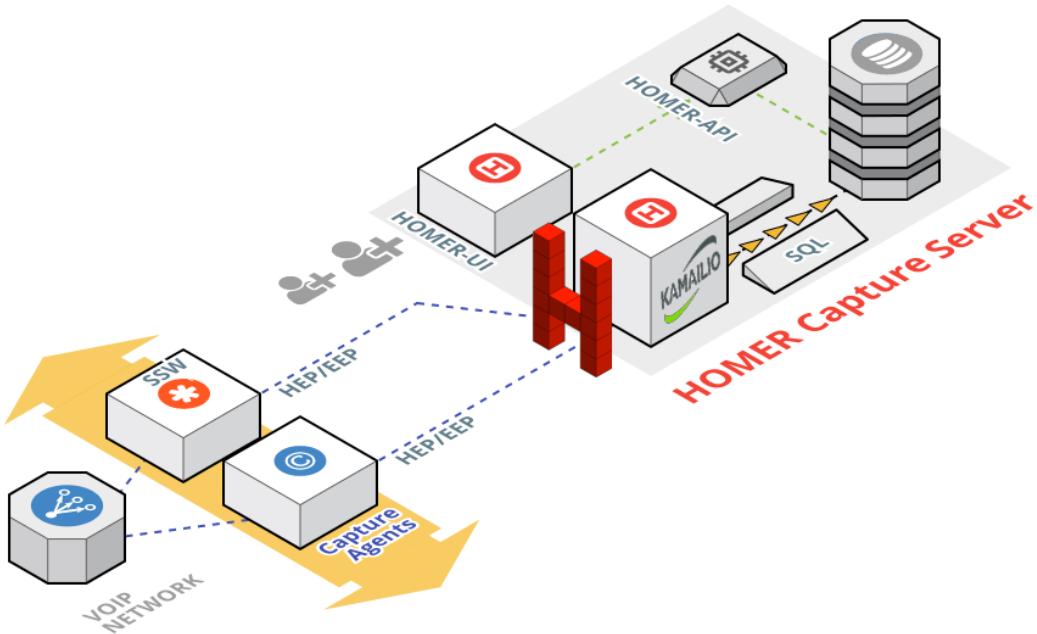


图 8.2: Homer

Homer 已升级为 Homer5，现在有了更便捷的安装方法。首先，准备一台空机器（就是刚装好操作系统别的什么也没装），然后执行如下命令即可：

```
bash < curl -s https://cdn.rawgit.com/sipcapture/homer-installer/master/homer_installer.sh )
```

默认的安装使用 PHP、Apache 和 Kamailio。笔者在 Debian 8 上安装完毕后显示如下信息，供参考：

```
*****
,;;;;
;;;;;;.. Congratulations! HOMER has been installed!
;;;;;;;;
;;;;; .. ;;; <----- INVITE -----
;;;;; .. ;;; ----- 200 OK ----->
;;;;; .. ;;; Your system should be now ready to rock!
;;;;; .. ;;; Please verify/complete the configuration
,;;; .. ;;; files generated by the installer below.
```

```
;;;;;;;;
      THIS SCRIPT IS PROVIDED AS-IS, USE AT
;;;;;;; YOUR *OWN* RISK, REVIEW LICENSE & DOCS
*****
* Verify configuration for HOMER-API:
  '/var/www/html//api/configuration.php'
  '/var/www/html//api/preferences.php'

* Verify capture settings for Homer/Kamailio:
  '/etc/kamailio/kamailio.cfg'

* Start/stop Homer SIP Capture:
  '/sbin/kamctl start|stop'

* Access HOMER UI:
  http://172.22.253.207 or http://172.22.253.207
  [default: admin/test123 or test1234]

* Send HEP/EEP Encapsulated Packets:
  hep://172.22.253.207:9060
*****
IMPORTANT: Do not forget to send Homer node some traffic! ;)
For our capture agents, visit http://github.com/sipcapture
For more help and information visit: http://sipcapture.org
*****
Installer Log saved to: /tmp/homer_installer.log
```

装完 Homer 后，便可以在 FreeSWITCH 中指定把所有消息都发送到 Homer 服务器。如，笔者在 `sofia.conf.xml` 中开启如下设置：

```
<param name="capture-server" value="udp:172.22.253.207:9060"/>
```

然后，在相关的 Profile (如internal.xml) 中，打开 Sip Capture:

```
<param name="sip-capture" value="yes"/>
```

新版 Homer 的界面非常简洁，可以很方便的查询 SIP 流程，如图8.3。

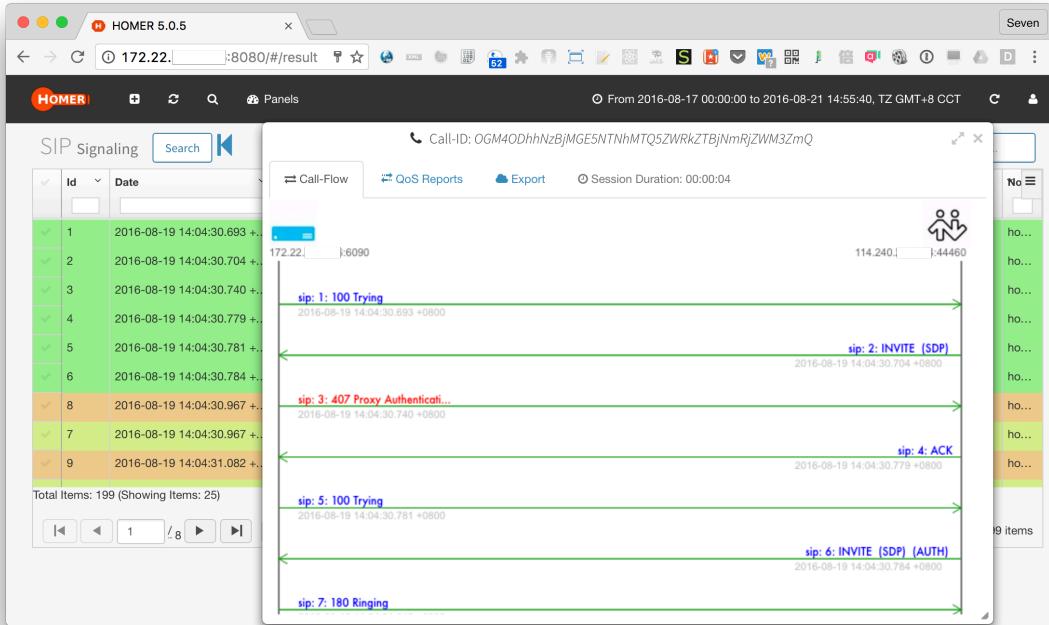


图 8.3: Homer 界面

8.3.7 CGRates

CDRates 是一个 CDR 和计费服务。

(TODO ...)

8.3.8 monit

一般来说，FreeSWITCH 是非常稳定的。但是，万一死掉怎么办呢？

- 我们要知道它死了
 - 尽快修复它
 - 事后，我们要知道它为什么死

monit 可以帮我们做到前面两点。…

安装 monit

我们这里为了简单方便，直接通过源进行安装。

```
apt-get install monit
```

配置 monit 的 web 服务

我们要先启用 monit 的 web 版控制台，需要编辑/etc/monit/monitrc文件，并修改一些内容

```
set httpd port 2812 and
# 此处的 localhost 要改为你所需要的监听地址
use address localhost # only accept connection from localhost
# 此处的 localhost 要改为允许访问的客户端地址
allow localhost      # allow localhost to connect to the server and
allow admin:monit    # require user 'admin' with password 'monit'
allow @monit         # allow users of group 'monit' to connect (rw)
allow @users readonly # allow users of group 'users' to connect readonly
```

改完后，我们尝试重启 monit 服务

```
/etc/init.d/monit restart
```

然后访问此地址：<http://localhost:2812>，输入用户和密码即可看到控制台页面。

为 FreeSWITCH 配置 monit 监控

在目录/etc/monit/monitrc.d/里面有可以参考的监控配置，我们这里用最简单的配置。创建并打开此vim /etc/monit/conf.d/freeswitch文件，然后添加以下内容

```
check process freeswitch with pidfile /usr/local/freeswitch/run/freeswitch.pid
group freeswitch
start program = "/etc/init.d/freeswitch start"
stop  program = "/etc/init.d/freeswitch stop"
if 5 restarts with 5 cycles then timeout
```

添加完毕后，执行/etc/init.d/monit reload重新加载配置文件。再次登录控制台页面，即可看到新加的 FreeSWITCH 服务。并且手动关闭 FreeSWITCH 后，会自动启动 FreeSWITCH 服务器。

8.3.9 iptables

控制对 SIP 端口发包的频率，防止过载。

(TODO …)

8.3.10 从 Debian7 升级到 Debian8

Debian Jessie (即 Debian8)是 2015 年 4 月份发布的，可是时间过去了一年多，还没有等到阿里云镜像更新。随着 FreeSWITCH 的发展，现在都 Debian8 都成标配了。因此，准备试一下能否从已有的 Debian7 升级。

首先，当然是备份。如果你有重要的数据，一定要备份。

备份 sources.list 文件

```
mv /etc/apt/sources.list /etc/apt/sources.list.bak
```

创建 sources.list 文件，内容如下：

```
deb http://mirrors.aliyuncs.com/debian/ jessie main non-free contrib
deb http://mirrors.aliyuncs.com/debian/ jessie-proposed-updates main non-free contrib
deb-src http://mirrors.aliyuncs.com/debian/ jessie main non-free contrib
deb-src http://mirrors.aliyuncs.com/debian/ jessie-proposed-updates main non-free contrib
```

更新软件包列表

```
apt-get update
```

升级到系统最新的版本

```
apt-get dist-upgrade
```

升级过程中会提示一些问题，一般选择默认选项即可。

升级完成后，系统会提示需删除一些不需要的软件包，这时候可执行：

```
apt-get autoremove
```

以删除旧的软件包，释放空间。

全部升级完成后建议重启下系统。

好了，我们可以享受 Jessie 了。

```
root@sipsip:~# cat /etc/debian_version
8.5
```

8.3.11 munin

munin 是一个轻量级的监控服务，笔者一直很喜欢这个工具。以下安装在 Debian 8 上进行。

这个服务安装起来很简单，首先安装 apache:

```
sudo apt-get install apache2
```

munin 需要后台的 CGI 脚本对监控图表进行缩放，安装相关的支持（如果不安装 munin 也能用，只是不支持缩放）：

```
sudo apt-get install -y libcgi-fast-perl libapache2-mod-fcgid
```

安装 munin：

```
sudo apt-get -y install munin
```

munin 的配置文件在 /etc/munin/，其中，有一个 apache 相关的配置 apache24.conf，实际上，该文件就是到 /etc/apache2/conf-available/munin.conf 的一个符号链接。

为了能从其它机器上访问 munin，打开相关的权限：

即，把

```
<Directory /var/www/munin>
    Require local
```

改成

```
<Directory /var/www/munin>
    Require all granted
```

同时，cgi 部分也改成如下的样子：

```
<Location /munin-cgi/munin-cgi-graph>
    Require all granted
    Options FollowSymLinks SymLinksIfOwnerMatch
    ...
</Location>
```

安装完整后，重启 apache：

```
systemctl restart apache2
```

打开浏览器访问：<http://ip/munin>

munin 是一个 client/server 的结构。在 server 端，也称为 master 端，有一个 cron 脚本定期去其它 client 端（也称为 node）采集数据。在上面的安装中，实际上我们把 master 和 node 都安装了。如果在同一局域网中有另外的 node，也可以直接加进来。如，在另外的 node 上执行：

```
sudo apt-get install -y munin-node
```

为了上 master 能访问 node，需要在 `/etc/munin/munin-node.conf` 中授权，如，以下配置允许来自 IP 1.2.3.4 访问该 node：

```
allow ^1\.2\.3\.4$
```

重启生效：

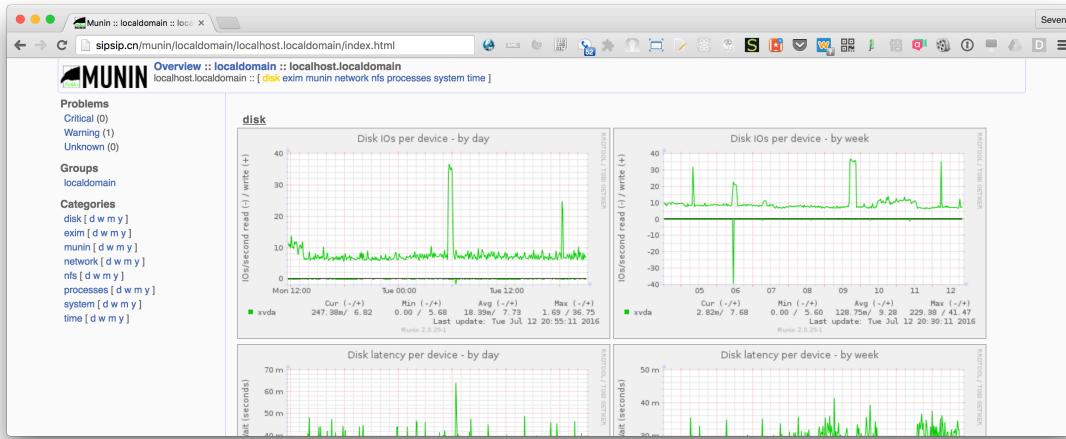


图 8.4: munin 截图

```
sudo systemctl restart munin-node
```

在 master 端，则需要修改 /etc/munin/munin.conf 加入一个 node，如

```
[MuninNode2]
address 1.2.3.5
use_node_name yes
```

默认情况下，master 每 5 分钟去 node 上检查一次。如果你当时看不到数据，别着急，喝杯咖啡等一会，就应该有数据了。

当然，如果你不使用 apache 而是用 nginx，也可以依照上面的 apache 配置进行配置。或参考：<http://guide.munin-monitoring.org/en/latest/example/webserver/nginx.html>。

第 V 部分 随笔杂谈

第九章 技术类话题

一些技术类的文章，暂没想好如何划分章节，先放到这里。

9.1 JSON API

FreeSWITCH早在1.4时代就有了JSON API，然而，却没有引起大家的重视。其实，JSON API的设计有更好的结构化和扩展性，配合HTTP和Websocket接口，必将大有作为。

JSON API，顾名思意，它的输入是JSON，输出也是JSON。

在Websocket接口中，FreeSWITCH提供了JSON API的入口。不过，为了测试方便，FreeSWITCH又提供了一个json API命令，也就是说，你可以直接在命令行上测试JSON API。

如：

```
freeswitch> json {"command":"status", "data": null}

{"command":"status", "data":null,"status":"success","response": {"systemStatus": "ready", "uptime": {"years":0, "days":0, "hours":0, "minutes":1, "seconds":30, "milliseconds":147, "microseconds":640}, "version": "1.4.19 git dd64594 2015-06-05 22:36:51Z 64bit", "sessions": {"count": {"total":0, "active":0, "peak":0, "peak5Min":0, "limit":5000}, "rate": {"current":0, "max":100, "peak":0, "peak5Min":0}, "idleCPU": {"used":0, "allowed":100}, "stackSizeKB": {"current":240, "max":8192}}}
```

当然，上面的结果比较乱，在实际测试时，我们可以用如下的命令看到比较『好看』的版本：

```
$ fs_cli -x 'json {"command":"status", "data": null}' | python -m json.tool
{
    "command": "status",
    "data": null,
    "response": {
        "idleCPU": {
            "allowed": 100,
```

```
        "used": 0
    },
    "sessions": {
        "count": {
            "active": 0,
            "limit": 5000,
            "peak": 0,
            "peak5Min": 0,
            "total": 0
        },
        "rate": {
            "current": 0,
            "max": 100,
            "peak": 0,
            "peak5Min": 0
        }
    },
    "stackSizeKB": {
        "current": 240,
        "max": 8192
    },
    "systemStatus": "ready",
    "uptime": {
        "days": 0,
        "hours": 0,
        "microseconds": 663,
        "milliseconds": 305,
        "minutes": 0,
        "seconds": 44,
        "years": 0
    },
    "version": "1.4.19 git dd64594 2015-06-05 22:36:51Z 64bit"
},
"status": "success"
}
```

上面的命令使用`fs_cli -x`执行命令，并将输出结果通过管道输送到一个 Python 脚本里，在 Python 中有一个`json.tool` 模块可以将 JSON 输出的漂亮些。

从上面可以看到，我们使用`json`是一个 API 命令，它的参数是一个 JSON 字符串。其中，字符串表示一个 JSON 对象，该对象的`command`属性是`status`，`data`是空，它将调用`status`这个 JSON API 命令。实际上，它跟普通的`status`命令执行结果是一样的，只是，它的输出是一个标准的 JSON，更便于程序解析。

再试一个：

```

fs_cli -x 'json {"command":"channelData", "data": {"uuid": "9ae30fab-0c49-4d77-8ce2-cec634a9b8d6"} }' | \
python -m json.tool
{
    "command": "channelData",
    "data": {
        "uuid": "9ae30fab-0c49-4d77-8ce2-cec634a9b8d6"
    },
    "response": {
        "channelData": {
            "app_log": {
                "applications": [
                    {
                        "app_data": "outside_call=true",
                        "app_name": "set"
                    },
                    {
                        "app_data": "RFC2822_DATE=Sun, 19 Jul 2015 09:58:06 +0800",
                        "app_name": "export"
                    },
                    {
                        "app_data": "9196 XML default",
                        "app_name": "transfer"
                    },
                    {
                        "app_data": "sound_prefix=/usr/local/freeswitch/sounds/zh/cn/link",
                        "app_name": "set"
                    },
                    {
                        "app_data": "language=zh",
                        "app_name": "set"
                    },
                    {
                        "app_data": "default_language=zh",
                        "app_name": "set"
                    },
                    {
                        "app_data": "insert/192.168.7.6-spymap/1000/9ae30fab-0c49-4d77-8ce2-cec634a9b8d6",
                        "app_name": "hash"
                    },
                    {
                        "app_data": "insert/192.168.7.6-last_dial/1000/9196",
                        "app_name": "hash"
                    },
                    {
                        "app_data": "insert/192.168.7.6-last_dial/global/9ae30fab-0c49-4d77-8ce2-cec634a9b8d6",
                        "app_name": "hash"
                    },
                    {
                        "app_data": "RFC2822_DATE=Sun, 19 Jul 2015 09:58:06 +0800",
                        "app_name": "set"
                    }
                ]
            }
        }
    }
}

```

```
        "app_name": "export"
    },
{
    "app_data": "",
    "app_name": "answer"
},
{
    "app_data": "",
    "app_name": "echo"
}
]
},
"callflow": {
    "caller_profile": {
        "ani": "1000",
        "aniii": "",
        "caller_id_name": "1000",
        "caller_id_number": "1000",
        "chan_name": "sofia/external/1000@192.168.7.6:5080",
        "context": "public",
        "destination_number": "9196",
        "dialplan": "XML",
        "network_addr": "192.168.7.6",
        "rdnis": "",
        "source": "mod_sofia",
        "username": "1000",
        "uuid": "9ae30fab-0c49-4d77-8ce2-cec634a9b8d6"
    },
    "dialplan": "XML",
    "extension": {
        "applications": [
            {
                "app_data": "outside_call=true",
                "app_name": "set"
            },
            {
                "app_data": "RFC2822_DATE=${strftime(%a, %d %b %Y %T %z)}",
                "app_name": "export"
            },
            {
                "app_data": "9196 XML default",
                "app_name": "transfer"
            }
        ],
        "name": "outside_call",
        "number": "9196"
    },
    "profile_index": "1",
    "times": {

```

```
        "answered_time": "0",
        "bridged_time": "0",
        "created_time": "1437271086371311",
        "hangup_time": "0",
        "hold_accum_time": "0",
        "last_hold_time": "0",
        "profile_created_time": "1437271086371311",
        "progress_media_time": "0",
        "progress_time": "0",
        "resurrect_time": "0",
        "transfer_time": "1437271086371311"
    }
},
"channel_data": {
    "caps": "1=1;2=1;3=1;4=1;5=1;6=1",
    "direction": "inbound",
    "flags": "0=1;1=1;3=1;37=1;43=1;53=1;74=1;111=1;112=1;116=1;118=1",
    "state": "CS_EXECUTE",
    "state_number": "4"
},
"core-uuid": "5c11bb8a-1f4f-4301-8541-b49e36866c08",
"switchname": "seven.local",
"variables": {
    "DP_MATCH": "ARRAY::DELAYED NEGOTIATION|:DELAYED NEGOTIATION",
    "RFC2822_DATE": "Sun, 19 Jul 2015 09:58:06 +0800",
    "advertised_media_ip": "192.168.7.6",
    "call_uuid": "9ae30fab-0c49-4d77-8ce2-cec634a9b8d6",
    "channel_name": "sofia/external/1000@192.168.7.6:5080",
    "current_application": "echo",
    "default_language": "zh",
    "direction": "inbound",
    "dtmf_type": "rfc2833",
    "endpoint_disposition": "ANSWER",
    "ep_codec_string": "opus@48000h@20i@2c,PCMA@8000h@20i@64000b,PCMU@8000h@20i@64000b",
    "export_vars": "RFC2822_DATE,RFC2822_DATE",
    "language": "zh",
    "local_media_ip": "192.168.7.6",
    "local_media_port": "22020",
    "max_forwards": "69",
    "original_read_codec": "opus",
    "original_read_rate": "48000",
    "outside_call": "true",
    "read_codec": "opus",
    "read_rate": "48000",
    "recovery_profile_name": "external",
    "remote_media_ip": "192.168.7.6",
    "remote_media_port": "62544",
    "rtp_2833_recv_payload": "101",
    "rtp_2833_send_payload": "101",
}
```

```
"rtp_audio_recv_pt": "123",
"rtp_last_audio_codec_string": "opus@48000h@20i@1c",
"rtp_local_sdp_str": "v=0\no=FreeSWITCH 1437249066 1437249067 IN IP4 192.168.7.6\\ns=FreeSWITCH\\nc=IN I
"rtp_use_codec_channels": "1",
"rtp_use_codec_fmtpt": "useinbandfec=1",
"rtp_use_codec_name": "opus",
"rtp_use_codec_ptime": "20",
"rtp_use_codec_rate": "48000",
"rtp_use_codec_string": "OPUS,G722,PCMU,PCMA,GSM",
"rtp_use_pt": "123",
"rtp_use_ssrc": "2872309422",
"rtp_use_timer_name": "soft",
"session_id": "1",
"sip_call_id": "MDk3N2Y3ZGQ2Njd1ZjY20TUwZmUwZjBmNTM5NDQ5NTM",
"sip_contact_host": "192.168.7.6",
"sip_contact_params": "transport=tcp",
"sip_contact_port": "52290",
"sip_contact_uri": "1000@192.168.7.6:52290",
"sip_contact_user": "1000",
"sip_cseq": "1",
"sip_from_display": "1000",
"sip_from_host": "192.168.7.6",
"sip_from_port": "5080",
"sip_from_tag": "2c58757a",
"sip_from_uri": "1000@192.168.7.6:5080",
"sip_from_user": "1000",
"sip_from_user_stripped": "1000",
"sip_full_from": "\"1000\" <sip:1000@192.168.7.6:5080>;tag=2c58757a",
"sip_full_to": "<sip:9196@192.168.7.6:5080>;tag=j8yaKv8ta460S",
"sip_full_via": "SIP/2.0/TCP 192.168.7.6:35100;branch=z9hG4bK-d8754z-7d03285fb4d0697a-1---d8754z;rpor
"sip_local_network_addr": "192.168.7.6",
"sip_network_ip": "192.168.7.6",
"sip_network_port": "52290",
"sip_received_ip": "192.168.7.6",
"sip_received_port": "52290",
"sip_recover_via": "SIP/2.0/TCP 192.168.7.6:35100;branch=z9hG4bK-d8754z-7d03285fb4d0697a-1---d8754z;rpor
"sip_req_host": "192.168.7.6",
"sip_req_port": "5080",
"sip_req_uri": "9196@192.168.7.6:5080",
"sip_req_user": "9196",
"sip_to_host": "192.168.7.6",
"sip_to_port": "5080",
"sip_to_tag": "j8yaKv8ta460S",
"sip_to_uri": "9196@192.168.7.6:5080",
"sip_to_user": "9196",
"sip_user_agent": "Bria 3 release 3.5.5 stamp 71243",
"sip_via_host": "192.168.7.6",
"sip_via_port": "35100",
"sip_via_protocol": "tcp",
```

```

        "sip_via_rport": "52290",
        "sofia_profile_name": "external",
        "sound_prefix": "/usr/local/freeswitch/sounds/zh/cn/link",
        "switch_r_sdp": "v=0\r\no=- 1437271086360654 1 IN IP4 192.168.7.6\r\ns=Bria 3 release 3.5.5 stamp 7124
        "transfer_history": "1437271086:4bfede71-17ca-43ce-aee7-41071eccd5b0:bl_xfer:9196/default/XML",
        "transfer_source": "1437271086:4bfede71-17ca-43ce-aee7-41071eccd5b0:bl_xfer:9196/default/XML",
        "uuid": "9ae30fab-0c49-4d77-8ce2-cec634a9b8d6",
        "write_codec": "opus",
        "write_rate": "48000"
    }
}
},
"status": "success"
}

```

上面的命令是一个 channelData 命令，执行它是要保证当前的 FreeSWITCH 里有一个活动的 Channel，可见，上面的命令类似于传统的 `uuid_dump`，可以得到一个 Channel 相关数据的 JSON 表示。

不过，现在直接可以测试的 JSON API 不多，所以，FreeSWITCH 又提供了一个超级的 JSON API，该 API 可以把所有的传统的 API 命令转换成一个统一的 JSON API。这样说起来有点绕，我们先来试一下：

status 命令：

```

$ fs_cli -x 'json {"command":"fsapi", "data": {"cmd": "status"}}' | python -m json.tool
{
    "command": "fsapi",
    "data": {
        "cmd": "status"
    },
    "response": {
        "message": "UP 0 years, 0 days, 0 hours, 6 minutes, 29 seconds, 871 milliseconds, 376 microseconds\nFreeSWITCH (Version 1.4.19 git dd64594 2015-06-05 22:36:51Z 64bit) is ready\n1 session(s) since start\n1 session(s) - peak 1, last 5min 1\n0 session(s) per Sec out of max 100, peak 1, last 5min 0\n5000 session(s) max\nmin idle cpu 0.00/100.00\nCurrent Stack Size/Max 240K/8192K\n"
    },
    "status": "success"
}

```

sofia status 命令：

```

fs_cli -x 'json {"command":"fsapi", "data": {"cmd": "sofia", "arg": "status"}}' | python -m json.tool
{

```

```

"command": "fsapi",
"data": {
    "arg": "status",
    "cmd": "sofia"
},
"response": {
    "message": "Name\t      Type\t          Data\tState\n"
    "-----\n"
    "192.168.7.6\t alias\t           internal\tALIASED\n"
    "external\tprofile\t       sip:mod_sofia@192.168.7.6:5080\tRUNNING (1)\n"
    "external::example.com\tgateway\t      sip:joeuser@example.com\tNOREG\n"
    "-----\n"
    "2 profiles 1 alias\n"
},
"status": "success"
}

```

从上面的 JSON API 可以看出，它将原来的 `status` 和 `sofia status` 这样的命令包装在了 JSON 中的 `data` 数据中，而得到的执行结果也是原来的文本字符串，放在结果 JSON 数据中的 `response` 里。

好了，关于 JSON API 就先讲这些，有兴趣的先练习一下，看看源代码，找出更多 JSON API。

9.2 视频直播

大家都知道，如今，视频直播比较火啊。

今天，在 FreeSWITCH 精英群里分享了 FreeSWITCH 做视频直播相关的技术。

首先，要做直播就得有好机器。笔者买了一台阿里云的主机，买的是按量付费的，4 核 4G 内存，装 Debian 8.0.4（话说阿里云终于有了 Debian 8 的镜象了）。

FreeSWITCH 直接有针对 Debian 8 的安装包，不过笔者通常是编译安装，今天还是编译安装的。

安装 FreeSWITCH 依赖：

```

apt-get install -y build-essential automake autoconf 'libtool-bin|libtool' wget curl python uuid-dev zlib1g-dev
'libjpeg8-dev|libjpeg62-turbo-dev' libncurses5-dev libssl-dev libpcre3-dev libcurl4-openssl-dev libldns-dev
libedit-dev libspeexdsp-dev libspeexdsp-dev libsqlite3-dev perl libgdbm-dev libdb-dev bison libvlc-dev
pkg-config libsndfile1-dev libopus-dev lua5.2-dev

```

更多

```

apt-get install -y yasm nasm libavformat-dev libswscale-dev

```

常用的工具

```
apt-get install -y git htop tcpdump
```

Clone FreeSWITCH，使用 master 版，使用国内的镜象，比官网的要快：

```
git clone http://git.coding.net/dujinfang/FreeSWITCH.git
```

编译安装：

```
./bootstrap.sh  
./configure  
make -j  
make install  
make sounds-install  
make moh-install
```

安装 mod_av

```
cd freeswitch.git  
cd src/mod/applications/mod_av  
make  
make install
```

修改几个地方：

首先，修改 `conf/dialplan/default.xml`，找到 1234，把 1234 改成任何其它的字符串，以防止 FreeSWITCH sleep 10 秒，表现就是打电话慢。

另外，修改 `conf/autoload_configs/conference.conf.xml`，把里面的 1920x1080 改成 1280x720。节省点机器资源和带宽。

FreeSWITCH 源代码里有个 `html5/verto/verto_communicator` 目录，安装 verto

```
cd html5/verto/verto_communicator  
.debian8-install.sh
```

执行完后将生成 dist 目录。

话说，上面说归说，笔者不是那么做的。原因如下：

Veto communicator 是用 Angular 框架开发的，依赖很多 node.js 相关的东西。很多，在阿里云的服务器上，下载不下来。当然，这不是阿里云本身的问题，你懂的。

所以，笔者早就在其它电脑上编译好了 dist。将里面的内容 copy 到 /root/www/vc 目录下，备用。

修改 conf/autoload_configs/verto.conf.xml，把下列内容加到 profile 里（前面）：

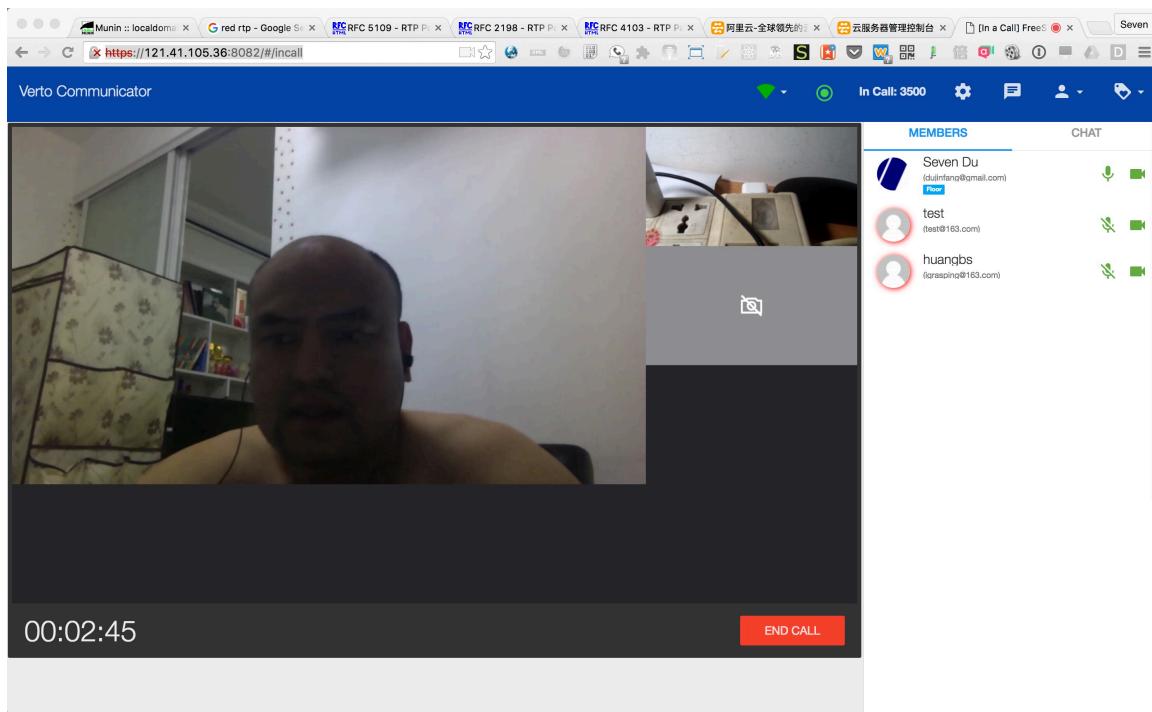
然后，mod_verto 就成了一个 http 服务器了。

启动 FreeSWITCH，load mod_av

用 Chrome 浏览器访问：<https://121.41.105.36:8082/>

每一次访问，需要输入你的姓名和电子邮件。

呼叫 3500，就可以进入视频会议了。

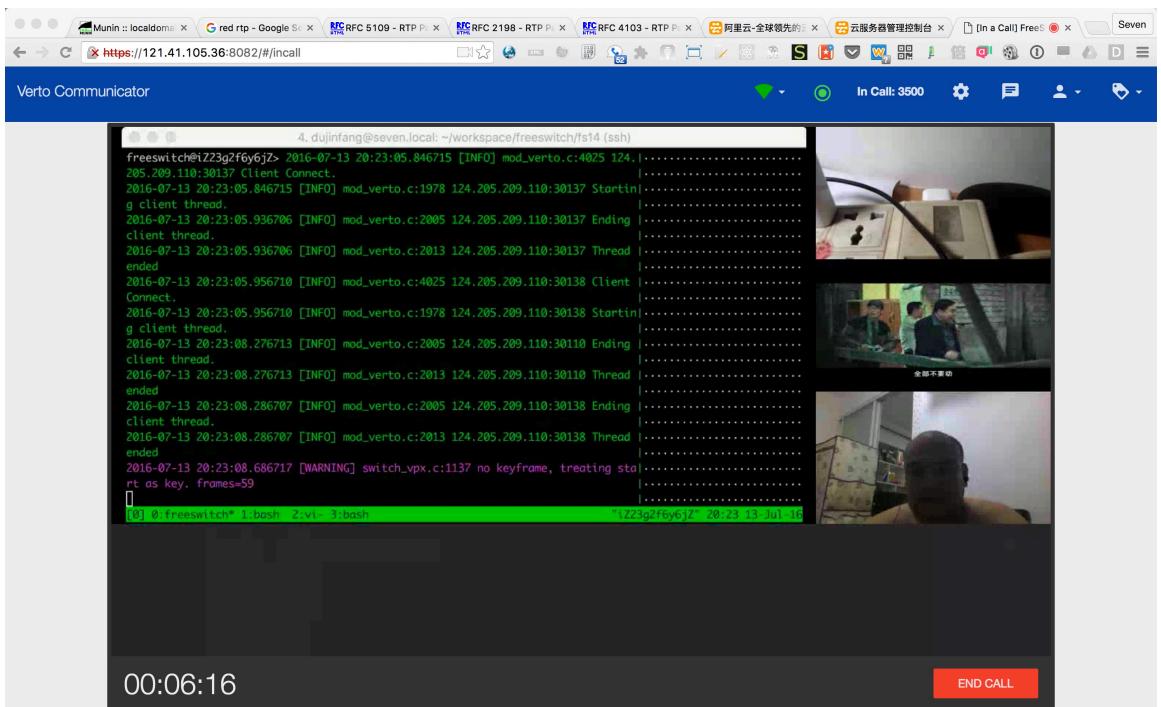
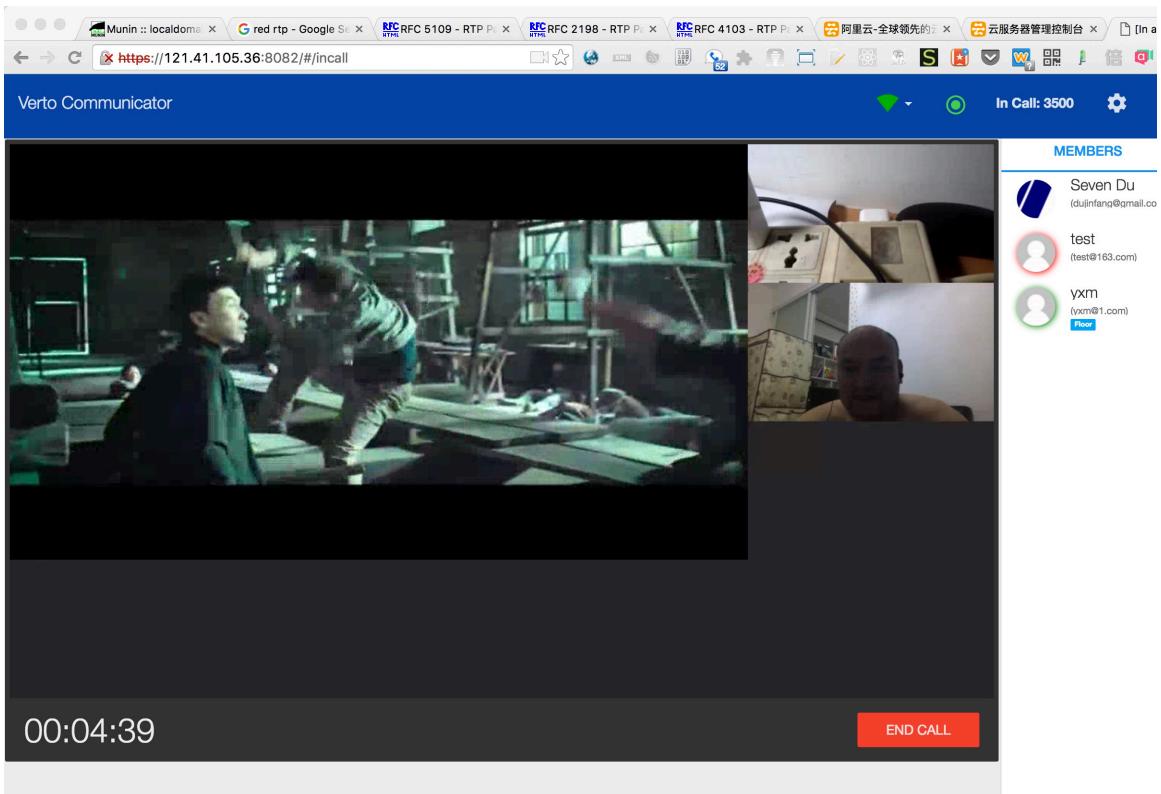


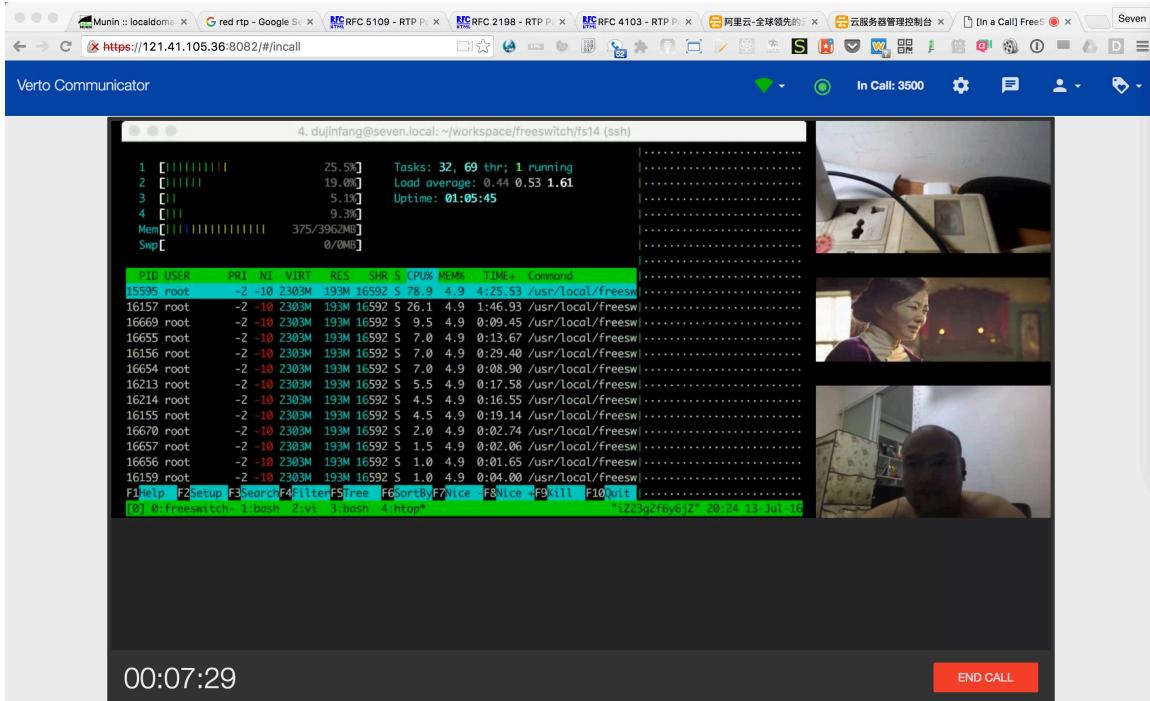
杜老师视频会议竟然不穿衣服！网警管不管？

还有人在视频会议里放起了视频（虚拟摄像头）：

我们自己的视频会议做好了，可以共享屏幕了，以后，再也不用 QQ 那么难用的屏幕共享功能了。下图，杜老师正在共享 FreeSWITCH 控制台。

htop 看下机器性能





有同学问，上面的会议用的什么编码？答：VP8，因为 Chrome 默认支持 VP8。

那 Chrome 支持 H264 吗？有一个选项可以试试启动 Chrome 时在命令行里加上：

```
--enable-features=WebRTC-H264WithOpenH264FFmpeg
```

好吧，上述会议普通 SIP 客户端也是可以呼进来的。

这也算直播？

不算，上面是视频会议。视频会议的流是双向的。而直播流是单向的。即，大部分人只是“看”。

其实，直播最关键的不是技术问题，而是，带宽和 CDN。

当前最流行的直播协议是 Adobe 的 RTMP 和 Apple 提出的 HLS。RTMP 是 Flash 时代提出的，HLS 的全称是 Http Live Stream，是基于 HTTP 的。这两种协议都适合 CDN (Content Delivery Network)。CDN 的作用是，如果有大量用户同时看直播，同一地区的用户会就近的访问离自己最近的 CDN 网络中的服务器，而 CDN 网络负责内容的分发和缓存。

好了，CDN 已超出我们讨论的范围了。再回到 FreeSWITCH。

FreeSWITCH 本身也可以通过 mod_rtmp 提供一路视频流，但是 HLS 就不行了。现在直播通用的解决方案是使用 FMS、Wowza、Nginx (with rtmp 模块) 或 crtmpserver。其它的配置都比较复杂，笔者发现 ctmpserver 还是比较省心的。装一个：

```
apt-get install ctmpserver
```

启动

```
/etc/init.d/crtmpserver start
```

服务启动后，可以给它从 FreeSWITCH 里推一路 RTMP 流试一下了：

```
conference 3500-121.41.105.36 record rtmp://121.41.105.36/live/stream1
```

是的，FreeSWITCH 通过 mod_av，使用录像（record）的功能将视频发送（推）到远程的 rtmp 服务器上。当然，这里的远程 rtmp 服务器就是 ctmpserver 提供的。

这个流也是可以“拉”的，即可以直观看。用 VLC 视频播放软件打开上述地址，理论上就能播放。但理论归理论，笔者的 VLC 日志中显示如下错误：

```
Server error: call to function _checkbw failed
```

错误的原因可能是 ctmpserver 不支持 _checkbw 函数造成的，也可能是笔者的 VLC 2.0 版好久没升级的缘故。总之，我们放弃了研究这一问题。

RTMP 流播放器有很多，比如，我们用 Google 随便就找到一个：

```
https://www.hlsplayer.net/rtmp-player
```

打开上述地址后，输入我们的视频流的路径：rtmp://121.41.105.36/live/stream1 就可以观看了。

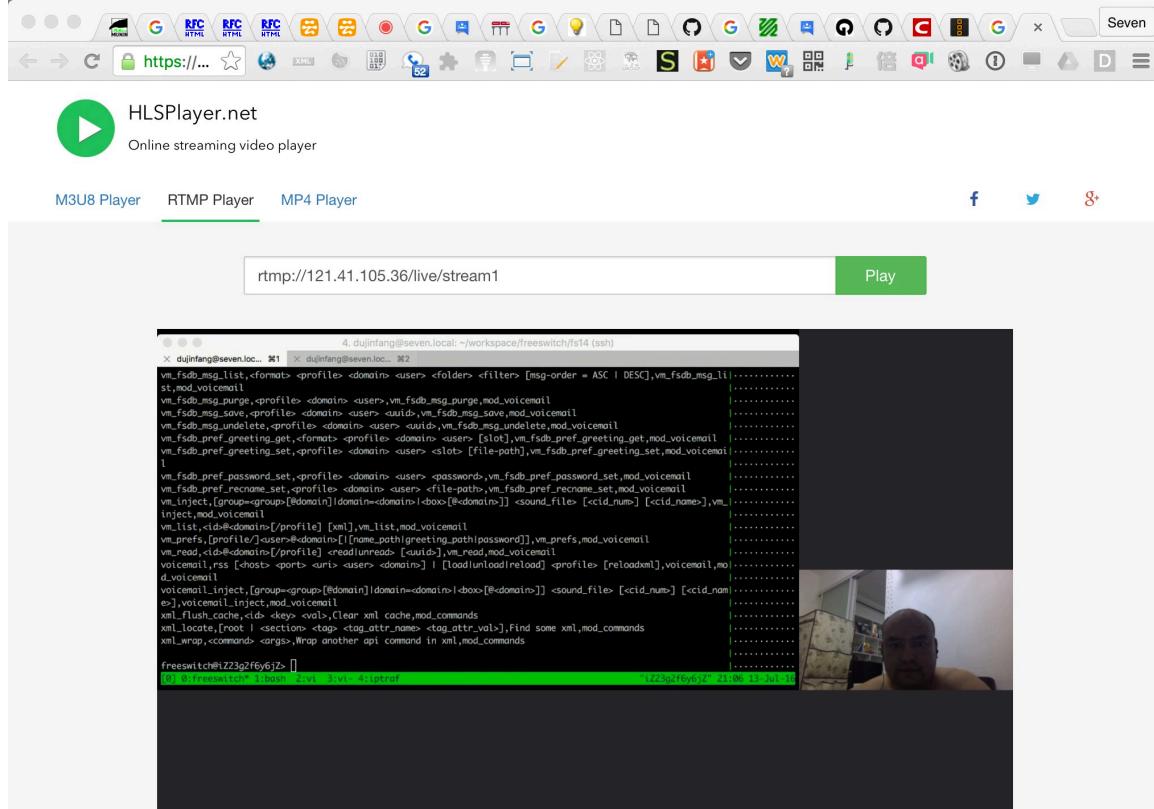
好玩吧？好吧，今天，就玩到这里了。小伙伴们退出会议了，就我一个人还在玩。

总结一下，FreeSWITCH 可以做直播吗？大概就是这个样子的。

上述 IP 地址已经打不开了。因为笔者买的是按量付费的，用完释放了。

盘点一下，大致花了这么多钱：

供参考：



流水号	日期	备注	收入	支出
52433084500673	2016-07-13 22:55:50	消费 云服务器ECS-按量付费	-	¥1.29
52433208420673	2016-07-13 21:55:47	消费 云服务器ECS-按量付费	-	¥1.90
52430878780673	2016-07-13 20:55:45	消费 云服务器ECS-按量付费	-	¥1.11



9.3 视频直播 Nginx 篇

阿海/2016.07.19

上一篇文章发过后，有同学问在 Nginx 下如何视频直播，有的同学就自己研究并写了博客，我们把他的全文转载到这里，供大家学习。同时欢迎大家来投稿，到时赞赏的钱都付稿费噢！

从下面的网址分别下载 nginx 和 nginx-rtmp-module：

<http://nginx.org/en/download.html>

我下载了 1.10.1 的版本。

<https://github.com/arut/nginx-rtmp-module>

我目前的现状是已经安装好了 Nginx。版本号为 nginx/1.10.1，需要在它的基础上安装 RTMP 这个模块，可以使用如下的方法实现：

```
cd /date/tools
git clone https://github.com/arut/nginx-rtmp-module.git (克隆 nginx-rtmp-module)
cd /date/tools/nginx-1.10.1
./configure --prefix=/usr/local/nginx --add-module=/date/tools/nginx-rtmp-module --with-http_ssl_module --with-debug
```

```
make  
make install
```

2. Nginx 配置

<https://github.com/arut/nginx-rtmp-module/wiki/Directives>

在原有的 /usr/local/nginx/conf/nginx.conf 中加入如下配置

```
rtmp {  
  
    server {  
  
        listen 1935;  
  
        chunk_size 4000;  
  
        #HLS  
  
        # For HLS to work please create a directory in tmpfs (/tmp/app here)  
        # for the fragments. The directory contents is served via HTTP (see  
        # http{} section in config)  
        #  
        # Incoming stream must be in H264/AAC. For iPhones use baseline H264  
        # profile (see ffmpeg example).  
        # This example creates RTMP stream from movie ready for HLS:  
        #  
        # ffmpeg -loglevel verbose -re -i movie.avi -vcodec libx264  
        #       -vprofile baseline -acodec libmp3lame -ar 44100 -ac 1  
        #       -f flv rtmp://localhost:1935/hls/movie  
        #  
        # If you need to transcode live stream use 'exec' feature.  
        #  
        application hls {  
            live on;  
            hls on;  
            hls_path /usr/local/nginx/html/hls;  
            hls_fragment 5s;  
        }  
    }  
}  
  
http {  
  
    server {
```

```
listen 8080;
location /hls {
    # Serve HLS fragments
    types {
        application/vnd.apple.mpegurl m3u8;
        video/mp2t ts;
    }
    root html;
    expires -1;
}
}
```

```
events {
    worker_connections 1024;
}
rtmp {
    server {
        listen 1935;
        chunk_size 4000;
        #HLS
        # For HLS to work please create a directory in tmpfs (/tmp/app here)
        # for the fragments. The directory contents is served via HTTP (see
        # http{} section in config)
        #
        # Incoming stream must be in H264/AAC. For iPhones use baseline H264
        # profile (see ffmpeg example).
        # This example creates RTMP stream from movie ready for HLS:
        #
        # ffmpeg -loglevel verbose -re -i movie.avi -vcodec libx264
        #         -vprofile baseline -acodec libmp3lame -ar 44100 -ac 1
        #         -f flv rtmp://localhost:1935/hls/movie
        #
        # If you need to transcode live stream use 'exec' feature.
        #
        application hls {
            live on;
            hls on;
            hls_path /usr/local/nginx/html/hls;
            hls_fragment 5s;
        }
    }
}
```

检查一下语法:

```
nginx -t
```

重启 Nginx

```
nginx -s reopen
```

查看 1935 端口是否开启

```
netstat -an |grep 1935
tcp      0      0 0.0.0.0:1935          0.0.0.0:*          LISTEN
```

```
netstat -an |grep 8080
cp      0      0 0.0.0.0:8080          0.0.0.0:*          LISTEN
```

这种状态就说明成功了。

在 FreeSWITCH 中配置 mod_av (编解码的模块)。下载 vc.tar.gz (注：FreeSWITCH 源代码里有个 html5/verto/verto_communicator 目录，安装 verto

```
cd html5/verto/verto_communicator
./debian8-install.sh
```

执行完将生成 dist 目录。)

```
cd /usr/local/freeswitch/conf/autoload_configs/
vim verto.conf.xml
```

打开后添加

```
<vhosts>
<vhost domain="localhost">
    <param name="alias" value="seven.local.freeswitch.org"/>
    <param name="root" value="/date/tools/vc"/>
    <param name="index" value="index.html"/>
<!--
    <param name="auth-realm" value="FreeSWITCH"/>
```

```

<param name="auth-user" value="freeswitch"/>
<param name="auth-pass" value="rocks"/>
-->
</vhost>

</vhosts>

```

在</profile>前，如图：

```

<param name="bind-local" value="$${local_ip_v4}:8081"/>
<param name="bind-local" value="$${local_ip_v4}:8082" secure="true"/>
<param name="force-register-domain" value="$${domain}"/>
<param name="secure-combined" value="$${certs_dir}/wss.pem"/>
<param name="secure-chain" value="$${certs_dir}/wss.pem"/>
<param name="userauth" value="true"/>
<!-- setting this to true will allow anyone to register even with no account so use with care -->
<param name="blind-reg" value="false"/>
<param name="mcast-ip" value="224.1.1.1"/>
<param name="mcast-port" value="1337"/>
<param name="rtp-ip" value="$${local_ip_v4}"/>
<!-- <param name="ext-rtp-ip" value="" /> -->
<param name="local-network" value="localnet.auto"/>
<param name="outbound-codec-string" value="opus,vp8"/>
<param name="inbound-codec-string" value="opus,vp8"/>

<param name="apply-candidate-acl" value="localnet.auto"/>
<param name="apply-candidate-acl" value="wan_v4.auto"/>
<param name="apply-candidate-acl" value="rfc1918.auto"/>
<param name="apply-candidate-acl" value="any_v4.auto"/>
<param name="timer-name" value="soft"/>

<vhosts>
<vhost domain="localhost">
<param name="alias" value="seven.local freeswitch.org"/>
<param name="root" value="/date/tools/vc"/>
<param name="index" value="index.html"/>
<!--
<param name="auth-realm" value="FreeSWITCH"/>
<param name="auth-user" value="freeswitch"/>
<param name="auth-pass" value="rocks"/>
-->
</vhost>
</vhosts>

</profile>
</profiles>
</configuration>

```

47,1

其中里面的`param name = "root" value="date/tools/vc"`是 vc 解压后的地址。这样 mod_verto 就变成了一个服务器了。

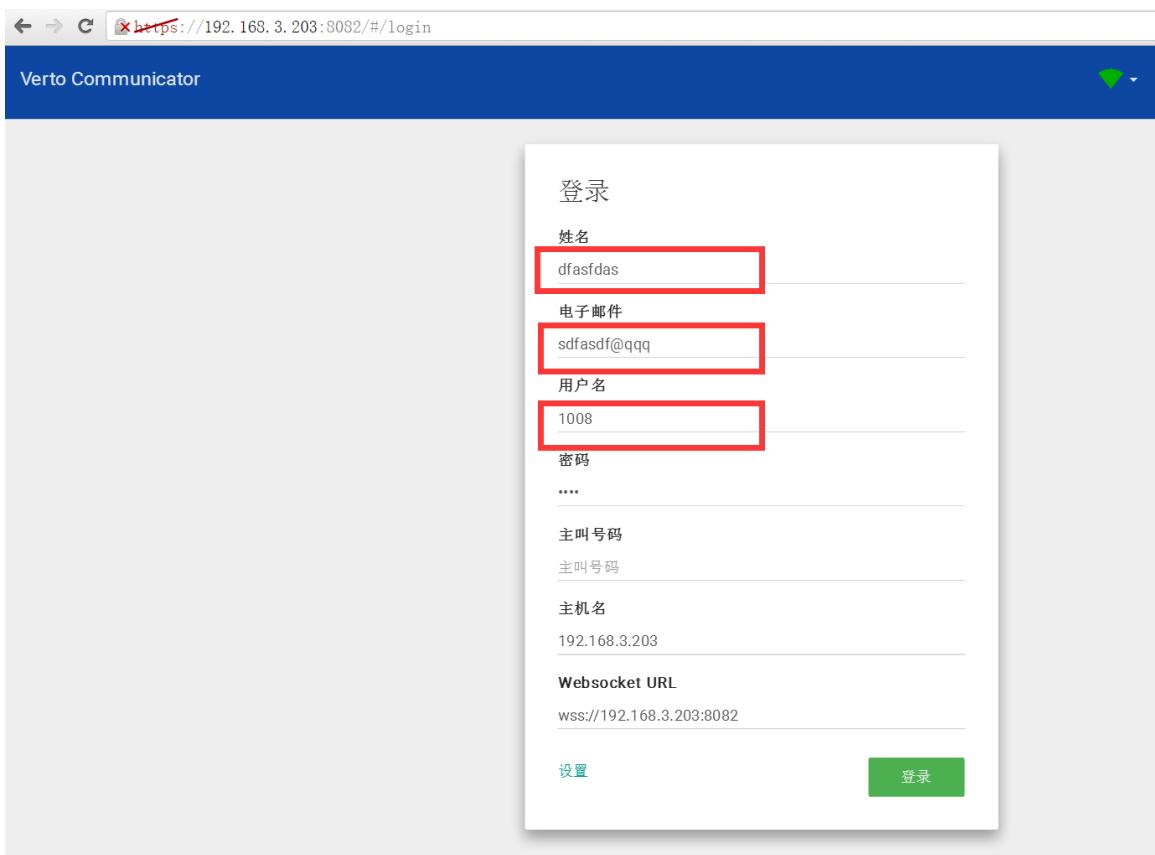
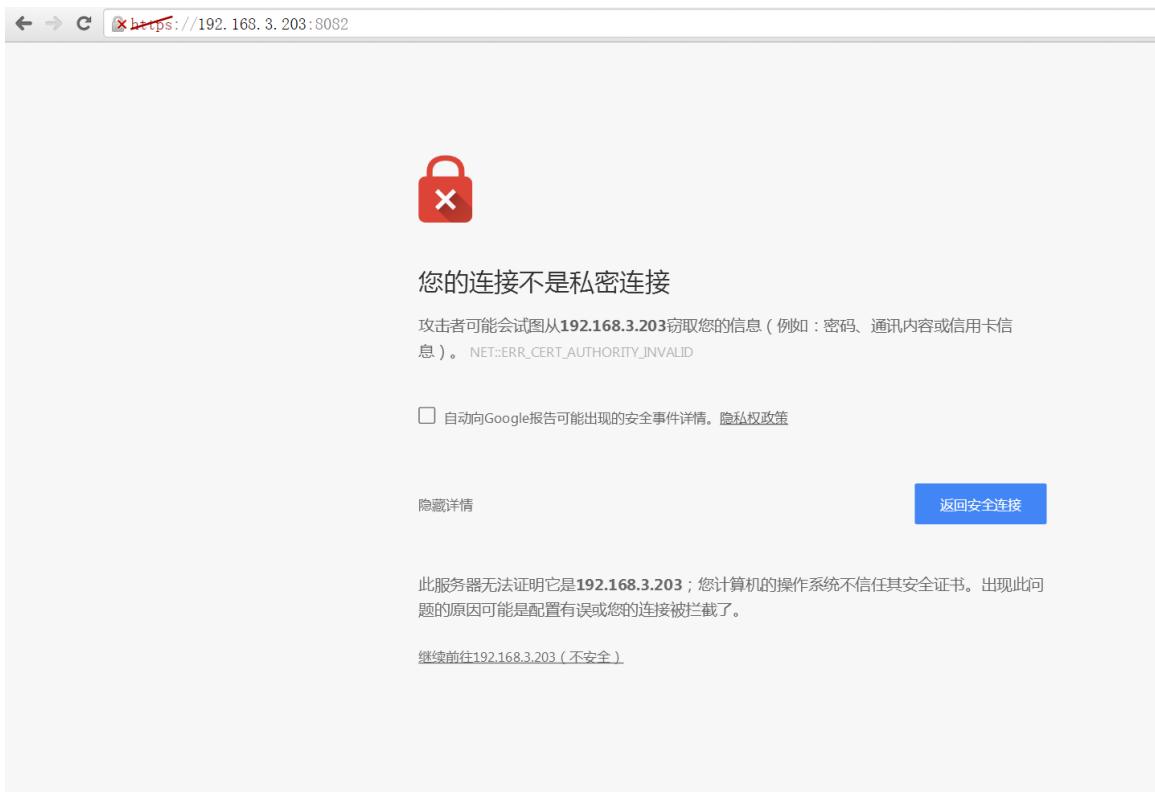
接着在 FreeSWITCH 中

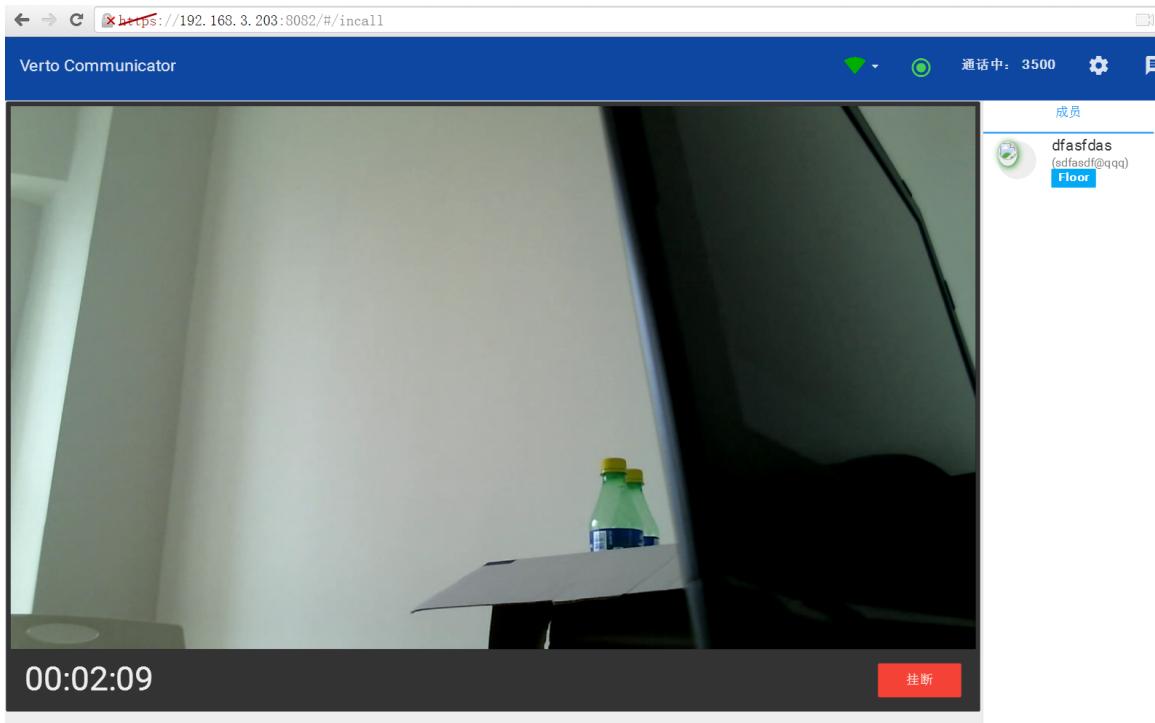
```
load mod_av
```

用 Chrome 浏览器访问：`https://你的 ip:8082/`

第一次访问，需要输入你的姓名和电子邮件。

呼叫 3500，就可以进入视频会议了。





这时候在 FreeSWITCH 中执行

```
conference 3500-121.41.105.36 record rtmp://192.168.3.203/hls
```

如果你觉得很麻烦

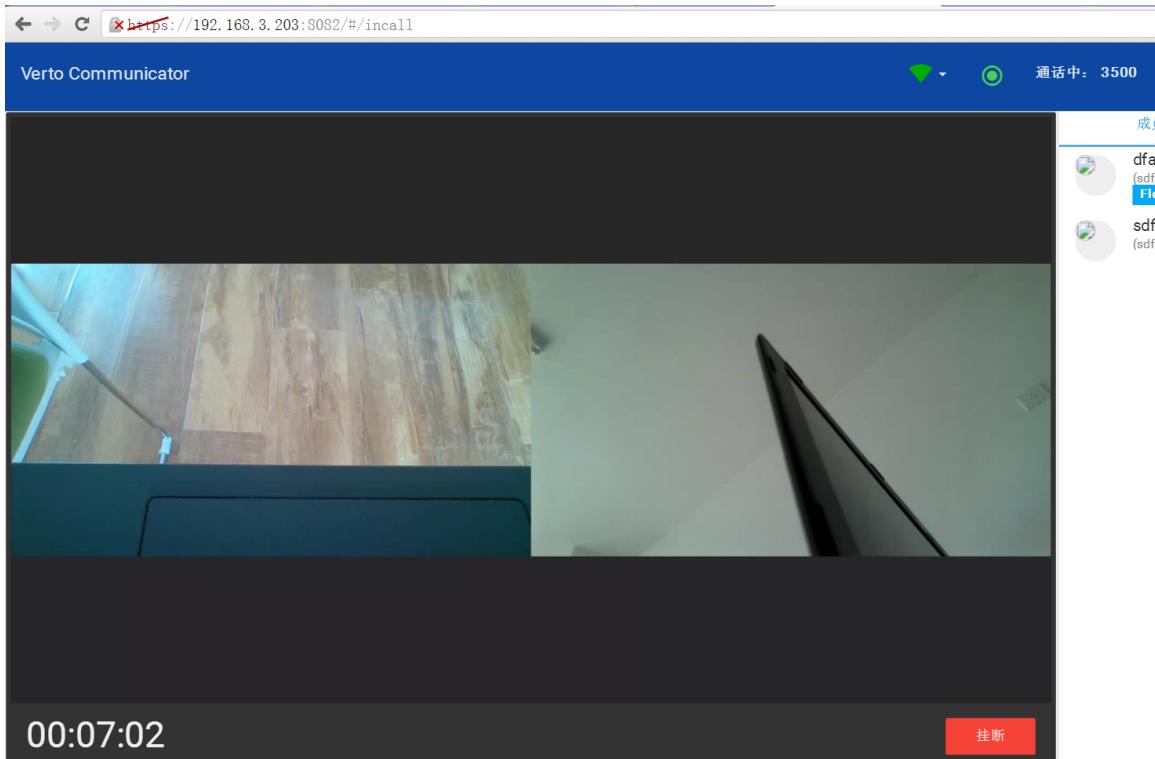
```
apt-get install crtmpserver -y  
/etc/init.d/crtmpserver start
```

在 FreeSWITCH 中

```
conference 3500-121.41.105.36 record rtmp://192.168.3.203/live/1
```

然后就可以在<https://www.hlsplayer.net/rtmp-player>中输入`rtmp://192.168.3.203/live/1`进行观看了。

因为之前虚拟机用了 1 个 CPU/1G 的内存，在转码的时候 CPU 使用率达到 96%。需要用多个 CPU 更大的内存才能流畅的使用。



9.4 FreeSWITCH 问题解决的几个例子

大家经常在 QQ 群里问到关于 FreeSWITCH 的问题，却总是不得要领。我花了很多时间写《FreeSWITCH 新手指南》，可是大家都不怎么爱看。当然，可能《指南》比较泛泛，没有针对性。今天，我们就拿几个实例来说吧。

0x01:

问题是这样的，在 QQ 群里，有个同学问：FreeSWITCH 启动一闪就退出了，怎么办？

我的判断：FreeSWITCH 一闪的情况，可能只会在 Windows 上出现。但是，这位同学确实应该这么问：

我使用 Win7，安装了 FreeSWITCH（或从源代码安装了 FreeSWITCH），版本是 XXXX。我尝试启动时双击开始菜单里的 FreeSWITCH 启动项，有个窗口一闪就退了，看不清，请问这是什么问题呢？怎么解决？

这样提问就比较明确了。而前面的提问其实有效信息真的很少。其实 FreeSWITCH 典型的应用是在 Linux 上。如果是 Linux，还要说明，是 CentOS，还是 Debian，还是 Ubuntu 等，还要说明版本，如 CentOS 5.5 或 Debian 8.2。但是遗憾的是，大多数同学的提问基本是就假设 QQ 群里的人都站在他身后，“你看，这个一闪就退了，怎么办”，他不知道其实别人不知道他用的是什么操作系统，用的什么 FreeSWITCH 版本。

好了，再回到我们的问题。根据经验判断，可能是因为 FreeSWITCH 在启动时没有权限无法锁

定 FreeSWITCH.pid 文件，致使进程退出。跟他说了用管理员身份运行一下（这个不用教了吧？）。

后来它以管理员身份运行了还不行。那就得找其它原因了。我让他进到命令行模式下，找到 FreeSWITCH 的可执行文件（如c:\Program Files\FreeSWITCH\freeswitchConsole.exe），手工执行一下，就应该能看到出错信息。

他照做了，出错信息是类似XML Parsing Error, Unclosed Tag <endpoint 之类的。

根据经验判断，他改过 XML 文件，改错了，所以才出现这样的错误。

他说他改回去了，还是这样的错误。我建议他全部重装。怎么解释呢？他说改回去了，但肯定没有完全恢复到原来默认的状态！

最后，其实这个问题应该这么问：

大家好，我使用 Win7，安装了 FreeSWITCH，版本是 XXXX。我尝试启动时双击开始菜单里的 FreeSWITCH 启动项，有个窗口一闪就退了，看不清。后来，我从命令行模式下输入 freeswitchconsole.exe，显示的出错信息如下：xxxxx，我记得改过 XML 的某某部分，现在又改回去了，还是出现这个错误，请问我该怎么办？

这样就是一个比较好的问题了。

我们没收到这个同学的反馈。这也是在 QQ 群里提问的问题。因为我回答完就睡觉去了，后来也没看到他回复。不管问题解决没解决，最好都有个结果反馈。当然，更好的方式是把问题发到 BBS 上，这样便会有完整的记录。

0x02:

A: 为什么按照官网上的语法api.executeString("version"); 会报错，Error in API::executeString expected 2..2 args, 通过api = freeswitch.API();获取的 api

B: 这样

```
api = freeswitch.API(); api:executeString( "version"):
```

A: 我就是这么写啊。

点评：A 和 B 写得其实不一样。找不同，你能找到几处不同？

0x03:

A: 我按照书的方式配置了 Dialplan，可以无论打什么号码都会进入这一个，进不了其它的，请问是什么原因？

B: 把日志贴出来

A: [日志]

B: 把 Dialplan 贴出来

A: [Dialplan]

B: 你是照着书上写的，但是把字母抄错了。

点评：为什么我们要求提供各种背景信息和日志？因为好多人非常坚定地声称他是怎么怎么做的，但是，你还是不能相信它。一切，以日志说话。

说到这里，其实我也犯过错误。我去给人家修网络，最后定位在线路的问题，打电话到联通客服，经过一番排查，最后发现是 ADSL 猫没加点，囧。

9.5 FreeSWITCH 代码最新变化

是的，FreeSWITCH 快要支持 VP9 了（实际上以前支持过，但跟不上 Chrome 更新的速度）…

记得以前说过关于 FS-353 的问题。FreeSWITCH 依赖很多现成的第三方库，最初，FreeSWITCH 把大部分第三方库直接放到 FreeSWITCH 的代码树里，编译安装比较方便，但是，编译起来也慢一点。

后来，FreeSWITCH 为了做安装包，就把大部分第三方依赖库从代码树里删除了，而是转而依赖系统提供的库。编译起来快多了，尤其是对我们这些一天编译好多遍的开发人员来说。但是，各种操作系统提供的库都有很多差异，即使同是 Linux，也差别很大。因此，FreeSWITCH 官方仅支持 Debian 8，对于其它的系统，都由社区的其它参与者支持。

一个不可忽略的事实是，很多人在用 CentOS、Ubuntu 等。编译起来不顺畅，这是大家比较抱怨的地方。

最近，也有人反映，即使在 Debian 8 编译也出错。为什么呢？命不好，正好遇到 FreeSWITCH 代码调整。

FreeSWITCH 内部使用了 `libvpx` 库，直接挪用了 `vpx` 提供的 I420 图片格式支持，以及 VP8/VP9 的支持。另外，还使用 `libyuv` 做图片的转换和缩放等。

这两个库是随着 Chrome 一起升级的。Chrome 更新了，库更新了，所以，我们也需要更新 FreeSWITCH，这就是现实。

本来，一切都做好了，更新也就是分分钟的事，但是，总会有不顺利的事情。

为什么这么麻烦呢？还得从以前说起。

Anthony 发现 `libvpx` 有 Bug，无法支持高并发（也就是说视频会议开到几路就不行了），但人家 Chrome 是个客户端软件，开发人员也不怎么积极去修，所以，FreeSWITCH 官方不得不维护了自己的版本。这就是为什么说如果编译 FreeSWITCH，要从 FreeSWITCH 官方下载这些第三方库的原因。

为了防止与实际的 `libvpx` 冲突，FreeSWITCH 官方将这个版本命名为 `libvpx2`，对应原 `libvpx1.4` 的版本。虽然经过大量测试，但难免挂一漏万，所以，编译起来有时多多少少总会出点小问题。不过，最后的安装包应该都是比较稳定的，也就是说，按官方的指南，直接 `apt-get` 安装问题应该不大。

问题来了。`libvpx` 更新到了 1.5.0，这个版本改变了内部的 ABI（没写错就是 ABI），但是，没有按规矩更新 ABI 版本号。这，是一个纯粹的 Bug，让我们无法适从。但问题是，FreeSWITCH 维

护的 vpx 代码 master 版本已经更新到这个版本了。所以，有些同学直接 clone/pull 代码，一编译就出了问题。因为 FreeSWITCH 还没有同步更新。

FreeSWITCH 团队其实也没闲着，早就给 Google 报了 Bug，但是，等了一周基本没什么进展，我们才又抓紧把代码适配到 1.5.0。说起来简单做起来难，且不说 vpx 有 Bug，就是没有 Bug，也得适配其 API/ABI 的变化，我们又看代码又查历史记录，真花了不少时间。

好吧，最后问题基本解决了。又发现，libvpx 又更新了，而且，这次，ABI 也更新了，而且，Chrome 里用了更新的 libvpx …

最终，FreeSWITCH 团队不得不痛下决定，算了，我们再也不能依赖操作系统提供的库了，因为操作系统提供的库总会滞后 N 个版本，而 Chrome 之流的软件更新又很激进。所以，就又回到了 FS-353 所描述的争论起点。把 libvpx 和 libyuv 的代码直接放到了 FreeSWITCH 源代码树里，跟 FreeSWITCH 一起编译。

嗯，放进去归放进去，还是有不少曲折的。因为，原来的 libvpx 是编译成动态库的，内置到 FreeSWITCH 里，标准的方法是编译成静态库，而在编译成静态库这一步，就偏偏出了问题。Mike 就是专门管这一块的，他问我对汇编熟悉不，问题出在一块汇编的代码文件里。我汇编也就会那么一丁点了，最后还是 Mike 找出了问题，编译通过了。

所以，如果你是使用最新的 master 代码，libvpx 和 libyuv 已经内置到 FreeSWITCH 里了，再自己编译的话，就省了不少麻烦。

另外，mod_vpx 已经不存在了，因为它已被移动到核心里去了。

有 Redhat 用户说编译可能还有问题，如果你使用 CentOS 或 Redhat，不妨测试一下，如果有问题向 FreeSWITCH 官方报 Jira，这也是为 FreeSWITCH 做贡献的好机会啊。

基本就是这样，FreeSWITCH 肯定会越变越好。当然，也离不开大家的支持。

9.6 FreeSWITCH 与 FFmpeg

杜金房 / 2016.03.11

关于 FreeSWITCH 与 FFmpeg 的恩怨可以讲很多，不过，让我们长话短说。

FFmpeg 是比较流行的多媒体库，可以处理语音视频之类的，在开源领域内得到了大量应用，包括 Android 和 Chrome。如果我没记错的话当年 QQ 也用过它，但因没有遵循开源协议开放源码而被钉在了耻辱柱上。

几年前（具体时间懒得查了）。由于开发团队的分歧，FFmpeg 分裂了。部分开发者另起一摊，Fork 了一下，起名叫 Libav。但问题是，虽然项目名称改了，但为了跟大多数现在应用兼容，库名称依然叫 libavcodec、libavformat 之类的。这是所有噩梦的开始。

其实我在更早的时间就开始在 FreeSWITCH 里基于 FFmpeg 写一个模块，最初叫 mod_ffmpeg。第一个可以运行的版本是在从 Cluecon 回来的飞机上调试成功的。

后来，由于 CentOS 的诡异问题，FreeSWITCH 开发团队将开发平台迁移到了 Debian，而 Debian 使用 Libav，所以，我们趁机将mod_ffmpeg改为两个模块，叫mod_avcodec和mod_avformat。

libavcodec是做编解码处理的，libavformat是处理多媒体文件的。相对的，mod_avcodec就实现了 H264、H263 之类的编码，而libavformat就支持mp4文件等。

后来，这两个模块合并成了一个模块，叫mod_av。就是大家在 FreeSWITCH 1.6 里看到的。不过，这个模块默认是不编译的，所以，如果需要的话要手工编译。原因很简单，libav/ffmpeg里有一些依赖库使用的是 GPL 的（如libx264）。

在 Debian 上编译很简单，要知道，为了能在 Debian 上顺利编译，开发团队也是费了很大劲的。大家也都希望能支持 CentOS，但 CentOS 上是 FFmpeg，需要做兼容，FreeSWITCH 团队的开发力量不够，因此这个依赖于社区支持。但遗憾的是，没有任何一个人贡献一行代码。我们中文社区的群里也有很多人遇到这个问题。不厌其烦地问，也没有任何一个人说出点钱资助一下解决这个问题。

所有人都告诉我 CentOS 是刚需，但对于我来讲，没有人愿意贡献代码或出资来做这件事就不是刚需。关于刚需这个话题，我改天再专门写文章来讲。今天就不多说了。我们小樱桃团队刚刚成立，作为一个有情怀的团队，我们投入了一些力量，把这个任务完成了。虽然过程很曲折，但其实最后改的代码也没有几行。不敢独享，跟大家分享一下相关的技术要点。

首先，FFmpeg 本身就有很多版本，分裂后版本就更多了。我最初开发是基于 0.8.x 的，后来就直接基于了 FFmpeg 的master版，后来，就试了libav 11.3、11.4、11.6上个月刚刚发布。最近试了FFmpeg 2.6.x、2.8.x、3.0，基本都能顺利编译过。

为了同时测多个版本，我们需要一些技巧。

首先，卸载所有随系统安装的版本。重新执行 FreeSWITCH 的configure，让 FreeSWITCH 找不到libav和 FFmpeg。

然后，编译安装各个版本的 Libav 和 FFmpeg。

编译步骤满大街都是，主要的几个参数是：

```
--prefix=/opt/av
```

我安装到了/opt/av，当然你也可以装到/opt/av-11.3、/opt/av-11.6之类的，FFmpeg 也是一样。

在开发过程中我们还遇到libx264新版本导致的问题，所以还测试了很多版本的libx264：

```
/configure --prefix=/opt/x264
```

为了让 Libav 能找到我们自己编译的x264，指定PKG_CONFIG_PATH：

```
PKG_CONFIG_PATH=/opt/x264/lib/pkgconfig ./configure --prefix=/opt/av
```

其它的参数就看你的需求选了，下面的例子是我们 Mac 上编译 Libav 的例子

```
/configure --prefix=/opt/av --enable-shared --enable-pthreads --enable-gpl --enable-version3 --enable-hardcoded-tables
```

FFmpeg 的命令行也类似。实际上，最重要的是--enable-libx264。

当然，`configure`完了后需要`make && make install`，这是 UNIX 在软件编译安装的标准步骤，不多说。

好了，有了多个 Libav 和 FFmpeg，怎么让 FreeSWITCH 找到它呢？

到 FreeSWITCH 源代码目录下

```
cd src/mod/applications/mod_av
```

创建如下 Makefile（如果已经有了就替换掉）

```
#AV=/opt/av/  
AV=/usr/local/Cellar/ffmpeg/2.8.6  
LOCAL_CFLAGS=-I$(AV)/include  
LOCAL_LDFLAGS=-L$(AV)/lib -lavcodec -lavformat -lavutil -lavresample -lswscale  
LOCAL_OBJS=avcodec.o avformat.o  
  
include ../../../../build/modmake.rules
```

然后在`mod_av`下面执行`make install`（你的 FreeSWITCH 必须正常编译过一遍啊，别说我没告诉你）

一切顺利的话，你就可以在 FreeSWITCH 里面`load mod_av`了。

什么？还是不行？

在 Linux 上，还需要设置动态库的加载路径。最简单的办法是启动 FreeSWITCH 的时候加到环境变量里，如，可以用以下命令启动 FreeSWITCH：

```
LD_LIBRARY_PATH=/opt/av/lib /usr/local/freeswitch
```

另一种就是放到`/etc/ld.so.conf`或`/etc/ld.so.conf.d`里，并执行`ldconfig`。如果你需要经常切换多个版本，还是用环境变量来得快些。关于`ldconfig`，也是 UNIX 开发者的必备修养，不知道的自己 Google 吧。

好了，正常`load mod_av`后，你就可以尝试使用它提供的 H264 编码，录音、录像、播放视频等功能了。

当然，如果你使用 Git master 代码，可能直接就什么也不用动，`yum install ffmpeg-devel`，然后`make mod_v-install`就行了，我没有试过。大家可以测一下如果使用 CentOS 自己带的库有没有问题，也可以告诉我各种版本的 CentOS 都带了 FFmpeg 的哪个版本。

9.7 如何录制喜欢的电视节目

杜金房/2015.12.15

今天，说个技术话题。

我上电视了。但是，我不能看。虽然家里有电视机，但房子住了好几年了，从来没有开通有线。前一阵，联通搞活动，装了个联通牌的机顶盒，也算是过上了小康生活。

小康归小康，其实我们家也不大看电视。

日子一天天过去，忙碌而又充实。

偶有小波澜——我上电视了。将在烟台台黄金时段播出。

拿起遥控器找台，几百个频道的样子，找了八遍，没找到烟台台。

给同学打电话，我上电视了，快快看吧。同学说，不看，还在加班呢。再说了，上电视有什么大惊小怪的，俺老婆经常上电视……

但俺是第一次啊！

上网看。别说，在胶东在线上还真搜到一个页面，烟台 1、2、3、4 套节目全有。

页面用 Flash 做的，有直播，有点播。很卡，但是，先看再说，至少截图发朋友圈是足够了。

终于在漫长的卡顿中看完了。

怎么存下来呢？做技术的，总喜欢钻研一下。网站上有三天内的视频回放，因此，先到视频回放页面找到相应的视频，确定可以播放。然后打开 Wireshark，启动抓包，刷新页面。看到好多 119.180.20.x 的包，可能就是它了。用该 IP 地址作为过滤条件 (`ip.addr == 119.180.20.x`)，包太多。一般 Flash 的直播都是用 rtmp 协议传送的，用 rtmp 过滤，没看到包，改用 http 过滤，这次看到包了，如下图（注意，真实网址用 example.com 代替了）

到这儿，似乎一切看起来都挺顺利。后面的『1450094400000,1450098000000』部分看起来就像一对时间戳，像起止时间之类的。

打开终端，输入以下命令（Mac 和 Linux 上）

Filter: http						Expression...	Clear	Apply	Save
No.	Time	Source	Destination	Protocol	Length	Info			
522	8.087538000	192.168.7.6	119.180.20.1	HTTP	508	GET /channels/yttv/xnpd_yt2/flv:500k/1450094400000,1450098000000 HTTP/1.1			
703	10.0528870	(119.180.20.192.168.7.6	HTTP		1494	Continuation			
HyperText Transfer Protocol									
GET /channels/yttv/xnpd_yt2/flv:500k/1450094400000,1450098000000 HTTP/1.1\r\n									
Host: live1.av.example.com\r\n									

```
$ date +%s
1450090255
```

可以看到时间戳是 10 位，单位是秒。那么，上述网址中的时间戳应该就是毫秒了。试一把，打开 Ruby 交互终端，使用 Time.at 函数可以看到如下输出：

```
$ irb
> Time.at(1450094400)
=> 2015-12-14 20:00:00 +0800

> Time.at(1450098000)
=> 2015-12-14 21:00:00 +0800
```

Yeah，确实是我们的黄金时间。

先用 VLC 试一下能不能播放。把图中的地址拼成：

```
http://live1.av.example.com/channels/yttv/xnpd_yt2/flv:500k/1450094400000,1450098000000
```

在 VLC 中打开以上网址，发现可以正常播放的。

成功一半了。

VLC 本自有录像功能，但我从来没用过。尝试一下 rtmpdump，由于直播地址不是 rtmp 的，rtmpdump 不好用。换 1avconv1：

```
/opt/av/bin/avconv -i "http://live1.av.example.com/channels/yttv/xnpd_yt2/flv:500k/1450094400000,1450098000000"
-acodec copy -vcodec copy x.flv
```

开始下载了，几秒钟后在 flv 中打开 x.flv，发现确实是我们要的时间段的视频。

有我的那块当然没有一小时那么长，因此，缩小了一下范围：

```
/opt/av/bin/avconv -i "http://live1.av.example.com/channels/yttv/xnpd_yt2/flv:500k/1450095270000,1450095540000"
-acodec copy -vcodec copy x.flv
```

当然，`x.flv`也可以换成`x.mp4`，区别只是`flv`的文件信息是写在头部的，可以直接在下载的过程中播放，`mp4`是写在文件尾部的，只能等下载完毕后再播放。

注意，我用了`avconv`，你应该也可以用`ffmpeg`。`avconv`本来也是`ffmpeg`，后来，Fork 了。技术圈那些事，不多说了。感兴趣的可以自己搜一搜`ffmpeg vs avconv`。

好玩不？

9.8 FreeSWITCH 精英群第 N 次活动小记

杜金房 / 2016.05.25

不记得是第几次了。今天晚上，我跟大家在 QQ 群里聊了聊。

主要聊了 `mod_verto` 这个模块，以及 `verto communicator`，后来又有人来，没赶上，我就简单的写一写吧。

`mod_verto` 这个模块以前讲过，是 FreeSWITCH 实现了一个私有的信令（其实也不算私有，人家都开源了啊），使用 `websocket` 传输一些 `json` 消息建立通话。

`verto communicator` 是一个 `verto` 的客户端，运行在浏览器里面。

安装 `verto communicator` 很简单，进入源代码的 `verto communicator` 目录

```
cd html5/verto/verto_communicator/
```

如果你是用 Debian 8 的话，直接执行下面的脚本就好了：

```
./debian8-install.sh
```

比较悲催的是，我运行了，但是没安装成功。

`verto communicator` 是用 `Angular` 框架写的，因此需要 `nodejs`, `npm`, `grunt` 等一大堆东西，在这个神奇的国度里，很多依赖都下载失败了。

我的 Debian 在阿里云上，没架过 VPN，后来还是把我本地打包好的文件上传到服务器上去了。话说阿里云本来不提供 Debian 8，我是从 Debian 7 上手工升级的。最新的 FreeSWITCH 需要用 Debian 8 才好，题外话。

我们近距离看一看脚本都干了些什么。

```
cat ./debian8-install.sh

#!/bin/bash
apt-get update
apt-get install npm nodejs-legacy
npm install -g grunt grunt-cli bower
npm install
bower --allow-root install
grunt build
```

很简单一个脚本，做前端的应该都很熟悉这些命令，这里就不多说了。总之，如果顺利安装并且 grunt build 的话，会在当前目录下产生一个 dist 目录，里面就是 html 的根目录。

话说，放 html 要有个 Web 服务器。典型的有 Apache 和 Nginx。官方的 Nginx 有 Apache 的配置例子，我则比较喜欢 Nginx，都可以配好。不过，由于最新版的 Chrome 要求使用 WebRTC 必须使用 https 和 was，所以，还需要在 Web 服务器里配置证书。这一步有些麻烦，先不讲了。

下面说一个简单方法：

将下列内容放到 FreeSWITCH 的 `verto.conf` 里，放在 `</profile>` 之前：

```
<vhosts>
<vhost domain="localhost">
<param name="alias" value="freeswitch.org"/>
<param name="root" value= "/var/www/html"/>
<param name="index" value="index.html"/>
</vhost>
</vhosts>
```

这样，FreeSWITCH 就成了一个简单的 Web 服务器了，支持 http 和 https，这样，测试起来不用配 Nginx 或 Apache，方便多了。

用 Chrome 浏览器访问 `https://你的 IP:8082/`（注意 s）就可以看到 Web 页面了。注意，要设置正确的 root 目录。我一般把 `verto communictor` 放在 `/var/www/html/vc` 目录中。

第一次访问时 Chrome 说证书不可信，继续就可以了。因为 FreeSWITCH 默认生成了一个自己签名的测试证书。注意，有些浏览器如 FireFox 可能会挑证书。如果实在不行那你得买个真证书了。

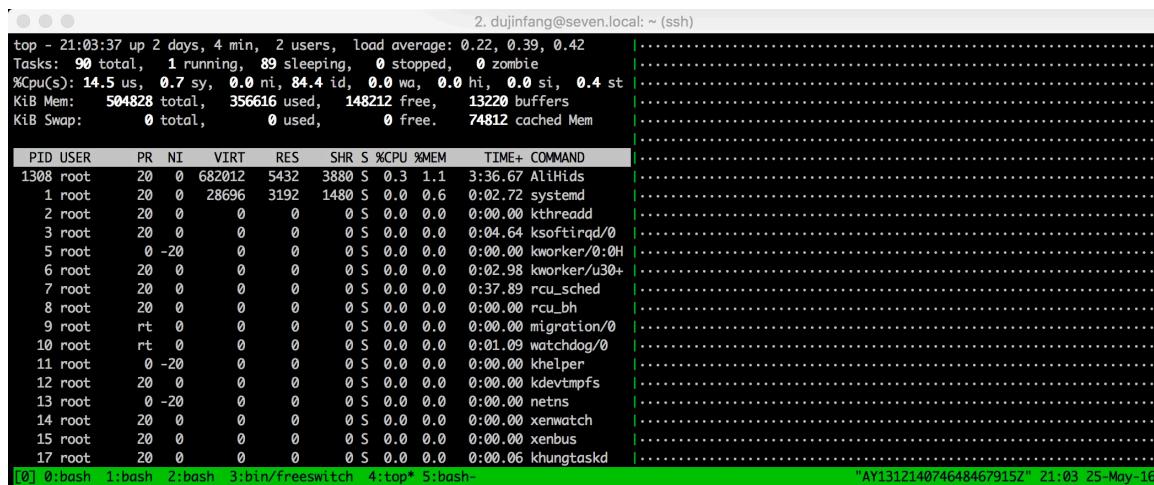
打开上述网址就可以登录了，保持 FreeSWITCH 运行，输入你的姓名和电子邮件，登录后呼叫 3500 就可以进入一个测试的视频会议。如果有多几台电脑的话将看到更好的效果。

好玩吧。

更高兴的是今天晚上找到了直播写代码的方法，以后直播写代码有望了。

由于我一直用 Mac，Windows 好多年不用都不熟了，而且，Windws 上各种弹出广告总会让我找不到北。但没办法，QQ 群视频只能在 Windows 上用。所以，我就用它直播。群视频中可以共享桌面，我共享了一个 Putty 客户端（SSH 客户端）。ssh 到我的服务器上，然后，我在 Mac 上也 ssh 到服务器上，执行 tmux（终端共享软件，类似于 screen）以及 tmux attach，这样，就可以在 Windows 和 Mac 上看到同样的服务器界面了。我在 Mac 上敲代码，在 Linux 上运行，在 Windows 上直播出去，Yeah!

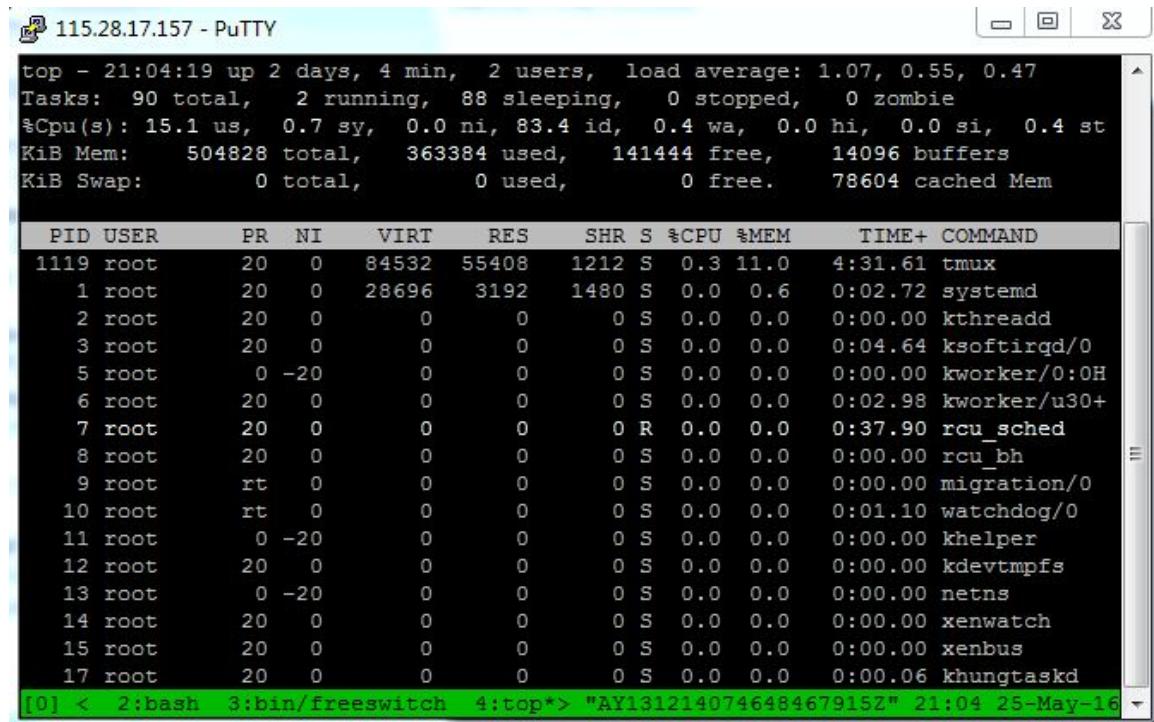
上图：



```
2. dujinfang@seven.local: ~ (ssh)
top - 21:03:37 up 2 days, 4 min, 2 users, load average: 0.22, 0.39, 0.42
Tasks: 90 total, 1 running, 89 sleeping, 0 stopped, 0 zombie
%Cpu(s): 14.5 us, 0.7 sy, 0.0 ni, 84.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.4 st
KiB Mem: 504828 total, 356616 used, 148212 free, 13220 buffers
KiB Swap: 0 total, 0 used, 0 free. 74812 cached Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1308 root 20 0 682012 5432 3880 S 0.3 1.1 3:36.67 AltHids
1 root 20 0 28696 3192 1480 S 0.0 0.6 0:02.72 systemd
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:04.64 ksoftirqd/0
5 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
6 root 20 0 0 0 0 S 0.0 0.0 0:02.98 kworker/u30+
7 root 20 0 0 0 0 S 0.0 0.0 0:37.89 rcu_sched
8 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_bh
9 root rt 0 0 0 0 S 0.0 0.0 0:00.00 migration/0
10 root rt 0 0 0 0 S 0.0 0.0 0:01.09 watchdog/0
11 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 khelper
12 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kdevtmpfs
13 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 netns
14 root 20 0 0 0 0 S 0.0 0.0 0:00.00 xenwatch
15 root 20 0 0 0 0 S 0.0 0.0 0:00.00 xenbus
17 root 20 0 0 0 0 S 0.0 0.0 0:00.06 khungtaskd

[0] 0:bash 1:bash 2:bash 3:bin/freeswitch 4:top* 5:bash- "AY131214074648467915Z" 21:03 25-May-16
```



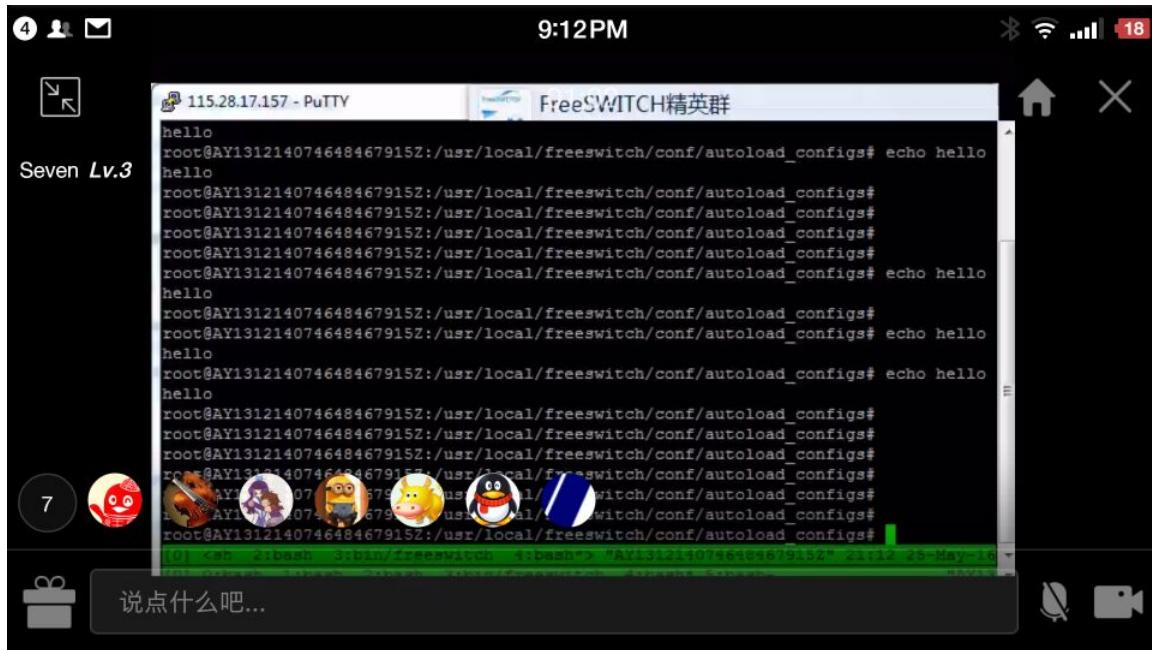
```
115.28.17.157 - PuTTY
top - 21:04:19 up 2 days, 4 min, 2 users, load average: 1.07, 0.55, 0.47
Tasks: 90 total, 2 running, 88 sleeping, 0 stopped, 0 zombie
%Cpu(s): 15.1 us, 0.7 sy, 0.0 ni, 83.4 id, 0.4 wa, 0.0 hi, 0.0 si, 0.4 st
KiB Mem: 504828 total, 363384 used, 141444 free, 14096 buffers
KiB Swap: 0 total, 0 used, 0 free. 78604 cached Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1119 root 20 0 84532 55408 1212 S 0.3 11.0 4:31.61 tmux
1 root 20 0 28696 3192 1480 S 0.0 0.6 0:02.72 systemd
2 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:04.64 ksoftirqd/0
5 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
6 root 20 0 0 0 0 S 0.0 0.0 0:02.98 kworker/u30+
7 root 20 0 0 0 0 R 0.0 0.0 0:37.90 rcu_sched
8 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_bh
9 root rt 0 0 0 0 S 0.0 0.0 0:00.00 migration/0
10 root rt 0 0 0 0 S 0.0 0.0 0:01.10 watchdog/0
11 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 khelper
12 root 20 0 0 0 0 S 0.0 0.0 0:00.00 kdevtmpfs
13 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 netns
14 root 20 0 0 0 0 S 0.0 0.0 0:00.00 xenwatch
15 root 20 0 0 0 0 S 0.0 0.0 0:00.00 xenbus
17 root 20 0 0 0 0 S 0.0 0.0 0:00.06 khungtaskd

[0] < 2:bash 3:bin/freeswitch 4:top*> "AY131214074648467915Z" 21:04 25-May-16
```

三人行，必有我师。今天还有人告诉我 QQ 手机版也可以看群视频，我还是刚刚知道呢，效果还

不错呢:)。



9.9 FreeSWITCH 第 N+1 次活动小记

这次，到场的人数较少，但杜老师兴致丝毫不减。直接就讲起了代码。

跟上次一样，在 QQ 群里直接一个视频窗口，窗口用 Putty 连接到一个 Linux 服务器上，杜老师在 Mac 上用 SSH 连接服务器，通过 tmux 共享屏幕。

杜老师先分享了一个 Bug

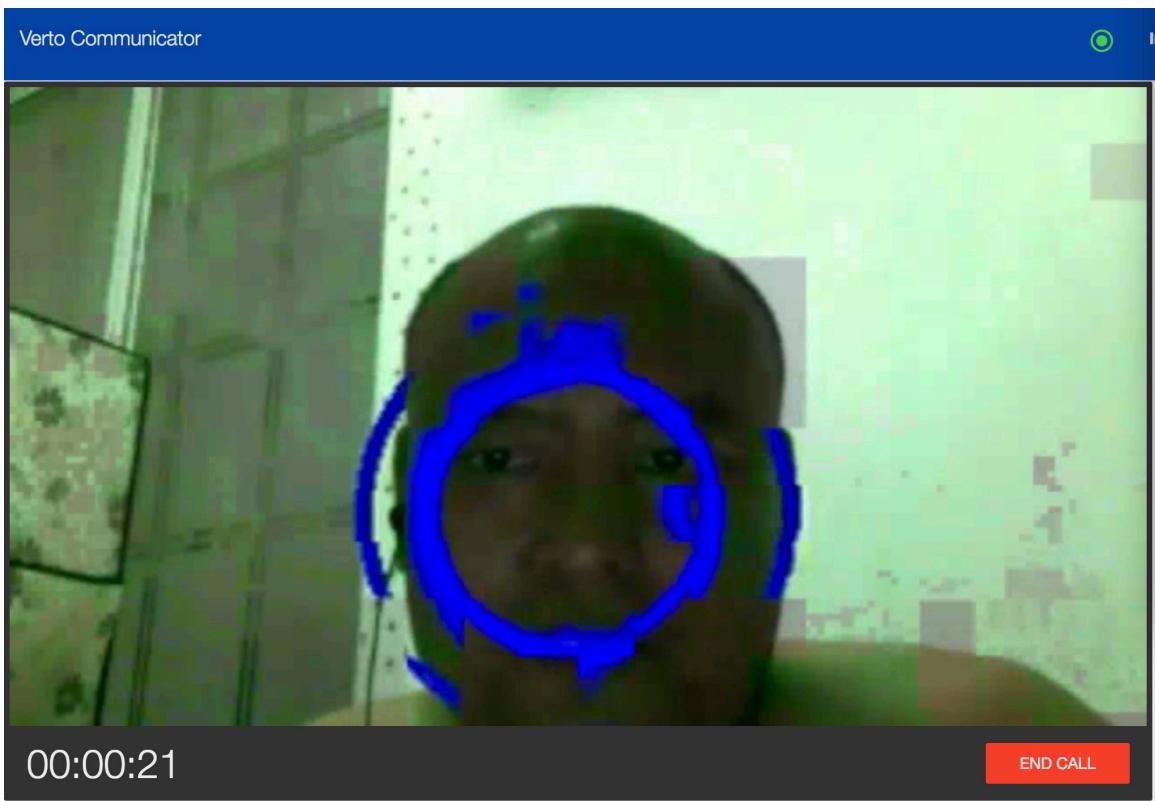
<https://freeswitch.org/jira/browse/FS-9266>

这个问题是早上解决的

<https://freeswitch.org/fisheye/changelog/freeswitch?cs=c0499fbb22b42839b7df8c7bcc8a15dfcb6c1c4e>

在某一次重构代码的时候，引入了这个 Bug，只是，由于 `mod_cv` 这个模块用的人少，竟然一直没人发现。

解决完这个 Bug，又发现了另外一个问题：在用 `mod_cv` 做人脸识别的时候，返回的画质会越来越差，直到遇到下一个关键帧，至少看起来是这样。



杜老师跟大家把代码过了一遍，加入了一些日志，不过，发现可能是画圈的时候引起的问题。是的，FreeSWITCH 支持人脸识别。甚至能通过检测指定的动作控制 FreeSWITCH 的行为。时间所限，并没有在这次代码分享活动中解决该问题。不过，后来该问题还是解决了。确切的说是 Anthony 解决的。

<https://freeswitch.org/jira/browse/FS-9267>

修复在这里

<https://freeswitch.org/fisheye/changelog/freeswitch?cs=b0be5d67375fb8b23bccf2e4e544bc8a2afeb4e1>

后来查明问题的原因并不在于`mod_cv`，而是…

好吧，解决问题的过程是这样的。该问题以前是不存在的。但现在出现了。经过一翻调试，怀疑这个跟 VPX 有关，而不是 FreeSWITCH 本身。后来用 H264 编码试了一下，果然，没有此问题。

所以，怀疑可能是 libvpx 库升级后进行了一些优化。由于`mod_cv`会改变图片数据的内容，但这个改变`libvpx`可能不知道，因此，`libvpx`就傻了…

最后，为了解决这个问题。只好不让`mod_cv`改变`libvpx`原始的图片。每个图片都复制一份。

嗯，是的，`libvpx`所做的的优化起了副作用。当然，不能说是`libvpx`的错，但是，确定在使用`mod_cv`的时候，就不能享受这些优化的好处了。

好在现在还没有人用`mod_cv`识别很多 Channel …

9.10 解决一个诡异的呼叫问题

杜金房 / 2016.07.30

今天，处理了一个比较特殊的 FreeSWITCH 对接问题。

事情的经过是这样的。

有人联系我们想帮忙解决个问题，并给出了问题的简单描述：

8 根线不稳定半小时就得重启 FreeSWITCH。

问题描述太过简略，因此我们建议用户交纳一定的费用，我们进一步沟通后再决定具体价格，为了便于理解，我们管这个费用叫挂号费。

但是客户不太接受这个方案，不过愿意出更高的价格只要我们能够保证解决。

其实客户的这种方案对双方风险都很大。我们最近挺忙，不想接这样的单子。

不过，经不住客户再三沟通，最后简单问了一下客户使用的什么网关评估了一下可能出现的问题。

其中有一个可能是迅时根本没挂断电话，可能是信号音检测问题，也可能是 NAT 穿越问题导致 FreeSWITCH 挂机后 BYE 消息没到达迅时因此电话未挂机，所以导致所有端口都被占用因此打不通电话。

但是为什么重启 FreeSWITCH 就能解决我们还不得而知。

这也是为什么我们要收挂号费，我们需要一些时间做进一步检测以便确认问题。多数情况下，如果解决方法极其简单，我们也就不再多收费用了。或者告诉用户我们解决不了，去其它大医院看看。当然，如果我们认为还得进一步治疗则评估下一步治疗的价格。我们一直是这么做的。

不过，这次不同，客户给了我们一些钱让我们保证解决问题。好吧，艺高人胆大，把单子接下了。

事实证明这是个坑。

事实跟我们想象的完全不同：

- 客户用 Windows 版的 FreeSWITCH。我们更熟悉 Linux，一般都默认是 Linux，在 Linux 上处理问题一般比 Windows 上快很多；
- 客户的 Windows 只能用 Teamviewer 访问，而 Linux 如果 SSH 访问效率会高很多
- 客户使用迅时 MX8 全 FXS 口网关，而我们之前一直以为是 FXO 口网关
- 客户使用 SIP 中继对接到某运营商，但之前客户根本就没跟我们提到这个问题
- 客户实际上是为客户的客户服务，客户也是用 Teamviewer 连接到客户的客户的 Windows 上，而客户的客户描述问题很不清

但骑上虎了也不能轻易就下来，硬着头皮看看。客户一再声称摘机听忙音，但从我们检测后判断是客户拨号后才听忙音的。而且问题原因是 SIP 中继侧不通，跟迅时网关关系不大。

后来发现，外呼流程是这样的：

客户 FSX 话机通过迅时呼到 FreeSWITCH，FreeSWITCH 再通过 SIP 中继呼出去，而问题就出现在 SIP 中继方面，FreeSWITCH 刚启动时是好的，但过一阵再外呼时就回 480，只有重启 FreeSWITCH 才能恢复正常状态。

后来我们检查了一番，决定不重启 FreeSWITCH，而是重新注册一下网关：

```
sofia profile external register 010-NNNNNNNN
```

其中 010-NNNNNNNN 是客户设置的网关的名字。本来是一串数字，保护隐私我们打了马赛克。

果然，注册后呼叫正常。

再查看网关配置，注册周期为默认值，尝试把注册周期改小一点：

```
<param name="expire-seconds" value= "150"/>
```

告诉客户问题解决了，收工。

此处省力一万字。

客户周末都在上班，也够拼的。

客户说问题并没实际解决，而是出问题的周期缩短了。好像每次自动注册后打电话是好的，很短时间内就不好。

好吧，我们告诉客户，对方回 480 这么诡异的问题，应该去找对方。

从 SIP 概念上讲，4 开头的响应码是客户端错误，应该在客户端修正然后重新发起呼叫可能就好了。

是的，注意上句中的可能。

我们已经在尝试修正了，比如改 User

Agent，开启 Ping 等。问题是这种错误要是有对端配合，可能很快就找到原因改好了，但是，如果对端不配合，你可能需要尝试一万种组合才可能解决问题。

果然，我们所有的尝试都失败了。要知道，这种尝试是非常花费时间的。你改了，必须等待它再次出现，再改，再等……死循环

所以事实往往是很悲催的——对端如果是大运营商的话，你能找到的所有人几乎都不懂技术；对端如果是小的“运营商”（姑且称之为是运营商吧）的话，往往也没有几个是懂技术的，他们往往只会告诉你一句话，人家别人用的都是好的……

说到别人，还真有。后来客户又告诉我们一个线索，人家迅时网关直接用这个 SIP 中继注册上去，打电话就没有问题！

也算是个线索吧，抓包，对比，直到我们的包改的跟迅时的一模一样我们也是不能解决问题。

而且我们发现迅时发的 REGISTER 消息 Expire 域为 600，对方回的是 60，但我们在 FreeSWITCH 里也写了 600，对方回的就是 1800。没头绪。

留给我们的时间已经不多了。

好吧，既然迅时好，那就让迅时做这个工作就好了。

改方案。

直接让迅时注册这个 SIP 中继。FSX 口的话机打电话时，先送到 FreeSWITCH，FreeSWITCH 再送到迅时，再通过这个中继出去。有点绕，但是这也算是一种解决方案吧。

迅时跟 FreeSWITCH 对接可以简单的通过 IP 对接，即将呼叫送到 FreeSWITCH 的 5080 端口，不过，这样有点不安全不是。所以，我们需要 FSX 口发起的电话呼叫到达 FreeSWITCH 时，FreeSWITCH 对迅时网关进行认证。

认证方式有两种：

1) 在 FreeSWITCH 中设置 ACL，允许 ACL 中指定的 IP (即迅时网关的 IP) 呼叫 5060 端口。这种方式需要改 ACL 及 Profile 的一些配置，客户没采用这种方式。

虽然没有采用，我们简单说一下配置方案，供参考。

在 ACL 中添加迅时的 IP。这样，迅时网关过来的呼叫就直接信任，不再发 Challenge。但 FreeSWITCH 默认还是走 public Dialplan，可以在 `internal.xml` 里设置将默认的 `context = public` 设成 `context = default` 以及任何其它值。

2) 在 FreeSWITCH 中添加跟 SIP 中继网关上一样的用户名和密码，当迅时来的呼叫到达 FreeSWITCH 5060 端口时发起 Challenge，此时，由于迅时注册到远端的 SIP 中继上，因此会用远端的 SIP 中继提供的用户名和密码回应 FreeSWITCH，即用户名 `010NNNNNNNN` 和密码 `PPPPPPPPP`。

因此，我们就在 FreeSWITCH 中添加了一个用户，用户名是 `010NNNNNNNN` 密码是 `PPPPPPPPP`。

这样，迅时网关就能呼叫 FreeSWITCH 的 5060 端口并正确的认证（仅在呼叫 INVITE 时认证，不注册）。

好了，在迅时网关上的路由是这么设的

```

FXS x ROUTE IP 192.168.1.100 # 任何来自 FSX 的呼叫都路由到 FreeSWITCH
IP 1001 ROUTE FSX 1           # 来自 FreeSWITCH 的呼到，如果被叫号码是 1001，则路由到第一个电话口
IP 1002 ROUTE FSX 2           # 以此类推
IP 1003 ROUTE FSX 3           # 以此类推
IP 1004 ROUTE FSX 4           # 以此类推
IP 1005 ROUTE FSX 5           # 以此类推
IP 1006 ROUTE FSX 6           # 以此类推
IP 1007 ROUTE FSX 7           # 以此类推
IP 1008 ROUTE FSX 8           # 以此类推
IP 010NNNNNNNN ROUTE IP 192.168.1.100:5080 # 来自 SIP 中继的呼叫，如果呼入，则路由到 FreeSWITCH，放 IVR
IP x ROUTE IP 1.2.3.4          # 其它任何号码都路由到 SIP 中继网关，以 1.2.3.4 为例

```

测试拨打一切正常。

以前，迅时网关是直接注册到 FreeSWITCH 上的，因此，一个回呼命令类似如下的样子：

```
originate user/1001 &bridge(sofia/gateway/010-NNNNNNNN/18612345678)
```

其中，`010-NNNNNNNN` 是 SIP 中继网关的名字。

但经我们改动后，迅时不再向 FreeSWITCH 注册，因此没有了本地用户，所以，呼叫命令改成：

```
originate sofia/gatewayxunshi/1001 &bridge(sofia/gateway/xunshi/18612345678)
```

其中 xunshi 时迅时对应的网关的名字。

小结：

- 我们坚持的挂号费是对的，对双方都是有利的
- 不要低估问题的复杂性，显然很多问题也可以分分钟解决，但不可否认，大部分问题都会有意外
- 有些问题不一定非得在 FreeSWITCH 里解决，本例就是一个例子

更新：后来客户反馈说，其中半小时需要重启 FreeSWITCH 还有一个原因，原因就是客户的 FreeSWITCH 是部署在 Windows 的，而 Windows 设置是 30 分钟睡眠… 睡眠，听懂了吗？他们每 30 分钟重启一次…

第十章 调试与排错

10.1 FreeSWITCH Bug 修复一例

好多人都问 FreeSWITCH 崩溃如何调试，昨天，我正好遇到一个崩溃的情况，很快就找到原因并修复了，简单记录一下，供大家参考。

崩溃发生在 master 版本上。在视频会议中，当试图播放一个 PDF 文件时崩溃：

```
conference 3000 play /tmp/test.pdf
```

PDF 文件支持用到了`mod_imagick`模块。该模块是我写的，以前是好的，现在出现了崩溃，说明遇到了意外的情况。

我接下来试了 PNG 文件：

```
conference 3000 play /tmp/test.png
```

一样的崩溃。

PDF 文件支持是在`mod_png`中。接着，我试了类似的 MP4 文件：

```
conference 3000 play /tmp/test.mp4
```

MP4 不崩溃。

MP4 文件在`mod_vlc`和`mod_av`中都有支持，我仅试了`mod_av`。

异同点：`mod_imagick`和`mod_png`崩溃，`mod_av`正常。

conference play用到了File Interface 接口，初步断该接口在mod_av中实现是正确的，而在其它两个模块中有问题。

该文件接口除了可以在conference中调用外，也可以直接用playback，因而，我试了：

```
<action application="playback" data="/tmp/test.pdf"/>
<action application="playback" data="/tmp/test.png"/>
<action application="playback" data="/tmp/test.mp4"/>
```

播放都正常，不崩溃。因而，问题缩小为仅在conference中使用该文件接口时崩溃。

好了，挂上lldb(我在 Mac 上使用llvm，相当于 Linux 上的gdb和gcc)

```
ps aux | grep freeswitch # 找到 FreeSWITCH 进程号
lldb
attach 30918 # FreeSWITCH 进程号
continue      # 继续
```

重新尝试conference 3000 play /tmp/test.pdf，系统崩溃，看lldb中显示以下消息：

```
Process 30918 stopped
* thread #36: tid = 0xb53e6, 0x00000001061b5a6b mod_imagick.so`imagick_file_read_video(handle=<unavailable>,
frame=0x0000000000000000, flags=<unavailable>) + 987 at mod_imagick.c:317, stop reason = EXC_BAD_ACCESS (code=1, a
frame #0: 0x00000001061b5a6b mod_imagick.so`imagick_file_read_video(handle=<unavailable>,
frame=0x0000000000000000, flags=<unavailable>) + 987 at mod_imagick.c:317
314
315     if ((context->reads++ % 20) == 0) {
316         switch_img_copy(context->img, &dup);
-> 317         frame->img = dup;
318         context->sent++;
319     } else {
320         if ((flags && SVR_BLOCK)) {
```

从上面的消息看在执行到mod_imagick.c的317行出错。尝试打印更详细的信息

```
(lldb) print frame->img
error: Couldn't apply expression side effects : Couldn't dematerialize a result variable: couldn't read its memory
(lldb) print frame
(switch_frame_t *) $2 = 0x0000000000000000
```

显示`frame`的值是NULL空指针。其实这一步不是必须的，因为在出错信息的第一行已经显示了`frame`是NULL了。检查了`imagick_file_read_video`函数，发现并没有处理`frame`为NULL的情况。那么，`frame`为什么是NULL了呢？为什么以前是好用的呢？继续跟踪，输入`bt`（back trace）看一下调用栈：

```
(lldb) bt
* thread #36: tid = 0xb53e6, 0x00000001061b5a6b mod_imagick.so`imagick_file_read_video(handle=<unavailable>,
    frame=0x0000000000000000, flags=<unavailable>) + 987 at mod_imagick.c:317, stop reason = EXC_BAD_ACCESS (code=1, a
* frame #0: 0x00000001061b5a6b mod_imagick.so`imagick_file_read_video(handle=<unavailable>,
    frame=0x0000000000000000, flags=<unavailable>) + 987 at mod_imagick.c:317
frame #1: 0x0000000100c41cec libfreeswitch.1.dylib`switch_core_file_read_video(fh=<unavailable>,
    frame=<unavailable>, flags=<unavailable>) + 44 at switch_core_file.c:588
frame #2: 0x00000001036c312f mod_conference.so`fnode_check_video(conference=0x00007fd43b847f20,
    fnode=0x00007fd43b10e320) + 47 at mod_conference.c:1937
frame #3: 0x00000001036be83d
```

配合源代码逐一检查每一层调用，发现在`fnode_check_video`函数中（`mod_conference.c`: 1937行）有如下调用

```
switch_core_file_read_video(&fnode->fh, NULL, SVR_CHECK)
```

该函数调用的第二个参数就是我们要的`frame`，而在调用时就直接传入了NULL指针。由于在`mod_imagick`中没有考虑`frame`是NULL的情况，因而出现崩溃。

问题是，既然以前没有处理NULL的情况一切都是正常的，说明是上面传入NULL的调用是后来又加上的。

果然，通过检查`mod_conference.c`的修改历史，发现在385a3b5这次提交中增加了该函数调用，并且，修改时仅修改了`mod_av`和`mod_vlc`，在里面加入了一项检查：

```
if ((flags & SVR_CHECK)) {
    return SWITCH_STATUS_BREAK;
}
```

而类似的检查函数并没有加到`mod_imagick`和`mod_png`中。因而导致调用时出错。

在上面的检查中，`SVR_CHECK`是原函数调用的第三个参数，在调用时仅检查了该标志值，而没有检查`frame`是否是NULL。因而，我们仅通过阅读代码不容易找到其中的关联性。所幸，这块代码加入的时间不长，我们很快就找到了对应的修改发现了其中的问题。

我把这段代码块也同样加入到了`mod_imagick`和`mod_png`中，就不再崩溃了。

<https://freeswitch.org/fisheye/changelog/freeswitch?cs=08c7a1de344fbe7b061c9da06754a6ed60b3aa66>

10.2 将 SIP Trace 放入日志文件中

今天，看大家在 QQ 群中聊到不知道如何在 FreeSWITCH 中将 SIP Trace 的结果放入日志文件中。我便答应大家我今晚研究一下。

事情的起因是这样的。FreeSWITCH 内置了 SIP Trace，可以很方便的在控制台或 fs_cli 中抓到 SIP 消息，配合日志调试起来非常方便。以前我应该也讲过，在 FreeSWITCH 中开启 SIP Trace 的命令是：

```
sofia global sip trace on
```

当然，也可以单独针对某个 Profile 开启或关闭日志，如：

```
sofia profile internal siptrace on
sofia profile internal siptrace off
```

问题时，开启日志后 SIP 消息只能在控制台上显示，但不会同时进入日志文件中，显然，如果想从日志文件中同时看到 SIP 就比较麻烦了。因而，便出现了今天的主题：如何将 SIP Trace 放入日志？

首先，我想，提到该问题的人可能一般的是 Windows 用户吧。因为一般来说，我在 Mac 上（或 Linux）上，Shell 足够好用，因而，可以毫不费力的在 Shell 窗口（即 FreeSWITCH 控制台上）中将带有 SIP 消息的日志一块 Copy 出来粘贴到其它文件中。但在 Windows 上就没有那么方便了，因为 Windows 的命令行窗口出奇的烂，从里面 Copy 个文字特别麻烦。而且，有很多人根本不知道如何 Copy（虽然确实有办法可以做到）。因而，常见的做法是直接用 QQ 抓屏，但很显然，抓屏是有限制的，那就是，一屏能显示的信息实在是太短了，如果要抓到 SIP 消息，那简直是个累死人的活。

而且，在此我也想插一句，我在帮人看日志时是非常讨厌抓屏的。因为看起来其实不直观，而且，没法搜索，没法 Copy，没法引用…，总之，不爽。

言归正传，所以，特别是 Windows 用户，一般是到日志文件中去找到相应的日志再复制出来，并且，有时候他们也希望 SIP Trace 出来的消息也同时在日志文件里面。

当然，既然我答应大家研究，就一定要出结果的（为什么要研究呢？因为我一直不用，所以没试过）。通过研究我发现，写日志文件的功能是在 mod_logfile 中实现的。里面有这么一行配置：

```
<map name="all" value="debug,info,notice,warning,err,crit,alert"/>
```

到这里科普一下。在 FreeSWITCH 中，日志是有级别的，一般来说就是上面列的一些级别。当然，上面没有包括一个特殊的级别，那就是 console 级别。而 SIP Trace 的消息正是在 console 级别

的，因而，它只有控制台上显示，而不会在日志文件中显示。明白了这个道理，想办法将 console 级别的日志放入日志文件就简单了，修改配置如下：

```
<map name="all" value="console,debug,info,notice,warning,err,crit,alert"/>
```

然后，重新加载模块即可：

```
reload mod_logfile
```

再看 SIP Trace 的结果，都写到日志文件中去了，Bingo!

当然，任何事情并不只有一种办法，第二种办法是什么呢？留点引子，明天再讲。

既然今天说到 mod_logfile，就索性把这一模块也讲了吧。其实，有了上面的，也没什么好讲的。mod_logfile 的作用就是将系统日志写到日志文件中去，并有相应的参数配置文件的路径以及文件的最大长度。如果文件写到一定的大小，则会自动发生轮转（rotate），以防止在长期运行过程中产生巨大无比的日志文件。

昨天讲了将 SIP Trace 放入日志文件的方法。有读者回复说正好用上，这也算是一点功德吧。

当然，昨天还留了一个小尾巴。将 SIP Trace 放入日志文件不止一种方法，其实还有更简单的方法，那就是，可以不用修改任何文件直接在控制台上修改 SIP Trace 日志的级别，如

```
sofia tracelevel info
```

将 Sofia 调试级别设为 info 以后，Trace 出来的日志就自然都到日志文件中去了。另外，在 FreeSWITCH 控制台上也将看到绿色的输出（info 级别的日志默认是绿色显示的）。

该命令极其简单，但还是有好多人不知道。不过，读者群中也有知道这一招的，他给我发来了正确的命令。

当然，你也可以尝试其它颜色，如：

```
sofia tracelevel error
```

使用如下命令改回原来的级别

```
sofia tracelevel console
```

需要指出，该命令不仅对于 SIP Trace 有效，而是对于所有的 sofia 调试信息有效，如，你可以使用如下命令打开 Sofia-SIP 底层协议栈的调试：

```
sofia loglevel all 9
```

然后，所有的调试信息不仅在控制台上输出，也会同时输出到日志里。

下面，再说一点 Windows 命令行窗口的一点小技巧。

有时候还是要从命令行窗口里复制文字信息（再强调一遍，我不喜欢看抓屏）。在命令行窗口标题栏上可以点击右键，选择标记，然后就可以在窗口中选择文字了。选择好以后，再次右击标题栏，点击复制就可以将标记的文字复制到剪贴板了。

当然，命令行窗口默认的缓冲区比较小，能回滚的行数有限，因此，需要看或者复制更多文本的话首先要调整回滚缓冲区，这个，可以右击标题栏选择属性，找一找相关的行数设置（具体的我忘了，一般我也不用 Windows）。当然，现在大家都用大屏分辨率了，顺便改一改窗口的尺寸也起来也不错…

第十一章 FreeSWITCH 文集

本章收集了一些有趣的文章，对了解 FreeSWITCH 的背景，尤其是出去吹牛是很有帮助的。

(注意：2016 年 7 月份，我们把一些 FreeSWITCH 文章单独整理成一本书，就叫《FreeSWITCH 文集》，因此，大部分内容都在新书中更新，本章在未来的版本中可能删去。本书纸质版不包含本章内容。)

11.1 FreeSWITCH 开源社区指南

FreeSWITCH 是一个开源项目，也是一个开发者社区。我们欢迎所有人加入 FreeSWITCH 社区，一起学习、讨论、并贡献自己的力量，使我们的社区和我们的项目越办越好。

中文社区

FreeSWITCH 中文社区的名称是 FreeSWITCH-CN，网址是<http://www.freeswitch.org.cn>，社区的最新消息都会在这里发布。

我们有一个 BBS (<http://bbs.freeswitch.org.cn>)，一个 QQ 群 (190435825) 一个微信公共平台 (FreeSWITCH-CN)。我们每年会不定期的举办中国区的 FreeSWITCH 沙龙线下活动。

鼓励大家在 BBS 上讨论。因为在 BBS 上提问和回答问题都是异步的，也就是说，想给你答案的人可以在任何时候回答，而不用像 QQ 群中那样你提问的问题被其它人冲走了。

另外，我们也建议大家到知乎 (<http://www.zhihu.com>) 上搜索『FreeSWITCH』相关话题并提问，也可以邀请社区里比较爱助人为乐的人回答。知乎是一个专业的问答社区，相信大家都能提出专业的问题，也能得到专业的解答。

在社区中（包括邮件列表、知乎和 QQ 群等）提问时，我们有以下建议：

- 我们是一个公共社区，社区成员来自不同的地方，有着不同的背景，对于同一个问题有着不同的理解。
- 礼貌问答，即使在别人不礼貌的时候也要保持冷静。我们是一个社区、不是战场。
- 还是礼貌，如果别人回答了你的问题，最好说声谢谢。如果别人的回答帮你解决了问题，也最好给一个反馈，让别人知道你的问题已经解决了。

- 如果在 QQ 等即时工具上，不用问『有人吗？』，或者『有人熟悉 XX 吗？』之类的问题，直接提问你自己的问题即可。
- 提问要问到点子上，要言之有物。不要提那些『我装上了 FreeSWITCH，怎么不好用啊』之类的别人无法回答的问题。
- 一般来说，提问时说明问题的现象，你使用的操作系统、版本号、FreeSWITCH 的版本号，贴上必要的日志等，有助于别人帮你快速定位并解决问题。
- 尽量不要使用截屏，有时候截屏很难辨认。
- 不要灌水。一个问题问多好遍也并不一定得到有效的回答，反而会影响大家对你的印象。一个例外是，如果你在 QQ 群中，你可以在一天中的不同时段或不同的日期重复提问，因为这时可能有不同的人上线。如果还没有人回答，而你又想知道答案，可以尝试将这些问题转到邮件列表和知乎上，得到回答的几率大一些。
- 大段的内容（如系统日志等）要贴到 pastebin 或 gist¹ 上。
- 大家都是志愿者，都愿意为社区做贡献，但没有义务回答你的问题。所以，在问题得不到回答时，也不要耍小脾气。
- 要看新手指南、橡皮鸭子问题，以及提问的智慧。多用 Google——其它的搜索引擎很难找到你想要的答案。
- 多做贡献。多为社区做贡献，就会越受到大家的爱戴，同时别人也越愿意帮助你。
- 不要贸然要求别人私下回答你的问题。有问题在公共邮件列表或 QQ 群中讨论，你得到回答的机会比向私人提问得到的机会更多，因为有那样的话会有更多的人看到你提出的问题。同时，别人也更乐意在公共场所回答你的问题，因为那样可以帮助更多与你遇到同样问题的人。
- 把以上这些再看一遍。

英文社区

英文社区的资源比中文社区的多。如果你想在一个国际化的环境中成长的话，建议从现在就开始做。如果英文表达不是很流利，那么也可以从现在开始就『潜水』。下面，简单列举一下英文社区中的一些资源。

- <http://www.freeswitch.org> 是 FreeSWITCH 的官方网站。
- <http://confluence.freeswitch.org> 是英文的 Wiki 网站，有无数的资料。
- <http://lists.freeswitch.org> 是一个邮件列表。大部分用户可以从 freeswitch-users 邮件列表开始。
- <http://freenode.net> 是一个 IRC 站点，其中有一个『#freeswitch』频道，大家都在那里交流，类似于中国的 QQ 群，但这个交流工具比 QQ 历史悠久得多。
- <http://jira.freeswitch.org> 是一个缺陷跟踪系统，你可以在那里汇报 Bug，或者提新需求。当然，如果希望你的新需求快速得到响应，也可以提一个 Bounty。一个 Bounty 的意思是说你希望实现一个新需求并愿意为之付费。

¹ 参见<http://pastebin.freeswitch.org> 及<http://gist.github.com>。

- <http://files.freeswitch.org> 上有 FreeSWITCH 各种资源的下载，包括源代码和已编译的程序等。
- <http://cluecon.com> 是 ClueCon 的主站。ClueCon 是一年一度的开发者技术交流会，每年 8 月份都在美国芝加哥举行。
- <https://freeswitch.org/confluence/display/FREESWITCH/ClueCon+Weekly+Conference+call> 列出了 FreeSWITCH 电话会议的参与方式。电话会议是 FreeSWITCH 社区成员使用语音交流的一种方式。基本上是每周三 UTC 时间下午 6 点开始，每周五不定时也有 Friday Free For All 的会议。支持使用 WebRTC、Flash、SIP、H323 以及 PSTN 等多种方式参加会议。该页面上也有历史会议的录音。

其它

总之，只要你学习或者使用 FreeSWITCH 就是为 FreeSWITCH 做贡献。不要不好意思说，有问题也不要不好意思问。众人捧柴火焰高嘛。

11.2 关于 FreeSWITCH 常用术语翻译的意见

本文所指翻译主要指英译汉。

众所周知，翻译的最高境界是『信、达、雅』，通俗来讲就是『正确、易懂、得体』，它实际上是从『内容、表达、风格』三方面来要求译作。但拘泥于以上标准往往回受到很多局限。尤其对于科技作品来说，用上述观点来评价翻译作品则更难保证评价的正确性和客观性。

按照信息论的观点，翻译的本质就是通过语种转换把一种语言的言语所承载的信息转移到另一种语言的言语当中，用该种语言表达出来以传递给目标语言读者的过程，是把一种语言的言语转换成另一种语言的言语的活动。由此看来，译文是信息从原著传递到目标语言读者的中间载体。因此，翻译包括原著与目标语言读者两个对象。那么，我们在翻译时，除了应该忠实于原文外，更应该考虑的是目标读者。比方说，某件事情或动作，对于原著语言的读者来说可能是习以为常的，但对于目标语言的读者来说可能很陌生或根本不知道是怎么回事。这时，我们就应该允许译者加入相关背景知识的脚注，或者直接用目标语言读者所熟悉的、类似的表达方式来表达。当然，我们这里的目的是为了讨论翻译，在此，我仅就 FreeSWITCH 资料的翻译来讲一下笔者的观点：

FreeSWITCH 资料大部分来自其官方 Wiki 或者邮件列表，是来自全球众多网友的贡献。其作者大部分是程序员或技术人员，因此他们大都不太注重语法和修辞，并且这种网络媒体写起来本身就比较自由，再加上还有好多俚语和网络语言，有不少文章也是出自非英语母语的人之手，就更难保证语法的正确性了，因而翻译起来就非常困难。在此，我主张，遇到某些难以翻译的句子时，尽量把长句拆成短句，在充分了解原作的基础上甚至可以用自己的语言表达（Anthony 最喜欢写长句了）。因为我们最重要的任务是读中文的人能理解它，而不是一味地『忠实』原文。

关于英文中的术语，可译可不译。如：没有人会把『HTTP』译成『超文本传输协议』，也没有人把 XML 译成『可扩展标记语言』。为什么？因为它们应该是众所周知的，至少在技术领域是应该

是这样的。但对于某些特定领域的术语，有一些甚至在中文中没有对应的词，怎么办呢？最简单的答案是不译，这些即能最大程度地忠实原文，又不会让读者不知所云。当然，如果你的水平很高，你创造一个新词也可能成为国际标准，那就另当别论了。

对于某些缩写，最好能标注原文。有些书会在附录里列一个术语对照表，但我认为大多数情况下还是直接注在正文(用括号或脚注)中较好，而且只需在第一次出现时加注。如 PSTN(Public Switched Telephone Network，公共电话交换网)、ENUM(E.164 NUmber Mapping，E164 号码映射)。当然，对于非专业的读者来说，他们可能还是不知道是什么，如果你面向的是这样的读者，那最好在脚注里(或超文本里)加个指向维基百科的链接(如果有人问『维基百科是什么』该怎么办？)。举个例子，大部分人都不知道自己家电话所在的网络叫『公用电话交换网』，更不用说 PSTN 了。那么，在某些场合说到它与 VoIP 的区别时，你也可以称之为『普通电话网』或『传统电话网』。

关于标点。标点在文章中也是很重要的。对于括号，可以用英文的也可以用中文的，但必须统一。引号如果只表示一个名字时，在大多数情况下都可以不用。如上段中的『普通电话网』，如果不加引号，『你也可以称为普通电话网或传统电话网』一句是没有歧义的，作者加了引号是为了强调。当然，在本段中，你必须加引号，因为本段中是引用。

当然，如果你的译作不是发在正式的出版物中，而只是在论坛里灌水或贴在自己的 Blog 上，那就随便怎么写都可以了。

以下是作者对 FreeSWITCH 相关术语的译法，不当之处请广大读者批评指正。

- **Dialplan**: 拨号计划。虽然在 Asterisk 的书中译为『拨号方案』，但笔者还是认为『拨号计划』较好。
- **Endpoint**: 终点。专有名词，可不译。
- **ASR/TTS**: 自动语音识别/文本语音转换（语音合成）。
- **Directory**: FreeSWITCH 中的用户目录，或者通用的目录服务（如 LDAP）。
- **Events**: 事件。
- **Event handlers**: 事件句柄。
- **Event Socket**: 事件套接字，可不译。
- **Formats**: 格式。
- **Loggers**: 日志。
- **Languages**: 语言。
- **Say**: 说。可不译。
- **Phrase**: 短语。可不译。
- **Timers**: 定时器/计时器。
- **Applications**: 应用。
- **FSAPI**: FreeSWITCH 应用程序接口，也可译为 FreeSWITCH 命令接口。
- **CDR**: 呼叫详单/呼叫详细记录。
- **PSTN**: 公共电话交换网。
- **SIP**: 会话初始协议。

- **SIP UA**: SIP 用户代理。
- **UAC/UAS**: 用户代理客户端/服务器端。
- **Sofia Profile**: Sofia 配置文件，可不译。
- **VoIP**: 不译。

11.3 FreeSWITCH 背后的故事(译)

Anthony Minessale/文Seven Du/译

本文由 Anthony Minessale 写于 2007 年 5 月。来自[www.freeswitch.org²](http://www.freeswitch.org)。翻译它是因为我觉得永远都不会过时… — 译者注

我开发 FreeSWITCH 已经近两年的时间了。在我们第一个发行版即将发布的黎明之际，我想花一点时间来与大家分享一下软件背后的故事，并透露一点即将到来的消息。本故事也将会上登在[OST Magazine](#)第一期上。呵，去下载一份吧，免费的！

写 FreeSWITCH 的想法诞生于 2005 年春的一次 Asterisk 开发者大会上。当时，我为 Asterisk 1.2 版贡献了大量的代码和新特性，并为其以后的发展提了好多思路。我们都认为在当时的 Asterisk 代码树中存在很严重的限制，并且面临着一些问题 修正一些问题时不仅会牺牲很多特性，而且还会花大量的时间。一种思路是建立新的代码分支，通过将大家已经熟悉的代码分开，新的分支就不会影响 Asterisk 用户的使用，从而能减轻好多开发的压力。问题是开发者都不希望将精力分散到同一份代码的两个版本上，因为那样他们就不得不将 BUG 修正和代码修改在两个分支间拷来拷去。我的建议是：在一个单独的代码库上从头创建一个 Asterisk 2.0 的分支，等一切就绪后再对外发布。我想那个想法确实打动了一些开发人员，但现实是，由于讨论的时间过长，最终大家都失去了兴趣。不过，我没有。

很明显，我是唯一一个认真考虑这一问题的。接下来我用了几天的时间在做一个白日梦 如何设计一个新的电话系统。之后，我再也无法抑制，便创建了一个新的目录开始从头写 choir.c。是的，Choir 是我为该工程选的第一个名字。我希望不同的通信部件能够步调一致地协同工作，就像教堂的唱诗班那样优雅和谐 (perfect harmony)。我又用了 5 天时间把我想到的点子都组织到几个文件里。最初的努力并不能装配成一个电话交换机。我知道，我需要创建一个稳定的核心，并应该能跨平台。所以最简单的是使用 Apache APR 库来搭建一个基本的子系统，让它能够动态装载共享模块并悠闲地停在屏幕上等待 shutdown 命令。实现这些代码用了另外 6 个月的时间。

在那 5 天里，我知道了写一个达到那种要求的可伸缩性程序并不是一件简单的事情。我最初的目标是捡起那些 Asterisk 丢掉的东西并加以改善。但随着思考的深入，我越来越觉得，实际上我想改进的是最基础的设计和功能。最终我得出结论，Asterisk 不能实现一些我期望的功能是因为它不是我所需要的软件。也就在那时，我知道我不是要编写另外一个 PBX，而是要开发一个完全不同级别的应用程序。我用了后续的几个月时间组织了第一届 ClueCon 年会来讨论如何合理的设计 Choir。我希望在继续写任何代码之前能确信我有正确的计划。从这一点上说，我仍在做相当的一部分 Asterisk 开

²<http://www.freeswitch.org/node/60>

发，并且我也在我写的一些第三方模块中来实验我的想法。另外，我也让我的同事们花了相当多的时间来争论在电话领域里如何『正确』地做事情，这也算是一种前瞻性吧。

在那年的 ClueCon 会议上，我有幸遇到一些在电话领域里很有影响力的开发者，并把很多灵感带回了家。同时，Asterisk 阵营中的关系开始有些紧张，因为有几个开发者也打算建立新的分支。新的项目称为 OpenPBX（现在叫 Call Weaver），从某种意义上讲它引起了 Asterisk 社区的一次严重地分裂。最初，OpenPBX 把我的许多第三方 Asterisk 模块加入到他们的新代码中，并就如何增强稳定性方面咨询我的建议。其中有一点，甚至他们还来看我是如何在我的新项目中实现的。可能我们能再打起精神来重写那些老的计划，但最终没有实现。不过，我想，最意义深远的一刻是 当有人问我：『多长时间以后它才能打电话呢？』我不知道，所以我决定搞清楚。简短的回答是一星期。

与我想达到的目标相比，能打电话只是一个小小的胜利。我还有很多要做的事情。这一点不起眼的业绩得不到太多关注。好消息是，这一项目最终上路了。我深居简出，用了三个月时间试图能做出点能吸引公众眼球的东西。那段时间，项目的名字曾改为 Pandora 并最终成为 FreeSWITCH。2006 年 1 月我们公开的 SVN 仓库和一个邮件列表上线了。那时仅有几个模块和一个很短的特性列表。但我们有了一个可以工作的内核，它能够在包括 Mac OS X, Linux 和 BSD 的几个 UNIX 变种上编译和运行，并能在 Microsoft Windows 上以一个控制台程序运行。

随着时间的推移，我们也越加有动力。有时也停下来犯几个错误，然后继续前进，写几个新的模块。在试验了四个不同的 SIP 终点模块后，我们决定使用 Sofia SIP。也曾试验过 5 个不同的 RTP 协议栈，最终决定还是我们自己写。同时我也开发了 mod_dingaling 来做与 Google Talk 的接口，以及一个多特性的会议桥。在第一年里，我主要关注如何在使核心尽量稳定的基础上，提供几个外部接口以便于其他模块的开发。如用于 IVR 的嵌入式的 javascript，一个 XML-RPC 接口和一个用于远程控制和事件监控的基于 TCP 的 Socket 的接口。

第二年的 ClueCon 大会，那一天仿佛就是我白日梦醒来的日子。我第一次向我的同行们演示了 FreeSWITCH。近九个月后，在距离第一个想法两年之际，第三届 ClueCon 一个月前，我们达到了开发的 BETA 阶段 FreeSWITCH 1.0 发布在际了。我们也吸引了一些勇敢的开发者一道。他们已经把 FreeSWITCH 用于生产系统，并给我们提供了保证我们第一个发布版本成功的很重要的反馈。

在发布之前，最后一点工作是我新写的一个叫做 OpenZAP 的开源的 TDM 抽象库。使用 BSD 协议的 mod_openzap 模块将取代当时特定于 Sangoma 的 mod_wanpipe，并提供 Sangoma 及其他几种 TDM 硬件（只需要开发对应的模块）支持。OpenZAP 也将为模拟和 ISDN 信令提供一个简单的接口。OpenZAP 的指导思想是 应用程序能使用同样的 API 去控制任何它所支持的 TDM 硬件。它提供一种方式能将所有不同的特性规范化。如果一种卡缺少某种特性，那么它就能以软件的形式实现，不管是在 OpenZAP 库中还是在与生产商硬件 API 通信的接口程序中。

我们在如此短的时间内做这么多事听起来好像是不可能的，但我想，最终，还是像格言里说的：『需要是发明之母。』接下来的路还很长。但我想就此机会感谢所有曾经帮助我们走了这么远的人。下面列表中是所有在我们 AUTHORS 文件中的人：

- Anthony Minessale II (就是我!)

- Michael Jerris (我们极具价值的编译专家和跨平台专家 cross-platformologist, [呵, 这个词是我创造的])
- Brian K. West (我们挚爱的 Mac 权威, 没有他的帮助, 我们将寸步难行)
- Joshua Colp (帮助我们做了第一个 SIP 模块, 虽然现在我们已经不用了)
- Michal “cypromis” Bielicki (他从第一天就加入进来了, 感谢信任!)
- James Martelletti (把 mono 集成进了 FreeSWITCH.)
- Johny Kadarisman (帮我们弄好了 python 模块)
- Yossi Neiman (写了 mod_cdr 收集通话详单)
- Stefan Knoblich (在我们的 SIP 之旅上帮助甚多)
- Justin Unger (找到很多 BUG)
- Paul D. Tinsley (SIP presence 以及其他好的建议)
- Ken Rice (为什么有这个名字? 它给了我们做了很多测试和补丁)
- Neal Horman (在会议模块上有巨大贡献)
- Michael Murdock (我们 CopperCom 的朋友, 有大量反馈和补丁)
- Matt Klein (大量 SIP 帮助, 将帮我们确保 FreeSWITCH 运行于 FreeBSD.)
- Justin Cassidy (幕后工作者, 确保一切正常)
- Bret McDanel (敢吃螃蟹的人, 试验了绝大多数的功能, 最早发现了很多隐藏的 BUG, 我指的是复活节彩蛋!)

11.4 FreeSWITCH 与 Asterisk

Anthony Minessale/文 Seven Du/译

VoIP 通信, 与传统的电话技术相比, 不仅仅在于绝对的资费优势, 更重要的是很容易地通过开发相应的软件, 使其与企业的业务逻辑紧密集成。Asterisk 作为开源 VoIP 软件的代表, 以其强大的功能及相对低廉的建设成本, 受到了全世界开发者的青睐。而 FreeSWITCH 作为 VoIP 领域的新秀, 在性能、稳定性及可伸缩性等方面则更胜一筹。本文原文在<http://www.freeswitch.org/node/117>, 发表于 2008 年 4 月, 相对日新月异的技术来讲, 似乎有点过时。但本文作为 FreeSWITCH 背后的故事, 仍很有翻译的必要。因此, 本人不揣鄙陋, 希望与大家共读此文, 请不吝批评指正。—译者注

FreeSWITCH 与 Asterisk 两者有何不同? 为什么又重新开发一个新的应用程序呢? 最近, 我听到很多这样的疑问。为此, 我想对所有在该问题上有疑问的电话专家和爱好者们解释一下。我曾有大约三年的时间用在开发 Asterisk 上, 并最终成为了 FreeSWITCH 的作者。因此, 我对两者都有相当丰富的经验。首先, 我想先讲一点历史以及我在 Asterisk 上的经验; 然后, 再来解释我开发 FreeSWITCH 的动机以及我是如何以另一种方式实现的。

我从 2003 年开始接触 Asterisk, 当时它还不到 1.0 版。那时对我来讲, VoIP 还是很新的东西。我下载并安装了它, 几分钟后, 从插在我电脑后面的电话机里传出了电话拨号音, 这令我非常兴奋。接下来, 我花了几分钟的时间研究拨号计划, 绞尽脑汁的想能否能在连接到我的 Linux PC 上的电话上

实现一些好玩的东西。由于做过许多 Web 开发，因此我积累了好多新鲜的点子，比如说根据来电显示号码与客户电话号码的对应关系来猜想他们为什么事情打电话等。我也想根据模式匹配来做我的拨号计划，并着手编写我的第一个模块。最初，我做的第一个模块是app_perl，现在叫做res_perl，当时曾用它在 Asterisk 中嵌入了一个 Perl5 的解释器。现在我已经把它从我的系统中去掉了。

后来我开始开发一个 Asterisk 驱动的系统架构，用于管理我们的呼入电话队列。我用app_queue和现在叫做 AMI（大写字母总是看起来比较酷）的管理接口开发了一个原型。它确实非常强大。你可以从一个 T1 线路的 PSTN 号码呼入，并进入一个呼叫队列，坐席代表也呼入该队列，从而可以对客户服务。非常酷！我一边想一边看着我的可爱的 Web 页显示着所有的队列以及他们的登录情况。并且它还能周期性的自动刷新。令人奇怪的是，有一次我浏览器一角上的小图标在过了好长时间后仍在旋转。那是我第一次听说一个词，一个令我永远无法忘记的词—死锁。

那是第一次，但决不是最后一次。那一天，我几乎学到了所有关于 GNU 调试器的东西，而那只是许多问题的开始。队列程序的死锁，管理器的死锁。控制台的死锁开始还比较少，后来却成了一个永无休止的过程。现在，我非常熟悉『段错误（Segmentation Fault）』这个词，它真是一个计算机开发者的玩笑。经过一年的辛勤排错，我发现我已出乎意料的非常精通 C 语言并且有绝地战士般的调试技巧。我有了一个分布于七台服务器、运行于 DS3 TDM 信道的服务平台。与此同时，我也为这一项目贡献了大量的代码，其中有好多是我具有明确版权的完整文件³。

到了 2005 年，我已经俨然成了非常有名的 Asterisk 开发者。他们甚至在 CREDITS 文件以及《Asterisk，电话未来之路》这本书中感谢我。在 Asterisk 代码树中我不仅有大量的程序，而且还有一些他们不需要或者不想要的代码，我把它们收集到了我的网站上(至今仍在<http://www.freeswitch.org/node/50>)。

Asterisk 使用模块化的设计方式。一个中央核心调入称为模块的共享目标文件以扩展功能。模块用于实现特定的协议（如 SIP）、程序（如个性化的 IVR）和其他外部接口（如管理接口）等。Asterisk 的核心是多线程的，但它非常保守。仅仅用于初始化的信道以及执行一个程序的信道才有线程。任何呼叫的 B 端都与 A 端都处于同一线程。当某些事件发生时（如一次转移呼叫必须首先转移到一个称作伪信道的线程模式），该操作把一个信道所有内部数据从一个动态内存对象中分离出来，放入另一个信道中。它的实现在代码注释中被注明是『肮脏的』⁴。反向操作也是如此，当销毁一个信道时，需要先克隆一个新信道，才能挂断原信道。同时也需要修改 CDR 的结构以避免将它视为一个新的呼叫。因此，对于一个呼叫，在呼叫转移时经常会看到 3 或 4 个信道同时存在。

这种操作成了从另一个线程中取出一个信道事实上的方法，同时它也正是开发者许许多多头痛的源头。这种不确定的线程模式是我决定着手重写这一应用程序的原因之一。

Asterisk 使用线性链表管理活动的信道。链表通过一种结构体将一系列动态内存串在一起，这种结构体本身就是链表中的一个成员，并有一个指针指向它自己，以使它能链接无限的对象并能随时访问它们。这确实是一项非常有用的编程技术，但是，在多线程应用中它非常难于管理。在线程中必须

³<http://www.cluecon.com/anthm.html>

⁴/* XXX This is a seriously wacked out operation. We're essentially putting the guts of the clone channel into the original channel. Start by killing off the original channel's backend. I'm not sure we're going to keep this function, because while the features are nice, the cost is very high in terms of pure nastiness. XXX */

使用一个信号量（互斥体，一种类似交通灯的东西）来确保在同一时刻只有一个线程可以对链表进行写操作，否则当一个线程遍历链表时，另一个线程可能会将元素移出。甚至还有比这更严重的问题当一个线程正在销毁或监听一个信道的同时，若有另外一个线程访问该链表时，会出现『段错误』。『段错误』在程序里是一种非常严重的错误，它会造成进程立即终止，这就意味着在绝大多数情况下会中断所有通话。我们所有人都看到过『防止初始死锁』⁵这样一个不太为人所知的信息，它试图锁定一个信道，在 10 次不成功之后，就会继续往下执行。

管理接口（或 AMI）有一个概念，它将用于连接客户端的套接字(socket)传给程序，从而使你的模块可以直接访问它。或者说，更重要的是你可以写入任何你想写入的东西，只要你所写入的东西符合 Manager Events 所规定的格式（协议）。但遗憾的是，这种格式没有很好的结构，因而很难解析。

Asterisk 的核心与某些模块有密切的联系。由于核心使用了一些模块中的二进制代码，当它所依赖的某个模块出现问题，Asterisk 就根本无法启动。如果你想打一个电话，至少在 Asterisk 1.2 中，除使用 `app_dial` 和 `res_features` 外你别无选择，这是因为建立一个呼叫的代码和逻辑实际上是在 `app_dial` 中，而不是在核心里。同时，桥接语音的顶层函数实际上包含在 `res_features` 中。

Asterisk 的 API 没有保护，大多数的函数和数据结构都是公有的，极易导致误用或被绕过。其核心非常混乱，它假设每个信道都必须有一个文件描述符，尽管实际上某些情况下并不需要。许多看起来是一模一样的操作，却使用不同的算法和截然不同的方式来实现，这种重复在代码中随处可见。

这仅仅是我遇到的最多的问题一个简要的概括。作为一个程序员，我贡献了大量的时间，并贡献了我的服务器来作为 CVS 代码仓库和 Bug 跟踪管理服务器。我曾负责组织每周电话会议来计划下一步的发展，并试图解决我在上面提到过的问题。问题是，当你对着长长的问题列表，思考着需要花多少时间和精力来删除或重写多少代码时，解决这些问题的动力就渐渐的没有了。值得一提的是，没有几个人同意我的提议并愿意同我一道做一个 2.0 的分支来重写这些代码。所以在 2005 年夏天我决定自己来。

在开始写 FreeSWITCH 时，我主要专注于一个核心系统，它包含所有的通用函数，即受到保护又能提供给高层的应用。像 Asterisk 一样，我从 Apache Web 服务器上得到很多启发，并选择了一种模块化的设计。第一天，我做的最基本的工作就是让每一个信道有自己的线程，而不管它要做什么。该线程会通过一个状态机与核心交互。这种设计能保证每一个信道都有同样的、可预测的路径和状态钩子，同时可以通过覆盖向系统增加重要的功能。这一点也类似其他面向对象的语言中的类继承。

做到这点其实不容易，容我慢慢讲。在开发 FreeSWITCH 的过程中我也遇到了段错误和死锁（在前面遇到的多，后来就少了）。但是，我从核心开始做起，并从中走了出来。由于所有信道都有它们自己的线程，有时候你需要与它们进行交互。我通过使用一个读、写锁，使得可以从一个散列表（哈希）中查找信道而不必遍历一个线性链表，并且能绝对保证当一个外部线程引用到它时，一个信道无法被访问也不能消失。这就保证了它的稳定，也不需要像 Asterisk 中『Channel Masquerades』之类的东西了。

FreeSWITCH 核心提供的大多数函数和对象都是有保护的，这通过强制它们按照设计的方式运行来实现。任何可扩展的或者由一个模块来提供方法或函数都有一个特定的接口，从而避免了核心

⁵ Avoiding initial deadlock

对模块的依赖性。

整个系统采用清晰分层的结构，最核心的函数在最底层，其他函数分布在各层并随着层数和功能的增加而逐渐减少。

例如，我们可以写一个大的函数，打开一个任意格式的声音文件向一个信道中播放声音。而其上层的 API 只需用一个简单的函数向一个信道中播放文件，这样就可以将其作为一个精简的应用接口函数扩展到拨号计划模块。因此，你可以从你的拨号计划中，也可以在你个性化的 C 程序中执行同样的 playback 函数，甚至你也可以自己写一个模块，手工打开文件，并使用模块的文件格式类服务而无需关注它的代码。

FreeSWITCH 由几个模块接口组成，列表如下：

- 拨号计划(Dialplan): 实现呼叫状态，获取呼叫数据并进行路由。
- 终点(Endpoint): 为不同协议实现的接口，如 SIP, TDM 等。
- 自动语音识别/文本语音转换(ASR/TTS): 语音识别及合成。
- 目录服务(Directory): LDAP 类型的数据库查询。
- 事件(Events): 模块可以触发核心事件，也可以注册自己的个性事件。这些事件可以在以后由事件消费者解析。
- 事件句柄(Event handlers): 远程访问事件和 CDR。
- 格式(Formats): 文件格式如 wav。
- 日志(Loggers): 控制台或文件日志。
- 语言(Languages): 嵌入式语言，如 Python 和 JavaScript。
- 语音(Say): 从声音文件中组织话语的特定的语言模块。
- 计时器(Timers): 可靠的计时器，用于间隔计时。
- 应用(Applications): 可以在一次呼叫中执行的程序，如语音信箱(Voicemail)。
- FSAPI(FreeSWITCH 应用程序接口): 命令行程序，XML RPC 函数，CGI 类型的函数，带输入输出原型的拨号计划函数变量。
- XML: 到核心 XML 的钩子可用于实时地查询和创建基于 XML 的 CDR。

所有的 FreeSWITCH 模块都协同工作并仅仅通过核心 API 或内部事件相互通信。我们非常小心地实现它以保证它能正常工作，并避免其他外部模块引起不期望的问题。

FreeSWITCH 的事件系统用于记录尽可能多的信息。在设计时，我假设大多数的用户会通过一个个性化的模块远程接入 FreeSWITCH 来收集数据。所以，在 FreeSWITCH 中发生的每一个重要事情都会触发一个事件。事件的格式非常类似于一个电子邮件，它具有一个事件头和一个事件主体。事件可被序列化为一个标准的 Text 格式或 XML 格式。任何数量的模块均可以连接到事件系统上接收在线状态，呼叫状态及失败等事件。事件树内部的**mod_event_socket** 可提供一个 TCP 连接，事件可以通过它被消费或记入日志。另外，还可以通过此接口发送呼叫控制命令及双向的音频流。该套接字可以通过一个正在进行的呼叫进行向外连接(Outbound)或从一个远程机器进行向内 (Inbound)连接。

FreeSWITCH 中另一个重要的概念是中心化的 XML 注册表。当 FreeSWITCH 装载时，它打开一个最高层的 XML 文件，并将其送入一个预处理器。预处理器可以解析特殊的指令来包含其他小的 XML 文件以及设置全局变量等。在此处设置的全局变量可以在后续的配置文件中引用。

如，你可以这样用预处理指令设置全局变量：

```
<X-PRE-PROCESS cmd="set" data="moh_uri=local_stream://moh"/>
```

现在，在文件中的下一行开始你就可以使用`$$ {moh_uri}`，它将在后续的输出中被替换为`local_stream://moh`。处理完成后 XML 注册表将装入内存，以供其他模块及核心访问。它有以下几个重要部分：

- 配置文件：配置数据用于控制程序的行为。
- 拨号计划：一个拨号计划的 XML 表示可以用于`mod_dialplan_xml`，用以路由呼叫和执行程序。
- 短语：可标记的 IVR 分词是一些可以『说』多种语言的宏。
- 目录：域及用户的集合，用于注册及账户管理。

通过使用 XML 钩子模块，你可以绑定你的模块来实时地查询 XML 注册表，收集必要的信息，以及返回到呼叫者的静态文件中。这样你可以像一个 WEB 浏览器和一个 CGI 程序一样，通过同一个模型来控制动态的 SIP 注册，动态语音邮件及动态配置集群。

通过使用嵌入式语言，如 Javascript、Java、Python 和 Perl 等，可以使用一个简单的高级接口来控制底层的应用。

FreeSWITCH 工程的第一步是建立一个稳定的核心，在其上可以建立可扩展性的应用。我很高兴的告诉大家在 2008 年 5 月 26 日将完成 FreeSWITCH 1.0 PHOENIX 版。有两位敢吃螃蟹的人已经把还没到 1.0 版的 FreeSWITCH 用于他们的生产系统。根据他们的使用情况来看，我们在同样的配置下能提供 Asterisk 10 倍的性能。

我希望这些解释能足够概括 FreeSWITCH 和 Asterisk 的不同之处以及我为何决定开始 FreeSWITCH 项目。我将永远是一个 Asterisk 开发者，因为我已深深的投入进去。并且，我也希望他们在以后的 Asterisk 开发方面有新的突破。我甚至还收集了很多过去曾经以为已经丢失的代码，放到我个人的网站上供大家使用，也算是作为我对引导我进入电话领域的这一工程的感激和美好祝愿吧。

Asterisk 是一个开源的 PBX，而 FreeSWITCH 则是一个开源的软交换机。与其他伟大的软件如 Call Weaver、Bayonne、sipX、OpenSER 以及更多其他开源电话程序相比，两者还有很大发展空间。我每年都期望能参加在芝加哥召开的 ClueCon 大会，并向其他开发者展示和交流这些项目（<http://www.cluecon.com>）。

我们所有人都可以相互激发和鼓励以推进电话系统的发展。你可以问的最重要的问题是：『它是完成该功能的最合适工具吗？』

11.5 FreeSWITCH 的历史⁶

Anthony Minessale/文 杜金房/译

为了恰当地介绍 FreeSWITCH 的起源，我们必须回到从前，那时，我们甚至还没想到要实现它。VoIP 革命真正的开始与成型是在世纪之交，以开源的 Asterisk PBX 和 OpenH323 项目为主要标志。这两个软件的革命先驱使得很多开发者得以访问 VoIP 的资源而无需付出高昂的商业解决方案的费用。这两个项目后来又导致了很多创新，真正的可能的 IP 电话通信是真实存在的被迅速传播开来。

我在 2002 年第一次进入这一行业。当时我们公司的业务是对外技术支持外包，我们需要一种方式管理电话呼叫，并把呼叫送到一个线下的地方（原文是 an off-site location）。当时我们用的是的解决方案，但是那种方案不仅部署费用太贵，而且我们还得支付不菲的每座席的费用。作为一个 Web 平台的架构师，我在过去的工作中曾经基于开源项目如 Apache 和 MySQL 做过很多开发，所以我决定研究一下在电话方面有没有相关的开源解决方案。很自然地我找到了 Asterisk。

当我第一次下载了 Asterisk 的时候，我惊呆了。为了使它工作，我找来了好多模块电话板卡，然后中，在我家里，装在我的 Linux PC 机上，当在后面插上电话线并在话机上听到第一声拨号音之后，我叫到：哇塞！简直是帅呆了！我很快就深入代码中，试着研究出它是怎么工作的。我很快的学到，类似 Apache，该软件竟然也可以以可加载模块的方式扩展它的功能以做其它的有用的事情。这比以前任何的东西都要好。现在，我不仅可以使自己的电话可以与计算机通信，我还可以让它在拨打特定号码的情况下执行我自己写的代码。

我尝试并验证了几种想法之后，忽然，我产生了一个新想法这些想法：嘿，我非常喜欢 Perl⁷，并且这些电话系统非常的酷，如果我把它们结合起来会怎么样？我研究了如何把 Perl 嵌入 C 语言程序的文档，很快我就有了一个 `app_perl.so`，它是一个 Asterisk 的可加载模块，通过该模块可以在电话路由到我的模块以后执行我的 Perl 代码。当时它并不完美，然后我开始很快地学习将 Perl 嵌入一个多线程的程序中的各种挑战。但至少是对我的想法的可行性的一种概念验证，在经过几天的修修补之后能够得以运行也算是一个不小的成熟了。

随后，我深入了 Asterisk 在线社区。在这些代码上玩了几周的之后，我用 Asterisk 作为电话引擎开始做一个呼叫中心解决方案，以及一个简单的 Web 前端程序。在实现的过程中，我遇到了 Asterisk 中的几个 Bug，然后我就把它们提交到了 Asterisk 开发分支的缺陷跟踪系统上。该过程重复地越多，我参与该项目的成长和发展就越深。除了零散的 Bug 修复之外，我也开始写代码对它进行改进。到 2004 年的时候，我除了修复我报告的 Bug 之外，也在修复其它人报告的 Bug 了。这是我感觉在我找到我自己的问题的免费解决方案以后，我所能做到的回报方式。如果我的问题解决了，大家也都能看到。

事件升级

当我在测度我的程序的时候，我会往系统中打很多电话并看着一个 Web 页面在更新、控制呼叫队列，以及看着各种状态统计信息。然而，我没有注意到的一件事就是并发的呼叫数以及呼叫量本

⁶本文译自《FreeSWITCH 1.2》(PACK 出版社, 2013)，附录 C: The History of FreeSWITCH。翻译得到作者授权。

——译者注

⁷一种编程语言。——译者注。

身。确实我在测试的时候也就一般同时打一两个电话，而没有全面的测试我的程序。当我第一次将程序放到生产系统上的时候，也是我第一次看到一个多线程的软件遇到无法解决的锁冲突的时候，这种锁冲突就是众所周知的死锁。我非常熟悉段错误⁸，因为我在开发自己的模块的时候遇到过无数次。但是，使我不解的是，我有时看到在某些非常无法解释的情况下也出现这种错误。

段错误是由于一个程序在运行时进行了不正当的内存访问，如多次释放同一段内存或者试图越界访问内存地址或者访问根本不存在的内存。由于你可以直接访问底层的操作系统，并且除非你非常自律，系统无法防止你犯错误，所以，在 C 语言编写的程序中这种错误是很常见的。但我不是轻言放弃的人，那样的话你会认为是一个诅咒抑或是恩赐。所以，当我遇到问题时，我已经准备好了奋斗到底。我花了无数的时间研究 GNU 调试器的输出，并尝试模拟出大量的呼叫以便重现我遇到的问题。经过一些试错（原文是 *trial-and-error*），我成功了。我终于通过一个呼叫发生器的帮助找到了导致系统崩溃方法。当时的感觉非常好，只是好的感觉太短暂。就在那天下午，我又在代码的另一处发现了另外一个类似的新问题。

我尝试小心的去掉我的程序中的一些可能导致死锁或崩溃的功能，便是我无法去掉全部。最终我发现导致我的不幸的是 `app_queue` 模块，这对我来说可不是个好消息，因为在我的呼叫中心程序中我主要就用了那个模块。我修复了那个模块中的问题，但一些修改对原有的模块影响太大，因而无法包含到主流的发布代码中。最后，我还是自己维护了我的模块代码，并继续更新 Asterisk 中其它的部分。这总算令它稳定下来了，但这种稳定只是在找到另一种解决方案前相对稳定的方案。

到那时为止，我往 Asterisk 里加入了很多的新特性，并对开发一些性功能有了很好的想法。我创建了一个新的概念，叫做功能变量（function variables），它允许模块可以对外提供一个接口，通过该接口，能从拨号计划（Dialplan）中扩展模块的功能（如果你读了本书，你会感觉本书中同样实现了类似的想法）。与此同时，我仍然在纠结那个队列死锁问题。所以来我与 Asterisk 社区中的另一个成员开始计划一个新的队列模块——`mod_icd`。

ICD 的代表智能呼叫分配（Intelligent Call Distribution），与首字母缩写的自动电话分配（ACD，Automatic Call Distribution）相对。我们找出了 `app_queue` 模块所有的问题，我们对做一些新的稳定的，再也不会导致无休止的死锁和崩溃的模块同样感兴趣。我们使用状机以及更高级的基于内存池的内存管理抽象以及其它一些在标准的 Asterisk 中不存在的创造性地概念。但问题是，我觉得我们在那个模块上做过了头，看起来几乎是我们把 Asterisk 核心的一些功能也边缘化了。当然，那只是其中的一个可载模块，完全边缘化 Asterisk 的核心是不可能的。

我们始终没有完全完成 `mod_icd`。在 2004 年底，我参与呼叫中心解决方案的机会被那些不可饶恕的段错误和死锁的深海击碎、涤荡殆尽。我们开始关注另外的与队列无关的电话服务。我使用了我添加到主流的 Asterisk 中的几个新特性以及我的几个未被批准的不太流行的小模块开发了一个新的被叫付费业务（类似中国的 800 电话）以及传真转电子邮件服务。我建立了一个由 7 台 Asterisk 主机组成了集群，将它们与电信部分提供的线路对接。这种部署 Asterisk 的方式并不是不会出问题，而比较美好的一点是，如果某台机器崩溃，就会有另一台顶上去，然后我们就有机会重启那台崩溃的机器。

新点子和新项目

⁸即 Segmentation Falult，是程序运行期间由于共享的内存遭到破坏而引起的程序崩溃。——译者注。

在这一点上我积累了一些新想法——有的测试过、有的没有，还有一些需要对 Asterisk 做一些大的改动。我跟我的队友 Brain West 以及 Michael Jerris 在 Asterisk 项目上贡献了很多时间。我们帮助管理缺陷跟踪系统（Issue Tracker），我们修复了很多 Bug，并且我们每周都主办开发者的电话会议，甚至我们还在我们的服务器上做了一个代码库镜像站点。我们参与的太多以至于我们的一些新想法在 Asterisk 社区中引起了一些政治骚乱，起因在于一场不同的开发者之间的一场没必要的竞争——每一个 Asterisk 的贡献者必须签署一个表格以声明他们写的所有日后可能用于 Asterisk 的代码都自动对 Digium 公司（Asterisk 的拥有者）有一个免版税的授权，以便他们可以用你的代码做任何他们喜欢的事。如，通过这种方式，他们可以将这种无限的许可证以高的多的价格卖给他们的潜在客户。这完全背离了开源精神，但这就是另一个故事了。我认为这种差异化引起了一些跟我一样的志愿开发者与 Digium 雇佣的一些 Asterisk 开发者之间的一些冲突。

即使在这么紧张的环境下，我们还是全力以赴地支持该项目真正希望它能成功。我们继续好召开每周的电话会议，他们也确实采取一些措施开始帮助开发者们增加动力。我们觉得我们应该有一场现场的见面会，以便我们所有人都可以聚到一起分享我们的电话技术知识，并一起玩几天。我们不知道我们要干什么，但我们决定要做，并把该聚会称为 ClueCon。有一个 Clue⁹意味着你知道你要做什么，所以 ClueCon 的意思就是帮每一个人找到一个『Clue』。我承认，即我刚刚说过我们也不知道我们要做什么。看起来非常有意思，一群没有 Clue 的人开了一个有 Clue 的会——ClueCon。不过，事实证明非常幸运，这好像根本不是个问题。前面我们所指的 Clue 都是指电话技术，而不是做会议。

因此，在第一届 ClueCon 之前的几个月中，即 2005 年春天，我们在一次例行的电话会议中开始专门深入讨论 Asterisk 中的几个缺点。这非常正常，因为我们主要的目标就是找出这些问题并找到解决方案。在那个时期，有一大群不守规矩的、受够了他们在 Asterisk 上遇到的无穷无尽的问题的人。那群人中的很多人都参加了那次周会，希望能说服我们帮忙看一看他们遇到的问题。我想得越多，越觉得解开折磨我们的核心架构问题越是任重道远。Asterisk 中的很多核心都具有单一的性质而无法扩展（Scale），其中的很多我发都有很多用户依赖于它们，任何企图改进他们的动作都有可能会引发功能上的退化。有些问题看起来是无解的，除非用一把大锤把那些旧代码砸个稀巴烂，并从核心的代码深入重写。但这种方法看似不可行，因为它将会使得 Asterisk 在几个月内甚至一年或更长的时间内都不可用。也就在那时候，我有了一个想法——我们做一个 2.0 版吧。

从一个 2.0 版并不是一个最坏的想法。我知道它将有很多挑战，但是，我想我们可以与旧的代码并行启动一个新的代码库，那样我们就可以删除那些有问题引起问题最多的部分旧代码并替换成新的，并仍然维护用户依赖的一些仍然可用的代码。有了这个想法后我感到很兴奋，同时也在我提出该想法后看到该项目的领导人的反映时得到了同样地震惊。他似乎也对我竟然提出这么一个想法同样震惊。总之，简单来讲，我们没有做 2.0 版。那时，我有无数的想法，我也非常清楚地知道我喜欢以及不喜欢 Asterisk 的地方，但没地方写。

我盯着那个在一个空目录中打开的空的编辑器缓冲区¹⁰，盯了足足一个小时。我知道我想要做什么，但不知道怎么写出来。在我在编辑顺中增加了一些奇怪的单词以及一些标点符号之后，我才知道该如何开始。那些单词并不是你常用的单词，还是一些符号以及变量声明——我是在写 C 语言代码。

⁹线索，后面的 Con 指会议（Conference）。——译者注。

¹⁰用于编写程序代码的编辑器。作者使用 Emacs 编辑器，在 Emacs 中，每一个文件（甚至 Shell 等）都称为一个缓冲区（Buffer）。——译者注。

几天后，我用 C 写了一个基本的程序，试验了几个我在过去编程中喜欢用的一些编程工具。我有 Apache 可移植性运行库（或称 APR），有 Perl 编程语言，以及一些其它的程序包。我建立了一个核心，以及一个可加载模块的结构，一些助手函数用于内存池管理，并且我有了一个简单的命令提示符，你可以在命令行上键入 `help`，如果你愿意看到命令行上显示一个尖刻的提示，提示你根据没有帮助信息的话，或者也可以键入 `exit` 以终止程序。我还写了一个示例模块，允许你使用 telnet 连接到一个特定的 TCP 端口上，你输入的任何字符都将原来的回显回来。另外我还实现了一个简单状态机。我把这个程序叫做 Choir。因为我希望我的一系统的想法都可以像教堂里的唱诗班一样发出和谐的声音。在那些最初的代码之后，我放了一段时间。因为 ClueCon 快要来了，我还有许多东西要准备，而没有想到时候太仓促。

第一届 ClueCon

2005 年 8 月的 ClueCon 是第一届。当时参加的有几个 VoIP 项目的领军人物，包括 OpenH323 的作者之一 Craig Southeren 以及 Asterisk 的创建者 Mark Spencer，当然，不喜欢我的 Asterisk 2.0 的想法的也是他（Mark），但无论如何，将这些人聚到同一间屋子里还是一件非常酷的事情。我们整天都有演示，以及反复地交流，并且，我们真正使得每人都开始想问题。那一届 ClueCon 非常成功，会议圆满结束以后我动力十足，并准备好继续写我的 Choir 代码。但事实是，我并没有立即开始行动，而是在我们的电话会议上讨论了几个月，同时挣扎在时时刻刻都有倾覆危险的 Asterisk 平台上。秋天马上就到了，Asterisk 社区的混乱最终导致了一场苦命——社区中占相当比例的人 Fork¹¹ 了 Asterisk，新的项目叫做 OpenPBX。

我完全理解他们为什么那么做，并尽我所能地支持他们。我贡献了我所有为 Asterisk 所写的代码，他们可以根据自己的喜好随时取用。如果有闲暇时间，我也会帮助他们，但我最终还是没有完全融入他们。因为我仍然有同样的问题没有解决——我认为有些问题必须从最底层解决，而这一新项目（指 OpenPBX）的创始人更倾向于解决那些 Asterisk 团队没有能够及时解决的现实的问题。我们仍然开电话会议，但大多数情况下 Asterisk 项目的人都不再参加了，因为我们为 Asterisk 社区的革命欢呼使用他们不高兴。由于在不将 Asterisk 核心完全推倒重来的情况下我无法修复任何问题，有一天我为此事道了歉。也就在那时候有人（如 Tootsie Pop commercials 的 Owl 先生）问我：『你觉得让你的新代码能打电话需要多长时间呢？』我不知道，所以我决定试一下——不管是一周、两周还是三周。

我写的第一个能够发出声音的模块是 `mod_woomera`，该模块是一个 Endpoint 模块，它使用了 Craig Southeren（与我在 ClueCon 上遇见的是同一个人）写的 Woomera 协议。我也为 Asterisk 写了一个类似的模块。Woomera 协议非常简单，它并不需要编解码或其它复杂的东西。它的实现思想是它屏蔽了 H323 协议的复杂性并允许应用程序使用该简单的协议与它通信以便更容易地集成到 VoIP 程序中。所以，从它开始好像是一个正确的选择。当我开始工作的时候，我意识到在我的核心代码中需要更多的元素，然后我就慢慢的添加，并最终将这些代码融合到一处，我终于可以给以 Woomera 协议武装的 H323 监听进程打电话了，同时，我也可以在我的 Pandora 代码里获取通话的状态。是的，我把我的项目名称改成了『Pandora』，因为大家都不喜欢 Choir 这个名字。我非常愉快的听着 Alan Parsons Project hit Sirius stream 第一次从我的扬声器里流出来。这一次比我第一次使用 Asterisk 打电话时更加兴奋，因为我白手起家从头开始写的代码现在开始工作了。

¹¹专业术语，指在原来代码库的基础上重新建了一个分支，然后两个分支分别独立发展。

现在，我已经有所进展了。我研究出了如何让两个 Channel 桥接到一起、如何支持更多的其它协议以及做一些其它的基础的事情，而不是仅仅打印一条尖刻的帮助信息并退出。有人建议把这些代码叫做 OpenPBX 2，有人也建议其它名字。在经历了足够的命名争论后，我知道了（并永远决定了）我应该叫它什么：FreeSWITCH。我终于有了一个我将为之坚持不懈的名字、一些可以工作的代码，以及很多野心。我埋下头继续工作。所有地方都有工作要做，多得甚至你都没时间去想。所以我就不停地写代码。时间很快到了 2006 年 1 月，那时我有足够的代码可以与公众分享了。我们向开发者们开放了我们的代码库。通过让他们注册一个开发者账号才能获取代码的访问权限，我们确保只有非常严肃的开发者才愿意完成整个注册过程。有一些人下载了¹²源代码并提供了一些反馈，我们当时真觉得我们有了一个真正的项目。

我们有了一个可以桥接电话的模块、一个可以放音的模块、一些编解码模块、一些作为例子的拨号计划（Dialplan）模块，以及一些其它的模块。哦，我有没有说过它在 Windows 上也可以运行¹³？

虽然页面上所有的链接都失效了，但我们原来的站点仍然保留着：http://www.freeswitch.org/old_index.html。

FreeSWITCH 诞生

在实现我们计划的过程中，我们确实写出了可以运行于 Windows、Linux 以及 Mac OSX 上的代码。我早期团队的伙伴——Michael 和 Brian，从一开始就跟在我一起。Mike¹⁴在 Windows 平台上很有经验，他确保了我们的代码能在 MSVC 里编译和运行。最初这确实是一个痛苦的过程，但在修复了无数情况下无数的编译错误之后，我第一次开始学习如何编写跨平台的代码。光阴似箭，下一次 ClueCon 的时间到了。在那一年，我进行了我的第一次 FreeSWITCH 演讲，演示了在本书开篇¹⁵中所描述的核心设计和基础架构。我们看到了非常令人兴奋的模块，如一个可以与 Google Talk 通信的模块。在演讲过程中，我也通过 mod_exosip 模块现场演示了在有几千个并发呼叫的情况下呼叫的实时建立和释放。那是一个很好的演示，但我们并不满足。

Exosip 仅仅是在 Osip 基础上进行了一些上层的封装，而原来的 Osip 库则是一个开源的 SIP 协议栈，它提供了大多数的 SIP 功能。Exosip 使得开发一个 Endpoint 模块更简单一些，所以，我们决定基础它来开发我们的 SIP 模块。但后来，我们还是遇到了几个灾难性的问题，使得我开始感觉我们陷入了与当时让 Asterisk 正常工作一样的境地。因而，我们开始寻找一个 Exosip 的替代者。我们寻找替代者的另一个原因是 Exosip 选择了 GPL 许可证，而不顾原始的 Osip 库本来是 LGPL（就我个人感觉，应该继续选择 LGPL 更合理一些），这就导致了潜在的许可证冲突。由于我们在我们的项目中使用的是 MPL 许可，而 GPL 协议不允许使用 GPL 许可的代码嵌入 MPL 许可的程序中。有关许可证的争论很有趣，也经常能令人兴奋，但当时我们没有时间参与这些。

由于在 FreeSWITCH 中对 SIP 功能有很高的要求，我们上天入地，找遍了开源界的每一寸土地，希望能找到一个可以用的新的 SIP 协议栈以及 RTP 协议栈。我们针对这两点评估了几个库，最终几乎每种协议都试用了至少 5 个库，但还是没有找到一个令我满意的 RTP 协议栈，所以最终我决定自己实现。当然，我并不至于傻到也自己去实现 SIP。在我看到 Asterisk 曾试图从头到尾写一个 SIP 协

¹²原文是 Check out，即从 SVN 仓库检出代码。

¹³FreeSWITCH 可以在 Windows 上原生的运行，而 Asterisk 不能，或者只能通过 Cygwin 等模拟环境运行。

¹⁴Michael 的简称。——译者注。

¹⁵该书第一章是 Architecture of FreeSWITCH，即 FreeSWITCH 的架构与设计。——译者注。

议栈并最终失败而转投 Exosip 之后，我继续在开源领域中寻找 SIP 协议栈，直到最后发现了诺基亚 (Nokia) 开发的 Sofia-SIP。我们写了一个可用的 mod_sofia 模块来进行测试，效果非常不错。我们继续打磨该模块直到它可以完全替代 mod_exosip，然后 mod_sofia 就成了我们系统中首要的 SIP 模块。不过，那仅仅是开始。因为到后来，即使是现在我还经常要往 mod_sofia 中添加代码。SIP 是一个非常复杂并且令信恐惧的协议，它带来了许多令人不愉快的思想，而不像它的名字看起来那样简洁。但现在不是讨论这个的时候。

我们在 ClueCon 2007 上再一次演示了 FreeSWITCH，那一次，有了一个新的 SIP 模块以及更多的代码。另外，还有 OpenZAP，它使用一个 TDM 库将 FreeSWITCH 连接到电话硬件上。OpenZAP 后来被 FreeTDM 替代，现在由 Sangoma 公司负责维护。我经历了使用同样的板卡，很久以前让它在 Asterisk 上工作，现在又让它在 FreeSWITCH 上工作的愉快过程。我们曾经很快地宣布我们将推出 FreeSWITCH 1.0.0 版。很多看过我们原来的主页的人可能会注意到当时我宣布了一个官方的新版本将『很快发布 (oming soon)』，条消息的发布时间是 2006 年 1 月，当时我们拼命的想让所有事情按我们希望的那种方式工作。我们非常希望专注于在添加任何其它功能前做一个稳定的核心，并且我们也有了很大的进展，但是我们还是没有准备好发布 1.0 版。

2008 年春天我们有了稳定的 SIP、有了 Event Socket 用于远程控制 FreeSWITCH、有了一个模块可以通过 HTTP 执行 FreeSWITCH 命令、有了 XML curl 等等一系列的新功能和特性。我们最终觉得可以发布一个版本了。所以我们就一举发布了 FreeSWITCH 1.0 凤凰版 (Phoenix)。我之所以将该版本取名为凤凰，是因为感觉到我们所有的辛苦的工作成果都是从先前的失败的骨灰中来的，并且这个名字也被很多其它人使用，包括 NASA 在同一时刻将『凤凰号』送上了火星。总之我认为那是一个很合适的名字。

在 ClueCon 2008 上，我们又一次宣布了曾于当年 5 月份发布的 1.0 版。当时还有另外一些与 FreeSWITCH 有关的演讲，以及与 Asterisk 有关的演讲，因为当前也发布了 Asterisk 1.6。在当前接下来的时间里，我们用了所有的时间专注于宽带（高清）语音的支持以及其它的一些功能，例如即时的进行采样率转换。此外，我们还增加了一些新的 SIP 功能，如状态呈现 (SIP Presence) 以及其它的一些简单通话以外的功能，并于后来发布了 1.0.1 版。

2009 年，我们发布了 1.0.2 至 1.0.4 并于第 5 届 ClueCon 年度会议上又一次演示了 FreeSWITCH。到那时候，我们早期的一些创新变成了现实。因为我们可以演示使用 Polycom 话机新的 Siren 编码进行高清语音的通话，并且我们还支持了与 Skype 互通。当年的 FreeSWITCH 演示概括了一些你可能根本就意识不到你可以做的事情，除非你具有四维空间的想象力。而这，正是 Emmett Brown 博士（来自电影《回到未来 (Back To The Future)》）都想做的。FreeSWITCH 有一些与 Asterisk 类似的行为，但是我们同时也有一个新的词汇表，该词汇表新像为你打开一个通向无限可能的空间的大门，使你可以通过一个 PC 和一个电话就可以做任何无法想象的事情。

2012 年的 ClueCon 是在 Trump Tower¹⁶举行的。它是一届有史以来最值得怀念的 ClueCon。当年我们出版了本书的第一版，我们还发了几本做为奖品。我们详细演示了 FreeSWITCH 以及它的性能。当年我们发布了 1.0.5 和 1.0.6。那年的主题好像是 Erlang，好像每个人都赶上的事件驱动 (Event Driven) 架构的时尚。所以，我们绝对是在命令的时候处在了合适的位置。

¹⁶一个国际性酒店。——译者注。

2011 年是该项目大跨跃的一年。受我们一年前自己关于性能的演讲的启发，我们对 Sofia SIP 模块进行了大刀阔斧地修改，将原来串行化的消息处理改为并行的处理，每个路电话的消息都会被推到它自己的线程中处理。这次改变产生了很多并行的操作，避免了由于单一的 Channel 发生问题时阻塞整个 SIP 协议栈的可能。那年的 ClueCon 是在壮丽的 Sofitel 酒店举行的。我们演示了很多新特性，包括一些用于帮助开发者进行开发的特性，如变量数组的概念以及范围变量——你可以使用它来设置一个通道变量，该变量仅在某一特定 App 执行的时候才有效。

2012 年我们宣布了一个新的计划，在 FreeSWITCH 代码库中支持稳定版的分支（stable branch）。这是一个令人望而却步的任务，因为你必须将成熟的代码与新的代码分离，并且在每次修改时都需要在不同的分支上做额外的检查以确保所有东西都平稳运行。我们为此非常努力地工作。并且本次新版的书将包含 FreeSWITCH 1.2 稳定版中最初版本的内容。我们在 Hyatt 酒店举行了一次很成功的 ClueCon，并演示了一些新的特性如支持 Hylafax 的软件模拟器以及一个新的 mod_httapi 模块，该模块也在本书前面的章节中讲到了。

在写本书的时候，已是临近 2013 年了。在活过了玛雅人的预言¹⁷之后，我们开始研究消除传统的电话与 HTML5 以及 WebRTC 之间的间隙以及第一个 FreeSWITCH 1.4 Alpha 版本。ClueCon 将继续在 Hyatt 酒店举行¹⁸，他们为给我们打造更温馨的环境重新进行了装修。我们希望在那儿见到你们所有人，并希望你通过本书多学到了一丁点儿的 FreeSWITCH 的知识以及了解为我为什么决定在那个空白的文本编辑器上开始打上那几个字符¹⁹并最终变成 FreeSWITCH 核心组件的近 50 万行代码的。

11.6 我与 FreeSWITCH 的故事

杜金房 / 2015.12.10

故事还得从几年前说起。

很久以前，我在电信（后来叫网通）做过程控交换机的维护，机型主要是上海贝尔的 S1240 和华为的 CC08，也了解西门子 EWS、朗讯 5ESS 等机型以及各种接入网设备等。由于喜欢写些小程序，所以在工作中也写过一些与交换机交互的程序，但大都是一些执行命令创建和删除用户，收集和分析日志等的操作，从来没有涉及呼叫控制方面的。

有一年去北京，一次偶然的机会认识了 Jonathan Palley。他正在做英语在线教学的项目，从他那里我了解到了 Asterisk。当时，Asterisk 都已经七八岁的样子了，我却是第一次听说，可见处于体制内的我是多么的坐井观天。

在运营商工作的好处就是能接触到各种程控交换机、大型的服务器以及各种 UNIX 和各种数据库，虽然我的工作主要是维护，但我还是利用工作和业余时间开发了许多程序，开发语言主要是 C 和 PHP，把跟我工作相关的交换机和 OA 等系统全部融合起来。我的很多理念和想法以及具体实现都是

¹⁷传说玛雅人的历法中预言 2012 年世界将灭绝。——译者注。

¹⁸本文写于 ClueCon 2013 之前，后来，ClueCon 2013 如期在 Hyatt 举行，当年的主题好像是 WebRTC。译者曾在现场。
——译者注

¹⁹指作者最初写 FreeSWITCH 代码的时候。——译者注。

比较超前的，系统也是很好用的。但是，在支撑系统全市统一、全省统一、全国统一的大环境下，我写的东西很显然推广不开。而且，后来，公司内部越来越轻运维，重市场，搞起了全员营销。说白了就是你必须发展多少小灵通用户，办多少宽带等。我越来越不喜欢那种工作状态，就准备去北京投奔 Jonathan Palley。

在去北京之前，我研究了一阵 Asterisk，在淘宝上买了一个单口的语音卡，接上我的办公电话，有人打进来就先放 IVR，然后通过逃生口接上话机，我再跟他们通话，感觉很神奇。我甚至配置了路由，可以在家里用软电话通过网络利用办公室的电话线往外打『免费』的电话。

后来 Jonathan 说 Asterisk 有问题，他准备换 FreeSWITCH，问我有什么意见。我对 FreeSWITCH 其实没有什么研究，便赶紧上 FreeSWITCH 的官方网站看了看，下载、编译都很顺利，简单测试了一下，有些功能用起来竟然比 Asterisk 还方便。我便跟 Jonathan 说：值得换。那时候，FreeSWITCH 还没有出 1.0 版。

后来到了北京，我的职位是系统管理员，主要是搞定教学平台的各种安装和部署。我很欣慰，像我这种自学成才的 Linux 菜鸟在整个团队中却成了 Linux 专家。而且，更让我喜欢的是，我可以完全使用 Linux 桌面工作了，以前的各种 Windows 上专用的软件都见鬼去了（后来有了钱，电脑换成了 Mac）。在工作中，我学会了 Ruby、Ruby on Rails、SVN 以及后来的 Erlang 等技术。我很勤奋，分内的分外的事情我都做，以至于后来整个系统从最底层到最上层我都懂。

FreeSWITCH 在我们的系统里是一个重要的部分，在使用和维护过程中，我也会发现一些系统的 Bug，并渐渐跟 FreeSWITCH 社区有了交流。

其实，FreeSWITCH 最初相关的开发都是由 Jonathan 负责的，公司也一直想招一个专门做 VoIP 开发的人员，但就是一直没招到。后来 Jonathan 公务越来越忙，我又什么都会，FreeSWITCH 相关的开发任务也落在了我身上。当然，我后来也忙不过来，就又招了一名系统管理员给我打下手。这样，我的工作重点就转到了研发，以及系统架构。

外企的工作环境还是很宽松的。虽然工作不少，但我勤奋啊，在北京只有我一个人，因此，除了公司以及团队自发组织的各种业余活动，不管是晚上还是周末，我都抱着电脑。当然，有一件比较悲催的事情是，我跟公司几个客服（其它他们销售和客服什么都干）人员住在一起，因此，系统一有问题，便总能找到我。所以其实我大部分的业余时间是为公司奉献了。当他们不找我的时候，我就自己优化系统。

我也跟 Jonathan 开玩笑说，我其实大部分时间都是在修他写的 Bug。

Jonathan 是个天才，他学东西非常快，想做的东西也很快就实现了。但如上面所说，他没有时间做精细的调试和打磨，因此，我正好是一个很好的补充。所以，我最初的 FreeSWITCH 技术全都是跟他学的，包括混进 FreeSWITCH 社区。

我们是属于使用 FreeSWITCH 比较早的一批用户。在我们基于 FreeSWITCH 的系统上线后的几天内，FreeSWITCH 发布了 1.0.0 凤凰版。

当然，在使用开源软件的同时我们也有贡献。最初主要是报 Bug 和写 Wiki，后来，就开始提交补丁了。那时候，FreeSWITCH 的开发人员都很专注，因此，Bug 修得很快，基本上都是我们今天

报上去，第二天上班一看就修好了（美国人在我们睡觉的时候工作）。有一阵，我接连不断的提交新特性的建议和补丁，在打完最后一个补丁后 Anthony 跟我说，我可以休息一下了吗？我说，可以了。

我基于一个 G729 库写了 FreeSWITCH 中最初的 G729 转码模块，后来网上流传的很多版本都是基于我的版本修改的（头文件里还有我的名字呢）。不过，写这个模块其实是一个错误。我后来才了解到，原来 FreeSWITCH 内部早就实现了 G729 的转码，只是，由于专利原因，无法对外发布。对此专利细节我不了解，但是，据我所知，在世界上绝大多数国家，即使你自己实现了 G729 的编码，也要交专利费。由于我的错误，FreeSWITCH 关闭了 Wiki 上 G729 页面的修改权限，也禁止在邮件列表里讨论 G729 转码。后来，FreeSWITCH 官方在 Linux 上实现了 G729 的 Licence 机制，10 美元买一个编码器和一个解码器，FreeSWITCH 最终才有了 G729 的解决方案。当然，那都是后话了。

当然，我们（外企）是很重视专利和版权问题的。所以，后来，我写的模块仅用于试验，但没用于生产。用于生产的，是我写的另外一个模块，叫 mod_recpld，是 Record Payload 的缩写。实现想法是，不经过转码，而把 G729 的内容直接录下来，后面，再用其它程序将 G729 转成 mp3 文件。而我们有这个『其它程序』的 G729 许可证。其实这种录音方法后来在 FreeSWITCH 核心中也实现了。另外，我们后来也发现 G729 编码在互联网上的声音质量不如 iLBC，因此，在我们决定不再使用一个仅提供 G729 编码的 PSTN 落地提供商的线路后，我们也不再使用 G729 了。

通过写上面两个模块，我深入了解到了 FreeSWITCH 内部的机理。以后再改代码，就容易得多了。

除了在线教育平台外，我们的销售和客服团队也逐渐壮大，因此，需要一个呼叫中心系统了。最初的呼叫中心是用 Trixbox 搭的，Trixbox 是一个集成 CentOS 和 Asterisk 的一个发行套件，使用起来很简单。我们使用很老的只有 700 多 M 内存的奔腾机器跑这个应用，竟也能带十来个座席。当时，使用的是『淘』来的 Asterisk 兼容卡连接模拟外线。我们甚至还用了 OSLEC 做回声消除。后来，随着需求越来越多，我们将它换成了 FreeSWITCH，我们自己实现了 CRM，跟我们的系统紧密集成。多紧密呢？我们用 Erlang 实现了 IVR 功能，当有电话打进来时，它便播放欢迎词，同时，根据主叫号码，异步到美国的服务器上去查询这个是谁的客户，进而分配座席和弹屏。去美国的服务器查询比较费时，但是我们用 Erlang 很优雅地通过异步操作实现了该功能。

另外，我们也购买了一个 Global IP Sound 的库，实现了自己的 SIP 客户端。没想到，后来 Global IP Sound 变成了 Global IP Solutions，后来又卖给了 Google，Google 又把那些代码开源了。

再回到我们的教学平台上。由于我们在美国的老师都是 Work-At-Home（在家工作）类型的，他们分布在全国各地，因此，实际上我们的教学平台就是一个分布式的呼叫中心系统。有一套很复杂的算法，根据老师上课的数量和质量给他们发钱。我们的平台用到了 Ruby on Rails，Erlang，Lua 和 FreeSWITCH，每个技术都有它们最擅长的地方。

同时，我们也在不断的招聘老师。我们做了一个自动面试程序，面试者在 Web 页面上注册，根据提示，输入自己的电话号码，我们呼叫他们的电话，阅读我们指定的文字并录音。这一切都是自动的，电话跟页面也是同步的。通过几轮自动面试之后，我们才会有相关的人员人工联系他们进行更深入的面试。这些，也都是用 FreeSWITCH 做的。

当然，这么多『分布式』的老师要开会啊，因而，我们又用 FreeSWITCH 实现了会议系统。

还有一个有意思的事情。当时，由于国内的网络条件和对 VoIP 的限制，我们在国内是直接使用 PSTN 电话跟学生连线的。有些学生反应他们的手机处于漫游状态（如到北京上新东方培训班），接电话很贵。因此我们又实现了 FreeSWITCH 跟 Skype 和 Google Talk 的互通。这样，我们就可以在 FreeSWITCH 中打电话到学生的 Skype 或 Google Talk 客户端上。

当然，事情也不是永远一帆风顺，最初跟 Skype 和 GTalk 互通，FreeSWITCH 总是崩溃，我们报了一些 Bug，打了一些补丁才解决。这也决定了我们的基础架构——我们专门起了几个 FreeSWITCH 进程，用于跟 Skype 或 GTalk 连，即使其中一个崩溃了，我们也可以迅速切换到另一个（为此，我们也注册了很多 Skype 账号：））。

总之，正是由于不断的『折腾』，我们的 FreeSWITCH 技术才不断进步。

花开两朵，各表一枝。再倒回去说说 FreeSWITCH 社区的事情。

一来二去，我在 FreeSWITCH 社区里混得熟了，便萌生了一个做 FreeSWITCH 中文社区的想法。我对开源很热爱的，我曾在 Linux Focus 上翻译文章，翻译 SQLite 的文档等。中国人也都很厉害，几乎任何东西都有了中文版，有了中文社区，但当时，FreeSWITCH 还没有。

我很快注册了域名，FreeSWITCH-CN 开张了。

在任何圈子里，总会找到志同道合的人，我发现了大熊。我发现他几乎在同一时间建立了 FreeSWITCH QQ 群，我甚至不记得我是怎么发现他的了。大熊人非常好，把我升级成了 QQ 群的管理员。他从来都很低调，不知道的还以为我是 QQ 群的群主呢。

对于 QQ，我以前不大用。最早注册的 QQ 号都忘记了，后来工作后一个同事让给我一个 QQ 号我使用至今。我前些年折腾 Linux，而 Linux 上的 QQ 总是各种不好用。后来在外企大家都用 MSN 和 Skype，而我的工作平台也由 Linux 换成了 Mac。离开外企之后，没有人用 MSN 了，而 Mac 版的 QQ 也开始争气了（比 Windows 上老是弹出垃圾广告好多了），所以我 QQ 就用得多了。

我很快取得了 FreeSWITCH SVN Contrib 代码库的提交权限。Contrib 就是 FreeSWITCH 外围的一些代码库，FreeSWITCH 扇丝们可以把自己实现的一些好玩的东西提交上去，全世界共享。我提交了一些我写的代码，后来，发现我在代码中『发明』的方法被大家广泛使用。甚至，有一次，在 ClueCon 上，一个保加利亚的哥们用非常不流利的英文跟我说那几个脚本真是太好用了，帮了他大忙。

再后来，由于我在 FreeSWITCH 社区表现突出，提交的补丁也得到了社区的认可，就取得了 FreeSWITCH 核心代码的提交权限。而这时，FreeSWITCH 的代码库也早已从 SVN 迁移到了 Git。

我最初建立社区的想法很简单，中文网站只是一个网站，用 Google Groups 做邮件列表，进行讨论，用 QQ 群做即时的讨论（我甚至也想过 IRC）。关于邮件列表，英文社区里也有很多讨论，到底邮件列表好还是一个 BBS 好？社区的管理人员认为邮件列表好，因为你可以用任何你喜欢的邮件客户端参与讨论。而有不服气的人去建了 BBS，结果就是没有人气。我受他们的影响很大，因此，从来没有建 BBS 的想法。但很遗憾，Google Groups 很快就无法访问了。我好像是受了诅咒一样，我注册了 Twitter，Twitter 不好用了，我注册了 Facebook，用了几天不好用了，我用 DropBox，隔几天就连不上了，我建立了 FreeSWITCH-CN Google Groups 中文讨论区，也无法访问了。到今天，我偶尔还会在 Google Groups 里发邮件，但是，无论如何都没大有人气。

而关于建立 BBS，在国内也是很困难的。你必须备案，必须有人监管，必须……。总之，有很多人提出我们该有个 BBS，但我总是没有精力去做这件事，我也说过有人愿意捐赠服务器并搞定一切事情的话，我们也可以建立个 BBS，但到今天还是没有结果²⁰。

我想，也许，大家有问题的话还是到专业的社区上去讨论吧，因此，我开了知乎专栏 (zhuanlan.zhihu.com/freeswitch)，并鼓励大家到那里去提问 FreeSWITCH 有关的问题。

QQ 群里的成员倒不少，目前也是最主要的交流平台了，群里风气很好，也没大有广告，我很高兴看到大家都在讨论技术。

最初，我在群里回答问题是很积极的，但是大家好像不怎么配合，总问些没法回答的问题。我便写了一系列的文章告诉大家应该怎么把问题描述清楚，也就是如何问到点子上，如何收集日志并贴到 Pastebin 上。后来，的确起到了一些效果，有些人已经学会了如何先把日志贴到 Pastebin 上再提问题了。

不可避免的，你需要一遍一遍的回答各种问题。因此，我想写一本书，给大家讲一些基础的知识。我的想法就是，再有人提问时，告诉他看书的第几页。但写一本书谈何容易，因此，最初我把写的东西以博客的形式发布出来，供大家免费阅读和挑刺。但我很讨厌把我的文章转载到自己的博客里去掉作者姓名当自己原创的人，也很讨厌那些把文档制成 Word 版上传到各种文档分享网站换积分的人。为此，我还专门跟百度干了一架。但得到的后果是在百度上搜 FreeSWITCH，再也到不了我们的网站了。

因此，后来，我的写作就不积极了，有时候写了也不愿意发。两年前，在我们国内 FreeSWITCH-CN 第一届沙龙前夕，我把书印了出来，大家竟然非常喜欢。也有读者告诉我，越看越爱看，每看都有收获，甚至在草原上放羊的时候也看。

但无论如何，小册子总有『非法出版物』之嫌。因此，能够出版，一直是我一个夙愿。值得欣慰的是，经过四年多的积累，在华章公司的支持下，这本书《FreeSWITCH 权威指南》终于跟读者见面了。这是我的第一本书，也是国内第一本讲 FreeSWITCH 的书，但就在第一本书上出现了『权威』二字，我还是有点心虚的。我确实想把我所知道的都写进书里，但是，即使每个主题都没有深入进去，这本书也已经很厚了。无论取舍是否得当，总之，这是我真心想送给广大 FreeSWITCH 爱好者的一个礼物。同时，也欢迎各种批评。

2011 年，我第一次参加在美国芝加哥举办的 ClueCon，我不想错过任何机会，便找了个空子上台讲了几句。第一次用不流利的英文在 200 多个纯老外面前演讲，真的很紧张，不过，坚持下来了，反应还不错。

2012 年，我就有备上台了，英文也稍微好了一点点。我用了几个性感美女图片轰动全场。

2013 年，FreeSWITCH 三剑客 (Anthony、Brian、Mike) 都在下面认真听我的演讲（以前他们基本都在外面跟别人聊天），让我在台上小激动了一把。

当年，还有一个好玩的事情。一个加拿大的哥们跟我说，我写得东西真好，他有问题了就到我的网站上找答案，看不懂中文就用 Google Translate 翻译。还把他做的系统演示给我看（他确实做得很好）。

²⁰后来，我们真的建立了一个 BBS，地址是：<http://bbs.freeswitch.org.cn>。

2014 年的 ClueCon 马上就要到了，我还会去。

2012 年开始，我们 FreeSWITCH-CN 中文社区也学着 ClueCon 的样子在国内搞活动，为避免版权问题，我们不叫 ClueCon，叫 FreeSWITCH 开发者沙龙。随着大家对 FreeSWITCH 的关注的提高，我们的活动竟也有模有样。

2013 年开始，我开始做公开的 FreeSWITCH 培训班，每批都有二、三十名学员，感觉也还不错。

花开两朵，再表一枝。大约就在我第一次参加 ClueCon 的时候，我们原先的公司因为各种原因，部分解散了，其中也包括我。

在老的没人要的时候，只能去创业了。

创业的过程是艰苦的，我们也学到了很多东西。在这些年里，我潜心研究 FreeSWITCH，实现了 MSRP、实现了类似 IMS 的架构、实现了 RTMP 基于 Flash 的视频（有大熊的支持）、实现了 RTSP、VLC 视频、实现了视频转发、视频转码、视频会议、视频录像、做了 N 个版本的指挥调度系统。这些系统有的是实验性地，有的还不很完善，也大部分没有开源，不过，我一直惦记着开源这回事呢。

关于做开源与赚钱的问题，我也一直在思考。但我想，我会一直做 FreeSWITCH 的培训与咨询，支持大家在 FreeSWITCH 上面做呼叫中心、做运营、做指挥调度等各种应用，而不是去做上层五花八门的应用，应该算是一种思路吧？

与诸君共勉。

11.7 The missing Link

本文主要讨论应用程序如果依赖其它库的话，应该静态链接还是动态链接。在这一点上，我们看看 FreeSWITCH 是怎么做的。

曾经，在 2007 年有个叫 Roman Shaposhnik 的哥们发了一篇博客文章—『[What does dynamic linking and communism have got in common?](#)』，上面讲到，在一个理想的世界中，所有的库都应该是动态链接的，这样可以最大限度的节约磁盘空间和内存，程序也更有效率，少出 Bug。

但是，FreeSWITCH 的作者 Anthony Minessale 却有不同的话说。该文章来自<http://www.freeswitch.org/node/56>，原文翻译如下：

最近，我读了 Roman Shaposhnik 的博客文章，是关于静态连接的：https://blogs.oracle.com/rvs/entry/what_does_dynamic_linking_and。

作为 FreeSWITCH 阵营的代表，我想，我应该说点什么。

对于整个问题，我可以提供一个完全不同的视角。我们的项目（FreeSWITCH）我很多我们自己的代码，同时也有一系列的依赖库，这些依赖库大多数在一些外部模块中使用。现在，Roman 在它的文章中说，在一个理想的世界里，应该只有一个操作系统，并且只有一个唯一实体管理着相同的环境。为了能在你的系统上运行我们的软件，我只需要向该实体请求我需要的环境，包括我们的软件依赖的各种程序库。

我们遇到两个主要问题。一是我们的目标是跨平台的，而『跨平台』在我们心目中的含义是它能尽可能地运行在任何操作系统以及任何硬件上。实际上，这本身就是个恶梦。与 Roman 所描述的理想世界相反，由于这个世界上有 Windows、Mac OS X，数十种 Linux 发行版、以及 Solaris 等各种不同的操作系统，它们又各有不同的要求，将这个世界带入极端的无政府主义的混乱状态。当我试图在任何一种平台上编译我们的软件时，我几乎必然会遇到问题。令人痛苦的事实是，所有这些平台或厂商的维护者以及我们所依赖的库的开发者们都没有像我们一样想让快一点使所有东西都完美无缺。我甚至怀疑他们是否跟我们一样对『跨平台』支持执着。这让我想到第二个问题——即使我们非常幸运地在某个平台上找到了我们依赖的库，我们又如何知道它是按我们所需的方式配置 (configured) 的呢？很多软件包都有数十种编译时的配置参数，其中有很多参数是我们必须的，或者说，少了某些参数我们的软件也就无法实现某些功能，这不是我们愿意见到的。另外，为了在所有不同的系统上礼貌地要求我们所需的依赖，我们应该如何打包我们的软件呢？只有一种选择，那就是，我们必须维护我们自己依赖的代码并编译我们所需要的版本的库。这么做也是因为我们并不想野蛮的将我们版本的库装到某人的系统上，并且，我们也绝对不想安装动态库，那样做通常会引起混乱（应用程序可能会找到错误的动态库）。所以，我们只是简单的将我们依赖的代码编译成静态库，并静态的连接到我们的可加载模块或我们的代码中而不会影响到别人。通过静态编译和连接，我们知道我们在运行时所用的代码就是我们开发时同样的代码，因此我们也能睡的安稳。所有这些决定跟运行效率或节约磁盘空间、节省内存等没有半毛钱关系，我的想法很简单，我就是想让我的软件能正常地运行。

必须指出，我们确实曾花了一些时间去研究是否有这样的代码存在——它是众所周知的、容易安装的或通常都是已经在操作系统上默认安装的。如果有这样的软件存在的话，并且我们能在所有目标系统上进行测试确认没有问题的话，我们是非常愿意用这样的系统提供的库的。这么说吧，迄今为止，似乎只有一个库符合这个标准，那就是 ODBC。它能入选的原因是它有一个定义很好的而且永久不变的二进制的 API 以及无数的实现。

11.8 在飞机上上网打电话

本文写于 2014 年 8 月在 ClueCon 完毕后回中国的飞机上。

在飞机上上网、打电话一直是人们的一个愿望。而前些日子看到有些关于在飞机上打电话的讨论。没想到，今天，我也用上了。

我乘坐的是美联航从芝加哥飞旧金山的航班，在芝加哥奥黑尔机场有貌似有免费 Wifi，不过，费了半天劲没登录上。航班延误 2 个小时，有些不爽，再加上昨天晚上睡得很晚，特别困，但也得坚持着。

终于登机了，在飞机门边上看到了 Wifi 的标志，小兴奋了一把。不过，后来想起大家的讨论，在飞机上上网可不是免费的，好像还比较贵。跟乘务员确认了一下 Wifi 是否可用，得到的答复是—Hopefully。

落座后，用我的 Nexus 平板试了一下，能连上 Wifi，但要飞机起飞后才能连互联网。困，就直接睡了。

一觉醒来，乘务员过来倒水。喝完后，试了一下 WiFi，果然可以购买了，\$1 和 \$2 两种选择。前者只能支持 Email 或 App，后者可以浏览网页，但两者都不支持流媒体。我选择了前者，一块钱（美元）可以上网一小时，真心不贵。

用信用卡支付后，互联网马上就开通了。上微信、QQ，都没什么障碍。Google Maps 也能正常打开，有点慢，但比起在国内翻墙上 Google Maps 的速度，感觉还是快。

Idapted（我的旧公司）竟然都有北美微信群了，正好有个朋友在，便聊了聊，商量接下来怎么玩。

收发了几封邮件，一切正常。我使用 Gmail 客户端，比起在国内家里上网，一点都不感觉慢。

微信、QQ 发图也很快，只不过，当时国内的朋友都在睡觉，也没什么能聊的。

试了一下 FreeSWITCH 官方的 Verto Demo（WebRTC），给 Brian 打电话，他正在通话，进入语音留言，便留了一段。本来再想给别人打电话，想了半天没想起来该打给谁，就算了。测了一下视频通话，第一帧看起来不错，后面的就花了。该 Demo 默认是使用 720p 的，需要的带宽比较高，所以，花屏可以理解，我估计如果改成 CIF 尺寸应该会有戏。

当然，不管视频是否有戏，语音通话是没有问题了（当然，是基于 VoIP/WebRTC 而不是基于手机的）。看来，虽然提示不支持互联网流媒体，但 WebRTC 还是通的。只是，我忘了上 Youtube 确认一下是否传统的流媒体真的就不行。

给 Brian 发邮件问他收到语音留言没，很快就得到了回复，说一切正常。我在准备给他回邮件的时候，一个小时的互联网时间结束了。

这是第一次在飞机上体验上网和打电话，希望所有飞机上都能以合理的价格尽快开通这项服务啊（美联航好像也还是 Beta 版）。当然，既然 VoIP 解决了打电话问题，用传统手机打电话的需求就不那么迫切了吧？

最后还想说一下，飞机座位下面是有电源的，要不然我也没法在飞机上用电脑写完这篇文章了。

11.9 You pay what I know

虽然一直是这个想法，可是听到 Brian 说出来，就感觉那么有哲理。

『You pay what I know』是说你要为我的知识付钱，而与之相对的『You pay what I do』则是为我的工作付钱。很容易理解，这绝对不是一个境界。

近几年，我一直做 FreeSWITCH 有关的咨询工作。为学习和使用 FreeSWITCH 的个人和企业提供一个强有力的商业支持，一直是我坚持做的。不过，很惭愧，我的大部分工作还是『You pay what I do』的，看来，接下来，我需要向更高的目标前进了。

其实，对于真正 Pay 的人来说，很多人都着眼于你是真正干了多少活，而不是你作的工作对他们有多少价值。而我认为，这是不对的。对他们来说，你是否干活或干了多少本不是他们应该关注的，

他们更应该关注的是你为他们解决了什么问题，帮他们省了或多赚了多少钱，进而决定应该支付你多少是否值得。

当然，要每个公司每个人都做到这一点还真是不容易的，这不仅在中国，在全世界都一样。而我们唯一能够做的，除了提高自身的能力外，便是不断的跟他们讲下面这个耳熟能详的故事（Brian 也给我讲到这个）：

20世纪初，美国福特公司正处于高速发展时期，一个个车间一片片厂房迅速建成并投入使用。客户的订单快把福特公司销售处的办公室塞满了。每一辆刚刚下线的福特汽车都有许多人等着购买。突然，福特公司一台电机出了毛病，几乎整个车间都不能运转了，相关的生产工作也被迫停了下来。公司调来大批检修工人反复检修，又请了许多专家来察看，可怎么也找不到问题出在哪儿，更谈不上维修了。福特公司的领导真是火冒三丈，别说停一天，就是停一分钟，对福特来讲也是巨大的经济损失。这时有人提议去请著名的物理学家、电机专家斯坦门茨帮助，大家一听有理，急忙派专人把斯坦门茨请来。

斯坦门茨仔细检查了电机，然后用粉笔在电机外壳画了一条线，对工作人员说：『打开电机，在记号处把里面的线圈减少 16 圈。』人们照办了，令人惊异的是，故障竟然排除了！生产立刻恢复了！

福特公司经理问斯坦门茨要多少酬金，斯坦门茨说：『不多，只需要 1 万美元。』1 万美元？就只简简单单画了一条线！当时福特公司最著名的薪酬口号就是『月薪 5 美元』，这在当时是很高的工资待遇，以至于全美国许许多多经验丰富的技术工人和优秀的工程师为了这 5 美元月薪从各地纷纷涌来。1 条线，1 万美元，一个普通职员 100 多年的收入总和！斯坦门茨看大家迷惑不解，转身开了个清单：画一条线，1 美元；知道在哪儿画线，9999 美元。福特公司经理看了之后，不仅照价付酬，还重金聘用了斯坦门茨。

11.10 180 还是 183？

在 FreeSWITCH 中怎么配置给客户端回 180 还是 183，是一个经常被问到的问题。然而，答案却远没有你想象中的那么简单。

要明白怎么配置，首先你需要明白 180 和 183 的来龙去脉。另外，你自己还要知道你要干什么。

『什么？我提的问题我当然知道我要干什么！』也许你会这样咆哮，也许你真的知道你要干什么，但是，你得让我知道你要干什么，我才能回答你的问题啊。

好了，我们先不争论这个，我们来说说什么是 180 和 183。

在 SIP 通信中，所有 1 开头的响应叫临时响应，常见的有 100, 180 和 183。这些响应一般是对 INVITE 请求的响应。使用场景是这样的：主叫用户（A）在发起一个呼叫时，会向被叫用户（B）发起一个 INVITE 请求，被叫用户在收到这个请求后，会给主叫用户一个响应，一般的响应流程是回 100，紧接着回 180，或（和）183。

INVITE
A <-----> B
180/183

其中，100 主要是信令层的，它的作用是 B 告诉 A 它收到了 A 的 INVITE 请求。关于它的作用实际也可以讲一讲的，但那就到信令的底层了，大家一般可以不用关注。

180/183 的作用是 B 告诉 A，你可以听回铃音了。

什么是回铃音？这要从更早的模拟电话时代讲起。A 给 B 打电话，B 的话机会振铃，同时 A 的话机回放回铃音（嘟嘟声），这样，A 就知道 B 的话机振铃了。

所以，回铃音是模拟电话时代的事情，它的作用就是给 A 一个提示，对方的电话正在振铃，这样，A 就可以放心地等待 B 接电话。当然，随着技术的进步和时代的发展，回铃音也在进步，典型地，除了嘟嘟声以外，电信运营商也会利用这段等待的时间给用户放一些音乐，甚至是广告，反正闲着也是闲着。这些音乐或广告就称为彩铃，这些是在 A 与 B 正式通话前播放的，因此不对 A 收费。但是由于彩铃也是资源，因此运营商可能会对 B 收费（B 放音乐提高自己的逼格，或者放广告提升自己的形象甚至获取商业利益，收费也是有道理的）。

到了 SIP 时代，就需要在 SIP 中描述这些回铃音或彩铃，这些回铃音或彩铃是真正的声音数据，称为媒体（Media）。但为了将它与真正的媒体（即 A 与 B 真正的通话数据）相区别，将其为早期媒体，即 Early Media。

SIP 的全称是（Session Initiation Protocol，即会话初始协议），它仅仅是完成会话的协商，但是实际的媒体如何传输却需要另外一个协议来协商，负责描述媒体的协议称为 SDP（Session Description Protocol），即会话描述协议。不过，SDP 寄生在 SIP 中，典型地，它寄生在 INVITE 消息和 183 消息中。当 A 向 B 发起呼叫时，它需要把自己的 SDP 放到 INVITE 消息中发给 B，同时，B 在 183 消息中放入自己的 SDP，回送给 A。当双方都知道对方的 SDP 后，真正的媒体数据就可以传输了，这时，A 才听到 B 的 Early Media。

为了帮助大家理解 SDP，我们再进一步。假设 B 接听了电话，B 会响应 200 OK 消息，该消息也带了 SDP（可能跟 183 中的相同也可能不同），用于建立 A 与 B 真正的通话，毕竟 A 给 B 打电话是为了跟 B 通话，而不是为了听 Early Media 中的音乐或广告。

所以，不管是 183 中的 SDP，还是 200 OK 中的 SDP，都是为了建立媒体服务的。区别只是一个叫 Early Media，另一个叫 Media。其实 Early Media 和 Media 差别除了一个是广告一个是真正的 B 的声音外，差别也不大，非要说差别的话，那就是运营商的收费方面有差别，因为 Early Media 是不收费的。

也许有人开始嫌我罗嗦了，但是，我确信，有些人真的不是从根本上理解我上面说的这些，包括一些开始嫌我罗嗦的人。不信，耐心点接着往下看。

好吧，183 和 200 好像有点懂了，那 180 和 183 又有什么区别呢？

180 和 183 之间就差个 3。你说我开玩笑？好吧，这些狗屎都是 RFC 定义的，RFC 就没定义他们的区别！

不过，RFC 划出道来大家就要走啊，因此，大家都按照自己的理解实现了 180 和 183。

现在最流行的实现方式是：180 不带 SDP，183 带 SDP。FreeSWITCH 也遵守这种约定。

所以，180 与 183 的区别不是 3，也不是其它的，关键是看它们带不带 SDP，即 180 也可以带 SDP，183 也可以不带 SDP。为了防止思维混乱，我们下面认为 180 不带 SDP，而 183 带 SDP。如上面所说，这些符合绝大多数人的思维。

那么，带 SDP 的 183 我们上面讲过了，180 不带 SDP 有什么用呢？

我们上面不是说到时代进步了吗？SIP 终端属于智能终端，比以前的模拟电话可先进多了。其中的一点就是它能区别 180 和 183。如果 A 的 SIP 终端收到 183，它就协商媒体，将 B 端发过来的 Early Media 在自己的扬声器里放出来；但如果收到的是 180，没有 SDP 就没法协商媒体，因此，B 就没法给 A 发 Early Media 了。怎么办，总不能让主叫用户干等着啊，所以，A 的话机在这种情况下能自己产生一个回铃音，或任何用户在 A 话机上设置的音乐。

好吧，到这里，不管你有没有理解，反正我把该说的都说了。

那么，言归正传，如何在 FreeSWITCH 中配置回 180 还是 183 呢？

FreeSWITCH 是一个多功能的 SIP 服务器，在此，为了简单起见，我们先把它当成一个普通的 SIP UA。比方说，它就是 B。

在 FreeSWITCH 内部，FreeSWITCH 的行为靠一些称为 Application 的功能函数控制的。当一个呼叫到来时，FreeSWITCH 会查找拨号计划（Dialplan）来决定执行哪些 Application。如，下面的 Dialplan，当一个呼叫到来时，它首先执行 Answer，给对方回 200 OK，然后执行 playback 给对方放一段声音（从声音文件中读取），然后挂机。

```
<action application="answer"/>
<action application="playback" data="/tmp/hello.wav"/>
<action application="hangup"/>
```

如果你自己测试这段 Dialplan，就会发现，FreeSWITCH 不会回 180 也不会回 183，而是直接回 200。这里，answer 想当于 B 摘机应答，在 SIP 中就直接回 200。

当然，『纸上得来终觉浅，绝知此事要躬行。』至于真是这样你还得自己去试，学习 SIP 最好的办法不是看 RFC，而是照着我说的这些（以及《FreeSWITCH 权威指南上的例子》）自己抓包去看。如果试着不对的话，也不要怪我，也许是你的 Dialplan 里边东西太多，在这几个 Application 前执行了你不知道的其它的 Application。

好吧，那怎么让它回 183 呢？

在上面的 Dialplan 中把 answer 那一行去掉，就回 183 了。

为什么呢？

因为 playback 的作用是向 A 播放一段声音，但，在 B 向 A 发送声音前要建立媒体通道。如果有 answer，FreeSWITCH 会发送 200 OK，带 SDP 建立媒体通道。如果没有 answer，那么 FreeSWITCH 就会发送 183，带 SDP 建立媒体通道，而这时，hello.wav 的媒体内容就成了 Early Media。

所以，送不送 183 就看你在 answer 前还是 answer 后执行 playback。

那么 180 呢？也很简单，那就是在发送 180 前执行一个 ring_ready，即：

```
<action application="ring_ready"/>
<action application="answer"/>
<action application="playback" data="/tmp/hello.wav"/>
<action application="hangup"/>
```

在上面的例子中，如果你抓包，就可以看到 180，但你很可能听不到回铃音。原因很简单，answer 执行的太快了。尝试在 ring_ready 和 answer 之间停顿一下，就可以听到回铃音了。下面例子中的 sleep 可以在发送完 180 后暂停 2 秒钟（2000 毫秒）再发送 200 OK：

```
<action application="ring_ready"/>
<action application="sleep" data="2000"/>
<action application="answer"/>
```

我们已经知道怎么让 FreeSWITCH 发 180 还是 183 了，问题解决了吗？

显然没有，我知道你在实际应用中没这么简单。我们在这里把 FreeSWITCH 当成了 B，但实际你希望 FreeSWITCH 是 FreeSWITCH，B 是 B。什么意思呢？FreeSWITCH 实际上是一个 B2BUA，它是一个中间人，你希望的拓扑结构是这样的：

A <-----> FreeSWITCH <-----> B

哎呀呀，你看你看，我第一句话说中了吧，针对你提的问题，答案远没你想象的简单。现在，我们需要列出（猜出）你想要的各种情况了。

好吧，就算这回我猜中了，继续往下说。

中间人也是不好当的，因为，他可以有 N 种不同的策略。至于使用哪种策略，看 B 的心情，也要看 FreeSWITCH 的心情。啊，说白了，又是没有标准答案。所以你还是要把你想要的告诉我。如果你不告诉我，我就得猜，或者把所有 N 种可能的情况都告诉你。

首先，我们先看一种熟悉的情况。FreeSWITCH 可以假装它就是 B，这样，配置方法跟上面讲的基本一样，只是它在假装后还要假戏真做，要用 bridge 这个 Application 再去呼叫 B，并把电话接通。

```
<action application="ring_ready"/>
<action application="sleep" data="2000"/>
```

```
<action application="answer"/>
<action application="playback" data="/tmp/hello.wav"/>
<action application="bridge" data="user/B"/>
```

所以在上面的配置中，至于是回 180 还是 183，配置方式跟上面讲的一模一样，就没必要多说了。

其次，FreeSWITCH 心情好，想听听 B 的意见。如果它即不执行`ring_ready`，也不执行`answer`，而是直接用`bridge`去呼叫 B。

```
<action application="bridge" data="user/B"/>
```

这种情况其实也简单，那就是，如果 B 向 FreeSWITCH 回复 180，FreeSWITCH 就向 A 回 180；如果 B 回 183，FreeSWITCH 就向 A 回 183。这种情况其实就相当于 FreeSWITCH 不存在，所有消息都是透明的。（不过，要记住：FreeSWITCH 是一个 B2BUA，即它是一个中间人，它不会直接拿 B 回给它的 180 或 183 消息『转』给 A，而是自己新产生了一个 180 或 183 消息回给 A。当然，也许你不关心这个，但你说得越不清楚，我越累啊，要不然人家还会说我的回答不严谨呢。或者，万一我猜错你问的意思呢？）

再次，FreeSWITCH 跟 B 这两天不大对付，什么事情都拧把。B 回 180，FreeSWITCH 就回 183，B 回 183，FreeSWITCH 就回 180。

好吧，看起来是越来越复杂了。又是两种情况。

先看 B 回 180 的情况。FreeSWITCH 要想给 A 回一个 183，由于 B 的 180 中不带媒体，FreeSWITCH 就要『造』一个媒体出来，因此，它想了一种办法，在 bridge 之前造一个媒体：

```
<action application="set" data="ringback=/tmp/ring.wav"/>
<action application="bridge" data="user/B"/>
```

由于在执行`bridge`之前还没有 B，因此 FreeSWITCH 不知道什么时候 B 回 180 还是 183。通过在 bridge 之前使用 set 设置一个变量（ringback），实际上相当于 FreeSWITCH 给 bridge 下了一个套，到了 bridge 阶段，不管你什么时候 B 回 180，FreeSWITCH 都会向 A 播放事先『造』好的回铃音 ring.wav。当然，FreeSWITCH 要向 A 发送媒体前要先用 183 建立媒体通道，这就完成了 180 到 183 的转换。

所以，这也是 FreeSWITCH 设计精巧之处——同是一个 bridge，通过一个 ringback 变量改变了它的行为。

再看 183 变 180 的情况。

如果 B 向 FreeSWITCH 回了 183，FreeSWITCH 要向 A 回 180，那就不能把媒体信息送给 A。所以，实现也很简单，还是一个简单的 bridge，只是，把 B 送来的 Early Media 忽略掉就行了：

```
<action application="ring_ready"/>
<action application="bridge" data="{ignore_early_media=true}user/B"/>
```

跟 set 不同。set 是一个 Application，它作用于当前的 Channel，即 A 那一个 Channel（那时候还没有 B）。而`{ignore_early_media=true}`这种语法，在建立 B 端的 Channel 的同时，将`ignore_early_media`作用于 B。再强调一次，FreeSWITCH 是一个 B2BUA，因此 A 跟 B 间的通话要产生两个 Channel，即所谓的 a-leg 和 b-leg。

在建立 B 通道的时候，`ignore_early_media`也是给`bridge`下了一个套。即不管什么时候 B 回了 183，忽略它。由于我们选择了忽略，因此，为了让 A 仍能听到回铃音，我们用`ring_ready`在`bridge`前送一个 180。严格来说，它不是 183 变 180，因为 FreeSWITCH 以收到 183 前就已经送出了 180，但是，如果你不趴在 FreeSWITCH 内部看，谁知道什么时候变得呢？

N 种情况讲了 N 种了，永远都会有 N+1。既然 FreeSWITCH 位于中间，那它能不能把 B 发过来的广告（彩铃）换成它自己的广告呢？能是能，但我不教你怎么做。不过，不幸的是，如果你不是特别笨的话，我上面已经教会你了……

读到这里，你认为这个问题好答吗？

11.11 从 3·15 看电话号码透传

杜金房/2015.03.17

2015 年的 3·15 晚会，报出了好多通信业违规群呼和透传电话号码的事。关于 3·15 的政治问题我们就不说了，说了也没意思，我们聊点技术方面的。

『透传』这个大众并不熟悉的专业术语，仿佛随着 3·15 一夜之间就普及了。

新闻稿上说：『为了让骚扰电话更加具有迷惑性，增加用户识别难度，金伦科技还会使出更厉害的手段，帮助呼叫中心随意显示主叫号码，这种技术，在这一行业被称作「透传」……』

『XX 公司展示了这一神奇的技术，轻而易举地在记者手机上显示了一个实际并不存在的电话号码』。

『同样，XX 也将记者随意提出的一个号码，轻松地拨打在了手机上。』

『……』

这都是些好高深的技术啊！

不懂的人打眼一看，哟，这么厉害的一个技术，如果掌握在坏人手里，这不天下大乱了吗？

其实『透传』真不是什么厉害技术，更不是什么洪水猛兽。透传就是在被叫的电话或手机上显示任意的主叫号码，包括所谓的 110。这一点，其实谁都可以做到。有合适的落地网关，在 FreeSWITCH 中就可以轻而易举地做到，如：

```
originate {origination_caller_id_number=110}sofia/gateway/gw1/138xxxxxxxx &playback(test.wav)
```

那么，如此说来，不真天下大乱了吗？

当然不是。其实主叫号码的显示跟写信是一样的。这些年写信少了，但大部分人至少还都写过信吧？（没写过信的人想想快递也行。）写信的时候，上面是收件人地址，下面是发件人地址。收件人地址当然要写正确，那发件人地址是不是想写什么就写什么？其实这里的发件人地址就相当于电话中的主叫号码，你想送什么就送什么。

电子邮件也一样。在发送电子邮件的时候，发件人的邮箱地址叫做 Envelope Address，也就是相当于信封上的地址，也是随便你想写什么就写什么的。所以，垃圾邮件泛滥就是这个原因。

普通信件要经过邮局，投递到收件人手中。一般来说，邮局是不会审核你的发件人地址的。而且你随便把信投入哪一个邮筒也没人管。所以，写匿名信或者伪造信件是很容易的事。虽然收件人可能通过笔迹等看出是伪造的信件，但追查发件人却是几乎不可能的。

电子邮件相对来说管得要严一点，所有的电子邮件提供商都不希望你往外发送垃圾邮件，所以，一般在你发送时要对发件人进行权限验证（一般是密码验证）。当然，对发件人验证是近十来年的事了，在十几年前，发邮件几乎没有验证的，随便发。

光在发件的服务器上验证也不行，有的人可以直接将电子邮件发送到收件人的服务器上。如果收件人的服务器不想被垃圾邮件骚扰，那就要采用一些手段如检查发件人的 IP 或域名。但这个也很难防止垃圾邮件。

电话系统也类似，你拿起电话，可以打通几乎任何人的电话。所以，骚扰电话是畅通无阻的。现在，有些手机里有一些功能，能使用黑名单过滤掉一些已知的骚扰电话。因此，打骚扰电话的人一个最简单的策略就是经常换电话号码，以提高电话的接通率。

电话从主叫用户到被叫用户，要经过很多中间的交换机。在被叫端交换机上检查主叫是否合法一般来说是不现实的。因此，为了防止恶意显示主叫号码，对主叫号码的合法性检查就需要在主叫侧进行，一般来说，是在最靠近主叫用户的交换机上。

对于普通的模拟电话来讲，是无法改变主叫号码的，因为电话线就唯一决定了电话号码。而当电话进入数字时代，尤其是一些大的呼叫中心都使用数字中继或 IP 中继（SIP 线路），随意显示电话号码就变得容易了。主叫用户只需要在发起呼叫时把电话信令中的主叫号码填成自己心中想要的号码即可。是的，所谓的透传就是这么简单！

当然，为了防止天下大乱，在主叫侧的交换机上是对主叫号码进行检查的，也就是说，运营商的交换机上实际上有一个检查员。如果检查员睁一只眼闭一只眼，你的主叫号码就过去了，送到其它的交换机上以后从此畅通无阻（少数时候也有阻，即所谓的主叫号码传送规范，我们就不深入讨论了）。如果检查员比较负责，那么，你发的主叫号码就无效了，检查员会把它替换成你真正的电话号码。由于绝大多数检查员都很认真负责，因此到现在没有天下大乱。

所以，检查员相当于一个开关，能否『透传』完全是检查员说了算，而不是什么厂家开发了什么先进的技术。简单是一派胡言！

那么，这个开关是怎么打开的呢？其实这个谁都知道。提供技术支持的厂商固然有利益驱动，但如果不是跟检查员勾结，再厉害的技术能过得第一道坎吗？

那么，是不是就应该所有人都遵守规则，永远都不需要透传呢？当然不是。透传有透传的用处。在需要的时候，就应该有透传。如下图：

客户来电 ——PSTN——> 公司总机 ——PSTN——>转出到个人手机

在上图这种情况下，如果客户的来电到了公司的总机上。总机发现被叫不在办公室，再通过 PSTN 转到被叫的手机上。在这种情况下，如果电话号码不能透传，那么，在被叫的手机上就显示公司总机号，但是被叫真正想看到的是客户实际的电话号码。

所以，这是一个实实在在的合法的需求。但是，95%（瞎猜的数字）以上的公司都做不到这一点。为什么？运营商不让你透传。

所以，问题的实质是，需要并且合法使用这种功能的人得不到应有的服务。坏人却总是轻而易举地能做到。而且，由于这些非法的使用，使得守法的人越来越难得到这种合法的业务。

无论如何，我今天想说的是—不要把『透传』变成一个贬义词。另外，媒体还说：『某某科技专门研发了一套系统，……，每天群呼上千万个。』技术无罪，不要误导了群众，把重点放到那些随意贩卖个人信息的那些人吧。

回到我们的技术原点，在 FreeSWITCH 中，默认是禁止透传的。但如果你想让『检查员』退休，『透传』你想要的电话号码，可以把用户配置文件中类似如下的两行删掉或注释掉：

```
<variable name="effective_caller_id_name" value="Extension 1009"/>
<variable name="effective_caller_id_number" value="1009"/>
```

好了，放过技术吧，技术无罪。看你怎么用。

最后，学会了但不要做坏事哟。《FreeSWITCH 权威指南》上还有对主叫号码更详细的讨论。

11.12 FreeSWITCH 帮我找到了工作

杜金房 / 2010.10

今天，跟 FreeSWITCH 邮件列表中的一位网友聊天，忽然想到了它曾发到列表中的一则故事。感悟颇多。

该网友是一个盲人，靠屏幕阅读软件看电脑，用语音识别软件转换成文字跟我聊天（Google Talk）。我一直有个疑问，我用 FreeSWITCH 毛算也有三年了，在学习和使用中经常遇到一些问题，我手眼健全都需要很常时间解决，作为一个盲人，他是怎么做到的呢？

从聊天的语言来看，跟正常人别无二致，而且，可能是他使用语音识别的缘故，反应起来甚至比正常人都快。

当然，跟他聊天并没能完全解答我的疑问。我想，如果有机会去阿尔及利亚，我一定找机会就见见他。

在工作的生活中，每当忙了、累了，我就想到这则故事。今天，我使得他同意，把全文翻译成中文，放到这里。需要指出，由于原文可能是用语音识别写的，因此有些句子不太顺，我尽量按我的理解翻译了。

邮件列表中的同学们，大家好，

我叫 Meftah Tayeb，来自阿尔及利亚。我双目失明，但自从 1.0.1 时代（版本）就开始使用 FreeSWITCH。

我最开始用的是 Asterisk，在我的生命中，它是一款非常奇怪的 VoIP 软件。感谢 Miconda 在 IRC 上的 #openser 频道，把我从 Asterisk『重定向』到 FreeSWITCH，那是 2008 年的事情。

我于 2008 年 12 月开始学习 VoIP 的基础知识。感谢 IRC 上 #freeswitch 频道上的伙计们对我的帮助。最开始有 anthm, Michael S Collin, brian (BKW)，感谢你们以及其它所有的好人。

在 2009 年 8 月，阿尔及利亚电信以及政府开始屏蔽 SIP 流。我并不使用 SIP 来做 VoIP 生意，但是，老实说，我只是想用 SIP 连接到每周一次的 FreeSWITCH 公共电话会议，在会议里，大家都讨论 VoIP。我开始向我当地政府（或电信，原文是 local city）抱怨此事，但没人理我。我想直接进入总经理办公室，因为没有 SIP 我无法使用我的 PC。我找到了总经理秘书，他接待了我，并听取了我的抱怨：『你们为什么屏蔽 SIP?』。然后他问我：『那你为什么一定需要 SIP? 你想无证经营 VoIP 业务吗?』我把我的实际情况告诉他。他说：『好吧，没问题，我会给你开通 SIP，但你下周一定要到这里来。』。我说：『OK，那没问题』。

然后，我回到家，看到了新的东西：

1. 一个静态 IP 地址，绑定在我的 ADSL 帐户上
2. 完全开放的 SIP

OK，在下个周一我到了总经理办公室，见到了总经理秘书。使我奇怪的是，他问我：『你有工作吗?』我说没有。他说：『你肯定有，告诉我』，我还是说没有。然后他说：『你在为阿尔及利亚电信工作』! ;)

他给了我一个惊喜！现在，他给了我一个好工作、免费的房子、以及免费的护理，还有司机。

所以，在这里谢谢 FreeSWITCH 项目，特别是 FreeSWITCH 的主人以及所有贡献者。

谢谢。

邮件原文在此 (<http://lists.freeswitch.org/pipermail/freeswitch-users/2010-June/058789.html>)，以下是全文拷贝：

[Freeswitch-users] get your job aguinst freeswitch

Meftah Tayeb tayeb.meftah at gmail.com
Sun Jun 6 11:34:09 PDT 2010
Previous message: [Freeswitch-users] Hi
Next message: [Freeswitch-users] get your job aguinst freeswitch
Messages sorted by: [date] [thread] [subject] [author]

hello list,

i am meftah tayeb, a blind person from algeria that was using freeswitch sunse 1.0.1 release

i started firstly with asterisk, that was the very strangett voip application in my life

and thank to miconda that redirected me to freeswitch, from asterisk in #openser in 2008

i started learning voip basic in dec 2008

thank to the #freeswitch folk that teached me all this, including firstly anthm, Michael S Collin, brian (BKW), sekil the nice GUI and all other

in ogust 2009, algeria telecom and the algeria gouvernmant started blocking sip traffic

i was not using it for voip business, but, honestly, just to connect to the public freeswitch conference and the weekly voip users conference

i start a complain in my local city, no reply from AT

i decided to go to the general office, because i can't use my PC without SIP

i got the general directore secrutary

ok, he receyved me and heare me saying why you are blocking sip?

so he asked me

why you need sip?

do you do voip business without autorisation?

i explaned to him my actual situation and he say: ok, no problem i will open the sip for you, but conditionaly

the next mondey you will be here

i say ok no problem

so, i returned to my home and i see something new:

1. a static ip address linked to my ADSL account
2. sip completly open

ok, next monday i was in the general office and i meet the gebnral directotore surprisingly

he asked me:

do you have a job?

i say no

but he say yes, you have, tel me

i say no aguin

and he say you work for algeria telecom

;)

he surprised me with this

now, he gave me a:

```
good job
free house
free care with driver
so please say thank to the freeswitch project especially the owner and
try to donate to him a much a pocible
thank you

--
Meftah Tayeb
algérie télécom SPA
phone: +2132176XXXX
phone (INUM): +88351000128XXXX
mobile : +21366034XXXX
mobile (INUM: +88351000128XXXX
http://www.algerietelecom.dz
```

11.13 Cluecon 2011 小记

杜金房/2011.8

搞 FreeSWITCH 好几年了，一直盼望能去 ClueCon 见识见识。今年，终于有了一个机会。

准备材料，办签证，虽然罗嗦，但还是顺利地通过了。保险起见，提前两天出发。所以，7号下午从北京出发，到了芝加哥还是7号下午，到酒店已经是晚上了。8号随便走了走，玩了一天。由于时差的关系，第二个晚上还是睡地比较凌乱，因此9号早上起的有些晚。又因为被人指错了路，所以到 ClueCon 会场的时候已经是九点多了。

在登记的时候遇到了 Michael Collions，他是大会的主要组织者，长得很高大也很帅。而且他的发音很纯正，因此听起来没有任何困难。

进到会场后发现几百人的会场坐无虚席，我在最后面靠墙的地方找了个位置坐下。正好没耽误 Anthony Minessale 的演讲。

他讲到 FreeSWITCH 的一些新特性，及最近做的一些优化。由于 sofia 是单线程的，所以做大并发一直有问题。最近的更新通过使用 FreeSWITCH 的内部信令队列使得 sofia 已经能承受 sipp 1000 cps 的话务，并能达到 30,000 个 channel (call legs)。Anthony 还幽默地说：『Dont try this at home.』。因为我坐在最后排，没看清他用了一个什么服务器，只是隐约看到 status 命令的输出数字都很长²¹。另外，与 Michael 比起来，听他说话就比较费劲了。一是他说得比较快，声音不洪亮；二是他好像也不照顾我这从中国来的，讲得慢一点：(。

²¹后来，Ken Rice 在 2012 在邮件列表中证实了：Tony demonstrated FreeSWITCH running 1000cps 30sec call duration with media for a total of 30K concurrent calls @ ClueCon last year (just a few weeks shy of a year ago)。参见：<http://lists.freeswitch.org/pipermail/freeswitch-users/2012-July/086156.html>。

偷眼看了一下会场，所有桌子上都放了好多插排，有的人也自带了各种各样的插座转换器，总之所有笔记本（除了像我坐在最靠墙一排的）都能得到电源。Mac 居多，使用其它本本的少数人用 Linux，有的用 Windows。

午餐是自助的，看起来应该挺贵，但没什么可选的，只好将就着选了点菜叶和牛排。不过看到外国人吃的都很 High。

中午休息的时候我看到 Giovanni，他是 mod_skypopen 模块的作者，意大利人。虽然第一次见，但由于以前在电子邮件和 IRC 上交流很多，因此一见如故。正好他旁边有个位子，我就坐到第一排了。

大牛们的演讲都很精彩（当然也有不精彩的），确实学到不少东西。比如 Mathieu René 他们除了做了 mod_rtmp，还把 FreeSWITCH 移植到 iPhone 上了；2600hz whistel 的新发展等等。

上新浪微博给大家直播了一下。由于现场人比较多，再加上手机等需要上网，因此也经常有网络不通的情况。另外国内是晚上，因此也没大有人回复，因此直播的热情自然就少了好多。

晚上是一个欢迎的 Party，大部分人都去了。我跟美国、印度、南非人共四个人打了两局保龄球，成绩还不错，每次都是第三名，要知道，以前仅仅在烟台打过一次啊。当然最后累得手都疼了。

遇到一个中国人，在美国上大学毕业后在纽约一个小公司工作。好几天没说中国话了，因此猛聊了一阵子。他老板是华裔，但不大会说中国话。

时间总是过得很快，半夜回去上上网很快就睡了。

噢，差点忘了说，由于我没有预订酒店房间，Michael 帮忙找了个室友，William Dale。他自己有公司，做了 labortimetracker，一个网上考勤打卡系统，在美国市场还不错。因此我们俩还就要不要把他们服务拓展到中国聊了好一阵子。

第二天的会议没什么值得多说的。通过昨天晚上的 Party，大家也都比较熟悉了，因此打打招呼，随便聊聊什么的。

好不容易找 Anthony 聊了半个多小时。找他聊天真是比较困难，因为大家都在找他说话，而且我想独占他一点时间，因为怕跟他交流有困难，所以想慢慢聊把问题说得清楚些。还不错，我们聊了一些过去的事情，请教了一些技术问题，也聊了些 Google Talk、Skype 以及 Microsoft，还有 FreeSWITCH 在中国的发展等等。有大约 10% 的东西听不懂，都哈哈过去了。

晚上跟 Michael 等 7、8 个人一块吃了晚餐。AA 制，花了 30 多刀，吃了个不怎么好吃的意大利面。话说在这边吃东西还是挺贵的，而且还得交税，还得给小费。

吃完饭以后又回去跟他们聊了会，Anthony 一伙人则弄了吉他什么的一堆乐器在那儿娱乐。后来我累得实在撑不住了。回房间后倒头就睡了。

第三天，也是最后一天了。早上四点多就醒了，再也睡不着，准备演讲的幻灯片。因为之前跟 Michael 打过招呼，午饭后他给安排了一个表现的机会。第一次用英语上去讲，而且台下都是界的大牛们，好紧张。心里默念了好多遍的台词都忘了说。不过还好吧，比我预想的讲得长了点。完了后看大家反应应该还算不错。尤其是 Giovanni，说了好几个 Very Good。也有人过来要名片，大概是因为我提到中国有 1,339,724,852 这么多人口吧。

然后跟 2600Hz 的 Darren 简单聊了会，他说我做得很东西跟他们都很像。我是啊，英雄所见略同嘛。由于他要赶飞机，只好以后再联系。

时间过得很快，三天的会议马上就结束了。不爽的是，会上抽奖有一个 MacPro，两个 iPad，若干 snom 及其它电话，始终没有叫到我的号。

会后大家都走了，我不急着赶路，磨蹭了一会。看到 Mike Jerris，请教了一些 GDB 的问题，感觉豁然开朗。好多年，没有人手把手给我讲东西了。

后来闲着没事帮他们收拾了一下东西，就打道回府了。

当然期间也跟 Brian West 打了招呼，不知道怎么形容，反正感觉他很帅很有意思。

很遗憾的是光顾聊天和兴奋了，忘了拿出手机跟他们挨个合个影。

回想起来整个会议还是安排得挺不错的。他们组织和服务的没几个人，却秩序井然。大家也随意把笔记本什么的丢在桌上出去说话，好像也不怕丢失。另外感觉这几天英语也有所进步，至少是说得比以前快了：)，而且，感觉有些在台上演讲的人发音还不如我好呢。尤其像我搞了这么多年英语教育，感受还是很深的。其实学习语言真的不难，当你听的说的甚至想的都是英语时，自然就进步了。所以，环境是很重要的。而我认为第二重要的是别人的鼓励—在国内，觉得自己说的不好就不敢说；而在美国，甚至你只会说几个单词别人就说『哇，你英语说的不错』。是吧？想想是一样的道理，我们在中国遇到有老外说几个『谢谢』，『你好』之类的也会夸奖他们。

好像扯远了。在飞机上写下这些，算是个小结吧。

11.14 ClueCon 2012

杜金房/2012.8 原文名称《八月旅行记》

2012，8月初，借 ClueCon 会议出去转了一圈，小有所感。

本来从很早就打算，但还是在机票上出了问题，定好的行程只好相应改变。

从北京出发，第一站是华盛顿。安检、过海关都很顺利，比去年快好多。在机场碰到奥巴马，还一块合了个影。

然而应该在两小时后出发去芝加哥的下一班班机始终不能到港。延迟、取消，美联航也用同样的招数招待旅客。时已至午夜，大部分人顺从的改签了第二天晚上的班机。但我等不了那么久，因为 ClueCon 在会议开始前一天晚上要举行一个热身晚宴，我不想错过。与其他几个同行的中国人商量，他们则更急，第二天一早就有会。后来经一再与航空公司客服磨蹭，我们 5 个人中有两个人转签了第二天最早的航班，其余三人则给了 Stand By 的登机牌。Stand By 的意思就是，如果有人在午夜到早上 6 点的时间突然取消机票，我们就有机会登机，或者，等待有的人早上睡过了头。不管怎么讲，这都算是最好的结果。



图 11.1: 华盛顿机场跟奥巴马合影

由于航空公司以天气原因为由（实际上两边天气都很好，我始终觉得是上一班飞机不知出了什么问题）拒绝提供住宿，又由于我们离登机也只有 6 个小时时间，几个人一商量决定有两个留在机场，其他三人打车去市里转一圈。

司机是巴基斯坦人，说话跟印度人口音差不多，很多都听不懂，当然，他听我们的英语也有困难，呵呵。机场离市区也不算近，转了三个来小时，路过国会大厦，白宫和五角大楼，其实晚上没什么好看的，但也算是『到此一游』，『不虚此行』吧。

值得一提的是，就在白宫正对面有一个小帐篷，有一个人在看守着，旁边的牌子上写着：『WHITE HOUSE 24 A DAY ANTINUCLEAR PEACE VIGIL SINCE 1981 MAINTAINED BY CONCEPCION THOMAS …』。大意是反核武器。与小伙简单聊几句，他倒是很健谈，口音很纯正，只是大部分都是哲学和逻辑，我也不记得多少。印象最深的就是他们有 10 多个人轮流到这里值班，在白宫正对面，坚持了 30 多年了²²。

回到机场很快就到登机的时间了，又多过了一次安检。还好，最后我们三个 Stand By 全部转正，路上一顿好睡，飞到芝加哥是早上 8 点。乘地铁到 Hosteling International Chicago，由于去年到过一次，因此很顺利，只是 Check In 的时候被告知前一晚由于我没到，订单已被取消，下一次 Check In 的时间是下午 3 点半。天啊，我告诉他们我必须睡觉，后来他们给我一个临时卡，我可以上二楼大厅里休息。

一觉醒来，已是下 5 点半了，没想到在沙发上睡了一整天！匆匆办理了 Check In，便急忙赶去晚宴。虽说有地图，但还是有点迷路，茫然间遇到了 mod_skypopen 和 mod_gsmopen 的作者 Giovanni 大

²²后来查了一个还真有那么回事，参见：https://en.wikipedia.org/wiki/White_House_Peace_Vigil。



图 11.2: White House Vigil

侠，顿时感觉找到了组织。赶到酒店，正赶集合。

其实晚宴也不怎么隆重，就是大家吃饭，随便聊一下。其间跟 OpenSIPs 的几个哥们聊到 OpenSER 及 Kamailio，对于历史他们则只字不提。从聊天中才知道他们的核心团队原来是来自罗马尼亚的，我之前光知道是欧洲，呵呵。

吃完饭回青旅，写演示文稿—由于最近事情有些杂乱，飞机也不正点，因此所有这些工作就压到了最后。

期间看到邮件，Anthony 还提交了一些代码，并且打上了一个我几个月前写的补丁，感觉到他甚是勤奋。

第二天按时赶到 ClueCon 会场，与几个熟悉的简单聊几句，正式的演讲便开始了。形式大约就是主题演讲，大家提问，茶余饭后大家也可以相互交流。

大家的演讲都很精彩，尤其是 Anthony 在演讲中说到『FreeSWITCH 1.2.0 版已经在一分钟前正式发布了』，真是非常有意思。怪不得他在前一晚还在孜孜不倦地提交代码。

另外，这次看到有中国人上去演讲了—来自 OpenVox 的 Belief Mo。午餐倒是没什么好吃的，随便吃了一些。晚上 OpenVox 的哥们请客，也对他们的产品加深了一些了解，聊到中国的开源软件，唏嘘不已。

晚上是 Anthony 的生日，有人准备了蛋糕，Party 就在我们开会的会场上举行。我把我的书送给他，他虽不懂中文，但看起图来很在行，指着一幅 X-Lite 的图兴奋地说，哈哈，我到现在还在用这个，新版的有 xxxx 问题…Brian 同志好像喝的不少，一直在唱卡拉OK。我就在后面写明天的演示文稿。很高兴遇到 William King(mod_vlc 作者)，学习了好多东西。

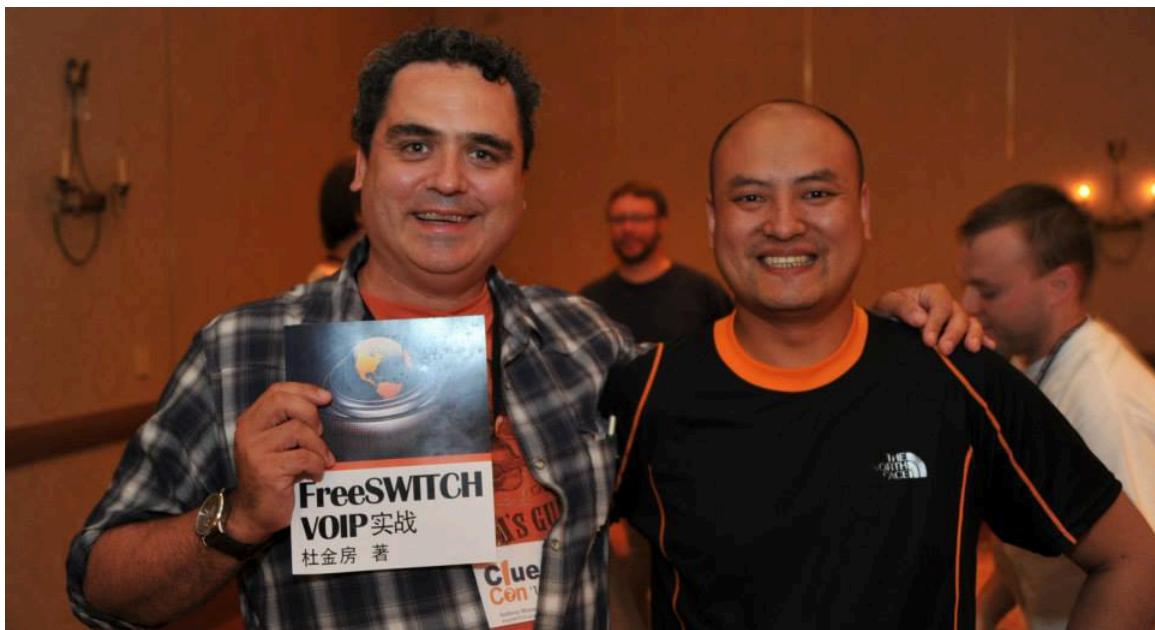


图 11.3: 本书作者与 Anthony 合影

晚上就住在开会的酒店，后来我回房间写到 2 点多，总算完成差不多。由于时差的关系，因此睡得也不算晚，第二天一早起来又修改了一下，匆匆赶去会场。就在我出去 711 买了点早点的工夫，据同桌说抽奖号码可能念到我了，后悔不已。

今天第一场演讲就是我的，过程还算顺利，英语也比去年感觉顺畅一些，只是还是自己注意到了有好多 Ur.. 和 Ah…。由于演示文稿使用了 Prezi，讲起来思路很清晰，事先准备的几个 Live Demo 也都很顺利。讲完，还剩下一分半钟的提问时间，算是刚刚好。当然，今年没忘让同桌帮我照几张照片。

后来过来找我聊天的就比较多了。有一个人说他用到我用 Lua 写的 Dialer，我说我写完后就一直没用，很高兴对他能有帮助；也有人关心我们在视频方面的研究问能不能开源；Michael Jerris 说他虽然看不懂中文字，但他都知道我写的什么，因为他自己对 FreeSWITCH 太了解了；Brian 也想要一本书，问我能不能寄给他……后来晚上有一个夏威夷的华裔哥们请我吃饭，他做 ISP，占有一部分市场，以前用 Asterisk，现在想改 FreeSWITCH，便来到大会听听。

晚饭后大家都去了一个玩的地方，我也不知道叫什么，但发现原来是管饭的，由于我们已经吃完，便直接上二楼去玩游戏，说实话我除了上高中时打过街机的三国志外，现代的 Video Games 基本都不会玩，便选了一个叫 Terminitor 的游戏打枪（机关枪！）打到手抽筋。

说话间就到第三天了，我坚持到最后终于中了一个 Andriod Pad 大奖，今年的奖品没有 Macbook 了。

散会后我又呆了一会，大家都走的差不多了，抓住 Ken Rice（负责维护 FreeSWITCH 的分支及稳定性）给我讲了半天 FreeSWITCH 代码管理的东西，受益匪浅。

总之，大会就这么结束了。应到 200 多人，实到 200 多人。OpenVox 的兄弟们去了，亿联的话机也去了，这次更多了一些中国人的面孔，让我也不感到那么孤单……

晚上又回到 Hostelling International，省钱嘛，同时这里会遇到好多人，不会那么闷。当然第一先睡一觉，这几天实在是太累了。3 小时后起来随便吃了点饼干，煮了包从国内带去的方便面。不像去年，这次对西餐倒有些适应了，剩下的一包一直到香港到了最后一晚才解决掉，那是后话。

第二天报名参加了一个林肯公园 Walking Tour。一个 70 多岁的老奶奶带路（志愿者）。还顺路带我们去了她家，三层楼，据说都是她的，一层租出去做店铺，我们去了二楼。屋子倒不小，但是堆满了东西，花花绿绿的，显得很狭窄。她好像以前挺爱跑步或骑自行车的，家里有好多的奖牌。

后来带我们在街上转转，中午随便找小店吃饭，下午去了林肯公园，免费，这里的空气很清新。噢，对了，忘了说，我们团队有来自香港、法国、LA、德国、墨西哥、加拿大等约 10 个人，很好玩。不过，逛完公园后就剩下好像 5 个人了，我们随后一直走到海边（密歇根湖边），好大的风。然后又上到 John Hancock 中心 96 层喝了杯啤酒。高处鸟瞰芝加哥别是一番风味。

晚上回青旅一直写代码，视频录像 mp4 取得了较大的进展，录的像基本上能放出来了。又收发了些邮件，在米国用 Gmail 是相当的爽啊，除了没有墙外，速度也相当快。Clone 个 FreeSWITCH 代码也就一两分钟的事。另外在 ClueCon 上听一哥们说他们是做接入的，直接用天线架 WIFI，网速百兆，可覆盖 30 英哩。不过万事都没有完美的，上新浪微博就相当的不爽。

翌日启程回国，由香港转机。我之前一直以为北京和香港到芝加哥距离是差不多的，后来才明白地球是圆的，从北京走飞机沿北极圈飞要近好多，大约差三个小时，我还得再用三个小时从香港回京，下次记着可不这么绕了。

由于又遇到流量控制，起飞晚了，到香港已是下午 6 点。之前没能订上 Hostelling International 连锁的青旅，只好在 Hostelworld 的订了。乘地铁到市区（香港站）再转公交，到旅舍已是 8 点多了。因为很难找，转了好几圈。后来发现自己竟然是在铜锣湾，离铜锣湾地铁也就几步远。

出去找饭吃，进了一家小店。点完了才发现公仔面竟然就是方便面。不过，猪扒和煎鸡蛋还是非常的好吃的，也可能是我太饿了。

回来，在路上慢慢走走，可以仔细看一看。这里街道很窄，一般也就两个车道，单行道比较多。再加上两边的楼都特别高，显得格外拥挤。后来第二天的时候我到其它地方看到也大抵如此，最宽的也不过双向 4 车道，不过，路上的车都井然有序，没看到有北京那种大堵车的感觉。

街上大都是汉字的招牌，对于刚从米国回来的人来说，看起来真有些亲切。再好好看旅舍，原来就相当于是家庭旅馆，这里并没有小区，就在路边，一楼二楼是店铺。楼道非常窄小，电梯虽然标明荷重 6 人，我看我一人在里面还差不多，最多的时候也就 5 个人，因为电梯是正方形的，怎么乘 6 个人呢？

后来看到其它地方的房子也差不多是这种格局，当然是否有富人住的小区我就知道了，总之，我住的房间是很狭小的，根本没有南北通透一说。

同屋的舍友晚上 1 点多才到，是新加坡的，刚从北京到香港。男男女女几个人一直吵到半夜，我也没睡好。不过还幸亏他们有一个人带了个万能插座适配器，我的 Mac 和 iPhone 才能充电。之前只想着去美国插座都是没问题的，就没想到香港的不一样。上了会网，简单查了一下攻略，才发现我乘的地铁并不是最省钱的交通方式，如果乘 A11 路公交可直达我的地方，只要 40 多元（以下都是港币），而我坐地铁单程就是 100 元。

第二天早晨起来已是 10 点了，吃饭就当中午饭了。去了据说很经典的『池记云吞』，确实挺好吃，也不贵。

到地铁站买了个一日任意坐的地铁卡，第一站到旺角，不知道为什么，只是因为以前听说过这个名字。刚出地铁站就下雨，只好钻进一家商场瞎逛，什么也没买—衣服什么的暂时不需要，瓶瓶罐罐里的东西我也不感兴趣。看了一下，大概除了奢侈品外，其它的还是比大陆贵，而且后来我去超市还特意看了一下酱油的价格，去香港打酱油一说我认为有点不靠谱。

我是后来才知道机场有免费地图的，当时不知道，所以没有地图，也不敢瞎走，又乘地铁到了油麻地。在街上随便走走，冷不防走进了一个农贸市场，里面什么都有，模式跟国内相似，又有差别。到底差在哪里，也说不上来，只是记得有一家卖肉的招牌上写着『保證足秤』，当然是用繁体字写的。

虽然听不懂也不会说广东话，但听起来还是很好听的，就像年青时喜欢的粤语歌一样。另外，以前也了解一些广东话的，粤语有 6 个音调，因此变化比较多，抑扬顿挫，比较好听。而且，旧时兵家必争之地都在中原，南方的战乱比较少，因此语言文化受到的冲击比较少，便保留了好多古音（如入声字在普通话里已经没有了），对于我这个古代文化的粉丝来话，还是很感兴趣的。另外香港和台

湾也一直使用繁体字，所以保留古代的韵味都比较浓。有趣的是，虽然他们近代受到西方文化的冲击非常大，但该保留的还是顽强的保留着。甚至，海外唐人街的主要语言都是广东话，新加坡、马来西亚的汉语报纸书籍行文也都类似港台的。在海外，西人更多的把广东话 Cantonese 与普通话的 Mandarin 放到同等重要的位置。我想，一国两制绝对是对的，它是香港繁荣的基础，它包容，宏大，每年来自世界各地的游客带来了世界各地的文化，又把这里的文化传播到海外。

香港使用繁体字，如同台湾。中国大陆推行简化字已有几十年，当然，它简化了笔划，易于书写，在一定程度上减少了文盲，带来了经济的发展。但同时，字体在简化的过程中也失去了原来的意义，失去了汉字的美。繁体字是美的，这也是大多数书法家都爱写繁体书的原因。

除简繁分别外，行文也大有不同。我看到一处店铺上写着『偷窃送官究治』感到非常有意思。在大陆，虽然『官』这个词也会常常说起，但当官的自己不用，小民用到时就多含贬义。而这个牌子上，没有用到公安，没有用到警署，没有用到 Police，看起来非常古朴，庄重，而，也许香港人看起来稀松平常。

说起普通话，在香港人眼里看来还是有一定份量的。书店里，就有专门讲香港语言与大陆语言的区别（之所以没说广东话，我觉得这里说的是书面语，而广东话本身更是一种口语方言，跟书面话还是有很大区别的）。在现在，遇到不会普通话的人，有会说英语的会让你说英语，英语也不会的，会让你慢慢说，以便他能听懂。从我的感觉来看，完全没必要因为不会广东话而感到有问题，相反，他们反而会因为不会普通话而觉得不安。而且，我见过一个人，普通话已经说的很不错了（还有英语），还老是跟我说她的普通话不够好。

从书店出来，随便上了一个公交车，竟意想不到的来到了尖沙咀码头，也就是著名的维多利亚岗。实在是太漂亮了，宠辱皆忘，只能这么形容了。陶醉半晌，才想起来拍照片。顺着星光大道轻轻走过，每一步都像是不同的角度不同的景色。去之前也听说过，但对一个住在海边的人来讲本不是怎么向往的。而来到之后，才知道这不一样的感受。烟台也有海，但没这么深，这么蓝。尤其，没有这漂亮的云！

好东西不可多用，美景再好，不能贪图享受。时间有限，看到香港艺术博物馆就在路边，我也不想错过机会。

门票 20 港币，一点都不贵。看了好多古董，走马观花而已。明显的感觉就是这里的瓷器都比较新，都比较细腻有光泽，大约出自唐汉，宋元明清等，而很少有远古时代的物件。书法字画也看了不少，大都一带而过。最使我驻足的，是丰子凯的画，有好几个展厅，我也看了一个来小时。以前也知道他的名字，但作品没什么印象，这次一看，确实非常深刻。作品以漫画为主，草草几笔，却把人物、故事勾勒的非常生动，把主题表现的非常鲜明，看起来令人有一种深切的共鸣。印象最深的是解放战争后，丰回家后看到原先的大宅只剩残垣而作的一幅。有空一定买一本画谱来看，虽与真迹相差太远，但总可以仰读俯思，在忙碌的生活中添一份宁静。

不敢久留，因为答应了朋友去苹果店看看。乘地铁倒没多远，只是去了两家苹果店，均没有最新的 Macpro retina 版，只好做空手而归。

大商场里晚饭没找到合适的地方，有个地方似乎卖北京、上海等各地小吃的，然后对我没吸引力，排队的人也多，我索性吃了麦当劳，华灯初上，站在四楼某天台看维多利亚港。后来，为了不虚



图 11.4: 丰子恺作品之一

此行，还是花了 2.5 元坐了一把天星小轮（Start Ferry），很不错，但维港的夜景没想象中的美。因此，又回到星光大道发了会儿呆以后，便打道回府了。

回到铜锣湾地区，到时代广场，已经快到晚上 10 点关门的时刻了，只是匆匆给儿子买了件玩具。

又到另外一个商场，忘了名字，好像是刚开业或搞什么活动，12: 00 才关门。溜达半天也没看到什么可以买的，最后进了一家叫『诚品』的书店，看了会书。多是中文繁体的，也有部分英文书。看到了英文版的《Steve Jobs》，而中文版的译作《贾伯斯传》。英文版要贵好多，具体价格忘记了。但繁体中文版也比大陆的简体中文版贵好几倍。

最后看到一本台湾出版的繁体中文竖排版《郎咸平說—中國即將面臨的 14 場經濟戰爭》，郎咸平说得不一定对，但是开卷有益嘛，不知道大陆有没有出版，便花了 100 大元买了回来。

第二天，6 点乘地铁赶 8 点班机回京，一切顺利。飞机上香港《星島日報》的信息量还挺大，有各种大小案件，有钓鱼岛，有王立军，有薄熙来，有 MM 公寓……

回到首都北京，虽然空气依然不好，但毕竟是家啊。

另外，附小词一首，是以记：

诉衷情 · 维多利亚港

初至维港，但见天高水阔，风清海蓝。登高一望，码头半壁连海，两岸高楼擎天，直教人流连忘返。
观熙攘旅客，穿梭游船，又有远处青山环黛，绿树含烟。海上清风拂面，天上云舒云卷，风景无边。
至若华灯初上，更有繁星点点，灯火斑斓。走星光大道，乘天星小轮，移步换景，千般妩媚，万种
风情。一时诗情涌动，因有词曰：

高楼两岸都接天，
绿树伴云眠。
一时宠辱皆忘，
信步看远山。

云卷卷，
风清清，
海蓝蓝。
沉吟之处，
即是仙境，
又是人间。

11.15 ClueCon 2014

杜金房/2014.09.28

光阴似箭，转眼，ClueCon 已经到了第十个年头了。

为了防止航班延误，我是提前两天到的。当然，为了省点钱，我住的还是芝加哥青年旅舍——Hosteling International Chicago (HIC)。不得不说，HIC 是我住过的最好的青年旅舍，没有之一。

到达 HIC 后是当天的下午，我凌乱的睡了一晚后，开始准备我在 ClueCon 上要讲的 Demo。是的，今年是我准备最糟糕的一年，可以说，我什么都没有准备。我花了几乎一整天的时间重构了一些关于视频转码的代码，最后，它终于不再那么频繁的 Crash 了。稍微松了一口气后，给 Anthony 发邮件，他回复说他们已经在会场准备了，所以，我可以过去看看。

就在我准备出发的时候发现外面下起了雨，还不小，不过，等了一会，雨就停了。HIC 离 ClueCon 的会场只有三站的地铁路程，因此，很快就到了 ClueCon 的会场。这次，ClueCon 的会场选在了洲际酒店 (InterContinental)，会场 1080i 的双投影，比以前气派了很多。当然，他们为了准备今年的音效，特地准备了好多音频及视频控制设备。

当天下午只是随便聊聊，我也没帮上什么忙，最后，跟他们一起晚餐。就餐的过程中又下起了大雨，冲着我们的小蓬子，别有一番风味。好不容易等到雨停了，他们回去可能继续准备，我便径直回酒店了。时差还没有完全倒过来，我需要好好睡一觉。

接下来的一天，可以说是 ClueCon 的第一天，也可以说是一个非正式的开始。中午开始签到，下午是一个黑客马拉松 (Hackathon)。其它也没什么，就是大家自己在写程序，随便聊天什么的。Anthony 在忙着跟 OpenBTS 的人做一些对接，我则忙着我的视频 Demo。其实，这一天跟 OpenSIPS Sumit 有点小冲突，其它我也挺想参加 OpenSIPs 的活动的。

当天，在会场遇到了 James Body。他是从英国来的，他们公司叫 TruPhone，做一种电话服务，可以在一张 SIM 卡上绑定多个国家或地区的手机号（如英国、美国、香港、德国等），在这些国家之

间打电话只收本地通话费。好像他们从最早就赞助 Anthony 的团队和 ClueCon。今年，他们赞助的是一场 Dangerous Demo 活动。去年的时候，James 见我的 Demo 有趣，特别给我发了一个 Raspberry Pi，今年，则嘱咐我一定来个刺激的 Demo。

其实我真的没有理解 Dangerous Demo 的含义，现在来看，大致就是与 FreeSWITCH 有关，越帅越出人意料越好。我没有别的选择，准备把我的视频 Demo 拆成两半，一部做为 Dangerous Demo，另一半则作为正常的演讲。当然，第一天光顾跟他们聊天，其实进展不大，主要的工作还是在晚上，反正也睡不着。

接下来的一天才是 ClueCon 真正的开始。第一个演讲的当然是 Anthony。他主要讲了 FreeSWITCH 最近的一些更新。当然，最重要的就是 FreeSWITCH 1.4 版的发布，其中包含了对 WebRTC 的支持——SIP over Websocket 以及 Verto。其中，Verto 的演示相当帅，不管是重新刷新网页，还是 FreeSWITCH Crash 后重启，都能自动恢复通话！

后面其他人的演讲大都是应用 FreeSWITCH 的成功案例，我们很高兴看到越来越多的人利用 FreeSWITCH 成功部署了自己的通信平台，大部分都是所谓的『云』。

当然，在听他们讲的同时，我还是在偷偷地准备我的 Demo。晚上散场以后，有一个 Party，我没有参加，太困了。回去睡了一觉，仍然感到不解乏，但我还是得在凌晨两点起来准备我的 Demo。还好，到早上 6 点多的时候我基本都测试通过了，然后匆匆睡了一个小时去会场。

第二天大家的演讲也都很不错，但我不记得许多了，当然有一部分原因是我在下面开小差准备下午演讲的 Presentation。Dangerous Demo 是在下午第一个开始的。Anthony 自然是第一个，他演示了 FreeSWITCH 对立体声的支持，以及崩溃恢复。一切环节近乎完美，不过，也许是因为太完美了，也许是因为评奖的人没有理解，他没有获奖，对此，他还小有些不高兴呢。

轮到我上台的时候，著名的墨菲法则起作用了——我遇到了非常不幸的一瞬——Wifi 突然不好用了！本来，该 Wifi 在过去的两天都没有问题，我也安排好了让 William 帮助我拉几下皮条，配合我拨打几个号，这下都没法演示了。实在是太 Danger 了吧？还好有应急预案，插上有线，IP 地址变了，重新配置 FreeSWITCH 及各终端，我终于在 4 分钟内完成了一个步骤，本来，我是准备了 19 个的。或许因为如此，我最后也跟获奖无缘。当然，或许 Wifi 不出问题我也不可能获奖，但我其实能演示更帅一些的。

值得庆幸的是，我还有机会在下午的最后一场演讲的时候把 Demo 走完，本来嘛，我其实都没准备参加 Dangerous Demo 的。下午的最后一场演讲可以说还是比较成功，除了我在现场演示的时候敲错了一个字符导致 FreeSWITCH 崩溃，没法演示最炫的两个步骤。或许是我学习罗永浩提前打预防针起了作用（It will crash, It must crash, if it crashes, I'll start over），大家对此也并不是很在意。

我演示的内容是视频转码和视频会议，会后，看到还是有些人对此挺感兴趣的。

结束了 Dangerous Demo 和我的演讲，身上一下轻松了许多，晚上就跟大家去酒吧 Happy 了。当然，不得不说今年这个节目不如去年的，去年的时候，有数不清的电子游戏及啤酒，今年，只有很单调的电子游戏，啤酒也没渴多少。后来，有一个哥们非要请我去别的地方吃饭，我只好提前离场了。当然，后来的三文鱼还是挺好吃的。

我们没有吃到很晚，便各自回去睡了。第一次，我睡了一整晚。

ClueCon 最后一天的时候，我感到精力充沛。没了 Demo 和 Presentation 的压力，便有了更多的时间跟不同的人聊聊。虽然在演讲的时候还是觉得有些卡壳，但跟别人交流的时候明显感觉到我的英语比以前说的顺当了。

ClueCon 的最后一场是一场圆桌会议，Anthony、Mike、Travis 以及 William 讲了 FreeSWITCH 开发最近的一些动作及发展规划，大家也都积极参与讨论。ClueCon 胜利闭幕。

总之，这一次 ClueCon 感觉从硬件和软件上都比往届要好。内容除涉及各 FreeSWITCH 成功案例外，还有 OpenSIPS、Kamailio、WebRTC、SIP/VoIP 安全、OpenBTS 等，知识面还是比较广的。

由于接下来的一天是 FreeSWITCH 培训，因而 FreeSWITCH 团队的主要人员都没有走，当晚我们又在酒店举行了一个小型的 Party。有意思的是，我给他们演示了怎么大口喝 Liquor（好像酒精度才 34）后，我俨然成了他们心目中的英雄，Anthony 也多次说：『Seven drink that liquor like water!』。酒后吐真言，话匣子打开，感觉他们都是挺可爱的。当晚，我们聊了一些以前从来都聊不到的一些话题，也听到了不少有趣的故事。酒，还真是好东西啊。当然，我能参加当晚的 Party，也是因为我搬到了洲际酒店，因为，按照规定，我必须在那里住一晚。只是，当晚，我大约 3 点半才回房间睡觉，早上 7 点离开，真有点辜负那么好的房间……

11.16 为什么会是我？

2015-08-04/Seven Du/FreeSWITCH 中文社区

ClueCon 的第一天是个 Hackathon。Hackathon 是 Hack 和 Marathone 组合出来的一个词，就是黑客马拉松。

这一天，除了有一些临时安排的演讲以外，最主要的是写代码游戏。游戏的名字叫 DTMF-u，任何人都可以用任何 API 写关于 DTMF 的游戏或应用。当然，由于该活动早就公布了，还是有一些人有备而来的，现场一个小时的时间真不够做点什么。

Anthony 的游戏是 Tic-Tac-Toe 的一个变种，叫 Tic-Tac-Verto，主要用了 Verto 模块和会议模块。有 9 个人加入 FreeSWITCH 的会议，每个人占据视频会议九宫格上的一个位置。主持人邀请两名志愿者上台，志愿者则轮流从 9 名与会人员中选择会议中的一个成员，并向它提问，问题的答案则由志愿者确认是否同意。询问的题目我大多不懂，总之有一个规则可以确定在某人九宫格的位置画 O 或画 X，如果 O 和 X 连成一条直线或斜线，则论输赢。

游戏不错，只是现场带宽不够好，始终不能完美地显示九宫格。

其它参与者大部分是用的 FreeSWITCH 或 Twilio 的 API 做的 DTMF 应用，做一些呼叫什么的，也有跟移动 App 结合的。他们做得还是很不错的，但是，我没看清。问题就在这里，现场是 1080i 分辨率的 HDMI 大屏幕，所以字体看上去很小，他们也不知道把字体调大一下。所以，不管他们的作品有多么好，对大部分人来说都是——看不见！

我当然也是有备而来，事先就研究了一下 VNC，用 libvncserver 做了一个简单的 VNC server，把 VNC 跟 FreeSWITCH 结合起来。但看不出有什么亮点。所以，我又趁夜里倒时差的时间，用

libvncclient 写了一个应用。模块的名字叫 mod_vnc。实现一个 App 可以连接到一个现有的 VNC Server 上。这样，用一个普通的视频客户端就可以经过 FreeSWITCH 呼叫到这个 VNC Server，并看到画面。

我用的 VNC Server 就是一个 X 服务器，里面运行了一个 xterm，这样，我就可以用 DTMF 控制 xterm 里的输入。我还演示了一个俄罗斯方块游戏、贪吃蛇等。这些游戏都是 Linux 上的，但我可以用 DTMF 转换成方向键进行控制。在大屏幕上演示出来，效果超赞。

最终，我超过了 Anthony 赢得了比赛第一名。奖品是一个崭新的 Chromebook。

所以，我觉得，游戏就是游戏，一定要让大家看见，让大家觉得与众不同，才有机会赢。

好了，ClueCon 的预热已经过去，期待后面更好的开始。

另外一件令人高兴的事就是，我终于有了赞赏功能了。你会赞赏吗？

11.17 为什么又是我？

2015-08-06/Seven Du/FreeSWITCH 中文社区

今年的 Dangerous Demos²³一共有 14 人参加。

说起来好险。上午的时候，不小心电脑触控板上洒了些水。没多少，擦了擦就没当回事。谁知道过了一会鼠标自己乱动，还乱点，就像是被人黑了一样。重启电脑好几次，好不容易禁了 Wifi，依然如故。看来应该不是被黑，而确实是触控板进水的原因。急忙回房间用电吹风猛吹一阵，才好一些。

一上午净折腾电脑了。中午草草吃了点饭，才把环境基本准备好。这才发现我被排在了第一个上场。第一个上场也有好处，可以事先把电脑插到投影上。投影是 1080i 的分辨率，比我电脑平时用的大些，因而字体就显得小了。我趁机把画面和字体调整了一下，以便大家都能看清。

我今年演示的是 mod_x11，通过 X11 协议把视频推到一个远程的 X11 服务器上，每一路视频都会在 X11 服务器上生成一个窗口。这样，当把多路视频发到 X11 服务器上时，X11 服务器就成了一个融屏的桌面。各窗口可以自由拖动。

这个 Demo 虽然与黑客马拉松那天的完全不一样，但看起来还是有些像，因而大家就不怎么觉得兴奋了。所以，我也没指望获奖。

看看还有时间，我又演示了一个解数独的游戏。事先设定一些数独的题目，当视频呼入 FreeSWITCH 以后直接把数独表格显示在视频上面。通过 DTMF，依次输入“行” + “列” + “数字”三个数字填数。刚填了几个时间就差不多到了，我用了一个快捷键，按“#”号让 FreeSWITCH 自己把数独解了。

这两个项目完全没有关联，我强行关联起来了。因而，觉得并不怎么好。后面的选手其实做地也都挺精彩的，看来也都是有备而来。Anthony 换了一种形式照样演示了 Tic-Tac-Verto 游戏。有一个哥们还演示了用 WebRTC 控制机器小车。其它大部分人都按照自己的作品内容写的一些应用。

²³一种黑客游戏，开发者开发个应用或程序，越炫，越超人意料越好。



投票是所有在场的人发短信到一个特定的号码，参与投票的人不算多，我自己投了自己一票，险胜。奖品是一个 Dangerous Demos 的水晶纪念品。上面写着：

Truphone Dangerous Demos
ClueCon - Chicago August 2015
Audience Vote



后来，发奖后，看到有一个人得票好像超过我了，哈哈，晚了。

这一次，我真不知道为什么自己得票最多。

11.18 ClueCon 2015

我的记忆已经不行了，这一年没写。翻译了一个篇别人的文章，比我写得好。更重要的是，里面提到了我的名字。-Seven

上周，Flowroute 参加了 ClueCon 2015 年度盛会。ClueCon 的口号是『源于开发者、用于开发者』(For Developers, By Developers)，这些开发者们都专注于创造有创造性的音视频、即时消息

App 及服务。Flowroute 从很久以前就开始支持 FreeSWITCH 以及 FreeSWITCH 相关的活动，因为这些活动在开发者、技术与创新之间有很好的平衡。今年，也是如此。我们的团队得以就业界很典型的一些问题如如何拥抱 WebRTC、攻击防御以及 API 等与很多开发者进行了交流。

用 API 促进下一波电信创新

API 是电信领域创意的源泉，在整个活动中都洋溢 API 的信息。会上有很多生动的演示，其中一个亮点是如何在 10 分钟内通过短短的 6 行代码在通信通道中加入即时消息支持。考虑到时间、资源及成本的因素，这种神奇的演示即使在两年以前也是很难想象的。另一个独特的案例来自于一个富有盛名的 FreeSWITCH 平台专家 Seven Du。他演示了如何通过 API 创建、修改以及挂断 Channel（通话），如何在会议中上传及演示 PDF、GIF 及 PNG 文件，以及在 FreeSWITCH 会议电话中播放幻灯片等。

防盗打仍然是一个挑战

盗打电话越来越得到大家的关注，不管你是一个消费者，还是一个企业，预防盗打电话仍然是成本很高而且效果非常有限的事情。

会上，通过机器学习来检测盗打是一个共同的话题。盗打检测一直以来就植根于我们服务的核心，我们早就在 Flowroute 服务中建立了自动的控制机制。你能看到，自从我们的用户第一天用我们的平台打电话起，我们就为每个用户创建了专用的外呼配置逻辑，包括已经打了多少分钟，每分钟能打多少电话，以及都打了哪些号码等。我们把这些号码又按一周内的天数及一天内的小时数分段分类。通过分析呼叫数据，我们可以发现违规的呼叫行为。如，如果我们发现短时间内有 10,000 个到古巴的呼叫，但你以前却从来没有往古巴打过电话，那么，我们会禁止这样的呼叫，并给用户发送 Email 告诉他们这种异常的行为。

如果这些呼叫是正常的呼叫，客户只需简单的点击 Email 中一个『解锁』链接，所有的外呼流量都会恢复。不可避免的是，对有些新客户而言，由于我们掌握的数据量不多，确实我们有时会误判。减轻攻击的另一个方法是使用号码白名单和设置最大呼叫频率，这些都可以在 Flowroute 的账户管理界面上直接设置。

WebRTC 的革命

大势所趋。随着一些全球性的运营商开始在 WebRTC 技术上的建模及应用，通信系统都纷纷开始拥抱 WebRTC。通信本身正在聚焦于一个以 WebRTC 为标准的平台，互联网工程任务组 (IETF) 及全球信息网协会 (W3C) 正在能力合作以将其标准化。

在会上，我们也看到很多 WebRTC 方面的兴趣以及创意。Verto 是一个很好的例子，它开发了一个很丰富的信令协议与 FreeSWITCH 交互，并在视频组件方面做了很多努力。另一个 WebRTC 创新是我们看到 ClueCon 使用多流视频会议、发言者检测、以及 3D 立体声会议。我们也跟面向对象的 RTC (ORTC) 的开发者聊过，ORTC 可以使开发和维护连接变得更简单。ORTC 标准支持高级的视频会议技术如分层视频编码 (SVC) 以及联播 (Simulcast) 等技术可以很容易的开发类似『Google Hangouts』之类的应用。

结束的那一天

很显然，使用 API 来提供有创造性的语音及即时消息 App 和服务是通信市场新的方向和核心。现在成了一个很令人兴奋的兴趣点。ClueCon 为我们提供了一个很好的论坛，我们得以看到第一手资料，看到开发者们是如何通过 API 创造应用的，并有幸一瞥未来的市场方向。

时间仓促，翻译里未仔细推敲。大家将就着看，不过瘾的看原文：

<http://blog.flowroute.com/2015/08/12/what-happened-at-cluecon-2015>

11.19 ClueCon 2016

时间过得好快，暖暖的 8 月，又迎来了新一届 ClueCon。

以前一直坐美联航的飞机，这一次，特意试坐了美国航空。周日早上 10 点钟从北京出发。

新飞机，飞机上有电源插座，我可以打开电脑做一些工作。飞机上也有 Wifi，不过比较贵，\$12 可以用两小时，\$17 可以用 4 小时，\$19 可以 Cover 整个飞行时间。也没什么要紧的事，就没有开。

在飞机上，发现我电脑的电池完全罢工了。以前充满电差不多还能撑十几分钟，现在断电就崩溃，比较郁闷。

飞机大约晚了一个小时，到芝加哥周日上午 11 点钟的样子。通关，到酒店差不多是下午 1 点了。这次 ClueCon 在芝加哥瑞士酒店（Swissotel）举行。

去酒店下面吃了点东西。菜单不认识，随便点了一个，不好吃。吃完回酒店睡觉。

一觉醒来，半夜了，国内是上午。发现微信上小伙伴们正在讨论一个线上的问题。花了几个小时研究代码，没搞定，睡觉。

又一觉醒来是早晨。匆匆吃了点早餐，到会场还不晚。见过从意大利过来的老朋友 Giovanni，热情打招呼。ClueCon 第一天是个黑客马拉松（Hack-A-Thon）以及一些自由的活动，也有零星的演讲什么的。Hack-A-Thon 我没有参加，主要有一些家伙用 3D 打印机以及 RaspberryPi 做电话什么的，有些意思。我没什么事，签到，跟大家打完招呼后，去苹果店修电脑。

在密歇根大街上就有一家，走路也不远，十多分钟的样子。检测完毕说要换电池，由于电池跟壳（Top Case）是连在一起的，都要换。\$199，但没有件，所以，要等大约三天的样子。我算了时间还来得及，就让他们进配件，我回 ClueCon 写代码。

由于一直比较忙，我的 ClueCon 主题也一直没有定，PPT 也没写。感觉都有点对不起这千里迢迢。过去的一年，零零散散修了些 Bug，也没有什么重大的东西做出来，因此也没什么好讲的。算了，想了想，还是拿出我写的一部分 XUI 代码，HTTP 以及 JSON API 来讲吧。准备 PPT 过程发现都是坑，因为有些代码是一年甚至两年前写的，年代久远，重新温习了一下，Debug 两天才都跑通，后话暂且不提。

下午美女 Kathlin 送来了 FreeSWITCH Solutions 的 T-Shirt，感觉真正被接纳为 FreeSWITCH Solutions 一员了。



图 11.5: Anthony



图 11.6: Anthony 和 William



图 11.7: Mike



图 11.8: Giovanni、Travis 等

第一天很快就过去了，晚上有其它公司赞助的 Pizza 晚宴，报名比较早的人可以参加，一起吃饭，扯扯淡，就过去了。时差原因，困得要死，回酒店什么都不想干，睡觉。



图 11.9: Pizza Party

周二，也是第二天，Cluecon 正式开始，起床一看都快 9 点了。吃了两块昨晚打包回来的 Pizza，算是早餐了。到达会场，正好 Anthony 开始讲。

Anthony 主要讲得什么是 FreeSWITCH，什么不是 FreeSWITCH 之类的，讲的内容其实我早就知道，但还是不容错过。其中最亮的一点就是演示了用 Microsoft Edge、FireFox Nightly、Safari 和 Chrome 四种浏览器用 WebRTC 加入同一个 FreeSWITCH 视频会议。



图 11.10: WebRTC

当天上午有一个 WebRTC 的圆桌会议，有 WebRTC 委员会的人参加，以及几个微软的负责 WebRTC 方面的哥们远程参加，远程视频会议当然是使用 FreeSWITCH 的 Verto Communicator 了。效果不错。我也被叫在了台上，说了几句，不过感觉发挥不是很好。

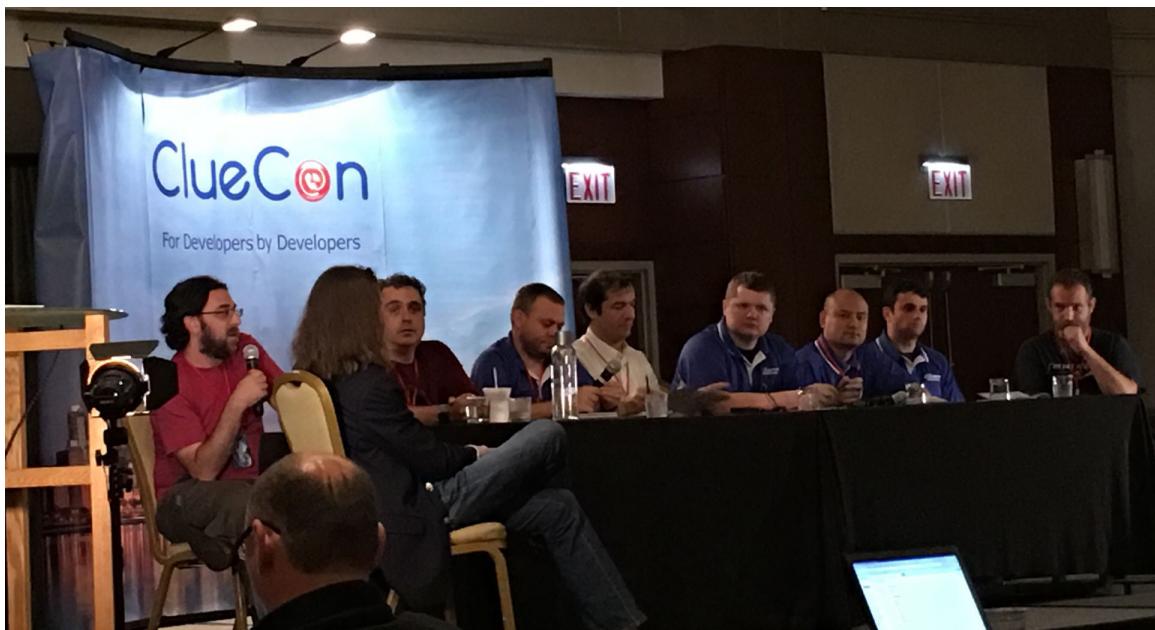


图 11.11: ClueCon Roundtable

其他的演讲也都很好，OpenSIPS、Matrix.org、2600Hz、CGRates ……都有很多新的进展。其它话题不是很吸引我的时候，我就在写我的 PPT。

晚上 ClueCon 请客，大吃一顿烤肉，回来，在会场有个小 Party，就是大家拿个酒瓶子随便找人聊天的那种。聊天的间隙我也不忘打开电脑干点活。10 点钟的样子吧，老戴来了，他飞机有些延误。老戴是我们 FreeSWITCH 中文社区的老朋友，很多年前就做 FreeSWITCH。现在在拉斯维加斯躲计划生育，又生了个美国娃。专程坐飞机过来看我。

晚上把 PPT 基本搞定，才陪老戴瞎聊会。他还要继续工作（他的同事都在国内），我就睡觉。

周三有点小迟到，上午听演讲，写 PPT，间或去大厅外面跟其它人瞎聊会。下午就轮到我演讲了。我就讲 HTTP 和 JSON API，以及我是怎么做的一些 Framework。还可以吧，演讲很顺利，大家还听得挺认真的。我告诉大家 $3+4=7$ ，而 7 (Seven) 是我的名字。并且，剧透了一下我明天要在 Dangerous Demos 上要演示的东西，跟这次演讲相呼应。其实，如果没有 Dangerous Demo 的话，我也就一块都讲完了。剩下那块留到 Dangerous Demo 上获奖。

晚上 ClueCon 还要请我吃饭，大致是看我远道而来没地方吃饭吧，不过我没去，因为约了老戴和另外一个朋友。

李小男是我以前在 Idapted 的同事，现在在加州上学。暑假出来玩正好来芝加哥，好多年不见，一块去 ChinaTown 吃小肥羊。话说，要想吃饱肚子还是得中餐啊。

吃完饭回来，听老戴讲他的创业故事。他真能讲，我都睡过去好几次。后来实在撑不住了，睡



图 11.12: 唐人街

下，他继续“上班”。

周四，又迟到，拜老戴所赐了。他上午睡觉，我还得去开会。上午最后一个环节是 Dangerous Demos。我都准备好了。

Dangerous Demo 其实就是个程序员的游戏，每个人 5 分钟演示个东西，越好玩、越刺激、越超人意料为好。现在 ClueCon 的老人几乎都知道我曾经有一年用一个电路板把主持人的手差点烫伤，从那以后我的每一次表演都是最 Dangerous，每年都得第一，Anthony 也就只得第二。他打趣说，我的 Demo 不 Dangerous，it always work :D。

今年我是这么做的。一上台，就告诉大家，我的电脑电池坏了，因此，上台只能先启动电脑，时间有限，导致演示更加 Dangerous。这是第一步，要让大家听到 Dangerous 这个字，很关键。

我演示的是用 Blocks 做 IVR，就是可视化编程的一种，用鼠标拖动一些组件，生成 FreeSWITCH 可以执行的 Lua 和 Javascript 程序。其实前一天我已经给大家演示过怎么拖动了，并用它做了个 $3+4=7$ ，为今天的演示更生动埋下伏笔。

接着，我演示了一个程序，就是写一个 IVR，有电话呼入后，用 TTS 播放 Hello ClueCon。

接着，做真正的 IVR，熟练的把不同的组件块拖出来一个 IVR。IVR 的内容是，欢迎致电 ClueCon，请按一个数字。如果按 1，就用 TTS 说“您按的是 1”，如果按 2 就说“您按的是 2”。实际上就是程序语言里的switch.. case语句。大家都知道switch.. case有default，就是，如果按的即不是1，又不是2，比如是x，就会说“x is too dangerous”。

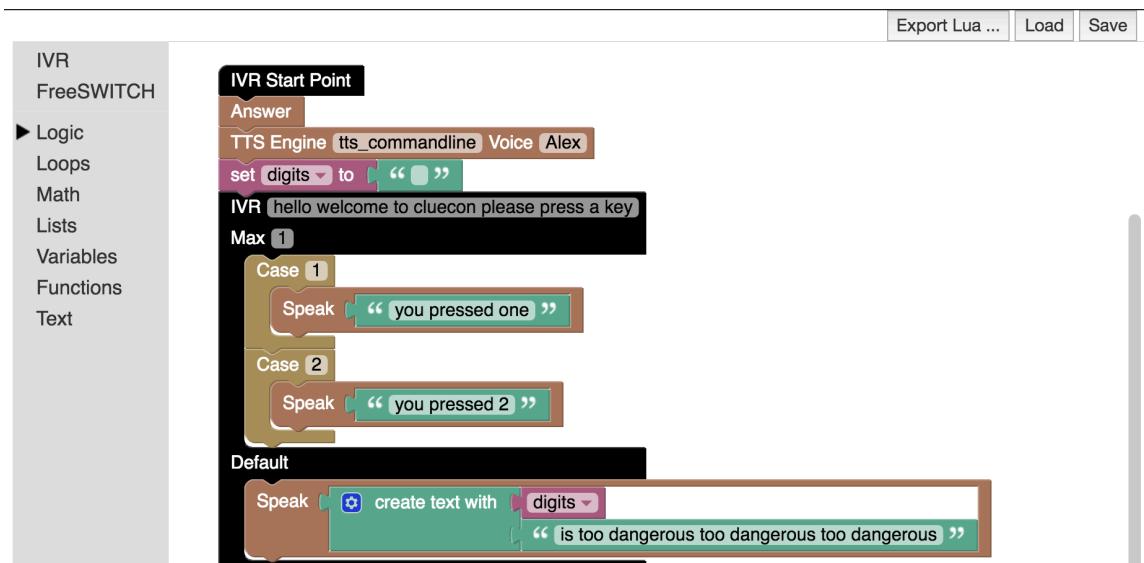


图 11.13: 用可视化编程拖出一个 IVR

最后演示的效果是：“7 is too dangerous！”为了增加效果，重要的事情说三遍：

Seven is too dangerous, too dangerous, too dangerous!

好了，最后，大家都记住了**SEVEN**，我的名字，是最危险的。

生成的 Lua 代码如下：

```

session:answer()
tts_engine = "tts_commandline"
tts_voice = "Alex"
session:set_tts_params("tts_commandline", "Alex")
session:setVariable("tts_engine", tts_engine)
session:setVariable("tts_voice", tts_voice)
digits = ''
digits = session:read(1, 1, "say:hello welcome to cluecon please press a key", 5000, "#")

if (digits == "1") then
    session:speak('you pressed one')
elseif (digits == "2") then
    session:speak('you pressed 2')
else
    session:speak(digits .. ' is too dangerous too dangerous too dangerous')
end

```

当然，它也可以生成 js 代码，只是略复杂些：

```

var digits;
var tts_engine = "tts_commandline"; var tts_voice = "Ting-Ting";
session.read = function(min, max, sound, timeout, terminator) {
    var dtmf = "";
    if (sound.indexOf(".") >= 0 || sound.indexOf("/") >= 0 || sound.indexOf("\\" >= 0) {
        session.streamFile(sound, function(s, type, obj, arg) {
            dtmf = obj.digit;
            return(false);
        });
    } else {
        session.speak(tts_engine, tts_voice, sound, function(s, type, obj, arg) {
            dtmf = obj.digit;
            return(false);
        });
    }
    if (max == 1) return dtmf;
    var digits = session.getDigits(max - 1, terminator, timeout);
    return dtmf + digits;
}
session.answer();
tts_engine = "tts_commandline"; tts_voice = "Alex";
session.setVariable("tts_engine", "tts_commandline");
session.setVariable("tts_voice", "Alex");
digits = '';

```

```

digits = session.read(1, 1, "hello welcome to cluecon please press a key", 5000, "#");
switch (digits) {
    case "1":   session.speak(tts_engine, tts_voice, 'you pressed one');
    break;
    case "2":   session.speak(tts_engine, tts_voice, 'you pressed 2');
    break;
default:
    session.speak(tts_engine, tts_voice, (String(digits) + String('is too dangerous too dangerous too dangerous')));
}

```

其他人的演示也都不错，但他们没有我这么能造。比如他们的演示太过复杂，屏幕上的字太小，让人看不清，最后的结果趋于平淡，在大家心中的印象就不是那么深。

这次的评分也很奇葩，主持人说一个名字，让大家喊，如果大家的声音越大，说明它的演示就越好。结果当然大家都知道了，说到我的名字时大家声音最大！

这次的奖品没看清具体是什么，在4种里随便选，我果断选了一本书—《Mastering FreeSWITCH》。这本书是两年前谋划的，本来是我主笔，但我瞎忙，这事就没做好，后来 Giovanni（也就是我第一天碰见的那个哥们，也是我们北京沙龙邀请的嘉宾）主笔了，因此，我的名字就没出现在作者名单里，而是在贡献者列表里，小有遗憾。

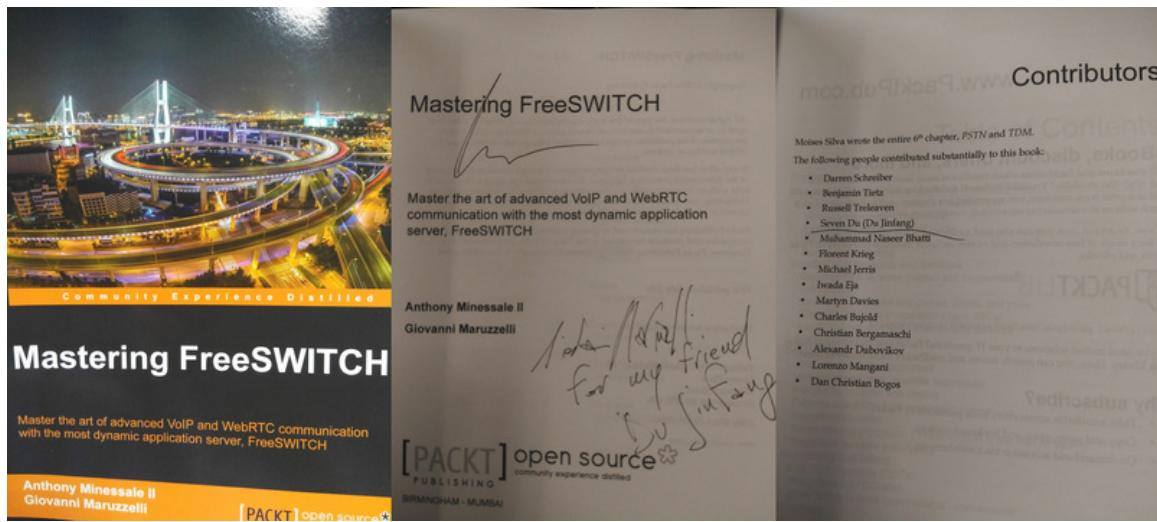


图 11.14: Mastering FreeSWITCH

下午，看到我的电池到了，抓紧跑去苹果店修，结果被告知需要 24 小时才能换好。我了个去，24 小时我得在飞机上了。让他们抓紧。他们说不是我想象的那么简单（其实我想不出来有多困难，就是换一下零件）。没办法，只好将电脑留在那里。他们说尽量在周五上午 10 点前修好。

下午收到邮件，说还有一个东西（Display Assembly, 15-inch）坏了，要换，600 多刀，我抓紧打电话过去说不换，600 多刀啊，不如买个新的了。他们说弄错了，他们已知那个硬件有问题，免费给换。晕死啊，有问题有问题有问题，我用了好几年…

该演示的都演示完了，又没了电脑，我感觉轻松很多（本来是想抓紧为公司干一些活的）。

下午又听了些演讲，在大厅聊聊天，ClueCon 就结束了。

晚上 ClueCon 又请客，跟他们一块去吃了号称芝加哥最好的鸡翅。吃完后还有一个 Party，由于某一年我喝酒表现突出，他们每年都会给我准备那种 Liquor，其实我也不知道牌子，说实话，我还是更喜欢喝点啤酒。

其中有好几个人都说喜欢我的 Demo，还有一个人说很喜欢我巧妙的植入了自己的名字，看来，我这个小伎俩有人识破了。

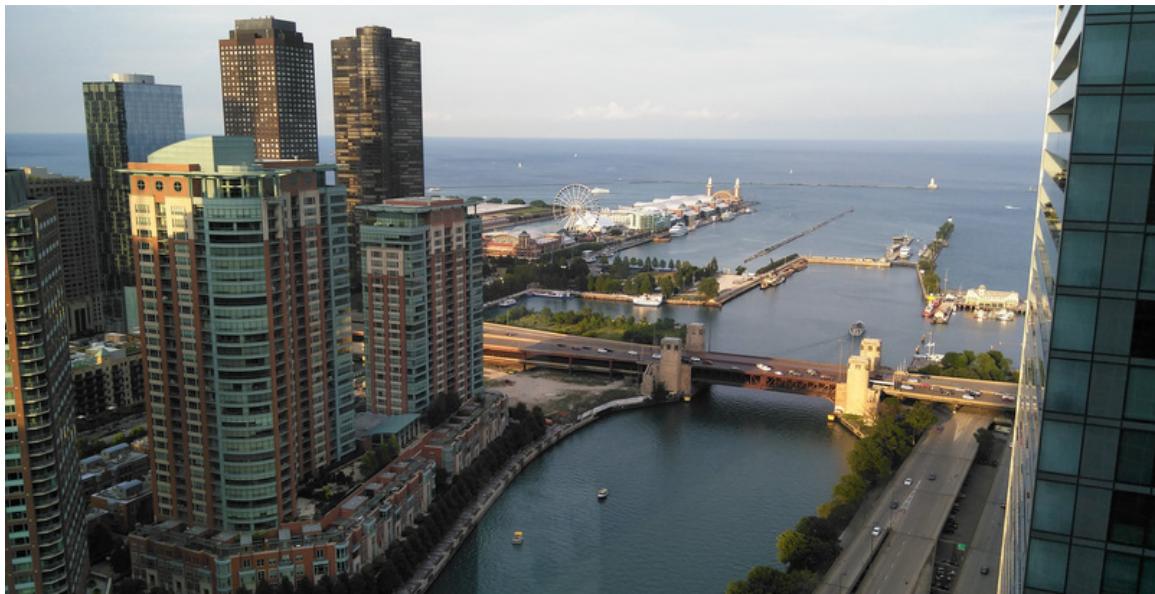


图 11.15: 酒店窗外

晚上回来见老戴不在，发信息说他在楼下抽烟，马上上来。上来后就又讲故事，讲到困死。我睡觉，他继续“工作”。

周五，有 FreeSWITCH 和 OpenSIPS 培训，两场同时在一个屋里，中间加个隔断。

我一觉醒来一看都 10 点了，直接就去苹果店拿电脑。却又发现多出来 400 多刀换了个内存相关的零件，我说，啊我都没收到邮件确认，怎么说换就换了呢，Blahblah blah，最后他们说，好吧，这钱我们出吧，我还是需要付\$199。

最后的价格是\$199 +\$39(人工) + 税，具体数字不记得了，反正是这么算的。

一次成型的电脑外壳都是全新的，显示器也是全新的（原来的脏了都擦不干净）…我想不出他们还有什么没换的，除了硬盘。

好吧，拿到电脑，我又可以工作了。当然，也有不好的地方，我本来希望今年秋季等新的 Mac Pro 出了再买新的，可能实现不了了。

回到酒店已经 11 点多了，收拾东西退房。老戴说他查到有个地方吃烤鸭，我说附近有个老四川，

他说咱俩这不说的一个地儿么，走起。烤鸭还是不错的，其它的酸菜鱼夫妻肺片就不正宗了，但中国味怎么也比老美的烂菜叶拌鸡胸好吃。

下午再回到酒店，找到 Mike，让他帮我调好一个模块的编译问题。时间不多了，一一告别，跟老戴一起坐地铁去机场。

地铁还是挺便宜的，从奥黑尔国际机场到市区是\$5，从市区到机场是一人\$3。老戴的飞机比我的晚，把我送到国际安检，依依惜别。

很遗憾，没有倒出时间给大家捎一星半点东西。

临登机的时候，惊奇的发现 Ethan 站在我面前。他也是我在 Idapted 时的同事，去美国转圈，跟我同一班飞机回国。简单叙叙旧，匆匆登机，相约飞机上继续聊。

上飞机先睡了一觉，跟 Ethan 聊半天，由于我们太兴奋，其它人都有意见了，我们才各自回到座位。

毫无睡意，拿出电脑写下这些文字。新电池！

第 VI 部分 附录

附录 A 通道变量

Channel Variables（通道变量）可以控制业务逻辑、影响呼叫流程。本章列举了一些常用的变量名称与显示的对应关系，以供读者参考：

`info` App 可以显示变量的值。有些变量在显示时会有『`variable_`』前缀，这在 Dialplan 中实际引用时要去掉；另外一些变量显示时会有『`Channel-`』前缀，引用时也要是行相应变换。下表列举了一份不太完整的对应关系：

显示名	变量名	说明
Channel-State	state	当前状态
Channel-State-Number	state_number	状态整数值
Channel-Name	channel_name	名称
Unique-ID	uuid	标志该信道的唯一 ID
Call-Direction	direction	方向，来话 (Inbound) 还是去话 (Outbound)
Answer-State	state	应答状态
Channel-Read-Codec-Name	read_codec	输入语音编码
Channel-Read-Codec-Rate	read_rate	输入比特率
Channel-Write-Codec-Name	write_codec	输出语音编码
Channel-Write-Codec-Rate	write_rate	输出比特率
Caller-Username	username	来话用户名
Caller-Dialplan	dialplan	拨号计划，如 XML、lua、ENUM 等
Caller-Caller-ID-Name	caller_id_name	主叫名称
Caller-Caller-ID-Number	caller_id_number	主叫号码
Caller-ANI-II	aniii	ANIII
Caller-Network-Addr	network_addr	网络地址

显示名	变量名	说明
Caller-Destination-Number	destination_number	被叫号码
Caller-Unique-ID	uuid	UUID
Caller-Source	source	呼叫源
Caller-Context	context	Context
Caller-RDNIS	rdnis	RDNIS
Caller-Channel-Name	channel_name	通道名称
Caller-Profile-Index	profile_index	Profile Index
Caller-Channel-Created-Time	created_time	创建时间
Caller-Channel-Answered-Time	answered_time	应答时间
Caller-Channel-Hangup-Time	hangup_time	挂机时间
Caller-Channel-Transfer-Time	transfer_time	转移时间
Caller-Screen-Bit	screen_bit	
Caller-Privacy-Hide-Name	privacy_hide_name	
Caller-Privacy-Hide-Number	privacy_hide_number	
variable_sip_received_ip	sip_received_ip	
variable_sip_received_port	sip_received_port	
variable_sip_authorized	sip_authorized	
variable_sip_mailbox	sip_mailbox	
variable_sip_auth_username	sip_auth_username	
variable_sip_auth_realm	sip_auth_realm	
variable_mailbox	mailbox	
variable_user_name	user_name	
variable_domain_name	domain_name	
variable_record_stereo	record_stereo	
variable_accountcode	accountcode	
variable_user_context	user_context	
variable_effective_caller_id_name	effective_caller_id_name	
id_name		

显示名	变量名	说明
variable_effective_caller_id_number	effective_caller_id_number	
variable_caller_domain	caller_domain	
variable_sip_from_user	sip_from_user	
variable_sip_from_uri	sip_from_uri	
variable_sip_from_host	sip_from_host	
variable_sip_from_user_stripped	sip_from_user_stripped	
variable_sip_from_tag	sip_from_tag	
variable_sofia_profile_name	sofia_profile_name	
variable_sofia_profile_domain_name	sofia_profile_domain_name	
variable_sip_req_params	sip_req_params	
variable_sip_req_user	sip_req_user	
variable_sip_req_uri	sip_req_uri	
variable_sip_req_host	sip_req_host	
variable_sip_to_params	sip_to_params	
variable_sip_to_user	sip_to_user	
variable_sip_to_uri	sip_to_uri	
variable_sip_to_host	sip_to_host	
variable_sip_contact_params	sip_contact_params	
variable_sip_contact_user	sip_contact_user	
variable_sip_contact_port	sip_contact_port	
variable_sip_contact_uri	sip_contact_uri	
variable_sip_contact_host	sip_contact_host	
variable_sip_invite_domain	sip_invite_domain	
variable_channel_name	channel_name	
variable_sip_call_id	sip_call_id	

显示名	变量名	说明
variable_sip_user_agent	sip_user_agent	
variable_sip_via_host	sip_via_host	
variable_sip_via_port	sip_via_port	
variable_sip_via_rport	sip_via_rport	
variable_presence_id	presence_id	
variable_sip_h_P-Key-Flags	sip_h_p-key-flags	
variable_switch_r_sdP	switch_r_sdP	
variable_remote_media_ip	remote_media_ip	
variable_remote_media_port	remote_media_port	
variable_write_codec	write_codec	
variable_write_rate	write_rate	
variable_endpoint_disposition	endpoint_disposition	
variable_dialed_ext	dialed_ext	
variable_transfer_ringback	transfer_ringback	
variable_call_timeout	call_timeout	
variable_hangup_after_bridge	hangup_after_bridge	
variable_continue_on_fail	continue_on_fail	
variable_dialed_user	dialed_user	
variable_dialed_domain	dialed_domain	
variable_sip_redirect_contact_user_0	sip_redirect_contact_user_0	
variable_sip_redirect_contact_host_0	sip_redirect_contact_host_0	
variable_sip_h_Referred-By	sip_h_referred-by	
variable_sip_refer_to	sip_refer_to	
variable_max_forwards	max_forwards	
variable_originate_disposition	originate_disposition	
variable_read_codec	read_codec	

显示名	变量名	说明
variable_read_rate	read_rate	
variable_open	open	
variable_use_profile	use_profile	
variable_current_application	current_application	

详见: <https://freeswitch.org/confluence/display/FREESWITCH/Channel+Variables>。

附录 B FreeSWITCH 架构详图

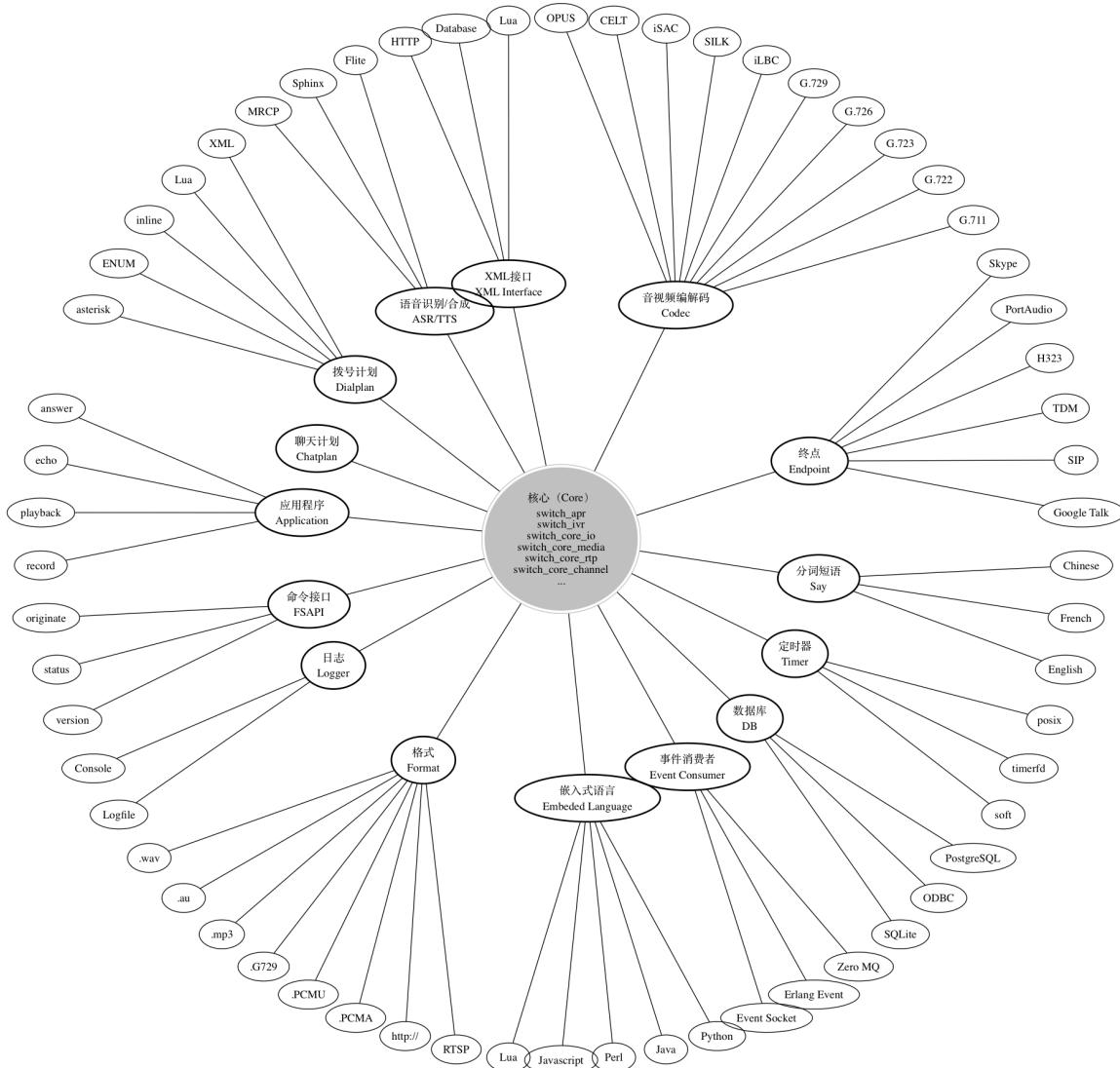


图 2.1: FreeSWITCH 架构详图

写在最后

本书将持续更新，这就是电子版的好处…

如果你对书中的内容和章节安排等有什么意见或建议，欢迎与我联系。如果你建议的内容适合放在本书里，我会考虑写进去；如果不适合放到本书中，我也会考虑写其它主题的书。

如果你的公司想在本书中植入广告或者赤裸裸地做广告，也欢迎与我们联系。

电子邮件：info@x-y-t.cn。

作者简介

杜金房 (网名: Seven) 资深网络通信技术专家，在网络通信领域耕耘近 15 年，精通 VoIP、SIP 和 FreeSWITCH 等各种网络协议和技术，经验十分丰富。有超过 7 年的 FreeSWITCH 应用和开发经验，不仅为国内大型通信服务厂商提供技术支持和解决方案，而且客户还遍及美国、印度等海外国家。

FreeSWITCH-CN 中文社区创始人兼执行主席，被誉为国内 FreeSWITCH 领域的『第一人』；在 FreeSWITCH 开源社区非常活跃，不仅经常为开源社区提交补丁和新功能、新特性，而且还开发了很多外围模块和外围软件；此外，他经常在 FreeSWITCH 的 Wiki 上分享自己的使用心得和经验、在 FreeSWITCH IRC、QQ 及微信群中热心回答网友提问，并不定期在国内不同城市举行 FreeSWITCH 技术培训；自 2011 年起每年都应邀参加在美国芝加哥举办的 ClueCon 大会，并发表主题演讲。

此外，他还精通 C、Erlang、Ruby、Lua 等语言相关的技术。

著有《FreeSWITCH 权威指南》，2014 年出版。

创办了[北京信悦通科技有限公司](#)和[烟台小樱桃网络科技有限公司](#)，提供 FreeSWITCH 培训和商业技术支持服务。

版权声明

本书版权归作者所有，任何人未经书面授权均不得分发此书。

本书电子版仅在 SelfStore (<https://selfstore.io/~dujinfang>) 上发布。如果您不小心从其它渠道获得本书，请删除您的版本并到 SelfStore 上购买正版。

本书纸质版仅随电子版赠送。SelfStore 支持买家自由定价。如果您购买电子书时支付超过一定金额，可以通过电子邮件联系我们赠送一本精美打印的纸质书。

关于 SelfStore：就在本书即将付印之时，SelfStore 宣布关闭。笔者实感遗憾。如果届时 SelfStore 真的关闭，我们将会寻找其它方式发布本书。关于 SelfStore 的声明见这里：

<http://blog.selfstore.io/2016/08/22/selfstore-will-be-closed/>

广告

关于广告的广告

请允许我在本书中发布广告。广告合作联系邮箱：info@x-y-t.cn。

FreeSWITCH 第五届开发者沙龙将于 2016 年 8 月 27 日在京举行

<http://www.freeswitch.org.cn/2016.html>

FreeSWITCH 培训 2016 夏季班（北京站）将于 2016 年 8 月 28-30 日在京举行

<http://x-y-t.cn/fst1608.html>

烟台小樱桃网络科技有限公司提供商业 FreeSWITCH 及 OpenSIPS 技术支持

网址：<http://x-y-t.cn> 邮箱：info@x-y-t.cn

烟台小樱桃网络科技有限公司是潮流网络（GrandStream）山东总代理

深圳市潮流网络技术有限公司（Grandstream）是全球知名的统一通讯和整体解决方案厂商和全球 TOP3 VoIP 终端供应商，是国内 IMS、统一通信、呼叫中心、调度市场、视频监控的主要 SIP 终端供货厂商，成功案例包括京东、网易、凡客诚品、平安、中集、东航、国家电网、中石化、中石油、

外交部、中移动、中电信等等，公司是中国三大运营商 IMS 市场主要核心合作 SIP 终端厂商，入围中国电信集团 IMS SIP 电话短名单，持续配合运营商研究院 SIP 硬终端的核心业务及协议定制，参与及进入中电信、中移动多个省试点应用和产品供应厂商。在全球与渠道及增值合作伙伴成功服务将近千万终端用户，包括全球主流运营商数十万套终端以及美国共和党和美国民主党数万套 SIP 电话和网关等大型成功部署案例，连续 10 年保持近 50% 年复合增长率。

烟台小樱桃网络科技有限公司，是潮流全线产品在山东省的总代理。

联系方式：邮箱：info@x-y-t.cn 电话 0535-6753997

FreeSWITCH 相关图书

《FreeSWITCH 文集》收集了一些 FreeSWITCH 文章，相比其它 FreeSWITCH 书来说，技术内容比较少，便于非技术人员快速了解 FreeSWITCH。

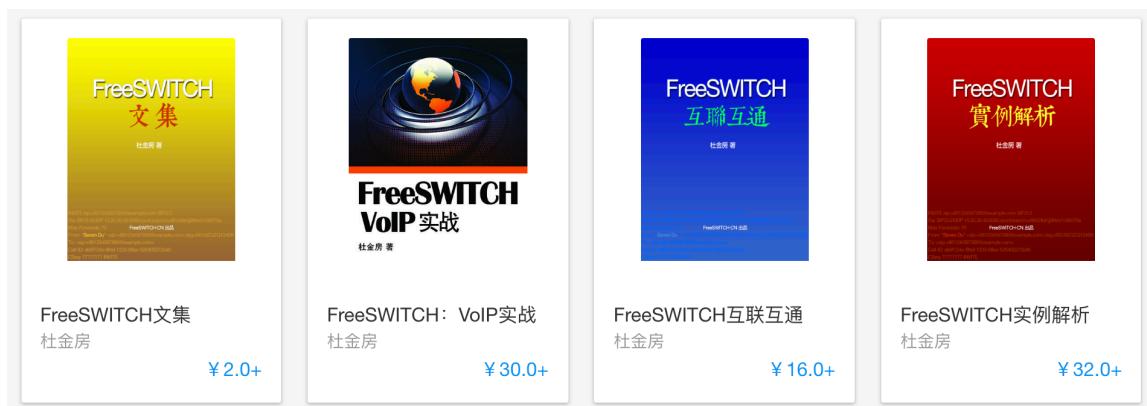
《FreeSWITCH 互联互通》主要收集了一些互联互通的例子。书中有些例子来自《FreeSWITCH 权威指南》。

《FreeSWITCH 实例解析》收集了一些如何使用 FreeSWITCH 的实际例子，方便读者参考。书中有些内容来自《FreeSWITCH 权威指南》。

《FreeSWITCH 实战》是《FreeSWITCH 权威指南》的前身，不再更新，但该书有其历史意义。

以上图书均为电子书，可在 SelfStore (<https://selfstore.io/~dujinfang>) 上购买。SelfStore 支持买家自由定价。如果您购买电子书时支付超过一定金额，可以联系通过电子邮件联系我们赠送一本精美打印的纸质书。

《FreeSWITCH 权威指南》是正式出版的纸质书，纸质版和电子版均可在以下网站购买：<http://book.dujinfang.com>。





作者简介



杜金房（网名：Sevan Du）资深网络通信技术专家，在各种通信领域耕耘15年，精通VoIP、SIP和FreeSWITCH等多种网络协议和技术。经验十分丰富。有超过6年的FreeSWITCH应用和开发经验，不仅为许多公司提供过FreeSWITCH的解决方案，而且多客户及项目，如老卖家家、FreeSWITCH-CNP中文社区创始人兼执行主席，被誉为国内FreeSWITCH领域的“第一人”。目前在FreeSWITCH开源社区工作，不仅贡献了大量代码，还提升了FreeSWITCH的可移植性，还开发了更多外围脚本和框架软件。此外，他经常在FreeSWITCH(Walk)上享受自己独特的心得和经验，直至FreeSWITCH IRC群(QQ群)成立，他都是群主之一。他是中国最早从事FreeSWITCH的研究者，2011年，2012年RSO大会受邀参加美国芝加哥举办的CeBit大会，并发表主题演讲。此外，他还精通与C、Erlang、Ruby、Lua等语言相关的技术。



FreeSWITCH 权威指南

FreeSWITCH: The Definitive Guide

FreeSWITCH是我2006年的一个梦想，到现在为止已经近10年了。它从最初的一个由5个文件组成的概念验证程序已经成长为一个拥有4000个核心代码，另外还有其他300万行以下开源核心的分布式电话系统。感谢大家，因为有了你们的贡献，FreeSWITCH才能发展到今天这个地步。感谢那些帮助FreeSWITCH发展的志愿者们，是你们默默的耕耘才让FreeSWITCH更加壮大起来。我感谢本书的合作者加入我的FreeSWITCH中文社区，与所有FreeSWITCH爱好者一起成长，并一起见证我们迈向下一个里程碑，迎接自由电话的巨大、成功。

我在2006年第一次遇到Sevan，从那时起一直到现在，他做了大量的工作——他几乎学习了所有的FreeSWITCH相关的技术，并贡献了很多的bug报告及修复代码，甚至都在为了写这本书而对FreeSWITCH进行翻译，还贡献了至少5个bug。没有他，我们可能永远不会有这么大的中国读者群体。——Anthony Minasale || FreeSWITCH之父

—— Jonathan Palley | FreeSWITCH CTO

ISBN: 978-7-111-56629-6
定价: 65.00元
出版时间: 2013-03-01
印次: 2013-03-01
页数: 600
作者: 杜金房 张令芳
译者: 杜金房 张令芳
出版社: 机械工业出版社

FreeSWITCH 权威指南

杜金房 张令芳 著
机械工业出版社



FreeSWITCH 权威指南

杜金房 张令芳 著

内容简介

FreeSWITCH是世界上的一个平台，种植性最好的，开箱即用的，多层次的软交换系统。本书是FreeSWITCH领域最具权威的著作之一。在这本书面前，FreeSWITCH无秘密！

由中国的FreeSWITCH领航者“第一人”，全球FreeSWITCH开发者及知名专家、FreeSWITCH-CNP中文社区创始人执行主席Sevan Du编写，FreeSWITCH之父布雷恩·米尔斯极尽全力，通过最直接的万能指南，从配置、部署、操作、管理、各种功能模块、调优与优化和安全、系统的阅读与操作，系统的故障恢复，详细的开发，甚至取代内核的二次开发和新的实践原理，深入探讨了FreeSWITCH的二次开发、各种实践案例、多人协作会议、话单计费等基本的配置实例和生产环境中的应用实践，从单个的FreeSWITCH应用到多个FreeSWITCH集群，从跨本开发到基于宿代的二次开发，各种实践应用都有。很多案例中还提供了可以拿来直接使用。



机械工业出版社



机械工业出版社

THIS PAGE INTENTIONALLY LEFT BLANK.



鬻王 宝金卦 卦受面桂
斐 高 茲 芮 宾 : 鼎鑄玉責