

FreeSWITCH

互聯互通

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 互联互通

杜金房

版权所有，侵权必究

图书不在版编目 (NCIP) 数据

FreeSWITCH 互联互通 / 杜金房 著 / 2015.7

ISBN 7-DU-777777-1

FreeSWITCH，连接一切！

FreeSWITCH 互联互通

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 3 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:)。

有很多读者都说喜欢读电子书。电子书有很多优势，如出版周期短，可以随时更新，阅读过程中也很方便搜索等。为此，我们摘录了《指南》中的一部分与其他系统和设备互联互通的内容，作一次电子书的尝试，以飨读者。

电子书没有篇幅限制，所以，有些内容我们就可以写得深入一些。以解部分读者感觉《指南》内容太多，有些却不够深入之憾。

本书的大部分内容来自《指南》，但也有一些更新。如果你读过《指南》，大可不用读此书，但如果你经常需要跟不同的系统对接，本书肯定是一本很好的参考书。无论如何，如果你喜欢FreeSWITCH，多读一本又何妨呢？

本书内容会不断更新，详细请参阅本书的网站：<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 左右互搏	8
1 双机对接	9
2 汇接	11
3 双归属	14
4 长途局	16
5 FreeSWITCH 作为 PBX	17
5.1 普通的 PBX 设置	17
5.2 DID	19
5.3 使用 PBX 上的网关呼出	24
II 武林大会	27
6 连接 SIP 软电话	28
6.1 X-Lite	28
6.2 Doubango	29
6.3 拨打测试	30

7 连接硬件 SIP 话机	32
7.1 Yealink	32
7.2 GrandStream	32
8 对接 IMS	34
8.1 网关配置	34
8.2 通过 IMS 呼出	35
8.3 通过 IMS 呼入	36
8.4 其他问题	37
9 连接模拟话机和模拟中继线	38
9.1 基本概念	38
9.2 拓扑结构	39
9.3 通过 Grandstream 网关连接模拟话机	41
9.4 通过迅时网关连接模拟话机和模拟中继线	42
9.4.1 配置 FXS 口	42
9.4.2 拨号规则	43
9.4.3 配置 FXO 连接外线	44
9.4.4 配置路由	45
9.4.5 其他	47
10 通过 E1 线路对接	49
10.1 配置 FS1	49
10.1.1 背景资料	49
10.1.2 安装板卡、操作系统和 FreeSWITCH	50
10.1.3 安装及配置 Wanpipe 驱动	51
10.1.4 安装和配置 ISDN 协议库	58
10.1.5 安装和配置 mod_freetdm	59
10.1.6 配置电话路由	62
10.2 配置 E1 网关设备	63
10.2.1 添加 PRI 中继	64

10.2.2 添加 SIP 中继	64
10.2.3 添加路由	66
10.3 配置 FS2	67
10.3.1 拨打测试	68
10.3.2 指定中继拨打	69
10.4 对接其他厂家的 E1 网关	70
10.4.1 ISDN 设置	70
10.4.2 路由设置	72
11 使用 GSM 网关连接 PSTN	73
11.1 通过网关呼出	73
11.2 通过网关呼入	74
12 对接 Asterisk	78
12.1 从 FreeSWITCH 呼叫 Asterisk	78
12.2 从 Asterisk 上呼叫 FreeSWITCH	79
12.3 其他	80
13 使用 H.323 协议对接	81
13.1 mod_h323	81
13.1.1 安装	81
13.1.2 配置和加载模块	83
13.1.3 呼叫测试	84
13.2 mod_opal	84
13.3 其他	85
III 闭关修炼	87
14 DTMF	88
14.1 带内 DTMF	88
14.2 RFC2833	89

14.3 SIP INFO	90
14.4 使用 FreeSWITCH 检测声音文件中的 DTMF 信息	91
15 ACL	94
16 多租户	96
16.1 Domain 简介	96
16.2 配置与实例	98
16.3 进阶	101
16.4 其它	101
IV 歪门邪道	103
17 号码连选	104
17.1 注册到运营商服务器	104
17.2 通过单个号码呼出	105
17.3 使用随机数做号码连选	105
17.4 使用 mod_distributor 进行连选	106
17.5 其它	107
18 在 Web 浏览器中打电话	109
18.1 Flash	109
18.2 WebRTC	113
18.2.1 JsSIP	114
18.2.2 SIP.js	115
18.2.3 sipML5	115
18.2.4 Verto	117
19 通过 OpenSIPS 服务与其它系统对接	118

V 附录	121
1 FreeSWITCH 开源社区指南	122
1.1 中文社区	122
1.2 英文社区	123
1.3 其它	124
2 关于 FreeSWITCH 常用术语翻译的意见	125
3 FreeSWITCH 背后的故事	127
4 FreeSWITCH 与 Asterisk	130
5 FreeSWITCH 的历史	135
6 模块列表	142
6.1 applications	142
6.2 asr_tts	145
6.3 codecs:	145
6.4 dialplans	146
6.5 directories	146
6.6 endpoints	146
6.7 event_handlers	147
6.8 formats	147
6.9 languages	148
6.10 loggers	148
6.11 say	148
6.12 timers	149
6.13 xml_int	149
7 Sofia Profile 参数参考	150
8 通道变量	164

写在最后	171
作者简介	172
版权声明	173
广告	174
关于广告的广告	174
FreeSWITCH 第五届开发者沙龙将于 2016 年 8 月 27 日在京举行	174
FreeSWITCH 培训 2016 夏季班（北京站）将于 2016 年 8 月 28-30 日在京举行	174
烟台小樱桃网络科技有限公司提供商业 FreeSWITCH 及 OpenSIPS 技术支持	174
烟台小樱桃网络科技有限公司是潮流网络（GrandStream）山东总代理	174
FreeSWITCH 相关图书	175
	177

第 I 部分 左右互搏

使用 FreeSWITCH，总免不了跟其它系统对接。不过，所谓「知己知彼，百战不殆」。在「知彼」之前，我们先要做到「知己」。在跟其它系统对接之前，我们先来看一下 FreeSWITCH 跟 FreeSWITCH 是如何对接的。通过研究 FreeSWITCH 与 FreeSWITCH 对接，我们不仅能即当客户端，又当服务器，即当甲方，又当乙方，还能享受周伯通那种双手**左右互博**的乐趣，全面了解 FreeSWITCH 在各种场景下的应用。不亦说乎？

第一章 双机对接

现在，我们先来看看简单的两个 FreeSWITCH 之间的对接。

再小的系统也需要有规划和设计。在做实验之前，我们需要先对系统进行一些简单的分析和规划。

我们现在有两台 FreeSWITCH 主机，分别为 A 和 B，IP 地址分别为 192.168.1.A 和 192.168.1.B。每台机器均使用默认配置，也就是说在每台机器上 1000 ~ 1019 这 20 个号码之间可以互打电话。位于同一机器上的用户称为「本地用户」，如果需要与其他机器上的用户通信，则其他机器上的用户就称为「外地用户¹」。

如果我们需要两台机器之间的用户可以互拨，最需要解决的问题不是技术上如何配置，而是一个逻辑问题，即我们需要先想一种拨号方案。例如：如果 A 上的 1000 想拨打 B 上的 1000，则 B 上的 1000 相对于 A 上的 1000 来说就是外地用户。就一般的企业 PBX 而言，一般拨打外地用户就需要加一个特殊的号码，比方说「0」。这样，「0」就称为「出局字冠²」。

基于上面的讨论，为了完成该实验，我们规定：不管是 A 的用户还是 B 上的用户，拨打外网用户均需要在实际的电话号码前加拨「0」。

计划做好了，我们可以动手工作了。

我们需要先做 Dialplan。在 A 机上，把以下 Dialplan 片断加到 conf/dialplan/default.xml 中：

```
<extension name="B">
<condition field="destination_number" expression="^0(.*)$">
    <action application="bridge" data="sofia/external/sip:$1@192.168.1.B:5080"/>
</condition>
</extension>
```

其中，正则表达式「^0(.*)\$」表示匹配所有以 0 开头的被叫号码，匹配完成后，括号中的匹配结果会被绑定到变量 \$1 中。因此，如果 A 上的用户呼叫 01000，则 \$1 的值就是 1000，bridge 是一个 App，它的参数就变成「sofia/external/sip:1000@192.168.1.B:5080」，它是一个呼叫字符

¹ 也称为「外网用户」，但为了跟「本地」相对应，我们这里用了「外地」。实际应用中也经常使用「本局用户」和「他局用户」这样的称呼。

² 以前的交换机也称为交换局，因此呼出和呼入就常称为出局和入局。

串。在一个呼叫中，当 FreeSWITCH 执行到这里的**bridge**时，就会从本机的**external Profile**（本机的5080端口）向 B 的5080端口发送 INVITE 呼叫请求。

注意，在上述过程中，被叫号码中的第一个 0 在到达 B 时丢失了。这是系统对接中常用的一个策略，俗称把 0「吃掉」了。

B 在5080端口上收到 INVITE 请求后，由于5080端口默认走**public Dialplan**，所以查找**public.xml**，并可以找到以下的 Dialplan 配置项：

```
<extension name="public_extensions">
  <condition field="destination_number" expression="^(10[01][0-9])$">
    <action application="transfer" data="$1 XML default"/>
  </condition>
</extension>
```

上述 Dialplan 中的正则表达式「`^(10[01][0-9])$`」会匹配被叫号码1000，然后执行**transfer**，并把来话**transfer**到**default Dialplan**。呼叫转到**default Dialplan**后，路由规则就跟本地用户的来话一样了，因而最终 B 上的1000就会振铃。如果 B 摘机接听，电话就可以接通了。

如果 B 上的用户也要拨打 A 上的用户，那么，只需要在 B 上也做类似 A 上的配置就可以了。因为 A 和 B 是完全对称的。感兴趣的读者可以自行试一下，我们就不罗嗦了。

第二章 汇接

理论上来讲，所有的对接模式都可以采用上一章的双机对接模式，即，上述的对接模式是一切对接的基础。但是，随着参与通信的人越来越多，网上的交换机越来越多，就需要更复杂的组网方式。

下面，我们来考虑一下更复杂的情况，比方说，在此，我们假设全世界的交换机都变成 FreeSWITCH 的情况。我们将各 FreeSWITCH 编号为 A、B、C、D、E、F、G……如果 A 上的用户要跟世界上所有的用户都能通话，那么，A 上就需要配置到所有其他主机的路由，这显然是不现实的。

为了解决这一问题，A、B、C、D 开会讨论。D 主动说：「我精力比较旺盛，我是雷锋，给你们 3 个提供转接服务吧，你们就不用费心了」。这时候，D 就成了一个汇接局，为 A、B、C 之间的通话做转接服务。A、B、C 就称为端局，因为他们只有终端用户（本地用户）。拓扑结构如下图所示：

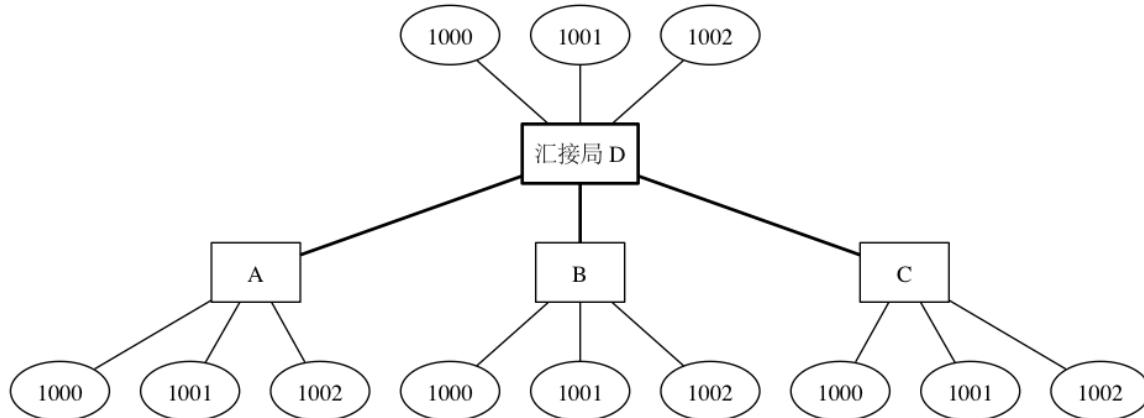


图 2.1: 汇接局模式

同时，大家商量了新的拨号规则：本地用户之间的通话拨号规则不变，但是如果拨打其他外地用户的话，则需要拨打相应的局号（即，如果 A 上的用户呼叫 B 上的 1000，则拨打 B1000），并统一送到 D 进行汇接。我们以 A 为例，它上面的 Dialplan 如下所示：

```
<extension name="D">
<condition field="destination_number" expression="^([B-Z].*)$">
<action application="bridge" data="sofia/external/sip:$1@192.168.1.D:5080"/>
```

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

其中，正则表达式「`^(B-Z) .*)$`」表示，任何以 B ~ Z 开头的号码（我们假设世界上只有这么多 FreeSWITCH）都送到 D（即 192.168.1.D）的5080端口上去。注意，这里并没有「吃掉」第一位号码，因为如果吃掉的话，D 就不知道如何进行下一步路由了。所以，如果 A 上的用户拨打B1000，在 D 上将收到B1000。

在 D 上，收到5080端口的呼叫请求后，查找public Dialplan 对来话进行路由。它的 Dialplan 设置如下：

```
<extension name="D">
<condition field="destination_number" expression="^D(.*)$">
<action application="transfer" data="$1 XML default"/>
</condition>
</extension>
```

上述 Dialplan 说明，如果被叫号码是 D 开始的，则说明是一个本地用户，所以「吃掉」最首位的「D」，然后把路由转（transfer）到default Dialplan 进行处理。

对于被叫号码不在本地的用户，则使用下列 Dialplan：

```
<extension name="D">
<condition field="destination_number" expression="^(A-CE-Z)(.*)$">
<action application="bridge" data="sofia/external/sip:$2@192.168.1.$1:5080"/>
</condition>
</extension>
```

其中，正则表示式「`^(A-CE-Z)(.*)$`」匹配所有除 D 以外的 A 到 Z 开头的被号码。我们还是以有人拨打 B1000 为例，匹配成功后，\$1 的值为 B，而 \$2 的值为 1000，所以，bridge 的参数中呼 叫字符串就会变成「`sofia/external/sip:1000@192.168.1.B:5080`」，因而相当于在 D 上「吃掉」了被叫号码中最首位的「B」，并把电话送到 B 的5080端口上。

电话到达 B 后，B 上默认的 Dialplan 就可以将电话路由到本地用户上，因而电话接通。

这种方式就称为汇接模式。由于 D 即带本地用户，又作为汇接局，因而，它是一种汇合模式的 汇接局。如果 D 上不带用户，而专门做汇接这项工作，就称为一个纯粹的汇接局。

所有局间的通话都是通过 D 的，因而它的压力比较大。在实际应用中，如果 A、B、C 之间的网 络直接可达的情况下，可以让 D 仅转发 SIP 信令，而让 RTP 流直接在端局之间传递。在上述配置 的bridge Action 之前增加如下参数可以让 D 工作在 Bypass Media（媒体绕过）模式，仅转发 SIP 信令，而让 RTP 媒体流在端局之间传送：

```
<action application="set" data="bypass_media=true"/>
```

上述参数是在 Dialplan 中设置的，因而仅针对当前通话有效。如果想让 D 对所有通话，不管三七二十一都使用 Bypass Media，则可以直接在 Profile (这里我们使用的是 external) 中添加如下设置：

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

当然，理论上讲，除了 D 之外，其他的端局也可以采用 Bypass Media 技术，让媒体流只在终端用户之间发送，而不经过 FreeSWITCH。但是，一般来说，终端用户可以会位于 NAT 设备的后面，他们之前的媒体流互通并不总是可行的，因而在端局很少这么做。

第三章 双归属

即使采用了 Bypass Media，D 的压力还是很大。当然，不一定是光是电话的压力，更可能是精神上的压力。因为，如果一旦 D 出现了故障，则 A、B、C 之间的用户就都打不通电话了。

为了解决这一问题，它们又找到了 E，让它做一个备份的汇接局。并且，把 D 和 E 的本地用户都分了一下，放到 A、B、C 上，让 D 和 E 专门做汇接。拓扑结构如图 3.1 所示。

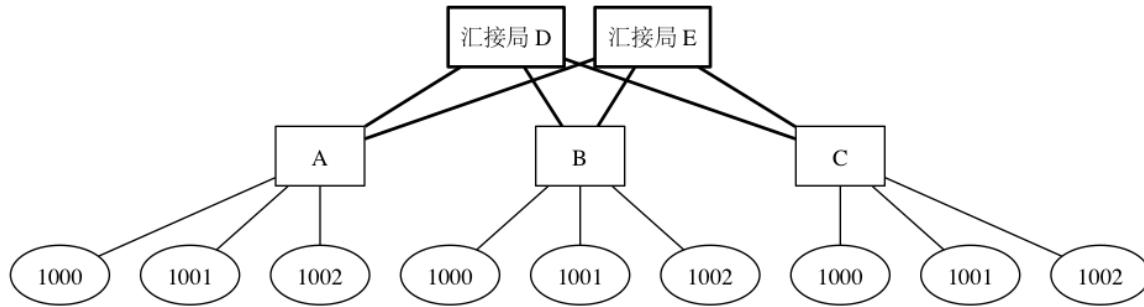


图 3.1: 双汇接局双归属网络拓扑

这样，把 E 上的路由规则配置的跟 D 上一模一样。但每个端局都同时连接到两个汇接局上。一旦其中一个出现故障，则另一个可以接替工作。这种拓扑方式就称为双归属（每个端局都归属于两个汇接局）。端局的配置如下（仍以 A 为例，注意，由于「192.168.1」太长了，为了排版方便，我们以「IP.」代替它，读者知道它是一个 IP 地址就行了）：

```
<extension name="DE">
<condition field="destination_number" expression="^([B-Z].*)$">
<action application="bridge"
      data="sofia/external/sip:$1@IP.D:5080|sofia/external/sip:$1@IP.E:5080"/>
</condition>
</extension>
```

我们修改了呼叫字符串，增加了使用「|」连接起来的两个呼叫字符串。它的意思是，如果第一个呼叫不成功（到 D 的），则使用下一个呼叫字符串（送到 E 上）。这种方式就称为主备用模式。

很容易看出，上述配置虽然解决了 D 的精神压力，但是，它实际的话务压力还是很大的。因为，只要它不出故障，E 就没事干，显然很不公平。所以，实际的双归属出局大部分是以话务负荷分担的方式进行的。所谓负荷分担，就是平时在端局将 50% 的通话送到 D 上，50% 通话送到 E 上，让两端的负载差不多。一旦其中一台发生故障，则所有的通话都送到另外一台上。负荷分担的算法有很多，这里，我们可以简单粗野地使用如下 Dialplan 实现（以 A 上的出局路由为例）：

```
<extension name="DE">
<condition field="destination_number" expression="^([B-Z].*[13579])$">
<action application="bridge"
  data="sofia/external/sip:$1@IP.D:5080|sofia/external/sip:$1@IP.E:5080"/>
</condition>
</extension>

<extension name="ED">
<condition field="destination_number" expression="^([B-Z].*[24680])$">
<action application="bridge"
  data="sofia/external/sip:$1@IP.E:5080|sofia/external/sip:$1@IP.D:5080"/>
</condition>
</extension>
```

通过上述配置，根据正则表达式「`^([B-Z].*[13579])$`」，所有被叫号码以奇数结尾的都优先送到出局局 D 上（如果 D 失败仍然会送到 E）。同理，所有以偶数结尾的被叫号码都会优先送到 E 上，如果失败后则尝试 D。这就实现了一个简单的「50%/50%」负荷分担策略。其他类似的策略可以使用随机数实现，也可以使用 `mod_distributer`（FreeSWITCH 中的一个模块，我们后面还会讲到）中提供的方法实现。当然，实际应用中需要考虑的事情还有很多，这里就不多讲了。

第四章 长途局

一般来说，一个地市级的电话网络两个汇接局就够了。如果需要打长途电话的话，上面就再建设长途局，与其他地市的长途局互通。有了长途局的网络拓扑结构如图 4.1 所示（我们将另外一个地市的名称全部以小写字母表示）。

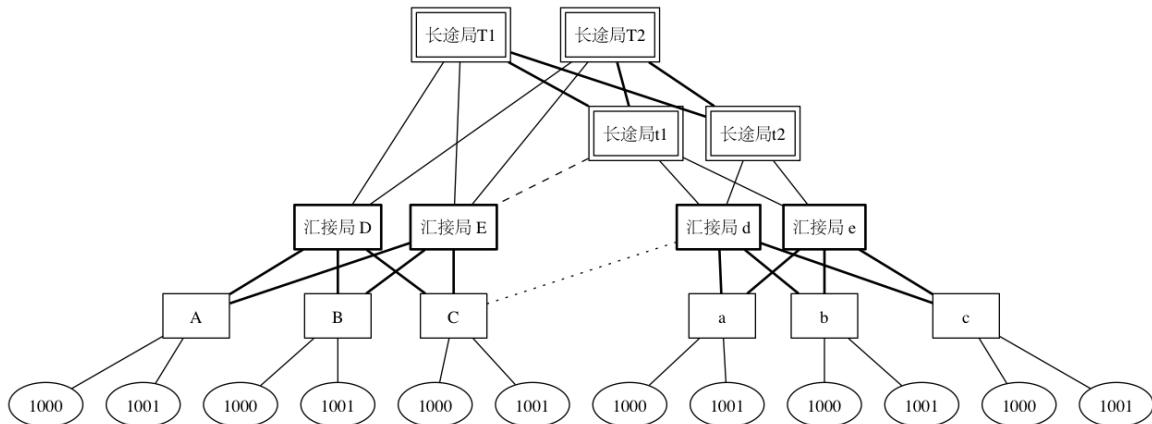


图 4.1: 有长途局的网络拓扑

长途局也是采用成对配置的方式，汇接局跟长途局之间也采用双归属。另外，如果某地市到另外一个地市的话务量比较高，也可以从汇接局直接向另外一个地市的长途局开中继并设置高速直达路由（如 E 和 t1 之间），它可以避免占用本地长途局的资源。当然，极少数情况下也可能从本地端局直接向另外一个地市的其他局（如汇接局，C 和 d 之间）开通高速直达中继。

具体的配置方式我们就不多讲了。总之，电话网就是这样以树状和网状的结构向外延伸，最终到达世界的每一个角落。

第五章 FreeSWITCH 作为 PBX

在图 4.1 所示的图网络结构中，假设 A、B、C 一层的都是运营商提供的端局交换机，最底层的 1000、1001 等是端局上的用户。如果再向下延伸，我们可以在端局用户这一层再搭建自己的 PBX，下面再挂分机用户。

5.1 普通的 PBX 设置

毕竟，FreeSWITCH 的配置就是一个全功能的 PBX。假设我们有了 A 上 1000 这个用户账号，我们以前是接了一个 SIP 软电话往外打电话的。现在，我们把它换成 FreeSWITCH（我们称为 F），并且，在 FreeSWITCH 上，我们创建 600 ~ 619 这 20 个用户¹，这时候的拓扑结构如图 5.1 所示。

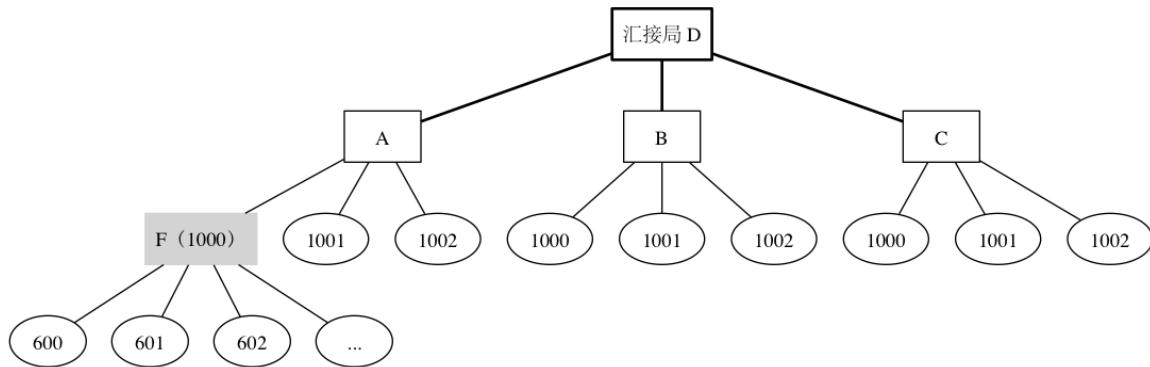


图 5.1: FreeSWITCH 作为 PBX

因为 F 是作为 A 上的一个用户（1000）存在的，所以，它只能作为一个普通用户向 A 去注册。对于 A 而言，它就认为 F 是一个普通的 SIP 电话客户端（或软电话）。

为了能使 F 能向 A 注册，我们在 F 上添加一个网关（gateway）指向 A，配置如下²:

¹ 创建用户我们应该已经很熟悉了。首先我们可以手工添加这些用户，也可以简单的修改原有的 1000 ~ 1019 这些用户配置文件，改成对应的 600 ~ 619。

² 如何添加网关我们应该已经很熟悉了，可以将下面的内容存放到 `conf/sip-profiles/external/gw_a.xml` 文件中，并在 FreeSWITCH 中使用「`sofia profile external rescan`」命令使之生效。

```
<include>
<gateway name="gw_A">
    <param name="realm" value="192.168.1.A"/>
    <param name="username" value="1000"/>
    <param name="password" value="1234"/>
</gateway>
</include>
```

添加完上述配置后这样，我们就有了一个名称为gw_A的网关。本地的用户600 ~ 619就可以通过如下的 Dialplan 拨打外部的电话了（假设 PBX 的出局码为 0）：

```
<extension name="ga_A">
<condition field="destination_number" expression="^0(.*)$">
    <action application="bridge" data="sofia/gateway/gw_A/$1"/>
</condition>
</extension>
```

该 Dialplan 的含义就是遇到以 0 开头的被叫号码，吃掉 0，然后将电话送到gw_A 定义的网关上。假设分机 600 想拨打 A 上的 1001，就可以直接拨打 01001，600 的 SIP 客户端将向 F 发送 INVITE（呼叫 01001），F 再向 A 发送 INVITE（1001），对于 A 而言，F 到底是用的 FreeSWITCH 还是用的 SIP 软电话都是一样的。在 1001 上收到呼叫以后，看起来也是从 1000 打过来的（来电显示的号码是 1000）。

如果 A 上的 1001 想呼叫 600，则它必须先呼叫 1000，因为 600 这个号码是无法直达的（如，600 这个人的名片上可能印着「电话：总机(A)1000转600」），然后 F 上会启动一个 IVR，让 1001 输入一个分机号，并为其转接到相应的分机号。

一般来说，端局 A 是不允许主叫号码透传的。即不管 F 上的哪个分机往外打电话，都会在对方的话机上显示 1000 这个主叫号码。当然，我们这里的 A 是 FreeSWITCH，那么我们就可以设置允许 1000 往外打电话时进行主叫号码透传。只需要在 A 上找到 1000 这个用户的配置文件（1000.xml），将下面的行注释掉就可以了。

```
<variable name="effective_caller_id_number" value="1000"/>
```

其中，`effective_caller_id_number` 就表示 1000 这个用户如果发起呼叫时（从 F 来的所有呼叫都变为是 1000 这个用户发起的）对外显示的号码是什么，默认的设置就是 1000。注释掉该项后，就会根据实际的来电号码对外进行发送。那么，如果 600 发起呼叫时，在 A 上看到的实际的来电号码是什么呢？首先，600 发起呼叫时，呼叫到达 F，F 查找 600 的用户配置文件，应该能找到类似的如下的配置，因此，F 将对外发送主叫号码 600。

```
<variable name="effective_caller_id_number" value="600"/>
```

呼叫到达 A 后，它看到 F 送过来的主叫号码是600，而1000这个用户又没有设置`effective_caller_id_number`这个参数（从 F 来的所有通话均认为是从1000来的，刚才该参数已经被我们注释掉了）便可以对外显示这个600，因而1001话机上就显示了600，实现了电话号码的透传。

可以看出，实现电话号码透传本身是很简单的事情。所以它本身不是什么技术问题，而在实际应用中主要是面临一个现实问题——那就是，如果所有人都可以做任意透传电话号码的话，那么所有人都可以任意冒充任何人打电话，试想一下如果有人冒充 110 或国务院的电话的情况。

电话号码透传带来的另一个问题是回呼。在上述情况下，如果1001接收到600来的呼叫，显示的主叫号码是600，显然，这个号码是无法回呼的，即1001无法通过这个号码直接呼通600，因为，600这个号码，理论是讲，对于1001所在的交换机 A 来讲，是不存在的。它最多只能呼通到1000。

为了解决这个问题，我们先把 600 呼出时对外显示的主叫号码变换一下，下列 Dialplan 将 600 的主叫号码变换为 1000600：

```
<extension name="ga_A">
<condition field="destination_number" expression="^0(.*$)">
<action application="set"
  data="effective_caller_id_number=1000${caller_id_number}"/>
<action application="bridge" data="sofia/gateway/gw_A/$1"/>
</condition>
</extension>
```

至于为什么这么变换，我们将在下一节给出答案，并给出回呼解决方案。

5.2 DID

我们在前面的章节中不止一次的讲到 DID 这个概念。在本节的第 (1) 种方式中，对于 F 而言，号码1000就是一个 DID。因为在 A 上的其他用户都可以通过拨打1000这个号码打到 F 上，其他交换机如 B、C 上的用户也可以通过拨打 A1000 呼叫到 F 上。当然，从第 (1) 节我们也看到，其他用户如果想呼叫 F 内部的分机，就只能先拨打1000这个 DID 号码，然后，再通过 IVR，以人工总机或自动总机转接的方式转到对应的分机上。那么，我们能不能直接呼叫到内部的分机呢？毕竟，DID 的真正含义是「对内直接呼叫」，目前看来，我们还未实现这个目标。

要实现这个在技术上当然是可以的。让我们来考虑这样一样拨号方案：对所有 A 上的用户而言，如果想直接呼叫 F 上的内部分机600~619，就需要呼叫1000加上这些分机号。如1001呼叫 F 上的600，则需要呼叫1000600。

我们已经在上一节在 600 对外呼出的将主叫号码设置成了可以对外显示 1000600，因此，只要能在 A 上设置正确的回呼路由，电话就应该能呼通了。

确定了拨号方案以后，我们再来看在 A 上的实现。为了要让 A 上其他用户打 1000 开头的电话号码都送到 F 上，我们需要在 A 上增加一个 Dialplan 项，于是，我们很快想到了如下的 Dialplan：

```
<extension name="F-DID">
  <condition field="destination_number" expression="^(1000.*$)">
    <action application="bridge" data="..."/>
  </condition>
</extension>
```

其中，我们让的正则表达式「`^(1000.*$)`」表示匹配任何以 1000 开头的电话号码。如果匹配到这样的电话号码，则将电话 bridge 到……可是，到这里，bridge 后面的参数怎么写呢？我们知道，这里，我们要给一个呼叫字符串，但是，这里的情况不同于我们上面讲到的任何一种情况，呼叫字符串该怎么写呢？

首先，让我们来看一下这里的情况与上述讲的有什么不同。在上面的各种情况中，各 FreeSWITCH 之间的对接都是通过 IP 方式的，即一般来说各 FreeSWITCH 之间都互相知道对方的 IP 地址，也就是说各 FreeSWITCH 的 IP 是相对固定的。而在这里，F 的 IP 地址对 A 来说可能是不固定的。因为 F 处于远端，它是以动态注册的方式注册到 A 上的，因而只有 F 向 A 注册的时候 A 才能知道 F 的 IP 地址，并且，F 的 IP 是可能是不断变化的，每次变化后 F 都会重新向 A 注册它的 IP 地址，以便 A 能找到它。在这种情况下，如果让 A 能找到 F，那么，我们就应该能在 A 上动态地获取 F 的 IP 地址。

实际上，我们已经知道，A 本来就能动态获取 F 的 IP 地址，下面的配置方式我们已经很熟悉了：

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

上面的呼叫字符串「`user/1000`」我们已经讲到很多次了，它的作用就是在 A 上找到 1000 这个注册用户注册时的 Contact 地址（即 F 的地址），然后向该地址上发送 INVITE 请求。

但在这里，如果我们还是这样配置的话，很显然不能达到我们的要求。因为，使用这种方法获取到的呼叫字符串，在呼叫到达 F 后，被叫号码永远是 1000，而我们想要在有人呼叫 1000606 时把被号码变成 1000606。所以，这里，我们就需要解决两个问题：

- 在 A 上动态能找到 F 的 IP 地址；
- 呼叫到达 F 后被叫号码变成我们指定的被叫号码。

我们先一步一步来看。当 F 向 A 注册时，它提供了自己的 Contact 地址，我们通过前面的学习已经知道可以用以下命令在 A 上查看（以 1000 为过滤条件显示注册用户）：

```
freeswitch> sofia status profile internal reg 1000
```

```
Registrations:
=====
Total items returned: 0
=====
```

但令人奇怪的是，它并没有显示出1000的注册信息。这是为什么呢？

在回答为什么之前，我们先来试一下以下的命令。在下列命令中我们在显示时去掉了1000这个过滤条件，从中，我们可以看到类似如下的输出信息（省略其他不相关的输出）：

```
freeswitch> sofia status profile internal reg

Registrations:
=====
Call-ID: d3db5f74-3fb9-4951-ac38-75b419ba4894
User: 1000@192.168.1.F
Contact: "user" <sip:gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A>
....
```

从中可以看出，「User」一行标明我们的确有一条来自192.168.1.F上的用户1000的注册信息，不过，该用户的注册信息中的Contact地址是比较奇怪的。由于它里面没有包含1000这样的用户名，所以我们在上面想用「1000」为过滤条件限制输出结果时失败了。

当然，该Contact字符串来自于F，我们来讲一下F为什么这么做。

F向A注册是以在F中添加一个到A的网关实现，并以「gw_A」这个名字来标志这个网关。F向A注册时使用了这样的Contact字符串。注册完成后，如果A上有电话需要呼叫F时，A就会向A发送INVITE请求。通过上面的注册信息，A向F发的INVITE请求如下：

```
INVITE gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A SIP/2.0
```

这样，当F上收到上面的INVITE请求后，就可以知道该呼叫是从gw_A这个网关来的，这样便于知道呼叫的来源。否则的话，F也可以使用像「1000@192.168.1.F:5080」这样的Contact字符串，可是那样的话它会收到类似「INVITE 1000@192.168.1.F:5080」这样的呼入请求，不利于将呼叫来源与F中本身已定义的gw_A网关关联。

继续看上面的INVITE请求。其中有两个「gw_A」。第一个「gw+gw_A」实际上出现在被叫号码的位置，这时候如果F检测到被号号码中有「gw+」以后，就能找到gw_A这个网关了。有时候，在A是其

他 SIP 服务器的情况下，可能会将该字段放入真正的被叫号码，如「INVITE 1000@192.168.1.F…」，这样，F 就会尝试从后面的「`gw=gw_A`」这个位置（分号后面都相当于参数）找到`gw_A`这个网关名称。当然，如果 A 也不发送这些参数的话，F 就无法与系统中配置的网关进行关联了（不过电话仍然可以通）。

好了，言归正传。F 收到上述的 INVITE 请求后就会在本地的网关`gw_A`中找到真正的被叫号码1000(从「`extension`」参数或「`username`」参数中找)。

实际上，上述的 Contact 地址已经包含了 F 的 IP，我们可以在 A 的命令行上使用下列命令找到它：

```
freeswitch> sofia_contact internal/1000@192.168.1.A
sofia/internal/sip:gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A
```

其中，`sofia_contact`是一个 API 命令，它的参数格式为「Profile/User@Domain」。可以看到，我们这里的 Profile 是`internal`，User 是1000，Domain 就是 A 的 IP 地址。

在继续往下研究之前，我们需要先来做一些试验。试验中，我们会用到`echo`和`expand`这两个 API 命令。其中，`echo`命令会将字符串原样输出，如：

```
freeswitch> echo test
test

freeswitch> echo $$domain
$$domain
```

我们虽然已经知道 A 上的 Domain (即 A 的 IP 地址) 是 192.168.1.A，而且它也是相对固定的。但是，在实际应用中还是用变量引用比较方便一些。默认情况下，在 XML 中的配置都是会进行变量替换的，但在命令行上不会。如果要在命令行上进行变量替换，就需要用到一个`expand` API 命令，如下：

```
freeswitch> expand echo $$domain
192.168.1.A
```

可以看到，由于`expand`的作用，`$$domain`被替换成了实际的值。接下来，我们继续进行实验：

```
freeswitch> expand sofia_contact internal/1000@$$domain
sofia/internal/sip:gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A
```

有了1000的 Contact 地址后，我们可以构造如下的命令呼叫它，如：

```
freeswitch> expand echo originate ${sofia_contact(internal/1000@${domain})} &echo
originate sofia/internal/sip:gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A &echo
```

可以使用「\${API 命令(参数)}」的形式引用一个 API 命令的结果。所以，上述的**originate**命令后面的参数需要的呼叫字符串就是用「\${sofia_contact()}」动态获取的。当然，上述的命令是**echo**输出后的结果，如果我们去掉**echo**，就可以直接呼叫1000了：

```
freeswitch> expand originate ${sofia_contact(internal/1000@${domain})} &echo
```

在 A 上执行上述**originate**命令后，它就会向 F 发送如下的 INVITE 请求：

```
INVITE sip:gw+gw_A@192.168.1.A:5080;transport=udp;gw=gw_A SIP/2.0
To: <sip:gw+gw_A@192.168.1.A:5080;transport=udp;gw=gw_A>
```

然后 F 在收到上述请求后就能找到`gw_A`这个在本地配置的网关，进而找到对就的被叫号码1000，并进行路由。

如果要将被叫号码改为我们指定的被叫号码（如1000606），我们只需要想办法改变 INVITE 请求中的被叫号码部分，即将`sip:gw+gw_A@192.168.1.A:5080`变成`sip:1000606@192.168.1.A:5080`即可。为了达到这个目的，我们使用正则表达式替换。其中，FreeSWITCH 提供了一个**regex** API 命令可以进行正则表达式替换，它的语法是「**regex** 原字符串 | 正则表达式 | 替换后的内容」，如：

```
freeswitch> regex sip:gw+gw_A|^sip:gw\+(.*)|sip:1000606
sip:1000606
```

其中，使用「`^sip:gw\+(.*)`」作为正则表达式匹配就可以将原字符串「`sip:gw+gw_A`」替换为`sip:1000606`，这也是我们主要需要替换的部分。好了，我们来看一个完整的替换，替换前的字符串为：

```
freeswitch> expand echo originate ${sofia_contact(internal/1005@${domain})} &echo
originate sofia/internal/sip:gw+gw_A@192.168.1.F:5080;transport=udp;gw=gw_A &echo
```

替换后：

```
freeswitch> expand echo originate ${regex(${sofia_contact(internal/1005@${domain})}|^(.*)sip:gw\+gw_A@(.*)|$1sip:1000606@$2)} &echo
originate sofia/internal/sip:1000606@192.168.1.F:5080;transport=udp;gw=gw_A &echo
```

可以看到，我们得到了需要的呼叫字符串，接下来，就可以去掉上面echo进行一下呼叫测试了。如果在 F 的日志中，看到类似「Processing <000000000000>->100606 in context public」这样的行就说明我们修改被叫号码成功了。

接下来，可以就可以完成我们的F-DID Dialplan 了：

```
<extension name="F-DID">
<condition field="destination_number" expression="^(1000.*$)">
<action application="bridge"
  data="${regex(${sofia_contact(internal/1005@${domain})}|^(.*)sip:gw\+gw_A@(.*)|%1sip:$1@%2})"/>
</condition>
</extension>
```

注意，与在命令行上不同的是，由于 Condition 条件中的正则表达式匹配中的 Capture 结果已经占用了\$1、\$2之类的变量，因而，为避免冲突，在 Dialplan 中使用\${regex()}之类的表达式时，其 Capture 结果中的变量用%1、%2之类的表示。

总之，本节给出了一个端局交换机对下面注册上来的 PBX 中的分机用户的一种直接呼叫方案。虽然该方案不是唯一的解决方案，但它至少是一种有效的解决方案。其它的解决方案原理与此差不多，都是要想办法找到用户注册上来的联系地址，并进行相应的替换。当然，在实际应用中，这种方案是否可能还得局端交换机 A 与 PBX 交换机 F 相互配合来实现。在此，我们就不多介绍了。

5.3 使用 PBX 上的网关呼出

下面，我们再来看另外一种场景。如图 5.2 所示，假设 A 是运行在公网上的 FreeSWITCH，而 F 是运行在私网上的 PBX（也是 FreeSWITCH），F 仍然使用 1000 这个号码向 A 注册，并且，F 上自己带了 600 ~ 619 之间的分机用户。另外，F 上还可以通过另外的一个网关 G 可以与外界沟通。在现实场景中，G 就可能是一个连接模拟线的模拟网关，该网关一端跟 FS 通过 SIP 相连，另一端则通过模拟电话线连接 PSTN 交换机。由于我们运行在公网上的 A 可能没有对外的中继，因而它上面的用户 1000 ~ 1019 可能也希望通过 F 上的网关 G 对外呼出。

有了上一节的经验，我们知道实现这个也很简单。首先，就是将 A 上的外呼请求先转发到 F 上。根据上一节的方案，我们可以使用下列 Dialplan 配置来做到：

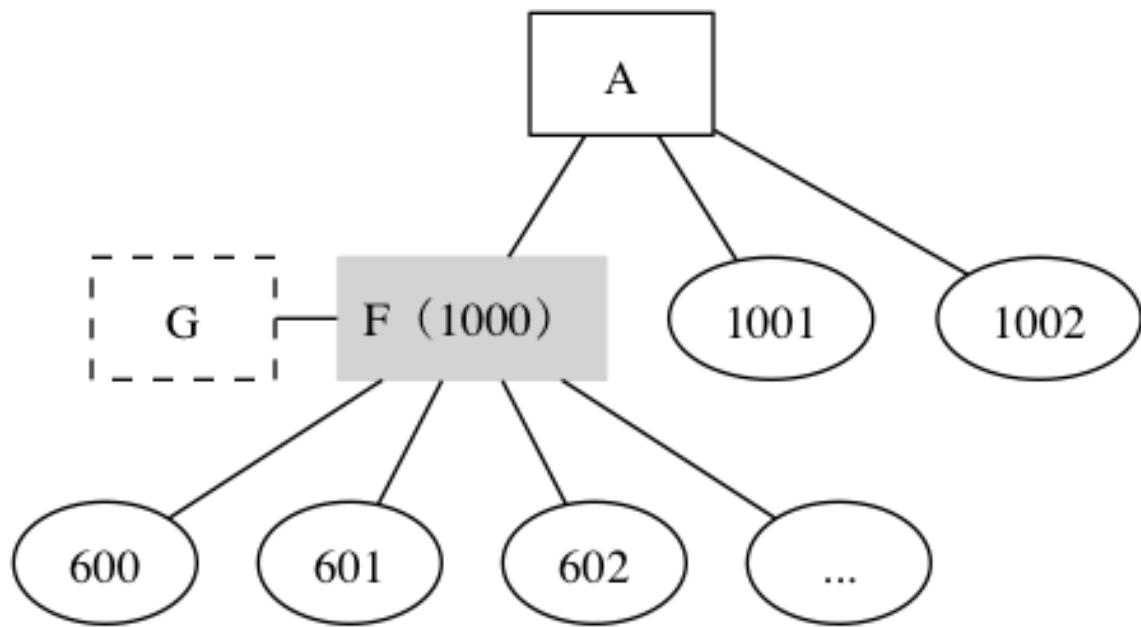


图 5.2: PBX 上带网关的结构图

```

<extension name="F-GW">
  <condition field="destination_number" expression="^(0.*)$">
    <action application="bridge"
      data="${regex(${sofia_contact(internal/1005@${domain})}|^(.*)sip:gw\+gw_A@(.*)|%1sip:${destination_number}@%2})"/>
  </condition>
</extension>

```

其实该 Dialplan 跟上一节的一样。我们只是用正则表达式「`^(0.*)$`」来匹配以 0 开头的被叫号码，然后通过 `bridge` 将这种呼叫送到 F 上。在 F 上收到这种呼叫请求后，就可以使用下列的 Dialplan 进行呼出了：

```

<extension name="F-GW">
  <condition field="destination_number" expression="^(0.*)$">
    <action application="bridge" data="sofia/gateway/G/$1"/>
  </condition>
</extension>

```

至此，我们从双机对接开始，一步步的讲了大型通信网络的组织及演变，以及 FreeSWITCH 在各个节点和环节发生的作用。同时也讲解了在 FreeSWITCH 中是怎么实现这些功能的。以后，如果 FreeSWITCH 需要与其他系统对接，不管 FreeSWITCH 是处于哪个位置，总能在本章找到合适的配置方式。读者熟悉了这些配置，就可以以不变应万变，在各种组网环境下都能很快的使用 FreeSWITCH 实现了。

最后，我们还讲了一个公网的 FreeSWITCH 服务器通过位于私网的 PBX 上的中继线或网关呼出的例子。这种方式并不是最好的配置，但在开发人员很难自由的获得公网上可用的 SIP 账号的情况下，做到公网上的服务器也 PSTN 网络的互联，也不失是一种办法。而且，确实有好多网友问到该问题。

当然，本书讲解这些例子的目的倒不是仅仅为了回答网友的提问。而是希望读者通过这些例子，能从不同的角度和维度深入的理解 FreeSWITCH、理解通信网络，以及通信网络中各个组成部分的协作和通信方式，以便在以后的工作中更有效地解决各种问题，创造更好的应用。

第 II 部分 武林大会

在第 I 部分中，我们讲了不同的 FreeSWITCH 之间的对接。通过学习 FreeSWITCH 与 FreeSWITCH 对接，我们不仅享受了周伯通那种双手左右互博的乐趣，更全面了解 FreeSWITCH 在各种场景下的应用。在了解了 FreeSWITCH 世界的配置及组网方式之后，我们该走出 FreeSWITCH，下山去看看外面的世界了。

第六章 连接 SIP 软电话

FreeSWITCH 最典型的应用是作为一个服务器(它实际上是一个背靠背的用户代理，B2BUA)，并用电话客户端软件 (一般叫软电话) 连接到它。虽然 FreeSWITCH 支持 IAX、H323、Skype、Gtalk 等众多通信协议，但其最主要的协议还是 SIP。支持 SIP 的软电话有很多，在此分别以 X-Lite 和 Doubango 软电话来测试一下 SIP 注册。

6.1 X-Lite

X-Lite 是常用的软电话，它可以跨平台的用于 Mac、Windows 和 Linux，在此，我们使用的是 Mac 平台上的 X-Lite 4.7.1 (74250)。

FreeSWITCH 默认配置了 1000 ~ 1019 共 20 个用户，可以随便选择一个用户进行配置。在此，我们使用 1000 这个账号。

在 X-Lite 主菜单上选择「Preference」→「Accounts」并按「+」号按钮添加一个账号，填入以下参数 (如图)：

```
Display Name: Seven Du (可以随便填)
User name: 1000
Password: 1234
Authorization user name: 1000
Domain: SIP 服务器 IP 地址
```

其他都使用默认设置，点「OK」就可以了。这时 X-Lite 将自动向 FreeSWITCH 注册。注册成功后会显示账号当前的状态。如果界面上显示绿色的注册成功标志就可以拨打电话了。

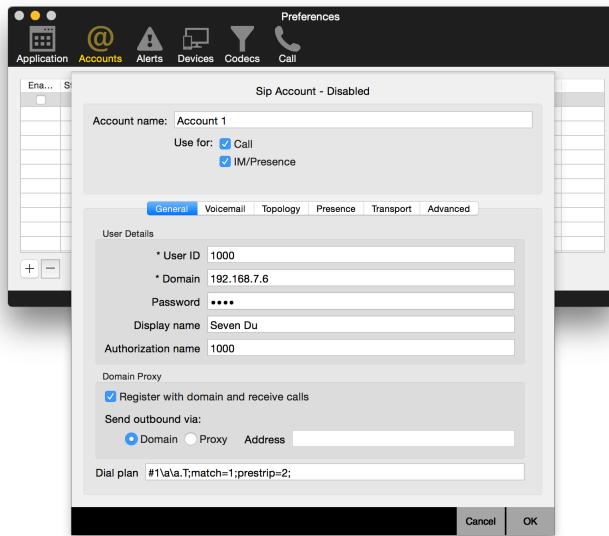


图 6.1: X-Lite 添加账号

6.2 Doubango

Doubango¹是一个不错的开源框架，它跟电信业务走的比较近，主要集中在以 3GPP、TISPAN、Packet Cabel、WiMax、GSMA、RCS-e、IETF 等标准的 NGN 技术、音视频处理技术、云计算以及 WebRTC 技术等。

该框架使用 ANSI C 编写，具有很好的可移植性。在很多平台上都有基于 Doubango 的客户端实现，如 Windows 上的 Boghe、Mac 上的 iDoubts 以及 Andriod 系统上的 IMSDriod 等。

但是，我们这里单独介绍 Doubango 的原因并不是因为他的强大，而是因为它总是倾向于把简单的东西搞得异常复杂。

一般来说，最简单的 SIP 注册一共就需要三个选项：服务器地址（Realm）、用户名（Username）和密码（Password），但在 Doubango 中，就没这么简单了。如果你要注册，就得填一大堆的东西，并且格式相当严格和专业，这令好多初学者都摸不着头脑。

下面我们以 Mac 平台上的 iDoubts 为例，讲一下它往 FreeSWITCH 上的注册方法。下图是笔者使用 Mac 版的 iDoubts 注册时的参数设置，在其他平台上可以参考使用。

要进行注册，首先，要到「Preferences」->「Network」中设置Proxy-CSCF-Host和Proxy-CSCF-Port。这是代理服务器的地址和端口号，相当于其他软件高级设置中的「Outbound Proxy」。这个地址的意思是，如果 Doubango 注册时，会将注册的包发到这个地址。而它实际请求域（或简单认为是注册的地址）将是下一步我们将要设置的 Realm 的地址。但由于我们的 FreeSWITCH 只有一个地址，因而这里就填 FreeSWITCH 服务器的 IP 地址。

¹参见<http://doubango.org/>。

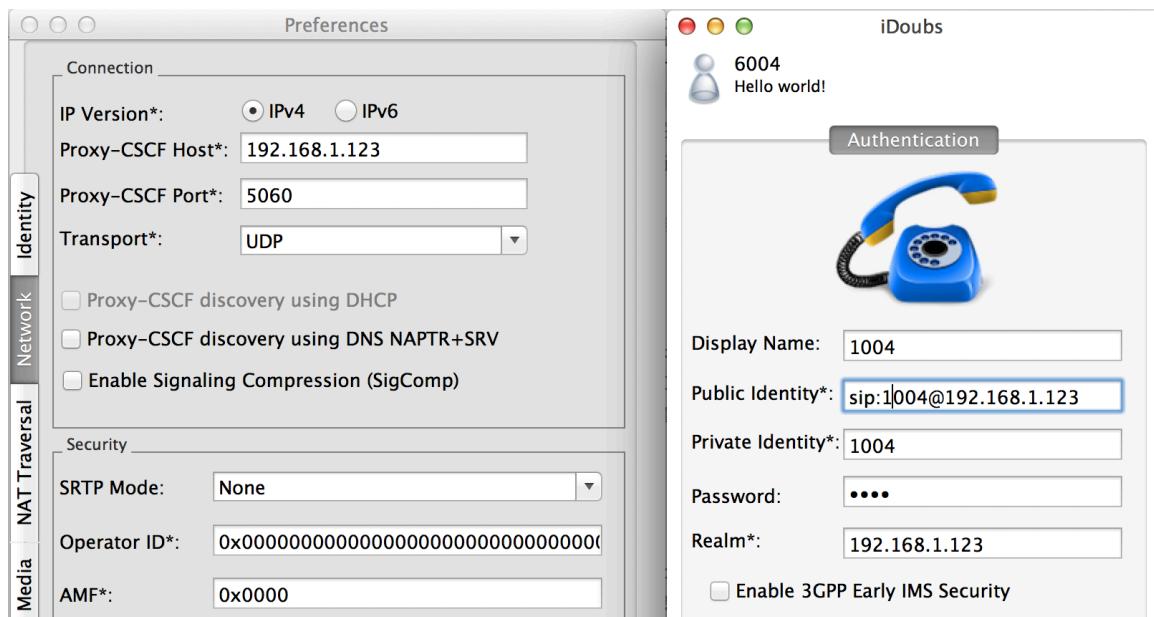


图 6.2: iDoubs 注册设置

接下来，要填写注册相关的参数。其中，Display Name 跟其他软件中的一样，是可以随便填的；Public Identity 是 IMS 里的概念，相当于你的公有 SIP 地址，就是说，别人呼叫你的时候要呼叫这个地址。注意这个地址的格式，其中的「sip:」是不能省略的；Private Identity 是你的私有账号，这个在认证的时候要使用，相当于其他软件中的 Username 或 Auth-User（会出现在 SIP 消息的 Authentication 头域中）；Password 就是密码，Realm 是服务器的域（或 IP 地址）。

总之，基于 Doubango 的所有默认应用都倾向于把填写注册信息弄的异常复杂²。不过，如果顺利过了这一关，大家也能学到不少的知识。尤其是在下一节的对接 IMS 系统时我们还会看到更复杂一些的参数。

6.3 拨打测试

注册完成后，就可以拨打一些号码进行测试了，下面是系统自带的一些测试号码：

号码	说明
9664	保持音乐
9191	注册 ClueCon
9192	在 log 中显示 Channel 信息
9195	echo，回音测试，延迟 5 秒

²实际上，其他类似的的 SIP 客户端软件也都需要这些信息，但它们都会要求填入最少的信息，如果可能的话，会自动计算这些值，或者有合理的默认值（如 FreeSWITCH 中的网关设置）。Doubango 官方提供的各种应用在这一点上都不是很人性化。

号码	说明
9196	echo, 回音测试
9197	milliwatte extension, 铃音生成
9198	TGML 铃音生成示例
9180	铃音测试, 使用远端生成的回铃音
9181	铃音测试, 产生英式铃音
9182	铃音测试, 使用音乐当铃音, 彩铃
9183	先应答, 然后发送英式铃音
9184	先应答, 然后发送音乐铃音
9178	收传真
9179	发传真
5000	示例 IVR
4000	听取语音信箱
33xx	电话会议, 48K(其中 xx 可为 00-99, 下同)
32xx	电话会议, 32K
31xx	电话会议, 16K
30xx	电话会议, 8K
2000-2002	呼叫组
1000-1019	默认分机号

第七章 连接硬件 SIP 话机

硬件 SIP 话机的品牌有很多，质量也良莠不齐。一般来说，SIP 话机属于「智能」终端，它自己需要处理很多的状态、信令、媒体等，需要比较强大的运算能力。而与此相对的普通模拟话机则简单的多，基础在拨号时每个一号码按键都需要送到后面的交换机进行处理。笔者常用的 SIP 话机有厦门亿联公司生产的 Yealink 话机和深圳潮流公司生产的 GrandStream 系列话机等。

7.1 Yealink

亿联话机是在国内能找到的质量比较好的 IP 话机。话机本身有一个液晶显示屏，并可以通过按键设置账号信息，但那样配置起来比较繁琐。在液晶屏上找到话机的 IP 地址以后¹，用浏览器打开，界面如下图所示：

账号配置跟软电话差不多，「显示名称」可以随便填，「注册名称」和「用户名」这里我们都用 1002，「密码」是默认的 1234，「SIP 服务器」处输入你的 IP 地址，其他的都保留默认设置，然后点击提交。如果一切顺利的话就能看到「账号状态」显示为「注册上」，可以拿机手柄拨叫 1000 或 1001 了。

7.2 GrandStream

GrandStream 也是质量比较好的 IP 话机，在使用 DHCP 的条件下，同样也可以在液晶屏上找到话机的 IP 地址，进而可以在浏览器中进行配置。

某款 GrandStream 话机的配置界面如下图：

图中，我们使用的是 1003 这个 SIP 账号，具体配置项跟其它 SIP 终端都大同小异，在此，不再赘述。

¹一般使用 DHCP 启动后会自动获得一个 IP 地址，否则，也可以设一个静态的 IP。

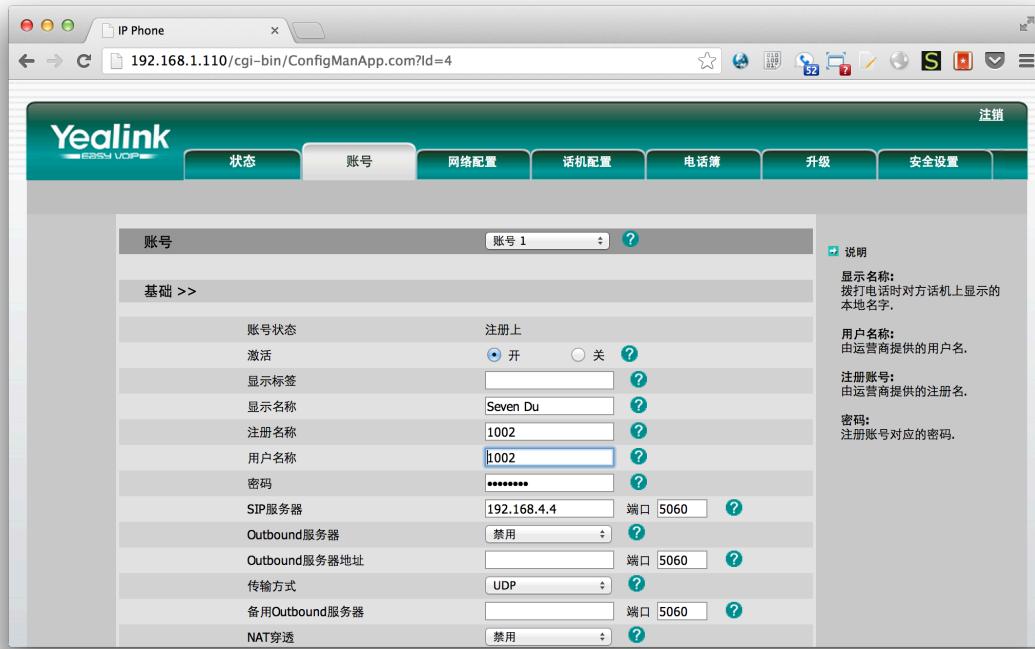


图 7.1: Yealink 话机的账号配置界面



图 7.2: GrandStream 话机的账号配置界面

第八章 对接 IMS

IMS 是新一代通信网络事实上的标准，它旨在在过渡阶段兼容原有的 PSTN TDM 网络（基于电路交换的网络，在 IMS 中称为 CS 域），并最终将所有的业务都转移到基于包交换的网络（PS 域）中。

目前，国内的运营商都部署了 IMS。IMS 网元众多，系统各网元之间通过高速网络相连。就与交换最相关的核心功能来讲，在 IMS 内部，包含 P-CSCF、I-CSCF、S-CSCF 及 AS 等功能与实体。其中 P-CSCF 位于网络的边缘，接收 SIP 消息，并将 SIP 消息转发到内部的 S-CSCF 及 AS 上。

一般来说，IMS 运行在运营商内部专门的网络上（称为承载网），而在外部是无法接触到运营商的内部网络的。如果要通过互联网与运营商的 IMS 对接，就需要通过 SBC 设备及层层防火墙。SBC 设备横跨运营商的承载网及互联网。所以，一般来说，我们可以通过 SBC 连接到 IMS，进而能够与 PSTN 网络上所有的电话进行通信。所以，我们可以把 SBC 设备和 IMS 看成一个 SIP 转 PSTN 的「大」网关。实际上，在实际使用中，大家也不关心是否有 SBC 设备，而只认为运营商有一个「大大」的 SIP 服务器，它可以给我们开放账号。

目前，有些运营商也开始在 IMS 系统上针对一些话务批发商或企业应用放号，因此，如何使用 FreeSWITCH 与其对接就是我们需要研究的了。

8.1 网关配置

当然，与 IMS 对接最好的方式是使用 SIP 中继方式对接，配置简单使用起来也灵活。但很少有运营商允许你这么做。比较典型的方式还是使用类似电话号码的一个一个的账号对接的方式。

下面是一个与云南移动的 IMS 使用账号对接的实例。笔者拿到的是一个试验号码，实际上，对我们来讲，它就是一个 SIP 服务器上的账号。为了与其对接，我们要将 FreeSWITCH 当作一个 SIP 客户端注册到 IMS 上去。在 FreeSWITCH 中，这可以通过添加一个网关实现。下面是一个网关的配置文件：

```
<gateway name="ims">
<param name="realm" value="ims.yn.chinamobile.com"/>
<param name="register-proxy" value="211.139.x.x"/>
```

```
<param name="username" value="+86871xxxxxxxx@ims.yn.chinamobile.com"/>
<param name="password" value="1234"/>
<param name="from-user" value="+86871xxxxxxxx"/>
<param name="from-domain" value="ims.yn.chinamobile.com"/>
<param name="register" value="true"/>
<param name="outbound-proxy" value="211.139.x.x"/>
</gateway>
```

在网关的配置参数中，「ims.yn.chinamobile.com」即为云南移动内部的域。在公网上，它并不是一个合法的域名。那么，我们怎么能注册到该服务器上去呢？实际注册的服务器地址是用「register-proxy」定义的，这里，「211.139.x.x」应该是一个 SBC 的地址。以前，我们在配置网关时都不写这个「register-proxy」参数，它默认就等于「realm」。

「username」是在 IMS 中的用户账号，它和「password」配合用于鉴权（前者将出现在 SIP 消息中的「Authorization」头域中）。该用户账号中包含了用户所属的域，其用户名部分是一个 PSTN 网络中的 E164 格式的电话号码，前面几位中的「+86」表示中国的国家代码，「871」是云南昆明的区号，后面的「xxxxxxxx」则是一个本地的电话号码。

「from-user」指定在 SIP 消息中的源用户信息，「from-domain」则是指定域，它们会影响 SIP 中的「From」头域。

「register」的值为「true」表示 FreeSWITCH 会向该网关发起注册。

前面的信息都是与注册相关的，最后的「outbound-proxy」表示呼叫（即 INVITE 消息）应该发到什么地址，它可以是与注册服务器不同的地址，不过，在本例中，它与「register-proxy」是相同的。

当然，上述这些参数的值都是在 IMS 系统上配置的。从 IMS 上取得正确值并进行配置后，我们就从 FreeSWITCH 注册到 IMS 上对外打电话了。

8.2 通过 IMS 呼出

当然，我们可以很快地在 FreeSWITCH 中输入以下命令试一试它是否真的能打到我们的手机上：

```
originate sofia/gateway/ims/0186xxxxxxxx &echo
```

电话接通后，应该能听到自己的声音（回音）。注意，上述账号就相当于一个云南昆明本地的电话号码，因而拨云南以外的手机号需要加0。

在 Dialplan 中，使用如下设置就可以通过上面的网关往外打电话了：

```
<extension name="IMS gateway outbound">
    <condition field="destination_number" expression="^(0.*$)">
        <action application="bridge" data="sofia/gateway/ims/$1"/>
    </condition>
</extension>
```

当然，如果运营商给我们开的账号支持号码透传的话，我们就可以透传任何号码了，如，我们也爽一把，透传个美国的 911 出去（注意，最好不要透传 110）：

```
<action application="set" data="effective_caller_id_number=911"/>
<action application="bridge" data="sofia/gateway/ims/$1"/>
```

其实，有意思的是，有的运营商开的账号默认就是支持透传的，这样，就会将 FreeSWITCH 内部的分机号（如 1000）透传出去。为了能对所有呼出的电话都显示运营商给我们分配的号码（这里的「xxxxxxxx」），我们可以使用如下 Dialplan 设置：

```
<action application="set" data="effective_caller_id_number=xxxxxxxx"/>
<action application="bridge" data="sofia/gateway/ims/$1"/>
```

8.3 通过 IMS 呼入

在我们正确向 IMS 注册后，这种号码也支持呼入的。如果有人呼叫该号码（即这里的「xxxxxxxx」），IMS 就会给我们的 FreeSWITCH 发 SIP INVITE 消息。收到后，就可以在 FreeSWITCH 中进行路由了。一般来说，呼入的 DID 是不带区号的本地号码，就是上面网关配置中的「xxxxxxxx」部分，然后，我们就可以设置如下 Dialplan 将来话路由到一个 IVR：

```
<extension name="IMS gateway inbound">
    <condition field="destination_number" expression="^(xxxxxxxx)$">
        <action application="answer" data="" />
        <action application="ivr" data="demo_ivr" />
    </condition>
</extension>
```

当然，如果我们很神奇地从运营商那里获得了一个号段（如一个千群或一个万群），也可以把这个号码做一也我们内部的分机号做成一对一的关系。如，下面 Dialplan 设置可以将来话 DID 的后 4 位与我们内部的分机号一一对应：

```
<extension name="IMS gateway outbound">
  <condition field="destination_number" expression="^xxxx([0-9]{4})$">
    <action application="answer" data="" />
    <action application="bridge" data="user/$1"/>
  </condition>
</extension>
```

8.4 其他问题

与这种 IMS 对接的难点就是调试比较困难。一般来说，也很难找到对端真正懂技术的技术人员配合。所以，在对接遇到问题时，解决的方法基本上就是抓包，看 SIP 消息，然后瞎猜，把所有可能的参数都试一遍。

在一次与 IMS 的对接测试中，发现主叫听不到被叫方的回铃音。经过无数次的探索，终于发现，对方需要一个特殊的 SIP 消息头「P-Early-Media: supported」。该消息头是在 RFC5009¹中定义的。在 FreeSWITCH 的 Dialplan 中，可以使用「sip_h_」开头的通道变量添加扩展的 SIP 消息头，实现添加上述 SIP 消息头的 Dialplan 设置如下：

```
<action application="bridge" data="{sip_h_P-Early-Media=supported}sofia/gateway/ims/$1"/>
```

另外，一般来讲，SIP 中的「1xx」响应消息是不需要证实的，而在 IMS 系统中，有时「183」消息也是需要证实的，否则无法听到正常的回铃音甚至无法正常接续。与普通的证实消息 ACK 不同的是，对于「1xx」的消息需要使用 PRACK²（即 Pre-ACK）消息证实。通过在 Profile 中开启如下参数可以让 FreeSWITCH 在收到「183」时发送 PRACK 证实消息：

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

通过这些配置，FreeSWITCH 就可以完美地与 IMS 系统对接进行呼入呼出了。

¹ 参见<http://tools.ietf.org/html/rfc5009>。

² 参见<http://tools.ietf.org/html/rfc3262>。

第九章 连接模拟话机和模拟中继线

在实际应用中，不可避免地要连接模拟话机。这里说的模拟电话机就是我们在家里或公司中常见的普通电话机。当然，在没有 FreeSWITCH 之前，我们的电话机就是通过一根普通模拟电话线连接出去的，并通过这条电话线往外打电话。

现在，我们有了 FreeSWITCH。那我们就会想到两个问题：

- 我们能不能把我们的模拟话机也连接到 FreeSWITCH 上呢？这样，模拟话机就可以与我们的 SIP 软电话（或硬电话）通话了；
- 我们的 FreeSWITCH 能不能通过模拟电话线也往外打电话呢？甚至，别人也可以通过该电话线对应的号码叫我们？

答案是肯定的。下面，我们就来看一下这两个问题是怎麽解决的。不过，在这之前，我们先来看一下一些与模拟电话相关的概念。

9.1 基本概念

普通电话机是模拟的，通过一根模拟线连接到距离最近的电话交换机上。根据所处位置的不同，这个交换机可能是处在运营商机房的交换机，也可能是运营商设在用户小区的一个模块局中的交换机，或者，可能是本企业自己建设的用户小交换机。近几年，响应国家光进铜退的号召，有些运营商就直接将光纤拉到用户家里，通过一个「小盒子（因特网接入设备，称为 IAD）」将光纤接口转成普通的以太网口和模拟电话接口，然后再在模拟电话接口上接电话机。总之，我们的普通电话都要通过一根电话线连接到某个设备上。该设备提供一个模拟电话线的接入端口，在技术上，称为 FXS 口。同时，我们的电话机上也有一个模拟线的接口，称为 FXO（在物理上，就是一个 RJ11 接口¹）。连接方式如图 9.1 所示。

从常识来讲，我们知道，电话线是带电的，即使家里停电了也可以打电话，如果不小心摸到裸露的电话线也会有「触电的感觉」；而话机本身是不带电的，直接在话机上接上电话线并也不会感觉到有

¹RJ11 接口最多支持 4 根线，不过，模块电话只需要用到其中的 2 根。常见的以太网接口为 RJ45 接口，有 8 根线，不过一般只是用到其中的 4 根。

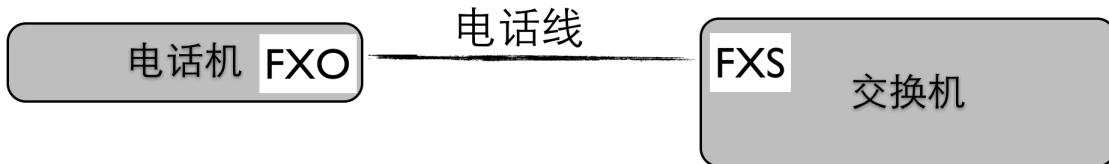


图 9.1: FXS 和 FXO 连接示意图

电²。由此，我们可以联想到，**FXS 口是带电的，而 FXO 口是不带电的**。这也是两者的根本区别。

实际上，FXS 提供的电压是负 48 伏的直流电，主要是为了能感知话机摘挂机的变化、给话机提供拨号音、振铃及其他信号音等。在摘机状态下，电压将降至³负 7 伏左右，而在振铃状态下，电压可能增大至负 90 伏。读到这里，也可以这样认为——**FXS 与 FXO 的区别是，前者能提供拨号音**。

注意，在实际使用中一定不要将两个 FXS 口用电话线连接在一起，两者互相供电，后果是不堪设想的。

9.2 拓扑结构

有了上述基本概念以后，我们就可以来看一下要解决我们的问题需要的拓扑结构。如图 9.2 所示，我们把图 9.1 中的电话线从中间截断，中间放上 FreeSWITCH。FreeSWITCH 提供一个 FXO 接口用于连接原来的交换机；另外提供一个 FXS 接口用于连接原来的模拟电话。此外，其他的 SIP 话机也可以通过以太(Ethernet) 网口接进来，互相通信。

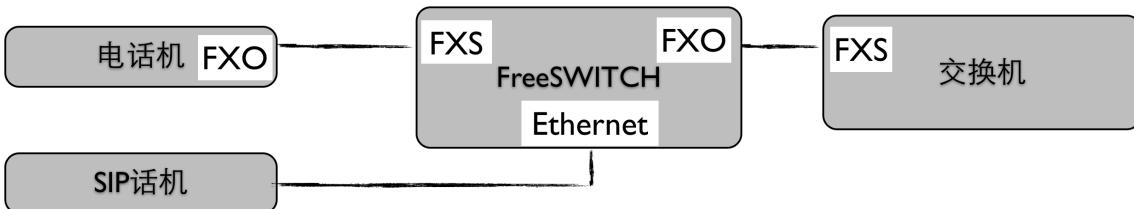


图 9.2: FreeSWITCH 连接 FXS 及 FXO 的拓扑结构

FreeSWITCH 本身是一个软件，因而它是不能提供 FXO 和 FXS 硬件接口的。要解决这一问题，有以下两种方案：

- 1) 在 FreeSWITCH 所在的服务器上安装相关的硬件板卡，该硬件板卡负责提供连接 FXO

²有些话机需要安装电池或接变压器电源，这些电源主要是为了支持话机的本地显示屏等功能用的，如支持「来电显示」等。远程交换机上的馈电的功率不足以支持这些功能。

³在通信设备中一般使用负电压。注意，这里说的是电压的绝对值，虽然-48 在数学上小于-7，但对于电压的绝对值来说，从 48 到 7 是下降了。

及 FXS 的硬件接口。FreeSWITCH 就可以通过相应的驱动程序去控制这些板卡。因而相当于给 FreeSWITCH 增加了 FXO 及 FXS 的接入能力。

这类板卡中比较有代表性的有 Sangoma 以及 Diguim 的生产的模拟和数字板卡等。国内也有许多厂商生产所谓的「Asterisk 兼容卡」，他们有的也能与 FreeSWITCH 搭配使用。

2) 通过外部网关⁴来实现。有些厂商针对这一问题专门生产了支持 SIP 到模拟线的转换网关，称为模拟网关。对于 FreeSWITCH 来讲，这种网关就是一个普通的 SIP UA；而对于模拟话机来讲它就是一个电话交换机；对于电话交换机来讲，它就相当于一个普通模拟话机⁵。

至于如何在以上两种方案之间选择，没有标准的答案。如果有的话，也是——具体问题具体分析⁶。不过，一般来说，笔者建议：对于模拟线路，选用第 2 种方案，因为这些网关设备很容易买到，而且坏了可以随时更换，兼容性也比较强；而第 1 种方案中支持这些板卡的驱动往往需要编译内核等，操作起来比较复杂。后面我们讲到用于数字线路的数字板卡时则可以考虑使用第 1 种方案，性价比和可控性会高一些。

增加了网关后的拓扑结构如图 9.3 所示。与图 9.2 比较起来，相当于在 FreeSWITCH 中把内置的板卡换成了外置的网关设备。

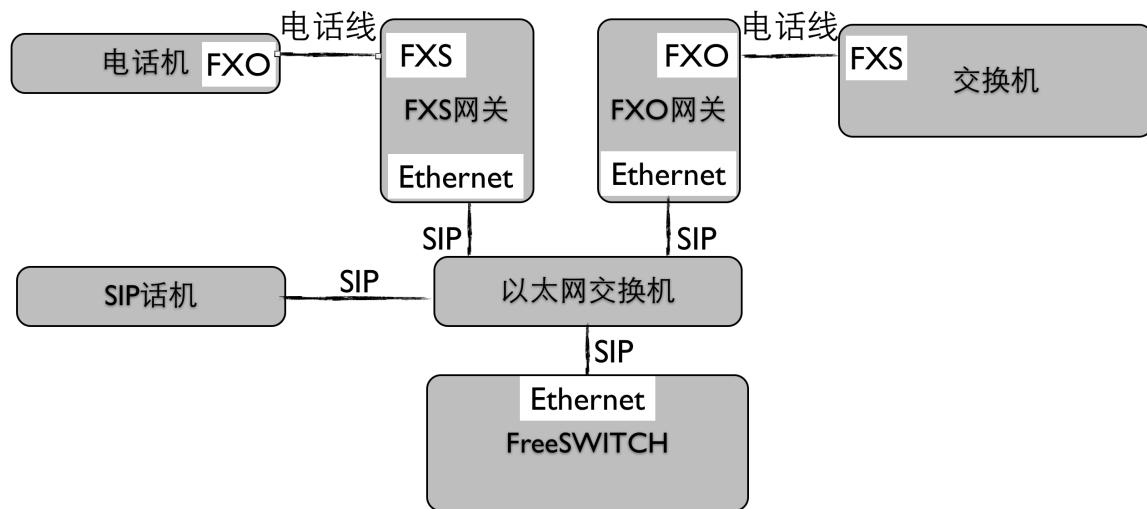


图 9.3: FreeSWITCH 通过网关连接模拟话机及电话线

在实际应用中，有单纯的 FXS 网关及 FXO 网关，端口数从 1 口、2 口、4 口到几十口不等。也有的网关是 FXS 和 FXO 混合装在同一设备上的，根据实际需求可以选用不同的网关。

下面，我们先来看一下模拟网关的配置和使用。

⁴一般来说，处于设备的中间，用于连接两个不同网络或不同协议的设备都称为网关设备。根据网关支持的不同的接口或协议类型以及应用场景又分别叫不同的名字，如模拟网关，E1 数字中继网关等。我们在这里主要讨论 SIP 网关，因此一般来说网关一端的协议是 SIP 的。当然，对于完成同样功能的设备有时也有不同的叫法，如，某些型号的路由器设备就支持模拟电话接口，因而也具有网关的功能。另外，有些设备（华为公司生产的终端设备）也称为 IAD（Internet Access Device，即因特网接入设备），也是能完成模拟网关的功能，并可能还支持其他的协议。

⁵也就是说，带有 FSX 口的设备就相当于一个电话交换机，带有 FXO 口的设备就相当于一个普通模拟话机。

⁶常见的答案是 It Depends。即应该根据具体的使用场景来选择。

9.3 通过 Grandstream 网关连接模拟话机

我们先来看一下如何将话机通过模拟网关连接到 FreeSWITCH 上。潮流网络公司有一款型号为 HT701 的单口模拟网关，它有一个 FXS 口和一个以太口，FXS 口用于连接话机，以太口用于通过以太网连接 FreeSWITCH。如果把它和与之相连的模拟话机看成一体的话，实际上就相当于一个 SIP 话机。也可以说，这款模拟网关能把普通的模拟话机「变」成 SIP 话机。该网关小巧方便，比较适合在桌面上使用。

该网关有一个简单的 Web 配置界面⁷，如图 14-5 所示。首先切换到「FXO PORT」配置页面，通过相关配置，可以令其向网关注册。读者在图中可以看到，它其实跟图 3-12 中亿联话机的账号配置界面是类似的（因为该网关的功能就相当于一个 SIP 话机）。其中，Primary SIP Server 填入我们 FreeSWITCH 服务器的 IP 地址；Failover SIP Server 是一个备份服务器，用于在 Primary SIP 服务器出现故障的时候自动倒换到 Failover 指定的服务器上，在这里我们不使用，可以不填；SIP User ID 即我们注册的账号，在这里我们使用 FreeSWITCH 默认提供的账号 1000；Authenticate ID 为认证 ID，跟账号一样；Authenticate Password 即密码，填入 1234，Name 为 SIP 中的显示的名字，可以随便起一个；其他的都保留默认配置就可以了。

Grandstream Device Configuration			
STATUS	BASIC SETTINGS	ADVANCED SETTINGS	FXS PORT
Account Active: <input type="radio"/> No <input checked="" type="radio"/> Yes			
Primary SIP Server: <input type="text" value="192.168.1.119"/> (e.g., sip.mycompany.com, or IP address)			
Failover SIP Server: <input type="text"/> (Optional, used when primary server no response)			
Prefer Primary SIP Server: <input type="radio"/> No <input checked="" type="radio"/> Yes (yes - will register to Primary Server if Failover registration expires)			
Outbound Proxy: <input type="text"/> (e.g., proxy.myprovider.com, or IP address, if any)			
SIP Transport: <input checked="" type="radio"/> UDP <input type="radio"/> TCP <input type="radio"/> TLS (default is UDP)			
NAT Traversal: <input checked="" type="radio"/> No <input type="radio"/> Keep-Alive <input type="radio"/> STUN <input type="radio"/> UPnP			
SIP User ID: <input type="text" value="1000"/> (the user part of an SIP address)			
Authenticate ID: <input type="text" value="1000"/> (can be identical to or different from SIP User ID)			
Authenticate Password: <input type="text" value="1234"/> (purposely not displayed for security protection)			
Name: <input type="text" value="Seven Du"/> (optional, e.g., John Doe)			

图 9.4: Grandstream HT701 的配置

正确配置完上述选项后，就可以切换到「Status」页面查看注册状态。如果注册正常，拿起相连的模拟话机的话筒就可以听到拨号音，然后就可以像正常的 SIP 话机一样打电话了。别人拨打 SIP 账号 1000 时，模拟话机也会振铃。

⁷其实使用这类 Web 界面最大的一个问题往往是如何先找到该设备的 IP 地址。不同厂家的设备都有不同的默认地址或者查找 IP 地址的方法，在使用时请参考相关设备说明书或相关资料。在此我们就不多介绍了。下同。

9.4 通过迅时网关连接模拟话机和模拟中继线

上海迅时的网关设备也是国内市场上常见的设备。在本节，我们以迅时 MX8 为例来讲解一下。MX8 根据情况可以配置 8 个 FXS 口，也可以配置 8 个 FXO 口，笔者使用的一款是配有 4 个 FXS 和 4 个 FXO 口的网关。

9.4.1 配置 FXS 口

MX8 网关 FXS 口的功能和配置跟我们上面讲到的潮流 HT701 差不多。首先，连接 MX8 的 Web 配置界面后，依次选择「基本配置 1」→「SIP」→「注册服务器和代理服务器」菜单项，在配置页面上选择注册方式为「按线路注册」，使用这种注册方式可以单独配置每个 FXS 口对应的 SIP 账号，相当于 4 个独立的 SIP 话机。

配置好 SIP 服务器的地址后，再转到「线路配置」→「用户线功能」页面，如图 14-6 所示。在线路号码中选择一个 FXS 端口，如 FXS-1；电话号码即我们的 SIP 注册账号，在图中的例子我们使用 1007，勾选「注册」复选框，然后在密码栏中输入「1234」，提交后，就可以成功注册到我们的 SIP 服务器上了。



图 9.5: MX8 网关 FXS 口的配置

如果有多个模拟话机，可以依次注册其他的。拿起话机听到拨号音，便可以类似普通 SIP 电话一样对外呼叫。当然，模拨话机与 SIP 话机毕竟是不同的，这一点尤其体现在拨号方式上，要理解这一点需要了解一下我们下一节中讲到的拨号规则。

9.4.2 拨号规则

在上一节我们讲到，SIP 话机与普通模拟话机的拨号方式不同。在普通的模拟电话中，拨号时，按下的每一位号码都会实时的传到后端的交换机上。因而在打电话时只需要拨完号码等着就行了，交换机在收齐相关号码后会自动帮我们接续。而 SIP 话机是比较「智能」的设备，它能本地存储号码，在收齐所有号后需要用户按一个「发送」键才能将被叫号码送出去。由于这一点，在刚刚从模拟电话换到 SIP 话机后，往往会很不适应——经常在拨完了号码后忘了按发送键，瞎等半天没有任何反应。不过，这一点倒是跟手机有些类似，手机也是在收齐了号码后需要按「发送」键才能拨出的。

这里我们还是讨论模拟话机。在用户摘机后，相连的交换机就开始向话机播放拨号音并启动收号程序。一般来说，电话号码是有规律的。比方说 110，交换机知道它是一个短号码，收到三位 110 后就不会等待下面的拨号了，因而保证比较快速的接续。另外大家熟悉的手机号码也都是定长的，所以交换机在收齐足够的位数后就会快速接续。

现在，我们将模拟话机接到 MX8 网关上，因而该网关就需要具备收号功能。为了能保证快速的接续，它定义了一些拨号规则。如果所拨的号码匹配该拨号规则，就会将号码立即送出去（即向 SIP 服务器发送 INVITE 消息）。笔者使用的网关默认的拨号规则如下：

```

01[3,5,8]xxxxxxxx
010xxxxxxxx
02xxxxxxxx
0[3-9]xxxxxxxx
120
11[0,2-9]
111xx
123xx
95xxx
100xx
1[3,5,8]xxxxxxxx
[2-3,5-7]xxxxxx
8[1-9]xxxxx
80[1-9]xxxx
800xxxxxxxx
4[1-9]xxxxxx
40[1-9]xxxx
400xxxxxxxx
x.T
x.#
#xx
*xx
##
```

可以比较容易地看出，其中第 1 行是匹配常见的手机号，对 2 行匹配北京的固定电话号码，其他的依此类推。我们在这里不介绍这些规则的具体含义，读者如果在使用时可以自行参考网关设备的参

考手册。在这里，需要注意的是，这些默认的拨号规则不一定适合你的需要。比如，如果你想拨打内网的一个分机号 1200，由于在规则匹配时匹配到第 5 行的 120 就终止了，因而 INVITE 消息中的被叫号码就只是 120，只要有这条规则存在，1200 就永远也拨不出去⁸。

当然，如果你不理解这些的具体含义又一时找不到手册的话也不要紧，可以尝试把这些规则全删掉，而在拨号时拨完所有号码后在后面再加上一个「#」号，MX8 网关收到「#」号后就会立即将前面收到的号码送出，而不包含最后的「#」号（当然，如果被叫号码中要求包含「#」号的情况下还是可能会有问题，那时候就真需要参考设备手册了）。

9.4.3 配置 FXO 连接外线

FXO 口可以连接运营商提供的电话线（我们称为外线）与外界通话。在 MX8 中，连接到 FXO 口的电话线称为中继线。切换到「线路配置」→「中继线功能」可以看到如图 9.6 所示的界面。在这里，在线路号码中选择一个端口，如 FXO-1；外线号码填入运营商给我们分配的电话号码（实际上在这里可以是任意号码，它只是作为一个标志使用）；我们不像 FXS 那样使用注册方式向 FreeSWITCH 注册，而是使用「中继对接」方式与 FreeSWITCH 对接，因此，我们不选注册复选框；接入方式选择「绑定」，并输入绑定号码（该绑定号码一般应该是运营商提供给我们的电话号码，但同样，也可以是做任意值，我们这里以 88888888 为例）。

线路号码	FXO-1
外线号码	88888888 不超过20位
注册	<input type="checkbox"/>
接入方式	绑定
绑定号码	88888888 不超过20位
<input type="checkbox"/> 反极信号检测 <input checked="" type="checkbox"/> 来电号码检测 <input type="checkbox"/> 禁止呼出 <input checked="" type="checkbox"/> 回音消除 <input type="checkbox"/> 延迟发送接通消息(与“高级配置 > 中继线特性”界面“呼出接通延时”配置项配合使用)	

图 9.6: MX8 网关 FXO 口配置

通过使用绑定方式，可以我们在 FXO 口有来话时自动路由到 FreeSWITCH 上（后面会讲到具体配置）。另外的方式是选择二次拨号方式，我们先来讲一下二次拨号方式。

注意，FXO 口的功能相当于一个电话机，如果有人拨打该外线对应的电话号码（为了简单起见，我们以 88888888 为例），它是无法知道我们在该外线上到底接了一个电话机还是模拟网关的。当然，如果我们直接在外线接了一个电话机，有人打电话时就会振铃，我们就可以进行接听。但是，这里，我们接了一个模拟网关，如果有电话呼进来，模拟网关就可以代替我们接听。但接听后下一步该怎么办呢？一般来讲，它可以给用户再放一个拨号音（有的网关可以设置在这里有一个比较简短的语音提示，称为 IVR），这样，主叫用户就可以通过再次通过 DTMF 按键进行拨号以拨打我们的内线号码。

当然，既然我们已经有了 FreeSWITCH，我们就可以用 FreeSWITCH 做任何我们想做的事。所以，这里我们选择绑定方式。在绑定号码 88888888 后，对于任何外线来话，该网关都会向 FreeSWITCH 发送 INVITE 消息（在设置了正确的路由的情况下），被叫号码为 88888888。FreeSWITCH 在收到

⁸通常的解决办法是去掉该行规则，如果拨打真正的外网的 120 时拨 0120 并通过相关的号码变换规则实现。

后，就可以进入 Dialplan，播放 IVR，引导来电用户进行下一步的操作。在这种情况下，FreeSWITCH 需要先对来电应答（发送 SIP 200 OK 消息），这样，网关收到 200 OK 消息后才应答进来的呼叫（相关于我们摘机），并建立通话。

当然，FreeSWITCH 也可以自己直接将该来电桥接（bridge）到一个内线号码号码上，让该内线号码振铃。如果有多条外线，可以分别插在不同的 FXO 口上，对内也可以一一对应不同的内线电话（分机）。当外面的人拨打这些外线号码时，电话就能从某一个 FXO 口进来，并最终向对应的内线电话振铃。这种方式，就称为 DID（Direct Inbound Dialing，对内直接呼叫）。同时这些外线号码就称为「DID 号码」，也简称 DID。广义上讲，即使对这些外线来话不是一一对应到内线分机上的，也称为 DID，在这种情况下，可以认为 DID 就是一个外线接入号，是一个被叫号码。

这儿配置好了以后，我们还不足以建立它与 FreeSWITCH 通话。要灵活地对来话去话进行处理，我们还要用到路由的概念。

9.4.4 配置路由

MX8 网关通过路由机制对各种设备的来、去话进行控制。可以通过编辑路由表配置这些路由。路由表的语法也很简单。首先进入「拨号及路由」页面，我们添加如下的路由：

IP	1007	ROUTE	FXS	1
FXO	x	ROUTE	IP	192.168.1.9:5080
IP	x	ROUTE	FXO	5,6,7,8

其中，第一行表示，所有从 IP 来的呼叫，如果被叫号码是 1007，则将该呼叫路由到 FXS 的第 1 个口上；第二行表示，所有从 FXO 来的呼叫，不管被叫号码是什么，统一路由到一个 IP 地址上，后面，我们就的是我们 FreeSWITCH 服务器的地址 192.168.1.9，并且，我们在这里使用 5080 端口，以避免对网关发来的 INVITE 请求进行鉴权。

下面，我们分几种情况进行说明。

1) 模拟话机做主叫。当它发起呼叫时，由于我们在这里使用了向 FreeSWITCH 「注册」的方式，因而相当于网关设备把它转换成了一个 SIP 话机，网关设备就会向 FreeSWITCH 发起 INVITE 请求，接下来的接续流程就是跟 SIP 话机做主叫是一样的，呼叫进入 FreeSWITCH 的 Dialplan，然后进行下一步的路由。

2) 模拟话机做被叫。如果在 FreeSWITCH 中有人呼叫该模拟话机的号码（如 1007），FreeSWITCH 只是认为它就是一个普通的 SIP 用户，因而会找到网关设备注册时的 Contact 地址（即网关的 IP），并向该网关发 INVITE 请求。网关收到 INVITE 请求后，查找路由表，并匹配到上述路由表中的第一行，因而，连接在相关的 FXS 口的模拟话机就会振铃。

3) 接受模拟外线呼入。如果外面的电话从外线呼入，来话就到达网关的 FXO 口。网关从上述路由表中查到第 2 行所示的路由，并向 192.168.1.9:5080 发送 INVITE 请求。FreeSWITCH 在收到 INVITE 请求后，会进入路由阶段，我们在日志中就可以看到类似如下的输出：

```
Processing 139xxxxxxxx <139xxxxxxxx>->88888888 in context public
```

注意，其中的 88888888 就是我们设置的 DID。因此，为了能处理该通话，我们需要在 Public Dialplan 中添加类似如下的配置：

```
<extension name="DID">
<condition field="destination_number" expression="^88888888$">
    <action application="info" data="" />
    <action application="ivr" data="welcome" />
</condition>
</extension>
```

上述配置可以在日志中显示来话 Channel 的相关信息，并执行 IVR，提示来电用户进行下一步操作。

4) 通过模拟外线呼出。如果需要通过模拟外线呼出，则需要首先在 FreeSWITCH 中做一条路由，将去话路由到该网关上。在这种情况下，FreeSWITCH 将 MX8 网关看成一个外部网关，因而我们可以在 FreeSWITCH 中添加一个网关（其中 192.168.1.8 为 MX8 网关的 IP 地址）：

```
<gateway name="mx8">
<param name="realm" value="192.168.1.8" />
<param name="register" value="false" />
</gateway>
```

其中，FreeSWITCH 仅把 MX8 网关看作一个以中继方式对接的设备，因而不需要向其注册（「register=false」）。

然后，就可以使用如下的 Dialplan 将本地用户的去话路由出去。

```
<extension name="DID">
<condition field="destination_number" expression="^0(.*$)">
    <action application="bridge" data="sofia/gateway/mx8/$1" />
</condition>
</extension>
```

另外，在这种中继对接的模式下，我们也可以不在 FreeSWITCH 中添加网关，而是直接把去话送到网关的 IP 的地址上，如：

```

<extension name="DID">
    <condition field="destination_number" expression="^0(.*)$">
        <action application="bridge" data="sofia/external/$1@192.168.1.8"/>
    </condition>
</extension>

```

通过上面的 Dialplan 可以将 FreeSWITCH 中的电话送到 MX8 上，然后，MX8 收到请求后查找到以下的路由表项：

IP	x	ROUTE	FXO	5,6,7,8
----	---	-------	-----	---------

该路由表表示，从 IP (以太网口) 进来的呼叫，不管被叫号码是什么 (x 代表任意号码)，全部路由到 FXO 口上，FXO 端口的选择顺序依次是 5、6、7、8。

注意，笔者使用的 MX8 有 4 个 FXS 口和 4 个 FXO 口，FXO 口的端口范围是 5 ~ 8。在呼出时，如果检测到某个端口忙，则顺序会选择下一个端口进行呼出。

9.4.5 其他

对于网关来讲，还有其他的一些常用功能，比较典型的如主叫号码检测。对于一个从外线 (FXO 口) 进来的来话而言，如果我们在外线上开通了来电显示功能，则可以在向 SIP 服务器发送 INVITE 之前检测主叫号码。

一般来说，现行的模拟线路都使用 FSK⁹方式来传送主叫号码。主叫号码在第一声振铃和第二声振铃的时间隔内传送。所以，如果使用主叫号码检测，接续时间就会长一点。从图 9.6 中可以看出我们勾选了「来电号码检测」功能。

开启了来电号码检测后，如果外线没有开通来电显示功能（通常该功能作为运营商的增值业务，是要单独收费和开通的），则肯定检测不到主叫号码。MX8 网关在这时通常使用我们「绑定」的号码做为 SIP 侧的主叫号码。

另一个值得一提的功能是「回声消除」。回声一般是在话机等终端设备上产生的，典型的场景就是从话筒中传出的声音又传回到了麦克风里去了（相当于声音反射，称为回声），这样，对端就会在电话中听到自己的声音，听起来会感觉不舒服¹⁰。如果设备有较好的回声消除功能，它可以检测麦克风的输入，如果里面包含话筒中刚刚输出过的内容（它应该能记住一段时间的内容），则自动在传到对方之间将这部分声音数据去掉。这种技术就称为回声消除。

在图 9.6 中我们也勾选了回声消除功能，它在检测到回声时能起到一定的作用。另外，一副比较好的耳机也能大大减少回声（声音的输出直接传的耳朵里，很少会扩散到麦克风中）。

⁹Frenquency Shift Keying，即移频键控。参考…

¹⁰所以，产生回声的原因是对端终端的问题，即，如果你能听到回声，那一定是对端的终端设备将你的声音又反射了回来。

在本节我们以迅时 MX8 网关为例子讲了模拟网关的配置以及与 FreeSWITCH 的对接。其他厂商的网关也有类似的功能，只是具体的配置和实现方式不同。具体的应用可以参考相应的设备手册。

第十章 通过 E1 线路对接

前面，我们讲过通过 IMS 或模拟中继线连接 PSTN 网络的例子。但很多时候，我们还是需要使用 E1 与其他系统对接。很典型地，某些运营商可能只提供 E1 线路，或者某些设备只提供 E1 接口。

类似我们在第 9.2 节讲到的拓扑结构和对接方式，我们也可以使用 E1 网关或板卡来与其他系统对接。在 9.2 节，我们只讲了使用网关方式对接的例子。在这里，我们把两种对接方式都讲一下，以帮助读者有一个全面的了解。

我们将搭建如下实验环境。其中 FS1 和 FS2 分别是两台装有 FreeSWITCH 的主机，它们分别具有 IP 地址 192.168.1.115 和 192.168.1.119。FS1 上装有 E1 板卡，可以通过 E1 线¹与 E1 网关相连，同时 E1 网关又通过以太网（使用 SIP 协议）与 FS2 相连。

其中 FreeSWITCH 我们使用默认的安装和默认的配置。下面我们就来看一下需要经过哪些步骤可以让分别注册到 FS1 和 FS2 上的 SIP 用户互相打电话。

10.1 配置 FS1

首先，我们来看一下 FS1 上的配置。FS1 上需要安装硬件板卡，安装驱动程序等，因而这一部分配置是比较复杂的。而且，我们也将在这里学到一些新的名词术语和概念等。

10.1.1 背景资料

传统的电话交换机都是以时分复用（TDM，Time Division Multiplex）方式工作的，因此，与传统交换机相连的设备及板卡都统称为 TDM 设备或 TDM 板卡。在 FS1 上，我们需要使用 E1 板卡来支持 E1 接口。生产这类 E1 板卡的厂商有 Sangoma 和 Digium 等（当然，它们也生产模拟板卡），国内也有一些生产这种兼容卡的厂商。从历史上来讲，Digium 成立于 1999 年，算是老牌子的板卡厂商，也就是它推动了 Asterisk 开源 PBX 软件的发展。Sangoma 是一家创立于 1984 年的 VoIP 系统供应商，也是相当有历史。Sangoma 从 FreeSWITCH 诞生起就一直与 FreeSWITCH 社区深度合作，并且贡献了 FreeTDM 软件和 mod_freetdm 模块，用于配合各种硬件板卡在 FreeSWITCH 上工作。

¹设备之间对接需要使用 E1 交叉线。E1 线有 4 条线组成，其中 2 条是收（接收），2 条是发（发送）。交叉线的含义就是一端的「收」要接到对端的「发」上。物理接口一般有两种，一种是 RJ48（物理上跟 RJ45 相同，只是线序不同）双绞线接口，阻抗为 120Ω ；另一种是同轴电缆接口，阻抗是 75Ω 。也有在这两种物理接口转换的适配电缆。

最初，这些硬件板卡都是靠 Zaptel²来驱动的，FreeSWITCH 中有一个对应的模块叫 mod_openzap³。后来，Sangoma 公司贡献了并维护着 mod_freetdm 模块，因此就很少有人用 mod_openzap 了。同时，Digium 也将 Zaptel 更名为 DAHDI⁴，并继续进行了改进。该协议主要在 Digium 的板卡中使用，国内也有一些「Asterisk 兼容卡」支持 DAHDI 协议。Sangoma 公司的板卡除了支持 Zaptel 和 DAHDI 外，还支持自己的驱动 Wanpipe。并且，一般认为在并发数较高的情况下 Wanpipe 协议比 DAHDI 表现要好。

FreeTDM 作为 Sangoma 公司贡献的开源软件，可以单独使用，也可以与 FreeSWITCH 配合使用。而与 FreeSWITCH 的配合就是靠 mod_freetdm 将它们联系起来的。

在本质上，FreeTDM 也是一个模块化的结构，我们可以在源代码目录中使用 ls 命令列出 FreeTDM 支持的子模块⁵。在 FreeSWITCH 源代码目录中执行 ls 命令的输出结果如下：

```
$ ls libs/freetdm/src/ftmod/
ftmod_analog/      ftmod_libpri/   ftmod_r2/          ftmod_wanpipe/
ftmod_analog_em/   ftmod_misdn/   ftmod_sangoma_isdn/ ftmod_zt/
ftmod_gsm/         ftmod_pika/   ftmod_sangoma_ss7/
ftmod_isdn/        ftmod_primap/  ftmod_skel/
```

上面列出的这些子模块基本上可以分为两种，信令模块和 IO 模块。其中前者主要负责通话的建立和释放（如 ftmod_isdn），类似于我们已经熟悉的 SIP；而后者主要负责话音数据的输入输出（如 ftmod_wanpipe），类似于我们熟悉的 RTP。

在 IO 层，FreeTDM 可以有两种工作模式，一种是 DAHDI 模式，它是由 ftmod_zt 驱动的，主要兼容 DAHDI 及 Zaptel 协议的板卡；另一种为 Wanpipe 模式，主要用于 Sangoma 生产的板卡。这里我们以 Sangoma 板卡为例来进行说明，因此，主要讲解 Wanpipe 模式。

10.1.2 安装板卡、操作系统和 FreeSWITCH

在此，我们以 Sangoma 数字语音板卡 A101 来作为实践的例子。A101 数字语音板卡是一个单口 E1 板卡，它有 PCI 和 PCI-E 两种接口类型，在购买前要先确定自己服务器所支持的接口类型。另外，Sangoma 也提供附加的回声消除模块，能比较有效地解决语音通信中的回声问题。

将板卡插入主机的 PCI 或 PCI-E 插槽即安装完毕。具体步骤我们就不多讲了。

Sangoma 系列语音板卡可以通过 Wanpipe 驱动很好地工作在 Linux/Windows 之上，在本节我们以 Linux(Ubuntu 12.04 32 位)为例，其他 Linux 及 Windows 用户请对照参考 Sangoma Wiki (<http://wiki.sangoma.com/Wanpipe-Windows-Driver>)。

²Zaptel 指 Jim Dixon 的开放电脑电话硬件驱动 API，后来 Digium 公司生产了相关的板卡并改进了该 API。参见 <http://www.voip-info.org/wiki/view/Zaptel>。

³注意，该模块的源代码不在 src/mod 目录下，而是在 libs/openzap 目录下。

⁴DAHDI 即 Digium Asterisk Hardware Device Interface 的缩写。由于注册商标问题，Zaptel 于 2008 年更名为 DAHDI。参见 <http://blogs.digium.com/2008/05/19/zaptel-project-being-renamed-to-dahdi/>。

⁵为了与 FreeSWITCH 中的模块相区别，我们在这里称 FreeTDM 的模块为「子模块」。

Ubuntu Linux 操作系统的安装采用基本的安装就可以，我们接下来会安装其他依赖的工具和包。**注意，安装的顺序很重要，如果是新手，请一定按这里讲的安装顺序安装。**在这里，Linux 操作系统安装完毕后，再安装好 FreeSWITCH 的基本系统（请参阅相关的安装方法，不再赘述）。

FreeSWITCH 安装完毕后，即可以尝试启动并简单测试一下，如果没有问题则可以继续下面的步骤。

10.1.3 安装及配置 Wanpipe 驱动⁶

接下来我们就来安装板卡的驱动。对于硬件板卡的访问一般需要 root 权限才能进行，因此，我们在这里所有操作都以 root 用户来执行⁷。板卡驱动一部分是需要放在内核中的，因而需要编译板卡的内核驱动程序。在编译时需要依赖于内核的头文件及编译工具。

以前，编译内核模块都需要非常复杂的操作。不过，现在，Sangoma 驱动包中提供的交互式安装脚本已经非常智能，因此，有一点 Linux 基础的人就可以很容易地完成安装。

首先，Sangoma 官方的 Wiki 页面⁸列举了以下一些依赖项，读者可以作一个参考。

```
C development tools ... (gcc)
C++ development tools
Make utility
Ncurses library
Perl development tools
AWK
FLEX
Patch
libtermcap-devel
bison
libtools
autoconf
automake
kernel-devel
```

在 Ubuntu Linux 中，我们可以很方便的使用下列命令安装这些依赖的包⁹：

⁶本章内容写于 2013 年，因此有些软件版本等可能有了更新，但总体步骤差不多，供参考。

⁷在实际使用中，除了板卡的驱动外，FreeSWITCH 是可以以普通用户运行的。只要对相关的设备文件（/dev/wanpipe*）赋予执行 FreeSWITCH 的用户相关的读写权限，就可以在 FreeSWITCH 中加载 mod_freetdm 模块与硬件板卡通信。这一部分属于 Linux 操作系统的知识，超出了本书的范围，有兴趣的读者可以自行研究。

⁸参见<http://wiki.sangoma.com/Wanpipe-Requirements>。

⁹注意，这些包在我们前面安装 FreeSWITCH 的时候有的已经安装过了，列在这里只是为了完整，也方便先安装驱动后安装 FreeSWITCH 的读者参考，而且，多次执行这些命令也不会有副作用。

```
# apt-get -y install gcc g++ automake autoconf libtool make
# apt-get -y install libncurses5-dev flex bison patch libtool autoconf
# apt-get -y install linux-headers-$(uname -r) sqlite3 libsdl1.2debian
```

Sangoma Wiki 下载页面<http://wiki.sangoma.com/wanpipe-linux-drivers> 提供了最新版本的 Wanpipe 驱动，可以使用如下的命令下载：

```
# wget ftp://ftp.sangoma.com/linux/current_wanpipe/wanpipe-current.tgz
```

在撰写本文时，版本是 7.0.5，因此，使用下面的命令解压后，将得到 wanpipe-7.0.5 目录：

```
# tar xvzf wanpipe-current.tgz
```

使用下列命令进入驱动程序目录，编译并且安装：

```
# cd wanpipe-7.0.5
# make freetdm
# make install
```

安装完成后，可以使用「wanrouter」命令来对板卡进行各种操作。如，可以使用下列命令检测系统中安装的板卡：

```
# wanrouter hwprobe
-----
| Wanpipe Hardware Probe Info |
-----
1 . AFT-A101-SH : SLOT=5 : BUS=2 : IRQ=10 : CPU=A : PORT=1 : HWEC=0 : V=37
Sangoma Card Count: A101-2=1
```

上面的输出结果表明，检测到一个类型为 A101 的板卡。其他的参数包括系统总线和中断号等，我们不用太关心。

接下来，可以执行「wancfg」命令进入一个图形界面的配置程序进行板卡的参数配置。不过，笔者发现纯文本界面的「wancfg_fs」配置向导更容易讲解和理解一些。因此，我们在这里以文本界面

的为例进行讲解。「`wancfg_fs`」是一个交互式的配置界面，读者在配置过程中只需要按照相关提示选择1/2/3或直接回车。下面是该命令刚开始执行的输出（为了节省篇幅，我们省略掉了一些提示性的信息）：

```
# wancfg_fs

Would you like to change FreeSWITCH Configuration Directory?
Default: /usr/local/freeswitch/conf
1. NO
2. YES
[1-2, ENTER='NO']:
```

上面的信息是询问 FreeSWITCH 的配置文件目录，如果 FreeSWITCH 在默认的安装位置，可以直接按回车，否则，按1并按回车，它会提示用户输入一个目录。这里，笔者使用的是默认的安装目录，因此，直接按回车。接下来看到如下选择：

```
-----
Configuring T1/E1 cards [A101/A102/A104/A108/A116/T116]
-----
A101 detected on slot:5 bus:2

-----
Configuring port 1 on A101 slot:5 bus:2.
-----

Select media type for AFT-A101 on port 1 [slot:5 bus:2 span:1]
1. T1
2. E1
3. Unused
4. Exit
[1-4]: 2
```

上面提示找到一个A101卡，让我们配置卡的类型。由于我们要对接E1设备，因此选2。回车后提示如下：

```
Configuring port 1 on AFT-A101 as E1, line coding:HDB3, framing:CRC4
1. YES - Keep these settings
2. NO - Configure line coding and framing
[1-2, ENTER='YES']:
```

上面提示将使用E1，线路编码使用HDB3码，我们直接按回车继续。

```
Select clock for AFT-101 on port 1 [slot:5 bus:2 span:1]
1. NORMAL
2. MASTER
[1-2] : 2
```

接着提示选择时钟类型。在与其他 E1 设备对接时，一端使用主时钟，一端使用从时钟，只要两端能配合就行了。因此在实际应用中需要跟对端进行确认（下面有的参数也是如此）。在这里，我们选择 2，使用主时钟（让对方设备使用从时钟）。回车后继续得到如下提示：

```
Select Switchtype for AFT-101 on port 1 [slot:2 bus:3 span:1]
1. EuroISDN/ETSI
2. QSIG
[1-2] : 1
```

上面提示选择交换协议的类型，这个也需要与对端对应，这里我们选用 EuroISDN。安按回车继续：

```
Select signalling type for AFT-101 on port 1 [slot:2 bus:3 span:1]
1. PRI CPE
2. PRI NET
[1-2] : 2
```

上面提示选择信令的类型，NET 为网络设备，CPE 为终端设备，一般与时钟的选择是对应的。由于上面我们选择了主时钟，因此我们这里也选择 NET 设备。输入 2 并按回车继续。

```
Select dialplan context for AFT-101 on port 1
1. default
2. public
3. Custom
[1-3] : 1
```

这里提示选择 Dialplan Context。Dialplan 我们已经非常熟悉了。该选项表示，如果在 E1 端口上有来话，将进入哪个 Dialplan Context 查找路由。在这里我们选择 default，即跟 SIP 电话一样。因此，输入 1 并按回车继续：

```
Input the dialing group for this port
: 1
```

上面提示输入一个组号，由于我们只有一个板卡，输入 1。按回车继续，可以看到该板卡的配置已经完成（complete）了。

```
Port 1 on AFT-A101 configuration complete...
Press any key to continue:
```

提示按任意键继续，我们继续按回车键，接下来配置脚本会尝试寻找并配置其他类型的板卡，如 BRI、GSM 等。由于我们没有安装其他板卡，因此，在下面的询问过程中，可以直接按回车键跳过：

```
T1/E1 card configuration complete.
Press any key to continue:
-----
Configuring ISDN BRI cards [A500/B500/B700]
-----
No Sangoma ISDN BRI cards detected

Press any key to continue:
-----
Configuring analog cards [A200/A400/B600/B610/B700/B800]
-----
Configuring USB devices [U100]
-----
#####
#          SUMMARY          #
#####

1 T1/E1 port(s) detected, 1 configured
0 ISDN BRI port(s) detected, 0 configured
0 analog card(s) detected, 0 configured
0 GSM card(s) detected, 0 configured
0 usb device(s) detected, 0 configured

Configurator will create the following files:
1. Wanpipe config files in /etc/wanpipe
2. freetdm config file /usr/local/freeswitch/conf/freetdm.conf
3. freetdm_xml config file /usr/local/freeswitch/conf/freetdm.conf.xml

Your configuration has been saved in /etc/wanpipe/debug-2013-09-25.tgz.
When requesting support, email this file to techdesk@sangoma.com
```

```
Configuration Complete! Please select following:  
1. YES - Continue  
2. NO - Exit  
[1-2]:1
```

最后，显示全部配置都完成了，询问是否保存这些配置：

```
Wanpipe configuration complete: choose action  
1. Save cfg: Stop Wanpipe now  
2. Do not save cfg: Exit  
[1-2]:1
```

```
Stopping Wanpipe...
```

如果选 1，则停止 Wanpipe 驱动，并保存信息，如果选 2，则不保存退出。我们选择 1，保存并停止 Wanpipe（由于我们还没有启动 Wanpipe，所以停止动作无效，该选项只是为了保证在重新配置时能正确的停止 Wanpipe）。

接下来可以看到它会提示删除旧的配置文件，并把新的配置文件拷贝到相关的配置目录中。其中有的配置文件要放到到 FreeSWITCH 的 conf 目录中，这也是为什么我们要先安装基本的 FreeSWITCH 的原因。屏幕输出如下：

```
Removing old configuration files...  
  
Copying new Wanpipe configuration files...  
  
Copying new sangoma_prid configuration files (/etc/wanpipe/smg_pri.conf)...  
  
Copying new freetdm configuration files (/usr/local/freeswitch/conf/freetdm.conf)...  
  
Copying new freetdm configuration files (/usr/local/freeswitch/conf/autoload_configs/freetdm.conf.xml)...
```

接下来会询问是否让 wanrouter 随操作系统一起启动，我们选择 YES，即允许它随操作系统一起启动。

```
Wanrouter start complete...  
cat: /etc/inittab: No such file or directory  
Warning: Failed to determine init boot level, assuming 3
```

```
Wanrouter boot scripts configuration...
```

```
Removing existing wanrouter boot scripts...OK
Would you like wanrouter to start on system boot?
1. YES
2. NO
[1-2]: 1
```

注意，上面有上警告信息（Warning）是因为 Ubuntu 在 12.04 中不使用 inittab 文件了，可以忽略该警告。

在驱动安装和配置完成后，就可以下列命令启动和停止 Wanpipe 了，命令如下：

# wanrouter start	# 启动
# wanrouter stop	# 停止
# wanrouter status	# 查看当前状态

例如，下面是 Wanpipe 启动时的输出信息。可以看出，它启动了一个 wanpipe 类型的接口设备，配置项是 wanpipe1，接口的名称是 w1g1。

```
# wanrouter start

Starting WAN Router...
Loading WAN drivers: wanpipe done.
Starting up device: wanpipe1
Configuring interfaces: w1g1
done.
```

启动之后，可以通过如下命令查看当前板卡的状态信息：

```
# wanrouter status

Devices currently active:
wanpipe1

Wanpipe Config:

Device name | Protocol Map | Adapter | IRQ | Slot/IO | If's | CLK | Baud rate |
wanpipe1    | N/A          | A101/1D/2/2D/4/4D/8/8D/16/16D| 19 | 2      | 1      | N/A | 0
```

```
Wanrouter Status:
```

Device name	Protocol	Station	Status
wanpipe1	AFT TE1	N/A	Disconnected

注意，上面最后一行，提示线路的状态是 Disconnected，是因为我们还没有接线。A101 板卡提供一个 RJ48 接口¹⁰，在跟其他设备对接之前，可以自己使用自环电缆「环」一下。RJ48 只使用 8 根线中的 4 根。其中，管脚 1、2 分别代表发，4、5 分别代表收，将对应的收和发接起来就做成了一根自环电缆。如图 H.1 所示：

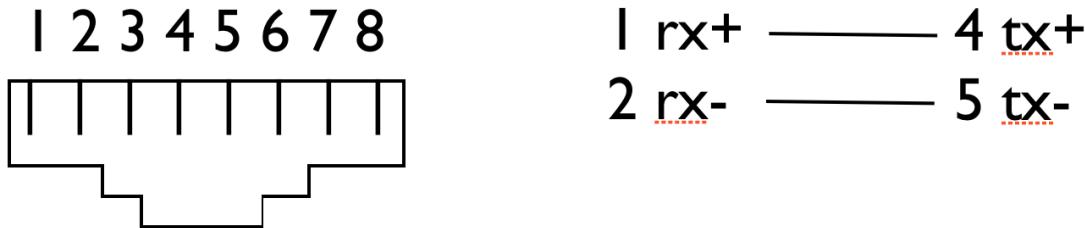


图 10.1: E1 自环电缆的接法

插入自环电缆后，可以使用 wanrouter status 命令查看板卡的状态变成 Connected 了¹¹。

Device name	Protocol	Station	Status
wanpipe1	AFT TE1	N/A	Connected

Wanpipe 的配置文件存放在 /etc/wanpipe 目录下，可以手工修改。我们刚才配置的板卡的配置文件存放在 /etc/wanpipe/wanpipe1.conf，读者在实践过程中可以自行查看一下。

到这时在，Wanpipe 的安装和配置就基本完成了。接下来，我们看一下信令模块的安装。

10.1.4 安装和配置 ISDN 协议库

FreeTDM 运行多种信令协议，如 PRI、BRI、SS7、MFC-R2 及 Analog 协议等。在这里，我们的 E1 线路需要使用 ISDN 中的 PRI 协议。FreeTDM 有三个模块支持以下三种 PRI 协议实现：

- **ftmod_libpri**: 该模块使用开源的 libpri 库支持 PRI 的 BRI 协议。
- **ftdm_isdn**: 该模块致力于在 FreeTDM 中对 PRI 提供原生的支持，目前仍在开发中，不知道是否稳定。

¹⁰与以太口 RJ45 在物理上是一样的，只是线序不同。

¹¹注意，只有在使用主时钟（Master Clock）的情况下自环才有效，否则，由于没有时钟，系统无法知道是否正常收发。

- **ftdm_sangoma_isdn**: 该模块使用 Sangoma 私有的商业电信级的 Trillium 协议栈¹²，支持 PRI 及 BRI 信令，能最好的配合 Sangoma 板卡。

既然我们这里使用的是 Sangoma 板卡，因此我们就选择第三种模块。按以下步骤即可完成下载和安装：

```
# wget ftp://ftp.sangoma.com/linux/libsgn_isdn/libsgn_isdn-current.i686.tgz
# tar zxvf libsgn_isdn-current.i686.tgz
# cd libsgn_isdn-7.27.2.i686/
# ./install.sh
```

其中，7.27.2是本书写作时的版本号。`install.sh`安装脚本会完成所有的安装步骤。

以上命令是在 32 位系统下执行的。如果使用的是 64 位的系统，则可以参考如下对应的命令：

```
# wget ftp://ftp.sangoma.com/linux/libsgn_isdn/libsgn_isdn-current.x86_64.tgz
# tar zxvf libsgn_isdn-current.x86_64.tgz
# cd libsgn_isdn-7.27.2.x86_64
# ./install.sh
```

至此，ISDN 信令协议的依赖库也安装好了。这些底层的驱动都安装好以后，就可以继续下一步安装**mod_freetdm**模块了。

10.1.5 安装和配置 mod_freetdm

只有正确安装了 Wanpipe 及 ISDN 信令协议后才能在编译**mod_freetdm**时正确的编译相关的模块。因此，需要确保上述步骤成功才能执行这一步。

跟安装其他模块类似，在 FreeSWITCH 源码目录直接执行以下命令就可以完成**mod_freetdm**的安装：

```
# make mod_freetdm
# make mod_freetdm-install
```

安装过程中安装脚本可以自动找到相应的驱动程序，并配置安装相关的 FreeTDM 子模块。

安装完成之后，可以看到 FreeSWITCH 的 conf 目录中找到两个 FreeTDM 相关的文件。其中，`conf/freetdm.conf`为 FreeTDM 的配置文件，内容如下：

¹²参见 <http://wiki.sangoma.com/Freeswitch-FreeTDM-Sangoma-ISDN-Library>。

```
[span wanpipe wp1]
trunk_type => e1
group=1
b-channel => 1:1-15
b-channel => 1:17-31
d-channel => 1:16
```

该配置文件的格式是类似于 Windows 中「.ini」文件的配置格式。其中第 1 行配置了一个 Span，一个 Span 相当于板卡上的一个 E1 端口，它的类型是 **wanpipe**，名称是 **wp1**；**trunk_type** 即中继的类型，这里是 **e1**；**group** 表示一个组号；该 E1 板卡上共有 32 个时隙（又称信道，即 Channel），其中 0 时隙是传时钟同步的，在这里没有列出，16 时隙为 D 信道（传信令），其他的 30 个时隙为 B 信道（传话音）。

另一个配置文件是 **mod_freetdm** 的模块配置文件，它位于 **conf/autoload_configs/freetdm.conf.xml**。详细内容我们就不详细列举了，其中一部分如下：

```
<sangoma_pri_spans>

<span name="wp1" cfgprofile="my_pri_nt_1">
  <param name="dialplan" value="XML"/>
  <param name="context" value="default"/>
</span>

</sangoma_pri_spans>
```

其中，它定义了一个 Span，名称 **wp1** 是与 **freetdm.conf** 中的 Span 的名字相对应的。并且，里面还定义了 Dialplan 和 Context，用于对该 E1 中继的来话进行路由。

当然，读者可以发现这些配置的值都是我们前面的配置向导生成的。后面也可以手工更改并重新加载。

确保 **wanrouter** 已经启动，然后就可以在 FreeSWITCH 控制台上使用如下命令加载 **mod_freetdm** 了：

```
freeswitch> load mod_freetdm
+OK Reloading XML
+OK
```

至此，安装的工作已经完成，并且看起来 **mod_freetdm** 也加载成功了。

加载 **mod_freetdm** 之后，可以在 FreeSWITCH 控制台上使用模块提供的 **ftdm** 命令查看 E1 板卡及各 Span 和 Channel 的状态，比如，使用以下命令列出当前系统配置的所有 Span：

```
freeswitch> ftdm list

+OK
span: 1 (wp1)
type: Sangoma (ISDN)
physical_status: alarmed
signaling_status: DOWN
chan_count: 31
dialplan: XML
context: default
dial_regex:
fail_dial_regex:
hold_music:
analog_options: none
```

从上面的输出可以看到，我们只有一个 Span (wp1)，它的物理状态 (`physical_status`) 是 `alarmed` (警告)，信令状态 (`signaling_status`) 是 `DOWN`，意味着未连接。

如果这时候插入自环电缆，则可以看到 FreeSWITCH 控制台上会输出好多 DEBUG 级别的日志信息，标志状态的变化。重复上述命令会看到物理状态变成 `ok`，但是信令状态仍然是 `DOWN` 的（因为自环只能模拟一个「对端设备」，可以模拟物理连接的情况，无法与「对端」建立信令连接）。插入自环电缆后查看状态的输出如下：

```
freeswitch@ubuntu32> ftdm list

+OK
span: 1 (wp1)
type: Sangoma (ISDN)
physical_status: ok
signaling_status: DOWN
chan_count: 31
dialplan: XML
context: default
dial_regex:
fail_dial_regex:
hold_music:
analog_options: none
```

另外，也可以使用「`ftdm dump`」命令显示实际的 Span 中指定的时际的状态信息，如：

```
freeswitch> ftdm dump 1 1
+OK
```

```
span_id: 1
chan_id: 1
physical_span_id: 1
physical_chan_id: 1
physical_status: ok
physical_status_red: 0
physical_status_yellow: 0
physical_status_rai: 0
physical_status_blue: 0
physical_status_ais: 0
physical_status_general: 0
signaling_status: DOWN
type: B
state: DOWN
last_state: DOWN
txgain: 0.00
rxgain: 0.00
cid_date:
cid_name:
cid_num:
ani:
anIII:
dnis:
rdnis:
cause: NONE
session: (none)
```

「**ftdm dump**」命令可以将当前 Span 和 Channel 的状态显示输出，上面的例子中后面两个参数分别是 Span 号和 Channel 号。当然，如果省略 Channel 号，则会**dump**整个 Span 里面所有的 Channel。

10.1.6 配置电话路由

虽然我们还没有与其他系统对接，但这里我们先把来、去话的路由准备好，等对端的设备准备好了以后就可以直接打电话进行测试了。

注意，这里说的「来、去」的概念都是相对的一路通话，对于一端来说是来话，对于另一端来说是去话。我们这里涉及到的设备比较多，因而一次要想清楚来话和去话是针对哪个设备来说的。

在 FS1 上，对于从 E1 上的来话而言，由于我们上面已经指定了将来话路由到 XML Dialplan 的**default Context** 中，因此暂时不需要任何设置。

对于去话，我们在**default Context** 中配置以下 Dialplan，让本地用户在拨打以 0 开头的电话号码时将电话从 E1 端口上送出去。Dialplan 的内容如下：

```

<extension name="E1">
    <condition field="destination_number" expression="^0(.*)$">
        <action application="bridge" data="freetdm/1/a/$1"/>
    </condition>
</extension>

```

其中，正则表达式「`0(.*)`」表示匹配所有以被叫号码 0 开头的通话。如果匹配，则将实际的电话号码存储到「`$1`」变量中。然后，执行后续的 Action。这里的 Action 只有一个 `bridge`。其中，`bridge` 是一个 App，对此我们大家都很熟悉了。后面的 `freetdm` 为 `mod_freetdm` 提供的呼叫字符串，各参数是以「/」分隔的，其中，`freetdm` 为 Endpoint 的类型，后面的参数含义如下：

- 「`1`」代表第一个 Span¹³。我们在这里只有一个 Span。
- 「`a`」代表选线规则。

其中，小写的「`a`」表示从小到大选择一个空闲的时隙（Channel）并将电话送出，大写的「`A`」表示从大到小选择一个空闲时隙。注意如果双方同时选择一个时隙的情况下，将会发生「撞车」的现象。为了最大限度避免这种碰撞，相互对接的双方可以从不同的方向优先选择时隙。当然，如果真发生了「撞车」，也有相关机制保证有一方会自动放弃并选择下一个可用的时隙。

- 「`$1`」表示实际送出的被叫电话号码。

其中，从 Dialplan 的正则表达式中可以看到，「`$1`」的值中是不包括「`0`」的，即如果 FS1 上的用户拨叫了 01000，实际送到 E1 线路上的被叫号码将是 1000。这是一种常见的做法，通俗的说法就是在电话路由的时候「吃」掉前面的第一个「`0`」。

至此，FS1 上的配置就结束了。我们再来看其他设备的配置。

10.2 配置 E1 网关设备

在这里，我们以鼎信通达的 E1 网关 MTG1000 为例。MTG1000 是一款 E1 转 SIP 的网关设备（它只是完成 E1/SIP 话路的信令转换和路由，因此，我们还需要另外一个 SIP 服务器 FS2 才能完成通话测试）。

在进行下一步的测试之前，先使用 E1 交叉线将 MTG1000 和 Sangoma A101 连接起来。

该设备默认有两个以太网口。其中，网口 0 的默认地址是 `192.168.1.111`，用于与其他 SIP 设备对接；网口 1 的默认地址是 `11.11.11.11`，一般用于接笔记本电脑对设备进行管理。

¹³ 在主机上装有多个 Span 的情况下，也可以将任何的时隙在逻辑上组成一个分组，因此，这里也可以填入一个组名。一个分组可以跨越多个 Span，在这里就不多介绍了，有兴趣的读者可以参见 <http://wiki.freeswitch.org/wiki/FreeTDM>。

10.2.1 添加 PRI 中继

我们打开浏览器，进入管理界面，选择添加一个 PRI 中继，如图 10.2 所示。

添加PRI中继	
中继编号	0
中继名称	FS-ISDN
接口标识符	0
D通道	启用
E1/T1端口号	0
协议类型	ISDN
接口属性	用户侧
振铃信号	ALERTING

确定 **重置** **取消**

图 10.2: 添加 PRI 中继

其中，中继编号使用默认的「0」；中继名称我们可以取名为「FS-ISDN」；接口标志符使用「0」；D 通道选择启用；E1/T1 端口号使用默认的「0」；协议类型选择 ISDN（与我们 FS1 中的 EuroISDN 对应）；接口属性选择设备侧（与 FS1 中的相对，对方应该选择 NET，即网络侧）；其他选项可以使用默认值。

点击确定后保存设置。然后可以转到状态页面查看 E1 中继端口的状态。如图 10.3 所示，如果看到端口状态变成绿色的（状态是 Activated）则表示链路正常了。

这时候也可以在 FS1 中使用「`ftdm list`」和「`ftdm dump`」等检测链路的状态。

10.2.2 添加 SIP 中继

与 PRI 中继相对应，在 MTG1000 中，它与我们的 SIP 服务器 FS2 相连的 SIP 链路称为 SIP 中继。

如下图所示。在管理界面上选择添加 SIP 中继，中继编码使用默认的 0；中继名称输入「FS-SIP」；对端地址输入我们 FS2 的 IP 地址，在这里是 192.168.1.119；对端端口我们使用 5080，以防止对该网关设备进行鉴权，简化配置。

其他的配置选项都使用默认的配置即可。点击确定以保存配置。

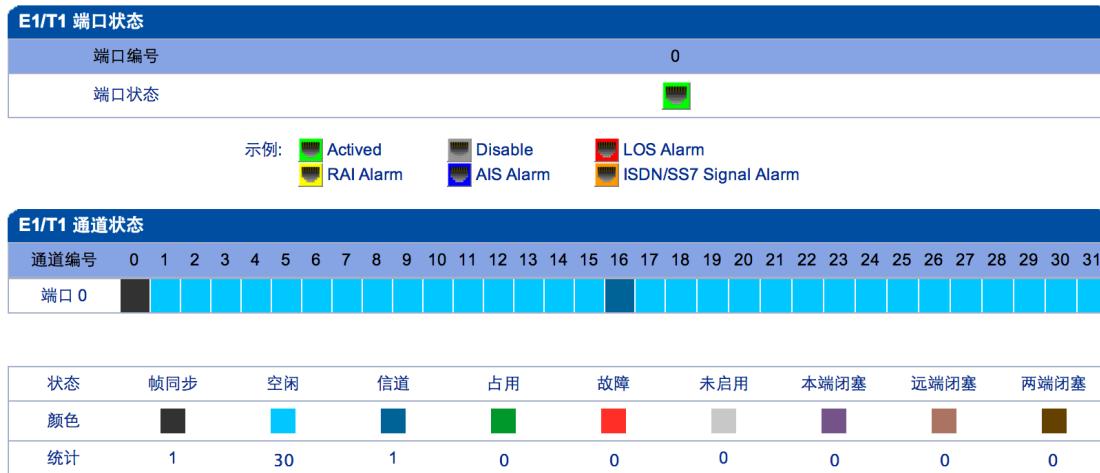


图 10.3: E1 中继状态

添加SIP中继

中继编号	0
中继名称	FS-SIP
对端地址	192.168.1.119
对端端口	5060
代理地址	

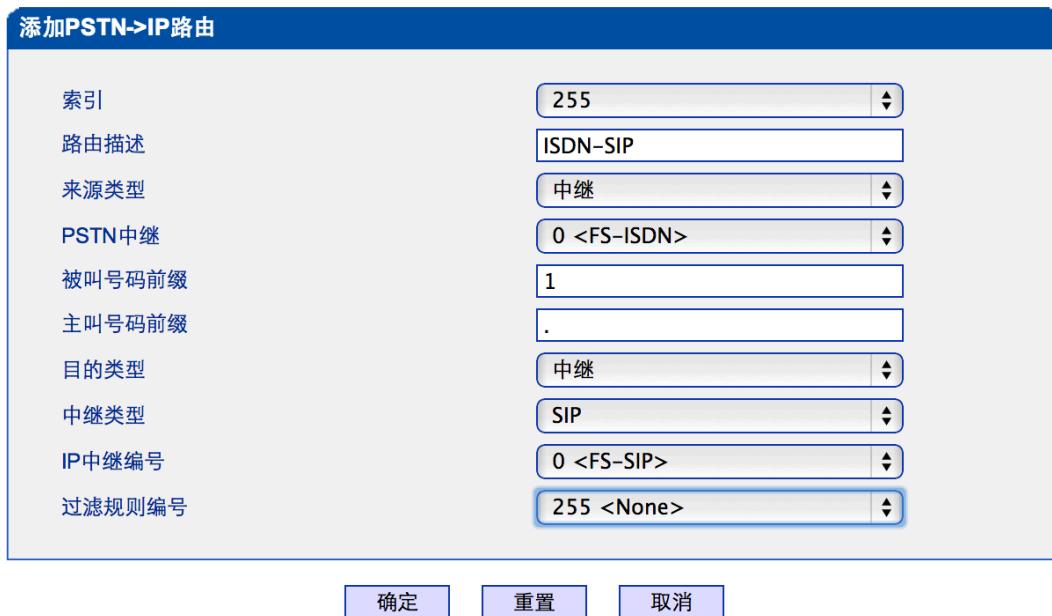
图 10.4: 添加 SIP 中继

10.2.3 添加路由

有了 E1 中继和 SIP 中继，该网关的作用就是把这些中继串起来。而这种「串」就是靠 MTG1000 的路由功能实现的。

首先，我们添加一条路由，将从 E1 端口进来的通话路由到 SIP 中继上去。在 MTG1000 的管理界面上选择添加 PSTN-IP 路由，如图 10.5 所示。

其中，索引号使用默认的 255；路由描述中我们填上「ISDN-IP」；来源类型选择「中继」；PSTN 中继选择我们刚刚创建的「0 <FS-PSTN>」；被叫号码前缀填入 1，即从该 E1 线路进来的，而且被叫号码为 1 的电话将走这一条路由；主叫号码前缀由于我们不关心，这里可以填入「.」以接受任意主叫号码；目的类型选择中继；中继类型选择 SIP；IP 中继编号即我们刚刚创建的 SIP 中继「0 <FS-SIP>」；其他参数使用默认值。



注意： '在'被叫号码前缀' 或者 '主叫号码前缀' 域中，可用'.'表示任意符号。

图 10.5: 添加 PSTN 到 SIP 路由

上面的配置的目的就是，从 E1 来的呼叫，满足一定的主、被叫号码规则，则路由到一条 SIP 中继上去。

用类似的方法可以配置一条 SIP 到 E1 的中继，让从 SIP 侧过来的通话能送到 E1 上去。如图 10.6 所示，参考的含义与上面的类似，在这里就不多介绍了。

添加IP->PSTN路由

索引	255
路由描述	SIP-ISDN
来源类型	中继
中继类型	SIP
中继编号	0 <FS-SIP>
被叫号码前缀	1
主叫号码前缀	.
目的类型	中继
PSTN中继	0 <FS-ISDN>
过滤规则编号	255 <None>

确定 **重置** **取消**

注意： 在'被叫号码前缀'或者'主叫号码前缀'域中，可用'.'表示任意符号。

图 10.6: 添加 SIP 到 PSTN 路由

10.3 配置 FS2

在 FS2 侧，对于来话而言，由于 MTG1000 将电话路由到了 5080 端口，因此来话将进入 public Dialplan，而 public Dialplan 已经有默认的 Dialplan 可以将电话路由于本地用户了。

对于去话而言，则需要把 MTG1000 也看成一个网关，在 FS2 上配置一个网关并将电话路由到该网关即可。或者，直接在 default Dialplan 中使用如下配置直接将去话送到 MTG1000 所在的 IP 地址上：

```
<extension name="MTG1000">
  <condition field="destination_number" expression="^0(.*)$">
    <action application="bridge" data="sofia/external/$1@192.168.1.111"/>
  </condition>
</extension>
```

其中，192.168.1.111 为 MTG1000 的 IP。

10.3.1 拨打测试

我们上面配置了 FS1、MTG1000 网关，及 FS2。如果 FS1 上的用户 1000 想呼叫 FS2 上的用户 1001，则拨打 0 加上对方的号码，如 01001；电话在 FS1 上进行路由时被叫号码最前面的 0 会被 Dialplan 「吃」掉，并将剩下的 1001 送到 E1 中继网关 MTG1000 上；MTG1000 在收到来自 E1 中继的来话后，查找本地的路由表，并将该电话送到配置好的 SIP 中继上，实际上就是送到我们的 FS2 的 5080 端口上；FS2 在 5080 端口上接收到来话后，即查找本 public Dialplan，将最终向本地用户 1000 振铃。呼叫流程如图 10.7 所示。

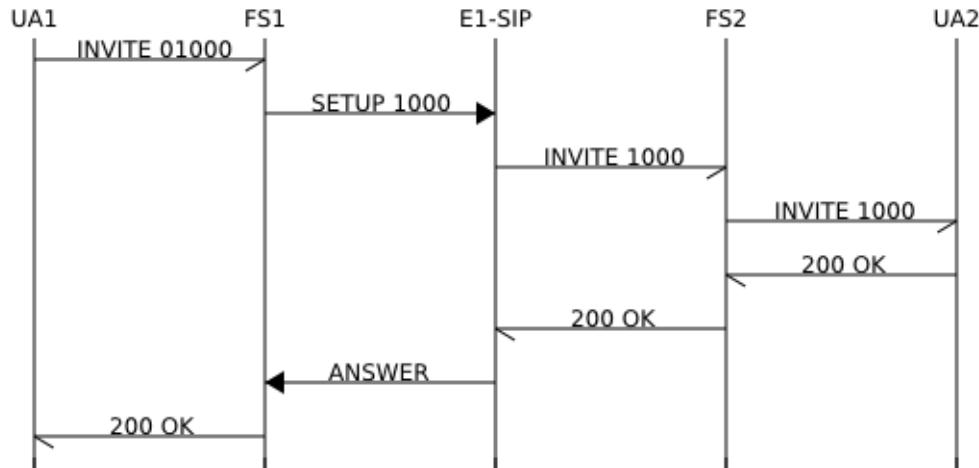


图 10.7: 呼叫流程

反过来从 FS2 到 FS1 的通话流程跟上面差不多，读者可以自行推演。

通过过程中，读者也可以使用「`fwdump dump 1 5`」命令查看当前 Span 的占用状态。如：

```

freeswitch> fwdump dump 1 5

span_id: 1
chan_id: 5
physical_span_id: 1
physical_chan_id: 5
physical_status: ok
physical_status_red: 0
physical_status_yellow: 0
physical_status_rais: 0
physical_status_blue: 0
physical_status_ais: 0
physical_status_general: 0
signaling_status: UP
type: B
state: UP
last_state: PROCEED

```

```

txgain: 0.00
rxgain: 0.00
cid_date:
cid_name:
cid_num: 1000
ani: 1000
anIII:
dnis: 1001
rdnis:
cause: NONE
session: c854e884-1c97-11e3-befe-0f1ace9d0d7b

-- States --          -- Function --          -- Location --          -- Time Offset --
DOWN   => RING    [sngisdn_process_con_ind]  [ftmod_sangoma_isdn_stack_hdl] 0ms
RING   => PROCEED [channel_on_routing]        [mod_freetdm.c:456]           5ms
PROCEED => UP      [channel_receive_message_b] [mod_freetdm.c:991]          10ms

Time since last state change: 91886ms

```

10.3.2 指定中继拨打

在实际应用中，有时需要经常测试某条线路的好坏。而如果使用出局路由自动选线的方式选择到的某一条中继或时隙可能是未知的。所以我们需要一种方法能指定选择某条中继或时隙来进行测试，这种测试方法就称为指定中继拨打。

我们先来看一下如下的freetdm呼叫字符串，我们在这里不使用「a」或「A」的选线方式，而是直接指定从某个时隙呼出，如：

```
freeswitch> originate freetdm/1/5/1000 &echo
+OK c854e884-1c97-11e3-befe-0f1ace9d0d7b
```

上述命令从第一个 Span 的第 5 个时隙呼出，并在本端执行echo，对端接听后可以听到自己的回声。

如果我们不挂机再重复执行一次，可以看到由于该时隙已被占用，并返回拥塞消息NORMAL_CIRCUIT_CONGESTION：

```
freeswitch> originate freetdm/1/5/1000 &echo
-ERR NORMAL_CIRCUIT_CONGESTION
```

了解了上述方法以后，我们可以设置以下 Dialplan 进行指定中继拨打。

```
<extension name="Dial specific trunk">
<condition field="destination_number" expression="^*\*22([0-9])([0-9])(.*)$">
    <action application="bridge" data="freetdm/$1/$2/$3"/>
</condition>
</extension>
```

其中，正则表达式为「`^**22([0-9])([0-9])(.*)$`」。如果我们呼叫「*22151000」，则它与正则表达式匹配。其中，「*22」相当于我们指定中继拨打的功能码；接下来的 1 匹配第一个括号里的「[0-9]」，被记到变量「\$1」中；同理，「5」被记到变量「\$2」中；后面的「1000」被记到变量「\$3」中。因此，最后呼叫字符串「`freetdm/$1/$2/$3`」将变成「`freetdm/1/5/1000`」，即从第一个 Span 的第 5 个时隙呼出。

如果我们想从第 6 个时隙呼出，则可以呼叫「*22161000」。通过这种方法，可以呼叫做任意的时隙¹⁴，这便是指定中继拨打的含义。

10.4 对接其他厂家的 E1 网关

作为一个对比，我们也来看一下与迅时 E1 数字网关的对接。这里，我们使用的设备型号是 MX100-TG。其中 FS1 与 FS2 的配置保持不变。在 MX100-TG 上，我们也需要与 14.4.2 节描述的类似的步骤——设置 ISDN 和 SIP 侧的参数、添加双向的路由等。不过，后面我们可以看到，迅时网关的配置步骤比较简洁一些。

10.4.1 ISDN 设置

MX100-TG 的配置界面与 MX8 类似。打开浏览器，进入管理界面，选择 ISDN，便看到如图 14-14 所示的配置界面。

其中，ISDN 链路的名称我们可以取名为「FS-ISDN」；收号方式选择默认的整体收号；D 通道选择默认的 16 时隙；接口协议选择默认的用户侧（与 FS1 中的相对，对方应该选择 NET，即网络侧）；信令标准选择默认的 CCITT；选线方式可以选择默认的顺序，也可以选择其他的选线方式；其他选项可以使用默认值。

点击提交后保存设置，即可生效。如果一切正常，则 E1 链路就会变成正常的连通状态，在图 10.8 的界面上也能观察到。

¹⁴实际上，这里的 Dialplan 不支持呼叫 2 位数的时隙，读者可以自己练习一下，看能否通过所学的知识自己实现。

基本配置	ISDN	高级配置	呼叫状态	日志管理	系统工具	版本信息																																																																																																																																																																																													
欢迎管理员登录																																																																																																																																																																																																			
登录时间: 1970-01-01 08:17:20																																																																																																																																																																																																			
ISDN 1 注销																																																																																																																																																																																																			
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">名称:</td> <td colspan="6">FS-ISDN</td> </tr> <tr> <td style="padding: 2px;">启用:</td> <td colspan="6"><input checked="" type="checkbox"/></td> </tr> <tr> <td colspan="7" style="padding: 2px;">应用选项</td> </tr> <tr> <td style="padding: 2px;">收号方式:</td> <td colspan="3"><input type="radio"/> 逐位收号</td> <td colspan="3"><input checked="" type="radio"/> 整体收号</td> </tr> <tr> <td style="padding: 2px;">D 通道:</td> <td colspan="3"><input checked="" type="radio"/> 16 时隙</td> <td colspan="3"><input type="radio"/> 24 时隙</td> </tr> <tr> <td style="padding: 2px;">接口协议:</td> <td colspan="3"><input checked="" type="radio"/> 用户侧</td> <td colspan="3"><input type="radio"/> 网络侧</td> </tr> <tr> <td colspan="7" style="padding: 2px;">该设置必须与对端设备相对应, 一边是网络侧, 另一边是用户侧</td> </tr> <tr> <td style="padding: 2px;">信令标准:</td> <td colspan="6">CCITT ▾ 一般 T1 卡采用 NI-2, E1 卡采用 CCITT</td> </tr> <tr> <td style="padding: 2px;">连线方式:</td> <td colspan="6">顺序 ▾</td> </tr> <tr> <td style="padding: 2px;">D 通道启用维护消息:</td> <td colspan="6"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 2px;">点对点连接:</td> <td colspan="6"><input type="checkbox"/> 无需被叫号和通道号</td> </tr> <tr> <td style="padding: 2px;">主叫号码类型:</td> <td colspan="3"><input checked="" type="radio"/> 规范</td> <td colspan="3"><input type="radio"/> 非规范</td> </tr> <tr> <td colspan="7" style="padding: 2px;">设置CPN 主叫 Category子域</td> </tr> <tr> <td style="padding: 2px;">主叫号码显示权限:</td> <td colspan="6"><input type="checkbox"/> 设置CPN 主叫 Presentation子域</td> </tr> <tr> <td style="padding: 2px;">被叫号码类型:</td> <td colspan="3"><input checked="" type="radio"/> 规范</td> <td colspan="3"><input type="radio"/> 非规范</td> </tr> <tr> <td colspan="7" style="padding: 2px;">设置CDPN 被叫 Category子域</td> </tr> <tr> <td style="padding: 2px;">被叫忙线处理:</td> <td colspan="3"><input checked="" type="radio"/> 语音提示</td> <td colspan="3"><input type="radio"/> 挂断</td> </tr> <tr> <td style="padding: 2px;">允许对端更换线路:</td> <td colspan="6"><input type="checkbox"/> 在CID中选择Exclusive</td> </tr> <tr> <td colspan="7" style="padding: 2px;">二次拨号</td> </tr> <tr> <td style="padding: 2px;">启用:</td> <td colspan="6"><input type="checkbox"/> 播放二次拨号提示并检测DTMF号码</td> </tr> <tr> <td style="padding: 2px;">提示方式:</td> <td colspan="3"><input type="radio"/> 语音提示</td> <td colspan="3"><input checked="" type="radio"/> 拨号音提示</td> </tr> <tr> <td style="padding: 2px;">主叫号:</td> <td colspan="3"><input type="radio"/> 原主叫号</td> <td colspan="3"><input checked="" type="radio"/> 原被叫号</td> </tr> <tr> <td style="padding: 2px;">被叫号:</td> <td colspan="3"><input type="radio"/> 原被叫号+二次输入号</td> <td colspan="3"><input checked="" type="radio"/> 二次输入号</td> </tr> <tr> <td colspan="7" style="padding: 2px;">时隙管理</td> </tr> <tr> <td colspan="7" style="padding: 2px;">配置该T1/E1线路上允许用于从IP往ISDN方向呼叫的通道(从ISDN方向呼入的呼叫不受此配置影响)</td> </tr> <tr> <td colspan="7" style="text-align: center; padding: 2px;"> <input type="checkbox"/> 0 <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5 <input type="checkbox"/> 6 <input type="checkbox"/> 7 <input type="checkbox"/> 8 <input type="checkbox"/> 9 <input type="checkbox"/> 10 <input type="checkbox"/> 11 <input type="checkbox"/> 12 <input type="checkbox"/> 13 <input type="checkbox"/> 14 <input type="checkbox"/> 15 <input type="checkbox"/> 16 <input type="checkbox"/> 17 <input type="checkbox"/> 18 <input type="checkbox"/> 19 <input type="checkbox"/> 20 <input type="checkbox"/> 21 <input type="checkbox"/> 22 <input type="checkbox"/> 23 <input type="checkbox"/> 24 <input type="checkbox"/> 25 <input type="checkbox"/> 26 <input type="checkbox"/> 27 <input type="checkbox"/> 28 <input type="checkbox"/> 29 <input type="checkbox"/> 30 <input type="checkbox"/> 31 </td> </tr> <tr> <td colspan="7" style="text-align: center; padding: 2px;">提 交</td> </tr> </table>							名称:	FS-ISDN						启用:	<input checked="" type="checkbox"/>						应用选项							收号方式:	<input type="radio"/> 逐位收号			<input checked="" type="radio"/> 整体收号			D 通道:	<input checked="" type="radio"/> 16 时隙			<input type="radio"/> 24 时隙			接口协议:	<input checked="" type="radio"/> 用户侧			<input type="radio"/> 网络侧			该设置必须与对端设备相对应, 一边是网络侧, 另一边是用户侧							信令标准:	CCITT ▾ 一般 T1 卡采用 NI-2, E1 卡采用 CCITT						连线方式:	顺序 ▾						D 通道启用维护消息:	<input type="checkbox"/>						点对点连接:	<input type="checkbox"/> 无需被叫号和通道号						主叫号码类型:	<input checked="" type="radio"/> 规范			<input type="radio"/> 非规范			设置CPN 主叫 Category子域							主叫号码显示权限:	<input type="checkbox"/> 设置CPN 主叫 Presentation子域						被叫号码类型:	<input checked="" type="radio"/> 规范			<input type="radio"/> 非规范			设置CDPN 被叫 Category子域							被叫忙线处理:	<input checked="" type="radio"/> 语音提示			<input type="radio"/> 挂断			允许对端更换线路:	<input type="checkbox"/> 在CID中选择Exclusive						二次拨号							启用:	<input type="checkbox"/> 播放二次拨号提示并检测DTMF号码						提示方式:	<input type="radio"/> 语音提示			<input checked="" type="radio"/> 拨号音提示			主叫号:	<input type="radio"/> 原主叫号			<input checked="" type="radio"/> 原被叫号			被叫号:	<input type="radio"/> 原被叫号+二次输入号			<input checked="" type="radio"/> 二次输入号			时隙管理							配置该T1/E1线路上允许用于从IP往ISDN方向呼叫的通道(从ISDN方向呼入的呼叫不受此配置影响)							<input type="checkbox"/> 0 <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5 <input type="checkbox"/> 6 <input type="checkbox"/> 7 <input type="checkbox"/> 8 <input type="checkbox"/> 9 <input type="checkbox"/> 10 <input type="checkbox"/> 11 <input type="checkbox"/> 12 <input type="checkbox"/> 13 <input type="checkbox"/> 14 <input type="checkbox"/> 15 <input type="checkbox"/> 16 <input type="checkbox"/> 17 <input type="checkbox"/> 18 <input type="checkbox"/> 19 <input type="checkbox"/> 20 <input type="checkbox"/> 21 <input type="checkbox"/> 22 <input type="checkbox"/> 23 <input type="checkbox"/> 24 <input type="checkbox"/> 25 <input type="checkbox"/> 26 <input type="checkbox"/> 27 <input type="checkbox"/> 28 <input type="checkbox"/> 29 <input type="checkbox"/> 30 <input type="checkbox"/> 31							提 交						
名称:	FS-ISDN																																																																																																																																																																																																		
启用:	<input checked="" type="checkbox"/>																																																																																																																																																																																																		
应用选项																																																																																																																																																																																																			
收号方式:	<input type="radio"/> 逐位收号			<input checked="" type="radio"/> 整体收号																																																																																																																																																																																															
D 通道:	<input checked="" type="radio"/> 16 时隙			<input type="radio"/> 24 时隙																																																																																																																																																																																															
接口协议:	<input checked="" type="radio"/> 用户侧			<input type="radio"/> 网络侧																																																																																																																																																																																															
该设置必须与对端设备相对应, 一边是网络侧, 另一边是用户侧																																																																																																																																																																																																			
信令标准:	CCITT ▾ 一般 T1 卡采用 NI-2, E1 卡采用 CCITT																																																																																																																																																																																																		
连线方式:	顺序 ▾																																																																																																																																																																																																		
D 通道启用维护消息:	<input type="checkbox"/>																																																																																																																																																																																																		
点对点连接:	<input type="checkbox"/> 无需被叫号和通道号																																																																																																																																																																																																		
主叫号码类型:	<input checked="" type="radio"/> 规范			<input type="radio"/> 非规范																																																																																																																																																																																															
设置CPN 主叫 Category子域																																																																																																																																																																																																			
主叫号码显示权限:	<input type="checkbox"/> 设置CPN 主叫 Presentation子域																																																																																																																																																																																																		
被叫号码类型:	<input checked="" type="radio"/> 规范			<input type="radio"/> 非规范																																																																																																																																																																																															
设置CDPN 被叫 Category子域																																																																																																																																																																																																			
被叫忙线处理:	<input checked="" type="radio"/> 语音提示			<input type="radio"/> 挂断																																																																																																																																																																																															
允许对端更换线路:	<input type="checkbox"/> 在CID中选择Exclusive																																																																																																																																																																																																		
二次拨号																																																																																																																																																																																																			
启用:	<input type="checkbox"/> 播放二次拨号提示并检测DTMF号码																																																																																																																																																																																																		
提示方式:	<input type="radio"/> 语音提示			<input checked="" type="radio"/> 拨号音提示																																																																																																																																																																																															
主叫号:	<input type="radio"/> 原主叫号			<input checked="" type="radio"/> 原被叫号																																																																																																																																																																																															
被叫号:	<input type="radio"/> 原被叫号+二次输入号			<input checked="" type="radio"/> 二次输入号																																																																																																																																																																																															
时隙管理																																																																																																																																																																																																			
配置该T1/E1线路上允许用于从IP往ISDN方向呼叫的通道(从ISDN方向呼入的呼叫不受此配置影响)																																																																																																																																																																																																			
<input type="checkbox"/> 0 <input type="checkbox"/> 1 <input type="checkbox"/> 2 <input type="checkbox"/> 3 <input type="checkbox"/> 4 <input type="checkbox"/> 5 <input type="checkbox"/> 6 <input type="checkbox"/> 7 <input type="checkbox"/> 8 <input type="checkbox"/> 9 <input type="checkbox"/> 10 <input type="checkbox"/> 11 <input type="checkbox"/> 12 <input type="checkbox"/> 13 <input type="checkbox"/> 14 <input type="checkbox"/> 15 <input type="checkbox"/> 16 <input type="checkbox"/> 17 <input type="checkbox"/> 18 <input type="checkbox"/> 19 <input type="checkbox"/> 20 <input type="checkbox"/> 21 <input type="checkbox"/> 22 <input type="checkbox"/> 23 <input type="checkbox"/> 24 <input type="checkbox"/> 25 <input type="checkbox"/> 26 <input type="checkbox"/> 27 <input type="checkbox"/> 28 <input type="checkbox"/> 29 <input type="checkbox"/> 30 <input type="checkbox"/> 31																																																																																																																																																																																																			
提 交																																																																																																																																																																																																			

图 10.8: 添加 ISDN 中继

10.4.2 路由设置

迅时系列的网关，可以直接将 SIP 通话路由到 IP，因而不需要设置明确的 SIP 中继。为了能让 FS1 与 FS2 之间互打，我们只需简单的设置以下两条路由：

ISDN	x	ROUTE	IP	192.168.1.119:5080
IP	x	ROUTE	ISDN	1

其中，第一条与在模拟网关 MX8 中的设置类似，即所有从 ISDN 侧来的呼叫，不管被叫号码是什么，全部（通过 SIP）路由到 FS2 的 IP 地址上去（192.168.1.119: 5080）；而第二条也很容易理解，即所有从 IP 侧来的 SIP 呼叫，全部路由到第 1 条 ISDN 链路（E1）上去（如果有多个 E1 的话，也可以使用类似模拟线的配置方式如「1,2,3,4」选择不同的 E1 链路）。

第十一章 使用 GSM 网关连接 PSTN

在某些特殊的场合，使用无线的 GSM 网关连接 PSTN 也是很有用且很有效的方法。

GSM 网关本身跟 FXO 口的网关功能差不多，只不过是把网关上本来用于插电话线的 FXO 口换成了 SIM 卡。这样，该网关的一端是 SIP，另一端则通过 SIM 卡连接到 PSTN。对于 PSTN 侧来讲，这种类型的网关就相当于一个手机，而对于我们的 FreeSWITCH 来讲，它就相当于一个普通的 SIP 网关。

笔者在测试时使用的是鼎信通达的 DWG2000-1G GSM 网关。将网关插上 SIM 卡，插上网线，加电后，网关应该能自己通过 DHCP 获取 IP 地址。笔者在测试时，网关的 IP 地址是 192.168.7.12。用浏览器访问以上地址，并输入正确的密码登录后，主界面如图 11.1 所示：

从上图的「Mobile 信息」部分可以看出，SIM 卡已经正常的注册到中国联通的网络上了，也就是说，该网关的「手机」部分已经能正常使用了，下面，我们再来配置一个 SIP 部分，以后能正常的呼入呼出。

11.1 通过网关呼出

跟我们讲得其它的例子一样，我们还是采用最简单的配置方式，并不需要把网关「注册」到 FreeSWITCH 上去。因此，我们到菜单「系统配置」→「SIP 配置」中，将是否注册改为「否」并保存。然后，我们就可以在 FreeSWITCH 中使用如下的命令快速测试一下能否正常外呼了：

```
freeswitch> bgapi originate sofia/external/133xxxxxxxx@192.168.7.12 &echo
```

上面的命令我们已经很熟悉了，它直接在 FreeSWITCH 中发起外呼，其中 133xxxxxxxx 是被叫号码，而 192.168.7.12 则是 GSM 网关的 IP 地址。如果一切正常，在被叫接听后应该能执行大家已经熟悉的 echo 程序，并在电话里听到自己的回音。



图 11.1: 网关初始页

当然，为了能上本地分机都能通过以上网关外呼，我们只需要构造类似如下的 Dialplan（默认情况下放到 default.xml 中），通过在被叫号码前加拨「0」就通使用该网关：

```
<extension name="GSM Outbound">
    <condition field="destination_number" expression="^0(.*)$">
        <action application="bridge" data="sofia/external/$1@192.168.1.12"/>
    </condition>
</extension>
```

类似这样的 Dialplan 在书中我们已经讲过很多了，在此就不多解释了。

11.2 通过网关呼入

如果有人呼叫我们 SIM 卡对应的手机号，那么，我们也要在网关上做一个路由送来话路由到我们的 FreeSWITCH 上。为了能达到这个目的，首先我们要如图 11.2 所示点击菜单「IP 中继配置」→「IP 中继」并添加一个 SIP 中继：

在此，192.168.7.6 是 FreeSWITCH 的 IP 地址，而为了避免对来话进行鉴权，我们使用了 FreeSWITCH 的 5080 端口。



图 11.2: 添加 SIP 中继

接下来，我们再点击菜单「路由配置」→「Tel→IP 路由」。该网关已经有了一条编号为 31 的路由，我们需要修改一下。如图 11.3 所示，其中，「主叫号码前缀」和「被叫号码前缀」都是「any」代表任意号码。只要号码匹配该规则，便会路由到我们刚刚创建的「IP 中继」上：

在默认情况下，网关会对 GSM 来话放二次拨号音，以便提示输入一个分机号并进行下一步的路由。在此，我们 FreeSWITCH 的功能非常强大，自然，这些活就让 FreeSWITCH 干就行了，因此，在此，我们并不需要网关直接播放二次拨号音，而是让网关直接将来话送给 FreeSWITCH 即可。为了达到这个目的，我们配置点击菜单「系统配置」→「端口配置」，并修改其中的配置，在「呼往 VoIP 热线」一栏填入一个号码即可，它将做为一个被叫号码（DID 号码）发送给 FreeSWITCH。一般来说，你可以输入 SIP 卡对应的手机号，不过，这里，为了简单通用起见，我们直接输入了 gsm，如图 11.4 所示：

用另外一个 PSTN 电话拨打 SIM 卡对应的手机号，就能在 FreeSWITCH 中看到熟悉的「绿色的行了」：

```
[INFO] mod_dialplan_xml.c:558 Processing 133xxxxxxxx <->gsm in context public
```

当然，由于我们还没有在 FreeSWITCH 中配置相关的路由，该呼叫是失败的。接下来，就可以根据自己的需要将来话路由到任意的目的地了，如下面的 Dialplan（放到 public.xml 中）将对来话进行应答并播放一个声音文件：



图 11.3: 配置 PSTN 来话路由



图 11.4: 设置 DID 以避免播放二次拨号音

```
<extension name="GSM Inbound">
<condition field="destination_number" expression="^gsm$">
    <action application="answer" data="" />
    <action application="playback" data="/tmp/welcome.wav" />
</condition>
</extension>
```

也可以将来话桥接到另外一个本地分机，如：

```
<extension name="GSM Inbound">
<condition field="destination_number" expression="^gsm$">
    <action application="bridge" data="user/1000" />
</condition>
</extension>
```

关于 GSM 网关，我们就讲到这里。有兴趣的读者可以对照一下网关的参考手册试验一下其它的配置参数，也可以结合本书讲的知识配置更灵活方便的 Dialplan。当然，最后值得一提的是，除了 GSM 网关外，鼎信通达也有 CDMA 型号的网关，以便连接到 CDMA 制式的移动网络，读者可以根据自己的情况选用。

第十二章 对接 Asterisk

Asterisk 是老牌的开源 VoIP 软件了。事实上，FreeSWITCH 是主要作者 Anthony Minessale 在早些年也是 Asterisk 的开发者，后来，由于对 Asterisk 的架构设计和性能问题有意思，开发一个新的 Asterisk 分支又得不到社区主要人特的支持，因此便从头开发了 FreeSWITCH¹。

由于 Asterisk 比 FreeSWITCH 资格老，因此，在国内有大量的用户群，很多情况下也需要与 FreeSWITCH 进行对接。因此，这里我们也来看一下 Asterisk 与 FreeSWITCH 对接的例子。

在本例中，我们仅使用中继方式（在 Asterisk 中称为 peer，即对等的连接方式）与 Asterisk 进行对接，其它的对接方式如相互向对方注册等留给读者练习。我们假定读者已经熟悉 Asterisk。先在 Asterisk 上添加一个分机 6000，并用一个 SIP 客户端注册上去，为下面的测试做准备。另外，下面的测试中 FreeSWITCH 的 IP 地址为 192.168.7.5，而 Asterisk 的 IP 地址为 192.168.7.99。

12.1 从 FreeSWITCH 呼叫 Asterisk

首先，我们已经很熟悉如何在 FreeSWITCH 将电话送出了，通过使用下列命令可以呼叫 Asterisk 上的 6000：

```
freeswitch> originate sofia/external/6000 &echo
```

或者，使用如下的 Dialplan，当 FreeSWITCH 上的用户拨打 6000 时，也可以送到 Asterisk 上：

```
<action application="bridge" data="sofia/external/6000@192.168.7.99"/>
```

¹在此，作者只是陈述一个事实，并无意将两者对立起来。其实，Asterisk 的新版本在近几年性能也有了很大的提升。并且，在市场上有很多成熟的应用，很多用户也将会长期使用 Asterisk，而这也是笔者写这一节的原因之一。FreeSWITCH 的主要作者 Anthony Minessale 在 2010 年接受 VOIPToday 采访并被问及 FreeSWITCH 与其它「竞争者」的关系时也说过：这并不一个「认为 FreeSWITCH 必须‘打败’Asterisk 才能‘胜利’的」公认的所谓的零和游戏，在很多情况下人们都会把 FreeSWITCH 用作他们已有的 Asterisk、CallWeaver、YATE 安装部署的一部分……」

呼叫到达 Asterisk 上后，就可以在 Asterisk 的控制台上看到类似如下的日志：

```
NOTICE[3684][C-0000000d]: chan_sip.c:25381 handle_request_invite:
Call from '' (192.168.7.5:5080) to extension '6000' rejected
because extension not found in context 'public'.
```

出现上述提示的原因是我们没有在 Asterisk 上配置入局路由。接下来，我们只需打开 Asterisk 的配置文件 `extension.conf`，在 public Context 中增加如下的设置就可以了。

```
exten => 6000,1,Dial(SIP/6000)
```

其中，「6000」匹配被叫号码；「1」是一个优先级标志；「Dial」是一个 Application，类似 FreeSWITCH 中的 `bridge`；最后「SIP/6000」是一个呼叫字符串。

在 Asterisk 的控制台上使用「`dialplan reload`」命令使之生效，然后，再一次在 FreeSWITCH 中发起呼叫，Asterisk 上的分机 6000 就可以振铃了。

12.2 从 Asterisk 上呼叫 FreeSWITCH

我们也可以很容易的在 Asterisk 中使用如下 Dialplan 将电话送到 FreeSWITCH 上。

```
exten => _01X.,1,Dial(SIP/${EXTEN:1}@192.168.7.5:5080)
```

其中，「_01X.」是 Asterisk 中的模式匹配规则，它匹配以「01」开头的被叫号码；「1」为优先级标志；而后面的 `Dial` 相当于 FreeSWITCH 中的 `bridge` App；再后面是一个以 `SIP` 开头的呼叫字符串；「\${EXTEN}」即模式匹配中的被叫号码，而「\${EXTEN:1}」即去掉最前面的一位（即「吃掉 0」），然后把剩下的号码作为被叫号码并将呼叫送到 FreeSWITCH 的地址 `192.168.7.5:5080` 上去。

因此，使用上述 Dialplan，在 Asterisk 上使用分机 6000 呼叫号码 01000 的话，Asterisk 就会「吃」掉被叫号码中最前面的「0」，并将呼叫送达 FreeSWITCH，然后 FreeSWITCH 上的分机 1000 就会振铃。

上述呼叫在 FreeSWITCH 中能够正确路由是因为 FreeSWITCH 中默认的 `public` Dialplan 包含了到用户 1000 的路由。当然，如果是呼叫其它号码的话，也可以根据实际情况在 FreeSWITCH 中配置入局路由。

12.3 其他

在上面的例子中，我们仅介绍了使用中继对接的最简单的情形。电话是可以呼通了，但至于用于生产环境中的安全性考虑，使用账号或 IP 地址对来话进行验证等就留给读者自行练习了。当然，读者也可以参考以下 Wiki 页面上的对接说明<https://freeswitch.org/confluence/display/FREESWITCH/Converting+Asterisk+to+FreeSWITCH>。

另外，从上面的例子可以看出，在 Asterisk 与 FreeSWITCH 中的用户及路由的配置概念都是很类似的。如果用户熟悉 Asterisk，则 FreeSWITCH 也应该很快能上手。Wiki 页面<https://freeswitch.org/confluence/display/FREESWITCH/Rosetta+Stone> 对两者中的相似的概念进行了对比，对于刚刚从 Asterisk 「转」到 FreeSWITCH 的读者来说，读一读或有帮助。

最后，关于 Asterisk 和 FreeSWITCH 的关系，我们来看一下 Anthony Minessale 在 2009 年接受媒体采访时的一段对话：

(媒体)：如果有人想把他或她的 Asterisk 系统换成 FreeSWITCH，是否需要编写很多的代码以及学习完全不同的配置文件语法？

Anthony：并不像你想象的那么难。理解 FreeSWITCH 最关键的地方有一个它与 Asterisk 之间的词形变化表。事实上，它们之间有很多相似的地方，如他们都使用可加载模块，都使用「application」执行相应的功能等。不同的是，在 FreeSWITCH 中，需要学习如何从一个尽量将一切复杂性简化的角度看问题。很多从 Asterisk 转移过来的 FreeSWITCH 新手往往由于想得太多而把问题搞得很难复杂。实际上我们无比强大的社区成员已经写了一个名为「Asterisk 罗塞塔石碑²」的 Wiki 页面³，该页面非常好地解释了从 Asterisk 转到 FreeSWITCH 的各种概念和要素。另外，我们也有一个模块允许你使用你熟悉的 Asterisk 的语法写 Dialplan 进行电话路由。

888VS: If someone wanted to replace his or her Asterisk system with FreeSWITCH, would it require lots of coding and relearning completely different config file syntax?

Anthony: It's not as hard as you might think. The biggest key to understanding FreeSWITCH is a paradigm shift from how Asterisk does things. Both have a lot in common in the fact that they employ modules and use「applications」and what not but the key to learning FreeSWITCH is to look at it from an angle of trying to remove as much of the complexity as possible. Many new Asterisk users「over think」what they want to do and end up trying way too hard. Our awesome community has actually created an「Asterisk Rosetta Stone」[http://wiki.freeswitch.org/wiki/Rosetta_stone] that does a really good job of translating some Asterisk concepts to FreeSWITCH. Also, we have a module that lets you implement your dialplan in a familiar extensions.conf format to get you started.

²Rosetta Stone，是一块刻有古埃及法老托勒密五世 (Ptolemy V) 詔书的石碑。由於這块石碑同时刻有同一段内容的三种不同语言版本，使得近代的考古学家得以有机会对照各语言版本的内容，解读出已经失传千余年的埃及象形文字的意义与结构。参见：<http://zh.wikipedia.org/wiki/罗塞塔石碑>。——笔者（译者）注

³参见<https://freeswitch.org/confluence/display/FREESWITCH/Rosetta+Stone>。

第十三章 使用 H.323 协议对接

H.323 是比较古老的 VoIP 协议，然而，现在还是有很多设备支持该协议。作为一款多协议的 VoIP 系统，FreeSWITCH 也支持 H.323。下面，我们就来简单讨论一下如何在 FreeSWITCH 中使用 H.323。

FreeSWITCH 中有两个 H.323 的实现——`mod_opal`¹和`mod_h323`²。前者基于 H323plus 库³，后者基于 Opal⁴库，两个库又都依赖于 ptlib⁵。

13.1 mod_h323

我们先来看`mod_h323`。下面，我们将讨论一下如何进行安装、配置和拨打测试。

13.1.1 安装

在安装`mod_h323`之前，首先要安装它依赖的库。这里需要注意的问题是，`mod_h323`依赖的两个库`ptlib`和`h323plus`需要特定的版本配合在一起才能正常工作。该模块对应的 Wiki 页面上有几个特定的组合列表。这里，笔者使用的是 h323plus 官方网站上最新的组合。

笔者是在 Ubuntu Linux 12.04 上进行测试的，在编译上述两个库之前，要确保系统上有 `flex` 和 `bison` 两个软件。可以用如下命令安装它们：

```
# apt-get install flex bison
```

然后，使用如下命令下载 `ptlib` 和 `h323plus`：

¹参考http://wiki.freeswitch.org/wiki/Mod_opal。

²参考http://wiki.freeswitch.org/wiki/Mod_h323。

³参考<http://www.h323plus.org/>。

⁴参考<http://www.opalvoip.org/>。

⁵`ptlib`是一个底层的支持库，提供一些抽象和跨平台的支持。真正的历史是这样的：所有这些都起源于一个开源的项目 OpenH323，后来，分离成了两个项目 Opal 和 H323plus，底层的一些抽象及跨平台支持也几乎在同时成了独立的库，叫 PWlib，后改名为 `ptlib`。

```
wget http://www.h323plus.org/source/download/ptlib-2.10.9.tar.bz2
wget http://www.h323plus.org/source/download/h323plus-v1_25_0.tar.gz
```

解压并配置安装 ptlib:

```
# tar xvjf ptlib-2.10.9.tar.bz2
# cd ptlib-2.10.9
# ./configure && make && make install
```

解压并配置安装 H323plus:

```
# tar xvzf h323plus-v1_25_0.tar.gz
# cd h323plus
# ./configure && make
```

在make过程中，笔者遇到以下错误：

```
/home/app/book/h323/h323plus/include/openh323builddopts.h:37:34:
fatal error: ptlib/../../revision.h: No such file or directory
```

为了解决该问题，可以修改对应的文件（`h323plus/include/openh323builddopts.h`），将第37行的「`#include <ptlib/../../revision.h>`」一行注释掉或删除。

然后，继续配置安装：

```
# make && make install
```

上述两个库安装完毕后，到 FreeSWITCH 的源代码目录下安装mod_h323。由于我们上面的步骤中，两个库的安装位置都在`/usr/local`目录下，因此，我们需要改变`mod_h323`的 Makefile (`src/mod/endpoint/mod_h323/Makefile`)，将里面所在的「`/usr`」路径都替换为「`/usr/local`」。修改完毕后，就可以在 FreeSWITCH 源代码目录中使用如下命令安装了：

```
# make mod_h323-install
```

或者，进入mod_h323源代码目录上，使用如下命令安装：

```
# cd src/mod/endpoint/mod_h323  
# make install
```

13.1.2 配置和加载模块

安装完毕后，如果一切正常，就可以在FreeSWITCH中加载该模块了：

```
freeswitch> load mod_h323
```

如果在加载时遇到类似下面这样的错误，可以在Linux命令行上执行「ldconfig」，然后重启FreeSWITCH。

```
**libh323_linux_x86_64_.so.1.25.0: cannot open shared object file:  
No such file or directory**
```

如果在加载模块时看到如下的错误，则说明FreeSWITCH默认没有安装相关的配置文件。

```
[ERR] mod_h323.cpp:483 open of h323.conf failed
```

可以在mod_h323源代码目录中找到h323.conf.xml，然后复制到conf/autoload_configs/目录中，如：

```
# cd src/mod/endpoint/mod_h323  
# cp h323.conf.xml /usr/local/freeswitch/conf/autoload_configs/
```

然后再重新加载该模块就可以了：

```
freeswitch> load mod_h323
```

13.1.3 呼叫测试

在正确加载完mod_h323以后，就可以进行拨打测试了。目前，该模块仅支持点对点的模式互通，并不支持 Gatekeeper。因此，可以在大部分的 H323 客户端上直接呼叫 FreeSWITCH 的 IP 地址。呼叫到达 FreeSWITCH 后，将按照h323.conf.xml中的配置进行路由。当然，有些客户端允许在呼叫的 IP 地址前面加一个被叫号码，如，可以尝试呼叫「1000@192.168.7.5」或「1000@192.168.7.5:1720」等。

如果在 FreeSWITCH 中使用mod_h323呼出，可以使用如下 Dialplan：

```
<action application="bridge" data="h323/6000@192.168.7.9"/>
```

其中，「h323」是该模块实现的一个呼叫字符串，后面的地址是欲呼叫的对端的 IP 地址。

13.2 mod_opal

mod_opal的安装步骤与mod_h323类似，它也是需要一对相互匹配的opal库与ptlib库。在测试时，笔者是该地址<http://sourceforge.net/projects/opalvoip/files/v3.12%20Eridani/Stable%206/> 下载的ptlib-2.12.6.tar.bz2 以及opal-3.12.6.tar.bz2。

ptlib的安装过程与在mod_h323中的类似，在此就不多说了。只是注意，如果同时在一台主机上安装两个版本的ptlib是可能会有冲突的，如何在同一台主机上安装并运行不同版本的库文件超出了本书的范围，如果对这些不熟悉的话，请分别在两台干净的机器上分别测试mod_h323与mod_opal。

接下来，使用以下命令安装 Opal：

```
# tar xvjf opal-3.12.6.tar.bz2
# cd opal-3.12.6
# ./configure && make && make install
```

在 FreeSWITCH 源代码目录下使用如下命令安装mod_opal：

```
# cd src/mod/endpoint/mod_opal
# make install
```

笔者在编译时遇到如下错误：

```
/home/app/work/freeswitch/src/mod/endpoints/mod_opal/mod_opal.cpp:1034:37:  
error: ‘class OpalConnection’ has no member named ‘SwitchT38’
```

错误的原因应该是`mod_opal`跟这里`opal`的版本不兼容。不过，T38 是与传真相关的，这里，我们不测试传真，所以，可以在`mod_opal.h`文件中找到如下的行：

```
#define HAVE_T38 (OPAL_CHECK_VERSION(3,11,2) && OPAL_T38_CAPABILITY)
```

修改为：

```
#define HAVE_T38 0
```

即暂时取消与 T38 相关的宏定义。然后就可以继续编译安装了。

安装完成后，在 FreeSWITCH 中加载：

```
freeswitch> load mod_opal
```

然后，如果有电话呼入的情况下，就可以看到 FreeSWITCH 中的日志了。一旦呼叫到达了 Dialplan，就是我们的天下了。通过我们前面学过的知识，如何配置 Dialplan 对来话进行路由对我们来说就是小菜一碟了。因此，在此就不多介绍了。

另外，如果使用`mod_opal`呼出的话，它提供了`opal`呼叫字符串，因而可以使用如下 Dialplan 进行呼出：

```
<action application="bridge" data="opal/h323:1000@192.168.7.9"/>
```

其中，「192.168.7.9」是指对端的 IP 地址。

13.3 其他

上面，我们重点讲了两个 H.323 模块的安装，因为该过程的复杂性往往是读者学习和实验这两个模块门槛。不过，笔者是仅仅做了几个实验。如果读者需要将这两个模块用于生产环境的话，还需要多测试一下，找一找这几个依赖库的最佳组合。

由于使用 H.323 的人比较少，因而大家对于这两个模块开发的积极性也不高，因而它们的功能也不是很完善。如，它们并不支持视频呼叫，也不支持使用 Gatekeeper 注册方式。不过，如果在需要 Gatekeeper 的环境的话，也可以考虑与 GnuGK⁶等专门的 H.323 Gatekeeper 配合部署，在此，我们就不多讲了。

另外，笔者已重新新写了一个 H.323 模块，以支持视频通话。不过，该模块还没有合并到 FreeSWITCH 代码库里，暂时就不写到书里了。

⁶是一款开源的 H.323 Gatekeeper，请参考<http://www.gnugk.org/>。

第 III 部分 闭关修炼

在跟其它系统对接时，总会出现一些奇奇怪怪莫名其妙的问题。但只要我们勤于练习，勤于思考研究，扎实打好基础，「一切反动派都是纸老虎」，无论什么问题，在我们手底下就都不是问题。

在本部分，我们就来讨论一下互联互通中一些相关的知识。

第十四章 DTMF

DTMF 用于在通话中传输按键信息，多用于一些自动 IVR 等需要按键选择的场景中。在 PSTN 中，一般不太关注 DTMF，因为 PSTN 对 DTMF 不作处理，只需要简单的透传即可。但在 FreeSWITCH 中就不同了。有时候，电话是通了，可是却取不到按键信息。所以，我们有必要深入地了解一下 DTMF。

DTMF (Double Tone Multiple Frequency, 双音多频) 是一种在通话中的按键传输方式，特别是在 IVR 类的应用中，一般的电话菜单都是通过按键控制的。在传统的 TDM 电话网络中，如果通话中的用户按下了话机上的一个键，话机就会产生一个 DTMF 的音频信号送到远端的交换机上，交换机会将该信号送到对端的用户那里，如果对端也是一个用户，则他就能听到一些「嘀嘀」的按键音；如果对端是一个 IVR，它就能检测这种信号，并还原成用户按键的数字，进而知道用户选择了哪个菜单。

在 SIP 通话中，传输按键信息的方式就多了一些。虽然有些信息可以完全不用双音多频的信号来表示，但沿用传统的叫法，我们仍然把按键信息称为 DTMF。

14.1 带内 DTMF

在基于 SIP/RTP 的通话中，如果跟传统的 TDM 电话对接，直接把 TDM 电话设备产生的双音频信号按与音频编码同样的编码放到 RTP 数据中传输，则称为带内 DTMF (Inband)。FreeSWITCH 默认不使用带内 DTMF。因为，检测带内 DTMF 必须要实时检测 RTP 包中的内容，会多耗费很多 CPU。不过，传统的 TDM 设备只支持这种方式，因而，如果跟传统的 PSTN 网络的交换机进行 SIP 对接时，它们往往默认会使用这种方式。由于 PSTN 网络上的交换机一般不会使用到 DTMF，它们只需要对 DTMF 信号进行简单的透传，而不需要进行深入地检测，因而对它们来说无所谓是否耗费资源。

如果在 FreeSWITCH 中使用带内 DTMF，则需要在对应的 Profile 中（如「internal」或「external」）中设置如下参数：

```
<param name="dtmf-type" value="inband"/>
```

或者，在通话时，在 Dialplan 中通过设置如下的通道变量实现：

```
<action application="set" data="dtmf_type=inband"/>
```

当然，设置了上述变量并不能使 FreeSWITCH 检测到带内的 DTMF，还必须在 Dialplan 中明确的使用「`start_dtmf`」App 打开 DTMF 检测。如，对于来话而言，可以使用如下的 Dialplan：

```
<action application="set" data="dtmf_type=inband"/>
<action application="start_dtmf"/>
<action application="ivr" data="some_ivr"/>
```

对于去话，则可以通过设置一个钩子通道变量¹「`execute_on_answer`」实现，如：

```
<action application="export" data="nolocal:execute_on_answer=start_dtmf"/>
<action application="bridge" data="sofia/gateway/pstn/${destination_number}"/>
```

在上述 Dialplan 中，我们使用「`export`」设置了一个变量「`execute_on_answer`」，该变量是以「`nolocal:`」开头的，因此，它将只在 bleg 上生效。在执行到「`bridge`」时，将产生 bleg。本例中的 bleg 通过一个「`pstn`」网关呼叫被叫号码，当对端接听后，FreeSWITCH 就会执行「`execute_on_answer`」钩子变量所指定的 App，即「`start_dtmf`」，它用于开启带内 DTMF 检测。

当然，如果对「`export`」App 以及其原理比较熟悉的话（参见 6.1.9 节），便知道上述 Dialplan 跟如下的写法是等价的：

```
<action application="bridge"
  data="${execute_on_answer=start_dtmf}sofia/gateway/pstn/${destination_number}"/>
```

14.2 RFC2833

RFC2833²中定义了在 RTP 中传输 DTMF 信息的另一种方式。该方式通过特殊的 RTP Payload 标志，标志出包含 DTMF 信息的 RTP 包，因而，不需要对所有的 RTP 包进行通过深度的频率检测就可以很容易地「认」出包含 DTMF 信息的包，而这种包里的 DTMF 信息是以文本表示的，因而也

¹FreeSWITCH 中有一系列的以`execute_on_`开头的通道变量，这些变量在系统中设置一些「钩子」，当 Channel 执行到一定状态时，便会执行这些钩子指定的 App，如，此处的`execute_on_answer`会在相关的 Channel 被应答时执行指定的 App。

²参见<http://tools.ietf.org/html/rfc6455>。

不需要计算。与 Inband 检测相比，RFC2833 方式大大节省了 CPU。这种方式是现行的 VoIP 通话中比较常用的方式，也是 FreeSWITCH 中默认的方式。

RTP 包中 DTMF 的 Payload Type 值是在 SDP 消息中使用「telephone-event」协商的，如下面的 SDP 消息中，表示 Payload Type 为 101 的 RTP 包为包含 DTMF 消息的包。

```
v=0
o=FreeSWITCH 1371848118 1371848119 IN IP4 192.168.1.9
s=FreeSWITCH
c=IN IP4 192.168.1.9
t=0 0
m=audio 31988 RTP/AVP 8 101
a=rtpmap:8 PCMA/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=silenceSupp:off - - -
a=ptime:20
m=video 19008 RTP/AVP 123
a=rtpmap:123 H264/90000
```

如果在媒体协商时看不到「telephone-event」，则证明它不支持 DTMF2833 模式。

注意：由于 RTP 底层使用了不可靠的 UDP 传输协议，因此 RFC2833 规定包含同一 DTMF 信息的包要按一定的规则重发，以减少可能的丢包带来的影响。而且，DTMF 的 RTP 包跟音频 RTP 包是在一个媒体流（相同的「IP 地址:端口号」）中传输的，因此，包含 DTMF 的 RTP 包中的时间戳以及 Marker 标志还需要跟普通音频的正确配合才能使用。这就带来了该协议的一些复杂性。比较令人遗憾的是，有很多设备的 DTMF2833 实现都有 Bug，而且几年甚至十几年都不改。为了能够跟这些顽固的设备对接，FreeSWITCH 内部也进行了相应的妥协，针对某些设备提供了一些专门的配置参数。当然，如何使用这些具体的参数超出了本书的范围，如果在与这样的设备对接时发现不符合 RFC2833 规定的实现，可以查阅有关资料尝试在 FreeSWITCH 中设置相关参数解决³。

14.3 SIP INFO

DTMF 也可以在 SIP 的 INFO 消息中传送。INFO 消息跟一路通话有关，用于通话中的双方传送一些消息，其中一种消息就是传输 DTMF 消息。

使用 SIP INFO 消息甚至比使用 RFC2833 更有效，但是，有一个缺点是由于 SIP 消息跟用于传送声音的 RTP 流使用不同的 UDP IP 地址和端口，在实际传输中可以经过完全不同的路由设备一个消息，因而用 SIP INFO 传送的 DTMF 可能会与实际听到的声音不同步。

³参见<https://freeswitch.org/confluence/display/FREESWITCH/RTP+Issues>。

如果在 SDP 协商时没有找到「telephone-event」(即不能使用 RFC2833)，则 FreeSWITCH 就会默认启用 INFO 来收发 DTMF。当然，也可以在 Profile 中使用如下参数明确指定使用 INFO 来传送 DTMF：

```
<param name="dtmf-type" value="info"/>
```

或者在 Dialplan 中针对某一路通话决定是否使用 INFO：

```
<action application="set" data="dtmf_type=info"/>
```

关于 DTMF 的知识我们就介绍这么多，请读者根据自己的需要多加练习。

14.4 使用 FreeSWITCH 检测声音文件中的 DTMF 信息

今天，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，在录音时就也一块将 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）而已，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:setInputCallbk() 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 (default.xml) 中加入以下内容：

```
<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。

第十五章 ACL

在生产环境中，只考虑把电话接通还是不够的，还得考虑安全性。比如在本书最开始的几章中，只使用5080端口从public Dialplan 做互通，而发送到5080端口的 INVITE 是不需要鉴权的，这意味着，你任何人均可以向它发送 INVITE 从而按你设定的路由规则打电话。这种方式用在端局上问题可能不大，因为你的public Dialplan 仅将外面的来话路由到本地用户。但在汇接局模式下，你可能将一个来话再转接到其他局去，那你就需要好好考虑一下安全问题了，因为你肯定不希望全世界的人都通过你的汇接局免费的往各端局打电话。

为了防止这个问题，我们在汇接局上关闭5080端口，而让所有来话都送到5060上（internal Profile）。5060上的来话是需要先鉴权才能路由的。在这种汇接局模式中，一般会使用 IP 地址鉴权的方式。而 IP 地址鉴权就会用到 ACL。

ACL (Access Control List) 即访问控制列表，它通过一个列表矩阵来控制哪些用户可以访问哪些资源。在 FreeSWITCH 中，实现了基于 ACL 的鉴权（当然，ACL 不仅用于 SIP 鉴权，还可以用在其他地方）。

其中，internal Profile 默认使用「domains」这个 ACL 进行鉴权。读者可以在internal.xml配置中找到如下的配置：

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

它说明，如果收到呼叫 (INVITE) 请求时，要查看「domains」这个 ACL，看是否允许来源 IP 地址进行呼叫。

ACL 是在conf/autoload_configs/acl.conf.xml中配置的，其中domains的默认配置如下：

```
<list name="domains" default="deny">
    <!-- domain= 是一种特殊情况，它会根据用户目录中的配置生成 ACL -->
    <node type="allow" domain="$$\{domain\}" />
    <!-- 如果你想允许某些 IP 段能通过该 ACL 检查的话，使用 cidr= 的配置方式 -->
    <!-- <node type="allow" cidr="192.168.1.123/32" /> -->
</list>
```

其中，第 1 行说明该列表项的名称是「**domains**」，可以在其他地方引用，默认（**default**）的规则是拒绝请求（**deny**）。

其他几行的的含义我们就不在这儿介绍了。如果我们在 D 上允许来自 A、B、C 的呼叫，我们就可以把 A、B、C 的地址加到上述配置里面，如，可以添加以下配置，以允许（**allow**）来自这些 IP 的呼叫。

```
<node type="allow" cidr="192.168.1.A/32"/>
<node type="allow" cidr="192.168.1.B/32"/>
<node type="allow" cidr="192.168.1.C/32"/>
```

其中，**cidr**为无状态域间路由（Classless Inter Domain Routing）的表示形式，以「/」分开的为 IP 地址和掩码中二进制 1 的位数，32 表示掩码中有 32 个 1，对应的十进制表示即 255.255.255.255，因而它仅表示一台主机；其他常用的掩码位数有 24（255.255.255.0，表示一个 C 类网段）、27（255.255.255.224，表示一个包含 30 台主机 IP 的子网）等。

通过设置上述 ACL，我们就保证了只有授权的用户才能从 D 上进行路由（当然要记得我们关闭了 5080 端口，因此所有呼叫要送到 5060 端口上）。

第十六章 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。

16.1 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 15-7 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`」。

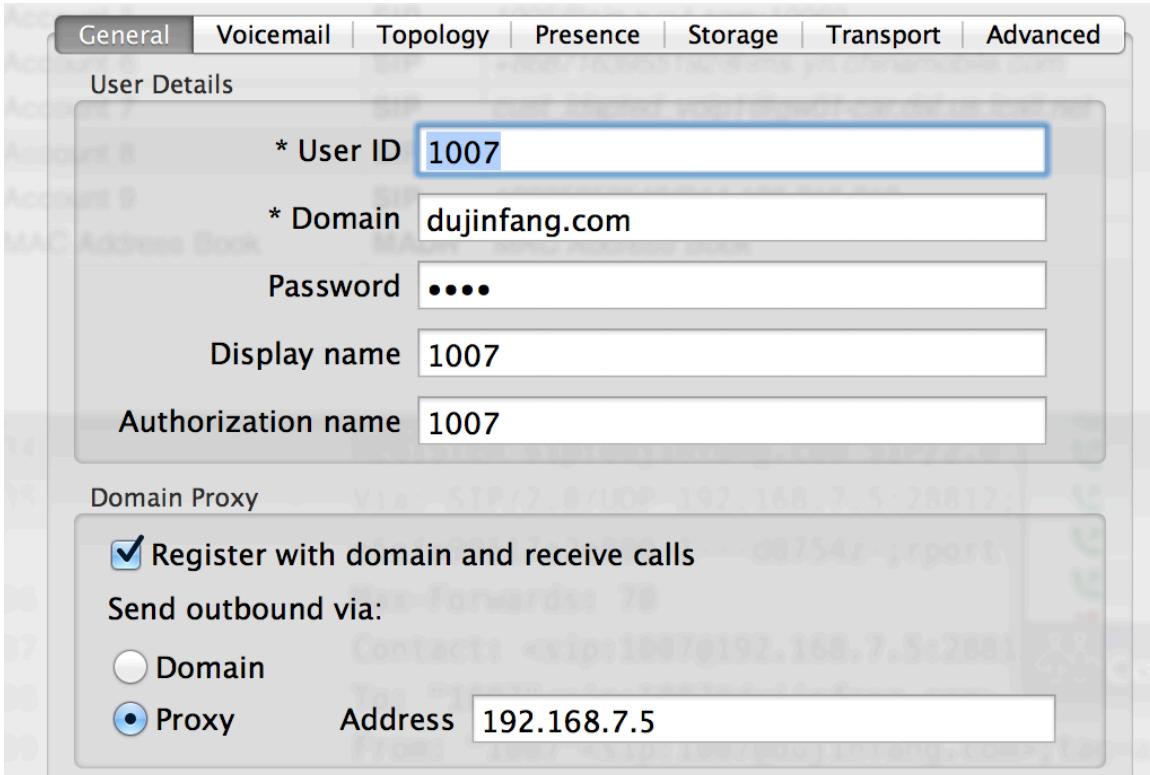


图 16.1: 图 15-7 通过 Proxy 注册

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: ZTBhNGVkMTJ10GQzMZlNjJjN2E2MjQ5NGEzZTZmM2U
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-badfc-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。

16.2 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。

16.3 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。

16.4 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 IV 部分 歪门邪道

对接中可能需要的一些功能与实例。法无定法，无招胜有招。

第十七章 号码连选

在对接中也经常需要号码连选这样的功能。

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。

17.1 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。

17.2 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。

17.3 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。

17.4 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」¹。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：

¹如果某条线路支持多线呼出，我们也可以增加它的权重

```
freeswitch> distributor dist1
32
freeswitch> distributor dist1
35
```

「**distributor**」是该模块提供的一个 API 命令，它可以用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选线了：

```
<extension name="gw">
<condition field="destination_number" expression="^(01[358].*)$">
<action application="bridge"
        data="sofia/gateway/gw${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"/>
```

17.5 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。

第十八章 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 Server¹以及 Wowza Media Server²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，并且普遍认为前景不好。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。

18.1 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
```

¹参见<http://www.red5.org/>。

²参见<http://www.wowza.com/>。

上面的命令显示了有一个 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>」，就可以看到下图所示的界面：

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），笔者的输入如下图：

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。

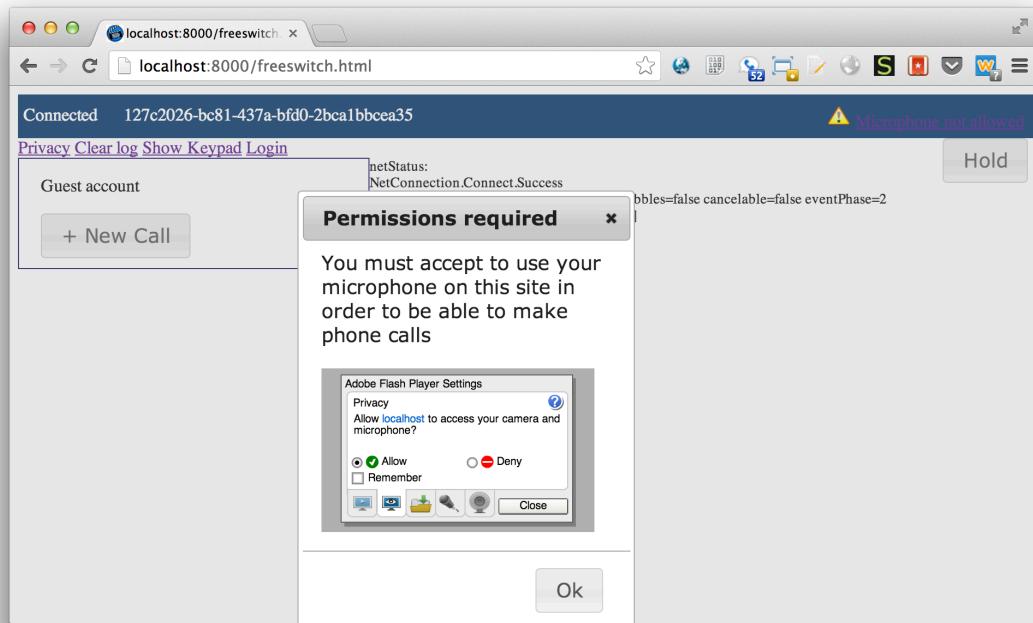


图 18.1: Flash 测试页面初始界面

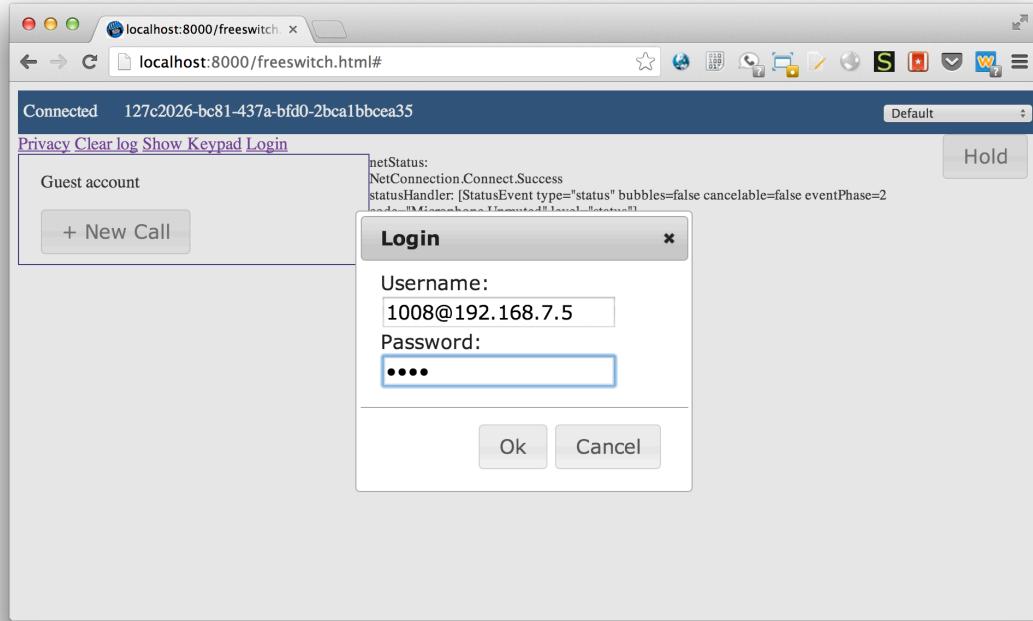


图 18.2: 将 Flash 软电话登录到 FreeSWITCH 上

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「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，可以看到如下图所示的来电提示页面：

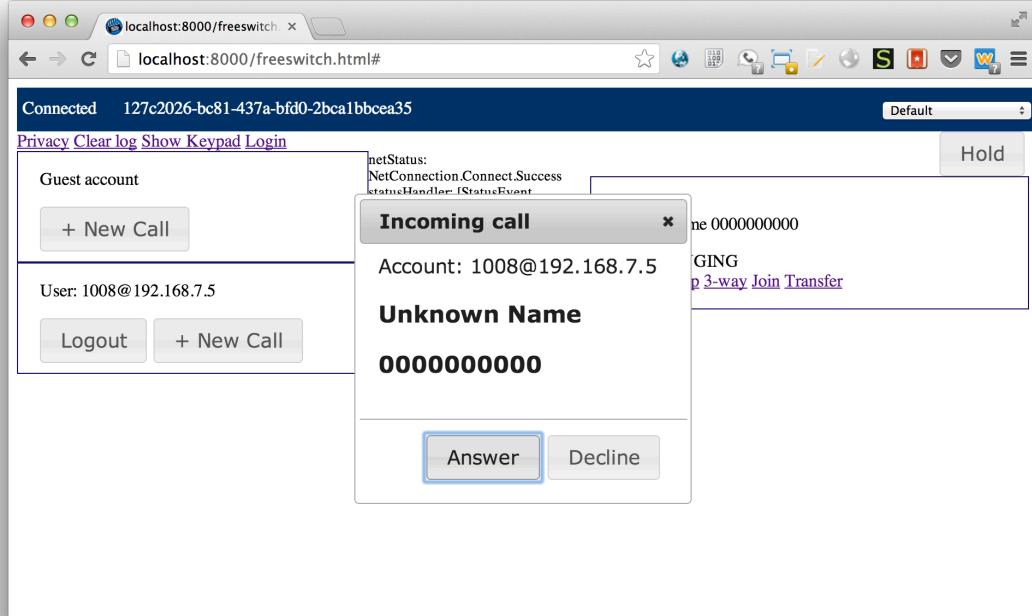


图 18.3: RTMP 来电提示页面

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)$">
        <action application="bridge"
            data="${rtmp_contact(default/$10${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18.4就是笔者做的一个例子（在源代码目录下可以找到flash-video.html）：

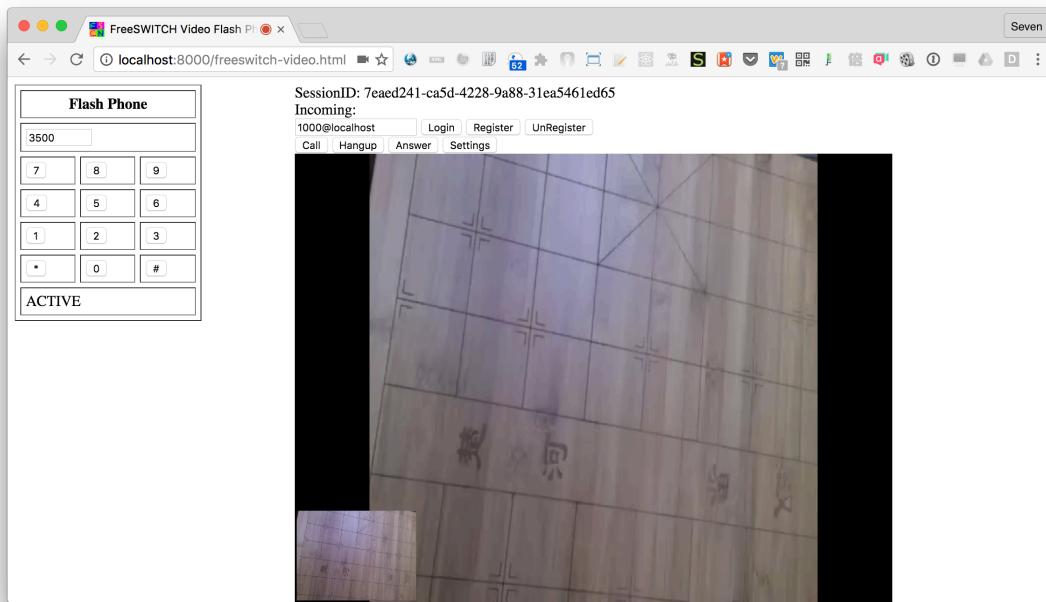


图 18.4: Flash 加入 FreeSWITCH 视频会议

18.2 WebRTC

WebRTC³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。

³<http://www.webrtc.org/>。

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 WebSocket⁴。

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的去请求服务器资源，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 WebSocket⁵，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。

18.2.1 JsSIP

JsSIP⁶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。

⁴<http://zh.wikipedia.org/wiki/WebSocket> <http://tools.ietf.org/html/rfc6455>。

⁵<http://datatracker.ietf.org/doc/draft-ietf-sipcore-sip-websocket/>。

⁶<http://jssip.net/>。

18.2.2 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/>。

18.2.3 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：

```
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 15-11。

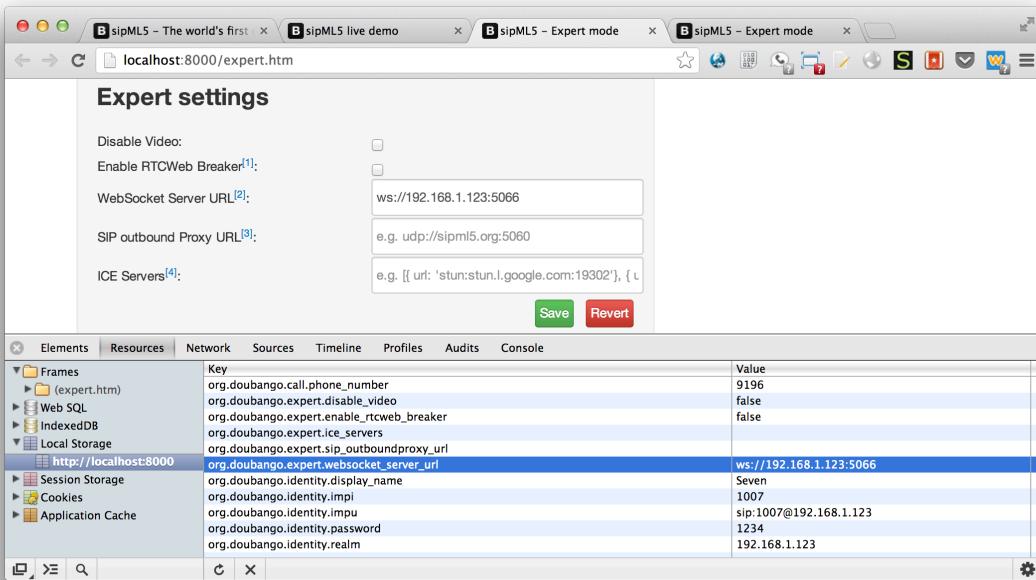


图 18.5: 在 Expert 模式进行设置

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/>。

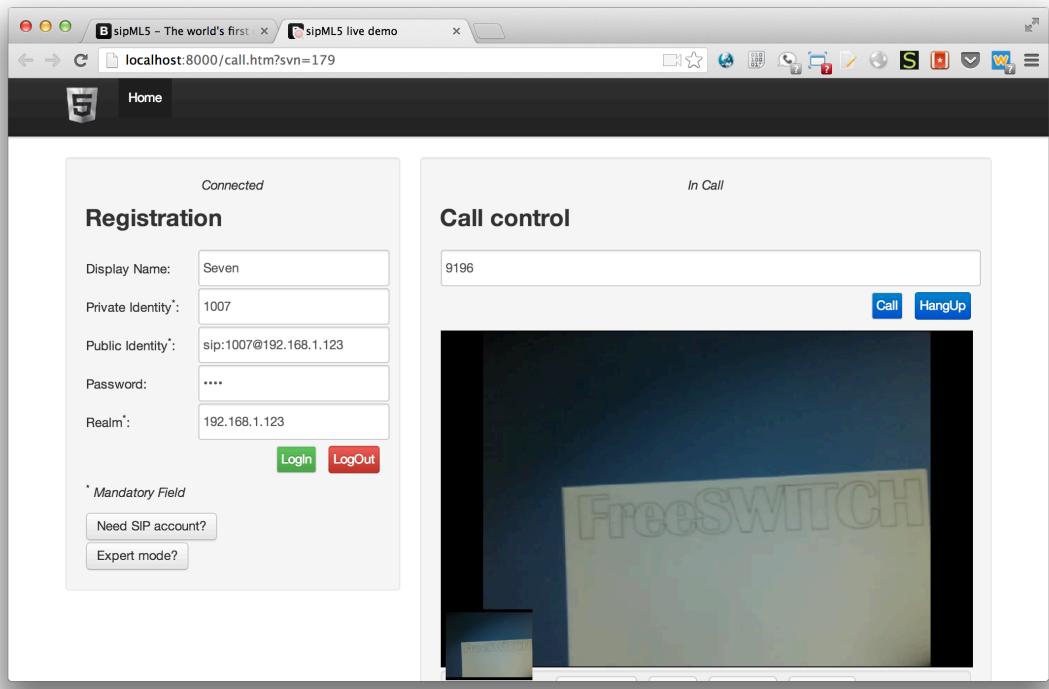


图 18.6: 使用 sipML5 进行 WebRTC 视频呼叫

18.2.4 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。

如果不是有特殊原因非要用 SIP 的话，建议使用 Verto 连接 FreeSWITCH 的视频功能，因为 Verto 支持更多的特性。

第十九章 通过 OpenSIPS 服务与其 它系统对接

有时候，有些特殊的原因，我们可能不希望 FreeSWITCH 直接跟其它系统（如 IMS）对接，而是通过 OpenSIPS 对接。在实际应用中，我们就遇到这么一个例子：IMS 提供若干账号，FreeSWITCH（或 OpenSIPS）注册到 IMS 上，每个账号均只支持一路通话。

在此，我们选择了从 OpenSIPS 向 IMS 注册。

使用 OpenSIPS 2.1 版，我们在`opensips.conf`中增加如下设置：

```
loadmodule "registrar.so"
loadmodule "uac_registrant.so"
```

配置完成后，只需要在相关的数据库表中插入相关记录，OpenSIPS 就可以向其它服务器注册了。我们可以试一把：

```
insert into registrant values (DEFAULT, 'sip:192.168.1.2', NULL,
'sip:1001@192.168.1.2', NULL, '1001', '1234', 'sip:1001@192.168.1.3:9060', NULL, 600);
```

其中，192.168.1.2为对方的地址（如 IMS，此处，我们先用 FreeSWITCH 模拟），192.168.1.3为 OpenSIPS 的地址，9060 为 OpenSIPS 监听的端口号。`registrant`数据表的结构如下（PostgreSQL）：

```
opensips=# \d registrant
          Table "public.registrant"
   Column    |      Type      |           Modifiers
-----+-----+-----+
 id       | integer      | not null default nextval('registrant_id_seq'::regclass)
 registrar | character varying(128) | not null default ''::character varying
```

```

proxy          | character varying(128) | default NULL::character varying
aor           | character varying(128) | not null default ''::character varying
third_party_registrant | character varying(128) | default NULL::character varying
username       | character varying(64)  | default NULL::character varying
password        | character varying(64)  | default NULL::character varying
binding_uri    | character varying(128) | not null default ''::character varying
binding_params | character varying(64)  | default NULL::character varying
expiry         | integer             |
forced_socket  | character varying(64)  | default NULL::character varying

```

过一会，就可以看到 OpenSIPS 自动向192.168.1.2上注册了。

在此，我们的192.168.1.2是使用另一台 FreeSWITCH 服务器模拟的，如果是真正的 IMS，则需要更多的参数，实际上，我们是使用类似如下的方式注册的：

```

insert into registrant values (DEFAULT, 'sip:bj.ims.cn', 'sip:1.2.3.4',
'sip:+8610xxxxxxxx@bj.ims.cn', NULL, '+8610xxxxxxxx@bj.ims.cn', 'password',
'sip:xxxxxxxx@1.2.3.5:9060', NULL, 600);

```

其中，`bj.ims.cn`是 IMS 给我们提供的 Domain，`1.2.3.4`是 IMS 的 IP 地址，而`1.2.3.5:9060`则是 OpenSIPS 的地址和端口。

注册成功后，在 FreeSWITCH 中发起呼叫测试（注意实际命令不要换行）：

```

originate {sip_h_P-Early-Media=supported,sip_invite_domain=bj.ims.cn,sip_route_uri=sip:1.2.3.5:9060,originatation_caller_id_number=+8610xxxxxxxx}sofia/external/00186xxxxxxxx@bj.ctcims.cn &echo

```

其中，我们通过`sip_invite_domain`设置 INVITE 消息中的 Domain，通过`sip_route_uri`指定 OpenSIPS 的地址以便让 FreeSWITCH 把 INVITE 消息先发送到 OpenSIPS 上。

当 INVITE 消息到达 OpenSIPS 后，就可以在 OpenSIPS 上设置相关的 Relay 路由，路由到 IMS 了。关于 OpenSIPS 我们就不多讲了，以下 Relay 配置供参考：

```

if ($si == "IP" && $rU=~"^00") { # temp test relay ims
    append_hf("P-hint: Media Server to IMS\r\n");
    xlog("L_ERR", "$C(gx)relay ims .....$C(xx)\n");
    route(relay_ims);
}

```

...

```
route[relay_ims] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {
        t_on_branch("per_branch_ops");
        #t_on_reply("handle_nat");
        t_on_failure("missed_call");
    }

    if (!t_relay("1.2.3.4")) {
        send_reply("500", "Internal Error");
    };
    exit;
}
```

第 V 部分 附录

附录 A 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。

1.1 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。
- 礼貌回答，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。

1.2 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开始「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。
- <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

¹ 参见 <http://pastebin.freeswitch.org> 及 <http://gist.github.com>。

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。

1.3 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。

附录 B 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，我们这里的目的一不是为了讨论翻译，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。

附录 C FreeSWITCH 背后的故事

Anthony Minessale/文Seven Du/译

本文由 Anthony Minessale 写于 2007 年 5 月。来自www.freeswitch.org¹。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

¹<http://www.freeswitch.org/node/60>

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 I 仍在做相当的一部分 Asterisk 开发，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!)

附录 D 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，我花了几天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，从而可以对客户进行服务。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¹。

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。它的实现现在代码注释中被注明是「肮脏的」²。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」³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

¹<http://www.cluecon.com/anthm.html>

²/* 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 */

³ Avoiding initial deadlock

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减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(Fормats): 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？」

附录 E FreeSWITCH 的历史¹

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 Perl²，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

¹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

²一种编程语言。——译者注。

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³，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，并对开发一些新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。所以后来我与 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。

³ 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。

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

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的日后可能用于 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。我们认为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 Clue⁴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

⁴线索，后面的 Con 指会议（Conference）。——译者注。

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⁵，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。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的状态机。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 Fork⁶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

⁵ 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。

⁶ 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。

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。

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⁷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⁸？

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。Mike⁹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¹⁰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

⁷原文是 Check out，即从 SVN 仓库检出代码。

⁸FreeSWITCH 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。

⁹Michael 的简称。——译者注。

¹⁰该书第一章是 Architecture of FreeSWITCH，即 FreeSWITCH 的架构与设计。——译者注。

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 后来初 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）以及其他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 Tower¹¹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。

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¹²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¹³，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¹⁴并最终变成 FreeSWITCH 核心组件的近 50 万行代码的。

¹¹一个国际性酒店。——译者注。

¹²传说玛雅人的历法中预言 2012 年世界将灭绝。——译者注。

¹³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

¹⁴指作者最初写 FreeSWITCH 代码的时候。——译者注。

附录 F 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 1.2.22 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。

6.1 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_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_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_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。可以存储一些简单的数据。
- **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_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**: 使用快速付立叶变换 (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

¹<http://www.soft-switch.org/>

后通知某人拨开指定的号码将来说话「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。

6.2 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。

6.3 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**:
- **mod_amrwb**:
- **mod_b64**: Base64 编码，可以传输任何数据
- **mod_bv**: BroadVoice 的编码
- **mod_celt**:
- **mod_codec2**:
- **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**:
- **mod_g729 只支持透传**:
- **mod_h26x**: 提供 H261, H263, H264 等视频编码
- **mod_ilbc**:
- **mod_isac**:

- **mod_mp4v:**
- **mod_opus:**
- **mod_sangoma_codec:** 通过硬件板卡支持包括 G729、iLBC 等多种编码
- **mod_silk:**
- **mod_siren:**
- **mod_skel_codec:** 例子模块
- **mod_speex:**
- **mod_theora:**
- **mod_voipcodecs:**
- **mod_vp8:**

6.4 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。

6.5 directories

目录服务。

- **mod_ldap:** 通过 LDAP 提供目录服务。

6.6 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_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。

6.7 event_handlers

事件处理。

- **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_json_cdr**: JSON 格式的 CDR。
- **mod_radius_cdr**: 将 CDR 写入 Radius 服务器。
- **mod_rayo**: Rayo 支持。
- **mod_snmp**: SNMP 网管接口。

6.8 formats

格式。

- **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_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。

6.9 languages

各种嵌入式编码语言接口。

- **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。
- **mod_yaml**: Yaml。

6.10 loggers

日志。

- **mod_console**: 控制台日志。
- **mod_logfile**: 日志文件。
- **mod_syslog**: 将日志写到 Syslog。

6.11 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_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_th**: 泰语。
- **mod_say_zh**: 汉语。

6.12 timers

定时器。

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

6.13 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。

附录 G Sofia Profile 参数参考

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：

```
<param name="dtmf-type" value="info"/>
```

- **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 向通过 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=<ip>:<port>"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：

```
<param name="auto-rtp-bugs" data="SONUS_SEND_INVALID_TIMESTAMP_2833"/>
```

- **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：

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

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" />
```

附录 H 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>。

插图列表

2.1 汇接局模式	11
3.1 双汇接局双归属网络拓扑	14
4.1 有长途局的网络拓扑	16
5.1 FreeSWITCH 作为 PBX	17
5.2 PBX 上带网关的结构图	25
6.1 X-Lite 添加账号	29
6.2 iDoubs 注册设置	30
7.1 Yealink 话机的账号配置界面	33
7.2 GrandStream 话机的账号配置界面	33
9.1 FXS 和 FXO 连接示意图	39
9.2 FreeSWITCH 连接 FXS 及 FXO 的拓扑结构	39
9.3 FreeSWITCH 通过网关连接模拟话机及电话线	40
9.4 Grandstream HT701 的配置	41
9.5 MX8 网关 FXS 口的配置	42
9.6 MX8 网关 FXO 口配置	44
10.1 E1 自环电缆的接法	58
10.2 添加 PRI 中继	64
10.3 E1 中继状态	65

10.4 添加 SIP 中继	65
10.5 添加 PSTN 到 SIP 路由	66
10.6 添加 SIP 到 PSTN 路由	67
10.7 呼叫流程	68
10.8 添加 ISDN 中继	71
11.1 网关初始页	74
11.2 添加 SIP 中继	75
11.3 配置 PSTN 来话路由	76
11.4 设置 DID 以避免播放二次拨号音	76
16.1 图 15-7 通过 Proxy 注册	97
18.1 Flash 测试页面初始界面	111
18.2 将 Flash 软电话登录到 FreeSWITCH 上	111
18.3 RTMP 来电提示页面	112
18.4 Flash 加入 FreeSWITCH 视频会议	113
18.5 在 Expert 模式进行设置	116
18.6 使用 sipML5 进行 WebRTC 视频呼叫	117

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是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，不仅为国内大家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，可以联系通过电子邮件联系我们**赠送**一本精美打印的纸质书。

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（右名：Sevan Du）资深架构师，通信技术专家，在智能通信领域耕耘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，不仅为客户提供过众多的VoIP解决方案，还提供过各种开源解决方案，而且客户遍及全国。热爱开源，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”。由FreeSWITCH开源项目主要贡献者之一，同时也是FreeSWITCH的维护者，不仅多次在国内外的开源项目上发表演讲，而且还发表了多篇国际级和国内期刊文章。此外，他已经将FreeSWITCH(Wall)上早已经使用的心得和经验，自FreeSWITCH INC CEO和CTO的职位辞职回国后，于2011年，帮助启动了FreeSWITCH中文社区。2012年，由Sevan Du发起的FreeSWITCH中文社区在刚刚结束的CicaCon大会，开始成为主要演讲者。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 权威指南

杜金房 张令芳◎著



内容简介

FreeSWITCH是世界上最受欢迎的一个平台，拥有数百万的、分布极广的、多层次的社交系统。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FreeSWITCH领航第一人”、全球FreeSWITCH开源社区知名专家撰写，FreeSWITCH之父Anthony Minasale倾力打造，汇集FreeSWITCH使用、维护、二次开发、高级分析等各方面；实践性强，社会大量实例，从单个独立应用的实现到集群的部署，应有尽有。



ISBN: 978-7-111-46624-6
定价: 65.00元

印制: 978-7-111-46625-3
定价: 65.00元

印制: 978-7-111-46626-0
定价: 65.00元

THIS PAGE INTENTIONALLY LEFT BLANK.