FreeSWITCH 权威指南（附录）

杜金房　张令考

本文档是《FreeSWITCH 权威指南》(ISBN 978-7-111-46626-0)一书的电子版附录。由于原书太“厚”了，因此单独将这一部分以电子版形式发布。

本书网站：book.dujinfang.com 。

目　录

附录 E　FreeSWITCH 中文 FAQ

本文的部分内容选自 FreeSWITCH 官方 FAQ[1]，部分问题是大家在 QQ 群或邮件列表中经常问到的问题，也有媒体对 Anthony 的采访等。有些问题在本书中直接有答案，我们便进行了简单回答，并指明了应该参考的章节。

1. 什么是 FreeSWITCH？FreeSWITCH 能做什么？

根据官方的定义，FreeSWITCH 是世界上第一个跨平台的、伸缩性极好的、开源免费的、多协议的电话软交换平台。FreeSWITCH 中实现了通过 SIP、IAX2、H.323、Skype、Jingle (Google Talk)、本地声卡、以及 TDM 语音卡等协议及接口的互连互通。你可以把它用做一个 VoIP 通信服务器，协议转换网关，语音及视频会议服务器、执行由 JavaScript、Perl、Lua 或 C# 写的 IVR 语音菜单脚本的 IVR 服务器等。另外它也具有丰富的接口与其它系统集成。详见第 3 章的有关内容。

2. FreeSWITCH™ 是用什么语言写的？

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

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

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

4. FreeSWITCH 与 Asterisk 有何异同？

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

5. PBX 与软交换的区别是什么？只是语义问题吗？

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

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

[1]参见http://wiki.freeswitch.org/wiki/FreeSwitch_FAQ 。

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

7. FreeSWITCH 可以商用吗？

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

8. 你们做该项目多久了？能简单说一下它的历史吗？

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

9. 什么是 ClueCon？

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

10. FreeSWITCH 能同时支持多少路通话？有基准测试吗？

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

11. FreeSWITCH 一定要以 root 用户运行吗？

FreeSWITCH 可以以任何用户运行，只要它有相关目录的写入权限。

12. 在 Sofia 呼叫字符串中，"%"和"@"有何不同？

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

13. ${var}与$${var}有何不同？

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

14. set和export App 有何区别？

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

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

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

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

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

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

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

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

18. 如何调试 SIP？

在控制台上执行“sofia global siptrace on”。另参见第 10.1.2 节，以及：http://wiki.freeswitch.org/wiki/Debugging_Freeswitch 及http://wiki.freeswitch.org/wiki/Sofia 。

19. 我收到“Invalid Application <name>” 是什么意思？

该错误最可能的原因是你没有加载正确的模块。

20. ICMP error是什么错误？

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

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

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

22. 我看到“Over Session Rate of 30”，是什么意思？

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

23. 在哪里下载 FreeSWITCH？

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

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

24. Pastebin 是什么？

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

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

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

26. 我遇到“NATIVE SQL ERR [database is locked]”。

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

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

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

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

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

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

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

30. FreeSWITCH 默认都会占用哪些端口？这些端口可以改变吗？

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

31. FreeSWITCH 可以对接 IMS 或在 IMS 中使用吗？

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

32. 我如何让 FreeSWITCH 运行在后台？

```
# freeswitch -nc
```

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

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

34. FreeSWITCH 运行在后台时，我如何停止它呢？

在操作系统上使用命令：

```
# freeswitch -stop
```

或在 FreeSWITCH 控制台中运行

```
freeswitch> shutdown
```

35. 如何让 FreeSWITCH 以更高的优先级运行？

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

```
# freeswitch -hp
```

36. 如何将 FreeSWITCH 注册为一个 Windows 服务？

在你安装 FreeSWITCH 的路径中，执行

```
FreeswitchConsole -install
```

或者，删除该服务

```
FreeswitchConsole -uninstall
```

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

你可以在命令行模式下启动和停止 FreeSWITCH：

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

如果使用“FreeswitchConsole -install”建立的启动项目不能启动，还可以试试其他的办法，如：。下载 winserv 并放到某一位置，如“C:\Program Files”。然后可以使用以下命令安装服务：

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

37. 如何在一台服务器上运行多个 FreeSWITCH 实例？

参见第 13.1 节。

38. 如何对接多台 FreeSWITCH？

参见第 13 章。

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

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

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

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

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

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

以使用以下命令进行配置：

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

42. 我如何选择编译哪些模块？

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

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

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

44. 在长时间收不到 RTP 后可以自动挂断电话吗？

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

45. 如何在 FS 控制台上看到 SIP 用户的注册情况？

可以在控制台上使用如下命令显示 Profile 的用户注册信息：

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

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

是的，通过 FreeTDM 模块（`mod_freetdm`）。详见 14.3、14.4 节以及http://wiki.freeswitch.org/wiki/FreeTDM 。

47. 如何在 FreeSWITCH 中发起一个呼叫？

使用`originate`命令，参见第 4.5 节及第 10.4 节。

48. `reloadxml`能重载所有 XML 文件吗？

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

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

有几种不同的实现方法：

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

50. 如何在整个服务器上使用单一的`domain`？

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

```
<param name="force-register-domain" value="domainname"/>
<param name="force-register-db-domain" value="domainname"/>
```

详见 15.7.1 节。

51. 是否有一个配置 FreeSWITCH 的图形界面？

参见第 10.6 节。

52. 有帮助文档吗？

是的，Wiki: http://wiki.freeswitch.org/ 是 FreeSWITCH 文档的大本营，自 2014 年 5 月份已迁移至http://confluence.freeswitch.org ，后者提供更好的 SSO 及更新的文档，同时前者已变成只读状态不再更新。另外，本书就是最好的中文文档。其它的中文的文档见：http://www.freeswitch.org.cn 。

53. 有没有关于排错与汇报 BUG 的指南？我该从哪里开始呢？

参见本书第 22 章。另外，从这里开始，Reporting Bugs：http://wiki.freeswitch.org/wiki/Reporting_Bugs ，它会回答你很多问题。

54. 我还有很多问题在这里没有列出，请问在哪里可以获得帮助？

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

附录 F　Sofia Profile 参数

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

- **media-option**：媒体选项。该选项有两个取值。

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

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

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

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

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

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

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

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

- **sip-trace**：是否开启 SIP 消息跟踪。如：

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

另外，也可以在控制台上用以下命令开启和关闭 SIP 跟踪，如：

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

- **log-auth-failures**：是否将认证错误写入日志。

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

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

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

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

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

- **sip-port**：设置监听的 SIP 端口号。如：

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

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

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

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

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

- **dialplan**：设置 Dialplan 的类型。如：

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

- **dtmf-duration**：设置 DTMF 的时长。如：

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

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

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

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

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

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

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

- **rtp-ip**：RTP 使用的地址。如：

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

- **sip-ip**：SIP 监听的 IP 地址。如：

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **record-path**：录音文件的默认存放路径。如：

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

- **record-template**：录音文件名模板。如：

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

- **manage-presence**：是否支持列席（Presence）。如果不用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 Apperance）。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。即在桥接电话是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：
- **disable-srv**或**disable-naptr**：这两个参数可以规避 DNS 中某些错误的 SRV 或 NAPTR 记录。如：

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

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

附录 G　使用 GSM 网关连接 PSTN

在某些特殊的场合使用 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 G.1 所示：

图 G.1: 网关初始页

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。

G.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。

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。

G.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，我们也需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 G.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。

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 G.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 G.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 13366938607 <>->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 G.2: 添加 SIP 中继

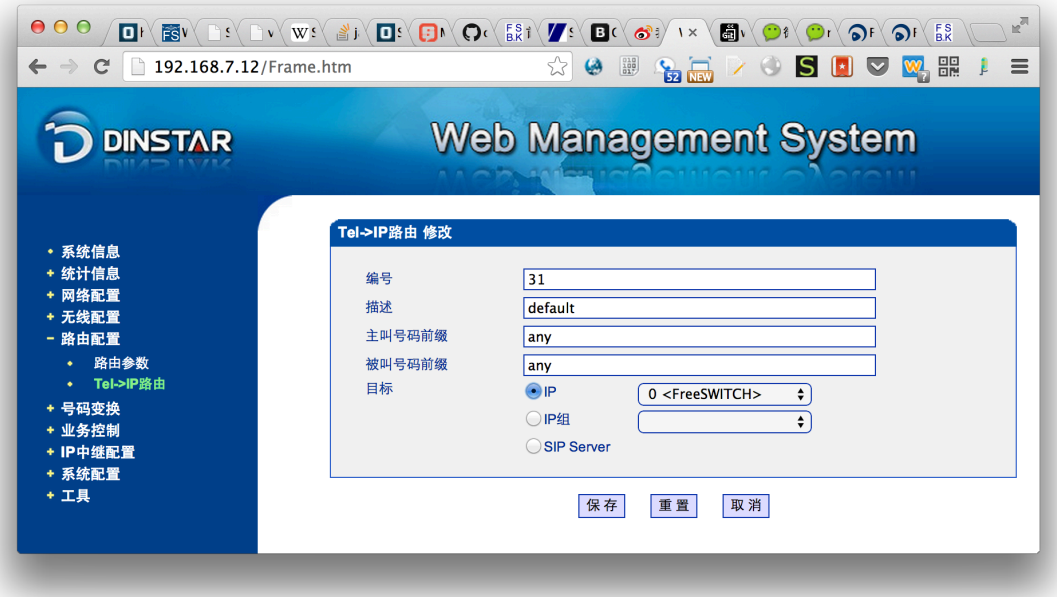

图 G.3: 配置 PSTN 来话路由

图 G.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。

附录 H　Sangoma 板卡及驱动的安装

在第 14.4.1 节，我们讲了`mod_freetdm`模块的安装。该模块在安装前需要先 Sangoma 板卡及驱动。但由于这部分内容篇幅比冗长，因此，我们把它单独列在这里。

H.1　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 ）。

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，我们就按第 3.2.1 中第 3 节所述的安装好 FreeSWITCH 的基本系统。

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。

H.2　安装及配置 Wanpipe 驱动

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[1]。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[2]列举了以下一些依赖项，读者可以作一个参考。

```
C developement tools ...(gcc)
C++ developement tools
Make utility
Ncurses library
```

[1]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*）赋于执行 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。

[2]参见http://wiki.sangoma.com/Wanpipe-Requirements 。

```
Perl developement 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[3]：

```
# 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 libsqlite3-dev
```

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）：

[3]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。

```
# 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 1，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[4]，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：

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[5]。

```
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。

[4]与以太口 RJ45 在物理上是一样的，只是线序不同。
[5]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。

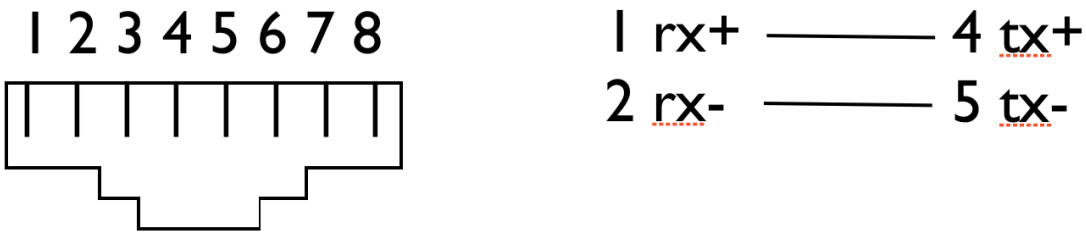

图 H.1: E1 自环电缆的接法

H.3 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。
- **ftdm_sangoma_isdn**：该模块使用 Sangoma 私有的商业电信级的 Trillium 协议栈[6]，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/libsng_isdn/libsng_isdn-current.i686.tgz
# tar zxvf libsng_isdn-current.i686.tgz
# cd libsng_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/libsng_isdn/libsng_isdn-current.x86_64.tgz
# tar zxvf libsng_isdn-current.x86_64.tgz
# cd libsng_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。参见 14.4.1 节。

[6]参见 http://wiki.sangoma.com/Freeswitch-FreeTDM-Sangoma-ISDN-Library 。

附录 I　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。

[1] http://www.cluecon.com/anthm.html

Asterisk 的核心是多线程的，但它非常保守。仅仅用于初始化的信道以及执行一个程序的信道才有线程。任何呼叫的 B 端都与 A 端都处于同一线程。当某些事件发生时（如一次转移呼叫必须首先转移到一个称作伪信道的线程模式），该操作把一个信道所有内部数据从一个动态内存对象中分离出来，放入另一个信道中。它的实现在代码注释中被注明是“肮脏的”[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杰然不同的方式来实现，这种重复在代码中随处可见。

这仅仅是我在 Asterisk 中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

[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

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(Formats)：文件格式如 wav。
- 日志(Loggers)：控制台或文件日志。
- 语言(Languages)：嵌入式语言，如 Python 和 JavaScript。
- 语音(Say)：从声音文件中组织话语的特定的语言模块。
- 计时器(Timers)：可靠的计时器，用于间隔计时。
- 应用(Applications)：可以在一次呼叫中执行的程序，如语音信箱(Voicemail) 。
- FSAPI(FreeSWITCH 应用程序接口)：命令行程序，XML RPC 函数，CGI 类型的函数，带输入输出原型的拨号计划函数变量。
- XML：到核心 XML 的钩子可用于实时地查询和创建基于 XML 的 CDR。

所有的 FreeSWITCH 模块都协同工作并仅仅通过核心 API 或内部事件相互通信。我们非常小心地实现它以保证它能正常工作，并避免其他外部模块引起不期望的问题。

FreeSWITCH 的事件系统用于记录尽可能多的信息。在设计时，我假设大多数的用户会通过一个个性化的模块远程接入 FreeSWITCH 来收集数据。所以，在 FreeSWITCH 中发生的每一个重要事情都会触发一个事件。事件的格式非常类似于一个电子邮件，它具有一个事件头和一个事件主体。事件可被序列化为一个标准的 Text 格式或 XML 格式。任何数量的模块均可以连接到事件系统上接收在线状态，呼叫状态及失败等事件。事件树内部的`mod_event_socket`可提供一个 TCP 连接，事件可以通过它被消费或记入日志。另外，还可以通过此接口发送呼叫控制命令及双向的音频流。该套接字可以通过一个正在进行的呼叫进行向外连接(Outbound)或从一个远程机器进行向内（Inbound)连接。

FreeSWITCH 中另一个重要的概念是中心化的 XML 注册表。当 FreeSWITCH 装载时，它打开一个最高层的 XML 文件，并将其送入一个预处理器。预处理器可以解析特殊的指令来包含其他小的 XML 文件以及设置全局变量等。在此处设置的全局变量可以在后续的配置文件中引用。

如，你可以这样用预处理指令设置全局变量：

```
<X-PRE-PROCESS cmd="set" data="moh_uri=local_stream://moh"/>
```

现在，在文件中的下一行开始你就可以使用`\$\$(moh_uri}`，它将在后续的输出中被替换为`local_stream://moh`。处理完成后 XML 注册表将装入内存，以供其他模块及核心访问。它有以下几个重要部分：

- 配置文件：配置数据用于控制程序的行为。

- 拨号计划：一个拨号计划的 XML 表示可以用于`mod_dialplan_xml`，用以路由呼叫和执行程序。
- 短语：可标记的 IVR 分词是一些可以“说”多种语言的宏。
- 目录：域及用户的集合，用于注册及账户管理。

通过使用 XML 钩子模块，你可以绑定你的模块来实时地查询 XML 注册表，收集必要的信息，以及返回到呼叫者的静态文件中。这样你可以像一个 WEB 浏览器和一个 CGI 程序一样，通过同一个模型来控制动态的 SIP 注册，动态语音邮件及动态配置集群。

通过使用嵌入式语言，如 Javascript、Java、Python 和 Perl 等，可以使用一个简单的高级接口来控制底层的应用。

FreeSWITCH 工程的第一步是建立一个稳定的核心，在其上可以建立可扩展性的应用。我很高兴的告诉大家在 2008 年 5 月 26 日将完成 FreeSWITCH 1.0 PHOENIX 版。有两位敢吃螃蟹的人已经把还没到 1.0 版的 FreeSWITCH 用于他们的生产系统。根据他们的使用情况来看，我们在同样的配置下能提供 Asterisk 10 倍的性能。

我希望这些解释能足够概括 FreeSWICH 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的工具吗？”

附录 J　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，我不仅可以使我自己的电话可以与计算机通信，我还可以让它在的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 Perl[2]，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补之后能够得以运行也算是一个不小的成熟了。

随后，我深入了 Asterisk 在线社区。在这些代码上玩了几周的之后，我用 Asterisk 作为电话引擎开始做一个呼叫中心解决方案，以及一个简单的 Web 前端程序。在实现的过程中，我遇到了 Asterisk 中的几个 Bug，然后我就把它们提交到了 Asterisk 开发分支的缺陷跟踪系统上。该过程重复地越多，我参与该项目的成长和发展不越深。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。

J.1　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

[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。

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 trail-and-error），我成功了。我终于通过一个呼叫发生器的帮助找到了导致系统崩溃方法。当时的感觉非常好，只是好的感觉太短暂。就在那天下午，我又在代码的另一处发现了另外一个类似的新问题。

我尝试小心的去掉我的程序中的一些可能导致死锁或崩溃的功能，便是我无法去掉全部。最终我发现导致我的不幸的是`app_queue`模块，这对我来说可不是个好消息，因为在我的呼叫中心程序中我主要就用了那个模块。我修复了那个模块中的问题，但一些修改对原有的模块影响太大，因而无法包含到主流的发布代码中。最后，我还是自己维护了我的模块代码，并继续更新 Asterisk 中其它的部分。这总算令它稳定下来了，但这种稳定只是在找到另一种解决方案前相对稳定的方案。

到那时为止，我往 Asterisk 里加入了很多的新特性，并对开发一些性功能有了很好的想法。我创建了一个新的概念，叫做功能变量（function variables），它允许模块可以对外提供一个接口，通过该接口，能从拨号计划（Dialplan）中扩展模块的功能（如果你读了本书，你会感觉本书中同样实现了类似的想法）。与此同时，我仍然在纠结那个队列死锁问题。所以后来我与 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。

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 主机组成的集群，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。

J.2 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 Astersisk 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。

[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。

即使在这么紧张的环境下，我们还是全力以赴地支持该项目真正希望它能成功。我们继续好召开每周的电话会议，他们也确实采取一些措施开始帮助开发者们增加动力。我们觉得我们应该有一场现场的见面会，以便我们所有人都可以聚到一起分享我们的电话技术知识，并一起玩几天。我们不知道我们要干什么，但我们决定要做，并把该聚会称为 ClueCon。有一个 Clue[4]意味着你知道你要做什么，所以 ClueCon 的意思就是帮每一个人找到一个“Clue”。我承认，即我刚刚说过我们也不知道我们要做什么。看起来非常有意思，一群没有 Clue 的人开了一个有 Clue 的会——ClueCon。不过，事实证明非常幸运，这好像根本不是个问题。前面我们所指的 Clue 都是指电话技术，而不是做会议。

因此，在第一届 ClueCon 之前的几个月中，即 2005 年春天，我们在一次例行的电话会议中开始专门深入讨论 Asterisk 中的几个缺点。这非常正常，因为我们主要的目标就是找出这些问题并找到解决方案。在那个时期，有一大群不守规矩的、受够了他们在 Asterisk 上遇到的无穷无尽的问题的人。那群人中的很多人都参加了那次周会，希望能说服我们帮忙看一看他们遇到的问题。我想得越多，越觉得解开折磨我们的核心架构问题越是任重道远。Asterisk 中的很多核心都具有单一的性质而无法扩展（Scale），其中的很多我发都有很多用户依赖于它们，任何企图改进他们的动作都有可能会引发功能上的退化。有些问题看起来是无解的，除非用一把大锤把那些旧代码砸个稀巴烂，并从核心的代码深入重写。但这种方法看似不可行，因为它将会使得 Asterisk 在几个月内甚至一年或更长的时间内都不可用。也就在那时候，我有了一个想法——我们做一个 2.0 版吧。

从一个 2.0 版并不是一个最坏的想法。我知道它将有很多挑战，但是，我想我们可以与旧的代码并行启动一个新的代码库，那样我们就可以删除那些有问题引起问题最多的部分旧代码并替换成新的，并仍然维护用户依赖的一些仍然可用的代码。有了这个想法后我感到很兴奋，同时也在我提出该想法后看到该项目的领导人的反映时得到了同样地震惊。他似乎也对我竟然提出这么一个想法同样震惊。总之，简单来讲，我们没有做 2.0 版。那时，我有无数的想法，我也非常清楚地知道我喜欢以及不喜欢 Asterisk 的地方，但没地方写。

我盯着那个在一个空目录中打开的空的编辑器缓冲区[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，而不想到时候太仓促。

J.3 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，我们真正使得每都人都开始想问题。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 Fork[6]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

[4]线索，后面的 Con 指会议（Conference）。——译者注。

[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。

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 Tootisie Pop commercials 的 Owl 先生）问我："你觉得让你的新代码能打电话需要多称时间呢？"我不知道，所以我决定试一下——不管是一周、两周还是三周。

我写的第一个能够发出声音的模块是`mod_woomera`，该模块是一个 Endpoint 模块，它使用了 Craig Southeren（与我在 ClueCon 上遇见的是同一个人）写的 Woomera 协议。我也为 Asterisk 写了一个类似的模块。Woomera 协议非常简单，它并不需要编解码或其它复杂的东西。它的实现思想是它屏蔽了 H323 协议的复杂性并允许应用程序使用该简单的协议与它通信以便更容易地集成到 VoIP 程序中。所以，从它开始好像是一个正确的选择。当我开始工作的时候，我意识到在我的核心代码中需要更多的元素，然后我就慢慢的添加，并最终将这些代码融合到一处，我终于可以给以 Woomera 协议武装的 H323 监听进程打电话了，同时，我也可以在我的 Pandora 代码里获取通话的状态。是的，我把我的项目名称改成了"Pandora"，因为大家都不喜欢 Choir 这个名字。我非常愉快的听着 Alan Parsons Project hit Sirius stream 第一次从我的扬声器里流出来。这一次比我第一次使用 Asterisk 打电话时更加兴奋，因为我白手起家从头开始写的代码现在开始工作了。

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` 。

J.4 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，从一开始就跟我在一起。Mike[9]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

[7]原文是 Check out，即从 SVN 仓库检出代码。
[8]FreeSWITCH 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。
[9]Michael 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。

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 Asterish 曾试图从头到始写一个 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人宣布了一个官方的新版本将“很快发布（oming soon）”，条消息的发布时间是 2006 年 1 月，当时我们拼命的想让所有事情按我们希望的那种方式工作。我们非常希望专注于在添加任何其它功能前做一个稳定的核心，并且我们也有了很大的进展，但是我们还是没有准备好发布 1.0 版。

2008 年春天我们有了稳定的 SIP、有了 Event Socket 用于远程控制 FreeSWITCH、有了一个模块可以通过 HTTP 执行 FreeSWITCH 命令、有了 XML curl 等等一系列的新功能和特性。我们最终觉得可以发布一个版本了。所以我们就一举发布了 FreeSWITCH 1.0 凤凰版（Phoenix）。我之所以将该版本取名为凤凰，是因为感觉到我们所有的辛苦的工作成果都是从先前的失败的骨灰中来的，并且这个名字也被很多其它人使用，包括 NASA 在同一时刻将“凤凰号”送上了火星。总之我认为那是一个很合适的名字。

在 ClueCon 2008 上，我们又一次宣布了曾于当年 5 月份发布的 1.0 版。当时还有另外一些与 FreeSWITCH 有关的演讲，以及与 Asterisk 有关的演讲，因为当前也发布了 Asterisk 1.6。在当前接下来的时间里，我们用了所有的时间专注于宽带（高清）语音的支持以及其它的一些功能，例如即时的进行采样率转换。此外，我们还增加了一些新的 SIP 功能，如状态呈现（SIP Presence）以及其它的一些简单通话以外的功能，并于后来发布了 1.0.1 版。

2009 年，我们发布了 1.0.2 至 1.0.4 并于第 5 届 ClueCon 年度会议上又一次演示了 FreeSWITCH。到那时候，我们早期的一些创新变成了现实。因为我们可以演示使用 Polycom 话机新的 Siren 编码进行高清语音的通话，并且我们还支持了与 Skype 互通。当年的 FreeSWITCH 演示概括了一些你可能根本就意识不到你你可以做的事情，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 Tower[11]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。

[11]一个国际性酒店。——译者注。

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。

[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。

写在最后

在笔者及出版社编辑和工作人员的共同努力下，本书终于要和广大读者见面了。欣慰之余，也算是给经历了漫长的等待时间的读者一个交代吧。

本书的写作时间跨度较长，而 FreeSWITCH 又更新太快，以至于有些章节都先后修改了好几遍。尤其是 FreeSWITCH 自去年起分成了两个分支，兼顾两个分支无疑大大增加了写作的难度。本书截稿后到出版前的这段时间，FreeSWITCH 又有了很大的变化，使我们多少有些被动。不过，令人高兴的是，FreeSWITCH 在三天前发布了 1.4.4 版—这是 FreeSWITCH 1.4 自去年发布以来的第一个正式版！同时，这也意味着，FreeSWITCH 自 1.2 版以来巨大变化的节奏终于可以消停一会儿了。这也使得笔者在本书即将面市之即，还有机会在电子版附录里写点东西。

FreeSWITCH 现在有两个版本—1.2 版和 1.4 版。它们之间的最大区别就是后者支持 WebRTC。为了支持 WebRTC，FreeSWITCH 核心代码也进行了一系列的重构，这些内容都已经涵盖在本书里了。而在本书截稿之后 FreeSWITCH 中最大的变化就是对 FS-353（jira.freeswitch.org/browse/FS-353）的支持。事情还得从多年前说起—在 FreeSWITCH 中，为了避免重复发明轮子，使用了很多的第三方库。最初大部分第三方库的代码也放在 FreeSWITCH 的代码库中，静态编译连接，这样非常易于从源代码进行编译；而 FS-353 指出，这种方式不利于将 FreeSWITCH 打包加入各发行版（典型的如不同的 Linux 发行套件），因而最新的 FreeSWITCH 不再使用自己代码库中的第三方代码，而是更多地依赖于操作系统提供的第三方库。现在，如果需要编译最新的 FreeSWITCH 代码，除了按第 3 章描述的安装依赖库外，还需要安装更多的依赖库。当然，它也会带来很多好处，其中最重要的一点就是 FreeSWITCH 有望进入各发行版，而普通人员将再也不需要从源代码编译安装，而是可以直接使用发行版提供的工具如 apt-get 或 yum 等来安装 FreeSWITCH 了。关于这一点更详细的解释请参考我专门写的文章（zhuanlan.zhihu.com/freeswitch/19746509），在此，就不赘述了。

在 FreeSWITCH1.2 版开发时，大部分是基于 CentOS 5 的，而随着时间的推移，CentOS 5 已经老去（官方已停止支持），而新的 CentOS 6 一直有一些性能问题（可能到现在已经解决了），因而 FreeSWITCH 的开发者们在开发 1.4 版时都纷纷转向了 Debian 7（同样老去的还有 Debian 6）。所以，1.2 版适于 CentOS 5 及 Debian 6，而 1.4 版本最好是在 Debian 7 或 CentOS 6 上运行[15]。当然，除 Linux 外，1.4 版还支持 Windows、Mac OS X、BSD 系列、Solaris 等多种操作系统。

在本书写作时，就最大程度地兼顾了 1.2 版和 1.4 版。除了上面讲的几点老需要注意外，本书讲的内容都适合两个版本[16]。因而，本书的内容在相当长的时间内都不会过时。当然，如果读者读完本书，对 FreeSWITCH 就应该了如只掌了，那时候如果再遇到什么问题，也都可以很容易地解决了。

除代码外，FreeSWITCH 的文档系统也将由wiki.freeswithc.org切换到confluence.freeswitch.org，后者将提供更好的 SSO（单点登录）以及更专业的文档，但前者也将在一定时期内以只读的形式继续存在，以便读者参考。

无论如何，本书的在线站点（book.dujinfang.com）都会有本书最新的更新以及勘误等，也欢迎广大读者订阅微信公众社区“FreeSWITCH-CN”获取更多更及时的信息。

杜金房/2014 年 5 月 26 日于烟台

[15] 当然，这并不是说 1.4 版不能在 CentOS 5 及 Debian 6 上运行，只是，需要费点劲，参见https://confluence.freeswitch.org/display/FREESWITCH/CentOS+5 。

[16] 当然，除了有些特性（如 WebRTC）仅在 1.4 版上存在。

THIS PAGE INTENTIONALLY LEFT BLANK.