第32章VoIP 服务器——Asterisk

可以自己用软件实现电话交换机？是的，这就是本章我们要讨论的话题。如何在 Linux 下实现免费的电话交换机功能。

32.1 用 Linux 实现免费电话大餐

在架设自己的电话系统之前，我们先看看传统的电话网络是如何工作的。传统的电话网络，叫做 Public Switched Telephone Network，公共交换电话网络，简称 PSTN，是一种全球联网的语音通信电路交换网络，通过中继和交换设备进行通讯。不难看出，我们现在使用的 IP 计算机网络和 PSTN 颇有几分相似之处，在 PSTN 中呼叫一个电话号码相当于通过 IP 网络到达一个 IP 地址，在骨干 PSTN 网络中，也和 IP 协议一样使用复用机制来传递数据。

对于私有电话系统来说，也和 IP 计算机网络有着类似的原理。公司的电话交换机直接对外（相当于 IP 网络中的公网地址），外部人员只能访问到电话交换机总机，由总机控制应该将这个连接转到哪个分机上（类似于 IP 中的 NAT），内部的分机不是合法的电话号码（192.168.x.x？），只有通过公司的电话交换机才能与外界正常通讯。

在这里，公司的电话交换机，也就是通常所说的程控交换机，也叫 PBX，Private Branch(telephone) eXchange 的缩写，意思是私有/专有电话交换系统，传统的 PBX 使用 PSTN 网络连接并提供语音通话服务。

随着技术的发展，出现了基于 IP 网络的语音传输业务，即 Voice over IP，简称 VoIP，人们可以通过计算机网络（通常是指 Internet 或 Intranet）进行语音通话，而后，又出现了基于 IP 网络的 PBX, 简称 IPPBX，IPPBX 不需要专用的网络，可以通过 Internet 和 VoIP 实现类似于传统电话系统的功能，而且，使用 IPPBX 进行通信，对内没有任何费用，只有网络建设成本，对外只需要支付接入到

Internet 的费用，没有任何额外的通话或漫游、长途费用，IPPBX 的优势不言而喻，而通过一些硬件设备（比如调制解调器），IPPBX 也可以和现有的传统 PSTN 网络互通。

也就是说，在所有使用 IPPBX 的用户之间，我们的通话都是通过 Internet 而非电话局的专有网络进行，都是免费的，而且不受任何电信运营商的控制，这对传统电话业务造成了很大的冲击，也使得 IPPBX 开始迅速发展。

随着计算机硬件处理能力的加强、计算机骨干网络传输能力的提升和 VoIP 软件技术的进一步发展，纯软件的 IPPBX 开始出现，相当于无需增加任何设备，一台 PC 就可以作为一个 IPPBX 使用，提供电话系统的功能，这大大节省了公司的成本和日常费用，同时，随着 Linux 的兴起，基于 Linux 的开源 IPPBX 软件出现了，也就是说，我们可以完全免费并且是合法的构建起自己的 VoIP 电话系统。而这些开源 IPPBX 软件中的佼佼者，就是 Asterisk。

32.2 用 Asterisk 提供免费的 VoIP 服务

1999 年，Linux Support Service 公司的 Mark Spencer 出于工作需要和资金的压力，开始捣鼓基于 Linux 的免费电话系统，因为当时 PBX 都是专有厂商而且价格昂贵（现在也是），他开始尝试自己开发纯软件的 PBX 系统，这就是 Asterisk 的起源。

随后，依靠全世界电信和程序爱好者的共同努力，Asterisk 开始迅速发展，并成为开源软件 IPPBX 领域的明星，2001 年，随着 Asterisk 的发展，Linux Support Service 改名为 Digium 公司，开始专门提供 Asterisk 服务及相关语音板卡硬件的销售业务。

之所以选择 Asterisk 这么古怪的名字（Asterisk 是*号的意思），只是因为创始人 Mark Spencer 选择了一个既存在于传统电话按键中，又存在于 Linux 中而且代表通配符的这样一个符号，我认为这也从侧面反应了他想让 Asterisk 成为基于 Linux 的电话系统标准的想法。

Asterisk 支持 几乎所有 Linux、 Unix 类操作系统 ，甚至可以支持 Windows 操作系统（目前最新版本的 Asterisk for Windows 是 0.6 版，构建于标准 Asterisk 1.2.13），除了提供最基本的 VoIP 语音通话功能，Asterisk 还支持传真、语音信箱、彩铃（RBT, Ring Back Tone）、电话会议（多方通话） 、IVR（Interactive Voice Response，互动式语音应答）等高级功能，Asterisk 的这些功能，和当前的任何硬件或商业软件 PBX，也可以说是丝毫不逊色，甚至在灵活性和可配置性方面比硬件 PBX 更加出色,相信接触过传统 PBX 系统的朋友一定感同身受，各式各样的古怪命令可不是那么好玩:-)。这些以往需要昂贵的硬件、软件和人员成本来构建的高级专业系统，才能完成的任务和功能，Asterisk 都可以完成，而 Asterisk 的座右铭是：这不过是软件！（It’s only software!）我们不禁会想，会不会有一天，硬件 PBX 完全消失，这个世界的电信系统是 Asterisk 的天下？

Asterisk 并不是一个特定的电话系统，它实际上是一个支持各种扩展的应用网关，我们可以在网关上开发出各种逻辑和流程，提供各式各样的服务。

例如，可以开发出基于 H.323、SIP、RTP 等协议的语音通话服务，可以支持 alaw/ulaw，GSM，ILBC，G.729a 等多种数据压缩格式，这样，就能以 Internet 为平台提供优质的语音服务。

虽然语音通讯是 Asterisk 的核心，但并不是 Asterisk 的全部，例如，可以围绕 Asterisk 方便的扩

展出传真、语音邮件等功能，甚至可能通过 Asterisk 的平台，用电话对系统进行管理。

Asterisk 如此强大的原因，是因为 Asterisk 具备完善的程序逻辑，这也可以说是 Asterisk 最耀眼的功能。例如，我们可以通过用户的按键操作，选择是为他接通某个分机还是给某人发邮件，或者把它加入到正在进行的某个会议中。我们可以控制当被呼叫的用户无法接通时，自动转到他的留言信箱，并把留言作为附件发送到用户指定的邮箱，总之，只要是可推断的操作，都能为之提供一定的功能作为响应。

我们也可以很容易就建立起企业自动电话服务系统，为客户提供已录制好的常见问题解答或企业服务介绍，依据用户的按键，选择播放不同的内容，当用户无法得到想要的解答时，还能及时把他们转到人工座席。

更重要的是，这一切，都是建立在免费的、纯软件的基础之上的。用户几乎不需要任何投入，只要有一台 586 PC，就能为企业提供完全免费的 VoIP 电话服务（当然，如果需要同时接入到传统电话网络，还是需要一点点投入的）。如果能为企业构建起这样一套电话系统，是不是很让人心动？还等什么呢？赶快投入到 Asterisk 系统中来吧！

32.3 获取 Asterisk 及其相关软件

前面说过，我们只需要一台 PC 就能构建出完善的商业级电话系统，但这个系统是基于纯粹的 Internet VoIP 的，如果要和传统电话互联，则需要投入一点费用用于申请电话号码和购买一块转接卡，用于将传统电话网络跟我们的 VoIP 电话网络进行互联，对于小型办公场所而言，我们涉及的传统电话都将是模拟信号，因此这块板卡通常都是模拟转换卡，也就是市面上常见的 Zapata 公司的 X100P 卡或兼容卡，它有专门的管理和驱动，这些驱动和管理工具软件被统称为 zaptel，用户可以通过 Asterisk 公司 Digium 网站 www.digium.com 或 Asterisk 网站 www.asterisk.org 获取。

如果我们使用一些 ISDN 设备比如数字方式的 PSTN 中继接入卡，那么还需要另一个库 libpri，它是 Primary Rate ISDN 的缩写，可以用于 ISDN 设备的管理和驱动，如果我们没有用到这种设备，那么就不需要它。

另外，Asterisk 默认只支持通过 ODBC 连接到外部数据源，因此，如果想使用 MySQL 来配置和管理 Asterisk，那么你需要安装 Asterisk-addons 这个包，它支持 asterisk 直接连接到 MySQL 数据库。

除此以外，还有 Asterisk-sounds 包，这个包主要是额外的语音文件，由于目前还没有中文语音，因此我认为对于我们中文用户来说意义不大。

上述软件都可以通过 Asterisk 网站找到。

最后，我们可能涉及的软件是 FreePBX（freepbx.org），它提供了一个管理 Asterisk 的 Web 界面，用户可以通过这个见面直观的对 Asterisk 进行管理，但 FreePBX 是基于 MySQL 的，它会覆盖掉所有现有的配置文件，而且它使用自己的配置数据结构，也就是说，对于已经存在的系统，你可能需要全部重新配置。这是我不喜欢 FreePBX 的原因。而且最新的 FreePBX 版本是 2.2.1，它只能支持 Asterisk 1.2.x，如果用来管理 1.4 版本的 Asterisk，可能会存在未知的问题。

通过集成 CentOS、Asterisk 和 FreePBX，Fonality 公司提供了一个完全傻瓜式的 Asterisk 系统（或

者叫产品系统），从安装操作系统到配置、管理路由和分机，都可以实现高度自动化，这个系统，被称为 Trixbox（www.trixbox.org），之前叫做 Asterisk@Home（简称 AAH），目前最新版本是 2.0。这是个非常流行的 Asterisk 系统，因为它有着迅速、方便的安装和管理能力，看上去更像一个成熟的产品而非黑客的玩具，它最适合于新买来的计算机，用户不需要什么前期准备，甚至不需要格式化硬盘，直接把光盘插入新买的计算机中，就能完成所有的安装和配置。最近开发 Asterisk 的公司 Digium 现在也开始推出自己的类似产品——AsteriskNOW（www.asterisknow.org），它也是一个具备和 Trixbox 相似扩展功能的，集成了操作系统和 Web 管理界面的产品级别的 Asterisk 系统，不过目前版本还是 BETA 4，使用的人还不多，今后究竟哪个系统更加符合用户需求，应用更加广泛，还需要我们拭目以待。

32.4 Asterisk 及其相关软件的安装

首先，确保系统的编译环境是完好的，包括 gcc、c libary、make 和内核源文件 kernel-source 等软件都已经被安装了，虽然对于大多数系统来说，这些都是默认安装的，但对于少数系统来说，出于安全考虑，会清除掉软件编译的环境，或者没有安装 kernel source，那么在安装 Asterisk 及相关软件时很可能出现错误，对这类系统，就需要首先安装好上面提及的这几个软件包。比如，在 Debian 系统中，可以使用下列命令安装上面的包：

Debian: apt-get update;apt-get install gcc make libc6-dev linux-source

然后，我们将所有需要的软件包下载下来，包括 asterisk、asterisk-addons、zaptel、libpri 等，需要的软件具体说明如下：

软件名	关联性	下载地址
Asterisk	必须，核心	http://ftp.digium.com/pub/asterisk/
Zaptel	接入 PSTN 时必须	http://ftp.digium.com/pub/zaptel/
Libpri	使用数字接口接入 PSTN 时必须	http://ftp.digium.com/pub/libpri/
Asterisk-addons	使用 MySQL 做数据源时必须	http://ftp.digium.com/pub/asterisk/
Asterisk-sounds	更多的 IVR 语音	http://ftp.digium.com/pub/telephony/sounds/
unixODBC	使用 ODBC 方式连接到数据库时必须	http://www.unixodbc.org/
MySQL ODBC Connector	使用 ODBC 方式连接到 MySQL 数据库时必须	http://dev.mysql.com/downloads/connector/odbc/3.51.html#linux
spandsp	需要 Asterisk 收发传真时必须	http://www.soft-switch.org/downloads/snapshots/spandsp/
spandsp-asterisk 补丁	需要 Asterisk 收发传真时必须	http://www.soft-switch.org/downloads/snapshots/spandsp/

根据用户情况下载必须的软件，并把他们统一解压到某个目录比如/usr/src/voip 下，然后我们开

始各种关联软件的配置、编译和安装。

首先安装的是 zaptel 和 libpri，这两个软件包是针对需要和传统 PSTN 网络互联的情况，当然在目前的环境下，我相信绝大部分用户都是需要这么做的：

CentOS: cd /usr/src/voip/zaptel-x.x.x

CentOS: make;make install

如果没有报错，说明 zaptel 已经安装完成了，如果有报错，也通常会是系统编译环境的问题。根据不同的系统，zaptel 会自动将自己的模块安装到不同的位置，通常是/lib/modules/kernel-version/目录下，并保证系统在启动时就加载必须的模块，在 CentOS/RedHat 系统中，zaptel 会修改/etc/modprobe.conf 添加必要的 install 语句。我们稍后介绍如何使用 zaptel 配置和管理系统的接口卡，我们接着把 libpri 安装到系统中：

CentOS: cd /usr/src/voip/libpri-x.x.x

CentOS: make;make install

和 zaptel 一样，libpri 也很容易安装到系统中，如果系统编译环境正常，那么一般不会出现任何问题。

接下来是非必须的数据库和数据源程序，这仅仅在系统通过数据库进行配置或者将通话记录保存到数据库时进行。前面我们提到过，Asterisk 默认只提供 ODBC 方式连接到外部数据源，asterisk-addons 这个包提供直接连接到 MySQL 的能力。

而如果你的系统需要连接到 SQL Server 等数据库，或者不想通过 asterisk-addons 连接到数据库，那就需要使用 unixODBC 连接，通过 unixODBC 连接的先决条件是，需要首先安装 unixODBC 程序和库，首先安装 unixODBC：

CentOS: cd /usr/src/voip/unixodbc-x.x.x

或许我们不应该将 unixodbc 也放到 voip 子目录中，因为它不是一个 voip 应用，不过考虑到 Asterisk 通常会独占一台计算机，这里也就不深究了:-)

CentOS: ./configure --prefix=/usr/local/unixodbc --enable-gui=no

CentOS: make

CentOS: make install

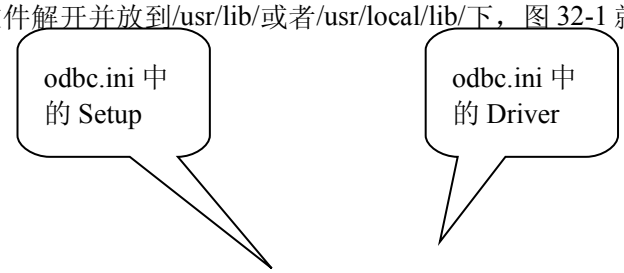
在配置 Asterisk 时加上--enable-odbc=你的 unixODBC 库目录，比如：

CentOS: cd /usr/src/voip/asterisk-1.4.x

CentOS: ./configure --enable-odbc=/usr/local/unixodbc

以上是假设你的 unixodbc 安装到了/usr/local/unixodbc 下，如果把 unixODBC 的库、头文件等安装到了系统自动识别的路径下如/usr/include，那么无需指定路径。

如果通过 unixODBC 连接到 MySQL 数据库，那么你还需要下载 MySQL ODBC 连接器的库（下载地址见前面的软件列表），将下载后的文件解开并放到/usr/lib/或者/usr/local/lib/下，图 32-1 就是一个放置到/usr/loca/lib/下的例子：



```
[root@CentOS ~]# ls /usr/local/lib/mysql/
libmyodbc3-3.51.14.so        libmyodbc3.la            libmyodbc3S-3.51.14.so
libmyodbc3_debug-3.51.14.so  libmyodbc3_r-3.51.14.so  libmyodbc3S.la
libmyodbc3_debug.la          libmyodbc3_r.la          libmyodbc3.so
libmyodbc3_debug.so          libmyodbc3_r.so          libmyodbc3S.so
[root@CentOS ~]#
```

图 32-1 MySQL ODBC 文件

unixODBC 连接到数据库的安装就完成了，具体配置我们稍后会提到。

如果我们同时需要 Asterisk 能够接收和发送传真（我相信大部分人即便不需要传真，也乐意尝试一下 Asterisk 的传真功能），那么还需要另一个应用/模块，spandsp 及其 Asterisk 的补丁，可以从 www.soft-switch.org 得到它，spandsp 使用 tiff 格式存储传真，因此系统中要有 tiff 的开发库，在 RedHat/CentOS 中，通常叫 ibtiff-devel，而在 Debian 中通常叫 libtiff-dev，使用以下命令可以在 Debian 上直接安装上 tiff 开发包：

Debian: apt-get install libtiff-dev

或者在 CentOS 上使用命令：

CentOS: rpm –ivh $LOCATION_TO_LIBTIFF_DEVEL_RPM

在安装完 libtiff 后，开始安装 spandsp。

CentOS: cd /usr/src/voip/spandsp-x.x.x

CentOS: make;make install

只要保证系统编译环境正常，并且安装了 libtiff 开发库，那么 spandsp 的编译、安装也不会出现什么问题。

spandsp 默认安装到/usr/local 下，完成安装后，可能需要设置环境变量 LD_LIBRARY_PATH 加入此目录：

CentOS: export LD_LIBRARY_PATH=$ LD_LIBRARY_PATH:/usr/local/lib

并将目录/usr/local/lib 加入到/etc/ld.so.conf 中，然后运行 ldconfig 使之生效。

CentOS:ldconfig

spandsp 就绪后，开始安装 spandsp 针对 Asterisk 的扩展和补丁，这通常是三个文本文件：app_rxfax.c、app_txfax.c 和 asterisk.patch，把 app_rxfax.c 和 app_txfax.c 拷贝到 Asterisk 目录下的 app 子目录，将 asterisk.patch 拷贝至 asterisk 源文件目录，并运行

CentOS: cd /usr/src/voip/asterisk-1.4.x

CentOS: patch < asterisk.patch。

patch 程序将自动搜索需要进行修改的文件及其位置，并进行修改，但由于 asterisk 和 spandsp 的开发不同步，在对最新版的 Asterisk 打补丁时，可能会出现 spandsp 提供的补丁定位与 asterisk 对应文件位置不一致的情况，举个简单的例子，spandsp 认为在 Asterisk 目录下的 configure.ac 文件的第 XXX 行是 AST_EXT_LIB_SETUP([PGSQL], [PostgreSQL], [postgres])这个内容，而由于 Asterisk 版本的更新，可能这一行已经不是这个内容了，这时就会造成补丁程序无法定位从而无法打补丁的现象，这时可以根据 asterisk.patch 的文件内容，手工对相应文件进行修改，详细的 patch 使用方法，请参考第 5 章《Linux 高级配置和管理》的相关内容。

asterisk.patch 可能包括如下内容：

--- configure.ac.orig 2006-09-03 16:25:31.000000000 +0800

+++ configure.ac 2006-09-03 16:27:19.000000000 +0800

@@ -174,10 +174,11 @@
AST_EXT_LIB_SETUP([PGSQL], [PostgreSQL], [postgres])
AST_EXT_LIB_SETUP([PRI], [ISDN PRI], [pri])
AST_EXT_LIB_SETUP([PWLIB], [PWlib], [pwlib])
AST_EXT_LIB_SETUP([QT], [Qt], [qt])
AST_EXT_LIB_SETUP([RADIUS], [Radius Client], [radius])
+AST_EXT_LIB_SETUP([SPANDSP], [spandsp Library], [spandsp])
AST_EXT_LIB_SETUP([SPEEX], [Speex], [speex])
AST_EXT_LIB_SETUP([SQLITE], [SQLite], [sqlite])
AST_EXT_LIB_SETUP([SUPPSERV], [mISDN Supplemental Services], [suppserv])
AST_EXT_LIB_SETUP([OPENSSL], [OpenSSL], [ssl])
AST_EXT_LIB_SETUP([FREETDS], [FreeTDS], [tds])
@@ -794,10 +795,12 @@

这段 patch 的意思是找到 configure.ac 并在其中的特定位置加上+后面的那一行。在由于程序版本不同步造成无法自动 patch 时，我们就能手工查找到对应文件的对应内容，并依据 patch 文件所描述的进行修改。

在修改完成后，可以运行

CentOS: cd /usr/src/voip/asterisk-1.4.x

CentOS: make menuselect

来确定补丁是否生效。该命令会出现一个文本菜单，进入第一个项目 Applications，可以看到所有的应用（或者我们理解为模块）。如果没有 app_rxfax 和 app_txfax 两个选项，说明我们前面的补丁操作根本没有进行任何改动，那么需要检查是否将 app_rxfax.c 和 app_txfax.c 拷贝到了正确的位置，也可能需要手工进行补丁操作。如果 app_rxfax 或 app_txfax 是 XXX 而非*号，说明补丁打得不对，此时可以尝试如下解决：

编辑 Asterisk 源程序目录下的 build_tools/menuselect-deps，找到

SPANDSP=@....

改为

SPANDSP=1

编辑 Asterisk 源程序目录下的 makeopts 文件

找到

SPANDSP_INCLUDE=

在这一行前面加上#，以注释掉改行

同样是 makeopts 文件找到

SPANDSP_LIB=

改为

SPANDSP_LIB=-lspandsp

然后运行 make menuselect 以确保 app_txfax 和 app_rxfax 都是以*号开头而非 XXX。

如果我们想要加入计费模块，例如 astpp，那么需要将 astpp 中的 asterisk_apps 下的 astpp 复制到 asterisk 目录下的 apps 目录中。

终于，我们可以按照正常的操作流程配置和安装 Asterisk 了：

CentOS: cd /usr/src/voip/asterisk-1.x.x

CentOS: ./configure ;在早期的 Asterisk 1.2.x 中可能不需要 configure 这一步。

CentOS: make

CentOS: make install

CentOS: make samples

CentOS: make config

make 会把所有源程序编译为二进制可执行程序或者动态加载文件，make install 则拷贝相关文件到相关目录，asterisk 用到的目录包括：

/etc/asterisk/ ;Asterisk 所有配置文件都放在这里。

/usr/sbin/ ;Asterisk 可执行文件的放置位置。

/var/lib/asterisk/ ;Asterisk 运行过程中经常访问和修改的文件会放到这里，比如声音文件、数据等

/usr/lib/asterisk/modules/ ;Asterisk 所有的应用、模块都放在这里。

make samples 则拷贝默认的配置文件到配置文件目录，也就是/etc/asterisk/。

make config 则会把控制 asterisk 启动和停止的脚本复制到系统启动脚本目录，通常是/etc/init.d/，通常也叫做 asterisk，这样，我们可以方便的管理 asterisk 启动和停止，执行

CentOS: /etc/init.d/asterisk

可以得到我们可用的控制命令，如启动、停止、重启和状态等，在 RedHat/CentOS 系统中，在 make config 后可以使用 ntsysv 这个程序来决定 asterisk 是否随系统启动而自动启动。

小技巧：Debian 上通常不会安装 ntsysv 这个程序，但我们有另一个替代品 sysv-rc-config，它用法基本和 CentOS 上的 ntsysv 一样，可以通过控制/etc/init.d/下的脚本与/etc/rc?.d/下的连接，来起到配置自启动程序的功能

接下来，是 asterisk-addons 包的安装，这个包相对简单：

CentOS: cd /usr/src/voip/asterisk-addons-1.x.x

CentOS: make;make install;make samples

早期版本的 asterisk-addons 没有 make samples 这个功能，所以用户需要手工拷贝其源文件目录下的配置文件到/etc/asterisk/目录下，同时，早期的 asterisk-addons 可能需要在 make 之前执行以下操作：

CentOS: perl -p -i.bak -e 's/CFLAGS.*D_GNU_SOURCE/CFLAGS+=-D_GNU_SOURCE\nCFLAGS+=-DMYSQL_LOGUNIQUEID/' Makefile

小技巧：请准许我再一次鼓吹系统管理员学习编程的好处，上面这个 perl 语句是什么意思？它的操作过程是：首先将 Makefile 备份为 Makefile.bak（通过-i.bak 选项），然后将 Makefile 文件中出现的第一个 CFLAGS.*D_GNU_SOURCE 替换为

CFLAGS+=-D_GNU_SOURCE

CFLAGS+=-DMYSQL_LOGUNIQUEID

言归正传，接下来我们进行 make 操作：

CentOS: make;make install

注意，1.4.x 版本的 asterisk-addons 不需要做这种 perl 的手工修改。

32.5 Asterisk 硬件及其相关配置

如果我们使用接入传统 PSTN 电话网络的模拟转接卡，那么首先需要配置这个硬件卡。市场上常见的有两种型号，都是 Asterisk 厂商 Digium 公司的，一种是可以接四个模块、有四个通道的 TMD400P 型号，一种是只有一个模块、一个通道的 X100P 型号。不过国内流行的大部分都是这两种型号的兼容卡，原版卡用得不多。

32.5.1 Asterisk 硬件板卡信令

Asterisk 的信令分为两种：一种驱动话机的信令，叫做 FXO 信令，一种是来自线路的信令，叫做 FXS 信令，例如，当我们家里接上了一部固定电话的时候，来自外面的信号使用的是 FXS 信令。与之对应的硬件板卡模块也有两种，一种叫 FXO 模块，专门处理来自线路的信令，也就是 FXO 模块处理 FXS 信令，FXO 模块在通信中的位置相当于我们家里的电话或调制解调器。另一种当然叫 FXS 模块，专门处理电话信令，也就是说，FXS 模块在通信中的位置相当于给用户提供信号的电信局。

如何区分 FXS 模块和 FXO 模块的工作方式呢？其实是看模块是否产生拨号音。当模块提供拨号音的时候，是 FXS 模块，处理 FXO 信令，当模块接受并处理拨号音的时候，是 FXO 模块，处理 FXS 信令。最简单的区分方式，是 FXO 模块接的是电信局外线，而 FXS 模块直接接内部的分机。

FXS、FXO 信令和模块，是最容易混淆和糊涂的概念，初学者一定要仔细想清楚，否则很容易在配置时陷入困惑当中。

32.5.2 Asterisk 信令协议

对于 FXS 信令（即 FXO 模块），Asterisk 还有几种不同的协议方式，Asterisk 常用的信令协议有三种：Loop start，简称 ls，Ground start，简称 gs，和 Kewlstart，简称 ks。

这几种协议的区别主要在于如何请求远程局方的拨号音，ls 使用 short 信号来请求远程的拨号音，而 Ground start 使用更加老式的请求方法，ks 则和 ls 类似，但能够更好的检测远程信号，所以 Asterisk 推荐使用 ks 协议。

我们国内一般也使用 ks 和 ls 协议。

32.5.3 Asterisk 板卡配置实例

假设我们使用一个 Asterisk 系统作为机构的程控电话交换机，该系统的外部接入电信的 PSTN 网络，而内部使用纯 VoIP 系统。那么我们仅需要一块接收电信信号的板卡来接受和处理拨号音，所以这块板卡是 FXO 卡，接收 FXS 信号。

假设我们使用的这块板卡是单通道的 X100P FXO 兼容卡，将这块板卡插入 Asterisk 服务器的 PCI 插槽后，启动电脑，开始配置该板卡。

前面我们说过，Asterisk 通过 zaptel 对硬件板卡进行驱动和管理。所以我们首先要使用 zaptel 正确配置该板卡，修改 zaptel 的配置文件/etc/zaptel.conf，保证文件中有这么一行而且没有被注释起来：

fxsks=1

这行配置包括两个部分，第一部分是前三个字母，告诉 zaptel 这个板卡接收的信令，这里是 FXS 信令，第二部分是后两个字母，告诉 FXS 信令的协议类型，这里是 KS 协议类型，即 Kewlstart 。在完成硬件信令和协议的配置后，我们可以使用 ztcfg 来看看配置是否正常：

CentOS: ztcfg –vvv

如果出现类似以下的画面，则说明硬件配置和信令处理方式都是正常的：

```
[root@ippbx ~]# ztcfg -vvv

Zaptel Version: 1.4.1
Echo Canceller: MG2
Configuration
=====================


Channel map:

Channel 01: FXS Kewlstart (Default) (Slaves: 01)

1 channels configured.

[root@ippbx ~]#
```

图 32-2 单通道 FXO 卡配置

图中显示的是该卡的信令和协议，而不是模块的类型，也就是说，这块 FXO 模块板卡处理的是 FXS 信令，使用 KS 协议。

否则可能是硬件错误、驱动错误或者配置错误，我们需要一步一步排查，首先用 lspci 命令确认系统识别到了我们的硬件：

CentOS:lspci

00:0d.0 Communication controller: Tiger Jet Network Inc. Tiger3XX Modem/ISDN interface

如果没有这种 Communication controller，那肯定是硬件错误，系统根本没发现插入了通讯卡。

如果硬件是正常的，那么我们可能需要检查必须的 zaptel 模块是否加载，用 lsmod 命令：

CentOS:lsmod|grep zaptel

zaptel 196132 4 zttranscode,wcfxo

这说明说明模块加载正常，使用的是 wcfxo 驱动。否则我们需要手工加载对应的模块：

CentOS:modprobe zaptel

CentOS: modprobe wcfxo

然后再尝试用 ztcfg –vvv 进行配置。对于一般的 X100P FXO 兼容卡，也就是单通道的硬件卡，使用 wcfxo 驱动，对于一般的 TDM400P 兼容卡，也就是 4 通道硬件卡，需要使用 wctdm 驱动而不是 wcfxo 驱动。

如果上述都正常而 ztcfg 仍然报错，那么可能是我们配置的通道类型不对，可以尝试修改配置文件/etc/zaptel.conf，将 fxsks=1 改为 fxoks=1，或执行相反的操作，然后用 ztcfg –vvv 检测是否正常。

32.6 Asterisk 软件基本配置

在正确的安装好 Asterisk 后，我们可以开始配置这一伟大的软件。如果我们不想使用标准的文本文件配置管理，可以使用数据源管理 Asterisk 用户。

32.6.1 可选的数据源配置

如果使用 unixODBC 作为数据源，那么需要编辑/etc/asterisk/res_odbc.conf 和 unixODBC 的 odbc.ini 这两个文件。

如果使用 asterisk-addons 的 MySQL 作为数据源，那么需要编辑/etc/asterisk/res_mysql.conf 这个文件。

我们首先看看第一种情况。这种情况是 Asterisk -> ODBC -> MySQL 或其他数据库。res_odbc.conf 可以确定 Asterisk 使用什么数据源名称（DSN，Data Source Name），这个 DSN 指向了 odbc.ini 中的一个配置，该配置确定 unixODBC 应该使用怎样的数据库驱动，并连接到哪个数据库服务器。最终，Asterisk 通过 res_odbc.conf 的配置访问 unixODBC 从而得到该数据源连接。

例如，我们在 res_odbc.conf 中指定 Asterisk 使用一个叫做 demo 的数据源：

CentOS:cat /etc/asterisk/res_odbc.conf

[demo]

enabled => yes

dsn =>mysql1

username => asterisk

password => passw0rd

pre-connect => yes

该数据源的 DSN 名字叫 mysql1，也就是 unixODBC 配置文件 odbc.ini 中，应当有一个对应的叫 mysql1 的配置。enable 告诉系统启用这个数据源，username 和 password 则是连接到这个数据源时使用的用户名、密码，pre-connect 设置为 yes，是告诉系统一启动就连接到该数据源，而不是在有请求时再

连接。

在 unixODBC 的配置文件 odbc.ini 中（这个文件依据 unixODBC 的安装路径而定，通常可以在/usr/local/etc/目录下找到），也要有一个名叫 mysql1 的数据源配置：

CentOS:cat /usr/local/etc/odbc.ini

[mysql1]

Driver=/usr/lib/mysql/libmyodbc3.so

Setup=/usr/lib/mysql/libmyodbc3S.so

Server =192.168.0.1

database=asterisk

Port =

Socket =

Option = 3

Stmt =

Driver 和 Setup 分别指向 unixODBC 对应数据库的驱动和配置库，这里是/usr/lib/mysql/libmyodbc3.so 和/usr/lib/mysql/libmyodbc3S.so，这两个文件的位置和名称依赖于系统安装的 MySQL ODBC 驱动的位置和版本，因此要根据实际情况进行设置（见前面的 MySQL ODBC 连接器设置）。

Server 说明了数据库服务器的 IP 地址或域名，database 则说明默认连接到哪个数据库，这样，这个配置和前面的 res_odbc.conf 配置合起来，就能完整的确定数据库服务器地址、数据库名称、用户名和密码，有了这些参数，系统就能访问合法的数据来源了。

第二种情况则更加简单，因为这是 Asterisk 直接连接到数据库，不需要通过 unixODBC 中转，所以只要一个配置文件就行。

32.6.2 Asterisk 基本配置

在硬件驱动和配置正常的情况下，我们可以尝试启动 Asterisk：

CentOS: asterisk –vvv

如果一切正常，在一系列加载模块的提示后，会出现 Asterisk Ready 的提示，这表示 Asterisk 已经正常工作了。

如果启动 Asterisk 时提示加载某个模块或应用失败，那么需要根据实际情况进行判断操作，如果这个模块是我们不需要的，可以在/etc/asterisk/modules.conf 中使用 noload => xxx.so 语法禁止该模块加载。

如果启动 Asterisk 时提示 app_rxfax.so 由于 libspandsp.so.0 而无法加载，那么可能是这个文件不在系统库路径中，可以考虑找到这个文件并复制到/usr/lib 中。启动 asterisk 使用 module show 保证 app_txfax 和 app_rxfax 是正常的。

我们来简单的了解一下 Asterisk 的运转流程。Asterisk 的主要配置文件是 extensions.conf，它定义了在不同环境下，针对某个按键或指令时，Asterisk 应当采取怎样的行动。我们以一个例子来说明，

假设在内部电话环境下，希望所有用户拨 0 到前台总机，那么相对于 Asterisk 的语言就是：

在[内部]环境下，拨打号码 0 -> 呼叫总机 555（假设总机是 555）

进一步翻译就是：

[internal] exten => 0, Dial 555

是否符合你的思维逻辑？其实，这条语句已经非常接近真实的 Asterisk 逻辑语言了，但还要更加细致。假设我们都使用 SIP 协议，那么首先要定义 SIP 分机 555（定义分机下面会说明），这个分机是连接到前台的，然后 Asterisk 是按照优先级的概念来分配流程的，就是当用户拨打一个号码时，Asterisk 会依照优先级进行依次操作，比如拨打 0，先执行优先级 1 的操作，然后执行优先级 2 的操作，然后是 3、4、5 依此类推。那么要实现上述功能，真实的 Asterisk 配置就是

[internal]

exten => 0, 1,Dial(SIP/555)

正如我们前面描述的那样，在[internal]内部环境下，拨打 exten 0，第一步操作（优先级为 1）就是替用户呼叫 SIP 555 这个用户。这个 internal 在 Asterisk 中称为环境 context，用户属于不同的环境，在不同的环境下，可以执行不同的判断和操作，同时可以作为识别用户权限的方式，比如如果 SIP 555 用户不属于 internal 这个环境（也就是说他不是内部用户），那么这条语句将执行失败。exten 则可以看作是用户的操作，Dial 是 Asterisk 执行的功能，在 Asterisk 中称为 application 应用。整个语句的模型是：

[环境]

exten => 用户操作, 优先级，Asterisk 执行的应用

是不是足够简单明了？理解了 Asterisk 的控制和逻辑，我们就很容易给 Asterisk 安排工作了。

32.7 我的第一台电话交换机——基本的分机、出入路由和 DialPlan

在确保 Asterisk 及硬件都正常后，我们终于开始配置我们自己的第一台电话交换机了，真是令人激动啊。一般来讲，我们使用基于 VoIP 协议的 IP 电话作为内部通讯节点，那么对于 Asterisk 来讲，通常就是 SIP 协议，如果以文件作为配置源的话，这个协议的配置在文件 sip.conf 中，如果使用数据库作为配置源的话，通常是定义为 sip_user 的数据表。也就是说，我们所有的电话号码或者叫分机，都在这个配置中实现。

sip.conf 的格式是

[用户名]

类型＝

密码＝

来自主机＝
默认环境=
...
例如，一个 555 的 SIP 内部分机定义为：
[555]
type = friend
secret = mypassword
host = dynamic
context = internal

意思是一个名叫 555 的 SIP 用户（当然不是一定要用数字，字母也是可以的比如 paul），密码是 mypassword，可以来自动态 IP 的主机地址，也就是不管来自哪的用户，只要有正确的用户名和密码都能注册成功，默认的环境是内部 internal（这个 internal 只是一个定义，可以是任何你喜欢的单词，只要保证 Asterisk 在流程控制中的 context 与之一致就可以了）。

照此操作，我们就能添加所有的 SIP 用户，也就是分机号码，在完成后重新加载 Asterisk，我们的内部员工就可以用 SIP 电话（包括硬件和软件电话）以自己的用户名/密码注册了。

现在我们可以配置一个简单的 DialPlan，所谓 DialPlan，其实就是 Asterisk 的工作逻辑指令，Asterisk 通过读取这些指令，来确认自己在碰到什么情况应该怎么处理。这些 DialPlan 通常放在 extensions.conf 中。假设我们有三个用户，一个总机 555，一个用户 paul，一个用户 315。那么 sip.conf 应该类似于这样的内容：

[555]
type = friend
secret = mypassword
host = dynamic
context = internal

[paul]
type = friend
secret = paulpw
host = dynamic
context = internal

[315]
type = friend
secret = pw315
host = dynamic
context = internal

这表示这三个用户都是内部用户，使用动态主机，那么在 extensions.conf 中，我们可以为这三个用户配置 DialPlan 了：

[internal]

exten => 555, 1, Dial(SIP/555)

exten => 501, 1, Dial(SIP/paul)

exten => 315, 1, Dial(SIP/315)

这个 DialPlan 非常简单，内部注册用户（即定义为 internal context 的注册用户）拨 555 会转给总机 555，拨 501 会打给用户 paul，拨 315 则打给用户 315，注意这里不像传统 PSTN 电话网络那样，必须给用户一个确定的终端，VoIP 不需要固定的主机，它通过帐号密码进行认证和连接，也就是说，内部用户可以在任何终端上注册，比如电脑、支持 SIP 的手机等，只要注册的是自己的帐号，当有人访问时都能直接呼叫到自己，这更类似于手机的网络，总机 555 可以在前台，也可以在公司的任何地方，甚至可以不在公司，如果公司无线网络覆盖足够远的话，总机可能在楼下的咖啡厅:-)。

到这里，我们的内部电话系统实际上已经完成，用户们可以相互通话了。但是，只能内部通话没有任何意义，我们需要和世界沟通（好像是广告？），那么，如何进入外面的世界同时让外部的朋友访问我们呢？首先是确定一个通道（Trunk），这个通道可能是一个传统 PSTN 节点，也可能是其他的 Asterisk 网络。我们假定是通过 PSTN 的通道连接到外部的（这可能更符合我们的实际情况）。

如果使用 PSTN 网络，那么我们假设你已经配置好了你的硬件和电话线，下面，让 Asterisk 使用这个连接，硬件连接在/etc/asterisk/zapata.conf 中配置：

[channels]

usercallerid=yes

context=internal

channel=>1

这段配置的意思是，第一个硬件线路 channel1 的默认环境是 internal（就是我们一直使用的那个 context），usercallerid=yes 用来设置来电显示。

配置完成后，我们就可以定义如何让用户拨打外线了，比如下面这个 extensions.conf 的例子：

[internal]

exten => 555, 1, Dial(SIP/555)

exten => 501, 1, Dial(SIP/paul)

exten => 315, 1, Dial(SIP/315)

exten => 9, 1, Dial(Zap/1)

后面新增的这个规则是什么意思呢？内部用户按 9，Asterisk 就会去执行 Dial(Zap/1)，这个 Zap/1 就是我们前面在 zapata.conf 中配置的 PSTN 通道的第一个 channel，说到这里，意思应该很明显了：内部用户按 9 拨外线，确切的说，是拨传统电话，从第一条线路出去（如果我们有多条线路的话）。要确定配置是否正确，最直接的办法，就是注册内部用户，然后拨 9，看是否能听到拨号音，并是否能和某个外部用户通话。

现在，我们拥有了一个可以实用的高级 PBX 了，它完全能取代传统 PBX，为公司提供高质量的语音通话，并且，相对于各种厂商自成体系的复杂的配置指令来说，它非常易于配置和使用。接下来，我们将对 Asterisk 的逻辑和高级配置做进一步的详细说明。

32.8 与其他 VoIP 网络互联

仅仅在本地实现软电话交换功能，没有太大意义，这就好像只有局域网，却不能连接到 Internet 上一样，因此，要让我们的电话交换机真正有实用价值，我们必须让它跟外部系统连接。

32.8.1 与其他 Asterisk 互联

随着 Asterisk 的兴起，相信我们会有很多机会与其他公司或者其他地区分公司的 VoIP 网络互联，这样，大家可以更加便利的交换信息，而且完全免费。假设我们是这样一家公司，有一个放在 Internet 上的 Asterisk 服务器，各地分支机构的 PBX 也都是 Asterisk，那么，让所有的 Asterisk 连接成为一个网络，无疑是非常有意义的。这种情况，我们应该如何配置和管理呢？

首先，我们了解一下 IAX 协议，IAX 是 Inter Asterisk eXchange 的缩写，是专为 Asterisk 互联开发的协议，目前版本是 2，如果我们提到 IAX，默认就是 IAX2，而非 IAX1。通过 IAX 协议，两台 Asterisk 服务器可以方便的认证、互联和通讯。IAX 的配置在/etc/asterisk/iax.conf。

我们定义一下假设的环境，服务器 Server A 是直接连接到 Internet 上具备固定合法 IP 地址的 Asterisk 服务器，域名是 a.company.com，服务器 Server B 是某个分支机构在当地的 Office 中，使用动态 IP 地址的一台 Asterisk 服务器。

IAX 连接的流程并不复杂，首先，A 上定义个 IAX 用户（类似于 SIP 用户），这个用户给 B，让它通过这个用户连接到自己，所以，这个用户我们定义为 serverb:

```
ServerA: cat /etc/asterisk/iax.conf
[serverb]
type=friend
host=dynamic
secret=asteriskpw
context=default
trunk=yes
permit=0.0.0.0/0.0.0.0
qualify=yes
username = servera
```

一个名为 serverb 的 iax 用户，由于 B 使用动态 IP，所以我们将 type 设置为 friend，将 host 设置为 dynamic，并将 permit 设置为 0.0.0.0/0.0.0.0，以便 B 能够以任意 IP 地址连接登陆，其登陆密码是 asteriskpw，username 这个选项是用于第二步当 A 连接到 B 时用的，context=default 表示，凡是通过这个 IAX 连接进来的用户都使用 default 这个 context。

同时，我们在 Server B 上也定义一个 IAX 配置：

```
ServerB: cat /etc/asterisk/iax.conf
```

[general]
allow=ulaw
register => serverb:asteriskpw@a.company.com

[servera]
host= a.company.com
type=friend
username=servera
secret=asteriskpw
qualify=yes
disallow=all
allow=all
context=incoming

注意，与 Server A 配置不同的是，我们在 Server B 的 iax.conf 的 general 小节中，增加了一个 register，它的作用是 Server B 的 Asterisk 启动后将以 IAX 用户 serverb 的身份注册到 Server A，这样可以让 Server A 知道 Server B 的连接源地址，从而能进行双方通信，在不能使用固定 IP 或者合法 IP 的时候，这个功能就非常有用。

然后是一个叫做 servera 的 IAX 用户（或连接），它与 Server A 上的配置大同小异，需要注意的是，context 必须设置为自己的 context，也就是 Server A 上的 context 是 Server A 默认的 context default，Server B 上的 context 是 Server B 默认的 context internal。同时要注意，两个 secret 最好一致。

IAX 的配置就这么多，在两边都重新加载了 Asterisk 后，我们可以登陆到 Server B 的 Asterisk 终端（也就是使用动态 IP、主动注册的那台服务器），使用 iax2 show registry 命令查看是否正确的注册到了 Server A：

ServerB: asterisk –r
CentOS*CLI>iax2 show registry
Host dnsmgr Username Perceived Refresh State
a.company.com:4569 N serverb 202.211.35.123:37304 60 **Registered**

如果能出现上面的提示，状态是 Registered，说明成功的注册到了 Server A。

接下来，我们要做的是为两台 Asterisk 配置 DialPlan，以便让他们互相通讯，我们考虑最简单的情况，也就是只有这两个网络互联，不考虑有三个以上网络连接的情况。我们假定让两边网络中的用户拨号时前面加 8，则进入对方的网络，否则仍然走原有的流程，那么，在 Server A 的 extensions.conf 中可以这样定义：

ServerA: cat /etc/asterisk/extensions.conf
[default]
exten => 555, 1, Dial(SIP/555)
exten => 501, 1, Dial(SIP/paul)
exten => 315, 1, Dial(SIP/315)
exten => 9, 1, Dial(Zap/1)
exten => _8., 1, Dial(IAX/officeb/${EXTEN}:1)

这个规则可能是我们目前接触到最复杂的一个拨号规则，首先，以_开头表示这不是一个普通的电话号码，而是一个模式匹配（也就是正则表达式，相信很多 UNIX/Linux 用户对它都是既爱又恨:-），后面的点号.表示一个或多个号码，最少要有一个号码。

那么这整个表达式_8.的意思就是任何以 8 开头的、最少两位的一串数字，具体到我们这个应用中，就是用户拨打以 8 开头的号码，Asterisk 都认为符合这条规则，从而执行后面的这个 Application：

Dial(IAX/officeb/${EXTEN}:1)

Dial 我们已经知道了，是拨打的意思，那么拨打到哪呢？到 IAX/officeb/${EXTEN}:1。

后面的 IAX 表示走的是 IAX 协议，那么应当去 iax.conf 中查找 IAX 连接，然后去往其中配置的 officeb 这个 IAX 连接，连接成功后，拨打该连接的一个号码，这个号码是${EXTEN}:1，那这又是什么意思呢？EXTEN 是 Asterisk 自动生成的变量，它称之为通道变量，总之，在这里，EXTEN 就是用户拨打的号码，如果用户拨 8999，那么 EXTEN 就等于 8999，而:1 则表示去掉这个变量开头的 1 位，也就是说，如果 EXTEN 等于 8999，那么${EXTEN}:1 就等于 999。整个 DialPlan 的效果就是当用户拨打 8999 的时候，Asterisk 会为用户接通 Server B 上的分机 999。

ServerB: cat /etc/asterisk/extensions.conf

```
[internal]
exten => 999, 1, Dial(SIP/999)
exten => _8., 1, Dial(IAX/servera/${EXTEN}:1)
...
```

这是 Server B 上默认 context internal 的配置，同样是用户如果以 8 开头的拨打，则呼叫 IAX 通道的 Server A 上的分机，在 Server B 上的效果则是，注册到 Server B 上的所有（internal）用户拨打 8555，则会接通 Server A 上的 555 用户。

当然，这里我们假设双方的网络都是一家公司专有的，这样不存在保密和私有的问题，这种结构其实破坏了 internal 这个概念，后面我们会详细描述，如何实施公共网络间的互联。

32.8.2 与公共 VoIP 网络相连

到目前为止，国内并没有公共的 VoIP 网络，不过，有家国外的网站提供免费的 VoIP 语音通话服务，同时也提供 IAX 协议的连接服务。这家网站叫：FreeWorld Dialup（www.freeworlddialup.com），简称 FWD。使用这家网站 VoIP 服务的用户大概有

用户可以免费在 FWD 上注册一个帐号，这个帐号可以作为用户普通 VoIP 电话号码帐号，也可以作为 IAX 帐号，让自己公司或者家庭的 Asterisk 通过这个帐号连接到自由的 FWD VoIP 网络。和前面配置 Asterisk 互联类似，我们在公司的 Asterisk 服务器上通过配置 iax.conf 来连接到 FWD 网络：

CentOS: cat /etc/asterisk/iax.conf

```
[iaxfwd]
type=user
context=fromiaxfwd
auth=rsa
```

inkeys=freeworlddialup

32.9 Asterisk DialPlan 拨号方案初探

通过前面章节的讨论，我们已经看到了，Asterisk 的灵活性，这种灵活性其实大部分来自 Asterisk 的 DialPlan，通过 DialPlan，我们可以灵活控制流程和操作，从而完成很多高级功能，这一节，我们就来重点讨论 DialPlan 的各种语法和应用。

前面我们接触了一些 DialPlan 的基本用法，知道 DialPlan 的格式是

[context]

exten => Name, Prioprity, Application

在符合某个 context 环境下，拨打一个 Name 名字或号码，将按照 Prioprity 优先级执行特定的 Application 应用。我们把这一条 DialPlan 称为 Asterisk 的一条规则。

比如前面经常用到的 DialPlan：

exten => _8., 1, Dial(IAX/servera/${EXTEN}:1)

就是表示拨打以 8 开头的号码，Asterisk 将首先执行 Dial 这个应用。下面我们分别讨论每种要素的作用和用法。

- Context

  我们首先看看 context 环境，环境实际上可以看作是一个权限组。用户定义到某个 context 中（这个用户可能包括 SIP、IAX 等各种用户），则该用户有权限执行该 context 中的操作，或者被该 context 中的操作访问。

  比如，一个 internal context 中的用户定义为拨打 VoIP 电话，但另一个 advance context 中的用户定义为可拨打 PSTN 电话，这样就将不同的用户权限区分开来。

- Name

  然后我们看看 Name，Name 是用户拨打的号码或名字，可以是字母或数字，这里，Name 可以指定为固定的号码，如 555，也可以指定为某个模式，如_8.，其他模式元素还包括：

  X：所有 0-9 的数字

  Z：所有 1-9 的数字

  N：所有 2-9 的数字

  [123]：1 或者 2 或者 3

  .：前面讨论过了，代表一个或多个任意数字或字母

  那么，这样一个模式：

  _1[35]XXXXXXXXX

  它代表什么呢？没错，它表示这是一个本地手机号码，以 1 开头，第二位是 3 或者 5，然后是 9 位任意的数字。

- Prioprity

  Prioprity 表示符合这一规则的方案，Asterisk 能够执行很多操作，按照 Prioprity 依次执行，例如：

exten => 555, 1, Answser()

exten => 555, 2, Playback(some voice)

exten => 555, 3, Hangup()

这个规则的意思是，用户拨打 555，系统首先应答，然后播放一个语音文件，接着挂断，通过优先级，我们就能控制 Asterisk 针对一个操作的流程。

- Application

Application 定义了针对每种状况所作的操作，通过这些操作，Asterisk 才能实现各种不同的功能，同时，这些 application 是可扩展的（也就是用 show modules 显示的那些以 app_开头的模块），也就是说，实际上，Asterisk 的功能是没有限制的，只要你能写出一个合适的扩展，就能让 Asterisk 做你想做的事情。application 类似于函数的调用：应用名（参数 1，参数 2……），例如我们常用的 Dial 应用，它包括四个参数：

参数 1 是目的地，Asterisk 呼叫的目的地，格式是通道类型/名称/扩展号码。

参数 2 是超时设置，也就是当这个呼叫一直处于等待状态时，多长时间放弃，单位是秒。

参数 3 是控制呼叫时的行为，最常用的是 r，设置这个参数，将使主叫方在等待接听时听到铃声，这个通常不用，因为振铃基本上都是自动的。

参数 4 是传递一个 URL，这个仅限于被叫终端支持显示 URL 时，它将显示这个 URL 到终端上。这个用得最少，因为很少终端会支持传入 URL。

了解了这些基本格式，我们应当对 extensions.conf 中的规则不再感到陌生了，接下来，我们可以开始学习一些常见的 DialPlan 规则。

一个普通 PBX，通常处理的情况不外乎呼出、呼入、电话会议等。

对于呼出，我们之前已经接触过了，这里我们再来进行一些详细的讨论。想想我们平时使用公司程控电话的情形，拨外线不外乎两种，一种是前面加个数字比如 9，后面直接跟要呼叫的号码，一种是拨某个数字，然后等待外线，通了再拨要呼叫的号码。

这对于 Asterisk 来说，也非常简单，第一种按 9 接着按呼叫号码的方式，可以用这种规则解决：

exten => _9., 1, Dial(Zap/1/${EXTEN}:1)

凡是以 9 开头后面跟一串数字的统统连接到 PSTN 网络，并把 9 删除然后替用户拨打他后面的号码。当然我们还可以做得更好，比如判断用户号码长度，只有当用户号码位数大于或等于 5 位的时候，我们才认为用户在拨打外线，否则认为他拨打的是内部分机：

exten => _XXXX.,1, Dial(Zap/1/${EXTEN}:1)

exten => _XXX, 1, Dial(SIP/${EXTEN})

XXXX.的意思就是至少要有 5 位数字，或者更多，这种情况下，Asterisk 会替用户呼叫外部号码。

对于第二种呼出方式，即按 9 后等待市话提示音，则更加简单：

exten => 9, 1, Dial(Zap/1)

只要用户是按的 9，就直接替他接通 PSTN 网络，剩下的，让他自个儿去处理吧。当然，Dial 应用并不局限于拨交市话 PSTN 网络，这个规则同样适合于拨打某个外联的分支机构的 Asterisk 系统，例如：

exten => 8, 1, Dial(IAX2/office1)

它的意思就是，凡是按 8，直接接通 office1 的 Asterisk 系统，让那边的 Asterisk 处理用户后面的输入吧。

上面这两个规则的问题是，分机不能是 8 或 9 开头的，否则 Asterisk 就会糊涂了，不过，糊涂的其实是我们管理员，Asterisk 并不知道我们的想法，所以，下面这个规则就能避免大家的糊涂：

exten => 9XX, 1, Dial(SIP/${EXTEN})

exten => 9,1, Dial(Zap/1)

当然，这对于手机或软件电话来说，并不是问题，因为他们都是输入完所有的号码然后选择拨出，800、80、8 都可以很容易区分，但一些固定电话则不然，它是摁一个号码就传送一个，交换机必须一个一个的判断怎么处理。

而对于呼入，我们首先要做的是应答 Answer()，然后再依据不同情况实施不同的方案。例如，一个公司总机在接到呼入的时候，通常会播放一段语音，提示用户输入分机号码，查号拨零等，这个规则如下：

[incoming]

exten => s, 1, Answer()

exten => s, n, Background(welcome)

exten => 0, 1, Dial(SIP/501)

exten => 555, Dial(SIP/555)

exten => 556, Dial(SIP/556)

... ...

这里，我们会接触到两个新东西，一个是命名为 s 而非任何号码或用户名的特殊扩展，这个 s 扩展代表用户没有任何扩展，则匹配这个策略，举个最简单的例子，用户打进一个总机的时候，交换机不会接收到任何扩展，那么这个 s 告诉交换机，在没有收到任何扩展时，默认到达这个地方。而第二步的 n 代表 next，也就是同一个扩展的下一步，这里 s 扩展第一步是 Answer，那么第二步就是播放一个 welcome 的语音，但我们可以不用写 2，直接用 n 表示，Asterisk 会自动用上一步的序号+1 来填充这个 n，那么它就自动变成了 2，当然我们还可以继续写 exten => s, n, ...，这里 n 就会自动取值为 3。

通常情况下，用户听到 welcome 的语音后，都会选择拨号或者挂机，当他拨号时，Asterisk 就会开始判断用户输入的号码，并接到对应的分机/用户。但如果用户不挂机一直等着怎么办呢？通常有三种方式处理，一是在等待一定时间用户没有输入直接转到主机，二是等待一定时间后直接挂机，三是重复播放欢迎语音。

第一种情况，直接转到总机，可以在所有流程完毕后加上

exten => s, n, Dial(SIP/501)

第二种情况，直接挂机，在所有流程完毕后加上

exten => s, n, Hangup()

第三种，我们直接跳到第一步：

exten => s, n, Goto(incoming,s,1)

当然，还有更友好的方法。除了前面我们提到的 s（start）这个特殊的扩展外，另外还有两个有用的特殊扩展是 i（invalid）和 t（timeout）。不难立即，这个两个特殊扩展用于处理无效输入（invalid）和超时（timeout）。比如，公司 Asterisk 只允许外部用户呼叫 5 开头的号码，那么，在 incoming context 中，只会定义一些 5XX 的扩展，那如果有人打进来后拨打了其他分机怎么办呢？当然我们可以不理会他，但最好的办法是告诉他没有这个用户，所以，在整个 incoming 环境中，最后可以加上：

exten => i, 1, Background(invalid_number)

用于在用户输入了非法号码时进行一个友好的提示。而如果用户长时间不做任何操作，我们也可以通过 t 扩展提示他赶紧按键，否则将挂机：

exten => t, 1, Background(timeout)

32.10更多有用的 DialPlan

好了，通过前面的配置，相信大家已经开始对 Asterisk 产生了好感（或者厌恶？）。不管怎样，我们还可以通过这一节来展示 Asterisk 一些更有用、更有趣的 DialPlan 规则。

比如说，我们可以来制作一个类似于中国移动 10086 客服系统式的互动式语音应答系统 IVR。这个系统其实逻辑非常简单，就像最简单的程序 if...else...那样，根据用户的按键，系统做出相应的反映，或播放一段录音，或着转接到人工台。

首先，当用户打进来后，开始播放提示音，在致过欢迎词后，告诉用户一个语音菜单：按 1 听说明，按 2 听音乐，按 3 听笑话，按 4 转接人工，如果用户长时间不按键，则自动挂断，如果用户摁错了，比如没有进行任何定义的键，那么将提示用户错误的输入，并重新播放主菜单。

[incoming]

exten => s, 1, Background(welcome)

exten => s, n, Background(main-menu)

exten => 1, 1, Background(description)

exten => 2, 1, Background(music)

exten => 3, 1, Background(joke)

exten => 4, 1, Dial(SIP/101)

exten => i, 1, Background(error_num)

exten => i, 2, Goto(incoming,s,1)

exten => t, 1, Background(timeout)

exten => t, 2, Hungup()

这就是整个 IVR 的流程。当用户拨入这个号码，并且没有任何扩展时，他将进入 s（tart）规则，这个规则的第一步是播放一个欢迎文件，播放完后，接下来 n（ext）播放主菜单说明，如果等到整个语音播放完毕，用户还没有任何输入的话，就会进入 t（imeout）规则，这个规则的第一步，播放一个提示用户系统将挂机的语音，接着就主动挂断电话。如果这整个过程用户有任何输入，则不会进入这个规则。而当用户按下 1－3 任何一个键，则进入对应的播放语音应用，之所以使用 Background 这个应用，是因为 Background 在播放语音的同时，仍然会监视用户的输入，一旦用户有输入，就会中断当前输入而跳入对应的规则。当用户按下 4 时，会呼叫人工座席，直接将用户转接给某个分机，让用户与服务人员直接沟通。当用户按下任何其他未定义的键时，则进入 i（nvalid）规则，这个规则第一步播放一个错误提示，提示用户没有这个按键，然后使用 Goto 应用重新开始播放欢迎词和主菜单。

通常来讲，一个机构不会只有一个电话号码，也就是说，很可能有多个电话信道接入到 Asterisk 系统上。假设有四条外线也就是四个信道接入，当内部用户拨打外线的时候，如何确认哪个外线是可

用的呢？这里就要用到更多的 Asterisk 命令了，使用 Asterisk 的状态、字符串比较和条件跳转等高级功能，我们可以很容易实现这种高级智能化管理。

假设我们有四个通道，分别是 Zap/1、Zap/2、Zap/3 和 Zap/4，都已经在 Asterisk 中配置好了，可以正常使用。我们可以按照这个顺序来轮询每个通道的情况使用判断，当第一个通道不可用的时候，就尝试一下个通道，直到所有通道都尝试一遍，最后可以决定是告诉用户因线路繁忙无法拨打，还是又回到第一个通道的测试，下面，我们就来看看如何在 Asterisk 中实施这个规则。

Asterisk 可以像编程语言一样，支持变量、表达式和运算，和 Shell、PHP 一样，使用$标记这是一个变量。变量可以分为全局变量、预定义变量和本地变量。全局变量是我们在[global]段中定义的，整个配置文件可用的变量。而预定义变量是 Asterisk 为方便管理而预先定义的一些变量，本地变量是在配置过程中临时定义的变量。

全局变量类似于 C 语言中的宏，只要定义一次，就可以在所有的流程中使用。例如，在 extensions.conf 中，我们可以首先在[global]中定义一个变量 RECEPTION 为总机的号码：

```
[global]
$RECEPTION=SIP/801
```

变量 RECEPTION 的值是 SIP/801，当我们在后面的流程中想让用户返回到总机时，就可以调用这个变量了，例如，我们前面实现了当用户首次拨入机构的线路时，会播放一段语音提示，告诉用户直接拨分机号，查号请拨零等，如果用到了全局变量，我们可以更加灵活的实现：

```
[incoming]
exten => s, 1, Answer()
exten => s, n, Background(welcome)
exten => 0, 1, Dial($RECEPTION)
exten => 555, Dial(SIP/555)
exten => 556, Dial(SIP/556)
```

我们把固定的总机号码换成了全局变量 RECEPTION。

如果我们在用户直接拨打一个分机后，设定一个时间比如 10 秒，超过这个时间还没有人接听，则提示用户无人接听，是否返回总机，同样可以使用这个全局变量：

```
[incoming]
...
exten => 503, 1, Dial(SIP/555,10)
exten => 503, 2, Background(notice)
exten => 0, 1, Dial($RECEPTION)
```

当用户拨打一个 503 超过 10 秒后，系统会提示用户超时，当用户选择 0 时，Asterisk 会把用户重新转接到总机。

这些都是很普通的应用，但问题是，如果我们有一个庞大而且复杂的拨号方案，里面涉及到很多转接到总机的情况，就会写入很多总机的分机号码，一旦这个号码发生改变，比如，临时转接到另一个地方，那么，可以在[global]中简单的修改 RECEPTION 变量的值，就能方便迅速的实现这种变化了。

Asterisk 可以执行一些基本的运算，例如逻辑运算、算术运算等。逻辑运算包括例如

$test1&&$test2，就是当当 Asterisk 调用 Dial 应用进行拨号的时候，会返回一个状态，我们可以通过这个状态判断当前的情况，并进行相应的操作，这个状态是一个叫做 DIALSTATUS 的内建变量，它包括以下几种情况：

ANSWER: 呼叫到达，被叫方正常应答，也就是双方开始通话了。

BUSY：被叫方忙，例如他之前正在通话，或者主动挂断电话。

NOANSWER：呼叫已经到达，但超过了设定的时长没有应答，就是无人接听。

CANCEL：呼叫已经到达，但在被叫方应答之前，主叫方就主动关断了电话。

CONGESTION：呼叫不可达，可能是网络忙，或者对方没有注册到网络中，例如 SIP 终端断线了。

CHANUNAVAIL：通道不可用，对于传统线路来说，可能是该通道无效，例如模块上就没有接上网线，对于 VoIP 来说，一般是终端没有注册。

DONTCALL：被叫方不允许呼入，例如某些系统可以设置为私有模式时。

TORTURE：这个我们用得不多，同样不允许呼入。

INVALIDARGS：错误的 Asterisk 命令参数，只在 1.4.1 以上版本有效。

很显然，这些返回状态相当有用，通过这些状态，我们能够很清楚的了解当前线路的状态。

同时，Asterisk 除了简单的 Goto 应用，还有一个更有用的 GotoIf 应用，他可以使用类似于高级编程语言中的三元操作符来进行判断和操作，例如 a>b?x:y 表示如果 a>b 为真就执行 x，否则执行 y，Asterisk 也一样，它的三元操作符如下：

GotoIf(逻辑判断?跳转序号 1:跳转序号 2)

当逻辑判断为真时，跳转到优先级 1 处，否则跳转到 2 处，1 或者 2 可以省略一个，但不能同时省略，例如$TEST=1?10 或者$TEST=1?:20 都是合法的，表示当 TEST 为 1 时，跳转到优先级为 10 的位置，或者当 TEST 不为 1 时，跳转到优先级为 20 的位置。

32.11Asterisk Realtime

从 Asterisk 1.1dev 开始，Asterisk 开始加入一个新的特性：Asterisk Realtime。Asterisk Realtime 架构（ARA，Asterisk Realtime Architecture）是一些新的驱动和 API，它的主要目标是存储的无关性，也就是想通过 ARA，实现 Asterisk 配置、用户、通话记录等所有数据的存储与 Asterisk 系统主机独立。

例如，通过 ARA，Asterisk 可以运行在一台 Linux 机器上，但它的所有配置可以在一台 UNIX 上，它的用户通讯记录可以在一台 Windows 上，这样，在实现数据独立的同时，也实现了部分冗余。现在的 ARA 主要支持 SQL 数据库方式存储，如 ODBC、MySQL，未来可能扩展到编程语言存储或者 LDAP 存储。

ARA 又提供两种工作模式：静态装载 Static 和动态装载 Realtime。

静态装载主要用于启动时只读取一次的数据，如系统配置，这些配置可以以配置文件形式存在于 Asterisk 主机上，但这就要求管理人员必须登陆主机并使用编辑器编辑文件，也可以以数据表形式存储在某个远程数据库服务器上，这样做的好处前面提到过了，就是提供数据的独立和冗余。

动态装着则主要用于 Asterisk 运行过程中会频繁读写的数据，如用户通信记录、用户登陆记录甚

至用户通话内容等，这些内容之前以文件形式存放在 Asterisk 系统主机上，而动态装载同样可以将这些数据放置到远程数据库服务器上供 Asterisk 使用。

通过配置 extconfig.conf，Asterisk 可以无差别对待这两种存储方式。

对于用户来说，Asterisk 如何处理后台数据，不会对他们产生任何影响，他们的行为也不会有任何改变。而对于管理员来说，将配置和用户数据存放在数据库上更便于管理和配置，而且，数据库提供更强大和方便的检索、回滚和冗余性。

下面，我们就来看看如何让 Asterisk 使用 ARA 特性。

Asterisk ARA 通过配置 extconfig.conf 文件来确定哪些配置会经由 ARA 去控制，例如，在 extconfig.conf 中添加一条语句：

sippeers => mysql,asterisk,sip_user

sipusers => mysql,asterisk,sip_user

表示，Asterisk 在查找 sip 连接端和用户时，都会查找 MySQL 数据源，其数据库名是 asterisk，数据表名叫 sip_user（这两个组件都使用同一个数据表存储数据），其它的与此类似，比如：

extensions => odbc, asterisk, extensions

表示 Asterisk 在读取 extension 配置时，去读取 ODBC 数据库 asterisk 中 extension 这个表的数据。

此时启动 Asterisk，它就会去查找对应的数据库，以读取其中的配置和用户数据，那么，Asterisk 怎么知道它应该读取哪个数据库、怎么样访问数据库呢？

extconfig.conf 中第一个位置是控制 Asterisk 的组件名称，即哪个组件（这里是 sipusers 和 extensions）的数据从哪里读取，箭头后面，第一个位置是数据源的类型，是 mysql，那么就去读取 MySQL 的配置（/etc/asterisk/res_mysql.conf），是 ODBC 就去读取 ODBC 的配置（/etc/asterisk/res_odbc.conf），第二个和第三个位置前面已经很清楚了，是数据源中数据库的名称和数据表的名称。

就像刚刚看到的那样，我们的 sipusers 是配置为访问 MySQL 数据库，那么像对应的配置文件 res_mysql.conf 中就要配置合适的数据库访问方式，例如：

[general]

dbhost = localhost

dbname = asterisk

dbuser = asterisk

dbpass = mypassword

dbport = 3306

dbsock = /var/lib/mysql/mysql.sock

在这个配置文件中，提供了访问一个 MySQL 数据库所需要的全部信息。如果 dbhost = localhost，表示这个 MySQL 服务器是在本机，那么系统会尝试直接使用 UNIX 套接字去访问数据库（通过读取 dbsock 选项），如果这是一个 IP 地址如 127.0.0.1，那么系统会尝试使用 TCP 端口去访问数据库（通过读取 dbport 选项）。

小提示：localhost 和 127.0.01，这是初学者很容易出错的地方，localhost 和 127.0.0.1 都代表本机，例如，在 UNIX 或 Windows 系统中，你 ping localhost，系统都会替你把 localhost 转换为 127.0.0.1，转为 ping 127.0.0.1。但实际上，这两个名字代表不同的意思，localhost 代表本机，系统可以通过 UNIX 套接字文件的方式访问某个网络监听接口，而 127.0.0.1 则只能以 TCP/IP 协议的方式，访问本机 loopback 接口上的某个端口。所以，经常出现的一个错误是，用户在 MySQL 中配置了将权限赋予了 user@localhost，但却使用 mysql -h 127.0.0.1 的方式访问，如果系统中没有两个用户 user@localhost 和 user@"%"，或者两个用户密码不同，就会出现 mysql –h 127.0.0.1 无法访问数据库的情况。

知道了 Asterisk 怎么访问数据库，然后我们就可以建立符合 Asterisk 要求的数据来源，这里是 MySQL 数据库。按照 Asterisk 的要求，建立一个名叫 asterisk 的数据库，并建立 sip_user 数据表，来替换 sip.conf 中用户定义的配置：

```
CREATE TABLE `sip_user` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(80) NOT NULL default '',
  `host` varchar(31) NOT NULL default '',
  `nat` varchar(5) NOT NULL default 'no',
  `type` enum('user','peer','friend') NOT NULL default 'friend',
  `accountcode` varchar(20) default NULL,
  `amaflags` varchar(13) default NULL,
  `callgroup` varchar(10) default NULL,
  `callerid` varchar(80) default NULL,
  `cancallforward` char(3) default 'yes',
  `canreinvite` char(3) default 'yes',
  `context` varchar(80) default NULL,
  `defaultip` varchar(15) default NULL,
  `dtmfmode` varchar(7) default NULL,
  `fromuser` varchar(80) default NULL,
  `fromdomain` varchar(80) default NULL,
  `insecure` varchar(4) default NULL,
  `language` char(2) default NULL,
  `mailbox` varchar(50) default NULL,
  `md5secret` varchar(80) default NULL,
  `deny` varchar(95) default NULL,
  `permit` varchar(95) default NULL,
  `mask` varchar(95) default NULL,
  `musiconhold` varchar(100) default NULL,
  `pickupgroup` varchar(10) default NULL,
  `qualify` char(3) default NULL,
  `regexten` varchar(80) default NULL,
```

```
  `restrictcid` char(3) default NULL,
  `rtptimeout` char(3) default NULL,
  `rtpholdtimeout` char(3) default NULL,
  `secret` varchar(80) default NULL,
  `setvar` varchar(100) default NULL,
  `disallow` varchar(100) default 'all',
  `allow` varchar(100) default 'g729;ilbc;gsm;ulaw;alaw',
  `fullcontact` varchar(80) NOT NULL default '',
  `ipaddr` varchar(15) NOT NULL default '',
  `port` smallint(5) unsigned NOT NULL default '0',
  `regseconds` int(11) NOT NULL default '0',
  `username` varchar(80) NOT NULL default '',
  PRIMARY KEY  (`id`),
  UNIQUE KEY `name` (`name`),
  KEY `name_2` (`name`)
)
```

这是完全符合 Asterisk 要求的 sip 用户数据表。当然针对具体的环境，也许有些字段根本用不上，但我还是建议按要求建立比较好。

在一切建成后，我们可以尝试在这个表中插入一条数据，假设我们之前在 sip.conf 中建立的用户信息如下：

```
[paul]
type=friend
host=dynamic
secret=8888
context=internal
```

那么，转化到数据库中，可以以这样一条语句代替：

mysql> insert into sip_user(name,host,context,secret)values('paul','dynamic','internal','8888');

也就是说，配置文件中的项目，对应了数据库中的字段，应该非常简单明了吧？数据表中其他的字段基本可以不用理会，有些 Asterisk 会以默认的方式去执行，而有些是 Asterisk 会自动更新的。

对于 Asterisk 最重要的配置：extensions，同样可以以这种方式去管理。

首先，在 extconfig.conf 中添加

extensions => odbc, asterisk, extension

然后在原有的配置文件中，在所有想进行 ARA 方式管理的 context 下增加一行，例如，我们想将所有 internal 这个环境的扩展规则纳入到 ARA 管理中来，那么就在 internal context 下加入：

[internal]

switch => Realtime/internal@extensions

这一行的意思是，将所有 internal 环境下的 extension，都转换为 ARA 方式去管理，至于说 ARA 是用什么数据源、数据库和数据表去管理，那就会返回到前面的 extconfig.conf 中的 extensions => odbc...的配置，这里，internal 是指定环境，如果不指定，就使用“default”作为环境，而 extensions 则

是家族。

接着，我们在数据库中添加一个对应 extensions.conf 的数据表：

```
create table extension (
  `id` int(11) NOT NULL auto_increment,
  `context` varchar(20) NOT NULL default '',
`exten` varchar(20) NOT NULL default '',
`priority` tinyint(4) NOT NULL default '0',
`app` varchar(20) NOT NULL default '',
`appdata` varchar(128) NOT NULL default '',
PRIMARY KEY  (`context`,`exten`,`priority`),
KEY `id` (`id`)
) TYPE=MyISAM;
```

对照 extensions.conf，假设 extensions.conf 为以下内容：

```
[incoming]
exten => s, 1, Answer()
exten => s, n, Background(welcome)
```

那么具体到数据库方式的这个 extension 表则变成：

mysql>INSERT INTO extension(context, exten, priority, app, appdata)values(‘incoming’,’s’,’1’,’Answer’,NULL);

mysql>INSERT INTO extension(context, exten, priority, app, appdata)values(‘incoming’,’s’,’n’,’ Background’, ‘welcome’);

可以看出，和文件配置略有不同的地方只有两点，第一点是每个规则都要指定环境 context，第二点是每个规则调用的应用，应用名称是一个字段，其参数是一个单独的字段：

数据库	文件
context	[incoming]
exten	s
priority	1,n
app	Answer(), Dial(),Background()...
appdata	给 app 传递的参数如 SIP/101, welcome 音乐等

通过这种映射，我们能够将原来的配置文件形式的数据，转换到数据库中。