Asterisk，电话未来之路

（CHN1.0）

QQ 群 20556356：Asterisk 爱好者协会

群星友集体译作

2006-4

译者序

确切地说，本文是翻译的组织者写的序。我是 jillyyang，QQ 里叫做 Chinasterisk，也叫 China*，一个的 VoIP 从业者，我的工作是市场调研和分析，这份工作让我喜欢钻研任何和 VoIP 有关的知识。出于对未来市场的乐观估计，我组织了 Asterisk 爱好者协会（QQ 群：20556356 和 23223948），这个 QQ 群在 2006 年 3 月 3 日成立，我儿子出生前整整一周。截止到 4 月 20 日，已经吸引了 170 多位会员。他们中间有研发专家、市场高手，有厂家、有虚拟运营商和运营商，也有行业客户，甚至还有一些老板级的人物潜水其中。或出于对 Asterisk 的爱好，或出于学习的目的，大家建议我组织翻译这本书。也许是出生牛犊不怕虎，我说，试试吧。于是，在开始翻译到现在一个多月的时间，遇到了根本未曾想过的困难，终于有了初步的结果！

这本书需要感谢 20 多位我的好朋友和我不熟悉的人，孙识宇、孙冰、山凌云、蒋运华和李凯原博士，是我最感激的人。本人不是做技术的，因此所有的翻译都要仰仗群里的朋友，审核全都靠他们几位。后面一大串的名字，希望一直留在这个稿子中而不要被人抹去，虽然我知道，这个很难。他们的辛勤工作，对 Asterisk 引入中国，让国人熟悉这个东西，应该是有一定帮助的。我能够做的，我会 best effort。www.openvoip.cn 作为本书的媒体支持，给我和星友们提供足够大的技术及市场文件的共享空间，ftp://ftp.openvoip.cn/asterisk，并给我们提供了论坛的 IP PBX 板块 http://www.asterisker.cn/，给了广大星友一

个非常好的交流和共享机会。在这里感谢他们的老总龙全录和网站编辑陈鲲鹏！

对于 Asterisk，目前中国人知道的还不是很多，并不像 GNUGK 那么流行。但是我想，一年，或者两年以后，应该有无数人对此非常熟悉，其适应的人群数量，应该远远大于 GNUGK。电信市场孕育的革命，也许就和它有关。

那么我们这些人就梦想着成为 Asterisk 在中国的先驱，给后人留下点值得回味的东西。今天我们大家的辛苦努力，希望是值得的！

CHN1.0 版时间仓促，水平有限，我们会继续修订新的版本，请各位读者提出宝贵意见！任何疑问请发邮件到：jilly_yang@263.net，或者到 www.asterisker.cn 里面写下你对本翻译稿的意见和建议。我和所有参与翻译的朋友，以及所有看到这本中文书并从中受益的朋友将不胜感激！

感谢大家的精神鼓励和热情参与！没有不断地催促，也不可能完成这个庞杂的翻译工程！

感谢腾讯，让众多星友通过 QQ 群聚集在一起！

一并感谢云南红塔集团！没有他们提供的红塔山牌香烟，我很难熬夜完成最后的合并工作！

另，本书只用于做技术学习，与任何版权无关。

Chinasterisk

2006 年 4 月

本书所有的翻译者名录

注：括号中为 QQ 名

组织者：杨波（china*）

审核：山凌云（大山），孙识宇（VOIP 计费），张良模

翻译：孙识宇，山凌云，姚海香，杨波，蒋运华，李凯原，才云，王信龙（龙飞），孙冰（飞云 Hoowa），王彦兵（QQ 糖），李君（amy，长安大学 2OO3 级研究生），黄宇（H 恒宇 Y，桂林电子科技大学），刘海兴（胖大海）

网站支持：龙全录（飞龙在线），陈鲲鹏（观摩）

另外还有一些，我至今不知道名字的，但是有他们的 QQ 代码：欣愚，椰皮，禅客，OVER。

作者序

很久以前，有一个小男孩。

有一台电脑……

以及一部电话……

这简单的开始确引来了麻烦!

在不久以后，包括声音、数据以及软件在内的远程通讯都是由两个俱乐部控制，一个是发明这项技术的公司们的，而另一个是利用这些产品提供服务的公司组成的。直到二十世纪九十年代末，Internet的膨胀使得数字远程通讯之门得以打开。通讯价格下落。一批新的、创新性的技术，服务及公司出现了。与此同时，诸如Richard Stallman, Linus Torvalds以及其它一些数不清的免费软件的先驱者们的工作最终形成了真正的开放软件平台Linux(或GNU/Linux)。但是，到处存在的声音通讯，却仍然是被少数公司控制。为什么？或许是因为通过旧的公共电话网络传送的声音缺少万维网(World Wide Web)的魔幻般的光辉吧。又或许是电话不能有效的满足成人们的娱乐。不管原因如何，有一样东西是清楚的，那就是声音通信的开源，即将与软件的开源一样普遍了。

需求(在某些情况下简单体现为便宜)是发明之母。1999年，Linux Support Services开始提供免费和商业的技术来支持Linux，我发现我需要(至少是意识到需要)一个电话系统来对我提供24小时的技术支持。这个想法是这样的：人们可能打进电话，输入消费都的身分，并留下信息。然后系统就近安排一个技术人员来对顾客的要求作出回复。因为我的公司的启动资金只有4000美金，我没有能力来承担我所想实现的这个电话系统。我从1994年就是Linux的用户，通过12tpd，gaim及cheops我已经了解开源软件了，加之在没有任何人来对我解释这样一个任务的复杂性，我决定我只是简单的使用从Adtran(我曾作为一个实习生在那工作)公司借来的硬件来制作我自己的电话系统。当我把电话打入一台PC机电，我发

现我高兴得什么都说不出来了。实际上，Asterisk公司的格言(格言对于一个需要效率的大项目来说是必须的)也就是从这个简单的猜想而来：

这只不过是软件！

不管好坏，我都很看重这个东西。也就是从这开始，我产生了Asterisk能做所以和电话技术有关的事的想法。选择“Asterisk”这个名字是因为它不仅是电话标准的关键，同时也是Linux中的通配符号(例如：rm-rf*)。

所以，在1999年，我已经拥有了免费的电话平台，并将其放到了网上，同时我仍继续提供Linux的技术服务以维持我的生活。而且到了2001年，经济萧条，从事Asterisk方面的Linux服务似乎比一般的Linux服务显得更好了。那一年，我们联系了Zapata电话技术项目的Jim“Dude”Dixon。Dude的令人激动的工作对Asterisk来说是一个很棒的方案，这个方案给提供了一个使Asterisk获得公众观注的商业模型。在与Dude合作制作了我们的第一款PCI电话接口卡后，“Linux Support Services”这个名字显然不再适合一个电话技术公司了，所以我们更名为“Digium”，那又是另一个故事了。随着语音IP技术(VoIP)的广泛使用，通过旧的电路开关网传播的声音转变为新的以IP技术的基础的网，一切终于走上正轨了。

现在，很明显大多数人并不会因为电话而感到新奇，兴奋。当然也只有少数可以感到当我听见声音从与电脑联接的电话中传出来时的喜悦，但是，也正是这些的确对旧式电话感到兴奋的人，由于网络的作用，这一小部分能联合起来使我们这种新奇的想法变成一个普通的，实际的能为大众服务的东西。

如果说电信已经对开源的方案成熟，那只是一种保守的说法。由于生活和工作中无处不在的电话，电信业是一个巨大的市场。电信产品的市场有一群高技术的员工，他们都能也愿意为这个事业做出。人们需要更人性化的技术，而垄断的电信业非常的昂贵。而Asterisk技术大有星火燎原之势。

Asterisk也提供了与旧的处理方法转形变的一座桥梁，从而位于这一系列技术的顶端(垄断→开源，电路开关形→VoIP，单一声音→声音，视频以及数据，数字信号进程→主机媒介进程，集总式的→点对点的)。通过Asterisk,你能与六十年代的脉冲拨号的电话通话，也能与最新的VoIP设备通话，并能提供从简单的开关一直到现在的蓝牙与DUNDi技术的功能。

但是，最重要的是它同时也展示了如何协调为了完成某些项目而共同工作的，却又相隔一定范围公司和个人是如何内工作的，而这些项目却是某一公司和个人无法单独完成的。为了现实Asterisk,我特别要感谢Linus Torvalds, Rachard Stallman以及整个Asterisk团队和那些发明Red Bull的人。

那么，Asterisk的发展如何呢？回想一下PC的发展历程。当PC在八十年代刚进入市场时，它功能并不强大。可能，你可以用它来制作电子表格，可以用它来进行文字处理，总之，用处不多。然而到后来，它那开放的体系使得其价格降低，同时，新产品的出现使其在应用方面逐渐超过了小型机，然后又超过了大型机。现在，即使是超级电脑Cray也采用的是以Linux为基础的x86构架。我期望Asterisk的发展也能像PC寻机。现在，大量的电话系统采用了Asterisk技术。到将来，谁又能知道这项技术的瓶颈在哪呢？

所以，你还等什么呢？赶快学习并加入到未来的开源通信中来吧，请加入Asterisk革命！

——Mark Spencer

前 言

这本书适用于任何刚开始学习 Asterisk 的人。

Asterisk 是一个开放源代码的，集中于一点的电话平台，并且其设计主要适用于 Linux 操作系统。Asterisk 把超过 100 年的电话知识融入到了一套充满活力的严密集成的电子通迅应用程序中去。它的强大的功能在于它的用户化的本质，除此还有其无可匹敌的对各种标准的适应性。没有其它的专用分组交换机能够具有如此之多的富于创新性的功能。

像声音邮件，会议主持，呼叫等待以及代理，等待音乐和呼叫搁起都是正好被加入到软件中去的一些标准的功能。除此之外，Asterisk 还能够集成一些其它的与之相近的商业技术，这些都是其它专用分组交换机几乎不可想象的。

对于初学者来说，Asterisk 可能会非常使人畏惧和复杂，而这也正是为什么文档对于它的成长有如此重要的作用的原因。文档降低了进入这个领域的难度并且能够帮助人们思考可能会发生的事物。

我们花了很长的一段时间去完成这本书，它的目的在于实现传递这样一个文档，它介绍了绝大部分 Asterisk 基本的组成部件——而这些正是一个 Asterisk 的初学者所必须知道的。这是这本书的第一版，同时我们坚信这将是一个关于 Asterisk 的巨大的知识图书馆。

这本书由 Asterisk 社团所著，并且也是为 Asterisk 社团所著。

关于读者

这本书是为 Asterisk 的初学者准备的，但我们假定你已经熟悉 Linux 操作系统的管理，网络设置和其它的 IT 技能。如果这是这样的话，我们建议你去较大较好的图书馆，找找关于这方面的由 O’Reilly 出版的书籍来学习。同时我们也假定你同样也是一个电子通迅的初学者，包括传统转换电话和新的 IP 间通话的世界。

本书结构

本书主要由以下章节组成：

第一章　一次电话的革命

在这一章里，我们摩擦火柴，并且点燃火焰。Asterisk 将要改变整个世界的电信行业，在这一章里我们所要讨论的正是我们为什么会有如此的信心的原因。

第二章　为 Asterisk 准备一个系统

这一章覆盖了当你在设计一个电信系统时，必须照顾到的一些工程学上的考虑。很多这些内容是可以被跳过的，如果你打算立刻安装的话。但是，如果你计划把一个 Asterisk 的系统应用到生产上的话，这些都是需要知道的重要内容。

第三章　安装 Asterisk

这一章覆盖了 Asterisk 的获得，编译和安装方面的内容。

第四章　Asterisk 的初始化设置

这一章介绍 Asterisk 的初始化设置。在这一章里我们将会介绍一些重要的设置文件，它们的存在是为了定义通路和特征，以适用于你的系统。

第五章　拨号方案基础

这一章介绍 Asterisk 的核心部件，拨号方案。

第六章　更多的拨号方案概念

这一章将介绍更多的高级的拨号方案的概念。

第七章　了解电话

这一章与 Asterisk 无关，而是讨论一些更加重要的应用于公共电话网的技术。

第八章　VoIP 协议

接着电话遗传的讨论，这一章将要讨论的是 IP 间如何通话。

第九章　Asterisk 的网关接口（AGI）

这一章介绍一个更加使人惊奇的部件——Asterisk 网关接口。通过使用 Perl，PHP 和 Python，我们将会示范怎么样使用外部程序而使你的专用分组交换机增加几乎无限制的功能。

第十章　Asterisk 与 Uber-Geek

这一章简要地介绍对于一些难以置信的特征和功能来说，什么是真正的富有。而这些都只是Asterisk现象的一部分。

第十一章　Asterisk：电话的未来

这一章预言将来开源电话将完全改变一个产业，这种改变将是通过对不顾一

切的革命的需求来完成。

关于软件

这本书主要是基于 Asterisk 的 1.2 版本的文档的，然而，这本书中的很多规定与资料信息并不能完全确定是基于哪个版本文档的。我们学习了 Red Hat 的语法，从而用 Linux 操作系统来进行对 Asterisk 的运行与测试。我们相信基于 Red Hat 的销售并不是每个人都喜欢的选择，但它的版面设计和功用对有经验的 Linux 操作系统的管理者来说仍然是很熟悉的。

本书中所用的一些范例

本书中将会用到以下排版上的范例：

斜体

用以说明新的术语，网址，电子邮件地址，文件名，文件的扩展名，路径名，目录和 Unix 操作系统的功能。

连续的宽体

用以说明命令，选择，参数和一些必须被替换成命令的意见。

连续的粗宽体

用以说明以这样的字体表示的文字或者命令需要使用者完全逐字地进行输入。同时也能够用以在代码中对一段代码进行强调的作用。

连续的斜宽体

用以说明以这样的字体表示的文字需要被用户自己提供的数值来进行替换。

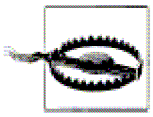
[关键字或者其它东西]

用以说明一些选择性的关键字和意见。

{ 选择-1 | 选择-2 }

用以表示或者选择 1，或者选择 2。

这个图标表示一个提示，建议或者是一个大致的注释。



这个图标表示警告和注意点。

使用代码范例

本书的作用是为了帮你更好地完成你的工作。总的来说，就是你可以在你的程序和文档中使用这本书中的代码。除非你是要再造很大的一部分代码，否则，你不必要跟我们联系并取得我们在代码使用上的同意。例如，如果你在写一个程序时要用到这本书中的几个程序块，那么你并不需要征得我们的同意。为了回答一个问题而要例举或者引用到本书中的代码示例，则也不需要征得我们的同意。但是，当你要把本书中大量的代码示例写入到你自己的产品的文档中的时候，就必须征得我们的同意了。

目 录

第1章　一场电话革命

它不需要多数人去效仿，而是需要少数充满愤怒、不知疲倦的人迫切的给人们的思想点上一把火。

——Samuel Adams

一场难以置信的革命正在进行中。这场革命孕育了很久，人们对它的到来也已经期待已久，但现在它已经开始启动，将没有任何东西使它停止。它发生在一种科技发展的时代，这种技术使其它每一种自认为高科技的产业都为之难堪的觉得自己远远落后了。这一产业就是电信业，而一种称为 Asterisk™ 的开源专用交换机更是成为了这一场革命的助燃剂。

毫无疑问地说，直到现在，电信业仍是最后一块开源革命尚未波及的主要电子行业领域。大部分电信制造商仍在令人可笑的在制造昂贵、不兼容的系统，在使用不寻常的设计但却陈旧的硬件上运行古老而复杂的代码。

例如，北电的商业通信管理器(Business Communications Manager)使用 Windows NT 4.0 服务器软件，一个有着十五年历史的基于 VSWorks 的按键电话交换机，一个 700MHZ 的 PC 等组合在一起的。花五千到一万五千美元就可以买到所有这些东西，当然不包括电话机。你若想让它做点真正有意义的事情，就必须得花额外的许可费用来购买只限少数人使用、功能有限的盒装软件。定制？还是别想了——那不在计划之中。兼容未来的技术和标准？给他们一到两年时间，他们正在研究当中！

所有的大电信厂商都是提供套路类似的产品。他们不希望你有灵活性和选择性；他们想把你锁在他们的产品圈子范围里。

Asterisk改变了这一切。有了Asterisk，没人会告诉你你的电话如何工作，或你会被什么技术所限制。如果你想得到它，你就可以拥有它。Asterisk亲切的接受标准兼容的概念，但它也在享受推进自我创新的自由。你所要实现的完全由你自己决定——Asterisk不强加任何限制。

当然，这种难以置信的灵活性是有代价的：Asterisk不是一个配置简单的系

统。这并不是因为它不合理、混乱或神秘性；相反，它有很强的判断能力和实用性。当人们第一次看到Asterisk的拨号方案时，便会眼前一亮，并开始思考它的各种可能性。但当表面上有数以千计的途径可以获得同一结果时，这个处理过程就需要额外的努力。也许这可以比作修建房子：各个部分都是相对比较容易理解的，但筹划这项任务的人必须a）获得有能力者的帮助，或者是b）通过培训、练习和一本关于这一题目的好书来获得所需的技能。

1.1 VoIP:连接传统电话和网络电话的桥梁

IP电话(VoIP)总是被认为并不比免费的长途电话更好，但VoIP真正的价值（老实说应该是其竞争力）却在于它使得声音在数据网络中变得仅仅只是另一种形式的应用。

有时候我们似乎忘记了电话的目的是用于交流的。实际上，这只是一个很简单的目标，而且我们应该用比现在更加灵活和创新的方法来使用它。自从在工业上追求这样一个目标显得勉强，一个由充满激情的人们所组成的大的社团便投入到这项工作中来了。

挑战来源于这样一个现实：一个行业在最近一个世纪内变化甚微，而现在它仍没有任何开始着手改革的想法。

Zapata 电话工程

Zapata电话工程的构思最初是由一个名叫Jim Dixon的电信通讯咨询设计师提出的，他从电脑速度以不可思议的速度在发展以至于电脑行业不得不接受它的例子中得到灵感。Dixon相信，如果有这样一张卡，这张卡上只有必须与电话电路接触的基本电子部件，那么就可以制造出成本大大降低的电话系统。与卡上需要昂贵的电子元件不同的是，在CPU内，将通过软件控制数字信号处理（DSP）。虽然这会大大增加CPU的负担，但Dixon很肯定的说，相对于他们的性能，CPU的低成本使它比昂贵的单片机更有吸引力，更重要的是，随着CPU功率的不断提高，它的性价比将得到不断改进。

与很多空想家一样，Dixon相信有很多人也会看到这样一个机会，而他需要

做的只是等待别人为他创造一个对他来说是明显改进的机会。几年之后，他注意到不但没人研制出这样的卡，而且看起来不可能有任何人打算从事这方面的研究。基于这一点，显而易见的是如果他希望这场革命发生，就必须由自己来亲自发起。因此，Zapata电话工程就着样诞生了。

因为这一概念是非常具有革命性的，它肯定会在行业内引起很多波动，所以我从研究墨西哥革命的动机上着手，并以著名的墨西哥革命家Emiliano Zapata来命名这一技术和机构。我决定把这种卡叫做‘tormenta’，在西班牙语中是暴风雨的意思，但在文中通常用来表示一场大的灾难的来临，比如飓风或诸如此类的东西。

也许我们应该把自己叫做Asteristas。不管怎样，我们都欠Dixon一个人情，一部分是要感谢他发起了这场革命，一部分是感谢他发现了这一机遇，但更重要的是因为他把自己的研究成果献给了开源社区。正是由于Jim的贡献，使Asterisk的公用电话网（PSTN）引擎变成了现实。

1.2　伟大的变化需要灵活的技术

世界上最成功的按键电话系统存在一个制约了它15年的设计局限，从用户的要求上看来，它只是一个简单的变化：当你决定让你的电话在转向语音信箱之前响铃几次时，你可以从2、3、4、6或10个响铃周期中选择。当用户要求五个响铃周期时，应该响铃几次？你对此有何感想？然而，在生产商们的头脑里，这肯定不是一个麻烦。他们说，这就是它的工作方式，用户们必须去克服它。

这只是其中一个例子——行业里普遍存在这样的问题。

同一个系统上另一个例子是，在话机上编程设置名字的时候，其长度只能是七个字符。回溯到20世纪80年代，当这种特殊的系统刚刚发明时，随机存储器非常昂贵，而储存几十个这样七字符长的文件就代表了巨大的硬件成本。但今天仍

沿用这样的规定的借口是什么呢？没有任何原因！他们有任何的计划去尝试改变这个局面么？几乎不可能——这个问题甚至还没有正式被公认为是一个麻烦。

虽然现在非常适合也非常应该在一种系统上集中精力，但现实的问题是，每一种现存的交换机都有缺点。不管那种系统功能如何全面，总会有所遗漏，因为即使是功能最强大的交换机都总是不能正确预见用户们的创造性。少部分顾客群或许会希望增加一个小小的临时功能，但设计团队预想不到也不能证明这样制作的成本是正确的，再者，因为这个系统是封闭的，因此用户们也不能亲手去制作它。

如果因特网会因为规则和商业利润问题而因此发展受阻，那么它能否发展到今天能有如此广泛的接受度，将会成为疑问。因特网的开放性意味着任何人都有涉及其中的能力。所以每个人都能参与其中。数以万计的人都构想着合作建立因特网收费传输系统，但没有一个公司能够做到。

和许多其他开源项目一样，比如Linux和因特网，人们认为一定会有比现在现有产业所生产的东西更好的产品，他们的梦想进一步促进了Asterisk的开发。开源团队的强大在于，它不是由分配着具体任务的雇员组成的，而是有来自不同行业的人组成，他们有着各种不同的经历，有着各种关于灵活和开放方法的想法。这些人认为如果一个人能够获取各种各样的交换机的最佳部件，并分别将它们组成一个相互联系的部件——类似于一箱LEGO砖块——那么他就可以在传统的合作风险分析方法中无法实现的情况下开始构思一样东西。这种东西看起来应该是什么样的，虽然还没有一个人能够郑重宣布对此有一个完整的描述，但这并不缺乏建议和想法。

许多刚接触Asterisk的人都认为它是未完全实现的。也许这些人可以比作进入一个艺术工作室希望得到一幅有签名、限量发行的印刷品的游客。他们常常失望地离去，因为他们认为Asterisk只是白帆布、涂漆的管子、可以使用而又从未使用过的刷子。

即使是在它成功的早期阶段，人数比任何其它一种交换机都多的艺术家在小心呵护着Asterisk的成长。大部分生产商在任何一种产品上最多只会分配几个开发者；而Asterisk有几十个。大部分私有的交换机拥有一个世界性的但只有几十个真正的专家组成的支持团队；而Asterisk拥有几百个。

在电信行业，围绕在这种产品上的专家的深度和广度是不可匹敌的。Asterisk正享受着那些仍记得手摇拨号电话的电话公司的家伙们，正在回忆语音信箱成为最热门的技术时刻的电信企业家们以及帮助建立因特网的从事于数据通讯的人们的友情关注。他们有一个共同的信心：电信通讯产业需要一场正确的改革！

Asterisk就是催化剂。

1.3 Asterisk：黑客的 PBX

选择忽视Asterisk的电信通讯公司正在经受危险。Asterisk的灵活性所能够创造的可能性就算是最好的私有系统都无法想象。这是因为Asterisk是黑客最终选择的交换机。如果有人叫你不要使用术语“黑客”这个词，拒绝他。这个术语并不属于大众媒体。他们盗用并滥用这个词，用它来表示“怀有恶意的破解者”。我们是时候应该收回了！黑客们建立因特网上的网络引擎。他们建立Apple Macintosh和Unix操作系统。黑客们也建立你的下一个电信系统。别害怕；他们是一群好人，他们能够建立一个远比现有系统安全的系统，因为与封闭系统的安全可疑性、容易被破解不同的是，他们能够对安全性的变化趋势做出快速反应，并可根据公司策略和业界最佳实践对电话系统做出优化调整。

和其他开源系统一样，Asterisk将发展成比任何私有系统更为安全的平台，尽管它来源于黑客，而正是由于这个来源，它才更加安全！

1.4 Asterisk：专业人员的 PBX

在电信通讯的历史上，从来没有一个不管以任何价钱为代价的，适应业务需求的系统。

Asterisk是一项可行的技术，正如Linux一样，将越来越难见到一家不是在运行Asterisk某个版本的系统的公司，在某种能力上，在网络的某些方面，只有Asterisk能解决问题。

基于以下几个原因，Asterisk的被接受程度很可能将会比Linux发生的更快：

1．因为Linux已经为开源的公众接受铺平了道路，所以Asterisk可以循着它的路走下去。

2．电信产业是不健全的，巨人产业玩家们不会提供任何领导能力。Asterisk

拥有一个引人注目的、现实的、令人兴奋的前景。

3．终端用户受够了不兼容、功能有限和令人生厌的支持问题的麻烦。Asterisk解决了前两个问题；团队已经对后一个问题表示出了足够的激情。

1.5 Asterisk 团队

Asterisk的一股引人注目的力量就是发展了并支持着它的团队。这个由Digium的Mark Spencer领导的团队，强烈意识到Asterisk文化的重要性，他们对未来充满了眼花缭乱的想法。

Asterisk团队力量产生的一个更为强大的副作用是萌芽于共同热爱着这项事业的电信通讯专家、网络技术专家和信息技术专家之间的相互合作精神。虽然在传统上这些专家们不能达成相互一致，但在Asterisk团队里他们都非常欣赏每个人的技能。这种相互合作精神的重要性是不能低估的。

尽管如此，Asterisk的梦想如果要实现，其团队必须壮大——目前团队面临的主要挑战就是新用户的不断涌入。现存的团队成员孕育产生了名为Asterisk的这种东西，他们都普遍欢迎新用户加入进来，但他们已经对被问到各种各样的问题开始感到不耐烦，如果一个人肯花必要的时间去做研究和实践的话，这些问题的答案往往是可以自己独立获得的。

很明显，新用户不适合任何独特类型的模型。尽管有些人非常乐意花几个小时去体验和阅读各种各样描写别人考验和苦难的博客，但很多开始对这种技术感到兴趣的人对这种探索缺乏足够的热情。他们希望获得一个简单的、直接的、按步骤执行可以使他们能启动和运行设备的指南，并提供几个容易理解的描述完善常见功能（比如语音信箱，自动服务和类似的功能）最佳方法的例子。

对于（准确地）认识到Asterisk就像一门设计语言一样的专家团队成员们而言，这种方法是没有任何意义的。对他们来说，很明显你必须自己沉浸到Asterisk之中去亲自感受它的微妙。你觉得会有人索要一本关于编程的操作步骤指南，并且希望从中学到语言必须提供的所有东西吗？

显然，没有一个人希望它对每个人都适用。Asterisk是一种全然不一样的动物，它需要一种完全不同的照看方法。你若要研究这一团队，可是请注意这里有

许多有着不同技能背景和态度的人。他们其中的一些人对新用户没有表现出足够的耐心，但那往往是因为他们把自己的热情都投入到了这项课题当中，而不是他们不欢迎你们的参与。

Asterisk 邮件列表

和任何团队一样，也有地方可以让Asterisk团队成员们聚在一起讨论大家共同感兴趣的东西。你可以在http://list.digium.com上找到邮件列表，下面所列的是目前最重要的三个：

Asterisk-Biz

这里包含任何与Asterisk有关的商业行为。如果你想出售一些Asterisk相关产品，请在此销售。如果你想购买一项Asterisk服务或产品，请发邮件到这里。

Asterisk-Dev

Asterisk开发者们在此漫游。这个列表的目的是讨论Asterisk软件的发展以及如何让参与者们精力旺盛地防护这一目的。如果你在此发表任何与编程或发展无关的话题，将会遭到很多人的愤怒。

Asterisk-users

这是大多数Asterisk用户光顾的地方。这里每天产生几百条信息，并且有上万个阅读者。虽然你可以到这里来获得帮助，但希望你在发表疑问之前，确信自己已经亲自阅读了一些东西。

Asterisk Wiki

Asterisk Wiki是很多启迪和困惑的发源地。http://www.voip-info.org 是团队拥有的一个VoIP知识库，网站上有关于很多学科的许多令人着迷、情报性的、经常相互对立的信息，其中一部分就是关于Asterisk的。

因为到目前为止Asterisk文件的大部分信息形成于这个网站上，而且很可能它所包含的Asterisk知识比所有其他来源加起来还要多（邮件列表档案保管处除外），因此它一般被认为是查找Asterisk知识的地方。

IRCC 频道

在irc.freenode.net上Asterisk团队拥有一些因特网中继聊天频道。最活跃的两个是#Asterisk和#Asterisk-Dev。为了避免广告机器人的闯入，这两个频道都必须要经过注册才能进入。

Asterisk 文件工程

Asterisk文件工程最初由Leif Madsen和Jared Smith提出的。团队里的很多人都对此做出了贡献。

文件工程的目的是为Asterisk的书面工作提供一个知识体系。与Wiki灵活而独特的特性截然不同的是，文件工程对建立与Asterisk相关的多方面学科更靠近的体系更加关注。

实现网上浏览是在Asterisk文件工程方面所做工作的一部分，这本书公布在 http://www.asteriskdocs.org 网站上，读者可以通过一个富有创造性的公用帐号就可以浏览到。

Asterisk 爱好者协会以及其它与 Astersisk 有关的 QQ 群

作为国内的Asterisk爱好者QQ群，2006年3月成立，主要致力于将Asterisk引入中国，吸引更多的人群接触、应用Asterisk，努力缩短国内Asterisk、开源IP PB方面和国外的差距，并寻求商业机会。本书的翻译就是在Asterisk爱好者协会的各位星友和其它几个友好群的支持下完成的。

目前已知的研究Asterisk的部分QQ群号如下：

Asterisk爱好者协会，20556356，23223948；

Asterisk，15805606；

Asterisk论坛，9638587；

H323-SIP研发协会，2095841；

中国SIP联盟，14188410，16354383。

中国和 Asterisk 有关的论坛

www.asterisker.cn，一个探讨IP PBX的板块；

www.openpbx.cn，一个探讨开源PBX的板块；

当然，至今中国还没有自己的Asterisk WiKi，相关的探讨还是以扫盲为主，希望在中国国内会有更多的爱好者和参与者，共同发展Asterisk。

1.6 商业案例

在当今世界，几乎很难见到有哪些商业行为在不到几年时间之内无需重新构建自身的发展的。同样也很难见到这样一个商业案例：在每次它进入一个新方向时，都能够承担得起更换它的基层通信机构组织。今天的商业，在所有的技术领域都要求有极端的灵活性，包括电信业。

在Geoffrey Moore编写的《Crossing the Chasm》（Harper Business）书中，他认为，“在安装提示那一刻，人们才会发现而不是知道这一系统的价值，反过来，产品的灵活性，可适应性以及正在进行的计算服务，应该成为购买者估价核对表的关键部件。”在局部上，这句话的意思是，一项技术的真正价值只有直到它已经被配置使用了，往往才能被人们所知道。

因此，拥有一个内心具备宽广的概念和不断创新价值的系统是多么引人注目！

1.7 关于本书

从哪里开始呢？当说到Asterisk，就有许多要谈论涉及的，这远比我们可以安排进一本书的内容要多得多。从现在开始，我们并不准备给你讲述技术人员所走过的每一条路——我们只讲述基本的东西。

在第一章，我们将介绍当设计一个电信通讯系统时，你头脑里应该有的一些工程性概念。如果你只是想知道如何安装，你可以跳过这里的很多内容，但如果你想把一个Asterisk系统变成产品，这些就是必须弄懂的重要概念。

第二章涉及获取，编辑和安装Asterisk的问题。第三章将处理Asterisk的初始配置问题。为了定义适用于你的系统的一些通路和特点，这里我们会提到一些必

须存在的设置文件。这也是为第四章的内容做准备，第四章我们将介绍Asterisk的心脏——拨号规则。介绍了拨号规则的基本知识后，第五章将介绍更多高级的拨号规则相关概念。

第六章我们暂时不谈论Asterisk，而是讨论一些在PSTN的使用中更为重要的技术。实际上，紧跟着传统电话的讨论，第七章将讨论IP电话。

第八章介绍一个更为令人惊奇的部分——Asterisk 网关接口（AGI）。我们利用 Perl，PHP 和 Python 来阐明外部程序可以毫无功能性限制的应用于你的交换机中。第九章主要介绍丰富而形式多样的，令人难以置信的特点和功能，而所有这些都只是 Asterisk 现象的一部分。作为结论性的部分，第九章将展望未来，它预言一个被开源电话技术完全改变的，迫切需要一场改革的产业的未来将会来临。在本书的附录部分你也将可以获得一份关于参考信息的宝贵财富。

在本书中，我们只能着眼于基本的知识，但从这个基础起步，你将可以理解 Asterisk 的概念——而且从此开始，还有谁会知道你将要构建什么？

第2章　Asterisk 的系统准备

很早以前，我知道在未来某个完美的一天，计算机内部就可以解决所有需要处理的功能，并且廉价地通过所需的外部硬件连接电信接口。

——Jim Dixon,"Zapata 电话系统的历史以及和 Asterisk PBX 的关系"

现在，您一定十分急切地想要安装运行Asterisk系统。如果只是感兴趣想建一个非专业的系统，你可以跳到下一章直接开始安装。然而如果要做一个任务性的部署，必须要考虑Asterisk系统运行的环境，不要有任何的错误。Asterisk是一个十分灵活软件，可以轻松地安装在任何Linux平台，也可以安装在几个非Linux平台上（有人在主WRAP板、 Linksys WRT54G路由器、Soekris系统、Pentium 100、PDA、Apple Macs、Sun SPARC、便携式计算机，以及其他系统上成功地编译和运行Asterisk。当然是否要把这样的系统用于生产环境完全事另一个问题。(实际上，Kristian Kielhofner的AstLinux发布在Soekris 4801主板上运行得很好。一旦掌握了 Asterisk 的基础，它是一个很值得进一步研究的东西，参见 http://www.astlinux.org) ）。然而，为了帮助你理解Asterisk在哪种环境上可以运行最佳，这章你需要认真阅读并可以得到一个设计良好、可靠性高的系统。

Asterisk的资源需求与其他的嵌入式、实时的应用类似，都是通过优先级的方式来访问处理器及系统总线。规定系统上的任何功能都不能直接调用Asterisk优先机级较低的进程。在类似的或非专业系统，这也许不是很重要。然而对于一个高性能的系统而言，性能上的缺陷将会引起用户诸如话质差的问题，比如经常出现回声、噪音等。这种情况在手机超出服务区外会出现，尽管主要的原因不在于此。当负载增加，系统保持连接将会出现困难，因此在选择平台处理器的时候，要特别注意性能的要求。

表2-1　系统需求指南

系统	通道数	最小要求
非专业系统	<5	400MCPU,256M内存
SOHO系统	5~10	1GCPU,521M内存
小型商用系统	10~15	3GCPU,1G内存

中等商用系统	>15	双处理器，在分布式架构里采用多个服务器

对于大规模的Asterisk系统安装，一般都把一些功能分布到几个服务器上，一个或多个核心单元专门负责呼叫处理；通过一个或几个冗余的服务器来管理外围设备（如：数据库、语音信箱、会议、管理、WEB界面、防火墙等等）。对于多数Linux系统，能够很好地满足Asterisk增长的需求：一个小的系统可以处理所有的进程调用，当需求超出负载能力时候，外围设备管理的任务就会分布到几个服务器上。灵活性是Asterisk可以满足业务快速增长的主要原因—因为在开始购买阶段，预算上没有特别明确、具体的最大或最小的方案。即使在可以预测预算的大多数电话系统中，我们仍就可以听到一个方案和Asterisk一样便宜。说这些的目的是为了解释，分布式Asterisk系统在设计上并非易事，尤其是对于Asterisk的新手来说（如果确定要建立一个分布式的 Asterisk系统，就要研究DUNDi 协议。或许会对Asterisk-Users邮件列表产生兴趣，但是要做好准备。由于某些原因，这个主题的争论很激烈。）。

2.1　服务器硬件选型

服务器的选择既简单有复杂：说它简单是因为任何X86架构的平台都可以胜任；说它复杂是因为系统可靠性完全依赖平台设计时所作的考虑。选择硬件时，必须认真考虑系统的整体设计、所需要支持的功能。这将帮助确定对CPU、主板、电源的要求。如果只是想要学习Asterisk，可以不用考虑本节提供的这些信息。但是，如果要建立一个使命性、适合于部署的系统，下列这些问题必须要给与足够的考虑。

性能问题

综合考虑，当你选择硬件环境用于Asterisk安装，必须要考虑的是：系统功能应该有多强大？这个问题很难回答，因为系统消耗的资源主要取决于系统的工作方式。没有Asterisk工作矩阵可以参考，所以你要Asterisk如何工作以便做出明智的决定，究竟需要什么样的资源。你应该考虑以下的因素：

系统能够支持的同时连接的最大数目，每个连接都会增加系统的负载。

致强处理器能够处理的压缩数据编码（G729和GSM）流量的能力。Asterisk用软件完成DSP工作，因此处理器性能对于它所能支持的并发通话数量有很大影响。一个系统可以处理突发的50个G711的连接请求，同时处理10G729扁平通道只用了它一小部分能力。

是否提供会议的功能，提供什么级别的功能。是否系统负载较大？会议功能需要系统转换编码并把每个通道的输入语音码流混合成多输出码流。混合多输出码流几乎是实时的并极大的加重了CPU的负载。

回声消除：

回声消除可能是需要的，尤其在PSTN接口中任何呼叫中基本包含。既然回声消除是一个数学的功能—系统要实现，CPU负载必然会增加。

拨号方案逻辑脚本：

只要Asterisk把呼叫控制传递给外部程序，就会有性能的损失。尽可能把逻辑功能放在拨号方案中。如果非要调用外部脚本，在设计脚本时，尽量要把性能和效率作为重要考虑的因素。

我们还不能准确地判断这些因素对性能的影响。这些因素都是一些基本常识，精确的性能计算方式还没有。这只是一部分，系统中每个部件的影响都是变化的，例如CPU的电源、主板的芯片组、系统的流量，Linux的内核优化，网络的流量、PSTN接口的数量及类型，PSTN的流量——没有提到的系统并行完成的非Asterisk服务，让我们看看几个关键因素的影响：

编码及转换：

简单的说，编码（编码/解码 压缩/解压）就是把模拟的波形数字化的数学计算规则。各种编码的区别主要在于压缩的级别和质量。一般来说，压缩的越厉害处理器编码解码的工作量越大。解压编码不会对处理器增加很多的工作量（但要求网络的带宽），编码的选择必须在带宽和处理器负载之间寻求一个平衡点。

中央处理单元：

一个CPU包含几个部分，一个是浮点运算单元（FPU）。CPU的性能主要取决于由它，决定系统的许多用户能否有效的得到支持。下一章〈如何选择处理器〉会帮助你选择CPU以满足你系统的要求。

系统上并行运行的其它进程：

如同UNIX，Linux设计上可以处理多个不同的进程。当一个进程（如Asterisk）需要较高的优先级得到系统的响应时会出现问题。默认情况下Linux对每一个请求平等地分配资源，当你在一个系统里安装许多不同的服务应用时，这些应用都允许自己的进程使用CPU。然而Asterisk频繁需要高优先级地访问CPU，这样就不会与其它的应用协调好，如果必须与其它应用共存，系统则需要特殊的优化。首先对系统中的各种应用赋予优先级，在安装时要注意哪个应用是作为服务安装的。

内核优化：

默认情况下，Linux发布中很少针对某个应用的性能进行内核优化。因此需要特别的考虑。至少，无论选择的是什么发布，应该下载一个新的Linux内核（http://www.kernel.org），并在你的平台上编译，也可以下载能够改善系统性能的补丁。但是这些是考虑到具有官方内核支持的版本。

中断请求延迟：

中断请求（IRQ）延迟是指外围设备卡（如电话接口卡）向CPU发请求，停止目前工作并做出响应并准备处理任务应这短时间的延迟。Asterisk外围设备（尤其是Zaptel卡）对中断请求延迟的容忍都极其有限（Linux对于迅速响应中断历史上就存在问题，这对语音开发来说是一个很大的麻烦，有一些补丁可以解决，有一些争论关于如何协调补丁与内核的关系）。

因为Digium卡要求很高，因此建议一个Asterisk系统上只安装一块Digium卡，如果系统需要更多Digium卡的连接，或者用一块高密度的卡替换现有的卡，或者在系统里添加另外一台服务器。（很多人报告，Sangoma卡在用于未知的主板芯片组具有更好的健壮性 ，并且能够处理共享的主板IRQ资源。不管怎样，还是值得考虑使用多个服务器，因为这样所获得的冗余能够很快抵消成本。）

内核版本：

Asterisk官方支持Linux V2.6.

Linux版本：Linux有许多种类，下一章，我们将讨论如何选择Linux版本，如何安装并同时运行Linux和Asterisk。

选择处理器

既然Asterisk系统的性能要求大量的数据运算，选择一个强大的浮点运算处理器至关重要。Asterisk的信号处理需要CPU能够处理惊人的复杂的数学计算。这些任务都取决于处理器内部浮点运算单元的能力。

为Asterisk选择一个最好的处理器需要了解飞速发展的计算机界。即便在编写出版这本书的时候，处理器的运算速度仍在飞速发展，可以满足Asterisk各种的架构。显然这是件好事，但对于选型却难以定夺。很自然，浮点运算单元越强大，处理器越可以处理越多的并行的连接数，这是最根本的考虑。当你选择一个处理器，主频只是考虑的部分，浮点运算能力才是关键，Asterisk处理器对此要求很高。

Intel和AMD的CPU都有强大的浮点运算单元，写本书的时候，Intel 芯片组都支持32位系统，而AMD已经可以支持64位了。不过在您阅读本书的时候，事实可能已经不是如此了。（如果你希望弄得非常清楚哪种CPU性能最好，在TOM硬件和ANANDTECH中浏览一下，在那里你将发现大量目前的和过时的CPU，主板和芯片组。）

显然如果预算允许，你应该选择强大的CPU。然而别急着买最贵的CPU。要时刻考虑你的系统需求，F1赛车法拉力在上下班高峰期的路段显然是不合适的。

为了对我们平台的建设提供一份很好的建议，我们选择定义了三种不同级别的Asterisk系统：小型、中型、大型。

小型：小型系统（10部以内的电话）不需要考虑Asterisk的性能需求，放在小系统上的典型的负载将处于现代处理器的能力之内。

如果你正在用已经存在的老部件构建一个小型系统，注意系统在最后不要被指望成为一个更强的系统。在更高负载下，性能将严重下降。性能很差的硬件可以成功运行一个业余系统，虽然对于所有对Linux性能调整不善长的人并不推荐使用。

如果你只是希望建立一个Asterisk系统只是为了学习和研究，你可以用比较差的CPU建立一个全功能的系统。作者运行了好几个Asterisk实验室系统，用433MHZ到700MHZ的赛扬处理器。通常这类系统的工作负载是很小的。

如果你希望弄得非常清楚哪种CPU性能最好，在TOM硬件和ANANDTECH

中浏览一下，在那里你将发现大量得目前的和过时的CPU，主板和芯片组。

大家所熟知的133MHZ奔腾系统可以运行Asterisk，但是会有性能问题。对这样的系统做正确的配置要求有Linux专家那样的知识。 不建议在低于500-MHz的系统上运行 Asterisk(对于生产系统，2 GHz可能是最低的配置)，但是Asterisk是非常灵活的！

中型：中等系统（10到50部电话）是对系统的性能考验最严峻的。一般来说，这些系统将运行在一个或者两个服务器上。因此每台机器将被需要处理多于1个的特定任务。随着负载增加，平台的限制将变得越来越严峻。用户可能开始观察到质量问题，而没有认识到系统并没有崩溃，只是稍微超过系统的容量。如果负载再不断增加，这些问题将不断恶化，用户的音质也会越来越下降。在性能问题没有被用户充分认识以前是很难识别和描述的，这种情况非常严重。

在这些系统上监测性能并快速处理任何正在发展的趋势是确保电话平台质量的一个关键。

大型：大的系统（超过50电话用户）可以分布在多个核心上，因此性能相关的内容能够通过增加服务器而得到管理。非常大的Asterisk系统，从500到1000用户，已经用这种方式建立起来。建立一个大的系统需要一个高水平的多学科的高级知识。本书中我们不讨论这些细节，也就是说，你遇到的问题和那些遇到多个服务器处理同一个分布任务的问题很相似。

主机板的选择

不要有任何的企望，本书我们不会给您推荐某类主机板。现在几乎每周都有新的主板面世，所以推荐的任何主板技术上都不是最新的。不仅如此，就像汽车一样：设计原则上都很类似，细节上区别却很大。Asterisk系统的性能表现往往取决与此。

我们更想做的是给您提供一些建议，以便与Asterisk系统可以运行的更好，以及好的主板的一些特点。高性能及稳定性的关键点，下面是一些相关的指导：

各种系统总线必须提供最小的反映周期。如果你正计划用一个PSTN连接使用模拟器件或PRI接口（后面讲），Digium Zaptel卡每秒产生1000个中断请求。接

口上总线上使用这些器件会降低请求响应的能力。Intel芯片组(for Intel CPU)和nVidia nForce(for AMD CPU)在这个领域目前达到了很高的标准。看看你正在评估主板的芯片组确保它们在中断响应时间上没什么大问题。

如果系统上使用Zaptel卡，你应该知道BIOS能够支持最大的中断分配数。一般的原则是高端的主板通过设置BIOS来提供非常大的灵活性。值钱的主板一般不用控制太多，这点可能有些争论，然而，一个APIC使能的主板是由操作系统来跳中断的。

服务器级别的主板与工作站级的主板在PCI总线实现上也不尽相同。实际上有很多不同之处，最明显和广为人知的是不同的版本电压都不同。根据你购买的主板，你是需要3.3V还是5.0V的插槽。下图给出了3.3V与5.0V之间的不同之处。多数服务器有这两种的电压，工作站只有5.0V。

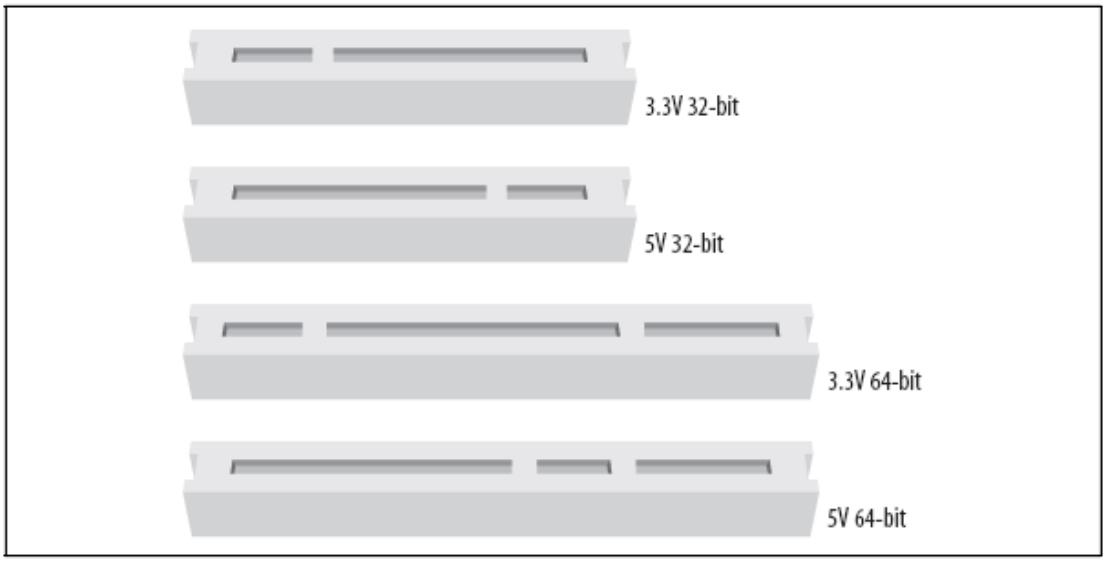


图2-1　PCI槽位的可视标识

考虑到多处理器的情况。处理多任务对于提高系统的稳定性会有很大的提高。在Asterisk系统中进行浮点运算特别有好处。

避免主板上集成声卡和显卡。如果您需要，单独装一个就可以了。至于显卡，也许你根本就不需要，因为Asterisk系统根本就不用。传统上讲高性价比的主板都集成了这些组件，这些板子在设计上都采取折中的方法降低花消。

如果可能的话在装一个外置MODEM。如果你非要装一个内置MODEM，要确保不是所谓的“WIN-MODEM”，必须是完全自主的单元（了解这个比较难，想知道更多自己看书）。

考虑集成的网卡，如果你的一个网络设备故障，整个主板都要更换。不过从另外一方面考虑，如果你安装一个网卡（NIC），考虑到硬件之间的连接，也许会增加故障率。一些性能参数通过主板上集成两块网卡（一主、一备）可以得到提升。

Asterisk系统的稳定性及质量取决于所选择主板的结构设计。Asterisk系统是一个猛兽，需要很好的喂养！其实任何事情都如此，最贵的不见得就是最好的，所以你需要成为一个计算机结构的专家。

已经说了那么多，我们回到开始：Asterisk系统可以轻松安装在几乎所有的PC平台上。比如我们写这本书，几乎包含了所的从奔腾三433-MHZ的INTEL芯片组到速龙XP2000的主板。在低于五个同时电话连接时我们没有遇到任何性能或稳定性的问题。为了学习Asterisk系统，别怕在你的任何系统上安装。当你把你的系统做成产品时，你就需要对自己每一个选择都有所了解。

电源要求

你会经常看到PC机的电源。对于电话系统，这些部件有着关键的作用。

计算机的供电

为系统所选择的电源在整个的系统稳定性中扮演着关键的角色。Asterisk不是一个大功率的系统，但是任何与多媒体相关的应用（电话、专业音响、视频等等）对电源的质量都十分敏感。这些易被忽视的组件可以把高质量的系统搞的很差。同样一个信令，好电源可以可以使它表现出众，而差的电源会将它丢弃。

给系统供电不仅要提供足够的功率使之完成任务，还要提供系统所需的稳定、清晰的信号。

冗余电源

在一个多级载波或高可靠性的环境里，一般都有冗余电源。本质上，这是连接了两个完全独立的供电系统，任何一个都能够提供满足系统要求的供电。

如果供电对系统很重要，记住最佳实践的建议就是具有适当的冗余，这些电源都要连到完全独立的UPS，每个UPS都由不同的供电线路供电。

2.2 运行环境

构建系统工作环境的不仅仅是服务器本身的部件，还有其它因素。这些因素却在系统可靠性和质量方面起到至关重要的作用。电力供应、房间温度和灰尘、干扰源、安全等等问题都要之前考虑。

电源条件以及 UPS

当给系统选择电源时，考虑的不仅仅是系统需要使用的电量，还要考虑电源提供的方式。电力并不简单地是来自墙上插座的电压，永远不要把生产系统随意插入电源插座（如果随意接插电源，可以工作，但是系统会有莫名奇妙的稳定性问题。再读一读本节的内容，好不好？）。对系统供电给与充分的考虑能够提供更加稳定的供电环境，从而使得系统更加稳定。一定要考虑逻辑地，这一点是不容忽视的。

适当接地，利用稳压器提供高质量的电力供应能够确保系统有干净的逻辑地（就是0V电压）参考电压（在电子设备上，二进制0一般用 0V信号表示，而二进制1则由很多不同的电压表示(通常在2.5和5V之间)。系统当作0V的参考地通常称为逻辑地。接地不良的系统可能在逻辑地上有这样的电势，导致二进制0和1的错误。这会严重影响系统处理指令的稳定性），并能使主板上的电子噪声降低到最小。这些是这种类型设备的业界标准最佳实践，不容忽视。达到这个目标的一个相对简单的办法是使用UPS（通常会误认为UPS提供的是干净的电源，但是实际上并不总是如此。）。

具有电源调节功能的 UPS

UPS作为电池备份是广为人知的，但是高端UPS单元才具有的电源调节功能所带来的好处却鲜为人知。

电源调节功能可以通过隔离变压器再生洁净电源，起到保护作用。UPS中的高质量电源调节器会去除电力供应的大部分电子噪声并向你的系统提供稳定的电源。

遗憾的是，不是所有的UPS都是这样的。许多便宜的不提供洁净电源。更可

怕的是，制造商经常承诺他们的产品可以提供各种保护：电涌、峰值电压、瞬变等，尽管这种类型的产品可以保护你的系统不受电子风暴的影响，但是并不能让给你提供能量的电源系统洁净，因此对你的系统稳定性没有任何帮助。

确认你的UPS是具有电源调节功能的。如果厂家说的不确定，那么就不是。

接地

电压的定义是两点之间电势的差。当考虑一个是地（就是一个通往大地的电气通路），通常假定它代表“0V”。如果没有定义相对于什么的0V，那么就无法定义电压的数字了。如果你在两个地之间测量电压，你将经常发现在这两者之间还有电势差，这两个之间的差很容易造成逻辑错误——甚至系统崩溃——在很多系统里面，甚至在有多个接地的系统里会产生损坏。

 作者之一回忆起，曾经烧坏过声卡，当时要把声卡连接到朋友的立体声系统上。即便是计算机和立体声音响都在一个房间内，所使用得两个电器插座的接地导体竟然有6V的电压差。连接立体声和PC的信号线提供了电压渴望的通路，于是烧坏了声卡（信号电流过大）。PC和立体声系统使用相同的插座就解决了问题。

在考虑电气规定的时候，接地的目的主要是为了人身安全。在计算机上，接地使用0V的逻辑参考电压。提供适当安全性的电气系统并不总是提供了适当的逻辑参考地。事实上，安全和电源质量的目标不一定是一致的。当然，在必须做选择的时候，安全要放在第一位。

 由于在计算机中代表二进制0和1的电压之间的差有时小于3V，这完全可能是因为不稳定的电源条件所导致的，包括接地不良或者导致系统断断续续续出现问题的电气噪声。一些电源和接地的提倡者估计80%以上的无法解释的计算机故障都与电源质量有关。

现代的开关式电源在某种程度上能够隔离电源质量的问题，但是高性能系统仍会从设计良好的电力环境中获得好处。对于大型主机、专有PBX和其他昂贵的计算平台，远远不会在接地方面留下人和隐患。这些系统的电气设备总是提供有专门的接地，它们不依赖于供电所提供的安全地。

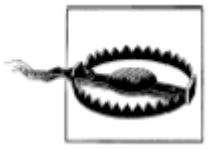
不管你愿意在接地上投资多少，当给PBX供电时，一定要确保电源电路是完全专用的（如下节所讨论的），并提供了绝缘的独立接地。这可能是一个昂贵的设施，但是对系统的电源环境的质量会非常有利（在业余系统上，这样的要求可能过高。但是如果打算把Asterisk用于处理重要的事情，至少要确保它不受影响——不要把空调、复印机、激光打印机，发动机等和Asterisk共用电源电路。）。

与系统连接的每一个外围设备也应该与系统使用同一个电源插座(或者更确切地说，使用共同的参考地)，这一点非常的重要！这样做能够避免出现接地回路，接地回路会产生吱吱声和嗡嗡的噪音，甚至会损毁设备。

电路

你一定有这样的经历，当一个用电器接入进来，灯就变得昏暗了。你也应该经历过，当高耗能的设备接入电路的时候的效果。如果你看到这些设备的效果，每个电源都有自己的方式，你会发现谐振在50－60HZ的正弦波在电路中是特别普通的。你可能觉得你获得电源的方式很多。但是谐振波噪音在电路上也是常见的，在敏感的电子设备上，是会产生可怕效果的。对于PBX，这些问题可以造成语音不清晰，逻辑错误和系统不稳定。

不要把服务器放置在和别的设备共享电源的位置上，电路上最好只有一个插座，你只能安装你的电话系统以及外设在一个插座上。电线（包括接地）必须直接连接在接线板上。接地是独立的。要远离空调等设备。



所在地区的电气规定必须先于此处的任何说法。如有疑问，请咨询电源质量专家如何确保遵守当地规定。记住，电器规定主要考虑人身安全，其次才是设备安全。

设备机房

环境状态可以削弱你系统的稳定性，也许你会看到很多精密的系统根本没有关注过这类问题。如果你查看统计，很明显，关注环境参数可以让系统更加稳定可靠。

1 湿度

简单地说，湿度是空气中的水含量。对于电子器件来说，水可以带来灾难。原因有两个：水含有杂质，水是导体，可以短路电子设备。不要在任何湿度高而没有采取除湿措施的的地方安装设备。

2 温度

热量是电子器件的敌人。你的系统温度越低，系统越可靠。如果你不能提供一个空调房间给你的系统，也必须提供一个稳定提供清洁冷空气的地方。同时，让温度恒定。温度的变化可以造成水汽凝结，致使系统不稳定，甚至造成其它破坏。

3 灰尘

在计算机行业，有一种陈旧的观念，说布满灰尘的计算机是幸运的。让我们来看看布满灰尘的现实情况如何：

- 大量的灰尘让空气无法正常流通，造成热度增加；
- 灰尘含有金属部分，如果具有充足的量，会导致信号衰减或者电路板短路。

让你的系统在一个过滤过的环境，不要让灰尘造成系统崩溃。

安全

服务器安全自然包含防范网络产生的一些东西，但是环境也是系统安全的一个重要部分。电话设备必须经常锁定，只有需要访问的人才能有权接近它。

2.3　电话硬件

如果准备把Asterisk系统连接到任何传统电信设备上去，需要有合适的硬件，所需硬件由希望达到的目标来决定。

连接到 PSTN

Asterisk允许电路交换的电信网络设备（由于在PSTN上采用时分复用的方式来承载话务，因此常被称作TDM网络。）和包交换的数据网络设备（通俗的被称为VoIP网络，尽管Voice over IP不是在包交换网络上传输语音的唯一办法（1990年代，Voice over Frame Relay非常流行）。由于Asterisk具有开放式的体系架构（以及开放的源代码），使它连接任何标准的接口硬件都是可能的。开源电话接口卡的选择目前是受限的，但是随着对Asterisk兴趣的不断增长，它将很快改变。到那时候，最流行和最经济的连接PSTN的方法是使用接口卡，这在Zapata电话系统有所涉及（http://www.zapatatelephony.org）。

1　模拟接口卡

除非需要很多通道（或者每月有许多钱可以花在电信设施上），否则PSTN接口会由一个或者多个模拟电路组成，每个电路需要一个FXO接口。

Digium是Asterisk开发的赞助者，为Asterisk生产最流行的模拟接口卡——TDM400P（事实上，TDM400P根本不是时分复用卡，它是一个模拟卡）。TDM400P是4口母卡，可以插最多4块子卡，既可以提供FXO，也可以提供FXS口。购买TDM400P的时候可以预装子卡，并且Digium制定了专门的部件号码来描述配置情况。命名规则是TDM x y B，其中x和y分别代表母卡上FXS和FXO（ FXS 和FXO指的得是模拟电路相对的两端。根据要连接到的对象来确定需要的是FXS还是FXO。第7章会详细讨论这个问题）卡的数量。查看Digium的网站（http://www.digium.com）可以得到更多有关该卡的信息。

另有一种名为X100P的卡，适合初学者用做学习或开发测试。

另外一些制造商也在设计Asterisk的兼容模拟卡。如有需求，可以联系QQ群20556356，你将获得相关信息，也可以联系jilly_yang@263.net，或者到

www.asterisker.cn。

2　数字接口卡

如果需要的电路多于10条，或者需要数字连接，应该到市场上寻找T1或者E1卡。但是要注意，数字PSTN电路每个月的费用差异很大。在某些地区，5条电路可以使用一个数字电路；但是在另外一些地方，技术还没有那么值钱。所在地区的竞争越激烈，就越能找到更便宜的电路。

Digium制造了若干不同的数字电路接口卡。这些卡的特性是一样的。主要区别在于是提供T1还是E1，以及每块卡提供的接口数量。虽然技术上可行，从Asterisk社区的多数人的意见是在一个系统中不要部署多于1块的这类接口卡。

Sangoma的卡包含FPGA，这使得这些卡具有极强的灵活性。比如，在Asterisk环境里面，他们已经进行了编程，可以与Zapata通道的驱动进行接口。

联系QQ群20556356的china*，你将获得相关信息，也可以联系jilly_yang@263.net，或者到www.asterisker.cn。

3　信道复用器

一个信道复用器是一种设备，它允许一个数字电路被解复用到多个模拟电路（或者相反）。更确切地说，一个信道复用器让你通过T1/E1线路连接模拟电话和线路到一个系统。Figure2-2显示了一个信道复用器应用于一个典型的办公电话系统。

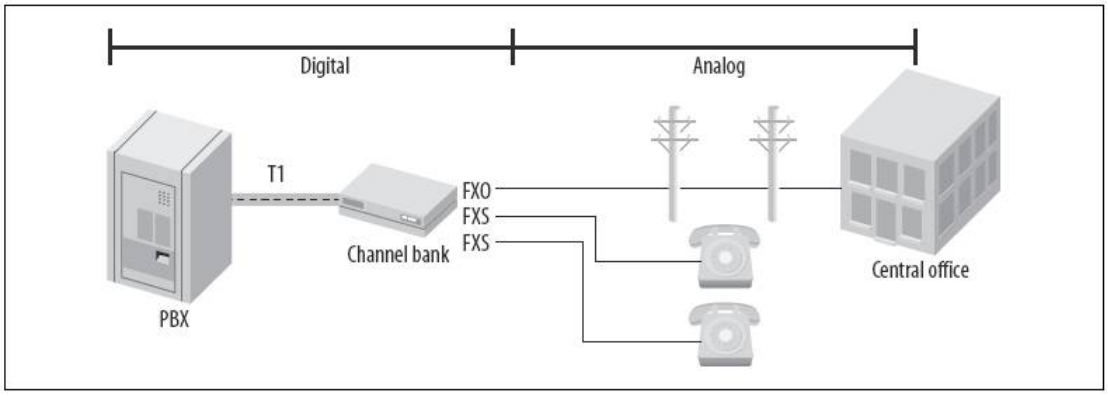


图2-2　信道复用器

虽然他们价格昂贵，许多人感觉只有通过信道复用器这一种方式才能集成模

拟电路和设备到Asterisk。

4 其它类型的 PSTN 接口

许多VOIP的网关可以配置为提供到PSTN电路的访问。总之，这些大部分都应用于一个小的系统里（一根或者两根电话线）。他们也可能配置非常复杂，在各种网络和设备之间的互操作，需要对电话系统和VOIP技术基础都有非常深入的理解。因此，本书我们不讨论这些设备的细节。但是这些是值得研究的，最流行的设备一般由Sipura、Grandstream、Digium和许多其它公司提供。

另外一个连接到PSTN的方法是通过BRI ISDN电路。BRI是一个数字电信标准，它指定两个信道来承载144kbps的流量。在北美以及世界上其它国家，BRI很少使用。但是在欧洲，BRI特别流行。由于这个技术采用各种不同的方式实施，我们将在本书不细节讨论BRI技术。

专线连接到包交换的电话网络

如果你不需要连接PSTN，Asterisk不需要除了NIC以外的硬件设备。

然而，如果你准备提供音乐保持或者会议，并且你没有物理时钟源，你将需要ztdummy这个Linux内核模块。Ztdummy是一个时钟机制，在不需要硬件时钟源支持的情况下提供一个时钟源。在Linux内核2.4版本，使用ztdummy， 你必须在主板上有一个UHCI类型的USB控制器。在Linux 2.6，就不需要这个了。

2.4 电话类型

既然本书的题目叫做《Asterisk: 电话未来之路》，如果我们不能忘了，我们还没有讨论一下所有电话系统最终需要连接的设备——电话呢！

我们都知道电话，但是五年后还是这样吗？Asterisk所致力的部分革命，就是电话机的变革，从一个简单的语音通信设备，到一个多媒体通信终端，并能够提供所有能够想象出的功能。

作为已经存在的概念的介绍，我们主要讨论我们目前称之为“电话机”的各

种终端设备。任何这种设备可以被轻松地集成到Asterisk里面。我们也将探讨一些涉及到哪些设备可能演进到未来的电话系统中（仍然可以轻松地被集成到Asterisk里面）。

物理电话

任何物理设备，如果它最主要的作用是在两个点之间终结语音通信电路的，这个设备叫做电话机。至少，这样的设备需要一个手柄和一个拨号盘，它可能也需要一个特征按键，一个显示屏和各种语音接口。

这个部分让我们大概看一下你可能要连接你的Asterisk系统的各种用户设备。我们将在第七章详细讲解模拟和数字电话系统的一些机理。

模拟电话

模拟电话自从电话问世就诞生了。直到20年以前，所有的电话都是模拟电话。虽然模拟电话在不同的国家具有不同的技术制式，但是他们都遵循类似的规则。

当人类说话，一个单词，发音，声调，或者唇音，将产生一个复杂的声音效果。电话的目的就是来获取这些声音并将他们转化为适合线缆传输的格式。在模拟电话上，传输的信号是模拟的，声音波形是直接由人的发声产生的。如果你能够看到声音波形从嘴到麦克风传送的过程，他们将成为带你子信号的一部分，你能在线缆上都可以测量。

 这个连续的连接一般被称作“电路”，电路被用来使用电子机械交换产生。因此被称为“电路交换网络”。

模拟电话是唯一的能够在电子市场看到的电话机。在未来的若干年，他们将被戏剧性地改变。

专用的数字电话

数字电路交换系统在20世纪80和90年代告诉发展，电信公司发展数字PBX和

KTS技术。在这些系统上发展起来的专用电话，但是他们必须依托于这些系统。也就是说，他们不能连接到其它系统上去。甚至电话只能被和系统相同的厂家来制造，而不能互相兼容。（比如，北电的Norstar终端不能和北电Meridian 1PBX互通）。这种专用的数字电话的特性限制了他们的未来。在这个基于标准通信的时代，他们很快被历史扔到了垃圾箱里。

ISDN 电话

先于VOIP，基于标准的数字电话是ISDN-BRI终端。在二十世纪80年代早期，ISDN被期盼成为电信业的革命性技术，就像VOIP之于今天大家的设想一样。

 有两种ISDN类型：PRI和BRI。PRI一般用作PBX和PSTN之间的中继，并被广泛使用。而BRI在北美并不流行，但是在欧洲很成功。

ISDN既然被电话公司广泛使用，许多人把它看作是一个标准，但是并没有象人们预测的那样发扬光大。实施的高额代价，在主流玩家中缺乏配合，导致了它的问题多到无法解决的地步。

BRI倾向于服务终端设备和小端点（一个BRI环路提供两路数字电路）。一个BRI设备，BRI已经被几种速度更快而便宜的技术——ADSL、cable猫和VOIP技术所取代。

BRI在会议电视设备中仍然被广泛使用，因为它提供固定带宽的链接。同时，BRI没有服务质量的类型，而VOIP有。原因是BRI是基于电路交换的。

BRI有时候被用来在模拟电路中提供中继，不管这个方法是否好，它都取决于你本地电话公司的服务价格，以及它给你提供什么特点的服务。

IP 话机

IP话机在电信业的出现是令人振奋的变化。在不久的未来，基于标准的IP电话将在零售市场见到。这个可能性的价值来自于这些设备将带来各种增值应用的大爆炸。从视频电话，到高频广播服务，到无线移动解决方案，到基于目的的特

定工业，以及多合一的多媒体系统的柔性。

IP话机的革命性还表现在不是一种新的类型的线缆连接到你的话机上，并可以给你任何通话能力。

早期的IP话机在多年前已经出现，但那并不代表哪些令人激动的应用的未来。他们是一个里程碑，一个相似的包裹，让人产生无限遐想。

未来和我们想象的一定不完全相同。

软电话

软电话是一个软件程序，在一台非电话设备上提供电话功能，比如在PC和PAD上。因此，如何识别这样一只“猛兽”？乍看起来就是一个简单的问题，但实际上问题很多。软电话应该有某种形式的拨号盘，并提供一个接口来提醒用户这是一个电话。但是是不是总是这样的？

预计“软电话”一词会快速演进，因为电话说到底是什么个概念，都正在发生革命性变异。下面来看这种演进的一个例子：我们是不是要正确定义即时通信软件为软电话？IM提供了初始化和接收基于标准VOIP连接的能力。难道这个和软电话的要求不一致吗？回答这个问题需要我们未来没有处理过的未来的知识。说实话，在这个问题上，软电话被期待为和传统电话具有相同功能，这个概念将在不久的将来迅速改变。

随着标准的演进和我们从传送电话到多媒体通信文化的迁移，软电话和物理电话之间的分界线将变得很模糊。比如，我们可能购买一个通信终端当作电话，并安装一个软电话程序来提供我们需要的功能。

呵呵，这样分析，把水给搅浑了。在这一点上，最好我们能够定义词语“软电话”来特指本书中有关的软电话，并理解这个词的意思在以后的几年还能经历一个大的变化。出于我们的目的，我们定义软电话为：任何运行在个人电脑上，外观和感觉是一个电话，提供通过E.164（E.164 是ITU标准。它定义了电话号码的分配方式。电话系统使用的就是E.164编址。）寻址接收全双工音频通信的能力（就是我们常说的电话呼叫）。

电话适配器

电话适配器（经常指ATA或者模拟终端适配器）能够被大概地描述为一个最终用户设备，通过它，从一个协议到另外一个协议转换通信电路。最常用的，这些设备被用来转换数字信号到模拟信号，这样，你就可以插入一个标准的电话或者传真机了。

这些适配器根据它的功能，可以被描述为“网关”。然而，流行的“电话网关”的是最恰当地描述一个多口电话适配器，一般带有复杂的路由功能。

只要有连接非通用标准的设备的需求，或者有将老设备连接到新的设备上的需求，电话适配器将在相当长时间存在。最终，我们对这些设备的依赖将会消失，如同当初依赖于MODEM——通过不断的改进而被淘汰。

通讯终端

通信终端是一个老词了，在一、二十年前消失了，但在这里将会重新被介绍，不是为别的，而是为了通过我们的探讨而使其再次消失。

首先，一段小历史。当数字PBX系统被率先发布的时候，这些设备的制造商意识到他们不能让他们的终端作为一个电话，他们天然的专利阻止了他们连接到PSTN去。他们因此被称为终端，或者“站”。当然，用户并没有任何一款设备。它看上去象电话而实际也作为一部电话使用，因此它“是”一个电话。你偶然也会发现PBX设备是指这种终端，但是大部分他们被称为“电话”。

翻新了的通信终端的相关性和专利没有任何关系，并且正好相反。我们发展更多创造性的相互通信的方式，我们访问允许我们连接的许多不同设备。考虑下列场景：

- 如果我使用我的PDA来连接我的语音邮箱，并获取我的语音信息（并将其转化为文本），那么我的PDA就成了一个电话吗？
- 如果我在PC上连接了一个视频照相机，连接一个公司的网站，并请求了一个和客户服务中心的聊天直播，我的PC就成了电话了吗？
- 如果我爱我的厨房使用了IP电话来浏览我的食谱，这是一个电话呼叫吗？

这一点其实很简单：我们可能总是在互相“通话”，但是我们总使用“电话”吗？

2.5 Linux 的考虑

如果你问自由软件基金会的任何人，他们会告诉你我们所说的Linux实际上是Linux/GNU。抛开用词上的争论，这种说法确实有一些有价值的事实。操作系统的内核确是Linux，主要的应用安装在Linux系统，使用上是GNU的应用。Linux的5%是Linux，75%是GNU，20%可能是其它的东西。

为什么讨论这些呢？Linux的灵活性既是好事也是坏事。好事是因为你确实可以根据源码编写自己的操作系统。既然没有什么人这么做，坏事就占了上风，你必须自己确定装哪些GNU功能，如何配置系统。

如果这些看起来很难办，别怕，下一节，我们会讨论选项、安装、配置Asterisk软件环境。

2.6 结束语

本章，我们讨论了可能影响Asterisk系统安装性能和稳定性的所有问题。在吓跑你之前，我要告诉你许多人已经安装Asterisk系统在顶级的Linux图形工作站上，跑WEB服务、数据库、X窗口环境，谁知道还有什么，根本没任何问题。本章提示你做最好的实践，你付出的时间和精力与你对Asterisk系统服务器性能、质量、可靠性的期望成正比。

这一章我们想给你一个感觉，最好的实践会帮助你建立一个可靠、稳定的Asterisk平台，Asterisk系统。Asterisk非常愿意工作在恶劣的条件下，但是您决定所付出思考和努力会使您的PBX更加稳定。您的决定完全取决于您的Asterisk系统有多重要。

第3章　安装 Asterisk

我渴望完成一项伟大而高尚的任务，但是我所做的和我心中理想伟大而高尚的任务相比都不值一提。世界是在前进的，不仅仅是由领袖和英雄推动的，而是由每一位最忠实的工作者以很小的积累来推动的。

——海伦.科尔

在前一章，我们讨论了如何准备一个系统来安装 Asterisk。现在是时候来获取，解压，编译和安装软件了。

虽然有很多的 LINUX 种类和 PC 体系结构都是非常好的选择，但是我们要选择一种以主张简短和清晰来贯穿的。说明文档应该尽可能的通俗易懂，但是我们可以着重偏好 Red Hat 的结构和有效性。我选择 Red Hat 是因为它的命令集、目录结构和大多数用户都很熟悉它（我们找到了许多 Red Hat 管理员熟悉的，即使他们不喜欢它。）。然而，这并不意味着 Red Hat 就是唯一的选择，或许有更适合你的。有一个问题经常出现在电子邮件答复清单中：“哪种 LINUX 最适合使用 Asterisk？”多数的回答都是“根据你的爱好。”

3.1　我们需要什么包？

Asterisk 主要使用三个包：Asterisk 主程序（Asterisk），Zapata 电话驱动（zaptel），和 PRI 库（libpri）。如果你打算搭建一个纯 VoIP 网络，那么只需要 Asterisk 包；如果你使用 ztdummy 驱动（将在以后的章节讨论）做为时间接口，除非你正在使用 ISDN PRI 接口，否则 libpri 库是可选择的技术，同时如果你不加载它的话，你需要保存一些 RAM，但是我们建议安装所有与 zaptel 联系的包。

另一个你可能要安装的包就是 asterisk-sounds，当 Asterisk 接收到分布在各地的用户的呼叫时，asterisk-sounds 包会给出一个适合的声音提示。如果你想为你的 Asterisk 系统提供许多扩展的专业提示，那么它就是必须的。后面几章的许多例子都使用了这个包中的文件，所以我们估计你是已经安装了的。

必须的包

编译 Asterisk，你必须安装 GCC 编译器（3.X 版本或者更高）以及附属包。要是你使用的是 GCC 2.96 那么以后的版本将得不到支持。Asterisk 同样需要 bison，一个取代 yacc 的解析器，和一个 CLI。Cryptographic 库在 Asterisk 中需要 OpenSSL 和它的开发包。如果你要实时使用 ztdummy，或者任何一个由 Zaptel 提供的硬件驱动，你也要安装 zaptel 包。如果你正在安装 libpri，请确认要安装在 Asterisk 之前。（参考“编译 libpri”）

为了安装 zttool 程序，Zaptel 需要 libnewt 和它的开发包（参考“使用 ztcfg 和 zttool”），还有，为了安装 ztdummy 需要 usb-uhci 模块。如果你在使用 PRI 接口，Zaptel 同样需要 libpri 包（同样，即使你没有使用 PRI 电路，我们建议你先安装 libpri 再安装 zaptel）。

下面的部分我将讨论如何得到，解压，编译和安装 Asterisk，zaptel, libpri, 和 asterisk-sounds 包。

3.2 获得源代码

Asterisk 原代码既可以通过 FTP 也可以通过 CVS 得到。我们将演示如何通过两种途径得到原代码。，虽然你只需要通过其中一种方法得到原代码（FTP 是比较好的方法）。

从 FTP 得到 Asterisk 原代码

Asterisk 原代码可以从 Digium FTP 服务器上得到，地址是 ftp://ftp.digium.com.。最容易的方法就是通过使用 wget 程序得到稳定的发行版。

Stable and Head

Asterisk 有两种不同的样式，一般是稳定型和引导型。稳定型是一种基于 Asterisk 分支，它常用于生产系统。引导型分支则是研发员用来测试新特性和调试错误的。

小缺陷将在合理的测试之后被稳定的部分合并。开发那一块在测试的某个时候出现问题是完全可能的，因此，稳定的部分就是你的产品系统将要运行的平台，也是贯穿这本书的内容。

你可以通过 FTP 得到稳定的发行版本。Asterisk 的两种类型都可以通过 CVS 得到，这在以后的章节有说明。然而，请注意区分从 CVS 得到的版本和发行版本之间的区别。发行版只是 CVS 树中的一个简单映像，要是稳定可以发行了，便会有一个版本号并在 FTP 服务器上提供下载。注意，通过 CVS 得到的原代码可能有很多的错误（比如根本不是很稳定），从 FTP 上下载的才是真正稳定的。

总的来说，通过 FTP 服务器用户可以得到稳定的发行版本。

注意，你将在/usr/src/目录下解压和编译 Asterisk 原代码。同时也要注意到只有使用 root 管理员才具有对/usr/src/目录的写权限和安装 Asterisk 以及与它相关联的包。

为了得到最新的稳定版本原代码，可以输入下面的命令行：

 # cd /usr/src/

 # wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-1.*.tar.gz

 # wget --passive-ftp ftp.digium.com/pub/asterisk/asterisk-sounds-*.tar.gz

 # wget --passive-ftp ftp.digium.com/pub/zaptel/zaptel-*.tar.gz

 # wget --passive-ftp ftp.digium.com/pub/libpri/libpri-*.tar.gz

只要 Digium 不改变上传 FTP 站点的方式，wget 命令行将自动得到最新的版本。你也可以使用有效的软件版本取代通配符（*）。

你已经得到了 Asterisk 文件和 Digium 硬件，现在准备解压原代码。

解压原代码

如果你使用 wget 从 FTP 服务器上得到原代码，在编译之前你首先需要解压。如果你没有下载包到/usr/src/，要不马上将包拷贝到那里，要不就指定一个完全路径到那里。我们将使用 GNU tar 应用程序从压缩文件中解压得到原代码。下面是一些例子，通过下面的的这些命令可以处理这些压缩文件：

 # cd /usr/src/

 # tar zxvf zaptel-*.tar.gz

 # tar zxvf libpri-*.tar.gz

 # tar zxvf asterisk-*.tar.gz

 # tar zxvf asterisk-sounds*.tar.gz

这些命令将解压各个包到相应的目录中。

通过 CVS 得到 Asterisk 原代码

版本控制系统是一个可以提供中心仓库功能的工具，它可以为一个大型的开发团队管理项目开发过程中繁多的相关文件。一旦出现了修改，它就会提交到 CVS 服务器上，在服务器上可以直接下载和编辑。使用 CVS 的附加功能可以将一个特定的文件回滚到特定的状态，要是文件被损坏，你可以很容易的恢复到一个可运行的版本。要是你发现安装最新的 Asterisk 引起了系统的某一部分损坏，你可以及时回滚到一个更早的还原点同时研究这个问题。

如果你是一个开发者想得到最新的更新，你需要通过 CVS 服务器。你通过 CVS 可以下载稳定的版本：

- 输出 CVSROOT 路径:

  # cd /usr/src/

  # export

  CVSROOT=:pserver:anoncvs:anoncvs@cvs.digium.com:/usr/cvsroot

- 从 CVS 下载头

  # cvs checkout zaptel libpri asterisk

- 从 CVS 下载 STABLE 1.0:

  # cvs checkout -r v1-0 zaptel libpri asterisk

- 从 CVS 下载 STABLE 1.2:

  # cvs checkout -r v1-2 zaptel libpri asterisk

- 从 CVS 下载可选择模块:

  # cvs checkout asterisk-sounds asterisk-addons

注意，能够从 CVS 上得到的稳定分枝并不是一个发布版本，也不应该被应用到生产系统中。

3.3　编译 Zaptel

图 3－1 显示了 Asterisk 和 LINUX 内核关于硬件控制之间各层接口之间的关系。在 Asterisk 一方是 Zapata 信道模块，chan_zap。Asterisk 使用这些接口与 LINUX 内核通信，而 LINUX 已经加载了硬件驱动。

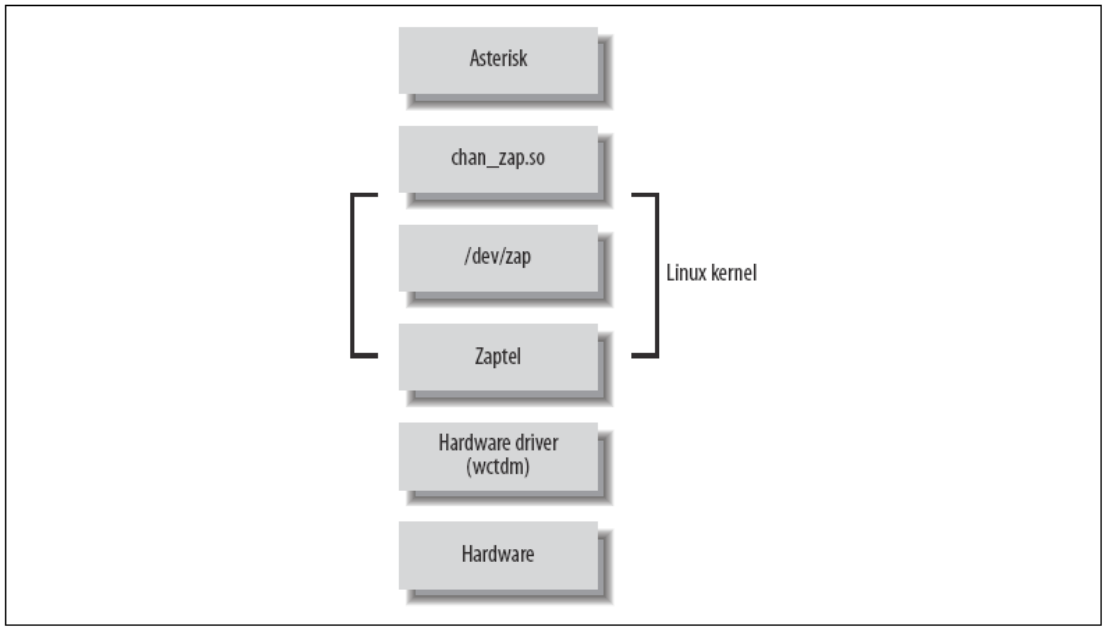


图 3-1　设备分层结构与 Asterisk 的关联

Zaptel 接口是一个内核的可加载模块。这个接口位于硬件设备和 Zapata 模块之间的抽象层。基于这个理论模式，在不影响 Asterisk 原代码的情况下，驱动设备可以更改。驱动设备一般通过 Zaptel 与硬件直接通信。

当 Asterisk 在不同的平台上编译的时候，Zaptel 驱动程序直接在 Linux 内核写入一个接口。虽然正在为 FreeBSD 上编写，但是现在在其他的操作系统上还没有正式的 Zaptel 驱动程序。

我们将马上在 zconfig.h 文件中讨论 Zaptel 的编译时选项。首先，让我们将注意力放在编译和安装驱动上。（Zaptel 驱动的配置将在下一章讨论）

在 Linux 2.4 内核上编译 Zaptel 驱动之前，你要确认在/usr/src/目录中包含了一个名为 linux-2.4 的连接指向你的内核代码。如果这个连接不存在，你可以用下面这个命令来创建（假定你的安装目录是/usr/src）：

 # ln -s /usr/src/'uname -r' /usr/src/linux-2.4

在各种基于 linux-2.6 的内核上不需要创建这个连接，它自动在编译的时候

会直接寻找内核。然而，你将编译目录放到不是一个标准的地方（例如，像/lib/moudules/内核版本号/build/这样的地方），那么你就需要一个连接。

ztdummy 驱动

在 Asterisk 中，某些应用程序和一些特别的命令的运行需要一个分时设备（如果 Asterisk 没找到这个设备，那么编译是不可能通过的。）。所有的 DigiumPCI 硬件头提供一个 1KHz 的接口。如果你的 PCI 设备不具有分时的特性，那么 Zedummy 驱动程序就可以做为分时硬件的替代品。在基于 2.4 内核的各种 Linux 中，ztdummy 必须使用由 UHCI USB 控制器提供的时钟。驱动程序加载 usb-uhci 模块的时候要求内核版本至少为 2.4.5 以上。

在一个基于 2.6 内核的系统上，ztdummy 不需要使用 USB 控制器。（像 2.6.0 的内核的驱动程序提供 1-kHz 的接口；这样，就不需要 USB 控制器这样一个硬件设备了。）

在 Makefile 文件中默认是不创建 ztdummy 的。要编译 ztdummy 的话，你必须在 Makefile 文件中删除所有相关的注释。找一个你喜欢的文件编辑器，找到下面这行：

MODULES=zaptel tor2 torisa wcusb wcfxo wctdm \

ztdynamic ztd-eth wct1xxp wct4xxp wcte11xp # ztdummy

删除所有在 ztdummy 之前的标号（#），保存文件，就可以编译 Zaptel 了。

Zapata 电话驱动

在 2.4 或者 2.6 的 Linux 的内核中直接运行 make 来编译 Zapata 电话驱动来使用 Digium 的硬件设备（Makefile 文件会自动检测你内核的版本）。使用以下命令来编译 Zaptel(用你使用的 zaptel 的版本号来代替 version)

cd /usr/src/zaptel-*version*

make clean

make

make install

每次在你重新编译任何模块的时候都运行 make clean 是很必要的，它将删除

所有由原代码生成的二进制文件。如果你不想丢掉这些已编译好的二进制文件，你也可以在安装之后将他们清除掉。注意这仅仅是从原代码目录中，而不是在系统中删除。

除了可执行文件以外，make clean 也删除一些生成的中间文件（例如 object 文件）。这样可以节省磁盘空间。

如果你在系统中使用/etc/rc.d 或者/etc/ini.d 目录，你同样可以运行 make config 命令。你需要安装启动脚本和配置系统，使用 chkconfig 命令在系统启动的时候自动装载 zaptel 模块。

Debian 中使用 chkconfig 是 update-rc.d 这样的。

使用 ztcfg 和 zttool

和 Zaptel 一起安装的两个程序是 ztcfg 和 zttol。Ztcfg 程序是用来读取/etc/zaptel.conf 这个配置文件来配置硬件的。Zttool 这个程序可以用来检测你的硬件的状态。例如，如果你使用 T1 卡并且在两个终端间没有通信，你将会看到一个红色的警报标志。如果你做好了所有配置并且可以通信了，那么你可以看到一个“OK”标志。Zttool 应用程序在模拟卡上，因为它告诉你当前的状态（配置，挂断等）。这个有用的程序将会在下一章详细介绍。

在编译 zttool 前，必须先安装 Libnewt 库和它的开发包。

zconfig.h 文件

Zconfig.h。多数情况下，你不需要编辑这个文件，但是下面的这些选项你可能会感兴趣。要使用这些选项，你要删除（/**/）这些标志。如果你想使用其中的某些选项，请在重新编译和安装 Zaptel 前运行 make clean。

1 Boost ringer

启用 BOOST_RINGER 选项，你的电话需要增大电压到 70V 到 89V 之间。低于某个电压值，一些设备不会被检测出来，所以这个设置是很有必要的。注意提高你的电压要满足要求，才能在整个电话系统中连接成功。基本上，你可以单独使用它，除非远端没有准备适当的铃声。使用这个选项，在下面这行删除注释：

/* #define BOOST_RINGER */

BOOST_RINGER 选项通过 modprobe 加载驱动的时候被声明，所以它不需要被编译到驱动程序当中。

2 禁止 □-law/A-law precomputation

定义 CONFIG_CALC_XLAW 告诉 Zaptel 不在每个例子都去在表中预计 -law/A-law 和重新计算。我们还没有与它同步，但是如果你有一些通道和/或一个小型二级缓存，初始编码器就会检测到，事实上它在执行代码的时候会比查询表加载到内存要更快。

要使用这个选项，在 zconfig.h 文件中删除下面这行的注释：

/* #define CONFIG_CALC_XLAW */

3 优化使用 MMX

通过删除下面这行的注释你可以优化使用 MMX（如果你的处理器支持的话）：

/* #define CONFIG_ZAPTEL_MMX */

CONFIG_ZAPTEL_MMX 这个选项被认为是与 AMD 处理器不兼容的，在 AMD 处理器上使用会使得系统不稳定。

4 Choose echo cancellation method

所有 Asterisk 中的的回声抑止都使用 FIR（Finite Impulse Response）算法。The differences between themmostly in code implementation and slight algorithm tweaksare minimal. By default, the MARK2 echo canceller is used, and it is generally considered the most robust. To change the default, add comment tags around the #define ECHO_CAN_MARK2 line and uncomment another line:

/* #define ECHO_CAN_STEVE */

/* #define ECHO_CAN_STEVE2 */

/* #define ECHO_CAN_MARK */

#define ECHO_CAN_MARK2

/* #define ECHO_CAN_MARK3 */

5 使用 aggressive suppression

Aggressive residual echo suppression with the MARK2 echo canceller can be enabled by removing the comment tags around the following line:

/* #define AGGRESSIVE_SUPPRESSOR */

The aggressive suppressor makes the nonlinear processor (NLP) stronger. What the NLP essentially does is say, "If the sample is that quiet anyway, make the volume level about 0."

6 Disable echo cancellation

When echo cancellation is enabled in Asterisk, it is possible to disable it by sending a 2100-Hz tone at the beginning of a call. If you do not want Asterisk to disable echo cancellation even when it detects the echo cancel disable tone, uncomment the following line:

/* #define NO_ECHOCAN_DISABLE */

Fax machines and modems use the 2100-Hz tone during negotiation , and Asterisk monitors for this tone during call setup.

7 使用 HDLC

当使用 Zaptel 来驱动 T1 或者 E1 卡的时候，你可以配置 Zaptel 来使用 TDM 数据通道替代语音。在驱动中打开 HDLC 功能，去掉下面这行的注释：

/* #define CONFIG_ZAPATA_NET */

这个修改的意义是重大的，你必须同时使用 sethdlc 和完成对 zapata.conf 文件的配置。

8 使用 ZapRAS

你同样可以使用 ZapRAS 程序让 Asterisk 通过 ISDN 连到远程访问服务器（RAS）上。使用这个功能，你必须在 zconfig.h 文件中去掉下面这行的注释：

/* #define CONFIG_ZAPATA_PPP */

你必须更新 Asterisk 和配置一个 PPP 进程，要知道这个任务可不寻常。

9 使用 Zaptel's watchdog

通过 Zaptel 内置的“watchdog”告诉它去监视接口状态。它将检测接口是否断开或者其他的非正常状态。如果确实不正常硬件将自动重启。要使用 watchdog, 去掉下面这行的注释：

/* #define CONFIG_ZAPTEL_WATCHDOG */

10 设置默认 tone zone

Tone zone 信息选项是用来选择哪一组音调（例如，拨号音，忙碌指示，铃声，stutter 等），在 zonedata.c 中定义，一般都使用默认值。在 zonedata.c 文件中包含频率以及 Asterisk 在各个国家 PSTN 网中通信和与电话连接的模式。对于北美呼叫的频率来说 Tone zone(0)常被设置为默认值。其他的 tone zones 包括 Australia(1),France(2),Japan(7),Taiwan(14),和其他的许多值。你可以在下面这行改变默认值：

#define DEFAULT_TONE_ZONE 0

11 Enable CAC ground start signaling

一些设备，例如像在运送通路公司(CAC)通道银行的 FXO 端口，有个非标准的 FXS 落地来发出开始状态（A=low, B=low），你可以通过删除下面这行的注释来配置使得驱动程序可以使用这一规定：

/* #define CONFIG_CAC_GROUNDSTART */

12 TDM400P 修正 H PCI ID 工作区

如果你已经使用一个老版本的 TDM400P 来修正一个 H 卡，你会发现它时常找不到它的 PCI ID。要使 wctdm 驱动程序根本上匹配所有的属主 IDs,去掉下面行的注释：

/* #define TDM_REVH_MATCHALL */

在用比较早期的 TDM400P 卡和比较新版本的 Asterisk 的时候，改变属主 ID 代码是必要的。当加载 wctdm 模块的时候会出现以下类型的错误：

ZT_CHANCONFIG failed on channel 12: No such device or address (6)

去掉#define 这行以上注释可以解决这个问题。

通过模块参数来配置 Zaptel

在加载 Zaptel 的时候，通过传递模块参数给 wctdm 驱动程序可以开启许多 Zaptel 选项。你可以使用 modinfo 命令来罗列这些参数的加载时间：

modinfo -p wctdm

debug int

loopcurrent int

robust int

_opermode int

opermode string

timingonly int

lowpower int

boostringer int

fxshonormode int

你可以传递模块参数给 modprobe 命令。例如，在 zconfig.h 文件中取代静态使用的#define BOOST_RINGER，你可以使用下列命令在模块加载的时候激活 boostringer 参数。

modprobe wctdm boostringer=1

另外的一个经常使用传递给模块的参数是 opermod。通过将 opermode 传递给 wctdm 驱动程序，你可以配置 TDM400P 以便在你的使用范围更好的处理电线电阻。Opermode 接受两个字母的国家代码为参数。

3.4 编译 libpri

编译和安装 libpri 和上一章安装 zaptel 相似。Libpri 被很多不同的分时系统硬件所使用，但是即使你没有相应的硬件，你同样可以编译和安装这个库。在编译和安装 Asterisk 之前你必须编译和安装 libpri， Asterisk 可以检测和使用它。

这里有一些命令（版本号使用你自己的 libpri 版本号）：

 # cd /usr/src/libpri-*version*

 # make clean

 # make

 # make install

3.5 编译 Asterisk

一旦你编译和安装了 zaptel 和 libpri 包（如果你需要它们的话），你可以把注意力移到 Asterisk 上了。这一节我们将介绍标准安装和一些有用的参数。你也可以考虑怎么编辑 Makefile 使得 Asterisk 的编译更为优化。

标准安装

Asterisk 是通过 GUN 的 make 程序中使用 gcc 来编译的。与其他的程序不同，它不需要运行配置脚本。为了开始编译 Asterisk，像下面那样运行命令（用你的 Asterisk 的版本号来取代版本号）：

 # cd /usr/src/asterisk-*version*

 # make clean

 # make

 # make install

 # make samples

编译时间因系统而异。在现代处理器，你最多只需要等待五分钟。据说，在 133-MHz 的处理器上也能成功编译 Asterisk，但是花费了将近五个小时。你自己可以计算一下。

运行 make samples 命令安装默认的配置文件。安装这些文件（手动替代这些配置文件）将使得你的 Asteirks 系统运行得更快。许多 Asterisk 默认值都设置都很好，一些需要设置的我们将在以后的章节里说明。

 如果你已经执行了 make samples 这个命令，将配置文件安装在/etc/asterisk/下，而.old 将附加到每个文件后，例如 extension.conf 将重命名为 extension.conf.old。要仔细，因为如果你运行了 make samples 将覆盖你原来的配置！

配置文件示例可以在 Asterisk 下的子目录 configs/中找到。

如果你的系统使用/etc/rc.d/init.d/ 或 /etc/init.d/ directories 目录，同时你也想运行 make config 命令。那么将需要安装启动脚本和配置系统（通过使用 chkconfig 命令）在启动的时候自动运行 Asterisk 。

可供选择的 make 参数

在编译的时候有许多 make 参数。我们将在这一节来讨论它，其他的也在文件内部使用但是和用户没什么关系。（当然，可能已经添加了许多新的函数。所以要检查 Makefile 文件是否有新的参数）

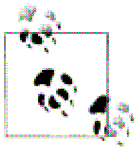
让我们看看有哪些有用的参数吧。

1 make clean

Make clean 命令的功能是从原代码目录中移除编译好的二进制文件。这个命令同样可以在你重新编译前运行，如果你的磁盘空间不足，你可以清除这些文件。

2 make update

这个命令常用于从 Digium CVS 服务器上得到更新。如果你从 FTP 服务器上下载原代码，你将遇到一些问题。

 一个常见的问题是：你用 update 命令从 CVS 服务器进行了更新，当你通过 Asterisk 命令行接口(CLI)显示版本号时，发现版本号并没有更新。这个问题在重新编译 Asterisk 原代码目录的时候隐式地解决，或者通过使用 make update 命令（它将移除你的文件）。

3 make upgrade

如果你运行 make install 命令后使用 make update 命令从 CVS 上的得到更新，.version 的文件不会得到更新。如果在运行 make 和 make install 之前，你不想手动删除.version 文件，那么你可以用 make upgrade 命令试一下。

4 make webvmail

Asterisk 的语音邮箱是使用图形界面，允许远程通过网页浏览器进行管理和交互。

当你运行 make webvmail 命令，Asterisk 语音邮箱脚本将在 cgi-bin/目录下放置一个 HTTP 进程。如果你对安全方面有特殊的要求，可以使用一个 setuid 管理员脚本。这个命令仅仅适用于 Red Hat 或者 Fedora，其他的系统将放置在不同的 cgi-bin/目录。（当然，你可以通过编辑 Makefile 来设置。）

5 make progdocs

开发者使用这个命令，通过 doxygen 软件将放置在原代码中的注释创建成文档。当然，在你的系统中必须安装有 doxygen 这个软件才能完成这个命令。注意，doxygen 认为原代码才是最好的文档，可悲的是，这一理论并不完全正确。

6 make mpg123

Asterisk 使用 mpg123 在等待音乐中处理 MP3s 音频流。因为 Asterisk 只对 mpg123 v0.59r 起作用，如果 mpg123 的版本正确，那就相当方便，否则，你将要下载，解压和编译它。要知道新的版本可能会导致工作不正常，与其他的 mpg321 和 mpg123 将生成一个完全不同的程序。如果你运行 make install 这个命令之后，Asterisk 会检测这个目录同时也会安装它。

7 make config

如果/etc/rc.d/init.d 或 /etc/init.d 目录没有找到或者不存在，Make config 将安装 RedHat 风格的初始化脚本。如果他们存在，那么文件的访问属性为 775。如果脚本检测到/etc/rc.d/init.d/存在，chkconfig 将自动添加 Asterisk 命令，在系统启动的时候自动运行 Asterisk。这不是唯一的情况，然而在其他类型的系统中仅仅使用/etc/init.d/目录。如果 Asterisk 已经运行，那么运行 make config 将不做任何事，如果 Asterisk 没有启动，那么将开启一个进程。

这个脚本仅仅适用于基于 Red Hat 的系统，虽然初始化在其他类型的系统中

同样可以使用（例如 Gentoo, Mandrake, 和 Slackware）在你 Asterisk 原代码目录的./contrib./init.d/中。

编辑 Makefile

在 Asterisk 原代码的目录中 Makefile 文件的顶部包含了编译 Asterisk 的一些操作和优化操作。你可以使用 GSM 编码器（通过使用 MMX 指令集），禁止覆盖配置文件，添加额外的调试信息，改变 Asterisk 的安装目录，改变编译时所处理的类型。要是你没编辑过或者没有任何其他的要求，那么这里所提到的已经很完整了。

1 优化使用 GSM

在 Asterisk Makefile 中下面这行没有注释，意思是在支持 MMX 指令集的 X86 CPU 体系结构中最优化使用 GSM 编码器：

#K6OPT = -DK6OPT

这里包含了许多新的处理器，Pentium Pros, and the AMD K6 and K7 processors;然而，除非你有一个真正的 Intel 处理器，否则你需要 MMX 的支持，MMX 指令集在非 Intel 处理器中已经有很多问题被报道。

2 禁止覆盖配置文件

Asterisk 的默认值是允许在你运行 make samples 后覆盖配置文件，为了修改这个操作，将下面这行 Y 改为 N：

OVERWRITE=y

3 允许使用调试信息

调试标记允许你做一些带标记的调试。信息压缩（-pg）标志在你运行 Asterisk 的时候可以知道每个函数的处理时间。-pg 并不是一个标准的参数，但是在开发

过程中却是很有用的。只要在下面这行中用 -pg 替代 -g：就能使用提示信息。

DEBUG=-g

4 在编译时指定 Asterisk 的安装位置

你可以在安装 Asterisk 的时候用下面一行命令来指定一个特定的位置：

INSTALL_PREFIX=

5 改变数据移动目录

Asterisk 是在安装的过程中拷贝临时文件到数据移动目录的。你可以将文件拷贝到像/tmp/asterisk 这样的目录中。如果没有指定目录（使用默认的），Asterisk 将使用原代码目录。要是需要指定一个目录，请在这行中输入指定目录：

DESTDIR=

6 在威胜（VIA）芯片的主板上编译

在基于威胜（VIA）芯片的主板上，你需要设置处理器为 i586，如果 Asterisk 检测到处理器为 i686，你可能得到很混乱的核心堆。在编译 Asterisk 的时候需要强制使用 i586，从 Makefile 文件中移除 PROC 这行的注释（Makefile 文件中的 81 行）：

Pentium & VIA processors optimize

PROC=i586

使用编译好的二进制文件

文档中已经告诉你怎么安装 Asterisk 的原代码了，在各种 Linux 中（例如 Debian）都包含了可以直接安装的二进制文件。如果没有的话，你可以使用包管理工具来安装各种编译好的文件（例如 Debian 的 apt-get 和 Gentoo 的 portage）。然而，你同样可以找到一些已经编译好的，而不必与 Asterisk 的开发周期同步。

最后，不管你选择何种的 Linux，都存在能在其中下载和安装的 Asterisk 已经编译好的二进制文件。然而，使用编译好的二进制文件并不见得节约很多时间，我们认为编译和安装 Asterisk 并不是一件麻烦的任务。我们认为安装 Asteirsk 的最好办法就是编译原代码，所以我们在本书讨论使用编译好的文件并不是很多。在下一章，我们将着眼于怎么初始化配置 Asterisk 和一些类型的通道。

3.6　其他的安装提示

asterisk-sounds 包包含了许多有用而专业的提示记录。它是被强烈推荐安装的，同时我们在以后的章节将使用这个包中的一些提示，为了使用它，运行下面的命令：

cd /usr/src/asterisk-sounds

make install

其他有用的附加信息

Asterisk-addons 包中包含了允许存储呼叫记录到 MySQL 数据库和自然播放 MP3S 的代码，也是一个 Asterisk 处理加载 Perl 到内存的解释程序。当受到有关许可的问题的阻碍时，不能在 Asterisk 源代码上直接实现，或者没有在最合适的时候准备好而不得不把项目放置在 asterisk-addons 内。

g729/这个目录包含了 G.729 的编码器的注册程序和代码。即使你的设备上已经安装了 G.729 编码器，为了让电话在使用了 G.729 的 Asterisk 进行通信（例如语言邮件），你必须准备一张证书。从 Digium 购买的证书中允许的编码器同时激活程序包含了 g729 目录。

3.7　更新原代码

你时常需要更新替代原代码和下载整个树性目录，当然你可以仅仅下载新版本的更新。改变目录中所包含的文件你需要运行更新命令：

cd /usr/src/asterisk/

make update

 # make clean

 # make upgrade

注意，这仅仅是通过 CVS 得到原代码的方法（在这章之前参考"make update,"）更新命令仅仅适用于 Asterisk 目录，其他的目录使用 make install.

3.8 普遍的编译主题

这里有用户遇到的一些编译主题。这里有许多普遍的问题和解决方法。

Asterisk

首先，让我们看一些在你编译 Asterisk 的时候将遇到的一些错误。

1 编译器不能生成可执行文件

如果你在编译 Asterisk 的时候看到以下的错误，你必须安装 gcc 编译器以及它附属的一些包：

checking whether the C compiler (gcc) works... no

configure: error: installation or configuration problem: C compiler cannot

create executables.

make: *** [editline/libedit.a] Error 1

下面这些包是 gcc 需要的：

- gcc
- glibc-kernheaders
- cpp
- binutils
- glibc-headers
- glibc-devel

 你可以通过各种类型的磁盘获取文件来手动安装，或者通过 yum 包管理，使用命令 yum 来安装 gcc。

2 bison：无效的命令

要是没有找到 bison 解析器可能遇到下面的错误， extensions.conf 文件中的表达式需要 bison 来解析：

```
bison ast_expr.y -name-prefix=ast_yy -o ast_expr.c
make: bison: Command not found
make: *** [ast_expr.c] Error 127
```

在安装 Asterisk 的时候需要的下面这些文件；它们可以通过 yum 来安装 bison 命令来安装：

- bison
- m4

3 /usr/bin/ld：找不到 -lssl

OpenSSL 开发包需要 Asterisk 的 res_crypto.so 模块中通过 IAX2 协议核对 RSA 的性能。如果 OpenSSL 开发包没有安装，将会看到下面的错误：

```
/usr/bin/ld: cannot find -lssl
collect2: ld returned 1 exit status
make: *** [asterisk] Error 1
```

安装 OpenSSL 开发包，你需要以下的包：

- openssl-devel
- e2fsprogs-devel
- zlib-devel
- krb5-devel

- krb5-libs

你可以通过使用 yum install openssl-devel 命令来安装这些文件。

4 找不到 rpmbuild 命令

使用 make rpm 命令，你需要安装 Red Hat 包管理工具。没有安装的话将遇到以下的错误：

```
make[1]: Leaving directory '/usr/src/asterisk-1.0.3'
/bin/sh: line 1: rpmbuild: command not found
make: *** [_  _rpm] Error 127
```

你可以使用 yum install rpmbuild 来设置环境。

Zaptel

这里有一些在编译 Zaptel 时经常遇到的错误和解决方法。

1 make: cc: 找不到命令

在你没安装 gcc 编译器前就尝试编译 Zaptel，那么你肯定会看到下面的这些错误：

```
make: cc: Command not found
make: *** [gendigits.o] Error 127
```

确保你已经安装了 gcc 以及相关的包。你可以参照以前的“C 编译器不能创建可执行文件“这一章。”

2 错误：找不到 wctdm/fxs/fxo 模块

TDM400P 卡要求 PCI 的总线在 2.2 版本以上。要是你试图在低于 2.2 版本下加载 Zapata 驱动，无疑你将看到以下错误：

- 尝试加载 wctdm 驱动，你会看到这样的错误：

  FATAL: Module wctdm not found

- 尝试加载 wctdm 或者 wcfxo 驱动，你会看到这样的错误:

  ZT_CHANCONFIG failed on channel 1: No such device or address (6)

  FATAL: Module wctdm not found

解决这些问题的唯一方法是使用至少支持 PCI 2.2 的主板。

如果你的 TDM400P 卡没有接上电源，你同样会遇到一些错误。

3 处理加载 ztdummy 的时候出现的连接问题

Ztdmmy 驱动需要 Linux2.4 内核上使用 UHCI USB 控制器（USB 控制器不一定由 Linux2.6 内核支持，因为它可以生成 1-kHz 的实例）。它有两级控制器，大家都知道 OHCI，可是它和 zteummy 并不兼容。如果 UHCI USB 控制器不受 Linux2.4 影响，那么会出现以下错误：

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol unlink_td

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol alloc_td

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol delete_desc

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol uhci_devices

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol uhci_interrupt

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol fill_td

/lib/modules/2.4.22/misc/ztdummy.o: /lib/modules/2.4.22/misc/ztdummy.o: unresolved

symbol insert_td_horizontal

/lib/modules/2.4.22/misc/ztdummy.o: insmod /lib/modules/2.4.22/misc/ztdummy.o failed

/lib/modules/2.4.22/misc/ztdummy.o: insmod ztdummy failed

你要确定你有正确的 USB 控制器并用 lsmod 命令来和驱动程序相关联：

lsmod

Module	Size	Used by
usb_uhci	26412	0
usbcore	79040	1 [hid usb-uhci]

如果你想看到上面的例子，那么你需要确定你已经加载了 usbcore 和 usb_uhci 这两个模块。如果这两个模块没有被加载，那么要确定 USB 已经在你的 BIOS 中激活同时有这两个模块并且被加载。

如果没有加载 USB 驱动，你可以用 dmesg 命令来检查 USB 的类型：

dmesg | grep -i usb

确认你有 UHCI USB 控制器，找到下面几行：

uhci_hcd 0000:00:04.2: new USB bus registered, assigned bus number 1

hub 1-0:1.0: USB hub found

uhci_hcd 0000:00:04.3: new USB bus registered, assigned bus number 2

hub 2-0:1.0: USB hub found

4　编译时 Depmod 错误

如果在编译的时候遇到 depmod 错误，你可能没有和你的 Linux 内核建立连接。如果你没有安装 Linux 内核代码，去找到并安装它，并且建立一个到 /usr/src/linux-2.4 的连接。下面就是一个 depmod 错误：

depmod: *** Unresolved symbols in /lib/modules/2.4.22/kernel/drivers/block/loop.o

3.9　加载 Zaptel 模块

在这部分，我们来看一下怎么加载 zaptel 和 ztdummy 模块。要是仅仅使用 Ztdummy 模块的话就不需要任何的配置。如果你计划将 ztdummy 模块做为分时资源来加载，（因此，你不需要在你的系统中运行任何 PCI 设备）现在是加载这两个模块的好时机。

系统运行 udevd

在早些时候的 Linux，系统的/dev/目录放置了系统潜在交互的设备清单。现在，这个清单里罗列了有将近 18.000 个设备。每当 devfs 有所变动的时候，都允许动态创建活动的设备（即插即用）。最近发布的一些类型已经被合并到系统的 udev 这个进程中以便动态的将设备结点加入/dev/中。

允许 Zaptel 和其他的设备通过 PCI 硬件安装到系统中，你必须添加一些规则。使用你喜欢的文本编辑器，打开 udevd 规则文件。例如在 Fedora Core 3 中，这个文件放在/etc/udev/rules.d.50-udev.rules，将下面这行添加到文件的末尾：

```
# Section for zaptel

device
KERNEL="zapctl",         NAME="zap/ctl"
KERNEL="zaptimer",       NAME="zap/timer"
KERNEL="zapchannel", NAME="zap/channel"
KERNEL="zappseudo",    NAME="zap/pseudo"
```

KERNEL="zap[0-9]*",　NAME="zap/%n"

保存文件并且重启系统就会生效了。

加载 Zaptel

zaptel 模块必须是在所有的模块中最先被加载的。注意，如果你要在 PCI 硬件上使用 zaptel 模块，在你加载它之前必须配置/etc/zaptel.conf（我们将在第四章讨论怎么配置/etc/zaptel.conf 来使用硬件）。如果你仅仅通过 ztdummy 来使用 zaptel，可以通过 modprobe 命令来加载她，像这样：

modprobe zaptel

如果一切正常，你也看不到任何的输出。要确认 zaptel 模块加载是否成功的话，使用 lsmod 命令查看。它将显示 zaptel 模块和它使用的内存：

lsmod | grep zaptel

zaptel　　201988　0

加载 ztdummy

Ztdummy 模块是硬件提供分时的一个接口，它使得 Asterisk 提供了许多应用程序和函数所需要的分时性。在 zaptel 模块被加载之后使用 modprobe 命令来加载 ztdummy 模块：

modprobe ztdummy

如果 ztdummy 加载成功，同样不会有任何的输出。使用 lsmod 命令可以确认 ztdummy 是否加载成功并且是否被 zaptel 使用。在 2.6 的内核下就得到如下显示：

lsmod | grep ztdummy

Module　　Size　Used by

ztdummy　　3796　0

zaptel　　201988　1 ztdummy

如果你的计算机运行的是 2.4 内核的系统，你使用 lsmod 将看到 ztdummy 正在使用 usb_uhci 模块：

```
# lsmod | grep ztdummy
Module                  Size  Used by
ztdummy                 3796  0
zaptel                201988  0 ztdummy
usb-uhci               24524  0 ztdummy
```

3.10　加载 libpri

libpri 库不需要像模块一样被加载。Asterisk 在编译的时候会寻找它同时如果找到的话会配置它。

3.11　加载 Asterisk

加载 Asterisk 可以有很多方法。最简单的方法就是通过 Linux 命令行接口直接运行二进制可执行文件。如果在系统启动的时候使用 init.d 脚本，你同样可以很容易启动和重新启动 Asterisk。然而，最好的方法是通过 safe_asterisk 脚本.

CLI 命令

Asterisk 二进制可执行的位置默认是放在/usr/sbin/asterisk。如果你运行/usr/sbin/asterisk，它将会做为一个进程被加载。有许多参数可供选择：允许连接到 Asterisk CLI，设置 CLI 输出，和如果 Asterisk 崩溃的时候允许使用核心堆（使用 gdb 调试）。研究整个操作范围，运行 Asterisk 的时候使用-h 开关。

```
#/user/sbin/asterisk –h
```

下面是最常用的选项列表：

-c

console 控制台。允许你连接到 Asterisk 的 CLI 界面。

-v

简繁程度。设置 CLI 的调试信息输出的多少。

-g

核心。如果 Asterisk 突然瘫痪，在接下来的时间，为了用 gdb 来做 trace，可以产生一个核心文件。

-r

远程。用来远程重新连接到一个已经运行的 Asterisk 进程中。（这个过程只是指和连接到系统上的 console 对比的，实际上也是机器上的一个本地过程。这和通过比如 IP 之类的协议连接到一个远程进程上去毫无关联）。

-rx　“restart now”

执行。和-r 组合，使用这个命令，能够执行一个 CLI 命令，而不需要连接到 CLI 并手动敲命令。

让我们看几个例子。为了启动 Asterisk 并连接到 CLI 并用 3 级的复杂度来调试，使用下述命令：

#/user/sbin/asterisk –cvvv

如果 Asterisk 进程已经运行（比如，如果你用/user/sbin/asterisk 启动 Asterisk），就要使用重新连接开关，比如：

#/user/sbin/asterisk –vvvr

要执行一个命令，而不希望连接到 CLI 并敲入这个命令（可能使用一个脚本），你可以输入-x 开关并与-r 开关组合：

#/user/sbin/asterisk –rx “restart now”

如果系统出现瘫痪并希望输出一个 debug 文件，使用下面的命令：

#/user/sbin/asterisk –vvvvvvvvvc | tee /tmp/debug.log

Red Hat 样式的初始化脚本

如果早一些运行 make config 命令（或者手动拷贝初始化脚本），你可以用下面的命令启动或者重新启动 Asterisk：

#/etc/rc.d/init.d/asterisk start

#/etc/rc.d/init/d/asterisk stop

Safe_asterisk 脚本

Safe_asterisk 脚本的主要目的是产生一个核心文件，如果 Asterisk 瘫痪或者自动重启。在脚本上有一个指示选项，如果设置了该选项，将发一封电子邮件，让你知道 Asterisk 已经非正常死机。脚本的一个附加好处是在终端接口 9 上装载 Asterisk CLI，因此你可以很容易地切换到哪个窗口来监视 Asterisk 系统的运行状态。

缺省的 safe_asterisk 脚本的位置在/user/sbin/safe_asterisk，并如此执行。让我们复习各种 safe_asterisk 脚本的选项吧。

```
CLIARGS="$*"    # Grab any args passed to safe_asterisk
TTY=9           # TTY (if you want one) for Asterisk to run on
CONSOLE=yes     # Whether or not you want a console
#NOTIFY=ben@alkaloid.net        # Email address for crash notifications
```

第一行简单地通过 Linux CLI 把参数传递给 safe_asterisk 脚本；他们不能直接被编译。TTY=9 标明运行 Asterisk CLI 输出的 Linux 控制台。你可以通过 CONSOLE＝no 禁止这个特性。当 Asterisk 突然死机并希望重启的时候，如果你希望被提示，将 NOTIFY 那一行的邮箱地址 ben@alkaloid.net 替换为你的邮箱地址。注意：瘫痪指示用 mail 命令发送，因此你的操作系统必须正在运行必须能够发邮件。

3.12 使用 Asterisk 目录

Asterisk 在 Linux 系统中使用许多目录来管理，例如语音邮件记录，声音提示，和配置文件。这一节将讨论这些目录，所有这些都在安装的时候在 asterisk.conf 文件中配置。

/etc/asterisk/

/etc/asterisk/这个目录包含了 Asterisk 配置文件。然而 zaptel.conf 这个文件在/etc/这个目录里。Zaptel 部件最初由给全球提供计算机硬件的 Zapata 科技开发小组中的 Jim Dixon 开发设计的。Asterisk 使用这个硬件，但是其他的任何软件都可以使用 Zaptel 这个硬件及其驱动。因此，zaptel.conf 这个配置文件并不是放在/etc/asterisk 这个目录中的。

/usr/lib/asterisk/modules/

/usr/lib/asterisk/modules/这个目录包含了所有可加载的 Asterisk 模块。在这个目录中有许多应用程序，编码器，格式，和有用的通道。在 Asterisk 启动的时候将加载这些模块。你可以在 modules.conf 这个文件中禁止你不使用的模块，但是必须明白各个模块之间的依附关系。没有这些模块，在启动 Asterisk 的时候会引起很多错误。

/var/lib/asterisk

/var/lib/asterisk/这个目录包含了 astdb 这个文件和许多子目录。astdb 这个文件包含了许多 Asterisk 当地数据库的信息，有点像微软的 Windows 系统的注册。Asterisk 数据库是在伯克利数据库 V1 上的简单实现。Asterisk 中选择 db.c 这个文件有以下原因：“DB3 在其证书对 GPL 不兼容的前提下将被释放。因而为了保证 Asterisk 证书的单纯化，它决定使用版本 1 作为它在 BSD 下的许可证书。

/var/lib/asterisk/这个子目录中包括：

agi-bin/

agi-bin/这个目录包含了所有的脚本，可以通过许多已经建立的 AGI 应用程序与 Asterisk 连接。关于 AGI 相关知识，请参考第八章。

firmware/

firmware 这个目录包含了许多与 Asterisk 相兼容的设备固件。它只有 iax/这一个子目录，其中有 Digium 的 IAXy 的二进制固件镜像。

images/

应用程序与有图形支持的信道通信相关内容请参考 images/目录。大多数的通道不支持图像的传输，因此，这一个目录很少使用。 然而，如果有较多的支持并且利用图解式的图像的设备被发布,这一个目录将会与相应的目录有更大的关联。

keys/

Asterisk 可以使用公钥/私钥系统来认证与一个由 RSA 数字签名形成一个对等的连接。如果你把公钥放到你的 keys/目录，那么通道将被这一方法支持。私钥不被大家所持有。相反这样也是可以的：你可以随意发放你的公钥，让它和你的私钥一起使用才能得以认证。公钥和私钥文件的扩展名分别为.pub 和.key，存放在 key/目录中。

mohmp3/

如果你配置了 Asterisk 音乐保持，应用程序会在 mohmp3 这个目录下寻找 MP3 文件。Asterisk 对 MP3 的格式要求相当严格，所以你可以用 CBR 从你的文件中去掉身份标签。

sounds/

所有用到的声音提示文件都在 sounds/这个目录下。其中包括 Asterisk 原代码中的 sounds.txt 文件。其他的提示内容都在以前所提到的从 asterisk-sounds 包中提取的 sounds-extra.txt 文件中。

/var/spool/asterisk/

Asterisk spool 目录中包含了许多子目录，包括 outgoing/,gcall/,tmp/,和 voicemail/（参见图 3－2）。Asterisk 监控 outgoing 和 qcall 目录将所有的呼叫请求信息以文本方式保存。这些文件允许你通过拷贝或者移动正确格式的文件到 outgoing/目录中发起一个简单的呼叫。

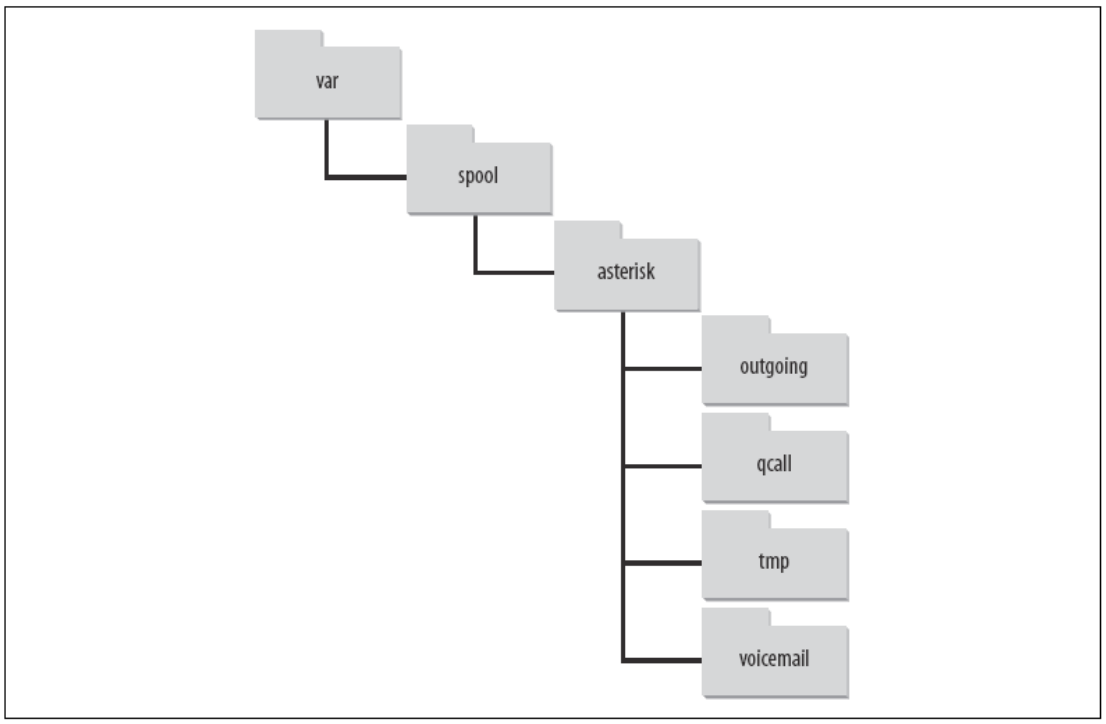


图 3-2　/var/spool/asterisk/目录结构

3.13　结束语

在这一章里，我们回顾了得到、编译和安装 Asterisk 以及相关包的流程。在以后的章节，我们将接触初始化配置系统中各种通信通道，例如带有 FXS 和 FXO 端口的设备，SIP 通道，和 IAX2 终端。

第4章　Asterisk 的基本配置

坚持，是你厌倦了重复做你已经做过的艰苦工作之后，你需要的艰苦工作

—Newt Gingrich

本章主要告诉用户如何配置以下四种通道：FXO 通道，FXS通道，SIP通道，和Asterisk 内部协议（Inter- eXchange protocol IAX）通道。目标不是对所有的通道或者拓扑进行全面的研究，而是提供一个基本的平台，在此基础上你可以建立自己的电话系统。进一步的场景和通道配置可以参阅附件D。使用Digium Dev-Lite工具，我们先探讨对模拟接口如FXS和FXO端口的基本配置。然后配置两个VoIP接口：一个连接软电话的本地SIP通道，一个通过IAX连接全球免费通信（Free World Dialup）。

当你通读了本章后，你应该掌握了由多种用途广泛的接口组成的基本系统，并且可以学习关于*extensions.conf 文件（将在第五章探讨）*的更多知识，其中包括了Asterisk需要的用法。

4.1　我到底需要什么？

星号字符(*)在很多不同的应用中被用作通配符。星号作为这个PBX系统的完美名称，原因很多，其中之一是Asterisk可以连接的数目庞大的接口类型，包括：

- 模拟接口，如你的电话线和模拟电话
- 数字线路，如T-1 和 E-1线路
- VoIP协议如SIP和IAX

* 目前的正式版本是IAX2，但是对IAX1的所有支持已经停掉，所以当你说“IAX”或者“IAX2”的时候，公认为你说的是第二个版本。

Asterisk并不需要任何特定的硬件，甚至是声卡。连接Asterisk到模拟电话或者电话线的板卡可以找到，但是这并不是必须的。你可以使用Windows,Linux和

其他操作系统支持的软电话进行和Asterisk的连接，无需使用特定的硬件接口。你也可以使用支持SIP或者IAX2的IP电话。另一方面，如果你并不从你的办公中心直接连接到模拟电话线，你可以把你的呼叫从互联网路由到电话服务提供商。

4.2 使用接口配置文件

本章中，我们终将“把手弄脏”，开始创建一个Asterisk配置。在最初的几节关于FXO和FXS通道中，我们假定你有包含一个FXO和一个FXS接口的Digium Dev-Lite工具包，可以连接一个模拟电话线（FXO）和一部模拟电话(FXS)。请注意，这种硬件接口不是必须的，如果你想建立一个纯IP的配置，你可以跳到SIP配置的章节。

本章我们的配置本身并不是特别的有用，却是核心所在。我们将要接触以下的文件：

zaptel.conf

这里，我们将做硬件接口的基层配置。我们将建立一个FXO和一个FXS通道

zapata.conf

在这个文件，我们将为硬件配置Asterisk的接口

extensions.conf

我们创建的拨号方案会比较粗糙，但是它们将证明系统可用。

sip.conf

我们配置SIP协议的文件

iax.conf

我们配置呼入和呼出IAX通道的文件

以下的章节中，你将编辑几个配置文件，并且将这些文件装载到你的交换机上，使它们生效。编辑完 *zaptel.conf*文件，你需要使用/sbin/ztcfg －vv （如果你不需要详细的输出可以省略－vv）装载配置到硬件。然而，改变信令方法需要重启。在编辑*iax.conf*和 *sip.conf*文档后，你分别需要装载chan_iax2.so和chan_sip.so。

4.3 FXO 和 FXS 通道

FXO和FXS通道的区别简单，就在于连接的哪端提供拨号音。FXO端口不生成拨号音，而是接受。通常的例子是拨号音由你的电话公司提供。

FXS端口提供拨号音和震铃电压，在有呼入的时候提醒用户。两种接口都提供双向通讯（同时双方向的通信传输）。

如果你的Asterisk服务器有一个兼容的FXO端口，你可以把你的电话公司的电话线插入这个端口。Asterisk可以使用这根电话线呼出和接受电话呼叫。同理，如果你的Asterisk服务器有一个兼容的FXS端口，你可以连接一部模拟电话，Asterisk就可以呼叫这部电话，同时你也可以进行呼叫。

端口是通过配置使用的信令进行定义的，而不会是物理端口类型。例如，一个物理的FXO端口可以在配置中使用FXS信令定义，而FXS端口可以用FXO信令定义。在没有理解原因之前，可能有点容易混淆。FX_板卡不是通过他们是什么进行命名，而是根据他们连接的设备。因此FXS卡，是连接终端的。可见，为了做它的工作，FXS卡必须像中心局(CO)进行运转，使用FXO信令。同理，FXO卡连接CO,意味着它需要像终端进行运转，使用FXS信令。你电脑里面的调制解调器就是FXO设备的典型例子。

老款的X100P使用摩托罗拉的芯片，X101P（Digium在转向更有

竞争力的TDM400P之前进行销售）基于Ambient/Intel MD3200芯片。这些卡是调制解调器，用驱动将这些卡改造用作单独的FXO设备（电话接口不能用作FXS端口）。为了TDM系列卡，不再支持X101P卡。不建议在生产环境中使用这些卡（或者复制品）。

在你的 TDM400P 上决定 FXO 和 FXS 端口

图4-1是有一个FXS模块和一个FXO模块的TDM400P图片。你看不到颜色，但是模块1是绿色的FXS模块，模块2是橙红色的FXO模块。在图片的右下角是Molex连接器，从这里利用电脑提供电源。

插入一个FXS端口（绿色的模块）到PSTN会毁坏模块和板卡。

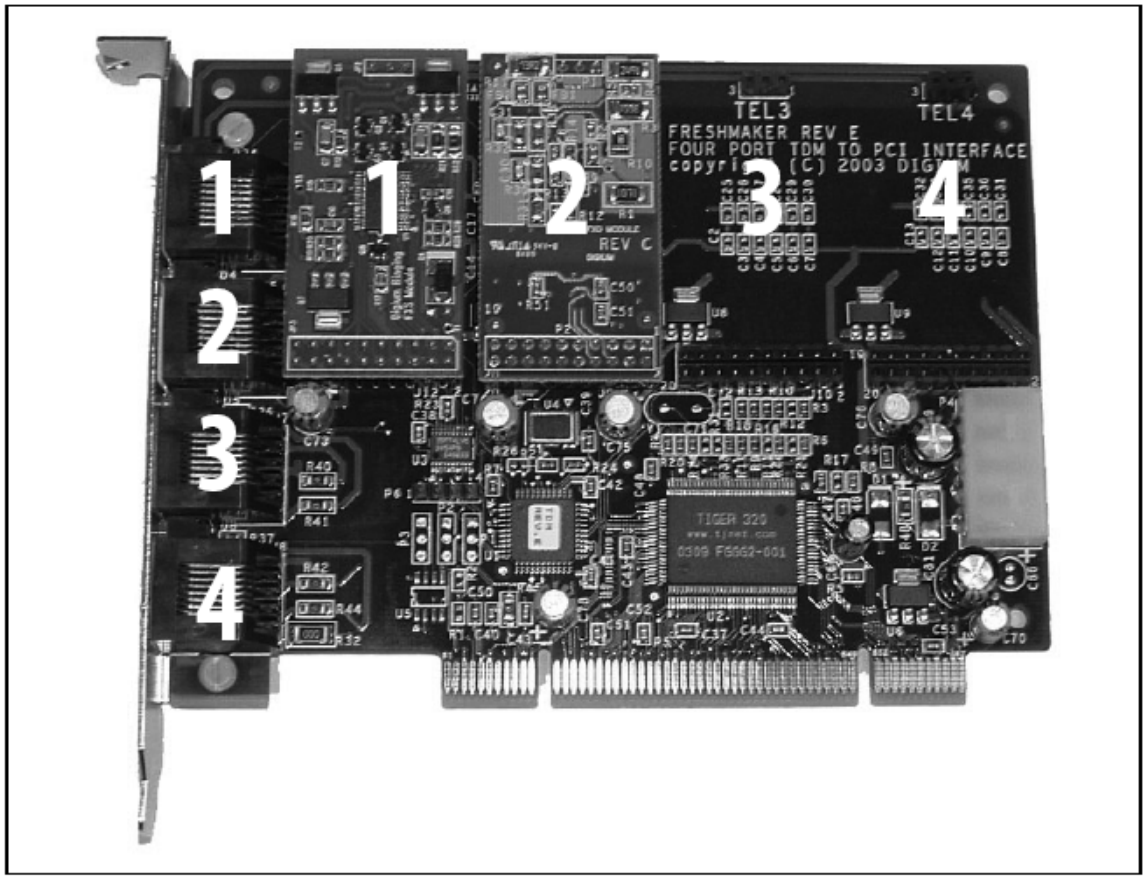


图4-1　TDM400P，带有一个FXS模块和一个FXO模块

注意如果你有FXS模块，一定要将你电脑的电源连接到TDM400P的Molex连

接器，用于生成电话震铃的电压。如果你只有FXO模块，那么不要求有Molex连接器。

4.4 配置 FXO 通道

我们将开始配置FXO通道。首先我们配置Zaptel硬件，然后是Zapata硬件。我们会建立一个非常基础的拨号方案，并且展示如何检测通道。

Zaptel 硬件配置

在*/etc/* 目录下的*zaptel.conf* 文件，用于配置硬件。下面是定义FXS端口的FXS信令的最低限度的配置：

```
fxsks=2
loadzone=us
defaultzone=us
```

除了在第一行，我们提到正在使用FXO或FXS信令，我们还为通道2定义以下协议中的一种：

- Loop start (ls)
- Ground start (gs)
- Kewlstart (ks)

*loop start*和*ground start*之间的区别在于设备如何请求拨号音：ground start电路信号通知远端它需要拨号音，即刻grounding one of the leads;环路启动电路使用一个short来请求拨号音。虽然新的使用中不常见，模拟ground start线路仍然在某些地区使用。例如，ground start线路的优势应用在于消除一种叫做“glare”的现象，这种现象在loop start线路和PBX较高的呼叫音量时出现。北美所有的家庭线路（和模拟电话、解调器、传真）使用loop start信令。*Kewlstart*实际上和loop start相同，只是更为智能，能够更好的检测远端的断线。Kewlstart是Asterisk的模拟电路推荐的信令协议。

要配置kewlstart以外的信令协议，用ls或者gs（分别对应loop start 或者ground start）替换fxsks 中的ks。

loadzone 配置通道使用的提示包（在*zonedata.c中配置*）。*zonedata.c*包含了在某个国家中电话系统需要使用的不同声音信息：拨号音，回铃音，忙音等等。当你为Zap通道应用一个装载过的音域，这个通道会模拟特定国家的提示音。不同的提示包可以为不同的通道配置。Defaultzone用于通道没有指定音域的情况。

配置完 *zaptel.conf*后，你可以载入板卡的驱动。Modprobe用于装载Linux kernel使用的模块，例如装载*wctdm*驱动，你需要运行

modprobe wctdm

当然，有一个称为通道化的T1上ground start信令，然而那个和实际的电路接地条件没有任何关系（完全是数字的）。

当一个呼叫从电路的一头发起，同时一个呼叫正好从电路的另一头发起。

远端断链（*far-end disconnect*）发生在远端关机的时候。在没有监视的线路上，没有办法告诉本端呼叫已经结束。如果你在没有问题的电话上通话，你如果知道呼叫结束，会主动挂断。然而如果你的语音邮件系统纪录一个信息，那么将没有办法知道远端是否挂机，就会一直录音，甚至是拨号音或催挂音。

Kewlstart可以检测到这些情况，断开电路。

如果驱动装载没有任何提示，则表明装载成功。你可以检测硬件和端口是否正确装载和配置，使用*ztcfg程序：*

/sbin/ztcfg －vv

配置的通道和使用的信令方法会显示出来。例如，带有一个FXO模块的TDM400P会有如下输出：

Zaptel Configuration

==========

Channel map:

Channel 02: FXS Kewlstart (Default) (Slaves: 02)

1 channels configured.

如果你收到了如下的错误，你就配置了错误的信令方式：

ZT_CHANCONFIG failed on channel 2: Invalid argument (22)

Did you forget that FXS interfaces are configured with FXO signalling

and that FXO interfaces use FXS signalling?

从内存中卸载驱动，使用rmmod（去除模块）命令，如下：

rmmod wctdm

*zttool*程序是一个分析工具，用于分析硬件的状态。运行后，你可以看到一个所有安装硬件的菜单。你可以选择某个硬件，查看目前状态。“OK”表示硬件安装正确。

Alarms Span

OK Wildcard TDM400P REV E/F Board 1

Asterisk通常情况下可以认为模块安装成功，但是要在装载模块时看调试输出，需要看控制台输出（默认输出在TTY terminal 9，可以在*safe_asterisk*中进行配置，参考以前章节）

Zapata 硬件配置

Asterisk 使用*zapata.conf* 文件确定系统中安装的电话硬件的设置和配置。*zapata.conf* 文件同时控制硬件通道相关的不同功能和特性，例如主叫号码显示，呼叫等待，回声消除，以及种种其它功能。

当你配置*zaptel.conf*文件并装载模块，Asterisk系统不知道你的任何配置。Asterisk不需要必备的硬件：系统可以使用其它使用Zaptel模块接口的软件。通过*zapata.conf文件*配置Asterisk硬件和相关特性的控制：

```
[trunkgroups]
; define any trunk groups
[channels]
; hardware channels
; default
usecallerid=yes
hidecallerid=no
callwaiting=no
threewaycalling=yes
```

transfer=yes

echocancel=yes

echotraining=yes

; define channels

context=incoming ; Incoming calls go to [incoming] in extensions.conf

signalling=fxs_ks ; Use FXS signalling for an FXO channel

channel => 2 ; PSTN attached to port 2

[trunkgroups]用于配置NFAS和GR-303连接，本书不会对此进行探讨。如果你需要此类的功能，更多信息见zapata.conf.sample文件

[channels]决定硬件通道和他们选项的信令方式。一旦一个选项定义完毕，文件的其它部分会继承这个定义。一个通道定义使用channel =>，则其他每个通道定义继承以上行所定义的所有选项。如果你想给每个通道定义不同的选项，要记住在channel =>定义**之前**配置选项。

我们用usecallerid=yes设置了来电显示，并且用hidecallerid=no设置呼出时不隐藏。FXO线路使用了callwaiting=no禁止呼叫等待。使用threewaycalling=yes开启三方通话，允许通过闪断键hook switch flash将当前的设置成保持并监视当前呼叫。然后你可以呼叫第三方，通过再一次使用闪断键hook switch flash使他们进入交谈中。系统缺省是不使用三方通话。

配置transfer=yes可以使用闪断键hook switch flash实现呼叫前转，它需要开启三方通话。Asterisk回声消除用于消除模拟线路可能产生的回声。你可以使用echocancel=yes开启回声消除。Asterisk的回声消除需要一些时间去学习回声，但是你可以使用回声训练echotraining=yes来加速这个过程。Asterisk将在会话的开始向线路发出一个声音，用于测量回声，然后会学习的更快。

当一个呼叫到达FXO接口时，你可能希望执行一些动作。这些在一个称为上下文*context*指令集中进行配置。FXO端口的呼入呼叫由context=incoming指向incoming context。Context内需要执行的指令在*extensions.conf*内进行定义。
最后，由于FXO通道使用FXS信令，我们用signalling=fxs_ks对此进行定义。

Dialplan 配置

以下的基本的拨号方案使用Echo( )应用来校验工作通道的双向通讯：

[incoming]

; incoming calls from the FXO port are directed to this context from zapata.conf

exten => s,1,Answer( )

exten => s,2,Echo( )

无论你说什么，Echo( )都将会回转给你。

呼入

现在FXO通道已经配置完毕，我们来测试一下。运行*zttool* 应用，并且把你的PSTN线路连接到你的TEM400P的FXO端口，你可以看到卡上会有一个红色的指示。

现在从另一个外部电话（例如手机）拨打这个PSTN号码，Asterisk 会接收这个呼叫并执行Echo( ) 应用。如果你可以听到回送过来的你的声音，则FXO通道的安装和配置已经成功。

4.5 FXO 和 FXS 通道

FXS通道的配置和FXO通道相似，我们可以看到：

Zaptel 硬件配置

以下是TDM400P的FXS通道的基本的配置。配置和以上FXO通道的配置相同，只是增加了fxoks=1。

回想一下前面我们关于FXO和FXS通道使用不同信令的讨论，我们将会为FXS通道配置FXO信令。以下的例子中我们配置通道1使用FXO信令，使用kewlstart协议：

fxoks=1

fxsks=2

loadzone=us

defaultzone=us

为硬件装载驱动后，你可以通过/sbin/ztcfg －vv检测他们的状态:

Zaptel Configuration

==========

Channel map:

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

Channel 02: FXS Kewlstart (Default) (Slaves: 02)

2 channels configured.

Zapata 硬件配置

以下的配置和FXO通道的配置相同，只是增加了FXS端口的字段和immediate=no行。FXS端口的context 是internal，信令是fxoks (kewlstart)，通道号设置为1。

FXS通道可以设置为电话摘机后执行两种不同动作中的一种。最普遍（最常期待）的是Asterisk产生拨号音，等待用户输入。这种用immediate=no进行配置。另一种是Asterisk自动执行一套拨号方案中设置的指令，而不是产生拨号音，通过配置immediate=yes进行配置。执行的指令在该通道的context中设置，并且匹配s扩展（这两种都会在后面的章节进一步探讨）。

我们的新的*zapata.conf*:

[trunkgroups]

; define any trunk groups

[channels]

; hardware channels

; default

usecallerid=yes

hidecallerid=no

callwaiting=no

threewaycalling=yes

transfer=yes

echocancel=yes

echotraining=yes

immediate=no

; define channels

context=internal ; Uses the [internal] context in extensions.conf

signalling=fxo_ks ; Use FXO signalling for an FXS channel

channel => 1 ; Telephone attached to port 1

context=incoming ; Incoming calls go to [incoming] in extensions.conf

signalling=fxs_ks ; Use FXS signalling for an FXO channel

channel => 2 ; PSTN attached to port 2

Dialplan 配置

为了测试我们新创建的Zap扩展，我们需要创建一个基本的拨号方案。以下的拨号方案包含一个称为internal 的context。这是我们为通道1*zapata.conf*配置的相同的context名称。当我们配置*zapata.conf*中的context=internal，我们在告诉 Asterisk 当用户在电话上拨号时候到哪里去找指令。本例中，唯一的工作的分机号码是611。当你在电话上拨611，Asterisk 会执行Echo( ) 应用，这样当你打电话的时候，你所说得话会返回给你，从而检测双向语音。

拨号方案如下所示：

[internal]

exten => 611,1,Answer( )

exten => 611,2,Echo( )

4.6 sip 配置

会话初始协议(SIP)，通常用于VoIP电话（硬件电话或者软电话），进行呼叫建立和呼叫结束，和呼叫进程中的协商。基本上，它帮助两个端点互相通话（如

果很可能，直接通话）。SIP不处理媒体；当呼叫建立后，它通过实时传输协议(RTP)在电话A和电话B之间直接传输媒体。

SIP 和 RTP

SIP是一个应用层的信令协议，使用众所周知的5060端口进行通信。SIP可以通过UDP或者TCP传输层协议进行传输。Asterisk目前没有TCP用于传输SIP信息，但是以后的版本有可能会支持（很高兴接受编码库的补丁）。SIP用于“建立，修改和终止如互联网电话呼叫的多媒体会话”。SIP不在端点之间传输媒体。

我们使用媒体表示端点之间传输的用于在另一端点重建你的声音数据。同样也指PBX的音乐或者声音提示。

RTP 用于在端点之间传输媒体(例如语音)。Asterisk 中RTP使用大数字的无特权的端口 (默认是10,000 到 20,000)。

描述SIP 和RTP的一个典型拓扑，通常称为 “SIP梯形”，如图4-2。当Alice想给Bob打电话，Alice的电话联系它的代理服务器，然后代理试图找到BOB（通常通过他的代理进行连接）。当电话呼叫开始后，他们直接和对方通讯（如果可能），因此数据不需要和代理的资源绑定。

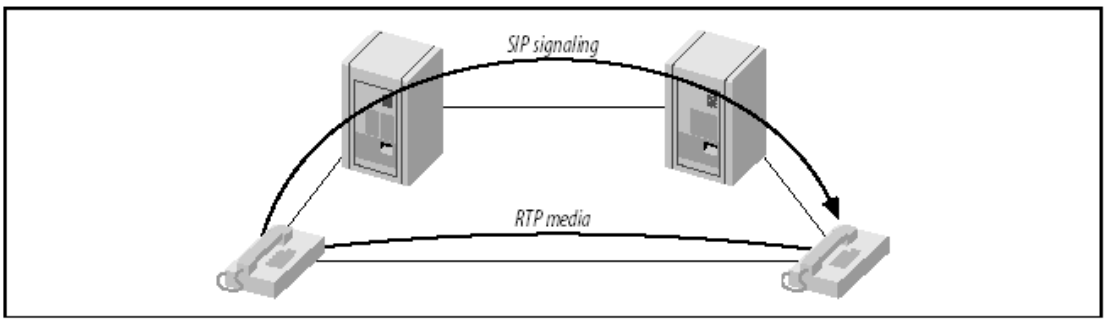


图4-2　SIP梯形

SIP不是第一个，也不是唯一我们今天使用的VOIP协议（其它包括H.323，MGCP,IAX等等），但是目前SIP似乎是硬件厂商最大的动力。SIP协议的优点是普遍的被接受和结构灵活（并且，我们经常说，简单）。

SIP 配置

下面是基本的*sip.conf* 文件:

```
[general]
context=default
srvlookup=yes
[john]
type=friend
secret=welcome
qualify=yes ; Qualify peer is no more than 2000 ms away
nat=no ; This phone is not natted
host=dynamic ; This device registers with us
canreinvite=no ; Asterisk by default tries to redirect
context=internal ; the internal context controls what we can do
```

sip.conf 文件开始是一个 [general] section，包含通道设置，所有用户和对端的默认选项。在一个端－用户/端的基础上，你可以不考虑默认的配置，而在user/peer定义中重新进行配置。

域名系统服务记录 (DNS SRV records)是建立一个逻辑和可解析地址的方法，你可以到达这个地址。这种方法可以是呼叫传输到不同的地点，而不需要改变逻辑地址。使用SRV记录，你可以获取DNS的很多好处，然而禁止它们打破SIP RFC的规定，并且禁止了基于域名的SIP呼叫。（注意如果多个记录返回，Asterisk只会使用第一个）。

DNS SRV 记录查询在Asterisk默认禁止，但是强烈建议你打开。启动使用*sip.conf*中[general]section里面的 srvlookup=yes。

每个连接都会定义为用户，端或者朋友。用户类型是用于认证呼入呼叫。端类型用于呼出呼叫。朋友类型两种都用。扩展名用中括号定义([])。本例中，我们已经定义了一个端点为朋友。

Secret是用于认证的密码。我们的secret是welcome。我们可以监视Asterisk服务器和电话之间的延时，使用qualify=yes，确认远端设备是否可达。qualify=yes可以用于监视任何远端设备，包括其它的Asterisk服务器。默认情况是Asterisk认为时延在2,000 ms (2 seconds)以内的设备可达。你可以配置Asterisk判断对端是否可达的时间，通过将yes替换为毫秒。

如果一个号码在网络地址翻译（NAT）设备后面，例如路由器或者防火墙，配置nat=yes，强迫Asterisk忽略号码的联系信息，使用收到的包的地址信息。设置host=dynamic将要求号码注册，以便Asterisk可以知道如何找到电话。将一个端点绑定到一个单独地址或者fullyqualified domain name (FQDN)，将dynamic替换IP地址或者域名。注意这只是限制你呼叫的目的地址，用户允许从任何地点发起呼叫（假定它成功鉴权）。如果你设置host=static，则终端设备不需要注册。

我们已经设置了canreinvite=no。在SIP协议，邀请用于发起呼叫，重定向媒体。在初始邀请后相同对话中发起的任何邀请都被视作重邀请（reinvite）。例如，假设双方正在交换媒体信息。如果一方呼叫等待，Asterisk设置为播放呼叫等待音乐，Asterisk会发起到第二个客户端的重邀请，告诉将他的媒体流定向到PBX。然后就可以传输音乐流，或者一个等待客户的通知

最开始的用户再到PBX的重邀请中发起一个摘机的命令，然后PBX发起一个重邀请到第二方，要求它将媒体流重新定向到发起方，结束呼叫等待音乐，重新连接客户端。

通常，当两个端点建立起呼叫，他们直接从一端到另一端传输媒体。Asterisk通常在媒体通道打破这个规则，允许它听到电话键盘上的拨号音。这是必须的，因为如果Asterisk不能确定呼叫长度，计费将不准确。配置canreinvite=no让Asterisk媒体通道经过自己，而不允许RTP信息直接在端点之间传送。

Asterisk在以下的任何情况下都不会发起重邀请：

如果客户端的任何一方配置为canreinvite=no；

如果客户端不能协商编码，Asterisk需要执行语音编码转换；

如果客户端的任何一方配置为 nat=yes；

如果Asterisk在呼叫中需要监听双音多频 (DTMF)音 (用于呼叫前转或者其他功能)。

最后context=internal定义了指令的地点，用于控制电话的权限，以及如何处理此号码的呼入呼叫。*sip.conf 中设置的*context名称和*extensions.conf*中的匹配。更多的信息，参阅以后章节。

如果你配置了相似的一组用户，你可以使用相似的命令在[general]字段。Asterisk会使用默认的定义，除非他们在用户的配置文档中明确改变。

客户端配置

由于不可能为所有的端点设备显示所有可能的配置，我们觉得至少提供一个免费的软电话的配置将会有所帮助，你可以用这个软电话测试你的系统是否配置正确。我们选择使用X-ten的X-Lite 客户端，你可以从他们的网站下载(*http://www.xten.com*).

配置客户端通常简单。最重要的是注册的用户名和密码，和你要注册的Asterisk服务器的地址。图4-3显示了一个X-Lite客户端的示例配置。确认修改各个值，符合你的配置需要。

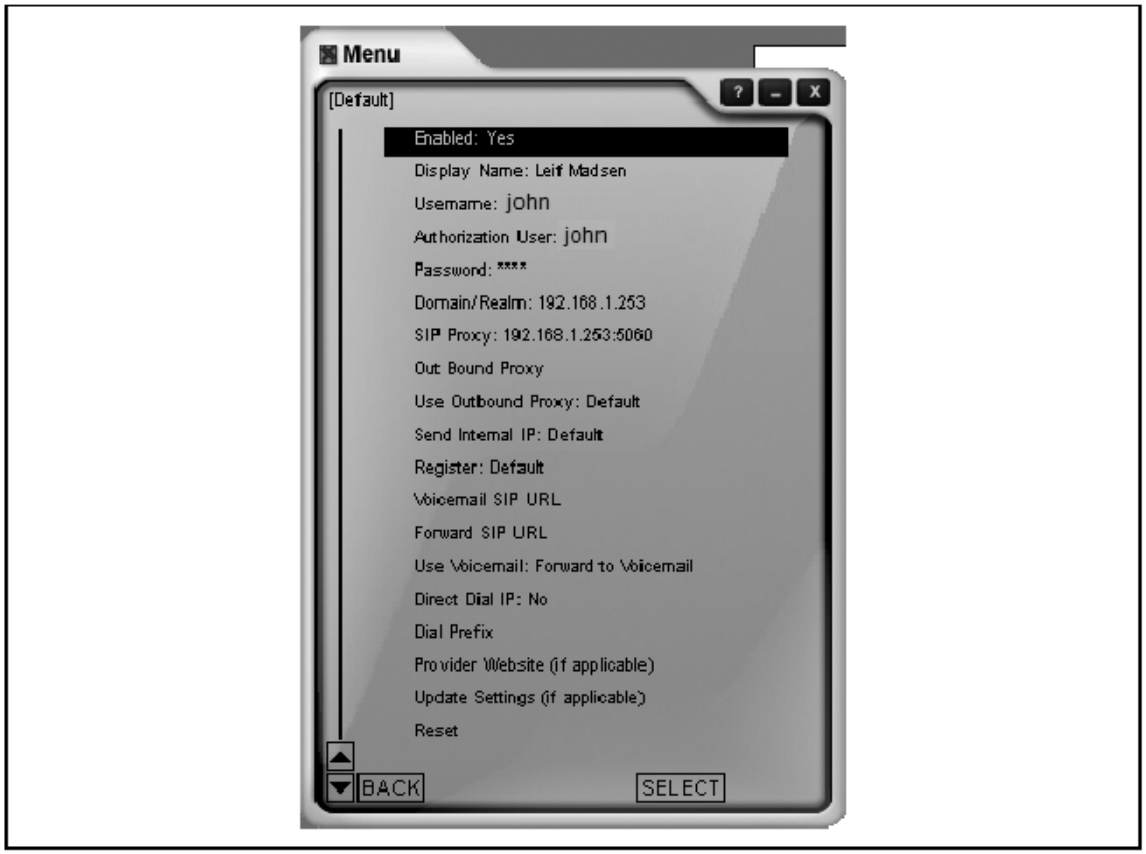


图4-3 X-Lite客户端示例

显示名是用于来电显示。用户名、鉴权用户和密码用于鉴权。domain/realm 是Asterisk IP 或者 FQDN地址。SIP代理服务器和domain/realm输入的一样，但是增加: 5060（定义SIP信令的端口号-和你在*sip.conf配置的相同*）

输入所有信息后，检查Enabled 设置为yes, 然后关闭配置菜单。X-Lite会注册到 Asterisk。如果X-Lite不注册，则重启客户端。因为X-Lite在你关闭X 按钮时会在任务栏最小华，你需要右击图标，然后点击 “Exit” 。

Dialplan 配置

很多SIP电话，包括软件和硬件，都是多线程电话，意味着可以接受多个同时呼入的电话。这样，测试你的X-Lite 软电话你可以呼叫你自己，然后呼叫在Asterisk环回，两个在线用户端。呼叫你自己，拨打100。如果你选择的客户端不支持多线功能，你可以使用611号码进入Echo( )应用

```
[internal]
exten => 100,1,Dial(SIP/john)
exten => 611,1,Echo( )
```

4.7　配置入局 IAX 连接

IAX (Inter-Asterisk eXchange)协议通常用于服务器间通信，就 sip 协议来说，硬件电话支持的更多些。然而，一些软电话已经支持 IAX 协议，并且有人正在致力于研究硬件电话对此协议的支持。IAX 和 SIP 协议最大的区别在于媒体（语音）在端点之间传输的方式不同。

SIP 协议中，相对于使用信令模式来说，RTP(媒体)流使用不同的端口进行传输。例如，Asterisk 默认的情况是在 5060 端口接收 SIP 信令，在 10000-20000 端口接收 RTP(媒体)流。IAX 协议则不同，所有的信令和媒体流都通过一个端口 4569 端口进行传输。这种方式的好处是 IAX 协议能更适合在有 NAT 相关拓扑的应用。

IAX 用户习惯对进入 PBX 系统的呼叫进行鉴权和处理。对从 PBX 系统呼出的呼叫，Asterisk 应用 IAX 的 iax.conf 文件的端点进入（条目）对远端进行鉴权。（IAX 端点将在“设置出局 IAX 连接”一章进行探讨）

本章说明如何通过 IAX 为你的系统设置一个 FWD（全球免费拨打 Free World Dialup）账号。FWD 是一个免费的 VOIP 运营商，允许你免费拨打系统内的任何号码，不受物理位置的限制。FWD 同时连接了一百多个其它你可以免费拨打的网络。


激活 IAX2 对你 FWD 账号的支持，访问

http://www.fwdnet.net/index.php?section_id=112.

本章讲述如何创建 iax.conf 和 extensions.conf，使你可以接受从其它 FWD 用户发出的呼叫。呼出 IAX 连接的章节讲述如何发起呼叫。

iax.conf 配置

在 iax.conf 文件中，段落是由方括号定义的([])。每个 iax.conf 文件至少需要一个主要段落: [general]。在这个[general]段落中，定义使用 IAX 协议的相关设置，例如默认编码和抖动缓冲。你可以不考虑在[general]段落中默认编码，而在 user or peer 定义中进行设置。

以下的[general]段落是 iax.conf.sample 配置文件（示例安装的文件）中的默认设置。更多相关信息，请参阅附件 A。

```
[general]
bandwidth=low
disallow=lpc10
jitterbuffer=no
forcejitterbuffer=no
tos=lowdelay
autokill=yes

register => fwd_number:password@iax2.fwdnet.net

[iaxfwd]
type=user
context=incoming
auth=rsa
inkeys=freeworlddialup
```

在[general]段落中，你需要增加一个 register 陈述。这个 register 陈述的目的是通知 FWD IAX 服务器你在互联网的位置（你的 IP 地址）。当呼叫你的 FWD 号码时，FWD 服务器在数据库中搜寻，并且将呼叫发送到和 FWD 号码关联的

IP 地址。

在 [iaxfwd] 段落，通过 type=user 为用户定义呼入呼叫。然后用 context=incoming 进行呼入呼叫的鉴权。公钥用 inkeys=freeworlddialup 进行定义。freeworlddialup 公钥是 Asterisk 的标准。

Dialplan 配置

在 extensions.conf 文件中处理呼入呼叫十分简单。首先，建立一个 context 命名为 incoming（和 iaxfwd 用户在 iax.conf 里面设置的 context 名称相同）。Context 后面是 Dial( )陈述，拨号本章前面创建的 sip 分机。用你的 FWD 账号替换号码 10001。

```
[incoming]
exten => 10001,1,Dial(SIP/john)
```

4.8 配置出局 IAX 连接

当 IAX 用户收到呼入呼叫，另一个 IAX 用户正在呼出。本章讲述建立 iax.conf 和 extensions.conf，使你可以呼出呼叫。

iax.conf 设置

下列 iax.conf 中的条目在 FWD 网络中用于呼出：

```
[iaxfwd]
type=peer
host=iax2.fwdnet.net
username=<fwd-account-number>
secret=<fwd-account-password>
qualify=yes
disallow=all
allow=ulaw
```

allow=gsm

allow=ilbc

allow=g726

端点通过 type=peer 进行定义。使用 host 设置用于呼叫的服务器(iax2.fwdnet.net)。你的 FWD 账户号码和密码用来进行 FWD 网络的鉴权，分别定义为 username 和 secret。

你可以使用 qualify=yes 陈述不时检查远端的服务器是否响应。响应时间（延时）可以用 ax2 show peers 通过 Asterisk 控制台进行察看。默认远端 2000MS(2秒)后视为不可达。你可以自己设定时间，将 yes 替换为你要的毫秒数。

每个端点可以设置自己的可用的编码和优先级。disallow=all 用于复位原来设置的所有编码信息。然后可以使用 allow=codec 设置你支持的编码和优先级（从高到低）。

在 Asterisk CLI 使用 iax2 show registry 命令，检验你是否成功注册。

Dialplan 设置

在 extensions.conf 定义一个 section，使我们能够将某个呼叫设置 FWD 回声测试应用。和前面的设置一样，我们要创建一个 CONTEXT，附带连接 FWD 回声测试的命令。使用连接 FXS 端口的电话，或者 SIP 电话，拨打 613，进行呼叫。

```
[internal]
exten => 613,1,Dial(IAX2/iaxfwd/613)
```

4.9 调试

Asterisk 提供了多种调试方法。连接控制台后，可以察看不同级别的冗长和调试输出，同时还有协议包 tracing。我们看一下本节不同的选项。(The Asterisk 控制台详细信息请参考附件 E)。

连接控制台

连接 Asterisk 控制台，你可以在控制台直接启动服务器（这样你如果不结束 Asterisk 进程就不能退出控制台），或者后台启动 Asterisk，然后远程控制台登录。

直接在控制台启动 Asterisk，使用下面的控制台命令：

 # /usr/sbin/asterisk －c

远程连接控制台，首先启动后台，然后连接：

 # /usr/sbin/asterisk

 # /usr/sbin/asterisk －r

如果遇到某个模块不能载入的问题，或者某个模块导致 Asterisk 不能载入，用－c flag 启动 Asterisk 进程，监视模块载入状态。例如，如果你试图载入 OSS 通道驱动（此驱动启动控制台通道），并且 Asterisk 不能打开/dev/dsp，你会收到如下的错误信息：

WARNING[32174]: chan_oss.c:470 soundcard_init: Unable to open /dev/dsp: No such file or directory

== No sound card detected -- console channel will be unavailable

== Turn off OSS support by adding 'noload=chan_oss.so' in /etc/asterisk/modules.

Conf

启动 Verbosity 和 Debugging

Asterisk 可以以 WARNING，NOTICE 和 ERROR 消息形式输出调试信息。这些消息可以为你提供系统信息，例如注册，状态，呼叫进程，和多种其它信息。请注意 WARNING 和 NOTICE 消息不是错误；然而，ERROR 信息需要进行检查。激活不同级别的 verbosity，使用 set verbose，附加一个数值。有效值从 3-10，例如设置 verbosity 的最高数值，使用：

 # set verbose 10

你也可以通过设置调试级别激活核心调试信息。激活控制台的 DEBUG 输

出，需要在 logger.conf 文件中增加 debug 到 console =>文段，例如：

console => warning,notice,error,event,**debug**

设置调试级别的有效值为 3-10，例如：

set debug 10

4.10 结束语

如果你已经学习了本章的所有小节，你就可以设置一对模拟接口，一个本地 SIP 通道连接软电话，还有使用 IAX2 建立了到 Free World Dialup 的连接。这些配置都是十分基础的配置，但是为我们建立了有用的通道。在以下的章节中，我们会使用这些配置，同时学习如何建立更有用的拨号方案。

第5章　拨号方案基础

任何事情都要尽可能简单，但绝对不是更简单。

阿尔伯特•爱因斯坦(1879-1955)

拨号方案是 Asterisk 系统的核心，因为它定义了 Asterisk 如何处理来话和去话。简单说来，它由指令和步骤列表组成，Asterisk 按步骤来执行这些指令。与传统电话系统不同，Asterisk 的拨号方案是完全可定制的。为了能够成功的设立您自己的 Asterisk 系统，您需要理解拨号方案。

如果编写拨号方案是一个繁重的任务，那么不要担心。本章将逐步解释拨号方案的工作原理并教您建立拨号方案所必要的技术。本章的例子被设计成后面的以前面的为基础，所以在有些内容不太明白的时候可能需要您回头看一下前面的章节。另外还要注意的是，本章并不是拨号方案的完整说明，它所能做的事情远不止本章所介绍的，我们的目的只是要把基本原理介绍给你。我们将在后面的章节中介绍拨号好方案的更高级的内容。

5.1　拨号方案语法

Asterisk 的拨号方案在文件 extensions.conf 中定义。

文件 extensions.conf 通常在/etc/asterisk/目录下。但是其位置可以改变，这取决于 Asterisk 的安装方式。其它常见的位置包括/usr/local/asterisk/etc/和/opt/asterisk/etc/。

拨号方案由 4 部分组成：contexts、extensions、priorities 和 applications。在下面的几节内，我们将介绍每一个部分并解释如何使用它们来建立拨号方案。在介绍了每一种元素在拨号方案中所扮演的角色之后，我们带领您进入创建具有一定功能的基本拨号方案的过程。

配置文件实例

在安装 Asterisk 的时候，如果你安装了样例配置文件的话，你就会有 extensions.conf 存在。我们建议你用自己的 extensions.conf 文件，而不是用样例配置文件来启动。这样做非常有利，会让你对拨号方案的概念和原理有更好的理解。

这就是说，样例配置文件 extensions.conf 仍旧保留那些稀奇古怪的资源，所有的例子和思想都可以在你了解了基本的拨号方案的概念和原理之后去使用。我们建议你把样例配置文件改一个名字，例如 extensions.conf.sample。这样，您可以在将来把它用作参考。你也可以在 Asterisk 原文件的/configs/目录内找到这些样例配置文件。

Context

Dialplans 被分成几个段，这些段称为 context。Context 用来对 extension 的组命名。简单说来，它把拨号方案的不同部分进行分离，免得彼此交织在一起。在一个 context 中定义的 extension 完全独立于在另一个 context 中定义的 extensiont，除非明确允许有交织。(在本章的末尾，我们会介绍如何允许 context 之间相互交织。)

作为一个简单的例子，我们来假定有两个公司共用一个 Asterisk 服务器。如果把每个公司的语音菜单放在各自的 context 中，就可以有效的区分这两个公司。比如说，在拨 extension 0 的时候，能够分别定义将要发生的情况：在公司 A 的语音菜单内按 0，可以转接到公司 A 的接线员，而在公司 B 的语音菜单内按 0，可以转接到公司 B 的接线员(这个例子假设在主叫按下 0 的时候，我们让 Asterisk 把呼叫转接到接线员)。

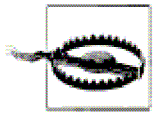
Contexts 的表示方法是把名字放在方括号([])的中间。这个名字可以由 A -Z (大小写都可以)、数字 0-9，以及连字号和下划线组成[*]。例如，一个来话的 context 看起来像：

[*] 请注意，空格很明显不在允许的字符里面。在 context 中不要使用空格。您肯定不会喜欢出现不良后果。

```
[incoming]
```

所有放在 context 定义之后的指令都是这个 context 一部分，直到下一个 context 定义的开始。 在拨号方案的开始，有两个特别的 context，名字分别为[general]和[globals]。我们会在本章的稍后一点来讨论[globals] context。现在只要知道这是连个特别的 context 就够了。

context 的一个重要用途就是加强安全性。正确使用 context，可以让某些主叫方能够访问别人不能访问的功能（比如打长途）。如果没有仔细设计拨号方案，可能会造成别人盗用你的系统的不良后果。在建立你的 Asterisk 系统时，请牢记这一点。



Asterisk 源程序包含一个非常重要的文件，叫做 SECURITY，它该括了几个步骤来保证你的 Asterisk 系统的安全。阅读和理解这个文件相当的重要。如果你忽视了所给出的安全预防建议，您可能要承担这样的后果：任何人都会用您的名义来进行长途通话，您要替他们承担这些费用！

如果您没有认真地对 Asterisk 系统采取安全措施，您可能就要付出相当的代价！请务必花时间和精力来防止您的系统遭到话费欺诈。

Extension

在每一个 context 内，可以定义一个或者多个 extension。 *extension* 是 Asterisk 要执行的指令，由来电或者通道上所拨数字来触发。Extension 规定了在呼叫通过拨号方案是，对其采取哪些处理。尽管 extension 可以在传统意义上被用来定义电话分机， (例如，please call John at extension 153)，但是在 in Asterisk 上还是有更多用途。

extension 的语法是单词 exten 后面跟着一个由等号和大于号组成的箭头，如下所示：

exten =>

之后是 extension 的名字。在与电话系统打交道的时候，我们把 extension 看作是呼叫另一部电话所拨的号码。在 Asterisk 上，还意味着更多的东西，例如，

extension 的名字可以是字母和数字的组合。在本章和后续的章节中，数字和字母的 extension 我们都会用到。


给 extension 分配一个名字似乎是一个革命性的概念，但是当你看到许多的 VoIP 传输支持（甚至是积极地鼓励）使用名字或者邮件地址拨号，而不是使用数字拨号，这就非常有意义了。这是 Asterisk 如此灵活和功能强大的功能特性之一。

一个完整的 extension 由三部分组成：

- Extension 的名字或者号码
- Priority（每个 extensio 可以有多个步骤，步骤的编号称作 Priority）
- 应用(或者命令)，针对呼叫完成一些动作

这三个部分用英文逗号分开，如：

exten => *name,priority,application( )*

下面是一个简单的例子，让我们来看一下实际的 extension 是个什么样子：

exten => 123,1,Answer( )

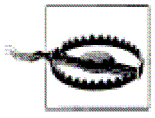
在这个例子中，extension 的名字是 123，priority 是 1，应用是 Answer( )。现在，我们进一步来解释 priority 和应用。

Priority

每个 extension 都可以有多个步骤，称作 priorities。每个 priority 都按顺序编号，从 1 开始 (实际上，这个规定有个例外，如在 Unnumbered Priorities Actually 所介绍的那样)。每个 priority 执行一个规定的应用。下面给出一个例子，这个 extension 接听电话（编号为 1 的 priority），然后挂断（编号为 2 的 priority）：

exten => 123,1,Answer( )

exten => 123,2,Hangup( )


你必须确保 priority 从 1 开始并且是连续的编号。如果中间跳过

一个 priority，Asterisk 有可能不会越过它。如果在给定的 extension 中发现 Asterisk 没有遵从 priority，您要确认是否跳过了某个序号或者标号有混乱。

如果您还没有理解 Answer( )和 Hangup( )，请不要着急，我们将在后面的章节中介绍它们。关键一点是要记住，对于特定的 extension，Asterisk 遵从 priority 的数字顺序。

应用（Application）

应用是拨号方案中的劳动力。每个应用针对当前通道完成规定的动作，比如播放声音、接受音频拨号输入，或者挂断电话等。在前面的例子里，您已经看到了两个简单的应用： Answer( )和 Hangup( )。您很快就会了解它们是如何工作的。

无序号的 Priority

在 Asterisk 1.2 版本里，Priority 编号增加了新的变化，引入了 n priority，表示“下一个”的意思。每次 Asterisk 遇 n 这个 priority 的时候，就取出前一个 priority 的编号加上 1。这样就使得拨号方案修改起来很方便，不必要去记忆每一步的编号。例如西面这个拨号方案的例子：

```
exten => 123,1,Answer( )
exten => 123,n,do something
exten => 123,n,do something else
exten => 123,n,do one last thing
exten => 123,n,Hangup( )
```

1.2 版也允许给 priority 分配一个文字标号。要给 priority 分配文字标号，只需要在 priority 后面的括号内加上这个标号，如：

```
exten => 123,n(label),do something
```

下一章会介绍如何根据拨号方案的逻辑在不同的 priority 之间进行跳转。

某些应用，如 Answer( )和 Hangup( )，完成其任务不需要其他的指令，而另外一些应用却要求附加的信息。这些信息，称为参量，可以传递给应用来影响这

些应用如何完成他们的任务。 为了给这些应用传递参量，把参量放在紧跟在应用名称后面的括号内，多个参量之间用英文的逗号分隔。

 有时候，你也能够看到管道符(|)当作操作符用在参量之间来使用，而不是使用逗号。本书的例子中，我们在应用的参量之间使用逗号。

我们在下一节建立第一个拨号方案的时候，您将学到如何使用应用（及其相关的参量）来获得更好的功能效果。

5.2 一个简单的拨号方案

现在我们准备来创建第一个拨号方案。我们以一个简单的例子来开始。要设计的这个拨号方案，在呼叫进来时，Asterisk 应答这个呼叫，播放声音文件，然后挂断。我们用这个简单例子来指出最重要的拨号方案基本原理。

为了使得本章中的例子工作正确，我们假定至少创建了一个 Zap 通道，并且作了配置(按照前面一章的说明)，所有进来的呼叫被送到[incoming]context。如果使用了其他类型的通道，则需要调整这些例子以便适合您具体的环境。

s Extension

在着手拨号方案之前，我们要先来介绍一个特别的 extension ，名字叫作 s。当没有指定 extension 的呼叫(例如，正在振铃的 FXO 线路)进入 context 的时候，就由这个 s extension 来处理。（s 表示"start"-开始，因为多数的呼叫都是从 s extension 开始）。因为这正是我们这个拨号方案所需要的，我们来 let's begin to fill in the pieces。我们要在呼叫上完成三个动作（接听、播放声音文件，最后挂断），所以要创建有三个 priority 的 s extension。我们把这三个 priority 放到[incoming]里，所有来话都应该从这个 context 开始：

```
[incoming]
exten => s,1,application( )
exten => s,2,application( )
```

exten => s,3,*application( )*

现在我们所要做的事情就是填写应用，到此为止，我们已经创建了第一个拨号方案。

Answer( ), Playback( ), 和 Hangup( )应用

假如我们要接听一个呼叫，播放语音文件，然后挂断的话，我们就要学习如何做到这一点。Answer( ) 应用用于接听正在振铃的通道。这并不为接受来话的通道进行初始化设定。 (确有少数应用不需要先应答通道，但是在完成其它动作之前进行适当的应答是一个很好的习惯。) 如同前面提到的，Answer( ) 不需要任何参量。

Playback( ) 应用用于在通道上播放事先录制好的语音文件。在使用 Playback( )应用时，系统不会理会来自用户的输入。

Asterisk 带有很多专业录制的语音文件，它们在默认的语音文件目录内(通常是/var/lib/asterisk/sounds/)。这些文件是 GSM 格式，因此扩展名是.gsm。我们会在很多例子中使用这些文件。李子中所使用的几个文件来自 asterisk-sounds 模块，所以请您要花时间把它安装上（见第 3 章）。

若要使用 Playback( )，要指定一个文件名(不要带扩展名)作为参量。例如，Playback(filename) 将播放文件名为 filename.gsm 的语音文件，并假定这个文件位于默认的语音文件目录内。要特别说明的事，您也可以在文件名上包括完整的路径，例如：

Playback(/home/john/sounds/filename)

这个例子将会播放/home/john/sounds/目录下的 filename.gsm 文件。也可以使用相对于 Asterisk 语音文件目录的相对路径：

Playback(custom/filename)

这个例子将会播放 Asterisk 语音文件目录内 custom/子目录中的 filename.gsm 文件。注意，如果所指定的目录包含多个使用不同扩展名的文件，Asterisk 会自动选择播放最佳的文件。 [*]

[*] Asterisk 根据转换代价来选择最佳文件，就是说，它选择占用 CPU 时间少的文件莱把它转换成本地的音频格式。在启动 Asterisk 的时候，它会计算不同于音格式之间的转换代价（一般来说，系统不同，计算出的结果也不同）。您可以在 Asterisk 命令行接口上输入命令 **show translation** 来查看这些转换代价。在第 8 章，我们会介绍有关不同音频格式（语音编解码）的更多内容。

Hangup( )应用完成其名字所意味的动作：挂断一个正在活动的通道。主叫方将收到通话挂断的指示。当年要结束当前通话的时候，您可以在 context 的末尾使用这个应用，以确保主叫不会继续停留在拨号方案内。这个应用不需要参量。

第一个拨号方案

我们现在已经创建 extension，并给了它 3 个不同的 priority，也了解了我们打算使用的应用。现在，把所有这些东西放在一起，创建第一个拨号方案。遵循许技术书籍（特别是计算机编程方面）的习惯，我们也把这第一个例子称为"Hello World!"

在这个 extension 的第 1 个 priority 里面，我们接听呼叫；在第 2 个里面，我们播放名字为 hello-world.gsm 的语音文件；在第 3 个里面，我们将通话挂断。下面就是这个拨号方案：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Playback(hello-world)
exten => s,3,Hangup( )
```

如果您已经有 1、2 个通道作了配置，那么就去试一下！用这个短小的拨号方案创建一个新的 extensions.conf。 (如果没有正常工作，检查一下 Asterisk 控制台输出的错误信息，确认你的通道是否把来话配置成送入[incoming] context)。

尽管这个例子短小而且简单，但是他强调了 context、extension、prioritie 和应用这些核心概念。至此，我们已经介绍了这些基本概念，并且作了一个例子。不过，说起来，一个简单的接听呼叫、播放语音，然后挂断的电话系统却不是那么有用。

5.3 在拨号方案中加入逻辑

我们刚刚建立的拨号方案是静态的，对每个呼叫总是做相同的事情。现在我们要给这个拨号方案添加一些逻辑，让它根据用户的输入来完成一些不同的动作。首先再介绍几个应用。

Background( )和 Goto( ) 应用

构建交互式 Asterisk 系统的关键是 Background( )应用。与 Playback( )相同的是，它也播放事先录制好的语音文件；与 Playback( ) 不同的是，当主叫方按下电话键（1 个或者多个）的时候，会中断语音的播放，转到与所按数字对应的 extension。例如，假设主叫方按下 5，Asterisk 停止播放语音，把呼叫的控制发送给 extension 5 的第一个 priority。

Background( )应用通常用于创建语音菜单(也称作自动话务员或者电话树)。很多公司使用语音菜单来引导主叫方到适当的分机，以免得接待员接听每一个电话。

Background( ) 的语法与 Playback( )类似：

```
exten => 123,1,Background(hello-world)
```

另外一个非常有用的应用是 Goto( )。如同其名字所表现的，它用于把呼叫发送到另一个 context, extension, 以及 priority。Goto( )应用使得在拨号方案的不同部分有序的转移非常容易。The syntax for the Goto( ) 应用的语法 calls for us to pass the destination context, extension, and priority as arguments to the application, like this:

```
exten => 123,1,Goto(context,extension,priority)
```

在下一个例子中，我们会用 Background( )和 Goto( )应用来建立一个略微复杂的拨号方案，允许主叫方利用号码盘上的按键与系统进行交互。首先利用 Background( )接受来自主叫方得输入：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
```

在这个例子中，播放一个样例语音文件，文件名为 enter-ext-of-person.gsm。虽然它不是很适合作为自动话务员的问候语，但是它肯定适合于这个例子。现在加入两个 extension，有主叫方根据语音提示所输入的 1 或 2 来触发：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits /1)
exten => 2,1,Playback(digits/2)
```

在继续之前，我们回顾一下到目前为止我们所做的事情。当用户呼叫进入拨号方案，会听到问候语“请输入分机号码”。如果按下 1，会听到数字 1，如果按下 2，会听到数字 2。。尽管这是一个不错的开端，我们还是要对它做一点修饰。使用 Goto( )应用来让拨号方案在回放数字之后，重复问候语。

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits/1)
exten => 1,2,Goto(incoming,s,1)
exten => 2,1,Playback(digits/2)
exten => 2,2,Goto(incoming,s,1)
```

新加入的这两行（重黑体）在回放所选择数字之后，把呼叫控制发送给 s extension。

如果你查看了 Goto( )应用的详细说明，会发现实际上可以传递 1、2、3 参量给这个应用。如果传递的是一个参量，它会认为被叫地的 priority 在当前的 extension.内；如果传递了两个参量，则当 extension 和 priority，转向当前的 context。

在这个例子中，出于清晰的考虑，传递了三个参量，如果只传递两个的话，效果也是一样的。

非法输入和超时的处理

现在，第一个语音菜单已经完成，我们在加入几个附加的特殊 extension。首先，需要一个用来处理非法输入的 extension，从而在主叫方按下一个无效输入(比如在上面的例子中按下 3)，呼叫被送到 i extension。其次，需要一个 extension 来处理主叫方没有及时（默认的时间是 10 秒）输入的情况。如果主叫方在 Background( )完成语音文件播放之后很久才按键，呼叫将被转移到 t extension。下面是增加了这两个 extension 之后的拨号方案：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits/1)
exten => 1,2,Goto(incoming,s,1)
exten => 2,1,Playback(digits/2)
exten => 2,2,Goto(incoming,s,1)
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )
```

使用 i 和 t extension 使得这个拨号方案更加健壮一些，更加用户友好一些。但说起来，它仍旧功能有限，因为外面的主叫无法与实际的人联系。为了做到这一点，我们还需要了解另外一个应用，叫做 Dial( )。

使用 Dial( )应用

Asterisk 的一个最有价值的功能是把不同的主叫方连接到一起。在主叫使用不同的通信方式时，这一功能尤其有用。例如，主叫 A 使用标准的模拟电话网络来通信，而用户 B 可能坐在地球上的某一个咖啡馆里，使用的是 IP 电话。幸运的是，Asterisk 承担了大多数在不同种类网络之间进行转换和连接的艰苦工作。您所要做的就是学习如何使用 Dial( )应用。

Dial( ) 应用的语法比我们用过的几个其它应用略复杂一些，但是不必要畏惧。Dial( ) 需要 4 个参量。第 1 个是呼叫的被叫地，由呼叫所采用的（传输）

技术、反斜线、远地资源（通常是通道名称或编号）等组成。例如，假定我们要呼叫名字为 Zap/1 的 Zap 通道（连接了普通模拟电话的 FXS 通道），那么技术是 Zap，资源是 1。与此类似，到一个 SIP 设备的呼叫的被叫地是 SIP/1234，而到一个 IAX 设备的呼叫的被叫地是 IAX/fred。假如在 extension 123 到达拨号方案时，要 Asterisk 对 Zap/1 通道振铃，要加入下面这个 extension：

exten => 123,1,Dial(Zap/1)

当这个 extension 被执行时，Asterisk 会使连接在通道上 Zap/1 的电话振铃。如果电话被接听， Asterisk 会桥接 Zap/1 通道的来电。也可以同时拨多个通道，方法是用&符号把多个被叫地连接起来，如下所示：

exten => 123,1,Dial(Zap/1&Zap/2&Zap/3)

Dial( )应用会桥接来电，无论被叫地中的哪一个通道先接听。

Dial( )应用的第 2 个参量是超时，单位为秒。如果给定了超时参量，Dial( ) 会一直对被叫地进行呼叫，直到超时后才放弃，然后转移到该 extension 中的下一个 priority。如果没有指定超时时间，Dial( )会一直呼叫该通道，直到有人接听，或者主叫挂机。现在我们把这一个 10 秒的超时加到 extension 中：

exten => 123,1,Dial(Zap/1,10)

如果呼叫在超时之前被接听，通道就被桥接，拨号方案完成，如果被叫地没有应答，Dial( ) 会继续到该 extension 的下一个 priority。但是，如果被叫通道忙，Dial( )将转到 priority n+101，如果其存在的话（其中的 n 是 Dial( )被调用的 priority）。这样我们就能够以不同于被叫地忙的方式来处理未接听电话。

现在我们把所学到的内容做成另外一个例子：

exten => 123,1,Dial(Zap/1,10)

exten => 123,2,Playback(vm-nobodyavail)

exten => 123,3,Hangup( )

exten => 123,102,Playback(tt-allbusy)

exten => 123,103,Hangup( )

正如您所看到的，在这个例子中，如果呼叫未被接听，将播放 vm-nobodyavail.gsm 语音文件，如果 Zap/1 通道正忙，则播放 tt-allbusy.gsm 语音文件。

Dial( )应用的第 3 个参量是可选择的字符串。它包含一个或多个能够影响 Dial( )应用行为的字符。 选项地列表太长，不能在这里逐一列出。最常用的一个是字母 r。如果把字母作为第 3 个参量，在通知被叫通道有来电的这段时间内，主叫方会听到振铃声音。

应该注意到在指示振铃时，r 选项不总是必需的，因为在建立通道时 Asterisk 会自动产生振铃音。不过，即使没有试图连接的时候，也可以使用 r 选项强制 Asterisk 指示振铃。为了把 r 选项加到最后这个例子中，只要简单的修改第一行：

```
exten => 123,1,Dial(Zap/1,10,r)
exten => 123,2,Playback(vm-nobodyavail)
exten => 123,3,Hangup( )
exten => 123,102,Playback(tt-allbusy)
exten => 123,103,Hangup( )
```

此时，由于已经知道如何使用 Dial( )应用，拨号方案中编号为 1 和 2 的 extension 变得没有用了。我们把它们用 extension 101 和 102 来替代，这样就允许外部的主叫方把电话打给 John 和 Jane：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(SIP/Jane,10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
```

exten => t,1,Playback(vm-goodbye)

exten => t,2,Hangup( )

Dial( )应用的最后一个参量是 URL。如果被叫通道支持在呼叫的同时接受 URL，那么所指定的 URL 将被发送(例如，如果你有一个能够接受 URL 的 IP 电话，它将会显示在显示屏上，同样，如果使用的是软电话，URL 会在屏幕上弹出来。) 这个参量很少使用。

如果要在 FXO Zap 通道上产生一个去话，可以使用下面的语法在那个通道上拨号：

exten => 123,1,Dial(Zap/4/5551212)

这个例子在 Zap/4 通道上拨号码 555-1212。对于其它类型的通道，如 SIP 和 IAX，简单的把被叫地作为资源，见下面两行：

exten => 123,1,Dial(SIP/1234)

exten => 124,1,Dial(IAX2/john@asteriskdocs.org)

注意：任何的参量都可以为空。例如，假设要指定一个选项，但不是超时，只要保留超时参量为空，如下所示：

exten => 123,1,Dial(Zap/1,,r)

给内部呼叫增加 Context

迄今为止所给出的例子，还一直限制在只有一个 context，但是可以想见，几乎所有的, Asterisk 安装在其拨号方案中都会有不止一个的 context。在本章的开始我们曾经提到，context 的一个重要功能是为不同的主叫用户区分特权(如长途通话，或者呼叫特定的 extension)。在下一个例子中，我们建立两个内部电话 extension，加到拨号方案里去，然后配置这两个 extension 可以彼此呼叫。为达到这个目的，创建一个新的叫做[internal]的 context。

 同前面的例子一样，仍旧假定 FXS Zap 通道 (在本例中就是 Zap/1) 已经配置好，并且配置了 zapata.conf，因此任何来自 Zap/1 的呼叫都从[internal] context 开始。对于本章末尾的几个例子，我们也将假定 FXO Zap 通道已经配置

成 Zap/4，对于来到这个通道的呼叫将被送到[incoming] context。这个通道用作去话。

我们也一直假定你有至少一个 SIP 通道 (名字为 SIP/jane) ，被配置为来自于[internal] context。. We've done this to introduce you to using other types of channels.

如果没有上述通道的硬件(比如 Zap/4)，或者使用了不同通到名字的硬件（如，不是 SIP/jane），请不必担心，您可以对例子进行修改来适合你的系统配置。

现在，拨号方案为：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(SIP/Jane,10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )

[internal]
exten => 101,1,Dial(Zap/1,,r)
exten => 102,1,Dial(SIP/jane,,r)
```

在这个例子中，我们在[internal] context 中加了两个新的 extension。这样，使用通道 Zap/1 的人可以拿起电话拨 102 来拨打通道 channel SIP/jane 上的人，同样，注册为 SIP/jane 的电话可以拨 101 来拨打 Zap/1 的电话。

在例子中使用 extensions 101 和 102 是随意确定的，您可以自由选择为 extension 使用什么样的编号约定。并没有限制一定要使用 3 位的 extension，可以多，也可以少，随您确定。 (当然，Extension 一定不能多于 80 个字符，也不能使用单个字符的 extension，这是保留的) 别忘了，你还可以使用名字，见下面：

```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(SIP/Jane,10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )

[internal]
exten => 101,1,Dial(Zap/1,,r)
exten => john,1,Dial(Zap/1,,r)
exten => 102,1,Dial(SIP/jane,,r)
exten => jane,1,Dial(SIP/jane,,r)
```

如果你认为你的用户可能通过支持名字的 VoIP 传输来拨号，添加使用名字的 extension 也不是什么不好的事情。

现在，内部的主叫者彼此之间可以呼叫，我们现在准备做一个完整的拨号方案。下面，我们来看如何使得拨号方案更加有伸缩性，更易于在未来修改。

使用变量

在拨号方案中使用变量可以帮助减少打字、增加清晰度，也有助于在拨号方案中加入逻辑。如果您有编程经验，您肯定已经知道什么是变量。否则的话，也不要担心，我们回来接式什么是变量，以及如何使用变量。

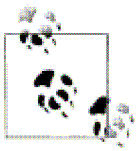
可以把变量看作是一个容器，一次能容纳一个值。例如，建立一个叫做 JOHN 的变量，并给其赋予值 of Zap/1。这样，在写拨号方案时，可以利用名字来引用 John 的通道，不需要直接记住 John 使用的是 Zap/1。为了把值赋给变量，只需要输入变量名称、等号和值，如下所示：

```
JOHN=Zap/1
```

引用变量有两种方法。若要引用变量的名字，仅仅需要输入变量的名字就可以了，例如；JOHN；但如果要引用变量的值，则必须输入美元符号，紧接着是大括号，在大括号内输入变量名。下面的例子说明了如何在应用中引用变量：

```
exten => 555,1,Dial(${JOHN},,r)
```

在拨号方案中，只要使用${JOHN}, ，Asterisk 都会自动用赋给 JOHN 的值来替代它。

 注意，变量名不一定要大写，本书中我们使用大写是为了方便阅读。

在拨号方案中有三种类型的变量可以使用：全局变量、通道变量和环境变量。下面就来看看每一种类型。

1　全局变量

如名字所说，全局变量适用于所有 context 里的所有 extensions。全局变量的好用之处在于它可以用于拨号方案中的任何地方，能够增加可读性和可管理性。

假设有这样一种情况：有一个很大的拨号方案，其中对 Zap/1 通道的引用有数百个。可以想象，您必须逐条的查看拨号方案，修改所有对 Zap/1 通道的引用。这肯定是一个错误百出的漫长过程。

另一方面，如果在拨号方案的开始定义了一个值为 Zap/1 的全局变量，然后引用它，那么只需要修改这一行。

全局变量应该在 extensions.conf 文件的开始利用[globals] context 说明(定义) 。特们也可以使用编程的方式定义，利用 SetGlobalVar( )应用。下面的例子说明了在拨号方案中使用的两种方法：

```
[globals]
JOHN=Zap/1


[internal]
exten => 123,1,SetGlobalVar(JOHN=Zap/1)
```

2 通道变量

通道变量与特定的呼叫相关的变量(如 Caller*IDnumber)，与全局变量不同，通道变量只能在当前呼叫存在其间定义，并只能用于参与该呼叫的通道。

有很多的预先定义的通道变量可以用于拨号方案，在 Asterisk 源程序的中 doc 子目录下 README 文件中有详细的说明。通道变量使用 Set( )应用来设置：

```
exten => 123,1,Set(MAGICNUMBER=42)
```

在下一章，会使用几个通道变量。

3 环境变量

环境变量是一种在 Asterisk 中访问操作系统环境变量的方法。这些变量以 ${ENV(*var*)}形式引用，其中的 *var* 是所要引用的操作系统环境变量。

4 在拨号方案中加入变量

我们已经了解了变量，现在把它们放到拨号方案中去。我们为两个人，John 和 Jane，加入变量：

```
[globals]
JOHN=Zap/1
JANE=SIP/jane
```

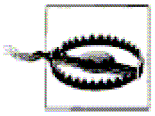
```
[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(${JOHN},10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(${JANE},10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )

[internal]
exten => 101,1,Dial(${JOHN},,r)
exten => 102,1,Dial(${JANE},,r)
```

模式匹配

在拨号方案中添加各种可能使用到的 extension 是非常繁琐的，尤其是在区划的情况下。能够想象定义了你可以拨任何号码的 extension 的拨号方案吗？幸运的是，Asterisk 有一个东西适合这种情形：模式匹配可以使用一段代码来对应许多不同的 extensions。

1　模式匹配语法

使用模式及匹配的时候，用不同的字母和符号来代表可能要匹配的数字。模式总是用一个下划线(_)开始，它告诉 Asterisk 要做模式匹配，这不是一个 extension 名字。(这意味着不能使用下划线作为 extension 名字的开始字符。)



如果在模式之前忘记了使用下划线，Asterisk 会认为它是一个命名的 extension，不会做任何模式匹配的动作。

在下划线之后，可以使用一个或者多个下面列出来的字符：

X

匹配 0-9 的任何数字。

Z

匹配 1 - 9 的任何数字。

N

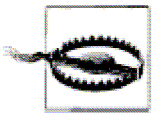
匹配 2 - 9 的任何数字。

[15-7]

匹配任何数字或者指定的数字范围。在这个例子中，匹配 1, 5, 6, 或 7。

. (句号)

通配符，匹配一个或多个字符。



如果不加小心的话，通配符可能会使得拨号网络做出你意想不到的事情。应该在尽可能匹配了其他数字之后再使用通配符。例如，下面的模式匹配应该用云不会用到：

_.

实际上，如果你试图使用它，Asterisk 会警告你。如果可能，尽量使用下面的模式：

_X.

若要在拨号方案中使用模式匹配，只要把模式放在 extension（或者号码）名字的位置：

exten => _NXX,1,Playback(auth-thankyou)

在这个例子中，模式会匹配 3 位的 extension，从 200 到 999 (N 匹配任何 2-9 之间的数字，每个 X 匹配 0 - 9 之间的数字)。这就是说，在这个 context 中，如果主叫拨 200-999 之间的任何 extension，都会听到声音文件 auth-thankyou.gsm 的声音。

有关模式匹配必须了解的一件事情是，如果 Asterisk 发现有多个模式与所拨的 extension 匹配，它会使用最接近的那一个模式。比如说定义了下面的两个模式，主叫方拨的号码是 888-555-1212：

exten => _555XXXX,1,Playback(digits/1)

exten => _55512XX,1,Playback(digits/2)

在这个例子中，会选择第 2 个 extension，因为它更接近。

2　模式匹配的实例

在继续之前，来多看看几个模式匹配的例子。对每一个例子，在阅读说明之前，看看您是否能够说出它会匹配什么。先从最简单的开始：

_NXXXXXX

看出来了？这个模式匹配任何 7 位号码，只要第 1 位为 2 或大于 2。按照北美号码方案(North American Number Plan，NANP)，这个模式匹配任何一个本地电话号码。

NANP 与话费欺诈

北美号码方案(NANP)是一个标准的电话号码编制体系，由北美和加勒比海地区的 19 个国家使用。

在美国和加拿大，电信的规定是一样的（明智的做法），允许您拨打到国家号码为 1 的长途，并收取适当的话费。然后，很多人没有意识到，在使用 NANP 的这 19 个国家中，多数都有不同的电信规定(详细的信息可以查看 http://www.nanpa.com)。

很多话费欺诈的办法是哄骗天真的北美人拨打话费极高的加勒比海国家的收费电话的号码，而主叫者确认为他们拨打的是 1-NPA-NXX-XXXX 这样的号

码，只付标准的国内长途费用。由于出现问题的这些国家的规定里允许这种形式的欺骗，主叫最终还是必须支付这些话费。

防止这类事情出现的唯一方法是组织到某些区号的呼叫，例如 809，并且只在必要的时候去除这种限制，然后立即恢复。请寄予额外注意，以保证用户不会滥用您的电话系统！

再来看另外一个：

_1NXXNXXXXXX

这一个稍微复杂一点，它匹配 1，紧接着是 200 和 999 之间的地区号码，最后是 7 位号码。在 NANP 中，可以用这个模式来匹配任意的长途号码。

现在这一个更绕一些：

_011.

不妨多看几遍。注意到末尾的句号了吗？这个模式匹配任何以 011 开始的号码，且至少为要多 1 位。在 NANP 中，这个模式表示的是国际电话号码。.[*]

[*]在后面的几节里，把这些模式添加到拨号方案中，增加去话功能。

3　使用${EXTEN}通道变量

我们知道你在想什么... 你坐在那里问自己，“我使用模式匹配的话，会发生什么？我需要知道实际上拨的是那些数字。”很幸运，Asterisk 有答案。一旦拨了某个 extension， Asterisk 会把通道变量${EXTEN}设置为所拨的数字。可以使用应用 SayDigits( )来检测出来：.

exten => _XXX,1,SayDigits(${EXTEN})

在这个例子中，SayDigits( ) 应用会把所拨的 3 位 extension 读出来。

通常，把 extension 的前面几位去掉对于处理${EXTEN}是很有用的。 者可以利用这样的语法来实现：${EXTEN:*x*}，其中 *x* 是要去掉的位数。例如，假设 EXTEN 的值是 95551212，那么${EXTEN:1 等于 5551212。再来看另外一个例子：

exten => _XXX,1,SayDigits(${EXTEN:1})

在这个例子中，SayDigits( )应用只把所拨的 extension 的最后两位读出来。如果 *x* 是负数，SayDigits( )给出所拨的 extension 的最后 *x* 位。在下面这个例子中，SayDigits( )只读出所拨的 extension 的最后 1 位：

exten => _XXX,1,SayDigits(${EXTEN:-1})

开启去话拨号

模式匹配已经介绍过了，现在可以继续下一个过程，允许用户向外拨打电话。首先要做的一件事情是给[globals] context 加一个变量，用于定义那一个通道可以用来向外拨打电话。：

```
[globals]
JOHN=Zap/1
JANE=SIP/jane
OUTBOUNDTRUNK=Zap/4
```

接下来，在拨号方案中添加一个用于去话的 context。

在此，你肯能会问，“为什么话要使用一个单独的 context？”这样做的目的是能够规定和控制谁可以拨打去话，以及可以拨打什么样的去话。

首先，建立一个用于本地电话的 context。为了与传统电话交换机保持一致，我们在模式之前放上 9，因此用户必须在呼叫外部号码之前拨 9：

```
[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion( )

exten => _9NXXXXXX,102,Congestion( )
```

注意：拨 9 并没有立即给你外线，这和传统的 PBX 系统不同。一旦在 FXS 线路上拨 9，拨号音会停止。如果在拨 9 之后还希望有拨号音，加入下面一行（就在 context 定义之后）：

```
ignorepat => 9
```

这个指令告诉 Asterisk 继续提供拨号音，即便是在主叫方已经拨了指示的模式。

现在回顾一下我们刚刚所做的事情。增加了一个全局变量 OUTBOUNDTRUNK，它会控制使用哪一个通道同于去话；还增加了一个用于本地去话的 context。在 priority 1 中，取出所拨的 extension，用${EXTEN:1}语法去

掉 9，然后试图在变量 OUTBOUNDTRUNK 所指定的通道上拨这个号码。如果呼叫成功，主叫方就与去话通道建立桥接。如果呼叫不成功(要么是通道忙，要么是因为某种原因不能拨这个号码，调用 Congestion( )应用，播放“快忙音”（拥挤声音）让主叫方知道呼叫不成功。

在进一步往下走之前，先确认一下这个拨号方案允许拨打紧急电话号码：

```
[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion( )
exten => _9NXXXXXX,102,Congestion( )

exten => 911,1,Dial(${OUTBOUNDTRUNK}/911)
exten => 9911,1,Dial(${OUTBOUNDTRUNK}/911)
```

我们再次假设，出于这些例子的原因，我们在美国或者加拿大。如果在此之外的其他地区，用当地的紧急电话号码替换掉 911。在拨号方案中永远不要忘记这一点。

下面，给拨号方案加一个用于长途电话的 context ：

```
[outbound-long-distance]
exten =>
_91NXXNXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Congestion( )
exten => _91NXXNXXXXXX,102,Congestion( )
```

现在有了两个新的 context，如何允许内部用户利用它们？我么需要一种办法来使得一个 context 能够使用另一个 context。

Includes

Asterisk 允许在一个 context 中使用另一个 context，通过 include 指令来实现。这用来授予访问权给不同的拨号方案段。我们使用 include 功能来让[internal] context 中的用户 能够拨打去话。首先介绍一下语法。

include 语句的形式如下所示，其中的是我们要包含在当前 context 的远地 context：

include => *context*

在当前 context 包含另外的 context 时，必须注意包含的顺序。Asterisk 首先试图在当前 context 中匹配 extension。如果不成功，会去尝试第一个包含进来的 context ，然后按照包含顺序再去尝试其他的 context。

到目前为止，拨号方案有两个 context 用于去话。但是[internal] context 中的人还不能够使用它们。我们用在[internal] context 包含这两个去话 context 来实现使用，如下所示：

```
[globals]
JOHN=Zap/1
JANE=SIP/jane
OUTBOUNDTRUNK=Zap/4

[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(${JOHN},10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(${JANE},10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )
```

```
[internal]
include => outbound-local
include => outbound-long-distance

exten => 101,1,Dial(${JOHN},,r)
exten => 102,1,Dial(${JANE},,r)

[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion( )
exten => _9NXXXXXX,102,Congestion( )

exten => 911,1,Dial(${OUTBOUNDTRUNK}/911)
exten => 9911,1,Dial(${OUTBOUNDTRUNK}/911)

[outbound-long-distance]
exten =>
_91NXXNXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Congestion( )
exten => _91NXXNXXXXXX,102,Congestion( )
```

这两个 include 语句让[internal] context 内的主叫方可以拨打去话。应该注意到，出于安全的考虑，要确保[inbound] context 永远不要允许拨打去话。(如果一旦给了这样的机会，外面的人可以拨入你的系统，然后再拨打收费电话出去，让你来承担通话费用！)

5.4 结束语

到此为止，我们已经有了一个基本的，而且具有一定功能的拨号。尽管这个拨号方案的功能还不够全面，但是已经覆盖了全部的基本原理。在接下来的几章，我们会在此基础上，增加更多的功能。

如果您对这个拨号方案中的某个部分还是不明白，请返回去，重新阅读一下，然后再进入下一章。你必须理解这些基本原理和它们的使用，否则你会很难理解后面几章的内容。我们希望你能够理解！

第6章　更多的拨号方案概念

技术手段已经不能使生活变得更好的时候，请按“3”。

——Alice Kahn

非常好，你已经学会了基本的拨号方案。而且，你希望了解更多。别担心，这里有更进一步的内容。如果你没有将前一章搞清楚，请你回到前一章，再读一遍。至今为止，我们已经构建了能够覆盖到的全部，我们需要你对这些材料感到熟悉。这样，我们就可以开始讨论更多高级的话题。

6.1　表达式和可变操作

我们了解拨号方案之前，我们必须先向你介绍一些诀窍。这些诀窍可以使你极大的增强试运行你的拨号方案的能力。通过激活以各种不同的标准做出决策的架构，这些架构增加了惊人的智能机制到你的拨号方案。开启你的思维，让我们开始吧。

基础表达式

表达式是变量、运算符和数值的联合，当你把它们组合到一起就会得到一个表达式结果——值。一个表达式能测试数值、变长字串或者履行精确的数学计算。我们说有一个变量称作“COUNT（计数）”。在英语中（译者按：汉语也当如此），两个表达式用变量可以做“COUNT+1”和“COUNT 除于 2”。根据给定的变量的值， 每一个表达式都有一个精确的结果或者数值，取决于赋给变量的值。

在 Asterisk 中，表达式总是以$符合作为开始，以方括号“[]”来扩住表达式，如下所示：

$[expression]

因此，上面的两个例子表示如下：

$[${COUNT} +1]

$[${COUNT}/ 2]

当 Asterisk 在拨号方案中遇到一个表达式时，它可以以完整的表达式代替表

达式结果的值。非常值得注意的是，表达式的值的计算在变量代入之后发生。为了验证，我们查看以下代码：

※记得当你使用一个变量的时候，你可以使用它的名字来调用，但是当你引用一个变量作为一个值的时候，你就必须把这个变量名用花括号“{}”括起来，而且在前面加上美元符号“$”标示出来。

exten =>321,1 , set (COUNT=3)

exten=> 321,2 , set(NEWCOUNT=$[${COUNT}+1])

exten=>321,3, SayNumber(${NEWCOUNT})

在第一步，我们将数值 3 赋予变量 COUNT.

在第二步，只和一个应用——Set()有关，但却有三个动作发生：

在表达式中，Asterisk 用数值 3 替代${COUNT}。 表达式实际上变成:

Exten=>321,2,set(NEWCOUNT=$[3+1])

然后，Asterisk 对　表达式求值，将 1 加 3，　并用 4 代替它：

Exten=>321,2,set(NEWCOUNT=4)

最后，通过 set()函数，数值 4 被赋予给变量 NEWCOUNT。

在第三步只是调用 saynumber()这个应用，说明变量 NEWCOUNT 的值为 4（已经在第二步中赋值）。

在自己的拨号方案中试一下。

运算符

当你创建一个 Asterisk 的拨号方案时，你确确实实使用专有的脚本语言写代码。这就意味着和其他的编程语言类似，Asterisk 的拨号方案识别符号化的运算符，这样，你就可以对变量进行操作。让我们看一下在 Asterisk 系统中可用的运算符的种类：

Boolean operators（布尔型运算符）

这些运算符是求一个语句的真假值。在计算机术语中，基本上语句就是是和否（0 或者非 0，开或者关，等等）。Boolean 运算符是：

expr 1 | expr2

如果 expr1 的值是真值，那么运算符将赋 expr1 的值，否则将赋 expr2 的值

（这个运算符称作“或”）

expr 1 & expr2

如果两个表达式的值都为 true，运算符将赋值为 expr1。否则赋值为 0（这个运算符称作“与”）

expr 1 {=,>,>=,<,<=,|=} expr2

如果自变量都是整数，这些运算符将得到一个整数的比较结果；否则，它们将得到字符串的结果。如果给定的关系是正确地，这个结果是 1，否则就是 0。

数学运算符

要执行一次计算吗？你需要这些：

expr1{+, -} expr2

这些运算符能得到整数自变量的加法或者减法地结果

expr1{,/,%} expr2*

这些运算符能分别得到整数自变量的乘法、除法或是余数的结果

正则表达式运算符

在 Asterisk 中也能用正则表达式运算符※。

expr 1 : expr 2

这个运算符匹配 expr 2 到 expr 1，这里的 expr2 必须是一个正则表达式★。

※ 对于更多的正则表达式，到以下连接获取一份根本上的参考：

- Jeffrey Friedl's Mastering Regular Expressions (O'Reilly; http://www.oreilly.com/catalog/regex2/)
- http://www.regular-expressions.info

★ 如果你不知道一个^号在正则表达式里的作用的话，你最好阅读上述的文档，这将改变你的人生。

如果匹配成功，被匹配的表达式包括了至少一个正则表达式的子表达式，整个表达式对应返回\1；另外，匹配操作符返回字符匹配上的数量。如果匹配失败，返回空值。其他情况返回 0。

Asterisk 的分析程序非常简单，因此在你输入时，在运算符和其他数值之间

至少需要一个空格。因此下面的表达式是不能运算的。

exten =>123,1,set(TEST=$[2+1])

这是将字符串“2＋1”赋给变量 Test，而不是数值 3。 相反，输入值并运算，要象以下输入：

exten => 234,1,Set(TEST=$[2 + 1])

为了把文字内容直接连接到一个变量的开始或是结束，简单的一个表达式，把两个变量放在一起就可以了。如下：

exten=> 234,1,set(NEWTEST=$[blah${TEST}])

6.2 拨号方案函数

拨号方案函数不是一个新的概念。在 Asterisk 1.2 中，这些函数在可用的地方都被使用过。许多履行了和相应的函数相同的操作的应用，将会被最终去掉，以利于函数的使用。函数可以使你增加更多的功能到你的表达式中。你可以想象他们运行起来和操作符类似，但是要更加高级一些。例如，拨号方案函数可以使你计算字符串长度、日期和时间，MD5 校验值等等，所有的这些都在一个拨号方案表达式中实现。

语法

拨号方案函数有以下基本语法：

FUNCTION_NAME(argument)

非常像变量，你可以类似上面引用函数名，但是你如果要引用函数的值，就要用美元符号“$”放在前面，用花括号“{}”括起函数表达式。

${FUNCTION_NAME(argument)}

函数也可以嵌套封装其他的函数，如下：

```
${FUNCTION_NAME(${FUNCTION_NAME(argument)})}
 ^              ^ ^              ^         ^^^^
 1              2 3              4         4321
```

当你已经大概描述出来的时候，你也必须认真的确认各种括号已经配对。在上面的例子中，我们已经标记了的左括弧和右括弧，左括号和右括号的总数是相同的。

拨号方案函数

函数常常用来连接 set()应用，来取得或者赋值一个变量的值。一个简单的例子，让我们看一下 LEN()这个函数。这个个函数能计算出它的自变量的字符串长度。我们计算一下一个字符串的长度，并读出这个长度：

exten=>123,1,set(TEST=example)

exten=>123,2,saynumber(${LEN(${TEST})})

上面这个例子能算出字符串 example 有 7 个字符，将字符串的数量赋值给变量长度，然后将数量送给 saynumber()。

我们看另一个简单的例子。如果我们要设置一个动态通道的超时，我们应该用 TIMEOUT()函数。这个函数可以接受以下三个中的一个做为自变量，分别是 absolute、digit 和 response。他们对应的应用是 AbsoluteTimeout()，DigitTimeout()和 ResponseTimeout()。 用 timeout()函数，设置数字的超时，我们可以 set()函数，如下：

exten=>s,1,set(TIMEOUT(digit)=30)

注意在这个函数中没有${}。就像如果我们给一个自变量赋值一样，我们就赋值给一个函数，是不使用${}封装的。

在 Asterisk 命令行的界面下，键入 show functions 就可以获得可用的函数的完整的列表建立。

6.3 条件分支

现在你已经能掌握一点关于表达式和函数了，可以使用他们了。通过使用表达式和函数，在拨号方案中，你能增加更多的高级逻辑。为了你的拨号方案下决定，你需要使用条件分支。让我们仔细看一下吧。

GotoIf( )应用

条件分支的关键是 GotoIf( )应用。根据表达式的值是 true 或是 false，程序将跳转到相应的目的（Destination）。

GotIf()使用一种特别的语法，经常叫做条件语法。

GotoIf（expression？destination1：destination2）

如果表达式求的值为真，程序被跳转到第一个目的。如果值是错误的，程序被跳转到第二个目的。那么，什么是 true，什么是 false 呢？ 一个空字符串和数 0 作为 false。其他的被认为是 true。

目的可以是下面的每一个：

相同的 extension（Exten）内的优先级，例如 10

相同上下文内的一个 extension 和优先级，如 123，10

一个上下文、extension 和优先级，如 incoming，123，10

在同一个 extension 内的命名过的 priority，如 passed

任何一个目的可以省略，但两个选择不能完全都省略掉。如果省略的目的被跟踪，Asterisk 能简单的跳入当前 extension 的下一个 priority 中。

作为例子，我们使用 GotoIf()如下：

```
exten =>345,1,set(TEST=1)
exten =>345,2,GotoIf($[{$TEST}= 1] ?10:20)
exten =>345,10,Playback(weasels-eaten-phonesys)
exten =>345,20,Playback(office-iguanas)
```

在第一行中，通过改变 TEST 得值，你应该能使 Asterisk server 播放不同的语音问候。

我们看一下另外的条件分支的例子。这次，我们用 Goto()和 GotoIf()，数到 10 然后挂断：

```
exten =>123,1,set(COUNT=10)
exten =>123,2,GotoIf($[${COUNT} > 0]?:10)
exten =>123,3,SayNumber(${COUNT})
exten =>123,4,SET(COUNT=$[${COUNT} – 1] )
exten =>123,5,Goto(2)
exten =>123,10,Hangup( )
```

我们分析这个例子。首先，我们设置了 10 给变量 COUNT。然后，我们查看 COUNT 是否远远大于 0。如果是，我们进行下一个优先级。当我们从 COUNT 得到数 1，然后回到优先级 2。如果 COUNT 小于或者等于 0，进入优先级 10，呼叫挂起。

这个典型的有条件的分支例子在“拒绝女友”逻辑一样，非常有名。如果呼入的主叫方 ID 号码匹配上了“其他女友”的容器中包括的号码。Asterisk 产生一个和其他普通的主叫不同的消息。虽然有点简单或者原始，但是在 Asterisk 拨号方案中学习条件分支，这是一个很好的例子。

这个例子用到一个叫做 CALLERIDNUM 的通道变量，它是自动通过 Asterisk 的呼入的呼叫的 Caller ID 号码设置的。我们假定这个例子的电话号码是 885-555-1212：

```
exten =>123,1,GotoIf($[${CALLERIDNUM} = 8855551212]? 20:10)
exten =>123,10,Dial(Zap/4)
exten =>123,20,Playback(abandon-all-hope)
exten =>123,21,Hangup()
```

在优先级一中，我们调用 GotoIf()程序。如果 Caller ID 号码与 8885551212 相配，它告诉 Asterisk 转到优先级 20，否则转到优先级 10。如果号码匹配，呼叫转到优先级 20，回放一个拒绝消息给不想要的主叫方。否则，这个呼叫尝试在通道 Zap/4 拨号。

使用 GotoIfTime( )实现基于时间的条件分支

在拨号方案中使用条件分支的另一种方法是 GotoIfTime()应用。鉴于 GotoIf()是对一个表达式求值来决定做什么，GotoIfTime()就设计为查看一个系统的时间并用它来决定是否跟随不同的分支。

这个程序最明显的应用是给拨打入你的系统的用户，在上班前、下班后的时间中，送出一个不同的语音问候。

GotoIfTime()程序的语法如下：

```
GotoIfTime(times,days_of_week,days_of_month,months?label)
```

简而言之，当满足上述的时间的条件的时候，GotoIfTime()跳转主叫方到指定的 Label。

我们更详细的看一下每个自变量：

times

在 24 小时格式中，列表中一个或是多个时间范围。例如，上午 9 点到下午 5 点，可以写做 09：00－17：00。一天从 0：00 开始 23：59 结束。

days_of_week

这是一周的一天或是多天的列表。这些天表示为 mon，tue，wed，thu，fri，sat 和 sun。周一到周五表示为 mon－fri。周二到周四表示为 tue，thu。

days_of_month

这是月的天数列表。天可以从 1 到 31。第七天到第 12 天表示为 7－12，第 15 田到 30 天表示为 15，30。

Months

这是一年的一个或多个月的列表。月被写为 jan，feb，mar，apr 等等。

如果你希望所有可能值与其他自变量匹配，简单输入*为自变量。

label 自变量可以如下：

相同的 extension（Exten）内的优先级，例如 10

相同上下文内的一个 extension 和优先级，如 123，10

一个上下文、extension 和优先级，如 incoming，123，10

在同一个 extension 内的命名过的 priority，如 passed

现在我们包含语法，一起看个一对例子。这个例子与：从上午 9 点到下午 5:59,从周一到周五，一个月的任何一天，一年的任何一个月匹配。

Exten =>s,1,GotoIfTime(09:00-17:59, mon-fri, *, *? Open,5,1)

如果主叫在这些小时发起呼叫，呼叫将被发送到 extension 的第一个优先级。如果呼叫在规定的时间之外，它将送到当前 extension 的下一个优先级。这允许你很方便的多次根据时间条件进行分支。见下一个例子（注意，你应该经常将你的最经常出现的特定时间的匹配过程放在最不容易出现的特定时间之前）：

；除了每月 4 日，和每年七月，我们都不关门。

exten=> s, 1, GotoIfTime(*,*,4,jul?open,s,1)

；当不关门的时间的时候，发送呼叫到开放的上下文去

Exten=>s,2,GotoIfTime (09:00-17:59|mon-fri|*|*?open,s,1)

Exten=>s,3,GotoIfTime (09:00-11:59|sat|*|*?open,s,1)

；其他情况，我们关门

Exten =>s,4,Goto(closed,s,1)

6.4　语音邮件（Voicemail）

在任何现代电话系统的最流行的特征之一就是语音邮件。事实上，Asterisk 有一个非常灵活的语音邮件系统。Asterisk 的语音邮件系统的特征包含如下：

- 没有限制的密码保护的语音邮箱，每一个包括的邮箱文件夹都是有组织的语音邮件
- 忙时和无效的状态有不同的问候语
- 默认的和定制的问候
- 能够一个电话对应多个邮箱和多个电话对应一个邮箱
- 语音邮件的 Email 通知，语音邮件作为一个声音附件附在传统邮件上

  ※ 而且你不用为此付费，这已经在正常工作了。
- 语音邮件转发和广播
- 在多种类型电话中的消息等候指示（闪光或者间断的拨号音）
- 基于语音邮箱的职工公司目录

这只是冰山的一角而已。在这一部分，我们将介绍语音邮件安装的原理。

语音邮件配置在配置文件 *voicemail.conf* 中配置和定义。这个文件包含语音邮件系统设置的分类，这样就可以客户化配置语音邮件系统到你想要的状态。在 *voicemail.conf* 中，覆盖所有的可用的可选项将是一个需要大量工作的过程，但是示例配置已经配非常好的注释过，而且很容易模仿着配置。

正像拨号方案上下文可以分别保持你的拨号方案的不同部分一样，语音邮件上下文允许从一个到另外一个邮箱，来定义不同的设置。这就允许你为几个不同的公司或是办公室在同一个服务器定义不同的语音邮件设置。语音邮件的上下文和拨号方案的上下文的定义方法相同，使用方括号括住上下文名称。在我们的例子中，我们使用[default]作为语音邮件的上下文。

创建邮箱

在每一个语音邮件上下文中，我们定义不同的邮箱。定义邮箱的语法如下：

Mailbox=> password, name [,email[, pager_email [, option]]]

我们解释一下邮箱定义的每一部分：

Mailbox

这是邮箱的号码。它通常符合关联设置的扩展号码。

Password

这是邮箱主人用来访问自己的语音邮箱的数字密码。如果使用者改变了她的密码，系统会升级在 *voicemail.conf* 文件中的相应字段。

Name

这是邮箱主人的名字。公司目录使用这个字段的文字使得主要用户拼写用户名。

Email

这是邮箱所有者的 email 地址。Asterisk 能发送语音邮件通知（包括语音邮件消息本身）到特定的邮箱。

Pager_email

这是邮箱所有者的呼机或是蜂窝电话的 email 地址。Asterisk 可以发送一个短的语音邮件通知消息到这个特定的 email 地址。

Options

这是一个选项列表，能设置邮箱所有者的时区等。 这有九个有效的选项：attach , servermail, tz , saycid, review, operator, callback, dialout,和 exitcontext。这些选项应该是按照 *option =value* 格式成对出现，使用管道符号“|”分隔。 tz 选项设置用户的时区到 *voicemail.conf* 的前面定义过的[zonemessage]中，其他八个选项在整个语音邮件范围内用同样的名字设置。

一个典型的邮箱定义如下：

101=>1234,JoePublic,jpublic@somedomain.com,jpublic@pagegateway.net,tz=central |attach=yes

在最后一段继续我们的拨号方案，我们设置语音邮件邮箱给 john 和 jane。我们设置 john 的密码是 1234，jane 的密码是 4444（记住，这些是设置在 *voicemail.conf* 中而不是 *exensions.conf* 中），如下：

```
[default]
101=>1234,john Doe, john@Asteriskdocs.org, jdoe@pagergateway, tld
101=>4444,jane Doe, jane@Asteriskdocs.org, jane@pagergateway.tld
```

增加语音邮件到拨号方案中

现在我们给 Jane 和 John 创建邮箱，我们允许主叫方在他们没有接听电话时

候，允许留下消息。为了实现这些，我们将使用 VoiceMail()应用。

VoiceMail()应用将主叫方呼叫送到特定的邮箱，因此他能留下一个消息。邮箱应该被指定为 mailbox@context 形式，在这的 context 是语音邮件的上下文。邮箱号码也可以被加上前缀 b 或者 u。如果是字母 b，主叫方将听到邮箱所有者忙音的消息。如果字母 u 使用，主叫方将听到邮箱所有者的用户不可用的消息（如果存在的话）。

在我们的示例拨号方案中使用者。在[internal]中先前有一行，他允许我们呼叫 John：

Exten => 101, 1, dial (${JOHN}, , z)

现在，我们改变一下在 John 忙（在接听另外一个呼叫）时，它将我们的呼叫送往语音邮件，我们将听到 John 忙的消息（别忘了，如果被拨打的线路忙，Dial( )应用发送主叫方到优先级 N+101）。

Exten=> 101, 1, dial (${JOHN}, , z)

Exten=>101,102, Voicemail (b101@default)

下一步，我们增加不可用的信息，如果 John 在十秒钟内没有接听电话，不可用的信息能被播放。记住，第二个自变量到 Dial()应用是一个超时。如果在超时之前，呼叫没有应答，这个呼叫被送到下个优先级。我们增加一个 10s 的超时，如果 John 没有按时应答呼叫将优先发送到语音邮件：

Exten=>101, 1, Dial ($ {JOHN}, 10, r)

Exten=>101, 2, Voicemail(u101@default)

Exten=>101, 102, Voicemail(b101@default)

如果我们在 Dial()应用中增加这两个新的优先级和一个超时自变量，John 忙或是不存在时，主叫方送到 John 的语音邮件中（伴随适当的语音提示）。然而，还有一个小问题，John 没有办法查看他的信息。我们可以如下修正。

接收语音邮件

用户能够检索他们的语音邮件消息，改变他们的语音邮件选项，以及通过使用 VoicemailMain()应用记录他们的语音邮件问候语。在它的典型情况中，VoicemailMain()以无任何变量形式调用。我们增加 extension500 到拨号方案的[internal]上下文中，为了内部用户能拨打 500，来访问他们的语音邮件的消息：

Exten=>500 , 1, VoicemailMain()

创建通过姓名拨打（Dial-by-Name）的目录

Asterisk 的语音邮件系统的最后一个特性，是我们能使用姓名拨号。这是由 Directory()应用而创建。这个应用在 voicemail.conf 中用名字定义邮箱，通过用户的姓名拨打目录，告诉主叫用户一个姓名拨打目录的用户。

Directory()采用三个变量：语音邮件上下文（从这个变量来读入姓名），可选的拨号方案上下文（拨入到用户）和可选字符串。默认情况下，Directory()能通过名（last name）搜寻用户，但是通过 f 选项改让他通过姓（first name）搜寻。我们在拨号方案中的[incoming]上下文中增加了两个姓名拨打的目录，这样主叫既能通过姓也可以通过名搜寻：

exten =>8, 1, Directory(default ,incoming, f)

exten =>9,1, Directory(default, incoming)

如果主叫按 8，他们将得到姓的目录。如果他们拨 9，他们将得到名的目录

6.5 宏指令（Macros）

在拨号方案中为了避免重复，宏指令是一个非常有用的结构。他们帮助你改变拨号方案。为了说明这一点，我们再一次看看我们的拨号方案。如果你记得我们对语音邮件做的改变，那么我们用下面的语句结束我们对于 John 的 extension：

exten =>101,1,Dial(${JOHN} ,10,r)

exten =>101,2, Voicemail(u101@default)

exten =>101,102, Voicemail (b101@default)

现在假如在 Asterisk 系统中有一百个用户，这将涉及大量的复制的 extension。那么假设你需要改变一点你的 extension。那就涉及大量的编辑工作，你不能有任何错误

相反，你可以定义一个宏指令，它包含多步的指令列表，然后让所有的电话 extension 指向这个宏指令。你要改变得全部就是这个宏指令，拨号方案中的任何地方都会参考宏指令的改变而改变。

 如果你很熟悉计算机编程，你将会认可在现代编程语言的子程序中宏的普遍存在。如果你不是很熟悉，别担心，我们会带领你创建一个宏的。

定义 Macros

对于我们第一个宏指令，我们用上文设置的 John 的语音邮件的拨号方案逻辑，将它变成一个宏指令。那么，我们将用这个宏指令配置 JOHN 和 Jane 相同的功能。

宏指令定义看起来非常像上下文。你定义了一个宏指令如下：

[macro-voicemail]

宏指令的名字必须以 macro-作为开始。这是他们与常规的上下文的区别。拨号方案中的宏指令的命令同其它任何命令很恰当。只唯一的限制因素是宏指令只能用“s”extension。我们增加语音邮件的逻辑到宏指令，改变 extension 为 s，如下：

```
[macro-voicemail]
exten =>s, 1, Dial ($ {JOHN}, 10, z)
exten =>s, 2, voicemail (u101@default)
exten =>s,102, voicemail (b101@default)
```

这是一个开始，但不是很完美，因为这个宏只是单独对应 John 和他的邮箱号码。为了使宏指令一般化，使它不仅能为 John 而且也能为全部的人工作，我们将利用宏指令的另一个性质：自变量。但首先，我们看看怎样在拨号方案中调用宏指令。

从拨号方案中调用宏指令

为了在我们的拨号方案中使用宏指令，我们使用 Macro()应用。这个应用调用特定的宏指令并传递任何自变量。例如，为了从拨号方案中调用我们的语音邮件宏指令，我们可以如下配置：

exten =>101, 1, Macro (语音邮件)

Macro()程序也定义了几种特别的变量为我们使用。他们包括：

$ {MACRO_CONTEXT}

这个被调用的宏中，初始的上下文

${MACRO_EXTEN}

这个被调用的宏中，初始的 extension

${MACRO_PRIORITY}

这个被调用的宏中，初始的优先级

${ARGn}

传递到宏指令的第 n 个变量。例如第一个自变量是${ARG1},第二个是${ARG2}等等。

正如我们以前解释过的，我们最初定义我们的宏指令的方式不是硬性代码编辑（hard-coded）方式，而是普通方式。我们来用${MACRO_EXTEN}改变我们的宏指令，来达到把 101 作为邮箱号码的目的。如果我们从 extension101 调用这个宏，语音邮件消息就会发到邮箱 101；如果我们从 extension102 调用这个宏，语音邮件消息就会发到邮箱 102，等等：

```
[macro-语音邮件]
exten =>s, 1, Dial(${JOHN}, 10, r)
exten =>s, 2, VoiceMail(u${MACRO_EXTEN}@default)
exten =>s,102, VoiceMail ( b${MACRO_EXTEN}@default)
```

宏指令中使用自变量

很好，现在我们越来越接近我们想要的宏指令的方法，但是我们仍然有一个事情需要在改变——我们需要在通道中传递拨号，它在当前对于${JOHN}仍然是硬编码（别忘了，我们定义了一个变量 JOHN 作为一个通道，当我们需要访问 JOHN 的时候我们就可以呼叫）。我们把通道作为一个自变量传递，这样我们第一个宏指令将被完成：

```
[macro-voicemail]
exten =>s, 1, Dial(${ARG1}, 10, r)
exten =>s,2, voicemail (u${MACRO_EXTEN}@default)
exten =>s, 102, voicemail (b${MACRO_EXTEN}@default)
```

现在我们的宏指令完成了，我们在拨号方案中使用它。这里说明我们怎么调用宏指令并提供语音邮件给 John，Jane 和 Jack:

```
exten =>101,1,Macro(voicemail, ${JOHN})
exten =>102,1, Macro(voicemail, ${JANE})
exten =>103,1, Macro(voicemail, ${JACK})
```

有 50 个或者更多的用户，拨号方案仍然是简洁和有组织的——通过宏，我们将为每一个用户分配一线，这会按照需求完成。我们还可以做一些小的不同的宏，对应动态的用户种类，如 executives, courtesy_phones, call_center_agents, analog_sets, sales_department 等等。

一个更高级的宏指令版本如下：

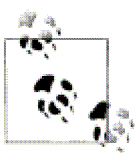
```
[macro-voicemail]
exten => s,1,Dial(${ARG1},20)
exten => s,2,Goto(s-${DIALSTATUS},1)
exten => s-NOANSWER,1,Voicemail(u${MACRO_EXTEN})
exten => s-NOANSWER,2,Goto(incoming,s,1)
exten => s-BUSY,1,Voicemail(b${MACRO_EXTEN})
exten => s-BUSY,2,Goto(incoming,s,1)
exten => _s-.,1,Goto(s-NOANSWER,1)
```

这个宏指令取决于 Dial()应用的返回值情况：当你使用 Dial()应用，它设置 DIALSTAUS 变量，指明呼叫是否成功。在这个例子中，我们处理了 NOANSWER 和 BUSY 例子，并处理了其它全部的结果，作为一个 NOANSWER（被叫方没有应答）。

6.6　使用 Asterisk 数据库

很有趣吧？ 会越来越好的！

Asterisk 提供了一个强大的机制，来存储数值，这叫做 Asterisk 数据库（AstDB）。这个数据库提供了一个简单的方式，用来存储拨号方案的数据。



对于你的关系数据库的经验，如 PostgreSQL 或者 MySQL 而言，Asterisk 数据库不是一个关系数据库。这是一个第一版本的 Berkeley 数据库。有

很多方法可以把 Asterisk 数据存入关系数据库，但这本书不讨论这些。

Asterisk 数据库在分组（family）中存储它的数据，一个分组使用一个键值（key）来确定，键只在一个分组中只能出现一个。例如，如果我们有一个 family 叫做 test， 我们可以存储一个键值叫 count，。每一个存储的键值必须和一个分组有关联。

在数据库中存储数据

我们使用 Set()应用，在 Asterisk 数据库中存储一个新的数值，而不是使用它去设置一个通道变量。我们使用它设置一个 AstDB 变量。例如，下面例子中，将 test 分组中的 count 键的值设置为 1：

※ 以前的 Asterisk 版本有叫做 DBput( )和 DBget( )的应用，这些应用用来从 AstDB 数据库写入和读取数值。如果你使用老版本的 Asterisk，你将需要使用它们。

exten =>456, 1, Set(${DB(test/count)=1})

如果一个命名为 count 的键已经在 test 分组中存在，它的值将被新的值覆盖。通过运行 database put family value 命令，你也能从 Asterisk 命令行中存储数值。对于上面的例子，命令可以写为 database put test count 1

数据库中的数据检索

为了从 Asterisk 数据库中检索一个值，并将它赋值给一个变量，我们再一次使用 Set()应用。我们检索 count 的值（再次地从 test 分组中），并将值赋给一个叫做 COUNT 的变量。如下：

```
exten =>456, 1, Set(DB(test/count)=1)
exten =>456,2, Set(COUNT=${DB(test/count)})
exten =>456, 3, SayNumber(${COUNT})
```

你也可以通过 Asterisk 命令行检查值，使用命令 database get family key。要显示完整的数据库，可以使用 database show 命令。

数据库中数据的删除

有两种方法删除 Asterisk 数据库中的数据。为了删除一个键，使用 DBdel()应用，此应用使用分组和键值作为参数，如下：

Exten =>457, 1, DBdel(test/count)

也可以使用 DBdeltree()应用删除键值分组。这个程序只有一个自变量：要删除键值分组的名字。 为了完全删除分组，如下：

Exten =>457, 1, DBdeltree(test)

为了删除键值和键值分组也可以使用命令行，分别使用 database del key 和 database deltree family 命令。

拨号方案中使用 AstDB

在拨号方案中使用数据库有无数方法。我们举两个简单例子介绍一下 AstDB。第一个是个简单的计算例子介绍了数据库的稳定性（从系统重启中恢复）。第二个例子，我们将用 LookupBlacklist()应用计算一个号码是否在黑名单中，是否应该被封锁。

作为计算的一个例子，我们首先从数据库检索一个号码并将它赋值给变量 COUNT（count 键的值）。如果不存在，DBget()将把脚本送到优先级为 n＋101，而优先级 n+101 被我们设置值为 1。下一个优先级将送回优先级 1。以上这些将发生在我们第一次拨号的时候：

exten =>678, 1, Set(COUNT=${DB(TEST/COUNT)})

exten =>678, 102, Set(DB(test/count)=1)

exten =>678, 103, Goto(1)

下一步，我们说说 COUNT 的最近的值，以及它的递增：

exten =>678, 1, Set(COUNT=${DB(TEST/COUNT)})

exten =>678, 2, SayNumber(${COUNT})

exten =>678, 3, Set(COUNT=$[${COUNT}+1])

exten =>678,102, Set(DB(test/count)=1)

exten =>678, 103, Goto(1)

现在我们已经增加了 COUNT，我们输入新的数值。记住已经存在地 key 将被覆盖：

exten =>678, 1, Set(COUNT=${DB(test/count)})

exten =>678, 2, SayNumber(${COUNT})

exten =>678, 3, set(COUNT=$[${COUNT}+1])

exten =>678,4, Set(DB(test/count)=${COUNT})

exten =>678, 102, Set(DB(test/count)=1)

exten =>678, 103, Goto(1)

最后，我们将对第一优先级循环。这种情况下，程序将继续计算：

```
exten =>678, 1, Set(COUNT=${DB(test/count)})
exten =>678, 2, SayNumber(${COUNT})
exten =>678, 3, SetVar(COUNT=$[${COUNT}+1])
exten =>678, 4, Set(DB(test/count)=${COUNT})
exten =>678, 5, Goto(1)
exten =>678, 102, Set(DB(test.count)=1)
exten =>678, 103, Goto(1)
```

继续并试试这个例子。等待他计算，然后挂断。当你再次拨号，它应该继续计算。存储在数据库中的值将是固定的，即使重启 astersik。

在下一个例子中，我们将使用 LookupBlacklist()程序创建拨号方案的逻辑，程序并检查主叫 ID 号码是否在黑名单中存在。如果程序 LookupBlacklist()在黑名单中找到这个号码，它发送呼叫到 N+101.否则，呼叫继续进入下一步：

```
exten =>124, 1, LookupBlacklist()
exten =>124, 2, Dial(${JOHN})
exten =>124, 102, Playback(privacy-you-are-blacklisted)
exten =>124, 103, Playback(vm-goodbye)
exten =>124, 104, Hangup()
```

增加一个号码到黑名单中的方法是，在 Asterisk 命令行中使用命令 database put blacklist number 1 。

6.7 Asterisk 特征

现在我们学习了一些基础以上的东西，我们看一下已经加入到 Asterisk 的流行的函数

Zapateller()

Zapateller()是一个简单的 Asterisk 应用，它在一个呼叫的开始播放一个特别的声音作为信息，这导致自动拨号器（常常用在电话推销中）认为线路已经断开了。不仅电话将挂断，而且拨入的系统将标记你的号码为无法接通的状态，这能帮助你避免各种各样类型的呼叫（尤其是电话推销的）。简单调用 Zapateller()应

用，使在拨号方案中起作用。

我们也能使用 nocallerid，这样在主叫方没有 caller id 的时候，只播放拨号音。例如，你可以在[incoming]中使用 zapateller()。

```
[incoming]
exten =>s, 1, Zapateller(nocallerid)
exten =>s, 2, Playback(enter-ext-of-person)
```

呼叫停泊（call parking）

另一个方便的特性叫做“呼叫停泊（call parking）”。 呼叫停泊允许你保持一个呼叫一段时间，因此你就能够在另外一个 extension 下保持通话状态。通话保持的参数是在 feature.conf 配置文件中的[general]部分控制。[general]一段包含了四个与呼叫停泊有关的设置：

Parkext

这是等待的 extension 的标签。一个呼叫传递到这个 extension 的时候，系统将告诉你呼叫停泊的位置。通过默认设置，等待的 extension 号是 700

Parkos

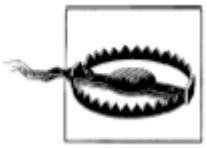
这个选项定义了呼叫停泊的空位数目。例如，设置它从 701 到 720，将创建 20 个等待位置，编号 701 到 720。

Context

这是等待上下文（Context）的名字。你必须包含这个上下文才能等待呼叫。

Parkingtime

如果设置，这个选项控制一个呼叫能等待多长时间。如果呼叫没有在设定时间内连接，extension 将把呼叫停泊的这个呼叫回拨。



当你编辑过 features.conf 以后，你必须重新启动 Asterisk，因为这个文件只有启动的时候才被读取，允许 reload 命令将不会读这个文件。

也要注意，你应该确认：你正在 Dial()应用中使用 t 或者 T 的选项。因为使用者需要传递呼叫到呼叫停泊的标签到 extension。

所以，我们创建一个简单的拨号方案，来演示呼叫停泊，如下：

```
[incoming]
Include =>parkedcalls

exten =>103, 1, Dial(SIP/bob, ,Tt)
exten =>104, 1, Dial(SIP/Charlie, , Tt)
```

为了说明呼叫停泊是如何工作，实例如下：Alice 访问系统，然后拨号 103，来连接 Bob。过了一会，Bob 传递这个呼叫到 700，并告诉他从 Alice 来的呼叫已经在 701 处等待。Bob 在 104 呼叫 Charlie，并告诉他 Alice 在 701。Charlie 拨打 extension701，开始和 Alice 通话。这是一个简单的方法在两个用户之间呼叫停泊的应用。

和 MeetMe( )进行会议

最后，但不是最少，我们使用 MeetMe()应用来设置一个语音的会议桥。这个应用允许多个主叫方互相谈话，就像他们在相同的物理地点一样。下面是一些主要的特性：

※ 在以前的传统 PBX 中，这个功能很昂贵。要么你拨入服务商掏一大笔钱，要么你就不得不安装一个昂贵的会议桥到你所有的 PBX 上去。

- 创建密码保护的会议
- 会议管理（将电话会场静音、锁定会议、断开某个参与者）
- 允许一个人讲话，其他的人安静的可选项（常常用在公司的会议、广播等等情况）
- 创建动态或是静态会议

我们进入设置一个基本会议室。对于 MeetMe 会议系统的配置项是在 meetme.conf 中建立。在配置文件内，你来定义会议室和进入会议室的密码。例如，我们设置一个 extension 为 600 的会议室。首先，我们将在 meetme.conf 中设置会议室，称作 600。而这次我们没有设置密码：

```
[rooms]
Conf => 600
```

现在配置文件完成了，我们需要重启 Asterisk 让他能重新读取 meetme.conf 文件。下一步，我们使用 meetme()应用，把支持会议室的特性增加到拨号方案中。

MeetMe()有三个参数：会议室的名字（定义在 meetme.conf）、一组选项设置、以及用户加入会议的密码。 我们设置一个简单的会议室，房间是 600，“i”可选项（通知用户进入和退出会议），密码是 54321：

```
exten => 600,1,MeetMe(600,i,54321)
```

这就是全部。当主叫方进入 extension600，密码提示。如果正确输入 54321，他们将加入会议。看到附件 B 关于 MeetMe()应用支持的可选项列表的清单。

另一个有用的程序是 MeetMeCount()。如名所示，这个程序是用来计数一个会议室用户数量的。它有两个参数：参与者的数量，可变得名字。如果可变得名字没有在第二个参数中设置，计数结果将宣读给主叫方：

```
exten =>601, 1, Playback(conf-thereare)
exten =>601, 2, MeetMeCount(600)
exten =>601, 3, Playback(conf-peopleinconf)
```

如果你在 MeetMe()中设置了一个变量作为第二个参数，计数结果将被赋值道这个变量，计数结果的回放将被跳过。你可以使用这个特性，来限制参与者的数量，如下：

```
；limit the conference room to 10 participants
exten =>600, 1, MeetMeCount(600, CONFCOUNT)
exten =>600, 2, GotoIf($[${CONFCOUNT} <=10]?3:100)
exten =>600, 3, MeetMe(600, i, 54321)
exten =>600, 100, Playback(conf-full)
```

Asterisk 很有趣吧？

6.8 结束语

在这一章中，我们已经了解了 Asterisk 的很多应用，很希望我们给你的这些使你能探索创建自己的拨号方案。在早先的章节中，我们邀请你回来重读其他的章节。

下面的一章中不再讲 Asterisk，为了讲述关于电话系统使用的一些技术。我们讲谈到很多关于 Asterisk，但是我们要讨论的更多的是普通的通信系统。

第7章　理解电话技术

拥有一台电话让你工作便利，拥有了两台电话便有些奢侈了，当你不得不使用三台电话的时候，你会觉得没有电话的日子象在天堂一样。

—Doug Larson

在接下来的两章里，我们暂时先不谈 Asterisk，因为我们需要花费一些时间讨论讨论与 Asterisk 系统接口的工程技术。在这一章里，我们将要讨论传统通信网络技术，特别是那些与 Asterisk 关联的电话网络技术。尽管电信技术有长篇累棳的大部头巨著，本章中选用的材料来自于我们的经验，有助于更好的理解我们最需要的那部分内容。尽管这些知识对于配置 Asterisk 系统来说并不严格需要，但这些知识对于将 Asterisk 系统与当今世界的传统电信系统联结起来大有帮助。

7.1　模拟电话

公众交换电话网络（PSTN）设立的目的是为了在任意两个端点之间建立并维持语音连接。

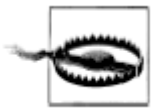
尽管人类能够感知 20-20000Hz 频率范围的声音震动，我们日常说话的的声音频率范围大部分在 250-3000Hz 之间。既然电话网络的目的是为了传输人们说话交流的声音，带宽的设计通常被定义为 300-3500Hz 这个范围。这样的带宽限制意味着一些声音品质将被损失（当人们不得不倾听音乐保持的时候能够证实这种声音品质的损失），尤其是高频声音。

模拟电话的组成

一个模拟电话由 5 部分组成：振铃器，拨号盘，混音电路，叉簧和手柄（叉簧和手柄也被认为是混音电路的一部分）。交流铃，拨号盘，混音电路三者能够被完全独立的操作。

1　振铃器

当中心局需要信号通知有电话呼入时，它将发送大约 90 伏特的交流信号到电话机的电路。这将产生一个交流振铃信号，电话机因此产生铃音。（对于电子式电话，这个铃音可能是一声清脆悦耳的鸟鸣音，而不是通常的电话振铃音，电话振铃音可以是铃流激发的任何表现形式——比如说，在嘈杂的工厂环境常常用闪烁灯来替代电话振铃音）



振铃电压可能导致危险。要十分小心的提防直接接触正在使用的电话线。

不少的人不能区分引起振铃音的交流电压和给电话机供电的直流电压。记住了，直流电不能产生振铃音。

在北美，REN （环状等效值）是帮助使用者了解一条电话线路最多能同时连接几个设备的参考值。（REN 值必须在每个设备上标注），电话线路的最大振铃器负荷为 5。某些电子式电话的 REN 值甚至少于 0.3。

2　拨号盘

当你发起一个电话呼叫的时候，你需要让电话网络知道你期望呼叫的对象地址。拨号盘就是用来提供这个功能的一个部件。在 PSTN 早期，拨号盘是一个旋转拨号设备，它使用脉冲信号表示数字。旋转拨号盘拨号十分慢，因此电话设备公司引入了按键式拨号盘。按键式拨号盘——也就是十分著名的双音多频拨号盘由 12 个按钮组成。每个按钮具有两个频率的组合。（如图）

表 7－1　DTMF 代表的数字

	1209 Hz	1336 Hz	1477 Hz	1633 Hz[a]
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

每次拨号，两个频率的信号便通过线路传送出去。

我们假定你经常使用电话，因此，我们不再在 DTMF 上花费更多的时间。

3 混音电路（或网络）

混音电路处理电话听筒线路和 PSTN 线路的信号发送和接受的 2/4 线转换。混音电路的另外一个功能是调节侧音，侧音就是通过电路反射回说话者耳朵的声音—它的目的是为了提供比较自然的声音大小，侧音太大，声音听起来过于喧吵，侧音太小，又会觉得太安静。

4 叉簧

叉簧的状态决定电话回路的接通和断开的状态。摘机的时候，叉簧接通电话和局端，同时发送一个拨号音请求。挂机的时候，叉簧断开电话回路标志当前通话结束。叉簧有时候也用信令处理，有些电话机有一个标志着”Link”的按钮，它能够产生闪断。你可以通过手动按住叉簧持续 200 到 1200ms 来产生闪断。但是如果你按下叉簧的时间太长，电路将假定你已经挂机。Link 按纽的目的就是为你产生闪断信号，在使用呼叫等待或者三方通话的时候，可以通过闪断来发送信令。

5 手柄

手柄包括受话器和发话器。它实现人类语音形式的能量和电话网络电子形式的能量的转换。

Tip 和 Ring

模拟电话线路使用两根电线。在北美，这两根电线称为 Tip 和 Ring。这两个词来源于通过人工接线员接通电话的年代。接线员使用的插头有两个触点，一个触点在插头末梢，另一个在触点在中间呈环状（图 7－1）。

Tip 线是线路的正极，用于构成回路。在北美 Tip 线是绿色的。Ring 线是负极，在北美是红色的。电话在挂机状态下，Ring 相对 Tip 的电压是直流－48V。

在摘机状态，Ring 降为直流－7V。

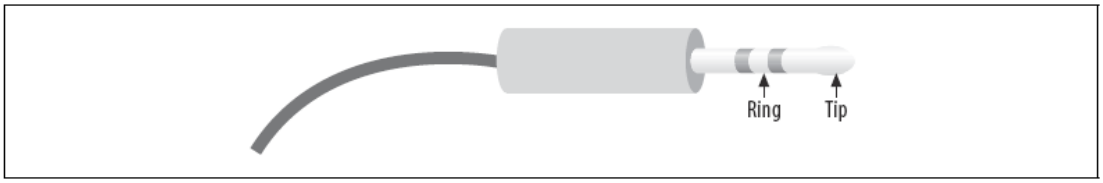


图7-1 TIP和RING

7.2　数字电话

模拟电话技术基本已经销声匿迹了。

PSTN 网络的最后一公里仍然保留了一百多年以前电话初创时的最后一点遗迹。

模拟信号传输的主要挑战之一是信号可能受到各种情况的干扰，引起音量降低，静电噪声以及各种其他噪声效果。不要试图把模拟波形原封不动地长距离传到几千英里之外。为什么不能简单测量原始声音的特征然后把特征信息传递到远端呢？虽然原始波形不能传递很远，但是特征信息可以传递很远，而根据特征信息可以重构原始波形。所有数字化声音的基本原理（也包括数字电话）就是：对信源波形做特征采样，保存采样信息，再发送到远端。远端使用接受到的采样信息生成全新的声音信号，声音信号的特征与信源的特征完全相同。声音再生可以做得很好，人耳无法分辨。数字音频的最主要优点是采样数据在传输到目的的过程中可以通过数学计算做错误检查,确保与信源完全一样的副本到达远端。长距离不再影话音质量,而干扰能被检测出来并且予以消除。

脉冲编码调制

音频的数字化编码有很多种，但最常用的方法（也是电话系统使用的）是脉冲编码调制（PCM）。为了说明它如何工作，我们看几个例子。

1　模拟波形的数字编码

PCM 的基本原理是以特定的间隔对模拟波形的幅度采样，这样波形在以后

可以重新生成。采样的精细程度取决于每一次采样的分辨率和采样的频度。采用更高的分辨率和更高的采样速率会更精确，但需要更多的带宽来传输这些信息。

为了更好地理解 PCM 如何工作，考虑图 7-2.显示的波形。

为了对波形做数字编码，必须以周期性采样为基础，而且要在每个采样时刻测量波形的幅度值。这种将波形切分为一系列时刻并在每一个时刻测量其强度的方法称为量化或采样。

为了远端能够重新生成十分相似的波形，需要有足够的采样频率，并且每次都捕获足够的信息。更精确的采样需要更多的比特。为了解释这个概念，我们从非常低的分辨率开始，用 4 个比特表示幅度。这样量化的过程和分辨率对话音质量的影响都更加直观。

图 7-3 显示了以 4 比特分辨率采样正弦波捕捉到的信息。

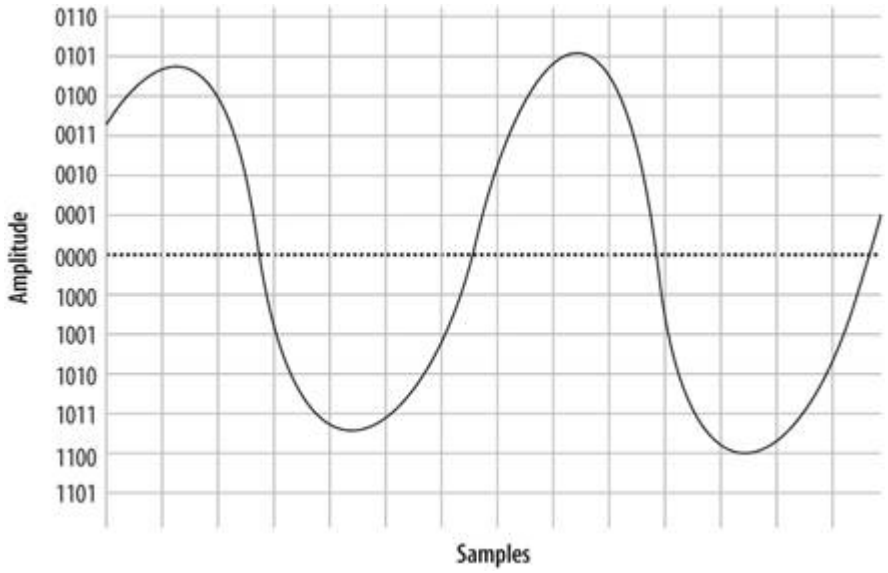


图 7-2 简单的正弦波

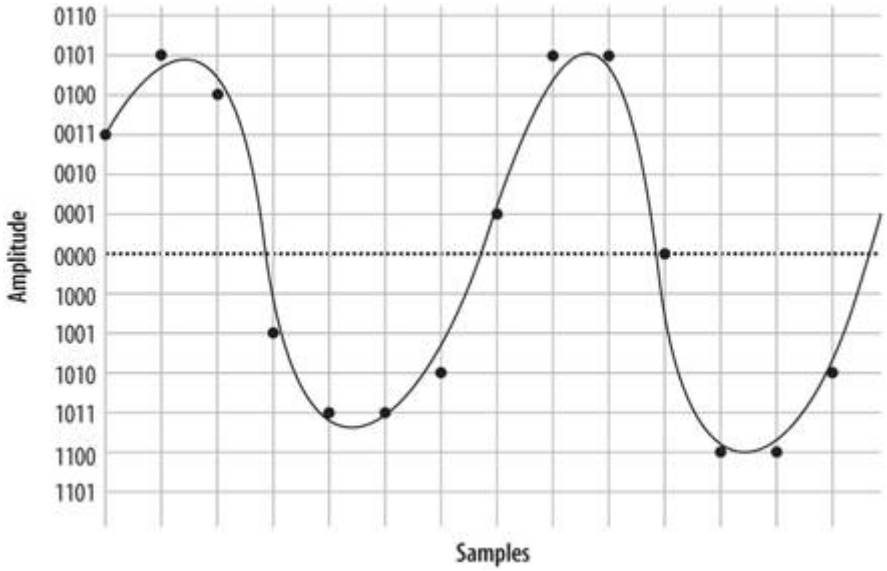


图 7-3 以 4 比特采样正弦波

我们每隔一段时间测量波形的幅度并记录相应的信号强度，换句话说，就是对它采样。你可能注意到 4 比特的分辨率限制了精度。第一个采样取 0011，下一个采样量化为 0101。然后是 0100，在后面是 1001，1011 等等。总的来说，我们得到了 14 个采样（现实中，每秒应该有几千次的采样）。如果把上述的值排成一串，我们可以向另一侧发送如下的内容：　　0011 0101 0100 1001 1011 1011 1010 0001 0101 0101 0000 1100 1100 1010

在线路上，这些代码可能看起来像图 7-4。

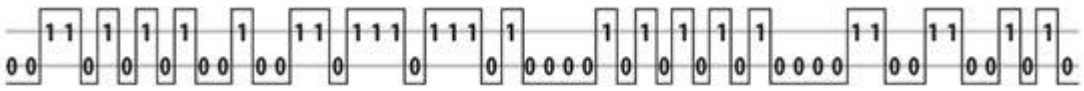


图 7-4　PCM 编码波形

当远端的数字－模拟（D/A）转换器收到这些信息，就可以用这些信息绘出这些采样，如图 7-5 所示。

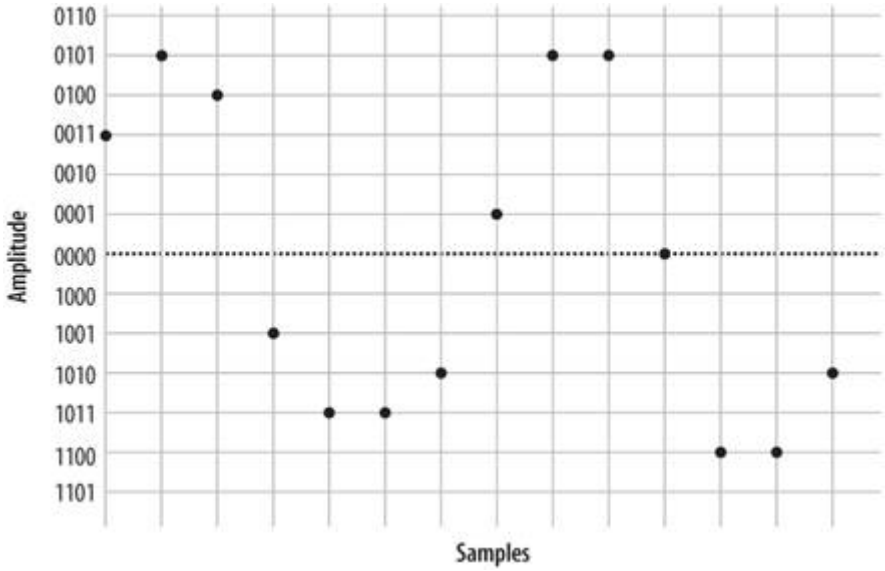


图7-5 绘出的PCM信息

根据这些信息，可以重构波形（见图 7-6）

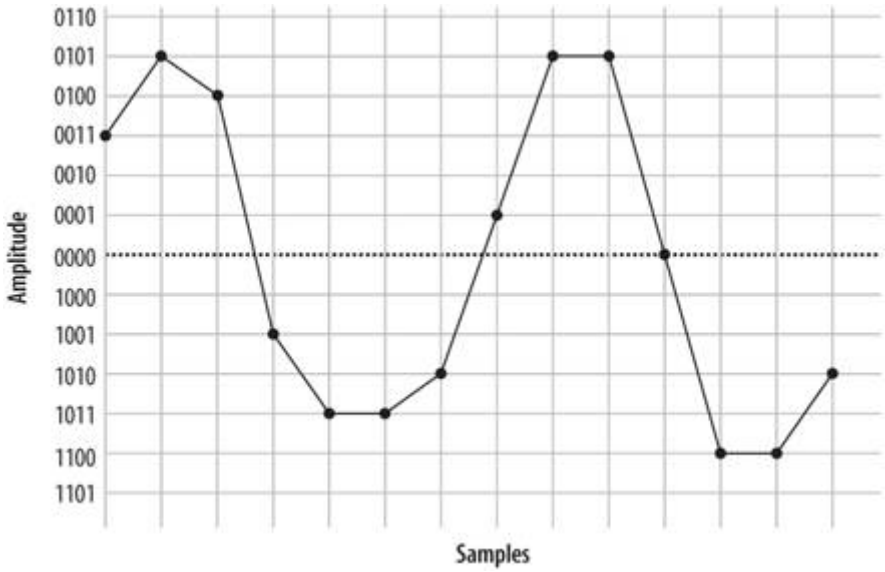


图7-6 描绘的信息

如果你比较图 7-7 和图 7-8，你会发现重构的波形不是很精确。故意这样做的原因是为了证明一个要点：采样的分辨率和采样频率影响波形的数字化编码的质量。如果采样频率过低并且分辨率过低，声音质量将是无法接受的。

2 提高采样分辨率和采样频率

让我们再次观察原始的波形，这次使用 5 个比特定义量化间隔（图 7-7）。

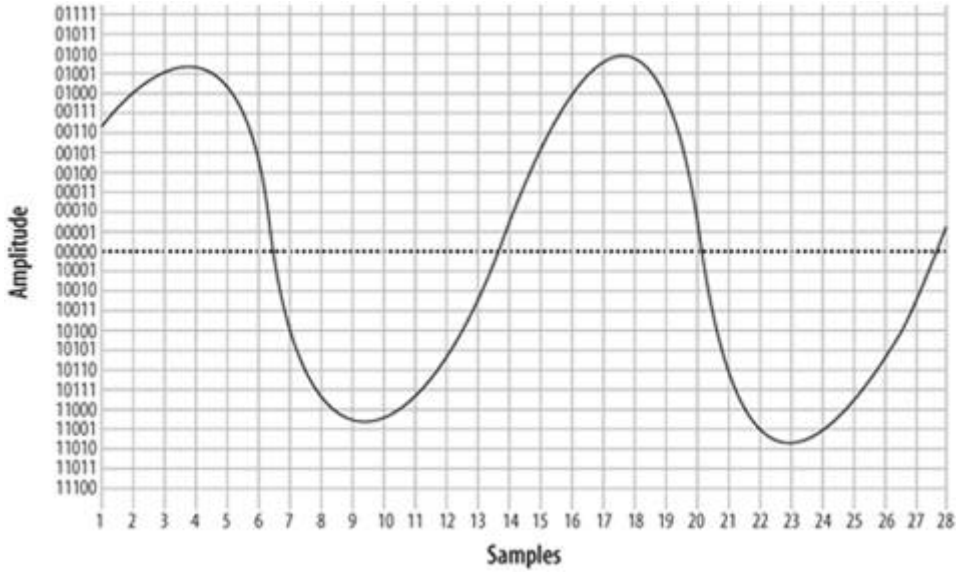


图7-7 同样的波形在更高分辨率的图纸上


现实中，并不存在 5 比特 PCM。电话网的 PCM 用 8 比特编码。[*]

[*]其他的数字音频使用 16 比特或者更多。

我们同样对采样频率加倍。图 7-8 显示了绘出的点。

00111 01000 01001 01001 01000 00101 10110 11000 11001 11001 11000 10111 10100 10001 00010 00111 01001 01010 01001 00111 00000 11000 11010 11010 11001 11000 10110 10001

我们现在有了数量加倍的采样，并且分辨率加倍。如下：

00111 01000 01001 01001 01000 00101 10110 11000 11001 11001 11000 10111 10100 10001 00010 00111 01001 01010 01001 00111 00000 11000 11010 11010 11001 11000 10110 10001

当另一端收到上述信息，可以图 7-9 显示的内容。

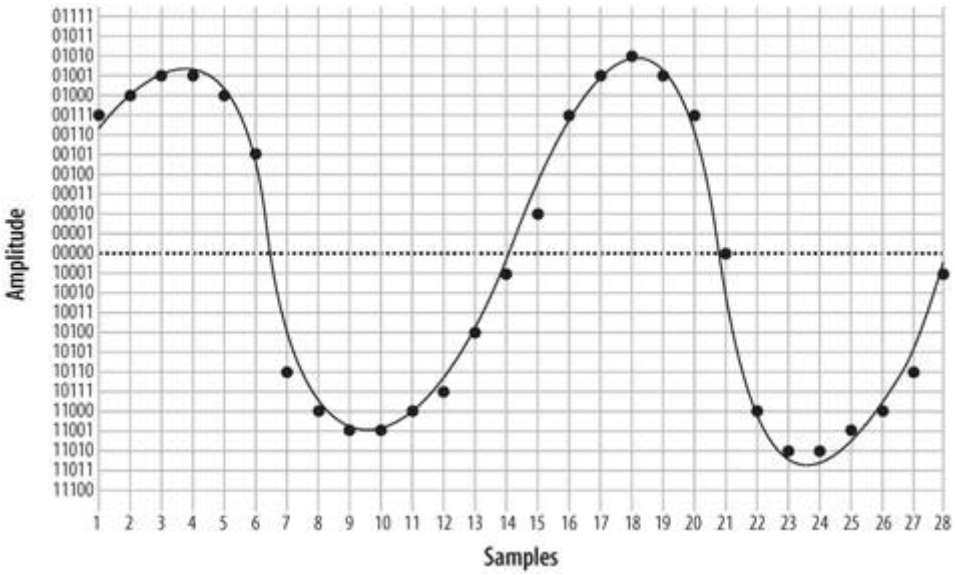


图 7-8 同样的波形使用两倍的分辨率

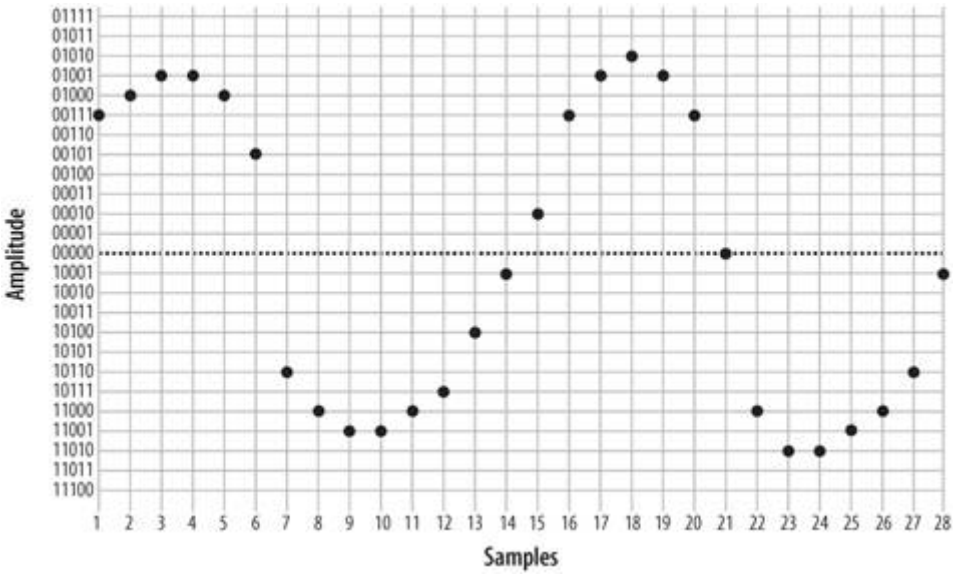
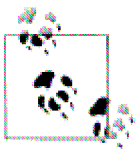


图 7-9 用 5 比特绘出的 PCM 信息

根据这些信息，可以产生图 7-10 显示的波形。

你会发现，合成的波形与原始的波形更加接近。但是，你还可以看到仍然有提高的余地。



注意以 4 比特分辨率编码需要 40 比特，而以 5 比特分辨率编

码编码需要 156 比特（同时采样速率加倍）。关键是要折中：如果你希望更好的声音质量，就需要更多的比特，也就需要发送根多的比特和消耗更多的带宽。

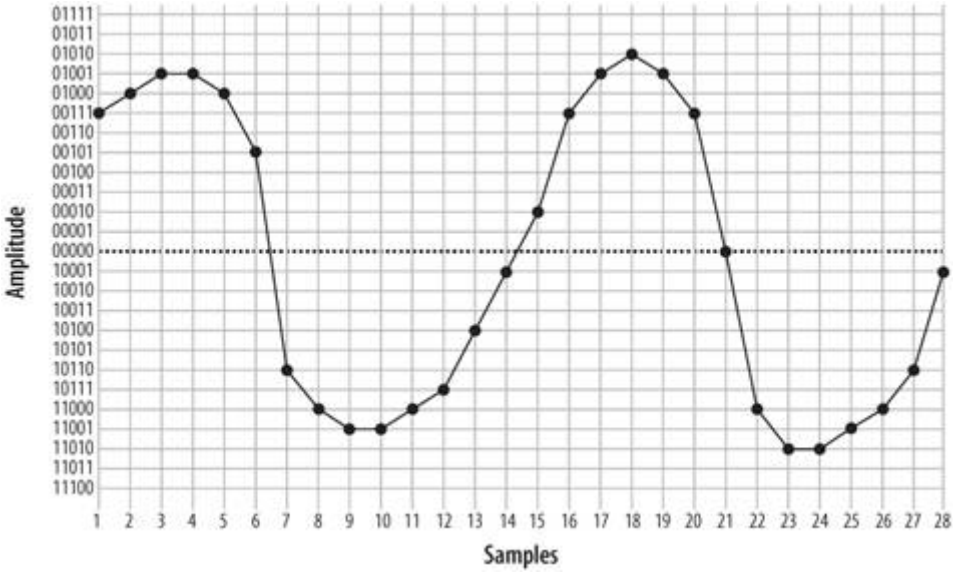


图 7-10 根据 5 比特 PCM 描绘的信息

3 奈奎斯特定理

那么如何采样才够用呢？1920 年一个名叫哈里－奈奎斯特电子工程师（也是 AT&T/贝尔的雇员）考虑了同样的问题。奈奎斯特定理表述为：“在对信号采样时，采样频率必须高于输入信号带宽的两倍以上才能根据采样信息很好的重构原始信号。”

奈奎斯特发表了两篇文章，"Certain Factors Affecting Telegraph Speed" (1924) and "Certain Topics in Telegraph Transmission Theory" (1928)。文中提出的假设被称为奈奎斯特定理。克劳德－香农 1949 年证明了这一假设，因此也被成为奈奎斯特－香农采样定理。

这个定理基本上上意味着为了精确地对模拟信号编码，采样频率必须两倍于需要再生信号的带宽。由于电话网不传输频率低于 300 赫兹和高于 3400 赫兹的信号，每秒 8000 的采样频率在用于再生模拟电话网带宽内的任何频率信号都是足够的。记住每秒 8000 的采样，我们在以后还将谈到它的更多内容。

4 对数压扩

以上我们回顾了量化的基础，还有我们讨论了一个有关的事实，就是更多的量化间隔（也就是更高的采样频率）会有更好的音质但需要更多的带宽。最后，我们还讨论了需要精确测量传送信号所需要的最小采样频率（在电话网中，为8000HZ）。这些是在线路上传递数据的开始，我们接下来将讨论压扩。

压扩是一种提高采样动态范围而不丢失重要精度的方法。它工作的原理是量化比较高的振幅时用更大的粒度，量化比较低振幅用比较小的粒度。换句话说，如果你对着电话喊叫，对你声音的采样不像正常讲话那样清晰。喊叫对你的血压不利，所以最好避免。

通常使用两种压扩方法：在北美用μ律，世界的其他地方用 A 律。这两种方法的工作原理相同，但是相互不兼容。

μ率经常被成为 u 率，原因是，让我们面对它，有几个人的键盘上有 μ 按键？μ 实际上是希腊字母 Mu，因此，你也经常看到写 Mu－law 来表示 μ 率的。在口头上，说 Mew 率更合适。但是别人会觉得你很奇怪。你即使很慷慨地告诉他们应该是 μ 率，他们也不会很感激你。

压扩把波形分成“段”（ 弦，cords），每段包含几个步长。量化包括在段的范围内将测量的振幅与适当步长匹配。步长和段的数字（也包括正负符号）就成为了信号。下图给出了压扩如何工作的直观概念。下图并非基于什么标准，而是为了说明的目的虚构的（电话网的压扩使用 8 比特分辨率，不是 5 比特）。

图 7-11 说明了 5 比特的压扩。你可以发现，对接近零点的振幅的采样比高振幅的采样更精确（无论正负）。但是，由于人耳，传送器，还有接收器都有对大音量失真的倾向，因此没什么问题。

图 7-12 显示了一个量化采样的例子。它产生如下的比率特流：

00000 10011 10100 10101 01101 00001 00011 11010 00010 00001 01000 10011
10100 10100 00101 00100 00101 10101 10011 10001 00011 00001 00000 10100
10010 10101 01101 10100 00101 11010 00100 00000 01000

5 混叠现象（Aliasing）

如果你曾经注意过早期西部片中向后转动的马车车轮，你就看到了混叠（Aliasing）的效果。电影的帧速率跟不上轮辐的转动频率，感受到的就是错误的转动方向。

在数字音频系统中（正如现代 PSTN），如果送到模拟－数字转换器的频率超过采样频率的二分之一就会发生混叠现象（Aliasing）。在 PSTN 中，这样的音频在 4000 赫兹以上（8000HZ 的采样频率的一半）。这个问题可以在送到 A/D 转换器之前，通过低通滤波器很容的纠正。

顾名思义，低通滤波器允许低于它标称的频率通过。其他类型的滤波器是高通滤波器（去掉高频）和带通滤波器（滤掉高频和低频）。

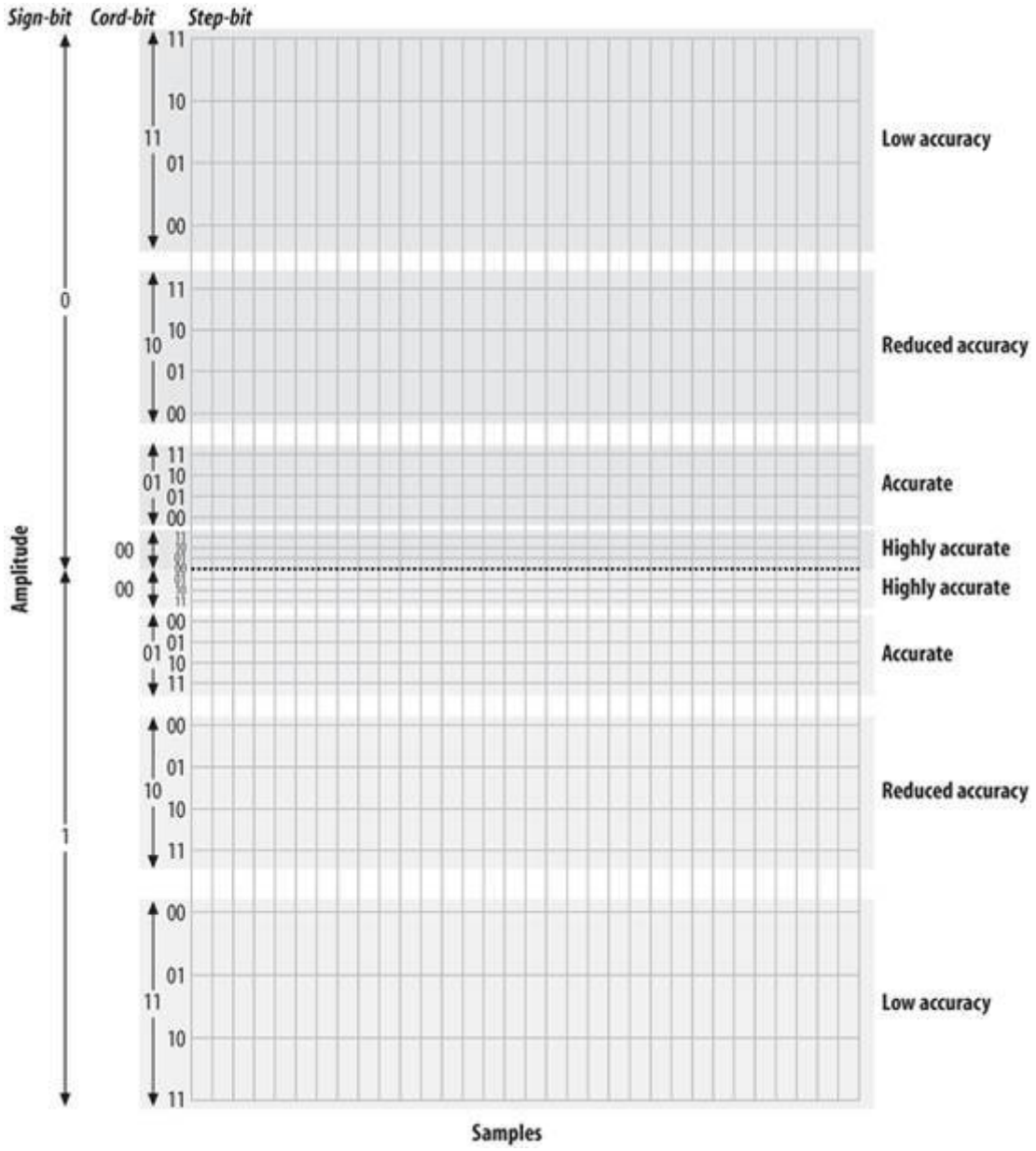


图7-11 5比特压扩

7.3　数字电路交换电话网

一百多年来，电话网络完全是电路交换的。这句话的意思是说，对于每一个电话呼叫，都会在两个端点之间建立一个专用的连接，兵分配固定的带宽给这个连接。创造这样的网络是昂贵的，尤其在长距离时，使用这样的网路也很昂贵。虽然我们预言电路交换网络的末日来了，但很多人仍然每天都使用它，并且它还运转得相当好。

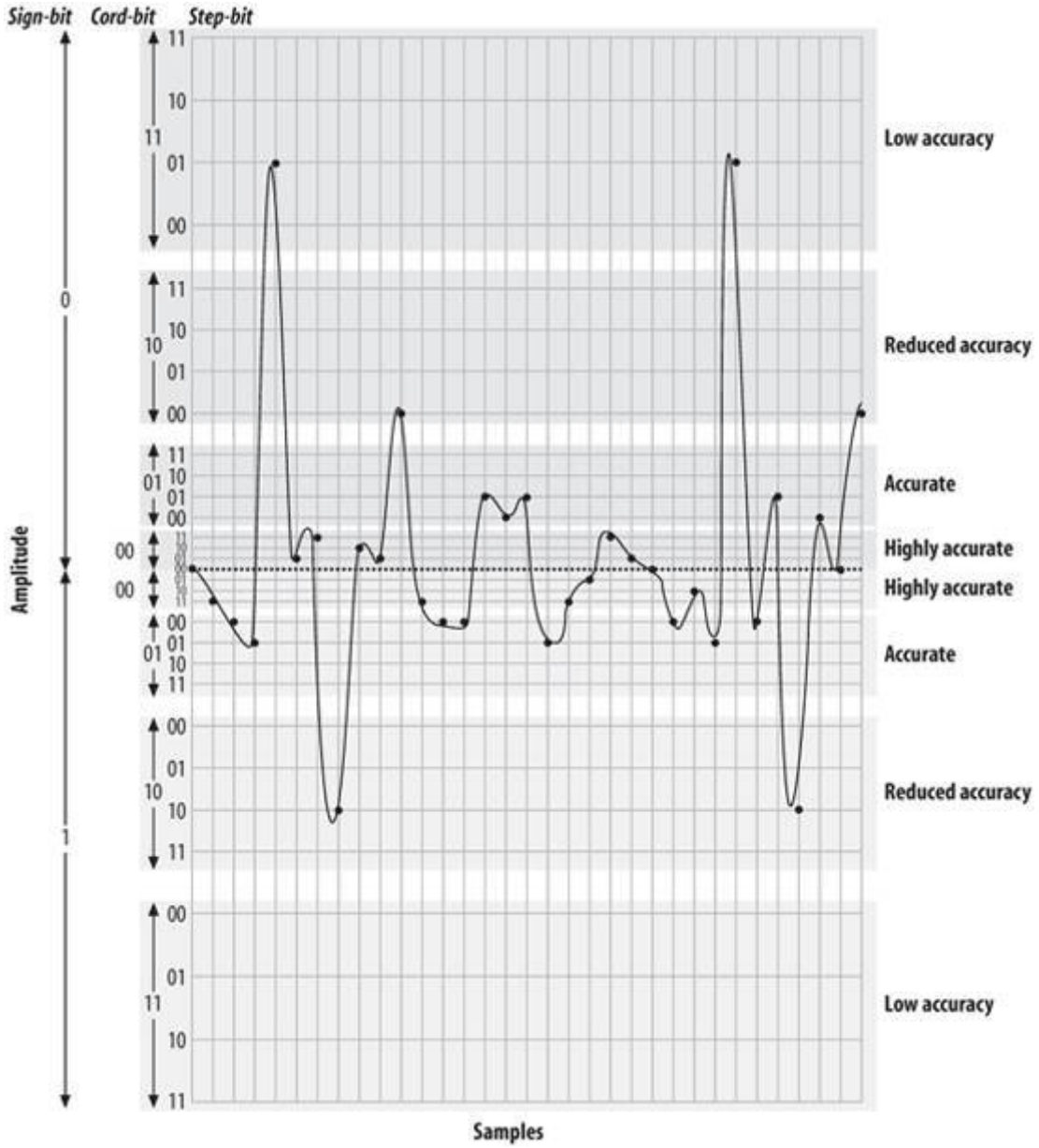


图7-12 经过量化和5比特压扩

电路的类型

PSTN 中，有许多不同容量的电路来满足网络的各种需求。局端和用户之间，通常有一条或多条模拟线路，或者通过数字电路传输的几十个通道。在 PSTN 的交换局之间（或者大客户），通常使用光纤。

1　DS-0，全部的基础

电话呼叫数字化的标准方法是以 8 比特每秒采样 8000 次，所以我们知道 PCM 编码的电话电路需要的带宽是 64,000 bps。这种 64-kbps 的通道被称为。DS-0 是数字化电信电路的基本模块。

普遍使用的模拟电路尽可能的采样为 DS-0。有时这是在终结模拟电路的局端完成的，有时在到局端之前就完成了。

2　T 承载电路

T-1 是公认的数字电话术语之一。一个 T-1 由复用在一起，形成一个 1.544-Mbps 的比特流[*]。这个比特流的正式定义是 DS-1。在 T-1 声音使用 μ 律压扩算法编码。

[*]24 个 DS-0 使用 1.536 Mbps，其余的 0.008 Mbps 用作分割位。

欧洲版的 T-1 由 European Conference of Postal and Telecommunications Administration（CEPT）开发，最初被称为 CEPT-1，现在称为 E-1。

E-1 由 32 个 DS-0 组成，但 PCM 编码方法有所不同。E-1 使用 A 律压扩。这意味着基于 E-1 的网络与一个基于 T-1 的网络连接时需要进行编码转换。注意，虽然一个 E-1 由 32 个通道，仍然被认为是 DS-1。

其他不同的 T 承载（T-2,，T-3，和 T-4）都是 T-1 的倍数，都基于基本的 DS-0。表 7-2 说明了不同的 T 承载电路之间的关系。

表 7-2 T 承载电路

Carrier	等效的数据比特率	DS-0数量	数据比特率
T-1	24 DS-0s	24	1.544 Mbps
T-2	4 T-1s	96	6.312 Mbps
T-3	7 T-2s	672	44.736 Mbps
T-4	6 T-3s	4032	274.176 Mbps

密度超过 T-3 的 T 承载电路很不常见。在这样的速率，可以使用光承载电路。

3 SONET 和 OC 电路

同步光网络（SONET）的开发目的是希望使 T 承载系统提升到一个新的技术水平：光纤网络。SONET 基于 T-3 (44.736Mbps)的带宽，并稍稍超出一点，达到 51.84 Mbps。这被称为 OC-1 或 STS-1。如表 7-3 所示，更高速的 OC 电路是这个基本速率的倍数。

表 7-3 OC 电路载波

载波	等效的数据比特率	DS-0数量	数据比特率
OC-1	1 DS-3 (plus overhead)	672	51.840 Mbps
OC-3	3 DS-3s	2016	155.520 Mbps
OC-12	12 DS-3s	8064	622.080 Mbps
OC-48	48 DS-3s	32256	2488.320 Mbps
OC-192	192 DS-3s	129024	9953.280 Mbps

创造 SONET 也是努力使光电路标准化，但由于它的高成本，加上许多更新的方法也很有价值，例如密集波分复用(DWDM)，关于 SONET 的未来会有一些争论。

数字信令协议

对于任何 PSTN 电路，仅仅承载端点的（话音）数据是不够的，还应该提供端点之间通道状态信息的机制。(拆线和监视应答是两个必备的基本信令的例子；主叫号码显示是信令更复杂形式的一个例子）

1 随路信令(CAS)

CAS 也被称为 robbed-bit 信令，当 ISDN 不可用时就会用它在 T-1 上传送话音。CAS 模仿模拟信道，而没有利用强大的数字电路。CAS 的工作原理是窃取音频流的一些比特用于信令目的。虽然这对音频质量的影响可以忽略，但缺乏强大的信令通道限制了灵活性。

在配置 CAS T-1 时，每一端的信令选项必须匹配。E&M (Ear & Mouth or recEive & transMit)信令通常时首选的，因为它提供最好的监督。

CAS 几乎不会在用于 PSTN 电路了，因为 ISDN-PRI 更优越。CAS 的一个限制是不允许为不同的功能动态分配通道。还有主叫号码显示信息（甚至可能不支持）必须作为音频流的一部分传送。CAS 一般用在 channel banks 的 T-1 链路中，虽然有时可以用 PRI（更好）。

2 ISDN

综合业务数字网的历史有 20 多年了。由于 ISDN 将承载流量的通道（承载通道或 B 通道）与承载信令信息的通道分离开，ISDN 可以传递比 CAS 更丰富的特性。最初，ISDN 还许诺传递很多与 Internet 相同的功能，包括话音，视频和数据传输的高级功能。

不幸的是，电信设备制造商都各自决定在协议中增加专有的部分，寄希望与他们的版本更出众并最终主宰市场。结果，两个 ISDN 兼容系统的互连变成了痛苦和高代价的工作。要实现和支持这项昂贵技术的运营商逐步提高定价，所以 ISDN 没有被迅速采用。当前，除了基本中继以外很少使用 ISDN。ISDN 的首字母缩写成了业界的一个笑话：“它仍然什么也做不了

即使说了这么多，ISDN 作为中继还是很受欢迎，并且现在（基本上）是标

注兼容的。如果你的 PBX 有很多条线连接到 PSTN，使用 ISDN-PRI 电路可能是个好机会。还有，在不能用 DSL 或者电视电缆接入 Internet 的地方，ISDN-BRI 电路可以提供给你一个付得起的 128 kbps。在北美的很多地方，人们不喜欢使用 ISDN –BRI 接入 Internet ，而更喜欢 DSL 和电缆调制解调器，但在世界的其他地方 ISDN-BRI 还是很受欢迎。

A） ISDN-BRI/BRA

ISDN 设计的基本速率接口（或者基本速率访问）部分用于为工作站等端点提供服务。

ISDN 描述的 BRI 部分也常常简称为"ISDN"，但这可能引起混淆，因为 ISDN 是一个协议，而不是一种类型的电路（别说 PRI 电路也被正确地称为 ISDN！）

一个 ISDN 的基本速率由两个 64-kbps 的 B 通道和控制它们通道的 16-kbps D 通道组成，总共 144 kbps。

由于标准兼容性问题，技术复杂性问题以及缺乏文档，基本速率的 ISDN 电路在它的生命周期中一直是混乱的根源。在一些欧洲国家，ISDN-BRI 电路仍然是一种受欢迎的连接 PSTN 方式。

B） ISDN-PRI/PRA

ISDN 的主要速率接口部分用于提供在更大的网络连接时的服务。一个 ISDN 主要速率电路使用一个 DS-0 通道作为信令链路（D 通道）；其余的通道用作 B 通道。

在北美，ISDN 的主要速率常由一个或者多个 T-1 电路承载。由于一个 T-1 有 24 个通道，典型的北美 PRI 电路有 23 个 B 通道和 1 个 D 通道。由于这个原因，PRI 经常被称为 23B+D。

[*]PRI 实际比上述更加灵活一些，因为可以将一个 PRI 电路跨接到多个 PRI 的范围。这可以引入，例如，47B+D 电路（一个 D 通道控制两个 T-1）或者 46B+2D 电路（主备两个 D 通道控制一对 T-1）。有时你会看到 PRI 被描述为 nB+nD，因

为事实上 B 通道和 D 通道的数量是可变的。

欧洲使用 32 个通道的 E-1 电路，所以 ISDN 主要速率被称为 30B+D（最后一个通道用于同步）。

ISDN 主要速率由于技术的优势和有竞争力的价格，所以非常通用。如果你需要多于一打的 PSTN 连线，你应该查查 ISDN PRI 的价格。

从技术角度看，ISDN-PRI 总是优于 CAS.。

3　七号信令系统

电信运营商使用七号信令系统。七号信令系统在概念上与 ISDN 相似，它用于为运营商提供了一种在 ISDN 端点间传递必要附加信息的机制。但是，七号信令在技术上不同于 ISDN——一个大的区别是七号信令运行在完全独立的另一个网络上，而不是在实际承载呼叫的中继上。Asterisk 支持 7 号信令初露端倪，人们感兴趣的是如何使 Asterisk 与运营商的网络兼容.。7 号信令有一个开源的版本 (http://www.openss7.org)，但为了与 7 号信令完全兼容还有需要完成很多工作。写做本书的时侯,还不知道将来是否将 7 号信令集成到 Asterisk 中.。另一个有希望对 SS7 堤供支持的是 Sangoma Technologies 公司，他们公司的许多产品支持 SS7 功能。 应该注意的是在 Asterisk 中增加对 SS7 的支持并不像写一个正确的驱动程序那样简单。如果不对设备做极其严格的验证，设备接入 SS7 网络是不可能的。既便如此，是否有运营商会轻易地允许这样做也是存在疑问的。

7.4　包交换电话网

在上世纪九十年代中期，网络性能的提升使得媒体信息流可以在网络上实时传送。因为媒体流被划分为一个个的片段，这些小的片段在目的地址被重新组装起来。这样的网络被称为基于包交换的网络。包交换网络面临的挑战是，成千上万个包在网络的两端传送，如何确保包到达的顺序、如何确保时延少于 300ms，如果保证没有丢包。这些挑战对于 VoIP 来说是至关重要的。

7.5 结束语

本章探究了当今 PSTN 网络使用的技术。下一章我们将讨论 VoIP 协议：通过 IP 分组网络传输电话连接。不同的协议采用不同的机制实现电话会话，但是他们的意义远不只是实现电话会话而已。将电话网络融入数据网络最终将会消除电话和计算机的界线，它将引起我们通信方式的革命性演进。

第8章　VOIP 协议

因特网是一个不易控制的电话系统。

——Clifford Stoll

通信业已经发展超过 100 年了，而 Asterisk 将上个世纪使用的大多数——如果不是全部——的技术整合在了一起。要弄明白 Asterisk，你不需要对所有领域都精通。但是了解各种编码和协议将使你正确评价地它和整体地认识这个系统。

这一章将解释 VOIP 和 VOIP 网络与上章介绍的传统的语音交换网络不同的原因。我们将探讨 VOIP 协议的需求，简略介绍每种协议的历史和未来的发展。我们也将看到安全性的考虑和这些协议在各种网络拓扑中的工作能力，例如 NAT。所要讨论的 VOIP 协议如下：

- IAX
- SIP
- H．323
- MGCP
- Skinny/SCCP
- UNISRIM

编码的含义是它将语音转换成数字信号然后在因特网上传输。带宽在任何地方都是有限的，而任何特定连接的并发通话数目可以直接与使用的编解码类型有关。在这一章，我们也将探讨下列编码在带宽需求（压缩级别）和质量方面的区别。

- G.711
- G.726
- G.723.1
- G.729A
- GSM
- iLBC
- Speex

- MP3

接下来讨论如何对语音流量进行可靠路由，什么导致回音以及如何降低回音。还有 Asterisk 如何控制打入、打出电话的认证，并以次结束本章。

8.1 VOIP 协议的需求

VoIP 的最基本前提是为了在以因特网协议为基础的网络上传输而将语音流信息包化。达到这个目标的挑战在于人们的通话习惯。不但信号要以本质上相同的格式到达，并且必须到达的间隔必须低于 300ms。如果包丢失或延迟，通话质量将会降低。

所有被称为“因特网”的传输协议最初设计时的思想都不是实时媒体流的。终端期望通过等待时间长一点直至丢失的包到达，请求重发或在某些情况下确认发送的信息完好并简单地不再重发好包来来解决丢包的问题。在典型的语言会话中，这些机制并不适合。会话不能很好的适应丢失字母或词组，任何可感知的发送和接收间的延时。

传统的 PSTN 是为语音传输的目的特殊设计的，从技术观点看它完美的适合该任务。从终端的灵活性，无论如何，它的缺点对任何对技术有点理解的人都是显而易见的。VoIP 允许我们在其他所有在网络上传输的协议中进行语音通信。但是基于语言会话的特殊需求，特殊的功能能需要进行设置、构建，并在这些网络使用。

阻止基于包的语音传输的根本问题在于我们说话的方式和 IP 网传输数据的方式是完全不匹配的。说话和听话由语音流的交互组成。无论何处，因特网协议都是设计为将任何东西分片，将信息位元封装到数以千计的包中，然后将每个包通过所有可能的路径传输到远端。显然，某种形式的桥接是需要的。

8.2 VOIP 协议

传递一个 VoIP 连接的机制通常包括终端（网关处于中间）间的一系列信号传输，最终形成两个持久的承载实际对话的持续媒体流（每个方向一个）。有几个协议可以达到这个目标。本节我们将讨论其中的一些在 VoIP 和 Asterisk 中比较重要的协议。

IAX（“Inter-Asterisk-eXchange”）

当你想发这个协议的名字的音时。新手发“eye-ay-ex”，知道的人发“eeks”。IAX（确切地说，当前的版本是 IAX2，但对 IAX1 的支持已经放弃，因此无论是说"IAX"还是"IAX2"，都指的是版本 2）。是一个开放的协议，这意味着任何人可以下载和开发它，但它仍不是任何类型的标准。

在 Asterisk 中，IAX 通过 than_iax2.so 模块进行支持。

1 历史

IAX 协议是 Digium 开发用来与其他 Asterisk 服务器通信的（因此叫 Asterisk 内部交换协议）。IAX 是一个传输协议（非常像 SIP），它使用一个单独的 UDP 端口（4569）来传输通道信号和实时传输协议（RTP）流。正如下面讨论的，这使它穿越防火墙和大多数 NAT 网络更加容易，更适合于工作在 NAT 网络内。

IAX 还具备将多个会话在一个数据流中进行中继的能力，这在向远端的设备发送大量通道的时候具有极大的带宽优势。中继允许多个数据流用一个数据报报头来表示，因此降低了使用单独通道所带来的开销。这有助于减少延迟、减少所占用的处理实践和带宽，从而在两点之间有大量的活动通道的时候协议具有很好的扩展性。

2 未来

由于 IAX 对语音做了优化，它因为不支持视频而受到了批评。但实际上，IAX 拥有支持任何期望的媒体流的潜力。因为它是一个开放的协议，当有人期望时，未来媒体格式将必然融入进去。

3 安全考虑

IAX 可通过三种方式进行认证：简单文本、MD5 HASH、RSA 密钥交换。当然，这不是将终端间的媒体路径或头全部加密。许多解决方法都包括了采用虚拟专网（VPN）应用或软件将流加密到另一个技术层中的流。这需要终端预先设

定获得这些通道配置和选项的方法。在未来，IAX 可以将终端间的流通过 RSA 密钥交换或呼叫建立时的动态密钥，或自动密钥轮换来加密。这在创建和某个组织例如你的银行的安全连接方面非常有吸引力。然而，执法部门却愿意能够对这些连接进行某种程度的介入。

4 IAX 和 NAT

IAX2 协议专门设计了 NAT 平台中的设备的工作方法。为媒体的信号化和传输采用了一个单独的 UDP 端口，这也使需要在防火墙中开放的端口最少。这些考虑使 IAX 在安全网络的应用中成为最易实现的协议中的一种（如果不是最易用的）。

SIP

会话初始化协议（SIP）暴风骤雨般地占据了 VOIP 世界。做为在网络终端的 VoIP 协议的特别选择，最初仅被认为是一个有趣的想法的 SIP 现在看起来已经取代了 H.323 的主体地位。SIP 的前提是连接的每个终端是一个对等的实体（peer），然后他们间进行协议协商。使 SIP 引人注目的是它是一个相对简单的协议，与它语法类似的协议有 HTTP、SMTP 等。

在 Asterisk 中 SIP 通过 chan_sip.so 模块支持。

1 历史

SIP 是 1996 年一月 IETF 作为“draft-ietf-mmusic-sip-00”提出的。最初的草稿跟我们现在知道的 SIP 完全不像。它只有一种单一的请求类型：呼叫建立请求。1999 年 4 月，11 次修改后，SIP RFC 2543 诞生了。

最初，SIP 差一点被忽视。因为当时 H.323 被认为是 VoIP 传输的选择协议。无论怎样，随着成长，Sip 开始变得流行。并且随着许多不同的因素促进了它的成长，我们认为它成功的很大部分归功于它自由应用的特点。

2 未来

SIP 已经赢得了它的位置，它被认为是证明 VoIP 正确的协议。所有新用户和产品都希望支持 SIP。已经存在的任何产品现在都在忍受销量直到提供移植到 SIP 的路径。广泛地认为 SIP 远远提高了 VoIP 的能力，包括视频传输能力，音乐和任何形式的实时多媒体。后面几年内 SIP 将占据新应用的主体位置。

3 安全考虑

SIP 用户使用申请/回应（challenge/response）系统来认证用户。所有初始邀请都送到和预连接的设备相关的代理处。然后代理回送 407 代理认证请求消息，这里面包含了一个随机设置的作为“nonce”的字符串。这个 nonce 用来和密码生成一个 MD5 hash，然后将其在下一次“INVITE”请求中送回。如果 MD5 hash 和代理生成的匹配，这个客户端就认证通过。

拒绝服务（DoS）攻击可能是 VoIP 通信中遇到的最常用的攻击方式。当很大数目的 INVITE 请求被送到代理服务器处企图压跨系统时，DoS 攻击就发生了。这种攻击执行比较简单，而他们的效果在系统的用户处相当直接。SIP 有几种最小化 DoS 攻击效果的方法，但从根本上说，它们是不可能被组织的。

SIP 采用某种方案来确保安全。加密传输机制（Transport Layer Security,或 TLS）用来在主叫和被叫域中建立通信。除了这个，在网络本地安全的方针下，请求被安全的送到终端设备。注意加密媒体（这里指 RTP 流）在 SIP 自己范围内并必须独立处理。

关于 SIP 安全考虑的更多信息，包括注册劫持，服务器伪装，进程拆除，可以在 SIP RFC 3261 第 26 部分找到。

4 SIP 和 NAT

可能 SIP 技术上最大的障碍就是通过 NAT 层传输的挑战。因为 SIP 在它数据帧里对地址信息进行了加密，而 NAT 层处于网络的较低层，地址信息不能被修改，因此当 NAT 存在时，媒体流没有正确的地址信息用来完成连接。另外，

与 NAT 结合的防火墙通常不把到来的媒体流当作 SIP 传输的一部分而阻断该连接。

H.323

这个国际电联（ITU）的协议最初设计用来为视频会议提供 IP 传输机制。它成为了基于 IP 传输的视频会议设备的标准，并作为一个 VoIP 协议它非常出名。当 SIP 还是 H.323 将主导 VoIP 协议世界的争论越来越热时，在 Asterisk 中，由于对 IAX 和 SIP 的宠爱，H.323 被忽视了。在用户和企业中，H.323 并没有获得更大的成功，尽管它仍是最广泛应用的 VoIP 协议。

在 Asterisk 中，两个版本的 H.323 通过 chan_h323.so（Asterisk 提供）和 chan_oh323.so（作为一个免费插件存在）来支持。


你很可能在不知道的情况下使用过 H.323。微软的 NetMetting 客户端证明是最广泛应用的 H.323 客户端。

1　历史

H.323 在 1996 年 5 月被 ITU 发展成为一种通过基于 IP 同时和 PSTN 连通的网络来传输语音、视频、数据和传真通信的方式。从那时起，H.323 已经发展了几个版本和附件（给协议添加功能），并允许它在纯粹的 VoIP 网络和更广泛分布的网络中运行。

2　未来

H.323 的未来是一个热烈争论的课题。如果任何形式的媒体不能与 H.323 很好的配合，它几乎不会被提起（当然在 SIP 协议也是）。H.323 通常认为是 SIP 的上一级技术，但是，在如此多的其他协议中，这个可能最终并不要紧。一个使 H.323 不流行的因素是它的复杂性。尽管许多人争辩说曾经简单的 SIP 也开始遇到同样的问题。

迄今为止，H.323 仍然是世界范围内 VoIP 传输的主体。但是因为人们的通信需求越来越少地依赖传统的传输方式，H.323 的未来的任何确定的预测都变得更加困难。但 H.323 虽然可能不是新应用的选择协议，我们在将来的某天肯定要处理 H.323 的互通性问题。

3 安全考虑

H.323 是相对安全的协议，它不需要除了在因特网上通信使用的通常做法外的其他许多安全考虑。因为 H.323 使用 RTP 做媒体通信，它并不天生支持加密媒体路径。在终端间使用 VPN 或其他加密通道是最通常的加密通信方式。当然，需要在终端间建立安全通道有其缺点。这就是可能不方便（或甚至不可能）。既然 VoIP 在和金融机构例如银行的通信中更广泛的使用，我们希望更加提出广泛应用的 VoIP 协议来天生支持健壮的加密方法。

4 H.323 和 NAT

H.323 标准使用因特网工程任务组（IETF）RTP 协议来在终端间传输媒体。因此，当处理 NAT 时，H.323 和 SIP 有相同的问题。比较容易的方法是简单地通过 NAT 设备的适当端口来前向传输到内部客户端。

为了接收呼叫，总是需要转发 TCP 端口 1720 到客户端。另外，为了 RTP 媒体流和 RTCP 控制流（可以在你设备的手册中找到它的端口范围），还需要转发 UDP 端口。老一点的客户端，例如微软的 Netmeeting，也需要为 H.245 通道（也可以在客户端手册中找到端口范围）转发 RCP 端口。

如果 NAT 设备后有许多客户端，则需要使用一个工作在代理模式下的关守。网关需要一个连接到私有 IP 网络的接口和一个连接到公众因特网的接口。在私有 IP 的子网上的 H.323 客户端注册到网守，关守为客户端代理呼叫。注意，任何外部的客户端若想呼叫内部的客户端，也需要注册到代理服务器上。

现在，Asterisk 不能作为 H.323 的关守。你不得不使用一个独立的程序，如开源的 OpenH323 关守(http://www.gnugk.org)。

MGCP

多媒体网关控制协议（MGCP）也来自于 IETF。但 MGCP 的部署比人们想象的都要广泛。但是它还是快速地败给了 SIP 和 IAX 等协议。但是，Asterisk 喜欢协议，自然在根本上也支持这个协议。

MGCP 在 RFC 3435 中定义。它被设计成使终端设备（例如电话）尽可能的简单，所有呼叫的逻辑和处理都有多媒体网关和呼叫代理处理。与 SIP 不同，MGCP 采用集中模式。MGCP 电话不能直接拨打其他 MGCP 电话；他们必须总是通过某种形式的控制器进行。

Asterisk 通过 chan_mgcp.so 模块支持 MGCP，终端在配置文件 mgcp.conf 中定义。因为 Asterisk 只提供了基本的呼叫代理服务，它不能仿效一个 MGCP 电话（例如，作为一个用户代理注册到另一个 MGCP 控制器）。

如果已经部署了一些 MGCP 电话，可以通过 Asterisk 使用它们。如果计划将 MGCP 电话列入到 Asterisk 系统中的产品，记住，人们已经转移到更流行的协议，因此你需要做软件支持的需求预算。如果可能，（例如思科的电话），应该升级 MGCP 电话到 SIP。

专有的协议

最后，让我们看一个在 Asterisk 中支持的两种专有协议。

1 Skinny/SCCP

Skinny Client Control Protocol（SCCP）专用于思科 VoIP 设备。它是在思科 Call Manager PBX 中终端的默认协议。Asterisk 支持 Skinny，但是如果想连接思科电话到 Asterisk，若电话支持 SIP，通常建议获取每种电话的 SIP 镜像文件，使用 SIP 协议而不是 Skinny。

2 UNISTIM

作为 Nortel 专有的 VoIP 协议，UNISTM 最近加入到了 Asterisk。这个里程

碑意味着 Asterisk 是历史上第一个原生支持 VoIP 领域中最大的两个设备商——北电和思科的 IP 终端的 PBX。

8.3 编码和解码

编码通常理解为多种用来数字化（或压缩）模拟音频信息的数学模型。许多这种模型都是从人脑对不完整信息的想象能力提取出的。我们有视觉想象；同样，语音压缩的运算法则利用了我们相信我们应该比实际听到的更清楚的趋向。不同编码规则的目的是在效率和质量中取得平衡。

最初，编解码器 Codec 的术语涉及到一个编码器/解码器：设备在模拟和数字间转换。现在，该术语看起来更多和压缩/解压相关。

在我们深入到某个编码前，看一下表 8-1。它是你想涉及到的编码的集合。

表 8-1　编解码（Codec 快速检索）

编码	数据位速率(kbps)	是否需要许可?
G.711	64 kbps	否
G.726	16, 24, 或 32 kbps	否
G.723.1	5.3 或 6.3 kbps	是 (作为通路是不许要)
G.729A	8 kbps	是 (作为通路是不许要)
GSM	13 kbps	否
iLBC	13.3 kbps (30-ms frames) 或 15.2 kbps (20-ms frames)	否
Speex	Variable (between 2.15 and 22.4 kbps)	否

G.711

G.711 是 PSTN 的最基本编码。实际上，如果某人提到与电话网相关的 PCM（在前面章节讨论过），你应该想到 G.711。两种压缩方法被使用了：在北美使用 μ-law，世界其他地区使用 A-law。每种编码采用每秒 8000 次 8 位字符传输。如果你计算一下，你将看到这需要每秒传输 64000 位。

许多人将告诉你 G.711 是没有压缩的编码。这不完全正确，因为压缩扩展认为是压缩的一种形式。真实情况是 G.711 是其他所有编码起源的基础编码。

G．726

这个编码被多次扩展（它被用于 G.721，现在被废弃了），并且它是原始压缩编码的一个。它也被作为自适应差分脉冲编码调制（ADPCM）为人们所知，并且它可以以几个速率运行。最通常的费率是 16kbps，24kbps 和 32kbps。作为应用，Asterisk 现在只支持 ADPCM-32 速率，这距离此编码的最流行速率差得很远。

G.726 提供了和 G.711 相近的质量，但是它只占用了一半的带宽。这很可能因为与其提供量度后量化的结果，不如提供描述当前样本和前一个区别的信息。G.726 在 20 世纪 90 年代因为不能对调制解调器和传真进行支持而失宠。但因为它的带宽和 cpu 占用率，现在有回头的趋势。G.726 特别吸引人的地方是它不需要系统的大量计算工作。

G.723.1

不要和 G.723（另一种废弃的 ADPCM 版本）混淆，这个编码是为低码率设计的。它有两种数据比特率：5.3 和 6.3kbps。G.723.1 是依据 H.323 协议需求设计的编码中的一种（尽管其他编码可能是 H.323 挪用的）。它现在的阻力在于专利。因此如果在商业应用中使用需要许可。这意味着，尽管你可以通过 Asterisk 转接 2 个 G.723.1 电话，但你不允许在没有许可的情况下对其解码。

G.729A

考虑一下它使用了多小的带宽，G.729A 的语音质量给人留下了深刻印象。它通过利用 Conjugate-Structure Algebraic-Code-Excited Linear 预测(CS-ACELP，CELP 是压缩语音的一种常见方法。利用数学模型来模拟人类发生的各种发声方式，可以建立声音码表。取代发送实际的声音抽样，而是发送对应声音的代码。[当然不仅仅如此]。Jason Woodward 的语音编码网页(http://www-mobile.ecs.soton.ac.uk/speech_codecs/)提供了非数学方法的帮助信息。这些资料相当多，请做好思想准备。)做到了这点。因为版权原因，你不能在没有支付许可费的情况下使用它。无论怎样，它确实很流行，并且被许多不同的电话和系统很好地支持。

为了达到它激动人心的压缩率，这个编码需要同样让人激动的 CPU 占用量。在 Asterisk 系统中，使用深度压缩的编码将快速的使 CPU 陷入困境。

G.729A 使用 8kpbs 的带宽。

GSM

GSM 是 Asterisk 钟爱的编码。这个编码并不受像 G.723.1 和 G.729A 那样许可的阻碍，并且它对 CPU 占用方面的需求进行了考虑。它的语音质量通常被认为比 G.729A 产生的差一个等级，这只是个人观点，想确认还需要验证。

GSM 占用 13kbps 带宽。

iLBC

因特网低比特率编码（iLBC ，Internet Low Bitrate Codec）提供低带宽和质量的有效混合，并且它特别适合在失败的网络连接中保证一定的传输质量

一般来说，Asterisk 支持 iLBC，但是它没有 ITU 的编码方式那么流行，因此可能会与一般 IP 电话和商业 VoIP 系统不相容。IETF RFC 3951 和 3952 已经发布了对 iLBC 的支持，并且 iLBC 已经在 IETF 的协议栈中。

因为 iLBC 为达到高压缩率使用了复杂的算法，在 Asterisk 中是十分消耗 CPU 资源的。

如果你在没有交纳专利费用的情况下就使用 iLBC，iLBC 专利的所有者－－GIPS 希望知道你是否将它用于商业用途。你做这件事的方法是下载并打印一份 iLBC 的许可，填写并返回给 GIPS，如果你要读有关 iLBC 和它的相关许可，你可以登录它的主页：http://www.ilbcfreeware.org

因特网低比特率在 13.3kbp 和 15.2kbps

Speex

Speex 是一种动态比特率（VBR，Variable Bitrate）编码方式，它意味着可以根据网络环境的变化来动态的修改它的比特率。它在窄带和宽带中都提供版本，选择哪种方式取决于你想要的电话质量。

Speex 是一个完全免费的编码，许可是基于与 BSD 许可相反的 Xiph.org。

在网络中 Speex 是可以得到的，并且关于 Speex 的更多信息可以在 http://www.speex.org/上找到。

Speex 可以在 2.15 到 22.4kbps 上执行，因为它是一种动态比特率的编码方式。

MP3

可以确定的是，MP3 是一种编码方式。具体来说，它是 Moving Experts Group Audio Layer 3（如果想要了解 MPEG 音频，请在网上搜索 Davis Pan 的文章"A Tutorial on MPEG/Audio Compression."）编码标准。在 Asterisk 中，MP3 编码是

用于等待音乐（MoH，Music on Hold）的一种典型的编码方式。MP3 并不是一种电话编码方式，因为它是专用于音乐的，而不是声音。无论如何，它是一种在 VoIP 电话系统中做为传输在线音乐的非常流行的方法。

注意，在没有许可的情况下，音乐是不能广播的。很多人认为连接到广播电台或者 CD 作为等待音乐的声源是没有法律问题的，但事实上很可能是不对的。

8.4 服务质量

服务质量，即 QoS，已经开始受到越来越多的关注，它涉及到对网络中传递时间敏感信息流的挑战，该网络是以尽力而为的方式在 ad hoc 网络中传递数据为目标设计的。尽管没有什么严格的规则，但是一般来说，如果由说话者发出的声音在 300 微秒内到达接听者的耳朵，那么一个正常的会话流是可以进行的。当时延达到 500 微秒时，就很难避免彼此之间声音的冲突。如果时延超过一秒钟，那么一个正常的通话就会变得十分难以忍受。

除了要求声音流准时到达接听者之外，保证所传递信息的完成无缺性也是十分必要的。过多的丢包会导致远端无法重组出需要的声音波形，数据的间隙处也许会被认为是静音的，或者在比较特殊的情况下，会丢失所有的词和句子。

TCP，UDP，and SCTP

如果你要在基于 IP 的网络上传递数据，传输数据的过程中会使用到这里将要讨论的三个协议中的一个。

1 传输控制协议（TCP）

传输控制协议（TCP）几乎从未在 VoIP 中使用过，因为它尽管有保证传递的机制，但是在本质上它现在并不急于这么做。除非在两个终端之间的连接延迟很大，否则使用 TCP 将要引起的麻烦会远远多于它所能够解决的问题。

TCP 的目标是保证数据包的传输，为了做到这一点，实现了几种机制，比如包编号（用于数据重建），传输响应，对丢包的重传请求。在 VoIP 的世界里，迅速将数据包送达端点是十分重要的，但是使用蜂窝电话 20 年的经历让我们已经能够容忍一定的丢包率（在语音通信中，包到达的顺序很重要，因为音频是尽可能快地进行处理和发送给主叫方。不过，有了抖动缓冲，到达顺序就不是那么重要了，因为它提供了一个时间窗口，可以在发送给主叫之前来对数据包重新排序。）。

TCP 大量的过程管理、状态管理和到达的响应机制对传输大量数据十分的有效，但是对于实时媒体流的通信就不那么有效了。

2　用户数据报协议（UDP）

和 TCP 协议不同，用户数据报协议（UDP）并不提供任何形式的传输保证。包会被尽快地置于线路中，然后释放来寻找它们要到达目的地的方式，并没有任何的字段返回来说明它们是否已经到达目的地。正是因为 UDP 不提供任何形式证明数据包是否到达的保证，所以它可以在传输中以很小的时延获得很高的效率。

 注意：TCP 更多的是一个“社交响应”协议，因为带宽是十分均匀地分配给客户端和服务器。随着 UDP 传输百分比的增加，网络有可能会被压垮。

3　流控制传输协议 Stream Control Transmission Protocol

作为 IETF 批准的作为 RFC2960 的 SCTP 协议是一个相当新的传输协议。从表面上来看，它是以解决 TCP 和 UDP 的缺点为目标而设计的，特别是涉及到用于传输电路交换电话网络的服务类型时。

SCTP 的几个目标包括：

- 更好的阻塞避免技术（特别是避免拒绝服务的攻击）

- 严格的传输数据序列
- 低时延来促进实时传输

通过克服 TCP 和 UDP 的主要缺点，SCTP 的开发者们希望创造一种基于 IP 网络的传输 7 号信令和其他类型 PSTN 信号的健壮的协议。

差异服务

差异服务（DiffServ），并不是一种 QoS 机制，它不给流量做标记，或者做一些特殊的处理。显然，差异服务可以通过提供特定类型的包使其优先级高于其他包从而帮助提高 QoS。虽然这一定会增加 VoIP 在线路中快速通过的机会，但是它并不能保证传输率。

担保服务

最终的 QoS 保证是由 PSTN 提供的。对于每一个会话来说，一个 64kbps 的通道需要完全给一个呼叫使用－带宽是能够得到保证的。类似的，提供担保服务的协议能够保证服务建立连接所需要的带宽能够得到保障。正如所有的基于包传输的网络技术一样，这些协议在传输量小于最大承受流量时的操作性能是非常好的。当一个连接超过其限制值时，就不能避免服务质量的下降了。

1 MPLS

多协议标记交换是一种独立于 layer-3 路由表的工程网络传输方式，协议的工作流程为，给网络中的包分配一种很短的标记，包经过的路由将包向前传输到 MPLS 出路由，最终达到这些包所要求的目的地。一般来说，路由器并不依赖于网络中的 IP 表的每一跳来做出包前进方向的决定。在一个 MPLS 网络中，当包在入路由进入 MPLS 云时这种 lookup 模式仅仅会被执行一次。接下来数据包将会分配给一个数据流，涉及到一个标记交换路径（LSP），并且由该标记来进行区分。这个标记在 MPLS 的向前的路由表中会作为一个 lookup 模式的索引，然后包会不管 la’yer－3 的决定直接进入 LSP。这就使得大型网络的管理员更容易调节路由器的决定然后作出最利于网络资源利用的决定。另外，信息也与一个向

前传递包的优先级标签绑定。

2 RSVP

MPLS 并不包含动态建立 LSP 的方法，但是你可以使用带 MPLS 的限制协议（RSVP）。RSVP 是一个用于建立简单 LSP 和报告 MPLS 入路由问题的简单协议。在带有 MPLS 的连接中使用 RSVP 的优点是减少了管理员的管理。如果你不使用带有 MPLS 的 RSVP，你将不得不到每一个单一的路由手工配置每一个标签和路由。使用 RSVP 通过打乱路由器的控制标签使得网络更加的动态化。这也使得网络对环境变化有了更好的响应能力，因为它能基于某种条件改变路径的设置，例如，在某个路径不通时（也许是归因于错误的路由器）。接下来路由器内部的配置就可以使用 RSVP 来打乱在 MPLS 网络中路由器的新标签，而不需要经过人工的干预。

尽力而为

最简单、最经济的 QoS 的方法是根本就不提供它——“劲力而为”方法。尽管这听起来并不是一个好主意，但是它事实上却十分有效。任何在公众网上传输的 VoIP 呼叫几乎都在某种程度上是劲力而为的，而在这个环境中 QoS 方法还不是很常见。

8.5 回声

也许你还没有注意到它，但是自从电话诞生时起回声就已经成为 PSTN 中的一个重要的问题。也许你并不经常遇见带有回声的情况，因为电话界花费了大量的金钱来设计昂贵的回声消除装置。另外，当远端在物理上比较近时－－比如说，当你打电话给你临街的邻居时——时延会非常小以至于你所传输的信息会很快的再传回来，无法辨别出现在电话中的侧音（如第 7 章所讨论的，侧音视电话机的一种功能，你所说的话会部分返回到自己的耳朵，以便提供更自然的对话声音。）。

为什么会发生回声

在我们讨论处理回声的方法之前，让我们首先看一下在模拟电路中为什么会发生回声。

如果你听到了回声，并不是电话产生了这个问题，而是这个电路的远端产生的问题。相反的，在电路的远端产生的回声是由你这一端产生的。回声是由类似的区域回环电路必须传送和接受相似的线圈而产生的。如果这个电路是电力不平衡的，或者如果质量不好的电话连接在电路的终端，它接受到的信号就会被反射回来，成为往回方向信息的一部分。当这种反射的电路信号回到你这里时，你就会听到你刚刚才说过的话。人的耳朵可以分辨时延大于 40 微妙的回声。

在一个廉价的电话中，很可能会在电话的听筒处产生回声。这就可以解释为什么即使在完全端对端不包含模拟电话的线路中一些廉价的电话仍然会发出回声。在 VoIP 的世界里，回声经常是由某处连接的模拟电路或者低廉的终端发射自身的信号引起的（通过一个无线或者设计较差的听筒反馈）。一个好的规则是保持时延在 250 毫秒以下。

管理回声

在配置文件 zconfig.h 中，你可以选择若干回声消除算法中的一种，它的默认值是 MARK2。通过在你的网络上对比各种回声消除算法可以找到一种最适合你的环境的算法。Asterisk 的 zconfig.h 文件也有一个选项来保证回声消除更加有效。你可以不注释下面这行语句来使之生效。

#define AGGRESSIVE_SUPPRESSOR

注意回声消除会产生对讲机的半双工效应，因此此法只应该在其他回声消除方法失效的时候使用。

在 zapata.conf 文件中使对 zaptel 的回声消除生效。在文件中默认值“echocancel=yes”可以使回声消除生效。设置“echocancelwhenbridged-yes”可以使 TDM 呼叫的回声消除生效。当呼叫不需要回声消除的时候，这也可以增加呼叫的质量。

当回声消除生效时，在呼叫的过程中回声消除线性的依赖于回声。因此，在

呼叫开始时可能会听见一些回声，经过一段时间以后才会完全听不到回声。为了避免这种现象，可以使用一种叫做“回声训练”的方法，此方法可以在呼叫开始时简单的消除回声，然后发一个音，从这个音我们可以知道回声的量是多少。这就使得 Asterisk 处理回声的速度更快了。回声训练可以通过设置“echotraining=yes”生效。

8.6 Asterisk 和 VoIP

Asterisk 喜欢 VoIP 应该是一件不让人感到惊讶的事情。但是为了这么做，Asterisk 需要知道它在执行哪个函数：包括客户端的，服务器端的，或者是它们两个的。在 Asterisk 中最复杂并且经常容易混淆的概念就是出入系统的认证的命名。

Users，Peers 以及 Friends

连接中对我们的认证，或者我们认证连接方，在 iax.conf 和 sip.conf 文件中以 users 和 peers 的方式定义。如果双发都要认证，那么使用 friends 进行定义。当已经决定用哪种方式进行认证时，从 Asterisk 角度看，认清方向是十分重要的，因为连接是被 Asterisk 服务器接受和建立的。

1 Users

定义为 user 的连接是我们允许对我们进行连接的任何一个系统、用户或终端。记住 user 的定义并不提供呼叫该用户的方法，user 类型仅仅是用于为来电创建一个通道（在 SIP 中，并不总是这样。假如端点是一个 SIP 代理服务(不是用户代理)，Asterisk 会根据 peer 定义进行认证，把 SIP 头域中的 Contact 域里面的 IP 地址和端口与为 peer 所定义的主机名（以及端口，如果定义了的话；如果没有指定端口，就使用[general]配置段中所定义的端口号）进行匹配。请参考附录 A 中有关 SIP insecure 选项的讨论。）。user 定义要求定义 context 名称来定义表示来电呼叫的认证放置在拨号方案（在 extensions.conf 中）中的什么位置。

2 Peers

在一个连接中定义的 peer 类型是呼出连接。让我们这样来想：user 对我们呼叫，而我们呼叫 peer。因为 peer 对我们的呼叫，所以 peer 的定义并不一定要求一个 context 名字的配置。然而，有一个例外：如果从我们的系统产生呼叫通过一个环路又回到我们的系统（从 SIP 代理服务器产生，并不是从一个用户代理产生），那么此时，来电就要配与一个 peer 定义匹配。尽管最好为每个 peer 定义 context，但是名字为 default 的 Context 应该能对这些来电做相应的处理。

为了知道把向一个主机的呼叫发送到哪里，必须知道在网络它的位置（即 IP 地址），peer 的位置可以静态的定义也可以动态的定义。动态的 peer 在 peer 定义的开始用“host=dynamic”来定义。因为一个动态的 peer 的 IP 地址可能会经常的发生变化，所以它一定要注册到 Asterisk 上使 Asterisk 知道它的 IP 地址是什么。这样呼叫才能成功的路由到它那里。如果远端是另外一个 Asterisk，需要使用注册状态，接下来我们会讨论这个问题。

3 Friend

定义一个 friend 类型是对定义既是 user 类型又是 peer 类型的简捷方式。然而那些既是 user 又是 peer 的连接往往不这样定义，因为分别定义每一个方向的呼叫可以使得粒度更细。图 8-1 显示了与 Asterisk 有关的认证控制流。

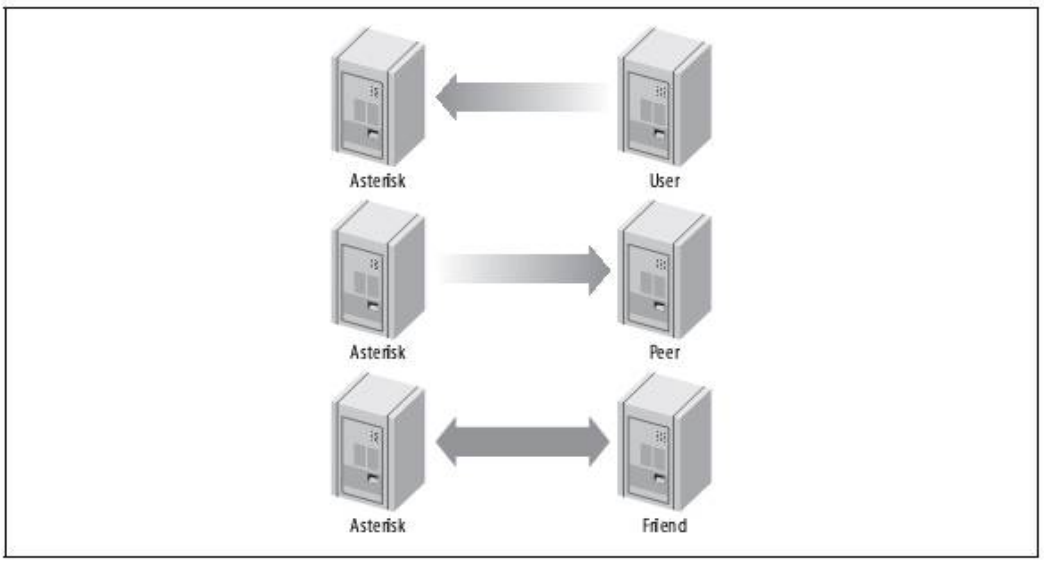


图 8-1 users、peers 和 friends 与 Asterisk 的呼叫初始关系

register 语句

register 语句告诉远端的 peer 你的 Asterisk 在网络中的位置，当你使用动态 IP 地址时或者远端的服务提供者没有你的 IP 地址的记录时，Asterisk 使用 register 语句来认证远端的服务提供者。有一些情况不需要 register 语句，但是通过演示可以证明 register 语句是需要的，让我们来看一个例子：

假如有一个远端的 peer 向你提供 DID 服务。当有人呼叫号码+1-800-555-1212 时，呼叫将会走物理的 PSTN 网络到达你的服务提供处或者 Asterisk 服务器，这个呼叫接下来就会经过 Internet 路由到你的 Asterisk 服务器。

你的服务提供者会在 sip.conf 或者 iax.conf 配置文件中有一个定义（决定你是否分别与 SIP 或者 IAX 协议相连）。如果你收到一个来自该服务器的呼叫，你就会把它们定义为 user（如果它们来自另外一个 Asterisk 系统，你可以在它们的系统中定义为 peer）。

接下来让我们来看一下当你使用一个动态的 IP 地址，你的服务器连接在本地网络上时的情况。你的服务提供者有一个静态的 IP 地址（或者是一个完全可靠的域名），这些你都已经写进你的配置文件中。因为你有一个动态的地址，你的服务提供者会在配置文件中配置“host=dynamic”。为了知道通过哪个路由进行+1-800-555-1212 呼叫，你的服务提供者需要知道你在网络中的什么位置。这就是要使用 register 语句的原因。

register 语句是认证的一种方式，并且能够告诉你你在哪里，在配置文件的[general]部分，你可以写一个类似下面的 register 语句：

register => username:secret@my_remote_peer

你可以在 Asterisk 控制台通过命令 iax2 show registry 和 sip show registry 来证实是否注册成功。

8.7 结束语

如果你在电话中听到嗡嗡声，你也许会认为 VoIP 是电话的未来。但是对于 Asterisk 来说，VoIP 并不仅仅是“在那里，做那个”这么简单。对 Asterisk 来说，

电话的未来是让人兴奋的。我们将会在第十一章展望未来。在下一章，我们将进入 Asterisk 更有革命性和强大概念的一部分：AGI－Asterisk 的网关接口。

第9章 Asterisk 网关接口（AGI）

尽管他想，这玩意确实不错，但是几乎所有的人对此保持沉默，而且都在想自己沉默的很明智。

——Douglas Adams, The Salmon of Doubt

Asterisk 网关接口（AGI）为外部程序控制 Asterisk 拨号方案提供了一个标准接口。AGI 脚本通常被用来做一些高级逻辑，与关系数据库（比如 PostgreSQL 和 MySQL）通信, 并且访问其他外部资源。让外部的 AGI 脚本去控制 Asterisk 拨号方案使得 Asterisk 比较容易的执行一些任务，如果没有 AGI 这项工作将会是复杂甚至是不可能的。

本章主要讲述 AGI 通信的基本原则，它不会教你怎样成为一个程序员，我们假定你已经是一名合格的程序员，这样我们可以向你介绍如果写 AGI 程序。如果你不了解如何进行编程，没有计算机编程基础，本章可能并不适合你，你可以跳过本章的学习内容进行下一章节。

在本章的学习过程中，我们会分别用 Perl、PHP、 Python 写一个 AGI 例程。需要说明的是，因为 Asterisk 为 AGI 脚本提供了一个标准接口，所以我们几乎可以用任何现代编程语言来写这些脚本。我们重点介绍 Perl、PHP、Python 是因为他们是开发 AGI 程序用的最多的程序设计语言。

9.1 AGI 通信的基本原则

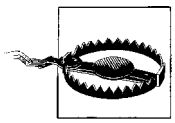
AGI 脚本通过通信通道与 Asterisk 通信，而不是通过 API 方式来编程。这些通道是一些文件指针，实际编程中有标准输入 STDIN、 标准输出 STDOUT 和标准错误 STDERR。很多程序开发人员都会认识这些通道，但以防你不熟悉这些，我们把他们包括在这里。

什么是 STDIN、STDOUT 和 STDERR?

STDIN 、STDOUT 和 STDERR 是一些通道，通过这些通道类似 UNIX 的环境可以从外部程序中接收和发送信息。STDIN 也就是标准输入，是指通过键盘或者其他程序送到本程序中的信息。在我们的例子中，从 Asterisk 传来的信息通过 STDIN 文件句柄自动进入到 STDIN 中。STDOUT 也就是标准输出，是 AGI 脚本用来把信息传回 Asterisk 的文件句柄。最后，AGI 脚本可以用 STDERR（标准错误）文件句柄把错误信息写回 Asterisk 控制台。

让我们总结一下这三个通信概念：

- AGI 脚本通过 STDIN 从 Asterisk 获取信息。
- AGI 脚本通过 STDOUT 把输出信息写到 Asterisk 中。
- AGI 脚本可以通过 STDERR 把调试信息送到 Asterisk 控制台。

 目前，在你的 AGI 脚本中向 STDERR 写信息只会在第一个 Asterisk 控制台输出。第一个控制台可以通过加参数-c 或 –r 启动。

这是非常不幸的，希望不久会被 Asterisk 开发者修补。如果你正在用 safe_asterisk 程序启动 Asterisk，事实上你很可能这样做，会在 TTY9 启动一个远端控制台（按 Ctrl-Alt-F9，你是否会看到一个 Asterisk 命令行界面）。这意味着所有的 AGI 调试信息只会在远端控制台上显示。你可能会在 safe_asterisk 中禁止这个控制台，从而可以在另外一个控制台看到调试信息。你也可能会出于安全原因禁止控制台，因为你可能不想让任何人可以接触 Asterisk 服务器并且在没有任何认证的情况下访问控制台。

AGI 通信的标准模式

Asterisk 和 AGI 脚本之间的通信遵循一个预先确定的模式。我们列举这些步骤，然后我们会通过 Asterisk 附带的 AGI 脚本例子来讲解这些步骤。

当一个 AGI 脚本启动时，Asterisk 向 AGI 脚本传送一系列变量及变量值，这些变量就像下例所示：

```
agi_request: test.py
```

agi_channel: Zap/1-1

agi_language: en

agi_callerid:

agi_context: default

agi_extension: 123

agi_priority: 2

传送完变量之后，Asterisk 会传送一个空行，这表示 Asterisk 已经完成传送变量并且 AGI 脚本开始控制拨号方案。

此时 AGI 脚本通过向 STDOUT 写信息的方式向 Asterisk 发送命令。AGI 脚本每发送一条命令，Asterisk 会向 AGI 脚本发送一条应答信息。在 AGI 脚本运行过程中，向 Asterisk 发送命令并且收到应答会一直持续。

你可能一直在问自己，通过 AGI 脚本可以使用哪些命令。这是个很好的问题，我们马上会概括这些基本命令。[*]

[*]要想获得可用的 AGI 命令列表，可以在 Asterisk 命令行界面中键入 show agi，你也可以参照附录 C 中关于 AGI 命令的参考。

在拨号方案中调用 AGI 脚本

要想正常运转，你的 AGI 脚本必须是可执行的。在拨号方案中运用 AGI 脚本，可以简单的调用 AGI( )，把 AGI 脚本的名字作为参数，象如下所示那样：

exten => 123,1,Answer( )

exten => 123,2,AGI(agi-test.agi)

AGI 脚本通常保存在 AGI 目录下(通常保存在/var/lib/asterisk/agi-bin)，但是你可以指定完整的 AGI 脚本目录。

AGI( ), EAGI( ), DeadAGI( ), and FastAGI( )

除了 AGI( )应用程序，还有一些其它的不同环境下使用的 AGI 应用程序。但是我们不会在本章中包含此内容，当你理解了基本的 AGI 脚本以后，你会认为其他它的 AGI 应用程序非常简单。

EAGI( )（增强型 AGI）的用法很像 AGI( )，但是它允许你的 AGI 脚本在文件描述符 3 读取收到的音频流信息。

DeadAGI ( )的用法也很像 AGI( )，但是它允许你的 AGI 脚本在死信道（比如，一个挂起的信道）中正常的运转。也就是说通用的 AGI( )不能在死信道中正常运转。

FastAGI( )应用程序允许通过网络调用你的脚本，这样在多 Asterisk 服务器的环境中，可以在某一位置集中调用这些 AGI 脚本。

在本章中，我们会首先介绍 Asterisk 附带的 agi-test.agi 脚本例子（用 Perl 写的脚本），然后用 PHP 写了一个关于天气报告的 AGI 程序，最后我们用 Python 写了一个数学游戏的 AGI 程序。

9.2　在 Perl 中编写 AGI 脚本

Asterisk 包含了名为 agi-test.agi 的例子。我们通过这个文件来了解 AGI 编程的核心概念。当然这个脚本文件是使用 Perl 语言编写的，不过请你记住，你以后所写的 AGI 程序可能使用更多的其他编程语言来完成。如果你需要证实其他语言来实现 AGI 就请微后看本章的其他部分。

现在开始!我们将提取出来代码中的每一段，并且介绍他们都是做什么用的。

! /usr/bin/per1

这一行告诉系统这脚本是使用 Perl 语言编写的，所以它应该使用 Perl 解释器来运行。如果你对 Linux 或 Unix 很在行，这句难不倒你！这一句假设，当然了，你的 Perl 执行程序肯定在/usr/bin/目录里。如果你的不是这样，就去寻找 Perl 解释器并且找到执行位置。

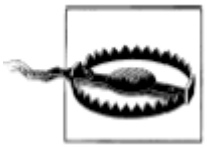
use strict;

这一句告诉 Perl 去执行一个更严格的语法检查，诸如使用了没有声明的变量都会被报告成为错误。其实这句并不是那么必要，只是如果有了它可以帮助你消除可能遇到的设计缺陷。

$|=1

这句告诉 Perl 不要去缓冲程序的输出字符，这样可以保证任何写操作都被立即的执行，不然系统就会等待其他数据库块先被输出完成后再处理。你会在本

章很多地方再看到这样的语法。



你必须一直使用不缓冲输出的方式来编写 AGI 脚本。另外，你的 AGI 程序可能并不象你想象的那样执行妥当，这是因为当你的程序发送输出给 Asterisk 等待 Asterisk 有所响应的时候，其实 Asterisk 并未收到任何东西。

```
# Set up some variables
my %AGI; my $tests = 0; my $fail = 0; my $pass = 0;
```

这里我们设置了四个变量。第一个是一个叫 AGI 的哈希结构，他从开始执行的有效期保存 Asterisk 传递来的变量。其他三个是标量变量，用来保存统计测试总数，失败数，通过数。

```
while(<STDIN>) {
        chomp;
        last unless length($_);
        if (/^agi_(\w+)\:\s+(.*)$/) {
                $AGI{$1} = $2;
        }
  }
```

就象我们前面讲过的一样，从开始的时候 Asterisk 会把一组数据传递给 AGI 程序。这个循环简单的取到每次的数据并且存储到哈希表 AGI 中。这些数据可以被程序以后使用或被忽略掉，但是从程序的逻辑结构角度上讲，他们应该被连续的从 STDIN 中读出。

```
    print STDERR "AGI Environment Dump:\n";
    foreach my $i (sort keys %AGI) {
          print STDERR " -- $i = $AGI{$i}\n";
    }
```

这句简单的打印出存储在 AGI 中的值到 STDERR。这个功能在调试模式很重要，因为 STDERR 的信息会被打印到 Asterisk 的控制台。

事实上是第一个 Asterisk 控制台（也就是，第一个使用-c 或-r 启动的

Asterisk）。如果使用 safe_asterisk 启动的 Asterisk 第一个 Asterisk 控制台应该在 TTY9 上。这句话的意思，你可能不会在远程连接里看到 AGI 的错误信息。

```
sub checkresult {
        my ($res) = @_;
        my $retval;
        $tests++;
        chomp $res;
        if ($res =~ /^200/) {
                $res =~ /result=(-?\d+)/;
                if (!length($1)) {
                        print STDERR "FAIL ($res)\n";
                        $fail++;
                } else {
                        print STDERR "PASS ($1)\n";
                        $pass++;
                }
        } else {
                print STDERR "FAIL (unexpected result '$res')\n";
                $fail++;
        }
```

这个子程序从 Asterisk 读取一个 AGI 命令获得的结果并且解码来判断是通过还是失败。

现在这只是我们去了解到 AGI 脚本核心逻辑路途中的起点。

```
print STDERR "1.    Testing 'sendfile'...";
print "STREAM FILE beep \"\"\n";
my $result = <STDIN>;
&checkresult($result);
```

上一个例子展示出如何使用 STREAM FILE 命令。这个命令告诉 Asterisk 发送一个声音文件给主叫方，就象使用 Background()做的。在这个案例中我们让

Asterisk 播放一个叫 beep.gsm 的文件。

Asterisk 会自己选择转换最容易并且能找的到的文件，所以不需要填写扩展名。

你可能注意到了第二个参数是两个被转义了的引号，如果没有他们俩，这个命令就无法被执行。

你必须把 AGI 命令所需要的所有参数都传递过去，如果你想少传几个那么必须用引号代替（有的时候需要适当的转义），就象上面的程序如果你不那么做程序肯定不会被执行。同样你还要在 print 的结尾打印出\n

在发送了 STREAM FILE 命令后，这个测试从 STDIN 读取结果并且呼叫 checkresult 子程序去检查 Asterisk 播放那个文件。STREAM FILE 命令需要三个参数，其中两个需要是：

- 要播放的声音文件名
- 主叫可以打断收听的数字键
- 开始播放声音的点，指定采样数字（可选）

简而言之，这个测试告诉 Asterisk 播放名字叫 beep.gsm 的文件，并且检查 Asterisk 确实执行了这个操作。

```
print STDERR "2.    Testing 'sendtext'...";
print "SEND TEXT \"hello world\"\n";
my $result = <STDIN>;
&checkresult($result);
```

这句告诉我们应该怎么使用 SEND TEXT 命令，用起来有点象 SendText()应用。这句将发送特定的文本给主叫方（除非主叫方不支持文本发送）。
SEND TEXT 接受两个参数：要发送给通道的数据。如果数据中有空格，必须进行转义。再次提醒您，一定要主意将转义符发送给 Asterisk。

```
print STDERR "3.    Testing 'sendimage'...";
print "SEND IMAGE asterisk-image\n";
my $result = <STDIN>;
&checkresult($result);
```

这个测试执行 SEND IMAGE 命令，他用起来象 SendImage()命令。他只需要图片文件的名称就可以把图片发给主叫方。 就象 SEND TEXT 命令，这个命令只能在主叫方支持接收图片的情况下。

```
print STDERR "4.    Testing 'saynumber'...";
print "SAY NUMBER 192837465 \"\"\n";
my
  $result = <STDIN>;
&checkresult($result);
```

这个测试将 SAY NUMBER 命令发送给 Asterisk。这个命令使用起来和 SayNumber()呼叫方案应用一样。他需要两个参数：

- 需要读出的数字
- 主叫可以打断收听的数字键

再次，因为我们没有提交任何数字键给第二个参数，所以我们需要发送转义内容。

```
print STDERR "5.    Testing 'waitdtmf'...";
print "WAIT FOR DIGIT 1000\n";
my $result = <STDIN>;
&checkresult($result);
```

这个测试实现了 WAIT FOR DIGIT 命令。这个命令将等待主叫方发送指定的 DTMF 数据（在一定毫秒内）。如果你不确定等待的时间使用-1。这个程序将返回输入的 ASCII 值。

```
print STDERR "6.    Testing 'record'...";
print "RECORD FILE testagi gsm 1234 3000\n";
my $result = <STDIN>;
&checkresult($result);
```

这部分代码让我们看到使用 RECORD FILE 命令。这个命令用来记录呼叫的语音，有点象 Record()呼叫方案应用。RECORD FILE 提取七个参数，最后三个是可选的：

- 被记录的文件名。
- 被记录的语音文件格式。
- 主叫可以打断收听的数字键。
- 超时时间，如果-1 就没有超时。
- 停止之前的记录的数字（可选）。
- BEEP 声音，如果你喜欢 Asterisk 在录音前先发出(可选)。
- Asterisk 知道用户已经完成记录并且返回的秒，不过超时参数将发送冲突并且没有 DTMF 数字被加入（可选）。这个参数必须在 s=.前面。

在这代码中，我们记录一个语音文件叫做 testagi(使用 GSM 格式)，接受 1 到 4 之间的数字来结束录音，并且最大记录 3000 毫秒。

```
print STDERR "6a.    Testing 'record' playback...";
        print "STREAM FILE testagi \"\"\n";
        my $result = <STDIN>;
        &checkresult($result);
```

这段的第二句使用 STREAM FILE 命令播放之前已经录好的音。我们之前已经讨论了 STREAM FILE，所以这了就不再说明了。

```
print STDERR "================== Complete ====================\n";
print STDERR "$tests tests completed, $pass passed, $fail failed\n";
print STDERR "================================================\n";
```

在最后的这一句，打印一个测试摘要到 STDERR 上，他将会在 Asterisk 的控制台显示出来。

在摘要中，你必须记得用 Perl 写 AGI 代码的几个事情：

- 打开严格的语法检查 use strict[*]

  [*]这句话适用在所有 perl 代码中，特别是象你一样刚开始学习 Perl 的人。
- 使用$|=1 关闭缓冲

- 使用 while(<STDIN>)读取 Asterisk 发来的数据
- 使用 print 函数发送命令
- 使用 print STDERR 发送调试信息到 Asterisk 控制台
- 有问题请到 www.perlchina.org 找国内的 Perl 高手

如果你对用 Perl 编写自己的 AGI 程序感兴趣，你一定会去查找 Asterisk::AGI 的 Perl 模块，他的作者是 james Golovich，在 http://asterisk.gnuinter.net 可以查到。这个 Asterisk::CGI 模块可以让你更容易的写 AGI 代码。

【译者注：在使用这个包中的 Asterisk::Manager 模块的时候如果开启了 Perl 的-w 选择可能因为警告导致程序报告错误，希望这个问题只有译者才遇到。

AGI 库很棒！但是它缺少一个对 Asterisk 奇怪的配置文件操作的函数。幸运的是我们还有一个 Asterisk::config 来解决这个问题，他的作者是孙冰，在 http://search.cpan.org/~hoowa/可以查到。使用 ActivePerl 的用户可以直接使用 PPM 命令进行安装，它是一个标准的 CPAN 模块。】

9.3　在 PHP 中产生 AGI 脚本

我们说过会使用几种语言，那么让我们先看看怎么用 PHP 写 AGI 脚本。AGI 程序设计的基本原则没有变化，只是程序开发的语言改变了。在本例中，我们会写一个 AGI 脚本来从互连网下载天气报告，把天气、风向、风速信息返回给调用者。

```
#!/usr/bin/php -q
<?php
```

第一行告诉系统将要用 PHP 来运行这个脚本，参数-q 屏蔽了 HTML 错误信息。你需要确保在第一行和开始的 PHP 标签之间没有额外的行，否则 Asterisk 不能正常解析。

```
# change this to match the code of your particular city
# for a complete list of US cities, go to
# http://www.nws.noaa.gov/data/current_obs/
$weatherURL="http://www.nws.noaa.gov/data/current_obs/KMDQ.xml";
```

这几行告诉 AGI 脚本到哪里去获取现在的天气情况。在本例中，我们获取的是阿拉巴马州亨城的天气。可以自由访问上述列举的网站，从而得到在美国提供此天气服务的城市的完整列表。[*]

[*]我们对于美国以外的读者表示歉意，因为我们用了一个只在美国提供天气情况的服务。如果你找到了一个使用 XML 格式提供国际天气情况的服务，修改 AGI 脚本来使用这项服务不会很复杂。当我们找到了这样的服务，我们会在以后版本中更新这个脚本。

```
# don't let this script run for more than 60 seconds
set_time_limit(60);
```

在这里我们告诉 PHP 不要让这个程序运行超过 60 秒，这是一个安全网，如果因为某些原因超过了 60 秒，脚本会终止。

```
# turn off output buffering
ob_implicit_flush(false);
```

这个命令会关闭输出缓冲区，意味着所有数据会被立即发送到 AGI 接口并且不会被缓存。

```
# turn off error reporting, as it will most likely interfere with
# the AGI interface
error_reporting(0);
```

这个命令会关闭所有错误报告，因为它可能会干扰 AGI 界面。（你可能会发现在测试过程中注释掉这行会非常有用。）

```
# create file handles if needed
if (!defined('STDIN'))
{
    define('STDIN', fopen('php://stdin', 'r'));
}
if (!defined('STDOUT'))
{
```

```
    define('STDOUT', fopen('php://stdout', 'w'));
}
if (!defined('STDERR'))
{
    define('STDERR', fopen('php://stderr', 'w'));
}
```

这段代码确保我们有打开的 STDIN、STDOUT 和 STDERR 文件句柄，这些句柄会处理 Asterisk 和我们的脚本之间的所有通信。

```
# retrieve all AGI variables from Asterisk
while (!feof(STDIN))
{
    $temp = trim(fgets(STDIN,4096));
    if (($temp == "") || ($temp == "\n"))
    {
        break;
    }
    $s = split(":",$temp);
    $name = str_replace("agi_","",$s[0]);
    $agi[$name] = trim($s[1]);
}
```

接下来我们会读取 Asterisk 传来的所有变量，在 PHP 中用 fgets 命令可以从 STDIN 读取数据，我们会把每一个变量存入名为$agi 的哈希变量中。我们可以在 AGI 脚本的逻辑中使用这些变量，在本例中我们并没使用这些变量。

```
# print all AGI variables for debugging purposes
foreach($agi as $key=>$value)
{
    fwrite(STDERR,"-- $key = $value\n");
    fflush(STDERR);
```

```
}
```

在这里为了调试我们把这些变量打印到 STDERR。

```
#retrieve this web page
$weatherPage=file_get_contents($weatherURL);
```

这行代码从国家气象服务中重新取得 XML 文件并且把这些内容存到名为 $weatherPage 的变量中。随后这个变量将被用来摘录几条我们想要的天气报告。

```
#grab temperature in Fahrenheit
if (preg_match("/<temp_f>([0-9]+)<\/temp_f>/i",$weatherPage,$matches))
{
    $currentTemp=$matches[1];
}
```

这段代码通过使用 preg_match 命令，从天气报告中摘录了气温（华氏温度）。这个命令使用与 Perl 兼容的正则表达式[*]来摘录所需要的数据。

[*] 关于正则表达式最好的指南是 O'Reilly 出版、Jeffrey Friedl 所著的《精通正则表达式》。

```
#grab wind direction
if (preg_match("/<wind_dir>North<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='northerly';
}
elseif (preg_match("/<wind_dir>South<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='southerly';
}
elseif (preg_match("/<wind_dir>East<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='easterly';
```

```
}
elseif (preg_match("/<wind_dir>West<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='westerly';
}
elseif (preg_match("/<wind_dir>Northwest<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='northwesterly';
}
elseif (preg_match("/<wind_dir>Northeast<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='northeasterly';
}
elseif (preg_match("/<wind_dir>Southwest<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='southwesterly';
}
elseif (preg_match("/<wind_dir>Southeast<\/wind_dir>/i",$weatherPage))
{
    $currentWindDirection='southeasterly';
}
```

风向可以通过使用 preg_match（位置在 wind_dir 目录下）来得到并且赋值到 $currentWindDirection 变量中。

```
#grab wind speed
if
(preg_match("/<wind_mph>([0-9.]+)<\/wind_mph>/i",$weatherPage,$matches))
{
    $currentWindSpeed = $matches[1];
}
```

最后我们得到当前风速并把它赋值到变量$currentWindSpeed 中。

```
# tell the caller the current conditions
if ($currentTemp)
{
    fwrite(STDOUT,"STREAM FILE temperature \"\"\n");
    fflush(STDOUT);
    $result = trim(fgets(STDIN,4096));
    checkresult($result);
    fwrite(STDOUT,"STREAM FILE is \"\"\n");
    fflush(STDOUT);
    $result = trim(fgets(STDIN,4096));
    checkresult($result);
    fwrite(STDOUT,"SAY NUMBER $currentTemp \"\"\n");
    fflush(STDOUT);
    $result = trim(fgets(STDIN,4096));
    checkresult($result);
    fwrite(STDOUT,"STREAM FILE degrees \"\"\n");
    fflush(STDOUT);
    $result = trim(fgets(STDIN,4096));
    checkresult($result);
    fwrite(STDOUT,"STREAM FILE fahrenheit \"\"\n");
    fflush(STDOUT);
    $result = trim(fgets(STDIN,4096));
    checkresult($result);
}

if ($currentWindDirection && $currentWindSpeed)
{
    fwrite(STDOUT,"STREAM FILE with \"\"\n");
```

```
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
        fwrite(STDOUT,"STREAM FILE $currentWindDirection \"\"\n");
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
        fwrite(STDOUT,"STREAM FILE wx/winds \"\"\n");
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
        fwrite(STDOUT,"STREAM FILE at \"\"\n";)
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
        fwrite(STDOUT,"SAY NUMBER $currentWindSpeed \"\"\n");
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
        fwrite($STDOUT,"STREAM FILE miles-per-hour \"\"\n");
        fflush(STDOUT);
        $result = trim(fgets(STDIN,4096));
        checkresult($result);
    }
```

既然我们已经收集到所需要的数据，我们可以向 Asterisk 发送 AGI 命令（发送的时候检查结果），通过使用 AGI 命令 STREAM FILE 和 SAY NUMBER，我们可以把当前的天气状况传递给调用者。

我们曾经说过，现在还要再强调一次：当我门调用 AGI 命令的时候，你必须传入所有需要的参数。在本例中，STREAM FILE 命令和 SAY NUMBER 命令都需要一个额外的参数，我们会传入空引号并且有个反斜杠。

我们还应当注意每次向 STDOUT 写信息的时候我们可以调用 fflush 命令。这样做可以认为是多余的，但是确保 AGI 命令不被缓存并立即被发送到 Asterisk 是没有坏处的。

```
function checkresult($res)
{
    trim($res);
    if (preg_match('/^200/',$res))
    {
        if (! preg_match('/result=(-?\d+)/',$res,$matches))
        {
            fwrite(STDERR,"FAIL ($res)\n");
            fflush(STDERR);
            return 0;
        }
        else
        {
            fwrite(STDERR,"PASS (".$matches[1].")\n");
            fflush(STDERR);
            return $matches[1];
        }
    }
    else
    {
        fwrite(STDERR,"FAIL (unexpected result '$res')\n");
        fflush(STDERR);
        return -1;
    }
```

}

函数 checkresult 和我们在 Perl 例子中的 checkresult 子程序有着相同的目的，从名字可以猜到这个函数的功能是当我们调用一个 AGI 命令时，这个函数检查 Asterisk 返回的结果。

?>

在文件的最后是一个 PHP 结束标签，在结束标签之后不要放置任何空行，这会干扰 AGI 接口。

到目前为止我们已经涉及了两种语言，为了示范用 PHP 开发一个 AGI 脚本和 Perl 开发一个脚本的相同点和不同之处，我们在用 PHP 写 AGI 脚本的时候必须记住以下几点：

- 调用 PHP 的时候加上参数-q 会关闭 HTML 错误信息。
- 关闭时间限制或者设置成一个合理的值（对于新版 PHP，在命令行调用 PHP 时会自动会禁止掉时间限制）。
- 用 ob_implicit_flush(false)命令关闭输出缓冲。
- 打开 STDIN、STDOUT 和 STDERR 的文件句柄（新版 PHP 会有一个或多个已经打开的这样的文件句柄。仔细检查上述代码使得这段程序可以平滑的移植到大多数 PHP 版本上）。
- 用 fgets 函数从 STDIN 读取变量。
- 用 fwrite 函数向 STDOUT 和 STDERR 输出信息。
- 通常在向 STDOUT 和 STDERR 输出信息之后会调用 fflush 函数。

PHP 语言的 AGI 库

如果想使用 PHP 语言进行高级 AGI 编程，你可以查看 PHPAGI 工程，网址是 http://phpagi.sourceforge.net 。这个工程最开始是 Matthew Asham 开发的，现在由 Asterisk 社区的几个成员开发。

9.4 在 Python 中写 AGI 脚本

我们将用 Python 写 AGI 的脚本，这被称为“减法游戏”。这是 Ed Guy 所写的一个 Perl 程序，并在 2004 年的 AsteriCon 会议中被讨论过。Ed 热情洋溢地描述了他的强大而简单的 Asterisk 系统，当时他发现他用一个快速的 Perl 脚本来帮助他年幼的女儿提高她的数学能力。

既然我们已经用 AGI 写了一个 Perl 程序并且 Ed 已经用 Perl 写了数学程序，我们发现我们需要把它限定在 Python 上了！让我们看看我们的 Python 脚本。

```
#!/usr/bin/python
```

这一行程序告诉系统，用 Python 编译器运行该脚本。对于小的脚本，你可以考虑增加-u 选项，从而运行 Pythonzai 非 buffer 模式。但是我们并不推荐这么做，对于大的或者经常用的 AGI 脚本，会影响系统性能。

```
import sys
import re
import time
import random
```

这里，我们要调用一些库（library）来用在我们的 AGI 脚本里面。

```
# Read and ignore AGI environment (read until blank line)
env = {}
tests = 0;
while 1:
line = sys.stdin.readline( ).strip( )
if line == '':
break
key,data = line.split(':')
if key[:4] <> 'agi_':
#skip input that doesn't begin with agi_
sys.stderr.write("Did not work!\n");
sys.stderr.flush( )
continue
```

```
        key = key.strip( )
        data = data.strip( )
        if key <> '':
        env[key] = data
        sys.stderr.write("AGI Environment Dump:\n");
        sys.stderr.flush( )
        for key in env.keys( ):
        sys.stderr.write(" -- %s = %s\n" % (key, env[key]))
        sys.stderr.flush( )
```

这个部分的代码将在变量里面读，这些变量将从 Asterisk 传送到我们的脚本。这些将被保存在 env 的目录里面。为了调试方便，这些值被写到 STDERR。

```
        def checkresult (params):
        params = params.rstrip( )
        if re.search('^200',params):
        result = re.search('result=(\d+)',params)
        if (not result):
        sys.stderr.write("FAIL ('%s')\n" % params)
        sys.stderr.flush( )
        return -1
        else:
        result = result.group(1)
    #debug("Result:%s Params:%s" % (result, params))
        sys.stderr.write("PASS (%s)\n" % result)
        sys.stderr.flush( )
        return result
        else:
        sys.stderr.write("FAIL (unexpected result '%s')\n" % params)
        sys.stderr.flush( )
        return -2
```

校验结果功能几乎和 Perl AGI 脚本样本程序中 checkresult 子程序的结果几乎一样。它在 Asterisk 命令行的结果里面读。扩展答案，并报告命令行是否正确。

```
def sayit (params):
sys.stderr.write("STREAM FILE %s \"\"\n" % str(params))
sys.stderr.flush( )
sys.stdout.write("STREAM FILE %s \"\"\n" % str(params))
sys.stdout.flush( )
result = sys.stdin.readline( ).strip( )
checkresult(result)
```

sayit 功能是一个简单的和 STREM FILE 命令有关的交换。

```
def saynumber (params):
sys.stderr.write("SAY NUMBER %s \"\"\n" % params)
sys.stderr.flush( )
sys.stdout.write("SAY NUMBER %s \"\"\n" % params)
sys.stdout.flush( )
result = sys.stdin.readline( ).strip( )
checkresult(result)
```

saynumber 功能是和 SAY NUMBER 命令有关的简单交换。

```
def getnumber (prompt, timelimit, digcount):
sys.stderr.write("GET DATA %s %d %d\n" % (prompt, timelimit,
digcount))
sys.stderr.flush( )
sys.stdout.write("GET DATA %s %d %d\n" % (prompt, timelimit,
digcount))
sys.stdout.flush( )
result = sys.stdin.readline( ).strip( )
result = checkresult(result)
sys.stderr.write("digits are %s\n" % result)
sys.stderr.flush( )
if result:
```

```
return result
else:
result = -1
```

getnumber 功能呼叫 GET DATA 命令来获得来自于呼叫的 DTMF 输入。这被用来获得呼叫者并压缩问题。

```
limit=20
digitcount=2
score=0
count=0
ttanswer=5000
```

这里，我们初始化几个变量在程序中使用。

```
starttime = time.time( )
t = time.time( ) - starttime
```

这两行中我们设置 starttime 变量来初始化 t 为 0。我们将用 t 变量，当 AGI 脚本开始的时候，按秒计时。

```
sayit("subtraction-game-welcome")
```

然后，我们欢迎呼叫者来做递减游戏。

```
while ( t < 180 ):
big = random.randint(0,limit+1)
big += 10
subt= random.randint(0,big)
ans = big - subt
count += 1
#give problem:
sayit("subtraction-game-next");
saynumber(big);
sayit("minus");
saynumber(subt);
res = getnumber("equals",ttanswer,digitcount);
if (int(res) == ans) :
```

```
score+=1
sayit("subtraction-game-good");
else :
sayit("subtraction-game-wrong");
saynumber(ans);
t = time.time( ) - starttime
```

这是 AGI 脚本的核心。我们从这个部分的代码和压缩问题给呼叫者循环，直到 180 秒结束。在循环开始的时候，我们计算两个不一样的随机数。然后我们向呼叫者提问，并读出呼叫者的回答。如果呼叫者回答不正确，我们给出正确回答。

```
pct = float(score)/float(count)*100;
sys.stderr.write("Percentage correct is %d\n" % pct)
sys.stderr.flush( )
sayit("subtraction-game-timesup")
saynumber(score)
sayit("subtraction-game-right")
saynumber(count)
sayit("subtraction-game-pct")
saynumber(pct)
```

在用户回答完压缩问题，他们给出他们的打分。

能够看出，你需要记住的基础的东西是在你用 Python 写 AGI 脚本时，

清空输出 buffer。这将加强当你的 Asterisk 正在等待缓存填充过程中，并且 Python 正在等待 Asterisk 反馈时，你的 AGI 程序不会挂起。

从 Asterisk 的 sys.stdin.readline 命令中读数据。

用 sys.stdout.write 命令向 Asterisk 写命令。不要忘记在等待以后呼叫 sys.stdout.flush。

Python 的 AGI 库

如果你正计划写很多 Python AGI 代码，你想检查 Karl Putland 的 Python 模式，Pyst。你可以在 http://www.sourceforge.net/projects/pyst/查到。

9.5　在 AGI 中调试

调试 AGI 程序，就象和其他程序一样让人灰心厌倦。幸运的是，还好我们有两个有利条件去调试 AGI 脚本。首先，所有的通讯都是在 Asterisk 和 AGI 程序之间碰巧都是使用 STDIN 和 STDOUT(当然还有 STDERR)实现的，你可以直接在操作系统中执行 AGI。其次，Asterisk 可以通过 agi debug 命令轻易的获得自己和 AGI 通讯的信息。

从操作系统中调试

就象上面提到的一样，你可以在操作系统中直接运行程序看到结果。调试的方式就象在 Asterisk 上一样，为你的脚本提供：

- 一个变量和值的列表，就象 agi_test:1 这样。
- 一个空行（/n）表示这些是变量
- 响应 AGI 中命令的每一个请求，典型的就是 **200 response=1** 这样就好。

在操作系统中直接调试程序，可以更容易的发现自己程序的问题。

使用 Asterisk 的 agi 调试命令

Asterisk 的命令行接口可以提供很有用的命令来帮助调试 AGI 程序，他叫做 agi debug。如果你能在 Asterisk 控制平台下执行 agi debug 并且运行 AGI 程序，你将会看到象这些信息：

```
    -- Executing AGI("Zap/1-1", "temperature.php") in new stack
     -- Launched AGI Script /var/lib/asterisk/agi-bin/temperature.php
AGI Tx >> agi_request: temperature.php
AGI Tx >> agi_channel: Zap/1-1
```

```
AGI Tx >> agi_language: en
AGI Tx >> agi_type: Zap
AGI Tx >> agi_uniqueid: 1116732890.8
AGI Tx >> agi_callerid: 101
AGI Tx >> agi_calleridname: Tom Jones
AGI Tx >> agi_callingpres: 0
AGI Tx >> agi_callingani2: 0
AGI Tx >> agi_callington: 0
AGI Tx >> agi_callingtns: 0
AGI Tx >> agi_dnid: unknown
AGI Tx >> agi_rdnis: unknown
AGI Tx >> agi_context: incoming
AGI Tx >> agi_extension: 141
AGI Tx >> agi_priority: 2
AGI Tx >> agi_enhanced: 0.0
AGI Tx >> agi_accountcode:
AGI Tx >>
AGI Rx << STREAM FILE temperature ""
AGI Tx >> 200 result=0 endpos=6400
AGI Rx << STREAM FILE is ""
AGI Tx >> 200 result=0 endpos=5440
AGI Rx << SAY NUMBER 67 ""
    -- Playing 'digits/60' (language 'en')
    -- Playing 'digits/7' (language 'en')
AGI Tx >> 200 result=0
AGI Rx << STREAM FILE degrees ""
AGI Tx >> 200 result=0 endpos=6720
AGI Rx << STREAM FILE fahrenheit ""
AGI Tx >> 200 result=0 endpos=8000
    -- AGI Script temperature.php completed, returning 0
```

你将看到 AGI 程序运行的三行类型内。第一个就是以 AGI TX >>开头的，这行表示 Asterisk 发送给你程序的 STDIN 的信号。第二行就是以 AGI RX <<开头的，这表示你的 AGI 程序把内容写给 Asterisk（使用 STDOUT）。第三个就是，这个是 Asterisk 标准消息表示它在执行对系统来讲可靠的命令。

如果想禁用 AGI 调试模式就在 Asterisk 控制平台上简单的输入 agi no debug 即可。

使用 agi debug 命令可以允许你看到 Asterisk 和你的程序之间的通讯，将会对你的调试起到非常有用的作用。希望上面的两招能帮助你写出来强大的 AGI 程序。

9.6 结束语

对于一个开发者来讲，AGI 技术绝对是一个革命性的让人完全有理由采用 Asterisk 实现企业电话交换机(PBX)的条件。当然，AGI 也只是描述整个远景的一部分。对那些人来说只有一些开发者和更多的系统管理员或大型用户，第十章将探讨 Asterisk 对更多人能产生的价值。敬请查看！

第10章 Asterisk 的外部扩展工具

刚开始的 90%的工作使用 90%的时间完成，最后的 10%的工作却要另外 90%的时间完成。

—The nighty，Ten

写作这本书，最头痛的部分不是去找材料往下写，而是决定我们什么不写在这本书里。现在，我们已经讨论完基础知识，你将知道事实如下：除了 Asterisk 之外我们没有任何东西没有可告诉你的。好，或许 5%，或许更少，请理解，并不是因为我们不想给你更好的，实在是因为 Asterisk 本身就是无限的（至少我是这样认为的）。

在本章中，我们将介绍 Asterisk 里面的奇迹。本章中的每一部分几乎都可以写成一本书（它将成为一本书，如果 Asterisk 如我们想的那样成功）。

10.1 Festival

Festival 是一个流行的开源的文本到语音的引擎。在 Asterisk 中使用 Festival 的前提是你的拨号方案把文本内容发送到 Festival，这样 Festival 就能够将一个文本读出来，Festival 最大的用处就是当你在路上的时候，它能将你的 E-mail 自动读给你听。

开始为 Asterisk 安装好 Festival

有两种方法能够得到带 Festival 的 Asterisk。

第一种（也是最容易的）方法不用打补丁，并重新编译——添加以下文本内容到 Festival 配置文件（文件名：festival.scm，文件路径：/etc/ 或者 /usr/share/festival/）

(define (tts_textasterisk string mode)

"(tts_textasterisk STRING MODE) Apply tts to STRING. This function is specifically designed for use in server mode so a single function call may synthesize

the string. This function name may be added to the server safe functions."

(let ((wholeutt (utt.synth (eval (list 'Utterance 'Text string)))))

(utt.wave.resample wholeutt 8000)

(utt.wave.rescale wholeutt 5)

(utt.send.wave.client wholeutt)))

你可以将以上语句放置在文档的任何位置，除了两个圆括号之间。

第二种方法是用一个 Asterisk 的特殊补丁，来编译 Festival（Asterisk 源代码的 contrib/directory）。

以上两种方法包含在 README.festival 文件中，文件放在 Asterisk 源代码的 contrib/directory 下。

对于这两种方法，你都需要在 festival.scm 文件中修改 Festival 访问列表。方法是搜索“localhost”关键字，并用你的服务器的域名来替换它。

以上两种方法都能够正确配置 Festival 同 Asterisk 进行通信。当完成配置 Festival 的工作后，你就启动 Festival 服务器。你就可以在你的拨号方案中调用 Festival( )应用了。

为 Festival 配置 Asterisk

Asterisk 对于 Festival 的配置是通过一个叫做 festival.conf 的文件进行的。在这个文件中，你需要指定你的主机名称和你的 Festival 服务器端口，最好还有 Festival 朗读的缓存配置。对于大多数的安装情况（如果你将要在你的 Asterisk 服务器上运行 Festival），默认值就能够工作的很好。

启动 Festival 服务器

为了调试目的，启动 Festival 服务，简单的带上—server 参数运行 Festival 即可：

[root@asterisk ~]#　festival --server

一旦确定你有 Festival 服务已经在运行，并且没有拒绝你的连接，你可以通过以下方式启动你的 Festival：

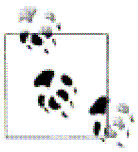
[root@asterisk ~]#　festival_server 2>&1 >/dev/null &

从拨号方案中调用 Festival

现在 Festival 已经配置好了，并且 Festival 服务已经开始，让我们从一个简单的拨号方案中对 Festival 进行调用：

exten => 123,1,Answer( )

exten => 123,2,Festival(Asterisk and Festival are working together)

 为了确定通道已经建立了，你必须先调用 Festival()应用之后后再调用 Answer()应用。

当 Asterisk 连接到 Festival，你应当在终端中看到以下输出：

[root@asterisk ~]# festival --server

server Sun May 1 18:38:51 2005 : Festival server started on port 1314 client（1） Sun May 1 18:39:20 2005 : accepted from asterisk.localdomain client（1） Sun May 1 18:39:21 2005 : disconnected

如果看到以下输出,则意味着你没有在 festival.scm 中添加这台主机到访问列表：

[root@asterisk ~]# festival --server

server Sun May 1 18:30:52 2005 : Festival server started on port 1314

client(1) Sun May 1 18:32:32 2005: rejected from asterisk.local domain not in access list

在 Asterisk 中使用 Festival 的其他方法

Asterisk 社团中的一些人成功的报道了一个成功事例，将文本送到 Festival 的 text2wave 组件，然后让 Asterisk 播放转换的结果：一个 wav 文件。实例如下：

exten => 124,1,Answer()

exten => 124,2,System（echo "This is a test of Festival" | /usr/bin/text2wave -scale 1.5 -F 8000 -o /tmp/festival.wav）exten => 124,3,Playback（/tmp/festival）

exten => 124,4,System（rm /tmp/festival.wav） exten => 124,5,Hangup()

这种方法也允许你调用其它的文本朗读工具，比如 Cepstral 中一些流行的朗读工具。例如，我们假定 Cepstral 安装在以下目录：/usr/local/cepstral/

exten => 125,1,Answer()

exten => 125,2,System（/usr/local/cepstral/bin/swift -o /tmp/swift.wav "This is a test of Cepstral"）

exten => 125,3,Playback （ /tmp/swift ） exten => 125,4,System （ rm /tmp/swift.wav） exten => 125,5,Hangup()

Cepstral 可以在相关网站：http://www.cepstral.com 中找到，Cepstral 是一个和 Festival 相关的便宜的语音朗读系统，而且声音相当不错。

10.2 呼叫详细纪录（Call Detail Record，CDR）

甚至没有告诉你，Asterisk 就被假定能够保存你需要的 CDR 信息。这是一台巧妙的机器，不是吗？

缺省情况下，Asterisk 将创建 CSV*文件，这些文件存放在/var/log/asterisk/cdr-csv/目录中。直接看这个文件的话，这个文件看起来有点乱。然而你可以用逗号将每一行区分开来。你将会发现这每一行都包含了呼叫的详细信息，而以下一些值将被逗号分开[*]：

[*] CSV（Comma Separated Values）文件是用来在一个文本文件中，描述类似于数据库类的信息。你可以在文本编辑器里面打开一个 CSV 文件，而且几乎所有的电子表格和数据库程序都能够读取 CSV 文件，而且能够将它们存储到表格的行和列之中。

accountcode

用来如果 SetAccount()应用在使用，或者如果通道配置文件（如：*sip.conf*）以及配置好了。计费码（Account Code）将作为基础信息分配到每一个通道中去。

src

接收到的主叫方 ID（80 字节的一个字符串）

dst

目的 Extension

Dcontext

目的地 context

clid

文本格式的主叫方 ID(长度 80 字符)

channel

使用的通道(长度 80 字符)

dstchannel

可能存在的目的通道(长度 80 字符)

lastapp

可能存在的最后一个应用(长度 80 字符)

lastdata

最后一个应用的数据(长度 80 字符)

start

呼叫开始的时间（日期/时间）

answer

呼叫接通的时间（日期/时间）

end

呼叫结束的时间（日期/时间）

duration

整个呼叫的时间，从呼叫开始到结束，使用秒来计费（数据类型：整数）

billsec

整个呼叫通话的时间，从呼叫接通开始到结束，使用秒来计费（数据类型：整数）

disposition

这个呼叫的情况（接通，没有接通，忙音）

amaflags

确定使用什么样的标志符（文档、计费、忽略），使用在每个基础的通道中，就像 accountcode 一样。AMA 标志符的意思是自动消息计费标示符（Automated Message Accounting flag），它被设想作为可能存在的工业用途而设计。

userfield

用户定义的字段，最长可以有 255 字符。

将 CDR 存储到一个数据库

CDR 也可以保存在数据库里。Asterisk 目前支持 SQLite，PostGreSQL，MySQL 和 unixODBC。本书并不包含关于数据库的配置细节，当然在 Asterisk 源代码的 doc 目录中有相关概述（因为许可证的原因，cdr_mysql 加入到了 Asterisk 附件中）。许多从更愿意把 CDR 保存在数据库中是因为更容易进行查询一些指定信息，比如计费或者长途记录。我们也可以用 CDR 应用程序来对当前拨号方案中的 CDR 熟练的进行处理（例如添加信息到一个自定义字段）。

CDR 方面的问题

虽然 Asterisk 能够恰当地保存通过他的一切呼叫信息，然而他不能保存未被给定的信息。例如，如果你有 SIP 设备允许 reinvite 消息的话，那么一旦 Asterisk 完成了呼叫的建立，设备将不再需要 Asterisk 的协助。无论怎么样，这些设备都不会如何报告其呼叫信息到 Asterisk，Asterisk 都不能控制这些设备。如果 CDR 很重要的话，请确认你的 IP 设备不允许 reivite 消息[*]。

[*] Reinvites 可以在 sip.conf 文件中配置 canreinvite=no 来关闭。类似的过程，你也可以在 iax.conf 文件中配置 notransfer=yes。

10.3 定制系统提示音

为了保持 Asterisk 的表面上的灵活性。你可能要编辑系统提示音。这是很容易理解的，但是做起来通常是很困难的。

大约有超过 300 种的系统的提示音已经包括在已经发布的主要版本中，而且还附加了超过 600 种的 Asterisk 声音，如果你想定制以上所有的话，最好有足够

的时间和金钱。

建议最好有一名音频工程师，确保记录是规范的-3dB，并且声音记录的开始和结束是在零点上（经过一段合适的静音后，声音再开始播放）。

语音

如果你对 Asterisk 语音感兴趣的话，她是 Allison Smith，她可以为你的系统定制你喜欢的声音。

这是一个非常不错的概念，因为只有很小的一部分 PBX 允许你在系统提示音方面使用相同的声音

要想使用 Allison 的专业服务，请登录：http://thevoice.deigum.com

一旦有了录音，实际的实现非常容易——用你创建的文件替换/var/lib/asterisk/sounds 下的相同文件就可以了。

作为其他选择，你可以用自己的提示音放在在你选择的文件夹下。当你在 Playback()和 Background()应用被调用的时候，你可以通过指定该文件的全部路径、或者使用/var/lib/asterisk/sounds/下的子路径，来播放你想要的文件。

将 WAV 文件转换为 GSM 格式的有用的方法是使用 sox 应用程序。使用 sox 转换你的文件，示例如下：

```
#   sox foo.wav -r 8000 foo.gsm resample –ql
```

如果你的 WAV 文件是立体声录制的话，请添加—c1 参数将文件按单声道存储。这些录音经常通过 PC 进行，但是请同时进行检查其他的地方，有人总是喜欢从拨号方案中录制。

10.4 管理

Asterisk 管理单元提供了一组 API，这些 API 允许外部程序对 Asterisk 进行创建、监控和管理工作[*]，管理端口是一组功能强大的多种的 Asterisk 外部扩展程序

[*] API（Application Program Interface）是一种机制，它允许其他程序控制这个程序。和 AGI 相类似的是，AGI 允许的是外部程序能够在拨号方案中被调用。

要想使用 Manager，你必须在以下目录中进行定义一个账号：/etc/asterisk/manager.conf，如下：

[general]

enabled = yes

从 Dailplan 录制声音

令人惊奇的是，得到高质量的好的录音并不是通过一台 PC 和一个充满想象力的软件，而是通过电话设备。这有多种原因。其中最重要的原因是电话能够过滤掉背景噪音（比如 HVAC 设备产生的白噪音）并能记录到一致的音频音量。

这一小段附加的文字能够使你的拨号方案支持你很容易的创建录音，录音将创建在系统的/tmp/文件夹（在这儿你可以重命名或者将其转移到别的地方）：

exten => _66XX,1,Wait（2）

exten => _66XX,2,Record（/tmp/prompt${EXTEN:2}:wav）

exten => _66XX,3,Wait（1）

exten => _66XX,4,Playback（/tmp/prompt${EXTEN:2}）

exten => _66XX,5,Wait（2）

exten => _66XX,6,Hangup()

以上一小段允许你从 6600 到 6699 拨号。并且录制语音提示，并且使用 prompt00.wav 到 prompt99.wav 作为文件名存入/tmp/文件夹。一旦你完成录制过程（通过按#键），它将会放录音，并挂断电话。

Be sure to move your prompts out of the /tmp/ dir to the Asterisk sounds directory. To keep the dialplan readable, rename your promptXX files to a meaningful names—e.g., mv /tmp/prompt00.wav /var/lib/asterisk/sounds/custom/welcome-message.wav.

你要确认已经把你的录音从/tmp/文件移动到你的 Asterisk 声音目录。为了保证拨号方案能够读取它们，重命名 promptXX 文件，并起一个有意义的名字，比如：

mv /tmp/prompt00.wav /var/lib/asterisk/sounds/custom/welcome-message.wav.

```
port = 5038
bindaddr = 0.0.0.0

[oreilly]
secret = notvery
;deny=0.0.0.0/0.0.0.0
;permit=209.16.236.73/255.255.255.0
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

在[general]一段，你可能通过设置参数 enabled = yes 来激活服务。缺省的 TCP 端口是 5038。

对于每一个用户，你可以方括号中指定用户名，下面是设置用户密码和不允许访问的 IP 地址，以及你允许访问的 IP 地址和用户读写权限的设定。

管理命令

记住管理界面是设计用来被程序使用，而不是手指来使用，是至关重要的。这并不是说你不能直接发出命令，而是不要希望有一个特殊的控制台，因为这不是管理者的目的。

管理命令是以以下语法，加载到这些包中（以 CRLF 结束的行）：

```
Action: <action type>
<Key 1>: <Value 1>
<Key 2>: <Value 2>
etc ...
<Variable>: <Value>
<Variable>: <Value>
etc...
```

例如，为了验证管理者（如果你希望无论怎么样都能有一个交互的话）你应

该发送以下内容：

Action: login Username: oreilly Secret: notvery

<CRLF>

空白行的额外 CRLF 将把整个包发送给管理者。

一旦认证通过，你就可以开始操作，以及看到 Asterisk 生成的事件。在一个繁忙的系统中，这看起来就有点复杂了，而且完全不可能通过肉眼来跟踪这些事件。

FOP

很久以来，FOP（The Flash Operator Panel）是一个流行的、强大的管理界面。FOP 创建了一个可视的界面用来管理你的系统，并允许你控制呼叫。

FFOP 像服务员一样，经常被用来观看系统上的用户情况、以及用户之间的呼叫。它也可以被用在呼叫中心环境中，提供 CRM 触发的屏幕显示内容[*]。

[*] 客户关系管理(CRM)是一个界面，公司用来管理客户信息以及进行客户互动工作。

FOP 管理端口是如图 10-1。如果想得到一个 FOP 的拷贝，请登录 http://www.asternic.orgs。

10.5 呼叫文件

呼叫文件允许你通过 Linux 外壳来创建呼叫。通过路径/var/spool/asterisk/outgoing/下产生的.call 文件，而触发这些强大的事件。文件的实际名称也许并不重要，但是给文件起一个有意义的名字并以.call 结尾，是一个好习惯。

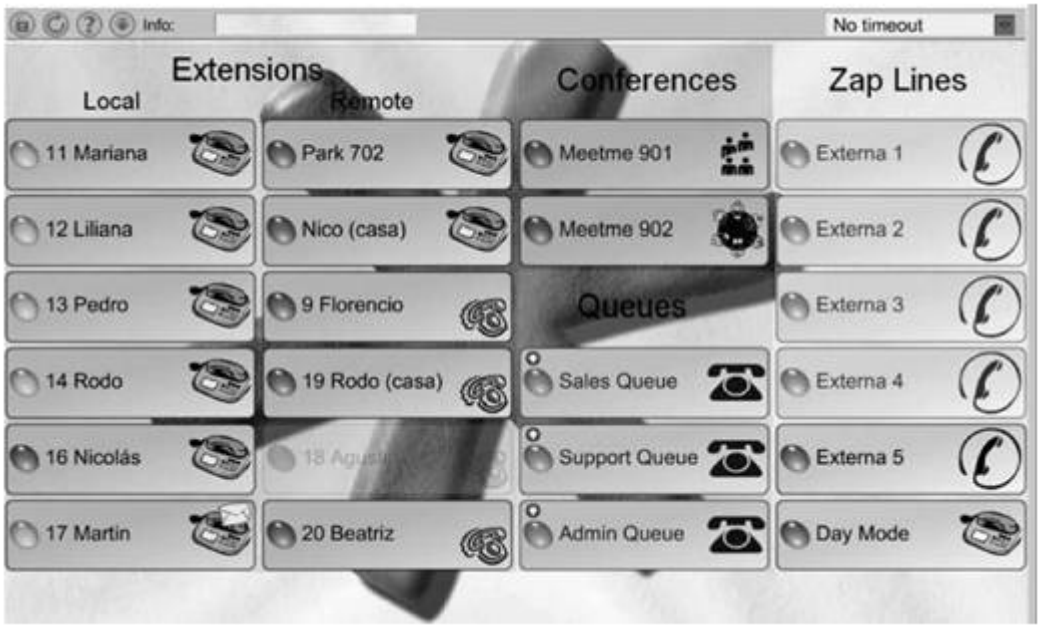


图 10-1　FOP 管理界面

当一个呼叫文件出现在输出文件夹中，Asterisk 将立刻[*]执行包含该文件的指令。

[*] 我们将很快讨论这个。

呼叫文件将被处理成以下格式，首先，我们定义我们的呼叫：

我们可以控制对一个呼叫的应答时长（缺省为 45 秒），呼叫重试之间的等待时长和最大的呼叫重试次数。如果省略最大的呼叫重试次数（MaxRetries），则呼叫只重试一次：

WaitTime: <number>

RetryTime: <number>

MaxRetries: <number>

如果呼叫被应答，我们来指定被呼叫被连接到哪里。

Context: <context-name>

Extension: <ext>

Priority: <priority>

做为选择，我们也可以指定一个单一的应用而且省略变量。

Application: Playback( )

Data: hello-world

下面，我们设置呼出者的主叫方 ID。

CallerID: Asterisk <800-555-1212>

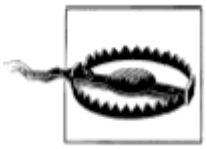
下面设置通道变量，如下：

SetVar: john=Zap/1/5551212

SetVar: sally=SIP/1000

最后加入 CDR 的计费码。

Account: documentation



当你创建一个呼叫文件的时候，不要在 spool 目录下编辑。Asterisk 不管你是不是编辑完了这个文件，就会主动的把这个文件拿来运行。正确的方法是，在其他的文件夹里完成呼叫文件，然后把文件放到 spool 目录下。

10.6 DUNDi

如果对于 Mark Spencer 用完了好主意，而陷入威胁的利害关系而言，分布式统一号码发现（Distributed Universal Number Discovery ，DUNDi）应当使这些想法平息下去。DUNDi 保持着和 Asterisk 一样，是革命性的软件的姿态。DUNDi 网站（http://www.dundi.com）宣称它是最好的：DUNDi 是一个从互联网到电话服务的点到点系统，不同于传统的集中式服务（比如传统的简单的 ENUM[*]标准）DUNDi 完全是一个没有集中认证的分布式系统。

[*] 请参阅 http://www.faqs.org/rfc/rfc2916.txt.

DUNDi 如何工作？

DUNDi 被认为是一个巨大的电话号码簿，而且允许你询问可能知道 VoIP 路由的 peer，以到达一个 extension 号码或者 PSTN 电话号码。

例如；假如你要连接到 DUNDi-test 网络（一个免费并且开放的网络，其终端可以呼叫到传统的 PSTN 号码）。你要呼叫一个你不能直接访问的号码：1-800-555-1212，你要先呼叫你的朋友 Bob 如果他知道如何接到达这个号码，Bob

回答说：我不知道如何接到这个号码，但是我可以问一下我的 peerSally。

Bob 问 Sally，如果 Sally 知道如何达到这个被呼叫的号码，Sally 就应答：你可以通过 IAX2/dundi.very_long_password@hostname/extension 来到达这个号码，Bob 于是把这些信息保存到数据库，然后传递这些信息给你，作为一个可选得方法，告诉你你如何通过 VoIP，访问号码 1-800-555-1212。

由于 Bob 保存了他发现的信息，他将能够为任何有相同号码需求的点提供应答。所以，这个查找过程就不会转发到更远的地方，这将减少网络负载但同时增加了经常被查询的号码的响应时间。（无论如何，这是 DUNDi 被创建的关键，在一个有限的时间周期存贮这些信息是很有效的）。DUNDi 通过在扩展的.conf 文件中执行 switch=>语句或者调用 DUNDi Lookup()应用来执行动态查找。而 DUNDi 仅仅在 Asterisk1.2 版本或者更高版本中可用。

你同样也可以在私有网络中使用 DUNDi。如果你是一个很大的公司的 Asterisk 管理员的话你希望安装并且十分容易的管理 extension 号码。这种情况下你可以使用 DUNDi，允许使用多种 Asterisk 盒子（也许定位于公司的每一个地理位置以及对等联结的另一个地方），在网络中，执行动态查找 extension 的 VoIP 地址。

在 Asterisk 中配置 Asterisk

为了运行 DUNDi，我们需要配置三个文件：dundi.conf、extensions.conf 和 iax.conf。dundi.conf 文件控制着我们允许能在系统中进行执行查找的 peer 的认证信息。该文件也管理着我们能递交的查询请求到 peer 的列表。既然能够在同一个 BOX 中可以运行不同的几个网络，那么我们就有必要为每一个 peer 定义不同的段，然后在每一个 peer 中配置这些网络以允许执行查找。另外，需要定义我们希望用哪个 peer 来执行查找。

1 通用的 peer 协议

通用的 peer 协议（General Peering Agreement，GPA），是为了防止滥用 DUNDi 协议，而和法律捆绑的一个授权协议。在连接到 DUNDi 测试组之前，你需要以

一个 GPA 来认证。GPA 是用来保护群组中的成员，并且在成员中创建一个可信任的群组。你的完整的、正确的联系信息将被要求配置到 dundi.conf 文件中，这样你的 peer 组中的成员，将能够联系到你。GPA 可以在你的 Asterisk 源代码中的 doc 目录以及子目录中找到。

2 通用配置

dundi.conf 文件的[general]部分包含 DUDNi 客户端和服务器端的，总体的操作参数：

```
; DUNDi configuration file
;
[general]
;
department=IT
organization= toronto.example.com
locality=Toronto
stateprov=ON
country=CA
email=support@toronto.example.com
phone=+19055551212
;
; Specify bind address and port number.    Default is 4520
;bindaddr=0.0.0.0
port=4520
entityid=FF:FF:FF:FF:FF:FF
ttl=32
autokill=yes
;secretpath=dundi
```

由 entityid 定义的实体标识符（Entity ID），通常用机器的 MAC 地址来定义。实体标识符缺省用服务器的以太网地址，当然你可以忽略 实体标识符，或者设置的和你的 MAC 地址相似。建议使用第一个对外的接口 MAC。这样其它的 peer 可以用这个 MAC 来确认你。

TTL 字段定义了允许多少个 peer 能够收到应答后，环路断开的数量。由于应答号码未知，每一次需要将号码完全传递。每传递一个号码 TTL 值减少一位。像 ICMP 包中的 TTL 字段，TTL 字段也定义一个我们希望等待用户应答的最大秒数值。

当需要查找一个号码时，一个初始的查询（又被称为 DPDISCOVER）伴随这个号码将被发送到 peer 的请求。如果在 2000ms（信息传送的最大时间）的时间内没有收到应答信号（ACK），而且 autokill 等于 Yes 的话，Asterisk 将发送 CANCEL 到 peer。（注意应答并不是必须对查询发送响应，这只是一个 peer 收到的请求信息的确认而已）。Autokill 的目的就是停止高度“冬眠”的机器。作为 Yes 或者 No 的可选项，你可以自定义需要等待多少毫秒。

Pbx_dundi 模块创建一个密钥并且保存在本地 Asterisk 数据库（AstDB）中。密钥的名字是加密的并且保存在 dundi 群组中。密钥值可以通过 Asterisk 控制台口用 show 命令在数据库中查看。数据库群组可以用 secretpath 选项进行隐藏。

3 创建映射上下文

Dundi.conf 文件定义了 DUNDi 的上下文，这些上下文映射到了拨号方案的 extensions.conf 文件中。DUNDi 的上下文定义了一种明确区分目录服务群的方法。到 DUNDi 的上下文映射段指向 extensions.conf 文件（此文件控制你发布的电话号码）。当创建一个 peer 时，你需要定义你允许这个 peer 来查询的那个映射的上下文。这些工作在 permit 声明中使用（每一个 peer 可能包含多种 permit 声明）。映射上下文是和拨号方案上下文相关，在感觉上，他们是你的 peer 的安全边界。

电话号码必须以如下格式对外发布（E164 格式——译者注）：

<country_code><area_code><prefix><number>

例如一个北美号码应该如下进行发布：14165551212。

所有的 DUNDi 映射上下文以如下格式：

dundi_context =>　local_context,weight,technology,destination[,options]]

以下的配置创建了一个 DUNDi 映射上下文，用来把你的本地号码发布 DUNDi 测试群组中。注意以下应该写在一行中：

dundi-test=>dundi-local,0,IAX2,dundi:${SECRET}@toronto.example.com/${NUMBER}, nounsolicited,nocomunsolicit,nopartial

以上例子中，映射上下文是 dundi-test，映射上下文在 extensions.conf 文件中被指向本地 dundi 上下文（只要提供一个号码表）。直接连接到 PBX 的号码将被宣告为权值为 0（直接连接）。大于 0 的权值表示到达目的的跳数的递增量。当多个查询结果到来的时候，这就很有用——一个小的权值的号码路由，就是最合适的路径。

如果我们应答查找，我们的响应应该包含其它的在这个系统上的终端的连接方法。这包含了用到的技术（比如：IAX2，SIP，H323 等等），认证用的用户名和密码，认证发送到的主机，以及 extension 号码。

Asterisk 也提供了一种简便的方法，允许我们通过模板来创建我们的应答。以下通道变量就可以用来创建模板：

${SECRET}

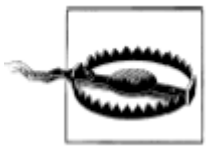
取代存在本地 AstDB 里的密码

${NUMBER}

请求的号码

${IPADDR}

要连接的 IP 地址


静态配置主机名要比起使用${IPADDR}变量，要安全很多。${IPADDR}变量有时候会返回一个私有 IP 地址空间，这在 Internet 上不可用。

4 定义 DUNDi peer

DUNDi peer 是在 dundi.conf 文件中被定义的。peer 是由远程系统的第二层 MAC 地址来唯一确定。dundi.conf 文件是：我们用来定义上下文来为 peer 查询，以及我们用来查询整个网络的 peer 的文件。

[00:00:00:00:00:00] ; Remote Office model = symmetric

host = montreal.example.com inkey = montreal

outkey = toronto include = dundi-test permit = dundi-test qualify = yes dynamic=yes

远程 peer 标识（MAC 地址）是括在方括号中。Inkey 和 outkey 是我们用来认证的公有/私有密钥对。密钥对是通过保存在/asterisk/contrib/scripts/目录中的 astgenkey 脚本来产生的。确定使用-n 参数，这样你才能在每一次启动 Asterisk 的时候，不用初始化密码。实例如下：

cd /var/lib/asterisk/keys

/usr/src/asterisk/contrib/scripts/astgenkey –n toronto

结果键 toronto.pub 和 tornoto.key 将被放置到你的/var/lib/asterisk/keys/目录中。tornoto.pub 是公用键，而且你应该将这个键公布到你的网页中以让其它相和你通话的人能够容易的通过认证。当你和对方通话时，你可以得到对方的 HTTP 访问公用键，它们将被放置到你的/var/lib/asterisk/keys/目录中。

当下载键之后，你应该重新在 Asterisk 中加载 res-crypto.so 和 pbx_dundi.so 两个模块：

*CLI> reload res_crypto.so

-- Reloading module 'res_crypto.so' （Cryptographic Digital Signatures）

-- Loaded PRIVATE key 'toronto'

-- Loaded PUBLIC key 'toronto'

*CLI> reload pbx_dundi.so

-- Reloading module 'pbx_dundi.so' （Distributed Universal Number Discovery

（DUNDi））

== Parsing '/etc/asterisk/dundi.conf': Found

然后，在 iax.conf 文件重创建 dundi 用户以允许连接你的 Asterisk 系统。当一个呼叫被认证后，这个呼叫就可以在 Asterisk 中开始的地方，其 extension 号码将被请求通过 dundi 并保存在 extensions.conf 文件的本地上下文中。

5 允许远程连接

下面是用户定义的 dundi 用户。

```
[dundi]
type=user
dbsecret=dundi/secret
context=dundi-local
disallow=all
allow=ulaw
allow=g726
```

代替使用静态密码的方式，Asterisk 每经过 3600 秒（1 小时）就会重新生成一个新的密码。密码将保存在 Astersik 数据库/dundi/secret 目录中并且通过使用${secret}变量来保存在 dundi.conf 文件的映射 context 中进行了定义。通过在 Asterisk CLI 命令行中用 show keys 命令，你可以为所有的呼叫查看当前的密码，包含所有的本地公有密码和私有密码。

上下文实体的本地 Dundi 是通过在 extensions.conf 文件中发送认证信息来实现的。在这儿，我们可以对呼叫进行操作，就好比我们可以在拨号方案中对呼入的连接的操作的类似。

6 拨号配置

extensions.conf 文件包含了你想公布的号码和你希望谁能够进行呼叫，dundi-loca 上下文执行两种操作。

- 他控制我们公布的号码，并且通过在 dundi.conf 映射 context 进行引用。
- 通过在 iax.conf 文件中被 dundi 用户引用来处理呼叫。

可以通过强大的拨号方案的模式匹配，来发布号码范围并控制呼入的呼叫。在下面的拨号方案中，我们仅仅发布号码：+1-416-555-1212，但模式匹配能够容易的被用来发布号码范围或者扩展的：

[dundi-local]

exten => 14165551212,1,NoOp（dundi-local: Number advertisement and incoming）

exten => 14165551212,n,Answer()

exten => 14165551212,n（call）,Dial（SIP/1000）

exten => 14165551212,n,Voicemail（u1000）

exten => 14165551212,n,Hangup()

exten => 14165551212,n（call）+101,Voicemail（b1000）

exten => 14165551212,n,Hangup()

10.7 结束语

在本章中有大量的知识可学，但是这些东西在任何地方都可以得到，幸运的是，你可以以一个好的方式开始学习 Asterisk。在下一章，我们将试着预言未来的电信系统。而且我们相信在未来 Asterisk 将有一个很好的发展和前景。

第11章　Asterisk：未来电话技术

起初，他们对你视而不见，然后他们开始嘲笑你，接着，他们开始和你战斗，最后，你赢得了胜利！

——圣雄甘地

我们来到了本书的最后一章。我们已经讲了很多内容，但是希望你现在能认识到，我们才刚刚涉及到 Asterisk 这种现象的表层。作为收尾，我们将用更多的时间来探寻：在不久的将来我们会从 Asterisk 和公开源码的电话中看到什么。

做预测常常是个不讨好的差使，但是我们有信心断言，像 Asterisk 这样公开源码的通信工具预示着思想上的一个转变，这个转变将使电信业为之一新。在本章，我们将探讨产生这种信念的原因。

11.1　传统电信业的问题

虽然亚历山大.格拉汉姆.贝尔作为电话之父而闻名于世，但是事实上自从 19 世纪下半叶以来，已有数十种理念被应用于在电报线路上传输语音的项目之中。这些发明者大多数都是基于生意上的考虑，希望创造出一种新产品而发财致富。

我们现在总是把传统的电话公司看作垄断者，但在他们在初创时期并非如此。早期的电话服务业处在一个激烈竞争的环境之中，全世界范围内新的公司不断出现，而且常常对他们可能会侵犯的专利权不屑一顾。某些垄断者就是通过专利战中的扯皮（或胜利）取得了在竞争中领先。

把电话的历史与 Linux 和 Internet 的历史相比较，会发现一些非常有趣的现象。电话作为一个商业性的试验被发明出来，电信业在诉讼和公司收购中稳步发展，Linux 和 Internet 则是在学术团体中产生，这些团体通常认为知识共享高于经济利益。

相比之下，两者在文化上的区别显而易见。电信技术往往是封闭、混乱而昂贵的，而网络技术则广泛开放，资料翔实，富于竞争性。

封闭的思想

如果你对比一下电信业和 INTERNT 的文化，也许有时会很难相信两者有着一定的联系。Internet 是由一群富于激情的爱好者发明的，然而对 PSTN 有所贡献却不是任何个人所能企及的。这是一个具有排他性的俱乐部，不是对任何人都开放的。

国际电信联盟（ITU）清楚的显示了这种封闭的思想。如果你想得到 ITU 的技术资料，你就必须为此付费。成为 ITU 的会员需要通过资格认证，并要支付数万美元的年费。*

虽然 ITU 是联合国批准成立的负责国际电信业的组织，然而多种 VOIP 协议（SIP, MGCP, RTP, STUN）并不是产生于 ITU 的神殿，而是来自 IETF（该组织免费发布所有的技术标准，并允许任何人提交 Internet 草案以供参考）。

像 SIP 这样的开放性协议相比于 H.323 等 ITU 协议具有战略性的优势，因为开放性协议更加容易被获得。虽然 H.323 已经被运营商作为 VOIP 协议在骨干网中广泛的应用了，但是很难找到基于 H.323 的终端产品，最新的产品往往更倾向于支持 SIP。

面对 IETF 开放性道路的成功，势力强大的 ITU 并没有无动于衷。最近，ITU 可能在其网站上提供最多三个文档的免费下载。*开放性已经清晰的出现在他们的思路当中。ITU 刚刚发布的声明显示 ITU 希望全社会以及学术界更多的参与进来。ITU 的电信标准局主任赵厚林先生相信：“ITU 会逐布采取措施对此进行鼓励”。[*]

[*]与此相比，IETF 的成员名册上写道：“IETF 并不是一个会员型的组织（没有组织成员卡，没有应得权利，也没有秘密的交换 :-)…它对任何有兴趣的个人开放…欢迎加入 IETF。”这就是一个团体应有的言论。

考虑到可以获得数以千计的文件，而且每个文件通常都包含十几本参考书目，这个免费信息的价值是难以估量的。

网站地址为：

http://www.itu.int/ITU-T/tsb-director/itut-wsis/files/wg-wsis-Zhao-rev1.pdf

达到这种开放的路线图尚不清楚，但是 ITU 已经开始认识开放是不可避免的。

对于 Asterisk，它既包含过去也面向未来——支持 H.323，虽然 Asterisk 组织尽量避开 H.323，而更倾向于 IETF 的 SIP 协议，以及自己最喜爱的 IAX。

有限的标准执行度

对于落伍的电信界，其所有现存标准中最奇怪的事情之一是，各个生产商在表面上都无法始终如一的执行这些标准。每个生产商都希望完全的垄断，因此他们都不愿在互通性上有所考虑，而是急于把有创意的想法第一个推向市场。

ISDN 协议是这方面最典型的例子。ISDN 的运用是（在很多其他方面更加是）一个令人痛苦而且代价昂贵的命题，每个生产厂商都决定以略微不同的方式去执行 ISDN 标准。ISDN 本可以早于 Internet 十年，带来一个庞大的公用信息网的产生。不幸的是，由于它成本巨大，结构复杂以及兼容性方面的问题，ISDN 从来没有把语音传输与视频或者数据相结合，提供给愿意付费的用户。ISDN 应用的非常广泛（尤其是在欧洲和东亚的大型交换机中）；然而它展现出的能力与人们预想的相差甚远。

随着 VOIP 的日益普及，对 ISDN 的需求终将消失。

缓慢的产品发布周期

可能需要几个月，有时要几年，那些高层人士才可能承认这个趋势，对符合该趋势的产品发布不再做限制。看来，在一项新技术被采用之前，它一定会先被分析得一无是处，然后成功的经过官僚机构的层层审核，最终才被排进研发周期之内。任何一个有用的产品的产生都要经过几个月甚至数年。当这些产品最终发布时，它们通常基于陈旧的硬件结构，而且往往价格昂贵且毫无特点。

缓慢的产品发布周期在当今的商业通信世界中显然无法生存。在 Internet 上，一个创意几周内就能产生，在极短的时间内就能具备可行性。既然其他的技术都必须适应这些变化，电信业更应当如此。

公开原码的开发，与生俱来的更能适应快速的技术变化，这就给了它巨大的竞争优势。

这种对于电信业的巨大冲击，很大程度上源于电信业缺乏应变能力。也许正是由于一直缺乏应变能力，电信业在收到冲击之后才会恢复得如此缓慢。现在已

经别无选择：变化，否则将死亡。团体驱动的技术，比如 Asterisk 充分注意到了这一点。

拒绝抛弃过去拥抱未来

传统的电信公司已经失去了和客户之间的紧密联系。很容易想到，应当在基本的电话服务之外，向客户提供其他功能。然而这些功能应该由用户来定义，这种想法却并不是那么容易理解的。

如今，在其他每种通信方式上，人们都几乎拥有了无限的灵活性。很难让人理解的是，电话业始终无法像它许多年前承诺的那样，具备类似的灵活性。灵活性并不是电信业所熟悉的概念，也许直到像 Asterisk 这样的公开源码产品开始改变电信业的基本特性的时候，这种情况才会有所改变。这是一场变革，类似于十多年前 Linux 和 Internet 的自发性革命（以及 IBM 在十五年以前无意中引发的个人电脑革命）。这场革命究竟是什么？是电话软件和硬件的大范围普及，从而使定制化的电信系统呈几何级数的增长。

11.2　范式的改变

提姆．奥莱里在他的文章《范式的改变》(http://tim.oreilly.com/articles/paradigmshift_0504. html) 中写到：在技术（包括硬件和软件）的发展道路上，范式正在出现变化。*奥莱里发现了三个趋势：软件的大范围普及，网络赋予的协作能力，软件的客户定制化（软件作为一种服务）。这些趋势证明公开源码电话的时代已经来临。

11.3　公开源码电话系统的承诺

每一个好的软件作品都源自于开发者的强烈渴望。

—埃里克 S. 雷蒙德 《教堂和杂货店》

埃里克 S. 雷蒙在他的著作《教堂和杂货店（O’Reilly）》中解释到：“如果给予足够的关注，所有的软件问题都可以轻易解决。”

*很多接下来的章节仅是奥莱里文章的翻译。如果想充分了解这些观点的要

旨，推荐仔细阅读奥莱里的文章。

Asterisk 所能解的燃眉之急

在用户定制数据库和网站飞速发展的年代，人们已经厌倦听到他们的电话系统“不能做那个”，用户们非常坦白的对此表示怀疑。客户的创造性需求，加上技术上的局限性，导致了一种必须的创造力的产生：电信工程师就像《废品大战》某一集中的竞争者们一样，正在试图由一堆不匹配的零件做出一个实用的装置。

发展方法论要求专利性的电话系统具备大量的特性，而且这些特性的数量在很大程度上决定了系统的价格。生产厂商会告诉你他们的产品将带给你数以百计的特性，但是如果你只需要其中的五个，没人会在意。更糟的是如果你不可或缺的一项功能刚好没有被提供，这套系统的价值就会因为无法完全满足你的需求而降低。

客户只需要五百种特性中的五种这一事实被视而不见，客户希望拥有五种目前未提供的功能来满足自己的需求，也被毫无道理的忽略。*灵活性已经成为标准，电信业还在墨守上世纪的陈规——虽然 VOIP 已经在全世界普及开来。

Asterisk 可以直接解决这个问题，其解决方法很少有其他电信系统可以做到。Asterisk 是一项极具分裂性的技术，这很大程度上是由于 Asterisk 基于一个被反复证明的思想：“即封闭源码无法在技术进化的竞争中取得对开发源码组织的胜利，因为后者可以在一个问题上投入多得多的更专业的时间。” *

开放性结构

传统电信业的一大绊脚石在于，各个厂商显然拒绝与同行业中的其他公司合作。电信业的巨头已经存在了上百年。封闭性私有系统的观念在他们的文化中是如此根深蒂固，以至于即使他们想让产品具备标准兼容性，也会出于在竞争中领先的目的，而加上一项其他厂商都无法支持的功能。只要简单看看目前电信业所提供的 VOIP 产品，就可以找到很多体现这种思路的例子。虽然这些产品号称具有标准兼容性，但是如果你希望 CISCO 的电话和 Nortel 的交换机能够互通，或者想把一个 Avaya 的语音信箱系统通过 IP 和西门子的交换机整合起来，那么这种想法是经不起推敲的。

[*]从封闭源码行业的前景可以看出，他们的这种态度是可以理解的。弗雷德.布鲁克斯在他的著作《人月神话：软体专案管理之道（阿狄森.韦斯利）》中写到：“一个项目的复杂性和沟通费用与开发人员数量的平方成正比，而完成的工作量则仅仅与其成线性正比”。因此，如果不采用“组织协作”的开发方法，新产品上市是非常困难的，最好的结果也不过是前一代的产品的增量改善，差一点的结果就是一堆补丁的集合。

[*]埃里克 S. 雷蒙德 《教堂和杂货店》

电脑业的情况就大不一样。二十年前，如果你买了一个 IBM 服务器，你就需要有 IBM 的网络和终端和服务器配合。现在，IBM 的服务器可以通过 CISCO 的网络（在 Linux 上运行任何东西）和戴尔的终端。在这种背景下，任何人都可以想出数以千计的变化。如果这些公司建议我们只使用他们告诉我们的产品，那么他们将在嘲笑声中退出市场。

电信正在面对同样的变化，但不急于接受这些变化。Asterisk 则相反，不但非常乐于接受变化，而且对变化给予热情的拥抱。

Cisco，Nortel，Avaya 以及 Polycom 的 IP 话机（仅仅有些公司是这样对该产品命名的）全部可以和 Asterisk 系统成功对接。世界上目前没有任何其他的交换机可以做到这一点。没有。

开放性是 Asterisk 强大的力量。

标准兼容

在过去的几年，我们可以清晰的发现，标准正在飞速发展，要跟上标准的发展就需要对出现的技术趋势能够做出快速回应。Asterisk 因为基于公开源码，在全社会的共同努力下发展，因此具备独一无二的适应飞速发展的标准的能力，而这种能力正是标准兼容所需要的

Asterisk 没有专注于成本收益分析或者市场调查。不论在这个领域中有任何令人兴奋的或必须的发现，Asterisk 都会做出回应并向前发展。

对新技术闪电般的快速响应

马克.斯班瑟在进行了第一次 SIP 互用性实验几天之后，就得到了初步可用

的 Asterisk 编码的 SIP 栈。

这种远见和适应性正是公开源码开发组织的特点（在大公司中就非常罕见）。

充满热情的团体

Asterisk 使用者名录每天都收到几百个 email 信息。超过 10 万人对 Asterisk 进行捐助。这种社会性的支持在私有性的电信行业中几乎是闻所未闻，而在公开源码的世界里则是比比皆是。

第一次 AstriCon 活动本来希望吸引一百个参与者，但有近五百人参加（更多的人想参加而没有实现）。社会性的支持实际上确保了公开源码的努力终将获得成功。

有些事现在已成为可能

那么可以开发出哪些东西以使用 Asterisk 呢？让我们看看一些已经面世的产品吧。

1 Legacy 融合了 PBX 的网关

Asterisk 是一座连接旧式的交换机与未来的奇妙的桥梁。可以把 Asterisk 置于交换机的前端作为一个网关（ 并根据需求的规定把用户移植到 PBX 之外 ），或者置于交换机的后端作为一个外设申请服务器。甚至可以同时完成上述两种功能，如图 11－1 所示。

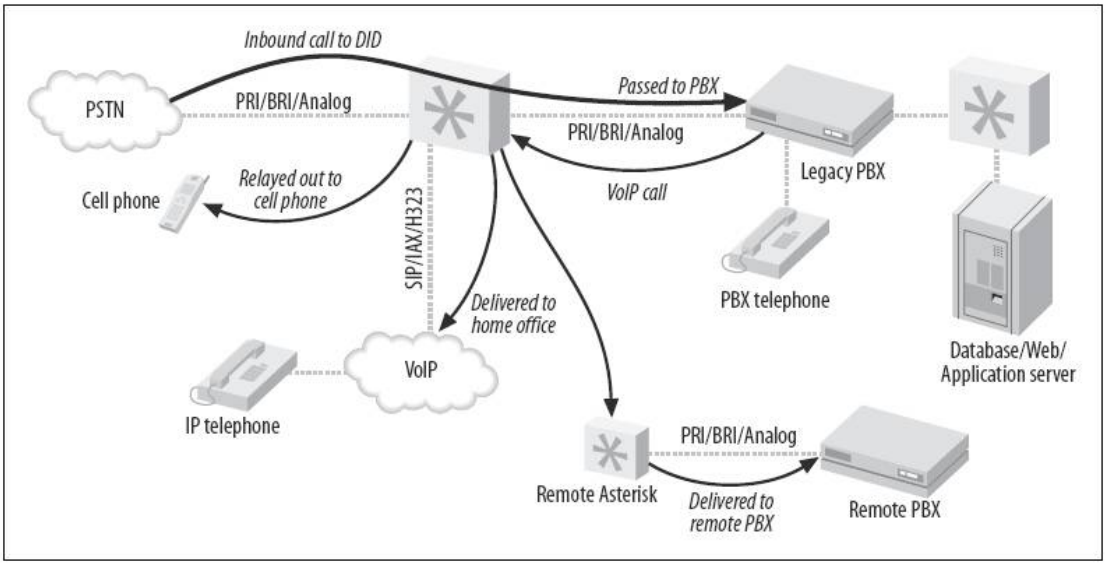


图 11-1　Asterisk 作为一个交换机网关

在此有一些选项可以实现：

保持原有的交换机不变，升级成 IP－PBX

那些在过去几年花高价购买私有交换机设备的公司，希望跳出私有性系统的限制，但是又不能容忍把所有设备弃之不用，因为它们还有其他功能。没有问题——Asterisk 可以解决各类问题，无论是更换一个语音信箱系统，还是提供在系统有名无实的容量之外添加基于 IP 的用户的方法。

“找到我跟随我”

向 PBX 提供一个你能接听的电话号码的列表，这样任何时候如果有电话打到你的 DID（直接拨入号码，即通常所说的电话号码），PBX 都会呼叫列表中所有号码。图 11-2 对这项技术做了说明：

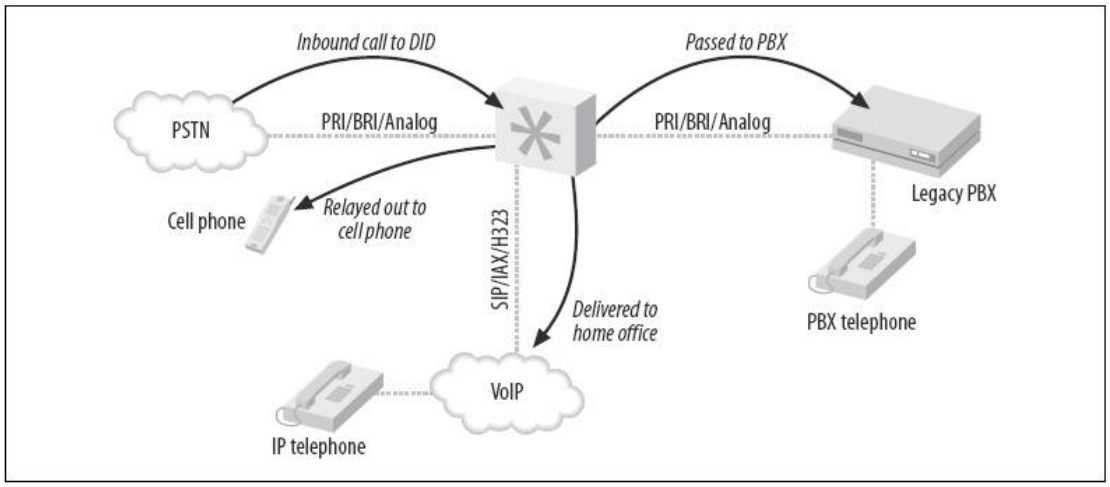


图 11-2　“找到我，跟随我”

VoIP 呼叫

如果能够建立一个从 Asterisk 交换机到旧式交换机的 legacy 电话连接，那么 Asterisk 就可以提供 VoIP 服务接入，而旧式交换机则像往常一样继续与外界相连接。作为一个网关，Asterisk 仅需要模拟 PSTN 的功能，旧式的交换机不会感觉到任何地方发生变化。图 11－3 显示如何利用 Asterisk 使 legacy 交换机具备 VoIP 的功能。

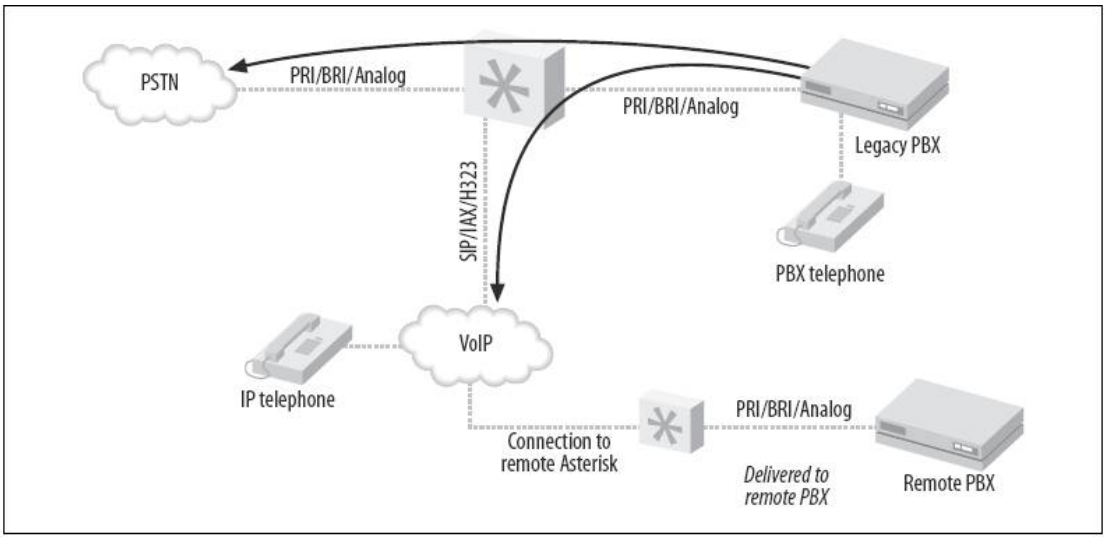


图 11-3　支持 VoIP 的 Legacy PBX

2 低门槛的 IVR

许多人搞不清“交互式语音应答”，或者说“IVR”与自动接线员（AA）的区别。这是可以理解的，因为 AA 是 IVR 最初的应用。不过，对电信业而言，IVR 这个术语所代表的远不止是 AA。AA 所能做的仅仅是把通话转移到分机，并被内置到绝大多数的私有语音信箱系统中——但是 IVR 能做的要广泛得多。

IVR 系统通常价格昂贵，配置服务也要花费不少。传统的 IVR 系统一般需要和内部的数据库或应用软件互通。Asterisk 可被证明是非常好的 IVR，因为它在最底层包含了与数据库和应用软件互通的概念。

以下是一些利用 Asterisk 创建相对简单的 IVR 的例子。

天气预报

通过 Internet，可以有无数种方法得到文本格式的全世界天气预报。拿到这些报告后，把它们在“以目的为导向”的分析器上运行（用 perl 可能就足够了），这样就可以通过某种拨号方式得到这些信息。Asterisk 的声音库里已经包含了所有需要的提示音，因此建立交互式的菜单来播放当前的天气预报，再也不是一项繁重的工作了。

数学程序

Pulver.com 艾德.盖伊在 2004 年 Astricon 做的报告中谈到，他为女儿制作了一个小程序，写这个程序花了他不到一小时的时间。这个程序可以向盖伊的女儿提出一系列数学问题，她可以通过电话按键回答这些问题。当所有问题都回答完毕之后，系统会自动算出得分。*假设方案可以充分实施，那么花一万美元，就可以使最简单的 Asterisk 应用软件在任何封闭的 PBX 平台上运行， 就像我们经常看到的一样——对于 Asterisk 是很简单的事情，对于其他 IVR 系统却非常的困难而且极其昂贵。

分布式 IVR

私有性 IVR 系统的成本很高，如果一个有很多分支机构的公司想提供 IVR，它就不得不把所有的通话都转移到中央服务器来进行处理。通过 Asterisk，则有

可能把这些请求分别送至每个节点，然后在本地处理这些请求。在世界各地的数千个小型的 Asterisk 系统可以用一种其他系统无法实现的方式提供 IVR 功能。不再需要把通话长途转移至 IVR 服务器，也不需要完成这个工作所特需的庞大的中继功能——更多的功能，更少的花费。

更多内容参见第九章。

下面是三个证明 Asterisk 潜力的简单例子。

3 会议室

这个小花苞将会被 Asterisk 的一个迷人的功能终结。在 Asterisk 组织里，每个人都发现自己正在越来越多的使用会议室功能，以实现以下的功能：

- 小公司需要一种更简单的方式和生意伙伴在一起交谈。
- 销售团队每周在会议室要开一次会，每个人都可以在任何地方拨入这个会议室。
- 开发小组指定一个共用的场所和时间，成员互相对程序进行修正。

4 住宅自动控制

Asterisk 是一个非常典型的 über-geek's 工具，可以为普通的家庭提供服务，但是仅需要一般的 Linux 和 Asterisk 技能，就可能实现下面这些应用：

对家里的孩子进行监控

家长如果想检查保姆（或独自在家的孩子）的工作，他可以拨一个密码保护的分机号。一经过认证，将创建对房间里所有的 IP 话机的双向语音连接，让妈妈或者爸爸听到是否有麻烦。有点鬼鬼祟祟的感觉？是的。不过仍不失为一个有趣的想法。

锁定你的电话

要在晚上外出？不想让保姆使用电话？没问题！只要对拨号方式做简单的改变，话机就只能拨打 911，你的手机或者是 PIZZA 外卖的电话了。如果想拨打其

他电话，就会听到自动录音“我们付钱是让你照顾我们的孩子，而不是打私人电话”。“很刺激吧，哈哈”。

控制报警系统

当你度假时可能会接到这样的电话：妈妈想借一些炊具，但找不到钥匙了，现在正站在房子前面发抖。这个问题很好解决，对你的 Asterisk 系统的电话进行一次呼叫，向你专门创建的 context 发送一个快速数据流，就可以使房间的报警系统在 15 分钟内暂时失效。这样妈妈就可以拿到她想要的东西。然后要迅速的离开房间，否则警察就要出面了。

管理十几岁孩子的电话

给你十几岁的孩子分配一个特别的电话时间限制怎么样？想使用电话，他们就得输入自己的接入代码。他们可以通过做家务活得到额外的通话时间，对所有这些进行记分，而不需要再对糟糕的发型唠唠叨叨了——这下你有办法了。一旦他们用完了自己的时间……点击……你可以把电话拿回来了。

打进来的电话也可以管理——通过主叫 ID。“东尼，我是苏丝的父亲。她再也没兴趣见你了，因为她想把自己的水准提高一点。而且，你应该考虑剪剪你的头发。”

11.4 Asterisk 的未来

我们已经爱上了 Internet，因为它的内容丰富价格便宜，也许更重要的是，它让我们能自己定义以何种方式通信。Internet 能带来更丰富的媒体发展的形式，因此我们将看到自己会越来越多的使用 Internet。一旦 Internet 上的语音传输质量赶得上（或者超过）PSTN，电话公司就只好转行了。PSTN 将不复存在，仅仅变成 Internet 带给我们的又一个通信协议。公开源码的技术将领导这次革命，

语音处理

我们的发明可以和我们自己对话，这是一个比电话本身更古老的梦。每一个技术上的进步都刺激着新一轮更热切的实验热潮。但通常结果很少和期望相符，

这可能是因为机器刚刚能说出一些听上去智能的东西，大多数人就假设它确实是有智能的。

为电脑编程或者维护电脑的人能认识到电脑的局限性，因此他们往往承认电脑的缺陷。而其他人仅仅希望其电脑和软件能正常运行。使用者必须和他们的电脑互动，有这种想法的人其数量通常与认为开发团队应该和电脑互动的人的数量成反比。简单的界面掩盖了复杂的设计决定。

因此，真正的挑战是设计一个系统，能够满足用户的共同愿望，并能够熟练的应付无法预期的困难。

1 Festival

Festival 文本语音转换服务器可以把文本转化成语音。虽然其中有很多有趣的东西可以尝试，但还是有很多挑战需要面对。

对 Asterisk 而言，文本语音转换最明显的价值可能是用电话系统为用户读出他的 EMAIL。当然，如果你注意到在这些天的 EMAIL 里有不太地道的语法和读音，还有典型的拼法，你也许会意识到这个功能带来的挑战。

有些人会情不自禁的想知道，文本语音转换的出现是否会鼓励新一代人专注于正确的写作。在屏幕上看到拼写或者标点错误已经够让人尴尬了，如果从电脑上听到这种错误，那就很少有人能拥有足够的修为忍受这种难堪了。

2 斯芬克斯

如果文本语音转换是一项飞速发展的科技，语音识别就会是一个科学神话。

语音识别技术实际上可以做的很好，但是很不幸，这要求你必须提供适当的环境——而适当的环境在电话网络里很难找到。既使是非常好的 PSTN 连接也被认为对精确的语音识别有着极大的限制。如果是压缩和有损的 VOIP 连接，或者是移动电话，你会发现更多的缺陷而不仅仅是使用问题了。

由于 Asterisk 的适应性，它具备成为出类拔萃的语音识别系统的潜力。不幸的是，语音识别本身并不足以满足各种我们想要的用途。当这项技术趋于成熟，公开源码组织将很有可能把语音识别纳入其中，并为它提供灵活而强大的运行平

台。

高保真的音质

随着带宽日益增加，我们可以越来越方便的使用低压缩率的编码。很多人还没有认识到，Skype 运用的是比电话更高的声音保真度；这在很大程度上决定了为什么 Skype 能以其音质而闻名。

如果你曾打电话给 CNN，难道你不希望听到詹姆斯.厄尔.琼斯迷人的嗓音“这是 CNN”吗？而不是冷冰冰的自动录音。如果你认为阿里森.史密斯[*]的声音在电话里听起来很棒，你就有可能听到她本人的声音。

在未来，我们期待能通过我们的通信设备获得高保真的音质。

[*]阿里森.史密斯是 Asterisk 的声音——所有的系统提示都采用她的声音。用 Allison 的声音做你自己的提示音，只需要访问 http://thevoice.digium.com。

视频

在某些方面，Asterisk 已经兼容了视频。实现视频功能与其说是功能问题，不如说是在于带宽的多少和处理能力的强弱了。更有意义的是，对于 Asterisk 组织而言，吸引所需的注意力，已经不是非常重要的了。

1 视频会议的挑战

自从显像管发明后，视频会议的想法就产生了。电信业几十年来一直承诺让视频会议设备出现在每个家庭。

随着很多其他的通讯技术的产生，如果你想在家里进行视频会议，你可以通过一个简单低价的软件电话在 Internet 上实现。然而似乎人们仍然把视频会议看作一个噱头。是的，你会发现，一些和你交谈过的人就有这样的想法，但是有些东西可能被忽略了。

2 为什么我们喜爱视频会议

视频会议提供了比电话更丰富的沟通经历。能看到对方的交谈可能比只闻其

声，不见其人的沟通方式更好。

3　为什么视频会议可能无法完全取代声音的交流

然而，仍有一些挑战需要解决，而且并不是所有的挑战都来自技术方面。

想想看：使用普通电话，人们可以在家庭办公室工作。用电话进行生意上的交流，不需要洗澡，穿着内衣，把脚跷在桌子上，手里端着咖啡。如果用视频做类似的交流，则可能需要半个小时在衣着上做准备，而且不能在厨房和院子里或者其他什么地方进行。现在你明白了吧。

而且只要参与者的焦点与摄像头不符，面对面的视频沟通就无法实现。如果你看着摄像头，对方可能会看到你正看着他们，但是你并不想这样。如果你看着屏幕上正和你交谈的人，摄像头会显示你正往下看着什么——而不是你的听众。这看起来不够人性化。也许视频电话可以设计得像 Tele-Prompt-R 一样把摄像头藏在屏幕之后，那样就不会感觉到如此不自然了。正是这样，视频会议忽略了使用者心理方面的一些问题。所以目前视频看起来还像是一个噱头。

无线

既然 Asterisk 可以充分支持 VoIP，那么无线也会成为其众多功能中的一项。

1　Wi-Fi

Wi-Fi 将成为 VOIP 话机的移动办公解决方案。这项技术已经相当成熟。最大的障碍是终端设备的成本，这个问题可以寄希望于全世界产品价格下降所带来的有效降价压力。

2　Wi-MAX

我们已经大胆的预见了很多东西，那么不难预测 Wi-MAX 的出现可能会是传统手机消失的开端。在无线 Internet 接入到绝大多数小区的情况下，昂贵的手机服务还有什么价值呢？

统一消息

电信业几年前就开始对这个术语做广告宣传了，然而正式使用却比预想的要慢得多。

统一消息试图把语音和文本消息系统结合到一起。通过 Asterisk，这两个系统不需要人为合并，因为 Asterisk 用同样的方式对他们进行处理。

看一下这个术语，“统一”和“消息”，我们可以发现 Email 和语音邮件的融合仅仅是一个开始——统一消息如果想名副其实，还有更多的东西要做。

也许我们需要把“消息”定义成一种非实时的通讯方式。换句话说，当你发送消息的时候，你可以想到回复可能要片刻，几分钟，几小时甚至几天才能到达。你写你想说的东西，并希望信息接受方在写回复。

比较一下会发现，即时通信刚好相反。如果你用电话和别人交谈，你可不希望过几秒中在能听到对方的声音。

提姆.奥莱里进行了一个题为《关注 Alpha 奇客：OS X 和下一个大事件》的演讲 (http://www.macdevcenter.com/pub/a/mac/2002/05/14/oreilly_wwdc_keynote.html)。其中提到最近一些人正在鼓吹通过文本语音转换器实现的在线聊天系统。也可以设想从相反的方向入手，利用 Asterisk PBX 提供语音到文本再到语音的转换，让我们能够通过 WI-FI 电话使用即时通信工具或者进入在线聊天系统。

对等互连（peering）

当垄断性网络如 PSTN 为 Internet 这样的公有性网络提供通路的时候，就需要一段必要的时间使两个网络互通。然而传统运营商更希望现有的 Model 能够适应新的范式，这就越来越使电话有可能变成 Internet 乐于支持的一个应用。

但是挑战仍然存在：如何管理我们已经熟悉而且适应的电话编号方式？

1 E.164

ITU 用 E.164 规范定义编号方式。如果你通过 PSTN 打过电话，你就可以肯定的说你对 E.164 编号方式的概念很熟悉。

既然我们希望呼叫能从 PSTN 传输到 Internet 或其他一些目前无法预见的网络上，那么就必须对 E.164 给予关注。

2 ENUM

为了应对这个挑战，IETF 发起了电话号码映射（ENUM）工作组，成立这个工作组是为了把 E.164 映射到 DNS。

虽然 ENUM 的构想有其合理性，但是要达成目标仍需要电信业的合作。但是合作并不是电信业的特点，所以至今 ENUM 还是步履维艰。

3 e164.org

e164.org 的成员一直在致力于 ENUM 的成功。你可以登录该网站，登记你的电话号码，然后告诉系统另一种与你联系的方法。这意味着某些知道你的电话号码的人可以通过 VOIP 呼叫你，因为 DNS 区域可以提供所需的 IP 寻址和协议信息，把呼叫与你的所在地连接起来。

4 DUNDi

分布式普遍数字发现（DUNDi）是一个公开路由的协议*，设计 DUNDi 是为了在兼容的系统之间维持动态电信路由选择表。虽然 Asterisk 目前只能通过 PBX 支持 DUNDi，但是公开的标准能确保每个人都能对其进行操作。

DUNDi 拥有巨大的潜力，但它现在才刚刚起步。DUNDi 值得我们的关注。

*更多内容参见前几章。

挑战

Asterisk 毫无疑问含有巨大的价值，但它将面临挑战。让我们大致了解一下，可能会出现哪些挑战。

1 太多的挑战，太少的标准

最近一段时间，Internet 飞速的变化，提供的内容如此丰富，以至于对于绝大多数的技术高手，想充分掌握这些内容几乎不太可能。虽然 Internet 的发展就是这样，但是它还意味着，如果想让任何一个通信系统不落伍，那么开发出数量众多的新技术，就将不可避免的成为其工作的很大一部分内容。

2 VoIP 垃圾信息

是的，它即将来到。经常会有人认为他们有权利为了钱而麻烦其他人并让其他人感到不快。VoIP 垃圾信息已经处在尝试和实现的起步阶段了，但是时间会证明这些垃圾信息是否有效。

3 恐惧，不确定和怀疑

电话业正在开始从不屑一顾到嘲笑的转变。甘地是对的，我们可以预见到战斗即将开始。

因为收益流已经越来越多的受到公开源码电话的威胁，传统电信业者已经决定要开始一场令人担心的战争，希望藉此来破坏公开源码的革命。

4 技术瓶颈

有一种说法一直在我们周围流传：主要的网络提供商将对付费 VoIP 服务做标示并进行优先级排序，从而影响 VoIP 话务的质量。更糟的是，他们会对未经他们许可的 VOIP 话务进行侦测和干扰。

某些这样的做法已经开始出现，网络提供商正在对他们网络中某些类型的话务进行阻塞，这表面上是由于对某种公开服务做出的判决（如关闭资源共享服务来保护我们的隐私权）。在美国，FCC 已经清楚的表明了对这种做法的立场，并对从事这些行为的公司做出了惩罚。但是在世界的其他地方，执法机构并不会对 VoIP 如此的认同。

现在可以清晰的看到 Asterisk 组织和网络将一如既往的找到办法，绕过封杀。

5 法规之战

美国联邦通信委员会的上一任主席，米切尔.鲍威尔为改变 VoIP 的发展道路提供一个礼物。不要仅仅把 VoIP 认定为一项电信服务，鲍威尔积极拥护这个观点，即 VoIP 代表着通信的全新发展方向，并需要相应的法规空间来保证 VoIP 的发展。

VoIP 方面的法规条例将会被专门制定，但并不是在每个地方 VoIP 都会被当作一项电话服务。制定的法规中可能要包括：

为紧急事件服务提供信息

传统 PSTN 环路的特点之一就是它始终处于同一个地点。这对于紧急事件服务是非常有帮助的，因为可以通过电话发起的环路位置对主叫方进行定位。手机使用量的飞速增长已经使定位变得困难得多，因为手机没有一个确定的地址。一部手机可以和他们的网络连接并注册到任何的服务器上。如果不能识别电话的物理地址，那么就无法通过其拨打出的紧急电话来对主叫进行定位。VoIP 面临着类似的挑战。

执法机构的通话监听

执法机构能够对传统的电路交换电话线路进行监听。虽然正在制定相应的法规以便能够在网络上做同样的监听，但是要满足功能上的要求，技术上的挑战很难被完全克服。

反垄断行动

在美国，反垄断行动已经开始了，那些试图根据内容过滤话务的网络提供商已经遭到了罚款。

当进入立法阶段时，Asterisk 既是圣人又是魔鬼：说它是圣人因为它使穷人收益，说它是魔鬼，因为它为黑客和垃圾邮件制造者提供了前所未有的机会。关于公开源码电话的法规在某种程度上决定于相关的组织如何自律。像 DUNDi 这样与反垃圾邮件程序一体化的想法，是一个非常好的开始。相反的，对于类似主叫 ID 欺骗的主意，惩罚的时机已经成熟了。

6 服务的质量

因为基于 TCP/IP 的 Internet 采用了“尽力服务”的模式，所以现在仍无法知道日益增长的 VoIP 话务是否会全面影响网络的性能。目前在骨干网上还有如此多的额外带宽，因此采用“尽力服务”模式的骨干网在通常状况下确实运行良好。所以还需要反复的证明，是否一有更多的带宽，我们就有办法完全占用这些带宽。一兆的 DSL 连接在五年前无法想象，现在却已经不够用了。

也许摩尔法则[*]的一个推论对网络带宽适用。因为在没有任何特殊处理的情况下，网络仍具有保证充分运行的能力，因此 QoS 可能会失去意义。需要更高可靠性的机构可能会选择为更高级的服务付费。也许按分钟对长途通信付费的时代将会让位给按毫秒付费的时代，以保证快速的反应时间，或按照百分比计费来减少丢包。付费服务将具备“五九”可靠性[※]，而这正是传统电话公司所吹嘘的自己强于 VoIP 的地方。

[*]戈登.摩尔在 1965 年撰写的论文中预言，处理器上的晶体管数量每隔几年就会翻倍一次。

[※]这个术语指的是 99.999%，传统电信公司以此来吹嘘自己网络的可靠性。要达到“五九”标准，需要每年服务中断的时间不超过五分钟十五秒。很多人相信在 VoIP 完全取代 PSTN 之前，需要先在可靠性上达到这个标准。更多的人则相信即使 PSTN 也无法到“五九”可靠性。我们觉得这是一个描述高度可靠性的非常好的术语，然而市场部人员会极其频繁的对“五九”标准大发牢骚。

7 复杂性

开放性系统需要新的方法来设计解决方案。这仅是因为硬件和软件的廉价并不意味着解决方案的廉价。Asterisk 并不是产生于一个即用型的盒式程序，它需要设计和发展，以及随后的维护。虽然底层的软件是免费的，但是硬件成本要基于日用品的价格，可以公平的讲，在很多情况下，由于高度定制化的系统非常复杂，需要很强的调试能力，因此系统的调试费用将是解决方案成本中很大的一部分——高于传统交换机相应的费用。

凭借经验就可以根据系统的拨号规则判定：那些有相似特征的传统交换机与

它的设计大致相同。除此之外，只有实践才能让人准确的估计建立一个系统所需要的时间。

还有很多的东西需要我们去学习。

机遇

公开源码的电话创造了无数的机会。下面是其中比较引人注目的几个：

1 定制私有通信网络

有些人会告诉你，价格才是关键，但是我们相信 Asterisk 能够胜利的真正原因是：现在有可能像制作网站一样建立一个电话系统，每一个系统以及系统的每个部分都是彻底完全的客户定制。客户有这样的需求已经很多年了。只有 Asterisk 能提供这种服务。

2 低进入门槛

任何人都能对通信的未来有所贡献。现在对某些人来说，已经有可能仅使用一个两百美元的旧电脑，就开发出一个在智能上可以与最昂贵的私有提供相媲美的通信系统。我们承认硬件还没有进入生产阶段，但没有理由认为软件也是如此。这就是封闭系统将在竞争中面临困难的原因之一。能获得所需硬件的绝对人数不可能比一个关闭的商店里的人数更多。

3 集中精力放在提供业务上

交换机常常是按某种艺术形式设计出来的，但是在 Asterisk 之前，是寻找创造性的方法来克服技术局限的艺术。使用没有局限性的技术，同样的创造力可以被用在使客户需求得到完全满足的工作上。电信业的设计者会高兴的跳舞，因为他们巨大的创造力将能实实在在的为客户服务，而不是全被用在应付一些杂牌货上。

4 通信技术的恰当融合

如果不能满足人们解决问题的需求，那么公开源码的承诺最终将不过是一句空话。封闭性的行业失去了用户的关注，他们正在努力使产品同用户的需求相适应。

公开源码的电话带来了符合其他信息技术的语音通信方式。为了回应用户的需求，而不是垄断者的奇想，我们可以把 email，语音，视频与其他任何能想到的灵活的传输网络（不管是有线还是无线）上的服务结合起来，这终将成为可能。

电信的未来欢迎你！