

Asterisk 文档计划 第一卷

Asterisk 介绍

原著 : Leif Madsen

Jared Smith

Jim Van Meggelen

Chris Tooley

译者 : xiaoxuanzi (zhaoqx_78@hotmail.com)

译者序

1998 年，我供职于一家当时国内电信行业最大的国有制造企业，开始负责 VoIP 网关的研究和开发。我也有幸成为了国内最早从事 VoIP 研究开发的人员之一。在此后的两、三年内，我对 VoIP 有了较深入的接触和了解，并参与了国内 VoIP 行业标准的制定。

2000 年左右，国内运营商开始大规模建设 VoIP 网络。在开始 VoIP 研发之前，我已经在电信行业打拼了六年，出于对电信行业的熟悉和了解以及对 IT 技术尤其是基于 IP 的互联网技术的崇拜；随着我对 VoIP 的熟悉和了解，我自己认为：电信行业与互联网行业的融合会在之后的 5 年内到来。因此，借助于工作之便，我一直希望能组建一个团队，实现我自己的梦想：即以纯软件的方式实现电信网络的核心—电话交换机。但此后不久，由于我所在企业的国有企业作风，我们被要求整体转型，放弃之前所有的工作，转向一个新的领域。很快，我就离开了那家公司。来到了一家创业型的小企业，开始了新的打拼，每天为了生存而奋斗—我的梦就此破灭了。

时间很快来到了 2007 年的 5 月，正在为工作而烦恼的我偶然在网上看到了一些关于 Asterisk 的信息，我知道 7 年前我的预言失败了：电信网络与互联网的融合远没有我想象的那样容易。而在这若干年中，业界也并没有放弃我原先的设想，还有一大批人在为此目标而努力着。回头看看自己走过的路，我很惭愧，最近的六七年间，虽然接触了电信行业的方方面面，开阔了更大的视野，但却没有一件事是可以拿得出手的。我应该反思了，我觉得我该为 Asterisk 做点什么。

我开始研读 Asterisk 的相关文档和代码，我被开源社区众多的不知名的专业人士所震撼，感动于他们的执着。但我同时也发现了一个严峻的事实，那就是关于 Asterisk 的专业文档太少了，即使是英文的文档也极端的欠缺。更别说中文的了。我在这个行业多年了，深深的体会到如果没有文档，对一些新手将

会面临巨大的障碍，在当前这个浮躁的社会中，这个障碍将足以摧毁掉那些新手最初的誓言和决心。

以上这些就成了我最初翻译本手册的动因。

由于作者接触 Asterisk 的时间有限，对原文中的例子并没有完全去验证，加之本人的英文水平实在有限，尤其是对本文使用较多的连接词和短语理解不足，翻译过程中，不可避免会出现各种错漏。在读者无法理解本文的含义时，建议您对照原文一起来看。

事实上，理解和掌握 Asterisk 的最好的办法就是自己动手去实验。所有的文档都只是我们的参考。这是我个人多年的经验。

本手册希望以开源的形式在互联网上流传；译者也希望能够接受更多读者尤其是专业人士的批评和建议。如果您对手册的翻译和推广 Asterisk 有任何其它的建议或者意见，我都将尽我的能力来认真对待。

抱歉，我没有 QQ 号码。MSN 的号码我放在页脚上了。为避免 MSN 的骚扰(我已经饱受 MSN 的骚扰了,害得我经常会不用 MSN),请在加我之前,注明 :Asterisk 爱好者。否则，我对陌生的 MSN 是不会接受的。

感谢诸位。

2007. 7. 18

Asterisk 文档计划 : 卷一 : Asterisk 简介

By Leif madsen、Jared Smith、Jim van Meggelen、and Chris Tooley

版权 © 2004 Asterisk 文档计划

Asterisk 安装及基本使用指南。

本文档可以在满足于 OPL (Open Publication License) v1.0 或之后版本 (后续版本可以在 <http://www.opencontent.org/openpub/>获得) 所述的条件下被传递和分发。

- 版本更新历史

修订 版本 0.1 日期 2004/09/19

目 录

1. 简介 (Introduction)	7
1.1. Asterisk 的基本概念	7
1.1.1. Asterisk: 一个电话革命	7
1.2. 本文档的目的	8
1.3. 预先需要的知识和技能	8
1.3.1. 电话技术	9
1.3.2. Linux 安装和管理	9
1.4. 我们期望什么	10
2. 为 Asterisk 准备你的系统	12
2.1. 硬件(Hardware)	12
2.1.1. 平台和 Digium 硬件	12
2.1.2. 硬件最小需求	12
2.2. 组装你的系统	13
2.2.1. 安装板卡 (Installing Cards)	15
2.2.2. 选择一个操作系统/分发版本	15
2.2.3. Linux 需求	16
2.2.4. Fedora Linux 与其它的分发/操作系统	17
2.2.4.1. 获得 FC1	17
2.2.4.2. 为 Asterisk 准备的 FC1 概览	17
2.2.4.3. 为 Asterisk 安装 FC1	19
2.3. 总结(conclusion)	23
3. 获取和编译 Asterisk	24
3.1. 从 CVS 上获取 Asterisk	24
3.2. 电话卡驱动	24
3.2.1. 获取	24
3.2.2. 编译	25
3.2.2.1. Zaptel	25
3.2.2.2. 编译 ztdummy	25
3.2.3. Modprobe	26
3.2.3.1. Zaptel	26
3.2.3.2. ztdummy	27
3.2.4. 配置 zaptel 参数	28
3.2.5. 加载驱动	30
3.2.6. ztcfg	31
3.2.7. zttool	31
3.3. Asterisk	31
3.3.1. 获取	31
3.3.2. 编译	32
3.3.3. 测试	32

4. 通道配置.....	34
4.1. 通道简介(introduction to channels).....	34
4.2. FXO 和 FXS	35
4.2.1. FXS	35
4.2.2. FXO.....	36
4.3. IAX	37
4.4. SIP.....	40
4.5. 其它的通道协议.....	42
5. 拨号方案(Dialplans)简要介绍.....	43
5.1. 创建拨号方案简介.....	43
5.1.1. Contexts	44
5.1.2. Extensions (分机).....	45
5.1.3. Priorities(优先级)	46
5.1.4. Applications(应用).....	46
5.2. 一个简单的示例.....	46
5.2.1. 特殊的's'分机.....	47
5.2.2. Answer()、Playback()和 Hangup()应用	47
5.2.3. 我们的第一个 dialplan	48
5.3. 一个更有用的例子.....	49
5.3.1. 使用 Dial 应用的主叫通道.....	51
5.3.2. 增加附加功能.....	52
5.3.2.1. 在内部用户间处理呼叫.....	52
5.3.2.2. 变量(Variables).....	53
5.3.2.3. 增加变量.....	54
5.3.2.4. 一个有用的调试方法.....	55
5.4. 呼叫流程.....	55
5.4.1. 模式匹配.....	55
5.4.2. 利用\${EXTEN} 通道变量.....	56
5.4.3. 使用 Includes 连接 Contexts.....	58
5.4.4. 一些其它的特殊 extensions.....	60
5.5. 创建提示语(Creating prompts)	61
5.5.1. Record()应用的使用.....	61
5.5.2. Authenticate()应用的使用.....	62
5.6. 总结.....	62
6. 后记 (Colophon).....	64

1. 简介 (Introduction)

1.1. Asterisk 的基本概念

1.1.1. Asterisk : 一个电话革命

欢迎来到 Asterisk 的精彩世界。你会发现最强大和流行的开源 PBX 已经可以使用。

Asterisk 允许你定制一个电话系统以满足你的特殊需求。它通过提供一个电话功能的基本库来实现这一点。你可以象脚本一样来使用这些组件。呼叫到系统后，系统将会通过数字模式匹配(后文称为 extension 或分机)来触发这些功能，它可以让你相对容易的去完全控制复杂的呼叫路由。通用的 PBX 功能诸如语音邮件(voicemail)、呼叫排队(call queueing)、会议(conferencing)、保持音乐(music on hold)等等都可以被包括在内。但这仅仅是开始。Asterisk 是现存不多的可以通过相同的基于 IP 技术(诸如 IAX2、H323、SIP)来实现逻辑交换功能的 PBX 之一，它可以通过 PRI 或者模拟中继以相同的交换技术实现与传统电话技术的互联。这个强大且简单的核心允许其它系统中的复杂的概念可以被很容易的部署在 Asterisk 中。例如，创建一个 IVR 应用或者部署 CTI 功能可以比其它系统更便宜。为什么呢？因为采用 Asterisk，所有的相关功能都已经被完整的设置在內了。

或许 Asterisk 最有价值之处是其开放的系统特性。与其它任何开源的应用类似，Asterisk 可以被社区内使用它的人进一步加强。这个强大的概念保证了 Asterisk 能够适应并跟踪业界的需求。直接去更改源代码去适应你的需求，或者更进一步，贡献一个活动促进开发社区的成长。

由于 Asterisk 是如此的强大和灵活，在本书中，我们仅仅只是一个开始，我们覆盖了所有使用和配置的可能。我们将把主要的精力集中在系统最常使用

的特性，回答一些最经常被问及的一些问题。如果你可以通过我们在此提供的材料来工作，你将会在你成为一个 Asterisk 方案开发专家的路上做的更好。

1.2. 本文档的目的

作为一个开源的实验性产品，Asterisk 的发展已经超出了其最初的根源。由 Mark Spencer 领导的社区内的积极分子和开发者最近已经提供了一个平台，引起了广大观众的广泛兴趣。人们欣赏这个平台出奇的潜力，并使用它来解决多变的和过多迷人的问题。

新用户看来会有很多疑惑，但他们最大的疑惑在于怎么让一个最基本的系统建立和运行起来。

在本文档中，我们将与你讨论以下这些过程：选择一个平台、安装硬件、获取 Asterisk、编译、配置并启动它。

当我们完成这些后，你应当就会有一个基本的全功能的 Asterisk PBX，具有 FXO、FXS、IAX 和 SIP 连接。同时具备了大量的 Asterisk 可以完成的功能示例。

以上这些是否为你解决了你需要 Asterisk 解决的那些特殊的挑战？也许还不能。但你将拥有一个系统并从此开始您的发现之旅。

学习 asterisk 与学习一门新的编程语言是很类似的。一旦你获得了一些基础，剩余的会比较简单：那就是练习和时间。

1.3. 预先需要的知识和技能

由于 Asterisk 近乎无限的灵活性，成功的配置一个系统需要对几类技术概念都比较熟悉：象对电话的理解，但最需要掌握的是 linux 安装和管理。在本文档中，我们将抛弃掉这些复杂技术的表象而讨论与 Asterisk 设计、安装和管

理相关的概念。如果你希望了解前面所述的各部分的更多知识，我们给你推荐一些达到这些目的的途径。

1.3.1. 电话技术

Asterisk 是一个 PBX，这意味者你需要具备更多的电信知识，这样学习 Asterisk 才能更容易。如果你计划使用传统的 PSTN 电路和电话，你需要明白 FXO 和 FXS 的不同。如果使用数字中继，则需要对诸如 ISDN-PRI (包含了 T1 的线缆) 这样的技术比较熟悉。术语 PSTN 或者 VoIP 对你应当比较熟悉，你也应当获取并掌握一些模数转换的概念；知道什么是编解码。

在你还没有被吓倒之前，请理解，现在有许多优秀的参考文档可以帮助你来获取这些知识。一个非常好的介绍性作品是《Noll's Introduction to Telephones and Telephone Systems : Noll 的电话和电话系统简介》，由 Artech House Publisher 出版。权威的电信行业百科全书是牛顿电信词典 (Newton's Telecom Dictionary) 由 CMP Books 出版，这本书也是任何电信专家书架上的必备书。

1.3.2. Linux 安装和管理

在你安装和使用 Asterisk 之前，你需要一个 i386 兼容的 Linux 安装系统。如果你对 Linux 管理的概念还缺乏深入的了解，我们建议你在开始 Asterisk 安装之前先关注并加强该方面知识的了解。在我们开始行走之前，所有人都必须先会爬行，不是么？在互联网上，Linux 文档项目 (<http://www.tldp.org>) 为初学者提供了大量的资源。在书店中，Frisch 的《基本系统管理》以及 Nemeth, et al 的《Linux 管理员手册》和《Unix 系统管理手册》会被推荐。由 Matt Welch, Lar Kaufman et al. 书写的《运行 Linux》也是一个全面的最成功的 Linux 介绍。读一本或者两本这样的书籍将在你后续的路程中减少你大量的头疼的时间。

1.4. 我们期望什么

- **Asterisk 不是一个成型的系统 (Asterisk is not a turnkey system)**

Asterisk PBX 系统是一个复杂的软件体系。学习曲线将会非常的陡，简单的阅读某个单一的章节(资源)不会让你掌握 Asterisk 能够做的任何事情。我们可以安全的打赌：即使是 Mark Spencer 也不可能知道它创建的系统所能实现的所有功能。

本书将试图定位于一些对新手而言会遇到的最通用的一些事情，采用一步一步的处理方法，最终完成一个功能完备的 Asterisk PBX 系统的创建。

学习 Asterisk 怎么工作与学习一门新的编程语言是非常类似的。为了掌握所有的配置文件是如何工作的以及怎么控制多个接口，我们需要把许多小时的时间花费在 Asterisk 上。为了追随我们更多的简易目标，我们将会覆盖更多的话题，但是一些事情必须被讨论和理解。

理解拨号方案(dialplan)是一个基础的概念，这对 Asterisk 而言是一个全新的概念，必须要掌握。Asterisk 用来通信的不同通道类型的配置最终也会被带到拨号方案中。这是 Asterisk 系统的核心。

当试图安装他们的第一个 Asterisk 系统时，许多人都会体验到更多的疼痛和挫折，因为他们以为他们在后续的几个小时中会完成一个产品化的系统。当然，一旦你学习了所有的概念，这也是有可能的；但是很少有人能够在开始创建他们的第一个系统之前有足够的时间。本书的目标就是带领你尽可能快的掌握 Asterisk 系统，但我们还是建议你花费一些时间并享受一下整个过程。如果你给出了相当的时间去爱上它(没错，你会爱上它的)，你将会感到使用 Asterisk 是一件非常幸福的事。

- **不喜欢它？你可以改变它**

Asterisk 是开源的软件。可以读到源代码是它的强大之处。大多数的其它 PBX 系统是完全源码封闭的，限制了你的决定只能被局限于设计者已经考虑的特

性之内。而在 Asterisk 中，如果有什么不能象你希望的那样去工作，你可以改变它！当然，如果你没有足够的编程能力，你还做不到这一点。与此相对，这些能力可以很轻松的从那些社区内已经开发的相似的产品哪里获得。用一个你所拥有的系统来试一下它们！

源码自身也是一个很优秀的调试和学习的资源。通读那些 Asterisk 源代码目录下的大量文本文件和代码文件可以教会你更多系统中复杂的工作机理。

- **自由和开源软件：GPL 和 LGPL 授权**

Gnu 的公共授权 (Public License) 是一个令人兴奋的概念，关于它已经有很多材料了。自由软件基金 (The Free Software Foundation) (www.fsf.org)，可能是开始搜索 GPL 最好的地方。

遵守 GPL 概念的 Asterisk 的工作是这样的：与大多数 PBX 而言，仅仅为了安装软件，你需要付大量的授权费用。但在 Asterisk 下，所有你要付出的就是关心 GPL 付与你的义务。简单的说，如果你不希望遵守 GPL 术语，那你应当：
a)、停止使用 Asterisk，b)、与 Asterisk 的版权持有者 Digium 签署一个独立的授权协议。

GPL 是完全自由的，你可以象你希望的那样，自由的下载和使用 Asterisk。但你应当知道：使用任何 GPL 软件，你都应该遵守 GPL 术语的规定。

2. 为 Asterisk 准备你的系统

本章将帮助你为安装 Asterisk 而准备相应的系统。Asterisk 可以工作在许多平台和操作系统上，但是为了保持尽可能的简单，我们将会选择并绑定到一个单一的平台和 Linux 分发。对你而言，这些指令可能能够工作在不同的 Linux 分发版本中，但它们并没有被测试过。

2.1. 硬件(Hardware)

2.1.1. 平台和 Digium 硬件

当 Asterisk 已经被成功的编译并运行在任何 x86 平台上(它当前仅只支持该平台)时，为了使它更简化，对一个开发者而言，业余或者教学系统是在任意的 PC 机上安装 Linux。为了更透彻和理解环境，平台的选择需要对系统架构、负载均衡等等有比较透彻的理解。这些讨论超出了本文档的范围。

2.1.2. 硬件最小需求

Asterisk 是非常处理器敏感的软件，由于它使用了 CPU 来完成数字信号处理(DSP)。如果你正在搭建一个复杂的、高使用量的系统，理解这一点是相当重要的。然而，对搭建你的第一个 Asterisk PBX 而言，你可以安全的使用任何 Intel 兼容的好于一个 300MHz、Pentium 256M RAM PC(x86)。基本上，你可以在上面安装和运行 Fedora Core 1。它将会适合于你的第一个 Asterisk 的安装(笔者已经在 700MHZ Celerons 到 Athlon XP 2000 上运行过 Asterisk)。

Asterisk 不需要很多空间(大约需要 100MB 用来编译)，但是源码、voicemail、客户提示等都需要存储。你可以很轻松的在大约 4G 硬盘空间的情况下创建一个开发系统。

如果你正在搭建的一个只有 VoIP 的系统，那你不需要额外的硬件去编译和使用 Asterisk。你的分机可以是使用自由的可用的 VoIP 软电话，例如 Xten 的 X-Lite SIP 电话。线路可以通过一些 VoIP 运营商例如 FWD 来提供。除了计算机系统外，一个只有 VoIP 的系统可以让你评估 Asterisk 而不要任何费用。然而，为了更全面的探索 Asterisk 的能力，你将会发现你需要安装一些 Digium 的硬件。我们建议你考虑从 Digium 那里获取一个 PCI 开发工具。

注意：许多只运行 VoIP 的 Asterisk 系统需要一个时钟源来提供时间。Digium 卡已经内置了该能力，因此，如果你已经有了一个这样的卡，你就已经获得了需要的时钟。对系统而言，如果没有时钟源，可以采用 ztdummy。Ztdummy 驱动使用了 USB 控制器作为一个时钟源为应用例如 MeetMe 会议应用提供所需要的时间。有两种主要类型的 USB 控制器芯片可以用在主板上。这包括 UHCI 和 OHCI 类型的芯片。为了让 ztdummy 工作，你应该需要 UHCI 类型的 USB 控制器。OHCI 类型的芯片也能工作，但它需要 zaprtc 模块。整个 ztdummy 模块的讨论超出了本文档的范围，但我们想提级这一点是因为缺乏时钟源可能会影响到一个只有 VoIP 的系统。

Ztdummy 模块通常被认为是一个黑客(Hack)，也没有真正想成为一个产品。任意一块 Digium 卡都可以提供时钟源。

2.2. 组装你的系统

Asterisk PBX 所需要的硬件平台实际上并不比拆开的一个 PC 更复杂。你可以不需要很强功能的视频卡以及其它诸如串口、并口、USB 口等都可以被完全禁止。仅仅一个网卡是必须的。

如果你正在使用任何 Digium 卡，你可能将需要根据你的主板的指示来决定哪个 PCI 插槽可以适用于该卡。许多主板在 PCI 插槽上共享了中断信号，在 Asterisk 中，中断冲突是影响语音质量的一个潜在原因。

一个解决 IRQs 的可能的的方法是，你可以在 BIOS 中禁止掉那些你不需要的所有设备。例如：串口、并口和 USB 口都是 Asterisk 所不需要的资源。于是，你可以考虑释放这些 IRQs 为了使资源分配更容易。

● IRQ 共享问题

许多电话卡例如 X100P 可能会产生大量的中断；这些中断的服务需要花费时间。驱动可能不能及时的处理它，如果其它的任何设备正在处理同一个共享的 IRQ 且 IRQ 线不能接受另外的中断。它往往可以在一个 SMP (APIC) 的系统上工作良好。在一个单一芯片的系统中，你可以获得中断缺失和非对齐时钟。Digium 的任意卡或者其它的电话卡可以解决这个问题。由于在电话系统中准确的 IRQ 延迟是必须的，因此，我们不当与其它任何设备共享 IRQs。这并不是说你将会需要有一个 IRQ 共享冲突，而是有一些事情你必须知道。

如果你正在决定采用那台计算机来安装 Asterisk，那你就尽可能多的释放一些 IRQs 例如：在 BIOS 中禁止 USB、串口和并口的支持。基本上来说，你希望释放尽可能多的 IRQs。你可以不用去看一个 NIC 是否与一个 TDM 或者 FXO 卡共享了同一个 IRQ。但这些卡有它自己的 IRQ 是最好的。

大多数的 BIOS 都允许你手工的去分配 IRQ 给某个特定的插槽。到 BIOS 系统中查看一下 IRQ 区域，通常会位于第二页。如果它缺省被设置为 AUTO，你试着把它设置为手工看看发生了什么。一个表格将会可用来让你手工的为每个插槽分配 IRQ。

注意：需要密切注意的是在一些主板上是插槽共享 (sharing) IRQs。查找一下 BIOS，检查是否在 IRQ 表中有一个类似” 1/5” 的入口。这是运行 Digium Zaptel 板卡根本不希望的系统。通常这类特性是高低价主板的不同点之一。

一旦系统启动，可以查看 `/proc/interrupts` 来看一下分配的 IRQs。

注意：以下输出仅只是一个附带 Digium 硬件的例子。它仅在你已经启动了硬件驱动后才可用(这将在第三章讲述)，但你应当知道这些。

```
cat /proc/interrupts
CPU0
0: 41353058 XT-PIC timer
1: 1988 XT-PIC keyboard
2: 0 XT-PIC cascade
3: 413437739 XT-PIC wctdm <-- TDM400
4: 5721494 XT-PIC eth0
7: 413453581 XT-PIC wcfxo <-- X100P
8: 1 XT-PIC rtc
9: 413445182 XT-PIC wcfxo <-- X100P
12: 0 XT-PIC PS/2 Mouse
14: 179578 XT-PIC ide0
15: 3 XT-PIC ide1
NMI: 0
ERR: 0
```

以上你所看到的是三块 Digium 卡每块都有自己的 IRQ；如果是这样的，你可以继续安装硬件驱动。如果不是这种情况，我们强烈建议你干扰一下 BIOS 直到 Digium 卡不再与其它任何设备共享 IRQs。

2.2.1. 安装板卡 (Installing Cards)

如果你已经从 Digium 购买了一个 Asterisk 开发工具，你将可以直接将你的系统与公共电话网相连，就象从这儿运行了一个模拟电话那样。如果你还没有一个开发工具，你将被限制于只能使用 IP 中继和客户端。

2.2.2. 选择一个操作系统/分发版本

Asterisk PBX 最初是为 Linux 操作系统所设计的。由于 Asterisk 不断的流行性(和一个活动的开发社区)，它已经被移植到了 BSD 和 OS X 上，不过，Digium

的 PSTN 卡被设计于工作在一个 i386 的 Linux 系统上，且 Linux 也仍然是官方支持的操作系统。因此我们建议那些 asterisk 的新手使用 Linux。

2.2.3. Linux 需求

- 已经测试的 Linux 分发版本

Asterisk 可以工作于大多数的 Linux 分发中。许多分发例如 RedHat、Fedora、Debian、Mandrake、Slackware 和 Gentoo 等都被开发者所成功的使用。如果你发现有什么东西在一个特定的系统上无法工作，你可以发送一个 bug 报告，查阅：<http://www.digium.com/bugtracker.html>

- 最小内核版本

Asterisk 被设计工作在 Linux 内核版本 V2.4 上。然而对内核 2.6 也有一些支持。如果你想创建一个稳定的系统，我们建议你使用 2.4 内核。以下章节将假定你运行在一个 2.4 内核的系统上。

- 需要的包

在此之前，安装 Asterisk 需要一些包，但现在象 readline 和 readline-devel 包都不再需要了。由于没有特殊的硬件需要(例如一个声卡)，因此，需要的包就是 Asterisk 自身了。如果你正在使用 Digium 的硬件或者 ztdummy，你将需要 zaptel 包。一些 Asterisk 应用需要 zaptel 包在编译时被包括在内。如果你选择编译 Asterisk 而没有 zaptel，你会发现你将遗失一些 zaptel 包有关的应用(例如：MeetMe)，你将不得不编译 zaptel 并为那些被包含的应用重新编译 Asterisk。对 T1 和 E1 接口而言，libpri 包是需要的。Bison 也需要来编译 Asterisk。如果你希望创建 newt 工具(例如：astman)，那么 ncurses 和 ncurses 开发包也是需要的。

注意：从 2004 年 10 月 18 日开始，zlib 和 zlib-devel 包现在也需要来编译 CVS-HEAD。这是由于附加的 DUNDi 协议(Distributed Universal Number

Discovery)。它可以使用 yum 来安装，从命令行执行命令 `yum install zlib zlib-devel`。

2.2.4. Fedora Linux 与其它的分发/操作系统

为了我们的安装，笔者选择使用 Linux 的 Fedora Core 分发版本。一个经常被问及的一个问题是：Linux 的哪个分发来运行 Asterisk 是最好的？那么答案就是：哪一个分发是你最熟悉的？Asterisk 可以运行在大多数(如果不是全部的 Linux 分发上，因此，在你最喜欢的系统上安装 Asterisk 是如此的感到自由。为了帮助你在第一次安装 Asterisk 过程中尽可能的降低出错的可能，我们决定使用一个单一分发。Fedora Core 被选中是由于它是笔者最熟悉的分发版本。

2.2.4.1. 获得 FC1

Fedora Core 1 和 2 都可以从 <http://fedora.redhat.com/> 来获得。两者的最大不同是分别使用了 Linux 的 2.4 和 2.6 内核。我们的安装将开始于基于使用 2.4 内核的 FC1，这个内核是 Asterisk 运行所建议的内核。这些指令将会在任意分发版本中工作，但那些不同我们会列出并解释他们。

2.2.4.2. 为 Asterisk 准备的 FC1 概览

- 产品型/干净的系统 与 实验型/业余爱好的系统

如果一个 Asterisk 系统将被部署在一个产品化的环境下，请小心评估。可能需要付费去进行系统设计。不仅 Asterisk 服务器需要被优化来确保电话功能采用最高的优先级，而且在多数情况下，Asterisk 系统都不应当被运行其它任何进程。如果希望有一个数据库或者 Web server 功能，仔细考虑这些服务应当被部署到第二台支撑的服务器上，我们反对其与 Asterisk 位于同一平台上。

我们已经听说，使用现代 PC 的性能，你可以运行 5 个(或者更多)电话在一个业余的系统上应该没有什么问题，即使你所选择的 Linux 分发有一个全安装。

底线是 Asterisk 可以被相当轻松的在一个具备所有铃声和口哨的 Linux 上安装。但是如果你想创建一个 30 或者更多的 PBX 站点，你最好是让操作系统不要安装那些除了 Asterisk 所必须的其他包。

- **什么是需要的**

以下软件将被安装：

Zaptel (模拟卡的驱动)

Asterisk (PBX)

Bison (编译源代码需要)

你可以通过本文档的剩余部分与我们粘在一起，由于这些项目我们将会更详细的解释。

- **哪些是不需要的**

Asterisk 是一个相当性能敏感的应用。这意味着你必须小心的考虑系统资源都会被哪些其它你正在运行的应用来使用。在这一点上，最重要和不必要的应用是 X Windows 系统。在 Linux 上流行的代表是 Gnome 和 KDE。不仅它们是不需要的，而且你如果在你的 Asterisk 上的运行它已经被论证是糟糕的事情。请记住 Asterisk 是一个服务型应用，因此我们不需要一个桌面。

你可以在 X 下面成功的运行和安装 Asterisk，但请知道从性能的观点来看这不是一个好主意。你应当从来都不这么配置以便进入到一个产品化的环境中。如果你想正确的学习 Asterisk。你就应当离开 X 而能够做的更好。

- **最新的补丁和更新**

Linux 的 FC 分发包含了一个便利的实用工具称为 yum。使用这个工具我们可以确保我们的包都被更新为最新的安全稳定的版本。当我们安装完 FC1 后，我们应当在登陆后运行这样的最初命令：`yum update`。这将联系主要的 yum 镜

像站点去获取最新的更新并安装它们和它们所依赖的包。我们需要确保在我们编译和安装 Asterisk 软件之前运行了这个命令。Yum 也会为你安装最新的内核，但是为了使它生效，你将不得不重新启动你的系统。在你编译 Asterisk 之前做这些，否则 asterisk 将不会工作，在你再次启动它之前，你将不得不重新编译它。

2.2.4.3. 为 Asterisk 安装 FC1

- 初始步骤

插入你的第一张 FC1 盘并启动系统。当你看到 splash 屏幕时，输入 linux text 来进入到非图形化的安装中。安装前，你将会被问到是否测试媒体。为了可能避免一些安装上的问题，在这个点上进入并测试媒体。

Fedora 将会探测你的硬件。当这些完成后你将会看到 FC 的欢迎屏幕。输入 OK 继续。

下面的屏幕将允许你客户化你的语言、键盘和鼠标型号。为语言和硬件作出合适的选择。

你将会被提示你希望安装那种类型的系统，高亮 custom 并选择 OK。

- 磁盘分区

现在我们必须要为我们的硬盘进行分区。Fedora 给了我们选择：或者由 Fedora 为我们创建分区 - 自动分区 auto partition；或者允许我们自己使用 Disk Druid 创建分区。如果你感觉自己创建分区更合适或者你喜欢客户化它，则选择 Disk Druid。我们假设你选择了自动分区 Auto partition。

警告：

我们假定你创建这个系统仅仅为了安装 Asterisk。如果你还有其它硬盘和分区想保留你所希望的内容，请小心不要破坏了这些信息。

选择 Remove all partitions on this system 并选择 OK。你将会看到由 Fedora 为你创建的分区的状态。如果没什么问题，选择 OK。

● Boot Loader

你会被给定一个选项： Use GRUB Boot Loader 还是 No Boot Loader ；选择缺省的 Use GRUB Boot Loader 并选择 OK。

一些系统在内核启动过程中，需要传递一些特殊的信息给内核。如果你不需要或者还不确定，只需要保留缺省设置并选择 OK 即可。

如果你喜欢在你的启动加载器中使用密码，现在就插入一个并选择 OK 继续。

注意：正常情况下，你不需要在一个产品化的系统中使用启动加载密码，由于你可能需要有效的保护系统，以使它能够在没有人工干预的情况下重启以恢复系统。在一个实验型系统中，一个启动加载器可能有效的防止人们弄乱你这个新玩具。

下一个屏幕显示了启动的标号和准备从那里启动的设备。缺省设置应该没什么问题，选择 OK 继续到下一个屏幕，这里你会被问及你希望将启动加载器安装在哪里。对大多数的安装而言，你需要缺省的设置 Master Boot Record。选择 OK 继续。

● 网络配置

如果你有网卡安装在计算机中，Fedora 将会询问你配置它们。为你的网卡配置你的网络信息选择 OK 继续进行防火墙的配置。

当计算机可能随时需要暴露到 Internet 上时，使用防火墙无疑是一个好的主意。Asterisk 将工作在防火墙之后只要将合适的端口打开。例如：SIP 用户使用端口 5060 来实现通信消息的传递，但 RTP 音频流会动态的使用其它端口。为了允许音频穿透防火墙，RTP 端口必须被打开。这样，我们才能听到主叫方的声音。IAX2 使用端口 4569，由于它的通信消息和 RTP 音频流都运行在同一个单

一的端口号上，因此在使用防火墙时会有一些小问题。由于这是我们的第一个 Asterisk 安装，我们更愿意不再限制任何可能的系统配置过程中需要的端口，我们将去禁止掉防火墙。然而，请确保在你部署和把它连接到 Internet 之前，需要合适的安全方案。Asterisk 系统的安全不在本文档的讨论范围之内。

- **语言支持和时区选择**

选择英语 English(USA) 作为语言并选择 OK 继续。你会被询问是否将系统时钟设为 UTC 时间且你位于哪个时区。作出合适的选择并选择 OK。

- **设置 root 用户密码**

为你的系统设置并确认 root 密码。

- **包选择**

反选掉所有的包包括 X Window 系统。然后从列表中选择以下三个包：

Editors

Development Tools

Kernel Development

注意：如果你正在使用 Fedora Core 2，你就不需要安装内核开发包，你只需要设置符号连接到相应点 `/lib/modules/`uname -r`/build` 就可以了。这会在以后解释。

当你选择了其它的可能不必要的包将会在编译和运行 Asterisk 时引起一些问题。在获取、便宜、安装和配置 Asterisk 时，我们都将通过选择我们需要的包来坚持我们的最小化原则。以上三个包将会使我们达到这个目的。选择 OK 继续。

下一个屏幕会告诉你安装已准备开始。选择 OK 继续包的安装。Fedora 将询问你确认具备以上三个安装包的 CD。选择 continue 开始拷贝和安装操作系统。

- **安装后**

如果你想创建一个启动的软盘，现在就制作一个。如果不需要，选择 No 继续。你的系统现在应该安装好了。选择 Reboot 重启系统并开始 FC1。在系统重启后，我们将运行几个命令来完成我们为 Asterisk 而做的准备。

一旦系统重启成功，以 root 用户登陆。登陆后，运行 yum update 命令更新系统文件和内核。Fedora 将下载 RPM 头文件并解析相互的依赖关系。完成这些后，系统会询问你是否安装这些文件，输入 Y，按下回车 enter。一旦系统文件被更新后，重启你的系统以使新内核能够生效。

系统重启后，校验符号连接到了内核源代码。执行以下命令：

```
cd /usr/src/  
ls -la
```

符号连接 linux-2.4 应该被指向了你的内核源文件。你可以校验一下这个符号连接是你当前的内核而不是一个老的内核：比较一下在符号连接上显示的版本号和 `uname -r` 输出显示的版本号。

```
#ls -la  
lrwxrwxrwx 1 root root 24 Sep 5 12:36 linux-2.4 -> linux-2.4.22-1.2199.npt1  
#uname -r  
2.4.22-1.2199.npt1
```

以上示例显示了符号连接正确的连接到了当前运行的内核上。如果连接不正确，我们可以通过手工执行 `ln` 命令来创建一个符号连接：

```
#ln -s linux-2.4.22-1.2199-npt1 linux-2.4
```

注意：如果正在运行 Fedora Core 2，那么你的符号将连接到不同的位置。2.6 内核包括了你可以 build 你的软件的目录来代替内核源代码。符号连接可以采用命令：`ln -s /lib/modules/`uname -r`/build` 来创建。

你的 Linux 系统现在应该准备好来安装 Asterisk 了。

2.3. 总结(conclusion)

你应当有一个完整的 Fedora Core 1 (或 2) 的 Linux 分发软件来准备安装 Asterisk。下一章我们将介绍获取、编译和安装 Asterisk。

提示：

1. <http://www.digium.com/bugtracker.html>
2. <http://fedora.redhat.com/>

3. 获取和编译 Asterisk

3.1. 从 CVS 上获取 Asterisk

- 什么是 CVS ?

CVS 是一个开发人员用来控制源代码的中心数据库。当源代码所做的变更提交到 CVS 服务器时，它就会立即可用来下载和编译。使用 CVS 的另外的辅助特性是如果有什么可以工作在一个点，但更改后引起了中断，那么任何特定的文件版本都会被回滚到那个特定点。这是事实对整个树而言可以非常好的工作。如果你发现有什么功能可以在一个点工作但安装了 Asterisk 的最新版本却引起了中断，那么，你可以及时回滚回那个点。我们可以查阅“从 CVS 获得文件”一节。

3.2. 电话卡驱动

3.2.1. 获取

为使用 Digium 硬件，我们需要获取 Zaptel 驱动，你将不得不从 Digium CVS 服务器检出(checkout) zaptel 分支。

示例 3-1 : 从 CVS 检出(check out) zaptel 驱动

```
cd /usr/src/  
export CVSROOT=:pserver:anoncvs@cvs.digium.com:/usr/cvsroot  
cvs login  
密码是 anoncvs  
cvs checkout zaptel
```

你将会连接到 CVS 服务器上，从那里下载所有的编译 Zaptel 驱动所必须的所有文件。这些文件被存储在 /usr/src/zaptel/ 目录下。

3.2.2. 编译

3.2.2.1. Zaptel

如果你计划使用 `ztdummy` 或者任意的 Digium 硬件，你就会需要编译 `zaptel` 模块。为任何你可以安装在你系统中的 Digium 硬件，以下命令将编译和安装这个硬件的驱动模块。如果你没有任何 Digium 硬件，请查阅 “关注 `ztdummy` 编译” 章节。

示例 3-2 编译 `zaptel` 驱动

```
cd /usr/src/zaptel/  
make clean  
make install
```

注意： 如果使用 Fedora Core 2 或者其它任何使用 2.6 内核的 Linux 分发版本，你将需要在 `make install` 之前做一个附加的步骤。如果使用了 2.6 内核，执行以下命令：

```
cd /usr/src/zaptel  
make clean  
make linux26  
make install
```

3.2.2.2. 编译 `ztdummy`

- 校验 USB 模块

当你没有任何 Digium 硬件但却需要定时器时(例如 Music On Hold 或者会议将需要定时)就必须使用 `ztdummy` 模块。`Ztdummy` 驱动需要在你的主板上具有一个 UHCI USB 控制芯片。如果你正在使用 OHCI USB 控制器，你会还需要使用 `zaprtc`。你可以检查一下看你的主板有没有 UHCI USB 控制器：从命令行运行 `lsmod`。

```
#lsmod
```

```
Module Size Used by Not tainted  
...  
usb-uhci 24524 0 [unused]  
<-- usb-uhci  
usbcore 71680 1 [hid usb-uhci]  
...
```

以上截屏显示已经加载了 USB 模块。你可以看到 `usb-uhci` 这一行。这显示了 UHCI 模块已经被加载可以为 `ztdummy` 模块使用。

- **编辑 Makefile 文件**

为了编译 `ztdummy` 模块，你还需要编辑 位于你的 `/usr/src/zaptel` 目录下的 Makefile 文件。找到包含了以下信息的这一行：

```
MODULES=zaptel tor2 torisa wcusb wcfxo wcfxs |  
ztdynamic ztd-eth wctlxxp wct4xxp # ztdummy
```

去掉#解除对 `ztdummy` 模块的注释。存储文件象正常一样执行编译。一旦你已经成功的编译了 `ztdummy` 模块，你可以使用 `modprobe` 将它加载到内存中。

3.2.3. Modprobe

`Modprobe` 被用来加载 `zaptel` 驱动到内存中以便我们可以在我们的系统中存取硬件。我们总是先加载 `zaptel` 驱动到内存中。然后再加载我们准备加载的特定类型设备的驱动 (`FXS`、`FX0`、`ztdummy` 等等。)

3.2.3.1. Zaptel

我们可以使用以下命令来加载 `Zaptel` 模块

示例 3-3 加载 `zaptel` 驱动

```
modprobe zaptel
```

如果 `zaptel` 模块加载成功，那么在回车后你不会看到任何输出。我们可以运行 `lsmod` 命令来校验 `zaptel` 模块是否加载成功。如果 `zaptel` 加载成功，我们应当可以看到以下类似的信息：

```
#lsmod  
Module Size Used by Not tainted  
zaptel 178144 0 (unused)  
← zaptel  
...
```

就象我们可以见到的那样，最初列出的模块就是 `zaptel`。`Zaptel` 模块被我们的通道类型模块使用，因此，目前还是未使用的 (`unused`) 状态，一旦我们加载了 `FXO` 和 `FXS` 端口这个状态将会改变。我们也可以使用 `tail` 命令检查一下 `/var/log/messages`。我们会看到有一行信息说 `zapata` 电话接口已被注册。

```
#tail /var/log/messages  
Sep 5 13:44:47 localhost kernel: Zapata Telephony Interface Registered  
on major 196
```

3.2.3.2. `ztdummy`

如果我们没有任何 `Digium` 硬件，我们会需要使用 `ztdummy` 模块。记得我们需要 `UHCI` `USB` 控制器来为 `ztdummy` 模块提供时钟源。加载 `ztdummy` 模块与 `zaptel` 模块非常相似，使用 `modprobe` 命令来实现。由于 `ztdummy` 模块依赖于 `zaptel`，你应当在加载 `zaptel` 模块后再加载 `ztdummy` 模块。

示例 3-4：加载 `ztdummy` 模块

```
modprobe ztdummy
```

重复说明一下，我们不会从 `modprobe` 命令看到任何输出。我们可以使用 `lsmod` 来检验以下 `ztdummy` 模块是否被正确加载。

```
#lsmod  
Module Size Used by Not tainted
```

```
ztdummy 2580 0 (unused)
<-- ztdummy
zaptel 178144 0 [ztdummy]
<-- zaptel
...
usb-uhci 24524 0 [ztdummy]
<-- usb-uhci used by ztdummy
usbcore 71680 1 [hid usb-uhci]
```

看一下 `lsmod` 的输出，我们会发现我们的 `ztdummy` 模块已经正常加载了。请注意 `ztdummy` 模块正在同时使用 `zaptel` 和 `usb-uhci` 模块。这意味着 `ztdummy` 同时依赖于 `zaptel` 和 `usb-uhci` 模块。

3.2.4. 配置 `zaptel` 参数

为了为物理的电话通道配置区域参数和信令，需要编辑 `zaptel.conf` 文件。这个文件包含了许多选项和参数，它们可能未在本书中提级。

为了本文档的目的，我们将把我们的系统限制在包括在一个开发工具中，它具备一个单一的 FX0 接口和一个单一的 FXS 接口。

- **编辑 `/etc/zaptel.conf`**

`Zaptel.conf` 文件位于 `/etc` 目录下。

```
#cd /etc/
```

使用你喜爱的编辑器打开 `zaptel.conf` 文档。你会注意到一大段的注释行。为增加自己的知识，你可以读一下这个信息，但现在我们将忽略大部分。缺省情况下，仅只有两行未被注释，这两行位于接近文件结束的附近。`loadzone` 和 `defaultzone` 参数。现在我们保持它们作为缺省。

注意：`loadzone` 参数允许你指定一些可以被接口产生的一些 tones。一个逗号隔离了两个代表国家码的字母它将会为这个国家或区域加载这些 tones。当前可用的 tone 的列表区域定义在 `zonedata.c` 文件中，这个文件位于 `/usr/src/zaptel` 内。

在 `zaptel.conf` 文件中，我们将定义通道将会使用的信令类型。我们也会定义哪些通道将被加载。使用 `ztcfg` 命令可以产生一些信息，这些信息可以被用来配置通道；`ztcfg` 命令我们将在本章的后面讨论。当你正在处理 FX 接口时，硬件描述必须基于它连接到了哪里，例如：一个 FX0 接口这样命名是因为它连接到了一个 Office。而 FXS 接口则连接一个 Station（用户）；然而，信令需要定义在我们现在正在仿真的设备上。由于 FX0 接口连接到一个局端，我们需要软件来仿真一个用户端。而针对 FXS 接口则恰恰相反，我们需要软件来仿真一个局端。总之，硬件需要根据它连接到的对端来描述；但软件或者驱动，则是根据它需要展现的行为特性来定义。

注意：FX0 和 FXS 设备怎么在软件中来描述对不同的厂家会有不同的做法，理解这一点是非常重要的。但 FX0 和 FXS 针对硬件的描述通常会与作者的决定相一致。

由于我们正在假定你配置 `zaptel.conf` 文件是为了使用 TDM400P 开发工具（它包含了一个 FXS 和 FX0 接口），我们的例子将集中在这个卡上。然而，那些已经学习的概念可以被其它的接口所接受，包括通过 FX0 和 FXS 模块在 TDM400P 卡上增加更多的通道。

示例 3-5：为使用 TDM11B 而配置 `zaptel.conf`

```
#  
# Zaptel Configuration File  
#  
fxoks=1  
fxsks=4  
loadzone=us  
defaultzone=us
```

TDM11B（在 TDM400P 卡上有一个单一的 FXS 和单一 FX0 模块）标准情况下配置了一个 FXS 模块位于 TDM400P 的第一个端口。一个 FX0 模块附着在这个卡的第 4 个端口上。`fxoks=1` 这行告诉 `wcfxs` 模块在 TDM400P 的第一个口上使用 FX0 信令。类似地，`fxsks=4` 行告诉模块采用 FXS 信令来加载第四个端口。一 ks 开

头的部分行指出使用 koolstart 监视协议。Koolstart 协议不是一个工业化的已知监视机制。Koolstart 为电路能力增加了智能来监视远端端点正在做什么。这是一个非常 cool 的特性，因此命名为 koolstart。由于 koolstart 合并了 loopstart 和 groundstart 的优势(分别以 ls 和 gs 表示)并超过了两者。你差不多总是会使用它除非你遇到了兼容性问题。Koolstart 已经成为使用 Asterisk 的非正式标准。

3.2.5. 加载驱动

配置了 zaptel.conf 配置文件后，你必须通过使用 modprobe 加载 wcfxs 驱动。这与你在前面章节中加载 zaptel 和 ztdummy 是非常类似的。

注意：如果你在前面章节中加载了 ztdummy 模块，我们建议你在加载 wcfxs 模块之前先卸载这个驱动。你可以使用 rmmod 命令来实现这一点。为了卸载 ztdummy 模块，执行以下命令：

```
rmmod ztdummy
```

如果成功了，你不会看到任何输出。

wcfxs 驱动可以使用 modprobe 命令来加载，如下：

```
#modprobe wcfxs  
Freshmaker version: 71  
Freshmaker passed register test  
Module 0: Installed — AUTO FXS/DPO  
Module 1: Not Installed  
Module 2: Not Installed  
Module 3: Installed — AUTO FXO (FCC mode)  
Found a Wildcard TDM: Wildcard TDM400P REV E/F (4 modules)
```

就象你所看到的那样，这个模块执行了一系列的测试并将结果输出到屏幕上。这允许你看到内核模块的关于物理接口正确的加载信息。加载驱动后，你必须使用 ztcfg 来配置通道。这将会在下一节解释。

3.2.6. ztcfg

ztcfg 命令被用来配置物理 FX 接口所使用的信令。Ztcfg 将使用描述在 zaptel.conf 中的信令配置。为了看到输出和 ztcfg 命令的结果，我们必须增加一个 -vv 以使程序输出详细信息。

```
#!/sbin/ztcfg -vv  
Zaptel Configuration  
=====  
Channel map:  
Channel 01: FXO Kewlstart (Default) (Slaves: 01)  
Channel 04: FXS Kewlstart (Default) (Slaves: 04)  
2 channels configured.
```

如果通道加载成功，你会看到以上类似的输出。一个通常的错误是通道的信令与 zaptel.conf 文件中的描述是相反的。如果这种情况发生，你将会看到以下输出：

```
ZT_CHANCONFIG failed on channel 1: Invalid argument (22)  
Did you forget that FXS interfaces are configured with FXO signalling  
and that FXO interfaces use FXS signalling?
```

一旦通道被成功的配置，你就已经为 Asterisk 准备好硬件环境了。

3.2.7. zttool

3.3. Asterisk

3.3.1. 获取

为了获取 Asterisk，你必须从 Digium CVS 服务器上检出(check out)这个软件。当前有两类获取 Asterisk 的方式被经常使用：Release Candidate 2 (RC2) 和 HEAD（开发部门）。在 HEAD 分支中是最近的版本它包含了所有的最新的在源

码中实现的包。尽管不常见，但就象其它开发分支那样，从这个 CVS 服务器上可能会获取一个 Asterisk 的坏的版本。你最安全的打赌是使用 RC2 版本，那里被认为是稳定的版本。

- 从 RC2 分支中获取：

```
cd /usr/src/  
export CVSROOT=:pserver:anoncvs@cvs.digium.com:/usr/cvsroot  
cvs login  
密码是 anoncvs  
cvs checkout -r v1_0_rc_2 asterisk
```

- 从 HEAD 分支中获取：

```
cd /usr/src/  
export CVSROOT=:pserver:anoncvs@cvs.digium.com:/usr/cvsroot  
cvs login  
密码是 anoncvs  
cvs checkout asterisk
```

3.3.2. 编译

如果你之前曾经编译过软件，编译 Asterisk 看起来非常直接。在你从 CVS 服务器上获取 Asterisk 后，运行以下的命令编译并安装 Asterisk:

示例 3-6 编译 Asterisk

```
cd /usr/src/asterisk/  
make clean ; make install
```

3.3.3. 测试

在你开始 Asterisk 之前，你必须创建配置文件。尽管有许多的配置文件，这些文件都指定了 Asterisk 支持的不同的应用和通道，但为了启动系统只需要几个配置文件就可以了。为了测试 Asterisk 可以加载成功，我们将引导你通过最小化配置系列来实现安装。

首先，我们必须创建一个目录，所有的 Asterisk 的配置文件都将保存在那里。缺省情况下，Asterisk 会在 `/etc/asterisk/` 目录下寻找它的配置文件。现在让我们创建这个目录：

```
mkdir /etc/asterisk/
```

现在我们有一个位置来存放我们的配置文件了，我们需要创建这些配置文件。幸运的是，Asterisk 给我们附带了很多配置文件的示例。这些示例的配置文件是非常有用的，它可以用来查找一些信息，以及一个命令在它代表的配置文件中该如何使用。这些示例配置文件位于 `/usr/src/asterisk/configs/` 目录下。现在，让我们来创建这些需要的文件。

示例 3-7 创建配置文件

```
cd /usr/src/asterisk/configs/  
cp ./modem.conf.sample /etc/asterisk/modem.conf  
cp ./modules.conf.sample /etc/asterisk/modules.conf  
cp ./phone.conf.sample /etc/asterisk/phone.conf  
cp ./voicemail.conf.sample /etc/asterisk/voicemail.conf  
cp ./zapata.conf.sample /etc/asterisk/zapata.conf
```

使用这个最小的配置文件系列，我们应当可以成功的启动 Asterisk。我们可以通过以下命令来实现。

示例 3-8 : 启动 Asterisk

```
/usr/sbin/asterisk -cvvv
```

Asterisk 现在应当启动了，你会看到一个 `CLI*>` 提示。你会注意到在启动时会有几个错误说未发现几个不同的文件。这些文件现在还不在于本文档的范围内，我们会在未来的卷中讨论它。至此，Asterisk 还不具备所有的功能，但我们知道它已经启动成功了。剩下的就是系统的配置了。

4. 通道配置

4.1. 通道简介(introduction to channels)

通道是 Asterisk 可以用来创建和连接呼叫的到不同信令和传送路径的逻辑连接。它们可以是物理的(诸如一个模拟的 FXO 端口)或者是基于软件的(诸如一个 IAX 通道), 依赖于它们的属性。拨号方案用来定义规则, 采用这个规则, 让 Asterisk 来知道你希望怎么连接这个通道。显然, 在我们开始创建拨号方案前, 我们必须决定我们需要哪种类型的通道, 配置它们让系统可用。

通道可以有多种不同的格式呈现, 物理的电信电路(诸如 FXO、FXS、PRI、BRI), 基于软件的、网络附加的实体(诸如 SIP 和 IAX), 和那些专门的 Asterisk 为各种特殊需求提供的内部通道(诸如 Agent、Console、本地的非常精彩的 TDMoE)。Asterisk 把这些所有的通道都看作是一个连接点, 你可以把它们带入到拨号方案中。记得这一点是非常重要的: 即使通道具有非常不同的技术和连接能力, 但 Asterisk 允许你一样的对待它们。

Asterisk 是一个非常灵活和强大的 PBX 很大程度上就是因为它处理通道的方法。在许多老式的、私有的 PBX 中, 不同的通道具有完全不同的通信方法。用户端口、中继端口和 IP 端口是如此的不同以至于它们需要数年的经验才能给出怎么创造性的把它们连接在一起。事实上, 它可以被引起争论: 成为一个在特定的 PBX 上的专家就是最大可能的知晓它的一些限制, 并学习一些创新性的知识来突破这些限制。Asterisk 改变了所有这些。

只要可能, Asterisk 就允许你和任何其它通道类型同样的方式对待任何通道类型。无论你是在一个 FXS 端口上连接到一个模拟终端、连接 IAX 通道到 IAXTel 还是一个 SIP 电话, 对你而言, 可用的功能通常是一样的。由于 Asterisk

将允许你创造一些还不工作的东西，因此这在你的设计者那里赋予了极大的职责。但这同时也意味着 Asterisk 在这一点上比其它的 PBX 而言是非常灵活的。

在本文档中，我们将带领你创建四类不同的通道类型：FXO、FXS、IAX 和 SIP。我们选择这四类是因为它们是当前在用的最流行的通道类型。如果你没有从 Digium 获得开发工具，你可能不能存取 FXO 和 FXS 卡，在这种情况下，请阅读一下 FXO 和 FXS 章节，但是不要创建 `/etc/asterisk/Zapata.conf` 文件。

4.2. FXO 和 FXS

术语“FXO”和“FXS”最初起源于老式电话服务，称为 外部交换 (Foreign eXchange) (FX)。一个 FX 电路的最初目的是允许一个模拟的电话在远端连接到一个 PBX 上。一个 FX 电路有两个端点 (用户端点，就是连接电话的端点和局端点，连接 PBX 的端点)。理解 FXO 和 FXS 容易混淆的地方就是 FX 卡并不是以它们自身是什么来命名，而是以它们连接的是什么来命名。因此，一个 FXS 卡就是你可以连接一个电话终端的卡。由于是这种情况，你可以看到，为了做它的工作，一个 FXS 卡必须具备类似一个中心交换局的能力。相似的，一个 FXO 卡连接到一个中心局，这意味着它需要具备象一个电话的能力 (猫 (Modem) 是一个 FXO 设备的传统例子.)

4.2.1. FXS

FXS 通道提供了与你的电话公司提供的到大多数家庭和小企业的传统的模拟线路相同的接口。另外，FXS 通道可以正常提供：

拨号音

振铃电压

DTMF (按键音) 检测

消息等待 (Message waiting)

主叫线路识别 (Calling Line ID)

在 Asterisk 中，当你配置了一个 Digium FXS 卡(诸如 TDM 400P 带 FXS 子卡)，你就需要定义一些关联到一个 FXS 卡的参数。最重要的是，你需要记得一个 FXS 卡提供了中心局的相关信令。你需要配置 `/etc/asterisk/zapata.conf` 如下：

```
language=en  
context=default  
switchtype=national  
signalling=fxo_ks  
channel => 1
```

注意：在线路中，我们定义 `signalling=fxo_ks`。这描述了这个卡目前提供的功能。由于 FXS 卡执行一个中心局的功能(提供拨号音、振铃电压、消息等待等等)，我们指定我们希望它提供的信令是中心局(Office)，因此 FXO_KS 就是其信令。它是一个 FXS 卡，因为你可以将一个终端(Station)连接到其上。但它需要提供一个中心局的功能，因此信令是 FXO。

以上例子没有包含所有可能的选择，但可以为你设置一个示例的 FXS 通道。它是我们截止目前所做的全部。

不要忘了一个 FXS 卡提供中心局功能给连接它的设备。

4.2.2. FXO

一个 FXO 卡是一个可以连接一个中心局的卡。一个 Modem 是一个 FXO 卡的传统示例(事实上，如果你有一个 Digium 的老的 FXO 卡，X100P，它实际上就是一个 modem)。一个 FXO 设备必须能够：

- 产生 DTMF (按键音)*
- 检测拨号音*
- 检测振铃*
- 检测消息等待*
- 解释 Caller ID*
- 按照摘/挂机的条件产生信号给远端，又称为拍插簧。*

当配置一个 FX0 通道来替代一个 FXS 通道时，提供以上功能在设置上的最主要不同就是信令。

在你的/etc/asterisk/zapata.conf 文件中，我们将增加以下行：

```
signalling=fxs_ks  
channel => 4
```

于是，你的整个文件看起来会象下面这样：

```
language=en  
context=default  
switchtype=national  
signalling=fxo_ks  
channel => 1  
signalling=fxs_ks  
channel => 4
```

在第一个例子中，我们列出了通道 1 作为一个 FXS 通道。为了在同一个 TDM400P 卡上创建一个 FX0 通道，我们列出这个通道的所有设置，然后定义通道号。我们只需要将信令设置为 fxs_ks 来代替 fxo_ks。因为其它设置没有被改变 (signalling=fxs_ks 代替了 signalling 以前的值。)，它们仍然是一样的。这意味着当通道 1 具有 language=en, context=default, switchtype=national, 和 signalling=fxo_ks 时，通道 4 会具有 language=en, context=default, switchtype=national 和 signalling=fxs_ks。

我们现在有了可以工作的 zapata.conf 示例文件，它具备一个 FXS 通道(1) 和一个 FX0 通道(4)可以让我们使用。通道 4 可以被连接到一个象你的电话公司提供的那样的模拟电路上。你可以直接插一个模拟的电话机到通道 1 上。

4.3. IAX

IAX (Inter-Asterisk eXchange)协议是一个基于 IP 的媒体传输协议。为了通过 IP 在两个人之间创建会话，你可以使用 IAX 作为传诵音频的方法。为了创建一个 IAX 通道，你需要在你的 iax.conf 文件中创建 IAX 通信。

注意：IAX 被 Asterisk 开发社区读作“eeks”，但如果你读作“AYE-AY-EX”，也没有人会介意。

● 通用设置

首先，我们需要创建我们的 IAX 全局使用的设置：

```
[general]
port=4569 ; What port to bind to (4569 is the default for IAX2)
bindaddr=0.0.0.0 ; Which IP address on your server to bind to (if
; you have multiple NICs on your sever, you can
; restrict IAX2 to only one of them. 0.0.0.0 will
; allow it to work on all NICs in your system.
deny=all ; You want to disallow the use of all audio
; codecs to ensure that
; your system won't tell the far end that it can
; support just any codec. Then, you specifically ALLOW
; the codecs that your system supports.
allow=ulaw ; The North American standard companding for G.711
allow=alaw ; The rest of the world's companding standard for G.711
allow=gsm ; A compressed codec, popular with Asterisk
```

以上例子是非常小的仅仅设置了基本的设置来监听连接并创建他们。

Apache 用同样的方法在 80 端口上监听 http 请求，Asterisk 将在 4569 端口上监听 IAX 请求。

● 定义 IAX 通道

现在，我们已经为我们的 IAX 接口到外部世界定义了一些全局参数，我们可以创建我们的 IAX 通道。IAX 通道是非常灵活的，已经成功的被用来连接多种终端。Digium 制造了一类基于 IAX 的设备俗称为 IAXy，它提供一个 FXS 接口在 IAX 通道的末端支持一个模拟电话。IAX 也是一个被 IAXTel 网络使用的协议，这个网络是我们的例子中将会把你连接到的那个地方。

尽管 IAX 不是一个 RFC 标准的协议，但它获得了极大的重视。许多学者预测 IAX 将替代 SIP。

注意：在创建本文件前，你需要获取一个 IAXTel 帐号来完成配置。

```
[general]
port=4569 ; What port to use
bindaddr=0.0.0.0 ; What IP address to bind to
allow=all ; Allow the use of all audio codecs
register => username:secret@iaxtel.com ;replace username:secret with your
credentials, [iaxtel-out]
type=peer ; Allow connections out
username=username ; TYour IAXTel username
secret=password ; Your secret password
deny=0.0.0.0/0.0.0.0 ; Not just anyone can be IAXTel
permit=216.207.245.47/255.255.255.255 ; This is a server at IAXTel
permit=69.73.19.178 ; This is a server at IAXTel
[iaxtel-in]
type=user ; Allow connections to come in
context=default ; Route calls to this context
; in the dialplan
username=username ; The IAXTel username
secret=password ; The secret password
deny=0.0.0.0/0.0.0.0 ; Not just anyone can be IAXTel
permit=216.207.245.47/255.255.255.255 ; This is a server at IAXTel
permit=69.73.19.178 ; This is a server at IAXTel
```

在以上例子中，你会注意到我们有两个入口来与 IAXTel 服务通信。IAXTel 是一个免费的 VoIP 呼叫服务它被用来作为 Asterisk 的测试，最好采用 IAX 作为通用的通信系统。

实际上，你会注意到第一个变化发生在通用段中。这一行告诉 IAXTel 我们在这儿因此，IAX 用户可以路由到你的 Asterisk 服务器。这就象连接到你的 IM(AOL 即时消息，Yahoo、MSN 等等)那样，无论你在哪里登陆，那些给你发送的消息你总会得到它。

我们有两种不同类型的到 IAXTel 的连接：Peer 和 user。这允许我们来决定：一个呼入呼叫可以从一个服务器中进来，或者一个呼出呼叫可以从另一个服务器出去。当你正在处理一个主要的 Asterisk 服务器网络且正在使用 IAX 作为连接那个服务器的中继时，这一点是非常有用的。

4.4. SIP

会话初始协议(SIP:Session Initiation Protocol)正迅速成为最广泛被支持的 VoIP 协议。就象 IAX 那样, SIP 相当容易配置。尽管我们对协议已经有一些认识了。但请记住即使你的通道已经可能被正确的配置了, 但 SIP 不能很好的处理 NAT。这可能是最头疼的地方的。为了配置 SIP, 你将需要创建一个 sip.conf 文件。

● 通用设置

第一个需要做的事就是创建通用设置。与 IAX 非常相似, 通用设置允许你给定一些所有 sip 连接都将使用到的一些设置。

```
[general]  
port = 5060 ; Port to bind to  
bindaddr = 10.78.1.90 ; Address to bind to  
context = default ; Default for incoming calls  
srvlookup=yes ; Enable SRV lookups on outbound calls  
dtmfmode=inband  
allow=all ; Allow all codecs
```

就象你看到的那样, 这个设置非常类似于 IAX。我们有一个端口、地址、context、以及 allow。srvlookup 设置是一个用来查找主机名称的方法。如果设置为 yes, DNS 查找将发生在 SRV 记录上以替代 A 记录来适应负载均衡。dtmfmode 设置被用来决定 Asterisk 应该怎么监听按键音, 例如某人正在拨叫一个分机。

● 定义 SIP 通道

为了使用 SIP 通道, 客户端应该被给定了通过 SIP 访问 Asterisk 的授权许可。Asterisk 也可以被用来作为一个客户端(例如 当使用通过 SIP 的 FWD 电话服务, client 端的设置必须被设置。)

```
[general]
port = 5060 ; Port to bind to
bindaddr = 10.78.1.90 ; Address to bind to
context = default ; Default for incoming calls
srvlookup=yes ; Enable SRV lookups on outbound calls
dtmfmode=inband
allow=all ; Allow all codecs
register => FWDNumber:secretpassword@fwd.pulver.com/EXTEN
[fwd.pulver.com]
type=user
username=FWDNumber
secret=secretpassword
host=fwd.pulver.com
nat=yes
canreinvite=no
[fwd.pulver.com]
type=peer
host=fwd.pulver.com
context=default
nat=yes
canreinvite=no
```

FWD 代表 Free World Dialup, 一个免费的 VoIP 到 VoIP 服务, 你可以在 <http://www.freeworlddialup.com> 发现它。在 FWD 和 IAXTel 之间通信是被允许的, 在它们各自的 web 站点上有相应的指示会告诉你怎么在这些服务间进行通信。

重复一下, 你会看到我们在 general 段中有一个 register 行, 它让服务提供商知道我们是一个路由到 FWD 号码(FWDNumber)的呼叫的正确的客户端。你也会注意到例子中有一个新的 context。fwd.pulver.com context 允许我们呼出到 FWD, 同时从 FWD 来的呼叫将会被 Asterisk 来处理。

第一个入口是使用 FWD 服务作为客户端时呼出呼叫的必须的信息。它几乎与 IAX 下面那个 type=user 的段是一致的。第二个入口是为了呼入呼叫的鉴权, 确保我们没有获得一个从其它来源来的伪装呼叫。

4.5. 其它的通道协议

Asterisk 是现存的最兼容标准的 PBX。它兼容每一个重要的电话标准，在业界无与伦比。

H. 323

ISDN

MGCP

SCCP (Skinny)

VoFR

Bluetooth

5. 拨号方案(Dialplans)简要介绍

拨号方案是 Asterisk 系统的核心，它定义了 Asterisk 应当如何处理呼叫。它有一系列的指令或 Asterisk 可以跟从的步骤组成。它们由从通道或应用接收到的数字(或字符)来触发。为了成功的创建你自己的 Asterisk 系统，你必须需要理解拨号方案。

本章将解释拨号方案是怎么以一步步的方式工作的，并给一些技能来创建你自己的拨号方案。本章中给出的例子的设计是与其它知识关联的，因此，如果有些地方没有理解，请直接返回去并重读相关的章节。当然，本章没有试图去给出拨号方案能做的所有可能情况的复杂考虑。我们的目的是覆盖所有的基础知识。

5.1. 创建拨号方案简介

最主要的拨号方案位于文件/etc/asterisk/extensions.conf 中。

注意：你的 extensions.conf 文件的位置可能会有不同，这依赖于你是怎么安装 Asterisk 的。

本文件由四个主要的部分组成：contexts、extensions、priorities 和 applications。在本节中，我们将覆盖它们的每一部分，解释他们怎么在一起工作并创建一个拨号方案。当我们结束时，你将会创建一个基本的，功能性的拨号方案。

如果你已经安装了 Asterisk 的例子文件，你可能会有一个已经存在的 extensions.conf 文件。我们现在不会去直接修改那个文件，相反，我们建议你从草稿开始创建你自己的 extensions.conf 文件。这会极大的帮助你学习，让你更好的理解 dialplan 文件中的每一部分都扮演什么角色。

在例子 `extensions.conf` 文件中保留了一些奇怪的资源以学习 `dialplan` 更多的不同和多样的精妙之处。我们建议你把这个文件变更为别的名字，例如：`extensions.conf.sample`。这样，以后你就可以总是能参考它。我们试着这样做：

```
#mv /etc/asterisk/extensions.conf /etc/asterisk/extensions.conf.sample
```

5.1.1. Contexts

在拨号方案中，Contexts 扮演了一个组织的角色。Contexts 也可以来定义范围。你可以认为 contexts 是一种分割 `dialplan` 不同部分的方法。作为一个简单的示例，contexts 可以被用来创建一个自动话务员菜单系统：此时，在 `[SalesDepartment]context` 内的菜单选项”1”所做的事情完全不同于在 `[ServiceDepartment]context` 内的菜单选项”1”。比可以把不同的用户放在不同的 contexts 内，这样，当某人从 `[ABCWidgetsInc]` 他们的组中拨号“0”时，他们将到达他们所属的接待员那里，而当 `[KeepItSimpleStoopid]` 的总裁拨”0”时，他将直接到 PSTN 话务员那里。如果我们想为不同的公司提供不同的接待目的地时，这将非常容易。而所有这些都共享在同一个 Asterisk 服务器上。Asterisk 处理的任何呼叫都将开始于一个特定的 context。定义在这个 context 上的指令将决定对该呼叫将发生什么事情。

注意：contexts 经常被用来创建语音菜单，这个菜单给主叫一系列的分机，通过在一个 DTMF 话机上按键来选择。这个功能是作为一个自动话务员最常用的参考。在本章中，我们将覆盖自动话务员更多的东西。

Contexts 通过位于方括号内的他们的名字来表示。例如，如果我们为呼入呼叫创建一个 context，我们会如下定义：

```
[incoming]
```

在一个 context 定义后面放置的所有指令是 context 的一部分。为了开始下一个 context，只需简单地定义下一个 context 即可。

在 extensions.conf 文件的最开始，有一个特殊的 context 称为 [globals]，在这个 context 中，设置和变量可以被定义，他们可以在你的整个 dialplan 中使用。我们现在不会去使用 [globals] context 的全部能力，但你应该知道它为什么存在。

5.1.2. Extensions (分机)

在每一个 context 内，我们将会定义一个或多个 extensions (分机)。在 Asterisk 中，一个分机是一个字符串，它可以被一个事件所触发。一个简单的示例看起来如下：

```
exten=>555, 1, Dial (Zap/1, 20)  
exten=>555, 2, Voicemail (u555)
```

“exten=>” 告诉拨号方案它后面看到的将是一个命令。

“555” 是实际接收到的数字（或者就是主叫所拨的号码，或者是做拨的分机号）。

“1” 或者 “2” 代表了优先级，它用来决定对那个分机的命令以何种顺序执行。在我们的例子中，“555” 将会对 Zap1 的通道 1 振铃 20 秒。然后携带一个不可用的消息连接到 voicemail 邮箱 555 中。

分机决定了呼叫的流程。尽管分机可以被用来指定传统意义上的电话分机（例如：呼叫 John，请拨分机 153），但他们在 Asterisk 中可以被表示更多的意义。在我们的 extensions.conf 文件中，一个分机被通过一个单词 “exten” 后跟一个箭头的形式来表示，箭头由一个等号符和一个大于符号组成。如下：

```
exten=>
```

在它后面跟了分机的号码(或者名字), 一个逗号, 优先级, 另一个逗号, 最后是在通道上呼叫的应用。我们将在下面解释优先级和应用, 但首先, 让我们先解释完语法。语法看起来如下:

```
[context-name]  
exten=>extension, priority, application
```

5.1.3. Priorities(优先级)

优先级是在分机中为每一个 extension 进行的编号的步骤。每一个优先级调用一个特定的应用。典型地, 这些优先级编号简单地从 1 开始并为 context 中的每一行连续的递增。优先级好可以不总是连续的, 但我们以后将会提醒你这一点。现在, 你只需要记得对每一个 extension, Asterisk 顺序地运行每个优先级。

5.1.4. Applications(应用)

应用在一个语音通道上执行特定的动作, 例如播放语音, 接收按键输入, 或者挂断一个呼叫。可选的被叫参数可以被传递给应用以影响到他们怎么去执行他们的动作。(为了查阅内置在 Asterisk 中的可能的应用列表, 你可以在 Asterisk 的命令行接口上输入 show applications) 由于我们创建了下面的第一个 dialplan, 这样你就可以学习使用应用来提升自己。

5.2. 一个简单的示例

现在, 我们准备创建我们的第一个 extensions.conf 文件。由于这是我们的第一步, 我们会从一个非常简单的例子开始。我们假定在这个例子中, 所有需要 Asterisk 做的就是应答呼叫, 播放一个声音文件说声” Goodbye”, 然后挂机。我们将使用这个简单的例子来指出创建一个 dialplan 的基础。

5.2.1. 特殊的's'分机

在我们开始深入到我们的例子之前，我们需要介绍一个特殊的 extension 称为's'，它代表了”start”，缺省情况下，呼叫将会在一个 context 中以's' extension 来开始。(你可以认为's' 是一个可以自动执行的 extension)在我们的例子中，我们将创建一个具有's' extension 的拨号方案。

5.2.2. Answer()、Playback()和 Hangup()应用

如果我们要应答呼叫，播放一个声音文件，然后挂机。我们将更好的学习怎么做到这一点。Answer()应用被用来应答一个正在振铃的通道。这将初始创建一个呼叫以使我们能够执行其它一些功能。较少的一部分应用不需要请求我们首先 Answer()通道，但是在做任何事情之前适当地 Answer()通道是一个非常好的习惯。

Playback()应用被用来在一个通道上播放一个先前录制好的声音文件。当使用 Playback()应用时，从用户来的输入被简单的不予理会。随着 Asterisk 附帶了许多专业化的录音声音文件，我们通常可以在/var/lib/asterisk/sounds/下发现它们。Playback()可以指定文件名(不包括文件扩展名)来作为参数。例如：Playback(filename) 将会播放位于缺省声音目录下的 filename.gsm 声音文件。

Hangup()应用的作用就象它的名称暗示的那样。它挂掉一个活动的通道。当你在 context 的结尾处，不再需要主叫连接到系统上时，你可以使用它来释放一个主叫。

5.2.3. 我们的第一个 dialplan

现在，我们正准备开始我们的第一个示例 dialplan。请注意每个优先级调用应用的方法。注意在这个例子中，我们仅只有一个 extension。在后面的例子中，我们将增加其它的 extensions。并演示怎么把它们从一个分机移动到另一个分机。

注意：以下例子并不意味着能够完整的工作和可用。我们只是简单的使用他们来解释 dialplan 是如何工作的。为了这些例子能够工作，你应当准备配置一些 Zap 通道(例如：使用来自于 Digium 的 Devkit)，配置这些通道使它们的呼入呼叫能够进入到[incoming] context 中。

在演示了我们的第一个例子后，我们将解释每一个步骤。

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

当一个呼叫被送到[incoming] context 后，它将首先来到' s' extension。象我们之前所学习的那样，呼叫通常开始于' s' extension。在这个 context 中，我们有三个优先级，编号为 1, 2, 和 3。每个优先级调用一个特定的应用。让我们再近一些来看看这三个优先级。

在我们的第一个有限级上，调用了 Answer()应用。此时，Asterisk 就会控制线路创建一个呼叫。在应答这个线路后，Asterisk 将进入到下一个优先级。在我们的第二个优先级中，我们调用 Playback()应用。这将播放一个由文件名指定的声音文件。在我们的例子中，我们将播放 goodbye。主叫将会听到一个声音说“goodbye”。注意这里没有文件扩展名。Asterisk 将会自动地决定声音文件的扩展名。在我们的第三个也就是最后一个优先级行中，我们调用 Hangup()应用，然后终止呼叫。

5.3. 一个更有用的例子

现在让我们通过一个简单的示例来继续深入，我们会创建并学习 Background() 和 Goto() 应用。这两个应用将使我们创建的 dialplan 具有更多的功能。

交互式 Asterisk 系统的关键就是 Background() 应用。这个应用给了你播放一个录制的声音文件的能力，但当主叫按键时，它可以打断播放并进入到主叫所拨号码对应的那个 extension。

另外一个非常有用的应用称为 GoTo()。就象它的名称所暗示的，它从当前的 context、extension 和 priority 跳转到指定的 context、extension 和 priority。Goto() 应用使程序化地从 dialplan 的两个不同部分之间移动变的容易。Goto() 应用的语法让我们传递目的 context、extension、和 priority 作为参数给应用，如下：

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

在这个例子中，让我们假定我们被请求为本地的电影院来创建一个交互式系统，在这里，主叫可以拨入并收听预先录制的影片列表。为了使这个例子简单，我们说这个电影院仅有两个放映厅。

我们将假定我们三个预录制的声音文件。首先，被称为 current-moves.gsm 的文件，内容为：Welcome to our movie theater. To hear what' s playing on screen one, press one. To hear what' s playing on screen two, press two. 欢迎来到我们的电影院，听一下我们的一号放映厅播放的内容请按 1 键，听我们 2 号放映厅播放的内容，请按 2 键。这两个声音文件，分别命名为 movie1.gsm 和 movie2.gsm 。告诉主叫关于正在那个放映厅中播放的影片的信息。

注意：我们将在本章的后面介绍录音声音文件。

```
[incoming]  
exten => s, 1, Answer()  
exten => s, 2, Background(current-movies)  
exten => s, 3, Hangup()  
exten => 1, 1, Playback(movie1)  
exten => 1, 2, Goto(incoming, s, 1)  
exten => 2, 1, Playback(movie2)  
exten => 2, 2, Goto(incoming, s, 1)
```

让我们来一步步看一下这个例子。当主叫进入系统时，Asterisk 自动执行 's' extension。开始于优先级 1。你可能注意到 's' extension 看起来和我们的第一个例子几乎一致。所不同的是使用 Background() 应用代替了 Playback()。就象我们前面解释的，Background() 应用允许我们在播放声音文件时，接受从主叫来的输入数字。当 current-movies.gsm 文件被播放给主叫时，用户按 1 键，Asterisk 将在当前 context 下查找匹配的分机来匹配它。当 Asterisk 发现了 "1" extension 时，它将执行那个 extension 下的所有优先级。

现在用户按 1 键，Asterisk 可以执行 extension 1 的那两个优先级，对应于以上例子中的第四行和第五行。Extension 1 的第一个优先级使用 Playback() 应用来播放放映厅 1 的影片的细节信息。在文件完成播放后，它将执行第二个优先级，它会调用 Goto() 应用。

Goto() 允许我们发送主叫到我们的 dialplan 中的任何位置。Goto() 的格式为 Goto(context, extension, priority)。在我们的例子中为：

exten=>1, 2, Goto(incoming, s, 1) 。我们将发送用户返回到我们的 'incoming' context 内的 's' extension 的第一个优先级。

如果用户在 background() 应用完成播放文件之前没有按键，我们的 's' extension 的第三个优先级将会被执行，挂断用户。这可能不是处理呼入呼叫的最好方法，但是我们现在这么做是由于我们在学习一些怎么控制呼叫流程的基础。考虑我们怎么围绕着拨号方案来移动用户。

5.3.1. 使用 Dial 应用的主叫通道

我们现在增加我们的电影院的例子来演示一下 Dial() 应用的使用。如果主叫在播放语音期间按 0 键，它将振铃票务办公室。我们假定连接票务办公室的通道已准备好。

```
[incoming]
exten => s, 1, Answer()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(Zap/1)
```

如果你把这个例子和前一个相比，相当明显，我们所做的仅仅是在我们的 [incoming] context 下增加了另外的 extension。我们增加了 extension 0，用它来执行 Dial 应用。Dial() 应用允许我们使用特殊格式[技术]/[资源]来呼叫一个指定的通道。在本例中，我们呼叫我们的 Zap1 接口的第一个通道(通过 Zap/1 来识别)。它最可能连接到一个模拟电话上。当主叫在语音菜单时按 0 键，电话将振铃。如果话务员应答这个呼叫，他或她将会被连接到主叫。

截止本章的现在为止，你应当懂得了几类应用的使用，例如：Answer()、Playback()、Background()、Hangup()、Goto() 和基本的 Dial()。如果你对它们是如何工作的仍不清晰，请返回去重读一下本节。为了更全面的获取一些我们将要探讨的概念，对这些应用的基本理解是最基础的。

仅只理解 extensions、优先级和应用的基本概念就可以简单的创建一些基本的 dialplan。在后面的几节中，我们将增加这些基础以使我们可以创建更强大的 dialplan。

5.3.2. 增加附加功能

5.3.2.1. 在内部用户间处理呼叫

截止目前，在我们的例子中，我们还把我们自己限制在一个单一的 context 内。这可能是不公平的：假设大多数的安装都会具有多个 context。Context 使用的最大场景是给外部主叫和内部分机 extensions 来的呼叫以不同的用户体验。除了提供不同的用户体验，contexts 也可以用来作为一种分割不同类型主叫权限的机制。在这种机制下，可以允许长途呼叫或者仅允许呼叫特定的分机 extensions。在我们的下一个例子中，我们将创建一个简单的 dialplan 它具有两个内部电话分机，并设置了这两个分机之间互相呼叫的能力。为完成这一点，让我们创建一个新的 context 称为 [internal]。

注意：就象前一个例子，我们假定通道已经被配置好。我们也假定从这些通道起源的任何呼叫将开始于 [internal] context。

```
[incoming]
exten => s, 1, Answer()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(Zap/1)
[internal]
exten => 1001, 1, Dial(SIP/1001)
exten => 1002, 1, Dial(SIP/1002)
```

上面这个例子显示了呼入到 [incoming] context 的呼叫可以从我们的菜单选项选择 0 来拨叫通道 Zap/1。然而，我们将标准化内部分机为 4 个数字号码。就象我们已经创建的那样，在通道 SIP/1 通道上的主叫可以拨号 1002 而呼叫通道 SIP/2。而 SIP/2 可以拨号 1001 而呼叫 SIP/1。

5.3.2.2. 变量(Variables)

变量是拨号方案中非常有用的工具。他们可以被用来减少输入、增加清晰度或者增加一个拨号方案的附加逻辑。考虑把变量看作是一个容器。当我们引用一个变量时，我们真正需要的是那个变量中包含的值。例如，变量\${JOHN}可以被指定为分配给那个叫 John 的人的通道的名称。在我们的拨号方案中，我们可以用\${JOHN}来引用通道，Asterisk 将会自动的用分配给这个变量的值来代替它。引用变量由一个美元符，一个开放的” { “号，变量的名称，和一个结束的” } ”号 来指定。

共有三种类型的变量，命名的全局变量、通道变量和环境变量。正象他们的名字所暗示的那样，全局变量可以在所有的 contexts 中的所有 extensions 使用，而通道变量仅只在当前呼叫的过程中可用。环境变量是一个在 Asterisk 中存取 Unix 环境变量的方法。

注意：术语参数和变量通常被用来互换地代表同一个含义。为了我们的目的，我们将使两者能够区别开来。参数是传递数值到嵌入应用中告诉他们怎么做。变量是一个容器它可以被分配一个特定的值。他们都可以被用户使用，尤其会被 Asterisk 自身频繁的使用。

- **全局变量**

全局变量使用形式 VARIABLE_NAME=value 来指定，它必须被放置在 [globals] context 内。全局变量是非常有用的，它允许我们在我们的拨号方案中使用它使该方案更易读。他们也可以使更新一个拨号方案的任务非常简单。全局变量也可以通过使用 SetGlobalVar() 应用来定义。

- **通道变量**

一个通道变量是一个与一个特定呼叫相关联的变量(例如: 主叫识别的号码 CallerID)。不象全局变量, 通道变量仅只定义在呼叫的持续过程中。有许多预定义好的通道变量可以在拨号方案中使用, 他们被列表在 `/usr/src/asterisk/doc/README.variables` 文件中。通道变量也可以通过使用 `SetVar()` 应用来设置。

- 环境变量

环境变量允许我们从 Asterisk 中存取 Unix 环境变量。他们以形式 `${ENV(foo)}` 来引用, 其中, `foo` 就是你希望引用的环境变量。

5.3.2.3. 增加变量

让我们扩展我们的电影院例程: 指定一些通道名字给全局变量。

```
[globals]
RECEPTIONIST=Zap/1
JOHN=SIP/1001
MARY=SIP/1002
[incoming]
exten => s, 1, Answer()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(${RECEPTIONIST})
[internal]
exten => 1001, 1, Dial(${JOHN})
exten => 1002, 1, Dial(${MARY})
```

在我们的应用中, 我们用一个变量名代替了直接将通道名作为参数。在我们想更改我们的变量相关的通道时, 这将使我们更新我们的 dialplan 更容易。我们的拨号方案也会变得易于阅读。

5.3.2.4. 一个有用的调试方法

NoOp() 应用 (No-Operation) 对调试是非常有用的。它可以用来回显信息到 Asterisk 的 console 上。例如, Zap 通道不会在呼入呼叫时打印主叫 ID 的信息, 但是我们可以如下做:

```
exten=>s, 1, Answer ()
exten=>s, 2, NoOp(${CALLERID})
```

使用预定义的通道变量 \${CALLERID}, 主叫标识信息将会被输出到 Asterisk 的 Console 上。

5.4. 呼叫流程

5.4.1. 模式匹配

Extensions 不仅仅简单地限制在号码上, 我们可以匹配号码的模式来控制我们的呼叫流程。为了做到这一点, 我们启用一个以一个下划线开始的 extension 号码。在进行模式匹配时, Asterisk 识别特定的符号解释为特殊的意义。这些符号包括:

- X* : 匹配任何从 0-9 的数字
- Z* : 匹配任何从 1-9 的数字
- N* : 匹配任何从 2-9 的数字
- [1237-9]* : 匹配任何在方括号内的数字或字母 (例如: 1, 2, 3, 7, 8, 9)
- .* : 通配符, 匹配一个或多个符号。

以下示例显示了你怎么匹配一个从 0000 到 9999 的 4 位数字的分机号码。注意在 XXXX 前面的下划线_告诉 Asterisk 我们是一个模式匹配, 而不要直接按字面去解释 extension。

```
exten=>_XXXX, 1, NoOp ()
```

5.4.2. 利用\${EXTEN} 通道变量

\${EXTEN}通道变量包含了所拨叫的分机号码。我们使用\${EXTEN}变量和模式匹配一起可以帮助减少我们请求拨号的行数。这允许我们简单地使用一个呼叫模式去匹配一个号码，而然后却使用一个特定的号码去拨号。下例示意了一个简单的模式匹配和\${EXTEN}的使用。

```
[globals]
RECEPTIONIST=Zap/1 ; Assign ' Zap/1' (FXS) to the global variable
${RECEPTIONIST}
JOHN=SIP/1001 ; Assign ' SIP/1001' to the global variable ${JOHN}
MARY=SIP/1002 ; Assign ' SIP/1002' to the global variable ${MARY}
[incoming]
exten => s, 1, Answer ()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup ()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(${RECEPTIONIST})
[internal]
exten => _1XXX, 1, Dial(SIP/${EXTEN}) ; replace separate Dial () statements with
a single pattern matching statement
```

注意：不要用连字符“-”来分割你的模式匹配，这样可能会导致错误。

Outgoing context 包含了一个简单的模式匹配的示例它允许我们呼叫一个以 1 起始的 4 位号码。与其它 extension 行的格式相同，紧跟着模式匹配的后面是一个优先级和应用。我们使用\${EXTEN}通道变量，来代替显式地传递给 Dial()应用一个我们希望的拨号号码。例如，如果我们拨了 1001，那么我们的模式就会匹配该号码。\${EXTEN}变量仍然包含了拨叫的号码，被传递给 Dial()应用。我们已经使用单行有效地替换了两个分离的 Dial()语句。所有在 1000->1999 范围的分机号码都会被这一行匹配。我们不需要在我们的 PBX 中为每一个分机变量增加一个分离的行。

现在，我们理解了模式匹配和\${EXTEN}通道变量是如何工作的，让我们创建一个呼出的 context 以允许呼出呼叫。重复一下，我们假定我们的呼出通道已经被创建了，命名为 Zap/2。

```
[globals]
RECEPTIONIST=Zap/1
JOHN=SIP/1001
MARY=SIP/1002
LOCALTRUNK=Zap/2
[incoming]
exten => s, 1, Answer()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(${RECEPTIONIST})
[internal]
exten => _1XXX, 1, Dial(SIP/${EXTEN}) ; replace separate Dial() statements with
a single pattern matching statement
[outgoing]
ignorepat => 9
exten => _9NXXNXXXXXX, 1, Dial(${LOCALTRUNK}/${EXTEN:1})
exten => _9NXXNXXXXXX, 2, Playback(invalid)
exten => _9NXXNXXXXXX, 3, Hangup
```

注意：ignorepat 语句允许我们甚至用户拨了 9 来存取外线后还保持给用户的拨号音。没有它，在用户拨 9 后将不会有拨号音。

我们现在创建了一个新的 context 称为” outgoing”，我们可以使用它来接入到外部的电话线。

我们这个新的 context 的第一行包含了一个新的命令称为 ignorepat 它代表 ignore 模式。它允许我们创建我们的 outgoing context：用户在拨叫他们的外部电话号码前需要先拨 9。一旦用户按下了 9，拨号音将继续播放给主叫。没有它，在用户拨号接入外线时将不再有拨号音。

我们的下一行包含了 Dial() 应用, 它通过我们的 \${LOCALTRUNK} 接口拨号到变量 \${EXTEN} 所包含的那个号码。 \${EXTEN} 变量包含一个丢弃了首个数字的我们所拨的号码, 这是由于我们不想把数字 9 和我们的电话号码一起使用。例如, 如果我们想拨电话号码 123-555-1234, 那么我们拨号 91235551234。如果没有 \${EXTEN:1}, 那么整个串就会被传递给 Dial() 应用, 这会造成错误(或者, 拨了一个错误的号码)。 \${EXTEN:1} 将仅使 1235551234 部分被传递, 而遗留了号码 9。

如果 Dial() 应用不能成功的完成, 那么下一个优先级将被执行。这将播放 invalid 文件给用户, 让他们知道他们的呼叫未能完成。在这个文件被播放后, 通道被挂机释放。

你可能还会注意到: 由于没有地方访问它, 我们没有办法测试这个 context, 我们必须利用 include 语句, 这个语句我们将在下一节进行解释。

5.4.3. 使用 Includes 连接 Contexts

Asterisk 给了我们在一个 context 下使用另一个 context 的能力。限制和授权对 PBX 不同部分的存取最通常被使用的。使用 include 语句, 我们可以控制谁可以存取长途业务。

Include 语句具有如下形式:

```
include=>context
```

当我们在我们现在的 context 内包括其它的 contexts 时, 我们必须要知道我们包括他们的顺序。 Asterisk 将首先试图在当前 context 下匹配 extension。 如果不成功, 它将试着在第一个被包括的 context 内匹配, 然后按照顺序向下查找。

我们现在已经创建的 dialplan 有一个 outgoing context, 但它位于我们无法存取的位置。 为了给我们 internal 的分机存取 outgoing context 的能力, 我们需要包括它。

```

[globals]
RECEPTIONIST=Zap/1 ; Assign 'Zap/1' (FXS) to the global variable
${RECEPTIONIST}
JOHN=SIP/1001 ; Assign 'SIP/1001' to the global variable ${JOHN}
MARY=SIP/1002 ; Assign 'SIP/1002' to the global variable ${MARY}
LOCALTRUNK=Zap/2 ; Assign 'Zap/2' (FXO) to the global variable ${LOCALTRUNK}

[incoming]
exten=>s, 1, Answer()
exten => s, 2, Background(current-movies)
exten => s, 3, Hangup()
exten => 1, 1, Playback(movie1)
exten => 1, 2, Goto(incoming, s, 1)
exten => 2, 1, Playback(movie2)
exten => 2, 2, Goto(incoming, s, 1)
exten => 0, 1, Dial(${RECEPTIONIST})

[internal]
exten => _1XXX, 1, Dial(SIP/${EXTEN}); replace separate Dial() statements with
a single pattern matching statement
include => outgoing ; include the outgoing context

[outgoing]
ignorepat => 9
exten => _9NXXNXXXXXX, 1, Dial(${LOCALTRUNK}/${EXTEN:1})
exten => _9NXXNXXXXXX, 2, Playback(invalid)
exten => _9NXXNXXXXXX, 3, Hangup

```

在我们以上的拨号方案例子中，我们增加了一个单行到 internal context 中把 outgoing context 包括了进去。这将给我们的内部用户通过拨 9 加上 10 位电话号码来呼叫外线的能力。现在，我们已经限制我们的用户仅能拨叫本地号码，但我们可以通过创建一个为长途呼叫服务的 context 并在我们的 internal context 中包括它来实现对这个例子的扩展以允许长途呼叫。

我们可以看到我们已经限制了呼出呼叫仅只到我们的内部分机中，没有到达 incoming context 的呼叫将会存取我们的 outgoing context。

5.4.4. 一些其它的特殊 extensions

's' : *start* 起始
'i' : *invalid* 无效输入
't' : *timeout* 超时
'h' : *hangup* 挂机
'T' : *Absolute Timeout* 绝对超时

start extension 用来为大多数没有其它已知信息的呼叫做初始化。

invalid 是当 Asterisk 已经判定从呼叫来的输入对当前 context 是无效的时 所触发的 extension。你可能希望播放一个提示来解释分机号无效，然后返回到包含了菜单提示的那个 extension。

timeout 是当一个用户正在被提供一个菜单选择但他们没有响应。在 *timeout* extension 中，你将会决定你是重复你的菜单还是简单的挂起呼叫，释放线路。

Hangup 是当检测到挂机时，呼叫继续处理的地方，或者是你可以在你希望挂机的呼叫上继续发送信息的地方。

警告：

当前在使用 *'h'* extension 时，有几个问题需要注意。尤其是，关于呼叫的变量将会丢失，同时，伴随通道的信息也不存在了。

Absolute Timeout 被用在当一个呼叫超过了 *Absolute Timeout* 变量所设置的那个时限而被终止时。了解这种情况不同于正常的超时。它用来警告用户他们已经超过了一些允许的限制。或者它可以被用来请求某人来在之后回呼，如果他们等在一个队列中太长的时间。最起码，它应当通知主叫他们将要被断开以便免得留给他们一个印象让他们认为是被故意断开的。

5.5. 创建提示语(Creating prompts)

随着 Asterisk 已经有许多的语音提示，我们可以用来创建我们自己的拨号方案。包含所有可能的提示和菜单是不可能的。Asterisk 包含了一个非常便利的应用它可以被我们用来重新录制我们自己的提示。很明显，这个应用称为 Record()。

5.5.1. Record()应用的使用

Record() 应用可以被用来在一个通道上录制音频流。我们可以利用这个应用去录制为了在我们的拨号方案中使用的客户化的语音提示。

让我们创建一个新的 context 它将允许我们为我们例子中的电影院拨号方案更改语音提示。通过按键*1 和*2，我们会重新分别为放映厅 1 和 2 录制语音提示。

示例 5-1：为客户化语音提示创建 context

```
[prompts]
exten => *1, 1, Answer()
exten => *1, 2, Record(movie1:gsm)
exten => *1, 3, Playback(movie1)
exten => *1, 4, Hangup()
exten => *2, 1, Answer()
exten => *2, 2, Record(movie2:gsm)
exten => *2, 3, Playback(movie2)
exten => *2, 4, Hangup()
```

我们会通过在我们的 internal context 中包含它而实现对该 context 的访问。我们的 internal context 看起来象下面那样：

```
[internal]
exten => _1XXX, 1, Dial(SIP/${EXTEN})
; replace separate Dial() statements with a single pattern matching statement
include => outgoing
; allow internal extens to make outgoing calls
include => prompts
; allow internal extens to record prompts
```

通过在我们的 internal context 中包含 prompts context, 我们已经给定通过存取*1 和 *2 分机允许我们改变我们的 movie1 和 movie2 语音提示。

5.5.2. Authenticate()应用的使用

正如我们在前面章节中看到的, 使用 Asterisk, 我们可以录制我们自己的客户化的语音提示。但是或许我们想增加一个安全层并在人们更改语音提示时请求对这些人进行认证。使用 Authenticate()应用, 主叫可以在他们存取更改任何东西前被得到认证。以下示例将演示给你我们怎么利用这个应用把它绑定到我们的 Record()应用中。

我们将直接编辑我们之前的例子以使我们的拨号方案具有更大的灵活性。我们将创建一个特殊属性的 context, 我们可以通过按键*来访问这个 context。Asterisk 然后将请求我们在访问特殊属性的菜单前进行鉴权。一旦鉴权通过, 我们将可以通过按相应的影院号来更改电影院的语音提示。

示例 5-2 : Authenticate()应用的使用

```
[special]
exten => *, 1, Answer()
exten => *, 2, Authenticate(1234)
exten => *, 3, Goto(prompts, s, 1)
[prompts]
exten => s, 1, Answer()
exten => s, 2, Background()
```

5.6. 总结

到此为止, 很明显, 我们已经简单地了解了在 Asterisk 中创建拨号方案的表皮。然而, 你需要更多的知识去学习更多的应用; 这些应用已经在 Asterisk 中内置了, 它会被用来扩展你的拨号方案。现在, 你应当在你创建更强大的拨号方案的道路上需要理解更多的基础应用。花费更多的时间来熟悉这些内嵌的应用。通过扩展我们在本章中已创建的拨号方案来增强你已经在本书中获得的

基本技巧。Asterisk 是一个奇妙的并一直在进化的应用。我们希望你可以学习到更多并和 Asterisk 平台一起持续进步。

6. 后记 (Colophon)

本文档是作为对 Asterisk PBX 系统的介绍而书写的。本文档的主题包括了安装、操作系统准备、通道的创建、编译和拨号方案的介绍。