第六章 Dialplan 基础

目录

Dialplan 是 Asterisk 的核心。它定义了一个呼叫怎么进入以及怎么离开 Asterisk 系统。Dialplan 是一种脚本语言，它包含着 Asterisk 响应外部触发的各种指令。不同于传统电话系统，Asterisk 的 dialplan 是可以完全由用户修改的。

本章介绍了 dialplan 的基本概念。这些信息对你理解或自己编写 dialplan 非常重要。我们希望你仔细阅读本章，因为本章的信息对于 Asterisk 非常基本和重要。同时也请您注意，本章并不能覆盖 dialplan 能做的全部工作。我们的目的只是覆盖那些最基本的东西。我们将在后续章节讨论 dialplan 的高级应用。

在本章的学习中，我们鼓励你多做实验。

6.1 Dialplan 语法

Asterisk 的 dialplan 定义在名为 extensions.conf 的配置文件中。

extensions.conf 文件一般安装在 */etc/asterisk/directory* 目录下，但是它的位置也可能会变化，这取决于你是如何安装 Asterisk 的。这个文件的一些其它常见安装目录还包括 */usr/local/etc/asterisk/* 和 */opt/etc/asterisk/*

Dialplan 由四个基本概念组成：上下文（contexts），分机（extensions），优先级（priorities），应用（applications）。在解释完上述每个要素的作用之后，我们会帮助你创建一个基本但可以工作的 dialplan。

示例配置文件

如果你安装 Asterisk 时选择了安装示例文件的话，你会发现一个已经存在的 *extensions.conf* 文件。我们建议你创建一个你自己的 *extensions.conf* 文件而不是从这个示例文件开始。从示例 *extensions.conf* 开始，并不是最佳和最好的学习如何创建 dialplan 的路径。

不得不说，示例 extensions.conf 文件包含了极其有用的资源，这些例子和想法，当你理解了 dialplan 的基本概念后，你都可以使用它们。如果你在安装 Asterisk 遵从了默认选择，你在目录 */usr/src/asterisk-complete/asterisk/1.8/configs* 下能找到这个 *extensions.conf.samples* （与其它示例文件在一起）。

6.1.1 上下文（Contexts）

Dialplan 是由一个个称为 Contexts 的部分组成的。Context 保持了 dialplan 中不同部分作用的相互独立。在一个 Context 中定义的 extension 和在另外一个 Context 中定义的 extension 是完全独立的，除非我们特别允许它们可以互相作用。（我们将在本章末讨论如何允许 context 中的相互作用）

作为一个简单的例子，让我们想像一下有两家公司共享一个 Asterisk 服务器的情况。如果我们把每个公司的应答流程放在它们各自的 Context 中，这两个公司就完全无关了。这就允许我们分别定义诸如按下“0”时会发生什么：当在 A 公司的语音菜单中按下 0 时，会转接给 A 公司的前台秘书；当在 B 公司的语音菜单中按下 0 时，会转接给 B 公司的前台秘书。（当然，这需要我们此前已经定义了按 0 会转给前台秘书[注1]）。

Conetexts 的定义通过在方括弧（[]）中定义一个名字来实现。这个名字可以包含字母 A

到 Z，数字 0 到 9，以及连字符和下划线[注2]。一个为来电定义的 context 看起来可能是这样的：

[incoming]

Context 名字最长为 79 个字符（80 个字符，减去最后的 NULL 结束符）。

所有跟在一个 context 定义之后的部分都是这个 context 的一部分，直到遇到下一个 context 定义。在 dialplan 的开头，有两个特殊的 context 命名为 **[general]** 和 **[globals]**。**[general]**上下文包含一系列 dialplan 的一般配置（你可能永远不需要理会这些配置），我们会在“全局变量”一节讨论**[globals]**上下文。现在，只要记住**[general]**和**[globals]**不是一般的 context 就可以了。请一定避免使用**[general]**，**[default]**，**[globals]**作为 context 名字，除此而外，你可以使用任何你喜欢的名字。

当我们定义 channel 时（channel 并不是定义在 *extensions.conf* 中，而是定义在诸如 *sip.conf*，*iax.conf*，*chan_dahdi.conf* 等文件中），对每一个 channel 都有一个必须的参数是 **context**。这个 context 参数定义了一个指向 dialplan 入口点的指针，它定义了这个 channel 是如何插入 dialplan 的。Fugure6-1 表示了 channel 配置文件和 diaplan 中 context 之间的关系。

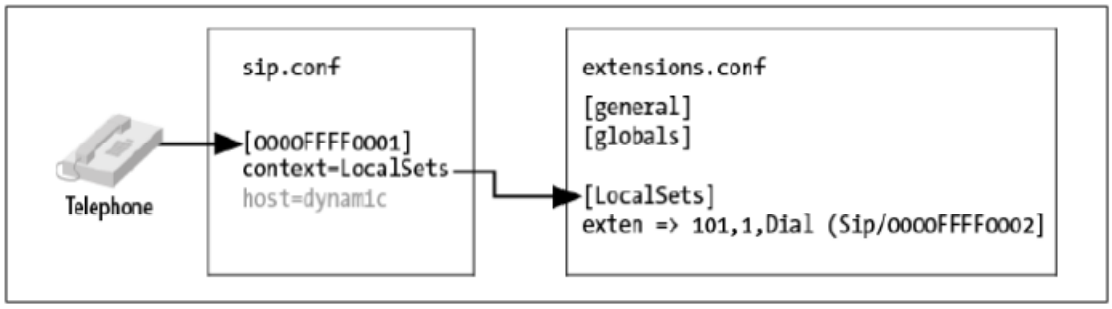


Figure 6-1. Relation between channel configuration files and contexts in the dialplan

这是我们处理 channels 和 dialplans 时需要理解的一个非常重要的概念。一旦你理解了 channel 配置文件中定义的 context 参数和 dialplan 的 context 的匹配关系，你就会发现给 Asterisk 的呼叫流程排错是非常轻松的工作。

Contexts 的一个非常重要的作用（也许是最重要的作用）是提供安全性。通过正确的使用 contexts，你可以为某些用户提供一些其它用户不可使用的功能（例如长途呼叫）。而当你没有仔细的设计好你的 dialplan 时，你可能会无意允许他人欺诈性的使用你的系统。当你构建你的 Asterisk 系统时请牢牢记住，Internet 上有许多专门编写的机器人程序，它们专门寻找和破解没有很好配置安全性的 Asterisk 系统。

Asterisk 的 wiki 文章 https://wiki.asterisk.org/wiki/display/AST/Important+Security+Considerations 中概述了保持 Asterisk 安全性所必须执行的几个步骤。（本书的第 26 章也会专门讨论安全性。）阅读及理解这篇文章是极其重要的。如果你忽略了在那里描述的安全措施，你就可能允许任何人都可以进行长途通话而由你来买单！

如果你不能很谨慎的配置你的 Asterisk 系统的安全性，你终将付出代价。请一定要花时间努力加强你系统的安全性以远离长途欺诈。

6.1.2 分机（Extensions）

在通讯的世界里，术语 *extension* 一般指一个数字号码，当这个号码被拨叫时，将使一个电话（或语音信箱，队列等系统资源）振铃。在 Asterisk 中，extension 的概念要强大的多，因为它定义了一个步骤序列（每个步骤又都包含一个应用），Asterisk 通过这个步骤序列来处理一个呼叫。

在每个 context 中，根据需要，我们可以定义很多（或几个）extensions。当一个特定的 extension 被触发后（被一个来电触发，或者被某个 Channel 上的拨号触发），Asterisk 将执行这个 extension 中已经定义好的步骤。Extension 定义了当一个呼叫通过 dialplan 时会发生什么。尽管 extension 可以（它当然可以）被用于类似传统 PBX 的场景来指定一个分机号码（例如，呼叫 extension 153 会导致 John 的桌面 SIP 电话振铃），在 Asterisk 的 dialplan 中，它还有更多的用途。

extension 的语法是 **exten**，然后紧跟一个等于大于符号，如下面所示：

```
exten =>
```

再然后跟着的是这个 extension 的名字（或者数字号码）。当我们用传统电话系统拨号时，我们一般把 extensions 理解为你拨叫并使该分机振铃的号码。在 Asterisk 中，它有更多的含义，例如，extension 名字可以是任何数字和字母的组合。在本章和下一章的课程中，我们将使到数字和字母两种 extensions。

为 extension 指定名字看似来似乎是个革命性的概念，但当你认识到许多 VoIP 应用都支持（甚至鼓励）用名字或 email 地址，而不是用数字号码来呼叫对方，这就非常浅显易见了。这是使得 Asterisk 如此灵活和强大的特性之一。

Extension 中的每一个步骤都由三个部分组成：

- extension 的名字（或号码）
- 优先级（priority），每个 extension 可以包含多个步骤，每个步骤的编号称为优先级
- 应用（application），或称命令（command），它们将在该步骤被执行时执行；

这三个部分被逗号分开，如下例所示：

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

下面是一个真实 extension 的实例：

```
exten => 123,1,Answer()
```

在这个例子中，extension 名是 **123**，priority 是 **1**，application 是 **Answer()**。

6.1.3 优先级（Priority）

每个 extension 都可以有多个步骤，称为 priorities。Priorities 是一个数字序列，从 1 开始，并且每次执行一个指定的 application。举个例子，下面的 extension 将应答一个呼叫（priority 为 1），然后再挂断它（priority 为 2）：

```
exten => 123,1,Answer()
exten => 123,2,Hangup()
```

很明显，这两行代码没有做任何有用的操作。我们举这个例子，关键是要说明对一个特定 extension 来说，Asterisk 将遵循 priorities 的顺序执行。下面风格的 dialplan 语法仍然经常会见到，尽管（一会你会见到）它在新的代码中不再常见：

```
exten => 123,1,Answer()
exten => 123,2,do something
exten => 123,3,do something else
exten => 123,4,do one last thing
exten => 123,5,Hangup()
```

6.1.3.1 非数字编号优先级 （Unnumbered priorities）

在 Asterisk 发行的老版本中，数字编号的 priorities 带来了很多问题。想像一下一个 15 个 priorities 的 extension，现在要在第二步增加一些东西：所有后续的 priorities 都需要手工重编号。Asterisk 不会执行未编号或错误的编号的步骤，调试这种错误是毫无意义及令人沮丧的。

从 Asterisk 1.2 版本开始，Asterisk 解决了这个问题：它引入了一种 n priority 的机制，n 的意思是下一个（”next”）。每当 Asterisk 遇到 priority 是 n 的语句时，会自动转换为前一个 priority 再加 1。这使得修改你的 dialplan 变得非常容易，因为你不再需要手工重新编号所有的步骤。例如，你的 dialplan 可能看起来像下面这样：

```
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()
```

Asterisk 会在每次遇到 priority 为 n 时计算实际的 priority。[注 3] 需要记住的是，你必须指定 priority 1。如果你不小心把上面例子中第一行的 1 用 n 来代替（一个常见的错误），你会发现当你重新加载 dialplan 后，这个 extension 不存在。

6.1.3.2 ‘same =>’操作符

在永无止境的简化编码的努力下，一个新的语法被创造出来，它可以使 extension 的创建和管理更加容易。当 extensions 中保留有相同的部分时，与其不得不在每一行输入相同的内容，你也可以简单的输入 **same =>** ，然后跟着输入 priority 和 application：

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

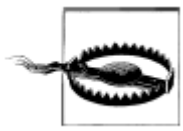
缩进格式不是必须的，但它会提高可读性。这种风格的 dialplan 会使从一个 extension 到另一个 extension 拷贝代码更加容易。我们非常欣赏这种代码风格，并强烈推荐。

6.1.3.3 Priority labels

Priority labels 允许你在一个 extension 中给 priority 指定一个名字。这使得你除了利用 priority 编号（这个编号有时是不知道的，比如你采用 unnumbered priorities 时）外还可以利用 label 来指代这个 priority。我们需要能够寻址到一个特定 extension 中的特定 priority 的原因是，你可能经常会用到需要将一个呼叫从 dialplan 的一个部分跳转到一个特定 extension 的特定 priority。一会我们将对此做更多的讨论。为了给 priority 指定一个文字 label，只要简单的在 priority 之后的括弧中增加一个 label 就可以了，如下例：

```
exten => 123,n(label),application()
```

后面我们还会讨论如何在 dialplan 的不同 priorities 之间跳转的内容。你会看到更多的 priority labels，并且你会在你的 dialplan 中经常使用它们。



一个常见的书写 labels 的错误是在 n 和 (之间错误的插入了一个逗号，例如：

exten => 123,n,(label),application(); <-- THIS IS NOT GOING TO WORK

这个错误会打断你的 dialplan，而且你会得到一个错误导致 application 无法被找到。

6.1.4 应用（Applications）

Applications 是 dialplan 中的驮马（workhorse，实际干活的部分的意思）。每个 application 都会在当前 channel 上执行一个特定的操作，例如播放一段声音，接受一个音频输入，在数据库中查询，拨叫某个 channel，挂机，等等。在前面的例子中，我们介绍了两个简单的 applications：**Answer()**和 **Hangup()**。你现在会学习更多关于它们是如何工作的知识。

一些 Applications，包括 **Answer()** 和 **Hangup()**，不需要其它的信息就能完成操作。然而，大部分 applications，需要一些额外的信息。这些额外的信息称为参数（*arguments*），被传递给 application 以指示如何执行操作。为了把 arguments 传递给 application，需要把它们放在 application 名字之后的括弧中，参数间用逗号分隔。

偶尔，你也可能看到用竖线（|）字符来隔开参数，而不是用逗号。从 Asterisk 1.6.0 开始，用竖线作为分隔符的作法被废弃了。[注 4]

6.1.5 The Answer(), Playback(), 和 Hangup() Applications

Answer() 用于应答一个呼叫。它对收到来电的 channel 执行初始配置操作。如同我们早先提到的那样，**Answer()**不需要参数。**Answer()**并不是必须的（实际上，在某些情况下它根本就不适合使用），但它是一个有效的办法来确保 channel 在执行进一步操作前已经被连接上。

> **The Progress() Application**
>
> 有些时候，在应答（answer）一个呼叫前把信息回送给网络的能力是非常有用的。**Progress ()**就是用来把呼叫过程信息提供给来源 channel 的 application。某些运营商希望你这么做，因此，在你的 dialplan 中处理来电的部分插入 **Progress()**可能有助于解决一些奇怪的信令问题。

Playback()用于在某个 channel 上播放一个预先记录下来的声音文件。在 **Playback()**操作过程中，用户的输入将被忽略，这就意味着你不能用 **Playback()**实现语音自动应答，除非你不希望接受任何信息。[注5]

Asterisk 中有许多专业的声音记录文件，你可以在默认的声音文件目录（通常在/var/lib/asterisk/sounds/）中找到它们。当你编译 Asterisk 时，你可以选择安装用不同语言和不同格式录制的声音样本文件。我们将在许多例子中用到这些声音文件。有几个我们例子中用到的声音文件来自于 Extra Sound Package，所以，请花点时间安装这个包（参见第 3 章）。你也可以通过访问 http://www.theivrvoice.com 来获得与现有声音提示文件一样的你自己的声音提示文件。在本书的后续章节，我们也将讨论更多关于如何利用你的电话和 dialplan 来创建和管理你的系统录音的话题。

为了使用 **Playback()**，需要指定一个文件名（不要扩展名）作为参数。例如，**Playback(filename)** 会播放一个名为 *filename.wav* 的声音文件，假如这个文件在默认声音文件安装目录下存在的话。注意，如果需要的话，你也可以包含完整的文件目录作为参数，例如：

```
Playback(/home/john/sounds/filename)
```

这样，*/home/john/sounds/*目录下的 *filename.wav* 文件会被播放。
你也可以在参数中使用相对路径，例如：

```
Playback(custom/filename)
```

这样，默认声音文件安装目录下的/custom 目录下的 filename.wav（比如/var/lib/asterisk/sounds/custom/filename）会被播放。注意，如果在指定目录下有多个文件名相同而扩展名不同的文件，Asterisk 会自动选择最适合的文件。[注6]

Hangup()的作用与其名字所暗示的完全一样：它挂断一个活动的 channel。你可以在 context 的最后使用这个 application 来结束当前的呼叫，从而确保通话双方都无法在 dialplan 中以你未预料到的方式继续进行操作。**Hangup()**不需要任何参数，但是如果你需要的话，也可以利

用它返回一个 ISDN 的原因代码（例如，**hangup(16)**）。

在本书的学习过程中，我们还将陆续介绍更多的 Asterisk applications。

6.2 一个简单的 Dialplan 例子

好了，我们已经学习了足够的理论。现在，请打开文件 */etc/asterisk/extensions.conf* ，让我们看看你的第一个 dialplan（还记的吗？我们在学习第 5 章时创建的它）。我们将继续在其上增加一些东西。

6.2.1 Hello World

按照许多技术书籍的典型的作法（尤其是计算机编程类书籍），我们的第一个例子称为“Hello World!”

在 extension 中的第一步，我们应答了这个呼叫。在第二步，我们播放了一个名为 hello-world 的声音文件。然后第三步，我们挂断了这个呼叫。在这个例子中，对应的代码就是：

```
exten => 200,1,Answer()
   same => n,Playback(hello-world)
   same => n,Hangup()
```

如果你已经完成了第 5 章的例子，那么你应该已经配置好了两个 IP 电话机，同时你也已经有了一个包含上述代码的 dialplan。如果你还没有实现第 5 章的例子，那么你需要将下述代码输入到 */etc/asterisk/* 目录下的 *extensions.conf* 文件中。

```
[LocalSets] ; this is the context name
exten => 100,1,Dial(SIP/0000FFFF0001) ; Replace 0000FFFF0001 with your device name

exten => 101,1,Dial(SIP/0000FFFF0002) ; Replace 0000FFFF0002 with your device name

exten => 200,1,Answer()
   same => n,Playback(hello-world)
   same => n,Hangup()
```

如果你还没有配置任何 channels，那么现在是做这件事的时候了。当你从头开始创建了一个 Asterisk 的 dialplan 并且利用它打通第一个电话时，满足感会油然而生。当人们意识到他刚刚创建了一个电话系统时，都会觉得非常有趣而开怀大笑。这些快乐当然也属于你，所以，请首先让这个最简单的 dialplan 工作起来。如果你遇到问题，那么请返回第 5 章并完成那里的例子。

如果你刚刚添加了这些 dialplan 代码，你需要通过 Asterisk CLI 相关命令重新加载这个 dialplan：

```
*CLI> dialplan reload
```

或者在 Linux Shell 下输入命令：

```
$ sudo /usr/sbin/asterisk -rx "dialplan reload"
```

然后从你已经配置好的任何一部 IP 电话机拨打分机 200，都会听到 Allison Smith 的声音“Hello World”。如果你没有听到，那么请通过 Asterisk CLI 检查错误信息，并确保你使用的 channel 被指定到 **LocalSets** 上了。

我们不建议你继续本书，直到你确认已经完成下述事宜：

1. 分机 100 和 101 之间呼叫正常；
2. 呼叫 200 可以听到“Hello World”；

尽管这个例子非常短也非常简单，但它依然强调了 contexts，extensions，priority，和 applications 这些核心概念。现在，你已经具备了创建 dialplan 的基础知识了。

6.3 创建一个交互式的 Dialplan

我们刚才创建的 dialplan 是静态的，它总是对与每个呼叫执行相同的操作。许多 dialplan 也需要实现根据用户的不同输入执行不同操作的业务逻辑，现在让我们看看如何做到这一点。

6.3.1 The Goto(), Background(), 和 WaitExten() Applications

如同名字暗示的那样，**Goto()**用于将一个呼叫跳转到 dialplan 的另一个部分。Goto()的语法需要将目的 context，extension，和 priority 作为参数传递给它，像这样：

```
same => n, Goto(context, extension, priority)
```

我们将创建一个名为 **TestMenu** 的新的 context，并且在 **LocalSets** context 中创建一个新的 extension，这个 extension 将利用 Goto()跳转到 **TestMenu**：

```
exten => 201,1,Goto(TestMenu,start,1) ; add this to the end of the
                                      ; [LocalSets] context

[TestMenu]
exten => start,1,Answer()
```

现在，每当一个设备进入 **LocalSets** context 并且拨叫 201，这个呼叫就会被传递给 **TestMenu** context 中的 **start** extension（它现在还没有做任何有意义的事，因为我们还有更多的代码需要添加）。

我们在这个例子中使用 start 作为 extension 的名字，它实际上可以用任何名字代替，不管是数字的还是字母的。我们更倾向于用不能直接拨号的字母作为 extension 的名字，是因为这可以提高可读性。重点是，我们可以用 123 或者 xyz123，或者 99luftballons，或者任何你想解的字符串代替 start。这个“start”在 dialplan 中没有任何意义，它只是代表另一个 extension。

在交互式 dialplan 中最有用的一个 application 是 **Background()**[注7]。像 **Playback()**一样，**Background()**可以播放一个预先录制好的声音文件，但是，当用户按下电话上的按键时，它会中断播放的声音，并根据用户输入的数字把这个呼叫跳转到对应的 extension 去。例如，当用户按下数字 **5** 时，Asterisk 会停止播放语音提示，并将呼叫跳转到 extension **5** 的第一步（假设存在 extension **5** ）。

Background()最常见的应用是创建语音菜单（一般称作 *auto attendants* [注8] 或 *phone trees*）。很多公司通过语音菜单将来电引导到合适的分机上，从而将前台秘书从不得不接听每个电话中解脱出来。

Background()采用和 **Playback()**相同的语法:

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
```

如果你希望 Asterisk 在播放完语音提示后继续等待用户输入一段时间，你可以使用 **WaitExten()**。**WaitExten()**一般跟在 **Background()**之后使用，其作用是等待用户的 DTMF 输入：

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
   same => n,WaitExten()
```

如果你希望为 WaitExten()指定等待用户响应的时间（以取代默认的超时时间[注9]），只需要简单的将代表秒数的数字代入 WaitExten()，像这样：

```
same => n,WaitExten(5); We recommend always passing a time argument to WaitExten()
```

Background()和 **WaitExten()**都允许用户输入 DTMF 数字。然后 Asterisk 会尝试在当前 context 中寻找与这个数字匹配的 extension。如果寻找到了，Asterisk 就会将呼叫传递给这个 extension。让我们通过在我们的示例 dialplan 中增加几行来说明这一点：

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
   same => n,WaitExten(5)

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

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

做完这些修改后，保存并重载你的 dialplan：

```
*CLI> dialplan reload
```

如果你呼叫分机 201，你会听到一个声音提示“main menu”。然后 Asterisk 会等待 5 秒来接收你输入的数字。如果你按下的数字是 **1** 或 **2** ，Asterisk 就会去匹配相应的 extension，然

后语音报出你按下的数字。由于我们没有再提供进一步的指示，所以再然后你的呼叫会被挂断。你也会发现，如果你按下不同的数字（例如 3），这个 dialplan 将无法处理。

让我们再做点改进。我们将利用 Goto()使这个 dialplan 能够在播报按下的数字音后重复播放问候语：

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
   same => n,WaitExten(5)

exten => 1,1,Playback(digits/1)
   same => n,Goto(TestMenu,start,1)

exten => 2,1,Playback(digits/2)
   same => n,Goto(TestMenu,start,1)
```

新增的这几行会在播报完按下的数字后把这个呼叫跳转回**start**，这比直接挂断友好的多了。

如果你仔细查看了 **Goto()** 的说明，会发现实际上你输入一个、两个、或三个参数给 **Goto()**都是可以的。如果你只输入一个参数，Asterisk 将假设这个参数是同一个 extension 中的 priority。如果你输入两个参数，Asterisk 将把它们处理为同一个 context 下的 extension 和 priority。

在这个例子中，我们使用了三个参数是为了清晰的缘故。如果只输入 extension 和 priority 也是相同的效果，因为目的 context 和源 context 是相同的。

6.3.2 处理无效的输入和超时

现在，我们的第一个语音菜单已经工作起来了，让我们再增加一些特殊的 extensions。首先，我们需要一个 extension 来处理错误的输入。在 Asterisk 中，如果一个 context 收到了一个针对不存在的 extension 的请求（例如，在我们上面的例子中输入 **9**），呼叫会被转给 **i** extension 处理。我们还需要一个 extension 来处理当用户在给定的时间（默认的超时时间是 10 秒）内没有按下任何按键的情况。如果用户在 **WaitExten()**被调用后太长时间没有按下按键，呼叫会被传递给 **t** extension。下面是增加了这两个 extension 后的 dialplan：

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
   same => n,WaitExten(5)

exten => 1,1,Playback(digits/1)
   same => n,Goto(TestMenu,start,1)

exten => 2,1,Playback(digits/2)
   same => n,Goto(TestMenu,start,1)

exten => i,1,Playback(pbx-invalid)
   same => n,Goto(TestMenu,start,1)

exten => t,1,Playback(vm-goodbye)
   same => n,Hangup()
```

增加了 **i** 和 **t** extension 使得我们的菜单更加可靠也更友好。当然还得说，它仍然非常简单，因为到目前为止，外线呼叫仍然没有办法联系到一个内线用户。为了做到这一点，我们需要学习另一个 application，称作 **Dial()**。

6.3.3 Dial() Application

Asterisk 最有价值的特性之一，就是它将不同的用户相互连接起来的能力。当不同的用户使用不同的通讯方式时，这一特性尤其有用。举例来说，用户 A 可能使用 PSTN 通话，而用户 B 则可能是在世界另一端的咖啡馆里使用 IP 电话机通话。幸运的是，Asterisk 已经完成了大部分在完全不同的网络之间连接和转化的艰苦工作。你需要做的全部工作就是学习如何使用 **Dial()** application。

Dial()的语法要比我们之前遇到的其它 application 复杂的多，但是别让困难把你吓跑了。**Dial()**使用了四个参数，下面让我们来看一看。

6.3.3.1 参数 1：Destination

这第一个参数是呼叫目的地，它由呼叫采用的技术（或通道）和远端分机或资源的地址组成，中间用斜线隔开。常见的技术类型包括 DAHDI（模拟电话接口和 T1/E1/J1 接口等），SIP，和 IAX2。

举例来说，假设你希望呼叫标识为 **DAHDI/1** 的 DAHDI 分机，这是一个 FXS 接口，可以连接一部普通模拟话机。**DAHDI/1** 的意思是采用的技术类型是 **DAHDI**，资源（或称信道标识）是 **1**。类似的，呼叫一个 SIP 分机（定义在 *sip.conf*）的 destination 可以表示为 **SIP/0004F2001122**，呼叫一个 IXA 分机（定义在 *iax.conf*）的 destination 可以表示为 **IAX2/Softphone**。[注 10] 如果当

dialplan 中的 extension 105 被执行时，我们希望 Asterisk 使 DADHI/1 channel 振铃，那么应该加入如下一行：

exten => 105,1,Dial(DAHDI/1)

我们也可以同时呼叫多个目标，不同的 destinations 之间用“**&**”隔开，像这样：

exten => 105,1,Dial(DAHDI/1&SIP/0004F2001122&IAX2/Softphone)

Dial()可以同时振铃所有的指定 channel，并且接通第一个应答的 channel（所有其它 channel 立即停止振铃）。如果 **Dial()**无法联系上任何一个 destinations，Asterisk 会将它无法完成呼叫的原因代码写进变量 DIALSTATUS，并且继续执行这个 extension 的下一个 priority。[注 11]

Dial()也可以用来连接在 channel 配置文件中没有定义的远端 VoIP 分机。完整的表达式如下例：

Dial(*technology/user[:password]@remote_host[:port][/remote_extension]*)

作为一个例子，你可以用下面的 extension 呼叫 Digium 的演示服务：

exten => 500,1,Dial(IAX2/guest@misery.digium.com/s)

Dial()的语法用于 DAHDI channel 时略有不同：

Dial(DAHDI/*[gGrR]channel_or_group[/remote_extension]*)

下面的例子是通过 DAHDI 的通道 4 [注 12] 拨打 1-800-555-1212：

extern => 501,1,Dial(DAHDI/4/18005551212)

6.3.3.2 参数 2：Timeout

Dial()的第二个参数是 **timeout**，以秒为单位。如果指定了 timeout，**Dial()**会尝试呼叫 destination(s)指定的秒数，超时后就会放弃呼叫而继续执行 extension 的下一条。如果没有指定 timeout，**Dial()**就会一直尝试呼叫被叫 channel(s)，直到被叫应答或主叫挂机。指定 10 秒超时的例子如下：

exten => 201,1,Dial(DAHDI/1,10)

如果呼叫在超时前被应答，channels 之间的连接就会建立，dialplan 完成。如果 destination 一直不应答，占线或者不可用，超时后 Asterisk 会设置变量 DIALSTATUS 并继续执行这个 extension 的下一条。

让我们把刚刚学习的这些用到下面这个例子中：

```
exten => 201,1,Dial(DAHDI/1,10)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()
```

如你所见，在这个例子中，如果呼叫无应答，Asterisk 会播放 *vm-nobodyavail.gsm*。

6.3.3.3 参数 3：Option

Dial()的第三个参数是 option 字符串。它可能包含一个或多个可以影响 **Dial()**行为的字符。所有可能的 option 太多了以至于我们无法都在本书讨论，我们只讨论一个最流行的 option，**m**。如果你将 **m** 作为 **Dial()**的第三个参数，主叫听到的回铃音会被 hold music 取代（译者注：就是咱们的“彩铃”功能了）。举例如下：

```
exten => 201,1,Dial(DAHDI/1,10,m)
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()
```

6.3.3.4 参数 4：URI

Dial()的第四个参数是 URI。如果被叫 channel 支持在呼叫时接收 URI，这个参数指定的 URI 就会被发送（例如，如果你的 IP 电话机支持接收 URI，它就会现实在 IP 电话机的显示屏上；同样地，如果你在使用软件电话，这个 URI 可能会弹出在你的计算机屏幕上）。这个参数非常少被用到。

很少（如果有的话）有电话支持 URI。如果你在寻找一些类似弹屏的应用，你可以参考第 18 章，“Using XMPP(Jabber) with Asterisk”一节。

6.3.3.5 更新我们的 dialplan 例子

让我们在前面语音菜单的例子中使用 **Dial()**：

```
[TestMenu]
exten => start,1,Answer()
   same => n,Background(main-menu)
   same => n,WaitExten(5)

exten => 1,1,Dial(SIP/0000FFFF0001,10) ; Replace 0000FFFF0001 with your device name
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => 2,1,Dial(SIP/0000FFFF0002,10) ; Replace 0000FFFF0002 with your device name
   same => n,Playback(vm-nobodyavail)
   same => n,Hangup()

exten => i,1,Playback(pbx-invalid)
   same => n,Goto(TestMenu,start,1)

exten => t,1,Playback(vm-goodbye)
   same => n,Hangup()
```

6.3.3.6 空白参数

请注意第二、第三、第四个参数都可以不用，只有第一个参数是必须的。举例来说，如果你想指定一个 option，但是并不想指定 timeout，你只要简单的将 timeout 参数留空就可以了，像这样：

exten => 1,1,Dial(DAHDI/1,,m)

6.3.4 使用变量（Using Variables）

在 Asterisk 中可以通过使用变量（Variables）来帮助我们减少输入，提高清晰度，和增加逻辑。如果你具有计算机编程经验的话，你应该已经理解变量是什么了。如果你没有编程经验，那么我们来简单解释下变量是什么以及怎么使用变量。变量是 Asterisk 中极其重要的一个概念。

变量是一个可存储数值的容器。变量的优点是它的值可以改变，但维持名字不变。这就意味着你可以在代码中引用变量名而不必关心值是什么。因此，举例来说，我们可以创建一个变量命名为 **JOHN** 并且指定它的值是 **DAHDI/1**。这样，我们可以在书写 dialplan 时通过 John 的名字来引用他的 channel，而不需要记住 John 使用的 channel 是 **DAHDI/1**。如果将来我们更改了 John 使用的 channel，我们不需要修改任何引用了变量 **JOHN** 的代码，我们只需要修改变量 **JOHN** 的值就可以了。

有两种方法引用变量。引用变量名时，简单的输入变量的名字就可以了，例如 **LEIF**。而如果你希望引用变量的值，你就必须输入 $ 字符，左大括弧，变量名，右大括弧（以 **LEIF** 为例，我们通过**${LEIF}**来引用它的取值）。下例说明如何在 **Dial()**中使用变量：

```
exten => 301,1,Set(LEIF=SIP/0000FFFF0001)
     same => n,Dial(${LEIF})
```

在 dialplan 中，每当遇到**${LEIF}**，Asterisk 都会自动用我们指定给变量 **LEIF** 的值来代替它。

注意，变量名是大小写敏感的。命名为 LEIF 的变量和命名为 Leif 的变量是不同的变量。出于易读性考虑，所有本书例子中的变量名都是大写。你可能也知道 Asterisk 设置的所有变量也是大写。某些变量，例如 CHANNEL 和 EXTEN ，是 Asterisk 保留的。你不应尝试设置这些变量。流行的作法是将全局变量（global variables）写作大写，而 channel 变量写作 Pascal/Camel 这样单词首字母大写的形式。

在 dialplan 中我们可以使用三种类型的变量：全局变量，channel 变量，和环境变量。让我们花一点时间来看看每种类型。

6.3.4.1 全局变量（Global Variables）

如同名字暗示的那样，全局变量对于所有 channel 在任何时间都是可见的。全局变量是非常有用的，它可以用在 dialplan 的任何地方以提高可读性和管理性。假设你有个很大的 dialplan 包含了数百条对 **SIP/0000FFFF0001** channel 的引用。现在，想象一下你不得不遍历整个 dialplan 并把所有的这个引用都换成 **SIP/0000FFFF0002**。这将是一个非常长而且很容易出错的过程。

在另一方面，如果你在 dialplan 一开始已经定义好了值为 **SIP/0000FFFF0001** 的全局变量，并且代码中只是引用这个变量。那么你只需要修改一行代码就可以作用到 dialplan 中所有用到这个 channel 的地方。

全局变量可以被声明在 *extensions.conf* 开始的**[globals]** context 中。作为一个例子，我们创建了一个名为 **LEIF** 的全局变量，它的值为 **SIP/0000FFFF0001**。这个变量会被 Asterisk 在解析 dialplan 时设置。

```
[globals]
LEIF = SIP/0000FFFF0001
```

6.3.4.2 Channel 变量

Channel 变量是一种只与特定呼叫关联的变量。不同于全局变量，channel 变量只存在于呼叫发生期间，并且只对参与呼叫的 channel 有效。

Asterisk 的默认 dialplan 中有许多预先定义好的 channel 变量，这些变量的说明可以在 Asterisk 的维基百科 https://wiki.asterisk.org/wiki/display/AST/Channel+Variables 中找到。

Channel 变量的设置通过 Set() application 来实现：

```
exten => 202,1,Set(MagicNumber=42)
    same => n,SayNumber(${MagicNumber})
```

6.3.4.3 环境变量（Environment variables）

Asterisk 环境变量是一种在 Asterisk 中访问 Unix 环境变量的方法。它通过在 dialplan 中使用 **ENV()** dialplan function 来实现[注13]。语法看起来像**${ENV(var)}**，其中 **var** 是你想引用的 Unix 环境变量。环境变量在 Asterisk dialplan 中并不常用，但是如果你需要的话，它是可以使用的。

6.3.4.4 将变量增加到我们的 dialplan 例子中

现在我们已经学习了变量，让我们把它们增加到我们的 dialplan 例子中。我们将增加三个与 channel 名相关的全局变量：

```
[globals]
LEIF=SIP/0000FFFF0001
JIM=SIP/0000FFFF0002
RUSSELL=SIP/0000FFFF0003

[LocalSets]
exten => 100,1,Dial(${LEIF})
exten => leif,1,Dial(${LEIF})

exten => 101,1,Dial(${JIM})
exten => jim,1,Dial(${JIM})

exten => 102,1,Dial(${RUSSELL})
exten => russell,1,Dial(${RUSSELL})

[TestMenu]
exten => 201,1,Answer
      same => n,Background(enter-ext-of-person)
      same => n,WaitExten()

exten => 1,1,Dial(DAHDI/1,10)
      same => n,Playback(vm-nobodyavail)
      same => n,Hangup()

exten => 2,1,Dial(SIP/Jane,10)
      same => n,Playback(vm-nobodyavail)
      same => n,Hangup()

exten => i,1,Playback(pbx-invalid)
      same => n,Goto(incoming,123,1)
```

```
exten => t,1,Playback(vm-goodby)
    same => n,hangup()
```

你可能注意到我们给每个 extension 号码都增加了一个别名。在 6.1.2 分机（Extensions）一节，我们解释过 Asterisk 并不关心你对 extension 的命名方式。在这个例子中，我们简单的给每个分机都增加了字符的和数字号码的两个名字。extension **100** 和 **leif** 都可以定位到 ***SIP/0000FFFF0001***，extension **101** 和 **jim** 都可以定位到 ***SIP/0000FFFF0002***，而 **102** 和 **russell** 也都可以定位到 ***SIP/0000FFFF0003***。这些设备通过全局变量**${LEIF}**，**${JIM}**，和**${RUSSELL}**来标识，通过 **Dial()**来实现呼叫。

在我们的测试菜单中，我们简单的随便选择了被叫分机，例如 **DAHDI/1** 和 **SIP/Jane**。你可以用任意分机替代它们。我们建立 **TestMenu** context 只是想给你一些 Asterisk dialplan 是什么样的概念。

6.3.5 样式匹配（Pattern Matching）

如果我们希望允许人们通过 Asterisk 拨打电话并且利用 Asterisk 连接外部资源，我们就需要一个方法能匹配所有可能被拨打的号码。为处理这类问题，Asterisk 提供了样式匹配（*pattern matching*）的机制。样式匹配机制可以允许你在你的 dialplan 中创建一个能够匹配许多不同号码的 extension。这个功能非常有用！

6.3.5.1 样式匹配语法

当我们使用样式匹配是，特定的字母和符合代表了我们希望匹配的东西。*样式总是从一个下划线(_)开始*。这告诉 Asterisk 我们正在匹配一个样式，而不是匹配一个精确的 extension 名字。

如果你忘记了样式开始的下划线，Asterisk 会认为这只是一个 extension 的名字，而不会做一个样式匹配。这是人们刚开始学习 Asterisk 时最容易犯的一个错误。

在下划线之后，你可以使用一个或多个下列字符：

X
: 匹配任意 0 到 9 之间的一个数字

Z
: 匹配任意 1 到 9 之间的一个数字

N
: 匹配任意 2 到 9 之间的一个数字

[15-7]
: 匹配指定范围的一个数字。在这个例子中的样式要求匹配一个 1，以及 5,6,7 中的任意一个数字

.（点）
: 通配符；匹配一个或多个字符，不论它们是什么

如果你不够小心，通配符匹配可能让你的 dialplan 做一些你没想到的事情（例如匹配了内建的 extensions 如 **i** 和 **h**）。你应该仅当你已经尽可能的匹配了尽量多的数字后才使用通配符。例如，下面的样式应该永远不要使用：

_.

事实上，当你这么使用时 Asterisk 会提示你一个告警。如果你真的需要一个匹配所有输入的样式，你也应该采用如下列用法，匹配所有数字开头的字符串：

_X.

或者如下例用法，匹配任意字符串：

_[0-0a-zA-Z].

!（叹号）

通配符；匹配零个或多个字符，不论它们是什么

如果想在你的 dialplan 中使用样式匹配，只要简单的把样式输入到 extension 名字（或号码）的位置就可以了：

```
exten => _NXX,1,Playback(silence/1&auth-thankyou)
```

在这个例子里，这个样式匹配任意从 200 到 999 之间的三个数字号码的 extension（**N** 代表任意 2 到 9 之间的数字，每个 **X** 代表一个 0 到 9 之间的数字）。也就是说，如果用户拨打这个 context 中的任意 200 到 999 之间的三位数字的分机号码，他都会听到一个声音文件 auth-thankyou.gsm。

关于 Asterisk 样式匹配的另一个需要知道的重要规则是，如果 Asterisk 发现一个样式可以匹配多个 extension，它将使用最精确的那个（从左到右）。比如说你已经定义了下面两个样式，并且有用户拨打 **555-1212**：

```
exten => _555XXXX,1,Playback(silence/1&digits/1)
exten => _55512XX,1,Playback(silence/1&digits/2)
```

在这个例子中，第二个 extension 会被选中，因为它更加精确。

6.3.5.2 样式匹配例子

下面这个例子匹配 7 位数字号码，并且首个数字大于 2：

_NXXXXXX

这个样式可以兼容 NANP（北美编号计划）的本地 7 位号码。
当采用 10 位号码拨号时，这个样式看起来会是：

_NXXNXXXXXX

注意，这两个样式都不能处理长途号码。我们马上会讲到长途号码的处理。

NANP 和费用欺诈

北美编号计划（NANP）是在北美和加勒比海地区 19 个国家共享的电话号码编号机制。所有这些国家共享国家代码 1。

在美国和加拿大，电信规则是相似的（和合理的），因此你可以拨打大部分国家代码 1 开头的电话并期望一个合理的费用。然而，很多人并没有认识到还有 17 个国家也共享 NANP，而且它们中的许多国家采用非常不同的电信规则。（更多信息参见 http://www.nanpa.com）

一种流行的骗局是利用 NANP 哄骗幼稚的美国人拨打昂贵的，按分钟计费的加勒比海地区国家的电话。这些用户受骗，是因为他们拨打的是 1-NPA-NXX-XXXX 的号码，所以他们认为他们只需要支付标准的国内长途话费。这些问题国家的电信规则可能允许这种类型的诈骗，所以这些用户最终将支付高额的话费。

避免这类骗局的唯一办法是禁止某些地区代码（例如 809），并且仅仅在需要时才移除这个限制。

让我们来试试另一个样式：

```
_1NXXNXXXXXX
```

这个样式可以匹配数字 1，跟着一个在 200 到 999 之间的地区代码，然后是任意的 7 位号码。在 NANP 地区，你可以用这个样式匹配任意长途号码。[注 14]

最后是这个样式：

```
_011.
```

注意最后的点。这个样式匹配 011 开始的，并且至少还有一位数字的任意号码。在 NANP 中，这代表一个国际电话号码（我们将在下一节使用这个样式来给我们的 dialplan 增加外呼能力）。

其它国家号码的样式匹配

在本节的例子中是以 NANP 区域为中心的，但是基本的逻辑可以用于任何国家。这里有一些其它国家的例子（注意我们并不能测试它们）：

```
; UK, Germany, Italy, China, etc.
_00. ; international dialing code
_0. ; national dialing prefix

; Australia
_0011. ; international dialing code
_0. ; national dialing prefix
```

这绝不全面，但它可以给你一个样式的通用概念，你可以进一步考虑如何应用在你自己的国家上。

6.3.5.3 使用${EXTEN} channel 变量

如果你使用了样式匹配，但又需要知道到底拨打的是什么号码时，可以怎么办呢？可以利用**${EXTEN}** channel 变量。每当你拨叫一个分机时，Asterisk 会设置**${EXTEN}** channel 变量为实际拨打的号码。我们可以用一个名为 **SayDigits()**的 application 来测试一下：

```
exten => _XXX,1,Answer()
    same => n,SayDigits(${EXTEN})
```

在这个例子中，**SayDigits()**会回读你拨打的三位分机号码。

通常，从${EXTEN}的前面去掉几位数的操作是有用的。这可以利用表达式**${EXTEN:X}**来实现，其中 X 是你希望去掉的位数，方向是从左到右。例如，如果**${EXTEN}**的值是 **95551212**，**${EXTEN：1}** 就等于 **5551212**。让我们再看另一个例子：

```
exten => _XXX,1,Answer()
    same => n,SayDigtis(${ExTEN:1})
```

在这个例子中，**SayDigits()**将从第二个数字开始，这样将只回读被叫分机的后两位数。

更高级的数字操作

${EXTEN}**变量还有一种表达式**${EXTEN:x:y}，其中 **x** 是起始位置，**y** 是返回的数字个数。给定下列字符串：

94169671111

我们可以利用**${EXTEN:x:y}**抽取下列数字：

- **${EXTEN:1:3}** 得到 **416**
- **${EXTEN:4:7}** 得到 **9671111**
- **${EXTEN:-4:4}** 将从倒数第 4 个数字开始，得到 **1111**
- **${EXTEN:2:-4}** 将从跳过 2 个数字开始，并不包括最后的 4 个数字，得到 **16967**
- **${EXTEN:-6:-4}** 将倒数第 6 个数字开始，并不包括最后 4 个数字，得到 **67**
- **${EXTEN:1}** 将返回第 1 个数字之后的全部数字，得到 **4169671111**（如果返回的数字个数为空的话，它就返回剩余的全部数字）

这是一个非常强大的表达式，但是大部分这些变化并不常用。在大多数情况下，你将使用**${EXTEN}**（或者**${EXTEN:1}**，如果你需要去掉外线识别码）。

6.3.6 Includes

Asterisk 的一个重要特性是允许在一个 context 中定义的 extension 可以在另一个 context 中使用。这是通过 **include** 指令实现的。通过 **include** 指令我们可以访问 dialplan 的不同部分。

Include 的语法如下所示，其中 **context** 是被包含的 context 的名字。

include => context

在一个 context 中包含另一个 context，将允许被包含 context 中的 extension 可以在当前 context 下拨打。

当我们在当前 context 下包含其它 contexts 时，我们一定要留意包含的顺序。Asterisk 将首先尝试匹配当前 context 中的 extensions。如果没成功，它再尝试匹配第一个被包含的 context 中的 extensions（包括这个 context 中包含的其它 context），然后再继续匹配下一个被包含的 context。

我们将在第 7 章进一步讨论 **include** 。

6.4 结论

现在你已经得到了一个基本但是具有一定功能的 dialplan。虽然仍然有许多东西我们没有讲到，但是你应该已经学习到了所有的基础知识。在后续的章节中，我们将在此基础上进一步展开讨论。

如果 dialplan 中的有些部分你还没有理解，你应该在进入下一章前重读一两遍。理解这些原理及如何应用它们是极其重要的，因为这是理解下一章的基础。

注释：

注 1：这是一个非常重要的考虑。对于传统 PBX 来说，一般都有对于前台秘书接听的默认设置。这就意味着即使你忘记配置了，它也可能能正常工作。但在 Asterisk 中，正好与之相反。如果你没有告诉 Asterisk 如何处理某个状态，Asterisk 就不会做任何处理，然后这个呼叫会被挂断。我们后续会通过一些实例来讨论如何避免这种情况发生。请参阅“处理无效的输入和超时”一节。

注 2：请注意“空格”是明显的非法字符。千万不要把“空格”用在 context 名字中——你不会喜欢那结果的。

注 3：Asterisk 允许在 priority 中包含简单的算式，例如 n+200，以及 priority s（same 的意思），但是由于 priority label 的关系我们并不赞成这么使用。需要注意的是 *extension s* 和 *priority s* 是截然不同的两个概念。

注 4：除了 *voicemail.conf* 的某些部分以外。

注 5：Asterisk 中还有一个 application 叫做 **Background()**，它与 **Playback()**非常相似，不过 **Background()**可以接受主叫的输入。你将在第 15 章和第 17 章学习到更多有关 **Background()**的知识。

注 6：Asterisk 是基于文件格式转化代价最低的原则来选择最合适的文件的——这意思是说，它会选择解码为可播放的语音时 CPU 消耗最低的格式。当你启动 Asterisk 时，它会计算不同音频格式之间转换编码的代价（这经常随系统不同而不同）。你可以通过在 Asterisk CLI 下输入命令 show translation 看到不同编码之间转换的代价。其中的数字表示 Asterisk 转换 1 秒声音需要用多少微秒（milliseconds）。我们将在后续章节讨论更多关于编码格式（称为 *codecs*）的内容。

注 7：需要注意的是，可能是由于 Background()名字的原因，有些人想当然的以为它的作用是“背景音乐”，即在继续执行 dialplan 后续步骤的同时声音一直播放。实际上，我们用 background 这个名字只是想说明它是在后台播放声音的同时，在前台等待 DTMF 输入。

注 8：更多关于自动应答（auto attendants）的信息可以在第 15 章找到。

注 9：参见 dialplan function TIMEOUT() 了解如何修改默认超时时间。参见第 10 章了解什么是 dialplan functions。

注 10：如果是在实际产品中使用，这实在不是一个好的设备名字。想像一下如果你的系统上使用了多于一个软件电话（Softphone），或者你未来增加了一部软件电话，你怎么区分它们？

注 11：我们将在下一小节“变量”讨论有关变量的内容。在后续章节，我们也将讨论如何使你的 dialplan 基于 DIALSTATUS 的值做出决定；

注 12：请记住这里假定这个通道（DAHDI/4）可以接通外部号码（1-800-555-1212）。

注 13：我们将在稍后讨论 dialplan functions，你无需过于担心环境变量，它们对理解 dialplan 无关紧要。

注 14：如果你是在美国长大的，你可能以为在拨打长途电话前拨的数字 1 是“长途代码”。这个认识是不

对的。数字 1 是 NANP 的国家代码。当你要把你的电话告诉其它国家的人时请一定记住这一点。接受者可能不知道你的国家代码，这样他就无法在只知道你的地区代码和电话号码的情况下呼叫你。你的带有国家代码的完整号码是 +1 NPA NXX XXXX（其中 NPA 是你的地区代码），例如，+1 416 555 1212

译者注 1，（2012.4.13）初稿成于 2011.12.2，其中 context 的翻译，觉得很是纠结。从意义上我们理解 context 其实指的是一个“作用域”，每个 context 在 dialplan 中就是一个独立小王国。但这个翻译还真不容易，初版我翻译为字段，今天看看，实在太容易理解成 field，非常不妥。说不得，还是先老老实实的直译为“上下文”吧。