# 第十四章 设备状态

## 目录

能够确定连接到电话系统的设备的状态通常是有用的。举例来说，前台接线员可能希望看到全体办公室人员状态的信息，以确定他们是否能否接听电话。Asterisk 本身也需要这些信息。再举一个例子，如果你按照第 13 章的描述创建了一个呼叫队列，Asterisk 需要了解坐席什么时候空闲以便分配另一个呼叫。本章讨论 Asterisk 中的设备状态概念，以及设备和应用程序如何使用和访问这些信息。

# 14.1 设备状态

设备状态涉及两种类型的设备：真实的设备和虚拟设备。真实的设备指可以接打电话的电话终端设备，例如 SIP 话机。虚拟设备指在 Asterisk 内部的一些提供了有用的状态信息的东西。Table14-1 列出了 Asterisk 中的有效虚拟设备。

*Table 14-1. Virtual devices in Asterisk*

| Virtual device | Description |
|---|---|
| `MeetMe:<conference bridge>` | The state of a MeetMe conference bridge. The state will reflect whether or not the conference bridge currently has participants called in. More information on using `MeetMe()` for call conferencing can be found in "Conferencing with MeetMe()" on page 218. |
| `SLA:<shared line>` | Shared Line Appearance state information. This state is manipulated by the `SLATrunk()` and `SLAStation()` applications. More detail can be found in "Shared Line Appearances" on page 318. |
| `Custom:<custom name>` | Custom device states. These states have custom names and are modified using the `DEVICE_STATE()` function. Example usage can be found in "Using Custom Device States" on page 307. |
| `Park:<exten@context>` | The state of a spot in a call parking lot. The state information will reflect whether or not a caller is currently parked at that extension. More information about call parking in Asterisk can be found in "Parking Lots" on page 228. |
| `Calendar:<calendar name>` | Calendar state. Asterisk will use the contents of the named calendar to set the state to `available` or `busy`. More information about calendar integration in Asterisk can be found in Chapter 18. |

设备状态是一对一的映射到设备上的。Figure14-1 显示这种对应关系。

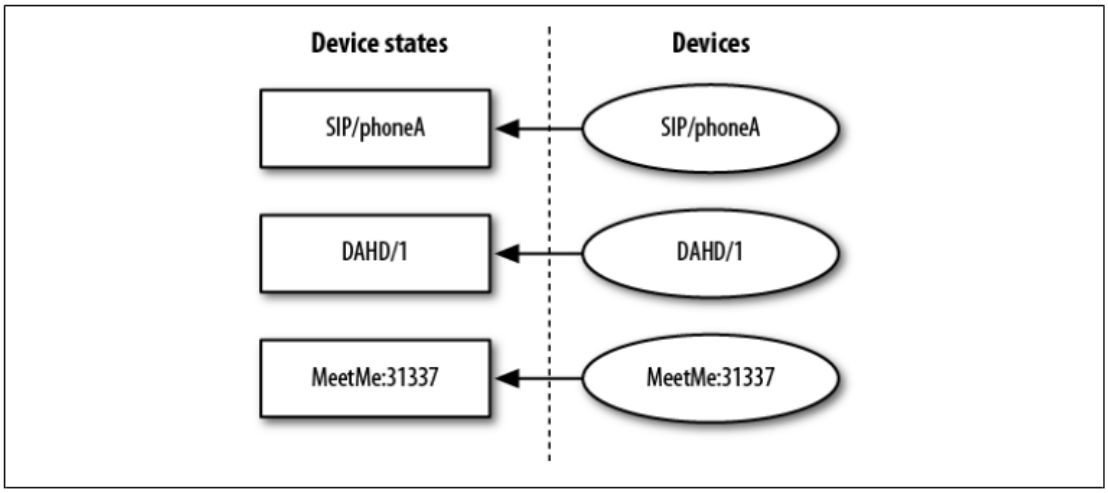


*Figure 14-1. Device state mappings*

### 14.1.1检查设备状态

DEVICE_STATE() dialplan 函数可以用于读取设备的当前状态。下面是一段可以用在dialplan 中的简单例子：

```
exten => 7012,1,Answer()
; *** This line should not have any line breaks
    same => n,Verbose(3,The state of SIP/0004F2060EB4 is
${DEVICE_STATE(SIP/0004F2060EB4)})
    same => n,Hangup()
```

如果我们从我们检查状态的设备拨打 7012，下述诊断信息将会显示在 Asterisk 控制台上：

```
-- The state of SIP/0004F2060EB4 is INUSE
```

第 20 章讨论 Asterisk 管理接口（Asterisk Manager Interface，AMI）。**Get Var** 管理操作可以用在外部程序检索设备状态中。你可以通过它获得一般变量或 dialplan 函数的值。例如 DEVICE_STATE()。

下面列出了 **DEVICE_STATE()**函数的所有可能返回值：

- UNKNOWN
- NOT_INUSE
- INUSE
- BUSY
- INVALID
- UNAVAILABLE
- RINGING
- RINGINUSE
- ONHOLD

## 14.2 Extension状态

Extension 状态是 Asterisk 中的另一个重要概念。Extension 状态可以用于 SIP 设备用户实现在线状态（presence）信息。（SIP 在线状态信息将在本章“14.3SIP 在线状态”中进一步详细讨论）。Extension 的状态取决于检查的一个或多个设备的状态。映射到 Extension 状态的设备列表定义在 Asterisk dialplan 中（*/etc/asterisk/extensions.conf*），采用特殊的 **hint** 指令。Figure 14-2 显示了设备，设备状态，和 extension 状态之间的映射关系。

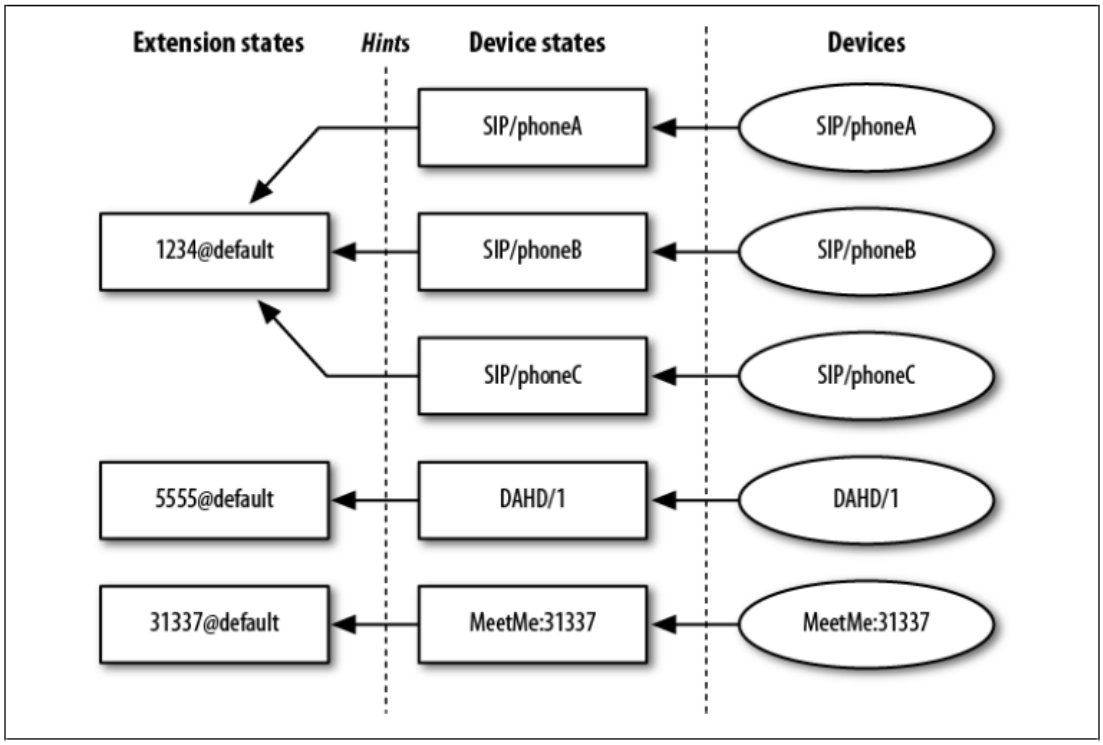


*Figure 14-2. Extension state mappings*

## 14.2.1 Hints

为了在 dialplan 中定义 extension 状态的 hints，关键词 **hint** 将用来取代原来 priority 的位置。下面是一个与 Figure 14-2 对应的简单例子：

```
[default]
exten => 1234,hint,SIP/phoneA&SIP/phoneB&SIP/phoneC
exten => 5555,hint,DAHDI/1
exten => 31337,hint,MeetMe:31337
```

典型地，“hints”可以简单的跟其余的 extension 一起定义。下一个例子增加了一行，用于说明这个 extension 被呼叫时的处理。

```
[default]
exten => 1234,hint,SIP/phoneA&SIP/phoneB&SIP/phoneC
exten => 1234,1,Dial(SIP/phoneA&SIP/phoneB&SIP/phoneC)
exten => 5555,hint,DAHDI/1
exten => 5555,1,Dial(DAHDI/1)
exten => 31337,hint,MeetMe:31337
exten => 31337,1,MeetMe(31337,dM)
```

在我们的例子中，我们将 hints 的 extension 号码和被呼叫的 extension 号码直接关联，尽管这并不是必须的。

## 14.2.2检查Extension状态

检查一个 extension 当前状态的最简单的办法是通过 Asterisk 命令行（CLI）。*core show hints* 命令将显示所有当前配置好的 hints。思考下面的 hint 定义：

```
[phones]
exten => 7001,hint,SIP/0004F2060EB4
```

当 *core show hints* 命令在 Asterisk CLI 执行时，如果设备正在使用中，则会出现下面的输出信息：

```
*CLI> core show hints
    -= Registered Asterisk Dial Plan Hints =-
        7001@phones : SIP/0004F2060EB4    State:InUse    Watchers 0
----------------
- 1 hints registered
```

除了显示 extension 的状态信息之外，*core show hints* 的输出还提供了 watchers 的计数。这里的 *watcher* 是指在 Asterisk 系统中订阅了接收这个 extension 状态变化的东西。比如一个 SIP 话机订阅了某个 extension 的状态，则 watcher 计数就会增加。

Extension 状态也可以通过 dialplan 函数 **EXTENSION_STATE()**获得。这个函数的操作类似于之前描述的 **DEVICE_STATE()**函数。下面的例子显示了一个 extension，它将向 Asterisk 控制台打印另一个 extension 的状态信息：

```
exten => 7013,1,Answer()
    same => n,Verbose(3,The state of 7001@phones is ${EXTENSION_STATE(7001@phones)})
    same => n,Hangup()
```

当这个 extension 被调用，就会在 Asterisk 控制台输出下列信息：

```
-- The state of 7001@phones is INUSE
```

下面的列表包含了 **EXTENSION_STATE()**的所有可能返回值：

- UNKNOWN
- NOT_INUSE
- INUSE
- BUSY
- UNAVAILABLE
- RINGING
- RINGINUSE
- HOLDINUSE
- ONHOLD

# 14.3 SIP在线状态

Asterisk 给设备提供了使用 SIP 协议订阅 extension 状态的能力。这个功能通常被称为 BLF（Busy Lamp Field）。[注1]

## 14.3.1 Asterisk配置

为了让这个功能工作,hints 必须在*/etc/asterisk/extensions.conf* 中定义。此外,SIP channel 驱动配置文件(*/etc/asterisk/sip.conf*)中的一些重要选项也必须被设置。下面的列表讨论了这些选项:

**callcounter**

使能/禁止呼叫计数器。这个选项必须使能,Asterisk 才能向 SIP 设备提供状态信息。这个选项被设置在 *sip.conf* 文件中**[general]**部分或者指定的 peer 部分。

如果你希望 SIP 设备状态信息可以工作,你必须至少将 **callcounter** 选项设置为 **yes**。否则,SIP channel 驱动不会自找麻烦跟踪呼叫到达和离开设备的情况,因此也就不能提供关于它们的状态信息。

**busylevel**

设置多少个呼叫并行在某个设备时,Asterisk 报告这个设备为忙(busy)。这个选项只能配置在 *sip.conf* 中的 peer 相关部分。默认情况不设置这个选项。这意味着 Asterisk 将报告这个设备状态为 in use,而且永远也不会报告为 busy。

**call-limit**

这个选项已经弃用。

**allowsubscribe**

允许你禁止对订阅的支持。如果这个选项没有被设置,则订阅是允许的。如果想完全禁止订阅支持,可以在 *sip.conf* 中**[general]**部分设置 **allowsubscribe** 为 **no**。

**subscribecontext**

允许你为订阅设置特定的 context。如果不设置这个选项,将使用 **context** 选项定义的 context。这个选项可以设置到 *sip.conf* 中**[general]**部分或 peer 特有的部分。

**notifyringing**

控制当 extension 进入振铃状态时是否发送通知。这个选项默认设置为 **yes**。它仅对使用了 *dialog-info* 事件包的订阅有效。这个选项只能全局设置在 *sip.conf* 的**[general]**部分。

**notifyhold**

允许 **chan_sip** 设置 SIP 设备的状态为 **ONHOLD**。这个选项默认设置为 **yes**。这个选项只能全局设置在 *sip.conf* 的**[general]**部分。

**notifycid**

使能/禁止发送来电的主叫号码给 extension。这个选项适用于订阅了基于 **dialog-info+xml** 的 extension 状态通知的设备,例如 Snom 电话。显示主叫号码信息对于帮助坐席决定是否代接(pickup)电话是有用的。这个选项默认设置为 **no**。

这个神奇的代接功能(pickup)仅工作在代接电话的 hint 的 extension 和 context 和振铃电话的 extension 和 context 是相同的。请注意,使用 subscribecontext 选项通常会破坏这个功能。这个选项也可以设置为 **ignore-context**。这将忽略 context 的比较,这只应该用于这个 extension 仅有单一实例被订阅的情况。否则,你可能会意外应答了不应该你应答的电话。

## 14.4 使用自定义设备状态

Asterisk 提供了创建自定义设备状态的能力。这有助于开发一些有趣的自定义应用。我们从展示控制自定义设备状态的基本语法开始，然后我们将创建一个使用它们的例子。

自定义设备状态都起始于前缀**Custom:**。跟在前缀之后的文字可以是任何你需要的文字。设置和读取自定义设备状态的值，使用 **DEVICE_STATE()** dialplan 函数。例如，设置自定义设备状态的语法为：

```
exten => example,1,Set(DEVICE_STATE(Custom:example)=BUSY)
```

类似地，读取自定义设备状态当前值得语法为：

```
exten => Verbose(1,The state of Custom:example is ${DEVICE_STATE(Custom:example)})
```

自定义设备状态可以被用作直接控制显示在设备上的，订阅的某 extension 状态的一种方法。我们在 dialplan 中用 hint 映射一个 extension 到一个设备状态：

```
exten => example,hint,Custom:example
```

### 14.4.1一个例子

对于自定义设备状态有很多有趣的使用案例。在本节中，我们将创建一个例子，在 SIP 话机上实现一个自定义的“免打搅”（DND, do not disturb）带灯按键。同样的方法可以用在许多其他你希望实现一个“乒乓”开关带灯按键的事情上。例如，这个方法可以用于坐席人员了解他们当前是否登录到了队列。

例子的第一部分是在 dialplan 中定义 hint。这是必须的，这样才能在 SIP 话机上配置 BLF 订阅这个 extension。在这个例子中，这个话机必须被配置成订阅 DND_7015 的状态：

```
exten => DND_7015,hint,Custom:DND_7015
```

下一步，我们将创建一个 extension，当用户按下和这个自定义的 DND 特性关联的按键时，这个 extension 会被调用。值得注意的是，这个 extension 并不进行语音操作。事实上，SIP 话机的用户可能并不知道当他按下这个按键时会产生一个呼叫。就用户而言，按下这个按键只是简单的打开或关闭一个反映 DND 是否使能的指示灯。这个 extension 看起来如下：

```
exten => DND_7015,1,Answer()
    same => n,GotoIf($["${DEVICE_STATE(Custom:DND_7015)}"="BUSY"]?turn_off:turn_on)
    same => n(turn_off),Set(DEVICE_STATE(Custom:DND_7015)=NOT_INUSE)
    same => n,Hangup()
    same => n(turn_on),Set(DEVICE_STATE(Custom:DND_7015)=BUSY)
    same => n,Hangup()
```

这个例子最后的部分展示了这个 DND 状态是如何在 dialplan 中使用的。如果 DND 开关使能，Asterisk 会向主叫播放一条信息说用户不可用。如果 DND 开关禁止，呼叫就会被转接给这个 SIP 设备：

```
exten => 7015,1,GotoIf($["${DEVICE_STATE(Custom:DND_7015)}"="BUSY"]?busy:available)
    same => n(available),Verbose(3,DND is currently off for 7015.)
    same => n,Dial(SIP/exampledevice)
    same => n,Hangup()
    same => n(busy),Verbose(3,DND is on for 7015.)
```

```
        same => n,Playback(vm-theperson)
        same => n,Playback(digits/7&digits/0&digits/1&digits/5)
        same => n,Playback(vm-isunavail)
        same => n,Playback(vm-goodbye)
        same => n,Hangup()
```

Example 14-1 展示了这个例子如果出现在*/etc/asterisk/extensions.conf* 中时的完整情况。

*Example 14-1. Custom "do not disturb" functionality using custom device states*

```
;
; A hint so a phone can use BLF to signal the DND state.
;
exten => DND_7015,hint,Custom:DND_7015
;
; An extension to dial when the user presses the custom DND
; key on his phone. This will toggle the state and will result
; in the light on the phone turning on or off.
;
exten => DND_7015,1,Answer()
    same => n,GotoIf($["${DEVICE_STATE(Custom:DND_7015)}"="BUSY"]?turn_off:turn_on)
    same => n(turn_off),Set(DEVICE_STATE(Custom:DND_7015)=NOT_INUSE)
    same => n,Hangup()
    same => n(turn_on),Set(DEVICE_STATE(Custom:DND_7015)=BUSY)
    same => n,Hangup()
;
; Example usage of the DND state.
;
exten => 7015,1,GotoIf($["${DEVICE_STATE(Custom:DND_7015)}"="BUSY"]?busy:available)
    same => n(available),Verbose(3,DND is currently off for 7015.)
    same => n,Dial(SIP/exampledevice)
    same => n,Hangup()
    same => n(busy),Verbose(3,DND is on for 7015.)
    same => n,Playback(vm-theperson)
    same => n,Playback(digits/7&digits/0&digits/1&digits/5)
    same => n,Playback(vm-isunavail)
    same => n,Playback(vm-goodbye)
    same => n,Hangup()
```

## 14.5 分布式设备状态

Asterisk 是主要设计运行于单机系统上的。然而，随着可扩展性需求的增加，经常需要部署多台 Asterisk 服务器。由于这种情况越来越常见，一些新的特性已经被增加以更容易的协调多台 Asterisk 服务器。这些特性之一就是支持分布式设备状态（*distributed device state*）。

这就意味着，如果一台设备在一台 Asterisk 服务器上处于通话状态中，这台设备在所有服务器上的状态都能反映这一点。说得更清楚些，这种工作方法就是从每个服务器的观点看，

每个服务器都知道每个设备的状态。使用这些收集的状态，每个服务器都要计算设备的总体状态并报告给其它的 Asterisk 服务器。

为了实现分布式设备状态，必须使用某种消息传递机制在服务器间通讯。Asterisk 1.8 支持两种这样的机制：AIS 和 XMPP。

## 14.5.1使用OpenAIS

Application Interface Specification（AIS）是一组标准化的消息传递中间件 API。这些 API 的定义由 Service Availability Forum（http://www.saforum.org）提供。用于开发和测试的开源 AIS 实现是 OpenAIS（http://www.openais.org），它由 Corosync（http://www.corosync.org）创建。

Corosync，也就是 OpenAIS，适合于所有节点位于同一个高速、低时延的以太网上。如果你的部署在地理上是分散的，你应该使用基于 XMPP 的分布式设备状态支持，这将在后续“14.6 使用 XMPP”一节讨论。

### 14.5.1.1 安装

第一步是获取安装 Corosync 和 OpenAIS 必须的组件。Corosync 依赖于 NSS 库。请在 Ubuntu 下安装 *libnss3-dev* 包，或者在 CentOS 下安装 *nss-devel* 包。

下一步，安装 Corosync 和 OpenAIS。它们可能有可用的安装包，但是从源码安装它们也是相当简单的。从 Corosync 和 OpenAIS 的主页下载最新的版本。然后，执行下面的命令编译和安装每个包：

```
$ tar xvzf corosync-1.2.8.tar.gz
$ cd corosync-1.2.8
$ ./configure
$ make
$ sudo make install
$ tar xvzf openais-1.1.4.tar.gz
$ cd openais-1.1.4
$ ./configure
$ make
$ sudo make install
```

如果你在安装 Corosync 和 OpenAIS 之前安装了 Asterisk，你需要重新编译和安装 Asterisk 来获得 AIS 支持。这从运行 Asterisk 配置脚本开始。配置脚本负责检查系统的依赖关系，以便编连系统了解需要编连哪些模块：

```
$ cd /path/to/asterisk
$ ./configure
```

运行配置脚本之后，运行 *menuselect* 工具确保 Asterisk 编连了 **res_ais** 模块（这个模块可以在 *menuselect* 的 *Resource Modules* 部分找到）：

```
$ make menuselect
```

最后，编译和安装 Asterisk：

```
$ make
$ sudo make install
```

这是相当地快速和粗糙的编译和安装 Asterisk 的说明。如果需要获取更多关于 Asterisk 编译和安装的说明，请参考第三章。

## 14.5.2 OpenAIS 配置

现在 OpenAIS 已经安装好了，我们需要配置它。首先必须找到 OpenAIS 和 Corosync 的配置文件。检查是否存在*/etc/ais/openais.conf* 和*/etc/corosync/corosync.conf* 文件。如果它们不存在，需要拷贝配置文件：

```
$ sudo mkdir -p /etc/ais
$ cd openais-1.1.4
$ sudo cp conf/openais.conf.sample /etc/ais/openais.conf
$ sudo mkdir -p /etc/corosync
$ cd corosync-1.2.8
$ sudo cp conf/corosync.conf.sample /etc/corosync/corosync.conf
```

下一步，你需要编辑这两个文件。这里有许多选项，但最重要的，必须修改的一个是在 **totem-interface** 部分的 **bindnetaddr** 选项。这必须设置为用于与其它服务器通讯的网卡的 IP 地址：

```
totem {
        ...
        interface {
                ringnumber: 0
                bindnetaddr: 10.24.22.144
                mcastaddr: 226.94.1.1
                mcastport: 5405
        }
}
```

如需获取这些配置文件中其它选项的详细文档，请参考相关的手册页（manpages）：

```
$ man openais.conf
$ man corosync.conf
```

为了测试 OpenAIS 的基本连通性，需要启动 *aisexec* 应用程序并观察输出：

```
$ sudo aisexec -f
```

例如，如果你当启动第二个节点时观察运行 *aisexec* 的第一个节点的输出，有会看到输出反映集群现在有两个连接的节点了：

```
Nov 13 06:55:30 corosync [CLM ] CLM CONFIGURATION CHANGE
Nov 13 06:55:30 corosync [CLM ] New Configuration:
Nov 13 06:55:30 corosync [CLM ] r(0) ip(10.24.22.144)
Nov 13 06:55:30 corosync [CLM ] r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [CLM ] Members Left:
Nov 13 06:55:30 corosync [CLM ] Members Joined:
```

```
Nov 13 06:55:30 corosync [CLM ] r(0) ip(10.24.22.242)
Nov 13 06:55:30 corosync [TOTEM ] A processor joined or left the membership and a new
membership was formed.
Nov 13 06:55:30 corosync [MAIN ] Completed service synchronization, ready to provide
service.
```

如果你在获得节点互相同步上遇到了麻烦，需要检查的一件事是在节点机上的防火墙规则没有禁止用于节点通讯的多播流量。

## 14.5.3 Asterisk 配置

Asterisk 的 **res_ais** 模块有单一的配置文件，*/etc/asterisk/ais.conf*。其中的一小部分是用来使能一个 AIS 集群中的分布式设备状态的。请将下述内容放入*/etc/asterisk/ais.conf* 文件：

```
[device_state]
type = event_channel
publish_event = device_state
subscribe_event = device_state
```

我们可以通过 Asterisk CLI 命令确保这个配置被正确的载入了：

```
*CLI> ais evt show event channels
=============================================================
=== Event Channels ========================================
=============================================================
===
=== ---------------------------------------------------------
=== Event Channel Name: device_state
=== ==> Publishing Event Type: device_state
=== ==> Subscribing to Event Type: device_state
=== ---------------------------------------------------------
===
=============================================================
```

另一个 **res_ais** 模块提供的有用的 Asterisk CLI 命令用于列出 AIS 集群的成员：

```
*CLI> ais clm show members
=============================================================
=== Cluster Members =======================================
=============================================================
===
=== ---------------------------------------------------------
=== Node Name: 10.24.22.144
=== ==> ID: 0x9016180a
=== ==> Address: 10.24.22.144
=== ==> Member: Yes
=== ---------------------------------------------------------
===
```

```
=== ------------------------------------------------------------
=== Node Name: 10.24.22.242
=== ==> ID: 0xf216180a
=== ==> Address: 10.24.22.242
=== ==> Member: Yes
=== ------------------------------------------------------------
===
============================================================
```

## 14.5.4测试设备状态变化

现在你已经利用 OpenAIS 建立和配置了分布式设备状态,我们可以利用自定义设备状态做一些简单的测试，以确保设备状态在服务器间实现了通讯。我们从在 Asterisk dialplan（*/etc/asterisk/extensions.conf*）中创建一个 hint 开始：

```
[devstate_test]
exten => foo,hint,Custom:abc
```

现在，你可以从 Asterisk CLI 使用 *dialplan set global* CLI 命令改变这个自定义设备状态，然后在其它服务器上使用 *core show hints* 命令检查状态。例如，我们可以使用这个命令在一个服务器上设置状态：

```
pbx1*CLI> dialplan set global DEVICE_STATE(Custom:abc) INUSE
        -- Global variable 'DEVICE_STATE(Custom:abc)' set to 'INUSE'
```

然后，在另一个服务器上检查状态，使用命令：

```
*CLI> core show hints
-= Registered Asterisk Dial Plan Hints =-
   foo@devstatetest    : Custom:abc      State:InUse       Watchers   0
```

如果你想深入研究一下分布式设备状态改变的过程,你可以使能一些有用的调试信息。首先,使能 Asterisk 控制台调试信息输出到*/etc/asterisk/logger.conf*。然后,在 Asterisk CLI 使能调试：

```
*CLI> core set debug 1
```

当使能调试输出时，你会看到一些信息显示了 Asterisk 如何处理每个状态变化的。当设备状态在一个服务器上改变时，Asterisk 检查所有服务器上这个设备的状态信息，然后综合判定设备的状态。举例说明如下：

```
*CLI> dialplan set global DEVICE_STATE(Custom:abc) NOT_INUSE
        -- Global variable 'DEVICE_STATE(Custom:abc)' set to 'NOT_INUSE'
[Nov 13 13:27:12] DEBUG[14801]: devicestate.c:652
handle_devstate_change: Processing device state change for 'Custom:abc'
[Nov 13 13:27:12] DEBUG[14801]: devicestate.c:602
process_collection: Adding per-server state of 'Not in use' for 'Custom:abc'
[Nov 13 13:27:12] DEBUG[14801]: devicestate.c:602
process_collection: Adding per-server state of 'Not in use' for 'Custom:abc'
[Nov 13 13:27:12] DEBUG[14801]: devicestate.c:609
process_collection: Aggregate devstate result is 'Not in use' for 'Custom:abc'
[Nov 13 13:27:12] DEBUG[14801]: devicestate.c:631
process_collection: Aggregate state for device 'Custom:abc' has changed to
'Not in use'
```

```
*CLI> dialplan set global DEVICE_STATE(Custom:abc) INUSE
    -- Global variable 'DEVICE_STATE(Custom:abc)' set to 'INUSE'
[Nov 13 13:29:30] DEBUG[14801]: devicestate.c:652 handle_devstate_change:
Processing device state change for 'Custom:abc'
[Nov 13 13:29:30] DEBUG[14801]: devicestate.c:602 process_collection:
Adding per-server state of 'Not in use' for 'Custom:abc'
[Nov 13 13:29:30] DEBUG[14801]: devicestate.c:602 process_collection:
Adding per-server state of 'In use' for 'Custom:abc'
[Nov 13 13:29:30] DEBUG[14801]: devicestate.c:609 process_collection:
Aggregate devstate result is 'In use' for 'Custom:abc'
[Nov 13 13:29:30] DEBUG[14801]: devicestate.c:631 process_collection:
Aggregate state for device 'Custom:abc' has changed to 'In use'
```

## 14.6 使用XMPP

The eXtensible Messaging and Presence Protocol (XMPP)，以前（现在仍然是）以 *Jabber* 的名字而著称，是 IETF 的标准通信协议。它通常被认为是一个即时通信（IM）协议，但是它也可以用于许多其它有趣的应用。XMPP 标准基金会（XSF，XMPP Standard Foundation）的工作目标是标准化 XMPP 协议的扩展应用。其中一个扩展应用，称为 PubSub，提供了一种发布/订阅（publish/subscribe）的机制。

Asterisk 能够使用 XMPP PubSub 来发布设备状态信息。用 XMPP 实现这项功能非常好的一面是，它对于地理上分散的 Asterisk 服务器集群支持的很好。

### 14.6.1 安装

为了使用 XMPP 发布设备状态，你需要支持 PubSub 的 XMPP 服务器。已经成功的与 Asterisk 测试通过的一个这种服务器是 Tigase。（http://www.tigase.org）

Tigase 的网站说明了如何安装和配置 Tigase 服务器。我们建议你首先遵照这些说明进行安装和配置，然后回到本书继续 Asterisk 相关的部分。

在 Asterisk 中，你需要确保你已经安装了 **res_jabber** 模块。你可以通过 Asterisk CLI 检查你是否加载了这个模块：

```
*CLI> module show like jabber
Module                  Description                     Use Count
res_jabber.so           AJI - Asterisk Jabber Interface 0
1 modules loaded
```

如果你使用自定义的/etc/asterisk/modules.conf 文件，并且只列出了你指定加载的模块，你还需要检查文件系统来看看这个模块是否被编译和安装了：

```
$ ls -l /usr/lib/asterisk/modules/res_jabber.so
```

```
-rwxr-xr-x 1 root root 837436 2010-11-12 15:33 /usr/lib/asterisk/modules/res_jabber.so
```

如果你还没有安装 res_jabber 模块，你需要安装 **ikseml** 和 OpenSSL 库。然后，你需要重新编译和安装 Asterisk。这从运行 Asterisk 配置脚本开始。配置脚本负责检查系统的依赖关系，以便编连系统了解需要编连哪些模块：

```
$ cd /path/to/asterisk
$ ./configure
```

运行配置脚本之后，运行 *menuselect* 工具确保 Asterisk 编连了 **res_jabber** 模块。这个模块可以在 *menuselect* 的 *Resource Modules* 部分找到：

```
$ make menuselect
```

最后，编译和安装 Asterisk：

```
$ make
$ sudo make install
```

这是相当地快速和粗糙的编译和安装 Asterisk 的说明。如果需要获取更多关于 Asterisk 编译和安装的说明，请参考第三章。

## 14.6.2 创建XMPP账号

不幸地是，Asterisk现在还不能在XMPP服务器上注册新账号。你必须通过其它机制为每个服务器创建账号。我们测试时用的方法是使用XMPP客户端，例如Pidgin（http://www.pidgin.im ），来完成账号注册过程。当账号注册完成之后，XMPP客户端就不再需要。在后面的例子中，我们将使用下面两个账号，它们是注册在服务器 *jabber.shifteight.org*上面的：

- server1@jabber.shifteight.org/astvoip1
- server2@jabber.shifteight.org/astvoip2

## 14.6.3 Asterisk配置

*/etc/asterisk/jabber.conf* 文件需要配置在每个服务器上。我们将在这里展示配置两个服务器，但这种配置方法很容易根据需要扩展到更多的服务器。Example 14-2 展示了 Server 1 的配置文件，Example 14-3 展示了 Server 2 的配置文件。关于 *jabber.conf* 选项中与分布式设备状态相关的更多信息，参见 *configs/jabber.conf.sample* 文件，它包含在 Asterisk 源文件目录下。

*Example 14-2. jabber.conf for server1*

```
[general]
autoregister = yes
[asterisk]
type = client
serverhost = jabber.shifteight.org
pubsub_node = pubsub.jabber.shifteight.org
username = server1@jabber.shifteight.org/astvoip1
```

```
secret = mypassword
distribute_events = yes
status = available
usetls = no
usesasl = yes
buddy = server2@jabber.shifteight.org/astvoip2
```

*Example 14-3. jabber.conf for server2*

```
[general]
autoregister = yes
[asterisk]
type = client
serverhost = jabber.shifteight.org
pubsub_node = pubsub.jabber.shifteight.org
username = server2@jabber.shifteight.org/astvoip2
secret = mypassword
distribute_events = yes
status = available
usetls = no
usesasl = yes
buddy = server1@jabber.shifteight.org/astvoip1
```

## 14.6.4测试

为了确保一切工作正常，首先从每个服务器上 *jabber.conf* 配置的验证工作开始。有一对 Asterisk CLI 命令可以用在这里。第一个是 *jabber show connected* 命令，它用来验证 Asterisk 已经成功的登录到 jabber 服务器上。在第一个服务器上的输出是：

```
*CLI> jabber show connected
Jabber   Users and their status:
            User: server1@jabber.shifteight.org/astvoip1 - Connected
----
        Number of users: 1
```

同时，如果 *jabber show connected* 在第二个服务器上执行，输出是：

```
*CLI> jabber show connected
Jabber   Users and their status:
            User: server2@jabber.shifteight.org/astvoip2 - Connected
----
        Number of users: 1
```

下一个有用的验证命令是 *jabber show buddies*。这个命令允许你验证其它服务器正确的显示在你的好友列表里。它也让你可以检查其它服务器当前是否处于连接状态。如果你在第一个服务器上运行这个命令，而此时没有 Asterisk 运行在第二个服务器上，则执行这个命令的输出看起来是这样的：

```
*CLI> jabber show buddies
```

```
Jabber buddy lists
Client: server1@jabber.shifteight.org/astvoip1
        Buddy:  server2@jabber.shifteight.org
                Resource: None
        Buddy:  server2@jabber.shifteight.org/astvoip2
                Resource: None
```

然后，在第二个服务器上启动 Asterisk 并执行 jabber show buddies 命令。则输出将包含更多的信息，因为第二个服务器将看到第一个服务器在线：

```
*CLI> jabber show buddies
Jabber buddy lists
Client: server2@jabber.shifteight.org/astvoip2
        Buddy:  server1@jabber.shifteight.org
                Resource:  astvoip1
                        node:  http://www.asterisk.org/xmpp/client/caps
                        version:  asterisk-xmpp
                        Jingle capable:  yes
                Status: 1
                Priority: 0
        Buddy:  server1@jabber.shifteight.org/astvoip1
                Resource:  None
```

这时候，你应该准备好测试分发设备状态了。测试过程和通过 AIS 测试设备状态一样，我们已经在“14.5.4 测试设备状态变化”一节介绍过了。

## 14.7 Shared Line Appearances

在 Asterisk 中，Shared Line Appearances（SLA）——在业内有时也被称作 Bridged Line Appearances（BLA）——可以被使用。这个功能主要被用于满足两个应用场景：模拟一个简单按键系统，和在 PBX 上创建一个共享分机。

建立按键系统模拟是 SLA 设计的主要应用场景。在这个应用场景中，你有几条连接到 PBX 上的外线，例如模拟电话线，并且每部电话都有专用的按键对应于每条外线。你可以称这些外线为 line1，line2，和 line3，等等。

第二个主要应用场景是在你的 PBX 建立共享的分机号码。这种应用场景目前看起来非常常见。有很多理由会让你想要这个功能。一个例子是你可能希望同一个分机号码共享在在经理和她的行政助理的电话上。另一个例子是你可能希望同一个分机号码共享在同一个实验室的所有电话上。

虽然这些应用场景得到了一定程度上的支持，但是也有局限性。为了让这些特性像人们希望的那样真正工作良好，仍然有很多工作要做。这些局限性将在后面“14.7.7 局限性”一节讨论。

## 14.7.1 安装SLA应用程序

在 Asterisk 中,SLA 应用程序创建在两个关键技术上。第一个关键技术是设备状态处理,第二个关键技术是会议技术。确切的说,SLA 使用的会议技术是 **MeetMe()**应用程序。SLA 应用程序和 **MeetMe()**应用程序由同一个模块提供,所以你必须安装 **app_meetme** 模块。

你可以在 Asterisk CLI 上检查你是否已经安装了这个模块:

```
pbx*CLI> module show like app_meetme.so
Module                    Description                    Use Count
0 modules loaded
```

在这个例子中,**app_meetme** 模块没有安装。Asterisk 系统没有安装 app_meetme 模块最常见的原因是因为 DAHDI 没有安装。**MeetMe()**应用程序使用 DAHDI 来执行会议混音。一旦 DAHDI 安装(安装信息请参见第 3 章),回到 Asterisk 配置脚本,重新编译并重新安装。一旦正确安装了这个模块,你可以通过 CLI 看到:

```
*CLI> module show like app_meetme.so
Module                    Description                    Use Count
app_meetme.so             MeetMe conference bridge       0
1 modules loaded
```

一旦 **app_meetme** 模块被加载,你就可以使用 **SLAStation()**和 **SLATrunk()**应用程序了:

```
*CLI> core show applications like SLA
    -= Matching Asterisk Applications =-
          SLAStation: Shared Line Appearance Station.
           SLATrunk: Shared Line Appearance Trunk.
    -= 2 Applications Matching =-
```

## 14.7.2 配置概述

配置 SLA 必须编辑的两个主要配置文件是 */etc/asterisk/extensions.conf* 和 */etc/asterisk/sla.conf*。*sla.conf* 用于定义外线(trunks)和话机(stations)。话机可以是任意支持 SLA 的 SIP 话机。外线可以是实际的外线(译者注:如从电信局接来的模拟电话线),或者是共享 extensions,共享 extensions 将出现在两个或更多的话机上。Asterisk 的 dialplan (*extensions.conf*)像胶水一样把 SLA 配置拖到一起。Dialplan 中包括一些 extension 状态 hints,以及定义呼叫如何进入及离开 SLA 配置的 extensions。在后面的几节中,提供了针对几种应用场景的详细配置例子。

## 14.7.3 使用模拟外线的按键系统的例子

SLA 的使用伴随着最简单的配置[注2]。这个场景典型用于相当小型的部署,你可能只有几条模拟外线和少量 SIP 话机,所有这些 SIP 话机都有 line 按键关联到每条模拟外线上。在这个例子中,我们假设有两条模拟外线和四部 SIP 话机。每部 SIP 话机都有一个按键与 line1 关联,有另一个按键与 line2 关联。本节假定你已经完成了前面的配置,包括:

- 配置四个 SIP 话机。更多配置 SIP 话机的信息,请参考第五章;

- 配置两根模拟线。更多配置 Asterisk 模拟外线的信息，请参考第七章。

在这个例子中，我们将使用下面的设备名字作为 SIP 话机和模拟外线的名字。请注意修改这些名字匹配你自己的配置：

- SIP/station1
- SIP/station2
- SIP/station3
- SIP/station4

- DAHDI/1
- DAHDI/2

### 14.7.3.1 sla.conf

如我们之前提到的，*sla.conf* 包含映射设备到外线和话机的配置。在这个例子中，我们从定义两个外线开始：

```
[line1]
type = trunk
device = DAHDI/1

[line2]
type = trunk
device = DAHDI/2
```

下一步，我们建立话机的定义。我们有四个话机，每部话机都可以使用这两个外线。请注意 *sla.conf* 中的段落名称并不需要与 SIP 设备名一直，但是为了方便起见，我们在这里让它们保持一致：

```
[station1]
type = station
device = SIP/station1
trunk = line1
trunk = line2

[station2]
type = station
device = SIP/station2
trunk = line1
trunk = line2

[station3]
type = station
device = SIP/station3
trunk = line1
trunk = line2
```

```
[station4]
type = station
device = SIP/station4
trunk = line1
trunk = line2
```

这么实现话机的配置有一点重复了。Asterisk 配置文件模板可以在这里派上用场以简化配置。下面是再次配置的例子，但是这次我们使用了模板：

```
[station](!)
type = station
trunk = line1
trunk = line2

[station1](station)
device = SIP/station1

[station2](station)
device = SIP/station2

[station3](station)
device = SIP/station3

[station4](station)
device = SIP/station4
```

## 14.7.3.2 extensions.conf

这个例子要求的下一个配置文件是*/etc/asterisk/extensions.conf*。有三个 contexts 需要被配置。首先，我们需要配置 **line1** 和 **line2** contexts。当呼叫通过一条模拟外线到达时，它会进入 dialplan 中的这些 contexts 之一并执行 **SLATrunk()**应用程序。这个应用程序会振铃所有适合的话机：

```
[line1]
exten => s,1,SLATrunk(line1)
[line2]
exten => s,1,SLATrunk(line2)
```

在 dialplan 中的下一个部分是 **sla_stations** context。所有来自 SIP 话机的呼叫都将发送到这个 context。进一步地，SIP 话机应该这样配置：一旦摘机，它们立即呼叫 **station1** extension（或 **station2**，**station3**，等等，视情况而定）。如果电话上的 line1 按键按下，呼叫应该被发送到 **station1_line1** extension（或 **station2_line1**，等等）。

在任何时候，如果某个电话摘机，或者 line 按键被按下，应当立即发起一个呼叫连接到对应的模拟外线。如果外线空闲，则该模拟外线将提供一个拨号音，然后用户可以按数字键拨打电话。如果用户按下的 line 按键对应的外线已经处于 in use 状态，用户会被加入到（bridged to）这根外线上已经存在的呼叫中（译者注：多方通话）。这个 **sla_stations** context

看起来如下：

```
[sla_stations]

exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)

exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)

exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)

exten => station4,1,SLAStation(station4)
exten => station4_line1,hint,SLA:station4_line1
exten => station4_line1,1,SLAStation(station4_line1)
exten => station4_line2,hint,SLA:station4_line2
exten => station4_line2,1,SLAStation(station4_line2)
```

### 14.7.3.3 额外的电话配置任务

之前的章节讨论了用于外线和话机的 dialplan。但要记住的是，我们还需要做一些额外的工作以配合这个 dialplan 配置。首先，每部话机应该配置成一摘机就发起一个呼叫。

另一个重要的工作是配置话机的 line 按键。Asterisk 使用 extension 状态订阅来控制紧挨着 line 按键的 LED 灯。除此之外，每个 line 按键还应该被配置为速拨按键（speed-dial）。请使用下面的查检表来检查你的 line 按键配置（如何完成这些任务取决于你所使用的 SIP 话机）：

- 将按键的标签设置为 *Line1*（等等），或任何你认为合适的值；
- 设置 **stations1** 上代表 *Line1* 的按键订阅 **station1_line1** 的状态，以此类推。这是必须的，这样 Asterisk 就可以使用 SIP 话机上的 LED 灯来反映外线的状态；
- 确保 **station1** 上 Line1 按键被按下时，会发起一个向 station1_line1 extension 的呼叫，以此类推；

## 14.7.4使用SIP外线的按键系统的例子

这个例子在功能上和前一个例子是完全相同的。区别在于不是使用模拟电话线作为外线（trunks），而是使用到 SIP 供应商的连接作为外线，SIP 供应商将提供 PSTN 落地服务。关于配置 Asterisk 连接 SIP 供应商的更多信息，参见第七章。

### 14.7.4.1 sla.conf

在这个场景的 sla.conf 文件有一点复杂[注3]。你可能期望看到在外线配置中的设备是一组 SIP channel，但是我们将要使用 Local channel 来代替。这将允许我们使用一些额外的 dialplan 逻辑来处理呼叫。Local channel 的用途将在下一节讨论 dialplan 例子时展示得更清楚。下面是外线（trunk）配置：

```
[line1]
type = trunk
device = Local/disa@line1_outbound
[line2]
type = trunk
device = Local/disa@line2_outbound
```

话机的配置与上一个例子一模一样，所以让我们马上开始：

```
[station](!)
type = station
trunk = line1
trunk = line2

[station1](station)
device = SIP/station1

[station2](station)
device = SIP/station2

[station3](station)
device = SIP/station3

[station4](station)
device = SIP/station4
```

### 14.7.4.2 extensions.conf

和上一个例子一样，你需要 **line1** 和 **line2** contexts 来处理从外线上呼入的呼叫：

```
[line1]
exten => s,1,SLATrunk(line1)
;
```

```
; If the provider specifies your phone number when sending you
; a call, you will need another rule in the dialplan to match that.
;
exten => _X.,1,Goto(s,1)
[line2]
exten => s,1,SLATrunk(line2)
exten => _X.,1,Goto(s,1)
```

这个例子同样需要 **sla_stations** context。它用于处理所以从话机上发起的呼叫。这与我们上个例子的内容相同：

```
[sla_stations]
exten => station1,1,SLAStation(station1)
exten => station1_line1,hint,SLA:station1_line1
exten => station1_line1,1,SLAStation(station1_line1)
exten => station1_line2,hint,SLA:station1_line2
exten => station1_line2,1,SLAStation(station1_line2)

exten => station2,1,SLAStation(station2)
exten => station2_line1,hint,SLA:station2_line1
exten => station2_line1,1,SLAStation(station2_line1)
exten => station2_line2,hint,SLA:station2_line2
exten => station2_line2,1,SLAStation(station2_line2)

exten => station3,1,SLAStation(station3)
exten => station3_line1,hint,SLA:station3_line1
exten => station3_line1,1,SLAStation(station3_line1)
exten => station3_line2,hint,SLA:station3_line2
exten => station3_line2,1,SLAStation(station3_line2)

exten => station4,1,SLAStation(station4)
exten => station4_line1,hint,SLA:station4_line1
exten => station4_line1,1,SLAStation(station4_line1)
exten => station4_line2,hint,SLA:station4_line2
exten => station4_line2,1,SLAStation(station4_line2)
```

必须的 dialplan 的最后一部分是实现 **line1_outbound** 和 **line2_outbound** contexts。这是 SLA 应用程序向 SIP 供应商发起呼叫时需要用到的。这个配置的关键是 **DISA()**应用程序的使用。在上一个例子中，SIP 话机直接连接到模拟电话线上。这将允许上游供应商（译者注：这里指模拟电话线路供应商，如中国电信）提供信号音，处理拨号，并且完成呼叫。在这个例子中，我们使用 **DISA()**应用程序本地提供拨号音和处理拨号。一旦完整的号码拨完，这个呼叫就将转接到 SIP 供应商：

```
[line1_outbound]
exten => disa,1,DISA(no-password,line1_outbound)

; Add extensions for whatever numbers you would like to
; allow to be dialed.
```

```
;
exten => _1NXXNXXXXXX,1,Dial(SIP/${EXTEN}@myprovider)
[line2_outbound]
exten => disa,1,DISA(no-password,line2_outbound)
exten => _1NXXNXXXXXX,1,Dial(SIP/${EXTEN}@myprovider)
```

## 14.7.5共享分机号码的例子

前两个例子是用于小型按键系统仿真。在这个例子中,我们将尝试一些完全不同的东西。许多 PBX 供应商提供在多部电话上共享同一个分机号码的能力。这不是一个简单的当这个号码被呼叫时让多部电话振铃的问题:它是跟深层次的集成。共享分机号码中的 line 按键的行为类似于按键系统中 line 按键的行为。例如,你可以简单的在一个电话上 hold 住一个呼叫,而在另一个电话上重新摘起这个呼叫。同样,如果多个电话按下了用于共享分机号码的按键,它们会被桥接到同一个呼叫中。这就是为什么这项功能也常常被称为 Bridged Line Apperances(BLA)。

在前面两个例子中,我们使用了两条外线和四部话机。在这个例子中,我们将在两部话机上配置一个共享的分机号码。这个共享的分机号码将被称为 extension **5001**。

### 14.7.5.1 sla.conf

每个 SLA 应用程序的使用都需要外线(trunk)和话机(station)定义。这个例子,和前面的例子一样,将使用 **DISA()**应用程序,并且 *sla.conf* 文件看起来非常相似:

```
[5001]
type = trunk
device = Local/disa@5001_outbound

[5001_phone1]
device = SIP/5001_phone1
trunk = 5001

[5001_phone2]
device = SIP/5001_phone2
trunk = 5001
```

### 14.7.5.2 extensions.conf

所必须的 dialplan 的第一部分是定义当 PBX 上的 5001 被拨打时执行什么。通常地,拨打电话应该使用 **Dial()**应用程序。在这个例子中,我们将使用 **SLATrunk()**应用程序。它将处理振铃两部电话和将它们桥接到一起:

```
exten => 5001,1,SLATrunk(5001)
```

下一步,我们需要一个 context 用于通过这个共享 extension 打出电话。这里假设 *sip.conf* 中

的 **5001_phone1** 和 **5002_phone2** 的 **context** 选项已经设置为 **5001**:

```
[5001]
;
; This extension is needed if you want the shared extension to
; be used by default. In that case, have this extension dialed
; when the phone goes off-hook.
;
exten => 5001_phone1,1,SLAStation(5001_phone1)
;
; This is the extension that should be dialed when the 5001 key is
; pressed on 5001_phone1.
;
exten => 5001_phone1_5001,hint,SLA:5001_phone1_5001
exten => 5001_phone1_5001,1,SLAStation(5001_phone1_5001)
exten => 5001_phone2,1,SLAStation(5001_phone2)
exten => 5001_phone2_5001,hint,SLA:5001_phone2_5001
exten => 5001_phone2_5001,1,SLAStation(5001_phone2_5001)
```

最后，我们需要实现 **5001_outbound** context。它将用于在桥接线路（bridged line）上提供拨号音和处理拨号：

```
[5001_outbound]
exten => disa,1,DISA(no-password,5001_outbound)
;
; This context will also need to be able to see whatever
; extensions you would like to be reachable from this extension.
;
include => pbx_extensions
```

## 14.7.6 附加配置

在*/etc/asterisk/sla.conf* 文件中还有一些没有用在本章任何例子中的可选配置参数。为了让你了解其他可以配置的行为，我们在本节讨论下这些选项。这个文件的**[general]**部分是预留给全局配置选项的。目前，在这部分只有一个选项：

attemptcallerid = *yes*

这个选项指定了 SLA 应用程序是否传递主叫号码（caller ID）信息。它默认配置为 **no**。如果这个选项使能，在有些情况下话机的显示可能和你期望的不同。

在前面例子中的外线（trunk）定义只指定了设备的类型。这里是一些可用于详细说明外线的附加选项：

autocontext = line1

如果这个选项被设置，Asterisk 将使用这个名字自动创建一个 dialplan context。这个 context 将包含一个带有适合这个 trunk 的合适参数执行 SLATrunk()应用程序的 **s** extension。在默认情况，所有 dialplan 入口都必须手工创建。

ringtimeout = 20

这个选项允许你指定一个秒数，来定义这个 trunk 上的呼入呼叫在 SLATrunk()应用程序退出并认为这是一个无应答呼叫前振铃的时间。在默认情况，这个选项不被设置。

barge = no

这个选项指定是否允许其它话机通过按下同样的 line 按键加入这个 trunk 上正在处理的呼叫。在默认情况下是允许的。

hold = private

这个选项指定这个 trunk 的 hold 权限。如果这个选项设置为 **open**，任何话机都可以将这个 trunk 置为 hold，而且任何其它话机都被允许恢复通话。如果这个选项设置为 **private**，只有将这个 trunk 置为 hold 的话机允许恢复通话。这个选项默认设置为 **open**。

当我们在前面的例子中定义话机（station）时，我们只提供了 **type**, **device**, 和一系列 trunks。然而，话机（station）定义也可以接受一些附加的配置选项。列表如下：

autocontext = sla_stations

如果这个选项被指定，Asterisk 会在这里指定的 context 自动创建一个用于处理从这个话机发起的呼叫的 extension。这个选项默认是关闭的，这就意味着所有的 extensions 必须手工指定。

ringtimeout = 20

指定一个以秒为单位的超时时间，它定义了这个话机在被认为是无应答之前的振铃时间。默认不设置振铃超时。

ringdelay = 5

以秒为单位的话机振铃时延。如果指定了延迟，这个话机在从呼叫到达这个共享线路后知道指定的秒数前不会开始振铃。默认不设置振铃时延。

hold = private

Hold 权限也可以按照话机指定。如果这个选项被设置为 private，任何被这部话机置入 hold 状态的 trunks 也只能由这部话机恢复。这个选项默认设置为 **open**。

trunk = line1,ringtimeout=20

可以针对指定的 trunk 设置振铃超时。

trunk = line1,ringdelay=5

可以针对指定的 trunk 设置振铃时延。

### 14.7.7 局限性

虽然 Asterisk 会是许多事情变得容易，但是 SLA 并不是其中之一。尽管这个功能打算模仿一个简单的特性，但是为了让它工作而必须的配置则相当复杂。对于只想实现一个简单按键系统的 Asterisk 新手来说，他必须学习大量复杂的 Asterisk 和 SIP 话机概念才能使这个系统工作。

另一个必须进行一些开发工作才能与 SLA 无缝的配合工作的是主叫号码（caller ID）。在我们写作本书时，Asterisk 还没有适当的架构来处理在呼叫过程中更新主叫号码信息。基于这些功能是怎么实现的，对于在话机上显示有用的信息来说，这个架构是必须的。这个架构在 Asterisk 1.8 中已经存在了，但是 SLA 应用程序还没有更新来使用它。最终的结果是你要么完全不使用主叫号码信息，要么使能这个功能并且理解当主叫号码信息在呼叫过程中变化时，话机上显示的信息并不总是对的。

另一个限制，主要和使用共享分机号码有关，就是呼转（transfers）功能此时不能工作。主要的原因是呼转功能通常包含把呼叫置入 hold 状态一小会时间。但是呼叫保留（call hold）在 SLA 中是以特殊的方式处理的，因此被保留的呼叫并不能被发起保留的话机控制。这将中断呼转的处理。

总的来说，SLA 不一定容易设置，而且它有一些重大的局限性。但就像老话说的，如果你认为它适合你的需要，那就试试吧。

## 14.8 总结

本章讨论了 Asterisk 中设备状态处理的许多方面。我们从讨论设备状态（device states）和分机状态（extension states）的核心概念开始，然后以此为基础展开。我们讨论了 SIP 话机怎么实现状态订阅，创建自定义状态的工具，以及两种用于多台服务器之间发布状态的机制。最后，我们讨论了 Asterisk 中的一个特性，Shared Line Appearances，它高度依赖于 Asterisk 的设备状态架构来运行。

**注释：**

注1. 有些人喜欢称其为“闪灯”（”blinky lamps” or “blinky lights”）。极客和他们的 LEDs……

注2. 无可否认地，SLA 不需要配置，所以说 SLA 的配置是最简单的。

注3. 阅读：《黑客》