第十三章 自动呼叫分配（ACD）

目录

自动呼叫分配（ACD - Automatic Call Distribution），或者称为呼叫排队（call queuing），为 PBX 提供了为一群用户的来电排队处理的能力：它将多个来电呼叫转接到呼叫保留状态，并为每个呼叫分配一个排名，这个排名用于决定来电被分配给可用坐席的顺序（典型的，采用先进先出）。当某个坐席变为可用时，队列中排在最前面的呼叫会被转给这个坐席处理，并且其它呼叫顺次向前移动一位。

如果你曾经给某些组织打电话时听到过“所有坐席忙，”这样的信息，就意味着你已经有了 ACD 的使用经验。对拨打电话的人来说，ACD 的优点在于他们不必反复拨打以尝试接通；而对于使用了 ACD 的组织来说，他们将能为客户提供更好的服务，并且可以临时处理一下同时来电数量多于坐席数的情况。

目前有两种呼叫中心：呼入型和呼出型。ACD 相关的技术用于处理呼入型呼叫中心，而预拨号器（Predictive Dialer）相关的技术用于处理呼出型呼叫中心。在本书中，我们主要集中讨论呼入型呼叫中心。

我们都有过由设计和管理拙劣的队列带来的糟糕体验：忍受着难听的 Hold Music，令人厌恶的等待时间，每 20 秒重复一遍的毫无意义的信息告诉你“你的来电时如何重要”，尽管你已经等待了 30 分钟并且重复听这个信息以至于能够背下来了。从客户服务的观点看，队列的设计可能是你电话系统中最重要的一个方面。与使用自动话务员一样，首先要牢牢记住的是，来电者对停留在队列里毫无兴趣。他们打来电话，是因为他们希望和你对话。你所有的设计决定必须以这样一个重要事实为中心：人们是希望与其他人对话，而不是和你的电话系统对话。注 2

本章的目标是教给你如何创建和设计一个队列，从而可以将呼叫者尽量快速而不费力的转接给合适的目标。

在本章中，我们将交替使用术语 *queue members* 和 *agents*。除非我们讨论的 agents 是通过 chan_agent （使用 **AgentLogin()**）登录的，我们都是在讨论通过 **AddQueueMember()** 或命令行（我们将在本章讨论这些命令）增加的 *queue members*。你只需要知道，虽然在 Asterisk 中 agent 和 queue member 是有区别的，但是我们将简单的使用术语 agent 来描述 **Queue()** 调用的终端（endpoint）。

13.1 创建简单的ACD队列

作为开始，我们首先创建一个简单的 ACD 队列。它将接收呼叫者，并将他们分配到几个队列中。

在 Asterisk 中，术语 *member* 指队列中可以被拨叫的一个终端，例如 SIP/0000FFFF0001。术语 *agent* 技术上是指用于拨叫终端的 Agent channel。不幸地是，Agent channel 是在 Asterisk 中废弃了的技术，因为它的灵活性非常有限，而且容易产生一些意想不到的错误，这些错误非常难于诊断和解决。我们不讨论使用 chan_agent 的情况，所以需要了解的是，我们将使用术语 member 指电话机设备，而使用术语 agent 指使用电话机的人。由于这两者单独出现并无意义，所以 member 或 agent 任何一个术语都可以用来表示电话机和使用电话的人两者。

我们将在 *queues.conf* 文件中创建队列，并且通过 Asterisk 控制台手工增加队列成员。在本章 13.2 队列的坐席成员一节，我们将看到如何创建一个 dialplan 来动态添加和删除队列成员（以及暂停和恢复）。

首先的步骤是在/etc/asterisk 配置目录中创建你的 *queues.conf* 文件：

```
$ cd /etc/asterisk/
$ touch queues.conf
```

然后在其中填入下述配置，这将创建两个命名为**[sales]**和**[support]**的队列。你可以将这两个队列命名为任何你希望的名字，但在本书中我们将使用这两个名字。所以，如果你使用了其它的名字，在后续阅读本书时请记得这一点：

```
[general]
autofill=yes                ; distribute all waiting callers to available members
shared_lastcall=yes         ; respect the wrapup time for members logged into more
                            ; than one queue
[StandardQueue](!)          ; template to provide common features
musicclass=default          ; play [default] music
strategy=rrmemory           ; use the Round Robin Memory strategy
joinempty=no                ; do not join the queue when no members available
leavewhenempty=yes          ; leave the queue when no members available
ringinuse=no                ; don't ring members when already InUse (prevents
                            ; multiple calls to an agent)
[sales](StandardQueue)      ; create the sales queue using the parameters in the
                            ; StandardQueue template
[support](StandardQueue)    ; create the support queue using the parameters in the
                            ; StandardQueue template
```

[general]部分定义了默认的行为和全局选项。我们仅仅在**[general]**部分指定了两个选项，这是因为在这个地方内建的默认值已经可以很好的满足我们的需要。

第一个选项是 **autofill**，它告诉队列将所有等待的呼叫立即分配给所有可用的坐席。在早期的版本中，Asterisk 每次只转接一个呼叫，这就意味着当 Asterisk 向一个坐席转接时，所有其他的呼叫都会进入呼叫保持状态（即使有可用坐席的情况）直到前一个呼叫转接成功（很明显，早期版本的 Asterisk 在大型、繁忙的队列应用中会导致瓶颈）。除非你有特别的向后兼容的需求，*这个选项应当永远被设为* ***yes***。

在 *queues.conf* 中**[general]**部分的第二个选项是 **shared_lastcall**。当我们使能 **shared_lastcall** 时，对于登录到多个队列的坐席，对于任何一个刚结束的电话，在所有队列中都会开始计算“结束时间”（wrapup time）注 3，以避免某个队列将呼叫转接给刚刚结束了另一个队列转接的呼叫，还处于“结束时间”的坐席。如果这个选项设置为 **no**，则“结束时间”的计算仅仅对本队列有效，这将导致一个刚刚接听了 support 队列的电话尚处于“结束时间”的坐席，仍然会收到 sales 队列转接的呼叫。这个选项应该一直被设置为 **yes**（默认值）。

下一部分，**[StandardQueue]**是我们打算应用到 sales 和 support 队列的模板（我们通过**(!)**声明其为模板）。和我们在 *musiconhold.conf* 文件中的配置一样，我们定义 **musicclass** 为 **default** 呼叫保持音乐。我们将 **strategy** 配置为 **rrmemory**，代表循环（Round-Robin）存储。**rrmemory** 策略的工作原理是将队列中按顺序排列的坐席循环起来，它跟踪接听了上一个呼叫的坐席，然后将下一个呼叫转接给下一个坐席。当达到最后一个坐席时，它返回队列顶部的坐席（当坐席登录时，他们被加入到队列的尾部）。我们设置 **joinenpty** 为 **no**，是因为将呼叫放入一个没有坐席资源的队列是一种坏的做法。

出于测试目的，你可以将这个值设置为 **yes**，但我们不推荐你在正式产品中这么做，除非你使用队列的目的不是转接来电给坐席。没有人愿意等待在一条无人处理的线路上。

leavewhenempty 选项用于控制当没有可用的坐席能够处理呼叫时，呼叫是否应该离开 **Queue()**应用程序继续执行 dialplan。我们将这个选项配置为 **yes** 是因为等待在一条无人处理的线路上是毫无意义的。

从商业的观点来看，你应该告诉你的坐席人员在当天登出系统前要处理完队列中所有的呼叫。如果你发现当一天快要结束时有太多的呼叫在队列中，你可能希望考虑延长某些坐席人员的工作时间以处理他们。否则，当他们第二天带着情绪重新打来时，只会给你带来更大的压力。
另一个选择是使用 **GotoIfTime()**在临近下班时间时将呼叫转接到语音信箱，或者 dialplan 中的其它合适地点。

最后，我们将 **ringinuse** 设置为 **no**，它告诉 Asterisk 不要振铃已经处于振铃状态的坐席。将 **ringinuse** 设置为 **no** 的目的是避免多个呼叫被转接到来自一个或多个队列的同一个坐席。

你可能注意到 joinempty 和 leavewhenempty 都是查找或者队列中已经没有登录的坐席，或者所有的坐席都失效。处于振铃（**Ringing**）或使用中（**InUse**）状态的坐席不被认为是失效的，所以当 **joinempty-no** 和/或 **leavewhenempty=yes** 时不会阻止呼叫加入队列或导致他们离开队列

一旦你完成了你的 *queues.conf* 文件的配置，你可以保存它并通过 Asterisk CLI 重载 *app_queue.so* 模块：

```
$ asterisk -r
*CLI> module reload app_queue.so
    -- Reloading module 'app_queue.so' (True Call Queueing)
```

然后检查一下你的队列是否装载到内存了：

```
localhost*CLI> queue show
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    No Members
    No Callers
sales has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
    No Members
    No Callers
```

现在，你已经创建好了队列，下一步你需要配置你的 dialplan 以允许呼叫进入你的队列。

在 *extensions.conf* 文件中增加如下 dialplan 逻辑：

```
[Queues]
exten => 7001,1,Verbose(2,${CALLERID(all)} entering the support queue)
same => n,Queue(support)
same => n,Hangup()

exten => 7002,1,Verbose(2,${CALLERID(all)} entering the sales queue)
same => n,Queue(sales)
same => n,Hangup()

[LocalSets]
include => Queues     ; allow phones to call queues
```

我们将 **Queue** context 包含在 **LocalSets** context 中，所以我们的电话机可以呼叫我们创建的这个队列。在第 15 章，我们将定义访问这些队列的菜单项。保存这些变化到你的 *extensions.conf* 文件，然后通过 *dialplan reload* 命令重载 dialplan。

如果你此时拨打分机 **7001** 和 **7002**，将以得到下面这样的输出而结束：

```
 -- Executing [7001@LocalSets:1] Verbose("SIP/0000FFFF0003-00000001",
   "2,"Leif Madsen" <100> entering the support queue") in new stack
== "Leif Madsen" <1--> entering the support queue
 -- Executing [7001@LocalSets:2] Queue("SIP/0000FFFF0003-00000001",
   "support") in new stack
   [2011-02-14 08:59:39] WARNING[13981]: app_queue.c:5738 queue_exec:
   Unable to join queue 'support'
 -- Executing [7001@LocalSets:3]
   Hangup("SIP/0000FFFF0003-00000001", "") in new stack
```

```
    == Spawn extension (LocalSets, 7001, 3) exited non-zero on
       'SIP/0000FFFF0003-00000001'
```

你现在还不能加入这个队列，因为在这个队列中没有坐席来应答呼叫。因为我们已经在 *queues.conf* 中配置 **joinempty=no** 和 **leavewhenempty=yes**，所以呼叫不会加入到队列中。（这是一个很好的实验 *queues.conf* 中的 **joinempty** 和 **leavewhenempty** 选项的机会，这将有助于更好的理解这两个选项对队列的影响。）

在下一节中，我们将说明如何在队列中增加坐席成员（以及其它坐席和队列的交互方法，例如暂停/恢复）。

13.2 队列的坐席成员

如果没有人应答队列中的呼叫，队列是没有什么用处的。所以我们需要一种方法允许坐席人员登录到队列中应答呼叫。有多种方法可以做到这一点，我们将教你如何手动（作为管理员）和自动（作为坐席人员）向队列中增加坐席成员。我们将从 Asterisk CLI 方法开始，这种方法允许你很容易的以最小的 dialplan 修改向队列增加坐席成员用于测试。接着我们将展开讨论，教你如何增加 dialplan 逻辑使得坐席人员可以登入或登出队列，以及在他们登录的队列中暂停或恢复他们自己。

13.2.1通过CLI控制队列成员

我们可以通过 Asterisk CLI 命令 *queue add* 向任何有效的队列增加坐席成员。*queue add* 命令的格式是（注：应该在同一行输入）：

*CLI> **queue add member <*channel*> to <*queue*> [[[penalty <*penalty*>] as <*membername*>] state_interface <*interface*>]**

其中<*channel*>是我们希望加入队列的 channel，例如 **SIP/0000FFFF0003**，<*queue*>是像 **support** 或 **sales** 这样的队列的名字——任何存在于*/etc/asterisk/queues.conf* 中的队列的名字。我们将暂时忽略<penalty>选项，但是我们会在本章“13.5 高级队列”一节讨论它（penalty 用于控制队列中坐席成员的排名，这对于登录到多个队列的坐席非常重要）。我们可以定义<*membername*>来提供队列登录的细节。**state_interface** 是需要我们仔细研究的一个选项。因为这个选项在 Asterisk 中对于队列及其成员的所有方面都非常重要，以至于我们要用几个小节来讨论它。所以，你可以继续先阅读本章的“13.2.4 设备状态简介”一节，然后再回到这里继续阅读。

现在，你已经在 sip.conf 中增加了 **callcounter=yes**（在我们后续的例子中，我们都将使用 SIP channels），让我们看看如何通过 Asterisk CLI 在你的队列中增加成员。（译者注：关于 **callcounter** 的说明要参看本章后面的“设备状态简介”一节）。

在 **support** 队里中增加队列成员，可以通过 *queue add member* 命令实现：

```
*CLI> queue add member SIP/0000FFFF0001 to support
Added interface 'SIP/0000FFFF0001' to queue 'support'
```

然后通过查询队列状态来确认新成员已经被增加：

```
*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0000FFFF0001 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

删除队列成员，你可以使用 *queue remove member* 命令：

```
*CLI> queue remove member SIP/0000FFFF0001 from support
Removed interface 'SIP/0000FFFF0001' from queue 'support'
```

当然，你可以再次使用 *queue show* 命令来确认目标成员已经被删除。

我们还可以从 Asterisk 控制台暂停和恢复队列成员，通过命令 *queue pause member* 和 *queue unpause member* 命令。它们和我们之前使用的命令格式差不多：

```
*CLI> queue pause member SIP/0000FFFF0001 queue support reason DoingCallbacks
paused interface 'SIP/0000FFFF0001' in queue 'support' for reason 'DoingCallBacks'

*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0000FFFF0001 (dynamic) (paused) (Not in use) has taken no calls yet
   No Callers
```

通过增加队列成员的暂停原因，例如 lunchtime，你可以保证你的队列日志包含一些可能有用的附加信息。下面是如何恢复队列成员：

```
*CLI> queue unpause member SIP/0000FFFF0001 queue support reason off-break
unpaused interface 'SIP/0000FFFF0001' in queue 'support' for reason 'off-break'

*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0000FFFF0001 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

在实际应用中，CLI 一般不是控制队列中坐席状态的最佳方式。替代地，我们利用 dialplan 应用程序允许坐席人员通知队列它们的可用状态。

13.2.2通过dialplan逻辑控制队列成员

在由坐席人员组成的呼叫中心中，最常见的做法是让坐席人员自己在上班和下班时登入及登出队列（或者当他们去吃饭，去洗手间，或因为其它原因不能接听电话时登出队列）。要做到这一点，我们需要使用下述 dialplan 应用程序：

- **AddQueueMember()**
- **RemoveQueueMember()**

当登入队列之后，还可能出现坐席人员临时不能接听电话的情况。下面的应用程序用于这个目的：

- **PauseQueueMember()**
- **UnpauseQueueMember()**

也许这样理解这些应用程序更加容易：加入/离开应用程序用于登入和登出，而暂停/恢复用于坐席短时间不可用。区别是非常简单的，暂停/恢复是在没有登出队列的情况下将成员状态设置为不可用和可用。这对于生成报表十分有用（如果一个成员是暂停的，队列管理员会看到她已经登录到队列，但只是暂时不能接听电话）。如果你不能确定应该用哪个运管用程序，我们推荐你在任何情况下都使用加入/离开。

使用暂停（Pause）和恢复（Unpause）

使用暂停和恢复大概算一种偏好。在某些环境下，这组选项会被用于在上班时间导致坐席不可用的任何活动（例如午饭时间或进行一些与接听电话无关的工作）。然而，在大部分呼叫中心，如果一个坐席人员不在她的电话旁准备接听电话的时候，她根本就不应当登入系统，即使她只是离开她的座位几分钟（例如上洗手间）。

有些主管喜欢利用加入/离开和暂停/恢复配置作为某种打卡钟，这样他们可以跟踪他的职员何时到达开始工作和何时结束工作离开，以及他们在座位上待了多长时间和休息了多长时间。我们并不认为这是一种合理的实践，因为这些应用程序的目的是通知队列坐席的可用状态，而不是跟踪员工的活动。

在这里要非常重视的一件事是 *queues.conf* 中 **joinempty** 的设置，我们之前讨论过这个选项。如果坐席是暂停的，她仍旧被认为是登录到队列中的。让我们假设接近下班的时候，某个坐席在几个小时前将自己设置为暂停状态。所有其他的坐席都登出系统并回家了。这时来了个电话。队列会注意到仍旧有一个坐席登录在队列中，然后就将这个来电加入到队列，尽管事实上此时已经没有人在处理这个队列了。结果是这个来电会被永远保留在这个无人处理的队列中。

简而言之，没有坐在座位旁边准备接听电话的坐席应该登出系统。暂停/恢复应该仅仅用于短暂的不可用状态（如果是真的的话）。如果你希望利用你的电话系统作为打卡钟，在 Asterisk 中有很多更好的办法，但是队列成员应用程序绝对不是我们推荐的方式。

让我们创建一些简单的 dialplan 逻辑来让我们的坐席可以指示队列他们的状态。我们将使用 **CUT()** dialplan 函数从来电中获取 channel 名，这样队列就可以知道哪个 channel 登入了队列。

我们已经建立了一个 dialplan 来演示登入及登出队列，以及暂停和恢复队列成员的的简单过程。我们仅对我们之前在 *queues.conf* 文件中定义的 **support** 队列操作。**AddQueueMember()**，**RemoveQueueMember()**，**PauseQueueMember()**，以及 **UnpauseQueueMember()** 应用程序设置的 channel 变量将在队列成员完成操作后通过 Playback()播放出来，以通知队列成员让他们知道他们是否成功的执行了登入/登出，或者暂停/恢复操作。

```
[QueueMemberFunctions]
exten => *54,1,Verbose(2,Logging In Queue Member)
    same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(peername)})
```

```
        same => n,AddQueueMember(support,${MemberChannel})

; ${AQMSTATUS}
        ; ADDED
        ; MEMBERALREADY
        ; NOSUCHQUEUE
exten => *56,1,Verbose(2,Logging Out Queue Member)
        same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(peername)})
        same => n,RemoveQueueMember(support,${MemberChannel})

; ${RQMSTATUS}:
        ; REMOVED
        ; NOTINQUEUE
        ; NOSUCHQUEUE
exten => *72,1,Verbose(2,Pause Queue Member)
        same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(peername)})
        same => n,PauseQueueMember(support,${MemberChannel})

; ${PQMSTATUS}:
        ; PAUSED
        ; NOTFOUND
exten => *87,1,Verbose(2,Unpause Queue Member)
        same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(peername)})
        same => n,UnpauseQueueMember(support,${MemberChannel})

; ${UPQMSTATUS}:
        ; UNPAUSED
        ; NOTFOUND
```

13.2.3 自动登入和登出多个队列

一个坐席人员属于多个队列的情况是非常常见的。与其为登录不同队列配置多个独立的 extension（或者询问坐席人员他们希望登录哪个队列），不如利用 Asterisk 数据库（**astdb**）来为每个坐席存储他所属于的队列信息，然后由 Asterisk 依次将坐席自动登录到他们所属的队列。

为了让这段代码能工作，需要通过 Asterisk CLI 对 AstDB 进行类似下面例子中的操作。例如，下面的例子将存储成员 **0000FFFF0001** 属于 **support** 和 **sales** 两个队列的信息：

```
*CLI> database put queue_agent 0000FFFF0001/available_queues support^sales
```

下面的 dialplan 代码用于示例如何将队列成员自动加入到 **support** 和 **sales** 队列中。我们已经定义了一个例程，该例程用于建立了三个 channel 变量（**MemberChannel**，**MemberChanType**，**AvailableQueues**）。然后这些 channel 变量被用于登入（*54），登出（*56），暂停（*72），

和恢复（*87）。这些 extensions 总的每个都使用 **subSetupAvailableQueies** 例程来设置这些 channel 变量和验证 AstDB 中包含的队列列表：

```
[subSetupAvailableQueues]
;
; This subroutine is used by the various login/logout/pausing/unpausing routines
; in the [ACD] context. The purpose of the subroutine is centralize the retrieval
; of information easier.
;
exten => start,1,Verbose(2,Checking for available queues)
; Get the current channel's peer name (0000FFFF0001)
    same => n,Set(MemberChannel=${CHANNEL(peername)})
; Get the current channel's technology type (SIP, IAX, etc)
    same => n,Set(MemberChanType=${CHANNEL(channeltype)})
; Get the list of queues available for this agent
    same => n,Set(AvailableQueues=${DB(queue_agent/${MemberChannel}/available_queues)})
; if there are no queues assigned to this agent we'll handle it in the
; no_queues_available extension
    same => n,GotoIf($[${ISNULL(${AvailableQueues})}]?no_queues_available,1)
    same => n,Return()
exten => no_queues_available,1,Verbose(2,No queues available for agent${MemberChannel})
; playback a message stating the channel has not yet been assigned
    same => n,Playback(silence/1&channel&not-yet-assigned)
    same => n,Hangup()
[ACD]
;
; Used for logging agents into all configured queues per the AstDB
;
;
; Logging into multiple queues via the AstDB system
exten => *54,1,Verbose(2,Logging into multiple queues per the database values)
; get the available queues for this channel
    same => n,GoSub(subSetupAvailableQueues,start,1())
    same => n,Set(QueueCounter=1) ; setup a counter variable
; using CUT(), get the first listed queue returned from the AstDB
    same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
; While the WorkingQueue channel variable contains a value, loop
    same => n,While($[${EXISTS(${WorkingQueue})}])
; AddQueueMember(queuename[,interface[,penalty[,options[,membername[,stateinterface]]]]])
; Add the channel to a queue, setting the interface for calling
; and the interface for monitoring of device state
;
; *** This should all be on a single line
    same => n,AddQueueMember(${WorkingQueue},${MemberChanType}/
${MemberChannel},,,${MemberChanType}/${MemberChannel})

        same => n,Set(QueueCounter=$[${QueueCounter} + 1]) ; increase our counter
; get the next available queue; if it is null our loop will end
        same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
        same => n,EndWhile()
; let the agent know they were logged in okay
        same => n,Playback(silence/1&agent-loginok)
        same => n,Hangup()
exten => no_queues_available,1,Verbose(2,No queues available for ${MemberChannel})

        same => n,Playback(silence/1&channel&not-yet-assigned)
        same => n,Hangup()
; -------------------------
; Used for logging agents out of all configured queues per the AstDB
exten => *56,1,Verbose(2,Logging out of multiple queues)
; Because we reused some code, we've placed the duplicate code into a subroutine
        same => n,GoSub(subSetupAvailableQueues,start,1())
        same => n,Set(QueueCounter=1)
        same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
        same => n,While($[${EXISTS(${WorkingQueue})}])
        same => n,RemoveQueueMember(${WorkingQueue},${MemberChanType}/${MemberChannel})
        same => n,Set(QueueCounter=$[${QueueCounter} + 1])
        same => n,Set(WorkingQueue=${CUT(AvailableQueues,^,${QueueCounter})})
        same => n,EndWhile()
        same => n,Playback(silence/1&agent-loggedoff)
        same => n,Hangup()
; -------------------------

; Used for pausing agents in all available queues
exten => *72,1,Verbose(2,Pausing member in all queues)
        same => n,GoSub(subSetupAvailableQueues,start,1())
; if we don't define a queue, the member is paused in all queues
        same => n,PauseQueueMember(,${MemberChanType}/${MemberChannel})
        same => n,GotoIf($[${PQMSTATUS} = PAUSED]?agent_paused,1:agent_not_found,1)
exten => agent_paused,1,Verbose(2,Agent paused successfully)
        same => n,Playback(silence/1&unavailable)
        same => n,Hangup()
; -------------------------
; Used for unpausing agents in all available queues
exten => *87,1,Verbose(2,UnPausing member in all queues)
        same => n,GoSub(subSetupAvailableQueues,start,1())
; if we don't define a queue, then the member is unpaused from all queues
        same => n,UnPauseQueueMember(,${MemberChanType}/${MemberChannel})
        same => n,GotoIf($[${PQMSTATUS} = PAUSED]?agent_unpaused,1:agent_not_found,1)
exten => agent_unpaused,1,Verbose(2,Agent paused successfully)
        same => n,Playback(silence/1&available)
        same => n,Hangup()
; -------------------------

;Used by both pausing and unpausing dialplan functionality
exten => agent_not_found,1,Verbose(2,Agent was not found)
        same => n,Playback(silence/1&cannot-complete-as-dialed)
```

你可以进一步改善这些登入和登出例程以实现每次用到 **AddQueueMember()**和 **RemoveQueueMember()**时设置 **AQMSTATUS** 和 **RQMSTATUS** channel 变量。举例来说，你可以建立一个标记，通过设置这个标注来使队列成员了解到他还没有加入队列，或者增加录音或文本装语音系统来播放出现问题的队列信息。或者，如果你通过 Asterisk Manager Interface 来实现监控的话，你可以使用一个弹屏，或者使用 **JobberSend()**通过一个短信息通知队列成员。

13.2.4设备状态简介

在 Asterisk 中的设备状态用于通知多个应用程序关于你的设备当前是否正在使用中。这对于队列来说特别重要，因为我们不会希望将呼叫转接给一个已经在接听电话的坐席。设备状态是由 channel 模块控制的，而且在 Asterisk 中只有 **chan_sip** 能做出合适的处理。当队列查询设备状态时，它首先询问 channel 驱动（例如，chan_sip）。如果某个 channel 不能直接提供设备状态信息（例如 chan_iax2），队列查询 Asterisk 内核来确定设备状态，这通过搜索当前在处理的所有 channels 来实现。

不幸地是，简单的要求内核去搜索活跃的 channels 是不精确的，所以对于队列来说，通过 **chan_sip** 以外的方法来获得设备状态的方法可靠性要差一些。我们将在本章的“高级队列”一节探索一些在其它类型 channels 上控制呼叫的方法，但是现在我们将集中讨论 SIP channels，它不需要复杂的设备状态要求。关于设备状态的更进一步信息，请参阅第十四章。

为了在 Asterisk 中正确的确定设备的状态，我们需要在 *sip.conf* 中使能呼叫计数器（call counters）。通过使能呼叫计数器，我们告诉 Asterisk 去跟踪设备的活跃呼叫，所以这些信息会被报告给 channel 模块，从而设备状态可以被精确的反映给队列。首先，让我们看一下没有配置 **callcounter** 选项时的情况：

```
*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
   Members:
      SIP/0000FFFF0001 (dynamic) (Not in use) has taken no calls yet
   No Callers
```

现在假设我们的 dialplan 中配置了分机 **555** 调用 **MusicOnHold()**。如果我们在没有使能 call counters 的情况下拨打这个分机，通过 Asterisk CLI 查询 **support** 队列（**SIP/0000FFFF0001** 是一个成员）将看到类似下面这样的信息：

```
-- Executing [555@LocalSets:1] MusicOnHold("SIP/0000FFFF0001-00000000",
  "") in new stack
-- Started music on hold, class 'default', on SIP/0000FFFF0001-00000000
*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
  Members:
     SIP/0000FFFF0001 (dynamic) (Not in use) has taken no calls yet
  No Callers
```

请注意尽管分机 **555** 因为正在通话中而应该被标记为 **In Use**，但是当我们查询这个队列状态时它并没有这么显示。这很明显是个问题，因为这个队列会以为这个设备是空闲的，尽管它实际上已经在进行一个通话了。

为了改正这个问题，我们需要在 *sip.conf* 文件的**[general]**部分增加 **callcounter=yes**。我们也可以指定这个配置给某个终端（因为它是一个终端级别的配置选项）；然而，这实在是一个你应该配置给所有可能加入一个队列的终端的选项，所以通常最佳的做法是将这个选项配置在**[general]**部分（也可以配置在一个被所有加入队列的终端引用的模板中）。

像下面这样编辑你的 *sip.conf* 文件：

```
[general]
context=unauthenticated          ; default context for incoming calls
allowguest=no                    ; disable unauthenticated calls
srvlookup=yes                    ; enabled DNS SRV record lookup on outbound calls
udpbindaddr=0.0.0.0              ; listen for UDP request on all interfaces
tcpenable=no                     ; disable TCP support
callcounter=yes                  ; enable device states for SIP devices
```

然后重载 chan_sip 模块并再次测试：

```
*CLI> sip reload
Reloading SIP
  == Parsing '/etc/asterisk/sip.conf': == Found
```

现在，当这个设备处于通话中时，它将显示 **In Use** 状态：

```
  == Parsing '/etc/asterisk/sip.conf': == Found
  == Using SIP RTP CoS mark 5
     -- Executing [555@LocalSets:1] MusicOnHold("SIP/0000FFFF0001-00000001",
     "") in new stack
     -- Started music on hold, class 'default', on SIP/0000FFFF0001-00000001
*CLI> queue show support
support has 0 calls (max unlimited) in 'rrmemory' strategy
(0s holdtime, 0s talktime), W:0, C:0, A:0, SL:0.0% within 0s
  Members:
     SIP/0000FFFF0001 (dynamic) (In use) has taken no calls yet
No Callers
```

简单的说，**Queue()**需要知道设备的状态以恰当的管理呼叫分配。在 *sip.conf* 中的 **callcounter** 选项是一个正确运行的队列的必要组成部分。

13.3 queues.conf 配置文件

我们之前已经提到过 *queues.conf* 文件，但是这个文件中有太多的选项，我们列表整理如下：

Table 13-1. Available options for [general] section of queues.conf

Options	Available values	Description
`persistentmembers`	yes, no	Set this to `yes` to store dynamically added members to queues in the AstDB so that they can be re-added upon Asterisk restart.
`autofill`	yes, no	With `autofill` disabled, the queue application will attempt to deliver calls to agents in a serial manner. This means only one call is attempted to be distributed to agents at a time. Additional callers are not distributed to agents until that caller is connected to an agent. With `autofill` enabled, callers are distributed to available agents simultaneously.
`monitor-type`	`MixMonitor`, *`<unspecified>`*	If you specify the value `MixMonitor` the `MixMonitor()` application will be used for recording calls within the queue. If you do not specify a value or comment the option out, the `Monitor()` application will be used instead.
`updatecdr`	yes, no	Set this to `yes` to populate the `dstchannel` field of the CDR records with the name of a dynamically added member on answer. The value is set with the `AddQueueMember()` application. This option is used to mimic the behavior of `chan_agent` channels.
`shared_lastcall`	yes, no	This value is used for members logged into more than one queue to have their last call be the same across all queues, in order for the queues to respect the wrap up time of other queues.

Table 13-2. Available options for defined queues in queues.conf

Options	Available values	Description
`musicclass`	Music class as defined by *musiconhold.conf*	Sets the music class to be used by a particular queue. You can also override this value with the `CHANNEL(musicclass)` channel variable.
`announce`	Filename of the announcement	Used for playing an announcement to the agent that answered the call, typically to let him know what queue the caller is coming from. Useful when the agent is in multiple queues, especially when set to auto-answer the queue.
`strategy`	`ringall`, `leastrecent`, `fewestcalls`, `random`, `rrmemory`, `linear`, `wrandom`	• `ringall`: rings all available callers (default) • `leastrecent`: rings the interface that least recently received a call • `fewestcalls`: rings the interface that has completed the fewest calls in this queue

Options	Available values	Description
		• `random`: rings a random interface • `rrmemory`: rings members in a round-robin fashion, remembering where we left off last for the next caller • `linear`: rings members in the order specified, always starting at the beginning of the list • `wrandom`: rings a random member, but uses the members' penalties as a weight.
`servicelevel`	Value in seconds	Used in statistics to determine the service level of the queue (calls answered within the service level time frame).
`context`	Dialplan context	Allows a caller to exit the queue by pressing a *single* DTMF digit. If a context is specified and the caller enters a number, that digit will attempt to be matched in the context specified, and dialplan execution will continue there.
`penaltymemberslimit`	Value of 0 or greater	Used to disregard penalty values if the number of members in the queue is lower than the value specified.
`timeout`	Value in seconds	Specifies the number of seconds to ring a member's device. Also see `timeoutpriority`.
`retry`	Value in seconds	Specifies the number of seconds to wait before attempting the next member in the queue if the `timeout` value is exhausted while attempting to ring a member of the queue.
`timeoutpriority`	`app`, `conf`	Used to control the priority of the two possible `timeout` options specified for a queue. The `Queue()` application has a timeout value that can be specified to control the absolute time a caller can be in the queue. The `timeout` value in *queues.conf* controls the amount of time (along with `retry`) to ring a member for. Sometime these values conflict, so you can control which value takes precedence. The default is `app`, as this is the way it works in previous versions.
`weight`	Value of 0 or higher	Defines the weight of a queue. A queue with a higher weight defined will get first priority when members are associated with multiple queues.
`wrapuptime`	Value in seconds	The number of seconds to keep a member unavailable in a queue after completing a call.
`autofill`	`yes`, `no`	Same as defined in the `[general]` section. This value can be defined per queue.
`autopause`	`yes`, `no`, `all`	Enables/disables the automatic pausing of members who fail to answer a call. A value of `all` causes this member to be paused in all queues she is a member of.
`maxlen`	Value of 0 or higher	Specifies the maximum number of callers allowed to be waiting in a queue. A value of zero means an unlimited number of callers are allowed in the queue.

Options	Available values	Description
`setinterfacevar`	yes, no	If set to yes, the following channel variables will be set just prior to connecting the caller with the queue member: • `MEMBERINTERFACE`: the member's interface, such as `Agent/1234` • `MEMBERNAME`: the name of the member • `MEMBERCALLS`: the number of calls the interface has taken • `MEMBERLASTCALL`: the last time the member took a call • `MEMBERPENALTY`: the penalty value of the member • `MEMBERDYNAMIC`: indicates whether the member was dynamically added to the queue or not • `MEMBERREALTIME`: indicates whether the member is included from real time or not
`setqueueentryvar`	yes, no	If set to yes, the following channel variables will be set just prior to the call being bridged: • `QEHOLDTIME`: the amount of time the caller was held in the queue • `QEORIGINALPOS`: the position the caller originally entered the queue at
`setqueuevar`	yes, no	If set to yes, the following channel variables will be set just prior to the call being bridged: • `QUEUENAME`: the name of the queue • `QUEUEMAX`: the maximum number of calls allowed in this queue • `QUEUESTRATEGY`: the strategy method defined for the queue • `QUEUECALLS`: the number of calls currently in the queue • `QUEUEHOLDTIME`: the current average hold time of callers in the queue • `QUEUECOMPLETED`: the number of completed calls in this queue • `QUEUEABANDONED`: the number of abandoned calls • `QUEUESRVLEVEL`: the queue service level • `QUEUESRVLEVELPERF`: the queue's service level performance

Options	Available values	Description
`membermacro`	Name of a macro defined in the dialplan	Defines a macro to be executed just prior to bridging the caller and the queue member.
`announce-frequency`	Value in seconds	Defines how often we should announce the caller's position and/or estimated hold time in the queue. Set this value to zero to disable.
`min-announce-frequency`	Value in seconds	Specifies the minimum amount of time that must pass before we announce the caller's position in the queue again. This is used when the caller's position may change frequently, to prevent the caller hearing multiple updates in a short period of time.
`periodic-announce-frequency`	Value in seconds	Indicates how often we should make periodic announcements to the caller.
`random-periodic-announce`	`yes`, `no`	If set to `yes`, will play the defined periodic announcements in a random order. See `periodic-announce`.
`relative-periodic-announce`	`yes`, `no`	If set to `yes`, the `periodic-announce-frequency` timer will start from when the end of the file being played back is reached, instead of from the beginning. Defaults to `no`.
`announce-holdtime`	`yes`, `no`, `once`	Defines whether the estimated hold time should be played along with the periodic announcements. Can be set to `yes`, `no`, or only `once`.
`announce-position`	`yes`, `no`, `limit`, `more`	Defines whether the caller's position in the queue should be announced to her. If set to `no`, the position will never be announced. If set to `yes`, the caller's position will always be announced. If the value is set to `limit`, the caller will hear her position in the queue only if it is within the limit defined by `announce-position-limit`. If the value is set to `more`, the caller will hear her position if it is beyond the number defined by `announce-position-limit`.
`announce-position-limit`	Number of zero or greater	Used if you've defined `announce-position` as either `limit` or `more`.
`announce-round-seconds`	Value in seconds	If this value is nonzero, we'll announce the number of seconds as well, and round them to the value defined.
`queue-thankyou`	Filename of prompt to play	If not defined, will play the default value ("Thank you for your patience"). If set to an empty value, the prompt will not be played at all.
`queue-youarenext`	Filename of prompt to play	If not defined, will play the default value ("You are now first in line"). If set to an empty value, the prompt will not be played at all.
`queue-thereare`	Filename of prompt to play	If not defined, will play the default value ("There are"). If set to an empty value, the prompt will not be played at all.

Options	Available values	Description
queue-callswaiting	Filename of prompt to play	If not defined, will play the default value ("calls waiting"). If set to an empty value, the prompt will not be played at all.
queue-holdtime	Filename of prompt to play	If not defined, will play the default value ("The current estimated hold time is"). If set to an empty value, the prompt will not be played at all.
queue-minutes	Filename of prompt to play	If not defined, will play the default value ("minutes"). If set to an empty value, the prompt will not be played at all.
queue-seconds	Filename of prompt to play	If not defined, will play the default value ("seconds"). If set to an empty value, the prompt will not be played at all.
queue-reporthold	Filename of prompt to play	If not defined, will play the default value ("Hold time"). If set to an empty value, the prompt will not be played at all.
periodic-announce	A set of periodic announcements to be played, separated by commas	Prompts are played in the order they are defined. Defaults to queue-periodic-announce ("All representatives are currently busy assisting other callers. Please wait for the next available representative").
monitor-format	gsm, wav, wav49, *<any valid file format>*	Specifies the file format to use when recording. If monitor-format is commented out, calls will not be recorded.
monitor-type	MixMonitor, *<unspecified>*	Same as monitor-type as defined in the [general] section, but on a per-queue basis.
joinempty	paused, penalty, inuse, ringing, unavailable, invalid, unknown, wrapup	Controls whether a caller is added to the queue when no members are available. Comma-separated options can be included to define how this option determines whether members are available. The definitions for the values are: • paused: members are considered unavailable if they are paused. • penalty: members are considered unavailable if their penalties are less than QUEUE_MAX_PENALTY. • inuse: members are considered unavailable if their device status is In Use. • ringing: members are considered unavailable if their device status is Ringing. • unavailable: applies primarily to agent channels; if the agent is not logged in but is a member of the queue, it is considered unavailable. • invalid: members are considered unavailable if their device status is Invalid. This is typically an error condition. • unknown: members are considered unavailable if device status is unknown.

Options	Available values	Description
		• `wrapup`: members are considered unavailable if they are currently in the wrapup time after the completion of a call.
`leavewhenempty`	`paused`, `penalty`, `inuse`, `ringing`, `unavailable`, `invalid`, `unknown`, `wrapup`	Used to control whether callers are kicked out of the queue when members are no longer available to take calls. See `joinempty` for more information on the assignable values.
`eventwhencalled`	`yes`, `no`, `vars`	If set to `yes`, the following manager events will be sent to the Asterisk Manager Interface (AMI): • `AgentCalled` • `AgentDump` • `AgentConnect` • `AgentComplete` If set to `vars`, all channel variables associated with the agent will also be sent to the AMI.
`eventmemberstatus`	`yes`, `no`	If set to `yes`, the `QueueMemberStatus` event will be sent to AMI. Note that this may generate a lot of manager events.
`reportholdtime`	`yes`, `no`	Enables reporting of the caller's hold time to the queue member prior to bridging.
`ringinuse`	`yes`, `no`	Used to avoid sending calls to members whose status is `In Use`. Recall from our discussion in the preceding section that only the SIP channel driver is currently able to accurately report this status.
`memberdelay`	Value in seconds	Used if you want there to be a delay prior to the caller and queue member being connected to each other.
`timeoutrestart`	`yes`, `no`	If set to `yes`, resets the timeout for an agent to answer if either a `BUSY` or `CONGESTION` status is received from the channel. This can be useful if the agent is allowed to reject or cancel a call.
`defaultrule`	Rule as defined in *queuerules.conf*	Associates a queue rule as defined in *queuerules.conf* to this queue, which is used to dynamically change the minimum and maximum penalties, which are then used to select an available agent. See "Changing Penalties Dynamically (queuerules.conf)" on page 285.
`member`	Device	Used to define static members in a queue. To define a static member, you supply its *`Technology/Device_ID`* (e.g., `Agent/1234`, `SIP/0000FFFF0001`, `DAHDI/g0/14165551212`).

13.4 agents.conf配置文件

如果你浏览了 *~/src/asterisk-complete/1.8/configs/* 目录下的例子，你可能已经注意到 agents.conf 文件。它看起来似乎是吸引人的，而且它有它的用处，但是总体而言，实现队列的最佳办法是使用 sip channels。这有两个原因。第一个原因是 SIP channels 是唯一能提供真实设备状态信息的 channel 类型。另一个原因是如果使用 agent channel 的话 agents 会一直处于登录状态，因此如果你使用远程 agents 的话，对带宽的要求可能比你想象的要大。然而，在繁忙的呼叫中心中，强制坐席立即接听电话而不是等待他们按下电话上的应答按钮也许是值得的。

配置文件 *agents.conf* 用于定义使用 agents channel 的坐席成员。这个 channel 在本质上与 Asterisk 中的其它 channel（local, SIP, IAX2, 等）类似，但它更像一个将来电连接到利用其它 channel 登录到系统中的坐席的伪 channel（pseudo-channel）。例如，假设我们使用我们的 SIP 话机通过 AgentLogin() dialplan 应用程序登录到 Asterisk。一旦登入，这个 channel 将在整个登录期间保持在线状态，然后呼叫通过 agent channel 转接给它。

下面让我们来看一下 *agents.conf* 中的不同选项，以更好的理解它的作用。Table 13-3 展示了 agents.conf 中[general]部分的唯一一个选项。Table13-4 展示了[agents]部分的有效选项。

Table 13-3. Options available under the [general] header in agents.conf

Options	Available values	Description
`multiplelogin`	yes, no	If set to yes, a single line on a device can log in as multiple agents. Defaults to yes.

Table 13-4. Options available under the [agents] header in agents.conf

Options	Available values	Description
`maxloginretries`	Integer value	Specifies the maximum number of tries an agent has to log in before the system considers it a failed attempt and ends the call. Defaults to 3.
`autologoff`	Value in seconds	Specifies the number of seconds for which an agent's device should ring before the agent is automatically logged off.
`autologoffunavail`	yes, no	If set to yes, the agent is automatically logged off when the device being called returns a status of CHANUNAVAIL.
`ackcall`	yes, no	If set to yes, the agent must enter a single DTMF digit to accept the call. To be used in conjunction with `acceptdtmf`. Defaults to no.
`acceptdtmf`	Single DTMF character	Used in conjunction with `ackcall`, this option defines the DTMF character to be used to accept a call. Defaults to #.
`endcall`	yes, no	If set to yes, allows an agent to end a call with a single DTMF digit. To be used in conjunction with `enddtmf`. Defaults to yes.

Options	Available values	Description
`enddtmf`	Single DTMF character	Used in conjunction with `endcall`, this option defines the DTMF character to be used to end a call. Defaults to `*`.
`wrapuptime`	Value in milliseconds	Specifies the amount of time after disconnection of a caller from an agent for which the agent will not be available to accept another call. Used in situations where agents must perform a function after each call (such as entering call details into a log).
`musiconhold`	Music class as defined in *musiconhold.conf*	Defines the default music class agents listen to when logged in.
`goodbye`	Name of file (relative to */var/lib/asterisk/sounds/<lang>/*)	Defines the default goodbye sound played to agents. Defaults to *vm-goodbye*.
`updatecdr`	`yes`, `no`	Used in call detail records to change the source channel field to the `agent/`*`agent_id`*.
`group`	Integer value	Allows you to define groups for sets of agents. *The use of agent groups is essentially deprecated functionality that we do not recommend you use.* If you define `group1`, you can use `Agent/@1` in *queues.conf* to call that group of agents. The call will be connected arbitrarily to one of those agents. If no agents are available, it will return back to the queue like any other unanswered call. If you use `Agent/:1`, it will wait for a member of the group to become available. *The use of strategies has no effect on agent groups. Do not use these.*
`recordagentcalls`	`yes`, `no`	Enables/disables the recording of agent calls. Disabled by default.
`recordformat`	File format (`gsm`, `wav`, etc.)	Defines the format to be used when recording agent calls. Default is `wav`.
`urlprefix`	String (URL)	Accepts a string as its argument. The string can be formed as a URL and is appended to the start of the text to be added to the name of the recording.
`savecallsin`	Filesystem path (e.g., */var/calls/*)	Accepts a filesystem path as its argument. Allows you to override the default path of */var/spool/asterisk/monitor/* with one of your choosing.[a]
`custom_beep`	Name of file (relative to */var/lib/asterisk/sounds/<lang>/*)	Accepts a filename as its argument. Can be used to define a custom notification tone to signal to an always-connected agent that there is an incoming call.
`agent`	Agent definition (see description)	Defines an agent for use by `Queue()` and `AgentLogin()`. These are agents that will log in and stay connected to the system, waiting for calls to be delivered by the `Queue()` dialplan application. Agents are defined like so: `agent => `*`agent_id,agent_password,name`* An example of a defined agent would be: `agent => 1000,1234,Danielle Roberts`

[a] Since the storage of calls will require a large amount of hard drive space, you will want to define a strategy to handle storing and managing these recordings. This location should probably reside on a separate volume, one with very high performance characteristics.

13.5 高级队列

在本节我们将讨论一些精细的队列控制，例如控制通知信息播放的选项，以及控制呼应应当何时进入（或离开）队列的选项。我们也将讨论惩罚值和优先级，探索如何控制队列中的坐席以实现让一组坐席优先应答电话以及根据来电在队列中的等待时间动态调整组成员的多少。最后，我们将讨论 Local channels 作为队列成员的情况，这将是我们可以在将呼叫

转接给坐席之前执行 dialplan 功能。

13.5.1优先队列（队列权重）

有时候你需要将某些呼叫放入比其它呼叫更高优先级的队列中。比如可能这些呼叫已经在这个队列中等待了一段时间，或者某个坐席在接听后意识到需要将呼叫转接到其它队列中。在这种情况下，为了最小化呼叫者的总体等待时间，最好将这个呼叫转接到一个更高优先级权重的队列中去，这样它将更快被接听。

给队列设置更高的优先级通过 **weight** 选项来实现。如果有两个不同权重的队列（例如 **support** 和 **support-priority**），同时加入两个队列的坐席将更优先从高优先级队列中获取呼叫。这些坐席在处理完高优先级队列中的呼叫前不会从低优先级队列中获取呼叫。（通常地，应该有一些坐席只被分配给低优先级队列，以保证低优先级队列中的呼叫可以被及时处理）。举例来说，如果我们将队列成员 James Shaw 分配给 **support** 和 **support-priority** 队列，**support-priority** 队列中的呼叫将比 **support** 队列中的呼叫优先得到 James 的处理。

让我们看一下如何实现这件工作。首先，我们需要创建两个队列，它们除了 **weight** 选项之外完全相同。我们使用模板来保证着两个队列保持一致，即便将来需要修改也是这样：

```
[support_template](!)
musicclass=default
strategy=rrmemory
joinempty=no
leavewhenempty=yes
ringinuse=no
[support](support_template)
weight=0
[support-priority](support_template)
weight=10
```

创建了队列之后（以及随后通过 Asterisk 控制台命令 *moudule reload app_queue.so* 进行了重新加载），我们现在可以创建两个 extensions 来转接呼叫。这段代码可以放在任何一个你需要放置 dialplan 逻辑实现转接的地方。我们将使用 LocalSets，这是我们之前在我们设备中使能的起始 context：

```
[LocalSets]
include => Queue ; allow direct transfer of calls to queues

[Queues]
exten => 7000,1,Verbose(2,Entering the support queue)
    same => n,Queue(support)              ; standard support queue available
                                          ; at extension 7000
    same => n,VoiceMail(7000@queues,u)    ; if there are no members in the queue,
                                          ; we exit and send the caller to voicemail
    same => n,Hangup()
exten => 8000,1,Verbose(2,Entering the priority support queue)
    same => n,Queue(support-priority)     ; priority queue available at
```

```
                                            ; extension 8000
   same => n,VoiceMail(7000@queues,u)      ; if there are no members in the queue,
                                            ; we exit and send the caller to voicemail
   same => n,Hangup()
```

现在你已经有了两个不同权重的队列。我们将标准队列配置在 extension **7000**，而高优先级队列配置在 **8000**。我们可以通过简单的在 **7XXX** 和 **8XXX** 范围对称的配置来镜像多个队列。举例来说，如果我们配置 **sales** 队列在 **7004**，则 **priority-sales** 队列可以被放在镜像队列 **8004**，它具有更高的权重。

剩下唯一要做的配置是确保你的部分或全部坐席在两个队列中都添加了。如果你在 7XXX 队列有更多的来电，你可能希望更多的坐席登录到这个队列，同时一定比例的坐席登录到两个队列。如何正确地配置你的队列取决于具体情况。

13.5.2队列成员优先级

在一个队列中，我们可以通过*惩罚（penalize）*某个成员以降低他接听电话的优先级。举例来说，当我们希望某些坐席是特定队列的成员，但是又只有当这个队列满了而所有高优先级的坐席又都在忙时才被使用，我们就可以通过*惩罚（penalize）*队列成员来实现。这意味着我们可以有三个队列（例如，**support**，**sales**，和 **billing**），每个队列都包含同样的三个坐席：James Shaw, Kay Madsen, 和 Danielle Roberts。

假设，无论如何，我们希望 James Shaw 是 **support** 队列的优先联系人，Kay Madsen 是 **sales** 队列的优先联系人，而 Danielle Roberts 是 **billing** 队列的优先联系人。通过在 **support** 队列惩罚 Kay Madsen 和 Danielle Roberts，我们可以保证 James Shaw 优先获得这个队列的呼叫。同样地，我们可以在 **sales** 队列惩罚 James Shaw 和 Danielle Roberts 来保证 Kay Madsen 是优先联系人，以及在 **billing** 队列惩罚 James Shaw 和 Kay Madsen 来保证 Danielle Roberts 是优先联系人。

惩罚队列成员可以通过编辑 queues.conf 文件来实现，如果你静态指定队列成员的话；或者通过 **AddQueueMember()** dialplan 应用程序来实现。让我们来看一下如何通过 *queues.conf* 在队列中设置静态成员。我们将使用之前在本章定义的 **StandardQueue** 模板：

```
[support](StandardQueue)
member => SIP/0000FFFF0001,0,James Shaw          ; preferred
member => SIP/0000FFFF0002,10,Kay Madsen         ; second preferred
member => SIP/0000FFFF0003,20,Danielle Roberts   ; least preferred

[sales](StandardQueue)
member => SIP/0000FFFF0002,0,Kay Madsen
member => SIP/0000FFFF0003,10,Danielle Roberts
member => SIP/0000FFFF0001,20,James Shaw
[billing](StandardQueue)
member => SIP/0000FFFF0003,0,Danielle Roberts
member => SIP/0000FFFF0001,10,James Shaw
member => SIP/0000FFFF0002,20,Kay Madsen
```

通过给每个队列成员定义不同的惩罚，可以帮助我们控制呼叫优先分配给谁，同时仍然保证当优先做些不可用时其它队列成员也可以接听。惩罚也可以使用 **AddQueueMember()** 来定义，如下面的例子所示：

```
exten => *54,1,Verbose(2,Logging In Queue Member)
    same => n,Set(MemberChannel=${CHANNEL(channeltype)}/${CHANNEL(peername)})

; *CLI> database put queue support/0000FFFF0001/penalty 0
    same => n,Set(QueuePenalty=${DB(queue/support/${CHANNEL(peername)}/penalty)})

; *CLI> database put queue support/0000FFFF0001/membername "James Shaw"
    same => n,Set(MemberName=${DB(queue/support/${CHANNEL(peername)}/membername)})

; AddQueueMember(queuename[,interface[,penalty[,options[,membername[,stateinterface]]]]])
    same => n,AddQueueMember(support,${MemberChannel},${QueuePenalty},,${MemberName})
```

通过 **AddQueueMember()**，我们展示了在一个队列中如何通过成员名获得惩罚值，以及如何在她登入队列的时候指定这个惩罚值。为了让这种办法在多队列的情况工作，还需要讨论额外的概念；进一步信息请参阅“13.2.3 自动登入和登出多个队列”一节。

13.5.3动态调整惩罚值（queuerules.conf）

通过使用 queuerules.conf 文件，可以指定一些规则来改变 **QUEUE_MIN_PENALTY** 和 **QUEUE_MAX_PENALTY** 的值。**QUEUE_MIN_PENALTY** 和 **QUEUE_MAX_PENALTY** channel 变量用于控制队列中的那些成员可以用于接听电话。假设我们有一个队列命名为 support，而且我们有 5 个队列成员，他们的惩罚值为从 **1** 到 **5**。如果在呼叫到达队列前，**QUEUE_MIN_PENALTY** 的值设置为 **2**，而 **QUEUE_MAX_PENALTY** 的值设置为 **4**，只有惩罚值被设置在 **2** 到 **4** 之间的队列成员可以应答电话：

```
[Queues]
exten => 7000,1,Verbose(2,Entering the support queue)
    same => n,Set(QUEUE_MIN_PENALTY=2)    ; set minimum queue member penalty to be used
    same => n,Set(QUEUE_MAX_PENALTY=4)    ; set maximum queue member penalty we'll use
    same => n,Queue(support)              ; entering the queue with minimum and maximum
                                          ; member penalties to be used
```

更进一步，在来电停留在队列中期间，我们可以动态改变 **QUEUE_MIN_PENALTY** 和 **QUEUE_MAX_PENALTY** 的值。这可以允许更多或不同的队列成员被使用，具体取决于来电在队列中等待了多久。例如，在前面的例子中，我们可以调整最小惩罚值为 1 同时最大惩罚值 5，如果来电在队列中的等待超过 60 秒的话。

这些规则在 *queuerules.conf* 中定义。为了实现通话过程中的多种不同惩罚值变化，可以创建多条规则。让我们看看如何定义上一段描述的变化：

```
[more_members]
penaltychange => 60,5,1
```

如果你修改了 *queuerules.conf* 文件，并重载了 app_queue.so，新的规则将对到达队列的新呼叫生效，不会影响已经存在的呼叫。

我们已经在 *queuerules.conf* 中定义的了规则 **more_members** 并且将后面的值赋给了 **penaltychange: 60** 是改变惩罚值前需要等待的秒数，**5** 是新的 **QUEUE_MAX_PENALTY** 值，而 **1** 是新的 **QUEUE_MIN_PENALTY** 值。对于我们新定义的规则，我们必须重新加载 *app_queue.so* 使它生效：

```
*CLI> module reload app_queue.so
    -- Reloading module 'app_queue.so' (True Call Queueing)
== Parsing '/etc/asterisk/queuerules.conf': == Found
```

我们可以在控制台通过 *queue show rules* 进行检查：

```
*CLI> queue show rules
Rule: more_members
    After 60 seconds, adjust QUEUE_MAX_PENALTY to 5 and adjust QUEUE_MIN_PENALTY to 1
```

随着我们的新规则被载入内存，我们可以修改我们的 dialplan 来使用它。只需要修改 Queue()这一行来包含新规则，就像这样：

```
[Queues]
exten => 7000,1,Verbose(2,Entering the support queue)
    same => n,Set(QUEUE_MIN_PENALTY=2)          ; set minimum queue member penalty
    same => n,Set(QUEUE_MAX_PENALTY=4)          ; set maximum queue member penalty

; Queue(queuename[,options[,URL[,announceoverride[,timeout[,AGI[,macro [,gosub[,rule[,position]]]]]]]]])
    same => n,Queue(support,,,,,,,,more_members)    ; entering queue with minimum and
                                                ; maximum member penalties
```

queuerules.conf 文件非常灵活。我们可以使用相对值而不是绝对值来定义我们的规则，从而可以定义多条规则：

```
[more_members]
penaltychange => 30,+1
penaltychange => 45,,-1
penaltychange => 60,+1
penaltychange => 120,+2
```

这里，我们修改了 **more_members** 规则为使用相对值。30 秒后，我们将最大惩罚值加 **1**（如果使用我们的 dialplan 例子，惩罚值将变为 **5**）。45 秒后，我们将最小惩罚值减 **1**，等等。我们可以在 Asterisk 控制台重新加载 *module reload app_queue.so* 之后检查我们的新规则变化：

```
*CLI> queue show rules
Rule: more_members
    After 30 seconds, adjust QUEUE_MAX_PENALTY by 1 and adjust QUEUE_MIN_PENALTY by 0
    After 45 seconds, adjust QUEUE_MAX_PENALTY by 0 and adjust QUEUE_MIN_PENALTY by -1
    After 60 seconds, adjust QUEUE_MAX_PENALTY by 1 and adjust QUEUE_MIN_PENALTY by 0
    After 120 seconds, adjust QUEUE_MAX_PENALTY by 2 and adjust QUEUE_MIN_PENALTY by 0
```

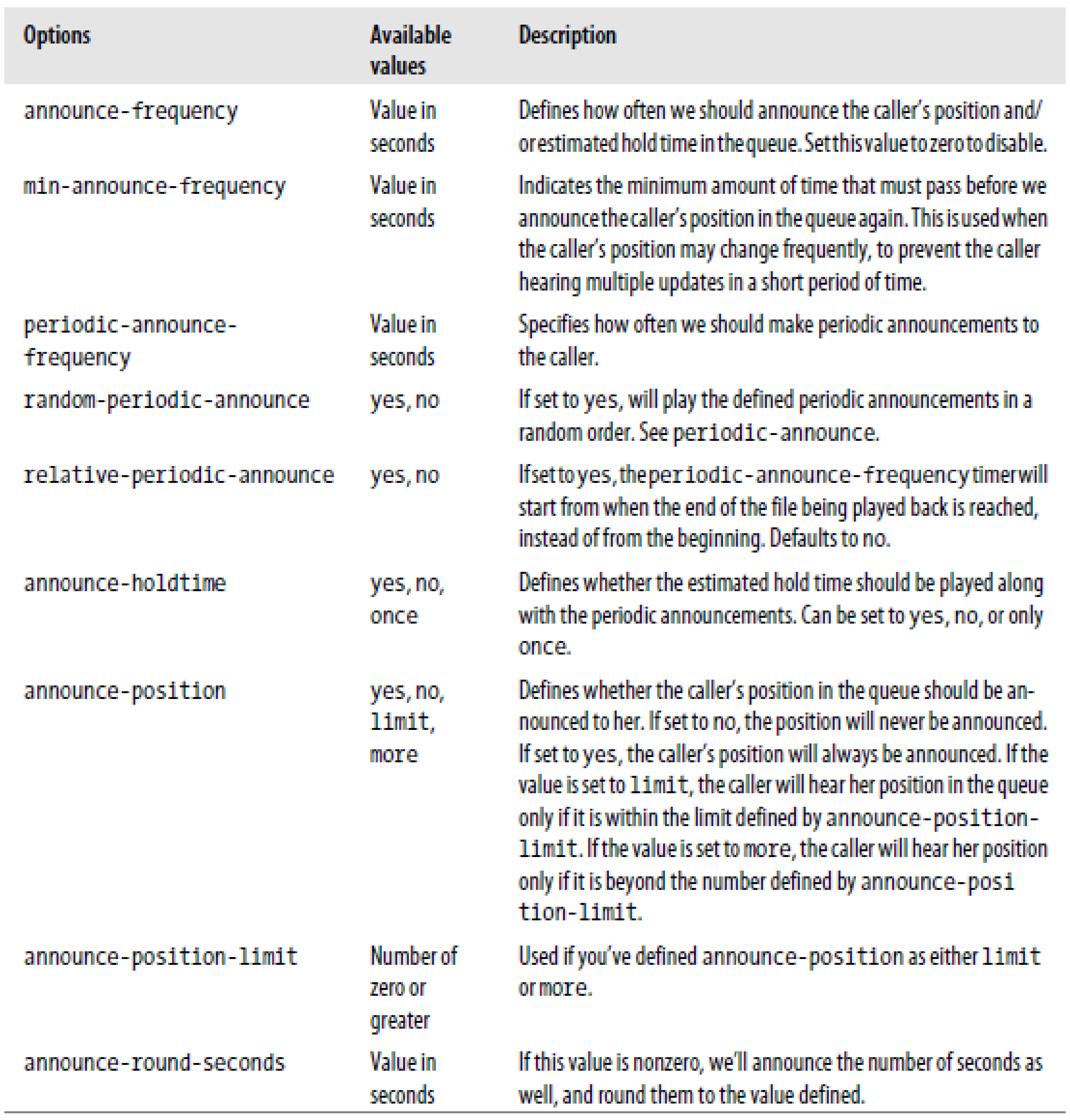
13.5.4通知控制

Asterisk 有能力向队列中等待的客户播放几条通知。例如，你可能想通知呼叫者在队列中的位置，平均等待时间，或者周期性的播放感谢客户等待的信息（或者任意你指定的声音文件）。调整控制何时向来电用户播放这些通知的参数十分重要，因为过于频繁的通知他们的位置，感谢他们等待，或者告诉他们等待时间的话可能会惹恼他们，导致他们或者挂断电话，或者向你的坐席人员发火。

在 *queues.conf* 文件中有几个选项用于微调什么内容及何时向你的客户播放通知。完整的 queue 选项我们在本章前几节已经介绍过了，但是我们仍将在这里复习一下相关的选项。

Table13-5 列出了用于控制何时向客户播放通知的选项。

Table 13-5. Options related to prompt control timing within a queue



Options	Available values	Description
`announce-frequency`	Value in seconds	Defines how often we should announce the caller's position and/or estimated hold time in the queue. Set this value to zero to disable.
`min-announce-frequency`	Value in seconds	Indicates the minimum amount of time that must pass before we announce the caller's position in the queue again. This is used when the caller's position may change frequently, to prevent the caller hearing multiple updates in a short period of time.
`periodic-announce-frequency`	Value in seconds	Specifies how often we should make periodic announcements to the caller.
`random-periodic-announce`	`yes`, `no`	If set to `yes`, will play the defined periodic announcements in a random order. See `periodic-announce`.
`relative-periodic-announce`	`yes`, `no`	If set to `yes`, the `periodic-announce-frequency` timer will start from when the end of the file being played back is reached, instead of from the beginning. Defaults to `no`.
`announce-holdtime`	`yes`, `no`, `once`	Defines whether the estimated hold time should be played along with the periodic announcements. Can be set to `yes`, `no`, or only `once`.
`announce-position`	`yes`, `no`, `limit`, `more`	Defines whether the caller's position in the queue should be announced to her. If set to `no`, the position will never be announced. If set to `yes`, the caller's position will always be announced. If the value is set to `limit`, the caller will hear her position in the queue only if it is within the limit defined by `announce-position-limit`. If the value is set to `more`, the caller will hear her position only if it is beyond the number defined by `announce-position-limit`.
`announce-position-limit`	Number of zero or greater	Used if you've defined `announce-position` as either `limit` or `more`.
`announce-round-seconds`	Value in seconds	If this value is nonzero, we'll announce the number of seconds as well, and round them to the value defined.

Table 13-6 显示了当向客户播放通知时会用到哪些文件。

Table 13-6. Options for controlling the playback of prompts within a queue

Options	Available values	Description
`musicclass`	Music class as defined by *musiconhold.conf*	Sets the music class to be used by a particular queue. You can also override this value with the CHANNEL(`musicclass`) channel variable.
`queue-thankyou`	Filename of prompt to play	If not defined, will play the default value ("Thank you for your patience"). If set to an empty value, the prompt will not be played at all.

Options	Available values	Description
`queue-youarenext`	Filename of prompt to play	If not defined, will play the default value ("You are now first in line"). If set to an empty value, the prompt will not be played at all.
`queue-thereare`	Filename of prompt to play	If not defined, will play the default value ("There are"). If set to an empty value, the prompt will not be played at all.
`queue-callswaiting`	Filename of prompt to play	If not defined, will play the default value ("calls waiting"). If set to an empty value, the prompt will not be played at all.
`queue-holdtime`	Filename of prompt to play	If not defined, will play the default value ("The current estimated hold time is"). If set to an empty value, the prompt will not be played at all.
`queue-minutes`	Filename of prompt to play	If not defined, will play the default value ("minutes"). If set to an empty value, the prompt will not be played at all.
`queue-seconds`	Filename of prompt to play	If not defined, will play the default value ("seconds"). If set to an empty value, the prompt will not be played at all.
`queue-reporthold`	Filename of prompt to play	If not defined, will play the default value ("Hold time"). If set to an empty value, the prompt will not be played at all.
`periodic-announce`	A set of periodic announcements to be played, separated by commas	Prompts are played in the order they are defined. Defaults to `queue-periodic-announce` ("All representatives are currently busy assisting other callers. Please wait for the next available representative").

如果这些专用于播放通知的选项是其价值的指示，大概我们最感兴趣的就是如何利用它们发挥最大的潜力。在 Table13-5 中的选项帮助我们定义何时播放通知，而 Table13-6 中的选项帮助我们控制我们向客户播放什么。有这两个表在手，让我们看一个队列的例子，在这里我们将定义一些值。我们将从我们的基本队列模板开始：

```
[general]
autofill=yes                ; distribute all waiting callers to available members
shared_lastcall=yes         ; respect the wrapup time for members logged into more
                            ; than one queue
[StandardQueue](!)          ; template to provide common features
musicclass=default          ; play [default] music
strategy=rrmemory           ; use the Round Robin Memory strategy
joinempty=yes               ; do not join the queue when no members available
leavewhenempty=no           ; leave the queue when no members available
ringinuse=no                ; don't ring members when already InUse (prevents
                            ; multiple calls to an agent)
[sales](StandardQueue)        ; create the sales queue using the parameters in the
                            ; StandardQueue template
[support](StandardQueue)      ; create the support queue using the parameters in the
                            ; StandardQueue template
```

现在修改 **StandardQueue** 模板来控制我们的通知：

```
[StandardQueue](!)            ; template to provide common features
musicclass=default            ; play [default] music
strategy=rrmemory             ; use the Round Robin Memory strategy
joinempty=yes                 ; do not join the queue when no members available
leavewhenempty=no             ; leave the queue when no members available
ringinuse=no                  ; don't ring members when already InUse (prevents multiple calls to an agent)
; -------- Announcement Control --------
announce-frequency=30              ; announces caller's hold time and position every 30
                                   ; seconds
min-announce-frequency=30          ; minimum amount of time that must pass before the
                                   ; caller's position is announced
periodic-announce-frequency=45     ; defines how often to play a periodic announcement to caller
random-periodic-announce=no        ; defines whether to play periodic announcements in
                                   ; a random order, or serially
relative-periodic-announce=yes     ; defines whether the timer starts at the end of
                                   ; file playback (yes) or the beginning (no)
announce-holdtime=once             ; defines whether the estimated hold time should be
                                   ; played along with the periodic announcement
announce-position=limit            ; defines if we should announce the caller's position
                                   ; in the queue
announce-position-limit=10         ; defines the limit value where we announce the
                                   ; caller's position (when announce-position is set to limit or more)
announce-round-seconds=30          ; rounds the hold time announcement to the nearest 30-second value
```

让我们描述一下刚才我们在 StandardQueue 模板中设置了什么。

我们将每隔 30 秒（**announce-frequency**）通知一次呼叫者等待时间和他在队列中的位置[注 4]，并且保证我们再次通知前的最短时间是 30 秒（**min-announce-frequency**）。我们通过这两个参数来限制我们的通知向客户播放的频率，以免惹人厌烦。周期性地，我们播放一条通知给客户感谢他们的耐心等待并向他们保证客服人员将很快接听电话。（这个通知信息通过 **periodic-announce** 选项定义。我们使用默认的通知，但是你也可以使用 **periodic-announce** 定义一个你自己的一个或多个通知。）

这个周期性的通知每隔 45 秒（**periodic-announce-frequency**）播放一次，按照预先定义的顺序播放（**random-period-announce**）。为了确定 **periodic-announce-frequency** 定时器何时启动，我们使用 **relative-periodic-announce**。这个选项设置为 **yes**，意味着定时器会在通知播放停止后启动，而不是在开始播放时启动。把这个选项设置为 **no** 的话你将遇到的问题是，如果你的通知很长（例如 30 秒），这将导致通知每隔 15 秒播放一次，而不是你打算的每隔 45 秒播放一次。

向呼叫者通知几次等待时间通过 **announce-holdtime** 来控制，我们的例子配置为 **once**。将这个值配置为 **yes** 将每次都通知等待时间，而设置为 **no** 将禁止通知等待时间。

如何及何时通知呼叫者的估计剩余等待时间通过 **announce-position** 控制，我们的例子中设置为 **limit**。设置 **announce-position** 为 **limit** 将使我们仅仅当呼叫者的位置在 **announce-position-limit** 限制之内时才通知客户。因此，在我们的例子中，仅当客户处于队列前 10 名的位置时我们才向他播放通知。我们也可以将这个选项配置为 **yes**，这样每次周期性播放时都会通知客户在队列中的位置，配置为 **no** 则不通知排队位置信息。或者可以将其配置为 **more**，如果我们希望仅当客户的排队位置大于 **announce-position-limit** 时才通知他排队位置信息。

我们要介绍的最后一个选项，**announce-round-seconds**，其作用是控制当通知客户等待时间时对时间值的四舍五入。在我们的例子中，不会播放“1 分钟 23 秒”，时间值会被舍入到最近似的 30 秒，结果是通知提示音为“1 分钟 30 秒”。

13.5.5溢出（Overflow）

队列溢出将发生在或者等待超时时，或者当没有有效的队列成员时（当定义了 joinempty 或 leavewhenempty 时）。在本节中，我们将讨论如何控制何时发生溢出。

13.5.5.1 控制超时

Queue()应用程序支持两种超时：一种是呼叫者在队列中的最大等待时间，另一种是当 Asterisk 试图将呼叫者转接给某个坐席时的最大振铃时间。我们将讨论呼叫者在这个呼叫溢出到 dialplan 的其它位置（例如 **VoiceMail()**）之前在队列中的最大等待时间。一旦这个呼叫放弃了这个队列，它可以转接到 dialplan 中任何呼叫可以转接到的位置。

超时被定义在两个地方。队列成员的最大振铃时间定义在 queues.conf 文件中。绝对超时时间（呼叫者在队列中的最大停留时间）通过 **Queue()**应用程序控制。要设置呼叫者在队列中的最大停留时间，只要简单的在 **Queue()**应用程序中的 queue name 参数之后指定就可以：

```
[Queues]
exten => 7000,1,Verbose(2,Joining the support queue for a maximum of 2 minutes)
    same => n,Queue(support,120)
    same => n,VoiceMail(support@queues,u)
    same => n,Hangup()
```

当然，我们可以定义不同的溢出转接目的地，但是 VoiceMail()应用程序已经足够好了。只是需要注意你应当将呼叫者转接到一个经常有人检查并回拨给客户的语音信箱。

现在假设有这么一个场景：我们设置绝对超时时间为 10 秒，队列成员的振铃超时时间为 5 秒，重拨坐席的超时时间为 4 秒。在这个场景下，我们将振铃队列成员 5 秒，然后在尝试拨打另一个队列成员前等待 4 秒。这些将用去 9 秒，而绝对超时时间是 10 秒。在这时候，

我们应该振铃下一个队列成员 1 秒然后退出队列，还是应该在退出队列前完整的振铃下一个队列成员 5 秒？

我们通过 *queues.conf* 文件中的 **timeoutpriority** 选项控制这种情况。这个选项的有效取值为 **app** 和 **conf**。如果我们希望应用程序超时（绝对超时）优先，这将导致呼叫者精确地在 10 秒后退出队列，我们应该设置 **timeoutpriority** 的值为 **app**。如果我们希望配置文件超时优先从而完成振铃一个队列成员——这将导致呼叫者在队列中的停留时间稍长，我们将设置 **timeoutpriority** 的值为 **conf**。这个选项的默认值是 **app**（这是旧版本 Asterisk 的默认行为）。

13.5.5.2 控制何时加入和离开队列

Asterisk 提供了两个选项控制呼叫者根据队列成员的不同状态何时可以加入及何时被强制移出队列。第一个选项，**joinempty**，用于控制是否允许呼叫者加入队列。而 leavewhenempty 选项用于控制何时将已经在队列中的呼叫者强制删除（例如，当所有的队列成员都无效时）。这两个选项都有一系列用逗号隔开的取值来控制它们的行为。参见 Table13-7.

Table 13-7. Options that can be set for joinempty or leavewhenempty

Value	Description
`paused`	Members are considered unavailable if they are paused.
`penalty`	Members are considered unavailable if their penalties are less than `QUEUE_MAX_PENALTY`.
`inuse`	Members are considered unavailable if their device status is `In Use`.
`ringing`	Members are considered unavailable if their device status is `Ringing`.
`unavailable`	Applies primarily to agent channels; if the agent is not logged in but is a member of the queue it is considered unavailable.
`invalid`	Members are considered unavailable if their device status is `Invalid`. This is typically an error condition.
`unknown`	Members are considered unavailable if device status is unknown.
`wrapup`	Members are considered unavailable if they are currently in the wrapup time after the completion of a call.

对于 **joinempty** 选项来说，在 将呼叫者放入队列前，所有的队列成员都将根据你配置的参数作为标准检查其有效性。如果所有的队列成员都被认为是无效的，呼叫者将不被允许加入队列，dialplan 将执行下一条注5。对于 **leavewhempty** 选项来说，将根据你配置的条件周期性的检查队列成员状态；如果判定已经没有有效成员可以接听电话，呼叫者会被移出这个队列，dialplan 将继续执行下一条。

举一个使用 **joinempty** 的例子：

```
joinempty=paused,inuse,invalid
```

在这个配置下，在呼叫者进入队列之前将会检查所有队列成员的状态，并且呼叫者不会被允许加入队列，除非至少有一个成员的状态不是 **paused**，**inuse**，或 **invalid**。

再举一个 leaveehempty 的例子：

```
leavewhenempty=inuse,ringing
```

在这个例子中，队列成员的状态被周期性的检查，而呼叫者将被移出队列，如果不能找到某

个队列成员的状态既不是 **inuse** 也不是 **ringing**。

早期的 Asterisk 版本使用 yes, no , strict, 以及 loose 作为这两个选项的有效取值。新旧取值之间的对照关系参见 Table13-8。

Table 13-8. Mapping between old and new values for controlling when callers join and leave queues

Value	Mapping (joinempty)	Mapping (leavewhenempty)
`yes`	(empty)	penalty,paused,invalid
`no`	penalty,paused,invalid	(empty)
`strict`	penalty,paused,invalid,unavailable	penalty,paused,invalid,unavailable
`loose`	penalty,invalid	penalty,invalid

13.5.6使用Local Channels

使用 Local channels 作为队列成员是一种在实际拨打坐席之前执行部分 dialplan 逻辑或执行检查的流行方式。举例来说，它将允许我们实现诸如启动录音，设置 channel 变量，写入日志文件，设置呼叫时长（比如这是个付费服务），或者做任何我们一旦知道转接目的地之后希望做的事。

当在队列中使用 Local channels 时，它们可以像任何其它 channels 一样加入队列。在 *queues.conf* 文件中，增加 Local channel 的做法如下：

```
; queues.conf
[support](StandardQueue)
member => Local/SIP-0000FFFF0001@MemberConnector     ; pass the technology to dial over
                                                     ; and the device identifier,
                                                     ; separated by a hyphen. We'll
                                                     ; break it apart inside the
                                                     ; MemberConnector context.
```

请注意我们是如何将你要拨打设备的技术类型和设备标示符传递给 **MemberConnector** context 的。我们简单的使用连字符（尽管我们可以使用差不多任意字符作为分隔符）作为字段标记。我们将在 **MemberConnector** context 中使用 **CUT()**函数并分配第一个字段（**SIP**）给一个 channel 变量，分配第二个字段（**0000FFFF0001**）给另一个 channel 变量，然后我们将使用这两个 channel 变量呼叫终端。

传递信息，然后在 context 中被 Local channel“炸开”的做法是一种常用且有用的技术（类似于 PHP 中的 explode()函数）。

当然，我们需要 MemberConnector context 实际连接呼叫者到坐席：

```
[MemberConnector]
exten => _[A-Za-z0-9].,1,Verbose(2,Connecting ${CALLERID(all)} to Agent at ${EXTEN})
        ; filter out any bad characters, allowing alphanumeric characters and the hyphen
        same => n,Set(QueueMember=${FILTER(A-Za-z0-9\-,${EXTEN})

        ; assign the first field of QueueMember to Technology using the hyphen separator
        same => n,Set(Technology=${CUT(QueueMember,-,1)})
        ; assign the second field of QueueMember to Device using the hyphen separator
        same => n,Set(Device=${CUT(QueueMember,-,2)})

        ; dial the agent
        same => n,Dial(${Technology}/${Device})
        same => n,Hangup()
```

这样，你就将队列成员传递给了 context，然后我们就可以拨打这个设备。然而，因为我们使用 Local channel 作为 queue member，**Queue()**不需要知道呼叫的状态，特别是当 Local channel 优化过之后（请参考 https://wiki.asterisk.org/wiki/display/AST/Local+Channel+Modifiers 获取关于 /n 修饰符的更多信息，这将导致 Local channel 不做优化）。队列将监视 Local channel 的状态，而不是我们实际要监视的设备的状态。

幸运地，我们可以让 **Queue()**监视实际的设备并将其状态与 Local channel 关联起来，所以 Local channel 的状态总是反应了我们最终拨打的设备的状态。我们的队列成员将在 *queues.conf* 中修改如下：

```
; queues.conf
[support](StandardQueue)
member => Local/SIP-0000FFFF0001@MemberConnector,,,SIP/0000FFFF0001
```

只有 SIP channels 有能力可靠地发回状态信息，所以我们强烈推荐你在使用 Local channels 作为队列成员时只使用 SIP channels。

你也可以使用 **AddQueueMember()**和 **RemoveQueueMember()**应用程序来增加或删除队列成员，就像处理任何其它 channel 一样。**AddQueueMember()**也具备设置状态接口（state interface）的能力，就像我们在 *queues.conf* 中静态定义的那样。举例说明如下：

```
[QueueMemberLogin]
exten => 500,1,Verbose(2,Logging in device ${CHANNEL(peername)} into the support queue)

        ; Save the device's technology to the MemberTech channel variable
        same => n,Set(MemberTech=${CHANNEL(channeltype)})

        ; Save the device's identifier to the MemberIdent channel variable
        same => n,Set(MemberIdent=${CHANNEL(peername)})

        ; Build up the interface name and assign it to the Interface channel variable
        same => n,Set(Interface=${MemberTech}/${MemberIdent})

        ; Add the member to the support queue using a Local channel. We're using the same
        ; format as before, separating the technology and the device indentifier with
        ; a hyphen and passing that information to the MemberConnector context. We then
        ; use the IF() function to determine if the member's technology is SIP and, if so,
        ; to pass back the contents of the Interface channel variable as the value to the
        ; state interface field of the AddQueueMember() application.
        ;
        ; *** This line should not have any line breaks
        same => n,AddQueueMember(support,Local/${MemberTech}-${MemberIdent}
        @MemberConnector,,,${IF($[${MemberTech} = SIP]?${Interface})})
        same => n,Playback(silence/1)

        ; Play back either the agent-loginok or agent-incorrect file, depending on what
        ; the AQMSTATUS variable is set to.
        same => n,Playback(${IF($[${AQMSTATUS} = ADDED]?agent-loginok:agent-incorrect)})
        same => n,Hangup()
```

现在我们可以使用 Local channels 向队列中增加设备了，让我们看一下如何控制若干呼叫向多于一线的非 SIP channels 或设备发起的情况。我们可以用 **GROUP()**和 **GROUP_COUNT()** 函数跟踪发向一个终端的呼叫计数。我们可以修改 **MemberConnector** context 来实现这一点：

```
[MemberConnector]
exten => _[A-Za-z0-9].,1,Verbose(2,Connecting ${CALLERID(all)} to Agent at ${EXTEN})

        ; filter out any bad characters, allowing alphanumeric characters and the hyphen
        same => n,Set(QueueMember=${FILTER(A-Za-z0-9\-,${EXTEN})

        ; assign the first field of QueueMember to Technology using the hyphen separator
        same => n,Set(Technology=${CUT(QueueMember,-,1)})

        ; assign the second field of QueueMember to Device using the hyphen separator
        same => n,Set(Device=${CUT(QueueMember,-,2)})

        ; Increase the value of the group inside the queue_members category by one
        same => n,Set(GROUP(queue_members)=${Technology}-${Device})

        ; Check if the group@category is greater than 1, and, if so, return Congestion()
        ; (too many channels)
        ;
        ; *** This line should not have any line breaks
        same => n,ExecIf($[${GROUP_COUNT(${Technology}-${Device}@queue_members)} > 1]
        ?Congestion())
        ; dial the agent
        same => n,Dial(${Technology}/${Device})
        same => n,Hangup()
```

执行 Congestion()返回会导致呼叫者返回队列（虽然这发生了，但呼叫者不会得到任何错误提示，并继续听到等待音乐直到我们将他实际连接到一个坐席）。虽然这不是理想的情

况，因为队列将不断尝试连接（或者至少将其包含到坐席循环中，这取决于你有多少个队列成员和他们的状态），但它比一个坐席同时得到多个呼叫要好。

我们也可以用这种方法来创建一种预定过程。如果你希望直接呼叫某个坐席（例如，呼叫者需要联系特定的坐席），你可以通过使用 GROUP()和 GROUP_COUNT()函数触发暂停这个坐席直到这个呼叫者被连接的方式来预定这个坐席（译者注：这是指采用上面例子中 ExecIf 类似的语法来实现）。这对于在连接呼叫者到某个坐席之前需要播放一些通知，而你又不希望这个坐席在通知播放的过程中被连接到其它呼叫者的情况特别有用。

13.6 队列统计：queue_log文件

在/var/log/asterisk 目录下的 queue_log 文件包含着关于你系统中定义的队列的信息（队列何时加载的，队列成员何时加入和删除的，等等），以及进入这个队列的呼叫的信息（例如，它们的状态和呼叫者连接的 channel 类型）。队列日志默认是打开的，但是可以通过 *logger.conf* 文件来控制。与 *queue_log* 文件相关的选项有三个：

queue_log

Controls whether the queue log is enabled or not. Valid values are **yes** or **no** (defaults to **yes**).

queue_log_to_file

Controls whether the queue log should be written to a file even when a real time backend is present. Valid values are **yes** or **no** (defaults to **no**).

queue_log_name

Controls the name of the queue log. The default is **queue_log**.

队列日志是用竖线符号分隔的一系列事件。queue_log 文件的字段如下：

- Epoch timestamp of the event
- Unique ID of the call
- Name of the queue
- Name of bridged channel
- Type of event
- Zero or more event parameters

包含在事件参数中的信息取决于事件类型。举例 *queue_log* 文件如下：

```
1292281046|psy1-1292281041.87|7100|NONE|ENTERQUEUE||4165551212|1
1292281046|psy1-1292281041.87|7100|Local/9996@MemberConnector|RINGNOANSWER|0
1292281048|psy1-1292281041.87|7100|Local/9990@MemberConnector|CONNECT|2
|psy1-1292281046.90|0
1292284121|psy1-1292281041.87|7100|Local/9990@MemberConnector|COMPLETECALLER|2|3073|1
1292284222|MANAGER|7100|Local/9990@MemberConnector|REMOVEMEMBER|
1292284222|MANAGER|7200|Local/9990@MemberConnector|REMOVEMEMBER|
1292284491|MANAGER|7100|Local/9990@MemberConnector|ADDMEMBER|
1292284491|MANAGER|7200|Local/9990@MemberConnector|ADDMEMBER|
1292284519|psy1-1292284515.93|7100|NONE|ENTERQUEUE||4165551212|1
1292284519|psy1-1292284515.93|7100|Local/9996@MemberConnector|RINGNOANSWER|0
```

```
1292284521|psy1-1292284515.93|7100|Local/9990@MemberConnector|CONNECT|2
|psy1-1292284519.96|0
1292284552|MANAGER|7100|Local/9990@MemberConnector|REMOVEMEMBER|
1292284552|MANAGER|7200|Local/9990@MemberConnector|REMOVEMEMBER|
1292284562|psy1-1292284515.93|7100|Local/9990@MemberConnector|COMPLETECALLER|2|41|1
```

如同我们在这个例子中见到的，对每个事件并不总是有唯一 ID。在某些情况下，例如 Asterisk Manager Interface（AMI）这样的外部服务程序，会对队列执行操作；在这种情况下，你会在唯一 ID 字段看到类似 MANAGER 这样的字符。

Table13-9 描述了有效事件及它们提供的信息。

Table 13-9. Events in the Asterisk queue log

Event	Information provided
ABANDON	Written when a caller in a queue hangs up before his call is answered by an agent. Three parameters are provided for ABANDON: the position of the caller at hangup, the original position of the caller when entering the queue, and the amount of time the caller waited prior to hanging up.
ADDMEMBER	Written when a member is added to the queue. The bridged channel name will be populated with the name of the channel added to the queue.
AGENTDUMP	Indicates that the agent hung up on the caller while the queue announcement was being played, prior to them being bridged together.
AGENTLOGIN	Recorded when an agent logs in. The bridged channel field will contain something like `Agent/9994` if logging in with `chan_agent`, and the first parameter field will contain the channel logging in (e.g., `SIP/0000FFFF0001`).
AGENTLOGOFF	Logged when an agent logs off, along with a parameter indicating how long the agent was logged in for.
COMPLETEAGENT	Recorded when a call is bridged to an agent and the agent hangs up, along with parameters indicating the amount of time the caller was held in the queue, the length of the call with the agent, and the original position at which the caller entered the queue.
COMPLETECALLER	Same as COMPLETEAGENT, except the caller hung up and not the agent.
CONFIGRELOAD	Indicates that the queue configuration was reloaded (e.g., via *module reload app_queue.so*).
CONNECT	Written when the caller and the agent are bridged together. Three parameters are also written: the amount of time the caller waited in the queue, the unique ID of the queue member's channel to which the caller was bridged, and the amount of time the queue member's phone rang prior to being answered.
ENTERQUEUE	Written when a caller enters the queue. Two parameters are also written: the URL (if specified) and the caller ID of the caller.
EXITEMPTY	Written when the caller is removed from the queue due to a lack of agents available to answer the call (as specified by the `leavewhenempty` parameter). Three parameters are also written: the position of the caller in the queue, the original position at which the caller entered the queue, and the amount of time the caller was held in the queue.
EXITWITHKEY	Written when the caller exits the queue by pressing a single DTMF key on his phone to exit the queue and continue in the dialplan (as enabled by the `context` parameter in *queues.conf*). Four parameters are recorded: the key used to exit the queue, the position of the caller in the queue upon exit, the original position the caller entered the queue at, and the amount of time the caller was waiting in the queue.
EXITWITHTIMEOUT	Written when the caller is removed from the queue due to timeout (as specified by the `timeout` parameter to `Queue()`). Three parameters are also recorded: the position the caller was in when exiting the queue, the original position of the caller when entering the queue, and the amount of time the caller waited in the queue.
PAUSE	Written when a queue member is paused.
PAUSEALL	Written when all members of a queue are paused.
UNPAUSE	Written when a queue member is unpaused.
UNPAUSEALL	Written when all members of a queue are unpaused.

Event	Information provided
PENALTY	Written when a member's penalty is modified. The penalty can be changed through several means, such as the QUEUE_MEMBER_PENALTY() function, through using Asterisk Manager Interface, or the Asterisk CLI commands.
REMOVEMEMBER	Written when a queue member is removed from the queue. The bridge channel field will contain the name of the member removed from the queue.
RINGNOANSWER	Logged when a queue member is rung for a period of time, and the timeout value for ringing the queue member is exceeded. A single parameter will also be written indicating the amount of time the member's extension rang.
TRANSFER	Written when a caller is transferred to another extension. Additional parameters are also written, which include: the extension and context the caller was transferred to, the hold time of the caller in the queue, the amount of time the caller was speaking to a member of the queue, and the original position of the caller when he entered the queue.[a]
SYSCOMPAT	Recorded if an agent attempts to answer a call, but the call cannot be set up due to incompatibilities in the media setup.

[a] Please note that when the caller is transferred using SIP transfers (rather than the built-in transfers triggered by DTMF and configured in *features.conf*), the TRANSFER event may not be reliable.

13.7 总结

本章我们从介绍基本呼叫队列开始，讨论了它们是什么，它们怎么工作，以及何时你会用到它们。在创建了一个简单队列之后，我们探索了如何通过不同的方法控制队列成员（包括使用 Local channels，它提供了在转接给队列成员之前执行一些 dialplan 逻辑的能力）。我们也探索了在 *queues.conf*，*agents.conf*，和 *queuerules.conf* 文件中的所有选项，它们为我们提供了精细控制队列的能力。当然，我们还需要监视我们的队列在做什么，所以最后我们介绍了队列日志，以及当不同情况发生时所记录的各种事件和事件参数。

根据本章提供的知识，你应该能够很好的以你的方式为你们公司实现一组成功的队列。

注释：

注1. 一个常见的误解是认为排队机制可以让你处理更多的呼叫。严格的说，这是不对的，你的客户仍然希望和服务人员通话，他们只是愿意多等一会。换句话说，如果你人手短缺，排队机制将没有意义，它会变得更像你为客户制造的障碍。一个理想的排队机制最好是客户根本感觉不到，因为他们总是立即得到应答而不会进入呼叫保留状态。

注2. 市面上有几本书是讨论呼叫中心指标和有效的排队策略的，例如 James C.Abbott 的 *The Executive Guide to Call Center Metrics* (Robert Houston Smith).

注3. 结束时间（Wrapup time）是为了给坐席人员提供一段时间在完成接听后处理做记录或其它相关工作。它提供了几秒的宽限期给坐席人员在接听下一个电话前处理这些任务。

注4. 呼叫者的位置和等待时间仅在队列中有多于一人等待时才通知。

注5. 如果 Queue()的下一条 dialplan 语句没有定义，则这个呼叫将被挂断。