第十章 深入 Dialplan

目录

好了，你已经有了关于 Dialplan 的基本概念，但是你知道，还有更多的内容需要学习。如果你已经忘记了第六章的内容，请回去重读一遍。我们将进入一些更高级的话题。

10.1 表达式和变量操作

由于我们开始深入的研究 Dialplan，是时间向你介绍几个可以极大增强你操作 Dialplan 能力的工具。这些设计可以通过你定义的不同条件来做出决定，从而给你的 dialplan 增加难以置信的智能处理能力。戴上你的思考帽，让我们开始。

10.1.1基本表达式

表达式是变量（variables），操作符（operators），和值（values）的组合，你把它们串在一起，并产生一个结果。一个表达式可以测试一个数值，修改一个字符串，或者执行一个数学计算。让我们假设我们有一个变量称为 **COUNT**。用通俗语言描述，表达式可以是“COUNT 加 1”和“COUNT 除以 2”。每个这样的表达式都可以根据输入变量的值得到一个特定的结果或值。

在 Asterisk 中，表达式总是开始于“$”符号并紧跟着一个方括号，如下所示：

```
$[expression]
```

这样，我们可以写两个例子如下：

```
$[${COUNT} + 1]
$[${COUNT} / 2]
```

当 Asterisk 在 Dialplan 中遇到表达式时，它会将整个表达式替换为结果值。需要重点注意的是，值替换是发生在变量替换之后。为了说明这一点，让我们看下面的代码[注1]：

```
exten => 321,1,Set(COUNT=3)
    same => n,Set(NEWCOUNT=$[${COUNT} + 1])
    same => n,SayNumber(${NEWCOUNT})
```

在第一步（the first priority），我们给变量 COUNT 赋值为 3。

第二步（the second priority），只有一个应用（application）——**Set()**——被调用，但实际上发生了三件事：

1. Asterisk 将表达式中的**${COUNT}**替换为数值 3，表达式等效为：
   ```
   exten => 321,n,Set(NEWCOUNT=$[3 + 1])
   ```
2. Asterisk 计算这个表达式，把 1 加到 3 上，并用计算结果 4 替换：
3. 利用 **Set()**给变量 **NEWCOUNT** 赋值为 4

第三步简单调用 SayNumber()，用来念出变量**${NEWCOUNT}**的当前值。

请将这个例子在你的 dialplan 上试一下。

10.1.2操作符（Operators）

当你创建一个 Asterisk 的 Dialplan 时，你实际上是在用一种专门的脚本语言编程。这意味着 Asterisk dialplan——像任何编程语言一样——能够识别被称为操作符（operators）的符号并允许你操作变量（variables）。让我们看一下 Asterisk 中可用操作符的类型：

布尔操作符（*Boolean operators*）

这类操作符判断语句的“真实性”。在计算机术语中，这实际上是指这个语句是什么还是不是什么（非零或零，真或假，开或关，等等）。布尔操作符是：

expr1 | expr2

这个操作符（称为“或”操作符，或者“管道”）当 ***expr1*** 为真时（不是空字符串也不是零）返回 ***expr1*** 的值。否则，它返回 ***expr2*** 的值。

expr1 & expr2

这个操作符（称为“与”）当两个表达式都为真时（例如，没有一个表达式的值是空字符串或者零）返回 ***expr1*** 的值。否则，它返回零。

expr1 {=, >, >=, <, <=, !=} expr2

这些表达式返回整数比较结果如果两个参数都是整数；否则，它们返回字符串比较结果。如果两个参数的关系为真的话，返回值为 **1**；如果两个参数的关系为假则返回值为 **0**。（如果你执行字符串比较，将采用与你的操作系统本地设置一致的方法。）

算术操作符（*Mathematical operators*）

希望执行一个计算？你将需要下述操作符之一：

expr1 {+, -} expr2

这些操作符返回整数值参数加或减运算的结果。

expr1 {*, /, %} expr2

这些操作符分别返回整数值参数相乘，整除，或者求余运算的结果。

正则表达式操作符（*Regular expression operator*）

你也可以在 Asterisk 中使用正则表达式操作符：

expr1 : expr2

这个操作符匹配 expr1 和 expr2，其中 expr2 必须是正则表达式[注2]。这个正则表达式固定以一个隐含的 ^ 开始[注3]。

如果匹配成功并且 expr2 中至少包含一个正则子表达式——\(…\)——则返回与\1关联的字符串；否则，匹配操作符返回匹配的字符个数。如果匹配失败并且 expr2 中包含一个正则子表达式，则返回空（null）字符串；否则，返回 0。

在 Asterisk 1.0 版本中的语法分析程序非常简单，所以它需要你在操作符与任何其它值之间至少放一个空格。因此，下面的表达式可能并不会像你期待的那样工作：

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

这个表达式将给 **TEST** 赋值为字符串“2+1”，而不是值 3。为了纠正这个错误，我们需要在操作符两边放上空格，像这样：

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

不过这种做法在当前版本的 Asterisk 中不再需要，因为现在的词法分析程序已经提供了足够的容错处理能力。然而，出于加强可读性的目的，我们仍旧推荐在操作符周围增加空格。

如果需要将文本附加到一个变量的开始或结尾，只要简单的把它们放在一起就可以了，像这样：

```
exten => 234,1,Set(NEWTEST=blah${TEST})
```

10.2 Dialplan 函数（Functions）

Dialplan 函数允许你给你的表达式增加更多作用。你可以认为它们是智能变量。Dialplan 函数允许你计算字符串长度、日期和时间、MD5 校验和、等等，所有这些都可以包含在 dialplan 表达式中。

10.2.1 语法（Syntax）

Dialplan 函数的基本语法如下：

FUNCTION_NAME(*argument*)

你可以像引用变量名一样来引用函数名，但是引用函数的值，则需要增加一个$ 符号，并跟着一个大括号把函数整个括起来：

${*FUNCTION_NAME*(*argument*)}

函数调用中还可以再包含其它函数调用，如下：

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

你大概可以想到，你必须非常小心的匹配每一个括号和大括号。在前面的例子中，我们在每个括号的左半边都标注了数字，并在每个括号的右半边匹配位置标注了相同的数字。

10.2.2 Dialplan 函数举例

函数经常和 Set()应用一起使用来获得或者设置变量的值。作为一个例子，让我们看看**LEN()**函数。这个函数计算参数字符串的长度。让我们计算变量的字符串长度并回读这个长度给调用者：

```
exten => 123,1,Set(TEST=example)
     same => n,SayNumber(${LEN(${TEST})})
```

这个例子首先给 **TEST** 赋值为 example。字符串“example”被传递给 **LEN()**函数，**LEN()**将计算出字符串的长度，**7**。最后，**7** 被作为参数传递给 **SayNumber()**应用。

让我们看另一个例子。如果我们希望设置一个 Channel 的超时，我们可以使用 **TIMEOUT()**函数。**TIMEOUT()**函数可以接受如下三种参数之一：**absolute**，**digit**，和 **response**。为了使用**TIMEOUT()**设置数字超时，我们可以通过 **Set()**应用，如下所示：

```
exten => s,1,Set(TIMEOUT(digit)=30)
```

请注意这里没有用**${ }**把函数包裹起来。正象我们给变量赋值时一样，我们给函数赋值也不需要用**${ }**包装。

完整的可用函数列表可以通过在 Asterisk 命令行接口下输入 *core show functions* 获得。

10.3 条件跳转（Conditional Branching）

现在，你已经学习了一些关于表达式和函数的知识，是使用它们的时候了。通过使用表达式和函数，你可以给你的 Dialplan 增加更多的高级逻辑。为了允许你的 dialplan 作决定，你可以使用条件跳转。让我们近距离观察一下。

10.3.1 The GotoIf() Application

条件跳转的关键是 GotoIf() 应用。GotoIf()对表达式求值，并根据结果的真假把调用转向不同的目的。

GotoIf()使用特别的语法，通常称作条件语法（*conditional syntax*）：

GotoIf(*expression*?*destination1*:*destination2*)

如果表达式（expression）的值为真，调用被转向 **destinations1**。如果表达式的值为假，调用被转向 **destionation2**。那么，什么是真，什么是假呢？ 空字符串（empty string）和数字 0 被判断为假。任何其它值均被视为真。

条件跳转的目的地可以是下列情况之一：

- 同一个 extension 下的 priority label，例如 **weasels**
- 同一个 context 下的 extension 和 priority 组合，例如 **123,weasels**
- Context, extension,和 priority 的组合，例如 **incoming,123,weasels**

条件跳转两个目的地中的任意一个都可以被忽略，但是不能两个都被忽略。当执行到被忽略的目的地时，Asterisk 将简单的执行当前 extension 中的下一个 priority。

举例如下：

```
exten => 345,1,Set(TEST=1)
    same => n,GotoIf($[${TEST} = 1]?weasels:iguanas)
    same => n(weasels),Playback(weasels-eaten-phonesys)
    same => n,Hangup()
    same => n(iguanas),Playback(office-iguanas)
    same => n,Hangup()
```

你会注意到我们在每次使用 Playback()应用之后都跟随了一个 Hangup()应用。这么做是为了当我们跳转到 weasels 时，这个呼叫会在执行播放 *office-iguanas* 前被停止。一个 extensions 被划分为多个部分（彼此间被 hangup()命令隔离开）的做法越来越流行，每个部分都有一系列步骤跟随在 GotoIf()之后。

仅提供假条件路径

如果我们愿意，我们可以像这样重新精心设计前面的例子：

```
exten => 345,1,Set(TEST=1)
    same => n,GotoIf($[${TEST} = 1]?:iguanas)      ; we don't have the weasels label anymore,
                                                   ; but this will still work
    same => n,Playback(weasels-eaten-phonesys)
    same => n,Hangup()
```

```
same => n(iguanas),Playback(office-iguanas)
same => n,Hangup()
```

在 ? 和 : 之间什么也没有，因此当表达式的值为真时，将继续执行下个步骤。由于这正是我们希望的，所以不再需要 label weasels。

我们不愿意推荐这种做法，因为它不易读。但是你会看到这种风格的 dialplans，所以你最好知道这种语法是完全正确的。

典型地，如果你希望有一种结构可以避免 Asterisk 持续执行跳转后的语句，大概更好的方法是跳转到一个独立的 extensions 中而不是跳转到一个 priority labels。如果有什么区别的话，这种结构还将使 dialplan 更加清晰易读。我们将重写前面的例子如下：

```
exten => 345,1,Set(TEST=1)
    same => n,GotoIf($[${TEST} = 1]?weasels,1:iguanas,1)    ; now we're going to
                                                            ; extension,priority
exten => weasels,1,Playback(weasels-eaten-phonesys)         ; this is NOT a label.
                                                            ; It is a different extension
    same => n,Hangup()
exten => iguanas,1,Playback(office-iguanas)
    same => n,Hangup()
```

通过改变在第一行给 TEST 的赋值，你可以让你的 Asterisk 服务器播放不同的问候语。

让我们再看一下另一个条件跳转的例子。这一次，我们使用 Goto()和 GotoIf()来从 10 开始倒数，然后挂断电话：

```
exten => 123,1,Set(COUNT=10)
    same => n(start),GotoIf($[${COUNT} > 0]?:goodbye)
    same => n,SayNumber(${COUNT})
    same => n,Set(COUNT=$[${COUNT} - 1])
    same => n,Goto(start)
    same => n(goodbye),Hangup()
```

让我们分析下这个例子。在第一步，我们给变量 **COUNT** 赋值为 10。接着，我们检查 **COUNT** 是否大于 **0**。如果结果为真，我们就继续执行下一步（请不要忘记，如果我们略去了 **GotoIf()** 中的跳转地址，Asterisk 将继续执行下一步）。从那里，我们读出这个数字，执行 **COUNT** 减 1，然后再跳转到 priority label 为 start 的地方。如果 COUNT 小于或等于 0，Asterisk 将转向 priority label 为 goodbye 的语句，然后这个呼叫被挂断。

经典的条件跳转例子被昵称为“反女朋友逻辑”。如果来电号码匹配为你的前女友，Asterisk 将播放一段不同于向其他来电者播放的信息。虽然这有点简单和原始，但它是一个非常好的学习 Asterisk dialplan 中条件跳转的例子。

这个例子会用到 **CALLERID** 函数，这个函数允许我们获得来电的号码。为了这个例子让我们假定受害人的电话号码是 888-555-1212：

```
exten => 123,1,GotoIf($[${CALLERID(num)} = 8885551212]?reject:allow)
    same => n(allow),Dial(DAHDI/4)
    same => n,Hangup()
    same => n(reject),Playback(abandon-all-hope)
    same => n,Hangup()
```

在第一步，我们调用 **GotoIf()**应用。它告诉 Asterisk 跳转到 priority label 为 **reject** 的语句，当

来电号码是 **8885551212** 时，否则跳转到 priority label 为 **allow** 的语句（我们可以简单的略去这个 label name，因为 GotoIf()继续执行就是这一句）。如果来电号码匹配，Asterisk 控制呼叫跳转到 **reject**，在这里将播放一个不友好的信息给这个不受欢迎致电人。否则，Asterisk 会通过 **DAHDI/4** 联系你。

10.3.2 基于时间的条件跳转 GotoIfTime()

另一个在你的 dialplan 中使用条件跳转的办法是 GotoIfTime()应用。与 **GotoIf()**根据表达式的值决定怎么做不同，**GotoIfTime()**着眼于当前系统时间并使用它决定是否需要在 dialplan 中执行不同的跳转。

这个应用的最明显用途是在上班及下班时间给致电者提供不同的问候语。

GotoIfTime()的语法如下：

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

简单的说，GotoIfTime()将呼叫处理跳转到一个指定的 label，当当前日期和时间匹配由 ***times, days_of_week, days_of_month***，以及 ***months*** 指定的条件时。让我们仔细看一下每个参数：

times

这是一个或多个时间范围列表，采用 24 小时制。作为一个例子，上午 9:00 到下午 5:00，将被指定为 **09:00-17:00**。一天开始于 0:00，结束于 23:59。

值得注意的是，时间循环会被正确的处理。因此，如果你需要定义下班时间，你可以在 times 参数的位置写上 18:00-9:00，它将如你期望的执行。注意这个技术对 GotoIfTime()的其它部分也有效。例如，你可以写 sat-sun 来指定周末。

days_of_week

这是一个或多个星期列表。日期被指定为 **mon, tue, wed, thu, fri, sat, and/or sun**。周一到周五被表示为 **mon-fri**。周二和周四被表示为 **tue&thu**。

注意，你可以指定范围和单个日期的组合，例如：sun-mon&wed&fri-sat，或者，更简单：wed&fri-mon。

days_of_month

这是一个日期的列表。日期用数字 **1** 到 **31** 表示。7 号到 12 号被表示为 **7-12**，15 号和 30 号被表示为 **15&30**。

months

这是一个或多个月份列表。月份被记为：**jan-apr** 表示范围，用&符号连接起来的单个月份表示一组不连续的月份，例如 **jan&mar&jun**。你也可以将它们这样组合：**jan-apr&jun&oct-dec**。

如果你希望某个参数对于任意值均匹配，你只要简单的放一个* 在这个参数的位置就可以了。***label*** 参数可以是下列之一：

- 在同一个 extension 中的 priority label，例如 **time_has_passed**
- 在同一个 context 中的 extension 和 priority label 组合，例如 **123,time_has_passed**
- Context, extension, 和 priority 的组合，例如 **incoming,123,time_has_passed**

现在我们已经讨论了语法，下面让我们看几个例子。下面的例子将匹配早 9:00 到下午 5:59，从周一到周五，在每月的任何一天，在每年的任何一个月：

```
exten => s,1,GotoIfTime(09:00-17:59,mon-fri,*,*?open,s,1)
```

如果在给定的时间内致电，呼叫处理会跳转到名为 **open** 的 context 中的 s extension 的第一句执行。如果在指定的时间外致电，呼叫处理会转到当前 extension 的下一句执行。这使得你能轻易的区分不同的时间段，如下例所示（注意，你应当总是把常用的时间匹配放在不常用的时间匹配之前）：

```
; If it's any hour of the day, on any day of the week,
; during the fourth day of the month, in the month of July,
; we're closed
        exten => s,1,GotoIfTime(*,*,4,jul?closed,s,1)
; During business hours, send calls to the open context
        same => n,GotoIfTime(09:00-17:59,mon-fri,*,*?open,s,1)
        same => n,GotoIfTime(09:00-11:59,sat,*,*?open,s,1)
; Otherwise, we're closed
        same => n,Goto(closed,s,1)
```

如果你遇到有人问这种问题，“为什么我指定的是 17:58 而现在已经是 17:59 了。为什么 Asterisk 还在做同样的操作？”那么你需要注意 GotoIfTime()应用的时间粒度是 2 分钟。因此，如果你指定 18:00 作为一个周期的结束时间，系统会持续执行到 18:01:59。

注释：

注1. 请记住，当你引用一个变量是，你可以直接用变量名调用它；但是当你引用变量值时，你必须使用$符号紧接一个用括弧括起来的变量名。

注2. 欲了解更多关于正则表达式的知识，最棒的参考书籍是 Jeffrey E.F.Friedl's *Mastering Regular Expressions* (O'Reilly)，或者访问 http://www.regular-expressions.info

注3. 如果你不知道 ^ 在正则表达式代表什么，你需要简单阅读一下 *Mastering Regular Expressions*。这本书将改变你的人生。