CTI使用手册

作者:祁伟

时间:2013年12月23日

**目录**

[前言 3](#_Toc389581272)

[1 SCM概述 3](#_Toc389581273)

[1.1 SCM介绍 3](#_Toc389581274)

[1.2 基本概念 3](#_Toc389581275)

[1.3 定义和缩写词 3](#_Toc389581276)

[2 SCM功能 3](#_Toc389581277)

[2.1 功能介绍 3](#_Toc389581278)

[2.2 状态跳转功能 3](#_Toc389581279)

[2.3 事件匹配 3](#_Toc389581280)

[2.4 日志 3](#_Toc389581281)

[2.5 发送外部消息 3](#_Toc389581282)

[2.6 script 3](#_Toc389581283)

[2.7 定时器 3](#_Toc389581284)

[2.8 JavaScript 全局变量 3](#_Toc389581285)

[2.9 JavaScript全局函数 3](#_Toc389581286)

[2.10 系统变量 3](#_Toc389581287)

[2.11 事件变量 3](#_Toc389581288)

[2.12 配置文件 3](#_Toc389581289)

[3 脚本元素 3](#_Toc389581290)

[3.1 <scm> 3](#_Toc389581291)

[*3.1.1* 属性明细 3](#_Toc389581292)

[*3.1.2* 子元素 3](#_Toc389581293)

[3.2 <state> 3](#_Toc389581294)

[*3.2.1* 属性明细 3](#_Toc389581295)

[*3.2.2* 子元素 3](#_Toc389581296)

[3.3 <event> 3](#_Toc389581297)

[3.3.1 属性明细 3](#_Toc389581298)

[*3.3.2* 子元素 3](#_Toc389581299)

[3.4 <transition> 3](#_Toc389581300)

[*3.4.1* 属性明细 3](#_Toc389581301)

[*3.4.2* 子元素 3](#_Toc389581302)

[3.5 <onentry> 3](#_Toc389581303)

[*3.5.1* 属性明细 3](#_Toc389581304)

[*3.5.2* 子元素 3](#_Toc389581305)

[3.6 <onexit> 3](#_Toc389581306)

[*3.6.1* 属性明细 3](#_Toc389581307)

[*3.6.2* 子元素 3](#_Toc389581308)

[3.7 <log> 3](#_Toc389581309)

[*3.7.1* 概述 3](#_Toc389581310)

[*3.7.2* 属性明细 3](#_Toc389581311)

[*3.7.3* 子元素 3](#_Toc389581312)

[3.8 <timer> 3](#_Toc389581313)

[*3.8.1* 属性明细 3](#_Toc389581314)

[*3.8.2* 子元素 3](#_Toc389581315)

[3.9 <script> 3](#_Toc389581316)

[*3.9.1* 属性明细 3](#_Toc389581317)

[*3.9.2* 子元素 3](#_Toc389581318)

[3.10 <send> 3](#_Toc389581319)

[*3.10.1* 属性明细 3](#_Toc389581320)

[*3.10.2* 子元素 3](#_Toc389581321)

[3.11 <datamodel> 3](#_Toc389581322)

[*3.11.1* 属性明细 3](#_Toc389581323)

[*3.11.2* 子元素 3](#_Toc389581324)

[3.12 <data> 3](#_Toc389581325)

[*3.12.1* 属性明细 3](#_Toc389581326)

[*3.12.2* 子元素 3](#_Toc389581327)

[3.13 <functionmodel> 3](#_Toc389581328)

[*3.13.1* 属性明细 3](#_Toc389581329)

[*3.13.2* 子元素 3](#_Toc389581330)

[3.14 <function> 3](#_Toc389581331)

[*3.14.1* 属性明细 3](#_Toc389581332)

[*3.14.2* 子元素 3](#_Toc389581333)

[4 JS变量 3](#_Toc389581334)

[4.1 系统变量 3](#_Toc389581335)

[4.1.1 \_event对象 3](#_Toc389581336)

[4.1.2 \_event.\_serviceid 3](#_Toc389581337)

[4.1.3 \_event.\_sessionid 3](#_Toc389581338)

[4.1.4 \_event.\_name 3](#_Toc389581339)

[4.1.5 \_event. \_app\_id 3](#_Toc389581340)

[4.1.6 \_event. \_script\_id 3](#_Toc389581341)

[4.1.7 \_event. \_rest\_host 3](#_Toc389581342)

[4.1.8 \_event. \_rest\_port 3](#_Toc389581343)

[4.1.9 \_event. \_script\_text 3](#_Toc389581344)

[4.1.10 \_event. \_callback\_url 3](#_Toc389581345)

[4.1.11 \_event.\_ thirdip 3](#_Toc389581346)

[4.1.12 \_event.\_thirdport 3](#_Toc389581347)

[4.1.13 \_event.\_msgtype 3](#_Toc389581348)

[4.1.14 \_event.\_bodydata。 3](#_Toc389581349)

[4.2 事件变量 3](#_Toc389581350)

[4.3 JS全局变量 3](#_Toc389581351)

[4.4 JS临时变量 3](#_Toc389581352)

[5 使用说明 3](#_Toc389581353)

[5.1 状态改变 3](#_Toc389581354)

[5.2 事件匹配 3](#_Toc389581355)

[5.3 日志 3](#_Toc389581356)

[5.4 定时器 3](#_Toc389581357)

[5.5 script 3](#_Toc389581358)

[5.6 回复消息到CallManager 3](#_Toc389581359)

[5.7 发送消息到第三方 3](#_Toc389581360)

[5.8 回复消息到巡检系统 3](#_Toc389581361)

[5.9 发送消息到巡检系统 3](#_Toc389581362)

[5.10 发送消息到RestServer 3](#_Toc389581363)

[5.11 全局变量 3](#_Toc389581364)

[5.12 全局函数 3](#_Toc389581365)

[5.13 系统变量 3](#_Toc389581366)

[5.14 事件变量 3](#_Toc389581367)

[6 脚本用例 3](#_Toc389581368)

[6.1 收到通知消息后，回复CallManager具体命令 3](#_Toc389581369)

[6.2 收到通知消息后，直接转给第三方服务器，收到第三方服务器反回的IVR命令后，再转给CallManager 3](#_Toc389581370)

[6.3 收到通知消息后，回复CallManager，把消息转给第三方服务器 3](#_Toc389581371)

[6.4 收到巡检系统的启动消息和停止消息，并主动发消息给巡检系统 3](#_Toc389581372)

[6.5 Agent\_Demo 3](#_Toc389581373)

[7 配置文件 3](#_Toc389581374)

[7.1 介绍 3](#_Toc389581375)

[7.2 基本元素 3](#_Toc389581376)

[7.2.1 <CONFIG> 3](#_Toc389581377)

[***7.2.1.1*** 属性明细 3](#_Toc389581378)

[***7.2.1.2*** 子元素 3](#_Toc389581379)

[7.2.2 <SERVER> 3](#_Toc389581380)

[***7.2.2.1*** 属性明细 3](#_Toc389581381)

[***7.2.2.2*** 子元素 3](#_Toc389581382)

[7.2.3 <IP> 3](#_Toc389581383)

[***7.2.3.1*** 属性明细 3](#_Toc389581384)

[***7.2.3.2*** 子元素 3](#_Toc389581385)

[7.2.4 <Port> 3](#_Toc389581386)

[***7.2.4.1*** 属性明细 3](#_Toc389581387)

[***7.2.4.2*** 子元素 3](#_Toc389581388)

[7.2.5 <StateMachine> 3](#_Toc389581389)

[***7.2.5.1*** 属性明细 3](#_Toc389581390)

[***7.2.5.2*** 子元素 3](#_Toc389581391)

[7.3 样例 3](#_Toc389581392)

[7.4 动态加载 3](#_Toc389581393)

[8 脚本语法要求 3](#_Toc389581394)

[8.1 介绍 3](#_Toc389581395)

[8.2 静态语法 3](#_Toc389581396)

[8.2.1 语法规则 3](#_Toc389581397)

[8.2.2 语法检测 3](#_Toc389581398)

[8.3 动态语法 3](#_Toc389581399)

# 前言

本文档是CTI使用人员的帮助文件，主要的阅读对象是CTI测试人员、IVR编辑器开发人员、任何手工编写CTI脚本人员。

本文档由五章组成，第一章CTI概述，讲述了CTI的作用，系统的功能，CTI基本概念，CTI能做什么。

第二章CTI功能，详细讲述了CTI所支持的功能。

第三章脚本元素，详细讲述了在脚本文件中所支持的元素，元素的属性和子元素。

第四章使用说明，讲述了如何编写一个CTI脚本并且在程序中使用。

第五章脚本语法要求，讲述了CTI脚本中需要遵循的语法规则。

# CTI概述

## CTI介绍

CTI技术是从传统的计算机电话集成(Computer Telephony Integration)技术发展而来的，最初是想将计算机技术应用到电话系统中，能够自动地对电话中的信令信息进行识别处理，并通过建立有关的话路连接，而向用户传送预定的录音文件、转接来话等。而到现在，CTI技术已经发展成“计算机电信集成”技术(Computer Telecommunication Integration)，即其中的“T”已经发展成“Telecommunication”，这意味着目前的CTI技术不仅要处理传统的电话语音，而且要处理包括传真、电子邮件等其它形式的信息媒体。

## 基本概念

CTI的基本概念是状态机，基本的状态机概念包括 状态（state）、 转换（transition）、 事件（event）。每个state都包含了event去定义如何匹配收到的事件消息。这个状态机总是处于一个状态，这个状态被成为当前状态。当遇到一个事件消息，这个状态机检查当前状态下的event，当找到一个与事件消息相匹配，会执行event下的transition动作，它从当前状态转移到transition指定的state，这个目标状态将变为新的当前状态。

在CTI的状态机中有几个扩展概念，在状态机匹配消息后会有其他动作，特别是，每个state都可能包含<onentry>和<onexit>动作，匹配消息后也可能包含其他动作。在状态机中还加入了对JavaScript脚本的支持，在事件匹配和执行动作中可以添加JavaScript脚本，完成一些自定义的动作。

## 定义和缩写词

1. state：状态。
2. transition：转换。
3. event：事件。
4. target：目标。
5. cond：条件，（condition）的缩写。
6. datamodel：数据模块。
7. data：数据元素。
8. functionmodel：函数集合。
9. function：函数元素。
10. id：对象标识。
11. description：描述。
12. name：名称。
13. type：类型。
14. onentry：进入。
15. onexit：退出。
16. log：日志。
17. script：脚本。
18. send：发送。
19. timer：定时器。
20. idexpr：id表达式。
21. interval：间隔。
22. JS：JavaScript。

# CTI功能

## 功能介绍

## 状态跳转功能

在状态机的可执行集合中可以包含<transition>元素，此元素是用来实现状态跳转功能，当遇到<transition>元素时，CTI会将状态改变为transition指定的目标状态。

## 事件匹配

状态机收到事件消息后，根据一定的规则对event进行匹配，匹配成功后会执行<event>元素下包含的执行集合。

## 日志

在状态机的可执行集合中可以包含<log>元素，当遇到log元素时，CTI会将log元素的内容记录到日志中。

## 发送外部消息

在状态机的可执行集合中可以包含<send>元素，当遇到send元素时，CTI会根据send元素的target属性，将内容发送到指定的目标。

## script

在状态机中可以执行JavaScript，在cond属性和<script>中的内容会被当作JavaScript脚本来执行。还可以包含script类型的日志内容和send子元素内容。

## 定时器

在状态机的可执行集合中可以添加<timer>元素，当遇到timer元素时，CTI会生成一个定时器，当定时器到期后，定时器会发送timer事件到状态机中。

## JavaScript 全局变量

在状态机脚本中可以定义在整个session中JavaScript都可以读写的全局变量。

## JavaScript全局函数

在状态机中可以预定义一部分比较常用的JavaScript函数为全局函数，在状态机脚本中所有的JavaScript中都可以使用这些函数。

## 系统变量

在状态机中事先定义好的变量，在状态机JavaScript脚本中可以直接读取。

## 事件变量

每次事件消息到来时生成的变量，只在此次事件中有效，在状态机JavaScript脚本中可以直接读取，当下次事件到来时会清除本次的事件变量。

## 配置文件

包括需要监听的端口号、状态机脚本配置等。

# 脚本元素

在CTI的状态机中，脚本文件是一个xml文件嵌入了部分JavaScript脚本来组成，状态机功能也是使用xml的元素来定义，所以预定义了一部分xml元素，这些元素在CTI脚本中是合法元素，能被CTI解析，未定义的元素不能被CTI解析。

## <fsm>

状态机脚本最外层元素，包含版本信息，实际的状态机都是它的子元素。注意在任意时刻，只有一个状态是活动状态。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| name | false | none | NMTOKEN | none | 任何有效的NMTOKEN | 状态机的名字，只是为了信息显示。 |
| version | false | none | decimal | none | 有效的数字 | 状态机版本号，例如 1.0 |

### 子元素

1、<state> 可以出现0到多次。

2、<datamodel> 定义部分或全部的数据模块，可以出现0到多次。

3、<functionmodel> 提供了定义JavaScript全局函数模块，可以出现0到多次。

一个符合逻辑的状态机必须至少包含一个<state>元素，当系统初始化时，状态机进入到第一个状态。

## <state>

此元素代表了一个状态。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| id | false | none | NMTOKEN | none | 任何有效的NMTOKEN | 状态的标志，在整个脚本中此id值应该唯一，如果没有此属性，transition将不能查找到此状态，也就是此状态不能作为目标状态。 |
| name | false | none | NMTOKEN | none | 任何有效的NMTOKEN | 状态的名称，只是为了信息显示。 |
| description | false | none | NMTOKEN | none | 任何有效的NMTOKEN | 状态的描述 |
| type | false | none | NMTOKEN | none | 任何有效的NMTOKEN | 为脚本编辑器预留的属性。 |

### 子元素

1、<onentry> 当进入<state>时执行的内容，可以出现0到多次。

2、<onexit> 当离开<state>时执行的内容，可以出现0到多次。

3、<event> 当遇到消息时匹配的事件元素，可以出现0到多次。

4、<state> 此状态的子状态，可以出现0到多次。

## <event>

此部分实现了状态机中消息的匹配，在<event>元素中包含的元素都是事件匹配后要执行的内容。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| event | false | none | NMTOKEN | true | 任何有效的NMTOKEN。 | 事件匹配时，状态机会在事件消息名称中查找此值，如果查找到，认为此属性匹配。 |
| cond | false | JavaScript脚本 | 布尔表达式 | true | JavaScript脚本表达式 | 事件匹配的一个附加条件，计算值为true时，此条件通过。 |

### 子元素

1、<transition> 当匹配event后执行的状态转换，可以出现0到多次。

2、<log> 当匹配event后执行的日志记录功能，可以出现0到多次。

3、<script> 当匹配event后执行的脚本功能，可以出现0到多次。

4、<send> 当匹配event后执行的和外部模块通信功能，可以出现0到多次。

5、<timer>当匹配event后执行的定时器功能，可以出现0到多次。

6、<raise>当匹配event后产生一个内部事件功能，可以出现0到多次。

【注意：如果省略event属性，认为事件消息名称是匹配的，如果省略cond属性，默认此值是true。】

【注意：如果第一个transition执行后，余下的元素都不再执行。】

## <transition>

transition 实现了在各个状态之间实现转换。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| cond | false | JavaScript表达式 | 布尔表达式 | true | JS布尔表达式 | 当表达式为’true’时，执行 transition。 |
| target | true | none | IDREFS | none | 指定的有效state id。 | 状态机会跳转到符合此id的state。 |

### 子元素

none

【当省略cond属性时，状态机会假定有一个默认值是true的cond属性。】

【一个符合逻辑的状态机脚本，必须指定 target。】

## <onentry>

当进入一个状态后包含在此元素下的内容都会被执行。

### 属性明细

none.

### 子元素

<onentry>下可以包含任何以下元素。

1、<log>，可以出现0到多次。

2、<script>，可以出现0到多次。

3、<send>，可以出现0到多次。

4、<timer>，可以出现0到多次。

5、<raise>，可以出现0到多次。

【注意：一个符合逻辑的状态机脚本在<onentry>中不能包含<transition>元素。】

## <onexit>

当离开一个状态后包含在此元素下的内容都会被执行。

### 属性明细

none.

### 子元素

<onexit>下可以包含任何以下元素。

1、<log>，可以出现0到多次。

2、<script>，可以出现0到多次。

3、<send>，可以出现0到多次。

4、<timer>，可以出现0到多次。

5、<raise>，可以出现0到多次。

【注意：一个符合逻辑的状态机脚本在<onexit>中不能包含<transition>元素。】

## <log>

### 概述

运行状态机时生成日志或者调试信息。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| level | false | fatal、warn、error、i**n**fo、debug、trace 之一 | NMTOKEN | info | fatal、warn、error、i**n**fo、debug、trace 之一 | 状态机输出的日志级别，如：fatal、warn、error、i**n**fo、debug、trace等。 |
| type | false | text 或者script | NMTOKEN | text | text或script | 指定此元素的内容是字符串常量或者是可执行脚本。 |
| cond | false | JavaScript表达式 | 布尔表达式 | true | JS布尔表达式 | 当表达式为’true’时，执行 log。 |

### 子元素

log元素中的内容是需要记录的日志内容，可以是字符串常量或者JS脚本。

## <timer>

当遇到此事件时，解析器会启动一个计时器，计时器超时后会向外部事件队列添加一个timer事件。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| idexpr | true | JS脚本表达式 | 可执行脚本 | none | JS表达式 | 解析器会根据此表达式产生一个timer ID，在产生的事件中会携带此ID. |
| interval | true | 整型值 | 整型值 | 0 |  | 事件定时间隔，单位毫秒。 |
| cond | false | JavaScript表达式 | 布尔表达式 | true | JS布尔表达式 | 当表达式为’true’时，执行 timer。 |

### 子元素

none

【注意：定时器的最小时间粒度是1毫秒。】

## <script>

<script>元素在状态机中添加了执行JS脚本能力。

### 属性明细

none

### 子元素

出现在<script>元素中的内容将会被当作JS执行。

## <send>

<send>元素是用来向状态机外部发送消息。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| target | true | 依赖平台的具体实现 | NMTOKEN | none | 依赖平台的具体实现 | 指定将要发送的目标。 |
| type | true | 依赖平台的具体实现 | NMTOKEN | none | 依赖平台的具体实现 | 指定发送消息的类型。依赖于具体的实现平台。 |
| cond | false | JavaScript表达式 | 布尔表达式 | true | JS布尔表达式 | 当表达式为’true’时，执行 transition。 |

### 子元素

xml类型的元素，子元素中的内容是send元素的扩展参数，在子元素中需要指定一个属性type，当type 属性为script时，此子元素的内容是JS脚本，需要在执行send元素的时候执行，其他情况或者是“text”，此子元素的内容是字符串常量。

子元素中至少要包含一个<cmddata>元素，是发送的消息内容。

## <raise>

<raise>元素会在当前会话中引起一个事件。注意，事件不会在当前事件执行完成前和所有的内部事件处理完成前处理。例如，假定<raise>元素首先发生在一个状态的<onentry>中的可执行内容ec1和ec2中，如果事件e1已经在内部事件队列中，当S状态进入后，这个事件将会被<raise>产生，直到ec1和ec2执行完毕和e1被处理。详细的事件处理见详细的FSM解释算法。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| event | true | none | NMTOKEN | none |  | 指定事件名称，它将会在 ‘event’属性中匹配。 |
| cond | false | 状态机可以执行的脚本 | 布尔表达式 | true | 状态机所支持的脚本表达式 | 执行元素匹配的一个条件，计算值为true时，此条件通过。 |

### 子元素

无

解析器将产生的事件放在状态机内部事件队列的结尾。

## <datamodel>

<datamodel>是用来封装任意数量的<data>元素的，每一项定义了一个单独的数据对象。

### 属性明细

none

### 子元素

<data>元素可以出现0到多次，每个实例被命名为数据元素。

## <data>

<data>元素是用来声明JS全局变量和初始值。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| id | true | NMTOKEN | ID | none | 符合JS变量命名规则。 | 此项数据的名称 |
| expr | false | 合法的JS表达式 | JS表达式 | none | 合法的JS右值表达式。 | 数据值的表达式。 |

### 子元素

none

【在data元素中定义的数据元素可以状态机文件中任何JS脚本中引用，此元素被关联在JS脚本的全局对象中。】

【在data元素中定义的id必须不能和系统变量和事件变量重名，系统变量和事件变量在以下会讲到。】

## <functionmodel>

<functionmodel>是用来封装任意数量的<function>元素的，每一项定义了一个单独JS全局函数。

### 属性明细

none

### 子元素

<function>元素可以出现0到多次，每个实例中包含一段合法的JS函数定义脚本。

## <function>

<function>元素是用来定义在JS脚本的全局函数。

### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| src | false | js 文件名称 | 文件名称 | none | 符合JS语法的文件的名称。 | 当此属性出现时，会编译指定的文件中JS内容。 |

### 子元素

在<function>元素中的内容是一段JS定义函数的脚本，声明的函数被关联在全局对象中，在状态机中任何JS中可以使用。

# JS变量

## 系统变量

在CTI脚本中为了能使用程序中的变量，事先在JS中定义了一组系统变量，并且和程序中的数据相关联，可以在JS中直接使用，这些变量都是只读变量。

### \_event对象

当一个状态机初始化后，CTI会自动生成一个“\_event”JS对象，并自动关联一些状态机生成时的属性。

### \_sessionid

当状态机启动当前session时FSM解析器必须绑定变量\_sessionid。解析器必须保持此变量直到session结束。

### \_name

在状态机启动加载时FSM解析器必须绑定变量\_name为<FSM>的’name’属性。解析器必须保持变量直到session结束。

### \_event.\_name

当一个事件消息到达后，CTI必须绑定消息的名称到\_event.\_name。保持变量直到下一次事件消息到来后更新。

### \_event.\_data

当一个事件消息到达后，SCM必须绑定此消息的消息体到\_event.\_data。保持变量直到下一次事件消息到来后更新。

## 事件变量

在系统中，在消息中会携带参数作为此次事件消息的参数，在CTI中将这些参数动态关联到JS的\_event对象中，关联后，此变量在此次事件内有效，当下一次事件消息到来时，CTI会删除上次事件消息关联的参数，重新关联本次事件消息中的参数。

【注意：事件变量的值可以在脚本中修改。】

【注意：事件变量和系统变量都是关联在了\_event类中，所以事件变量的变量名称不能使用系统变量中的变量名称。】

## JS全局变量

在脚本的JS代码中可以直接定义JS全局变量，也可以使用<data>元素预先自定义JS的全局变量，在整个session中有效。变量名称要求符合xml格式和JS命名规则，注意也不能使用系统变量和事件变量中的名称（如\_event.\*\*\*）。

在<data>元素定义JS全局变量的规则参考3.12<data>，以下是一个样例：

<data id=”startTime” expr=”new Date();”/>

定义了一个startTime变量，变量值为session的开始时间（因为data元素是在session开始时执行的）。

## JS临时变量

参考JS语法规则。

# 使用说明

## 状态改变

每个state都包含了event去定义如何匹配收到的事件消息。当前状态机总是处于一个状态，当前状态被成为当前状态。当遇到一个事件消息，当前状态机检查当前状态下的event，当找到一个与事件消息相匹配，会执行event下的transition动作，它从当前状态转移到transition指定的state，这个目标状态将变为新的当前状态。

因此，下面的例子，当遇到 事件 e 并且x等于1 时，跳转到s1，当遇到事件e并且x不等于1时，跳转的s2，其他情况跳转到s3。

<state id=s">

<event event="e">

<transition cond="x==1" target="s1"/>

<transition target="s2"/>

</event>

<event >

<transition target="s3"/>

</event>

</state>

## 事件匹配

一个 <state> 元素可以嵌套<state>元素，这样的状态叫做组合状态并且称其为父状态，嵌套状态称其为子状态，这些子状态也可以有自己的嵌套并且可以嵌套任意深度。最终，我们会到达未包含任何子状态的状态。这样的状态成为原子状态。

组合状态会影响到消息匹配的选择，当查找event元素时，状态机首先查找当前的活动状态，如果在当前状态中没有匹配，状态机会查找它的父状态，然后是父状态的父状态，以此类推。因此在祖先状态中event匹配消息将是子孙状态下没有匹配到event元素的默认匹配。如果在任何状态下都没有配合，事件将被丢弃。

因此，下面的例子中，假如当前的状态是s3，当遇到事件e31并且x等于2时，跳转到s2；当遇到事件e31并且x不等于2时，跳转到s1；当遇到事件e32时跳转的s2；当遇到事件s21时，跳转到s1；当遇到e13时，跳转到s3；其他情况将事件丢弃。

<state id="s">

<event event="e13">

<transition target="s3"/>

</event>

<state id="s1"></state>

<state id="s2">

<event event="e21">

<transition target="s1"/>

</event>

<state id="s3">

<event event="e31" cond="x==2">

<transition target="s2"/>

</event>

<event event="e31" >

<transition target="s1"/>

</event>

<event event="e32">

<transition target="s2"/>

</event>

<state id="s4">

<event event="e31">

<transition target="s3"/>

</event>

</state>

</state>

</state>

</state>

## 日志

在CTI脚本中定义了日志元素，可以将日志元素的内容记录到后台日志文件中，日志元素的详细描述见3.7<log>。日志元素中可以支持JS内容。

因此，在下面的例子中，假如serviceid=40012345678，callid=12324324894303，当状态机初始化时，会记录“start a session,serviceid=40012345678,sessionid=12324324894303”日志，当收到事件“e”时，会记录“ recive a event,name=e”和“leave init state.”。

<scm>

<state id=”init”>

<onentry>

<log type=”script”> “start a session,serviceid=”+\_event.\_serviceid+”,sessionid=”+\_event.\_sessionid;</log>

</onentry>

<event event=”e”>

<log type=”script”>”recive a event,name=”+\_event.\_name;</log>

<transition target=”s1”/>

</event>

<onexit>

<log>leave init state.</log>

</onexit>

</state>

<state id=”s1”>

</state>

</scm

## 定时器

在SCM脚本中定义了定时器元素，定时器元素的详细描述见3.8<timer>。定时器元素可实现简单的定时功能，定时器元素会产生一个定时器，当定时器到期后，此session会收到一个定时器事件消息。

因此，在以下的例子中，当收到事件”e”后，SCM会产生一个定时器，当30秒后，定时器到期，状态机会收到一个timer事件，匹配事件后会记录下日志”timer test”。

<scm>

<state id=”init”>

<event event=”e”>

<script>timerid=” timertest123”;</script>

<timer idexpr=”timertest123” interval=”30000”/>

</event>

<event event=”timer” cond=”\_event.id==timerid”>

<log>timer test</log>

</event>

</state>

</scm>

## script

script元素提供了在状态机脚本中执行JS的功能，在script元素中的内容都会被当作JS脚本来执行，5.4定时器的例子中已经包含了script元素的使用。

当收到事件”e”后，会生成一个timerid的全局变量，赋值为” timertest123”;

## 回复消息到CallManager

回复消息给CallManager是SCM的一个重要功能，在状态机脚本中使用send元素来实现。当执行send元素时，SCM会根据它的target属性来寻找发送目标，type属性决定了回复消息还是请求消息，send元素的子元素<cmddata>为发送的消息内容。

在下面的例子中，假如当前状态是”init”，当遇到事件”startservice时”，跳转到”play”，执行<onentry>下的<send>元素发送一个消息给CallManager。

<scm>

<state id="init">

<event event="startservice">

<transition target="play"/>

</event>

</state>

<state id="play">

<onentry>

<send type="response" target="callmanager">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<Get action="handle-user-input.jsp" numdigits="1">

<Play>menu.wav</Play>

</Get>

<Play>sorrybye.wav</Play>

<Redirect>/welcome/voice?id=9999</Redirect>

</Response>]]>

</cmddata>

</send>

</onentry>

</state>

</scm>

send元素的type属性为response，所以SCM会自动添加http头部，实际发送的消息内容为：

HTTP/1.1 200 OK

Date:Mon Dec 23 14:41:47 2013

Accept: application/xml

Content-Type: application/xml;charset=utf-8

Connection:close

Content-Type:text/plain

Content-Length:240

<?xml version="1.0" encoding="UTF-8"?>

<Response>

<Get action="handle-user-input.jsp" numdigits="1">

<Play>menu.wav</Play>

</Get>

<Play>sorrybye.wav</Play>

<Redirect>/welcome/voice?id=9999</Redirect>

</Response>

## 发送消息到第三方

发送消息到第三方是SCM的一个重要功能，在状态机脚本中使用send元素来实现。当执行send元素时，SCM会根据它的target属性来寻找发送目标，type属性决定了是回复消息还是请求消息，send元素的子元素<cmddata>为发送的消息内容。

在下面的例子中，假如当前状态是”init”，当遇到事件”startservice” 时，跳转到”play”状态，执行<onentry>下的<send>元素发送一个消息给第三方。

<scm>

<state id="init">

<event event="startservice">

<transition target="play"/>

</event>

</state>

<state id="play">

<onentry>

<send type="post" target="third" >

<cmddata type="script">\_event.\_bodydata</cmddata>

</send>

</onentry>

</state>

</scm>

在type和target属性中指定了此操作是发送给第三方的消息，<cmddata>子元素中包含的是转发给第三方的命令内容，在此处是通过JS脚本获取的系统变量\_bodydata。

假如SCM收到的消息为：

POST /192.168.111.27:8812/serviceid=4001001200/callid=131118153908189/ivrnotify HTTP/1.1

Accept: \*/\*

Content-Type: text/xml; charset=utf-8

User-Agent: Hisun

Host: 127.0.0.1:8080

Cache-Control: no-cache

Connection: close

Content-Length: 211

post http://42.121.15.99:8080/startservice HTTP/1.1

Accept: \*/\*

Content-Type: text/xml; charset=utf-8

User-Agent: Hisun

Host: 127.0.0.1:8080

Cache-Control: no-cache

Connection: close

Content-Length: 0

则转发给第三方的消息为：

post http://42.121.15.99:8080/startservice HTTP/1.1

Accept: \*/\*

Content-Type: text/xml; charset=utf-8

User-Agent: Hisun

Host: 127.0.0.1:8080

Cache-Control: no-cache

Connection: close

Content-Length: 0

## 回复消息到巡检系统

在状态机脚本中使用send元素来实现。当执行send元素时，SCM会根据它的target属性来寻找发送目标，type属性决定了回复消息还是请求消息，send元素的子元素<cmddata>为发送的消息内容。

在下面的例子中，假如当前状态是”init”，当遇到事件”start”， <event>下的第一个<send>元素回复一个消息给巡检系统。

<?xml version="1.0" encoding="UTF-8"?>

<scm>

<functionmodel>

<function src="../etc/time.js"></function>

<function src="../etc/uuid.js"></function>

</functionmodel>

<state id="init">

<event event="start">

<send type="response" target="inspector">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<statusCode>000000</statusCode>

</Response>]]>

</cmddata>

</send>

<send type="post" target="inspector">

<notify\_seq type="script">uuid();</notify\_seq>

<notify\_type>0</notify\_type>

<notify\_text><![CDATA[notify\_test 测试内容。]]></notify\_text>

</send>

</event>

<event event="stop">

<send type="response" target="inspector">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<statusCode>000000</statusCode>

</Response>]]>

</cmddata>

</send>

</event>

</state>

## 发送消息到巡检系统

在状态机脚本中使用send元素来实现。当执行send元素时，SCM会根据它的target属性来寻找发送目标，type属性决定了是回复消息还是请求消息，send元素的子元素为发送的消息内容。

在下面的例子中，假如当前状态是”init”，当遇到事件”start” 时，<event> 下的第二个<send>元素发送一个消息给巡检系统。

<?xml version="1.0" encoding="UTF-8"?>

<scm>

<functionmodel>

<function src="../etc/time.js"></function>

<function src="../etc/uuid.js"></function>

</functionmodel>

<state id="init">

<event event="start">

<send type="response" target="inspector">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<statusCode>000000</statusCode>

</Response>]]>

</cmddata>

</send>

<send type="post" target="inspector">

<notify\_seq type="script">uuid();</notify\_seq>

<notify\_type>0</notify\_type>

<notify\_text><![CDATA[notify\_test 测试内容。]]></notify\_text>

</send>

</event>

<event event="stop">

<send type="response" target="inspector">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<statusCode>000000</statusCode>

</Response>]]>

</cmddata>

</send>

</event>

</state>

</scm>

在type和target属性中指定了此操作是发送给巡检系统的消息，< notify\_seq >子元素中是一个生成uuid的js脚本，< notify\_type >子元素是发送给巡检系统的消息类型，<notify\_text>子元素是消息内容。

## 发送消息到RestServer

在状态机脚本中使用send元素来实现。当执行send元素时，SCM会根据它的target属性来寻找发送目标，type属性决定了是回复消息还是请求消息，send元素的子元素<cmddata>为发送的消息内容。

在下面的例子中， <send>元素发送一个消息给RestServer。

<send type="response" target="rest">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<statusCode>000000</statusCode>

</Response>]]>

</cmddata>

</send>

在type和target属性中指定了此操作是发送给RestServer的消息。

## 全局变量

全局变量的使用见4.3JS全局变量。

## 全局函数

在脚本JS中如果经常需要使用某段代码，可以将此段代码编写成JS全局函数，这样在脚本中使用时只需要调用函数名即可。在SCM脚本中的全局函数是在<functionmodel>下<function>元素定义的，<function>的详细描述见3.14<function>。以下是定义了一个返回当前小时的函数样例：

<function>function getHour(){ var today= new Date(); return today.getHours(); }</function>

当在脚本中的JS中使用getHour()函数时，会返回当前的小时。

## 系统变量

系统变量见4.1系统变量。

## 事件变量

事件变量见4.2事件变量。

# 脚本用例

## 收到通知消息后，回复CallManager具体命令

脚本文件a.xml。

重要部分说明：

<send type="response" target="callmanager">

<cmddata><![CDATA[<?xml version="1.0" encoding="UTF-8"?>

<Response>

<Get action="handle-user-input.jsp" numdigits="1">

<Play>menu.wav</Play>

</Get>

<Play>sorrybye.wav</Play>

<Redirect>/welcome/voice?id=9999</Redirect>

</Response>]]>

</cmddata>

</send>

在type 和target属性中指定了此操作是发送给callmanager的回复命令，<cmddata>子元素中包含的是回复的命令内容。SCM会在消息头部自动添加：

HTTP/1.1 200 OK

Date:Thu Dec 19 13:49:43 2013

Accept: application/xml

Content-Type: application/xml;charset=utf-8

Connection:close

Content-Type:text/plain

Content-Length:240

最终向callmanager回复的完整命令为：

HTTP/1.1 200 OK

Date:Thu Dec 19 13:49:43 2013

Accept: application/xml

Content-Type: application/xml;charset=utf-8

Connection:close

Content-Type:text/plain

Content-Length:240

<?xml version="1.0" encoding="UTF-8"?>

<Response>

<Get action="handle-user-input.jsp" numdigits="1">

<Play>menu.wav</Play>

</Get>

<Play>sorrybye.wav</Play>

<Redirect>/welcome/voice?id=9999</Redirect>

</Response>

【注意：消息头部的日期和长度部分为发送时动态添加的。】

## 收到通知消息后，直接转给第三方服务器，收到第三方服务器反回的IVR命令后，再转给CallManager

脚本文件b.xml

重要部分说明：

<send type="response" target="callmanager">

<cmddata type="script">\_event.\_bodydata.substring(\_event.\_bodydata.indexOf("\r\n\r\n")+4);</cmddata>

</send>

在type和target属性中指定了此操作是回复给CallManager的消息，<cmddata>子元素中包含的是回复给CallManager的命令内容，在此处是通过js脚本获取的事件消息中的bodydata 中的部分内容。

## 收到通知消息后，回复CallManager，把消息转给第三方服务器

脚本文件c.xml

重要部分说明：

<send type="post" target="third">

<cmddata type="script">\_event.\_bodydata;</cmddata>

</send>

在type和target中制定了此操作是转发给第三方的消息，<cmddata>子元素中包含的是转发给第三方的命令内容，在此处是通过js脚本获取的事件消息中的bodydata。

## 收到巡检系统的启动消息和停止消息，并主动发消息给巡检系统

脚本文件d.xml

重要部分说明：

<send type="post" target="inspector">

<notify\_seq type="script">uuid();</notify\_seq>

<notify\_type>0</notify\_type>

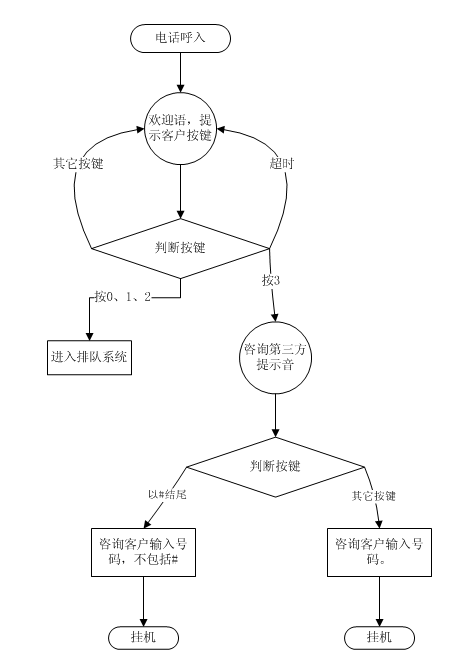
<notify\_text><![CDATA[notify\_test 测试内容。]]></notify\_text>

</send>

在type和target 中指定了此操作是主动发送消息给巡检系统，send元素的子元素是按照巡检系统要求发送的数据内容。

## Agent\_Demo

脚本文件Agent\_Demo.xml。Agent\_Demo是根据CallManager脚本文件内容使用SCM脚本重新编写的，其中的业务流程为：



# 配置文件

## 介绍

配置文件的功能是为了存储一些在程序中需要经常变动的参数，每次程序启动时（或重新加载配置时）读取配置文件，根据配置文件的内存初始化程序中的对象。

## 基本元素

### <CONFIG>

【说明：配置文件的根元素，所有的配置内容都是此元素的子元素。】

#### 属性明细

none

#### 子元素

1、<SERVER>，<SERVER>元素是 SCM 配置的根元素，只能出现一次。

### <SERVER>

【说明：SERVER元素是SCM模块配置的根元素，所有SCM的配置内容都在此元素下。】

#### 属性明细

none

#### 子元素

1、IP，IP元素是SCM监听的IP地址，只能出现一次。

2、Port，Port元素是SCM监听消息的端口，只能出现一次。

3、StateMachine，StateMachine元素是SCM中状态机脚本配置，可以出现0到多次。

### <IP>

#### 属性明细

none

#### 子元素

元素中的内容是IP地址格式的字符串。

【注意：元素中的内容一定是\*.\*.\*.\*格式的IPV4地址，必须是一个合法的地址。】

### <Port>

#### 属性明细

none

#### 子元素

元素中的内容是一个整型的端口值。

【注意，元素中的内容必须是一个合法的IP端口，此端口必须是一个没有使用过的端口，不然会监听失败。】

### <StateMachine>

#### 属性明细

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **名称** | **必须** | **约束** | **类型** | **默认值** | **有效值** | **描述** |
| serviceid | true | NMTOKEN | NMTOKEN | none | 系统中存在的serviceid。 | 此状态机脚本的serviceid值，用来标识此脚本属于那个业务。 |

#### 子元素

元素中的内容是状态机脚本的路径，是执行程序的相对路径。

## 样例

<CONFIG>

<SERVER>

<IP>0.0.0.0</IP>

<Port>8811</Port>

<StateMachine serviceid="4001001200">../etc/StateMachine.xml</StateMachine>

<StateMachine serviceid="4001001201">../etc/StateMachine.xml</StateMachine>

<StateMachine serviceid="4001001200">../etc/StateMachine.xml</StateMachine>

<StateMachine serviceid="4001234567">../etc/Agent\_demo.xml</StateMachine>

</SERVER>

</CONFIG>

## 动态加载

【此部分未实现】

<IP>和<Port>部分是程序一经启动后就不能改变，如果需要重新改变比部分配置，必须在修改完配置后重启程序，<StateMachine>配置部分应该支持可以随时修改和重新加载，加载后立即生效，在加载过程中不能重启程序和影响程序运行，在程序加载配置文件过程中程序还继续使用原配置，加载完成后产生的新业务使用新配置。

# 脚本语法要求

## 介绍

SCM程序的启动和运行主要是依赖配置文件和状态机脚本文件。配置文件的规则比较简单，只需要遵循xml语法和第7章[配置文件](#_配置文件)介绍的元素定义。

SCM依赖的状态机脚本文件的语法要求比较复杂，状态机脚本文件是由xml和JavaScript两部分组成，所以脚本的编写需要符合这两种语言的语法。文件的主题内容是用xml语言编写，JavaScript是嵌入到xml的内容中。

## 静态语法

静态语法是脚本文件编写后的内容，必须符合的一些定义的规则和内容，包括关键字、命名规则，是内容还没有执行就必须符合的语法。

### 语法规则

1、整个文件必须是合法的xml文档。

2、文档中的元素必须符合在《3脚本元素》、《4 JS变量》中定义的元素。

3、文档中编写JavaScript的内容必须同时符合xml语法和JavaScript语法。

4、所有<transition>元素的target属性所制定的目标必须在文档中。

### 语法检测

xml语法检测可以根据状态机的设计编写XML DTD文档，根据DTD检测文件内容是否符合基本的静态语法。

JavaScript的语法检测可以使用SpiderMonkey中的只编译脚本但不执行脚本的函数。

## 动态语法

动态语法是在程序执行过程中脚本内容是否能按照自己所期望的运行，是否会产生异常。

1. 书写错误，代码编写书写错误，但能执行正确，产生的是非预期结果。例如将1==1，书写成1=1，运算结果为非boolean值
2. 理解错误，编写的代码执行正确，自我检查也认为内容正确，但是没有产生期望的结果。例如认为1+1的结果为3，但实际的执行结果为2。
3. 异常错误，状态机脚本在执行过程中产生了异常，例如：打开不存在的文件，死循环等。