# AITALK 4.0 语音识别产品语法开发指南

本文档将说明 Aitalk4.0 的语法规范的定义，并向用户展示如何利用该规范写出优秀的识别语法

## 基本概念

语音识别的语法定义了语音识别所支持的命令词的集合，Aitalk 4.0 产品利用巴科斯范式（ BNF ）描述语音识别的语法。语法文档被编译成识别网络后，将被送往语音识别器。语音识别器提取输入语音的特征信息并在识别网络上进行路径匹配，最终识别出用户说话的内容。因此语法是语音识别系统的输入之一，它是现阶段语音识别得以应用的必要条件。

例如，开发一个简单的语音拨号应用，可以定义一个如下的语法：

```
#BNF+IAT 1.0;
!grammar call;
!slot <name>;
!start <commands>;

<commands>:(找一下|打电话给) <name>;


<name>: 张三|李四;
```

该语法使识别引擎可以支持以下说法：

找一下张三

打电话给张三

找一下李四

打电话给李四

凡是用户说出这个范围中的任意一句话，均可以被识别系统识别。如果用户说的话不在上述范围之类，识别系统可能拒绝识别。可见语法使用一种结构描述了用户可能说出的语言范围和构成模式。

## 定义

- 记号，对应英文为 **Token**，其描述了用户语音所对应的文本内容，类似于“说法 “，如“中国|美国”表示支持中国与美国两个记号的并列。
- 语义，对应英文为 **Sementic**，表示用户说法所对应的用户所关心的内容，在应用开发中，可以将部分语义内置于语法文本中，以方便应用程序的处理。如在语音拨号的场景中，记号对应为“火警” ，而语义为“**119**”。通过在语法中定义“火警”的语义，当用户说法为“火警”时，识别引擎将“**119**”返回给应用程序，而无须在应用程序内部再进行文本的解释工作，大大方便了应用程序的开发。
- 规则，规则定义了一系列记号及其相互关系的集合，且可以包含其它子规则。通过指定规则的唯一的名称，使其它的规则可以通过名称引用该规则。
- 槽，槽是一种特殊的规则。槽描述了一系列记号的并列关系，且不包含任何子规则。利用 **AitaIk SDK**，用户可以在程序运行过程中实时增减槽中的记号。借用此功能，用户通过定义槽，描述语法中的频繁变化的内容，如通讯录中的人名，而无须更改语法。

# 语法规范

## 简介

Aitalk 语法文档包含两个部分，文档首部（Header）和文档主体（Body）。

文档首部定义了文档的各种属性，而文档的主体则具体定义了用户说话的内容和模式，其由若干个规则组成。

Aitalk 支持块注释及行注释，表示对语法内容的注释，语法编译器将忽略注释。如：

```
!grammar call;    /*定义语法名称，块注释*/

!slot <name>;    //声明槽，行注释
```

文档首部必须出现在文档的开头部分，也就是说一旦出现了第一个规则定义，即宣告文档首部的结束，出现在文档主体中的文档首部声明，作为文档主体（规则扩展）看待，造成语法编译失败。

Aitalk 语法文档以半角分号";"作为每个定义的分隔符。每个分号表示某个定义结束。

```
#BNF+IAT 1.0;

!grammar call;

!slot <name>;

<commands>:(找一下|打电话给) <name>;

!start <commands>;    /*错误！一旦定义了规则，则表示首部结束，再定义首部出错*/

<name>: 张三|李四;
```

## 命名规则

Aitalk 语法规范定义的所有的操作符及关键词均为半角字符，而不支持全角字符。

Aitalk 可以使用关键词及操作符定义诸如规划名、语法名等变量名，Aialk 仅支持英文字符及数字的组合，对于其它字符将会导致编译错误。

```
!grammar call;          /*纯字母，合法*/

!slot <name123>;        /*字母与数字的混用，合法*/

<344>:1|2|3;            /*纯数字，合法*/

<rule_>:1|2|3;          /*非法字符_，编译出错*/
```

Aitalk 定义的最大变量名长度为 15 个英文字符，超过该长度将会导致编译错误。

## 规范总表

Aitalk 语法规范使用一系列操作符及内置关键词描述语法内容。

语法支持的操作符详见下表：

| 操作符 | 描述 | 示例 |
| --- | --- | --- |
| ! | 标识内置关键词的开始 | !grammar |
| <> | 定义规则名称 | <name> |
| ; | 结束符 | 表示一行结束 |
| \| | 或，定义并列结构 | 张三\|李四 |
| [] | 可选，表示内容可说可不说 | <call>: [找] <name> |
| : | 定义规则 | <name>:张三\|李四 |

| | | |
|---|---|---|
| () | 封装操作，定义隐式规则 | <call>:找(张三\|李四) |
| /* */ | 块注释 | /*注释*/ |
| // | 行注释 | //注释 |

支持的内置关键词见下表：

| 关键词 | 描述 | 示例 |
|---|---|---|
| grammar | 定义语法名称 | !grammar dial |
| slot | 声明槽 | !slot <name> |
| start | 定义开始规则 | !star <call> |
| id | 定义说法所对应语义返回值，id 仅适用于记号，而对规则不适用 | <call>:打火警!id(110) |
| void | 保留关键字 | |
| garbage | 保留关键字 | |
| Null | 保留关键字 | |

## 文档首部

文档首部定义了文档的各种属性，包含以下几个组成部分：

- 文档自标识头
- 语法名称
- 槽声明
- 初始规则

## 文档自标识头

BNF 文档自标识头（Self-Identifying Header，强制）

BNF 文档必须在其第一行包含一个形式如下的自标识头，其中的留空必须为一个半角空格字符。

```
#BNF+IAT VersionNumber CharEncoding;
```

文档自标识头定义了文档的版本和编码格式。其中：

- VersionNumber 指定语法文档版本号，目前版本号必须是 1.0。
- CharEncoding 指定语法文档字符编码类型，其默认值是 utf-16。

开发语法时，请确保文档自标识头声明的字符编码类型和文件的真实字符编码类型是统一的。例如，如果声明的文档编码格式是 UTF-8 的编码格式，文档的内容必须采用这一格式。因为语法编译系统依靠这一声明的字符编码，把文档转换成统一的 Unicode 格式。如果声明的字符编码和文件真实的字符编码不一致，那么转换会出错。因而用户需要保证声明的字符编码与文本真实编码一致。

Aitalk4.0 可以支持 GB2312、GBK、UTF-8、UTF-16LE、UTF-16BE 等多种格式的文本编码方式。

推荐中国大陆用户使用 GB2312 编码类型，并在语法中预先声明它。

```
#BNF+IAT 1.0 GB2312;
```

语法名称

BNF 文档需要在文档首部定义语法的名称，借助于语法名称，应用程序可以在特定的业务节点使用特定的语法。用户使用 Aitalk SDK 提供的识别接口，可以指定每次调用使用的语法名称。

语法名称的定义形式如下

```
!grammar GrammarName;
```

槽声明

槽为支持动态修改语法内容的槽。Aitalk 支持语法编译完成后动态修改语法的内容，以动态支持新的说法。用户通过在语法首部声明槽规则，指定用户可动态更改的规则。形式如下：

```
!slot SlotName;
```

用户可以在文档主体定义规则，也可以通过开发接口中定义规则，即用户可以在语法中声明某个槽而无须在文档文体中定义，而是程序运行过程中通过 Aitalk SDK 进行动态添加与删除槽中所含的条目。Aitalk 要求定义为槽的规则必须满足为一系列的词表的并列结构，否则会导致编译失败。

```
!slot name;                          /*声明槽*/

!start <call>;

<call>: <prefix> <name>;

<prefix>: 找[一]下;      /*非简单的并列结构，不能定义为槽*/

<name>:张三|李四;     /*可以定义为槽*/
```

初始规则

语法文档内容定义为的一系列的规则，用户必须在文档首部定义语法的初始规则。形式如下

```
!start <call>;
```

## 文档主体

语音识别语法是通过规则定义（Rule Definition）和规则引用（Rule Reference）来组成语法主体（Body）的。规则引用的各种组合通称为规则扩展（Rule Expansion）。

## 规则定义

一个规则定义用赋值操作符“:”把规则名称和规则内容联系起来，规则名称左右需要被包含在“<>”中，而规则内容就是所谓的规则扩展。规则扩展的最基本的结构是顺序、选择和循环。

规则的定义形式如下：

```
<ruleName>:ruleExpansion;
```

其中<ruleName>表示规则名称，而 ruleExpansion 表示规则内容，其是一系列的规则及词的组合，均不能为空。

每个规则都必须保证其定义的唯一性，也就是规则名在同一个语法中不能够被定义两次。在定义规则时，要注意规则名称不能是 VOID、NULL 和 GARBAGE，这三个规则名称被作为保留关键词为后续功能扩展使用。

## 记号

记号定义了单词或者其它可以被说出的实体。任何记号都是合法规则扩展。因此识别系统最终识别出的是记号。如下所示，规则 rule 定义为三个可选的记号:中国、美国、德国。

```
<rule> = 中国|美国|德国;
```

## 语义

可以通过关键词!id 定义了记号所对应的语义。语义只能支持数字 ,其它数字将会导致编译错误。

语义支持的数字范围为 32 位有符号整形，也就是-2^(32-1) ~ 2^(32-1) -1（其中^表示幂操作），

当定义的数字操作此范围时，会导致编译错误或者被截断。

注意，语义不能支持规则的语义。

```
<rule> = 中国!id(0)|美国!id(1)|德国!id(2);
```

## 槽定义

在文档首部声明槽时，可以在文档主体中进行槽的定义，也可以不进行定义，而是在识别过程中

通过 SDK 动态定义。

槽必须定义为一系列记号的并列组合。如下所示，

```
<rule> = 中国|美国|德国;
```

也可以为其中的记号标识语义。

```
<rule> = 中国!id(0)|美国!id(1)|德国!id(2);
```

注意，如果记号中出现空格、制表符等分隔符，需要用成对的双引号进行标识。如

```
<rule> = 中国!id(0)|美国!id(1)|德国!id(2)|"the United States "!id(1);
```

## 规则引用

规则引用是指引用本文档中所定义的规则。如下所示，在规则<main>中引用了<name>的规则。

```
<name>:张三|李四;

<main>:打一下<name>;
```

Aitalk 系统可以支持引用未定义的槽。未被定义为槽的规则未定义时，语法编译器将会返回编译错误的信息。

规则引用是上下文无关的，即用户可以预先引用规则，而在后面定义规则，如下所示，<main>规则预先引用<name>规则，而在<main>规则后再定义<name>规则：

```
<main>:打一下<name>;

<name>:张三|李四;
```

## 序列和封装

一个规则定义可以连续引用多个规则名，记号名以及它们的各种组合。序列相当于程序设计中的顺序结构。如

```
This is a test                  //记号的序列

<action> <object>               //规则引用的序列

The <object> is <color>         //记号和规则引用的序列
```

将规则名、记号及其组合以左右小括号括起可以使之作为一个整体参与规则的定义。如，以下两条规则的等同的。

```
<rule>:我找张三|我找李四;

<rule>:我找（张三|李四）;
```

选择结构

在规则扩展中选择结构表示说话时只可能覆盖其中的一条路径，它相当于程序设计语言中的选择结构。选择结构以”|”来分隔每个可能的分支。如

```
<rule>:张三|李四|王五;
```

优先级

不同的操作符之间具有不同的优先级。以下为不同操作符由高到低的排序。

- 规则引用、加引号的记号
- 圆括号(“(“，”)”)及方括号(“[“,”]”)
- 规则扩展序列
- |，选择操作符