科大讯飞语义开放平台

——abnf 文法规范<Version 3.0>

目录

第1章 规范概述

《搜索句文法规范》基于万维网联盟（World Wide Web Consortium 简称 W3C）的语音识别语法规范 1.0 标准（简称 SRGS1.0）进行了删减、修改和扩展，采用 ABNF 格式；配套的有从文法档到解析网络的编译工具。标准的 ABNF（Augmented BNF syntax）格式的 SRGS1.0 语法文档不一定能在编译工具上正确运行。

正确编写的文法档经编译工具处理可生成匹配网络，网络作为匹配引擎的输入，可由匹配引擎对用户输入进行匹配。因此，文法档的编写是句文法匹配的基础，本文档提供给用户的搜索句文法的开发指南，用户阅读本指南可以迅速的开发出自己需要的文法；文档的最后给出了支持本规范的编译工具的使用说明。

相关参考信息：

1. SRGS1.0 语法规范：http://www.w3.org/TR/speech-grammar/
2. ABNF 格式文档：http://www.ietf.org/rfc/rfc2234.txt

1.1. 基本概念

文法的目的：文法等同于正则表达式，它定义了一个句子集合。解码器将根据文法生成的集合，对输入的句子进行一个搜索句文法主要包含两部分：文档头部和文档正文，后续章节将对它们详细阐述，下面给出一个简单的文法例子：开发一个文法用来匹配话费查询类的用户输入，可有如下定义：

```
#ABNF 1.0 UTF-8;

root task_final;

#ABNF HEAD-END;

$want = 要 | 想;

$查询 = 查 | 查询;
```

```
$费用 = 手机费 | 话费;

$task_final = [我] [$want] $查询 $费用;
```

以上实现了一个简单的话费查询文法，涵盖了如下可能的说法：

查手机费

查话费

查询手机费

查询话费

我查手机费

我查话费

我查询手机费

我查询话费

要查手机费

要查话费

要查询手机费

要查询话费

想查手机费

想查话费

想查询手机费

想查询话费

我要查手机费

我要查话费

我要查询手机费

我要查询话费

```
我想查手机费

我想查话费

我想查询手机费

我想查询话费
```

凡是以上范围中的任意一个，均可以被系统匹配。如果用户输入不在上述范围之内，会被“拒识”。

1.2. 名词解释

- 空白符：空格、回车、换行、横/竖向制表符。

1.3. 文档注释

文发文档中可以使用注释，形式如下（同 C++编程语言）：

```
// C++/Java-style single-line comment

/* C/C++/Java-style comment */
```

第2章 文法档头部

2.1. 简介

文档头部定义了文档的各种属性，主要包括：

- ABNF 文档自标识头（Self-Identifying Header）
- 模式（Mode）type
- 元数据（Meta）
- 引用文件声明部分
- 类型说明 type
- 文档头部结束标记

文档头部必须出现在文档的开头部分,文档头部的每一语句必须以英文半角分号 ； 标记语句结束。从现在开始，如果不特别说明，所有的符号均为英文半角。

一个典型的头部如下：

```
#ABNF 1.0 UTF-8;
mode SMS;
meta;
#include "..\..\..\Common\common.abnf";
#include "..\..\..\BasicPhrase\BP_declare.abnf";
mount "video_name";
root main;
#ABNF HEAD-END;
```

2.2. ABNF 文档自标识头

ABNF 文档自标识头（Self-Identifying Header），在所有语句之前强制出现，定义了文档的版本和编码格式，形式如下：

```
#ABNF VersionNumber CharEncoding;
```

其中：

- VersionNumber 指定文法档版本号，与相应的编译工具对应。
- CharEncoding 指定文法档字符编码类型，目前只支持 UTF-8,如果文法是 gbk 编码将不会编译。

目前必须书写为：#ABNF 1.0 UTF-8; 不能有任何不同。

2.3. 引用文件声明

3.0 扩充了引用的功能，即可以引用其他文法，也可以引用词典源文件，也可以引用声明文件。必须使用#include，格式和 C 语义一致，并用分号结尾，否则会出错。

同时提供了运行时“加载关键字”，此关键词和解析器相匹配。起资源检查的作用。

```
#ABNF 1.0 UTF-8;
....
#include "..\..\..\Common\common.abnf";
#include "..\..\..\lst\singer_name.lst";
#mount "video_name";


#ABNF HEAD-END;
```

命令名	功能	备注
#mount	#mount “词典的逻辑名称”; 声明解码时需要的词典资源。解码时会检查输入，查找有没有符合该逻辑名的词典资源存在。 编译时并不要求该资源存在。一般用于同一个文法，加载不同的定制资源	在资源头部会有额外记录 3.0 新增
#include	#include “引用的文法或者词典的相对路径” 编译时会检查该路径上的文件，必须存在	引用的资源会和文法网络合成一个文件，比如引用词典，词典不会单独生成一个

		文件，而包含于资源内部

2.4. 类型

type 模式名 一般不需要额外指定

2.5. Root 规则

用法	功能	备注
root 规则名;	将 root 指定的规则，编译成二进制文法网络并输出。规则名不要加$，只名字即可 此功能在检查子文法是否正确时很便利，只需指定 root 为该子文法规则即可 规则名不需要加$前缀	此为 3.0 新增句式

2.6. 文档头部结束标记

文档头部的最后，必须有结束标记语句（指明文档头部的结束）：

```
#ABNF HEAD-END;
```

如果是 lst 文件，则为

```
#DICT HEAD-END;
```

2.7. 词典资源文件格式

词典 lst 文件格式简单：

头部是：

```
#DICT 1.0 UTF-8;

#DICT HEAD-END;
```

体：格式与前半本相同。另外，文件最后一行必须加上：

```
#IFLY_NLI_RES_END;
```

否则会报告文件不完整。

词典规则名只允许字母数字和下划线。

第3章 文法档正文

3.1. 综述

文法文件,实际上就是一些列规则定义的组合。文法档正文由一个或多个“赋值”表达式组成。基本格式如下：

```
$vName = vDefinition;
```

- 表达式必须包含三部分：变量声明，等号，变量定义。这三个部分合起来，叫做“规则”，在本文档中不严格的叫做“变量”。
- 变量声明以 $ 或者@（辅助符）开始，后续紧跟变量名称(*vName*)，变量名称可以是英文字母、数字、下划线的任意组合(反对中文），见到空白符视为变量名结束。
- 变量定义(vDefinition)包含四种基本元素：终结词、其他变量、运算符、辅助符；变量定义是这四种元素在符合文法要求情况下的各种组合。

3.2. 终结词

在文法中，终结词不能被继续扩展。它定义了用户输入可能表达的实体，句文法匹配最终的目的就是识别出这些终结词。例如在下面的文法中：

```
$fee = 手机费 | 话费;
```

“手机费”、“话费”就是终结词。而$fee 就不是。

目前定义的终结词包括以下这些：

终结词类型	示例
英文(半角)	a abc
数字(半角)	1

	1234
汉字	好 中国
特殊字符	4.1 节
以上各种组合	

3.3. 变量分类

一般只需要使用$,其后加规则名来定义一个规则。规则名可以包含中文，但不推荐。

3.3.1. 显式和隐式变量的效果

以下截图取自 2.0 引擎执行 weather 业务的结果

$w2_2{biz:weather} = [$query{opera:query}] $date{param:date} [$interval] [$time] [$SW_De] [$place] [$here] [$SW_De] [$interval] $r1 [情况|状况];

其生成的调试信息为：

```
<main Score="-0.101563">
        明天天气
        <weather Score="-0.101563">
                明天天气
                <w2 Score="-0.101563">
                        明天天气
                        <w2_2 SemanticInfo="biz:weather" Score="-0.101563">
                                明天天气
                                <ElementsList>
                                        <date SemanticInfo="param:date" Score="-0.101563">
                                                明天
                                        </date>
                                        <r1 Score="0.000000">
                                                天气
                                        </r1>
                                </ElementsList>
                        </w2_2>
                </w2>
        </weather>
</main>
```

可以清楚的看到 w2 引用了 w2_2。

如果定义 w2_2 为隐式变量，

@w2_2{biz:weather} = [$query{opera:query}] $date{param:date} [$interval] [$time] [$SW_De] [$place] [$here] [$SW_De] [$interval] $r1 [情况|状况];

调试信息为：

```
<main Score="-0.101563">
        明天天气
        <weather Score="-0.101563">
                明天天气
                <w2 Score="-0.101563">
                        明天天气
                </w2>
        </weather>
</main>
```

用@定义的变量不会输出到调试信息中，也就没有精细结构。

3.3.2. 变量引用

变量引用有三类

1. 本地引用：引用本文法中所定义的变量。
2. 外部引用：引用其他文档中定义的变量。
3. 预定义变量引用：引用系统预定义的特殊变量

这三类引用的格式已经统一到本地引用的格式。

3.3.2.1. 本地引用

对于本地引用，仍以前面的话费查询的文法为例，如下

```
$want = 要 | 想;

$查询 = 查 | 查询;

$费用 = 手机费 | 话费;

$task_final = [我] [$want] $查询 $费用;
```

其中最后一行变量$task_final 的定义部分的$want、$查询、$费用三个变量即为本地引用，它们都是在本文档内定义的。

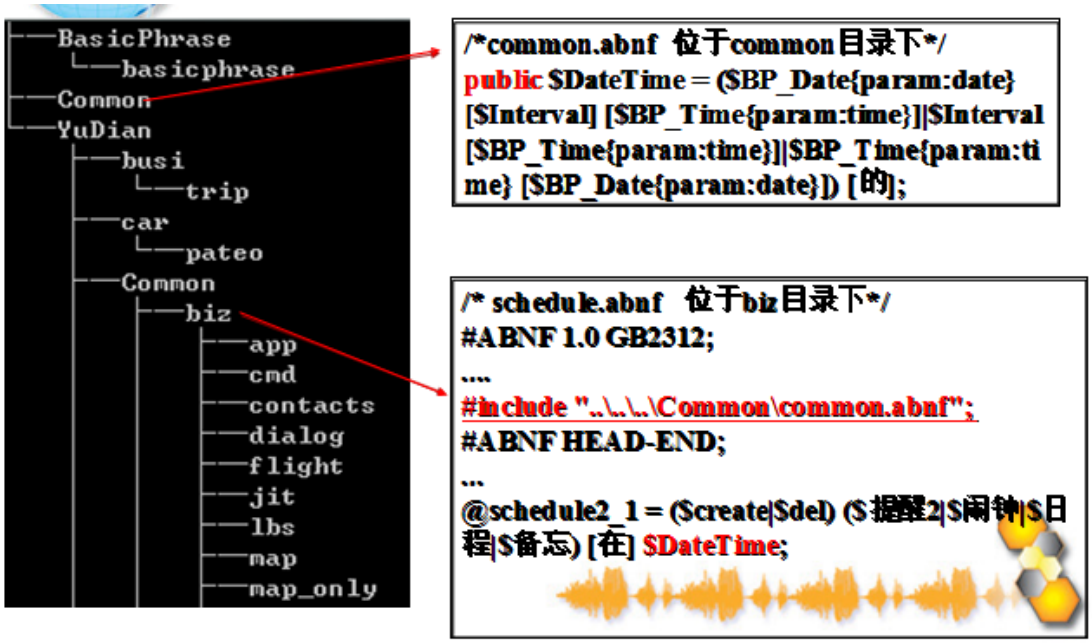
3.3.2.2. 外部引用

外部引用的格式和本地引用一样。和 C 编程语言保持一致。但是需要两点：

1. 被引用的规则必须用 public 修饰，声明为公有变量；
2. 引用者的头部中使用#include "XXX"; 包含需要引用的文法文件。然后就可以像引用本地变量一样引用外部的变量。

include 中使用相对路径，且一定要以分号结尾，并写在文法头部。

举例：schedule.abnf 引用 common.abnf 中的$DateTime：



另外需要特别注意的是，外部引用时，不能出现循环引用，即 A 引用 B，B 不能再引用 A。

3.3.2.3. 转义字符变量和通配符

所有的特殊变量均已 $_ti_ （英文半角）开头，以下划线 _（英文半角）结尾，注意，此形式的变量都作为系统预留，用户定义的变量不能采用这样的形式。目前主要有两大类特殊变量。

第一，在文法中用到的特殊意义的符号，要想出现在文法中作为匹配实体（终结词），需要给它们特殊的格式，此时需使用特殊变量。

以下本版本不支持 但可以进行转义	
$_ti_blank_	空格或制表符
$_ti_space_	空格
$_ti_tab_	制表符
$_ti_backslash_	\
$_ti_slash_	/
$_ti_dollar_	$
$_ti_semicolon_	;
$_ti_or_	|
$_ti_eq_	=
$_ti_la_ (left angle)	<
$_ti_ra_ (right angle)	>
$_ti_lb_ (left brace)	{
$_ti_rb_ (right brace)	}
$_ti_ls_ (left square)	[
$_ti_rs_ (right square)	]
$_ti_lr_ (left round)	(
$_ti_rr_ (right round)	)
$_ti_qt_	"

第二，系统预定义了一些有特殊含义的变量：

$_ti_ch_	通配符： 任意一个字符（字母、数字、汉字）； 支持自定义权重，提取需要提取的内容。
$_ti_filler_	通配符： 0 到任意多个字符； 不支持自定义权重，纯粹用于过滤。
$_ti_main_	现已废弃。
$_ti_garbage_	保留，暂未定义。

3.3.2.4. 声明变量

如果要应用的规则，所在的文法不可见，但又要引用它，则可以使用动态变量关键字来声明，小型文法一般不需要。

3.4. 运算符

变量定义部分从逻辑上看，是几种不同的逻辑结构，这些结构由不同的运算符共同形成。主要涉及的结构和运算符为：串连（空白符）、选择（分割符）、可选（中括号）、分组（小括号）、重复（尖括号）。

3.4.1. 规则声明引用符——$' 和 '@'

单目操作，用于规则定义中的声明或规则的引用。

如“$x = $y 一下”，第一个’$’表示声明，声明 x 规则，第二个表示引用，引用 y 规则。

引用时只用$，定义时两者皆可。区别前述已经介绍。

3.4.2. 定义符—— '='

双目操作，如上规则中的‘=’，在一个规则定义中必须且只能出现一次。

3.4.3. 注释符—— ‘//’或‘/*….*/’

前者行注释，后者块注释。和 C 语言的使用方式一致，但是//行注释在末尾时不能用反斜杠续行。

```
$rule =   a /* 这里是注释 */b;
```

相当于

```
$rule = a b;
```

3.4.4. 语义符——‘{’‘}’

必须左右配对使用，不可嵌套

内容包含两种操作符‘：’和‘%’分别表示语义输出符号分级及幅值

左右配对操作符之间可嵌套，但不可交错

3.4.5. 串连—— 空格符、TAB

双目操作，左右结合顺序无关紧要， 在后接符号为其他操作符时可以不写，如ab

串连相当于程序设计中的顺序结构。由终结词、变量以空白符相隔、顺序排列，其形式如下：

```
我 查 话费                  // 终结词的串连

$查询 $费用                 // 变量引用的串连

我 $want $查询 $费用        // 终结词和变量引用的串连
```

3.4.6. 选择—— ‘|’

选择表示匹配用户输入时，经过几条并行路径中的一条路径。不同的选择分支使用英文半角 | 来分隔，形式如下：

```
查 | 查询                   // 终结词的选择

$查询 | $取消               // 变量引用的选择

开通 | $查询 | $取消        // 终结词和变量引用的选择
```

3.4.7. 可选——中括号 ‘[’‘]’

可选表示匹配用户输入时，可以经过一条路径也可以不经过此路径。可选部分由英文半角中括号 [] 来标记，[ ]之间的部分为可选，形式如下：

```
[我]                  // 终结词的可选

[$want]               // 变量的可选
```

3.4.8. 分组——小括号 ‘(’‘)’

通过英文半角小括号 () 可以将若干个文法单元封装界定为一个结构， () 以内的部分为一个整体单元，可避免文法在阅读上产生歧义。

必须左右配对使用，可嵌套，通常用于改变优先级可以改变文法单元的结合优先级。例如：

```
$order = 我 想 | 要 查 话费;
```

这样的结构阅读起来让人费解，因为优先级不如串联符号，这一句等价于：

```
$order = (我 想 ) | (要 查 话费);
```

而文法开发者的意图并非如此，此时可以加上括号来改变结合优先级：

```
$order = 我 (想 | 要) 查 话费;
```

又比如，想要给多个文法单元加上一个语义的时候，会起到很好的作用。后文会介绍语义的详细内容。

3.4.9. 重复——尖括号‘<’‘>’

注意，3.0 的重复结构和语义的使用方式完全相同。重复结构用来在文法中表示需要重复说出的内容，它特别适合表示诸如数字串等一些终结词反复出现的结构。重复结构的基本格式有如下四种：

```
$digit <n>          // 变量重复出现 n 次 (n>0)

$digit <n+>         // 变量重复出现 n 次以上 (n>=0)

$digit <m->         // 变量重复出现 m 次以下 (m>0)

                    // 等价于 <$digit, 0-m>

$digit <n-m>       // 变量重复出现 n 至 m 次(n>=0，m>n)
```

可以看到，重复结构处于英文半角 <> 以内，其中有一英文半角“逗号”，作为分隔辅助符，“逗号”之前的部分为要重复的内容，“逗号”之后的部分给定了重复的次数信息。

以上$digit 变量也可以是终结词或者终结词、变量、运算符等组成的复杂语句。

如果要匹配重复 0 次，比如 0 到 3 个数字建议写成［$digit〈1-3〉］而不是 $digit〈3-〉，这样能够很容易的发现空弧穿透的错误。

3.4.10. 语义——‘{’‘}’

语义符必须左右配对使用，不可嵌套，具体用法见后文。

3.4.11. 权重——反引号‘`’‘`’

不可以嵌套。其使用方式和语义相同。

用法：反引号包上数字：` 数值`。默认所有的文法单元都是 0 分。不可以嵌套使用。详细内容请见后文。

3.4.12. 文法单元

文法单元是指文法中相对独立的文法部分，某种程度上可以当作一个整体来看待：

如果文法中只有普通字符，则称为独立文法单元。普通字符，就是中英文字母字符文本。

如果包含任何非普通字符，则称为复合文法单元。非普通字符，就是上文提到的各种运算符。

对于终结词，各符号直接相连的终结词是一个文法单元，如“手机话费”，若中间有空白符隔开，如

```
手机　话费
```

则表示两个独立文法单元：手机、话费。

由运算符（()、<>、[]）构成的文法单元、由辅助符（{}、\\）构成的文法单元以及变量彼此直接相连，或它们与终结词直接相连，都仍然是不同的文法单元，比如：

```
[我]想$查询(话费 | 手机费)
```

包含四个文法单元：[我]、想、$查询、(话费 | 手机费)。

另外，一个文法单元内部可以嵌套 “更小”的文法单元，如以下文法单元

```
([手机]话费)
```

由两个“小”文法单元：[手机]、话费，串联而成。

文法单元的概念与语义信息、属性、权重信息的定义直接相关，后面将详细介绍。

3.4.13. 语义信息

语义，可以理解成一种集合。概括表明用户输入的真实意图，例如“话费”、“电话费”、“电话费用”、“手机费”等不同的讲法都是一种费用。而“我想”“我要”“我想要”都表示“需要做某事”的意向。后续的业务处理程序需要这些信息。

- 语义信息放在一对英文半角大括号｛｝（语义辅助符）之间。
- 语义信息给定语义的对象是：语义左边与之最邻近的文法单元。
- 一个文法单元可以被两个或两个以上语义信息“修饰”。

例如：

匹配成功字串：“我想查合肥明天的天气”，其中包含 3 个语义：地点“合肥”，日期“明天”，操作“查天气”

用 json 结构来表示就是：

```
{
    "biz":"weather",
    "opera":"query",
    "param:city":"合肥",
    "param:date":"明天"
}
```

而 biz:weather, opera:query 等都是由《语义处理协议（互联网电视）v1.1.pdf》中定义，文法中只需要将这些语义信息的添加涉及到“语义信息档”，其内容格式如下：

```
1    业务
2    操作
3    资费
4    旧密码
5    新密码
```

每一行指定一个语义，分为三个部分：第一部分为唯一的语义 ID，要求是**英文半角**阿拉伯数字串；第二部分为分隔符，分隔符包括空格，Tab 和逗号（分隔符同样都是**英文半角**）；第三部分为语义定义，定义内部不能有第二部分中的分隔符。

文法档中进行语义信息的添加时，目前暂只支持将一个语义定义（不是语义 ID）放置于大括号｛｝中，并且此处添加的语义定义必须包含于上述“语义信息档”的定义中。举例如下：

```
$order = 我 (想 | 要) 查 {操作} 话费 {资费};
```

注意，语义添加时，语义须与“语义信息档”的语义定义部分保持一致，语义与两个大括号｛ ｝之间可以有空白符。另外，语义定义在字符的使用上有一定的限制，具体可见 4.2 节中的描述。

禁止写法：对于可选结构，如果要附加语义，必须要将语义放在可选结构的中括号内，否则编译器将会报错。因为文法的目的是：只有可选单元中的文法单元匹配成功，语义才会输出。

```
$ruleName = [ 文法单元 ] { 语义内容 };   错误
$ruleName = [ 文法单元 { 语义内容 } ];   正确
```

3.4.14. 权重（分值）详细解释

权重直接修饰左边最靠近的文法单元，给其添加上一个得分。该得分可以是小数或整数，支持正负号，但不支持指数形式！

举例：

```
$rule `+1.0`;

(你好|大家好)`-0.4`;
```

权重和语义的用法和注意事项一致禁止 ［ XXX ］`权值` 的写法 ，需写成 [XXX `权值`]！

3.4.14.1. 通配符默认权重：

系统会给如下两个变量隐式的增加分值，

$_ti_filler_ -1 分（此分值由编译器决定，不支持自定义）

$_ti_ch_ -0.8 分 （此分值由编译器决定。）

文法中，所有的单元，初始权重都是 0，但系统会在通配符的后面增加`-1` 和 `-0.8` ，这是隐藏于编译器内部的实现。

3.4.14.2. 通配符自定义权重

如果用户给$_ti_ch_附加权重，则使用用户所写的权重。

$_ti_ch_ 自定义权重，但注意用法：

```
$_ti_ch_ `-0.4` <2-5>

每个被通配符匹配的字都会扣-1.2 分

$_ti_ch_ ,<2-5> `-0.4`

只要是满足 2-5 个字的字符串。$_ti_ch_ 后面没有用户自定义权重，使用系统内置权重，除了每个字扣 0.8 分，总共再扣 0.4 分
```

3.4.14.3. 使用举例：

之所以支持通配符权重，是为了某些有明确的上下文的场景。比如：“我想听××这首歌”。此时，XX 可以确定是歌名，可以减少通配符的扣分，有可能让此条规则胜出。

另外有些场景，是因为通配符权重的存在要特别小心：

如 map 中，$adr 允许全通配符。下面 rule1 和 rule2 都可以走通 “XX 到 XXX 怎么走” 这句话。

```
$rule1 =  $adr $how $go    /* XX 到 XXX 可以匹配$adr */
$rule2 =  $adr (到|去) $adr $how $go
```

若通配符的权重设置为 0，两规则得分相同，输出结果未定义。

此时，文法中就应提高“到|去”的权重。让 rule2 胜出，或让通配符扣分

再看权重对文法的匹配结果的影响

比如 （改写自 BasicPhrase.abnf）

```
$rule =ab {param:ab}| a{param:a} | b {param:b};
$output = < $rule ,1-2 > ;
```

对于输入串“ab”有两种匹配方式，一种是直接匹配 ab，循环一次。另一种是先匹配 a，再匹配 b，循环两次。 这两种都合法，故给出的结果不确定。

有可能是{param:ab}，也可以是{param:a}{param:b}

```
$rule =ab {param:ab}`0.1`| a{param:a} | b {param:b};
$output = < $rule ,1-2 > ;
```

则 {param:ab}会胜出

3.4.15. 运算符、辅助符的优先级

符号优先级（由高至低）

注释 > 串 > 语义符 > 声明引用 > 配对操作符及循环 > 连接 > 可选 > 定义符

运算符	含义
[](){}	括号会改变优先级

{ } < > ``	一元运算符
tab space 串联	二元运算符 串联
|	二元运算符 并联
=	二元运算符 赋值

第4章 文法规范的其他说明

4.1.1. 不能写的结构

- 重复结构的嵌套要谨慎，否则会引起效率的降低。通常情况下会导致文法上正确，但实际意义很不明确 例如：

```
$ruleName = [<请, 3-5>] <2-4>;
```

或者

```
$ruleName = 我[(([想][要])<1->)]查话费;
```

- 严禁空弧穿透结构：

```
$ruleName = [我想要] [打电话];
```

这个规则可以什么实际内容也不匹配，却仍然可以通过编译生成的网络。这会引起引擎异常。

同理，可以允许零次的重复结构也会有问题:一下文法，可以什么文字也不匹配，却抛出语义内容。

```
$ruleName = 内容 < 0-3> {语义};
```

需要写成

```
$ruleName = [ <内容 , 1-3> {语义} ]
```

- 严禁连续多个通配符直接相连：

```
$ruleName = $_ti_filler_ $_ti_filler_
```

假设输入的句子长为 n，文法中连续串联了 k 个通配，则增加的复杂度是：O(n 的 k 次方)

4.1.2. 语义中的字符

语义定义的大括号｛｝内可以是如下字符：大部分 ASCII 字符(但不包括 4.1 节表格第一列中所列字符)、全角符号、中文；如果语义定义中出现第三章介绍过的节表格第一列中所列字符，会导致文法编译错误。如果一定要使用，请在前面增加转义字符‘\’。

4.1.3. 文法书写的方法

原则：模块化

方法：自顶向下和自底向上。

请参考《从零开始写 abnf 文法》的内容。