词法分析器

ZhangXu

2019 年 1 月 23 日

词法分析器产生*tokens*，之后输入*parser*。此章节描述词法分析器如何将文件内容拆分成tokens。

Python以Unicode编号形式读取程序文件，定义编码声明对文件编码，默认的是UTF-8（PEP 3120)否则SyntaxError异常产生。

1 行结构

Python程序被划分为**逻辑行**

1.0.1 逻辑行

逻辑行的结尾以token $NEWLINE$表示。逻辑行可由物理行通过显示或隐式的连接线规则构造。

1.0.2 物理行

物理行是由行尾序列终止的字符序列。Unix使用ASCII LF,Windows使用ASCII CR LF,Macintosh 使用 ASCII CR。**无关平台，所有这些可以平等使用**。输入的结尾充当物理行的隐式终止符。

当嵌入Python时，源代码字符应该使用标准C约定传递给Python API(\n字符表示ASCII LF，行终止符)。

1.1 注释

注释使用#号来开始，物理行结尾。注释并非tokens

1.2 编码声明

若注释再Python文件的第一或第二行中，且符合正则匹配规则

$$coding[=:]\s*([-\w.]+)$$

则解析为编码声明；推荐使用以下格式

$$\#-*-coding:<encoding-name>-*-$$

$$\#vim:fileencoding=<encoding-name>$$

默认的是UTF-8。（若UTF-8字节串以b'\xef\xbb\xbf'开头，微软的笔记本支持此）

1.2.1 显式行连接

通过\(*backslash*)物理行可连接成一个逻辑行

1.2.2 隐式行连接

圆括号，方括号或大括号看看可以在使用反斜杠的情况下分割多个物理行。

1.2.3 空行

仅包含空格、制表符、换页符和可能的注释信息的逻辑行将被忽略（即不产生NEWLINE token）

1.2.4 缩进

逻辑行开头的空格或制表符用于计算缩进级别，确定语句的分组。缩进不能使用斜线分割多个物理行。

同时使用tabs和空格spaces会触发TabError异常。

连续行的缩进级别用来产生INDENT和DEDENT token,其使用堆栈数据结构。如下描述：

读入文件第一行之前，堆栈push一个永远不会被pop的0。push至堆栈的数字从栈底到栈顶严格增加。每个逻辑行的开头，将行的缩进级别与栈定比较。如果相等，保持原状。若缩进级别大于栈定数值，其将被push至栈顶且INDENT token产生。反过来若小于栈顶值，那么此缩进数值必须是堆栈中的一个数字；所有比缩进数值大的数将被pop直至相等，每个被pop的数值会产生一个DEDENT

token。在文件的末尾，为留在堆栈中每个大于0的数值产生一个DEDENT token。

1.2.5 tokens间的空格

空格，tab，换页符均可用于拆分token成为两个token。

2 其他tokens

除了NEWLINE INDENT DEDENT，其他tokens如下：

$$identifiers, keywords, literals, operators, anddelimiters$$

（标识符，关键字，字面值，操作符，分隔符）。空格不是tokens，其可分隔tokens。在存在歧义的地方，token包括从左至右可组成的合法token的最长字符串。

3 标识符和关键字

标识符也被成为**名称**

python3引入了ASCII范围之外的其他字符。对于这些字符，分类使用*unicodedata*模块中包含的Unicode字符数据库的版本。

标识符长度不受限。

所有的标识符在解析时被转换成**正规形式NFKC**；通过**NFKC正规形式**比较标识符。Unicode等价性

3.0.1 关键字

下列标识符被用作**保留字**或**关键字**。

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

3.0.2 保留的类标识符

_* 不能被$from module import *$导入。交互式解释器中使用特殊标识符_来存储上次计算的结果；它存储在builtins模块中。当不处于交互模式时，_没有特殊含义，也没有定义。

__*__ 系统定义名称。由解释器或标准库定义名称。

__* 类私有名称。

4 字面值

字面值指一些内置类型的常量值。

4.0.1 字符串和字节串字面值

字符串文字由一下词法定义描述： 无论字符串或字节串的前缀和剩下的字面值均不允许有空格。默认是UTF-8编码的字符串。

两种类型的文字均可用单引号或双引号括起来。也可以包含在三个单引号或双引号的组中（称其为$triple-quoted strings$）。\用来转义特殊意思的字符。

字节串总是以'b'或'B'为前缀；其产生一个$bytes$类型的实例而不是str类型。它们包含ASCII字符，数字值大于128的字节必须用转义表示。

字符串或字节串都可以选择以字母'r'或'R'作前缀；称之为**raw strings**且视反斜杠为字面字符。（例如，'\U'和'\u'在raw strings中不再特殊对待。）

带有前缀'f'或'F'的字符串称**formatted string literal**。'f'可以和'r'组合，但是不能和'b'或'u'组合，因此raw formatted strings是可行的，但是格式化的字节串则不可行!

在triple-quoted文字中，不必转义换行符和引号。除非'r'或'R'前缀存在，否则转义序列将根据与标准C使用的类似规则解释。如下：

- \newline 反斜杠和换行符被忽略
- \ \ 代表反斜杠
- \' 代表单引号
- \" 双引号
- \a ASCII Bell(BEL)

- \b ASCII Backspace (BS)
- \f ASCII Formfeed (FF)
- \n ASCII Linefeed (LF)
- \r ASCII Carriage Return (CR)
- \t ASCII Horizontal Tab (TAB)
- \v ASCII Vertical Tab (VT)
- \ooo Character with octal value ooo
- \xhh Character with hex value hh

仅在字符串中识别的转义序列为：

- \Nname
- \uxxx 16位十六进制xxxx的字符
- \Uxxx 32位十六进制的字符

4.0.2 字符串连接

使用"或""相邻的字符串会连接成一个整体。此特性定义在词法等级，在**编译时**实现；而"+"号在**运行时**实现。

4.0.3 格式化字符串

'f'或'F'开头的字符串表示格式化字符串。其可能包含可替代字段，由大括号分隔。其他字符串总是具有常量值，而**格式化字符串是在运行时计算的表达式**。花括号外的部分按字面值处理，任何"双大括号"被替换成相应的单花括号。替换部分由''开头，之后是**一个表达式**，表达式后可能有一个'!'号引入的**转换字段(s,r,a)**。之后可能跟一个由':'号引入的格式说明符(format specifier)，最后由''闭合。

其中的表达式被视为由**括号**括起来的Python常规表达式。不允许使用空表达式，且Lambda表达式必须显式括起来。替换表达式看可以包含换行符，但是不能包含注释。表达式从左至右计算。

若指定了转换，格式化之前需计算表达式结果。'!s'使结果调用str(),'!r'调用repr(),'!a'调用ascii()。

结果出来后使用format()协议。格式说明符(例%d,# 0x)传递给表达式或转换结果的__format__()方法。然后得出最终结果。

顶级格式说明符包含嵌套替换区域。嵌套区域也可能包含它们自己的转换字段和格式说明符，但不能包含更深层次的替换字段。The format specifier mini-language与字符串的.format()使用方法相同。

格式化字符串可以相连，但替换区域不可跨文字分割。

格式表达式中不允许使用反斜杠。需要包含反斜杠转义的值，创建一个临时变量。

格式化字符串文字不能用作文档字符串，即使它们仅是字符串不包含表达式。

4.0.4 数字字面值

三种：整数，浮点数和虚数。没有复数的直接定义，但可以使用一个实数和虚数。

注意数字字面值不包括符号，因此-1实际上是一个表达式，由'-'和1字面值组成。

4.0.5 整数字面值

定义上整数没有限制大小。**确定字面值的数值时会忽略下划线**。(仅为分组提高数字可读性)数字之间可出现一个下划线，且在如0x之类的说明符之后。

非零十进制数的前导零不被允许使用。 整数包括十进制整数(100_985_211)，二进制整数(0b 0B 0b_ 0B_001_001)，八进制整数(0-7)如上，十六进制整数(0-9 a-f)

4.0.6 浮点数

举例：3.14 10. .001 1e100 3.14e-10 0e0 3.14_13_33 3.1415e13_31

4.0.7 虚数

形如(3+4j)创造一个实部非零的复数。

5 操作符

+	-	*	**	/	//
%	@	≫	≪	&	—
∧	∼	<	>	≤	≥
==	!=				

6 分隔符

(	)	[	]	{	}	
,	:	.	;	@	=	→
+ =	− =	∗ =	/ =	// =	% =	@ =
% =	| =	∧ =	≫=	≪=	** =	